Package 'DescTools'

May 7, 2026

Type Package

Title Tools for Descriptive Statistics

Version 0.99.60

Date 2025-03-28

Description A collection of miscellaneous basic statistic functions and convenience wrappers for efficiently describing data. The author's intention was to create a toolbox, which facilitates the (notoriously time consuming) first descriptive tasks in data analysis, consisting of calculating descriptive statistics, drawing graphical summaries and reporting the results. The package contains furthermore functions to produce documents using MS Word (or PowerPoint) and functions to import data from Excel. Many of the included functions can be found scattered in other packages and other sources written partly by Titans of R. The reason for collecting them here, was primarily to have them consolidated in ONE instead of dozens of packages (which themselves might depend on other packages which are not needed at all), and to provide a common and consistent interface as far as function and arguments naming, NA handling, recycling rules etc. are concerned. Google style guides were used as naming rules (in absence of convincing alternatives). The 'BigCamelCase' style was consequently applied to functions borrowed from contributed R packages as well.

Suggests RDCOMClient, tcltk, VGAM, R.rsp, testthat (>= 3.0.0)

Depends base, stats, R (>= 4.2.0)

Imports graphics, grDevices, methods, MASS, utils, boot, mvtnorm, expm, Rcpp (>= 0.12.10), rstudioapi, Exact, gld, data.table, readxl, readr, haven, httr, withr, cli, fs

LinkingTo Rcpp

License GPL (>= 2)

LazyLoad yes

LazyData yes

Additional_repositories http://www.omegahat.net/R

URL https://andrisignorell.github.io/DescTools/, https://github.com/AndriSignorell/DescTools/

BugReports https://github.com/AndriSignorell/DescTools/issues

RoxygenNote 7.3.2

Encoding UTF-8

NeedsCompilation yes

SystemRequirements C++17

VignetteBuilder R.rsp

Config/testthat/edition 3

Author Andri Signorell [aut, cre] (ORCID:
<https://orcid.org/0000-0003-4311-1969>),
Ken Aho [ctb],
Andreas Alfons [ctb],
Nanina Anderegg [ctb],
Tomas Aragon [ctb],
Chandima Arachchige [ctb],
Antti Arppe [ctb],
Adrian Baddeley [ctb],
Kamil Barton [ctb],
Ben Bolker [ctb],
Hans W. Borchers [ctb],
Frederico Caeiro [ctb],
Stephane Champely [ctb],
Daniel Chessel [ctb],
Leanne Chhay [ctb],
Nicholas Cooper [ctb],
Clint Cummins [ctb],
Michael Dewey [ctb],
Harold C. Doran [ctb],
Stephane Dray [ctb],
Charles Dupont [ctb],
Dirk Eddelbuettel [ctb],
Claus Ekstrom [ctb],
Martin Elff [ctb],
Jeff Enos [ctb],
Richard W. Farebrother [ctb],
John Fox [ctb],
Romain Francois [ctb],
Michael Friendly [ctb],
Tal Galili [ctb],
Matthias Gamer [ctb],
Joseph L. Gastwirth [ctb],
Vilmantas Gegzna [ctb],
Yulia R. Gel [ctb],
Sereina Graber [ctb],
Juergen Gross [ctb],
Gabor Grothendieck [ctb],
Frank E. Harrell Jr [ctb],
Richard Heiberger [ctb],
Michael Hoehle [ctb],

Christian W. Hoffmann [ctb],
Soeren Hojsgaard [ctb],
Torsten Hothorn [ctb],
Markus Huerzeler [ctb],
Wallace W. Hui [ctb],
Pete Hurd [ctb],
Rob J. Hyndman [ctb],
Christopher Jackson [ctb],
Matthias Kohl [ctb],
Mikko Korpela [ctb],
Max Kuhn [ctb],
Detlew Labes [ctb],
Friederich Leisch [ctb],
Jim Lemon [ctb],
Dong Li [ctb],
Martin Maechler [ctb],
Arni Magnusson [ctb],
Ben Mainwaring [ctb],
Daniel Malter [ctb],
George Marsaglia [ctb],
John Marsaglia [ctb],
Alina Matei [ctb],
David Meyer [ctb],
Weiwen Miao [ctb],
Giovanni Millo [ctb],
Yongyi Min [ctb],
David Mitchell [ctb],
Cyril Flurin Moser [ctb],
Franziska Mueller [ctb],
Markus Naepflin [ctb],
Danielle Navarro [ctb],
Henric Nilsson [ctb],
Klaus Nordhausen [ctb],
Derek Ogle [ctb],
Hong Ooi [ctb],
Nick Parsons [ctb],
Sandrine Pavoine [ctb],
Tony Plate [ctb],
Luke Prendergast [ctb],
Roland Rapold [ctb],
William Revelle [ctb],
Tyler Rinker [ctb],
Brian D. Ripley [ctb],
Caroline Rodriguez [ctb],
Nathan Russell [ctb],
Nick Sabbe [ctb],
Ralph Scherer [ctb],
Venkatraman E. Seshan [ctb],

Michael Smithson [ctb],
Greg Snow [ctb],
Karline Soetaert [ctb],
Werner A. Stahel [ctb],
Alec Stephenson [ctb],
Mark Stevenson [ctb],
Ralf Stubner [ctb],
Matthias Templ [ctb],
Duncan Temple Lang [ctb],
Terry Therneau [ctb],
Yves Tille [ctb],
Luis Torgo [ctb],
Adrian Trapletti [ctb],
Joshua Ulrich [ctb],
Kevin Ushey [ctb],
Jeremy VanDerWal [ctb],
Bill Venables [ctb],
John Verzani [ctb],
Pablo J. Villacorta Iglesias [ctb],
Gregory R. Warnes [ctb],
Stefan Wellek [ctb],
Hadley Wickham [ctb],
Rand R. Wilcox [ctb],
Peter Wolf [ctb],
Daniel Wollschlaeger [ctb],
Joseph Wood [ctb],
Ying Wu [ctb],
Thomas Yee [ctb],
Achim Zeileis [ctb]

Maintainer Andri Signorell <andri@signorell.net>

Repository CRAN

Date/Publication 2025-03-28 12:00:02 UTC

Contents

DescTools-package 13
ABCCoords 25
Abind 26
Abstract 29
AddMonths 31
Agree 32
AllDuplicated 34
AllIdentical 35
AndersonDarlingTest 36
Append 38
AppendRowNames 39
as.matrix.xtabs 40

as.ym . . . 41
AscToChar . . . 42
Asp . . . 43
Association measures . . . 44
Assocs . . . 46
Atkinson . . . 48
AUC . . . 49
AxisBreak . . . 51
axTicks.POSIXct . . . 52
BarnardTest . . . 53
BartelsRankTest . . . 56
BarText . . . 58
Base Conversions . . . 60
Benford . . . 61
Between, Outside . . . 63
Bg . . . 66
BhapkarTest . . . 67
BinomCI . . . 68
BinomCIn . . . 71
BinomDiffCI . . . 73
BinomRatioCI . . . 75
BinTree . . . 78
BootCI . . . 80
BoxCox . . . 81
BoxCoxLambda . . . 82
BoxedText . . . 83
BreslowDayTest . . . 85
BreuschGodfreyTest . . . 87
BrierScore . . . 89
BrierScoreCI . . . 90
BubbleLegend . . . 92
Canvas . . . 93
CartToPol . . . 94
CatTable . . . 95
CCC . . . 96
Clockwise . . . 99
Closest . . . 100
Coalesce . . . 101
CochranArmitageTest . . . 102
CochranQTest . . . 104
CoefVar . . . 106
CohenD . . . 109
CohenKappa . . . 110
CollapseTable . . . 113
ColorLegend . . . 115
ColToGrey . . . 117
ColToHex . . . 118
ColToHsv . . . 119

ColToOpaque . . . 120
ColToRgb . . . 121
ColumnWrap . . . 122
CombPairs . . . 123
CompleteColumns . . . 124
ConDisPairs . . . 124
Conf . . . 125
ConnLines . . . 129
ConoverTest . . . 130
Contrasts . . . 133
ConvUnit . . . 134
Cor . . . 136
CorCI . . . 139
CorPart . . . 140
CorPolychor . . . 142
CountCompCases . . . 143
CountWorkDays . . . 144
CourseData . . . 145
CramerVonMisesTest . . . 146
CronbachAlpha . . . 148
Cross . . . 149
CrossN . . . 150
Cstat . . . 151
CstatCI . . . 153
CutAge . . . 154
CutQ . . . 155
d.countries . . . 157
d.diamonds . . . 158
d.periodic . . . 159
d.pizza . . . 160
d.whisky . . . 161
Datasets for Simulation . . . 163
Date Functions . . . 164
day.name . . . 166
DegToRad . . . 167
Depreciation . . . 168
Desc . . . 169
DescTools Aliases . . . 179
DescTools Palettes . . . 179
DescToolsOptions . . . 181
DigitSum . . . 184
DivCoef . . . 185
DivCoefMax . . . 186
Divisors . . . 188
DoBy . . . 189
DoCall . . . 191
Dot . . . 192
Downloads . . . 193

DrawArc . . . 194
DrawBand . . . 195
DrawBezier . . . 196
DrawCircle . . . 197
DrawEllipse . . . 199
DrawRegPolygon . . . 201
Dummy . . . 202
DunnettTest . . . 204
DunnTest . . . 206
DurbinWatsonTest . . . 209
Entropy . . . 211
Eps . . . 213
ErrBars . . . 214
EtaSq . . . 215
EX . . . 217
ExpFreq . . . 218
Extremes . . . 219
ExtrVal . . . 221
Factorize . . . 222
FctArgs . . . 223
Fibonacci . . . 224
FindColor . . . 225
FindCorr . . . 227
FisherZ . . . 228
FixToTable . . . 229
Format . . . 230
Frac . . . 235
Frechet . . . 236
Freq . . . 238
Freq2D . . . 240
GCD, LCM . . . 241
GenExtrVal . . . 242
GenPareto . . . 244
GenRandGroups . . . 245
GeomSn . . . 246
GeomTrans . . . 247
GetCalls . . . 248
GetCurrWrd . . . 249
GetNewWrd . . . 250
GetNewXL . . . 252
Gini . . . 253
GiniSimpson . . . 255
Gmean . . . 257
Gompertz . . . 259
GoodmanKruskalGamma . . . 260
GoodmanKruskalTau . . . 262
GTest . . . 264
Gumbel . . . 266

Herfindahl . . . 268
HexToCol . . . 269
HexToRgb . . . 270
Hmean . . . 271
HmsToSec . . . 272
HodgesLehmann . . . 273
HoeffD . . . 275
HosmerLemeshowTest . . . 276
HotellingsT2Test . . . 278
HuberM . . . 280
ICC . . . 281
identify.formula . . . 284
IdentifyA . . . 285
ImputeKnn . . . 286
InDots . . . 287
IQRw . . . 288
IsDate . . . 289
IsDichotomous . . . 290
IsEuclid . . . 291
IsOdd . . . 292
IsPrime . . . 292
IsValidHwnd . . . 293
JarqueBeraTest . . . 294
JonckheereTerpstraTest . . . 296
KappaM . . . 298
KendallTauA . . . 300
KendallTauB . . . 301
KendallW . . . 303
Keywords . . . 305
KrippAlpha . . . 306
Label, Unit . . . 307
Lambda . . . 309
Lc . . . 311
LehmacherTest . . . 314
LeveneTest . . . 316
LillieTest . . . 318
lines.lm . . . 319
lines.loess . . . 321
LineToUser . . . 323
LinScale . . . 324
List Variety Of Objects . . . 325
LOCF . . . 326
LOF . . . 327
Logit . . . 328
LogSt . . . 329
MAD . . . 331
MADCI . . . 332
Mar and Mgp . . . 334

matpow . . . 335
mbeta . . . 336
mbinom . . . 337
mchisq . . . 337
Mean . . . 338
MeanAD . . . 339
MeanCI . . . 340
MeanCIn . . . 342
MeanDiffCI . . . 343
MeanSE . . . 345
Measures of Accuracy . . . 346
Measures of Shape . . . 349
Median . . . 351
MedianCI . . . 353
mexp . . . 355
mf . . . 356
mgamma . . . 356
mgeom . . . 357
Mgsub . . . 358
MHChisqTest . . . 359
mhyper . . . 360
Midx . . . 361
MixColor . . . 362
mlnorm . . . 362
mnbinom . . . 363
mnorm . . . 364
Mode . . . 364
MosesTest . . . 366
MoveAvg . . . 368
mpois . . . 369
mt . . . 370
MultinomCI . . . 370
MultMerge . . . 373
NALevel . . . 374
NemenyiTest . . . 375
Nf . . . 377
NPV . . . 378
NZ . . . 379
OddsRatio . . . 380
Order . . . 382
ORToRelRisk . . . 384
Outlier . . . 386
PageTest . . . 387
PairApply . . . 390
ParseFormula . . . 392
ParseSASDatalines . . . 393
PasswordDlg . . . 395
PDFManual . . . 396

PearsonTest . . . 396
PercentRank . . . 398
PercTable . . . 399
Permn . . . 402
Phrase . . . 403
PlotACF . . . 404
PlotArea . . . 405
PlotBag . . . 407
PlotBubble . . . 411
PlotCandlestick . . . 413
PlotCashFlow . . . 414
PlotCirc . . . 416
PlotConDens . . . 417
PlotCorr . . . 419
PlotDot . . . 422
PlotECDF . . . 425
PlotFaces . . . 426
PlotFdist . . . 429
PlotFun . . . 432
PlotLinesA . . . 435
PlotLog . . . 437
PlotMarDens . . . 438
PlotMiss . . . 439
PlotMonth . . . 440
PlotMosaic . . . 441
PlotMultiDens . . . 442
PlotPairs . . . 444
PlotPolar . . . 445
PlotProbDist . . . 448
PlotPyramid . . . 450
PlotQQ . . . 453
PlotTernary . . . 454
PlotTreemap . . . 456
PlotVenn . . . 457
PlotViolin . . . 459
PlotWeb . . . 461
PMT . . . 463
PoissonCI . . . 464
PolarGrid . . . 466
PostHocTest . . . 467
power.chisq.test . . . 470
PowerPoint Interface . . . 471
pRevGumbel . . . 474
Primes . . . 475
PseudoR2 . . . 476
PtInPoly . . . 478
Quantile . . . 479
QuantileCI . . . 480

Quot . . . 482
Range . . . 483
Rank . . . 484
ReadSPSS . . . 486
Recode . . . 486
Recycle . . . 488
RelRisk . . . 489
Rename . . . 491
reorder.factor . . . 492
Rev . . . 494
RevCode . . . 495
RevWeibull . . . 497
RgbToCmy . . . 498
RgbToCol . . . 499
RndPairs . . . 500
RobScale . . . 501
RomanToInt . . . 502
Rotate . . . 503
RoundTo . . . 504
RSessionAlive . . . 506
RSqCI . . . 507
rSum21 . . . 508
RunsTest . . . 509
Sample . . . 512
SampleTwins . . . 513
SaveAs . . . 514
ScheffeTest . . . 515
SD . . . 517
SendOutlookMail . . . 518
SetAlpha . . . 519
SetNames . . . 520
Shade . . . 521
ShapiroFranciaTest . . . 522
SiegelTukeyTest . . . 523
SignTest . . . 526
SmoothSpline . . . 529
Some . . . 531
Some numeric checks . . . 532
SomersDelta . . . 534
Sort . . . 535
SortMixed . . . 537
SpearmanRho . . . 539
split.formula . . . 541
SplitAt . . . 542
SplitPath . . . 543
SplitToCol . . . 544
SplitToDummy . . . 545
SpreadOut . . . 546

Stamp . . . 547
StdCoef . . . 548
Str . . . 550
StrAbbr . . . 551
StrAlign . . . 552
Strata . . . 554
StrCap . . . 556
StrChop . . . 557
StrCountW . . . 558
StrDist . . . 559
StrExtract . . . 561
StripAttr . . . 562
StrIsNumeric . . . 563
StrLeft, StrRight . . . 564
StrPad . . . 565
StrPos . . . 566
StrRev . . . 567
StrSpell . . . 567
StrSplit . . . 568
StrTrim . . . 569
StrTrunc . . . 571
StrVal . . . 572
StuartMaxwellTest . . . 573
StuartTauC . . . 575
SumCI . . . 577
SysInfo . . . 578
TextContrastColor . . . 578
TextToTable . . . 580
TheilU . . . 581
TitleRect . . . 582
TMod . . . 583
ToLong, ToWide . . . 585
TOne . . . 587
ToWrd . . . 591
ToWrdB . . . 595
ToWrdPlot . . . 596
Triangular . . . 598
Trim . . . 601
TTestA . . . 602
TukeyBiweight . . . 604
TwoGroups . . . 606
UncertCoef . . . 607
UnirootAll . . . 609
Untable . . . 611
Unwhich . . . 613
VanWaerdenTest . . . 614
Var . . . 616
VarCI . . . 617

VarTest . . . 619
VecRot . . . 621
VIF . . . 622
Vigenere . . . 624
VonNeumannTest . . . 625
wdConst . . . 626
Winsorize . . . 627
WithOptions . . . 628
WoolfTest . . . 629
WrdBookmark . . . 630
WrdCaption . . . 632
WrdCellRange . . . 633
WrdFont . . . 634
WrdFormatCells . . . 635
WrdMergeCells . . . 636
WrdPageBreak . . . 637
WrdParagraphFormat . . . 638
WrdPlot . . . 640
WrdSaveAs . . . 642
WrdStyle . . . 643
WrdTable . . . 644
WrdTableBorders . . . 645
WrdTableHeading . . . 646
XLDateToPOSIXct . . . 647
XLGetRange . . . 648
XLSaveAs . . . 651
XLView . . . 652
YuenTTest . . . 654
ZeroIfNA . . . 657
Zodiac . . . 658
ZTest . . . 659
%like% . . . 661
%nin% . . . 662
%overlaps% . . . 663
%c% . . . 665

Index **666**

`DescTools-package` *Tools for Descriptive Statistics and Exploratory Data Analysis*

Description

DescTools is an extensive collection of miscellaneous basic statistics functions and comfort wrappers not available in the R basic system for efficient description of data. The author's intention was to create a toolbox, which facilitates the (notoriously time consuming) first descriptive tasks in data analysis, consisting of calculating descriptive statistics, drawing graphical summaries and reporting the results. Special attention was paid to the integration of various approaches to the calculation

of confidence intervals. For most basic statistics functions, variants are included that allow the use of weights. The package contains furthermore functions to produce documents using MS Word (or PowerPoint) and functions to import data from Excel.
A considerable part of the included functions can be found scattered in other packages and other sources written partly by Titans of R. The reason for collecting them here, was primarily to have them consolidated in ONE instead of dozens of packages (which themselves might depend on other packages which are not needed at all), and to provide a common and consistent interface as far as function and arguments naming, NA handling, recycling rules etc. are concerned. Google style guides were used as naming rules (in absence of convincing alternatives). The 'CamelStyle' was consequently applied to functions borrowed from contributed R packages as well.

Feedback, feature requests, bugreports and other suggestions are welcome! Please report problems to Stack Overflow using tag [desctools] or directly to the maintainer.

Details

A grouped list of the functions:

Operators, calculus, transformations:

%()%	Between operators determine if a value lies within a range [a,b]
%)(%	Outside operators: %)(%, %](%, %)[%, %][%
%nin%	"not in" operator
%overlaps%	Do two collections have common elements?
%like%, %like any%	Simple operator to search for a specified pattern
%^%	Powers of matrices
Interval	The number of days of the overlapping part of two date periods
AUC	Area under the curve
Primes	Find all primes less than n
Factorize	Prime factorization of integers
Divisors	All divisors of an integer
GCD	Greatest common divisor
LCM	Least common multiple
Permn	Determine all possible permutations of a set
Fibonacci	Generates single Fibonacci numbers or a Fibonacci sequence
DigitSum	Digit sum of a number
Frac	Return the fractional part of a numeric value
Ndec	Count decimal places of a number
MaxDigits	Maximum used digits for a vector of numbers
Prec	Precision of a number
BoxCox, BoxCoxInv	Box Cox transformation and its inverse transformation
BoxCoxLambda	Return the optimal lambda for a BoxCox transformation
LogSt, LogStInv	Calculate started logarithmic transformation and it's inverse
Logit, LogitInv	Generalized logit and inverse logit function
LinScale	Simple linear scaling of a vector x
Winsorize	Data cleaning by winsorization

Trim	Trim data by omitting outlying observations
CutQ	Cut a numeric variable into quartiles or other quantiles
Recode	Recode a factor with altered levels
Rename	Change name(s) of a named object
Sort	Sort extension for matrices and data.frames
SortMixed, OrderMixed	Mixed sort order
Rank	Calculate ranks including dense type for ties
PercentRank	Calculate the percent rank
RoundTo	Round to a multiple
Large, Small	Returns the kth largest, resp. smallest values
HighLow	Combines `Large` and `Small`.
Rev	Reverses the order of rows and/or columns of a matrix or a data.frame
Untable	Recreates original list based on a n-dimensional frequency table
CollapseTable	Collapse some rows/columns in a table.
Dummy	Generate dummy codes for a factor
FisherZ, FisherZInv	Fisher's z-transformation and its inverse
Midx	Calculate sequentially the midpoints of the elements of a vector
Unwhich	Inverse function to `which`, create a logical vector/matrix from indices
Vigenere	Implements a Vigenere cypher, both encryption and decryption
BinTree, PlotBinTree	Create and plot a binary tree structure with a given length

Information and manipulation functions:

AllDuplicated	Find all values involved in ties
Closest	Return the value in a vector being closest to a given one
Coalesce	Return the first value in a vector not being `NA`
ZeroIfNA, NAIfZero	Replace NAs by 0, resp. vice versa
Impute	Replace NAs by the median or another value
LOCF	Imputation of datapoints following the "last observation carried forward" rule
CombN	Returns the number of subsets out of a list of elements
CombSet	Generates all possible subsets out of a list of elements
CombPairs	Generates all pairs out of one or two sets of elements
SampleTwins	Create sample using stratifying groups
RndPairs	Create pairs of correlated random numbers
RndWord	Produce random combinations of characters
IsNumeric	Check a vector for being numeric, zero Or a whole number
IsWhole	Is x a whole number?
IsDichotomous	Check if x contains exactly 2 values
IsOdd	Is x even or odd?
IsPrime	Is x a prime number?
IsZero	Is numeric(x) == 0, say x < machine.eps?
IsEuclid	Check if a distance matrix is euclidean
Label, Unit	Get or set the `label`, resp. `unit`, attribute of an object
Abind	Bind matrices to n-dimensional arrays
Append	Append elements to several classes of objects
VecRot, VecShift	Shift the elements of a vector in a circular mode to the right or to the left by n characters.

Clockwise	Transform angles from counter clock into clockwise mode
split.formula	A formula interface for the base function split
reorder.factor	Reorder the levels of a factor
ToLong, ToWide	Simple reshaping of a vector
SetNames	Set the names, rownames or columnnames in an object and return it
Some	Return some randomly chosen elements of an object
SplitAt	Split a vector into several pieces at given positions
SplitToCol	Splits the columns of a data frame using a split character
SplitPath	Split a path string in drive, path, filename
Str	Compactly display the structure of any R object
TextToTable	Converts a string to a table

String functions:

StrCountW	Count the words in a string
StrTrim	Delete white spaces from a string
StrTrunc	Truncate string on a given length and add ellipses if it really was truncated
StrLeft, StrRight	Returns the left/right part or the a string.
StrAlign	Align strings to the left/right/center or to a given character
StrAbbr	Abbreviates a string
StrCap	Capitalize the first letter of a string
StrPad	Fill a string with defined characters to fit a given length
StrRev	Reverse a string
StrChop	Split a string by a fixed number of characters.
StrExtract	Extract a part of a string, defined as regular expression.
StrVal	Extract numeric values from a string
StrIsNumeric	Check whether a string does only contain numeric data
StrPos	Find position of first occurrence of a string in another one
StrDist	Compute Levenshtein or Hamming distance between strings
FixToTable	Create table out of a running text, by using columns of spaces as delimiter

Conversion functions:

AscToChar, CharToAsc	Converts ASCII codes to characters and vice versa
DecToBin, BinToDec	Converts numbers from binmode to decimal and vice versa
DecToHex, HexToDec	Converts numbers from hexmode to decimal and vice versa
DecToOct, OctToDec	Converts numbers from octmode to decimal and vice versa
DegToRad, RadToDeg	Convert degrees to radians and vice versa
CartToPol, PolToCart	Transform cartesian to polar coordinates and vice versa
CartToSph, SphToCart	Transform cartesian to spherical coordinates and vice versa
RomanToInt	Convert roman numerals to integers
RgbToLong, LongToRgb	Convert a rgb color to a long number and vice versa
ColToGray, ColToGrey	Convert colors to gcrey/grayscale
ColToHex, HexToCol	Convert a color into hex string
HexToRgb	Convert a hexnumber to an RGB-color
ColToHsv	R color to HSV conversion
ColToRgb, RgbToCol	Color to RGB conversion and back

ConvUnit	Return the most common unit conversions

Colors:

SetAlpha	Add transperancy (alpha channel) to a color.
ColorLegend	Add a color legend to a plot
FindColor	Get color on a defined color range
MixColor	Get the mix of two colors
TextContrastColor	Choose textcolor depending on background color
Pal	Some custom color palettes

Plots (low level):

Canvas	Canvas for geometric plotting
Mar	Set margins more comfortably.
Asp	Return aspect ratio of the current plot
LineToUser	Convert line coordinates to user coordinates
lines.loess	Add a loess smoother and its CIs to an existing plot
lines.lm	Add the prediction of linear model and its CIs to a plot
lines.smooth.spline	Add the prediction of a smooth.spline and its CIs to a plot
BubbleLegend	Add a legend for bubbles to a bubble plot
TitleRect	Add a main title to a plot surrounded by a rectangular box
BarText	Add the value labels to a barplot
ErrBars	Add horizontal or vertical error bars to an existing plot
DrawArc, DrawRegPolygon	Draw elliptic, circular arc(s) or regular polygon(s)
DrawCircle, DrawEllipse	Draw a circle, a circle annulus or a sector or an annulus
DrawBezier	Draw a Bezier curve
DrawBand	Draw confidence band
BoxedText	Add text surrounded by a box to a plot
Rotate	Rotate a geometric structure
SpreadOut	Spread out a vector of numbers so that there is a minimum interval between any two elements. This can be used to place textlabels in a plot so that they do not overlap.
IdentifyA	Helps identifying all the points in a specific area.
identify.formula	Formula interface for `identify`.
PtInPoly	Identify all the points within a polygon.
ConnLines	Calculate and insert connecting lines in a barplot
AxisBreak	Place a break mark on an axis
Shade	Produce a shaded curve
Stamp	Stamp the current plot with Date/Time/Directory or any other expression

Plots (high level):

PlotACF, PlotGACF	Create a combined plot of a time series including its autocorrelation and partial autocorrelation
PlotMonth	Plot seasonal effects of a univariate time series
PlotArea	Create an area plot
PlotBag	Create a two-dimensional boxplot

PlotBagPairs	Produce pairwise 2-dimensional boxplots (bagplot)
PlotBubble	Draw a bubble plot
PlotCandlestick	Plot candlestick chart
PlotCirc	Create a circular plot
PlotCorr	Plot a correlation matrix
PlotDot	Plot a dotchart with confidence intervals
PlotFaces	Produce a plot of Chernoff faces
PlotFdist	Frequency distribution plot, combination of histogram, boxplot and ecdf.plot
PlotMarDens	Scatterplot with marginal densities
PlotMultiDens	Plot multiple density curves
PlotPolar	Plot values on a circular grid
PlotFun	Plot mathematical expression or a function
PolarGrid	Plot a grid in polar coordinates
PlotPyramid	Pyramid plot (back-back histogram)
PlotTreemap	Plot of a treemap.
PlotVenn	Plot a Venn diagram
PlotViolin	Plot violins instead of boxplots
PlotQQ	QQ-plot for an optional distribution
PlotWeb	Create a web plot
PlotTernary	Create a triangle or ternary plot
PlotMiss	Plot missing values
PlotECDF	Plot empirical cumulative distribution function
PlotLinesA	Plot the columns of one matrix against the columns of another
PlotLog	Create a plot with logarithmic axis and log grid
PlotMosaic	Plots a mosaic describing a contingency table in array form

Distributions:

_Benf	Benford distribution, including qBenf, dBenf, rBenf
_ExtrVal	Extreme value distribution (dExtrVal)
_Frechet	Frechet distribution (dFrechet)
_GenExtrVal	Generalized Extreme Value Distribution (dGenExtrVal)
_GenPareto	Generalized Pareto Distribution (dGenPareto)
_Gompertz	Gompertz distribution (dGompertz)
_Gumbel	Gumbel distribution (dGumbel)
_NegWeibull	Negative Weibull distribution (dNegWeibull)
_Order	Distributions of Order Statistics (dOrder)
_RevGumbel	Reverse Gumbel distribution (dRevGumbel),
_RevGumbelExp	Expontial reverse Gumbel distribution (quantile only)
_RevWeibull	Reverse Weibull distribution (dRevWeibull)

Statistics:

Freq	Univariate frequency table
PercTable	Bivariate percentage table
Margins	(Extended) margin tables of a table
ExpFreq	Expected frequencies of a n-dimensional table

Mode	Mode, the most frequent value (including frequency)
Gmean, Gsd	Geometric mean and geometric standard deviation
Hmean	Harmonic Mean
Median	Extended median function supporting weights and ordered factors
HuberM, TukeyBiweight	Huber M-estimator of location and Tukey's biweight robust mean
HodgesLehmann	the Hodges-Lehmann estimator
HoeffD	Hoeffding's D statistic
MeanSE	Standard error of mean
MeanCI, MedianCI	Confidence interval for the mean and median
MeanDiffCI	Confidence interval for the difference of two means
MoveAvg	Moving average
MeanAD	Mean absolute deviation
VarCI	Confidence interval for the variance
CoefVar	Coefficient of variation and its confidence interval
RobScale	Robust data standardization
Range	(Robust) range
BinomCI, MultinomCI	Confidence intervals for binomial and multinomial proportions
BinomDiffCI	Calculate confidence interval for a risk difference
BinomRatioCI	Calculate confidence interval for the ratio of binomial proportions.
PoissonCI	Confidence interval for a Poisson lambda
Skew, Kurt	Skewness and kurtosis
YuleQ, YuleY	Yule's Q and Yule's Y
TschuprowT	Tschuprow's T
Phi, ContCoef, CramerV	Phi, Pearson's Contingency Coefficient and Cramer's V
GoodmanKruskalGamma	Goodman Kruskal's gamma
KendallTauA	Kendall's tau-a
KendallTauB	Kendall's tau-b
StuartTauC	Stuart's tau-c
SomersDelta	Somers' delta
Lambda	Goodman Kruskal's lambda
GoodmanKruskalTau	Goodman Kruskal's tau
UncertCoef	Uncertainty coefficient
Entropy, MutInf	Shannon's entropy, mutual information
DivCoef, DivCoefMax	Rao's diversity coefficient ("quadratic entropy")
TheilU	Theil's U1 and U2 coefficient
Assocs	Combines the association measures above.
OddsRatio, RelRisk	Odds ratio and relative risk
ORToRelRisk	Transform odds ratio to relative risk
CohenKappa, KappaM	Cohen's Kappa, weighted Kappa and Kappa for more than 2 raters
CronbachAlpha	Cronbach's alpha
ICC	Intraclass correlations
KrippAlpha	Return Kripp's alpha coefficient
KendallW	Compute the Kendall coefficient of concordance
Lc	Calculate and plot Lorenz curve
Gini, Atkinson	Gini- and Atkinson coefficient
Herfindahl, Rosenbluth	Herfindahl- and Rosenbluth coefficient
GiniSimpson	Compute Gini-Simpson Coefficient

CorCI	Confidence interval for Pearson's correlation coefficient
CorPart	Find the correlations for a set x of variables with set y removed
CorPolychor	Polychoric correlation coefficient
SpearmanRho	Spearman rank correlation and its confidence intervals
ConDisPairs	Return concordant and discordant pairs of two vectors
FindCorr	Determine highly correlated variables
CohenD	Cohen's Effect Size
EtaSq	Effect size calculations for ANOVAs
Contrasts	Generate pairwise contrasts for using in a post-hoc test
Strata	Stratified sampling with equal/unequal probabilities
Outlier	Outliers following Tukey's boxplot definition
LOF	Local outlier factor
BrierScore	Brier score, assessing the quality of predictions of binary events
Cstat	C statistic, equivalent to the area under the ROC curve)
CCC	Lin's concordance correlation coef for agreement on a continuous measure
MAE	Mean absolute error
MAPE, SMAPE	Mean absolute and symmetric mean absolute percentage error
MSE, RMSE	Mean squared error and root mean squared error
NMAE, NMSE	Normalized mean absolute and mean squared error
Conf	Confusion matrix, a cross-tabulation of observed and predicted classes with associated statistics
Sens, Spec	Sensitivity and specificity
PseudoR2	Variants of pseudo R squared statistics: McFadden, Aldrich-Nelson, Nagelkerke, CoxSnell, Effron, McKelvey-Zavoina, Tjur
Mean, SD, Var, IQRw	Variants of base statistics, allowing to define weights: Mean,
Quantile, MAD, Cor	standard deviation, variance, quantile, mad, correlation
VIF, StdCoef	Variance inflation factors and standardised coefficents for linear models

Tests:

SignTest	Signtest to test whether two groups are equally sized
ZTest	Z–test for known population variance
TTestA	Student's t-test based on sample statistics
JonckheereTerpstraTest	Jonckheere-Terpstra trend test for medians
PageTest	Page test for ordered alternatives
CochranQTest	Cochran's Q-test to find differences in matched sets of three or more frequencies or proportions.
VarTest	ChiSquare test for one variance and F test for two variances
SiegelTukeyTest	Siegel-Tukey test for equality in variability
SiegelTukeyRank	Calculate Siegel-Tukey's ranks (auxiliary function)
LeveneTest	Levene's test for homogeneity of variance
MosesTest	Moses Test of extreme reactions
RunsTest	Runs test for detecting non-randomness
DurbinWatsonTest	Durbin-Watson test for autocorrelation
BartelsRankTest	Bartels rank test for randomness
JarqueBeraTest	Jarque-Bera Test for normality
AndersonDarlingTest	Anderson-Darling test for normality
CramerVonMisesTest	Cramer-von Mises test for normality

LillieTest	Lilliefors (Kolmogorov-Smirnov) test for normality
PearsonTest	Pearson chi-square test for normality
ShapiroFranciaTest	Shapiro-Francia test for normality
MHChisqTest	Mantel-Haenszel Chisquare test
StuartMaxwellTest	Stuart-Maxwell marginal homogeneity test
LehmacherTest	Lehmacher marginal homogeneity test
CochranArmitageTest	Cochran-Armitage test for trend in binomial proportions
BreslowDayTest, WoolfTest	Test for homogeneity on 2x2xk tables over strata
PostHocTest	Post hoc tests by Scheffe, LSD, Tukey for a aov-object
ScheffeTest	Multiple comparisons Scheffe test
DunnTest	Dunn's test of multiple comparisons
DunnettTest	Dunnett's test of multiple comparisons
ConoverTest	Conover's test of multiple comparisons (following a kruskal test)
NemenyiTest	Nemenyi's test of multiple comparisons
HotellingsT2Test	Hotelling's T2 test for the one and two sample case
YuenTTest	Yuen's robust t-Test with trimmed means and winsorized variances
BarnardTest	Barnard's test for 2x2 tables
BreuschGodfreyTest	Breusch-Godfrey test for higher-order serial correlation.
GTest	Chi-squared contingency table test and goodness-of-fit test
HosmerLemeshowTest	Hosmer-Lemeshow goodness of fit tests
VonNeumannTest	Von Neumann's successive difference test

Date functions:

day.name, day.abb	Defined names of the days
AddMonths	Add a number of months to a given date
IsDate	Check whether x is a date object
IsWeekend	Check whether x falls on a weekend
IsLeapYear	Check whether x is a leap year
LastDayOfMonth	Return the last day of the month of the date x
DiffDays360	Calculate the difference of two dates using the 360-days system
Date	Create a date from numeric representation of year, month, day
Day, Month, Year	Extract part of a date
Hour, Minute, Second	Extract part of time
Week, Weekday	Returns ISO week and weekday of a date
Quarter	Quarter of a date
Timezone	Timezone of a POSIXct/POSIXlt date
YearDay, YearMonth	The day in the year of a date
Now, Today	Get current date or date-time
HmsToSec, SecToHms	Convert h:m:s times to seconds and vice versa
Overlap	Determine if and how extensively two date ranges overlap
Zodiac	The zodiac sign of a date :-)

Finance functions:

OPR	One period returns (simple and log returns)
NPV	Net present value
NPVFixBond	Net present value for fix bonds

IRR	Internal rate of return
YTM	Return yield to maturity for a bond
SLN, DB, SYD	Several methods of depreciation of an asset

GUI-Helpers:

PasswordDlg	Display a dialog containing an edit field, showing only ***.

Reporting, InOut:

CatTable	Print a table with the option to have controlled linebreaks
Format, Fmt	Easy format for numbers and dates
Desc	Produce a rich description of an object
Abstract	Display compact overview of the structure of a data frame
TMod	Create comparison table for (general) linear models
TOne	Create "Table One"" describing baseline characteristics
GetNewWrd, GetNewXL, GetNewPP	Create a new Word, Excel or PowerPoint Instance
GetCurrWrd, GetCurrXL, GetCurrPP	Get a handle to a running Word, Excel or PowerPoint instance
WrdKill, XLKill	Ends a (possibly hidden) Word/Excel process
IsValidHwnd	Check if the handle to a MS Office application is valid or outdated
WrdCaption	Insert a title in Word
WrdFont	Get and set the font for the current selection in Word
WrdParagraphFormat	Get and set the paragraph format
WrdTable	Create a table in Word
WrdCellRange	Select a cell range of a table in Word
WrdMergeCells	Merge cells of a table in Word
WrdFormatCells	Format selected cells of a table in word
WrdTableBorders	Set or edit table border style of a table in Word
ToWrd, ToXL	Mord flexible wrapper to send diverse objects to Word, resp. Excel
WrdPlot	Insert the active plot to Word
WrdInsertBookmark	Insert a new bookmark in a Word document
WrdDeleteBookmark	Delete an existing bookmark in a Word document
WrdGoto	Place cursor to a specific bookmark, or another text position.
WrdUpdateBookmark	Update the text of a bookmark's range
WrdSaveAs	Saves documents in Word
WrdStyle	Get and set the style of a paragraph in Word
XLDateToPOSIXct	Convert XL-Date format to POSIXct format
XLGetRange	Get the values of one or several cell range(s) in Excel
XLGetWorkbook	Get the values of all sheets of an Excel workbook
XLView	Use Excel as viewer for a data.frame
PpPlot	Insert active plot to PowerPoint
PpAddSlide	Adds a slide to a PowerPoint presentation
PpText	Adds a textbox with text to a PP-presentation
ParseSASDatalines	Parse a SAS "datalines" statement to read data

Tools:

PairApply	Helper for calculating functions pairwise

LsFct, LsObj	List the functions (or the data, all objects) of a package
FctArgs	Retrieve the arguments of a functions
InDots	Check if an argument is contained in ... argument and return it's value
ParseFormula	Parse a formula and return the splitted parts of if
Recycle	Recycle a list of elements to the maximal found dimension
Keywords	Get the keywords of a man page
SysInfo	Get some more information about system and environment
DescToolsOptions	Get the DescTools specific options
PDFManual	Get the pdf-manual of any package on CRAN and open it

Data:

d.pizza	Synthetic dataset created for testing the description
d.whisky	of Scotch Single Malts

Reference Data:

d.units, d.prefix	Unit conversion factors and metric prefixes
d.periodic	Periodic table of elements
d.countries	ISO 3166-1 country codes
roulette, cards, tarot	Datasets for probabilistic simulation

Warning

This package is still under development. Although the code seems meanwhile quite stable, until release of version 1.0 (which is expected in hmm: near future?) you should be aware that everything in the package might be subject to change. Backward compatibility is not yet guaranteed. Functions may be deleted or renamed and new syntax may be inconsistent with earlier versions. By release of version 1.0 the "deprecated-defunct process" will be installed.

MS-Office

To make use of MS-Office features you must have Office in one of its variants installed, as well as the package **RDCOMClient**. This package uses the COM interface to control the Office applications. There is no direct equivalent to COM interface for Mac or Linux, hence the use of these functions is restricted to Windows systems. All `Wrd*`, `XL*` and `Pp*` functions require this basis to run.

RDCOMClient can be installed with:

```
install.packages("RDCOMClient", repos="http://www.omegahat.net/R")
```

The omegahat repository does not benefit from the same update service as CRAN. So you may be forced to install a package compiled with an earlier version, which usually is no problem. For R 4.2 you can use:

```
url <- "http://www.omegahat.net/R/bin/windows/contrib/4.2/RDCOMClient_0.96-1.zip"
install.packages(url, repos=NULL, type="binary")
```

RDCOMClient does not exist for Mac or Linux, sorry.

Author(s)

Andri Signorell
Helsana Versicherungen AG, Health Sciences, Zurich
HWZ University of Applied Sciences in Business Administration Zurich.

R is a community project. This can also be seen in this package, which contains R source code and/or documentation previously published elsewhere by (in alphabetical order):

Ken Aho, Andreas Alfons, Nanina Anderegg, Tomas Aragon, Antti Arppe, Adrian Baddeley, Kamil Barton, Ben Bolker, Hans W. Borchers, Frederico Caeiro, Stephane Champely, Daniel Chessel, Leanne Chhay, Clint Cummins, Michael Dewey, Harold C. Doran, Stephane Dray, Charles Dupont, Dirk Eddelbuettel, Jeff Enos, Claus Ekstrom, Martin Elff, Kamil Erguler, Richard W. Farebrother, John Fox, Romain Francois, Michael Friendly, Tal Galili, Matthias Gamer, Joseph L. Gastwirth, Yulia R. Gel, Juergen Gross, Gabor Grothendieck, Frank E. Harrell Jr, Richard Heiberger, Michael Hoehle, Christian W. Hoffmann, Soeren Hojsgaard, Torsten Hothorn, Markus Huerzeler, Wallace W. Hui, Pete Hurd, Rob J. Hyndman, Pablo J. Villacorta Iglesias, Christopher Jackson, Matthias Kohl, Mikko Korpela, Max Kuhn, Detlew Labes, Duncan Temple Lang, Friederich Leisch, Jim Lemon, Dong Li, Martin Maechler, Arni Magnusson, Daniel Malter, George Marsaglia, John Marsaglia, Alina Matei, David Meyer, Weiwen Miao, Giovanni Millo, Yongyi Min, David Mitchell, Franziska Mueller, Markus Naepflin, Danielle Navarro, Henric Nilsson, Klaus Nordhausen, Derek Ogle, Hong Ooi, Nick Parsons, Sandrine Pavoine, Tony Plate, Roland Rapold, William Revelle, Tyler Rinker, Brian D. Ripley, Caroline Rodriguez, Nathan Russell, Nick Sabbe, Venkatraman E. Seshan, Greg Snow, Michael Smithson, Karline Soetaert, Werner A. Stahel, Alec Stephenson, Mark Stevenson, Matthias Templ, Terry Therneau, Yves Tille, Adrian Trapletti, Joshua Ulrich, Kevin Ushey, Jeremy VanDerWal, Bill Venables, John Verzani, Gregory R. Warnes, Stefan Wellek, Hadley Wickham, Rand R. Wilcox, Peter Wolf, Daniel Wollschlaeger, Thomas Yee, Achim Zeileis

Special thanks go to Beat Bruengger, Mathias Frueh, Daniel Wollschlaeger, Vilmantas Gegzna for their valuable contributions and testing.

The good things come from all these guys, any problems are likely due to my tweaking. Thank you all!

Maintainer: Andri Signorell <andri@signorell.net>

Examples

```
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# There are no examples defined here. But see the demos:
#
# demo(describe)
# demo(plots))
#
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

ABCCoords *Coordinates for "bottomright", etc.*

Description

Return the xy.coordinates for the literal positions "bottomright", etc. as used to place legends.

Usage

```
ABCCoords(x = "topleft", region = "figure", cex = NULL, linset = 0, ...)
```

Arguments

x	one out of "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right", "center"
region	one out of plot or figure
cex	the character extension for the text.
linset	line inset in lines of text.
...	the dots are passed to the strwidth() and strheight() functions in case there where more specific text formats.

Details

The same logic as for the legend can be useful for placing texts, too. This function returns the coordinates for the text, which can be used in the specific text functions.

Value

nothing returned

Author(s)

Andri Signorell <andri@signorell.net>

See Also

text, BoxedText

Examples

```
plot(x = rnorm(10), type="n", xlab="", ylab="")
# note that plot.new() has to be called before we can grab the geometry
ABCCoords("bottomleft")

lapply(c("bottomleft", "left"), ABCCoords)

plot(x = rnorm(10), type="n", xlab="", ylab="")
```

```
text(x=(xy <- ABCCoords("bottomleft", region = "plot"))$xy,
     labels = "My Maybe Long Text", adj = xy$adj, xpd=NA)

text(x=(xy <- ABCCoords("topleft", region = "figure"))$xy,
     labels = "My Maybe Long Text", adj = xy$adj, xpd=NA)

plot(x = rnorm(10), type="n", xlab="", ylab="")
sapply(c("topleft", "top", "topright", "left", "center",
         "right", "bottomleft", "bottom", "bottomright"),
       function(x)
         text(x=(xy <- ABCCoords(x, region = "plot", linset=1))$xy,
              labels = "MyMarginText", adj = xy$adj, xpd=NA)
)


plot(x = rnorm(100), type="n", xlab="", ylab="",
     panel.first={Bg(c("red", "lightyellow"))
             grid()})
xy <- ABCCoords("topleft", region = "plot")
par(xpd=NA)
BoxedText(x=xy$xy$x, y=xy$xy$y, xpad = 1, ypad = 1,
          labels = "My Maybe Long Text", adj = xy$adj, col=SetAlpha("green", 0.8))
```

Abind *Combine Multidimensional Arrays*

Description

Base R functions cbind and rbind bind columns and rows, but there's no built-in function for binding higher dimensional datastructures like matrices. Abind takes a sequence of vectors, matrices, or arrays and produces a single array of the same or higher dimension.

Usage

```
Abind(..., along = N, rev.along = NULL, new.names = NULL, force.array = TRUE,
      make.names = FALSE, use.first.dimnames = FALSE, hier.names = FALSE,
      use.dnns = FALSE)
```

Arguments

...	Any number of vectors, matrices, arrays, or data frames. The dimensions of all the arrays must match, except on one dimension (specified by along=). If these arguments are named, the name will be used for the name of the dimension along which the arrays are joined. Vectors are treated as having a dim attribute of length one. Alternatively, there can be one (and only one) list argument supplied, whose components are the objects to be bound together. Names of the list components are treated in the same way as argument names.

`along` The dimension along which to bind the arrays. The default is the last dimension, i.e., the maximum length of the dim attribute of the supplied arrays. `along=` can take any non-negative value up to the minimum length of the dim attribute of supplied arrays plus one. When `along=` has a fractional value, a value less than 1, or a value greater than N (N is the maximum of the lengths of the dim attribute of the objects to be bound together), a new dimension is created in the result. In these cases, the dimensions of all arguments must be identical.

`rev.along` Alternate way to specify the dimension along which to bind the arrays: `along = N + 1 - rev.along`. This is provided mainly to allow easy specification of `along = N + 1` (by supplying `rev.along=0`). If both `along` and `rev.along` are supplied, the supplied value of `along` is ignored.

`new.names` If new.names is a list, it is the first choice for the dimnames attribute of the result. It should have the same structure as a dimnames attribute. If the names for a particular dimension are `NULL`, names for this dimension are constructed in other ways.

If `new.names` is a character vector, it is used for dimension names in the same way as argument names are used. Zero length ("") names are ignored.

`force.array` If `FALSE`, rbind or cbind are called when possible, i.e., when the arguments are all vectors, and along is not 1, or when the arguments are vectors or matrices or data frames and along is 1 or 2. If rbind or cbind are used, they will preserve the data.frame classes (or any other class that r/cbind preserve). Otherwise, Abind will convert objects to class array. Thus, to guarantee that an array object is returned, supply the argument `force.array=TRUE`. Note that the use of rbind or cbind introduces some subtle changes in the way default dimension names are constructed: see the examples below.

`make.names` If `TRUE`, the last resort for dimnames for the along dimension will be the deparsed versions of anonymous arguments. This can result in cumbersome names when arguments are expressions. The default is `FALSE`.

`use.first.dimnames`
When dimension names are present on more than one argument, should dimension names for the result be take from the first available (the default is to take them from the last available, which is the same behavior as `rbind` and `cbind`.)

`hier.names` If `TRUE`, dimension names on the concatenated dimension will be composed of the argument name and the dimension names of the objects being bound. If a single list argument is supplied, then the names of the components serve as the argument names. `hier.names` can also have values `"before"` or `"after"`; these determine the order in which the argument name and the dimension name are put together (`TRUE` has the same effect as `"before"`).

`use.dnns` (default `FALSE`) Use names on dimensions, e.g., so that `names(dimnames(x))` is non-empty. When there are multiple possible sources for names of dimnames, the value of `use.first.dimnames` determines the result.

Details

The dimensions of the supplied vectors or arrays do not need to be identical, e.g., arguments can be a mixture of vectors and matrices. `Abind` coerces arguments by the addition of one dimension in

order to make them consistent with other arguments and along=. The extra dimension is added in the place specified by along=.

The default action of Abind is to concatenate on the last dimension, rather than increase the number of dimensions. For example, the result of calling Abind with vectors is a longer vector (see first example below). This differs from the action of rbind and cbind which is to return a matrix when called with vectors. Abind can be made to behave like cbind on vectors by specifying along=2, and like rbind by specifying along=0.

The dimnames of the returned object are pieced together from the dimnames of the arguments, and the names of the arguments. Names for each dimension are searched for in the following order: new.names, argument name, dimnames (or names) attribute of last argument, dimnames (or names) attribute of second last argument, etc. (Supplying the argument use.first.dimnames=TRUE changes this to cause Abind to use dimnames or names from the first argument first. The default behavior is the same as for rbind and cbind: use dimnames from later arguments.) If some names are supplied for the along dimension (either as argument names or dimnames in arguments), names are constructed for anonymous arguments unless maken.ames=FALSE.

Value

An array with a dim attribute calculated as follows.

Let rMin=min(sapply(list(...), function(x) length(dim(x)))) and rMax=max(sapply(list(...), function(x) length(dim(x)))) (where the length of the dimensions of a vector are taken to be 1). Then rMax should be equal to or one greater than rMin.

If along refers to an existing dimension, then the length of the dim attribute of the result is rMax. If along does not refer to an existing dimension, then rMax should equal rMin and the length of the dim attribute of the result will be rMax+1.

rbind or cbind are called to compute the result if (a) force.array=FALSE; and (b) the result will be a two-dimensional object.

Note

It would be nice to make Abind() an S3 generic, but S3 generics cannot dispatch off anonymous arguments.

The ability of Abind() to accept a single list argument removes much of the need for constructs like do.call("Abind", list.of.arrays). Instead, just do Abind(list.of.arrays). The direct construct is preferred because do.call() construct can sometimes consume more memory during evaluation.

Author(s)

Tony Plate <tplate@acm.org> and Richard Heiberger

See Also

rbind, cbind, array

Examples

```
# Five different ways of binding together two matrices
x <- matrix(1:12, 3, 4)
y <- x + 100
dim(Abind(x, y, along=0))     # binds on new dimension before first
dim(Abind(x, y, along=1))     # binds on first dimension
dim(Abind(x, y, along=1.5))
dim(Abind(x, y, along=2))
dim(Abind(x, y, along=3))
dim(Abind(x, y, rev.along=1)) # binds on last dimension
dim(Abind(x, y, rev.along=0)) # binds on new dimension after last

# Unlike cbind or rbind in that the default is to bind
# along the last dimension of the inputs, which for vectors
# means the result is a vector (because a vector is
# treated as an array with length(dim(x))==1).
Abind(x=1:4, y=5:8)

# Like cbind
Abind(x=1:4, y=5:8, along=2)
Abind(x=1:4, matrix(5:20, nrow=4), along=2)
Abind(1:4, matrix(5:20, nrow=4), along=2)

# Like rbind
Abind(x=1:4, matrix(5:20, nrow=4), along=1)
Abind(1:4, matrix(5:20, nrow=4), along=1)

# Create a 3-d array out of two matrices
Abind(x=matrix(1:16, nrow=4), y=matrix(17:32, nrow=4), along=3)

# Use of hier.names
Abind(x=cbind(a=1:3, b=4:6), y=cbind(a=7:9, b=10:12), hier.names=TRUE)

# Use a list argument
Abind(list(x=x, y=x), along=3)
# Use lapply(..., get) to get the objects
an <- c('x', 'y')
names(an) <- an
Abind(lapply(an, get), along=3)
```

`Abstract` *Display Compact Abstract of a Data Frame*

Description

Compactly display the content and structure of a `data.frame`, including variable labels. `str()` is optimized for lists and its output is relatively technical, when it comes to e.g. attributes. `summary()` on the other side already calculates some basic statistics.

Usage

```
Abstract(
  x,
  sep = ", ",
  zero.form = ".",
  maxlevels = 5,
  trunc = TRUE,
  list.len = 999
)

## S3 method for class 'abstract'
print(x, sep = NULL, width = NULL, trunc = NULL, print.gap = 2, ...)
```

Arguments

x	a data.frame to be described
sep	the separator for concatenating the levels of a factor
zero.form	a symbol to be used, when a variable has zero NAs.
maxlevels	(integer, Inf) Max. number of factor levels to display. Default is 5. Set this to Inf, if all levels are needed.
trunc	logical, defining if level names exceeding the column with should be truncated. Default is TRUE.
list.len	numeric; maximum number of list elements to display.
width	Console width. If NULL, defaults to options("width").
print.gap	(integer) Number of spaces between columns.
...	Further arguments to print method.

Details

The levels of a factor and describing variable labels (as created by Label()) will be wrapped within the columns.

The first 4 columns are printed with the needed fix width, the last 2 (Levels and Labels) are wrapped within the column. The width is calculated depending on the width of the screen as given by getOption("width").

ToWord has an interface for the class abstract.

Value

an object of class abstract, essentially a character matrix with 5 or 6 columns containing:

1. a column number (Nr),
2. the name of the column (ColName),
3. the column class (Class),
4. the number of NAs (NAs),
5. the levels if the variable is a factor (Levels),
6. (if there are any) descriptive labels for the column (Labels).

Author(s)

Andri Signorell andri@signorell.net

See Also

`utils::str()`, `base::summary()`, `ColumnWrap()`, `Desc()`

Other Statistical summary functions: `Desc()`

Examples

```
d.mydata <- d.pizza
# let's use some labels
Label(d.mydata) <- "Lorem ipsum dolor sit amet, consetetur sadipscing elitr,
sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat,
sed diam voluptua. At vero eos et accusam."

Label(d.mydata$temperature) <- "Amet, consetetur sadipscing elitr, sed diam nonumy "

Abstract(d.mydata)
```

AddMonths *Add a Month to a Date*

Description

Clueless adding numbers of months to a date will in some cases lead to invalid dates, think of e.g. 2012-01-30 + 1 month.
AddMonths ensures that the result is always a valid date, e.g. `as.Date("2013-01-31") + 1 month` will be `"2013-02-28"`. If number `n` is negative, the months will be subtracted.

Usage

```
AddMonths(x, n, ...)
```

Arguments

`x`	a Date object (or something which can be coerced by `as.Date`(x, ...) to such an object) to which a number of months has to be added.
`n`	the number of months to be added. If n is negative the months will be subtracted.
`...`	the dots are passed to `as.Date`, e.g. for supplying `origin`.

Details

All parameters will be recyled if necessary.

Value

a vector of class Date with the same dimension as x, containing the transformed dates.

Author(s)

Andri Signorell <andri@signorell.net>, based on code by Roland Rapold and Antonio

References

Thanks to Antonio: https://stackoverflow.com/questions/14169620/add-a-month-to-a-date

See Also

as.ym; Date functions: Year, Month, etc.

Examples

```
# characters will be coerced to Date
AddMonths("2013-01-31", 1)

# negative n
AddMonths(as.Date("2013-03-31"), -1)

# Arguments will be recycled
# (with warning if the longer is not a multiple of length of shorter)
AddMonths(c("2013-01-31", "2013-03-31", "2013-10-31", "2013-12-31"), c(1,-1))


x <- as.POSIXct(c("2015-01-31", "2015-08-31"))
n <- c(1, 3)
AddMonths(x, n)

# mind the origin if x supplied as numeric ...
x <- as.numeric(as.Date(x))
AddMonths(x, n, origin=as.Date("1970-01-01"))
```

Agree *Raw Simple And Extended Percentage Agreement*

Description

Computes raw simple and extended percentage agreement among raters.

Usage

```
Agree(x, tolerance = 0, na.rm = FALSE)
```

Arguments

x	a data.frame or a $k \times m$ matrix, k subjects (in rows) m raters (in columns).
tolerance	number of successive rating categories that should be regarded as rater agreement (see details).
na.rm	logical, indicating whether NA values should be stripped before the computation proceeds. If set to TRUE only the complete cases of the ratings will be used. Defaults to FALSE.

Details

Using extended percentage agreement (`tolerance != 0`) is only possible for numerical values. If tolerance equals 1, for example, raters differing by one scale degree are interpreted as agreeing.

Value

numeric value of coefficient of interrater reliability

The number of finally (potentially after omitting missing values) used subjects and raters are returned as attributes:

subjects	the number of subjects examined.
raters	the number of raters.

Author(s)

Matthias Gamer <m.gamer@uke.uni-hamburg.de>,
some editorial amendments Andri Signorell <andri@signorell.net>

See Also

CohenKappa, KappaM

Examples

```
categ <- c("V", "N", "P")
lvls  <- factor(categ, levels=categ)
rtr1  <- rep(lvls, c(60, 30, 10))
rtr2  <- rep(rep(lvls, nlevels(lvls)), c(53,5,2, 11,14,5, 1,6,3))
rtr3  <- rep(rep(lvls, nlevels(lvls)), c(48,8,3, 15,10,7, 3,4,2))

Agree(cbind(rtr1, rtr2))       # Simple percentage Agreement
Agree(data.frame(rtr1, rtr2))  # can be a data.frame
Agree(cbind(rtr1, rtr2, rtr3)) # Simple percentage Agreement

Agree(cbind(rtr1, rtr2), 1)    # Extended percentage Agreement
```

`AllDuplicated` *Index Vector of All Values Involved in Ties*

Description

The function `duplicated` returns a logical vector indicating which elements x are duplicates, but will not include the very first appearance of subsequently duplicated elements. `AllDuplicated` returns an index vector of ALL the values in `x` which are involved in ties.
So `!AllDuplicated` can be used to determine all elements of x, which appear exactly once (thus with frequency 1).

Usage

```
AllDuplicated(x)
```

Arguments

x	vector of any type.

Value

logical vector of the same dimension as x.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`unique` returns a unique list of all values in x
`duplicated` returns an index vector flagging all elements, which appeared more than once (leaving out the first appearance!)
`union`(A, B) returns a list with the unique values from A and B
`intersect` returns all elements which appear in A and in B
`setdiff`(A, B) returns all elements appearing in A but not in B
`setequal`(A, B) returns `TRUE` if A contains exactly the same elements as B
`split`(A, A) returns a list with all the tied values in A (see examples)

Examples

```
x <- c(1:10, 4:6)

AllDuplicated(x)

# compare to:
duplicated(x)
```

```
x[!AllDuplicated(x)]

# union, intersect and friends...
A <- c(sort(sample(1:20, 9)),NA)
B <- c(sort(sample(3:23, 7)),NA)

# all elements from A and B (no duplicates)
union(A, B)
# all elements appearing in A and in B
intersect(A, B)
# elements in A, but not in B
setdiff(A, B)
# elements in B, but not in A
setdiff(B, A)
# Does A contain the same elements as B?
setequal(A, B)


# Find ties in a vector x
x <- sample(letters[1:10], 20, replace=TRUE)
ties <- split(x, x)

# count tied groups
sum(sapply(ties, length) > 1)

# length of tied groups
(x <- sapply(ties, length))[x>1]

# by means of table
tab <- table(x)
tab[tab>1]

# count elements involved in ties
sum(tab>1)
# count tied groups
sum(tab[tab>1])
```

`AllIdentical` *Test Multiple Objects for Exact Equality*

Description

The function `identical()` is the safe and reliable way to test two objects for being exactly equal. But it is restricted to the comparison of two objects. `AllIdentical()` allows the input of multiple objects and returns `TRUE` in the case that all of them are exactly equal, `FALSE` in every other case.

Usage

```
AllIdentical(...)
```

Arguments

...	any R objects

Details

The function checks the first object against all others, so if the first object is identical to the second and to the third, then also the second and the third are identical. (If A=B and A=C then is B=C)

Author(s)

Andri Signorell <andri@signorell.net>

See Also

identical()

Examples

```
A <- LETTERS[1:5]
B <- LETTERS[1:5]
C <- LETTERS[1:5]
D <- LETTERS[1:5]
E <- factor(LETTERS[1:5])

# all ok
AllIdentical(A, B, C, D)

# at least one odd man
AllIdentical(A, B, C, D, E)
```

AndersonDarlingTest *Anderson-Darling Test of Goodness-of-Fit*

Description

Performs the Anderson-Darling test of goodness-of-fit to a specified continuous univariate probability distribution.

Usage

```
AndersonDarlingTest(x, null = "punif", ..., nullname)
```

Arguments

x	numeric vector of data values.
null	a function, or a character string giving the name of a function, to compute the cumulative distribution function for the null distribution.
...	additional arguments for the cumulative distribution function.
nullname	optional character string describing the null distribution. The default is "uniform distribution".

Details

This command performs the Anderson-Darling test of goodness-of-fit to the distribution specified by the argument `null`. It is assumed that the values in `x` are independent and identically distributed random values, with some cumulative distribution function F. The null hypothesis is that F is the function specified by the argument `null`, while the alternative hypothesis is that F is some other function.

The procedures currently implemented are for the case of a SIMPLE null hypothesis, that is, where all the parameters of the distribution are known. Note that other packages such as 'normtest' support the test of a COMPOSITE null hypothesis where some or all of the parameters are unknown leading to different results concerning the test statistic and the p-value. Thus in 'normtest' you can test whether the data come from a normal distribution with some mean and variance (which will be estimated from the same data).

The discrepancies can be large if you don't have a lot of data (say less than 1000 observations).

Value

An object of class `"htest"` representing the result of the hypothesis test.

Author(s)

Original C code by George Marsaglia and John Marsaglia. R interface by Adrian Baddeley.

References

Anderson, T.W. and Darling, D.A. (1952) Asymptotic theory of certain 'goodness-of-fit' criteria based on stochastic processes. *Annals of Mathematical Statistics* **23**, 193–212.

Anderson, T.W. and Darling, D.A. (1954) A test of goodness of fit. *Journal of the American Statistical Association* **49**, 765–769.

Marsaglia, G. and Marsaglia, J. (2004) Evaluating the Anderson-Darling Distribution. *Journal of Statistical Software* **9** (2), 1–5. February 2004. https://www.jstatsoft.org/v09/i02

See Also

`shapiro.test` and all other tests for normality.

Examples

```
x <- rnorm(10, mean=2, sd=1)
AndersonDarlingTest(x, "pnorm", mean=2, sd=1)
```

Append *Append Elements to Objects*

Description

Append elements to a number of various objects as vectors, matrices, data.frames and lists. In a matrix either rows or columns can be inserted at any position. In data frames any vectors can be inserted. `values` will be recycled to the necessary length.

Usage

```
Append(x, values, after = NULL, ...)

## S3 method for class 'matrix'
Append(x, values, after = NULL, rows = FALSE, names = NULL, ...)
## S3 method for class 'data.frame'
Append(x, values, after = NULL, rows = FALSE, names = NULL, ...)
## Default S3 method:
Append(x, values, after = NULL, ...)
```

Arguments

`x`	object for the elements to be inserted
`values`	the elements to be inserted
`after`	a subscript, after which the values are to be appended. If it's missing the values will be appended after the last element (or column/row).
`rows`	logical, defining if vector should be added as row or as column. Default is column (`rows=FALSE`).
`names`	the dimension names for the inserted elements(s)
`...`	further arguments (not used here)

Details

The vector `x` will be recycled to a length of the next multiple of the number of rows (or columns) of the matrix `m` and will be inserted such that the first inserted row (column) has the index `i`. If the dimnames are given, they will be used no matter if the matrix m has already dimnames defined or not.

Value

An object containing the values in x with the elements of values appended after the specified element of x.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

rbind, cbind, append

Examples

```
Append(1:5, 0:1, after = 3)    # the same as append

# Insert columns and rows
x <- matrix(runif(25), 5)

Append(x, values=1:10, after=2, names = c("X","Y"))
Append(x, values=1:10, after=2)

Append(x, values=1:10, after=2, names = c("X","Y"))
Append(x, values=1:10, after=2)

# append to a data.frame
d.frm <- data.frame("id"   = c(1,2,3),
                    "code" = c("AAA", "BBB", "CCC"),
                    "val"  = c(111, 222, 333))
z <- c(10, 20, 30)

Append(d.frm, z, after=2, names="ZZZ")
```

AppendRowNames *Append Rownames to a Data Frame*

Description

Append rownames to a data.frame as first column.

Usage

```
AppendRowNames(x, names = "rownames", after = 0, remove_rownames = TRUE)
```

Arguments

x	a data.frame
names	the name of the new inserted column containing the rownames.
after	a subscript, after which the values are to be appended. If missing the rownames will be inserted as first column.
remove_rownames	logical defining if the existing rownames should be removed. Default is TRUE.

Value

the object x with appended rownames

Author(s)

Andri Signorell <andri@signorell.net>

See Also

Append

Examples

```
(dd <- data.frame(x=1:5, y=6:10, z=LETTERS[1:5],
                  row.names = letters[1:5]))
AppendRowNames(dd)
```

---

as.matrix.xtabs *Convert xtabs To matrix*

Description

This function converts an xtabs object to a matrix.

Usage

```
## S3 method for class 'xtabs'
as.matrix(x, ...)
```

Arguments

x	an object of class xtabs
...	additional arguments to be passed to or from methods.

Details

An xtabs object is indeed already a matrix, but won't be converted to a pure matrix by as.matrix.default function, as its class definition will remain unchanged. Some functions expecting a pure matrix may fail, when fed with a xtabs object. as.matrix.xtabs will drop the classes and the call attribute. Note that unclass would as well discard the classes xtabs and table, but retain the "call" attribute.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

as.matrix, xtabs, unclass

Examples

```
tab <- xtabs( ~ driver + operator, data=d.pizza)

str(tab)
class(tab)

str(as.matrix(tab))
class(as.matrix(tab))
```

`as.ym` *A Class for Dealing with the Yearmonth Format*

Description

The representation of year and month information in YYYYYMM format as an integer is often handy and a useful and efficient data structure. Adding a number of months to such a date is not quite catchy, however, since the date structure is to be retained. For example, 201201 - 2 [months] is expected to result in 201111 instead of 201199. AddMonthsYM does this job.

Usage

```
as.ym(x)
## S3 method for class 'ym'
as.Date(x, d = 1, ...)

## S3 method for class 'ym'
AddMonths(x, n, ...)
```

Arguments

`x`	a vector of integers, representing the dates in the format YYYYMM, to which a number of months has to be added. YYYY must lie in the range of 1000-3000, MM in 1-12.
`d`	the day to be used for converting a yearmonth to a date. Default is 1.
`n`	the number of months to be added. If n is negative the months will be subtracted.
`...`	further arguments (not used here).

Details

All parameters will be recyled if necessary. The therefore used function `mapply` will display a warning, if the longer argument is not a multiple of the length of the shorter one.

Value

a vector of class `integer` with the same dimension as x, containing the transformed dates.

Author(s)

Andri Signorell <andri@signorell.net>, originally based on code by Roland Rapold

See Also

AddMonths; Date functions, like Year, Month, etc.

Examples

```
Month(as.ym(202408))
Year(as.ym(202408))

Year(as.Date("2024-12-05"))
Year(as.ym(202412))

Month(as.Date("2024-12-05"), fmt = "mm")
Month(as.ym(202412), fmt="mm")

AddMonths(201511, 5)

AddMonths(c(201511, 201302), c(5, 15))
AddMonths(c(201511, 201302), c(5, -4))
```

AscToChar *Convert ASCII Codes to Characters and Vice Versa*

Description

AscToChar returns a character for each ASCII code (integer) supplied.
CharToAsc returns integer codes in 0:255 for each (one byte) character in all strings in x.

Usage

```
AscToChar(i)
CharToAsc(x)
```

Arguments

i	numeric (integer) vector of values in 1:255.
x	vector of strings.

Details

Only codes in 1:127 make up the ASCII encoding which should be identical for all R versions, whereas the *'upper'* half is often determined from the ISO-8859-1 (aka "ISO-Latin 1)" encoding, but may well differ, depending on the locale setting, see also Sys.setlocale.

Note that 0 is no longer allowed since, R does not allow \0 aka nul characters in a string anymore.

Value

`AscToChar` returns a vector of the same length as i. `CharToAsc` returns a list of numeric vectors of character length of each string in x.

Author(s)

unknown guy out there, help text partly taken from M. Maechler's **sfsmisc**.

See Also

`charToRaw`

Examples

```
(x <- CharToAsc("Silvia"))

# will be pasted together
AscToChar(x)

# use strsplit if the single characters are needed
strsplit(AscToChar(x), split=NULL)

# this would be an alternative, but the latter would be of class raw
DecToHex(CharToAsc("Silvia"))
charToRaw("Silvia")
```

`Asp` *Get Aspect Ratio of the Current Plot*

Description

Returns the aspect ratio of the current plot in user coordinates.

Usage

```
Asp()
```

Details

The aspect ratio of the plot is calculated as

```
w   <- par("pin")[1] / diff(par("usr")[1:2])
h   <- par("pin")[2] / diff(par("usr")[3:4])
asp <- w/h
```

Author(s)

Andri Signorell <andri@signorell.net>

Examples

```
Asp()
```

`Association measures` *Cramer's V, Pearson's Contingency Coefficient and Phi Coefficient Yule's Q and Y, Tschuprow's T*

Description

Calculate Cramer's V, Pearson's contingency coefficient and phi, Yule's Q and Y and Tschuprow's T of `x`, if `x` is a table. If both, `x` and `y` are given, then the according table will be built first.

Usage

```
Phi(x, y = NULL, ...)
ContCoef(x, y = NULL, correct = FALSE, ...)
CramerV(x, y = NULL, conf.level = NA,
        method = c("ncchisq", "ncchisqadj", "fisher", "fisheradj"),
        correct = FALSE, ...)

YuleQ(x, y = NULL, ...)
YuleY(x, y = NULL, ...)
TschuprowT(x, y = NULL, correct = FALSE, ...)
```

Arguments

`x`	can be a numeric vector, a matrix or a table.
`y`	NULL (default) or a vector with compatible dimensions to `x`. If `y` is provided, `table(x, y, ...)` is calculated.
`conf.level`	confidence level of the interval. This is only implemented for Cramer's V. If set to `NA` (which is the default) no confidence interval will be calculated. See examples for calculating bootstrap intervals.
`method`	string defining the method to calculate confidence intervals for Cramer's V. One out of `"ncchisq"` (using noncentral chisquare), `"ncchisqadj"`, `"fisher"` (using fisher z transformation), `"fisheradj"` (using fisher z transformation and bias correction). Default is `"ncchisq"`.
`correct`	logical. Applying to `ContCoef` this indicates, whether the Sakoda's adjusted Pearson's C should be returned. For `CramerV()` and `TschuprowT()` it defines, whether a bias correction should be applied or not. Default is `FALSE`.
`...`	further arguments are passed to the function `table`, allowing i.e. to set `useNA`.

Details

For x either a matrix or two vectors x and y are expected. In latter case `table(x, y, ...)` is calculated. The function handles NAs the same way the `table` function does, so tables are by default calculated with NAs omitted.

A provided matrix is interpreted as a contingency table, which seems in the case of frequency data the natural interpretation (this is e.g. also what `chisq.test` expects).

Use the function `PairApply` (pairwise apply) if the measure should be calculated pairwise for all columns. This allows matrices of association measures to be calculated the same way `cor` does. NAs are by default omitted pairwise, which corresponds to the `pairwise.complete` option of `cor`. Use `complete.cases`, if only the complete cases of a `data.frame` are to be used. (see examples)

The maximum value for Phi is $\sqrt{(min(r,c)-1)}$. The contingency coefficient goes from 0 to $\sqrt{(\frac{min(r,c)-1}{min(r,c)})}$. For the corrected contingency coefficient and for Cramer's V the range is 0 to 1. A Cramer's V in the range of [0, 0.3] is considered as weak, [0.3,0.7] as medium and > 0.7 as strong. The minimum value for all is 0 under statistical independence.

Value

a single numeric value if no confidence intervals are requested,
and otherwise a numeric vector with 3 elements for the estimate, the lower and the upper confidence interval

Author(s)

Andri Signorell <andri@signorell.net>,
Michael Smithson <michael.smithson@anu.edu.au> (confidence intervals for Cramer V)

References

Yule, G. Uday (1912) On the methods of measuring association between two attributes. *Journal of the Royal Statistical Society, LXXV*, 579-652

Tschuprow, A. A. (1939) *Principles of the Mathematical Theory of Correlation*, translated by M. Kantorowitsch. W. Hodge & Co.

Cramer, H. (1946) *Mathematical Methods of Statistics*. Princeton University Press

Agresti, Alan (1996) *Introduction to categorical data analysis*. NY: John Wiley and Sons

Sakoda, J.M. (1977) Measures of Association for Multivariate Contingency Tables, *Proceedings of the Social Statistics Section of the American Statistical Association* (Part III), 777-780.

Smithson, M.J. (2003) *Confidence Intervals, Quantitative Applications in the Social Sciences Series*, No. 140. Thousand Oaks, CA: Sage. pp. 39-41

Bergsma, W. (2013) A bias-correction for Cramer's V and Tschuprow's T *Journal of the Korean Statistical Society* 42(3) DOI: 10.1016/j.jkss.2012.10.002

See Also

`table`, `PlotCorr`, `PairApply`, `Assocs`

Examples

```
tab <- table(d.pizza$driver, d.pizza$wine_delivered)
Phi(tab)
ContCoef(tab)
CramerV(tab)
TschuprowT(tab)

# just x and y
CramerV(d.pizza$driver, d.pizza$wine_delivered)

# data.frame
PairApply(d.pizza[,c("driver","operator","area")], CramerV, symmetric = TRUE)


# useNA is passed to table
PairApply(d.pizza[,c("driver","operator","area")], CramerV,
          useNA="ifany", symmetric = TRUE)

d.frm <- d.pizza[,c("driver","operator","area")]
PairApply(d.frm[complete.cases(d.frm),], CramerV, symmetric = TRUE)


m <- as.table(matrix(c(2,4,1,7), nrow=2))
YuleQ(m)
YuleY(m)


# Bootstrap confidence intervals for Cramer's V
# http://support.sas.com/documentation/cdl/en/statugfreq/63124/PDF/default/statugfreq.pdf, p. 1821

tab <- as.table(rbind(
  c(26,26,23,18, 9),
  c( 6, 7, 9,14,23)))
d.frm <- Untable(tab)

n <- 1000
idx <- matrix(sample(nrow(d.frm), size=nrow(d.frm) * n, replace=TRUE), ncol=n, byrow=FALSE)
v <- apply(idx, 2, function(x) CramerV(d.frm[x,1], d.frm[x,2]))
quantile(v, probs=c(0.025,0.975))

# compare this to the analytical ones
CramerV(tab, conf.level=0.95)
```

Assocs *Association Measures*

Description

Collects a number of association measures for nominal and ordinal data.

Usage

```
Assocs(x, conf.level = 0.95, verbose = NULL)

## S3 method for class 'Assocs'
print(x, digits = 4, ...)
```

Arguments

x	a 2 dimensional contingency table or a matrix.
conf.level	confidence level of the interval. If set to NA no confidence interval will be calculated. Default is 0.95.
verbose	integer out of c(2, 1, 3) defining the verbosity of the reported results. 2 (default) means medium, 1 less and 3 extensive results. Applies only to tables and is ignored else.
digits	integer which determines the number of digits used in formatting the measures of association.
...	further arguments to be passed to or from methods.

Details

This function wraps the association measures phi, contingency coefficient, Cramer's V, Goodman Kruskal's Gamma, Kendall's Tau-b, Stuart's Tau-c, Somers' Delta, Pearson and Spearman correlation, Guttman's Lambda, Theil's Uncertainty Coefficient and the mutual information.

Value

numeric matrix, dimension [1:17, 1:3]
the first column contains the estimate, the second the lower confidence interval, the third the upper one.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

Phi, ContCoef, CramerV, GoodmanKruskalGamma, KendallTauB, StuartTauC, SomersDelta, SpearmanRho, Lambda, UncertCoef, MutInf

Examples

```
options(scipen=8)

# Example taken from: SAS/STAT(R) 9.2 User's Guide, Second Edition, The FREQ Procedure
# http://support.sas.com/documentation/cdl/en/statugfreq/63124/PDF/default/statugfreq.pdf
# Hair-Eye-Color pp. 1816

tob <- as.table(matrix(c(
  69, 28, 68, 51,  6,
```

```
  69, 38, 55, 37,  0,
  90, 47, 94, 94, 16
), nrow=3, byrow=TRUE,
   dimnames=list(eye=c("blue","green","brown"),
                 hair=c("fair","red","medium","dark","black")) ))
Desc(tob)
Assocs(tob)

# Example taken from: http://www.math.wpi.edu/saspdf/stat/chap28.pdf
# pp. 1349

pain <- as.table(matrix(c(
   26,  6,
   26,  7,
   23,  9,
   18, 14,
    9, 23
   ), ncol=2, byrow=TRUE))

Desc(pain)
Assocs(pain)
```

`Atkinson` *Atkinson Index - A Measure of Inequality.*

Description

The Atkinson index is an inequality measure and is useful in determining which end of the distribution contributed most to the observed inequality.

Usage

```
Atkinson(x, n = rep(1, length(x)), parameter = 0.5, na.rm = FALSE)
```

Arguments

`x`	a vector containing at least non-negative elements.
`n`	a vector of frequencies, must be same length as `x`.
`parameter`	parameter of the inequality measure (if set to `NULL` the default parameter of the respective measure is used).
`na.rm`	logical. Should missing values be removed? Defaults to FALSE.

Value

the value of the Akinson Index.

Note

This function was previously published as `ineq()` in the **ineq** package and has been integrated here without logical changes, but with some extensions for `NA`-handling and the use of weights.

Author(s)

Achim Zeileis <Achim.Zeileis@R-project.org>

References

Cowell, F. A. (2000) Measurement of Inequality in Atkinson, A. B. / Bourguignon, F. (Eds): *Handbook of Income Distribution*. Amsterdam.

Cowell, F. A. (1995) *Measuring Inequality* Harvester Wheatshef: Prentice Hall.

Marshall, Olkin (1979) *Inequalities: Theory of Majorization and Its Applications*. New York: Academic Press.

Atkinson, A. B. (1970): On the Measurment of Inequality, *Journal of Economic Theory*, Vol. 2(3), pp. 244-263.

See Also

See `Herfindahl`, `Rosenbluth` for concentration measures and `ineq()` in the package **ineq** for additional inequality measures

Examples

```
# generate vector (of incomes)
x <- c(541, 1463, 2445, 3438, 4437, 5401, 6392, 8304, 11904, 22261)

# compute Atkinson coefficient with parameter=1
Atkinson(x, parameter=1)
```

AUC *Area Under the Curve*

Description

Calculate the area under the curve with a naive algorithm and with a more elaborated spline approach. The curve must be given by vectors of xy-coordinates.

Usage

```
AUC(x, y, from = min(x, na.rm = TRUE), to = max(x, na.rm = TRUE),
    method = c("trapezoid", "step", "spline", "linear"),
    absolutearea = FALSE, subdivisions = 100, na.rm = FALSE, ...)
```

Arguments

x, y	the xy-points of the curve
method	The type of interpolation. Can be "trapezoid" (default), "step", "linear" or "spline". The value "spline" results in the area under the natural cubic spline interpolation.
from	The value from where to start calculating the area under the curve. Defaults to the smallest x value.
to	The value from where to end the calculation of the area under the curve. Defaults to the greatest x value.
absolutearea	A logical value that determines if negative areas should be added to the total area under the curve. By default the auc function subtracts areas that have negative y values. Set absolutearea=TRUE to _add_ the absolute value of the negative areas to the total area. Ignored if method is not spline.
subdivisions	an integer telling how many subdivisions should be used for integrate (for non-linear approximations). Ignored if method is not spline.
na.rm	logical, indicating whether NA values should be stripped before the computation proceeds. In this case only the complete.cases of x and y will be used. na.rm defaults to FALSE.
...	additional arguments passed on to approx (for linear approximations). In particular rule can be set to determine how values outside the range of x is handled.

Details

If method is set to "trapezoid" then the curve is formed by connecting all points by a direct line (composite trapezoid rule). If "step" is chosen then a stepwise connection of two points is used.

For linear interpolation the AUC() function computes the area under the curve using the composite trapezoid rule. For area under a spline interpolation, AUC() uses the splinefun function in combination with the integrate to calculate a numerical integral.

The AUC() function can handle unsorted time values (by sorting x), missing observations, ties for the x values (by ignoring duplicates), and integrating over part of the area or even outside the area.

Value

Numeric value of the area under the curve.

Author(s)

Andri Signorell <andri@signorell.net>, spline part by Claus Ekstrom <claus@rprimer.dk>

See Also

integrate, splinefun

Examples

```
AUC(x=c(1,3), y=c(1,1))

AUC(x=c(1,2,3), y=c(1,2,4), method="trapezoid")
AUC(x=c(1,2,3), y=c(1,2,4), method="step")

plot(x=c(1,2,2.5), y=c(1,2,4), type="l", col="blue", ylim=c(0,4))
lines(x=c(1,2,2.5), y=c(1,2,4), type="s", col="red")

x <- seq(0, pi, length.out=200)
AUC(x=x, y=sin(x))
AUC(x=x, y=sin(x), method="spline")
```

AxisBreak *Place a Break Mark on an Axis*

Description

Places a break mark on an axis on an existing plot.

Usage

```
AxisBreak(axis = 1, breakpos = NULL, pos = NA, bgcol = "white",
          breakcol = "black", style = "slash", brw = 0.02)
```

Arguments

axis	which axis to break.
breakpos	where to place the break in user units.
pos	position of the axis (see axis).
bgcol	the color of the plot background.
breakcol	the color of the "break" marker.
style	Either 'gap', 'slash' or 'zigzag'.
brw	break width relative to plot width.

Details

The 'pos' argument is not needed unless the user has specified a different position from the default for the axis to be broken.

Note

There is some controversy about the propriety of using discontinuous coordinates for plotting, and thus axis breaks. Discontinuous coordinates allow widely separated groups of values or outliers to appear without devoting too much of the plot to empty space.
The major objection seems to be that the reader will be misled by assuming continuous coordinates. The 'gap' style that clearly separates the two sections of the plot is probably best for avoiding this.

Author(s)

Jim Lemon and Ben Bolker

Examples

```
plot(3:10, main="Axis break test")

# put a break at the default axis and position
AxisBreak()
AxisBreak(2, 2.9, style="zigzag")
```

---

axTicks.POSIXct *Compute Axis Tickmark Locations (For POSIXct Axis)*

Description

Compute pretty tickmark locations, the same way as R does internally. By default, gives the at values which axis.POSIXct(side, x) would use.

Usage

```
axTicks.POSIXct(side, x, at, format, labels = TRUE, ...)

axTicks.Date(side = 1, x, ...)
```

Arguments

side	See axis.
x, at	A date-time or date object.
format	See strptime.
labels	Either a logical value specifying whether annotations are to be made at the tickmarks, or a vector of character strings to be placed at the tickpoints.
...	Further arguments to be passed from or to other methods.

Details

axTicks has no implementation for POSIXct axis. This function fills the gap.

Value

numeric vector of coordinate values at which axis tickmarks can be drawn.

Author(s)

Andri Signorell <andri@signorell.net> simply copying R-Core code

See Also

axTicks, axis.POSIXct

Examples

```
with(beaver1, {
  time <- strptime(paste(1990, day, time %/% 100, time %% 100),
                   "%Y %j %H %M")
  plot(time, temp, type = "l") # axis at 4-hour intervals.
  # now label every hour on the time axis
  plot(time, temp, type = "l", xaxt = "n")
  r <- as.POSIXct(round(range(time), "hours"))
  axis.POSIXct(1, at = seq(r[1], r[2], by = "hour"), format = "%H")
  # place the grid
  abline(v=axTicks.POSIXct(1, at = seq(r[1], r[2], by = "hour"), format = "%H"),
         col="grey", lty="dotted")
})
```

BarnardTest *Barnard's Unconditional Test*

Description

Barnard's unconditional test for superiority applied to 2×2 contingency tables using Score or Wald statistics for the difference between two binomial proportions.

Usage

```
BarnardTest(x, y = NULL, alternative = c("two.sided", "less", "greater"),
            method = c("csm", "csm approximate", "z-pooled", "z-unpooled",
                       "boschloo", "santner and snell"),
            fixed = 1, useStoredCSM = FALSE, ...)
```

Arguments

x	a numeric vector or a two-dimensional contingency table in matrix form. x and y can also both be factors.
y	a factor object; ignored if x is a matrix.
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter.
method	Indicates the method for finding the more extreme tables: must be either "Zpooled", "Z-unpooled", "Santner and Snell", "Boschloo", "CSM", or "CSM approximate". CSM tests cannot be calculated for multinomial models.
fixed	indicates which margins are fixed. 1 stands for row, 2 for columns, NA for none of both.

useStoredCSM	logical, use a stored ordering matrix for the CSM test to greatly reduce the computation time (default is FALSE).
...	the dots are passed on to the Exact::exact.test() function.

Details

There are two fundamentally different exact tests for comparing the equality of two binomial probabilities - Fisher's exact test (Fisher, 1925), and Barnard's exact test (Barnard, 1945). Fisher's exact test (Fisher, 1925) is the more popular of the two. In fact, Fisher was bitterly critical of Barnard's proposal for esoteric reasons that we will not go into here. For 2 x 2 tables, Barnard's test is more powerful than Fisher's, as Barnard noted in his 1945 paper, much to Fisher's chagrin. Anyway, perhaps due to its computational difficulty the Barnard's is not widely used. (Mehta et.al., 2003)

Unconditional exact tests can be performed for binomial or multinomial models. The binomial model assumes the row or column margins (but not both) are known in advance, while the multinomial model assumes only the total sample size is known beforehand. For the binomial model, the user needs to specify which margin is fixed (default is rows). Conditional tests (e.g., Fisher's exact test) have both row and column margins fixed, but this is a very uncommon design. (See Calhoun (2019) for more details.)

If x is a matrix, it is taken as a two-dimensional contingency table, and hence its entries should be nonnegative integers. Otherwise, both x and y must be vectors of the same length. Incomplete cases are removed, the vectors are coerced into factor objects, and the contingency table is computed from these.

For a 2x2 contingency table, such as $X = [n_1, n_2; n_3, n_4]$, the normalized difference in proportions between the two categories, given in each column, can be written with pooled variance (Score statistic) as

$$T(X) = \frac{\hat{p}_2 - \hat{p}_1}{\sqrt{\hat{p}(1-\hat{p})(\frac{1}{c_1} + \frac{1}{c_2})}},$$

where $\hat{p} = (n_1+n_3)/(n_1+n_2+n_3+n_4)$, $\hat{p}_2 = n_2/(n_2+n_4)$, $\hat{p}_1 = n_1/(n_1+n_3)$, $c_1 = n_1+n_3$ and $c_2 = n_2 + n_4$. Alternatively, with unpooled variance (Wald statistic), the difference in proportions can we written as

$$T(X) = \frac{\hat{p}_2 - \hat{p}_1}{\sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{c_1} + \frac{\hat{p}_2(1-\hat{p}_2)}{c_2}}}.$$

The probability of observing X is

$$P(X) = \frac{c_1!c_2!}{n_1!n_2!n_3!n_4!} p^{n_1+n_2}(1-p)^{n_3+n_4},$$

where p is the unknown nuisance parameter.

Barnard's test considers all tables with category sizes c_1 and c_2 for a given p. The p-value is the sum of probabilities of the tables having a score in the rejection region, e.g. having significantly large difference in proportions for a two-sided test. The p-value of the test is the maximum p-value calculated over all p between 0 and 1.

If useStoredCSM is set to TRUE a companion data package called **ExactData** must be installed from GitHub.

The author states: *"The CSM test is computationally intensive due to iteratively maximizing the p-value calculation to order the tables. The CSM ordering matrix has been stored for all possible*

sample sizes less than or equal to 100 (i.e., max(n1,n2)<=100). Thus, using the useStoredCSM = TRUE can greatly improve computation time. However, the stored ordering matrix was computed with npNumbers=100 and it is possible that the ordering matrix was not optimal for larger npNumbers. Increasing npNumbers and setting useStoredCSM = FALSE ensures the p-value is correctly calculated at the expense of significantly greater computation time. The stored ordering matrix is not used in the calculation of confidence intervals or non-inferiority tests, so CSM can still be very computationally intensive."

Value

A list with class `"htest"` containing the following components:

`p.value`	the p-value of the test.
`estimate`	an estimate of the nuisance parameter where the p-value is maximized.
`alternative`	a character string describing the alternative hypothesis.
`method`	the character string `"Barnards Unconditional 2x2-test"`.
`data.name`	a character string giving the names of the data.
`statistic.table`	The contingency tables considered in the analysis represented by `n1` and `n2`, their scores, and whether they are included in the one-sided (`1`), two-sided (`2`) tests, or not included at all (`0`)
`nuisance.matrix`	Nuisance parameters, `p`, and the corresponding p-values for both one- and two-sided tests

Author(s)

Peter Calhoun <calhoun.peter@gmail.com>, Andri Signorell <andri@signorell.net> (interface)

References

Barnard, G.A. (1945) A new test for 2x2 tables. *Nature*, 156:177.

Barnard, G.A. (1947) Significance tests for 2x2 tables. *Biometrika*, 34:123-138.

Suissa, S. and Shuster, J. J. (1985), Exact Unconditional Sample Sizes for the 2x2 Binomial Trial, *Journal of the Royal Statistical Society*, Ser. A, 148, 317-327.

Cardillo G. (2009) MyBarnard: a very compact routine for Barnard's exact test on 2x2 matrix. https://ch.mathworks.com/matlabcentral/fileexchange/25760-mybarnard

Galili T. (2010) https://www.r-statistics.com/2010/02/barnards-exact-test-a-powerful-alternative-for-fi

Lin C.Y., Yang M.C. (2009) Improved p-value tests for comparing two independent binomial proportions. *Communications in Statistics-Simulation and Computation*, 38(1):78-91.

Trujillo-Ortiz, A., R. Hernandez-Walls, A. Castro-Perez, L. Rodriguez-Cardozo N.A. Ramos-Delgado and R. Garcia-Sanchez. (2004). Barnardextest:Barnard's Exact Probability Test. A MATLAB file. [WWW document]. https://www.mathworks.com/

Mehta, C.R., Senchaudhuri, P. (2003) Conditional versus unconditional exact tests for comparing two binomials. https://www.researchgate.net/publication/242179503_Conditional_versus_Unconditional_Exact_Tests_for_Comparing_Two_Binomials

Calhoun, P. (2019) Exact: Unconditional Exact Test. R package version 2.0.
https://CRAN.R-project.org/package=Exact

See Also

fisher.test

Examples

```
tab <- as.table(matrix(c(8, 14, 1, 3), nrow=2,
                dimnames=list(treat=c("I","II"), out=c("I","II"))))
BarnardTest(tab)

# Plotting the search for the nuisance parameter for a one-sided test
bt <- BarnardTest(tab)

# Plotting the tables included in the p-value
ttab <- as.table(matrix(c(40, 14, 10, 30), nrow=2,
                 dimnames=list(treat=c("I","II"), out=c("I","II"))))


bt <- BarnardTest(ttab)
bts <- bt$statistic.table


# Mehta et. al (2003)
tab <- as.table(matrix(c(7, 12, 8, 3), nrow=2,
                       dimnames=list(treat=c("vaccine","placebo"),
                                     infection=c("yes","no"))))
BarnardTest(tab, alternative="less")
```

BartelsRankTest *Bartels Rank Test of Randomness*

Description

Performs the Bartels rank test of randomness, which tests if a sample is sampled randomly from an underlying population. Data must at least be measured on an ordinal scale.

Usage

```
BartelsRankTest(x, alternative = c("two.sided", "trend", "oscillation"),
                method = c("normal", "beta", "auto"))
```

Arguments

x	a numeric vector containing the observations
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "trend" or "oscillation".

`method` a character string specifying the method used to compute the p-value. Must be one of `normal` (default), `beta` or `auto`.

Details

The RVN test statistic is

$$RVN = \frac{\sum_{i=1}^{n-1}(R_i - R_{i+1})^2}{\sum_{i=1}^{n}(R_i - (n+1)/2)^2}$$

where $R_i = rank(X_i), i = 1, \ldots, n$. It is known that $(RVN - 2)/\sigma$ is asymptotically standard normal, where $\sigma^2 = \frac{4(n-2)(5n^2-2n-9)}{5n(n+1)(n-1)^2}$.

By using the alternative "`trend`" the null hypothesis of randomness is tested against a trend. By using the alternative "`oscillation`" the null hypothesis of randomness is tested against a systematic oscillation.

Missing values are silently removed.

Bartels test is a rank version of von Neumann's test.

Value

A list with class "htest" containing the components:

`statistic`	the value of the normalized statistic test.
`parameter`, `n`	the size of the data, after the remotion of consecutive duplicate values.
`p.value`	the p-value of the test.
`alternative`	a character string describing the alternative hypothesis.
`method`	a character string indicating the test performed.
`data.name`	a character string giving the name of the data.
`rvn`	the value of the RVN statistic (not show on screen).
`nm`	the value of the NM statistic, the numerator of RVN (not show on screen).
`mu`	the mean value of the RVN statistic (not show on screen).
`var`	the variance of the RVN statistic (not show on screen).

Author(s)

Frederico Caeiro <fac@fct.unl.pt>

References

Bartels, R. (1982) The Rank Version of von Neumann's Ratio Test for Randomness, *Journal of the American Statistical Association*, **77** (377), 40-46.

Gibbons, J.D. and Chakraborti, S. (2003) *Nonparametric Statistical Inference*, 4th ed. (pp. 97-98). URL: http://books.google.pt/books?id=dPhtioXwI9cC&lpg=PA97&ots=ZGaQCmuEUq

von Neumann, J. (1941) Distribution of the ratio of the mean square successive difference to the variance. *Annals of Mathematical Statistics* **12**, 367-395.

See Also

rank.test, RunsTest

Examples

```
## Example 5.1 in Gibbons and Chakraborti (2003), p.98.
## Annual data on total number of tourists to the United States for 1970-1982.

years <- 1970:1982
tourists <- c(12362, 12739, 13057, 13955, 14123,  15698, 17523, 18610, 19842,
      20310, 22500, 23080, 21916)
plot(years, tourists, pch=20)

BartelsRankTest(tourists, alternative="trend", method="beta")

#  Bartels Ratio Test
#
# data:  tourists
# statistic = -3.6453, n = 13, p-value = 1.21e-08
# alternative hypothesis: trend


## Example in Bartels (1982).
## Changes in stock levels for 1968-1969 to 1977-1978 (in $A million), deflated by the
## Australian gross domestic product (GDP) price index (base 1966-1967).
x <- c(528, 348, 264, -20, - 167, 575, 410, -4, 430, - 122)

BartelsRankTest(x, method="beta")
```

BarText *Place Value Labels on a Barplot*

Description

It can sometimes make sense to display data values directly on the bars in a barplot. There are a handful of obvious alternatives for placing the labels, either on top of the bars, right below the upper end, in the middle or at the bottom. Determining the required geometry - although not difficult - is cumbersome and the code is distractingly long within an analysis code. The present function offers a short way to solve the task. It can place text either in the middle of the stacked bars, on top or on the bottom of a barplot (side by side or stacked).

Usage

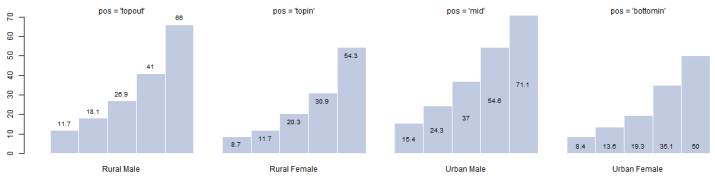
```
BarText(height, b, labels = height, beside = FALSE, horiz = FALSE,
        cex = par("cex"), adj = NULL,
        pos = c("topout", "topin", "mid", "bottomin", "bottomout"),
        offset = 0, ...)
```

Arguments

`height`	either a vector or matrix of values describing the bars which make up the plot exactly as used for creating the barplot.
`b`	the returned mid points as returned by `b <- barplot(...)`.
`labels`	the labels to be placed on the bars.
`beside`	a logical value. If `FALSE`, the columns of height are portrayed as stacked bars, and if `TRUE` the columns are portrayed as juxtaposed bars.
`horiz`	a logical value. If `FALSE`, the bars are drawn vertically with the first bar to the left. If `TRUE`, the bars are drawn horizontally with the first at the bottom.
`cex`	numeric character expansion factor; multiplied by `par("cex")` yields the final character size. `NULL` and `NA` are equivalent to `1.0`.
`adj`	one or two values in [0, 1] which specify the x (and optionally y) adjustment of the labels. On most devices values outside that interval will also work.
`pos`	one of `"topout"`, `"topin"`, `"mid"`, `"bottomin"`, `"bottomout"`, defining if the labels should be placed on top of the bars (inside or outside) or at the bottom of the bars (inside or outside).
`offset`	a vector indicating how much the bars should be shifted relative to the x axis.
`...`	the dots are passed to the `BoxedText`.

Details

The x coordinates of the labels can be found by using `barplot()` result, if they are to be centered at the top of each bar. `BarText()` calculates the rest.



Notice that when the labels are placed on top of the bars, they may be clipped. This can be avoided by setting `xpd=TRUE`.

Value

returns the geometry of the labels invisibly

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`BoxedText`

Examples

```
# simple vector
x <- c(353, 44, 56, 34)
b <- barplot(x)
BarText(x, b, x)

# more complicated
b <- barplot(VADeaths, horiz = FALSE, col=DescTools::hblue, beside = TRUE)
BarText(VADeaths, b=b, horiz = FALSE, beside = TRUE, cex=0.8)
BarText(VADeaths, b=b, horiz = FALSE, beside = TRUE, cex=0.8, pos="bottomin",
        col="white", font=2)

b <- barplot(VADeaths, horiz = TRUE, col=DescTools::hblue, beside = TRUE)
BarText(VADeaths, b=b, horiz = TRUE, beside = TRUE, cex=0.8)

b <- barplot(VADeaths)
BarText(VADeaths, b=b)

b <- barplot(VADeaths, horiz = TRUE)
BarText(VADeaths, b=b, horiz = TRUE, col="red", cex=1.5)


# position of the text
old <- par(mfrow=c(3,2), xpd=NA)
off <- c(10, 4, 1, 20, -15)

for(pos in eval(formals(BarText)$pos)) {
  b <- barplot(x, offset=off,
  main=gettextf("Textposition pos = '%s'", pos), horiz=TRUE)
  abline(h=0)
  BarText(x, b, x, offset = off, pos=pos, cex=1.5, horiz=TRUE)
}
par(old)
```

`Base Conversions` *Converts Numbers From Binmode, Octmode or Hexmode to Decimal and Vice Versa*

Description

These functions convert numbers from one base to another. There are several solutions for this problem out there, but the naming is quite heterogeneous and so consistent function names might be helpful.

Usage

```
BinToDec(x)
DecToBin(x)
OctToDec(x)
```

```
DecToOct(x)
HexToDec(x)
DecToHex(x)
```

Arguments

x	a vector of numbers, resp. alphanumerical representation of numbers (hex), to be converted.

Details

BinToDec converts numbers from binary mode into decimal values. DecToBin does it the other way round.
Oct means octal system and hex hexadecimal.

Value

A numeric or character vector of the same length as x containing the converted values. Binary, octal and decimal values are numeric, hex-values are returned as class `hexmode`.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`strtoi`

Examples

```
DecToBin(c(17, 25))
BinToDec(c(101, 11101))

DecToOct(c(17, 25))
OctToDec(c(11, 77))

DecToHex(c(17, 25))
HexToDec(c("FF", "AA", "ABC"))
```

Benford *Benford's Distribution*

Description

Density, distribution function, quantile function, and random generation for Benford's distribution.

Usage

```
dBenf(x, ndigits = 1, log = FALSE)
pBenf(q, ndigits = 1, log.p = FALSE)
qBenf(p, ndigits = 1)
rBenf(n, ndigits = 1)
```

Arguments

x, q	Vector of quantiles. See ndigits.
p	vector of probabilities.
n	number of observations. A single positive integer. Else if length(n) > 1 then the length is taken to be the number required.
ndigits	Number of leading digits, either 1 or 2. If 1 then the support of the distribution is {1,...,9}, else {10,...,99}.
log, log.p	Logical. If log.p = TRUE then all probabilities p are given as log(p).

Details

Benford's Law (aka *the significant-digit law*) is the empirical observation that in many naturally occuring tables of numerical data, the leading significant (nonzero) digit is not uniformly distributed in $\{1, 2, \ldots, 9\}$. Instead, the leading significant digit ($= D$, say) obeys the law

$$P(D = d) = \log_{10}\left(1 + \frac{1}{d}\right)$$

for $d = 1, \ldots, 9$. This means the probability the first significant digit is 1 is approximately 0.301, etc.

Benford's Law was apparently first discovered in 1881 by astronomer/mathematician S. Newcombe. It started by the observation that the pages of a book of logarithms were dirtiest at the beginning and progressively cleaner throughout. In 1938, a General Electric physicist called F. Benford re-discovered the law on this same observation. Over several years he collected data from different sources as different as atomic weights, baseball statistics, numerical data from *Reader's Digest*, and drainage areas of rivers.

Applications of Benford's Law has been as diverse as to the area of fraud detection in accounting and the design computers.

Value

dBenf gives the density, pBenf gives the distribution function, and qBenf gives the quantile function, and rBenf generates random deviates.

Author(s)

T. W. Yee

Source

These functions were previously published as dbenf() etc. in the **VGAM** package and have been integrated here without logical changes.

References

Benford, F. (1938) The Law of Anomalous Numbers. *Proceedings of the American Philosophical Society*, **78**, 551–572.

Newcomb, S. (1881) Note on the Frequency of Use of the Different Digits in Natural Numbers. *American Journal of Mathematics*, **4**, 39–40.

Examples

```
dBenf(x <- c(0:10, NA, NaN, -Inf, Inf))
pBenf(x)

## Not run:
xx <- 1:9
barplot(dBenf(xx), col = "lightblue", las = 1, xlab = "Leading digit",
        ylab = "Probability", names.arg = as.character(xx),
        main = paste("Benford's distribution",  sep = ""))

hist(rBenf(n = 1000), border = "blue", prob = TRUE,
     main = "1000 random variates from Benford's distribution",
     xlab = "Leading digit", sub="Red is the true probability",
     breaks = 0:9 + 0.5, ylim = c(0, 0.35), xlim = c(0, 10.0))
lines(xx, dBenf(xx), col = "red", type = "h")
points(xx, dBenf(xx), col = "red")

## End(Not run)
```

Between, Outside *Operators To Check, If a Value Lies Within Or Outside a Given Range*

Description

The between and outside operators are used to check, whether a vector of given values x lie within a defined range (or outside respectively). The values can be numbers, text or dates. Ordered factors are supported.

Usage

```
x %()% rng
x %(]% rng
x %[)% rng
x %[]% rng

x %][% rng
x %](% rng
x %)[% rng
x %)(% rng

x %:% rng
x %::% rng
```

Arguments

x	is a variable with at least ordinal scale, usually a numeric value, but can be an ordered factor or a text as well. Texts would be treated alphabetically.
rng	a vector of two values or a matrix with 2 columns, defining the minimum and maximum of the range for x. If rng is a matrix, x or rng will be recycled.

Details

The "BETWEEN" operators basically combine two conditional statements into one and simplify the query process.
They are merely a wrapper for: `x >= rng[1] & x <= rng[2]`, where the round bracket `(` means *strictly greater (>)* and the square bracket `[` means *greater or equal (>=)*. Numerical values of x will be handled by C-code, which is significantly faster than two comparisons in R (especially when x is huge). .
`%][%` is the negation of `%()%`, meaning all values lying outside the given range. Elements on the limits will return `TRUE`.

Both arguments, `x` and `rng`, will be recycled to the highest dimension, which is either the length of the vector (`x`) or the number of rows of the matrix (`rng`).
See also the routines used to check, whether two ranges overlap (`Overlap`, `Interval`).

`%:%` returns all the elements of a vector between the (first found) element `rng[1]` and `rng[2]`. If no match is found it returns `NA`. If `rng[2]` occurs before `rng[1]` in the vector the elements will be returned in reverse order (which is the same behaviour as the `:` operator).
`%::%` does the same in greedy mood. It uses the first match for `from` and the last match for `to`.

Value

A logical vector of the same length as x.

Author(s)

Andri Signorell <andri@signorell.net> based on C-code by Kevin Ushey <kevinushey@gmail.com>

See Also

`if`, `ifelse`, `Comparison`, `Overlap`, `Interval`

Examples

```
x <- 1:9
x %[]% c(3,5)

# outside
x <- 1:9
x %][% c(3,5)

c(x,NA) %[]% c(3,5)
```

```
x %(]% c(3,5)

# no result when from > to:
x %[]% c(5,3)
x %(]% c(5,5)

# no problem:
ordered(x) %[]% c(3,5)

# not meaningful:
factor(x) %[]% c(3,5)

# characters
letters[letters %(]% c("d","h")]

data(d.pizza)
x <- levels(d.pizza$driver)
x %[]% c("C","G")

# select diamonds with a price between 2400 and 2510
data(d.diamonds)
d.diamonds[d.diamonds$price %[]% c(2400,2510),]

# use it with an ordered factor and select all diamonds with
#   symmetry between G (included) and X (excluded).
mean(d.diamonds[d.diamonds$symmetry %[)% c("G","X"),"price"])


# use multiple ranges
2 %[]% cbind(1:4,2:5)

# both arguments are recycled
c(2,3) %[]% cbind(1:4,2:5)


# between operator for vector positions
set.seed(4)
(x <- sample(LETTERS, size=10, replace=TRUE))
# [1] "X" "K" "S" "C" "G" "L" "S" "V" "U" "Z"

# return all elements between "S" and "L"
x %:% c("S","L")
# [1] "S" "C" "G" "L"

x %:% c("S","A")
# [1] "S" "C" "G" "L" "S" "V" "U" "Z"

x %:% c("A","S")
# [1] "X" "K" "S"

# reverted matches return the elements in reverse order
x %:% c("G","X")
# [1] "G" "C" "S" "K" "X"
```

```
# no match results in NA
x %:% c("Y","B")

(x <- c("B", "A", "X", "K", "S", "K", "G", "L", "K", "V", "K", "Z"))
# lazy
x %:% c("A", "K")
# greedy
x %::% c("A", "K")
```

Bg *Background of a Plot*

Description

Paints the background of the plot, using either the figure region, the plot region or both. It can sometimes be cumbersome to elaborate the coordinates and base R does not provide a simple function for that.

Usage

```
Bg(col = "grey", region = c("plot", "figure"), border = NA)
```

Arguments

col	the color of the background, if two colors are provided, the first is used for the plot region and the second for the figure region.
region	either "plot" or "figure"
border	color for rectangle border(s). Default is NA for no borders.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

rect

Examples

```
# use two different colors for the figure region and the plot region
plot(x = rnorm(100), col="blue", cex=1.2, pch=16,
     panel.first={Bg(c("red", "lightyellow"))
                  grid()})
```

`BhapkarTest` *Bhapkar Marginal Homogeneity Test*

Description

Bhapkar (1966) tested marginal homogeneity by exploiting the asymptotic normality of marginal proportion, and so this test is also called Bhapkar's test. The idea of constructing test statistic is similar to the one of generalized McNemar's test statistic used in `StuartMaxwellTest`, and the major difference lies in the calculation of elements in variance-covariance matrix.

Usage

```
BhapkarTest(x, y = NULL)
```

Arguments

`x`	either a 2-way $k \times k$ contingency table in matrix form, or a factor.
`y`	a factor with the same levels as `x`; ignored if `x` is a matrix.

Details

Although the Bhapkar and Stuart-Maxwell tests are asymptotically equivalent (Keefe, 1982). Generally, the Bhapkar (1966) test is a more powerful alternative to the Stuart-Maxwell test. With a large N, both will produce the same Chi-square value. As the Bhapkar test is more powerful, it is preferred.

Author(s)

Andri Signorell <andri@signorell.net>

References

Bhapkar V.P. (1966) A note on the equivalence of two test criteria for hypotheses in categorical data. *Journal of the American Statistical Association*, 61: 228-235.

Ireland C.T., Ku H.H., and Kullback S. (1969) Symmetry and marginal homogeneity of an r x r contingency table. *Journal of the American Statistical Association*, 64: 1323-1341.

Keefe T.J. (1982) On the relationship between two tests for homogeneity of the marginal distributions in a two-way classification. *Biometrika*, 69: 683-684.

Sun X., Yang Z. (2008) Generalized McNemar's Test for Homogeneity of the Marginal Distributions. *SAS Global Forum 2008: Statistics and Data Analysis*, Paper 382-208.

See Also

`StuartMaxwellTest`, `mcnemar.test`, `chisq.test`, `MHChisqTest`, `BreslowDayTest`

Examples

```
# Source: http://www.john-uebersax.com/stat/mcnemar.htm#stuart
mc <- as.table(matrix(c(20,3,0,10,30,5,5,15,40), nrow=3))

BhapkarTest(mc)
```

BinomCI *Confidence Intervals for Binomial Proportions*

Description

Compute confidence intervals for binomial proportions according to a number of the most common proposed methods.

Usage

```
BinomCI(x, n, conf.level = 0.95, sides = c("two.sided", "left", "right"),
        method = c("wilson", "wald", "waldcc", "agresti-coull", "jeffreys",
                   "modified wilson", "wilsoncc","modified jeffreys",
                   "clopper-pearson", "arcsine", "logit", "witting", "pratt",
                   "midp", "lik", "blaker"),
        rand = 123, tol = 1e-05, std_est = TRUE)
```

Arguments

x	number of successes.
n	number of trials.
conf.level	confidence level, defaults to 0.95.
sides	a character string specifying the side of the confidence interval, must be one of "two.sided" (default), "left" or "right". You can specify just the initial letter. "left" would be analogue to a hypothesis of "greater" in a t.test.
method	character string specifing which method to use; this can be one out of: "wald", "wilson" (default), "wilsoncc", "agresti-coull", "jeffreys", "modified wilson", "modified jeffreys", "clopper-pearson", "arcsine", "logit", "witting", "pratt", "midp", "lik" and "blaker". Abbreviation of method is accepted. See details.
rand	seed for random number generator; see details.
tol	tolerance for method "blaker".
std_est	logical, specifying if the standard point estimator for the proportion value x/n should be returned (TRUE, default) or the method-specific internally used alternative point estimate (FALSE).

Details

All arguments are being recycled.

The **Wald** interval is obtained by inverting the acceptance region of the Wald large-sample normal test.

The **Wald with continuity correction** interval is obtained by adding the term 1/(2*n) to the Wald interval.

The **Wilson** interval, which here is the default method, was introduced by Wilson (1927) and is the inversion of the CLT approximation to the family of equal tail tests of p = p0. The Wilson interval is recommended by Agresti and Coull (1998) as well as by Brown et al (2001). It is also returned as `conf.int` from the function `prop.test` with the `correct` option set to `FALSE`.

The **Wilson cc** interval is a modification of the Wilson interval adding a continuity correction term. This is returned as `conf.int` from the function `prop.test` with the `correct` option set to `TRUE`.

The **modified Wilson** interval is a modification of the Wilson interval for x close to 0 or n as proposed by Brown et al (2001).

The **Agresti-Coull** interval was proposed by Agresti and Coull (1998) and is a slight modification of the Wilson interval. The Agresti-Coull intervals are never shorter than the Wilson intervals; cf. Brown et al (2001). The internally used point estimator p-tilde is returned as attribute.

The **Jeffreys** interval is an implementation of the equal-tailed Jeffreys prior interval as given in Brown et al (2001).

The **modified Jeffreys** interval is a modification of the Jeffreys interval for `x == 0 | x == 1` and `x == n-1 | x == n` as proposed by Brown et al (2001).

The **Clopper-Pearson** interval is based on quantiles of corresponding beta distributions. This is sometimes also called exact interval.

The **arcsine** interval is based on the variance stabilizing distribution for the binomial distribution.

The **logit** interval is obtained by inverting the Wald type interval for the log odds.

The **Witting** interval (cf. Beispiel 2.106 in Witting (1985)) uses randomization to obtain uniformly optimal lower and upper confidence bounds (cf. Satz 2.105 in Witting (1985)) for binomial proportions.

The **Pratt** interval is obtained by extremely accurate normal approximation. (Pratt 1968)

The **Mid-p** approach is used to reduce the conservatism of the Clopper-Pearson, which is known to be very pronounced. The method midp accumulates the tail areas. The lower bound p_l is found as the solution to the equation

$$\frac{1}{2} f(x; n, p_l) + (1 - F(x; m, p_l)) = \frac{\alpha}{2}$$

where $f(x; n, p)$ denotes the probability mass function (pmf) and $F(x; n, p)$ the (cumulative) distribution function of the binomial distribution with size n and proportion p evaluated at x. The upper bound p_u is found as the solution to the equation

$$\frac{1}{2} f(x; n, p_u) + F(x - 1; m, p_u) = \frac{\alpha}{2}$$

In case x=0 then the lower bound is zero and in case x=n then the upper bound is 1.

The **Likelihood-based** approach is said to be theoretically appealing. Confidence intervals are based on profiling the binomial deviance in the neighbourhood of the MLE.

For the **Blaker** method refer to Blaker (2000).

For more details we refer to Brown et al (2001) as well as Witting (1985).

Some approaches for the confidence intervals are capable of violating the [0, 1] boundaries and potentially yield negative results or values beyond 1. These would be truncated such as not to exceed the valid range of [0, 1].

So now, which interval should we use? The Wald interval often has inadequate coverage, particularly for small n and values of p close to 0 or 1. Conversely, the Clopper-Pearson Exact method is very conservative and tends to produce wider intervals than necessary. Brown et al. recommends the Wilson or Jeffreys methods for small n and Agresti-Coull, Wilson, or Jeffreys, for larger n as providing more reliable coverage than the alternatives.

For the methods `"wilson"`, `"wilsoncc"`, `"modified wilson"`, `"agresti-coull"` and `"arcsine"` the internally used alternative point estimator for the proportion value can be returned (by setting `std_est = FALSE`). The point estimate typically is slightly shifted towards 0.5 compared to the standard estimator. See the literature for the more details.

Value

A vector with 3 elements for estimate, lower confidence intervall and upper for the upper one.

For more than one argument each, a 3-column matrix is returned.

Note

The base of this function once was `binomCI()` from the **SLmisc** package. In the meantime, the code has been updated on several occasions and it has undergone numerous extensions and bug fixes.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>, Rand R. Wilcox (Pratt's method), Michael Hoehle <hoehle@math.su.se> (Mid-p), Ralph Scherer <shearer.ra76@gmail.com> (Blaker), Andri Signorell <andri@signorell.net> (interface issues and all the rest)

References

Agresti A. and Coull B.A. (1998) Approximate is better than "exact" for interval estimation of binomial proportions. *American Statistician*, **52**, pp. 119-126.

Brown L.D., Cai T.T. and Dasgupta A. (2001) Interval estimation for a binomial proportion *Statistical Science*, **16**(2), pp. 101-133.

Witting H. (1985) *Mathematische Statistik I*. Stuttgart: Teubner.

Pratt J. W. (1968) A normal approximation for binomial, F, Beta, and other common, related tail probabilities *Journal of the American Statistical Association*, 63, 1457- 1483.

Wilcox, R. R. (2005) *Introduction to robust estimation and hypothesis testing*. Elsevier Academic Press

Newcombe, R. G. (1998) Two-sided confidence intervals for the single proportion: comparison of seven methods, *Statistics in Medicine*, 17:857-872 https://pubmed.ncbi.nlm.nih.gov/16206245/

Blaker, H. (2000) Confidence curves and improved exact confidence intervals for discrete distributions, *Canadian Journal of Statistics* 28 (4), 783-798

See Also

binom.test, binconf, MultinomCI, BinomDiffCI, BinomRatioCI

Examples

```
BinomCI(x=37, n=43,
        method=eval(formals(BinomCI)$method))   # return all methods

prop.test(x=37, n=43, correct=FALSE) # same as method wilson
prop.test(x=37, n=43, correct=TRUE)  # same as method wilsoncc


# the confidence interval computed by binom.test
#   corresponds to the Clopper-Pearson interval
BinomCI(x=42, n=43, method="clopper-pearson")
binom.test(x=42, n=43)$conf.int


# all arguments are being recycled:
BinomCI(x=c(42, 35, 23, 22), n=43, method="wilson")
BinomCI(x=c(42, 35, 23, 22), n=c(50, 60, 70, 80), method="jeffreys")

# example Table I in Newcombe (1998)
meths <- c("wald", "waldcc", "wilson", "wilsoncc",
           "clopper-pearson","midp", "lik")
round(cbind(
  BinomCI(81, 263, m=meths)[, -1],
  BinomCI(15, 148, m=meths)[, -1],
  BinomCI(0, 20, m=meths)[, -1],
  BinomCI(1, 29, m=meths)[, -1]), 4)


# returning p.tilde for agresti-coull ci
BinomCI(x=81, n=263, meth="agresti-coull", std_est = c(TRUE, FALSE))
```

BinomCIn *Sample Size for a Given Width of a Binomial Confidence Interval*

Description

Returns the necessary sample size to achieve a given width of a binomial confidence interval, as calculated by BinomCI(). The function uses uniroot() to find a numeric solution.

Usage

```
BinomCIn(p = 0.5, width, interval = c(1, 100000),
         conf.level = 0.95, sides = "two.sided", method = "wilson")
```

Arguments

p	probability for success, defaults to 0.5.
width	the width of the confidence interval
interval	a vector containing the end-points of the interval to be searched for the root. The defaults are set to c(1, 100000).
conf.level	confidence level, defaults to 0.95.
sides	a character string specifying the side of the confidence interval, must be one of "two.sided" (default), "left" or "right". You can specify just the initial letter. "left" would be analogue to a hypothesis of "greater" in a t.test.
method	character string specifing which method to use; this can be one out of: "wald", "wilson", "wilsoncc", "agresti-coull", "jeffreys", "modified wilson", "modified jeffreys", "clopper-pearson", "arcsine", "logit", "witting" or "pratt". Defaults to "wilson". Abbreviation of method are accepted. See details in BinomCI().

Details

The required sample sizes for a specific width of confidence interval depends on the proportion in the population. This value might be unknown right from the start when a study is planned. In such cases the sample size needed for a given level of accuracy can be estimated using the worst case percentage which is p=50%. When a better estimate is available you can you can use it to get a smaller interval.

Value

a numeric value

Author(s)

Andri Signorell <andri@signorell.net>

See Also

BinomCI()

Examples

```
BinomCIn(p=0.1, width=0.05, method="pratt")
```

BinomDiffCI *Confidence Interval for a Difference of Binomials*

Description

Several confidence intervals for the difference between proportions are available, but they can produce markedly different results. Traditional approaches, such as the Wald interval do not perform well unless the sample size is large. Better intervals are available. These include the Agresti/Caffo method (2000), Newcombe Score method (1998) and more computing intensive ones as by Miettinen and Nurminen (1985) or Mee (1984). The latter ones are favoured by Newcombe (when forced to choose between a rock and a hard place).

Usage

```
BinomDiffCI(x1, n1, x2, n2, conf.level = 0.95, sides = c("two.sided","left","right"),
            method = c("ac", "wald", "waldcc", "score", "scorecc", "mn",
                       "mee", "blj", "ha", "hal", "jp"))
```

Arguments

x1	number of successes for the first group.
n1	number of trials for the first group.
x2	number of successes for the second group.
n2	number of trials for the second group.
conf.level	confidence level, defaults to 0.95.
sides	a character string specifying the side of the confidence interval, must be one of "two.sided" (default), "left" or "right". You can specify just the initial letter. "left" would be analogue to a hypothesis of "greater" in a t.test.
method	one of "wald", "waldcc", "ac", "score", "scorecc", "mn", "mee", "blj", "ha", "hal", "jp".

Details

All arguments are being recycled.

We estimate the difference between proportions using the sample proportions:

$$\hat{\delta} = \hat{p}_1 - \hat{p}_2 = \frac{x_1}{n_1} - \frac{x_2}{n_2}$$

The traditional **Wald** confidence interval for the difference of two proportions δ is based on the asymptotic normal distribution of $\hat{\delta}$.

The **Corrected Wald** interval uses a continuity correction included in the test statistic. The continuity correction is subtracted from the numerator of the test statistic if the numerator is greater than zero; otherwise, the continuity correction is added to the numerator. The value of the continuity correction is (1/n1 + 1/n2)/2.

The **Agresti-Caffo** (code "ac") is equal to the Wald interval with the adjustment according to Agresti, Caffo (2000) for difference in proportions and independent samples. It adds 1 to x1 and x2 and adds 2 to n1 and n2 and performs surpringly well.

Newcombe (code "scorecc") proposed a confidence interval for the difference based on the Wilson score confidence interval for a single proportion. A variant uses a continuity correction for the Wilson interval (code "scorecc").

Miettinen and Nurminen showed that the restricted maximum likelihood estimates for p1 and p2 can be obtained by solving a cubic equation and gave unique closed-form expressions for them. The Miettinen-Nurminen confidence interval is returned with code "mn".

The **Mee** (code "mee") interval proposed by Mee (1984) and Farrington-Manning (1990) is using the same maximum likelihood estimators as Miettinen-Nurminen but with another correcting factor.

The **Brown, Li's Jeffreys** (code "blj") interval was proposed by Brown, Li's Jeffreys (2005).

The **Hauck-Anderson** (code "ha") interval was proposed by Hauck-Anderson (1986).

The **Haldane** (code "hal") interval is described in Newcombe (1998) and so is the **Jeffreys-Perks** (code "jp").

Some approaches for the confidence intervals can potentially yield negative results or values beyond [-1, 1]. These would be reset such as not to exceed the range of [-1, 1].

Which of the methods to use is currently still the subject of lively discussion and has not yet been conclusively clarified. See e.g. Fagerland (2011).

The general consensus is that the most widely taught method method="wald" is inappropriate in many situations and should not be used. Recommendations seem to converge around the Miettinen-Nurminen based methods (method="mn").

Value

A matrix with 3 columns containing the estimate, the lower and the upper confidence intervall.

Author(s)

Andri Signorell <andri@signorell.net>

References

Agresti, A, Caffo, B (2000) Simple and effective confidence intervals for proportions and difference of proportions result from adding two successes and two failures. *The American Statistician* 54 (4), 280-288.

Beal, S L (1987) Asymptotic Confidence Intervals for the Difference Between Two Binomial Parameters for Use with Small Samples; *Biometrics*, 43, 941-950.

Brown L, Li X (2005) Confidence intervals for two sample binomial distribution, *Journal of Statistical Planning and Inference*, 130(1), 359-375.

Hauck WW, Anderson S. (1986) A comparison of large-sample confidence interval methods for the difference of two binomial probabilities *The American Statistician* 40(4): 318-322.

Farrington, C. P. and Manning, G. (1990) Test Statistics and Sample Size Formulae for Comparative Binomial Trials with Null Hypothesis of Non-zero Risk Difference or Non-unity Relative Risk *Statistics in Medicine*, 9, 1447-1454.

Mee RW (1984) Confidence bounds for the difference between two probabilities, *Biometrics* 40:1175-1176 .

Miettinen OS, Nurminen M. (1985) Comparative analysis of two rates. *Statistics in Medicine* 4, 213-226.

Newcombe, R G (1998). Interval Estimation for the Difference Between Independent Proportions: Comparison of Eleven Methods. *Statistics in Medicine*, 17, 873–890.

Fagerland M W, Lydersen S and Laake P (2011) Recommended confidence intervals for two independent binomial proportions, *Statistical Methods in Medical Research* 0(0) 1-31

See Also

BinomCI, MultinomCI, binom.test, prop.test, BinomRatioCI

Examples

```
x1 <- 56; n1 <- 70; x2 <- 48; n2 <- 80
xci <- BinomDiffCI(x1, n1, x2, n2, method=c("wald", "waldcc", "ac", "score",
            "scorecc", "mn", "mee", "blj", "ha"))
Format(xci[,-1], digits=4)

x1 <- 9; n1 <- 10; x2 <- 3; n2 <- 10
yci <- BinomDiffCI(x1, n1, x2, n2, method=c("wald", "waldcc", "ac", "score",
            "scorecc", "mn", "mee", "blj", "ha"))
Format(yci[, -1], digits=4)

# https://www.lexjansen.com/wuss/2016/127_Final_Paper_PDF.pdf, page 9
SetNames(round(
  BinomDiffCI(56, 70, 48, 80,
              method=c("wald", "waldcc", "hal",
                       "jp", "mee",
                       "mn", "score", "scorecc",
                       "ha", "ac", "blj"))[,-1], 4),
  rownames=c("1. Wald, no CC", "2. Wald, CC", "3. Haldane", "4. Jeffreys-Perks",
             "5. Mee", "6. Miettinen-Nurminen", "10. Score, no CC", "11. Score, CC",
             "12. Hauck-Andersen", "13. Agresti-Caffo", "16. Brown-Li"))
```

BinomRatioCI *Confidence Intervals for the Ratio of Binomial Proportions*

Description

A number of methods have been develeloped for obtaining confidence intervals for the ratio of two binomial proportions. These include the Wald/Katz-log method (Katz et al. 1978), adjusted-log (Walter 1975, Pettigrew et al. 1986), Koopman asymptotic score (Koopman 1984), Inverse hyperbolic sine transformation (Newman 2001), the Bailey method (Bailey (1987), and the Noether (1957) procedure. Koopman results are found iteratively for most intervals using root finding.

Usage

```
BinomRatioCI(x1, n1, x2, n2, conf.level = 0.95,
             sides = c("two.sided", "left", "right"),
             method = c("katz.log", "adj.log", "bailey", "koopman", "noether",
                        "sinh-1", "boot"),
             tol = .Machine$double.eps^0.25, R = 1000)
```

Arguments

x1	number of successes for the ratio numerator.
n1	number of trials for the ratio numerator.
x2	number of successes for the ratio denominator.
n2	number of successes for the ratio denominator.
conf.level	confidence level, defaults to 0.95.
sides	a character string specifying the side of the confidence interval, must be one of "two.sided" (default), "left" or "right". You can specify just the initial letter. "left" would be analogue to a hypothesis of "greater" in a t.test.
method	confidence interval method, one of "katz.log" (default), "adj.log", "bailey", "boot", "koopman", "noether" or "sinh-1". Can be abbreviated.
tol	The desired accuracy (convergence tolerance) for the iterative root finding procedure when finding Koopman intevals. The default is taken to be the smallest positive floating-point number of the workstation implementing the function, raised to the 0.25 power, and will normally be approximately 0.0001.
R	If method "boot" is chosen, the number of bootstrap iterations.

Details

All arguments are being recycled.

Let Y_1 and Y_2 be multinomial random variables with parameters n_1, π_{1i}, and n_2, π_{2i}, respectively; where $i = \{1, 2, 3, \ldots, r\}$. This encompasses the binomial case in which $r = 1$. We define the true selection ratio for the *i*th resource of *r* total resources to be:

$$\theta_i = \frac{\pi_{1i}}{\pi_{2i}}$$

where π_{1i} and π_{2i} represent the proportional use and availability of the *i*th resource, respectively. Note that if $r = 1$ the selection ratio becomes relative risk. The maximum likelihood estimators for π_{1i} and π_{2i} are the sample proportions:

$$\hat{\pi}_{1i} = \frac{y_{1i}}{n_1},$$

and

$$\hat{\pi}_{2i} = \frac{y_{2i}}{n_2}$$

where y_{1i} and y_{2i} are the observed counts for use and availability for the *i*th resource. The estimator for θ_i is:

$$\hat{\theta}_i = \frac{\hat{\pi}_{1i}}{\hat{\pi}_{2i}}.$$

Method	Algorithm
Katz-log	$\hat{\theta}_i \times \exp(\pm z_1 - \alpha/2\hat{\sigma}_W)$, where $\hat{\sigma}^2_W = \frac{(1-\hat{\pi}_{1i})}{\hat{\pi}_{1i} n_1} + \frac{(1-\hat{\pi}_{2i})}{\hat{\pi}_{2i} n_2}$.
Adjusted-log	$\hat{\theta}_{Ai} \times \exp(\pm z_1 - \alpha/2\hat{\sigma}_A)$, where $\hat{\theta}_{Ai} = \frac{y_{1i}+0.5/n_1+0.5}{y_{2i}+0.5/n_2+0.5}$, $\hat{\sigma}^2_A = \frac{1}{y_1+0.5} - \frac{1}{n_1+0.5} + \frac{1}{y_2+0.5} - \frac{1}{n_2+0.5}$.
Bailey	$\hat{\theta}_i \left[\frac{1 \pm z_1 - (\alpha/2)\left(\hat{\pi}'_{1i}/y_{1i} + \hat{\pi}'_{2i}/y_{2i} - z_1 - (\alpha/2)^2 \hat{\pi}'_{1i}\hat{\pi}'_{2i}/9y_{1i}y_{2i}\right)^{1/2}/3}{1 - z_{1-(\alpha/2)^2}\hat{\pi}'_{2i}/9y_{2i}} \right]^3$, where $\hat{\pi_{1i}}'$ = 1 - $\hat{\pi}_{1i}$, and $\hat{\pi}'_{2i}$ = 1 - $\hat{\pi}_{2i}$.
Inv. hyperbolic sine	$\ln(\hat{\theta}_i) \pm \left[2sinh^{-1}\left(\frac{z_{(1-\alpha/2)}}{2} \sqrt{\frac{1}{y_{1i}} - \frac{1}{n_1} + \frac{1}{y_{2i}} - \frac{1}{n_2}} \right) \right]$,
Koopman	Find $X^2(\theta_0) = \chi^2_1(1-\alpha)$, where $\tilde{\pi}_{1i} = \frac{\theta_0(n_1+y_{2i})+y_{1i}+n_2 - [\{\theta_0(n_1+y_{2i})+y_{1i}+n_2\}^2 - 4\theta_0(n_1+n_2)(y_{1i}+y_{2i})]^{0.5}}{2(n_1+n_2)}$, $\tilde{\pi}_{2i} = \frac{\tilde{\pi}_{1i}}{\theta_0}, and X^2(\theta_0) = \frac{(y_{1i}-n_1\tilde{\pi}_{1i})^2}{n_1\tilde{\pi}_{1i}(1-\tilde{\pi}_{1i})} \left\{ 1 + \frac{n_1(\theta_0-\tilde{\pi}_{1i})}{n_2(1-\tilde{\pi}_{1i})} \right\}$.
Noether	$\hat{\theta}_i \pm z_1 - \alpha/2\hat{\sigma}_N$, where $\hat{\sigma}^2_N = \hat{\theta}^2_i \left(\frac{1}{y_{1i}} - \frac{1}{n_1} + \frac{1}{y_{2i}} - \frac{1}{n_2} \right)$.

Exception handling strategies are generally necessary in the cases x_1 = 0, n_1 = x_1, x_2 = 0, and n_2 = x_2 (see Aho and Bowyer, in review).

The bootstrap method currently employs percentile confidence intervals.

Value

A matrix with 3 columns containing the estimate, the lower and the upper confidence intervall.

Author(s)

Ken Aho <kenaho1@gmail.com>, some tweaks Andri Signorell <andri@signorell.net>

References

Agresti, A., Min, Y. (2001) On small-sample confidence intervals for parameters in discrete distributions. *Biometrics* 57: 963-97.

Aho, K., and Bowyer, T. (In review) Confidence intervals for ratios of multinomial proportions: implications for selection ratios. *Methods in Ecology and Evolution*.

Bailey, B.J.R. (1987) Confidence limits to the risk ratio. *Biometrics* 43(1): 201-205.

Katz, D., Baptista, J., Azen, S. P., and Pike, M. C. (1978) Obtaining confidence intervals for the risk ratio in cohort studies. *Biometrics* 34: 469-474

Koopman, P. A. R. (1984) Confidence intervals for the ratio of two binomial proportions. *Biometrics* 40:513-517.

Manly, B. F., McDonald, L. L., Thomas, D. L., McDonald, T. L. and Erickson, W.P. (2002) *Resource Selection by Animals: Statistical Design and Analysis for Field Studies. 2nd edn.* Kluwer, New York, NY

Newcombe, R. G. (2001) Logit confidence intervals and the inverse sinh transformation. *The American Statistician* 55: 200-202.

Pettigrew H. M., Gart, J. J., Thomas, D. G. (1986) The bias and higher cumulants of the logarithm of a binomial variate. *Biometrika* 73(2): 425-435.

Walter, S. D. (1975) The distribution of Levins measure of attributable risk. *Biometrika* 62(2): 371-374.

See Also

BinomCI, BinomDiffCI

Examples

```
# From Koopman (1984)

BinomRatioCI(x1 = 36, n1 = 40, x2 = 16, n2 = 80, method = "katz")
BinomRatioCI(x1 = 36, n1 = 40, x2 = 16, n2 = 80, method = "koop")
```

BinTree *Binary Tree*

Description

Create a binary tree of a given number of nodes n. Can be used to organize a sorted numeric vector as a binary tree.

Usage

```
BinTree(n)

PlotBinTree(x, main="Binary tree", horiz=FALSE, cex=1.0, col=1, ...)
```

Arguments

n	integer, size of the tree
x	numeric vector to be organized as binary tree.
main	main title of the plot.

horiz	logical, should the plot be oriented horizontally or vertically. The latter is default.
cex	character extension factor for the labels.
col	color of the linesegments of the plot.
...	the dots are sent to Canvas.

Details

If we index the nodes of the tree as 1 for the top, 2–3 for the next horizontal row, 4–7 for the next, ... then the parent-child traversal becomes particularly easy. The basic idea is that the rows of the tree start at indices 1, 2, 4, ...

BinTree(13) yields the vector `c(8, 4, 9, 2, 10, 5, 11, 1, 12, 6, 13, 3, 7)` meaning that the smallest element will be in position 8 of the tree, the next smallest in position 4, etc.

Value

an integer vector of length n

Author(s)

Terry Therneau <therneau.terry@mayo.edu>
Andri Signorell <andri@signorell.net> (plot)

Examples

```
BinTree(12)

x <- sort(sample(100, 24))
z <- PlotBinTree(x, cex=0.8)


# Plot example - Titanic data, for once from a somwhat different perspective
tab <- apply(Titanic, c(2,3,4), sum)
cprob <- c(1, prop.table(apply(tab, 1, sum))
           , as.vector(aperm(prop.table(apply(tab, c(1,2), sum), 1), c(2, 1)))
           , as.vector(aperm(prop.table(tab, c(1,2)), c(3,2,1)))
)

PlotBinTree(round(cprob[BinTree(length(cprob))],2), horiz=TRUE, cex=0.8,
            main="Titanic")
text(c("sex","age","survived"), y=0, x=c(1,2,3)+1)
```

BootCI *Simple Bootstrap Confidence Intervals*

Description

Convenience wrapper for calculating bootstrap confidence intervals for univariate and bivariate statistics.

Usage

```
BootCI(x, y = NULL, FUN, ..., bci.method = c("norm", "basic", "stud", "perc", "bca"),
       conf.level = 0.95, sides = c("two.sided", "left", "right"), R = 999)
```

Arguments

x	a (non-empty) numeric vector of data values.
y	NULL (default) or a vector with compatible dimensions to x, when a bivariate statistic is used.
FUN	the function to be used
bci.method	A vector of character strings representing the type of intervals required. The value should be any subset of the values "norm", "basic", "stud", "perc", "bca", as it is passed on as method to boot.ci.
conf.level	confidence level of the interval.
sides	a character string specifying the side of the confidence interval, must be one of "two.sided" (default), "left" or "right". You can specify just the initial letter. "left" would be analogue to a hypothesis of "greater" in a t.test.
...	further arguments are passed to the function FUN.
R	The number of bootstrap replicates. Usually this will be a single positive integer. For importance resampling, some resamples may use one set of weights and others use a different set of weights. In this case R would be a vector of integers where each component gives the number of resamples from each of the rows of weights.

Value

a named numeric vector with 3 elements:

est	the specific estimate, as calculated by FUN
lwr.ci	lower bound of the confidence interval
upr.ci	upper bound of the confidence interval

Author(s)

Andri Signorell <andri@signorell.net>

See Also

MeanCI, MedianCI

Examples

```
set.seed(1984)
BootCI(d.pizza$temperature, FUN=mean, na.rm=TRUE, bci.method="basic")
BootCI(d.pizza$temperature, FUN=mean, trim=0.1, na.rm=TRUE, bci.method="basic")

BootCI(d.pizza$temperature, FUN=Skew, na.rm=TRUE, bci.method="basic")

BootCI(d.pizza$operator, d.pizza$area, FUN=CramerV)

spearman <- function(x,y) cor(x, y, method="spearman", use="p")
BootCI(d.pizza$temperature, d.pizza$delivery_min, FUN=spearman)
```

BoxCox *Box Cox Transformation*

Description

`BoxCox()` returns a transformation of the input variable using a Box-Cox transformation. `BoxCoxInv()` reverses the transformation.

Usage

```
BoxCox(x, lambda)
BoxCoxInv(x, lambda)
```

Arguments

x	a numeric vector
lambda	transformation parameter

Details

The Box-Cox transformation is given by

$$f_\lambda(x) = \begin{cases} \frac{x^\lambda - 1}{\lambda} & \text{for } \lambda \neq 0 \\ log(x) & \text{for } \lambda = 0 \end{cases}$$

Value

a numeric vector of the same length as x.

Note

These two functions are borrowed from `library(forecast)`.

Author(s)

Rob J Hyndman <rob.hyndman@monash.edu>

References

Box, G. E. P. and Cox, D. R. (1964) An analysis of transformations. *JRSS B* **26** 211–246.

See Also

Use BoxCoxLambda or boxcox in library(MASS) to find optimal lambda values.

Examples

```
# example by Greg Snow
x <- rlnorm(500, 3, 2)

par(mfrow=c(2,2))
qqnorm(x, main="Lognormal")
qqnorm(BoxCox(x, 1/2), main="BoxCox(lambda=0.5)")
qqnorm(BoxCox(x, 0), main="BoxCox(lambda=0)")

PlotFdist(BoxCox(x, 0))

bx <- BoxCox(x, lambda = BoxCoxLambda(x) )
```

BoxCoxLambda *Automatic Selection of Box Cox Transformation Parameter*

Description

An automatic selection of the Box Cox transformation parameter is estimated with two methods. Guerrero's (1993) method yields a lambda which minimizes the coefficient of variation for subseries of x. For method "loglik", the value of lambda is chosen to maximize the profile log likelihood of a linear model fitted to x. For non-seasonal data, a linear time trend is fitted while for seasonal data, a linear time trend with seasonal dummy variables is used.

Usage

```
BoxCoxLambda(x, method = c("guerrero", "loglik"), lower = -1, upper = 2)
```

Arguments

x	a numeric vector or time series
method	method to be used in calculating lambda. Can be either "guerrero" (default) or "loglik".
lower	lower limit for possible lambda values, default is -1.
upper	upper limit for possible lambda values, default is 2.

Value

a number indicating the Box-Cox transformation parameter.

Note

This function was previously published as `BoxCox.lambda()` in the **forecast** package and has been integrated here without logical changes.

Author(s)

Leanne Chhay and Rob J Hyndman

References

Box, G. E. P. and Cox, D. R. (1964) An analysis of transformations. *JRSS B* **26** 211–246.

Guerrero, V.M. (1993) Time-series analysis supported by power transformations. *Journal of Forecasting*, **12**, 37–48.

See Also

`BoxCox`

Examples

```
lambda <- BoxCoxLambda(AirPassengers, lower=0)
```

BoxedText *Add Text in a Box to a Plot*

Description

BoxedText draws the strings given in the vector labels at the coordinates given by x and y, surrounded by a rectangle.

Usage

```
BoxedText(x, ...)

## Default S3 method:
BoxedText(x, y = NULL, labels = seq_along(x), adj = NULL, pos = NULL, offset = 0.5,
          vfont = NULL, cex = 1, col = NULL, font = NULL, srt = 0,
          xpad = 0.2, ypad = 0.2, density = NULL, angle = 45, bg = NA,
          border = par("fg"), lty = par("lty"), lwd = par("lwd"), ...)
```

Arguments

x, y	numeric vectors of coordinates where the text labels should be written. If the length of x and y differs, the shorter one is recycled.
labels	a character vector or expression specifying the text to be written. An attempt is made to coerce other language objects (names and calls) to expressions, and vectors and other classed objects to character vectors by as.character. If labels is longer than x and y, the coordinates are recycled to the length of labels.
adj	The value of adj determines the way in which text strings are justified. A value of 0 produces left-justified text, 0.5 (the default) centered text and 1 right-justified text. (Any value in [0, 1] is allowed, and on most devices values outside that interval will also work.) Note that the adj argument of text also allows adj = c(x, y) for different adjustment in x- and y- directions.
pos	a position specifier for the text. If specified this overrides any adj value given. Values of 1, 2, 3 and 4, respectively indicate positions below, to the left of, above and to the right of the specified coordinates.
offset	when pos is specified, this value gives the offset of the label from the specified coordinate in fractions of a character width.
vfont	NULL for the current font family, or a character vector of length 2 for Hershey vector fonts. The first element of the vector selects a typeface and the second element selects a style. Ignored if labels is an expression.
cex	numeric character expansion factor; multiplied by par("cex") yields the final character size. NULL and NA are equivalent to 1.0.
col, font	the color and (if vfont = NULL) font to be used, possibly vectors. These default to the values of the global graphical parameters in par().
srt	The string rotation in degrees.
xpad, ypad	The proportion of the rectangles to the extent of the text within.
density	the density of shading lines, in lines per inch. The default value of NULL means that no shading lines are drawn. A zero value of density means no shading lines whereas negative values (and NA) suppress shading (and so allow color filling).
angle	angle (in degrees) of the shading lines.
bg	color(s) to fill or shade the rectangle(s) with. The default NA (or also NULL) means do not fill, i.e., draw transparent rectangles, unless density is specified.
border	color for rectangle border(s). The default is par("fg"). Use border = NA to omit borders (this is the default). If there are shading lines, border = TRUE means use the same colour for the border as for the shading lines.
lty	line type for borders and shading; defaults to "solid".
lwd	line width for borders and shading. Note that the use of lwd = 0 (as in the examples) is device-dependent.
...	additional arguments are passed to the text function.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

SpreadOut, similar function in package **plotrix** boxed.labels (lacking rotation option)

Examples

```
Canvas(xpd=TRUE)

BoxedText(0, 0, adj=0, label="This is boxed text", srt=seq(0,360,20), xpad=.3, ypad=.3)
points(0,0, pch=15)
```

BreslowDayTest *Breslow-Day Test for Homogeneity of the Odds Ratios*

Description

Calculates the Breslow-Day test of homogeneity for a $2 \times 2 \times k$ table, in order to investigate if all k strata have the same OR. If OR is not given, the Mantel-Haenszel estimate is used.

Usage

```
BreslowDayTest(x, OR = NA, correct = FALSE)
```

Arguments

x	a $2 \times 2 \times k$ table.
OR	the odds ratio to be tested against. If left undefined (default) the Mantel-Haenszel estimate will be used.
correct	If TRUE, the Breslow-Day test with Tarone's adjustment is computed, which subtracts an adjustment factor to make the resulting statistic asymptotically chi-square.

Details

For the Breslow-Day test to be valid, the sample size should be relatively large in each stratum, and at least 80% of the expected cell counts should be greater than 5. Note that this is a stricter sample size requirement than the requirement for the Cochran-Mantel-Haenszel test for tables, in that each stratum sample size (not just the overall sample size) must be relatively large. Even when the Breslow-Day test is valid, it might not be very powerful against certain alternatives, as discussed in Breslow and Day (1980).

Alternatively, it might be better to cast the entire inference problem into the setting of a logistic regression model. Here, the underlying question of the Breslow-Day test can be answered by investigating whether an interaction term with the strata variable is necessary (e.g. using a likelihood ratio test using the anova function).

Author(s)

Michael Hoehle <hoehle@math.su.se>

References

Breslow, N. E., N. E. Day (1980) The Analysis of Case-Control Studies *Statistical Methods in Cancer Research: Vol. 1*. Lyon, France, IARC Scientific Publications.

Tarone, R.E. (1985) On heterogeneity tests based on efficient scores, *Biometrika*, 72, pp. 91-95.

Jones, M. P., O'Gorman, T. W., Lemka, J. H., and Woolson, R. F. (1989) A Monte Carlo Investigation of Homogeneity Tests of the Odds Ratio Under Various Sample Size Configurations *Biometrics*, 45, 171-181

Breslow, N. E. (1996) Statistics in Epidemiology: The Case-Control Study *Journal of the American Statistical Association*, 91, 14-26.

See Also

WoolfTest

Examples

```
migraine <- xtabs(freq ~ .,
            cbind(expand.grid(treatment=c("active", "placebo"),
                              response =c("better", "same"),
                              gender   =c("female", "male")),
                  freq=c(16, 5, 11, 20, 12, 7, 16, 19))
            )

# get rid of gender
tab <- xtabs(Freq ~ treatment + response, migraine)
Desc(tab)

# only the women
female <- migraine[,, 1]
Desc(female)

# .. and the men
male <- migraine[,, 2]
Desc(male)

BreslowDayTest(migraine)
BreslowDayTest(migraine, correct = TRUE)


salary <- array(
      c(38, 12, 102, 141, 12, 9, 136, 383),
      dim=c(2, 2, 2),
      dimnames=list(exposure=c("exposed", "not"),
                    disease =c("case", "control"),
                    salary  =c("<1000", ">=1000"))
                    )

# common odds ratio = 4.028269
BreslowDayTest(salary, OR = 4.02)
```

`BreuschGodfreyTest` *Breusch-Godfrey Test*

Description

`BreuschGodfreyTest` performs the Breusch-Godfrey test for higher-order serial correlation.

Usage

```
BreuschGodfreyTest(
  formula,
  order = 1,
  order.by = NULL,
  type = c("Chisq", "F"),
  data = list(),
  fill = 0
)
```

Arguments

`formula`	a symbolic description for the model to be tested (or a fitted `"lm"` object).
`order`	integer. maximal order of serial correlation to be tested.
`order.by`	Either a vector `z` or a formula with a single explanatory variable like `~ z`. The observations in the model are ordered by the size of `z`. If set to `NULL` (the default) the observations are assumed to be ordered (e.g., a time series).
`type`	the type of test statistic to be returned. Either `"Chisq"` for the Chi-squared test statistic or `"F"` for the F test statistic.
`data`	an optional data frame containing the variables in the model. By default the variables are taken from the environment which `BreuschGodfreyTest` is called from.
`fill`	starting values for the lagged residuals in the auxiliary regression. By default `0` but can also be set to `NA`.

Details

Under H_0 the test statistic is asymptotically Chi-squared with degrees of freedom as given in `parameter`. If `type` is set to `"F"` the function returns a finite sample version of the test statistic, employing an F distribution with degrees of freedom as given in `parameter`.

By default, the starting values for the lagged residuals in the auxiliary regression are chosen to be 0 (as in Godfrey 1978) but could also be set to `NA` to omit them.

`BreuschGodfreyTest` also returns the coefficients and estimated covariance matrix from the auxiliary regression that includes the lagged residuals. Hence, `CoefTest` (package: RegClassTools) can be used to inspect the results. (Note, however, that standard theory does not always apply to the standard errors and t-statistics in this regression.)

Value

A list with class "BreuschGodfreyTest" inheriting from "htest" containing the following components:

statistic	the value of the test statistic.
p.value	the p-value of the test.
parameter	degrees of freedom.
method	a character string indicating what type of test was performed.
data.name	a character string giving the name(s) of the data.
coefficients	coefficient estimates from the auxiliary regression.
vcov	corresponding covariance matrix estimate.

Note

This function was previously published as bgtest in the **lmtest** package and has been integrated here without logical changes.

Author(s)

David Mitchell david.mitchell@dotars.gov.au, Achim Zeileis

References

Johnston, J. (1984): *Econometric Methods*, Third Edition, McGraw Hill Inc.

Godfrey, L.G. (1978): 'Testing Against General Autoregressive and Moving Average Error Models when the Regressors Include Lagged Dependent Variables', *Econometrica*, 46, 1293-1302.

Breusch, T.S. (1979): 'Testing for Autocorrelation in Dynamic Linear Models', *Australian Economic Papers*, 17, 334-355.

See Also

DurbinWatsonTest

Examples

```
## Generate a stationary and an AR(1) series
x <- rep(c(1, -1), 50)

y1 <- 1 + x + rnorm(100)

## Perform Breusch-Godfrey test for first-order serial correlation:
BreuschGodfreyTest(y1 ~ x)

## or for fourth-order serial correlation
BreuschGodfreyTest(y1 ~ x, order = 4)

## Compare with Durbin-Watson test results:
DurbinWatsonTest(y1 ~ x)
```

```
y2 <- stats::filter(y1, 0.5, method = "recursive")
BreuschGodfreyTest(y2 ~ x)
```

`BrierScore` *Brier Score for Assessing Prediction Accuracy*

Description

Calculate Brier score for assessing the quality of the probabilistic predictions of binary events.

Usage

```
BrierScore(x, pred = NULL, scaled = FALSE, ...)
```

Arguments

`x`	either a model object if pred is not supplied or the response variable if it is.
`pred`	the predicted values
`scaled`	logical, defining if scaled or not. Default is `FALSE`.
`...`	further arguments to be passed to other functions.

Details

The Brier score is a proper score function that measures the accuracy of probabilistic predictions. It is applicable to tasks in which predictions must assign probabilities to a set of mutually exclusive discrete outcomes. The set of possible outcomes can be either binary or categorical in nature, and the probabilities assigned to this set of outcomes must sum to one (where each individual probability is in the range of 0 to 1).

It's calculated as

$$\frac{1}{n} \cdot \sum_{i=1}^{n} (p_i - o_i)^2 \quad \text{where } p_i \text{ predicted probability and } o_i \text{ observed value out of (0,1)}$$

The lower the Brier score is for a set of predictions, the better the predictions are calibrated. Note that the Brier score, in its most common formulation, takes on a value between zero and one, since this is the largest possible difference between a predicted probability (which must be between zero and one) and the actual outcome (which can take on values of only 0 and 1). (In the original (1950) formulation of the Brier score, the range is double, from zero to two.)

The scaled Brier score is scaled by its maximum score under a non-informative model:

$$Brier_{scaled} = 1 - \frac{Brier}{Brier_{max}} \quad \text{where } Brier_{max} = mean(p_i) * (1 \breve{} mean(p_i))$$

to let it range between 0% and 100%. This scaled Brier score happens to be very similar to Pearson's R^2 statistic.

Value

a numeric value

Author(s)

Andri Signorell <andri@signorell.net>

References

Brier, G. W. (1950) Verification of forecasts expressed in terms of probability. *Monthly Weather Review, 78*, 1-3.

Hu B, Palta M, Shao J. (2006) Properties of R(2) statistics for logistic regression. *Stat Med.* 2006;25(8):1383–95. doi: 10.1002/sim.2300.

See Also

Conf

Examples

```
r.glm <- glm(Survived ~ ., data=Untable(Titanic), family=binomial)

BrierScore(r.glm)
```

BrierScoreCI *Confidence Intervals for the BrierScore*

Description

Calculate bootstrap intervals for the Brier score, based on a glm.

Usage

```
BrierScoreCI(
  object,
  scaled = FALSE,
  conf.level = 0.95,
  sides = c("two.sided", "left", "right"),
  ...
)
```

Arguments

object	the model object as returned by glm.
scaled	logical, defining if scaled or not. Default is FALSE.
conf.level	confidence level of the interval.
sides	a character string specifying the side of the confidence interval, must be one of "two.sided" (default), "left" or "right". "left" would be analogue to a hypothesis of "greater" in a t.test. You can specify just the initial letter.
...	further arguments are passed to the boot function. Supported arguments are type ("norm", "basic", "stud", "perc", "bca"), parallel and the number of bootstrap replicates R. If not defined those will be set to their defaults, being "basic" for type, option "boot.parallel" (and if that is not set, "no") for parallel and 999 for R.

Value

a numeric vector with 3 elements:

mean	mean
lwr.ci	lower bound of the confidence interval
upr.ci	upper bound of the confidence interval

Author(s)

Andri Signorell andri@signorell.net

See Also

BrierScore

Examples

```
utils::data(Pima.te, package = "MASS")
r.logit <- glm(type ~ ., data=Pima.te, family="binomial")

# calculate Brier score with confidence intervals
BrierScore(r.logit)
BrierScoreCI(r.logit, R=99)   # use higher R in real life!
```

BubbleLegend *Add a Legend to a Bubble Plot*

Description

Add a legend for bubbles to a bubble plot.

Usage

```
BubbleLegend(x, y = NULL, area, cols, labels = NULL, cols.lbl = "black",
             width = NULL, xjust = 0, yjust = 1, inset = 0, border = "black",
             frame = TRUE, adj = c(0.5, 0.5), cex = 1, cex.names = 1,
             bg = NULL, ...)
```

Arguments

x	the left x-coordinate to be used to position the legend. See 'Details'.
y	the top y-coordinate to be used to position the legend. See 'Details'.
area	the area(s) for the bubbles in bubble legend.
cols	the color appearing in the legend.
labels	a vector of labels to be placed at the right side of the legend.
cols.lbl	the textcolor for the labels of the bubbles.
width	the width of the legend.
xjust	how the legend is to be justified relative to the legend x location. A value of 0 means left justified, 0.5 means centered and 1 means right justified.
yjust	the same as xjust for the legend y location.
inset	inset distance(s) from the margins as a fraction of the plot region when legend is placed by keyword.
border	defines the bordor color of each rectangle. Default is none (NA).
frame	defines the bordor color of the frame around the whole legend. Default is none (NA).
adj	text alignment, horizontal and vertical.
cex	extension factor for the area, default 1.0.
cex.names	character extension for the labels, default 1.0.
bg	the background color for the bubble legend.
...	further arguments are passed to the function text.

Details

The labels are placed in the middle of the legend.

The location of the legend may be specified by setting x to a single keyword from the list "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" and "center". This places the legend on the inside of the plot frame at the given location. Partial argument matching is used. The optional inset argument specifies how far the legend is inset from the plot margins. If a single value is given, it is used for both margins; if two values are given, the first is used for x-distance, the second for y-distance. This is the same behaviour as it's implemented in legend.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

legend, FindColor, legend

Examples

```
PlotBubble(x=d.pizza$delivery_min, y=d.pizza$temperature, area=d.pizza$price,
           xlab="delivery time", ylab="temperature",
           col=SetAlpha(as.numeric(d.pizza$area)+2, .5), border="darkgrey",
           na.rm=TRUE, main="Price-Bubbles", panel.first=grid())

BubbleLegend("bottomleft", area=c(1500, 1000, 500), frame=TRUE,
             cols=SetAlpha("steelblue",0.5), bg="green",
             labels=c(1500, 1000, 500), cex=0.8,
             cols.lbl=c("yellow", "red","blue"))
```

Canvas *Canvas for Geometric Plotting*

Description

This is just a wrapper for creating an empty plot with suitable defaults for plotting geometric shapes.

Usage

```
Canvas(xlim = NULL, ylim = xlim, main = NULL, xpd = par("xpd"),
       mar=c(5.1,5.1,5.1,5.1), asp = 1, bg = par("bg"), usrbg = "white", ...)
```

Arguments

xlim, ylim	the xlims and ylims for the plot. Default is c(-1, 1).
xpd	expand drawing area, defaults to par("xpd").
main	the main title on top of the plot.

mar	set margins. Defaults to c(5.1,5.1,5.1,5.1).
asp	numeric, giving the aspect ratio y/x. (See plot.window for details. Default is 1.
bg	the background color of the plot, defaults to par("bg"), which usually will be "white".
usrbg	the color of the user space of the plot, defaults to "white".
...	additional arguments are passed to the plot() command.

Details

The plot is created with these settings:
asp = 1, xaxt = "n", yaxt = "n", xlab = "", ylab = "", frame.plot = FALSE.

Value

a list of all the previous values of the parameters changed (returned invisibly)

Author(s)

Andri Signorell <andri@signorell.net>

Examples

```
Canvas(7)
text(0, 0, "Hello world!", cex=5)
```

CartToPol *Transform Cartesian to Polar/Spherical Coordinates and Vice Versa*

Description

Transform cartesian into polar coordinates, resp. to spherical coordinates and vice versa.

Usage

```
CartToPol(x, y)
PolToCart(r, theta)

CartToSph(x, y, z, up = TRUE)
SphToCart(r, theta, phi, up = TRUE)
```

Arguments

x, y, z	vectors with the xy-coordianates to be transformed.
r	a vector with the radius of the points.
theta	a vector with the angle(s) of the points.
phi	a vector with the angle(s) of the points.
up	logical. If set to TRUE (default) theta is measured from x-y plane, else theta is measured from the z-axis.

Details

Angles are in radians, not degrees (i.e., a right angle is pi/2). Use DegToRad to convert, if you don't wanna do it by yourself.
All parameters are recycled if necessary.

Value

PolToCart returns a list of x and y coordinates of the points.
CartToPol returns a list of r for the radius and theta for the angles of the given points.

Author(s)

Andri Signorell <andri@signorell.net>, Christian W. Hoffmann <christian@echoffmann.ch>

Examples

```
CartToPol(x=1, y=1)
CartToPol(x=c(1,2,3), y=c(1,1,1))
CartToPol(x=c(1,2,3), y=1)


PolToCart(r=1, theta=pi/2)
PolToCart(r=c(1,2,3), theta=pi/2)

CartToSph(x=1, y=2, z=3)   # r=3.741657, theta=0.930274, phi=1.107149
```

CatTable *Function to write a table*

Description

CatTable helps printing a table, if is has to be broken into multiple rows. Rowlabels will be repeated after every new break.

Usage

```
CatTable(tab, wcol, nrepchars, width = getOption("width"))
```

Arguments

tab	the rows of a table to be printed, pasted together in one string with constant columnwidth.
wcol	integer, the width of the columns. All columns must have the same width.
nrepchars	integer, the number of characters to be repeated with every break. This is typically the maximum width of the rowlabels.
width	integer, the width of the whole table. Default is the width of the current command window (getOption("width")).

Author(s)

Andri Signorell <andri@signorell.net>

See Also

table, paste

Examples

```
options(scipen=8)

# used in bivariate description functions
Desc(temperature ~ cut(delivery_min, breaks=40), data=d.pizza)


txt <- c(
   paste(sample(letters, 500, replace=TRUE), collapse="")
 , paste(sample(letters, 500, replace=TRUE), collapse="")
 , paste(sample(letters, 500, replace=TRUE), collapse="")
)
txt <- paste(c("aaa","bbb","ccc"), txt, sep="")

CatTable(txt, nrepchars=3, wcol=5)
```

CCC *Concordance Correlation Coefficient*

Description

Calculates Lin's concordance correlation coefficient for agreement on a continuous measure.

Usage

```
CCC(x, y, ci = "z-transform", conf.level = 0.95, na.rm = FALSE)
```

Arguments

x	a vector, representing the first set of measurements.
y	a vector, representing the second set of measurements.
ci	a character string, indicating the method to be used. Options are z-transform or asymptotic.
conf.level	magnitude of the returned confidence interval. Must be a single number between 0 and 1.
na.rm	logical, indicating whether NA values should be stripped before the computation proceeds. If set to TRUE only the complete cases of the ratings will be used. Defaults to FALSE.

Details

Computes Lin's (1989, 2000) concordance correlation coefficient for agreement on a continuous measure obtained by two methods. The concordance correlation coefficient combines measures of both precision and accuracy to determine how far the observed data deviate from the line of perfect concordance (that is, the line at 45 degrees on a square scatter plot). Lin's coefficient increases in value as a function of the nearness of the data's reduced major axis to the line of perfect concordance (the accuracy of the data) and of the tightness of the data about its reduced major axis (the precision of the data).

Both `x` and `y` values need to be present for a measurement pair to be included in the analysis. If either or both values are missing (i.e. coded `NA`) then the measurement pair is deleted before analysis.

Value

A list containing the following:

`rho.c`	the concordance correlation coefficient.
`s.shift`	the scale shift.
`l.shift`	the location shift.
`C.b`	a bias correction factor that measures how far the best-fit line deviates from a line at 45 degrees. No deviation from the 45 degree line occurs when C.b = 1. See Lin (1989, page 258).
`blalt`	a data frame with two columns: `mean` the mean of each pair of measurements, `delta` vector `y` minus vector `x`.

Author(s)

Mark Stevenson <mark.stevenson1@unimelb.edu.au>

References

Bland J, Altman D (1986). Statistical methods for assessing agreement between two methods of clinical measurement. *The Lancet* 327: 307 - 310.

Bradley E, Blackwood L (1989). Comparing paired data: a simultaneous test for means and variances. *American Statistician* 43: 234 - 235.

Dunn G (2004). *Statistical Evaluation of Measurement Errors: Design and Analysis of Reliability Studies*. London: Arnold.

Hsu C (1940). On samples from a normal bivariate population. *Annals of Mathematical Statistics* 11: 410 - 426.

Krippendorff K (1970). Bivariate agreement coefficients for reliability of data. In: Borgatta E, Bohrnstedt G (eds) *Sociological Methodology*. San Francisco: Jossey-Bass, pp. 139 - 150.

Lin L (1989). A concordance correlation coefficient to evaluate reproducibility. *Biometrics* 45: 255 - 268.

Lin L (2000). A note on the concordance correlation coefficient. *Biometrics* 56: 324 - 325.

Pitman E (1939). A note on normal correlation. *Biometrika* 31: 9 - 12.

Reynolds M, Gregoire T (1991). Comment on Bradley and Blackwood. *American Statistician* 45: 163 - 164.

Snedecor G, Cochran W (1989). *Statistical Methods*. Ames: Iowa State University Press.

See Also

ICC, KendallW

Examples

```
## Concordance correlation plot:
set.seed(seed = 1234)
method1 <- rnorm(n = 100, mean = 0, sd = 1)
method2 <- method1 + runif(n = 100, min = 0, max = 1)

## Introduce some missing values:
method1[50] <- NA
method2[75] <- NA

tmp.ccc <- CCC(method1, method2, ci = "z-transform",
   conf.level = 0.95)

lab <- paste("CCC: ", round(tmp.ccc$rho.c[,1], digits = 2), " (95% CI ",
   round(tmp.ccc$rho.c[,2], digits = 2), " - ",
   round(tmp.ccc$rho.c[,3], digits = 2), ")", sep = "")
z <- lm(method2 ~ method1)

par(pty = "s")
plot(method1, method2, xlim = c(0, 5), ylim = c(0,5), xlab = "Method 1",
   ylab = "Method 2", pch = 16)
abline(a = 0, b = 1, lty = 2)
abline(z, lty = 1)
legend(x = "topleft", legend = c("Line of perfect concordance",
   "Reduced major axis"), lty = c(2,1), lwd = c(1,1), bty = "n")
text(x = 1.55, y = 3.8, labels = lab)

## Bland and Altman plot (Figure 2 from Bland and Altman 1986):
x <- c(494,395,516,434,476,557,413,442,650,433,417,656,267,
   478,178,423,427)

y <- c(512,430,520,428,500,600,364,380,658,445,432,626,260,
   477,259,350,451)

tmp.ccc <- CCC(x, y, ci = "z-transform", conf.level = 0.95)
tmp.mean <- mean(tmp.ccc$blalt$delta)
tmp.sd <- sqrt(var(tmp.ccc$blalt$delta))

plot(tmp.ccc$blalt$mean, tmp.ccc$blalt$delta, pch = 16,
   xlab = "Average PEFR by two meters (L/min)",
   ylab = "Difference in PEFR (L/min)", xlim = c(0,800),
   ylim = c(-140,140))
abline(h = tmp.mean, lty = 1, col = "gray")
```

```
abline(h = tmp.mean - (2 * tmp.sd), lty = 2, col = "gray")
abline(h = tmp.mean + (2 * tmp.sd), lty = 2, col = "gray")
legend(x = "topleft", legend = c("Mean difference",
   "Mean difference +/ 2SD"), lty = c(1,2), bty = "n")
legend(x = 0, y = 125, legend = c("Difference"), pch = 16,
    bty = "n")
```

`Clockwise` *Calculates Begin and End Angle From a List of Given Angles in Clockwise Mode*

Description

Transforms given angles in counter clock mode into clockwise angles.

Usage

```
Clockwise(x, start = 0)
```

Arguments

`x`	a vector of angles
`start`	the starting angle for the transformation. Defaults to 0.

Details

Sometimes there's need for angles being defined the other way round.

Value

a data.frame with two columns, containing the start and end angles.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`PlotPolar`

Examples

```
Clockwise( c(0, pi/4, pi/2, pi))
```

Closest *Find the Closest Value*

Description

Find the closest value(s) of a number in a vector x. Multiple values will be reported, if the differences are the same or if there are duplicates of the same value.

Usage

```
Closest(x, a, which = FALSE, na.rm = FALSE)
```

Arguments

x	the vector to be searched in
a	the reference value
which	a logical value defining if the index position or the value should be returned. By default will the value be returned.
na.rm	a logical value indicating whether NA values should be stripped before the computation proceeds.

Value

the value or index in x which is closest to a

Author(s)

Andri Signorell <andri@signorell.net>

See Also

which

Examples

```
# basic
set.seed(8)
x <- runif(10) * 10
Closest(x, 3.1)
sort(x)

y <- sample(10, size=10, replace=TRUE)
# multiple observations of the same closest value
Closest(y, a=6)
# get the relevant positions
Closest(y, a=6, which=TRUE)

# two different values having the same distance
```

```
Closest(c(2, 3, 4, 5), a=3.5)

# vectorize "a"
Closest(c(2, 3, 4, 5), a=c(3.1, 3.9))

# vectorize "which"
Closest(c(2, 3, 4, 5), a=3.1, which=c(FALSE, TRUE))

# vectorize both
Closest(c(2, 3, 4, 5), a=c(3.1, 3.9), which=c(FALSE, TRUE))
```

`Coalesce` *Return the First Element Not Being NA*

Description

Return the first element of a vector, not being NA.

Usage

```
Coalesce(..., method = c("is.na", "is.null", "is.finite"), flatten = TRUE)
```

Arguments

`...`	the elements to be evaluated. This can either be a single vector, several vectors of same length, a matrix, a data.frame or a list of vectors (of same length). See examples.
`method`	one out of `"is.na"` (default), `"is.null"` or `"is.finite"`. The `"is.na"` option allows `Inf` values to be in the result, the second one eliminates them.
`flatten`	logical, defines whether lists are going to be flattened (default `TRUE`).

Details

If several vectors are supplied, the evaluation will be elementwise, resp. rowwise if x is a data.frame or a matrix. The first element of the result is the first non `NA` element of the first elements of all the arguments, the second element of the result is the one of the second elements of all the arguments and so on.
Shorter inputs (of non-zero length) are NOT recycled. The function will bark, if multiple vectors do not all have the same dimension.
The idea is borrowed from SQL. Might sometimes be useful when preparing data in R instead of in SQL.

Value

return a single vector of the first non `NA` element(s) of the given data structure.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

is.na, is.finite

Examples

```
Coalesce(c(NA, NA, NA, 5, 3))
Coalesce(c(NA, NULL, "a"))
Coalesce(NULL, 5, 3)

d.frm <- data.frame(matrix(c(
  1, 2, NA, 4,
  NA, NA, 3, 1,
  NaN, 2, 3, 1,
  NA, Inf, 1, 1), nrow=4, byrow=TRUE)
)

Coalesce(d.frm)
Coalesce(as.matrix(d.frm))
Coalesce(d.frm$X1, d.frm$X2, d.frm$X3, d.frm$X4)
Coalesce(d.frm$X1, d.frm$X2, d.frm$X3, d.frm$X4, method="is.finite")
Coalesce(list(d.frm[,1], d.frm[,2]))

# returns the first finite element
Coalesce(d.frm, method="is.finite")

# with characters (take care, factors won't work!)
# is.finite does not make sense here...
d.frm <- data.frame(matrix(c(
  "a", "b", NA, "4",
  NA, NA, "g", "m",
  NA_character_,"hfdg", "rr", "m",
  NA, Inf, 1, 1), nrow=4, byrow=TRUE)
, stringsAsFactors = FALSE)

Coalesce(d.frm$X1, d.frm$X2, d.frm$X3, d.frm$X4)
Coalesce(d.frm)
Coalesce(as.list(d.frm))
```

CochranArmitageTest *Cochran-Armitage Test for Trend*

Description

Perform a Cochran Armitage test for trend in binomial proportions across the levels of a single variable. This test is appropriate only when one variable has two levels and the other variable is

ordinal. The two-level variable represents the response, and the other represents an explanatory variable with ordered levels. The null hypothesis is the hypothesis of no trend, which means that the binomial proportion is the same for all levels of the explanatory variable.

Usage

```
CochranArmitageTest(x, alternative = c("two.sided", "one.sided"))
```

Arguments

x	a frequency table or a matrix.
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "one.sided". You can specify just the initial letter.

Value

A list of class htest, containing the following components:

statistic	the z-statistic of the test.
parameter	the dimension of the table.
p.value	the p-value for the test.
alternative	a character string describing the alternative hypothesis.
method	the character string "Cochran-Armitage test for trend".
data.name	a character string giving the names of the data.

Author(s)

Andri Signorell <andri@signorell.net> strongly based on code from Eric Lecoutre <lecoutre@stat.ucl.ac.be>
https://stat.ethz.ch/pipermail/r-help/2005-July/076371.html

References

Agresti, A. (2002) *Categorical Data Analysis*. John Wiley & Sons

See Also

prop.trend.test

https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/procstat/procstat_freq_details76.htm

Examples

```
# http://www.lexjansen.com/pharmasug/2007/sp/sp05.pdf, pp. 4
dose <- matrix(c(10,9,10,7, 0,1,0,3), byrow=TRUE, nrow=2, dimnames=list(resp=0:1, dose=0:3))
Desc(dose)

CochranArmitageTest(dose)
CochranArmitageTest(dose, alternative="one.sided")
```

```
# not exactly the same as in package coin:
# independence_test(tumor ~ dose, data = lungtumor, teststat = "quad")
lungtumor <- data.frame(dose = rep(c(0, 1, 2), c(40, 50, 48)),
                        tumor = c(rep(c(0, 1), c(38, 2)),
                                  rep(c(0, 1), c(43, 7)),
                                  rep(c(0, 1), c(33, 15))))
tab <- table(lungtumor$dose, lungtumor$tumor)
CochranArmitageTest(tab)

# but similar to
prop.trend.test(tab[,1], apply(tab,1, sum))
```

`CochranQTest` *Cochran's Q test*

Description

Perform the Cochran's Q test for unreplicated randomized block design experiments with a binary response variable and paired data.

Usage

```
CochranQTest(y, ...)

## Default S3 method:
CochranQTest(y, groups, blocks, ...)

## S3 method for class 'formula'
CochranQTest(formula, data, subset, na.action, ...)
```

Arguments

`y`	either a numeric vector of data values, or a data matrix.
`groups`	a vector giving the group for the corresponding elements of y if this is a vector; ignored if y is a matrix. If not a factor object, it is coerced to one.
`blocks`	a vector giving the block for the corresponding elements of y if this is a vector; ignored if y is a matrix. If not a factor object, it is coerced to one.
`formula`	a formula of the form `y ~ groups | blocks`.
`data`	an optional matrix or data frame (or similar: see `model.frame`) containing the variables in the formula. By default the variables are taken from `environment(formula)`.
`subset`	an optional vector specifying a subset of observations to be used.
`na.action`	a function which indicates what should happen when the data contain `NA`s. Defaults to `getOption("na.action")`.
`...`	further arguments to be passed to or from methods.

Details

CochranQTest() can be used for analyzing unreplicated complete block designs (i.e., there is exactly one binary observation in y for each combination of levels of groups and blocks) where the normality assumption may be violated.

The null hypothesis is that apart from an effect of blocks, the location parameter of y is the same in each of the groups.

If y is a matrix, groups and blocks are obtained from the column and row indices, respectively. NA's are not allowed in groups or blocks; if y contains NA's, corresponding blocks are removed.

Note that Cochran's Q Test is analogue to the Friedman test with 0, 1 coded response. This is used here for a simple implementation.

Value

A list with class htest containing the following components:

statistic	the value of Cochran's chi-squared statistic.
parameter	the degrees of freedom of the approximate chi-squared distribution of the test statistic.
p.value	the p-value of the test.
method	the character string "Cochran's Q-Test".
data.name	a character string giving the names of the data.

Author(s)

Andri Signorell <andri@signorell.net>

References

Cochran, W.G. (1950) The Comparison of Percentages in Matched Samples. *Biometrika*. 37 (3/4): 256-266. doi:10.1093/biomet/37.3-4.256. JSTOR 2332378.

Examples

```
# example in:
# http://support.sas.com/documentation/cdl/en/statugfreq/63124/PDF/default/statugfreq.pdf
# pp. S. 1824

# use expand.grid, xtabs and Untable to create the dataset
d.frm <- Untable(xtabs(c(6,2,2,6,16,4,4,6) ~ .,
    expand.grid(rep(list(c("F","U")), times=3))),
    colnames = LETTERS[1:3])

# rearrange to long shape
d.long <- reshape(d.frm, varying=1:3, times=names(d.frm)[c(1:3)],
                  v.names="resp", direction="long")


# after having done the hard work of data organisation, performing the test is a piece of cake....
```

```
CochranQTest(resp ~ time | id, data=d.long)

# and let's perform a post hoc analysis using mcnemar's test
z <- split(d.long, f=d.long$time)
pairwise.table(function(i, j) {
    mcnemar.test(z[[i]]$resp, z[[j]]$resp, correct=FALSE)$p.value
  },
  level.names = names(z),
  p.adjust.method = "fdr"
)
```

CoefVar *Coefficient of Variation*

Description

Calculates the coefficient of variation and its confidence limits using various methods.

Usage

```
CoefVar(x, ...)

## S3 method for class 'lm'
CoefVar(x, unbiased = FALSE, na.rm = FALSE, ...)

## S3 method for class 'aov'
CoefVar(x, unbiased = FALSE, na.rm = FALSE, ...)

## Default S3 method:
CoefVar(x, weights = NULL, unbiased = FALSE,
       na.rm = FALSE, ...)

CoefVarCI(K, n, conf.level = 0.95,
          sides = c("two.sided", "left", "right"),
          method = c("nct","vangel","mckay","verrill","naive"))
```

Arguments

x	a (non-empty) numeric vector of data values.
weights	a numerical vector of weights the same length as x giving the weights to use for elements of x.
unbiased	logical value determining, if a bias correction should be used (see. details). Default is FALSE.
K	the coefficient of variation as calculated by CoefVar().
n	the number of observations used for calculating the coefficient of variation.

conf.level	confidence level of the interval. Defaults to 0.95.
sides	a character string specifying the side of the confidence interval, must be one of "two.sided" (default), "left" or "right". You can specify just the initial letter. "left" would be analogue to a hypothesis of "greater" in a t.test.
method	character string specifing the method to use for calculating the confidence intervals, can be one out of: "nct" (default), "vangel", "mckay", "verrill" (currently not yet implemented) and "naive". Abbreviation of method is accepted. See details.
na.rm	logical. Should missing values be removed? Defaults to FALSE.
...	further arguments (not used here).

Details

In order for the coefficient of variation to be an unbiased estimate of the true population value, the coefficient of variation is corrected as:

$$CV_{korr} = CV \cdot \left(1 - \frac{1}{4 \cdot (n-1)} + \frac{1}{n} \cdot CV^2 + \frac{1}{2 \cdot (n-1)^2}\right)$$

For determining **the confidence intervals** for the coefficient of variation a number of methods have been proposed. CoefVarCI() currently supports five different methods. The details for the methods are given in the specific references.

The **"naive" method** is based on dividing the standard confidence limit for the standard deviation by the sample mean.

McKay's approximation is asymptotically exact as n goes to infinity. McKay recommends this approximation only if the coefficient of variation is less than 0.33. Note that if the coefficient of variation is greater than 0.33, either the normality of the data is suspect or the probability of negative values in the data is non-neglible. In this case, McKay's approximation may not be valid. Also, it is generally recommended that the sample size should be at least 10 before using McKay's approximation.

Vangel's modified McKay method is more accurate than the McKay in most cases, particilarly for small samples.. According to Vangel, the unmodified McKay is only more accurate when both the coefficient of variation and alpha are large. However, if the coefficient of variation is large, then this implies either that the data contains negative values or the data does not follow a normal distribution. In this case, neither the McKay or the modified McKay should be used. In general, the Vangel's modified McKay method is recommended over the McKay method. It generally provides good approximations as long as the data is approximately normal and the coefficient of variation is less than 0.33. This is the default method.

See also: https://www.itl.nist.gov/div898/software/dataplot/refman1/auxillar/coefvacl.htm

nct uses the noncentral t-distribution to calculate the confidence intervals. See Smithson (2003).

Value

if no confidence intervals are requested: the estimate as numeric value (without any name)

else a named numeric vector with 3 elements

est	estimate
lwr.ci	lower confidence interval
upr.ci	upper confidence interval

Author(s)

Andri Signorell <andri@signorell.net>,
Michael Smithson <michael.smithson@anu.edu.au> (noncentral-t)

References

McKay, A. T. (1932). Distribution of the coefficient of variation and the extended *t* distribution, *Journal of the Royal Statistical Society*, *95*, 695–698.

Johnson, B. L., Welch, B. L. (1940). Applications of the non-central t-distribution. *Biometrika*, 31, 362–389.

Mark Vangel (1996) Confidence Intervals for a Normal Coefficient of Variation, *American Statistician*, Vol. 15, No. 1, pp. 21-26.

Kelley, K. (2007). Sample size planning for the coefcient of variation from the accuracy in parameter estimation approach. *Behavior Research Methods, 39* (4), 755-766

Kelley, K. (2007). Constructing confidence intervals for standardized effect sizes: Theory, application, and implementation. *Journal of Statistical Software, 20* (8), 1-24

Smithson, M.J. (2003) *Confidence Intervals, Quantitative Applications in the Social Sciences Series*, No. 140. Thousand Oaks, CA: Sage. pp. 39-41

Steve Verrill (2003) Confidence Bounds for Normal and Lognormal Distribution Coefficients of Variation, *Research Paper 609*, USDA Forest Products Laboratory, Madison, Wisconsin.

Verrill, S. and Johnson, R.A. (2007) Confidence Bounds and Hypothesis Tests for Normal Distribution Coefficients of Variation, *Communications in Statistics Theory and Methods*, Volume 36, No. 12, pp 2187-2206.

See Also

Mean, SD, (both supporting weights)

Examples

```
set.seed(15)
x <- runif(100)
CoefVar(x, conf.level=0.95)

#       est    low.ci    upr.ci
# 0.5092566 0.4351644 0.6151409

# Coefficient of variation for a linear model
r.lm <- lm(Fertility ~ ., swiss)
CoefVar(r.lm)

# the function is vectorized, so arguments are recyled...
# https://www.itl.nist.gov/div898/software/dataplot/refman1/auxillar/coefvacl.htm
```

```
CoefVarCI(K = 0.00246, n = 195, method="vangel",
          sides="two.sided", conf.level = c(.5,.8,.9,.95,.99,.999))
```

CohenD *Cohen's Effect Size*

Description

Computes the Cohen's d and Hedges'g effect size statistics.

Usage

```
CohenD(x, y = NULL, pooled = TRUE, correct = FALSE, conf.level = NA, na.rm = FALSE)
```

Arguments

x	a (non-empty) numeric vector of data values.
y	a (non-empty) numeric vector of data values.
pooled	logical, indicating whether compute pooled standard deviation or the whole sample standard deviation. Default is TRUE.
correct	logical, indicating whether to apply the Hedges correction. (Default: FALSE)
conf.level	confidence level of the interval. Set this to NA, if no confidence intervals should be calculated. (This is the default)
na.rm	logical. Should missing values be removed? Defaults to FALSE.

Value

a numeric vector with 3 elements:

d	the effect size d
lwr.ci	lower bound of the confidence interval
upr.ci	upper bound of the confidence interval

Author(s)

Andri Signorell <andri@signorell.net>, William Revelle <revelle@northwestern.edu> (CI)

References

Cohen, J. (1988) *Statistical power analysis for the behavioral sciences (2nd ed.)* Academic Press, New York.

Hedges, L. V. & Olkin, I. (1985) *Statistical methods for meta-analysis* Academic Press, Orlando, FL

Smithson, M.J. (2003) *Confidence Intervals, Quantitative Applications in the Social Sciences Series*, No. 140. Thousand Oaks, CA: Sage. pp. 39-41

See Also

mean, var

Examples

```
x <- d.pizza$price[d.pizza$driver=="Carter"]
y <- d.pizza$price[d.pizza$driver=="Miller"]

CohenD(x, y, conf.level=0.95, na.rm=TRUE)
```

CohenKappa *Cohen's Kappa and Weighted Kappa*

Description

Computes the agreement rates Cohen's kappa and weighted kappa and their confidence intervals.

Usage

```
CohenKappa(x, y = NULL, weights = c("Unweighted", "Equal-Spacing", "Fleiss-Cohen"),
           conf.level = NA, ...)
```

Arguments

x	can either be a numeric vector or a confusion matrix. In the latter case x must be a square matrix.
y	NULL (default) or a vector with compatible dimensions to x. If y is provided, table(x, y, ...) is calculated. In order to get a square matrix, x and y are coerced to factors with synchronized levels. (Note, that the vector interface can not be used together with weights.)
weights	either one out of "Unweighted" (default), "Equal-Spacing", "Fleiss-Cohen", which will calculate the weights accordingly, or a user-specified matrix having the same dimensions as x containing the weights for each cell.
conf.level	confidence level of the interval. If set to NA (which is the default) no confidence intervals will be calculated.
...	further arguments are passed to the function table, allowing i.e. to set useNA. This refers only to the vector interface.

Details

Cohen's kappa is the diagonal sum of the (possibly weighted) relative frequencies, corrected for expected values and standardized by its maximum value.
The equal-spacing weights (see Cicchetti and Allison 1971) are defined by

$$1 - \frac{|i-j|}{r-1}$$

r being the number of columns/rows, and the Fleiss-Cohen weights by

$$1 - \frac{(i-j)^2}{(r-1)^2}$$

The latter attaches greater importance to closer disagreements.

Data can be passed to the function either as matrix or data.frame in x, or as two numeric vectors x and y. In the latter case table(x, y, ...) is calculated. Thus NAs are handled the same way as table does. Note that tables are by default calculated **without** NAs. The specific argument useNA can be passed via the ... argument.
The vector interface (x, y) is only supported for the calculation of unweighted kappa. This is because we cannot ensure a safe construction of a confusion table for two factors with different levels, which is independent of the order of the levels in x and y. So weights might lead to inconsistent results. The function will raise an error in this case.

Value

if no confidence intervals are requested: the estimate as numeric value

else a named numeric vector with 3 elements

kappa	estimate
lwr.ci	lower confidence interval
upr.ci	upper confidence interval

Author(s)

David Meyer <david.meyer@r-project.org>, some changes and tweaks Andri Signorell <andri@signorell.net>

References

Cohen, J. (1960) A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20, 37-46.

Everitt, B.S. (1968), Moments of statistics kappa and weighted kappa. *The British Journal of Mathematical and Statistical Psychology*, 21, 97-103.

Fleiss, J.L., Cohen, J., and Everitt, B.S. (1969), Large sample standard errors of kappa and weighted kappa. *Psychological Bulletin*, 72, 332-327.

Cicchetti, D.V., Allison, T. (1971) A New Procedure for Assessing Reliability of Scoring EEG Sleep Recordings *American Journal of EEG Technology*, 11, 101-109.

See Also

CronbachAlpha, KappaM, KrippAlpha

Examples

```
# from Bortz et. al (1990) Verteilungsfreie Methoden in der Biostatistik, Springer, pp. 459
m <- matrix(c(53,  5, 2,
              11, 14, 5,
               1,  6, 3), nrow=3, byrow=TRUE,
            dimnames = list(rater1 = c("V","N","P"), rater2 = c("V","N","P")) )

# confusion matrix interface
CohenKappa(m, weight="Unweighted")

# vector interface
x <- Untable(m)
CohenKappa(x$rater1, x$rater2, weight="Unweighted")

# pairwise Kappa
rating <- data.frame(
  rtr1 = c(4,2,2,5,2, 1,3,1,1,5, 1,1,2,1,2, 3,1,1,2,1, 5,2,2,1,1, 2,1,2,1,5),
  rtr2 = c(4,2,3,5,2, 1,3,1,1,5, 4,2,2,4,2, 3,1,1,2,3, 5,4,2,1,4, 2,1,2,3,5),
  rtr3 = c(4,2,3,5,2, 3,3,3,4,5, 4,4,2,4,4, 3,1,1,4,3, 5,4,4,4,4, 2,1,4,3,5),
  rtr4 = c(4,5,3,5,4, 3,3,3,4,5, 4,4,3,4,4, 3,4,1,4,5, 5,4,5,4,4, 2,1,4,3,5),
  rtr5 = c(4,5,3,5,4, 3,5,3,4,5, 4,4,3,4,4, 3,5,1,4,5, 5,4,5,4,4, 2,5,4,3,5),
  rtr6 = c(4,5,5,5,4, 3,5,4,4,5, 4,4,3,4,5, 5,5,2,4,5, 5,4,5,4,5, 4,5,4,3,5)
)

PairApply(rating, FUN=CohenKappa, symmetric=TRUE)

# Weighted Kappa
cats <- c("<10%", "11-20%", "21-30%", "31-40%", "41-50%", ">50%")
m <- matrix(c(5,8,1,2,4,2, 3,5,3,5,5,0, 1,2,6,11,2,1,
              0,1,5,4,3,3, 0,0,1,2,5,2, 0,0,1,2,1,4), nrow=6, byrow=TRUE,
            dimnames = list(rater1 = cats, rater2 = cats) )
CohenKappa(m, weight="Equal-Spacing")


# supply an explicit weight matrix
ncol(m)
(wm <- outer(1:ncol(m), 1:ncol(m), function(x, y) {
        1 - ((abs(x-y)) / (ncol(m)-1)) } ))
CohenKappa(m, weight=wm, conf.level=0.95)


# however, Fleiss, Cohen and Everitt weight similarities
fleiss <- matrix(c(
  106, 10,  4,
  22,  28, 10,
   2,  12,  6
  ), ncol=3, byrow=TRUE)

#Fleiss weights the similarities
weights <- matrix(c(
 1.0000, 0.0000, 0.4444,
 0.0000, 1.0000, 0.6666,
```

```
 0.4444, 0.6666, 1.0000
 ), ncol=3)

CohenKappa(fleiss, weights)
```

`CollapseTable` *Collapse Levels of a Table*

Description

Collapse (or re-label) variables in a a contingency table or `ftable` object by re-assigning levels of the table variables.

Usage

```
CollapseTable(x, ...)
```

Arguments

`x`	A `table` or `ftable` object
`...`	A collection of one or more assignments of factors of the table to a list of levels

Details

Each of the `...` arguments must be of the form `variable = levels`, where `variable` is the name of one of the table dimensions, and `levels` is a character or numeric vector of length equal to the corresponding dimension of the table. Missing argument names are allowed and will be interpreted in the order of the dimensions of the table.

Value

A `table` object (even if the input was an ftable), representing the original table with one or more of its factors collapsed or rearranged into other levels.

Author(s)

Michael Friendly <friendly@yorku.ca>, Andri Signorell <andri@signorell.net>

See Also

`Untable`

`margin.table` "collapses" a table in a different way, by summing over table dimensions.

Examples

```
# create some sample data in table form
sex <- c("Male", "Female")
age <- letters[1:6]
education <- c("low", 'med', 'high')
data <- expand.grid(sex=sex, age=age, education=education)
counts <- rpois(36, 100)
data <- cbind(data, counts)
t1 <- xtabs(counts ~ sex + age + education, data=data)

Desc(t1)

##                  age   a   b   c   d   e   f
## sex    education
## Male   low          119 101 109  85  99  93
##        med           94  98 103 108  84  84
##        high          81  88  96 110 100  92
## Female low          107 104  95  86 103  96
##        med          104  98  94  95 110 106
##        high          93  85  90 109  99  86


# collapse age to 3 levels
t2 <- CollapseTable(t1, age=c("A", "A", "B", "B", "C", "C"))
Desc(t2)

##                  age   A   B   C
## sex    education
## Male   low          220 194 192
##        med          192 211 168
##        high         169 206 192
## Female low          211 181 199
##        med          202 189 216
##        high         178 199 185


# collapse age to 3 levels and pool education: "low" and "med" to "low"
t3 <- CollapseTable(t1, age=c("A", "A", "B", "B", "C", "C"),
    education=c("low", "low", "high"))
Desc(t3)

##                  age   A   B   C
## sex    education
## Male   low          412 405 360
##        high         169 206 192
## Female low          413 370 415
##        high         178 199 185



# change labels for levels of education to 1:3
t4 <- CollapseTable(t1,  education=1:3)
```

```
Desc(t4)
```

```
##                  age   a   b   c   d   e   f
## sex    education
## Male   1             119 101 109  85  99  93
##        2              94  98 103 108  84  84
##        3              81  88  96 110 100  92
## Female 1             107 104  95  86 103  96
##        2             104  98  94  95 110 106
##        3              93  85  90 109  99  86
```

ColorLegend *Add a ColorLegend to a Plot*

Description

Add a color legend, an image of a sequence of colors, to a plot.

Usage

```
ColorLegend(x, y = NULL, cols = rev(heat.colors(100)), labels = NULL,
            width = NULL, height = NULL, horiz = FALSE, xjust = 0, yjust = 1,
            inset = 0, border = NA, frame = NA, cntrlbl = FALSE,
            adj = ifelse(horiz, c(0.5, 1), c(1, 0.5)), cex = 1,
            title = NULL, title.adj = 0.5, ...)
```

Arguments

x	the left x-coordinate to be used to position the colorlegend. See 'Details'.
y	the top y-coordinate to be used to position the colorlegend. See 'Details'.
cols	the color appearing in the colorlegend.
labels	a vector of labels to be placed at the right side of the colorlegend.
width	the width of the colorlegend.
height	the height of the colorlegend.
horiz	logical indicating if the colorlegend should be horizontal; default FALSE means vertical alignment.
xjust	how the colorlegend is to be justified relative to the colorlegend x location. A value of 0 means left justified, 0.5 means centered and 1 means right justified.
yjust	the same as xjust for the legend y location.
inset	inset distance(s) from the margins as a fraction of the plot region when colorlegend is placed by keyword.
border	defines the bordor color of each rectangle. Default is none (NA).
frame	defines the bordor color of the frame around the whole colorlegend. Default is none (NA).

cntrlbl	defines, whether the labels should be printed in the middle of the color blocks or start at the edges of the colorlegend. Default is FALSE, which will print the extreme labels centered on the edges.
adj	text alignment, horizontal and vertical.
cex	character extension for the labels, default 1.0.
title	a character string or length-one expression giving a title to be placed at the top of the legend.
title.adj	horizontal adjustment for title: see the help for par("adj").
...	further arguments are passed to the function text.

Details

The labels are placed at the right side of the colorlegend and are reparted uniformly between y and y - height.

The location may also be specified by setting x to a single keyword from the list "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" and "center". This places the colorlegend on the inside of the plot frame at the given location. Partial argument matching is used. The optional inset argument specifies how far the colorlegend is inset from the plot margins. If a single value is given, it is used for both margins; if two values are given, the first is used for x- distance, the second for y-distance.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

legend, FindColor, BubbleLegend

Examples

```
plot(1:15,, xlim=c(0,10), type="n", xlab="", ylab="", main="Colorstrips")

# A
ColorLegend(x="right", inset=0.1, labels=c(1:10))

# B: Center the labels
ColorLegend(x=1, y=9, height=6, col=colorRampPalette(c("blue", "white", "red"),
  space = "rgb")(5), labels=1:5, cntrlbl = TRUE)

# C: Outer frame
ColorLegend(x=3, y=9, height=6, col=colorRampPalette(c("blue", "white", "red"),
  space = "rgb")(5), labels=1:4, frame="grey")

# D
ColorLegend(x=5, y=9, height=6, col=colorRampPalette(c("blue", "white", "red"),
  space = "rgb")(10), labels=sprintf("%.1f",seq(0,1,0.1)), cex=0.8)
```

```
# E: horizontal shape
ColorLegend(x=1, y=2, width=6, height=0.2, col=rainbow(500), labels=1:5,horiz=TRUE)

# F
ColorLegend(x=1, y=14, width=6, height=0.5, col=colorRampPalette(
  c("black","blue","green","yellow","red"), space = "rgb")(100), horiz=TRUE)

# G
ColorLegend(x=1, y=12, width=6, height=1, col=colorRampPalette(c("black","blue",
            "green","yellow","red"), space = "rgb")(10), horiz=TRUE,
            border="black", title="From black to red", title.adj=0)

text(x = c(8,0.5,2.5,4.5,0.5,0.5,0.5)+.2, y=c(14,9,9,9,2,14,12), LETTERS[1:7], cex=2)
```

`ColToGrey` *Convert Colors to Grey/Grayscale*

Description

Convert colors to grey/grayscale so that you can see how your plot will look after photocopying or printing to a non-color printer.

Usage

```
ColToGrey(col)
ColToGray(col)
```

Arguments

`col`	vector of any of the three kind of R colors, i.e., either a color name (an element of colors()), a hexadecimal string of the form "#rrggbb" or "#rrggbbaa" (see rgb), or an integer i meaning palette()[i]. Non-string values are coerced to integer.

Details

Converts colors to greyscale using the formula grey = 0.3*red + 0.59*green + 0.11*blue. This allows you to see how your color plot will approximately look when printed on a non-color printer or photocopied.

Value

A vector of colors (greys) corresponding to the input colors.

Note

These function was previously published as `Col2Grey()` in the **TeachingDemos** package and has been integrated here without logical changes.

Author(s)

Greg Snow <greg.snow@imail.org>

See Also

grey, ColToRgb, dichromat package

Examples

```
par(mfcol=c(2,2))
tmp <- 1:3
names(tmp) <- c('red','green','blue')

barplot(tmp, col=c('red','green','blue'))
barplot(tmp, col=ColToGrey(c('red','green','blue')))

barplot(tmp, col=c('red','#008100','#3636ff'))
barplot(tmp, col=ColToGrey(c('red','#008100','#3636ff')))
```

ColToHex *Convert a Color or a RGB-color Into Hex String*

Description

Convert a color given by name, by its palette index or by rgb-values into a string of the form "#rrggbb" or "#rrggbbaa".

Usage

```
ColToHex(col, alpha = 1)
```

Arguments

col	vector of any of either a color name (an element of colors()), or an integer i meaning palette()[i]. Non-string values are coerced to integer.
alpha	the alpha value to be used. This can be any value from 0 (fully transparent) to 1 (opaque). Default is 1.

Value

Returns the colorvalue in #rrggbb" or #rrggbbaa" format. (character)

Author(s)

Andri Signorell <andri@signorell.net>

See Also

HexToCol, ColToRgb, colors

Examples

```
ColToHex(c("lightblue", "salmon"))

x <- ColToRgb("darkmagenta")
x[2,] <- x[2,] + 155
RgbToCol(x)

RgbToHex(c(255,0,0))
```

`ColToHsv` *R Color to HSV Conversion*

Description

ColToHsv transforms colors from R color into HSV space (hue/saturation/value).

Usage

```
ColToHsv(col, alpha = FALSE)
```

Arguments

`col`	vector of any of the three kind of R colors, i.e., either a color name (an element of `colors`()), a hexadecimal string of the form "#rrggbb" or "#rrggbbaa", or an integer i meaning palette()[i]. Non-string values are coerced to integer.
`alpha`	logical value indicating whether alpha channel (opacity) values should be returned.

Details

Converts a color first into RGB an from there into HSV space by means of the functions `rgb2hsv` and `col2rgb`.

Value (brightness) gives the amount of light in the color. Hue describes the dominant wavelength. Saturation is the amount of Hue mixed into the color.

An HSV colorspace is relative to an RGB colorspace, which in R is sRGB, which has an implicit gamma correction.

Value

A matrix with a column for each color. The three rows of the matrix indicate hue, saturation and value and are named "h", "s", and "v" accordingly.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

rgb2hsv, ColToRgb

Examples

```
ColToHsv("peachpuff")
ColToHsv(c(blu = "royalblue", reddish = "tomato")) # names kept

ColToHsv(1:8)
```

ColToOpaque *Equivalent Opaque Color for Transparent Color*

Description

Determine the equivalent opaque RGB color for a given partially transparent RGB color against a background of any color.

Usage

```
ColToOpaque(col, alpha = NULL, bg = NULL)
```

Arguments

col	the color as hex value (use converters below if it's not available). col and alpha are recycled.
alpha	the alpha channel, if left to NULL the alpha channels of the colors are used
bg	the background color to be used to calculate against (default is "white")

Details

Reducing the opacity against a white background is a good way to find usable lighter and less saturated tints of a base color. For doing so, we sometimes need to get the equivalent opaque color for the transparent color.

Value

An named vector with the hexcodes of the opaque colors.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

ColToHex, DecToHex, RgbToHex

Examples

```
cols <- c(SetAlpha("limegreen", 0.4), ColToOpaque(ColToHex("limegreen"), 0.4), "limegreen")
barplot(c(1, 1.2, 1.3), col=cols, panel.first=abline(h=0.4, lwd=10, col="grey35"))
```

ColToRgb *Color to RGB Conversion*

Description

R color to RGB (red/green/blue) conversion.

Usage

```
ColToRgb(col, alpha = FALSE)
```

Arguments

col	vector of any of the three kind of R colors, i.e., either a color name (an element of colors()), a hexadecimal string of the form "#rrggbb" or "#rrggbbaa", or an integer i meaning palette()[i]. Non-string values are coerced to integer.
alpha	logical value indicating whether alpha channel (opacity) values should be returned.

Details

This is merely a wrapper to col2rgb, defined in order to follow this package's naming conventions.

Value

A matrix with a column for each color. The three rows of the matrix indicate red, green and blue value and are named "red", "green", and "blue" accordingly. The matrix might have a 4th row if an alpha channel is requested.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

col2rgb, RgbToCol

Examples

```
ColToRgb("peachpuff")
ColToRgb(c(blu = "royalblue", reddish = "tomato")) # names kept

ColToRgb(1:8)
```

`ColumnWrap` *Column Wrap*

Description

Wraps text in a character matrix so, that it's displayed over more than one line.

Usage

```
ColumnWrap(x, width = NULL)
```

Arguments

`x`	the matrix with one row
`width`	integer, the width of the columns in characters

Details

A data.frame containing character columns with long texts is often wrapped by columns. This can lead to a loss of overview. `ColumnWrap()` wraps the lines within the columns.

Value

a character matrix

Author(s)

Andri Signorell andri@signorell.net

See Also

`strwrap()`

Examples

```
Abstract(d.pizza)
```

CombPairs *Get All Pairs Out of One or Two Sets of Elements*

Description

Returns all combinations of 2 out of the elements in x or x and y (if defined). Combinations of the same elements will be dropped (no replacing).

Usage

```
CombPairs(x, y = NULL)
```

Arguments

x	a vector of elements
y	a vector of elements, need not be same dimension as x. If y is not NULL then all combination x and y are returned.

Details

If y = NULL then all combination of 2 out of x are returned.
If y is defined then all combinations of x and y are calculated.

Value

CombPairs returns a data.frame with 2 columns X1 and X2.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

combn, expand.grid, outer, lower.tri

Examples

```
CombPairs(letters[1:4])
CombPairs(x = letters[1:4], y = LETTERS[1:2])

# get all pairs of combinations between factors and numerics out of a data.frame
CombPairs(which(sapply(d.pizza, IsNumeric)), which(sapply(d.pizza, is.factor)))
```

CompleteColumns *Find Complete Columns*

Description

Return either the columnnames or a logical vector indicating which columns are complete, i.e., have no missing values.

Usage

```
CompleteColumns(x, which = TRUE)
```

Arguments

x	a data.frame containing the data
which	logical, determining if the names of the variables should be returned or a if a logical vector indicating which columns are complete should be returned.

Value

A logical vector specifying which columns have no missing values across the entire sequence.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

is.na, na.omit, complete.cases

Examples

```
CompleteColumns(d.pizza)
CompleteColumns(d.pizza, which=FALSE)
```

ConDisPairs *Concordant and Discordant Pairs*

Description

This function counts concordant and discordant pairs for two variables x, y with at least ordinal scale, aggregated in a 2way table. This is the base for many association measures like Goodman Kruskal's gamma, but also all tau measures.

Usage

```
ConDisPairs(x)
```

Arguments

x	a 2-dimensional table. The column and the row order must be the logical one.

Details

The code is so far implemented in R (O(n^2)) and therefore slow for large sample sizes (>5000).

An O(n log(n)) implementation is available as (so far) undocumented function `DescTools:::.DoCount(x, y, wts)` returning only concorant and discordant pairs (not including standard errors to be used for calculating confidence intervals).

Value

a list with the number of concordant pairs, the number of discordant pairs and the matrix

Author(s)

Andri Signorell <andri@signorell.net>

References

Agresti, A. (2002) *Categorical Data Analysis*. John Wiley & Sons, pp. 57-59.

Goodman, L. A., & Kruskal, W. H. (1954) Measures of association for cross classifications. *Journal of the American Statistical Association*, 49, 732-764.

Goodman, L. A., & Kruskal, W. H. (1963) Measures of association for cross classifications III: Approximate sampling theory. *Journal of the American Statistical Association*, 58, 310-364.

See Also

Association measures:
`KendallTauA` (tau-a), `cor` (method="kendall") for tau-b, `StuartTauC` (tau-c), `SomersDelta` `Lambda`, `GoodmanKruskalTau` (tau), `UncertCoef`, `MutInf`

Examples

```
tab <- as.table(rbind(c(26,26,23,18,9),c(6,7,9,14,23)))
ConDisPairs(tab)
```

Conf *Confusion Matrix And Associated Statistics*

Description

Calculates a cross-tabulation of observed and predicted classes with associated statistics.

Usage

```
Conf(x, ...)

## S3 method for class 'table'
Conf(x, pos = NULL, ...)
## S3 method for class 'matrix'
Conf(x, pos = NULL, ...)
## Default S3 method:
Conf(x, ref, pos = NULL, na.rm = TRUE, ...)

## S3 method for class 'rpart'
Conf(x, ...)
## S3 method for class 'multinom'
Conf(x, ...)
## S3 method for class 'glm'
Conf(x, cutoff = 0.5, pos = NULL, ...)
## S3 method for class 'randomForest'
Conf(x, ...)
## S3 method for class 'svm'
Conf(x, ...)
## S3 method for class 'regr'
Conf(x, ...)

## S3 method for class 'Conf'
plot(x, main = "Confusion Matrix", ...)

## S3 method for class 'Conf'
print(x, digits = max(3, getOption("digits") - 3), ...)

Sens(x, ...)
Spec(x, ...)
```

Arguments

x	a vector, normally a factor, of predicted classes or an object of following classes rpart, randomForest, svm, C50, glm, multinom, regr, lda, qda or table, resp. matrix. When a model is given, the predicted classes will be determined. A table or a matrix will be interpreted as a confusion matrix.
ref	a vector, normally a factor, of classes to be used as the reference. This is ignored if x is a table or matrix.
pos	a character string that defines the factor level corresponding to the "positive" results. Will be ignored for a $n \times n$ table n > 2.
cutoff	used in logit models. The cutoff for changing classes.
main	overall title for the plot.
digits	controls the number of digits to print.

na.rm	a logical value indicating whether or not missing values should be removed. Defaults to FALSE.
...	further arguments to be passed to or from methods.

Details

The functions require the factors to have the same levels.

For two class problems, the sensitivity, specificity, positive predictive value and negative predictive value is calculated using the positive argument. Also, the prevalence of the "event" is computed from the data (unless passed in as an argument), the detection rate (the rate of true events also predicted to be events) and the detection prevalence (the prevalence of predicted events).

Suppose a 2×2 table with notation

	Reference	
Predicted	Event	No Event
Event	A	B
No Event	C	D

The formulas used here are:

$$Sensitivity = A/(A + C)$$

$$Specificity = D/(B + D)$$

$$Prevalence = (A + C)/(A + B + C + D)$$

$$PPV = (sensitivity * Prevalence)/((sensitivity * Prevalence) + ((1 - specificity) * (1 - Prevalence)))$$

$$NPV = (specificity * (1 - Prevalence))/(((1 - sensitivity) * Prevalence) + ((specificity) * (1 - Prevalence)))$$

$$DetectionRate = A/(A + B + C + D)$$

$$DetectionPrevalence = (A + B)/(A + B + C + D)$$

$$F - valAccuracy = 2/(1/PPV + 1/Sensitivity)$$

$$MatthewsCor. - Coef = (A * D - B * C)/sqrt((A + B) * (A + C) * (D + B) * (D + C))$$

See the references for discusions of the first five formulas.

For more than two classes, these results are calculated comparing each factor level to the remaining levels (i.e. a "one versus all" approach).

The overall accuracy and unweighted Kappa statistic are calculated. A p-value from McNemar's test is also computed using mcnemar.test (which can produce NA values with sparse tables).

The overall accuracy rate is computed along with a 95 percent confidence interval for this rate (using BinomCI) and a one-sided test to see if the accuracy is better than the "no information rate," which is taken to be the largest class percentage in the data.

The sensitivity is defined as the proportion of positive results out of the number of samples which were actually positive. When there are no positive results, sensitivity is not defined and a value of NA is returned. Similarly, when there are no negative results, specificity is not defined and a value of NA is returned. Similar statements are true for predictive values.

Confidence intervals for sensitivity, specificity etc. could be calculated as binomial confidence intervals (see BinomCI). BinomCI(A, A+C) yields the ci for sensitivity.

Value

a list with elements

`table`	the results of `table` on `data` and `reference`
`positive`	the positive result level
`overall`	a numeric vector with overall accuracy and Kappa statistic values
`byClass`	the sensitivity, specificity, positive predictive value, negative predictive value, prevalence, dection rate and detection prevalence for each class. For two class systems, this is calculated once using the `positive` argument

Author(s)

Andri Signorell <andri@signorell.net>
rewritten based on the ideas of `confusionMatrix` by Max Kuhn <Max.Kuhn@pfizer.com>

References

Kuhn, M. (2008) Building predictive models in R using the caret package *Journal of Statistical Software*, (https://www.jstatsoft.org/v28/i05/).

Powers, David M W (2011) Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation (PDF). *Journal of Machine Learning Technologies* 2 (1): 37-63.

Collett D (1999) Modelling Binary Data. *Chapman & Hall/CRC*, Boca Raton Florida, pp. 24.

Matthews, B. W. (1975) Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA) - Protein Structure* 405 (2): 442-451. doi:10.1016/0005-2795(75)90109-9. PMID 1180967.

See Also

`OddsRatio`, `RelRisk`

Examples

```
# let tab be a confusion table
tab <- TextToTable("
   lo hi
lo 23 13
hi 10 18 ", dimnames=c("pred", "obs"))

Conf(tab, pos="hi")


pred <- Untable(tab)[,"pred"]
obs <- Untable(tab)[,"obs"]

Conf(x = pred, ref = obs)
Conf(x = pred, ref = obs, pos="hi")

Sens(tab)   # Sensitivity
```

```
Spec(tab)   # Specificity


tab <- TextToTable("
      terrible poor marginal clear
terrible       10    4        1     0
poor            5   10       12     2
marginal        2    4       12     5
clear           0    2        6    13
", dimnames=c("pred", "obs"))

Conf(tab)
```

ConnLines *Add Connection Lines to a Barplot*

Description

Add connection lines to a stacked barplot (beside = TRUE is not supported). The function expects exactly the same arguments, that were used to create the barplot.

Usage

```
ConnLines(..., col = 1, lwd = 1, lty = "solid", xalign = c("mar", "mid"))
```

Arguments

...	the arguments used to create the barplot. (The dots are sent directly to barplot).
col	the line color of the connection lines. Defaults to black.
lwd	the line width for the connection lines. Default is 1.
lty	the line type for the connection lines. Line types can either be specified as an integer (0=blank, 1=solid (default), 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash) or as one of the character strings "blank", "solid", "dashed", "dotted", "dotdash", "longdash", or "twodash". Default is "solid".
xalign	defines where the lines should be aligned to on the x-axis. Can be set either to the margins of the bars ("mar" which is the default) or to "mid". The latter will lead the connecting lines to the middle of the bars.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

BarText, barplot

Examples

```
tab <- with(
  subset(d.pizza, driver %in% c("Carpenter", "Miller", "Farmer", "Butcher")),
  table(factor(driver), Weekday(date, "dd", stringsAsFactor=TRUE))
)
tab

barplot(tab, beside=FALSE, space=1.2)
ConnLines(tab, beside=FALSE, space=1.2, col="grey50", lwd=1, lty=2)

barplot(tab, beside=FALSE, space=1.2, horiz=TRUE)
ConnLines(tab, beside=FALSE, space=1.2, horiz=TRUE, col="grey50", lwd=1, lty=2)


cols <- Pal("Helsana")[1:4]
b <- barplot(tab, beside=FALSE, horiz=FALSE, col=cols)
ConnLines(tab, beside=FALSE, horiz=FALSE, col="grey50", lwd=1, lty=2)

# set some labels
BarText(tab, b,
        labels=Format(tab, zero.form = "", d=0), pos = "mid",
        col=(matrix(rep(TextContrastColor(cols), each=ncol(tab)),
                    nrow=nrow(tab), byrow=FALSE )))

# align to the middle of the bars
barplot(tab, beside=FALSE, space=1.2)
ConnLines(tab, beside=FALSE, space=1.2, col="grey50", lwd=1, lty=2, method="mid")
```

ConoverTest *Conover's Test of Multiple Comparisons*

Description

Perform Conover's test of multiple comparisons using rank sums as post hoc test following a significant kruskal.test.

Usage

```
ConoverTest(x, ...)

## Default S3 method:
ConoverTest(x, g,
            method = c("holm", "hochberg", "hommel", "bonferroni", "BH",
                       "BY", "fdr", "none"),
            alternative = c("two.sided", "less", "greater"),
            out.list = TRUE, ...)

## S3 method for class 'formula'
ConoverTest(formula, data, subset, na.action, ...)
```

Arguments

x	a numeric vector of data values, or a list of numeric data vectors.
g	a vector or factor object giving the group for the corresponding elements of x. Ignored if x is a list.
method	the method for adjusting p-values for multiple comparisons. The function is calling p.adjust and this parameter is directly passed through.
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter.
out.list	logical, indicating if the results should be printed in list mode or as a square matrix. Default is list (TRUE).
formula	a formula of the form lhs ~ rhs where lhs gives the data values and rhs the corresponding groups.
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
...	further arguments to be passed to or from methods.

Details

ConoverTest performs the post hoc pairwise multiple comparisons procedure appropriate to follow the rejection of a Kruskal-Wallis test. Conover's test is more powerful than Dunn's post hoc multiple comparisons test (DunnTest). The interpretation of stochastic dominance requires an assumption that the CDF of one group does not cross the CDF of the other.
ConoverTest makes m = k(k-1)/2 multiple pairwise comparisons based on the Conover-Iman t-test-statistic for the rank-sum differences:

$$\left|\bar{R}_i - \bar{R}_j\right| > t_{1-\alpha/2,n-k} \cdot \sqrt{s^2 \cdot \left[\frac{n-1-\hat{H}^*}{n-k}\right] \cdot \left[\frac{1}{n_i}+\frac{1}{n_j}\right]}$$

with the (tie corrected) statistic of the Kruskal Wallis test

$$\hat{H}^* = \frac{\frac{12}{n\cdot(n+1)} \cdot \sum_{i=1}^{k} \frac{R_i^2}{n_i} - 3\cdot(n+1)}{1 - \frac{\sum_{i=1}^{r}\left(t_i^3 - t_i\right)}{n^3-n}}$$

and the s^2 being

$$s^2 = \frac{1}{n-1} \cdot \left[\sum R_i^2 - n \cdot \frac{(n+1)^2}{4}\right]$$

If x is a list, its elements are taken as the samples to be compared, and hence have to be numeric data vectors. In this case, g is ignored, and one can simply use ConoverTest(x) to perform the test. If the samples are not yet contained in a list, use ConoverTest(list(x, ...)).

Otherwise, x must be a numeric data vector, and g must be a vector or factor object of the same length as x giving the group for the corresponding elements of x.

Value

A list with class "DunnTest" containing the following components:

res	an array containing the mean rank differencens and the according p-values

Author(s)

Andri Signorell <andri@signorell.net>, the interface is based on R-Core code

References

Conover W. J., Iman R. L. (1979) On multiple-comparisons procedures, *Tech. Rep.* LA-7677-MS, Los Alamos Scientific Laboratory.

Conover, W. J. (1999) Practical Nonparametric Statistics *Wiley*, Hoboken, NJ. 3rd edition.

See Also

DunnTest, NemenyiTest, kruskal.test, wilcox.test, p.adjust

Examples

```
## Hollander & Wolfe (1973), 116.
## Mucociliary efficiency from the rate of removal of dust in normal
##  subjects, subjects with obstructive airway disease, and subjects
##  with asbestosis.
x <- c(2.9, 3.0, 2.5, 2.6, 3.2) # normal subjects
y <- c(3.8, 2.7, 4.0, 2.4)      # with obstructive airway disease
z <- c(2.8, 3.4, 3.7, 2.2, 2.0) # with asbestosis
ConoverTest(list(x, y, z))

## Equivalently,
x <- c(x, y, z)
g <- factor(rep(1:3, c(5, 4, 5)),
            labels = c("Normal subjects",
                       "Subjects with obstructive airway disease",
                       "Subjects with asbestosis"))

# do the kruskal.test first
kruskal.test(x, g)

# ...and the pairwise test afterwards
ConoverTest(x, g)

## Formula interface.
boxplot(Ozone ~ Month, data = airquality)
ConoverTest(Ozone ~ Month, data = airquality)
```

`Contrasts` *Pairwise Contrasts*

Description

Generate all pairwise contrasts for using in a post-hoc test, e.g. ScheffeTest.

Usage

```
Contrasts(levs)
```

Arguments

`levs`	the levels to be used

Value

A matrix with all possible pairwise contrasts, that can be built with the given levels.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`ScheffeTest`

Examples

```
Contrasts(LETTERS[1:5])

#   B-A C-A D-A E-A C-B D-B E-B D-C E-C E-D
# A  -1  -1  -1  -1   0   0   0   0   0   0
# B   1   0   0   0  -1  -1  -1   0   0   0
# C   0   1   0   0   1   0   0  -1  -1   0
# D   0   0   1   0   0   1   0   1   0  -1
# E   0   0   0   1   0   0   1   0   1   1
```

ConvUnit *Unit Conversion and Metrix Prefixes*

Description

Converts a numerical vector from one measurement system to another. Metric prefixes (as unit prefixes that precede a basic unit of measure to indicate a multiple or fraction of the unit) are respected.

Usage

```
CmToPts(x)
PtsToCm(x)

ConvUnit(x, from, to)

data(d.units)
data(d.prefix)
```

Arguments

x	the numeric to be converted.
from	a character defining the original unit.
to	a character defining the target unit.

Details

The two functions `CmToPts()` and `PtsToCm()` convert centimeters to points and vice versa. 1 cm corresponds to 28.35 points.

The units as defined by the International System of Units (SI) (m, g, s, A, K, mol, cd, Hz, rad, sr, N, Pa, J, W, C, V, F, Ohm, S, Wb, T, H, lm, lx, Bq, Gy, Sv, kat, l) can be used to convert between different prefixes. The following non SI-units can be chosen for conversion between different systems. `NA` will be returned if a conversion can't be found.
The function is using the conversion factors stored in the dataset `d.units`.

Weight and mass		
Gram	g	metric
Pound mass (avoirdupois)	lb	
Ounce mass (avoirdupois)	oz	
Metric ton	ton (or tn)	
Distance		
Meter	m	metric

Statute mile	mi	
Nautical mile	nmi	
Inch	in	
Foot	ft	
Yard	yd	
Angstrom	AA	(accepted) metric
Astronomical unit	au	

Time

Year	a	
Day	d	
Hour	h	
Minute	min	
Second	s	

Pressure

Pascal	Pa	
Atmosphere	atm	
mm of Mercury	mmHg	
bar	bar	
Pound-force per quare inch	psi	

Energy

Joule	J	metric
IT calorie	cal	(accepted) metric
Electron volt	eV (or ev)	

Power

Horsepower (mechanical)	hp	
Horsepower (metric)	HP	
Watt	W (or w)	metric

Temperature

Degree Celsius	C	
Degree Fahrenheit	F	
Kelvin	K	metric

Liquid measure

Fluid ounce	fl oz	
Gallon	gal	
Liter	l (or lt)	(accepted) metric

Additional details can be found in the `d.units` data.frame.

Author(s)

Andri Signorell <andri@signorell.net>

Examples

```
ConvUnit(c(1.2, 5.4, 6.7), "in", "m")

# from kilometers to pico meters
ConvUnit(1, from="km", to="pm")

# from miles to kilometers
ConvUnit(1, from="mi", to="km")
# nautical miles
ConvUnit(1, from="nmi", to="km")
# from kilo Kelvin to Fahrenheit
ConvUnit(10, from="kK", to="F")
# from metric to more quirky units
ConvUnit(c(10, 1), from="hl", to="gal")
ConvUnit(500, from="ml", to="fl oz")

# conversion between non-SI units
ConvUnit(1000, "yd", "mi")
# ... must be the same as
ConvUnit(ConvUnit(1000, "yd", "m"), "m", "mi")
```

Cor *Covariance and Correlation (Matrices)*

Description

Cov and Cor compute the covariance or correlation of x and y if these are vectors. If x and y are matrices then the covariances (or correlations) between the columns of x and the columns of y are computed.

Usage

```
Cov(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))

Cor(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))
```

Arguments

x	a numeric vector, matrix or data frame.
y	NULL (default) or a vector, matrix or data frame with compatible dimensions to x. The default is equivalent to y = x (but more efficient).

use — an optional character string giving a method for computing covariances in the presence of missing values. This must be (an abbreviation of) one of the strings "everything", "all.obs", "complete.obs", "na.or.complete", or "pairwise.complete.obs".

method — a character string indicating which correlation coefficient (or covariance) is to be computed. One of "pearson" (default), "kendall", or "spearman": can be abbreviated.

Details

For Cov and Cor one must *either* give a matrix or data frame for x *or* give both x and y.

The inputs must be numeric (as determined by is.numeric: logical values are also allowed for historical compatibility): the "kendall" and "spearman" methods make sense for ordered inputs but xtfrm can be used to find a suitable prior transformation to numbers.

If use is "everything", NAs will propagate conceptually, i.e., a resulting value will be NA whenever one of its contributing observations is NA.
If use is "all.obs", then the presence of missing observations will produce an error. If use is "complete.obs" then missing values are handled by casewise deletion (and if there are no complete cases, that gives an error).
"na.or.complete" is the same unless there are no complete cases, that gives NA. Finally, if use has the value "pairwise.complete.obs" then the correlation or covariance between each pair of variables is computed using all complete pairs of observations on those variables. This can result in covariance or correlation matrices which are not positive semi-definite, as well as NA entries if there are no complete pairs for that pair of variables. For Cov and Var, "pairwise.complete.obs" only works with the "pearson" method. Note that (the equivalent of) Var(double(0), use = *) gives NA for use = "everything" and "na.or.complete", and gives an error in the other cases.

The denominator $n - 1$ is used which gives an unbiased estimator of the (co)variance for i.i.d. observations. These functions return NA when there is only one observation (whereas S-PLUS has been returning NaN), and fail if x has length zero.

For Cor(), if method is "kendall" or "spearman", Kendall's τ or Spearman's ρ statistic is used to estimate a rank-based measure of association. These are more robust and have been recommended if the data do not necessarily come from a bivariate normal distribution.
For Cov(), a non-Pearson method is unusual but available for the sake of completeness. Note that "spearman" basically computes Cor(R(x), R(y)) (or Cov(., .)) where R(u) := rank(u, na.last = "keep"). In the case of missing values, the ranks are calculated depending on the value of use, either based on complete observations, or based on pairwise completeness with reranking for each pair.

Scaling a covariance matrix into a correlation one can be achieved in many ways, mathematically most appealing by multiplication with a diagonal matrix from left and right, or more efficiently by using sweep(.., FUN = "/") twice.

Value

For r <- Cor(*, use = "all.obs"), it is now guaranteed that all(abs(r) <= 1).

Note

Some people have noted that the code for Kendall's tau is slow for very large datasets (many more than 1000 cases). It rarely makes sense to do such a computation, but see function cor.fk in

package **pcaPP**.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`cor.test` for confidence intervals (and tests).

`cov.wt` for *weighted* covariance computation.

`Var`, `SD` for variance and standard deviation (vectors).

Examples

```
## Two simple vectors
Cor(1:10, 2:11) # == 1

## Correlation Matrix of Multivariate sample:
(Cl <- Cor(longley))
## Graphical Correlation Matrix:
symnum(Cl) # highly correlated

## Spearman's rho  and  Kendall's tau
symnum(clS <- Cor(longley, method = "spearman"))
symnum(clK <- Cor(longley, method = "kendall"))
## How much do they differ?
i <- lower.tri(Cl)
Cor(cbind(P = Cl[i], S = clS[i], K = clK[i]))


##--- Missing value treatment:
C1 <- Cov(swiss)
range(eigen(C1, only.values = TRUE)$values) # 6.19        1921

## swM := "swiss" with  3 "missing"s :
swM <- swiss
colnames(swM) <- abbreviate(colnames(swiss), min=6)
swM[1,2] <- swM[7,3] <- swM[25,5] <- NA # create 3 "missing"

## Consider all 5 "use" cases :
(C. <- Cov(swM)) # use="everything"  quite a few NA's in cov.matrix
try(Cov(swM, use = "all")) # Error: missing obs...
C2 <- Cov(swM, use = "complete")
stopifnot(identical(C2, Cov(swM, use = "na.or.complete")))
range(eigen(C2, only.values = TRUE)$values) # 6.46   1930
C3 <- Cov(swM, use = "pairwise")
range(eigen(C3, only.values = TRUE)$values) # 6.19   1938

## Kendall's tau doesn't change much:
symnum(Rc <- Cor(swM, method = "kendall", use = "complete"))
symnum(Rp <- Cor(swM, method = "kendall", use = "pairwise"))
```

```
symnum(R. <- Cor(swiss, method = "kendall"))

## "pairwise" is closer componentwise,
summary(abs(c(1 - Rp/R.)))
summary(abs(c(1 - Rc/R.)))

## but "complete" is closer in Eigen space:
EV <- function(m) eigen(m, only.values=TRUE)$values
summary(abs(1 - EV(Rp)/EV(R.)) / abs(1 - EV(Rc)/EV(R.)))
```

`CorCI` *Confidence Intervals for Pearson Correlation*

Description

Find the confidence intervals for a specified correlation based on Fisher's z-transformation.

Usage

```
CorCI(
  rho,
  n,
  conf.level = 0.95,
  alternative = c("two.sided", "less", "greater")
)
```

Arguments

`rho`	the Pearson's correlation coefficient
`n`	sample size used for calculating the confidence intervals
`conf.level`	confidence level for the returned confidence interval, restricted to lie between zero and one.
`alternative`	is a character string, one of `"greater"`, `"less"`, or `"two.sided"`, or the initial letter of each, indicating the specification of the alternative hypothesis. `"greater"` corresponds to positive association, `"less"` to negative association.

Details

The sampling distribution of Pearson's r is not normal. Fisher developed a transformation now called "Fisher's z-transformation" used for the calculation of normal distributed confidence intervals.

Value

rho, lower and upper confidence intervals (CorCI)

Author(s)

William Revelle revelle@northwestern.edu,
slight modifications Andri Signorell andri@signorell.net based on R-Core code

See Also

FisherZ, FisherZInv

Examples

```
cors <- seq(-.9, .9, .1)

zs <- FisherZ(cors)
rs <- FisherZInv(zs)
round(zs, 2)
n <- 30
r <- seq(0, .9, .1)
rc <- t(sapply(r, CorCI, n=n))
t <- r * sqrt(n-2) / sqrt(1-r^2)
p <- (1 - pt(t, n-2)) / 2

r.rc <- data.frame(r=r, z=FisherZ(r), lower=rc[,2], upper=rc[,3], t=t, p=p)

round(r.rc,2)
```

CorPart *Find the Correlations for a Set x of Variables With Set y Removed*

Description

A straightforward application of matrix algebra to remove the effect of the variables in the y set from the x set. Input may be either a data matrix or a correlation matrix. Variables in x and y are specified by location.

Usage

```
CorPart(m, x, y)
```

Arguments

m	a data or correlation matrix.
x	the variable numbers associated with the X set.
y	the variable numbers associated with the Y set.

Details

It is sometimes convenient to partial the effect of a number of variables (e.g., sex, age, education) out of the correlations of another set of variables. This could be done laboriously by finding the residuals of various multiple correlations, and then correlating these residuals. The matrix algebra alternative is to do it directly.

Value

The matrix of partial correlations.

Author(s)

William Revelle

References

Revelle, W. *An introduction to psychometric theory with applications in R* Springer.
(working draft available at http://personality-project.org/r/book/

See Also

cor

Examples

```
# example from Bortz, J. (1993) Statistik fuer Sozialwissenschaftler, Springer, pp. 413

abstr <- c(9,11,13,13,14,9,10,11,10,8,13,7,9,13,14)
coord <- c(8,12,14,13,14,8,9,12,8,9,14,7,10,12,12)
age <- c(6,8,9,9,10,7,8,9,8,7,10,6,10,10,9)

# calculate the correlation of abstr and coord, after without the effect of the age
CorPart(cbind(abstr, coord, age), 1:2, 3)

# by correlation matrix m
m <- cor(cbind(abstr, coord, age))
CorPart(m, 1:2, 3)

# ... which would be the same as:
lm1 <- lm(abstr ~ age)
lm2 <- lm(coord ~ age)

cor(resid(lm1), resid(lm2))
```

`CorPolychor` *Polychoric Correlation*

Description

Computes the polychoric correlation (and its standard error) between two ordinal variables or from their contingency table, under the assumption that the ordinal variables dissect continuous latent variables that are bivariate normal. Either the maximum-likelihood estimator or a (possibly much) quicker "two-step" approximation is available. For the ML estimator, the estimates of the thresholds and the covariance matrix of the estimates are also available.

Usage

```
CorPolychor(x, y, ML = FALSE, control = list(), std.err = FALSE, maxcor=.9999)

## S3 method for class 'CorPolychor'
print(x, digits = max(3, getOption("digits") - 3), ...)
```

Arguments

`x`	a contingency table of counts or an ordered categorical variable; the latter can be numeric, logical, a factor, or an ordered factor, but if a factor, its levels should be in proper order.
`y`	if `x` is a variable, a second ordered categorical variable.
`ML`	if `TRUE`, compute the maximum-likelihood estimate; if `FALSE`, the default, compute a quicker "two-step" approximation.
`control`	optional arguments to be passed to the `optim` function.
`std.err`	if `TRUE`, return the estimated variance of the correlation (for the two-step estimator) or the estimated covariance matrix (for the ML estimator) of the correlation and thresholds; the default is `FALSE`.
`maxcor`	maximum absolute correlation (to insure numerical stability).
`digits`	integer, determining the number of digits used to format the printed result
`...`	not used

Value

If `std.err` is `TRUE`, returns an object of class `"polycor"` with the following components:

`type`	set to `"polychoric"`.
`rho`	the CorPolychoric correlation.
`var`	the estimated variance of the correlation, or, for the ML estimate, the estimated covariance matrix of the correlation and thresholds.
`n`	the number of observations on which the correlation is based.

chisq	chi-square test for bivariate normality.
df	degrees of freedom for the test of bivariate normality.
ML	TRUE for the ML estimate, FALSE for the two-step estimate.

Othewise, returns the polychoric correlation.

Note

This is a verbatim copy from polchor function in the package polycor.

Author(s)

John Fox <jfox@mcmaster.ca>

References

Drasgow, F. (1986) CorPolychoric and polyserial correlations. Pp. 68–74 in S. Kotz and N. Johnson, eds., *The Encyclopedia of Statistics, Volume 7.* Wiley.

Olsson, U. (1979) Maximum likelihood estimation of the CorPolychoric correlation coefficient. *Psychometrika* **44**, 443-460.

See Also

hetcor, polyserial, print.CorPolychor, optim

Examples

```
set.seed(12345)
z <- RndPairs(1000, 0.6)
x <- z[,1]
y <- z[,2]

cor(x, y)                                  # sample correlation
x <- cut(x, c(-Inf, .75, Inf))
y <- cut(y, c(-Inf, -1, .5, 1.5, Inf))

CorPolychor(x, y)                          # 2-step estimate
CorPolychor(x, y, ML=TRUE, std.err=TRUE)   # ML estimate
```

CountCompCases *Count Complete Cases*

Description

Return for each variable of a data frame the number of missing values and the complete cases to be expected if this variable would be omitted.

Usage

```
CountCompCases(x)

## S3 method for class 'CountCompCases'
print(x, digits=1, ...)
```

Arguments

x	a data.frame containg the data.
digits	the number of digits to be used when printing the results.
...	the dots are not further used.

Value

A list with three elements. The first gives the number of rows, the second the number of complete cases for the whole data frame. The third element tab contains the data for the single variables.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

PlotMiss, CompleteColumns, complete.cases, is.na, na.omit

Examples

```
CountCompCases(d.pizza)
```

CountWorkDays *Count Work Days Between Two Dates*

Description

Returns the number of work days between two dates taking into account the provided holiday dates.

Usage

```
CountWorkDays(from, to, holiday = NULL, nonworkdays = c("Sat", "Sun"))
```

Arguments

from	the initial dates
to	the final dates
holiday	a vector with dates to be excluded.
nonworkdays	a character vector containing the abbreviations of the weekdays (as in day.abb) to be considered non work days. Default is c("Sat","Sun").

Details

The function is vectorised so that multiple initial and final dates can be supplied. The dates are recycled, if their number are different

Value

an integer vector

Author(s)

Andri Signorell <andri@signorell.net>

See Also

weekdays, Date Functions

Examples

```
from <- as.Date("2019-01-01") + rep(0, 10)
to <- as.Date("2020-01-15") + seq(0, 9)

CountWorkDays(from, to)

x <- seq(from[1], from[1]+11, "days")
data.frame(
  date = x,
  day  = Format(x, fmt="ddd"))

CountWorkDays(from = min(x), to = max(x), holiday = c("2019-01-06", "2019-01-07"))
```

CourseData *Get HWZ Datasets*

Description

Wrapper for didactical datasets used in statistic courses.

Usage

```
CourseData(name, url = NULL, header = TRUE, sep = ";", ...)
```

Arguments

name	the name of the file, usually without extension.
url	a url where the data reside, should have the form "http://www.mysite.net/data/". Defaults to the data folder on my site.

header	a logical value indicating whether the file contains the names of the variables as its first line. If missing, the value is determined from the file format: header is set to TRUE if and only if the first row contains one fewer field than the number of columns.
sep	the field separator character. Values on each line of the file are separated by this character. Default is - unlike in read.table - ";" instead of 'white space'.
...	the dots are sent to read.table.

Value

A data.frame containing a representation of the data in the file.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

read.table

Examples

```
## Not run:
d.farm <- CourseData("farmer")

## End(Not run)
```

CramerVonMisesTest *Cramer-von Mises Test for Normality*

Description

Performs the Cramer-von Mises test for the composite hypothesis of normality, see e.g. Thode (2002, Sec. 5.1.3).

Usage

```
CramerVonMisesTest(x)
```

Arguments

x	a numeric vector of data values, the number of which must be greater than 7. Missing values are allowed.

Details

The Cramer-von Mises test is an EDF omnibus test for the composite hypothesis of normality. The test statistic is

$$W = \frac{1}{12n} + \sum_{i=1}^{n} \left(p_{(i)} - \frac{2i-1}{2n} \right),$$

where $p_{(i)} = \Phi([x_{(i)} - \overline{x}]/s)$. Here, Φ is the cumulative distribution function of the standard normal distribution, and $\overline{x}$ and s are mean and standard deviation of the data values. The p-value is computed from the modified statistic $Z = W(1.0 + 0.5/n)$ according to Table 4.9 in Stephens (1986).

Value

A list of class `htest`, containing the following components:

`statistic`	the value of the Cramer-von Mises statistic.
`p.value`	the p-value for the test.
`method`	the character string "Cramer-von Mises normality test".
`data.name`	a character string giving the name(s) of the data.

Author(s)

Juergen Gross <gross@statistik.uni-dortmund.de>

References

Stephens, M.A. (1986) Tests based on EDF statistics In: D'Agostino, R.B. and Stephens, M.A., eds.: *Goodness-of-Fit Techniques*. Marcel Dekker, New York.

Thode Jr., H.C. (2002) *Testing for Normality* Marcel Dekker, New York.

See Also

`shapiro.test` for performing the Shapiro-Wilk test for normality. `AndersonDarlingTest`, `LillieTest`, `PearsonTest`, `ShapiroFranciaTest` for performing further tests for normality. `qqnorm` for producing a normal quantile-quantile plot.

Examples

```
CramerVonMisesTest(rnorm(100, mean = 5, sd = 3))
CramerVonMisesTest(runif(100, min = 2, max = 4))
```

CronbachAlpha *Cronbach's Coefficient Alpha*

Description

Cronbach's alpha is a measure of internal consistency and often used for validating psychometric tests. It determines the internal consistency or average correlation of items in a survey instrument to gauge its reliability. This reduces to Kuder-Richardson formula 20 (KR-20) when the columns of the data matrix are dichotomous.

Usage

```
CronbachAlpha(x, conf.level = NA, cond = FALSE, na.rm = FALSE)
```

Arguments

x	$n \times m$ matrix or dataframe with item responses, k subjects (in rows) m items (in columns).
conf.level	confidence level of the interval. If set to NA (which is the default) no confidence interval will be calculated.
cond	logical. If set to TRUE, alpha is additionally calculated for the dataset with each item left out.
na.rm	logical, indicating whether NA values should be stripped before the computation proceeds. If set to TRUE only the complete cases of the ratings will be used. Defaults to FALSE.

Value

Either a numeric value or
a named vector of 3 columns if confidence levels are required (estimate, lower and upper ci) or

a list containing the following components, if the argument cond is set to TRUE:

unconditional	Cronbach's Alpha, either the single value only or with confidence intervals
condCronbachAlpha	The alpha that would be realized if the item were excluded

Author(s)

Andri Signorell <andri@signorell.net>, based on code of Harold C. Doran

References

Cohen, J. (1960), A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20, 37-46.

See Also

CohenKappa, KappaM

Examples

```
set.seed(1234)
tmp <- data.frame(
  item1=sample(c(0,1), 20, replace=TRUE),
  item2=sample(c(0,1), 20, replace=TRUE),
  item3=sample(c(0,1), 20, replace=TRUE),
  item4=sample(c(0,1), 20, replace=TRUE),
  item5=sample(c(0,1), 20, replace=TRUE)
  )

CronbachAlpha(tmp[,1:4], cond=FALSE, conf.level=0.95)
CronbachAlpha(tmp[,1:4], cond=TRUE, conf.level=0.95)

CronbachAlpha(tmp[,1:4], cond=FALSE)
CronbachAlpha(tmp[,1:2], cond=TRUE, conf.level=0.95)

## Not run:
# Calculate bootstrap confidence intervals for CronbachAlpha
library(boot)
cronbach.boot <- function(data,x) {CronbachAlpha(data[x,])[[3]]}
res <- boot(datafile, cronbach.boot, 1000)
quantile(res$t, c(0.025,0.975))  # two-sided bootstrapped confidence interval of Cronbach's alpha
boot.ci(res, type="bca")       # adjusted bootstrap percentile (BCa) confidence interval (better)

## End(Not run)
```

Cross *Vector Cross Product*

Description

Vector or cross product

Usage

```
Cross(x, y)
```

Arguments

x	numeric vector or matrix
y	numeric vector or matrix

Details

Computes the cross (or: vector) product of vectors in 3 dimensions. In case of matrices it takes the first dimension of length 3 and computes the cross product between corresponding columns or rows.

The more general cross product of `n-1` vectors in n-dimensional space is realized as `CrossN`.

Value

3-dim. vector if `x` and `<` are vectors, a matrix of 3-dim. vectors if `x` and `y` are matrices themselves.

Author(s)

Hans W. Borchers <hwborchers@googlemail.com>

See Also

`Dot`, `CrossN`

Examples

```
Cross(c(1, 2, 3), c(4, 5, 6))  # -3  6 -3

# Triple product can be calculated as:
va <- c(1, 2, 3)
vb <- c(4, 3, 0)
vc <- c(5, 1, 1)

Dot(va, Cross(vb, vc))
```

CrossN *n-dimensional Vector Cross Product*

Description

Vector cross product of `n-1` vectors in n-dimensional space

Usage

```
CrossN(A)
```

Arguments

`A`	matrix of size `(n-1) x n` where `n >= 2`.

Details

The rows of the matrix `A` are taken as`(n-1)` vectors in `n`-dimensional space. The cross product generates a vector in this space that is orthogonal to all these rows in `A` and its length is the volume of the geometric hypercube spanned by the vectors.

Value

a vector of length n

Note

The 'scalar triple product' in R^3 can be defined as

```
spatproduct <- function(a, b, c) Dot(a, CrossN(b, c))
```

It represents the volume of the parallelepiped spanned by the three vectors.

Author(s)

Hans W. Borchers <hwborchers@googlemail.com>

See Also

Cross, Dot

Examples

```
A <- matrix(c(1,0,0, 0,1,0), nrow=2, ncol=3, byrow=TRUE)
CrossN(A)  #=> 0 0 1

x <- c(1.0, 0.0, 0.0)
y <- c(1.0, 0.5, 0.0)
z <- c(0.0, 0.0, 1.0)
identical(Dot(x, CrossN(rbind(y, z))), det(rbind(x, y, z)))
```

Cstat *C Statistic (Area Under the ROC Curve)*

Description

Calculate the C statistic, a measure of goodness of fit for binary outcomes in a logistic regression or any other classification model. The C statistic is equivalent to the area under the ROC-curve (Receiver Operating Characteristic).

Usage

```
Cstat(x, ...)

## S3 method for class 'glm'
Cstat(x, ...)

## Default S3 method:
Cstat(x, resp, ...)
```

Arguments

x	the logistic model for the glm interface or the predicted probabilities of the model for the default.
resp	the response variable (coded as c(0, 1))
...	further arguments to be passed to other functions.

Details

Values for this measure range from 0.5 to 1.0, with higher values indicating better predictive models. A value of 0.5 indicates that the model is no better than chance at making a prediction of membership in a group and a value of 1.0 indicates that the model perfectly identifies those within a group and those not. Models are typically considered reasonable when the C-statistic is higher than 0.7 and strong when C exceeds 0.8.

Confidence intervals for this measure can be calculated by bootstrap.

Value

numeric value

Author(s)

Andri Signorell <andri@signorell.net>

References

Hosmer D.W., Lemeshow S. (2000) Applied Logistic Regression (2nd Edition). New York, NY: *John Wiley & Sons*

See Also

BrierScore

Examples

```
d.titanic = Untable(Titanic)
r.glm <- glm(Survived ~ ., data=d.titanic, family=binomial)
Cstat(r.glm)

# default interface
Cstat(x = predict(r.glm, method="response"),
      resp = model.response(model.frame(r.glm)))


# calculating bootstrap confidence intervals
FUN <- function(d.set, i) {
   r.glm <- glm(Survived ~ ., data=d.set[i,], family=binomial)
   Cstat(r.glm)
   }

## Not run:
```

```
library(boot)
boot.res <- boot(d.titanic, FUN, R=999)

# the percentile confidence intervals
boot.ci(boot.res, type="perc")

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = res, type = "perc")
##
## Intervals :
## Level     Percentile
## 95%   ( 0.7308,  0.7808 )
## Calculations and Intervals on Original Scale

## End(Not run)
```

`CstatCI` *Confidence Intervals for the C Statistic (AUC)*

Description

Calculate bootstrap intervals for the C statistic (Area under the curve AUC), based on a glm.

Usage

```
CstatCI(
  object,
  conf.level = 0.95,
  sides = c("two.sided", "left", "right"),
  ...
)
```

Arguments

`object`	the model object as returned by glm.
`conf.level`	confidence level of the interval.
`sides`	a character string specifying the side of the confidence interval, must be one of `"two.sided"` (default), `"left"` or `"right"`. `"left"` would be analogue to a hypothesis of `"greater"` in a `t.test`. You can specify just the initial letter.
`...`	further arguments are passed to the `boot` function. Supported arguments are `type` (`"norm"`, `"basic"`, `"stud"`, `"perc"`, `"bca"`), `parallel` and the number of bootstrap replicates `R`. If not defined those will be set to their defaults, being `"basic"` for `type`, option `"boot.parallel"` (and if that is not set, `"no"`) for `parallel` and `999` for `R`.

Value

a numeric vector with 3 elements:

mean	mean
lwr.ci	lower bound of the confidence interval
upr.ci	upper bound of the confidence interval

Author(s)

Andri Signorell andri@signorell.net

See Also

BrierScore

Examples

```
utils::data(Pima.te, package = "MASS")
r.logit <- glm(type ~ ., data=Pima.te, family="binomial")

# calculate Brier score with confidence intervals
Cstat(r.logit)
CstatCI(r.logit, R=99)  # use higher R in real life!
```

CutAge *Create a Factor Variable by Cutting an Age Variable*

Description

Dividing the range of an age variable x into intervals is a frequent task. The commonly used function cut has unfavourable default values for this. CutAge() is a convenient wrapper for cutting age variables in groups of e.g. 10 years with more suitable defaults.
CutGen yields the generation of a person based on the year of birth.

Usage

```
CutAge(x, from = 0, to = 90, by = 10, right = FALSE, ordered_result = TRUE, ...)
CutGen(vintage)
```

Arguments

x	continuous variable.
from, to	the starting and (maximal) end values of the sequence.
by	number: increment of the sequence. Default is 10, alternatives could be 5 or 20.
right	logical, indicating if the intervals should be closed on the right (and open on the left) or vice versa. Default is FALSE - unlike in cut!

ordered_result	logical: should the result be an ordered factor? Default is TRUE - unlike in cut!
...	the dots are passed on to the underlying function cut(). Use these for e.g. change the labels.
vintage	year of birth

Details

The generations are defined as:

1946-1964 Babyboomer
1965-1979 Generation X
1980-1995 Generation Y – also known as Millennials
1996-2010 Generation Z
2011-.... Generation Alpha

Value

A factor is returned, unless labels = FALSE which results in an integer vector of level codes.

Values which fall outside the range of breaks are coded as NA, as are NaN and NA values.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

cut, seq

Examples

```
Desc(CutAge(sample(100, 100)))
```

CutQ *Create a Factor Variable Using the Quantiles of a Continuous Variable*

Description

Create a factor variable using the quantiles of a continous variable.

Usage

```
CutQ(x, breaks = quantile(x, seq(0, 1, by = 0.25), na.rm = TRUE),
     labels = NULL, na.rm = FALSE, ...)
```

Arguments

x	continous variable.
breaks	the breaks for creating groups. By default the quartiles will be used, say `quantile seq(0, 1, by = 0.25)` quantiles. See quantile for details. If breaks is given as a single integer it is interpreted as the intended number of groups, e.g. `breaks=10` will return x cut in deciles.
labels	labels for the levels of the resulting category. By default, labels are defined as Q1, Q2 to the length of breaks - 1. The parameter ist passed to cut, so if `labels` are set to FALSE, simple integer codes are returned instead of a factor.
na.rm	Boolean indicating whether missing values should be removed when computing quantiles. Defaults to TRUE.
...	Optional arguments passed to cut.

Details

This function uses quantile to obtain the specified quantiles of x, then calls cut to create a factor variable using the intervals specified by these quantiles.

It properly handles cases where more than one quantile obtains the same value, as in the second example below. Note that in this case, there will be fewer generated factor levels than the specified number of quantile intervals.

Value

Factor variable with one level for each quantile interval given by q.

Author(s)

Gregory R. Warnes <greg@warnes.net>, some slight modifications Andri Signorell <andri@signorell.net>

See Also

cut, quantile

Examples

```
# create example data

x <- rnorm(1000)

# cut into quartiles
quartiles <- CutQ(x)
table(quartiles)

# cut into deciles
deciles <- CutQ(x, breaks=10, labels=NULL)
table(deciles)

# show handling of 'tied' quantiles.
x <- round(x)  # discretize to create ties
```

```
stem(x)        # display the ties
deciles <- CutQ(x, breaks=10)

table(deciles) # note that there are only 5 groups (not 10)
               # due to duplicates
```

`d.countries` *ISO 3166-1 Country Codes*

Description

Country codes published by the International Organization for Standardization (ISO) define codes for the names of countries, dependent territories, and special areas of geographical interest.

Usage

```
data("d.countries")
```

Format

A data frame with 249 observations on the following 4 variables.

`name` a character vector, the name of the country.

`a2` a character vector, two-letter country codes (aka alpha-2) which are the most widely used of the three, and used most prominently for the Internet's country code top-level domains (with a few exceptions).

`a3` a character vector, three-letter country codes (aka alpha-3) which allow a better visual association between the codes and the country names than the alpha-2 codes.

`code` a numeric vector, three-digit country codes which are identical to those developed and maintained by the United Nations Statistics Division, with the advantage of script (writing system) independence, and hence useful for people or systems using non-Latin scripts.

`region` the region of the country. One of "East Asia & Pacific" (35), "Europe & Central Asia" (52), "Latin America & Caribbean" (41), "Middle East & North Africa" (20), "North America" (3), "South Asia" (8), "Sub-Saharan Africa" (47)

`pop2012` the population in 2012

`gcpi2012` the gross national income (per capita) in dollars per country in 2012.

`latitude` geographic coordinate that specifies the north–south position of a point on the Earth's surface. Latitude is an angle (defined below) which ranges from 0° at the Equator to 90° (North or South) at the poles.

`longitude` geographic coordinate that specifies the east–west position of a point on the Earth's surface, or the surface of a celestial body

References

https://en.wikipedia.org/wiki/ISO_3166-1
https://datacatalog.worldbank.org/search/dataset/0037652

Examples

```
head(d.countries)
```

`d.diamonds` *Data diamonds*

Description

As I suppose, an artificial dataset

Usage

```
data(d.diamonds)
```

Format

A data frame with 440 observations on the following 10 variables.

`index` a numeric vector

`carat` a numeric vector

`colour` a factor with levels `D E F G H I J K L`

`clarity` an ordered factor with levels `I2 < I1 < SI3 < SI2 < SI1 < VS2 < VS1 < VVS2 < VVS1`

`cut` an ordered factor with levels `F < G < V < X < I`

`certification` a factor with levels `AGS DOW EGL GIA IGI`

`polish` an ordered factor with levels `F < G < V < X < I`

`symmetry` an ordered factor with levels `F < G < V < X < I`

`price` a numeric vector

`wholesaler` a factor with levels `A B C`

Details

P Poor F Fair G Good V Very good X Excellent I Ideal

Source

somewhere from the net...

Examples

```
data(d.diamonds)
str(d.diamonds)
```

d.periodic *Periodic Table of Elements*

Description

This data.frame contains the most important properties of the periodic table of the elements.

Usage

```
data(d.periodic)
```

Format

A data frame with 110 observations on the following 24 variables.

`symbol` symbol of an element.

`nr` atomic number of an atomic symbol.

`name` name of an element.

`group` group of an element. Possible results are: Alkali Earth, Alkali Met., Halogen, Metal, Noble Gas, Non-Metal, Rare Earth and Trans. Met.

`weight` atomic weight of an element. The values are based upon carbon-12. () indicates the most stable or best known isotope.

`meltpt` melting point of an element in [K].

`boilpt` boiling point of an element in Kelvin [K].

`dens` density of an element in [g/cm3] at 300K and 1 atm.

`elconf` electron configuration of an element.

`oxstat` oxidation states of an element. The most stable is indicated by a "!".

`struct` crystal structure of an element. Possible results are: Cubic, Cubic body centered, Cubic face centered, Hexagonal, Monoclinic, Orthorhombic, Rhombohedral, Tetragonal

`covrad` covalent radius of an element in Angstroem [A].

`arad` atomic radius of an element in Angstroem.

`avol` atomic volume of an element in [cm3/mol].

`spheat` specific heat of an element in [J/(g K)].

`eneg` electronegativity (Pauling's) of an element.

`fusheat` heat of fusion of an element in [kJ/mol].

`vapheat` heat of vaporization of an element in [kJ/mol].

`elcond` electrical conductivity of an element in [1/(Ohm cm].

`thermcond` thermal conductivity of an element in [W/(cm K)].

`ionpot1` first ionization potential of an element in [V].

`ionpot2` second ionization potential of an element in [V].

`ionpot3` third ionization potential of an element in [V].

`discyear` year of discovery of the element

References

https://en.wikipedia.org/wiki/Periodic_table

`d.pizza` *Data pizza*

Description

An artificial dataset inspired by a similar dataset pizza.sav in *Arbeitsbuch zur deskriptiven und induktiven Statistik* by Toutenburg et.al.
The dataset contains data of a pizza delivery service in London, delivering pizzas to three areas. Every record defines one order/delivery and the according properties. A pizza is supposed to taste good, if its temperature is high enough, say 45 Celsius. So it might be interesting for the pizza delivery service to minimize the delivery time.
The dataset is designed to be as evil as possible. As far as the description is concerned, it should pose the same difficulties that we have to deal with in everyday life. It contains the most used datatypes as numerics, factors, ordered factors, integers, logicals and a date. NAs are scattered everywhere partly systematically, partly randomly (except in the index).

Usage

```
data(d.pizza)
```

Format

A data frame with 1209 observations on the following 17 variables.

`index` a numeric vector, indexing the records (no missings here).

`date` Date, the delivery date

`week` integer, the weeknumber

`weekday` integer, the weekday

`area` factor, the three London districts: `Brent`, `Camden`, `Westminster`

`count` integer, the number of pizzas delivered

`rabate` logical, `TRUE` if a rabate has been given

`price` numeric, the total price of delivered pizza(s)

`operator` a factor with levels `Allanah` `Maria` `Rhonda`

`driver` a factor with levels `Carpenter` `Carter` `Taylor` `Butcher` `Hunter` `Miller` `Farmer`

`delivery_min` numeric, the delivery time in minutes (decimal)

`temperature` numeric, the temperature of the pizza in degrees Celsius when delivered to the customer

`wine_ordered` integer, 1 if wine was ordered, 0 if not

`wine_delivered` integer, 1 if wine was delivered, 0 if not

`wrongpizza` logical, `TRUE` if a wrong pizza was delivered

`quality` ordered factor with levels `low` < `medium` < `high`, defining the quality of the pizza when delivered

Details

The dataset contains NAs randomly scattered.

References

Toutenburg H, Schomaker M, Wissmann M, Heumann C (2009): *Arbeitsbuch zur deskriptiven und induktiven Statistik* Springer, Berlin Heidelberg

Examples

```
str(d.pizza)
head(d.pizza)

Desc(d.pizza)
```

---

d.whisky *Classification of Scotch Single Malts*

Description

86 malt whiskies are scored between 0-4 for 12 different taste categories including sweetness, smoky, nutty etc. Additionally, coordinates of distilleries allow us to obtain pairwise distance information. Using a combination of these variables it is possible to look for correlations between particular attributes of taste and physical location, for example does a shared local resource have a significant effect on nearby whiskies.
By using correlation analysis it may be possible to provide whisky recommendations based upon an individual's particular preferences. By computing the Pearson correlation coefficient and specifying a threshold value between 0 and 1, we can establish an adjacency matrix where each node is a malt whisky and an edge represents a level of similarity above the threshold.

Usage

```
data("d.whisky")
```

Format

A data frame with 86 observations on the following 16 variables.

`distillery` a character `Aberfeldy`, `Aberlour`, `AnCnoc`, `Ardbeg`, ...

`brand` a grouping factor to separate the better known distilleries (`A`) from the lesser known ones (`B`).

`region` a factor with levels `campbeltown`, `highland`, `islands`, `islay`, `lowland`, `speyside`.

`body` a numeric vector

`sweetness` a numeric vector

`smoky` a numeric vector

`medicinal` a numeric vector

`tobacco` a numeric vector

`honey` a numeric vector

`spicy` a numeric vector

`winey` a numeric vector

`nutty` a numeric vector

`malty` a numeric vector

`fruity` a numeric vector

`floral` a numeric vector

`postcode` a character `AB30 1YE`, `AB35 5TB`, ...

`latitude` a numeric vector, coordinate pairs of distilleries.

`longitude` a numeric vector, coordinate pairs of distilleries.

Source

http://www.mathstat.strath.ac.uk/outreach/nessie/nessie_whisky.html

References

http://www.mathstat.strath.ac.uk/outreach/nessie/index.html

Examples

```
head(d.whisky)

opar <- par(mfrow=c(3,3), cex.main=1.8)
for(i in 1:9)
  PlotPolar(d.whisky[i, 4:15], rlim=4, type="l", col=DescTools::hecru,
            lwd=2, fill=SetAlpha(DescTools::hecru, 0.4),
            panel.first=PolarGrid(
              ntheta=ncol(d.whisky[i, 2:13]), nr = NA, col="grey",
              lty="dotted", las=1, cex=1.4, alabels=StrCap(colnames(d.whisky[i, 3:14])),
              lblradians=TRUE),
            main=d.whisky[i, "distillery"])


par(mfrow=c(3,3), cex.main=1.8, xpd=NA)
id <- d.whisky$distillery %in% c("Ardbeg","Caol Ila","Cragganmore","Lagavulin","Laphroig",
                                 "Macallan","Mortlach","Talisker","Tobermory")
PlotFaces(d.whisky[id, 4:15], nr=3, nc=3, col=hecru, scale=TRUE, fill=TRUE,
          labels=d.whisky$distillery[id])

par(opar)
```

Datasets for Simulation

Datasets for Probabilistic Simulation

Description

For performing elementary probability calculations in introductory statistic courses, we might want to simulate random games. The dataset `roulette` contains the standard sample space for one spin on a roulette wheel. `cards` contains the standard set of 52 playing cards in four colours (without Jokers). `tarot` does the same with a classic tarot deck.

Usage

```
cards
tarot
roulette
```

Value

`cards` is a `data.frame` with three columns named `card`, `rank` and `suit`

`tarot` is a `data.frame` with four columns named `card`, `rank`, `suit` and `desc`

`roulette` is a `data.frame` with seven columns named `num` and `col`, `parity`, `highlow`, `dozens`, `column`, `pocketrange`

See Also

`Sample`, `sample()`

Examples

```
head(cards)
head(tarot)
head(roulette)

# drawing 5 cards
sample(cards$card, 5)

# drawing 5 cards with jokers
sample(c(cards$card, rep("Joker", 3)), 5)

# spin the wheel by using the DescTools::Sample() for sampling
# rows from a data frame
Sample(roulette, size=1)

# simulate the evening in Las Vegas with 10 games
Sample(roulette, 10, replace=TRUE)
```

Date Functions *Basic Date Functions*

Description

Some more date functions for making daily life a bit easier. The first ones extract a specific part of a given date, others check some conditions.

Usage

```
Year(x)
Quarter(x)
Month(x, fmt = c("m", "mm", "mmm"), lang = DescToolsOptions("lang"),
      stringsAsFactors = TRUE)
Week(x, method = c("iso", "us"))
Day(x)
Weekday(x, fmt = c("d", "dd", "ddd"), lang = DescToolsOptions("lang"),
        stringsAsFactors = TRUE)
YearDay(x)
YearWeek(x, method = c("iso", "us"))
YearMonth(x)

Day(x) <- value

IsWeekend(x)
IsLeapYear(x)

Hour(x)
Minute(x)
Second(x)
Timezone(x)
HmsToMinute(x)

Now()
Today()

DiffDays360(start_d, end_d, method = c("eu", "us"))
LastDayOfMonth(x)
YearDays(x)
MonthDays(x)
```

Arguments

x	the date to be evaluated.
fmt	format string, defines how the month or the weekday are to be formatted. Defaults to "m", resp. "d". Is ignored for other functions.
value	new value

`lang`	optional value setting the language for the months and daynames. Can be either `"local"` for current locale or `"engl"` for english. If left to `NULL`, the option `"lang"` will be searched for and if not found `"local"` will be taken as default.
`stringsAsFactors`	logical. Defines if the result should be coerced to a factor, using the local definitions as levels. The result would be an ordered factor. Default is TRUE.
`start_d, end_d`	the start, resp. end date for `DiffDays360`.
`method`	one out of `"eu", "us"`, setting either European or US-Method calculation mode. Default is `"eu"`.

Details

These functions are mainly convenience wrappers for the painful `format()` and its strange codes... Based on the requested time component, the output is as follows:

`Year` returns the year of the input date in yyyy format or a yearmonth yyyymm.
`Quarter` returns the quarter of the year (1 to 4) for the input date.
`Month` returns the month of the year (1 to 12) for the input date or for a yearmonth yyyymm.
`Week` returns the week of the year for the input date (0 to 53), as defined in ISO8601.
`Weekday` returns the week day of the input date. (1 - Monday, 2 - Tuesday, ... 7 - Sunday). (Names and abbreviations are either english or in the current locale!)
`YearDay` returns the day of the year numbering (1 to 366).
`Day` returns the day of the month (1 to 31).
`YearDay` returns the yearday representation (yyyyddd) of a date as long integer.
`YearWeek` returns the yearweek representation (yyyyww) of a date as long integer.
`YearMonth` returns the yearmonth representation (yyyymm) of a date as long integer.
`Hour`, `Minute`, `Second`, `Timezone` return the hour, minute, second or timezone from a POSIXlt object.
`HmsToMinute` converts the time parts of a POSIXlt object to minutes.
`Today`, `Now` return the current date, resp. the current date and time.

`IsWeekend` returns `TRUE`, if the date x falls on a weekend.
`IsLeapYear` returns `TRUE`, if the year of the date x is a leap year.

The day can not only be extracted, but as well be defined. See examples.

`DiffDays360` calculates the difference between 2 dates using the 360-days convention.
`LastDayOfMonth` returns the last day of the month of the given date(s). `YearDays` returns the total number of days of the given date(s). `MonthDays` returns the numer of days of the month of the given date(s).

The language in Weekday and Moth can be set with an option as well. The functions will check for an existing option named "lang" and take this value if it exists. So simply set option(lang="engl") if the results should always be reported in English.

Value

a vector of the same dimension as x, consisting of either numeric values or characters depending on the function used.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

strptime, DateTimeClasses, as.POSIXlt

Examples

```
x <- Today()    # the same as Sys.Date() but maybe easier to remember..

Year(x)
Quarter(x)

Month(x)
Month(x, fmt = "mm", lang="engl")
Month(x, fmt = "mm", lang="local")
Month(x, fmt = "mmm", lang="engl")
Month(x, fmt = "mmm", lang="local")

Week(x)

Day(x)
Day(x) <- 20
x

Weekday(x)
Weekday(x, fmt = "dd", lang="engl")
Weekday(x, fmt = "dd", lang="local")
Weekday(x, fmt = "ddd", lang="engl")
Weekday(x, fmt = "ddd", lang="local")

YearDay(x)

IsWeekend(x)

IsLeapYear(x)

# let's generate a time sequence by weeks
Month(seq(from=as.Date(Sys.Date()), to=Sys.Date()+150, by="weeks"), fmt="mm")

LastDayOfMonth(as.Date(c("2014-10-12","2013-01-31","2011-12-05")))
```

---

day.name *Build-in Constants Extension*

Description

There's a small number of built-in constants in R. We have month.name and month.abb but nothing similar for weekdays. Here it is.

Usage

```
day.name
day.abb
```

Details

The following constants are available in DescTools:

- `day.name`: the English names for the day of the week (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday);
- `day.abb`: the three-letter abbreviations for the English day names (Mon, Tue, Wed, Thu, Fri, Sat, Sun);

See Also

`month.name`, `month.abb`

DegToRad *Convert Degrees to Radians and Vice Versa*

Description

Convert degrees to radians (and back again).

Usage

```
DegToRad(deg)
RadToDeg(rad)
```

Arguments

`deg`	a vector of angles in degrees.
`rad`	a vector of angles in radians.

Value

DegToRad returns a vector of the same length as `deg` with the angles in radians.
RadToDeg returns a vector of the same length as `rad` with the angles in degrees.

Author(s)

Andri Signorell <andri@signorell.net>

Examples

```
DegToRad(c(90,180,270))
RadToDeg( c(0.5,1,2) * pi)
```

Depreciation *Several Methods of Depreciation of an Asset*

Description

Return the depreciation of an asset for a specified period using different methods. SLN returns the straight-line depreciation DB uses the fixed-declining balance method and SYD returns the sum-of-years' digits depreciation.

Usage

```
SLN(cost, salvage, life)
DB(cost, salvage, life, period = 1:life)
SYD(cost, salvage, life, period = 1:life)
```

Arguments

cost	initial cost of the asset.
salvage	value at the end of the depreciation (sometimes called the salvage value of the asset).
life	number of periods over which the asset is depreciated (sometimes called the useful life of the asset).
period	period for which you want to calculate the depreciation. Period must use the same units as life.

Value

val

Author(s)

Andri Signorell <andri@signorell.net>

See Also

NPV()

Examples

```
# depreciation allowance for each year
SLN(cost = 50000, salvage = 10000, life = 5)
DB(cost = 50000, salvage = 10000, life = 5)

50000 - c(0, cumsum(SYD(cost = 50000, salvage = 10000, life = 5)))
```

Desc *Describe Data*

Description

Produce summaries of various types of variables. Calculate descriptive statistics for x and use Word as reporting tool for the numeric results and for descriptive plots. The appropriate statistics are chosen depending on the class of x. The general intention is to simplify the description process for lazy typers and return a quick, but rich summary.

Usage

```
Desc(x, ..., main = NULL, plotit = NULL, wrd = NULL)

## S3 method for class 'numeric'
Desc(
  x,
  main = NULL,
  maxrows = NULL,
  plotit = NULL,
  sep = NULL,
  digits = NULL,
  ...
)

## S3 method for class 'integer'
Desc(
  x,
  main = NULL,
  maxrows = NULL,
  plotit = NULL,
  sep = NULL,
  digits = NULL,
  ...
)

## S3 method for class 'factor'
Desc(
  x,
  main = NULL,
  maxrows = NULL,
  ord = NULL,
  plotit = NULL,
  sep = NULL,
  digits = NULL,
  ...
)
```

```
## S3 method for class 'labelled'
Desc(
  x,
  main = NULL,
  maxrows = NULL,
  ord = NULL,
  plotit = NULL,
  sep = NULL,
  digits = NULL,
  ...
)

## S3 method for class 'ordered'
Desc(
  x,
  main = NULL,
  maxrows = NULL,
  ord = NULL,
  plotit = NULL,
  sep = NULL,
  digits = NULL,
  ...
)

## S3 method for class 'character'
Desc(
  x,
  main = NULL,
  maxrows = NULL,
  ord = NULL,
  plotit = NULL,
  sep = NULL,
  digits = NULL,
  ...
)

## S3 method for class 'ts'
Desc(x, main = NULL, plotit = NULL, sep = NULL, digits = NULL, ...)

## S3 method for class 'logical'
Desc(
  x,
  main = NULL,
  ord = NULL,
  conf.level = 0.95,
  plotit = NULL,
  sep = NULL,
```

```
  digits = NULL,
  ...
)

## S3 method for class 'Date'
Desc(
  x,
  main = NULL,
  dprobs = NULL,
  mprobs = NULL,
  plotit = NULL,
  sep = NULL,
  digits = NULL,
  ...
)

## S3 method for class 'table'
Desc(
  x,
  main = NULL,
  conf.level = 0.95,
  verbose = 2,
  rfrq = "111",
  margins = c(1, 2),
  plotit = NULL,
  sep = NULL,
  digits = NULL,
  ...
)

## Default S3 method:
Desc(
  x,
  main = NULL,
  maxrows = NULL,
  ord = NULL,
  conf.level = 0.95,
  verbose = 2,
  rfrq = "111",
  margins = c(1, 2),
  dprobs = NULL,
  mprobs = NULL,
  plotit = NULL,
  sep = NULL,
  digits = NULL,
  ...
)
```

```
## S3 method for class 'data.frame'
Desc(x, main = NULL, plotit = NULL, enum = TRUE, sep = NULL, ...)

## S3 method for class 'list'
Desc(x, main = NULL, plotit = NULL, enum = TRUE, sep = NULL, ...)

## S3 method for class 'formula'
Desc(
  formula,
  data = parent.frame(),
  subset,
  main = NULL,
  plotit = NULL,
  digits = NULL,
  ...
)

## S3 method for class 'Desc'
print(
  x,
  digits = NULL,
  plotit = NULL,
  nolabel = FALSE,
  sep = NULL,
  nomain = FALSE,
  ...
)

## S3 method for class 'Desc'
plot(x, main = NULL, ...)

## S3 method for class 'palette'
Desc(x, ...)
```

Arguments

x	the object to be described. This can be a data.frame, a list, a table or a vector of the classes: numeric, integer, factor, ordered factor, logical.
...	further arguments to be passed to or from other methods. For the internal default method these can include: p a vector of probabilities of the same length of x. An error is given if any entry of p is negative. This argument will be passed on to chisq.test(). Default is rep(1/length(x), length(x)). add_ni logical. Indicates if the group length should be displayed in the boxplot. smooth character, either "loess" or "smooth.spline" defining the type of smoother to be used in num ~ num plots. Default is "loess" for n < 500 and "smooth.spline" otherwise.
main	(character|NULL|NA), the main title(s).

- If NULL, the title will be composed as:
  - variable name (class(es)),
  - resp. number - variable name (class(es)) if the enum option is set to TRUE.
- Use NA if no caption should be printed at all.

plotit — logical. Should a plot be created? The plot type will be chosen according to the classes of variables (roughly following a numeric-numeric, numeric-categorical, categorical-categorical logic). Default can be defined by option plotit, if it does not exist then it's set to FALSE.

wrd — the pointer to a running MS Word instance, as created by GetNewWrd() (for a new one) or by GetCurrWrd() for an existing one. All output will then be redirected there. Default is NULL, which will report all results to the console.

maxrows — numeric; defines the maximum number of rows in a frequency table to be reported. For factors with many levels it is often not interesting to see all of them. Default is set to 12 most frequent ones (resp. the first ones if ord is set to "levels" or "names").

For a numeric argument x maxrows is the minimum number of unique values needed for a numeric variable to be treated as continuous. If left to its default NULL, x will be regarded as continuous if it has more than 12 single values. In this case the list of extreme values will be displayed and the frequency table else.

If maxrows is < 1 it will be interpreted as percentage. In this case just as many rows, as the maxrows most frequent levels will be shown. Say, if maxrows is set to 0.8, then the number of rows is fixed so, that the highest cumulative relative frequency is the first one going beyond 0.8.

Setting maxrows to Inf will unconditionally report all values and also produce a plot with type "h" instead of a histogram.

sep — character. The separator for the title. By default a line of "-" for the current width of the screen (options("width")) will be used.

digits — integer. With how many digits should the relative frequencies be formatted? Default can be set by DescToolsOptions(digits=x).

ord — character out of "name" (alphabetical order), "level", "asc" (by frequencies ascending), "desc" (by frequencies descending) defining the order for a frequency table as used for factors, numerics with few unique values and logicals. Factors (and character vectors) are by default ordered by their descending frequencies, ordered factors by their natural order.

conf.level — confidence level of the interval. If set to NA no confidence interval will be calculated. Default is 0.95.

dprobs, mprobs — a vector with the probabilities for the Chi-Square test for days, resp. months, when describing a Date variable. If this is left to NULL (default) then a uniform distribution will be used for days and a monthdays distribution in a non leap year (p = c(31/365, 28/365, 31/365, ...)) for the months.
Applies only to Dates and is ignored else.

verbose — integer out of c(2, 1, 3) defining the verbosity of the reported results. 2 (default) means medium, 1 less and 3 extensive results.
Applies only to tables and is ignored else.

rfrq	a string with 3 characters, each of them being 1 or 0, defining which percentages should be reported. The first position is interpreted as total percentages, the second as row percentages and the third as column percentages. "011" hence produces a table output with row and column percentages. If set to NULL rfrq is defined in dependency of verbose (verbose = 1 sets rfrq to "000" and else to "111", latter meaning all percentages will be reported.) Applies only to tables and is ignored else.
margins	a vector, consisting out of 1 and/or 2. Defines the margin sums to be included. Row margins are reported if margins is set to 1. Set it to 2 for column margins and c(1,2) for both. Default is NULL (none). Applies only to tables and is ignored else.
enum	logical, determining if in data.frames and lists a sequential number should be included in the main title. Default is TRUE. The reason for this option is, that if a Word report with enumerated headings is created, the numbers may be redundant or inconsistent.
formula	a formula of the form lhs ~ rhs where lhs gives the data values and rhs the corresponding groups.
data	an optional matrix or data frame containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset	an optional vector specifying a subset of observations to be used.
nolabel	logical, defining if labels (defined as attribute with the name label, as done by Label) should be plotted.
nomain	logical, determines if the main title of the output is printed or not, default is TRUE.

Details

A **2-dimensional table** will be described with it's relative frequencies, a short summary containing the total cases, the dimensions of the table, chi-square tests and some association measures as phi-coefficient, contingency coefficient and Cramer's V.
Tables with higher dimensions will simply be printed as flat table, with marginal sums for the first and for the last dimension.

Desc is a **generic function**. It dispatches to one of the methods above depending on the class of its first argument. Typing ?Desc + TAB at the prompt should present a choice of links: the help pages for each of these Desc methods (at least if you're using RStudio, which anyway is recommended). You don't need to use the full name of the method although you may if you wish; i.e., Desc(x) is idiomatic R but you can bypass method dispatch by going direct if you wish: Desc.numeric(x).

This function produces a rich description of a **factor**, containing length, number of NAs, number of levels and detailed frequencies of all levels. The order of the frequency table can be chosen between descending/ascending frequency, labels or levels. For ordered factors the order default is "level". Character vectors are treated as unordered factors Desc.char converts x to a factor an processes x as factor.
Desc.ordered does nothing more than changing the standard order for the frequencies to it's intrinsic order, which means order "level" instead of "desc" in the factor case.

Description interface for **dates**. We do here what seems reasonable for describing dates. We start with a short summary about length, number of NAs and extreme values, before we describe the frequencies of the weekdays and months, rounded up by a chi-square test.

A **2-dimensional table** will be described with it's relative frequencies, a short summary containing the total cases, the dimensions of the table, chi-square tests and some association measures as phi-coefficient, contingency coefficient and Cramer's V.
Tables with higher dimensions will simply be printed as flat table, with marginal sums for the first and for the last dimension.

Note that `NA`s cannot be handled by this interface, as tables in general come in "as.is", say basically as a matrix without any further information about potentially previously cleared NAs.

Description of a **dichotomous variable**. This can either be a logical vector, a factor with two levels or a numeric variable with only two unique values. The confidence levels for the relative frequencies are calculated by `BinomCI()`, method `"Wilson"` on a confidence level defined by `conf.level`. Dichotomous variables can easily be condensed in one graphical representation. Desc for a set of flags (=dichotomous variables) calculates the frequencies, a binomial confidence interval and produces a kind of dotplot with error bars. Motivation for this function is, that dichotomous variable in general do not contain intense information. Therefore it makes sense to condense the description of sets of dichotomous variables.

The **formula interface** accepts the formula operators `+`, `:`, `*`, `I()`, `1` and evaluates any function. The left hand side and right hand side of the formula are evaluated the same way. The variable pairs are processed in dependency of their classes.

`Word` This function is not thought of being directly run by the end user. It will normally be called automatically, when a pointer to a Word instance is passed to the function `Desc()`.
However `DescWrd` takes some more specific arguments concerning the Word output (like `font` or `fontsize`), which can make it necessary to call the function directly.

Value

A list containing the following components:

`length`	the length of the vector (n + NAs).
`n`	the valid entries (NAs are excluded)
`NAs`	number of NAs
`unique`	number of unique values.
`0s`	number of zeros
`mean`	arithmetic mean
`MeanSE`	standard error of the mean, as calculated by `MeanSE()`.
`quant`	a table of quantiles, as calculated by quantile(x, probs = c(.05,.10,.25,.5,.75,.9,.95), na.rm = TRUE).
`sd`	standard deviation
`vcoef`	coefficient of variation: `mean(x)` / `sd(x)`.
`mad`	median absolute deviation (`stats::mad()`).
`IQR`	interquartile range
`skew`	skewness, as calculated by `Skew()`.

kurt	kurtosis, as calculated by Kurt().
highlow	the lowest and the highest values, reported with their frequencies in brackets, if > 1.
frq	a data.frame of absolute and relative frequencies given by Freq() if maxlevels > unique values in the vector.

Author(s)

Andri Signorell andri@signorell.net

See Also

base::summary(), base::plot()

Other Statistical summary functions: Abstract()

Examples

```
opt <- DescToolsOptions()

# implemented classes:
Desc(d.pizza$wrongpizza)              # logical
Desc(d.pizza$driver)                  # factor
Desc(d.pizza$quality)                 # ordered factor
Desc(as.character(d.pizza$driver))    # character
Desc(d.pizza$week)                    # integer
Desc(d.pizza$delivery_min)            # numeric
Desc(d.pizza$date)                    # Date

Desc(d.pizza)

Desc(d.pizza$wrongpizza, main="The wrong pizza delivered", digits=5)

Desc(table(d.pizza$area))                                    # 1-dim table
Desc(table(d.pizza$area, d.pizza$operator))                  # 2-dim table
Desc(table(d.pizza$area, d.pizza$operator, d.pizza$driver))  # n-dim table

# expressions
Desc(log(d.pizza$temperature))
Desc(d.pizza$temperature > 45)

# supported labels
Label(d.pizza$temperature) <- "This is the temperature in degrees Celsius
measured at the time when the pizza is delivered to the client."
Desc(d.pizza$temperature)
# try as well:      Desc(d.pizza$temperature, wrd=GetNewWrd())

z <- Desc(d.pizza$temperature)
print(z, digits=1, plotit=FALSE)
# plot (additional arguments are passed on to the underlying plot function)
plot(z, main="The pizza's temperature in Celsius", args.hist=list(breaks=50))
```

```
# formula interface for single variables
Desc(~ uptake + Type, data = CO2, plotit = FALSE)

# bivariate
Desc(price ~ operator, data=d.pizza)                  # numeric ~ factor
Desc(driver ~ operator, data=d.pizza)                 # factor ~ factor
Desc(driver ~ area + operator, data=d.pizza)          # factor ~ several factors
Desc(driver + area ~ operator, data=d.pizza)          # several factors ~ factor
Desc(driver ~ week, data=d.pizza)                     # factor ~ integer

Desc(driver ~ operator, data=d.pizza, rfrq="111")   # alle rel. frequencies
Desc(driver ~ operator, data=d.pizza, rfrq="000",
     verbose=3)                                    # no rel. frequencies

Desc(price ~ delivery_min, data=d.pizza)              # numeric ~ numeric
Desc(price + delivery_min ~ operator + driver + wrongpizza,
     data=d.pizza, digits=c(2,2,2,2,0,3,0,0) )

Desc(week ~ driver, data=d.pizza, digits=c(2,2,2,2,0,3,0,0))   # define digits

Desc(delivery_min + weekday ~ driver, data=d.pizza)


# without defining data-parameter
Desc(d.pizza$delivery_min ~ d.pizza$driver)


# with functions and interactions
Desc(sqrt(price) ~ operator : factor(wrongpizza), data=d.pizza)
Desc(log(price+1) ~ cut(delivery_min, breaks=seq(10,90,10)),
     data=d.pizza, digits=c(2,2,2,2,0,3,0,0))

# response versus all the rest
Desc(driver ~ ., data=d.pizza[, c("temperature","wine_delivered","area","driver")])

# all the rest versus response
Desc(. ~ driver, data=d.pizza[, c("temperature","wine_delivered","area","driver")])

# pairwise Descriptions
p <- CombPairs(c("area","count","operator","driver","temperature","wrongpizza","quality"), )
for(i in 1:nrow(p))
  print(Desc(formula(gettextf("%s ~ %s", p$X1[i], p$X2[i])), data=d.pizza))


# get more flexibility, create the table first
tab <- as.table(apply(HairEyeColor, c(1,2), sum))
tab <- tab[,c("Brown","Hazel","Green","Blue")]

# display only absolute values, row and columnwise percentages
Desc(tab, row.vars=c(3, 1), rfrq="011", plotit=FALSE)

# do the plot by hand, while setting the colours for the mosaics
```

```
cols1 <- SetAlpha(c("sienna4", "burlywood", "chartreuse3", "slategray1"), 0.6)
cols2 <- SetAlpha(c("moccasin", "salmon1", "wheat3", "gray32"), 0.8)
plot(Desc(tab), col1=cols1, col2=cols2)

# choose alternative flavours for graphing numeric ~ factor using pipe
# (colors are recyled)
Desc(temperature ~ driver, data = d.pizza) |> plot(type="dens", col=Pal("Tibco"))


# use global format options for presentation
Fmt(abs=as.fmt(digits=0, big.mark=""))
Fmt(per=as.fmt(digits=2, fmt="%"))
Desc(area ~ driver, d.pizza, plotit=FALSE)

Fmt(abs=as.fmt(digits=0, big.mark="'"))
Fmt(per=as.fmt(digits=3, ldigits=0))
Desc(area ~ driver, d.pizza, plotit=FALSE)

# plot arguments can be fixed in detail
z <- Desc(BoxCox(d.pizza$temperature, lambda = 1.5))
plot(z, mar=c(0, 2.1, 4.1, 2.1), args.rug=TRUE, args.hist=list(breaks=50),
     args.dens=list(from=0))

# The default description for count variables can be inappropriate,
# the density curve does not represent the variable well.
set.seed(1972)
x <- rpois(n = 500, lambda = 5)
Desc(x)
# but setting maxrows to Inf gives a better plot
Desc(x, maxrows = Inf)


# Output into word document (Windows-specific example) -----------------------
# by simply setting wrd=GetNewWrd()
## Not run:

  # create a new word instance and insert title and contents
  wrd <- GetNewWrd(header=TRUE)

  # let's have a subset
  d.sub <- d.pizza[,c("driver", "date", "operator", "price", "wrongpizza")]

  # do just the univariate analysis
  Desc(d.sub, wrd=wrd)

## End(Not run)

DescToolsOptions(opt)
```

DescTools Aliases *Some Aliases Set for Convenience*

Description

Some aliases are defined either for having shorter names or for following the Google naming convention.

Usage

```
N()
```

Details

`N()` is the same as `as.numeric()`.
`D()` is the same as `as.Date()`

Author(s)

Andri Signorell <andri@signorell.net>

Examples

```
head(N(d.pizza$driver))
```

DescTools Palettes *Some Custom Palettes*

Description

Some more custom palettes.

Usage

```
Pal(pal, n = 100, alpha = 1)

## S3 method for class 'palette'
plot(x, cex = 3, ...)

hred
horange
hyellow
hecru
hblue
hgreen
```

Arguments

pal	name or number of the palette. One of RedToBlack (1), RedBlackGreen (2), SteeblueWhite (3), RedWhiteGreen (4), RedWhiteBlue0 (5), RedWhiteBlue1 (6), RedWhiteBlue2 (7), RedWhiteBlue3 (8), Helsana (9), Tibco (10), RedGreen1 (11), Spring (12), Soap (13), Maiden (14), Dark (15), Accent (16), Pastel (17), Fragile (18), Big (19), Long (20), Night (21), Dawn (22), Noon (23), Light (24)
n	integer, number of colors for the palette.
alpha	the alpha value to be added. This can be any value from 0 (fully transparent) to 1 (opaque). NA is interpreted so as to delete a potential alpha channel. Default is 0.5.
x	a palette to be plotted.
cex	extension for the color squares. Defaults to 3.
...	further arguments passed to the function.

Details

hred, horange, hyellow, hecru, hblue and hgreen are constants, pointing to the according color from the palette Pal("Helsana").

Value

a vector of colors

Author(s)

Andri Signorell <andri@signorell.net>

See Also

colorRampPalette

Examples

```
Canvas(c(0,1))
ColorLegend(x=0, y=1, width=0.1, col=Pal(1, n=50))
ColorLegend(x=0.15, y=1, width=0.1, col=Pal(2, n=50))
ColorLegend(x=0.3, y=1, width=0.1, col=Pal(3, n=50))
ColorLegend(x=0.45, y=1, width=0.1, col=Pal(4, n=50))
ColorLegend(x=0.6, y=1, width=0.1, col=Pal(5, n=50))
ColorLegend(x=0.75, y=1, width=0.1, col=Pal(6, n=50))
ColorLegend(x=0.9, y=1, width=0.1, col=Pal(7))
ColorLegend(x=1.05, y=1, width=0.1, col=Pal(8))

text(1:8, y=1.05, x=seq(0,1.05,.15)+.05)
title(main="DescTools palettes")

par(mfrow=c(4,2), mar=c(1,1,2,1))
barplot(1:9, col=Pal("Tibco"), axes=FALSE, main="Palette 'Tibco'" )
```

```
barplot(1:7, col=Pal("Helsana"), axes=FALSE, main="Palette 'Helsana'" )
barplot(1:7, col=SetAlpha(Pal("Helsana")[c("ecru","hellgruen","hellblau")], 0.6),
        axes=FALSE, main="Palette 'Helsana' (Alpha)" )

barplot(1:10, col=Pal("RedToBlack", 10), axes=FALSE, main="Palette 'RedToBlack'" )
barplot(1:10, col=Pal("RedBlackGreen", 10), axes=FALSE, main="Palette 'RedGreenGreen'" )
barplot(1:10, col=Pal("SteeblueWhite", 10), axes=FALSE, main="Palette 'SteeblueWhite'" )
barplot(1:10, col=Pal("RedWhiteGreen", 10), axes=FALSE, main="Palette 'RedWhiteGreen'" )
```

`DescToolsOptions` *DescTools Options*

Description

Get and set a variety of options which affect the way in which DescTools functions display results.

Usage

```
DescToolsOptions(..., default = NULL, reset = FALSE)
```

Arguments

`...`	any options can be defined, using `name = value`. However, only the ones below are used by DescTools functions.
`default`	if the specified option is not set in the options list, this value is returned. This facilitates retrieving an option and checking whether it is set and setting it separately if not.
`reset`	logical. If this is set to `TRUE`, the options will be overwritten with their default values. Other arguments will be ignored in this case. Default is `FALSE`.

Details

Invoking `DescToolsOptions()` with no arguments returns a list with the current values of the options. Note that not all options listed below are set initially. To access the value of a single option, one can simply use `DescToolsOptions("plotit")`.
To set a new value use the same rationale as with the R options: `DescToolsOptions(plotit=FALSE)`

Options used by DescTools

`col`**:** a vector of colours, defined as names or as RGB-longs (`"#RRGGBB"`). By now three colors are used in several plots as defaults. By default they're set to `hblue`, `hred` and `horange`. Change the values by defining `DescToolsOptions(col=c("pink", "blue", "yellow"))`. Any color definition can be used here.

`digits`**:** the number of **FIXED** digits, used throughout the print functions.

`fixedfont`: this font will be used by default, when `Desc` writes to a Word document. Must be defined as a font object, say enumerating `name`, `face` and `size` of the font and setting the class `font`, e.g. `structure(list(name="Courier New", size=7), class="font")`.

`fmt`: Three number format definitions are currently used in the `Desc` routines. The format used for integer values is named `"abs"`, for percentages `"perc"` and for floating point numeric values `"num"`. The format definitions must be of class `"fmt"` and may contain any argument used in the function `Format`.
Use `Fmt` to access and update formats (as they are organised in a nested list).

`footnote`: a character vector, containing characters to be used as footnote signs. Any character can be defined here. This is currently used by `TOne`.

`lang`: either `"engl"` or `"local"`, defining the language to be used for the names of weekdays and months when using `Format`.

`plotit`: logical, defining whether the `Desc`-procedures should produce plots by default. This is usually a good thing, but it may clutter up your desktop, if you're not using RStudio. Therefore it can be turned off.

`stamp`: text or expression to be placed in the right bottom corner of the `DescTools` plots. This can be useful, if some author or date information should automatically be inserted by default. Any text can be set as option, but also dynamic expressions can be used. The default would use an expression as <username>/<date>, which will use the username from the system and the current date. See defaults below.

Calling `DescToolsOptions(reset=TRUE)` will reset the options to these defaults:

```
options(DescTools = list(
  col       = c(hblue="#8296C4", hred="#9A0941", horange="#F08100"),
  digits    = 3,
  fixedfont = structure(list(name = "Consolas", size = 7), class = "font"),
  fmt       = list(abs = structure(list(digits = 0, big.mark = "'"),
                name = "abs", label = "Number format for counts", default = TRUE,
                     class = "fmt"),
                   per = structure(list(digits = 1, fmt = "%"),
                name = "per", label = "Percentage number format", default = TRUE,
                     class = "fmt"),
                   num = structure(list(digits = 3, big.mark = "'"),
                name = "num", label = "Number format for floats", default = TRUE,
                     class = "fmt")
              ),
  footnote  = c("'", "\"", "\"\""),
  lang      = "engl",
  plotit    = TRUE,
  stamp     = expression(gettextf("%s/%s", Sys.getenv("USERNAME"),
                                  Format(Today(), fmt = "yyyy-mm-dd")))
))
```

This code can as well be copied and pasted to the users' `RProfile` file, in order to have the options permanently available.

Value

For a given vector of strings the current value set for option x, or NULL if the option is unset.

If called with no arguments, returns all option settings in a list. Otherwise, it changes the named settings and invisibly returns their previous values.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

Format, Pal

Examples

```
DescToolsOptions("plotit")

## Not run:

# Get all options, defaults are attributed as such
DescToolsOptions()

# get some options
DescToolsOptions("plotit", "lang")

# get some potentially undefined option, while taking a user default and
# overriding system defaults
DescToolsOptions("stamp", default="Condor, 2016")

# get an undefined option, should return default
DescToolsOptions("stampede", default="Condor, 2016")

# set options, while getting the old values
opt <- DescToolsOptions(plotit=789, lang="portugues")
DescToolsOptions()
# output the old values
opt

# just a single argument
DescToolsOptions(digits=2)

# reset the old values
DescToolsOptions(opt)
DescToolsOptions()

# reset factory defaults
DescToolsOptions(reset=TRUE)

## End(Not run)
```

DigitSum *Calculate Digit Sum*

Description

Calculate digit sum of a number x.

Usage

```
DigitSum(x)
```

Arguments

x	an integer number

Value

the digit sum

Author(s)

Andri Signorell <andri@signorell.net> based on code by Julius benchmarked by Uwe

References

URL: https://stackoverflow.com/questions/18675285/digit-sum-function-in-r

See Also

IsPrime

Examples

```
DigitSum(c(124, 45, 268))
# [1]  7  9 16
```

DivCoef *Rao's Diversity Coefficient*

Description

Calculates Rao's diversity coefficient (also known as "Quadratic Entropy") within samples.

Usage

```
DivCoef(df, dis, scale)
```

Arguments

df	a data frame with elements as rows, samples as columns, and abundance, presence-absence or frequencies as entries
dis	an object of class dist containing distances or dissimilarities among elements. If dis is NULL, Gini-Simpson index is performed.
scale	a logical value indicating whether or not the diversity coefficient should be scaled by its maximal value over all frequency distributions.

Value

Returns a data frame with samples as rows and the diversity coefficient within samples as columns

Note

This function was previously published as divc() in the **ade4** package and has been integrated here without logical changes.

Author(s)

Sandrine Pavoine <pavoine@biomserv.univ-lyon1.fr>

References

Rao, C.R. (1982) Diversity and dissimilarity coefficients: a unified approach. *Theoretical Population Biology*, **21**, 24–43.

Gini, C. (1912) Variabilita e mutabilita. *Universite di Cagliari III*, Parte II.

Simpson, E.H. (1949) Measurement of diversity. *Nature*, **163**, 688.

Champely, S. and Chessel, D. (2002) Measuring biological diversity using Euclidean metrics. *Environmental and Ecological Statistics*, **9**, 167–177.

Examples

```
# data(ecomor)
# dtaxo <- dist.taxo(ecomor$taxo)
# DivCoef(ecomor$habitat, dtaxo)

# data(humDNAm)
# DivCoef(humDNAm$samples, sqrt(humDNAm$distances))
```

`DivCoefMax` *Maximal value of Rao's diversity coefficient also called quadratic entropy*

Description

For a given dissimilarity matrix, this function calculates the maximal value of Rao's diversity coefficient over all frequency distribution. It uses an optimization technique based on Rosen's projection gradient algorithm and is verified using the Kuhn-Tucker conditions.

Usage

```
DivCoefMax(dis, epsilon, comment)
```

Arguments

`dis`	an object of class `dist` containing distances or dissimilarities among elements.
`epsilon`	a tolerance threshold : a frequency is non null if it is higher than epsilon.
`comment`	a logical value indicating whether or not comments on the optimization technique should be printed.

Value

Returns a list

`value`	the maximal value of Rao's diversity coefficient.
`vectors`	a data frame containing four frequency distributions : sim is a simple distribution which is equal to $\frac{D1}{1^t D1}$, pro is equal to $\frac{z}{1^t z1}$, where z is the nonnegative eigenvector of the matrix containing the squared dissimilarities among the elements, met is equal to z^2, num is a frequency vector maximizing Rao's diversity coefficient.

Author(s)

Stéphane Champely <Stephane.Champely@univ-lyon1.fr>
Sandrine Pavoine <pavoine@biomserv.univ-lyon1.fr>

References

Rao, C.R. (1982) Diversity and dissimilarity coefficients: a unified approach. *Theoretical Population Biology*, **21**, 24–43.

Gini, C. (1912) Variabilita e mutabilita. *Universite di Cagliari III*, Parte II.

Simpson, E.H. (1949) Measurement of diversity. *Nature*, **163**, 688.

Champely, S. and Chessel, D. (2002) Measuring biological diversity using Euclidean metrics. *Environmental and Ecological Statistics*, **9**, 167–177.

Pavoine, S., Ollier, S. and Pontier, D. (2005) Measuring diversity from dissimilarities with Rao's quadratic entropy: are any dissimilarities suitable? *Theoretical Population Biology*, **67**, 231–239.

Examples

```
## Not run:
par.safe <- par()$mar
data(elec88)
par(mar = c(0.1, 0.1, 0.1, 0.1))
# Departments of France.
area.plot(elec88$area)

# Dissimilarity matrix.
d0 <- dist(elec88$xy)

# Frequency distribution maximizing spatial diversity in France
# according to Rao's quadratic entropy.
France.m <- DivCoefMax(d0)
w0 <- France.m$vectors$num
v0 <- France.m$value
(1:94) [w0 > 0]

# Smallest circle including all the 94 departments.
# The squared radius of that circle is the maximal value of the
# spatial diversity.
w1 = elec88$xy[c(6, 28, 66), ]
w.c = apply(w1 * w0[c(6, 28, 66)], 2, sum)
symbols(w.c[1], w.c[2], circles = sqrt(v0), inc = FALSE, add = TRUE)
s.value(elec88$xy, w0, add.plot = TRUE)
par(mar = par.safe)


# Maximisation of Rao's diversity coefficient
# with ultrametric dissimilarities.
data(microsatt)
mic.genet <- count2genet(microsatt$tab)
mic.dist <- dist.genet(mic.genet, 1)
mic.phylog <- hclust2phylog(hclust(mic.dist))
plot.phylog(mic.phylog)
mic.maxpond <- DivCoefMax(mic.phylog$Wdist)$vectors$num
dotchart.phylog(mic.phylog, mic.maxpond)

## End(Not run)
```

`Divisors` *Calculate Divisors*

Description

Calculate divisors of positive natural numbers.

Usage

```
Divisors(x)
```

Arguments

`x`	integer number for which the divisors are to be returned

Details

Divisibility is a mathematical relationship between two integers. An integer is divisible by another integer if there is no remainder in the division. The number 11 has only two divisors: 1 and the number 11 itself, whereas the number 12 has many divisors: 1, 2, 3, 4, 6 and 12. In elementary number theory, the concept of divisibility is limited to natural numbers. The number of its divisors can be determined with the function `length()`.

Value

an integer vector containg the divisors

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`Primes`, `IsPrime`, `GCD`, `LCM`

Examples

```
Divisors(c(145, 786))
```

DoBy *Evaluates a Function Groupwise*

Description

Split the vector x into partitions and apply the function to each partition separately. Computation restarts for each partition.
The logic is the same as the OLAP functions in SQL, e.g. `SUM(x) OVER (PARTITION BY group)`.

Usage

```
DoBy(x, ...)

## S3 method for class 'formula'
DoBy(formula, data = parent.frame(), subset, na.action,
     vnames = NULL, ...)
## Default S3 method:
DoBy(x, by, FUN, vnames = NULL, collapse = FALSE, ...)
```

Arguments

x	a vector that should be operated.
by	list of one or more factors, each of same length as x. If by is not a factor, the elements are coerced to factors by `as.factor()`.
FUN	Function to apply for each factor level combination.
formula	a formula of the form `lhs ~ rhs` where lhs gives the data values and rhs the corresponding groups.
data	an optional matrix or data frame (or similar: see `model.frame`) containing the variables in the formula formula. By default the variables are taken from the `parent.frame()`.
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to `getOption("na.action")`.
vnames	name for the new variables.
collapse	logical, determining if the results should be collapsed to groups. Default is FALSE.
...	optional arguments to FUN: See the "Note" section.

Details

This is more or less the same as the function ave, with the arguments organized a bit different and offering more flexibility.

Value

a data.frame with the same number of rows as length as x containing the groupwise results of FUN and the used group factors.
The attribute response denotes the name of the response variable in case the formula interface was used.

Note

Optional arguments to FUN supplied by the ... argument are not divided into cells. It is therefore inappropriate for FUN to expect additional arguments with the same length as x.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

ave, tapply, aggregate

Examples

```
d.frm <- data.frame(x=rep(1:4,3), v=sample(x=1:3, size=12, replace=TRUE),
                    g=gl(4,3,labels=letters[1:4]), m=gl(3,4,labels=LETTERS[1:3]))

# SQL-OLAP: sum() over (partition by g)
DoBy(d.frm$x, d.frm$g, FUN=sum)
# DoBy(d.frm$x, FUN=sum)

# more than 1 grouping variables are organized as list as in tapply:
DoBy(d.frm$x, list(d.frm$g, d.frm$m), mean)

# count
d.frm$count <- DoBy(d.frm$x, d.frm$g, length)

# rank
d.frm$rank <- DoBy(d.frm$v, d.frm$g, rank)
d.frm$dense_rank <- DoBy(d.frm$v, d.frm$g, Rank, ties.method="dense")
d.frm$rank_desc <- DoBy(d.frm$x, d.frm$g, function(x) rank(-x))

# row_number
d.frm$row_number <- DoBy(d.frm$v, d.frm$g, function(x) order(x))
d.frm
```

DoCall *Fast Alternative To The Internal* do.call

Description

The do.call can be somewhat slow, especially when working with large objects. This function is based upon the suggestions from Hadley Wickham on the R mailing list (reference not available anymore). Also thanks to *Tommy* at StackOverflow for suggesting how to handle double and triple colon operators, ::, further enhancing the function.

Usage

```
DoCall(what, args, quote = FALSE, envir = parent.frame())
```

Arguments

what	either a function or a non-empty character string naming the function to be called.
args	a *list* of arguments to the function call. The names attribute of args gives the argument names.
quote	a logical value indicating whether to quote the arguments.
envir	an environment within which to evaluate the call. This will be most useful if what is a character string and the arguments are symbols or quoted expressions.

Note

While the function attempts to do most of what do.call can it has limitations. It can currently not parse the example code from the original function:
do.call(paste, list(as.name("A"), as.name("B")), quote = TRUE) and the funcitonality of quote has not been thoroughly tested.

Note

This is a verbatim copy from Gmisc::fastDoCall.

Author(s)

Max Gordon <max@gforge.se>

Examples

```
DoCall("complex", list(imaginary = 1:3))

## if we already have a list (e.g. a data frame)
## we need c() to add further arguments
tmp <- expand.grid(letters[1:2], 1:3, c("+", "-"))
DoCall("paste", c(tmp, sep = ""))
```

```
## examples of where objects will be found.
A <- 2
f <- function(x) print(x^2)
env <- new.env()
assign("A", 10, envir = env)
assign("f", f, envir = env)
f <- function(x) print(x)
f(A)                                        # 2
DoCall("f", list(A))                        # 2
DoCall("f", list(A), envir = env)           # 4
DoCall(f, list(A), envir = env)             # 2
DoCall("f", list(quote(A)), envir = env)    # 100
DoCall(f, list(quote(A)), envir = env)      # 10
DoCall("f", list(as.name("A")), envir = env) # 100

eval(call("f", A))                          # 2
eval(call("f", quote(A)))                   # 2
eval(call("f", A), envir = env)             # 4
eval(call("f", quote(A)), envir = env)      # 100
```

Dot *Scalar Product*

Description

'dot' or 'scalar' product of vectors or pairwise columns of matrices.

Usage

```
Dot(x, y)
```

Arguments

x	numeric vector or matrix
y	numeric vector or matrix

Details

Returns the 'dot' or 'scalar' product of vectors or columns of matrices. Two vectors must be of same length, two matrices must be of the same size. If x and y are column or row vectors, their dot product will be computed as if they were simple vectors.

Value

A scalar or vector of length the number of columns of x and y.

Author(s)

Hans W. Borchers <hwborchers@googlemail.com>

See Also

Cross

Examples

```
Dot(1:5, 1:5)  #=> 55
# Length of space diagonal in 3-dim- cube:
sqrt(Dot(c(1,1,1), c(1,1,1)))  #=> 1.732051
```

Downloads *Return Downloaded File in XL- or CSV Format*

Description

The function searches for a file in the downloads folder and returns the file, provided it is in a readable Excel- or data format.

Usage

```
Downloads(fname, ...)
```

Arguments

fname	the filename, if no extension specified the first file found will be used.
...	are passed on to the open functions readxl() or read_table().

Value

the file

Author(s)

Andri Signorell andri@signorell.net

Examples

```
## Not run:
Downloads("yourfilename")
Downloads("yourfilename.xls")
Downloads("yourfilename.txt", sep=",", header=TRUE)

## End(Not run)
```

DrawArc *Draw Elliptic Arc(s)*

Description

Draw one or more elliptic (or circular) arcs from `theta.1` to `theta.2` on an existing plot using classic graphics.

Usage

```
DrawArc(x = 0, y = x, rx = 1, ry = rx,
        theta.1 = 0, theta.2 = 2*pi, nv = 100,
        col = par("col"), lty = par("lty"), lwd = par("lwd"),
        plot = TRUE)
```

Arguments

`x, y`	a vector (or scalar) of xy-coordinates of the center(s) of the arc(s).
`rx`	a scalar or a vector giving the semi-major axis of the ellipse for the arc(s)
`ry`	a scalar or a vector giving the semi-minor axis of the ellipse for the arc(s). Default is radius.x which will result in a circle arc with radius.x.
`theta.1`	a scalar or a vector of starting angles in radians.
`theta.2`	a scalar or a vector of ending angles in radians.
`nv`	number of vertices used to plot the arc. Scalar or vector.
`col`	color for the arc(s). Scalar or vector.
`lty`	line type used for drawing.
`lwd`	line width used for drawing.
`plot`	logical. If `TRUE` the structure will be plotted. If `FALSE` only the xy-points are calculated and returned. Use this if you want to combine several geometric structures to a single polygon.

Details

All parameters are recycled if necessary.
Be sure to use an aspect ratio of 1 as shown in the example to avoid distortion.

Value

`DrawArc` invisibly returns a list of the calculated coordinates for all shapes.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

DrawCircle, polygon

Examples

```
curve(sin(x), 0, pi, col="blue", asp=1)
DrawArc(x = pi/2, y = 0, rx = 1, theta.1 = pi/4, theta.2 = 3*pi/4, col="red")
```

DrawBand *Draw Confidence Band*

Description

Draw a band using a simple syntax. Just a wrapper for the function `polygon()` typically used to draw confidence bands.

Usage

```
DrawBand(x, y, col = SetAlpha("grey", 0.5), border = NA)
```

Arguments

x	a vector or a matrix with x coordinates for the band. If x is given as matrix it must be a $2 \times n$ matrix and the second column will be reversed. x will be recyled in the case y is a 2dimensional matrix.
y	a vector or a matrix with y coordinates for the band. If y is given as matrix it must be a $2 \times n$ matrix and the second column will be reversed. y will be recyled in the case x is a 2dimensional matrix.
col	the color of the band.
border	the border color of the band.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

polygon

Examples

```
set.seed(18)
x <- rnorm(15)
y <- x + rnorm(15)

new <- seq(-3, 3, 0.5)
pred.w.plim <- predict(lm(y ~ x), newdata=data.frame(x=new), interval="prediction")
pred.w.clim <- predict(lm(y ~ x), newdata=data.frame(x=new), interval="confidence")

plot(y ~ x)
DrawBand(y = c(pred.w.plim[,2], rev(pred.w.plim[,3])),
  x=c(new, rev(new)), col= SetAlpha("grey90", 0.5))

# passing y as matrix interface allows more intuitive arguments
DrawBand(y = pred.w.clim[, 2:3],
         x = new, col= SetAlpha("grey80", 0.5))

abline(lm(y~x), col="brown")
```

`DrawBezier` *Draw a Bezier Curve*

Description

Draw a Bezier curve.

Usage

```
DrawBezier(x = 0, y = x, nv = 100, col = par("col"), lty = par("lty"),
           lwd = par("lwd"), plot = TRUE)
```

Arguments

x, y	a vector of xy-coordinates to define the Bezier curve. Should at least contain 3 points.
nv	number of vertices to draw the curve.
col	color(s) for the curve. Default is `par("fg")`.
lty	line type for borders and shading; defaults to `"solid"`.
lwd	line width for borders and shading.
plot	logical. If `TRUE` the structure will be plotted. If `FALSE` only the xy-points are calculated and returned. Use this if you want to combine several geometric structures to a single polygon.

Details

Bezier curves appear in such areas as mechanical computer aided design (CAD). They are named after P. Bezier, who used a closely related representation in Renault's UNISURF CAD system in the early 1960s (similar, unpublished, work was done by P. de Casteljau at Citroen in the late 1950s and early 1960s). The 1970s and 1980s saw a flowering of interest in Bezier curves, with many CAD systems using them, and many important developments in their theory. The usefulness of Bezier curves resides in their many geometric and analytical properties. There are elegant and efficient algorithms for evaluation, differentiation, subdivision of the curves, and conversion to other useful representations. (See: Farin, 1993)

Value

`DrawBezier` invisibly returns a list of the calculated coordinates for all shapes.

Author(s)

Frank E Harrell Jr <f.harrell@vanderbilt.edu>

References

G. Farin (1993) *Curves and surfaces for computer aided geometric design. A practical guide*, Acad. Press

See Also

`polygon`, `DrawRegPolygon`, `DrawCircle`, `DrawArc`

Examples

```
Canvas(xlim=c(0,1))
grid()
DrawBezier( x=c(0,0.5,1), y=c(0,0.5,0), col="blue", lwd=2)
DrawBezier( x=c(0,0.5,1), y=c(0,1,0), col="red", lwd=2)
DrawBezier( x=c(0,0.25,0.5,0.75,1), y=c(0,1,1,1,0), col="darkgreen", lwd=2)
```

DrawCircle *Draw a Circle*

Description

Draw one or several circle on an existing plot.

Usage

```
DrawCircle(x = 0, y = x, r.out = 1, r.in = 0,
           theta.1 = 0, theta.2 = 2*pi, border = par("fg"),
           col = NA, lty = par("lty"), lwd = par("lwd"),
           nv = 100, plot = TRUE)
```

Arguments

x, y	a vector (or scalar) of xy-coordinates for the center(s) of the circle(s).
r.out	a vector (or scalar) of the outer radius of the circle.
r.in	a vector (or scalar) of a potential inner radius of an annulus.
theta.1	a vector (or scalar) of the starting angle(s). The sectors are built counterclockwise.
theta.2	a vector (or scalar) of the ending angle(s).
nv	number of vertices to draw the circle.
border	color for circle borders. The default is par("fg"). Use border = NA to omit borders.
col	color(s) to fill or shade the circle(s) with. The default NA (or also NULL) means do not fill, i.e., draw transparent circles, unless density is specified.
lty	line type for borders and shading; defaults to "solid".
lwd	line width for borders and shading.
plot	logical. If TRUE the structure will be plotted. If FALSE only the points are calculated and returned. Use this option if you want to combine several geometric structures to a polygon.

Details

All geometric arguments will be recycled.

Value

The function invisibly returns a list of the calculated coordinates for all shapes.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

polygon, DrawRegPolygon, DrawEllipse, DrawArc

Examples

```
Canvas(xlim = c(-5,5), xpd=TRUE)
cols <- Pal("Helsana")[1:4]

# Draw ring
DrawCircle (r.in = 1, r.out = 5, border="darkgrey",
            col=SetAlpha(DescTools::hyellow, 0.2), lwd=2)

# Draw circle
DrawCircle (r.in = 6, border=DescTools::hgreen, lwd=3)

# Draw sectors
```

```
geom <- rbind(c(-pi, 0, .25, .5), c(0, pi, 1, 2),
              c(-pi/2, pi/2, 2, 2.5), c(pi/2, 3 * pi/2, 3, 4),
              c(pi - pi/8, pi + pi/8, 1.5, 2.5))

DrawCircle (r.in = geom[,3], r.out = geom[,4],
            theta.1 = geom[,1], theta.2 = geom[,2],
            col = SetAlpha(cols, 0.6),
            border = cols, lwd=1)


# clipping
Canvas(bg="lightgrey", main="Yin ~ Yang")
DrawCircle (r.out = 1, col="white")
clip(0, 2, 2, -2)
DrawCircle(col="black")
clip(-2, 2, 2, -2)
DrawCircle (y = c(-0.5,0.5), r.out = 0.5, col=c("black", "white"), border=NA)
DrawCircle (y = c(-0.5,0.5), r.out = 0.1, col=c("white", "black"), border=NA)
DrawCircle ()


# overplotting circles
Canvas(xlim=c(-5,5))
DrawCircle (r.out=4:1, col=c("white", "steelblue2", "white", "red"), lwd=3, nv=300)


# rotation
x <- seq(-3, 3, length.out=10)
y <- rep(0, length.out=length(x))

Canvas(xlim=c(-5,5), bg="black")

sapply( (0:11) * pi/6, function(theta) {
  xy <- Rotate(x, y=y, theta=theta)
  DrawCircle (x=xy$x, y=xy$y, r.in=2.4, border=SetAlpha("white", 0.2))
} )
```

DrawEllipse *Draw an Ellipse*

Description

Draw one or several ellipses on an existing plot.

Usage

```
DrawEllipse(x = 0, y = x, radius.x = 1, radius.y = 0.5, rot = 0,
            nv = 100, border = par("fg"), col = par("bg"),
            lty = par("lty"), lwd = par("lwd"), plot = TRUE)
```

Arguments

x, y	the x and y co-ordinates for the centre(s) of the ellipse(s).
radius.x	a scalar or a vector giving the semi-major axis of the ellipse.
radius.y	a scalar or a vector giving the semi-minor axis of the ellipse.
rot	angle of rotation in radians.
nv	number of vertices to draw the ellipses.
border	color for borders. The default is par("fg"). Use border = NA to omit borders.
col	color(s) to fill or shade the annulus sector with. The default NA (or also NULL) means do not fill (say draw transparent).
lty	line type for borders and shading; defaults to "solid".
lwd	line width for borders and shading.
plot	logical. If TRUE the structure will be plotted. If FALSE only the points are calculated and returned. Use this if you want to combine several geometric structures to a single polygon.

Details

Use DegToRad if you want to define rotation angle in degrees.

Value

The function invisibly returns a list of the calculated coordinates for all shapes.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

polygon, DrawRegPolygon, DrawCircle, DrawArc

Examples

```
par(mfrow=c(1,2))

Canvas()
DrawEllipse(rot = c(1:3) * pi/3, col=SetAlpha(c("blue","red","green"), 0.5) )


plot(cars)
m <- var(cars)
eig <- eigen(m)
eig.val <- sqrt(eig$values)
eig.vec <- eig$vectors

DrawEllipse(x=mean(cars$speed), y=mean(cars$dist), radius.x=eig.val[1] , radius.y=eig.val[2]
  , rot=acos(eig.vec[1,1]), border="blue", lwd=3)
```

DrawRegPolygon *Draw Regular Polygon(s)*

Description

Draw a regular polygon with n corners. This is the workhorse function for drawing regular polygons. Drawing a circle can be done by setting the vertices to a value of say 100.

Usage

```
DrawRegPolygon(x = 0, y = x, radius.x = 1, radius.y = radius.x, rot = 0,
            nv = 3, border = par("fg"), col = par("bg"), lty = par("lty"),
            lwd = par("lwd"), plot = TRUE)
```

Arguments

x, y	a vector (or scalar) of xy-coordinates of the center(s) of the regular polygon(s).
radius.x	a scalar or a vector giving the semi-major axis of the ellipse for the polygon(s).
radius.y	a scalar or a vector giving the semi-minor axis of the ellipse for the polygon(s). Default is radius.x which will result in a polygon with radius.x.
rot	angle of rotation in radians.
nv	number of vertices to draw the polygon(s).
border	color for borders. The default is par("fg"). Use border = NA to omit borders.
col	color(s) to fill or shade the shape with. The default NA (or also NULL) means do not fill (say draw transparent).
lty	line type for borders and shading; defaults to "solid".
lwd	line width for borders and shading.
plot	logical. If TRUE the structure will be plotted. If FALSE only the points are calculated and returned. Use this if you want to combine several geometric structures to a polygon.

Details

All geometric arguments will be recycled.

Value

The function invisibly returns a list of the calculated coordinates for all shapes.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

polygon, DrawCircle, DrawArc

Examples

```
# Draw 4 triangles (nv = 3) with different rotation angles
plot(c(0,1),c(0,1), asp=1, type="n", xaxt="n", yaxt="n", xlab="", ylab="")
DrawRegPolygon(x = 0.5, y = 0.5, rot = (1:4)*pi/6, radius.x = 0.5, nv = 3,
  col = SetAlpha("yellow",0.5))


# Draw several polygons
plot(c(0,1),c(0,1), asp=1, type="n", xaxt="n", yaxt="n", xlab="", ylab="")
DrawRegPolygon(x = 0.5, y = 0.5, radius.x=seq(50, 5, -10) * 1 /100,
  rot=0, nv = c(50, 10, 7, 4, 3), col=SetAlpha("blue",seq(0.2,0.7,0.1)))


# Combine several polygons by sorting the coordinates
# Calculate the xy-points for two concentric pentagons
d.pts <- do.call("rbind", lapply(DrawRegPolygon(radius.x=c(1,0.38), nv=5,
  rot=c(pi/2, pi/2+pi/5), plot=FALSE ), data.frame))

# prepare plot
plot(c(-1,1),c(-1,1), asp=1, type="n", xaxt="n", yaxt="n", xlab="", ylab="")

# .. and draw the polygon with reordered points
polygon( d.pts[order(rep(1:6, times=2), rep(1:2, each=6)), c("x","y")], col="yellow")



# Move the center
plot(c(0,1),c(0,1), asp=1, type="n", xaxt="n", yaxt="n", xlab="", ylab="")
theta <- seq(0, pi/6, length.out=5)
xy <- PolToCart( exp(theta) /2, theta)
DrawRegPolygon(x=xy$x, y=xy$y + 0.5, radius.x=seq(0.5, 0.1, -0.1),
  nv=4, rot=seq(0, pi/2, length.out=5), col=rainbow(5) )


# Plot a polygon with a "hole"
plot(c(-1,1),c(-1,1), asp=1, type="n", xaxt="n", yaxt="n", xlab="", ylab="")
DrawRegPolygon(nv = 4, rot=pi/4, col="red" )
text(x=0,y=0, "Polygon", cex=6, srt=45)

# Calculate circle and hexagon, but do not plot
pts <- DrawRegPolygon(radius.x=c(0.7, 0.5), nv = c(100, 6), plot=FALSE )

# combine the 2 shapes and plot the new structure
polygon(x = unlist(lapply(pts, "[", "x")),
  y=unlist(lapply(pts, "[", "y")), col="green", border=FALSE)
```

Dummy *Generate Dummy Codes for a Factor*

Description

Generate a matrix of dummy codes (class indicators) for a given factor.

Usage

```
Dummy(x, method = c("treatment", "sum", "helmert", "poly", "full"),
      base = 1, levels = NULL)
```

Arguments

`x`	factor or vector of classes for cases.
`method`	defines the method of the contrasts being formed. Can be one out of `"treatment"`, `"sum"`, `"helmert"`, `"poly"`, `"full"`, whereas `"treatment"` is the default one. Abbreviations are accepted. The option `"full"` returns a full set of class indicators, say a dummy factor for **each** level of x. Note that this would be redundant for `lm()` and friends!
`base`	an integer specifying which group is considered the baseline group.
`levels`	an optional vector of the values (as character strings) that `x` might have taken. The default is the unique set of values taken by as.character(x), sorted into increasing order of x. This is directly passed on to `factor`.

Details

For reverting dummy codes see the approach in the examples below.

Value

a matrix with the dummy codes. The number of rows correspond to the number of elements in `x` and the number of columns to the number of its levels - 1, respectively to the number of levels given as argument -1.

When `method = "full"` is chosen the number of columns will correspond to the number of levels.

Author(s)

Andri Signorell <andri@signorell.net>

References

Venables, W N and Ripley, B D (2002): *Modern Applied Statistics with S*. Fourth edition. Springer.

See Also

`model.frame`, `contrasts`, `class.ind` in the package **nnet**

Examples

```
x <- c("red","blue","green","blue","green","red","red","blue")
Dummy(x)
Dummy(x, base=2)

Dummy(x, method="sum")


y <- c("Max","Max","Max","Max","Max","Bill","Bill","Bill")

Dummy(y)
Dummy(y, base="Max")

Dummy(y, base="Max", method="full")


# "Undummy" (revert the dummy coding)
m <- Dummy(y, method="full")
m
z <- apply(m, 1, function(x) colnames(m)[x==1])
z
identical(y, as.vector(z))

m <- Dummy(y)
m
z <- apply(m, 1, function(x) ifelse(sum(x)==0, attr(m,"base"), colnames(m)[x==1]))
z
```

DunnettTest *Dunnett's Test for Comparing Several Treatments With a Control*

Description

Performs Dunnett's test for comparing several treatments with a control.

Usage

```
DunnettTest(x, ...)

## Default S3 method:
DunnettTest(x, g, control = NULL, conf.level = 0.95, ...)

## S3 method for class 'formula'
DunnettTest(formula, data, subset, na.action, ...)
```

Arguments

x	a numeric vector of data values, or a list of numeric data vectors.
g	a vector or factor object giving the group for the corresponding elements of x. Ignored if x is a list.
control	the level of the control group against which the others should be tested. If there are multiple levels the calculation will be performed for every one.
conf.level	confidence level of the interval.
formula	a formula of the form lhs ~ rhs where lhs gives the data values and rhs the corresponding groups.
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
...	further arguments to be passed to or from methods.

Details

DunnettTest does the post hoc pairwise multiple comparisons procedure.

If x is a list, its elements are taken as the samples to be compared, and hence have to be numeric data vectors. In this case, g is ignored, and one can simply use DunnettTest(x) to perform the test. If the samples are not yet contained in a list, use DunnettTest(list(x, ...)).

Otherwise, x must be a numeric data vector, and g must be a vector or factor object of the same length as x giving the group for the corresponding elements of x.

Value

A list of class c("PostHocTest"), containing one matrix named after the control with columns diff giving the difference in the observed means, lwr.ci giving the lower end point of the interval, upr.ci giving the upper end point and pval giving the p-value after adjustment for the multiple comparisons.

There are print and plot methods for class "PostHocTest". The plot method does not accept xlab, ylab or main arguments and creates its own values for each plot.

Author(s)

Andri Signorell <andri@signorell.net>, the interface is based on R-Core code

References

Dunnett C. W. (1955) A multiple comparison procedure for comparing several treatments with a control, *Journal of the American Statistical Association*, 50:1096-1121.

See Also

PostHocTest

Examples

```
## Hollander & Wolfe (1973), 116.
## Mucociliary efficiency from the rate of removal of dust in normal
##  subjects, subjects with obstructive airway disease, and subjects
##  with asbestosis.
x <- c(2.9, 3.0, 2.5, 2.6, 3.2) # normal subjects
y <- c(3.8, 2.7, 4.0, 2.4)      # with obstructive airway disease
z <- c(2.8, 3.4, 3.7, 2.2, 2.0) # with asbestosis

DunnettTest(list(x, y, z))

## Equivalently,
x <- c(x, y, z)
g <- factor(rep(1:3, c(5, 4, 5)),
            labels = c("Normal subjects",
                       "Subjects with obstructive airway disease",
                       "Subjects with asbestosis"))

DunnettTest(x, g)

## Formula interface
boxplot(Ozone ~ Month, data = airquality)
DunnettTest(Ozone ~ Month, data = airquality)

DunnettTest(Ozone ~ Month, data = airquality, control="8", conf.level=0.9)
```

DunnTest *Dunn's Test of Multiple Comparisons*

Description

Performs Dunn's test of multiple comparisons using rank sums.

Usage

```
DunnTest(x, ...)

## Default S3 method:
DunnTest(x, g,
         method = c("holm", "hochberg", "hommel", "bonferroni", "BH",
                    "BY", "fdr", "none"),
         alternative = c("two.sided", "less", "greater"),
         out.list = TRUE, ...)

## S3 method for class 'formula'
```

```
DunnTest(formula, data, subset, na.action, ...)

## S3 method for class 'DunnTest'
print(x, digits = getOption("digits", 3), ...)
```

Arguments

x	a numeric vector of data values, or a list of numeric data vectors.
g	a vector or factor object giving the group for the corresponding elements of x. Ignored if x is a list.
method	the method for adjusting p-values for multiple comparisons. The function is calling p.adjust and this parameter is directly passed through.
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter.
out.list	logical, indicating if the results should be printed in list mode or as a square matrix. Default is list (TRUE).
formula	a formula of the form lhs ~ rhs where lhs gives the data values and rhs the corresponding groups.
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
digits	controls the number of fixed digits to print.
...	further arguments to be passed to or from methods.

Details

DunnTest performs the post hoc pairwise multiple comparisons procedure appropriate to follow the rejection of a Kruskal-Wallis test. The Kruskal-Wallis test, being a non-parametric analog of the one-way ANOVA, is an omnibus test of the null hypothesis that none of k groups stochastically dominate one another. Dunn's test is constructed in part by summing jointly ranked data. The rank sum test, itself a non-parametric analog of the unpaired t-test, is possibly intuitive, but inappropriate as a post hoc pairwise test, because (1) it fails to retain the dependent ranking that produced the Kruskal-Wallis test statistic, and (2) it does not incorporate the pooled variance estimate implied by the null hypothesis of the Kruskal-Wallis test.

If x is a list, its elements are taken as the samples to be compared, and hence have to be numeric data vectors. In this case, g is ignored, and one can simply use DunnTest(x) to perform the test. If the samples are not yet contained in a list, use DunnTest(list(x, ...)).

Otherwise, x must be a numeric data vector, and g must be a vector or factor object of the same length as x giving the group for the corresponding elements of x.

Value

A list with class "DunnTest" containing the following components:

res	an array containing the mean rank differencens and the according p-values

Author(s)

Andri Signorell <andri@signorell.net>, the interface is based on R-Core code

References

Dunn, O. J. (1961) Multiple comparisons among means *Journal of the American Statistical Association*, 56(293):52-64.

Dunn, O. J. (1964) Multiple comparisons using rank sums *Technometrics*, 6(3):241-252.

See Also

kruskal.test, wilcox.test, p.adjust

Examples

```
## Hollander & Wolfe (1973), 116.
## Mucociliary efficiency from the rate of removal of dust in normal
##  subjects, subjects with obstructive airway disease, and subjects
##  with asbestosis.
x <- c(2.9, 3.0, 2.5, 2.6, 3.2) # normal subjects
y <- c(3.8, 2.7, 4.0, 2.4)      # with obstructive airway disease
z <- c(2.8, 3.4, 3.7, 2.2, 2.0) # with asbestosis
DunnTest(list(x, y, z))

## Equivalently,
x <- c(x, y, z)
g <- factor(rep(1:3, c(5, 4, 5)),
            labels = c("Normal subjects",
                       "Subjects with obstructive airway disease",
                       "Subjects with asbestosis"))

# do the kruskal.test first
kruskal.test(x, g)

# ...and the pairwise test afterwards
DunnTest(x, g)

## Formula interface.
boxplot(Ozone ~ Month, data = airquality)
DunnTest(Ozone ~ Month, data = airquality)
```

DurbinWatsonTest *Durbin-Watson Test*

Description

Performs the Durbin-Watson test for autocorrelation of disturbances.

Usage

```
DurbinWatsonTest(formula, order.by = NULL,
                 alternative = c("greater", "two.sided", "less"),
                 iterations = 15, exact = NULL, tol = 1e-10, data = list())
```

Arguments

formula	a symbolic description for the model to be tested (or a fitted "lm" object).
order.by	Either a vector z or a formula with a single explanatory variable like ~ z. The observations in the model are ordered by the size of z. If set to NULL (the default) the observations are assumed to be ordered (e.g., a time series).
alternative	a character string specifying the alternative hypothesis.
iterations	an integer specifying the number of iterations when calculating the p-value with the "pan" algorithm.
exact	logical. If set to FALSE a normal approximation will be used to compute the p value, if TRUE the "pan" algorithm is used. The default is to use "pan" if the sample size is < 100.
tol	tolerance. Eigenvalues computed have to be greater than tol to be treated as non-zero.
data	an optional data frame containing the variables in the model. By default the variables are taken from the environment which DurbinWatsonTest is called from.

Details

The Durbin-Watson test has the null hypothesis that the autocorrelation of the disturbances is 0. It is possible to test against the alternative that it is greater than, not equal to, or less than 0, respectively. This can be specified by the alternative argument.

Under the assumption of normally distributed disturbances, the null distribution of the Durbin-Watson statistic is the distribution of a linear combination of chi-squared variables. The p-value is computed using the Fortran version of Applied Statistics Algorithm AS 153 by Farebrother (1980, 1984). This algorithm is called "pan" or "gradsol". For large sample sizes the algorithm might fail to compute the p value; in that case a warning is printed and an approximate p value will be given; this p value is computed using a normal approximation with mean and variance of the Durbin-Watson test statistic.

Examples can not only be found on this page, but also on the help pages of the data sets bondyield, currencysubstitution, growthofmoney, moneydemand, unemployment, wages.

For an overview on R and econometrics see Racine & Hyndman (2002).

Value

An object of class `"htest"` containing:

`statistic`	the test statistic.
`p.value`	the corresponding p-value.
`method`	a character string with the method used.
`data.name`	a character string with the data name.

Note

This function was previously published as `dwtest` in the **lmtest** package and has been integrated here without logical changes.

Author(s)

Torsten Hothorn, Achim Zeileis, Richard W. Farebrother (pan.f), Clint Cummins (pan.f), Giovanni Millo, David Mitchell

References

J. Durbin & G.S. Watson (1950), Testing for Serial Correlation in Least Squares Regression I. *Biometrika* **37**, 409–428.

J. Durbin & G.S. Watson (1951), Testing for Serial Correlation in Least Squares Regression II. *Biometrika* **38**, 159–178.

J. Durbin & G.S. Watson (1971), Testing for Serial Correlation in Least Squares Regression III. *Biometrika* **58**, 1–19.

R.W. Farebrother (1980), Pan's Procedure for the Tail Probabilities of the Durbin-Watson Statistic (Corr: 81V30 p189; AS R52: 84V33 p363- 366; AS R53: 84V33 p366- 369). *Applied Statistics* **29**, 224–227.

R. W. Farebrother (1984), [AS R53] A Remark on Algorithms AS 106 (77V26 p92-98), AS 153 (80V29 p224-227) and AS 155: The Distribution of a Linear Combination of χ^2 Random Variables (80V29 p323-333) *Applied Statistics* **33**, 366–369.

W. Krämer & H. Sonnberger (1986), *The Linear Regression Model under Test*. Heidelberg: Physica.

J. Racine & R. Hyndman (2002), Using R To Teach Econometrics. *Journal of Applied Econometrics* **17**, 175–189.

See Also

`lm`

Examples

```
## generate two AR(1) error terms with parameter
## rho = 0 (white noise) and rho = 0.9 respectively
err1 <- rnorm(100)

## generate regressor and dependent variable
```

```
x <- rep(c(-1,1), 50)
y1 <- 1 + x + err1

## perform Durbin-Watson test
DurbinWatsonTest(y1 ~ x)

err2 <- stats::filter(err1, 0.9, method="recursive")
y2 <- 1 + x + err2
DurbinWatsonTest(y2 ~ x)

## for a simple vector use:
e_t <- c(-32.33, -26.603, 2.215, -16.967, -1.148, -2.512, -1.967, 11.669,
         -0.513, 27.032, -4.422, 40.032, 23.577, 33.94, -2.787, -8.606,
          0.575, 6.848, -18.971, -29.063)
DurbinWatsonTest(e_t ~ 1)
```

`Entropy` *Shannon Entropy and Mutual Information*

Description

Computes Shannon entropy and the mutual information of two variables. The entropy quantifies the expected value of the information contained in a vector. The mutual information is a quantity that measures the mutual dependence of the two random variables.

Usage

```
Entropy(x, y = NULL, base = 2, ...)

MutInf(x, y, base = 2, ...)
```

Arguments

`x`	a vector or a matrix of numerical or categorical type. If only x is supplied it will be interpreted as contingency table.
`y`	a vector with the same type and dimension as x. If y is not `NULL` then the entropy of `table(x, y, ...)` will be calculated.
`base`	base of the logarithm to be used, defaults to 2.
`...`	further arguments are passed to the function `table`, allowing i.e. to set `useNA`.

Details

The Shannon entropy equation provides a way to estimate the average minimum number of bits needed to encode a string of symbols, based on the frequency of the symbols.
It is given by the formula $H = -\sum(\pi log(\pi))$ where π is the probability of character number i showing up in a stream of characters of the given "script".
The entropy is ranging from 0 to Inf.

Value

a numeric value.

Author(s)

Andri Signorell <andri@signorell.net>

References

Shannon, Claude E. (July/October 1948). A Mathematical Theory of Communication, *Bell System Technical Journal* 27 (3): 379-423.

Ihara, Shunsuke (1993) *Information theory for continuous systems*, World Scientific. p. 2. ISBN 978-981-02-0985-8.

See Also

package **entropy** which implements various estimators of entropy

Examples

```
Entropy(as.matrix(rep(1/8, 8)))

# http://r.789695.n4.nabble.com/entropy-package-how-to-compute-mutual-information-td4385339.html
x <- as.factor(c("a","b","a","c","b","c"))
y <- as.factor(c("b","a","a","c","c","b"))

Entropy(table(x), base=exp(1))
Entropy(table(y), base=exp(1))
Entropy(x, y, base=exp(1))

# Mutual information is
Entropy(table(x), base=exp(1)) + Entropy(table(y), base=exp(1)) - Entropy(x, y, base=exp(1))
MutInf(x, y, base=exp(1))

Entropy(table(x)) + Entropy(table(y)) - Entropy(x, y)
MutInf(x, y, base=2)

# http://en.wikipedia.org/wiki/Cluster_labeling
tab <- matrix(c(60,10000,200,500000), nrow=2, byrow=TRUE)
MutInf(tab, base=2)

d.frm <- Untable(as.table(tab))
str(d.frm)
MutInf(d.frm[,1], d.frm[,2])

table(d.frm[,1], d.frm[,2])

MutInf(table(d.frm[,1], d.frm[,2]))


# Ranking mutual information can help to describe clusters
```

```
#
#   r.mi <- MutInf(x, grp)
#   attributes(r.mi)$dimnames <- attributes(tab)$dimnames
#
#   # calculating ranks of mutual information
#   r.mi_r <- apply( -r.mi, 2, rank, na.last=TRUE )
#   # show only first 6 ranks
#   r.mi_r6 <- ifelse( r.mi_r < 7, r.mi_r, NA)
#   attributes(r.mi_r6)$dimnames <- attributes(tab)$dimnames
#   r.mi_r6
```

Eps *Greenhouse-Geisser And Huynh-Feldt Epsilons*

Description

Calculate Greenhouse-Geisser and Huynh-Feldt epsilons.

Usage

```
Eps(S, p, g, n)
```

Arguments

S	pxp covariance matrix
p	dimension of observation vectors
g	number of groups
n	number of subjects

Value

a numeric value

Author(s)

Hans Rudolf Roth <hroth@retired.ethz.ch>

References

Vonesh, E.F., Chinchilli, V.M. (1997) *Linear and Nonlinear Models for the Analysis of Repeated Measurements* Marcel Dekker, New York, p.84-86

Crowder, M.J., Hand, D.J. (1990) *Analysis of Repeated Measures*. Chapman & Hall, London, p.54-55

See Also

aov

Examples

```
## find!
```

ErrBars *Add Error Bars to an Existing Plot*

Description

Add error bars to an existing plot.

Usage

```
ErrBars(from, to = NULL, pos = NULL, mid = NULL, horiz = FALSE, col = par("fg"),
        lty = par("lty"), lwd = par("lwd"), code = 3, length = 0.05,
       pch = NA, cex.pch = par("cex"), col.pch = par("fg"), bg.pch = par("bg"),
        ...)
```

Arguments

from	coordinates of points **from** which to draw (the lower end of the error bars). If to is left to NULL and from is a $k \times 2$ dimensional matrix, the first column will be interpreted as from and the second as to.
to	coordinates of points **to** which to draw (the upper end of the error bars).
pos	numeric, position of the error bars. This will either be the x-coordinate in case of vertical error bars and the y-coordinate in case of horizontal error bars.
mid	numeric, position of midpoints. Defaults to the mean of from and to.
horiz	logical, determining whether horizontal error bars are needed (default is FALSE).
col	the line color.
lty	the line type.
lwd	line width.
code	integer code, determining where end lines are to be drawn. code = 0 will draw no end lines, code = 1 will draw an end line on the left (lower) side at (x0[i], y0[i]), code = 2 on the right (upper) side (x1[i], y1[i]) and code = 3 (default) will draw end lines at both ends.
length	the length of the end lines.
pch	plotting character for the midpoints. The position of the points is given by mid. If mid is left to NULL the points will be plotted in the middle of from and to. No points will be plotted if this is set to NA, which is the default.
cex.pch	the character extension for the plotting characters. Default is par("cex").
col.pch	the color of the plotting characters. Default is par("fg").
bg.pch	the background color of the plotting characters (if pch is set to 21:25). Default is par("bg").
...	the dots are passed to the arrows function.

Details

A short wrapper for plotting error bars by means of arrows.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

arrows, lines.loess

Examples

```
par(mfrow=c(2,2))
b <- barplot(1:5, ylim=c(0,6))
ErrBars(from=1:5-rep(0.5,5), to=1:5+rep(0.5,5), pos=b, length=0.2)

# just on one side
b <- barplot(1:5, ylim=c(0,6))
ErrBars(from=1:5, to=1:5+rep(0.5,5), pos=b, length=0.2, col="red", code=2, lwd=2)

b <- barplot(1:5, xlim=c(0,6), horiz=TRUE)
ErrBars(from=1:5, to=1:5+rep(0.2,5), pos=b, horiz=TRUE, length=0.2, col="red", code=2, lwd=2)

par(xpd=FALSE)
dotchart(1:5, xlim=c(0,6))
ErrBars(from=1:5-rep(0.2,5), to=1:5+rep(0.2,5), horiz=TRUE, length=0.1)
```

EtaSq *Effect Size Calculations for ANOVAs*

Description

Calculates eta-squared, partial eta-squared and generalized eta-squared

Usage

```
EtaSq(x, type = 2, anova = FALSE)

## S3 method for class 'lm'
EtaSq(x, type = 2, anova = FALSE)

## S3 method for class 'aovlist'
EtaSq(x, type = 2, anova = FALSE)
```

Arguments

x	An analysis of variance (aov, aovlist) object.
type	What type of sum of squares to calculate? EtaSq.aovlist requires type=1.
anova	Should the full ANOVA table be printed out in addition to the effect sizes?

Details

Calculates the eta-squared, partial eta-squared, and generalized eta-squared measures of effect size that are commonly used in analysis of variance. The input x should be the analysis of variance object itself. For between-subjects designs, generalized eta-squared equals partial eta-squared. The reported generalized eta-squared for repeated-measures designs assumes that all factors are manipulated, i.e., that there are no measured factors like gender (see references).

For unbalanced designs, the default in EtaSq is to compute Type II sums of squares (type=2), in keeping with the Anova function in the car package. It is possible to revert to the Type I SS values (type=1) to be consistent with anova, but this rarely tests hypotheses of interest. Type III SS values (type=3) can also be computed. EtaSq.aovlist requires type=1.

Value

If anova=FALSE, the output for EtaSq.lm is an M x 2 matrix, for EtaSq.aovlist it is an M x 3 matrix. Each of the M rows corresponds to one of the terms in the ANOVA (e.g., main effect 1, main effect 2, interaction, etc), and each of the columns corresponds to a different measure of effect size. Column 1 contains the eta-squared values, and column 2 contains partial eta-squared values. Column 3 contains the generalized eta-squared values. If anova=TRUE, the output contains additional columns containing the sums of squares, mean squares, degrees of freedom, F-statistics and p-values. For EtaSq.aovlist, additional columns contain the error sum of squares and error degrees of freedom corresponding to an effect term.

Author(s)

Danielle Navarro <djnavarro@protonmail.com>, Daniel Wollschlaeger <dwoll@psychologie.uni-kiel.de>

References

Bakeman, R. (2005). Recommended effect size statistics for repeated measures designs. Behavior Research Methods 37(3), 379-384.

Olejnik, S. and Algina, J. (2003). Generalized Eta and Omega Squared Statistics: Measures of Effect Size for Some Common Research Designs. Psychological Methods 8(4), 434-447.

See Also

aov, anova, Anova

Examples

```
#### Example 1: one-way ANOVA ####

outcome <- c(1.4,2.1,3.0,2.1,3.2,4.7,3.5,4.5,5.4)    # data
treatment1 <- factor(c(1,1,1,2,2,2,3,3,3))           # grouping variable
anova1 <- aov(outcome ~ treatment1)                  # run the ANOVA
summary(anova1)                                      # print the ANOVA table
EtaSq(anova1)                                        # effect size

#### Example 2: two-way ANOVA ####

treatment2 <- factor(c(1,2,3,1,2,3,1,2,3))       # second grouping variable
anova2 <- aov(outcome ~ treatment1 + treatment2) # run the ANOVA
summary(anova2)                                  # print the ANOVA table
EtaSq(anova2)                                    # effect size

#### Example 3: two-way ANOVA unbalanced cell sizes ####
#### data from Maxwell & Delaney, 2004              ####
#### Designing experiments and analyzing data       ####

dfMD <- data.frame(IV1=factor(rep(1:3, c(3+5+7, 5+6+4, 5+4+6))),
                   IV2=factor(rep(rep(1:3, 3), c(3,5,7, 5,6,4, 5,4,6))),
                 DV=c(c(41, 43, 50), c(51, 43, 53, 54, 46), c(45, 55, 56, 60, 58, 62, 62),
                      c(56, 47, 45, 46, 49), c(58, 54, 49, 61, 52, 62), c(59, 55, 68, 63),
                     c(43, 56, 48, 46, 47), c(59, 46, 58, 54), c(55, 69, 63, 56, 62, 67)))

# use contr.sum for correct sum of squares type 3
dfMD$IV1s <- C(dfMD$IV1, "contr.sum")
dfMD$IV2s <- C(dfMD$IV2, "contr.sum")
dfMD$IV1t <- C(dfMD$IV1, "contr.treatment")
dfMD$IV2t <- C(dfMD$IV2, "contr.treatment")

EtaSq(aov(DV ~ IV1s*IV2s, data=dfMD), type=3)
EtaSq(aov(DV ~ IV1t*IV2t, data=dfMD), type=1)

#### Example 4: two-way split-plot ANOVA -> EtaSq.aovlist ####

DV_t1 <- round(rnorm(3*10, -0.5, 1), 2)
DV_t2 <- round(rnorm(3*10,  0,   1), 2)
DV_t3 <- round(rnorm(3*10,  0.5, 1), 2)
dfSPF <- data.frame(id=factor(rep(1:(3*10), times=3)),
                    IVbtw=factor(rep(LETTERS[1:3], times=3*10)),
IVwth=factor(rep(1:3, each=3*10)),
DV=c(DV_t1, DV_t2, DV_t3))
spf <- aov(DV ~ IVbtw*IVwth + Error(id/IVwth), data=dfSPF)
EtaSq(spf, type=1, anova=TRUE)
```

EX *Expected Value and Variance*

Description

Expected Value and Variance for the distribution of a discrete random variable. (For didactical purposes..)

Usage

```
EX(x, p)
VarX(x, p)
```

Arguments

x	the values of the random variable
p	the probabilities of the values

Value

numeric value

Author(s)

Andri Signorell <andri@signorell.net>

Examples

```
EX(x=c(1:3), p=c(0.2, 0.5, 0.3))
VarX(x=c(1:3), p=c(0.2, 0.5, 0.3))
```

ExpFreq *Expected Frequencies*

Description

Calculate the expected frequencies of an n-way table assuming independence.

Usage

```
ExpFreq(x, freq = c("abs", "rel"))
```

Arguments

x	a table.
freq	indicates, whether absolute or relative frenquencies should be computed. Can either be "abs" or "rel". Partial matching is supported.

Value

A table with either the absolute or the relative expected frequencies.

Note

This is a copy of the function `independence_table` in **vcd**.

Author(s)

David Meyer <David.Meyer@R-project.org>

See Also

`chisq.test`

Examples

```
ExpFreq(Titanic)

ExpFreq(UCBAdmissions, freq="r")
```

Extremes *Kth Smallest/Largest Values*

Description

Find the kth smallest, resp. largest values from a vector `x` and return the values and their frequencies.

Usage

```
Small(x, k = 5, unique = FALSE, na.last = NA)
Large(x, k = 5, unique = FALSE, na.last = NA)

HighLow(x, nlow = 5, nhigh = nlow, na.last = NA)
```

Arguments

`x`	a `numeric` vector
`k`	an integer >0 defining how many extreme values should be returned. Default is `k = 5`. If `k > length(x)`, all values will be returned.
`unique`	logical, defining if unique values should be considered or not. If this is set to `TRUE`, a list with the `k` extreme values and their frequencies is returned. Default is `FALSE` (as unique is a rather expensive function).
`na.last`	for controlling the treatment of `NA`s. If `TRUE`, missing values in the data are put last; if `FALSE`, they are put first; if `NA`, they are removed.
`nlow`	a single integer. The number of the smallest elements of a vector to be printed. Defaults to 5.
`nhigh`	a single integer. The number of the greatest elements of a vector to be printed. Defaults to the number of `nlow`.

Details

This does not seem to be a difficult problem at first sight. We could simply tabulate and sort the vector and finally take the first or last k values. However sorting and tabulating the whole vector when we're just interested in the few smallest values is a considerable waste of resources. This approach becomes already impracticable for medium vector lengths (~10^5). There are several points and solutions of this problem discussed out there. The present implementation is based on highly efficient C++ code and proved to be very fast.

HighLow combines the two upper functions and reports the k extreme values on both sides together with their frequencies in parentheses. It is used for describing univariate variables and is interesting for checking the ends of the vector, where in real data often wrong values accumulate. This is in essence a printing routine for the highest and the lowest values of x.

Value

if `unique` is set to `FALSE`: a vector with the k most extreme values,
else: a list, containing the k most extreme values and their frequencies.

Author(s)

Andri Signorell <andri@signorell.net>
C++ parts by Nathan Russell and Romain Francois

References

https://stackoverflow.com/questions/36993935/find-the-largest-n-unique-values-and-their-frequencies

https://gallery.rcpp.org/articles/top-elements-from-vectors-using-priority-queue/

See Also

`max`, `max`, `sort`, `rank`

Examples

```
x <- sample(1:10, 1000, rep=TRUE)
Large(x, 3)
Large(x, k=3, unique=TRUE)

# works fine up to x ~ 1e6
x <- runif(1000000)
Small(x, 3, unique=TRUE)
Small(x, 3, unique=FALSE)

# Both ends
cat(HighLow(d.pizza$temperature, na.last=NA))
```

ExtrVal *Distributions of Maxima and Minima*

Description

Density function, distribution function, quantile function and random generation for the maximum/minimum of a given number of independent variables from a specified distribution.

Usage

```
dExtrVal(x, densfun, distnfun, ..., distn, mlen = 1, largest = TRUE,
    log = FALSE)
pExtrVal(q, distnfun, ..., distn, mlen = 1, largest = TRUE,
    lower.tail = TRUE)
qExtrVal(p, quantfun, ..., distn, mlen = 1, largest = TRUE,
    lower.tail = TRUE)
rExtrVal(n, quantfun, ..., distn, mlen = 1, largest = TRUE)
```

Arguments

x, q	Vector of quantiles.
p	Vector of probabilities.
n	Number of observations.
densfun, distnfun, quantfun	Density, distribution and quantile function of the specified distribution. The density function must have a log argument (a simple wrapper can always be constructed to achieve this).
...	Parameters of the specified distribution.
distn	A character string, optionally given as an alternative to densfun, distnfun and quantfun such that the density, distribution and quantile functions are formed upon the addition of the prefixes d, p and q respectively.
mlen	The number of independent variables.
largest	Logical; if TRUE (default) use maxima, otherwise minima.
log	Logical; if TRUE, the log density is returned.
lower.tail	Logical; if TRUE (default) probabilities are P[X <= x], otherwise P[X > x].

Value

dExtrVal gives the density function, pExtrVal gives the distribution function and qExtrVal gives the quantile function of the maximum/minimum of mlen independent variables from a specified distibution. rExtrVal generates random deviates.

Author(s)

Alec Stephenson <alec_stephenson@hotmail.com>

See Also

rGenExtrVal, rOrder

Examples

```
dExtrVal(2:4, dnorm, pnorm, mean = 0.5, sd = 1.2, mlen = 5)
dExtrVal(2:4, distn = "norm", mean = 0.5, sd = 1.2, mlen = 5)
dExtrVal(2:4, distn = "exp", mlen = 2, largest = FALSE)
pExtrVal(2:4, distn = "exp", rate = 1.2, mlen = 2)
qExtrVal(seq(0.9, 0.6, -0.1), distn = "exp", rate = 1.2, mlen = 2)
rExtrVal(5, qgamma, shape = 1, mlen = 10)
p <- (1:9)/10
pexp(qExtrVal(p, distn = "exp", rate = 1.2, mlen = 1), rate = 1.2)
## [1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
```

Factorize *Prime Factorization of Integers*

Description

Compute the prime factorization(s) of integer(s) n.

Usage

```
Factorize(n)
```

Arguments

n	vector of integers to factorize.

Details

works via Primes, currently in a cheap way, sub-optimal for large composite n.

Value

A named list of the same length as n, each element a 2-column matrix with column "p" the prime factors and column~"m" their respective exponents (or multiplities), i.e., for a prime number n, the resulting matrix is cbind(p = n, m = 1).

Author(s)

Martin Maechler, Jan. 1996.

See Also

GCD, LCM, Primes, IsPrime, Divisors

For factorization of moderately or really large numbers, see the **gmp** package, and its factorize() (which is ~20x faster!).

Examples

```
Factorize(47)
Factorize(seq(101, 120, by=2))
```

FctArgs *Retrieve a Function's Arguments*

Description

Retrieve a function's arguments and default values in a list.

Usage

```
FctArgs(name, sort = FALSE)
```

Arguments

name	name of the function.
sort	logical. Should the function arguments be sorted? Defaults to FALSE.

Value

a data.frame with the default in the first columns and with row.names as argument names.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

formalArgs, formals, args, alist, body

Examples

```
formalArgs(PlotFdist)
formals(PlotFdist)

# compare:
FctArgs(PlotFdist)

# alternative also spotting unexported functions
GetArgs <- function(FUN) {
  a <- formals(getAnywhere(FUN)$objs[[1]])
  arg.labels <- names(a)
  arg.values <- as.character(a)
  char <- sapply(a, is.character)
  arg.values[char] <- paste("\"", arg.values[char], "\"", sep="")

  c(fname=FUN,
```

```
    args=paste(StrTrim(gsub("= $", "",
               paste(arg.labels, arg.values, sep=" = "))),
               collapse=", "))

}

fcts <- grep("plot.Desc", unclass(lsf.str(envir = asNamespace("DescTools"),
                                          all.names = TRUE)), value=TRUE)
fargs <- t(unname(sapply(fcts, GetArgs)))
fargs
```

`Fibonacci` *Fibonacci Numbers*

Description

Generates Fibonacci numbers.

Usage

```
Fibonacci(n)
```

Arguments

`n`	nonnegative integer or vector of nonnegative integers.

Details

Generates the `n`-th Fibonacci number, whereas Fibonacci(0) = 0.

Value

A single integer, or a vector of integers.

Author(s)

Andri Signorell <andri@signorell.net>

References

https://en.wikipedia.org/wiki/Fibonacci_number

Examples

```
Fibonacci(0)                          # 1
Fibonacci(2)                          # 2
Fibonacci(0:3)                        # 0 1 1 2


# Golden ratio
F <- Fibonacci(1:25)                  # ... 75025 121393
f25 <- F[25]/F[24]                    #     1.618033989
phi <- (sqrt(5) + 1)/2
abs(f25 - phi)                        # 7.945178e-11

# Fibonacci numbers without iteration
fibo <- function(n) {
  phi <- (sqrt(5) + 1)/2
  fib <- (phi^(n+1) - (1-phi)^(n+1)) / (2*phi - 1)
  round(fib)
}

fibo(30:33)                           # 1346269 2178309 3524578 5702887
```

FindColor *Get Color on a Defined Color Range*

Description

Find a color on a defined color range depending on the value of x. This is helpful for colorcoding numeric values.

Usage

```
FindColor(x, cols = rev(heat.colors(100)),
          min.x = NULL, max.x = NULL, all.inside = FALSE)
```

Arguments

x	numeric.
cols	a vector of colors.
min.x	the x-value to be used for the left edge of the first color. If left to the default NULL min(pretty(x)) will be used.
max.x	the x-value to be used for the right edge of the last color. If left to the default NULL max(pretty(x)) will be used.
all.inside	logical; if true, the returned indices are coerced into 1, ..., N-1, i.e., 0 is mapped to 1 and N to N-1.

Details

For the selection of colors the option `rightmost.closed` in the used function `findInterval` is set to TRUE. This will ensure that all values on the right edge of the range are assigned a color. How values outside the boundaries of min.x and max.x should be handled can be controlled by `all.inside`. Set this value to TRUE, if those values should get the colors at the edges or set it to FALSE, if they should remain white (which is the default).

Note that `findInterval` closes the intervals on the left side, e.g. [0, 1). This option can't be changed. Consequently will x-values lying on the edge of two colors get the color of the bigger one.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`findInterval`

Examples

```
Canvas(7, main="Use of function FindColor()")

# get some data
x <- c(23, 56, 96)
# get a color range from blue via white to red
cols <- colorRampPalette(c("blue","white","red"))(100)
ColorLegend(x="bottomleft", cols=cols, labels=seq(0, 100, 10), cex=0.8)

# and now the color coding of x:
(xcols <- FindColor(x, cols, min.x=0, max.x=100))

# this should be the same as
cols[x+1]

# how does it look like?
y0 <- c(-5, -2, 1)
text(x=1, y=max(y0)+2, labels="Color coding of x:")
text(x=1.5, y=y0, labels=x)
DrawRegPolygon(x=3, y=y0, nv=4, rot=pi/4, col=xcols)
text(x=6, y=y0, labels=xcols)

# how does the function select colors?
Canvas(xlim = c(0,1), ylim = c(0,1))
cols <- c(red="red", yellow="yellow", green="green", blue="blue")
ColorLegend(x=0, y=1, width=1, cols=rev(cols), horiz = TRUE,
            labels=Format(seq(0, 1, .25), digits=2), frame="grey", cex=0.8 )
x <- c(-0.2, 0, 0.15, 0.55, .75, 1, 1.3)
arrows(x0 = x, y0 = 0.6, y1 = 0.8, angle = 15, length = .2)
text(x=x, y = 0.5, labels = x, adj = c(0.5,0.5))
text(x=x, y = 0.4, labels = names(FindColor(x, cols=cols,
   min.x = 0, max.x = 1, all.inside = TRUE)), adj = c(0.5,0.5))
```

```
text(x=x, y = 0.3, labels = names(FindColor(x, cols=cols,
   min.x = 0, max.x = 1, all.inside = FALSE)), adj = c(0.5,0.5))
```

FindCorr *Determine Highly Correlated Variables*

Description

This function searches through a correlation matrix and returns a vector of integers corresponding to columns to remove to reduce pair-wise correlations.

Usage

```
FindCorr(x, cutoff = .90, verbose = FALSE)
```

Arguments

x	A correlation matrix
cutoff	A numeric value for the pair-wise absolute correlation cutoff
verbose	A boolean for printing the details

Details

The absolute values of pair-wise correlations are considered. If two variables have a high correlation, the function looks at the mean absolute correlation of each variable and removes the variable with the largest mean absolute correlation.

There are several function in the **subselect** package that can also be used to accomplish the same goal. However the package was removed from CRAN and available in the archives.

Value

A vector of indices denoting the columns to remove. If no correlations meet the criteria, `numeric(0)` is returned.

Author(s)

Original R code by Dong Li, modified by Max Kuhn

References

Max Kuhn. Contributions from Jed Wing, Steve Weston, Andre Williams, Chris Keefer, Allan Engelhardt, Tony Cooper, Zachary Mayer and the R Core Team (2014). caret: Classification and Regression Training. R package version 6.0-35. https://cran.r-project.org/package=caret

Examples

```
corrMatrix <- diag(rep(1, 5))
corrMatrix[2, 3] <- corrMatrix[3, 2] <- .7
corrMatrix[5, 3] <- corrMatrix[3, 5] <- -.7
corrMatrix[4, 1] <- corrMatrix[1, 4] <- -.67

corrDF <- expand.grid(row = 1:5, col = 1:5)
corrDF$correlation <- as.vector(corrMatrix)
PlotCorr(xtabs(correlation ~ ., corrDF), las=1, border="grey")

FindCorr(corrMatrix, cutoff = .65, verbose = TRUE)

FindCorr(corrMatrix, cutoff = .99, verbose = TRUE)

# d.pizza example
m <- cor(data.frame(lapply(d.pizza, as.numeric)), use="pairwise.complete.obs")
FindCorr(m, verbose = TRUE)
m[, FindCorr(m)]
```

FisherZ *Fisher z-Transformation*

Description

Convert a correlation to a z score or z to r using the Fisher transformation.

Usage

```
FisherZ(rho)

FisherZInv(z)
```

Arguments

rho	the Pearson's correlation coefficient
z	a Fisher z transformed value

Details

The sampling distribution of Pearson's r is not normally distributed. Fisher developed a transformation now called "Fisher's z-transformation" that converts Pearson's r to the normally distributed variable z. The formula for the transformation is:

$$z_r = tanh^{-1}(r) = \frac{1}{2} log\left(\frac{1+r}{1-r}\right)$$

Value

z value corresponding to r (in FisherZ)
r corresponding to z (in FisherZInv)

Author(s)

William Revelle revelle@northwestern.edu,
slight modifications Andri Signorell andri@signorell.net based on R-Core code

See Also

cor.test

Examples

```
cors <- seq(-.9, .9, .1)

zs <- FisherZ(cors)
rs <- FisherZInv(zs)
round(zs, 2)
n <- 30
r <- seq(0, .9, .1)
rc <- t(sapply(r, CorCI, n=n))
t <- r * sqrt(n-2) / sqrt(1-r^2)
p <- (1 - pt(t, n-2)) / 2

r.rc <- data.frame(r=r, z=FisherZ(r), lower=rc[,2], upper=rc[,3], t=t, p=p)

round(r.rc,2)
```

FixToTable *Convert a Text to a Table*

Description

Convert a text to a table by using complete columns of spaces (or any other separator) as delimiting point.

Usage

```
FixToTable(txt, sep = " ", delim = "\t", trim = TRUE, header = TRUE)
```

Arguments

txt	the text to be partitioned. Works best, if txt is a matrix.
sep	the separator to use. Will frequently be " ".
delim	the new delimiter to insert. (default tab)
trim	logical. Should the separated text be trimmed from whitespace? Defaults to TRUE.
header	logical. Should the first line be interpreted as header?

Details

Only a complete appearance of the separator character in the same position over all rows will be accepted as column delimiter.

Value

a matrix of the separated text.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

StrChop

Examples

```
# let's get some tabbed text
txt <- matrix(capture.output(Titanic[,,2,1]))
FixToTable(txt[-1,])
```

Format *Format Numbers and Dates*

Description

Formatting numbers with base R tools often degenerates into a major intellectual challenge for us little minds down here in the valley of tears. There are a number of options available and quite often it's hard to work out which one to use, when a more uncommon setting is needed. The Format() function wraps all these functions and tries to offer a simpler, less technical, but still flexible interface.

There's also an easygoing interface for format templates, defined as a list consisting of any accepted format features. This enables to define templates globally and easily change or modify them later.

Usage

```
Format(x, digits = NULL, sci = NULL, big.mark = NULL,
       ldigits = NULL, zero.form = NULL, na.form = NULL,
       fmt = NULL, align = NULL, width = NULL, lang = NULL,
       eps = NULL, ...)

## S3 method for class 'table'
Format(x, digits = NULL, sci = NULL, big.mark = NULL,
       ldigits = NULL, zero.form = NULL, na.form = NULL,
       fmt = NULL, align = NULL, width = NULL, lang = NULL,
       eps = NULL, ...)

## S3 method for class 'matrix'
Format(x, digits = NULL, sci = NULL, big.mark = NULL,
       ldigits = NULL, zero.form = NULL, na.form = NULL,
       fmt = NULL, align = NULL, width = NULL, lang = NULL,
       eps = NULL, ...)

## Default S3 method:
Format(x, digits = NULL, sci = NULL, big.mark = NULL,
       ldigits = NULL, zero.form = NULL, na.form = NULL,
       fmt = NULL, align = NULL, width = NULL, lang = NULL,
       eps = NULL, ...)

Fmt(...)

as.fmt(...)

as.CDateFmt(fmt)
```

Arguments

x	an atomic numerical, typically a vector of real numbers or a matrix of numerical values. Factors will be converted to strings.
digits	integer, the desired (fixed) number of digits after the decimal point. Unlike formatC you will always get this number of digits even if the last digit is 0. Negative numbers of digits round to a power of ten (digits=-2 would round to the nearest hundred).
sci	integer. The power of 10 to be set when deciding to print numeric values in exponential notation. Fixed notation will be preferred unless the number is larger than 10^scipen. If just one value is set it will be used for the left border 10^(-scipen) as well as for the right one (10^scipen). A negative and a positive value can also be set independently. Default is getOption("scipen"), whereas scipen=0 is overridden.
big.mark	character; if not empty used as mark between every 3 decimals before the decimal point. Default is "" (none).

ldigits	number of leading zeros. ldigits=3 would make sure that at least 3 digits on the left side will be printed, say 3.4 will be printed as 003.4. Setting ldigits to 0 will yield results like .452 for 0.452. The default NULL will leave the numbers as they are (meaning at least one 0 digit).
zero.form	character, string specifying how zeros should be specially formatted. Useful for pretty printing 'sparse' objects. If set to NULL (default) no special action will be taken.
na.form	character, string specifying how NAs should be specially formatted. If set to NULL (default) no special action will be taken.
fmt	either a format string, allowing to flexibly define special formats or an object of class fmt, consisting of a list of Format arguments. See Details.
align	the character on whose position the strings will be aligned. Left alignment can be requested by setting sep = "\\l", right alignment by "\\r" and center alignment by "\\c". Mind the backslashes, as if they are omitted, strings would be aligned to the **character** l, r or c respectively. The default is NULL which would just leave the strings as they are. This argument is send directly to the function StrAlign() as argument sep.
width	integer, the defined fixed width of the strings.
lang	optional value setting the language for the months and daynames. Can be either "local" for current locale or "engl" for english. If left to NULL, the DescToolsOption "lang" will be searched for and if not found "local" will be taken as default.
eps	a numerical tolerance used mainly for formatting p values, those less than eps are formatted as "< [eps]" (where '[eps]' stands for format(eps, digits)). Default is .Machine$double.eps.
...	further arguments to be passed to or from methods.

Details

Format() is the workhorse here and formats numbers and dates.

The argument fmt is very flexible and is used to generate a variety of different formats. When x is a date, it can take ISO-8601-date-and-time-format codes consisting of (d, m and y for day, month or year) and defining the combination of day month and year representation. Repeating the specific code defines the degree of abbreviation. The format 'yyyy-mm-dd' would yield a date as 2020-10-12.

Date Codes

d	day of the month without leading zero (1 - 31)
dd	day of the month with leading zero (01 - 31)
ddd	abbreviated name for the day of the week (e.g. Mon) in the current user's language
dddd	full name for the day of the week (e.g. Monday) in the current user's language
m	month without leading zero (1 - 12)
mm	month with leading zero (01 - 12)
mmm	abbreviated month name (e.g. Jan) in the current user's language
mmmm	full month name (e.g. January) in the current user's language

y	year without century, without leading zero (0 - 99)
yy	year without century, with leading zero (00 - 99)
yyyy	year with century. For example: 2005

The function `as.CDateFmt()` converts ISO-8601 codes into the C-format codes used in base R. So `as.CDateFmt("yyyy mm dd")` yields `"%Y %m %d"`.

Even more variability is needed to display numeric values. For the most frequently used formats there are the following special codes available:

Code		
e	scientific	forces scientific representation of x, e.g. 3.141e-05. The number of digits, alignment and zero values are further respected.
eng	engineering	forces scientific representation of x, but only with powers that are a multiple of 3.
engabb	engineering abbr.	same as eng, but replaces the exponential representation by codes, e.g. M for mega (1e6). See `d.prefix`.
%	percent	will divide the given number by 100 and append the %-sign (without a separator).
p	p-value	will wrap the function `format.pval` and return a p-value format. Use eps to define the threshold to switch to a `< 000` representation.
frac	fractions	will (try to) convert numbers to fractions. So 0.1 will be displayed as 1/10. See `fractions()`.
*	significance	will produce a significance representation of a p-value consisting of * and ., while the breaks are set according to the used defaults e.g. in `lm` as [0, 0.001] = *** (0.001, 0.01] = ** (0.01, 0.05] = * (0.05, 0.1] = . (0.1,1] =
p*	p-value stars	will produce p-value and significance stars

`fmt` can as well be an object of class `fmt` consisting of a list out of the arguments above. This allows to store and manage the full format in variables or as options (in `DescToolsOptions()`) and use it as format template subsequently.

Finally `fmt` can also be a function in x, which makes formatting very flexible.

New formats can be created by means of `as.fmt()`. This works quite straight on. We can use any of the arguments from `Format()` and combine them to a list.
The following code will define a new format template named `"myNumFmt"` of the class `"fmt"`. Provided to `Format()` this will result in a number displayed with 2 fixed digits and a comma as big mark:

```
myNumFmt <- as.fmt(digits=2, big.mark=",")
Format(12222.89345, fmt=myNumFmt) = 12,222.89
```

The latter returns the same result as if the arguments would have been supplied directly: `Format(12222.89345, digits=2, big.mark=",")`.

Many report functions (e.g. `TOne()`) in **DescTools** use three default formats for counts (named `"abs"`), numeric values (`"num"`) and percentages (`"per"`). These formats can be set by the user as options (see `DescToolsOptions()`. For other purposes any number of any named formats can be defined.

`Fmt()` is used to access and edit already defined Formats. It can directly adapt defined properties and returns the format template. `Fmt("num", digits=1, sci=10)` will use the current version of the numeric format and change the digits to 1 and the threshold to switch to scientifc presentation to numbers >1e10 and <1e-10. Format templates can be altered using their names. With `Fmt(abs=Fmt("abs", big.mark=" "))` the format template for count values `"abs"` will be overwritten with the new values and stored as option for the current session.

The formats can as well be organized as options. `DescToolsOptions("fmt")` would display the currently defined formats. This mechanic works analogously to the `options()` procedure of base R. So to store the current settings we can use

```
opt <- DescToolsOptions("fmt")
... do some stuff like redefining the global formats ...
DescToolOptions(opt)
```

The last command resets the options and so we have again the initial definitions for the format templates.

Value

the formatted values as characters.
If `x` was a `matrix`, then a the result will also be a `matrix`. (Hope this will not surprise you...)

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`format`, `formatC`, `prettyNum`, `sprintf`, `symnum`,
`StrAlign`, `StrPad`, `Sys.setlocale`,
`Weekday`, `Month`, `DescToolsOptions`

Examples

```
Format(as.Date(c("2014-11-28", "2014-1-2")), fmt="ddd, d mmmm yyyy")
Format(as.Date(c("2014-11-28", "2014-1-2")), fmt="ddd, d mmmm yyyy", lang="engl")

x <- pi * 10^(-10:10)

Format(x, digits=3, fmt="%", sci=NA)
Format(x, digits=4, sci=c(4, 6), ldigits=0, width=9, align=".")
```

```
# format a matrix
m <- matrix(runif(100), nrow=10,
            dimnames=list(LETTERS[1:10], LETTERS[1:10]))

Format(m, digits=1)

# engineering format
Format(x, fmt="eng",  digits=2)
Format(x, fmt="engabb", ldigits=2, digits=2)
# combine with grams [g]
paste(Format(x, fmt="engabb", ldigits=2, digits=2), "g", sep="")

# example form symnum
pval <- rev(sort(c(outer(1:6, 10^-(1:3)))))
noquote(cbind(Format(pval, fmt="p"), Format(pval, fmt="*")))

# special formats, e.g. confidence intervals
Format(c(3.2, 4.5, 6.7)/100, fmt="ci%", digits=2, conf.level=0.90)

# use Fmt() to get and define new formats stored as option
Fmt()                        # all defined formats
Fmt("abs")                   # only format named "abs"
Fmt("nexist")                # only format named "nexist" (nonexisting)
Fmt("abs", "per", "nexist")
Fmt("abs", digits=3)         # get Fmt("abs") and overwrite digits
Fmt("abs", na.form="-")      # get Fmt("abs") and add user defined na.form

# define totally new format and store as option
Fmt(nob=as.fmt(digits=10, na.form="nodat"))

# overwrite an existing format
Fmt(nob=Fmt("nob", digits=5))
Fmt("nob")

# change the character to be used as the decimal point
opt <- options(OutDec=",")
Format(1200, digits=2, big.mark = ".")
options(opt)
```

Frac *Fractional Part and Maximal Digits of a Numeric Value*

Description

`Frac()` returns the fractional part of a numeric value. `MaxDigits()` return the number of digits in `x`.
`Ndec()` returns the number of decimals.
`Prec()` returns the precision of a number `x`.

Usage

```
Frac(x, dpwr = NA)
MaxDigits(x)
Ndec(x)
Prec(x)
```

Arguments

x	the numeric value (or a vector of numerics), whose fractional part is to be calculated.
dpwr	power of 10 for a factor z, the fractional part will be multiplied with. The result will be returned rounded to integer. Defaults to NA and will then be ignored.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

format.info, as.integer, trunc

Examples

```
x <- rnorm(5)*100
x
Frac(x)

# multiply by 10^4
Frac(x, dpwr=4)

MaxDigits(c(1.25, 1.8, 12.0, 1.00000))

x <- c("0.0000", "0", "159.283", "1.45e+10", "1.4599E+10" )
Ndec(x)
Prec(as.numeric(x))
```

Frechet *The Frechet Distribution*

Description

Density function, distribution function, quantile function and random generation for the Frechet distribution with location, scale and shape parameters.

Usage

```
dFrechet(x, loc=0, scale=1, shape=1, log = FALSE)
pFrechet(q, loc=0, scale=1, shape=1, lower.tail = TRUE)
qFrechet(p, loc=0, scale=1, shape=1, lower.tail = TRUE)
rFrechet(n, loc=0, scale=1, shape=1)
```

Arguments

x, q	Vector of quantiles.
p	Vector of probabilities.
n	Number of observations.
loc, scale, shape	Location, scale and shape parameters (can be given as vectors).
log	Logical; if TRUE, the log density is returned.
lower.tail	Logical; if TRUE (default), probabilities are P[X <= x], otherwise, P[X > x]

Details

The Frechet distribution function with parameters $loc = a$, $scale = b$ and $shape = s$ is

$$G(z) = \exp\left\{-\left(\frac{z-a}{b}\right)^{-s}\right\}$$

for $z > a$ and zero otherwise, where $b > 0$ and $s > 0$.

Value

dFrechet gives the density function, pFrechet gives the distribution function, qFrechet gives the quantile function, and rFrechet generates random deviates.

Author(s)

Alec Stephenson <alec_stephenson@hotmail.com>

See Also

rGenExtrVal, rGumbel, rRevWeibull

Examples

```
dFrechet(2:4, 1, 0.5, 0.8)
pFrechet(2:4, 1, 0.5, 0.8)
qFrechet(seq(0.9, 0.6, -0.1), 2, 0.5, 0.8)
rFrechet(6, 1, 0.5, 0.8)
p <- (1:9)/10
pFrechet(qFrechet(p, 1, 2, 0.8), 1, 2, 0.8)
## [1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
```

Freq *Frequency Table for a Single Variable*

Description

Calculates absolute and relative frequencies of a vector x. Continuous (numeric) variables will be cut using the same logic as used by the function hist. Categorical variables will be aggregated by table. The result will contain single and cumulative frequencies for both, absolute values and percentages.

Usage

```
Freq(x, breaks = hist(x, plot = FALSE)$breaks, include.lowest = TRUE,
     ord = c("level", "desc", "asc", "name"),
     useNA = c("no", "ifany", "always"), ...)

## S3 method for class 'Freq'
print(x, digits = NULL, ...)
```

Arguments

x	the variable to be described, can be any atomic type.
breaks	either a numeric vector of two or more cut points or a single number (greater than or equal to 2) giving the number of intervals into which x is to be cut. Default taken from the function hist(). This is ignored if x is not of numeric type.
include.lowest	logical, indicating if an x[i] equal to the lowest (or highest, for right = FALSE) "breaks" value should be included. Ignored if x is not of numeric type.
ord	how should the result be ordered? Default is "level", other choices are 'by frequency' ("descending" or "ascending") or 'by name of the levels' ("name"). The argument can be abbreviated. This is ignored if x is numeric.
useNA	one out of "no", "ifany", "always". Defines whether to include extra NA levels in the table. Defaults to "no" which is the table() default too.
digits	integer, determining the number of digits used to format the relative frequencies.
...	further arguments are passed to the function cut(). Use dig.lab to control the format of numeric group names. Use the argument right to define if the intervals should be closed on the right (and open on the left) or vice versa. In print.Freq the dots are not used.

Details

By default only the valid cases are considered for the frequencies, say NA values are excluded. (This is in accordance with the default behavior of the R function table, which seemed a reasonable reference.) If the NAs should be included you can set the useNA argument to either "ifany" or "always".

For numeric variables, if `breaks` is specified as a single number, the range of the data is divided into breaks pieces of equal length, and then the outer limits are moved away by 0.1% of the range to ensure that the extreme values both fall within the break intervals. (If `x` is a constant vector, equal-length intervals are created that cover the single value.) See `cut`.

Value

an object of type `"Freq"`, which is basically a data.frame with 5 columns (earning a specific print routine), containing the following components:

`level`	factor. The levels of the grouping variable.
`freq`	integer. The absolute frequencies.
`perc`	numeric. The relative frequencies (percent).
`cumfreq`	integer. The cumulative sum of the absolute frequencies.
`cumperc`	numeric. The cumulative sum of the relative frequencies.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`cut`, `hist`, `cumsum`, `table`, `prop.table`, `PercTable`, `Freq2D`

Examples

```
data(d.pizza)

# result is a data.frame
d.freq <- Freq(d.pizza$price)
d.freq

# it is printed by default with 3 digits for the percent values,
# but the number of digits can be defined in the print function
print(d.freq, digits=5)

# sorted by frequency
Freq(d.pizza$driver, ord="desc")

# sorted by name using all the observations, say including NAs
Freq(d.pizza$driver, ord="name", useNA="ifany")

# percentages and cumulative frequencies for a vector of count data
Freq(as.table(c(2,4,12,8)))
```

Freq2D *Bivariate (Two-Dimensional) Frequency Distribution*

Description

Calculate a frequency distribution for two continuous variables.

Usage

```
Freq2D(x, ...)

## S3 method for class 'formula'
Freq2D(formula, data, subset, ...)

## Default S3 method:
Freq2D(x, y, n=20, pad=0, dnn=NULL, ...)
```

Arguments

x	a vector of x values, or a data frame whose first two columns contain the x and y values.
y	a vector of y values.
formula	a formula, such as y~x.
data	a data.frame, matrix, or list from which the variables in formula should be taken.
subset	an optional vector specifying a subset of observations to be used.
n	the desired number of bins for the output, a scalar or a vector of length 2.
pad	number of rows and columns to add to each margin, containing only zeros.
dnn	the names to be given to the dimensions in the result.
...	named arguments to be passed to the default method.

Details

The exact number of bins is determined by the pretty function, based on the value of n.

Padding the margins with zeros can be helpful for subsequent analysis, such as smoothing.

The print logical flag only has an effect when layout=1.

Value

The layout argument specifies one of the following formats for the binned frequency output:

1. matrix that is easy to read, aligned like a scatterplot.
2. list with three elements (x, y, matrix) that can be passed to various plotting functions.
3. data.frame with three columns (x, y, frequency) that can be analyzed further.

Author(s)

Arni Magnusson <thisisarni@gmail.com»

See Also

cut, table, and print.table are the basic underlying functions.
Freq, PercTable

Examples

```
Freq2D(quakes$long, quakes$lat, dnn="")
Freq2D(lat ~ long, quakes, n=c(10, 20), pad=1)

# range(Freq2D(saithe, print=FALSE))

# Layout, plot
# Freq2D(saithe, layout=2)
# Freq2D(saithe, layout=3)
# contour(Freq2D(saithe, layout=2))
# lattice::contourplot(Freq ~ Bio + HR, Freq2D(saithe,layout=3))
```

GCD, LCM *Greatest Common Divisor and Least Common Multiple*

Description

Calculates the greatest common divisor (GCD) and least common multiple (LCM) of all the values present in its arguments.

Usage

```
GCD(..., na.rm = FALSE)
LCM(..., na.rm = FALSE)
```

Arguments

...	integer or logical vectors.
na.rm	logical. Should missing values (including NaN) be removed?

Details

The computation is based on the Euclidean algorithm without using the extended version.The greatest common divisor for all numbers in the integer vector x will be computed (the multiple GCD).

Value

A numeric (integer) value.

Note

The following relation is always true:

`n * m = GCD(n, m) * LCM(n, m)`

Author(s)

Dirk Eddelbuettel <edd@debian.org> (RCPP part), Andri Signorell <andri@signorell.net>, originally based on code in package **numbers** by Hans W Borchers <hwborchers@googlemail.com>

References

Eddelbuettel, D. (2013). Seamless R and C++ Integration with Rcpp. New York, NY: Springer.

See Also

`Factorize`, `Primes`, `IsPrime`

Examples

```
GCD(12, 10)
GCD(144, 233)    # Fibonacci numbers are relatively prime to each other

LCM(12, 10)
LCM(144, 233)    # = 144 * 233

# all elements will be flattened by unlist
GCD(2, 3, c(5, 7) * 11)
GCD(c(2*3, 3*5, 5*7))
LCM(c(2, 3, 5, 7) * 11)
LCM(2*3, 3*5, 5*7)
```

GenExtrVal *The Generalized Extreme Value Distribution*

Description

Density function, distribution function, quantile function and random generation for the generalized Extreme value (GenExtrVal) distribution with location, scale and shape parameters.

Usage

```
dGenExtrVal(x, loc=0, scale=1, shape=0, log = FALSE)
pGenExtrVal(q, loc=0, scale=1, shape=0, lower.tail = TRUE)
qGenExtrVal(p, loc=0, scale=1, shape=0, lower.tail = TRUE)
rGenExtrVal(n, loc=0, scale=1, shape=0)
```

Arguments

x, q	Vector of quantiles.
p	Vector of probabilities.
n	Number of observations.
loc, scale, shape	Location, scale and shape parameters; the shape argument cannot be a vector (must have length one).
log	Logical; if TRUE, the log density is returned.
lower.tail	Logical; if TRUE (default), probabilities are P[X <= x], otherwise, P[X > x]

Details

The GenExtrVal distribution function with parameters $loc = a$, $scale = b$ and $shape = s$ is

$$G(z) = \exp\left[-\{1 + s(z - a)/b\}^{-1/s}\right]$$

for $1 + s(z - a)/b > 0$, where $b > 0$. If $s = 0$ the distribution is defined by continuity. If $1 + s(z - a)/b \leq 0$, the value z is either greater than the upper end point (if $s < 0$), or less than the lower end point (if $s > 0$).

The parametric form of the GenExtrVal encompasses that of the Gumbel, Frechet and reverse Weibull distributions, which are obtained for $s = 0$, $s > 0$ and $s < 0$ respectively. It was first introduced by Jenkinson (1955).

Value

dGenExtrVal gives the density function, pGenExtrVal gives the distribution function, qGenExtrVal gives the quantile function, and rGenExtrVal generates random deviates.

Author(s)

Alec Stephenson <alec_stephenson@hotmail.com>

References

Jenkinson, A. F. (1955) The frequency distribution of the annual maximum (or minimum) of meteorological elements. *Quart. J. R. Met. Soc.*, **81**, 158–171.

See Also

rFrechet, rGumbel, rRevWeibull

Examples

```
dGenExtrVal(2:4, 1, 0.5, 0.8)
pGenExtrVal(2:4, 1, 0.5, 0.8)
qGenExtrVal(seq(0.9, 0.6, -0.1), 2, 0.5, 0.8)
rGenExtrVal(6, 1, 0.5, 0.8)
p <- (1:9)/10
pGenExtrVal(qGenExtrVal(p, 1, 2, 0.8), 1, 2, 0.8)
## [1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
```

GenPareto *The Generalized Pareto Distribution*

Description

Density function, distribution function, quantile function and random generation for the generalized Pareto distribution (GenPareto) with location, scale and shape parameters.

Usage

```
dGenPareto(x, loc=0, scale=1, shape=0, log = FALSE)
pGenPareto(q, loc=0, scale=1, shape=0, lower.tail = TRUE)
qGenPareto(p, loc=0, scale=1, shape=0, lower.tail = TRUE)
rGenPareto(n, loc=0, scale=1, shape=0)
```

Arguments

x, q	Vector of quantiles.
p	Vector of probabilities.
n	Number of observations.
loc, scale, shape	Location, scale and shape parameters; the shape argument cannot be a vector (must have length one).
log	Logical; if TRUE, the log density is returned.
lower.tail	Logical; if TRUE (default), probabilities are P[X <= x], otherwise, P[X > x]

Details

The generalized Pareto distribution function (Pickands, 1975) with parameters $loc = a$, $scale = b$ and $shape = s$ is

$$G(z) = 1 - \{1 + s(z - a)/b\}^{-1/s}$$

for $1 + s(z - a)/b > 0$ and $z > a$, where $b > 0$. If $s = 0$ the distribution is defined by continuity.

Value

dGenPareto gives the density function, pGenPareto gives the distribution function, qGenPareto gives the quantile function, and rGenPareto generates random deviates.

Author(s)

Alec Stephenson <alec_stephenson@hotmail.com>

References

Pickands, J. (1975) Statistical inference using Extreme Order statistics. *Annals of Statistics*, **3**, 119–131.

See Also

rGenExtrVal

Examples

```
dGenPareto(2:4, 1, 0.5, 0.8)
pGenPareto(2:4, 1, 0.5, 0.8)
qGenPareto(seq(0.9, 0.6, -0.1), 2, 0.5, 0.8)
rGenPareto(6, 1, 0.5, 0.8)
p <- (1:9)/10
pGenPareto(qGenPareto(p, 1, 2, 0.8), 1, 2, 0.8)
## [1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
```

GenRandGroups *Generate Random Groups*

Description

Generates a random grouping from a given data vector, where the group sizes correspond to the numeric vector grp_n.

Usage

```
GenRandGroups(x, grp_n)
```

Arguments

x	a vector containing the objects which should be grouped
grp_n	an integer vector with the required group sizes

Details

For group divisions in class, it is often useful to have a function available that randomizes these divisions.

Value

a list sized length of grp_n with the x elements assigned to their group.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

CombN, CombSet

Examples

```
# say we have 12 students and want 3 groups with sizes 4,3, and 5

GenRandGroups(x=LETTERS[1:12], grp_n=c(4,3,5))
```

GeomSn *Geometric Series*

Description

A geometric sequence is a sequence, such that each term is given by a multiple of q of the previous one. A geometric series consists out of the sum of all former values of a geometric sequence..

Usage

```
GeomSn(a1, q, n)
```

Arguments

a1	the first element of the sequence
q	the factor of the sequence
n	number of elements to include in the sum

Value

the sum as numeric value

Author(s)

Andri Signorell <andri@signorell.net>

See Also

sum

Examples

```
GeomSn(a1=3, q=2, n=5)

# calculates the sum of the first 5 elements of the sequence
(gseq <- 3 * (2^(0:5)))
sum(gseq)

GeomSn(a1=3, q=2, n=0:5)
```

GeomTrans *Geometric Transformations*

Description

This function transforms geometric structures by translating, scaling and/or rotating them.

Usage

```
GeomTrans(x, y = NULL, trans = 0, scale = 1, theta = 0)
```

Arguments

x, y	vectors containing the coordinates of the vertices of the polygon , which has to be transformed. The coordinates can be passed in a plotting structure (a list with x and y components), a two-column matrix, ... See xy.coords.
trans	a vector of two values for the translation in x-, resp. y-direction. If only one value is supplied it will be recycled.
scale	a vector of two values for the scaling factor in x-, resp. y-direction. If only one value is supplied it will be recycled.
theta	angle of the rotation in radians starting from 3 o'clock counterclockwise.

Value

The function invisibly returns a list of the coordinates for the transformed shape(s).

Author(s)

Andri Signorell <andri@signorell.net>

See Also

polygon, DrawRegPolygon, DrawEllipse, DrawArc

Examples

```
# let's have a triangle
Canvas(main="Rotation")
x <- DrawRegPolygon(nv=3)[[1]]

xt <- GeomTrans(x, trans=c(1, 3), scale=c(2, 2), theta=pi/4)
polygon(xt)
```

GetCalls *Return All Used Functions Within a Function*

Description

For screening purposes it can be useful to get a list of all function calls our function may depend on. `GetCall()` parses the function source and return all found function calls grouped by their package.

Usage

```
GetCalls(fun, alphabetic = TRUE, package = NULL)
```

Arguments

`fun`	the name of the function to be parsed
`alphabetic`	logic, determining the order of the result
`package`	name of the package, if only functions of this specific package should be returned.

Value

a list of vectors structered by package

Author(s)

Nicholas Cooper <njcooper at gmx.co.uk> (in package NCmisc) with some tweaking by Andri Signorell <andri@signorell.net>

See Also

`LsFct()`

Examples

```
GetCalls("t.test.default")

sapply(c("Closest", "Format"),
       function(x) paste(unname(unlist(GetCalls(x))), collapse=", "))
```

GetCurrWrd *Get a Handle to a Running Word/Excel Instance*

Description

Look for a running Word, resp. Excel instance and return its handle. If no running instance is found a new instance will be created (which will be communicated with a warning).

Usage

```
GetCurrWrd()
GetCurrXL()
```

Value

a handle (pointer) to the running Word, resp. Excel instance.

Note

When closing an application instance, the value of the pointer in R is not somehow automatically invalidated. In such cases the corresponding variable contains an invalid address. Whether the pointer still refers to a valid running application instance can be checked by IsValidHwnd.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

GetNewWrd, IsValidHwnd

Examples

```
## Not run: # Windows-specific example

# Start a new instance
GetNewWrd()

# grab the handle to this instance
wrd <- GetCurrWrd()

# this should be valid
IsValidHwnd(wrd)

# close the instance
wrd$quit()

# now it should be gone and the pointer invalid
if(IsValidHwnd(wrd)){
```

```
  print("Ouups! Still there?")
} else {
  print("GetCurrWrd: no running word instance found...")
}

## End(Not run)
```

GetNewWrd *Create a New Word Instance*

Description

Start a new instance of Word and return its handle. By means of this handle we can then control the word application.
`WrdKill` ends a running MS-Word task.

Usage

```
GetNewWrd(visible = TRUE, template = "Normal", header = FALSE,
          main = "Descriptive report")

WrdKill()
```

Arguments

`visible`	logical, should Word made visible? Defaults to `TRUE`.
`template`	the name of the template to be used for creating a new document.
`header`	logical, should a caption and a list of contents be inserted? Default is `FALSE`.
`main`	the main title of the report

Details

The package **RDCOMClient** reveals the whole VBA-world of MS-Word. So generally speaking any VBA code can be run fully controlled by R. In practise, it might be a good idea to record a macro and rewrite the VB-code in R.

Here's a list of some frequently used commands. Let's assume we have a handle to the application and a handle to the current selection defined as:

```
wrd <- GetNewWrd()
sel <- wrd$Selection()
```

Then we can access the most common properties as follows:

new document	`wrd[["Documents"]]$Add(template, FALSE, 0)`, template is the templatename.
open document	`wrd[["Documents"]]$Open(Filename="C:/MyPath/MyDocument.docx").`
save document	`wrd$ActiveDocument()$SaveAs2(FileName="P:/MyFile.docx")`
quit word	`wrd$quit()`
kill word task	`WrdKill` kills a running word task (which might not be ended with quit.)
normal text	Use `ToWrd` which offers many arguments as fontname, size, color, alignment etc. `ToWrd("Lorem ipsum dolor sit amet, consetetur", font=list(name="Arial", size=10, col=wdConst$wdColorRed)`
simple text	`sel$TypeText("sed diam nonumy eirmod tempor invidunt ut labore")`
heading	`WrdCaption("My Word-Story", index=1)`
insert R output	`ToWrd(capture.output(str(d.diamonds)))`
pagebreak	`sel$InsertBreak(wdConst$wdPageBreak)`
sectionbreak	`sel$InsertBreak(wdConst$wdSectionBreakContinuous)` `(wdSectionBreakNextPage)`
move cursor right	`sel$MoveRight(Unit=wdConst$wdCharacter, Count=2, Extend=wdConst$wdExtend)`
goto end	`sel$EndKey(Unit=wdConst$wdStory)`
pagesetup	`sel[["PageSetup"]][["Bottommargin"]] <- 4 * 72`
orientation	`sel[["PageSetup"]][["Orientation"]] <- wdConst$wdOrientLandscape`
add bookmark	`wrd[["ActiveDocument"]][["Bookmarks"]]$Add("myBookmark")`
goto bookmark	`sel$GoTo(wdConst$wdGoToBookmark, 0, 0, "myBookmark")`
update bookmark	`WrdUpdateBookmark("myBookmark", "New text for my bookmark")`
show document map	`wrd[["ActiveWindow"]][["DocumentMap"]] <- TRUE`
create table	`WrdTable()` which allows to define the table's geometry
insert caption	`sel$InsertCaption(Label="Abbildung", TitleAutoText="InsertCaption", Title="My Title")`
tables of figures	`wrd$ActiveDocument()$TablesOfFigures()$Add(Range=sel$range(), Caption="Figure")`
insert header	`wview <- wrd[["ActiveWindow"]][["ActivePane"]][["View"]][["SeekView"]]` `wview <- ifelse(header, wdConst$wdSeekCurrentPageHeader, wdConst$wdSeekCurrentPageFo` `ToWrd(x, ..., wrd=wrd)`

Value

a handle (pointer) to the created Word instance.

Note

Note that the list of contents has to be refreshed by hand after inserting text (if inserted by `header = TRUE`).

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`GetNewXL`, `GetNewPP`

Examples

```
## Not run:  # Windows-specific example

wrd <- GetNewWrd()
Desc(d.pizza[,1:4], wrd=wrd)

wrd <- GetNewWrd(header=TRUE)
Desc(d.pizza[,1:4], wrd=wrd)

# enumerate all bookmarks in active document
for(i in 1:wrd[["ActiveDocument"]][["Bookmarks"]]$count()){
  print(wrd[["ActiveDocument"]][["Bookmarks"]]$Item(i)$Name())
}

## End(Not run)
```

GetNewXL *Create a New Excel Instance*

Description

Start a new instance of Excel and return its handle. This is needed to address the Excel application and objects afterwards.

Usage

```
GetNewXL(visible = TRUE, newdoc = TRUE)
```

Arguments

visible	logical, should Excel made visible? Defaults to TRUE.
newdoc	logical, determining if a new workbook should be created. Defaults to TRUE.

Details

Here's a list of some frequently used commands.
Let's assume:

```
xl <- GetNewXL()
```

workbooks	`xl$workbooks()$count()`
quit excel	`xl$quit()`

Author(s)

Andri Signorell <andri@signorell.net>

See Also

XLView, XLGetRange, XLGetWorkbook

Examples

```
## Not run:  # Windows-specific example
# get a handle to a new excel instance
xl <- GetNewXL()

## End(Not run)
```

Gini *Gini Coefficient*

Description

Compute the Gini coefficient, the most commonly used measure of inequality.

Usage

```
Gini(x, weights = NULL, unbiased = TRUE,
     conf.level = NA, R = 10000, type = "bca", na.rm = FALSE)
```

Arguments

x	a vector containing at least non-negative elements. The result will be NA, if x contains negative elements.
weights	a numerical vector of weights the same length as x giving the weights to use for elements of x.
unbiased	logical. In order for G to be an unbiased estimate of the true population value, calculated gini is multiplied by $n/(n-1)$. Default is TRUE. (See Dixon, 1987)
conf.level	confidence level for the confidence interval, restricted to lie between 0 and 1. If set to TRUE the bootstrap confidence intervals are calculated. If set to NA (default) no confidence intervals are returned.
R	number of bootstrap replicates. Usually this will be a single positive integer. For importance resampling, some resamples may use one set of weights and others use a different set of weights. In this case R would be a vector of integers where each component gives the number of resamples from each of the rows of weights. This is ignored if no confidence intervals are to be calculated.
type	character string representing the type of interval required. The value should be one out of the c("norm","basic", "stud", "perc" or "bca"). This argument is ignored if no confidence intervals are to be calculated.
na.rm	logical. Should missing values be removed? Defaults to FALSE.

Details

The range of the Gini coefficient goes from 0 (no concentration) to $\sqrt{(\frac{n-1}{n})}$ (maximal concentration). The bias corrected Gini coefficient goes from 0 to 1.
The small sample variance properties of the Gini coefficient are not known, and large sample approximations to the variance of the coefficient are poor (Mills and Zandvakili, 1997; Glasser, 1962; Dixon et al., 1987), therefore confidence intervals are calculated via bootstrap re-sampling methods (Efron and Tibshirani, 1997).
Two types of bootstrap confidence intervals are commonly used, these are percentile and bias-corrected (Mills and Zandvakili, 1997; Dixon et al., 1987; Efron and Tibshirani, 1997). The bias-corrected intervals are most appropriate for most applications. This is set as default for the `type` argument (`"bca"`). Dixon (1987) describes a refinement of the bias-corrected method known as 'accelerated' - this produces values very closed to conventional bias corrected intervals.
(Iain Buchan (2002) *Calculating the Gini coefficient of inequality*, see: https://www.statsdirect.com/help/default.htm#nonparametric_methods/gini.htm)

Value

If `conf.level` is set to `NA` then the result will be

`a`	single numeric value

and if a `conf.level` is provided, a named numeric vector with 3 elements:

`gini`	Gini coefficient
`lwr.ci`	lower bound of the confidence interval
`upr.ci`	upper bound of the confidence interval

Author(s)

Andri Signorell <andri@signorell.net>

References

Cowell, F. A. (2000) Measurement of Inequality in Atkinson, A. B. / Bourguignon, F. (Eds): *Handbook of Income Distribution*. Amsterdam.

Cowell, F. A. (1995) *Measuring Inequality* Harvester Wheatshef: Prentice Hall.

Marshall, Olkin (1979) *Inequalities: Theory of Majorization and Its Applications*. New York: Academic Press.

Glasser C. (1962) Variance formulas for the mean difference and coefficient of concentration. *Journal of the American Statistical Association* 57:648-654.

Mills JA, Zandvakili A. (1997). Statistical inference via bootstrapping for measures of inequality. *Journal of Applied Econometrics* 12:133-150.

Dixon, PM, Weiner J., Mitchell-Olds T, Woodley R. (1987) Boot-strapping the Gini coefficient of inequality. *Ecology* 68:1548-1551.

Efron B, Tibshirani R. (1997) Improvements on cross-validation: The bootstrap method. *Journal of the American Statistical Association* 92:548-560.

See Also

See Herfindahl, Rosenbluth for concentration measures, Lc for the Lorenz curve ineq() in the package **ineq** contains additional inequality measures

Examples

```
# generate vector (of incomes)
x <- c(541, 1463, 2445, 3438, 4437, 5401, 6392, 8304, 11904, 22261)

# compute Gini coefficient
Gini(x)

# working with weights
fl <- c(2.5, 7.5, 15, 35, 75, 150)    # midpoints of classes
n  <- c(25, 13, 10, 5, 5, 2)          # frequencies

# with confidence intervals
Gini(x=fl, weights=n, conf.level=0.95, unbiased=FALSE)

# some special cases
x <- c(10, 10, 0, 0, 0)
plot(Lc(x))

Gini(x, unbiased=FALSE)

# the same with weights
Gini(x=c(10, 0), weights=c(2,3), unbiased=FALSE)

# perfect balance
Gini(c(10, 10, 10))
```

GiniSimpson *Gini-Simpson Coefficient, Gini-Deltas coefficient and Hunter-Gaston Index*

Description

Calculate the Gini-Simpson coefficient, the Gini variant proposed by Deltas and the Hunter-Gaston Index.

Usage

```
GiniSimpson(x, na.rm = FALSE)
GiniDeltas(x, na.rm = FALSE)

HunterGaston(x, na.rm = FALSE)
```

Arguments

x	a factor containing at least non-negative elements.
na.rm	logical. Should missing values be removed? Defaults to FALSE.

Details

The original Simpson index λ equals the probability that two entities taken at random from the dataset of interest (with replacement) represent the same type. The Simpson index was introduced in 1949 by Edward H. Simpson to measure the degree of concentration when individuals are classified into types. The same index was rediscovered by Orris C. Herfindahl in 1950. The square root of the index had already been introduced in 1945 by the economist Albert O. Hirschman. As a result, the same measure is usually known as the Simpson index in ecology, and as the Herfindahl index or the Herfindahl-Hirschman index (HHI) in economics.
Its transformation 1 - λ therefore equals the probability that the two entities represent different types. This measure is also known in ecology as the probability of interspecific encounter (PIE) and the Gini-Simpson index.

Value

a numeric value.

Author(s)

Andri Signorell <andri@signorell.net>

References

Cover Thomas M. and Thomas Joy A. (1991) *Elements of Information Theory*. Wiley.

Hunter, P., Gaston, A. G. (1988) Numerical Index of the Discriminatory Ability of Typing Systems: an Application of Simpson's Index of Diversity, *JOURNAL OF CLINICAL MICROBIOLOGY*, Nov. 1988, p. 2465-2466, 0095-1137/88/112465-02$02.00/0

Deltas (2003) DOI:10.1162/rest.2003.85.1.226.

See Also

DivCoef, Entropy, Gini, Herfindahl

Examples

```
x <- c(261,29,33,15,39,28,95,5,6,28,69,8,105,38,15)

GiniSimpson(x)

# is the same as
1 - Herfindahl(x)

GiniSimpson(c(783,121,112,70,201,153,425,19,37,126,325,51,442,193,41))
```

Gmean *Geometric Mean and Standard Deviation*

Description

Calculates the geometric mean, its confidence interval and the geometric standard deviation of a vector x.

Usage

```
Gmean(x, method = c("classic", "boot"), conf.level = NA,
      sides = c("two.sided","left","right"), na.rm = FALSE, ...)

Gsd(x, na.rm = FALSE)
```

Arguments

x	a positive numeric vector. An object which is not a vector is coerced (if possible) by as.vector.
method	a vector of character strings representing the type of intervals required. The value should be any subset of the values "classic", "boot". See boot.ci.
conf.level	confidence level of the interval. Default is NA.
sides	a character string specifying the side of the confidence interval, must be one of "two.sided" (default), "left" or "right". You can specify just the initial letter. "left" would be analogue to a hypothesis of "greater" in a t.test.
na.rm	logical, indicating whether NA values should be stripped before the computation proceeds. Defaults to FALSE.
...	further arguments are passed to the boot function. Supported arguments are type ("norm", "basic", "stud", "perc", "bca"), parallel and the number of bootstrap replicates R. If not defined those will be set to their defaults, being "basic" for type, option "boot.parallel" (and if that is not set, "no") for parallel and 999 for R.

Details

The geometric mean is defined as:

$$\sqrt[n]{x_1 \cdot x_2 \cdot x_3 \ldots \cdot x_n}$$

The geometric mean and geometric standard deviation are restricted to positive inputs (because otherwise the answer can have an imaginary component). Hence if any argument is negative, the result will be NA. If any argument is zero, then the geometric mean is zero.
For strict positive values the geometric mean is computed as exp(MeanCI(log(x))).

Considerations (Roenfeldt 2018) "The calculation of the geometric mean requires that all values are non-zero and positive. So what should you do if you have data that do not meet this requirement? If you have values that equal zero, you have a few options:

- Adjust your scale so that you add 1 to every number in the data set, and then subtract 1 from the resulting geometric mean.
- Ignore zeros or missing data in your calculations.
- Convert zeros to a very small number (often called "below the detection limit") that is less than the next smallest number in the data set.

If you have negative numbers, you will need to convert those numbers to a positive value before calculating the geometric mean. You can then assign the resulting geometric mean a negative value. If your data set contains both positive and negative values, you will have to separate them and find the geometric means for each group, and you can then find the weighted average of their individual geometric means to find the total geometric mean for the full data set. If none of these options appeals to you, you are not alone! There is controversy among statisticians about what is the best method for dealing with these values. You may want to calculate several types of averages and decide what makes the most sense for you and the results you are trying to report."

Value

a numeric value.

Author(s)

Andri Signorell <andri@signorell.net>

References

Snedecor, G. W., Cochran, W. G. Cochran (1989) Statistical Methods, 8th ed. Ames, *IA: Iowa State University Press*

Roenfeldt K. (2018) Better than Average: Calculating Geometric Means Using SAS, Henry M. Jackson Foundation for the Advancement of Military Medicine, https://www.lexjansen.com/wuss/2018/56_Final_Paper_PDF.pdf

See Also

mean, Hmean

Examples

```
x <- runif(5)
Gmean(x)

m <- matrix(runif(50), nrow = 10)
apply(m, 2, Gmean)

sapply(as.data.frame(m), Gmean)

# ..................................................
# example in https://www.stata.com/manuals13/rameans.pdf
x <- c(5,4,-4,-5,0,0,NA,7)

# positives only
```

```
Gmean(x[x>0], na.rm=TRUE, conf.level=0.95)

# add 5 to original values and remove zeros
Gmean(NAIfZero(x+5), na.rm=TRUE, conf.level = 0.95)
```

Gompertz *The Gompertz distribution*

Description

Density, distribution function, quantile function and random generation for the Gompertz distribution with unrestricted shape.

Usage

```
dGompertz(x, shape, rate = 1, log = FALSE)
pGompertz(q, shape, rate = 1, lower.tail = TRUE, log.p = FALSE)
qGompertz(p, shape, rate = 1, lower.tail = TRUE, log.p = FALSE)
rGompertz(n, shape = 1, rate = 1)
```

Arguments

x, q	vector of quantiles.
shape, rate	vector of shape and rate parameters.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P(X \leq x)$, otherwise, $P(X > x)$.
p	vector of probabilities.
n	number of observations. If length(n) > 1, the length is taken to be the number required.

Details

The Gompertz distribution with shape parameter a and rate parameter b has probability density function

$$f(x|a,b) = be^{ax}\exp(-b/a(e^{ax}-1))$$

For $a = 0$ the Gompertz is equivalent to the exponential distribution with constant hazard and rate b.

The probability distribution function is

$$F(x|a,b) = 1-\exp(-b/a(e^{ax}-1))$$

Thus if a is negative, letting x tend to infinity shows that there is a non-zero probability $1-\exp(b/a)$ of living forever. On these occasions qGompertz and rGompertz will return Inf.

Value

dGompertz gives the density, pGompertz gives the distribution function, qGompertz gives the quantile function, and rGompertz generates random deviates.

Note

Some implementations of the Gompertz restrict a to be strictly positive, which ensures that the probability of survival decreases to zero as x increases to infinity. The more flexible implementation given here is consistent with streg in Stata.

The functions dGompertz and similar available in the package **eha** label the parameters the other way round, so that what is called the shape there is called the rate here, and what is called 1 / scale there is called the shape here. The terminology here is consistent with the exponential dexp and Weibull dweibull distributions in R.

Author(s)

Christopher Jackson <chris.jackson@mrc-bsu.cam.ac.uk>

References

Stata Press (2007) Stata release 10 manual: Survival analysis and epidemiological tables.

See Also

dexp

GoodmanKruskalGamma *Goodman Kruskal's Gamma*

Description

Calculate Goodman Kruskal's Gamma statistic, a measure of association for ordinal factors in a two-way table.
The function has interfaces for a contingency table (matrix) and for single vectors (which will then be tabulated).

Usage

```
GoodmanKruskalGamma(x, y = NULL, conf.level = NA, ...)
```

Arguments

x	a numeric vector or a contingency table. A matrix will be treated as a table.
y	NULL (default) or a vector with compatible dimensions to x. If y is provided, table(x, y, ...) is calculated.
conf.level	confidence level of the interval. If set to NA (which is the default) no confidence intervals will be calculated.

... further arguments are passed to the function table, allowing i.e. to control the handling of NAs by setting the useNA argument. This refers only to the vector interface, the dots are ignored if x is a contingency table.

Details

The estimator of γ is based only on the number of concordant and discordant pairs of observations. It ignores tied pairs (that is, pairs of observations that have equal values of X or equal values of Y). Gamma is appropriate only when both variables lie on an ordinal scale.
It has the range [-1, 1]. If the two variables are independent, then the estimator of gamma tends to be close to zero. For 2×2 tables, gamma is equivalent to Yule's Q (YuleQ).
Gamma is estimated by

$$G = \frac{P - Q}{P + Q}$$

where P equals twice the number of concordances and Q twice the number of discordances.

Value

a single numeric value if no confidence intervals are requested,
and otherwise a numeric vector with 3 elements for the estimate, the lower and the upper confidence interval

Author(s)

Andri Signorell <andri@signorell.net>

References

Agresti, A. (2002) *Categorical Data Analysis*. John Wiley & Sons, pp. 57-59.

Brown, M.B., Benedetti, J.K.(1977) Sampling Behavior of Tests for Correlation in Two-Way Contingency Tables, *Journal of the American Statistical Association*, 72, 309-315.

Goodman, L. A., & Kruskal, W. H. (1954) Measures of association for cross classifications. *Journal of the American Statistical Association*, 49, 732-764.

Goodman, L. A., & Kruskal, W. H. (1963) Measures of association for cross classifications III: Approximate sampling theory. *Journal of the American Statistical Association*, 58, 310-364.

See Also

There's another implementation of gamma in **vcdExtra** GKgamma
ConDisPairs yields concordant and discordant pairs

Other association measures:
KendallTauA (tau-a), KendallTauB (tau-b), cor (method="kendall") for tau-b, StuartTauC (tau-c), SomersDelta
Lambda, GoodmanKruskalTau (tau), UncertCoef, MutInf

Examples

```
# example in:
# http://support.sas.com/documentation/cdl/en/statugfreq/63124/PDF/default/statugfreq.pdf
# pp. S. 1821 (149)

tab <- as.table(rbind(
  c(26,26,23,18, 9),
  c( 6, 7, 9,14,23))
  )

GoodmanKruskalGamma(tab, conf.level=0.95)
```

GoodmanKruskalTau *Goodman Kruskal's Tau*

Description

Calculate Goodman Kruskal's tau statistic, a measure of association for ordinal factors in a two-way table.
The function has interfaces for a table (matrix) and for single vectors.

Usage

```
GoodmanKruskalTau(x, y = NULL, direction = c("row", "column"), conf.level = NA, ...)
```

Arguments

x	a numeric vector or a table. A matrix will be treated as table.
y	NULL (default) or a vector with compatible dimensions to x. If y is provided, table(x, y, ...) is calculated.
direction	direction of the calculation. Can be "row" (default) or "column", where "row" calculates Goodman Kruskal's tau-a (R|C) ("column dependent").
conf.level	confidence level of the interval. If set to NA (which is the default) no confidence interval will be calculated.
...	further arguments are passed to the function table, allowing i.e. to set useNA. This refers only to the vector interface.

Details

Goodman-Kruskal tau measures association for cross tabulations of nominal level variables. Goodman-Kruskal tau is based on random category assignment. It measures the percentage improvement in predictability of the dependent variable (column or row variable) given the value of other variables (row or column variables). Goodman-Kruskal tau is the same as Goodman-Kruskal lambda except the calculations of the tau statistic are based on assignment probabilities specified by marginal or conditional proportions. Misclassification probabilities are based on random category assignment with probabilities specified by marginal or conditional proportion.

Goodman Kruskal tau reduces to ϕ^2 (see: Phi) in the 2x2-table case.

Value

a single numeric value if no confidence intervals are requested,
and otherwise a numeric vector with 3 elements for the estimate, the lower and the upper confidence interval

Author(s)

Andri Signorell <andri@signorell.net>, based on code from Antti Arppe <antti.arppe@helsinki.fi>

References

Agresti, A. (2002) *Categorical Data Analysis*. John Wiley & Sons, pp. 57-59.

Goodman, L. A., & Kruskal, W. H. (1954) Measures of association for cross classifications. *Journal of the American Statistical Association*, 49, 732-764.

Somers, R. H. (1962) A New Asymmetric Measure of Association for Ordinal Variables, *American Sociological Review*, 27, 799-811.

Goodman, L. A., & Kruskal, W. H. (1963) Measures of association for cross classifications III: Approximate sampling theory. *Journal of the American Statistical Association*, 58, 310-364.

Liebetrau, A. M. (1983) *Measures of Association*, Sage University Papers Series on Quantitative Applications in the Social Sciences, 07-004. Newbury Park, CA: Sage, pp. 24–30

See Also

`ConDisPairs` yields concordant and discordant pairs

Other association measures:
`KendallTauA` (Tau a), `cor` (method="kendall") for Tau b, `StuartTauC`, `GoodmanKruskalGamma`
`Lambda`, `UncertCoef`, `MutInf`

Examples

```
# example in:
# http://support.sas.com/documentation/cdl/en/statugfreq/63124/PDF/default/statugfreq.pdf
# pp. S. 1821

tab <- as.table(rbind(c(26,26,23,18,9),c(6,7,9,14,23)))

# Goodman Kruskal's tau C|R
GoodmanKruskalTau(tab, direction="column", conf.level=0.95)
# Goodman Kruskal's tau R|C
GoodmanKruskalTau(tab, direction="row", conf.level=0.95)

# http://support.sas.com/documentation/cdl/en/statugfreq/63124/PDF/default/statugfreq.pdf
# pp. 1814 (143)
tab <- as.table(cbind(c(11,2),c(4,6)))

GoodmanKruskalTau(tab, direction="row", conf.level=0.95)
GoodmanKruskalTau(tab, direction="column", conf.level=0.95)
# reduce both to:
```

```
Phi(tab)^2


# example 1 in Liebetrau (1983)

tt <- matrix(c(549,93,233,119,225,455,402,
               212,124,78,42,41,12,132,
               54,54,33,13,46,7,153), ncol=3,
            dimnames=list(rownames=c("Gov", "Mil", "Edu", "Eco", "Intel", "Rel", "For"),
                          colnames=c("One", "Two", "Multi")))

GoodmanKruskalTau(tt, direction = "row", conf.level = 0.95)
GoodmanKruskalTau(tt, direction = "column", conf.level = 0.95)


# SPSS
ttt <- matrix(c(225,53,206,3,1,12), nrow=3,
              dimnames=list(rownames=c("right","center", "left"),
                            colnames=c("us","ussr")))

round(GoodmanKruskalTau(ttt, direction = "r", con=0.95), d=3)
round(GoodmanKruskalTau(ttt, direction = "c"), d=3)
```

GTest *G-Test for Count Data*

Description

GTest performs chi-squared contingency table tests and goodness-of-fit tests.

Usage

```
GTest(x, y = NULL, correct = c("none", "williams", "yates"),
      p = rep(1/length(x), length(x)), rescale.p = FALSE)
```

Arguments

x	a numeric vector or matrix. x and y can also both be factors.
y	a numeric vector; ignored if x is a matrix. If x is a factor, y should be a factor of the same length.
correct	one out of "none" (default), "williams", "yates" . See Details.
p	a vector of probabilities of the same length of x. An error is given if any entry of p is negative.
rescale.p	a logical scalar; if TRUE then p is rescaled (if necessary) to sum to 1. If rescale.p is FALSE, and p does not sum to 1, an error is given.

Details

The G-test is also called "Likelihood Ratio Test" and is asymptotically equivalent to the Pearson ChiSquare-test but not usually used when analyzing 2x2 tables. It is used in logistic regression and loglinear modeling which involves contingency tables. The G-test is also reported in the standard summary of `Desc` for tables.

If `x` is a matrix with one row or column, or if `x` is a vector and `y` is not given, then a *goodness-of-fit test* is performed (`x` is treated as a one-dimensional contingency table). The entries of `x` must be non-negative integers. In this case, the hypothesis tested is whether the population probabilities equal those in `p`, or are all equal if `p` is not given.

If `x` is a matrix with at least two rows and columns, it is taken as a two-dimensional contingency table: the entries of `x` must be non-negative integers. Otherwise, `x` and `y` must be vectors or factors of the same length; cases with missing values are removed, the objects are coerced to factors, and the contingency table is computed from these. Then G-test is performed on the null hypothesis that the joint distribution of the cell counts in a 2-dimensional contingency table is the product of the row and column marginals.

Test of independence Yates' correction taken from Mike Camann's 2x2 G-test function. Goodness of Fit Yates' correction as described in Zar (2000).

Value

A list with class `"htest"` containing the following components:

`statistic`	the value the chi-squared test statistic.
`parameter`	the degrees of freedom of the approximate chi-squared distribution of the test statistic, `NA` if the p-value is computed by Monte Carlo simulation.
`p.value`	the p-value for the test.
`method`	a character string indicating the type of test performed, and whether Monte Carlo simulation or continuity correction was used.
`data.name`	a character string giving the name(s) of the data.
`observed`	the observed counts.
`expected`	the expected counts under the null hypothesis.

Author(s)

Pete Hurd <phurd@ualberta.ca>, Andri Signorell <andri@signorell.net> (tiny tweaks)

References

Hope, A. C. A. (1968) A simplified Monte Carlo significance test procedure. *J. Roy, Statist. Soc. B* **30**, 582–598.

Patefield, W. M. (1981) Algorithm AS159. An efficient method of generating r x c tables with given row and column totals. *Applied Statistics* **30**, 91–97.

Agresti, A. (2007) *An Introduction to Categorical Data Analysis, 2nd ed.*, New York: John Wiley & Sons. Page 38.

Sokal, R. R., F. J. Rohlf (2012) *Biometry: the principles and practice of statistics in biological research*. 4th edition. W. H. Freeman and Co.: New York. 937 pp.

See Also

chisq.test.

Examples

```
## From Agresti(2007) p.39
M <- as.table(rbind(c(762, 327, 468), c(484,239,477)))
dimnames(M) <- list(gender=c("M","F"),
                    party=c("Democrat","Independent", "Republican"))

(Xsq <- GTest(M))   # Prints test summary

Xsq$observed        # observed counts (same as M)
Xsq$expected        # expected counts under the null


## Testing for population probabilities
## Case A. Tabulated data
x <- c(A = 20, B = 15, C = 25)
GTest(x)
GTest(as.table(x))             # the same
x <- c(89,37,30,28,2)
p <- c(40,20,20,15,5)
try(
GTest(x, p = p)                # gives an error
)
# works
p <- c(0.40,0.20,0.20,0.19,0.01)
# Expected count in category 5
# is 1.86 < 5 ==> chi square approx.
GTest(x, p = p)                # maybe doubtful, but is ok!

## Case B. Raw data
x <- trunc(5 * runif(100))
GTest(table(x))                # NOT 'GTest(x)'!
```

Gumbel *The Gumbel Distribution*

Description

Density function, distribution function, quantile function and random generation for the Gumbel distribution with location and scale parameters.

Usage

```
dGumbel(x, loc=0, scale=1, log = FALSE)
pGumbel(q, loc=0, scale=1, lower.tail = TRUE)
qGumbel(p, loc=0, scale=1, lower.tail = TRUE)
rGumbel(n, loc=0, scale=1)
```

Arguments

x, q	Vector of quantiles.
p	Vector of probabilities.
n	Number of observations.
loc, scale	Location and scale parameters (can be given as vectors).
log	Logical; if TRUE, the log density is returned.
lower.tail	Logical; if TRUE (default), probabilities are P[X <= x], otherwise, P[X > x]

Details

The Gumbel distribution function with parameters $loc = a$ and $scale = b$ is

$$G(z) = \exp\left\{-\exp\left[-\left(\frac{z-a}{b}\right)\right]\right\}$$

for all real z, where $b > 0$.

Value

dGumbel gives the density function, pGumbel gives the distribution function, qGumbel gives the quantile function, and rGumbel generates random deviates.

Author(s)

Alec Stephenson <alec_stephenson@hotmail.com>

See Also

rFrechet, rGenExtrVal, rRevWeibull

Examples

```
dGumbel(-1:2, -1, 0.5)
pGumbel(-1:2, -1, 0.5)
qGumbel(seq(0.9, 0.6, -0.1), 2, 0.5)
rGumbel(6, -1, 0.5)
p <- (1:9)/10
pGumbel(qGumbel(p, -1, 2), -1, 2)
## [1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
```

`Herfindahl` *Concentration Measures*

Description

Computes the concentration within a vector according to the specified concentration measure.

Usage

```
Herfindahl(x, n = rep(1, length(x)), parameter = 1, na.rm = FALSE)
Rosenbluth(x, n = rep(1, length(x)), na.rm = FALSE)
```

Arguments

`x`	a vector containing non-negative elements
`n`	a vector of frequencies (weights), must be same length as x.
`parameter`	parameter of the concentration measure (if set to `NULL` the default parameter of the respective measure is used)
`na.rm`	logical. Should missing values be removed? Defaults to FALSE.

Value

the value of the concentration measure

Note

The same measure is usually known as the Simpson index in ecology, and as the Herfindahl index or the Herfindahl-Hirschman index (HHI) in economics.

Note

These functions were previously published as `conc()` in the **ineq** package and have been integrated here without logical changes. `NA` and weights support were added.

Author(s)

Achim Zeileis <achim.zeileis@r-project.org>

References

Cowell, F. A. (2000) Measurement of Inequality, in Atkinson, A. B., Bourguignon, F. *Handbook of Income Distribution*. (Eds) Amsterdam

Cowell, F. A. (1995) *Measuring Inequality*. Prentice Hall/Harvester Wheatshef

Hall, M., Tidemann, N. (1967) *Measures of Concentration*, JASA 62, 162-168.

See Also

See Gini, Atkinson and ineq() for additional inequality measures

Examples

```
# generate vector (of sales)
x <- c(541, 1463, 2445, 3438, 4437, 5401, 6392, 8304, 11904, 22261)

# compute Herfindahl coefficient with parameter 1
Herfindahl(x)

# compute coefficient of Hall/Tiedemann/Rosenbluth
Rosenbluth(x)

# Some more examples
Herfindahl(c(261,29,33,15,39,28,95,5,6,28,69,8,105,38,15))
Herfindahl(c(783,121,112,70,201,153,425,19,37,126,325,51,442,193,41))
```

HexToCol *Identify Closest Match to a Color Given by a Hexadecimal String*

Description

Given a color as a hex string #rrggbb, find the closest match in the table of known (named) colors.

Usage

```
HexToCol(hexstr, method = "rgb", metric = "euclidean")
```

Arguments

hexstr	a color or a vector of colors specified as hexadecimal string of the form "#RRGGBB" or "#RRGGBBAA"
method	character string specifying the color space to be used. Can be "rgb" (default) or "hsv".
metric	character string specifying the metric to be used for calculating distances between the colors. Available options are "euclidean" (default) and "manhattan". Euclidean distances are root sum-of-squares of differences, and manhattan distances are the sum of absolute differences.

Details

Finds the color with the minimum squared distance in RGB space.

Value

The colorname(s) of the closest match(es) (if more than one).

Author(s)

Ben Bolker, vector support Andri Signorell <andri@signorell.net>

See Also

ColToHex, ColToRgb, colors

Examples

```
ColToHex(c("lightblue", "salmon"))

HexToCol(c("#ADD8E6", "#FA1572"))
HexToCol(Pal("Helsana"))

x <- ColToRgb("darkmagenta")
x[2,] <- x[2,] + 155
RgbToCol(x)
```

HexToRgb — *Convert a Hexstring Color to a Matrix With Three Red/Green/Blue Rows*

Description

`HexToRgb()` converts a hexstring color the its red/green/blue representation.

Usage

```
HexToRgb(hex)
```

Arguments

`hex`	a color or a vector of colors specified as hexadecimal string of the form "#RRGGBB" or "#RRGGBBAA"

Details

A hex color is written as a hash character, "#", followed by 3 or 4 hexadecimal numbers, say 6, resp. 8, digits (0-9A-F). The first 3 pairs of digits specify the red, green and blue components. When there are 8 digits, then the last pair is interpreted as alpha channel defining transparency, where `00` represents a fully transparent color and `FF` represent a fully opaque color.
The result will be returned as a matrix having 3 or 4 rows, depending on if the input contained a `RRGGBBAA` definition or not. No distinction is made between upper and lower case. A missing leading # is tolerated.

Value

a matrix with 3 or 4 rows.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

HexToCol

Examples

```
HexToRgb(c("#ADD8E6", "#FA1572"))

# 4-digit representation returns a 4 row matrix
HexToRgb(hex=c("#A52A2ABB","#A52A3B","C52A3B"))
```

Hmean *Harmonic Mean and Its Confidence Interval*

Description

Calculates the harmonic mean and its confidence interval of a vector x.

Usage

```
Hmean(x, method = c("classic", "boot"), conf.level = NA,
      sides = c("two.sided","left","right"), na.rm = FALSE, ...)
```

Arguments

x	a positive numeric vector. An object which is not a vector is coerced (if possible) by as.vector.
method	a vector of character strings representing the type of intervals required. The value should be any subset of the values "classic", "boot". See boot.ci.
conf.level	confidence level of the interval. Default is NA.
sides	a character string specifying the side of the confidence interval, must be one of "two.sided" (default), "left" or "right". You can specify just the initial letter. "left" would be analogue to a hypothesis of "greater" in a t.test.
na.rm	logical, indicating whether NA values should be stripped before the computation proceeds. Defaults to FALSE.
...	further arguments are passed to the boot function. Supported arguments are type ("norm", "basic", "stud", "perc", "bca"), parallel and the number of bootstrap replicates R. If not defined those will be set to their defaults, being "basic" for type, option "boot.parallel" (and if that is not set, "no") for parallel and 999 for R.

Details

To compute the harmonic mean, `1/x` is first calculated, before the arithmetic mean and its confidence interval are computed by `MeanCI`. The harmonic mean is then the reciprocal of the arithmetic mean of the reciprocals of the values. The same applies to the confidence interval.

The harmonic mean is restricted to strictly positive inputs, if any argument is negative, then the result will be `NA`. If the lower bound of the confidence interval is not greater than zero, then the confidence interval is not defined, and thus `NA` will be reported.

Use `sapply` to calculate the measures from data frame, resp. from a matrix.

Value

a numeric value.

Author(s)

Andri Signorell <andri@signorell.net>

References

Snedecor, G. W., Cochran, W. G. (1989) Statistical Methods, 8th ed. Ames, *IA: Iowa State University Press*

See Also

`Gmean`

Examples

```
x <- runif(5)
Hmean(x)

m <- matrix(runif(50), nrow = 10)
apply(m, 2, Hmean)

sapply(as.data.frame(m), Hmean)
```

`HmsToSec` *Convert h:m:s To/From Seconds*

Description

`HmsToSec` - Converts a vector of h:m:s to seconds.

`SecToHms` - Converts a vector of seconds to h:m:s.

Usage

```
HmsToSec(x)
SecToHms(x, digits = NULL)
```

Arguments

x	A vector of times in h:m:s (for HmsToSec) or seconds (for SecToHms).
digits	the number of digits to use for potential fractions of seconds.

Value

HmsToSec - Returns a vector of times in seconds.

SecToHms - Returns a vector of times in h:m:s format.

Author(s)

Tyler Rinker <tyler.rinker@gmail.com>

See Also

times

Examples

```
HmsToSec(c("02:00:03", "04:03:01"))
HmsToSec(SecToHms(c(222, 1234, 55)))
SecToHms(c(256, 3456, 56565))
```

HodgesLehmann *Hodges-Lehmann Estimator of Location*

Description

Function to compute the Hodges-Lehmann estimator of location in the one and two sample case following a clever fast algorithm by John Monahan (1984).

Usage

```
HodgesLehmann(x, y = NULL, conf.level = NA, na.rm = FALSE)
```

Arguments

x	a numeric vector.
y	an optional numeric vector of data values: as with x non-finite values will be omitted.
conf.level	confidence level of the interval.
na.rm	logical. Should missing values be removed? Defaults to FALSE.

Details

The Hodges-Lehmann estimator is the median of the combined data points and Walsh averages. It is the same as the Pseudo Median returned as a by-product of the function `wilcox.test` (which however does not calculate correctly as soon as ties are present).
Note that in the two-sample case the estimator for the difference in location parameters does not estimate the difference in medians (a common misconception) but rather the median of the difference between a sample from x and a sample from y.

(The calculation of the confidence intervals is not yet implemented.)

Value

the Hodges-Lehmann estimator of location as a single numeric value if no confidence intervals are requested,
and otherwise a numeric vector with 3 elements for the estimate, the lower and the upper confidence interval

Author(s)

Cyril Flurin Moser (Cyril did the lion's share and coded Monahan's algorithm in C++), Andri Signorell <andri@signorell.net>

References

Hodges, J.L., and Lehmann, E.L. (1963), Estimates of location based on rank tests. *The Annals of Mathematical Statistics*, **34**, 598–611.

Monahan, J. (1984), Algorithm 616: Fast Computation of the Hodges-Lehmann Location Estimator, *ACM Transactions on Mathematical Software*, Vol. 10, No. 3, pp. 265-270

See Also

`wilcox.test`, `median`, `MedianCI`

Examples

```
set.seed(1)
x <- rt(100, df = 3)
y <- rt(100, df = 5)

HodgesLehmann(x)
HodgesLehmann(x, y)

# same as
wilcox.test(x, conf.int = TRUE)$estimate
```

HoeffD *Matrix of Hoeffding's D Statistics*

Description

Computes a matrix of Hoeffding's (1948) D statistics for all possible pairs of columns of a matrix. D is a measure of the distance between F(x,y) and G(x)H(y), where F(x,y) is the joint CDF of X and Y, and G and H are marginal CDFs. Missing values are deleted in pairs rather than deleting all rows of x having any missing variables. The D statistic is robust against a wide variety of alternatives to independence, such as non-monotonic relationships. The larger the value of D, the more dependent are X and Y (for many types of dependencies). D used here is 30 times Hoeffding's original D, and ranges from -0.5 to 1.0 if there are no ties in the data. print.HoeffD prints the information derived by HoeffD. The higher the value of D, the more dependent are x and y.

Usage

```
HoeffD(x, y)
## S3 method for class 'HoeffD'
print(x, ...)
```

Arguments

x	a numeric matrix with at least 5 rows and at least 2 columns (if y is absent), or an object created by HoeffD
y	a numeric vector or matrix which will be concatenated to x
...	ignored

Details

Uses midranks in case of ties, as described by Hollander and Wolfe. P-values are approximated by linear interpolation on the table in Hollander and Wolfe, which uses the asymptotically equivalent Blum-Kiefer-Rosenblatt statistic. For P<.0001 or >0.5, P values are computed using a well-fitting linear regression function in log P vs. the test statistic. Ranks (but not bivariate ranks) are computed using efficient algorithms (see reference 3).

Value

a list with elements D, the matrix of D statistics, n the matrix of number of observations used in analyzing each pair of variables, and P, the asymptotic P-values. Pairs with fewer than 5 non-missing values have the D statistic set to NA. The diagonals of n are the number of non-NAs for the single variable corresponding to that row and column.

Author(s)

Frank Harrell <f.harrell@vanderbilt.edu>
Department of Biostatistics
Vanderbilt University

References

Hoeffding W. (1948) A non-parametric test of independence. *Ann Math Stat* 19:546–57.

Hollander M., Wolfe D.A. (1973) *Nonparametric Statistical Methods*, pp. 228–235, 423. New York: Wiley.

Press W.H., Flannery B.P., Teukolsky S.A., Vetterling, W.T. (1988) *Numerical Recipes in C* Cambridge: Cambridge University Press.

See Also

rcorr, varclus

Examples

```
x <- c(-2, -1, 0, 1, 2)
y <- c(4,   1, 0, 1, 4)
z <- c(1,   2, 3, 4, NA)
q <- c(1,   2, 3, 4, 5)

HoeffD(cbind(x, y, z, q))


# Hoeffding's test can detect even one-to-many dependency
set.seed(1)
x <- seq(-10, 10, length=200)
y <- x * sign(runif(200, -1, 1))
plot(x, y)

HoeffD(x, y)
```

HosmerLemeshowTest *Hosmer-Lemeshow Goodness of Fit Tests*

Description

The function computes Hosmer-Lemeshow goodness of fit tests for C and H statistic as well as the le Cessie-van Houwelingen-Copas-Hosmer unweighted sum of squares test for global goodness of fit.

Usage

```
HosmerLemeshowTest(fit, obs, ngr = 10, X, verbose = FALSE)
```

Arguments

fit	numeric vector with fitted probabilities.
obs	numeric vector with observed values.
ngr	number of groups for C and H statistic.

X	covariate(s) for le Cessie-van Houwelingen-Copas-Hosmer global goodness of fit test.
verbose	logical, print intermediate results.

Details

Hosmer-Lemeshow goodness of fit tests are computed; see Lemeshow and Hosmer (1982).

If X is specified, the le Cessie-van Houwelingen-Copas-Hosmer unweighted sum of squares test for global goodness of fit is additionally determined; see Hosmer et al. (1997).

Value

A list of tests.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References

Lemeshow, S. Hosmer, D.W., (1982): A review of goodness of fit statistics for use in the development of logistic regression models. *American Journal of Epidemiology,* ***115****(1), 92-106.*

Hosmer, D.W., Hosmer, T., le Cessie, S., Lemeshow, S. (1997). A comparison of goodness-of-fit tests for the logistic regression model. *Statistics in Medicine*, **16**, 965-980.

See Also

glm

Examples

```
set.seed(111)

x1 <- factor(sample(1:3, 50, replace = TRUE))
x2 <- rnorm(50)
obs <- sample(c(0,1), 50, replace = TRUE)

fit <- glm(obs ~ x1+x2, family = binomial)

HosmerLemeshowTest(fit = fitted(fit), obs = obs, X = cbind(x1, x2))
```

`HotellingsT2Test` *Hotelling's T2 Test*

Description

Hotelling's T2 test is the multivariate generlisation of the Student's t test. A one-sample Hotelling's T2 test can be used to test if a set of vectors of data (which should be a sample of a single statistical population) has a mean equal to a hypothetical mean. A two-sample Hotelling's T2 test may be used to test for significant differences between the mean vectors (multivariate means) of two multivariate data sets are different.

Usage

```
HotellingsT2Test(x, ...)

## Default S3 method:
HotellingsT2Test(x, y = NULL, mu = NULL, test = "f", ...)

## S3 method for class 'formula'
HotellingsT2Test(formula, data, subset, na.action, ...)
```

Arguments

`x`	a numeric data frame or matrix.
`y`	an optional numeric data frame or matrix for the two sample test. If `NULL` a one sample test is performed.
`mu`	a vector indicating the hypothesized value of the mean (or difference in means if a two sample test is performed). `NULL` represents origin or no difference between the groups.
`test`	if `"f"`, the decision is based on the F-distribution, if `"chi"` a chi-squared approximation is used.
`formula`	a formula of the form `x ~ g` where `x` is a numeric matrix giving the data values and `g` a factor with two levels giving the corresponding groups.
`data`	an optional matrix or data frame (or similar: see `model.frame`) containing the variables in the formula `formula`. By default the variables are taken from `environment(formula)`.
`subset`	an optional vector specifying a subset of observations to be used.
`na.action`	a function which indicates what should happen when the data contain NAs. Defaults to `getOption("na.action")`.
`...`	further arguments to be passed to or from methods.

Details

The classical test for testing the location of a multivariate population or for testing the mean difference for two multivariate populations. When `test = "f"` the F-distribution is used for the test statistic and it is assumed that the data are normally distributed. If the chisquare approximation is used, the normal assumption can be relaxed to existence of second moments. In the two sample case both populations are assumed to have the same covariance matrix.

The formula interface is only applicable for the 2-sample tests.

Value

A list with class 'htest' containing the following components:

`statistic`	the value of the T2-statistic. (That is the scaled value of the statistic that has an F distribution or a chisquare distribution depending on the value of `test`).
`parameter`	the degrees of freedom for the T2-statistic.
`p.value`	the p-value for the test.
`null.value`	the specified hypothesized value of the mean or mean difference depending on whether it was a one-sample test or a two-sample test.
`alternative`	a character string with the value 'two.sided'.
`method`	a character string indicating what type of test was performed.
`data.name`	a character string giving the name of the data (and grouping vector).

Author(s)

Klaus Nordhausen, <klaus.nordhausen@uta.fi>

References

Nordhausen K., Sirkia S., Oja H. and Tyler D. E. (2012) *ICSNP: Tools for Multivariate Nonparametrics*. R package version 1.0-9.
https://cran.r-project.org/package=ICSNP

Anderson, T.W. (2003), *An introduction to multivariate analysis*, New Jersey: Wiley.

Examples

```
math.teach <- data.frame(
  teacher = factor(rep(1:2, c(3, 6))),
  satis   = c(1, 3, 2, 4, 6, 6, 5, 5, 4),
  know    = c(3, 7, 2, 6, 8, 8, 10, 10, 6))

with(math.teach,
  HotellingsT2Test(cbind(satis, know) ~ teacher))
```

HuberM *Safe (generalized) Huber M-Estimator of Location*

Description

(Generalized) Huber M-estimator of location with MAD scale, being sensible also when the scale is zero where `huber()` returns an error.

Usage

```
HuberM(x, k = 1.345, mu = median(x), s = mad(x, center = mu),
       na.rm = FALSE, conf.level = NA, ci.type = c("wald", "boot"), ...)
```

Arguments

`x`	numeric vector.
`k`	positive factor; the algorithm winsorizes at `k` standard deviations.
`mu`	initial location estimator.
`s`	scale estimator held constant through the iterations.
`na.rm`	logical, indicating whether `NA` values should be stripped before the computation proceeds. Defaults to FALSE.
`conf.level`	confidence level of the interval. If set to `NA` (which is the default) no confidence interval will be calculated.
`ci.type`	The type of confidence interval required. The value should be any subset of the values `"wald"`, `"boot"`.
`...`	the dots are passed to the function `boot.ci`, when confidence intervalls are calculated.

Details

The standard error is computed using the τ correction factor but no finite sample correction.
The original function is not exported, but can be accessed as `DescTools:::.huberM`.

Value

If `conf.level` is set to `NA` then the result will be

`a`	single numeric value

and if a `conf.level` is provided, a named numeric vector with 3 elements:

`huberm`	the estimate for location
`lwr.ci`	lower bound of the confidence interval
`upr.ci`	upper bound of the confidence interval

Author(s)

Martin Maechler, building on the MASS code mentioned.
Andri Signorell <andri@signorell.net> (confidence intervals and interface)

References

Huber, P. J. (1981) *Robust Statistics.* Wiley.

See Also

hubers (and huber) in package **MASS**; mad.

Examples

```
HuberM(c(1:9, 1000))
mad   (c(1:9, 1000))

set.seed(7)
x <- c(round(rnorm(1000), 1), round(rnorm(50, m=10, sd = 10)))
HuberM(x, conf.level=0.95)


## Not run:

# scale zero
HuberM(rep(9, 100))
mad   (rep(9, 100))

# bootstrap confidence intervals
HuberM(x, conf.level=0.95, ci.type="boot")

## End(Not run)
```

ICC *Intraclass Correlations (ICC1, ICC2, ICC3 From Shrout and Fleiss)*

Description

The Intraclass correlation is used as a measure of association when studying the reliability of raters. Shrout and Fleiss (1979) outline 6 different estimates, that depend upon the particular experimental design. All are implemented and given confidence limits.

Usage

```
ICC(x, type = c("all", "ICC1", "ICC2", "ICC3", "ICC1k", "ICC2k", "ICC3k"),
    conf.level = NA, na.rm = FALSE)

## S3 method for class 'ICC'
```

```
print(x, digits = 3, ...)
```

Arguments

x	$n \times m$ matrix or dataframe, k subjects (in rows) m raters (in columns).
type	one out of "all", "ICC1", "ICC2", "ICC3", "ICC1k", "ICC2k", "ICC3k". See details.
conf.level	confidence level of the interval. If set to NA (which is the default) no confidence intervals will be calculated.
na.rm	logical, indicating whether NA values should be stripped before the computation proceeds. If set to TRUE only the complete cases of the ratings will be used. Defaults to FALSE.
digits	number of digits to use in printing
...	further arguments to be passed to or from methods.

Details

Shrout and Fleiss (1979) consider six cases of reliability of ratings done by k raters on n targets.

ICC1	Each target is rated by a different judge and the judges are selected at random. (This is a one-way ANOVA fixed effects model and is found by (MSB- MSW)/(MSB+ (nr-1)*MSW))
ICC2	A random sample of k judges rate each target. The measure is one of absolute agreement in the ratings. Found as (MSB- MSE)/(MSB + (nr-1)*MSE + nr*(MSJ-MSE)/nc)
ICC3	A fixed set of k judges rate each target. There is no generalization to a larger population of judges. (MSB - MSE)/(MSB+ (nr-1)*MSE)

Then, for each of these cases, is reliability to be estimated for a single rating or for the average of k ratings? (The 1 rating case is equivalent to the average intercorrelation, the k rating case to the Spearman Brown adjusted reliability.)

ICC1 is sensitive to differences in means between raters and is a measure of absolute agreement.

ICC2 and ICC3 remove mean differences between judges, but are sensitive to interactions of raters by judges.
The difference between ICC2 and ICC3 is whether raters are seen as fixed or random effects.

ICC1k, ICC2k, ICC3K reflect the means of k raters.

The intraclass correlation is used if raters are all of the same “class". That is, there is no logical way of distinguishing them. Examples include correlations between pairs of twins, correlations between raters. If the variables are logically distinguishable (e.g., different items on a test), then the more typical coefficient is based upon the inter-class correlation (e.g., a Pearson r) and a statistic such as alpha or omega might be used.

Value

if method is set to "all", then the result will be

results	A matrix of 6 rows and 8 columns, including the ICCs, F test, p values, and confidence limits
summary	The anova summary table
stats	The anova statistics
MSW	Mean Square Within based upon the anova

if a specific type has been defined, the function will first check, whether no confidence intervals are requested: if so, the result will be the estimate as numeric value

else a named numeric vector with 3 elements

ICCx	estimate (name is the selected type of coefficient)
lwr.ci	lower confidence interval
upr.ci	upper confidence interval

Note

The results for the lower and upper Bounds for ICC(2,k) do not match those of SPSS 9 or 10, but do match the definitions of Shrout and Fleiss. SPSS seems to have been using the formula in McGraw and Wong, but not the errata on p 390. They seem to have fixed it in more recent releases (15).

Author(s)

William Revelle <revelle@northwestern.edu>, some editorial amendments Andri Signorell <andri@signorell.net>

References

Shrout, P. E., Fleiss, J. L. (1979) Intraclass correlations: uses in assessing rater reliability. *Psychological Bulletin*, 86, 420-3428.

McGraw, K. O., Wong, S. P. (1996) Forming inferences about some intraclass correlation coefficients. *Psychological Methods*, 1, 30-46. + errata on page 390.

Revelle, W. (in prep) *An introduction to psychometric theory with applications in R* Springer. (working draft available at http://personality-project.org/r/book/

Examples

```
sf <- matrix(c(
      9, 2, 5, 8,
      6, 1, 3, 2,
      8, 4, 6, 8,
      7, 1, 2, 6,
      10,5, 6, 9,
      6, 2, 4, 7),
      ncol=4, byrow=TRUE,
      dimnames=list(paste("S", 1:6, sep=""), paste("J", 1:4, sep="")))
)

sf  #example from Shrout and Fleiss (1979)
ICC(sf)
```

identify.formula *Identify Points In a Plot Using a Formula*

Description

The function identify reads the position of the graphics pointer when the (first) mouse button is pressed. It then searches the coordinates given in x and y for the point closest to the pointer. If this point is close enough to the pointer, its index will be returned as part of the value of the call.

Usage

```
## S3 method for class 'formula'
identify(formula, data, subset, na.action, ...)
```

Arguments

formula	a formula of the form lhs ~ rhs where lhs gives the data values and rhs the corresponding groups.
data	The data frame from which the formula should be evaluated.
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
...	Other arguments to be passed to identify.

Details

This function is meant to make it easier to call identify after plot has been called using a formula and the data argument.

A two dimensional plot must be active and the vectors in x and data frame in data must correspond to the x- and y-axes and the data of the plot.

Value

If pos is FALSE, an integer vector containing the indices of the identified points, in the order they were identified. If pos is TRUE, a list containing a component ind, indicating which points were identified and a component pos, indicating where the labels were placed relative to the identified points (1=below, 2=left, 3=above, 4=right and 0=no offset, used if atpen = TRUE).

Author(s)

Derek Ogle <dogle@northland.edu>

See Also

identify, locator, text
https://www.rforge.net/NCStats/files/

Examples

```
## Not run:
## Copy and try in an interactive R session
plot(dist ~ speed, data = cars, subset = speed < 17)
identify(dist ~ speed, data = cars, subset = speed < 17)

## End(Not run)
```

IdentifyA *Identify Points in Plot Lying Within a Rectangle or Polygon*

Description

Find all the points lying either in a rectangle area spanned by an upper left and a bottom-right point or by a polygon area consisting of any number of points defined by point and click.

Usage

```
IdentifyA(x, ...)

## S3 method for class 'formula'
 IdentifyA(formula, data, subset, poly = FALSE, ...)
## Default S3 method:
 IdentifyA(x, y = NULL, poly = FALSE, ...)
```

Arguments

x, y	x and y values of the points used to create the plot.
formula	a formula, such as y ~ x specifying x and y values. Here the formula must be entered that was used to create the scatterplot.
data	a data frame (or list) from which the variables in formula should be taken.
subset	an optional vector specifying a subset of observations to be used.
poly	logical. Defines if a polygon or a rectangle should be used to select the points. Default is rectangle. If a polygon should be used, set this argument to TRUE and select all desired points. The polygon will be closed automatically when finished.
...	Other arguments to be passed to IdentifyA.

Value

Index vector with the points lying within the selected area. The coordinates are returned as text in the attribute "cond".

Author(s)

Andri Signorell <andri@signorell.net>

See Also

identify, locator

Examples

```
## Not run:
# run the example via copy and paste

plot(temperature ~ delivery_min, data=d.pizza)
idx <- IdentifyA(temperature ~ delivery_min, data=d.pizza)

# you selected the following points
d.pizza[idx,]
points(temperature ~ delivery_min, data = d.pizza[idx,], col="green")

# use the attr("cond") for subsets in code
attr(idx, "cond")

# create a group variable for the found points
d.pizza$grp <- seq(nrow(d.pizza)) %in% idx

# try the polygon option
idx <- IdentifyA(temperature ~ delivery_min, data=d.pizza, poly=TRUE)
points(temperature ~ delivery_min, data = d.pizza[idx,], col="red")

## End(Not run)
```

ImputeKnn *Fill in NA values with the values of the nearest neighbours*

Description

Function that fills in all NA values using the k Nearest Neighbours of each case with NA values. By default it uses the values of the neighbours and obtains an weighted (by the distance to the case) average of their values to fill in the unknows. If meth='median' it uses the median/most frequent value, instead.

Usage

```
ImputeKnn(data, k = 10, scale = TRUE, meth = "weighAvg", distData = NULL)
```

Arguments

data	A data frame with the data set
k	The number of nearest neighbours to use (defaults to 10)
scale	Boolean setting if the data should be scale before finding the nearest neighbours (defaults to TRUE)

meth	String indicating the method used to calculate the value to fill in each NA. Available values are 'median' or 'weighAvg' (the default).
distData	Optionally you may sepecify here a data frame containing the data set that should be used to find the neighbours. This is usefull when filling in NA values on a test set, where you should use only information from the training set. This defaults to NULL, which means that the neighbours will be searched in data

Details

This function uses the k-nearest neighbours to fill in the unknown (NA) values in a data set. For each case with any NA value it will search for its k most similar cases and use the values of these cases to fill in the unknowns.

If meth='median' the function will use either the median (in case of numeric variables) or the most frequent value (in case of factors), of the neighbours to fill in the NAs. If meth='weighAvg' the function will use a weighted average of the values of the neighbours. The weights are given by exp(-dist(k,x) where dist(k,x) is the euclidean distance between the case with NAs (x) and the neighbour k.

Value

A data frame without NA values

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2010) *Data Mining using R: learning with case studies*, CRC Press (ISBN: 9781439810187).

See Also

complete.cases, na.omit

Examples

```
cleanPizza <- ImputeKnn(d.pizza[, -2])   # no dates allowed
summary(cleanPizza)
```

InDots *Is a Specific Argument in the Dots-Arguments?*

Description

Returns the value of a specific named argument if it was comprised in the dots or a default value, if it wasn't.

Usage

```
InDots(..., arg, default)
```

Arguments

...	the dots arguments to be checked.
arg	the name of argument to test for.
default	the default value to return, if the argument arg does not exist in the dots.

Value

the value of the argument, if it exists else the specified default value.

Author(s)

Andri Signorell <andri@signorell.net>

Examples

```
# Function returns the argument A, if supplied or 999
foobar <-  function(...){
  DescTools::InDots(..., arg="A", default=99)
}

foobar(A=5)
foobar(B=5, C=8)
```

IQRw *The (weighted) Interquartile Range*

Description

computes interquartile range of the x values. Weights are supported.

Usage

```
IQRw(x, weights = NULL, na.rm = FALSE, type = 7)
```

Arguments

x	a numeric vector.
weights	an optional numeric vector giving the sample weights.
na.rm	logical. Should missing values be removed?
type	an integer selecting one of the many quantile algorithms, see Quantile().

Details

This implementation is based on `Quantile()` function, which allows to define weights.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`Median()`, `Quantile()`, `IQR()`, `quantile()`

Examples

```
x <- c(3.7,3.3,3.5,2.8)
w <- c(5, 5, 4, 1)/15

IQRw(x=x, weights=w)
```

IsDate *Check If an Object Is of Type Date*

Description

Check if the given x is of any known Date type.

Usage

```
IsDate(x, what = c("either", "both", "timeVaries"))
```

Arguments

`x`	a vector or values to be checked.
`what`	can be any value out of "`either`" (default), "`both`" or "`timeVaries`".

Details

This checks for many known Date and Time classes: "POSIXt", "POSIXct", "dates", "times", "chron", "Date".

Value

logical vector of the same dimension as x.

Author(s)

Frank E Harrell

See Also

Year, Month, etc.

Examples

```
IsDate(as.Date("2013-04-10"))

IsDate(31002)
```

IsDichotomous *Test If a Variable Contains Only Two Unique Values*

Description

Test if a variable contains only two values. The variable does not need to be a numerical value, factors and logicals are supported as well. NAs can be skipped by setting na.rm to TRUE.

Usage

```
IsDichotomous(x, strict = FALSE, na.rm = FALSE)

Flags(x, na.rm = FALSE)
```

Arguments

x	a numeric or integer vector, a logical vector or a factor (ordered and unordered)
strict	logical. If set to TRUE, the result will only be TRUE, if x contains exactly 2 levels. If set to FALSE the result will be TRUE for 1 and for 2 levels.
na.rm	logical, indicating whether NA values should be stripped before the computation proceeds. Defaults to FALSE.

Details

IsDichotomous tests a single variable. Flags returns the names of all the dichotomous variables in a list or data.frame.

Value

TRUE if x contains only two unique values, FALSE else

Author(s)

Andri Signorell <andri@signorell.net>

Examples

```
IsDichotomous(sample(10, 5, replace=TRUE))
```

IsEuclid *Is a Distance Matrix Euclidean?*

Description

Confirmation of the Euclidean nature of a distance matrix by the Gower's theorem.
IsEuclid is used in summary.dist.

Usage

```
IsEuclid(distmat, plot = FALSE, print = FALSE, tol = 1e-07)
```

Arguments

distmat	an object of class 'dist'
plot	a logical value indicating whether the eigenvalues bar plot of the matrix of the term $-\frac{1}{2}d_{ij}^2$ centred by rows and columns should be diplayed
print	a logical value indicating whether the eigenvalues of the matrix of the term $-\frac{1}{2}d_{ij}^2$ centred by rows and columns should be printed
tol	a tolerance threshold : an eigenvalue is considered positive if it is larger than -tol*lambda1 where lambda1 is the largest eigenvalue.

Value

returns a logical value indicating if all the eigenvalues are positive or equal to zero

Author(s)

Daniel Chessel
Stephane Dray <dray@biomserv.univ-lyon1.fr>

References

Gower, J.C. and Legendre, P. (1986) Metric and Euclidean properties of dissimilarity coefficients. *Journal of Classification*, **3**, 5–48.

Examples

```
w <- matrix(runif(10000), 100, 100)
w <- dist(w)
summary(w)
IsEuclid (w) # TRUE
```

IsOdd *Checks If An Integer Is Even Or Odd*

Description

Checks if the elements of an integer vector x are even or odd.

Usage

```
IsOdd(x)
```

Arguments

x	vector of integers

Value

a logic vector

Author(s)

Andri Signorell <andri@signorell.net>

See Also

IsWhole

Examples

```
IsOdd(1:10)
```

IsPrime *IsPrime Property*

Description

Returns for a vector or matrix of positive integers a logical object of the same dimension(s) containing TRUE for the elements that are prime and FALSE otherwise.

Usage

```
IsPrime(x)
```

Arguments

x	vector or matrix of nonnegative integers

Details

Given a vector or a matrix of positive integers returns a vector of the same size of FALSE and TRUE. Use which(IsPrime(1:21)) to get the positions.

Value

logical vector

Author(s)

Hans W. Borchers <hwborchers@googlemail.com>

See Also

Factorize, Primes

Examples

```
x <- matrix(1:10, nrow=10, ncol=10, byrow=TRUE)
x * IsPrime(x)

# Find first prime number octett:
octett <- c(0, 2, 6, 8, 30, 32, 36, 38) - 19
while (TRUE) {
    octett <- octett + 210
    if (all(IsPrime(octett))) {
        cat(octett, "\n", sep="  ")
        break
    }
}
```

IsValidHwnd *Check Windows Pointer*

Description

Check if a pointer points to a valid and running MS-Office instance. The function does this by first checking for NULL and nil pointer and then trying to get the current selection of the application.

Usage

```
IsValidHwnd(hwnd)
```

Arguments

hwnd	the pointer to a word instance as created by GetNewWrd() or GetCurrWrd(). Default is the last created pointer stored in DescToolsOptions("lastWord").

Value

logical value, TRUE if hwnd is a valid pointer to a running application

Author(s)

Andri Signorell <andri@signorell.net>

See Also

GetCurrWrd(), GetCurrXL(), GetCurrPP()

JarqueBeraTest *(Robust) Jarque Bera Test*

Description

This function performs the Jarque-Bera tests of normality either the robust or the classical way.

Usage

```
JarqueBeraTest(x, robust = TRUE, method = c("chisq", "mc"),
               N = 0, na.rm = FALSE)
```

Arguments

x	a numeric vector of data values.
robust	defines, whether the robust version should be used. Default is TRUE.
method	a character string out of chisq or mc, specifying how the critical values should be obtained. Default is approximated by the chisq-distribution or empirically via Monte Carlo.
N	number of Monte Carlo simulations for the empirical critical values
na.rm	defines if NAs should be omitted. Default is FALSE.

Details

The test is based on a joint statistic using skewness and kurtosis coefficients. The robust Jarque-Bera (RJB) version of utilizes the robust standard deviation (namely the mean absolute deviation from the median, as provided e. g. by MeanAD(x, FUN=median)) to estimate sample kurtosis and skewness. For more details see Gel and Gastwirth (2006).
Setting robust to FALSE will perform the original Jarque-Bera test (see Jarque, C. and Bera, A (1980)).

Value

A list with class `htest` containing the following components:

`statistic`	the value of the test statistic.
`parameter`	the degrees of freedom.
`p.value`	the p-value of the test.
`method`	type of test was performed.
`data.name`	a character string giving the name of the data.

Note

This function is melted from the `jarque.bera.test` (in `tseries` package) and the `rjb.test` from the package `lawstat`.

Author(s)

W. Wallace Hui, Yulia R. Gel, Joseph L. Gastwirth, Weiwen Miao

References

Gastwirth, J. L.(1982) *Statistical Properties of A Measure of Tax Assessment Uniformity*, Journal of Statistical Planning and Inference 6, 1-12.

Gel, Y. R. and Gastwirth, J. L. (2008) *A robust modification of the Jarque-Bera test of normality*, Economics Letters 99, 30-32.

Jarque, C. and Bera, A. (1980) *Efficient tests for normality, homoscedasticity and serial independence of regression residuals*, Economics Letters 6, 255-259.

See Also

Alternative tests for normality as `shapiro.test`, `AndersonDarlingTest`, `CramerVonMisesTest`, `LillieTest`, `PearsonTest`, `ShapiroFranciaTest`

`qqnorm`, `qqline` for producing a normal quantile-quantile plot

Examples

```
x <- rnorm(100)    # null hypothesis
JarqueBeraTest(x)

x <- runif(100)    # alternative hypothesis
JarqueBeraTest(x, robust=TRUE)
```

JonckheereTerpstraTest *Exact Version of Jonckheere-Terpstra Test*

Description

Jonckheere-Terpstra test to test for ordered differences among classes.

Usage

```
JonckheereTerpstraTest(x, ...)

## Default S3 method:
JonckheereTerpstraTest(x, g, alternative = c("two.sided", "increasing", "decreasing"),
                       nperm = NULL, exact = NULL, ...)

## S3 method for class 'formula'
JonckheereTerpstraTest(formula, data, subset, na.action, ...)
```

Arguments

x	a numeric vector of data values, or a list of numeric data vectors.
g	a vector or factor object giving the group for the corresponding elements of x. Ignored if x is a list.
alternative	means are monotonic (two.sided), increasing, or decreasing
nperm	number of permutations for the reference distribution. The default is NULL in which case the permutation p-value is not computed. It's recommended to set nperm to 1000 or higher if permutation p-value is desired.
formula	a formula of the form lhs ~ rhs where lhs gives the data values and rhs the corresponding groups.
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
exact	logical, defining if the exact test should be calculated. If left to NULL, the function uses the exact test up to a samplesize of 100 and falls back to normal approximation for larger samples. The exact procedure can not be applied to samples containing ties.
...	further argument to be passed to methods.

Details

JonckheereTerpstraTest is the exact (permutation) version of the Jonckheere-Terpstra test. It uses the statistic

$$\sum_{k<l}\sum_{ij} I(X_{ik} < X_{jl}) + 0.5I(X_{ik} = X_{jl}),$$

where i, j are observations in groups k and l respectively. The asymptotic version is equivalent to `cor.test(x, g, method="k")`. The exact calculation requires that there be no ties and that the sample size is less than 100. When data are tied and sample size is at most 100 permutation p-value is returned.

If x is a list, its elements are taken as the samples to be compared, and hence have to be numeric data vectors. In this case, g is ignored, and one can simply use JonckheereTerpstraTest(x) to perform the test. If the samples are not yet contained in a list, use JonckheereTerpstraTest(list(x, ...)).

Otherwise, `x` must be a numeric data vector, and `g` must be a vector or factor object of the same length as `x` giving the group for the corresponding elements of `x`.

Note

The function was previously published as `jonckheere.test()` in the **clinfun** package and has been integrated here without logical changes. Some argument checks and a formula interface were added.

Author(s)

Venkatraman E. Seshan <seshanv@mskcc.org>, minor adaptations Andri Signorell

References

Jonckheere, A. R. (1954). A distribution-free k-sample test again ordered alternatives. *Biometrika* 41:133-145.

Terpstra, T. J. (1952). The asymptotic normality and consistency of Kendall's test against trend, when ties are present in one ranking. *Indagationes Mathematicae* 14:327-333.

Examples

```
set.seed(1234)
g <- ordered(rep(1:5, rep(10,5)))
x <- rnorm(50) + 0.3 * as.numeric(g)

JonckheereTerpstraTest(x, g)

x[1:2] <- mean(x[1:2]) # tied data

JonckheereTerpstraTest(x, g)
JonckheereTerpstraTest(x, g, nperm=5000)

# Duller, S. 222
coffee <- list(
  c_4=c(447,396,383,410),
```

```
  c_2=c(438,521,468,391,504,472),
  c_0=c(513,543,506,489,407))

# the list interface:
JonckheereTerpstraTest(coffee)

# the formula interface
breaking <- data.frame(
  speed=c(20,25,25,25,25,30,30,30,35,35),
  distance=c(48,33,59,48,56,60,101,67,85,107))

JonckheereTerpstraTest(distance ~ speed, breaking, alternative="increasing")
```

KappaM *Kappa for m Raters*

Description

Computes kappa as an index of interrater agreement between m raters on categorical data.

Usage

```
KappaM(x, method = c("Fleiss", "Conger", "Light"), conf.level = NA)
```

Arguments

x	$n \times m$ matrix or dataframe, n subjects m raters.
method	a logical indicating whether the exact Kappa (Conger, 1980), the Kappa described by Fleiss (1971) or Light's Kappa (1971) should be computed.
conf.level	confidence level of the interval. If set to NA (which is the default) no confidence intervals will be calculated.

Details

Missing data are omitted in a listwise way.
The coefficient described by Fleiss (1971) does not reduce to Cohen's Kappa (unweighted) for m=2 raters. Therefore, the exact Kappa coefficient, which is slightly higher in most cases, was proposed by Conger (1980).
Light's Kappa equals the average of all possible combinations of bivariate Kappas between raters.
The confidence levels can only be reported using Fleiss' formulation of Kappa.

Value

a single numeric value if no confidence intervals are requested,
and otherwise a numeric vector with 3 elements for the estimate, the lower and the upper confidence interval

Note

This function was previously published as `kappam.fleiss()` in the **irr** package and has been integrated here with some changes in the interface.

Author(s)

Matthias Gamer, with some modifications by Andri Signorell <andri@signorell.net>

References

Conger, A.J. (1980): Integration and generalisation of Kappas for multiple raters. *Psychological Bulletin*, 88, 322-328

Fleiss, J.L. (1971): Measuring nominal scale agreement among many raters *Psychological Bulletin*, 76, 378-382

Fleiss, J.L., Levin, B., & Paik, M.C. (2003): *Statistical Methods for Rates and Proportions*, 3rd Edition. New York: John Wiley & Sons

Light, R.J. (1971): Measures of response agreement for qualitative data: Some generalizations and alternatives. *Psychological Bulletin*, 76, 365-377.

See Also

CohenKappa

Examples

```
statement <- data.frame(
  A=c(2,3,1,3,1,2,1,2,3,3,3,3,3,2,1,3,3,2,2,1,
      2,1,3,3,2,2,1,2,1,1,2,3,3,3,3,3,1,2,1,1),
  B=c(2,2,2,1,1,2,1,2,3,3,2,3,1,3,1,1,3,2,1,2,
      2,1,3,2,2,2,3,2,1,1,2,2,3,3,3,3,2,2,2,3),
  C=c(2,2,2,1,1,2,1,2,3,3,2,3,3,3,3,2,2,2,2,3,
      2,2,3,3,2,2,3,2,2,2,2,3,3,3,3,3,3,2,2,2),
  D=c(2,2,2,1,1,2,1,2,3,3,2,3,3,3,3,3,2,2,2,2,
      3,1,3,2,2,2,1,2,2,1,2,3,3,3,3,3,3,2,2,1),
  E=c(2,2,2,3,3,2,3,1,3,3,2,3,3,3,3,3,2,2,2,3,
      2,3,3,2,2,2,3,2,1,3,2,3,3,1,3,3,3,2,2,1)
)

KappaM(statement)

KappaM(statement, method="Conger")   # Exact Kappa
KappaM(statement, conf.level=0.95)   # Fleiss' Kappa and confidence intervals

KappaM(statement, method="Light")   # Exact Kappa
```

`KendallTauA` *Kendall's τ_a*

Description

Calculate Kendall's tau-a statistic, a measure of association for ordinal factors in a two-way table. The function has interfaces for a table (matrix) and for single vectors.

Usage

```
KendallTauA(x, y = NULL, direction = c("row", "column"), conf.level = NA, ...)
```

Arguments

`x`	a numeric vector or a table. A matrix will be treated as table.
`y`	NULL (default) or a vector with compatible dimensions to `x`. If y is provided, `table(x, y, ...)` is calculated.
`direction`	direction of the calculation. Can be `"row"` (default) or `"column"`, where `"row"` calculates Kendall's tau-a (R|C) ("column dependent").
`conf.level`	confidence level of the interval. If set to `NA` (which is the default) no confidence interval will be calculated.
`...`	further arguments are passed to the function `table`, allowing i.e. to set useNA. This refers only to the vector interface.

Details

Kendall's tau coefficient (sometimes called "Kendall rank correlation coefficient"), is a statistic used to measure the association between two measured quantities. It is a measure of rank correlation: the similarity of the orderings of the data when ranked by each of the quantities.
Kendall's tau-a is computed as

$$\tau_a(C|R) = \frac{P - Q}{\frac{1}{2} \cdot n \cdot (n-1)}$$

where P equals twice the number of concordances and Q twice the number of discordances. Its range is [-1, 1].
(Note that Kendall tau-a does not take into consideration any ties, which makes it unpractical. Consider using `KendallTauB` (Tau-b) when ties are present.)

Value

a single numeric value if no confidence intervals are requested,
and otherwise a numeric vector with 3 elements for the estimate, the lower and the upper confidence interval

Author(s)

Andri Signorell <andri@signorell.net>

References

Agresti, A. (2002) *Categorical Data Analysis*. John Wiley & Sons, pp. 57-59.

Hollander, M, Wolfe, D. A., Chicken, E. (2014) *Nonparametric Statistical Methods*, Third edition, Wiley,

Liebetrau, A. M. (1983) *Measures of Association*, Sage University Papers Series on Quantitative Applications in the Social Sciences, 07-004. Newbury Park, CA: Sage, pp. 49-56

See Also

`ConDisPairs` yields concordant and discordant pairs

Other association measures:
`cor (method="kendall")` for Tau b, `StuartTauC`, `GoodmanKruskalGamma`
`Lambda`, `UncertCoef`, `MutInf`

Examples

```
# example in:
# http://support.sas.com/documentation/cdl/en/statugfreq/63124/PDF/default/statugfreq.pdf
# pp. S. 1821

tab <- as.table(rbind(c(26,26,23,18,9),c(6,7,9,14,23)))

# Kendall's tau-a C|R
KendallTauA(tab, direction="column", conf.level=0.95)
# Kendall's tau-a R|C
KendallTauA(tab, direction="row", conf.level=0.95)

# http://support.sas.com/documentation/cdl/en/statugfreq/63124/PDF/default/statugfreq.pdf
# pp. 1814 (143)
tab <- as.table(cbind(c(11,2),c(4,6)))

KendallTauA(tab, direction="row", conf.level=0.95)
KendallTauA(tab, direction="column", conf.level=0.95)

# Liebetrau, pp. 52
x <- c(1,2,2,3,3,3,4,5)
y <- c(1,3,2,1,5,3,4,5)

ConDisPairs(table(x, y))
KendallTauA(x, y, conf.level=0.95)
```

KendallTauB *Kendall's τ_b*

Description

Calculate Kendall's tau-b. The estimator could also be calculated with `cor(..., method="kendall")`. The calculation of confidence intervals however would not be found there.

Usage

```
KendallTauB(x, y = NULL, conf.level = NA, ...)
```

Arguments

x	a numeric vector, matrix or data.frame.
y	NULL (default) or a vector with compatible dimensions to x. If y is provided, table(x, y, ...) is calculated.
conf.level	confidence level of the interval. If set to NA (which is the default) no confidence interval will be calculated.
...	further arguments are passed to the function table, allowing i.e. to set useNA. This refers only to the vector interface.

Value

a single numeric value if no confidence intervals are requested,
and otherwise a numeric vector with 3 elements for the estimate, the lower and the upper confidence interval

Author(s)

Andri Signorell <andri@signorell.net>

References

Agresti, A. (2002) *Categorical Data Analysis*. John Wiley & Sons, pp. 57-59.

Kendall, M. (1955) *Rank Correlation Methods*, Second Edition, London: Charles Griffin and Co.

Brown, M.B.andBenedetti, J.K.(1977) Sampling Behavior of Tests for Correlation in Two-Way Contingency Tables, *Journal of the American Statistical Association*, 72, 309-315.

See Also

ConDisPairs yields concordant and discordant pairs

Other association measures:
GoodmanKruskalGamma, KendallTauA (tau-a), cor (method="kendall") for tau-b, StuartTauC (tau-c), SomersDelta
Lambda, GoodmanKruskalTau, UncertCoef, MutInf

Examples

```
# example in:
# http://support.sas.com/documentation/cdl/en/statugfreq/63124/PDF/default/statugfreq.pdf
# pp. S. 1821

tab <- as.table(rbind(c(26,26,23,18,9),c(6,7,9,14,23)))

KendallTauB(tab, conf.level=0.95)
```

KendallW *Kendall's Coefficient of Concordance W*

Description

Computes Kendall's coefficient of concordance, a popular measure of association. It is an index of interrater reliability of ordinal data. The coefficient could be corrected for ties within raters.

Usage

```
KendallW(x, correct = FALSE, test = FALSE, na.rm = NULL)
```

Arguments

x	$n \times m$ matrix or dataframe, k subjects (in rows) m raters (in columns).
correct	a logical indicating whether the coefficient should be corrected for ties within raters. (Default FALSE)
test	a logical indicating whether the test statistic and p-value should be reported. (Default FALSE)
na.rm	is not longer supported and will be ignored.

Details

The test for Kendall's W is completely equivalent to friedman.test. The only advantage of this test over Friedman's is that Kendall's W has an interpretation as the coefficient of concordance. The test itself is only valid for large samples.
Kendall's W should be corrected for ties, if raters did not use a true ranking order for the subjects. The function warns if ties are present and no correction has been required.

In the presence of NAs the algorithm is switched to a generalized form for randomly incomplete datasets introduced in Brueckl (2011). This approach uses the mean Spearman ρ of all pairwise comparisons (see Kendall, 1962):

$$W = (1 + mean(\rho) * (k - 1))/k$$

where k is the mean number of (pairwise) ratings per object and mean(ρ) is calculated weighted, according to Taylor (1987), since the pairwise are possibly based on a different number of ratings, what must be reflected in weights. In case of complete datasets, it yields the same results as usual implementations of Kendall's W, except for tied ranks. In case of tied ranks, the (pairwise) correction of s used, which (already with complete datasets) results in slightly different values than the tie correction explicitly specified for W.

Value

Either a single value if `test = FALSE` or else

a list with class "`htest`" containing the following components:

`statistic`	the value of the chi-square statistic.
`p.value`	the p-value for the test.
`method`	the character string "Kendall's coefficient of concordance W".
`data.name`	a character string giving the name(s) of the data.
`estimate`	the coefficient of concordance.
`parameter`	the degrees of freedom df, the number of subjects examined and the number of raters.

Author(s)

Andri Signorell andri@signorell.net
based on code by Matthias Gamer m.gamer@uke.uni-hamburg.de
and Markus Brueckl markus.brueckl@tu-berlin.de

References

Kendall, M.G. (1948) *Rank correlation methods*. London: Griffin.

Kendall, M.G. (1962). Rank correlation methods (3rd ed.). London: Griffin.

Brueckl, M. (2011). Statistische Verfahren zur Ermittlung der Urteileruebereinstimmung. in: Altersbedingte Veraenderungen der Stimme und Sprechweise von Frauen, Berlin: Logos, 88–103.

Taylor, J.M.G. (1987). Kendall's and Spearman's correlation coefficients in the presence of a blocking variable. *Biometrics*, 43, 409–416.

See Also

`cor`, `KappaM`, `CronbachAlpha`, `ICC`, `friedman.test`

Examples

```
anxiety <- data.frame(rater1=c(3,3,3,4,5,5,2,3,5,2,2,6,1,5,2,2,1,2,4,3),
                      rater2=c(3,6,4,6,2,4,2,4,3,3,2,3,3,3,2,2,1,3,3,4),
                      rater3=c(2,1,4,4,3,2,1,6,1,1,1,2,3,3,1,1,3,3,2,2))

KendallW(anxiety, TRUE)

# with test results
KendallW(anxiety, TRUE, test=TRUE)

# example from Siegel and Castellan (1988)
d.att <- data.frame(
  id       = c(4,21,11),
  airfare  = c(5,1,4),
```

```
  climate   = c(6,7,5),
  season    = c(7,6,1),
  people    = c(1,2,3),
  program   = c(2,3,2),
  publicity = c(4,5,7),
  present   = c(3,4,6),
  interest  = c(8,8,8)
)

KendallW(t(d.att[, -1]), test = TRUE)

# which is perfectly the same as
friedman.test(y=as.matrix(d.att[,-1]), groups = d.att$id)
```

`Keywords` *List Keywords For R Manual Pages*

Description

List the keywords for specific R man pages or return a list of valid R keywords.

Usage

```
Keywords(topic)
```

Arguments

`topic`	optional, object or man page topic

Details

If `topic` is provided, return a list of the Keywords associated with `topic`. Otherwise, display the list of valid R Keywords from the R doc/Keywords file.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

`help`

Examples

```
## Show all valid R Keywords
Keywords()

## Show Keywords associated with the 'merge' function
Keywords(merge)
Keywords("merge")
```

`KrippAlpha` *Krippendorff's Alpha Reliability Coefficient*

Description

Calculate the alpha coefficient of reliability proposed by Krippendorff.

Usage

```
KrippAlpha(x, method=c("nominal", "ordinal", "interval", "ratio"))
```

Arguments

`x`	classifier x object matrix of classifications or scores
`method`	data level of x

Value

A list with class '"irrlist"' containing the following components:

`method`	a character string describing the method.
`subjects`	the number of data objects.
`raters`	the number of raters.
`irr.name`	a character string specifying the name of the coefficient.
`value`	value of alpha.
`stat.name`	here "nil" as there is no test statistic.
`statistic`	the value of the test statistic (NULL).
`p.value`	the probability of the test statistic (NULL).
`cm`	the concordance/discordance matrix used in the calculation of alpha
`data.values`	a character vector of the unique data values
`levx`	the unique values of the ratings
`nmatchval`	the count of matches, used in calculation
`data.level`	the data level of the ratings ("nominal","ordinal", "interval","ratio")

Note

Krippendorff's alpha coefficient is particularly useful where the level of measurement of classification data is higher than nominal or ordinal. https://bmcmedresmethodol.biomedcentral.com/articles/10.1186/s12874-016-0200-9

Note

This function was previously published as `kripp.alpha()` in the **irr** package and has been integrated here without logical changes, but with some adaptations in the result structure.

Author(s)

Jim Lemon <jim@bitwrit.com.au>

References

Krippendorff, K. (1980) *Content analysis: An introduction to its methodology*. Beverly Hills, CA: Sage.

See Also

CronbachAlpha, KappaM, CohenKappa

Examples

```
# the "C" data from Krippendorff
 nmm <- matrix(c(1,1,NA,1,2,2,3,2,3,3,3,3,3,3,3,3,2,2,2,2,1,2,3,4,4,4,4,4,
                 1,1,2,1,2,2,2,2,NA,5,5,5,NA,NA,1,1,NA,NA,3,NA), nrow=4)

 # first assume the default nominal classification
 KrippAlpha(nmm)

 # now use the same data with the other three methods
 KrippAlpha(nmm, "ordinal")
 KrippAlpha(nmm, "interval")
 KrippAlpha(nmm, "ratio")
```

Label, Unit *Label, Unit Attribute of an Object*

Description

Set and retrieve the `label`, resp. `unit` attribute of `x`. This can be helpful for documenting the specific meaning of a variable, of an entire data.frame or any other object. For single vectors it can be useful to store the unit.

Usage

```
Label(x)
Label(x) <- value

Labels(x)
Labels(x) <- value

Unit(x)
Unit(x) <- value
```

Arguments

x	any object
value	a single string describing the object

Details

The label should consist of a single text (length of 1). The text may contain line feeds. It can be deleted by setting the label to NULL.

Labels() can be used to retrieve and assign vectorized labels to data.frames or lists.

Value

Label and Unit return the label attribute of x, if any; otherwise, NULL.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

A more elaborated label version can be found in package **Hmisc** label().

Examples

```
# add a descriptive label to a variable
Label(d.diamonds$colour) <- "The rating scale applied to diamonds ranges from colorless
to yellow, as any other color is extremely rare."

# technically just appending the text as attribute to the variable
attributes(d.diamonds$colour)

# label is supported while describing data
Desc(d.diamonds$colour)

# The label can be deleted by setting it to NULL
Label(d.diamonds$colour) <- NULL

# Labelling the columns of a data.frame is best done with a loop
# (all so far seen *apply aproaches lead to more complicated code...)
lbl <-  RndWord(16, 7)
for(i in seq_along(lbl))
  Label(d.pizza[, i]) <- lbl[i]

Str(d.pizza)
```

`Lambda` *Goodman Kruskal Lambda*

Description

Calculate symmetric and asymmetric Goodman Kruskal lambda and their confidence intervals. Lamdba is a measure of proportional reduction in error in cross tabulation analysis. For any sample with a nominal independent variable and dependent variable (or ones that can be treated nominally), it indicates the extent to which the modal categories and frequencies for each value of the independent variable differ from the overall modal category and frequency, i.e. for all values of the independent variable together

Usage

```
Lambda(x, y = NULL, direction = c("symmetric", "row", "column"), conf.level = NA, ...)
```

Arguments

`x`	a numeric vector, a matrix or a table.
`y`	`NULL` (default) or a vector with compatible dimensions to `x`. If `y` is provided, `table(x, y, ...)` is calculated.
`direction`	type of lambda. Can be one out of `"symmetric"` (default), `"row"`, `"column"` (abbreviations are allowed). If direction is set to `"row"` then Lambda(R|C) (column dependent) will be reported. See details.
`conf.level`	confidence level for the returned confidence interval, restricted to lie between 0 and 1.
`...`	further arguments are passed to the function `table`, allowing i.e. to set `useNA = c("no", "ifany", "always")`.

Details

Asymmetric lambda is interpreted as the probable improvement in predicting the column variable Y given knowledge of the row variable X.
The nondirectional lambda is the average of the two asymmetric lambdas, Lambda(C|R) and Lambda(R|C). Lambda (asymmetric and symmetric) has a scale ranging from 0 to 1.

Data can be passed to the function either as matrix or data.frame in `x`, or as two numeric vectors `x` and `y`. In the latter case `table(x, y, ...)` is calculated. Thus `NA`s are handled the same way as `table` does. Note that tables are by default calculated **without** NAs (which breaks the package's law to in general not omit NAs silently). The specific argument `useNA` can be passed via the ... argument.
`PairApply` can be used to calculate pairwise lambdas.

Value

if no confidence intervals are requested: the estimate as numeric value

else a named numeric vector with 3 elements

`lambda`	estimate
`lwr.ci`	lower confidence interval
`upr.ci`	upper confidence interval

Author(s)

Andri Signorell <andri@signorell.net> based on code from Antti Arppe <antti.arppe@helsinki.fi>, Nanina Anderegg (confidence interval symmetric lambda)

References

Agresti, A. (2002) *Categorical Data Analysis*. John Wiley & Sons

Goodman, L. A., Kruskal W. H. (1979) Measures of Association for Cross Classifications. New York: Springer-Verlag (contains articles appearing in *J. Amer. Statist. Assoc.* in 1954, 1959, 1963, 1972).
http://www.nssl.noaa.gov/users/brooks/public_html/feda/papers/goodmankruskal1.pdf (might be outdated)

Liebetrau, A. M. (1983) *Measures of Association*, Sage University Papers Series on Quantitative Applications in the Social Sciences, 07-004. Newbury Park, CA: Sage, pp. 17–24

See Also

`GoodmanKruskalTau`, `GoodmanKruskalGamma`, `KendallTauA`, `KendallTauB`, `StuartTauC`, `SomersDelta`, `cor`

Examples

```
# example from Goodman Kruskal (1954)
m <- as.table(cbind(c(1768,946,115), c(807,1387,438), c(189,746,288), c(47,53,16)))
dimnames(m) <- list(paste("A", 1:3), paste("B", 1:4))
m

# direction default is "symmetric"
Lambda(m)
Lambda(m, conf.level=0.95)

Lambda(m, direction="row")
Lambda(m, direction="column")
```

Lc *Lorenz Curve*

Description

Lc computes the (empirical) ordinary and generalized Lorenz curve of a vector x. Desc calculates some key figures for a Lorenz curve and produces a quick description.

Usage

```
Lc(x, ...)

## Default S3 method:
Lc(x, n = rep(1, length(x)), na.rm = FALSE, ...)

## S3 method for class 'formula'
Lc(formula, data, subset, na.action, ...)

## S3 method for class 'Lc'
plot(x, general = FALSE, lwd = 2, type = "l", xlab = "p", ylab = "L(p)",
     main = "Lorenz curve", las = 1, pch = NA, ...)

## S3 method for class 'Lclist'
plot(x, col = 1, lwd = 2, lty = 1, main = "Lorenz curve",
     xlab = "p", ylab = "L(p)", ...)

## S3 method for class 'Lc'
lines(x, general = FALSE, lwd = 2, conf.level = NA, args.cband = NULL, ...)

## S3 method for class 'Lc'
predict(object, newdata, conf.level=NA, general=FALSE, n=1000, ...)
```

Arguments

x	a vector containing non-negative elements, or a Lc-object for plot and lines.
n	a vector of frequencies, must be same length as x.
na.rm	logical. Should missing values be removed? Defaults to FALSE.
general	logical. If TRUE the empirical Lorenz curve will be plotted.
col	color of the curve
lwd	the linewidth of the curve
lty	the linetype of the curve
type	type of the plot, default is line ("l").
xlab, ylab	label of the x-, resp. y-axis.

pch	the point character (default is NA, meaning no points will be drawn)
main	main title of the plot.
las	las of the axis.
formula	a formula of the form lhs ~ rhs where lhs gives the data values and rhs the corresponding groups.
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
conf.level	confidence level for the bootstrap confidence interval. Set this to NA, if no confidence band should be plotted. Default is NA.
args.cband	list of arguments for the confidence band, such as color or border (see DrawBand).
object	object of class inheriting from "Lc"
newdata	an optional vector of percentages p for which to predict. If omitted, the original values of the object are used.
...	further argument to be passed to methods.

Details

Lc(x) computes the empirical ordinary Lorenz curve of x as well as the generalized Lorenz curve (= ordinary Lorenz curve * mean(x)). The result can be interpreted like this: p*100 percent have L(p)*100 percent of x.

If n is changed to anything but the default x is interpreted as a vector of class means and n as a vector of class frequencies: in this case Lc will compute the minimal Lorenz curve (= no inequality within each group).

Value

A list of class "Lc" with the following components:

p	vector of percentages
L	vector with values of the ordinary Lorenz curve
L.general	vector with values of the generalized Lorenz curve
x	the original x values (needed for computing confidence intervals)
n	the original n values

Note

These functions were previously published as Lc() in the **ineq** package and have been integrated here without logical changes.

Author(s)

Achim Zeileis <Achim.Zeileis@R-project.org>, extensions Andri Signorell <andri@signorell.net>

References

Arnold, B. C. (1987) Majorization and the Lorenz Order: A Brief Introduction, *Springer*

Cowell, F. A. (2000) Measurement of Inequality in Atkinson, A. B. / Bourguignon, F. (Eds): *Handbook of Income Distribution*. Amsterdam.

Cowell, F. A. (1995) Measuring Inequality *Harvester Wheatshef: Prentice Hall*.

See Also

The original location `Lc()`,
inequality measures `Gini()`, `Atkinson()`

Examples

```
priceCarpenter <- d.pizza$price[d.pizza$driver=="Carpenter"]
priceMiller <- d.pizza$price[d.pizza$driver=="Miller"]

# compute the Lorenz curves
Lc.p <- Lc(priceCarpenter, na.rm=TRUE)
Lc.u <- Lc(priceMiller, na.rm=TRUE)
plot(Lc.p)
lines(Lc.u, col=2)

# the picture becomes even clearer with generalized Lorenz curves
plot(Lc.p, general=TRUE)
lines(Lc.u, general=TRUE, col=2)

# inequality measures emphasize these results, e.g. Atkinson's measure
Atkinson(priceCarpenter, na.rm=TRUE)
Atkinson(priceMiller, na.rm=TRUE)


# income distribution of the USA in 1968 (in 10 classes)
# x vector of class means, n vector of class frequencies
x <- c(541, 1463, 2445, 3438, 4437, 5401, 6392, 8304, 11904, 22261)
n <- c(482, 825, 722, 690, 661, 760, 745, 2140, 1911, 1024)

# compute minimal Lorenz curve (= no inequality in each group)
Lc.min <- Lc(x, n=n)
plot(Lc.min)


# input of frequency tables with midpoints of classes
fl <- c(2.5,7.5,15,35,75,150)   # midpoints
n  <- c(25,13,10,5,5,2)         # frequencies

plot(Lc(fl, n),                 # Lorenz-Curve
     panel.first=grid(10, 10),
     main="Lorenzcurve Farmers",
     xlab="Percent farmers (cumulative)",
     ylab="Percent of area (%)"
)
```

```
# add confidence band
lines(Lc(fl, n), conf.level=0.95,
      args.cband=list(col=SetAlpha(DescToolsOptions("col")[2], 0.3)))

Gini(fl, n)

# find specific function values using predict
x <- c(1,1,4)
lx <- Lc(x)
plot(lx)

# get interpolated function value at p=0.55
y0 <- predict(lx, newdata=0.55)
abline(v=0.55, h=y0$L, lty="dotted")

# and for the inverse question use approx
y0 <- approx(x=lx$L, y=lx$p, xout=0.6)
abline(h=0.6, v=y0$y, col="red")

text(x=0.1, y=0.65, label=expression(L^{-1}*(0.6) == 0.8), col="red")
text(x=0.65, y=0.2, label=expression(L(0.55) == 0.275))

# input of frequency tables with midpoints of classes
fl <- c(2.5,7.5,15,35,75,150)     # midpoints
n  <- c(25,13,10,5,5,2)           # frequencies

# the formula interface for Lc
lst <- Lc(count ~ cut(price, breaks=5), data=d.pizza)

plot(lst, col=1:length(lst), panel.first=grid(), lwd=2)
legend(x="topleft", legend=names(lst), fill=1:length(lst))

# Describe with Desc-function
lx <- Lc(fl, n)
Desc(lx)
```

`LehmacherTest` *Lehmacher's Test for Marginal Homogenity*

Description

Performs Lehmacher's chi-squared test for marginal homogenity in a symmetric two-dimensional contingency table.

Usage

```
LehmacherTest(x, y = NULL)

## S3 method for class 'mtest'
```

```
print(x, digits = 4L, ...)
```

Arguments

x	either a two-dimensional contingency table in matrix form, or a factor object.
y	a factor object; ignored if x is a matrix.
digits	a non-null value for digits specifies the minimum number of significant digits to be printed in values. See details in print.default.
...	further arguments to be passed to or from other methods. They are ignored in this function.

Details

The null is that the probabilities of being classified into cells [i,j] and [j,i] are the same.

If x is a matrix, it is taken as a two-dimensional contingency table, and hence its entries should be nonnegative integers. Otherwise, both x and y must be vectors or factors of the same length. Incomplete cases are removed, vectors are coerced into factors, and the contingency table is computed from these.

Value

A list with class "mtest" containing the following components:

statistic	a vector with the value of the test statistics.
parameter	the degrees of freedom, which is always 1 in LehmacherTest.
p.value	a vector with the p-values of the single tests.
p.value.corr	a vector with the "hochberg" adjusted p-values of the single tests. (See p.adjust)
method	a character string indicating what type of test was performed.
data.name	a character string giving the name of the data.

Author(s)

Andri Signorell <andri@signorell.net>

References

Lehmacher, W. (1980) Simultaneous sign tests for marginal homogeneity of square contingency tables *Biometrical Journal*, Volume 22, Issue 8, pages 795-798

See Also

mcnemar.test (resp. BowkerTest for a CxC-matrix), StuartMaxwellTest, WoolfTest

Examples

```
x <- matrix(c(400,40,20,10,
              50,300,60,20,
              10,40,120,5,
              5,90,50,80), nrow=4, byrow=TRUE)

LehmacherTest(x)
```

LeveneTest *Levene's Test for Homogeneity of Variance*

Description

Computes Levene's test for homogeneity of variance across groups.

Usage

```
LeveneTest(y, ...)

## S3 method for class 'formula'
LeveneTest(formula, data, ...)
## S3 method for class 'lm'
LeveneTest(y, ...)
## Default S3 method:
LeveneTest(y, group, center=median, ...)
```

Arguments

y	response variable for the default method, or a lm or formula object. If y is a linear-model object or a formula, the variables on the right-hand-side of the model must all be factors and must be completely crossed.
group	factor defining groups.
center	The name of a function to compute the center of each group; mean gives the original Levene's test; the default, median, provides a more robust test (Brown-Forsythe-Test).
formula	a formula of the form lhs ~ rhs where lhs gives the data values and rhs the corresponding groups.
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
...	arguments to be passed down, e.g., data for the formula and lm methods; can also be used to pass arguments to the function given by center (e.g., center=mean and trim=0.1 specify the 10% trimmed mean).

Value

returns an object meant to be printed showing the results of the test.

Note

This function was previously published as leveneTest() in the library(car) and has been integrated here without logical changes.

Author(s)

John Fox <jfox@mcmaster.ca>; original generic version contributed by Derek Ogle adapted from a response posted by Brian Ripley to the r-help email list.

References

Fox, J. (2008) *Applied Regression Analysis and Generalized Linear Models*, Second Edition. Sage.

Fox, J. and Weisberg, S. (2011) *An R Companion to Applied Regression*, Second Edition, Sage.

See Also

fligner.test for a rank-based (nonparametric) k-sample test for homogeneity of variances; mood.test for another rank-based two-sample test for a difference in scale parameters; var.test and bartlett.test for parametric tests for the homogeneity in variance.

ansari_test in package **coin** for exact and approximate *conditional* p-values for the Ansari-Bradley test, as well as different methods for handling ties.

Examples

```
## example from ansari.test:
## Hollander & Wolfe (1973, p. 86f):
## Serum iron determination using Hyland control sera
ramsay <- c(111, 107, 100, 99, 102, 106, 109, 108, 104, 99,
            101, 96, 97, 102, 107, 113, 116, 113, 110, 98)
jung.parekh <- c(107, 108, 106, 98, 105, 103, 110, 105, 104,
            100, 96, 108, 103, 104, 114, 114, 113, 108, 106, 99)

LeveneTest( c(ramsay, jung.parekh),
  factor(c(rep("ramsay",length(ramsay)), rep("jung.parekh",length(jung.parekh)))))

LeveneTest( c(rnorm(10), rnorm(10, 0, 2)), factor(rep(c("A","B"),each=10)) )

## Not run:
# original example from package car

with(Moore, LeveneTest(conformity, fcategory))
with(Moore, LeveneTest(conformity, interaction(fcategory, partner.status)))

LeveneTest(conformity ~ fcategory * partner.status, data = Moore)
LeveneTest(conformity ~ fcategory * partner.status, data = Moore, center = mean)
LeveneTest(conformity ~ fcategory * partner.status, data = Moore, center = mean, trim = 0.1)

LeveneTest(lm(conformity ~ fcategory*partner.status, data = Moore))

## End(Not run)
```

`LillieTest` *Lilliefors (Kolmogorov-Smirnov) Test for Normality*

Description

Performs the Lilliefors (Kolmogorov-Smirnov) test for the composite hypothesis of normality, see e.g. Thode (2002, Sec. 5.1.1).

Usage

```
LillieTest(x)
```

Arguments

`x`	a numeric vector of data values, the number of which must be greater than 4. Missing values are allowed.

Details

The Lilliefors (Kolmogorov-Smirnov) test is an EDF omnibus test for the composite hypothesis of normality. The test statistic is the maximal absolute difference between empirical and hypothetical cumulative distribution function. It may be computed as $D = \max\{D^+, D^-\}$ with

$$D^+ = \max_{i=1,\ldots,n}\{i/n - p_{(i)}\}, D^- = \max_{i=1,\ldots,n}\{p_{(i)} - (i-1)/n\},$$

where $p_{(i)} = \Phi([x_{(i)} - \overline{x}]/s)$. Here, Φ is the cumulative distribution function of the standard normal distribution, and $\overline{x}$ and s are mean and standard deviation of the data values. The p-value is computed from the Dallal-Wilkinson (1986) formula, which is claimed to be only reliable when the p-value is smaller than 0.1. If the Dallal-Wilkinson p-value turns out to be greater than 0.1, then the p-value is computed from the distribution of the modified statistic $Z = D(\sqrt{n} - 0.01 + 0.85/\sqrt{n})$, see Stephens (1974), the actual p-value formula being obtained by a simulation and approximation process.

Value

A list of class `htest`, containing the following components:

`statistic`	the value of the Lilliefors (Kolomogorv-Smirnov) statistic.
`p.value`	the p-value for the test.
`method`	the character string "Lilliefors (Kolmogorov-Smirnov) normality test".
`data.name`	a character string giving the name(s) of the data.

Note

The Lilliefors (Kolomorov-Smirnov) test is the most famous EDF omnibus test for normality. Compared to the Anderson-Darling test and the Cramer-von Mises test it is known to perform worse. Although the test statistic obtained from `LillieTest(x)` is the same as that obtained from `ks.test(x, "pnorm", mean(x), sd(x))`, it is not correct to use the p-value from the latter for the composite hypothesis of normality (mean and variance unknown), since the distribution of the test statistic is different when the parameters are estimated.

The function call `LillieTest(x)` essentially produces the same result as the S-PLUS function call `ks.gof(x)` with the distinction that the p-value is not set to 0.5 when the Dallal-Wilkinson approximation yields a p-value greater than 0.1. (Actually, the alternative p-value approximation is provided for the complete range of test statistic values, but is only used when the Dallal-Wilkinson approximation fails.)

Author(s)

Juergen Gross <gross@statistik.uni-dortmund.de>

References

Dallal, G.E. and Wilkinson, L. (1986) An analytic approximation to the distribution of Lilliefors' test for normality. *The American Statistician*, 40, 294–296.

Stephens, M.A. (1974) EDF statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, 69, 730–737.

Thode Jr., H.C. (2002) *Testing for Normality* Marcel Dekker, New York.

See Also

`shapiro.test` for performing the Shapiro-Wilk test for normality. `AndersonDarlingTest`, `CramerVonMisesTest`, `PearsonTest`, `ShapiroFranciaTest` for performing further tests for normality. `qqnorm` for producing a normal quantile-quantile plot.

Examples

```
LillieTest(rnorm(100, mean = 5, sd = 3))
LillieTest(runif(100, min = 2, max = 4))
```

---

lines.lm *Add a Linear Regression Line*

Description

Add a linear regression line to an existing plot. The function first calculates the prediction of a `lm` object for a reasonable amount of points, then adds the line to the plot and inserts a polygon with the confidence and, if required, the prediction intervals. In addition to `abline` the function will also display polynomial models.

Usage

```
## S3 method for class 'lm'
lines(x, col = Pal()[1], lwd = 2, lty = "solid",
      type = "l", n = 100, conf.level = 0.95, args.cband = NULL,
      pred.level = NA, args.pband = NULL, xpred = NULL, ...)
```

Arguments

x	linear model object as result from lm(y~x).
col	linecolor of the line. Default is the color returned by Pal()[1].
lwd	line width of the line.
lty	line type of the line.
type	character indicating the type of plotting; actually any of the types as in plot.default. Type of plot, defaults to "l".
n	number of points used for plotting the fit.
conf.level	confidence level for the confidence interval. Set this to NA, if no confidence band should be plotted. Default is 0.95.
args.cband	list of arguments for the confidence band, such as color or border (see DrawBand).
pred.level	confidence level for the prediction interval. Set this to NA, if no prediction band should be plotted. Default is 0.95.
args.pband	list of arguments for the prediction band, such as color or border (see DrawBand).
xpred	a numeric vector c(from, to), if the x limits can't be defined based on available data, xpred can be used to provide the range where the line and especially the confidence intervals should be plotted.
...	further arguments are not used specifically.

Details

It's sometimes illuminating to plot a regression line with its prediction, resp. confidence intervals over an existing scatterplot. This only makes sense, if just a simple linear model explaining a target variable by (a function of) one single predictor is to be visualized.

Value

nothing

Author(s)

Andri Signorell <andri@signorell.net>

See Also

lines, lines.loess, lm

Examples

```
opar <- par(mfrow=c(1,2))

plot(hp ~ wt, mtcars)
lines(lm(hp ~ wt, mtcars), col="steelblue")

# add the prediction intervals in different color
plot(hp ~ wt, mtcars)
r.lm <- lm(hp ~ wt, mtcars)
lines(r.lm, col="red", pred.level=0.95, args.pband=list(col=SetAlpha("grey",0.3)) )

# works with transformations too
plot(dist ~ sqrt(speed), cars)
lines(lm(dist ~ sqrt(speed), cars), col=DescTools::hred)

plot(dist ~ log(speed), cars)
lines(lm(dist ~ log(speed), cars), col=DescTools::hred)

# and with more specific variables based on only one predictor
plot(dist ~ speed, cars)
lines(lm(dist ~ poly(speed, degree=2), cars), col=DescTools::hred)

par(opar)
```

---

lines.loess *Add a Loess or a Spline Smoother*

Description

Add a loess smoother to an existing plot. The function first calculates the prediction of a loess object for a reasonable amount of points, then adds the line to the plot and inserts a polygon with the confidence intervals.

Usage

```
## S3 method for class 'loess'
lines(x, col = Pal()[1], lwd = 2, lty = "solid",
      type = "l", n = 100, conf.level = 0.95, args.band = NULL, ...)

## S3 method for class 'smooth.spline'
lines(x, col = Pal()[1], lwd = 2, lty = "solid",
      type = "l", conf.level = 0.95, args.band = NULL, ...)

## S3 method for class 'SmoothSpline'
lines(x, col = Pal()[1], lwd = 2, lty = "solid",
      type = "l", conf.level = 0.95, args.band = NULL, ...)
```

Arguments

x	the loess or smooth.spline object to be plotted.
col	linecolor of the smoother. Default is DescTools's col1.
lwd	line width of the smoother.
lty	line type of the smoother.
type	type of plot, defaults to "l".
n	number of points used for plotting the fit.
conf.level	confidence level for the confidence interval. Set this to NA, if no confidence band should be plotted. Default is 0.95.
args.band	list of arguments for the confidence band, such as color or border (see DrawBand).
...	further arguments are passed to the smoother (loess() or SmoothSpline()).

Note

Loess can result in heavy computational load if there are many points!

Author(s)

Andri Signorell <andri@signorell.net>

See Also

loess, scatter.smooth, smooth.spline, SmoothSpline

Examples

```
par(mfrow=c(1,2))

x <- runif(100)
y <- rnorm(100)
plot(x, y)
lines(loess(y~x))

plot(temperature ~ delivery_min, data=d.pizza)
lines(loess(temperature ~ delivery_min, data=d.pizza))

plot(temperature ~ delivery_min, data=d.pizza)
lines(loess(temperature ~ delivery_min, data=d.pizza), conf.level = 0.99,
            args.band = list(col=SetAlpha("red", 0.4), border="black") )

# the default values from scatter.smooth
lines(loess(temperature ~ delivery_min, data=d.pizza,
            span=2/3, degree=1, family="symmetric"), col="red")
```

`LineToUser` *Convert Line Coordinates To User Coordinates*

Description

Functions like `mtext` or `axis` use the `line` argument to set the distance from plot. Sometimes it's useful to have the distance in user coordinates. `LineToUser()` does this nontrivial conversion.

Usage

```
LineToUser(line, side)
```

Arguments

`line`	the number of lines
`side`	the side of the plot

Details

For the `LineToUser` function to work, there must be an open plot.

Value

the user coordinates for the given lines

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`mtext`

Examples

```
plot(1:10)
LineToUser(line=2, side=4)
```

LinScale *Linear Scaling*

Description

This will scale the numeric vector x linearly from an old scale between low and high to a new one between newlow and newhigh.

Usage

```
LinScale(x, low = NULL, high = NULL, newlow = 0, newhigh = 1)
```

Arguments

x	a numeric matrix(like object).
low	numeric. The minimum value of the scale, defaults to min(x). This is calculated columnwise by default; defined low or high arguments will be recycled if necessary.
high	numeric. The maximum value of the scale, defaults to max(x). This is calculated columnwise by default; when a maxval is entered, it will be recycled.
newlow	numeric. The minimum value of the new scale, defaults to 0, resulting in a 0-1 scale for x. newlow is recycled if necessary.
newhigh	numeric. The maximum value of the scale, defaults to 1. newhigh is recycled if necessary.

Details

Hmm, hardly worth coding...

Value

The centered and scaled matrix. The numeric centering and scalings used (if any) are returned as attributes "scaled:center" and "scaled:scale"

Author(s)

Andri Signorell <andri@signorell.net>

See Also

scale, RobScale, sweep

Examples

```
# transform the temperature from Celsius to Fahrenheit
LinScale(d.pizza[1:20, "temperature"], 0, 100, -17.8, 37.8 )

# and the price from Dollar to Euro
LinScale(d.pizza[1:20, "price"], 0, 1, 0, 0.76)

# together
LinScale(d.pizza[1:20, c("temperature", "price")],
  0, c(100, 1), c(-17.8, 0), c(37.8, 0.76) )


## Not run:
par(mfrow=c(3,1), mar=c(0,5,0,3), oma=c(5,0,5,0))
plot(LinScale(d.frm[,1]), ylim=c(-2,2), xaxt="n", ylab="LinScale")
plot(RobScale(d.frm[,1]), ylim=c(-2,2), xaxt="n", ylab="RobScale")
plot(scale(d.frm[,1]), ylim=c(-2,2), ylab="scale")
title("Compare scales", outer = TRUE)

## End(Not run)
```

List Variety Of Objects

List Objects, Functions Or Data in a Package

Description

List all the objects, functions or data in a package.

Usage

```
LsObj(package)
LsFct(package)
```

Arguments

package	the name of the package

Details

This is just a wrapper for `ls`, `ls.str` and `lsf.str` with the appropriate arguments (as I always forgot how to do the trick). `LsObj()` lists all objects, `LsFct()` just the functions in a package.

Author(s)

Andri Signorell <andri@signorell.net>

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

ls, ls.str, lsf.str

Examples

```
LsFct("DescTools")
```

LOCF *Last Observation Carried Forward*

Description

In longitudinal studies it's common that individuals drop out before all responses can be obtained. Measurements obtained before the individual dropped out can be used to impute the unknown measurement(s). The last observation carried forward method is one way to impute values for the missing observations. For the last observation carried forward (LOCF) approach the missing values are replaced by the last observed value of that variable for each individual regardless of when it occurred.

LOCF() replaces NAs with the most recent non-NA prior to it.

Usage

```
LOCF(x)

## Default S3 method:
LOCF(x)
## S3 method for class 'data.frame'
LOCF(x)
## S3 method for class 'matrix'
LOCF(x)
```

Arguments

x	a vector, a data.frame or a matrix containing NAs.

Details

The function will replace all NAs found in a vector with the last earlier value not being NA. In data.frames each column will be treated as described.

It should be noted, that the last observation carried forward approach may result in biased estimates and may underestimate the variability.

Value

a vector with the same dimension as x.

Author(s)

Daniel Wollschlaeger <dwoll@psychologie.uni-kiel.de>

See Also

See also the package **Hmisc** for less coarse imputation functions.

Examples

```
d.frm <- data.frame(
  tag=rep(c("mo", "di", "mi", "do", "fr", "sa", "so"), 4)
, val=rep(c(runif(5), rep(NA,2)), 4) )

d.frm$locf <- LOCF( d.frm$val )
d.frm
```

LOF *Local Outlier Factor*

Description

A function that finds the local outlier factor (Breunig et al.,2000) of the matrix "data" using k neighbours. The local outlier factor (LOF) is a measure of outlyingness that is calculated for each observation. The user decides whether or not an observation will be considered an outlier based on this measure. The LOF takes into consideration the density of the neighborhood around the observation to determine its outlyingness.

Usage

```
LOF(data, k)
```

Arguments

data	The data set to be explored
k	The kth-distance to be used to calculate the LOF's.

Details

The LOFs are calculated over a range of values, and the max local outlier factor is determined over this range.

Value

lof	A vector with the local outlier factor of each observation

Note

This function was originally published in the library dprep.

Author(s)

Caroline Rodriguez

References

Breuning, M., Kriegel, H., Ng, R.T, and Sander. J. (2000). LOF: Identifying density-based local outliers. *In Proceedings of the ACM SIGMOD International Conference on Management of Data*

Examples

```
# Detecting the top 10 outliers using the LOF algorithm

(iris.lof <- LOF(iris[,-5], 10))
```

Logit *Generalized Logit and Inverse Logit Function*

Description

Compute generalized logit and generalized inverse logit functions.

Usage

```
Logit(x, min = 0, max = 1)
LogitInv(x, min = 0, max = 1)
```

Arguments

x	value(s) to be transformed
min	lower end of logit interval
max	upper end of logit interval

Details

The generalized logit function takes values on [min, max] and transforms them to span $[-\infty, \infty]$. It is defined as:

$$y = log\left(\frac{p}{1-p}\right) \quad \text{where} \quad p = \frac{x - min}{max - min}$$

The generalized inverse logit function provides the inverse transformation:

$$x = p' \cdot (max - min) + min \quad \text{where} \quad p' = \frac{exp(y)}{1 + exp(y)}$$

Value

Transformed value(s).

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

logit

Examples

```
x <- seq(0,10, by=0.25)
xt <- Logit(x, min=0, max=10)
cbind(x,xt)

y <- LogitInv(xt, min=0, max=10)
cbind(x, xt, y)
```

LogSt *Started Logarithmic Transformation and Its Inverse*

Description

Transforms the data by a log transformation, modifying small and zero observations such that the transformation is linear for $x <= threshold$ and logarithmic for x > threshold. So the transformation yields finite values and is continuously differentiable.

Usage

```
LogSt(x, base = 10, calib = x, threshold = NULL, mult = 1)

LogStInv(x, base = NULL, threshold = NULL)
```

Arguments

x	a vector or matrix of data, which is to be transformed
base	a positive or complex number: the base with respect to which logarithms are computed. Defaults to 10. Use=exp(1) for natural log.
calib	a vector or matrix of data used to calibrate the transformation(s), i.e., to determine the constant c needed
threshold	constant c that determines the transformation. The inverse function LogStInv will look for an attribute named "threshold" if the argument is set to NULL.
mult	a tuning constant affecting the transformation of small values, see Details.

Details

In order to avoid $log(x) = -\infty$ for $x = 0$ in log-transformations there's often a constant added to the variable before taking the log. This is not always a pleasable strategy. The function `LogSt` handles this problem based on the following ideas:

- The modification should only affect the values for "small" arguments.
- What "small" is should be determined in connection with the non-zero values of the original variable, since it should behave well (be equivariant) with respect to a change in the "unit of measurement".
- The function must remain monotone, and it should remain (weakly) convex.

These criteria are implemented here as follows: The shape is determined by a threshold c at which - coming from above - the log function switches to a linear function with the same slope at this point.

This is obtained by

$$g(x) = \begin{cases} log_{10}(x) & \text{for } x \geq c \\ log_{10}(c) - \frac{c-x}{c \cdot log(10)} & \text{for } x < c \end{cases}$$

Small values are determined by the threshold c. If not given by the argument `threshold`, it is determined by the quartiles q_1 and q_3 of the non-zero data as those smaller than $c = \frac{q_1^{1+r}}{q_3^r}$ where r can be set by the argument `mult`. The rationale is, that, for lognormal data, this constant identifies 2 percent of the data as small.
Beyond this limit, the transformation continues linear with the derivative of the log curve at this point.

Another idea for choosing the threshold c was: median(x) / (median(x)/quantile(x, 0.25))^2.9)

The function chooses log_{10} rather than natural logs by default because they can be backtransformed relatively easily in mind.

A generalized log (see: Rocke 2003) can be calculated in order to stabilize the variance as:

```
function (x, a) {
 return(log((x + sqrt(x^2 + a^2)) / 2))
}
```

Value

the transformed data. The value c used for the transformation and needed for inverse transformation is returned as `attr(.,"threshold")` and the used base as `attr(.,"base")`.

Author(s)

Werner A. Stahel, ETH Zurich
slight modifications Andri Signorell <andri@signorell.net>

References

Rocke, D M, Durbin B (2003): Approximate variance-stabilizing transformations for gene-expression microarray data, *Bioinformatics*. 22;19(8):966-72.

See Also

log, log10

Examples

```
dd <- c(seq(0,1,0.1), 5 * 10^rnorm(100, 0, 0.2))
dd <- sort(dd)
r.dl <- LogSt(dd)
plot(dd, r.dl, type="l")
abline(v=attr(r.dl, "threshold"), lty=2)

x <- rchisq(df=3, n=100)
# should give 0 (or at least something small):
LogStInv(LogSt(x)) - x
```

MAD *Median Absolute Deviation*

Description

Compute the median absolute deviation, i.e., the (lo-/hi-) median of the absolute deviations from the median, and (by default) adjust by a factor for asymptotically normal consistency. This function wraps the specific base R function mad and extends it for the use of weights.

Usage

```
MAD(x, weights = NULL, center = Median, constant = 1.4826,
    na.rm = FALSE, low = FALSE, high = FALSE)
```

Arguments

x	a numeric vector.
weights	a numerical vector of weights the same length as x giving the weights to use for elements of x.
center	the centre given either as numeric value or as a function to be applied to x (defaults to the DescTools::Median(x)). Note in cases when weights are defined to provide a function that also support weights. If this is not possible fall back to a numeric value.
constant	scale factor (default is 1.4826)
na.rm	if TRUE then NA values are stripped from x before computation takes place.
low	if TRUE, compute the 'lo-median', i.e., for even sample size, do not average the two middle values, but take the smaller one.
high	if TRUE, compute the 'hi-median', i.e., take the larger of the two middle values for even sample size.

Details

The actual value calculated is `constant * cMedian(abs(x - center))` with the default value of `center` being `median(x)`, and `cMedian` being the usual, the 'low' or 'high' median, see the arguments description for `low` and `high` above.

The default `constant = 1.4826` (approximately $1/\Phi^{-1}(\frac{3}{4})$ = `1/qnorm(3/4)`) ensures consistency, i.e.,

$$E[mad(X_1, \ldots, X_n)] = \sigma$$

for X_i distributed as $N(\mu, \sigma^2)$ and large n.

If `na.rm` is `TRUE` then `NA` values are stripped from `x` before computation takes place. If this is not done then an `NA` value in `x` will cause `MAD` to return `NA`.

See Also

`IQR` which is simpler but less robust, `IQRw` for weights, `mad`, `median`, `var`, `MADCI` (confidence intervals).

Examples

```
MAD(c(1:9))
print(MAD(c(1:9),     constant = 1)) ==
      MAD(c(1:8, 100), constant = 1)       # = 2 ; TRUE
x <- c(1,2,3,5,7,8)
sort(abs(x - median(x)))
c(MAD(x, constant = 1),
  MAD(x, constant = 1, low = TRUE),
  MAD(x, constant = 1, high = TRUE))

# use weights
x <- sample(20, 30, replace = TRUE)
z <- as.numeric(names(w <- table(x)))

(m1 <- MAD(z, weights=w))
(m2 <- MAD(x))
stopifnot(identical(m1, m2))
```

`MADCI` *Confidence Intervals for Median Absolute Deviations*

Description

A function for the median absolute deviation is included in base R, `mad`, but there's no function for calculating confidence intervals. Arachchige/Prendergast introduce interval estimators of the MAD to make reliable inferences for dispersion for a single population and ratios and differences of MADs for comparing two populations.

Usage

```
MADCI(x, y = NULL, two.samp.diff = TRUE, gld.est = "TM",
      conf.level = 0.95, sides = c("two.sided","left","right"),
      na.rm = FALSE, ...)
```

Arguments

x	a (non-empty) numeric vector of data values.
y	a second (non-empty) numeric vector of data values.
two.samp.diff	logical, defining if the confidence intervals for the difference (mad(x)-mad(y)) (default) or for the squared ratio ((mad(x)/mad(y))^2) should be calculated. Ignored if y is not given.
gld.est	A character string, to select the estimation method for the generalized lambda distribution. One of: ML for numerical Maximum Likelihood, MPS or MSP for Maximum Spacings Product, TM for Titterington's Method (default), SM for Starship Method, TL for method of Trimmed L-moments, Lmom for method of L-moments, DLA for the method of Distributional Least Absolutes, or Mom for method of Moments. See fit.fkml().
conf.level	confidence level of the interval.
sides	a character string specifying the side of the confidence interval, must be one of "two.sided" (default), "left" or "right". You can specify just the initial letter. "left" would be analogue to a hypothesis of "greater" in a t.test.
na.rm	logical. Should missing values be removed? Defaults to FALSE.
...	further arguments, not used here

Value

a numeric vector with 3 elements:

mad	median absolute deviation
lwr.ci	lower bound of the confidence interval
upr.ci	upper bound of the confidence interval

Author(s)

Arachchige Chandima N. P. G., Prendergast Luke A., Andri Signorell <andri@signorell.net> (only interface)

References

Arachchige Chandima N. P. G., Prendergast Luke A. (2019) Confidence intervals for median absolute deviations, arXiv:1910.00229 [math.ST]

See Also

mad, MAD

Examples

```
x <- rlnorm(100)
y <- rlnorm(200, meanlog=1.2)

MADCI(x)                          # single sample

MADCI(x, y)                       # two sample difference
MADCI(x, y, two.samp.diff = FALSE) # two sample squared ratio
```

Mar and Mgp *Set Plot Margins and Distances*

Description

Plot margins are normally set by `par("mar")`. However one is forced to always define all margins, even if just one should be altered. The convenience function `Mar()` allows to set one single margin (or several) while leaving the others unchanged.

`Mgp()` does the same for the distances of axis title, labels and line.

Usage

```
Mar(bottom = NULL, left = NULL, top = NULL, right = NULL, outer = FALSE,
    reset = FALSE)
Mgp(title = NULL, labels = NULL, line = NULL, reset = FALSE)
```

Arguments

`bottom`	the bottom margin, if set to `NULL` the current value will be maintained.
`left`	the left margin, if set to `NULL` the current value will be maintained.
`top`	the top margin, if set to `NULL` the current value will be maintained.
`right`	the right margin, if set to `NULL` the current value will be maintained.
`outer`	logical, defining if inner margins (`par("mar")`) or the outer margins (`par("oma")`) should be set. Default is `FALSE`, meaning that the inner margins will be concerned.
`reset`	if set to `TRUE` the margins are reset to the defaults (respecting `outer`). Other arguments are ignored.
`title`	margin line for the axis title (default 3)
`labels`	margin line for the axis labels (default 1)
`line`	margin line for the axis line (default 0)

Details

Running `Mar()` without any arguments will return the current settings, either `par("mar")`, when outer is set to `FALSE` or `par("oma")` for `outer = TRUE`.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

par

Examples

```
# largen the left margin only
Mar(left=10.1)   # or as alternative: Mar(, 10.1)
Mgp(title=6)     # ylab must be placed a little further to the left
barplot(1:7, names=levels(d.pizza$driver), horiz=TRUE, las=1,
        ylab="driver", col=Pal("Helsana"))
```

---

matpow *Matrix Power*

Description

Compute the k-th power of a matrix. Whereas `x^k` computes *element wise* powers, `x %^% k` corresponds to $k-1$ matrix multiplications, `x %*% x %*% ... %*% x`.

Usage

```
x %^% k
```

Arguments

x	a square matrix.
k	an integer, $k \geq 0$.

Details

Argument k is coerced to integer using as.integer.

The algorithm uses $O(log_2(k))$ matrix multiplications.

Value

A matrix of the same dimension as x.

Note

If you think you need `x^k` for $k < 0$, then consider instead `solve(x %^% (-k))`.

Author(s)

Based on an R-help posting of Vicente Canto Casasola, and Vincent Goulet's C implementation in **actuar**.

See Also

%*% for matrix multiplication.

Examples

```
A <- cbind(1, 2 * diag(3)[,-1])
A
A %^% 2
stopifnot(identical(A, A %^% 1),
          A %^% 2 == A %*% A)
```

---

mbeta *Mean and Variance of the Beta Distribution*

Description

Formula: $\mu = \frac{\alpha}{\alpha+\beta}$ $\mathrm{Var}(X) = \frac{\alpha \cdot \beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$

Usage

```
mbeta(shape1, shape2)
```

Arguments

shape1	Alpha parameter
shape2	Beta parameter

Value

List with mean and variance

See Also

dbeta

Examples

```
mbeta(shape1 = 2, shape2 = 3)
```

mbinom *Mean and Variance of the Binomial Distribution*

Description

Formula: $\mu = n \cdot p \; \mathrm{Var}(X) = n \cdot p \cdot (1 - p)$

Usage

```
mbinom(size, prob)
```

Arguments

size	Number of trials
prob	Probability of success

Value

List with mean and variance

See Also

dbinom

Examples

```
mbinom(size = 10, prob = 0.3)
```

mchisq *Mean and Variance of the Chi-Squared Distribution*

Description

Formula: $\mu = df \; \mathrm{Var}(X) = 2 \cdot df$

Usage

```
mchisq(df)
```

Arguments

df	Degrees of freedom

Value

List with mean and variance

See Also

dchisq

Examples

```
mchisq(df = 4)
```

Mean *(Weighted) Arithmetic Mean*

Description

Generic function for the (trimmed) arithmetic mean, possibly with given weights.

Usage

```
Mean(x, ...)

## S3 method for class 'Freq'
Mean(x, breaks, ...)

## Default S3 method:
Mean(x, weights = NULL, trim = 0, na.rm = FALSE, ...)
```

Arguments

x	An object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for `trim = 0`, only.
...	further arguments passed to or from other methods.
breaks	breaks for calculating the mean for classified data as composed by Freq.
weights	a numerical vector of weights the same length as x giving the weights to use for elements of x.
trim	the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
na.rm	a logical value indicating whether NA values should be stripped before the computation proceeds.

Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed, as a numeric or complex vector of length one. If `x` is not logical (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned, with a warning.

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

`trim` and `weights` can't be used together at the same time.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`weighted.mean`, `mean.POSIXct`, `colMeans` for row and column means.

Examples

```
x <- c(0:10, 50)
xm <- Mean(x)
c(xm, Mean(x, trim = 0.10))
```

`MeanAD` *Mean Absolute Deviation From a Center Point*

Description

Calculates the mean absolute deviation from a center point, typically the sample mean or the median. %% ~~ A concise (1-5 lines) description of what the function does. ~~

Usage

```
MeanAD(x, weights = NULL, center = Mean, na.rm = FALSE)
```

Arguments

`x`	a vector containing the observations. %% ~~Describe `x` here~~
`weights`	a numerical vector of weights the same length as `x` giving the weights to use for elements of `x`.
`center`	a single numerical value or the name of a function applied to `x` to be used as center. Can as well be a self defined function. Default is `Mean()`.
`na.rm`	a logical value indicating whether or not missing values should be removed. Defaults to `FALSE`.

Details

The `MeanAD` function calculates the mean absolute deviation from the mean value (or from another supplied center point) of x, after having removed `NA` values (if requested):

$$\frac{1}{n} \cdot \sum_{i=1}^{n} |x_i - c| \quad \text{where } c = mean(x) \text{ or } c = med(x)$$

The function supports the use of weights. The default function for the center value `Mean()` has a weights arguments, too. If a user defined function is used it must be assured that it has a weights argument.

Value

Numeric value.

Author(s)

Andri Signorell andri@signorell.net following an idea of Danielle Navarro (aad in the **lsr** package)

See Also

mad

Examples

```
x <- runif(100)
MeanAD(x)

speed <- c(58, 88, 40, 60, 72, 66, 80, 48, NA)
MeanAD(speed)
MeanAD(speed, na.rm=TRUE)


# using the median as centerpoint
x <- c(2,3,5,3,1,15,23)

MeanAD(x, center=mean)
MeanAD(x, center=median)

# define a fixed center
MeanAD(x, center=4)

# use of weights
MeanAD(x=0:6, weights=c(21,46,54,40,24,10,5))
```

MeanCI *Confidence Intervals for the Mean*

Description

Collection of several approaches to determine confidence intervals for the mean. Both, the classical way and bootstrap intervals are implemented for both, normal and trimmed means.

Usage

```
MeanCI(
  x,
  sd = NULL,
  trim = 0,
  conf.level = 0.95,
```

```
  sides = c("two.sided", "left", "right"),
  method = c("classic", "boot"),
  na.rm = FALSE,
  ...
)
```

Arguments

x	a (non-empty) numeric vector of data values.
sd	the standard deviation of x. If provided it's interpreted as sd of the population and the normal quantiles will be used for constructing the confidence intervals. If left to NULL (default) the sample sd(x) will be calculated and used in combination with the t-distribution.
trim	the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
conf.level	confidence level of the interval.
sides	a character string specifying the side of the confidence interval, must be one of "two.sided" (default), "left" or "right". "left" would be analogue to a hypothesis of "greater" in a t.test. You can specify just the initial letter.
method	A vector of character strings representing the type of intervals required. The value should be any subset of the values "classic", "boot". See boot.ci.
na.rm	a logical value indicating whether NA values should be stripped before the computation proceeds. Defaults to FALSE.
...	further arguments are passed to the boot function. Supported arguments are type ("norm", "basic", "stud", "perc", "bca"), parallel and the number of bootstrap replicates R. If not defined those will be set to their defaults, being "basic" for type, option "boot.parallel" (and if that is not set, "no") for parallel and 999 for R.

Details

The confidence intervals for the trimmed means use winsorized variances as described in the references.

Value

a numeric vector with 3 elements:

mean	mean
lwr.ci	lower bound of the confidence interval
upr.ci	upper bound of the confidence interval

Author(s)

Andri Signorell andri@signorell.net

References

Wilcox, R. R., Keselman H. J. (2003) Modern robust data analysis methods: measures of central tendency *Psychol Methods*, 8(3):254-74

Wilcox, R. R. (2005) *Introduction to robust estimation and hypothesis testing* Elsevier Academic Press

See Also

Mean, t.test, MeanDiffCI, MedianCI, VarCI, MeanCIn

Examples

```
x <- d.pizza$price[1:20]

MeanCI(x, na.rm=TRUE)
MeanCI(x, conf.level=0.99, na.rm=TRUE)

MeanCI(x, sides="left")
# same as:
t.test(x, alternative="greater")

MeanCI(x, sd=25, na.rm=TRUE)

# the different types of bootstrap confints
MeanCI(x, method="boot", type="norm", na.rm=TRUE)
MeanCI(x, trim=0.1, method="boot", type="norm", na.rm=TRUE)
MeanCI(x, trim=0.1, method="boot", type="basic", na.rm=TRUE)
MeanCI(x, trim=0.1, method="boot", type="stud", na.rm=TRUE)
MeanCI(x, trim=0.1, method="boot", type="perc", na.rm=TRUE)
MeanCI(x, trim=0.1, method="boot", type="bca", na.rm=TRUE)

MeanCI(x, trim=0.1, method="boot", type="bca", R=1999, na.rm=TRUE)

# Getting the MeanCI for more than 1 column
round(t(sapply(d.pizza[, 1:4], MeanCI, na.rm=TRUE)), 3)
```

MeanCIn *Sample Size for a Given Width of a Confidence Interval for a Mean*

Description

Returns the required sample size to obtain a given width of a confidence interval for the sample mean. The function uses uniroot() to find a numeric solution. The t distribution is used.

Usage

```
MeanCIn(ci, sd, interval = c(2, 100000), conf.level = 0.95,
        norm = FALSE, tol = .Machine$double.eps^0.5)
```

Arguments

ci	the left and right bound of the interval, which is presumed to be symmetric.
sd	the standard deviation of the sample.
interval	the interval for the sample size to be searched into, (default is c(2, 100000)).
conf.level	confidence level, defaults to 0.95.
norm	logical, determining if the t- or normaldistribution should be used.
tol	the desired accuracy (convergence tolerance).

Details

The required sample sizes for a specific width of confidence interval for the mean depends recursively on the sample size, as the samplesize defines the degrees of freedom in the t-distribution. Although in most practical cases it will be sufficient to use the normal distribution, we might be interested in exact results.

Value

a numeric value

Author(s)

Andri Signorell <andri@signorell.net>

See Also

BinomCIn()

Examples

```
MeanCIn(ci=c(25, 27), sd=5)
```

MeanDiffCI *Confidence Interval For Difference of Means*

Description

Calculates the confidence interval for the difference of two means either the classical way or with the bootstrap approach.

Usage

```
MeanDiffCI(x, ...)

## Default S3 method:
MeanDiffCI(x, y, method = c("classic", "norm", "basic", "stud", "perc", "bca"),
        conf.level = 0.95, sides = c("two.sided", "left", "right"), paired = FALSE,
            na.rm = FALSE, R = 999, ...)

## S3 method for class 'formula'
MeanDiffCI(formula, data, subset, na.action, ...)
```

Arguments

x	a (non-empty) numeric vector of data values.
y	a (non-empty) numeric vector of data values.
method	a vector of character strings representing the type of intervals required. The value should be any subset of the values "classic", "norm", "basic", "stud", "perc", "bca". See boot.ci.
conf.level	confidence level of the interval.
sides	a character string specifying the side of the confidence interval, must be one of "two.sided" (default), "left" or "right". You can specify just the initial letter. "left" would be analogue to a hypothesis of "greater" in a t.test.
paired	a logical indicating whether you want confidence intervals for a paired design. Defaults to FALSE.
na.rm	logical. Should missing values be removed? Defaults to FALSE.
R	the number of bootstrap replicates. Usually this will be a single positive integer. For importance resampling, some resamples may use one set of weights and others use a different set of weights. In this case R would be a vector of integers where each component gives the number of resamples from each of the rows of weights. See boot.
formula	a formula of the form lhs ~ rhs where lhs is a numeric variable giving the data values and rhs a factor with two levels giving the corresponding groups.
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
...	further argument to be passed to or from methods.

Details

This function collects code from two sources. The classical confidence interval is calculated by means of t.test. The bootstrap intervals are strongly based on the example in boot.

Value

a numeric vector with 3 elements:

meandiff	the difference: mean(x) - mean(y)
lwr.ci	lower bound of the confidence interval
upr.ci	upper bound of the confidence interval

Author(s)

Andri Signorell <andri@signorell.net>

See Also

MeanCI, VarCI, MedianCI, boot.ci

Examples

```
x <- d.pizza$price[d.pizza$driver=="Carter"]
y <- d.pizza$price[d.pizza$driver=="Miller"]

MeanDiffCI(x, y, na.rm=TRUE)
MeanDiffCI(x, y, conf.level=0.99, na.rm=TRUE)

# the different types of bootstrap confints
MeanDiffCI(x, y, method="norm", na.rm=TRUE)
MeanDiffCI(x, y, method="basic", na.rm=TRUE)
# MeanDiffCI(x, y, method="stud", na.rm=TRUE)
MeanDiffCI(x, y, method="perc", na.rm=TRUE)
MeanDiffCI(x, y, method="bca", na.rm=TRUE)

# the formula interface
MeanDiffCI(price ~ driver, data=d.pizza, subset=driver %in% c("Carter","Miller"))
```

MeanSE *Standard Error of Mean*

Description

Calculates the standard error of mean.

Usage

```
MeanSE(x, sd = NULL, na.rm = FALSE)
```

Arguments

x	a (non-empty) numeric vector of data values.
sd	the standard deviation of x. If provided it's interpreted as sd of the population. If left to NULL (default) the sample sd(x) will be used.
na.rm	logical. Should missing values be removed? Defaults to FALSE.

Details

MeanSE calculates the standard error of the mean defined as:

$$\frac{\sigma}{\sqrt{n}}$$

σ being standard deviation of x and n the length of x.

Value

the standard error as numeric value.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

MeanCI

Examples

```
data(d.pizza)

MeanSE(d.pizza$price, na.rm=TRUE)

# evaluate data.frame
sapply(d.pizza[,1:4], MeanSE, na.rm=TRUE)
```

Measures of Accuracy *Measures of Accuracy*

Description

Some measures of model accuracy like mean absolute error (MAE), mean absolute percentage error (MAPE), symmetric mean absolute percentage error (SMAPE), mean squared error (MSE) and root mean squared error (RMSE).

Usage

```
MAE(x, ...)
## Default S3 method:
MAE(x, ref, na.rm = FALSE, ...)
## S3 method for class 'lm'
MAE(x, ...)

MAPE(x, ...)
## Default S3 method:
MAPE(x, ref, na.rm = FALSE, ...)
```

```
## S3 method for class 'lm'
MAPE(x, ...)

SMAPE(x, ...)
## Default S3 method:
SMAPE(x, ref, na.rm = FALSE, ...)
## S3 method for class 'lm'
SMAPE(x, ...)

MSE(x, ...)
## Default S3 method:
MSE(x, ref, na.rm = FALSE, ...)
## S3 method for class 'lm'
MSE(x, ...)

RMSE(x, ...)
## Default S3 method:
RMSE(x, ref, na.rm = FALSE, ...)
## S3 method for class 'lm'
RMSE(x, ...)


NMAE(x, ref, train.y)
NMSE(x, ref, train.y)
```

Arguments

x	the predicted values of a model or a model-object itself.
ref	the observed true values.
train.y	the observed true values in a train dataset.
na.rm	a logical value indicating whether or not missing values should be removed. Defaults to FALSE.
...	further arguments

Details

The function will remove NA values first (if requested).
MAE calculates the mean absolute error:

$$\frac{1}{n} \cdot \sum_{i=1}^{n} |ref_i - x_i|$$

MAPE calculates the mean absolute percentage error:

$$\frac{1}{n} \cdot \sum_{i=1}^{n} \left| \frac{ref_i - x_i}{ref_i} \right|$$

SMAPE calculates the symmetric mean absolute percentage error:

$$\frac{1}{n} \cdot \sum_{i=1}^{n} \frac{2 \cdot |ref_i - x_i|}{|ref_i| + |x_i|}$$

MSE calculates mean squared error:

$$\frac{1}{n} \cdot \sum_{i=1}^{n} (ref_i - x_i)^2$$

RMSE calculates the root mean squared error:

$$\sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} (ref_i - x_i)^2}$$

Value

the specific numeric value

Author(s)

Andri Signorell <andri@signorell.net>

References

Armstrong, J. S. (1985) *Long-range Forecasting: From Crystal Ball to Computer*, 2nd. ed. Wiley. ISBN 978-0-471-82260-8
https://en.wikipedia.org/wiki/Symmetric_mean_absolute_percentage_error

Torgo, L. (2010) *Data Mining with R: Learning with Case Studies*, Chapman and Hall/CRC Press

See Also

lm, resid

Examples

```
r.lm <- lm(Fertility ~ ., data=swiss)

MAE(r.lm)

# the same as:
```

```
MAE(predict(r.lm), swiss$Fertility)

MAPE(r.lm)
MSE(r.lm)
RMSE(r.lm)
```

`Measures of Shape` *Skewness and Kurtosis*

Description

`Skew` computes the skewness, `Kurt` the excess kurtosis of the values in `x`.

Usage

```
Skew(x, weights = NULL, na.rm = FALSE, method = 3, conf.level = NA,
     ci.type = "bca", R = 1000, ...)

Kurt(x, weights = NULL, na.rm = FALSE, method = 3, conf.level = NA,
     ci.type = "bca", R = 1000, ...)
```

Arguments

`x`	a numeric vector. An object which is not a vector is coerced (if possible) by `as.vector`.
`weights`	a numerical vector of weights the same length as `x` giving the weights to use for elements of `x`.
`na.rm`	logical, indicating whether `NA` values should be stripped before the computation proceeds. Defaults to `FALSE`.
`method`	integer out of 1, 2 or 3 (default). See Details.
`conf.level`	confidence level of the interval. If set to `NA` (which is the default) no confidence interval will be calculated.
`ci.type`	The type of confidence interval required. The value should be any subset of the values `"classic"`, `"norm"`, `"basic"`, `"stud"`, `"perc"` or `"bca"` (`"all"` which would compute all five types of intervals, is not supported).
`R`	The number of bootstrap replicates. Usually this will be a single positive integer. For importance resampling, some resamples may use one set of weights and others use a different set of weights. In this case `R` would be a vector of integers where each component gives the number of resamples from each of the rows of weights.
`...`	the dots are passed to the function `boot`, when confidence intervalls are calculated.

Details

`Kurt()` returns the excess kurtosis, therefore the kurtosis calculates as `Kurt(x) + 3` if required.

If `na.rm` is `TRUE` then missing values are removed before computation proceeds.

The methods for calculating the skewness can either be:
`method = 1: g_1 = m_3 / m_2^(3/2)`
`method = 2: G_1 = g_1 * sqrt(n(n-1)) / (n-2)`
`method = 3: b_1 = m_3 / s^3 = g_1 ((n-1)/n)^(3/2)`

and the ones for the kurtosis:
`method = 1: g_2 = m_4 / m_2^2 - 3`
`method = 2: G_2 = ((n+1) g_2 + 6) * (n-1) / ((n-2)(n-3))`
`method = 3: b_2 = m_4 / s^4 - 3 = (g_2 + 3) (1 - 1/n)^2 - 3`

method = 1 is the typical definition used in Stata and in many older textbooks.
method = 2 is used in SAS and SPSS.
method = 3 is used in MINITAB and BMDP.

Cramer (1997) mentions the asymptotic standard error of the skewness, resp. kurtosis:

```
ASE.skew = sqrt( 6*n*(n-1)/((n-2)*(n+1)*(n+3)) )
ASE.kurt = sqrt((24*n*(n - 1)^2) / ((n - 3)*(n - 2)*(n + 3)*(n + 5)))
```

to be used for calculating the confidence intervals. This is implemented here with `ci.type="classic"`. However, Joanes and Gill (1998) advise against this approach, pointing out that the normal assumptions would virtually always be violated. They suggest using the bootstrap method. That's why the default method for the confidence interval type is set to `"bca"`.

This implementation of the two functions is comparably fast, as the expensive sums are coded in C.

Value

If `conf.level` is set to `NA` then the result will be

`a`	single numeric value

and if a `conf.level` is provided, a named numeric vector with 3 elements:

`skew`, `kurt`	the specific estimate, either skewness or kurtosis
`lwr.ci`	lower bound of the confidence interval
`upr.ci`	upper bound of the confidence interval

Author(s)

Andri Signorell <andri@signorell.net>, David Meyer <david.meyer@r-project.org> (method = 3)

References

Cramer, D. (1997): *Basic Statistics for Social Research* Routledge.

Joanes, D. N., Gill, C. A. (1998): Comparing measures of sample skewness and kurtosis. *The Statistician*, 47, 183-189.

See Also

mean, sd, similar code in `library(e1071)`

Examples

```
Skew(d.pizza$price, na.rm=TRUE)
Kurt(d.pizza$price, na.rm=TRUE)

# use sapply to calculate skewness for a data.frame
sapply(d.pizza[,c("temperature","price","delivery_min")], Skew, na.rm=TRUE)

# or apply to do that columnwise with a matrix
apply(as.matrix(d.pizza[,c("temperature","price","delivery_min")]), 2, Skew, na.rm=TRUE)
```

Median *(Weighted) Median Value*

Description

Compute the sample median. The function basically wraps the function `Quantile()`, which offers the option to define weights.
For grouped data the median can be estimated by linear interpolation within the class containing the median, which is implemented in the interface for `Freq`-objects.

Usage

```
Median(x, ...)

## S3 method for class 'factor'
Median(x, na.rm = FALSE, ...)

## S3 method for class 'Freq'
Median(x, breaks, ...)

## Default S3 method:
Median(x, weights = NULL, na.rm = FALSE, ...)
```

Arguments

x	an object for which a method has been defined, or a numeric vector containing the values whose median is to be computed.
weights	a numerical vector of weights the same length as x giving the weights to use for elements of x.
breaks	breaks for calculating the mean for classified data as composed by Freq.
na.rm	a logical value indicating whether NA values should be stripped before the computation proceeds.
...	further arguments passed to or from other methods.

Details

This is a generic function for which methods can be written. However, the default method makes use of is.na, sort and mean from package **base** all of which are generic, and so the default method will work for most classes (e.g., "Date") for which a median is a reasonable concept.

Calculating the median for ordered factors is not implemented in standard R, as it's not well defined (it is not clear what to do if the median sits between two levels in factors of even length). This function returns the high median and prints a warning if the low median would be different (which is supposed to be a rare event). There's a vivid discussion between experts going on whether this should be defined or not. We'll wait for definitive results and enjoy the function's comfort so far...

Note that there are alternative approaches for calculating weighted median (e.g. matrixstats::weightedMedian).

Value

The default method returns a length-one object of the same type as x, except when x is integer of even length, when the result will be double.

If there are no values or if na.rm = FALSE and there are NA values the result is NA of the same type as x (or more generally the result of x[FALSE][NA]).

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

quantile for general quantiles. https://stat.ethz.ch/pipermail/r-help/2003-November/042684.html

https://stackoverflow.com/questions/7925102/idiomatic-method-of-finding-the-median-of-an-ordinal

Examples

```
Median(1:4)               # = 2.5 [even number]
Median(c(1:3, 100, 1000))  # = 3 [odd, robust]

# Approximation for classified data
breaks <- seq(10,70, 10)
```

```
Median(
  Freq(cut(d.pizza$temperature, breaks=breaks)),
  breaks=breaks)

# compared to
Median(d.pizza$temperature)

# starting from a classified table
# from     to  income
#    0   4000      20
# 4000   6000      42
# 6000   8000      31
# 8000  10000      12

# Freq(as.table(c(20,42,31,12)))
#    level  freq   perc  cumfreq  cumperc
# 1      A    20  19.0%       20    19.0%
# 2      B    42  40.0%       62    59.0%
# 3      C    31  29.5%       93    88.6%
# 4      D    12  11.4%      105   100.0%

Median(Freq(as.table(c(20,42,31,12))), breaks=c(0,4000,6000,8000,10000))

# use weights
x <- sample(20, 30, replace = TRUE)
z <- as.numeric(names(w <- table(x)))

(m1 <- Median(z, weights=w))
(m2 <- Median(x))
stopifnot(identical(m1, m2))
```

MedianCI *Confidence Interval for the Median*

Description

Calculate the confidence interval for the median.

Usage

```
MedianCI(
  x,
  conf.level = 0.95,
  sides = c("two.sided", "left", "right"),
  method = c("exact", "boot"),
  na.rm = FALSE,
  ...
)
```

Arguments

x	a (non-empty) numeric vector of data values.
conf.level	confidence level of the interval
sides	a character string specifying the side of the confidence interval, must be one of "two.sided" (default), "left" or "right". You can specify just the initial letter. "left" would be analogue to a hypothesis of "greater" in a t.test.
method	defining the type of interval that should be calculated (one out of "exact", "boot"). Default is "exact". See Details.
na.rm	logical. Should missing values be removed? Defaults to FALSE.
...	the dots are passed on to boot.ci. In particular, the type of bootstrap confidence interval can be defined via this. The defaults are R=999 and type="perc".

Details

The "exact" method is the way SAS is said to calculate the confidence interval. This is also implemented in SignTest. The boot confidence interval type is calculated by means of boot.ci with default type "perc".
Use sapply, resp.apply, to get the confidence intervals from a data.frame or from a matrix.

Value

a numeric vector with 3 elements:

median	median
lwr.ci	lower bound of the confidence interval
upr.ci	upper bound of the confidence interval

Author(s)

Andri Signorell andri@signorell.net

See Also

wilcox.test, MeanCI, median, HodgesLehmann

Examples

```
MedianCI(d.pizza$price, na.rm=TRUE)
MedianCI(d.pizza$price, conf.level=0.99, na.rm=TRUE)

t(round(sapply(d.pizza[,c("delivery_min","temperature","price")], MedianCI, na.rm=TRUE), 3))

MedianCI(d.pizza$price, na.rm=TRUE, method="exact")
MedianCI(d.pizza$price, na.rm=TRUE, method="boot")


x <- runif(100)
```

```
set.seed(448)
MedianCI(x, method="boot")

# ... the same as
set.seed(448)
MedianCI(x, method="boot", type="bca")

MedianCI(x, method="boot", type="basic")
MedianCI(x, method="boot", type="perc")
MedianCI(x, method="boot", type="norm", R=499)
# not supported:
MedianCI(x, method="boot", type="stud")

MedianCI(x, method="boot", sides="right")
```

---

mexp *Mean and Variance of the Exponential Distribution*

Description

Formula: $\mu = \frac{1}{\lambda}$ $\mathrm{Var}(X) = \frac{1}{\lambda^2}$

Usage

```
mexp(rate)
```

Arguments

rate	Rate parameter (1 / mean)

Value

List with mean and variance

See Also

dexp

Examples

```
mexp(rate = 0.5)
```

mf *Mean and Variance of the F Distribution*

Description

Formula: $\mu = \frac{df[2]}{df[2]-2} \quad (df[2] > 2)\ \mathrm{Var}(X) = \frac{2df[2]^2(df[1]+df[2]-2)}{df[1](df[2]-2)^2(df[2]-4)} \quad (df[2] > 4)$

Usage

```
mf(df1, df2)
```

Arguments

df1	Numerator degrees of freedom
df2	Denominator degrees of freedom

Value

List with mean and variance

See Also

df

Examples

```
mf(df1 = 5, df2 = 10)
```

mgamma *Mean and Variance of the Gamma Distribution*

Description

Formula: $\mu = \frac{\mathrm{shape}}{\mathrm{rate}}\ \mathrm{Var}(X) = \frac{\mathrm{shape}}{\mathrm{rate}^2}$

Usage

```
mgamma(shape, rate)
```

Arguments

shape	Shape parameter
rate	Rate parameter

Value

List with mean and variance

See Also

dgamma

Examples

```
mgamma(shape = 2, rate = 0.5)
```

---

mgeom *Mean and Variance of the Geometric Distribution*

Description

Formula: $\mu = \frac{1-p}{p}$ $\mathrm{Var}(X) = \frac{1-p}{p^2}$

Usage

```
mgeom(prob)
```

Arguments

prob	Probability of success

Value

List with mean and variance

See Also

dgeom

Examples

```
mgeom(prob = 0.2)
```

Mgsub *Multiple Gsub*

Description

Performs multiple substitions in (a) string(s).

Usage

```
Mgsub(pattern, replacement, x, ...)
```

Arguments

pattern	character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Coerced by as.character to a character string if possible.
replacement	a replacement for matched pattern as in sub and gsub. See there for more information.
x	a character vector where matches are sought, or an object which can be coerced by as.character to a character vector. Long vectors are supported.
...	all dots are passed on to gsub.

Value

a character vector of the same length and with the same attributes as x (after possible coercion to character).

Author(s)

Andri Signorell <andri@signorell.net>

See Also

gsub

Examples

```
x <- c("ABC", "BCD", "CDE")
Mgsub(pattern=c("B", "C"), replacement=c("X","Y"), x)
```

`MHChisqTest` *Mantel-Haenszel Chi-Square Test*

Description

The Mantel-Haenszel chi-square statistic tests the alternative hypothesis that there is a linear association between the row variable and the column variable. Both variables must lie on an ordinal scale.

Usage

```
MHChisqTest(x, srow = 1:nrow(x), scol = 1:ncol(x))
```

Arguments

`x`	a frequency table or a matrix.
`srow`	scores for the row variable, defaults to 1:nrow.
`scol`	scores for the columnn variable, defaults to 1:ncol.

Details

The statistic is computed as $Q_{MH} = (n-1) \cdot r^2$, where r^2 is the Pearson correlation between the row variable and the column variable. The Mantel-Haenszel chi-square statistic use the scores specified by srow and scol. Under the null hypothesis of no association, Q_{MH} has an asymptotic chi-square distribution with one degree of freedom.

Value

A list with class `"htest"` containing the following components:

`statistic`	the value the Mantel-Haenszel chi-squared test statistic.
`parameter`	the degrees of freedom of the approximate chi-squared distribution of the test statistic.
`p.value`	the p-value for the test.
`method`	a character string indicating the type of test performed.
`data.name`	a character string giving the name(s) of the data.

Author(s)

Andri Signorell <andri@signorell.net>

References

Agresti, A. (2002) *Categorical Data Analysis*. John Wiley & Sons, pp 86 ff.

See Also

chisq.test, for calculating correlation of a table: corr

Examples

```
## A r x c table  Agresti (2002, p. 57) Job Satisfaction
Job <- matrix(c(1,2,1,0, 3,3,6,1, 10,10,14,9, 6,7,12,11), 4, 4,
              dimnames = list(income = c("< 15k", "15-25k", "25-40k", "> 40k"),
                              satisfaction = c("VeryD", "LittleD", "ModerateS", "VeryS"))
       )

MHChisqTest(Job, srow=c(7.5,20,32.5,60))
```

---

mhyper *Mean and Variance of the Hypergeometric Distribution*

Description

Formula: $\mu = k \cdot \frac{m}{m+n}$ $\mathrm{Var}(X) = k \cdot \frac{m}{m+n} \cdot \frac{n}{m+n} \cdot \frac{m+n-k}{m+n-1}$

Usage

```
mhyper(m, n, k)
```

Arguments

m	Number of successes in the population
n	Number of failures in the population
k	Sample size

Value

List with mean and variance

See Also

dhyper

Examples

```
mhyper(m = 20, n = 30, k = 10)
```

`Midx` *Find the Midpoints of a Numeric Vector*

Description

Calculate the midpoints of a sequence of numbers. This is e.g. useful for labelling stacked barplots.

Usage

```
Midx(x, incl.zero = FALSE, cumulate = FALSE)
```

Arguments

`x`	the numeric vector
`incl.zero`	should zero be appended to x before proceeding? If `TRUE` the first value will be one half of the first value of x. Default is FALSE.
`cumulate`	should the result be calculated as cumulative sum? Default is FALSE.

Value

numeric vector with the calculated midpoins

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`MoveAvg`

Examples

```
x <- c(1, 3, 6, 7)

Midx(x)
Midx(x, incl.zero = TRUE)
Midx(x, incl.zero = TRUE, cumulate = TRUE)

# an alternative to
head(MoveAvg(c(0, x), order = 2, align = "l"), n = -1)

tab <- matrix(c(401,216,221,254,259,169), nrow=2, byrow=TRUE)
b <- barplot(tab, beside = FALSE, horiz=TRUE)

x <- t(apply(tab, 2, Midx, incl.zero=TRUE, cumulate=TRUE))
text(tab, x=x, y=b, col="red")
```

MixColor *Compute the Convex Combination of Two Colors*

Description

This function can be used to compute the result of color mixing (it assumes additive mixing).

Usage

```
MixColor(col1, col2, amount1 = 0.5)
```

Arguments

col1	the first color.
col2	the second color.
amount1	the amount of color1. The amount of color2 results in (1-amount1).

Value

The mixed color as hexstring

Author(s)

Andri Signorell <andri@signorell.net>

See Also

colorRamp, rgb

Examples

```
# a mix between red and yellow with rates 3:7
MixColor("red", "yellow", 0.3)
```

mlnorm *Mean and Variance of the Log-Normal Distribution*

Description

Formula: $\mu = \exp(\mu_{log} + 0.5 \cdot \sigma_{log}^2)$ $\mathrm{Var}(X) = (\exp(\sigma_{log}^2) - 1) \cdot \exp(2\mu_{log} + \sigma_{log}^2)$

Usage

```
mlnorm(meanlog, sdlog)
```

Arguments

meanlog	Mean on the log scale
sdlog	Standard deviation on the log scale

Value

List with mean and variance

See Also

dlnorm

Examples

```
mlnorm(meanlog = 0, sdlog = 1)
```

---

mnbinom *Mean and Variance of the Negative Binomial Distribution*

Description

Formula: $\mu = r \cdot \frac{1-p}{p}$ $\mathrm{Var}(X) = r \cdot \frac{1-p}{p^2}$

Usage

```
mnbinom(size, prob)
```

Arguments

size	Number of successes
prob	Probability of success

Value

List with mean and variance

See Also

dnbinom

Examples

```
mnbinom(size = 5, prob = 0.4)
```

mnorm *Mean and Variance of the Normal Distribution*

Description

Formula: $\mu = \mu \operatorname{Var}(X) = \sigma^2$

Usage

```
mnorm(mean, sd)
```

Arguments

mean	Mean of the distribution
sd	Standard deviation

Value

List with mean and variance

See Also

dnorm

Examples

```
mnorm(mean = 0, sd = 1)
```

Mode *Mode, Most Frequent Value(s)*

Description

Calculate the mode, the most frequent value, of a numeric or character vector x.

Usage

```
Mode(x, na.rm = FALSE)
```

Arguments

x	a (non-empty) numeric vector of data values.
na.rm	logical. Should missing values be removed? Defaults to FALSE.

Details

The mode is usually useful for qualitative data, sometimes still for an integer vector. For numerical vectors, it is not so much the central tendency property of the mode that is interesting as the information about conspicuous accumulation points, which sometimes can indicate data errors. In `Desc()` it is integrated in the numeric description to draw the analyst's attention to strikingly high frequencies of a single value as soon as they exceed a certain treshold. (In a numeric vector we would in general rather expect low numbers of tied values, or we should be aware of the process properties that generates them.)

The handling of `NA` values follows the standards of the package. As soon as a single `NA` value occurs, `NA` is returned as result. This approach can sometimes be conservative when calculating the mode. The mode could be determined unambiguously in cases where the number of missing values is small enough that - regardless of what value they have - they cannot alter the sample mode. The modal frequency could then be determined within a lower and upper range. In the example of `x=c(1,1,1,1,2,2,NA)` we know that the mode of x is 1 regardless of what the true value is for the one missing value; and we know that the modal frequency must be between 4 and 5. However this is not implemented in the function and further considerations in this direction are left to the user here.

The mode is elsewhere often calculated in a crude and wasteful way by tabulating the frequency for all elements of the vector and returning the most frequent one. This function uses a sophisticated data structure in C++ and is limited to determining the most frequent element only. Therefore it is orders of magnitude faster than other implementations, especially for large numeric vectors with large numbers of distinct values.

You might furthermore consider using `density(x)$x[which.max(density(x)$y)]` for quantitative data or alternatively use `hist()`.
Another interesting idea for a more robust estimation of the mode:

```
peak <- optimize(function(x, model) predict(model, data.frame(x = x)),
                 c(min(x), max(x)),
                 maximum = TRUE,
                 model = y.loess)

points(peak$maximum, peak$objective, pch=FILLED.CIRCLE <- 19)
```

Value

The most frequent value as number or character, depending of `class(x)`. If there is more than one, all are returned in a vector.
The modal frequency is attached as attribute named `"freq"`.

Author(s)

Andri Signorell <andri@signorell.net>, great Rcpp part by Joseph Wood and Ralf Stubner

References

https://stackoverflow.com/questions/55212746/rcpp-fast-statistical-mode-function-with-vector-input-of-any-type/ https://stackoverflow.com/a/55213471/8416610

See Also

Mean, Median

Examples

```
# normal mode
Mode(c(0:5, 5))

Mode(5)
Mode(NA)
Mode(c(NA, NA))
Mode(c(NA, 0:5))
Mode(c(NA, 0:5), na.rm=TRUE)
Mode(c(NA, 0:5, 5), na.rm=TRUE)

# returns all encountered modes, if several exist
Mode(c(0:5, 4, 5, 6))

Mode(d.pizza$driver)
Mode(d.pizza$driver, na.rm=TRUE)
Mode(as.character(d.pizza$driver), na.rm=TRUE)

# use sapply for evaluating data.frames (resp. apply for matrices)
sapply(d.pizza[,c("driver", "temperature", "date")], Mode, na.rm=TRUE)
```

MosesTest *Moses Test of Extreme Reactions*

Description

Perform Moses test of extreme reactions, which is a distribution-free non-parametric test for the difference between two independent groups in the extremity of scores (in both directions) that the groups contain. Scores from both groups are pooled and converted to ranks, and the test statistic is the span of scores (the range plus 1) in one of the groups chosen arbitrarily. An exact probability is computed for the span and then recomputed after dropping a specified number of extreme scores from each end of its range. The exact one-tailed probability is calculated.

Usage

```
MosesTest(x, ...)

## Default S3 method:
MosesTest(x, y, extreme = NULL, ...)

## S3 method for class 'formula'
MosesTest(formula, data, subset, na.action, ...)
```

Arguments

x	numeric vector of data values. x will be treated as control group. Non-finite (e.g. infinite or missing) values will be omitted.
y	numeric vector of data values. y will be treated as experiment group. Non-finite (e.g. infinite or missing) values will be omitted.
formula	a formula of the form lhs ~ rhs where lhs gives the data values and rhs the corresponding groups.
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
extreme	integer, defines the number of extreme values to be dropped from the control group before calculating the span. Default (NULL) is the integer part of 0.05 * length(x) or 1, whichever is greater. If extreme is too large, it will be cut down to floor(length(x)-2)/2.
...	further arguments to be passed to or from methods.

Details

For two independent samples from a continuous field, this tests whether extreme values are equally likely in both populations or if they are more likely to occur in the population from which the sample with the larger range was drawn.

Note that the ranks are calculated in decreasing mode.

Value

A list with class "htest" containing the following components:

statistic	the value of the Moses Test statistic.
p.value	the p-value for the test.
method	the character string "Moses Test of Extreme Reactions".
data.name	a character string giving the name(s) of the data.

Author(s)

Andri Signorell <andri@signorell.net>

References

Moses, L.E. (1952) A Two-Sample Test, *Psychometrika*, 17, 239-247.

See Also

wilcox.test, ks.test

Examples

```
x <- c(0.80, 0.83, 1.89, 1.04, 1.45, 1.38, 1.91, 1.64, 0.73, 1.46)
y <- c(1.15, 0.88, 0.90, 0.74, 1.21)

MosesTest(x, y)


set.seed(1479)
x <- sample(1:20, 10, replace=TRUE)
y <- sample(5:25, 6, replace=TRUE)

MosesTest(x, y)
```

MoveAvg *Moving Average*

Description

Compute a simple moving average (running mean).

Usage

```
MoveAvg(x, order, align = c("center", "left", "right"),
        endrule = c("NA", "keep", "constant"))
```

Arguments

x	univariate time series.
order	order of moving average.
align	specifies whether result should be centered (default), left-aligned or right-aligned.
endrule	character string indicating how the values at the beginning and the end (of the data) should be treated.
	"keep" keeps the first and last k_2 values at both ends, where k_2 is the half-bandwidth k2 = k %/% 2, i.e., y[j] = x[j] for $j \in \{1, \ldots, k_2; n - k_2 + 1, \ldots, n\}$;
	"constant" fill the ends with first and last calculated value in output array (out[1:k2] = out[k2+1])
	"NA" the default, leaves the values to NA, as they are returned by filter.

Details

The implementation is using the function filter to calculate the moving average.

Value

Returns a vector of the same size and same class as x.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

There's a faster implementation of running mean in the package **caTools** `runmean()` and a slower one in **forecast** `ma()`. There's similar code in `Midx()`.

Examples

```
MoveAvg(AirPassengers, order=5)
```

---

mpois *Mean and Variance of the Poisson Distribution*

Description

Formula: $\mu = \lambda$ $\mathrm{Var}(X) = \lambda$

Usage

```
mpois(lambda)
```

Arguments

`lambda`	Rate parameter (mean)

Value

List with mean and variance

See Also

`dpois`

Examples

```
mpois(lambda = 4)
```

mt *Mean and Variance of the t-Distribution*

Description

Formula: $\mu = 0 \quad (df > 1)\ \mathrm{Var}(X) = \frac{df}{df-2} \quad (df > 2)$

Usage

```
mt(df)
```

Arguments

df	Degrees of freedom

Value

List with mean and variance

See Also

dt

Examples

```
mt(df = 5)
```

MultinomCI *Confidence Intervals for Multinomial Proportions*

Description

Confidence intervals for multinomial proportions are often approximated by single binomial confidence intervals, which might in practice often yield satisfying results, but is properly speaking not correct. This function calculates simultaneous confidence intervals for multinomial proportions either according to the methods of Sison and Glaz, Goodman, Wald, Wald with continuity correction or Wilson.

Usage

```
MultinomCI(x, conf.level = 0.95, sides = c("two.sided", "left", "right"),
           method = c("sisonglaz", "cplus1", "goodman", "wald", "waldcc",
                      "wilson", "qh", "fs"))
```

Arguments

x	A vector of positive integers representing the number of occurrences of each class. The total number of samples equals the sum of such elements.
conf.level	confidence level, defaults to 0.95.
sides	a character string specifying the side of the confidence interval, must be one of "two.sided" (default), "left" or "right". You can specify just the initial letter. "left" would be analogue to a hypothesis of "greater" in a t.test.
method	character string specifing which method to use; can be one out of "sisonglaz", "cplus1", "goodman", "wald", "waldcc", "wilson", "qh", "fs". Method can be abbreviated. See details. Defaults to "sisonglaz".

Details

Given a vector of observations with the number of samples falling in each class of a multinomial distribution, builds the simultaneous confidence intervals for the multinomial probabilities according to the method proposed by the mentioned authors. The R code for Sison and Glaz (1995) has been translated from thes SAS code written by May and Johnson (2000). See the references for the other methods (qh = Quesensberry-Hurst, fs = Fitzpatrick-Scott).
Some approaches for the confidence intervals can potentially yield negative results or values beyond 1. These would be reset such as not to exceed the range of [0, 1].

Value

A matrix with 3 columns:

est	estimate
lwr.ci	lower bound of the confidence interval
upr.ci	upper bound of the confidence interval

The number of rows correspond to the dimension of x.

Author(s)

Pablo J. Villacorta Iglesias <pjvi@decsai.ugr.es>
Department of Computer Science and Artificial Intelligence, University of Granada (Spain) (Sison-Glaz)

Andri Signorell <andri@signorell.net> (Goodman, Wald, Wilson, Fitzpatrick-Scott, Quesensberry-Hurst)

References

Fitzpatrick, S. and Scott, A. (1987). Quick simultaneous confidence interval for multinomial proportions. *Journal of American Statistical Association* 82(399): 875-878.

Glaz, J., Sison, C.P. (1999) Simultaneous confidence intervals for multinomial proportions. *Journal of Statistical Planning and Inference* 82:251-262.

Goodman, L. A. (1965) On Simultaneous Confidence Intervals for Multinomial Proportions *Technometrics*, 7, 247-254.

May, W.L., Johnson, W.D.(2000) Constructing two-sided simultaneous confidence intervals for multinomial proportions for small counts in a large number of cells. *Journal of Statistical Software* 5(6) . Paper and code available at https://www.jstatsoft.org/v05/i06.

Quesensberry, C.P. and Hurst, D.C. (1964). Large Sample Simultaneous Confidence Intervals for Multinational Proportions. *Technometrics*, 6: 191-195.

Sangeetha, U., Subbiah, M., Srinivasan, M. R. (2013) Mathematical Analysis of propensity of aberration on the methods for interval estimation of the multinomial proportions. *IOSR Journal of Mathematics*, e-ISSN: 2278-5728,p-ISSN: 2319-765X, Volume 7, Issue 4 (Jul. - Aug. 2013), PP 23-28

Sison, C.P and Glaz, J. (1995) Simultaneous confidence intervals and sample size determination for multinomial proportions. *Journal of the American Statistical Association*, 90:366-369.

Wald, A. Tests of statistical hypotheses concerning several parameters when the number of observations is large, *Trans. Am. Math. Soc.* 54 (1943) 426-482.

Wilson, E. B. Probable inference, the law of succession and statistical inference, *J.Am. Stat. Assoc.* 22 (1927) 209-212.

Examples

```
# Multinomial distribution with 3 classes, from which a sample of 79 elements
# were drawn: 23 of them belong to the first class, 12 to the
# second class and 44 to the third class. Punctual estimations
# of the probabilities from this sample would be 23/79, 12/79
# and 44/79 but we want to build 95% simultaneous confidence intervals
# for the true probabilities

MultinomCI(c(23, 12, 44), conf.level=0.95)

# single sided
MultinomCI(c(23, 12, 44), conf.level=0.95, sides="left")
MultinomCI(c(23, 12, 44), conf.level=0.95, sides="right")


x <- c(35, 74, 22, 69)

MultinomCI(x, method="goodman")
MultinomCI(x, method="sisonglaz")
MultinomCI(x, method="cplus1")
MultinomCI(x, method="wald")
MultinomCI(x, method="waldcc")
MultinomCI(x, method="wilson")

# compare to
BinomCI(x, n=sum(x))

# example in Goodman (1965)
MultinomCI(x=c(91, 49, 37, 43), conf.level=0.95, method="goodman")

# example from Sison, Glaz (1999) in Sangeetha (2013) - Table 2
#
#    Wald          Wald_CC       Wilson        Quesnberry-Hurst
```

```
#    LL     UL     LL     UL     LL     UL     LL     UL
# 1  0.090 0.149  0.089  0.150  0.094 0.153  0.076  0.183
# 2  0.121 0.187  0.120  0.188  0.124 0.190  0.104  0.222
# 3  0.123 0.189  0.122  0.190  0.126 0.192  0.106  0.225
# 4  0.096 0.156  0.095  0.158  0.099 0.160  0.081  0.191
# 5  0.102 0.164  0.101  0.165  0.105 0.167  0.087  0.198
# 6  0.151 0.222  0.150  0.223  0.154 0.224  0.131  0.258
# 7  0.094 0.154  0.093  0.155  0.097 0.157  0.080  0.188

#    Goodman        Fitzpatrick-Scott  Sison-Glaz
#    LL     UL      LL     UL          LL    UL
# 1  0.085 0.166   0.075  0.165       0.079 0.164
# 2  0.115 0.204   0.109  0.200       0.114 0.199
# 3  0.116 0.207   0.111  0.202       0.116 0.201
# 4  0.091 0.173   0.081  0.172       0.086 0.171
# 5  0.096 0.181   0.087  0.178       0.092 0.177
# 6  0.143 0.239   0.141  0.232       0.146 0.231
# 7  0.089 0.171   0.079  0.170       0.084 0.169

x <- c(56, 72, 73, 59, 62, 87, 58)
do.call(cbind, lapply(c("wald", "waldcc", "wilson",
                        "qh", "goodman", "fs", "sisonglaz"),
                      function(m) round(MultinomCI(x, method=m)[,-1], 3)))
```

`MultMerge` *Merge Multiple Data Frames*

Description

Merge multiple data frames by row names, or do other versions of database join operations.

Usage

```
MultMerge(..., all.x = TRUE, all.y = TRUE, by = NULL)
```

Arguments

`...`	data frames to be coerced to one.
`all.x`	logical; if `TRUE`, then extra rows will be added to the output, one for each row in x that has no matching row in y. These rows will have `NA`s in those columns that are usually filled with values from y. The default is `FALSE`, so that only rows with data from both x and y are included in the output.
`all.y`	logical; analogous to `all.x`.
`by`	column used for merging, if this is not defined rownames will be used by default. The column must be included in all the provided data frames.

Value

A data frame. The rows are sorted according to the appearance of previously unobserved rownames. So the rownames appearing in the first data frame are first, then the rownames in the second data frame, which have no corespondence in the first data frame and so on. The columns are the remaining columns in x1 and then those in x2 and then those in x3. The result has the row names resulting from the merge.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

merge

Examples

```
x1 <- SetNames(data.frame(v=letters[1:6], w=1:6),
               rownames=c("A", "B", "C", "D", "E", "F"))
x2 <- SetNames(data.frame(v=letters[1:3], ww=11:13),
               rownames=c("B", "C", "D"))
x3 <- SetNames(data.frame(v=letters[12:16], wwww=22:26),
               rownames=c("A", "C", "E", "G", "J"))

# default is "merge by rownames"
MultMerge(x1, x2, x3)
# ... which does not really make sense here

# merge by column v
MultMerge(x1, x2, x3, by="v")
```

NALevel *Replace NAs in a Factor by a Given Level*

Description

In order to replace the NAs in a factor an additional level has to be defined first. This function does this and replaces the NAs by the given level.

Usage

```
NALevel(x, level)
```

Arguments

x	a vector which will be turned into a factor.
level	the name for the new level

Value

the vector x with the NAs replaced by level

Author(s)

Andri Signorell <andri@signorell.net>

See Also

factor, levels

Examples

```
x <- c(LETTERS[1:5], NA)
table(NALevel(x, "something else"))
```

NemenyiTest *Nemenyi Test*

Description

Performs Nemenyi's test of multiple comparisons.

Usage

```
NemenyiTest(x, ...)

## Default S3 method:
NemenyiTest(x, g, dist = c("tukey", "chisq"), out.list = TRUE, ...)

## S3 method for class 'formula'
NemenyiTest(formula, data, subset, na.action, ...)
```

Arguments

x	a numeric vector of data values, or a list of numeric data vectors.
g	a vector or factor object giving the group for the corresponding elements of x. Ignored if x is a list.
dist	the distribution used for the test. Can be tukey (default) or chisq.
out.list	logical, defining if the output should be organized in listform.
formula	a formula of the form lhs ~ rhs where lhs gives the data values and rhs the corresponding groups.
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).

subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
...	further arguments to be passed to or from methods.

Details

Nemenyi proposed a test based on rank sums and the application of the family-wise error method to control Type I error inflation, if multiple comparisons are done. The Tukey and Kramer approach uses mean rank sums and can be employed for equally as well as unequally sized samples without ties.

Value

A list of class htest, containing the following components:

statistic	Nemenyi test
p.value	the p-value for the test
null.value	is the value of the median specified by the null hypothesis. This equals the input argument mu.
alternative	a character string describing the alternative hypothesis.
method	the type of test applied
data.name	a character string giving the names of the data.

Author(s)

Andri Signorell <andri@signorell.net>

References

Nemenyi, P. B. (1963) *Distribution-Free Multiple Comparisons* New York, State University of New York, Downstate Medical Center

Hollander, M., Wolfe, D.A. (1999) *Nonparametric Statistical Methods* New York, Wiley, pp. 787

Friedman, M. (1937) The use of ranks to avoid the assumption of normality implicit in the analysis of variance *Journal of the American Statistical Association*, 32:675-701

Friedman, M. (1940) A comparison of alternative tests of significance for the problem of m rankings *Annals of Mathematical Statistics*, 11:86-92

See Also

DunnTest, ConoverTest

Examples

```
## Hollander & Wolfe (1973), 116.
## Mucociliary efficiency from the rate of removal of dust in normal
##  subjects, subjects with obstructive airway disease, and subjects
##  with asbestosis.
x <- c(2.9, 3.0, 2.5, 2.6, 3.2) # normal subjects
y <- c(3.8, 2.7, 4.0, 2.4)      # with obstructive airway disease
z <- c(2.8, 3.4, 3.7, 2.2, 2.0) # with asbestosis

NemenyiTest(list(x, y, z))

## Equivalently,
x <- c(x, y, z)
g <- factor(rep(1:3, c(5, 4, 5)),
            labels = c("Normal subjects",
                       "Subjects with obstructive airway disease",
                       "Subjects with asbestosis"))

NemenyiTest(x, g)

## Formula interface.
boxplot(Ozone ~ Month, data = airquality)
NemenyiTest(Ozone ~ Month, data = airquality)

# Hedderich & Sachs, 2012, p. 555
d.frm <- data.frame(x=c(28,30,33,35,38,41, 36,39,40,43,45,50, 44,45,47,49,53,54),
                    g=c(rep(LETTERS[1:3], each=6)), stringsAsFactors=TRUE)

NemenyiTest(x~g, d.frm)
```

Nf *As Numeric Factor*

Description

Encode a vector x to a factor and then to a numeric value. It's a simple shortcut for `as.numeric(factor(x, ...))`

Usage

```
Nf(x, ...)
```

Arguments

x	a vector of data, usually taking a small number of distinct values.
...	the dots are passed on to factor

Value

numeric vector

Author(s)

Andri Signorell <andri@signorell.net>

See Also

N

Examples

```
x <- LETTERS[10:15]
Nf(x)

# same as ..
as.numeric(factor(x))
```

NPV *Short Selection of Financial Mathematical Functions*

Description

Calculate the one period returns, the net present value (NPV()), the internal rate of return (IRR()) of a sequence of payments. NPVFixBond() returns the netpresent value for a fixed-rate bond, YTM() the yield to maturity for a bond.

Usage

```
OPR(K, D = NULL, log = FALSE)
NPV(i, cf, t = seq(along = cf) - 1)
IRR(cf, t = seq(along = cf) - 1, interval = c(-1.5, 1.5), ...)

NPVFixBond(i, Co, RV, n)
YTM(Co, PP, RV, n)
```

Arguments

i	the interest rate
cf	numeric vector with the payments
t	periods
K	the capital at time t
D	dividend at time t
log	logical, determining if the simple returns (default) or log returns are to be calculated.
interval	a vector containing the end-points of the interval to be searched for the root in the function IRR.
Co	coupon payments of a fixed-rate bond

PP	purchase price for a fixed-rate bond
RV	redemption value
n	the term of the bond, total number of periods
...	the dots are passed to the UnirootAll function in IRR

Details

The one period returns are calculated as

$$r_t = \frac{D_t + K_t - K_t - 1}{K_t - 1}$$

Value

a numeric value

Author(s)

Andri Signorell <andri@signorell.net>

See Also

Gmean

Examples

```
# one root
IRR(cf <- c(-900, -250+450-90, 460-100, 500-120, 550-140))
# several IRR solutions
IRR(cf = c(-100, 500, -600))
# no solution
IRR(cf = c(-100, 400, -600))
# negative and huge solution
IRR(cf = c(-100, 1000, -600), interval = c(-1.5, 1000))
```

NZ *Non Zero Elements*

Description

Return a vector with all zero elements removed.

Usage

```
NZ(x)
```

Arguments

x	numeric vector

Value

numerich vector

Author(s)

Andri Signorell <andri@signorell.net>

Examples

```
x <- c(1,2,0,3)
NZ(x)
```

`OddsRatio` *Odds Ratio Estimation and Confidence Intervals*

Description

Calculates odds ratio by unconditional maximum likelihood estimation (`wald`), conditional maximum likelihood estimation (`mle`) or median-unbiased estimation (`midp`). Confidence intervals are calculated using normal approximation (`wald`) and exact methods (`midp`, `mle`).

Usage

```
OddsRatio(x, conf.level = NULL, ...)

## S3 method for class 'glm'
OddsRatio(x, conf.level = NULL, digits = 3, use.profile = FALSE, ...)

## S3 method for class 'multinom'
OddsRatio(x, conf.level = NULL, digits = 3, ...)

## S3 method for class 'zeroinfl'
OddsRatio(x, conf.level = NULL, digits = 3, ...)

## Default S3 method:
OddsRatio(x, conf.level = NULL, y = NULL, method = c("wald", "mle", "midp"),
          interval = c(0, 1000), ...)
```

Arguments

`x`	a vector or a 2×2 numeric matrix, resp. table.
`y`	NULL (default) or a vector with compatible dimensions to `x`. If y is provided, `table(x, y, ...)` will be calculated.
`digits`	the number of fixed digits to be used for printing the odds ratios.
`method`	method for calculating odds ratio and confidence intervals. Can be one out of "`wald`", "`mle`", "`midp`". Default is "`wald`" (not because it is the best, but because it is the most commonly used.)

conf.level	confidence level. Default is NA for tables and numeric vectors, meaning no confidence intervals will be reported. 0.95 is used as default for models.
interval	interval for the function uniroot that finds the odds ratio median-unbiased estimate and midp exact confidence interval.
use.profile	logical. Defines if profile approach should be used, which normally is a good choice. Calculating profile can however take ages for large datasets and not be necessary there. So we can fallback to normal confidence intervals.
...	further arguments are passed to the function table, allowing i.e. to set useNA. This refers only to the vector interface.

Details

If a 2×2 table is provided the following table structure is preferred:

```
                disease=1   disease=0
exposed=1          n11         n10
exposed=0          n01         n00
```

however, for odds ratios the following table is equivalent:

```
                disease=0   disease=1
exposed=0 (ref)    n00         n01
exposed=1          n10         n11
```

If the table to be provided to this function is not in the preferred form, the function Rev() can be used to "reverse" the table rows, resp. -columns. Reversing columns or rows (but not both) will lead to the inverse of the odds ratio.

In case of zero entries, 0.5 will be added to the table.

Value

a single numeric value if conf.level is set to NA
a numeric vector with 3 elements for estimate, lower and upper confidence interval if conf.level is provided

Author(s)

Andri Signorell <andri@signorell.net>, strongly based on code from Tomas Aragon, <aragon@berkeley.edu>

References

Kenneth J. Rothman and Sander Greenland (1998): *Modern Epidemiology*, Lippincott-Raven Publishers

Kenneth J. Rothman (2002): *Epidemiology: An Introduction*, Oxford University Press

Nicolas P. Jewell (2004): *Statistics for Epidemiology*, 1st Edition, 2004, Chapman & Hall, pp. 73-81

Agresti, Alan (2013) *Categorical Data Analysis*. NY: John Wiley and Sons, Chapt. 3.1.1

See Also

RelRisk

Examples

```
# Case-control study assessing whether exposure to tap water
#   is associated with cryptosporidiosis among AIDS patients

tab <- matrix(c(2, 29, 35, 64, 12, 6), 3, 2, byrow=TRUE)
dimnames(tab) <- list("Tap water exposure" = c("Lowest", "Intermediate", "Highest"),
                      "Outcome" = c("Case", "Control"))
tab <- Rev(tab, margin=2)

OddsRatio(tab[1:2,])
OddsRatio(tab[c(1,3),])

OddsRatio(tab[1:2,], method="mle")
OddsRatio(tab[1:2,], method="midp")
OddsRatio(tab[1:2,], method="wald", conf.level=0.95)

# in case of zeros consider using glm for calculating OR
dp <- data.frame (a=c(20, 7, 0, 0), b=c(0, 0, 0, 12), t=c(1, 0, 1, 0))
fit <- glm(cbind(a, b) ~ t, data=dp, family=binomial)

exp(coef(fit))

# calculation of log oddsratios in a 2x2xk table
migraine <- xtabs(freq ~ .,
                  cbind(expand.grid(treatment=c("active","placebo"),
                                    response=c("better","same"),
                                    gender=c("female","male")),
                        freq=c(16,5,11,20,12,7,16,19))
)

log(apply(migraine, 3, OddsRatio))

# OddsRatio table for logistic regression models
r.glm <- glm(type ~ ., data=MASS::Pima.tr2, family=binomial)
OddsRatio(r.glm)

plot(OddsRatio(r.glm), xlim=c(0.5, 2), main="OddsRatio - glm", pch=NA,
     lblcolor=DescTools::hred, args.errbars=list(col=DescTools::horange, pch=21,
     col.pch=DescTools::hblue,
     bg.pch=DescTools::hyellow, cex.pch=1.5))
```

Order *Distributions of Order Statistics*

Description

Density function, distribution function and random generation for a selected Order statistic of a given number of independent variables from a specified distribution.

Usage

```
dOrder(x, densfun, distnfun, ..., distn, mlen = 1, j = 1,
      largest = TRUE, log = FALSE)
pOrder(q, distnfun, ..., distn, mlen = 1, j = 1, largest = TRUE,
      lower.tail = TRUE)
rOrder(n, quantfun, ..., distn, mlen = 1, j = 1, largest = TRUE)
```

Arguments

x, q	Vector of quantiles.
n	Number of observations.
densfun, distnfun, quantfun	Density, distribution and quantile function of the specified distribution. The density function must have a log argument (a simple wrapper can always be constructed to achieve this).
...	Parameters of the specified distribution.
distn	A character string, optionally specified as an alternative to densfun, distnfun and quantfun such that the density, distribution and quantile functions are formed upon the addition of the prefixes d, p and q respectively.
mlen	The number of independent variables.
j	The Order statistic, taken as the jth largest (default) or smallest of mlen, according to the value of largest.
largest	Logical; if TRUE (default) use the jth largest Order statistic, otherwise use the jth smallest.
log	Logical; if TRUE, the log density is returned.
lower.tail	Logical; if TRUE (default) probabilities are P[X <= x], otherwise P[X > x].

Value

dOrder gives the density function, pOrder gives the distribution function and qOrder gives the quantile function of a selected Order statistic from a sample of size mlen, from a specified distibution. rOrder generates random deviates.

Author(s)

Alec Stephenson <alec_stephenson@hotmail.com>

See Also

rExtrVal, rGenExtrVal

Examples

```
dOrder(2:4, dnorm, pnorm, mean = 0.5, sd = 1.2, mlen = 5, j = 2)
dOrder(2:4, distn = "norm", mean = 0.5, sd = 1.2, mlen = 5, j = 2)
dOrder(2:4, distn = "exp", mlen = 2, j = 2)
pOrder(2:4, distn = "exp", rate = 1.2, mlen = 2, j = 2)
rOrder(5, qgamma, shape = 1, mlen = 10, j = 2)
```

ORToRelRisk *Transform Odds Ratio to Relative Risk*

Description

The odds ratio is a common measure when comparing two groups in terms of an outcome that is either present or absent. As the odds ratio is in general poorly understood, odds ratios are often discussed in terms of risks, relying on the approximation, that odds ratio and relative risk are about the same when the outcome is rare. However the relative risk also depends on the risk of the baseline group and if the outcome is not rare there can be large differences between both measures and the odds ratio may substantially overestimate the relative risk. In fact, the same odds ratio could imply a very different relative risk for subgroups of the population with different baseline risks.

The present function transforms a given odds-ratio (OR) to the respective relative risk (RR) either for simple odds ratios but also for odds ratios resulting from a logistic model.

Usage

```
ORToRelRisk(...)

## S3 method for class 'OddsRatio'
ORToRelRisk(x, ... )
## Default S3 method:
ORToRelRisk(or, p0, ...)
```

Arguments

x	the odds ratios of a logistic model as returned by OddsRatio()
or	numeric vector, containing odds-ratios.
p0	numeric vector, incidence of the outcome of interest in the nonexposed group ("baseline risk").
...	further arguments, are not used here.

Details

The function transforms a given odds-ratio (or) to the respective relative risk (rr). It can also be used to transform the limits of confidence intervals.

The formula for converting an odds ratio to a relative risk is

$$rr = \frac{or}{1 - p_0 + p_0 \cdot or}$$

where p_0 is the baseline risk.

For transformation of odds ratios resulting from a logit model, we use the formula of Zhang and Yu (1998).

Value

relative risk.

Author(s)

Matthias Kohl <matthias.kohl@stamats.de>

References

Zhang, J. and Yu, K. F. (1998). What's the relative risk? A method of correcting the odds ratio in cohort studies of common outcomes. *JAMA*, **280**(19):1690-1691.

Grant, R. L. (2014) Converting an odds ratio to a range of plausible relative risks for better communication of research findings. BMJ 2014;348:f7450 doi: 10.1136/bmj.f7450

Examples

```
(heart <- as.table(matrix(c(11, 2, 4, 6), nrow=2,
                          dimnames = list(Exposure = c("High", "Low"),
                                          Response = c("Yes", "No")))))
RelRisk(heart)
# calculated as (11/15)/(2/8)

OddsRatio(heart)
# calculated as (11/4)/(2/6)

ORToRelRisk(OddsRatio(heart), p0 = 2/8)
# Relative risk = odds ratio / (1 - p0 + (p0 * odds ratio))
# where p0 is the baseline risk


## single OR to RR
ORToRelRisk(14.1, 0.05)

## OR and 95% confidence interval
ORToRelRisk(c(14.1, 7.8, 27.5), 0.05)

## Logistic OR and 95% confidence interval
logisticOR <- rbind(c(14.1, 7.8, 27.5),
                    c(8.7, 5.5, 14.3),
                    c(27.4, 17.2, 45.8),
                    c(4.5, 2.7, 7.8),
                    c(0.25, 0.17, 0.37),
                    c(0.09, 0.05, 0.14))
colnames(logisticOR) <- c("OR", "2.5%", "97.5%")
rownames(logisticOR) <- c("7.4", "4.2", "3.0", "2.0", "0.37", "0.14")
logisticOR
```

```
## p0
p0 <- c(0.05, 0.12, 0.32, 0.27, 0.40, 0.40)

## Compute corrected RR
## helper function
ORToRelRisk.mat <- function(or, p0){
  res <- matrix(NA, nrow = nrow(or), ncol = ncol(or))
  for(i in seq_len(nrow(or)))
    res[i,] <- ORToRelRisk(or[i,], p0[i])
  dimnames(res) <- dimnames(or)
  res
}
RR <- ORToRelRisk.mat(logisticOR, p0)
round(RR, 2)

## Results are not completely identical to Zhang and Yu (1998)
## what probably is caused by the fact that the logistic OR values
## provided in the table are rounded and not true values.
```

`Outlier` *Outlier*

Description

Return outliers following Tukey's boxplot and Hampel's median/mad definition.

Usage

```
Outlier(x, method = c("boxplot", "hampel"), value = TRUE,na.rm = FALSE)
```

Arguments

`x`	a (non-empty) numeric vector of data values.
`method`	the method to be used. So far Tukey's boxplot and Hampel's rule are implemented.
`value`	logical. If `FALSE`, a vector containing the (integer) indices of the outliers is returned, and if `TRUE` (default), a vector containing the matching elements themselves is returned.
`na.rm`	logical. Should missing values be removed? Defaults to `FALSE`.

Details

Outlier detection is a tricky problem and should be handled with care. We implement Tukey's boxplot rule as a rough idea of spotting extreme values.

Hampel considers values outside of median +/- 3 * (median absolute deviation) to be outliers.

Value

the values of x lying outside the whiskers in a boxplot
or the indices of them

Author(s)

Andri Signorell <andri@signorell.net>, performance improvement by Luis Gustavo Schuck

References

Hampel F. R. (1974) The influence curve and its role in robust estimation, *Journal of the American Statistical Association*, 69, 382-393

See Also

boxplot

Examples

```
Outlier(d.pizza$temperature, na.rm=TRUE)

# it's the same as the result from boxplot
sort(d.pizza$temperature[Outlier(d.pizza$temperature, value=FALSE, na.rm=TRUE)])
b <- boxplot(d.pizza$temperature, plot=FALSE)
sort(b$out)

# nice to find the corresponding rows
d.pizza[Outlier(d.pizza$temperature, value=FALSE, na.rm=TRUE), ]

# compare to Hampel's rule
Outlier(d.pizza$temperature, method="hampel", na.rm=TRUE)


# outliers for the each driver
tapply(d.pizza$temperature, d.pizza$driver, Outlier, na.rm=TRUE)

# the same as:
boxplot(temperature ~ driver, d.pizza)$out
```

PageTest *Exact Page Test for Ordered Alternatives*

Description

Performs a Page test for ordered alternatives using an exact algorithm by Stefan Wellek (1989) with unreplicated blocked data.

Usage

```
PageTest(y, ...)

## Default S3 method:
PageTest(y, groups, blocks, ...)

## S3 method for class 'formula'
PageTest(formula, data, subset, na.action, ...)
```

Arguments

y	either a numeric vector of data values, or a data matrix.
groups	a vector giving the group for the corresponding elements of y if this is a vector; ignored if y is a matrix. If not a factor object, it is coerced to one.
blocks	a vector giving the block for the corresponding elements of y if this is a vector; ignored if y is a matrix. If not a factor object, it is coerced to one.
formula	a formula of the form a ~ b | c, where a, b and c give the data values and corresponding groups and blocks, respectively.
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
...	further arguments to be passed to or from methods.

Details

PageTest can be used for analyzing unreplicated complete block designs (i.e., there is exactly one observation in y for each combination of levels of groups and blocks) where the normality assumption may be violated.

The null hypothesis is that apart from an effect of blocks, the location parameter of y is the same in each of the groups.
The implemented alternative is, that the location parameter will be monotonly greater along the groups,
$H_A : \theta_1 \leq \theta_2 \leq \theta_3$ ... (where at least one inequality is strict).
If the other direction is required, the order of the groups has to be reversed.

The Page test for ordered alternatives is slightly more powerful than the Friedman analysis of variance by ranks.

If y is a matrix, groups and blocks are obtained from the column and row indices, respectively. NA's are not allowed in groups or blocks; if y contains NA's, corresponding blocks are removed.

For small values of k (methods) or N (data objects), 'PageTest' will calculate the exact p-values. For 'k, N > 15, Inf', a normal approximation is returned. Only one of these values will be returned.

Value

A list with class "htest" containing the following components:

statistic	the L-statistic with names attribute "L".
p.value	the p-value of the test.
method	the character string "Page test for ordered alternatives".
data.name	a character string giving the names of the data.

Note

Special thanks to Prof. S. Wellek for porting old GAUSS code to R.

Author(s)

Stefan Wellek <stefan.wellek@zi-mannheim.de> (exact p-values), Andri Signorell <andri@signorell.net> (interface) (strongly based on R-Core code)

References

Page, E. (1963): Ordered hypotheses for multiple treatments: A significance test for linear ranks. *Journal of the American Statistical Association*, 58, 216-230.

Siegel, S. & Castellan, N. J. Jr. (1988): *Nonparametric statistics for the behavioral sciences*. Boston, MA: McGraw-Hill.

Wellek, S. (1989): Computing exact p-values in Page's nonparametric test against trend. *Biometrie und Informatik in Medizin und Biologie 20*, 163-170

See Also

friedman.test

Examples

```
 # Craig's data from Siegel & Castellan, p 186
 soa.mat <- matrix(c(.797,.873,.888,.923,.942,.956,
  .794,.772,.908,.982,.946,.913,
  .838,.801,.853,.951,.883,.837,
  .815,.801,.747,.859,.887,.902), nrow=4, byrow=TRUE)
 PageTest(soa.mat)


# Duller, pg. 236
pers <- matrix(c(
1, 72, 72, 71.5, 69, 70, 69.5, 68, 68, 67, 68,
2, 83, 81, 81, 82, 82.5, 81, 79, 80.5, 80, 81,
3, 95, 92, 91.5, 89, 89, 90.5, 89, 89, 88, 88,
4, 71, 72, 71, 70.5, 70, 71, 71, 70, 69.5, 69,
5, 79, 79, 78.5, 77, 77.5, 78, 77.5, 76, 76.5, 76,
6, 80, 78.5, 78, 77, 77.5, 77, 76, 76, 75.5, 75.5
), nrow=6, byrow=TRUE)
```

```
colnames(pers) <- c("person", paste("week",1:10))

# Alternative: week10 < week9 < week8 ...
PageTest(pers[, 11:2])


# Sachs, pg. 464

pers <- matrix(c(
  3,2,1,4,
  4,2,3,1,
  4,1,2,3,
  4,2,3,1,
  3,2,1,4,
  4,1,2,3,
  4,3,2,1,
  3,1,2,4,
  3,1,4,2),
  nrow=9, byrow=TRUE, dimnames=list(1:9, LETTERS[1:4]))

# Alternative: B < C < D < A
PageTest(pers[, c("B","C","D","A")])


# long shape and formula interface
plng <- data.frame(expand.grid(1:9, c("B","C","D","A")),
                   as.vector(pers[, c("B","C","D","A")]))
colnames(plng) <- c("block","group","x")

PageTest(plng$x, plng$group, plng$block)

PageTest(x ~ group | block, data = plng)



score <- matrix(c(
  3,4,6,9,
  4,3,7,8,
  3,4,4,6,
  5,6,8,9,
  4,4,9,9,
  6,7,11,10
  ), nrow=6, byrow=TRUE)

PageTest(score)
```

PairApply *Pairwise Calculations*

Description

Implements a logic to run pairwise calculations on the columns of a data.frame or a matrix.

Usage

```
PairApply(x, FUN = NULL, ..., symmetric = FALSE)
```

Arguments

x	a list, a data.frame or a matrix with columns to be processed pairwise.
FUN	a function to be calculated. It is assumed, that the first 2 arguments denominate x and y.
...	the dots are passed to FUN.
symmetric	logical. Does the function yield the same result for FUN(x, y) and FUN(y, x)? If TRUE just the lower triangular matrix is calculated and transposed. Default is FALSE.

Details

This code is based on the logic of `cor()` and extended for asymmetric functions.

Value

a matrix with the results of FUN.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`outer`, `CombPairs`, `pairwise.table`

Examples

```
PairApply(d.diamonds[,c("colour","clarity","cut","polish")], FUN = CramerV,
          symmetric=TRUE)

# user defined functions are ok as well
PairApply(d.diamonds[,c("clarity","cut","polish","symmetry")],
  FUN = function(x,y) wilcox.test(as.numeric(x), as.numeric(y))$p.value, symmetric=TRUE)

# asymetric measure
PairApply(d.diamonds[,c("colour", "clarity", "cut", "polish")],
  FUN = Lambda, direction = "row")

# ... compare to:
Lambda(x=d.diamonds$colour, y=d.diamonds$clarity, direction="row")
Lambda(x=d.diamonds$colour, y=d.diamonds$clarity, direction="column")
```

```
# the data.frame
dfrm <- d.diamonds[, c("colour","clarity","cut","polish")]
PairApply(dfrm, FUN = CramerV, symmetric=TRUE)

# the same as matrix (columnwise)
m <- as.matrix(dfrm)
PairApply(m, FUN = CramerV, symmetric=TRUE)

# ... and the list interface
lst <- as.list(dfrm)
PairApply(lst, FUN = CramerV, symmetric=TRUE)
```

`ParseFormula` *Parse a Formula and Create a Model Frame*

Description

Create a model frame for a formula object, by handling the left hand side the same way the right hand side is handled in model.frame. Especially variables separated by + are interpreted as separate variables.

Usage

```
ParseFormula(formula, data = parent.frame(), drop = TRUE)
```

Arguments

`formula`	an object of class "formula" (or one that can be coerced to that class): a symbolic description for the variables to be described.
`data`	an optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which lm is called.
`drop`	if `drop` is `TRUE`, unused factor levels are dropped from the result when creating interaction terms. The default is to drop all unused factor levels.

Details

This is used by `Desc.formula` for describing data by groups while remaining flexible for using `I(...)` constructions, functions or interaction terms.

Value

a list of 3 elements

`formula`	the formula which had to be parsed

lhs	a list of 3 elements: mf: data.frame, the model.frame of the left hand side of the formula mf.eval: data.frame, the evaluated model.frame of the left hand side of the formula vars: the names of the evaluated model.frame
rhs	a list of 3 elements: mf: data.frame, the model.frame of the right hand side of the formula mf.eval: data.frame, the evaluated model.frame of the right hand side of the formula vars: the names of the evaluated model.frame

Author(s)

Andri Signorell <andri@signorell.net>

See Also

The functions used to handle formulas: model.frame, terms, formula
Used in: Desc.formula

Examples

```
set.seed(17)
piz <- d.pizza[sample(nrow(d.pizza),10), c("temperature","price","driver","weekday")]

f1 <- formula(. ~ driver)
f2 <- formula(temperature ~ .)
f3 <- formula(temperature + price ~ .)
f4 <- formula(temperature ~ . - driver)
f5 <- formula(temperature + price ~ driver)
f6 <- formula(temperature + price ~ driver * weekday)
f7 <- formula(I(temperature^2) + sqrt(price) ~ driver + weekday)
f8 <- formula(temperature + price ~ 1)
f9 <- formula(temperature + price ~ driver * weekday - price)

ParseFormula(f1, data=piz)
ParseFormula(f2, data=piz)
ParseFormula(f3, data=piz)
ParseFormula(f4, data=piz)
ParseFormula(f5, data=piz)
ParseFormula(f6, data=piz)
ParseFormula(f7, data=piz)
ParseFormula(f8, data=piz)
```

ParseSASDatalines *Parse a SAS Dataline Command*

Description

A parser for simple SAS dataline command texts. A `data.frame` is being built with the column-names listed in the input section. The data object will be created in the given environment.

Usage

```
ParseSASDatalines(x, env = .GlobalEnv, overwrite = FALSE)
```

Arguments

`x`	the SAS text
`env`	environment in which the dataset should be created.
`overwrite`	logical. If set to TRUE, the function will silently overwrite a potentially existing object in `env` with the same name as declared in the SAS `DATA` section. If set to `FALSE` (default) an error will be raised if there already exists an object with the same name.

Details

The SAS function `DATA` is designed for quickly creating a dataset from scratch. The whole step normally consists out of the `DATA` part defining the name of the dataset, an `INPUT` line declaring the variables and a `DATALINES` command followed by the values.
The default delimiter used to separate the different variables is a space (thus each variable should be one word). The $ after the variable name indicates that the variable preceding contain character values and not numeric values. Without specific instructions, SAS assumes that variables are numeric. The function will fail, if it encounters a character in the place of an expected numeric value. Each new row in datalines will create a corresponding unique row in the dataset. Notice that a ; is not needed after every row, rather it is included at the end of the entire data step.

More complex command structures, i.e. other delimiters (dlm), in the `INPUT`-section are not (yet) supported.

Value

a data.frame

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`scan`

Examples

```
txt <- "
DATA asurvey;
INPUT id sex $ age inc r1 r2 r3 ;
DATALINES;
```

```
1   F  35 17  7 2 2
17  M  50 14  5 5 3
33  F  45  6  7 2 7
49  M  24 14  7 5 7
65  F  52  9  4 7 7
81  M  44 11  7 7 7
2   F  34 17  6 5 3
18  M  40 14  7 5 2
34  F  47  6  6 5 6
50  M  35 17  5 7 5
;
"

(d.frm <- ParseSASDatalines(txt))
```

`PasswordDlg` *Password Dialog*

Description

Brings up a tcltk dialog centered on the screen, designed for entering passwords while displaying only ****.

Usage

```
PasswordDlg(option_txt = NULL)
```

Arguments

`option_txt`	an optional text, if it is defined, there will be a checkbox added to the dialog with the label being set with `option_txt`.

Value

the entered password
the status of the optional checkbox will be returned as attribute: `attr(pw, "option")`

Author(s)

Markus Naepflin <markus@naepfl.in>

Examples

```
## Not run:
pw <- PasswordDlg()
pw
## End(Not run)
```

PDFManual *Get PDF Manual of a Package From CRAN*

Description

PDF versions of the manual are usually not included as vignettes in R packages. Still this format is convenient for reading and doing full text search.
This function creates the appropriate link to the pdf file on CRAN and opens the pdf manual in a browser window.

Usage

```
PDFManual(package)
```

Arguments

package	name of the package.

Author(s)

Andri Signorell <andri@signorell.net>

Examples

```
## Not run:
PDFManual(DescTools)

## End(Not run)
```

PearsonTest *Pearson Chi-Square Test for Normality*

Description

Performs the Pearson chi-square test for the composite hypothesis of normality.

Usage

```
PearsonTest(x, n.classes = ceiling(2 * (n^(2/5))), adjust = TRUE)
```

Arguments

x	a numeric vector of data values. Missing values are allowed.
n.classes	The number of classes. The default is due to Moore (1986).
adjust	logical; if TRUE (default), the p-value is computed from a chi-square distribution with n.classes-3 degrees of freedom, otherwise from a chi-square distribution with n.classes-1 degrees of freedom.

Details

The Pearson test statistic is $P = \sum(C_i - E_i)^2/E_i$, where C_i is the number of counted and E_i is the number of expected observations (under the hypothesis) in class i. The classes are build is such a way that they are equiprobable under the hypothesis of normality. The p-value is computed from a chi-square distribution with `n.classes`-3 degrees of freedom if `adjust` is `TRUE` and from a chi-square distribution with `n.classes`-1 degrees of freedom otherwise. In both cases this is not (!) the correct p-value, lying somewhere between the two, see also Moore (1986).

Value

A list of class `htest`, containing the following components:

`statistic`	the value of the Pearson chi-square statistic.
`p.value`	the p-value for the test.
`method`	the character string "Pearson chi-square normality test".
`data.name`	a character string giving the name(s) of the data.
`n.classes`	the number of classes used for the test.
`df`	the degress of freedom of the chi-square distribution used to compute the p-value.

Note

The Pearson chi-square test is usually not recommended for testing the composite hypothesis of normality due to its inferior power properties compared to other tests. It is common practice to compute the p-value from the chi-square distribution with `n.classes` - 3 degrees of freedom, in order to adjust for the additional estimation of two parameters. (For the simple hypothesis of normality (mean and variance known) the test statistic is asymptotically chi-square distributed with `n.classes` - 1 degrees of freedom.) This is, however, not correct as long as the parameters are estimated by `mean(x)` and `var(x)` (or `sd(x)`), as it is usually done, see Moore (1986) for details. Since the true p-value is somewhere between the two, it is suggested to run `PearsonTest` twice, with `adjust = TRUE` (default) and with `adjust = FALSE`. It is also suggested to slightly change the default number of classes, in order to see the effect on the p-value. Eventually, it is suggested not to rely upon the result of the test.

The function call `PearsonTest(x)` essentially produces the same result as the S-PLUS function call `chisq.gof((x-mean(x))/sqrt(var(x)), n.param.est=2)`.

Author(s)

Juergen Gross <gross@statistik.uni-dortmund.de>

References

Moore, D.S., (1986) Tests of the chi-squared type. In: D'Agostino, R.B. and Stephens, M.A., eds.: *Goodness-of-Fit Techniques*. Marcel Dekker, New York.

Thode Jr., H.C., (2002) *Testing for Normality*. Marcel Dekker, New York. Sec. 5.2

See Also

shapiro.test for performing the Shapiro-Wilk test for normality. AndersonDarlingTest, CramerVonMisesTest, LillieTest, ShapiroFranciaTest for performing further tests for normality. qqnorm for producing a normal quantile-quantile plot.

Examples

```
PearsonTest(rnorm(100, mean = 5, sd = 3))
PearsonTest(runif(100, min = 2, max = 4))
```

PercentRank *Percent Ranks*

Description

PercentRank() takes a vector x and returns the percentile that elements of x correspond to.

Usage

```
PercentRank(x)
```

Arguments

x	a numeric, complex, character or logical vector.

Value

A numeric vector of the same length as x with names copied from x (unless na.last = NA, when missing values are removed). The vector is of integer type unless x is a long vector.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

Rank, rank, factor, order, sort

Examples

```
(r1 <- rank(x1 <- c(3, 1, 4, 15, 92)))

x2 <- c(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5)
names(x2) <- letters[1:11]
(r2 <- rank(x2))        # ties are averaged

PercentRank(x2)
```

PercTable *Percentage Table*

Description

Prints a 2-way contingency table along with percentages, marginal, and conditional distributions. All the frequencies are nested into one single table.

Usage

```
## Default S3 method:
PercTable(x, y = NULL, ...)

## S3 method for class 'table'
PercTable(tab, row.vars = NULL, col.vars = NULL, justify = "right",
          freq = TRUE, rfrq = "100", expected = FALSE, residuals = FALSE,
          stdres = FALSE, margins = NULL, digits = NULL, ...)

## S3 method for class 'formula'
PercTable(formula, data, subset, na.action, ...)

## S3 method for class 'PercTable'
print(x, vsep = NULL, ...)

Margins(tab, ...)
```

Arguments

x, y	objects which can be interpreted as factors (including character strings). x and y will be tabulated via `table(x, y)`. If x is a matrix, it will be coerced to a table via as.table(x).
tab	a r x c-contingency table
row.vars	a vector of row variables (see Details).
col.vars	a vector of column variables (see Details). If this is left to `NULL` the table structure will be preserved.
justify	either `"left"` or `"right"` for defining the alignment of the table cells.
freq	boolean. Should absolute frequencies be included? Defaults to TRUE.
rfrq	a string with 3 characters, each of them being 1 or 0. The first position means total percentages, the second means row percentages and the third column percentages. "011" produces a table output with row and column percentages.
expected	the expected counts under the null hypothesis.
residuals	the Pearson residuals, (observed - expected) / sqrt(expected).
stdres	standardized residuals, (observed - expected) / sqrt(V), where V is the residual cell variance (for the case where x is a matrix, n * p * (1 - p) otherwise).

margins	a vector, consisting out of 1 and/or 2. Defines the margin sums to be included. 1 stands for row margins, 2 for column margins, c(1,2) for both. Default is NULL (none).
digits	integer. With how many digits shoud the relative frequencies be formatted? Default can be set by DescToolsOptions(digits=x).
formula	a formula of the form lhs ~ rhs where lhs will be tabled versus rhs (table(lhs, rhs)).
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
vsep	logical, defining if an empty row should be introduced between the table rows. Default is FALSE, if only a table with one single description (either frequencies or percents) should be returned and TRUE in any other case.
...	the dots are passed from PercTable.default() to PercTable.table() and from Margins to the function Freq.

Details

PercTable prints a 2-dimensional table. The absolute and relative frequencies are nested into one flat table by means of ftable. row.vars, resp. col.vars can be used to define the structure of the table. row.vars can either be the names of the dimensions (included percentages are named "idx") or numbers (1:3, where 1 is the first dimension of the table, 2 the second and 3 the percentages). Use Sort() if you want to have your table sorted by rows.

The style in which numbers are formatted is selected by Fmt() from the DescTools options. Absolute frequencies will use Fmt("abs") and Fmt("per") will do it for the percentages. The options can be changed with Fmt(abs=as.fmt(...)) which is basically a "fmt"-object containing any format information used in Format.

Margins() returns a list containing all the one dimensional margin tables of a n-dimensional table along the given dimensions. It uses margin.table() for all the dimensions and adds the appropriate percentages.

Value

Returns an object of class "ftable".

Author(s)

Andri Signorell <andri@signorell.net>

References

Agresti, Alan (2007) *Introduction to categorical data analysis*. NY: John Wiley and Sons, Section 2.4.5

See Also

Freq, table, ftable, prop.table, addmargins, DescToolsOptions, Fmt
There are similar functions in package **sfsmisc** printTable2 and package **vcd** table2d_summary, both lacking some of the flexibility we needed here.

Examples

```
tab <- table(driver=d.pizza$driver, area=d.pizza$area)

PercTable(tab=tab, col.vars=2)

PercTable(tab=tab, col.vars=2, margins=c(1,2))
PercTable(tab=tab, col.vars=2, margins=2)
PercTable(tab=tab, col.vars=2, margins=1)
PercTable(tab=tab, col.vars=2, margins=NULL)

PercTable(tab=tab, col.vars=2, rfrq="000")

# just the percentages without absolute values
PercTable(tab=tab, col.vars=2, rfrq="110", freq=FALSE)

# just the row percentages in percent format (pfmt = TRUE)
PercTable(tab, freq= FALSE, rfrq="010", pfmt=TRUE, digits=1)

# just the expected frequencies and the standard residuals
PercTable(tab=tab, rfrq="000", expected = TRUE, stdres = TRUE)


# rearrange output such that freq are inserted as columns instead of rows
PercTable(tab=tab, col.vars=c(3,2), rfrq="111")

# putting the areas in rows
PercTable(tab=tab, col.vars=c(3,1), rfrq="100", margins=c(1,2))

# formula interface with subset
PercTable(driver ~ area, data=d.pizza, subset=wine_delivered==0)

# sort the table by rows, order first column (Zurich), then third, then row.names (0)
PercTable(tab=Sort(tab, ord=c(1,3,0)))

# reverse the row variables, so that absolute frequencies and percents
# are not nested together
PercTable(tab, row.vars=c(3, 1))

# the vector interface
PercTable(x=d.pizza$driver, y=d.pizza$area)
PercTable(x=d.pizza$driver, y=d.pizza$area, margins=c(1,2), rfrq="000", useNA="ifany")

# one dimensional x falls back to the function Freq()
PercTable(x=d.pizza$driver)
```

```
# the margin tables
Margins(Titanic)
```

Permn *Number and Samples for Permutations or Combinations of a Set*

Description

Return the set of permutations for a given set of values. The values can be numeric values, characters or factors. `CombN` computes the number of combinations with and without replacement and order, whereas `CombSet` returns the value sets.

Usage

```
Permn(x, sort = FALSE)

CombN(n, m, repl = FALSE, ord = FALSE)
CombSet(x, m, repl = FALSE, ord = FALSE, as.list = FALSE)
```

Arguments

`x`	a vector of numeric values or characters. Characters need not be unique.
`n`	number of elements from which to choose.
`m`	number of elements to choose. For `CombSet` can `m` be a numeric vector too.
`repl`	logical. Should repetition of the same element be allowed? Defaults to FALSE
`ord`	logical. Does the order matter? Default is FALSE.
`sort`	logical, defining if the result set should be sorted. Default is FALSE.
`as.list`	logical, defining if the results should be returned in a flat list, say every sample is a single element of the resulting list. Default is FALSE.

Details

The vector x need not contain unique values. The permutations will automatically be filtered for unique sets, if the same element is given twice or more.

Value

a matrix with all possible permutations or combinations of the values in x for `Permn` and `CombSet` if m contains more than one element the result will be a list of matrices or a flat list if `as.list` is set to `TRUE`
an integer value for `CombN`

Author(s)

Friederich Leisch <Friedrich.Leisch@boku.ac.at>
Andri Signorell <andri@signorell.net> (CombSet, CombN)

See Also

combn, choose, factorial, CombPairs
vignette("Combinatorics")

Examples

```
Permn(letters[2:5])
Permn(2:5)

# containing the same element more than once
Permn(c("a", "b", "c", "a"))


# only combinations of 2, but in every possible order
x <- letters[1:4]
n <- length(x)
m <- 2

# the samples
CombSet(x, m, repl=TRUE, ord=FALSE)
CombSet(x, m, repl=TRUE, ord=TRUE)
CombSet(x, m, repl=FALSE, ord=TRUE)
CombSet(x, m, repl=FALSE, ord=FALSE)

# the number of the samples
CombN(n, m, repl=TRUE, ord=FALSE)
CombN(n, m, repl=TRUE, ord=TRUE)
CombN(n, m, repl=FALSE, ord=TRUE)
CombN(n, m, repl=FALSE, ord=FALSE)

# build all subsets of length 1, 3 and 5 and return a flat list
x <- letters[1:5]
CombSet(x=x, m=c(1, 3, 5), as.list=TRUE)
```

Phrase *Phrasing Results of t-Test*

Description

Formulating the results of a comparison of means is quite common. This function assembles a descriptive text about the results of a t-test, describing group sizes, means, p-values and confidence intervals.

Usage

```
Phrase(x, g, glabels = NULL, xname = NULL, unit = NULL, lang = "engl", na.rm = FALSE)
```

Arguments

x	a (non-empty) numeric vector of data values.
g	a vector or factor object giving the group for the corresponding elements of x. The number of levels must equal 2.
glabels	the labels of the two groups, if left to NULL, the levels will be used.
xname	the name of the variable to be used in the text.
unit	an optional unit for be appended to the numeric results.
lang	the language to be used. Only english (default) and german implemented (so far).
na.rm	logical, should NAs be omitted? Defaults to FALSE.

Value

a text

Author(s)

Andri Signorell <andri@signorell.net>

See Also

t.test

Examples

```
data("cats", package = "MASS")
cat(Phrase(cats$Bwt, cats$Sex, xname="weight", unit="grams", glabels=c("female", "male")))

# oder auf deutsch
cat(Phrase(cats$Bwt, cats$Sex, xname="Geburtsgewicht",
           glabels=c("weiblich", "maennlich"), lang="german"))
```

PlotACF *Combined Plot of a Time Series and Its ACF and PACF*

Description

Combined plot of a time Series and its autocorrelation and partial autocorrelation

Usage

```
PlotACF(series, lag.max = 10 * log10(length(series)), main = NULL, cex = NULL, ...)
PlotGACF(series, lag.max = 10 * log10(length(series)), type = "cor", ylab = NULL, ...)
```

Arguments

series	univariate time series.
lag.max	integer. Defines the number of lags to be displayed. The default is 10 * log10(length(series)).
main	an overall title for the plot
cex	numerical value giving the amount by which plotting text and symbols should be magnified relative to the default.
type	character string giving the type of acf to be computed. Allowed values are "cor" (the default), "cov" or "part" for autocorrelation, covariance or partial correlation.
ylab	a title for the y axis: see title.
...	the dots are passed to the plot command.

Details

PlotACF plots a combination of the time series and its autocorrelation and partial autocorrelation. PlotGACF is used as subfunction to produce the acf- and pacf-plots.

Author(s)

Markus Huerzeler (ETH Zurich), some minor modifications Andri Signorell <andri@signorell.net>

See Also

ts

Examples

```
PlotACF(AirPassengers)
```

PlotArea *Create an Area Plot*

Description

Produce a stacked area plot, or add polygons to an existing plot.

Usage

```
## Default S3 method:
PlotArea(x, y = NULL, prop = FALSE, add = FALSE, xlab = NULL,
         ylab = NULL, col = NULL, frame.plot = FALSE, ...)

## S3 method for class 'formula'
PlotArea(formula, data, subset, na.action, ...)
```

Arguments

x	numeric vector of x values, or if y=NULL a numeric vector of y values. Can also be a 1-dimensional table (x values in names, y values in array), matrix or 2-dimensional table (x values in row names and y values in columns), a data frame (x values in first column and y values in subsequent columns), or a time-series object of class ts/mts.
y	numeric vector of y values, or a matrix containing y values in columns.
prop	whether data should be plotted as proportions, so stacked areas equal 1.
add	whether polygons should be added to an existing plot.
xlab	label for x axis.
ylab	label for y axis.
col	fill color of polygon(s). The default is a vector of gray colors.
frame.plot	a logical indicating whether a box should be drawn around the plot.
formula	a formula, such as y ~ x or cbind(y1, y2) ~ x, specifying x and y values. A dot on the left-hand side, formula = . ~ x, means all variables except the one specified on the right-hand side.
data	a data frame (or list) from which the variables in formula should be taken.
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NA values. Defaults to getOption("na.action").
...	further arguments are passed to matplot and polygon.

Value

Matrix of cumulative sums that was used for plotting.

Author(s)

Arni Magnusson <thisisarni@gmail.com>

See Also

barplot, polygon, areaplot

Examples

```
# PlotArea with stapled areas
tab <- table( d.pizza$date, d.pizza$driver )
PlotArea(x=as.Date(rownames(tab)), y=tab, xaxt="n", xlab="Date", ylab="Pizzas delivered" )

# add x-axis and some text labels
xrng <- pretty(range(as.Date(rownames(tab))))
axis(side=1, at=xrng, labels=xrng)
text( x=min(d.pizza$date + .5, na.rm=TRUE), y=cumsum(tab[2,])-2.5, label=levels(d.pizza$driver),
  adj=c(0,0.5), col=TextContrastColor(gray.colors(7)))


# formula
PlotArea(Armed.Forces~Year, data=longley)
PlotArea(cbind(Armed.Forces,Unemployed)~Year, data=longley)

# add=TRUE
plot(1940:1970, 500*runif(31), ylim=c(0,500))
PlotArea(Armed.Forces~Year, data=longley, add=TRUE)

# matrix
PlotArea(WorldPhones)
PlotArea(WorldPhones, prop=TRUE, col=rainbow(10))

# table
PlotArea(table(d.pizza$weekday))
PlotArea(table(d.pizza$weekday, d.pizza$driver))

# ts/mts
PlotArea(austres)
PlotArea(Seatbelts[,c("drivers","front","rear")],
         ylab="Killed or seriously injured")
abline(v=1983+1/12, lty=3)
```

PlotBag *Bivariate Boxplot*

Description

`PlotBag()` creates a twodimensional boxplot called "bagplot" based on two numerical variables x and y. `plot.PlotBag()` is the plotting routine for a bagplot object. `compute.PlotBag()` contains the computation logic the object.

Usage

```
PlotBag(x, y, factor = 3, na.rm = FALSE, approx.limit = 300,
        show.outlier = TRUE, show.whiskers = TRUE,
        show.looppoints = TRUE, show.bagpoints = TRUE,
        show.loophull = TRUE, show.baghull = TRUE,
```

```
        create.plot = TRUE, add = FALSE, pch = 16, cex = 0.4,
        dkmethod = 2, precision = 1, verbose = FALSE,
        debug.plots = "no",   col.loophull = "#aaccff",
        col.looppoints = "#3355ff", col.baghull = "#7799ff",
        col.bagpoints = "#000088", transparency = FALSE, ...
)
PlotBagPairs(dm, trim = 0.0, main, numeric.only = TRUE,
             factor = 3, approx.limit = 300, pch = 16,
             cex = 0.8, precision = 1, col.loophull = "#aaccff",
             col.looppoints = "#3355ff", col.baghull = "#7799ff",
             col.bagpoints = "#000088", ...)

compute.bagplot(x, y, factor = 3, na.rm = FALSE, approx.limit = 300,
              dkmethod = 2, precision = 1, verbose = FALSE, debug.plots = "no" )

## S3 method for class 'bagplot'
plot(x, show.outlier = TRUE, show.whiskers = TRUE,
     show.looppoints = TRUE, show.bagpoints = TRUE,
     show.loophull = TRUE, show.baghull = TRUE, add = FALSE,
     pch = 16, cex = .4, verbose = FALSE, col.loophull = "#aaccff",
     col.looppoints = "#3355ff", col.baghull = "#7799ff",
     col.bagpoints = "#000088", transparency = FALSE,...)
```

Arguments

x	x values of a data set; in PlotBag: an object of class PlotBag computed by compute.PlotBag
y	y values of the data set
factor	factor defining the loop
na.rm	if TRUE 'NA' values are removed otherwise exchanged by median
approx.limit	if the number of data points exceeds approx.limit a sample is used to compute some of the quantities; default: 300
show.outlier	if TRUE outlier are shown
show.whiskers	if TRUE whiskers are shown
show.looppoints	if TRUE loop points are plottet
show.bagpoints	if TRUE bag points are plottet
show.loophull	if TRUE the loop is plotted
show.baghull	if TRUE the bag is plotted
create.plot	if FALSE no plot is created
add	if TRUE the bagplot is added to an existing plot
pch	sets the plotting character
cex	sets characters size

dkmethod	1 or 2, there are two method of approximating the bag, method 1 is very rough (only based on observations
precision	precision of approximation, default: 1
verbose	automatic commenting of calculations
debug.plots	if TRUE additional plots describing intermediate results are constructed
col.loophull	color of loop hull
col.looppoints	color of the points of the loop
col.baghull	color of bag hull
col.bagpoints	color of the points of the bag
transparency	see section details
dm	x
trim	x
main	x
numeric.only	x
...	additional graphical parameters

Details

A bagplot is a bivariate generalization of the well known boxplot. It has been proposed by Rousseeuw, Ruts, and Tukey. In the bivariate case the box of the boxplot changes to a convex polygon, the bag of bagplot. In the bag are 50 percent of all points. The fence separates points within the fence from points outside. It is computed by increasing the the bag. The loop is defined as the convex hull containing all points inside the fence. If all points are on a straight line you get a classical boxplot. `PlotBag()` plots bagplots that are very similar to the one described in Rousseeuw et al. Remarks: The two dimensional median is approximated. For large data sets the error will be very small. On the other hand it is not very wise to make a (graphical) summary of e.g. 10 bivariate data points.

In case you want to plot multiple (overlapping) bagplots, you may want plots that are semi-transparent. For this you can use the `transparency` flag. If `transparency==TRUE` the alpha layer is set to '99' (hex). This causes the bagplots to appear semi-transparent, but ONLY if the output device is PDF and opened using: `pdf(file="filename.pdf", version="1.4")`. For this reason, the default is `transparency==FALSE`. This feature as well as the arguments to specify different colors has been proposed by Wouter Meuleman.

Value

`compute.bagplot` returns an object of class `bagplot` that could be plotted by `plot.bagplot()`. An object of the bagplot class is a list with the following elements: `center` is a two dimensional vector with the coordinates of the center. `hull.center` is a two column matrix, the rows are the coordinates of the corners of the center region. `hull.bag` and `hull.loop` contain the coordinates of the hull of the bag and the hull of the loop. `pxy.bag` shows you the coordinates of the points of the bag. `pxy.outer` is the two column matrix of the points that are within the fence. `pxy.outlier` represent the outliers. The vector `hdepths` shows the depths of data points. `is.one.dim` is `TRUE` if the data set is (nearly) one dimensional. The dimensionality is decided by analysing the result of `prcomp` which is stored in the element `prdata`. `xy` shows you the data that are used for the bagplot. In the case of very large data sets subsets of the data are used for constructing the bagplot. A data set is very large if there are more data points than `approx.limit`. `xydata` are the input data structured in a two column matrix.

Note

Version of bagplot: 10/2012

Author(s)

Hans Peter Wolf <pwolf@wiwi.uni-bielefeld.de>

References

P. J. Rousseeuw, I. Ruts, J. W. Tukey (1999): The bagplot: a bivariate boxplot, *The American Statistician*, vol. 53, no. 4, 382–387

See Also

boxplot

Examples

```
# example: 100 random points and one outlier
dat <- cbind(rnorm(100) + 100, rnorm(100) + 300)
dat <- rbind(dat, c(105,295))

PlotBag(dat,factor=2.5,create.plot=TRUE,approx.limit=300,
   show.outlier=TRUE,show.looppoints=TRUE,
   show.bagpoints=TRUE,dkmethod=2,
   show.whiskers=TRUE,show.loophull=TRUE,
   show.baghull=TRUE,verbose=FALSE)

# example of Rousseeuw et al., see R-package rpart
cardata <- structure(as.integer( c(2560,2345,1845,2260,2440,
 2285, 2275, 2350, 2295, 1900, 2390, 2075, 2330, 3320, 2885,
 3310, 2695, 2170, 2710, 2775, 2840, 2485, 2670, 2640, 2655,
 3065, 2750, 2920, 2780, 2745, 3110, 2920, 2645, 2575, 2935,
 2920, 2985, 3265, 2880, 2975, 3450, 3145, 3190, 3610, 2885,
 3480, 3200, 2765, 3220, 3480, 3325, 3855, 3850, 3195, 3735,
 3665, 3735, 3415, 3185, 3690, 97, 114, 81, 91, 113, 97, 97,
 98, 109, 73, 97, 89, 109, 305, 153, 302, 133, 97, 125, 146,
 107, 109, 121, 151, 133, 181, 141, 132, 133, 122, 181, 146,
 151, 116, 135, 122, 141, 163, 151, 153, 202, 180, 182, 232,
 143, 180, 180, 151, 189, 180, 231, 305, 302, 151, 202, 182,
 181, 143, 146, 146)), .Dim = as.integer(c(60, 2)),
 .Dimnames = list(NULL, c("Weight", "Disp.")))

PlotBag(cardata,factor=3,show.baghull=TRUE,
  show.loophull=TRUE,precision=1, dkmethod=2)

title("car data Chambers/Hastie 1992")

# points of y=x*x
PlotBag(x=1:30,y=(1:30)^2,verbose=FALSE,dkmethod=2)

# one dimensional subspace
```

```
PlotBag(x=1:50,y=1:50)

# pairwise bagplots
par(las=1)
PlotBagPairs(swiss[, 1:2],
             main="Swiss Fertility and Socioeconomic Indicators (1888) Data")
```

PlotBubble *Draw a Bubble Plot*

Description

Draw a bubble plot, defined by a pair of coordinates x, y to place the bubbles, an area definition configuring the dimension and a color vector setting the color of the bubbles. The legitimation to define a new function instead of just using `plot(symbols(...))` is the automated calculation of the axis limits, ensuring that all bubbles will be fully visible.

Usage

```
PlotBubble(x, ...)

## Default S3 method:
PlotBubble(x, y, area, col = NA, cex = 1, border = par("fg"),
           xlim = NULL, ylim = NULL, na.rm = FALSE, ...)

## S3 method for class 'formula'
PlotBubble(formula, data = parent.frame(), ..., subset, ylab = varnames[response])
```

Arguments

x, y	the x and y co-ordinates for the centres of the bubbles. They can be specified in any way which is accepted by `xy.coords`.
area	a vector giving the area of the bubbles.
col	colors for the bubbles, passed to `symbol`. The default `NA` (or also `NULL`) means do not fill, i.e., draw transparent bubbles.
cex	extension factor for the area.
border	the border color fot the bubbles. The default means `par("fg")`. Use `border = NA` to omit borders.
xlim, ylim	axes limits.
na.rm	logical, should NAs be omitted? Defaults to FALSE.
formula	a formula of the form `lhs ~ rhs` where `lhs` gives the data values and rhs the corresponding groups.
data	an optional matrix or data frame (or similar: see `model.frame`) containing the variables in the formula `formula`. By default the variables are taken from `environment(formula)`.

subset	an optional vector specifying a subset of observations to be used.
ylab	the y-label for the plot used in the formula interface.
...	the dots are passed to the plot function.

Details

Argument inches controls the sizes of the symbols. If TRUE (the default), the symbols are scaled so that the largest dimension of any symbol is one inch. If a positive number is given the symbols are scaled to make largest dimension this size in inches (so TRUE and 1 are equivalent). If inches is FALSE, the units are taken to be those of the appropriate axes. This behaviour is the same as in symbols.

Note

A legend can be added with BubbleLegend.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

BubbleLegend, symbols, sunflowerplot

Examples

```
PlotBubble(latitude ~ longitude, area=(smoky+1)*2e8,
           col=SetAlpha(1, 0.5), data=d.whisky)


cols <- c("olivedrab1","orange","green","mediumturquoise","mediumorchid2","firebrick1")
PlotBubble(x = state.x77[,"Income"], y = state.x77[,"Life Exp"], cex=.00004,
           area = state.x77[,"Population"], col = cols[state.region], border="grey50",
           panel.first=grid(), xlab="Income", ylab="Life Exp.", las=1
)

BubbleLegend(x = "topright", area = c(20000, 10000, 1000), cex=.00004, frame=NA,
             cols=cols[1:3], labels = c(20000, 10000, 1000), cex.names=0.7)

legend(x="bottomright", fill=cols[1:4], legend=levels(state.region))
```

PlotCandlestick *Plot Candlestick Chart*

Description

Plot a candlestick chart. This is used primarily to describe price movements of a security, derivative, or currency over time. Candlestick charts are a visual aid for decision making in stock, foreign exchange, commodity, and option trading.

Usage

```
PlotCandlestick(x, y, vol = NA, xlim = NULL, ylim = NULL,
                col = c("springgreen4","firebrick"),
                border = NA, args.bar = NULL, args.grid = NULL, ...)
```

Arguments

x	a numeric vector for the x-values. Usually a date.
y	the y-values in a matrix (or a data.frame that can be coerced to a matrix) with 4 columns, whereas the first column contains the open price, the second the high, the third the lowest and the 4th the close price of daily stock prices.
vol	the volume, if it should be included in the plot as separate part.
xlim	the x limits (x1, x2) of the plot. The default value, NULL, indicates that the range of the finite values to be plotted should be used.
ylim	the y limits of the plot.
col	color for the body. To better highlight price movements, modern candlestick charts often replace the black or white of the candlestick body with colors such as red for a lower closing and blue or green for a higher closing.
border	the border color of the rectangles. Default is NA, meaning no border will be plotted.
args.grid	the arguments of a potential grid. Default is NULL, which will have a grid plotted. If arguments are provided, they have to be organized as list with the names of the arguments. (For example: ..., args.grid = list(col="red"))
args.bar	optional additional arguments for the volume barplot.
...	the dots are passed to plot() command

Details

Candlesticks are usually composed of the body (black or white), and an upper and a lower shadow (wick): the area between the open and the close is called the real body, price excursions above and below the real body are called shadows. The wick illustrates the highest and lowest traded prices of a security during the time interval represented. The body illustrates the opening and closing trades. If the security closed higher than it opened, the body is white or unfilled, with the opening

price at the bottom of the body and the closing price at the top. If the security closed lower than it opened, the body is black, with the opening price at the top and the closing price at the bottom. A candlestick need not have either a body or a wick.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

PlotBubble, stars

Examples

```
nov <- rbind(
 "2013-05-28"= c(70.99,71.82,70.49,71.49),
 "2013-05-29"= c(71.13,71.90,70.81,71.57),
 "2013-05-30"= c(71.25,71.53,70.90,71.01),
 "2013-05-31"= c(70.86,70.92,70.30,70.30),
 "2013-06-03"= c(70.56,70.89,70.05,70.74),
 "2013-06-04"= c(70.37,71.11,69.67,69.90),
 "2013-06-05"= c(69.76,69.76,68.92,68.99),
 "2013-06-06"= c(69.13,70.02,68.56,70.02),
 "2013-06-07"= c(70.45,70.52,69.51,70.20),
 "2013-06-10"= c(70.53,70.75,70.05,70.20),
 "2013-06-11"= c(69.36,69.66,69.01,69.17),
 "2013-06-12"= c(69.65,70.03,68.85,69.21),
 "2013-06-13"= c(69.21,70.18,69.13,70.10),
 "2013-06-14"= c(70.17,70.48,69.30,69.58),
 "2013-06-17"= c(70.14,70.96,69.98,70.44),
 "2013-06-18"= c(70.55,71.97,70.55,71.49),
 "2013-06-19"= c(71.33,72.00,70.89,70.97),
 "2013-06-20"= c(70.04,70.06,68.40,68.55),
 "2013-06-21"= c(69.15,69.27,67.68,68.21)
)
colnames(nov) <- c("open","high","low","close")

PlotCandlestick(x=as.Date(rownames(nov)), y=nov, border=NA, las=1, ylab="")

# include some volume information
v <- c(213,108,310,762,70,46,411,652,887,704,289,579,934,619,860,35,215,211,8)
PlotCandlestick(x=as.Date(rownames(nov)), y=nov, vol=v,
                border=NA, las=1, ylab="")
```

PlotCashFlow *Cash Flow Plot*

Description

A cash flow plot is a plot used in finance and allows you to graphically depict the timing of the cash flows as well as their nature as either inflows or outflows. An "up" arrow represents money received and a "down" arrow money paid out.

Usage

```
PlotCashFlow(x, y, xlim = NULL, labels = y, mar = NULL,
             cex.per = par("cex"), cex.tck = par("cex") * 0.8,
             cex.cash = par("cex"))
```

Arguments

x	time period of the cashflows (in and out)
y	amount of the cashflows
xlim	range of the x-axis, defaults to `range(x)`.
labels	the labels of the cashflows will be printed outside the arrows.
mar	a vector with 4 elements, defining the margins for the plot
cex.per	the character extension for the period labels
cex.tck	character extension for the ticklabels, tipically years
cex.cash	the character extension for the labels of the cashflows

Author(s)

Andri Signorell <andri@signorell.net>

See Also

NPV

Examples

```
PlotCashFlow(x=c(6:9, 13:15), y=-c(rep(40, 4), rep(50,3)),
             xlim=c(6,17), labels=c(rep(40, 4), rep(50,3)))

PlotCashFlow(x=c(6,8,9,12,17), y=c(10,30,40,50,70))
```

PlotCirc *Plot Circular Plot*

Description

This visualising scheme represents the unidirectional relationship between the rows and the columns of a contingency table.

Usage

```
PlotCirc(tab, acol = rainbow(sum(dim(tab))), aborder = "darkgrey",
         rcol = SetAlpha(acol[1:nrow(tab)], 0.5), rborder = "darkgrey",
         gap = 5, main = "", labels = NULL, cex.lab = 1.0, las = 1,
         adj = NULL, dist = 2)
```

Arguments

tab	a table to be visualised.
acol	the colors for the peripheral annuli.
aborder	the border colors for the peripheral annuli.
rcol	the colors for the ribbons.
rborder	the border colors for the ribbons.
gap	the gap between the entities in degrees.
main	the main title, defaults to "".
labels	the labels. Defaults to the column names and rownames of the table.
las	alignment of the labels, 1 means horizontal, 2 radial and 3 vertical.
adj	adjustments for the labels. (Left: 0, Right: 1, Mid: 0.5)
dist	gives the distance of the labels from the outer circle. Default is 2.
cex.lab	the character extension for the labels.

Details

The visual scheme of representing relationships can be applied to a table, given the observation that a table cell is a relationship (with a value) between a row and column. By representing the row and columns as segments along the circle, the information in the corresponding cell can be encoded as a link between the segments. In general, the cell represents a unidirectional relationship (e.g. row->column) - in this relationship the role of the segments is not interchangeable (e.g. (row,col) and (col,row) are different cells). To identify the role of the segment, as a row or column, the ribbon is made to terminate at the row segment but slightly away from the column segment. In this way, for a given ribbon, it is easy to identify which segment is the row and which is the column.

Value

the calculated points for the labels, which can be used to place userdefined labels.

Author(s)

Andri Signorell <andri@signorell.net>

References

Inspired by https://circos.ca/presentations/articles/vis_tables1/

See Also

PlotPolar

Examples

```
tab <- matrix(c(2,5,8,3,10,12,5,7,15), nrow=3, byrow=FALSE)
dimnames(tab) <- list(c("A","B","C"), c("D","E","F"))
tab


PlotCirc( tab,
  acol = c("dodgerblue","seagreen2","limegreen","olivedrab2","goldenrod2","tomato2"),
  rcol = SetAlpha(c("red","orange","olivedrab1"), 0.5)
)

tab <- table(d.pizza$weekday, d.pizza$operator)
par(mfrow=c(1,2))
PlotCirc(tab, main="weekday ~ operator")
PlotCirc(t(tab), main="operator ~ weekday")
```

PlotConDens *Plot Conditional Densities*

Description

Plot conditional densities by group. For describing how the conditional distribution of a categorical variable y changes over a numerical variable x we have the function cdplot. But if we want to compare multiple densities much work is required. PlotConDens allows to easily enter a grouping variable.

Usage

```
PlotConDens(formula, data, col = NULL, lwd = 2, lty = 1, xlim = NULL, rev = TRUE,
            args.dens = NULL, ...)
```

Arguments

formula	a "formula" of type y ~ x | g with a single dependent factor, a single numerical explanatory variable and a grouping factor g.
data	a data frame containing values for any variables in the formula. By default the environment where PlotConDens was called from is used.
col	a vector of colors to be used to plot the lines. If too short, the values are recycled.
lwd	a vector of linewidths to be used to plot the lines. If too short, the values are recycled.
lty	a vector of linetypes to be used to plot the lines. If too short, the values are recycled.
xlim	the range for the x axis.
rev	logical, should the values of the response variable be reversed? Default is TRUE.
args.dens	additional arguments for the densitiy curves.
...	the dots are passed on to plot().

Details

Especially when we're modelling binary response variables we might want to know, how the binary variable behaves along some numeric predictors.

Value

the functions for the curves

Author(s)

Andri Signorell <andri@signorell.net>

See Also

cdplot, spineplot, density, PlotMultiDens

Examples

```
data(Pima.tr2, package="MASS")
PlotConDens (type ~ age | I((npreg > 0)*1L),
             data=Pima.tr2, col=c(DescTools::hblue, DescTools::hred), rev=FALSE,
             panel.first=quote(grid()))
```

PlotCorr *Plot a Correlation Matrix*

Description

This function produces a graphical display of a correlation matrix. The cells of the matrix can be shaded or colored to show the correlation value.

Usage

```
PlotCorr(x, cols = colorRampPalette(c(Pal()[2], "white",
                                      Pal()[1]), space = "rgb")(20),
         breaks = seq(-1, 1, length = length(cols) + 1),
         border = "grey", lwd = 1,
         args.colorlegend = NULL, xaxt = par("xaxt"), yaxt = par("yaxt"),
         cex.axis = 0.8, las = 2, mar = c(3, 8, 8, 8), mincor = 0,
         main = "", clust = FALSE, ...)
```

Arguments

x	x is a correlation matrix to be visualized.
cols	the colors for shading the matrix. Uses the package's option "col1" and "col2" as default.
breaks	a set of breakpoints for the colours: must give one more breakpoint than colour. These are passed to image() function. If breaks is specified then the algorithm used follows cut, so intervals are closed on the right and open on the left except for the lowest interval.
border	color for borders. The default is grey. Set this argument to NA if borders should be omitted.
lwd	line width for borders. Default is 1.
args.colorlegend	list of arguments for the ColorLegend. Use NA if no color legend should be painted.
xaxt	parameter to define, whether to draw an x-axis, defaults to "n".
yaxt	parameter to define, whether to draw an y-axis, defaults to "n".
cex.axis	character extension for the axis labels.
las	the style of axis labels.
mar	sets the margins, defaults to mar = c(3, 8, 8, 8) as we need a bit more room on the right.
mincor	numeric value between 0 and 1, defining the smallest correlation that is to be displayed. If this is >0 then all correlations with a lower value are suppressed.
main	character, the main title.
clust	logical. If set to TRUE, the correlations will be clustered in order to aggregate similar values.
...	the dots are passed to the function image, which produces the plot.

Value

no values returned.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

image, ColorLegend, corrgram(), PlotWeb()

Examples

```
m <- cor(d.pizza[,sapply(d.pizza, IsNumeric, na.rm=TRUE)], use="pairwise.complete.obs")

PlotCorr(m, cols=colorRampPalette(c("red", "black", "green"), space = "rgb")(20))
PlotCorr(m, cols=colorRampPalette(c("red", "black", "green"), space = "rgb")(20),
         args.colorlegend=NA)

m <- PairApply(d.diamonds[, sapply(d.diamonds, is.factor)], CramerV, symmetric=TRUE)
PlotCorr(m, cols = colorRampPalette(c("white", "steelblue"), space = "rgb")(20),
         breaks=seq(0, 1, length=21), border="black",
       args.colorlegend = list(labels=sprintf("%.1f", seq(0, 1, length = 11)), frame=TRUE)
)
title(main="Cramer's V", line=2)
text(x=rep(1:ncol(m),ncol(m)), y=rep(1:ncol(m),each=ncol(m)),
     label=sprintf("%0.2f", m[,ncol(m):1]), cex=0.8, xpd=TRUE)

# Spearman correlation on ordinal factors
csp <- cor(data.frame(lapply(d.diamonds[,c("carat", "clarity", "cut", "polish",
                     "symmetry", "price")], as.numeric)), method="spearman")
PlotCorr(csp)

m <- cor(mtcars)
PlotCorr(m, col=Pal("RedWhiteBlue1", 100), border="grey",
         args.colorlegend=list(labels=Format(seq(-1,1,.25), digits=2), frame="grey"))

# display only correlation with a value > 0.7
PlotCorr(m, mincor = 0.7)
x <- matrix(rep(1:ncol(m),each=ncol(m)), ncol=ncol(m))
y <- matrix(rep(ncol(m):1,ncol(m)), ncol=ncol(m))
txt <- Format(m, digits=3, ldigits=0)
idx <- upper.tri(matrix(x, ncol=ncol(m)), diag=FALSE)

# place the text on the upper triagonal matrix
text(x=x[idx], y=y[idx], label=txt[idx], cex=0.8, xpd=TRUE)

# or let's get rid of all non significant correlations
p <- PairApply(mtcars,  function(x, y) cor.test(x, y)$p.value, symmetric=TRUE)
# or somewhat more complex with outer
p0 <- outer(1:ncol(m),  1:ncol(m),
           function(a, b)
```

```
            mapply(
              function(x, y) cor.test(mtcars[, x], mtcars[, y])$p.value,
              a, b))
# ok, got all the p-values, now replace > 0.05 with NAs
m[p > 0.05] <- NA
PlotCorr(m)

# the text
n <- ncol(m)
text(x=rep(seq(n), times=n),
     y=rep(rev(seq(n)), rep.int(n, n)),
     labels=Format(m, digits=2, na.form=""),
     cex=0.8, xpd=TRUE)
# the text could also be set with outer, but this function returns an error,
# based on the fact that text() does not return some kind of result
# outer(X = 1:nrow(m),  Y = ncol(m):1,
#  FUN = "text", labels = Format(m, digits=2, na.form = ""),
#  cex=0.8, xpd=TRUE)


# put similiar correlations together
PlotCorr(m, clust=TRUE)

# same as
idx <- order.dendrogram(as.dendrogram(
          hclust(dist(m), method = "mcquitty")
       ))
PlotCorr(m[idx, idx])


# plot only upper triangular matrix and move legend to bottom
m <- cor(mtcars)
m[lower.tri(m, diag=TRUE)] <- NA

p <- PairApply(mtcars,  function(x, y) cor.test(x, y)$p.value, symmetric=TRUE)
m[p > 0.05] <- NA

PlotCorr(m, mar=c(8,8,8,8), yaxt="n",
         args.colorlegend = list(x="bottom", inset=-.15, horiz=TRUE,
                                 height=abs(LineToUser(line = 2.5, side = 1)),
                                 width=ncol(m)))
mtext(text = rev(rownames(m)), side = 4, at=1:ncol(m), las=1, line = -5, cex=0.8)

text(1:ncol(m), ncol(m):1, colnames(m), xpd=NA, cex=0.8, font=2)

n <- ncol(m)
text(x=rep(seq(n), times=n),
     y=rep(rev(seq(n)), rep.int(n, n)),
     labels=Format(t(m), digits=2, na.form=""),
     cex=0.8, xpd=TRUE)
```

PlotDot *Cleveland's Dot Plots*

Description

Draw a Cleveland dot plot. This is an extended version of dotchart with an added option for error bars, an add argument and several more options. PlotCI() is a small helpfunction to facilitate ci-plots of several models.

Usage

```
PlotDot(x, labels = NULL, groups = NULL, gdata = NULL,
        cex = par("cex"), pch = 21, gpch = 21, bg = par("bg"),
      color = par("fg"), gcolor = par("fg"), lcolor = "gray", lblcolor = par("fg"),
        xlim = NULL, main = NULL, xlab = NULL, ylab = NULL,
        xaxt = NULL, yaxt = NULL, add = FALSE, args.errbars = NULL,
        cex.axis = par("cex.axis"), cex.pch = 1.2, cex.gpch = 1.2,
        gshift = 2, automar = TRUE, ...)


PlotDotCI(..., grp = 1, cex = par("cex"),
       pch = 21, gpch = 21, bg = par("bg"), color = par("fg"), gcolor = par("fg"),
         lcolor = "gray", lblcolor = par("fg"), xlim = NULL, main = NULL,
         xlab = NULL, ylab = NULL, xaxt = NULL, yaxt = NULL,
         cex.axis = par("cex.axis"), cex.pch = 1.2, cex.gpch = 1.2,
         gshift = 2, automar = TRUE)
```

Arguments

x	either a vector or matrix of numeric values (NAs are allowed). If x is a matrix the overall plot consists of juxtaposed dotplots for each row. Inputs which satisfy is.numeric(x) but not is.vector(x) || is.matrix(x) are coerced by as.numeric, with a warning.
labels	a vector of labels for each point. For vectors the default is to use names(x) and for matrices the row labels dimnames(x)[[1]].
groups	an optional factor indicating how the elements of x are grouped. If x is a matrix, groups will default to the columns of x.
gdata	data values for the groups. This is typically a summary such as the median or mean of each group.
cex	the character size to be used. Setting cex to a value smaller than one can be a useful way of avoiding label overlap. Unlike many other graphics functions, this sets the actual size, not a multiple of par("cex").
pch	the plotting character or symbol to be used. Default is 21.
gpch	the plotting character or symbol to be used for group values.

bg	the background color of plotting characters or symbols to be used; use par(bg= *) to set the background color of the whole plot.
color	the color(s) to be used for points and labels.
gcolor	the single color to be used for group labels and values.
lcolor	the color(s) to be used for the horizontal lines.
lblcolor	the color(s) to be used for labels.
xlim	horizontal range for the plot, see plot.window, e.g.
main	overall title for the plot, see title.
xlab, ylab	axis annotations as in title.
xaxt	a character which specifies the x axis type. Specifying "n" suppresses plotting of the axis.
yaxt	a character which specifies the y axis type. Specifying "n" suppresses plotting of the axis.
add	logical specifying if bars should be added to an already existing plot; defaults to FALSE.
args.errbars	optional arguments for adding error bars. All arguments for ErrBars can be supplied. If left to NULL (default), no error bars will be plotted.
cex.axis	The magnification to be used for axis annotation relative to the current setting of cex.
cex.pch	The magnification to be used for plot symbols relative to the current setting of cex.
cex.gpch	The magnification to be used for group symbols relative to the current setting of cex.
gshift	the number of characters, for which the grouplabels should be shift to the left compared to the sublabels.
automar	logical (default TRUE), defining if the left margin should be set according to the width of the given labels, resp. grouplabels. If set to FALSE the margins are taken from par("mar").
...	graphical parameters can also be specified as arguments.
grp	an integer, defining if the the coefficients should be grouped along the first or the second dimension (default is 1).

Details

Dot plots are a reasonable substitute for bar plots. This function is invoked to produce dotplots as described in Cleveland (1985).

For PlotDotCI() the dots are a list of matrices with 3 columns, whereas the first is the coefficent, the second the lower and the third the upper end of the confidence interval.

Value

Return the y-values used for plotting.

Author(s)

R-Core with some extensions by Andri Signorell <andri@signorell.net>

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Cleveland, W. S. (1985) *The Elements of Graphing Data.* Monterey, CA: Wadsworth.

Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

See Also

dotchart, PlotDotCI

Examples

```
PlotDot(VADeaths, main = "Death Rates in Virginia - 1940")
op <- par(xaxs = "i")  # 0 -- 100%
PlotDot(t(VADeaths), xlim = c(0,100),
        main = "Death Rates in Virginia - 1940")
par(op)

# add some error bars
PlotDot(VADeaths, main="Death Rates in Virginia - 1940", col="red", pch=21,
        args.errbars = list(from=VADeaths-2, to=VADeaths+2, mid=VADeaths,
                            cex=1.4))

# add some other values
PlotDot(VADeaths+3, pch=15, col="blue", add=TRUE)

# same as PlotDotCI
xci <- do.call(rbind, tapply( d.pizza$delivery_min, d.pizza$driver,
                              MeanCI, conf.level=0.99, na.rm=TRUE))

PlotDot(xci[,1], main="delivery_min ~ driver", pch=21, bg="grey80", col="black",
     args.errbars = list(from=xci[,2], to=xci[,3], mid=xci[,1], lwd=2, col="grey40", cex=1.5),
        xlim=c(15,35), panel.before=grid())

# with group data
x <- with(d.pizza, tapply(temperature, list(area, driver), mean, na.rm=TRUE))

PlotDot(x, gdata = tapply(d.pizza$temperature, d.pizza$driver, mean, na.rm=TRUE),
        gpch = 15)

# special format
par(lend=1)

PlotDot(VADeaths, main="Death Rates in Virginia - 1940", pch="|",
        lcolor = DescTools::hecru, col=DescTools::hred,
        args.errbars = list(from=VADeaths-2, to=VADeaths+2, mid=VADeaths,
                            cex=1.3, lwd=8, code=0, col=DescTools::hgreen))
```

```
# Error bars for binomial confidence intervals
tab <- table(d.pizza$driver, d.pizza$wine_delivered)
xci <- SetNames(BinomCI(tab[,1], rowSums(tab)), rownames=rownames(tab))
PlotDot(xci[,1], main="wine delivered ~ driver ", xlim=c(0,1),
        args.errbars=list(from=xci[,-1], mid=xci[,1], pch=21))


# Error bars for confidence intervals for means
xci <- do.call(rbind, tapply(d.pizza$delivery_min, d.pizza$driver,
                             MeanCI, conf.level=0.99, na.rm=TRUE))

PlotDot(xci[, 1], main="delivery_min ~ driver", args.errbars=list(from=xci))


# Setting the colours
# define some error bars first
lci <- sweep(x = VADeaths, MARGIN = 2, FUN = "-", 1:4)
uci <- sweep(x = VADeaths, MARGIN = 1, FUN = "+", 1:5)

PlotDot(VADeaths, main="This should only show how to set the colours, not be pretty",
        pch=21, col=c("blue","grey"), bg=c("red", "yellow"),
        gcolor = c("green", "blue", "orange", "magenta"), gdata=c(10,20,30,40),
        gpch = c(15:18), lcolor = "orange",
        args.errbars = list(from=lci, to=uci, mid=VADeaths, cex=1.4))
```

`PlotECDF` *Empirical Cumulative Distribution Function*

Description

Faster alternative for plotting the empirical cumulative distribution function (ecdf). The function offers the option to construct the ecdf on the base of a histogram, which makes sense, when x is large. So the plot process is much faster, without loosing much precision in the details.

Usage

```
PlotECDF(x, breaks = NULL, col = Pal()[1], ylab = "",
         lwd = 2, xlab = NULL, ...)
```

Arguments

`x`	numeric vector of the observations for ecdf.
`breaks`	will be passed directly to `hist`. If left to `NULL`, no histogram will be used.
`col`	color of the line.
`ylab`	label for the y-axis.
`lwd`	line width.
`xlab`	label for the x-axis.
`...`	arguments to be passed to subsequent functions.

Details

The stats function `plot.ecdf` is fine for vectors that are not too large. However for n ~ 1e7 we would observe a dramatic performance breakdown (possibly in combination with the use of `do.call`).

`PlotECDF` is designed as alternative for quicker plotting the ecdf for larger vectors. If `breaks` are provided as argument, a histogram with that number of breaks will be calculated and the ecdf will use those frequencies instead of respecting every single point.
Note that a plot will rarely need more than ~1'000 points on x to have a sufficient resolution on usual terms. `PlotFdist` will also use this number of breaks by default.

Value

no value returned, use `plot.ecdf` if any results are required.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`plot.ecdf`, `PlotFdist`

Examples

```
PlotECDF(d.pizza$temperature)

# make large vector
x <- rnorm(n=1e7)

# plot only 1000 points instead of 1e7
PlotECDF(x, breaks=1000)
```

PlotFaces *Chernoff Faces*

Description

Plot Chernoff faces. The rows of a data matrix represent cases and the columns the variables.

Usage

```
PlotFaces(xy, which.row, fill = FALSE, nr, nc,
          scale = TRUE, byrow = FALSE, main, labels, col = "white")
```

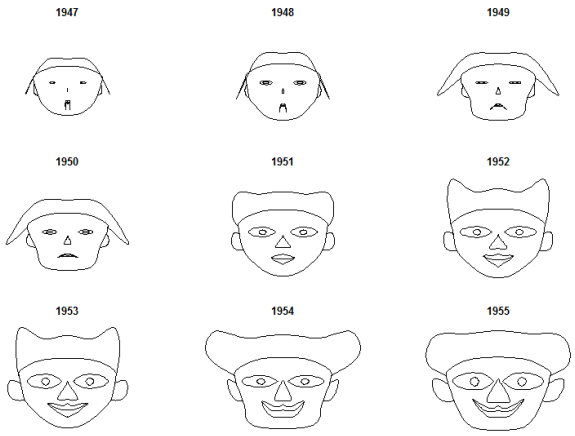
Arguments

xy	xy data matrix, rows represent individuals and columns attributes.
which.row	defines a permutation of the rows of the input matrix.
fill	logic. If set to TRUE, only the first nc attributes of the faces are transformed, nc is the number of columns of x.
nr	number of columns of faces on graphics device
nc	number of rows of faces
scale	logic. If set to TRUE, attributes will be normalized.
byrow	if(byrow==TRUE), x will be transposed.
main	title.
labels	character strings to use as names for the faces.
col	a vector of colors used for the parts of the faces. Colors are recycled in the order: "nose", "eyes", "hair", "face", "lips", "ears". Default is NA, which will omit colors.

Details

The features paramters of this implementation are:

- 1 height of face
- 2 width of face
- 3 shape of face
- 4 height of mouth
- 5 width of mouth
- 6 curve of smile
- 7 height of eyes
- 8 width of eyes
- 9 height of hair
- 10 width of hair
- 11 styling of hair
- 12 height of nose
- 13 width of nose
- 14 width of ears
- 15 height of ears



For details look at the literate program of `faces`

Value

information about usage of variables for face elements is returned invisibly

Note

based on version 12/2009

Author(s)

H. P. Wolf, some changes Andri Signorell <andri@signorell.net>

References

Chernoff, H. (1973) The use of faces to represent statistiscal assoziation, *JASA*, 68, pp 361–368.

The smooth curves are computed by an algorithm found in:
Ralston, A. and Rabinowitz, P. (1985) *A first course in numerical analysis*, McGraw-Hill, pp 76ff.

Examples

```
PlotFaces(rbind(1:3,5:3,3:5,5:7))

data(longley)
PlotFaces(longley[1:9,])

set.seed(17)
PlotFaces(matrix(sample(1:1000,128,), 16, 8), main="random faces")
```

```
means <- lapply(iris[,-5], tapply, iris$Species, mean)
m <- t(do.call(rbind, means))
m <- cbind(m, matrix(rep(1, 11*3), nrow=3))

# define the colors, first for all faces the same
col <- replicate(3, c("orchid1", "olivedrab", "goldenrod4",
                      "peachpuff", "darksalmon", "peachpuff3"))
rownames(col) <- c("nose","eyes","hair","face","lips","ears")
# change haircolor individually for each face
col[3, ] <- c("lightgoldenrod", "coral3", "sienna4")

z <- PlotFaces(m, nr=1, nc=3, col=col)

# print the used coding
print(z$info, right=FALSE)
```

PlotFdist *Frequency Distribution Plot*

Description

This function was developed to create a univariate graphical representation of the frequency distribution of a numerical vector. It combines a histogram, a density curve, a boxplot and the empirical cumulative distribution function (ecdf) in one single plot. A rug as well as a model distribution curve (e.g. a normal curve) can optionally be superposed. This results in a dense and informative picture of the facts. Still the function remains flexible as all possible arguments can be passed to the single components (`hist`, `boxplot` etc.) as a list (see examples).

Usage

```
PlotFdist(
  x,
  main = deparse(substitute(x)),
  xlab = "",
  xlim = NULL,
  args.hist = NULL,
  args.rug = NA,
  args.dens = NULL,
  args.curve = NA,
  args.boxplot = NULL,
  args.ecdf = NULL,
  args.curve.ecdf = NA,
  heights = NULL,
  pdist = NULL,
  na.rm = FALSE,
  cex.axis = NULL,
  cex.main = NULL,
  mar = NULL,
```

```
  las = 1
)
```

Arguments

x	the numerical variable, whose distribution is to be plotted.
main	main title of the plot.
xlab	label of the x-axis, defaults to "". (The name of the variable is typically placed in the main title and would be redundant here.)
xlim	range of the x-axis, defaults to a pretty range(x, na.rm = TRUE).
args.hist	list of additional arguments to be passed to the histogram hist(). The defaults chosen when setting args.hist = NULL are more or less the same as in hist. The argument type defines, whether a histogram ("hist") or a plot with type = "h" (for 'histogram' like vertical lines for mass representation) should be used. The arguments for a "h-plot"" will be col, lwd, pch.col, pch, pch.bg for the line and for an optional point character on top. The default type used will be chosen on the structure of x. If x is an integer with up to 12 unique values there will be a "h-plot" and else a histogram!
args.rug	list of additional arguments to be passed to the function rug(). Use args.rug = NA if no rug should be added. This is the default. Use args.rug = NULL to add rug with reasonable default values.
args.dens	list of additional arguments to be passed to density. Use args.dens = NA if no density curve should be drawn. The defaults are taken from density.
args.curve	list of additional arguments to be passed to curve. This argument allows to add a fitted distribution curve to the histogram. By default no curve will be added (args.curve = NA). If the argument is set to NULL, a normal curve with mean(x) and sd(x) will be drawn. See examples for more details.
args.boxplot	list of additional arguments to be passed to the boxplot boxplot(). The defaults are pretty much the same as in boxplot. The two additional arguments pch.mean (default 23) and col.meanci (default "grey80") control, if the mean is displayed within the boxplot. Setting those arguments to NA will prevent them from being displayed.
args.ecdf	list of additional arguments to be passed to ecdf(). Use args.ecdf = NA if no empirical cumulation function should be included in the plot. The defaults are taken from plot.ecdf.
args.curve.ecdf	list of additional arguments to be passed to curve. This argument allows to add a fitted distribution curve to the cumulative distribution function. By default no curve will be added (args.curve.ecdf = NA). If the argument is set to NULL, a normal curve with mean(x) and sd(x) will be drawn. See examples for more details.
heights	heights of the plotparts, defaults to c(2,0.5,1.4) for the histogram, the boxplot and the empirical cumulative distribution function, resp. to c(2,1.5) for a histogram and a boxplot only.

pdist	distances of the plotparts, defaults to c(0, 0), say there will be no distance between the histogram, the boxplot and the ecdf-plot. This can be useful for instance in case that the x-axis has to be added to the histogram.
na.rm	logical, should NAs be omitted? Histogram and boxplot could do without this option, but the density-function refuses to plot with missings. Defaults to FALSE.
cex.axis	character extension factor for the axes.
cex.main	character extension factor for the main title. Must be set in dependence of the plot parts in order to get a harmonic view.
mar	A numerical vector of the form c(bottom, left, top, right) which gives the number of lines of outer margin to be specified on the four sides of the plot. The default is c(0, 0, 3, 0).
las	numeric in c(0,1,2,3); the orientation of axis labels. See par.

Details

Performance has been significantly improved, but if x is growing large (n > 1e7) the function will take its time to complete. Especially the density curve and the ecdf, but as well as the boxplot (due to the chosen alpha channel) will take their time to calculate and plot.
In such cases consider taking a sample, i.e. PlotFdist(x[sample(length(x),size=5000)]), the big picture of the distribution won't usually change much. .

Author(s)

Andri Signorell andri@signorell.net

See Also

hist, boxplot, ecdf, density, rug, layout

Examples

```
PlotFdist(x=d.pizza$delivery_min, na.rm=TRUE)

# define additional arguments for hist, dens and boxplot
# do not display the mean and its CI on the boxplot
PlotFdist(d.pizza$delivery_min, args.hist=list(breaks=50),
  args.dens=list(col="olivedrab4"), na.rm=TRUE,
  args.boxplot=list(col="olivedrab2", pch.mean=NA, col.meanci=NA))


# do a "h"-plot instead of a histogram for integers
x <- sample(runif(10), 100, replace = TRUE)
PlotFdist(x, args.hist=list(type="mass"))

pp <- rpois(n = 100, lambda = 3)
PlotFdist(pp, args.hist = list(type="mass", pch=21, col=DescTools::horange,
          cex.pch=2.5, col.pch=DescTools::hred, lwd=3, bg.pch="white"),
          args.boxplot = NULL, args.ecdf = NA, main="Probability mass function")
```

```
# special arguments for hist, density and ecdf
PlotFdist(x=faithful$eruptions,
          args.hist=list(breaks=20), args.dens=list(bw=.1),
          args.ecdf=list(cex=1.2, pch=16, lwd=1), args.rug=TRUE)

# no density curve, no ecdf but add rug instead, make boxplot a bit higher
PlotFdist(x=d.pizza$delivery_min, na.rm=TRUE, args.dens=NA, args.ecdf=NA,
  args.hist=list(xaxt="s"),  # display x-axis on the histogram
  args.rug=TRUE, heights=c(3, 2.5), pdist=2.5, main="Delivery time")

# alpha channel on rug is cool, but takes its time for being drawn...
PlotFdist(x=d.pizza$temperature, args.rug=list(col=SetAlpha("black", 0.1)), na.rm=TRUE)

# plot a normal density curve, but no boxplot nor ecdf
x <- rnorm(1000)
PlotFdist(x, args.curve = NULL, args.boxplot=NA, args.ecdf=NA)

# compare with a t-distribution
PlotFdist(x, args.curve = list(expr="dt(x, df=2)", col="darkgreen"),
          args.boxplot=NA, args.ecdf=NA)
legend(x="topright", legend=c("kernel density", "t-distribution (df=2)"),
       fill=c(getOption("col1", DescTools::hred), "darkgreen"), xpd=NA)

# add a gamma distribution curve to both, histogram and ecdf
ozone <- airquality$Ozone; m <- mean(ozone, na.rm = TRUE); v <- var(ozone, na.rm = TRUE)
PlotFdist(ozone, args.hist = list(breaks=15),
  args.curve = list(expr="dgamma(x, shape = m^2/v, scale = v/m)", col=DescTools::hecru),
 args.curve.ecdf = list(expr="pgamma(x, shape = m^2/v, scale = v/m)", col=DescTools::hecru),
  na.rm = TRUE, main = "Airquality - Ozone")

legend(x="topright", xpd=NA,
       legend=c(expression(plain("gamma:  ") * Gamma * " " * bgroup("(", k * " = " *
          over(bar(x)^2, s^2) * " , " * theta * plain(" = ") * over(s^2, bar(x)), ")") ),
                "kernel density"),
       fill=c(DescTools::hecru, getOption("col1", DescTools::hred)), text.width = 0.25)
```

PlotFun *Plot a Function*

Description

Plots mathematical expressions in one variable using the formula syntax.

Usage

```
PlotFun(FUN, args = NULL, from = NULL, to = NULL, by = NULL,
        xlim = NULL, ylim = NULL, polar = FALSE, type = "l",
        col = par("col"), lwd = par("lwd"), lty = par("lty"),
        pch = NA, mar = NULL, add = FALSE, ...)
```

Arguments

FUN	a mathematical expression defined using the formula syntax: f(x) ~ x. x and y can as well be functions of a parameter t: y(t) ~ x(t) (see examples).
args	a list of additional parameters defined in the expression besides the independent variable.
from, to	the range over which the function will be plotted.
by	number: increment of the sequence.
xlim, ylim	NULL or a numeric vector of length 2; if non-NULL it provides the defaults for c(from, to) and, unless add=TRUE, selects the x-limits of the plot - see plot.window.
polar	logical. Should polar coordinates be used? Defaults to FALSE.
type	plot type: see plot.default
col	colors of the lines.
lwd	line widths for the lines.
lty	line type of the lines.
pch	plotting 'character', i.e., symbol to use.
mar	A numerical vector of the form c(bottom, left, top, right) which gives the number of lines of margin to be specified on the four sides of the plot. The default is c(3,3,3,3).
add	logical; if TRUE add to an already existing plot; if NA start a new plot taking the defaults for the limits and log-scaling of the x-axis from the previous plot. Taken as FALSE (with a warning if a different value is supplied) if no graphics device is open.
...	the dots are passed to the plot, resp. lines function.

Details

A function can be plotted with curve. This function here adds some more features, one enabling to use a formula for defining the function to plot. This enables as well a parametric equation to be entered straight forward. Parameters of a function can be set separatedly. The aspect ratio y/x will be set to 1 by default. (See plot.window for details.)

If the argument axes (given in the dots) is not set to FALSE centered axis at a horizontal and vertical position of 0 will be drawn, containing major and minor ticks.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

curve

Examples

```
# simple quadratic function y = x^2
PlotFun(x^2 ~ x)

par(mfrow=c(3,4))

# Cartesian leaf
PlotFun(3*a*z^2 /(z^3+1) ~ 3*a*z /(z^3+1+b), args=list(a=2, b=.1), from=-10, to=10, by=0.1,
        xlim=c(-5,5), ylim=c(-5,5), col="magenta", asp=1, lwd=2 )

# family of functions
PlotFun(a*exp(-x/5)*sin(n*x) ~ x, args=list(n=4, a=3), from=0, to=10, by=0.01,
        col="green")

PlotFun(a*exp(-x/5)*sin(n*x) ~ x, args=list(n=6, a=3), from=0, to=10, by=0.01,
        col="darkgreen", add=TRUE)

# cardioid
PlotFun(a*(1+cos(t)) ~ t, args=list(a=2), polar=TRUE, from=0, to=2*pi+0.1, by=0.01, asp=1)

PlotFun(13*cos(t) - 5*cos(2*t) - 2*cos(3*t) - cos(4*t) ~ 16*sin(t)^3,
        from=0, to=2*pi, by=0.01, asp=1, xlim=c(-20,20), col="red", lwd=2)


PlotFun(a*sin(2*t)*cos(2*t) ~ t, args=list(a=6), polar=TRUE, from=0, to=2*pi+0.1, by=0.01,
        col="orange")

# astroid
PlotFun(a*sin(t)^3 ~ a*cos(t)^3, args=list(a=2), from=0, to=2*pi+0.1, lwd=3, by=0.01,
        col="red")

# lemniscate of Bernoulli
PlotFun((2*a^2*cos(2*t))^2 ~ t, args=list(a=1), polar=TRUE, from=0, to=2*pi+0.1, by=0.01,
        col="darkblue")

# Cycloid
PlotFun(a*(1-cos(t)) ~ a*(t-sin(t)), args=list(a=0.5), from=0, to=30, by=0.01,
        col="orange")

# Kreisevolvente
PlotFun(a*(sin(t) - t*cos(t)) ~ a*(cos(t) + t*sin(t)), args=list(a=0.2), from=0, to=50, by=0.01,
        col="brown")


PlotFun(sin(2*t) ~ sin(t), from=0, to=2*pi, by=0.01, col="blue", lwd=2)

# multiple values for one parameter
sapply(1:3, function(a) PlotFun(sin(a*x) ~ x,
                                args=list(a=a), from=0, to=2*pi, by=0.01,
                                add=(a!=1), col=a))

PlotFun(sin(3*x) ~ x, polar=TRUE, from=0, to=pi, by=0.001, col=DescTools::hred, lwd=2)
```

```
PlotFun(1 + 1/10 * sin(10*x) ~ x, polar=TRUE, from=0, to=2*pi, by=0.001,
        col=DescTools::hred)
PlotFun(sin(x) ~ cos(x), polar=FALSE, from=0, to=2*pi, by=0.01, add=TRUE, col="blue")
```

PlotLinesA *Plot Lines*

Description

Plot the columns of one matrix against the columns of another. Adds a legend on the right at the endpoints of lines.

Usage

```
PlotLinesA(x, y, col = 1:5, lty = 1, lwd = 1, lend = par("lend"),
       xlab = NULL, ylab = NULL, xlim = NULL, ylim = NULL, xaxt = NULL, yaxt = NULL,
           cex = 1, args.legend = NULL, main = NULL, grid = TRUE, mar = NULL,
           pch = NA, pch.col = par("fg"), pch.bg = par("bg"), pch.cex = 1, ...)
```

Arguments

x, y	vectors or matrices of data for plotting. The number of rows should match. If one of them are missing, the other is taken as y and an x vector of 1:n is used. Missing values (NAs) are allowed.
col	vector of colors. Colors are used cyclically.
lty, lwd, lend	vector of line types, widths, and end styles. The first element is for the first column, the second element for the second column, etc., even if lines are not plotted for all columns. Line types will be used cyclically until all plots are drawn.
xlab, ylab	titles for x and y axes, as in plot.
xlim, ylim	ranges of x and y axes, as in plot.
xaxt, yaxt	a character which specifies the x axis type. Specifying "n" suppresses plotting of the axis. The standard value is "s", any value other than "n" implies plotting.
cex	character expansion factor relative to current par("cex").
args.legend	list of additional arguments for the legend; names of the list are used as argument names. If set to NA, the legend will be suppressed. See details.
main	an overall title for the plot.
grid	logical adds an nx by ny rectangular grid to an existing plot.
mar	the margins of the plot.

pch	character string or vector of 1-characters or integers for plotting characters, see points. The first character is the plotting-character for the first plot, the second for the second, etc. The default is the digits (1 through 9, 0) then the lowercase and uppercase letters. If no points should be drawn set this argument to NA (this is the default).
pch.col	vector of colors for the points. Colors are used cyclically. Ignored if pch = NA.
pch.bg	vector of background (fill) colors for the open plot symbols given by pch = 21:25 as in points. The default is set to par("bg"). Ignored if pch = NA.
pch.cex	vector of character expansion sizes, used cyclically. This works as a multiple of par("cex"). Default is 1.0. Ignored if pch = NA.
...	the dots are sent to matplot
.	

Details

This function is rather a template, than a function. It wraps matplot to generate a lines plot and adds a rather sophisticated legend on the right side, while calculating appropriate margins. A grid option is included (as panel.first does not work in matplot).

As in matplot, the first column of x is plotted against the first column of y, the second column of x against the second column of y, etc. If one matrix has fewer columns, plotting will cycle back through the columns again. (In particular, either x or y may be a vector, against which all columns of the other argument will be plotted.)

The legend can be controlled by following arguments:

```
list(line = c(1, 1), width = 1, y = SpreadOut(unlist(last),
     mindist = 1.2 * strheight("M") * par("cex")),
     labels = names(last), cex = par("cex"), col = col[ord],
     lwd = lwd[ord], lty = lty[ord])
```

All arguments are recycled.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

matplot, par

Examples

```
m <- matrix(c(3,4,5,1,5,4,2,6,2), nrow = 3,
            dimnames = list(dose = c("A","B","C"),
                            age  = c("2010","2011","2012")))

PlotLinesA(m, col=c(Pal("Helsana")), main="Dose ~ age_grp", lwd=3, ylim=c(1, 10))
```

```
# example from MASS
shoes <- list(
  A = c(13.2, 8.2, 10.9, 14.3, 10.7, 6.6, 9.5, 10.8, 8.8, 13.3),
  B = c(14, 8.8, 11.2, 14.2, 11.8, 6.4, 9.8, 11.3, 9.3, 13.6))

PlotLinesA(do.call(rbind, shoes), xlim=c(0.75,2.25), col=1, main="shoes",
           pch=21, pch.bg="white", pch.col=1, pch.cex=1.5)

# let's define some arbitrary x-coordinates
PlotLinesA(x=c(1,2,6,8,15), y=VADeaths)
```

PlotLog *Logarithmic Plot*

Description

The base function grid() does not support logarithmic scales very well. Especially when more lines are required, grids have to be created manually. PlotLog creates a plot with at least one logarithmic axis and places a logarithmic grid in the background of the data.

Usage

```
PlotLog(x, ..., args.grid = NULL, log = "xy")
```

Arguments

x	the coordinates of points in the plot. Alternatively, a single plotting structure, function or any R object with a plot method can be provided.
...	the dots are passed on to the function plot().
args.grid	a list of arguments for the grid. This contains line type, line width and line color, separately for major gridlines and for minor gridlines. args.grid=list(lwd=1, lty=3, col="grey85", col.min="grey60") are used as default. If the argument is set to NA, no grid will be plotted.
log	a character string which contains "x" if the x axis is to be logarithmic, "y" if the y axis is to be logarithmic and "xy" or "yx" if both axes are to be logarithmic.

Value

nothing

Author(s)

Andri Signorell <andri@signorell.net>

See Also

axis

Examples

```
PlotLog(brain ~ body, data=MASS::Animals, log="xy",
        xlim=c(.01, 1e5), ylim=c(.1, 1e4), main="Animal brain/body size",
        pch=21, bg="grey", cex=1.5)
```

PlotMarDens *Scatterplot With Marginal Densities*

Description

Draw a scatter plot with marginal densities on the x- and y-axis. Groups can be defined by grp.

Usage

```
PlotMarDens(x, y, grp = 1, xlim = NULL, ylim = NULL,
            col = rainbow(nlevels(factor(grp))),
            mardens = c("all","x","y"), pch = 1, pch.cex = 1,
            main = "", na.rm = FALSE, args.legend = NULL,
            args.dens = NULL, ...)
```

Arguments

x	numeric vector of x values.
y	numeric vector of y values (of same length as x).
grp	grouping variable(s), typically factor(s), all of the same length as x.
xlim	the x limits of the plot.
ylim	the y limits of the plot.
col	the colors for lines and points. Uses `rainbow()` colors by default.
mardens	which marginal densities to plot. Can be set to either just x or y, or both (`"all"`, latter being the default).
pch	a vector of plotting characters or symbols.
pch.cex	magnification to be used for plotting characters relative to the current setting of cex.
main	a main title for the plot, see also title.
na.rm	logical, should NAs be omitted? Defaults to `FALSE`.
args.legend	list of additional arguments for the legend. `args.legend` set to `NA` prevents a legend from being drawn.
args.dens	list of additional arguments to be passed to `density`. Use `args.dens = NA` if no density curve should be drawn. The defaults are taken from `density`.
...	further arguments are passed to the function `plot()`.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

plot, points, density, layout

Examples

```
# best seen with: x11(7.5, 4.7)

# just one variable with marginal densities
PlotMarDens( y=d.pizza$temperature, x=d.pizza$delivery_min, grp=1
             , xlab="delivery_min", ylab="temperature", col=SetAlpha("brown", 0.4)
             , pch=15, lwd=3
             , panel.first= grid(), args.legend=NA
             , main="Temp ~ delivery"
)

# use a group variable
PlotMarDens( y=d.pizza$temperature, x=d.pizza$delivery_min, grp=d.pizza$area
  , xlab="delivery_min", ylab="temperature", col=c("brown","orange","lightsteelblue")
  , panel.first=list( grid() )
  , main = "temperature ~ delivery_min | area"
)
# reset layout
par(mfrow=c(1,1))
```

PlotMiss *Plot Missing Data*

Description

Takes a data frame and displays the location of missing data. The missings can be clustered and be displayed together.

Usage

```
PlotMiss(x, col = DescTools::hred, bg = SetAlpha(DescTools::hecru, 0.3),
         clust = FALSE, main = NULL, ...)
```

Arguments

x	a data.frame to be analysed.
col	the colour of the missings.
bg	the background colour of the plot.
clust	logical, defining if the missings should be clustered. Default is FALSE.
main	the main title.
...	the dots are passed to plot.

Details

A graphical display of the position of the missings can be help to detect dependencies or patterns within the missings.

Value

if clust is set to TRUE, the new order will be returned invisibly.

Author(s)

Andri Signorell <andri@signorell.net>, following an idea of Henk Harmsen <henk@carbonmetrics.com>

See Also

`hclust`, `CountCompCases`

Examples

```
PlotMiss(d.pizza, main="Missing pizza data")
```

PlotMonth *Cycle Plot for Seasonal Effects of an Univariate Time Series*

Description

Plot seasonal effects of a univariate time series following Cleveland's definition for cycle plots.

Usage

```
PlotMonth(x, type = "l", labels, xlab = "", ylab = deparse(substitute(x)), ...)
```

Arguments

`x`	univariate time series
`type`	one out of `"l"` (line) or `"h"` (histogram), defines the plot type of the year components
`labels`	the labels for the cyclic component to be displayed on the x-axis
`xlab`	a title for the x axis: see `title`.
`ylab`	a title for the y axis: see `title`.
`...`	the dots are passed to the plot command.

Details

A cycle plot is a graphical method invented to study teh behaviour of a seasonal time series. The seasonal component of a univariate series is graphed. First the January values are graphed for successive years, then the February values and so forth. For each monthly subseries the mean of the values is portrayed by a horizontal line.

Author(s)

Markus Huerzeler (ETH Zurich), slight changes Andri Signorell <andri@signorell.net>

References

Cleveland, W. S. (1985) *The Elements of Graphing Data.* Monterey, CA: Wadsworth.

See Also

ts

Examples

```
PlotMonth(AirPassengers)
```

PlotMosaic *Mosaic Plots*

Description

Plots a mosaic on the current graphics device.

Usage

```
PlotMosaic(x, main = deparse(substitute(x)), horiz = TRUE, cols = NULL,
           off = 0.02, mar = NULL, xlab = NULL, ylab = NULL,
           cex = par("cex"), las = 2, ...)
```

Arguments

x	a contingency table in array form, with optional category labels specified in the dimnames(x) attribute. The table is best created by the table() command. So far only 2-way tables are allowed.
main	character string for the mosaic title.
horiz	logical, defining the orientation of the mosaicplot. TRUE (default) makes a horizontal plot.
cols	the colors of the plot.
off	the offset between the rectangles. Default is 0.02.
mar	the margin for the plot.
xlab, ylab	x- and y-axis labels used for the plot; by default, the first and second element of names(dimnames(X)) (i.e., the name of the first and second variable in X).
cex	numeric character expansion factor; multiplied by par("cex") yields the final character size. NULL and NA are equivalent to 1.0.
las	the style of axis labels. 0 - parallel to the axis, 1 - horizontal, 2 - perpendicular, 3 - vertical.
...	additional arguments are passed to the text function.

Details

The reason for this function to exist are the unsatisfying labels in base mosaicplot.

Value

list with the midpoints of the rectangles

Author(s)

Andri Signorell <andri@signorell.net>

References

Friendly, M. (1994) Mosaic displays for multi-way contingency tables. *Journal of the American Statistical Association*, **89**, 190-200.

See Also

mosaicplot

Examples

```
PlotMosaic(HairEyeColor[,,1])
```

PlotMultiDens *Plot Multiple Density Curves*

Description

Multiple density curves are plotted on the same plot. The function plots the density curves in the defined colors and linetypes, after having calculated the globally appropriate xlim- and ylim-values. A legend can directly be included.

Usage

```
PlotMultiDens(x, ...)

## Default S3 method:
PlotMultiDens(x, xlim = NULL, ylim = NULL, col = Pal(), lty = "solid",
              lwd = 2, fill = NA, xlab = "x", ylab = "density", args.dens = NULL,
               args.legend = NULL, na.rm = FALSE, flipxy = FALSE, ...)

## S3 method for class 'formula'
PlotMultiDens(formula, data, subset, na.action, ...)
```

Arguments

x	a list of vectors whose densities are to be plotted. Use split to separate a vector by groups. (See examples)
xlim, ylim	xlim, ylim of the plot.
col	colors of the lines, defaults to Pal(), returning the default palette.
lty	line type of the lines.
lwd	line widths for the lines.
fill	colors for fill the area under the density curve. If set to NA (default) there will be no color.
xlab, ylab	a title for the x, resp. y axis. Defaults to "x" and "density".
args.dens	list of additional arguments to be passed to the density function. If set to NULL the defaults will be used. Those are n = 4096 (2^12) and kernel = "epanechnikov".
args.legend	list of additional arguments to be passed to the legend function. Use args.legend = NA if no legend should be added.
na.rm	should NAs be omitted? Defaults to FALSE.
flipxy	logical, should x- and y-axis be flipped? Defaults to FALSE.
formula	a formula of the form lhs ~ rhs where lhs gives the data values and rhs the corresponding groups.
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
...	the dots are passed to plot(...).

Details

All style arguments, density arguments and data list elements will be recycled if necessary.
The argument flipxy leads to exchanged x- and y-values. This option can be used to plot density curves with a vertical orientation for displaying marginal densities.

Value

data.frame with 3 columns, containing the bw, n and kernel parameters used for the list elements. The number of rows correspond to the length of the list x.

Note

Consider using:

```
library(lattice)
densityplot( ~ delivery_min | driver, data=d.pizza)
```

as alternative when not all curves should be plotted in the same plot.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

PlotViolin, density

Examples

```
x <- rnorm(1000,0,1)
y <- rnorm(1000,0,2)
z <- rnorm(1000,2,1.5)

# the input of the following function MUST be a numeric list
PlotMultiDens(list(x=x,y=y,z=z))

# use area fill
PlotMultiDens(list(x=x,y=y,z=z), fill=SetAlpha(c("red","green","blue"), 0.4))


PlotMultiDens( x=split(d.pizza$delivery_min, d.pizza$driver), na.rm=TRUE
  , main="delivery time ~ driver", xlab="delivery time [min]", ylab="density"
  , lwd=1:7, lty=1:7
  , panel.first=grid())
# this example demonstrates the definition of different line types and -colors
# an is NOT thought as recommendation for good plotting practice... :-)


# the formula interface
PlotMultiDens(delivery_min ~ driver, data=d.pizza)

# recyling of the density parameters
res <- PlotMultiDens(x=split(d.pizza$temperature, d.pizza$driver),
             args.dens = list(bw=c(5,2), kernel=c("rect","epanechnikov")), na.rm=TRUE)
res

# compare bandwidths
PlotMultiDens(x=split(d.pizza$temperature, d.pizza$driver)[1],
                    args.dens = list(bw=c(1:5)), na.rm=TRUE,
                    args.legend=NA, main="Compare bw")
legend(x="topright", legend=gettextf("bw = %s", 1:5), fill=rainbow(5))
```

PlotPairs *Extended Scatterplot Matrices*

Description

A matrix of scatterplots is produced.The upper triangular matrices contain nothing else than the correlation coefficient. The diagonal displays a histogram of the variable. The lower triangular matrix

displays a scatterplot superposed by a smoother. It's possible to define groups to be differntiated by color and also by individual smoothers. The used code is not much more than the pairs() code and some examples, but condenses it to a practical amount.

Usage

```
PlotPairs(x, g = NULL, col = 1, pch = 19, col.smooth = 1, main = "",
          upper = FALSE, ...)
```

Arguments

x	the coordinates of points given as numeric columns of a matrix or data frame. Logical and factor columns are converted to numeric in the same way that data.matrix does. Will directly be passed on to pairs.
g	a group variable
col	color for pointcharacter
pch	point character
col.smooth	color for the smoother(s)
main	the main title
upper	logical, determines if the correlation coefficients should be displayed in the upper triangular matrix (default) or in the lower one.
...	additional arguments passed to pairs function.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

pairs

Examples

```
PlotPairs(iris[1:4], g=iris$Species, main = "Anderson's Iris Data -- 3 species",
          col=c(DescTools::hred, DescTools::hblue, DescTools::hgreen),
          col.smooth=c("black", DescTools::hred, DescTools::hblue, DescTools::hgreen))
```

PlotPolar *Plot Values on a Circular Grid*

Description

PlotPolar creates a polar coordinate plot of the radius r in function of the angle theta. 0 degrees is drawn at the 3 o'clock position and angular values increase in a counterclockwise direction.

Usage

```
PlotPolar(r, theta = NULL, type = "p", rlim = NULL, main = "", lwd = par("lwd"),
          lty = par("lty"), col = par("col"), pch = par("pch"), fill = NA,
          cex = par("cex"), mar = c(2, 2, 5, 2), add = FALSE, ...)
```

Arguments

r	a vector of radial data.
theta	a vector of angular data specified in radians.
type	one out of c("p","l","h"), the plot type, defined following the definition in plot type. "p" means points, "l" will connect the points with lines and "h" is used to plot radial lines from the center to the points. Default is "p".
rlim	the r limits (r1, r2) of the plot
main	a main title for the plot, see also title.
lwd	a vector of line widths, see par.
lty	a vector of line types, see par.
col	The colors for lines and points. Multiple colors can be specified so that each point can be given its own color. If there are fewer colors than points they are recycled in the standard fashion. Lines will all be plotted in the first colour specified.
pch	a vector of plotting characters or symbols: see points.
fill	fill color, defaults to NA (none).
cex	a numerical vector giving the amount by which plotting characters and symbols should be scaled relative to the default. This works as a multiple of par("cex"). NULL and NA are equivalent to 1.0.
mar	A numerical vector of the form c(bottom, left, top, right) which gives the number of lines of margin to be specified on the four sides of the plot.
add	defines whether points should be added to an existing plot.
...	further arguments are passed to the plot command.

Details

The function is rather flexible and can produce quite a lot of of different plots. So is it also possible to create spider webs or radar plots.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

PolarGrid

Examples

```
testlen <- c(sin(seq(0, 1.98*pi, length=100))+2+rnorm(100)/10)
testpos <- seq(0, 1.98*pi, length=100)

PlotPolar(testlen, testpos, type="l", main="Test Polygon", col="blue")
PolarGrid(ntheta=9, col="grey", lty="solid", lblradians=TRUE)

# start at 12 o'clock and plot clockwise
PlotPolar(testlen, -(testpos - pi/2), type="p", main="Test Polygon",
          col="green", pch=16)

PolarGrid(ntheta = rev(seq(0, 2*pi, by=2*pi/9) + pi/2),
          alabels=Format(seq(0, 2*pi, by=2*pi/9), digits=2)[-10], col="grey",
          lty="solid", lblradians=TRUE)


# just because of it's beauty
t <- seq(0,2*pi,0.01)
PlotPolar( r=sin(2*t)*cos(2*t), theta=t, type="l", lty="dashed", col="red" )
PolarGrid()


# use some filled polygons
ions <- c(3.2,5,1,3.1,2.1,5)
ion.names <- c("Na","Ca","Mg","Cl","HCO3","SO4")

PlotPolar(r = ions, type="l", fill="yellow")

# the same, but let's have a grid first
PlotPolar(r = ions, type="l", lwd=2, col="blue", main="Ions",
          panel.first=PolarGrid(nr=seq(0, 6, 1)) )

# leave the radial grid out
PlotPolar(r = ions, type="l", fill="yellow")
PolarGrid(nr = NA, ntheta = length(ions), alabels = ion.names,
          col = "grey", lty = "solid" )

# display radial lines
PlotPolar(r = ions, type="h", col="blue", lwd=3)
# add some points
PlotPolar(r = ions, type="p", pch=16, add=TRUE, col="red", cex=1.5)

# spiderweb (not really recommended...)
posmat <- matrix(sample(2:9,30,TRUE),nrow=3)
PlotPolar(posmat, type="l", main="Spiderweb plot", col=2:4, lwd=1:3)
PolarGrid(nr=NA, ntheta=ncol(posmat), alabels=paste("X", 1:ncol(posmat), sep=""),
          col="grey", lty="solid" )

# example from: The grammar of graphics (L. Wilkinson)
data("UKgas")
m <- matrix(UKgas, ncol=4, byrow=TRUE)
cols <- c(SetAlpha(rep("green", 10), seq(0,1,0.1)),
```

```
          SetAlpha(rep("blue", 10), seq(0,1,0.1)),
          SetAlpha(rep("orange", 10), seq(0,1,0.1)))

PlotPolar(r=m, type="l", col=cols, lwd=2 )
PolarGrid(ntheta=4, alabels=c("Winter","Spring","Summer","Autumn"), lty="solid")
legend(x="topright", legend=c(1960,1970,1980), fill=c("green","blue","orange"))


# radarplot (same here, consider alternatives...)
data(mtcars)
d.car <- scale(mtcars[1:6,1:7], center=FALSE)

# let's have a palette with transparent colors (alpha = 32)
cols <- SetAlpha(colorRampPalette(c("red","yellow","blue"), space = "rgb")(6), 0.25)
PlotPolar(d.car, type="l", fill=cols, main="Cars in radar")
PolarGrid(nr=NA, ntheta=ncol(d.car), alabels=colnames(d.car), lty="solid", col="black")

# a polar barplot
x <- c(4,8,2,8,2,6,5,7,3,3,5,3)
theta <- (0:12) * pi / 6
PlotPolar(x, type = "n", main="Some data")
PolarGrid(nr = 0:9, ntheta = 24, col="grey", lty=1, rlabels = NA, alabels = NA)
DrawCircle(x=0, y=0, r.in=0, r.out=x,
                 theta.1 = theta[-length(theta)], theta.2 = theta[-1],
                 col=SetAlpha(rainbow(12), 0.7), border=NA)

segments(x0 = -10:10, y0 = -.2, y1=0.2)
segments(x0=-10, x1=10, y0 = 0)

segments(y0 = -10:10, x0 = -.2, x1=0.2)
segments(y0=-10, y1=10, x0 = 0)

BoxedText(x=0, y=c(0,3,6,9), labels = c(0,3,6,9), xpad = .3, ypad=.3, border="grey35")

# USJudgeRatings
PlotPolar(USJudgeRatings[1,], type="l", col=DescTools::hblue, lwd=2, cex=0.8,
          panel.first=PolarGrid(ntheta=ncol(USJudgeRatings), col="grey", lty="solid",
                                las=1, alabels=colnames(USJudgeRatings), lblradians=TRUE))
PlotPolar(USJudgeRatings[2,], type="l", col=DescTools::hred, lwd=2, add=TRUE)
PlotPolar(USJudgeRatings[5,], type="l", col=DescTools::horange, lwd=2, add=TRUE)

legend(x="topright", inset=-0.18,
       col = c(DescTools::hblue, DescTools::hred, DescTools::horange), lwd=2,
       legend=rownames(USJudgeRatings)[c(1, 2, 5)])
```

PlotProbDist *Plot Probability Distribution*

Description

Produce a plot from a probability distribution with shaded areas. This is often needed in theory texts for classes in statistics.

Usage

```
PlotProbDist(breaks, FUN,
             blab = NULL, main = "", xlim = NULL, col = NULL, density = 7,
             alab = LETTERS[1:(length(breaks) - 1)],
             alab_x = NULL, alab_y = NULL, ylab = "density", ...)
```

Arguments

breaks	a numeric vector containing the breaks of different areas. The start and end must not be infinity.
FUN	the (typically) distribution function
blab	text for labelling the breaks
main	main title for the plot
xlim	the x-limits for the plot
col	the color for the shaded areas
density	the density for the shaded areas
alab	the labels for areas
alab_x	the x-coord for the area labels
alab_y	the y-coord for the area labels, if left to default they will be placed in the middle of the plot
ylab	the label for they y-axis
...	further parameters passed to internally used function curve()

Details

The function sets up a two-step plot procedure based on curve() and Shade() with additional labelling for convenience.

Value

nothing returned

Author(s)

Andri Signorell <andri@signorell.net>

See Also

Shade, curve, polygon

Examples

```
# plot t-distribution
PlotProbDist(breaks=c(-6, -2.3, 1.5, 6),
             function(x) dt(x, df=8),
             blab=c("A","B"), xlim=c(-4,4), alab=NA,
             main="t-Distribution (df=8)",
             col=c(DescTools::hred, DescTools::hblue, DescTools::horange),
             density=c(20, 7))

# Normal
PlotProbDist(breaks=c(-10, -1, 12),
             function(x) dnorm(x, mean=2, sd=2),
             blab="A", xlim=c(-7,10),
             main="Normal-Distribution N(2,2)",
             col=c(DescTools::hred, DescTools::hblue), density=c(20, 7))

# same for Chi-square
PlotProbDist(breaks=c(0, 15, 35),
             function(x) dchisq(x, df=8),
             blab="B", xlim=c(0, 30),
             main=expression(paste(chi^2-Distribution, " (df=8)")),
             col=c(DescTools::hblue, DescTools::hred), density=c(0, 20))
```

PlotPyramid *Draw a Back To Back Pyramid Plot*

Description

Pyramid plots are a common way to display the distribution of age groups.

Usage

```
PlotPyramid(lx, rx = NA, ylab = "", ylab.x = 0,
            col = c("red", "blue"), border = par("fg"),
            main = "", lxlab = "", rxlab = "",
            xlim = NULL, gapwidth = NULL,
            xaxt = TRUE, args.grid = NULL, cex.axis = par("cex.axis"),
            cex.lab = par("cex.axis"), cex.names = par("cex.axis"),
            adj = 0.5, rev = FALSE, ...)
```

Arguments

lx	either a vector or matrix of values describing the bars which make up the plot. If lx is a vector, it will be used to construct the left barplot. If lx is a matrix the first column will be plotted to the left side and the second to the right side. Other columsn are ignored.

rx	a vector with the values used to build the right barplot. lx and rx should be of equal length.
ylab	a vector of names to be plotted either in the middle or at the left side of the plot. If this argument is omitted, then the names are taken from the names attribute of lx if this is a vector.
ylab.x	the x-position of the y-labels.
col	the color(s) of the bars. If there are more than one the colors will be recycled.
border	the border color of the bars. Set this to NA if no border is to be plotted.
main	overall title for the plot.
lxlab	a label for the left x axis.
rxlab	a label for the right x axis.
xlim	limits for the x axis. The first value will determine the limit on the left, the second the one on the right.
gapwidth	the width of a gap in the middle of the plot. If set to 0, no gap will be plotted. Default is NULL which will make the gap as wide, as it is necessary to plot the longest ylab.
xaxt	a character which specifies the x axis type. Specifying "n" suppresses plotting of the axis.
args.grid	list of additional arguments for the grid. Set this argument to NA if no grid should be drawn.
cex.axis	expansion factor for numeric axis labels.
cex.lab	expansion factor for numeric variable labels.
cex.names	expansion factor for y labels (names).
adj	one or two values in [0, 1] which specify the x (and optionally y) adjustment of the labels.
rev	logical, if set to TRUE the order of data series and names will be reversed.
...	the dots are passed to the barplot function.

Details

Pyramid plots are a common way to display the distribution of age groups in a human population. The percentages of people within a given age category are arranged in a barplot, typically back to back. Such displays can be used to distinguish males vs. females, differences between two different countries or the distribution of age at different timepoints. The plot type can also be used to display other types of opposed bar charts with suitable modification of the arguments.

Value

A numeric vector giving the coordinates of all the bar midpoints drawn, useful for adding to the graph.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

barplot

Examples

```
d.sda <- data.frame(
  kt_x =  c("ZH","BL","ZG","SG","LU","AR","SO","GL","SZ",
            "NW","TG","UR","AI","OW","GR","BE","SH","AG",
            "BS","FR","GE","JU","NE","TI","VD","VS"),
  apo_n = c(18,16,13,11,9,12,11,8,9,8,11,9,7,9,24,19,
            19,20,43,27,41,31,37,62,38,39),
  sda_n = c(235,209,200,169,166,164,162,146,128,127,
            125,121,121,110,48,34,33,0,0,0,0,0,0,0,0,0)
)

PlotPyramid(lx=d.sda[,c("apo_n","sda_n")], ylab=d.sda$kt_x,
            col=c("lightslategray", "orange2"), border = NA, ylab.x=0,
            xlim=c(-110,250),
            gapwidth = NULL, cex.lab = 0.8, cex.axis=0.8, xaxt = TRUE,
            lxlab="Drugstores", rxlab="General practitioners",
            main="Density of general practitioners and drugstores in CH (2010)",
            space=0.5, args.grid=list(lty=1))


par(mfrow=c(1,3))

m.pop<-c(3.2,3.5,3.6,3.6,3.5,3.5,3.9,3.7,3.9,3.5,
         3.2,2.8,2.2,1.8,1.5,1.3,0.7,0.4)
f.pop<-c(3.2,3.4,3.5,3.5,3.5,3.7,4,3.8,3.9,3.6,3.2,
         2.5,2,1.7,1.5,1.3,1,0.8)
age <- c("0-4","5-9","10-14","15-19","20-24","25-29",
         "30-34","35-39","40-44","45-49","50-54",
         "55-59","60-64","65-69","70-74","75-79","80-44","85+")

PlotPyramid(m.pop, f.pop,
            ylab = age, space = 0, col = c("cornflowerblue", "indianred"),
            main="Age distribution at baseline of HELP study",
            lxlab="male", rxlab="female" )

PlotPyramid(m.pop, f.pop,
            ylab = age, space = 0, col = c("cornflowerblue", "indianred"),
            xlim=c(-5,5),
            main="Age distribution at baseline of HELP study",
            lxlab="male", rxlab="female", gapwidth=0, ylab.x=-5 )


PlotPyramid(c(1,3,5,2,0.5), c(2,4,6,1,0),
            ylab = LETTERS[1:5], space = 0.3, col = rep(rainbow(5), each=2),
            xlim=c(-10,10), args.grid=NA, cex.names=1.5, adj=1,
            lxlab="Group A", rxlab="Group B", gapwidth=0, ylab.x=-8, xaxt="n")
```

`PlotQQ` *QQ-Plot for Any Distribution*

Description

Create a QQ-plot for a variable of any distribution. The assumed underlying distribution can be defined as a function of f(p), including all required parameters. Confidence bands are provided by default.

Usage

```
PlotQQ(x, qdist=qnorm, main = NULL, xlab = NULL, ylab = NULL, datax = FALSE, add = FALSE,
       args.qqline = NULL, conf.level = 0.95, args.cband = NULL, ...)
```

Arguments

`x`	the data sample
`qdist`	the quantile function of the assumed distribution. Can either be given as simple function name or defined as own function using the required arguments. Default is `qnorm()`. See examples.
`main`	the main title for the plot. This will be "Q-Q-Plot" by default
`xlab`	the xlab for the plot
`ylab`	the ylab for the plot
`datax`	logical. Should data values be on the x-axis? Default is `FALSE`.
`add`	logical specifying if the points should be added to an already existing plot; defaults to `FALSE`.
`args.qqline`	arguments for the qqline. This will be estimated as a line through the 25% and 75% quantiles by default, which is the same procedure as `qqline()` does for normal distribution (instead of set it to `abline(a = 0, b = 1)`). The quantiles can however be overwritten by setting the argument `probs` to some user defined values. Also the method for calculating the quantiles can be defined (default is 7, see `quantile`). The line defaults are set to `col = par("fg")`, `lwd = par("lwd")` and `lty = par("lty")`. No line will be plotted if `args.qqline` is set to `NA`.
`conf.level`	confidence level for the confidence interval. Set this to `NA`, if no confidence band should be plotted. Default is `0.95`. The confidence intervals are calculated pointwise method based on a Kolmogorov-Smirnov distribution.
`args.cband`	list of arguments for the confidence band, such as color or border (see `DrawBand`).
`...`	the dots are passed to the plot function.

Details

The function generates a sequence of points between 0 and 1 and transforms those into quantiles by means of the defined assumed distribution.

Note

The code is inspired by the tip 10.22 "Creating other Quantile-Quantile plots" from R Cookbook and based on R-Core code from the function `qqline`. The calculation of confidence bands are rewritten based on an algorithm published in the package `BoutrosLab.plotting.general`.

Author(s)

Andri Signorell <andri@signorell.net>, Ying Wu <Ying.Wu@stevens.edu>

References

Teetor, P. (2011) *R Cookbook*. O'Reilly, pp. 254-255.

See Also

`qqnorm`, `qqline`, `qqplot`

Examples

```
y <- rexp(100, 1/10)
PlotQQ(y, function(p) qexp(p, rate=1/10))

w <- rweibull(100, shape=2)
PlotQQ(w, qdist = function(p) qweibull(p, shape=4))

z <- rchisq(100, df=5)
PlotQQ(z, function(p) qchisq(p, df=5),
       args.qqline=list(col=2, probs=c(0.1, 0.6)),
       main=expression("Q-Q plot for" ~~ {chi^2}[nu == 3]))
abline(0,1)

# add 5 random sets
for(i in 1:5){
  z <- rchisq(100, df=5)
  PlotQQ(z, function(p) qchisq(p, df=5), add=TRUE, args.qqline = NA,
         col="grey", lty="dotted")
}
```

PlotTernary *Ternary or Triangular Plots*

Description

`PlotTernary` plots in a triangle the values of three variables. Useful for mixtures (chemistry etc.).

Usage

```
PlotTernary(x, y = NULL, z = NULL, args.grid = NULL, lbl = NULL, main = "",...)
```

Arguments

x	vector of first variable. Will be placed on top of the triangle.
y	vector of second variable (the right corner).
z	vector of third variable (on the left corner).
args.grid	list of additional arguments for the grid. Set this argument to NA if no grid should be drawn. The usual color and linetype will be used.
main	overall title for the plot.
lbl	the labels for the corner points. Default to the names of x, y, z.
...	the dots are sent to points

Author(s)

Andri Signorell <andri@signorell.net> based on example code by W. N. Venables and B. D. Ripley mentioned

References

J. Aitchison (1986) *The Statistical Analysis of Compositional Data.* Chapman and Hall, p.360.

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S.* Fourth edition. Springer.

See Also

example in Skye

Examples

```
# some random data in three variables
c1 <- runif(25)
c2 <- runif(25)
c3 <- runif(25)

# basic plot
par(mfrow=c(1, 2))
PlotTernary(c1, c2, c3, args.grid=NA)

## Not run:
# plot with different symbols and a grid using a dataset from MASS
data(Skye, package="MASS")

PlotTernary(Skye[c(1,3,2)], pch=15, col=DescTools::hred, main="Skye",
            lbl=c("A Sodium", "F Iron", "M Magnesium"))

## End(Not run)
```

PlotTreemap *Create a Treemap*

Description

Creates a treemap where rectangular regions of different size, color, and groupings visualize the elements.

Usage

```
PlotTreemap(x, grp = NULL, labels = NULL, cex = 1, text.col = "black",
            col = rainbow(length(x)), labels.grp = NULL, cex.grp = 3,
            text.col.grp = "black", border.grp = "grey50",
            lwd.grp = 5, main = "")
```

Arguments

x	a vector storing the values to be used to calculate the areas of rectangles.
grp	a vector specifying the group (i.e. country, sector, etc.) to which each element belongs.
labels	a vector specifying the labels.
cex	the character extension for the area labels. Default is 1.
text.col	the text color of the area labels. Default is "black".
col	a vector storing the values to be used to calculate the color of rectangles.
labels.grp	a character vector specifying the labels for the groups.
cex.grp	the character extension for the group labels. Default is 3.
text.col.grp	the text color of the group labels. Default is "black".
border.grp	the border color for the group rectangles. Default is "grey50". Set this to NA if no special border is desired.
lwd.grp	the linewidth of the group borders. Default is 5.
main	a title for the plot.

Details

A treemap is a two-dimensional visualization for quickly analyzing large, hierarchical data sets. Treemaps are unique among visualizations because they provide users with the ability to see both a high level overview of data as well as fine-grained details. Users can find outliers, notice trends, and perform comparisons using treemaps. Each data element contained in a treemap is represented with a rectangle, or a cell. Treemap cell arrangement, size, and color are each mapped to an attribute of that element. Treemap cells can be grouped by common attributes. Within a group, larger cells are placed towards the bottom left, and smaller cells are placed at the top right.

Value

returns a list with groupwise organized midpoints in x and y for the rectangles within a group and for the groups themselves.

Author(s)

Andri Signorell <andri@signorell.net>, strongly based on code from Jeff Enos <jeff@kanecap.com>

See Also

PlotCirc, mosaicplot, barplot

Examples

```
set.seed(1789)
N <- 20
area <- rlnorm(N)

PlotTreemap(x=sort(area, decreasing=TRUE), labels=letters[1:20], col=Pal("RedToBlack", 20))


grp <- sample(x=1:3, size=20, replace=TRUE, prob=c(0.2,0.3,0.5))

z <- Sort(data.frame(area=area, grp=grp), c("grp","area"), decreasing=c(FALSE,TRUE))
z$col <- SetAlpha(c("steelblue","green","yellow")[z$grp],
                  unlist(lapply(split(z$area, z$grp),
                  function(...) LinScale(..., newlow=0.1, newhigh=0.6))))

PlotTreemap(x=z$area, grp=z$grp, labels=letters[1:20], col=z$col)


b <- PlotTreemap(x=z$area, grp=z$grp, labels=letters[1:20], labels.grp=NA,
                 col=z$col, main="Treemap")

# the function returns the midpoints of the areas
# extract the group midpoints from b
mid <- do.call(rbind, lapply(lapply(b, "[", 1), data.frame))

# and draw some visible text
BoxedText( x=mid$grp.x, y=mid$grp.y, labels=LETTERS[1:3], cex=3, border=NA,
  col=SetAlpha("white",0.7) )
```

PlotVenn *Plot a Venn Diagram*

Description

This function produces Venn diagrams for up to 5 datasets.

Usage

```
PlotVenn(x, col = "transparent", plotit = TRUE, labels = NULL)
```

Arguments

x	the list with the sets to be analysed. Those can be factors or something coercable to a factor.
col	the colors for the sets on the plot.
plotit	logical. Should a plot be produced or just the results be calculated.
labels	special labels for the plot. By default the names of the list x will be used. If those are missing, the LETTERS A..E will be chosen. Set this argument to NA, if no labels at all should be plotted.

Details

The function calculates the necessary frequencies and plots the venn diagram.

Value

a list with 2 elements, the first contains a table with the observed frequencies in the given sets. The second returns a data.frame with the xy coordinates for the labels in the venn diagram, the specific combination of factors and the frequency in that intersection area. The latter can be 0 as well.

Author(s)

Andri Signorell <andri@signorell.net>

References

Venn, J. (1880): On the Diagrammatic and Mechanical Representation of Propositions and Reasonings. *Dublin Philosophical Magazine and Journal of Science* 9 (59): 1-18.

Edwards, A.W.F. (2004): Cogwheels of the mind: the story of Venn diagrams. *JHU Press* ISBN 978-0-8018-7434-5.

Examples

```
element <- function() paste(sample(LETTERS, 5, replace=TRUE), collapse="")
group <- replicate(1000, element())

GroupA <- sample(group, 400, replace=FALSE)
GroupB <- sample(group, 750, replace=FALSE)
GroupC <- sample(group, 250, replace=FALSE)
GroupD <- sample(group, 300, replace=FALSE)

x <- list(GroupA, GroupB, GroupC, GroupD)
x

PlotVenn(x=list(GroupA, GroupB))
PlotVenn(x=list(Set_1=GroupA, Set_2=GroupB))
```

```
PlotVenn(x=list(GroupA, GroupB), labels=c("English","Spanish"))

PlotVenn(x=x[1:3])
PlotVenn(x=x[1:4], col=SetAlpha(c("blue","red","yellow","green","lightblue"), 0.2))

r.venn <- PlotVenn(x=x[1:5], col=SetAlpha(c("blue","red","yellow","green","lightblue"), 0.2))
r.venn
```

`PlotViolin` *Plot Violins Instead of Boxplots*

Description

This function serves the same utility as side-by-side boxplots, only it provides more detail about the different distribution. It plots violins instead of boxplots. That is, instead of a box, it uses the density function to plot the density. For skewed distributions, the results look like "violins". Hence the name.

Usage

```
PlotViolin(x, ...)

## Default S3 method:
PlotViolin(x, ..., horizontal = FALSE, bw = "SJ", na.rm = FALSE,
           names = NULL, args.boxplot = NULL)

## S3 method for class 'formula'
PlotViolin(formula, data, subset, na.action, ...)
```

Arguments

`x`	Either a sequence of variable names, or a data frame, or a model formula
`horizontal`	logical indicating if the densityplots should be horizontal; default `FALSE` means vertical arrangement.
`bw`	the smoothing bandwidth (method) being used by `density`. `bw` can also be a character string giving a rule to choose the bandwidth. See `bw.nrd`. The default, has been switched from `"nrd0"` to `"SJ"`, following the general recommendation in Venables & Ripley (2002). In case of a method, the average computed bandwidth is used.
`na.rm`	logical, should NAs be omitted? The density-function can't do with missings. Defaults to `FALSE`.
`names`	a vector of names for the groups.
`formula`	a formula, such as y ~ grp, where y is a numeric vector of data values to be split into groups according to the grouping variable grp (usually a factor).
`data`	a data.frame (or list) from which the variables in formula should be taken.
`subset`	an optional vector specifying a subset of observations to be used for plotting.

na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
...	The dots are passed to polygon. Notably, you can set the color to red with col="red", and a border color with border="blue"
args.boxplot	list of arguments for a boxplot to be superposed to the densityplot. By default (NULL) a black boxplot will be drawn. Set this to NA to suppress the boxplot.

Value

If a boxplot was drawn then the function returns a list with the following components:

stats	a matrix, each column contains the extreme of the lower whisker, the lower hinge, the median, the upper hinge and the extreme of the upper whisker for one group/plot. If all the inputs have the same class attribute, so will this component.
n	a vector with the number of observations in each group.
conf	a matrix where each column contains the lower and upper extremes of the notch.
out	the values of any data points which lie beyond the extremes of the whiskers.
group	a vector of the same length as out whose elements indicate to which group the outlier belongs.
names	a vector of names for the groups.

Note

This function is based on violinplot (package **UsingR**). Some adaptions were made in the interface, such as to accept the same arguments as boxplot does. Moreover the function was extended by the option to have a boxplot superposed.

Author(s)

John Verzani, Andri Signorell <andri@signorell.net>

References

The code is based on the boxplot function from R/base.

See Also

boxplot, PlotMultiDens , density

Examples

```
# make a "violin"
x <- c(rnorm(100), rnorm(50,5))

PlotViolin(x, col = "brown")

par(mfrow=c(1,2))
f <- factor(rep(1:5, 30))
# make a quintet. Note also choice of bandwidth
```

```
PlotViolin(x ~ f, col = SetAlpha("steelblue",0.3), bw = "SJ", main="Vertical")

# and the same, but in horizontal arrangement
PlotViolin(x ~ f, col = SetAlpha("steelblue",0.3), bw = "SJ", horizontal = TRUE,
  las=1, main="Horizontal")


# example taken from boxplot
boxplot(count ~ spray, data = InsectSprays, col = "lightgray", main="Boxplot")
PlotViolin(count ~ spray, data = InsectSprays, col = "lightgray", main="Violinplot")


# groupwise densityplots defined the same way as in boxplot
boxplot(len ~ supp*dose, data = ToothGrowth,
        main = "Guinea Pigs' Tooth Growth",
        xlab = "Vitamin C dose mg", ylab = "tooth length",
        col=c("yellow", "orange"), lty=c(1,2)
)

b <- PlotViolin(len ~ supp*dose, data = ToothGrowth,
           main = "Guinea Pigs' Tooth Growth",
           xlab = "Vitamin C dose mg", ylab = "tooth length",
           col=c("yellow", "orange"), lty=c(1,2)
)
# use points, if the medians deserve special attention
points(x=1:6, y=b$stats[3,], pch=21, bg="white", col="black", cex=1.2)
```

`PlotWeb` *Plot a Web of Connected Points*

Description

This plot can be used to graphically display a correlation matrix by using the linewidth between the nodes in proportion to the correlation of two variables. It will place the elements homogenously around a circle and draw connecting lines between the points.

Usage

```
PlotWeb(m, col = c(DescTools::hred, DescTools::hblue), lty = NULL,
        lwd = NULL, args.legend=NULL,
        pch = 21, pt.cex = 2, pt.col = "black", pt.bg = "darkgrey",
        cex.lab = 1, las = 1, adj = NULL, dist = 0.5, ...)
```

Arguments

`m`	a symmetric matrix of numeric values
`col`	the color for the connecting lines
`lty`	the line type for the connecting lines, the default will be `par("lty")`.

lwd	the line widths for the connecting lines. If left to NULL it will be linearly scaled between the minimum and maximum value of m.
args.legend	list of additional arguments to be passed to the legend function. Use args.legend = NA if no legend should be added.
pch	the plotting symbols appearing in the plot, as a non-negative numeric vector (see points, but unlike there negative values are omitted) or a vector of 1-character strings, or one multi-character string.
pt.cex	expansion factor(s) for the points.
pt.col	the foreground color for the points, corresponding to its argument col.
pt.bg	the background color for the points, corresponding to its argument bg.
las	alignment of the labels, 1 means horizontal, 2 radial and 3 vertical.
adj	adjustments for the labels. (Left: 0, Right: 1, Mid: 0.5)
dist	gives the distance of the labels from the outer circle. Default is 2.
cex.lab	the character extension for the labels.
...	dots are passed to plot.

Details

The function uses the lower triangular matrix of m, so this is the order colors, linewidth etc. must be given, when the defaults are to be overrun.

Value

A list of x and y coordinates, giving the coordinates of all the points drawn, useful for adding other elements to the plot.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

PlotCorr

Examples

```
m <- cor(d.pizza[, which(sapply(d.pizza, IsNumeric, na.rm=TRUE))[-c(1:2)]],
         use="pairwise.complete.obs")
PlotWeb(m=m, col=c(DescTools::hred, DescTools::hblue), main="Pizza Correlation")


# let's describe only the significant corrs and start with a dataset
d.m <- d.pizza[, which(sapply(d.pizza, IsNumeric, na.rm=TRUE))[-c(1:2)]]

# get the correlation matrix
m <- cor(d.m, use="pairwise.complete.obs")
```

```
# let's get rid of all non significant correlations
ctest <- PairApply(d.m,  function(x, y) cor.test(x, y)$p.value, symmetric=TRUE)

# ok, got all the p-values, now replace > 0.05 with NAs
m[ctest > 0.05] <- NA
# How does that look like now?
Format(m, na.form = ". ", ldigits=0, digits=3, align = "right")

PlotWeb(m, las=2, cex=1.2)

# define line widths
PlotWeb(m, lwd=abs(m[lower.tri(m)] * 10))
```

PMT *Periodic Payment of an Annuity.*

Description

PMT computes the periodic payment of an annuity. IPMT calculates what portion of a period payment is going towards interest in a particular period and PPMT what portion of a period payment is going towards principal in a particular period. RBAL yields the remaining balance in a particular period.

Usage

```
PMT(rate, nper, pv, fv = 0, type = 0)
IPMT(rate, per, nper, pv, fv = 0, type = 0)
PPMT(rate, per, nper, pv, fv = 0, type = 0)
RBAL(rate, per, nper, pv, fv = 0, type = 0)
```

Arguments

rate	specifies the interest rate.
per	specifies the period of the payment to be applied to interest or to principal.
nper	specifies the number of payment periods.
pv	specifies the present value or the lump-sum amount that a series of future payments is worth currently. pv can be 0 if a positive fv argument is included.
fv	specifies the future value or a cash balance that you want to attain after the last payment is made. Default is 0.
type	specifies the number 0 or 1 and indicates when payments are due. Default is 0.

Value

a numeric value

Author(s)

Andri Signorell <andri@signorell.net>

See Also

NPV, SLN

Examples

```
# original principal:    20'000
# loan term (years):     5
# annual interest rate:  8%
# annual payment:        -4'156.847

# simple amortization schedule
cbind(
  year      = 1:5,
  payment   = PMT(rate=0.08, nper=5, pv=20000, fv=-5000, type=0),
  interest  = IPMT(rate=0.08, per=1:5, nper=5, pv=20000, fv=-5000, type=0),
  principal = PPMT(rate=0.08, per=1:5, nper=5, pv=20000, fv=-5000, type=0),
  balance   = RBAL(rate=0.08, per=1:5, nper=5, pv=20000, fv=-5000, type=0)
)

#     year   payment   interest principal   balance
# [1,]    1 -4156.847 -1600.0000 -2556.847 17443.153
# [2,]    2 -4156.847 -1395.4523 -2761.395 14681.759
# [3,]    3 -4156.847 -1174.5407 -2982.306 11699.452
# [4,]    4 -4156.847  -935.9562 -3220.891  8478.562
# [5,]    5 -4156.847  -678.2849 -3478.562  5000.000
```

PoissonCI *Poisson Confidence Interval*

Description

Computes the confidence intervals of a poisson distributed variable's lambda. Several methods are implemented, see details.

Usage

```
PoissonCI(x, n = 1, conf.level = 0.95, sides = c("two.sided","left","right"),
          method = c("exact", "score", "wald", "byar"))
```

Arguments

x	number of events.
n	time base for event count.
conf.level	confidence level, defaults to 0.95.

sides	a character string specifying the side of the confidence interval, must be one of "two.sided" (default), "left" or "right". You can specify just the initial letter. "left" would be analogue to a hypothesis of "greater" in a t.test.
method	character string specifing which method to use; can be one out of "wald", "score", "exact" or "byar". Method can be abbreviated. See details. Defaults to "score".

Details

The Wald interval uses the asymptotic normality of the test statistic.

Byar's method is quite a good approximation. Rothman and Boice (1979) mention that these limits were first proposed by Byar (unpublished).

Value

A vector with 3 elements for estimate, lower confidence intervall and upper for the upper one.

Author(s)

Andri Signorell <andri@signorell.net>

References

Agresti, A. and Coull, B.A. (1998) Approximate is better than "exact" for interval estimation of binomial proportions. *American Statistician*, **52**, pp. 119-126.

Rothman KJ, Boice JD, Jr. (1979) Epidemiologic Analysis with a Programmable Calculator (NIH Publication 79-1649). Washington DC: US Government Printing Office.

Garwood, F. (1936) Fiducial Limits for the Poisson distribution. *Biometrika* 28:437-442.

https://www.ine.pt/revstat/pdf/rs120203.pdf

See Also

poisson.test, BinomCI, MultinomCI

Examples

```
# the horse kick example
count <- 0:4
deaths <- c(144, 91, 32, 11, 2)

n <- sum(deaths)
x <- sum(count * deaths)

lambda <- x/n

PoissonCI(x=x, n=n, method = c("exact","score", "wald", "byar"))

exp <- dpois(0:4, lambda) * n
```

```
barplot(rbind(deaths, exp * n/sum(exp)), names=0:4, beside=TRUE,
  col=c(DescTools::hred, DescTools::hblue), main = "Deaths from Horse Kicks",
  xlab = "count")
legend("topright", legend=c("observed","expected"),
  fill=c(DescTools::hred, DescTools::hblue), bg="white")


## SMR, Welsh Nickel workers
PoissonCI(x=137, n=24.19893)
```

PolarGrid *Plot a Grid in Polar Coordinates*

Description

PolarGrid adds a polar grid to an existing plot. The number of radial gridlines are set by ntheta and the tangential lines by nr. Labels for the angular grid and the radial axis can be provided.

Usage

```
PolarGrid(nr = NULL, ntheta = NULL, col = "lightgray", lty = "dotted", lwd = par("lwd"),
         rlabels = NULL, alabels = NULL, lblradians = FALSE, cex.lab = 1, las = 1,
           adj = NULL, dist = NULL)
```

Arguments

nr	number of circles. When NULL, as per default, the grid aligns with the tick marks on the corresponding default axis (i.e., tickmarks as computed by axTicks). When NA, no circular grid lines are drawn.
ntheta	number of radial grid lines. Defaults to 12 uniformly distributed between 0 and 2*pi (each pi/3).
col	character or (integer) numeric; color of the grid lines.
lty	character or (integer) numeric; line type of the grid lines.
lwd	non-negative numeric giving line width of the grid lines.
rlabels	the radius labels. Use NA if no labels should be to be added.
alabels	the labels for the angles, they are printed on a circle outside the plot. Use NA for no angle labels.
lblradians	logic, defines if angle labels will be in degrees (default) or in radians.
cex.lab	the character extension for the labels.
las	alignment of the labels, 1 means horizontal, 2 radial and 3 vertical.
adj	adjustments for the labels. (Left: 0, Right: 1, Mid: 0.5) The default is 1 for the levels on the right side of the circle, 0 for labels on the left and 0.5 for labels exactly on north on south.
dist	gives the radius for the labels, in user coordinates. Default is par("usr")[2] * 1.07.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

PlotPolar

Examples

```
Canvas(xlim=c(-5,5), xpd=TRUE)
PolarGrid()

Canvas(xlim=c(-5,5), xpd=TRUE)
PolarGrid(nr=0:5, ntheta=6)

Canvas(xlim=c(-5,5), xpd=TRUE)
PolarGrid(ntheta=36, rlabels=NA, lblradians=TRUE)
```

PostHocTest *Post-Hoc Tests*

Description

A convenience wrapper for computing post-hoc test after having calculated an ANOVA.

Usage

```
PostHocTest(x, ...)

## S3 method for class 'aov'
PostHocTest(x, which = NULL,
        method = c("hsd", "bonferroni", "lsd", "scheffe", "newmankeuls", "duncan"),
            conf.level = 0.95, ordered = FALSE, ...)

## S3 method for class 'table'
PostHocTest(x, method = c("none", "fdr", "BH", "BY", "bonferroni",
                          "holm", "hochberg", "hommel"),
            conf.level = 0.95, ...)


## S3 method for class 'PostHocTest'
print(x, digits = getOption("digits", 3), ...)
## S3 method for class 'PostHocTest'
plot(x, ...)
```

Arguments

x	an aov object.
method	one of "hsd", "bonf", "lsd", "scheffe", "newmankeuls", defining the method for the pairwise comparisons. For the post hoc test of tables the methods of p.adjust can be supplied. See the detail there.
which	a character vector listing terms in the fitted model for which the intervals should be calculated. Defaults to all the terms.
conf.level	a numeric value between zero and one giving the family-wise confidence level to use. If this is set to NA, just a matrix with the p-values will be returned.
ordered	a logical value indicating if the levels of the factor should be ordered according to increasing average in the sample before taking differences. If ordered is TRUE then the calculated differences in the means will all be positive. The significant differences will be those for which the lower end point is positive. This argument will be ignored if method is not either hsd or newmankeuls.
digits	controls the number of fixed digits to print.
...	further arguments, not used so far.

Details

The function is designed to consolidate a couple of post-hoc tests with the same interface for input and output.

Choosing Tests. Different post hoc tests use different methods to control familywise (FW) and per experiment error rate (PE). Some tests are very conservative. Conservative tests go to great lengths to prevent the user from committing a type 1 error. They use more stringent criterion for determining significance. Many of these tests become more and more stringent as the number of groups increases (directly limiting the FW and PE error rate). Although these tests buy you protection against type 1 error, it comes at a cost. As the tests become more stringent, you loose power (1-B). More liberal tests, buy you power but the cost is an increased chance of type 1 error. There is no set rule for determining which test to use, but different researchers have offered some guidelines for choosing. Mostly it is an issue of pragmatics and whether the number of comparisons exceeds k-1.

The Fisher's LSD (Least Significant Different) sets alpha level per comparison. alpha = .05 for every comparison. df = df error (i.e. df within). This test is the most liberal of all post hoc tests. The critical t for significance is unaffected by the number of groups. This test is appropriate when you have 3 means to compare. In general the alpha is held at .05 because of the criterion that you can't look at LSD's unless the ANOVA is significant. This test is generally not considered appropriate if you have more than 3 means unless there is reason to believe that there is no more than one true null hypothesis hidden in the means.

Dunn's (Bonferroni) t-test is sometimes referred to as the Bonferroni t because it used the Bonferroni PE correction procedure in determining the critical value for significance. In general, this test should be used when the number of comparisons you are making exceeds the number of degrees of freedom you have between groups (e.g. k-1). This test sets alpha per experiment; alpha = (.05)/c for every comparison. df = df error (c = number of comparisons (k(k-1))/2) This test is extremely conservative and rapidly reduces power as the number of comparisons being made increase.

Newman-Keuls is a step down procedure that is not as conservative as Dunn's t test. First, the means of the groups are ordered (ascending or descending) and then the largest and smallest means are tested for significant differences. If those means are different, then test smallest with next largest, until you reach a test that is not significant. Once you reach that point then you can only test differences between means that exceed the difference between the means that were found to be non-significant. Newman-Keuls is perhaps one of the most common post hoc test, but it is a rather controversial test. The major problem with this test is that when there is more than one true null hypothesis in a set of means it will overestimate the FW error rate. In general we would use this when the number of comparisons we are making is larger than k-1 and we don't want to be as conservative as the Dunn's test is.

Tukey's HSD (Honestly Significant Difference) is essentially like the Newman-Keuls, but the tests between each mean are compared to the critical value that is set for the test of the means that are furthest apart (rmax e.g. if there are 5 means we use the critical value determined for the test of X1 and X5). This method corrects for the problem found in the Newman-Keuls where the FW is inflated when there is more than one true null hypothesis in a set of means. It buys protection against type 1 error, but again at the cost of power. It tends to be the most common and preferred test because it is very conservative with respect to type 1 error when the null hypothesis is true. In general, HSD is preferred when you will make all the possible comparisons between a large set of means (6 or more means).

The Scheffe test is designed to protect against a type 1 error when all possible complex and simple comparisons are made. That is we are not just looking the possible combinations of comparisons between pairs of means. We are also looking at the possible combinations of comparisons between groups of means. Thus Scheffe is the most conservative of all tests. Because this test does give us the capacity to look at complex comparisons, it essentially uses the same statistic as the linear contrasts tests. However, Scheffe uses a different critical value (or at least it makes an adjustment to the critical value of F). This test has less power than the HSD when you are making pairwise (simple) comparisons, but it has more power than HSD when you are making complex comparisons. In general, only use this when you want to make many post hoc complex comparisons (e.g. more than k-1).

Tables For tables pairwise chi-square test can be performed, either without correction or with correction for multiple testing following the logic in `p.adjust`.

Value

an object of type "PostHocTest", which will either be
A) a list of data.frames containing the mean difference, lower ci, upper ci and the p-value, if a conf.level was defined (something else than NA) or
B) a list of matrices with the p-values, if conf.level has been set to NA.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`TukeyHSD`, `aov`, `pairwise.t.test`, `ScheffeTest`

Examples

```
PostHocTest(aov(breaks ~ tension, data = warpbreaks), method = "lsd")
PostHocTest(aov(breaks ~ tension, data = warpbreaks), method = "hsd")
PostHocTest(aov(breaks ~ tension, data = warpbreaks), method = "scheffe")

r.aov <- aov(breaks ~ tension, data = warpbreaks)

# compare p-values:
round(cbind(
    lsd= PostHocTest(r.aov, method="lsd")$tension[,"pval"]
  , bonf=PostHocTest(r.aov, method="bonf")$tension[,"pval"]
), 4)

# only p-values by setting conf.level to NA
PostHocTest(aov(breaks ~ tension, data = warpbreaks), method = "hsd",
            conf.level=NA)
```

---

power.chisq.test *Power Calculations for ChiSquared Tests*

Description

Compute power of test or determine parameters to obtain target power (same as power.anova.test).

Usage

```
power.chisq.test(n = NULL, w = NULL, df = NULL, sig.level = 0.05, power = NULL)
```

Arguments

n	total number of observations.
w	effect size.
df	degree of freedom (depends on the chosen test.
sig.level	Significance level (Type I error probability).
power	Power of test (1 minus Type II error probability).

Details

Exactly one of the parameters w, n, power or sig.level must be passed as NULL, and this parameter is determined from the others. Note that the last one has non-NULL default, so NULL must be explicitly passed, if you want to compute it.

Value

Object of class "power.htest", a list of the arguments (including the computed one) augmented with 'method' and 'note' elements.

Note

uniroot is used to solve power equation for unknowns, so you may see errors from it, notably about inability to bracket the root when invalid arguments are given.

Author(s)

Stephane Champely <champely@univ-lyon1.fr>
but this is a mere copy of Peter Dalgaard's work on power.t.test

References

Cohen, J. (1988) *Statistical power analysis for the behavioral sciences (2nd ed.)* Hillsdale, NJ: Lawrence Erlbaum.

See Also

power.t.test

Examples

```
## Exercise 7.1 P. 249 from Cohen (1988)
power.chisq.test(w=0.289, df=(4-1), n=100, sig.level=0.05)

## Exercise 7.3 p. 251
power.chisq.test(w=0.346, df=(2-1)*(3-1), n=140, sig.level=0.01)

## Exercise 7.8 p. 270
power.chisq.test(w=0.1, df=(5-1)*(6-1), power=0.80, sig.level=0.05)
```

PowerPoint Interface *Add Slides, Insert Texts and Plots to PowerPoint*

Description

A couple of functions to get R-stuff into MS-Powerpoint.

GetNewPP() starts a new instance of PowerPoint and returns its handle. A new presentation with one empty slide will be created thereby. The handle is needed for addressing the presentation afterwards.
GetCurrPP() will look for a running PowerPoint instance and return its handle. NULL is returned if nothing's found. PpAddSlide() inserts a new slide into the active presentation.
PpPlot() inserts the active plot into PowerPoint. The image is transferred by saving the picture to a file in R and inserting the file in PowerPoint. The format of the plot can be selected, as well as crop options and the size factor for inserting.
PpText() inserts a new textbox with given text and box properties.

Usage

```
GetNewPP(visible = TRUE, template = "Normal")
GetCurrPP()

PpAddSlide(pos = NULL, pp = DescToolsOptions("lastPP"))

PpPlot(type = "png", crop = c(0, 0, 0, 0), picscale = 100, x = 1, y = 1,
     height = NA, width = NA, res=200, dfact=1.6, pp = DescToolsOptions("lastPP"))

PpText(txt, x = 1, y = 1, height = 50, width = 100, fontname = "Calibri", fontsize = 18,
       bold = FALSE, italic = FALSE, col = "black", bg = "white",
       hasFrame = TRUE, pp = DescToolsOptions("lastPP"))
```

Arguments

visible	logical, should PowerPoint made visible by GetNewPP()? Defaults to TRUE.
template	the name of the template to be used for the new presentation.
pos	position of the new inserted slide within the presentation.
type	the format for the picture file, default is "png".
crop	crop options for the picture, defined by a 4-elements-vector. The first element is the bottom side, the second the left and so on.
picscale	scale factor of the picture in percent, default ist 100.
x, y	left/upper xy-coordinate for the plot or for the textbox.
height	height in cm, this overrides the picscale if both are given.
width	width in cm, this overrides the picscale if both are given.
res	resolution for the png file, defaults to 200.
dfact	the size factor for the graphic.
txt	text to be placed in the textbox
fontname	used font for textbox
fontsize	used fontsize for textbox
bold	logic. Text is set bold if this is set to TRUE (default is FALSE).
italic	logic. Text is set italic if this is to TRUE (default is FALSE).
col	font color, defaults to "black".
bg	background color for textboxdefaults to "white".
hasFrame	logical. Defines if a textbox is to be framed. Default is TRUE.
pp	the pointer to a PowerPoint instance, can be a new one, created by GetNewPP() or the last created by DescToolsOptions("lastPP") (default).

Details

See PowerPoint-objectmodel for further informations.

Value

The functions return the pointer to the created object.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

WrdPlot

Examples

```
## Not run: # Windows-specific example

# let's have some graphic
plot(1,type="n", axes=FALSE, xlab="", ylab="", xlim=c(0,1), ylim=c(0,1))
rect(0,0,1,1,col="black")
segments(x0=0.5, y0=seq(0.632,0.67, length.out=100),
  y1=seq(0.5,0.6, length.out=100), x1=1, col=rev(rainbow(100)))
polygon(x=c(0.35,0.65,0.5), y=c(0.5,0.5,0.75), border="white",
  col="black", lwd=2)
segments(x0=0,y0=0.52, x1=0.43, y1=0.64, col="white", lwd=2)
x1 <- seq(0.549,0.578, length.out=50)
segments(x0=0.43, y0=0.64, x1=x1, y1=-tan(pi/3)* x1 + tan(pi/3) * 0.93,
  col=rgb(1,1,1,0.35))


# get a handle to a new PowerPoint instance
pp <- GetNewPP()
# insert plot with a specified height
PpPlot(pp=pp,  x=150, y=150, height=10, width=10)

PpText("Remember?\n", fontname="Arial", x=200, y=70, height=30, fontsize=14,
       bold=TRUE, pp=pp, bg="lemonchiffon", hasFrame=TRUE)

PpAddSlide(pp=pp)
# crop the picture
pic <- PpPlot(pp=pp, x=1, y=200, height=10, width=10, crop=c(9,9,0,0))
pic


# some more automatic procedure
pp <- GetNewPP()
PpText("Hello to my presentation", x=100, y=100, fontsize=32, bold=TRUE,
       width=300, hasFrame=FALSE, col="blue", pp=pp)

for(i in 1:4){
  barplot(1:4, col=i)
  PpAddSlide(pp=pp)
  PpPlot(height=15, width=21, x=50, y=50, pp=pp)
  PpText(gettextf("This is my barplot nr %s", i), x=100, y=10, width=300, pp=pp)
```

```
}

## End(Not run)
```

---

pRevGumbel *"Reverse" Gumbel Distribution Functions*

Description

Density, distribution function, quantile function and random generation for the "Reverse" Gumbel distribution with parameters `location` and `scale`.

Usage

```
dRevGumbel (x, location = 0, scale = 1)
pRevGumbel (q, location = 0, scale = 1)
qRevGumbel (p, location = 0, scale = 1)
rRevGumbel (n, location = 0, scale = 1)

qRevGumbelExp(p)
```

Arguments

x, q	numeric vector of abscissa (or quantile) values at which to evaluate the density or distribution function.
p	numeric vector of probabilities at which to evaluate the quantile function.
location	location of the distribution
scale	scale (> 0) of the distribution.
n	number of random variates, i.e., `length` of resulting vector of `rRevGumbel(..)`.

Value

a numeric vector, of the same length as `x`, `q`, or `p` for the first three functions, and of length `n` for `rRevGumbel()`.

Author(s)

Werner Stahel; partly inspired by package **VGAM**. Martin Maechler for numeric cosmetic.

See Also

the `Weibull` distribution functions in R's **stats** package.

Examples

```
curve(pRevGumbel(x, scale= 1/2), -3,2, n=1001, col=1, lwd=2,
      main = "RevGumbel(x, scale = 1/2)")
abline(h=0:1, v = 0, lty=3, col = "gray30")
curve(dRevGumbel(x, scale= 1/2),       n=1001, add=TRUE,
      col = (col.d <- adjustcolor(2, 0.5)), lwd=3)
legend("left", c("cdf","pdf"), col=c("black", col.d), lwd=2:3, bty="n")

med <- qRevGumbel(0.5, scale=1/2)
cat("The median is:",  format(med),"\n")
```

Primes *Find All Primes Less Than n*

Description

Find all prime numbers aka 'primes' less than n.

Uses an obvious sieve method and some care, working with logical and integers to be quite fast.

Usage

```
Primes(n)
```

Arguments

n	a (typically positive integer) number.

Details

As the function only uses `max(n)`, `n` can also be a *vector* of numbers.

Value

numeric vector of all prime numbers $\leq n$.

Note

This function was previously published in the package **sfsmisc** as `primes` and has been integrated here without logical changes.

Author(s)

Bill Venables ($\leq n$); Martin Maechler gained another 40% speed, working with logicals and integers.

See Also

`Factorize`, `GCD`, `LCM`, `IsPrime`

Examples

```
(p1 <- Primes(100))
system.time(p1k <- Primes(1000)) # still lightning ..

stopifnot(length(p1k) == 168)
```

PseudoR2 *Pseudo R2 Statistics*

Description

Although there's no commonly accepted agreement on how to assess the fit of a logistic regression, there are some approaches. The goodness of fit of the logistic regression model can be expressed by some variants of pseudo R squared statistics, most of which being based on the deviance of the model.

Usage

```
PseudoR2(x, which = NULL)
```

Arguments

x	the glm, polr or multinom model object to be evaluated.
which	character, one out of "McFadden", "McFaddenAdj", "CoxSnell", "Nagelkerke", "AldrichNelson", "VeallZimmermann", "Efron", "McKelveyZavoina", "Tjur", "all". Partial matching is supported.

Details

Cox and Snell's R^2 is based on the log likelihood for the model compared to the log likelihood for a baseline model. However, with categorical outcomes, it has a theoretical maximum value of less than 1, even for a "perfect" model.

Nagelkerke's R^2 (also sometimes called Cragg-Uhler) is an adjusted version of the Cox and Snell's R^2 that adjusts the scale of the statistic to cover the full range from 0 to 1.

McFadden's R^2 is another version, based on the log-likelihood kernels for the intercept-only model and the full estimated model.

Veall and Zimmermann concluded that from a set of six widely used measures the measure suggested by McKelvey and Zavoina had the closest correspondance to ordinary least square R2. The Aldrich-Nelson pseudo-R2 with the Veall-Zimmermann correction is the best approximation of the McKelvey-Zavoina pseudo-R2. Efron, Aldrich-Nelson, McFadden and Nagelkerke approaches severely underestimate the "true R2".

Value

the value of the specific statistic. `AIC`, `LogLik`, `LogLikNull` and `G2` will only be reported with option `"all"`.

`McFadden`	McFadden pseudo-R^2
`McFaddenAdj`	McFadden adjusted pseudo-R^2
`CoxSnell`	Cox and Snell pseudo-R^2 (also known as ML pseudo-R^2)
`Nagelkerke`	Nagelkerke pseudoR^2 (also known as CraggUhler R^2)
`AldrichNelson`	AldrichNelson pseudo-R^2
`VeallZimmermann`	VeallZimmermann pseudo-R^2
`McKelveyZavoina`	McKelvey and Zavoina pseudo-R^2
`Efron`	Efron pseudo-R^2
`Tjur`	Tjur's pseudo-R^2
`AIC`	Akaike's information criterion
`LogLik`	log-Likelihood for the fitted model (by maximum likelihood)
`LogLikNull`	log-Likelihood for the null model. The null model will include the offset, and an intercept if there is one in the model.
`G2`	differenz of the null deviance - model deviance

Author(s)

Andri Signorell <andri@signorell.net> with contributions of Ben Mainwaring <benjamin.mainwaring@yougov.com> and Daniel Wollschlaeger

References

Aldrich, J. H. and Nelson, F. D. (1984): Linear Probability, Logit, and probit Models, *Sage University Press*, Beverly Hills.

Cox D R & Snell E J (1989) *The Analysis of Binary Data* 2nd ed. London: Chapman and Hall.

Efron, B. (1978). Regression and ANOVA with zero-one data: Measures of residual variation. *Journal of the American Statistical Association, 73*(361), 113–121.

Hosmer, D. W., & Lemeshow, S. (2000). *Applied logistic regression* (2nd ed.). Hoboke, NJ: Wiley.

McFadden D (1979). Quantitative methods for analysing travel behavior of individuals: Some recent developments. In D. A. Hensher & P. R. Stopher (Eds.), *Behavioural travel modelling* (pp. 279-318). London: Croom Helm.

McKelvey, R. D., & Zavoina, W. (1975). A statistical model for the analysis of ordinal level dependent variables. *The Journal of Mathematical Sociology, 4*(1), 103–120

Nagelkerke, N. J. D. (1991). A note on a general definition of the coefficient of determination. *Biometrika, 78*(3), 691–692.

Tjur, T. (2009) Coefficients of determination in logistic regression models - a new proposal: The coefficient of discrimination. *The American Statistician*, 63(4): 366-372

Veall, M.R., & Zimmermann, K.F. (1992) Evalutating Pseudo-R2's fpr binary probit models. *Quality&Quantity*, 28, pp. 151-164

See Also

logLik, AIC, BIC

Examples

```
r.glm <- glm(Survived ~ ., data=Untable(Titanic), family=binomial)
PseudoR2(r.glm)

PseudoR2(r.glm, c("McFadden", "Nagel"))
```

PtInPoly *Point in Polygon*

Description

PtInPoly works out, whether XY-points lie within the boundaries of a given polygon.

Note: Points that lie on the boundaries of the polygon or vertices are assumed to lie within the polygon.

Usage

```
PtInPoly(pnts, poly.pnts)
```

Arguments

pnts	a 2-column matrix or dataframe defining locations of the points of interest
poly.pnts	a 2-column matrix or dataframe defining the locations of vertices of the polygon of interest

Details

The algorithm implements a sum of the angles made between the test point and each pair of points making up the polygon. The point is interior if the sum is 2pi, otherwise, the point is exterior if the sum is 0. This works for simple and complex polygons (with holes) given that the hole is defined with a path made up of edges into and out of the hole.

This sum of angles is not able to consistently assign points that fall on vertices or on the boundary of the polygon. The algorithm defined here assumes that points falling on a boundary or polygon vertex are part of the polygon.

Value

A 3-column dataframe where the first 2 columns are the original locations of the points. The third column (names pip) is a vector of binary values where 0 represents points not with the polygon and 1 within the polygon.

Author(s)

Jeremy VanDerWal <jjvanderwal@gmail.com>

Examples

```
#define the points and polygon
pnts <- expand.grid(x=seq(1,6,0.1), y=seq(1,6,0.1))
polypnts <- cbind(x=c(2,3,3.5,3.5,3,4,5,4,5,5,4,3,3,3,2,2,1,1,1,1,2),
                  y=c(1,2,2.5,2,2,1,2,3,4,5,4,5,4,3,3,4,5,4,3,2,2) )

#plot the polygon and all points to be checked
plot(rbind(polypnts, pnts))
polygon(polypnts, col='#99999990')

#create check which points fall within the polygon
out <- PtInPoly(pnts, polypnts)
head(out)

#identify points not in the polygon with an X
points(out[which(out$pip==0), 1:2], pch='X')
```

Quantile *(Weighted) Sample Quantiles*

Description

Compute weighted quantiles (Eurostat definition).

Usage

```
Quantile(x, weights = NULL, probs = seq(0, 1, 0.25),
         na.rm = FALSE, names = TRUE, type = 7, digits = 7)
```

Arguments

x	a numeric vector.
weights	an optional numeric vector giving the sample weights.
probs	numeric vector of probabilities with values in $[0, 1]$.
na.rm	a logical indicating whether missing values in x should be omitted.
names	logical; if true, the result has a names attribute. Set to FALSE for speedup with many probs.
type	an integer between 1 and 9 selecting one of the nine quantile algorithms detailed below to be used. Currently only types 5 and 7 (default) are implemented.
digits	used only when names is true: the precision to use when formatting the percentages. In R versions up to 4.0.x, this had been set to max(2, getOption("digits")), internally.

Details

The implementation strictly follows the Eurostat definition.

Value

A named numeric vector containing the weighted quantiles of values in `x` at probabilities `probs` is returned.

Author(s)

Andreas Alfons, Matthias Templ, some tweaks Andri Signorell <andri@signorell.net>

References

Working group on Statistics on Income and Living Conditions (2004) Common cross-sectional EU indicators based on EU-SILC; the gender pay gap. *EU-SILC 131-rev/04*, Eurostat.

See Also

`Median`, `quantile`, `QuantileCI`

Examples

```
Quantile(d.pizza$temperature, rep(c(1:3), length.out=nrow(d.pizza)))
```

QuantileCI *Confidence Interval for Any Quantile*

Description

Calculates the confidence interval for any quantile. Although bootstrapping might be a good approach for getting senisble confidence intervals there's sometimes need to have a nonparameteric alternative. This function offers one.

Usage

```
QuantileCI(x, probs=seq(0, 1, .25), conf.level = 0.95,
           sides = c("two.sided", "left", "right"),
           na.rm = FALSE, method = c("exact", "boot"), R = 999)
```

Arguments

`x`	a (non-empty) numeric vector of data values.
`probs`	numeric vector of probabilities with values in *[0,1]*. (Values up to `2e-14` outside that range are accepted and moved to the nearby endpoint.)
`conf.level`	confidence level of the interval

sides	a character string specifying the side of the confidence interval, must be one of "two.sided" (default), "left" or "right" (abbreviations allowed). "left" would be analogue to a "greater" hypothesis in a t.test.
na.rm	logical. Should missing values be removed? Defaults to FALSE.
method	defining the type of interval that should be calculated (one out of "exact", "boot"). Default is "exact". See Details.
R	The number of bootstrap replicates. Usually this will be a single positive integer. See boot.ci for details.

Details

The "exact" method corresponds to the way the confidence interval for the median is calculated in SAS.
The boot confidence interval type is calculated by means of boot.ci with default type "basic".

Value

if probs was of length 1 a numeric vector with 3 elements:

est	est
lwr.ci	lower bound of the confidence interval
upr.ci	upper bound of the confidence interval

or, if probs was a vector, a matrix with 3 columns consisting of estimate, lower ci, upper ci est, lwr.ci, upr.ci

Author(s)

Andri Signorell <andri@signorell.net> based on code of W Huber on StackExchange

See Also

Quantile, quantile, MedianCI

Examples

```
QuantileCI(d.pizza$price, probs=0.25, na.rm=TRUE)

QuantileCI(d.pizza$price, na.rm=TRUE)
QuantileCI(d.pizza$price, conf.level=0.99, na.rm=TRUE)

# multiple probs
QuantileCI(1:100, method="exact" , probs = c(0.25, 0.75, .80, 0.95))
QuantileCI(1:100, method="boot" , probs = c(0.25, 0.75, .80, 0.95))
```

Quot *Lagged Quotients*

Description

Returns suitably lagged and iterated quotients

Usage

```
Quot(x, lag = 1, quotients = 1, ...)
```

Arguments

x	a numeric vector or matrix containing the values to be used for calculating the quotients.
lag	an integer indicating which lag to use.
quotients	an integer indicating the order of the quotient.
...	further arguments to be passed to or from methods.

Details

NA's propagate.

Value

If x is a vector of length n and quotients = 1, then the computed result is equal to the successive quotients x[(1+lag):n] - x[1:(n-lag)].

If quotients is larger than one this algorithm is applied recursively to x. Note that the returned value is a vector which is shorter than x.

If x is a matrix then the division operations are carried out on each column separately.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

diff

Examples

```
Quot(1:10, 2)
Quot(1:10, 2, 2)
x <- cumprod(cumprod(1:10))
Quot(x, lag = 2)
Quot(x, quotients = 2)
```

`Range` *(Robust) Range*

Description

Determines the range of the data, which can possibly be trimmed before calculating the extreme values. The robust range version is calculated on the basis of the trimmed mean and variance (see Details).

Usage

```
Range(x, trim = NULL, robust = FALSE, na.rm = FALSE, ...)
```

Arguments

`x`	a numeric vector.
`trim`	the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of trim outside that range are taken as the nearest endpoint. Default is 0 for `robust=FALSE` and 0.2 for `robust=TRUE`
`robust`	logical, determining whether the robust or the convential range should be returned.
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.
`...`	the dots are sent to `RobRange` and can be used to set `fac` (See details).

Details

The R base function range returns the minimum and maximum value of a numeric object. Here we return the span of a (possibly trimmed) numeric vector, say the difference of maximum and minimum value.

If robust is set to `TRUE` the function determines the trimmed mean m and then the "upper trimmed mean" s of absolute deviations from m, multiplied by `fac` (fac is 3 by default). The robust minimum is then defined as m-fac*s or min(x), whichever is larger, and similarly for the maximum.

Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed, as a numeric or complex vector of length one. If `x` is not logical (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned, with a warning.

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

Author(s)

Werner Stahel, ETH Zurich (robust range)
Andri Signorell andri@signorell.net

See Also

range, min, max

Examples

```
x <- c(0:10, 50)
xm <- Range(x)
c(xm, Range(x, trim = 0.10))

x <- c(rnorm(20), rnorm(3, 5, 20))
Range(x, robust=TRUE)

# compared to
Range(x)
```

Rank *Fast Sample Ranks*

Description

The function base::rank has various weaknesses. Apart from the fact that it is not very fast, the option to calculate dense ranks is not implemented. Then, an argument for specifying the ranking direction is missing (assuming that this can be done with the ranking of the negative variables) and finally, multiple columns cannot be used in the case of ties for further ranking.
The function data.table::frankv provides a more elaborated interface and convinces by very performant calculations and is *much faster* than the original. It further accepts vectors, lists, data.frames or data.tables as input. In addition to the ties.method possibilities provided by base::rank, it also provides ties.method="dense".
The present function Rank is merely a somewhat customized parameterization of the data.table function.

Usage

```
Rank(..., decreasing = FALSE, na.last = TRUE,
     ties.method = c("average", "first", "last", "random",
                     "max", "min", "dense"))
```

Arguments

...	A vector, or list with all its elements identical in length or data.frame or data.table.
decreasing	An logical vector corresponding to ascending and descending order. decreasing is recycled to length(...).
na.last	Control treatment of NAs. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA, they are removed; if "keep" they are kept with rank NA.
ties.method	A character string specifying how ties are treated, see Details.

Details

To be consistent with other `data.table` operations, `NA`s are considered identical to other `NA`s (and `NaN`s to other `NaN`s), unlike `base::rank`. Therefore, for `na.last=TRUE` and `na.last=FALSE`, `NA`s (and `NaN`s) are given identical ranks, unlike `rank`.

`Rank` is not limited to vectors. It accepts `data.table`s (and `list`s and `data.frame`s) as well. It accepts unquoted column names (with names preceded with a `-` sign for descending order, even on character vectors), for e.g., `Rank(DT, a, -b, c, ties.method="first")` where `a,b,c` are columns in `DT`.

In addition to the `ties.method` values possible using base's `rank`, it also provides another additional argument `"dense"`. Dense ranks are consecutive integers beginning with 1. No ranks are skipped if there are ranks with multiple items. So the largest rank value is the number of unique values of x. See examples.

Like `forder`, sorting is done in "C-locale"; in particular, this may affect how capital/lowercase letters are ranked. See Details on `forder` for more.

`bit64::integer64` type is also supported.

Value

A numeric vector of length equal to `NROW(x)` (unless `na.last = NA`, when missing values are removed). The vector is of integer type unless `ties.method = "average"` when it is of double type (irrespective of ties).

See Also

`frankv`, `data.table`, `setkey`, `setorder`

Examples

```
# on vectors
x <- c(4, 1, 4, NA, 1, NA, 4)
# NAs are considered identical (unlike base R)
# default is average
Rank(x) # na.last=TRUE
Rank(x, na.last=FALSE)

# ties.method = min
Rank(x, ties.method="min")
# ties.method = dense
Rank(x, ties.method="dense")

# on data.frame, using both columns
d.set <- data.frame(x, y=c(1, 1, 1, 0, NA, 0, 2))
Rank(d.set, na.last="keep")
Rank(d.set, ties.method="dense", na.last=NA)

# decreasing argument
Rank(d.set, decreasing=c(FALSE, TRUE), ties.method="first")
```

ReadSPSS — *Read SPSS Datafiles and Return It in Good Old Style Data Frame Structure*

Description

While haven is a great package it uses tibbles as basic data structures. Older R users (as myself) might prefer a more archaic structures. This function returns SPSS files in form of a data.frame and the nominal variables as factors.

Usage

```
ReadSPSS(fn, encoding = NULL)
```

Arguments

fn	Either a path to a file, a connection, or literal data (either a single string or a raw vector).
encoding	The character encoding used for the file. The default, NULL, use the encoding specified in the file, but sometimes this value is incorrect and it is useful to be able to override it.

Value

A data frame.

Variable labels are stored in the "label" attribute of each variable. It is not printed on the console, but the RStudio viewer will show it.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

read_spss

Recode — *Recode a Factor*

Description

Combining or rearranging a factor can be tedious if it has many levels. Recode supports this step by accepting a direct definition of new levels by enumerating old levelnames as argument and adding an "elselevel" option. If new levels are given as integer values they will be translated in the according levels.

Usage

```
Recode(x, ..., elselevel = NA, use.empty = FALSE, num = FALSE)
```

Arguments

x	the factor whose levels are to be altered. If x is character it will be factorized (using factor defaults).
...	the old levels (combined by c() if there are several) named with the new level: newlevel_a=c("old_a", "old_b"), newlevel_b=c("old_c", "old_d") See examples.
elselevel	the value for levels, which are not matched by newlevel list. If this is set to NULL, the elselevels will be left unchanged. If set to NA (default) non matched levels will be set to NA.
use.empty	logical. Defines how a new level, which can't be found in x, should be handled. Should it be left in the level's list or be dropped? The default is FALSE, which drops empty levels.
num	logical. If set to TRUE the result will be numeric.

Value

the factor having the new levels applied.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

factor, levels
There's another solution in the package **car**.

Examples

```
set.seed(1984)
x <- factor(sample(1:15, 20, replace=TRUE))
levels(x) <- paste("old", levels(x), sep="_")

y <- Recode(x,
            "new_1"   = c("old_1","old_4","old_5"),
            "new_2"   = c("old_6","old_10","old_11"),
            "new_3"   = c("old_12","old_13"),
            elselevel = "other")
data.frame(x=x, y=y)

# Coding NAs, NA is recoded to new_1
x[5:6] <- NA
x <- x[1:7]
```

```
data.frame(
  x,
  RecodeNA = Recode(x,
                    "new_1"   = c("old_4","old_8", NA),
                    elselevel = "other"),

  # NAs remain untouched
  NoRecodeNA = Recode(x,
                      "new_1"   = c("old_4","old_8"),
                      elselevel = "other")
)


x <- factor(letters[1:6])

z1 <- Recode(x, AB=c("a","b"), CD=c("c","d"), elselevel="none of these")
z2 <- Recode(x, AB=c("a","b"), CD=c("c","d"), elselevel=NA)
z3 <- Recode(x, AB=c("a","b"), CD=c("c","d"), elselevel=NULL)
z4 <- Recode(x, AB=c("a","b"), GH=c("g","h"), elselevel=NA, use.empty=TRUE)
z5 <- Recode(x, AB=c("a","b"), GH=c("g","h"), elselevel=NA, use.empty=FALSE)

data.frame(z1, z2, z3, z4, z5)

lapply(data.frame(z1, z2, z3, z4, z5), levels)

# empty level GH exists in z4...
table(z4, useNA="ifany")
# and is dropped in z5
table(z5, useNA="ifany")

# use integers to define the groups to collapse
set.seed(1972)
(likert <- factor(sample(1:10, size=15, replace=TRUE),
                  levels=1:10, labels=gettextf("(%s)", 1:10)))
Recode(likert, det=1:6, pas=7:8, pro=9:10)

# or directly turned to numeric
Recode(likert, "1"=1:6, "2"=7:8, "5"=9:10, num=TRUE)
```

`Recycle` *Recyle a List of Elements*

Description

This function recycles all supplied elments to the maximal dimension.

Usage

```
Recycle(...)
```

Arguments

...	a number of vectors of elements.

Value

a list of the supplied elements
attr(,"maxdim") contains the maximal dimension of the recyled list

Author(s)

Andri Signorell <andri@signorell.net>

See Also

rep, replicate

Examples

```
Recycle(x=1:5, y=1, s=letters[1:2])

z <- Recycle(x=letters[1:5], n=2:3, sep=c("-"," "))
sapply(1:attr(z, "maxdim"), function(i) paste(rep(z$x[i], times=z$n[i]), collapse=z$sep[i]))
```

RelRisk *Relative Risk*

Description

Computes the relative risk and its confidence intervals. Confidence intervals are calculated using normal approximation ("wald"), ("score") or by using odds ratio ("use.or")

Usage

```
RelRisk(x, y = NULL, conf.level = NA,
        method = c("score", "wald", "use.or"), delta = 0.5, ...)
```

Arguments

x	a numeric vector or a 2x2 numeric matrix, resp. table.
y	NULL (default) or a vector with compatible dimensions to x. If y is provided, table(x, y, ...) will be calculated.
conf.level	confidence level. Default is NA, meaning no confidence intervals will be reported.
method	method for calculating the relative risk and the confidence intervals. Can be one out of "score", "wald", "use.or". Default is "score".
delta	small constant to be added to the numerator for calculating the log risk ratio (Wald method). Usual choice is 0.5 although there does not seem to be any theory behind this. (Dewey, M. 2006)
...	further arguments are passed to the function table, allowing i.e. to set useNA.

Details

Best is to always put the outcome variable (disease yes/no) in the columns and the exposure variable in the rows. In other words, put the dependent variable – the one that describes the problem under study – in the columns. And put the independent variable – the factor assumed to cause the problem – in the rows. (Gerritsen, 2010)

According to this, the function expects the following table structure:

```
                  diseased=1   diseased=0
exposed=1 (ref)      n00          n01
exposed=0            n10          n11
```

The relative risk is then calculated as:

$$rr = \frac{(\text{exposed} \;\&\; \text{diseased}) \;/\; \text{exposed}}{(\text{unexposed} \;\&\; \text{diseased}) \;/\; \text{unexposed}}$$

If the table to be used is not in the required shape, use the function `Rev()` and/or `t()` to reverse rows, columns, or both, resp. to transpose the table.

Value

If `conf.level` is not `NA` then the result will be a vector with 3 elements for estimate, lower confidence intervall and upper for the upper one. Else the relative risk will be reported as a single value.

Author(s)

Andri Signorell <andri@signorell.net>, based on code of Yongyi Min and Michael Dewey

References

Rothman, K. J. and Greenland, S. (1998) *Modern Epidemiology*. Lippincott-Raven Publishers

Rothman, K. J. (2002) *Epidemiology: An Introduction*. Oxford University Press

Jewell, N. P. (2004) *Statistics for Epidemiology*. 1st Edition, 2004, Chapman & Hall, pp. 73-81

Selvin, S. (1998) *Modern Applied Biostatistical Methods Using S-Plus*. 1st Edition, Oxford University Press

Gerritsen, A (2010) https://www.theanalysisfactor.com/cross-tabulation-in-cohort-and-case-control-studi

See Also

`OddsRatio`

Examples

```
m <- matrix(c(78,50,1422,950),
            nrow=2,
            dimnames = list(water=c("cont", "clean"),
                            diarrhea=c("yes", "no")))

RelRisk(m, conf.level = 0.95)


mm <- cbind(c(9,20),c(41,29))
mm

RelRisk(t(mm), conf.level=0.95)
RelRisk(t(mm), conf.level=0.95, method="wald")
RelRisk(t(mm), conf.level=0.95, method="use.or")
```

Rename *Change Names of a Named Object*

Description

Rename changes the names of a named object.

Usage

```
Rename(x, ..., gsub = FALSE, fixed = TRUE, warn = TRUE)
```

Arguments

x	Any named object
...	A sequence of named arguments, all of type character
gsub	a logical value; if TRUE, gsub is used to change the row and column labels of the resulting table. That is, instead of substituting whole names, substrings of the names of the object can changed.
fixed	a logical value, passed to gsub. If TRUE, substitutions are by fixed strings and not by regular expressions.
warn	a logical value; should a warning be issued if those names to change are not found?

Details

This function changes the names of x according to the remaining arguments. If gsub is FALSE, argument tags are the *old* names, the values are the new names. If gsub is TRUE, arguments are substrings of the names that are substituted by the argument values.

Value

The object x with new names defined by the ... arguments.

Note

This function was previously published in the package **memisc** as `rename` and has been integrated here without logical changes.

Author(s)

Martin Elff <melff@essex.ac.uk>

See Also

`SetNames`, `Recode` for recoding of a factor (renaming or combining levels)

Examples

```
x <- c(a=1, b=2)
Rename(x, a="A", b="B")

str(Rename( iris,
            Sepal.Length="Sepal_Length",
            Sepal.Width ="Sepal_Width",
            Petal.Length="Petal_Length",
            Petal.Width ="Petal_Width"
            ))

str(Rename(iris, .="_", gsub=TRUE))
```

---

reorder.factor *Reorder the Levels of a Factor*

Description

Reorder the levels of a factor

Usage

```
## S3 method for class 'factor'
reorder(x, X, FUN, ...,
        order = is.ordered(x), new.order, sort = SortMixed)
```

Arguments

`x`	factor
`X`	auxillary data vector
`FUN`	function to be applied to subsets of `X` determined by `x`, to determine factor order
`...`	optional parameters to `FUN`
`order`	logical value indicating whether the returned object should be an `ordered` factor

new.order	a vector of indexes or a vector of label names giving the order of the new factor levels
sort	function to use to sort the factor level names, used only when new.order is missing

Details

This function changes the order of the levels of a factor. It can do so via three different mechanisms, depending on whether, X *and* FUN, new.order or sort are provided.

If X *and* Fun are provided: The data in X is grouped by the levels of x and FUN is applied. The groups are then sorted by this value, and the resulting order is used for the new factor level names.

If new.order is provided: For a numeric vector, the new factor level names are constructed by reordering the factor levels according to the numeric values. For vectors, new.order gives the list of new factor level names. In either case levels omitted from new.order will become missing (NA) values.

If sort is provided (as it is by default): The new factor level names are generated by applying the supplied function to the existing factor level names. With sort=mixedsort the factor levels are sorted so that combined numeric and character strings are sorted in according to character rules on the character sections (including ignoring case), and the numeric rules for the numeric sections. See mixedsort for details.

Value

A new factor with reordered levels

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

factor and reorder

Examples

```
# Create a 4 level example factor
trt <- factor( sample( c("PLACEBO", "300 MG", "600 MG", "1200 MG"),
              100, replace=TRUE ) )
summary(trt)
# Note that the levels are not in a meaningful order.

# Change the order to something useful
# default "mixedsort" ordering
trt2 <- reorder(trt)
summary(trt2)
# using indexes:
trt3 <- reorder(trt, new.order=c(4, 2, 3, 1))
summary(trt3)
# using label names:
trt4 <- reorder(trt, new.order=c("PLACEBO", "300 MG", "600 MG", "1200 MG"))
```

```
summary(trt4)
# using frequency
trt5 <- reorder(trt, X=as.numeric(trt), FUN=length)
summary(trt5)

# drop out the '300 MG' level
trt6 <- reorder(trt, new.order=c("PLACEBO", "600 MG", "1200 MG"))
summary(trt6)
```

Rev *Reverse Elements of a Vector, a Matrix, a Table, an Array or a Data.frame*

Description

Rev provides a reversed version of its argument. Unlike the basic function, it does in higher-dimensional structures such as matrices not reverse the elements, but the order of the rows and/or columns. It further offers additional interfaces for higher dimensional arrays or tables.

Usage

```
Rev(x, ...)

## S3 method for class 'matrix'
Rev(x, margin, ...)

## S3 method for class 'table'
Rev(x, margin, ...)

## S3 method for class 'array'
Rev(x, margin, ...)

## S3 method for class 'data.frame'
Rev(x, margin, ...)
```

Arguments

x	a vector, a matrix or a higher dimensional table to be reversed.
margin	vector of dimensions which to be reversed (1 for rows, 2 for columns, etc.). If not defined, all dimensions will be reverted.
...	the dots are passed to the array interface.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

rev, order, sort, seq

Examples

```
tab <- matrix(c(1, 11, 111,
                2, 22, 222,
                3, 33, 333),
              byrow=TRUE, nrow=3,
              dimnames=list(mar1=1:3, mar2=c("a","b","c")))

Rev(tab, margin=1)
Rev(tab, margin=2)

# reverse both dimensions
Rev(tab, margin=c(1, 2))

t(tab)

# reverse 3dimensional array
aa <- Abind(tab, 2 * tab, along=3)
dimnames(aa)[[3]] <- c("A","Z")

# reverse rows
Rev(aa, 1)
# reverse columns
Rev(aa, 2)
# reverse 3th dimension
Rev(aa, 3)

# reverse all dimensions
Rev(aa)
# same as
Rev(aa, margin=(1:3))
```

RevCode *Reverse Codes*

Description

In psychology variables often need to be recoded into reverse order in cases that items are negatively worded. So it can be ensured that a high value indicate the same type of response on every item. Let's say we have a Likert scale from 1 to 5 and we want to recode the variable so that a 5 becomes a 1, 4 a 2 and so on.

Usage

```
RevCode(x, ...)
```

Arguments

x	a numerical or logical vector, or a factor.
...	the dots are sent to min/max, such as possibly to remove NAs before reversing numeric values.

Details

The function recodes based on:

`min(x, na.rm=TRUE) + max(x, na.rm=TRUE) - x`

Value

the recoded vector

Author(s)

Andri Signorell <andri@signorell.net>

See Also

Recode

Examples

```
x <- 1:5
data.frame(x, rev_num=RevCode(x), rev_fac=RevCode(factor(x)))

s <- c(3,4,2,7,4,9,NA,10)
RevCode(factor(s, levels=1:10))

i <- c(1,0,0,0,1,1)
cbind(i, RevCode(i))

k <- as.logical(c(1,0,0,0,1,1))
cbind(k, RevCode(k))

x <- factor(sample(letters[1:5], 10, replace = TRUE))
RevCode(x)

# we want to set the level 5 to NA before reversing
RevCode(factor(NAIf(x, "e")))
```

RevWeibull *The Reverse Weibull Distribution*

Description

Density function, distribution function, quantile function and random generation for the reverse (or negative) Weibull distribution with location, scale and shape parameters.

Usage

```
dRevWeibull(x, loc=0, scale=1, shape=1, log = FALSE)
pRevWeibull(q, loc=0, scale=1, shape=1, lower.tail = TRUE)
qRevWeibull(p, loc=0, scale=1, shape=1, lower.tail = TRUE)
rRevWeibull(n, loc=0, scale=1, shape=1)

dNegWeibull(x, loc=0, scale=1, shape=1, log = FALSE)
pNegWeibull(q, loc=0, scale=1, shape=1, lower.tail = TRUE)
qNegWeibull(p, loc=0, scale=1, shape=1, lower.tail = TRUE)
rNegWeibull(n, loc=0, scale=1, shape=1)
```

Arguments

x, q	Vector of quantiles.
p	Vector of probabilities.
n	Number of observations.
loc, scale, shape	Location, scale and shape parameters (can be given as vectors).
log	Logical; if TRUE, the log density is returned.
lower.tail	Logical; if TRUE (default), probabilities are P[X <= x], otherwise, P[X > x]

Details

The reverse (or negative) Weibull distribution function with parameters $loc = a$, $scale = b$ and $shape = s$ is

$$G(z) = \exp\left\{-\left[-\left(\frac{z-a}{b}\right)\right]^{s}\right\}$$

for $z < a$ and one otherwise, where $b > 0$ and $s > 0$.

Value

dRevWeibull and dNegWeibull give the density function, pRevWeibull and pNegWeibull give the distribution function, qRevWeibull and qNegWeibull give the quantile function, rRevWeibull and rNegWeibull generate random deviates.

Note

Within extreme value theory the reverse Weibull distibution (also known as the negative Weibull distribution) is often referred to as the Weibull distribution. We make a distinction to avoid confusion with the three-parameter distribution used in survival analysis, which is related by a change of sign to the distribution given above.

Author(s)

Alec Stephenson <alec_stephenson@hotmail.com>

See Also

rFrechet, rGenExtrVal, rGumbel

Examples

```
dRevWeibull(-5:-3, -1, 0.5, 0.8)
pRevWeibull(-5:-3, -1, 0.5, 0.8)
qRevWeibull(seq(0.9, 0.6, -0.1), 2, 0.5, 0.8)
rRevWeibull(6, -1, 0.5, 0.8)
p <- (1:9)/10
pRevWeibull(qRevWeibull(p, -1, 2, 0.8), -1, 2, 0.8)
## [1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
```

RgbToCmy *Conversion Between RGB and CMYK*

Description

These function convert colors between RGB and CMYK system.

Usage

```
RgbToCmy(col, maxColorValue = 1)
CmykToRgb(cyan, magenta, yellow, black, maxColorValue=1)
CmyToCmyk(col)
CmykToCmy(col)
```

Arguments

col	the matrix of the color to be converted
cyan	cyan values of the color(s) to be converted
magenta	magenta values of the color(s) to be converted
yellow	yellow values of the color(s) to be converted
black	black values of the color(s) to be converted
maxColorValue	the value for the color

Value

the converted value

Author(s)

Andri Signorell <andri@signorell.net>

See Also

RgbToCol

Examples

```
CmykToRgb(0.42, 45.23, 85.14, maxColorValue=100)
```

RgbToCol *Find the Nearest Named R-Color to a Given RGB-Color*

Description

Converting a RGB-color to a named R-Color means looking for a color in the R-palette, which is nearest to the given RGB-color. This function uses the minimum of squared distance ("euclidean") or the minimum absolute distance ("manhattan") as proximity measure.
RgbToLong() converts a RGB-color to a long integer in numeric format. LongToRGB() does it the other way round.

Usage

```
RgbToCol(col, method = "rgb", metric = "euclidean")

RgbToLong(col)
LongToRgb(col)
```

Arguments

col	the color in rgb code, say a matrix with the red, green and blue code in the rows.
method	character string specifying the color space to be used. Can be "rgb" (default) or "hsv".
metric	character string specifying the metric to be used for calculating distances between the colors. Available options are "euclidean" (default) and "manhattan". Euclidean distances are root sum-of-squares of differences, and manhattan distances are the sum of absolute differences.

Details

It may not be clear from the start which method, rgb or hsv, yield the more natural results. Trying and comparing is a recommended strategy. Moreover the shortest numerical distance will not always be the best choice, when comparing the colours visually.

Value

the name of the nearest found R color.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

ColToRgb and the other conversion functions

Examples

```
RgbToCol(matrix(c(162,42,42), nrow=3))

RgbToLong(matrix(c(162,42,42), nrow=3))
```

RndPairs *Create Pairs of Correlated Random Numbers*

Description

Create pairs of correlated random numbers.

Usage

```
RndPairs(n, r, rdist1 = rnorm(n = n, mean = 0, sd = 1),
         rdist2 = rnorm(n = n, mean = 0, sd = 1), prop = NULL)

RndWord(size, length, x = LETTERS, replace = TRUE, prob = NULL)
```

Arguments

n	number of pairs. If length(n) > 1, the length is taken to be the number required.
r	the correlation between the two sets.
rdist1, rdist2	the distribution of the random vector X1 and X2. Default is standard normal distribution.
size	a non-negative integer giving the number of artificial words to build.
length	a non-negative integer giving the length of the words.
x	elements to choose from.
replace	Should sampling be with replacement?
prop	proportions for ordinal variable, must sum to 1.
prob	a vector of probability weights for obtaining the elements of the vector being sampled.

Value

a data.frame with 2 columns, X1 and X2 containing the random numbers

Author(s)

Andri Signorell <andri@signorell.net>

See Also

runif, rnorm, Random and friends

Examples

```
# produce 100 pairs of a normal distributed random number with a correlation of 0.7
d.frm  <- RndPairs(n=100, r=0.7)

plot(d.frm)
lines(lm(y ~ x,d.frm))

# change the distribution
d.frm  <- RndPairs(n=100, r=0.7, rdist2 = rlnorm(n = 100, meanlog = 1, sdlog = .8))
d.frm  <- RndPairs(n=100, r=0.7, rdist2 = runif(n = 100, -1, 4))

x <- StrCap(sapply(sample(3:15, 10), function(i) RndWord(1, i, x=letters)))


# produce some artificial words with defined probabilities for the letters
p <- c(6.51,1.89,3.06,5.08,17.4,1.66,3.01,4.76,7.55,0.27,1.21,3.44,2.53,
       9.78,2.51,0.79,0.02,7,7.27,6.15,4.35,0.67,1.89,0.03,0.04,1.13)
sapply(sample(3:15, 10), function(i) RndWord(1, i, x=letters, prob=p))

# produce associated ordinal variables
d.ord <- RndPairs(500, r=0.8, prop = list(c(.15, .3, .55),
                                          c(.3, .5, .2)))
levels(d.ord$y) <- levels(d.ord$x) <- LETTERS[1:3]
PlotMosaic(table(d.ord$x, d.ord$y), las=1, main="")
```

RobScale *Robust Scaling With Median and Mad*

Description

RobScale is a wrapper function for robust standardization, using median and mad instead of mean and sd.

Usage

```
RobScale(x, center = TRUE, scale = TRUE)
```

Arguments

x	a numeric matrix(like object).
center	a logical value defining whether x should be centered by the median. Centering is done by subtracting the column medians (omitting NAs) of x from their corresponding columns. If center is FALSE, no centering is done.
scale	a logical value defining whether x should be scaled by the mad. Scaling is done by dividing the (centered) columns of x by their mad. If scale is FALSE, no scaling is done.

Value

the centered, scaled matrix. The numeric centering and scalings used (if any) are returned as attributes "scaled:center" and "scaled:scale"

Author(s)

Andri Signorell <andri@signorell.net>

See Also

scale, sweep

Examples

```
x <- d.pizza$temperature
plot(x=seq_along(x), y=RobScale(x), xlim=c(0,100))
points(x=seq_along(x), y=scale(x), col="red" )
```

RomanToInt *Convert Roman Numerals to Integers*

Description

Convert roman numerals to integers

Usage

```
RomanToInt(x)
```

Arguments

x	character vector containing roman numerals

Details

This functon will convert roman numerals to integers without the upper bound imposed by R (3899), ignoring case.

Value

A integer vector with the same length as roman. Character strings which are not valid roman numerals will be converted to NA.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

as.roman

Examples

```
RomanToInt( c('I', 'V', 'X', 'C', 'L', 'D', 'M' )  )

# works regardless of case
RomanToInt( 'MMXVI' )
RomanToInt( 'mmxvi' )

# works beyond R's limit of 3899
val.3899 <- 'MMMDCCCXCIX'
val.3900 <- 'MMMCM'
val.4000 <- 'MMMM'
as.numeric(as.roman( val.3899 ))
as.numeric(as.roman( val.3900 ))
as.numeric(as.roman( val.4000 ))

RomanToInt(val.3899)
RomanToInt(val.3900)
RomanToInt(val.4000)
```

Rotate *Rotate a Geometric Structure*

Description

Rotate a geometric structure by an angle theta around a centerpoint xy.

Usage

```
Rotate(x, y = NULL, mx = NULL, my = NULL, theta = pi/3, asp = 1)
```

Arguments

x, y	vectors containing the coordinates of the vertices of the polygon , which has to be rotated. The coordinates can be passed in a plotting structure (a list with x and y components), a two-column matrix, ... See xy.coords.
mx, my	xy-coordinates of the center of the rotation. If left to NULL, the centroid of the structure will be used.
theta	angle of the rotation
asp	the aspect ratio for the rotation. Helpful for rotate structures along an ellipse.

Value

The function invisibly returns a list of the coordinates for the rotated shape(s).

Author(s)

Andri Signorell <andri@signorell.net>

See Also

polygon, DrawRegPolygon, DrawEllipse, DrawArc

Examples

```
# let's have a triangle
Canvas(main="Rotation")
x <- DrawRegPolygon(nv=3)[[1]]

# and rotate
sapply( (0:3) * pi/6, function(theta) {
  xy <- Rotate( x=x, theta=theta )
  polygon(xy, col=SetAlpha("blue", 0.2))
} )

abline(v=0, h=0)
```

RoundTo *Round to Multiple*

Description

Returns a number rounded to the nearest specified multiple.

Usage

```
RoundTo(x, multiple = 1, FUN = round)
```

Arguments

x	numeric. The value to round.
multiple	numeric. The multiple to which the number is to be rounded. Default is 1.
FUN	the rounding function as character or as expression. Can be one out of trunc, ceiling, round (default) or floor.

Details

There are several functions to convert to integers. round rounds to the nearest integer or to any number of digits. Using a negative number rounds to a power of ten, so that round(x, -3) rounds to thousands. Each of trunc, floor and ceiling round in a fixed direction, towards zero, down and up respectively. round is documented to round to even, so round(2.5) is 2.

RoundTo uses round(x/multiple)*multiple to get the result. So if x is equally close to two multiples, the multiple with the smaller absolute value will be returned when round(x/multiple) is even (and the greater when it's odd).
If FUN is set to ceiling it will always round up, and if set to floor it will always round down. See examples for comparison).

Value

the rounded value

Author(s)

Andri Signorell <andri@signorell.net>

See Also

round, trunc, ceiling, floor

Examples

```
RoundTo(10, 3)     # Rounds 10 to a nearest multiple of 3 (9)
RoundTo(-10, -3)   # Rounds -10 to a nearest multiple of -3 (-9)

RoundTo(1.3, 0.2)  # Rounds 1.3 to a nearest multiple of 0.2 (1.2)
RoundTo(-1.3, 0.2) # Rounds -1.3 to a nearest multiple of 0.2 (-1.2)
RoundTo(5, -2)     # Returns an error, because -2 and 5 have different signs

# Round down
RoundTo(c(1,-1) * 1.2335, 0.05, floor)
RoundTo(c(1,-1) * 1233.5, 100, floor)

# Round up
RoundTo(c(1,-1) * 1.2335, 0.05, ceiling)
RoundTo(c(1,-1) * 1233.5, 100, ceiling)

# Round towards zero
RoundTo(c(1,-1) * 1.2335, 0.05, trunc)
RoundTo(c(1,-1) * 1233.5, 100, trunc)
```

```
x <- c(-1.5,-1.3, 1.3, 1.5)
cbind(x =       x,
      round =   RoundTo(x, 0.2, FUN=round),
      trunc =   RoundTo(x, 0.2, FUN=trunc),
      ceiling = RoundTo(x, 0.2, FUN=ceiling),
      floor =   RoundTo(x, 0.2, FUN=floor)
)

x <- -10:10
cbind(x =       x,
      round =   RoundTo(x, 2, FUN=round),
      trunc =   RoundTo(x, 2, FUN=trunc),
      ceiling = RoundTo(x, 2, FUN=ceiling),
      floor =   RoundTo(x, 2, FUN=floor)
)
```

`RSessionAlive` *How Long Has the RSession Been Running?*

Description

`RSessionAlive()` returns the time the R session has been running in hours. The function uses powershell in Windows and is thus restricted to run in windows only. `RTempdirAlive()` does the same for temporary directories, but runs on all systems.

Usage

```
RSessionAlive()
RTempdirAlive()
```

Value

time in hours

Author(s)

Markus Napflin <markus.naepfl@in>, Andri Signorell <andri@signorell.net>

See Also

`Sys.getenv`

RSqCI *Confidence Intervals for the R squared of a Linear Model*

Description

Calculate bootstrap intervals for the the R squared of a linear model as returned by `lm`.

Usage

```
RSqCI(
  object,
  conf.level = 0.95,
  sides = c("two.sided", "left", "right"),
  adjusted = TRUE,
  ...
)
```

Arguments

`object`	the model object as returned by glm.
`conf.level`	confidence level of the interval.
`sides`	a character string specifying the side of the confidence interval, must be one of `"two.sided"` (default), `"left"` or `"right"`. `"left"` would be analogue to a hypothesis of `"greater"` in a `t.test`. You can specify just the initial letter.
`adjusted`	logical, defining if the R squared or the adjusted R squared should be used. Default is `TRUE`, returning the latter.
`...`	further arguments are passed to the `boot` function. Supported arguments are `type` (`"norm"`, `"basic"`, `"stud"`, `"perc"`, `"bca"`), `parallel` and the number of bootstrap replicates `R`. If not defined those will be set to their defaults, being `"basic"` for `type`, option `"boot.parallel"` (and if that is not set, `"no"`) for `parallel` and `999` for `R`.

Value

a numeric vector with 3 elements:

`mean`	mean
`lwr.ci`	lower bound of the confidence interval
`upr.ci`	upper bound of the confidence interval

Author(s)

Andri Signorell andri@signorell.net

See Also

`BrierScore`

Examples

```
# get linear model
r.lm <- lm(Fertility ~ Agriculture + Examination + Education
                        + Catholic + Infant.Mortality, data=swiss)

# calculate confidence intervals for the R2
summary(r.lm)$r.squared

RSqCI(r.lm, R=99)   # use higher R in real life!
```

---

rSum21 *Random Numbers Adding Up to 1*

Description

Generates a vector of uniformly distributed random numbers which sum to 1.

Usage

```
rSum21(size, digits = NULL)
```

Arguments

size	a non-negative integer giving the number of numbers to generate.
digits	integer indicating the number of decimal places to be used.

Value

a vector of length size with elements drawn

Author(s)

Andri Signorell <andri@signorell.net>

See Also

runif, (Dirichlet distribution)

Examples

```
# generate 5 numbers
x <- rSum21(5)
sum(x)
```

`RunsTest` *Runs Test for Randomness*

Description

Performs a test whether the elements of `x` are serially independent - say, whether they occur in a random order - by counting how many runs there are above and below a threshold. If `y` is supplied a two sample Wald-Wolfowitz-Test testing the equality of two distributions against general alternatives will be computed.

Usage

```
RunsTest(x, ...)

## Default S3 method:
RunsTest(x, y = NULL, alternative = c("two.sided", "less", "greater"),
         exact = NULL, correct = TRUE, na.rm = FALSE, ...)

## S3 method for class 'formula'
RunsTest(formula, data, subset, na.action, ...)
```

Arguments

`x`	a dichotomous vector of data values or a (non-empty) numeric vector of data values.
`y`	an optional (non-empty) numeric vector of data values.
`formula`	a formula of the form `lhs ~ rhs` where `lhs` gives the data values and rhs the corresponding groups.
`data`	an optional matrix or data frame (or similar: see `model.frame`) containing the variables in the formula `formula`. By default the variables are taken from `environment(formula)`.
`subset`	an optional vector specifying a subset of observations to be used.
`na.action`	a function which indicates what should happen when the data contain NAs. Defaults to `getOption("na.action")`.
`alternative`	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"less"` or `"greater"`.
`exact`	a logical indicating whether an exact p-value should be computed. By default exact values will be calculated for small vectors with a total length <= 30 and the normal approximation for longer ones.
`correct`	a logical indicating whether to apply continuity correction when computing the test statistic. Default is `TRUE`. Ignored if `exact` is set to `TRUE`. See details.
`na.rm`	defines if `NA`s should be omitted. Default is `FALSE`.
`...`	further arguments to be passed to or from methods.

Details

The runs test for randomness is used to test the hypothesis that a series of numbers is random.

For a categorical variable, the number of runs correspond to the number of times the category changes, that is, where x_i belongs to one category and x_{i+1} belongs to the other. The number of runs is the number of sign changes plus one.

For a numeric variable x containing more than two values, a run is a set of sequential values that are either all above or below a specified cutpoint, typically the median. This is not necessarily the best choice. If another threshold should be used use a code like: `RunsTest(x > mean(x))`.

The exact distribution of runs and the p-value based on it are described in the manual of SPSS "Exact tests" https://www.sussex.ac.uk/its/pdfs/SPSS_Exact_Tests_21.pdf.

The normal approximation of the runs test is calculated with the expected number of runs under the null

$$\mu_r = \frac{2n_0n_1}{n_0 + n_1} + 1$$

and its variance

$$\sigma_r^2 = \frac{2n_0n_1(2n_0n_1 - n_0 - n_1)}{(n_0 + n_1)^2 \cdot (n_0 + n_1 - 1)}$$

as

$$\hat{z} = \frac{r - \mu_r + c}{\sigma_r}$$

where n_0, n_1 the number of values below/above the threshold and r the number of runs.

Setting the continuity correction `correct = TRUE` will yield the normal approximation as SAS (and SPSS if n < 50) does it, see http://support.sas.com/kb/33/092.html. The c is set to $c = 0.5$ if $r < \frac{2n_0n_1}{n_0+n_1} + 1$ and to $c = -0.5$ if $r > \frac{2n_0n_1}{n_0+n_1} + 1$.

The Wald-Wolfowitz test is a 2-sample nonparametric test to evaluate if two continuous cumulative distributions are significantly different or not. Ideally there should be no ties in the data. In practice there is no problem with ties within a group, but if ties occur between members of the different groups then there is no unique sequence of observations. For example the data sets A: 10,14,17,19,34 and B: 12,13,17,19,22 can give four possible sequences, with two possible values for r (7 or 9). The "solution" to this is to list every possible combination, and calculate the test statistic for each one. If all test statistics are significant at the chosen level, then one can reject the null hypothesis. If only some are significant, then Siegel (1956) suggests that the average of the P-values is taken. Help for finding all permutations of ties can be found at: https://stackoverflow.com/questions/47565066/all-possible-permutations-in-factor-variable-when-ties-exist-in-r

However this solutions seems quite coarse and in general, the test should not be used if there are more than one or two ties. We have better tests to distinguish between two samples!

Value

A list with the following components.

`statistic`	z, the value of the standardized runs statistic, if not exact p-values are computed.
`parameter`	the number of runs, the total number of zeros (m) and ones (n)
`p.value`	the p-value for the test.

data.name	a character string giving the names of the data.
alternative	a character string describing the alternative hypothesis.

Author(s)

Andri Signorell <andri@signorell.net>, exact p-values by Detlew Labes <detlewlabes@gmx.de>

References

Wackerly, D., Mendenhall, W. Scheaffer, R. L. (1986) *Mathematical Statistics with Applications*, 3rd Ed., Duxbury Press, CA.

Wald, A. and Wolfowitz, J. (1940): On a test whether two samples are from the same population, *Ann. Math Statist*. 11, 147-162.

Siegel, S. (1956) *Nonparametric Statistics for the Behavioural Sciences*, McGraw-Hill Kogakusha, Tokyo.

See Also

Run Length Encoding rle

Examples

```
# x will be coerced to a dichotomous variable
x <- c("S","S", "T", "S", "T","T","T", "S", "T")
RunsTest(x)


x <- c(13, 3, 14, 14, 1, 14, 3, 8, 14, 17, 9, 14, 13, 2, 16, 1, 3, 12, 13, 14)
RunsTest(x)
# this will be treated as
RunsTest(x > median(x))

plot( (x < median(x)) - 0.5, type="s", ylim=c(-1,1) )
abline(h=0)

set.seed(123)
x <- sample(0:1, size=100, replace=TRUE)
RunsTest(x)
# As you would expect of values from a random number generator, the test fails to reject
# the null hypothesis that the data are random.


# SPSS example
x <- c(31,23,36,43,51,44,12,26,43,75,2,3,15,18,78,24,13,27,86,61,13,7,6,8)
RunsTest(x, exact=TRUE)       # exact probability
RunsTest(x, exact=FALSE)      # normal approximation

# SPSS example small dataset
x <- c(1, 1, 1, 1, 0, 0, 0, 0, 1, 1)
RunsTest(x)
RunsTest(x, exact=FALSE)
```

```
# if y is not NULL, the Wald-Wolfowitz-Test will be performed
A <- c(35,44,39,50,48,29,60,75,49,66)
B <- c(17,23,13,24,33,21,18,16,32)

RunsTest(A, B, exact=TRUE)
RunsTest(A, B, exact=FALSE)
```

Sample *Random Samples and Permutations*

Description

Sample takes a sample of the specified size from the elements of x using either with or without replacement. The function does the same as the base::sample() and offers additionally an interface for data frames.

Usage

```
Sample(x, size, replace = FALSE, prob = NULL)
```

Arguments

x	either a vector of one or more elements from which to choose, or a positive integer.
size	a positive number, the number of items to choose from.
replace	a non-negative integer giving the number of items to choose.
prob	should sampling be with replacement?

Value

sampled elements in the same structure as x

Author(s)

Andri Signorell <andri@signorell.net>

See Also

sample

Examples

```
sample(d.pizza, size=5)
```

SampleTwins *Sample Twins*

Description

Draw a twin sample out of a population for a given recordset, by matching some strata criteria.

Usage

```
SampleTwins(x, stratanames = NULL, twins,
            method = c("srswor", "srswr", "poisson", "systematic"),
            pik, description = FALSE)
```

Arguments

x	the data to draw the sample from
stratanames	the stratanames to use
twins	the twin sample
method	method to select units; the following methods are implemented: simple random sampling without replacement (srswor), simple random sampling with replacement (srswr), Poisson sampling (poisson), systematic sampling (systematic); if "method" is missing, the default method is "srswor". See Strata.
pik	vector of inclusion probabilities or auxiliary information used to compute them; this argument is only used for unequal probability sampling (Poisson and systematic). If an auxiliary information is provided, the function uses the inclusion-probabilities function for computing these probabilities. If the method is "srswr" and the sample size is larger than the population size, this vector is normalized to one.
description	a message is printed if its value is TRUE; the message gives the number of selected units and the number of the units in the population. By default, the value is FALSE.

Value

The function produces an object, which contains the following information:

id	the identifier of the selected units.
stratum	the unit stratum.
prob	the final unit inclusion probability.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

Strata, sample

Examples

```
m <- rbind(matrix(rep("nc",165), 165, 1, byrow=TRUE),
           matrix(rep("sc", 70), 70, 1, byrow=TRUE))
m <- cbind.data.frame(m, c(rep(1, 100), rep(2,50), rep(3,15),
                           rep(1,30), rep(2,40)), 1000*runif(235))
names(m) <- c("state","region","income")

# this would be our sample to be reproduced by a twin sample
d.smp <- m[sample(nrow(m), size=10, replace=TRUE),]

# draw the sample
s <- SampleTwins(x = m, stratanames=c("state","region"), twins = d.smp, method="srswor")

d.twin <- m[s$id,]
d.twin
```

SaveAs *Saves an R Object Under a Different Name*

Description

An R object cannot be saved in binary mode under a different name using the default `save()` function. `SaveAs()` extends the save function for this option.

Usage

```
SaveAs(x, objectname, file, ...)
```

Arguments

`x`	the object to save
`objectname`	the new name for the object.
`file`	a (writable binary-mode) connection or the name of the file where the data will be saved (when tilde expansion is done).
`...`	the dots are passed to the save function.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`save`

Examples

```
x <- stats::runif(20)
SaveAs(x=x, objectname="NewX", file = "NewXFile.rda")
unlink("NewXFile.rda")
```

ScheffeTest *Scheffe Test for Pairwise and Otherwise Comparisons*

Description

Scheffe's method applies to the set of estimates of all possible contrasts among the factor level means, not just the pairwise differences considered by Tukey's method.

Usage

```
ScheffeTest(x, ...)

## S3 method for class 'formula'
ScheffeTest(formula, data, subset, na.action, ...)
## S3 method for class 'aov'
ScheffeTest(x, which = NULL, contrasts = NULL,
            conf.level = 0.95, ...)
## Default S3 method:
ScheffeTest(x, g = NULL, which = NULL,
            contrasts = NULL, conf.level = 0.95, ...)
```

Arguments

x	either a fitted model object, usually an aov fit, when g is left to NULL or a response variable to be evalutated by g (which mustn't be NULL then).
g	the grouping variable.
which	character vector listing terms in the fitted model for which the intervals should be calculated. Defaults to all the terms.
contrasts	a $r \times c$ matrix containing the contrasts to be computed, while r is the number of factor levels and c the number of contrasts. Each column must contain a full contrast ("sum") adding up to 0. Note that the argument which must be defined, when non default contrasts are used. Default value of contrasts is NULL. In this case all pairwise contrasts will be reported.
conf.level	numeric value between zero and one giving the confidence level to use. If this is set to NA, just a matrix with the p-values will be returned.
formula	a formula of the form lhs ~ rhs where lhs gives the data values and rhs the corresponding groups.
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
...	further arguments, currently not used.

Value

A list of classes `c("PostHocTest")`, with one component for each term requested in `which`. Each component is a matrix with columns `diff` giving the difference in the observed means, `lwr.ci` giving the lower end point of the interval, `upr.ci` giving the upper end point and `pval` giving the p-value after adjustment for the multiple comparisons.

There are print and plot methods for class `"PostHocTest"`. The plot method does not accept `xlab`, `ylab` or `main` arguments and creates its own values for each plot.

Author(s)

Andri Signorell <andri@signorell.net>

References

Robert O. Kuehl, Steel R. (2000) *Design of experiments*. Duxbury

Steel R.G.D., Torrie J.H., Dickey, D.A. (1997) *Principles and Procedures of Statistics, A Biometrical Approach*. McGraw-Hill

See Also

`pairwise.t.test`, `TukeyHSD`

Examples

```
fm1 <- aov(breaks ~ wool + tension, data = warpbreaks)

ScheffeTest(x=fm1)
ScheffeTest(x=fm1, which="tension")

TukeyHSD(fm1)

# some special contrasts
y <- c(7,33,26,27,21,6,14,19,6,11,11,18,14,18,19,14,9,12,6,
       24,7,10,1,10,42,25,8,28,30,22,17,32,28,6,1,15,9,15,
       2,37,13,18,23,1,3,4,6,2)
group <- factor(c(1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,3,3,3,3,3,
       3,3,3,4,4,4,4,4,4,4,4,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6))

r.aov <- aov(y ~ group)

ScheffeTest(r.aov, contrasts=matrix( c(1,-0.5,-0.5,0,0,0,
                                       0,0,0,1,-0.5,-0.5), ncol=2) )

# just p-values:
ScheffeTest(r.aov, conf.level=NA)
```

SD *(Weighted) Standard Deviation*

Description

This function computes the standard deviation of the values in `x`. If `na.rm` is `TRUE` then missing values are removed before computation proceeds. `SDn` returns the uncorrected sample standard deviation (which is biased estimator for the sample standard deviation).

Usage

```
SD(x, weights = NULL, na.rm = FALSE, ...)

SDN(x, na.rm = FALSE)
```

Arguments

`x`	a numeric vector or an R object which is coercible to one by `as.double(x)`.
`weights`	a numerical vector of weights the same length as `x` giving the weights to use for elements of `x`.
`na.rm`	logical. Should missing values be removed?
`...`	further arguments passed to or from other methods.

Details

Like `var` this uses denominator $n - 1$.

The standard deviation of a zero-length vector (after removal of `NA`s if `na.rm = TRUE`) is not defined and gives an error. The standard deviation of a length-one vector is `NA`.

See Also

`var` for its square, and `mad`, the most robust alternative.

Examples

```
SD(1:2)^2
```

SendOutlookMail *Send a Mail Using Outlook as Mail Client*

Description

Sending emails in R can be required in some reporting tasks. As we already have RDCOMClient available we wrap the send code in a function.

Usage

```
SendOutlookMail(to, cc = NULL, bcc = NULL, subject, body, attachment = NULL)
```

Arguments

to	a vector of recipients
cc	a vector of recipients receiving a carbon copy
bcc	a vector of recipients receiving a blind carbon copy
subject	the subject of the mail
body	the body text of the mail
attachment	a vector of paths to attachments

Value

Nothing is returned

Author(s)

Andri Signorell <andri@signorell.net> strongly based on code of Franziska Mueller

See Also

ToXL

Examples

```
## Not run:
SendOutlookMail(to=c("me@microsoft.com", "you@rstudio.com"), subject = "Some Info",
                body = "Hi all\r Find the files attached\r Regards, Andri",
                attachment = c("C:/temp/fileA.txt",
                               "C:/temp/fileB.txt"))

## End(Not run)
```

SetAlpha *Add an Alpha Channel To a Color*

Description

Add transparency to a color defined by its name or number. The function first converts the color to RGB and then appends the alpha channel. `Fade()` combines `ColToOpaque(SetAlpha(col))`.

Usage

```
SetAlpha(col, alpha = 0.5)
Fade(col, ...)
```

Arguments

`col`	vector of two kind of R colors, i.e., either a color name (an element of `colors()`) or an integer i meaning `palette()[i]`.
`alpha`	the alpha value to be added. This can be any value from 0 (fully transparent) to 1 (opaque). `NA` is interpreted so as to delete a potential alpha channel. Default is 0.5.
`...`	the dots in `Fade` are passed on to `SetAlpha`.

Details

All arguments are recyled as necessary.

Value

Vector with the same length as `col`, giving the rgb-values extended by the alpha channel as hex-number (#rrggbbaa).

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`ColToHex`, `col2rgb`, `adjustcolor`, `ColToOpaque`

Examples

```
SetAlpha("yellow", 0.2)
SetAlpha(2, 0.5)   # red

Canvas(3)
DrawCircle(x=c(-1,0,1), y=c(1,-1,1), r.out=2, col=SetAlpha(2:4, 0.4))

x <- rnorm(15000)
```

```
par(mfrow=c(1,2))
plot(x, type="p", col="blue" )
plot(x, type="p", col=SetAlpha("blue", .2), main="Better insight with alpha channel" )
```

SetNames *Set the Names in an Object*

Description

This is a convenience function that sets the names of an object and returns it including the new names. It is most useful at the end of a function definition where one is creating the object to be returned and would prefer not to store it under a name just that the names can be assigned. In addition to the function setNames in base R the user can decide, whether rownames, colnames or simply the names are to be set. Names are recyled.

Usage

```
SetNames(x, ...)
```

Arguments

x	an object for which a names attribute will be meaningful
...	the names to be assigned to the object. This should be a character vector of names named dimnames, rownames, colnames or names. Setting rownames=NULL would remove existing rownames. All kind of names can be changed at the same time. Default would be names. Abbreviations are supported.

Value

An object of the same sort as object with the new names assigned.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

setNames, Rename

Examples

```
SetNames(1:5, names=letters[1:5])

# the default, if no argument names are provided, is "names"
SetNames(1:5, letters[1:5])

tab <- table(d.pizza$driver, d.pizza$wine_delivered)
```

```
# rownames and columnnames can be set at the same time
SetNames(BinomCI(tab[,1], rowSums(tab)),
         rownames=rownames(tab), colnames=c("perc", "lci", "uci"))

# can also be used to set the names to an empty string
SetNames(diag(6), rownames="", colnames="")

# setting dimnames works as well
tab <- SetNames(
  as.table(rbind(c(84,43), c(10,92))),
    dimnames= list(
       dipstick=c("positive","negative"),
       culture=c("positive","negative")))
```

Shade *Produce a Shaded Curve*

Description

Sometimes the area under a density curve has to be color shaded, for instance to illustrate a p-value or a specific region under the normal curve. This function draws a curve corresponding to a function over the interval `[from, to]`. It can plot also an expression in the variable `xname`, default `x`.

Usage

```
Shade(expr, col = par("fg"), breaks, density = 10, n = 101, xname = "x", ...)
```

Arguments

`expr`	the name of a function, or a `call` or an `expression` written as a function of `x` which will evaluate to an object of the same length as `x`.
`col`	color to fill or shade the shape with. The default is taken from `par("fg")`.
`breaks`	numeric, a vector giving the breakpoints between the distinct areas to be shaded differently. Should be finite as there are no plots with infinite limits.
`density`	the density of the lines as needed in polygon.
`n`	integer; the number of x values at which to evaluate. Default is 101.
`xname`	character string giving the name to be used for the x axis.
`...`	the dots are passed on to `polygon`.

Details

Useful for shading the area under a curve as often needed for explaining significance tests.

Value

A list with components `x` and `y` of the points that were drawn is returned invisibly.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

polygon, curve

Examples

```
curve(dt(x, df=5), xlim=c(-6,6),
    main=paste("Student t-Distribution Probability Density Function, df = ", 5, ")", sep=""),
      type="n", las=1, ylab="probability", xlab="t")

Shade(dt(x, df=5), breaks=c(-6, qt(0.025, df=5), qt(0.975, df=5), 6),
      col=c(DescTools::hred, DescTools::hblue), density=c(20, 7))
```

ShapiroFranciaTest *Shapiro-Francia Test for Normality*

Description

Performs the Shapiro-Francia test for the composite hypothesis of normality.

Usage

```
ShapiroFranciaTest(x)
```

Arguments

x	a numeric vector of data values, the number of which must be between 5 and 5000. Missing values are allowed.

Details

The test statistic of the Shapiro-Francia test is simply the squared correlation between the ordered sample values and the (approximated) expected ordered quantiles from the standard normal distribution. The p-value is computed from the formula given by Royston (1993).

Value

A list of class htest, containing the following components:

statistic	the value of the Shapiro-Francia statistic.
p.value	the p-value for the test.
method	the character string "Shapiro-Francia normality test".
data.name	a character string giving the name(s) of the data.

Note

The Shapiro-Francia test is known to perform well, see also the comments by Royston (1993). The expected ordered quantiles from the standard normal distribution are approximated by `qnorm(ppoints(x, a = 3/8))`, being slightly different from the approximation `qnorm(ppoints(x, a = 1/2))` used for the normal quantile-quantile plot by `qqnorm` for sample sizes greater than 10.

Author(s)

Juergen Gross <gross@statistik.uni-dortmund.de>

References

Royston, P. (1993): A pocket-calculator algorithm for the Shapiro-Francia test for non-normality: an application to medicine. *Statistics in Medicine*, 12, 181–184.

Thode Jr., H.C. (2002): *Testing for Normality*. Marcel Dekker, New York. (2002, Sec. 2.3.2)

See Also

`shapiro.test` for performing the Shapiro-Wilk test for normality. `AndersonDarlingTest`, `CramerVonMisesTest`, `LillieTest`, `PearsonTest` for performing further tests for normality. `qqnorm` for producing a normal quantile-quantile plot.

Examples

```
ShapiroFranciaTest(rnorm(100, mean = 5, sd = 3))
ShapiroFranciaTest(runif(100, min = 2, max = 4))
```

SiegelTukeyTest *Siegel-Tukey Test For Equality In Variability*

Description

Non-parametric Siegel-Tukey test for equality in variability. The null hypothesis is that the variability of x is equal between two groups. A rejection of the null hypothesis indicates that variability differs between the two groups. `SiegelTukeyRank` returns the ranks, calculated after Siegel Tukey logic.

Usage

```
SiegelTukeyTest(x, ...)

## Default S3 method:
SiegelTukeyTest(x, y, adjust.median = FALSE,
                alternative = c("two.sided", "less", "greater"),
                mu = 0, exact = NULL, correct = TRUE, conf.int = FALSE,
                conf.level = 0.95, ...)
```

```
## S3 method for class 'formula'
SiegelTukeyTest(formula, data, subset, na.action, ...)


SiegelTukeyRank(x, g, drop.median = TRUE)
```

Arguments

x, y	numeric vector of data values. Non-finite (e.g. infinite or missing) values will be omitted.
g	a vector or factor object giving the group for the corresponding elements of x.
adjust.median	Should between-group differences in medians be leveled before performing the test? In certain cases, the Siegel-Tukey test is susceptible to median differences and may indicate significant differences in variability that, in reality, stem from differences in medians. Default is FALSE.
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter.
mu	a number specifying an optional parameter used to form the null hypothesis. See Details.
exact	a logical indicating whether an exact p-value should be computed. This is passed directly to wilcox.test.
correct	a logical indicating whether to apply continuity correction in the normal approximation for the p-value.
conf.int	a logical indicating whether a confidence interval should be computed.
conf.level	confidence level of the interval.
formula	a formula of the form lhs ~ rhs where lhs gives the data values and rhs the corresponding groups.
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
drop.median	logical, defining whether the median of the combined samples should be left out, ensuring that there's an even number of elements (which is a requirement of the Siegel-Tukey test). Defaults to TRUE.
...	further arguments to be passed to or from methods.

Details

The Siegel-Tukey test has relatively low power and may, under certain conditions, indicate significance due to differences in medians rather than differences in variabilities (consider using the argument adjust.median). Consider also using mood.test or ansari.test.

Value

A list of class `htest`, containing the following components:

`statistic`	Siegel-Tukey test (Wilcoxon test on tie-adjusted Siegel-Tukey ranks, after the median adjustment if specified).
`p.value`	the p-value for the test
`null.value`	is the value of the median specified by the null hypothesis. This equals the input argument `mu`.
`alternative`	a character string describing the alternative hypothesis.
`method`	the type of test applied
`data.name`	a character string giving the names of the data.

Author(s)

Daniel Malter, Tal Galili <tal.galili@gmail.com>, Andri Signorell <andri@signorell.net>
published on: https://www.r-statistics.com/2010/02/siegel-tukey-a-non-parametric-test-for-equality-in

References

Siegel, S., Tukey, J. W. (1960): A nonparametric sum of ranks procedure for relative spread in unpaired samples. *Journal of the American Statistical Association*.

Sheskin, D. J. (2004): *Handbook of parametric and nonparametric statistical procedures* 3rd edition. Chapman and Hall/CRC. Boca Raton, FL.

See Also

`mood.test`, `ansari.test`, `wilcox.test`, `LeveneTest`

Examples

```
# Duller, S. 183
x <- c(12, 13, 29, 30)
y <- c(15, 17, 18, 24, 25, 26)
SiegelTukeyTest(x, y)
SiegelTukeyTest(x, y, alternative="greater")

# Duller, S. 323
old <- c(870,930,935,1045,1050,1052,1055)
new <- c(932,970,980,1001,1009,1030,1032,1040,1046)
SiegelTukeyTest(old, new, alternative = "greater")
# compare to the recommended alternatives
mood.test(old, new, alternative="greater")
ansari.test(old, new, alternative="greater")

# Bortz, S. 250
x <- c(26.3,26.5,26.8,27.0,27.0,27.2,27.3,27.3,27.4,27.5,27.6,27.8,27.9)
id <- c(2,2,2,1,2,2,1,2,2,1,1,1,2)-1
SiegelTukeyTest(x ~ id)
```

```
# Sachs, Angewandte Statistik, 12. Auflage, 2007, S. 314
A <- c(10.1,7.3,12.6,2.4,6.1,8.5,8.8,9.4,10.1,9.8)
B <- c(15.3,3.6,16.5,2.9,3.3,4.2,4.9,7.3,11.7,13.1)
SiegelTukeyTest(A, B)



### 1
x <- c(4,4,5,5,6,6)
y <- c(0,0,1,9,10,10)
SiegelTukeyTest(x, y)

### 2
# example for a non equal number of cases:
x <- c(4,4,5,5,6,6)
y <- c(0,0,1,9,10)
SiegelTukeyTest(x, y)

### 3
x <- c(33, 62, 84, 85, 88, 93, 97, 4, 16, 48, 51, 66, 98)
id <- c(0,0,0,0,0,0,0,1,1,1,1,1,1)
SiegelTukeyTest(x ~ id)

### 4
x <- c(177,200,227,230,232,268,272,297,47,105,126,142,158,172,197,220,225,230,262,270)
id <- c(rep(0,8),rep(1,12))
SiegelTukeyTest(x ~ id, adjust.median=TRUE)

### 5
x <- c(33,62,84,85,88,93,97)
y <- c(4,16,48,51,66,98)
SiegelTukeyTest(x, y)

### 6
x <- c(0,0,1,4,4,5,5,6,6,9,10,10)
id <- c(0,0,0,1,1,1,1,1,1,0,0,0)
SiegelTukeyTest(x ~ id)

### 7
x <- c(85,106,96, 105, 104, 108, 86)
id <- c(0,0,1,1,1,1,1)
SiegelTukeyTest(x ~ id)
```

SignTest *Sign Test*

Description

Performs one- and two-sample sign tests on vectors of data.

Usage

```
SignTest(x, ...)

## Default S3 method:
SignTest(x, y = NULL, alternative = c("two.sided", "less", "greater"),
         mu = 0, conf.level = 0.95, ... )

## S3 method for class 'formula'
SignTest(formula, data, subset, na.action, ...)
```

Arguments

x	numeric vector of data values. Non-finite (e.g. infinite or missing) values will be omitted.
y	an optional numeric vector of data values: as with x non-finite values will be omitted.
mu	a number specifying an optional parameter used to form the null hypothesis. See Details.
alternative	is a character string, one of "greater", "less", or "two.sided", or the initial letter of each, indicating the specification of the alternative hypothesis. For one-sample tests, alternative refers to the true median of the parent population in relation to the hypothesized value of the median.
conf.level	confidence level for the returned confidence interval, restricted to lie between zero and one.
formula	a formula of the form lhs ~ rhs where lhs gives the data values and rhs the corresponding groups.
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
...	further arguments to be passed to or from methods.

Details

The formula interface is only applicable for the 2-sample test.

SignTest computes a "Dependent-samples Sign-Test" if both x and y are provided. If only x is provided, the "One-sample Sign-Test" will be computed.

For the one-sample sign-test, the null hypothesis is that the median of the population from which x is drawn is mu. For the two-sample dependent case, the null hypothesis is that the median for the differences of the populations from which x and y are drawn is mu. The alternative hypothesis indicates the direction of divergence of the population median for x from mu (i.e., "greater", "less", "two.sided".)

The confidence levels are exact.

Value

A list of class `htest`, containing the following components:

`statistic`	the S-statistic (the number of positive differences between the data and the hypothesized median), with names attribute "S".
`parameter`	the total number of valid differences.
`p.value`	the p-value for the test.
`null.value`	is the value of the median specified by the null hypothesis. This equals the input argument `mu`.
`alternative`	a character string describing the alternative hypothesis.
`method`	the type of test applied.
`data.name`	a character string giving the names of the data.
`conf.int`	a confidence interval for the median.
`estimate`	the sample median.

Author(s)

Andri Signorell <andri@signorell.net>

References

Gibbons, J.D. and Chakraborti, S. (1992): *Nonparametric Statistical Inference*. Marcel Dekker Inc., New York.

Kitchens, L. J. (2003): *Basic Statistics and Data Analysis*. Duxbury.

Conover, W. J. (1980): *Practical Nonparametric Statistics, 2nd ed*. Wiley, New York.

See Also

`t.test`, `wilcox.test`, `ZTest`, `binom.test`, `SIGN.test` in the package **BSDA** (reporting approximative confidence intervals).

Examples

```
x <- c(1.83,  0.50,  1.62,  2.48, 1.68, 1.88, 1.55, 3.06, 1.30)
y <- c(0.878, 0.647, 0.598, 2.05, 1.06, 1.29, 1.06, 3.14, 1.29)

SignTest(x, y)
wilcox.test(x, y, paired = TRUE)


d.light <- data.frame(
  black = c(25.85,28.84,32.05,25.74,20.89,41.05,25.01,24.96,27.47),
  white <- c(18.23,20.84,22.96,19.68,19.5,24.98,16.61,16.07,24.59),
  d <- c(7.62,8,9.09,6.06,1.39,16.07,8.4,8.89,2.88)
)

d <- d.light$d
```

```
SignTest(x=d, mu = 4)
wilcox.test(x=d, mu = 4, conf.int = TRUE)

SignTest(x=d, mu = 4, alternative="less")
wilcox.test(x=d, mu = 4, conf.int = TRUE, alternative="less")

SignTest(x=d, mu = 4, alternative="greater")
wilcox.test(x=d, mu = 4, conf.int = TRUE, alternative="greater")

# test die interfaces
x <- runif(10)
y <- runif(10)
g <- rep(1:2, each=10)
xx <- c(x, y)

SignTest(x ~ group, data=data.frame(x=xx, group=g ))
SignTest(xx ~ g)
SignTest(x, y)

SignTest(x - y)
```

SmoothSpline *Formula Interface For* smooth.spline

Description

smooth.spline has no formula interface, which is sometimes inconvenient, if one simply wants to copy a formula of a linear model or a plot to spline.

Usage

```
SmoothSpline(x, ...)

## Default S3 method:
SmoothSpline(x, y = NULL, w = NULL, df, spar = NULL, cv = FALSE,
             all.knots = FALSE, nknots = .nknots.smspl, keep.data = TRUE,
             df.offset = 0, penalty = 1, control.spar = list(),
             tol = 0.000001 * IQR(x), ...)

## S3 method for class 'formula'
SmoothSpline(formula, data, subset, na.action, ...)
```

Arguments

x	a vector giving the values of the predictor variable, or a list or a two-column matrix specifying x and y.

y	responses. If y is missing or NULL, the responses are assumed to be specified by x, with x the index vector.
w	optional vector of weights of the same length as x; defaults to all 1.
df	the desired equivalent number of degrees of freedom (trace of the smoother matrix).
spar	smoothing parameter, typically (but not necessarily) in $(0, 1]$. The coefficient λ of the integral of the squared second derivative in the fit (penalized log likelihood) criterion is a monotone function of spar, see the details below.
cv	ordinary (TRUE) or 'generalized' cross-validation (GCV) when FALSE; setting it to NA skips the evaluation of leverages and any score.
all.knots	if TRUE, all distinct points in x are used as knots. If FALSE (default), a subset of x[] is used, specifically x[j] where the nknots indices are evenly spaced in 1:n, see also the next argument nknots.
nknots	integer or function giving the number of knots to use when all.knots = FALSE. If a function (as by default), the number of knots is nknots(nx). By default for $n_x > 49$ this is less than n_x, the number of unique x values, see the Note.
keep.data	logical specifying if the input data should be kept in the result. If TRUE (as per default), fitted values and residuals are available from the result.
df.offset	allows the degrees of freedom to be increased by df.offset in the GCV criterion.
penalty	the coefficient of the penalty for degrees of freedom in the GCV criterion.
control.spar	optional list with named components controlling the root finding when the smoothing parameter spar is computed, i.e., missing or NULL, see below. **Note** that this is partly *experimental* and may change with general spar computation improvements! **low:** lower bound for spar; defaults to -1.5 (used to implicitly default to 0 in R versions earlier than 1.4). **high:** upper bound for spar; defaults to +1.5. **tol:** the absolute precision (**tol**erance) used; defaults to 1e-4 (formerly 1e-3). **eps:** the relative precision used; defaults to 2e-8 (formerly 0.00244). **trace:** logical indicating if iterations should be traced. **maxit:** integer giving the maximal number of iterations; defaults to 500. Note that spar is only searched for in the interval $[low, high]$.
tol	a tolerance for same-ness or uniqueness of the x values. The values are binned into bins of size tol and values which fall into the same bin are regarded as the same. Must be strictly positive (and finite).
formula	a formula of the form lhs ~ rhs where lhs gives the data values and rhs the corresponding groups.
data	The data frame from which the formula should be evaluated.
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
...	Other arguments to be passed to smooth.spline.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

smooth.spline, lines.smooth.spline

Examples

```
plot(temperature ~ delivery_min, data=d.pizza)
lines(SmoothSpline(temperature ~ delivery_min, data=d.pizza))
```

Some *Return Some Randomly Chosen Elements of an Object*

Description

For displaying the first and last elements of an object there are the functions head and tail. Sometimes one might want to see more randomly scattered elements. This function returns some random parts of a vector, matrix or a data frame. The order of the elements within the object will be preserved.

Usage

```
Some(x, n = 6L, ...)
## Default S3 method:
Some(x, n = 6L, ...)
## S3 method for class 'data.frame'
Some(x, n = 6L, ...)
## S3 method for class 'matrix'
Some(x, n = 6L, addrownums = TRUE, ...)
```

Arguments

x	an object
n	a single integer. If positive, size for the resulting object: number of elements for a vector (including lists), rows for a matrix or data frame or lines for a function. If negative, all but the n last/first number of elements of x.
addrownums	if there are no row names, create them from the row numbers.
...	arguments to be passed to or from other methods.

Details

For matrices, 2-dim tables and data frames, `Some()` returns some `n` rows when `n > 0` or all but the some `n` rows when `n < 0`. `Some.matrix()` is not exported (unlike `head.matrix`).

If a matrix has no row names, then `Some()` will add row names of the form `"[n,]"` to the result, so that it looks similar to the last lines of `x` when printed. Setting `addrownums = FALSE` suppresses this behaviour.

I desisted from implementing interfaces for tables, ftables and functions, as this would not make much sense.

Value

An object (usually) like `x` but generally smaller.

Author(s)

Andri Signorell, basically copying and just slightly modifying Patrick Burns and R-Core code.

See Also

`head`

Examples

```
Some(letters)
Some(letters, n = -6L)

Some(freeny.x, n = 10L)
Some(freeny.y)
```

Some numeric checks *Check a Vector For Being Numeric, Zero Or a Whole Number*

Description

Test if x contains only integer numbers, or if is numeric or if it is zero.

Usage

```
IsWhole(x, all = FALSE, tol = sqrt(.Machine$double.eps), na.rm = FALSE)
IsZero(x, tol = sqrt(.Machine$double.eps), na.rm = FALSE)
IsNumeric(x, length.arg = Inf, integer.valued = FALSE, positive = FALSE, na.rm = FALSE)
```

Arguments

`x`	a (non-empty) numeric vector of data values.
`all`	logical, specifying if the whole vector should be checked. If set to `TRUE` the function will return the result of `all(IsWhole(x))`.
`tol`	tolerance to be used
`length.arg`	integer, the length of the vector to be checked for.
`integer.valued`	logical, should x be checked as integer?
`positive`	logical, is x supposed to be positive?
`na.rm`	logical, indicating whether `NA` values should be stripped before the computation proceeds. Defaults to `FALSE`.

Details

IsWhole is the suggested solution for checking for an integer value, as `is.integer` tests for `class(x) == "integer"` and does NOT test whether x (which might be of class "numeric") contains only integer numbers. (Why not simply implement it in **base**?)

IsZero tests float numeric values for being zero.

IsNumeric combines a test for numeric and integers.

Value

logical vector of the same dimension as x.

Author(s)

R-Core, Andri Signorell <andri@signorell.net>, Thomas W. Yee

See Also

`is.integer`

Examples

```
(x <- seq(1,5, by=0.5))
IsWhole( x ) #-->  \code{TRUE} \code{FALSE} \code{TRUE} ...


# ... These are people who live in ignorance of the Floating Point Gods.
# These pagans expect ... (Burns, 2011)" the following to be TRUE:
(.1 - .3 / 3) == 0

# they might be helped by
IsZero(.1 - .3 / 3)
```

SomersDelta *Somers' Delta*

Description

Calculate Somers' Delta statistic, a measure of association for ordinal factors in a two-way table. The function has interfaces for a table (matrix) and for single vectors.

Usage

```
SomersDelta(x, y = NULL, direction = c("row", "column"), conf.level = NA, ...)
```

Arguments

x	a numeric vector or a table. A matrix will be treated as table.
y	NULL (default) or a vector with compatible dimensions to x. If y is provided, table(x, y, ...) is calculated.
direction	direction of the calculation. Can be "row" (default) or "column", where "row" calculates Somers' D (R | C) ("column dependent").
conf.level	confidence level of the interval. If set to NA (which is the default) no confidence interval will be calculated.
...	further arguments are passed to the function table, allowing i.e. to set useNA. This refers only to the vector interface.

Details

Somers' D(C|R) and Somers' D(R|C) are asymmetric modifications of τ_b and Goodman-Kruskal's Gamma. C|R indicates that the row variable x is regarded as the independent variable and the column variable y is regarded as dependent. Similarly, R|C indicates that the column variable y is regarded as the independent variable and the row variable x is regarded as dependent. It is logically very similar to Gamma, but differs in that it uses a correction only for pairs that are tied on the dependent variable. As Gamma and the Taus, D is appropriate only when both variables lie on an ordinal scale.
Somers' D is computed as

$$D(C|R) = \frac{P - Q}{n^2 - \sum(n_i.^2)}$$

where P equals twice the number of concordances and Q twice the number of discordances and $n_i.$ rowSums(tab). Its range lies [-1, 1]. The interpretation of d is analogous to Gamma.

Value

a single numeric value if no confidence intervals are requested
and otherwise a numeric vector with 3 elements for the estimate, the lower and the upper confidence interval

Author(s)

Andri Signorell <andri@signorell.net>

References

Agresti, A. (2002) *Categorical Data Analysis*. John Wiley & Sons, pp. 57–59.

Brown, M.B., Benedetti, J.K.(1977) Sampling Behavior of Tests for Correlation in Two-Way Contingency Tables, *Journal of the American Statistical Association*, 72, 309-315.

Goodman, L. A., & Kruskal, W. H. (1954) Measures of association for cross classifications. *Journal of the American Statistical Association*, 49, 732-764.

Somers, R. H. (1962) A New Asymmetric Measure of Association for Ordinal Variables, *American Sociological Review*, 27, 799–811.

Goodman, L. A., & Kruskal, W. H. (1963) Measures of association for cross classifications III: Approximate sampling theory. *Journal of the American Statistical Association*, 58, 310–364.

See Also

There's an implementation of Somers's D in Frank Harrell's **Hmisc** `somers2`, which is quite fast for large sample sizes. However it is restricted to computing Somers' Dxy rank correlation between a variable x and a binary (0-1) variable y.
`ConDisPairs` yields concordant and discordant pairs

Other association measures:
`KendallTauA` (tau-a), `KendallTauB` (tau-b), `cor` (method="kendall") for tau-b, `StuartTauC` (tau-c), `GoodmanKruskalGamma`
`Lambda`, `GoodmanKruskalTau`, `UncertCoef`, `MutInf`

Examples

```
# example in:
# http://support.sas.com/documentation/cdl/en/statugfreq/63124/PDF/default/statugfreq.pdf
# pp. S. 1821

tab <- as.table(rbind(c(26,26,23,18,9),c(6,7,9,14,23)))

# Somers' D C|R
SomersDelta(tab, direction="column", conf.level=0.95)
# Somers' D R|C
SomersDelta(tab, direction="row", conf.level=0.95)
```

Sort *Sort a Vector, a Matrix, a Table or a Data.frame*

Description

Sort a vector, a matrix, a table or a data.frame. The base sort function does not have an interface for classes other than vectors and coerces the whole world to a vector. This means you get a sorted vector as result while passing a matrix to `sort`.
`Sort` wraps the base sort function and adds an interface for sorting the rows of the named 2-dimensional data structures by the order of one or more of its columns.

Usage

```
Sort(x, ...)

## Default S3 method:
Sort(x, ...)
## S3 method for class 'matrix'
Sort(x, ord = NULL, decreasing = FALSE, na.last = TRUE, ...)
## S3 method for class 'table'
Sort(x, ord = NULL, decreasing = FALSE, na.last = TRUE, ...)
## S3 method for class 'data.frame'
Sort(x, ord = NULL, decreasing = FALSE,
                factorsAsCharacter = TRUE, na.last = TRUE, ...)
```

Arguments

`x`	a numeric, complex. character or logical vector, a factor, a table or a data.frame to be sorted.
`decreasing`	logical. Should the sort be increasing or decreasing?
`factorsAsCharacter`	logical. Should factors be sorted by the alphabetic order of their labels or by the order or their levels. Default is `TRUE` (by labels).
`ord`	vector of integers or columnames. Defines the columns in a table, in a matrix or in a data.frame to be sorted for. 0 means row.names, 1:n the columns and n+1 the marginal sum. See examples.
`na.last`	for controlling the treatment of `NA`s. If `TRUE`, missing values in the data are put last; if `FALSE`, they are put first; if `NA`, they are removed (see `order`.)
`...`	further arguments to be passed to or from methods.

Details

The sort order for factors is the order of their levels (which is particularly appropriate for ordered factors), and usually confusing for unordered factors, whose levels may be defined in the sequence in which they appear in the data (which normally is unordered).

Value

the sorted object.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

sort, order

Examples

```
d.frm <- d.pizza[1:10, c("driver","temperature","delivery_min")]

Sort(d.frm[,1])
# Sort follows the levels by default
levels(d.frm[,1])

Sort(x=d.frm, ord="driver", decreasing=FALSE)
# set factorsAsCharacter = TRUE, if alphabetical order is required
Sort(x=d.frm, ord="driver", decreasing=FALSE, factorsAsCharacter=TRUE)

Sort(x=d.frm, ord=c("driver","delivery_min"), factorsAsCharacter = TRUE)
Sort(x=d.frm, ord=c("driver","delivery_min"), factorsAsCharacter = FALSE)

Sort(x=d.frm, ord=c("driver","delivery_min"), decreasing=c(FALSE, TRUE),
  factorsAsCharacter = FALSE)

# Sorting tables
tab <- table(d.pizza$driver, d.pizza$area)

Sort(x=tab, ord=c(0,2), decreasing=c(TRUE, FALSE))
Sort(x=tab, ord=2, decreasing=TRUE)

# partial matching ok:
Sort(tab, o=1, d=TRUE)
```

SortMixed *Sort Strings with Embedded Numbers Based on Their Numeric Order*

Description

These functions sort or order character strings containing embedded numbers so that the numbers are numerically sorted rather than sorted by character value. I.e. "Asprin 50mg" will come before "Asprin 100mg". In addition, case of character strings is ignored so that "a", will come before "B" and "C".

Usage

```
SortMixed(x, decreasing=FALSE, na.last=TRUE, blank.last=FALSE,
          numeric.type=c("decimal", "roman"),
          roman.case=c("upper","lower","both") )
```

```
OrderMixed(x, decreasing=FALSE, na.last=TRUE, blank.last=FALSE,
          numeric.type=c("decimal", "roman"),
          roman.case=c("upper","lower","both") )
```

Arguments

x	vector to be sorted.
decreasing	logical. Should the sort be increasing or decreasing? Note that descending=TRUE reverses the meanings of na.last and blanks.last.
na.last	logical, controlling the treatment of NA values. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA, they are removed.
blank.last	logical, controlling the treatment of blank values. If TRUE, blank values in the data are put last; if FALSE, they are put first; if NA, they are removed.
numeric.type	either "decimal" (default) or "roman". Are numeric values represented as decimal numbers (numeric.type="decimal") or as Roman numerals (numeric.type="roman")?
roman.case	one of "upper", "lower", or "both". Are roman numerals represented using only capital letters ('IX') or lower-case letters ('ix') or both?

Details

I often have character vectors (e.g. factor labels), such as compound and dose, that contain both text and numeric data. This function is useful for sorting these character vectors into a logical order.

It does so by splitting each character vector into a sequence of character and numeric sections, and then sorting along these sections, with numbers being sorted by numeric value (e.g. "50" comes before "100"), followed by characters strings sorted by character value (e.g. "A" comes before "B") *ignoring case* (e.g. 'A' has the same sort order as 'a').

By default, sort order is ascending, empty strings are sorted to the front, and NA values to the end. Setting descending=TRUE changes the sort order to descending and reverses the meanings of na.last and blank.last.

Parsing looks for decimal numbers unless numeric.type="roman", in which parsing looks for roman numerals, with character case specified by roman.case.

Value

OrderMixed returns a vector giving the sort order of the input elements. SortMixed returns the sorted vector.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

sort, order

Examples

```
## compound & dose labels
Treatment <- c("Control", "Asprin 10mg/day", "Asprin 50mg/day",
               "Asprin 100mg/day", "Acetomycin 100mg/day",
               "Acetomycin 1000mg/day")

## ordinary sort puts the dosages in the wrong order
sort(Treatment)

## but SortMixed does the 'right' thing
SortMixed(Treatment)

## Here is a more complex example
x <- rev(c("AA 0.50 ml", "AA 1.5 ml", "AA 500 ml", "AA 1500 ml",
           "EXP 1", "AA 1e3 ml", "A A A", "1 2 3 A", "NA", NA, "1e2",
           "", "-", "1A", "1 A", "100", "100A", "Inf"))

OrderMixed(x)

SortMixed(x)  # Notice that plain numbers, including 'Inf' show up
              # before strings, NAs at the end, and blanks at the
              # beginning .


SortMixed(x, na.last=TRUE)  # default
SortMixed(x, na.last=FALSE) # push NAs to the front


SortMixed(x, blank.last=FALSE) # default
SortMixed(x, blank.last=TRUE)  # push blanks to the end

SortMixed(x, decreasing=FALSE) # default
SortMixed(x, decreasing=TRUE)  # reverse sort order

## Roman numerals
chapters <- c("V. Non Sequiturs", "II. More Nonsense",
              "I. Nonsense", "IV. Nonesensical Citations",
              "III. Utter Nonsense")
SortMixed(chapters, numeric.type="roman" )

## Lower-case Roman numerals
vals <- c("xix", "xii", "mcv", "iii", "iv", "dcclxxii",   "cdxcii",
          "dcxcviii",   "dcvi",   "cci")
(ordered <- SortMixed(vals, numeric.type="roman", roman.case="lower"))
RomanToInt(ordered)
```

SpearmanRho *Spearman Rank Correlation*

Description

Calculate Spearman correlation coefficient and its confidence interval. In addition to the base R function `cor()`, frequency tables are also accepted as arguments (i.e. actually weights are used).

Usage

```
SpearmanRho(x, y = NULL, use = c("everything", "all.obs", "complete.obs",
            "na.or.complete","pairwise.complete.obs"),
            conf.level = NA)
```

Arguments

`x`	a numeric vector, an ordered factor, matrix or data frame. An ordered factor will be coerced to numeric.
`y`	`NULL` (default) or a vector, an ordered factor, matrix or data frame with compatible dimensions to x. An ordered factor will be coerced to numeric.
`use`	an optional character string giving a method for computing covariances in the presence of missing values. This must be (an abbreviation of) one of the strings `"everything"`, `"all.obs"`, `"complete.obs"`, `"na.or.complete"`, or `"pairwise.complete.obs"`.
`conf.level`	confidence level of the interval. If set to `NA` (which is the default) no confidence interval will be calculated.

Details

The function calculates Spearman's rho statistic by means of `cor(..., method="spearman")` when two variables `x` and `y` are supplied. If a frequency table is provided an implementation based on SAS documentation is used.
The confidence intervals are calculated via z-Transformation.

Value

Either a single numeric value, if no confidence interval is required,
or a vector with 3 elements for estimate, lower and upper confidence intervall.

Author(s)

Andri Signorell <andri@signorell.net>

References

Conover W. J. (1999) *Practical Nonparametric Statistics (3rd edition)*. Wiley

See Also

`cor`

Examples

```
pain <- as.table(matrix(c(26,  6, 26, 7, 23,
                          9, 18, 14, 9, 23),
                        ncol=5, byrow=TRUE,
        dimnames=list(adverse=c("no", "yes"), dose=1:5)))

SpearmanRho(pain)

SpearmanRho(pain, conf.level=0.95)

# must be the same as
with(Untable(pain),
     SpearmanRho(adverse, dose, conf.level=0.95))
```

---

split.formula *Formula Interface for Split*

Description

Implementation of a simple formula interface for the split function.

Usage

```
## S3 method for class 'formula'
 split(x, f, drop = FALSE, data = NULL, ...)
```

Arguments

x	a formula of the form y ~ x.
f	a 'factor' in the sense that as.factor(f) defines the grouping, or a list of such factors in which case their interaction is used for the grouping.
drop	logical indicating if levels that do not occur should be dropped (if f is a factor or a list). Defaults to FALSE.
data	the data frame from which the formula should be evaluated.
...	other arguments to be passed to split.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

split

Examples

```
split(extra ~ group, data = sleep)
```

SplitAt *Split a Vector Into Several Pieces at Given Positions*

Description

Split a vector into several pieces at given positions.

Usage

```
SplitAt(x, pos)
```

Arguments

x	the vector to be splitted.
pos	integer vector, giving the positions at which the vector should be splitted.

Value

a list with the splitted parts of x.

Author(s)

flodel (on StackOverflow)

References

https://stackoverflow.com/questions/16357962/r-split-numeric-vector-at-position

See Also

split, strsplit

Examples

```
x <- 1:10
SplitAt(x, pos=c(3, 8))
```

SplitPath *Split Path In Drive, Path, Filename*

Description

Split a full path in its components. This is specifically an issue in Windows and not really interesting for other OSs.

Usage

```
SplitPath(path, last.is.file = NULL)
```

Arguments

path	a path
last.is.file	logical, determining if the basename should be interpreted as filename or as last directory. If set to NULL (default), the last entry will be interpreted if the last character is either \ or / and as filename else.

Value

A list, containing the following components:

normpath	the normalized path as returned by normalizePath
drive	the drive if the OS is Windows, NA else
dirname	the path without directory and without filename
fullfilename	the filename including extension
filename	the filename without extension
extension	the file extension

Author(s)

Andri Signorell <andri@signorell.net>

See Also

dirname, basename

Examples

```
## Not run:  # Windows-specific example
path <- "C:/Documents/Projects/Import/eyestudy.dta"
SplitPath(path)

path <- "C:/Documents/Projects/Import/"
SplitPath(path)
```

```
path <- "C:/Documents/Projects/Import"
SplitPath(path)  # last entry will be taken as filename
SplitPath(path, last.is.file=FALSE)

## End(Not run)
```

SplitToCol *Split Data Frame String Column Into Multiple Columns*

Description

Splitting the string columns of a data frame into multiple columns requires a considerable number of codelines, which are condensed in this function for convenience.

Usage

```
SplitToCol(x, split = " ", fixed = TRUE, na.form = "", colnames = NULL)
```

Arguments

x	a data frame containing the string columns to be splitted.
split	character vector (or object which can be coerced to such) containing regular expression(s) (unless fixed = TRUE) to use for splitting. If empty matches occur, in particular if split has length 0, x is split into single characters. If split has length greater than 1, it is re-cycled along x.
fixed	logical. If TRUE match split exactly, otherwise use regular expressions. Has priority over perl.
na.form	character, string specifying how NAs should be specially formatted. Default is a blank "".
colnames	columnnames for the resulting data.frame. Will be recycled. Can easily be set to "" if no columnnames should be set.

Value

A data.frame with all the columns splitted

A vector with the length of the number of columns of the data.frame containing the number of the found columns is returned as attribute namede "ncols".

Author(s)

Andri Signorell <andri@signorell.net>

See Also

strsplit

Examples

```
d.frm <- data.frame(res1=c("2 [-3,5] **", "5 [-2,6] ***", "9 [-3,1]"),
                    res2=c("5 [6,8] **", "7 [-2,9]", "4 [3,5] **"),
                    stringsAsFactors=FALSE)

SplitToCol(d.frm, na.form="-", colnames=c("coef", "ci", "pval"))
```

SplitToDummy *Split Strings of a Vector and Provide Dummy Codes for Found Pieces*

Description

Split the strings of a character vector, put together all the unique pieces and return a matrix of dummy vectors for each single value.

Usage

```
SplitToDummy(x, split = ",", ...)
```

Arguments

x	character vector, each element of which is to be split. Other inputs, including a factor, will give an error.
split	character vector (or object which can be coerced to such) containing regular expression(s) (unless `fixed = TRUE`) to use for splitting. If empty matches occur, in particular if `split` has length 0, x is split into single characters. If `split` has length greater than 1, it is re-cycled along x.
...	the dots are passed on to `strsplit`

Value

a data.frame containing x and all the found dummy vectors

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`strsplit`

Examples

```
d.frm <- data.frame(id=1:5, txt=c("A,C,D", "A","B,C","D","D,E"))
SplitToDummy(d.frm$txt)
```

SpreadOut *Spread Out a Vector of Numbers To a Minimum Interval*

Description

Spread the numbers of a vector so that there is a minimum interval between any two numbers (in ascending or descending order). This is helpful when we want to place textboxes on a plot and ensure, that they do not mutually overlap.

Usage

```
SpreadOut(x, mindist = NULL, cex = 1.0)
```

Arguments

x	a numeric vector which may contain NAs.
mindist	the minimum interval between any two values. If this is left to NULL (default) the function will check if a plot is open and then use 90% of strheight().
cex	numeric character expansion factor; multiplied by par("cex") yields the final character size; the default NULL is equivalent to 1.

Details

SpreadOut() starts at or near the middle of the vector and increases the intervals between the ordered values. NAs are preserved. SpreadOut() first tries to spread groups of values with intervals less than mindist out neatly away from the mean of the group. If this doesn't entirely succeed, a second pass that forces values away from the middle is performed.

SpreadOut() can also be used to avoid overplotting of axis tick labels where they may be close together.

Value

On success, the spread out values. If there are less than two valid values, the original vector is returned.

Note

This function is based on plotrix::spreadout() and has been integrated here with some minor changes.

Author(s)

Jim Lemon <jim@bitwrit.com.au>
some extensions Andri Signorell <andri@signorell.net>

See Also

strheight()

Examples

```
SpreadOut(c(1, 3, 3, 3, 3, 5), 0.2)
SpreadOut(c(1, 2.5, 2.5, 3.5, 3.5, 5), 0.2)
SpreadOut(c(5, 2.5, 2.5, NA, 3.5, 1, 3.5, NA), 0.2)

# this will almost always invoke the brute force second pass
SpreadOut(rnorm(10), 0.5)
```

Stamp *Date/Time/Directory Stamp the Current Plot*

Description

Stamp the current plot in the extreme lower right corner. A free text or expression can be defined as text to the stamp.

Usage

```
Stamp(txt = NULL, las = par("las"), cex = 0.6)
```

Arguments

txt	an optional single text string. If it is not given, the function will look for a defined option named stamp. If not found the current date will be taken as text. If the stamp option is defined as expression the function will evaluate it. This can be used to define dynamic texts.
las	numeric in c(1, 3), defining direction of the text. 1 means horizontal, 3 vertical. Default is taken from par("las").
cex	numeric **c**haracter **ex**pansion factor; multiplied by par("cex") yields the final character size. Defaults to 0.6.

Details

The text can be freely defined as option. If user and date should be included by default, the following option using an expression will help:

```
DescToolsOptions(stamp=expression(gettextf('%s/%s',
          Sys.getenv('USERNAME'), Format(Today(), fmt='yyyy-mm-dd') )))
```

For R results may not be satisfactory if par(mfrow=) is in effect.

Author(s)

Frank E Harrell Jr <f.harrell@vanderbilt.edu>
with some amendments by Andri Signorell <andri@signorell.net>

See Also

text

Examples

```
plot(1:20)
Stamp()
```

StdCoef *Standardized Model Coefficients*

Description

Standardize model coefficients by Standard Deviation or Partial Standard Deviation.

Usage

```
StdCoef(x, partial.sd = FALSE, ...)

PartialSD(x)
```

Arguments

x	a fitted model object.
partial.sd	logical, if set to TRUE, model coefficients are multiplied by partial SD, otherwise they are multiplied by the ratio of the standard deviations of the independent variable and dependent variable.
...	additional arguments passed to coefTable, e.g. dispersion.

Details

The standardized coefficients are meant to allow for a comparison of the importance of explanatory variables that have different variances. Each of them shows the effect on the response of increasing its predictor X(j) by one standard deviation, as a multiple of the response's standard deviation. This is often a more meaningful comparison of the relevance of the input variables.

Note, however, that increasing one X(j) without also changing others may not be possible in a given application, and therefore, interpretation of coefficients can always be tricky. Furthermore, for binary input variables, increasing the variable by one standard deviation is impossible, since an increase can only occur from 0 to 1, and therefore, the standardized coeffient is somewhat counter-intuitive in this case.

Standardizing model coefficients has the same effect as centring and scaling the input variables.

"Classical" standardized coefficients are calculated as $\beta_i^* = \beta_i \frac{s_{X_i}}{s_y}$, where β is the unstandardized coefficient, s_{X_i} is the standard deviation of associated depenent variable X_i and s_y is SD of the response variable.

If the variables are intercorrelated, the standard deviation of X_i used in computing the standardized coefficients β_i^* should be replaced by a partial standard deviation of X_i which is adjusted for the multiple correlation of X_i with the other X variables included in the regression equation. The partial standard deviation is calculated as $s^*_{X_i} = s_{X_i} VIF(X_i)^{-0.5}(\frac{n-1}{n-p})^{0.5}$, where VIF is the variance inflation factor, n is the number of observations and p number of predictors in the model. Coefficient is then transformed as $\beta_i^* = \beta_i s^*_{X_i}$.

Value

A matrix with at least two columns for standardized coefficient estimate and its standard error. Optionally, third column holds degrees of freedom associated with the coefficients.

Author(s)

Kamil Bartoń

References

Cade, B.S. (2015) Model averaging and muddled multimodel inferences. *Ecology* 96, 2370-2382.

Afifi A., May S., Clark V.A. (2011) *Practical Multivariate Analysis*, Fifth Edition. CRC Press.

Bring, J. (1994). How to standardize regression coefficients. *The American Statistician* 48, 209-213.

See Also

coef

Examples

```
# Fit model to original data:
fm  <- lm(Fertility ~ Agriculture + Examination + Education + Catholic,
          data = swiss)

# Partial SD for the default formula:
psd <- PartialSD(lm(data = swiss))[-1] # remove first element for intercept

# Standardize data:
zswiss <- scale(swiss, scale = c(NA, psd), center = TRUE)
# Note: first element of 'scale' is set to NA to ignore the first column 'y'

# Coefficients of a model fitted to standardized data:
# zapsmall(coefTable(stdizeFit(fm, data = zGPA)))
# Standardized coefficients of a model fitted to original data:
# zapsmall(StdCoef(fm, partial = TRUE))


# Standardizing nonlinear models:
fam <- Gamma("inverse")
fmg <- glm(log(Fertility) ~ Agriculture + Examination + Education + Catholic,
           data = swiss, family = fam)
```

```
psdg <- PartialSD(fmg)
# zGPA <- stdize(GPA, scale = c(NA, psdg[-1]), center = FALSE)
# fmgz <- glm(log(y) ~ z.x1 + z.x2 + z.x3 + z.x4, zGPA, family = fam)

# Coefficients using standardized data:
# coef(fmgz) # (intercept is unchanged because the variables haven't been
           #  centred)
# Standardized coefficients:
# coef(fmg) * psdg
```

Str *Compactly Display the Structure of any R Object*

Description

Basically a wrapper for `str()`, extended with an enumeration for the variables of a data.frame.

Usage

```
Str(x, ...)
```

Arguments

x	any R object about which you want to have some information.
...	dots are passed to `str`.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`str`

Examples

```
Str(d.pizza)
```

StrAbbr *String Abbreviation*

Description

Abbreviate a character vector. The function includes starting from the first character as many characters as there are needed to result in a vector of unique values.

Usage

```
StrAbbr(x, minchar = 1, method = c("left", "fix"))
```

Arguments

x	character vector to be abbreviated
minchar	integer, minimal number of characters for the abbreviations.
method	one out of `left` or `fix`. While `left` restricts the result to as many characters are needed to ensure uniqueness, does `fix` yield a vector with all the elements being as long, as the the longest needed substring for differentiating the terms.

Value

The abbreviated strings.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

abbreviate, StrTrunc, StrTrim

Examples

```
StrAbbr(x=levels(d.pizza$driver), minchar=2)
StrAbbr(x=levels(d.pizza$driver), minchar=2, method="left")
StrAbbr(x=levels(d.pizza$driver), minchar=2, method="fix")


x <- c("Aaron", "Aaramis", "Berta", "Bello", "Claudia", "Cardinale", "Doretta", "Emilia")
StrAbbr(x, minchar=2, method="left")
StrAbbr(x, minchar=2, method="fix")
```

`StrAlign` *String Alignment*

Description

Align a vector of strings to the left, to the right, to the center or to the first occurance of a specified character, e.g. to the decimal separator. Alignment is achieved by padding the strings with empty spaces (which evidently only will have an alignment effect if the text is displayed with a monospaced font).

Usage

```
StrAlign(x, sep = "\\r")
```

Arguments

`x`	a character vector to be aligned.
`sep`	the character on whose position the strings will be aligned. Left alignment can be requested by setting `sep = "\\l"`, right alignment by `"\\r"` and center alignment by `"\\c"`. Mind the backslashes, as if they are omitted, strings would be aligned to the **character** l, r or c respectively. Default value is `"\\r"`, thus right alignment.

Details

Alignment to the left or right leave no room for misinterpretation. The function will determine the maximum string size in the vector, resize all the strings to this size by padding empty spaces either at the beginning or at the end.

```
cbind(StrAlign(c("here", "there", "everywhere"), sep = "\r"))
[1,] "      here"
[2,] "     there"
[3,] "everywhere"
```

When it comes to center strings, it's not clear where to place strings with an even length in case the maximum length is odd (or vice versa). We will put the shorter distance of an uneven string to the left (note the second term, that has 2 spaces on the left and 3 spaces on the right).

```
cbind(StrAlign(c("here", "there", "everywhere"), sep = "\c"))
[1,] "   here   "
[2,] "  there   "
[3,] "everywhere"
```

Any specific length of the strings can be created by `StrPad` if required.

In case of a given character as separator the strings will be aligned towards this separator. Frequently this might be the decimal separator. If a string does not contain the separator, the affected string will be aligned as if it had a separator as last character. This seems to be a good default, when integer

numbers are to be aligned with numerical values. Note that the character length of the resulting strings can excceed the maximum length of the supplied strings.

```
z <- c(" 6.0", "6.00 ", " 45.12 ", "784", NA)
cbind(StrAlign(z, sep="."))
     [,1]
[1,] "  6.0 "
[2,] "  6.00"
[3,] " 45.12"
[4,] "784   "
[5,] NA
```

The character strings will not be pruned of whitespaces, if the requested alignment does not explicitly require it. StrTrim can be used for that.

Value

a character vector containing the aligned strings

Author(s)

Andri Signorell <andri@signorell.net>

See Also

StrTrim, StrPad, Format

Examples

```
# align on (the first occuring) B
x <- c("ABCDMNB", "CDGHEBK", "BCI")
cbind(StrAlign(x, sep="B"))

# align to decimal separator (here point)
z <- c("    6.0", "6.00  ", " 45.12    ", "784", NA)
cbind(StrAlign(z, sep="."))

# right align, the width will be the max number of characters in x
cbind(StrAlign(x, sep="\\r"))
# left align
cbind(StrAlign(x, sep="\\l"))
# center
cbind(StrAlign(x, sep="\\c"))
```

`Strata` *Stratified Sampling*

Description

Stratified sampling with equal/unequal probabilities.

Usage

```
Strata(x, stratanames = NULL, size,
       method = c("srswor", "srswr", "poisson", "systematic"),
       pik, description = FALSE)
```

Arguments

`x`	a data frame or a matrix; its number of rows is n, the population size.
`stratanames`	vector of stratification variables.
`size`	vector of stratum sample sizes (in the order in which the strata are given in the input data set).
`method`	method to select units; implemented are: a) simple random sampling without replacement (`"srswor"`), b) simple random sampling with replacement (`"srswr"`), c) Poisson sampling (`"poisson"`), d) systematic sampling (`"systematic"`) (default is `"srswor"`).
`pik`	vector of inclusion probabilities or auxiliary information used to compute them; this argument is only used for unequal probability sampling (Poisson and systematic). If an auxiliary information is provided, the function uses the inclusion-probabilities function for computing these probabilities. If the method is "srswr" and the sample size is larger than the population size, this vector is normalized to one.
`description`	a message is printed if its value is TRUE; the message gives the number of selected units and the number of the units in the population. By default, the value is FALSE.

Value

The function produces an object, which contains the following information:

`id`	the identifier of the selected units.
`stratum`	the unit stratum.
`prob`	the final unit inclusion probability.

Author(s)

Andri Signorell <andri@signorell.net>
rewritten based on the ideas of Yves Tille <yves.tille@unine.ch> and Alina Matei <alina.matei@unine.ch>

See Also

sample

Examples

```
# Example from An and Watts (New SAS procedures for Analysis of Sample Survey Data)
# generates artificial data (a 235X3 matrix with 3 columns: state, region, income).
# the variable "state" has 2 categories ('nc' and 'sc').
# the variable "region" has 3 categories (1, 2 and 3).
# the sampling frame is stratified by region within state.
# the income variable is randomly generated

m <- rbind(matrix(rep("nc",165), 165, 1, byrow=TRUE),
           matrix(rep("sc", 70), 70, 1, byrow=TRUE))
m <- cbind.data.frame(m, c(rep(1, 100), rep(2,50), rep(3,15),
                      rep(1, 30), rep(2, 40)), 1000 * runif(235))
names(m) <- c("state", "region", "income")

# computes the population stratum sizes
table(m$region, m$state)

# not run
#     nc  sc
#  1 100  30
#  2  50  40
#  3  15   0
# there are 5 cells with non-zero values
# one draws 5 samples (1 sample in each stratum)
# the sample stratum sizes are 10,5,10,4,6, respectively
# the method is 'srswor' (equal probability, without replacement)

s <- Strata(m, c("region", "state"), size=c(10, 5, 10, 4, 6), method="srswor")

# extracts the observed data
data.frame(income=m[s$id, "income"], s)

# see the result using a contigency table
table(s$region, s$state)


# The same data as in Example 1
# the method is 'systematic' (unequal probability, without replacement)
# the selection probabilities are computed using the variable 'income'
s <- Strata(m,c("region", "state"), size=c(10, 5, 10, 4, 6),
            method="systematic", pik=m$income)

# extracts the observed data
data.frame(income=m[s$id, "income"], s)

# see the result using a contigency table
table(s$region, s$state)
```

StrCap *Capitalize the First Letter of a String*

Description

Capitalize the first letter of each element of the string vector.

Usage

```
StrCap(x, method=c("first", "word", "title"))
```

Arguments

x	string to be capitalized.
method	one out of "first" (default), "word", "title". "first" will only capitalize the first character of a string. "word" will capitalize all found words and "title" will also capitalize wordwise, but leave out: a, an, the, at, by, for, in, of, on, to, up, and, as, but, s, or and nor.)

Value

Returns a vector of charaters with the first letter capitalized

Author(s)

Charles Dupont <charles.dupont@vanderbilt.edu>, Andri Signorell <andri@signorell.net> (methods word and title)

Examples

```
# capitalize first character
StrCap(c("Hello", "bob", "daN"))
# but not all...
StrCap(c("Hello bob, how are you?", "And you, DANIEL?"))

# wordwise
StrCap(c("Capitalize all words in titles of publications and documents",
             "but Up and UP, not all and all", NA), method="word")

# wordwise omitting the ones listed above
StrCap(c("Capitalize all words in titles of publications and documents",
         "but Up and UP, not all and all", NA), method="title")

# do not touch non alphabetic characters
z <- c("Lorem ipsum dolor", "-- sit amet", "consectetur --", " adipiscing elit ",
       "sed,.--(do) / +-*eiusmod")
StrCap(z, method="title")
```

StrChop *Split a String into a Number of Sections of Defined Length*

Description

Splitting a string into a number of sections of defined length is needed, when we want to split a table given as a number of lines without separator into columns. The cutting points can either be defined by the lengths of the sections or directly by position.

Usage

```
StrChop(x, len, pos)
```

Arguments

x	the string to be cut in pieces.
len	a vector with the lengths of the pieces.
pos	a vector of cutting positions. Will be ignored when len has been defined.

Details

If length is going over the end of the string the last part will be returned, so if the rest of the string is needed, it's possible to simply enter a big number as last partlength.

len and pos can't be defined simultaneously, only alternatively.

Typical usages are

```
StrChop(x, len)
StrChop(x, pos)
```

Value

a vector with the parts of the string.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

FixToTable, StrLeft, substr

Examples

```
x <- paste(letters, collapse="")
StrChop(x=x, len = c(3,5,2))

# and with the rest integrated
StrChop(x=x, len = c(3, 5, 2, nchar(x)))

# cutpoints at 5th and 10th position
StrChop(x=x, pos=c(5, 10))
```

StrCountW *Count Words in a String*

Description

Count the number of words that appear within a character string.

Usage

```
StrCountW(x)
```

Arguments

x	a vector of strings to be parsed.

Details

This is just a wrapper for a fine regexpr. It uses the expression `\b\W+\b` to separate the words. The code `\W` is equivalent to `[^[:alnum:]_])` wheras `[:alnum:]` contains `[:alpha:]` and `[:digit:]`. So everything that is not an alphanumeric character, a digit or a _ (underscore) is used as separator for the words to be counted.

Value

an integer defining the number of word in the string

Author(s)

Andri Signorell <andri@signorell.net>, based on code from Adam Bradley <hisself@adambradley.net>

References

http://stackoverflow.com/questions/8920145/count-the-number-of-words-in-a-string-in-r

See Also

nchar

Examples

```
StrCountW("This is a true story!")

StrCountW("Just_one_word")
StrCountW("Not-just.one/word")

StrCountW("And what about numbers 8899 or special characters $$$/*?")
StrCountW("   Starting'n ending with some whitespace  ")

StrCountW(c("This is a", "text in more", "than one line."))
```

StrDist *Compute Distances Between Strings*

Description

`StrDist` computes distances between strings following to Levenshtein or Hamming method.

Usage

```
StrDist(x, y, method = "levenshtein", mismatch = 1, gap = 1, ignore.case = FALSE)
```

Arguments

`x`	character vector, first string.
`y`	character vector, second string.
`method`	character, name of the distance method. This must be `"levenshtein"`, `"normlevenshtein"` or `"hamming"`. Default is `"levenshtein"`, the classical Levenshtein distance.
`mismatch`	numeric, distance value for a mismatch between symbols.
`gap`	numeric, distance value for inserting a gap.
`ignore.case`	if `FALSE` (default), the distance measure will be case sensitive and if `TRUE`, case is ignored.

Details

The function computes the Hamming and the Levenshtein (edit) distance of two given strings (sequences). The Hamming distance between two vectors is the number mismatches between corresponding entries.

In case of the Hamming distance the two strings must have the same length.

In case of the Levenshtein (edit) distance a scoring and a trace-back matrix are computed and are saved as attributes `"ScoringMatrix"` and `"TraceBackMatrix"`. The numbers in the trace-back matrix reflect insertion of a gap in string `y` (1), match/missmatch (2), and insertion of a gap in string `x` (3).

The edit distance is useful, but normalizing the distance to fall within the interval [0,1] is preferred because it is somewhat diffcult to judge whether an LD of for example 4 suggests a high or low

degree of similarity. The method "normlevenshtein" for normalizing the LD is sensitive to this scenario. In this implementation, the Levenshtein distance is transformed to fall in this interval as follows:

$$lnd = 1 - \frac{ld}{max(length(x), length(y))}$$

where ld is the edit distance and max(length(x), length(y)) denotes that we divide by the length of the larger of the two character strings. This normalization, referred to as the Levenshtein normalized distance (lnd), yields a statistic where 1 indicates perfect agreement between the two strings, and a 0 denotes imperfect agreement. The closer a value is to 1, the more certain we can be that the character strings are the same; the closer to 0, the less certain.

Value

StrDist returns an object of class "dist"; cf. dist.

Note

For distances between strings and for string alignments see also Bioconductor package **Biostrings**

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References

R. Merkl and S. Waack (2009) *Bioinformatik Interaktiv*. Wiley.

Harold C. Doran (2010) *MiscPsycho. An R Package for Miscellaneous Psychometric Analyses*

See Also

adist, dist

Examples

```
x <- "GACGGATTATG"
y <- "GATCGGAATAG"
## Levenshtein distance
d <- StrDist(x, y)
d
attr(d, "ScoringMatrix")
attr(d, "TraceBackMatrix")

## Hamming distance
StrDist(x, y, method="hamming")
```

`StrExtract` *Extract Part of a String*

Description

Extract a part of a string, defined as regular expression. `StrExtractBetween()` is a convenience function used to extract parts between a left and right delimiter.

Usage

```
StrExtract(x, pattern, ...)

StrExtractBetween(x, left, right, greedy = FALSE)
```

Arguments

`x`	a character vector where matches are sought, or an object which can be coerced by `as.character` to a character vector.
`pattern`	character string containing a regular expression (or character string for `fixed = TRUE`) to be matched in the given character vector. Coerced by `as.character` to a character string if possible. If a character vector of length 2 or more is supplied, the first element is used with a warning. Missing values are not allowed.
`left`	left character(s) limiting the string to be extracted
`right`	right character(s) limiting the string to be extracted
`greedy`	logical, determines whether the first found match for `right` should be used (`FALSE`, default) or the last (`TRUE`).
`...`	the dots are passed to the the internally used function `regexpr()`, which allows to use e.g. Perl-like regular expressions.

Details

The function wraps `regexpr` and `regmatches`.

Value

A character vector.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`regexpr`, `regmatches`

Examples

```
txt <- c("G1:E001", "No points here", "G2:E002", "G3:E003", NA)

# extract everything after the :
StrExtract(x=txt, pattern=":.*")

# extract everything between "left" and "right"
z <- c("yBS (23A) 890", "l 89Z) 890.?/", "WS (55X) 8(90)", "123 abc", "none", NA)
# everything enclosed by spaces
StrExtractBetween(z, " ", " ")

# note to escape special characters
StrExtractBetween(z, "\\(", "\\)")
```

StripAttr *Remove Attributes from an Object*

Description

For convenience we sometimes want to strip some or all attributes in a oneliner.

Usage

```
SetAttr(x, attr, attr_val)
StripAttr(x, attr_names = NULL)
```

Arguments

x	the object whose attributes should be removed or to which an attribute should be added.
attr	name of a new attribute
attr_val	value for the new attribute attr
attr_names	a vector with attribute names, which will be removed. Leaving the default to NULL will cause all the attributes to be deleted.

Value

the object x without the attributes contained in attr_names

Author(s)

Andri Signorell <andri@signorell.net>

See Also

SetNames, unname

Examples

```
x <- runif(10)
x <- SetAttr(x,
             attr=c("some_attr", "other_attr"),
             attr_val=c("First attribute", "Second attribute"))

# strip only single
StripAttr(x, "other_attr")

# strip all attributes
StripAttr(x)
```

StrIsNumeric *Does a String Contain Only Numeric Data*

Description

Check whether a string does only contain numeric data.

Usage

```
StrIsNumeric(x)
```

Arguments

x	a character vector

Value

a logical vector with the same dimension as x

Author(s)

Andri Signorell <andri@signorell.net>

See Also

Other string functions, e.g. StrTrunc

Examples

```
x <- c("123", "-3.141", "foobar123")
StrIsNumeric(x)
```

`StrLeft, StrRight` *Returns the Left Or the Right Part Of a String*

Description

Returns the left part or the right part of a string. The number of characters are defined by the argument `n`. If `n` is negative, this number of characters will be cut off from the other side.

Usage

```
StrLeft(x, n)
StrRight(x, n)
```

Arguments

`x`	a vector of strings.
`n`	a positive or a negative integer, the number of characters to cut. If n is negative, this number of characters will be cut off from the right with `StrLeft` and from the right with `StrRight`. n will be recycled.

Details

The functions `StrLeft` and `StrRight` are simple wrappers to `substr`.

Value

the left (right) n characters of x

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`substr`, `StrTrim`

Examples

```
StrLeft("Hello world!", n=5)
StrLeft("Hello world!", n=-5)

StrRight("Hello world!", n=6)
StrRight("Hello world!", n=-6)

StrLeft(c("Lorem", "ipsum", "dolor","sit","amet"), n=2)

StrRight(c("Lorem", "ipsum", "dolor","sit","amet"), n=c(2,3))
```

StrPad *Pad a String With Justification*

Description

StrPad will fill a string x with defined characters to fit a given length.

Usage

```
StrPad(x, width = NULL, pad = " ", adj = "left")
```

Arguments

x	a vector of strings to be padded.
width	resulting width of padded string. If x is a vector and width is left to NULL, it will be set to the length of the largest string in x.
pad	string to pad with. Will be repeated as often as necessary. Default is " ".
adj	adjustement of the old string, one of "left", "right", "center". If set to "left" the old string will be adjusted on the left and the new characters will be filled in on the right side.

Details

If a string x has more characters than width, it will be chopped on the length of width.

Value

the string

Author(s)

Christian W. Hoffmann <c-w.hoffmann@sunrise.ch>
some extensions Andri Signorell <andri@signorell.net>

Examples

```
StrPad("My string", 25, "XoX", "center")
 # [1] "XoXXoXXoMy stringXXoXXoXX"
```

StrPos *Find Position of First Occurrence Of a String*

Description

Returns the numeric position of the first occurrence of a substring within a string. If the search string is not found, the result will be NA.

Usage

```
StrPos(x, pattern, pos = 1, ...)
```

Arguments

x	a character vector in which to search for the pattern, or an object which can be coerced by as.character to a character vector.
pattern	character string (search string) containing the pattern to be matched in the given character vector. This can be a character string or a regular expression.
pos	integer, defining the start position for the search within x. The result will then be relative to the begin of the truncated string. Will be recycled.
...	the dots are passed to the function regexpr.

Details

This is just a wrapper for the function regexpr.

Value

a vector of the first position of pattern in x

Author(s)

Andri Signorell <andri@signorell.net>

See Also

StrChop, regexpr

Examples

```
StrPos(x = levels(d.pizza$driver), pattern = "t")
```

StrRev *Reverse a String*

Description

Returns a string in reverse order.

Usage

```
StrRev(x)
```

Arguments

x	a string to be processed.

Value

string

Author(s)

Andri Signorell <andri@signorell.net> solely copying R core code from strsplit example

See Also

String functions: nchar, match, grep, regexpr, substr, sub, gsub, StrTrunc, StrDist

Examples

```
StrRev("home")
StrRev("Anna")
```

StrSpell *Spell a String Using the NATO Phonetic or the Morse Alphabet*

Description

The function splits a string into single characters and returns their representation in either the NATO phonetic alphabet or the Morse alphabet. The 26 code words in the NATO phonetic alphabet are assigned to the 26 letters of the English alphabet in alphabetical order as follows: Alfa, Bravo, Charlie, Delta, Echo, Foxtrot, Golf, Hotel, India, Juliett, Kilo, Lima, Mike, November, Oscar, Papa, Quebec, Romeo, Sierra, Tango, Uniform, Victor, Whiskey, X-ray, Yankee, Zulu. Digits 0-9 are also supported.

Usage

```
StrSpell(x, upr = "CAP", type = c("NATO", "Morse"))
```

Arguments

x	character, the string to be encoded.
upr	character, a shortcut to be used to characterise capital letters. Ignored if type is set to "Morse". No distinction is made between upper and lower case if upr is set to NA or to an empty string "".
type	the type of phonetic alphabet, either "NATO" or "Morse".

Value

a character vector containing the code words

Author(s)

Andri Signorell <andri@signorell.net>

References

https://en.wikipedia.org/wiki/NATO_phonetic_alphabet

See Also

strsplit

Examples

```
# ... ever had to communicate a password by phone? ;-)
StrSpell("Yailov9teb6i")

paste(StrSpell("Andri", type="Morse"), collapse="|")
```

StrSplit *Split the Elements of a Character Vector*

Description

Split the elements of a character vector x into substrings according to the matches to substring split within them.
This is a verbatim copy of the base R function strsplit, but with a split default of "" and returning a vector instead of a list, when x had the length 1.

Usage

```
StrSplit(x, split = "", fixed = FALSE, perl = FALSE, useBytes = FALSE)
```

Arguments

x	character vector, each element of which is to be split. Other inputs, including a factor, will give an error.
split	character vector (or object which can be coerced to such) containing regular expression(s) (unless fixed = TRUE) to use for splitting. If empty matches occur, in particular if split has length 0, x is split into single characters. If split has length greater than 1, it is re-cycled along x.
fixed	logical. If TRUE match split exactly, otherwise use regular expressions. Has priority over perl.
perl	logical. Should Perl-compatible regexps be used?
useBytes	logical. If TRUE the matching is done byte-by-byte rather than character-by-character, and inputs with marked encodings are not converted. This is forced (with a warning) if any input is found which is marked as "bytes" (see Encoding).

Details

See strsplit for the details.

Value

A list of the same length as x, the i-th element of which contains the vector of splits of x[i].

If the length x was 1 a vecotor with the splits will be returned.

See Also

paste for the reverse, grep and sub for string search and manipulation; also nchar, substr.

'regular expression' for the details of the pattern specification.

Examples

```
noquote(StrSplit("A text I want to display with spaces"))

# the same as ...
noquote(strsplit("A text I want to display with spaces", NULL)[[1]])
```

StrTrim *Remove Leading/Trailing Whitespace From A String*

Description

The function removes whitespace characters as spaces, tabs and newlines from the beginning and end of the supplied string. Whitespace characters occurring in the middle of the string are retained. Trimming with method "left" deletes only leading whitespaces, "right" only trailing. Designed for users who were socialized by SQL.

Usage

```
StrTrim(x, pattern = " \t\n", method = "both")
```

Arguments

x	the string to be trimmed.
pattern	the pattern of the whitespaces to be deleted, defaults to space, tab and newline: " \t\n".
method	one out of "both" (default), "left", "right". Determines on which side the string should be trimmed.

Details

The functions are defined depending on method as
both: gsub(pattern=gettextf("^[%s]+|[%s]+$", pattern, pattern), replacement="", x=x)
left: gsub(pattern=gettextf("^[%s]+",pattern), replacement="", x=x)
right: gsub(pattern=gettextf("[%s]+$",pattern), replacement="", x=x)

Value

the string x without whitespaces

Author(s)

Andri Signorell <andri@signorell.net>

See Also

String functions: trimws, nchar, match, grep, regexpr, substr, sub, gsub, StrTrunc, StrDist

Examples

```
StrTrim("  Hello world! ")

StrTrim("  Hello world! ", method="left")
StrTrim("  Hello world! ", method="right")

# user defined pattern
StrTrim(" ..Hello ... world! ", pattern=" \\.")
```

StrTrunc *Truncate Strings and Add Ellipses If a String is Truncated.*

Description

Truncates one or more strings to a specified length, adding an ellipsis (...) to those strings that have been truncated. The truncation can also be performed using word boundaries. Use StrAlign() to justify the strings if needed.

Usage

```
StrTrunc(x, maxlen = 20, ellipsis = "...", wbound = FALSE)
```

Arguments

x	a vector of strings.
maxlen	the maximum length of the returned strings (NOT counting the appended ellipsis). maxlen is recycled.
ellipsis	the string to be appended, if the string is longer than the given maximal length. The default is "...".
wbound	logical. Determines if the maximal length should be reduced to the next smaller word boundary and so words are not chopped. Default is FALSE.

Value

The string(s) passed as 'x' now with a maximum length of 'maxlen' + 3 (for the ellipsis).

Author(s)

Andri Signorell,
once following an idea of Jim Lemon in truncString()

See Also

String functions: nchar, match, grep, regexpr, substr, sub, gsub, StrTrim, StrDist

Examples

```
x <- c("this is short", "and this is a longer text",
       "whereas this is a much longer story, which could not be told shorter")

# simple truncation on 10 characters
StrTrunc(x, maxlen=10)

# NAs remain NA
StrTrunc(c(x, NA_character_), maxlen=15, wbound=TRUE)

# using word boundaries
```

```
for(i in -5:20)
  print(StrTrunc(x, maxlen=i, wbound=TRUE))

# compare
for(i in -5:20)
  print(StrTrunc(x, maxlen=i, wbound=FALSE))
```

StrVal *Extract All Numeric Values From a String*

Description

Extract all numeric values from a string using a regular expression and return a list of all found values. If there are several, the values can be either pasted and/or casted from characters to numeric values.

Usage

```
StrVal(x, paste = FALSE, as.numeric = FALSE, dec = getOption("OutDec"))
```

Arguments

x	a character vector
paste	should separatetly extracted numbers be pasted together? This can be useful to reverse a prior format action. Default is FALSE.
as.numeric	logical value, determining if the extracted values should be converted to a number or be returned as characters. Default is FALSE.
dec	character string containing a single character. The preferred character to be used as the decimal point. Defaults getOption("OutDec").

Details

If there are multiple numbers in the same string to paste and cast to numeric, pasting will be done first and after pasting the conversion will be performed. So if for example the numbers in x = "34 way 066" should be extracted StrVal(x, paste = TRUE, as.numeric = TRUE) will lead to 34066. This is a useful choice for converting formatted numbers having some kind of bigmark.

Value

depending on the results the function will return either a character vector, in the case every element of x contained only one number, or a list of character vectors containing the found numbers.

Author(s)

Andri Signorell <andri@signorell.net>, Markus Naepflin <markus@naepfl.in> provided an optimized regex

See Also

other string functions in DescTools-package, section String functions

Examples

```
# a simple vector with only one number per element
StrVal(x=c("week 1", "week 3", "week 4", "week 5"))

# several numbers per element, extract each part, do not paste and return characters
StrVal(x=c("This is 1. place: 45.2", "none", "12.1 but -2.7 follow, 10.2e23 "),
       paste = FALSE, as.numeric = FALSE)

# critical are numbers combined with signs, where we sequentially extract valid numbers
StrVal(x=c("78-23-99", "1e-15-34*789+9", "- 34values"),
       paste = FALSE, as.numeric = FALSE)

# a typical use case for this function is to reverse a previously
#   applied number format

x <- c(100000, 4564654632, -456463)
xf <- Format(x, big.mark="'")

StrVal(xf, paste = TRUE, as.numeric = TRUE)

StrVal(xf, paste = TRUE, as.numeric = FALSE)
StrVal(xf, paste = FALSE, as.numeric = TRUE)
StrVal(xf, paste = FALSE, as.numeric = FALSE)

# use an alternative decimal point
StrVal("8 452,12", dec=",")
```

StuartMaxwellTest *Stuart-Maxwell Marginal Homogeneity Test*

Description

This function computes the marginal homogeneity test for a $k \times k$ matrix of assignments of objects to k categories or two vectors x, y of category scores for n data objects by two raters. The statistic is distributed as χ^2 with k-1 degrees of freedom.
It can be viewed as an extension of the McNemar test to $k \times k$ table.

Usage

```
StuartMaxwellTest(x, y = NULL)
```

Arguments

x	either a 2-way $k \times k$ contingency table in matrix form, or a factor.
y	a factor with the same levels as x; ignored if x is a matrix.

Details

The null is that the probabilities of being classified into cells [i, j] and [j, i] are the same.

If x is a matrix, it is taken as a two-dimensional contingency table, and hence its entries should be nonnegative integers. Otherwise, both x and y must be vectors or factors of the same length and with the same levels.
Incomplete cases are removed, vectors are coerced into factors, and the contingency table is computed from these.

If there is perfect agreement for any category k, that category must be omitted in order to invert matrix S.

If for any category k, all frequencies in row k and column k are 0, except possibly for the main diagonal element (e.g., for perfect agreement for category k, in such cases also the corresponding row and column marginal frequencies would be equal), then the category is not included in the test and should be ignored, say the Stuart-Maxwell test is performed with respect to the remaining categories only. The degree of freedom df in this case can still be considered k - 1, where k is the number of original categories; this treats omitted categories as if they were included but contributed 0 to the value of χ^2 - a reasonable view since such categories have equal row and column marginals. (See: https://www.john-uebersax.com/stat/mcnemar.htm#stuart)

Value

A list with class "htest" containing the following components:

statistic	the value of the test statistic.
parameter	the degrees of freedom.
p.value	the p-value of the test.
method	a character string indicating what type of test was performed.
data.name	a character string giving the name of the data.

Author(s)

Andri Signorell <andri@signorell.net>, based on Code from Jim Lemon

References

Stuart, A (1955) A test for homogeneity of the marginal distributions in a two-way classification. *Biometrika*, 42, 412-416.

Maxwell, A.E. (1970) Comparing the classification of subjects by two independent judges. *British Journal of Psychiatry*, 116, 651-655.

Agresti, A. (2002) *Categorical Data Analysis*. John Wiley & Sons, pp 86 ff.

See Also

BhapkarTest for a more powerful alternative to the Stuart-Maxwell test

mcnemar.test, chisq.test, MHChisqTest, BreslowDayTest

Examples

```
# Source: https://www.john-uebersax.com/stat/mcnemar.htm#stuart
hyp <- as.table(matrix(c(20,3,0,10,30,5,5,15,40), nrow=3))
StuartMaxwellTest(hyp)

# same as defined with two vectors
d.hyp <- Untable(hyp)
StuartMaxwellTest(x=d.hyp[,1], y=d.hyp[,2])


mc <- as.table(matrix(c(
         732, 1524, 1575, 1577, 1602, 837, 1554, 1437,
         1672, 1600, 841, 1363, 1385, 1484, 1524, 791), nrow=4))

StuartMaxwellTest(mc)
```

StuartTauC *Stuart τ_c*

Description

Calculate Stuart's τ_c statistic, a measure of association for ordinal factors in a two-way table. The function has interfaces for a table (matrix) and for single vectors.

Usage

```
StuartTauC(x, y = NULL, conf.level = NA, ...)
```

Arguments

x	a numeric vector or a table. A matrix will be treated as table.
y	NULL (default) or a vector with compatible dimensions to x. If y is provided, table(x, y, ...) is calculated.
conf.level	confidence level of the interval. If set to NA (which is the default) no confidence interval will be calculated.
...	further arguments are passed to the function table, allowing i.e. to set useNA. This refers only to the vector interface.

Details

Stuart's τ_c makes an adjustment for table size in addition to a correction for ties. τ_c is appropriate only when both variables lie on an ordinal scale.
It is estimated by

$$\tau_c = \frac{2m \cdot (P - Q)}{n^2 \cdot (m - 1)}$$

where P equals the number of concordances and Q the number of discordances, n is the total amount of observations and m = min(R, C). The range of τ_c is [-1, 1].
See http://support.sas.com/documentation/cdl/en/statugfreq/63124/PDF/default/statugfreq.pdf, pp. 1739 for the estimation of the asymptotic variance.

The use of Stuart's Tau-c versus Kendall's Tau-b is recommended when the two ordinal variables under consideration have different numbers of values, e.g. good, medium, bad versus high, low.

Value

a single numeric value if no confidence intervals are requested,
and otherwise a numeric vector with 3 elements for the estimate, the lower and the upper confidence interval

Author(s)

Andri Signorell <andri@signorell.net>

References

Agresti, A. (2002) *Categorical Data Analysis*. John Wiley & Sons, pp. 57–59.

Brown, M.B., Benedetti, J.K.(1977) Sampling Behavior of Tests for Correlation in Two-Way Contingency Tables, *Journal of the American Statistical Association*, 72, 309-315.

Goodman, L. A., & Kruskal, W. H. (1954) Measures of association for cross classifications. *Journal of the American Statistical Association*, 49, 732-764.

Goodman, L. A., & Kruskal, W. H. (1963) Measures of association for cross classifications III: Approximate sampling theory. *Journal of the American Statistical Association*, 58, 310-364.

See Also

ConDisPairs yields concordant and discordant pairs

Other association measures:
GoodmanKruskalGamma, KendallTauA (τ_a), cor (method="kendall") for τ_b, SomersDelta
Lambda, GoodmanKruskalTau, UncertCoef, MutInf

Examples

```
# example in:
# http://support.sas.com/documentation/cdl/en/statugfreq/63124/PDF/default/statugfreq.pdf
# pp. S. 1821

tab <- as.table(rbind(c(26,26,23,18,9),c(6,7,9,14,23)))

StuartTauC(tab, conf.level=0.95)
```

SumCI *Add Up Partial Confidence Intervals to a Total CI*

Description

Starting with a response variable that obtains different confidence intervals (CI) when calculated with different explanatory variables, all the values of the response variable should be added up. This function returns the CI for the sum.

Usage

```
SumCI(x)
```

Arguments

x	a matrix with 3 columns, containing the estimate in the first column followed by the lower and the upper confidence interval .

Value

a vector with the sum and the lower, upper confidence bound of the confidence interval

Author(s)

Andri Signorell andri@signorell.net

References

https://stats.stackexchange.com/questions/223924/how-to-add-up-partial-confidence-intervals-to-crea

See Also

BinomCI,

Examples

```
x <- do.call(rbind,
             tapply(d.pizza$delivery_min,
                    d.pizza$area, MeanCI))
SumCI(x)
```

SysInfo *System Information*

Description

SysInfo is a convenience function to compile some information about the computing system and environment used.

Usage

```
SysInfo()
FindRProfile()
```

Details

The function SysInfo is mainly used to save the system environment information in ncdf files containing the results of some calculations.

FindRProfile returns path candidates where the profile could be found.

Value

character string with all version and system information of the current R system

Author(s)

Jannis v. Buttlar <jbuttlar@bgc-jena.mpg.de>, Andri Signorell <andri@signorell.net>

TextContrastColor *Choose Textcolor Depending on Background Color*

Description

Text of a certain color when viewed against certain backgrounds can be hard to see. TextContrastColor returns either black or white depending on which has the better contrast.

Usage

```
TextContrastColor(col, white = "white", black = "black", method = c("glynn", "sonego"))
```

Arguments

col	vector of any of the three kind of R colors, i.e., either a color name (an element of colors()), a hexadecimal string of the form "#rrggbb" or "#rrggbbaa" (see rgb), or an integer i meaning palette()[i]. Non-string values are coerced to integer.
white	the color for the dark backgrounds, default is "white".
black	the color for the bright backgrounds, default is "black"
method	defines the algorithm to be used. Can be one out of "glynn" or "sonego". See details.

Details

A simple heuristic in defining a text color for a given background color, is to pick the one that is "farthest" away from "black" or "white". The way Glynn chooses to do this is to compute the color intensity, defined as the mean of the RGB triple, and pick "black" (intensity 0) for text color if the background intensity is greater than 127, or "white" (intensity 255) when the background intensity is less than or equal to 127. Sonego calculates `L <- c(0.2, 0.6, 0) %*% col2rgb(color)/255` and returns "black" if L >= 0.2 and "white" else.

Value

a vector containing the contrast color (either black or white)

Author(s)

Andri Signorell <andri@signorell.net> based on code of Earl F. Glynn, Stowers Institute for Medical Research, 2004

Examples

```
# works fine for grays
PlotArea( y=matrix(rep(1, times=3, each=8), ncol=8), x=1:3,
  col=gray(1:8 / 8), ylab="", xlab="", axes=FALSE )
text( x=2, y=1:8-0.5, levels(d.pizza$driver),
  col=TextContrastColor(gray(1:8 / 8)))

# and not so fine, but still ok, for colors
par(mfrow=c(1,2))
PlotArea( y=matrix(rep(1, times=3, each=12), ncol=12), x=1:3,
  col=rainbow(12), ylab="", xlab="", axes=FALSE, main="method = Glynn" )
text( x=2, y=1:12-0.5, levels(d.pizza$driver),
  col=TextContrastColor(rainbow(12)))

PlotArea( y=matrix(rep(1, times=3, each=12), ncol=12), x=1:3,
  col=rainbow(12), ylab="", xlab="", axes=FALSE, main="method = Sonego" )
text( x=2, y=1:12-0.5, levels(d.pizza$driver),
  col=TextContrastColor(rainbow(12), method="sonego"))
```

TextToTable *Converts String To a Table*

Description

Try to convert a string to a table, by first creating a data frame using `read.table`. This can then be coerced to a matrix first, and subsequently to a table. The names of the dimensions can be specified.

Usage

```
TextToTable(x, dimnames = NULL, check.names = FALSE, ...)
```

Arguments

`x`	the string to be interpreted as table.
`dimnames`	the names of the dimensions.
`check.names`	passed on to `read.table` and determines, if invalid column names should be adapted to valid ones. The default here is changed to `FALSE`.
`...`	the dots will be passed to the function `read.table` and can be used for example to specify `header`, `sep` and `row.names` arguments.

Value

a table

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`read.table`, `as.table`, `as.matrix`

Examples

```
txt <- "
    Democrat, Independent, Republican
  M, 762, 327, 468
  F, 484, 239, 477"

(tab <- TextToTable(txt, header=TRUE, sep=",", dimnames=c("gender", "party")))
```

`TheilU` *Theil's U Index of Inequality*

Description

Calculate Theil's U index of inequality.

Usage

```
TheilU(a, p, type = c(2, 1), na.rm = FALSE)
```

Arguments

`a`	a numeric vector with the actual observed values.
`p`	a numeric vector containing the predictions.
`type`	defining the type of Theil's two U measures, see Details. Default is 2.
`na.rm`	logical, indicating whether `NA` values should be stripped before the computation proceeds. If set to `TRUE` complete cases of `cbind(x, y)` will be used. Defaults to `FALSE`.

Details

Theil proposed two error measures, but at different times and under the same symbol U, which has caused some confusion. U `type = 1` is taken from Theil (1958, pp. 31-42). The argument `a` represents the actual observations and `p` the corresponding predictions. He left it open whether `a` and `p` should be used as absolute values or as observed and predicted changes.
Theil (1966, chapter 2) proposed U `type = 2` as a measure of forecast quality: *"...where A_i and P_i stand for a pair of predicted and observed changes. ..."*
As U_1 has some serious disadvantages (see Bliemel 1973) it is recommended to use U_2.

Author(s)

Andri Signorell <andri@signorell.net>

References

Theil, H. (1958): *Economic Forecasts and Policy*. Amsterdam: North Holland.

Thiel, H. (1966): *Applied Economic Forecasting*. Chicago: Rand McNally.

Bliemel, F. (1973): Theil's Forecast Accuracy Coefficient: A Clarification, *Journal of Marketing Research* Vol. 10, No. 4 (Nov., 1973), pp. 444-446

See Also

`Gini`

Examples

```
TheilU(1:10, 2:11, type=1)
TheilU(1:10, 2:11, type=2)
```

`TitleRect` *Plot Boxed Annotation*

Description

The function can be used to add a title to a plot surrounded by a rectangular box. This is useful for plotting several plots in narrow distances.

Usage

```
TitleRect(label, bg = "grey", border = 1, col = "black", xjust = 0.5,
          line = 2, ...)
```

Arguments

`label`	the main title
`bg`	the background color of the box.
`border`	the border color of the box
`col`	the font color of the title
`xjust`	the x-justification of the text. This can be `c(0, 0.5, 1)` for left, middle- and right alignement.
`line`	on which MARgin line, starting at 0 counting outwards
`...`	the dots are passed to the `text` function, which can be used to change font and similar arguments.

Value

nothing is returned

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`title`

Examples

```
plot(pressure)
TitleRect("pressure")
```

TMod *Comparison Table For Linear Models*

Description

Collect the coefficients and some qualifying statistics of linear models and organize it in a table for comparison and reporting. The function supports linear and general linear models.

Usage

```
TMod(..., FUN = NULL, order = NA, verb = FALSE)

ModSummary(x, ...)

## S3 method for class 'lm'
ModSummary(x, conf.level = 0.95, ...)
## S3 method for class 'glm'
ModSummary(x, conf.level = 0.95, use.profile = TRUE, ...)

## S3 method for class 'TMod'
plot(x, terms = NULL, intercept = FALSE, ...)
## S3 method for class 'TMod'
print(x, digits = 3, na.form = "-", verb = NULL, ...)
```

Arguments

x	a (general) linear model object.
...	a list of (general) linear models.
conf.level	the level for the confidence intervals.
FUN	function with arguments est, se, tval, pval, lci, uci to display the coefficients. The default function will display the coefficient and significance stars for the p-values.
order	row of the results table to be used as order for the models (as typically "AIC"). Can be any label in the first column of the results table. Default is NA for no special order.
verb	logical, determining whether the full set of model performance indicators (TRUE) or a reduced set should be displayed (FALSE is default).
terms	a vector with the terms of the model formula to be plotted. By default this will be all of them.
use.profile	logical. Defines if profile approach should be used, which normally is a good choice for small datasets. Calculating profile can however take ages for large datasets and not be necessary there. So we can fallback to normal confidence intervals.
intercept	logical, defining whether the intercept should be plotted (default is FALSE).

digits	integer, the desired (fixed) number of digits after the decimal point. Unlike formatC you will always get this number of digits even if the last digit is 0.
na.form	character, string specifying how NAs should be specially formatted. If set to NULL (default) no special action will be taken.

Details

In order to compare the coefficients of linear models, the user is left to his own devices. R offers no support in this respect. TMod() jumps into the breach and displays the coefficients of several models in tabular form. For this purpose, different quality indicators for the models are displayed, so that a comprehensive comparison of the models is possible. In particular, it is easy to see the effect that adding or omitting variables has on forecast quality.

A plot function for a TMod object will produce a dotchart with the coefficients and their confidence intervals.

Value

character table

Author(s)

Andri Signorell <andri@signorell.net>

See Also

help

Examples

```
r.full <- lm(Fertility ~ . , swiss)
r.nox <- lm(Fertility ~ . -Examination - Catholic, swiss)
r.grp <- lm(Fertility ~ . -Education - Catholic + CutQ(Catholic), swiss)
r.gam <- glm(Fertility ~ . , swiss, family=Gamma(link="identity"))
r.gama <- glm(Fertility ~ .- Agriculture , swiss, family=Gamma(link="identity"))
r.gaml <- glm(Fertility ~ . , swiss, family=Gamma(link="log"))

TMod(r.full, r.nox, r.grp, r.gam, r.gama, r.gaml)

# display confidence intervals
TMod(r.full, r.nox, r.gam, FUN = function(est, se, tval, pval, lci, uci){
  gettextf("%s [%s, %s]",
           Format(est, fmt=Fmt("num")),
           Format(lci, digits=3),
           Format(uci, digits=2)
           )
})


# cbind interface is not supported!!
# d.titanic <- reshape(as.data.frame(Titanic),
#                      idvar = c("Class","Sex","Age"),
```

```
#                       timevar="Survived",
#                       direction = "wide")
#
# r.glm0 <- glm(cbind(Freq.Yes, Freq.No) ~ 1, data=d.titanic, family="binomial")
# r.glm1 <- glm(cbind(Freq.Yes, Freq.No) ~ Class, data=d.titanic, family="binomial")
# r.glm2 <- glm(cbind(Freq.Yes, Freq.No) ~ ., data=d.titanic, family="binomial")

d.titanic <- Untable(Titanic)

r.glm0 <- glm(Survived ~ 1, data=d.titanic, family="binomial")
r.glm1 <- glm(Survived ~ Class, data=d.titanic, family="binomial")
r.glm2 <- glm(Survived ~ ., data=d.titanic, family="binomial")

TMod(r.glm0, r.glm1, r.glm2)

# plot OddsRatios
d.pima <- MASS::Pima.tr2

r.a <- glm(type ~ npreg + bp + skin + bmi + ped + age, data=d.pima, family=binomial)
r.b <- glm(type ~ npreg + glu + bp + skin, data=d.pima, family=binomial)
r.c <- glm(type ~ npreg + age, data=d.pima, family=binomial)

or.a <- OddsRatio(r.a)
or.b <- OddsRatio(r.b)
or.c <- OddsRatio(r.c)


# create the model table
tm <- TMod(m_A=or.a, m_B=or.b, m_C=or.c)
# .. and plotit
plot(tm, main="ORs for Models A, B, C", intercept=FALSE,
     pch=15, col=c(DescTools::hred, DescTools::hblue, DescTools::horange),
     panel.first=abline(v=1, col="grey30"))
```

`ToLong, ToWide` *Reshape a Vector From Long to Wide Shape Or Vice Versa*

Description

Simple reshaping a vector from long to wide or from wide to long shape by means of a single factor.

Usage

```
ToLong(x, varnames = NULL)
ToWide(x, g, by = NULL, varnames = NULL)
```

Arguments

x	the vector to be reshaped

g	the grouping vector to be used for the new columns. The resulting data.frame will return one column per grouplevel.
by	a vector to be used to merge the pieces of x. If this is left to NULL the pieces will be merged by rownames in the order they are supplied.
varnames	the variable names if not the grouping levels should be used.

Details

ToLong expects x as a matrix or a data.frame and reshapes it to a (long) factor representation. ToWide expects the vectors x, g, by, wheras x being the variable, g the splitting factor and by a vector for rowwise merging.

Value

the reshaped object

Author(s)

Andri Signorell <andri@signorell.net>

See Also

reshape

Examples

```
d.x <- read.table(header=TRUE, text="
AA BB CC DD EE FF GG
7.9 18.1 13.3 6.2 9.3 8.3 10.6
9.8 14.0 13.6 7.9 2.9 9.1 13.0
6.4 17.4 16.0 10.9 8.6 11.7 17.5
")

ToLong(d.x)

# ToWide by row numbers (by = NULL)
ToWide(PlantGrowth$weight, PlantGrowth$group)

# To wide aligned by key
set.seed(41)
PlantGrowth$nr <- c(sample(12, 10), sample(12, 10), sample(12, 10))
head(PlantGrowth)

ToWide(PlantGrowth$weight, PlantGrowth$group, by=PlantGrowth$nr)
```

TOne *Create Table One Describing Baseline Characteristics*

Description

Create a table summarizing continuous, categorical and dichotomous variables, optionally stratified by one or more variables, while performing adequate statistical tests.

Usage

```
TOne(
  x,
  grp = NA,
  add.length = TRUE,
  colnames = NULL,
  vnames = NULL,
  total = TRUE,
  align = "\\l",
  FUN = NULL,
  TEST = NULL,
  intref = "high",
 fmt = list(abs = Fmt("abs"), num = Fmt("num"), per = Fmt("per"), pval = as.fmt(fmt =
    "*", na.form = "   "))
)
```

Arguments

x	a data.frame containing all the variables to be included in the table.
grp	the grouping variable.
add.length	logical. If set to TRUE (default), a row with the group sizes will be inserted as first row of the table.
colnames	a vector of column names for the result table.
vnames	a vector of variable names to be placed in the first column instead of the real names.
total	logical (default TRUE), defines whether the results should also be displayed for the whole, ungrouped variable.
align	the character on whose position the strings will be aligned. Left alignment can be requested by setting sep = "\l", right alignment by "\r" and center alignment by "\c". Mind the backslashes, as if they are omitted, strings would be aligned to the **character l**, **r** or **c** respectively. Default value is "\l", thus left alignment.
FUN	the function to be used as location and dispersion measure for numeric (including integer) variables (mean/sd is default, alternatives as median/IQR are possible by defining a function). See examples.

TEST	a list of functions to be used to test the variables. Must be named as "num", "cat" and "dich" and be defined as function with arguments (x, g), generating something similar to a p-value. Use TEST=NA to suppress test. (See examples.)
intref	one out of "high" (default) or "low", defining which value of a dichotomous numeric or logical variable should be reported. Usually this will be 1 or TRUE. Setting it to "low" will report the lower value 0 or FALSE.
fmt	format codes for absolute, numeric and percentage values, and for the p-values of the tests.

Details

In research the characteristics of study populations are often characterised through some kind of a "Table 1", containing descriptives of the used variables, as mean/standard deviation for continuous variables, and proportions for categorical variables. In many cases, a comparison is made between groups within the framework of the scientific question.

```
 var Brent Camden Westminster n 474 (39.5
(31.8
Butcher 72 (15.2
(58.2
11 (2.9
77 (20.3
(50.3
exact test, "') Chi-Square test Signif. codes: 0 '***' 0.001 '**' 0.01 '*'
0.05 '.' 0.1 ' ' 1
```

Creating such a table can be very time consuming and there's a need for a flexible function that helps us to solve the task. TOne() is designed to be easily used with sensible defaults, and yet flexible enough to allow free definition of the essential design elements.

This is done by breaking down the descriptive task to three types of variables: quantitative (numeric, integer), qualitative (factor, characters) and dichotomous variables (the latter having exactly two values or levels). Depending on the variable type, the descriptives and the according sensible tests are chosen. By default mean/sd are chosen to describe numeric variables.

```
 FUN = function(x) gettextf("
Format(mean(x, na.rm = TRUE), digits = 1), Format(sd(x, na.rm = TRUE),
digits = 3))
```

Their difference is tested with the Kruskal-Wallis test. For categorical variables the absolute and relative frequencies are calculated and tested with a chi-square test.
The tests can be changed with the argument TEST. These must be organised as list containing elements named "num", "cat" and "dich". Each of them must be a function with arguments (x, g), returning something similar to a p-value.

```
 TEST = list( num = list(fun = function(x,
g){summary(aov(x ~ g))[[1]][1, "Pr(>F)"]}, lbl = "ANOVA"), cat = list(fun =
function(x, g){chisq.test(table(x, g))$p.val}, lbl = "Chi-Square test"),
dich = list(fun = function(x, g){fisher.test(table(x, g))$p.val}, lbl =
"Fisher exact test"))
```

The legend text of the test, which is appended to the table together with the significance codes, can be set with the variable `lbl`.

Great importance was attached to the free definition of the number formats. By default, the optionally definable format templates of **DescTools** are used. Deviations from this can be freely passed as arguments to the function. Formats can be defined for integers, floating point numbers, percentages and for the p-values of statistical tests. All options of the function `Format()` are available and can be provided as a list. See examples which show several different implementations.

```
fmt = list(abs = Fmt("abs"), num = Fmt("num"), per = Fmt("per"), pval =
as.fmt(fmt = "*", na.form = " "))
```

The function returns a character matrix as result, which can easily be subset or combined with other matrices. An interface for `ToWrd()` is available such that the matrix can be transferred to MS-Word. Both font and alignment are freely selectable in the Word table.

Value

a character matrix

Author(s)

Andri Signorell andri@signorell.net

See Also

`WrdTable()`, `ToWrd.TOne()`

Examples

```
options(scipen = 8)
opt <- DescToolsOptions()

# define some special formats for count data, percentages and numeric results
# (those will be supported by TOne)
Fmt(abs = as.fmt(digits = 0, big.mark = "'"))   # counts
Fmt(per = as.fmt(digits = 1, fmt = "%"))        # percentages
Fmt(num = as.fmt(digits = 1, big.mark = "'"))   # numeric

TOne(x = d.pizza[, c("temperature", "delivery_min", "driver", "wine_ordered")],
  grp = d.pizza$quality)

# the same but no groups now...
TOne(x = d.pizza[, c("temperature", "delivery_min", "driver", "wine_ordered")])

# define median/IQR as describing functions for the numeric variables
TOne(iris[, -5], iris[, 5],
  FUN = function(x) {
    gettextf("%s / %s",
      Format(median(x, na.rm = TRUE), digits = 1),
      Format(IQR(x, na.rm = TRUE), digits = 3))
  }
```

```
)

# replace kruskal.test by ANOVA and report the p.value
# Change tests for all the types
TOne(x = iris[, -5], grp = iris[, 5],
     FUN = function(x) gettextf("%s / %s",
            Format(mean(x, na.rm = TRUE), digits = 1),
            Format(sd(x, na.rm = TRUE), digits = 3)),

     TEST = list(
       num = list(fun = function(x, g){summary(aov(x ~ g))[[1]][1, "Pr(>F)"]},
                  lbl = "ANOVA"),
             cat = list(fun = function(x, g){chisq.test(table(x, g))$p.val},
                  lbl = "Chi-Square test"),
             dich = list(fun = function(x, g){fisher.test(table(x, g))$p.val},
                   lbl = "Fisher exact test")),
       fmt = list(abs = Fmt("abs"), num  = Fmt("num"), per = Fmt("per"),
                pval = as.fmt(fmt = "*", na.form = "   "))
)

t1 <- TOne(x     = d.pizza[,c("temperature", "driver", "rabate")],
           grp   = d.pizza$area,
           align = " ",
           total = FALSE,

           FUN = function(x) gettextf("%s / %s (%s)",
                                      Format(mean(x, na.rm = TRUE), digits = 1),
                                      Format(sd(x, na.rm = TRUE), digits = 3),
                                      Format(median(x, na.rm = TRUE), digits = 1)),

           TEST = NA,

           fmt = list(abs  = as.fmt(big.mark = " ", digits=0),
                      num  = as.fmt(big.mark = " ", digits=1),
                      per  = as.fmt(fmt=function(x)
                          StrPad(Format(x, fmt="%", d=1), width=5, adj = "r")),
                      pval = as.fmt(fmt = "*", na.form = "   "))
)
# add a userdefined legend
attr(t1, "legend") <- "numeric: mean / sd (median)), factor: n (n%)"

t1


# dichotomous integer or logical values can be reported by the high or low value
x <- sample(x = c(0, 1), size = 100, prob = c(0.3, 0.7), replace = TRUE)
y <- sample(x = c(0, 1), size = 100, prob = c(0.3, 0.7), replace = TRUE) == 1
z <- factor(sample(x = c(0, 1), size = 100, prob = c(0.3, 0.7), replace = TRUE))
g <- sample(x = letters[1:4], size = 100, replace = TRUE)
d.set <- data.frame(x = x, y = y, z = z, g = g)

TOne(d.set[1:3], d.set$g, intref = "low")
```

```
TOne(d.set[1:3], d.set$g, intref = "high")

# intref would not control factors, use relevel to change reported value
TOne(data.frame(z = relevel(z, "1")), g)

TOne(data.frame(z = z), g)

options(opt)


## Not run:

# Send the whole stuff to Word
wrd <- GetNewWrd()
ToWrd(
  TOne(x   = d.pizza[, c("temperature", "delivery_min", "driver", "wine_ordered")],
       grp = d.pizza$quality,
       fmt = list(num=Fmt("num", digits=1))
       ),
  font = list(name="Arial narrow", size=8),
  align = c("l","r")      # this will be recycled: left-right-left-right ...
)

## End(Not run)
```

ToWrd *Send Objects to Word*

Description

Send objects like tables, ftables, lm tables, TOnes or just simple texts to a MS-Word document.

Usage

```
ToWrd(x, font = NULL, ..., wrd = DescToolsOptions("lastWord"))

## S3 method for class 'Freq'
ToWrd(x, font = NULL, main = NULL, ..., wrd = DescToolsOptions("lastWord"))

## S3 method for class 'table'
ToWrd(x, font = NULL, main = NULL, align = NULL,
      tablestyle = NULL, autofit = TRUE,
     row.names = TRUE, col.names = TRUE, ..., wrd = DescToolsOptions("lastWord"))

## S3 method for class 'data.frame'
ToWrd(x, font = NULL, main = NULL, row.names = NULL, ...,
      wrd = DescToolsOptions("lastWord"))
```

```
## S3 method for class 'ftable'
ToWrd(x, font = NULL, main = NULL, align = NULL,
      method = "compact", ..., wrd = DescToolsOptions("lastWord"))

## S3 method for class 'TOne'
ToWrd(x, font = NULL, para = NULL, main = NULL, align = NULL,
      autofit = TRUE, ..., wrd = DescToolsOptions("lastWord"))

## S3 method for class 'TMod'
ToWrd(x, font = NULL, para = NULL, main = NULL, align = NULL,
      split = " ", fixed=TRUE, autofit = TRUE, digits = 3, na.form = "-", ...,
      wrd = DescToolsOptions("lastWord"))

## S3 method for class 'lm'
ToWrd(x, font = NULL, ..., wrd = DescToolsOptions("lastWord"))

## S3 method for class 'character'
ToWrd(x, font = NULL, para = NULL, style = NULL, bullet = FALSE,
      ..., wrd = DescToolsOptions("lastWord"))

## Default S3 method:
ToWrd(x, font = NULL, ..., wrd = DescToolsOptions("lastWord"))
```

Arguments

x	the object to be transferred to Word.
font	the font to be used to the output. This should be defined as a list containing fontname, fontsize, bold and italic flags: `list(name="Arial", size=10, bold=FALSE, italic=TRUE)`.
para	list containing paragraph format properties to be applied to the inserted text. For right align the paragraph one can set: `list(alignment="r", LineBefore=0.5)`. See details for the full set of properties.
main	a caption for a table. This will be inserted by `WrdCaption` in Word and can be listed afterwards in a specific index. Default is `NULL`, which will insert nothing. Ignored if `x` is not a table.
align	character vector giving the alignment of the table columns. `"l"` means left, `"r"` right and `"c"` center alignement. The code will be recyled to the length of thenumber of columns.
method	string specifying how the `"ftable"` object is formatted (and printed if used as in `write.ftable()` or the `print` method). Can be abbreviated. Available methods are (see the examples):
	`"non.compact"` the default representation of an `"ftable"` object.
	`"row.compact"` a row-compact version without empty cells below the column labels.

	`"col.compact"` a column-compact version without empty cells to the right of the row labels. `"compact"` a row- and column-compact version. This may imply a row and a column label sharing the same cell. They are then separated by the string `lsep`.
`autofit`	logical, defining if the columns of table should be fitted to the length of their content.
`row.names`	logical, defining whether the row.names should be included in the output. Default is `FALSE`.
`col.names`	logical, defining whether the col.names should be included in the output. Default is `TRUE`.
`tablestyle`	either the name of a defined Word tablestyle or its index.
`style`	character, name of a style to be applied to the inserted text.
`...`	further arguments to be passed to or from methods.
`bullet`	logical, defines if the text should be formatted as bullet points.
`split`	character vector (or object which can be coerced to such) containing regular expression(s) (unless `fixed = TRUE`) to use for splitting. If empty matches occur, in particular if `split` has length 0, x is split into single characters. If `split` has length greater than 1, it is re-cycled along x.
`fixed`	logical. If TRUE match split exactly, otherwise use regular expressions. Has priority over perl.
`digits`	integer, the desired (fixed) number of digits after the decimal point. Unlike `formatC` you will always get this number of digits even if the last digit is 0.
`na.form`	character, string specifying how `NA`s should be specially formatted. If set to `NULL` (default) no special action will be taken.
`wrd`	the pointer to a word instance. Can be a new one, created by `GetNewWrd()` or an existing one, created by `GetCurrWrd()`. Default is the last created pointer stored in `DescToolsOptions("lastWord")`.

Details

The paragraph format can be defined by means of these properties:

`LeftIndent`, `RightIndent`, `SpaceBefore`, `SpaceBeforeAuto`, `SpaceAfter`, `SpaceAfterAuto`, `LineSpacingRule`, `Alignment`, `WidowControl`, `KeepWithNext`, `KeepTogether`, `PageBreakBefore`, `NoLineNumber`, `Hyphenation`, `FirstLineIndent`, `OutlineLevel`, `CharacterUnitLeftIndent`, `CharacterUnitRightIndent`, `CharacterUnitFirstLineIndent`, `LineUnitBefore`, `LineUnitAfter`, `MirrorIndents`.

Value

if `x` is a table a pointer to the table will be returned

Author(s)

Andri Signorell <andri@signorell.net>

See Also

GetNewWrd

Examples

```
## Not run:
# we can't get this through the CRAN test - run it with copy/paste to console

wrd <- GetNewWrd()
ToWrd("This is centered Text in Arial Black\n",
      para=list(Alignment=wdConst$wdAlignParagraphCenter,
                SpaceBefore=3, SpaceAfter=6),
      font=list(name="Arial Black", size=14),
      wrd=wrd)

sel <- wrd$Selection()$Borders(wdConst$wdBorderBottom)
sel[["LineStyle"]] <- wdConst$wdLineStyleSingle


t1 <- TOne(x = d.pizza[, c("temperature","delivery_min","driver","wine_ordered")],
           grp=d.pizza$wine_delivered)

ToWrd(t1, font=list(name="Algerian"), wrd=wrd)


tab <- table(d.pizza$driver, d.pizza$area)

tab <- table(d.pizza$driver, d.pizza$area)
ToWrd(tab, font = list(size=15, name="Arial"), row.names = TRUE, col.names = TRUE,
      main= "my Title", wrd=wrd)
ToWrd(tab, font = list(size=10, name="Arial narrow"),
      row.names = TRUE, col.names=FALSE, wrd=wrd)
ToWrd(tab, font = list(size=15, name="Arial"), align="r",
      row.names = FALSE, col.names=TRUE, wrd=wrd)
ToWrd(tab, font = list(size=15, name="Arial"),
      row.names = FALSE, col.names=FALSE, wrd=wrd)

ToWrd(tab, tablestyle = "Mittlere Schattierung 2 - Akzent 4",
      row.names=TRUE, col.names=TRUE, wrd=wrd)

ToWrd(Format(tab, big.mark = "'", digits=0), wrd=wrd)

zz <- ToWrd(Format(tab, big.mark = "'", digits=0), wrd=wrd)
zz$Rows(1)$Select()
WrdFont(wrd = wrd) <- list(name="Algerian", size=14, bold=TRUE)


# Send a TMod table to Word using a split to separate columns
r.ols <- lm(Fertility ~ . , swiss)
r.gam <- glm(Fertility ~ . , swiss, family=Gamma(link="identity"))

# Build the model table for some two models, creating a user defined
```

```
# reporting function (FUN) with | as column splitter
tm <- TMod(OLS=r.ols, Gamma=r.gam,
           FUN=function(est, se, tval, pval, lci, uci){
              gettextf("%s|[%s, %s]|%s",
                       Format(est, fmt=Fmt("num"), digits=2),
                       Format(lci, fmt=Fmt("num"), digits=2),
                       Format(uci, fmt=Fmt("num"), digits=2),
                       Format(pval, fmt="*")
              )})

# send it to Word, where we get a table with 3 columns per model
# coef | confint | p-val
wrd <- GetNewWrd()
ToWrd(tm, split="|", align=StrSplit("lrclrcl"))
)

## End(Not run)
```

ToWrdB *Send Objects to Word and Bookmark Them*

Description

Send objects like tables, ftables, lm tables, TOnes or just simple texts to a MS-Word document and place a bookmark on them. This has the advantage, that objects in a Word document can be updated later, provided the bookmark name has been stored.

Usage

```
ToWrdB(x, font = NULL, ..., wrd = DescToolsOptions("lastWord"),
        bookmark = gettextf("bmt%s", round(runif(1, min = 0.1) * 1e+09)))
```

Arguments

x	the object to be transferred to Word.
font	the font to be used to the output. This should be defined as a list containing fontname, fontsize, bold and italic flags: `list(name="Arial", size=10, bold=FALSE, italic=TRUE)`.
...	further arguments to be passed to or from methods.
wrd	the pointer to a word instance. Can be a new one, created by `GetNewWrd()` or an existing one, created by `GetCurrWrd()`. Default is the last created pointer stored in `DescToolsOptions("lastWord")`.
bookmark	the name of the bookmark.

Details

This function encapsulates ToWrd, by placing a bookmark over the complete inserted results. The given name can be questioned with `bm$name()`.

Value

a handle to the set bookmark

Author(s)

Andri Signorell <andri@signorell.net>

See Also

ToWrd, WrdInsertBookmark

Examples

```
## Not run:
# we can't get this through the CRAN test - run it with copy/paste to console

wrd <- GetNewWrd()
bm <- ToWrdB("This is text to be possibly replaced later.")

# get the automatically created name of the bookmark
bm$name()

WrdGoto(bm$name())
UpdateBookmark(...)

## End(Not run)
```

ToWrdPlot *Send a Plot to Word and Bookmark it*

Description

Evaluate given plot code to a tiff() device and imports the created plot in the currently open MS-Word document. The imported plot is marked with a bookmark that can later be used for a potential update (provided the bookmark name has been stored).

Usage

```
ToWrdPlot(plotcode, width = NULL, height = NULL, scale = 100,
          pointsize = 12, res = 300, crop = 0, title = NULL,
          wrd = DescToolsOptions("lastWord"),
          bookmark = gettextf("bmp%s", round(runif(1, min = 0.1) * 1e+09)))
```

Arguments

plotcode	code chunk needed for producing the plot
bookmark	character, the name of the bookmark
width	the width in cm of the plot in the Word document (default 15)
height	the height in cm of the plot in the Word document (default 9.3)
scale	the scale of the plot (default 100)
pointsize	the default pointsize of plotted text, interpreted as big points (1/72 inch) at res ppi. (default is 12)
res	the resolution for the graphic (default 300)
crop	a vector of 4 elements, the crop factor for all 4 sides of a picture in cm (default all 0)
title	character, the title of the plot to be inserted in the word document
wrd	the pointer to a word instance. Can be a new one, created by GetNewWrd() or an existing one, created by GetCurrWrd(). Default is the last created pointer stored in DescToolsOptions("lastWord").

Details

An old and persistent problem that has existed for a long time is that as results once were loaded into a Word document the connection broke so that no update was possible. It was only recently that I realized that bookmarks in Word could be a solution for this. The present function evaluates some given plot code chunk using a tiff device and imports the created plot in a word document. The imported plot is given a bookmark, that can be used afterwards for changing or updating the plot.

This function is designed for use with the **DescToolsAddIns** functions ToWrdPlotWithBookmark() and ToWrdWithBookmark() allowing to assign keyboard shortcuts. The two functions will also insert the newly defined bookmark in the source file in a format, which can be interpreted by the function UpdateBookmark().

Value

a list

plot_hwnd	a windows handle to the inserted plot
bookmark	a windows handle to the bookmark

Author(s)

Andri Signorell <andri@signorell.net>

See Also

ToWrdB, WrdInsertBookmark

Examples

```
## Not run:
# we can't get this through the CRAN test - run it with copy/paste to console

wrd <- GetNewWrd()
bm <- ToWrdB("This is text to be possibly replaced later.")

# get the automatically created name of the bookmark
bm$name()

WrdGoto(bm$name())
UpdateBookmark(...)

## End(Not run)
```

`Triangular` *The Triangular Distribution*

Description

Density, distribution function, quantile function, and random generation for the triangular distribution with parameters `min`, `max`, and `mode`.

Usage

```
dTri(x, min = 0, max = 1, mode = 1/2)
pTri(q, min = 0, max = 1, mode = 1/2)
qTri(p, min = 0, max = 1, mode = 1/2)
rTri(n, min = 0, max = 1, mode = 1/2)
```

Arguments

`x`	vector of quantiles. Missing values (`NA`s) are allowed.
`q`	vector of quantiles. Missing values (`NA`s) are allowed.
`p`	vector of probabilities between 0 and 1. Missing values (`NA`s) are allowed.
`n`	sample size. If `length(n)` is larger than 1, then `length(n)` random values are returned.
`min`	vector of minimum values of the distribution of the random variable. The default value is `min=0`.
`max`	vector of maximum values of the random variable. The default value is `max=1`.
`mode`	vector of modes of the random variable. The default value is `mode=1/2`.

Details

Let X be a triangular random variable with parameters `min`=a, `max`=b, and `mode`=c.

Probability Density and Cumulative Distribution Function
The density function of X is given by:

$$f(x; a, b, c) = \begin{array}{ll} \frac{2(x-a)}{(b-a)(c-a)} & \text{for } a \leq x \leq c \\ \frac{2(b-x)}{(b-a)(b-c)} & \text{for } c \leq x \leq b \end{array}$$

where $a < c < b$.

The cumulative distribution function of X is given by:

$$F(x; a, b, c) = \begin{array}{ll} \frac{(x-a)^2}{(b-a)(c-a)} & \text{for } a \leq x \leq c \\ 1 - \frac{(b-x)^2}{(b-a)(b-c)} & \text{for } c \leq x \leq b \end{array}$$

where $a < c < b$.

Quantiles
The p^th quantile of X is given by:

$$x_p = \begin{array}{ll} a + \sqrt{(b-a)(c-a)p} & \text{for } 0 \leq p \leq F(c) \\ b - \sqrt{(b-a)(b-c)(1-p} & \text{for } F(c) \leq p \leq 1 \end{array}$$

where $0 \leq p \leq 1$.

Random Numbers
Random numbers are generated using the inverse transformation method:

$$x = F^{-1}(u)$$

where u is a random deviate from a uniform $[0, 1]$ distribution.

Mean and Variance
The mean and variance of X are given by:

$$E(X) = \frac{a + b + c}{3}$$

$$Var(X) = \frac{a^2 + b^2 + c^2 - ab - ac - bc}{18}$$

Value

`dTri` gives the density, `pTri` gives the distribution function, `qTri` gives the quantile function, and `rTri` generates random deviates.

Note

The triangular distribution is so named because of the shape of its probability density function. The average of two independent identically distributed uniform random variables with parameters min=α and max=β has a triangular distribution with parameters min=α, max=β, and mode=$(\beta-\alpha)/2$.

The triangular distribution is sometimes used as an input distribution in probability risk assessment.

Author(s)

Steven P. Millard (<EnvStats@ProbStatInfo.com>)

References

Forbes, C., M. Evans, N. Hastings, and B. Peacock. (2011). Statistical Distributions. Fourth Edition. John Wiley and Sons, Hoboken, NJ.

Johnson, N. L., S. Kotz, and N. Balakrishnan. (1995). *Continuous Univariate Distributions, Volume 2*. Second Edition. John Wiley and Sons, New York.

See Also

Uniform, Probability Distributions and Random Numbers.

Examples

```
# Density of a triangular distribution with parameters
# min=10, max=15, and mode=12, evaluated at 12, 13 and 14:

dTri(12:14, 10, 15, 12)
#[1] 0.4000000 0.2666667 0.1333333

#----------

# The cdf of a triangular distribution with parameters
# min=2, max=7, and mode=5, evaluated at 3, 4, and 5:

pTri(3:5, 2, 7, 5)
#[1] 0.06666667 0.26666667 0.60000000

#----------

# The 25'th percentile of a triangular distribution with parameters
# min=1, max=4, and mode=3:

qTri(0.25, 1, 4, 3)
#[1] 2.224745

#----------

# A random sample of 4 numbers from a triangular distribution with
# parameters min=3 , max=20, and mode=12.
# (Note: the call to set.seed simply allows you to reproduce this example.)
```

```
set.seed(10)
rTri(4, 3, 20, 12)
#[1] 11.811593  9.850955 11.081885 13.539496
```

`Trim` *Trim a Vector*

Description

Clean data by means of trimming, i.e., by omitting outlying observations.

Usage

```
Trim(x, trim = 0.1, na.rm = FALSE)
```

Arguments

`x`	a numeric vector to be trimmed.
`trim`	the fraction (0 to 0.5) of observations to be trimmed from each end of `x`. Values of trim outside that range (and < 1) are taken as the nearest endpoint. If `trim` is set to a value >1 it's interpreted as the number of elements to be cut off at each tail of `x`.
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.

Details

A symmetrically trimmed vector `x` with a fraction of trim observations (resp. the given number) deleted from each end will be returned. If `trim` is set to a value >0.5 or to an integer value > n/2 then the result will be `NA`.

Value

The trimmed vector `x`. The indices of the trimmed values will be attached as attribute named `"trim"`.

Note

This function is basically an excerpt from the base function `mean`, which allows the vector `x` to be trimmed before calculating the mean. But what if a trimmed standard deviation is needed?

Author(s)

R-Core (function mean), Andri Signorell <andri@signorell.net>

See Also

`Winsorize`

Examples

```
## generate data
set.seed(1234)     # for reproducibility
x <- rnorm(10)     # standard normal
x[1] <- x[1] * 10  # introduce outlier

## Trim data
x
Trim(x, trim=0.1)

## Trim fixed number, say cut the 3 extreme elements from each end
Trim(x, trim=3)

## check function
s <- sample(10:20)
s.tr <- Trim(s, trim = 2)
setequal(c(s[attr(s.tr, "trim")], s.tr), s)
```

TTestA *Student's t-Test Based on Sample Statistics*

Description

Performs one and two sample t-tests based on user supplied summary information instead of data as in `t.test()`.

Usage

```
TTestA(mx, sx, nx, my = NULL, sy = NULL, ny = NULL,
       alternative = c("two.sided", "less", "greater"),
       mu = 0, paired = FALSE, var.equal = FALSE,
       conf.level = 0.95, ...)
```

Arguments

`mx`	a single number representing the sample mean of x.
`my`	an optional single number representing the sample mean of y.
`sx`	a single number representing the sample standard deviation of x.
`sy`	an optional single number representing the sample standard deviation of y.
`nx`	a single number representing the sample size of x.
`ny`	an optional single number representing the sample size of y.
`alternative`	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`. You can specify just the initial letter.
`mu`	a number indicating the true value of the mean (or difference in means if you are performing a two sample test).

paired	paired = TRUE is not supported here and only present for consistency of arguments. Use the one-sample-test for the differences instead.
var.equal	a logical variable indicating whether to treat the two variances as being equal. If TRUE then the pooled variance is used to estimate the variance otherwise the Welch (or Satterthwaite) approximation to the degrees of freedom is used.
conf.level	confidence level of the interval.
...	further arguments to be passed to or from methods.

Details

alternative = "greater" is the alternative that x has a larger mean than y.

The option paired is not supported here, as the variance of the differences can't be calculated on the base of the variances of the two samples. However, for calculating the paired test we can simply supply the mean and standard deviation of the differences and use the one-sample test with mu = 0.

If var.equal is TRUE then the pooled estimate of the variance is used. By default, if var.equal is FALSE then the variance is estimated separately for both groups and the Welch modification to the degrees of freedom is used.

If the input data are effectively constant (compared to the larger of the two means) an error is generated.

Value

A list with class "htest" containing the following components:

statistic	the value of the t-statistic.
parameter	the degrees of freedom for the t-statistic.
p.value	the p-value for the test.
conf.int	a confidence interval for the mean appropriate to the specified alternative hypothesis.
estimate	the estimated mean or difference in means depending on whether it was a one-sample test or a two-sample test.
null.value	the specified hypothesized value of the mean or mean difference depending on whether it was a one-sample test or a two-sample test.
alternative	a character string describing the alternative hypothesis.
method	a character string indicating what type of t-test was performed.
data.name	a character string giving the name(s) of the data.

See Also

t.test

Examples

```
## Classical example: Student's sleep data
mx <- 0.75
my <- 2.33
sx <- 1.789010
sy <- 2.002249
nx <- ny <- 10
TTestA(mx=mx, my=my, sx=sx, sy=sy, nx=nx, ny=ny)

# compare to
with(sleep, t.test(extra[group == 1], extra[group == 2]))

# use the one sample test for the differences instead of paired=TRUE option
x <- with(sleep, extra[group == 1])
y <- with(sleep, extra[group == 2])

TTestA(mx=mean(x-y), sx=sd(x-y), nx=length(x-y))

# compared to
t.test(x, y, paired = TRUE)
```

TukeyBiweight *Calculate Tukey's Biweight Robust Mean*

Description

This calculates a robust average that is unaffected by outliers.

Usage

```
TukeyBiweight(x, const = 9, na.rm = FALSE,
              conf.level = NA, ci.type = "bca", R=1000, ...)
```

Arguments

x	a numeric vector
const	a constant. *const* is preassigned a value of 9 according to the Cook reference below but other values are possible.
na.rm	logical, indicating whether NA values should be stripped before the computation proceeds. Defaults to FALSE.
conf.level	confidence level of the interval. If set to NA (which is the default) no confidence interval will be calculated.
ci.type	The type of confidence interval required. The value should be any subset of the values "basic", "stud", "perc", "bca" or simply "all" which will compute all four types of intervals.

R	The number of bootstrap replicates. Usually this will be a single positive integer. For importance resampling, some resamples may use one set of weights and others use a different set of weights. In this case R would be a vector of integers where each component gives the number of resamples from each of the rows of weights.
...	the dots are passed to the function boot, when confidence intervalls are calculated.

Details

This is a one step computation that follows the Affy whitepaper below, see page 22. *const* determines the point at which outliers are given a weight of 0 and therefore do not contribute to the calculation of the mean. *const* = 9 sets values roughly +/-6 standard deviations to 0. *const* = 6 is also used in tree-ring chronology development. Cook and Kairiukstis (1990) have further details.

An exact summation algorithm (Shewchuk 1997) is used. When some assumptions about the rounding of floating point numbers and conservative compiler optimizations hold, summation error is completely avoided. Whether the assumptions hold depends on the platform, i.e. compiler and CPU.

Value

A numeric mean.

Author(s)

Mikko Korpela <mikko.korpela@aalto.fi>

References

Statistical Algorithms Description Document, 2002, Affymetrix.

Cook, E. R. and Kairiukstis, L. A. (1990) *Methods of Dendrochronology: Applications in the Environmental Sciences*. Springer. ISBN-13: 978-0792305866.

Mosteller, F. and Tukey, J. W. (1977) *Data Analysis and Regression: a second course in statistics*. Addison-Wesley. ISBN-13: 978-0201048544.

Shewchuk, J. R. (1997) Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. *Discrete and Computational Geometry*, 18(3):305-363. Springer.

See Also

HuberM, Range, RobScale

Examples

```
TukeyBiweight(rnorm(100))
```

TwoGroups *Describe a Variable by a Factor with Two Levels*

Description

This function describes a numeric variable by a grouping factor with two levels. First, a descriptive text listing the frequencies and means of the two groups and the results of the significance test is generated. The results of `Desc(x~g)` are reported as they are provided by the function, followed by a plot consisting of a density plot and a box plot. This description makes sense, for example, if the age distribution of a collective is to be represented for both sexes.

Usage

```
TwoGroups(x, ..., plotit = TRUE)

## Default S3 method:
TwoGroups(x, g, main = NULL, vname = NULL, ..., plotit = TRUE)

## S3 method for class 'formula'
TwoGroups(formula, data, subset, na.action, ...)

## S3 method for class 'TwoGroups'
ToWrd(x, font = NULL, ..., wrd = DescToolsOptions("lastWord"))
```

Arguments

`x`	the numeric variable to describe.
`g`	the grouping factor (preferably with two levels.)
`main`	the main title.
`vname`	the variable names used in the description text.
`plotit`	boolean. Should a plot be created? Default can be defined by `DescToolsOptions(plotit=TRUE/FALSE)`, if it does not exist then it's set to `FALSE`.
`formula`	a formula of the form `lhs ~ rhs` where `lhs` gives the data values and rhs the corresponding groups.
`data`	an optional matrix or data frame (or similar: see `model.frame`) containing the variables in the formula `formula`. By default the variables are taken from `environment(formula)`.
`subset`	an optional vector specifying a subset of observations to be used.
`na.action`	a function which indicates what should happen when the data contain NAs. Defaults to `getOption("na.action")`.
`font`	the first font will be chosen for the introducing text, when sending the output to Word, the second for the description.

wrd	the pointer to a running MS Word instance, as created by GetNewWrd() (for a new one) or by GetCurrWrd() for an existing one. Default is NULL, which will report all results to the console.
...	the dots are sent to the internally used function Phrase(). They can be used to choose the language (lang) or provide variable name (xname).

Value

list with the results calculated by the used functions

Author(s)

Andri Signorell <andri@signorell.net>

See Also

Desc, PlotMultiDens, Phrase

Examples

```
x <- d.pizza$temperature
g <- factor(d.pizza$rabate)

# we can change the colors for the plot by setting the DescToolsOptions
DescToolsOptions(col=c(DescTools::horange, DescTools::hgreen))
TwoGroups(x, g, main="Temperature ~ Rebate")

# for an output to Word simply define the wrd argument
# wrd <- GetNewWrd()
# TwoGroups(x, g, font.desc=list(name="Consolas", size=8),
#           main="Temperature ~ Rebate", wrd=wrd)
```

UncertCoef *Uncertainty Coefficient*

Description

The uncertainty coefficient U(C|R) measures the proportion of uncertainty (entropy) in the column variable Y that is explained by the row variable X. The function has interfaces for a table, a matrix, a data.frame and for single vectors.

Usage

```
UncertCoef(x, y = NULL, direction = c("symmetric", "row", "column"),
           conf.level = NA, p.zero.correction = 1/sum(x)^2, ...)
```

Arguments

x	a numeric vector, a factor, matrix or data frame.
y	NULL (default) or a vector, an ordered factor, matrix or data frame with compatible dimensions to x.
direction	direction of the calculation. Can be "row" (default) or "column", where "row" calculates UncertCoef (R|C) ("column dependent").
conf.level	confidence level of the interval. If set to NA (which is the default) no confidence interval will be calculated.
p.zero.correction	slightly nudge zero values so that their logarithm can be calculated
...	further arguments are passed to the function table, allowing i.e. to set useNA. This refers only to the vector interface.

Details

The uncertainty coefficient is computed as

$$U(C|R) = \frac{H(X) + H(Y) - H(XY)}{H(Y)}$$

and ranges from [0, 1].

Value

Either a single numeric value, if no confidence interval is required,
or a vector with 3 elements for estimate, lower and upper confidence intervall.

Author(s)

Andri Signorell <andri@signorell.net> strongly based on code from Antti Arppe <antti.arppe@helsinki.fi>

References

Theil, H. (1972), *Statistical Decomposition Analysis*, Amsterdam: North-Holland Publishing Company.

See Also

Entropy, Lambda, Assocs

Examples

```
# example from Goodman Kruskal (1954)

m <- as.table(cbind(c(1768,946,115), c(807,1387,438), c(189,746,288), c(47,53,16)))
dimnames(m) <- list(paste("A", 1:3), paste("B", 1:4))
m
```

```
# direction default is "symmetric"
UncertCoef(m)
UncertCoef(m, conf.level=0.95)

UncertCoef(m, direction="row")
UncertCoef(m, direction="column")
```

`UnirootAll` *Finds many (all) roots of one equation within an interval*

Description

The function `UnirootAll` searches the interval from lower to upper for several roots (i.e., zero's) of a function `f` with respect to its first argument.

Usage

```
UnirootAll(f, interval, lower = min(interval), upper = max(interval),
            tol = .Machine$double.eps^0.5, maxiter = 1000, n = 100, ...)
```

Arguments

`f`	the function for which the root is sought.
`interval`	a vector containing the end-points of the interval to be searched for the root.
`lower`	the lower end point of the interval to be searched.
`upper`	the upper end point of the interval to be searched.
`tol`	the desired accuracy (convergence tolerance).
`maxiter`	the maximum number of iterations.
`n`	number of subintervals in which the root is sought.
`...`	additional named or unnamed arguments to be passed to `f` (but beware of partial matching to other arguments).

Details

`f` will be called as `f(x, ...)` for a numeric value of `x`.

Run `demo(Jacobandroots)` for an example of the use of `UnirootAll` for steady-state analysis.

See also second example of `gradient` This example is discussed in the book by Soetaert and Herman (2009).

Value

a vector with the roots found in the interval

Note

This is a verbatim copy from rootSolve::uniroot.all (v. 1.7).

Note

The function calls `uniroot`, the basic R-function.

It is not guaranteed that all roots will be recovered.

This will depend on `n`, the number of subintervals in which the interval is divided.

If the function "touches" the X-axis (i.e. the root is a saddle point), then this root will generally not be retrieved. (but chances of this are pretty small).

Whereas `unitroot` passes values one at a time to the function, `UnirootAll` passes a vector of values to the function. Therefore `f` should be written such that it can handle a vector of values. See last example.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

See Also

`uniroot` for more information about input.

Examples

```
## ======================================================================
##   Mathematical examples
## ======================================================================

# a well-behaved case...
fun <- function (x) cos(2*x)^3

curve(fun(x), 0, 10,main = "UnirootAll")

All <- UnirootAll(fun, c(0, 10))
points(All, y = rep(0, length(All)), pch = 16, cex = 2)

# a difficult case...
f <- function (x) 1/cos(1+x^2)
AA <- UnirootAll(f, c(-5, 5))
curve(f(x), -5, 5, n = 500, main = "UnirootAll")
points(AA, rep(0, length(AA)), col = "red", pch = 16)

f(AA)  # !!!


## ======================================================================
## Vectorisation:
## ======================================================================
# from R-help Digest, Vol 130, Issue 27
# https://stat.ethz.ch/pipermail/r-help/2013-December/364799.html

integrand1 <- function(x) 1/x*dnorm(x)
integrand2 <- function(x) 1/(2*x-50)*dnorm(x)
integrand3 <- function(x, C) 1/(x+C)
```

```
res <- function(C) {
  integrate(integrand1, lower = 1, upper = 50)$value +
  integrate(integrand2, lower = 50, upper = 100)$value -
  integrate(integrand3, C = C, lower = 1, upper = 100)$value
}

# uniroot passes one value at a time to the function, so res can be used as such
uniroot(res, c(1, 1000))

# Need to vectorise the function to use UnirootAll:
res <- Vectorize(res)
UnirootAll(res, c(1,1000))
```

Untable *Recover Original Data From Contingency Table*

Description

Recreates the data.frame out of a contingency table x.

Usage

```
Untable(x, ...)

## S3 method for class 'data.frame'
Untable(x, freq = "Freq", rownames = NULL, ...)

## Default S3 method:
Untable(x, dimnames = NULL, type = NULL, rownames = NULL, colnames = NULL, ...)
```

Arguments

x	a numeric vector, a matrix, a table or a data.frame. If x is a vector, a matrix or a table it is interpreted as frequencies which are to be inflated to the original list. If x is a data.frame it is interpreted as a table in frequency form (containing one or more factors and a frequency variable).
dimnames	the dimension names of x to be used for expanding. Can be used to expand a weight vector to its original values. If set to NULL (default) the dimnames of x will be used.
type	defines the data type generated. This allows to directly define factors or ordered factors, but also numeric values. See examples.
rownames	A names vector for the rownames of the resulting data.frame. If set to NULL (default) the names will be defined according to the table's dimnames.

colnames	A names vector for the colnames of the resulting data.frame. If set to NULL (default) the names will be defined according to the table's dimnames.
freq	character, the name of the frequency variable in case x is a data.frame.
...	further arguments passed to or from functions (not used here).

Details

For x being a vector this reduces to `rep(..., n)` with n as vector (which is not supported by `rep()`). NAs in the table will be treated as 0 without raising an error.

Value

a data.frame with the detailed data (even if x was a 1-dimensional table)

Author(s)

Andri Signorell <andri@signorell.net>

See Also

expand.grid, rep, gl, xtabs

Examples

```
d.titanic <- Untable(Titanic)
str(d.titanic)

# ... not the same as:
data.frame(Titanic)


tab <- table(set1=sample(letters[1:5], size=40, replace=TRUE),
             set2=sample(letters[11:15], size=40, replace=TRUE))
Untable(tab)


# return a numeric vector by setting type and coerce to a vector by [,]
Untable(c(6,2,2), type="as.numeric")[,]


# how to produce the original list based on frequencies, given as a data.frame
d.freq <- data.frame(xtabs(Freq ~ Sex + Survived, data=Titanic))

# a data list with each individual
d.data <- Untable( xtabs(c(1364, 126, 367, 344) ~ .,
             expand.grid(levels(d.freq$Sex),levels(d.freq$Survived))))
head(d.data)

# expand a weights vector
Untable(c(1,4,5), dimnames=list(c("Zurich","Berlin","London")))
```

```
# and the same with a numeric vector
Untable(c(1,4,5), dimnames=list(c(5,10,15)), type="as.numeric")[,]
# ... which again is nothing else than
rep(times=c(1,4,5), x=c(5,10,15))

# the data.frame interface
d.freq <- data.frame(f1=c("A","A","B","B"), f2=c("C","D","C","D"), Freq=c(1,2,3,4))
Untable(d.freq)
```

Unwhich *Inverse Which*

Description

The inverse function to which creates a logical vector/matrix from indices.

Usage

```
Unwhich(idx, n = max(idx), useNames = TRUE)
```

Arguments

idx	the indices as returned by which.
n	integer, the length of the original vector. This must not be less than max(idx), which is also the default.
useNames	logical, determining if the names of the indices should be preserved.

Value

a logical vector of the length n, with TRUE on the positions i.

Author(s)

Nick Sabbe

References

https://stackoverflow.com/questions/7659833/inverse-of-which

See Also

which

Examples

```
ll <- c(TRUE, FALSE, TRUE, NA, FALSE, FALSE, TRUE)
names(ll) <- letters[seq(ll)]
i <- which(ll)
# back again (loosing the names of the FALSEs)
Unwhich(i, length(ll))
```

VanWaerdenTest *van der Waerden Test*

Description

Performs a van der Waerden normal scores test.

Usage

```
VanWaerdenTest(x, ...)

## Default S3 method:
VanWaerdenTest(x, g, ...)

## S3 method for class 'formula'
VanWaerdenTest(formula, data, subset, na.action, ...)
```

Arguments

x	a numeric vector of data values, or a list of numeric data vectors. Non-numeric elements of a list will be coerced, with a warning.
g	a vector or factor object giving the group for the corresponding elements of x. Ignored with a warning if x is a list.
formula	a formula of the form response ~ group where response gives the data values and group a vector or factor of the corresponding groups.
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
...	further arguments to be passed to or from methods.

Details

VanWaerdenTest performs a van der Waerden test of the null that the location parameters of the distribution of x are the same in each group (sample). The alternative is that they differ in at least one.

The van der Waerden rank scores are defined as the ranks of data, i.e., $R[i], i = 1, 2, ..., n$, divided by $1 + n$ transformed to a normal score by applying the inverse of the normal distribution function, i.e., $\Phi^(- 1)(R[i]/(1 + n))$. The ranks of data are obtained by ordering the observations from all groups (the same way as kruskal.test does it).

If x is a list, its elements are taken as the samples to be compared, and hence have to be numeric data vectors. In this case, g is ignored, and one can simply use VanWaerdenTest(x) to perform the test. If the samples are not yet contained in a list, use VanWaerdenTest(list(x, ...)).

Otherwise, x must be a numeric data vector, and g must be a vector or factor object of the same length as x giving the group for the corresponding elements of x.

Value

A list with class "htest" containing the following components:

statistic	the van der Waerden statistic.
parameter	the degrees of freedom of the approximate chi-squared distribution of the test statistic.
p.value	the p-value of the test.
method	the character string "van-der-Waerden normal scores test".
data.name	a character string giving the names of the data.

References

Conover, W. J., Iman, R. L. (1979). On multiple-comparisons procedures, Tech. Rep. LA-7677-MS, Los Alamos Scientific Laboratory.

Conover, W. J. (1999). *Practical Nonparameteric Statistics* (Third Edition ed.). Wiley. pp. 396406.

See Also

normal_test in package **coin** where the test is implemented in a more general context (but has a quite unpractical interface).

Examples

```
## Hollander & Wolfe (1973), 116.
## Mucociliary efficiency from the rate of removal of dust in normal
##  subjects, subjects with obstructive airway disease, and subjects
##  with asbestosis.
x <- c(2.9, 3.0, 2.5, 2.6, 3.2) # normal subjects
y <- c(3.8, 2.7, 4.0, 2.4)      # with obstructive airway disease
z <- c(2.8, 3.4, 3.7, 2.2, 2.0) # with asbestosis

VanWaerdenTest(list(x, y, z))

## Equivalently,
x <- c(x, y, z)
g <- factor(rep(1:3, c(5, 4, 5)),
            labels = c("Normal subjects",
                       "Subjects with obstructive airway disease",
                       "Subjects with asbestosis"))
VanWaerdenTest(x, g)

## Formula interface.
require(graphics)
boxplot(Ozone ~ Month, data = airquality)
VanWaerdenTest(Ozone ~ Month, data = airquality)
```

Var *Variance*

Description

Var() computes the variance of x. If x is a matrix variances of the columns of x are computed. Varn returns the uncorrected sample variance (which is biased estimator for the sample variance).

Usage

```
Var(x, ...)

## S3 method for class 'Freq'
Var(x, breaks, ...)

## Default S3 method:
Var(x, weights = NULL, na.rm = FALSE, method = c("unbiased",  "ML"), ...)

VarN(x, na.rm = FALSE)
```

Arguments

x	a numeric vector, matrix or data frame.
weights	a numerical vector of weights the same length as x giving the weights to use for elements of x.
na.rm	logical. Should missing values be removed?
method	determines the estimator type; if "unbiased" (the default) then the usual unbiased estimate (using Bessel's correction) is returned, if "ML" then it is the maximum likelihood estimate for a Gaussian distribution. Uses stats:cov.wt for both methods.
breaks	breaks for calculating the variance for classified data as composed by Freq.
...	further arguments passed to or from other methods.

Details

Var is just another interface to Cov.

The denominator $n - 1$ is used which gives an unbiased estimator of the (co)variance for i.i.d. observations. These functions return NA when there is only one observation (whereas S-PLUS has been returning NaN), and fail if x has length zero.

Value

For r <- Cor(*, use = "all.obs"), it is now guaranteed that all(abs(r) <= 1).

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`cor`, `cov` for covariance and correlation matrices

`cor.test` for confidence intervals (and tests).

`cov.wt` for *weighted* covariance computation.

`sd` for standard deviation (vectors).

Examples

```
Var(1:10)  # 9.166667

Var(1:5, 1:5) # 2.5

# weighted Variance
set.seed(45)
(z <- as.numeric(names(w <- table(x <- sample(-10:20, size=50, replace=TRUE)))))
Var(z, w=w)

# check!
all.equal(Var(x), Var(z, w=w))


# Variance for frequency tables
Var(Freq(as.table(c(6,16,24,25,17))),
     breaks=c(0, 10, 20, 30, 40, 50))
```

VarCI *Confidence Intervals for the Variance*

Description

Calculates confidence intervals for the variance. Available approachs are the classical one using the ChiSquare distribution, a more robust version proposed by Bonett and the bootstrap options available in the package `boot`.

Usage

```
VarCI(x, method = c("classic", "bonett", "norm", "basic", "stud", "perc", "bca"),
      conf.level = 0.95, sides = c("two.sided", "left", "right"),
      na.rm = FALSE, R = 999)
```

Arguments

x	a (non-empty) numeric vector of data values.
method	vector of character strings representing the type of intervals required. The value should be any subset of the values "classic", "bonett", "norm", "basic", "stud", "perc", "bca". See boot.ci.
conf.level	confidence level of the interval.
sides	a character string specifying the side of the confidence interval, must be one of "two.sided" (default), "left" or "right". You can specify just the initial letter. "left" would be analogue to a hypothesis of "greater" in a t.test.
na.rm	logical. Should missing values be removed? Defaults to FALSE.
R	number of bootstrap replicates. Usually this will be a single positive integer. For importance resampling, some resamples may use one set of weights and others use a different set of weights. In this case R would be a vector of integers where each component gives the number of resamples from each of the rows of weights. See boot.

Details

The confidence interval for the variance is very sensitive to non-normality in the data. Bonett (2006) has proposed an interval that is nearly exact when the data is normally distributed and provides good performance for moderately non-normal data. See the references for the details.

Value

a numeric vector with 3 elements:

var	variance
lwr.ci	lower bound of the confidence interval
upr.ci	upper bound of the confidence interval

Author(s)

Andri Signorell <andri@signorell.net>

References

Bonett (2006) Approximate Confidence Interval for Standard Deviation of Nonnormal Distributions, *Computational Statistics and Data Analysis*, Vol. 50, pp. 775 - 782.
https://www.itl.nist.gov/div898/software/dataplot/refman1/auxillar/sdconfli.htm (might be outdated)

See Also

MeanCI, MedianCI, VarTest, Var

Examples

```
VarCI(d.pizza$price, na.rm=TRUE)
VarCI(d.pizza$price, conf.level=0.99, na.rm=TRUE)

x <- c(14.816, 14.863, 14.814, 14.998, 14.965, 14.824, 14.884, 14.838, 14.916,
       15.021, 14.874, 14.856, 14.860, 14.772, 14.980, 14.919)
VarCI(x, conf.level=0.9)

# and for the standard deviation
sqrt(VarCI(x, conf.level=0.9))


# from Bonett's paper
# expected results:
# ------------------------------------
#  conf.lvl       sd      lci      uci
# ------------------------------------
#      90.0   0.5168   0.3592   0.9359
#      95.0   0.5168   0.3263   1.0841
#      99.0   0.5168   0.2607   1.5109

p <- c(15.83, 16.01, 16.24, 16.42, 15.33, 15.44, 16.88, 16.31)
sqrt(VarCI(p, method="bonett", conf.level=0.9))
sqrt(VarCI(p, method="bonett"))
sqrt(VarCI(p, method="bonett", conf.level=0.99))

# some bootstrap intervals
VarCI(x, method="norm")
VarCI(x, method="perc")
VarCI(x, method="bca")
```

VarTest *ChiSquare Test for One Variance and F Test to Compare Two Variances*

Description

Performs either a one sample chi-squared test to compare the variance of a vector with a given value or an F test to compare the variances of two samples from normal populations.

Usage

```
VarTest(x, ...)

## Default S3 method:
VarTest(x, y,
        alternative = c("two.sided", "less", "greater"),
        ratio = 1, sigma.squared = 1,
        conf.level = 0.95, ...)
```

```
## S3 method for class 'formula'
VarTest(formula, data, subset, na.action, ...)
```

Arguments

x, y	numeric vectors of data values.
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter.
ratio	the hypothesized ratio of the population variances of x and y.
sigma.squared	a number indicating the true value of the variance, if one sample test is requested.
conf.level	confidence level for the returned confidence interval.
formula	a formula of the form lhs ~ rhs where lhs is a numeric variable giving the data values and rhs a factor with two levels giving the corresponding groups.
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
...	further arguments to be passed to or from methods.

Details

The formula interface is only applicable for the 2-sample tests.

The null hypothesis is that the ratio of the variances of the populations from which x and y were drawn, or in the data to which the linear models x and y were fitted, is equal to ratio.

Value

A list with class "htest" containing the following components:

statistic	the value of the F test statistic.
parameter	the degrees of the freedom of the F distribution of the test statistic.
p.value	the p-value of the test.
conf.int	a confidence interval for the ratio of the population variances.
estimate	the ratio of the sample variances of x and y.
null.value	the ratio of population variances under the null.
alternative	a character string describing the alternative hypothesis.
method	the character string "F test to compare two variances".
data.name	a character string giving the names of the data.

Author(s)

Andri Signorell <andri@signorell.net> (One sample test)
Two Sample test and help text from R-Core.

See Also

`var.test`, `bartlett.test` for testing homogeneity of variances in more than two samples from normal distributions; `ansari.test` and `mood.test` for two rank based (nonparametric) two-sample tests for difference in scale.

Examples

```
x <- rnorm(50, mean = 0, sd = 2)

# One sample test
VarTest(x, sigma.squared = 2.5)

# two samples
y <- rnorm(30, mean = 1, sd = 1)
VarTest(x, y)                 # Do x and y have the same variance?
VarTest(lm(x ~ 1), lm(y ~ 1))  # The same.
```

VecRot *Vector Rotation (Shift Elements)*

Description

Shift the elements of a vector in circular mode by k elements to the right (for positive k) or to the left (for negative k), such that the first element is at the (k+1)th position of the new vector and the last k elements are appended to the beginning.
`VecShift` does not attach the superfluous elements on one side to the other, but fills the resulting gaps with `NA`s.

Usage

```
VecRot(x, k = 1)
VecShift(x, k = 1)
```

Arguments

x	a vector of any type.
k	the number of elements to shift.

Details

The function will repeat the vector two times and select the appropriate number of elements from the required shift on.

Value

the shifted vector in the same dimensions as x.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

[, rep, lag

Examples

```
VecRot(c(1,1,0,0,3,4,8), 3)

VecRot(letters[1:10], 3)
VecRot(letters[1:10], -3)

VecShift(letters[1:10], 3)
VecShift(letters[1:10], -3)
```

VIF *Variance Inflation Factors*

Description

Calculates variance-inflation and generalized variance-inflation factors for linear and generalized linear models. It's a measure describing how much the variance of an estimated coefficient is increased because of collinearity.

Usage

```
VIF(mod)
```

Arguments

mod	an object that responds to coef, vcov, and model.matrix, such as an lm or glm object.

Details

If all terms in an unweighted linear model have 1 df, then the usual variance-inflation factors are calculated.

The vif are defined as

$$vif_j = \frac{1}{1 - R_j^2}$$

where R_j^2 equals the coefficient of determination for regressing the explanatory variable `j` in question on the other terms in the model. This is one of the well-known collinearity diagnostics.

If any terms in an unweighted linear model have more than 1 df, then generalized variance-inflation factors (Fox and Monette, 1992) are calculated. These are interpretable as the inflation in size of the confidence ellipse or ellipsoid for the coefficients of the term in comparison with what would be obtained for orthogonal data.

The generalized vifs are invariant with respect to the coding of the terms in the model (as long as the subspace of the columns of the model matrix pertaining to each term is invariant). To adjust for the dimension of the confidence ellipsoid, the function also prints $GVIF^{1/(2\times df)}$ where df is the degrees of freedom associated with the term.

Through a further generalization, the implementation here is applicable as well to other sorts of models, in particular weighted linear models and generalized linear models.

Values of vif up to 5 are usually interpreted as uncritical, values above 5 denote a considerable multicollinearity.

Value

A vector of vifs, or a matrix containing one row for each term in the model, and columns for the GVIF, df, and $GVIF^{1/(2\times df)}$.

Note

This is a verbatim copy from the function `car::vif`.

Author(s)

Henric Nilsson and John Fox <jfox@mcmaster.ca>

References

Fox, J. and Monette, G. (1992) Generalized collinearity diagnostics. *JASA*, **87**, 178–183.

Fox, J. (2008) *Applied Regression Analysis and Generalized Linear Models*, Second Edition. Sage.

Fox, J. and Weisberg, S. (2011) *An R Companion to Applied Regression*, Second Edition, Sage.

Examples

```
VIF(lm(Fertility ~ Agriculture + Education, data=swiss))
VIF(lm(Fertility ~ ., data=swiss))
```

Vigenere *Vigenere Cypher*

Description

Implements a Vigenere cypher, both encryption and decryption. The function handle keys and text of unequal length and discards non-alphabetic characters.

Usage

```
Vigenere(x, key = NULL, decrypt = FALSE)
```

Arguments

x	the text to be encrypted
key	the key to be used. If this remains to NULL the PasswordDlg will be presented and the key can be entered there.
decrypt	boolean defining if the text should be encrypted or decrypted.

Details

All characters beside charlist = c(LETTERS, letters, 0:9) will be discarded from the text and from the key.

Value

the encrypted, resp. decrypted text

Author(s)

Andri Signorell <andri@signorell.net>
strongly based on code found at https://rosettacode.org/wiki/Vigen%C3%A8re_cipher#R
(credits to the unknown soldier)

Examples

```
key <- "My FavoriteKey452"
(xenc <- Vigenere("Beware the Jabberwock, my son! The jaws that bite, the claws that catch!", key))

Vigenere(xenc, key, decrypt = TRUE)
# note that everything besides the characters in the list will be discarded
```

`VonNeumannTest` *Von Neumann's Successive Difference Test*

Description

A popular statistic to test for independence is the von Neumann ratio.

Usage

```
VonNeumannTest(x, alternative = c("two.sided", "less", "greater"), unbiased = TRUE)
```

Arguments

`x`	a numeric vector containing the observations
`alternative`	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`. You can specify just the initial letter.
`unbiased`	logical. In order for VN to be an unbiased estimate of the true population value, the calculated value is multiplied by $n/(n-1)$. Default is TRUE.

Details

The VN test statistic is in the unbiased case

$$VN = \frac{\sum_{i=1}^{n-1}(x_i - x_{i+1})^2 \cdot n}{\sum_{i=1}^{n}(x_i - \bar{x})^2 \cdot (n-1)}$$

It is known that $(VN - \mu)/\sigma$ is asymptotically standard normal, where $\mu = \frac{2n}{n-1}$ and $\sigma^2 = 4 \cdot n^2 \frac{(n-2)}{(n+1)(n-1)^3}$.

The VN test statistic is in the original (biased) case

$$VN = \frac{\sum_{i=1}^{n-1}(x_i - x_{i+1})^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

The test statistic $(VN - 2)/\sigma$ is asymptotically standard normal, where $\sigma^2 = \frac{4 \cdot (n-2)}{(n+1)(n-1)}$.

Missing values are silently removed.

Value

A list with class "htest" containing the components:

`statistic`	the value of the VN statistic and the normalized statistic test.
`parameter`, `n`	the size of the data, after the remotion of consecutive duplicate values.
`p.value`	the p-value of the test.
`alternative`	a character string describing the alternative hypothesis.
`method`	a character string indicating the test performed.
`data.name`	a character string giving the name of the data.

Author(s)

Andri Signorell <andri@signorell.net>

References

von Neumann, J. (1941) Distribution of the ratio of the mean square successive difference to the variance. *Annals of Mathematical Statistics* **12**, 367-395.

See Also

BartelsRankTest

Examples

```
VonNeumannTest(d.pizza$temperature)
```

---

wdConst *Word VBA Constants*

Description

This is a list with all VBA constants for MS Word 2010, which is useful for writing R functions based on recorded macros in Word. This way the constants need not be replaced by their numeric values and can only be complemented with the list's name, say the VBA-constant `wd10Percent` for example can be replaced by `wdConst$wd10Percent`. A handful constants for Excel are consolidated in `xlConst`.

Usage

```
data(wdConst)
data(xlConst)
```

Format

The format is:
List of 2755
$ wd100Words: num -4
$ wd10Percent: num -6
$ wd10Sentences: num -2
...

Source

Microsoft

`Winsorize` *Winsorize (Replace Extreme Values by Less Extreme Ones)*

Description

Winsorizing a vector means that a predefined quantum of the smallest and/or the largest values are replaced by less extreme values. Thereby the substitute values are the most extreme retained values.

Usage

```
Winsorize(x, val = quantile(x, probs = c(0.05, 0.95), na.rm = FALSE))
```

Arguments

`x`	a numeric vector to be winsorized.
`val`	the low border, all values being lower than this will be replaced by this value. The default is set to the 5%-quantile of x.

Details

The winsorized vector is obtained by

$$g(x) = \begin{cases} -c & \text{for } x \leq c \\ x & \text{for } |x| < c \\ c & \text{for } x \geq c \end{cases}$$

You may also want to consider standardizing (possibly robustly) the data before you perform a winsorization.

Value

A vector of the same length as the original data `x` containing the winsorized data.

Author(s)

Andri Signorell andri@signorell.net

See Also

`winsorize` from the package `robustHD` contains an option to winsorize multivariate data

`scale`, `RobScale`

Examples

```
library(DescTools)

## generate data
set.seed(9128)
x <- round(runif(100) * 100, 1)

(d.frm <- DescTools::Sort(data.frame(
  x,
  default   = Winsorize(x),
  quantile  = Winsorize(x, quantile(x, probs=c(0.1, 0.8), na.rm = FALSE)),
  fixed_val = Winsorize(x, val=c(15, 85)),
  fixed_n   = Winsorize(x, val=c(Small(x, k=3)[3], Large(x, k=3)[1])),
  closest   = Winsorize(x, val=unlist(Closest(x, c(30, 70))))
)))[c(1:10, 90:100), ]

# use Large and Small, if a fix number of values should be winsorized (here k=3)

PlotLinesA(SetNames(d.frm, rownames=NULL), lwd=2, col=Pal("Tibco"),
           main="Winsorized Vector")
z <- 0:10
# twosided (default):
Winsorize(z, val=c(2,8))

# onesided:
# ... replace all values > 8 with 8
Winsorize(z, val=c(min(z), 8))
# ... replace all values < 4 with 4
Winsorize(z, val=c(4, max(z)))
```

`WithOptions` *Execute Function with Temporary Options*

Description

Setting and resetting options is lengthy in command mode. `WithOptions()` allows to evaluate a function with temporary set options.

Usage

```
WithOptions(optlist, expr)
```

Arguments

`optlist`	a list with new option settings.
`expr`	the expression to be evaluated

Value

the function result

Author(s)

Thomas Lumley <t.lumley@auckland.ac.nz>

See Also

options, getOption

Examples

```
# original:
print((1:10)^-1)

# with new options
WithOptions(list(digits=3), print((1:10)^-1))
```

`WoolfTest` *Woolf Test For Homogeneity in 2x2xk Tables*

Description

Test for homogeneity on $2 \times 2 \times k$ tables over strata (i.e., whether the log odds ratios are the same in all strata).

Usage

```
WoolfTest(x)
```

Arguments

`x`	a $2 \times 2 \times k$ table, where the last dimension refers to the strata.

Value

A list of class `"htest"` containing the following components:

`statistic`	the chi-squared test statistic.
`parameter`	degrees of freedom of the approximate chi-squared distribution of the test statistic.
`p.value`	p-value for the test.
`method`	a character string indicating the type of test performed.
`data.name`	a character string giving the name(s) of the data.
`observed`	the observed counts.
`expected`	the expected counts under the null hypothesis.

Note

This function was previously published as `woolf_test()` in the **vcd** package and has been integrated here without logical changes.

Author(s)

David Meyer, Achim Zeileis, Kurt Hornik, Michael Friendly

References

Woolf, B. 1955: On estimating the relation between blood group and disease. *Ann. Human Genet.* (London) **19**, 251-253.

See Also

`mantelhaen.test`, `BreslowDayTest`

Examples

```
migraine <- xtabs(freq ~ .,
            cbind(expand.grid(treatment=c("active","placebo"),
                              response=c("better","same"),
                              gender=c("female","male")),
                  freq=c(16,5,11,20,12,7,16,19))
            )

WoolfTest(migraine)
```

WrdBookmark *Some Functions to Handle MS-Word Bookmarks*

Description

Accessing bookmarks by name is only possible by browsing the bookmark names. `WrdBookmark` returns a handle to a bookmark by taking its name as argument. `WrdInsertBookmark`, `WrdDeleteBookmark` inserts/deletes a bookmark in a Word document. `WrdGotoBookmark` allows to place the cursor on the bookmark and `WrdUpdateBookmark` replaces the content within the range of the bookmark in a Word document with the given text.

Usage

```
WrdBookmark(name, wrd = DescToolsOptions("lastWord"))

WrdInsertBookmark(name, wrd = DescToolsOptions("lastWord"))
WrdDeleteBookmark(name, wrd = DescToolsOptions("lastWord"))

WrdGoto(name, what = wdConst$wdGoToBookmark, wrd = DescToolsOptions("lastWord"))

WrdUpdateBookmark(name, text, what = wdConst$wdGoToBookmark,
                  wrd = DescToolsOptions("lastWord"))
```

Arguments

name	the name of the bookmark.
text	the text of the bookmark.
what	a word constant, defining the type of object to be used to place the cursor.
wrd	the pointer to a word instance. Can be a new one, created by GetNewWrd() or an existing one, created by GetCurrWrd(). Default is the last created pointer stored in DescToolsOptions("lastWord").

Details

Bookmarks are useful to build structured documents, which can be updated later.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

WrdFont, WrdPlot, GetNewWrd, GetCurrWrd

Examples

```
## Not run:  # we can't get this through the CRAN test - run it with copy/paste to console
wrd <- GetNewWrd()
WrdText("a)\n\n\nb)", fontname=WrdGetFont()$name, fontsize=WrdGetFont()$size)
WrdInsertBookmark("chap_b")
WrdText("\n\n\nc)\n\n\n", fontname=WrdGetFont()$name, fontsize=WrdGetFont()$size)

WrdGoto("chap_b")
WrdUpdateBookmark("chap_b", "Goto chapter B and set text")

WrdInsertBookmark("mybookmark")
ToWrd("A longer text\n\n\n")

# Now returning the bookmark
bm <- WrdBookmark("mybookmark")

# get the automatically created name of the bookmark
bm$name()

## End(Not run)
```

WrdCaption *Insert Caption to Word*

Description

Insert a caption in a given level to a Word document. The caption is inserted at the current cursor position.

Usage

```
WrdCaption(x, index = 1, wrd = DescToolsOptions("lastWord"))
```

Arguments

x	the text of the caption.
index	integer from 1 to 9, defining the number of the heading style.
wrd	the pointer to a word instance. Can be a new one, created by `GetNewWrd()` or an existing one, created by `GetCurrWrd()`. Default is the last created pointer stored in `DescToolsOptions("lastWord")`.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

ToWrd, WrdPlot, GetNewWrd, GetCurrWrd

Examples

```
## Not run:  # Windows-specific example
wrd <- GetNewWrd()

# insert a title in level 1
WrdCaption("My First Caption level 1", index=1, wrd=wrd)

# works as well for several levels
sapply(1:5, function(i)
  WrdCaption(gettextf("My First Caption level %s",i), index=i, wrd=wrd)
)

## End(Not run)
```

WrdCellRange *Return the Cell Range Of a Word Table*

Description

Return a handle of a cell range of a word table. This is useful for formating the cell range.

Usage

```
WrdCellRange(wtab, from, to)
```

Arguments

wtab	a handle to the word table as returned i.g. by WrdTable
from	a vector containing row- and column number of the left/upper cell of the cell range.
to	a vector containing row- and column number of the right/lower cell of the cell range.

Details

Cell range selecting might be complicated. This function makes it easy.

Value

a handle to the range.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

WrdTable

Examples

```
## Not run:

# Windows-specific example
wrd <- GetNewWrd()
WrdTable(nrow=3, ncol=3, wrd=wrd)
crng <- WrdCellRange(from=c(1,2), to=c(2,3))
crng$Select()

## End(Not run)
```

WrdFont *Get or Set the Font in Word*

Description

WrdFont can be used to get and set the font in Word for the text to be inserted. WrdFont returns the font at the current cursor position.

Usage

```
WrdFont(wrd = DescToolsOptions("lastWord"))
WrdFont(wrd) <- value
```

Arguments

value	the font to be used to the output. This should be defined as a list containing fontname, fontsize, bold and italic flags: list(name="Arial", size=10, bold=FALSE, italic=TRUE, color=wdConst$wdColorBlack).
wrd	the pointer to a word instance. Can be a new one, created by GetNewWrd() or an existing one, created by GetCurrWrd(). Default is the last created pointer stored in DescToolsOptions("lastWord").

Details

The font color can be defined by a Word constant beginning with wdConst$wdColor. The defined colors can be listed with grep("wdColor", names(wdConst), val=TRUE).

Value

a list of the attributes of the font in the current cursor position:

name	the fontname
size	the fontsize
bold	bold
italic	italic
color	the fontcolor

Author(s)

Andri Signorell <andri@signorell.net>

See Also

ToWrd, WrdPlot, GetNewWrd, GetCurrWrd

Examples

```
## Not run:  # Windows-specific example

wrd <- GetNewWrd()

for(i in seq(10, 24, 2))
  ToWrd(gettextf("This is Arial size %s \n", i), font=list(name="Arial", size=i))

for(i in seq(10, 24, 2))
  ToWrd(gettextf("This is Times size %s \n", i), font=list(name="Times", size=i))

## End(Not run)
```

WrdFormatCells *Format Cells Of a Word Table*

Description

Format cells of a Word table.

Usage

```
WrdFormatCells(wtab, rstart, rend, col = NULL, bg = NULL,
               font = NULL, border = NULL, align = NULL)
```

Arguments

wtab	a handle to the word table as returned i.g. by WrdTable
rstart	the left/upper cell of the cell range
rend	the right/lower cell of the cell range
col	the foreground colour
bg	the background colour
font	the font to be used to the output. This should be defined as a list containing fontname, fontsize, bold and italic flags: list(name="Arial", size=10, bold=FALSE, italic=TRUE, color=wdConst$wdColorBlack).
border	the border of the cell range, defined as a list containing arguments for border, linestyle, linewidth and color. border is a vector containing the parts of the border defined by the Word constants wdConst$wdBorder..., being $wdBorderBottom, $wdBorderLeft, $wdBorderTop, $wdBorderRight, $wdBorderHorizontal, $wdBorderVertical, $wdBorderDiagonalUp, $wdBorderDiagonalDown. linestyle, linewidth and color will be recycled to the required dimension.
align	a character out of "l", "c", "r" setting the horizontal alignment of the cell range.

Details

Cell range selecting might be complicated. This function makes it easy.

Value

a handle to the range.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

WrdTable

Examples

```
## Not run:    # Windows-specific example

m <- matrix(rnorm(12)*100, nrow=4,
            dimnames=list(LETTERS[1:4], c("Variable","Value","Remark")))

wrd <- GetNewWrd()
wt <- ToWrd(m)

WrdFormatCells(wt, rstart=c(3,1), rend=c(4,3),
               bg=wdConst$wdColorGold, font=list(name="Arial Narrow", bold=TRUE),
               align="c", border=list(color=wdConst$wdColorTeal,
                                      linewidth=wdConst$wdLineWidth300pt))


## End(Not run)
```

WrdMergeCells *Merges Cells Of a Defined Word Table Range*

Description

Merges a cell range of a word table.

Usage

```
WrdMergeCells(wtab, rstart, rend)
```

Arguments

wtab	a handle to the word table as returned i.g. by WrdTable
rstart	the left/upper cell of the cell range.
rend	the right/lower cell of the cell range.

Value

nothing

Author(s)

Andri Signorell <andri@signorell.net>

See Also

WrdTable, WrdCellRange

Examples

```
## Not run:

# Windows-specific example
wrd <- GetNewWrd()
wtab <- WrdTable(nrow=3, ncol=3, wrd=wrd)
WrdMergeCells(wtab, rstart=c(1,2), rend=c(2,3))

## End(Not run)
```

WrdPageBreak *Insert a Page Break*

Description

Insert a page break in a MS-Word (R) document at the position of the cursor.

Usage

```
WrdPageBreak(wrd = DescToolsOptions("lastWord"))
```

Arguments

wrd	the pointer to a word instance. Can be a new one, created by GetNewWrd() or an existing one, created by GetCurrWrd(). Default is the last created pointer stored in DescToolsOptions("lastWord").

Author(s)

Andri Signorell <andri@signorell.net>

See Also

WrdFont, WrdPlot, GetNewWrd, GetCurrWrd

Examples

```
## Not run:  # Windows-specific example
wrd <- GetNewWrd()
WrdText("This is text on page 1.\n\n")
WrdPageBreak()
WrdText("This is text on another page.\n\n")

## End(Not run)
```

WrdParagraphFormat *Get or Set the Paragraph Format in Word*

Description

WrdParagraphFormat can be used to get and set the font in Word for the text to be inserted.

Usage

```
WrdParagraphFormat(wrd = DescToolsOptions("lastWord"))
WrdParagraphFormat(wrd) <- value
```

Arguments

value	a list defining the paragraph format. This can contain any combination of: LeftIndent, RightIndent, SpaceBefore, SpaceBeforeAuto, SpaceAfter, SpaceAfterAuto, LineSpacingRule, Alignment, WidowControl, KeepWithNext, KeepTogether, PageBreakBefore, NoLineNumber, Hyphenation, FirstLineIndent, OutlineLevel, CharacterUnitLeftIndent, CharacterUnitRightIndent, CharacterUnitFirstLineIndent, LineUnitBefore, LineUnitAfter and/or MirrorIndents. The possible values of the arguments are found in the Word constants with the respective name. The alignment for example can be set to wdAlignParagraphLeft, wdAlignParagraphRight, wdAlignParagraphCenter and so on. Left alignment with indentation would be set as: list(Alignment=wdConst$wdAlignParagraphLeft, LeftIndent=42.55).
wrd	the pointer to a word instance. Can be a new one, created by GetNewWrd() or an existing one, created by GetCurrWrd(). Default is the last created pointer stored in DescToolsOptions("lastWord").

Value

an object with the class paragraph, basically a list with the attributes of the paragraph in the current cursor position:

LeftIndent	left indentation in (in points) for the specified paragraphs.
RightIndent	right indent (in points) for the specified paragraphs.
SpaceBefore	spacing (in points) before the specified paragraphs.

SpaceBeforeAuto
: TRUE if Microsoft Word automatically sets the amount of spacing before the specified paragraphs.

SpaceAfter
: amount of spacing (in points) after the specified paragraph or text column.

SpaceAfterAuto
: TRUE if Microsoft Word automatically sets the amount of spacing after the specified paragraphs.

LineSpacingRule
: line spacing for the specified paragraph formatting. Use wdLineSpaceSingle, wdLineSpace1pt5, or wdLineSpaceDouble to set the line spacing to one of these values. To set the line spacing to an exact number of points or to a multiple number of lines, you must also set the LineSpacing property.

Alignment
: WdParagraphAlignment constant that represents the alignment for the specified paragraphs.

WidowControl
: TRUE if the first and last lines in the specified paragraph remain on the same page as the rest of the paragraph when Word repaginates the document. Can be TRUE, FALSE or wdUndefined.

KeepWithNext
: TRUE if the specified paragraph remains on the same page as the paragraph that follows it when Microsoft Word repaginates the document. Read/write Long.

KeepTogether
: TRUE if all lines in the specified paragraphs remain on the same page when Microsoft Word repaginates the document.

PageBreakBefore
: TRUE if a page break is forced before the specified paragraphs. Can be TRUE, FALSE, or wdUndefined.

NoLineNumber
: TRUE if line numbers are repressed for the specified paragraphs. Can be TRUE, FALSE, or wdUndefined.

Hyphenation
: TRUE if the specified paragraphs are included in automatic hyphenation. FALSE if the specified paragraphs are to be excluded from automatic hyphenation.

FirstLineIndent
: value (in points) for a first line or hanging indent. Use a positive value to set a first-line indent, and use a negative value to set a hanging indent.

OutlineLevel
: outline level for the specified paragraphs.

CharacterUnitLeftIndent
: left indent value (in characters) for the specified paragraphs.

CharacterUnitRightIndent
: right indent value (in characters) for the specified paragraphs.

LineUnitBefore
: amount of spacing (in gridlines) before the specified paragraphs.

LineUnitAfter
: amount of spacing (in gridlines) after the specified paragraphs.

MirrorIndents
: Long that represents whether left and right indents are the same width. Can be TRUE, FALSE, or wdUndefined.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

ToWrd, WrdPlot, GetNewWrd, GetCurrWrd

Examples

```
## Not run:
# Windows-specific example
wrd <- GetNewWrd()  # get the handle to a new word instance

WrdParagraphFormat(wrd=wrd) <- list(Alignment=wdConst$wdAlignParagraphLeft,
                                    LeftIndent=42.55)

ToWrd("Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy
eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.
At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd
gubergren, no sea takimata sanctus est.\n", wrd=wrd)

# reset
WrdParagraphFormat(wrd=wrd) <- list(LeftIndent=0)

## End(Not run)
```

WrdPlot *Insert Active Plot to Word*

Description

This function inserts the plot on the active plot device to Word. The image is transferred by saving the picture to a file in R and inserting the file in Word. The format of the plot can be selected, as well as crop options and the size factor for inserting.

Usage

```
WrdPlot(type = "png", append.cr = TRUE, crop = c(0, 0, 0, 0), main = NULL,
        picscale = 100, height = NA, width = NA, res = 300,
        dfact = 1.6, wrd = DescToolsOptions("lastWord"))
```

Arguments

type	the format for the picture file, default is "png".
append.cr	should a carriage return be appended? Default is TRUE.
crop	crop options for the picture, defined by a 4-elements-vector. The first element is the bottom side, the second the left and so on.
main	a caption for the plot. This will be inserted by InserCaption in Word. Default is NULL, which will insert nothing.
picscale	scale factor of the picture in percent, default ist 100.
height	height in cm, this overrides the picscale if both are given.

width	width in cm, this overrides the picscale if both are given.
res	resolution for the png file, defaults to 300.
dfact	the size factor for the graphic.
wrd	the pointer to a word instance. Can be a new one, created by GetNewWrd() or an existing one, created by GetCurrWrd(). Default is the last created pointer stored in DescToolsOptions("lastWord").

Value

Returns a pointer to the inserted picture.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

ToWrd, WrdCaption, GetNewWrd

Examples

```
## Not run:  # Windows-specific example
# let's have some graphics
plot(1,type="n", axes=FALSE, xlab="", ylab="", xlim=c(0,1), ylim=c(0,1), asp=1)
rect(0,0,1,1,col="black")
segments(x0=0.5, y0=seq(0.632,0.67, length.out=100),
  y1=seq(0.5,0.6, length.out=100), x1=1, col=rev(rainbow(100)))
polygon(x=c(0.35,0.65,0.5), y=c(0.5,0.5,0.75), border="white",
  col="black", lwd=2)
segments(x0=0,y0=0.52, x1=0.43, y1=0.64, col="white", lwd=2)
x1 <- seq(0.549,0.578, length.out=50)
segments(x0=0.43, y0=0.64, x1=x1, y1=-tan(pi/3)* x1 + tan(pi/3) * 0.93,
  col=rgb(1,1,1,0.35))


# get a handle to a new word instance
wrd <- GetNewWrd()
# insert plot with a specified height
WrdPlot(wrd=wrd, height=5)
ToWrd("Remember?\n", fontname="Arial", fontsize=14, bold=TRUE, wrd=wrd)
# crop the picture
WrdPlot(wrd=wrd, height=5, crop=c(9,9,0,0))


wpic <- WrdPlot(wrd=wrd, height=5, crop=c(9,9,0,0))
wpic

## End(Not run)
```

WrdSaveAs *Open and Save Word Documents*

Description

Open and save MS-Word documents.

Usage

```
WrdOpenFile(fn, wrd = DescToolsOptions("lastWord"))
WrdSaveAs(fn, fileformat = "docx", wrd = DescToolsOptions("lastWord"))
```

Arguments

fn	filename and -path for the document.
fileformat	file format, one out of "doc", "htm", "pdf".
wrd	the pointer to a word instance. Can be a new one, created by GetNewWrd() or an existing one, created by GetCurrWrd(). Default is the last created pointer stored in DescToolsOptions("lastWord").

Value

nothing returned

Author(s)

Andri Signorell <andri@signorell.net>

See Also

GetNewWrd()

Examples

```
## Not run:
#   Windows-specific example
wrd <- GetNewWrd()
WrdCaption("A Report")
WrdSaveAs(fn="report", fileformat="htm")

## End(Not run)
```

WrdStyle *Get or Set the Style in Word*

Description

WrdStyle can be used to get and set the style in Word for the text to be inserted. WrdStyle returns the style at the current cursor position.

Usage

```
WrdStyle(wrd = DescToolsOptions("lastWord"))
WrdStyle(wrd) <- value
```

Arguments

value	the name of the style to be used to the output. This should be defined an existing name.
wrd	the pointer to a word instance. Can be a new one, created by GetNewWrd() or an existing one, created by GetCurrWrd(). Default is the last created pointer stored in DescToolsOptions("lastWord").

Value

character, name of the style

Author(s)

Andri Signorell <andri@signorell.net>

See Also

ToWrd, WrdPlot, GetNewWrd, GetCurrWrd

Examples

```
## Not run:  # Windows-specific example

wrd <- GetNewWrd()
# the current stlye
WrdStyle(wrd)

## End(Not run)
```

WrdTable *Insert a Table in a Word Document*

Description

Create a table with a specified number of rows and columns in a Word document at the current position of the cursor.

Usage

```
WrdTable(nrow = 1, ncol = 1, heights = NULL, widths = NULL, main = NULL,
         wrd = DescToolsOptions("lastWord"))
```

Arguments

nrow	number of rows.
ncol	number of columns.
heights	a vector of the row heights (in [cm]). If set to NULL (which is the default) the Word defaults will be used. The values will be recyled, if necessary.
widths	a vector of the column widths (in [cm]). If set to NULL (which is the default) the Word defaults will be used. The values will be recyled, if necessary.
main	a caption for the plot. This will be inserted by InserCaption in Word. Default is NULL, which will insert nothing.
wrd	the pointer to a word instance. Can be a new one, created by GetNewWrd() or an existing one, created by GetCurrWrd(). Default is the last created pointer stored in DescToolsOptions("lastWord").

Value

A pointer to the inserted table.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

GetNewWrd, ToWrd

Examples

```
## Not run:  # Windows-specific example
wrd <- GetNewWrd()
WrdTable(nrow=3, ncol=3, wrd=wrd)

## End(Not run)
```

WrdTableBorders *Draw Borders to a Word Table*

Description

Drawing borders in a Word table is quite tedious. This function allows to select any range and draw border lines around it.

Usage

```
WrdTableBorders(wtab, from = NULL, to = NULL, border = NULL,
                lty = wdConst$wdLineStyleSingle, col = wdConst$wdColorBlack,
                lwd = wdConst$wdLineWidth050pt)
```

Arguments

wtab	a pointer to a Word table as returned by WrdTable or TOne.
from	integer, a vector with two elements specifying the left upper bound of the cellrange.
to	integer, a vector with two elements specifying the right bottom of the cellrange.
border	a Word constant (wdConst$wdBorder...) defining the side of the border.
lty	a Word constant (wdConst$wdLineStyle...) defining the line type.
col	a Word constant (wdConst$wdColor...) defining the color of the border. See examples for converting R colors to Word colors.
lwd	a Word constant (wdConst$wdLineWidth...pt) defining the line width.

Value

nothing

Author(s)

Andri Signorell <andri@signorell.net>

See Also

WrdTable

Examples

```
## Not run:

# create table
tab <- table(op=d.pizza$operator, area=d.pizza$area)

# send it to Word
wrd <- GetNewWrd()
```

```
wtab <- ToWrd(tab, wrd=wrd, tablestyle = NA)

# draw borders
WrdTableBorders(wtab, from=c(2,2), to=c(3,3), border=wdConst$wdBorderBottom, wrd=wrd)
WrdTableBorders(wtab, from=c(2,2), to=c(3,3), border=wdConst$wdBorderDiagonalUp, wrd=wrd)

# demonstrate linewidth and color
wtab <- ToWrd(tab, wrd=wrd, tablestyle = NA)
WrdTableBorders(wtab, col=RgbToLong(ColToRgb("olivedrab")),
                lwd=wdConst$wdLineWidth150pt, wrd=wrd)

WrdTableBorders(wtab, border=wdConst$wdBorderBottom,
                col=RgbToLong(ColToRgb("dodgerblue")),
                lwd=wdConst$wdLineWidth300pt, wrd=wrd)

# use an R color in Word
RgbToLong(ColToRgb("olivedrab"))

# find a similar R-color for a Word color
ColToRgb(RgbToCol(LongToRgb(wdConst$wdColorAqua)))

## End(Not run)
```

`WrdTableHeading` *Insert Headings for a Table in Word*

Description

Inserting headings in a table can be hard, when column headings should span several columns. This function helps to easily insert headings and merge cells.

Usage

```
WrdTableHeading(wtab, text, bold = TRUE,
                alignment = wdConst$wdAlignParagraphCenter,
                merge_cols = NULL, wrd = DescToolsOptions("lastWord"))
```

Arguments

`wtab`	the handle to a table in a word document
`text`	the text for the headings
`bold`	logical, for setting bold column headings, will be recycled. Default is `TRUE`.
`alignment`	the alignment in the column headings, must be one out of the Word constant list `wdConst$wdAlignParagraph`.
`merge_cols`	a vector consisting of entries to merge cells in the form `"<first cell>:<last cell>"`, example `merge_cols=c("2:4", "7:8")` would merge the column headings 2:4 to one cell and 7:8 as well.

wrd	the pointer to a word instance. Can be a new one, created by GetNewWrd() or an existing one, created by GetCurrWrd(). Default is the last created pointer stored in DescToolsOptions("lastWord").

Value

Nothing returned.

Author(s)

Andri Signorell <andri@signorell.net

See Also

WrdTable

Examples

```
## Not run:  # Windows-specific example
wrd <- GetNewWrd()
wtab <- WrdTable(nrow=3, ncol=5, wrd=wrd)

# insert headings and merge 1:2 and 4:5, such as there are
# only 3 headings
WrdTableHeading(wtab, text=c("First","Second",
                             "Third"),
                 alignment=c(wdConst$wdAlignParagraphLeft,
                             rep(wdConst$wdAlignParagraphCenter, 2)),
                 merge_cols = c("1:2", "4:5"))

## End(Not run)
```

XLDateToPOSIXct *Convert Excel Dates to POSIXct*

Description

As I repeatedly forgot how to convert Excel dates to POSIX here's the specific function.

Usage

```
XLDateToPOSIXct(x, tz = "GMT", xl1904 = FALSE)
```

Arguments

x	the integer vector to be converted.
tz	a time zone specification to be used for the conversion, if one is required. See as.POSIXct.
xl1904	logical, defining if the unspeakable 1904-system should be used. Default is FALSE.

Details

XLGetRange will return dates as integer values, because XL stores them as integers. An Excel date can be converted with the (unusual) origin of `as.Date(myDate, origin="1899-12-30")`, which is implemented here.

Microsoft Excel supports two different date systems, the 1900 date system and the 1904 date system. In the 1900 date system, the first day that is supported is January 1, 1900. A date is converted into a serial number that represents the number of elapsed days since January 1, 1900. In the 1904 date system, the first day that is supported is January 1, 1904. By default, Microsoft Excel for the Macintosh uses the 1904 date system, Excel for Windows the 1900 system. See also: https://support.microsoft.com/en-us/kb/214330.

Value

return an object of the class POSIXct. Date-times known to be invalid will be returned as NA.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

as.POSIXct

Examples

```
XLDateToPOSIXct(41025)
XLDateToPOSIXct(c(41025.23, 41035.52))
```

XLGetRange *Import Data Directly From Excel*

Description

The package RDCOMClient is used to open an Excel workbook and return the content (value) of one (or several) given range(s) in a specified sheet. This is helpful, whenever pathologically scattered data on an Excel sheet, which can't simply be saved as CSV-file, has to be imported in R.

`XLGetWorkbook()` does the same for all the sheets in an Excel workbook.

Usage

```
XLGetRange(file = NULL, sheet = NULL, range = NULL, as.data.frame = TRUE,
           header = FALSE, stringsAsFactors = FALSE, echo = FALSE,
           na.strings = NULL, skip = 0)

XLGetWorkbook(file, compactareas = TRUE)

XLCurrReg(cell)
XLNamedReg(x)
```

Arguments

file	the fully specified path and filename of the workbook. If it is left as NULL, the function will look for a running Excel-Application and use its current sheet. The parameter sheet will be ignored in this case.
sheet	the name of the sheet containing the range(s) of interest.
range	a scalar or a vector with the address(es) of the range(s) to be returned (characters). Use "A1"-address mode to specify the ranges, for example "A1:F10". If set to NULL (which is the default), the function will look for a selection that contains more than one cell. If found, the function will use this selection. If there is no selection then the current region of the selected cell will be used. Use XLCurrReg() if the current region of a cell, which is currently not selected, should be used. Range names can be provided with XLNamedReg("name").
as.data.frame	logical. Determines if the cellranges should be coerced into data.frames. Defaults to TRUE, as this is probably the common use of this function.
header	a logical value indicating whether the range contains the names of the variables as its first line. Default is FALSE. header is ignored if as.data.frame has been set to FALSE.
stringsAsFactors	logical. Should character columns be coerced to factors? The default is FALSE, which will return character vectors.
echo	logical. If set to TRUE, the function will print the full command used, such that it can be copied into the R-script for future use.
na.strings	a character vector of strings which are to be interpreted as NA values. Blank fields are always considered to be missing values. Default is NULL, meaning none.
compactareas	logical, defining if areas should be returned by XLGetWorkbook as list or as matrix (latter is default).
cell	range of the left uppe cell, when current region should be used.
x	the name or the index of the XL-name to be used.
skip	the number of lines of the data file to skip before beginning to read data.

Details

The result consists of a list of lists, if as.data.frame is set to FALSE. Be then prepared to encounter NULL values. Those will prevent from easily being able to coerce the square data structure to a data.frame.

The following code will replace the NULL values by NA and coerce the data to a data.frame.

```
# get the range D1:J69 from an excel file
xlrng <- XLGetRange(file="myfile.xlsx", sheet="Tabelle1",
                    range="D1:J69", as.data.frame=FALSE)

# replace NULL values by NA
xlrng[unlist(lapply(xlrng, is.null))] <- NA
```

```
  # coerce the square data structure to a data.frame
  d.lka <- data.frame(lapply(data.frame(xlrng), unlist))
```

This of course can be avoided by setting `as.data.frame = TRUE`.

The function will return dates as integers, because MS-Excel stores them internally as integers. Such a date can subsequently be converted with the (unusual) origin of `as.Date(myDate, origin="1899-12-30")`. See also `XLDateToPOSIXct`, which does the job. The conversion can directly be performed by `XLGetRange()` if `datecols` is used and contains the date columns in the sheet data.

Value

If `as.data.frame` is set to `TRUE`, a single data.frame or a list of data.frames will be returned. If set to `FALSE` a list of the cell values in the specified Excel range, resp. a list of lists will be returned.

`XLGetWorkbook()` returns a list of lists of the values in the given workbook.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`GetNewXL`, `GetCurrXL`, `XLView`

Examples

```
## Not run:  # Windows-specific example

XLGetRange(file="C:/My Documents/data.xls",
           sheet="Sheet1",
           range=c("A2:B5","M6:X23","C4:D40"))


# if the current region has to be read (incl. a header), place the cursor in the interesting region
# and run:
d.set <- XLGetRange(header=TRUE)

# Get XL nameslist
nm <- xl$ActiveWorkbook()$names()

lst <- list()
for(i in 1:nm$count())
  lst[[i]] <- c(name=nm[[i]]$name(),
                address=nm[[i]]$refersToRange()$Address())

# the defined names
as.data.frame(do.call(rbind, lst), stringsAsFactors = FALSE)

## End(Not run)
```

XLSaveAs *Save Excel File*

Description

Save the current workbook under the given name and format.

Usage

```
XLSaveAs(fn, file_format = xlConst$XlFileFormat$xlWorkbookNormal,
         xl = DescToolsOptions("lastXL"))
```

Arguments

fn	the filename
file_format	the file format using the xl constant.
xl	the pointer to a MS-Excel instance. An new instance can be created with GetNewXL(), returning the appropriate handle. A handle to an already running instance is returned by GetCurrXL(). Default is the last created pointer stored in DescToolsOptions("lastXL").

Value

returns TRUE if the save operation has been successful

Author(s)

Andri Signorell <andri@signorell.net>

See Also

XLView

Examples

```
## Not run: # Windows-specific example
XLView(d.diamonds)
XLSaveAs("Diamonds")
xl$quit()

## End(Not run)
```

XLView *Use MS-Excel as Viewer for a Data.Frame*

Description

XLView can be used to view and edit a data.frame directly in MS-Excel, resp. to create a new data.frame in MS-Excel.

Usage

```
XLView(x, col.names = TRUE, row.names = FALSE, na = "",
       preserveStrings = FALSE, sep = ";")

ToXL(x, at, ..., xl=DescToolsOptions("lastXL"))
## S3 method for class 'data.frame'
ToXL(x, at, ..., xl=DescToolsOptions("lastXL"))
## S3 method for class 'matrix'
ToXL(x, at, ..., xl=DescToolsOptions("lastXL"))
## Default S3 method:
ToXL(x, at, byrow = FALSE, ..., xl=DescToolsOptions("lastXL"))

XLKill()
```

Arguments

x	is a data.frame to be transferred to MS-Excel. If data is missing a new file will be created.
row.names	either a logical value indicating whether the row names of x are to be written along with x, or a character vector of row names to be written.
col.names	either a logical value indicating whether the column names of x are to be written along with x, or a character vector of column names to be written. See the section on 'CSV files' write.table for the meaning of col.names = NA.
na	the string to use for missing values in the data.
preserveStrings	logical, will preserve strings from being converted to numerics when imported in MS-Excel. See details. Default is FALSE.
sep	the field separator string used for export of the object. Values within each row of x are separated by this string.
at	can be a range adress as character (e.g. "A1"), a vector of 2 integers (e.g c(1,1)) or a cell object as it is returned by xl$Cells(1,1), denominating the left upper cell, where the data.frame will be placed in the MS-Excel sheet.
byrow	logical, defines if the vector should be inserted by row or by column (default).
xl	the pointer to a MS-Excel instance. An new instance can be created with GetNewXL(), returning the appropriate handle. A handle to an already running instance is returned by GetCurrXL(). Default is the last created pointer stored in DescToolsOptions("lastXL").
...	further arguments are not used.

Details

The data.frame will be exported in CSV format and then imported in MS-Excel. When importing data, MS-Excel will potentially change characters to numeric values. If this seems undesirable (maybe we're loosing leading zeros) then you should enclose the text in quotes and preset a =. x <- `gettextf('="%s"', x)` would do the trick.

Take care: Changes to the data made in MS-Excel will NOT automatically be updated in the original data.frame. The user will have to read the csv-file into R again. See examples how to get this done.

`ToXL()` is used to export data frames or vectors directly to MS-Excel, without export the data to a csv-file and import it on the XL side. So it it possible to export several data.frames into one Workbook and edit the tables after ones needs.

`XLKill` will kill a running XL instance (which might be invisible). Background is the fact, that the simple XL$quit() command would not terminate a running XL task, but only set it invisible (observe the TaskManager). This ghost version may sometimes confuse XLView and hinder to create a new instance. In such cases you have to do the garbage collection...

Value

the name/path of the temporary file edited in MS-Excel.

Note

The function works only in Windows and requires **RDCOMClient** to be installed (see: Additional_repositories in DESCRIPTION of the package).

Author(s)

Andri Signorell <andri@signorell.net>, `ToXL()` is based on code of Duncan Temple Lang <duncan@r-project.org>

See Also

`GetNewXL`, `XLGetRange`, `XLGetWorkbook`

Examples

```
## Not run:
# Windows-specific example
XLView(d.diamonds)

# edit an existing data.frame in MS-Excel, make changes and save there, return the filename
fn <- XLView(d.diamonds)
# read the changed file and store in new data.frame
d.frm <- read.table(fn, header=TRUE, quote="", sep=";")

# Create a new file, edit it in MS-Excel...
fn <- XLView()
```

```
# ... and read it into a data.frame when in R again
d.set <- read.table(fn, header=TRUE, quote="", sep=";")

# Export a ftable object, quite elegant...
XLView(format(ftable(Titanic), quote=FALSE), row.names = FALSE, col.names = FALSE)


# Export a data.frame directly to XL, combined with subsequent formatting

xl <- GetNewXL()
owb <- xl[["Workbooks"]]$Add()
sheet <- xl$Sheets()$Add()
sheet[["name"]] <- "pizza"

ToXL(d.pizza[1:10, 1:10], xl$Cells(1,1))

obj <- xl$Cells()$CurrentRegion()
obj[["VerticalAlignment"]] <- xlConst$xlTop

row <- xl$Cells()$CurrentRegion()$rows(1)
# does not work:   row$font()[["bold"]] <- TRUE
# works:
obj <- row$font()
obj[["bold"]] <- TRUE

obj <- row$borders(xlConst$xlEdgeBottom)
obj[["linestyle"]] <- xlConst$xlContinuous

cols <- xl$Cells()$CurrentRegion()$columns(1)
cols[["HorizontalAlignment"]] <- xlConst$xlLeft

xl$Cells()$CurrentRegion()[["EntireColumn"]]$AutoFit()
cols <- xl$Cells()$CurrentRegion()$columns(4)
cols[["WrapText"]] <- TRUE
cols[["ColumnWidth"]] <- 80
xl$Cells()$CurrentRegion()[["EntireRow"]]$AutoFit()

sheet <- xl$Sheets()$Add()
sheet[["name"]] <- "whisky"
ToXL(d.whisky[1:10, 1:10], xl$Cells(1,1))
## End(Not run)
```

YuenTTest *Yuen t-Test For Trimmed Means*

Description

Performs one and two sample Yuen t-tests for trimmed means on vectors of data.

Usage

```
YuenTTest(x, ...)

## Default S3 method:
YuenTTest(x, y = NULL, alternative = c("two.sided", "less", "greater"),
          mu = 0, paired = FALSE, conf.level = 0.95, trim = 0.2, ... )

## S3 method for class 'formula'
YuenTTest(formula, data, subset, na.action, ...)
```

Arguments

x	numeric vector of data values. Non-finite (e.g. infinite or missing) values will be omitted.
y	an optional numeric vector of data values: as with x non-finite values will be omitted.
alternative	is a character string, one of "greater", "less", or "two.sided", or the initial letter of each, indicating the specification of the alternative hypothesis. For one-sample tests, alternative refers to the true median of the parent population in relation to the hypothesized value of the mean.
paired	a logical indicating whether you want a paired z-test.
mu	a number specifying the hypothesized mean of the population.
conf.level	confidence level for the interval computation.
trim	the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
formula	a formula of the form lhs ~ rhs where lhs gives the data values and rhs the corresponding groups.
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
...	further arguments to be passed to or from methods.

Value

An object of class htest containing the following components:

statistic	the value of the t-statistic.
parameter	the degrees of freedom for the t-statistic and the trim percentage used.
p.value	the p-value for the test.

conf.int	a confidence interval for the trimmed mean appropriate to the specified alternative hypothesis.
estimate	the estimated trimmed mean or difference in trimmed means depending on whether it was a one-sample test or a two-sample test.
null.value	the specified hypothesized value of the trimmed mean or trimmed mean difference depending on whether it was a one-sample test or a two-sample test.
alternative	a character string describing the alternative hypothesis.
method	a character string indicating what type of test was performed.
data.name	a character string giving the name(s) of the data.

Author(s)

Andri Signorell <andri@signorell.net>, based on R-Core code of t.test

References

Wilcox, R. R. (2005) Introduction to robust estimation and hypothesis testing. *Academic Press*.
Yuen, K. K. (1974) The two-sample trimmed t for unequal population variances. *Biometrika*, 61, 165-170.

See Also

t.test, print.htest

Examples

```
x <- rnorm(25, 100, 5)
YuenTTest(x, mu=99)

# the classic interface
with(sleep, YuenTTest(extra[group == 1], extra[group == 2]))

# the formula interface
YuenTTest(extra ~ group, data = sleep)


# Stahel (2002), pp. 186, 196
d.tyres <- data.frame(A=c(44.5,55,52.5,50.2,45.3,46.1,52.1,50.5,50.6,49.2),
                      B=c(44.9,54.8,55.6,55.2,55.6,47.7,53,49.1,52.3,50.7))
with(d.tyres, YuenTTest(A, B, paired=TRUE))


d.oxen <- data.frame(ext=c(2.7,2.7,1.1,3.0,1.9,3.0,3.8,3.8,0.3,1.9,1.9),
                     int=c(6.5,5.4,8.1,3.5,0.5,3.8,6.8,4.9,9.5,6.2,4.1))
with(d.oxen, YuenTTest(int, ext, paired=FALSE))
```

ZeroIfNA *Replace NAs by 0*

Description

Replace NAs in a numeric vector x with 0. This function has the same logic as the zeroifnull function in SQL. `NAIfZero()` does replace zeros with `NA`. `BlankIfNA()` and `NAIfBlank()` do the same, but for character vectors.

Usage

```
ZeroIfNA(x)
NAIfZero(x)
NAIf(x, what)

BlankIfNA(x, blank="")
NAIfBlank(x)

Impute(x, FUN = function(x) median(x, na.rm = TRUE))
```

Arguments

`x`	the vector x, whose NAs should be overwritten with 0s.
`blank`	a character to be used for "blank". Default is an empty string ("").
`what`	a vector of elements to be set to `NA` in `x`.
`FUN`	the name of a function to be used as imputation. Can as well be a self defined function or a constant value. Default is `median`.

Value

the edited vector x

Author(s)

Andri Signorell <andri@signorell.net>

See Also

`replace`

Examples

```
z <- c(8, NA, 9, NA, 3)

ZeroIfNA(z)
# [1] 8 0 9 0 3
```

```
# set 8 and 9 to NA
NAIf(ZeroIfNA(z), what=c(8, 9))


Impute(z)
# [1] 8 8 9 8 3


z <- c("a", NA, "", NA, "k")

BlankIfNA(z)
# [1] "a" "" "" "" "k"
```

`Zodiac` *Calculate the Zodiac of a Date*

Description

Calculate the sign of zodiac of a date.

Usage

```
Zodiac(x, lang = c("engl", "deu"), stringsAsFactors = TRUE)
```

Arguments

`x`	the date to be transformed.
`lang`	the language of the zodiac names, can be english (default) or german (`"deu"`).
`stringsAsFactors`	logical. If set to `TRUE` (default) the result will consist of a factor with zodiac signs as levels.

Details

The really relevant things can sometimes hardly be found. You just discovered such a function... ;-)

The following rule to determine zodiac symbols is implemented:

```
Dec. 22  - Jan. 19  : Capricorn
Jan. 20  - Feb. 17  : Aquarius
Feb. 18  - Mar. 19  : Pisces
March 20 - April 19 : Aries
April 20 - May 19   : Taurus
May 20   - June 20  : Gemini
June 21  - July 21  : Cancer
July 22  - Aug. 22  : Leo
Aug 23   - Sept. 21 : Virgo
Sept. 22 - Oct. 22  : Libran
Oct. 23  - Nov. 21  : Scorpio
Nov. 22  - Dec. 21  : Sagittarius
```

Value

character vector or factor with the zodiac.

Author(s)

Andri Signorell <andri@signorell.net>, based on code from Markus Naepflin

See Also

Year and other date functions

Examples

```
Zodiac(as.Date(c("1937-07-28", "1936-06-01", "1966-02-25",
                 "1964-11-17", "1972-04-25")), lang="deu")

d <- sample(seq(as.Date("2015-01-01"), as.Date("2015-12-31"), 1), 120)
z <- Zodiac(d)
Desc(z)
```

ZTest *Z Test for Known Population Standard Deviation*

Description

Compute the test of hypothesis and compute confidence interval on the mean of a population when the standard deviation of the population is known.

Usage

```
ZTest(x, ...)

## Default S3 method:
ZTest(x, y = NULL, alternative = c("two.sided", "less", "greater"),
      paired = FALSE, mu = 0, sd_pop, conf.level = 0.95, ... )

## S3 method for class 'formula'
ZTest(formula, data, subset, na.action, ...)
```

Arguments

x	numeric vector of data values. Non-finite (e.g. infinite or missing) values will be omitted.
y	an optional numeric vector of data values: as with x non-finite values will be omitted.
mu	a number specifying the hypothesized mean of the population.

sd_pop	a number specifying the known standard deviation of the population.
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter. For one-sample tests, alternative refers to the true mean of the parent population in relation to the hypothesized value of the mean.
paired	a logical indicating whether you want a paired z-test.
conf.level	confidence level for the interval computation.
formula	a formula of the form lhs ~ rhs where lhs gives the data values and rhs a factor with two levels giving the corresponding groups.
data	an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").
...	further arguments to be passed to or from methods.

Details

Most introductory statistical texts introduce inference by using the z-test and z-based confidence intervals based on knowing the population standard deviation. However statistical packages often do not include functions to do z-tests since the t-test is usually more appropriate for real world situations. This function is meant to be used during that short period of learning when the student is learning about inference using z-procedures, but has not learned the t-based procedures yet. Once the student has learned about the t-distribution the t.test() function should be used instead of this one (but the syntax is very similar, so this function should be an appropriate introductory step to learning t.test()).

The formula interface is only applicable for the 2-sample tests.

Value

A list with class "htest" containing the following components:

statistic	the value of the z-statistic.
p.value	the p-value for the test
conf.int	a confidence interval for the mean appropriate to the specified alternative hypothesis.
estimate	the estimated mean or difference in means depending on whether it was a one-sample test or a two-sample test.
null.value	the specified hypothesized value of the mean or mean difference depending on whether it was a one-sample test or a two-sample test.
alternative	a character string describing the alternative hypothesis.
method	a character string indicating what type of test was performed.
data.name	a character string giving the name(s) of the data.

Author(s)

Andri Signorell <andri@signorell.net>, based on R-Core code of t.test,
documentation partly from Greg Snow <greg.snow@imail.org>

References

Stahel, W. (2002) *Statistische Datenanalyse, 4th ed*, vieweg

See Also

t.test, print.htest

Examples

```
x <- rnorm(25, 100, 5)
ZTest(x, mu=99, sd_pop=5)

# the classic interface
with(sleep, ZTest(extra[group == 1], extra[group == 2], sd_pop=2))

# the formula interface
ZTest(extra ~ group, data = sleep, sd_pop=2)


# Stahel (2002), pp. 186, 196

d.tyres <- data.frame(A=c(44.5,55,52.5,50.2,45.3,46.1,52.1,50.5,50.6,49.2),
                      B=c(44.9,54.8,55.6,55.2,55.6,47.7,53,49.1,52.3,50.7))
with(d.tyres, ZTest(A, B, sd_pop=3, paired=TRUE))


d.oxen <- data.frame(ext=c(2.7,2.7,1.1,3.0,1.9,3.0,3.8,3.8,0.3,1.9,1.9),
                     int=c(6.5,5.4,8.1,3.5,0.5,3.8,6.8,4.9,9.5,6.2,4.1))
with(d.oxen, ZTest(int, ext, sd_pop=1.8, paired=FALSE))
```

%like% *Like Operator*

Description

The like operator is a simple wrapper for grep(..., value=TRUE), whose complexity is hard to crack for R-newbies.

Usage

```
x %like% pattern

x %like any% pattern
```

Arguments

x	a vector, typically of character or factor type
pattern	simple character string to be matched in the given character vector.

Details

Follows the logic of simple SQL or basic commands.

Value

a vector (numeric, character, factor), matching the mode of x

Author(s)

Andri Signorell <andri@signorell.net>

See Also

match, pmatch, grep, %[]%, %overlaps%

Examples

```
# find names ending on "or"
names(d.pizza) %like% "%or"

# find names starting with "d"
names(d.pizza) %like% "d%"

# ... containing er?
names(d.pizza) %like% "%er%"

# and combined, search for a name containing "un", ending on "or"
# or beginning with "F"
levels(d.pizza$driver) %like any% c("%un%", "%or", "F%")

# the positions on the vector
match(names(d.pizza) %like% "%er%", names(d.pizza))
```

%nin% *Find Matching (or Non-Matching) Elements*

Description

%nin% is a binary operator, which returns a logical vector indicating if there is a match or not for its left operand. A true vector element indicates no match in left operand, false indicates a match.

Usage

```
x %nin% table
```

Arguments

x	a vector (numeric, character, factor)
table	a vector (numeric, character, factor), matching the mode of x

Value

vector of logical values with length equal to length of x.

Author(s)

Frank E Harrell Jr <f.harrell@vanderbilt.edu>

See Also

match, %in%

Examples

```
c('a','b','c') %nin% c('a','b')
```

%overlaps% *Determines If And How Extensively Two Date Ranges Overlap*

Description

%overlaps% determines if two date ranges overlap at all and returns a logical value. Interval returns the number of days of the overlapping part of the two date periods. Inspired by the eponymous SQL-functions.

Usage

```
x %overlaps% y

Overlap(x, y)

Interval(x, y)
```

Arguments

x	range 1, vector of 2 numeric values or matrix with 2 columns, the first defining the left point the second the right point of the range.
y	range 2, vector of 2 numeric values or matrix with 2 columns, the first defining the left point the second the right point of the range.

Details

`%overlaps%` returns `TRUE` or `FALSE` depending on if the two ranges overlap. The function `Overlap` returns the range of the overlapping region as numeric value. This will be 0, if the ranges do not overlap.
`Interval` returns the width of the empty space between 2 ranges. Again this will be 0 if the ranges overlap.

To handle overlapping ranges there are 4 cases to consider:

```
range a:     |--------------|
range b:  |-----|
range c:              |--------|
range d:           |-----|
          1  2  3  4  5  6  7  8
```

Ranges a and b overlap, the function `Overlap` will return the absolute value of the overlapping region (which will be 3 - 2 = 1 in this case). The result will be the same for `Overlap(a, b)` and `Overlap(b, a)`.
`Interval` will have a direction. Ranges b and c do not overlap, `Overlap` will return 0, `%overlaps%` FALSE. `Interval` will return 2 for the case `Interval(a, b)` and -2 for `Interval(b, a)`.

This functions can be of value, if one has to decide, whether confidence intervals overlap or not.

Value

returns a logical vector (match or not for each element of x).
Interval and Overlap return a numeric vector.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

similar operators: `Between`, `%like%`
for calculating the overlapping time: `difftime`

Examples

```
as.Date(c("2012-01-03", "2012-02-03")) %overlaps%
  as.Date(c("2012-03-01", "2012-03-03"))
as.Date(c("2012-01-03", "2012-02-03")) %overlaps%
  as.Date(c("2012-01-15", "2012-01-21"))

Interval(as.Date(c("2012-01-03", "2012-02-03")), as.Date(c("2012-03-01", "2012-03-03")))


# both ranges are recyled if necessary
as.Date("2012-01-03") %overlaps% as.Date(c("2012-03-01", "2012-03-03"))

# works with numerics as well
c(1, 18) %overlaps% c(10, 45)
```

%c% *Concatenates Two Strings Without Any Separator*

Description

%c% is just a short operator implementation for paste(x, y, separator="").

Usage

```
x %c% y
```

Arguments

x	first string
y	second string, which will be pasted behind the first one.

Details

R-Core does not consider it a good idea to use + as an operator not being commutative. So we use c here.
See the discussion: https://stat.ethz.ch/pipermail/r-devel/2006-August/039013.html and https://stackoverflow.com/questions/1319698/why-doesnt-operate-on-characters-in-r?lq=1

Still the paste syntax seems sometimes clumsy in daily life and so %c% might spare some keys.

Value

returns the concatenation as string.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

Between, %like%

Examples

```
"foo" %c% "bar"

# works with numerics as well
345 %c% 457
```

Index

∗ **Desc**
Abstract, 29
Desc, 169
∗ **IO**
AllIdentical, 35
Format, 230
IQRw, 288
ParseSASDatalines, 393
RndPairs, 500
Sample, 512
TOne, 587
∗ **MS-Office**
PowerPoint Interface, 471
SendOutlookMail, 518
WrdBookmark, 630
WrdCaption, 632
WrdCellRange, 633
WrdFont, 634
WrdFormatCells, 635
WrdMergeCells, 636
WrdPageBreak, 637
WrdParagraphFormat, 638
WrdPlot, 640
WrdSaveAs, 642
WrdStyle, 643
WrdTable, 644
WrdTableBorders, 645
XLDateToPOSIXct, 647
XLGetRange, 648
XLSaveAs, 651
XLView, 652
∗ **Statistical summary functions**
Abstract, 29
Desc, 169
∗ **aplot**
Asp, 43
AxisBreak, 51
axTicks.POSIXct, 52
BarText, 58
BoxedText, 83
BubbleLegend, 92
ColorLegend, 115
ConnLines, 129
DrawArc, 194
DrawBand, 195
DrawBezier, 196
DrawCircle, 197
DrawEllipse, 199
DrawRegPolygon, 201
ErrBars, 214
GeomTrans, 247
lines.lm, 319
lines.loess, 321
LineToUser, 323
PolarGrid, 466
Rotate, 503
Shade, 521
Stamp, 547
TitleRect, 582
∗ **arith**
Base Conversions, 60
CartToPol, 94
ConvUnit, 134
DegToRad, 167
DigitSum, 184
Divisors, 188
Extremes, 219
Factorize, 222
Frac, 235
Gmean, 257
Hmean, 271
IsDichotomous, 290
IsEuclid, 291
IsOdd, 292
matpow, 335
NPV, 378
PMT, 463
Primes, 475

Quot, 482
RomanToInt, 502
* **array**
Abind, 26
Cor, 136
Cross, 149
CrossN, 150
Dot, 192
matpow, 335
MultMerge, 373
Var, 616
* **attribute**
CollapseTable, 113
Unwhich, 613
* **category**
BinomDiffCI, 73
split.formula, 541
SplitAt, 542
* **character**
%nin%, 662
Phrase, 403
SplitToCol, 544
StrAbbr, 551
StrAlign, 552
StrCap, 556
StrChop, 557
StrCountW, 558
StrDist, 559
StrExtract, 561
StrIsNumeric, 563
StrLeft, StrRight, 564
StrPad, 565
StrPos, 566
StrRev, 567
StrSpell, 567
StrSplit, 568
StrTrim, 569
StrTrunc, 571
StrVal, 572
* **chron**
AddMonths, 31
as.ym, 41
axTicks.POSIXct, 52
CountWorkDays, 144
Date Functions, 164
HmsToSec, 272
IsDate, 289
XLDateToPOSIXct, 647
Zodiac, 658
* **color**
ColToGrey, 117
ColToHex, 118
ColToHsv, 119
ColToOpaque, 120
ColToRgb, 121
DescTools Palettes, 179
FindColor, 225
HexToCol, 269
HexToRgb, 270
MixColor, 362
RgbToCol, 499
SetAlpha, 519
TextContrastColor, 578
* **combinatorics**
CombPairs, 123
GenRandGroups, 245
Permn, 402
* **confidence interval**
BinomCIn, 71
MeanCIn, 342
QuantileCI, 480
* **confidence**
BrierScoreCI, 90
CstatCI, 153
RSqCI, 507
* **datagen**
Triangular, 598
* **datasets**
CourseData, 145
d.countries, 157
d.diamonds, 158
d.periodic, 159
d.pizza, 160
d.whisky, 161
Datasets for Simulation, 163
day.name, 166
wdConst, 626
* **data**
MultMerge, 373
Rank, 484
SplitAt, 542
* **dates**
Date Functions, 164
* **dictionary sort**
SortMixed, 537
* **distribution**

Benford, 61
ExtrVal, 221
Frechet, 236
Freq2D, 240
GenExtrVal, 242
GenPareto, 244
Gompertz, 259
GTest, 264
Gumbel, 266
IQRw, 288
Order, 382
pRevGumbel, 474
RevWeibull, 497
RndPairs, 500
Sample, 512
Triangular, 598
∗ **documentation**
Keywords, 305
∗ **dplot**
CartToPol, 94
Clockwise, 99
ColToOpaque, 120
FindColor, 225
Freq2D, 240
∗ **file**
SaveAs, 514
∗ **goodness-of-fit**
GTest, 264
∗ **hplot**
Canvas, 93
identify.formula, 284
PlotACF, 404
PlotArea, 405
PlotBag, 407
PlotBubble, 411
PlotCandlestick, 413
PlotCirc, 416
PlotCorr, 419
PlotDot, 422
PlotECDF, 425
PlotFaces, 426
PlotFdist, 429
PlotFun, 432
PlotLinesA, 435
PlotLog, 437
PlotMarDens, 438
PlotMiss, 439
PlotMonth, 440
PlotMosaic, 441
PlotMultiDens, 442
PlotPairs, 444
PlotPolar, 445
PlotPyramid, 450
PlotQQ, 453
PlotTernary, 454
PlotTreemap, 456
PlotVenn, 457
PlotViolin, 459
PlotWeb, 461
∗ **htest**
AndersonDarlingTest, 36
BarnardTest, 53
BartelsRankTest, 56
BhapkarTest, 67
BreslowDayTest, 85
BreuschGodfreyTest, 87
CochranArmitageTest, 102
CochranQTest, 104
ConoverTest, 130
Contrasts, 133
CramerVonMisesTest, 146
DunnettTest, 204
DunnTest, 206
DurbinWatsonTest, 209
EtaSq, 215
GTest, 264
HoeffD, 275
HotellingsT2Test, 278
JarqueBeraTest, 294
JonckheereTerpstraTest, 296
LehmacherTest, 314
LeveneTest, 316
LillieTest, 318
MHChisqTest, 359
MosesTest, 366
NemenyiTest, 375
PageTest, 387
PearsonTest, 396
PostHocTest, 467
power.chisq.test, 470
RunsTest, 509
ScheffeTest, 515
ShapiroFranciaTest, 522
SiegelTukeyTest, 523
SignTest, 526
StuartMaxwellTest, 573

TTestA, 602
VanWaerdenTest, 614
VarTest, 619
VonNeumannTest, 625
WoolfTest, 629
YuenTTest, 654
ZTest, 659
∗ **interval**
BrierScoreCI, 90
CstatCI, 153
RSqCI, 507
∗ **iteration**
AllIdentical, 35
∗ **list**
BinTree, 78
SetNames, 520
∗ **logic**
AllIdentical, 35
Between, Outside, 63
Closest, 100
CompleteColumns, 124
IsDate, 289
Some numeric checks, 532
Unwhich, 613
∗ **manipulate**
as.matrix.xtabs, 40
∗ **manipulation**
StripAttr, 562
∗ **manip**
%c%, 665
%like%, 661
%nin%, 662
%overlaps%, 663
Abind, 26
AllDuplicated, 34
Append, 38
AscToChar, 42
Between, Outside, 63
Coalesce, 101
CollapseTable, 113
CombPairs, 123
CutQ, 155
FctArgs, 223
FindCorr, 227
FixToTable, 229
Freq2D, 240
IdentifyA, 285
InDots, 287
MultMerge, 373
PairApply, 390
ParseFormula, 392
Recode, 486
Rename, 491
reorder.factor, 492
Rev, 494
RoundTo, 504
Some, 531
Sort, 535
SortMixed, 537
SplitPath, 543
StrLeft, StrRight, 564
StrTrim, 569
TextToTable, 580
ToLong, ToWide, 585
Untable, 611
VecRot, 621
XLGetRange, 648
XLSaveAs, 651
XLView, 652
ZeroIfNA, 657
∗ **math**
AUC, 49
DigitSum, 184
Divisors, 188
Dummy, 202
Factorize, 222
lines.lm, 319
lines.loess, 321
Logit, 328
LogSt, 329
Measures of Shape, 349
Permn, 402
Primes, 475
StrDist, 559
Vigenere, 624
∗ **methods1**
MultMerge, 373
∗ **methods**
LOF, 327
∗ **misc**
BoxedText, 83
GetCurrWrd, 249
GetNewWrd, 250
GetNewXL, 252
KrippAlpha, 306
Label, Unit, 307

PlotBag, 407
SpreadOut, 546
* **models**
BrierScore, 89
CorCI, 139
CorPolychor, 142
Eps, 213
FisherZ, 228
ImputeKnn, 286
OddsRatio, 380
StdCoef, 548
* **model**
Measures of Accuracy, 346
PseudoR2, 476
* **mod**
Cstat, 151
TMod, 583
* **multivariate**
Association measures, 44
Assocs, 46
ConDisPairs, 124
Cor, 136
CorCI, 139
CorPart, 140
Desc, 169
DivCoef, 185
DivCoefMax, 186
ExpFreq, 218
FisherZ, 228
Freq2D, 240
HotellingsT2Test, 278
ICC, 281
PercTable, 399
PlotCorr, 419
PlotViolin, 459
PlotWeb, 461
RelRisk, 489
UncertCoef, 607
Var, 616
* **multivar**
CohenKappa, 110
Conf, 125
CronbachAlpha, 148
GoodmanKruskalGamma, 260
GoodmanKruskalTau, 262
KappaM, 298
KendallTauA, 300
KendallTauB, 301
KendallW, 303
OddsRatio, 380
SomersDelta, 534
SpearmanRho, 539
StuartTauC, 575
TheilU, 581
* **natural sort**
SortMixed, 537
* **nonparametric**
BarnardTest, 53
BootCI, 80
GoodmanKruskalGamma, 260
GoodmanKruskalTau, 262
HodgesLehmann, 273
HoeffD, 275
KendallTauA, 300
KendallTauB, 301
SomersDelta, 534
StuartTauC, 575
* **optimize**
UnirootAll, 609
* **package**
DescTools-package, 13
* **print**
Abstract, 29
CatTable, 95
ColumnWrap, 122
Desc, 169
PowerPoint Interface, 471
ToWrd, 591
ToWrdB, 595
ToWrdPlot, 596
WrdBookmark, 630
WrdCaption, 632
WrdCellRange, 633
WrdFont, 634
WrdFormatCells, 635
WrdMergeCells, 636
WrdPageBreak, 637
WrdParagraphFormat, 638
WrdPlot, 640
WrdSaveAs, 642
WrdStyle, 643
WrdTable, 644
WrdTableBorders, 645
* **programming**
AllIdentical, 35
* **regression**

Eps, 213
VIF, 622
∗ **robust location**
HuberM, 280
∗ **robust**
HodgesLehmann, 273
HuberM, 280
IQRw, 288
MAD, 331
Median, 351
RobScale, 501
Trim, 601
TukeyBiweight, 604
Winsorize, 627
∗ **smooth**
SmoothSpline, 529
∗ **string**
StrAbbr, 551
StrAlign, 552
StrCap, 556
StrChop, 557
StrCountW, 558
StrDist, 559
StrExtract, 561
StripAttr, 562
StrIsNumeric, 563
StrLeft, StrRight, 564
StrPad, 565
StrPos, 566
StrRev, 567
StrSpell, 567
StrSplit, 568
StrTrim, 569
StrTrunc, 571
StrVal, 572
∗ **survey**
SampleTwins, 513
Strata, 554
∗ **test**
HosmerLemeshowTest, 276
PostHocTest, 467
∗ **ts**
BoxCoxLambda, 82
∗ **univar**
Agree, 32
Atkinson, 48
BinomCI, 68
BootCI, 80
BoxCox, 81
CCC, 96
CoefVar, 106
Cor, 136
CutQ, 155
Desc, 169
DoBy, 189
Entropy, 211
Freq, 238
Gini, 253
GiniSimpson, 255
Herfindahl, 268
HodgesLehmann, 273
HosmerLemeshowTest, 276
HuberM, 280
IQRw, 288
Lambda, 309
Lc, 311
LinScale, 324
LOCF, 326
MAD, 331
Mean, 338
MeanAD, 339
MeanCI, 340
MeanDiffCI, 343
MeanSE, 345
Median, 351
MedianCI, 353
Midx, 361
Mode, 364
MoveAvg, 368
MultinomCI, 370
ORToRelRisk, 384
Outlier, 386
PercentRank, 398
PlotQQ, 453
PoissonCI, 464
Quantile, 479
Range, 483
RobScale, 501
SD, 517
SortMixed, 537
Trim, 601
TukeyBiweight, 604
Var, 616
VarCI, 617
Winsorize, 627
∗ **utilities**

Label, Unit, 307
List Variety Of Objects, 325
Mar and Mgp, 334
PasswordDlg, 395
Recycle, 488
Str, 550
StrAbbr, 551
StrCap, 556
StrCountW, 558
StrDist, 559
StrExtract, 561
StrPos, 566
StrTrunc, 571
StrVal, 572
[, *622*
%)[% (Between, Outside), 63
%::% (Between, Outside), 63
%:% (Between, Outside), 63
%[)% (Between, Outside), 63
%[]% (Between, Outside), 63
%^% (matpow), 335
%][% (Between, Outside), 63
%like any% (%like%), 661
%()%, *14*
%)(%, *14*
%*%, *336*
%[]%, *662*
%^%, *14*
%c%, 665
%in%, *663*
%like any%, *14*
%like%, *14*, 661, *664, 665*
%nin%, *14*, 662
%overlaps%, *14*, *662*, 663

abbreviate, *551*
ABCCoords, 25
Abind, *15*, 26
abline, *319*
Abstract, *22*, 29, *176*
addmargins, *401*
AddMonths, *21*, 31, *42*
AddMonths.ym (as.ym), 41
adist, *560*
adjustcolor, *519*
aggregate, *190*
Agree, 32
AIC, *478*
alist, *223*
AllDuplicated, *15*, 34
AllIdentical, 35
AndersonDarlingTest, *20*, 36, *147*, *295*, *319*, *398*, *523*
Anova, *216*
anova, *216*
ansari.test, *524, 525*, *621*
ansari_test, *317*
aov, *213*, *216*, *469*, *515*
aovlDetails (EtaSq), 215
aovlErrorTerms (EtaSq), 215
Append, *15*, 38, *40*
append, *39*
AppendRowNames, 39
apply, *354*
areaplot, *406*
args, *223*
array, *28*
arrows, *214, 215*
as.CDateFmt (Format), 230
as.character, *561*
as.Date, *31*, *179*
as.Date.ym (as.ym), 41
as.factor, *189*, *541*
as.fmt (Format), 230
as.integer, *236*, *335*
as.matrix, *40*, *580*
as.matrix.xtabs, 40
as.numeric, *179*, *422*
as.POSIXct, *647, 648*
as.POSIXlt, *166*
as.roman, *503*
as.table, *580*
as.ym, *32*, 41
AscToChar, *16*, 42
Asp, *17*, 43
Association measures, 44
Assocs, *19*, *45*, 46, *608*
Atkinson, *19*, 48, *269*, *313*
AUC, *14*, 49
ave, *190*
axis, *51, 52*, *437*
axis.POSIXct, *53*
AxisBreak, *17*, 51
axTicks, *52, 53*
axTicks.Date (axTicks.POSIXct), 52
axTicks.POSIXct, 52

BarnardTest, *21*, 53

barplot, *59*, *129*, *406*, *451*, *452*, *457*
BartelsRankTest, *20*, 56, *626*
BarText, *17*, 58, *129*
bartlett.test, *317*, *621*
Base Conversions, 60
base::plot(), *176*
base::summary(), *31*, *176*
basename, *543*
Benford, 61
Between, *664*, *665*
Between *(*Between, Outside*)*, 63
Between, Outside, 63
Bg, 66
BhapkarTest, 67, *574*
BIC, *478*
binconf, *71*
binom.test, *71*, *75*, *528*
BinomCI, *19*, 68, *71*, *72*, *75*, *78*, *127*, *465*, *577*
BinomCI(), *175*
BinomCIn, 71, *343*
BinomDiffCI, *19*, *71*, 73, *78*
BinomRatioCI, *19*, *71*, *75*, 75
BinToDec, *16*
BinToDec *(*Base Conversions*)*, 60
BinTree, *15*, 78
BlankIfNA *(*ZeroIfNA*)*, 657
body, *223*
bondyield, *209*
boot, *91*, *153*, *257*, *271*, *341*, *344*, *349*, *507*, *605*, *618*
boot.ci, *80*, *257*, *271*, *280*, *341*, *344*, *345*, *354*, *481*, *618*
BootCI, 80
BoxCox, *14*, 81, *83*
boxcox, *82*
BoxCoxInv, *14*
BoxCoxInv *(*BoxCox*)*, 81
BoxCoxLambda, *14*, *82*, 82
boxed.labels, *85*
BoxedText, *17*, *25*, *59*, 83
boxplot, *387*, *410*, *430*, *431*, *460*
BreslowDayTest, *21*, *67*, 85, *574*, *630*
BreuschGodfreyTest, *21*, 87
BrierScore, *20*, 89, *91*, *152*, *154*, *507*
BrierScoreCI, 90
BubbleLegend, *17*, 92, *116*, *412*
bw.nrd, *459*

call, *521*
Canvas, *17*, *79*, 93
cards, *23*
cards *(*Datasets for Simulation*)*, 163
CartToPol, *16*, 94
CartToSph, *16*
CartToSph *(*CartToPol*)*, 94
CatTable, *22*, 95
cbind, *28*, *39*
CCC, *20*, 96
cdplot, *418*
ceiling, *505*
CharToAsc, *16*
CharToAsc *(*AscToChar*)*, 42
charToRaw, *43*
chisq.test, *45*, *67*, *219*, *266*, *360*, *574*
chisq.test(), *172*
choose, *403*
class.ind, *203*
Clockwise, *16*, 99
Closest, *15*, 100
CmToPts *(*ConvUnit*)*, 134
CmykToCmy *(*RgbToCmy*)*, 498
CmykToRgb *(*RgbToCmy*)*, 498
CmyToCmyk *(*RgbToCmy*)*, 498
Coalesce, *15*, 101
CochranArmitageTest, *21*, 102
CochranQTest, *20*, 104
coef, *549*, *622*
CoefVar, *19*, 106
CoefVarCI *(*CoefVar*)*, 106
CohenD, *20*, 109
CohenKappa, *19*, *33*, 110, *149*, *299*, *307*
col2rgb, *119*, *121*, *519*
CollapseTable, *15*, 113
colMeans, *339*
ColorLegend, *17*, 115, *419*, *420*
colorRamp, *362*
colorRampPalette, *180*
colors, *118*, *119*, *121*, *270*
ColToGray, *16*
ColToGray *(*ColToGrey*)*, 117
ColToGrey, *16*, 117
ColToHex, *16*, 118, *120*, *270*, *519*
ColToHsv, *16*, 119
ColToOpaque, 120, *519*
ColToRgb, *16*, *118*, *120*, 121, *270*, *500*
ColumnWrap, 122
ColumnWrap(), *31*

CombN, *15*, *245*
CombN *(Permn)*, 402
combn, *123*, *403*
CombPairs, *15*, 123, *391*, *403*
CombSet, *15*, *245*
CombSet *(Permn)*, 402
Comparison, *64*
complete.cases, *45*, *124*, *144*, *287*
CompleteColumns, 124, *144*
compute.bagplot *(PlotBag)*, 407
Concatenate Strings *(%c%)*, 665
ConDisPairs, *20*, 124, *261*, *263*, *301*, *302*, *535*, *576*
Conf, *20*, *90*, 125
confusionMatrix, *128*
ConnLines, *17*, 129
ConoverTest, *21*, 130, *376*
ContCoef, *19*, *47*
ContCoef *(Association measures)*, 44
Contrasts, *20*, 133
contrasts, *203*
ConvUnit, *17*, 134
Cor, *20*, 136
cor, *45*, *125*, *141*, *261*, *263*, *301*, *302*, *304*, *310*, *535*, *540*, *576*, *617*
cor.fk, *137*
cor.test, *138*, *229*, *617*
CorCI, *20*, 139
CorPart, *20*, 140
CorPolychor, *20*, 142
corr, *360*
corrgram, *420*
CountCompCases, 143, *440*
CountWorkDays, 144
CourseData, 145
Cov *(Cor)*, 136
cov, *617*
cov.wt, *138*, *617*
CramerV, *19*, *47*
CramerV *(Association measures)*, 44
CramerVonMisesTest, *20*, 146, *295*, *319*, *398*, *523*
createCOMReference *(GetNewWrd)*, 250
CronbachAlpha, *19*, *111*, 148, *304*, *307*
Cross, 149, *151*, *193*
CrossN, *150*, 150
Cstat, *20*, 151
CstatCI, 153
cumsum, *239*
currencysubstitution, *209*
curve, *430*, *433*, *449*, *522*
cut, *154–156*, *238*, *239*, *241*, *419*
CutAge, 154
CutGen *(CutAge)*, 154
CutQ, *15*, 155

d.countries, *23*, 157
d.diamonds, 158
d.periodic, *23*, 159
d.pizza, *23*, 160
d.prefix, *23*, *233*
d.prefix *(ConvUnit)*, 134
d.units, *23*
d.units *(ConvUnit)*, 134
d.whisky, *23*, 161
data.frame, *146*
data.table, *485*
Datasets for Simulation, 163
Date, *21*, *352*
date, *338*
Date Functions, 164
date-time, *338*
DateTimeClasses, *166*
Day, *21*
Day *(Date Functions)*, 164
day.abb, *21*
day.abb *(day.name)*, 166
day.name, *21*, 166
Day<- *(Date Functions)*, 164
DB, *22*
DB *(Depreciation)*, 168
dBenf, *18*
dBenf *(Benford)*, 61
dbeta, *336*
dbinom, *337*
dchisq, *338*
DecToBin, *16*
DecToBin *(Base Conversions)*, 60
DecToHex, *16*, *120*
DecToHex *(Base Conversions)*, 60
DecToOct, *16*
DecToOct *(Base Conversions)*, 60
DegToRad, *16*, *95*, 167, *200*
density, *418*, *430*, *431*, *438*, *439*, *444*, *459*, *460*
Depreciation, 168
Desc, *22*, *31*, 169, *607*

Desc(), *31*, *175*
Desc.formula, *392, 393*
DescTools *(DescTools-package)*, 13
DescTools Aliases, 179
DescTools Palettes, 179
DescTools-package, 13
DescToolsOptions, *23*, 181, *234*, *401*
DescToolsOptions(digits=x), *173*
dexp, *260*, *355*
dExtrVal, *18*
dExtrVal *(ExtrVal)*, 221
df, *356*
dFrechet, *18*
dFrechet *(Frechet)*, 236
dgamma, *357*
dGenExtrVal, *18*
dGenExtrVal *(GenExtrVal)*, 242
dGenPareto, *18*
dGenPareto *(GenPareto)*, 244
dgeom, *357*
dGompertz, *18*
dGompertz *(Gompertz)*, 259
dGumbel, *18*
dGumbel *(Gumbel)*, 266
dhyper, *360*
diff, *482*
DiffDays360, *21*
DiffDays360 *(Date Functions)*, 164
difftime, *664*
DigitSum, *14*, 184
dirname, *543*
dist, *560*
DivCoef, *19*, 185, *256*
DivCoefMax, *19*, 186
Divisors, *14*, 188, *222*
dlnorm, *363*
dnbinom, *363*
dNegWeibull, *18*
dNegWeibull *(RevWeibull)*, 497
dnorm, *364*
do.call, *191*, *426*
DoBy, 189
DoCall, 191
dOrder, *18*
dOrder *(Order)*, 382
Dot, *150, 151*, 192
dotchart, *422*, *424*
Downloads, 193
dpois, *369*
DrawArc, *17*, 194, *197, 198*, *200, 201*, *247*, *504*
DrawBand, *17*, 195, *312*, *320*, *322*, *453*
DrawBezier, *17*, 196
DrawCircle, *17*, *195*, *197*, 197, *200, 201*
DrawEllipse, *17*, *198*, 199, *247*, *504*
DrawRegPolygon, *17*, *197, 198*, *200*, 201, *247*, *504*
dRevGumbel, *18*
dRevGumbel *(pRevGumbel)*, 474
dRevWeibull, *18*
dRevWeibull *(RevWeibull)*, 497
dt, *370*
dTri *(Triangular)*, 598
Dummy, *15*, 202
DunnettTest, *21*, 204
DunnTest, *21*, *131, 132*, 206, *376*
duplicated, *34*
DurbinWatsonTest, *20*, *88*, 209
dweibull, *260*

ecdf, *431*
Encoding, *569*
Entropy, *19*, 211, *256*, *608*
Eps, 213
ErrBars, *17*, 214, *423*
EtaSq, *20*, 215
EX, 217
expand.grid, *123*, *612*
ExpFreq, *18*, 218
expression, *521*
Extremes, 219
ExtrVal, 221

factor, *203*, *375*, *377*, *398*, *487*, *493*
factorial, *403*
Factorize, *14*, 222, *242*, *293*, *475*
factorize, *222*
Fade *(SetAlpha)*, 519
FctArgs, *23*, 223
Fibonacci, *14*, 224
filter, *368*
FindColor, *17*, *93*, *116*, 225
FindCorr, *20*, 227
findInterval, *226*
FindRProfile *(SysInfo)*, 578
fisher.test, *56*
FisherZ, *15*, *140*, 228
FisherZInv, *15*, *140*

FisherZInv (FisherZ), 228
fit.fkml, *333*
FixToTable, *16*, 229, *557*
Flags (IsDichotomous), 290
fligner.test, *317*
floor, *505*
Fmt, *22*, *182*, *400*, *401*
Fmt (Format), 230
forder, *485*
formalArgs, *223*
formals, *223*
Format, *22*, *182*, *183*, 230, *400*, *553*, *589*
format, *234*
format.info, *236*
format.pval, *233*
formatC, *231*, *234*, *584*, *593*
formula, *240*, *285*, *393*, *406*
Frac, *14*, 235
fractions, *233*
frankv, *485*
Frechet, 236
Freq, *18*, 238, *241*, *338*, *352*, *400*, *401*, *616*
Freq(), *176*
Freq2D, *239*, 240
friedman.test, *303*, *304*, *389*
ftable, *401*
function, *530*

GCD, *14*, *188*, *222*, *475*
GCD (GCD, LCM), 241
GCD, LCM, 241
GenExtrVal, 242
GenPareto, 244
GenRandGroups, 245
GeomSn, 246
GeomTrans, 247
GetCalls, 248
GetCurrPP, *22*, *294*
GetCurrPP (PowerPoint Interface), 471
GetCurrWrd, *22*, 249, *294*, *631*, *632*, *634*, *637*, *640*, *643*
GetCurrWrd(), *173*
GetCurrXL, *22*, *294*, *650*
GetCurrXL (GetCurrWrd), 249
GetNewPP, *22*, *251*
GetNewPP (PowerPoint Interface), 471
GetNewWrd, *22*, *249*, 250, *594*, *631*, *632*, *634*, *637*, *640–644*
GetNewWrd(), *173*
GetNewXL, *22*, *251*, 252, *650*, *653*
getOption, *629*
Gini, *19*, 253, *256*, *269*, *313*, *581*
GiniDeltas (GiniSimpson), 255
GiniSimpson, *19*, 255
GKgamma, *261*
gl, *612*
glm, *90*, *277*
Gmean, *19*, 257, *272*, *379*
Gompertz, 259
GoodmanKruskalGamma, *19*, *47*, 260, *263*, *301*, *302*, *310*, *535*, *576*
GoodmanKruskalTau, *19*, *125*, *261*, 262, *302*, *310*, *535*, *576*
graphical parameters, *423*
grep, *567*, *569–571*, *661*, *662*
grey, *118*
grid, *437*
growthofmoney, *209*
Gsd, *19*
Gsd (Gmean), 257
gsub, *358*, *491*, *567*, *570*, *571*
GTest, *21*, 264
Gumbel, 266

hblue (DescTools Palettes), 179
hclust, *440*
head, *532*
hecru (DescTools Palettes), 179
help, *305*, *584*
Herfindahl, *19*, *49*, *255*, *256*, 268
hetcor, *143*
HexToCol, *16*, *118*, 269, *271*
HexToDec, *16*
HexToDec (Base Conversions), 60
HexToRgb, *16*, 270
hgreen (DescTools Palettes), 179
HighLow, *15*
HighLow (Extremes), 219
hist, *238*, *239*, *425*, *430*, *431*
Hmean, *19*, *258*, 271
HmsToMinute (Date Functions), 164
HmsToSec, *21*, 272
HodgesLehmann, *19*, 273, *354*
HoeffD, *19*, 275
horange (DescTools Palettes), 179
HosmerLemeshowTest, *21*, 276
HotellingsT2Test, *21*, 278
Hour, *21*

Hour (Date Functions), 164
hred (DescTools Palettes), 179
huber, *280*
HuberM, *19*, 280, *605*
hubers, *281*
HunterGaston (GiniSimpson), 255
hyellow (DescTools Palettes), 179

ICC, *19*, *98*, 281, *304*
identical, *35*, *36*
identify, *17*, *284*, *286*
identify.formula, *17*, 284
IdentifyA, *17*, 285
if, *64*
ifelse, *64*
image, *419*, *420*
Impute, *15*
Impute (ZeroIfNA), 657
ImputeKnn, 286
InDots, *23*, 287
ineq, *49*, *255*, *269*
integrate, *50*
intersect, *34*
Interval, *14*, *64*
Interval (%overlaps%), 663
IPMT (PMT), 463
IQR, *289*, *332*
IQRw, *20*, 288, *332*
IRR, *22*
IRR (NPV), 378
is.finite, *102*
is.integer, *533*
is.na, *102*, *124*, *144*
is.numeric, *137*, *422*
IsDate, *21*, 289
IsDichotomous, *15*, 290
IsEuclid, *15*, 291
IsLeapYear, *21*
IsLeapYear (Date Functions), 164
IsNumeric, *15*
IsNumeric (Some numeric checks), 532
IsOdd, *15*, 292
IsPrime, *15*, *184*, *188*, *222*, *242*, 292, *475*
IsValidHwnd, *22*, *249*, 293
IsWeekend, *21*
IsWeekend (Date Functions), 164
IsWhole, *15*, *292*
IsWhole (Some numeric checks), 532
IsZero, *15*
IsZero (Some numeric checks), 532

JarqueBeraTest, *20*, 294
JonckheereTerpstraTest, *20*, 296

KappaM, *19*, *33*, *111*, *149*, 298, *304*, *307*
KendallTauA, *19*, *125*, *261*, *263*, 300, *302*, *310*, *535*, *576*
KendallTauB, *19*, *47*, *261*, *300*, 301, *310*, *535*
KendallW, *19*, *98*, 303
Keywords, *23*, 305
KrippAlpha, *19*, *111*, 306
kruskal.test, *130*, *132*, *208*, *614*
ks.test, *367*
Kurt, *19*
Kurt (Measures of Shape), 349
Kurt(), *176*

Label, *15*
Label (Label, Unit), 307
label, *308*
Label(), *30*
Label, Unit, 307
Label<- (Label, Unit), 307
Labels (Label, Unit), 307
Labels<- (Label, Unit), 307
lag, *622*
Lambda, *19*, *47*, *125*, *261*, *263*, *301*, *302*, 309, *535*, *576*, *608*
Large, *15*
Large (Extremes), 219
LastDayOfMonth, *21*
LastDayOfMonth (Date Functions), 164
layout, *431*, *439*
Lc, *19*, *255*, 311, *313*
LCM, *14*, *188*, *222*, *475*
LCM (GCD, LCM), 241
legend, *93*, *116*
LehmacherTest, *21*, 314
length, *188*, *474*
levels, *375*, *487*
LeveneTest, *20*, 316, *525*
LillieTest, *21*, *147*, *295*, 318, *398*, *523*
lines, *320*
lines.Lc (Lc), 311
lines.lm, *17*, 319
lines.loess, *17*, *215*, *320*, 321
lines.smooth.spline, *17*, *531*
lines.smooth.spline (lines.loess), 321

lines.SmoothSpline (lines.loess), 321
LineToUser, 17, 323
LinScale, 14, 324
list, 222
List Variety Of Objects, 325
lm, 203, 210, 320, 348, 507
locator, 284, 286
LOCF, 15, 326
loess, 322
LOF, 20, 327
log, 331
log10, 331
Logit, 14, 328
logit, 329
LogitInv, 14
LogitInv (Logit), 328
logLik, 478
LogSt, 14, 329
LogStInv, 14
LogStInv (LogSt), 329
LongToRgb, 16
LongToRgb (RgbToCol), 499
Lorenz curve (Lc), 311
lower.tri, 123
ls, 325, 326
ls.str, 325, 326
lsf.str, 325, 326
LsFct, 23, 248
LsFct (List Variety Of Objects), 325
LsObj, 23
LsObj (List Variety Of Objects), 325

ma, 369
MAD, 20, 331, 333
mad, 281, 331–333, 340, 501, 517
MADCI, 332, 332
MAE, 20
MAE (Measures of Accuracy), 346
mantelhaen.test, 630
MAPE, 20
MAPE (Measures of Accuracy), 346
mapply, 41
Mar, 17
Mar (Mar and Mgp), 334
Mar and Mgp, 334
margin.table, 113, 400
Margins, 18
Margins (PercTable), 399
match, 567, 570, 571, 662, 663
matplot, 436
matpow, 335
matrix, 335
max, 220, 475, 484
MaxDigits, 14
MaxDigits (Frac), 235
mbeta, 336
mbinom, 337
mchisq, 337
mcnemar.test, 67, 127, 315, 574
Mean, 20, 108, 338, 339, 342, 366
mean, 110, 258, 351, 501, 601
mean.POSIXct, 339
MeanAD, 19, 294, 339
MeanCI, 19, 81, 272, 340, 345, 346, 354, 618
MeanCIn, 342, 342
MeanDiffCI, 19, 342, 343
MeanSE, 19, 345
MeanSE(), 175
Measures of Accuracy, 346
Measures of Shape, 349
Median, 19, 289, 351, 366, 480
median, 274, 332, 354, 501, 657
MedianCI, 19, 81, 274, 342, 345, 353, 481, 618
merge, 374
mexp, 355
mf, 356
mgamma, 356
mgeom, 357
Mgp (Mar and Mgp), 334
Mgsub, 358
MHChisqTest, 21, 67, 359, 574
mhyper, 360
Midx, 15, 361, 369
min, 484
Minute, 21
Minute (Date Functions), 164
MixColor, 17, 362
mixedsort, 493
mlnorm, 362
mnbinom, 363
mnorm, 364
Mode, 19, 364
model.frame, 104, 131, 189, 203, 205, 207, 278, 296, 312, 316, 344, 367, 375, 388, 393, 400, 411, 443, 509, 515, 524, 527, 606, 614, 620, 655, 660
model.matrix, 622

ModSummary (TMod), 583
moneydemand, *209*
Month, *21*, *32*, *42*, *234*, *290*
Month (Date Functions), 164
month.abb, *166*, *167*
month.name, *166*, *167*
MonthDays (Date Functions), 164
mood.test, *317*, *524*, *525*, *621*
mosaicplot, *442*, *457*
MosesTest, *20*, 366
MoveAvg, *19*, *361*, 368
mpois, 369
MSE, *20*
MSE (Measures of Accuracy), 346
mt, 370
mtext, *323*
MultinomCI, *19*, *71*, *75*, 370, *465*
MultMerge, 373
MutInf, *19*, *47*, *125*, *261*, *263*, *301*, *302*, *535*, *576*
MutInf (Entropy), 211

N, *378*
N (DescTools Aliases), 179
NA, *137*, *482*, *616*
na.omit, *124*, *144*, *287*
NAIf (ZeroIfNA), 657
NAIfBlank (ZeroIfNA), 657
NAIfZero, *15*
NAIfZero (ZeroIfNA), 657
NALevel, 374
names, *479*
nchar, *558*, *567*, *569–571*
Ndec, *14*
Ndec (Frac), 235
NemenyiTest, *21*, *132*, 375
Nf, 377
NMAE, *20*
NMAE (Measures of Accuracy), 346
NMSE, *20*
NMSE (Measures of Accuracy), 346
normal_test, *615*
normalizePath, *543*
Now, *21*
Now (Date Functions), 164
NPV, *21*, *168*, 378, *415*, *464*
NPVFixBond, *21*
NPVFixBond (NPV), 378
NZ, 379

OctToDec, *16*
OctToDec (Base Conversions), 60
OddsRatio, *19*, *128*, 380, *384*, *490*
OPR, *21*
OPR (NPV), 378
optim, *143*
options, *629*
options(width), *30*
Order, 382
order, *398*, *495*, *536–538*
ordered, *492*
OrderMixed, *15*
OrderMixed (SortMixed), 537
ORToRelRisk, *19*, 384
outer, *123*, *391*
Outlier, *20*, 386
Overlap, *21*, *64*
Overlap (%overlaps%), 663

p.adjust, *131*, *132*, *207*, *208*, *315*, *468*, *469*
PageTest, *20*, 387
PairApply, *22*, *45*, *309*, 390
pairs, *445*
pairwise.t.test, *469*, *516*
pairwise.table, *391*
Pal, *17*, *183*
Pal (DescTools Palettes), 179
par, *59*, *116*, *335*, *423*, *431*, *436*, *446*, *546*
ParseFormula, *23*, 392
ParseSASDatalines, *22*, 393
PartialSD (StdCoef), 548
PasswordDlg, *22*, 395
paste, *96*, *569*
pBenf (Benford), 61
PDFManual, *23*, 396
PearsonTest, *21*, *147*, *295*, *319*, 396, *523*
PercentRank, *15*, 398
PercTable, *18*, *239*, *241*, 399
Permn, *14*, 402
pExtrVal (ExtrVal), 221
pFrechet (Frechet), 236
pGenExtrVal (GenExtrVal), 242
pGenPareto (GenPareto), 244
pGompertz (Gompertz), 259
pGumbel (Gumbel), 266
Phi, *19*, *47*, *262*
Phi (Association measures), 44
Phrase, 403, *607*
plot, *284*, *412*, *435*, *439*

plot.bagplot (PlotBag), 407
plot.Conf (Conf), 125
plot.default, *320*, *433*
plot.Desc (Desc), 169
plot.ecdf, *426*, *430*
plot.Lc (Lc), 311
plot.Lclist (Lc), 311
plot.palette (DescTools Palettes), 179
plot.PostHocTest (PostHocTest), 467
plot.TMod (TMod), 583
plot.window, *94*, *423*, *433*
PlotACF, *17*, 404
PlotArea, *17*, 405
PlotBag, *17*, 407
PlotBagPairs, *18*
PlotBagPairs (PlotBag), 407
PlotBinTree, *15*
PlotBinTree (BinTree), 78
PlotBubble, *18*, 411, *414*
PlotCandlestick, *18*, 413
PlotCashFlow, 414
PlotCirc, *18*, 416, *457*
PlotConDens, 417
PlotCorr, *18*, *45*, 419, *462*
PlotDot, *18*, 422
PlotDotCI, *424*
PlotDotCI (PlotDot), 422
PlotECDF, *18*, 425
PlotFaces, *18*, 426
PlotFdist, *18*, *426*, 429
PlotFun, *18*, 432
PlotGACF, *17*
PlotGACF (PlotACF), 404
PlotLinesA, *18*, 435
PlotLog, *18*, 437
PlotMarDens, *18*, 438
PlotMiss, *18*, *144*, 439
PlotMonth, *17*, 440
PlotMosaic, *18*, 441
PlotMultiDens, *18*, *418*, 442, *460*, *607*
PlotPairs, 444
PlotPolar, *18*, *99*, *417*, 445, *467*
PlotProbDist, 448
PlotPyramid, *18*, 450
PlotQQ, *18*, 453
PlotTernary, *18*, 454
PlotTreemap, *18*, 456
PlotVenn, *18*, 457
PlotViolin, *18*, *444*, 459
PlotWeb, *18*, *420*, 461
pmatch, *662*
PMT, 463
pNegWeibull (RevWeibull), 497
points, *436*, *439*, *446*, *455*, *462*
poisson.test, *465*
PoissonCI, *19*, 464
PolarGrid, *18*, *446*, 466
PolToCart, *16*
PolToCart (CartToPol), 94
polygon, *195*, *197*, *198*, *200*, *201*, *247*, *406*, *449*, *460*, *504*, *521*, *522*
polyserial, *143*
pOrder (Order), 382
PostHocTest, *21*, *206*, 467
power.anova.test, *470*
power.chisq.test, 470
power.t.test, *471*
PowerPoint Interface, 471
PpAddSlide, *22*
PpAddSlide (PowerPoint Interface), 471
PPMT (PMT), 463
PpPlot, *22*
PpPlot (PowerPoint Interface), 471
PpText, *22*
PpText (PowerPoint Interface), 471
Prec, *14*
Prec (Frac), 235
predict.Lc (Lc), 311
pretty, *240*
prettyNum, *234*
pRevGumbel, 474
pRevWeibull (RevWeibull), 497
Primes, *14*, *188*, *222*, *242*, *293*, 475
primes, *475*
print.abstract (Abstract), 29
print.Assocs (Assocs), 46
print.Conf (Conf), 125
print.CorPolychor, *143*
print.CorPolychor (CorPolychor), 142
print.CountCompCases (CountCompCases), 143
print.default, *315*
print.Desc (Desc), 169
print.DunnTest (DunnTest), 206
print.Freq (Freq), 238
print.HoeffD (HoeffD), 275

print.htest, *656*, *661*
print.ICC *(ICC)*, 281
print.mtest *(LehmacherTest)*, 314
print.PercTable *(PercTable)*, 399
print.PostHocTest *(PostHocTest)*, 467
print.table, *241*
print.TMod *(TMod)*, 583
printTable2, *401*
prop.table, *239*, *401*
prop.test, *69*, *75*
prop.trend.test, *103*
PseudoR2, *20*, 476
PtInPoly, *17*, 478
pTri *(Triangular)*, 598
PtsToCm *(ConvUnit)*, 134

qBenf, *18*
qBenf *(Benford)*, 61
qExtrVal *(ExtrVal)*, 221
qFrechet *(Frechet)*, 236
qGenExtrVal *(GenExtrVal)*, 242
qGenPareto *(GenPareto)*, 244
qGompertz *(Gompertz)*, 259
qGumbel *(Gumbel)*, 266
qNegWeibull *(RevWeibull)*, 497
qqline, *295*, *453*, *454*
qqnorm, *147*, *295*, *319*, *398*, *454*, *523*
qqplot, *454*
qRevGumbel *(pRevGumbel)*, 474
qRevGumbelExp *(pRevGumbel)*, 474
qRevWeibull *(RevWeibull)*, 497
qTri *(Triangular)*, 598
Quantile, *20*, *288*, *289*, *351*, 479, *481*
quantile, *156*, *289*, *352*, *453*, *480*, *481*
quantile(x, probs =
        c(.05,.10,.25,.5,.75,.9,.95),
        na.rm = TRUE), *175*
QuantileCI, *480*, 480
Quarter, *21*
Quarter *(Date Functions)*, 164
Quot, 482

RadToDeg, *16*
RadToDeg *(DegToRad)*, 167
Random, *501*
Range, *19*, 483, *605*
range, *484*
Rank, *15*, *398*, 484
rank, *220*, *398*, *485*
rank.test, *58*
RBAL *(PMT)*, 463
rBenf, *18*
rBenf *(Benford)*, 61
rbind, *28*, *39*
rcorr, *276*
read.table, *146*, *580*
read_spss, *486*
read_table, *193*
ReadSPSS, 486
readxl, *193*
Recode, *15*, 486, *492*, *496*
rect, *66*
Recycle, *23*, 488
regexpr, *561*, *566*, *567*, *570*, *571*
regmatches, *561*
regular expression, *544*, *569*
RelRisk, *19*, *128*, *382*, 489
Rename, *15*, 491, *520*
rename, *492*
reorder, *493*
reorder.factor, *16*, 492
rep, *489*, *612*, *622*
replace, *657*
replicate, *489*
reshape, *586*
resid, *348*
Rev, *15*, *381*, *490*, 494
rev, *495*
RevCode, 495
RevWeibull, 497
rExtrVal, *383*
rExtrVal *(ExtrVal)*, 221
rFrechet, *243*, *267*, *498*
rFrechet *(Frechet)*, 236
rgb, *362*, *579*
rgb2hsv, *119*, *120*
RgbToCmy, 498
RgbToCol, *16*, *121*, *499*, 499
RgbToHex, *120*
RgbToHex *(ColToHex)*, 118
RgbToLong, *16*
RgbToLong *(RgbToCol)*, 499
rGenExtrVal, *222*, *237*, *245*, *267*, *383*, *498*
rGenExtrVal *(GenExtrVal)*, 242
rGenPareto *(GenPareto)*, 244
rGompertz *(Gompertz)*, 259
rGumbel, *237*, *243*, *498*

rGumbel (Gumbel), 266
rle, *511*
RMSE, *20*
RMSE (Measures of Accuracy), 346
RndPairs, *15*, 500
RndWord, *15*
RndWord (RndPairs), 500
rNegWeibull (RevWeibull), 497
rnorm, *501*
RobScale, *19*, *324*, 501, *605*, *627*
RomanToInt, *16*, 502
rOrder, *222*
rOrder (Order), 382
Rosenbluth, *19*, *49*, *255*
Rosenbluth (Herfindahl), 268
Rotate, *17*, 503
roulette, *23*
roulette (Datasets for Simulation), 163
round, *505*
RoundTo, *15*, 504, *505*
rRevGumbel (pRevGumbel), 474
rRevWeibull, *237*, *243*, *267*
rRevWeibull (RevWeibull), 497
RSessionAlive, 506
RSqCI, 507
rSum21, 508
RTempdirAlive (RSessionAlive), 506
rTri (Triangular), 598
rug, *431*
runif, *501*, *508*
runmean, *369*
RunsTest, *20*, *58*, 509

Sample, *163*, 512
sample, *163*, *512*, *513*, *555*
SampleTwins, *15*, 513
sapply, *272*, *354*
save, *514*
SaveAs, 514
scale, *324*, *627*
scan, *394*
scatter.smooth, *322*
ScheffeTest, *21*, *133*, *469*, 515
SD, *20*, *108*, *138*, 517
sd, *351*, *501*, *617*
SDN (SD), 517
Second, *21*
Second (Date Functions), 164
SecToHms, *21*
SecToHms (HmsToSec), 272
SendOutlookMail, 518
Sens, *20*
Sens (Conf), 125
seq, *155*, *495*
SetAlpha, *17*, 519
SetAttr (StripAttr), 562
setdiff, *34*
setequal, *34*
setkey, *485*
SetNames, *16*, *492*, 520, *562*
setNames, *520*
setorder, *485*
Shade, *17*, *449*, 521
shapiro.test, *37*, *147*, *295*, *319*, *398*, *523*
ShapiroFranciaTest, *21*, *147*, *295*, *319*, *398*, 522
SiegelTukeyRank, *20*
SiegelTukeyRank (SiegelTukeyTest), 523
SiegelTukeyTest, *20*, 523
SIGN.test, *528*
SignTest, *20*, *354*, 526
Skew, *19*
Skew (Measures of Shape), 349
Skew(), *175*
Skye, *455*
SLN, *22*, *464*
SLN (Depreciation), 168
Small, *15*
Small (Extremes), 219
SMAPE, *20*
SMAPE (Measures of Accuracy), 346
smooth.spline, *322*, *530*, *531*
SmoothSpline, *322*, 529
Some, *16*, 531
Some numeric checks, 532
somers2, *535*
SomersDelta, *19*, *47*, *125*, *261*, *302*, *310*, 534, *576*
Sort, *15*, 535
sort, *220*, *398*, *495*, *537*, *538*
SortMixed, *15*, 537
SpearmanRho, *20*, *47*, 539
Spec, *20*
Spec (Conf), 125
SphToCart, *16*
SphToCart (CartToPol), 94
spineplot, *418*

splinefun, *50*
split, *34*, *443*, *541, 542*
split.formula, *16*, 541
SplitAt, *16*, 542
SplitPath, *16*, 543
SplitToCol, *16*, 544
SplitToDummy, 545
SpreadOut, *17*, *85*, 546
sprintf, *234*
Stamp, *17*, 547
stars, *414*
stats::mad(), *175*
StdCoef, *20*, 548
Str, *16*, 550
str, *550*
StrAbbr, *16*, 551
StrAlign, *16*, *232*, *234*, 552, *571*
Strata, *20*, *513*, 554
StrCap, *16*, 556
StrChop, *16*, *230*, 557, *566*
StrCountW, *16*, 558
StrDist, *16*, 559, *567*, *570, 571*
StrExtract, *16*, 561
StrExtractBetween *(*StrExtract*)*, 561
strheight, *546*
StripAttr, 562
StrIsNumeric, *16*, 563
StrLeft, *16*, *557*
StrLeft *(*StrLeft, StrRight*)*, 564
StrLeft, StrRight, 564
StrPad, *16*, *234*, *552, 553*, 565
StrPos, *16*, 566
strptime, *166*
StrRev, *16*, 567
StrRight, *16*
StrRight *(*StrLeft, StrRight*)*, 564
StrSpell, 567
StrSplit, 568
strsplit, *542*, *544, 545*, *568, 569*
strtoi, *61*
StrTrim, *16*, *551*, *553*, *564*, 569, *571*
StrTrunc, *16*, *551*, *563*, *567*, *570*, 571
StrVal, *16*, 572
strwrap(), *122*
StuartMaxwellTest, *21*, *67*, *315*, 573
StuartTauC, *19*, *47*, *125*, *261*, *263*, *301, 302*, *310*, *535*, 575
sub, *358*, *567*, *569–571*
substr, *557*, *564*, *567*, *569–571*
sum, *246*
SumCI, 577
summary.dist *(*IsEuclid*)*, 291
sunflowerplot, *412*
sweep, *137*, *324*
SYD, *22*
SYD *(*Depreciation*)*, 168
symbol, *411*
symbols, *412*
symnum, *234*
Sys.getenv, *506*
Sys.setlocale, *42*, *234*
SysInfo, *23*, 578

t, *490*
t.test, *342*, *344*, *404*, *528*, *603*, *656*, *661*
table, *44, 45*, *96*, *110, 111*, *126*, *211*, *238, 239*, *241*, *261, 262*, *300*, *302*, *309*, *381*, *401*, *489*, *534*, *575*, *608*
table2d_summary, *401*
tapply, *190*
tarot, *23*
tarot *(*Datasets for Simulation*)*, 163
terms, *393*
text, *25*, *284*, *548*, *582*
TextContrastColor, *17*, 578
TextToTable, *16*, 580
TheilU, *19*, 581
tiff, *596*
time interval, *338*
times, *273*
Timezone, *21*
Timezone *(*Date Functions*)*, 164
title, *405*, *423*, *438*, *440*, *446*, *582*
TitleRect, *17*, 582
TMod, *22*, 583
Today, *21*
Today *(*Date Functions*)*, 164
ToLong, *16*
ToLong *(*ToLong, ToWide*)*, 585
ToLong, ToWide, 585
TOne, *22*, *182*, *234*, 587, *645*
ToWide, *16*
ToWide *(*ToLong, ToWide*)*, 585
ToWrd, *22*, *251*, *589*, 591, *595, 596*, *632*, *634*, *640, 641*, *643, 644*
ToWrd.TOne, *589*
ToWrd.TwoGroups *(*TwoGroups*)*, 606

ToWrdB, 595, *597*
ToWrdPlot, 596
ToXL, *22*, *518*
ToXL *(*XLView*)*, 652
Triangular, 598
Trim, *15*, 601
trunc, *236*, *505*
truncString, *571*
ts, *405*, *441*
TschuprowT, *19*
TschuprowT *(*Association measures*)*, 44
TTestA, *20*, 602
TukeyBiweight, *19*, 604
TukeyHSD, *469*, *516*
TwoGroups, 606

UncertCoef, *19*, *47*, *125*, *261*, *263*, *301*, *302*, *535*, *576*, 607
unclass, *40*
unemployment, *209*
Uniform, *600*
union, *34*
unique, *34*
uniroot, *71*, *342*, *381*, *471*, *610*
UnirootAll, *379*, 609
Unit, *15*
Unit *(*Label, Unit*)*, 307
Unit<- *(*Label, Unit*)*, 307
unname, *562*
Untable, *15*, *113*, 611
Unwhich, *15*, 613
utils::str(), *31*

VanWaerdenTest, 614
Var, *20*, *138*, 616, *618*
var, *110*, *332*, *517*
var.test, *317*, *621*
VarCI, *19*, *342*, *345*, 617
varclus, *276*
VarN *(*Var*)*, 616
VarTest, *20*, *618*, 619
VarX *(*EX*)*, 217
vcov, *622*
VecRot, *15*, 621
VecShift, *15*
VecShift *(*VecRot*)*, 621
VIF, *20*, 622
Vigenere, *15*, 624
violinplot, *460*
VonNeumannTest, *21*, 625

wages, *209*
wdConst, 626
Week, *21*
Week *(*Date Functions*)*, 164
Weekday, *21*, *234*
Weekday *(*Date Functions*)*, 164
weekdays, *145*
Weibull, *474*
weighted.mean, *339*
which, *15*, *100*, *613*
wilcox.test, *132*, *208*, *274*, *354*, *367*, *524*, *525*, *528*
Winsorize, *14*, *601*, 627
winsorize, *627*
WithOptions, 628
WoolfTest, *21*, *86*, *315*, 629
WrdBookmark, 630
WrdCaption, *22*, *592*, 632, *641*
WrdCellRange, *22*, 633, *637*
WrdDeleteBookmark, *22*
WrdDeleteBookmark *(*WrdBookmark*)*, 630
WrdFont, *22*, *631*, 634, *637*
WrdFont<- *(*WrdFont*)*, 634
WrdFormatCells, *22*, 635
WrdGoto, *22*
WrdGoto *(*WrdBookmark*)*, 630
WrdInsertBookmark, *22*, *596*, *597*
WrdInsertBookmark *(*WrdBookmark*)*, 630
WrdKill, *22*
WrdKill *(*GetNewWrd*)*, 250
WrdMergeCells, *22*, 636
WrdOpenFile *(*WrdSaveAs*)*, 642
WrdPageBreak, 637
WrdParagraphFormat, *22*, 638
WrdParagraphFormat<- *(*WrdParagraphFormat*)*, 638
WrdPlot, *22*, *473*, *631*, *632*, *634*, *637*, *640*, 640, *643*
WrdSaveAs, *22*, 642
WrdStyle, *22*, 643
WrdStyle<- *(*WrdStyle*)*, 643
WrdTable, *22*, *251*, *589*, *633*, *635–637*, 644, *645*, *647*
WrdTableBorders, *22*, 645
WrdTableHeading, 646
WrdUpdateBookmark, *22*
WrdUpdateBookmark *(*WrdBookmark*)*, 630

write.table, *652*

xlConst *(wdConst)*, 626
XLCurrReg *(XLGetRange)*, 648
XLDateToPOSIXct, *22*, 647, *650*
XLGetRange, *22*, *253*, *648*, 648, *653*
XLGetWorkbook, *22*, *253*, *653*
XLGetWorkbook *(XLGetRange)*, 648
XLKill, *22*
XLKill *(XLView)*, 652
XLNamedReg *(XLGetRange)*, 648
XLSaveAs, 651
XLView, *22*, *253*, *650, 651*, 652
xtabs, *40*, *612*
xtfrm, *137*
xy.coords, *247*, *411*, *504*

Year, *21*, *32*, *42*, *290*, *659*
Year *(Date Functions)*, 164
YearDay, *21*
YearDay *(Date Functions)*, 164
YearDays *(Date Functions)*, 164
YearMonth, *21*
YearMonth *(Date Functions)*, 164
YearWeek *(Date Functions)*, 164
YTM, *22*
YTM *(NPV)*, 378
YuenTTest, *21*, 654
YuleQ, *19*, *261*
YuleQ *(Association measures)*, 44
YuleY, *19*
YuleY *(Association measures)*, 44

ZeroIfNA, *15*, 657
Zodiac, *21*, 658
ZTest, *20*, *528*, 659