Package 'NCmisc'

July 21, 2025

Type Package

Version 1.2.0

Date 2022-10-14

Title Miscellaneous Functions for Creating Adaptive Functions and Scripts

Author Nicholas Cooper

Maintainer Nicholas Cooper <njcooper@gmx.co.uk>

Depends R (>= 3.10), grDevices, graphics, stats, utils

Imports tools, methods

Suggests KernSmooth, Matrix

Description A set of handy functions. Includes a versatile one line progress bar, one line function timer with detailed output, time delay function, text histogram, object preview, CRAN package search, simpler package installer, Linux command install check, a flexible Mode function, top function, simulation of correlated data, and more.

License GPL (>= 2)

NeedsCompilation no

Repository CRAN

Date/Publication 2022-10-17 09:15:22 UTC

Contents

`NCmisc-package` *Miscellaneous Functions for Creating Adaptive Functions and Scripts*

Description

A set of handy functions. Includes a versatile one line progress bar, one line function timer with detailed output, time delay function, text histogram, object preview, CRAN package search, simpler package installer, Linux command install check, a flexible Mode function, top function, simulation of correlated data, and more.

Details

Package:	NCmisc
Type:	Package
Version:	1.2.0
Date:	2022-10-14
License:	GPL (>= 2)

A package of general purpose functions that might save time or help tidy up code. Some of these functions are similar to existing functions but are simpler to use or have more features (e.g, timeit and loop.tracker reduce an initialisation, 'during' and close three-line call structure, to a single function call. Also, some of these functions are useful for building packages and pipelines, for instance: Header(), to provide strong visual deliniation between procedures in console output, by an ascii bordered heading; loop.tracker() to track the progress of loops (called with only 1 line of code), with the option to periodically backup a key object during the loop; estimate.memory() to determine whether the object may exceed some threshold before creating it, timeit(), a one line wrapper for proftools which gives a detailed breakdown of time taken, and time within each function called during a procedure; and check.linux.install() to verify installation status of terminal commands before using system(), top() to examine current memory and CPU usage [using the system 'top' command]. prv() is useful for debugging as it allows a detailed preview of objects, and is as easy as placing print statements within loops/functions but gives more information, and gives compact output for large objects. For testing sim.cor() provides a simple way to simulate a correlated data matrix, as often this is more realistic than completely random data. Otherwise summarise.r.datasets gives a list of all available datasets and their structure and dimensionality.

List of key functions:

- *check.linux.install* Check whether a given system command is installed (e.g, bash)
- *comma.list* Nicely format output lists with comma separation and length control
- *comify* Function to add commas for large numbers

- *cor.with* simulate a variable with a specified correlation to an existing variable
- *Dim* same as dim() function but works for more objects, including vectors
- *dup.pairs* Obtain an ordered index of all instances of values with duplicates
- *estimate.memory* Estimate the memory required for an object
- *exists.not.function* same as exists() function but ignores functions
- *extend.pc* Extend an interval by percentage
- *fakeLines* Create randomized lines of text for testing
- *force.percentage* Force argument to be a decimal percentage
- *force.scalar* Force argument to be a scalar
- *get.distinct.cols* Return up to 22 distinct colours
- *getRepositories* Return list of available repositories
- *has.method* Determine whether a function can be applied to an S4 class/object
- *headl* A good way to preview large lists
- *Header* Print heading text with a border
- *is.vec.logical* Test whether vector is logical independent of type
- *is.vec.numeric* Test whether vector is numeric independent of type
- *list.functions.in.file* Show all functions used in an R script file, by package
- *list.to.env* Inserts new variables in current environment from a named list
- *loess.scatter* Draw a scatterplot with a fit line
- *loop.tracker* Creates a progess bar within a loop with only 1 line
- *Mode* Find the mode(s) of a vector
- *must.use.package* Do everything possible to load an R package
- *narm* Return an object with missing values removed
- *nearest.to* Similar to base match function but picks nearest instead of exact match
- *Numerify* Convert only suitable columns to numeric format in data.frame
- *out.of* Simplify outputting fractions/percentages
- *p.to.Z* Convert p-values to Z-scores
- *packages.loaded* quietly test whether packages are loaded without using require
- *pad.left* Print a vector with appropriate padding so each has equal char length
- *pctile* Find data thresholds corresponding to percentiles
- *ppa* Posterior probability for p-values
- *preview* same as prv, but enter arguments as strings
- *prv.large* tidy representation for large matrices/data.frames
- *prv* compact preview of objects (more complete than 'print')
- *replace.missing.df* replace missing values in data.frame automatically
- *Rfile.index* Create an index file for an R function file
- *rmv.names* Remove names from object

- *rmv.spc* Remove leading and trailing spaces (or other character)
- *search.cran* Search all CRAN packages for those containing keyword(s)
- *sim.cor* simulate a correlated dataset
- *simple.date* generate a string with compact summary of date/time
- *spc* Print a character a specified number of times
- *standardize* Convert a numeric vector to Z-scores
- *Substitute* multivariable version of substitute (base)
- *summary2* Extension of base:summary that adds SD, SE and keeps names fixed and cleaner
- *summarise.r.datasets* show and summarise all available example datasets
- *table2d* Extension of base:table that forces fixed rows and columns
- *textogram* Make an ascii histogram in the console
- *timeit* Times an expression, with breakdown of time spent in functions
- *toheader* Return a string with each first letter of each word in upper case
- *top* report on CPU and memory usage, overall or by process
- *Unlist* Unlist a list, starting only from a set depth
- *wait* Wait for a period of time
- *which.outlier* Return indexes of univariate outliers
- *Z.to.p* Convert Z-scores to p-values

Author(s)

Nicholas Cooper

Maintainer: Nicholas Cooper <njcooper@gmx.co.uk>

See Also

reader ~~

Examples

```
#text histogram suited to working from a console without GUI graphics
textogram(rnorm(10000),range=c(-3,3))
# wait 0.2 seconds
wait(0.2,silent=FALSE)
# see whether a system command is installed
check.linux.install("sed")
# a nice progress bar
max <- 100; for (cc in 1:max) { loop.tracker(cc,max); wait(0.004,"s") }
# nice header
Header(c("SPACE","The final frontier"))
# memory req'd for proposed or actual object
estimate.memory(matrix(rnorm(100),nrow=10))
# a mode function (there isn't one included as part of base)
Mode(c(1,2,3,3,4,4,4))
# search for packages containing text, eg, 'misc'
```

```
search.cran("misc", repos="http://cran.ma.imperial.ac.uk/")
# simulate a correlated dataset
corDat <- sim.cor(200,5)
cor(corDat) # show correlation matrix
prv(corDat) # show compact preview of matrix
# Dim() versus dim()
Dim(1:10); dim(1:10)
# find nearest match in a vector:
nearest.to(1:100, 50.5)
```

`check.linux.install` *Check whether a given system command is installed (e.g, bash)*

Description

Tests whether a command is installed and callable by system(). Will return a warning if run on windows when linux.more=TRUE

Usage

```
check.linux.install(cmd = c("plink", "perl", "sed"), linux.mode = FALSE)
```

Arguments

`cmd`	character vector of commands to test
`linux.mode`	logical, alternate way of command testing that only works on linux and mac OS X, to turn this on, set to TRUE.

Value

returns true or false for each command in 'cmd'

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

Examples

```
check.linux.install("R") # should be standard
check.linux.install(c("perl","sed","fake-cmd"))
```

comify *Function to add commas for large numbers*

Description

Often for nice presentation of genomic locations it is helpful to insert commas every 3 digits when numbers are large. This function makes it simple and allows specification of digits if a decimal number is in use.

Usage

```
comify(x, digits = 2)
```

Arguments

x	a vector of numbers, either as character, integer or numeric form
digits	integer, if decimal numbers are in use, how many digits to display, same as input to base::round()

Value

returns a character vector with commas inserted every 3 digits

Examples

```
comify("23432")
comify(x=c(1,25,306,999,1000,43434,732454,65372345326))
comify(23432.123456)
comify(23432.123456,digits=0)
```

comma.list *Print out comma separated list of values in X, truncating if many (good for error messages)*

Description

Often for nice presentation of error messages you wish to display a list of values. This adds commas between entries and will truncate the list above a length of 50 items with an ellipsis. Very simple but convenient function.

Usage

```
comma.list(X)
```

Arguments

X	a vector to be displayed

Value

string with entries separated by commas, and if long, entries skipped indicated by an ellipsis.

Examples

```
comma.list(1:100)
cat("The following entries were ignored: ", comma.list(c(1,7,10:14)), "\n")
```

`cor.with` *Simulate a correlated variable*

Description

Simulate a variable correlated at level 'r' with cector x (of the same length). Can either 'preserve' the mean and standard-deviation, leave standardizeed, or select new mean 'mn' and standard deviation 'st'.

Usage

```
cor.with(x, r = 0.5, preserve = FALSE, mn = NA, st = NA)
```

Arguments

`x`	existing variable, to which you want to simulate a new correlated variable
`r`	the 'expected' correlation you want to target (randomness will mean that the actual correlation will vary around this value)
`preserve`	logical, whether to preserve the same mean and standard deviation(SD) as x, for the new variable
`mn`	optional, set the mean for the new simulated variable [must also set st if using this]
`st`	optional, set the SD for the new simulated variable [must also set mn if using this]

Value

return the new variable with an expected correlation of 'r' with x

Author(s)

Nicholas Cooper

References

http://www.uvm.edu/~dhowell/StatPages/More_Stuff/CorrGen.html

See Also

sim.cor

Examples

```
X <- rnorm(10,100,14)
cor.with(X,r=.5) # create a variable correlated .5 with X
cor(X,cor.with(X)) # check the actual correlation
# some variability in the actual correlation, so run 1000 times:
print(mean(replicate(1000,{cor(X,cor.with(X))})))
cor.with(X,preserve=TRUE) # preserve original mean and standard deviation
X[c(4,10)] <- NA # works fine with NAs, but new var will have same missing
cor.with(X,mn=50,st=2) # specify new mean and standard deviation
```

Dim *A more general dimension function*

Description

A more general 'dim' function. For arrays simply calls the dim() function, but for other data types, tries to provide an equivalent, for instance will call length(x) for vectors, and will recursively report dims for lists, and will attempt something sensible for other datatypes.

Usage

```
Dim(x, cat.lists = TRUE)
```

Arguments

x	the object to find the dimension for
cat.lists	logical, for lists, TRUE will concatenate the dimesions to a single string, or FALSE will return the sizes as a list of the same structure as the original.

Value

dimension(s) of the object

See Also

prv, preview

Examples

```
# create variables of different types to show output styles #
Dim(193)
Dim(1:10)
testvar <- matrix(rnorm(100),nrow=25)
Dim(matrix(rnorm(100),nrow=25))
Dim(list(first="test",second=testvar,third=100:110))
Dim(list(first="test",second=testvar,third=100:110),FALSE)
```

dup.pairs *Obtain an index of all instances of values with duplicates (ordered)*

Description

The standard 'duplicated' function, called with which(duplicated(x)) will only return the indexes of the extra values, not the first instances. For instance in the sequence: A,B,A,C,D,B,E; it would return: 3,6. This function will also return the first instances, so in this example would give: 1,3,2,6 [note it will also be ordered]. This index can be helpful for diagnosis if duplicates are unexpected, for instance in a data.frame, and you wish to compare the differences between the rows with the duplicate values occuring. Also, duplicate values are sorted to be together in the listing, which can help for manual troubleshooting of undesired duplicates.

Usage

```
dup.pairs(x)
```

Arguments

x	a vector that you wish to extract duplicates from

Value

vector of indices of which values in 'x' are duplicates (including the first observed value in pairs, or sets of >2), ordered by set, then by appearance in x.

Examples

```
set <- c(1,1,2,2,3,4,5,6,2,2,2,2,12,1,3,3,1)
dup.pairs(set) # shows the indexes (ordered) of duplicated values
set[dup.pairs(set)] # shows the values that were duplicated (only 1's, 2's and 3's)
```

estimate.memory *Estimate the memory required for an object.*

Description

Can enter an existing object or just the dimensions or total length of a proposed object. The estimate is based on the object being of numeric type. Integers use half the space of numeric, raw() use 1/8th of the space. Factors and characters can vary, although factors will always use less than numeric, and character variables may easily use up to twice as much depending on the length [nchar()] of each element.

Usage

```
estimate.memory(
  dat,
  integer = FALSE,
  raw = FALSE,
  unit = c("gb", "mb", "kb", "b"),
  add.unit = FALSE
)
```

Arguments

dat	either a vector/matrix/dataframe object, or else up to 10 dimensions of such an object, or a potential object, i.e; c(nrow,ncol). If entering an object directly, you can leave out the 'integer' and 'raw' arguments as these will be detected from the object type. Any set of dimensions >10 will be assumed to be a vector, so if you have such an object, better to submit the total product [base::prod()].
integer	if the object or potential object is integer or logical type, set this argument to TRUE, if this is TRUE, the parameter 'RAW' will be ignored; integer and logical types use 1/2 of the memory of numeric types
raw	if the object or potential object is of 'raw' type, set this argument to TRUE, note that if 'integer' is TRUE, this parameter 'RAW' will be ignored; raw types use 1/8 of the memory of numeric types
unit	the storage units to use for the result, ie, "gb", "mb","kb", "b" for gigabytes, megabytes, kilobytes, or bytes respectively.
add.unit	logical, whether to append the unit being used to the result, making the result character type instead of numeric.

Value

returns the minimum memory requirement to store and object of the specified size, as a numeric scalar, in gigabytes (default) or else using the units specified by 'unit', and if add.unit = TRUE, then the result will be character type instead of numeric, with the units appended.

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

Examples

```
myMatrix <- matrix(rnorm(100),nrow=10)
myVec <- sample(1:1000)
estimate.memory(myMatrix,unit="bytes") # enter a matrix object
estimate.memory(myVec,unit="kb" ,add.unit=TRUE) # enter a vector object
estimate.memory(c(10,10,10,10,10),unit="kb") # 5 dimensional array
estimate.memory(c(10^6,10^4), add.unit=TRUE) # large matrix
estimate.memory(5.4*10^8, add.unit=TRUE)  # entering argument as # total cells, rather than dims
estimate.memory(5.4*10^8, integer=TRUE, add.unit=TRUE)
estimate.memory(5.4*10^8, raw=TRUE, add.unit=TRUE)
estimate.memory(5.4*10^8, TRUE, TRUE, add.unit=TRUE) #  'integer' overrides 'raw'
```

exists.not.function	*Does object exist ignoring functions The exists() function can tell you whether an object exists at all, or whether an object exists with a certain type, but it can be useful to know whether an object exists as genuine data (and not a function) which can be important when a variable or object is accidently or intentionally given the same name as a function. This function usually returns a logical value as to the existence of the object (ignoring functions) but can also be set to return the non-function type if the object exists.*

Description

Does object exist ignoring functions

The exists() function can tell you whether an object exists at all, or whether an object exists with a certain type, but it can be useful to know whether an object exists as genuine data (and not a function) which can be important when a variable or object is accidently or intentionally given the same name as a function. This function usually returns a logical value as to the existence of the object (ignoring functions) but can also be set to return the non-function type if the object exists.

Usage

```
exists.not.function(x, ret.type = FALSE)
```

Arguments

x	the object name to search for
ret.type	logical, if TRUE then will return the objects' type (if it exists) rather than TRUE or FALSE. If the object doesn't exist the empty string will be returned as the type.

Value

logical, whether non-function object exists, or else the type if ret.type=TRUE

Author(s)

Nicholas Cooper

Examples

```
x <- "test"
# the standard exists function, for all modes, correct mode, and other modes:
exists("x")
exists("x",mode="character")
exists("x",mode="numeric")
# standard case for a non-function variable
exists.not.function("x",TRUE)
# compare results for a non-existent variable
```

```
exists("aVarNotSeen")
exists.not.function("aVarNotSeen")
# compare results for variable that is a function
exists("mean")
exists.not.function("mean")
# define a variable with same name as a function
mean <- 1.4
# exists.not.function returns the type of the variable ignoring the function of the same name
exists.not.function("mean",TRUE)
exists("mean",mode="function")
exists("mean",mode="numeric")
```

`extend.pc` *Extend an interval by percentage*

Description

For various reasons, such as applying windows, setting custom range limits for plots, it may be desirable to extend an interval by a certain percentage.

Usage

```
extend.pc(X, pc = 0.5, pos = TRUE, neg = TRUE, swap = FALSE)
```

Arguments

X	a numeric range, should be length 2. If a longer numeric, will be coerced with range()
pc	percentage by which to extend X, can be entered in either percentage style: 0<pc<1; or 1<pc<100
pos	logical, if TRUE, make an extension in the positive direction
neg	logical, if TRUE, make an extension in the negative direction
swap	logical, if TRUE, flip the extension directions if X[2]<X[1], ie, not in numerical order

Examples

```
extend.pc(c(2,10),0.25) # extend X symmetrically
extend.pc(c(2:10),0.25) # extend the range of X
# the following 3 examples extend X by 1% only in the 'positive' direction
extend.pc(c(25000,55000),.01,neg=FALSE) # standard positive extension
extend.pc(c(55000,25000),.01,neg=FALSE) # ranges in reverse order, not swapped
extend.pc(c(55000,25000),.01,neg=FALSE,swap=TRUE) # ranges in reverse order, swapped
```

`fakeLines` *Create fake text for testing purposes*

Description

Returns randomized input as if reading lines from a file, like 'readLines()' Can be used to test i/o functions, robustness.

Usage

```
fakeLines(
  max.lines = 10,
  max.chars = 100,
  pc.space = 0.35,
  delim = " ",
  can.null = TRUE
)
```

Arguments

`max.lines`	maxmimum number of fake lines to read
`max.chars`	maximum number of characters per line
`pc.space`	percentage of randomly generated characters that should be a delimiter
`delim`	what should the simulated delimiter be, e.g, a space, comma etc. If you wish not to include such either set the delimiter as "", or set pc.space=0.
`can.null`	whether with probability 1/max.lines to return NULL instead of any lines of text, which simulates an empty file, which for testing purposes you may want to be able to handle

Value

a vector of character entries up 'max.chars' long, or sometimes only NULL if can.null=TRUE

Author(s)

Nicholas Cooper

Examples

```
fakeLines() # should produce between zero and ten lines of random text, 35% of which are spaces
```

`file.split` *Split a text file into multiple parts*

Description

Wrapper for the bash command 'split' that can separate a text file into multiple roughly equal sized parts. This function removes the need to remember syntax and suffixes of the bash command

Usage

```
file.split(
  fn,
  size = 50000,
  same.dir = FALSE,
  verbose = TRUE,
  suf = "part",
  win = TRUE
)
```

Arguments

`fn`	character, file name of the text file to split, if the file is an imcompatible format the linux command should return an error message to the console
`size`	integer, the maximum number of lines for the split parts of the file produced
`same.dir`	logical, whether the resulting files should be moved to the same directory as the original file, or simply left in the working directory [getwd()]
`verbose`	logical, whether to report the resulting file names to the console
`suf`	character, suffix for the split files, default is 'part', the original file extension will be appended after this suffix
`win`	logical, set to FALSE if running a standard windows setup (cmd.ext), and the file split will run natively in R. Set to TRUE if you have a unix-alike command system, such as CygWin, sh.exe, csh.exe, tsh.exe, running, and this will then check to see whether the POSIX 'split' command is present (this provides a speed advantage). If in doubt, windows users can always set win=TRUE; the only case where this will cause an issue is if there is a different command installed with the same name (i.e, 'split').

Value

returns the list of file names produced (including path)

Author(s)

Nicholas Cooper

Examples

```
orig.dir <- getwd(); setwd(tempdir()); # move to temporary dir
file.name <- "myfile.txt"
writeLines(fakeLines(max.lines=1000),con=file.name)
new.files <- file.split(file.name,size=50)
unlink(new.files); unlink(file.name)
setwd(orig.dir) # reset working dir to original
```

`force.percentage` *Force argument to be a percentage with length one*

Description

Sometimes it is nice to be able to take a percentage as an argument and not have to specify whether it should be entered as a number between 0 and 100, e.g, 50 = 50 than 1 and less than 100 will be divided by 100. Anything outside 0,100 will be set to 0,100 respectively.

Usage

```
force.percentage(x, default = 0.5)
```

Arguments

`x`	the object to ensure is a oercentage
`default`	the value to revert to if the format of x is illegal

Value

the object x if already legal, first element if a vector, the min or max value if x is outside the specified bounds, or the value of default otherwise

See Also

`force.scalar`

Examples

```
# create variables of different types to show output styles #
force.percentage(45)
force.percentage(450)
force.percentage(.45)
force.percentage(-45)
force.percentage("twenty")
force.percentage(NA,default=0.25)
```

`force.scalar` *Force argument to be a numeric type with length one*

Description

Sometimes arguments must be numeric, scalar and within a certain range. Rather than using many if statements, this will do everything possible to coerce input to a scalar, failing that will replace with a default value. Can also provide a maximum and minimum range that the result must lie within.

Usage

```
force.scalar(x, default = 1, min = -10^10, max = 10^10)
```

Arguments

`x`	the object to ensure is a scalar
`default`	the value to revert to if the format of x is illegal
`min`	a lower bound for the output, anything below this is set to min
`max`	an upper bound for the output, anything above this is set to max

Value

the object x if already legal, first element if a vector, the min or max value if x is outside the specified bounds, or the value of default otherwise

See Also

`force.percentage`

Examples

```
force.scalar(1.5)
force.scalar(NULL,default=.5)
force.scalar(NA,default=.4,min=5,max=10) # default is outside range!
force.scalar(rnorm(1000))
force.scalar(101,max=50)
force.scalar(list(0.4,1,2,3,4,"test"))
force.scalar(data.frame(test=c(1,2,3),name=c("test","me","few")))
force.scalar(Inf)
```

get.distinct.cols *Return up to 22 distinct colours.*

Description

Useful if you want to colour 22 autosomes, etc, because most R colour palettes only provide 12 or fewer colours, or else provide, a gradient which is not distinguishable for discrete categories. Manually curated so the most similar colours aren't side by side.

Usage

```
get.distinct.cols(n = 22)
```

Arguments

n	number of unique colours to return

Value

returns vector of n colours

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

Examples

```
get.distinct.cols(10)
plot(1:22,pch=19,col=get.distinct.cols(22))
```

getRepositories *Detect all available R repositories.*

Description

In addition to the default CRAN repository, there are other repositories such as R-Forge, Omegahat, and bioConductor (which is split in to software, annotation, experiments and extras). This function allows you to retrieve which are available. This function complements (and takes code from) utils::setRepositories(), which will just set, not return which are available, but see there for more information about how this works. Detecting the available repositories can be useful to precede a call to setRepositories, and allows you to utilise these repositories without calling setRepositories (which is hard to reverse). This function can be used to expand the search space of the function search.cran() to include bioconductor packages.

Usage

```
getRepositories(ind = NULL, table = FALSE)
```

Arguments

ind	index, same as for 'setRepositories', if NULL this function returns all available repositories, or if an index, returns a subset.
table	logical, if TRUE, return a table of information, else just return the URLs, which are the required input for the 'repos' argument for relevant functions, e.g, available.packages() or search.cran()

Value

list of repositories with URLS, note that it is the URL that works best for use for passing a value for 'repos' to various functions.

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

Examples

```
repos <- "http://cran.ma.imperial.ac.uk/" # OR: repos <- getOption("repos")
getRepositories(table=TRUE) # shows all available
getRepositories(2:5,FALSE) # returns index for all bioconductor repositories (on my system at least)
# does not find this bioconductor package on CRAN
## not run # search.cran("genoset",repos=getRepositories(1))
# should now, because all repositories are used
## not run # search.cran("genoset",repos=getRepositories())
```

---

has.method *Determine whether a function can be applied to an S4 class/object*

Description

Wrapper for 'showMethods', allows easy testing whether a function (can be specified as a string, or the actual function itself (FUN)) can be applied to a specific object or class of objects (CLASS)

Usage

```
has.method(FUN, CLASS, false.if.error = FALSE, ...)
```

Arguments

FUN	the function to test, can be specified as a string, or the actual function itself
CLASS	a specific object or a class of objects specified by a string, e.g, "GRanges"
false.if.error	logical, the default value is FALSE, in which case an error is returned when FUN is not an S4 generic function. If this parameter is set to TRUE, 'FALSE' will be returned with a warning instead of an error.
...	additional arguments to showMethods(), e.g, 'where' to specify the environment

Value

returns logical (TRUE/FALSE), or if the function is not S4 will return an error, although this could potentially be because the function's package has not been loaded.

Examples

```
require(Matrix); require(methods)
has.method("t","dgeMatrix") # t() is the transpose method for a dgeMatrix object
has.method(t,"dgeMatrix") # also works without quotes for the method
m.example <- as(matrix(rnorm(100),ncol=5),"dgeMatrix")
has.method(t, m.example) # works with an instance of an object type too
has.method("band", m.example) # band is a function for a 'denseMatrix' but not 'dgeMatrix'
## not run # has.method("notAFunction","GRanges") # should return error
## not run # has.method("notAFunction","GRanges",TRUE) # should return FALSE and a warning
```

Header *Print heading text with a border.*

Description

Makes highly visible headings, can separately horizontal, vertical and corner characters

Usage

```
Header(txt, h = "=", v = h, corner = h, align = "center")
```

Arguments

txt	The text to display in the centre
h	the ascii character to use on the horizontal sections of the border, and used for v,corner too if not specified separately
v	the character to use on vertical sections of the border
corner	the character to use on corner sections of the border
align	alignment of the writing, when there are multiple lines, e.g, "right", "left", "centre"/"center"

Value

returns nothing, simply prints the heading to the console

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

Examples

```
Header("Section 1")
Header("Section 1",h="-",v="|",corner="*")
Header(c("SPACE","The final frontier"))
Header(c("MY SCRIPT","Part 1"),align="left",h=".")
```

---

headl *A good way to preview large lists.*

Description

An alternative to head(list) which allows limiting of large list components in the console display

Usage

```
headl(x, n = 6, skip = 20, skip2 = 10, ind = "", ind2 = "  ")
```

Arguments

x	a list to preview
n	The number of values to display for the deepest nodes of the list
skip	number of first level elements to display before skipping the remainder
skip2	number of subsequent level elements to display before skipping the remainder
ind	indent character for first level elements
ind2	indent character for subsequent level elements

Value

prints truncated preview of a large list

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

Examples

```
sub1 <- list(list(1:100),list(2:101),list(101:200),list(201:300),list(301:400))
big.list <- list(sub1,sub1,sub1,sub1,sub1,sub1)
headl(sub1)
headl(big.list,skip=2)
```

`is.vec.logical` *Determine robustly whether a vector contains logical data*

Description

This is an improvement on base:is.logical because data may be encoded as a different type (e.g, string, "TRUE", "FALSE") especially if imported from a file. This does not include logical vectors coded as 0,1; such will return FALSE with this function.

Usage

```
is.vec.logical(x, thresh = 0.9)
```

Arguments

`x`	a vector to check for logical status
`thresh`	threshold to decide that a variable is logical. NA values will be ignored in the test. Then it looks at the proportion of values that are successfully coerced to logical without giving 'NA'. If this threshold is 0.9, then any column where at least 90 converted to logical type, will return TRUE for this function call.

Value

returns a logical TRUE or FALSE for the logical status of x.

Author(s)

Nicholas Cooper

Examples

```
numeric <- 1:10
string <- paste("one", "two", "three", "four")
logic1 <- c(TRUE,FALSE,FALSE,TRUE,FALSE,NA)
logic2 <- c("TRUE", "FALSE", "TRUE", NA, "TRUE", NA, NA, NA)
logic3 <- c("True", "False", "True", "False")
numlogic <- c(0,1,0,0,0,1,1,1,0)
is.vec.logical(numeric)
is.vec.logical(string)
is.vec.logical(logic1)
is.vec.logical(logic2)
is.vec.logical(logic3)
is.vec.logical(numlogic)
```

`is.vec.numeric` *Determine robustly whether a vector contains numeric data*

Description

This is an improvement on base:is.numeric because data may be encoded as a different type (e.g, string) especially if imported from a file.

Usage

```
is.vec.numeric(x, logical.is.numeric = FALSE, thresh = 0.9)
```

Arguments

`x`	a vector to check for numeric status
`logical.is.numeric`	by default this is FALSE, which means logical vectors will return FALSE to being numeric. If set to TRUE, then a variable will get a return value of TRUE if it is based on numbers or appears to be of 'logical' type.
`thresh`	threshold to decide that a variable is numeric. NA values will be ignored in the test. Then it looks at the proportion of values that are successfully coerced to numeric without giving 'NA'. If this threshold is 0.9, then any column where at least 90 converted to numeric type, will return TRUE for this function call.

Value

returns a logical TRUE or FALSE for the numeric status of x.

Author(s)

Nicholas Cooper

Examples

```
numeric1 <- 1:10
numeric2 <- paste(1:10)
string <- paste("one", "two", "three", "four")
logic1 <- c(TRUE,FALSE,FALSE,TRUE,FALSE,NA)
numericish <- paste(c(NA, NA, 6:10, "5|6", "7|8", 1))
is.vec.numeric(numeric1)
is.vec.numeric(numeric2)
is.vec.numeric(string)
is.vec.numeric(logic1)
is.vec.numeric(logic1, logical.is.numeric=TRUE)
is.vec.numeric(numericish)
is.vec.numeric(numericish, thresh=0.7)
```

`list.functions.in.file` *Show all functions used in an R script file, by package*

Description

Parses all functions called by an R script and then lists them by package. Wrapper for 'getParseData'. Inspired by 'hrbrmstr', on StackExchange 3/1/2015. May be of great use for those developing a package to help see what namespace 'importsFrom' calls will be required.

Usage

```
list.functions.in.file(filename, alphabetic = TRUE)
```

Arguments

`filename`	path to an R file containing R code.
`alphabetic`	logical, whether to list functions alphabetically. If FALSE, will list in order of appearance.

Value

Returns a list. Parses all functions called by an R script and then lists them by package. Those from the script itself are listed under '.GlobalEnv' and any functions that may originate from multiple packages have all possibilities listed. Those listed under 'character(0)' are those for which a package could not be found- may be functions within functions, or from packages that aren't loaded.

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

See Also

`Rfile.index`

Examples

```
# not run:  rfile <- file.choose() # choose an R script file with functions
# not run:  list.functions.in.file(rfile)
```

`list.to.env` *Create variables from a list*

Description

Places named objects in a list into the working environment as individual variables. Can be particularly helpful when you want to call a function that produces a list of multiple return variables; this gives a way to access them all at once in the environment from which the function was called.

Usage

```
list.to.env(list)
```

Arguments

`list`	list, with named objects, each element will become a named variable in the current environment

Value

New variables will be added to the current environment. Use with care as any already existing with the same name will be overwritten.

See Also

base::list2env

Examples

```
list.to.env(list(myChar="a string", myNum=1234, myList=list("list within a list",c(1,2,3))))
print(myChar)
print(myNum)
print(myList)
two.arg.return <- function(X) { return(list(Y=X+1,Z=X*10)) }
result <- two.arg.return(11) # function returns list with 2 variables
list.to.env(result)
print(Y); print(Z)
```

`loess.scatter` *Draw a scatterplot with a fit line*

Description

Drawing a fit line usually requires some manual steps requiring several lines of code, such as ensuring the data is sorted by x, and for some functions doesn't contain missing values. This function takes care of these steps and automatically adds a loess fitline, or non-linear fitline. The type of scatter defaults to 'plot', but other scatter plot functions can be specified, such as graphics::smoothScatter(), for example. If 'file' is specifed, will automatically plot to a pdf of that name.

Usage

```
loess.scatter(
  x,
  y,
  file = NULL,
  loess = TRUE,
  span = 0.75,
  scatter = plot,
  ...,
  ylim = NULL,
  return.vectors = FALSE,
  fit.col = "red",
  fit.lwd = 2,
  fit.lty = "solid",
  fit.leg = TRUE,
  fit.r2 = TRUE,
  fast.loess = FALSE
)
```

Arguments

`x`	data for the horizontal axis (independent variable)
`y`	data for the vertical axis (dependent variable)
`file`	file name for pdf export, leave as NULL if simply plotting to the GUI. File extension will be added automatically if missing
`loess`	logical, if TRUE, fit using loess(), else use a polynomial fit
`span`	numeric scalar, argument passed to the 'span' parameter of loess(), see ?loess for details
`scatter`	function, by default is graphics::plot(), but any scatter-plot function of the form F(x,y,...) can be used, for example graphics::smoothScatter().
`...`	further arguments to the plot function specified by 'scatter', e.g, 'main', 'xlab', etc
`ylim`	numeric range for y axis, argument passed to plot(), see ?plot.

return.vectors	logical, if TRUE, do not plot anything, just return the x and y coordinates of the fit line as a list of vectors, x and y.
fit.col	colour of the fit line
fit.lwd	width of the fit line
fit.lty	type of the fit line
fit.leg	whether to include an automatic legend for the fit line (will alter the y-limits to fit)
fit.r2	logical, whether to display r squared of the fit in the fit legend
fast.loess	logical, if TRUE will alter control parameters to make the loess calculation faster, which is useful for datasets with more than 1000 points. Also reduce the value of 'span' to increase speed.

Value

if file is a character argument, plots data x,y to a file, else will generate a plot to the current plotting environment/GUI. The display of the x,y points defaults to 'plot', but alternate scatter plot functions can be specified, such as graphics::smoothScatter() which used density smoothing, for example. Also, another option is to set return.vectors=TRUE, and then the coordinates of the fit line will be returned, and no plot will be produced.

Examples

```
library(NCmisc)
require(KernSmooth)
DD <- sim.cor(1000,4) # create a simulated, correlated dataset
loess.scatter(DD[,3],DD[,4],loess=FALSE,bty="n",pch=".",cex=2)
loess.scatter(DD[,3],DD[,4],scatter=smoothScatter)
xy <- loess.scatter(DD[,3],DD[,4],return.vectors=TRUE)
prv(xy) # preview the vectors produced
```

---

loop.tracker *Creates a progess bar within a loop*

Description

Only requires a single line within a loop to run, in contrast with the built-in tracker which requires a line to initialise, and a line to close. Also has option to backup objects during long loops. Ideal for a loop with a counter such as a for loop. Tracks progress as either percentage of time remaining or by intermittently displaying the estimated number of minutes to go

Usage

```
loop.tracker(
  cc,
  max,
  st.time = NULL,
```

```
  sav.obj = NULL,
  sav.fn = NA,
  sav.freq = 10,
  unit = c("m", "s", "h")[1]
)
```

Arguments

cc	integer, current value of the loop counter
max	integer, final value of the loop counter
st.time	'start time' when using 'time to go' mode, taken from a call to proc.time()
sav.obj	optionally an object to backup during the course of a very long loop, to restore in the event of a crash.
sav.fn	the file name to save 'save.obj'
sav.freq	how often to update 'sav.obj' to file, in terms of percentage of run-time
unit	time units h/m/s if using 'time to go' mode

Value

returns nothing, simply prints progress to the console

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

Examples

```
# simple example with a for-loop
max <- 100; for (cc in 1:max) { loop.tracker(cc,max); wait(0.004,"s") }
#example using the 'time to go' with a while loop
cc <- 0; max <- 10; start <- proc.time()
while(cc < max) { cc <- cc + 1; wait(0.05,"s"); loop.tracker(cc,max,start,unit="s") }
# example with saving an object, and restoring after a crash
X <- matrix(rnorm(5000),nrow=50); max <- nrow(X); sums <- numeric(max)
for (cc in 1:max) {
  sums[cc] <- sum(X[cc,])
  wait(.05) # just so this trivial loop doesn't finish so quickly
  loop.tracker(cc,max, sav.obj=sums, sav.fn="temp.rda", sav.freq=5);
  if(cc==29) { warning("faked a crash at iteration 29!"); rm(sums); break }
}
cat("\nloaded latest backup from iteration 28:",paste(load("temp.rda")),"\n")
print(sav.obj); unlink("temp.rda")
```

`memory.summary` *Summary of RAM footprint for all R objects in the current session. Not my function, but taken from an R-Help response by Elizabeth Purdom, at Berkeley. Simply applies the function 'object.size' to the objects in ls(). Also very similar to an example in the 'Help' for the utils::object.size() function.*

Description

Summary of RAM footprint for all R objects in the current session. Not my function, but taken from an R-Help response by Elizabeth Purdom, at Berkeley. Simply applies the function 'object.size' to the objects in ls(). Also very similar to an example in the 'Help' for the utils::object.size() function.

Usage

```
memory.summary(unit = c("kb", "mb", "gb", "b"))
```

Arguments

`unit`	default is to display "kb", but you can also choose "b"=bytes, "mb"= megabyte, or "gb" = gigabytes. Only the first letter is used, and is not case sensitive, so enter units how you like.

Value

a list of object names with memory usage in bytes

Examples

```
memory.summary() # shows memory used by all objects in the current session in kb
memory.summary("mb") # change units to megabytes
```

`Mode` *Find the mode of a vector.*

Description

The mode is the most common value in a series. This function can return multiple values if there are equally most frequent values, and can also work with non-numeric types.

Usage

```
Mode(x, multi = FALSE, warn = FALSE)
```

Arguments

x	The data to take the mode from. Dimensions and NA's are removed if possible, strings, factors, numeric all permitted
multi	Logical, whether to return multiple modes if values have equal frequency
warn	Logical, whether to give warnings when multiple values are found (if multi=FALSE)

Value

The most frequent value, or sorted set of most frequent values if multi==TRUE and there are more than one. Numeric if x is numeric, else as strings

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

Examples

```
Mode(c(1,2,3,3,4,4)) # 2 values are most common, as multi=FALSE,
# selects the last value (after sort)
Mode(c(1,2,3,3,4,4),multi=TRUE) # same test with multi=T,
# returns both most frequent
Mode(matrix(1:16,ncol=4),warn=TRUE) # takes mode of the entire
# matrix treating as a vector, but all values occur once
Mode(c("Tom","Dick","Harry"),multi=FALSE,warn=TRUE) # selects last
# sorted value, but warns there are multiple modes
Mode(c("Tom","Dick","Harry"),multi=TRUE,warn=TRUE) # multi==TRUE so
# warning is negated
```

---

must.use.package *Do everything possible to load an R package.*

Description

Like 'require()' except it will attempt to install a package if necessary. Installation of bioconductor packages is deprecated. Useful if you wish to share code with people who may not have the same libraries as you, you can include a call to this function which will simply load the library if present, or else install, then load, if they do not have it.

Usage

```
must.use.package(
  pcknms,
  ask = FALSE,
  reload = FALSE,
  avail = FALSE,
  quietly = FALSE
)
```

Arguments

pcknms	list of packages to load/install
ask	whether to get the user's permission to install a required package, or just go ahead and do it
reload	indicates to reload the package even if loaded
avail	see whether pcknms are in the list of available CRAN packages
quietly	passed to library/require, display installation text or not

Value

nothing, simply loads the packages specified if possible

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

Examples

```
# not run : run if you are ok to install/already have these packages
# must.use.package(c("MASS","nlme","lme4"),ask=FALSE)
# search() # show packages have loaded, then detach them again:
# sapply(paste("package",c("MASS","nlme","lme4"),sep=":"),detach,character.only=TRUE)
```

---

narm *Return an object with missing values removed.*

Description

Convenience function, removes NAs from most standard objects. Uses function na.exclude for matrices and dataframes. Main difference to na.exlude is that it simply performs the transformation, without adding attributes For unknown types, leaves unchanged with a warning.

Usage

```
narm(X)
```

Arguments

X	The object to remove NAs, any vector, matrix or data.frame

Value

Vector minus NA's, or the matrix/data.frame minus NA rows. If it's a character vector then values of "NA" will also be excluded in addition to values = NA, so be careful if "NA" is a valid value of your character vector. Note that "NA" values occur when 'paste(...,NA,...)' is applied to a vector of any type, whereas 'as.character(...,NA,...)' avoids this.

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

Examples

```
narm(c(1,2,4,NA,5))
DF <- data.frame(x = c(1, 2, 3), y = c(0, 10, NA))
DF; narm(DF)
# if a list, will only completely remove NA from the lowest levels
# empty places will be left at top levels
print(narm(list(1,2,3,NA,list(1,2,3,NA))))
```

---

nearest.to *Select the nearest point in an array to a given value*

Description

Similar to the base function match() but allows for data where you won't find an exact match. Selects the nearest value from 'array' to the value 'point'. Sometimes there are multiple points with equal distance in which case choose from 3 possible 'dispute.method's for choosing which of the equidistant array values to index. returns the index of 'array' to which 'point' is nearest.

Usage

```
nearest.to(array, point, dispute.method = c("first", "last", "random"))
```

Arguments

array	a numeric vector or POSIXct vector of date-times.
point	the value that you want to find the nearest point to.
dispute.method	when there are equidistant values to 'point' in array, choose either the first, last, or a random select, based on the original order in 'array.

Value

index value of the nearest point in 'array'.

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

Examples

```
myArray <- 1:100
nearest.to(myArray, 7.7)
nearest.to(myArray, 50.5)
nearest.to(myArray, 50.5, dispute.method="last")
```

Numerify *Convert all possible columns of a data.frame to numeric*

Description

Importing data from csv files can often lead to numeric variables being coded as factors or strings. This will not work well with many R functions. This function provides a quick way to deal with this across a whole data frame while attempting to leave columns untouched that are not genuinely numeric data. In edge cases you might need to adjust 'threshold' to get the correct result, usually an issue if mostly numeric columns often have strings amongst them, for instance a column with mostly numbers, but occassionally pipe-separated values like '4.4|5.0|6.1', etc.

Usage

```
Numerify(df, except = NULL, force = FALSE, digits = NA, thresh = 0.9)
```

Arguments

df	data.frame to transform to numeric (where possible)
except	avoid changing any colnames in this array
force	force all columns to numeric without checking types
digits	if a non-NA integer value is used, will round numeric columns to this many decimal places after making numeric.
thresh	threshold to decide that a variable is numeric. NA values will be ignored in the test. Then it looks at the proportion of values that are successfully coerced to numeric without giving 'NA'. If this threshold is 0.9, then any column where at least 90 converted to numeric type, will be kept as numeric, else they will be left as they were.

Value

data.frame with numeric type for any applicable columns

Author(s)

Nicholas Cooper

Examples

```
df <- data.frame(first=c(1:5),
 second=paste(6:10),
 third=c("jake", "fred", "cathy", "sandra", "mike"))
sapply(sapply(df, is), "[", 1) # check type of each column
dfN <- Numerify(df)
sapply(sapply(dfN, is), "[", 1) # now second column is numeric
df2 <- data.frame(first=c(1:10),
 second=paste(c(NA, NA, 6:10, "5|6", "7|8", 1)),
```

```
 third=rep(c("jake", "fred", "cathy", "sandra", "mike"),2))
sapply(sapply(df2, is), "[", 1)
df2N1 <- Numerify(df2, thresh=0.7)
df2N2 <- Numerify(df2, thresh=0.8)
sapply(sapply(df2N1, is), "[", 1) # at this threshold second column goes to numeric
sapply(sapply(df2N2, is), "[", 1) # second column stays a string at this threshold
```

`out.of` *Easily display fraction and percentages*

Description

For a subset 'n' and total 'N', nicely prints text n/N and/or percentage Often we want to display proportions and this simple function reduces the required amount of code for fraction and percentage reporting. If insufficient digits are provided small percentage may truncate to zero.

Usage

```
out.of(n, N = 100, digits = 2, pc = TRUE, oo = TRUE, use.sci = FALSE)
```

Arguments

`n`	numeric, the count for the subset of N (the numerator)
`N`	numeric, the total size of the full set (the denominator)
`digits`	integer, the number of digits to display in the percentage
`pc`	logical, whether to display the percentage of N that n comprises
`oo`	logical, whether to display n/N as a fraction
`use.sci`	logical, whether to allow scientific notation for small/large percentages.

Value

A string showing the fraction n/N and percentage (or just one of these)

Examples

```
out.of(345,12144)
out.of(345,12144,pc=FALSE)
out.of(3,10^6,digits=6,oo=FALSE)
out.of(3,10^6,digits=6,oo=FALSE,use.sci=TRUE)
```

p.to.Z *Convert p-values to Z-scores*

Description

Simple conversion of two-tailed p-values to Z-scores. Written in a way that allows maximum precision for small p-values.

Usage

```
p.to.Z(p)
```

Arguments

p	p-values (between 0 and 1), numeric, scalar, vector or matrix, or other types coercible using as.numeric()

Value

Z scores with the same dimension as the input

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

See Also

Z.to.p

Examples

```
p.to.Z(0.0001)
p.to.Z("5E-8")
p.to.Z(c(".05",".01",".005"))
p.to.Z(matrix(runif(16),nrow=4))
```

packages.loaded *Check whether a set of packages has been loaded*

Description

Returns TRUE if the whole set of packages entered has been loaded, or FALSE otherwise. This can be useful when developing a package where there is optional functionality depending if another package is in use (but the other package is not part of 'depends' because it is not essential). Because 'require' cannot be used within functions submitted as part of a CRAN package.

Usage

```
packages.loaded(pcks = "", ..., cran.check = FALSE, repos = getRepositories())
```

Arguments

pcks	character, a package name, or vector of names, if left blank will return all loaded
...	further package names as character (same as entering via pcks, but avoids need for c() in pcks)
cran.check	logical, in the case at least one package is not found, whether to search CRAN and see whether the package(s) even exist on CRAN.
repos	repository to use if package is not loaded and cran.check=TRUE, if NULL, will attempt to use the repository in getOptions("repos") or will default to the imperial.ac.uk mirror. Otherwise the default is to use all available repositories from getRepositories()

Value

logical TRUE or FALSE whether the whole list of packages are available

Author(s)

Nicholas Cooper

Examples

```
packages.loaded("NCmisc","reader")
packages.loaded() # no argument means all loaded packages are listed
```

---

pad.left *Print a vector with appropriate padding so each has equal char length.*

Description

Print a vector with appropriate padding so each has equal char length.

Usage

```
pad.left(X, char = " ", numdigits = NA)
```

Arguments

X	vector of data to pad to equal length
char	character to pad with, space is default, but zero might be a desirable choice for padding numbers
numdigits	if using numeric data, the number of digits to keep

Value

returns the vector in character format with equal nchar()

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

Examples

```
pad.left(1:10)
phone.numbers <- c("07429719234","7876345123","7123543765")
pad.left(phone.numbers,"0")
pad.left(rnorm(10),numdigits=3)
```

---

pctile *Find data thresholds corresponding to percentiles*

Description

Finds the top and bottom bounds corresponding to percentile 'pc' of the data 'dat'.

Usage

```
pctile(dat, pc = 0.01)
```

Arguments

dat	numeric vector of data
pc	the percentile to seek, c(pc, 1-pc)

Value

returns the upper and lower threshold

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

Examples

```
pctile(rnorm(100000),.025)
pctile(sample(100),.9)
```

ppa *Posterior probability of association function*

Description

Estimate the probability of your hypothesis being true, given the observed p-value and a prior probability of the hypothesis being true.

Usage

```
ppa(p = 0.05, prior = 0.5, BF = NULL, quiet = TRUE)
```

Arguments

p	p-value you want to test [p<0.367], or 'bayes factor'
prior	prior odds for the hypothesis (Ha) being tested
BF	logical, set to TRUE if you have entered a bayes factor as 'p' rather than a p-value
quiet	logical, whether to display verbose information for calculation

Value

prints calculations, then returns the posterior probability of association given the observed p-value under the specified prior

References

Equations 1, 2 from http://www.readcube.com/articles/10.1038/nrg2615 Equations 2, 3 from http://www.tandfonline.com/doi

Examples

```
ps <- rep(c(.05,.01),3)
prs <- rep(c(.05,.50,.90),each=2)
mapply(ps,prs,FUN=ppa)  # replicate Nuzzo 2014 table
# try with bayes factors
ppa(BF=3,prior=.9)
ppa(BF=10,prior=.5)
```

preview *Output variable states within functions during testing/debugging*

Description

A versatile function to compactly display most common R objects. Will return the object name, type, dimension, and a compact representation of object contents, for instance using prv.large() to display matrices, so as to not overload the console for large objects. Useful for debugging, can be placed inside loops and functions to track values, dimensions, and data types. Particularly when debugging complex code, the automatic display of the variable name prevents confusion versus using regular print statements. By listing variables to track as character(), provides 'cat()' output of compact and informative variable state information, e.g, variable name, value, datatype and dimension. Can also specify array or list elements, or custom labels. prv() is the same as preview() except it can take objects without using double quotes and has no 'labels' command (and doesn't need one).

Usage

```
preview(
  varlist,
  labels = NULL,
  counts = NULL,
  assume.char = FALSE,
  prv.call = FALSE
)
```

Arguments

varlist	character vector, the list of variable(s) to report, which will trigger automatic labelling of the variable name, otherwise if entered as the variable value (ie. without quotes, then will by default be displayed as 'unknown variable')
labels	will label 'unknown variables' (see above) if entered as variables without quotes
counts	a list of array index values; so if calling during a counting loop, the value can be reported each iteration, also printing the count index; if the list is named the name will also appear, e.g, variable[count=1]. This list must be the same length as varlist (and labels if not NULL), and each element [[i]] must contain as many values as the original corresponding varlist[i] has dimensions. The dimensions must result in a 1x1 scalar
assume.char	usually 'varlist' is a character vector of variable names, but in the case that it is actually a character variable, using assume.char=TRUE will ensure that it will be assumed the character variable is the object to preview, rather than a list of variable names. So long as none of the values are found to be variable names in the global environment. preview() can also find variables in local environments, and if this is where the target variable lies, it is best to use assume.char=FALSE, otherwise the search for alternative environments might not happen. Note that in most cases the automatic detection of the input should understand what you want, regardless of the value of assume.char.

prv.call — It is recommended to always leave this argument as FALSE when calling preview() directly. If set to TRUE, it will first search 2 generations back for the parent frame, instead of one, as it will assume that the variable(s) to preview are not directly called by preview(), but through a wrapper for preview, such as prv().

See Also

Dim

Examples

```
# create variables of different types to show output styles #
testvar1 <- 193
testvar2 <- "Atol"
testvar3 <- c(1:10)
testvar4 <- matrix(rnorm(100),nrow=25)
testvar5 <- list(first="test",second=testvar4,third=100:110)
preview("testvar1")
preview("testvar4")
preview(paste("testvar",1:5,sep=""))
preview(testvar1,"myvarname")
preview(testvar1)
# examples with loops and multiple dimensions / lists
for (cc in 1:4) {
 for (dd in 1:4) { preview("testvar4",counts=list(cc,dd)) }}

for (dd in 1:3) { preview("testvar5",counts=list(dd=dd)) }
```

---

prv — *Output variable states within functions/loops during testing/debugging*

Description

Same as preview but no labels command, and input is without quotes and should be plain variable names of existing variables (no indices, args, etc) A versatile function to compactly display most common R objects. Will return the object name, type, dimension, and a compact representation of object contents, for instance using prv.large() to display matrices, so as to not overload the console for large objects. Useful for debugging, can be placed inside loops and functions to track values, dimensions, and data types. Particularly when debugging complex code, the automatic display of the variable name prevents confusion versus using regular print statements. By listing variables to track as character(), provides 'cat()' output of compact and informative variable state information, e.g, variable name, value, datatype and dimension. Can also specify array or list elements, or custom labels. prv() is the same as preview() except it can take objects without using double quotes and has no 'labels' command (and doesn't need one). If expressions are entered rather than variable names, then prv() will attempt to pass the arguments to preview(). prv() assumes that the variable(s) to report originate from the environment calling prv(), and if not found there, then it will search through all accessible environments starting with the global environment, and then will report the

first instance found, which in exceptional circumstances (be warned) may not be the instance you intended to retrieve.

Usage

```
prv(..., counts = NULL)
```

Arguments

...	series of variable(s) to report, separated by commas, which will trigger automatic labelling of the variable name
counts	a list of array index values; so if calling during a counting loop, the value can be reported each iteration, also printing the count index; if the list is named the name will also appear, e.g, variable[count=1]. This list must be the same length as the variable list ... , and each element [[i]] must contain as many values as the original corresponding variable list[i] has dimensions

See Also

Dim

Examples

```
# create variables of different types to show output styles #
testvar1 <- 193
testvar2 <- "Atol"
testvar3 <- c(1:10)
testvar4 <- matrix(rnorm(100),nrow=25)
testvar5 <- list(first="test",second=testvar4,third=100:110)
preview("testvar1"); prv(testvar1)
prv(testvar1,testvar2,testvar3,testvar4)
prv(matrix(rnorm(100),nrow=25)) # expression sent to preview() with no label
prv(193) # fails as there are no object names involved
```

---

prv.large *Tidy display function for matrix objects*

Description

This function prints the first and last columns and rows of a matrix, and more, if desired. Allows previewing of a matrix without overloading the console. Most useful when data has row and column names.

Usage

```
prv.large(
  largeMat,
  rows = 3,
  cols = 2,
  digits = 4,
  rL = "Row#",
  rlab = "rownames",
  clab = "colnames",
  rownums = T,
  ret = FALSE,
  warn = TRUE
)
```

Arguments

largeMat	a matrix
rows	number of rows to display
cols	number of columns to display
digits	number of digits to display for numeric data
rL	row label to describe the row names/numbers, e.g, row number, ID, etc
rlab	label to describe the data rows
clab	label to describe the data columns
rownums	logical, whether to display rownumbers or ignore them
ret	logical, whether to return the result as a formatted object, or just print to console
warn	logical, whether to warn if the object type is not supported

Examples

```
mat <- matrix(rnorm(1000),nrow=50)
rownames(mat) <- paste("ID",1:50,sep="")
colnames(mat) <- paste("Var",1:20,sep="")
prv.large(mat)
prv.large(mat,rows=9,cols=4,digits=1,rlab="samples",clab="variables",rownums=FALSE)
```

---

replace.missing.df *Iterate through numeric columns of a dataframe and replace missing with the mean*

Description

To simple replace missing data without changing column means. This will also use criteria to decide whether each column is numeric, so that illegal operations aren't performed on strings, etc. Also adjusting the 'error' parameter allows adding variance to the missing observations to help to reduce bias associated with inserting many of the same replacement value.

Usage

```
replace.missing.df(
  X,
  repl.fun = mean,
  error = 0,
  thresh = 0.9,
  digits = 99,
  force = FALSE
)
```

Arguments

X	a data.frame to replace missing values in
repl.fun	the function to perform the replacement. Default is 'mean'. A replacement should take a vector 'x' and produce a single scalar as a result.
error	default value is 0, meaning replacements will be all the same value for each column of the data.frame X. If you give a positive value, this amount of gaussian noise (in StDev units of the original variable) will be added to the replacement values.
thresh	passed to function 'is.vec.numeric', see explanation there.
digits	Trim replacement values to this many digits
force	TRUE means replace missing for all columns with testing for numeric

Value

returns a data.frame with the same dimensions with missing values for numeric values imputed using the repl.fun function, optionally with noise added.

Author(s)

Nicholas Cooper

Examples

```
df <- data.frame(first=c(1,2,NA,4,5),
  second=paste(c(6,7,8,NA,10)),
  third=c("jake", "fred", "cathy", "sandra", "mike"))
df
replace.missing.df(df)
replace.missing.df(df, force=TRUE)
df2 <- data.frame(first=c(1:5, NA, NA, NA,9, 10),
  second=paste(c(NA, NA, 6:10, "5|6", "7|8", 1)),
  third=rep(c("jake", "fred", "cathy", "sandra", "mike"),2))
df2
replace.missing.df(df2)
replace.missing.df(df2, thresh=0.7)
replace.missing.df(df2, error = 1, thresh=0.7, digits=4)
```

`Rfile.index` *Create an index file for an R function file*

Description

Create a html index for an R function file by looking for functions, add descriptions using comments directly next to the function() command. Note that if too much code other than well-formatted functions is in the file then the result is likely not to be a nicely formatted index.

Usage

```
Rfile.index(fn, below = TRUE, fn.out = "out.htm", skip.indent = TRUE)
```

Arguments

`fn`	an R file containing functions in standard R script
`below`	whether to search for comment text below or above the function() calls
`fn.out`	optional name for the output file, else will be based on the name of the input file
`skip.indent`	whether to skip functions that are indented, the assumption being they are functions within functions

Value

creates an html file with name and description of each function

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

See Also

`list.functions.in.file`

Examples

```
# not run:  rfile <- file.choose() # choose an R script file with functions
# not run:  out <- Rfile.index(rfile,fn.out="temp.htm")
# unlink("temp.htm") # run once you've inspected this file in a browser
```

rmv.names *Remove names from a named vector or list*

Description

Convenience function, it's very easy to set names to NULL, but this requires a dedicated line of code. Using this function can make your code simpler.

Usage

```
rmv.names(X)
```

Arguments

X	object for which you want to remove name

Value

the original object but without names

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

Examples

```
x <- c(boo=1, hiss=2)
rmv.names(x)
X <- list(testing=c(1,2,3), thankyou=TRUE)
rmv.names(X)
```

rmv.spc *Remove leading and trailing spaces (or other character).*

Description

Remove leading and trailing spaces (or other character).

Usage

```
rmv.spc(str, before = TRUE, after = TRUE, char = " ")
```

Arguments

str	character vector, may containing leading or trailing chars
before	logical, whether to remove leading spaces
after	logical, whether to remove trailing spaces
char	an alternative character to be removed instead of spaces

Value

returns vectors without the leading/trailing characters

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

See Also

spc

Examples

```
rmv.spc("  mid sentence  ")
rmv.spc("0012300",after=FALSE,char="0")
rmv.spc(" change nothing ",after=FALSE,before=FALSE)
```

---

search.cran *Search all CRAN packages for those containing keyword(s).*

Description

Can be useful for trying to find new packages for a particular purpose. No need for these packages to be installed or loaded. Further searching can be done using utils::RSiteSearch()

Usage

```
search.cran(txt, repos = "", all.repos = FALSE)
```

Arguments

txt	text to search for, a character vector, not case-sensitive
repos	repository(s) (CRAN mirror) to use, "" defaults to getOption("repos")
all.repos	logical, if TRUE, then use all available repositories from getRepositories()

Value

list of hits for each keyword (txt)

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

Examples

```
## not run # repos <- "http://cran.ma.imperial.ac.uk/" # OR: repos <- getOption("repos")
## not run # search.cran("draw")
## not run # search.cran(c("hmm", "markov", "hidden"))
```

`sim.cor` *Simulate a dataset with correlated measures*

Description

Simulate a dataset with correlated measures (normal simulation with e.g, rnorm() usually only gives small randomly distributed correlations between variables). This is a quick and unsophisticated method, but should be able to provide a dataset with slightly more realistic structure than simple rnorm() type functions. Varying the last three parameters gives some control on the way the data is generated. It starts with a seed random variable, then creates 'k' random variables with an expected correlation of r=genr() with that seed variable. Then after this, one of the variables in the set (including the seed) is randomly selected to run through the same process of generating 'k' new variables; this is repeated until columns are full up. 'mix.order' then randomizes the column order destroying the relationship between column number and correlation structure, although in some cases, such relationships might be desired as representative of some real life datasets.

Usage

```
sim.cor(
  nrow = 100,
  ncol = 100,
  genx = rnorm,
  genr = runif,
  k = 3,
  mix.order = TRUE
)
```

Arguments

`nrow`	integer, number of rows to simulate
`ncol`	integer, number of columns to simulate
`genx`	the generating function for data, e.g rnorm(), runif(), etc
`genr`	the generating function for desired correlation, e.g, runif()
`k`	number of steps generating from the same seed before choosing a new seed
`mix.order`	whether to randomize the column order after simulating

Author(s)

Nicholas Cooper

See Also

`cor.with`

Examples

```
corDat <- sim.cor(200,5)
prv(corDat) # preview of simulated normal data with r uniformly varying
cor(corDat) # correlation matrix
corDat <- sim.cor(500,4,genx=runif,genr=function(x) { 0.5 },mix.order=FALSE)
prv(corDat) # preview of simulated uniform data with r fixed at 0.5
cor(corDat) # correlation matrix
```

---

simple.date *Simple representation and retrieval of Date/Time*

Description

Retrieve a simple representation of date_time or just date, for generating day/time specific file names, etc.

Usage

```
simple.date(sep = "_", long = FALSE, time = TRUE)
```

Arguments

sep	character, separator to use for the date/time, eg, underscore or <space> " ".
long	logical, whether to display a longer version of the date and time, or just a simple version
time	logical, whether to include the time, or just the date

Value

A string containing the date: MMMDD and optionally time HRam/pm. Or if long=TRUE, a longer representation: DAY MM DD HH.MM.SS YYYY.

Examples

```
simple.date()
simple.date(" ",long=TRUE)
simple.date(time=FALSE)
```

spc — *Print a character a specified number of times.*

Description

Returns 'char' X_i number of times for each element i of X. Useful for padding for alignment purposes.

Usage

```
spc(X, char = " ")
```

Arguments

X	numeric vector of number of repeats
char	The character to repeat (longer will be shortened)

Value

returns vectors of strings of char, lengths X

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

See Also

rmv.spc

Examples

```
cat(paste(spc(9),"123\n"))
cat(paste(spc(8),"1234\n"))
spc(c(1:5),".")
```

standardize — *Convert a numeric vector to Z-scores.*

Description

Transform a vector to z scores by subtracting its mean and dividing by its standard deviation

Usage

```
standardize(X)
```

Arguments

X	numeric vector to standardize

Value

vector of the same length in standardised form

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

Examples

```
x1 <- rnorm(10,100,15); x2 <- sample(10)
print(x1) ;  standardize(x1)
print(x2) ;  standardize(x2)
```

Substitute *Convert objects as arguments to object names*

Description

Equivalent to the base function substitute() but can do any length of arguments instead of just one. Converts the objects in parentheses into text arguments as if they had been entered with double quote strings. The objects must exist and be accessible in the environment the function is called from for the function to work (same as for substitute()). One application for this is to be able to create functions where object arguments can be entered without quotation marks (simpler), or where you want to use the name of the object as well as the data in the object.

Usage

```
Substitute(x = NULL, ...)
```

Arguments

x	compulsory, simply the first object in the list, no difference to any further objects
...	any further objects to return string names for.

Value

character list of x,... object names

Author(s)

Nicholas Cooper

See Also

prv, preview

Examples

```
myvar <- list(test=c(1,2,3)); var2 <- "testme"; var3 <- 10:14
print(myvar)
# single variable case, equivalent to base::substitute()
print(substitute(myvar))
print(Substitute(myvar))
# multi variable case, substitute won't work
Substitute(myvar,var2,var3)
# prv() is a wrapper for preview() allowing arguments without parentheses
# which is achieved internally by passing the arguments to Substitute()
preview(c("myvar","var2","var3"))
prv(myvar,var2,var3)
```

---

summarise.r.datasets *Summarise the dimensions and type of available R example datasets*

Description

This function will parse the current workspace to see what R datasets are available. Using the toHTML function from the 'tools' package to interpret the data() call, each dataset is examined in turn for type and dimensionality. Can also use a filter for dataset types, to only show, for instance, matrix datasets. Also you can specify whether to only look for base datasets, or to search for datasets in all available packages. Result is a printout to the console of the available datasets and their characteristics.

Usage

```
summarise.r.datasets(
  filter = FALSE,
  types = c("data.frame", "matrix"),
  all = FALSE,
  ...
)
```

Arguments

filter	logical, whether to filter datasets by 'types'
types	if filter=TRUE, which data types to include in the result
all	logical, if all=TRUE, look for datasets in all available packages, else just base
...	if all is false, further arguments to the data() function to search datasets

Author(s)

Nicholas Cooper

Examples

```
summarise.r.datasets()
summarise.r.datasets(filter=TRUE,"matrix")
```

---

summary2 *Descriptive summary with SD/SE + improved formatting*

Description

Wrapper for the base function summary() but adds standard deviation, standard error, and an 'N' and missing 'NA' count that are consistent.

Usage

```
summary2(x, digits = NULL, neaten.names = TRUE)
```

Arguments

x	vector of numeric data
digits	number of digits to round resulting values to
neaten.names	logical, TRUE removes period and space from names of the results returned by summary() to make the names better for use in a data.frame.

Value

array of descriptive statistics for x

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

Examples

```
x <- 1:100
summary2(x, digits=2)
summary2(c(x, NA, NA), digits=2)
```

table2d *Wrapper for the base table() function that includes zero counts - useful to get consistent dimensions across multiple runs with different responding patterns Forces a 2d table with every possible cell (allow zero counts) Only for tables where there are two vectors entered, while the base function allows for more, or also allows just 1. If the wrong arguments are entered, attempts to pass the input to the base version of 'table' instead.*

Description

Wrapper for the base table() function that includes zero counts - useful to get consistent dimensions across multiple runs with different responding patterns Forces a 2d table with every possible cell (allow zero counts) Only for tables where there are two vectors entered, while the base function allows for more, or also allows just 1. If the wrong arguments are entered, attempts to pass the input to the base version of 'table' instead.

Usage

```
table2d(
  ...,
  col,
  row,
  rn = NULL,
  cn = NULL,
  use.order = TRUE,
  remove.na = FALSE
)
```

Arguments

`...`	vector arguments, see input for base:table() function
`col`	categories to include as columns of the table
`row`	categories to include as rows of the table
`rn`	optionally replace the raw value names with desired row names. Must be same length as 'row'.
`cn`	optionally replace the raw value names with desired column names. Must be same length as 'col'.
`use.order`	TRUE to use the order in 'col' and 'row' for table, otherwise use the default order of table() - which is usually alphabetical
`remove.na`	remove NA values from row/col if present

Value

returns a table, just like the base:table() function but the row and column names are fixed regardless of count

Author(s)

Nicholas Cooper

Examples

```
nm <- c("Mike", "Anna", "John", "Tony")
vec_r <- sample(tolower(nm)[c(1,3,4)], 50, replace=TRUE)
vec_c <- sample(c(1,2,4,5), 50, replace=TRUE)
table(vec_r, vec_c)
table2d(vec_r, vec_c, row=tolower(nm), col=paste(1:5))
table2d(vec_r, vec_c, row=tolower(nm), col=paste(1:5), use.order = FALSE)
table2d(vec_r, vec_c, row=tolower(nm), col=paste(1:5), rn=nm, cn=c("I", "II", "III", "IV", "V"))
```

---

textogram *Make an ascii histogram in the console.*

Description

Uses a call to base::hist(...) and uses the densities to make a a text histogram in the console Particularly useful when working in the terminal without graphics.

Usage

```
textogram(X, range = NA, ...)
```

Arguments

X	numeric vector of data
range	optional sub-range of X to test; c(low,high)
...	additional arguments passed to base::hist()

Value

outputs an ascii histogram to the console

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

Examples

```
textogram(runif(100000))
textogram(rnorm(10000),range=c(-3,3))
```

timeit *Times an expression, with breakdown of time spent in each function !DEPRECATED October 14, 2022!*

Description

A wrapper for the proftools package Rprof() function. It is to Rprof() as system.time() is to proc.time() (base) Useful for identifying which functions are taking the most time. This procedure will return an error unless expr takes more than ~0.1 seconds to evaluate. I could not see any simple way to avoid this limitation. Occassionally other errors are produced for no apparent reason which are due to issues within the proftools package that are out of my control.

Usage

```
timeit(expr, suppressResult = F, total.time = TRUE)
```

Arguments

expr	an expression, must take at least 1 second (roughly)
suppressResult	logical, if true, will return timing information rather than the result of expr
total.time	to sort by total.time, else by self.time

Value

returns matrix where rows are function names, and columns are self.time and total.time. total.time is total time spent in that function, including function calls made by that function. self.time doesn't count other functions within a function

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

Examples

```
# this function writes and removes a temporary file
# run only if ok to do this in your temporary directory
#not run# timeit(wait(0.1,"s") ,total.time=TRUE)
#not run# timeit(wait(0.1,"s") ,total.time=FALSE)
```

toheader *Return a string with each first letter of each word in upper case.*

Description

Return a string with each first letter of each word in upper case.

Usage

```
toheader(txt, strict = FALSE)
```

Arguments

txt	a character string
strict	whether to force non-leading letters to lowercase

Value

Vector minus NA's, or the matrix/data.frame minus NA rows

Author(s)

via R Core

Examples

```
toheader(c("using AIC for model selection"))
toheader(c("using AIC", "for MODEL selection"), strict=TRUE)
```

top *Monitor CPU, RAM and Processes*

Description

This function runs the unix 'top' command and returns the overall CPU and RAM usage, and optionally the table of processes and resource use for each. Works only with unix-based systems such as Mac OS X and Linux, where 'top' is installed. Default is to return CPU and RAM overall stats, to get detailed stats instead, set Table=TRUE.

Usage

```
top(
  CPU = !Table,
  RAM = !Table,
  Table = FALSE,
  procs = 20,
  mem.key = NULL,
  cpu.key = NULL
)
```

Arguments

CPU	logical, whether to return overall CPU usage information
RAM	logical, whether to return overall RAM usage information
Table	logical, whether to return system information for separate processes. This is returned as table with all of the same columns as a command line 'top' command. If 'Table=TRUE' is set, then the default becomes not to return the overall CPU/RAM usage stats. The dataframe returned will have been sorted by descending memory usage.
procs	integer, if Table=TRUE, then the maximum number of processes to return (default 20)
mem.key	character, default for Linux is 'mem' and for Mac OS X, 'physmem', but if the 'top' command on your system displays memory usage using a different label, then enter it here (case insensitive) to override defaults.
cpu.key	character, default for Linux and Mac OS X is 'cpu', but if the top command on your system displays CPU usage using a different label, then enter it here.

Value

a list containing CPU and RAM usage, or with alternate parameters can return stats for each process

Author(s)

Nicholas Cooper

Examples

```
# not run #  top()
# not run #  top(Table=TRUE,proc=5)
```

Unlist *Unlist a list, starting only from a set depth.*

Description

Allows unlisting preserving the top levels of a list. Can specify the number of list depth levels to skip before running unlist()

Usage

```
Unlist(obj, depth = 1)
```

Arguments

obj	the list to unlist
depth	skip to what layer of the list before unlisting; eg. the base unlist() function would correspond to depth=0

Value

returns vectors of strings of char, lengths X

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

Examples

```
complex.list <- list(1,1:3,list(2,2:4,list(3,3:4,list(10))),list(4,5:7,list(3)))
Unlist(complex.list,0) # equivalent to unlist()
Unlist(complex.list,1) # unlist within the top level lists
Unlist(complex.list,2) # unlist within the second level lists
Unlist(complex.list,10) # once depth >= list-depth, no difference
unlist(complex.list,recursive=FALSE) # not the same as any of the above
```

wait *Wait for a period of time.*

Description

Waits a number of hours minutes or seconds (doing nothing). Note that this 'waiting' will use 100

Usage

```
wait(dur, unit = "s", silent = TRUE)
```

Arguments

dur	waiting time
unit	time units h/m/s, seconds are the default
silent	print text showing that waiting is in progress

Value

no return value

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

Examples

```
wait(.25,silent=FALSE) # wait 0.25 seconds
wait(0.005, "m")
wait(0.0001, "Hours", silent=FALSE)
```

---

which.outlier *Return vector indexes of statistical univariate outliers*

Description

Performs simplistic outlier detection and returns indexes for outliers. Acts like the which() function, return indices of elements of a vector satisfying the condition, which by default are outliers exceeding 2 SD above or below the mean. However, the threshold can be specified, only high or low values can be considered outliers, and percentile and interquartile range thresholds can also be used.

Usage

```
which.outlier(
  x,
  thr = 2,
  method = c("sd", "iq", "pc"),
  high = TRUE,
  low = TRUE
)
```

Arguments

x	numeric, or coercible, the vector to test for outliers
thr	numeric, threshold for cutoff, e.g, when method="sd", standard deviations, when 'iq', interquartile ranges (thr=1.5 is most typical here), or when 'pc', you might select the extreme 1%, 5%, etc.

method	character, one of "sd","iq" or "pc", selecting whether to test for outliers by standard deviation, interquartile range, or percentile.
high	logical, whether to test for outliers greater than the mean
low	logical, whether to test for outliers less than the mean

Value

indexes of the vector x that are outliers according to either a SD cutoff, interquartile range, or percentile threshold, above (high) and/or below (low) the mean/median.

Examples

```
test.vec <- rnorm(200)
summary(test.vec)
ii <- which.outlier(test.vec) # 2 SD outliers
prv(ii); vals <- test.vec[ii]; prv(vals)
ii <- which.outlier(test.vec,1.5,"iq") # e.g, 'stars' on a box-plot
prv(ii)
ii <- which.outlier(test.vec,5,"pc",low=FALSE) # only outliers >mean
prv(ii)
```

Z.to.p *Convert Z-scores to p-values*

Description

Simple conversion of Z-scores to two-tailed p-values. Written in a way that allows maximum precision for small p-values.

Usage

```
Z.to.p(Z, warn = FALSE)
```

Arguments

Z	Z score, numeric, scalar, vector or matrix, or other types coercible using as.numeric()
warn	logical, whether to give a warning for very low p-values when precision limits are exceeded.

Value

p-valuues with the same dimension as the input

Author(s)

Nicholas Cooper <njcooper@gmx.co.uk>

See Also

p.to.Z

Examples

```
Z.to.p("1.96")
Z.to.p(p.to.Z(0.0001))
Z.to.p(37, TRUE)
Z.to.p(39, TRUE) # maximum precision exceeded, warnings on
Z.to.p(39) # maximum precision exceeded, warnings off
```

Index