Package 'caroline'

November 9, 2023

Version 0.9.2

Date 2023-11-10

Title A Collection of Database, Data Structure, Visualization, and Utility Functions for R

Author David Schruth

Maintainer David Schruth <code@anthropoidea.org>

Depends R (>= 2.0.0), methods

Suggests MASS, RSQLite, grid

Enhances DBI, gplots

Description The caroline R library contains dozens of functions useful for: database migration (dbWriteTable2), database style joins & aggregation (nerge, groupBy & bestBy), data structure conversion (nv, tab2df), legend table making (sstable & leghead), plot annotation (labsegs & mvlabs), data visualization (pies, sparge, & raPlot), character string manipulation (m & pad), file I/O (write.delim), batch scripting and more. The package's greatest contributions lie in the database style merge, aggregation and interface functions as well as in it's extensive use and propagation of row, column and vector names in most functions.

License Artistic-2.0

LazyLoad yes

NeedsCompilation no

Repository CRAN

Date/Publication 2023-11-09 08:20:02 UTC

R topics documented:

`addFactLevs` *Add new levels to the Factors in a DataFrame.*

Description

This function loops through all of the factor columns in a dataframe and adds new.levs to the factor levels list.

Usage

```
addFactLevs(x, new.levs=NA)
```

Arguments

x	a dataframe.
new.levs	new levels to be added.

See Also

factor, levels

---

bestBy *Find the "best" record within subgroups of a dataframe.*

Description

Finding the an extreme record for each group within a dataset is a more challenging routine task in R and SQL. This function provides a easy interface to that functionality either using R (fast for small data frames) or SQL (fastest for large data)

Usage

```
bestBy(df, by, best, clmns=names(df), inverse=FALSE, sql=FALSE)
```

Arguments

df	a data frame.
by	the factor (or name of a factor in df) used to determine the grouping.
clmns	the colums to include in the output.
best	the column to sort on (both globally and for each sub/group)
inverse	the sorting order of the sort column as specified by 'best'
sql	whether or not to use SQLite to perform the operation.

Value

A data frame of 'best' records from each factor level

Author(s)

David Schruth

See Also

groupBy

Examples

```
blast.results <- data.frame(score=c(1,2,34,4,5,3,23),
                            query=c('z','x','y','z','x','y','z'),
                            target=c('a','b','c','d','e','f','g')
                            )
best.hits.R <- bestBy(blast.results, by='query', best='score', inverse=TRUE)
best.hits.R
## or using SQLite
best.hits.sql <- bestBy(blast.results, by='query', best='score', inverse=TRUE, sql=TRUE)
best.hits.sql
```

`dbWriteTable2` *Data Import Wrapper for dbWriteTable.*

Description

This is wrapper for dbWriteTable written with the the primary improvements focusing on database import into an existing table definition schema. The function matches and rearranges columns of the dataframe to database feilds and additionally performs checks for NA's in required variables, overlength strings, and type mismatches. There also exists support for updating of the PostgreSQL specific sequence for tables with auto incrementing primary keys.

Usage

```
dbWriteTable2(con, table.name, df, fill.null = TRUE, add.id=TRUE,
                   row.names=FALSE, pg.update.seq=FALSE, ...)
```

Arguments

`con`	connection.
`table.name`	The name of the table to which the data frame is to be loaded.
`df`	A dataframe to be loaded to the database.
`fill.null`	Should new db present fields be added to the data.frame before it is loaded?.
`add.id`	Should a new column should be added for the database id?
`row.names`	Should the row names be loaded as a seperate column? (unlike the original dbWriteTable, default is FALSE)
`pg.update.seq`	should the table primary key's sequence be updated to the highest id value +1? (Postgres specific)
`...`	other parameters passed to dbWriteTable.

Value

If successful, the ids of the newly added database records (invisible)

See Also

dbWriteTable

`distro.dots` *Distribution plot of points*

Description

Plot the raw distribution of points, like a series of horizontal box plots.

Usage

```
distro.dots(x, jit.f=1, add=FALSE, pd=0,  vv=names(x), vvlabs=NULL,
                  xlim=range(unlist(x)), ...)
```

Arguments

`x`	a list of vectors of values to be plotted
`jit.f`	factor for random jittering (see 'jitter()'
`add`	should we add to the existing plot?
`pd`	'position dodge' moves all y axis plotting positions up or down by this provided value (useful for adding multiple distributions for the same variable)
`vv`	the variable vector for ordering the y-axis labels
`vvlabs`	the variable vector labels for labeling the plot (defaults to vv)
`xlim`	x axis plot limits
`...`	other parameters passed on to plot

Value

a 'distro dot plot' of variable distributions

Examples

```
n <- rnorm(130, 10, 3)
p <- rpois(110, 4)
u <- runif(300, 0, 20)
l <- rlnorm(130, log(2))
g <- rgamma(140, 3)

X <- list(a=u, random=n, array=p,of=l, variable=u, spreads=g)
distro.dots(x=X, jit.f=3)
```

geomean *Calculate the Geometric Mean*

Description

A trivial one-line function for exp(mean(log()))

Usage

```
geomean(x)
```

Arguments

x	a vector of numeric values

Value

the geometric mean (a scalar value)

See Also

geometric.mean

Examples

```
geomean(rnorm(20,5))
```

groupBy *Group a datafame by a factor and perform aggreate functions.*

Description

The R equvalent of a SQL 'group by' call.

Usage

```
groupBy(df, by, aggregation,  clmns=names(df), collapse=',',
                distinct=FALSE, sql=FALSE, full.names=FALSE, ...)
```

Arguments

df	a data frame.
by	the factor (or name of a factor in df) used to determine the grouping.
aggregation	the functions to perform on the output (default is to sum). Suggested functions are: 'sum','mean','var','sd','max','min','length','paste',NULL.
clmns	the colums to include in the output.
collapse	string delimiter for columns aggregated via 'paste' concatenation.
distinct	used in conjunction with paste and collapse to only return unique elements in a delimited concatenated string
sql	whether or not to use SQLite to perform the grouping (not yet implimented).
full.names	names of the aggregation functions should be appended to the output column names
...	additional parameters (such as na.rm) passed to the underlying aggregate functions.

Value

an summary/aggregate dataframe

See Also

aggregate, bestBy

Examples

```
df <- data.frame(a=runif(12),b=c(runif(11),NA),
                 z=rep(letters[13:18],2),w=rep(letters[20:23],3))

groupBy(df=df, by='w', clmns=c(rep(c('a','b'),3),'z','w'),
 aggregation=c('sum','mean','var','sd','min','max','paste','length'),
 full.names=TRUE, na.rm=TRUE)
# or using SQLite
groupBy(df=df, by='w', clmns=c(rep(c('a','b'),2),'z','w'),
        aggregation=c('sum','mean','min','max','paste','length'),
        full.names=TRUE, sql=TRUE)


## passing a custom function
meantop <- function(x,n=2, ...)
  mean(x[order(x, decreasing=TRUE)][1:n], ...)

groupBy(df, by='w', aggregation=rep(c('mean','max','meantop'),2),
                    clmns=rep(c('a','b'),3), na.rm=TRUE)
```

`heatmatrix` *Simple Heatmap Plot*

Description

This is a very simplified heatmap function: basically a convienent wrapper around the 'image' function.

Usage

```
heatmatrix(x, values=TRUE, clp=c('bottom','top'), rlp=c('left','right'),
                    xadj=.02, yadj=.3, ylab.cntr=FALSE, cex=1, cex.axis=1, ...)
```

Arguments

`x`	A matrix
`values`	boolean: should the values be plotted over each cell?
`clp`	column label position: either 'bottom' or 'top'.
`rlp`	row label position: either 'right' or 'left'
`xadj`	x-adjust of the row labels
`yadj`	y-adjust of the column labels.
`ylab.cntr`	boolean for justification of row labels.
`cex,`	character expansion factor for values in cells if values == true
`cex.axis`	character expansion factor for axis tick mark labels
`...`	other paramters passed on to image()

Value

a heatmap plot

See Also

image, heatmap, heatmap.2

Examples

```
data(mtcars)
x  <- as.matrix(mtcars)

heatmatrix(x)
```

hyperplot *Annotate Outliers in a Scatterplot via an HTML Image-Map*

Description

This simple function makes R scatter plots interactive by creating an image and wrapping HTML around it: creating a hyperlinked hyperplot. Hover over the points to see what each is. Click to connect to a table below that will tell you more about each point (if browse ==TRUE).

Usage

```
hyperplot(x, y = NULL, annout = 1:length(x),
                  name ="hyperplot.imagemap", w = 72 * 8, h = 72 * 6,
                  link ="internal", browse = TRUE, cex = 1, ...)
```

Arguments

x	a plot-able object, a numeric vector or the name of a numeric vector column in annout.
y	a numeric vector or the name of a numeric vector column in annout. Must be the same length as x.
annout	a named data.frame or table of outliers to annotate the points in the plot. 'x' and 'y' params can indicate column names or numbers of annout.
name	base name of the image & html (map) page that get generated.
w	width of the png image in inches.
h	height of the png image in inches.
link	create a linked lookup table from image to the annout table.
browse	load the html page automatically via R.
cex	character expansion for points
...	other paramters passed on to plot()

Value

HTML page with annotation mapped image

See Also

browseURL

Examples

```
if(capabilities()["png"] && interactive()) {

main.hov <- 'Hover over a point to see the name'
main.subsets <- '(annotated subset in red only)'
main.click.in <- 'click on points to visit table'
main.click.out <-  'click on points to visit external site'
cols <- c('black','red')
ext.url <- 'http://cran.r-project.org'

################################
## x and y as numeric vectors ##
################################
x.out <- nv(rnorm(13,2,sd=5),toupper(letters)[1:13])
x.in <- nv(rnorm(13,1,sd=.5),toupper(letters)[14:26])
y.out <- nv(rnorm(13,2,sd=5),toupper(letters)[1:13])
y.in <- nv(rnorm(13,1,sd=.5),toupper(letters)[14:26])
x <- c(x.out, x.in)
y <- c(y.out, y.in)

## simplest version
hyperplot(x,y, main=main.hov)

## same but with annotations being supplied as a paramter (instead of names on x)
names(x) <- NULL
hyperplot(x,y, annout=toupper(letters), main=main.hov)

## annotate only a subset
hyperplot(x,y, annout=1:13, col=cols[rep(c(T,F), each=13)+1],
  main=paste(main.hov,main.subsets,sep='\n'))


########################
## annout as dataframe #
########################
## x and y as vectors
x <- nv(x,toupper(letters)) # reinstate the names of x
df <- data.frame(ab=rep(c('a','b'),13),row.names=toupper(letters))
hyperplot(x,y, annout=df,
                main=paste(main.hov, main.click.in, sep='\n'))

## x and y as names of columns in df
df <- cbind.data.frame(data.frame(x=x, y=y), df)
hyperplot(x='x',y='y', annout=df,
                main=paste(main.hov, main.click.in, sep='\n'))

## using 'link' column name parameter to specify external links
df <- cbind.data.frame(df,
            data.frame(url=ext.url, stringsAsFactors=FALSE))
hyperplot(x='x',y='y', annout=df, link='url',
                main=paste(main.hov, main.click.out,sep='\n'))
```

```
## using reserved column name 'out' as a way to annotate a subset
df <- cbind.data.frame(df, data.frame(out=rep(c(TRUE,FALSE), each=13)))
hyperplot(x='x',y='y', annout=df, col=cols[df$out+1],
                main=paste(main.hov, main.click.in, main.subsets,sep='\n'))



}
```

`install.prev.pkg` *Install the next oldest package*

Description

This funciton will recursively search the package archives backwards in time until it finds a version that installs successfully. This function is useful for installing or troubleshooting package dependency trees where one or more packages "require" the very most recent version of R. Rather than upgrading your base R installation, you can use this function to search back in time until you find a version of the package that works with your existing version of R.

Usage

```
install.prev.pkg(pkg.nm, version=NULL,
  repo.url='https://cran.r-project.org/src/contrib/Archive/')
```

Arguments

`pkg.nm`	The package name.
`version`	The version number as #.#-# or #.#.# .
`repo.url`	The base url for archives of old versions of packages on CRAN.

Value

NULL

Examples

```
#install.prev.pkg('mnormt')
#install.prev.pkg('mvtnorm')
```

labsegs *Buffered Segments for Point Labels*

Description

This function is a wrapper for segments which trigonometrically shortens the lines that are near the "1" end so as not to cluter or overplot the text label it is attached to.

Usage

```
labsegs(x0, y0, x1, y1, buf=.3, ...)
```

Arguments

x0	initial x point coordinate
y0	initial x point coordinate
x1	initial x point coordinate
y1	initial x point coordinate
buf	the buffer between the label at point "1" and the actual segment
...	other parameters passed to segments.

See Also

segments

Examples

```
x <- rnorm(1000,0,.5)
y <- rnorm(1000,-.3,.15)

labdb <- data.frame(x=seq(-.5,.5,by=.5), y=rep(.85,3))
xlims <- c(-1,1)
ylims <- c(-.5,1)

x0.lbd <- x[rev(order(y))][1:3]
y0.lbd <- y[rev(order(y))][1:3]

par(mfrow=c(1,2))
plot(x,y, xlim=xlims, ylim=ylims, main='segments')
segments(x0=x0.lbd, y0=y0.lbd, x1=labdb$x, y1=labdb$y, col=rainbow(3), lwd=3)
text(x=labdb$x, y=labdb$y, labels=letters[1:3], cex=3, col=rainbow(3))

plot(x,y, xlim=xlims, ylim=ylims, main='labsegs')
labsegs(x0=x0.lbd, y0=y0.lbd, x1=labdb$x, y1=labdb$y, col=rainbow(3), lwd=3, buf=.07)
text(x=labdb$x, y=labdb$y, labels=letters[1:3], cex=3, col=rainbow(3))
```

leghead *Generate a Color Coded Legend dataframe via head and sum.*

Description

'leghead' is part 'head' and part 'summary'. It works best on a sorted dataframe where all you are interestesd in only the most (or least) abundant rows. An ideal place to use it is in a legend for ~lognormally distributed data. Additionally, an optional row-wise color coding column is added (the color 'gray' is used for missing row names).

Usage

```
leghead(x, n=7, tabulate=FALSE, colors=TRUE, na.name='NA',
        na.col = "white", other.col = "gray",  na.last = TRUE)
```

Arguments

x	dataframe or table you wish to summarize
n	the number of rows you wish to display as is
colors	list of vectors or a dataframe
tabulate	the column name to tabulate on if x is an untabulated dataframe and FALSE otherwise
na.name	the new rowname for a row with a missing name
na.col	color for rows labeled as 'NA'
other.col	color for the rows labeled as 'unknown'
na.last	boolean specifying if the na category should be listed last in the table.

Value

A truncated dataframe with a new bottom row summarizing all the truncated ones.

See Also

summary, head, sstable

Examples

```
 e <- data.frame(a=runif(12),b=runif(12), z=rep(letters[13:18],2),w=rep(letters[20:23],3))
tab <- sstable(e, idx.clmns=c('z'), ct.clmns=c('a','b'))
lh <- leghead(tab)
plot(x=lh$a, y=lh$b, cex=lh$sum*3, col=lh$color, pch=20)
legend('topleft',legend=rownames(lh), col=lh$color, pch=20)
```

m *Regexp Match Operator*

Description

A grep/sub-like function that returns one or more back-referenced pattern matches in the form of a vector or as columns in a dataframe (respectively). Unlike sub, this function is more geared towards data extraction rather than data cleaning. The name is derived from the popular PERL regular expression 'match' operator function 'm' (eg. 'extraction =~ m/sought_text/').

Usage

```
m(pattern, vect, names="V", types="character", mismatch=NA, ...)
```

Arguments

pattern	A regular expression pattern with at least one back reference.
vect	A string or vector of strings one which to apply the pattern match.
names	The vector of names of the new variables to be created out of vect. Must be the same length as vect.
types	The vector of types of the new variables to be created out of vect. Must be the same length as vect.
mismatch	What do to when no pattern is found. NA returns NA, TRUE returns original value (currently only implimented for single match, vector returns)
...	other parameters passed on to grep

Value

Either a vector or a dataframe depending on the number of backreferences in the pattern.

See Also

sub, gsub, regexpr, grep, gregexpr.

Examples

```
## single vector output examples
m(pattern="asdf.([A-Z]{4}).",
  vect=c('asdf.AS.fds','asdf.ABCD.asdf', '12.ASDF.asdf','asdf.REWQ.123'))


Rurls <- c('http://www.r-project.org',    'http://cran.r-project.org',
           'http://journal.r-project.org','http://developer.r-project.org')
m(pattern="http://([a-z]+).r-project.org", vect=Rurls)
```

```
# dataframe output examples

data(mtcars)
m(pattern="^([A-Za-z]+) ?(.*)$",
  vect=rownames(mtcars), names=c('make','model'), types=rep('character',2))
```

---

makeElipseCoords *Make Elipse Coordinates*

Description

Create x & y coordinates for an elipse from parameters. save.

Usage

```
makeElipseCoords(x0 = 0, y0 = 0, b = 1, a = 1, alpha = 0, pct.range = c(0,1), len = 50)
```

Arguments

x0	x coordinate of center of elipse.
y0	y coordinate of center of elipse.
b	y axis stretch factor.
a	x axis stretch factor.
alpha	rotation factor.
pct.range	percentage of the way around the elipse.
len	number of points used to draw elipse.

Value

a 2 column (x and y) dataframe with coordinates for drawing an elipse

Examples

```
makeElipseCoords(x0 = 0, y0 = 0, b = 1, a = 2, alpha = 0)
```

`mvlabs` *Move Text Labels Interactively*

Description

There is no easy way to move point labels around interactively on an a plot in R. This function allows a point and click way to select (using identify) and move (using locator) points by modifying the underlying dataframe.

Usage

```
mvlabs(df, n=nrow(df), x='x', y='y', l='lab', cols=colors()[grep("dark",colors())], ...)
```

Arguments

`df`	A dataframe with x and y coordinates and text labels
`n`	the number of points you wish to move
`x`	the column name of the x axis coordinates
`y`	the column name of the y axis coordinates
`l`	the column name of the point labels
`cols`	the color vector to iterate through while assigning new positions.
`...`	other paramters passed on to text

Value

a series of violin plots

See Also

locator,identify,labsegs

Examples

```
x <- rnorm(20); y <- rnorm(20)
df <- data.frame(x,y, lab=as.character(letters[1:20]))
plot(df$x, df$y, pch=''); text(df$x, df$y, df$lab)
## df <- mvlabs(df, 'x','y','lab', n=3)
plot(df$x, df$y, pch=''); text(df$x, df$y, df$lab)
```

nerge *Named Merge*

Description

This function is a wrapper for merge that supports merging multiple vectors and or dataframes

Usage

```
nerge(l, ...)
```

Arguments

l	A named list of named vectors (and/or dataframes)
...	Other parameters passed on to each sub-merge

See Also

merge

Examples

```
df <- data.frame(a=c(6,7,8), b=c(9,8,7))
rownames(df) <- c('a','d','c')

l <- list(x=nv(c(1,2),c('a','b')),y=nv(c(2,3),c('b','d')),z=nv(c(4,1),c('c','d') ), w=df)
nerge(l, all=TRUE)


l2 <- list(a=nv(c(1.23, 1.423, 2.343), c('z','y','x')),b=nv(c(6.34,7.34, 12.545),c('z','w','y')))
nerge(l2, all=TRUE)
```

nv *Create a named vector from a dataframe, table or vector*

Description

The '$' or "[," operators for dataframes and tables do not carry along with them the row names. This function provides a solution for this problem. Additionally this function will accept a vector of values and a corresponding vector of value names–an ideal, in-line way for setting named-vectors as default parameters in new functions.)

Usage

```
nv(x, name)
```

Arguments

x	The source dataframe, table, vector, or factor
name	The column name you would like to pull out as a named vector. OR the names of the vector (if x is a vector)

Value

a named vector or factor

Author(s)

David Schruth

See Also

vector,name

Examples

```
## example 1: pulling a row.named vector out of a dataframe
df <- data.frame(a=c(1,2,34,4,5,3,23), b=c('z','x','y','z','x','n','p'))
rownames(df) <- letters[1:nrow(df)]
nv(df,'a')
nv(df,'b')

## example 2: a naming vectors from scratch
nv(c(1,2,3), c('a','b','c'))
nv(df$a, df$b)
```

---

pad *Pad a vector of numerical string with zeros.*

Description

This function helps to pad numbers on the left side with zeros so that they may be used to create strings used in filesystem names (for example).

Usage

```
pad(vect,np)
```

Arguments

vect	a vector of strings representing numbers.
np	number of zeros to pad to the left of the string.

Value

a (vector of) string(s) with np number of zeros padded on as a prefix

Author(s)

Jeremy Tantrum

Examples

```
pad(c(1,10,1000,10000), 4)
```

---

parseArgString *Process Command Line Arguments*

Description

generic function for parsing delimited lists from BATCH mode argument strings.

Usage

```
parseArgString(string, delimiter=',', min.param.ct=2, max.param.ct=2, param.range=NULL)
```

Arguments

string	string to parse.
delimiter	how the string is delmited into a vector.
min.param.ct	minimum number of parameters in the vector.
max.param.ct	maximum number of parameters in the vector.
param.range	the range of the parameter values.

Value

a vector or value that has been check for validity

Examples

```
## passes
parseArgString('apple,banana,pear', param.range=c("apple","banana","pear","pineapple"))
parseArgString('1,2,3', param.range=c(1,4))


## fails
## Not run:
parseArgString('apple,banana,pear', param.range=c("apple","banana"))
parseArgString('1,2,3', param.range=c(1,2))

## End(Not run)
```

---

pct *Add Percentage Columns to a Dataframe*

Description

This function will add extra columns to an existing dataframe. The second argument 'clmn' should specify which column(s) of the dataframe the percentage should be calculated by dividing each column's row-element by it's sum.

Usage

```
pct(df, clmns)
```

Arguments

df	A dataframe with numeric columns.
clmns	the names of the columns for which the percentage column should be calculated from.

Value

The original dataframe plus extra percentage columns corresponding to original columns in the dataframe.

Examples

```
df <- data.frame(a=c(1,2,3), b=c('x','y','z'), c=c(5,3,2))
pct(df, c('a','c'))
```

pies *Pie chart scatterplot*

Description

Plot pie charts in an XY scatterplot. An overhauled wrapper of the original pie plot function. It is currently very slow: a recommened work around is to plot to something other than the default device (aka png, pdf, etc).

Usage

```
pies(x, show.labels = FALSE, show.slice.labels = FALSE, color.table = NULL,
radii = rep(2,length(x)), x0=NULL, y0=NULL,
edges = 200,  clockwise = FALSE,
              init.angle = if (clockwise) 90 else 0, density = NULL, angle = 45,
                border = NULL, lty = NULL,
                other.color='gray', na.color='white', ...)
```

Arguments

x	a list of named vectors.
show.labels	boolean specifying if the pie point lables should be plotted.
show.slice.labels	boolean specifying if the pie slice labels should be plotted.
color.table	a named vector of colors. names should correspond to all possible levels of x
radii	a vector of radii used to size the pie points.
x0,y0	a vector of x and y positions for the pie points.
edges	the circular outline of the pie is approximated by a polygon with this many edges.
clockwise	logical indicating if slices are drawn clockwise or counter clockwise (i.e., mathematically positive direction), the latter is default.
init.angle	number specifying the starting angle (in degrees) for the slices. Defaults to 0 (i.e., 3 o'clock) unless clockwise is true where init.angle defaults to 90 (degrees), (i.e., 12 o'clock).
density	the density of shading lines, in lines per inch. The default value of NULL means that no shading lines are drawn. Non-positive values of density also inhibit the drawing of shading lines.
angle	the slope of shading lines, given as an angle in degrees (counter-clockwise).
border	(possibly vectors) arguments passed to polygon which draws each slice.
lty	(possibly vectors) arguments passed to polygon which draws each slice.
other.color	color used for x vector elements for names without corresponding names in the color table
na.color	color used for x vector elements with missing names
...	other arguments passed to polygon

Value

Pie charts as points on a plot

See Also

pie

Examples

```
## these examples are to the default plot window, which can be slow
## try instead to plot to png or pdf for example

## example 1
  pies(
       list(
            a=nv(c(1,2,3),c('one','two','thre')),
            b=nv(c(2,2,3),c('one','two','thre')),
            c=nv(c(1,2,3),c('one','two','thre'))
            ),
       x0=c(0,.5,1),
       y0=c(0,.5,1), radii=6, border=c('gray', 'black', 'red')
       )


## example 2
n <- 200
n.groups <- 10
n.subgroups <-  6

grps <- paste('gene',seq(1,n.groups), sep='')[round(runif(n,1,n.groups))]
subgrps <- paste('species',seq(1,n.subgroups), sep='')[round(runif(n,1,n.subgroups))]
group.df <- cbind.data.frame(grps,subgrps)
subgroup.list <- by(group.df, group.df$grps, function(x) x$subgrps)

pie.list <- lapply(subgroup.list, table)
col.tab <- nv(rainbow(6), unique(subgrps))

pies(x=pie.list, x0=rnorm(n.groups), y0=rnorm(n.groups),
     radii=10, show.labels=TRUE, show.slice.labels=TRUE, color.table=col.tab)


## example 3  reading from external flat file
## salt.df <- read.delim('/path/to/my/file.tab')
## create a dummy dataset that might live inside the above file
salt.df <- data.frame(salinity=rnorm(25,5), temperature=rnorm(25,25),spec_a=rpois(25,4),
   spec_b=rpois(25,4),
   spec_c=rpois(25,4),
   spec_d=rpois(25,4),
   spec_e=rpois(25,4)
   )
## pull out the colnumn names that are specific to pie wedge numbers
```

```
salt.spec.nms <- names(salt.df)[grep('spec',names(salt.df))]
## turn them into a list
pie.list <- lapply(1:nrow(salt.df),
 function(i) as.table(nv(as.vector(as.matrix(salt.df[i,salt.spec.nms])),salt.spec.nms)))
names(pie.list)<- letters[1:25]
with(salt.df, pies(x=pie.list, x0=salinity, y0=temperature, radii=2))
```

`plot.sparge` *Visually compare all points from different univariate distributions*

Description

Visually compare continuuous univariate distributions using jittered and transparent points.

Usage

```
## S3 method for class 'sparge'
plot(x, jit.f=1, zl=TRUE, xlim=range(unlist(unlist(x))),
                  add=FALSE, pd=0, box.brdrs='gray', col=1, alpha=.3, ...)
```

Arguments

`x`	a list of vectors each with values drawn from the same distribution
`jit.f`	factor for random jittering (see 'jitter()'
`zl`	should we add a horizontal [zero] line at x=0?
`add`	should we add to the existing plot?
`pd`	'position dodge' moves all y axis plotting positions up or down by this factor
`xlim`	x axis plot limits
`alpha`	transparency level for [overlapping] points
`box.brdrs`	the color of the borders of the box plots surrounding all distributions
`col`	(vector of) [base] colors of the points of the distribution(s)
`...`	other parameters passed on to plot

Value

a 'sparge' [sprinkle/smear] plot of point distributions

See Also

See also 'boxplot' and 'stripchart' in package 'graphics' as well as 'violin', 'bean', 'ridgelines', and 'raincloud' plots.

Examples

```
N=300
x=lapply(sample(1:5), function(avg) (rnorm(N,avg)))
plot.sparge(x, col=rep('blue',length(x)), main='sparge plots:\nfor distributional comparison')
```

`plotClock` *Plot a simple clock.*

Description

Used to create a clock on a plot as a way to keep track of the additional parameter of time for use in animated movies of multiple plots.

Usage

```
plotClock(hour, minute, x0 = 0, y0 = 0, r = 1)
```

Arguments

`hour`	integer specifying the position of the hour hand.
`minute`	integer specifying the position of the minute hand.
`x0`	number specifying the x position of the clock.
`y0`	number specifying the y position of the clock.
`r`	number specifying the radius of the clock.

Value

a plot of a clock

`raAddArms` *Add Arms to a RA plot.*

Description

.

Usage

```
raAddArms(epsilon=.55, start=1, end=6, A.shift=0, R.shift=0, ...)
```

Arguments

epsilon	.
start	.
end	.
A.shift	.
R.shift	.
...	other parameters passed to lines.

See Also

raPlot

---

raAddAxLabs *Add axis labels to an RA plot.*

Description

.

Usage

```
raAddAxLabs(conditions=nv(c('a','b'),c('ref','obs')), normalize=T, add=TRUE, line=2)
```

Arguments

conditions	.
normalize	.
add	.
line	.

See Also

raPlot

raAddSigLines *Add Significance Lines to an RA plot.*

Description

.

Usage

```
raAddSigLines(n, end=20, alpha=1e-3, nr=0, A.shift=0, plot=FALSE, ...)
```

Arguments

n	.
end	.
alpha	.
nr	a numeric value indicating the asymptotic normalization ratio line.
A.shift	.
plot	.
...	other parameters passed to lines.

See Also

raPlot

raPlot *Generate a Ratio Average [RAy] Plot.*

Description

A plot which turns two vectors of count data into log scaled fold change ratio and average abundance. The plot derives from a Bland-Altman plot and is also very similar to an MA plot. The RA plot is unique, however, in it's creative inclusion of the vector-unique 'arms' which are artificially introduced into the plot by adding a <1 epsilon factor before the log function is applied. The name RAy comes from the fact that the aforementioned 'uniques' arms addition makes it strongly resemble a geometric ray. Many of the parameters to the function play off of this convenient anatomical analogy.

Usage

```
raPlot(a, b=NULL, uniques=5, normalize=FALSE,
        nr=0, alpha = 0.01, jitter=FALSE, jit.wgts=NULL,
      rex=1, flat=TRUE, tail=.5, arms=.5, spine=1, border=NULL, plot=TRUE,  ...)
```

Arguments

a	a vector of counts for a. can also be a matrix with two columns 1 for a and 2 for b.
b	a vector of counts for b.
uniques	a boolean specifying whether or not to plot the library-unique genes (those with zero counts in one or the other library).
normalize	A boolean specifying whether or not to normalize the counts into proportions.
nr	a numeric value indicating the asymptotic normalization ratio line.
alpha	a statisical significance value.
jitter	whether or not or how much to jitter the a and b counts into surrounding, non-overlapping space.
jit.wgts	a weight vector used to spread the counts of a and b into surrounding, non-overlapping space.
rex	a numeric value specifying the radial expansion of the plot points.
flat	a boolean for the radial expansion of points as a function of both R and A axes.
tail	a numeric or boolean value indicating the line thickness of the two trailing curved significance lines of the RAy.
arms	a numeric or boolean value indicating the line thickness of the two leading straight separator lines of the RAy.
spine	a numeric or boolean value indicating the line thickness of the normalization line (whose y position is specified by mm).
border	a vector of strings used to color the borders of the points.
plot	whether or not to do the actual plot.
...	other parameters passed to plot.

Value

a RAy plot

See Also

limma::plotMA, edgeR::maPlot

Examples

```
a <- rnbinom(n=10000, mu=5, size=2)
b <- rnbinom(n=10000, mu=5, size=2)

## the alternative
plot(a,b)
## the raPlot version
raPlot(a, b)

## highlight the condition unique points in the same way as  edgeR's "maPlot"
```

```
RA <- raPlot(a, b, pch='')
cond.unique <- apply(cbind(a,b), 1, function(d) any(d==0))
points(RA$A,RA$R, col=c('black','orange')[cond.unique+1])

## try playing with jittering over plotted points
raPlot(a, b, jitter=.3)
```

`read.tab` *Read in a Tab Delimited File.*

Description

This function is a slight (genome annotation friendly) variant of the built-in read.delim function in R. Two non-standard defaults have been set: stringsAsFactors=TRUE, quote="". An additional parameter "check.row.ct", triggering a count.fields call, has been added to further ensure the integrity of large data files.

Usage

```
read.tab(file, check.row.ct = TRUE, stringsAsFactors = FALSE,
               quote = "", header=TRUE, ...)
```

Arguments

`file`	the name of the file which the data are to be read from.
`check.row.ct`	logical: use 'count.fields' to independently verify the number of rows read.table reads into memory?
`stringsAsFactors`	logical: should character vectors be converted to factors?.
`quote`	the set of quoting characters.
`header`	boolean specifying if the first row serves as labels for the columns
`...`	other paramters passed to read.delim.

Value

a dataframe.

regroup *Regroup a dataframe.*

Description

Used to group a dataframe of numbers by a factor that need not be the same length. Find the a factor in the old df and use it to group by the new trumping factor (NA's allowed)

Usage

```
regroup(df, old, new, clmns, funcs=rep('sum',length(clmns)), combine=TRUE)
```

Arguments

df	a dataframe.
old	the ids to match the rows in df to the 'new' grouping ids.
new	the new ids (must be a vector of the same length as 'old'.
clmns	the colums to include in the output.
funcs	the functions to perform on the output (default is to sum) .
combine	Determines wether to combine with existing groupings or to start fresh.

Value

a dataframe with number of rows equal to the number of factor levels in 'new'

Examples

```
df <- data.frame(a=rnorm(20),b=rpois(20,1))

mapping <- data.frame(old=rownames(df), new=rep(c('a','b'),10))
regroup(df, old=mapping$old, new=mapping$new)
```

rerowname *Rename select rows of a dataframe*

Description

Used to easily rename the rows of a dataframe.

Usage

```
rerowname(df, old='NA', new='unknown')
```

Arguments

df	A dataframe with rownames.
old	The row name to be replaced.
new	The replacment row name.

Value

A dataframe with one new rowname

Examples

```
df <- data.frame(a=c(1,2,3), b=c('x','y','z'), c=c(5,3,2))
rownames(df) <- c('p','q','NA')
rerowname(df)
```

---

spie *Spie charts*

Description

Spie Chart

Usage

```
spie(p1, p2, init.angle=pi, multi, col = rainbow(length(x$radii)), bg=col, lwd=2,
          pie.labs=TRUE, grid=TRUE, grid.labs=TRUE, scale=TRUE, p1.circle=TRUE)
```

Arguments

p1	a positive numeric vector.
p2	a positive numeric vector. Angles are the same than those used for the first pie but radii change according to the values in .
init.angle	initial angle
multi	radius scale multiplier
col	colors of the p2 (foreground) slices
bg	colors of the p1 (background) slices
lwd	line width of the pie wedge boundaries
pie.labs	boolean labels for the pies
grid	boolean
grid.labs	boolean, scale indicators
scale	boolean
p1.circle	boolean

Author(s)

Romain Francois <francoisromain@free.fr> & David Schruth <dschruth@uw.edu>

References

D. G. Feitelson (2003), "Comparing Partitions with Spie Charts". School of Computer Science and Engineering, The Hebrew University of Jerusalem.

Michael Friendly (2022), Spie chart – a comparison of two pie charts.

See Also

pie

Examples

```
p1  <- c(0.12, 0.3, 0.26, 0.16, 0.04, 0.12)
p2<- c(0.06, 0.15, 0.52, 0.14, 0.08, 0.05)
plot(p1, p2, multi=c(.5, 1, 1.5, 2))
```

---

sstable *Sum Sorted Tabulation*

Description

A wrapper for the "table()" function that also calculates the row-wise sum and sorts by the new column.

Usage

```
sstable(x, idx.clmns, ct.clmns = NULL, na.label = "NA")
```

Arguments

x	list of vectors or a dataframe
idx.clmns	index columns
ct.clmns	count columns
na.label	row label used for na columns

Value

A dataframe sorted by the count columns.

Author(s)

David Schruth

See Also

ledghead, table, order, sort

Examples

```
e <- data.frame(a=runif(12),b=runif(12), z=rep(letters[13:18],2),w=rep(letters[20:23],3))
e <- data.frame(a=runif(10),b=runif(10), z=rep(letters[12:16],2),w=rep(letters[20:24],2))
 sstable(e, idx.clmns=c('z','w'), ct.clmns='a')
 sstable(e, idx.clmns=c('z'), ct.clmns=c('a','b'))
 sstable(e, idx.clmns=c('z','w'))
 e <- data.frame(a=10,b=0, z=as.factor(NA))
 sstable(e, 'z', c('a','b'))
 e <- data.frame(a=10,b=0, z=NA, w=NA)
 sstable(e, 'z', c('a','b'))
 e <- data.frame(a=runif(10),b=runif(10),m=rep(c('one','two'),5),
                 z=factor(rep('z',10), levels=c('z','x')))
 sstable(e, idx.clmns=c('m','z'))
```

---

tab2df *Table to Data Frame*

Description

Convert a table to a dataframe while perserving the same number of columns and rows and names of each.

Usage

```
tab2df(x, ...)
```

Arguments

x	a table or matrix class object (output from the table command).
...	other arguments passed to data.frame(...).

Value

a dataframe

See Also

table

Examples

```
x <- data.frame(a=runif(10),b=runif(10), z=rep(letters[1:5],2))
as.data.frame(x)
tab2df(x)
x <- nv(rnorm(10), letters[1:10])
  as.data.frame(x)
tab2df(x)
x <- nv(rnorm(2), c('x.b','y.b'))
  as.data.frame(x)
tab2df(x)
x <- nv(rnorm(2), c('b.x','b.y'))
  as.data.frame(x)
tab2df(x)
e <- data.frame(a=runif(10),b=runif(10), z=rep(letters[13:17],2))
x <- as.table(sapply(c('a','b'),function(cc) by(e[,'a'],list(e$z), sum)))
  as.data.frame(x)
tab2df(x)
x <- as.table(by(1:10, list(a=rep(1:5,2),b=rep(1:2,5)), sum))
  as.data.frame(x)
tab2df(x)
x <- as.table(nv(c(54,34), c('a','b')))
  as.data.frame(x)
tab2df(x)

x <- table(a='x',b='y')
tab2df(x)
```

---

textplot *A Text-Only Plot*

Description

Generate a new plot window with just text centered in the middle. This is ideally used in conjunction with the 'layout' command to label columns and rows of the grid.

Usage

```
textplot(..., x=1, y=1)
```

Arguments

...	parameters passed to the 'text' function
x	the x position of the text.
y	the y position of the text.

Value

A new plot window wiht just text

See Also

layout, text

Examples

```
layout(rbind(c(1,1,1),c(2,3,4), c(5, 6,7)),
       widths=c(5, 10,10) , heights=c(5, 10,10))
textplot('title', cex=2)
textplot('row 1', srt=90, cex=2)
plot(1,2)
hist(c(1,2,34,4,3,2,2))
textplot('row 2', srt=90, cex=2)
pie(c(1,23,3,1,1,2,3,4,54,5))
plot(c(1,2,4,4,23,2), c(1,2,4,3,2,2))
```

---

usr2lims *Grab and adjust the current plot dimensions*

Description

This is a simple function which grabs the current plot dimentions and adjusts them by shrinking them by 4

Usage

```
usr2lims(adj=.04)
```

Arguments

adj	The automatic adjustment factor 'plot' adds to buffer the specified plot dimentions.

Value

A 2 item (x and y) list of 2 item (min and max) vectors for x and y limits of the current plot area

See Also

par

Examples

```
plot(c(0,1), c(0,1))
usr2lims()
```

vennMatrix *Create a Venn Ready Matrix out of a List of Factors*

Description

The limma package has great functions for making venn diagrams from a matrix. This function is provides upstream functionality to turn a list of factors into this required input format.

Usage

```
vennMatrix(l)
```

Arguments

l	a named list of factors

Value

a matrix with columns for list elements and rows with globally unique factor levels

See Also

venCounts

Examples

```
l <- list(a=factor(c('x','y','z')), b=factor(c('w','x','v')))

vennMatrix(l)
```

wjitter *Weighted Jitter*

Description

Use weights to jitter values away fromt their current value.

Usage

```
wjitter(x, w, amount=.43)
```

Arguments

x	a vector of values
w	a vector of weights of the same lenght as x
amount	the amount to jitter (passed to the parameter by the same name in the jitter function)

Value

A weighted jittered vector of the same length as x

Examples

```
x <- seq(1,20)
w <- runif(20, 0,1)
plot(x,wjitter(w,x))
```

---

write.delim *Write a (tab) delimited text file.*

Description

A simple wrapper for write.table with the same options as read.delim

Usage

```
write.delim(df, file, quote = FALSE, row.names = FALSE, sep = "\t", ...)
```

Arguments

df	a dataframe.
file	outputfile path.
quote	should elements of the dataframe be quoted for output.
row.names	should the output include rownames.
sep	the delimiter between fields.
...	other parameters passed to write.table.

Value

A tab delimited text file

See Also

read.delim

Examples

```
## Not run:
x <- data.frame(a = I("a \" quote"), b = pi)
write.delim(x, file = "foo.tab")

## End(Not run)
```

Index