Package: smvgraph (via r-universe)

November 4, 2024

Type Package

Title Visualization and Clustering of Data in a Shiny App

Version 0.2.0

Date 2022-12-04

Description Various visualisations of univariate and multivariate graphs (e.g. mosaic diagram, scatterplot matrix, Andrews curves, parallel coordinate diagram, radar diagram and Chernoff plots) as well as clustering methods (e.g. k-means, agglomerative, EM clustering and DBSCAN) are implemented as a Shiny app. The app allows interactive changes, e.g. of the order of variables. It is intended for use in teaching.

License GPL-3

URL https://github.com/sigbertklinke/smvgraph (development version)

Encoding UTF-8

Imports tools, devtools, formatR, highlight, shiny, DT, base64enc, shinyWidgets, shinydashboardPlus, sortable, mrfDepth

RoxygenNote 7.2.2

Roxygen list(markdown = TRUE)

Suggests MASS, knitr, rmarkdown

Depends R (>= 2.10)

LazyData true

VignetteBuilder knitr

Repository https://sigbertklinke.r-universe.dev

RemoteUrl https://github.com/sigbertklinke/smvgraph

RemoteRef HEAD

RemoteSha 8f713920a32cbeab3f8032a58458080d7d45cef9

Contents

`andrews` *andrews*

Description

Andrews curves for visualization of multidimensional data. `step` determines the number of line segments for each curve. If `ymax==NA` then the maximum y coordinate will be determined from the curves. Note that for `type==3` the x range is $[0, 4 * pi]$ otherwise $[-pi, pi]$. Observations containing `NA`, `Nan`, `-Inf`, or `+Inf` will be deleted before plotting

Usage

```
andrews(x, type = 1, step = 100, ..., normalize = 1, ymax = NA)
```

Arguments

`x`	data frame or matrix
`type`	type of curve (default: `1`) • 1: $f(t) = x1/(2^0.5) + x2 * sin(t) + x3 * cos(t) + x4 * sin(2 * t) + x5 * cos(2 * t) + ...$ • 2: $f(t) = x1 * sin(t) + x2 * cos(t) + x3 * sin(2 * t) + x4 * cos(2 * t) + ...$ • 3: $f(t) = x1 * cos(t) + x2 * cos((2 * t)^0.5) + x3 * cos((3 * t)^0.5) + ...$ • 4: $f(t) = 1/(2^0.5) * (x1 + x2 * (sin(t) + cos(t)) + x3 * (sin(t) - cos(t)) + x4 * (sin(2 * t) + cos(2 * t)) + x5 * (sin(2 * t) - cos(2 * t)) + ...)$
`step`	smoothness of curves
`...`	further parameters given to graphics::plot and graphics::lines
`normalize`	integer: normalization method (default: `1`) • 0: no rescaling • 1: $(x - min(x))/(max(x) - min(x))$ • 2: $(x - mean(x))/sd(x)$
`ymax`	numeric: maximum of y coordinate (default: `NA`)

Value

nothing

References

- Andrews, D. F. (1972) Plots of High-Dimensional Data. Biometrics, vol. 28, no. 1, pp. 125-136.
- Khattree, R., Naik, D. N. (2002) Andrews Plots for Multivariate Data: Some New Suggestions and Applications. Journal of Statistical Planning and Inference, vol. 100, no. 2, pp. 411-425.

See Also

In package andrews or at CRAN

Examples

```
andrews(iris[,-5], col=as.factor(iris[,5]))
andrews(iris[,-5], type=4, col=as.factor(iris[,5]), ymax=2)
```

---

as_param *as_param*

Description

Create a parameter list or a function call. For a function call fun must be explicitly given.

Usage

```
as_param(..., fun = NULL)

txt(x)
```

Arguments

...	list of named and unnamed parameters
fun	character:
x	character: replaces "x" by "'x'"

Value

a character as parameter list of function call

Examples

```
as_param(letters[1:5])
as_param(txt(letters[1:5]))
as_param(a=txt("a"))
as_param(txt(letters[1:5]), fun="c")
```

availablePlots *availablePlots*

Description

Returns a data frame with columns about the available plots in `smvgraph`:

- `module`: the internal name used. If you want to call the Shiny app then you might need this.
- `label`: the label used in the Shiny app
- `help`: the R help topic for the plot
- `packages`: packages which are required to make the plot
- `code`: if code block exists, should always be `TRUE`
- `ui`: if plot specific interactive UI elements exists
- `condition`: the condition when a plot is offered in the Shiny app to the user

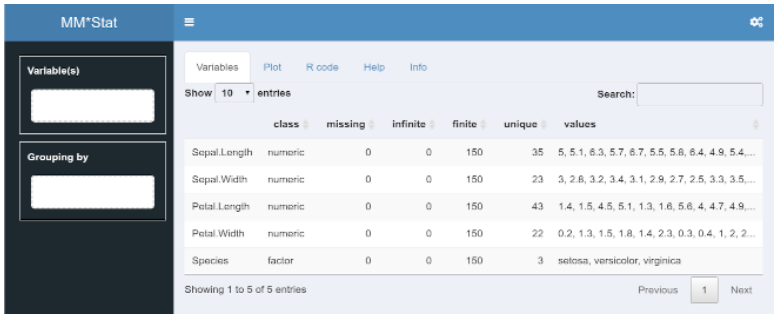
Usage

```
availablePlots()
```

Details

To understand `condition`:

- `nrow(analysis)`: the number of variables in "Analysis" field
- `nrow(group)`: the number of variables in "Grouping by" field
- `xxx$unique`: the number of unique values in a variable, for other elements then `unique` see the "Variable" panel of the Shiny app



`13` was choosen because twelve has the largest number of divisors below 20 and `43` was choosen because forty-two is the answer of the ultimate question ;)

Value

a data frame with information about all available plots

Examples

```
availablePlots()
```

`bagplot2` *bagplot*

Description

A non-ggplot2 bagplot based on mrfDepth::bagplot.

Usage

```
bagplot2(
  x,
  y = NULL,
  colorbag = NULL,
  colorloop = NULL,
  colorchull = NULL,
  databag = TRUE,
  dataloop = TRUE,
  plot.fence = FALSE,
  type = "hdepth",
  sizesubset = 500,
  extra.directions = FALSE,
  options = NULL,
  ...
)
```

Arguments

x, y	the x and y arguments provide the x and y coordinates for the bagplot. Any reasonable way of defining the coordinates is acceptable. See the function `xy.coords` for details. If supplied separately, they must be of the same length.
colorbag	The color of the bag (which contains the 50% observations with largest depth).
colorloop	The color of the loop (which contains the regular observations).
colorchull	When the bagplot is based on halfspace depth, the depth region with maximal depth is plotted. This argument controls its color.
databag	Logical indicating whether data points inside the bag need to be plotted. Defaults to `TRUE`.
dataloop	Logical indicating whether data points inside the fence need to be plotted. Defaults to `TRUE`.
plot.fence	Logical indicating whether the fence should be plotted. Defaults to `FALSE`.
type	Determines the depth function used to construct the bagplot: `"hdepth"` for halfspace depth, `"projdepth"` for projection depth and `"sprojdepth"` for skewness-adjusted projection depth. Defaults to `"hdepth"`.

`sizesubset` When computing the bagplot based on halfspace depth, the size of the subset used to perform the main computations. See Details for more information. Defaults to 500.

`extra.directions`
Logical indicating whether additional directions should be considered in the computation of the fence for the bagplot based on projection depth or skewness-adjusted projection depth. If set to `TRUE` an additional 250 equispaced directions are added to the directions defined by the points in `x` themselves and the center. If `FALSE` only directions determined by the points in `x` are considered. Defaults to `FALSE`.

`options` A list of options to pass to the `projdepth` or `sprojdepth` function. In addition the following option may be specified:

- `max.iter`
  The maximum number of iterations in the bisection algorithm used to compute the depth contour corresponding to the cutoff. See `depthContour` for more information.
  Defaults to 100.

`...` further parameters given to `plot`

Details

The bagplot has been proposed by Rousseeuw et al. (1999) as a generalisation of the boxplot to bivariate data. It is constructed based on halfspace depth and as such is invariant under affine transformations. Similar graphical representations can be obtained by means of other depth functions, as illustrated in Hubert and Van der Veeken (2008) and in Hubert et al. (2015). See mrfDepth::compBagplot for more details.

The deepest point is indicated with a "*" sign, the outlying observations with red points.

Value

Invisibly the result of the call to mrfDepth::compBagplot

References

Rousseeuw P.J., Ruts I., Tukey, J.W. (1999). The bagplot: a bivariate boxplot. The American Statistician, 53, 382–387.

Hubert M., Van der Veeken S. (2008). Outlier detection for skewed data. Journal of Chemometrics, 22, 235–246.

Hubert M., Rousseeuw P.J., Segaert, P. (2015). Rejoinder to 'Multivariate functional outlier detection'. Statistical Methods & Applications, 24, 269–277.

See Also

mrfDepth::compBagplot and mrfDepth::bagplot

Examples

```
bagplot2(iris$Sepal.Length, iris$Sepal.Width)
bagplot2(iris[,1:2])
bagplot2(iris[,3:4], title="Bagplot with Tukey depth", xlab="Petal.Length", ylab="Petal.Width")
#
library("mrfDepth")
data("bloodfat")
result <- compBagplot(bloodfat)
bagplot(result, colorbag = rgb(0.2,0.2,0.2), colorloop = "green")
```

`binData` *binData*

Description

Bins each variable in `data` in `bins` Bins. It can return a data frame (`out="data.frame"`), a table with the counts (`out="table"`), or a table converted to a data frame with an additional variable `Freq`. The values can be either the bin mids (`val="mids"`) or the bin numbers (`val="interval"`). If possible all variables contain an attribute `breaks` with breaks used.

Usage

```
binData(
  data,
  bins,
  out = c("data.frame", "table", "binned"),
  val = c("mid", "interval"),
  pretty = TRUE,
  numeric = TRUE
)
```

Arguments

`data`	object: a data.frame or object that can be converted to a data frame with variables to bin
`bins`	integer: number of bins, will be recycled if necessary
`out`	character: output type, either `"data.frame"`, `"table"`, or `"binned"`
`val`	character: values for outer, eiter `"mids"` (interval centers), or `"interval` (interval number)
`pretty`	logical: should be base::pretty used or minimum and maximum (default: `TRUE`)
`numeric`	logical: return output a `factor` or as `numeric` (default: `TRUE`)

Value

a data frame or table with the results

Examples

```
df <- data.frame(x=runif(25), y=runif(25))
binData(df, 5, 'data.frame')
binData(df, 5, 'table')
binData(df, 5, 'binned')
```

`character_data` *character_data*

Description

Converts a matrix or data frame into a character vector, matrix or data frame. If `na.action` is a character then all `NA`s are replaced by `na.action` (default: `na.action="NA"`). If `na.action` is a function then the function will be applied to the result.

Usage

```
character_data(
  x,
  select = NULL,
  out = c("data.frame", "matrix", "vector"),
  na.action = "NA",
  ...,
  title = NULL
)
```

Arguments

`x`	vector, matrix or data frame
`select`	vector: indicating columns to select (default: `NULL`)
`out`	output as `data.frame` (default), `matrix`, or `vector`
`na.action`	function or character: indicates what should happen when the data contain NAs
`...`	unused
`title`	character: title attribute (default `NULL`)

Value

the desired R object

Examples

```
character_data(iris)
character_data(iris, out="matrix")
character_data(iris, out="vector")
```

checkPackages *checkPackages, installPackages*

Description

Checks if a package is installed without loading it. Returns a logical vector with TRUE or FALSE for each package checked.

Usage

```
checkPackages(
  ...,
  plotmodule = NULL,
  add = c("tools", "devtools", "formatR", "highlight", "shiny", "shinydashboard",
    "shinydashboardPlus", "shinyWidgets", "DT", "sortable", "base64enc"),
  error = FALSE
)

installPackages(
  plotmodule = NULL,
  add = c("tools", "devtools", "formatR", "highlight", "shiny", "shinydashboard",
    "shinydashboardPlus", "shinyWidgets", "DT", "sortable", "base64enc")
)
```

Arguments

...	character: name(s) of package
plotmodule	character: name(s) of plot modules to check for packages
add	character: names of default packages to check (default: c("highlight", "formatR", "shiny", "shinydashboard", "shinydashboardPlus", "DT"))
error	logical: should a error thrown if one or more package are missing? (default: FALSE)

Value

TRUE if successful otherweise an error will be thrown

Examples

```
checkPackages("graphics", add=NULL)          # checks if 'graphics' is installed
if (interactive()) checkPackages("graphics") # checks if 'graphics', 'shiny', ... are installed
if (interactive()) installPackages()         # installs all packages to show ALL plots
```

`color_data` *color_data*

Description

Assigns a color to the data `x` based on the color palette `colpal`.

Usage

```
color_data(x, colpal = grDevices::hcl.colors, select = NULL, ..., title = NULL)
```

Arguments

`x`	vector, matrix, or data frame
`colpal`	color palette (default: grDevices::hcl.colors)
`select`	vector: indicating columns to select (default: 1)
`...`	further parameters to factor_data
`title`	character: title attribute (default `NULL`)

Value

a color vector

Examples

```
color_data(iris)
color_data(iris$Species)
```

`color_hclust` *color_hclust*

Description

Determines colors for `x` based on stats::hclust. `x` is normalized according normalize.

Usage

```
color_hclust(
  x,
  normalize = 1,
  ncol = 2,
  colpal = grDevices::hcl.colors,
  dist = "euclidean",
  na.action = stats::na.pass,
  ...
)
```

Arguments

x	a numeric matrix, data frame or "dist" object.
normalize	integer: normalization method (default: 1) • 0: no rescaling • 1: $(x - min(x))/(max(x) - min(x))$ • 2: $(x - mean(x))/sd(x)$
ncol	integer: maximal number colors
colpal	color palette: a function which generates "ncol" colors with "colpal(ncol)" (default: grDevices::hcl.colors)
dist	the distance measure to be used. This must be one of "euclidean", "maximum", "manhattan", "canberra" or "binary"(default: euclidean)
na.action	a function which indicates what should happen when the data contain NAs (default: na.pass)
...	further parameters given to stats::hclust

Value

a color vector

Examples

```
color_hclust(iris[,-5], ncol=6)
```

---

convertTo *convertTo*

Description

Converts an input object (vector, matrix or data frame) to an output asccording to the format in out. Variable, Row and column names are set, if possible as well as a attribute title.

Usage

```
convertTo(x, coln, rown, title, out = c("data.frame", "matrix", "vector"))
```

Arguments

x	vector, matrix, or data frame: input
coln	character: column names if possible
rown	character: row names if possible
title	character: for title attribute
out	character: either

Value

the desired output object

Examples

```
str(convertTo(pi, "Col1", "Row1", "Title", out='data.frame'))
str(convertTo(pi, "Col1", "Row1", "Title", out='matrix'))
str(convertTo(pi, "Col1", "Row1", "Title", out='vector'))
```

`factor_data` *factor_data*

Description

Creates a single group variable from the data `x`.

Usage

```
factor_data(
  x,
  select = NULL,
  out = c("data.frame", "matrix", "vector"),
  exclude = NULL,
  na.action = stats::na.pass,
  ...,
  title = NULL
)
```

Arguments

`x`	vector, matrix, or data frame
`select`	vector: indicating columns to select (default: `NULL`)
`out`	output as `data.frame` (default), `matrix`, or `vector`
`exclude`	vector: values to be excluded when forming the set of levels (default: `NULL`)
`na.action`	a function which indicates what should happen when the data contain NAs (default: stats::na.pass)
`...`	further parameters to character_data
`title`	character: title attribute (default `NULL`)

Value

a one-column matrix with the merged groups

Examples

```
factor_data(iris$Species, out="vector")
factor_data(iris)
```

formatCommands *formatCommands*

Description

formatCommands

Usage

```
formatCommands(cmds)
```

Arguments

cmds	characater: R code

Value

HTML code for the splot app

Examples

```
formatCommands('print("Hello World!")')
```

getModules *getModules*

Description

Returns a list of available module as a list.

Usage

```
getModules(pattern, path = getShinyOption("smvgraph.path"))
```

Arguments

pattern	character: character string containing a regular expression, currently are used plot_*.R and color_*.R
path	character: containing the path where to search the modules

Value

a list with the modules

Examples

```
library("shiny")
getModules('plot_*.R')   # get plots
getModules('color_*.R')  # get colors
```

---

getval *getval*

Description

Returns `val` if `length(val)>1`. Otherwise it runs through `args=list(...)` until it finds an element with `length(args[[i]])>0` and returns it. If everything fails `NULL` will be returned.

Usage

```
getval(val, ...)
```

Arguments

`val`	current value
`...`	sequence of alternative values

Value

a value

Examples

```
getval(NULL, 0)
getval(1, 0)
```

---

getVariableInfo *getVariableInfo*

Description

Returns a data frame with one row for each variable in `data`:

Usage

```
getVariableInfo(data, n = 47)
```

Arguments

`data`	data frame: input data set
`n`	integer: character length for `values` (default: `47`)

Details

- `class` the base::class of the variable
- `missing` the number of missing values
- `infinite` the number of infinite values
- `unique` the number of unique values
- `valid` the number of unique valid values (see valid)
- `values` the values with the decreasing frequency

Value

a data frame with information about the variables of the input data set

Examples

```
getVariableInfo(iris)
```

`getVariableNames` *getVariableNames*

Description

Extracts variable names from a data frame or matrix (column names).

Usage

```
getVariableNames(x, xvar = NULL, num = TRUE)
```

Arguments

`x`	data frame/matrix: data set to analyse
`xvar`	character: variable names to analyse (default: `character(0)` = all variables)
`num`	logical: should numerical or non-numerical variable use (default: `TRUE`)

Value

character vector with variable names

Examples

```
getVariableNames(iris)
getVariableNames(iris, num=FALSE)
getVariableNames(normalize(iris, 0))
getVariableNames(normalize(iris, 0), num=FALSE)
```

jitter_min *jitter_min*

Description

Add a small amount of noise to a numeric vector. The result is `x + runif(n, -a, a)` where `n <- length(x)` and `a <- abs(factor*amount)` argument. If `amount==0` then `amount` is set to `1e-6` times the smallest non-zero distance between adjacent unique `x` values. In case of no non-zero distances `amount` is set to `1e-6*(1+min(abs(x)))`. Note that `jitter_min` delivers different results then base::jitter.

Usage

```
jitter_min(x, factor = 1, amount = 0)
```

Arguments

`x`	numeric: vector to which jitter should be added
`factor`	numeric: multiplier for `amount` (default: `1`)
`amount`	numeric: amount for jittering (default: `0`)

Value

jittered data

Examples

```
jitter_min(runif(6))
jitter_min(rep(0, 7))
jitter_min(rep(10000, 5))
```

loggit *loggit*

Description

Stores in a temporary file the log messages including messages, warnings and errors.

Usage

```
loggit(log_lvl, log_msg)

read_logs()

set_logfile()
```

Arguments

log_lvl	character: Level of log output. In actual practice, one of "DEBUG", "INFO", "WARN", and "ERROR" are common, but any string may be supplied
log_msg	character: Main log message

Value

Nothing.

Examples

```
if (interactive()) {
  set_logfile()      # create a temporary file for logging
  loggit("DEBUG", "Hello world")
  read_logs()        # get a data frame with the current messages.
}
```

---

normalize *normalize*

Description

Extracts the numeric vectors from a data frame and normalizes each vector. Note: In case that a variable is constant for `method==1` (minmax) the entries will be replaced by `0.5` and for `method==2` (standardization) the entries will be replaced by `0`.

Usage

```
normalize(x, method = 1)
```

Arguments

x	data.frame or matrix
method	integer: normalization method (default: 1) • 0: no rescaling • 1: $(x - min(x))/(max(x) - min(x))$ • 2: $(x - mean(x))/sd(x)$

Value

numeric matrix

See Also

In package normalize or at CRAN

Examples

```
normalize(iris, 2)
```

numeric_data *numeric_data*

Description

Converts a vector, matrix or data frame into a numeric vector, matrix or data frame.

Usage

```
numeric_data(
  x,
  select = NULL,
  out = c("data.frame", "matrix", "vector"),
  na.action = stats::na.pass,
  ...,
  title = NULL
)
```

Arguments

x	vector, matrix or data frame
select	vector: indicating columns to select (default: NULL)
out	output as data.frame (default), matrix, or vector
na.action	a function which indicates what should happen when the data contain NAs (default: stats::na.pass)
...	unused
title	character: title attribute (default NULL)

Value

the desired R object

Examples

```
numeric_data(iris)
numeric_data(iris, out="matrix")
numeric_data(iris, out="vector")
```

order_andrews *order_andrews*

Description

Returns a reording of the columns of `x` to visualize outliers or clusters better. If no colum names are given then `V1`, `V2`, ... will be used.

Usage

```
order_andrews(x, method = 1)
```

Arguments

`x`	data matrix
`method`	numeric: order method (default: `1`) • 1: interquartile range • 2: $max(x - median(x))/IQR(x)$ (outlier) • 3: fit to a Ward cluster solution with euclidean distance

Value

order of column vectors

Examples

```
order_andrews(iris)
```

order_parcoord *order_parcoord*

Description

Returns a reordering of the columns of `x` to visualize highly correlated variable pairs based on a cluster analysis of the correlation matrix. If no colum names are given then `V1`, `V2`, ... will be used.

Usage

```
order_parcoord(x, method = "spearman", ...)
```

Arguments

`x`	data matrix
`method`	numeric: order method (default: `"spearman"`)
`...`	further parameters given to stats::cor

Value

order of column vectors

Examples

```
order_parcoord(iris)
```

`pyramid` *pyramid*

Description

pyramid

Usage

```
pyramid(
  tab,
  gap = 0,
  left = list(col = "red"),
  right = list(col = "blue"),
  ...
)
```

Arguments

`tab`	table: a table with two columns
`gap`	numeric(2): relative size of gap in y- and x-direction (default: `c(0,0)`)
`left`	list: parameters for the left polygons (default: `list(col="red")`)
`right`	list: parameters for the right polygons (default: `list(col="blue")`)
`...`	further parameters to use in graphics::plot.default

Value

a pyramid plot

Examples

```
data("Boston", package="MASS")
tab <- table(data.frame(Boston$rad, Boston$chas))
pyramid(tab, main="Absolute frequencies")
pyramid(tab, gap=c(0.2, 0.2))
rtab <- tab/sum(tab)
pyramid(rtab, gap=c(0.2, 0.2), main="Relative frequencies")
ctab <- proportions(tab, 2)
pyramid(ctab, gap=c(0.2, 0.2), main="Conditional frequencies on columns")
rtab <- proportions(tab, 1)
```

```
pyramid(rtab, gap=c(0.2, 0.2), main="Conditional frequencies on rows")
# zebraing
pyramid(tab, gap=c(0.2, 0.2),
        left=list(list(col="black"), list(col="white")),
        right=list(list(col="blue"), list(col="green")))
```

`resetpar` *resetpar*

Description

Resets the par if necessary.

Usage

```
resetpar(oldpar)
```

Arguments

`oldpar`	graphical parameters

Value

nothing

Examples

```
# no examples
```

`sandrews` *sandrews*

Description

Shiny app for creating an Andrews curve diagram with interactive variable selection.

Usage

```
sandrews(data, xvar = character(0), ...)
```

Arguments

`data`	matrix or data frame
`xvar`	character: names of selected variables for the plot
`...`	unused

Value

nothing

Examples

```
if (interactive()) sandrews(iris)
```

`schernoff` *schernoff*

Description

Shiny app for creating a Chernoff faces plot with interactive variable selection.

Usage

```
schernoff(data, xvar = character(0), ...)
```

Arguments

`data`	matrix or data.frame
`xvar`	character: names of selected variables for the plot
`...`	further parameters given to DescTools::PlotFaces

Value

nothing

Examples

```
if (interactive()) schernoff(normalize(iris))
```

`sdbscan` *sdbscan*

Description

Shiny app which allows to run a cluster analysis with DBSCAN with interactive choice of variables, core distance, and minimal neighbours.

Usage

```
sdbscan(data, xvar = character(0), ...)
```

Arguments

data	matrix or data.frame
xvar	character: names of selected variables for the clustering
...	unused

Value

nothing

Examples

```
if (interactive()) sdbscan(iris)
```

---

sdistance *sdistance*

Description

Shiny app which shows the contribution of each variable to the distance between two observations with interactive variable selection. If $dijk$ is the distance between observations i and j in variable k then the contribution is computed:

Usage

```
sdistance(data, xvar = character(0), ...)
```

Arguments

data	matrix or data.frame
xvar	character: names of selected variables for the plot
...	unused

Details

- Total variance: $var_k/sum(var_k)$ with var_k the variance of the kth variable
- Minimum: $dijk == min_k(dijk)$
- Manhattan: $dijk/sum(dijk)$
- Gower: $dijk$ is rescaled to $[0, 1]$ in each variable and then $dijk/sum(dijk)$
- Euclidean: $dijk^2/sum(dijk^2)$
- Manhattan: $dijk/sum(dijk)$
- Maximum: $dijk == max_k(dijk)$

Value

nothing

Examples

```
if (interactive()) sdistance(iris)
```

`sfactor` *sfactor*

Description

Shiny app for doing a factor analysis with interactive variable selection.

Usage

```
sfactor(data, xvar = character(0), ...)
```

Arguments

`data`	matrix or data frame
`xvar`	character: names of selected variables for the plot
`...`	unused

Value

nothing

Examples

```
if (interactive()) sfactor(iris)
```

`shclust` *shclust*

Description

Shiny app which allows to run a hierarchical cluster analysis with interactive choice of variables, distance, and agglomeration method.

Usage

```
shclust(data, xvar = character(0), ...)
```

Arguments

`data`	matrix or data.frame
`xvar`	character: names of selected variables for the clustering
`...`	unused

Value

nothing

Examples

```
if (interactive()) shclust(iris)
```

---

skmeans *skmeans*

Description

Shiny app which allows to run a k-means cluster analysis with interactive choice of variables.

Usage

```
skmeans(data, xvar = character(0), ...)
```

Arguments

data	matrix or data.frame
xvar	character: names of selected variables for the clustering
...	unused

Value

nothing

Examples

```
if (interactive()) skmeans(iris)
```

---

smclust *smclust*

Description

Shiny app which allows to run a EM clustering with interactive choice of variables.

Usage

```
smclust(data, xvar = character(0), ...)
```

Arguments

data	matrix or data.frame
xvar	character: names of selected variables for the clustering
...	unused

Value

nothing

Examples

```
if (interactive()) smclust(iris)
```

---

smosaic *smosaic*

Description

Shiny app for creating a Mosaic plot with interactive variable selection.

Usage

```
smosaic(data, xvar = character(0), yvar = character(0), ...)
```

Arguments

data	table or data.frame
xvar	character: names of selected variables for x-axis
yvar	character: names of selected variables for y-axis
...	further parameters given to graphics::mosaicplot

Value

nothing

Examples

```
if (interactive()) smosaic(Titanic)
dfTitanic <- toDataframe(Titanic)
if (interactive()) smosaic(dfTitanic)
```

sortbin *sortbin*

Description

Sorts and bins the rows of the data frame `x` according the sorting columns in `sortCol`. `decreasing` and `na.last` are recycled is necessary. If `equibin` is `TRUE` and `nBins==NA` then `nBins` is set to `100`. If `equibin` is `FALSE` and `nBins==NA` then the bins are returned as they come from sorting; only identical values are in one bin. If `nBins` is positive then the bins are merged until `nBins` reached. Note that the numbers of observations per bin may vary.

Usage

```
sortbin(
  x,
  sortCol = 1,
  decreasing = FALSE,
  na.last = TRUE,
  nBins = NA,
  equibin = TRUE
)
```

Arguments

`x`	data frame
`sortCol`	numeric/character: names or indices of variable used for sorting (default: `1`)
`decreasing`	logical: should the sort order be increasing or decreasing (default: `FALSE`)
`na.last`	logical: for controlling the treatment of NAs (default: `TRUE`)
`nBins`	integer: maximal number of bins (default: `NA`).
`equibin`	logical: should the number of observations equal per bin (default: `TRUE`)

Value

(non-sequential) bin numbers as integer

Examples

```
data("Boston", package="MASS")
tableplot(Boston, bin=sortbin(Boston))
```

spairs *spairs*

Description

Shiny app for creating a scatterplot matrix with interactive variable selection.

Usage

```
spairs(data, xvar = character(0), ...)
```

Arguments

data	matrix or data.frame
xvar	character: names of selected variables for the plot
...	further parameters given to graphics::pairs

Value

nothing

Examples

```
if (interactive()) spairs(iris)
```

sparcoord *sandrews*

Description

Shiny app for creating a Parallel Coordinate plot with interactive variable selection.

Usage

```
sparcoord(data, xvar = character(0), ...)
```

Arguments

data	matrix or data.frame
xvar	character: names of selected variables for the plot
...	further parameters given to MASS::parcoord

Value

nothing

Examples

```
if (interactive()) sparcoord(iris)
```

`splot` *splot*

Description

Shiny app for choosing a specific plot.

Usage

```
splot(data, xvar = character(0), path = NULL)
```

Arguments

`data`	data.frame: input data
`xvar`	character: selected variables (default: `character(0)`)
`path`	character: path where to read the plot modules (default: `NULL`)

Value

nothing

Examples

```
if (interactive()) splot(iris)
```

`sradar` *sradar*

Description

Shiny app for creating radar charts with interactive variable selection.

Usage

```
sradar(data, xvar = character(0), ...)
```

Arguments

`data`	matrix or data.frame
`xvar`	character: names of selected variables for the plot
`...`	unused

Value

nothing

Examples

```
if (interactive()) sradar(iris)
```

`tableplot` *tableplot*

Description

A tableplot is a visualisation of multivariate data sets. Each column represents a variable and each row bin is an aggregate of a certain number of records. For numeric variables, a value box is plotted with minimum, mean (black line) and maximum value. If any missing values in a bin of a numeric variable appear the box left from the value box is plotted in gray. For categorical variables, a stacked bar chart is depicted of the proportions of categories. Missing values are taken into account.

Usage

```
tableplot(
  x,
  select = NULL,
  subset = NULL,
  bin = NULL,
  yj = NA,
  IQR_bias = 5,
  colpal = grDevices::rainbow,
  color.NA_num = "gray75",
  color.NA = "grey75",
  color.num = "lightblue",
  color.box = "deepskyblue",
  color.line = "black",
  box.lower = NULL,
  box.upper = NULL,
  box.line = NULL,
  cex.main = 1,
  cex.legend = 1,
  width = 1,
  height = 0.15
)
```

Arguments

`x`	data frame
`select`	numeric/character: variable to show in the plot (default: `NULL`)

subset	numeric: index of observations to show
bin	integer: bin numbers to which a observations belongs (default: NULL = all)
yj	numeric: Yeo Johnson coefficient (default: NA). If NA then it will be set to 0 (=log) or 1 (=identity)
IQR_bias	numeric: parameter that determines when a logarithmic scale is used when yj is set to NA. The argument IQR_bias is multiplied by the interquartile range as a test.
colpal	color palette to draw (default: rainbow)
color.NA_num	color for missing of infinity values for numeric variables (default: gray75)
color.NA	color for missing values for categorical variables (default: grey75)
color.num	color for lower box for numeric variables (default: lightblue)
color.box	color for upper box for numeric variables (default: deepskyblue)
color.line	color for line in upper box for numeric variables (default: black)
box.lower	function: determine lower border in upper box for numeric variables (default: NULL). If NULL then min(.,na.rm=TRUE) is used.
box.upper	function: determine upper border in upper box for numeric variables (default: NULL). If NULL then max(.,na.rm=TRUE) is used.
box.line	function: determine line position in upper box for numeric variables (default: NULL). If NULL then mean(.,na.rm=TRUE) is used.
cex.main	number: magnification to be used for the titles (default: 1)
cex.legend	number: magnification to be used for the legends (default: 1)
width	number: width of percentage axis (default: 1). If 1 then the width is as wide as a plot.
height	number: percentage of the height of the legends (default: 0.15)

Details

The idea and some code of the tableplot is taken from tableplot package by Martijn Tennekes and Edwin de Jonge. It differs from their package by

- multicolumn sorting is possible, and
- no support for 'ff' (out of memory vectors).

Value

nothing

References

Tennekes, M., Jonge, E. de, Daas, P.J.H. (2013), Visualizing and Inspecting Large Datasets with Tableplots, Journal of Data Science 11 (1), 43-58.

Examples

```
data("Boston", package="MASS")
tableplot(Boston, bin=sortbin(Boston))
```

template *template*

Description

Each line of a code template consists of condition based on the unnamed parameters and R code in which replacements with named parameters done.

Usage

```
template(text, ...)
```

Arguments

text	a code template
...	further parameters

Value

a character vector

Examples

```
template("
1:   'Hello {{letter}}'
!1:  'Good-bye {{letter}}'
         ",
         letter=sample(LETTERS, 1),
         runif(1)<0.5 #1 = first unnamed parameter
         )
```

testdata *Test data*

Description

A data frame containing various variable types and special values.

Usage

```
testdata
```

Format

A data frame with n=25 rows and 8 variables:

xu runif(n) with a NA, NaN, Inf, -Inf

xn rnorm(n, 0, 2) with a NA, NaN

x0 rep(0, n)

xi as.integer(rnorm(n, 0, 2) with a NA, NaN

x2 sample(c(0,1), size=n, replace=TRUE)

gf factor(as.integer(rnorm(n, 0, 2)) with a NA

go ordered(as.integer(rnorm(n, 0, 2)) with a new level 10

gn ordered(as.integer(rnorm(n, 0, 2)) with a NA

gc as.character(as.integer(rnorm(n, 0, 2)) with a NA and ""

gl sample(c(T,F), size=n, replace=TRUE)) with a NA

g0 rep("constant, n)

g2 sample(c(T,F), size=n, replace=TRUE)

---

toChoice *toChoice*

Description

The elements in ... will coerced into one text vector. The entries will either the text (method==NA) or integer number starting at method. The first letter of the list element names will be capitalized.

Usage

```
toChoice(method = NA, ...)
```

Arguments

method	integer: which method is used for creating the list elements
...	character: choice values

Value

a list

Examples

```
txt <- c("the", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog")
toChoice(NA, txt)
toChoice(0, txt) # integer sequence starts at zero
```

`toDataframe` *toDataframe*

Description

Converts a table to a full data frame.

Usage

```
toDataframe(obj, name = NULL, ...)
```

Arguments

`obj`	R object (`table` or `ts`) to convert to a data frame
`name`	character: vector of variable name(s), only use for a `ts` object
`...`	further parameters given to base::as.data.frame.table

Value

a data frame

Examples

```
toDataframe(Titanic)
toDataframe(austres)
```

`toLayout` *toLayout*

Description

Given a (minimal) length `len` and the number unused entries a layout is generated. If `sel<0` then less rows and more columns are used and if `sel>0` then more rows and less columns are used.

Usage

```
toLayout(len, sel = 0, unused = 0)
```

Arguments

`len`	integer: minimal size of layout
`sel`	integer: select less or more rows (default: `0`)
`unused`	integer: number of unused entries (default: `0`)

Value

a matrix

Examples

```
toLayout(13)
```

---

toRDS *toRDS*

Description

Saves one or more data sets in RDS format to a temporary directory (`tmpdir()`). Data sets must have the class `ts` or something that can be converted to a data frame, e.g. `matrix`, `table`, etc.

Usage

```
toRDS(...)
```

Arguments

`...`	data sets to save

Value

returns the name of the created files

Examples

```
toRDS(Titanic) # saves to tempdir/Titanic.rds
```

---

trend_season *Trend and aeasonality estimation of a univariate time series*

Description

Estimate a trend and seasonaliyt for a time series. Available functions:

- `trend_season` to generate an estimate
- `print` to print the estimate
- `summary` to summarize the etsimate result
- `plot` to plot the time series, its estimation and the residuals
- `coef` to extract the coefficients if a seasonality estimation was done
- `residuals` to extract the residuals of the model
- `fitted` to the fitted values

Usage

```
trend_season(t, ...)

## Default S3 method:
trend_season(
  t,
  trend = c("constant", "linear", "exponential"),
  season = c("none", "additive", "multiplicative"),
  ...
)

## S3 method for class 'trend_season'
print(x, ...)

## S3 method for class 'trend_season'
summary(object, ...)

## S3 method for class 'trend_season'
plot(x, y, which = 1, ...)
```

Arguments

t	ts: time series object
...	unused
trend	character: trend method, either none (default), linear or exponential
season	character: seasonality method, either none (default), additive or multiplicative
x, object	trend_season: estimated time series
y	unused
which	integer: what to plot, 1 time series and estimation (default) or 2 residuals

Value

trend_season returns a trend_season object with

- call the function call
- ts the input time series
- trend the trend estimation (ts object)
- trend.residuals the residuals of the trend estimation (ts object)
- season the trend and season estimation (ts object)
- season.residuals the residuals of the trend and season estimation (ts object)
- coefficients the coefficients used in the seasonality estimation
- residuals the residuals of the model
- fitted.values the fitted values of the model

Examples

```
tts <- trend_season(austres, "linear")
print(tts)
summary(tts)
plot(tts)
plot(tts, which=2)
residuals(tts)
fitted(tts)
coef(tts)  # if NULL then no seasonality was estimated
```

`UIdatanormalization` *General UI elements*

Description

Some general UI elements for common use where last selected value is stored for reuse:

- `UIplottype` plot type, defines `smvgraph_type`
- `UIpointsymbol` plot symbol for point, defines `smvgraph_pch`
- `UIpointsize` point size, defines `smvgraph_cex`
- `UIlinetype` line type, defines `smvgraph_lty`
- `UIlinewidth` line width, defines `smvgraph_lwd`
- `UItextsize` text size, defines `smvgraph_tex`
- `UIlegend` legend position, defines `smvgraph_legend`
- `UIlegendsize` legend size, defines `smvgraph_lex`
- `UIdatanormalization` should data be rescaled, defines `smvgraph_normalize` (no, minMax, mtandardization)
- `UIdistance` distance to use, defines `smvgraph_distance`
- `UIobservations` range of observations, defines `smvgraph_obs`
- `UImergegroups` should a set of grouping variables merged into one group variable, defines `smvgraph_single`

From the top menu are the following input elements are defined

- `input$smvgraph_pch` point symbol,
- `input$smvgraph_cex` point size,
- `input$smvgraph_lty` line type,
- `input$smvgraph_lwd` line width,
- `input$smvgraph_tex` text size, and
- `inpus$smvgraph_legend` legend position.

Usage

```
UIdatanormalization(
  sel = getShinyOption("smvgraph.current")$smvgraph_normalize
)

UIdistance(sel = getShinyOption("smvgraph.current")$smvgraph_distance)

UIobservations(n, sel = getShinyOption("smvgraph.current")$smvgraph_obs)

UImergegroups(n, sel = getShinyOption("smvgraph.current")$smvgraph_single)

UIpointsize(n, sel = getShinyOption("smvgraph.current")$smvgraph_cex)

UIpointsymbol(n, sel = getShinyOption("smvgraph.current")$smvgraph_pch)

UIlinewidth(n, sel = getShinyOption("smvgraph.current")$smvgraph_lwd)

UIlinetype(n, sel = getShinyOption("smvgraph.current")$smvgraph_lty)

UItextsize(n, sel = getShinyOption("smvgraph.current")$smvgraph_tex)

UIlegend(n, sel = getShinyOption("smvgraph.current")$smvgraph_legend)

UIlegendsize(n, sel = getShinyOption("smvgraph.current")$mvgraph_lex)

UIplottype(n, sel = getShinyOption("smvgraph.current")$smvgraph_type)
```

Arguments

sel	selected element
n	integer: number of observations

Value

an UI element for shiny

Examples

```
# none
```

`valid` *valid, invalid*

Description

Computes the number a logical matrix or vector if all values are valid in `x` or each colum or row of `x`. Valid values for numeric variables are `is.finite(v)` and for other types `!is.na`

Computes the number a logical matrix or vector if any values are valid in `x` or each colum or row of `x`.

Usage

```
valid(x, margin = 1:2, n = FALSE)

invalid(x, margin = 1:2, n = FALSE)
```

Arguments

`x`	object: anything taht can be coerced to a data frame checked for valid/invalid values
`margin`	integer: a vector giving the subscripts for which valid/invalid values looked for, e.g. `margin==1` indicates rows, `margin==2` indicates columns, otherwise indicates rows and columns.
`n`	logical: should just the number of valid/invalid values returned or a logical matrix/vector

Value

a logical data frame, a logical vector or an integer

Examples

```
data("testdata")
valid(testdata)             # matrix with logical entries if x has valid entry
valid(testdata, n=TRUE)     # number of valid entries in x
valid(testdata, 1)        # vector with logical entries if each row if x has valid entries
valid(testdata, 1, n=TRUE)  # number of rows with valid entries in x
valid(testdata$xu)
```

`with_progress` *Progress*

Description

Reports progress to the user during long-running operations.

Usage

```
with_progress(...)

set_progress(...)

inc_progress(...)
```

Arguments

...	see [shiny::withProgress]

Value

see [shiny::withProgress]

Examples

```
## Only run examples in interactive R sessions
if (interactive()) {
   options(device.ask.default = FALSE)
   ui <- fluidPage(plotOutput("plot"))
   #
   server <- function(input, output) {
     output$plot <- renderPlot({
       with_progress(message = 'Calculation in progress',
                     detail = 'This may take a while...', value = 0, {
                     for (i in 1:15) {
                       inc_progress(1/15)
                       Sys.sleep(0.25)
                     }
                   })
       plot(cars)
   })
 }
 #
 shinyApp(ui, server)
}
```

---

yeo.johnson *yeo.johnson*

Description

Computes the Yeo-Johnson transformation, which is a normalizing transformation. The code and documentation is taken from the VGAM package (see function yeo.johnson) with some slight modifications, e.g. NA's are kept and do not produce an error.

Usage

```
yeo.johnson(
  y,
  lambda,
  derivative = 0,
  epsilon = sqrt(.Machine$double.eps),
  inverse = FALSE
)
```

Arguments

y	numeric: a vector or matrix.
lambda	numeric: It is recycled to the same length as y if necessary.
derivative	non-negative integer: the default is the ordinary function evaluation, otherwise the derivative with respect to lambda (default: 0)
epsilon	numeric and positive value: the tolerance given to values of lambda when comparing it to 0 or 2.
inverse	logical: return the inverse transformation? (default: FALSE)

Details

The Yeo-Johnson transformation can be thought of as an extension of the Box-Cox transformation. It handles both positive and negative values, whereas the Box-Cox transformation only handles positive values. Both can be used to transform the data so as to improve normality.

Value

The Yeo-Johnson transformation or its inverse, or its derivatives with respect to lambda, of y.

Note

If inverse = TRUE then the argument derivative = 0 is required.

References

Yeo, I.-K. and Johnson, R. A. (2000). A new family of power transformations to improve normality or symmetry. *Biometrika*, **87**, 954–959.

See Also

VGAM::yeo.johnson, boxcox.

Examples

```
y <- seq(-4, 4, len = (nn <- 200))
ltry <- c(0, 0.5, 1, 1.5, 2)  # Try these values of lambda
lltry <- length(ltry)
psi <- matrix(as.numeric(NA), nn, lltry)
for (ii in 1:lltry)
  psi[, ii] <- yeo.johnson(y, lambda = ltry[ii])
matplot(y, psi, type = "l", ylim = c(-4, 4), lwd = 2, lty = 1:lltry,
        ylab = "Yeo-Johnson transformation", col = 1:lltry, las = 1,
        main = "Yeo-Johnson transformation with some values of lambda")
abline(v = 0, h = 0)
legend(x = 1, y = -0.5, lty = 1:lltry, legend = as.character(ltry),
       lwd = 2, col = 1:lltry)
```

zzz *zzz*

Description

Runs all `s...` functions for test purposes if interactively called.

Usage

```
zzz()
```

Value

nothing

Examples

```
zzz()
```

Index