Package ‘ufs’

August 22, 2019

Type Package

Title Quantitative Analysis Made Accessible

Version 0.3.1

Date 2019-8-22

Maintainer Gjalt-Jorn Peters <gjalt-jorn@userfriendlyscience.com>

License GPL (>= 3)

Description This is a new version of the 'userfriendlyscience' package, which has grown a bit unwieldy. Therefore, distinct functionalities are being 'consciously uncoupled' into different packages. This package contains the general-purpose tools and utilities (see the 'behaviorchange' package, the 'rosetta' package, and the soon-to-be-released 'scd' package for other functionality), and is the most direct 'successor' of the original 'userfriendlyscience' package. For example, this package contains a number of basic functions to create higher level plots, such as diamond plots, to easily plot sampling distributions, to generate confidence intervals, to plan study sample sizes for confidence intervals, and to do some basic operations such as (dis)attenuate effect size estimates.

URL https://r-packages.gitlab.io/ufs

BugReports https://gitlab.com/r-packages/ufs/issues

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Depends R (>= 3.0.0)

Suggests bootES (>= 1.2), lavaan (>= 0.6), MBESS (>= 4.5.1), psych (>= 1.8), rio (>= 0.5), rmarkdown

Imports digest (>= 0.6.19), diptest (>= 0.75.7), dplyr (>= 0.7.6), GGally (>= 1.4.0), ggplot2 (>= 2.2.1), ggrepel (>= 0.8), ggridges (>= 0.5.0), grDevices (>= 3.0.0), gridExtra (>= 2.3), gtable (>= 0.2.0), knitr (>= 1.22), pander (>= 0.6.3), plyr (>= 1.8.4), scales (>= 1.0.0), SuppDists (>= 1.1.9), viridis (>= 0.5.1)

VignetteBuilder knitr

NeedsCompilation no

Author Gjalt-Jorn Peters [aut, cre] (<https://orcid.org/0000-0002-0336-9589>)

Repository CRAN

Date/Publication 2019-08-22 17:30:02 UTC

R topics documented:

`areColors` *Check whether elements of a vector are valid colors*

Description

This function by Josh O'Brien checks whether elements of a vector are valid colors. It has been copied from a Stack Exchange answer (see `http://stackoverflow.com/questions/13289009/check-if-character-string-is-a-valid-color-representation`).

Usage

```
areColors(x)
```

Arguments

`x`	The vector.

Value

A logical vector.

Author(s)

Josh O'Brien

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

Examples

```
ufs::areColors(c(NA, "black", "blackk", "1", "#00", "#000000"));
```

`associationMatrix` *associationMatrix*

Description

associationMatrix produces a matrix with confidence intervals for effect sizes, point estimates for those effect sizes, and the p-values for the test of the hypothesis that the effect size is zero, corrected for multiple testing.

Usage

```
associationMatrix(dat = NULL, x = NULL, y = NULL,
  conf.level = 0.95, correction = "fdr", bootstrapV = FALSE,
  info = c("full", "ci", "es"), includeSampleSize = "depends",
  bootstrapV.samples = 5000, digits = 2, pValueDigits = digits + 1,
  colNames = FALSE, type = c("R", "html", "latex"), file = "",
  statistic = associationMatrixStatDefaults,
  effectSize = associationMatrixESDefaults, var.equal = TRUE)

## S3 method for class 'associationMatrix'
print(x, type = x$input$type,
  info = x$input$info, file = x$input$file, ...)

## S3 method for class 'associationMatrix'
pander(x, info = x$input$info,
  file = x$input$file, ...)
```

Arguments

`dat`	A dataframe with the variables of interest. All variables in this dataframe will be used if both x and y are NULL. If dat is NULL, the user will be presented with a dialog to select a datafile.
`x`	If not NULL, this should be a character vector with the names of the variables to include in the rows of the association table. If x is NULL, all variables in the dataframe will be used.
`y`	If not NULL, this should be a character vector with the names of the variables to include in the columns of the association table. If y is NULL, the variables in x will be used for the columns as well (which produces a symmetric matrix, similar to most correlation matrices).
`conf.level`	Level of confidence of the confidence intervals.
`correction`	Correction for multiple testing: an element out of the vector c("holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none"). NOTE: the p-values are corrected for multiple testing; The confidence intervals are not!
`bootstrapV`	Whether to use bootstrapping to compue the confidence interval for Cramer's V or whether to use the Fisher's Z conversion.
`info`	Information to print: either both the confidence interval and the point estimate for the effect size (and the p-value, corrected for multiple testing), or only the confidence intervals, or only the point estimate (and the corrected p-value). Must be on element of the vector c("full", "ci", "es").
`includeSampleSize`	Whether to include the sample size when the effect size point estimate and p-value are shown. If this is "depends", it will depend on whether all associations have the same sample size (and the sample size will only be printed when they don't). If "always", the sample size will always be added. If anything else, it will never be printed.
`bootstrapV.samples`	If using boostrapping for Cramer's V, the number of samples to generate.
`digits`	Number of digits to round to when printing the results.
`pValueDigits`	How many digits to use for formatting the p values.
`colNames`	If true, the column heading will use the variables names instead of numbers.
`type`	Type of output to generate: must be an element of the vector c("R", "html", "latex").
`file`	If a file is specified, the output will be written to that file instead of shown on the screen.
`statistic`	This is the complicated bit; this is where associationMatrix allows customization of the used statistics to perform null hypothesis significance testing. For everyday use, leaving this at the default value, associationMatrixStatDefaults, works fine. In case you want to customize, read the 'Notes' section below.
`effectSize`	Like the 'statistics' argument, 'effectSize also allows customization, in this case of the used effect sizes. Again, the default value, associationMatrixESDefaults, works for everyday use. Again, see the 'Notes' section below if you want to customize.

`var.equal`	Whether to test for equal variances ('test'), assume equality ('yes'), or assume unequality ('no'). See `userfriendlyscience::meanDiff()` for more information.
`...`	Addition arguments are passed on to the `print()` amd `pander::pander()` functions.

Value

An object with the input and several output variables, one of which is a dataframe with the association matrix in it. When this object is printed, the association matrix is printed to the screen. If the 'file' parameter is specified, a file with this matrix will also be written to disk.

Note

The 'statistic' and 'effectSize' parameter make it possible to use different functions to conduct null hypothesis significance testing and compute effect sizes. In both cases, the parameter needs to be a list containing four lists, named 'dichotomous', 'nominal', 'ordinal', and 'interval'. Each of these lists has to contain four elements, character vectors of length one (i.e. just one string value), again named 'dichotomous', 'nominal', 'ordinal', and 'interval'.

The combination of each of these names (e.g. 'dichotomous' and 'nominal', or 'ordinal' and 'interval', etc) determine which test should be done when computing the p-value to test the association between two variables of those types, or which effect sizes to compute. When called, associationMatrix determines the measurement levels of the relevant variables. It then uses these two levels (their string representation, e.g. 'dichotomous' etc) to find a string in the 'statistic' and 'effectSize' objects. Two functions with these names are then called from two lists, 'computeStatistic' and computeEffectSize. These lists list contain functions that have the same names as the strings in the 'statistic' list.

For example, when the default settings are used, the string (function name) found for two dichotomous variables when searching in associationMatrixStatDefaults is 'chisq', and the string found in associationMatrixESDefaults is 'v'. associationMatrix then calls `computeStatistic[['chisq']]` and `computeEffectSize[['v']]`, providing the two variables as arguments, as well as passing the 'conf.level' argument. These two functions then each return an object that associationMatrix extracts the information from. Inspect the source code of these functions (by typing their names without parentheses in the R prompt) to learn how this object should look, if you want to write your own functions.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

Examples

```
### Generate a simple association matrix using all three variables in the
### Orange tree dataframe
associationMatrix(Orange);
```

```
### Or four variables from infert:
associationMatrix(infert, c("education", "parity",
                            "induced", "case"), colNames=TRUE);

### Use variable names in the columns and generate html
associationMatrix(Orange, colNames=TRUE, type='html');
```

---

associationsDiamondPlot *A diamondplot with confidence intervals for associations*

Description

This function produces is a diamondplot that plots the confidence intervals for associations between a number of covariates and a criterion. It currently only supports the Pearson's r effect size metric; other effect sizes are converted to Pearson's r.

Usage

```
associationsDiamondPlot(dat, covariates, criteria, labels = NULL,
  criteriaLabels = NULL, decreasing = NULL, sortBy = NULL,
  conf.level = 0.95,
  criteriaColors = viridis::viridis(length(criteria)),
  criterionColor = "black", returnLayerOnly = FALSE, esMetric = "r",
  multiAlpha = 0.33, singleAlpha = 1, showLegend = TRUE,
  xlab = "Effect size estimates", ylab = "",
  theme = ggplot2::theme_bw(), lineSize = 1, outputFile = NULL,
  outputWidth = 10, outputHeight = 10, ggsaveParams = list(units =
  "cm", dpi = 300, type = "cairo"), ...)

associationsToDiamondPlotDf(dat, covariates, criterion, labels = NULL,
  decreasing = NULL, conf.level = 0.95, esMetric = "r")
```

Arguments

`dat`	The dataframe containing the relevant variables.
`covariates`	The covariates: the list of variables to associate to the criterion or criteria, usually the predictors.
`criteria, criterion`	The criteria, usually the dependent variables; one criterion (one dependent variable) can also be specified of course. The helper function `associationsToDiamondPlotDf` always accepts only one criterion.
`labels`	The labels for the covariates, for example the questions that were used (as a character vector).
`criteriaLabels`	The labels for the criteria (in the legend).

`decreasing`	Whether to sort the covariates by the point estimate of the effect size of their association with the criterion. Use `NULL` to not sort at all, `TRUE` to sort in descending order, and `FALSE` to sort in ascending order.
`sortBy`	When specifying multiple criteria, this can be used to indicate by which criterion the items should be sorted (if they should be sorted).
`conf.level`	The confidence of the confidence intervals.
`criteriaColors, criterionColor`	The colors to use for the different associations can be specified in `criteriaColors`. This should be a vector of valid colors with at least as many elements as criteria are specified in `criteria`. If only one criterion is specified, the color in `criterionColor` is used.
`returnLayerOnly`	Whether to return the entire object that is generated, or just the resulting ggplot2 layer.
`esMetric`	The effect size metric to plot - currently, only 'r' is supported, and other values will return an error.
`multiAlpha, singleAlpha`	The transparency (alpha channel) value of the diamonds for each association can be specified in `multiAlpha`, and if only one criterion is specified, the alpha level of the diamonds can be specified in `singleAlpha`.
`showLegend`	Whether to show the legend.
`xlab, ylab`	The label to use for the x and y axes (for `duoComparisonDiamondPlot`, must be vectors of two elements). Use `NULL` to not use a label.
`theme`	The `ggplot()` theme to use.
`lineSize`	The thickness of the lines (the diamonds' strokes).
`outputFile`	A file to which to save the plot.
`outputWidth, outputHeight`	Width and height of saved plot (specified in centimeters by default, see `ggsaveParams`).
`ggsaveParams`	Parameters to pass to ggsave when saving the plot.
`...`	Any additional arguments are passed to `diamondPlot()` and eventually to `ggDiamondLayer()`.

Details

associationsToDiamondPlotDf is a helper function that produces the required dataframe.

This function can be used to quickly plot multiple confidence intervals.

Value

A plot.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

See Also

diamondPlot(), ggDiamondLayer(), behaviorchange::CIBER()

Examples

```
### Simple diamond plot with correlations
### and their confidence intervals

associationsDiamondPlot(mtcars,
                        covariates=c('cyl', 'hp', 'drat', 'wt',
                                     'am', 'gear', 'vs', 'carb', 'qsec'),
                        criteria='mpg');

### Same diamond plot, but now with two criteria,
### and colouring the diamonds based on the
### correlation point estimates: a gradient
### is created where red is used for -1,
### green for 1 and blue for 0.

associationsDiamondPlot(mtcars,
                        covariates=c('cyl', 'hp', 'drat', 'wt',
                                     'am', 'gear', 'vs', 'carb', 'qsec'),
                        criteria=c('mpg', 'disp'),
                        generateColors=c("red", "blue", "green"),
                        fullColorRange=c(-1, 1));
```

---

attenuate.d *Attenuate a Cohen's d estimate for unreliability in the continuous variable*

Description

Attenuate a Cohen's d estimate for unreliability in the continuous variable

Usage

```
attenuate.d(d, reliability)
```

Arguments

d	The (disattenuated) value of Cohen's d
reliability	The reliability of the measurements of the continuous variable

Value

The attenuated value of Cohen's d

Examples

```
attenuate.d(.5, .8);
```

attenuate.r *Attenuate a Pearson's r estimate for unreliability in the measurements*

Description

Attenuate a Pearson's r estimate for unreliability in the measurements

Usage

```
attenuate.r(r, reliability1, reliability2)
```

Arguments

`r`	The (disattenuated) value of Pearson's r
`reliability1, reliability2`	The reliabilities of the two variables

Value

The attenuated value of Pearson's r

Examples

```
attenuate.r(.5, .8, .9);
```

biAxisDiamondPlot *Diamondplot with two Y axes*

Description

This is basically a `meansDiamondPlot()`, but extended to allow specifying subquestions and anchors at the left and right side. This is convenient for psychological questionnaires when the anchors or dimensions were different from item to item. This function is used to function the left panel of the `behaviorchange::CIBER()` plot.

Usage

```
biAxisDiamondPlot(dat, items = NULL, leftAnchors = NULL,
  rightAnchors = NULL, subQuestions = NULL, decreasing = NULL,
  conf.level = 0.95, showData = TRUE, dataAlpha = 0.1,
  dataColor = "#444444", diamondColors = NULL, jitterWidth = 0.45,
  jitterHeight = 0.45, xbreaks = NULL, xLabels = NA,
  xAxisLab = paste0("Scores and ", round(100 * conf.level, 2), "% CIs"),
  drawPlot = TRUE, returnPlotOnly = TRUE, baseSize = 1,
  dotSize = baseSize, baseFontSize = 10 * baseSize,
  theme = ggplot2::theme_bw(base_size = baseFontSize),
  outputFile = NULL, outputWidth = 10, outputHeight = 10,
  ggsaveParams = list(units = "cm", dpi = 300, type = "cairo"), ...)
```

Arguments

`dat`	The dataframe containing the variables.
`items`	The variables to include.
`leftAnchors`	The anchors to display on the left side of the left hand panel. If the items were measured with one variable each, this can be used to show the anchors that were used for the respective scales. Must have the same length as `items`.
`rightAnchors`	The anchors to display on the left side of the left hand panel. If the items were measured with one variable each, this can be used to show the anchors that were used for the respective scales. Must have the same length as `items`.
`subQuestions`	The subquestions used to measure each item. This can also be used to provide pretty names for the variables if the items were not measured by one question each. Must have the same length as `items`.
`decreasing`	Whether to sort the items. Specify `NULL` to not sort at all, `TRUE` to sort in descending order, and `FALSE` to sort in ascending order.
`conf.level`	The confidence levels for the confidence intervals.
`showData`	Whether to show the individual datapoints.
`dataAlpha`	The alpha level (transparency) of the individual datapoints. Value between 0 and 1, where 0 signifies complete transparency (i.e. invisibility) and 1 signifies complete 'opaqueness'.
`dataColor`	The color to use for the individual datapoints.
`diamondColors`	The colours to use for the diamonds. If NULL, the `generateColors` argument can be used which will then be passed to `diamondPlot()`.
`jitterWidth`	How much to jitter the individual datapoints horizontally.
`jitterHeight`	How much to jitter the individual datapoints vertically.
`xbreaks`	Which breaks to use on the X axis (can be useful to override `ggplot()`'s defaults).
`xLabels`	Which labels to use for those breaks (can be useful to override `ggplot()`'s defaults; especially useful in combination with `xBreaks` of course).
`xAxisLab`	Axis label for the X axis.

`drawPlot`	Whether to draw the plot, or only return it.
`returnPlotOnly`	Whether to return the entire object that is generated (including all intermediate objects) or only the plot.
`baseSize`	This can be used to efficiently change the size of most plot elements.
`dotSize`	This is the size of the points used to show the individual data points in the left hand plot.
`baseFontSize`	This can be used to set the font size separately from the `baseSize`.
`theme`	This is the theme that is used for the plots.
`outputFile`	A file to which to save the plot.
`outputWidth, outputHeight`	Width and height of saved plot (specified in centimeters by default, see `ggsaveParams`).
`ggsaveParams`	Parameters to pass to ggsave when saving the plot.
`...`	These arguments are passed on to diamondPlot].

Details

This is a diamondplot that can be used for items/questions where the anchors of the response scales could be different for every item. For the rest, it is very similar to `meansDiamondPlot()`.

Value

Either just a plot (a `gtable::gtable()` object) or an object with all produced objects and that plot.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

See Also

`behaviorchange::CIBER()`, `associationsDiamondPlot()`

Examples

```
biAxisDiamondPlot(dat=mtcars,
                  items=c('cyl', 'wt'),
                  subQuestions=c('cylinders', 'weight'),
                  leftAnchors=c('few', 'light'),
                  rightAnchors=c('many', 'heavy'),
                  xbreaks=0:8);
```

`cat0` *Concatenate to screen without spaces*

Description

The cat0 function is to cat what paste0 is to paste; it simply makes concatenating many strings without a separator easier.

Usage

```
cat0(..., sep = "")
```

Arguments

`...`	The character vector(s) to print; passed to cat.
`sep`	The separator to pass to cat, of course, `""` by default.

Value

Nothing (invisible `NULL`, like cat).

Examples

```
cat0("The first variable is '", names(mtcars)[1], "'.");
```

`checkPkgs` *Check for presence of a package*

Description

This function efficiently checks for the presence of a package without installing it (unlike `library()` or `require()`. This is useful to force yourself to use the package::function syntax for addressing functions; you can make sure required packages are installed, but their namespace won't attach to the search path.

Usage

```
checkPkgs(..., install = FALSE, load = FALSE,
  repos = "https://cran.rstudio.com")
```

Arguments

`...`	A vector with packages. If this is a names vector, the names are the package names, and the values are the minimum required package versions.
`install`	Whether to install missing packages from `repos`.
`load`	Whether to load packages (which is exactly *not* the point of this package, but hey, YMMV).
`repos`	The repository to use if installing packages; default is the RStudio repository.

Value

Invisibly, a vector of the available packages.

Examples

```
checkPkgs('justifier');
checkPkgs(justifier = "99");
```

CIM *Conceptual Independence Matrix*

Description

Conceptual Independence Matrix

Usage

```
CIM(data, scales, outputFile = NULL, outputWidth = 100,
  outputHeight = 100, outputUnits = "cm", faMethod = "minres",
  n.iter = 100, skipRegex = NULL, headingLevel = 2,
  printAbbreviations = TRUE, drawPlot = TRUE, returnPlotOnly = TRUE)

## S3 method for class 'CIM'
knit_print(x, headingLevel = x$input$headingLevel,
  quiet = TRUE, echoPartial = FALSE, partialFile = NULL, ...)
```

Arguments

`data`	The dataframe containing the variables.
`scales`	The scales: a names list of character vectors, where the character vectors specify the variable names, and the names of each character vector specifies the relevant scale.
`outputFile`	The file to write the output to.
`outputWidth, outputHeight, outputUnits`	The width, height, and units for the output file.
`faMethod`	The method to pass on to `psych::fa()`.
`n.iter`	The number of iterations to pass on to `psych::fa()`.
`skipRegex`	A character vector of length 2 containing two regular expressions; if the two scales both match one or both of those regular expressions, that cell is skipped.
`headingLevel`	The level for the heading; especially useful when knitting an Rmd partial.
`printAbbreviations`	Whether to print a table with the abbreviations that are used.
`drawPlot`	Whether to draw the plot or only return it.

`returnPlotOnly`	Whether to return the plot only, or the entire object.
`x`	The object to print.
`quiet`	Whether to be quiet or chatty.
`echoPartial`	Whether to echo the code in the Rmd partial.
`partialFile`	Can be used to override the Rmd partial file.
`...`	Additional arguments are passed on the respective default methods.

Value

A `ggplot2::ggplot()` plot.

Examples

```
### Load data from psych package
data(bfi, package= 'psych');

### Specify scales
bfiScales <-
  list(Agreeableness     = paste0("Agreeableness_item_", 1:5),
       Conscientiousness = paste0("Conscientiousness_item_", 1:5),
       Extraversion      = paste0("Extraversion_item_", 1:5),
       Neuroticism       = paste0("Neuroticism_item_", 1:5),
       Openness          = paste0("Openness_item_", 1:5));

names(bfi) <- c(unlist(bfiScales),
                c('gender', 'education', 'age'));

### Only select first two and the first three items to
### keep it quick; just pass the full 'bfiScales'
### object to run for all five the full scales
CIM(bfi,
    scales=lapply(bfiScales, head, 3)[1:2],
    n.iter=10);
```

`cohensdCI` *The distribution of Cohen's* d

Description

These functions use some conversion to and from the *t* distribution to provide the Cohen's *d* distribution. There are four versions that act similar to the standard distribution functions (the `d.`, `p.`, `q.`, and `r.` functions, and their longer aliases `.Cohensd`), three convenience functions (`pdExtreme`, `pdMild`, and `pdInterval`), a function to compute the confidence interval for a Cohen's *d* estimate `cohensdCI`, and a function to compute the sample size required to obtain a confidence interval around a Cohen's *d* estimate with a specified accuracy (`pwr.cohensdCI` and its alias `pwr.confIntd`).

Usage

```
cohensdCI(d, n, conf.level = 0.95, plot = FALSE, silent = TRUE)

dCohensd(x, df = NULL, populationD = 0, n = NULL, n1 = NULL,
  n2 = NULL, silent = FALSE)

pCohensd(q, df, populationD = 0, lower.tail = TRUE)

qCohensd(p, df, populationD = 0, lower.tail = TRUE)

rCohensd(n, df, populationD = 0)

pdInterval(ds, n, populationD = 0)

pdExtreme(d, n, populationD = 0)

pdMild(d, n, populationD = 0)

pwr.cohensdCI(d, w = 0.1, conf.level = 0.95, extensive = FALSE,
  silent = TRUE)
```

Arguments

`n, n1, n2`	Desired number of Cohen's *d* values for `rCohensd` and `rd` (`n`), and the number of participants/datapoints in total (`n`) or in each group (`n1` and `n2`) for `dd`, `dCohensd`, `pdExtreme`, `pdMild`, `pdInterval`, and `cohensdCI`.
`conf.level`	The level of confidence of the confidence interval.
`plot`	Whether to show a plot of the sampling distribution of Cohen's *d* and the confidence interval. This can only be used if specifying one value for `d`, `n`, and `conf.level`.
`silent`	Whether to provide `FALSE` or suppress (`TRUE`) warnings. This is useful because function 'qt', which is used under the hood (see `qt()` for more information), warns that 'full precision may not have been achieved' when the density of the distribution is very close to zero. This is normally no cause for concern, because with sample sizes this big, small deviations have little impact.
`x, q, d`	Vector of quantiles, or, in other words, the value(s) of Cohen's *d*.
`df`	Degrees of freedom.
`populationD`	The value of Cohen's *d* in the population; this determines the center of the Cohen's *d* distribution. I suppose this is the noncentrality parameter.
`lower.tail`	logical; if TRUE (default), probabilities are the likelihood of finding a Cohen's *d* smaller than the specified value; otherwise, the likelihood of finding a Cohen's *d* larger than the specified value.
`p`	Vector of probabilites (*p*-values).
`ds`	A vector with two Cohen's *d* values.
`w`	The desired maximum 'half-width' or margin of error of the confidence interval.
`extensive`	Whether to only return the required sample size, or more extensive results.

Details

The functions use `convert.d.to.t()` and `convert.t.to.d()` to provide the Cohen's *d* distribution.

The confidence interval functions, `cohensdCI` and `pwr.cohensdCI`, now use the same method as MBESS (a slightly adapted version of the `MBESS` function `conf.limits.nct` is used).

More details about `cohensdCI` and `pwr.cohensdCI` are provided in Peters & Crutzen (2017).

Value

`dCohensd` (or `dd`) gives the density, `pCohensd` (or `pd`) gives the distribution function, `qCohensd` (or `qd`) gives the quantile function, and `rCohensd` (or `rd`) generates random deviates.

`pdExtreme` returns the probability (or probabilities) of finding a Cohen's *d* equal to or more extreme than the specified value(s).

`pdMild` returns the probability (or probabilities) of finding a Cohen's *d* equal to or *less* extreme than the specified value(s).

`pdInterval` returns the probability of finding a Cohen's *d* that lies in between the two specified values of Cohen's *d*.

`cohensdCI` provides the confidence interval(s) for a given Cohen's *d* value.

`pwr.cohensdCI` provides the sample size required to obtain a confidence interval for Cohen's *d* with a desired width.

Author(s)

Gjalt-Jorn Peters (Open University of the Netherlands), with the exported MBESS function conf.limits.nct written by Ken Kelley (University of Notre Dame), and with an error noticed by Guy Prochilo (University of Melbourne).

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

References

Peters, G. J. Y. & Crutzen, R. (2017) Knowing exactly how effective an intervention, treatment, or manipulation is and ensuring that a study replicates: accuracy in parameter estimation as a partial solution to the replication crisis. http://dx.doi.org/

Maxwell, S. E., Kelley, K., & Rausch, J. R. (2008). Sample size planning for statistical power and accuracy in parameter estimation. Annual Review of Psychology, 59, 537-63. https://doi.org/10.1146/annurev.psych.59.10300

Cumming, G. (2013). The New Statistics: Why and How. Psychological Science, (November). https://doi.org/10.1177/0956797613504966

See Also

`convert.d.to.t()`, `convert.t.to.d()`, `dt()`, `pt()`, `qt()`, `rt()`

Examples

```
### Confidence interval for Cohen's d of .5
### from a sample of 200 participants, also
### showing this visually: this clearly shows
### how wildly our Cohen's d value can vary
### from sample to sample.
cohensdCI(.5, n=200, plot=TRUE);

### How many participants would we need if we
### would want a more accurate estimate, say
### with a maximum confidence interval width
### of .2?
pwr.cohensdCI(.5, w=.1);

### Show that 'sampling distribution':
cohensdCI(.5,
          n=pwr.cohensdCI(.5, w=.1),
          plot=TRUE);

### Generate 10 random Cohen's d values
rCohensd(10, 20, populationD = .5);

### Probability of findings a Cohen's d smaller than
### .5 if it's 0 in the population (i.e. under the
### null hypothesis)
pCohensd(.5, 64);

### Probability of findings a Cohen's d larger than
### .5 if it's 0 in the population (i.e. under the
### null hypothesis)
1 - pCohensd(.5, 64);

### Probability of findings a Cohen's d more extreme
### than .5 if it's 0 in the population (i.e. under
### the null hypothesis)
pdExtreme(.5, 64);

### Probability of findings a Cohen's d more extreme
### than .5 if it's 0.2 in the population.
pdExtreme(.5, 64, populationD = .2);
```

`computeStatistic_t` *associationMatrix Helper Functions*

Description

These objects contain a number of settings and functions for associationMatrix.

Usage

```
computeStatistic_t(var1, var2, conf.level = 0.95, var.equal = TRUE,
  ...)

computeStatistic_r(var1, var2, conf.level = 0.95, ...)

computeStatistic_f(var1, var2, conf.level = 0.95, ...)

computeStatistic_chisq(var1, var2, conf.level = 0.95, ...)

computeEffectSize_d(var1, var2, conf.level = 0.95, var.equal = TRUE,
  ...)

computeEffectSize_r(var1, var2, conf.level = 0.95, ...)

computeEffectSize_etasq(var1, var2, conf.level = 0.95, ...)

computeEffectSize_omegasq(var1, var2, conf.level = 0.95, ...)

computeEffectSize_v(var1, var2, conf.level = 0.95, bootstrap = FALSE,
  samples = 5000, ...)
```

Arguments

`var1`	One of the two variables for which to compute a statistic or effect size
`var2`	The other variable for which to compute the statistic or effect size
`conf.level`	The confidence for the confidence interval for the effect size
`var.equal`	Whether to test for equal variances (`test`), assume equality (`yes`), or assume unequality (`no`). See `userfriendlyscience::meanDiff()` for more information.
`...`	Any additonal arguments are sometimes used to specify exactly how statistics and effect sizes should be computed.
`bootstrap`	Whether to bootstrap to estimate the confidence interval for Cramer's V. If FALSE, the Fisher's Z conversion is used.
`samples`	If bootstrapping, the number of samples to generate (of course, more samples means more accuracy and longer processing time).

Value

associationMatrixStatDefaults and associationMatrixESDefaults contain the default functions from computeStatistic and computeEffectSize that are called (see the help file for associationMatrix for more details).

The other functions return an object with the relevant statistic or effect size, with a confidence interval for the effect size.

For computeStatistic, this object always contains:

`statistic`	The relevant statistic
`statistic.type`	The type of statistic
`parameter`	The degrees of freedom for this statistic
`p.raw`	The p-value of this statistic for NHST

And in addition, it often contains (among other things, sometimes):

`object`	The object from which the statistics are extracted

For computeEffectSize, this object always contains:

`es`	The point estimate for the effect size
`esc.type`	The type of effect size
`ci`	The confidence interval for the effect size

And in addition, it often contains (among other things, sometimes):

`object`	The object from which the effect size is extracted

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

See Also

`userfriendlyscience::meanDiff()`, `associationMatrix()`

Examples

```
computeStatistic_f(Orange$Tree, Orange$circumference)
computeEffectSize_etasq(Orange$Tree, Orange$circumference)
```

`confIntOmegaSq` *Confidence intervals for Omega Squared*

Description

This function uses the MBESS functions `conf.limits.ncf()` (which has been copied into this package to avoid the dependency on `MBESS`) and `convert.ncf.to.omegasq()` to compute the point estimate and confidence interval for Omega Squared (which have been lifted out of MBESS to avoid importing the whole package)

Usage

```
confIntOmegaSq(var1, var2, conf.level = 0.95)

## S3 method for class 'confIntOmegaSq'
print(x, ..., digits = 2)
```

Arguments

`var1, var2`	The two variables: one should be a factor (or will be made a factor), the other should have at least interval level of measurement. If none of the variables is a factor, the function will look for the variable with the least unique values and change it into a factor.
`conf.level`	Level of confidence for the confidence interval.
`x, digits, ...`	Respectively the object to print, the number of digits to round to, and any additonal arguments to pass on to the `print` function.

Value

A `confIntOmegaSq` object is returned, with as elements:

`input`	The input arguments
`intermediate`	Objects generated while computing the output
`output`	The output of the function, consisting of:
`output$es`	The point estimate
`output$ci`	The confidence interval

Note

Formula 16 in Steiger (2004) is used for the conversion in `convert.ncf.to.omegasq()`.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

References

Steiger, J. H. (2004). Beyond the F test: Effect size confidence intervals and tests of close fit in the analysis of variance and contrast analysis. Psychological Methods, 9(2), 164-82. https://doi.org/10.1037/1082-989X.9.2.164

Examples

```
confIntOmegaSq(mtcars$mpg, mtcars$cyl);
```

confIntProp — *Confidence intervals for proportions, vectorized over all arguments*

Description

This function simply computes confidence intervals for proportions.

Usage

```
confIntProp(x, n, conf.level = 0.95, plot = FALSE)
```

Arguments

`x`	The number of 'successes', i.e. the number of events, observations, or cases that one is interested in.
`n`	The total number of cases or observatons.
`conf.level`	The confidence level.
`plot`	Whether to plot the confidence interval in the binomial distribution.

Details

This function is the adapted source code of `binom.test()`. Ir uses `pbeta()`, with some lines of code taken from the `binom.test()` source. Specifically, the count for the low category is specified as first 'shape argument' to `pbeta()`, and the total count (either the sum of the count for the low category and the count for the high category, or the total number of cases if `compareHiToLo` is `FALSE`) minus the count for the low category as the second 'shape argument'.

Value

The confidence interval bounds in a twodimensional matrix, with the first column containing the lower bound and the second column containing the upper bound.

Author(s)

Unknown (see `binom.test()`; adapted by Gjalt-Jorn Peters)

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

See Also

`binom.test()` and `ggProportionPlot`, `the function for which this was written.`

Examples

```
### Simple case
confIntProp(84, 200);

### Using vectors
confIntProp(c(2,3), c(10, 20), conf.level=c(.90, .95, .99));
```

`confIntR` *A function to compute a correlation's confidence interval*

Description

This function computes the confidence interval for a given correlation and its sample size. This is useful to obtain confidence intervals for correlations reported in papers when informing power analyses.

Usage

```
confIntR(r, N, conf.level = 0.95, plot = FALSE)
```

Arguments

`r`	The observed correlation coefficient.
`N`	The sample size of the sample where the correlation was computed.
`conf.level`	The desired confidence level of the confidence interval.
`plot`	Whether to show a plot.

Value

The confidence interval(s) in a matrix with two columns. The left column contains the lower bound, the right column the upper bound. The `rownames()` are the observed correlations, and the `colnames()` are 'lo' and 'hi'. The confidence level and sample size are stored as attributes. The results are returned like this to make it easy to access single correlation coefficients from the resulting object (see the examples).

Author(s)

Douglas Bonett (UC Santa Cruz, United States), with minor edits by Murray Moinester (Tel Aviv University, Israel) and Gjalt-Jorn Peters (Open University of the Netherlands, the Netherlands).

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

References

Bonett, D. G., Wright, T. A. (2000). Sample size requirements for estimating Pearson, Kendall and Spearman correlations. *Psychometrika, 65*, 23-28.

Bonett, D. G. (2014). CIcorr.R and sizeCIcorr.R http://people.ucsc.edu/~dgbonett/psyc181.html

Moinester, M., & Gottfried, R. (2014). Sample size estimation for correlations with pre-specified confidence interval. *The Quantitative Methods of Psychology, 10*(2), 124-130. http://www.tqmp.org/RegularArticles/vol10-2/p124/p124.pdf

Peters, G. J. Y. & Crutzen, R. (forthcoming) An easy and foolproof method for establishing how effective an intervention or behavior change method is: required sample size for accurate parameter estimation in health psychology.

See Also

`confIntR()`

Examples

```
### To request confidence intervals for one correlation
confIntR(.3, 100);

### The lower bound of a single correlation
confIntR(.3, 100)[1];

### To request confidence intervals for multiple correlations:
confIntR(c(.1, .3, .5), 250);

### The upper bound of the correlation of .5:
confIntR(c(.1, .3, .5), 250)['0.5', 'hi'];
```

`convert` *conversion functions*

Description

These are a number of functions to convert statistics and effect size measures from/to each other.

Arguments

`chisq, cohensf, cohensfsq, d, etasq, f, logodds, means, omegasq, or, p, r, t, z`	The value of the relevant statistic or effect size.
`ncf`	The value of a noncentrality parameter of the F distribution.

`n, n1, n2, N, ns`	The number of observations that the r or t value is based on, or the number of observations in each of the two groups for an anova, or the total number of participants when specifying a noncentrality parameter.
`df, df1, df2`	The degrees of freedrom for that statistic (for F, the first one is the numerator (i.e. the effect), and the second one the denominator (i.e. the error term).
`proportion`	The proportion of participants in each of the two groups in a t-test or anova. This is used to compute the sample size in each group if the group sizes are unknown. Thus, if you only provide df1 and df2 when converting an F value to a Cohen's d value, equal group sizes are assumed.
`b`	The value of a regression coefficient.
`se, sds`	The standard error of standard errors of the relevant statistic (e.g. of a regression coefficient) or variables.
`minDim`	The smallest of the number of columns and the number of rows of the crosstable for which the chisquare is translated to a Cramer's V value.
`lower.tail`	For the F and chisquare distributions, whether to get the probability of the lower or upper tail.
`akfEq8`	When converting Cohen's *d* to *r*, for small sample sizes, bias is introduced when the commonly suggested formula is used (Aaron, Kromrey & Ferron, 1998). Therefore, by default, this function uses different equations depending on the sample size (for n < 50 and for n > 50). When `akfEq8` is set to TRUE or FALSE, the corresponding action is taken; when `akfEq8` is not logical (i.e. TRUE or FALSE), the function depends on the sample size.
`var.equal`	Whether to compute the value of *t* or Cohen's *d* assuming equal variances ('yes'), unequal variances ('no'), or whether to test for the difference ('test').

Details

Note that by default, the behavior of `convert.d.to.r` depends on the sample size (see Bruce, Kromrey & Ferron, 1998).

Value

The converted value as a numeric value.

Author(s)

Gjalt-Jorn Peters and Peter Verboon

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

References

Aaron, B. Kromrey J. D. & Ferron, J. (1998) *Equating "r"-based and "d"-based Effect Size Indices: Problems with a Commonly Recommended Formula.* Paper presented at the Annual Meeting of the Florida Educational Research Association (43rd, Orlando, FL, November 2-4, 1998).

Examples

```
convert.t.to.r(t=-6.46, n=200);
convert.r.to.t(r=-.41, n=200);

### Compute some p-values
convert.t.to.p(4.2, 197);
convert.chisq.to.p(5.2, 3);
convert.f.to.p(8.93, 3, 644);

### Convert d to r using both equations
convert.d.to.r(d=.2, n1=5, n2=5, akfEq8 = FALSE);
convert.d.to.r(d=.2, n1=5, n2=5, akfEq8 = TRUE);
```

`convert.cer.to.d` *Helper functions for Numbers Needed for Change*

Description

These functions are used by `behaviorchange::nnc()` to compute the Numbers Needed for Change, but are also available for manual use.

Usage

```
convert.cer.to.d(cer, eer, eventDesirable = TRUE, eventIfHigher = TRUE)

convert.d.to.eer(d, cer, eventDesirable = TRUE, eventIfHigher = TRUE)

convert.d.to.nnc(d, cer, r = 1, eventDesirable = TRUE,
  eventIfHigher = TRUE)

convert.eer.to.d(eer, cer, eventDesirable = TRUE, eventIfHigher = TRUE)
```

Arguments

`cer`	The Control Event Rate.
`eer`	The Experimental Event Rate.
`eventDesirable`	Whether an event is desirable or undesirable.
`eventIfHigher`	Whether scores above or below the threshold are considered 'an event'.
`d`	The value of Cohen's *d*.
`r`	The correlation between the determinant and behavior (for mediated Numbers Needed for Change).

Value

The converted value.

Author(s)

Gjalt-Jorn Peters & Stefan Gruijters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

References

Gruijters, S. L., & Peters, G. Y. (2019). Gauging the impact of behavior change interventions: A tutorial on the Numbers Needed to Treat. *PsyArXiv.* doi:10.31234/osf.io/2bau7

See Also

`behaviorchange::nnc()`

Examples

```
convert.d.to.eer(d=.5, cer=.25);
convert.d.to.nnc(d=.5, cer=.25);
```

`convertToNumeric` *Conveniently convert vectors to numeric*

Description

Tries to 'smartly' convert factor and character vectors to numeric.

Usage

```
convertToNumeric(vector, byFactorLabel = FALSE)
```

Arguments

`vector`	The vector to convert.
`byFactorLabel`	When converting factors, whether to do this by their label value (`TRUE`) or their level value (`FALSE`).

Value

The converted vector.

Examples

```
ufs::convertToNumeric(as.character(1:8));
```

`cramersV` *Cramer's V and its confidence interval*

Description

These functions compute the point estimate and confidence interval for Cramer's V.

Usage

```
cramersV(x, y = NULL, digits = 2)

## S3 method for class 'CramersV'
print(x, digits = x$input$digits, ...)

confIntV(x, y = NULL, conf.level = 0.95, samples = 500, digits = 2,
  method = c("bootstrap", "fisher"), storeBootstrappingData = FALSE)

## S3 method for class 'confIntV'
print(x, digits = x$input$digits, ...)
```

Arguments

`x`	Either a crosstable to analyse, or one of two vectors to use to generate that crosstable. The vector should be a factor, i.e. a categorical variable identified as such by the 'factor' class).
`y`	If x is a crosstable, y can (and should) be empty. If x is a vector, y must also be a vector.
`digits`	Minimum number of digits after the decimal point to show in the result.
`...`	Any additional arguments are passed on to the `print` function.
`conf.level`	Level of confidence for the confidence interval.
`samples`	Number of samples to generate when bootstrapping.
`method`	Whether to use Fisher's Z or bootstrapping to compute the confidence interval.
`storeBootstrappingData`	Whether to store (or discard) the data generating during the bootstrapping procedure.

Value

A point estimate or a confidence interval for Cramer's V, an effect size to describe the association between two categorical variables.

Examples

```
### Get confidence interval for Cramer's V
### Note that by using 'table', and so removing the raw data, inhibits
### bootstrapping, which could otherwise take a while.
confIntV(table(infert$education, infert$induced));
```

`dataShape` *normalityAssessment and samplingDistribution*

Description

normalityAssessment can be used to assess whether a variable and the sampling distribution of its mean have an approximately normal distribution.

Usage

```
dataShape(sampleVector, na.rm = TRUE, type = 2, digits = 2,
  conf.level = 0.95, plots = TRUE, xLabs = NA, yLabs = NA,
  qqCI = TRUE, labelOutliers = TRUE, sampleSizeOverride = NULL)

## S3 method for class 'dataShape'
print(x, digits = x$input$digits,
  extraNotification = TRUE, ...)

## S3 method for class 'dataShape'
pander(x, digits = x$input$digits,
  extraNotification = TRUE, ...)

normalityAssessment(sampleVector, samples = 10000, digits = 2,
  samplingDistColor = "#2222CC", normalColor = "#00CC00",
  samplingDistLineSize = 2, normalLineSize = 1,
  xLabel.sampleDist = NULL, yLabel.sampleDist = NULL,
  xLabel.samplingDist = NULL, yLabel.samplingDist = NULL,
  sampleSizeOverride = TRUE)

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

## S3 method for class 'normalityAssessment'
pander(x, headerPrefix = "#####",
  suppressPlot = FALSE, ...)

samplingDistribution(popValues = c(0, 1), popFrequencies = c(50, 50),
  sampleSize = NULL, sampleFromPop = FALSE, ...)
```

Arguments

`sampleVector`	Numeric vector containing the sample data.
`na.rm`	Whether to remove missing data first.
`type`	Type of skewness and kurtosis to compute; either 1 (g1 and g2), 2 (G1 and G2), or 3 (b1 and b2). See Joanes & Gill (1998) for more information.
`digits`	Number of digits to use when printing results.
`conf.level`	Confidence of confidence intervals.
`plots`	Whether to display plots.
`xLabs, yLabs`	The axis labels for the three plots (should be vectors of three elements; the first specifies the X or Y axis label for the rightmost plot (the histogram), the second for the middle plot (the QQ plot), and the third for the rightmost plot (the box plot).
`qqCI`	Whether to show the confidence interval for the QQ plot.
`labelOutliers`	Whether to label outliers with their row number in the box plot.
`sampleSizeOverride`	Whether to use the sample size of the sample as sample size for the sampling distribution, instead of the sampling distribution size. This makes sense, because otherwise, the sample size and thus sensitivity of the null hypothesis significance tests is a function of the number of samples used to generate the sampling distribution.
`x`	The object to print/pander.
`extraNotification`	Whether to be particularly informative.
`...`	Additional arguments are passed on, usually to the default methods.
`samples`	Number of samples to use when constructing sampling distribution.
`samplingDistColor`	Color to use when drawing the sampling distribution.
`normalColor`	Color to use when drawing the standard normal curve.
`samplingDistLineSize`	Size of the line used to draw the sampling distribution.
`normalLineSize`	Size of the line used to draw the standard normal distribution.
`xLabel.sampleDist`	Label of x axis of the distribution of the sample.
`yLabel.sampleDist`	Label of y axis of the distribution of the sample.
`xLabel.samplingDist`	Label of x axis of the sampling distribution.
`yLabel.samplingDist`	Label of y axis of the sampling distribution.
`headerPrefix`	A prefix to insert before the heading (e.g. to use Markdown headings).
`suppressPlot`	Whether to suppress (`TRUE`) or print (`FALSE`) the plot.

`popValues`	The possible values (levels) of the relevant variable. For example, for a dichotomous variable, this can be "c(1:2)" (or "c(1, 2)"). Note that samplingDistribution is for manually specifying the frequency distribution (or proportions); if you have a vector with 'raw' data, just call normalityAssessment directly.
`popFrequencies`	The frequencies corresponding to each value in popValues; must be in the same order! See the examples.
`sampleSize`	Size of the sample; the sum of the frequencies if not specified.
`sampleFromPop`	If true, the sample vector is created by sampling from the population information specified; if false, rep() is used to generate the sample vector. Note that is proportions are supplied in popFrequencies, sampling from the population is necessary!

Details

samplingDistribution is a convenient wrapper for normalityAssessment that makes it easy to quickly generate a sample and sampling distribution from frequencies (or proportions).

dataShape computes the skewness and kurtosis.

normalityAssessment provides a number of normality tests and draws histograms of the sample data and the sampling distribution of the mean (most statistical tests assume the latter is normal, rather than the first; normality of the sample data guarantees normality of the sampling distribution of the mean, but if the sample size is sufficiently large, the sampling distribution of the mean is approximately normal even when the sample data are not normally distributed). Note that for the sampling distribution, the degrees of freedom are usually so huge that the normality tests, negligible deviations from normality will already result in very small p-values.

samplingDistribution makes it easy to quickly assess the distribution of a variables based on frequencies or proportions, and dataShape computes skewness and kurtosis.

Value

An object with several results, the most notably of which are:

`plot.sampleDist`	Histogram of sample distribution
`sw.sampleDist`	Shapiro-Wilk normality test of sample distribution
`ad.sampleDist`	Anderson-Darling normality test of sample distribution
`ks.sampleDist`	Kolmogorov-Smirnof normality test of sample distribution
`kurtosis.sampleDist`	Kurtosis for sample distribution
`skewness.sampleDist`	Skewness for sample distribution
`plot.samplingDist`	Histogram of sampling distribution
`sw.samplingDist`	Shapiro-Wilk normality test of sampling distribution
`ad.samplingDist`	Anderson-Darling normality test of sampling distribution

`ks.samplingDist`	Kolmogorov-Smirnof normality test of sampling distribution
`dataShape.samplingDist`	Skewness and kurtosis for sampling distribution

Examples

```
### Note: the 'not run' is simply because running takes a lot of time,
###       but these examples are all safe to run!
## Not run:

normalityAssessment(rnorm(35));

### Create a distribution of three possible values and
### show the sampling distribution for the mean
popValues <- c(1, 2, 3);
popFrequencies <- c(20, 50, 30);
sampleSize <- 100;
samplingDistribution(popValues = popValues,
                     popFrequencies = popFrequencies,
                     sampleSize = sampleSize);

### Create a very skewed distribution of ten possible values
popValues <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
popFrequencies <- c(2, 4, 8, 6, 10, 15, 12, 200, 350, 400);
samplingDistribution(popValues = popValues,
                     popFrequencies = popFrequencies,
                     sampleSize = sampleSize, digits=5);

## End(Not run)
```

`descr` *descr (or descriptives)*

Description

This function provides a number of descriptives about your data, similar to what SPSS's DESCRIPTIVES (often called with DESCR) does.

Usage

```
descr(x, digits = 4, errorOnFactor = FALSE,
  include = c("central tendency", "spread", "range",
  "distribution shape", "sample size"), maxModes = 1, t = FALSE,
  conf.level = 0.95, quantileType = 2)

## Default S3 method:
```

```
descr(x, digits = 4, errorOnFactor = FALSE,
  include = c("central tendency", "spread", "range",
  "distribution shape", "sample size"), maxModes = 1, t = FALSE,
  conf.level = 0.95, quantileType = 2)

## S3 method for class 'descr'
print(x, digits = attr(x, "digits"), t = attr(x,
  "transpose"), row.names = FALSE, ...)

## S3 method for class 'descr'
pander(x, headerPrefix = "", headerStyle = "**", ...)

## S3 method for class 'descr'
as.data.frame(x, row.names = NULL, optional = FALSE,
  ...)

## S3 method for class 'data.frame'
descr(x, ...)
```

Arguments

`x`	The vector for which to return descriptives.
`digits`	The number of digits to round the results to when showing them.
`errorOnFactor`	Whether to show an error when the vector is a factor, or just show the frequencies instead.
`include`	Which elements to include when showing the results.
`maxModes`	Maximum number of modes to display: displays "multi" if more than this number of modes if found.
`t`	Whether to transpose the dataframes when printing them to the screen (this is easier for users relying on screen readers).
`conf.level`	Confidence of confidence interval around the mean in the central tendency measures.
`quantileType`	The type of quantiles to be used to compute the interquartile range (IQR). See `quantile` for more information.
`row.names`	Whether to show row names (`TRUE`) or not (`FALSE`).
`...`	Additional arguments are passed to the default `print` and `pander` methods.
`headerPrefix`	The prefix for the heading; can be used to insert hashes (#) to create Markdown headings.
`headerStyle`	A string to insert before and after the heading (to make stuff bold or italic in Markdown).
`optional`	Provided for compatibility with the default `as.data.frame()` method - see that help page for details.

Details

Note that R (of course) has many similar functions, such as `summary`, `psych::describe()` in the excellent psych::psych package.

The Hartigans' Dip Test may be unfamiliar to users; it is a measure of uni- vs. multidimensionality, computed by `diptest::dip.test()` from the `dip.test` package. Depending on the sample size, values over .025 can be seen as mildly indicative of multimodality, while values over .05 probably warrant closer inspection (the p-value can be obtained using `diptest::dip.test()`; also see Table 1 of Hartigan & Hartigan (1985) for an indication as to critical values).

Value

A list of dataframes with the requested values.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

References

Hartigan, J. A.; Hartigan, P. M. The Dip Test of Unimodality. Ann. Statist. 13 (1985), no. 1, 70–84. doi:10.1214/aos/1176346577. http://projecteuclid.org/euclid.aos/1176346577.

See Also

`summary`, [psych::describe()

Examples

```
descr(mtcars$mpg);
```

`diamondCoordinates` *Basic ggplot2 diamond plot layer construction functions*

Description

These functions are used by `diamondPlot()` to construct a diamond plot. It's normally not necessary to call this function directly: instead, use `meansDiamondPlot()`, `meanSDtoDiamondPlot()`, and `factorLoadingDiamondCIplot()`.

Usage

```
diamondCoordinates(values, otherAxisValue = 1,
  direction = "horizontal", autoSize = NULL, fixedSize = 0.15)

ggDiamondLayer(data, ciCols = 1:3, colorCol = NULL,
  generateColors = NULL, fullColorRange = NULL, color = "black",
  lineColor = NA, otherAxisCol = 1:nrow(data), autoSize = NULL,
  fixedSize = 0.15, direction = "horizontal", ...)

rawDataDiamondLayer(dat, items = NULL, itemOrder = 1:length(items),
  dataAlpha = 0.1, dataColor = "#444444", jitterWidth = 0.5,
  jitterHeight = 0.4, size = 3, ...)

varsToDiamondPlotDf(dat, items = NULL, labels = NULL,
  decreasing = NULL, conf.level = 0.95)
```

Arguments

`values`	A vector of 2 or more values that are used to construct the diamond coordinates. If three values are provided, the middle one becomes the diamond's center. If two, four, or more values are provided, the median becomes the diamond's center.
`otherAxisValue`	The value on the other axis to use to compute the coordinates; this will be the Y axis value of the points of the diamond (if `direction` is 'horizontal') or the X axis value (if `direction` is 'vertical').
`direction`	Whether the diamonds should be constructed horizontally or vertically.
`autoSize`	Whether to make the height of each diamond conditional upon its length (the width of the confidence interval).
`fixedSize`	If not using relative heights, `fixedSize` determines the height to use.
`data, dat`	A dataframe (or matrix) containing lower bounds, centers (e.g. means), and upper bounds of intervals (e.g. confidence intervals) for `ggDiamondLayer` or items and raw data for `varsToDiamondPlotDf` and `rawDataDiamondLayer`.
`ciCols`	The columns in the dataframe with the lower bounds, centers (e.g. means), and upper bounds (in that order).
`colorCol`	The column in the dataframe containing the colors for each diamond, or a vector with colors (with as many elements as the dataframe has rows).
`generateColors`	A vector with colors to use to generate a gradient. These colors must be valid arguments to `colorRamp()` (and therefore, to `col2rgb()`).
`fullColorRange`	When specifying a gradient using `generateColors`, it is usually desirable to specify the minimum and maximum possible value corresponding to the outer anchors of that gradient. For example, when plotting numbers from 0 to 100 using a gradient from 'red' through 'orange' to 'green', none of the means may actually be 0 or 100; the lowest mean may be, for example, 50. If no `fullColorRange` is specified, the diamond representing that lowest mean of 50 wil be red, not orange. When specifying the `fullColorRange`, the lowest and

	highest 'colors' in `generateColors` are anchored to the minimum and maximum values of `fullColorRange`.
`color`	When no colors are automatically generated, all diamonds will have this color.
`lineColor`	If NA, lines will have the same colors as the diamonds' fill. If not NA, must be a valid color, which is then used as line color. Note that e.g. `linetype` and `color` can be used as well, which will be passed on to `geom_polygon()`.
`otherAxisCol`	A vector of values, or the index of the column in the dataframe, that specifies the values for the Y axis of the diamonds. This should normally just be a vector of consecutive integers.
`...`	Any additional arguments are passed to `geom_polygon()`. This can be used to set, for example, the `alpha` value of the diamonds. Additional arguments for `rawDataDiamondLayer` are passed on to `geom_jitter()`.
`items`	The items from the dataframe to include in the diamondplot or dataframe.
`itemOrder`	Order of the items to use (if not sorting).
`dataAlpha`	This determines the alpha (transparency) of the data points.
`dataColor`	The color of the data points.
`jitterWidth`	How much to jitter the individual datapoints horizontally.
`jitterHeight`	How much to jitter the individual datapoints vertically.
`size`	The size of the data points.
`labels`	The item labels to add to the dataframe.
`decreasing`	Whether to sort the items (rows) in the dataframe decreasing (TRUE), increasing (FALSE), or not at all (NULL).
`conf.level`	The confidence of the confidence intervals.

Value

`ggDiamondLayer` returns a `ggplot()` `geom_polygon()` object, which can then be used in `ggplot()` plots (as `diamondPlot()` does).

`diamondCoordinates` returns a set of four coordinates that together specify a diamond.

`varsToDiamondPlotDf` returns a dataframe of diamondCoordinates.

`rawDataDiamondLayer` returns a `geom_jitter()` object.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

See Also

`meansDiamondPlot()`, `meanSDtoDiamondPlot()`, `factorLoadingDiamondCIplot()`, `diamondPlot()`

Examples

```
## Not run:
### (Don't run this example as a test, because we
###  need the ggplot function which isn't part of
###  this package.)

### The coordinates for a simple diamond
diamondCoordinates(values = c(1,2,3));

### Plot this diamond
ggplot() + ggDiamondLayer(data.frame(1,2,3));

## End(Not run)
```

`diamondPlot` *Basic diamond plot construction function*

Description

This function constructs a diamond plot using `ggDiamondLayer()`. It's normally not necessary to call this function directly: instead, use `meansDiamondPlot()` `meanSDtoDiamondPlot()`, and `factorLoadingDiamondCIplot()`.

Usage

```
diamondPlot(data, ciCols = 1:3, colorCol = NULL, otherAxisCol = NULL,
  yValues = NULL, yLabels = NULL, ylab = NULL, autoSize = NULL,
  fixedSize = 0.15, xlab = "Effect Size Estimate",
  theme = ggplot2::theme_bw(), color = "black",
  returnLayerOnly = FALSE, outputFile = NULL, outputWidth = 10,
  outputHeight = 10, ggsaveParams = list(units = "cm", dpi = 300, type
  = "cairo"), ...)
```

Arguments

`data`	A dataframe (or matrix) containing lower bounds, centers (e.g. means), and upper bounds of intervals (e.g. confidence intervals).
`ciCols`	The columns in the dataframe with the lower bounds, centers (e.g. means), and upper bounds (in that order).
`colorCol`	The column in the dataframe containing the colors for each diamond, or a vector with colors (with as many elements as the dataframe has rows).
`otherAxisCol`	The column in the dataframe containing the values that determine where on the Y axis the diamond should be placed. If this is not available in the dataframe, specify it manually using `yValues`.

`yValues`	The values that determine where on the Y axis the diamond should be placed (can also be a column in the dataframe; in that case, use `otherAxisCol`.
`yLabels`	The labels to use for for each diamond (placed on the Y axis).
`autoSize`	Whether to make the height of each diamond conditional upon its length (the width of the confidence interval).
`fixedSize`	If not using relative heights, `fixedSize` determines the height to use.
`xlab, ylab`	The labels of the X and Y axes.
`theme`	The theme to use.
`color`	Color to use if colors are specified for each diamond.
`returnLayerOnly`	Set this to TRUE to only return the `ggplot()` layer of the diamondplot, which can be useful to include it in other plots.
`outputFile`	A file to which to save the plot.
`outputWidth, outputHeight`	Width and height of saved plot (specified in centimeters by default, see `ggsaveParams`).
`ggsaveParams`	Parameters to pass to ggsave when saving the plot.
`...`	Additional arguments will be passed to `ggDiamondLayer()`.

Value

A `ggplot2::ggplot()` plot with a `ggDiamondLayer()` is returned.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

See Also

`meansDiamondPlot()`, `meanSDtoDiamondPlot()`, `ggDiamondLayer()`, `factorLoadingDiamondCIplot()`

Examples

```
tmpDf <- data.frame(lo = c(1, 2, 3),
                    mean = c(1.5, 3, 5),
                    hi = c(2, 4, 10),
                    color = c('green', 'red', 'blue'));

### A simple diamond plot
diamondPlot(tmpDf);

### A diamond plot using the specified colours
diamondPlot(tmpDf, colorCol = 4);

### A diamond plot using automatically generated colours
### using a gradient
```

```
diamondPlot(tmpDf, generateColors=c('green', 'red'));

### A diamond plot using automatically generated colours
### using a gradient, specifying the minimum and maximum
### possible values that can be attained
diamondPlot(tmpDf, generateColors=c('green', 'red'),
            fullColorRange=c(1, 10));
```

`disattenuate.d` *Disattentuate a Cohen's d estimate for unreliability in the continuous variable*

Description

Disattentuate a Cohen's d estimate for unreliability in the continuous variable

Usage

```
disattenuate.d(d, reliability)
```

Arguments

`d`	The (attenuated) value of Cohen's d
`reliability`	The reliability of the measurements of the continuous variable

Value

The disattenuated value of Cohen's d

Examples

```
disattenuate.d(.5, .8);
```

`disattenuate.r` *Disattentuate a Pearson's r estimate for unreliability*

Description

Disattentuate a Pearson's r estimate for unreliability

Usage

```
disattenuate.r(r, reliability1, reliability2)
```

Arguments

`r`	The (attenuated) value of Pearson's r
`reliability1, reliability2`	The reliabilities of the two variables

Value

The disattenuated value of Pearson's r

Examples

```
disattenuate.r(.5, .8, .9);
```

`duoComparisonDiamondPlot` *meansComparisonDiamondPlot and duoComparisonDiamondPlot*

Description

These are two diamond plot functions to conveniently make diamond plots to compare subgroups or different samples. They are both based on a univariate diamond plot where colors are used to distinguish the data points and diamonds of each subgroup or sample. The means comparison diamond plot produces only this plot, while the duo comparison diamond plot combines it with a diamond plot visualising the effect sizes of the associations. The latter currently only works for two subgroups or samples, while the simple meansComparisonDiamondPlot also works when comparing more than two sets of datapoints. These functions are explained more in detail in Peters (2017).

Usage

```
duoComparisonDiamondPlot(dat, items = NULL, compareBy = NULL,
  labels = NULL, compareByLabels = NULL, decreasing = NULL,
  conf.level = c(0.95, 0.95), showData = TRUE, dataAlpha = 0.1,
  dataSize = 3, comparisonColors = viridis::viridis(length(unique(dat[,
  compareBy]))), associationsColor = "grey", alpha = 0.33,
  jitterWidth = 0.5, jitterHeight = 0.4, xlab = c("Scores and means",
  "Effect size estimates"), ylab = c(NULL, NULL), plotTitle = NULL,
  theme = ggplot2::theme_bw(), showLegend = TRUE,
  legend.position = "top", lineSize = 1, drawPlot = TRUE,
  xbreaks = "auto", outputFile = NULL, outputWidth = 10,
  outputHeight = 10, ggsaveParams = list(units = "cm", dpi = 300, type
  = "cairo"), ...)

meansComparisonDiamondPlot(dat, items = NULL, compareBy = NULL,
  labels = NULL, compareByLabels = NULL, decreasing = NULL,
  sortBy = NULL, conf.level = 0.95, showData = TRUE,
  dataAlpha = 0.1, dataSize = 3,
  comparisonColors = viridis::viridis(length(unique(dat[, compareBy]))),
  alpha = 0.33, jitterWidth = 0.5, jitterHeight = 0.4,
  xlab = "Scores and means", ylab = NULL, plotTitle = NULL,
  theme = ggplot2::theme_bw(), showLegend = TRUE,
  legend.position = "top", lineSize = 1, xbreaks = "auto",
  outputFile = NULL, outputWidth = 10, outputHeight = 10,
  ggsaveParams = list(units = "cm", dpi = 300, type = "cairo"), ...)
```

Arguments

`dat`	The dataframe containing the relevant variables.
`items`	The variables to plot (on the y axis).
`compareBy`	The variable by which to compare (i.e. the variable indicating to which subgroup or sample a row in the dataframe belongs).
`labels`	The labels to use on the y axis; these values will replace the variable names in the dataframe (specified in `items`).
`compareByLabels`	The labels to use to replace the value labels of the `compareBy` variable.
`decreasing`	Whether to sort the variables by their mean values (`NULL` to not sort, `TRUE` to sort in descending order (i.e. items with lower means are plotted more to the bottom), and `FALSE` to sort in ascending order (i.e. items with lower means are plotted more to the top).
`conf.level`	The confidence level of the confidence intervals specified by the diamonds for the means (for `meansComparisonDiamondPlot`) and for both the means and effect sizes (for `duoComparisonDiamondPlot`).
`showData`	Whether to plot the data points.
`dataAlpha`	The transparency (alpha channel) value for the data points: a value between 0 and 1, where 0 denotes complete transparency and 1 denotes complete opacity.
`dataSize`	The size of the data points.
`comparisonColors`	The colors to use for the different subgroups or samples. This should be a vector of valid colors with at least as many elements as sets of data points that should be plotted.
`associationsColor`	For `duoComparisonDiamondPlot`, the color to use to plot the effect sizes in the right-hand plot.
`alpha`	The alpha channel (transparency) value for the diamonds: a value between 0 and 1, where 0 denotes complete transparency and 1 denotes complete opacity.
`jitterWidth, jitterHeight`	How much noise to add to the data points (to prevent overplotting) in the horizontal (x axis) and vertical (y axis) directions.
`xlab, ylab`	The label to use for the x and y axes (for `duoComparisonDiamondPlot`, must be vectors of two elements). Use `NULL` to not use a label.

`plotTitle`	Optionally, for `meansComparisonDiamondPlot`, a title for the plot (can also be specified for `duoComparisonDiamondPlot`, in which case it's passed on to `meansComparisonDiamondPlot` for the left panel - but note that this messes up the alignment of the two panels).
`theme`	The theme to use for the plots.
`showLegend`	Whether to show the legend (which color represents which subgroup/sample).
`legend.position`	Where to place the legend in `meansComparisonDiamondPlot` (can also be specified for `duoComparisonDiamondPlot`, in which case it's passed on to `meansComparisonDiamondPlot` for the left panel - but note that this messes up the alignment of the two panels).
`lineSize`	The thickness of the lines (the diamonds' strokes).
`drawPlot`	Whether to draw the plot, or only (invisibly) return it.
`xbreaks`	Where the breaks (major grid lines, ticks, and labels) on the x axis should be.
`outputFile`	A file to which to save the plot.
`outputWidth, outputHeight`	Width and height of saved plot (specified in centimeters by default, see `ggsaveParams`).
`ggsaveParams`	Parameters to pass to ggsave when saving the plot.
`...`	Any additional arguments are passed to `diamondPlot()` by `meansComparisonDiamondPlot` and to both `meansComparisonDiamondPlot` and `associationsDiamondPlot()` by `duoComparisonDiamondPlot`.
`sortBy`	If the variables should be sorted (see `decreasing`), this variable specified which subgroup should be sorted by. Therefore, the value specified here must be a value label ('level label') of the `compareBy` variable.

Details

These functions are explained in Peters (2017).

Value

A Diamond plots: a `ggplot2::ggplot()` plot `meansComparisonDiamondPlot`, and a `gtable()` by `duoComparisonDiamondPlot`.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

References

Peters, G.-J. Y. (2017). Diamond Plots: a tutorial to introduce a visualisation tool that facilitates interpretation and comparison of multiple sample estimates while respecting their inaccuracy. *PsyArXiv.* http://doi.org/10.17605/OSF.IO/9W8YV

See Also

`diamondPlot()`, `meansDiamondPlot()`, `behaviorchange::CIBER()`

Examples

```
meansComparisonDiamondPlot(mtcars,
                           items=c('disp', 'hp'),
                           compareBy='vs',
                           xbreaks=c(100,200, 300, 400));
meansComparisonDiamondPlot(chickwts,
                           items='weight',
                           compareBy='feed',
                           xbreaks=c(100,200,300,400),
                           showData=FALSE);
duoComparisonDiamondPlot(mtcars,
                         items=c('disp', 'hp'),
                         compareBy='vs',
                         xbreaks=c(100,200, 300, 400));
```

`extractVarName` *Extract variable names*

Description

Functions often get passed variables from within dataframes or other lists. However, printing these names with all their dollar signs isn't very userfriendly. This function simply uses a regular expression to extract the actual name.

Usage

```
extractVarName(x)
```

Arguments

x	A character vector of one or more variable names.

Value

The actual variables name, with all containing objectes stripped off.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

Examples

```
extractVarName('mtcars$mpg');
```

`faConfInt` *Extract confidence bounds from psych's factor analysis object*

Description

This function contains some code from a function in psych::psych-package that's not exported `print.psych.fa.ci` but useful nonetheless. It basically takes the outcomes of a factor analysis and extracted the confidence intervals.

Usage

```
faConfInt(fa)
```

Arguments

`fa`	The object produced by the `psych::fa()` function from the psych::psych-package package. It is important that the `n.iter` argument of`psych::fa()` was set to a realistic number, because otherwise, no confidence intervals will be available.

Details

THis function extract confidence interval bounds and combines them with factor loadings using the code from the `print.psych.fa.ci` in psych::psych-package.

Value

A list of dataframes, one for each extracted factor, with in each dataframe three variables:

`lo`	lower bound of the confidence interval
`est`	point estimate of the factor loading
`hi`	upper bound of the confidence interval

Author(s)

William Revelle (extracted by Gjalt-Jorn Peters)

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

Examples

```
## Not run:
### Not run because it takes too long to run to test it,
### and may produce warnings, both because of the bootstrapping
### required to generate the confidence intervals in fa
faConfInt(psych::fa(Thurstone.33, 2, n.iter=100, n.obs=100));

## End(Not run)
```

`factorLoadingDiamondCIplot` *Two-dimensional visualisation of factor analyses*

Description

This function uses the `diamondPlot()` to visualise the results of a factor analyses. Because the factor loadings computed in factor analysis are point estimates, they may vary from sample to sample. The factor loadings for any given sample are usually not relevant; samples are but means to study populations, and so, researchers are usually interested in population values for the factor loadings. However, tables with lots of loadings can quickly become confusing and intimidating. This function aims to facilitate working with and interpreting factor analysis based on confidence intervals by visualising the factor loadings and their confidence intervals.

Usage

```
factorLoadingDiamondCIplot(fa, xlab = "Factor Loading",
  colors = (viridis::viridis_pal())(max(2, fa$factors)), labels = NULL,
  theme = ggplot2::theme_bw(), ...)
```

Arguments

`fa`	The object produced by the `psych::fa()` function from the psych::psych package. It is important that the `n.iter` argument of `psych::fa()` was set to a realistic number, because otherwise, no confidence intervals will be available.
`xlab`	The label for the x axis.
`colors`	The colors used for the factors. The default uses the discrete `viridis()` palette, which is optimized for perceptual uniformity, maintaining its properties when printed in grayscale, and designed for colourblind readers. A vector can also be supplied; the colors must be valid arguments to `colorRamp()` (and therefore, to `col2rgb()`).
`labels`	The labels to use for the items (on the Y axis).
`theme`	The ggplot2 theme to use.
`...`	Additional arguments will be passed to `ggDiamondLayer()`. This can be used to set, for example, the transparency (alpha value) of the diamonds to a lower value using e.g. `alpha=.5`.

Value

A `ggplot2::ggplot()` plot with several `ggDiamondLayer()`s is returned.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

See Also

psych::fa()ss, meansDiamondPlot(), meanSDtoDiamondPlot(), diamondPlot(), ggDiamondLayer()

Examples

```
## Not run:
### (Not run during testing because it takes too long and
###  may generate warnings because of the bootstrapping of
###  the confidence intervals)

factorLoadingDiamondCIplot(psych::fa(Bechtoldt,
                                     nfactors=2,
                                     n.iter=50,
                                     n.obs=200));

### And using a lower alpha value for the diamonds to
### make them more transparent

factorLoadingDiamondCIplot(psych::fa(Bechtoldt,
                                     nfactors=2,
                                     n.iter=50,
                                     n.obs=200),
                           alpha=.5,
                           size=1);

## End(Not run)
```

---

findShortestInterval   *Find the shortest interval*

Description

This function takes a numeric vector, sorts it, and then finds the shortest interval and returns its length.

Usage

```
findShortestInterval(x)
```

Arguments

x	The numeric vector.

Value

The length of the shortest interval.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

Examples

```
findShortestInterval(c(1, 2, 4, 7, 20, 10, 15));
```

formatCI *Pretty formatting of confidence intervals*

Description

Pretty formatting of confidence intervals

Usage

```
formatCI(ci, sep = "; ", prefix = "[", suffix = "]", digits = 2,
  noZero = FALSE)
```

Arguments

`ci`	A confidence interval (a vector of 2 elements; longer vectors work, but I guess that wouldn't make sense).
`sep`	The separator of the values, usually "; " or ", ".
`prefix, suffix`	The prefix and suffix, usually a type of opening and closing parenthesis/bracket.
`digits`	The number of digits to which to round the values.
`noZero`	Whether to strip the leading zero (before the decimal point), as is typically done when following APA style and displaying correlations, *p* values, and other numbers that cannot reach 1 or more.

Value

A character vector of one element.

See Also

`noZero()`, `formatR()`, `formatPvalue()`

Examples

```
### With leading zero ...
formatCI(c(0.55, 0.021));

### ... and without
formatCI(c(0.55, 0.021), noZero=TRUE);
```

`formatPvalue` *Pretty formatting of* p *values*

Description

Pretty formatting of *p* values

Usage

```
formatPvalue(values, digits = 3, spaces = TRUE, includeP = TRUE)
```

Arguments

`values`	The p-values to format.
`digits`	The number of digits to round to. Numbers smaller than this number will be shown as <.001 or <.0001 etc.
`spaces`	Whether to include spaces between symbols, operators, and digits.
`includeP`	Whether to include the 'p' and '='-symbol in the results (the '<' symbol is always included).

Value

A formatted P value, roughly according to APA style guidelines. This means that the noZero function is used to remove the zero preceding the decimal point, and p values that would round to zero given the requested number of digits are shown as e.g. p<.001.

See Also

`formatCI()`, `formatR()`, `noZero()`

Examples

```
formatPvalue(cor.test(mtcars$mpg,
                      mtcars$disp)$p.value);
formatPvalue(cor.test(mtcars$drat,
                      mtcars$qsec)$p.value);
```

formatR *Pretty formatting of correlation coefficients*

Description

Pretty formatting of correlation coefficients

Usage

```
formatR(r, digits = 2)
```

Arguments

`r`	The Pearson correlation to format.
`digits`	The number of digits to round to.

Value

The formatted correlation.

See Also

`noZero()`, `formatCI()`, `formatPvalue()`

Examples

```
formatR(cor(mtcars$mpg, mtcars$disp));
```

getData *Basic SPSS translation functions*

Description

Basic functons to make working with R easier for SPSS users: getData and getDat provide an easy way to load SPSS datafiles, and exportToSPSS to write to a datafile and syntax file that SPSS can import; filterBy and useAll allow easy temporary filtering of rows from the dataframe; mediaan and modus compute the median and mode of ordinal or numeric data.

Usage

```
getData(filename = NULL, file = NULL,
  errorMessage = "[defaultErrorMessage]", applyRioLabels = TRUE,
  use.value.labels = FALSE, to.data.frame = TRUE,
  stringsAsFactors = FALSE, silent = FALSE, ...)

getDat(..., dfName = "dat", backup = TRUE)
```

Arguments

`filename, file`	It is possible to specify a path and filename to load here. If not specified, the default R file selection dialogue is shown. `file` is still available for backward compatibility but will eventually be phased out.
`errorMessage`	The error message that is shown if the file does not exist or does not have the right extension; `[defaultErrorMessage]` is replaced with a default error message (and can be included in longer messages).
`applyRioLabels`	Whether to apply the labels supplied by Rio. This will make variables that has value labels into factors.
`use.value.labels`	Only useful when reading from SPSS files: whether to read variables with value labels as factors (TRUE) or numeric vectors (FALSE).
`to.data.frame`	Only useful when reading from SPSS files: whether to return a dataframe or not.
`stringsAsFactors`	Whether to read strings as strings (FALSE) or factors (TRUE).
`silent`	Whether to suppress potentially useful information.
`...`	Additional options, passed on to the function used to import the data (which depends on the extension of the file).
`dfName`	The name of the dataframe to create in the parent environment.
`backup`	Whether to backup an object with name `dfName`, if one already exists in the parent environment.

Value

getData returns the imported dataframe, with the filename from which it was read stored in the 'filename' attribute.

getDat is a simple wrapper for `getData()` which creates a dataframe in the parent environment, by default with the name 'dat'. Therefore, calling `getDat()` in the console will allow the user to select a file, and the data from the file will then be read and be available as 'dat'. If an object with `dfName` (i.e. 'dat' by default) already exists, it will be backed up with a warning. `getDat()` therefore returns nothing.

mediaan returns the median, or, in the case of a factor where the median is in between two categories, both categories.

modus returns the mode.

Note

getData() currently can't read from LibreOffice or OpenOffice files. There doesn't seem to be a platform-independent package that allows this. Non-CRAN package ROpenOffice from OmegaHat should be able to do the trick, but fails to install (manual download and installation using http://www.omegahat.org produces "ERROR: dependency 'Rcompression' is not available for package 'ROpenOffice'" - and manual download and installation of RCompression produces "Please define LIB_ZLIB; ERROR: configuration failed for package 'Rcompression'"). If you have any suggestions, please let me know!

Examples

```
## Not run:
### Open a dialogue to read an SPSS file
getData();

## End(Not run)
```

`ggBarChart` *Bar chart using ggplot*

Description

This function provides a simple interface to create a `ggplot2::ggplot()` bar chart.

Usage

```
ggBarChart(vector, plotTheme = ggplot2::theme_bw(), ...)
```

Arguments

`vector`	The vector to display in the bar chart.
`plotTheme`	The theme to apply.
`...`	And additional arguments are passed to `ggplot2::geom_bar()`.

Value

A `ggplot2::ggplot()` plot is returned.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

See Also

`ggplot2::geom_bar()`

Examples

```
ggBarChart(mtcars$cyl);
```

ggBoxplot *Box plot using ggplot*

Description

This function provides a simple interface to create a `ggplot` box plot, organising different boxplots by levels of a factor is desired, and showing row numbers of outliers.

Usage

```
ggBoxplot(dat, y = NULL, x = NULL, labelOutliers = TRUE,
  outlierColor = "red", theme = ggplot2::theme_bw(), ...)
```

Arguments

`dat`	Either a vector of values (to display in the box plot) or a dataframe containing variables to display in the box plot.
`y`	If `dat` is a dataframe, this is the name of the variable to make the box plot of.
`x`	If `dat` is a dataframe, this is the name of the variable (normally a factor) to place on the X axis. Separate box plots will be generate for each level of this variable.
`labelOutliers`	Whether or not to label outliers.
`outlierColor`	If labeling outliers, this is the color to use.
`theme`	The theme to use for the box plot.
`...`	Any additional arguments will be passed to `geom_boxplot`.

Details

This function is based on JasonAizkalns' answer to a question on Stack Exchange (Cross Validated; see `http://stackoverflow.com/questions/33524669/labeling-outliers-of-boxplots-in-r`).

Value

A `ggplot` plot is returned.

Author(s)

Jason Aizkalns; implemented in this package (and tweaked a bit) by Gjalt-Jorn Peters.

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

See Also

`geom_boxplot`

Examples

```
### A box plot for miles per gallon in the mtcars dataset:
ggBoxplot(mtcars$mpg);

### And separate for each level of 'cyl' (number of cylinder):
ggBoxplot(mtcars, y='mpg', x='cyl');
```

`ggEasyBar` *Convenience functions for ggplots based on multiple variables*

Description

These are convenience functions to quickly generate plots for multiple variables, with the variables in the y axis.

Usage

```
ggEasyBar(data, items = NULL, labels = NULL, sortByMean = TRUE,
  xlab = NULL, ylab = NULL, scale_fill_function = NULL,
  fontColor = "white", fontSize = 2, labelMinPercentage = 1,
  showInLegend = "both", legendRows = 2, legendValueLabels = NULL,
  biAxisLabels = NULL)

ggEasyRidge(data, items = NULL, labels = NULL, sortByMean = TRUE,
  xlab = NULL, ylab = NULL)
```

Arguments

`data`	The dataframe containing the variables.
`items`	The variable names (if not provided, all variables will be used).
`labels`	Labels can optionally be provided; if they are, these will be used instead of the variable names.
`sortByMean`	Whether to sort the variables by mean value.
`xlab, ylab`	The labels for the x and y axes.
`scale_fill_function`	The function to pass to `ggplot()` to provide the colors of the bars. If `NULL`, set to `ggplot2::scale_fill_viridis_d(labels = legendValueLabels,guide = ggplot2::guide_legenc = NULL,nrow=legendRows,byrow=TRUE))`.
`fontColor, fontSize`	The color and size of the font used to display the labels
`labelMinPercentage`	The minimum percentage that a category must reach before the label is printed (in whole percentages, i.e., on a scale from 0 to 100).

`showInLegend`	What to show in the legend in addition to the values; nothing (`"none"`), the frequencies (`"freq"`), the percentages (`"perc"`), or both (`"both"`). This is only used if only one variable is shown in the plot; afterwise, after all, the absolute frequencies and percentages differ for each variable.
`legendRows`	Number or rows in the legend.
`legendValueLabels`	Labels to use in the legend; must be a vector of the same length as the number of categories in the variables.
`biAxisLabels`	This can be used to specify labels to use if you want to use labels on both the left and right side. This is mostly useful when plotting single questions or semantic differentials. This must be a list with two character vectors, `leftAnchors` and `rightAnchors`, which must each have the same length as the number of items specified in `items`. See the examples for, well, examples.

Value

A `ggplot()` plot is returned.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

See Also

`geom_ridgeline()`, `geom_bar()`

Examples

```
ggEasyBar(mtcars, c('gear', 'carb'));
ggEasyRidge(mtcars, c('disp', 'hp'));

### When plotting single questions, if you want to show the anchors:
ggEasyBar(mtcars, c('gear'),
          biAxisLabels=list(leftAnchors="Fewer",
                            rightAnchors="More"));

### Or for multiple questions (for e.g. semantic differentials):
ggEasyBar(mtcars, c('gear', 'carb'),
          biAxisLabels=list(leftAnchors=c("Fewer", "Lesser"),
                            rightAnchors=c("More", "Greater")));
```

ggProportionPlot *Sample distribution based plotting of proportions*

Description

This function visualises percentages, but avoids a clear cut for the sample point estimate, instead using the confidence (as in confidence interval) to create a gradient. This effectively hinders drawing conclusions on the basis of point estimates, thereby urging a level of caution that is consistent with what the data allows.

Usage

```
ggProportionPlot(dat, items = NULL, loCategory = NULL,
  hiCategory = NULL, subQuestions = NULL, leftAnchors = NULL,
  rightAnchors = NULL, compareHiToLo = TRUE, showDiamonds = FALSE,
  diamonds.conf.level = 0.95, diamonds.alpha = 1, na.rm = TRUE,
  barHeight = 0.4, conf.steps = seq(from = 0.001, to = 0.999, by =
  0.001), scale_color = viridis::viridis(option = "viridis", 2, begin =
  0.5, end = 1), scale_fill = viridis::viridis(option = "viridis", 2,
  begin = 0.5, end = 1), rank.conf = FALSE, linetype = 1,
  theme = ggplot2::theme_bw(), returnPlotOnly = TRUE)

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

## S3 method for class 'ggProportionPlot'
grid.draw(x, ...)
```

Arguments

`dat`	The dataframe containing the items (variables), or a vector.
`items`	The names of the items (variables). If none are specified, all variables in the dataframe are used.
`loCategory`	The value of the low category (usually 0). If not provided, the minimum value is used.
`hiCategory`	The value of the high category (usually 1). If not provided, the maximum value is used.
`subQuestions`	The labels to use for the variables (for example, different questions). The variable names are used if these aren't provided.
`leftAnchors`	The labels for the low categories. The values are used if these aren't provided.
`rightAnchors`	The labels for the high categories. The values are used if these aren't provided.
`compareHiToLo`	Whether to compare the percentage of low category values to the total of the low category values and the high category values, or whether to ignore the high category values and compute the percentage of low category values relative to all cases. This can be useful when a variable has more than two values, and you only want to know/plot the percentage relative to the total number of cases.

`showDiamonds`	Whether to add diamonds to illustrate the confidence intervals.
`diamonds.conf.level`	The confidence level of the diamonds' confidence intervals.
`diamonds.alpha`	The alpha channel (i.e. transparency, or rather 'obliqueness') of the diamonds.
`na.rm`	Whether to remove missing values.
`barHeight`	The height of the bars, or rather, half the height. Use .5 to completely fill the space.
`conf.steps`	The number of steps to use to generate the confidence levels for the proportion.
`scale_color, scale_fill`	A vector with two values (valid colors), that are used for the colors (stroke) and fill for the gradient; both vectors should normally be the same, but if you feel adventurous, you can play around with the number of `conf.steps` and this. If you specify only one color, no gradient is used but a single color (i.e. specifying the same single color for both `scale_color` and `scale_fill` simply draws bars of that color).
`rank.conf`	Whether to let the fill and color gradients use the confidence or the ranked confidence.
`linetype`	The `linetype()` to use (0 = blank, 1 = solid, 2 = dashed, 3 = dotted, 4 = dotdash, 5 = longdash, 6 = twodash).
`theme`	The theme to use.
`returnPlotOnly`	Whether to only return the `ggplot2()` plot or the full object including intermediate values and objects.
`x`	The object to print/plot.
`...`	Any additional arguments are passed on to `print` and `grid.draw`.

Details

This function used `confIntProp()` to compute confidence intervals for proportions at different levels of confidence. The confidence interval bounds at those levels of confidence are then used to draw rectangles with colors in a gradient that corresponds to the confidence level.

Note that percentually, the gradient may not look continuous because at the borders between lighter and darker rectangles, the shade of the lighter rectangle is perceived as even lighter than it is, and the shade of the darker rectangle is perceived as even darker. This makes it seem as if each rectange is coloured with a gradient in the opposite direction.

Value

A `ggplot2()` object (if `returnPlotOnly` is TRUE), or an object containing that `ggplot2()` object and intermediate products.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

See Also

confIntProp() and binom.test()

Examples

```
### V/S (no idea what this is: ?mtcars only mentions 'V/S' :-))
### and transmission (automatic vs manual)
ggProportionPlot(mtcars, items=c('vs', 'am'));

### Number of cylinders, by default comparing lowest value
### (4) to highest (8):
ggProportionPlot(mtcars, items=c('cyl'));

## Not run:
### Not running these to save time during package building/checking

### We can also compare 4 to 6:
ggProportionPlot(mtcars, items=c('cyl'),
                 hiCategory=6);

### Now compared to total records, instead of to
### highest value (hiCategory is ignored then)
ggProportionPlot(mtcars, items=c('cyl'),
                 compareHiToLo=FALSE);

### And for 6 cylinders:
ggProportionPlot(mtcars, items=c('cyl'),
                 loCategory=6, compareHiToLo=FALSE);

### And for 8 cylinders:
ggProportionPlot(mtcars, items=c('cyl'),
                 loCategory=8, compareHiToLo=FALSE);

### And for 8 cylinders with different labels
ggProportionPlot(mtcars, items=c('cyl'),
                 loCategory=8,
                 subQuestions='Cylinders',
                 leftAnchors="Eight",
                 rightAnchors="Four\nor\nsix",
                 compareHiToLo=FALSE);

### ... And showing the diamonds for the confidence intervals
ggProportionPlot(mtcars, items=c('cyl'),
                 loCategory=8,
                 subQuestions='Cylinders',
                 leftAnchors="Eight",
                 rightAnchors="Four\nor\nsix",
                 compareHiToLo=FALSE,
                 showDiamonds=TRUE);

## End(Not run)
```

```
### Using less steps for the confidence levels and changing
### the fill colours
ggProportionPlot(mtcars,
                 items=c('vs', 'am'),
                 showDiamonds = TRUE,
                 scale_fill = c("#B63679FF", "#FCFDBFFF"),
                 conf.steps=seq(from=0.0001, to=.9999, by=.2));
```

---

ggqq *Easy ggplot Q-Q plot*

Description

This function creates a qq-plot with a confidence interval.

Usage

```
ggqq(x, distribution = "norm", ..., ci = TRUE, line.estimate = NULL,
  conf.level = 0.95, sampleSizeOverride = NULL, observedOnX = TRUE,
  scaleExpected = TRUE, theoryLab = "Theoretical quantiles",
  observeLab = "Observed quantiles", theme = ggplot2::theme_bw())
```

Arguments

`x`	A vector containing the values to plot.
`distribution`	The distribution to (a 'd' and 'q' are prepended, and the resulting functions are used, e.g. `dnorm` and `qnorm` for the normal curve).
`...`	Any additional arguments are passed to the quantile function (e.g. `qnorm`). Because of these dots, any following arguments must be named explicitly.
`ci`	Whether to show the confidence interval.
`line.estimate`	Whether to show the line showing the match with the specified distribution (e.g. the normal distribution).
`conf.level`	THe confidence of the confidence leven arround the estimate for the specified distribtion.
`sampleSizeOverride`	It can be desirable to get the confidence intervals for a different sample size (when the sample size is very large, for example, such as when this plot is generated by the function `normalityAssessment`). That different sample size can be specified here.
`observedOnX`	Whether to plot the observed values (if `TRUE`) or the theoretically expected values (if `FALSE`) on the X axis. The other is plotted on the Y axis.
`scaleExpected`	Whether the scale the expected values to match the scale of the variable. This option is provided to be able to mimic SPSS' Q-Q plots.
`theoryLab`	The label for the theoretically expected values (on the Y axis by default).
`observeLab`	The label for the observed values (on the Y axis by default).
`theme`	The theme to use.

Details

This is strongly based on the answer by user Floo0 to a Stack Overflow question at Stack Exchange (see `http://stackoverflow.com/questions/4357031/qqnorm-and-qqline-in-ggplot2/27191036#27191036`), also posted at GitHub (see `https://gist.github.com/rentrop/d39a8406ad8af2a1066c`). That code is in turn based on the `car::qqPlot()` function from the `car` package.

Value

A `ggplot` plot is returned.

Author(s)

John Fox and Floo0; implemented in this package (and tweaked a bit) by Gjalt-Jorn Peters.

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

Examples

```
ggqq(mtcars$mpg);
```

`ggSave` *Save a ggplot with specific defaults*

Description

This function is vectorized over all argument except 'plot': so if you want to save multiple versions, simply provide vectors. Vectors of length 1 will be recycled using `rep()`; otherwise vectors have to all be the same length as `file`.

Usage

```
ggSave(file = NULL, plot = ggplot2::last_plot(), height = 8,
  width = 8, units = "in", dpi = 300, device = NULL, type = NULL,
  bg = "transparent", ...)
```

Arguments

`file`	The file where to save to.
`plot`	The plot to save; if omitted, the last drawn plot is saved.
`height, width`	The dimensions of the plot, specified in `units`.
`units`	The units, `'cm'`, 'mm', or `'in'`.
`dpi`	The resolution (dots per inch). This argument is vectorized.
`device`	The graphic device; is inferred from the file if not specified.
`type`	An additional arguments for the graphic device.
`bg`	The background (e.g. 'white').
`...`	Any additional arguments are passed on to `ggplot2::ggsave()`.

Value

The plot, invisibly.

Examples

```
plot <- ufs::ggBoxplot(mtcars, 'mpg');
ggSave(file=tempfile(fileext=".png"), plot=plot);
```

---

ifelseObj *Conditional returning of an object*

Description

The ifelseObj function just evaluates a condition, returning one object if it's true, and another if it's false.

Usage

```
ifelseObj(condition, ifTrue, ifFalse)
```

Arguments

condition	Condition to evaluate.
ifTrue	Object to return if the condition is true.
ifFalse	Object to return if the condition is false.

Value

One of the two objects

Examples

```
dat <- ifelseObj(sample(c(TRUE, FALSE), 1), mtcars, Orange);
```

`iqrOutlier` *Identify outliers according to the IQR criterion*

Description

The IQR criterion holds that any value lower than one-and-a-half times the interquartile range below the first quartile, or higher than one-and-a-half times the interquartile range above the third quartile, is an outlier. This function returns a logical vector that identifies those outliers.

Usage

```
iqrOutlier(x)
```

Arguments

`x`	The vector to scan for outliers.

Value

A logical vector where TRUE identifies outliers.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

See Also

`IQR`

Examples

```
### One outlier in the miles per gallon
iqrOutlier(mtcars$mpg);
```

is.nr — NULL *and* NA *'proof' checking of whether something is a number*

Description

Convenience function that returns TRUE if the argument is not null, not NA, and is.numeric.

Usage

```
is.nr(x)
```

Arguments

x	The value or vector to check.

Value

TRUE or FALSE.

Examples

```
is.nr(8);    ### Returns TRUE
is.nr(NULL); ### Returns FALSE
is.nr(NA);   ### Returns FALSE
```

is.odd — *Checking whether numbers are odd or even*

Description

Checking whether numbers are odd or even

Usage

```
is.odd(vector)

is.even(vector)
```

Arguments

vector	The vector to process

Value

A logical vector.

Examples

```
is.odd(4);
```

`isTrue` *More flexible version of isTRUE*

Description

Returns `TRUE` for `TRUE` elements, `FALSE` for `FALSE` elements, and whatever is specified in `na` for `NA` items.

Usage

```
isTrue(x, na = FALSE)
```

Arguments

`x`	The vector to check for `TRUE`, `FALSE`, and `NA` values.
`na`	What to return for `NA` values.

Value

A logical vector.

Examples

```
isTrue(c(TRUE, FALSE, NA));
isTrue(c(TRUE, FALSE, NA), na=TRUE);
```

`knitAndSave` *knitAndSave*

Description

knitAndSave

Usage

```
knitAndSave(plotToDraw, figCaption, file = NULL, path = NULL,
  figWidth = 8, figHeight = 8,
  catPlot = getOption("ufs.knitAndSave.catPlot", FALSE), ...)
```

Arguments

`plotToDraw`	The plot to knit using `knitFig()` and save using `ggSave()`.
`figCaption`	The caption of the plot (used as filename if no filename is specified).
`file, path`	The filename to use when saving the plot, or the path where to save the file if no filename is provided (if `path` is also omitted, `getWd()` is used).
`figWidth, figHeight`	The plot dimensions, by default specified in inches (but 'units' can be set which is then passed on to `ggSave()`.
`catPlot`	Whether to use `cat()` to print the knitr fragment.
`...`	Additional arguments are passed on to `ggSave()`. Note that file (and ...) are vectorized (see the `ggSave()` manual page).

Value

The `knitFig()` result, visibly.

Examples

```
plot <- ggBoxplot(mtcars, 'mpg');
knitAndSave(plot, figCaption="a boxplot", file=tempfile(fileext=".png"));
```

`knitFig` *Easily knit a custom figure fragment*

Description

THis function was written to make it easy to knit figures with different, or dynamically generated, widths and heights (and captions) in the same chunk when working with R Markdown.

Usage

```
knitFig(plotToDraw, template = getOption("ufs.knitFig.template", NULL),
  figWidth = getOption("ufs.knitFig.figWidth", 16/2.54),
  figHeight = getOption("ufs.knitFig.figHeight", 16/2.54),
  figCaption = "A plot.", chunkName = NULL, returnRaw = FALSE,
  catPlot = getOption("ufs.knitFig.catPlot", FALSE), ...)
```

Arguments

`plotToDraw`	The plot to draw, e.g. a `ggplot` plot.
`template`	A character value with the `knit_expand` template to use.
`figWidth`	The width to set for the figure (in inches).
`figHeight`	The height to set for the figure (in inches).
`figCaption`	The caption to set for the figure.

`chunkName`	Optionally, the name for the chunk. To avoid problems because multiple chunks have the name `"unnamed-chunk-1"`, if no chunk name is provided, `digest::digest()` is used to generate an MD5-hash from `Sys.time`.
`returnRaw`	Whether to `cat()` the result (`TRUE`) or whether to return it as `knitr::asis_output()` object (`FALSE`).
`catPlot`	Whether to use the `base::cat()` function to print the code for the plot, and return the result invisibly. If not, the result is returned visible, and so probably printed anyway.
`...`	Any additional arguments are passed on to `knit_expand`.

Value

This function returns nothing, but uses `knit_expand` and `knit` to `cat` the result.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

See Also

`knit_expand` and `knit`

Examples

```
knitFig(ggBoxplot(mtcars, 'mpg'))
```

`makeScales` *Title*

Description

Title

Usage

```
makeScales(data, scales, append = TRUE)
```

Arguments

`data`	The dataframe containing the variables (the items).
`scales`	A list of character vectors with the items in each scale, where each vectors' name is the name of the scale.
`append`	Whether to return the dataframe including the new variables (`TRUE`), or a dataframe with only those new variables (`FALSE`).

Value

Either a dataframe with the newly created variables, or the supplied dataframe with the newly created variables appended.

Examples

```
### First generate a list with the scales
scales <- list(scale1 = c('mpg', 'cyl'), scale2 = c('disp', 'hp'));

### Create the scales and add them to the dataframe
makeScales(mtcars, scales);
```

`massConvertToNumeric` *Converting many dataframe columns to numeric*

Description

This function makes it easy to convert many dataframe columns to numeric.

Usage

```
massConvertToNumeric(dat, byFactorLabel = FALSE,
  ignoreCharacter = TRUE, stringsAsFactors = FALSE)
```

Arguments

`dat`	The dataframe with the columns.
`byFactorLabel`	When converting factors, whether to do this by their label value (`TRUE`) or their level value (`FALSE`).
`ignoreCharacter`	Whether to convert (`FALSE`) or ignore (`TRUE`) character vectors.
`stringsAsFactors`	In the returned dataframe, whether to return string (character) vectors as factors or not.

Value

A data.frame.

Examples

```
### Create a dataset
a <- data.frame(var1 = factor(1:4),
                var2 = as.character(5:6),
                stringsAsFactors=FALSE);

### Ignores var2
b <- ufs::massConvertToNumeric(a);
```

```
### Converts var2
c <- ufs::massConvertToNumeric(a,
                               ignoreCharacter = FALSE);
```

`meanConfInt` *A confidence interval for the mean*

Description

A confidence interval for the mean

Usage

```
meanConfInt(vector = NULL, mean = NULL, sd = NULL, n = NULL,
  se = NULL, conf.level = 0.95)

## S3 method for class 'meanConfInt'
print(x, digits = 2, ...)
```

Arguments

`vector`	A vector with raw data points - either specify this or a mean and then either an sd and n or an se.
`mean`	A mean.
`sd, n`	A standard deviation and sample size; can be specified to compute the standard error.
`se`	The standard error (cna be specified instead of `sd` and `n`).
`conf.level`	The confidence level of the interval.
`x, digits, ...`	Respectively the object to print, the number of digits to round to, and any additonal arguments to pass on to the `print` function.

Value

And object with elements `input`, `intermediate`, and `output`, where `output` holds the result in list `ci`.

Examples

```
meanConfInt(mean=5, sd=2, n=20);
```

`meansDiamondPlot` *Diamond plots*

Description

This function generates a so-called diamond plot: a plot based on the forest plots that are commonplace in meta-analyses. The underlying idea is that point estimates are uninformative, and it would be better to focus on confidence intervals. The problem of the points with errorbars that are commonly employed is that the focus the audience's attention on the upper and lower bounds, even though those are the least relevant values. Using diamonds remedies this.

Usage

```
meansDiamondPlot(data, items = NULL, labels = NULL,
  decreasing = NULL, conf.level = 0.95, showData = TRUE,
  dataAlpha = 0.1, dataSize = 3, dataColor = "#444444",
  diamondColors = NULL, jitterWidth = 0.5, jitterHeight = 0.4,
  returnLayerOnly = FALSE, xlab = "Scores and means", ylab = NULL,
  theme = ggplot2::theme_bw(), xbreaks = "auto", outputFile = NULL,
  outputWidth = 10, outputHeight = 10, ggsaveParams = list(units =
  "cm", dpi = 300, type = "cairo"), dat = NULL, ...)
```

Arguments

`data, dat`	The dataframe containing the variables (`items`) to show in the diamond plot (the name `dat` for this argument is deprecated but still works for backward compatibility).
`items`	Optionally, the names (or numeric indices) of the variables (items) to show in the diamond plot. If NULL, all columns (variables, items) will be used.
`labels`	A character vector of labels to use instead of column names from the dataframe.
`decreasing`	Whether to sort the variables (rows) in the diamond plot decreasing (TRUE), increasing (FALSE), or not at all (NULL).
`conf.level`	The confidence of the confidence intervals.
`showData`	Whether to show the raw data or not.
`dataAlpha`	This determines the alpha (transparency) of the data points. Note that argument `alpha` can be used to set the alpha of the diamonds; this is eventually passed on to `ggDiamondLayer()`.
`dataSize`	The size of the data points.
`dataColor`	The color of the data points.
`diamondColors`	A vector of the same length as there are rows in the dataframe, to manually specify colors for the diamonds.
`jitterWidth`	How much to jitter the individual datapoints horizontally.
`jitterHeight`	How much to jitter the individual datapoints vertically.

`returnLayerOnly`	Set this to TRUE to only return the `ggplot()` layer of the diamondplot, which can be useful to include it in other plots.
`xlab, ylab`	The labels of the X and Y axes.
`theme`	The theme to use.
`xbreaks`	Where the breaks (major grid lines, ticks, and labels) on the x axis should be.
`outputFile`	A file to which to save the plot.
`outputWidth, outputHeight`	Width and height of saved plot (specified in centimeters by default, see `ggsaveParams`).
`ggsaveParams`	Parameters to pass to ggsave when saving the plot.
`...`	Additional arguments are passed to `diamondPlot()` and eventually to `ggDiamondLayer()`. This can be used to, for example, specify two or more colors to use to generate a gradient (using `generateColors` and maybe `fullColorRange`).

Value

A `ggplot()` plot with a `ggDiamondLayer()` is returned.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

See Also

`diamondPlot()`, `meanSDtoDiamondPlot()`, `ggDiamondLayer()`factorLoadingDiamondCIplot()

Examples

```
tmpDf <- data.frame(item1 = rnorm(50, 1.6, 1),
                    item2 = rnorm(50, 2.6, 2),
                    item3 = rnorm(50, 4.1, 3));

### A simple diamond plot
meansDiamondPlot(tmpDf);

### A diamond plot with manually
### specified labels and colors
meansDiamondPlot(tmpDf,
                 labels=c('First',
                          'Second',
                          'Third'),
                  diamondColors=c('blue', 'magenta', 'yellow'));

### Using a gradient for the colors
meansDiamondPlot(tmpDf,
                 labels=c('First',
                          'Second',
                          'Third'),
                 generateColors = c("magenta", "cyan"),
                 fullColorRange = c(1,5));
```

`meanSDtoDiamondPlot` *A diamond plot based on means, standard deviations, and sample sizes*

Description

This function generates a so-called diamond plot: a plot based on the forest plots that are commonplace in meta-analyses. The underlying idea is that point estimates are uninformative, and it would be better to focus on confidence intervals. The problem of the points with errorbars that are commonly employed is that the focus the audience's attention on the upper and lower bounds, even though those are the least relevant values. Using diamonds remedies this.

Usage

```
meanSDtoDiamondPlot(dat = NULL, means = 1, sds = 2, ns = 3,
  labels = NULL, colorCol = NULL, conf.level = 0.95,
  xlab = "Means", outputFile = NULL, outputWidth = 10,
  outputHeight = 10, ggsaveParams = list(units = "cm", dpi = 300, type
  = "cairo"), ...)
```

Arguments

`dat`	The dataset containing the means, standard deviations, sample sizes, and possible labels and manually specified colors.
`means`	Either the column in the dataframe containing the means, as numeric or as character index, or a vector of means.
`sds`	Either the column in the dataframe containing the standard deviations, as numeric or as character index, or a vector of standard deviations.
`ns`	Either the column in the dataframe containing the sample sizes, as numeric or as character index, or a vector of sample sizes.
`labels`	Optionally, either the column in the dataframe containing labels, as numeric or as character index, or a vector of labels.
`colorCol`	Optionally, either the column in the dataframe containing manually specified colours, as numeric or as character index, or a vector of manually specified colours.
`conf.level`	The confidence of the confidence intervals.
`xlab`	The label for the x axis.
`outputFile`	A file to which to save the plot.
`outputWidth, outputHeight`	Width and height of saved plot (specified in centimeters by default, see `ggsaveParams`).

`ggsaveParams`	Parameters to pass to ggsave when saving the plot.
`...`	Additional arguments are passed to `diamondPlot()` and eventually to `ggDiamondLayer()`. This can be used to, for example, specify two or more colors to use to generate a gradient (using `generateColors` and maybe `fullColorRange`).

Value

A `ggplot()` plot with a `ggDiamondLayer()` is returned.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

See Also

`meansDiamondPlot()`, `diamondPlot()`, `factorLoadingDiamondCIplot()`, `ggDiamondLayer()`

Examples

```
tmpDf <- data.frame(means = c(1, 2, 3),
                    sds = c(1.5, 3, 5),
                    ns = c(2, 4, 10),
                    labels = c('first', 'second', 'third'),
                    color = c('purple', 'grey', 'orange'));

### A simple diamond plot
meanSDtoDiamondPlot(tmpDf);

### A simple diamond plot with labels
meanSDtoDiamondPlot(tmpDf, labels=4);

### When specifying column names, specify column
### names for all columns
meanSDtoDiamondPlot(tmpDf, means='means',
                    sds='sds', ns='ns',
                    labels='labels');

### A diamond plot using the specified colours
meanSDtoDiamondPlot(tmpDf, labels=4, colorCol=5);

### A diamond plot using automatically generated colours
### using a gradient
meanSDtoDiamondPlot(tmpDf,
                    generateColors=c('green', 'red'));

### A diamond plot using automatically generated colours
### using a gradient, specifying the minimum and maximum
### possible values that can be attained
meanSDtoDiamondPlot(tmpDf,
                    generateColors=c('red', 'yellow', 'blue'),
                    fullColorRange=c(0, 5));
```

---

multiResponse — *Generate a table for multiple response questions*

Description

The `multiResponse` function mimics the behavior of the table produced by SPSS for multiple response questions.

Usage

```
multiResponse(data, items = NULL, regex = NULL, perlRegex = TRUE,
  endorsedOption = 1)
```

Arguments

`data`	Dataframe containing the variables to display.
`items, regex`	Arguments `items` and `regex` can be used to specify which variables to process. `items` should contain the variable (column) names (or indices), and `regex` should contain a regular expression used to match to the column names of the dataframe. If none is provided, all variables in the dataframe are processed.
`perlRegex`	Whether to use the perl engine to match the regex.
`endorsedOption`	Which value represents the endorsed option (note that producing this kind of table requires dichotomous items, where each variable is either endorsed or not endorsed, so this is also a way to treat other variables as dichotomour).

Value

A dataframe with columns `Option`, `Frequency`, `Percentage`, and `Percentage of (X) cases`, where X is the number of cases.

Author(s)

Ananda Mahto; implemented in this package (and tweaked a bit) by Gjalt-Jorn Peters.

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

References

This function is based on the excellent and extensive Stack Exchange answer by Ananda Mahto at https://stackoverflow.com/questions/9265003/analysis-of-multiple-response.

Examples

```
multiResponse(mtcars, c('vs', 'am'));
```

`multiVarFreq` — *Generate a table collapsing frequencies of multiple variables*

Description

This function can be used to efficiently combine the frequencies of variables with the same possible values. The frequencies are collapsed into a table with the variable names as row names and the possible values as column (variable) names.

Usage

```
multiVarFreq(data, items = NULL, labels = NULL, sortByMean = TRUE)
```

Arguments

`data`	The dataframe containing the variables.
`items`	The variable names.
`labels`	Labels can be provided which will be set as row names when provided.
`sortByMean`	Whether to sort the rows by mean value for each variable (only sensible if the possible values are numeric).

Value

The resulting dataframe, but with class 'multiVarFreq' prepended to allow pretty printing.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

See Also

`table()`

Examples

```
multiVarFreq(mtcars, c('gear', 'carb'));
```

normalHist *normalHist*

Description

normalHist generates a histogram with a density curve and a normal density curve.

Usage

```
normalHist(vector, histColor = "#0000CC",
  distributionColor = "#0000CC", normalColor = "#00CC00",
  distributionLineSize = 1, normalLineSize = 1, histAlpha = 0.25,
  xLabel = NULL, yLabel = NULL, normalCurve = TRUE,
  distCurve = TRUE, breaks = 30, theme = ggplot2::theme_minimal(),
  rug = NULL, jitteredRug = TRUE, rugSides = "b", rugAlpha = 0.2,
  returnPlotOnly = FALSE)

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

Arguments

`vector`	A numeric vector.
`histColor`	The colour to use for the histogram.
`distributionColor`	The colour to use for the density curve.
`normalColor`	The colour to use for the normal curve.
`distributionLineSize`	The line size to use for the distribution density curve.
`normalLineSize`	The line size to use for the normal curve.
`histAlpha`	Alpha value ('opaqueness', as in, versus transparency) of the histogram.
`xLabel`	Label to use on x axis.
`yLabel`	Label to use on y axis.
`normalCurve`	Whether to display the normal curve.
`distCurve`	Whether to display the curve showing the distribution of the observed data.
`breaks`	The number of breaks to use (this is equal to the number of bins minus one, or in other words, to the number of bars minus one).
`theme`	The theme to use.
`rug`	Whether to add a rug (i.e. lines at the bottom that correspond to individual datapoints.
`jitteredRug`	Whether to jitter the rug (useful for variables with several datapoints sharing the same value.

`rugSides`	This is useful when the histogram will be rotated; for example, this can be set to 'r' if the histogram is rotated 270 degrees.
`rugAlpha`	Alpha value to use for the rug. When there is a lot of overlap, this can help get an idea of the number of datapoints at 'popular' values.
`returnPlotOnly`	Whether to return the usual `normalHist` object that also contains all settings and intermediate objects, or whether to only return the `ggplot2::ggplot()` plot.
`x`	The object to print.
`...`	Any additional arguments are passed to the default `print` method.

Value

An object, with the following elements:

`input`	The input when the function was called.
`intermediate`	The intermediate numbers and distributions.
`dat`	The dataframe used to generate the plot.
`plot`	The histogram.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

Examples

```
normalHist(mtcars$mpg)
```

`noZero` *Remove one or more zeroes before the decimal point*

Description

Remove one or more zeroes before the decimal point

Usage

```
noZero(str)
```

Arguments

`str`	The character string to process.

Value

The processed string.

See Also

`formatCI()`, `formatR()`, `formatPvalue()`

Examples

```
noZero("0.3");
```

`pomegaSq` *The distribution of Omega Squared*

Description

These functions use some conversion to and from the *F* distribution to provide the Omega Squared distribution.

Usage

```
pomegaSq(q, df1, df2, populationOmegaSq = 0, lower.tail = TRUE)

qomegaSq(p, df1, df2, populationOmegaSq = 0, lower.tail = TRUE)

romegaSq(n, df1, df2, populationOmegaSq = 0)

domegaSq(x, df1, df2, populationOmegaSq = 0)
```

Arguments

`df1, df2`	Degrees of freedom for the numerator and the denominator, respectively.
`populationOmegaSq`	The value of Omega Squared in the population; this determines the center of the Omega Squared distribution. This has not been implemented yet in this version of `ufs`. If anybody has the inverse of `convert.ncf.to.omegasq()` for me, I'll happily integrate this.
`lower.tail`	logical; if TRUE (default), probabilities are the likelihood of finding an Omega Squared smaller than the specified value; otherwise, the likelihood of finding an Omega Squared larger than the specified value.
`p`	Vector of probabilites (*p*-values).
`n`	Desired number of Omega Squared values.
`x, q`	Vector of quantiles, or, in other words, the value(s) of Omega Squared.

Details

The functions use `convert.omegasq.to.f()` and `convert.f.to.omegasq()` to provide the Omega Squared distribution.

Value

domegaSq gives the density, pomegaSq gives the distribution function, qomegaSq gives the quantile function, and romegaSq generates random deviates.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

See Also

convert.omegasq.to.f(), convert.f.to.omegasq(), df(), pf(), qf(), rf()

Examples

```
### Generate 10 random Omega Squared values
romegaSq(10, 66, 3);

### Probability of findings an Omega Squared
### value smaller than .06 if it's 0 in the population
pomegaSq(.06, 66, 3);
```

---

pwr.bootES *Estimate required sample size for accuracy in parameter estimation using bootES*

Description

This function uses bootES::bootES() to compute

Usage

```
pwr.bootES(data = data, ci.type = "bca", ..., w = 0.1,
  silent = TRUE)
```

Arguments

data	The dataset, as you would normally supply to bootES::bootES(); you will probably have to simulate this.
ci.type	The estimation method; by default, the default of bootES::bootES() is used ('bca'), but this is changed to 'basic' if it encounters problems.
...	Other options for bootES::bootES() (see that help page).
w	The desired 'halfwidth' of the confidence interval.
silent	Whether to provide a lot of information about progress ('FALSE') or not ('TRUE').

Value

A single numeric value (the sample size).

References

Kirby, K. N., & Gerlanc, D. (2013). BootES: An R package for bootstrap confidence intervals on effect sizes. *Behavior Research Methods, 45*, 905–927. doi: 10.3758/s1342801303305

Examples

```
### To estimate a mean
x <- rnorm(500, mean=8, sd=3);
pwr.bootES(data.frame(x=x),
           R=500,
           w=.5);

### To estimate a correlation (the 'effect.type' parameter is
### redundant here; with two columns in the data frame, computing
### the confidence interval for the Pearson correlation is the default
### ehavior of bootES)
y <- x+rnorm(500, mean=0, sd=5);
cor(x, y);
requiredN <-
  pwr.bootES(data.frame(x=x,
                        y=y),
             effect.type='r',
             R=500,
             w=.2);
print(requiredN);
### Compare to parametric confidence interval
### based on the computed required sample size
confIntR(r = cor(x, y),
         N = requiredN);
### WIdth of obtained confidence interval
print(round(diff(as.numeric(confIntR(r = cor(x, y),
                            N = requiredN))), 2));
```

---

pwr.confIntProp   *Estimate required sample size for accuracy in parameter estimation of a proportion*

Description

This function uses `confIntProp()` to compute the required sample size for estimating a proportion with a given accuracy.

Usage

```
pwr.confIntProp(prop, conf.level = 0.95, w = 0.1, silent = TRUE)
```

Arguments

`prop`	The proportion you expect to find, or a vector of proportions to enable easy sensitivity analyses.
`conf.level`	The confidence level of the desired confidence interval.
`w`	The desired 'halfwidth' of the confidence interval.
`silent`	Whether to provide a lot of information about progress ('FALSE') or not ('TRUE').

Value

A single numeric value (the sample size).

Examples

```
### Required sample size to estimate a prevalence of .03 in the
### population with a confidence interval of a maximum half-width of .01
pwr.confIntProp(.03, w=.01);

### Vectorized over prop, so you can easily see how the required sample
### size varies as a function of the proportion
pwr.confIntProp(c(.03, .05, .10), w=.01);
```

`qVec` *Convenience function to quickly copy-paste a vector*

Description

Convenience function to quickly copy-paste a vector

Usage

```
qVec(x, fn = NULL)

qVecSum(x)
```

Arguments

`x`	A string with numbers, separated by arbitrary whitespace.
`fn`	An optional function to apply to the vecor before returning it.

Value

The numeric vector or result of calling the function

Examples

```
qVec('23  9  11  14  12  20');
```

repeatStr *Repeat a string a number of times*

Description

Repeat a string a number of times

Usage

```
repeatStr(n = 1, str = " ")
```

Arguments

`n, str`	Normally, respectively the frequency with which to repeat the string and the string to repeat; but the order of the inputs can be switched as well.

Value

A character vector of length 1.

Examples

```
### 10 spaces:
repStr(10);

### Three euro symbols:
repStr("\u20ac", 3);
```

report *Output report from results*

Description

This method can be used to format results in a way that can directly be included in a report or manuscript.

Usage

```
report(x, headingLevel = 3, quiet = TRUE, ...)

## Default S3 method:
report(x, headingLevel = 3, quiet = TRUE, ...)
```

Arguments

`x`	The object to show.
`headingLevel`	The level of the Markdown heading to provide; basically the number of hashes ('#') to prepend to the headings.
`quiet`	Passed on to `knitr::knit()` whether it should b chatty (`FALSE`) or quiet (`TRUE`).
`...`	Passed to the specific method; for the default method, this is passed to the print method.

`safeRequire` *Load a package, install if not available*

Description

Load a package, install if not available

Usage

```
safeRequire(packageName, mirrorIndex = NULL)
```

Arguments

`packageName`	The package
`mirrorIndex`	The index of the mirror (1 is used if not specified)

`scaleDiagnosis` *scaleDiagnosis*

Description

scaleDiagnosis provides a number of diagnostics for a scale (an aggregative measure consisting of several items).

Usage

```
scaleDiagnosis(data = NULL, items = NULL, plotSize = 180,
  sizeMultiplier = 1, axisLabels = "none",
  scaleReliability.ci = FALSE, conf.level = 0.95, normalHist = TRUE,
  digits = 3, headingLevel = 3, scaleName = NULL, ...)

## S3 method for class 'scaleDiagnosis'
print(x, digits = x$digits, ...)

## S3 method for class 'scaleDiagnosis'
knit_print(x, headingLevel = x$headingLevel,
  quiet = TRUE, echoPartial = FALSE, partialFile = NULL, ...)
```

Arguments

`data`	A dataframe containing the items in the scale. All variables in this dataframe will be used if items is NULL.
`items`	If not NULL, this should be a character vector with the names of the variables in the dataframe that represent items in the scale.
`plotSize`	Size of the final plot in millimeters.
`sizeMultiplier`	Allows more flexible control over the size of the plot elements
`axisLabels`	Passed to ggpairs function to set axisLabels.
`scaleReliability.ci`	TRUE or FALSE: whether to compute confidence intervals for Cronbach's Alpha and Omega (uses bootstrapping function in MBESS, takes a while).
`conf.level`	Confidence of confidence intervals for reliability estimates (if requested with scaleReliability.ci).
`normalHist`	Whether to use the default ggpairs histogram on the diagonal of the scattermatrix, or whether to use the `normalHist()` version.
`digits`	The number of digits to pass to the `print` method for the descriptives dataframe.
`headingLevel`	The level of the heading (number of hash characters to insert before the heading, to be rendered as headings of that level in Markdown).
`scaleName`	Optionally, a name for the scale to print as heading for the results.
`...`	Additional arguments for `scaleDiagnosis()` are passed on to `scatterMatrix()`, and additional arguments for the `print` method are passed to the default `print` method.
`x`	The object to print.
`quiet`	Whether to be chatty (`FALSE`) or quiet (`TRUE`).
`echoPartial`	Whether to show the code in the partial (`TRUE`) or hide it (`FALSE`).
`partialFile`	The file with the Rmd partial (if you want to overwrite the default).

Details

Function to generate an object with several useful statistics and a plot to assess how the elements (usually items) in a scale relate to each other, such as Cronbach's Alpha, omega, the Greatest Lower Bound, a factor analysis, and a correlation matrix.

Value

An object with the input and several output variables. Most notably:

`scaleReliability`	The results of scaleReliability.
`pca`	A Principal Components Analysis
`fa`	A Factor Analysis
`describe`	Decriptive statistics about the items
`scatterMatrix`	A scattermatrix with histograms on the diagonal and correlation coefficients in the upper right half.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

Examples

```
### Note: the 'not run' is simply because running takes a lot of time,
###       but these examples are all safe to run!
## Not run:
### This will prompt the user to select an SPSS file
scaleDiagnosis();

### Generate a datafile to use
exampleData <- data.frame(item1=rnorm(100));
exampleData$item2 <- exampleData$item1+rnorm(100);
exampleData$item3 <- exampleData$item1+rnorm(100);
exampleData$item4 <- exampleData$item2+rnorm(100);
exampleData$item5 <- exampleData$item2+rnorm(100);

### Use a selection of two variables
scaleDiagnosis(data=exampleData, items=c('item2', 'item4'));

### Use all items
scaleDiagnosis(data=exampleData);

## End(Not run)
```

`scaleStructure` *scaleStructure*

Description

The scaleStructure function (which was originally called scaleReliability) computes a number of measures to assess scale reliability and internal consistency. Note that to compute omega, the `MBESS` and/or the `psych` packages need to be installed, which are suggested packages and therefore should be installed separately (i.e. won't be installed automatically).

Usage

```
scaleStructure(data = NULL, items = "all", digits = 2, ci = TRUE,
  interval.type = "normal-theory", conf.level = 0.95, silent = FALSE,
  samples = 1000, bootstrapSeed = NULL, omega.psych = TRUE,
  poly = TRUE, suppressSuggestedPkgsMsg = FALSE, headingLevel = 3)

## S3 method for class 'scaleStructure'
print(x, digits = x$input$digits, ...)
```

```
## S3 method for class 'scaleStructure'
knit_print(x,
  headingLevel = x$input$headingLevel, quiet = TRUE,
  echoPartial = FALSE, partialFile = NULL, ...)
```

Arguments

`data`	A dataframe containing the items in the scale. All variables in this dataframe will be used if items = 'all'. If `dat` is `NULL`, a the `getData` function will be called to show the user a dialog to open a file.
`items`	If not 'all', this should be a character vector with the names of the variables in the dataframe that represent items in the scale.
`digits`	Number of digits to use in the presentation of the results.
`ci`	Whether to compute confidence intervals as well. This requires the suggested MBESS package, which has to be installed separately. If true, the method specified in `interval.type` is used. When specifying a bootstrapping method, this can take quite a while!
`interval.type`	Method to use when computing confidence intervals. The list of methods is explained in the help file for `ci.reliability` in MBESS. Note that when specifying a bootstrapping method, the method will be set to `normal-theory` for computing the confidence intervals for the ordinal estimates, because these are based on the polychoric correlation matrix, and raw data is required for bootstrapping.
`conf.level`	The confidence of the confidence intervals.
`silent`	If computing confidence intervals, the user is warned that it may take a while, unless `silent=TRUE`.
`samples`	The number of samples to compute for the bootstrapping of the confidence intervals.
`bootstrapSeed`	The seed to use for the bootstrapping - setting this seed makes it possible to replicate the exact same intervals, which is useful for publications.
`omega.psych`	Whether to also compute the interval estimate for omega using the `omega` function in the `psych` package. The default point estimate and confidence interval for omega are based on the procedure suggested by Dunn, Baguley & Brunsden (2013) using the `MBESS` function `ci.reliability` (because it has more options for computing confidence intervals, not always requiring bootstrapping), whereas the `psych` package point estimate was suggested in Revelle & Zinbarg (2008). The `psych` estimate usually (perhaps always) results in higher estimates for omega.
`poly`	Whether to compute ordinal measures (if the items have sufficiently few categories).
`suppressSuggestedPkgsMsg`	Whether to suppress the message about the suggested `MBESS` and `psych` packages.
`headingLevel`	The level of the Markdown heading to provide; basically the number of hashes ('#') to prepend to the headings.

`x`	The object to print
`...`	Any additional arguments for the default print function.
`quiet`	Passed on to `knitr::knit()` whether it should b chatty (`FALSE`) or quiet (`TRUE`).
`echoPartial`	Whether to show the executed code in the R Markdown partial (`TRUE`) or not (`FALSE`).
`partialFile`	This can be used to specify a custom partial file. The file will have object `x` available, which is the result of a call to `scaleStructure()`.

Details

If you use this function in an academic paper, please cite Peters (2014), where the function is introduced, and/or Crutzen & Peters (2015), where the function is discussed from a broader perspective.

This function is basically a wrapper for functions from the psych and MBESS packages that compute measures of reliability and internal consistency. For backwards compatibility, in addition to `scaleStructure`, `scaleReliability` can also be used to call this function.

Value

An object with the input and several output variables. Most notably:

`input`	Input specified when calling the function
`intermediate`	Intermediate values and objects computed to get to the final results
`output`	Values of reliability / internal consistency measures, with as most notable elements:
`output$dat`	A dataframe with the most important outcomes
`output$omega`	Point estimate for omega
`output$glb`	Point estimate for the Greatest Lower Bound
`output$alpha`	Point estimate for Cronbach's alpha
`output$coefficientH`	Coefficient H
`output$omega.ci`	Confidence interval for omega
`output$alpha.ci`	Confidence interval for Cronbach's alpha

Author(s)

Gjalt-Jorn Peters and Daniel McNeish (University of North Carolina, Chapel Hill, US).

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

References

Crutzen, R., & Peters, G.-J. Y. (2015). Scale quality: alpha is an inadequate estimate and factor-analytic evidence is needed first of all. *Health Psychology Review.* http://dx.doi.org/10.1080/17437199.2015.1124240

Dunn, T. J., Baguley, T., & Brunsden, V. (2014). From alpha to omega: A practical solution to the pervasive problem of internal consistency estimation. *British Journal of Psychology*, 105(3), 399-412. doi:10.1111/bjop.12046

Eisinga, R., Grotenhuis, M. Te, & Pelzer, B. (2013). The reliability of a two-item scale: Pearson, Cronbach, or Spearman-Brown? *International Journal of Public Health*, 58(4), 637-42. doi:10.1007/s00038-012-0416-3

Gadermann, A. M., Guhn, M., Zumbo, B. D., & Columbia, B. (2012). Estimating ordinal reliability for Likert-type and ordinal item response data: A conceptual, empirical, and practical guide. *Practical Assessment, Research & Evaluation*, 17(3), 1-12.

Peters, G.-J. Y. (2014). The alpha and the omega of scale reliability and validity: why and how to abandon Cronbach's alpha and the route towards more comprehensive assessment of scale quality. *European Health Psychologist*, 16(2), 56-69. http://ehps.net/ehp/index.php/contents/article/download/ehp.v16.i2.p56/1

Revelle, W., & Zinbarg, R. E. (2009). Coefficients Alpha, Beta, Omega, and the glb: Comments on Sijtsma. *Psychometrika*, 74(1), 145-154. doi:10.1007/s11336-008-9102-z

Sijtsma, K. (2009). On the Use, the Misuse, and the Very Limited Usefulness of Cronbach's Alpha. *Psychometrika*, 74(1), 107-120. doi:10.1007/s11336-008-9101-0

Zinbarg, R. E., Revelle, W., Yovel, I., & Li, W. (2005). Cronbach's alpha, Revelle's beta and McDonald's omega H: Their relations with each other and two alternative conceptualizations of reliability. *Psychometrika*, 70(1), 123-133. doi:10.1007/s11336-003-0974-7

See Also

`psych::omega()`, `psych::alpha()`, and `MBESS::ci.reliability()`.

Examples

```
## Not run:
### (These examples take a lot of time, so they are not run
###  during testing.)

### This will prompt the user to select an SPSS file
scaleStructure();

### Load data from simulated dataset testRetestSimData (which
### satisfies essential tau-equivalence).
data(testRetestSimData);

### Select some items in the first measurement
exampleData <- testRetestSimData[2:6];

### Use all items (don't order confidence intervals to save time
### during automated testing of the example)
```

```
scaleStructure(dat=exampleData, ci=FALSE);

### Use a selection of three variables (without confidence
### intervals to save time
scaleStructure(dat=exampleData, items=c('t0_item2', 't0_item3', 't0_item4'),
               ci=FALSE);

### Make the items resemble an ordered categorical (ordinal) scale
ordinalExampleData <- data.frame(apply(exampleData, 2, cut,
                                       breaks=5, ordered_result=TRUE,
                                       labels=as.character(1:5)));

### Now we also get estimates assuming the ordinal measurement level
scaleStructure(ordinalExampleData, ci=FALSE);

## End(Not run)
```

---

scatterMatrix *scatterMatrix*

Description

scatterMatrix produced a matrix with jittered scatterplots, histograms, and correlation coefficients.

Usage

```
scatterMatrix(dat, items = NULL, plotSize = 180, sizeMultiplier = 1,
  axisLabels = "none", normalHist = TRUE, progress = NULL,
  theme = ggplot2::theme_minimal(), hideGrid = TRUE, ...)

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

Arguments

`dat`	A dataframe containing the items in the scale. All variables in this dataframe will be used if items is NULL.
`items`	If not NULL, this should be a character vector with the names of the variables in the dataframe that represent items in the scale.
`plotSize`	Size of the final plot in millimeters.
`sizeMultiplier`	Allows more flexible control over the size of the plot elements
`axisLabels`	Passed to ggpairs function to set axisLabels.
`normalHist`	Whether to use the default ggpairs histogram on the diagonal of the scattermatrix, or whether to use the `normalHist()` version.

`progress`	Whether to show a progress bar; set to `FALSE` to disable. See `GGally::ggpairs()` help for more information.
`theme`	The ggplot2 theme to use.
`hideGrid`	Whether to hide the gridlines in the plot.
`...`	Additional arguments for `scatterMatrix()` are passed on to `normalHist()`, and additional arguments for the `print` method are passed on to the default `print` method.
`x`	The object to print.

Value

An object with the input and several output variables. Most notably:

`output$scatterMatrix`
A scattermatrix with histograms on the diagonal and correlation coefficients in the upper right half.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

Examples

```
### Note: the 'not run' is simply because running takes a lot of time,
###       but these examples are all safe to run!
## Not run:

### Generate a datafile to use
exampleData <- data.frame(item1=rnorm(100));
exampleData$item2 <- exampleData$item1+rnorm(100);
exampleData$item3 <- exampleData$item1+rnorm(100);
exampleData$item4 <- exampleData$item2+rnorm(100);
exampleData$item5 <- exampleData$item2+rnorm(100);

### Use all items
scatterMatrix(dat=exampleData);

## End(Not run)
```

`setFigCapNumbering` *Set caption numbering*

Description

Set caption numbering

Usage

```
setFigCapNumbering(captionName = "fig.cap", prefix = "Figure %s: ",
  suffix = "", optionName = paste0("setCaptionNumbering_",
  captionName), resetCounterTo = 1)
```

Arguments

`captionName`	THe name of the caption; normally `fig.cap` or `tab.cap`.
`prefix, suffix`	The prefix and suffix; any occurrences of `\%s` will be replaced by the number.
`optionName`	THe name to use for the option that keeps track of the numbering.
`resetCounterTo`	Whether to reset the counter (as stored in the options), and if so, to what value (set to `FALSE` to prevent resetting).

Value

`NULL`, invisibly.

Examples

```
setFigCapNumbering();

### For table captions
setFigCapNumbering("tab.cap", "Table %s: ");
```

`sharedSubString` *sharedSubString*

Description

A function to find the longest shared substring in a character vector.

Usage

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

Arguments

`x`	The character vector to process.
`y`	Optionally, two single values can be specified. This is probably not useful to end users, but it's used by the function when it calls itself.

Value

A vector of length one with either the longest substring that occurs in all values of the character vector, or NA if no overlap an be found.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

Examples

```
sharedSubString(c("t0_responseTime", "t1_responseTime", "t2_responseTime"));
### Returns "_responseTime"
```

`spearmanBrown` *Spearman-Brown formula*

Description

Spearman-Brown formula

Usage

```
spearmanBrown(nrOfItems, itemReliability)

spearmanBrown_reversed(nrOfItems, scaleReliability)

spearmanBrown_requiredLength(scaleReliability, itemReliability)
```

Arguments

`nrOfItems`	Number of items (or 'subtests') in the scale (or 'test').
`itemReliability`	The reliability of one item (or 'subtest').
`scaleReliability`	The reliability of the scale (or, desired reliability of the scale).

Value

For `spearmanBrown`, the predicted scale reliability; for `spearmanBrown_requiredLength`, the number of items required to achieve the desired scale reliability; and for `spearmanBrown_reversed`, the reliability of one item.

Examples

```
spearmanBrown(10, .4);
spearmanBrown_reversed(10, .87);
spearmanBrown_requiredLength(.87, .4);
```

---

strToFilename   *Convert a string to a safe filename*

Description

Convert a string to a safe filename

Usage

```
strToFilename(str, ext = NULL)
```

Arguments

`str`	The string to convert.
`ext`	Optionally, an extension to append.

Value

The string, processed to remove potentially problematic characters.

Examples

```
strToFilename("this contains: illegal characters, spaces, et cetera.");
```

`testRetestSimData` *testRetestSimData is a simulated dataframe used to demonstrate the testRetestAlpha coefficient function.*

Description

This dataset contains the true scores of 250 participants on some variable, and 10 items of a scale administered twice (at t0 and at t1).

Format

A data frame with 250 observations on the following 21 variables.

trueScore The true scores

t0_item1 Score on item 1 at test

t0_item2 Score on item 2 at test

t0_item3 Score on item 3 at test

t0_item4 Score on item 4 at test

t0_item5 Score on item 5 at test

t0_item6 Score on item 6 at test

t0_item7 Score on item 7 at test

t0_item8 Score on item 8 at test

t0_item9 Score on item 9 at test

t0_item10 Score on item 10 at test

t1_item1 Score on item 1 at retest

t1_item2 Score on item 2 at retest

t1_item3 Score on item 3 at retest

t1_item4 Score on item 4 at retest

t1_item5 Score on item 5 at retest

t1_item6 Score on item 6 at retest

t1_item7 Score on item 7 at retest

t1_item8 Score on item 8 at retest

t1_item9 Score on item 9 at retest

t1_item10 Score on item 10 at retest

Details

This dataset was generated with the code in the reliabilityTest.r test script.

Author(s)

Gjalt-Jorn Peters

Maintainer: Gjalt-Jorn Peters gjalt-jorn@userfriendlyscience.com

Examples

```
data(testRetestSimData);
head(testRetestSimData);
hist(testRetestSimData$t0_item1);
cor(testRetestSimData);
```

`vecTxt` *Easily parse a vector into a character value*

Description

Easily parse a vector into a character value

Usage

```
vecTxt(vector, delimiter = ", ", useQuote = "",
  firstDelimiter = NULL, lastDelimiter = " & ", firstElements = 0,
  lastElements = 1, lastHasPrecedence = TRUE)

vecTxtQ(vector, useQuote = "'", ...)
```

Arguments

`vector`	The vector to process.
`delimiter`, `firstDelimiter`, `lastDelimiter`	The delimiters to use for respectively the middle, first `firstElements`, and last `lastElements` elements.
`useQuote`	This character string is pre- and appended to all elements; so use this to quote all elements (`useQuote="'"`), doublequote all elements (`useQuote='"'`), or anything else (e.g. `useQuote='|'`). The only difference between `vecTxt` and `vecTxtQ` is that the latter by default quotes the elements.
`firstElements`, `lastElements`	The number of elements for which to use the first respective last delimiters
`lastHasPrecedence`	If the vector is very short, it's possible that the sum of firstElements and lastElements is larger than the vector length. In that case, downwardly adjust the number of elements to separate with the first delimiter (`TRUE`) or the number of elements to separate with the last delimiter (`FALSE`)?
`...`	Any addition arguments to `vecTxtQ` are passed on to `vecTxt`.

Value

A character vector of length 1.

Examples

```
vecTxtQ(names(mtcars));
```

%IN% *Case insensitive version of %in%*

Description

Case insensitive version of %in%

Usage

```
find %IN% table
```

Arguments

`find`	The element(s) to look up in the vector or matrix.
`table`	The vector or matrix in which to look up the element(s).

Value

A logical vector.

Examples

```
letters[1:4] %IN% LETTERS
```

Index