Package ‘userfriendlyscience’

September 24, 2018

Type Package

Title Quantitative Analysis Made Accessible

Version 0.7.2

Date 2018-09-25

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

License GPL (>= 3)

Description Contains a number of functions that serve two goals. First, to make R more accessible to people migrating from SPSS by adding a number of functions that behave roughly like their SPSS equivalents (also see <https://rosettastats.com>). Second, to make a number of slightly more advanced functions more user friendly to relatively novice users. The package also conveniently houses a number of additional functions that are intended to increase the quality of methodology and statistics in psychology, not by offering technical solutions, but by shifting perspectives, for example towards reasoning based on sampling distributions as opposed to on point estimates.

URL `http://userfriendlyscience.com`

BugReports `https://github.com/matherion/userfriendlyscience/issues`

LazyData true

Imports BiasedUrn, car, data.tree, DiagrammeR, diptest, digest, GGally, ggplot2, ggrepel, ggridges, gridExtra, GPArotation, gtable, knitr, lavaan, lme4, MASS, MBESS, minpack.lm, pander, plyr, psych, pwr, RColorBrewer, rio, scales, SCRT, SuppDists, ufs (>= 0.0.1), viridis, XML, xtable

Suggests multcompView

NeedsCompilation no

Author Gjalt-Jorn Peters [aut, cre, ctb],
Peter Verboon [ctb],
James Green [ctb]

Repository CRAN

Date/Publication 2018-09-24 18:30:03 UTC

R topics documented:

`userfriendlyscience-package` *Userfriendlyscience (UFS)*

Description

This package contains a number of functions that serve two goals. First, to make R more accessible to people migrating from SPSS by adding a number of functions that behave roughly like their SPSS equivalents (also see `http://rosettastats.com`). Second, to make a number of slightly more advanced functions more user friendly to relatively novice users. The package also conveniently houses a number of additional functions that are intended to increase the quality of methodology and statistics in psychology, not by offering technical solutions, but by shifting perspectives, for example towards reasoning based on sampling distributions as opposed to on point estimates.

Details

Package:	userfriendlyscience
Type:	Package
Version:	0.7-1
Date:	2018-05-01
License:	GPL (>= 3)

Userfriendlyscience (UFS) contains a number of functions that serve two goals. First, to make R more accessible to people migrating from SPSS by adding a number of functions that behave roughly like their SPSS equivalents (also see `http://rosettastats.com` for a tool that helps with this). Second, to make a number of slightly more advanced functions more user friendly to relatively novice users. The package also conveniently houses a number of additional functions that are intended to increase the quality of methodology and statistics in psychology, not by offering technical solutions, but by shifting perspectives, for example towards reasoning based on sampling distributions as opposed to on point estimates.

The package imports functions from many other packages, which is in line with its function as a 'wrapper package': UFS aims to make many existing functions easier for users coming from SPSS, so sometimes a function is added when it saves the user just some data preparing.

The package implements many solutions provided by people all over the world, most from Stack Exchange (both from Cross Validated and Stack Overflow). I credit these authors in the help pages of those functions and in the Author(s) section of this page. If you wrote a function included here, and you want me to take it out, feel free to contact me of course (also, see `http://meta.stackoverflow.com/questions/319171/i-would-like-to-use-a-function-written-by-a-stack-overflow-memb`

Author(s)

Author: Gjalt-Jorn Peters (Open University of the Netherlands, Greater Good, and Maastricht University).

Contributors: Peter Verboon (`convert.omegasq.to.cohensf`, `genlog`, and `piecewiseRegr`, Open University of the Netherlands), Amy Chan (`ggPie`), Jeff Baggett (`posthocTGH`, University of Wisconsin - La Crosse), Daniel McNeish (`scaleStructure`, University of North Carolina), Nick Sabbe (`curfnfinder`, Arteveldehogeschool), Douglas Bonett (`confIntR`, `pwr.confIntR`, UC Santa Cruz, United States), Murray Moinester (`confIntR`, `pwr.confIntR`, Tel Aviv University, Israel), Stefan Gruijters (`nnc`, `ggNNC`, `convert.d.to.eer`, `convert.d.to.nnc`, `erDataSeq`, Maastricht University), Ron Pat-El (`logRegr`, Open University of the Netherlands), Ananda Mahto (`multiResponse`).

A number of functions in this package use code fragments that were used without explicit communicating with the author (because I've been unable to find contact details of the authors, or because I haven't gotten around to contacting them yet). The authors of these fragments are John Fox (`car` code in `ggqq`), Floo0 (`ggqq`), Jason Aizkalns (`ggBoxplot`), Luke Tierney (in `pwr.cohensdCI`, its alias `pwr.confIntd`, and `cohensdCI`).

In addition, the function `escapeRegEx` from package `Hmisc` is included and used internally to avoid importing that entire package just for that function. This function was written by Charles Dupont (Department of Biostatistics, Vanderbilt University). The help page was also taken from that package. The `ad.test` function from package `nortest` was included for the same reason; this was written by Juergen Gross. These functions are not exported.

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

References

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.

Peters, G.-J. Y. (2018). Diamond Plots: a tutorial to introduce a visualisation tool that facilitates interpretation and comparison of multiple sample estimates while respecting their inaccuracy. *PsyArXiv; under review at Health Psychology Bulletin*. Preprint doi: 10.17605/osf.io/9w8yv

Peters, G.-J. Y. & Crutzen, R. (2018). 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. *PsyArXiv; under review at Psychology & Health*. Preprint doi: 10.17605/osf.io/cjsk2

Crutzen, R., Peters, G.-J. Y., & Noijen, J. (2018). Using Confidence Interval-Based Estimation of Relevance to Select Social-Cognitive Determinants for Behavior Change Interventions. Frontiers in Public Health 5:165. http://dx.doi.org/10.3389/fpubh.2017.00165

Crutzen, R. (2014). Time is a jailer: what do alpha and its alternatives tell us about reliability? *The European Health Psychologist*, 1(2), 70-74.

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*. doi: 10.1080/17437199.2015.1124240

Verboon, P. & Peters, G.-J. Y. (2018). Applying the generalized logistic model in single case designs. *PsyArXiv*. Preprint doi: 10.17605/osf.io/ad5eh

See Also

`psych` and `MBESS` contain many useful functions for researchers in psychology.

Examples

```
### Create simple dataset
dat <- PlantGrowth[1:20,];
### Remove third level from group factor
dat$group <- factor(dat$group);

### Examine normality
normalityAssessment(dat$weight);

### Compute mean difference and show it
meanDiff(dat$weight ~ dat$group, plot=TRUE);

### Show the t-test
didacticPlot(meanDiff(dat$weight ~ dat$group)$t,
             statistic='t',
             df1=meanDiff(dat$weight ~ dat$group)$df);

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

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

## Not run:
### Show reliabilities
scaleStructure(dat=exampleData, ci=FALSE,
               omega.psych=FALSE, poly=FALSE);

## End(Not run)
```

```
### Show item distributions and means
meansDiamondPlot(exampleData);

### Create a dichotomous variable
exampleData$group <- cut(exampleData$t0_item2, 2);

### Show a dlvPlot
dlvPlot(exampleData, x="group", y="t0_item1");

### show a dlvPlot with less participants, showing the confidence
### interval and standard error bars better
dlvPlot(exampleData[1:30, ], x="group", y="t0_item1");
```

`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

```
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 = .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 = 'test')
```

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 `meanDiff` for more information.

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');
```

`associationMatrix Helper Functions`

associationMatrix Helper Functions

Description

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

Usage

```
computeStatistic_t(var1, var2, conf.level=.95, var.equal='test', ...)
computeStatistic_r(var1, var2, conf.level=.95, ...)
computeStatistic_f(var1, var2, conf.level=.95, ...)
computeStatistic_chisq(var1, var2, conf.level=.95, ...)

computeEffectSize_d(var1, var2, conf.level=.95, var.equal='test', ...)
computeEffectSize_r(var1, var2, conf.level=.95, ...)
computeEffectSize_etasq(var1, var2, conf.level=.95, ...)
computeEffectSize_omegasq(var1, var2, conf.level=.95, ...)
```

```
computeEffectSize_v(var1, var2, conf.level=.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
`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).
`var.equal`	Whether to test for equal variances ('test'), assume equality ('yes'), or assume unequality ('no'). See `meanDiff` for more information.
`...`	Any additonal arguments are sometimes used to specify exactly how statistics and effect sizes should be computed.

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

`meanDiff`, `associationMatrix`

Examples

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

`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.

associationsToDiamondPlotDf is a helper function that produces the required dataframe.

Usage

```
associationsDiamondPlot(dat, covariates, criteria,
                        labels = NULL,
                        criteriaLabels = NULL,
                        decreasing=NULL,
                        sortBy=NULL,
                        conf.level=.95,
                        criteriaColors = brewer.pal(8, 'Set1'),
                        criterionColor = 'black',
                        returnLayerOnly = FALSE,
                        esMetric = 'r',
                        multiAlpha=.33,
                        singleAlpha = 1,
                        showLegend=TRUE,
                        xlab="Effect size estimates",
                        ylab="",
                        theme=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

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`, `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));
```

`asymmetricalScatterMatrix` *asymmetricalScatterMatrix*

Description

This function generates an asymmetrical `scatterMatrix` with histograms showing the distribution of each variable.

Usage

```
asymmetricalScatterMatrix(dat, cols, rows, theme = dlvTheme(),
                          autoSize = TRUE,
                          txtHeight = 1, histHeight = 3,
                          scatterWidth = 6, scatterHeight = 6, unit = "cm",
                          dpi = 200, showCorrelations=c('top-left',
                          'top-right', 'bottom-left', 'bottom-right'),
                          correlationSize = 15,
                          correlationColor = "#cadded",
                          pointSize = 1.5)
```

Arguments

`dat`	The dataframe containing the items to show in the `scatterMatrix`.
`cols`	The variable names of the variables to place on the columns.
`rows`	The variable names of the variables to place on the rows.
`theme`	Which ggplot theme to use.
`autoSize`	Whether to resize the plot depending on the viewport (i.e. device that is being drawn to) or whether to use the four measurements specified below (`txtHeight`, `histHeight`, `scatterWidth`, and `scatterHeight`) to size the plot.
`txtHeight, histHeight, scatterWidth, scatterHeight`	These numbers are used to determine the space used for displaying the scatterplots, histograms, and labels in the final scatterMatrix.
`unit`	The unit in which txtHeight, histHeight, scatterWidth, and scatterheight are provided.
`dpi`	The DPI of the final plot.
`showCorrelations`	Where to display correlation coefficients; set to NULL to display no correlation coefficients.
`correlationSize`	The size(s) of the correlation coefficient(s).
`correlationColor`	The color of the correlation coefficient(s).
`pointSize`	The size of the points in the scatterplots.

Value

A `scatterMatrix`, just not symmetrical.

Author(s)

Gjalt-Jorn Peters

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

Examples

```
asymmetricalScatterMatrix(infert, cols=c("parity", "age"),
                          rows=c("induced", "case", "spontaneous"),
                          showCorrelations="top-right");
```

`averageFishersZs` *averageFishersZs*

Description

Takes pairs of Fisher's z's and the accompanying n's (sample sizes) and returns their average.

Usage

```
averageFishersZs(zs, ns)
```

Arguments

`zs`	The values of Fisher's z.
`ns`	The sample sizes (ns).

Value

The average of the Fisher's z values.

Author(s)

Gjalt-Jorn Peters

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

See Also

`averagePearsonRs`

Examples

```
averageFishersZs(c(1.1, 5.4), c(10, 30));
```

`averagePearsonRs` *averagePearsonRs*

Description

Takes pairs of Pearson r's (correlation coefficients) and the accompanying n's (sample sizes) and returns their average.

Usage

```
averagePearsonRs(rs, ns, FishersZ = TRUE)
```

Arguments

`rs`	The correlation coefficients.
`ns`	The sample sizes.
`FishersZ`	Whether to compute the average through Fisher's z (only method implemented as of the writing of this document).

Author(s)

Gjalt-Jorn Peters

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

See Also

`averageFishersZs`, `convert.r.to.fisherz`

Examples

```
averagePearsonRs(c(.3, .4, .6), c(70, 80, 50));
```

`basicSPSStranslationFunctions` *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)

exportToSPSS(dat, savfile = NULL, datafile = NULL, codefile = NULL,
             fileEncoding = "UTF-8", newLinesInString = " |n| ")

filterBy(dat, expression, replaceOriginalDataframe = TRUE,
         envir = parent.frame())

useAll(dat, replaceFilteredDataframe = TRUE)

mediaan(vector)

modus(vector)
```

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.
`dat`	Dataframe to process: for filterBy, dataframe to filter rows from; for useAll, dataframe to restore ('unfilter').
`datafile`	The name of the data file, a comma separated values file that can be read into SPSS by using the code file.

`codefile`	The name of the code file, the SPSS syntax file that can be used to import the data file.
`savfile`	The name of the SPSS format .sav file (alternative for writing a datafile and a codefile).
`fileEncoding`	The encoding to use to write the files.
`newLinesInString`	A string to replace newlines with (SPSS has problems reading newlines).
`expression`	Logical expression determining which rows to keep and which to drop. Can be either a logical vector or a string which is then evaluated. If it's a string, it's evaluated using 'with' to evaluate the expression using the variable names.
`replaceOriginalDataframe`	Whether to also replace the original dataframe in the parent environment. Very messy, but for maximum compatibility with the 'SPSS way of doing things', by default, this is true. After all, people who care about the messiness/inappropriateness of this function wouldn't be using it in the first place :-)
`envir`	The environment where to create the 'backup' of the unfiltered dataframe, for when useAll is called and the filter is deactivated again.
`replaceFilteredDataframe`	Whether to replace the filtered dataframe passed in the 'dat' argument (see replaceOriginalDataframe).
`vector`	For mediaan and modus, the vector for which to find the median or mode.

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 Omega-Hat 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)

### Get a median and a mode
mediaan(c(1,2,2,3,4,4,5,6,6,6,7));
modus(c(1,2,2,3,4,4,5,6,6,6,7));

### Create an example dataframe
(exampleDat <- data.frame(x=rep(8, 8), y=rep(c(0,1), each=4)));
### Filter it, replacing the original dataframe
(filterBy(exampleDat, "y=0"));
### Restore the old dataframe
(useAll(exampleDat));
```

`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 `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 = 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` 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

`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);
```

`checkDataIntegrity` *Conveniently checking data integrity*

Description

This function is designed to make it easy to perform some data integrity checks, specifically checking for values that are impossible or unrealistic. These values can then be replaced by another value, or the offending cases can be deleted from the dataframe.

Usage

```
checkDataIntegrity(x,
                   dat,
                   newValue = NA,
                   removeCases = FALSE,
                   validValueSuffix = "_validValue",
                   newValueSuffix = "_newValue",
                   totalVarName = "numberOfInvalidValues",
                   append = TRUE,
                   replace = TRUE,
                   silent = FALSE,
                   rmarkdownOutput=FALSE,
                   callingSelf = FALSE)
```

Arguments

`x`	This can be either a vector or a list. If it is a vector, it should have two elements, the first one being a regular expression matching one or more variables in the dataframe specified in `dat`, and second one being the condition the matching variables have to satisfy. If it is a list, it should be a list of such vectors. The conditions should start with a `Comparison` operator followed by a value (e.g. "<30" or ">=0).
`dat`	The dataframe containing the variables of which we should check the integrity.
`newValue`	The new value to be assigned to cases not satisfying the specified conditions.
`removeCases`	Whether to delete cases that do not satisfy the criterion from the dataframe (if `FALSE`, they're not deleted, but the offending value is replaced by `newValue`).
`validValueSuffix`	Suffix to append to variable names when creating variable names for new variables that contain TRUE and FALSE to specify for each original variable whether its value satisfied the specified criterion.
`newValueSuffix`	If `replace` is `FALSE`, original values are not replaced, but instead new variables are created where the offending values have been replaced. This suffix is appended to each original variable name to create the new variable name.
`totalVarName`	This is the name of a variable that contains, for each case, the total number of invalid values among all variables checked.
`append`	Whether to append the columns to the dataframe, or only return the new columns.
`replace`	Whether to replace the offending values with the value specified in `newValue` or whether to create new columns (see `newValueSuffix`).
`silent`	Whether to display the log, or only set it as attribute of the returned dataframe.
`rmarkdownOutput`	Whether to format the log so that it's ready to be included in RMarkdown reports.
`callingSelf`	For internal use; whether the function calls itself.

Value

The dataframe with the corrections, and the log stored in attribute `checkDataIntegrity_log`.

Author(s)

Gjalt-Jorn Peters

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

Examples

```
### Default behavior: return dataframe with
### offending values replaced by NA

checkDataIntegrity(c('mpg', '<30'),
                   mtcars);

### Check two conditions, and instead of returning the
### dataframe with the results appended, only return the
### columns indicating which cases 'pass', what the new
### values would be, and how many invalid values were
### found for each case (to easily remove cases that
### provided many invalid values)

checkDataIntegrity(list(c('mpg', '<30'),
                        c('gear', '<5')),
                   mtcars,
                   append=FALSE);
```

`CIBER` *Confidence Interval-Based Estimation of Relevance (CIBER)*

Description

This function generates a high-level plot consisting of several diamond plots. This function is useful for estimating the relative relevance of a set of determinants of, for example, behavior. The plot in the left hand panel shows each determinant's distribution with a diamond representing the confidence interval. The right hand plot shows the determinants' associations to one or more 'target' variables, such as behavior or determinants of behavior.

Usage

```
CIBER(data, determinants, targets, conf.level = list(means = 0.9999,
  associations = 0.95), subQuestions = NULL, leftAnchors = rep("Lo",
  length(determinants)), rightAnchors = rep("Hi", length(determinants)),
  orderBy = NULL, decreasing = NULL, numberSubQuestions = FALSE,
  generateColors = list(means = c("red", "blue", "green"), associations =
  c("red", "grey", "green")),
  strokeColors = viridis::viridis(length(targets)),
```

```
  titlePrefix = "Means and associations with", titleVarLabels = NULL,
  titleSuffix = "", fullColorRange = NULL, associationsAlpha = 0.5,
  returnPlotOnly = TRUE, drawPlot = TRUE, baseSize = 0.8, dotSize = 2.5
  * baseSize, baseFontSize = 10 * baseSize,
  theme = ggplot2::theme_bw(base_size = baseFontSize), xbreaks = NULL, ...)

detStructCIBER(determinantStructure, data, conf.level = list(means = 0.9999,
  associations = 0.95), subQuestions = NULL, leftAnchors = rep("Lo",
  length(determinants)), rightAnchors = rep("Hi", length(determinants)),
  orderBy = 1, decreasing = NULL, generateColors = list(means = c("red",
  "blue", "green"), associations = c("red", "grey", "green")),
  strokeColors = NULL, titlePrefix = "Means and associations with",
  titleVarLabels = NULL, titleSuffix = "", fullColorRange = NULL,
  associationsAlpha = 0.5, baseSize = 0.8, dotSize = 2.5 * baseSize,
  baseFontSize = 10 * baseSize, theme = ggplot2::theme_bw(base_size =
  baseFontSize), ...)
```

Arguments

`data`	The dataframe containing the variables.
`determinants`	The 'determinants': the predictors (or 'covariates') of the target variables(s) (or 'criteria').
`targets`	The 'targets' or 'criteria' variables: the variables predicted by the determinants.
`conf.level`	The confidence levels for the confidence intervals: has to be a named list with two elements: `means` and `associations`, specifying the desired confidence levels for the means and associations, respectively. The confidence level for the associations is also used for the intervals for the proportions of explained variance.
`subQuestions`	The subquestions used to measure each determinants. This can also be used to provide pretty names for the variables if the determinants were not measured by one question each. Must have the same length as `determinants`.
`leftAnchors`	The anchors to display on the left side of the left hand panel. If the determinants 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 `determinants`.
`rightAnchors`	The anchors to display on the left side of the left hand panel. If the determinants 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 `determinants`.
`orderBy`	Whether to sort the determinants. Set to `NULL` to not sort at all; specify the name or index of one of the `targets` to sort by the point estimates of the associations with that target variable. Use `decreasing` to determine whether to sort in ascending or descending order. For convenience, if `orderBy` is not `NULL`, but `decreasing` is, the determinants are sorted in descending (decreasing) order.
`decreasing`	Whether to sort the determinants. Specify `NULL` to not sort at all, `TRUE` to sort in descending order, and `FALSE` to sort in ascending order. If `decreasing` is

	nor `NULL`, but `orderBy` is `NULL`, the determinants are sorted by their means. For convenience, if `orderBy` is not `NULL`, but `decreasing` is, the determinants are sorted in descending (decreasing) order.
`numberSubQuestions`	Whether or not to number the subquestions. If they are numbered, they are numbered from the top to the bottom.
`generateColors`	The colors to use to generate the gradients for coloring the diamonds representing the confidence intervals. Has to be a named list with two elements: `means` and `associations`, specifying the desired colors for the means and associations, respectively.
`strokeColors`	The palette to use to color the stroke of the confidence intervals for the associations between the determinants and the targets. Successive colors from this palette are used for the targets.
`titlePrefix`	Text to add before the list of target names and the proportions of explained variance for each target. This plot title also serves as legend to indicate which target 'gets' which each color.
`titleVarLabels`	Optionally, variable labels to use in the plot title. Has to be the exact same length as `targets`.
`titleSuffix`	Text to add after the list of target names and the proportions of explained variance for each target.
`fullColorRange`	If colors are specified, this can be used to specify which values, for the determinant confidence intervals in the left hand panel, are the minimum and maximum. This is useful if those scores are not actually in the data (e.g. for extremely skewed distributions). If `NULL`, the range of all individual scores on the determinants is used. For the associations, `c(-1, 1)` is always used as `fullColorRange`.
`associationsAlpha`	The alpha level (transparency) of the confidence interval diamonds in the right hand plot. Value between 0 and 1, where 0 signifies complete transparency (i.e. invisibility) and 1 signifies complete 'opaqueness'.
`returnPlotOnly`	Whether to return the entire object that is generated (including all intermediate objects) or only the plot.
`drawPlot`	Whether the draw the plot, or only return it.
`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.
`xbreaks`	Which breaks to use on the X axis (can be useful to override `ggplot2`'s defaults).
`...`	These arguments are passed on to `biAxisDiamondPlot` (for the left panel) and `diamondPlot` (for the right panel). Note that all argument are passed to both those functions.

`determinantStructure`	When using `detStructCIBER`, the determinant structure as generated by `determinantStructure` is included here. `determinants`, `targets`, `subQuestions`, `leftAnchors`, and `rightAnchors` are then read from the `determinantStructure` object. In other words: once a `determinantStructure` has been generated, only `dat` and `determinantStructure` have to be provided as argument to generate a CIBER diamond plot.

Details

Details are explained in Crutzen & Peters (2017).

Value

Depending on the value of `returnPlotOnly`, either the plot only (a `gtable` object) or an object containing most objects created along the way (in which case the plot is stored in `$output$plot`).

The plot has `width` and `height` attributes which can be used when saving the plot.

References

Crutzen, R., Peters, G.-J. Y., & Noijen, J. (2017). How to Select Relevant Social-Cognitive Determinants and Use them in the Development of Behaviour Change Interventions? Confidence Interval-Based Estimation of Relevance. http://dx.doi.org/

See Also

`determinantStructure`

Examples

```
## Not run:
CIBER(data=mtcars,
      determinants=c('drat', 'wt', 'am',
                     'gear', 'vs', 'carb'),
      targets=c('mpg', 'cyl'));

## End(Not run)
```

`confIntOmegaSq` *Confidence intervals for Omega Squared*

Description

This function used the MBESS function `conf.limits.ncf` and `convert.ncf.to.omegasq` to compute the point estimate and confidence interval for Omega Squared.

Usage

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

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.

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)
```

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.

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 of the hypothesized sampling distribution (assuming the sample value happens to be the population value) of Pearson's *r*.

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'];
```

`confIntV` *crossTab, confIntV and cramersV*

Description

These functions compute the point estimate and confidence interval for Cramer's V. The crossTab function also shows a crosstable.

Usage

```
crossTab(x, y=NULL, conf.level=.95,
         digits=2, pValueDigits=3, ...)
cramersV(x, y = NULL, digits=2)
confIntV(x, y = NULL, conf.level=.95,
         samples = 500, digits=2,
         method=c('bootstrap', 'fisher'),
         storeBootstrappingData = FALSE)
```

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.
`pValueDigits`	Minimum number of digits after the decimal point to show in the Chi Square p value in the result.

`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.
`...`	Extra arguments to `crossTab` are passed on to `confIntV`.

Value

The cramersV and confIntV functions return either a point estimate or a confidence interval for Cramer's V, an effect size to describe the association between two categorical variables. The crossTab function is just a wrapper around confIntV.

Examples

```
crossTab(infert$education, infert$induced, samples=50);

### 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));
```

`convert` *conversion functions*

Description

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

Usage

```
convert.b.to.t(b, se)

convert.chisq.to.p(chisq, df, lower.tail=FALSE)
convert.chisq.to.V(chisq, n, minDim)

convert.cohensf.to.omegasq(cohensf)

convert.cohensfsq.to.omegasq(cohensfsq)

convert.d.to.logodds(d)
convert.d.to.r(d, n1 = NULL, n2 = NULL, akfEq8='if (n1 + n2) < 50')
convert.d.to.t(d, df = NULL, n1 = NULL, n2 = NULL, proportion = 0.5)
convert.d.to.variance(d, n1, n2)
```

```
convert.etasq.to.cohensf(etasq)

convert.f.to.etasq(f, df1, df2)
convert.f.to.omegasq(f, df1, df2)
convert.f.to.p(f, df1, df2, lower.tail=FALSE)
convert.f.to.d(f, df1, df2 = NULL, n1=NULL, n2=NULL, proportion=.5)

convert.fisherz.to.r(z)

convert.logodds.to.d(logodds)
convert.logodds.to.r(logodds)

convert.means.to.d(means, sds, ns = NULL, var.equal = NULL)

convert.ncf.to.omegasq(ncf, N)

convert.omegasq.to.cohensf(omegasq)
convert.omegasq.to.cohensfsq(omegasq)
convert.omegasq.to.f(omegasq, df1, df2)

convert.or.to.d(or)
convert.or.to.r(or)

convert.percentage.to.se(p, n)

convert.r.to.t(r, n)
convert.r.to.d(r)
convert.r.to.p(r, n)
convert.r.to.fisherz(r)

convert.t.to.r(t, n)
convert.t.to.d(t, df=NULL, n1=NULL, n2=NULL, proportion=.5)
convert.t.to.p(t, df)
```

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.d.to.nnc` *Helper functions for Numbers Needed for Change*

Description

These two functions are used by `nnc` to compute the Numbers Needed for Change.

Usage

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

Arguments

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

Details

These two functions are used by `nnc` to compute the 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. K., & Peters, G.-J. Y. (2017). Introducing the Numbers Needed for Change (NNC): A practical measure of effect size for intervention research.

See Also

`nnc`

Examples

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

---

createSigma *createSigma: convenience function for mvrnorm*

Description

This function is made to quickly generate a Sigma matrix of the type required by `mvrnorm`. By specifying the number of variables, the mean correlation, and how much variation there should be in the correlations, it's easy to quickly generate a correlation matrix.

Usage

```
createSigma(nVar, meanR = 0.3, sdR = 0, diagonal = 1)
```

Arguments

`nVar`	The number of variables in the correlation matrix.
`meanR`	The average correlation, provided to `rnorm` together with `sdR` to generate the correlations.
`sdR`	The variation in the correlations, provided to `rnorm` together with `meanR` to generate the correlations.
`diagonal`	The value on the diagonal of the returned matrix: will normally be 1.

Value

A matrix of nVar x nVar.

Author(s)

Gjalt-Jorn Peters

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

See Also

`mvrnorm`, `rnorm`, `matrix`

Examples

```
createSigma(3, .5, .1);
```

`curfnfinder` *Function to find the name of the calling function*

Description

This function finds and returns the name of the function calling it. This can be useful, for example, when generating functions algorithmically.

Usage

```
curfnfinder(skipframes = 0, skipnames = "(FUN)|(.+apply)|(replicate)",
            retIfNone = "Not in function", retStack = FALSE,
            extraPrefPerLevel = "\t")
```

Arguments

`skipframes`	Number of frames to skip; useful when called from an anonymous function.
`skipnames`	A regular expression specifying which substrings to delete.
`retIfNone`	What to return when called from outside a function.
`retStack`	Whether to return the entire stack or just one function.
`extraPrefPerLevel`	Extra prefixes to return for each level of the function.

Details

This function was written by Nick Sabbe for his package `addendum`. He posted it on Stack Exchange at `http://stackoverflow.com/questions/7307987/logging-current-function-name` and I included this here with this permission.

Value

The current function.

Author(s)

Nick Sabbe (Arteveldehogeschool)

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

Examples

```
functionA <- functionB <- function() {
  curFn <- curfnfinder();
  if (curFn == 'functionA') {
    cat('Doing something\n');
  } else {
    cat('Doing something else\n');
  }
```

```
  cat('Doing something generic.');
}
functionA();
functionB();
```

`dCohensd` *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

```
dd(x, df=NULL, populationD = 0,
   n=NULL, n1=NULL, n2=NULL,
   silent=FALSE)
pd(q, df, populationD = 0, lower.tail = TRUE)
qd(p, df, populationD = 0, lower.tail = TRUE)
rd(n, df, populationD = 0)

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)

pdExtreme(d, n, populationD=0)
pdMild(d, n, populationD=0)
pdInterval(ds, n, populationD=0)

cohensdCI(d, n, conf.level = .95, plot=FALSE, silent=TRUE)
confIntD(d, n, conf.level = .95, plot=FALSE, silent=TRUE)

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

Arguments

`x, q, d`	Vector of quantiles, or, in other words, the value(s) of Cohen's *d*.
`ds`	A vector with two Cohen's *d* values.
`p`	Vector of probabilites (*p*-values).
`df, n1, n2`	Degrees of freedom or sample sizes for each group (`n1` and `n2`); also see explanation for `n` below.
`n`	Total `n` (`df` + 2) for `dd` and `dCohensd`, desired number of Cohen's *d* values for `rCohensd` and `rd`, and the number of participants/datapoints for `pdExtreme`, `pdMild`, `pdInterval`, and `cohensdCI`.
`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.
`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`.
`w`	The desired 'half-width' or margin of error of the confidence interval.
`extensive`	Whether to only return the required sample size, or more extensive results.
`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.

Details

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

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

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);
```

`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=.95,
      quantileType = 2);
```

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.

Details

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

The Hartigans' Dip Test may be unfamiliar to users; it is a measure of uni- vs. multidimensionality, computed by `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 `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`, `describe`

Examples

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

`detectRareWords` *Looking up word frequencies*

Description

This function checks, for each word in a text, how frequently it occurs in a given language. This is useful for eliminating rare words to make a text more accessible to an audience with limited vocabulary. `htmlParse` and `xpathSApply` from the `XML` package are used to process HTML files, if necessary. `textToWords` is a helper function that simply breaks down a character vector to a vector of words.

Usage

```
detectRareWords(textFile = NULL,
                wordFrequencyFile = "Dutch",
                output = c("file", "show", "return"),
                outputFile = NULL,
                wordCol = "Word", freqCol = "FREQlemma",
                textToWordsFunction = "textToWords",
                encoding = "ASCII",
                xPathSelector = "/text()",
                silent = FALSE)
textToWords(characterVector)
```

Arguments

`textFile`	If NULL, a dialog will be shown that enables users to select a file. If not NULL, this has to be either a filename or a character vector. An HTML file can be provided; this will be parsed using
`wordFrequencyFile`	The file with word frequencies to use. If 'Dutch' or 'Polish', files from the Center for Reading Research (http://crr.ugent.be/) are downloaded.
`output`	How to provide the output, as a character vector. If `file`, the filename to write to should be provided in `outputFile`. If `show`, the output is shown; and if `return`, the output is returned invisibly.
`outputFile`	The name of the file to store the output in.
`wordCol`	The name of the column in the `wordFrequencyFile` that contains the words.
`freqCol`	The name of the column in the `wordFrequencyFile` that contains the frequency with which each word occurs.
`textToWordsFunction`	The function to use to split a character vector, where each element contains one or more words, into a vector where each element is a word.
`encoding`	The encoding used to read and write files.
`xPathSelector`	If the file provided is an HTML file, `xpathSApply` is used to extract the content. `xPathSelector` specifies which content to extract (the default value extracts all text content).
`silent`	Whether to suppress detailed feedback about the process.
`characterVector`	A character vector, the elements of which are to be broken down into words.

Value

`detectRareWords` return a dataframe (invisibly) if `output` contains `return`. Otherwise, NULL is returned (invisibly), but the output is printed and/or written to a file depending on the value of `output`.

`textToWords` returns a vector of words.

Author(s)

Gjalt-Jorn Peters

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

Examples

```
## Not run:
detectRareWords(paste('Dit is een tekst om de',
                      'werking van de detectRareWords',
                      'functie te demonstreren.'),
                output='show');

## End(Not run)
```

`determinantStructure` *Determinant Structure specification*

Description

These functions can be used to specify a determinant structure: a hierarchical structure of determinants that can then be conveniently plotted and analysed, for example using `detStructCIBER`.

These functions are made to be used together; see the example and the forthcoming article for more information.

Usage

```
determinantStructure(name, selection = NULL, ...)

determinantVar(name, selection = NULL, ...)

subdeterminants(name, selection = NULL, ...)

subdeterminantProducts(name, selection = NULL, ...)

## S3 method for class 'determinantStructure'
print(x, ...)
## S3 method for class 'determinantStructure'
plot(x, useDiagrammeR = FALSE, ...)
```

Arguments

`name`	The name of the variable that is specified.
`selection`	A regular expression to use to select the variables in a dataframe that are considered items that together form this variable. For `determinantStructure`, a list can be provided that also contains a named regular expression with the name ’behaviorRegEx’, which specifies the name of the behavior to which this determinant structure pertains.

`x`	The `determinantStructure` object to print or plot.
`useDiagrammeR`	Whether to simply use `print(plot(x))` (if `FALSE`) or whether to use `ToDiagrammeRGraph`, tweak it a bit, by setting global graph attributes, and then using `render_graph` (if `TRUE`).
`...`	Any additional arguments are other determinant structure building functions. These are used to construct the determinant structure 'tree'.

Details

This family of functions will be explained more in detail in a forthcoming paper.

`plot` and `print` methods plot and print a determinantStructure object.

Value

A determinantStructure object, which is a `data.tree` object.

Author(s)

Gjalt-Jorn Peters

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

References

(Forthcoming)

See Also

`detStructAddVarLabels`, `detStructAddVarNames`, `detStructComputeProducts`, `detStructComputeScales`, `detStructCIBER`

Examples

```
determinantStructure('using R',
                     list('using R',
                          behaviorRegEx = 'some RegEx'),
                     determinantVar("Intention",
                                    "another RegEx",
                                    determinantVar("Attitude",
                                                   "third RegEX",
                                                   subdeterminants("Likelihood",
                                                                   "4th RegEx"),
                                                  subdeterminants("Evaluation",
                                                                  "5th RegEx"),
                                                  subdeterminantProducts("attProduct",
                                                                         c("4th RegEx",
                                                                         "5th RegEx"))),
                                    determinantVar("perceivedNorm",
                                                   "6th RegEx",
                                                   subdeterminants("Approval",
                                                                   "7th RegEx"),
```

```
                                                   subdeterminants("Motivation to comply",
                                                                   "8th RegEx"),
                                                     subdeterminantProducts("normProduct",
                                                                            c("7th RegEx",
                                                                              "8th RegEx"))),
                                              determinantVar("pbc",
                                                             "9th RegEx",
                                                        subdeterminants("Control beliefs",
                                                                        "10th RegEx"))));
```

---

determinantStructure Preprocessing

Functions to preprocess determinant structures

Description

These functions are used in conjunction with the `determinantStructure` family of funtions to conveniently work with determinant structures.

Usage

```
detStructAddVarLabels(determinantStructure,
                      varLabelDf,
                      varNameCol = "varNames.cln",
                      leftAnchorCol = "leftAnchors",
                      rightAnchorCol = "rightAnchors",
                      subQuestionCol = "subQuestions",
                      questionTextCol = "questionText")

detStructAddVarNames(determinantStructure, names)

detStructComputeProducts(determinantStructure, dat, append = TRUE)

detStructComputeScales(determinantStructure, dat, append = TRUE, separator = "_")
```

Arguments

`determinantStructure`	The `determinantStructure` object.
`varLabelDf`	The variable label dataframe as generated by `processLSvarLabels`. It is also possible to specify 'homemade' dataframe, in which case the column names have to specified (see the next arguments).
`varNameCol`	The name of the column of the `varLabelDf` that contains the variable name. Only needs to be changed from the default value if `varLabelDf` is not a dataframe as produced by `processLSvarLabels`.

`leftAnchorCol`	The name of the column of the `varLabelDf` that contains the left anchor. Only needs to be changed from the default value if `varLabelDf` is not a dataframe as produced by `processLSvarLabels`.
`rightAnchorCol`	The name of the column of the `varLabelDf` that contains the right anchor. Only needs to be changed from the default value if `varLabelDf` is not a dataframe as produced by `processLSvarLabels`.
`subQuestionCol`	The name of the column of the `varLabelDf` that contains the subquestion. Only needs to be changed from the default value if `varLabelDf` is not a dataframe as produced by `processLSvarLabels`.
`questionTextCol`	The name of the column of the `varLabelDf` that contains the question text. Only needs to be changed from the default value if `varLabelDf` is not a dataframe as produced by `processLSvarLabels`.
`names`	A character vector with the variable names. These are matched against the regular expressions as specified in the `determinantStructure` object, and any matches will be stored in the `determinantStructure` object.
`dat`	The dataframe containing the data; the variables names specified in `names` (when calling `detStructAddVarNames`) must be present in this dataframe.
`append`	Whether to only return the products or scales, or whether to append these to the dataframe and return the entire dataframe.
`separator`	The separator to use when constructing the scale variables names.

Details

This family of functions will be explained more in detail in a forthcoming paper.

Value

`detStructAddVarLabels` and `detStructAddVarNames` just change the `determinantStructure` object; `detStructComputeProducts` and `detStructComputeScales` return either the dataframe with the new variables appended (if `append` = `TRUE`) or just a dataframe with the new variables (if `append` = `FALSE`).

Author(s)

Gjalt-Jorn Peters

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

References

(Forthcoming)

See Also

`determinantStructure`, `determinantVar`, `subdeterminants`, `subdeterminantProducts`, `detStructCIBER`

Examples

```
### Generate a silly determinant structure
detStruct <- determinantStructure('This makes no sense',
                                  list('mpg',
                                       behaviorRegEx = 'mpg'),
                                  determinantVar("Proximal determinant",
                                                 "t",
                                                 determinantVar("Determinant",
                                                                "p",
                                                      subdeterminants("Subdeterminants",
                                                                      "a"))));

### Add the variable names
detStructAddVarNames(detStruct, names(mtcars));

### Add the determinant scale variable to the dataframe
mtcarsPlus <- detStructComputeScales(detStruct, mtcars);

### Show its presence
names(mtcarsPlus);
mean(mtcarsPlus$mpg_Determinant);
```

`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 = 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).
`xlab, ylab`	The labels of the X and Y axes.
`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.
`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 `ggplot` plot with a `ggDiamondLayer` is returned.

Author(s)

Gjalt-Jorn Peters

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

See Also

`meansDiamondPlot`, `meanSDtoDiamondPlot`, `factorLoadingDiamondCIplot`, `ggDiamondLayer`

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));
```

`didacticPlot` *didacticPlot*

Description

didacticPlot is useful for making ggplot2 plots of distributions of t, F, Chi^2, and Pearson r, showing a given value, and shading the arie covering the more extreme values. didacticPlotTheme is the basic theme.

Usage

```
didacticPlot(foundValue, statistic, df1, df2 = NULL,
             granularity = 1000, xLim = NULL, yLab = NULL,
             lineCol = "red", lineSize=1,
             surfaceCol = "red", textMarginFactor = 20,
             sided="two")
didacticPlotTheme(base_size = 14, base_family = "")
```

Arguments

`foundValue`	The value to indicate (the 'found' value).
`statistic`	One of "r", "t", "f" or "chisq".
`df1, df2`	The degrees of freedom; only use df1 for the r, t and chi^2 test; for the F-test, use df1 for the degrees of freedom of the denominator and df2 for the degrees of freedom of the numerator.

`granularity`	Steps to use for x-axis.
`xLim`	Vector; minimum and maximum values on x axis.
`yLab`	Label on y axis.
`lineCol`	Colour of density line.
`lineSize`	Size of density line.
`surfaceCol`	Colour of coloured surface area.
`textMarginFactor`	Used to calculate how close to the vertical line text labels should appear.
`sided`	Whether to make a plot for a 2-sided or 1-sided test.
`base_size, base_family`	Passed on to the grey ggplot theme.

Value

didacticPlot returns an object that contains the plot in the $plot element.

Examples

```
didacticPlot(1, statistic='chisq', df1=2);

didacticPlot(1, statistic='t', df1=40);

didacticPlot(2.02, statistic='t', df1=40, textMarginFactor=25);

### Two sample t-test for n1 = n2 = 250, showing
### p-value of 5%
# a<-didacticPlot(1.96, statistic='t', df1=498);
```

`dlvPlot` *dlvPlot*

Description

The dlvPlot function produces a dot-violin-line plot, and dlvTheme is the default theme.

Usage

```
dlvPlot(dat, x = NULL, y, z = NULL,
        conf.level = .95,
        jitter = "FALSE",
        binnedDots = TRUE, binwidth=NULL,
        error="lines",
        dotsize="density",
        singleColor = "black",
        comparisonColors = brewer.pal(8, 'Set1'),
```

```
          densityDotBaseSize=3,
          normalDotBaseSize=1,
          violinAlpha = .2,
          dotAlpha = .4,
          lineAlpha = 1,
          connectingLineAlpha = 1,
          meanDotSize=5,
          posDodge=0.2,
          errorType = "both",
          outputFile = NULL,
          outputWidth = 10,
          outputHeight = 10,
          ggsaveParams = list(units='cm',
                              dpi=300,
                              type="cairo"))
dlvTheme(base_size = 11, base_family = "", ...)
```

Arguments

`dat`	The dataframe containing x, y and z.
`x`	Character value with the name of the predictor ('independent') variable, must refer to a categorical variable (i.e. a factor).
`y`	Character value with the name of the critetion ('dependent') variable, must refer to a continuous variable (i.e. a numeric vector).
`z`	Character value with the name of the moderator variable, must refer to a categorical variable (i.e. a factor).
`conf.level`	Confidence of confidence intervals.
`jitter`	Logical value (i.e. TRUE or FALSE) whether or not to jitter individual datapoints. Note that jitter cannot be combined with posDodge (see below).
`binnedDots`	Logical value indicating whether to use binning to display the dots. Overrides jitter and dotsize.
`binwidth`	Numeric value indicating how broadly to bin (larger values is more binning, i.e. combining more dots into one big dot).
`error`	Character value: "none", "lines" or "whiskers"; indicates whether to show the confidence interval as lines with (whiskers) or without (lines) horizontal whiskers or not at all (none)
`dotsize`	Character value: "density" or "normal"; when "density", the size of each dot corresponds to the density of the distribution at that point.
`singleColor`	The color to use when drawing one or more univariate distributions (i.e. when no z is specified.
`comparisonColors`	The colors to use when a z is specified. This should be at least as many colors as z has levels. By default, palette `Set1` from `RColorBrewer` is used.
`densityDotBaseSize`	Numeric value indicating base size of dots when their size corresponds to the density (bigger = larger dots).

`normalDotBaseSize`	Numeric value indicating base size of dots when their size is fixed (bigger = larger dots).
`violinAlpha`	Numeric value indicating alpha value of violin layer (0 = completely transparent, 1 = completely opaque).
`dotAlpha`	Numeric value indicating alpha value of dot layer (0 = completely transparent, 1 = completely opaque).
`lineAlpha`	Numeric value indicating alpha value of the confidence interval line layer (0 = completely transparent, 1 = completely opaque).
`connectingLineAlpha`	Numeric value indicating alpha value of the layer with the lines connecting the means (0 = completely transparent, 1 = completely opaque).
`meanDotSize`	Numeric value indicating the size of the dot used to indicate the mean in the line layer.
`posDodge`	Numeric value indicating the distance to dodge positions (0 for complete overlap).
`errorType`	If the error is shown using lines, this argument indicates Whether the errorbars should show the confidence interval (`errorType='ci'`), the standard errors (`errorType='se'`), or both (`errorType='both'`). In this last case, the standard error will be wider than the confidence interval.
`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.
`base_size, base_family, ...`	Passed on to the ggplot theme_grey() function.

Details

This function creates Dot Violin Line plots. One image says more than a thousand words; I suggest you run the example :-)

Value

The behavior of this function depends on the arguments.

If no x and z are provided and y is a character value, dlvPlot produces a univariate plot for the numerical y variable.

If no x and z are provided, and y is c character vector, dlvPlot produces multiple Univariate plots, with variable names determining categories on x-axis and with numerical y variables on y-axis

If both x and y are a character value, and no z is provided, dlvPlot produces a bivariate plot where factor x determines categories on x-axis with numerical variable y on the y-axis (roughly a line plot with a single line)

Finally, if x, y and z are each a character value, dlvPlot produces multivariate plot where factor x determines categories on x-axis, factor z determines the different lines, and with the numerical y variable on the y-axis

An object is returned with the following elements:

`dat.raw`	Raw datafile provided when calling dlvPlot
`dat`	Transformed (long) datafile dlvPlot uses
`descr`	Dataframe with extracted descriptives used to plot the mean and confidence intervals
`yRange`	The range of the Y variable used to construct the plot
`plot`	The plot itself

Examples

```
### Note: the 'not run' is simply because running takes a lot of time,
###       but these examples are all safe to run!
## Not run:
### Create simple dataset
dat <- data.frame(x1 = factor(rep(c(0,1), 20)),
                  x2 = factor(c(rep(0, 20), rep(1, 20))),
                  y=rep(c(4,5), 20) + rnorm(40));
### Generate a simple dlvPlot of y
dlvPlot(dat, y='y');
### Now add a predictor
dlvPlot(dat, x='x1', y='y');
### And finally also a moderator:
dlvPlot(dat, x='x1', y='y', z='x2');
### The number of datapoints might be a bit clearer if we jitter
dlvPlot(dat, x='x1', y='y', z='x2', jitter=TRUE);
### Although just dodging the density-sized dots might work better
dlvPlot(dat, x='x1', y='y', z='x2', posDodge=.3);

## End(Not run)
```

`escapeRegex` *Escapes any characters that would have special meaning in a reqular expression.*

Description

Escapes any characters that would have special meaning in a reqular expression.

Usage

```
escapeRegex(string)
```

Arguments

`string`	string being operated on.

Details

`escapeRegex` will escape any characters that would have special meaning in a reqular expression. For any string `grep(regexpEscape(string), string)` will always be true.

Value

The value of the string with any characters that would have special meaning in a reqular expression escaped.

Note

Note that this function was copied literally from the `Hmisc` package (to prevent importing the entire package for one line of code).

Author(s)

Charles Dupont
Department of Biostatistics
Vanderbilt University

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

See Also

`grep`, `Hmisc`, `http://biostat.mc.vanderbilt.edu/wiki/Main/Hmisc`, `https://github.com/harrelfe/Hmisc`

Examples

```
string <- "this\\(system) {is} [full]."
escapeRegex(string)
```

`examine` *Examine one or more variables*

Description

These functions are one of many R functions enabling users to assess variable descriptives. They have been developed to mimic SPSS' 'EXAMINE' syntax command ('Explore' in the menu) as closely as possible to ease the transition for new R users and facilitate teaching courses where both programs are taught alongside each other.

Usage

```
examine(..., stem = TRUE, plots = TRUE,
        extremeValues = 5, descr.include = NULL,
        qqCI = TRUE, conf.level = 0.95)
examineBy(..., by=NULL, stem = TRUE, plots = TRUE,
          extremeValues = 5, descr.include=NULL,
          qqCI = TRUE, conf.level=.95)
```

Arguments

`...`	The first argument is a list of variables to provide descriptives for. Because these are the first arguments, the other arguments must be named explicitly so R does not confuse them for something that should be part of the dots.
`by`	A variable by which to split the dataset before calling `examine`. This can be used to show the descriptives separate by levels of a factor.
`stem`	Whether to display a stem and leaf plot.
`plots`	Whether to display the plots generated by the `dataShape` function.
`extremeValues`	How many extreme values to show at either end (the highest and lowest values). When set to FALSE (or 0), no extreme values are shown.
`qqCI`	Whether to display confidence intervals in the QQ-plot.
`descr.include`	Which descriptives to include; see `descr` for more information.
`conf.level`	The level of confidence of the confidence interval.

Details

This function basically just calls the `descr` function, optionally supplemented with calls to `stem`, `dataShape`.

Value

A list that is displayed when printed.

Author(s)

Gjalt-Jorn Peters

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

See Also

`descr`, `dataShape`, `stem`

Examples

```
### Look at the miles per gallon descriptives:
examine(mtcars$mpg, stem=FALSE, plots=FALSE);

### Separate for the different number of cylinders:
examineBy(mtcars$mpg, by=mtcars$cyl,
          stem=FALSE, plots=FALSE,
          extremeValues=FALSE,
          descr.include=c('central tendency', 'spread'));
```

`exceptionalScore` *exceptionalScore*

Description

This function can be used to detect exceptionally high or low scores in a vector.

Usage

```
exceptionalScore(x, prob = 0.025, both = TRUE, silent = FALSE,
                 quantileCorrection = 1e-04, quantileType = 8)
```

Arguments

`x`	Vector in which to detect exceptional scores.
`prob`	Probability that a score is exceptionally positive or negative; i.e. scores with a quartile lower than `prob` or higher than 1-`prob` are considered exceptional (if both is TRUE, at least). So, note that a `prob` of .025 means that if both=TRUE, the most exceptional 5% of the values is marked as such.
`both`	Whether to consider values exceptional if they're below `prob` as well as above 1-`prob`, or whether to only consider values exceptional if they're below `prob` is `prob` is < .5, or above `prob` if `prob` > .5.
`silent`	Can be used to suppress messages.
`quantileCorrection`	By how much to correct the computed quantiles; this is used because when a distribution is very right-skewed, the lowest quantile is the lowest value, which is then also the mode; without subtracting a correction, almost all values would be marked as 'exceptional'.
`quantileType`	The algorithm used to compute the quantiles; see `quantile`.

Details

Note that of course, by definition, `prob` of `2*prob` percent of the values is exceptional, so it is usually not a wise idea to remove scores based on their 'exceptionalness'. Instead, use `exceptionalScores`, which calls this function, to see how often participants answered exceptionally, and remove them based on that.

Value

A logical vector, indicating for each value in the supplied vector whether it is exceptional.

Author(s)

Gjalt-Jorn Peters

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

See Also

`quantile`, `exceptionalScores`

Examples

```
exceptionalScore(c(1,1,2,2,2,3,3,3,4,4,4,5,5,5,5,6,6,7,8,20), prob=.05);
```

`exceptionalScores` *exceptionalScores*

Description

A function to detect participants that consistently respond exceptionally.

Usage

```
exceptionalScores(dat, items = NULL, exception = 0.025, totalOnly = TRUE,
                  append = TRUE, both = TRUE, silent = FALSE,
                 suffix = "_isExceptional", totalVarName = "exceptionalScores")
```

Arguments

`dat`	The dataframe containing the variables to inspect, or the vector to inspect (but for vectors, `exceptionalScore` might be more useful).
`items`	The names of the variables to inspect.
`exception`	When an item will be considered exceptional, passed on as `prob` to `exceptionalScore`.
`totalOnly`	Whether to return only the number of exceptional scores for each row in the dataframe, or for each inspected item, which values are exceptional.
`append`	Whether to return the supplied dataframe with the new variable(s) appended (if TRUE), or whether to only return the new variable(s) (if FALSE).
`both`	Whether to look for both low and high exceptional scores (TRUE) or not (FALSE; see `exceptionalScore`).
`silent`	Can be used to suppress messages.
`suffix`	If not returning the total number of exceptional values, for each inspected variable, a new variable is returned indicating which values are exceptional. The text string is appended to each original variable name to create the new variable names.
`totalVarName`	If returning only the total number of exceptional values, and appending these to the provided dataset, this text string is used as variable name.

Value

Either a vector containing the number of exceptional values, a dataset containing, for each inspected variable, which values are exceptional, or the provided dataset where either the total or the exceptional values for each variable are appended.

Author(s)

Gjalt-Jorn Peters

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

See Also

`exceptionalScore`

Examples

```
exceptionalScores(mtcars)
```

`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');
```

`facComAnalysis` *Wrapper for psych's factor analysis and principal components analysis functions*

Description

This function is meant as a wrapper to the excellent `fa` and `principal` functions from the `psych` package. This function was developed to provide a more familiar interface for users coming from SPSS.

Usage

```
facComAnalysis(data,
               items = NULL,
               nfactors = NULL,
               fm = "pa",
               rotate = "default",
               covar = FALSE,
               screeplot = TRUE,
               SMC = FALSE,
               maskLoadingsUnder = NULL,
               showUnrotatedLoadings = FALSE,
               factorCorrelations = "score.cor",
               ...)
```

Arguments

`data`	The dataframe containt the items to analyse.
`items`	The items to analyse; if none are specified, all items in the dataframe are analyses.
`nfactors`	The number of factors to extract. The default, `NULL`, applies the Kaiser criterion (like SPSS does).
`fm`	The method to use. Specify `pca` to conduct a principal components analysis (PCA; using `principal`), or one of the methods accepted by `psych`'s `fa` to conduct a factor analysis. The default (`pa`) performs an exploratory factor analysis using principal axis factoring. Note that SPSS' default is to perform PCA, which is usually wrong in psychological research. In PCA, the components (called factors in factor analysis) are constructed to optimally explain all item variance (i.e. the variances if a covariance matrix is analysed, and `1` if a correlation matrix is analysed, see argument `covar` below) as well as the associations between items (i.e. the covariances or correlations, depending on the value of `covar`). This means that variance that is unique to an item is considered important. In factor analysis, the factors (called components in PCA) are constructed to optimally explain only the variation shared between all items. This means that only a part of the variance of each item is explained: therefore, the diagonal of the correlation matrix (or covariance matrix) is replaced by an estimate

	of the so-called communality: that portion of variance that each item shared with other items. Of course, this is unknown: therefore, an iterative procedure is used where some initial value is used as communality estimate (and placed on the diagonal in the first iteration). Which value is used as initial estimate can be specified in the `SMC` argument. In any case, this difference means that in factor analysis, the unique variance in each item (the uniqueness or unicity) is assumed to reflect measurement error. This is consistent with latent variable models, which are usually what underlie operationalisations of psychological constructs. These operationalisations usually consist of items (or 'indicators') where scores registered for each item are assumed to reflect (more accurately, be caused by) a psychological construct (the latent variable). Therefore, in such cases, PCA is inappropriate. If the operationalisation is an index rather than a scale (see Peters, 2014, for a discussion of this distinction), however, factor analysis is inappropriate, and PCA should be used. This difference nicely coincides with the distinction between reflective models (where factor analysis is appropriate) and formative models (where PCA is appropriate): see Freid (2017) for an introduction.
`rotate`	Which rotation to use. The default, aptly called `default`, uses the `psych` defaults: `oblimin` for factor analysis and `varimax` for PCA.
`covar`	Whether to analyse the covariance matrix or the correlation matrix. If the items are to be aggregated without first standardizing them (which is by far the more common approach), the covariance matrix should be analysed. However, if the items are going to be standardized before aggregation, the correlation matrix will be used. However, because analysing the correlation matrix is the default setting in both SPSS and `psych`'s functions, it is also retained as default here.
`screeplot`	Whether to generate and show a screeplot.
`SMC`	The SMC argument can be used to specify, for factor analysis, whether to start the initial iteration with 1 (`SMC=FALSE`), the squared multiple correlations or R squared values obtained when regressing each item on all the others (`SMC=TRUE`), or custom values which then have to be provided in `SMC` in a numeric vector (of equal length to the number of items).
`maskLoadingsUnder`	Whether to show all factor loadings (if set to `NULL` or anything non-numeric) or whether to mask (hide) factor loadings under a given value, e.g. only showing factor loadings of .3 or higher.
`showUnrotatedLoadings`	Whether to only show the rotated factor loadings, or the original (unrotated) factor loadings as well.
`factorCorrelations`	Whether to show the factor correlations from `score.cor` ("The correlation matrix of course coded (unit weighted) factor score estimates, if they were to be found, based upon the loadings matrix rather than the weights matrix.", see `fa`) or from `r.scores` ("The correlations of the factor score estimates using the specified model, if they were to be found.", see `fa`) from the objects produced by the `psych` functions.
`...`	Any additional arguments are passed to `psych`'s `fa` and `principal` functions.

Value

This function returns an object with the original `psych` function objects in the `intermediate` sub-object, and the primary results such as the factor loading and the plot in the `output` sub-object.

Author(s)

Gjalt-Jorn Peters

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

References

Fried, E. I. (2017). What are psychological constructs? On the nature and statistical modeling of emotions, intelligence, personality traits and mental disorders. *Health Psychology Review*, 11(2), 130-134. `http://doi.org/10.1080/17437199.2017.1306718`

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`

See Also

`psych`, `fa`, `principal` and `reliability`

Examples

```
### Generate data frame to use for the example
dat <- as.data.frame(apply(mtcars, 2, scale));

### Conduct principal components analysis
facComAnalysis(dat, fm='pca');

### Conduct factor analysis, and mask
### all factor loadings under .3
facComAnalysis(dat, maskLoadingsUnder = .3);
```

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

Description

This function contains some code from a function in `psych` 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 `fa` function from the `psych` package. It is important that the `n.iter` argument of `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`.

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(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_pal()(max(2, fa$factors)),
                           labels=NULL,
                           theme=theme_bw(),
                           ...)
```

Arguments

`fa`	The object produced by the `fa` function from the `psych` package. It is important that the `n.iter` argument of `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 `ggplot` plot with several `ggDiamondLayer`s is returned.

Author(s)

Gjalt-Jorn Peters

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

See Also

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

## End(Not run)
```

`fanova` *Flexible anova*

Description

This function is meant as a userfriendly wrapper to approximate the way analysis of variance is done in SPSS.

Usage

```
fanova(data,
       y,
       between = NULL,
       covar = NULL,
       plot = FALSE,
       levene = FALSE,
       digits = 2,
       contrast = NULL)
```

Arguments

`data`	The dataset containing the variables to analyse.
`y`	The dependent variable. For oneway anova, factorial anova, or ancova, this is the name of a variable in dataframe `data`. For repeated measures anova, this is a vector with the names of all variable names in dataframe `data`, e.g. `c('t0_value', 't1_value', 't2_value')`.
`between`	A vector with the variables name(s) of the between subjects factor(s).
`covar`	A vector with the variables name(s) of the covariate(s).
`plot`	Whether to produce a plot. Note that a plot is only produced for oneway and twoway anova and oneway repeated measures designs: if covariates or more than two between-subjects factors are specified, not plot is produced. For twoway anova designs, the second predictor is plotted as moderator (and the first predictor is plotted on the x axis).

`levene`	Whether to show Levene's test for equality of variances (using `car`'s `leveneTest` function but specifying `mean` as function to compute the center of each group).
`digits`	Number of digits (actually: decimals) to use when printing results. The p-value is printed with one extra digit.
`contrast`	This functionality has not been implemented yet.

Details

This wrapper uses `oneway` and `lm` and `lmer` in combination with `car`'s `Anova` function to conduct the analysis of variance.

Value

Mainly, this function prints its results, but it also returns them in an object containing three lists:

`input`	The arguments specified when calling the function
`intermediate`	Intermediat objects and values
`output`	The results such as the plot.

Author(s)

Gjalt-Jorn Peters

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

See Also

`regr` and `logRegr` for similar functions for linear and logistic regression and `oneway`, `lm`, `lmer` and `Anova` for the functions used behind the scenes.

Examples

```
### Oneway anova with a plot
fanova(dat=mtcars, y='mpg', between='cyl', plot=TRUE);

### Factorial anova
fanova(dat=mtcars, y='mpg', between=c('vs', 'am'), plot=TRUE);

### Ancova
fanova(dat=mtcars, y='mpg', between=c('vs', 'am'), covar='hp');

### Repeated measures anova; first generate datafile
dat <- mtcars[, c('am', 'drat', 'wt')];
names(dat) <- c('factor', 't0_dependentVar' ,'t1_dependentVar');
dat$factor <- factor(dat$factor);

### Then do the repeated measures anova
fanova(dat, y=c('t0_dependentVar' ,'t1_dependentVar'),
       between='factor', plot=TRUE);
```

`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 much does what the title says.

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`	The prefix, usually a type of opening parenthesis/bracket.
`suffix`	The suffix, usually a type of 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.

Author(s)

Gjalt-Jorn Peters

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

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);
```

`freq` *Frequency tables*

Description

Function to show frequencies in a manner similar to what SPSS' "FREQUENCIES" command does. Note that `frequency` is an alias for `freq`.

Usage

```
freq(vector, digits = 1, nsmall=1, transposed=FALSE,
     round=1, plot=FALSE, plotTheme = theme_bw())
frequencies(..., digits = 1, nsmall = 1,
            transposed = FALSE, round = 1,
            plot = FALSE, plotTheme = theme_bw())
```

Arguments

`vector`	A vector of values to compute frequencies for.
`digits`	Minimum number of significant digits to show in result.
`nsmall`	Minimum number of digits after the decimal point to show in the result.
`transposed`	Whether to transpose the results when printing them (this can be useful for blind users).
`round`	Number of digits to round the results to (can be used in conjunction with digits to determine format of results).
`plot`	If true, a histogram is shown of the variable.
`plotTheme`	The ggplot2 theme to use.
`...`	The variables of which to provide frequencies

Value

An object with several elements, the most notable of which is:

`dat`	A dataframe with the frequencies

For `frequencies`, these objects are in a list of their own.

Examples

```
### Create factor vector
ourFactor <- factor(mtcars$gear, levels=c(3,4,5),
                    labels=c("three", "four", "five"));
### Add some missing values
factorWithMissings <- ourFactor;
factorWithMissings[10] <- factorWithMissings[20] <- NA;

### Show frequencies
freq(ourFactor);
freq(factorWithMissings);

### ... Or for all of them at one
frequencies(ourFactor, factorWithMissings);
```

`fullFact` *fullFact*

Description

This function provides a userfriendly interface to a number of advanced factor analysis functions in the `psych` package.

Usage

```
fullFact(dat = NULL, items = NULL, rotate = "oblimin")
```

Arguments

`dat`	Datafile to analyse; if NULL, a pop-up is provided to select a file.
`items`	Which variables (items) to factor-analyse. If NULL, all are selected.
`rotate`	Which rotation to use (see `psych` package).

Value

The outcomes, which are printed to the screen unless assigned.

Author(s)

Gjalt-Jorn Peters

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

See Also

`fa.parallel`, `vss`

Examples

```
## Not run:
  ### Not run to save processing during package testing
  fullFact(attitude);

## End(Not run)
```

`genlog` *Generalized Logistic Analysis*

Description

This function implements the generalized logistic analysis introduced in Verboon & Peters (2017). This analysis fits a logistic function (i.e. a sigmoid) to a data series. This is useful when analysing single case designs. The function enables easy customization of the main plot elements and easy saving of the plot with anti-aliasing. `ggGenLogPlot` does most of the plotting, and can be useful when trying to figure out sensible starting and boundary/constraint values. `genlogCompleteStartValues` tries to compute sensible starting and boundary/constraint values based on the data.

Usage

```
genlog(data,
       timeVar = 1,
       yVar = 2,
       phaseVar = NULL,
       baselineMeasurements = NULL,
       yRange = NULL,
       startInflection = NULL,
       startBase = NULL,
       startTop = NULL,
       startGrowthRate = NULL,
       startV = 1,
       inflectionPointBounds = NULL,
       growthRateBounds = c(-2, 2),
       baseMargin = c(0, 3),
       topMargin = c(-3, 0),
       baseBounds = NULL,
       topBounds = NULL,
       vBounds = c(1, 1),
       changeDelay = 4,
       colors = list(bottomBound = viridis(4)[4],
                     topBound = viridis(40)[37],
                     curve = viridis(4)[3],
                     mid = viridis(4)[2],
                     intervention = viridis(4)[1],
                     points = "black",
                     outsideRange = "black"),
       alphas = list(outsideRange = .2,
                     bounds = 0,
                     points = .5,
                     mid = 0),
       theme = theme_minimal(),
       pointSize = 2,
       lineSize = 0.5,
       yBreaks = NULL,
       initialValuesLineType = "blank",
       curveSizeMultiplier = 2,
       showPlot = TRUE,
       plotLabs = NULL,
       outputFile = NULL,
       outputWidth = 16,
       outputHeight = 16,
       ggsaveParams = list(units = "cm",
                           dpi = 300,
                           type = "cairo"),
       maxiter = NULL)
```

Arguments

`data`	The dataframe containing the variables for the analysis.
`timeVar`	The name of the variable containing the measurement moments (or an index of measurement moments). An index can also be specified, and assumed to be 1 if omitted.
`yVar`	The name of the dependent variable. An index can also be specified, and assumed to be 2 if omitted.
`phaseVar`	The variable containing the phase of each measurement. Note that this normally should only have two possible values.
`baselineMeasurements`	If no phaseVar is specified, `baselineMeasurements` can be used to specify the number of baseline measurements, which is then used to construct the `phaseVar` dummy variable.
`yRange`	This can be used to manually specify the possible values that the dependent variable can take. If no `startBase` and `startTop` are specified, the range of the dependent variable is used instead.
`startInflection, startBase, startTop, startGrowthRate, startV`	The starting values used when estimating the sigmoid using `minpack.lm`'s `nlsLM` function. `startX` specifies the starting value to use for the measurement moment when the change is fastest (i.e. the slope of the sigmoid has the largest value); `startBase` and `startTop` specify the starting values to use for the base (floor) and top (ceiling), the plateaus of relative stability between which the sigmoid described the shift; `startGrowthRate` specifies the starting value for the growth rate; and `startV` specifies the starting value for the *v* parameter.
`inflectionPointBounds, growthRateBounds, baseMargin, topMargin, baseBounds, topBounds, vBounds`	These values specify constraints to respect when estimating the parameters of the sigmoid function using `minpack.lm`'s `nlsLM`. `changeInitiationBounds` specifies between which values the initiation of the shift must occur; `growthRateBounds` describes the bounds constraining the possible values for the growth rate; `baseBounds` and `topBounds` specify the constraints for possible values for the base (floor) and top (ceiling), the plateaus of relative stability between which the sigmoid described the shift; and if these are not specified, `baseMargin` and `topMargin` are used in combination with the range of the dependent variable to set these bounds (also see `yRange`); and finally, `vBounds` specifies the possible values that constrain the *v* parameter.
`changeDelay`	The number of measurements to add to the intervention moment when setting the initial value for the inflection point.
`colors`	The colors to use for the different plot elements.
`alphas`	The alpha values (transparency, or rather, 'obliqueness', with 0 indicating full transparency and 1 indicating full visibility) to use for the different plot elements.
`theme`	The theme to use in the plot.
`pointSize,lineSize`	The sizes of points and lines in the plot.

`yBreaks`	If `NULL`, the `pretty` function is used to estimate the best breaks for the Y axis. If a value is supplied, this value is used as the size of intervals between the (floored) minimum and (ceilinged) maximum of `yRange` (e.g. if `yBreaks` is 1, a break point every integer; if 2 and the minimum is 1 and the maximum is 7, breaks at 1, 3, 5 and 7; etc).
`initialValuesLineType`	The line type to use for the initial values; by default set to `"blank"` for `genlog`, to hide them, and to `"dashed"` for ggGenLogPlot.
`curveSizeMultiplier`	A multiplyer for the curve size compared to the other lines (e.g. specify '2' to have a curve of twice the size).
`showPlot`	Whether to show the plot or not.
`plotLabs`	A list with arguments to the `ggplot2` `labs` function, which can be used to conveniently set plot labels.
`outputFile`	If not `NULL`, the path and filename specifying where to save the plot.
`outputWidth, outputHeight`	The dimensions of the plot when saving it (in units specified in `ggsaveParams`).
`ggsaveParams`	The parameters to use when saving the plot, passed on to `ggsave`.
`maxiter`	The maximum number of iterations used by `nlsLM`.

Details

For details, see Verboon & Peters (2017).

Value

Mainly, this function prints its results, but it also returns them in an object containing three lists:

`input`	The arguments specified when calling the function
`intermediate`	Intermediat objects and values
`output`	The results such as the plot.

Author(s)

Peter Verboon & Gjalt-Jorn Peters (both at the Open University of the Netherlands)

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

References

Verboon, P. & Peters, G.-J. Y. (2018) Applying the generalised logistic model in single case designs: modelling treatment-induced shifts. *PsyArXiv* `https://doi.org/10.17605/osf.io/ad5eh`

See Also

`genlogFunction`

Examples

```
### Load dataset
data(Singh);

### Extract Jason
dat <- Singh[Singh$tier==1, ];

### Conduct piecewise regression analysis
genlog(dat,
       timeVar='time',
       yVar='score_physical',
       phaseVar='phase');
```

`genlogFunction` *Generalized Logistic Function*

Description

This is the core function of the generalized logistic analysis used in `genlog`.

Usage

```
genlogFunction(x, x0, Ab, At, B, v)
```

Arguments

`x`	A numeric vector with measurement moments or indices of measurement moments.
`x0`	A single numeric value specifying at which moment the curve is at its midpoint (when `v` = 1).
`Ab, At`	Respectively the lowest and highest possible values of the dependent variable.
`B`	The growth rate (curve steepness).
`v`	Um - Peter, wat is 'v' eigenlijk?

Details

For details, see Verboon & Peters (2017).

Author(s)

Peter Verboon (Open University of the Netherlands)

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

References

Verboon, P. & Peters, G.-J. Y. (2017) Applying the generalised logistic model in SCD to deal with ceiling effects. *PsyArXiv* http://INSERTLINK

See Also

`genlog`

Examples

```
time <- 1:20;
yVar <- genlogFunction(1:20, 10, 1, 7, 1, 1);
plot(time, yVar, type='l', xlab='time', ylab='y');
```

`ggBarChart` *Bar chart using ggplot*

Description

This function provides a simple interface to create a `ggplot` bar chart.

Usage

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

Arguments

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

Value

A `ggplot` plot is returned.

Author(s)

Gjalt-Jorn Peters

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

See Also

`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 = 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');
```

`ggConfidenceCurve` *Confidence Curves*

Description

Confidence curves are a way to show the confidence in an estimate computed from sample data. They are useful because they show all confidence levels simultaneously, thereby giving a good sense of the accuracy of the estimate, without forcing the researchers to make a more or less arbitrary choice for one confidence level.

Usage

```
ggConfidenceCurve(metric = "d",
                  value = NULL,
                  n = NULL,
                  conf.level = NULL,
                  wRange = c(0.05, 0.8),
                  curveSize = 1,
                  curveColor = "black",
                  confRange = c(1e-04, 0.9999),
                  confLines = c(0.5, 0.8, 0.95, 0.99),
                  widthLines = c(min(wRange), 0.1, 0.2, 0.3, max(wRange)),
                  lineColor = brewer.pal(9, 'Set1'),
                  lineSize = 1,
                  lineAlpha = .5,
                  xlab = metric,
                  steps = 1000,
                  theme = theme_bw(),
                  gradient=NULL,
                  gradientWidth=.01,
                  outputFile = NULL,
                  outputWidth = 16,
                  outputHeight = 16,
                  ggsaveParams = list(units='cm',
                                      dpi=300,
                                      type="cairo"))
```

Arguments

`metric`	The metric, currently only 'd' (Cohen's *d*) and 'r' (Pearson's *r*) are implemented.
`value`	The value for which to create the confidence curve plot.
`n`	The sample size for which to create the confidence curve plot. If `n` is specified, the y axis shows confidence levels (i.e. a conventional confidence curve is generated). If `n` is set to `NULL`, the y axis shows sample sizes. Either `n` or `conf.level` must be `NULL`.
`conf.level`	The confidence level for which to create the confidence curve plot. If `conf.level` is specified, the y axis shows sample sizes. If `conf.level` is set to `NULL`, the y axis shows confidence levels (i.e. a conventional confidence curve is generated). Either `n` or `conf.level` must be `NULL`.
`wRange`	The range of 'half-widths', or margins of error, to plot in the confidence curve plot if no sample size is specified (if `n=NULL`).
`curveSize`	The line size of the confidence curve line.
`curveColor`	The color of the confidence curve line.
`confRange`	The range of confidence levels to plot.
`confLines, widthLines`	If a traditional confidence curve is generated, lines can be added to indicate the metric values corresponding to the lower and upper confidence interval bounds. For an inverse confidence curve, lines can be added to inficate the metric values and sample sizes corresponding to specific margins of error (or 'half-widths').
`lineColor`	If confidence or 'interval width lines' lines are added (see `confLines`), this is the color in which they are drawn. Specify a vector (e.g. `brewer.pal(9, 'Set1')`) to have the colors drawn in different colors for each confidence level or width.
`lineSize`	If confidence lines or 'interval width lines' are added (see `confLines` and `widthLines`), these arguments specify the color and size in which they are drawn.
`lineAlpha`	The alpha value (transparency) of the confidence lines or 'interval width lines'.
`xlab`	The label on the x axis.
`steps`	The number of steps to use when generating the data for the confidence curves' more steps yield prettier, smoother curves, but take more time.
`theme`	The `ggplot` theme to use.
`gradient`	Whether to use a gradient as background to make the confidence more explicit. This is experimental and pretty influential in terms of how the plot looks. The default gradient, used when passing `TRUE`, uses black as background color when the confidence is 0 percent, and white for 100 percent. If two colors are specified, these are used instead.
`gradientWidth`	If using a gradient, the width of the `geom_tile` geoms used to create the gradient.
`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.

Value

A `ggplot2` plot.

Author(s)

Gjalt-Jorn Peters

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

References

Bender, R., Berg, G., & Zeeb, H. (2005). Tutorial: Using confidence curves in medical research. *Biometrical Journal*, 47(2), 237-247. http://doi.org/10.1002/bimj.200410104

Birnbaum, A. (1961). Confidence curves: An omnibus technique for estimation and testing statistical hypotheses. *Journal of the American Statistical Association*, 56(294), 246-249. http://doi.org/10.1080/01621459.1961.104

See Also

`cohensdCI`, `pwr.cohensdCI`, `confIntR`, `pwr.confIntR`

Examples

```
ggConfidenceCurve(metric='d', value = .5, n = 128);
ggConfidenceCurve(metric='d', value = .5, conf.level = .95);
```

`ggDiamondLayer` *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

```
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",
               ...)
diamondCoordinates(values,
```

```
                     otherAxisValue = 1,
                     direction = "horizontal",
                     autoSize = NULL,
                     fixedSize = 0.15)
varsToDiamondPlotDf(dat, items = NULL,
                     labels = NULL, decreasing = NULL,
                     conf.level = 0.95)
rawDataDiamondLayer(dat, items = NULL,
                     itemOrder = 1:length(items),
                     dataAlpha = 0.1, dataColor = "#444444",
                     jitterWidth = 0.5, jitterHeight = 0.4,
                     size = 3, ...)
```

Arguments

`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.
`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.
`...`	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`.

`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.
`items`	The items from the dataframe to include in the diamondplot or dataframe.
`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.
`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.

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)
```

`ggEasyPlots` *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

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

ggEasyBar(data,
          items = NULL,
          labels = NULL,
          sortByMean = TRUE,
          xlab = NULL,
          ylab = NULL,
          scale_fill_function =
            scale_fill_viridis(discrete = TRUE,
                               guide = guide_legend(title = NULL,
                                                    nrow=1)),
          fontColor = "white",
          fontSize = 2,
          labelMinPercentage = 1,
          showInLegend = "both",
          biAxisLabels = 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.
`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.
`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”)));
```

`ggNNC` *Visualising Numbers Needed for Change*

Description

These functions can be used to visualise Numbers Needed for Change. `erDataSeq` is a helper function to generate an Event Rate Data Sequence, and it uses `convert.threshold.to.er` and `convert.er.to.threshold` to convert thresholds to event rates and vice versa.

Usage

```
erDataSeq(er = NULL, threshold = NULL,
          mean = NULL, sd = NULL,
          eventIfHigher = TRUE,
          pRange = c(1e-06, 0.99999),
          xStep = 0.01)

ggNNC(cerDataSeq, d = NULL, eventDesirable = TRUE,
      r = 1, xlab = "Continuous outcome",
      plotTitle = c("Numbers Needed for Change = ", ""),
      theme = theme_bw(), lineSize = 1,
      cerColor = "#EBF2F8", eerColor = "#172F47",
      cerLineColor = "#888888", eerLineColor = "#000000",
      dArrowColor = "#000000", cerAlpha = 0.66,
      eerAlpha = 0.66, xLim = NULL,
      xLimAutoDensityTolerance = 0.001,
      showLegend = TRUE, verticalLineColor = "#172F47",
      desirableColor = "#00FF00", desirableAlpha = 0.2,
      undesirableColor = "#FF0000", undesirableAlpha = 0.2,
      desirableTextColor = "#009900",
      undesirableTextColor = "#990000",
      dArrowDistance = 0.04 * max(cerDataSeq$density),
      dLabelDistance = 0.08 * max(cerDataSeq$density))

convert.threshold.to.er(threshold, mean, sd,
                        eventIfHigher = TRUE,
                        pdist = pnorm)

convert.er.to.threshold(er, mean, sd,
                        eventIfHigher = TRUE,
                        qdist = qnorm)
```

Arguments

`er`	Event rate to visualise (or convert).

`threshold`	If the event rate is not available, a threshold value can be specified instead, which is then used in conjunction with the mean (`mean`) and standard deviation (`sd`) and assuming a normal distribution to compute the event rate.
`mean`	The mean of the control group distribution.
`sd`	The standard deviation (of the control distribution, but assumed to be the same for both distributions).
`eventIfHigher`	Whether scores above or below the threshold are considered 'an event'.
`pRange`	The range of probabilities for which to so the distribution.
`xStep`	Precision of the drawn distribution; higher values mean lower precision/granularity/resolution.
`cerDataSeq`	The `cerDataSeq` object.
`d`	The value of Cohen's *d*.
`eventDesirable`	Whether an event is desirable or undesirable.
`r`	The correlation between the determinant and behavior (for mediated NNC's).
`xlab`	The label to display for the X axis.
`plotTitle`	The title of the plot; either one character value, this value if used; if two, they are considered a prefix and suffix to be pre/appended to the NNC value.
`theme`	The theme to use for the plot.
`lineSize`	The thickness of the lines in the plot.
`cerColor`	The color to use for the event rate portion of the control group distribution.
`eerColor`	The color to use for the event rate portion of the experimental group distribution.
`cerLineColor`	The line color to use for the control group distribution.
`eerLineColor`	The line color to use for the experimental group distribution.
`dArrowColor`	The color of the arrow to show the effect size.
`cerAlpha`	The alpha value (transparency) to use for the control group distribution.
`eerAlpha`	The alpha value (transparency) to use for the control group distribution.
`xLim`	This can be used to manually specify the limits for the X axis; if `NULL`, sensible limits will be derived using `xLimAutoDensityTolerance`.
`xLimAutoDensityTolerance`	If `xLim` is `NULL`, the limits will be set where the density falls below this proportion of its maximum value.
`showLegend`	Whether to show the legend (only if showing two distributions).
`verticalLineColor`	The color of the vertical line used to indicate the threshold.
`desirableColor`	The color for the desirable portion of the X axis.
`desirableAlpha`	The alpha for the desirable portion of the X axis.
`undesirableColor`	The color for the undesirable portion of the X axis.
`undesirableAlpha`	The color for the undesirable portion of the X axis.

`desirableTextColor`	The color for the text to indicate the desirable portion of the X axis.
`undesirableTextColor`	The color for the text to indicate the undesirable portion of the X axis.
`dArrowDistance`	The distance of the effect size arrow from the top of the distributions.
`dLabelDistance`	The distance of the effect size label from the top of the distributions.
`pdist, qdist`	Distributions to use when converting thresholds to event rates and vice versa; defaults to the normal distribution.

Details

These functions are used by `nnc` to show the distributions, and event rates. They probably won't be used much on their own.

Value

`erDataSeq` returns a data sequence; `ggNNC` a `ggplot`.

Author(s)

Gjalt-Jorn Peters & Stefan Gruijters

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

References

Gruijters, S. L. K., & Peters, G.-J. Y. (2017). Introducing the Numbers Needed for Change (NNC): A practical measure of effect size for intervention research.

See Also

`nnc`

Examples

```
### Show distribution for an event rate value of 125
ggNNC(erDataSeq(threshold=125, mean=90, sd=30));

### If the event occurs under the threshold instead of
### above it
ggNNC(erDataSeq(threshold=125, mean=90, sd=30,
      eventIfHigher = FALSE));

### ... And for undesirable events (note how
### desirability is an argument for ggNNC, whereas
### whether an event occurs 'above' or 'below' the
### threshold is an argument for erDataSeq):
ggNNC(erDataSeq(threshold=125, mean=90, sd=30,
                eventIfHigher = FALSE),
      eventDesirable = FALSE);
```

```
### Show event rate for both experimental and
### control conditions, and show the numbers
### needed for change
ggNNC(erDataSeq(threshold=125, mean=90, sd=30), d=.5);

### Illustration of how even with very large effect
### sizes, if the control event rate is very high,
### you'll still need a high number of NNC
ggNNC(erDataSeq(er=.9), d=1);
```

`ggPie` *A ggplot pie chart*

Description

THis function creates a pie chart. Note that these are generally quite strongly advised against, as people are not good at interpreting relative frequencies on the basis of pie charts.

Usage

```
ggPie(vector,
      scale_fill = scale_fill_viridis(discrete=TRUE))
```

Arguments

`vector`	The vector (best to pass a factor).
`scale_fill`	The ggplot scale fill function to use for the colors.

Value

A ggplot pie chart.

Note

This function is very strongly based on the Mathematical Coffee post at http://mathematicalcoffee.blogspot.com/2014/06/ggpi
pie-graphs-in-ggplot2.html.

Author(s)

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

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

Examples

```
ggPie(mtcars$cyl);
```

`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=.95,
                 diamonds.alpha=1,
                 na.rm = TRUE,
                 barHeight = 0.4,
                 conf.steps = seq(from=0.001, to=.999, by=.001),
                 scale_color = viridis(option="magma", 2, begin=0, end=.5),
                 scale_fill = viridis(option="magma", 2, begin=0, end=.5),
                 linetype=1,
                 theme = theme_bw(),
                 returnPlotOnly = TRUE)
```

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).
`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.

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 = 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 `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);
```

`importLimeSurveyData` *importLimeSurveyData*

Description

This function can be used to import files exported by LimeSurvey, a powerful Open Source online survey application that can be used for, for example, psychological experiments and other research.

Usage

```
importLimeSurveyData(datafile = NULL,
  dataPath = NULL,
  datafileRegEx = NULL,
  scriptfile = NULL,
  limeSurveyRegEx.varNames =
    "names\\(data\\)\\[\\d*\\] <- ",
  limeSurveyRegEx.toChar =
    "data\\[, \\d*\\] <- as.character\\(data\\[, \\d*\\]\\)",
```

```
  limeSurveyRegEx.varLabels =
    "attributes\\(data\\)\$variable.labels\\[\\d*\\] <- \".*\"",
  limeSurveyRegEx.toFactor =
    paste0("data\\[, \\d*\\] <- factor\\(data\\[, \\d*\\], ",
           "levels=c\\(.*\\),.*labels=c\\(.*\\)\\)"),
  limeSurveyRegEx.varNameSanitizing =
    list(list(pattern = "#", replacement = "."),
         list(pattern = "\$", replacement = ".")),
  setVarNames = TRUE,
  setLabels = TRUE,
  convertToCharacter = FALSE,
  convertToFactor = FALSE,
  categoricalQuestions = NULL,
  massConvertToNumeric = TRUE,
  dataHasVarNames = TRUE,
  encoding = "NULL",
  dataEncoding = "unknown",
  scriptEncoding = "ASCII")
```

Arguments

`datafile`	The path and filename of the file containing the data (comma separated values).
`dataPath, datafileRegEx`	Path containing datafiles: this can be used to read multiple datafiles, if the data is split between those. This is useful when downloading the entire datafile isn't possible because of server restrictions, for example when the processing time for the script in LimeSurvey that generates the datafiles is limited. In that case, the data can be downloaded in portions, and specifying a path here enables reading all datafiles in one go. Use the regular expression to indicate which files in the path should be read.
`scriptfile`	The path and filename of the file containing the R script to import the data.
`limeSurveyRegEx.varNames`	The regular expression used to extract the variable names from the script file. The first regex expression (i.e. the first expression between parentheses) will be extracted as variable name.
`limeSurveyRegEx.toChar`	The regular expression to detect the lines in the import script where variables are converted to the character type.
`limeSurveyRegEx.varLabels`	The regular expression used to detect the lines in the import script where variable labels are set.
`limeSurveyRegEx.toFactor`	The regular expression used to detect the lines in the import script where vectors are converted to factors.
`limeSurveyRegEx.varNameSanitizing`	A list of regular expression patterns and their replacements to sanitize the variable names (e.g. replace hashes/pound signs ('#') by something that is not considered the comment symbol by R).

`setVarNames, setLabels, convertToCharacter, convertToFactor`	Whether to set variable names or labels, or convert to character or factor, using the code isolated using the specified regular expression.
`categoricalQuestions`	Which variables (specified using LimeSurvey variable names) are considered categorical questions; for these, the script to convert the variables to factors, as extracted from the LimeSurvey import file, is applied.
`massConvertToNumeric`	Whether to convert all variables to numeric using `massConvertToNumeric`.
`dataHasVarNames`	Whether the variable names are included as header (first line) in the comma separated values file (data file).
`encoding, dataEncoding, scriptEncoding`	The encoding of the files; `encoding` overrides `dataEncoding` and `scriptEncoding`, and so can be used to specify the same encoding for both.

Details

This function was intended to make importing data from LimeSurvey a bit easier. The default settings used by LimeSurvey are not always convenient, and this function provides a bit more control.

Value

The dataframe.

Author(s)

Gjalt-Jorn Peters

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

See Also

`getData`

Examples

```
## Not run:
### Of course, you need valid LimeSurvey files. This is an example of
### what you'd do if you have them, assuming you specified that path
### containing the data in 'dataPath', the name of the datafile in
### 'dataFileName', the name of the script file in 'dataLoadScriptName',
### and that you only want variables 'informedConsent', 'gender', 'hasJob',
### 'currentEducation', 'prevEducation', and 'country' to be converted to
### factors.
dat <- importLimeSurveyData(datafile = file.path(dataPath, dataFileName),
                            scriptfile = file.path(dataPath, dataLoadScriptName),
                            categoricalQuestions = c('informedConsent',
                                                     'gender',
                                                     'hasJob',
                                                     'currentEducation',
```

```
                                                       'prevEducation',
                                                       'country'));

## End(Not run)
```

`invertItems` *invertItems*

Description

Inverts items (as in, in a questionnaire), by calling `invertItem` on all relevant items.

Usage

```
invertItems(dat, items = NULL, ...)
```

Arguments

`dat`	The dataframe containing the variables to invert.
`items`	The names or indices of the variables to invert. If not supplied (i.e. NULL), all variables in the dataframe will be inverted.
`...`	Arguments (parameters) passed on to data.frame when recreating that after having used lapply.

Value

The dataframe with the specified items inverted.

Author(s)

Gjalt-Jorn Peters

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

See Also

`invertItem`

Examples

```
invertItems(mtcars, c('cyl'));
```

`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` *is.nr*

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.

Author(s)

Gjalt-Jorn Peters

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

Examples

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

`isTrue` *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.

Author(s)

Gjalt-Jorn Peters

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

Examples

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

`itemInspection` *itemInspection*

Description

Function to generate a PDF with four panels per page, showing some basic item characteristics.

Usage

```
itemInspection(dat, items,
               docTitle = "Scale inspection", docAuthor = "Author",
               pdfLaTexPath, rnwPath, filename="itemInspection",
               convertFactors = TRUE, digits=4)
```

Arguments

`dat`	Dataframe containing the items of the relevant scale
`items`	Either a character vector with the itemnames, or, if the items are organised in scales, a list of character vectors with the items in each scale.
`docTitle`	Title to use when generating the PDF.
`docAuthor`	Author(s) to include when generating the PDF.
`pdfLaTexPath`	The path to PdfLaTex. This file is part of a LaTeX installation that creates a pdf out of a .tex file. In Windows, you can download (portable) MikTex from http://miktex.org/portable. You then decide yourself where to install MikTex; pdflatex will end up in a subfolder 'miktex\bin', so if you installed MikTex in, for example, 'C:\Program Files\MikTex', the total path becomes 'C:\Program Files\MikTex\miktex\bin'. Note that R uses slashes instead of backslashes to separate folders, so in this example, pdfLaTexPath should be 'C:/Program Files/MikTex/miktex/bin' In MacOS, you can install MacTex from http://tug.org/mactex/ By default, pdflatex ends up in folder '/user/texbin', which is what pdfLaTexPath should be in that default case. In Ubuntu, you can install TexLive base by using your package manager to install texlive-latex-base, or using the terminal: 'sudo apt-get install texlive-latex-base' In ubuntu, by default pdflatex ends un in folder '/usr/bin', which is what pdfLaTexPath should be in that default case.
`rnwPath`	The path where the temporary files and the resulting PDF should be stored.
`filename`	The filename to use to save the pdf.
`convertFactors`	Whether to convert factors to numeric vectors for the analysis.
`digits`	The number of digits to use in the tables.

Value

This function returns nothing; it just generates a PDF.

Examples

```
## Not run:
itemInspection(mtcars, items=c('disp', 'hp', 'drat'), pdfLaTexPath="valid/path/here");

## End(Not run)
```

`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, ...)
```

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` is used to generate an MD5-hash from `Sys.time`.
`...`	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

```
## Not run:
knitFig(ggProportionPlot(mtcars$cyl))

## End(Not run)
```

---

logRegr *Userfriendly wrapper to do logistic regression in R*

Description

This function is meant as a userfriendly wrapper to approximate the way logistic regression is done in SPSS.

Usage

```
logRegr(formula,
        data = NULL,
        conf.level = 0.95,
        digits = 2,
        pvalueDigits = 3,
        crossTabs = TRUE,
        plot = FALSE,
        collinearity = FALSE,
        env = parent.frame(),
        predictionColor = viridis(3)[3],
        predictionAlpha = 0.5,
        predictionSize = 2,
        dataColor = viridis(3)[1],
        dataAlpha = 0.33,
        dataSize = 2,
        observedMeansColor = viridis(3)[2],
        binObservedMeans = 7,
        observedMeansSize = 2,
        observedMeansWidth = NULL,
        observedMeansAlpha = 0.5,
        theme = theme_bw())
```

Arguments

`formula`	The formula, specified in the same way as for `glm` (which is used for the actual analysis).

`data`	Optionally, a dataset containing the variables in the formula (if not specified, the variables must exist in the environment specified in `env`.
`conf.level`	The confidence level for the confidence intervals.
`digits`	The number of digits used when printing the results.
`pvalueDigits`	The number of digits used when printing the p-values.
`crossTabs`	Whether to show cross tabulations of the correct predictions for the null model and the tested model, as well as the percentage of correct predictions.
`plot`	Whether to display the plot.
`collinearity`	Whether to show collinearity diagnostics.
`env`	If no dataframe is specified in `data`, use this argument to specify the environment holding the variables in the formula.
`predictionColor, dataColor, observedMeansColor`	The color of, respectively, the line and confidence interval showing the prediction; the points representing the observed data points; and the means based on the observed data.
`predictionAlpha, dataAlpha, observedMeansAlpha`	The alpha of, respectively, the confidence interval of the prediction; the points representing the observed data points; and the means based on the observed data (set to 0 to hide an element).
`predictionSize, dataSize, observedMeansSize`	The size of, respectively, the line of the prediction; the points representing the observed data points; and the means based on the observed data (set to 0 to hide an element).
`binObservedMeans`	Whether to bin the observed means; either FALSE or a single numeric value specifying the number of bins.
`observedMeansWidth`	The width of the lines of the observed means. If not specified (i.e. `NULL`), this is computed automatically and set to the length of the shortest interval between two successive points in the predictor data series (found using `findShortestInterval`.
`theme`	The theme used to display the plot.

Details

This function

Value

Mainly, this function prints its results, but it also returns them in an object containing three lists:

`input`	The arguments specified when calling the function
`intermediate`	Intermediat objects and values
`output`	The results, such as the plot, the cross tables, and the coefficients.

Author(s)

Ron Pat-El & Gjalt-Jorn Peters (both while at the Open University of the Netherlands)

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

See Also

`regr` and `fanova` for similar functions for linear regression and analysis of variance and `glm` for the regular interface for logistic regression.

Examples

```
### Simplest way to call logRegr
logRegr(data=mtcars, formula = vs ~ mpg);

### Also ordering a plot
logRegr(data=mtcars, formula = vs ~ mpg, plot=TRUE);

### Only use five bins
logRegr(data=mtcars, formula = vs ~ mpg, plot=TRUE, binObservedMeans=5);
```

`meanDiff` *meanDiff*

Description

The meanDiff function compares the means between two groups. It computes Cohen's d, the unbiased estimate of Cohen's d (Hedges' g), and performs a t-test. It also shows the achieved power, and, more usefully, the power to detect small, medium, and large effects.

Usage

```
meanDiff(x, y=NULL, paired = FALSE, r.prepost = NULL, var.equal = "test",
         conf.level = .95, plot = FALSE, digits = 2, envir = parent.frame())
```

Arguments

`x`	Dichotomous factor: variable 1; can also be a formula of the form y ~ x, where x must be a factor with two levels (i.e. dichotomous).
`y`	Numeric vector: variable 2; can be empty if x is a formula.
`paired`	Boolean; are x & y independent or dependent? Note that if x & y are dependent, they need to have the same length.
`r.prepost`	Correlation between the pre- and post-test in the case of a paired samples t-test. This is required to compute Cohen's d using the formula on page 29 of Borenstein et al. (2009). If NULL, the correlation is simply computed from the provided scores (but of course it will then be lower if these is an effect - this will lead to an underestimate of the within-groups variance, and therefore, of

	the standard error of Cohen's d, and therefore, to confidence intervals that are too narrow (too liberal). Also, of course, when using this data to compute the within-groups correlation, random variations will also impact that correlation, which means that confidence intervals may in practice deviate from the null hypothesis significance testing p-value in either direction (i.e. the p-value may indicate a significant association while the confidence interval contains 0, or the other way around). Therefore, if the test-retest correlation of the relevant measure is known, please provide this here to enable computation of accurate confidence intervals.
`var.equal`	String; only relevant if x & y are independent; can be "test" (default; test whether x & y have different variances), "no" (assume x & y have different variances; see the Warning below!), or "yes" (assume x & y have the same variance)
`conf.level`	Confidence of confidence intervals you want.
`plot`	Whether to print a dlvPlot.
`digits`	With what precision you want the results to print.
`envir`	The environment where to search for the variables (useful when calling meanDiff from a function where the vectors are defined in that functions environment).

Details

This function uses the formulae from Borenstein, Hedges, Higgins & Rothstein (2009) (pages 25-32).

Value

An object is returned with the following elements:

`variables`	Input variables
`groups`	Levels of the x variable, the dichotomous factor
`ci.confidence`	Confidence of confidence intervals
`digits`	Number of digits for output
`x`	Values of dependent variable in first group
`y`	Values of dependent variable in second group
`type`	Type of t-test (independent or dependent, equal variances or not)
`n`	Sample sizes of the two groups
`mean`	Means of the two groups
`sd`	Standard deviations of the two groups
`objects`	Objects used; the t-test and optionally the test for equal variances
`variance`	Variance of the difference score
`meanDiff`	Difference between the means
`meanDiff.d`	Cohen's d
`meanDiff.d.var`	Variance of Cohen's d

`meanDiff.d.se`	Standard error of Cohen's d
`meanDiff.J`	Correction for Cohen's d to get to the unbiased Hedges g
`power`	Achieved power with current effect size and sample size
`power.small`	Power to detect small effects with current sample size
`power.medium`	Power to detect medium effects with current sample size
`power.largel`	Power to detect large effects with current sample size
`meanDiff.g`	Hedges' g
`meanDiff.g.var`	Variance of Hedges' g
`meanDiff.g.se`	Standard error of Hedges' g
`ci.usedZ`	Z value used to compute confidence intervals
`meanDiff.d.ci.lower`	Lower bound of confidence interval around Cohen's d
`meanDiff.d.ci.upper`	Upper bound of confidence interval around Cohen's d
`meanDiff.g.ci.lower`	Lower bound of confidence interval around Hedges' g
`meanDiff.g.ci.upper`	Upper bound of confidence interval around Hedges' g
`meanDiff.ci.lower`	Lower bound of confidence interval around raw mean
`meanDiff.ci.upper`	Upper bound of confidence interval around raw mean
`t`	Student t value for Null Hypothesis Significance Testing
`df`	Degrees of freedom for t value
`p`	p-value corresponding to t value

Warning

Note that when different variances are assumed for the t-test (i.e. the null-hypothesis test), the values of Cohen's d are still based on the assumption that the variance is equal. In this case, the confidence interval might, for example, not contain zero even though the NHST has a non-significant p-value (the reverse can probably happen, too).

References

Borenstein, M., Hedges, L. V., Higgins, J. P., & Rothstein, H. R. (2011). Introduction to meta-analysis. John Wiley & Sons.

Examples

```
### Create simple dataset
dat <- PlantGrowth[1:20,];
### Remove third level from group factor
dat$group <- factor(dat$group);
### Compute mean difference and show it
```

```
meanDiff(dat$weight ~ dat$group);

### Look at second treatment
dat <- rbind(PlantGrowth[1:10,], PlantGrowth[21:30,]);
### Remove third level from group factor
dat$group <- factor(dat$group);
### Compute mean difference and show it
meanDiff(x=dat$group, y=dat$weight);
```

`meanDiff.multi` *meanDiff.multi*

Description

The meanDiff.multi function compares many means for many groups. It presents the results in a dataframe summarizing all relevant information, and produces plot showing the confidence intervals for the effect sizes for each predictor (i.e. dichotomous variable). Like meanDiff, it computes Cohen's d, the unbiased estimate of Cohen's d (Hedges' g), and performs a t-test. It also shows the achieved power, and, more usefully, the power to detect small, medium, and large effects.

Usage

```
meanDiff.multi(dat, y, x=NULL, var.equal = "yes", conf.level = .95,
               digits = 2, orientation = "vertical",
               zeroLineColor = "grey", zeroLineSize = 1.2,
               envir = parent.frame())
```

Arguments

`dat`	The dataframe containing the variables involved in the mean tests.
`y`	Character vector containing the list of interval variables to include in the tests.
`x`	Character vector containing the list of the dichotomous variables to include in the tests. If x is empty, paired samples t-tests will be conducted.
`var.equal`	String; only relevant if x & y are independent; can be "test" (default; test whether x & y have different variances), "no" (assume x & y have different variances; see the Warning below!), or "yes" (assume x & y have the same variance)
`conf.level`	Confidence of confidence intervals you want.
`digits`	With what precision you want the results to print.
`orientation`	Whether to plot the effect size confidence intervals vertically (like a forest plot, the default) or horizontally.
`zeroLineColor`	Color of the horizontal line at an effect size of 0 (set to 'white' to not display the line; also adjust the size to 0 then).
`zeroLineSize`	Size of the horizontal line at an effect size of 0 (set to 0 to not display the line; also adjust the color to 'white' then).
`envir`	The environment where to search for the variables (useful when calling meanDiff from a function where the vectors are defined in that functions environment).

Details

This function uses the meanDiff function, which uses the formulae from Borenstein, Hedges, Higgins & Rothstein (2009) (pages 25-32).

Value

An object is returned with the following elements:

`results.raw`	Objects returned by the calls to meanDiff.
`plots`	For every comparison, a plot with the datapoints, means, and confidence intervals in the two groups.
`results.compiled`	Dataframe with the most important results from each comparison.
`plots.compiled`	For every dichotomous (x) variable, a plot with the confidence interval for the effect size of each dependent (y) variable.
`input`	The arguments with which the function was called.

Warning

Note that when different variances are assumed for the t-test (i.e. the null-hypothesis test), the values of Cohen's d are still based on the assumption that the variance is equal. In this case, the confidence interval might, for example, not contain zero even though the NHST has a non-significant p-value (the reverse can probably happen, too).

References

Borenstein, M., Hedges, L. V., Higgins, J. P., & Rothstein, H. R. (2011). Introduction to meta-analysis. John Wiley & Sons.

Examples

```
### Create simple dataset
dat <- data.frame(x1 = factor(rep(c(0,1), 20)),
                  x2 = factor(c(rep(0, 20), rep(1, 20))),
                  y=rep(c(4,5), 20) + rnorm(40));
### Compute mean difference and show it
meanDiff.multi(dat, x=c('x1', 'x2'), y='y', var.equal="yes");
```

`meansComparisonDiamondPlot`

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

```
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 = brewer.pal(8, "Set1"),
                           alpha = 0.33,
                           jitterWidth = 0.5, jitterHeight = 0.4,
                           xlab = "Scores and means",
                           ylab = NULL,
                           theme = theme_bw(),
                           showLegend = TRUE,
                           lineSize = 1,
                           xbreaks = "auto",
                           outputFile = NULL,
                           outputWidth = 10,
                           outputHeight = 10,
                           ggsaveParams = list(units='cm',
                                               dpi=300,
                                               type="cairo"),
                           ...)

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 = brewer.pal(8, "Set1"),
```

```
                    associationsColor = "grey",
                    alpha = 0.33,
                    jitterWidth = 0.5, jitterHeight = 0.4,
                    xlab = c("Scores and means", "Effect size estimates"),
                    ylab = c(NULL, NULL),
                    theme = theme_bw(),
                    showLegend = TRUE,
                    lineSize = 1,
                    drawPlot = TRUE,
                    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).
`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 `comparisonBy` variable.
`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.
`theme`	The theme to use for the plots.
`showLegend`	Whether to show the legend (which color represents which subgroup/sample).
`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`.

Details

These functions are explained in Peters (2017).

Value

Diamond plots: a `ggplot` by `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`, `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));
```

`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(dat, items = NULL,
                 labels = NULL,
                 decreasing = NULL,
                 conf.level = 0.95,
                 showData = TRUE,
                 dataAlpha = .1,
                 dataSize = 3,
                 dataColor = "#444444",
                 diamondColors = NULL,
                 jitterWidth = 0.5,
                 jitterHeight = 0.4,
                 returnLayerOnly = FALSE,
                 xlab = "Scores and means",
                 ylab = NULL,
                 theme = theme_bw(),
                 xbreaks="auto",
                 outputFile = NULL,
                 outputWidth = 10,
                 outputHeight = 10,
```

```
ggsaveParams = list(units='cm',
                    dpi=300,
                    type="cairo"),
...)
```

Arguments

`dat`	The dataframe containing the variables (`items`) to show in the diamond plot.
`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`, `factorLoadingDiamondCIplot`, `ggDiamondLayer`

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,
              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.
`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`, `freq`

Examples

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

`nnc` *Numbers Needed for Change*

Description

This function computes the Numbers Needed for Change, and shows a visualisation to illustrate them. `nnt` is an alias for `nnc`.

Usage

```
nnc(d = NULL, cer = NULL, r = 1, n = NULL,
    threshold = NULL, mean = 0, sd = 1,
    poweredFor = NULL, thresholdSensitivity = NULL,
    eventDesirable = TRUE, eventIfHigher = TRUE,
    conf.level=.95,
    d.ci = NULL, cer.ci = NULL, r.ci = NULL,
    d.n = NULL, cer.n = NULL, r.n = NULL,
    plot = TRUE, returnPlot = TRUE, silent = FALSE)
```

Arguments

`d`	The value of Cohen's *d*.
`cer`	The Control Event Rate.
`r`	The correlation between the determinant and behavior (for mediated Numbers Needed for Change).
`n`	The sample size.
`threshold`	If the event rate is not available, a threshold value can be specified instead, which is then used in conjunction with the mean (`mean`) and standard deviation (`sd`) and assuming a normal distribution to compute the event rate.
`mean`	The mean value, used to draw the plot, or, if no CER is provided but instead the threshold value, to compute the CER.
`sd`	The standard deviation, used to draw the plot (and to compute the CER if a threshold value is supplied instead of the CER).
`poweredFor`	The Cohen's *d* value for which the study was powered. This expected Cohen's *d* value can be used to compute the threshold, which then in turn is used to compute the CER. To use this approach, also specify the mean and the standard deviation.
`thresholdSensitivity`	This argument can be used to provide a vector of potential threshold values, each of which is used to compute an NNC. This enables easy inspection of whether the value chosen as threshold matters much for the NNC.
`eventDesirable`	Whether an event is desirable or undesirable.
`eventIfHigher`	Whether scores above or below the threshold are considered 'an event'.
`conf.level`	The confidence level of the confidence interval.
`d.ci`	Instead of providing a point estimate for Cohen's *d*, a confidence interval can be provided.
`cer.ci`	Instead of providing a point estimate for the Control Event Rate, a confidence interval can be provided.
`r.ci`	Instead of providing a point estimate for the correlation, a confidence interval can be provided.
`d.n`	In addition to providing a point estimate for Cohen's *d*, a sample size can be provided; if it is, the confidence interval is computed.

`cer.n`	In addition to providing a point estimate for the Control Event Rate, a sample size can be provided; if it is, the confidence interval is computed.
`r.n`	In addition to providing a point estimate for the correlation, a sample size can be provided; if it is, the confidence interval is computed.
`plot`	Whether to generate and show the plot.
`returnPlot`	Whether to return the plot (as an attribute), or to only display it.
`silent`	Whether to suppress notifications.

Details

This function computes the Numbers Needed for Change. See Gruijters & Peters (2017) for details.

Value

The Numbers Needed for Change (NNC), potentially with a plot visualising the NNC in an attribute.

Author(s)

Gjalt-Jorn Peters & Stefan Gruijters

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

References

Gruijters, S. L. K., & Peters, G.-J. Y. (2017). Introducing the Numbers Needed for Change (NNC): A practical measure of effect size for intervention research.

Examples

```
### Simple example
nnc(d=.4, cer=.3);

### Or for a scenario where events are undesirable, and the
### intervention effective (therefore having a negative value for d):
nnc(d=-.4, cer=.3, eventDesirable=FALSE);
```

`normalityAssessment` *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.

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.

Usage

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

Arguments

`sampleVector`	Numeric vector containing the sample data.
`samples`	Number of samples to use when constructing sampling distribution.
`digits`	Number of digits to use when printing results.
`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.
`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).

`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.
`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.
`conf.level`	Confidence of confidence intervals.
`plots`	Whether to display plots.
`qqCI`	Whether to show the confidence interval for the QQ plot.
`labelOutliers`	Whether to label outliers with their row number in the box plot.
`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!
`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.
`...`	Anything else is passed on my sampingDistribution to normalityAssessment.

Details

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)
```

`oddsratio` *oddsratio*

Description

The oddsratio function simply computes a point estimate and confidence interval for an odds ratio.

Usage

```
oddsratio(x, y = NULL, conf.level = .95, digits=2)
```

Arguments

`x`	x can be either a table (then y can be NULL) or a factor.
`y`	If x is a factor, y also has to be a factor; x and y are then used to create the crosstable.
`conf.level`	The confidence level of the confidence interval.
`digits`	Number of digits to round output to

Value

The oddsratio function returns an object with the input and output.

`input`	List with input arguments
`or`	Point estimate for odds ratio
`or.ci`	Confidence interval for odds ratio

Examples

```
### Generate two factor vectors
treatment <- factor(c(rep(0, 33), rep(1, 45), rep(0, 63), rep(1, 21)),
                    levels=c(0,1), labels=c("no", "yes"));
survival <- factor(c(rep(0, 78), rep(1, 84)),
                   levels=c(0, 1), labels=c("no", "yes"));

### Compute and display odds ratio
oddsratio(treatment, survival);

### Or present a table
oddsratio(table(treatment, survival));
```

`omegaSqDist` *The distribution of Omega Squared*

Description

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

Usage

```
domegaSq(x, df1, df2, populationOmegaSq = 0)
pomegaSq(q, df1, df2, populationOmegaSq = 0, lower.tail = TRUE)
qomegaSq(p, df1, df2, populationOmegaSq = 0, lower.tail = TRUE)
romegaSq(n, df1, df2, populationOmegaSq = 0)
```

Arguments

`x, q`	Vector of quantiles, or, in other words, the value(s) of Omega Squared.
`p`	Vector of probabilites (*p*-values).
`df1, df2`	Degrees of freedom for the numerator and the denominator, respectively.
`n`	Desired number of Omega Squared values.
`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 `userfriendlyscience`. 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.

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);
```

`oneway` *oneway*

Description

The oneway function wraps a number of analysis of variance functions into one convenient interface that is similar to the oneway anova command in SPSS.

Usage

```
oneway(y, x, posthoc=NULL, means=FALSE,
       fullDescribe=FALSE, levene=FALSE,
       plot=FALSE, digits=2, omegasq = TRUE,
       etasq = TRUE, corrections = FALSE,
       pvalueDigits=3, t=FALSE, conf.level=.95,
       posthocLetters = FALSE, posthocLetterAlpha = .05,
       silent=FALSE)
```

Arguments

`y`	y has to be a numeric vector.
`x`	x has to be vector that either is a factor or can be converted into one.
`posthoc`	Which post-hoc tests to conduct. Valid values are any correction methods in p.adjust.methods (at the time of writing of this document, "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none"), as well as "tukey" and "games-howell".
`means`	Whether to show the means for the y variable in each of the groups determined by the x variable.
`fullDescribe`	If TRUE, not only the means are shown, but all statistics acquired through the 'describe' function in the 'psych' package are shown.
`levene`	Whether to show Levene's test for equality of variances (using `car`'s `leveneTest` function but specifying `mean` as function to compute the center of each group).

`plot`	Whether to show a plot of the means of the y variable in each of the groups determined by the x variable.
`digits`	The number of digits to show in the output.
`omegasq`	Whether to show the omega squared effect size.
`etasq`	Whether to show the eta squared effect size (this is biased and generally advised against; omega squared is less biased).
`corrections`	Whether to show the corrections for unequal variances (Welch and Brown-Forsythe).
`pvalueDigits`	The number of digits to show for p-values; smaller p-values will be shown as <.001 or <.0001 etc.
`t`	Whether to transpose the dataframes with the means (if requested) and the anova results. This can be useful for blind people.
`conf.level`	Confidence level to use when computing the confidence interval for eta^2. Note that the function we use doubles the 'unconfidence' level to maintain consistency with the NHST value (see http://yatani.jp/HCIstats/ANOVA#RCodeOneWay, http://daniellakens.blogspot.nl/2014/06/calculating-confidence-intervals-for.html or 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. doi:10.1037/1082-989X.9.2.164
`posthocLetters`	Whether to also compute and show the letters signifying differences between groups when conducting post hoc tests. This requires package `multcompView` to be installed.
`posthocLetterAlpha`	The alpha to use when determining whether groups have different means when using `posthocLetters`.
`silent`	Whether to show warnings and other diagnostic information or remain silent.

Value

A list of three elements:

`input`	List with input arguments
`intermediate`	List of intermediate objects, such as the aov and Anova (from the car package) objects.
`output`	List with etasq, the effect size, and dat, a dataframe with the Oneway Anova results.

Note

By my knowledge the Brown-Forsythe correction was not yet available in R. I took this from the original paper (directed there by Field, 2014). Note that this is the corrected *F* value, not the Brown-Forsythe test for normality!

Author(s)

Gjalt-Jorn Peters

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

References

Brown, M., & Forsythe, A. (1974). *The small sample behavior of some statistics which test the equality of several means.* Technometrics, 16(1), 129-132. https://doi.org/10.2307/1267501

Field, A. (2014) *Discovering statistics using SPSS* (4th ed.). London: Sage.

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. doi:10.1037/1082-989X.9.2.164

Examples

```
### Do a oneway Anova
oneway(y=ChickWeight$weight, x=ChickWeight$Diet);

### Also order means and transpose the results
oneway(y=ChickWeight$weight, x=ChickWeight$Diet, means=TRUE, t=TRUE);
```

`paginatedAsymmetricalScatterMatrix`
paginatedAsymmetricalScatterMatrix

Description

A function that generates a series of asymmetricalScatterMatrices, so that they can be printed or included in PDFs.

Usage

```
paginatedAsymmetricalScatterMatrix(dat, cols, rows, maxRows = 5, ...)
```

Arguments

`dat`	The dataframe containing the variables specified in `cols` and `rows`.
`cols`	The names of the variables to use for the columns.
`rows`	The names of the variables to use for the rows.
`maxRows`	The maximum number of rows on one 'page' (i.e. in one `asymmetricalScatterMatrix`).
`...`	Extra arguments to pass on to each `asymmetricalScatterMatrix` call.

Value

An object containing the asymmetricalScatterMatrices in a list:

`input`	Input values.
`intermediate`	Some values/objects generated in the process.
`output`	A list containing the object 'scatterMatrices', which is a list of the generated scatterMatrices.

Author(s)

Gjalt-Jorn Peters

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

See Also

`asymmetricalScatterMatrix`

Examples

```
## Not run:
### (Not run by default because it's quite timeconsuming.)
tmp <- paginatedAsymmetricalScatterMatrix(infert, cols=c("parity"),
                                          rows=c("induced", "case",
                                                 "spontaneous", "age",
                                                 "pooled.stratum"),
                                          maxRows = 3,
                                          showCorrelations="top-right");
tmp$output$scatterMatrices[[1]];

## End(Not run)
```

`piecewiseRegr` *Piecewise regression analysis*

Description

This function conducts a piecewise regression analysis and shows a plot illustrating the results. The function enables easy customization of the main plot elements and easy saving of the plot with anti-aliasing.

Usage

```
piecewiseRegr(data,
              timeVar = 1,
              yVar = 2,
              phaseVar = NULL,
              baselineMeasurements = NULL,
              robust = FALSE,
              digits = 2,
              colors = list(pre = viridis(4)[1],
                            post = viridis(4)[4],
                            diff = viridis(4)[3],
                            intervention = viridis(4)[2],
                            points = "black"),
              theme = theme_minimal(),
              pointSize = 2,
```

```
                 pointAlpha = 1,
                 lineSize = 1,
                 yRange=NULL,
                 yBreaks = NULL,
                 showPlot = TRUE,
                 plotLabs = NULL,
                 outputFile = NULL,
                 outputWidth = 16,
                 outputHeight = 16,
                 ggsaveParams = list(units = "cm",
                                     dpi = 300,
                                     type = "cairo"))
```

Arguments

`data`	The dataframe containing the variables for the analysis.
`timeVar`	The name of the variable containing the measurement moments (or an index of measurement moments). An index can also be specified, and assumed to be 1 if omitted.
`yVar`	The name of the dependent variable. An index can also be specified, and assumed to be 2 if omitted.
`phaseVar`	The variable containing the phase of each measurement. Note that this normally should only have two possible values.
`baselineMeasurements`	If no phaseVar is specified, `baselineMeasurements` can be used to specify the number of baseline measurements, which is then used to construct the `phaseVar` dummy variable.
`robust`	Whether to use normal or robust linear regression.
`digits`	The number of digits to show in the results.
`colors`	The colors to use for the different plot elements.
`theme`	The theme to use in the plot.
`pointSize,lineSize`	The sizes of points and lines in the plot.
`yRange`	This can be used to manually specify the possible values that the dependent variable can take. If not provided, the observed range of the dependent variable values is used instead.
`yBreaks`	If `NULL`, the `pretty` function is used to estimate the best breaks for the Y axis. If a value is supplied, this value is used as the size of intervals between the (floored) minimum and (ceilinged) maximum of `yRange` (e.g. if `yBreaks` is 1, a break point every integer; if 2 and the minimum is 1 and the maximum is 7, breaks at 1, 3, 5 and 7; etc).
`pointAlpha`	The alpha channel (transparency, or rather, 'opaqueness') of the points.
`showPlot`	Whether to show the plot or not.
`plotLabs`	A list with arguments to the `ggplot2` `labs` function, which can be used to conveniently set plot labels.

`outputFile`	If not `NULL`, the path and filename specifying where to save the plot.
`outputWidth, outputHeight`	The dimensions of the plot when saving it (in units specified in `ggsaveParams`).
`ggsaveParams`	The parameters to use when saving the plot, passed on to `ggsave`.

Value

Mainly, this function prints its results, but it also returns them in an object containing three lists:

`input`	The arguments specified when calling the function
`intermediate`	Intermediat objects and values
`output`	The results such as the plot.

Author(s)

Peter Verboon & Gjalt-Jorn Peters (both at the Open University of the Netherlands)

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

References

Verboon, P. & Peters, G.-J. Y. (2018) Applying the generalised logistic model in single case designs: modelling treatment-induced shifts. *PsyArXiv* `https://doi.org/10.17605/osf.io/ad5eh`

See Also

`genlog`

Examples

```
### Load dataset
data(Singh);

### Extract Jason
dat <- Singh[Singh$tier==1, ];

### Conduct piecewise regression analysis
piecewiseRegr(dat,
              timeVar='time',
              yVar='score_physical',
              phaseVar='phase');

### Pretend treatment started between measurements
### 5 and 6
piecewiseRegr(dat,
              timeVar='time',
              yVar='score_physical',
              baselineMeasurements=5);
```

`posthocTGH` *posthocTGH*

Description

This function is used by the 'oneway' function for oneway analysis of variance in case a user requests post-hoc tests using the Tukey or Games-Howell methods.

Usage

```
posthocTGH(y, x, method=c("games-howell", "tukey"),
           conf.level = 0.95, digits=2, p.adjust="none",
           formatPvalue = TRUE)
```

Arguments

`y`	y has to be a numeric vector.
`x`	x has to be vector that either is a factor or can be converted into one.
`method`	Which post-hoc tests to conduct. Valid values are "tukey" and "games-howell".
`conf.level`	Confidence level of the confidence intervals.
`digits`	The number of digits to show in the output.
`p.adjust`	Any valid `p.adjust` method.
`formatPvalue`	Whether to format the p values according to APA standards (i.e. replace all values lower than .001 with '<.001'). This only applies to the printing of the object, not to the way the p values are stored in the object.

Value

A list of three elements:

`input`	List with input arguments
`intermediate`	List of intermediate objects.
`output`	List with two objects 'tukey' and 'games.howell', containing the outcomes for the respective post-hoc tests.

Note

This function is based on a file that was once hosted at http://www.psych.yorku.ca/cribbie/6130/games_howell.R, but has been removed since. It was then adjusted for implementation in the `userfriendlyscience` package. Jeffrey Baggett needed the confidence intervals, and so emailed them, after which his updated function was used. In the meantime, it appears Aaron Schlegel (`https://rpubs.com/aaronsc32`) independently developed a version with confidence intervals and posted it on RPubs at `https://rpubs.com/aaronsc32/games-howell-test`.

Also, for some reason, `p.adjust` can be used to specify additional correction of *p* values. I'm not sure why I implemented this, but I'm not entirely sure it was a mistake either. Therefore, in `userfriendlyscience` version 0.6-2, the default of this setting changed from `"holm"` to `"none"` (also see https://stats.stackexchange.com/questions/83941/games-howell-post-hoc-test-in-r).

Author(s)

Gjalt-Jorn Peters (Open University of the Netherlands) & Jeff Bagget (University of Wisconsin - La Crosse)

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

Examples

```
### Compute post-hoc statistics using the tukey method
posthocTGH(y=ChickWeight$weight, x=ChickWeight$Diet, method="tukey");
### Compute post-hoc statistics using the games-howell method
posthocTGH(y=ChickWeight$weight, x=ChickWeight$Diet);
```

`powerHist` *powerHist*

Description

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

Usage

```
powerHist(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 = dlvTheme(),
          rug = NULL, jitteredRug = TRUE, rugSides = "b",
          rugAlpha = .2, returnPlotOnly = FALSE)
```

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 `powerHist` object that also contains all settings and intermediate objects, or whether to only return the `ggplot` plot.

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

```
powerHist(mtcars$mpg)
```

`prevalencePower` *Power analysis for establishing a prevalence*

Description

This function can be used to establish how many participants are required to establish a prevalence rate with a given margin of error.

Usage

```
prevalencePower(expectedPrevalence,
                 marginOfError = 0.05,
                 conf.level = 0.95)
```

Arguments

`expectedPrevalence`	The expected prevalence.
`marginOfError`	The desired precision.
`conf.level`	The confidence of the confidence interval.

Details

Note that when uncertain as to the expected prevalence, it's better to assume a prevalence closer to 50%. Prevalences closer to 0% or 100% are easier to detect and therefore have more power.

Value

The required number of participants.

Author(s)

Gjalt-Jorn Peters

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

See Also

`convert.percentage.to.se`

Examples

```
### Required participants for detecting a prevalence of 10%
### with a 95% confidence interval of 10% wide:
prevalencePower(.1);

### Required participants for detecting a prevalence of 10%
### with a 95% confidence interval of 4% wide:
prevalencePower(.1, .02);
```

```
### Required participants for detecting a prevalence of 60%
### with a 95% confidence interval of 10% wide:
prevalencePower(.6);
```

`processLimeSurveyDropouts` *processLimeSurveyDropouts*

Description

This function makes it easy to parse the dropouts from a LimeSurvey questionnaire.

Usage

```
processLimeSurveyDropouts(lastpage,
                          pagenames = NULL,
                          relevantPagenames = NULL)
```

Arguments

`lastpage`	A vector with the 'lastpage' variable as LimeSurvey stores it (an integer denoting the last page a participant visited, in other words, where they dropped out).
`pagenames`	Optional: names for each page.
`relevantPagenames`	Optional: the names of those pages that should be included.

Details

This will be described more in detail in a forthcoming publications.

Value

A list with information about the dropout, including `ggplot`s.

Author(s)

Gjalt-Jorn Peters

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

Examples

```
processLimeSurveyDropouts(c(1,2,1,1,2,3,2,2,3,2,1));
```

`processLSvarLabels` *A function to conveniently process LimeSurvey labels*

Description

This function is meant to quickly parse the variable labels set by LimeSurvey. It works particularly well with dual anchor array questions, where the left and right anchors as well as the subquestions are extracted automatically.

Usage

```
processLSvarLabels(dat,
                    varnameRegExPairs = NULL,
                    labelExtractionRegExPair = c("\\[(.*)\\].*", "\\1"),
                    lengthToWrap = 50,
                    lengthToWrapAnchors = 20,
                leftAnchorRegExPairs = list(c(".*[[:graph:]]([A-Z][a-z][^|]*)\\|(.+)",
                                               "\\1")),
                rightAnchorRegExPairs = list(c(".*[[:graph:]]([A-Z][a-z][^|]*)\\|(.+)",
                                                "\\2")))
```

Arguments

`dat`	The dataframe as produced by `importLimeSurveyData`.
`varnameRegExPairs`	Pairs of regular expressions to replace in the variable names. This is useful when some pattern can be applied to the variable names to, for example, add underscores te denote different parts of the variable name. This has to be a list of character vectors that each have length 2.
`labelExtractionRegExPair`	The regular expression pair used to extract the labels.
`lengthToWrap`	At how many characters to wrap the subquestions.
`lengthToWrapAnchors`	At how many characters to wrap the anchors.
`leftAnchorRegExPairs`	The regular expression pairs to use to extract the left anchors.
`rightAnchorRegExPairs`	The regular expression pairs to use to extract the right anchors.

Details

This function processes LimeSurvey variable labels and applies regular expressions to automatically extract subquestions and left and right anchors.

Value

A dataframe that can conveniently be used with `detStructAddVarLabels`.

Author(s)

Gjalt-Jorn Peters

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

References

(Forthcoming)

See Also

`determinantStructure`, `determinantVar`, `subdeterminants`, `subdeterminantProducts`, `detStructAddVarLabels`, `detStructAddVarNames`, `detStructComputeProducts`, `detStructComputeScales`, `detStructCIBER`

Examples

```
### No examples provided yet; this would require data to be included,
### and that's not available yet.
```

`processOpenSesameIAT` *processOpenSesameIAT*

Description

This function reads IAT files as generated by the OpenSesame script available at [INSERT URL].

Usage

```
processOpenSesameIAT(dataPath,
                     blocks.sizes = c(18, 36, 48, 36, 48),
                     blocks.congruent = c(2, 3),
                     blocks.incongruent = c(4, 5),
                     blocks.realTrials = c(3, 5),
                     blocks.practiceTrials = c(2, 4),
                     congruentLarger = TRUE,
                     responseTime.min = 400,
                     responseTime.max = 2500,
                     responseTime.penalty = 600,
                     outputFile = NULL,
                     wideOutputFile = NULL,
                     showLog = TRUE,
                     filenameRegEx = "subject-(\\d+)(\\w+)\\.csv",
                     regExValues = c("subject","session"),
                     participantVarName = "subject",
                     taskVarName = "session",
                     openSesameVarNames = list(correct = "correct",
                                               response_time = "response_time"),
                     stimulusSelectionVarName = NULL,
```

```
                  stimulusSelectionValues = NULL,
                  roundOutput = 6,
                  decimalSeparator = ".",
                  inputDecimalSeparator = ".",
                  inputfileSelectionColumns = NULL,
                  inputfileSelectionValues = NULL)
```

Arguments

`dataPath`	A directory containing the .csv files that OpenSesame provides.
`blocks.sizes`	A vector containing the number of trials of each block.
`blocks.congruent`	A vector containing the numbers of the congruent blocks.
`blocks.incongruent`	A vector containing the numbers of the incongruent blocks.
`blocks.realTrials`	A vector containing the numbers of the real trials.
`blocks.practiceTrials`	A vector containing the numbers of the practice trials.
`congruentLarger`	Whether the response latencies for the congruent trials (TRUE) or the incongruent trials (FALSE) are expected to be larger. This simply multiplies the final D600 measures with -1.
`responseTime.min`	Minimum number of milliseconds of response time (all shorter times will be removed).
`responseTime.max`	Maximum number of milliseconds of response time (all longer times will be replaced with this number).
`responseTime.penalty`	Penalty in milliseconds to add to the response times for incorrect responses.
`outputFile`	If specified, the aggregated datafile is stored in this file.
`wideOutputFile`	If specified, the wide version of the datafile will be stored in this file.
`showLog`	Boolean; if TRUE, shows the log (is stored in the resulting object anyway).
`filenameRegEx`	Regular expression describing the filenames. This has two purposes. First, only files matching this regular expression will be processed (note that you can set it to `NULL` to process all files). Second, by using `"\1"`, `"\2"`, etc, matched patterns can be extracted from the filenames and stored as variables in the final datafile (see `sub` for more information on regular expression matching). The default pattern, `"subject-(\d+)(\w+)\.csv"`, which is read by R as `"subject-(\d+)(\w+)\.csv"` (because the backslash is the escape symbol, double backspaces are needed to specify one backspace, see `Quotes`), assumes that all filenames start with `'subject-'`, followed by the subject number (`"\d+"` matches one or more digits), immediately followed by one or more letters and digits (`"\w+"` matches one or more letters or digits) indicating the session that the datafile pertains to. If you only have subject numbers, you'd

use `"subject-(\d+)\.csv"` or perhaps `"subject-(\w+)\.csv"` if the subjects could also have letters in their identifiers. Note that you have to include the variable names of each of these extractable patterns in `regExValues`!

`regExValues`	Here, the names of the variables extracted using the regular expression specified in `filenameRegEx` are provided. Must of course have the same length as the number of patterns specified in `filenameRegEx`, and in the same order.
`participantVarName, taskVarName`	Variable name of the variable identifying participants and tasks (usually extracted from the filename, so should be a value in regExValues). Tasks are usually different within-subject conditions.
`openSesameVarNames`	A list with the two elements 'correct' and 'response_time', which should be the variable names that OpenSesame used to write, for each trial, whether the response was correct or not ('correct') and what the response time was ('response_time');
`stimulusSelectionVarName, stimulusSelectionValues`	These arguments can be used to specify a subset of stimuli to process. Specify which column contains the values to select in `stimulusSelectionVarName`, and specify the value(s) to select in `stimulusSelectionValues`.
`roundOutput`	Number of digits to round the output to. This is useful for importing into a program that doesn't quite get how storing numbers works, such as SPSS or Excel; they sometimes don't manage to import numbers with many decimals.
`decimalSeparator`	When working with e.g. Excel, it can be easier to just specify the decimal separator rather than switch Excel's (and therefore Windows') locale.
`inputDecimalSeparator`	The decimal separator to specify to `read.csv` when reading the data files.
`inputfileSelectionColumns, inputfileSelectionValues`	This functionality still has to be implemented. Once implemented, these arguments can be used to specify a column, and a (set of) value(s) in that column to use to select which rows to process (also see `stimulusSelectionVarName` and `stimulusSelectionValues`).

Details

Note that this function was developed to read the OpenSesame IAT datafiles created by the OpenSesame script developed by Kenny Wolfs, Jacques van Lankveld, and Frederik van Acker at the Open University of the Netherlands. If you use a different version (for example, the one contributed to the OpenSesame paradigm repository by Hansika Kapoor, see `http://osdoc.cogsci.nl/3.0/standard-tasks/#implicit-association-test-iat`), you will have to specify the variable names you specified to OpenSesame for the response time and for whether the response was correct in `openSesameVarNames`. For example, if you use Hansika's IAT task, you'll have to specify `openSesameVarNames = list(correct = "correct",response_time = "avg_rt")`

Similarly, of course you will probably have to specify the number of trials per block etc. Also, you may want to set showLog to FALSE, as the logging is quite detailed.

Value

An object with the raw files, the processed files, and the file converted to wide format. But most users will probably specify `outputFile` and/or `wideOutputFile` to just export the output files directly.

Author(s)

Gjalt-Jorn Peters

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

References

Mathot, S., Schreij, D., & Theeuwes, J. (2012). OpenSesame: An open-source, graphical experiment builder for the social sciences. *Behavior Research Methods*, 44(2), 314-324. doi:10.3758/s13428-011-0168-7

Kapoor, H. (2015). The creative side of the Dark Triad. *Creativity Research Journal*, 27(1), 58-67. doi:10.1080/10400419.2014.961775.

Examples

```
## Not run:
### This will process all files in the specified directory, but not
### export anything
processed <- processOpenSesameIAT("C:/directory/with/datafiles");

### This will export both the aggregated datafile and the wide datafile
processed <- processOpenSesameIAT("C:/directory/with/datafiles",
                                  outputFile="C:/directory/aggregated.csv",
                                  wideOutputFile="C:/directory/wide.csv");


## End(Not run)
```

`pwr.confIntR` *Determine required sample size for a given confidence interval width for Pearson's r*

Description

This function computes how many participants you need if you want to achieve a confidence interval of a given width. This is useful when you do a study and you are interested in how strongly two variables are associated.

Usage

```
pwr.confIntR(r, w = 0.1, conf.level = 0.95)
```

Arguments

`r`	The correlation you expect to find (confidence intervals for a given level of confidence get narrower as the correlation coefficient increases).
`w`	The required half-width (or margin of error) of the confidence interval.
`conf.level`	The level of confidence.

Value

The required sample size, or a vector or matrix of sample sizes if multiple correlation coefficients or required (half-)widths were supplied. The row and column names specify the `r` and `w` values to which the sample size in each cell corresponds. The confidence level is set as attribute to the resulting vector or matrix.

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

`pwr.confIntR`

Examples

```
pwr.confIntR(c(.4, .6, .8), w=c(.1, .2));
```

pwr.omegasq	*Power calculations for Omega Squared.*

Description

This function uses `pwr.anova.test` from the `pwr` package in combination with `convert.cohensf.to.omegasq` and `convert.omegasq.to.cohensf` to provide power analyses for Omega Squared.

Usage

```
pwr.omegasq(k = NULL, n = NULL, omegasq = NULL,
            sig.level = 0.05, power = NULL, digits = 4)
```

Arguments

`k`	The number of groups.
`n`	The sample size.
`omegasq`	The Omega Squared value.
`sig.level`	The significance level (alpha).
`power`	The power.
`digits`	The number of digits desired in the output (4, the default, is quite high; but omega squared value tend to be quite low).

Details

This function was written to work similarly to the power functions in the `pwr` package.

Value

An `power.htest.ufs` object that contains a number of input and output values, most notably:

`power`	The (specified or computed) power
`n`	The (specified or computed) sample size in each group
`sig.level`	The (specified or computed) significance level (alpha)
`sig.level`	The (specified or computed) Omega Squared value
`cohensf`	The computed value for the Cohen's f effect size measure

Author(s)

Gjalt-Jorn Peters & Peter Verboon

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

See Also

`pwr.anova.test`, `convert.cohensf.to.omegasq`, `convert.omegasq.to.cohensf`

Examples

```
pwr.omegasq(omegasq=.06, k=3, power=.8)
```

`randomizationSuccess` *Computations for successful randomization*

Description

`prob.randomizationSuccess` computes the probability that two groups are equivalent given a specific sample size, number of nuisance variables, and definition of 'equivalence' (in terms of the Cohen's d expressing the maximum acceptable difference between the groups on any of the nuisance variables).

`pwr.randomizationSuccess` computes the sample size required to make randomization succeed in a specified proportion of the studies with a two-cell design. 'Success' is defined as the two groups differing at most with a specified effect size on any of a given number or nuisance variables.

Usage

```
prob.randomizationSuccess(n = 1000,
                          dNonequivalence = .2,
                          nNuisanceVars = 100)
pwr.randomizationSuccess(dNonequivalence = 0.2,
                         pRandomizationSuccess = 0.95,
                         nNuisanceVars = 100)
```

Arguments

`n`	The sample size.
`dNonequivalence`	The maximum difference between the two groups that is deemed acceptable.
`pRandomizationSuccess`	The desired probability that the randomization procedure succeeded in generating two equivalent groups (i.e. differing at most with `dNonequivalence`).
`nNuisanceVars`	The number of nuisance variables that the researchers assumes exists.

Details

For more details, see Peters & Gruijters (2017).

Value

For `prob.randomizationSuccess`, the probability that the two groups are equivalent. The function is vectorized, so returns either a vector of length one, a vector of length > 1, a matrix, or an array.

For `pwr.randomizationSuccess`, the required sample size. The function is vectorized, so returns either a vector of length one, a vector of length > 1, a matrix, or an array.

Author(s)

Gjalt-Jorn Peters

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

References

Peters, G. J.-Y. & Gruijters, S. Why your experiments fail: sample sizes required for randomization to generate equivalent groups as a partial solution to the replication crisis (2017). http://dx.doi.org/

See Also

`dCohensd`

Examples

```
### To be on the safe side: sample size required to
### obtain 95% likelihood of success when assuming
### 100 nuisance variables exist.
pwr.randomizationSuccess(dNonequivalence = 0.2,
                         pRandomizationSuccess = 0.95,
                         nNuisanceVars = 100);

### Living on the edge:
pwr.randomizationSuccess(dNonequivalence = 0.2,
                         pRandomizationSuccess = 0.60,
                         nNuisanceVars = 10);

### For those with quite liberal ideas of 'equivalence':
pwr.randomizationSuccess(dNonequivalence = 0.5,
                         pRandomizationSuccess = 0.95,
                         nNuisanceVars = 100);

### And these results can be checked with
### prob.randomizationSuccess:
prob.randomizationSuccess(1212, .2, 100);
prob.randomizationSuccess(386, .2, 10);
prob.randomizationSuccess(198, .5, 100);

### Or in one go:
prob.randomizationSuccess(n=c(198, 386, 1212), c(.2, .5), c(10, 100));
```

`regr` *regr: a simple regression analysis wrapper*

Description

The `regr` function wraps a number of linear regression functions into one convenient interface that provides similar output to the regression function in SPSS. It automatically provides confidence intervals and standardized coefficients. Note that this function is meant for teaching purposes, and therefore it's only for very basic regression analyses.

Usage

```
regr(formula, data = NULL, conf.level = .95,
     digits = 2, pvalueDigits = 3,
     coefficients = c("raw", "scaled"),
     plot = FALSE, pointAlpha = .5,
     collinearity = FALSE, influential = FALSE,
     ci.method = c("widest", "r.con", "olkinfinn"),
     ci.method.note = FALSE, env = parent.frame())
```

Arguments

`formula`	The formula of the regression analysis, of the form `y ~ x1 + x2`, where y is the dependent variable and x1 and x2 are the predictors.
`data`	If the terms in the formula aren't vectors but variable names, this should be the dataframe where those variables are stored.
`conf.level`	The confidence of the confidence interval around the regression coefficients.
`digits`	Number of digits to round the output to.
`pvalueDigits`	The number of digits to show for p-values; smaller p-values will be shown as <.001 or <.0001 etc.
`coefficients`	Which coefficients to show; can be "raw" to only show the raw (unstandardized) coefficients; "scaled" to only show the scaled (standardized) coefficients), or c("raw", "scaled') to show both.
`plot`	For regression analyses with only one predictor (also sometimes confusingly referred to as 'univariate' regression analyses), scatterplots with regression lines and their standard errors can be produced.
`pointAlpha`	The alpha channel (transparency, or rather: 'opaqueness') of the points drawn in the plot.
`collinearity`	Whether to compute and show collinearity diagnostics (specifically, the tolerance (*1 - R^2*, where *R^2* is the one obtained when regressing each predictor on all the other predictors) and the Variance Inflation Factor (VIF), which is the reciprocal of the tolerance, i.e. *VIF = 1 / tolerance*).
`influential`	Whether to compute diagnostics for influential cases. These are stored in the returned object in the `lm.influence.raw` and `lm.influence.scaled` objects in the `intermediate` object.
`ci.method, ci.method.note`	Which method to use for the confidence interval around R squared, and whether to display a note about this choice.
`env`	The enviroment where to evaluate the formula.

Value

A list of three elements:

`input`	List with input arguments
`intermediate`	List of intermediate objects, such as the lm and confint objects.
`output`	List with two dataframes, one with the raw coefficients, and one with the scaled coefficients.

Author(s)

Gjalt-Jorn Peters

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

Examples

```
### Do a simple regression analysis
regr(age ~ circumference, dat=Orange);

### Show more digits for the p-value
regr(Orange$age ~ Orange$circumference, pvalueDigits=18);
```

`regrInfluential` *Detecting influential cases in regression analyses*

Description

This function combines a number of criteria for determining whether a datapoint is an influential case in a regression analysis. It then sum the criteria to compute an index of influentiality. A list of cases with an index of influentiality of 1 or more is then displayed, after which the regression analysis is repeated without those influantial cases. A scattermatrix is also displayed, showing the density curves of each variable, and in the scattermatrix, points that are colored depending on how influential each case is.

Usage

```
regrInfluential(formula, data)
```

Arguments

`formula`	The formule of the regression analysis.
`data`	The data to use for the analysis.

Value

A `regrInfluential` object, which, if printed, shows the influential cases, the regression analyses repeated without those cases, and the scatter matrix.

Author(s)

Gjalt-Jorn Peters & Marwin Snippe

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

Examples

```
regrInfluential(mpg ~ hp, mtcars);
```

`reliability` *Reliability function similar to the SPSS RELIABILITY command*

Description

This function was developed to offer a function that roughly works similar to the SPSS RELIABILITY command.

Usage

```
reliability(data,
            items = NULL,
            itemDiagnostics = FALSE,
            digits = 2)
```

Arguments

`data`	The dataframe containing the variables (items, questions) of interest.
`items`	Optionally, the variables (items, questions) of interest. If omitted, all variables (items, questions) in the dataframe will be used.
`itemDiagnostics`	Whether to also display the item diagnostics (specifically, the corrected item-total correlation, mean and variance excluding each item, and the reliability coefficients excluding each item).
`digits`	The number of digits to use when displaying the results.

Author(s)

Gjalt-Jorn Peters

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

See Also

`scaleStructure`, the excellent `psych` package

Examples

```
## Not run:
## (Not run to test because it takes a long time.)

data(testRetestSimData);
reliability(testRetestSimData[, 2:11], itemDiagnostics = TRUE);

## End(Not run)
```

`removeExceptionalValues`

removeExceptionalValues

Description

A function to replace exceptional values with NA. This can be used to quickly remove impossible values, for example, when participants entered their age as 344.

Usage

```
removeExceptionalValues(dat, items = NULL, exception = 0.005,
                           silent = FALSE, stringsAsFactors = FALSE)
```

Arguments

`dat`	The dataframe containing the items to inspect.
`items`	The items to inspect.
`exception`	How rare a value must be to be considered exceptional (and replaced by NA).
`silent`	Can be used to suppress messages.
`stringsAsFactors`	Whether to convert strings to factors when creating a dataframe from lapply output.

Details

Note that exceptional values may be errors (e.g. participants accidently pressed a key twice, or during data entry, something went wrong), but they may also be indicative of participants who did not seriously participate in the study. Therefore, it is advised to first use `exceptionalScores` to look for patterns where participants enter many exceptional scores.

Value

The dataframe, with exceptional values replaced by NA.

Author(s)

Gjalt-Jorn Peters

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

See Also

`exceptionalScores`

Examples

```
removeExceptionalValues(mtcars, exception=.1);
```

---

rMatrix *rMatrix*

Description

rMatrix provides a correlation matrix with confidence intervals and a p-value adjusted for multiple testing.

Usage

```
rMatrix(dat, x, y=NULL, conf.level = .95, correction = "fdr",
        digits = 2, pValueDigits=3, colspace=2, rowspace=0,
        colNames ="numbers",
        output="R",
        env.LaTeX = 'tabular',
        pboxWidthMultiplier = 1)
```

Arguments

`dat`	A dataframe containing the relevant variables.
`x`	Vector of 1+ variable names.
`y`	Vector of 1+ variable names; if this is left empty, a symmetric matrix is created; if this is filled, the matrix will have the x variables defining the rows and the y variables defining the columns.
`conf.level`	The 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 (yet :-)).
`digits`	With what precision do you want the results to print.
`pValueDigits`	Determines the number of digits to use when displaying p values. P-values that are too small will be shown as p<.001 or p<.00001 etc.
`colspace`	Number of spaces between columns (only for R output, ignored for LaTeX output)

`rowspace`	Number of rows between table rows (note: one table row is 2 rows; only for R output, ignored for LaTeX output).
`colNames`	colNames can be "numbers" or "names". "Names" cause variables names to be printed in the heading; "numbers" causes the rows to become numbered and the numbers to be printed in the heading.
`output`	Can be "R" or "LaTeX"; if output is set to "LaTeX", the result is a LaTeX table (e.g. for use in knitr).
`env.LaTeX`	For LaTeX output, the environment can be set with env.LaTeX.
`pboxWidthMultiplier`	When using LaTeX, pboxWidthMultiplier can be used to make the cells narrower or wider (1 works for anything up until 4 or 5 digits).

Details

rMatrix provides a symmetric or asymmetric matrix of correlations their confidence intervals, and p-values. The p-values can be corrected for multiple testing.

Value

An object with the input and several output variables. Most notably a number of matrices:

`r`	Pearson r values.
`parameter`	Degrees of freedom.
`ci.lo`	Lower bound of Pearson r confidence interval.
`ci.hi`	Upper bound of Pearson r confidence interval.
`p.raw`	Original p-values.
`p.adj`	p-values adjusted for multiple testing.

Author(s)

Gjalt-Jorn Peters

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

Examples

```
rMatrix(mtcars, x=c('disp', 'hp', 'drat'))
```

`rnwString` *rnwString functions*

Description

The rnwString functions make knitting PFDs a bit more userfriendly.

The sanitizeLatexString function sanitizes a LaTeX string by escaping special characters. It is strongly based on the function described on http://stackoverflow.com/questions/5406071/r-sweave-latex-escape-variables-to-be-printed-in-latex by Aaron Rendahl.

Usage

```
rnwString.initiate(studyName, authorName,
                   docClassArgs = 'a4paper,portrait,11pt',
                   newPage = TRUE, pageMargins=15)
rnwString.terminate(rnwString)
rnwString.generate(rnwString, rnwPath, fileName, pdfLatexPath,
                   envir = parent.frame())
sanitizeLatexString(str)
hasLaTeX(pdfLatexPath)
```

Arguments

`studyName`	The name of the study - used as the title of the PDF.
`authorName`	The name of the author(s) - also inserted on title page of the PDF.
`docClassArgs`	Default arguments for the document class in LaTeX. For example, to use landscape pages, this should be 'a4paper,landscape,11pt'.
`newPage`	Whether to end the initiation string with a newpage command. This can be set to false if you want to add more information on the first page(s).
`pageMargins`	Margin of the pages in millimeters.
`rnwString`	The rnwString to terminate or (after termination) generate.
`rnwPath`	The path where the temporary files (.rnw, .tex, etc) should be created. Use forward slashes. Note: the last character should not be a slash!
`fileName`	The filename to use for the temporary files. Omit the extension!
`pdfLatexPath`	The path to PdfLaTex. This file is part of a LaTeX installation that creates a pdf out of a .tex file. In Windows, you can download (portable) MikTex from http://miktex.org/portable. You then decide yourself where to install MikTex; pdflatex will end up in a subfolder 'miktex\bin', so if you installed MikTex in, for example, 'C:\Program Files\MikTex', the total path becomes 'C:\Program Files\MikTex\miktex\bin'. Note that R uses slashes instead of backslashes to separate folders, so in this example, pdfLatexPath should be 'C:/Program Files/MikTex/miktex/bin' In MacOS, you can install MacTex from http://tug.org/mactex/ By default, pdflatex ends up in folder '/user/texbin', which is what pdfLatexPath should be in that default case.

	In Ubuntu, you can install TexLive base by using your package manager to install texlive-latex-base, or using the terminal: 'sudo apt-get install texlive-latex-base' In ubuntu, by default pdflatex ends un in folder '/usr/bin', which is what pdfLatexPath should be in that default case.
`envir`	The environment where to evaluate the expressions (normally the environment where the function is called).
`str`	The character string to sanitize.

Value

rnwString.initiate starts an rnwString; rnwString.terminate closes it; and rnwString.generate takes an rnwString and creates a pdf.

sanitizeLatexString returns the sanitized string.

hasLaTeX checks pdfLatexPath to make sure pdflatex or pdflatex.exe exists.

Author(s)

Gjalt-Jorn Peters

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

Examples

```
### sanitize a string
newString <- sanitizeLatexString('this is a tilde: ~.');
newString;
### newString is now: "this is a tilde: ~."
```

RsqDist *The distribution of R squared (as obtained in a regression analysis)*

Description

These functions use the beta distribution to provide the R Squared distribution.

Usage

```
dRsq(x, nPredictors, sampleSize, populationRsq = 0)
pRsq(q, nPredictors, sampleSize, populationRsq = 0, lower.tail = TRUE)
qRsq(p, nPredictors, sampleSize, populationRsq = 0, lower.tail = TRUE)
rRsq(n, nPredictors, sampleSize, populationRsq = 0)
```

Arguments

`x, q`	Vector of quantiles, or, in other words, the value(s) of R Squared.
`p`	Vector of probabilites (*p*-values).
`nPredictors`	The number of predictors.
`sampleSize`	The sample size.
`n`	The number of R Squared values to generate.
`populationRsq`	The value of R Squared in the population; this determines the center of the R Squared distribution. This has not been implemented yet in this version of `userfriendlyscience`. If anybody knows how to do this and lets me know, I'll happily integrate this of course.
`lower.tail`	logical; if TRUE (default), probabilities are the likelihood of finding an R Squared smaller than the specified value; otherwise, the likelihood of finding an R Squared larger than the specified value.

Details

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

Value

`dRsq` gives the density, `pRsq` gives the distribution function, `qRsq` gives the quantile function, and `rRsq` generates random deviates.

Note

These functions are based on the Stack Exchange (Cross Validated) post at `http://stats.stackexchange.com/questions/130069/what-is-the-distribution-of-r2-in-linear-regression-under-the-null-hypothesis`. Thus, the credits go to Alecos Papadopoulos, who provided the answer that was used to write these functions.

Author(s)

Gjalt-Jorn Peters (based on a CrossValidated answer by Alecos Papadopoulos)

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

See Also

`dbeta`, `pbeta`, `qbeta`, `rbeta`

Examples

```
### Generate 10 random R Squared values
### with 2 predictors and 100 participants
rRsq(10, 2, 100);

### Probability of finding an R Squared of
### .15 with 4 predictors and 100 participants
```

```
pRsq(.15, 4, 100, lower.tail = FALSE);

### Probability of finding an R Squared of
### .15 with 15 predictors and 100 participants
pRsq(.15, 15, 100, lower.tail=FALSE);
```

`scaleDiagnosis` *scaleDiagnosis*

Description

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

Usage

```
scaleDiagnosis(dat=NULL, items=NULL, plotSize=180, sizeMultiplier = 1,
               axisLabels = "none", scaleReliability.ci=FALSE, conf.level=.95,
               powerHist=TRUE, ...)
```

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.
`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).
`powerHist`	Whether to use the default ggpairs histogram on the diagonal of the scattermatrix, or whether to use the powerHist version.
`...`	Additional arguments are passed on to powerHist.

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(dat=exampleData, items=c('item2', 'item4'));

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

## End(Not run)
```

`scaleDiagnosisToPDF` *scaleDiagnosisToPDF*

Description

`scaleDiagnosis` provides a number of diagnostics for a scale (an aggregative measure consisting of several items), and scaleDiagnosisToPDF takes the resulting object and generates a PDF file, which is then saved to disk.

Usage

```
scaleDiagnosisToPDF(scaleDiagnosisObject,
                    docTitle = "Scale diagnosis", docAuthor = "Author",
                    pdfLatexPath, rnwPath=getwd(),
                    filename = "scaleDiagnosis",
                    digits=2,
                    rMatrixColsLandscape = 6,
                    pboxWidthMultiplier = 1,
                    scatterPlotBaseSize = 4,
                    maxScatterPlotSize = NULL,
                    pageMargins=15,
                    pval=TRUE)
```

Arguments

`scaleDiagnosisObject`	An object generated by `scaleDiagnosis`.
`docTitle`	The title of the PDF file (printed on the first page).
`docAuthor`	The author to show in the PDF file (printed on the first page).
`pdfLatexPath`	The path to PdfLaTex. This file is part of a LaTeX installation that creates a pdf out of a .tex file. See `rnwString` for more information.
`rnwPath`	The path where the temporary files will be stored.
`filename`	Filename of the PDF (".pdf" is appended).
`digits`	Number of digits to show.
`rMatrixColsLandscape`	This number determines when the page(s) in the PDF is/are rotated; pages with matrices that have this number of columns or more are rotated.
`pboxWidthMultiplier`	Passed on to (unexported method) `print.rMatrix`.
`scatterPlotBaseSize`	Basic size of scatterplots in centimeters. If this number, multiplied by the number of items (i.e. columns/rows in scattermatrix) is larger than maxScatterPlotSize, it is ignored.
`maxScatterPlotSize`	Maximum size of scatterplots; automatically calculated if NULL.
`pageMargins`	Margins of landscape pages in millimeters.
`pval`	Whether to print p-values using the p-value formatting. Passed on to (unexported method) `print.rMatrix`.

Details

This functon generates a PDF file from a `scaleDiagnosis` object. `scaleDiagnosis` generates 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

Nothing is returned; the file is printed to disk.

Author(s)

Gjalt-Jorn Peters

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

Examples

```
## 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 and create object
scaleDiagnosisObject <- scaleDiagnosis(dat=exampleData);

### Generate a PDF
scaleDiagnosisToPDF(scaleDiagnosisObject);


## End(Not run)
```

`scaleInspection` *scaleInspection and a number of useful helper functions*

Description

scaleInspection is a function to generate a PDF with information to diagnose and inspect scales (aggregate measures); makeScales actually generates the scales; and meanConfInt and sdConfInt provide confidence intervals for means and standard deviations.

Usage

```
scaleInspection(dat, items = NULL,
                docTitle = "Scale inspection", docAuthor = "Author",
                pdfLaTexPath, rnwPath=getwd(),
                filename = "scaleInspection", convertFactors=TRUE,
                scaleReliability.ci=FALSE, conf.level=.95, digits=2,
                rMatrixColsLandscape = 6,
                pboxWidthMultiplier = 1,
                scatterPlotBaseSize = 4,
```

```
             pageMargins=15, show=FALSE,
             pval=TRUE)
makeScales(dat, scales, append=TRUE)
meanConfInt(vector=NULL, mean=NULL, sd=NULL, n=NULL, se=NULL, conf.level=.95)
sdConfInt(vector=NULL, sd=NULL, n=NULL, conf.level=.95)
```

Arguments

`dat`	Dataframe containing the items of the relevant scale
`items`	Either a character vector with the itemnames, or, if the items are organised in scales, a list of character vectors with the items in each scale.
`scales`	A list of character vectors with the items in each scale, where each vectors' name is the name of the scale.
`docTitle`	Title to use when generating the PDF.
`docAuthor`	Author(s) to include when generating the PDF.
`pdfLaTexPath`	The path to PdfLaTex. This file is part of a LaTeX installation that creates a pdf out of a .tex file. In Windows, you can download (portable) MikTex from http://miktex.org/portable. You then decide yourself where to install MikTex; pdflatex will end up in a subfolder 'miktex\bin', so if you installed MikTex in, for example, 'C:\Program Files\MikTex', the total path becomes 'C:\Program Files\MikTex\miktex\bin'. Note that R uses slashes instead of backslashes to separate folders, so in this example, pdfLaTexPath should be 'C:/Program Files/MikTex/miktex/bin' In MacOS, you can install MacTex from http://tug.org/mactex/ By default, pdflatex ends up in folder '/user/texbin', which is what pdfLaTexPath should be in that default case. In Ubuntu, you can install TexLive base by using your package manager to install texlive-latex-base, or using the terminal: 'sudo apt-get install texlive-latex-base' In ubuntu, by default pdflatex ends un in folder '/usr/bin', which is what pdfLaTexPath should be in that default case.
`rnwPath`	The path where the temporary files and the resulting PDF should be stored.
`filename`	The filename to use to save the pdf.
`convertFactors`	Whether to convert factors to numeric vectors for the analysis.
`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), meand, and sd, for respectively scaleInspection, meanConfInt and sdConfInt).
`digits`	The number of digits to use in the tables.
`rMatrixColsLandscape`	At how many columns (or rather, variables) or more should rMatrices be printed landscape?
`pboxWidthMultiplier`	Used for print.rMatrix; used to tweak the width of columns in the correlation matrix.

`scatterPlotBaseSize`	Size of one scatterplot in the scattermatrix in centimeters. If the total scattermatrix becomes larger than 18 cm, it’s scaled down to 18 cm.
`pageMargins`	Margins of the page in millimeters.
`show`	Whether to show the results (or only write them to the PDF).
`pval`	Whether to print p-values as p-values in correlation matrix.
`append`	Whether to return the dataframe including the new variables (`TRUE`), or a dataframe with only those new variables (`FALSE`).
`vector`	Numeric vector to use when computing confidence intervals.
`mean`	Mean to use when computing confidence intervals (when no vector is provided).
`sd`	Standard deviaton to use when computing confidence intervals (when no vector is provided).
`n`	Number of datapoints to base confidence intervals on.
`se`	Standard errorto use when computing confidence intervals (when no standard deviation or vector is provided).

Details

scaleInspection generates a PDF with useful diagnostics to assess a scale; those from scaleDiagnosis and an rMatrix.

makeScales generates the scales and stores them in the dataframe.

meanConfInt and sdConfInt just compute and return a confidence interval for a mean or standard deviation.

Value

scaleInspection returns nothing; it just generates a PDF.

makeScales returns the provided dataframe, now including the new scale variables.

meanConfInt and sdConfInt return an object, with in its ’output’ list, the confidence interval for a mean or standard deviation.

Author(s)

Gjalt-Jorn Peters

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

Examples

```
## Not run:
scaleInspection(mtcars, items=c('disp', 'hp', 'drat'), pdfLaTexPath="valid/path/here");

## End(Not run)

newDataframe <- makeScales(mtcars, list(senselessScale = c('disp', 'hp', 'drat')));
```

```
sdConfInt(sd=4, n=30);

meanConfInt(mean=5, sd=4, n=30)
```

---

scaleStructure *scaleStructure*

Description

The scaleStructure function (which was originally called scaleReliability) computes a number of measures to assess scale reliability and internal consistency.

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.

Usage

```
scaleStructure(dat=NULL, items = 'all', digits = 2, ci = TRUE,
               interval.type="normal-theory", conf.level=.95,
               silent=FALSE, samples=1000, bootstrapSeed = NULL,
               omega.psych = TRUE, poly = TRUE)
scaleReliability(dat=NULL, items = 'all', digits = 2, ci = TRUE,
                 interval.type="normal-theory", conf.level=.95,
                 silent=FALSE, samples=1000, bootstrapSeed = NULL,
                 omega.psych = TRUE, poly = TRUE)
```

Arguments

`dat`	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. 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 `ci.reliability`. 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).

Details

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

`omega`, `alpha`, and `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", powerHist=TRUE, ...)
```

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.
`powerHist`	Whether to use the default ggpairs histogram on the diagonal of the scattermatrix, or whether to use the powerHist version.
`...`	Additional arguments are passed on to powerHist.

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)
```

`scatterPlot` *Easy ggplot2 scatter plots*

Description

This function is intended to provide a very easy interface to generating pretty (and pretty versatile) `ggplot` scatter plots.

Usage

```
scatterPlot(x, y, pointsize = 3,
            theme = theme_bw(),
            regrLine = FALSE, regrCI = FALSE,
            regrLineCol = "blue",
            regrCIcol = regrLineCol,
            regrCIalpha = 0.25,
            width = 0, height = 0,
            position = "identity",
            xVarName = NULL,
            yVarName = NULL,
            ...)
```

Arguments

`x`	The variable to plot on the X axis.
`y`	The variable to plot on the Y axis.
`pointsize`	The size of the points in the scatterplot.
`theme`	The theme to use.
`regrLine`	Whether to show the regression line.
`regrCI`	Whether to display the confidence interval around the regression line.
`regrLineCol`	The color of the regression line.
`regrCIcol`	The color of the confidence interval around the regression line.
`regrCIalpha`	The alpha value (transparency) of the confidence interval around the regression line.
`width`	If `position` is 'jitter', the points are 'jittered': some random noise is added to change their location slightly. In that case 'width' can be set to determine how much the location should be allowed to vary on the X axis.
`height`	If `position` is 'jitter', the points are 'jittered': some random noise is added to change their location slightly. In that case 'height' can be set to determine how much the location should be allowed to vary on the Y axis.
`position`	Whether to 'jitter' the points (adding some random noise to change their location slightly, used to prevent overplotting). Set to 'jitter' to jitter the points.
`xVarName, yVarName`	Can be used to manually specify the names of the variables on the x and y axes.
`...`	And additional arguments are passed to `geom_point` or `geom_jitter` (if `jitter` is set to 'jitter').

Details

Note that if `position` is set to 'jitter', unless `width` and/or `height` is set to a non-zero value, there will still not be any jittering.

Value

A `ggplot` plot is returned.

Author(s)

Gjalt-Jorn Peters

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

See Also

`geom_point`, `geom_jitter`, `geom_smooth`

Examples

```
### A simple scatter plot
scatterPlot(mtcars$mpg, mtcars$hp);

### The same scatter plot, now with a regression line
### and its confidence interval added.
scatterPlot(mtcars$mpg, mtcars$hp, regrLine=TRUE, regrCI=TRUE);
```

`setCaptionNumbering` *Convenience function for numbered captions in knitr (and so, RMarkdown)*

Description

This function makes it easy to tell knitr (and so RMarkdown) to use numbered captions of any type.

Usage

```
setCaptionNumbering(captionName = "tab.cap",
                    prefix = ":Table %s: ",
                    suffix = "",
                    captionBefore = FALSE,
                    romanNumeralSetting = "counter_roman",
                    optionName = paste0("setCaptionNumbering_", captionName),
                    resetCounterTo = 1)
```

Arguments

`captionName`	The name of the caption; this is used both as unique identifier for the counter, and to set the caption text (included between the prefix and suffix) in the chunk options.
`prefix`	The text to add as prefix before the action caption; this will typically include '%s%' which will be replaced by the number of this caption.
`suffix`	The text to add as suffix after the action caption; this can also include '%s%' which will be replaced by the number of this caption. Together with the `prefix`, this can also be used to enclose the caption in html.
`captionBefore`	Whether the caption should appear before or after the relevant chunk output.
`romanNumeralSetting`	The name of the option (should be retrievable with `getOption`) where it's configured whether to use Roman (TRUE) or Latin (FALSE) numerals. FALSE is assumed if this option isn't set.
`optionName`	The name of the option to use to retrieve and set the counter. This can be used, for example, to have multiple caption types use the same counter.
`resetCounterTo`	If not `NULL` and numeric, the counter will start at this number.

Value

This function returns nothing, but instead sets the appropriate `knit_hooks`. Or rather, just one hook.

Author(s)

Gjalt-Jorn Peters

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

Examples

```
## Not run:
  setCaptionNumbering(captionName='tab.cap',
                      prefix = ":Table %s: ");

## End(Not run)
```

`setFigCapNumbering` *Automatic caption numbering knitr hooks for figures and tables*

Description

These function implement ideas by Max Gordon and DeanK (see Details) to add `knitr` hooks to automate the numbering of figures and tables when generating R Markdown documents.

Usage

```
setFigCapNumbering(captionName = "fig.cap",
                     figure_counter_str = "Figure %s: ",
                     figureClass = "", imgClass = "",
                     figureInlineStyle = c("display:block"),
                     imgInlineStyle = NULL,
                     optionName = paste0("setCaptionNumbering_", captionName),
                     resetCounterTo = 1)
setTabCapNumbering(table_counter_str = ":Table %s: ",
                     resetCounterTo = 1)
```

Arguments

`captionName`	The name of the caption, used in the `knitr` chunk options to provide the caption text.
`figure_counter_str, table_counter_str`	The string in which to add the number of the figure or table. The text '%s' will be replaced by the number.
`figureClass`	Optionally, a css class to pass to the <fig> HTML element that surrounds the <img>.
`imgClass`	Optionall, a css class to pass to the <img> HTML element.

`figureInlineStyle`	Any css style to pass to the figure element directly ('inline').
`imgInlineStyle`	Any css style to pass to the image element directly ('inline').
`optionName`	The name of the option to use to retrieve and set the counter. This can be used, for example, to have multiple caption types use the same counter.
`resetCounterTo`	If not `NULL` and numeric, the counter will start at this number.

Details

The figure caption function is basically the one designed by Max Gordon (see `http://gforge.se/2014/01/fast-track-publishing-using-knitr-part-iii/`.

The table caption function is an implementation of the ideas of DeanK (see `http://stackoverflow.com/questions/15258233/using-table-caption-on-r-markdown-file-using-knitr-to-use-in-pandoc-to-conv` combined with Max Gordon's function.

Value

Nothing is returned; the correct hooks are configured for `knitr`.

Author(s)

Max Gordon (setFigCapNumbering) and DeanK (setTabCapNumbering); implemented by Gjalt-Jorn Peters

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

See Also

`knitr`

Examples

```
## Not run:
  setFigCapNumbering("This is figure number %s, with caption text: ");

## End(Not run)
```

`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"
```

`showPearsonPower` *Visualisation of the power of a Pearson correlation test*

Description

This fuction is useful when conducting power analyses for a Pearson correlation. It draws the sampling distribution of Pearson's *r* assuming a null hypothesis value of *r* and assuming a the hypothetical population value. The probability of making a Type 1 error is also illustrated.

Usage

```
showPearsonPower(n = 100, rho = 0.3, rNull = 0,
                 distLabels = c("Null Hypothesis", "Population"),
                 rhoColor = "green", rhoFill = "green",
                 rhoAlpha = 0.1, rhoLineSize = 1,
                 rNullColor = "blue", rNullFill = "blue",
                 rNullAlpha = 0.1, rNullLineSize = 1,
                 type2Color = "red", type2Fill = "red",
                 type2Alpha = 0.1, type2LineSize = 0,
                 theme = dlvTheme(), alpha = 0.05, digits = 3)
```

Arguments

`n`	The number of participants.
`rho`	The value of the correlation coefficient in the population.
`rNull`	The value of the correlation coefficient according to the null hypothesis.
`distLabels`	Labels for the two distributions; the first one is the null hypothesis distribution, the second one the alternative distribution.
`rhoColor, rNullColor, type2Color`	The border colors of the distributions and the region used to illustrate the Type 2 error probability.
`rhoFill, rNullFill, type2Fill`	The fill colors of the distributions and the region used to illustrate the Type 2 error probability.
`rhoAlpha, rNullAlpha, type2Alpha`	The alpha (transparency) of the distributions and the region used to illustrate the Type 2 error probability.
`rhoLineSize, rNullLineSize, type2LineSize`	The line thicknesses of the distributions and the region used to illustrate the Type 2 error probability.
`theme`	The theme to use.
`alpha`	The significance level (alpha) of the null hypothesis test.
`digits`	The number of digits to round to.

Value

A `ggplot` plot is returned.

Author(s)

Gjalt-Jorn Peters

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

See Also

`didacticPlot`

Examples

```
## Not run:
showPearsonPower();

## End(Not run)
```

`simDataSet` *simDataSet*

Description

simDataSet can be used to conveniently and quickly simulate a dataset that satisfies certain constraints, such as a specific correlation structure, means, ranges of the items, and measurement levels of the variables. Note that the results are approximate; mvrnorm is used to generate the correlation matrix, but the factor are only created after that, so cutting the variable into factors may change the correlations a bit.

Usage

```
simDataSet(n,
           varNames,
           correlations = c(0.1, 0.4),
           specifiedCorrelations = NULL,
           means = 0,
           sds = 1,
           ranges = c(1, 7),
           factors = NULL,
           cuts = NULL,
           labels = NULL,
           seed = 20160503,
           empirical = TRUE,
           silent = FALSE)
```

Arguments

`n`	Number of requires cases (records, entries, participants, rows) in the final dataset.
`varNames`	Names of the variables in a vector; note that the length of this vector will determine the number of variables simulated.
`correlations`	The correlations between the variables are randomly sampled from this range using the uniform distribution; this way, it's easy to have a relatively 'messy' correlation matrix without the need to specify every correlation manually.
`specifiedCorrelations`	The correlations that have to have a specific value can be specified here, as a list of vectors, where each vector's first two elements specify variables names, and the last one the correlation between those two variables. Note that tweaking the correlations may take some time; the `mvrnorm` function will complain that "'Sigma' is not positive definite", or in other words, you supplied a combination of correlations that can't exist simultaneously, if you get it wrong.
`means, sds`	The means and standard deviations of the variables. Note that is you set `ranges` for one or more variables (see below), those ranges are used to rescale those variables, overriding any specified means and standard deviations. If only one mean or standard deviation is supplied, it's recycled along the variables.

`ranges`	The desired ranges of the variables, supplied as a named list where the name of each element corresponds to a variable. The `rescale` function will be used to rescale those variables for which a desired scale is specified here. Note that for those variables, the means and standard deviations will be determined by these new ranges.
`factors`	A vector of variable names that should be converted into factors (using `cut`). Make sure to specify lists for `cuts` and `labels` as well (of the same length).
`cuts`	A list of vectors that specify, for each factor, where to 'cut' the numeric vector into factor levels.
`labels`	A list of vectors that specify, for each factor, and for each level, the labels that should be assigned to the factor levels. Each vector in this list has to have one more element than each vector in the `cuts` list.
`seed`	The seed to use when generating the dataset (to make sure the exact same dataset can be generated repeatedly).
`empirical`	Whether to generate the data using the exact (`empirical = TRUE`) or approximate (`empirical = FALSE`) correlation matrix; this is passed on to `mvrnorm`.
`silent`	Whether to show intermediate and final descriptive information (correlation and covariance matrices as well as summaries).

Details

This function was intended to allow relatively quick generation of datasets that satisfy specific constraints, e.g. including a number of factors, variables with a specified minimum and maximum value or specified means and standard deviations, and of course specific correlations. Because all correlations except those specified are randomly generated from a uniform distribution, it's quite convenient to generate messy kind of real looking datasets quickly. Note that it's mostly a convenience function, and datasets will still require tweaking; for example, factors are simply numeric vectors that are `cut` *after* `mvrnorm` generated the data, so the associations will change slightly.

Value

The generated dataframe is returned invisibly.

Author(s)

Gjalt-Jorn Peters

See Also

`mvrnorm`

Examples

```
dat <- simDataSet(500, varNames=c('age',
                                  'sex',
                                  'educationLevel',
                                  'negativeLifeEventsInPast10Years',
                                  'problemCoping',
```

```
                'emotionCoping',
                'resilience',
                'depression'),
means = c(40,
          0,
          0,
          5,
          3.5,
          3.5,
          3.5,
          3.5),
sds = c(10,
        1,
        1,
        1.5,
        1.5,
        1.5,
        1.5,
        1.5),
specifiedCorrelations =
  list(c('problemCoping', 'emotionCoping', -.5),
       c('problemCoping', 'resilience', .5),
       c('problemCoping', 'depression', -.4),
       c('depression', 'emotionCoping', .6),
       c('depression', 'resilience', -.3)),
ranges = list(age = c(18, 54),
              negativeLifeEventsInPast10Years = c(0,8),
              problemCoping = c(1, 7),
              emotionCoping = c(1, 7)),
factors=c("sex", "educationLevel"),
cuts=list(c(0),
          c(-.5, .5)),
labels=list(c('female', 'male'),
            c('lower', 'middle', 'higher')),
silent=FALSE);
```

`Singh` *Verbal and physical aggression scores from Singh et al. (2007)*

Description

This is a dataset originally described in Singh et al. (2007), and digitized by Rumen Manolov using plot digitizer software and used to illustrate a number of single case design analysis approaches in Manolov & Moeyaert (2016). It is also used by Verboon & Peters (2017) to illustrate the `piecewiseRegr` and the `genlog` functions.

Usage

```
data("Singh")
```

Format

A data frame with 56 observations on the following 6 variables.

`tier` A numeric subject identifier.

`id` A character subject identifier (i.e. a name).

`time` An index of the measurement moment.

`phase` A dummy variable indicating the phase of the experiment: 0 means that treatment has not yet started, 1 means that treatment has started.

`score_physical` The subjects' scores on physical aggression.

`score_verbal` The subjects' scores on verbal aggression.

Source

See Rumen Manolov's Open Science Framework repository at `https://osf.io/t6ws6` for the tutorial and the original dataset.

References

Singh, N. N., Lancioni, G. E., Winton, A. S., Adkins, A. D., Wahler, R. G., Sabaawi, M., & Singh, J. (2007). Individuals with mental illness can control their aggressive behavior through mindfulness training. *Behavior Modification, 31*(3), 313-328. http://doi.org/10.1177/0145445506293585

Manolov, R., & Moeyaert, M. (2017). How Can Single-Case Data Be Analyzed? Software Resources, Tutorial, and Reflections on Analysis. *Behavior Modification, 41*(2), 179-228. http://doi.org/10.1177/014544551666

Verboon, P. & Peters, G.-J. Y. (2017) Applying the generalised logistic model in SCD to deal with ceiling effects. *PsyArXiv* http://INSERTLINK

See Also

`piecewiseRegr` and `genlog` both contain examples using this dataset.

Examples

```
### To load the data, use:
data(Singh);
```

`sort.associationMatrix`

sort.associationMatrix

Description

This function sorts an `associationMatrix` ascendingly or descendingly by one of its columns.

Usage

```
## S3 method for class 'associationMatrix'
sort(x, decreasing = TRUE, byColumn = 1, ...)
```

Arguments

`x`	The `associationMatrix` object to sort.
`decreasing`	Whether to sort ascendingly (FALSE) or descending (TRUE).
`byColumn`	Which column to sort the matrix by, as an index.
`...`	Passed on to `sort`.

Details

Note that if the `associationMatrix` contains values of different effectsizes, the sorting may be misleading. For example, a value of Cohen's d of .45 is higher than a value of Pearson's r of .35, and so will end up higher in a 'decreasing' sort - even though the association represented by an r of .35 is stronger than that represented by a d of .45.

Furthermore, only asymmetrical associationMatrices can be sorted; sorting a symmetrical association matrix would also change the order of the columns, after all.

Value

The `associationMatrix`, but sorted.

Author(s)

Gjalt-Jorn Peters

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

See Also

`associationMatrix`

Examples

```
sort(associationMatrix(infert, y=c("parity", "age"),
                         x=c("induced", "case", "spontaneous"), colNames=TRUE));
```

`testRetestAlpha` *testRetestAlpha*

Description

The testRetestAlpha function computes the test-retest alpha coefficient (Green, 2003).

Usage

```
testRetestAlpha(dat = NULL, moments = NULL,
                testDat = NULL, retestDat = NULL,
                sortItems = FALSE, convertToNumeric = TRUE)
```

Arguments

`dat`	A dataframe containing the items in the scale at both measurement moments. If no dataframe is specified, a dialogue will be launched to allow the user to select an SPSS datafile. If only one dataframe is specified, either the items have to be ordered chronologically (i.e. first all items for the first measurement, then all items for the second measurement), or the vector 'moments' has to be used to indicate, for each item, to which measurement moment it belongs.
`moments`	Used to indicate to which measurement moment each item in 'dat' belongs; should be a vector with the same length as dat has columns, and with two possible values (e.g. 1 and 2).
`testDat, retestDat`	Dataframes with the items for each measurement moment: note that the items have to be in the same order (unless sortItems is TRUE).
`sortItems`	If true, the columns (items) in each dataframe are ordered alphabetically before starting. This can be convenient to ensure that the order of the items at each measurement moment is the same.
`convertToNumeric`	When TRUE, the function will attempt to convert all vectors in the dataframes to numeric.

Details

This function computes the test-retest alpha coefficient as described in Green (2003).

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$testRetestAlpha`	The value of the test-retest alpha coefficient.

Author(s)

Gjalt-Jorn Peters

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

References

Green, S. N. (2003). A Coefficient Alpha for Test-Retest Data. Psychological Methods, 8(1), 88-101. doi:10.1037/1082-989X.8.1.88

Examples

```
## Not run:
### This will prompt the user to select an SPSS file
testRetestAlpha();

## End(Not run)

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

### The first column is the true score, so it's excluded in this example.
exampleData <- testRetestSimData[, 2:ncol(testRetestSimData)];

### Compute test-retest alpha coefficient
testRetestAlpha(exampleData);
```

`testRetestCES` *testRetestCES*

Description

The testRetestCES function computes the test-retest Coefficient of Equivalence and Stability (Schmidt, Le & Ilies, 2003).

Usage

```
testRetestCES(dat = NULL, moments = NULL,
              testDat = NULL, retestDat = NULL,
              parallelTests = 'means',
              sortItems = FALSE, convertToNumeric = TRUE,
              digits=4)
parallelSubscales(dat, convertToNumeric = TRUE)
```

Arguments

`dat`	A dataframe. For testRetestCES, this dataframe must contain the items in the scale at both measurement moments. If no dataframe is specified, a dialogue will be launched to allow the user to select an SPSS datafile. If only one dataframe is specified, either the items have to be ordered chronologically (i.e. first all items for the first measurement, then all items for the second measurement), or the vector 'moments' has to be used to indicate, for each item, to which measurement moment it belongs. The number of columns in this dataframe MUST be even! Note that instead of providing this dataframe, the items of each measurement moment can be provided separately in testDat and retestDat as well.

`moments`	Used to indicate to which measurement moment each item in 'dat' belongs; should be a vector with the same length as dat has columns, and with two possible values (e.g. 1 and 2).
`testDat, retestDat`	Dataframes with the items for each measurement moment: note that the items have to be in the same order (unless sortItems is TRUE).
`parallelTests`	A vector indicating which items belong to which parallel test; like the moments vector, this should have two possible values (e.g. 1 and 2). Alternatively, it can be character value with 'means' or 'variances'; in this case, parallelSubscales will be used to create roughly parallel halves.
`sortItems`	If true, the columns (items) in each dataframe are ordered alphabetically before starting. This can be convenient to ensure that the order of the items at each measurement moment is the same.
`convertToNumeric`	When TRUE, the function will attempt to convert all vectors in the dataframes to numeric.
`digits`	Number of digits to print.

Details

This function computes the test-retest Coefficient of Equivalence and Stability (CES) as described in Schmidt, Le & Ilies (2003). Note that this function only computes the test-retest CES for a scale that is administered twice and split into two parallel halves post-hoc (this procedure is explained on page 210, and the equations that are used, 16 and 17a are explained on page 212).

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$testRetestCES`	The value of the test-retest Coefficient of Equivalence and Stability.

Note

This function uses equations 16 and 17 on page 212 of Schmidt, Le & Ilies (2003): in other words, this function assumes that one scale is administered twice. If you'd like the computation for two different but parellel scales/measures to be implemented, please contact me.

Author(s)

Gjalt-Jorn Peters

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

References

Schmidt, F. L., Le, H., & Ilies, R. (2003) Beyong Alpha: An Empirical Examination of the Effects of Different Sources of Measurement Error on Reliability Estimates for Measures of Individual-differences Constructs. Psychological Methods, 8(2), 206-224. doi:10.1037/1082-989X.8.x.206

Examples

```
## Not run:
### This will prompt the user to select an SPSS file
testRetestCES();

## End(Not run)

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

### The first column is the true score, so it's excluded in this example.
exampleData <- testRetestSimData[, 2:ncol(testRetestSimData)];

### Compute test-retest alpha coefficient
testRetestCES(exampleData);
```

`testRetestReliability` *testRetestReliability*

Description

The testRetestReliability function is a convenient interface to testRetestAlpha and testRetestCES.

Usage

```
testRetestReliability(dat = NULL, moments = NULL,
                      testDat = NULL, retestDat = NULL,
                      parallelTests = 'means',
                      sortItems = FALSE, convertToNumeric = TRUE,
                      digits=2)
```

Arguments

`dat`	A dataframe. This dataframe must contain the items in the scale at both measurement moments. If no dataframe is specified, a dialogue will be launched to allow the user to select an SPSS datafile. If only one dataframe is specified, either the items have to be ordered chronologically (i.e. first all items for the first measurement, then all items for the second measurement), or the vector 'moments' has to be used to indicate, for each item, to which measurement moment

	it belongs. The number of columns in this dataframe MUST be even! Note that instead of providing this dataframe, the items of each measurement moment can be provided separately in testDat and retestDat as well.
`moments`	Used to indicate to which measurement moment each item in 'dat' belongs; should be a vector with the same length as dat has columns, and with two possible values (e.g. 1 and 2).
`testDat, retestDat`	Dataframes with the items for each measurement moment: note that the items have to be in the same order (unless sortItems is TRUE).
`parallelTests`	A vector indicating which items belong to which parallel test; like the moments vector, this should have two possible values (e.g. 1 and 2). Alternatively, it can be character value with 'means' or 'variances'; in this case, parallelSubscales will be used to create roughly parallel halves.
`sortItems`	If true, the columns (items) in each dataframe are ordered alphabetically before starting. This can be convenient to ensure that the order of the items at each measurement moment is the same.
`convertToNumeric`	When TRUE, the function will attempt to convert all vectors in the dataframes to numeric.
`digits`	Number of digits to show when printing the output

Details

This function calls both testRetestAlpha and testRetestCES to compute and print measures of the test-retest reliability.

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$testRetestAlpha`	The value of the test-retest alpha coefficient.
`output$testRetestCES`	The value of the test-retest Coefficient of Equivalence and Stability.

Author(s)

Gjalt-Jorn Peters

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

Examples

```
## Not run:
### This will prompt the user to select an SPSS file
testRetestReliability();
```

```
## End(Not run)

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

### The first column is the true score, so it's excluded in this example.
exampleData <- testRetestSimData[, 2:ncol(testRetestSimData)];

### Compute test-retest alpha coefficient
testRetestReliability(exampleData);
```

`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).

Usage

`data(testRetestSimData)`

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);
```

`therapyMonitor` *therapyMonitor & therapyMonitor.multi*

Description

therapyMonitor & therapyMonitor.multi are useful for simple n-of-1 designs, and were written to make it easy for therapists or other practitioners to get some insight into the effects of their treatments.

Usage

```
therapyMonitor(dat = NULL, design="AB", statistic="|A-B|",
               conditionColumn = NULL, variableColumn = NULL,
               timeColumn = NULL, conditionMoment = NULL,
               limit=NULL, lines=NULL, ylab=NULL, xlab=NULL,
               outputFile = NULL, outputFormats = c('svg', 'png'),
               plotTitle = "therapyMonitor results",
               plotWidth=25, plotHeight=15)
therapyMonitor.multi(dat = NULL,
                     variableColumn = NULL, conditionColumn = NULL,
```

```
                 conditionMoment = NULL, minLevels = 5,
                 outputFiles = FALSE, outputFilePath = getwd(),
                 outputFormats = c('svg', 'png'), silent=FALSE,
                 ...)
```

Arguments

`dat`	A dataframe containing the variables to analyse. If not dataframe is specified, get `getData` function is used to present a dialog to the user.
`design`	The design to use; see `pvalue.systematic` in the `SCRT-package` for more information. Note that currently, this function always assumes an "AB" design; changing this only changes the way `pvalue.systematic` is called.
`statistic`	The statistic to use; see `pvalue.systematic` in the `SCRT-package` for more information. Note that currently, this function always assumes the "|A-B|" statistic; changing this only changes the way `pvalue.systematic` is called.
`conditionColumn`	The name of the variable containing, for each measurement, the condition, or the phase of the treatment. This variable should normally only have two levels (e.g. 'A' and 'B'), indicating when the treatment changed from condition 'A' to condition 'B'.
`variableColumn`	For `therapyMonitor`, this must be a single value: the name of the variable to analyse as dependent variable. For `therapyMonitor.multi`, this can be a vector, in which case all the specified variables are analysed sequentially. In any case, the variable(s) specified here must have the 'interval' measurement level (i.e. be roughly continuous). For `therapyMonitor.multi`, if this argument is empty, all variables are used, provided they have at least `minLevels` levels.
`timeColumn`	The variable containing the time (datetime) of each measurement moment. If not specified in R's POSIXct format, the function tries to guess whether SPSS, SAS, or Stata timestamps were specified, and tries to convert. If the timeColumn isn't specified, the function will assume that all measurements were equidistant, and they'll simply be assigned consecutive numbers als measurement moments.
`conditionMoment`	The conditionMoment argument provides an alternative method of specifying when the condition changed; this can be the number of the first measurement in the new (second) condition/phase. For example, if the treatment started after the 6th measurement, this can be specified by passing 'conditionMoment=7'.
`limit`	The minimum number of consecutive measurements that has to be available within one condition/phase to enable the analysis (see `pvalue.systematic`).
`lines`	Which lines in the `dat` dataframe to use.
`ylab, xlab`	Labels to use when creating the plots.
`outputFile`	If not NULL, the filename to write the plot to. Note that this filename should not include the extension - this is appended based on the `outputFormats` argument.
`outputFormats`	Which format to use for the plot or plots to export.
`plotTitle`	The title for the plot.
`plotWidth, plotHeight`	The size of the plot (in centimeters).

`minLevels`	The minimum number of levels that a variable in the datafile has to have before it's included in the analyses.
`outputFiles`	Whether to export the plots and regular output to files.
`outputFilePath`	If `outputFiles` is TRUE, the path where to store the output files.
`silent`	Whether to suppress messages about progress etc.
`...`	Additional arguments to `therapyMonitor.multi` are passed on to `therapyMonitor`.

Details

This function started as a wrapper to the `pvalue.systematic` function in the `SCRT-package`, but it now also does some extra stuff.

Value

For therapyMonitor, an object with the input and several output variables, as well as a plot. For therapyMonitor.multi, an object containing several therapyMonitor objects, as well as collated output.

Author(s)

Gjalt-Jorn Peters

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

Examples

```
### Explore and plot the weight of a chick in the ChickWeight dataset
therapyMonitor(ChickWeight, variableColumn='weight',
               conditionMoment=6, lines=1:12);
```

`therapyMonitorData` *Data originally published with therapyMonitor*

Description

This dataset was originally published along with a Dutch language article that described the `therapyMonitor` function. This version only contains the aggregated scales.

Usage

```
data("therapyMonitorData")
```

Format

A data frame with 38 observations on the following 12 variables.

`time` The measurement moment as stored by Google Forms.

`datetime` The measurement moment converted to POSIXct, R's time format.

`measurementNumber` The rank (number) of each measurement.

`positiveAffect` The positive affect scale.

`negativeAffect` The negative affect scale.

`selfEsteem` A self esteem scale.

`intimacy` An intimacy scale.

`erectionMasturbation` Erection when masturbating.

`erectionPartnerSex` Erection while having sex with partner.

`experienceMasturbation` Experience when masturbating.

`experiencePartnerSex` Experience while having sex with partner.

`erectionCombined` Aggregated scale of both erection experience scales.

Details

This dataset is an n-of-1 dataset collected during a series of therapy sessions.

Source

van Lankveld, J., Leusink, P., & Peters, G.-J. Y. (2017). Therapie-monitoring in een blended online en face-to-face behandeling van een jonge man met situatieve erectieproblemen. *Tijdschrift voor Seksuologie, 41*, 15-22.

Examples

```
data(therapyMonitorData)
## maybe str(therapyMonitorData) ; plot(therapyMonitorData) ...
```

`userfriendlyscienceBasics` *userfriendlyscience basics*

Description

The userfriendlyscience basics functions are some very basic functions to make life that little bit easier.

Usage

```
safeRequire(packageName, mirrorIndex=NULL)
trim(str)
noZero(str)
formatPvalue(values, digits = 3, spaces=TRUE, includeP = TRUE)
formatR(r, digits)
repStr(n = 1, str = " ")
repeatStr(n = 1, str = " ")
ifelseObj(condition, ifTrue, ifFalse)
invertItem(item, fullRange=NULL, ignorePreviousInversion = FALSE)
is.odd(vector)
is.even(vector)
convertToNumeric(vector, byFactorLabel = FALSE)
massConvertToNumeric(dat, byFactorLabel = FALSE,
                     ignoreCharacter = TRUE,
                     stringsAsFactors = FALSE)
vecTxt(vector, delimiter = ", ", useQuote = "",
      firstDelimiter = NULL, lastDelimiter = " & ",
      firstElements = 0, lastElements = 1, lastHasPrecedence = TRUE)
vecTxtQ(vector, useQuote = "'", ...)
find %IN% table
cat0(..., sep="")
addToLog(fullLog, ..., showLog = FALSE);
```

Arguments

packageName	The name of the package, as character string.
mirrorIndex	The index of the mirror to use, in case you want to specify the mirror in the call (see e.g. /code/linkgetCRANmirrors()[, 1:4] for an overview of these mirrors. For example, at the time of writing, Antwerp is 7, Amsterdam is 60, and Auckland is 62).
str	The character string to process.
values	The p-values to format.
digits	For formatPvalue, number of digits to round to. Numbers smaller than this number will be shown as <.001 or <.0001 etc. For formatR, the number of digits to use when formatting the Pearson correlation.
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).
r	The Pearson correlation to format.
n	The number of times to repeat the string.
condition	Condition to evaluate.
ifTrue	Object to return if the condition is true.
ifFalse	Object to return if the condition is false.

`item`	Item to invert
`fullRange`	If provided it must be a numeric vector with the minimum and the maximum of the scale. If not provided, the range function is used (so, use this range argument if the scale minimum and/or maximum do not occur in the data).
`ignorePreviousInversion`	If this item has already been inverted, the function will halt with an error unless it's told to ignore previous inversions with this boolean.
`dat, vector`	The dataframe of vector to process.
`byFactorLabel`	If TRUE, convertToNumeric and massConvertToNumeric use the factor labels, interpreted as character vectors, to determine the numeric value, instead of the level's indices (which is what as.numeric() does).
`ignoreCharacter`	If TRUE, character vectors are ignored. If FALSE, character vectors are converted (or, an attempt is made :-)).
`stringsAsFactors`	If TRUE, strings (character vectors) in the dataframe will be converted to factors (by as.data.frame, after the function called lapply).
`find`	The element(s) to look up in the vector or matrix.
`table`	The vector or matrix in which to look up the element.
`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`)?
`sep`	The separator to pass to `cat`, of course, "" by default.
`fullLog`	The full log - the character vector(s) provided are appended to this character vector.
`showLog`	Whether to `cat` the log.
`...`	Extra arguments are passed to whatever function is wrapped (e.g. `cat` for `cat0`). For addToLog, the dots are used to provide character vectors that are concatenated using `paste0` and (potentially shown and) added to the log.

Details

The safeRequire function checks whether a package is already installed. If so, it loads the package (using require/library). If not, it first installs it, and then loads it.

The trim function removes whitespaces from the start and end of a text string.

The noZero function removes the first zero from a string that was originally a number.

The formatPvalue function formats a P value, roughly according to APA style guidelines. This means that the noZero 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.

The formatR function format a Pearson correlation for pretty printing (using noZero).

The repeatStr (or repStr) function repeats a string a given number of times.

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

The invertItem function 'unmirrors' an inverted item (i.e. for a 1-3 item, 1 becomes 3, 2 stays 2, and 3 becomes 1).

is.odd and is.even check whether a number is, or numbers in a vector are, odd or even.

The infix function %IN% is a case-insensitive version of `%in%`.

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

The addToLog function adds a character vector to a log.

Value

safeRequire returns nothing.

trim, formatPvalue, noZero, formatR, and repeatStr return a string.

ifelseObj return an object.

The invertItem function returns the inverted item vector, with an attribute "inverted" set to TRUE.

is.odd and is.even return a logical vector.

%IN% returns a logical vector of the same length as its first argument.

cat0 returns a string.

addToLog returns a string.

Author(s)

Gjalt-Jorn Peters

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

Examples

```
### load a package
safeRequire('ggplot2');

### trim a string
trim(' this is a string with whitespace in front and at the end               ');
### Returns "this is a string with whitespace in front and at the end"

repeatStr("-", 8);
```

```
### Returns "--------" (incredibly useful, no? :-))

tempVector <- c(1,2,3,3,2,4,3,2,1,1,3,4,5,4,3,2,2,1,1,2);
invertedTempVector <- invertItem(tempVector);

### We can also invert it back, but then we have to override the security
### that prevents accidently inverting items back.
invertItem(tempVector, ignorePreviousInversion=TRUE);
```

---

userfriendlysciencePanderMethods

userfriendlyscience methods to pander objects

Description

These methods try to provide output that's ready for R Markdown. Note that they are not all documented; most of them are quite straightforward.

Usage

```
## S3 method for class 'freq'
pander(x, ...)
## S3 method for class 'meanDiff'
pander(x, digits=x$digits, powerDigits=x$digits + 2, ...)
## S3 method for class 'normalityAssessment'
pander(x, headerPrefix = "#####", suppressPlot = FALSE, ...)
## S3 method for class 'dataShape'
pander(x, digits=x$input$digits, extraNotification=TRUE, ...)
## S3 method for class 'associationMatrix'
pander(x, info = x$input$info, file = x$input$file, ...)
## S3 method for class 'crossTab'
pander(x, digits = x$input$digits,
                          pValueDigits=x$input$pValueDigits, ...)
## S3 method for class 'oneway'
pander(x, digits = x$input$digits,
                        pvalueDigits=x$input$pvalueDigits,
                        headerStyle = "**",
                        na.print="", ...)
## S3 method for class 'regr'
pander(x, digits=x$input$digits,
                      pvalueDigits=x$input$pvalueDigits, ...)
## S3 method for class 'descr'
pander(x, headerPrefix = "", headerStyle = "**", ...)
## S3 method for class 'examine'
pander(x, headerPrefix = "", headerStyle = "**",
                         secondaryHeaderPrefix = "", secondaryHeaderStyle =
                         "*", ...)
```

```
## S3 method for class 'examineBy'
pander(x, headerPrefix = "", headerStyle = "**",
                           secondaryHeaderPrefix = "", secondaryHeaderStyle =
                           "*", tertairyHeaderPrefix = "--> ",
                           tertairyHeaderStyle = "", separator = paste0("\n\n",
                           repStr("-", 10), "\n\n"), ...)
## S3 method for class 'frequencies'
pander(x, prefix = "###", ...)
```

Arguments

`x`	The object to print.
`digits`	The number of significant digits to print.
`powerDigits`	Number of digits to use when printing the power.
`headerPrefix, secondaryHeaderPrefix, tertairyHeaderPrefix, prefix`	Prefix for headers, can be used to output headers for pandoc using R Markdown by specifying e.g. '####' for a level 4 header.
`headerStyle, secondaryHeaderStyle, tertairyHeaderStyle`	A character value to pre- and append to the header. This can be used to make the header appear bold ('**') or italic ('*') when not using an actual header (see `headerPrefix`).
`separator`	Separator to show between sections of output.
`suppressPlot`	Whether to suppress printing plots.
`pValueDigits`	Output to produce; see /code/linkrMatrix for details.
`info, file`	Output to produce and file to write to; see /code/linkassociationMatrix for details.
`extraNotification`	Whether an extra notification about the type of skewness and kurtosis returned by dataShape is shown.
`pvalueDigits`	The number of digits to show for p-values; smaller p-values will be shown as <.001 or <.0001 etc.
`na.print`	What to print for missing values, for example for a oneway anova table.
`...`	Additional arguments that are passed on to the print functions when it is called.

Value

These printing methods use `cat`, `cat0`, and `grid.draw` to print stuff.

Author(s)

Gjalt-Jorn Peters

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

Examples

```
userfriendlyscience:::pander.oneway(oneway(y=ChickWeight$weight, x=ChickWeight$Diet));
```

---

userfriendlysciencePrintMethods
userfriendlyscience print methods

Description

These methods print the userfriendlyscience objects. Note that they are not all documented; most of them are quite straightforward.

Usage

```
## S3 method for class 'dlvPlot'
print(x, ...)
## S3 method for class 'freq'
print(x, digits=x$input$digits, nsmall=x$input$nsmall,
transposed=x$input$transposed, ...)
## S3 method for class 'meanConfInt'
print(x, digits=2, ...)
## S3 method for class 'meanDiff'
print(x, digits=x$digits,powerDigits=x$digits + 2, ...)
## S3 method for class 'meanDiff.multi'
print(x, digits=x$digits,
powerDigits=x$digits + 2, ...)
## S3 method for class 'normalityAssessment'
print(x, ...)
## S3 method for class 'oddsratio'
print(x, digits=x$input$digits, ...)
## S3 method for class 'powerHist'
print(x, ...)
## S3 method for class 'rMatrix'
print(x, digits=x$digits, output=x$output,
pValueDigits = x$pValueDigits, env.LaTeX = x$env.LaTeX,
pboxWidthMultiplier = x$pboxWidthMultiplier,
colNames = x$colNames, ...)
## S3 method for class 'scaleDiagnosis'
print(x, ...)
## S3 method for class 'scaleStructure'
print(x, digits=x$input$digits, ...)
## S3 method for class 'sdConfInt'
print(x, digits=2, ...)
## S3 method for class 'testRetestAlpha'
print(x, ...)
## S3 method for class 'testRetestCES'
print(x, digits=x$input$digits, ...)
## S3 method for class 'testRetestReliability'
print(x, digits=x$input$digits, ...)
## S3 method for class 'parallelSubscales'
```

```
print(x, nsmall=2, ...)
## S3 method for class 'dataShape'
print(x, digits=x$input$digits, extraNotification=TRUE, ...)
## S3 method for class 'didacticPlot'
print(x, ...)
## S3 method for class 'scatterMatrix'
print(x, ...)
## S3 method for class 'CramersV'
print(x, digits = x$input$digits, ...)
## S3 method for class 'associationMatrix'
print(x, type = x$input$type,
                                  info = x$input$info,
                                  file = x$input$file, ...)
## S3 method for class 'confIntV'
print(x, digits = x$input$digits, ...)
## S3 method for class 'cohensdCI'
print(x, ...)
## S3 method for class 'crossTab'
print(x, digits = x$input$digits,
                         pValueDigits=x$input$pValueDigits, ...)
## S3 method for class 'oneway'
print(x, digits = x$input$digits,
                       pvalueDigits=x$input$pvalueDigits,
                       na.print="", ...)
## S3 method for class 'posthocTGH'
print(x, digits = x$input$digits, ...)
## S3 method for class 'scaleInspection'
print(x, show=x$show, ...)
## S3 method for class 'regr'
print(x, digits=x$input$digits,
                     pvalueDigits=x$input$pvalueDigits, ...)
## S3 method for class 'processOpenSesameIAT'
print(x, ...)
## S3 method for class 'processOpenSesameIAT.log'
print(x, ...)
## S3 method for class 'descr'
print(x, digits = attr(x, 'digits'),
                      t = attr(x, "transpose"),
                      row.names = FALSE, ...)
## S3 method for class 'therapyMonitor'
print(x, digits=2, printPlot = TRUE, ...)
## S3 method for class 'therapyMonitor.multi'
print(x, ...)
## S3 method for class 'asymmetricalScatterMatrix'
print(x, ...)
## S3 method for class 'fullFact'
print(x, ...)
## S3 method for class 'confIntOmegaSq'
```

```
print(x, ..., digits = 2)
## S3 method for class 'examine'
print(x, ...)
## S3 method for class 'examineBy'
print(x, ...)
## S3 method for class 'frequencies'
print(x, ...)
## S3 method for class 'power.htest.ufs'
print(x, digits = x$digits, ...)
## S3 method for class 'regrInfluential'
print(x, ...)
## S3 method for class 'nnc'
print(x, digits=2, ...)
## S3 method for class 'fanova'
print(x, digits = x$input$digits, ...)
## S3 method for class 'genlog'
print(x, digits = 3, ...)
## S3 method for class 'ggProportionPlot'
print(x, ...)
## S3 method for class 'logRegr'
print(x, digits = x$input$digits,
                        pvalueDigits = x$input$pvalueDigits,
                        ...)
## S3 method for class 'piecewiseRegr'
print(x, digits = x$input$digits, ...)
```

Arguments

`x`	The object to print.
`digits`	The number of significant digits to print.
`nsmall`	The minimum number of digits to the right of the decimal point in formatting real/complex numbers in non-scientific formats. Allowed values are 0 <= nsmall <= 20.
`transposed, t`	Whether the frequency object should be printed transposed (this can be useful for blind users).
`powerDigits`	Number of digits to use when printing the power.
`output, env.LaTeX, pboxWidthMultiplier, colNames, pValueDigits`	Output to produce; see /code/linkrMatrix for details.
`type, info, file`	Output to produce and file to write to; see /code/linkassociationMatrix for details.
`extraNotification`	Whether an extra notification about the type of skewness and kurtosis returned by dataShape is shown.
`pvalueDigits`	The number of digits to show for p-values; smaller p-values will be shown as <.001 or <.0001 etc.
`printPlot`	Whether to also print the plot.

`na.print`	What to print for missing values, for example for a oneway anova table.
`show`	To override the 'show' argument, which is sometimes used to inhibit printing of extensive information with e.g. many plots, which is useful for some functions that, for example, primarily generate a PDF.
`row.names`	Whether to print rownames.
`...`	Addition arguments that are passed on to the print functions when it's called.

Value

These printing methods return nothing, but print stuff.

Author(s)

Gjalt-Jorn Peters

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

Examples

```
print(sdConfInt(sd=4, n=20));

print(oneway(y=ChickWeight$weight, x=ChickWeight$Diet), na.print="[NO VALUE]");
```

`validComputations` *Only compute means or sums for cases with enough nonmissings*

Description

These functions have been written as equivalents of SPSS' `MEAN.x` and `SUM.x` functions, which only compute means and sums if enough cases have valid values.

Usage

```
validMeans(...,
           requiredValidValues = 0,
           returnIfInvalid = NA,
           silent = FALSE)
validSums(...,
          requiredValidValues = 0,
          returnIfInvalid = NA,
          silent = FALSE)
```

Arguments

`...`	Either a dataframe or vectors for which to compute the mean or sum.
`requiredValidValues`	How many values must be valid (i.e. nonmissing) to compute the mean or sum. If a number lower than 1 is provided, it is interpreted as proportion, and the number of variables is computed. For example, if `requiredValidValues=.8`, 80% of the variables must have valid values. If 'all' is specified, all values must be valid (in which case the functions are equal to `rowMeans` and `rowSums`).
`returnIfInvalid`	Wat to return for cases that don't have enough valid values.
`silent`	Whether to show the number of cases that have to be valid if `requiredValidValues` is a proportion.

Value

A numeric vector with the resulting means or sums.

Author(s)

Gjalt-Jorn Peters

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

See Also

`rowMeans`, `rowSums`

Examples

```
validMeans(mtcars$cyl, mtcars$disp);
validSums(mtcars$cyl, mtcars$disp, requiredValidValues = .8);

### Or specifying a dataframe
validSums(mtcars);
```

Index