Package ‘misty’

June 8, 2020

Type Package

Title Miscellaneous Functions 'T. Yanagida'

Version 0.3.2

Date 2020-06-08

Author Takuya Yanagida [aut, cre]

Maintainer Takuya Yanagida <takuya.yanagida@univie.ac.at>

Description Miscellaneous functions for descriptive statistics (e.g., frequency table, cross tabulation, multilevel descriptive statistics, coefficient alpha and omega, and various effect size measures), missing data (e.g., descriptive statistics for missing data, missing data pattern and auxiliary variable analysis), data management (e.g., grand-mean and group-mean centering, recode variables and reverse code items, scale and group scores, reading and writing SPSS and Excel files), and statistical analysis (e.g., confidence intervals, collinearity diagnostics, Levene's test, z-test, and sample size determination).

Depends R (>= 3.3.3)

License MIT + file LICENSE

Imports haven, lavaan, lme4, readxl

Suggests mnormt, nlme, plyr, R.rsp

VignetteBuilder R.rsp

Encoding UTF-8

RoxygenNote 7.1.0

NeedsCompilation no

Repository CRAN

Date/Publication 2020-06-08 14:20:02 UTC

R topics documented:

`alpha.coef` *Coefficient Alpha and Item Statistics*

Description

This function computes point estimate and confidence interval for the (ordinal) coefficient alpha (aka Cronbach's alpha) along with the corrected item-total correlation and coefficient alpha if item deleted.

Usage

```
alpha.coef(x, exclude = NULL, std = FALSE, ordered = FALSE, na.omit = FALSE,
           print = c("all", "alpha", "item"), digits = 2, conf.level = 0.95,
           as.na = NULL, check = TRUE, output = TRUE)
```

Arguments

`x`	a matrix, data frame, variance-covariance or correlation matrix. Note that raw data is needed to compute ordinal coefficient alpha, i.e., `ordered = TRUE`.
`exclude`	a character vector indicating items to be excluded from the analysis.
`std`	logical: if `TRUE`, the standardized coefficient alpha is computed.
`ordered`	logical: if `TRUE`, variables are treated as ordered (ordinal) variables to compute ordinal coefficient alpha.
`na.omit`	logical: if `TRUE`, incomplete cases are removed before conducting the analysis (i.e., listwise deletion); if `FALSE`, pairwise deletion is used.
`print`	a character vector indicating which results to show, i.e. `"all"` (default), for all results `"alpha"` for the coefficient alpha, and `"item"` for item statistics.
`digits`	an integer value indicating the number of decimal places to be used for displaying coefficient alpha and item-total correlations.
`conf.level`	a numeric value between 0 and 1 indicating the confidence level of the interval.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown.

Details

Ordinal coefficient alpha was introduced by Zumbo, Gadermann and Zeisser (2007) which is obtained by applying the formula for computing coefficient alpha to the polychoric correlation matrix instead of the variance-covariance or product-moment correlation matrix. Note that Chalmers (2018) highlighted that the ordinal coefficient alpha should be interpreted only as a hypothetical estimate of an alternative reliability, whereby a test's ordinal categorical response options have be modified to include an infinite number of ordinal response options and concludes that coefficient alpha should not be reported as a measure of a test's reliability. However, Zumbo and Kroc (2019) argued that Chalmers' critique of ordinal coefficient alpha is unfounded and that ordinal coefficient alpha may be the most appropriate quantifier of reliability when using Likert-type measurement to study a latent continuous random variable. Confidence intervals are computed using the procedure by Feldt, Woodruff and Salih (1987). When computing confidence intervals using pairwise deletion, the average sample size from all pairwise samples is used. Note that there are at least 10 other procedures for computing the confidence interval (see Kelley and Pornprasertmanit, 2016), which are implemented in the `ci.reliability()` function in the **MBESSS** package by Ken Kelley (2019).

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, matrix or data frame specified in `x` (`data`), specification of function arguments (`args`), and list with results (`result`).

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Chalmers, R. P. (2018). On misconceptions and the limited usefulness of ordinal alpha. *Educational and Psychological Measurement, 78*, 1056-1071. https://doi.org/10.1177/0013164417727036

Cronbach, L.J. (1951). Coefficient alpha and the internal structure of tests. *Psychometrika, 16*, 297-334. https://doi.org/10.1007/BF02310555

Cronbach, L.J. (2004). My current thoughts on coefficient alpha and successor procedures. *Educational and Psychological Measurement, 64*, 391-418. https://doi.org/10.1177/0013164404266386

Feldt, L. S., Woodruff, D. J., & Salih, F. A. (1987). Statistical inference for coefficient alpha. *Applied Psychological Measurement*, 11 93-103. https://doi.org/10.1177/014662168701100107

Kelley, K., & Pornprasertmanit, S. (2016). Confidence intervals for population reliability coefficients: Evaluation of methods, recommendations, and software for composite measures. *Psychological Methods, 21*, 69-92. https://doi.org/10.1037/a0040086.

Ken Kelley (2019). *MBESS: The MBESS R Package*. R package version 4.6.0. https://CRAN.R-project.org/package=MBESS

Zumbo, B. D., & Kroc, E. (2019). A measurement is a choice and Stevens' scales of measurement do not help make it: A response to Chalmers. *Educational and Psychological Measurement, 79*, 1184-1197. https://doi.org/10.1177/0013164419844305

Zumbo, B. D., Gadermann, A. M., & Zeisser, C. (2007). Ordinal versions of coefficients alpha and theta for Likert rating scales. *Journal of Modern Applied Statistical Methods, 6*, 21-29. https://doi.org/10.22237/jmasm/1177992180

See Also

omega.coef, reverse.item, scores

Examples

```
dat <- data.frame(item1 = c(4, 2, 3, 4, 1, 2, 4, 2),
                  item2 = c(4, 3, 3, 3, 2, 2, 4, 1),
                  item3 = c(3, 2, 4, 2, 1, 3, 4, 1),
                  item4 = c(4, 1, 2, 3, 2, 3, 4, 2))

# Compute unstandardized coefficient alpha and item statistics
alpha.coef(dat)

# Compute standardized coefficient alpha and item statistics
alpha.coef(dat, std = TRUE)

# Compute unstandardized coefficient alpha
alpha.coef(dat, print = "alpha")

# Compute item statistics
alpha.coef(dat, print = "item")

# Compute unstandardized coefficient alpha and item statistics while excluding item3
alpha.coef(dat, exclude = "item3")

# Compute variance-covariance matrix
dat.cov <- cov(dat)
# Compute unstandardized coefficient alpha based on the variance-covariance matrix
alpha.coef(dat.cov)

# Compute correlation matrix
dat.cor <- cor(dat)
# Compute standardized coefficient alpha based on the correlation matrix
alpha.coef(dat.cor)

# Compute ordinal coefficient alpha
alpha.coef(dat, ordered = TRUE)
```

---

as.na *Replace User-Specified Values With Missing Values*

Description

This function replaces user-specified values in the argument `as.na` in a vector, factor, matrix, data frame or list with `NA`.

Usage

```
as.na(x, as.na, check = TRUE)
```

Arguments

x	a vector, factor, matrix, data frame, or list.
as.na	a vector indicating values or characters to replace with NA.
check	logical: if TRUE, argument specification is checked.

Value

Returns x with values specified in na replaced with NA.

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

na.as, na.auxiliary, na.coverage, na.descript, na.indicator, na.pattern, na.prop.

Examples

```
#--------------------------------------
# Numeric vector
x.num <- c(1, 3, 2, 4, 5)

# Replace 2 with NA
as.na(x.num, as.na = 2)

# Replace 2, 3, and 4 with NA
as.na(x.num, as.na = c(2, 3, 4))

#--------------------------------------
# Character vector
x.chr <- c("a", "b", "c", "d", "e")

# Replace "b" with NA
as.na(x.chr, as.na = "b")

# Replace "b", "c", and "d" with NA
as.na(x.chr, as.na = c("b", "c", "d"))

#--------------------------------------
# Factor
x.factor <- factor(c("a", "a", "b", "b", "c", "c"))

# Replace "b" with NA
as.na(x.factor, as.na = "b")
```

```
# Replace "b" and "c" with NA
as.na(x.factor, as.na = c("b", "c"))

#--------------------------------------
# Matrix
x.mat <- matrix(1:20, ncol = 4)

# Replace 8 with NA
as.na(x.mat, as.na = 8)

# Replace 8, 14, and 20 with NA
as.na(x.mat, as.na = c(8, 14, 20))

#--------------------------------------
# Data frame
x.df <- data.frame(x1 = c(1, 2, 3),
                   x2 = c(2, 1, 3),
                   x3 = c(3, 1, 2), stringsAsFactors = FALSE)

# Replace 1 with NA
as.na(x.df, as.na = 1)

# Replace 1 and 3 with NA
as.na(x.df, as.na = c(1, 3))

#--------------------------------------
# List
x.list <- list(x1 = c(1, 2, 3, 1, 2, 3),
               x2 = c(2, 1, 3, 2, 1),
               x3 = c(3, 1, 2, 3))

# Replace 1 with NA
as.na(x.list, as.na = 1)
```

---

center *Centering at the Grand Mean or Centering Within Cluster*

Description

This function is used to center predictors at the grand mean (CGM, i.e., grand mean centering) or within cluster (CWC, i.e., group-mean centering).

Usage

```
center(x, type = c("CGM", "CWC"), group = NULL, value = NULL, as.na = NULL,
       check = TRUE)
```

Arguments

`x`	a numeric vector.
`type`	a character string indicating the type of centering, i.e., `"CGM"` for centering at the grand mean (i.e., grand mean centering) or `"CWC"` for centering within cluster (i.e., group-mean centering).
`group`	a numeric vector, character vector or factor denoting the group membership of each unit in `x`. Note, this argument is required for centering at the grand mean (CGM) of a level-2 predictor or centering within cluster (CWC) of a level-1 predictor.
`value`	a numeric value for centering on a specific user-defined value.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis. Note that `as.na()` function is only applied to `x` but not to `group`.
`check`	logical: if `TRUE`, argument specification is checked.

Details

Predictors in a single-level regression can only be centered at the grand mean (CGM) by specifying `type = "CGM"` (default) in conjunction with `group = NULL` (default).

Level-1 (L1) predictors in a multilevel regression can be centered at the grand mean (CGM) by specifying `type = "CGM"` (default) in conjunction with `group = NULL` (default) or within cluster (CWC) by specifying `type = "CWC"` in conjunction with specifying a group membership variable using the `group` argument.

Level-2 (L2) predictors in a multilevel regression can only be centered at the grand mean (CGM) by specifying `type = "CGM"` (default) in conjunction with specifying a group membership variable using the `group` argument.

Note that predictors can be centered on any meaningful value using the argument `value`.

Value

Returns a numeric vector with the same length as `x` containing centered values.

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Enders, C. K. (2013). Centering predictors and contextual effects. In M. A. Scott, J. S. Simonoff, & B. D. Marx (Eds.), *The Sage handbook of multilevel modeling* (pp. 89-109). Sage. https://dx.doi.org/10.4135/9781446247600

Enders, C. K., & Tofighi, D. (2007). Centering predictor variables in cross-sectional multilevel models: A new look at an old issue. *Psychological Methods, 12*, 121-138. https://doi.org/10.1037/1082-989X.12.2.121

See Also

dummy.c, group.scores, rec, reverse.item, rwg.lindell, scores.

Examples

```
#--------------------------------------
# Predictors in a single-level regression
dat.sl <- data.frame(x = c(4, 2, 5, 6, 3, 4, 1, 3, 4),
                     y = c(5, 3, 6, 3, 4, 5, 2, 6, 5), stringsAsFactors = FALSE)

# Center predictor at the sample mean
center(dat.sl$x)

# Center predictor at the value 3
center(dat.sl$x, value = 3)

#--------------------------------------
# Predictors in a multilevel regression
dat.ml <- data.frame(id = c(1, 2, 3, 4, 5, 6, 7, 8, 9),
                     group = c(1, 1, 1, 2, 2, 2, 3, 3, 3),
                     x.l1 = c(4, 2, 5, 6, 3, 4, 1, 3, 4),
                     x.l2 = c(4, 4, 4, 1, 1, 1, 3, 3, 3),
                     y = c(5, 3, 6, 3, 4, 5, 2, 6, 5))

# Center level-1 predictor at the grand mean (CGM)
center(dat.ml$x.l1)

# Center level-1 predictor within cluster (CWC)
center(dat.ml$x.l1, type = "CWC", group = dat.ml$group)

# Center level-2 predictor at the grand mean (CGM)
center(dat.ml$x.l2, type = "CGM", group = dat.ml$group)
```

---

ci.mean *Confidence Interval for the Arithmetic Mean*

Description

This function computes a confidence interval for the arithmetic mean with known or unknown population standard deviation or population variance for one or more variables, optionally by a grouping and/or split variable.

Usage

```
ci.mean(x, sigma = NULL, sigma2 = NULL,
        alternative = c("two.sided", "less", "greater"), conf.level = 0.95,
        group = NULL, split = NULL, sort.var = FALSE, na.omit = FALSE,
        digits = 2, as.na = NULL, check = TRUE, output = TRUE)
```

Arguments

x	a numeric vector, matrix or data frame with numeric variables, i.e., factors and character variables are excluded from x before conducting the analysis.
sigma	a numeric vector indicating the population standard deviation when computing confidence intervals for the arithmetic mean with known standard deviation Note that either argument sigma or argument sigma2 is specified and it is only possible to specify one value for the argument sigma even though multiple variables are specified in x.
sigma2	a numeric vector indicating the population variance when computing confidence intervals for the arithmetic mean with known variance. Note that either argument sigma or argument sigma2 is specified and it is only possible to specify one value for the argument sigma2 even though multiple variables are specified in x.
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".
conf.level	a numeric value between 0 and 1 indicating the confidence level of the interval.
group	a numeric vector, character vector or factor as grouping variable. Note that a grouping variable can only be used when computing confidence intervals with unknown population standard deviation and population variance.
split	a numeric vector, character vector or factor as split variable. Note that a split variable can ?nly be used when computing confidence intervals with unknown population standard deviation and population variance.
sort.var	logical: if TRUE, output table is sorted by variables when specifying group.
na.omit	logical: if TRUE, incomplete cases are removed before conducting the analysis (i.e., listwise deletion) when specifying more than one outcome variable.
digits	an integer value indicating the number of decimal places to be used.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis. Note that as.na() function is only applied to x, but not to group or split.
check	logical: if TRUE, argument specification is checked.
output	logical: if TRUE, output is shown on the console.

Value

Returns an object of class misty.object, which is a list with following entries: function call (call), type of analysis type, list with the input specified in x, group, and split (data), specification of function arguments (args), and result table (result).

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

See Also

ci.mean.diff, ci.median, ci.prop, ci.var, ci.sd, descript

Examples

```
dat <- data.frame(group1 = c(1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2),
                  group2 = c(1, 1, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2),
                  x1 = c(3, 1, 4, 2, 5, 3, 2, 4, NA, 4, 5, 3),
                  x2 = c(4, NA, 3, 6, 3, 7, 2, 7, 5, 1, 3, 6),
                  x3 = c(7, 8, 5, 6, 4, NA, 8, NA, 6, 5, 8, 6))

# Two-Sided 95% Confidence Interval for x1
ci.mean(dat$x1)

# Two-Sided 95% Confidence Interval with known standard deviation for x1
ci.mean(dat$x1, sigma = 1.2)

# Two-Sided 95% Confidence Interval with known variance for x1
ci.mean(dat$x1, sigma2 = 2.5)

# One-Sided 95% Confidence Interval for x1
ci.mean(dat$x1, alternative = "less")

# Two-Sided 99% Confidence Interval
ci.mean(dat$x1, conf.level = 0.99)

# Two-Sided 95% Confidence Interval, print results with 3 digits
ci.mean(dat$x1, digits = 3)

# Two-Sided 95% Confidence Interval for x1, convert value 4 to NA
ci.mean(dat$x1, as.na = 4)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# listwise deletion for missing data
ci.mean(dat[, c("x1", "x2", "x3")], na.omit = TRUE)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# analysis by group1 separately
ci.mean(dat[, c("x1", "x2", "x3")], group = dat$group1)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# analysis by group1 separately, sort by variables
ci.mean(dat[, c("x1", "x2", "x3")], group = dat$group1, sort.var = TRUE)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# split analysis by group1
ci.mean(dat[, c("x1", "x2", "x3")], split = dat$group1)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# analysis by group1 separately, split analysis by group2
ci.mean(dat[, c("x1", "x2", "x3")], group = dat$group1, split = dat$group2)
```

ci.mean.diff *Confidence Interval for the Difference in Arithmetic Means*

Description

This function computes a confidence interval for the difference in arithmetic means from independent and paired samples with known or unknown population standard deviation or population variance for one or more variables, optionally by a grouping and/or split variable.

Usage

```
ci.mean.diff(x, ...)

## Default S3 method:
ci.mean.diff(x, y, sigma = NULL, sigma2 = NULL,
             var.equal = FALSE, paired = FALSE,
             alternative = c("two.sided", "less", "greater"),
             conf.level = 0.95, group = NULL, split = NULL,
             sort.var = FALSE, digits = 2, as.na = NULL,
             check = TRUE, output = TRUE, ...)

## S3 method for class 'formula'
ci.mean.diff(formula, data, sigma = NULL, sigma2 = NULL,
             var.equal = FALSE,
             alternative = c("two.sided", "less", "greater"),
             conf.level = 0.95, group = NULL, split = NULL,
             sort.var = FALSE, na.omit = FALSE, digits = 2,
             as.na = NULL, check = TRUE, output = TRUE, ...)
```

Arguments

`x`	a numeric vector of data values.
`y`	a numeric vector of data values.
`sigma`	a numeric vector indicating the population standard deviation(s) when computing confidence intervals for the difference in arithmetic means with known standard deviation(s). In case of independent samples, equal standard deviation is assumed when specifying one value for the argument `sigma`; when specifying two values for the argument `sigma`, unequal variance is assumed Note that either argument `sigma` or argument `sigma2` is specified and it is only possible to specify one value (i.e., equal variance assumption) or two values (i.e., unequal variance assumption) for the argument `sigma` even though multiple variables are specified in `x`.
`sigma2`	a numeric vector indicating the population variance(s) when computing confidence intervals for the difference in arithmetic means with known variance(s). In case of independent samples, equal variance is assumed when specifying one value for the argument `sigma2`; when specifying two values for the argument

sigma, unequal variance is assumed. Note that either argument sigma or argument sigma2 is specified and it is only possible to specify one value (i.e., equal variance assumption) or two values (i.e., unequal variance assumption) for the argument sigma even though multiple variables are specified in x.

var.equal logical: if TRUE, the population variance in the independent samples are assumed to be equal.

paired logical: if TRUE, confidence interval for the difference of arithmetic means in paired samples is computed.

alternative a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".

conf.level a numeric value between 0 and 1 indicating the confidence level of the interval.

group a numeric vector, character vector or factor as grouping variable. Note that a grouping variable can only be used when computing confidence intervals with unknown population standard deviation and population variance.

split a numeric vector, character vector or factor as split variable. Note that a split variable can only be used when computing confidence intervals with unknown population standard deviation and population variance.

sort.var logical: if TRUE, output table is sorted by variables when specifying group.

digits an integer value indicating the number of decimal places to be used.

as.na a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis. Note that as.na() function is only applied to x, but not to group or split.

check logical: if TRUE, argument specification is checked.

output logical: if TRUE, output is shown on the console.

formula in case of a between-subject design (i.e., paired = FALSE), a formula of the form y ~ group for one outcome variable or cbind(y1,y2,y3) ~ group for more than one outcome variable where y is a numeric variable giving the data values and group a numeric variable, character variable or factor with two values or factor levels given the corresponding groups; in case of a within-subject design (i.e., paired = TRUE), a formula of the form post ~ pre where post and pre are numeric variables. Note that analysis for more than one outcome variable is not permitted in within-subject design.

data a matrix or data frame containing the variables in the formula formula.

na.omit logical: if TRUE, incomplete cases are removed before conducting the analysis (i.e., listwise deletion) when specifying more than one outcome variable.

... further arguments to be passed to or from methods.

Value

Returns an object of class misty.object, which is a list with following entries: function call (call), type of analysis type, list with the input specified in x, group, and split (data), specification of function arguments (args), and result table (result).

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Fagerland, M. W., Lydersen S., & Laake, P. (2011). Recommended confidence intervals for two independent binomial proportions. *Statistical Methods in Medical Research, 24*, 224-254.

Newcombe, R. G. (1998a). Interval estimation for the difference between independent proportions: Comparison of eleven methods. *Statistics in Medicine, 17*, 873-890.

Newcombe, R. G. (1998b). Improved confidence intervals for the difference between binomial proportions based on paired data. *Statistics in Medicine, 17*, 2635-2650.

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

See Also

ci.mean, ci.median, ci.prop, ci.var, ci.sd, descript

Examples

```
dat.bs <- data.frame(group1 = c(1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2,
                                1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2),
                     group2 = c(1, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2,
                                1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2),
                     group3 = c(1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
                                1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2),
                     x1 = c(3, 1, 4, 2, 5, 3, 2, 3, 6, 4, 3, NA, 5, 3,
                            3, 2, 6, 3, 1, 4, 3, 5, 6, 7, 4, 3, 6, 4),
                     x2 = c(4, NA, 3, 6, 3, 7, 2, 7, 3, 3, 3, 1, 3, 6,
                            3, 5, 2, 6, 8, 3, 4, 5, 2, 1, 3, 1, 2, NA),
                     x3 = c(7, 8, 5, 6, 4, 2, 8, 3, 6, 1, 2, 5, 8, 6,
                            2, 5, 3, 1, 6, 4, 5, 5, 3, 6, 3, 2, 2, 4))

#--------------------------------------
# Between-Subject Design

# Two-Sided 95% Confidence Interval for y1 by group1
# unknown population variances, unequal variance assumption
ci.mean.diff(x1 ~ group1, data = dat.bs)

# Two-Sided 95% Confidence Interval for y1 by group1
# unknown population variances, equal variance assumption
ci.mean.diff(x1 ~ group1, data = dat.bs, var.equal = TRUE)

# Two-Sided 95% Confidence Interval with known standard deviations for x1 by group1
# known population standard deviations, equal standard deviation asssumption
ci.mean.diff(x1 ~ group1, data = dat.bs, sigma = 1.2)

# Two-Sided 95% Confidence Interval with known standard deviations for x1 by group1
# known population standard deviations, unequal standard deviation asssumption
ci.mean.diff(x1 ~ group1, data = dat.bs, sigma = c(1.5, 1.2))

# Two-Sided 95% Confidence Interval with known variance for x1 by group1
# known population variances, equal variance asssumption
```

```
ci.mean.diff(x1 ~ group1, data = dat.bs, sigma2 = 1.44)

# Two-Sided 95% Confidence Interval with known variance for x1 by group1
# known population variances, unequal variance asssumption
ci.mean.diff(x1 ~ group1, data = dat.bs, sigma2 = c(2.25, 1.44))

# One-Sided 95% Confidence Interval for y1 by group1
# unknown population variances, unequal variance assumption
ci.mean.diff(x1 ~ group1, data = dat.bs, alternative = "less")

# Two-Sided 95% Confidence Interval for y1 by group1
# unknown population variances, unequal variance assumption
ci.mean.diff(x1 ~ group1, data = dat.bs, conf.level = 0.99)

# Two-Sided 95% Confidence Interval for y1 by group1
# unknown population variances, unequal variance assumption
# print results with 3 digits
ci.mean.diff(x1 ~ group1, data = dat.bs, digits = 3)

# Two-Sided 95% Confidence Interval for y1 by group1
# unknown population variances, unequal variance assumption
# convert value 4 to NA
ci.mean.diff(x1 ~ group1, data = dat.bs, as.na = 4)

# Two-Sided 95% Confidence Interval for y1, y2, and y3 by group1
# unknown population variances, unequal variance assumption
ci.mean.diff(cbind(x1, x2, x3) ~ group1, data = dat.bs)

# Two-Sided 95% Confidence Interval for y1, y2, and y3 by group1
# unknown population variances, unequal variance assumption,
# listwise deletion for missing data
ci.mean.diff(cbind(x1, x2, x3) ~ group1, data = dat.bs, na.omit = TRUE)

# Two-Sided 95% Confidence Interval for y1, y2, and y3 by group1
# unknown population variances, unequal variance assumption,
# analysis by group2 separately
ci.mean.diff(cbind(x1, x2, x3) ~ group1, data = dat.bs, group = dat.bs$group2)

# Two-Sided 95% Confidence Interval for y1, y2, and y3 by group1
# unknown population variances, unequal variance assumption,
# analysis by group2 separately, sort by variables
ci.mean.diff(cbind(x1, x2, x3) ~ group1, data = dat.bs, group = dat.bs$group2,
             sort.var = TRUE)

# Two-Sided 95% Confidence Interval for y1, y2, and y3 by group1
# unknown population variances, unequal variance assumption,
# split analysis by group2
ci.mean.diff(cbind(x1, x2, x3) ~ group1, data = dat.bs, split = dat.bs$group2)

# Two-Sided 95% Confidence Interval for y1, y2, and y3 by group1
# unknown population variances, unequal variance assumption,
# analysis by group2 separately, split analysis by group3
ci.mean.diff(cbind(x1, x2, x3) ~ group1, data = dat.bs,
```

```
            group = dat.bs$group2, split = dat.bs$group3)

#-----------------

group1 <- c(3, 1, 4, 2, 5, 3, 6, 7)
group2 <- c(5, 2, 4, 3, 1)

# Two-Sided 95% Confidence Interval for the mean difference between group1 amd group2
# unknown population variances, unequal variance assumption
ci.mean.diff(group1, group2)

# Two-Sided 95% Confidence Interval for the mean difference between group1 amd group2
# unknown population variances, equal variance assumption
ci.mean.diff(group1, group2, var.equal = TRUE)

#--------------------------------------
# Within-Subject Design
dat.ws <- data.frame(pre = c(1, 3, 2, 5, 7),
                     post = c(2, 2, 1, 6, 8))

# Two-Sided 95% Confidence Interval for the mean difference in x1 and x2
# unknown poulation variance of difference scores
ci.mean.diff(dat.ws$pre, dat.ws$post, paired = TRUE)

# Two-Sided 95% Confidence Interval for the mean difference in x1 and x2
# known population standard deviation of difference scores
ci.mean.diff(dat.ws$pre, dat.ws$post, sigma = 2, paired = TRUE)

# Two-Sided 95% Confidence Interval for the mean difference in x1 and x2
# known population variance of difference scores
ci.mean.diff(dat.ws$pre, dat.ws$post, sigma2 = 4, paired = TRUE)

# One-Sided 95% Confidence Interval for the mean difference in x1 and x2
# unknown population variances, unequal variance assumption
ci.mean.diff(dat.ws$pre, dat.ws$post, alternative = "less", paired = TRUE)

# Two-Sided 95% Confidence Interval for the mean difference in x1 and x2
# unknown population variances, unequal variance assumption
ci.mean.diff(dat.ws$pre, dat.ws$post, conf.level = 0.99, paired = TRUE)

# Two-Sided 95% Confidence Interval for for the mean difference in x1 and x2
# unknown population variances, unequal variance assumption
# print results with 3 digits
ci.mean.diff(dat.ws$pre, dat.ws$post, paired = TRUE, digits = 3)

# Two-Sided 95% Confidence Interval for y1 by group1
# unknown population variances, unequal variance assumption
# convert value 1 to NA
ci.mean.diff(dat.ws$pre, dat.ws$post, as.na = 1, paired = TRUE)
```

`ci.median` *Confidence Interval for the Median*

Description

This function computes a confidence interval for the median for one or more variables, optionally by a grouping and/or split variable.

Usage

```
ci.median(x, alternative = c("two.sided", "less", "greater"),
          conf.level = 0.95, group = NULL, split = NULL,
          sort.var = FALSE, na.omit = FALSE, digits = 2,
          as.na = NULL, check = TRUE, output = TRUE)
```

Arguments

`x`	a numeric vector, matrix or data frame with numeric variables, i.e., factors and character variables are excluded from `x` before conducting the analysis.
`alternative`	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`.
`conf.level`	a numeric value between 0 and 1 indicating the confidence level of the interval.
`group`	a numeric vector, character vector or factor as grouping variable.
`split`	a numeric vector, character vector or factor as split variable.
`sort.var`	logical: if `TRUE`, output table is sorted by variables when specifying `group`.
`na.omit`	logical: if `TRUE`, incomplete cases are removed before conducting the analysis (i.e., listwise deletion) when specifying more than one outcome variable.
`digits`	an integer value indicating the number of decimal places to be used.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis. Note that `as.na()` function is only applied to `x`, but not to `group` or `split`.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown on the console.

Details

The confidence interval for the median is computed by using the Binomial distribution to determine which values in the sample are the lower and the upper confidence limits. Note that at least six valid observations are needed to compute the confidence interval for the median.

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, list with the input specified in `x`, `group`, and `split` (`data`), specification of function arguments (`args`), and result table (`result`).

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

See Also

ci.mean, ci.mean.diff, ci.prop, ci.prop.diff, ci.var, ci.sd, descript

Examples

```
dat <- data.frame(group1 = c(1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2,
                             1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2),
                  group2 = c(1, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2,
                             1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2),
                  x1 = c(3, 1, 4, 2, 5, 3, 2, 3, 6, 4, 3, NA, 5, 3,
                         3, 2, 6, 3, 1, 4, 3, 5, 6, 7, 4, 3, 5, 4),
                  x2 = c(4, NA, 3, 6, 3, 7, 2, 7, 3, 3, 3, 1, 3, 6,
                         3, 5, 2, 6, 8, 3, 4, 5, 2, 1, 3, 1, 2, NA),
                  x3 = c(7, 8, 5, 6, 4, 2, 8, 3, 6, 1, 2, 5, 8, 6,
                         2, 5, 3, 1, 6, 4, 5, 5, 3, 6, 3, 2, 2, 4))

# Two-Sided 95% Confidence Interval for x1
ci.median(dat$x1)

# One-Sided 95% Confidence Interval for x1
ci.median(dat$x1, alternative = "less")

# Two-Sided 99% Confidence Interval
ci.median(dat$x1, conf.level = 0.99)

# Two-Sided 95% Confidence Interval, print results with 3 digits
ci.median(dat$x1, digits = 3)

# Two-Sided 95% Confidence Interval for x1, convert value 4 to NA
ci.median(dat$x1, as.na = 4)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# listwise deletion for missing data
ci.median(dat[, c("x1", "x2", "x3")], na.omit = TRUE)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# analysis by group1 separately
ci.median(dat[, c("x1", "x2", "x3")], group = dat$group1)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# analysis by group1 separately, sort by variables
ci.median(dat[, c("x1", "x2", "x3")], group = dat$group1, sort.var = TRUE)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# split analysis by group1
ci.median(dat[, c("x1", "x2", "x3")], split = dat$group1)
```

```
# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# analysis by group1 separately, split analysis by group2
ci.median(dat[, c("x1", "x2", "x3")], group = dat$group1, split = dat$group2)
```

---

ci.prop *Confidence Interval for Proportions*

Description

This function computes a confidence interval for proportions for one or more variables, optionally by a grouping and/or split variable.

Usage

```
ci.prop(x, method = c("wald", "wilson"),
        alternative = c("two.sided", "less", "greater"),
        conf.level = 0.95, group = NULL, split = NULL,
        sort.var = FALSE, na.omit = FALSE, digits = 3,
        as.na = NULL, check = TRUE, output = TRUE)
```

Arguments

`x`	a numeric vector, matrix or data frame with numeric variables with 0 and 1 values, i.e., factors and character variables are excluded from `x` before conducting the analysis.
`method`	a character string specifying the method for computing the confidence interval, must be one of `"wald"`, or `"wilson"` (default).
`alternative`	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`.
`conf.level`	a numeric value between 0 and 1 indicating the confidence level of the interval.
`group`	a numeric vector, character vector or factor as grouping variable.
`split`	a numeric vector, character vector or factor as split variable.
`sort.var`	logical: if `TRUE`, output table is sorted by variables when specifying `group`.
`na.omit`	logical: if `TRUE`, incomplete cases are removed before conducting the analysis (i.e., listwise deletion) when specifying more than one outcome variable.
`digits`	an integer value indicating the number of decimal places to be used.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis. Note that `as.na()` function is only applied to `x`, but not to `group` or `split`.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown on the console.

Details

The Wald confidence interval which is based on the normal approximation to the binomial distribution are computed by specifying `method = "wald"`, while the Wilson (1927) confidence interval (aka Wilson score interval) is requested by specifying `method = "wilson"`. By default, Wilson confidence interval is computed which have been shown to be reliable in small samples of n = 40 or less, and larger samples of n > 40 (Brown, Cai & DasGupta, 2001), while the Wald confidence intervals is inadequate in small samples and when p is near 0 or 1 (Agresti & Coull, 1998).

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, list with the input specified in `x`, `group`, and `split` (`data`), specification of function arguments (`args`), and result table (`result`).

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Agresti, A. & Coull, B.A. (1998). Approximate is better than "exact" for interval estimation of binomial proportions. *American Statistician, 52*, 119-126.

Brown, L. D., Cai, T. T., & DasGupta, A., (2001). Interval estimation for a binomial proportion. *Statistical Science, 16*, 101-133.

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

Wilson, E. B. (1927). Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association, 22*, 209-212.

See Also

`ci.mean`, `ci.mean.diff`, `ci.median`, `ci.prop.diff`, `ci.var`, `ci.sd`, `descript`

Examples

```
dat <- data.frame(group1 = c(1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2),
                  group2 = c(1, 1, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2),
                  x1 = c(0, 1, 0, 0, 1, 1, 0, 1, NA, 0, 1, 0),
                  x2 = c(0, NA, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1),
                  x3 = c(1, 1, 1, 0, 1, NA, 1, NA, 0, 0, 0, 1))

# Two-Sided 95% Confidence Interval for x1
ci.prop(dat$x1)

# Two-Sided 95% Confidence Interval for x1 using Wald method
ci.prop(dat$x1, method = "wald")

# One-Sided 95% Confidence Interval for x1
ci.prop(dat$x1, alternative = "less")
```

```
# Two-Sided 99% Confidence Interval
ci.prop(dat$x1, conf.level = 0.99)

# Two-Sided 95% Confidence Interval, print results with 4 digits
ci.prop(dat$x1, digits = 4)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# listwise deletion for missing data
ci.prop(dat[, c("x1", "x2", "x3")], na.omit = TRUE)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# analysis by group1 separately
ci.prop(dat[, c("x1", "x2", "x3")], group = dat$group1)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# analysis by group1 separately, sort by variables
ci.prop(dat[, c("x1", "x2", "x3")], group = dat$group1, sort.var = TRUE)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# split analysis by group1
ci.prop(dat[, c("x1", "x2", "x3")], split = dat$group1)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# analysis by group1 separately, split analysis by group2
 ci.prop(dat[, c("x1", "x2", "x3")],
         group = dat$group1, split = dat$group2)
```

`ci.prop.diff` *Confidence Interval for the Difference in Proportions*

Description

This function computes a confidence interval for the difference in proportions from independent and paired samples for one or more variables, optionally by a grouping and/or split variable.

Usage

```
ci.prop.diff(x, ...)


## Default S3 method:
## Default S3 method:
ci.prop.diff(x, y, method = c("wald", "newcombe"), paired = FALSE,
                               alternative = c("two.sided", "less", "greater"),
                               conf.level = 0.95, group = NULL, split = NULL,
                               sort.var = FALSE, digits = 2, as.na = NULL,
                               check = TRUE, output = TRUE, ...)
```

```
## S3 method for class 'formula'
## S3 method for class 'formula'
ci.prop.diff(formula, data, method = c("wald", "newcombe"),
                               alternative = c("two.sided", "less", "greater"),
                               conf.level = 0.95, group = NULL, split = NULL,
                               sort.var = FALSE, na.omit = FALSE, digits = 2,
                               as.na = NULL, check = TRUE, output = TRUE, ...)
```

Arguments

`x`	a numeric vector with 0 and 1 values.
`...`	further arguments to be passed to or from methods.
`y`	a numeric vector with 0 and 1 values.
`method`	a character string specifying the method for computing the confidence interval, must be one of `"wald"`, or `"newcombe"` (default).
`paired`	logical: if `TRUE`, confidence interval for the difference of proportions in paired samples is computed.
`alternative`	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`.
`conf.level`	a numeric value between 0 and 1 indicating the confidence level of the interval.
`group`	a numeric vector, character vector or factor as grouping variable. Note that a grouping variable can only be used when computing confidence intervals with unknown population standard deviation and population variance.
`split`	a numeric vector, character vector or factor as split variable. Note that a split variable can only be used when computing confidence intervals with unknown population standard deviation and population variance.
`sort.var`	logical: if `TRUE`, output table is sorted by variables when specifying `group`.
`digits`	an integer value indicating the number of decimal places to be used.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis. Note that `as.na()` function is only applied to `x`, but not to `group` or `split`.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown on the console.
`formula`	a formula of the form `y ~ group` for one outcome variable or `cbind(y1,y2,y3) ~ group` for more than one outcome variable where `y` is a numeric variable with 0 and 1 values and `group` a numeric variable, character variable or factor with two values of factor levels given the corresponding group.
`data`	a matrix or data frame containing the variables in the formula `formula`.
`na.omit`	logical: if `TRUE`, incomplete cases are removed before conducting the analysis (i.e., listwise deletion) when specifying more than one outcome variable.

Details

The Wald confidence interval which is based on the normal approximation to the binomial distribution are computed by specifying `method = "wald"`, while the Newcombe Hybrid Score interval (Newcombe, 1998a; Newcombe, 1998b) is requested by specifying `method = "newcombe"`. By default, Newcombe Hybrid Score interval is computed which have been shown to be reliable in small samples (less than n = 30 in each sample) as well as moderate to larger samples(n > 30 in each sample) and with proportions close to 0 or 1, while the Wald confidence intervals does not perform well unless the sample size is large (Fagerland, Lydersen & Laake, 2011).

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, list with the input specified in `x`, `group`, and `split` (`data`), specification of function arguments (`args`), and result table (`result`).

Author(s)

Takuya Yanagida `<takuya.yanagida@univie.ac.at>`

References

Fagerland, M. W., Lydersen S., & Laake, P. (2011) Recommended confidence intervals for two independent binomial proportions. *Statistical Methods in Medical Research, 24*, 224-254.

Newcombe, R. G. (1998a). Interval estimation for the difference between independent proportions: Comparison of eleven methods. *Statistics in Medicine, 17*, 873-890.

Newcombe, R. G. (1998b). Improved confidence intervals for the difference between binomial proportions based on paired data. *Statistics in Medicine, 17*, 2635-2650.

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

See Also

`ci.prop`, `ci.mean`, `ci.mean.diff`, `ci.median`, `ci.var`, `ci.sd`, `descript`

Examples

```
dat.bs <- data.frame(group1 = c(1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2,
                                1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2),
                     group2 = c(1, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2,
                                1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2),
                     group3 = c(1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
                                1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2),
                     x1 = c(0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, NA, 0, 0,
                            1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0),
                     x2 = c(0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1,
                            1, 0, 1, 0, 1, 1, 1, NA, 1, 0, 0, 1, 1, 1),
                     x3 = c(1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0,
                            1, 0, 1, 1, 0, 1, 1, 1, 0, 1, NA, 1, 0, 1))
```

```
#--------------------------------------
# Between-Subject Design

# Two-Sided 95% Confidence Interval for x1 by group1
ci.prop.diff(x1 ~ group1, data = dat.bs)

# Two-Sided 95% Confidence Interval for x1 by group1
# Wald confidence interval
ci.prop.diff(x1 ~ group1, data = dat.bs, method = "wald")

# One-Sided 95% Confidence Interval for x1 by group1
# Newcombes Hybrid Score interval
ci.prop.diff(x1 ~ group1, data = dat.bs, alternative = "less")

# Two-Sided 95% Confidence Interval for x1 by group1
# Newcombes Hybrid Score interval
ci.prop.diff(x1 ~ group1, data = dat.bs, conf.level = 0.99)

# Two-Sided 95% Confidence Interval for y1 by group1
# # Newcombes Hybrid Score interval, print results with 3 digits
ci.prop.diff(x1 ~ group1, data = dat.bs, digits = 3)

# Two-Sided 95% Confidence Interval for y1 by group1
# # Newcombes Hybrid Score interval, convert value 0 to NA
ci.prop.diff(x1 ~ group1, data = dat.bs, as.na = 0)

# Two-Sided 95% Confidence Interval for y1, y2, and y3 by group1
# Newcombes Hybrid Score interval
ci.prop.diff(cbind(x1, x2, x3) ~ group1, data = dat.bs)

# Two-Sided 95% Confidence Interval for y1, y2, and y3 by group1
# # Newcombes Hybrid Score interval, listwise deletion for missing data
ci.prop.diff(cbind(x1, x2, x3) ~ group1, data = dat.bs, na.omit = TRUE)

# Two-Sided 95% Confidence Interval for y1, y2, and y3 by group1
# Newcombes Hybrid Score interval, analysis by group2 separately
ci.prop.diff(cbind(x1, x2, x3) ~ group1, data = dat.bs, group = dat.bs$group2)

# Two-Sided 95% Confidence Interval for y1, y2, and y3 by group1
# Newcombes Hybrid Score interval, analysis by group2 separately, sort by variables
ci.prop.diff(cbind(x1, x2, x3) ~ group1, data = dat.bs, group = dat.bs$group2,
             sort.var = TRUE)

# Two-Sided 95% Confidence Interval for y1, y2, and y3 by group1
# split analysis by group2
ci.prop.diff(cbind(x1, x2, x3) ~ group1, data = dat.bs, split = dat.bs$group2)

# Two-Sided 95% Confidence Interval for y1, y2, and y3 by group1
# Newcombes Hybrid Score interval, analysis by group2 separately, split analysis by group3
ci.prop.diff(cbind(x1, x2, x3) ~ group1, data = dat.bs,
             group = dat.bs$group2, split = dat.bs$group3)

#-----------------
```

```
group1 <- c(0, 1, 1, 0, 0, 1, 0, 1)
group2 <- c(1, 1, 1, 0, 0)

# Two-Sided 95% Confidence Interval for the mean difference between group1 amd group2
# Newcombes Hybrid Score interval
ci.prop.diff(group1, group2)

#--------------------------------------
# Within-Subject Design
dat.ws <- data.frame(pre = c(0, 1, 1, 0, 1),
                     post = c(1, 1, 0, 1, 1), stringsAsFactors = FALSE)

# Two-Sided 95% Confidence Interval for the mean difference in x1 and x2
# Newcombes Hybrid Score interval
ci.prop.diff(dat.ws$pre, dat.ws$post, paired = TRUE)

# Two-Sided 95% Confidence Interval for the mean difference in x1 and x2
# Wald confidence interval
ci.prop.diff(dat.ws$pre, dat.ws$post, method = "wald", paired = TRUE)

# One-Sided 95% Confidence Interval for the mean difference in x1 and x2
# Newcombes Hybrid Score interval
ci.prop.diff(dat.ws$pre, dat.ws$post, alternative = "less", paired = TRUE)

# Two-Sided 95% Confidence Interval for the mean difference in x1 and x2
# Newcombes Hybrid Score interval
ci.prop.diff(dat.ws$pre, dat.ws$post, conf.level = 0.99, paired = TRUE)

# Two-Sided 95% Confidence Interval for for the mean difference in x1 and x2
# Newcombes Hybrid Score interval, print results with 3 digits
ci.prop.diff(dat.ws$pre, dat.ws$post, paired = TRUE, digits = 3)
```

`ci.sd` *Confidence Interval for the Standard Deviation*

Description

This function computes a confidence interval for the standard deviation for one or more variables, optionally by a grouping and/or split variable.

Usage

```
ci.sd(x, method = c("chisq", "bonett"),
      alternative = c("two.sided", "less", "greater"),
      conf.level = 0.95, group = NULL, split = NULL,
      sort.var = FALSE, na.omit = FALSE, digits = 2,
      as.na = NULL, check = TRUE, output = TRUE)
```

Arguments

`x`	a numeric vector, matrix or data frame with numeric variables, i.e., factors and character variables are excluded from `x` before conducting the analysis.
`method`	a character string specifying the method for computing the confidence interval, must be one of `"chisq"`, or `"bonett"` (default).
`alternative`	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`.
`conf.level`	a numeric value between 0 and 1 indicating the confidence level of the interval.
`group`	a numeric vector, character vector or factor as grouping variable.
`split`	a numeric vector, character vector or factor as split variable.
`sort.var`	logical: if `TRUE`, output table is sorted by variables when specifying `group`.
`na.omit`	logical: if `TRUE`, incomplete cases are removed before conducting the analysis (i.e., listwise deletion) when specifying more than one outcome variable.
`digits`	an integer value indicating the number of decimal places to be used.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis. Note that `as.na()` function is only applied to `x`, but not to `group` or `split`.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown on the console.

Details

The confidence interval based on the chi-square distribution is computed by specifying `method = "chisq"`, while the Bonett (2006) confidence interval is requested by specifying `method = "bonett"`. By default, the Bonett confidence interval interval is computed which performs well under moderate departure from normality, while the confidence interval based on the chi-square distribution is highly sensitive to minor violations of the normality assumption and its performance does not improve with increasing sample size.

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, list with the input specified in `x`, `group`, and `split` (`data`), specification of function arguments (`args`), and result table (`result`).

Author(s)

Takuya Yanagida `<takuya.yanagida@univie.ac.at>`

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

Bonett, D. G. (2006). Approximate confidence interval for standard deviation of nonnormal distributions. *Computational Statistics and Data Analysis, 50*, 775-782. https://doi.org/10.1016/j.csda.2004.10.003

See Also

ci.mean, ci.mean.diff, ci.median, ci.prop, ci.prop.diff, ci.var, descript

Examples

```
dat <- data.frame(group1 = c(1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2,
                             1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2),
                  group2 = c(1, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2,
                             1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2),
                  x1 = c(3, 1, 4, 2, 5, 3, 2, 3, 6, 4, 3, NA, 5, 3,
                         3, 2, 6, 3, 1, 4, 3, 5, 6, 7, 4, 3, 5, 4),
                  x2 = c(4, NA, 3, 6, 3, 7, 2, 7, 3, 3, 3, 1, 3, 6,
                         3, 5, 2, 6, 8, 3, 4, 5, 2, 1, 3, 1, 2, NA),
                  x3 = c(7, 8, 5, 6, 4, 2, 8, 3, 6, 1, 2, 5, 8, 6,
                         2, 5, 3, 1, 6, 4, 5, 5, 3, 6, 3, 2, 2, 4))

# Two-Sided 95% Confidence Interval for x1
ci.sd(dat$x1)

# Two-Sided 95% Confidence Interval for x1 using chi square distribution
ci.sd(dat$x1, method = "chisq")

# One-Sided 95% Confidence Interval for x1
ci.sd(dat$x1, alternative = "less")

# Two-Sided 99% Confidence Interval
ci.sd(dat$x1, conf.level = 0.99)

# Two-Sided 95% Confidence Interval, print results with 3 digits
ci.sd(dat$x1, digits = 3)

# Two-Sided 95% Confidence Interval for x1, convert value 4 to NA
ci.sd(dat$x1, as.na = 4)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# listwise deletion for missing data
ci.sd(dat[, c("x1", "x2", "x3")], na.omit = TRUE)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# analysis by group1 separately
ci.sd(dat[, c("x1", "x2", "x3")], group = dat$group1)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# analysis by group1 separately, sort by variables
ci.sd(dat[, c("x1", "x2", "x3")], group = dat$group1, sort.var = TRUE)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# split analysis by group1
ci.sd(dat[, c("x1", "x2", "x3")], split = dat$group1)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# analysis by group1 separately, split analysis by group2
```

```
ci.sd(dat[, c("x1", "x2", "x3")],
      group = dat$group1, split = dat$group2)
```

---

ci.var *Confidence Interval for the Variance*

Description

This function computes a confidence interval for the variance for one or more variables, optionally by a grouping and/or split variable.

Usage

```
ci.var(x, method = c("chisq", "bonett"),
       alternative = c("two.sided", "less", "greater"),
       conf.level = 0.95, group = NULL, split = NULL,
       sort.var = FALSE, na.omit = FALSE, digits = 2,
       as.na = NULL, check = TRUE, output = TRUE)
```

Arguments

`x`	a numeric vector, matrix or data frame with numeric variables, i.e., factors and character variables are excluded from `x` before conducting the analysis.
`method`	a character string specifying the method for computing the confidence interval, must be one of `"chisq"`, or `"bonett"` (default).
`alternative`	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`.
`conf.level`	a numeric value between 0 and 1 indicating the confidence level of the interval.
`group`	a numeric vector, character vector or factor as grouping variable.
`split`	a numeric vector, character vector or factor as split variable.
`sort.var`	logical: if `TRUE`, output table is sorted by variables when specifying `group`.
`na.omit`	logical: if `TRUE`, incomplete cases are removed before conducting the analysis (i.e., listwise deletion) when specifying more than one outcome variable.
`digits`	an integer value indicating the number of decimal places to be used.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis. Note that `as.na()` function is only applied to `x`, but not to `group` or `split`.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown on the console.

Details

The confidence interval based on the chi-square distribution is computed by specifying `method = "chisq"`, while the Bonett (2006) confidence interval is requested by specifying `method = "bonett"`. By default, the Bonett confidence interval interval is computed which performs well under moderate departure from normality, while the confidence interval based on the chi-square distribution is highly sensitive to minor violations of the normality assumption and its performance does not improve with increasing sample size. Note that at least four valid observations are needed to compute the Bonett confidence interval.

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, list with the input specified in `x`, `group`, and `split` (`data`), specification of function arguments (`args`), and result table (`result`).

Author(s)

Takuya Yanagida `<takuya.yanagida@univie.ac.at>`

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

Bonett, D. G. (2006). Approximate confidence interval for standard deviation of nonnormal distributions. *Computational Statistics and Data Analysis, 50*, 775-782. https://doi.org/10.1016/j.csda.2004.10.003

See Also

`ci.mean`, `ci.mean.diff`, `ci.median`, `ci.prop`, `ci.prop.diff`, `ci.sd`, `descript`

Examples

```
dat <- data.frame(group1 = c(1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2,
                             1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2),
                  group2 = c(1, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2,
                             1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2),
                  x1 = c(3, 1, 4, 2, 5, 3, 2, 3, 6, 4, 3, NA, 5, 3,
                         3, 2, 6, 3, 1, 4, 3, 5, 6, 7, 4, 3, 5, 4),
                  x2 = c(4, NA, 3, 6, 3, 7, 2, 7, 3, 3, 3, 1, 3, 6,
                         3, 5, 2, 6, 8, 3, 4, 5, 2, 1, 3, 1, 2, NA),
                  x3 = c(7, 8, 5, 6, 4, 2, 8, 3, 6, 1, 2, 5, 8, 6,
                         2, 5, 3, 1, 6, 4, 5, 5, 3, 6, 3, 2, 2, 4))

# Two-Sided 95% Confidence Interval for x1
ci.var(dat$x1)

# Two-Sided 95% Confidence Interval for x1 using chi square distribution
ci.var(dat$x1, method = "chisq")

# One-Sided 95% Confidence Interval for x1
```

```
ci.var(dat$x1, alternative = "less")

# Two-Sided 99% Confidence Interval
ci.var(dat$x1, conf.level = 0.99)

# Two-Sided 95% Confidence Interval, print results with 3 digits
ci.var(dat$x1, digits = 3)

# Two-Sided 95% Confidence Interval for x1, convert value 4 to NA
ci.var(dat$x1, as.na = 4)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# listwise deletion for missing data
ci.var(dat[, c("x1", "x2", "x3")], na.omit = TRUE)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# analysis by group1 separately
ci.var(dat[, c("x1", "x2", "x3")], group = dat$group1)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# analysis by group1 separately, sort by variables
ci.var(dat[, c("x1", "x2", "x3")], group = dat$group1, sort.var = TRUE)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# split analysis by group1
ci.var(dat[, c("x1", "x2", "x3")], split = dat$group1)

# Two-Sided 95% Confidence Interval for x1, x2, and x3,
# analysis by group1 separately, split analysis by group2
ci.var(dat[, c("x1", "x2", "x3")],
       group = dat$group1, split = dat$group2)
```

`cohens.d` *Cohen's d for Between- and Within-Subject Design*

Description

This function computes Cohen's d for between- and within-subject designs with confidence intervals. By default, the function computes the standardized mean difference divided by the weighted pooled standard deviation without applying the correction factor for removing the small sample bias.

Usage

```
cohens.d(formula, data, paired = FALSE, weighted = TRUE,
         ref = NULL, correct = FALSE, digits = 2,
         conf.level = 0.95, as.na = NULL, check = TRUE,
         output = TRUE)
```

Arguments

- `formula` — in case of a between-subject design (i.e., `paired = FALSE`), a formula of the form `y ~ group` for one outcome variable or `cbind(y1,y2,y3) ~ group` for more than one outcome variable where `y` is a numeric variable giving the data values and `group` a numeric variable, character variable or factor with two values or factor levels giving the corresponding group; in case of a within-subject design (i.e., `paired = TRUE`), a formula of the form `post ~ pre` where `post` and `pre` are numeric variables. Note that analysis for more than one outcome variable is not permitted in within-subject design.
- `data` — a matrix or data frame containing the variables in the `formula`.
- `paired` — logical: if `TRUE`, Cohen's d for within-subject design is computed.
- `weighted` — logical: if `TRUE` (default), in case of a between-subject design the weighted pooled standard deviation is used; in case of a within-subject design the correlation between measures is controlled when computing the pooled standard deviation.
- `ref` — a numeric value or character string indicating the reference group in a between-subject design or a character string indicating the reference variable in a within-subject design. The standard deviation of the reference group or reference variable is used to standardized the mean difference. If the standard deviation of the control group is used (e.g. `group = "control"`), the effect size is usually called Glass' delta.
- `correct` — logical: if `TRUE`, correction factor to remove positive bias in small samples is used. Note that correction factor is only applied when `weighted = TRUE` and `ref = NULL`.
- `digits` — an integer value indicating the number of decimal places to be used for displaying results.
- `conf.level` — a numeric value between 0 and 1 indicating the confidence level of the interval.
- `as.na` — a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis. Note that `as.na()` function is only applied to `y` but not to `group` in a between-subject design, while `as.na()` function is applied to `pre` and `post` in a within-subject design.
- `check` — logical: if `TRUE`, argument specification is checked.
- `output` — logical: if `TRUE`, output is shown on the console.

Details

Cohen (1988, p.67) proposed to compute the standardized mean difference by dividing the mean difference by the unweighted pooled standard deviation (i.e., `weighted = FALSE`).

Glass et al. (1981, p. 29) suggested to use the standard deviation of the control group (e.g., `ref = "control"`) to compute the standardized mean difference since the standard deviation of the control group is unaffected by the treatment and will therefore more closely reflect the population standard deviation.

Hedges (1981, p. 110) recommended to weight each group's standard deviation by its sample size resulting in a weighted and pooled standard deviation (i.e., `weighted = TRUE`). According to Hedges and Olkin (1985, p. 81), the standardized mean difference based on the weighted and pooled standard deviation has a positive small sample bias, i.e., standardized mean difference is overestimates in small samples (i.e., sample size less than 20 or less than 10 in each group). However, a correction factor can be applied to remove the small sample bias (i.e., `correct = TRUE`). Note that a gamma function is used for computing the correction factor when n < 200, while a approximation method is used when n >= 200.

Note that the terminology is inconsistent because the standardized mean difference based on the weighted and pooled standard deviation is usually called Cohen's d, but sometimes called Hedges' g. Oftentimes, Cohen's d is called Hedges' d as soon as the correction factor is applied. It is recommended to avoid the term Hedges' g (Cumming & Calin-Jageman, 2017, p. 171), but to report which standard deviation was used to standardized the mean difference (e.g., unweighted/weighted pooled standard deviation, or the standard deviation of the control group) and whether a small sample correction factor was applied.

As for the terminology according to Lakens (2013), in between subject design (`paired = FALSE`) Cohen's d_s is computed when using `weighted = TRUE` and Hedges's g_s is computed when using `correct = TRUE` in addition. In within-subject designs (`paired = TRUE`), Cohen's d_rm is computed when using `weighted = TRUE`, while Cohen's d_av is computed when using `weighted = FALSE`, and corresponding Hedges' g_rm and Hedges' g_av are computed when using `correct = TRUE`.

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, matrix or data frame specified in `x` (`data`), specification of function arguments (`args`), and list with results (`result`).

Author(s)

Takuya Yanagida `<takuya.yanagida@univie.ac.at>`

References

Cohen, J. (1988). *Statistical power analysis for the behavioral sciences* (2nd ed.). Academic Press.

Cumming, G., & Calin-Jageman, R. (2017). *Introduction to the new statistics: Estimation, open science, & beyond*. Routledge.

Glass. G. V., McGaw, B., & Smith, M. L. (1981). *Meta-analysis in social research*. Sage Publication.

Goulet-Pelletier, J.-C., & Cousineau, D. (2018) A review of effect sizes and their confidence intervals, Part I: The Cohen's d family. *The Quantitative Methods for Psychology, 14*, 242-265. https://doi.org/10.20982/tqmp.14.4.p242

Hedges, L. V. (1981). Distribution theory for Glass's estimator of effect size and related estimators. *Journal of Educational Statistics, 6*(3), 106-128.

Hedges, L. V. & Olkin, I. (1985). *Statistical methods for meta-analysis*. Academic Press.

Lakens, D. (2013). Calculating and reporting effect sizes to facilitate cumulative science: A practical primer for t-tests and ANOVAs. *Frontiers in Psychology, 4*, 1-12. https://doi.org/10.3389/fpsyg.2013.00863

See Also

eta.sq, cont.coef, cramers.v,cor.matrix, na.auxiliary

Examples

```
#--------------------------------------
# Between-Subject Design
dat.bs <- data.frame(group = c("cont", "cont", "cont", "treat", "treat"),
                     y1 = c(1, 3, 2, 5, 7),
                     y2 = c(4, 3, 3, 6, 4),
                     y3 = c(7, 5, 7, 3, 2), stringsAsFactors = FALSE)

# Standardized mean difference divided by the weighted pooled
# standard deviation without small sample correction factor
cohens.d(y1 ~ group, data = dat.bs)

# Standardized mean difference divided by the unweighted pooled
# standard deviation without small sample correction factor
cohens.d(y1 ~ group, data = dat.bs, weighted = FALSE)

# Standardized mean difference divided by the weighted pooled
# standard deviation with small sample correction factor
cohens.d(y1 ~ group, data = dat.bs, correct = TRUE)

# Standardized mean difference divided by the standard deviation
# of the control group without small sample correction factor
cohens.d(y1 ~ group, data = dat.bs, ref = "cont")

# Cohens's d for for more than one outcome variable
cohens.d(cbind(y1, y2, y3) ~ group, data = dat.bs)

#--------------------------------------
# Within-Subject Design
dat.ws <- data.frame(pre = c(1, 3, 2, 5, 7),
                     post = c(2, 2, 1, 6, 8))

# Standardized mean difference divided by the pooled
# standard deviation while controlling for the correlation
# without small sample correction factor
cohens.d(post ~ pre, data = dat.ws, paired = TRUE)

# Standardized mean difference divided by the pooled
# standard deviation whithout controlling for the correlation
# without small sample correction factor
cohens.d(post ~ pre, data = dat.ws, paired = TRUE, weighted = FALSE)

# Standardized mean difference divided by the pooled
# standard deviation while controlling for the correlation
# with small sample correction factor
cohens.d(post ~ pre, data = dat.ws, paired = TRUE, correct = TRUE)

# Standardized mean difference divided by the standard deviation
```

```
# of the pretest without small sample correction factor
cohens.d(post ~ pre, data = dat.ws, paired = TRUE, ref = "pre")
```

---

collin.diag *Collinearity Diagnostics*

Description

This function computes tolerance, standard error inflation factor, variance inflation factor, eigenvalues, condition index, and variance proportions for linear, generalized linear, and mixed-effects models.

Usage

```
collin.diag(model, print = c("all", "vif", "eigen"), digits = 3,
            p.digits = 3, check = TRUE, output = TRUE)
```

Arguments

`model`	a fitted model of class `"lm"`, `"glm"`, `"lmerMod"`, `"lmerModLmerTest"`, `"glmerMod"`, `"lme"`, or `"glmmTMB"`.
`print`	a character vector indicating which results to show, i.e. `"all"`, for all results, `"vif"` for tolerance, std. error inflation factor, and variance inflation factor, or `eigen` for eigenvalue, condition index, and variance proportions.
`digits`	an integer value indicating the number of decimal places to be used for displaying results.
`p.digits`	an integer value indicating the number of decimal places to be used for displaying the *p*-value.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown on the console.

Details

Collinearity diagnostics can be conducted for objects returned from the `lm()` and `glm()` function, but also from objects returned from the `lmer()` and `glmer()` function from the **lme4** package, `lme()` function from the **nlme** package, and the `glmmTMB()` function from the **glmmTMB** package.

The generalized variance inflation factor (Fox & Monette, 1992) is computed for terms with more than 1 df resulting from factors with more than two levels. The generalized VIF (GVIF) is interpretable as the inflation in size of the confidence ellipse or ellipsoid for the coefficients of the term in comparison with what would be obtained for orthogonal data. GVIF is invariant to the coding of the terms in the model. In order to adjust for the dimension of the confidence ellipsoid, $\text{GVIF}^{\frac{1}{2df}}$ is computed. Note that the adjusted GVIF (aGVIF) is actually a generalized standard error inflation factor (GSIF). Thus, the aGIF needs to be squared before applying a common cutoff threshold for the VIF (e.g., VIF > 10). Note that the output of `collin.diag()` function reports either the variance inflation factor or the squared generalized variance inflation factor in the column `VIF`, while the standard error inflation factor or the adjusted generalized variance inflation factor is reported in the column `SIF`.

Note that the computation of the VIF and the GVIF is based on the `vif()` function in the **car** package by John Fox, Sanford Weisberg and Brad Price (2020), and the computation of eigenvalues, condition index, and variance proportions is based on the `ols_eigen_cindex()` function in the **olsrr** package by Aravind Hebbali (2020).

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, object specified in the `model` argument (`model`), specification of function arguments (`args`), list with results (`result`).

Author(s)

Takuya Yanagida `<takuya.yanagida@univie.ac.at>`

References

Fox, J., & Monette, G. (1992). Generalized collinearity diagnostics. *Journal of the Americaln Statistical Association, 87*, 178-183.

Fox, J., Weisberg, S., & Price, B. (2020). *car: Companion to Applied Regression*. R package version 3.0-8. https://cran.r-project.org/web/packages/car/

Hebbali, A. (2020). *olsrr: Tools for building OLS regression models*. R package version 0.5.3. https://cran.r-project.org/web/packages/olsrr/

Examples

```
dat <- data.frame(group = c(1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4),
                  x1 = c(3, 2, 4, 9, 5, 3, 6, 4, 5, 6, 3, 5),
                  x2 = c(1, 4, 3, 1, 2, 4, 3, 5, 1, 7, 8, 7),
                  x3 = c(7, 3, 4, 2, 5, 6, 4, 2, 3, 5, 2, 8),
                  x4 = c("a", "b", "a", "c", "c", "c", "a", "b", "b", "c", "a", "c"),
                  y1 = c(2, 7, 4, 4, 7, 8, 4, 2, 5, 1, 3, 8),
                  y2 = c(0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1),
                  stringsAsFactors = TRUE)

#----------------------------
# Linear model

# Estimate linear model with continuous predictors
mod.lm1 <- lm(y1 ~ x1 + x2 + x3, data = dat)

# Tolerance, std. error, and variance inflation factor
collin.diag(mod.lm1)

# Tolerance, std. error, and variance inflation factor
# Eigenvalue, Condition index, and variance proportions
collin.diag(mod.lm1, print = "all")
```

```
# Estimate model with continuous and categorical predictors
mod.lm2 <- lm(y1 ~ x1 + x2 + x3 + x4, data = dat)

# Tolerance, generalized std. error, and variance inflation factor
collin.diag(mod.lm2)

#----------------------------
# Generalized linear model

# Estimate logistic regression model with continuous predictors
mod.glm <- glm(y2 ~ x1 + x2 + x3, data = dat, family = "binomial")

# Tolerance, std. error, and variance inflation factor
collin.diag(mod.glm)

## Not run:
#----------------------------
# Linear mixed-effects model

# Estimate linear mixed-effets model with continuous predictors using lme4 package
mod.lmer <- lme4::lmer(y1 ~ x1 + x2 + x3 + (1|group), data = dat)

# Tolerance, std. error, and variance inflation factor
collin.diag(mod.lmer)

# Estimate linear mixed-effets model with continuous predictors using nlme package
mod.lme <- nlme::lme(y1 ~ x1 + x2 + x3, random = ~ 1 | group, data = dat)

# Tolerance, std. error, and variance inflation factor
collin.diag(mod.lme)

# Estimate linear mixed-effets model with continuous predictors using glmmTMB package
mod.glmmTMB1 <- glmmTMB::glmmTMB(y1 ~ x1 + x2 + x3 + (1|group), data = dat)

# Tolerance, std. error, and variance inflation factor
collin.diag(mod.glmmTMB1)

#----------------------------
# Generalized linear mixed-effects model

# Estimate mixed-effects logistic regression model with continuous predictors using lme4 package
mod.glmer <- lme4::glmer(y2 ~ x1 + x2 + x3 + (1|group), data = dat, family = "binomial")

# Tolerance, std. error, and variance inflation factor
collin.diag(mod.glmer)

# Estimate mixed-effects logistic regression model with continuous predictors using glmmTMB package
mod.glmmTMB2 <- glmmTMB::glmmTMB(y2 ~ x1 + x2 + x3 + (1|group), data = dat, family = "binomial")

# Tolerance, std. error, and variance inflation factor
collin.diag(mod.glmmTMB2)

## End(Not run)
```

cont.coef — *Pearson's Contingency Coefficient*

Description

This function computes the (adjusted) Pearson's contingency coefficient between two or more than two variables.

Usage

```
cont.coef(x, adjust = FALSE, tri = c("both", "lower", "upper"),
          digits = 2, as.na = NULL, check = TRUE, output = TRUE)
```

Arguments

`x`	a matrix or data frame with integer vectors, character vectors or factors..
`adjust`	logical: if `TRUE`, the adjusted contingency coefficient (i.e., Sakoda's adjusted Pearson's C) is computed.
`tri`	a character string indicating which triangular of the matrix to show on the console, i.e., `both` for upper and lower triangular, `lower` (default) for the lower triangular, and `upper` for the upper triangular.
`digits`	an integer value indicating the number of decimal places digits to be used for displaying contingency coefficients.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown on the console.

Details

The maximum contingency coefficient is determined by the distribution of the two variables, i.e., the contingency coefficient cannot achieve the value of 1 in many cases. According to Sakoda (1977), the contingency coefficient can be adjusted by relating the coefficient to the possible maximum, C/C_max.

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, matrix or data frame specified in `x` (`data`), specification of function arguments (`args`), and list with results (`result`).

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

`cohens.d`, `cor.matrix`, `cramers.v`, `phi.coef`.

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

Sakoda, J.M. (1977). Measures of association for multivariate contingency tables. *Proceedings of the Social Statistics Section of the American Statistical Association (Part III)*, 777-780.

Examples

```
dat <- data.frame(x = c(1, 1, 2, 1, 3, 3, 2, 2, 1, 2),
                  y = c(3, 2, 3, 1, 2, 4, 1, 2, 3, 4),
                  z = c(2, 2, 2, 1, 2, 2, 1, 2, 1, 2))

# Contingency coefficient between x and y
cont.coef(dat[, c("x", "y")])

# Adjusted contingency coefficient between x and y
cont.coef(dat[, c("x", "y")], adjust = TRUE)

# Contingency coefficient matrix between x, y, and z
cont.coef(dat)

# Adjusted contingency coefficient matrix between x, y, and z
cont.coef(dat, adjust = TRUE)
```

---

cor.matrix — *Correlation Matrix with Statistical Significance Testing*

Description

This function computes a correlation matrix and computes significance values (p-values) for testing the hypothesis H0: $\rho = 0$ for all possible pairs of variables.

Usage

```
cor.matrix(x, method = c("pearson", "spearman", "kendall-b", "kendall-c"),
           use = c("listwise", "pairwise"), group = NULL,
           print = c("all", "cor", "n", "p"),
           tri = c("both", "lower", "upper"),
           p.adj = c("none", "bonferroni", "holm", "hochberg", "hommel",
                     "BH", "BY", "fdr"),
           digits = 2, p.digits = 3, as.na = NULL, check = TRUE, output = TRUE)
```

Arguments

x	a matrix or data frame.

method	a character vector indicating which correlation coefficient is to be computed, i.e. "pearson" for Pearson product-moment correlation coefficient (default), "spearman" for Spearman's rank-order correlation coefficient, kendall-b for Kendall's Tau-b correlation coefficient or kendall-c for Kendall-Stuart's Tau-c correlation coefficient.
use	a character vector giving a method for computing a correlation matrix in the presence of missing values, i.e., "listwise" for listwise deletion and "pairwise" for pairwise deletion
group	a numeric vector, character vector of factor as grouping variable to show results for each group separately, i.e., upper triangular for one group and lower triangular for another group. Note that the grouping variable is limited to two groups.
print	a character string or character vector indicating which additional results to show, i.e. "all", for all additional results: "n" for the sample sizes, and "p" for *p*-values.
tri	a character string indicating which triangular of the matrix to show on the console, i.e., both for upper and lower triangular, lower (default) for the lower triangular, and upper for the upper triangular.
p.adj	a character string indicating an adjustment method for multiple testing based on p.adjust, i.e., none (default), bonferroni, holm, hochberg, hommel, BH, BY, or fdr.
digits	an integer value indicating the number of decimal places to be used for displaying correlation coefficients.
p.digits	an integer value indicating the number of decimal places to be used for displaying *p*-values.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
check	logical: if TRUE, argument specification is checked.
output	logical: if TRUE, output is shown on the console.

Value

Returns an object of class misty.object, which is a list with following entries: function call (call), type of analysis type, matrix or data frame specified in x (data), specification of function arguments (args), and list with results (result).

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

See Also

alpha.coef, cohens.d, cont.coef, cramers.v, multilevel.icc, phi.coef, na.auxiliary, size.cor.

Examples

```
dat <- data.frame(group = c("a", "a", "a", "a", "a",
                            "b", "b", "b", "b", "b"),
                  x = c(5, NA, 6, 4, 6, 7, 9, 5, 8, 7),
                  y = c(3, 3, 5, 6, 7, 4, 7, NA, NA, 8),
                  z = c(1, 3, 1, NA, 2, 4, 6, 5, 9, 6), stringsAsFactors = FALSE)

# Pearson product-moment correlation coefficient matrix using pairwise deletion
cor.matrix(dat[, c("x", "y", "z")])

# Spearman's rank-order correlation matrix using pairwise deletion
cor.matrix(dat[, c("x", "y", "z")], method = "spearman")

# Kendall's Tau-b correlation matrix using pairwise deletion
cor.matrix(dat[, c("x", "y", "z")], method = "kendall-b")

# Kendall's Tau-c correlation matrix using pairwise deletion
cor.matrix(dat[, c("x", "y", "z")], method = "kendall-c")

# Pearson product-moment correlation coefficient matrix using pairwise deletion,
# print sample size and significance values
cor.matrix(dat[, c("x", "y", "z")], print = "all")

# Pearson product-moment correlation coefficient matrix using listwise deletion,
# print sample size and significance values
cor.matrix(dat[, c("x", "y", "z")], use = "listwise", print = "all")

# Pearson product-moment correlation coefficient matrix using listwise deletion,
# print sample size and significance values with Bonferroni correction
cor.matrix(dat[, c("x", "y", "z")], use = "listwise", print = "all", p.adj = "bonferroni")

# Pearson product-moment correlation coefficient matrix using pairwise deletion,
# results for group "a" and "b" separately
cor.matrix(dat[, c("x", "y", "z")], group = dat$group, print = "all")
```

---

cramers.v *Cramer's V*

Description

This function computes the (bias-corrected) Cramer's V between two or more than two variables.

Usage

```
cramers.v(x, correct = TRUE, tri = c("both", "lower", "upper"),
          digits = 2, as.na = NULL, check = TRUE, output = TRUE)
```

Arguments

`x`	a matrix or data frame with integer vectors, character vectors or factors.
`correct`	logical: if `TRUE` (default), the bias-corrected Cramer's V is computed.
`tri`	a character string or character vector indicating which triangular of the matrix to show on the console, i.e., `both` for upper and lower triangular, `lower` (default) for the lower triangular, and `upper` for the upper triangular.
`digits`	an integer value indicating the number of decimal places digits to be used for displaying Cramer's V.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown on the console.

Details

Cramer's V can have large bias tending to overestimate the strength of association which depends on the size of the table and the sample size. As proposed by Bergsma (2013) a bias correction can be applied to obtain the bias-corrected Cramer's V.

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, matrix or data frame specified in `x` (`data`), specification of function arguments (`args`), and list with results (`result`).

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

Bergsma, W. (2013). A bias correction for Cramer's V and Tschuprow's T. *Journal of the Korean Statistical Society, 42*, 323-328. https://doi.org/10.1016/j.jkss.2012.10.002

See Also

`cohens.d`, `cont.coef`, `cor.matrix`, `phi.coef`.

Examples

```
dat <- data.frame(x = c(1, 1, 2, 1, 3, 3, 2, 2, 1, 2),
                  y = c(1, 2, 2, 1, 3, 4, 1, 2, 3, 1),
                  z = c(1, 1, 2, 1, 2, 3, 1, 2, 3, 2))

# Bias-corrected Cramer's V between x and y
cramers.v(dat[, c("x", "y")])
```

```
# Cramer's V between x and y
cramers.v(dat[, c("x", "y")], correct = FALSE)

# Bias-corrected Cramer's V matrix between x, y, and z
cramers.v(dat[, c("x", "y", "z")])

# Cramer's V matrix between x, y, and z
cramers.v(dat[, c("x", "y", "z")], correct = FALSE)
```

---

crosstab *Cross Tabulation*

Description

This function creates a two-way and three-way cross tabulation with absolute frequencies and row-wise, column-wise and total percentages.

Usage

```
crosstab(x, print = c("no", "all", "row", "col", "total"), freq = TRUE, split = FALSE,
         na.omit = TRUE, digits = 2, as.na = NULL, check = TRUE, output = TRUE)
```

Arguments

`x`	a matrix or data frame with two or three columns.
`print`	a character string or character vector indicating which percentage(s) to be printed on the console, i.e., no percentages (`"no"`) (default), all percentages (`"all"`), row-wise percentages (`"row"`), column-wise percentages (`"col"`), and total percentages (`"total"`).
`freq`	logical: if `TRUE`, absolute frequencies will be included in the cross tabulation.
`split`	logical: if `TRUE`, output table is split in absolute frequencies and percentage(s).
`na.omit`	logical: if `TRUE`, incomplete cases are removed before conducting the analysis (i.e., listwise deletion).
`digits`	an integer indicating the number of decimal places digits to be used for displaying percentages.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is printed on the console.

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, matrix or data frame specified in `x` (`data`), specification of function arguments (`args`), and list with results (`result`).

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

freq, descript, multilevel.descript, na.descript.

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

Examples

```
dat <- data.frame(x1 = c(1, 2, 2, 1, 1, 2, 2, 1, 1, 2),
                  x2 = c(1, 2, 2, 1, 2, 1, 1, 1, 2, 1),
                  x3 = c(-99, 2, 1, 1, 1, 2, 2, 2, 2, 1))

# Cross Tabulation for x1 and x2
crosstab(dat[, c("x1", "x2")])

# Cross Tabulation for x1 and x2
# print all percentages
crosstab(dat[, c("x1", "x2")], print = "all")

# Cross Tabulation for x1 and x2
# print row-wise percentages
crosstab(dat[, c("x1", "x2")], print = "row")

# Cross Tabulation for x1 and x2
# print col-wise percentages
crosstab(dat[, c("x1", "x2")], print = "col")

# Cross Tabulation x1 and x2
# print total percentages
crosstab(dat[, c("x1", "x2")], print = "total")

# Cross Tabulation for x1 and x2
# print all percentages, split output table
crosstab(dat[, c("x1", "x2")], print = "all", split = TRUE)

# Cross Tabulation for x1 and x3
# do not apply listwise deletion, convert value -99 to NA
crosstab(dat[, c("x1", "x3")], na.omit = FALSE, as.na = -99)

# Cross Tabulation for x1 and x3
# print all percentages, do not apply listwise deletion, convert value -99 to NA
crosstab(dat[, c("x1", "x3")], print = "all", na.omit = FALSE, as.na = -99)

# Cross Tabulation for x1, x2, and x3
crosstab(dat[, c("x1", "x2", "x3")])

# Cross Tabulation for x1, x2, and x3
# print all percentages
```

```
crosstab(dat[, c("x1", "x2", "x3")], print = "all")

# Cross Tabulation for x1, x2, and x3
# print all percentages, split output table
crosstab(dat[, c("x1", "x2", "x3")], print = "all", split = TRUE)
```

---

descript *Descriptive Statistics*

Description

This function computes summary statistics for one or more variables optionally by a grouping variable.

Usage

```
descript(x,
         print = c("all", "n", "nNA", "pNA", "m", "var", "sd", "min", "p25",
                   "med", "p75", "max", "range", "iqr", "skew", "kurt"),
         group = NULL, split = NULL, sort.var = FALSE, na.omit = FALSE,
         digits = 2, as.na = NULL, check = TRUE, output = TRUE)
```

Arguments

`x`	a numeric vector, matrix or data frame with numeric variables, i.e., factors and character variables are excluded from `x` before conducting the analysis.
`print`	a character vector indicating which statistical measures to be printed on the console, i.e. `n` (number of observations), `nNA` (number of missing values), `pNA` (percentage of missing values), `m` (arithmetic mean), `var` (variance), `sd` (standard deviation), `med` (median),`min` (minimum), `p25` (25th percentile, first quartile), `p75` (75th percentile, third quartile), `max` (maximum), `range` (range), `iqr` (interquartile range), `skew` (skewness), and `kurt` (excess kurtosis). The default setting is `print = ("n","nNA","pNA","m","sd","min","max","skew","kurt")`.
`group`	a numeric vector, character vector or factor as grouping variable.
`split`	a numeric vector, character vector or factor as split variable.
`sort.var`	logical: if `TRUE`, output table is sorted by variables when specifying `group`.
`na.omit`	logical: if `TRUE`, incomplete cases are removed before conducting the analysis (i.e., listwise deletion).
`digits`	an integer value indicating the number of decimal places to be used.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis. Note that `as.na()` function is only applied to `x`, but not to `group` or `split`.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown on the console.

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, matrix or data frame specified in `x` (`data`), specification of function arguments (`args`), and list with results (`result`).

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

See Also

`crosstab`, `freq`, `multilevel.descript`, `na.descript`.

Examples

```
dat <- data.frame(group1 = c(1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2),
                  group2 = c(1, 1, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2),
                  x1 = c(3, 1, 4, 2, 5, 3, 2, 4, NA, 4, 5, 3),
                  x2 = c(4, NA, 3, 6, 3, 7, 2, 7, 5, 1, 3, 6),
                  x3 = c(7, 8, 5, 6, 4, NA, 8, NA, 6, 5, 8, 6))

# Descriptive statistics for x1
descript(dat$x1)

# Descriptive statistics for x1, print results with 3 digits
descript(dat$x1, digits = 3)

# Descriptive statistics for x1, convert value 4 to NA
descript(dat$x1, as.na = 4)

# Descriptive statistics for x1, print all available statistical measures
descript(dat$x1, print = "all")

# Descriptive statistics for x1, x2, and x3
descript(dat[, c("x1", "x2", "x3")])

# Descriptive statistics for x1, x2, and x3,
# listwise deletion for missing data
descript(dat[, c("x1", "x2", "x3")], na.omit = TRUE)

# Descriptive statistics for x1, x2, and x3,
# analysis by group1 separately
descript(dat[, c("x1", "x2", "x3")], group = dat$group1)

# Descriptive statistics for x1, x2, and x3,
# analysis by group1 separately, sort by variables
```

```
descript(dat[, c("x1", "x2", "x3")], group = dat$group1, sort.var = TRUE)

# Descriptive statistics for x1, x2, and x3,
# split analysis by group1
descript(dat[, c("x1", "x2", "x3")], split = dat$group1)

# Descriptive statistics for x1, x2, and x3,
# analysis by group1 separately, split analysis by group2
descript(dat[, c("x1", "x2", "x3")], group = dat$group1, split = dat$group2)
```

`df.duplicated` *Extract Duplicated or Unique Rows*

Description

This function extracts duplicated or unique rows from a matrix or data frame.

Usage

```
df.duplicated(x, ..., first = TRUE, keep.all = TRUE, from.last = FALSE,
              keep.row.names = TRUE, check = TRUE)

df.unique(x, ..., keep.all = TRUE, from.last = FALSE, keep.row.names = TRUE,
          check = TRUE)
```

Arguments

`x`	a matrix or data frame.
`...`	a variable or multiple variables which are specified without quotes `''` or double quotes `""` used to determine duplicated or unique rows. By default, all variables in `x` are used.
`first`	logical: if `TRUE`, the `df.duplicated()` function will return duplicated rows including the first of identical rows.
`keep.all`	logical: if `TRUE`, the function will return all variables in `x` after extracting duplicated or unique rows based on the variables specified in the argument `...`.
`from.last`	logical: if `TRUE`, duplication will be considered from the reversed side, i.e., the last of identical rows would correspond to `duplicated = FALSE`. Note that this argument is only used when `first = FALSE`.
`keep.row.names`	logical: if `TRUE`, the row names from `x` are kept, otherwise they are set to `NULL`.
`check`	logical: if `TRUE`, argument specification is checked.

Details

Note that `df.unique(x)` is equivalent to `unique(x)`. That is, the main difference between the `df.unique()` and the `unique()` function is that the `df.unique()` function provides the `...` argument to specify a variable or multiple variables which are used to determine unique rows.

Value

Returns duplicated or unique rows of the matrix or data frame in x.

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

df.merge, df.rbind, df.rename, df.sort

Examples

```
dat <- data.frame(x1 = c(1, 1, 2, 1, 4),
                  x2 = c(1, 1, 2, 1, 6),
                  x3 = c(2, 2, 3, 2, 6),
                  x4 = c(1, 1, 2, 2, 4),
                  x5 = c(1, 1, 4, 4, 3))

#--------------------------------------
# df.duplicated() function

# Extract duplicated rows based on all variables
df.duplicated(dat)

# Extract duplicated rows based on x4
df.duplicated(dat, x4)

# Extract duplicated rows based on x2 and x3
df.duplicated(dat, x2, x3)

# Extract duplicated rows based on all variables
# exclude first of identical rows
df.duplicated(dat, first = FALSE)

# Extract duplicated rows based on x2 and x3
# do not return all variables
df.duplicated(dat, x2, x3, keep.all = FALSE)

# Extract duplicated rows based on x4
# consider duplication from the reversed side
df.duplicated(dat, x4, first = FALSE, from.last = TRUE)

# Extract duplicated rows based on x2 and x3
# set row names to NULL
df.duplicated(dat, x2, x3, keep.row.names = FALSE)
```

```
#--------------------------------------
# df.unique() function

# Extract unique rows based on all variables
unique(dat)

# Extract unique rows based on x4
df.unique(dat, x4)

# Extract unique rows based on x1, x2, and x3
df.unique(dat, x1, x2, x3)

# Extract unique rows based on x2 and x3
# do not return all variables
df.unique(dat, x2, x3, keep.all = FALSE)

# Extract unique rows based on x4
# consider duplication from the reversed side
df.unique(dat, x4, from.last = TRUE)

# Extract unique rows based on x2 and x3
# set row names to NULL
df.unique(dat, x2, x3, keep.row.names = FALSE)
```

`df.merge` *Merge Multiple Data Frames*

Description

This function merges data frames by a common column (i.e., matching variable).

Usage

```
df.merge(..., by, all = TRUE, check = TRUE, output = TRUE)
```

Arguments

`...`	a sequence of matrices or data frames and/or matrices to be merged to one.
`by`	a character string indicating the column used for merging (i.e., matching variable), see 'Details'.
`all`	logical: if `TRUE`, then extra rows with `NA`s will be added to the output for each row in a data frame that has no matching row in another data frame.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown on the console.

Details

There are following requirements for merging multiple data frames: First, each data frame has the same matching variable specified in the by argument. Second, matching variable in the data frames have all the same class. Third, there are no duplicated values in the matching variable in each data frame. Fourth, there are no missing values in the matching variables. Last, there are no duplicated variable names across the data frames except for the matching variable.

Note that it is possible to specify data frames matrices and/or in the argument `...`. However, the function always returns a data frame.

Value

Returns a merged data frame.

Author(s)

Takuya Yanagida `<takuya.yanagida@univie.ac.at>`

See Also

`df.duplicated`, `df.unique`, `df.rbind`, `df.rename`, `df.sort`

Examples

```
adat <- data.frame(id = c(1, 2, 3),
                   x1 = c(7, 3, 8))

bdat <- data.frame(id = c(1, 2),
                   x2 = c(5, 1))

cdat <- data.frame(id = c(2, 3),
                   y3 = c(7, 9))

ddat <- data.frame(id = 4,
                   y4 = 6)

# Merge adat, bdat, cdat, and data by the variable id
df.merge(adat, bdat, cdat, ddat, by = "id")

# Do not show output on the console
df.merge(adat, bdat, cdat, ddat, by = "id", output = FALSE)

## Not run:
#----------------------------------------#'
# Error messages

adat <- data.frame(id = c(1, 2, 3),
                   x1 = c(7, 3, 8))

bdat <- data.frame(code = c(1, 2, 3),
                   x2 = c(5, 1, 3))
```

```
cdat <- data.frame(id = factor(c(1, 2, 3)),
                   x3 = c(5, 1, 3))

ddat <- data.frame(id = c(1, 2, 2),
                   x2 = c(5, 1, 3))

edat <- data.frame(id = c(1, NA, 3),
                   x2 = c(5, 1, 3))

fdat <- data.frame(id = c(1, 2, 3),
                   x1 = c(5, 1, 3))

# Error: Data frames do not have the same matching variable specified in 'by'.
df.merge(adat, bdat, by = "id")

# Error: Matching variable in the data frames do not all have the same class.
df.merge(adat, cdat, by = "id")

# Error: There are duplicated values in the matching variable specified in 'by'.
df.merge(adat, ddat, by = "id")

# Error: There are missing values in the matching variable specified in 'by'.
df.merge(adat, edat, by = "id")

#' # Error: There are duplicated variable names across data frames.
df.merge(adat, fdat, by = "id")

## End(Not run)
```

---

df.rbind *Combine Data Frames by Rows, Filling in Missing Columns*

Description

This function takes a sequence of data frames and combines them by rows, while filling in missing columns with NAs.

Usage

```
df.rbind(...)
```

Arguments

...	a sequence of data frame to be row bind together. This argument can be a list of data frames, in which case all other arguments are ignored. Any NULL inputs are silently dropped. If all inputs are NULL, the output is also NULL.

Details

This is an enhancement to rbind that adds in columns that are not present in all inputs, accepts a sequence of data frames, and operates substantially faster.

Column names and types in the output will appear in the order in which they were encountered.

Unordered factor columns will have their levels unified and character data bound with factors will be converted to character. POSIXct data will be converted to be in the same time zone. Array and matrix columns must have identical dimensions after the row count. Aside from these there are no general checks that each column is of consistent data type.

Note that this function is a copy of the `rbind.fill()` function in the **plyr** package by Hadley Wickham.

Value

Returns a single data frame

Author(s)

Hadley Wickham

References

Wickham, H. (2011). The split-apply-combine strategy for data analysis. *Journal of Statistical Software, 40*, 1-29. https://doi.org/10.18637/jss.v040.i01

Wickham, H. (2019). plyr: Tools for Splitting, Applying and Combining Data. R package version 1.8.5.

See Also

df.duplicated, df.unique, df.merge, df.rename, df.sort

Examples

```
adat <- data.frame(id = c(1, 2, 3),
                   a = c(7, 3, 8),
                   b = c(4, 2, 7))

bdat <- data.frame(id = c(4, 5, 6),
                   a = c(2, 4, 6),
                   c = c(4, 2, 7))

cdat <- data.frame(id = c(7, 8, 9),
                   a = c(1, 4, 6),
                   d = c(9, 5, 4))

df.rbind(adat, bdat, cdat)
```

df.rename — *Rename Columns in a Matrix or Variables in a Data Frame*

Description

This function renames columns in a matrix or variables in a data frame by specifying a character string or character vector indicating the columns or variables to be renamed and a character string or character vector indicating the corresponding replacement values.

Usage

```
df.rename(x, from, to, check = TRUE)
```

Arguments

`x`	a matrix or data frame.
`from`	a character string or character vector indicating the column(s) or variable(s) to be renamed.
`to`	a character string or character vector indicating the corresponding replacement values for the column(s) or variable(s) specified in the argument `name`.
`check`	logical: if `TRUE`, argument specification is checked.

Value

Returns a matrix or data frame with renamed columns or variables.

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

See Also

`df.duplicated`, `df.unique`, `df.merge`, `df.rbind`, `df.sort`

Examples

```
dat <- data.frame(a = c(3, 1, 6),
                  b = c(4, 2, 5),
                  c = c(7, 3, 1))

# Rename variable b in the data frame 'dat' to y
df.rename(dat, from = "b", to = "y")

# Rename variabley a, b, and c in the data frame 'dat' to x, y, and z
df.rename(dat, from = c("a", "b", "c"), to = c("x", "y", "z"))
```

`df.sort` *Data Frame Sorting*

Description

This function arranges a data frame in increasing or decreasing order according to one or more variables.

Usage

```
df.sort(x, ..., decreasing = FALSE, check = TRUE)
```

Arguments

`x`	a data frame.
`...`	a sorting variable or a sequence of sorting variables which are specified without quotes `''` or double quotes `""`.
`decreasing`	logical: if `TRUE`, the sort is decreasing.
`check`	logical: if `TRUE`, argument specification is checked.

Value

Returns data frame `x` sorted according to the variables specified in `...`, a matrix will be coerced to a data frame.

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Knuth, D. E. (1998) *The Art of Computer Programming, Volume 3: Sorting and Searching* (2nd ed.). Addison-Wesley.

See Also

`df.duplicated`, `df.unique`, `df.merge`, `df.rbind`, `df.rename`

Examples

```
dat <- data.frame(x = c(5, 2, 5, 5, 7, 2),
                  y = c(1, 6, 2, 3, 2, 3),
                  z = c(2, 1, 6, 3, 7, 4))

# Sort data frame 'dat' by "x" in increasing order
df.sort(dat, x)

# Sort data frame 'dat' by "x" in decreasing order
df.sort(dat, x, decreasing = TRUE)

# Sort data frame 'dat' by "x" and "y" in increasing order
df.sort(dat, x, y)

# Sort data frame 'dat' by "x" and "y" in decreasing order
df.sort(dat, x, y, decreasing = TRUE)
```

`dummy.c` *Dummy Coding*

Description

This function creates $k - 1$ dummy coded 0/1 variables for a vector with k distinct values.

Usage

```
dummy.c(x, ref = NULL, names = "d", as.na = NULL, check = TRUE)
```

Arguments

`x`	a numeric vector with integer values, character vector or factor.
`ref`	a numeric value or character string indicating the reference group. By default, the last category is selected as reference group.
`names`	a character string or character vector indicating the names of the dummy variables. By default, variables are named `"d"` with the category compared to the reference category (e.g., `"d1"` and `"d2"`). Variable names can be specified using a character string (e.g., `names = "dummy_"` leads to `dummy_1` and `dummy_2`) or a character vector matching the number of dummy coded variables (e.g. `names = c("x.3_1","x.3_2")`) which is the number of unique categories minus one.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`check`	logical: if `TRUE`, argument specification is checked.

Value

Returns a matrix with k - 1 dummy coded 0/1 variables.

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. New York: John Wiley & Sons.

Examples

```
dat <- data.frame(x = c(1, 1, 1, 2, 2, 2, 3, 3, 3),
                  y = c("a", "a", "a", "b", "b", "b", "c", "c", "c"),
                  z = factor(c("B", "B", "B", "A", "A", "A", "C", "C", "C")),
                  stringsAsFactors = FALSE)

# Dummy coding of a numeric variable, reference = 3
dummy.c(dat$x)

# Dummy coding of a numeric variable, reference = 1
dummy.c(dat$x, ref = 1)

# Dummy coding of a numeric variable, reference = 3
# assign user-specified variable names
dummy.c(dat$x, names = c("x.3_1", "x.3_2"))

# Dummy coding of a numeric variable, reference = 3
# assign user-specified variable names and attach to the data frame
dat <- data.frame(dat, dummy.c(dat$x, names = c("x.3_1", "x.3_2")))

# Dummy coding of a character variable, reference = "c"
dummy.c(dat$y)

# Dummy coding of a character variable, reference = "a"
dummy.c(dat$y, ref = "a")

# Dummy coding of a numeric variable, reference = "c"
# assign user-specified variable names
dummy.c(dat$y, names = c("y.c_a", "y.c_b"))

# Dummy coding of a character variable, reference = "c"
# assign user-specified variable names and attach to the data frame
dat <- data.frame(dat, dummy.c(dat$y, names = c("y.c_a", "y.c_b")))

# Dummy coding of a factor, reference = "C"
dummy.c(dat$z)

# Dummy coding of a factor, reference = "A"
dummy.c(dat$z, ref = "A")

# Dummy coding of a numeric variable, reference = "C"
# assign user-specified variable names
dummy.c(dat$z, names = c("z.C_A", "z.C_B"))
```

```
# Dummy coding of a factor, reference = "C"
# assign user-specified variable names and attach to the data frame
dat <- data.frame(dat, dummy.c(dat$z, names = c("z.C_A", "z.C_B")))
```

---

eta.sq *Eta Squared*

Description

This function computes eta squared for one or more outcome variables in combination with one or more grouping variables.

Usage

```
eta.sq(x, group, digits = 2, as.na = NULL, check = TRUE, output = TRUE)
```

Arguments

`x`	a numeric vector, matrix or data frame with numeric vectors for the outcome variables.
`group`	a vector, matrix or data frame with integer vectors, character vectors or factors for the grouping variables.
`digits`	an integer value indicating the number of decimal places to be used for displaying eta squared.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis. Note that `as.na()` function is only applied to the argument `x`.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown on the console.

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, matrix or data frame specified in `x` (`data`), specification of function arguments (`args`), and list with results (`result`).

Author(s)

Takuya Yanagida `<takuya.yanagida@univie.ac.at>`

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. New York: John Wiley & Sons.

See Also

cohens.d, cont.coef, cor.matrix, cramers.v, phi.coef

Examples

```
dat <- data.frame(x1 = c(1, 1, 1, 1, 2, 2, 2, 2, 2),
                  x2 = c(1, 1, 1, 2, 2, 2, 3, 3, 3),
                  y1 = c(3, 2, 4, 5, 6, 4, 7, 5, 7),
                  y2 = c(2, 4, 1, 5, 3, 3, 4, 6, 7))

# Eta squared for y1 explained by x1
eta.sq(dat$y1, group = dat$x1)

# Eta squared for y1 and y2 explained by x1 and x2
eta.sq(dat[, c("y1", "y2")], group = dat[, c("x1", "x2")])
```

---

freq *Frequency Table*

Description

This function computes a frequency table with absolute and percentage frequencies for one or more than one variable.

Usage

```
freq(x, print = c("no", "all", "perc", "v.perc"), freq = TRUE, split = FALSE,
     labels = TRUE, val.col = FALSE, exclude = 15, digits = 2, as.na = NULL,
     check = TRUE, output = TRUE)
```

Arguments

`x`	a vector, factor, matrix or data frame.
`print`	a character string indicating which percentage(s) to be printed on the console, i.e., no percentages (`"no"`), all percentages (`"all"`), percentage frequencies (`"print"`), and valid percentage frequencies (`"v.perc"`). Default setting when specifying one variable in `x` is `print = "all"`, while default setting when specifying more than one variable in `x` is `print = "no"` unless `split = TRUE`.
`freq`	logical: if `TRUE` (default), absolute frequencies will be shown on the console.
`split`	logical: if `TRUE`, output table is split by variables when specifying more than one variable in `x`.
`labels`	logical: if `TRUE` (default), labels for the factor levels will be used.
`val.col`	logical: if `TRUE`, values are shown in the columns, variables in the rows.
`exclude`	an integer value indicating the maximum number of unique values for variables to be included in the analysis when specifying more than one variable in `x`, i.e., variables with the number of unique values exceeding `exclude` will be excluded from the analysis.

`digits`	an integer value indicating the number of decimal places to be used for displaying percentages.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown on the console.

Details

By default, the function displays the absolute and percentage frequencies when specifying one variable in the argument `x`, while the function displays only the absolute frequencies when more than one variable is specified. The function displays valid percentage frequencies only in the presence of missing values and excludes variables with all values missing from the analysis. Note that it is possible to mix numeric variables, factors, and character variables in the data frame specified in the argument `x`.

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, matrix or data frame specified in `x` (`data`), specification of function arguments (`args`), and list with results (`result`).

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Becker, R. A., Chambers, J. M., & Wilks, A. R. (1988). *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`crosstab`, `descript`, `multilevel.descript`, `na.descript`.

Examples

```
dat <- data.frame(x1 = c(3, 3, 2, 3, 2, 3, 3, 2, 1, -99),
                  x2 = c(2, 2, 1, 3, 1, 1, 3, 3, 2, 2),
                  y1 = c(1, 4, NA, 5, 2, 4, 3, 5, NA, 1),
                  y2 = c(2, 3, 4, 3, NA, 4, 2, 3, 4, 5),
                  z = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10))

# Frequency table for one variable
freq(dat$x1)

# Frequency table for one variable,
# values shown in columns
freq(dat$x1, val.col = TRUE)
```

```
# Frequency table for one variable,
# convert value -99 into NA
freq(dat$x1, as.na = -99)

# Frequency table for one variable
# use 3 digit for displaying percentages
freq(dat$x1, digits = 3)

# Frequency table for more than one variable
freq(dat[, c("x1", "x2", "y1", "y2")])

# Frequency table for more than one variable,
# values shown in columns
freq(dat[, c("x1", "x2", "y1", "y2")], val.col = TRUE)

# Frequency table for more than one variable,
# with percentage frequencies
freq(dat[, c("x1", "x2", "y1", "y2")], print = "all")

# Frequency table for more than one variable,
# with percentage frequencies, values shown in columns
freq(dat[, c("x1", "x2", "y1", "y2")], print = "all", val.col = TRUE)

# Frequency table for more than one variable,
# split output table
freq(dat[, c("x1", "x2", "y1", "y2")], split = TRUE)

# Frequency table for more than one variable,
# exclude variables with more than 5 unique values
freq(dat, exclude = 5)

# Frequency table for a factor
freq(factor(c("a", "a", "b", "c", "b")))

# Frequency table for one variable,
# do not use labels of the factor levels
freq(factor(c("a", "a", "b", "c", "b")), labels = FALSE)
```

`group.scores` *Group Scores*

Description

This function computes group means by default.

Usage

```
group.scores(x, group, fun = c("mean", "sum", "median", "var", "sd", "min", "max"),
             expand = TRUE, as.na = NULL, check = TRUE)
```

Arguments

x	a numeric vector.
group	a integer vector, character vector, or factor representing the grouping structure (i.e., group variable).
fun	character string indicating the function used to compute group scores, default: "mean".
expand	logical: if TRUE, vector of group scores is expanded to match the input vector x.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis. Note that as.na() function is only applied to the argument x, but not to group.
check	logical: if TRUE, argument specification is checked.

Value

Returns a numeric vector containing group scores with the same length as x if expand = TRUE or with the length length(unique(group)) if expand = FALSE.

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

#' @seealso scores, multilevel.descript, multilevel.icc

References

Hox, J., Moerbeek, M., & van de Schoot, R. (2018). *Multilevel analysis: Techniques and applications* (3rd. ed.). Routledge.

Snijders, T. A. B., & Bosker, R. J. (2012). *Multilevel analysis: An introduction to basic and advanced multilevel modeling* (2nd ed.). Sage Publishers.

Examples

```
dat.ml <- data.frame(id = c(1, 2, 3, 4, 5, 6, 7, 8, 9),
                     group = c(1, 1, 1, 2, 2, 2, 3, 3, 3),
                     x = c(4, 2, 5, 6, 3, 4, 1, 3, 4))

# Compute group means and expand to match the input x
group.scores(dat.ml$x, group = dat.ml$group)

# Compute standard deviation for each group and expand to match the input x
group.scores(dat.ml$x, group = dat.ml$group, fun = "sd")

# Compute group means without expanding the vector
group.scores(dat.ml$x, group = dat.ml$group, expand = FALSE)
```

`kurtosis` *Excess Kurtosis*

Description

This function computes the excess kurtosis.

Usage

```
kurtosis(x, as.na = NULL, check = TRUE)
```

Arguments

`x`	a numeric vector.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`check`	logical: if `TRUE`, argument specification is checked.

Details

The same method for estimating kurtosis is used in SAS and SPSS. Missing values (`NA`) are stripped before the computation. Note that at least 4 observations are needed to compute excess kurtosis.

Value

Returns the estimated excess kurtosis of `x`.

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. New York: John Wiley & Sons.

See Also

`skewness`

Examples

```
# Set seed of the random number generation
set.seed(123)
# Generate random numbers according to N(0, 1)
x <- rnorm(100)

# Compute excess kurtosis
kurtosis(x)
```

`levenes.test` *Levene's Test for Homogeneity of Variance*

Description

This function computes Levene's test for homogeneity of variance across two or more independent groups.

Usage

```
levenes.test(formula, data, method = c("median", "mean"), conf.level = 0.95,
             digits = 2, p.digits = 3, as.na = NULL, check = TRUE, output = TRUE)
```

Arguments

`formula`	a formula of the form `y ~ group` where `y` is a numeric variable giving the data values and `group` a numeric variable, character variable or factor with two or more than two values or factor levels giving the corresponding groups.
`data`	a matrix or data frame containing the variables in the formula `formula`.
`method`	a character string specifying the method to compute the center of each group, i.e. `method = "median"` (default) to compute the Levene's test basd on the median (aka Brown-Forsythe test) or `method = "mean"` to compute the Levene's test based on the arithmetic mean.
`conf.level`	a numeric value between 0 and 1 indicating the confidence level of the interval.
`digits`	an integer value indicating the number of decimal places to be used for displaying results.
`p.digits`	an integer value indicating the number of decimal places to be used for displaying the *p*-value.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown.

Details

Levene's test is equivalent to a one-way analysis of variance (ANOVA) with the absolute deviations of observations from the mean of each group as dependent variable (`center = "mean"`). Brown and Forsythe (1974) modified the Levene's test by using the absolute deviations of observations from the median (`center = "median"`). By default, the Levene's test uses the absolute deviations of observations from the median.

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, formula (`formula`), data frame with the outcome and grouping variable (`data`), specification of function arguments (`args`), and a list with descriptive statistics including confidence intervals and an object of class `"anova"` (`result`).

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Brown, M. B., & Forsythe, A. B. (1974). Robust tests for the equality of variances. *Journal of the American Statistical Association, 69*, 364-367.

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

See Also

t.test, aov

Examples

```
dat <- data.frame(y = c(2, 3, 4, 5, 5, 7, 8, 4, 5, 2, 4, 3),
                  group = c(1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3))

# Levene's test based on the median with 95% confidence interval
levenes.test(y ~ group, data = dat)

# Levene's test based on the arithmetic mean  with 95% confidence interval
levenes.test(y ~ group, data = dat, method = "mean")

# Levene's test based on the median with 99% confidence interval
levenes.test(y ~ group, data = dat, conf.level = 0.99)
```

---

mgsub *Multiple Pattern Matching And Replacements*

Description

This function is a multiple global string replacement wrapper that allows access to multiple methods of specifying matches and replacements.

Usage

```
mgsub(pattern, replacement, x, recycle = FALSE, ...)
```

Arguments

`pattern`	a character vector with character strings to be matched.
`replacement`	a character vector equal in length to `pattern` or of length one which are a replacement for matched patterns.
`x`	a character vector where matches and replacements are sought.
`recycle`	logical: if `TRUE`, replacement is recycled if lengths differ.
`...`	additional arguments to pass to the `regexpr` or `sub` function.

Details

Note that the function was adapted from the mgsub() function in the **mgsub** package by Mark Ewing (2019).

Value

Return a character vector of the same length and with the same attributes as x (after possible coercion to character).

Author(s)

Mark Ewing

References

Mark Ewing (2019). *mgsub: Safe, Multiple, Simultaneous String Substitution*. R package version 1.7.1. https://CRAN.R-project.org/package=mgsub

See Also

stromit, trim

Examples

```
string <- c("hey ho, let's go!")
mgsub(c("hey", "ho"), c("ho", "hey"), string)

string <- "they don't understand the value of what they seek."
mgsub(c("the", "they"), c("a", "we"), string)

string <- c("hey ho, let's go!")
mgsub(c("hey", "ho"), "yo", string, recycle = TRUE)

string <- "Dopazamine is not the same as dopachloride or dopastriamine, yet is still fake."
mgsub(c("[Dd]opa([^ ]*?mine)","fake"), c("Meta\\1","real"), string)
```

---

multilevel.descript *Multilevel Descriptive Statistics*

Description

This function computes descriptive statistics for multilevel data, e.g. average group size, intraclass correlation coefficient, design effect and effective sample size.

Usage

```
multilevel.descript(x, group, method = c("aov", "lme4", "nlme"), REML = TRUE,
                    digits = 2, icc.digits = 3, as.na = NULL, check = TRUE,
                    output = TRUE)
```

Arguments

x	a vector, matrix or data frame.
group	a vector representing the grouping structure (i.e., group variable).
method	a character string indicating the method used to estimate intraclass correlation coefficients, i.e., "aov" (default) ICC estimated using the aov function, "lme4" ICC estimated using the lmer function in the **lme4** package, "nlme" ICC estimated using the lme function in the **nlme** package.
REML	logical: if TRUE, restricted maximum likelihood is used to estimate the null model when using the lmer() function in the **lme4** package or the lme() function in the **nlme** package.
digits	an integer value indicating the number of decimal places to be used.
icc.digits	an integer indicating the number of decimal places to be used for displaying intraclass correlation coefficients.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis. Note that as.na() function is only applied to x but not to group.
check	logical: if TRUE, argument specification is checked.
output	logical: if TRUE, output is shown on the console.

Details

Note that this function is restricted to two-level models.

Value

Returns an object of class misty.object, which is a list with following entries: function call (call), type of analysis type, matrix or data frame specified in x (data), specification of function arguments (args), and list with results (result).

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Hox, J., Moerbeek, M., & van de Schoot, R. (2018). *Multilevel analysis: Techniques and applications* (3rd. ed.). Routledge.

Snijders, T. A. B., & Bosker, R. J. (2012). *Multilevel analysis: An introduction to basic and advanced multilevel modeling* (2nd ed.). Sage Publishers.

See Also

multilevel.icc

Examples

```
dat <- data.frame(id = c(1, 2, 3, 4, 5, 6, 7, 8, 9),
                  group = c(1, 1, 1, 1, 2, 2, 3, 3, 3),
                  x1 = c(2, 3, 2, 2, 1, 2, 3, 4, 2),
                  x2 = c(3, 2, 2, 1, 2, 1, 3, 2, 5),
                  x3 = c(2, 1, 2, 2, 3, 3, 5, 2, 4))

# Multilevel descriptive statistics for x1
multilevel.descript(dat$x1, group = dat$group)

# Multilevel descriptive statistics for x1, print ICC with 5 digits
multilevel.descript(dat$x1, group = dat$group, icc.digits = 5)

# Multilevel descriptive statistics for x1, convert value 1 to NA
multilevel.descript(dat$x1, group = dat$group, as.na = 1)

# Multilevel descriptive statistics for x1,
# use lmer() function in the lme4 package to estimate ICC
multilevel.descript(dat$x1, group = dat$group, method = "lme4")

# Multilevel descriptive statistics for x1, x2, and x3
multilevel.descript(dat[, c("x1", "x2", "x3")], group = dat$group)
```

`multilevel.icc` *Intraclass Correlation Coefficient, ICC(1) and ICC(2)*

Description

This function computes the intraclass correlation coefficient ICC(1), i.e., proportion of the total variance explained by the grouping structure, and ICC(2), i.e., reliability of aggregated variables.

Usage

```
multilevel.icc(x, group, type = 1, method = c("aov", "lme4", "nlme"), REML = TRUE,
               as.na = NULL, check = TRUE)
```

Arguments

`x`	a vector, matrix or data frame.
`group`	a vector representing the grouping structure (i.e., group variable).
`type`	numeric value indicating the type of intraclass correlation coefficient, i.e., `type = 1` for ICC(1) and `type = 2` for ICC(2).
`method`	a character string indicating the method used to estimate intraclass correlation coefficients, i.e., `method = "aov"` (default) ICC estimated using the `aov` function, `method = "lme4"` ICC estimated using the `lmer` function in the **lme4** package, `method = "nlme"` ICC estimated using the `lme` function in the **nlme** package.

REML	logical: if TRUE, restricted maximum likelihood is used to estimate the null model when using the lmer function in the **lme4** package or the lme function in the **nlme** package.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
check	logical: if TRUE, argument specification is checked.

Details

Note that this function is restricted to two-level models.

Value

Returns a numeric vector with intraclass correlation coefficient(s).

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Hox, J., Moerbeek, M., & van de Schoot, R. (2018). *Multilevel analysis: Techniques and applications* (3rd. ed.). Routledge.

Snijders, T. A. B., & Bosker, R. J. (2012). *Multilevel analysis: An introduction to basic and advanced multilevel modeling* (2nd ed.). Sage Publishers.

See Also

multilevel.descript

Examples

```
dat <- data.frame(id = c(1, 2, 3, 4, 5, 6, 7, 8, 9),
                  group = c(1, 1, 1, 1, 2, 2, 3, 3, 3),
                  x1 = c(2, 3, 2, 2, 1, 2, 3, 4, 2),
                  x2 = c(3, 2, 2, 1, 2, 1, 3, 2, 5),
                  x3 = c(2, 1, 2, 2, 3, 3, 5, 2, 4))

# ICC(1) for x1
multilevel.icc(dat$x1, group = dat$group)

# ICC(1) for x1, convert value 1 to NA
multilevel.icc(dat$x1, group = dat$group, as.na = 1)

# ICC(2) for x1
multilevel.icc(dat$x1, group = dat$group, type = 2)

# ICC(1) for x1,
# use lmer() function in the lme4 package to estimate ICC
multilevel.icc(dat$x1, group = dat$group, method = "lme4")
```

```
# ICC(1) for x1, x2, and x3
multilevel.icc(dat[, c("x1", "x2", "x3")], group = dat$group)
```

`na.as` *Replace Missing Values With User-Specified Values*

Description

This function replaces `NA` in a vector, factor, matrix or data frame with user-specified values in the argument `value`.

Usage

```
na.as(x, value, as.na = NULL, check = TRUE)
```

Arguments

`x`	a vector, factor, matrix or data frame.
`value`	a numeric value or character string with which `NA` is replaced.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`check`	logical: if `TRUE`, argument specification is checked.

Value

Returns `x` with `NA` replaced with the numeric value or character string specified in `value`.

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

`as.na`, `na.auxiliary`, `na.coverage`, `na.descript`, `na.indicator`, `na.pattern`, `na.prop`.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Examples

```
#--------------------------------------
# Numeric vector
x.num <- c(1, 3, NA, 4, 5)

# Replace NA with 2
na.as(x.num, value = 2)

#--------------------------------------
```

```
# Character vector
x.chr <- c("a", NA, "c", "d", "e")

# Replace NA with "b"
na.as(x.chr, value = "b")

#--------------------------------------
# Factor
x.factor <- factor(c("a", "a", NA, NA, "c", "c"))

# Replace NA with "b"
na.as(x.factor, value = "b")

#--------------------------------------
# Matrix
x.mat <- matrix(c(1, NA, 3, 4, 5, 6), ncol = 2)

# Replace NA with 2
na.as(x.mat, value = 2)

#--------------------------------------
# Data frame
x.df1 <- data.frame(x1 = c(NA, 2, 3),
                    x2 = c(2, NA, 3),
                    x3 = c(3, NA, 2))

# Replace NA with -99
na.as(x.df1, value = -99)

#--------------------------------------
# Recode value in data frame
x.df2 <- data.frame(x1 = c(1, 2, 30),
                    x2 = c(2, 1, 30),
                    x3 = c(30, 1, 2))

# Replace 30 with NA and then replace NA with 3
na.as(x.df2, value = 3, as.na = 30)
```

na.auxiliary *Auxiliary variables*

Description

This function computes (1) Pearson product-moment correlation matrix to identify variables related to the incomplete variable and (2) Cohen's d comparing cases with and without missing values to identify variables related to the probability of missigness.

Usage

```
na.auxiliary(x, tri = c("both", "lower", "upper"), weighted = TRUE, correct = FALSE,
             digits = 2, as.na = NULL, check = TRUE, output = TRUE)
```

Arguments

`x`	a matrix or data frame with numeric vectors.
`tri`	a character string indicating which triangular of the correlation matrix to show on the console, i.e., `both` for upper and lower triangular, `lower` (default) for the lower triangular, and `upper` for the upper triangular.
`weighted`	logical: if `TRUE` (default), the weighted pooled standard deviation is used.
`correct`	logical: if `TRUE`, correction factor for Cohen's d to remove positive bias in small samples is used.
`digits`	integer value indicating the number of decimal places digits to be used for displaying correlation coefficients and Cohen's d estimates.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown on the console.

Details

Note that non-numeric variables (i.e., factors, character vectors, and logical vectors) are excluded from to the analysis.

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, matrix or data frame specified in `x` (`data`), specification of function arguments (`args`), and list with results (`result`).

Author(s)

Takuya Yanagida `<takuya.yanagida@univie.ac.at>`

References

Enders, C. K. (2010). *Applied missing data analysis*. Guilford Press.

Graham, J. W. (2009). Missing data analysis: Making it work in the real world. *Annual Review of Psychology, 60*, 549-576. https://doi.org/10.1146/annurev.psych.58.110405.085530

van Buuren, S. (2018). *Flexible imputation of missing data* (2nd ed.). Chapman & Hall.

See Also

`as.na`, `na.as`, `na.coverage`, `na.descript`, `na.indicator`, `na.pattern`, `na.prop`.

Examples

```
dat <- data.frame(x1 = c(1, NA, 2, 5, 3, NA, 5, 2),
                  x2 = c(4, 2, 5, 1, 5, 3, 4, 5),
                  x3 = c(NA, 3, 2, 4, 5, 6, NA, 2),
                  x4 = c(5, 6, 3, NA, NA, 4, 6, NA))

# Auxiliary variables
na.auxiliary(dat)
```

`na.coverage` *Variance-Covariance Coverage*

Description

This function computes the proportion of cases that contributes for the calculation of each variance and covariance.

Usage

```
na.coverage(x, tri = c("both", "lower", "upper"), digits = 2, as.na = NULL,
            check = TRUE, output = TRUE)
```

Arguments

`x`	a matrix or data frame.
`tri`	a character string or character vector indicating which triangular of the matrix to show on the console, i.e., `both` for upper and lower triangular, `lower` (default) for the lower triangular, and `upper` for the upper triangular.
`digits`	an integer value indicating the number of decimal places to be used for displaying proportions.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown on the console.

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, matrix or data frame specified in `x` (`data`), specification of function arguments (`args`), and list with results (`result`).

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Enders, C. K. (2010). *Applied missing data analysis*. Guilford Press.

Graham, J. W. (2009). Missing data analysis: Making it work in the real world. *Annual Review of Psychology, 60*, 549-576. https://doi.org/10.1146/annurev.psych.58.110405.085530

van Buuren, S. (2018). *Flexible imputation of missing data* (2nd ed.). Chapman & Hall.

See Also

`as.na`, `na.as`, `na.auxiliary`, `na.descript`, `na.indicator`, `na.pattern`, `na.prop`.

Examples

```
dat <- data.frame(x = c(1, NA, NA, 6, 3),
                  y = c(7, NA, 8, 9, NA),
                  z = c(2, NA, 3, NA, 5))

# Create missing data indicator matrix R
na.coverage(dat)
```

---

na.descript *Descriptive Statistics for Missing Data*

Description

This function computes descriptive statistics for missing data, e.g. number (of missing values, and summary statistics for the number (

Usage

```
na.descript(x, table = FALSE, digits = 2, as.na = NULL, check = TRUE, output = TRUE)
```

Arguments

`x`	a matrix or data frame.
`table`	logical: if `TRUE`, a frequency table with number of observed values (`"nObs"`), percent of observed values (`"pObs"`), number of missing values (`"nNA"`), and percent of missing values (`"pNA"`) is printed for each variable on the console.
`digits`	an integer value indicating the number of decimal places to be used for displaying percentages.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown on the console.

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, matrix or data frame specified in `x` (`data`), specification of function arguments (`args`), and list with results (`result`).

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Enders, C. K. (2010). *Applied missing data analysis*. Guilford Press.

Graham, J. W. (2009). Missing data analysis: Making it work in the real world. *Annual Review of Psychology, 60*, 549-576. https://doi.org/10.1146/annurev.psych.58.110405.085530

van Buuren, S. (2018). *Flexible imputation of missing data* (2nd ed.). Chapman & Hall.

See Also

`as.na`, `na.as`, `na.auxiliary`, `na.coverage`, `na.indicator`, `na.pattern`, `na.prop`.

Examples

```
dat <- data.frame(x1 = c(1, NA, 2, 5, 3, NA, 5, 2),
                  x2 = c(4, 2, 5, 1, 5, 3, 4, 5),
                  x3 = c(NA, 3, 2, 4, 5, 6, NA, 2),
                  x4 = c(5, 6, 3, NA, NA, 4, 6, NA))

# Descriptive statistics for missing data
na.descript(dat)

# Descriptive statistics for missing data, print results with 3 digits
na.descript(dat, digits = 3)

# Descriptive statistics for missing data, convert value 2 to NA
na.descript(dat, as.na = 2)

# Descriptive statistics for missing data with frequency table
na.descript(dat, table = TRUE)
```

`na.indicator` *Missing Data Indicator Matrix*

Description

This function creates a missing data indicator matrix R that denotes whether values are observed or missing, i.e., $r = 1$ if a value is observed, and $r = 0$ if a value is missing.

Usage

```
na.indicator(x, as.na = NULL, check = TRUE)
```

Arguments

x	a matrix or data frame.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
check	logical: if TRUE, argument specification is checked.

Value

Returns a matrix or data frame with $r = 1$ if a value is observed, and $r = 0$ if a value is missing.

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Enders, C. K. (2010). *Applied missing data analysis*. Guilford Press.

Graham, J. W. (2009). Missing data analysis: Making it work in the real world. *Annual Review of Psychology, 60*, 549-576. https://doi.org/10.1146/annurev.psych.58.110405.085530

van Buuren, S. (2018). *Flexible imputation of missing data* (2nd ed.). Chapman & Hall.

See Also

as.na, na.as, na.auxiliary, na.coverage, na.descript, na.pattern, na.prop.

Examples

```
dat <- data.frame(x = c(1, NA, NA, 6, 3),
                  y = c(7, NA, 8, 9, NA),
                  z = c(2, NA, 3, NA, 5))

# Create missing data indicator matrix \eqn{R}
na.indicator(dat)
```

na.pattern *Missing Data Pattern*

Description

This function computes a summary of missing data patterns, i.e., number (

Usage

```
na.pattern(x, order = FALSE, digits = 2, as.na = NULL, check = TRUE, output = TRUE)
```

Arguments

`x`	a matrix or data frame with incomplete data, where missing values are coded as `NA`.
`order`	logical: if `TRUE`, variables are ordered from left to right in increasing order of missing values.
`digits`	an integer value indicating the number of decimal places to be used for displaying percentages.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown.

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, matrix or data frame specified in `x` (`data`), specification of function arguments (`args`), list with results (`result`), and a vector with the number of missing data pattern for each case (`pattern`),

Author(s)

Takuya Yanagida `<takuya.yanagida@univie.ac.at>`

References

Enders, C. K. (2010). *Applied missing data analysis*. Guilford Press.

Graham, J. W. (2009). Missing data analysis: Making it work in the real world. *Annual Review of Psychology, 60*, 549-576. https://doi.org/10.1146/annurev.psych.58.110405.085530

van Buuren, S. (2018). *Flexible imputation of missing data* (2nd ed.). Chapman & Hall.

See Also

`as.na`, `na.as`, `na.auxiliary`, `na.coverage`, `na.descript`, `na.indicator`, `na.prop`.

Examples

```
dat <- data.frame(x = c(1, NA, NA, 6, 3),
                  y = c(7, NA, 8, 9, NA),
                  z = c(2, NA, 3, NA, 5))

# Compute a summary of missing data patterns
dat.pattern <- na.pattern(dat)

# Vector of missing data pattern for each case
dat.pattern$pattern
# Data frame without cases with missing data pattern 2 and 5
dat[!dat.pattern$pattern %in% c(2, 5), ]
```

`na.prop` *Proportion of Missing Data for Each Case*

Description

This function computes the proportion of missing data for each case in a matrix or data frame.

Usage

```
na.prop(x, digits = 2, as.na = NULL, check = TRUE)
```

Arguments

`x`	a matrix or data frame.
`digits`	an integer value indicating the number of decimal places to be used for displaying proportions.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`check`	logical: if `TRUE`, argument specification is checked.

Value

Returns a numeric vector with the same length as the number of rows in `x` containing the proportion of missing data.

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Enders, C. K. (2010). *Applied missing data analysis*. Guilford Press.

Graham, J. W. (2009). Missing data analysis: Making it work in the real world. *Annual Review of Psychology, 60*, 549-576. https://doi.org/10.1146/annurev.psych.58.110405.085530

van Buuren, S. (2018). *Flexible imputation of missing data* (2nd ed.). Chapman & Hall.

See Also

as.na, na.as, na.auxiliary, na.coverage, na.descript, na.indicator, na.pattern.

Examples

```
dat <- data.frame(x = c(1, NA, NA, 6, 3),
                  y = c(7, NA, 8, 9, NA),
                  z = c(2, NA, 3, NA, 5))

# Compute proportion of missing data (\code{NA}) for each case in the data frame
na.prop(dat)
```

---

omega.coef *Coefficient Omega, Hierarchical Omega, and Categorical Omega*

Description

This function computes point estimate and confidence interval for the coefficient omega (McDonald, 1978), hierarchical omega (Kelley & Pornprasertmanit, 2016), and categorical omega (Green & Yang, 2009) along with standardized factor loadings and omega if item deleted.

Usage

```
omega.coef(x, resid.cov = NULL, type = c("omega", "hierarch", "categ"),
           exclude = NULL, std = FALSE, na.omit = FALSE,
           print = c("all", "omega", "item"), digits = 2,
           conf.level = 0.95, as.na = NULL, check = TRUE, output = TRUE
)
```

Arguments

`x`	a matrix or data frame. Note that at least three items are needed for computing omega.
`resid.cov`	a character vector or a list of character vectors for specifying residual covariances when computing coefficient omega, e.g. `resid.cov = c("x1","x2")` for specifying a residual covariance between items `x1` and `x2` or `resid.cov = list(c("x1","x2"),c("x3","x4"))` for specifying residual covariances between items between items `x1` and `x2`, and items `x3` and `x4`.
`type`	a character string indicating the type of omega to be computed, i.e., `omega` (default) for coefficient omega, `hierarch` for hierarchical omega, and `categ` for categorical omega.
`exclude`	a character vector indicating items to be excluded from the analysis.
`std`	logical: if `TRUE`, the standardized coefficient omega is computed.
`na.omit`	logical: if `TRUE`, incomplete cases are removed before conducting the analysis (i.e., listwise deletion); if `FALSE`, full information maximum likelihood (FIML) is used for computing coefficient omega or hierarchical omega, while pairwise deletion is used for computing categorical omega.

print	a character vector indicating which results to show, i.e. "all" (default), for all results "omega" for omega, and "item" for item statistics.
digits	an integer value indicating the number of decimal places to be used for displaying omega and standardized factor loadings.
conf.level	a numeric value between 0 and 1 indicating the confidence level of the interval.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
check	logical: if TRUE, argument specification is checked.
output	logical: if TRUE, output is shown.

Details

Omega is computed by estimating a confirmatory factor analysis model using the `cfa()` function in the **lavaan** package by Yves Rosseel (2019). Maximum likelihood ("ML") estimator is used for computing coefficient omega and hierarchical omega, while diagonally weighted least squares estimator ("DWLS") is used for computing categorical omega.

Note that the computation of the hierarchical and categorical omega is based on the `ci.reliability()` function in the **MBESS** package by Ken Kelley (2019).

Approximate confidence intervals are computed using the procedure by Feldt, Woodruff and Salih (1987). Note that there are at least 10 other procedures for computing the confidence interval (see Kelley and Pornprasertmanit, 2016), which are implemented in the `ci.reliability()` function in the **MBESSS** package by Ken Kelley (2019).

Value

Returns an object of class `mistsy.object`, which is a list with following entries: function call (`call`), type of analysis `type`, matrix or data frame specified in `x` (`data`), specification of function arguments (`args`), fitted lavaan object (`mod.fit`), and list with results (`result`).

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Feldt, L. S., Woodruff, D. J., & Salih, F. A. (1987). Statistical inference for coefficient alpha. *Applied Psychological Measurement*, 11 93-103.

Green, S. B., & Yang, Y. (2009). Reliability of summed item scores using structural equation modeling: An alternative to coefficient alpha. *Psychometrika, 74*, 155-167. https://doi.org/10.1007/s11336-008-9099-3

Kelley, K., & Pornprasertmanit, S. (2016). Confidence intervals for population reliability coefficients: Evaluation of methods, recommendations, and software for composite measures. *Psychological Methods, 21*, 69-92. http://dx.doi.org/10.1037/a0040086

Ken Kelley (2019). *MBESS: The MBESS R Package*. R package version 4.6.0. https://CRAN.R-project.org/package=MBESS

McDonald, R. P. (1978). Generalizability in factorable domains: "Domain validity and generalizability" *Educational and Psychological Measurement, 38*, 75-79.

See Also

omega.coef, reverse.item, scores

Examples

```
dat <- data.frame(item1 = c(5, 2, 3, 4, 1, 2, 4, 2),
                  item2 = c(5, 3, 3, 5, 2, 2, 5, 1),
                  item3 = c(4, 2, 4, 5, 1, 3, 5, 1),
                  item4 = c(5, 1, 2, 5, 2, 3, 4, 2))

# Compute unstandardized coefficient omega and item statistics
omega.coef(dat)

# Compute unstandardized coefficient omega with a residual covariance
# and item statistics
omega.coef(dat, resid.cov = c("item1", "item2"))

# Compute unstandardized coefficient omega with residual covariances
# and item statistics
omega.coef(dat, resid.cov = list(c("item1", "item2"), c("item3", "item4")))

# Compute unstandardized hierarchical omega and item statistics
omega.coef(dat, type = "hierarch")

# Compute categorical omega and item statistics
omega.coef(dat, type = "categ")

# Compute standardized coefficient omega and item statistics
omega.coef(dat, std = TRUE)

# Compute unstandardized coefficient omega
omega.coef(dat, print = "omega")

# Compute item statistics
omega.coef(dat, print = "item")

# Compute unstandardized coefficient omega and item statistics while excluding item3
omega.coef(dat, exclude = "item3")

# Summary of the CFA model used to compute coefficient omega
lavaan::summary(omega.coef(dat, output = FALSE)$mod.fit,
                fit.measures = TRUE, standardized = TRUE)
```

---

phi.coef *Phi Coefficient*

Description

This function computes the (adjusted) Phi coefficient between two or more than two dichotomous variables.

Usage

```
phi.coef(x, adjust = FALSE, tri = c("both", "lower", "upper"), digits = 2,
         as.na = NULL, check = TRUE, output = TRUE)
```

Arguments

`x`	a matrix or data frame.
`adjust`	logical: if `TRUE`, phi coefficient is adjusted by relating the coefficient to the possible maximum.
`tri`	a character string or character vector indicating which triangular of the matrix to show on the console, i.e., `both` for upper and lower triangular, `lower` (default) for the lower triangular, and `upper` for the upper triangular.
`digits`	an integer value indicating the number of decimal places digits to be used for displaying phi coefficients.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown on the console.

Details

The maximum Phi coefficient is determined by the distribution of the two variables, i.e., the Phi coefficient cannot achieve the value of 1 in many cases. According to Cureton (1959), the' phi coefficient can be adjusted by relating the coefficient to the possible maximum, ϕ/ϕ_max.

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, matrix or data frame specified in `x` (`data`), specification of function arguments (`args`), and list with results (`result`).

Author(s)

Takuya Yanagida `<takuya.yanagida@univie.ac.at>`

References

Cureton, E. E. (1959). Note on Phi/Phi max. *Psychometrika, 24*, 89-91.

Davenport, E. C., & El-Sanhurry, N. A. (1991). Phi/Phimax: Review and synthesis. *Educational and Psychological Measurement, 51*, 821-828. https://doi.org/10.1177/001316449105100403

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. New York: John Wiley & Sons.

Examples

```
dat <- data.frame(x1 = c(0, 1, 0, 1, 0, 1, 0, 1, 1, 0),
                  x2 = c(0, 1, 0, 0, 1, 1, 1, 1, 1, 1),
                  x3 = c(0, 1, 0, 1, 1, 1, 1, 1, 0, 0))

# Phi coefficient between x1 and x2
phi.coef(dat[, c("x1", "x2")])

# Adjusted phi coefficient between x1 and x2
phi.coef(dat[, c("x1", "x2")], adjust = TRUE)

# Phi coefficient matrix between x1, x2, and x3
phi.coef(dat)

# Adjusted phi coefficient matrix between x1, x2, and x3
phi.coef(dat, adjust = TRUE)
```

`poly.cor` *Polychoric Correlation Matrix*

Description

This function computes a polychoric correlation matrix, which is the estimated Pearson product-moment correlation matrix between underlying normally distributed latent variables which generate the ordinal scores.

Usage

```
poly.cor(x, smooth = TRUE, global = TRUE, weight = NULL, correct = 0, progress = TRUE,
         na.rm = TRUE, delete = TRUE, tri = c("both", "lower", "upper"),
         digits = 2, as.na = NULL, check = TRUE, output = TRUE)
```

Arguments

`x`	a matrix or data frame of discrete values.
`smooth`	logical: if `TRUE` and if the polychoric matrix is not positive definite, a simple smoothing algorithm using `cor.smooth()` function is applied.
`global`	logical: if `TRUE`, the global values of the tau parameter is used instead of the local values.
`weight`	a vector of length of the number of observations that specifies the weights to apply to each case. The `NULL` case is equivalent of weights of 1 for all cases.
`correct`	a numeric value indicating the correction value to use to correct for continuity in the case of zero entry. Note that unlike in the `polychoric()` function in the **psych** package the default value is 0.
`progress`	logical: if `TRUE`, the progress bar is shown.
`na.rm`	logical: if `TRUE`, missing data are deleted.

`delete`	logical: if `TRUE`, cases with no variance are deleted with a warning before proceeding.
`tri`	a character string indicating which triangular of the matrix to show on the console, i.e., `both` for upper and lower triangular, `lower` (default) for the lower triangular, and `upper` for the upper triangular.
`digits`	an integer value indicating the number of decimal places to be used for displaying correlation coefficients.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown on the console.

Details

Note that this function is based on the `polychoric()` function in the **psych** by William Revelle.

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, matrix or data frame specified in `x` (`data`), specification of function arguments (`args`), and list with results (`result`).

Author(s)

William Revelle

References

Revelle, W. (2018) *psych: Procedures for personality and psychological research*. Northwestern University, Evanston, Illinois, USA, https://CRAN.R-project.org/package=psych Version = 1.8.12.

Examples

```
dat <- data.frame(x1 = c(1, 1, 3, 2, 1, 2, 3, 2, 3, 1),
                  x2 = c(1, 2, 1, 1, 2, 2, 2, 1, 3, 1),
                  x3 = c(1, 3, 2, 3, 3, 1, 3, 2, 1, 2))

# Polychoric correlation matrix
poly.cor(dat)
```

`print.misty.object` *Print misty.object object*

Description

This function prints the `misty.object` object

Usage

```
## S3 method for class 'misty.object'
print(x, print = x$args$print, tri = x$args$tri,
      freq = x$args$freq, split = x$args$split, table = x$args$table,
      digits = x$args$digits, p.digits = x$args$p.digits,
      icc.digits = x$args$icc.digits, sort.var = x$args$sort.var,
      order = x$args$order, check = TRUE, ...)
```

Arguments

`x`	`misty.object` object.
`print`	a character string or character vector indicating which results to to be printed on the console.
`tri`	a character string or character vector indicating which triangular of the matrix to show on the console, i.e., `both` for upper and lower triangular, `lower` for the lower triangular, and `upper` for the upper triangular.
`freq`	logical: if `TRUE`, absolute frequencies will be included in the cross tabulation (`freq()` function).
`split`	logical: if `TRUE`, cross table is split in absolute frequencies and percentage(s) (`crosstab()` function).
`table`	logical: if `TRUE`, a frequency table with number of observed values (`"nObs"`), percent of observed values (`"pObs"`), number of missing values (`"nNA"`), and percent of missing values (`"pNA"`) is printed for each variable on the console (`na.descript()` function).
`digits`	an integer value indicating the number of decimal places digits to be used for displaying results.
`p.digits`	an integer indicating the number of decimal places to be used for displaying *p*-values.
`icc.digits`	an integer indicating the number of decimal places to be used for displaying intraclass correlation coefficients (`multilevel.descript()` function).
`sort.var`	logical: if `TRUE`, output is sorted by variables.
`order`	logical: if `TRUE`, variables are ordered from left to right in increasing order of missing values (`na.descript()` function).
`check`	logical: if `TRUE`, argument specification is checked.
`...`	further arguments passed to or from other methods.

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

See Also

alpha.coef, ci.mean.diff, ci.mean, ci.median, ci.prop.diff, ci.prop, ci.sd, ci.var, cohens.d, collin.diag, cont.coef, cor.matrix, cramers.v, crosstab, descript, eta.sq, freq, levenes.test, multilevel.descript, na.auxiliary, na.coverage, na.descript, na.pattern, omega.coef, phi.coef, poly.cor, size.cor, size.mean, size.prop, z.test,

---

read.mplus *Read Mplus Data File and Variable Names*

Description

This function reads a Mplus data file and/or Mplus input/output file to return a data frame with variable names extracted from the Mplus input/output file.

Usage

```
read.mplus(file, sep = "", input = NULL, print = FALSE, return.var = FALSE,
           fileEncoding = "UTF-8-BOM", check = TRUE)
```

Arguments

file	a character string indicating the name of the Mplus data file with or without the file extension .dat, e.g., "Mplus_Data.dat" or "Mplus_Data". Note that it is not necessary to specify this argument when return.var = TRUE.
sep	a character string indicating the field separator (i.e., delimiter) used in the data file specified in file. By default, the separator is 'white space', i.e., one or more spaces, tabs, newlines or carriage returns.
input	a character string indicating the Mplus input (.inp) or output file (.out) in which the variable names are specified in the VARIABLE: section. Note that if input = NULL, this function is equivalent to read.table(file).
print	logical: if TRUE, variable names are printed on the console.
return.var	logical: if TRUE, the function returns the variable names extracted from the Mplus input or output file only.
fileEncoding	character string declaring the encoding used on file so the character data can be re-encoded. See df.sort.
check	logical: if TRUE, argument specification is checked.

Value

A data frame containing a representation of the data in the file.

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Muthen, L. K., & Muthen, B. O. (1998-2017). *Mplus User's Guide* (8th ed.). Muthen & Muthen.

See Also

run.mplus, write.mplus

Examples

```
## Not run:
# Read Mplus data file and variable names extracted from the Mplus input file
dat <- read.mplus("Mplus_Data.dat", input = "Mplus_Input.inp")

# Read Mplus data file and variable names extracted from the Mplus input file,
# print variable names on the console
dat <- read.mplus("Mplus_Data.dat", input = "Mplus_Input.inp", print = TRUE)

# Read variable names extracted from the Mplus input file
varnames <- read.mplus(input = "Mplus_Input.inp", return.var = TRUE)

## End(Not run)
```

---

read.sav — *Read SPSS File*

Description

This function calls the `read_spss` function in the **haven** package by Hadley Wickham and Evan Miller (2019) to read an SPSS file.

Usage

```
read.sav(file, use.value.labels = FALSE, use.missings = TRUE, as.data.frame = TRUE,
         check = TRUE)
```

Arguments

`file`	a character string indicating the name of the SPSS data file with or without file extension '.sav', e.g., `"My_SPSS_Data.sav"` or `"My_SPSS_Data"`.
`use.value.labels`	logical: if `TRUE`, variables with value labels are converted into factors.
`use.missings`	logical: if `TRUE` (default), user-defined missing values are converted into NAs.
`as.data.frame`	logical: if `TRUE` (default), function returns a data frame (default); if `FALSE` function returns a tibble.
`check`	logical: if `TRUE`, argument specification is checked.

Value

Returns a data frame or tibble.

Author(s)

Hadley Wickham and Evan Miller

References

Hadley Wickham and Evan Miller (2019). *haven: Import and Export 'SPSS', 'Stata' and 'SAS' Files*. R package version 2.1.1.https://CRAN.R-project.org/package=haven

See Also

write.sav

Examples

```
## Not run:
# Read SPSS data
read.sav("SPSS_Data.sav")
read.sav("SPSS_Data")

# Read SPSS data, convert variables with value labels into factors
read.sav("SPSS_Data.sav", use.value.labels = TRUE)

# Read SPSS data, user-defined missing values are not converted into NAs
read.sav("SPSS_Data.sav", use.missing = FALSE)

# Read SPSS data as tibble
read.sav("SPSS_Data.sav", as.data.frame = FALSE)

## End(Not run)
```

---

read.xlsx *Read Excel File*

Description

This function calls the `read_xlsx()` function in the **readxl** package by Hadley Wickham and Jennifer Bryan (2019) to read an Excel file (.xlsx).

Usage

```
read.xlsx(file, sheet = NULL, header = TRUE, range = NULL,
       coltypes = c("skip", "guess", "logical", "numeric", "date", "text", "list"),
          na = "", trim = TRUE, skip = 0, nmax = Inf, guessmax = min(1000, nmax),
           progress = readxl::readxl_progress(), name.repair = "unique",
           as.data.frame = TRUE, check = TRUE)
```

Arguments

`file`	a character string indicating the name of the Excel data file with or without file extension '.xlsx', e.g., `"My_Excel_Data.xlsx"` or `"My_Excel_Data"`.
`sheet`	a character string indicating the name of a sheet or a numeric value indicating the position of the sheet to read. By default the first sheet will be read.
`header`	logical: if `TRUE` (default), the first row is used as column names, if `FALSE` default names are used. A character vector giving a name for each column can also be used. If `coltypes` as a vector is provided, `colnames` can have one entry per column, i.e. have the same length as `coltypes`, or one entry per unskipped column.
`range`	a character string indicating the cell range to read from, e.g. typical Excel ranges like `"B3:D87"`, possibly including the sheet name like `"Data!B2:G14"`. Interpreted strictly, even if the range forces the inclusion of leading or trailing empty rows or columns. Takes precedence over `skip`, `nmax` and `sheet`.
`coltypes`	a character vector containing one entry per column from these options `"skip"`, `"guess"`, `"logical"`, `"numeric"`, `"date"`, `"text"` or `"list"`. If exactly one `coltype` is specified, it will be recycled. By default (i.e., `coltypes = NULL`) coltypes will be guessed. The content of a cell in a skipped column is never read and that column will not appear in the data frame output. A list cell loads a column as a list of length 1 vectors, which are typed using the type guessing logic from coltypes = NULL, but on a cell-by-cell basis.
`na`	a character vector indicating strings to interpret as missing values. By default, blank cells will be treated as missing data.
`trim`	logical: if `TRUE` (default), leading and trailing whitespace will be trimmed
`skip`	a numeric value indicating the minimum number of rows to skip before reading anything, be it column names or data. Leading empty rows are automatically skipped, so this is a lower bound. Ignored if the argument `range` is specified.
`nmax`	a numeric value indicating the maximum number of data rows to read. Trailing empty rows are automatically skipped, so this is an upper bound on the number of rows in the returned data frame. Ignored if the argument `range` is specified.
`guessmax`	a numeric value indicating the maximum number of data rows to use for guessing column types.
`progress`	display a progress spinner? By default, the spinner appears only in an interactive session, outside the context of knitting a document, and when the call is likely to run for several seconds or more.
`name.repair`	a character string indicating the handling of column names. By default, the function ensures column names are not empty and are unique.
`as.data.frame`	logical: if `TRUE` (default), function returns a regular data frame (default); if `FALSE` function returns a tibble.
`check`	logical: if `TRUE`, argument specification is checked.

Value

Returns a data frame or tibble.

Author(s)

Hadley Wickham and Jennifer Bryan

See Also

read.sav, read.mplus

Examples

```
## Not run:
# Read Excel file (.xlsx)
read.xlsx("data.xlsx")

# Read Excel file (.xlsx), use default names as column names
read.xlsx("data.xlsx", header = FALSE)

# Read Excel file (.xlsx), interpret -99 as missing values
read.xlsx("data.xlsx", na = "-99")

# Read Excel file (.xlsx), use x1, x2, and x3 as column names
read.xlsx("data.xlsx", header = c("x1", "x2", "x3"))

# Read Excel file (.xlsx), read cells A1:B5
read.xlsx("data.xlsx", range = "A1:B5")

# Read Excel file (.xlsx), skip 2 rows before reading data
read.xlsx("data.xlsx", skip = 2)

# Read Excel file (.xlsx), returns a tibble
read.xlsx("data.xlsx", as.data.frame = FALSE)

## End(Not run)
```

---

rec *Recode Variable*

Description

This function recodes a numeric vector, character vector, or factor according to recode specifications.

Usage

```
rec(x, spec, as.factor = FALSE, levels = NULL, as.na = NULL, table = FALSE,
    check = TRUE)
```

Arguments

`x`	a numeric vector, character vector or factor.
`spec`	a character string of recode specifications (see 'Details').
`as.factor`	logical: if `TRUE`, character vector will be coerced to a factor.
`levels`	a character vector for specifying the levels in the returned factor.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`table`	logical: if `TRUE`, a cross table variable x recoded variable is printed on the console.
`check`	logical: if `TRUE`, argument specification is checked.

Details

Recode specifications appear in a character string, separated by semicolons (see the examples below), of the form input = output. If an input value satisfies more than one specification, then the first (from left to right) applies. If no specification is satisfied, then the input value is carried over to the result. `NA` is allowed in input and output. Several recode specifications are supported:

- single value For example, 0 = NA

- vector of values For example, c(7, 8, 9) = 'high'

- range of values For example, 7:9 = 'C'. The special values lo (lowest value) and hi (highest value) may appear in a range. For example, `lo:10 = 1`. Note that `:` is not the R sequence operator. In addition you may not use `:` with the collect operator, e.g., `c(1,3,5:7)` will cause an error.

- else For example, `else = NA`. Everything that does not fit a previous specification. Note that `else` matches all otherwise unspecified values on input, including `NA`.

Note that the function was adapted from the `recode()` function in the **car** package by John Fox and Sanford Weisberg (2019).

Value

Returns a numeric vector with the same length as `x` containing the recoded variable.

Author(s)

Takuya Yanagida `<takuya.yanagida@univie.ac.at>`

References

Fox, J., & Weisberg S. (2019). *An R Companion to Applied Regression* (3rd ed.). Thousand Oaks CA: Sage. URL: https://socialsciences.mcmaster.ca/jfox/Books/Companion/

See Also

`reverse.item`

Examples

```
#--------------------------------------
# Numeric vector
x.num <- c(1, 2, 4, 5, 6, 8, 12, 15, 19, 20)

# Recode 5 = 50 and 19 = 190
rec(x.num, "5 = 50; 19 = 190")

# Recode 1, 2, and 5 = 100 and 4, 6, and 7 = 200 and else = 300
rec(x.num, "c(1, 2, 5) = 100; c(4, 6, 7) = 200; else = 300")

# Recode lowest value to 10 = 100 and 11 to highest value = 200
rec(x.num, "lo:10 = 100; 11:hi = 200")

# Recode 5 = 50 and 19 = 190 and check recoding
rec(x.num, "5 = 50; 19 = 190", table = TRUE)

#--------------------------------------
# Character vector
x.chr <- c("a", "c", "f", "j", "k")

# Recode a to x
rec(x.chr, "'a' = 'X'")

# Recode a and f to x, c and j to y, and else to z
rec(x.chr, "c('a', 'f') = 'x'; c('c', 'j') = 'y'; else = 'z'")

# Recode a to x and coerce to a factor
rec(x.chr, "'a' = 'X'", as.factor = TRUE)

#--------------------------------------
# Factor
x.factor <- factor(c("a", "b", "a", "c", "d", "d", "b", "b", "a"))

# Recode a to x, factor levels ordered alphabetically
rec(x.factor, "'a' = 'x'")

# Recode a to x, user-defined factor levels
rec(x.factor, "'a' = 'x'", levels = c("x", "b", "c", "d"))
```

`reverse.item` *Reverse Code Scale Item*

Description

This function reverse codes an inverted item, i.e., item that is negatively worded.

Usage

```
reverse.item(x, min = NULL, max = NULL, keep = NULL, as.na = NULL, table = FALSE,
             check = TRUE)
```

Arguments

`x`	a numeric vector with integer values.
`min`	an integer indicating the minimum of the item (i.e., lowest possible scale value).
`max`	an integer indicating the maximum of the item (i.e., highest possible scale value).
`keep`	a numeric vector indicating values not to be reverse coded.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`table`	logical: if `TRUE`, a cross table item x reverse coded item is printed on the console.
`check`	logical: if `TRUE`, argument specification is checked.

Details

If arguments `min` and/or `max` are not specified, empirical minimum and/or maximum is computed from the vector. Note, however, that reverse coding might fail if the lowest or highest possible scale value is not represented in the vector. That is, it is always preferable to specify the arguments `min` and `max`.

Value

Returns a numeric vector with the same length as `x` containing the reverse coded scale item.

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. New York: John Wiley & Sons.

See Also

`alpha.coef`, `rec`, `scores`

Examples

```
dat <- data.frame(item1 = c(5, 2, 3, 4, 1, 2, 4, 2),
                  item2 = c(1, 5, 3, 1, 4, 4, 1, 5),
                  item3 = c(4, 2, 4, 5, 1, 3, 5, -99))

# Reverse code item2
reverse.item(dat$item2, min = 1, max = 5)

# Reverse code item3 while keeping the value -99
reverse.item(dat$item3, min = 1, max = 5, keep = -99)

# Reverse code item3 while keeping the value -99 and check recoding
dat$item3r <- reverse.item(dat$item3, min = 1, max = 5, keep = -99, table = TRUE)
```

run.mplus *Run Mplus Models*

Description

This function runs a group of Mplus models (.inp files) located within a single directory or nested within subdirectories.

Usage

```
run.mplus(target = getwd(), recursive = FALSE, filefilter = NULL, showOutput = FALSE,
         replaceOutfile = c("always", "never", "modifiedDate"), logFile = NULL,
          Mplus = "Mplus", killOnFail = TRUE, local_tmpdir = FALSE)
```

Arguments

target	a character string indicating the directory containing Mplus input files (.inp) to run or the single .inp file to be run. May be a full path, relative path, or a filename within the working directory.
recursive	logical: if TRUE, run all models nested in subdirectories within directory. Not relevant if target is a single file.
filefilter	a Perl regular expression (PCRE-compatible) specifying particular input files to be run within directory. See regex or http://www.pcre.org/pcre.txt for details about regular expression syntax. Not relevant if target is a single file.
showOutput	logical: if TRUE, estimation output (TECH8) is show on the R console. Note that if run within Rgui, output will display within R, but if run via Rterm, a separate window will appear during estimation.
replaceOutfile	a character string for specifying three settings: "always" (default), which runs all models, regardless of whether an output file for the model exists, "never", which does not run any model that has an existing output file, and "modifiedDate", which only runs a model if the modified date for the input file is more recent than the output file modified date.
logFile	a character string specifying a file that records the settings passed into the function and the models run (or skipped) during the run.
Mplus	a character string for specifying the name or path of the Mplus executable to be used for running models. This covers situations where Mplus is not in the system's path, or where one wants to test different versions of the Mplus program.Note that there is no need to specify this argument for most users since it has intelligent defaults.
killOnFail	logical: if TRUE, all processes named mplus.exe when mplus.run() does not terminate normally are killed. Windows only.
local_tmpdir	logical: if TRUE, the TMPDIR environment variable is set to the location of the .inp file prior to execution. This is useful in Monte Carlo studies where many instances of Mplus may run in parallel and we wish to avoid collisions in temporary files among processes. Linux/Mac only.

Details

Note that this function is a copy of the `runModels()` function in the **MplusAutomation** package by Michael Hallquist.

Value

None.

Author(s)

Michael Hallquist

References

Hallquist, M. N. & Wiley, J. F. (2018). MplusAutomation: An R package for facilitating large-scale latent variable analyses in Mplus. *Structural Equation Modeling: A Multidisciplinary Journal, 25*, 621-638. https://doi.org/10.1080/10705511.2017.1402334.

Muthen, L. K., & Muthen, B. O. (1998-2017). *Mplus User's Guide* (8th ed.). Muthen & Muthen.

Examples

```
## Not run:
# Run Mplus models located within a single directory
run.mplus(Mplus = "C:/Program Files/Mplus/Mplus.exe")

# Run Mplus models located nested within subdirectories
run.mplus(recursive = TRUE,
          Mplus = "C:/Program Files/Mplus/Mplus.exe")

## End(Not run)
```

`rwg.lindell` *Lindell, Brandt and Whitney (1999) r*wg(j) Within-Group Agreement Index for Multi-Item Scales*

Description

This function computes r*wg(j) within-group agreement index for multi-item scales as described in Lindell, Brandt and Whitney (1999).

Usage

```
rwg.lindell(x, group, A = NULL, ranvar = NULL, z = TRUE, expand = TRUE, na.omit = FALSE,
             as.na = NULL, check = TRUE)
```

Arguments

`x`	a matrix or data frame with numeric vectors.
`group`	a vector representing the grouping structure (i.e., group variable).
`A`	a numeric value indicating the number of discrete response options of the items from which the random variance is computed based on $(A^2 - 1)/12$. Note that either the argument `j` or the argument`ranvar` is specified.
`ranvar`	a numeric value indicating the random variance to which the mean of the item variance is divided. Note that either the argument `j` or the argument`ranvar` is specified.
`z`	logical: if `TRUE`, Fisher z-transformation based on the formula $z = 0.5*log((1+r)/(1-r))$ is applied to the vector of r*wg(j) estimates.
`expand`	logical: if `TRUE`, vector of r*wg(j) estimates is expanded to match the input vector `x`.
`na.omit`	logical: if `TRUE`, incomplete cases are removed before conducting the analysis (i.e., listwise deletion).
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis. Note that `as.na()` function is only applied to `x`, but not to `group`.
`check`	logical: if `TRUE`, argument specification is checked.

Details

The r*wg(j) index is calculated by dividing the mean of the item variance by the expected random variance (i.e., null distribution). The default null distribution in most research is the rectangular or uniform distribution calculated with $\sigma_e^2 u = (A^2 - 1)/12$, where A is the number of discrete response options of the items. However, what constitutes a reasonable standard for random variance is highly debated. Note that the r*wg(j) allows that the mean of the item variances to be larger than the expected random variances, i.e., r*wg(j) values can be negative.

Note that the `rwg.j.lindell()` function in the **multilevel** package uses listwise deletion by default, while the `rwg.lindell()` function uses all available information to compute the r*wg(j) agreement index by default. In order to obtain equivalent results in the presence of missing values, listwise deletion (`na.omit = TRUE`) needs to be applied.

Examples for the application of r*wg(j) within-group agreement index for multi-item scales can be found in Bardach, Yanagida, Schober and Lueftenegger (2018), Bardach, Lueftenegger, Yanagida, Schober and Spiel (2018), and Bardach, Lueftenegger, Yanagida, Spiel and Schober (2019).

Value

Returns a numeric vector containing r*wg(j) agreement index for multi-item scales with the same length as `group` if `expand = TRUE` or a data frame with following entries if `expand = FALSE`:

`group`	group identifier
`n`	group size `x`
`rwg.lindell`	r*wg(j) estimate for each group
`z.rwg.lindell`	Fisher z-transformed r*wg(j) estimate for each group

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Bardach, L., Lueftenegger, M., Yanagida, T., & Schober, B. (2019). Achievement or agreement - Which comes first? Clarifying the temporal ordering of achievement and within-class consensus on classroom goal structures. *Learning and Instruction, 61*, 72-83. https://doi.org/10.1016/j.learninstruc.2019.01.003

Bardach, L., Lueftenegger, M., Yanagida, T., Schober, B. & Spiel, C. (2019). The role of within-class consensus on mastery goal structures in predicting socio-emotional outcomes. *British Journal of Educational Psychology, 89*, 239-258. https://doi.org/10.1111/bjep.12237

Bardach, L., Yanagida, T., Schober, B. & Lueftenegger, M. (2018). Within-class consensus on classroom goal structures: Relations to achievement and achievement goals in mathematics and language classes. *Learning and Individual Differences, 67*, 78-90. https://doi.org/10.1016/j.lindif.2018.07.002

Lindell, M. K., Brandt, C. J., & Whitney, D. J. (1999). A revised index of interrater agreement for multi-item ratings of a single target. *Applied Psychological Measurement*, *23*, 127-135. https://doi.org/10.1177/01466219922031257

O'Neill, T. A. (2017). An overview of interrater agreement on Likert scales for researchers and practitioners. *Frontiers in Psychology*, *8*, Article 777. https://doi.org/10.3389/fpsyg.2017.00777

See Also

group.scores

Examples

```
dat <- data.frame(id = c(1, 2, 3, 4, 5, 6, 7, 8, 9),
                  group = c(1, 1, 1, 2, 2, 2, 3, 3, 3),
                  x1 = c(2, 3, 2, 1, 1, 2, 4, 3, 5),
                  x2 = c(3, 2, 2, 1, 2, 1, 3, 2, 5),
                  x3 = c(3, 1, 1, 2, 3, 3, 5, 5, 4))

# Compute Fisher z-transformed r*wg(j) for a multi-item scale with A = 5 response options
rwg.lindell(dat[, c("x1", "x2", "x3")], group = dat$group, A = 5)

# Compute Fisher z-transformed r*wg(j) for a multi-item scale with a random variance of 2
rwg.lindell(dat[, c("x1", "x2", "x3")], group = dat$group, ranvar = 2)

# Compute r*wg(j) for a multi-item scale with A = 5 response options
rwg.lindell(dat[, c("x1", "x2", "x3")], group = dat$group, A = 5, z = FALSE)

# Compute Fisher z-transformed r*wg(j) for a multi-item scale with A = 5 response options,
# do not expand the vector
rwg.lindell(dat[, c("x1", "x2", "x3")], group = dat$group, A = 5, expand = FALSE)
```

`scores` *Compute Scale Scores*

Description

This function computes (prorated) scale scores by averaging the (available) items that measure a single construct by default.

Usage

```
scores(x, fun = c("mean", "sum", "median", "var", "sd", "min", "max"),
       prorated = TRUE, p.avail = NULL, n.avail = NULL, as.na = NULL,
       check = TRUE)
```

Arguments

`x`	a matrix or data frame with numeric vectors.
`fun`	a character string indicating the function used to compute scale scores, default: `"mean"`.
`prorated`	logical: if `TRUE` (default), prorated scale scores are computed (see 'Details'); if `FALSE`, scale scores of only complete cases are computed.
`p.avail`	a numeric value indicating the minimum proportion of available item responses needed for computing a prorated scale score for each case, e.g. `p.avail = 0.8` indicates that scale scores are only computed for cases with at least 80% of item responses available. By default prorated scale scores are computed for all cases with at least one item response. Note that either argument `p.avail` or `n.avail` is used to specify the proration criterion.
`n.avail`	an integer indicating the minimum number of available item responses needed for computing a prorated scale score for each case, e.g. `n.avail = 2` indicates that scale scores are only computed for cases with item responses on at least 2 items. By default prorated scale scores are computed for all cases with at least one item response. Note that either argument `p.avail` or `n.avail` is used to specify the proration criterion.
`as.na`	a numeric vector indicating user-defined missing values, i.e. these values are converted to `NA` before conducting the analysis.
`check`	logical: if `TRUE`, argument specification is checked.

Details

Prorated mean scale scores are computed by averaging the available items, e.g., if a participant answers 4 out of 8 items, the prorated scale score is the average of the 4 responses. Averaging the available items is equivalent to substituting the mean of a participant's own observed items for each of the participant's missing items, i.e., *person mean imputation* (Mazza, Enders & Ruehlman, 2015) or *ipsative mean imputation* (Schafer & Graham, 2002).

Proration may be reasonable when (1) a relatively high proportion of the items (e.g., 0.8) and never fewer than half are used to form the scale score, (2) means of the items comprising a scale are similar and (3) the item-total correlations are similar (Enders, 2010; Graham, 2009; Graham, 2012). Results of simulation studies indicate that proration is prone to substantial bias when either the item means or the inter-item correlation vary (Lee, Bartholow, McCarthy, Pederson & Sher, 2014; Mazza et al., 2015).

Value

Returns a numeric vector with the same length as `nrow(x)` containing (prorated) scale scores.

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Enders, C. K. (2010). *Applied missing data analysis*. New York, NY: Guilford Press.

Graham, J. W. (2009). Missing data analysis: Making it work in the real world. *Annual Review of Psychology, 60*, 549-576. https://doi.org/10.1146/annurev.psych.58.110405.085530

Graham, J. W. (2012). Missing data: Analysis and design. New York, NY: Springer

Lee, M. R., Bartholow, B. D., McCarhy, D. M., Pederson, S. L., & Sher, K. J. (2014). Two alternative approaches to conventional person-mean imputation scoring of the self-rating of the effects of alcohol scale (SRE). *Psychology of Addictive Behaviors, 29*, 231-236. https://doi.org/10.1037/adb0000015

Mazza, G. L., Enders, C. G., & Ruehlman, L. S. (2015). Addressing item-level missing data: A comparison of proration and full information maximum likelihood estimation. *Multivariate Behavioral Research, 50*, 504-519. https://doi.org/10.1080/00273171.2015.1068157

Schafer, J. L., & Graham, J. W. (2002). Missing data: Our view of the state of the art. *Psychological Methods, 7*, 147-177. https://doi.org/10.1037/1082-989X.7.2.147

See Also

`group.scores`, `alpha.coef`

Examples

```
dat <- data.frame(item1 = c(3,  2,  4, 1,  5, 1,  3, NA),
                  item2 = c(2,  2, NA, 2,  4, 2, NA,  1),
                  item3 = c(1,  1,  2, 2,  4, 3, NA, NA),
                  item4 = c(4,  2,  4, 4, NA, 2, NA, NA),
                  item5 = c(3, NA, NA, 2,  4, 3, NA,  3))

# Prorated mean scale scores
scores(dat)

# Prorated standard deviation scale scores
scores(dat, fun = "sd")

# Sum scale scores without proration
```

```
scores(dat, fun = "sum", prorated = FALSE)

# Prorated mean scale scores,
# minimum proportion of available item responses = 0.8
scores(dat, p.avail = 0.8)

# Prorated mean scale scores,
# minimum number of available item responses = 3
scores(dat, n.avail = 3)
```

`size.cor` *Sample Size Determination for Testing Pearson's Correlation Coefficient*

Description

This function performs sample size computation for testing Pearson's product-moment correlation coefficient based on precision requirements (i.e., type-I-risk, type-II-risk and an effect size).

Usage

```
size.cor(rho, delta, alternative = c("two.sided", "less", "greater"),
         alpha = 0.05, beta = 0.1, check = TRUE, output = TRUE)
```

Arguments

`rho`	a number indicating the correlation coefficient under the null hypothesis, ρ.0.
`delta`	a numeric value indicating the minimum difference to be detected, δ.
`alternative`	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`.
`alpha`	type-I-risk, α.
`beta`	type-II-risk, β.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown.

Value

Returns an object of class `misty.object` with following entries:

`call`	function call
`type`	type of the test (i.e., correlation coefficient)
`spec`	specification of function arguments
`res`	list with the result, i.e., optimal sample size

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>,

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. New York: John Wiley & Sons.

Rasch, D., Pilz, J., Verdooren, L. R., & Gebhardt, G. (2011). *Optimal experimental design with R*. Boca Raton: Chapman & Hall/CRC.

See Also

size.mean, size.prop

Examples

```
#--------------------------------------
# H0: rho = 0.3, H1: rho != 0.3
# alpha = 0.05, beta = 0.2, delta = 0.2

size.cor(rho = 0.3, delta = 0.2, alpha = 0.05, beta = 0.2)

#--------------------------------------
# H0: rho <= 0.3, H1: rho > 0.3
# alpha = 0.05, beta = 0.2, delta = 0.2

size.cor(rho = 0.3, delta = 0.2, alternative = "greater", alpha = 0.05, beta = 0.2)
```

---

size.mean — *Sample Size Determination for Testing Arithmetic Means*

Description

This function performs sample size computation for the one-sample and two-sample t-test based on precision requirements (i.e., type-I-risk, type-II-risk and an effect size).

Usage

```
size.mean(delta, sample = c("two.sample", "one.sample"),
          alternative = c("two.sided", "less", "greater"), alpha = 0.05,
          beta = 0.1, check = TRUE, output = TRUE)
```

Arguments

`delta`	a numeric value indicating the relative minimum difference to be detected, δ.
`sample`	a character string specifying one- or two-sample t-test, must be one of `"two.sample"` (default) or `"one.sample"`.
`alternative`	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`.
`alpha`	type-I-risk, α.
`beta`	type-II-risk, β.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown.

Value

Returns an object of class `misty.object` with following entries:

`call`	function call
`type`	type of the test (i.e., arithmetic mean)
`spec`	specification of function arguments
`res`	list with the result, i.e., optimal sample size

Author(s)

Takuya Yanagida `<takuya.yanagida@univie.ac.at>`,

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. New York: John Wiley & Sons.

Rasch, D., Pilz, J., Verdooren, L. R., & Gebhardt, G. (2011). *Optimal experimental design with R*. Boca Raton: Chapman & Hall/CRC.

See Also

`size.prop`, `size.cor`

Examples

```
#--------------------------------------
# Two-sided one-sample test
# H0: mu = mu.0, H1: mu != mu.0
# alpha = 0.05, beta = 0.2, delta = 0.5

size.mean(delta = 0.5, sample = "one.sample",
          alternative = "two.sided", alpha = 0.05, beta = 0.2)

#--------------------------------------
```

```
# One-sided one-sample test
# H0: mu <= mu.0, H1: mu > mu.0
# alpha = 0.05, beta = 0.2, delta = 0.5

size.mean(delta = 0.5, sample = "one.sample",
          alternative = "greater", alpha = 0.05, beta = 0.2)

#-------------------------------------
# Two-sided two-sample test
# H0: mu.1 = mu.2, H1: mu.1 != mu.2
# alpha = 0.01, beta = 0.1, delta = 1

size.mean(delta = 1, sample = "two.sample",
          alternative = "two.sided", alpha = 0.01, beta = 0.1)

#-------------------------------------
# One-sided two-sample test
# H0: mu.1 <= mu.2, H1: mu.1 > mu.2
# alpha = 0.01, beta = 0.1, delta = 1

size.mean(delta = 1, sample = "two.sample",
          alternative = "greater", alpha = 0.01, beta = 0.1)
```

`size.prop` *Sample Size Determination for Testing Proportions*

Description

This function performs sample size computation for the one-sample and two-sample test for proportions based on precision requirements (i.e., type-I-risk, type-II-risk and an effect size).

Usage

```
size.prop(pi = 0.5, delta, sample = c("two.sample", "one.sample"),
           alternative = c("two.sided", "less", "greater"), alpha = 0.05,
           beta = 0.1, correct = FALSE, check = TRUE, output = TRUE)
```

Arguments

`pi`	a number indicating the true value of the probability under the null hypothesis (one-sample test), π.0 or a number indicating the true value of the probability in group 1 (two-sample test), π.1.
`delta`	minimum difference to be detected, δ.
`sample`	a character string specifying one- or two-sample proportion test, must be one of "two.sample" (default) or "one.sample".
`alternative`	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"less"` or `"greater"`.
`alpha`	type-I-risk, α.

beta	type-II-risk, β.
correct	a logical indicating whether continuity correction should be applied.
check	logical: if TRUE, argument specification is checked.
output	logical: if TRUE, output is shown.

Value

Returns an object of class misty.object with following entries:

call	function call
type	type of the test (i.e., proportion)
spec	specification of function arguments
res	list with the result, i.e., optimal sample size

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>,

References

Fleiss, J. L., Levin, B., & Paik, M. C. (2003). *Statistical methods for rates and proportions* (3rd ed.). New York: John Wiley & Sons.

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. New York: John Wiley & Sons.

Rasch, D., Pilz, J., Verdooren, L. R., & Gebhardt, G. (2011). *Optimal experimental design with R*. Boca Raton: Chapman & Hall/CRC.

See Also

size.mean, size.cor

Examples

```
#--------------------------------------
# Two-sided one-sample test
# H0: pi = 0.5, H1: pi != 0.5
# alpha = 0.05, beta = 0.2, delta = 0.2

size.prop(pi = 0.5, delta = 0.2, sample = "one.sample",
          alternative = "two.sided", alpha = 0.05, beta = 0.2)

#--------------------------------------
# One-sided one-sample test
# H0: pi <= 0.5, H1: pi > 0.5
# alpha = 0.05, beta = 0.2, delta = 0.2

size.prop(pi = 0.5, delta = 0.2, sample = "one.sample",
          alternative = "less", alpha = 0.05, beta = 0.2)
```

```
#--------------------------------------
# Two-sided two-sample test
# H0: pi.1 = pi.2 = 0.5, H1: pi.1 != pi.2
# alpha = 0.01, beta = 0.1, delta = 0.2

size.prop(pi = 0.5, delta = 0.2, sample = "two.sample",
          alternative = "two.sided", alpha = 0.01, beta = 0.1)

#--------------------------------------
# One-sided two-sample test
# H0: pi.1 <=  pi.1 = 0.5, H1: pi.1 > pi.2
# alpha = 0.01, beta = 0.1, delta = 0.2

size.prop(pi = 0.5, delta = 0.2, sample = "two.sample",
          alternative = "greater", alpha = 0.01, beta = 0.1)
```

---

skewness *Skewness*

Description

This function computes the skewness.

Usage

```
skewness(x, as.na = NULL, check = TRUE)
```

Arguments

x	a numeric vector.
as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
check	logical: if TRUE, argument specification is checked.

Details

The same method for estimating skewness is used in SAS and SPSS. Missing values (NA) are stripped before the computation. Note that at least 3 observations are needed to compute skewness.

Value

Returns the estimated skewness of x.

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. New York: John Wiley & Sons.

See Also

kurtosis

Examples

```
# Set seed of the random number generation
set.seed(123)
# Generate random numbers according to N(0, 1)
x <- rnorm(100)

# Compute skewness
skewness(x)
```

---

std.coef *Standardized Coefficients*

Description

This function computes standardized coefficients for linear models estimated by using the `lm()` function.

Usage

```
std.coef(model, print = c("all", "stdx", "stdy", "stdyx"),
         digits = 3, p.digits = 3, check = TRUE, output = TRUE)
```

Arguments

`model`	a fitted model of class `"lm"`.
`print`	a character vector indicating which results to show, i.e. `"all"`, for all results, `"stdx"` for standardizing only the predictor, `"stdy"` for for standardizing only the criterion, and `"stdyx"` for for standardizing both the predictor and the criterion. Note that the default setting is depending on the level of measurement of the predictors, i.e., if all predictors are continuous, the default setting is `print = "stdyx"`; if all predictors are binary, the default setting is `print = "stdy"`; if predictors are continuous and binary, the default setting is `print = c("stdy","stdyx")`.
`digits`	an integer value indicating the number of decimal places to be used for displaying results.
`p.digits`	an integer value indicating the number of decimal places to be used for displaying the *p*-value.
`check`	logical: if `TRUE`, argument specification is checked.
`output`	logical: if `TRUE`, output is shown on the console.

Details

The slope β can be standardized with respect to only x, only y, or both y and x:

$$StdX(\beta_1) = \beta_1 SD(x)$$

$StdX(\beta_1)$ standardizes with respect to x only and is interpreted as the change in y when x changes one standard deviation referred to as $SD(x)$.

$$StdY(\beta_1) = \frac{\beta_1}{SD(x)}$$

$StdY(\beta_1)$ standardizes with respect to y only and is interpreted as the change in y standard deviation units, referred to as $SD(y)$, when x changes one unit.

$$StdYX(\beta_1) = \beta_1 \frac{SD(x)}{SD(y)}$$

$StdYX(\beta_1)$ standardizes with respect to both y and x and is interpreted as the change in y standard deviation units when x changes one standard deviation.

Note that the $StdYX(\beta_1)$ and the $StdY(\beta_1)$ standardizations are not suitable for the slope of a binary predictor because a one standard deviation change in a binary variable is generally not of interest (Muthen, Muthen, & Asparouhov, 2016).

The standardization of the slope β_3 in a regression model with an interaction term uses the product of standard deviations $SD(x_1)SD(x_2)$ rather than the standard deviation of the product $SD(x_1 x_2)$ for the interaction variable $x_1 x_2$ (see Wen, Marsh & Hau, 2010). Likewise, the standardization of the slope β_3 in a polynomial regression model with a quadratic term uses the product of standard deviations $SD(x)SD(x)$ rather than the standard deviation of the product $SD(xx)$ for the quadratic term x^2.

Value

Returns an object of class `misty.object`, which is a list with following entries: function call (`call`), type of analysis `type`, model specified in the `model` argument (`model`), specification of function arguments (`args`), list with results (`result`).

Author(s)

Takuya Yanagida `<takuya.yanagida@univie.ac.at>`

References

Muthen, B. O., Muthen, L. K., & Asparouhov, T. (2016). *Regression and mediation analysis using Mplus*. Muthen & Muthen.

Wen, Z., Marsh, H. W., & Hau, K.-T. (2010). Structural equation models of latent interactions: An appropriate standardized solution and its scale-free properties. *Structural Equation Modeling: A Multidisciplinary Journal, 17*, 1-22. https://doi.org/10.1080/10705510903438872

Examples

```
dat <- data.frame(x1 = c(3, 2, 4, 9, 5, 3, 6, 4, 5, 6, 3, 5),
                  x2 = c(1, 4, 3, 1, 2, 4, 3, 5, 1, 7, 8, 7),
                  x3 = c(0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1),
                  y = c(2, 7, 4, 4, 7, 8, 4, 2, 5, 1, 3, 8))

#----------------------------
# Linear model

#...........
# Regression model with continuous predictors
mod.lm1 <- lm(y ~ x1 + x2, data = dat)
std.coef(mod.lm1)

# Print all standardized coefficients
std.coef(mod.lm1, print = "all")

#...........
# Regression model with dichotomous predictor
mod.lm2 <- lm(y ~ x3, data = dat)
std.coef(mod.lm2)

#...........
# Regression model with continuous and dichotomous predictors
mod.lm3 <- lm(y ~ x1 + x2 + x3, data = dat)
std.coef(mod.lm3)

#...........
# Regression model with continuous predictors and an interaction term
mod.lm4 <- lm(y ~ x1*x2, data = dat)

#...........
# Regression model with a quadratic term
mod.lm5 <- lm(y ~ x1 + I(x1^2), data = dat)
std.coef(mod.lm5)
```

---

stromit *Omit Strings*

Description

This function omits user-specified values or strings from a numeric vector, character vector or factor.

Usage

```
stromit(x, omit = "", na.omit = FALSE, check = TRUE)
```

Arguments

`x`	a numeric vector, character vector or factor.
`omit`	a numeric vector or character vector indicating values or strings to be omitted from the vector `x`, the default setting is the empty strings `""`.
`na.omit`	logical: if `TRUE`, missing values (`NA`) are also omitted from the vector.
`check`	logical: if `TRUE`, argument specification is checked.

Value

Returns a numeric vector, character vector or factor with values or strings specified in `omit` omitted from the vector specified in `x`.

Author(s)

Takuya Yanagida `<takuya.yanagida@univie.ac.at>`

See Also

`mgsub`, `trim`

Examples

```
#----------------------------------------
# Charater vector
x.chr <- c("a", "", "c", NA, "", "d", "e", NA)

# Omit character string ""
stromit(x.chr)

# Omit character string "" and missing values (NA)
stromit(x.chr, na.omit = TRUE)

# Omit character string "c" and "e"
stromit(x.chr, omit = c("c", "e"))

# Omit character string "c", "e", and missing values (NA)
stromit(x.chr, omit = c("c", "e"), na.omit = TRUE)

#----------------------------------------
# Numeric vector
x.num <- c(1, 2, NA, 3, 4, 5, NA)

# Omit values 2 and 4
stromit(x.num, omit = c(2, 4))

# Omit values 2, 4, and missing values (NA)
stromit(x.num, omit = c(2, 4), na.omit = TRUE)

#----------------------------------------
# Factor
```

```
x.factor <- factor(letters[1:10])

# Omit factor levels "a", "c", "e", and "g"
stromit(x.factor, omit = c("a", "c", "e", "g"))
```

`trim` *Trim Whitespace from String*

Description

This function removes whitespace from start and/or end of a string.

Usage

```
trim(x, side = c("both", "left", "right"), check = TRUE)
```

Arguments

`x`	a character vector.
`side`	a character string indicating the side on which to remove whitespace, i.e., `"both"` (default), `"left"` or `"right"`.
`check`	logical: if `TRUE`, argument specification is checked.

Details

Note that this function is based on the `str_trim()` function in the **stringr** package by Hadley Wickham.

Value

Returns a character vector with whitespaces removed from the vector specified in `x`.

Author(s)

Takuya Yanagida `<takuya.yanagida@univie.ac.at>`

References

Wickham, H. (2019). *stringr: Simple, consistent wrappers for common string operations*. R package version 1.4.0. https://CRAN.R-project.org/package=stringr

See Also

`mgsub`, `stromit`

Examples

```
x <- "  string  "

# Remove whitespace at both sides
trim(x)

# Remove whitespace at the left side
trim(x, side = "left")

# Remove whitespace at the right side
trim(x, side = "right")
```

`write.mplus` *Write Mplus Data File*

Description

This function writes a matrix or data frame to a tab-delimited file without variable names and a text file with variable names. Only numeric values are allowed, missing data will be coded as a single numeric value.

Usage

```
write.mplus(x, file = "Mplus_Data.dat", var = TRUE, print = FALSE, na = -99,
            check = TRUE)
```

Arguments

`x`	a matrix or data frame to be written to a tab-delimited file.
`file`	a character string naming a file with or without the file extension '.dat', e.g., `"Mplus_Data.dat"` or `"Mplus_Data"`.
`var`	logical: if `TRUE`, variable names are written in a text file named according to the argument`file` with the extension `_VARNAMES.txt`.
`print`	logical: if `TRUE`, variable names are printed on the console.
`na`	a numeric value or character string representing missing values (`NA`) in the data set.
`check`	logical: if `TRUE`, argument specification is checked.

Value

None.

Author(s)

Takuya Yanagida `<takuya.yanagida@univie.ac.at>`

References

Muthen, L. K., & Muthen, B. O. (1998-2017). *Mplus User's Guide* (8th ed.). Muthen & Muthen.

See Also

read.mplus, run.mplus

Examples

```
## Not run:
dat <- data.frame(id = 1:5,
                  x = c(NA, 2, 1, 5, 6),
                  y = c(5, 3, 6, 8, 2),
                  z = c(2, 1, 1, NA, 4))

# Write Mplus Data File and a text file with variable names
write.mplus(dat)

# Write Mplus Data File "Data.dat" and a text file with variable name,
# print variable names on the console, missing values coded with -999
write.mplus(dat, file = "Data.dat", print = TRUE, na = -999)

## End(Not run)
```

---

write.sav *Write SPSS File*

Description

This function writes a data frame or matrix into a SPSS file by either using the `write_sav()` function in the **haven** package by Hadley Wickham and Evan Miller (2019) or the free software *PSPP* (see: https://www.gnu.org/software/pspp/pspp.html).

Usage

```
write.sav(x, file = "SPSS_Data.sav", var.attr = NULL, pspp.path = NULL, digits = 2,
          write.csv = FALSE, sep = c(";", ","), na = "", write.sps = FALSE,
          check = TRUE)
```

Arguments

`x`	a matrix or data frame to be written in SPSS, vectors are coerced to a data frame.
`file`	a character string naming a file with or without file extension '.sav', e.g., `"My_SPSS_Data.sav"` or `"My_SPSS_Data"`.
`var.attr`	a matrix or data frame with variable attributes used in the SPSS file, only 'variable labels' (column name `label`), 'value labels' column name `values`, and 'user-missing values' column name `missing` are supported (see 'Details').

pspp.path	a character string indicating the path where the PSPP folder is located on the computer, e.g.`C:/Program Files/PSPP/`.
digits	an integer value indicating the number of decimal places shown in the SPSS file for non-integer variables.
write.csv	logical: if `TRUE`, CSV file is written along with the SPSS file.
sep	a character string for specifying the CSV file, either `";"` for the separator and `"."` for the decimal point (default, i.e. equivalent to `write.csv2`) or `"."` for the decimal point and `","` for the separator (i.e. equivalent to `write.csv`), must be one of both `";"` (default) or `","`.
na	a character string for specifying missing values in the CSV file.
write.sps	logical: if `TRUE`, SPSS syntax is written along with the SPSS file when using PSPP.
check	logical: if `TRUE`, variable attributes specified in the argument `var.attr` is checked.

Details

If arguments `pspp.path` is not specified (i.e., `pspp.path = NULL`), `write_sav()` function in the **haven** is used. Otherwise the object `x` is written as CSV file, which is subsequently imported into SPSS using the free software *PSPP* by executing a SPSS syntax written in R. Note that *PSPP* needs to be installed on your computer when using the `pspp.path` argument.

A SPSS file with 'variable labels', 'value labels', and 'user-missing values' is written by specifying the `var.attr` argument. Note that the number of rows in the matrix or data frame specified in `var.attr` needs to match with the number of columns in the data frame or matrix specified in `x`, i.e., each row in `var.attr` represents the variable attributes of the corresponding variable in `x`. In addition, column names of the matrix or data frame specified in `var.attr` needs to be labeled as `label` for 'variable labels, `values` for 'value labels', and `missing` for 'user-missing values'.

Labels for the values are defined in the column `values` of the matrix or data frame in `var.attr` using the equal-sign (e.g., `0 = female`) and are separated by a semicolon (e.g., `0 = female; 1 = male`).

User-missing values are defined in the column `missing` of the matrix or data frame in `var.attr`, either specifying one user-missing value (e.g., `-99`) or more than one but up to three user-missing values separated by a semicolon (e.g., `-77; -99`.

Note that the part of the function using *PSPP* was adapted from the `write.pspp()` function in the **miceadds** package by Alexander Robitzsch, Simon Grund and Thorsten Henke (2019).

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

GNU Project (2018). *GNU PSPP for GNU/Linux* (Version 1.2.0). Boston, MA: Free Software Foundation. urlhttps://www.gnu.org/software/pspp/

Wickham H., & Miller, E. (2019). *haven: Import and Export 'SPSS', 'Stata' and 'SAS' Files*. R package version 2.2.0. https://CRAN.R-project.org/package=haven

Robitzsch, A., Grund, S., & Henke, T. (2019). *miceadds: Some additional multiple imputation functions, especially for mice*. R package version 3.4-17. https://CRAN.R-project.org/package=miceadds

See Also

read.sav

Examples

```
## Not run:
dat <- data.frame(id = 1:5,
                  gender = c(NA, 0, 1, 1, 0),
                  age = c(16, 19, 17, NA, 16),
                  status = c(1, 2, 3, 1, 4),
                  score = c(511, 506, 497, 502, 491))

# Write SPSS file using the haven package
write.sav(dat, file = "Dataframe_haven.sav")

# Write SPSS file using PSPP,
# write CSV file and SPSS syntax along with the SPSS file
write.sav(dat, file = "Dataframe_PSPP.sav", pspp.path = "C:/Program Files/PSPP",
          write.csv = TRUE, write.sps = TRUE)

# Specify variable attributes
# Note that it is recommended to manually specify the variables attritbues in a CSV or
# Excel file which is subsequently read into R
attr <- data.frame(# Variable names
                   var = c("id", "gender", "age", "status", "score"),
                   # Variable labels
                   label = c("Identification number", "Gender", "Age in years",
                             "Migration background", "Achievement test score"),
                   # Value labels
                   values = c("", "0 = female; 1 = male", "",
                            "1 = Austria; 2 = former Yugoslavia; 3 = Turkey; 4 = other",
                             ""),
                   # User-missing values
                   missing = c("", "-99", "-99", "-99", "-99"))

# Write SPSS file with variable attributes using the haven package
write.sav(dat, file = "Dataframe_haven_Attr.sav", var.attr = attr)

# Write SPSS with variable attributes using PSPP
write.sav(dat, file = "Dataframe_PSPP_Attr.sav", var.attr = attr,
          pspp.path = "C:/Program Files/PSPP")

## End(Not run)
```

z.test *z-Test*

Description

This function computes one sample, two sample, and paired sample z-test.

Usage

```
z.test(x, ...)

## Default S3 method:
z.test(x, y = NULL, sigma = NULL, sigma2 = NULL, mu = 0, paired = FALSE,
       alternative = c("two.sided", "less", "greater"), conf.level = 0.95,
       digits = 2, p.digits = 3, as.na = NULL, check = TRUE,
       output = TRUE, ...)

## S3 method for class 'formula'
z.test(formula, data, as.na = NULL, check = TRUE, output = TRUE, ...)
```

Arguments

x	a numeric vector of data values.
y	a numeric vector of data values.
sigma	a numeric vector indicating the population standard deviation(s). In case of two sample z-test, equal standard deviations are assumed when specifying one value for the argument `sigma`; when specifying two values for the argument `sigma`, unequal standard deviations are assumed. Note that either argument `sigma` or argument `sigma2` is specified.
sigma2	a numeric vector indicating the population variance(s). In case of two sample z-test, equal variances are assumed when specifying one value for the argument `sigma2`; when specifying two values for the argument `sigma`, unequal variance are assumed. Note that either argument `sigma` or argument `sigma2` is specified.
mu	a numeric value indicating the population mean under the null hypothesis. Note that the argument `mu` is only used when computing a one sample z-test.
paired	logical: if `TRUE`, paired sample z-test is computed.
alternative	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`.
conf.level	a numeric value between 0 and 1 indicating the confidence level of the interval.
digits	an integer value indicating the number of decimal places to be used for displaying descriptive statistics and confidence interval.
p.digits	an integer value indicating the number of decimal places to be used for displaying the *p*-value.

as.na	a numeric vector indicating user-defined missing values, i.e. these values are converted to NA before conducting the analysis.
check	logical: if TRUE, argument specification is checked.
output	logical: if TRUE, output is shown on the console.
formula	in case of two sample z-test (i.e., paired = FALSE), a formula of the form y ~ group where group is a numeric variable, character variable or factor with two values or factor levels giving the corresponding groups.
data	a matrix or data frame containing the variables in the formula formula.
...	further arguments to be passed to or from methods.

Value

Returns an object of class misty.object, which is a list with following entries: function call (call), type of analysis type, list with the input specified in x (data), specification of function arguments (args), and result table (result).

Author(s)

Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

Rasch, D., Kubinger, K. D., & Yanagida, T. (2011). *Statistics in psychology - Using R and SPSS*. John Wiley & Sons.

See Also

t.test, ci.mean.diff, ci.mean

Examples

```
dat.bs <- data.frame(group = c(1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2),
                     x = c(3, 1, 4, 2, 5, 3, 2, 3, 6, 4, 3, NA))

#-------------------------------------
# Between-Subject Design

# Two-sided one sample z-test with 95% confidence interval
# population mean = 3, population standard deviation = 1.2
z.test(dat.bs$x, sigma = 1.2, mu = 3)

# Two-sided one sample z-test with 95% confidence interval
# population mean = 3, population variance = 1.44
z.test(dat.bs$x, sigma2 = 1.44, mu = 3)

# One-sided one sample z-test with 95% confidence interval
# population mean = 3, population standard deviation = 1.2
z.test(dat.bs$x, sigma = 1.2, mu = 3, alternative = "greater")

# Two-sided one sample z-test with 95% confidence interval
```

```
# population mean = 3, population standard deviation = 1.2
# # convert value 3 to NA
z.test(dat.bs$x, sigma = 1.2, mu = 3, as.na = 3)

# Two-sided one sample z-test with 99% confidence interval
# population mean = 3, population standard deviation = 1.2
z.test(dat.bs$x, sigma = 1.2, mu = 3, conf.level = 0.99)

# Two-sided one sample z-test with 95% confidence interval
# population mean = 3, population standard deviation = 1.2
# print descriptive statistics with 3 digits and p-value with 5 digits
z.test(dat.bs$x, sigma = 1.2, mu = 3, digits = 3, p.digits = 5)

# Two-sided two sample z-test with 95% confidence interval
# population standard deviation (SD) = 1.2, equal SD assumption
z.test(x ~ group, sigma = 1.2, data = dat.bs)

# Two-sided two sample z-test with 95% confidence interval
# population standard deviation = 1.2 and 1.5
z.test(x ~ group, sigma = c(1.2, 1.5), data = dat.bs)

# Two-sided two sample z-test with 95% confidence interval
# population variance = 1.44 and 2.25
z.test(x ~ group, sigma = c(1.44, 2.25), data = dat.bs)

# One-sided two sample z-test with 95% confidence interval
# population standard deviation (SD) = 1.2, equal SD assumption
z.test(x ~ group, sigma = 1.2, data = dat.bs, alternative = "less")

#-----------------

group1 <- c(3, 1, 4, 2, 5, 3, 6, 7)
group2 <- c(5, 2, 4, 3, 1)

# Two-sided two sample z-test with 95% confidence interval
# population standard deviation (SD) = 1.2, equal SD assumption
z.test(group1, group2, sigma = 1.2, data = dat.bs)

#--------------------------------------
# Within-Subject Design
dat.ws <- data.frame(pre = c(1, 3, 2, 5, 7),
                     post = c(2, 2, 1, 6, 8), stringsAsFactors = FALSE)

# Two-sided paired sample z-test with 95% confidence interval
# population standard deviation of difference score = 1.2
z.test(dat.ws$pre, dat.ws$post, sigma = 1.2, paired = TRUE)

# Two-sided paired sample z-test with 95% confidence interval
# population variance of difference score = 1.44
z.test(dat.ws$pre, dat.ws$post, sigma2 = 1.44, paired = TRUE)

# One-sided paired sample z-test with 95% confidence interval
# population standard deviation of difference score = 1.2
```

Index