Package ‘misty’

June 8, 2022

Type Package
Title Miscellaneous Functions 'T. Yanagida'
Version 0.4.6
Date 2022-06-08
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, multilevel R-squared measures, within-group and between-group correlation matrix, various effect size measures), 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), missing data (e.g., descriptive statistics for missing data, missing data pattern, Little's test of Missing Completely at Random, and auxiliary variable analysis), item analysis (e.g., coefficient alpha and omega, confirmatory factor analysis), and statistical analysis (e.g., confidence intervals, collinearity diagnostics, analysis of variance, Levene's test, t-test, z-test, sample size determination).
Depends R (>= 3.5.0)
License MIT + file LICENSE
Imports ggplot2, haven, lavaan, lme4, nlme, norm, r2mlm, readxl, writexl
Suggests mnormt, plyr
Encoding UTF-8
NeedsCompilation no
Repository CRAN
Date/Publication 2022-06-08 10:50:02 UTC

R topics documented:

  aov.b  ......................................................... 3
  as.na  ......................................................... 6
  center ......................................................... 8
R topics documented:

- chr.gsub
- chr.omit
- chr.trim
- ci.mean
- ci.mean.diff
- ci.median
- ci.prop
- ci.prop.diff
- ci.sd
- ci.var
- cluster.scores
- cohens.d
- collin.diag
- cor.cont
- cor.cramer
- cor.matrix
- cor.phi
- cor.poly
- crosstab
- descrip
- df.duplicated
- df.merge
- df.rbind
- df.rename
- df.sort
- dummy.c
- eta.sq
- freq
- indirect
- item.alpha
- item.cfa
- item.omega
- item.reverse
- item.scores
- kurtosis
- multilevel.cor
- multilevel.descript
- multilevel.icc
- multilevel.indirect
- multilevel.r2
- na.as
- na.auxiliary
- na.coverage
- na.descript
- na.indicator
- na.pattern
- na.prop
- na.test
Between-Subject Analysis of Variance

This function performs an one-way between-subject analysis of variance (ANOVA) including Tukey HSD post hoc test for multiple comparison and provides descriptive statistics, effect size measures, and a plot showing error bars for confidence intervals with jittered data points.

Usage

```r
aov.b(formula, data, posthoc = TRUE, conf.level = 0.95, hypo = TRUE,
      descript = TRUE, effsize = FALSE, weighted = FALSE, correct = FALSE,
      plot = FALSE, point.size = 4, error.width = 0.1, xlab = NULL, ylab = NULL,
      ylim = NULL, breaks = ggplot2::waiver(), jitter = TRUE,
      jitter.size = 1.25, jitter.width = 0.05, jitter.alpha = 0.1,
      title = "", subtitle = "Confidence Interval",
      digits = 2, p.digits = 4, as.na = NULL, check = TRUE,
      output = TRUE, ...)
```
Arguments

formula a formula of the form \( y \sim \text{group} \) where \( y \) is a numeric variable giving the data values and \( \text{group} \) a numeric variable, character variable or factor with more than two values or factor levels giving the corresponding groups.
data a matrix or data frame containing the variables in the formula \( \text{formula} \).
posthoc logical: if TRUE, Tukey HSD post hoc test for multiple comparison is conducted.
conf.level a numeric value between 0 and 1 indicating the confidence level of the interval for Cohen’s \( d \). Note that this argument is only used when conducting Welch’s two-sample t-test.
hypo logical: if TRUE, null and alternative hypothesis are shown on the console.
descript logical: if TRUE, descriptive statistics are shown on the console.
effsize logical: if TRUE, effect size measure Cohen’s \( d \) for Welch’s two-sample t-test (see \texttt{cohens.d}), \( \eta^2 \) and \( \omega^2 \) for the ANOVA and Cohen’s \( d \) for the post hoc tests are shown on the console.
weighted logical: if TRUE, the weighted pooled standard deviation is used to compute Cohen’s \( d \).
correct logical: if TRUE, correction factor to remove positive bias in small samples is used.
plot logical: if TRUE, a plot showing error bars for confidence intervals is drawn.
point.size a numeric value indicating the size aesthetic for the point representing the mean value.
error.width a numeric value indicating the horizontal bar width of the error bar.
xlab a character string specifying the labels for the x-axis.
ylab a character string specifying the labels for the y-axis.
ylim a numeric vector of length two specifying limits of the limits of the y-axis.
breaks a numeric vector specifying the points at which tick-marks are drawn at the y-axis.
jitter logical: if TRUE (default), jittered data points are drawn.
jitter.size a numeric value indicating the size aesthetic for the jittered data points.
jitter.width a numeric value indicating the amount of vertical and horizontal jitter.
jitter.alpha a numeric value indicating the opacity of the jittered data points.
title a character string specifying the text for the title for the plot.
subtitle a character string specifying the text for the subtitle for the plot.
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.
... further arguments to be passed to or from methods.
**Details**

Note that by default Tukey HSD post hoc test reports Cohen’s d based on the weighted standard deviation (i.e., `weighted = TRUE`) when requesting an effect size measure (i.e., `effsize = TRUE`) following the recommendation by Delacre et al. (2021).

**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(s) (`result`).

**Author(s)**

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

**References**


**See Also**

test.t, test.z, test.levene, test.welch, cohens.d, ci.mean.diff, ci.mean

**Examples**

dat <- data.frame(group = c(1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3),
                  y = c(3, 1, 4, 2, 5, 3, 2, 3, 6, 6, 3, NA))

# Between-subject ANOVA
aov.b(y ~ group, data = dat)

# Between-subject ANOVA
# print effect size measures
aov.b(y ~ group, data = dat, effsize = TRUE)

# Between-subject ANOVA
# do not print hypotheses and descriptive statistics,
aov.b(y ~ group, data = dat, descript = FALSE, hypo = FALSE)

## Not run:
# Between-subject ANOVA
# plot results
aov.b(y ~ group, data = dat, plot = TRUE)

# Load ggplot2 package
library(ggplot2)

# Save plot, ggsave() from the ggplot2 package
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, array, list, or data frame with NA.

Usage

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

Arguments

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

Value

Returns x with values specified in na replaced with NA.

Author(s)

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

References

See Also

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

Examples

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

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

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

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

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

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

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

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

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

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

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

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

#--------------------------------------
# Array
x.array <- array(1:20, dim = c(2, 3, 2))

# Replace 1 and 10 with NA
as.na(x.array, na = c(1, 10))
# 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, na = 1)

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

---

center

Centering at the Grand Mean and 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"), cluster = 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).

cluster
a vector representing the nested grouping structure (i.e., group or cluster variable) 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 cluster.

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 cluster = 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 cluster = NULL (default) or within cluster (CWC) by specifying type = "CWC" in conjunction with specifying a cluster membership variable using the cluster 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 cluster membership variable using the cluster 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


See Also
dummy.c, cluster.scores, rec, item.reverse, rwg.lindell, item.scores.

Examples

```r
#--------------------------------------
# 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))

# 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),
                      cluster = 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$x.l1, type = "CWC", cluster = dat.ml$cluster)

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

---

### chr.gsub

**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**

`chr.gsub(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.
**chr.omit**

**Value**

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

**Note**

This function was adapted from the `mgsub()` function in the `mgsub` package by Mark Ewing (2019).

**Author(s)**

Mark Ewing

**References**


**See Also**

`chr.omit`, `chr.trim`

**Examples**

```r
string <- c("hey ho, let's go!")
chr.gsub(c("hey", "ho"), c("ho", "hey"), string)

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

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

string <- "Dopazamine is not the same as dopachloride or dopastriamine, yet is still fake."
chr.gsub(c("[Dd]opa\([\s\S]*mine\)\,"fake"), c("Meta\[\]|"real"), string)
```

---

**Description**

This function omits user-specified values or strings from a numeric vector, character vector or factor.

**Usage**

```r
chr.omit(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

- `chr.gsub`, `chr.trim`

Examples

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

# Omit character string ","
chr.omit(x.chr)

# Omit character string "," and missing values (NA)
chr.omit(x.chr, na.omit = TRUE)

# Omit character string "c" and "$"
chr.omit(x.chr, omit = c("c", "$"))

# Omit character string "c", "$", and missing values (NA)
chr.omit(x.chr, omit = c("c", "$"), na.omit = TRUE)

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

# Omit values 2 and 4
chr.omit(x.num, omit = c(2, 4))

# Omit values 2, 4, and missing values (NA)
chr.omit(x.num, omit = c(2, 4), na.omit = TRUE)

#--------------------------------------
# Factor
```
x.factor <- factor(letters[1:10])

# Omit factor levels "a", "c", "e", and "g"
chr.omit(x.factor, omit = c("a", "c", "e", "g"))

---

**chr.trim**

*Trim Whitespace from String*

**Description**

This function removes whitespace from start and/or end of a string.

**Usage**

```r
chr.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.

**Value**

Returns a character vector with whitespaces removed from the vector specified in `x`.

**Note**

This function is based on the `str_trim()` function from the `stringr` package by Hadley Wickham.

**Author(s)**

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

**References**


**See Also**

`chr.gsub, chr.omit`
Examples

```r
x <- " string 

# Remove whitespace at both sides
chr.trim(x)

# Remove whitespace at the left side
chr.trim(x, side = "left")

# Remove whitespace at the right side
chr.trim(x, side = "right")
```

---

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

```r
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.
ci.mean

- **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.

- **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**


**See Also**

test.z, test.t, ci.mean.diff, ci.median, ci.prop, ci.var, ci.sd, descript

**Examples**

```r
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)
```
ci.mean.diff

Confidence Interval for the Difference in Arithmetic Means

Description

This function computes a confidence interval for the difference in arithmetic means in a two-sample and paired-sample design 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'

**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 sigma2, 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


See Also

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

Examples

dat1 <- data.frame(group1 = c(1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2),
  1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2),
group2 = c(1, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2, 1, 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),
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, NA))

#--------------------------------------
# Two-sample design
# Two-Sided 95% CI for y1 by group1
# unknown population variances, unequal variance assumption
ci.mean.diff(x1 ~ group1, data = dat1)

# Two-Sided 95% CI for y1 by group1
# unknown population variances, equal variance assumption
ci.mean.diff(x1 ~ group1, data = dat1, var.equal = TRUE)

# Two-Sided 95% CI with known standard deviations for x1 by group1
# known population standard deviations, equal standard deviation assumption
ci.mean.diff(x1 ~ group1, data = dat1, sigma = 1.2)

# Two-Sided 95% CI with known standard deviations for x1 by group1
# known population standard deviations, unequal standard deviation assumption
ci.mean.diff(x1 ~ group1, data = dat1, sigma = c(1.5, 1.2))

# Two-Sided 95% CI with known variance for x1 by group1
# known population variances, equal variance assumption
ci.mean.diff(x1 ~ group1, data = dat1, sigma2 = 1.44)

# Two-Sided 99% CI for y1 by group1
# known population variances, unequal variance assumption
ci.mean.diff(x1 ~ group1, data = dat1, conf.level = 0.99)

# Convert value 4 to NA
# unknown population variances, unequal variance assumption
ci.mean.diff(x1 ~ group1, data = dat1, as.na = 4)

# Two-Sided 95% CI for y1, y2, and y3 by group1
# unknown population variances, unequal variance assumption

ci.mean.diff(cbind(x1, x2, x3) ~ group1, data = dat1)

# Two-Sided 95% CI for y1, y2, and y3 by group1
# unknown population variances, unequal variance assumption,
# listwise deletion for missing data
# Two-Sided 95% CI for y1, y2, and y3 by group1
# unknown population variances, unequal variance assumption,
# analysis by group2 separately
# Two-Sided 95% CI for y1, y2, and y3 by group1
# unknown population variances, unequal variance assumption,
# analysis by group2 separately, sort by variables
# Two-Sided 95% CI for y1, y2, and y3 by group1
# unknown population variances, unequal variance assumption,
# split analysis by group2
# Two-Sided 95% CI for y1, y2, and y3 by group1
# unknown population variances, unequal variance assumption,
# split analysis by group2 separately, split analysis by group3

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

# Paired sample design

dat2 <- data.frame(pre = c(1, 3, 2, 5, 7, 6),
                   post = c(2, 2, 1, 6, 8, 9),
                   group = c(1, 1, 1, 2, 2, 2), stringsAsFactors = FALSE)

# Two-Sided 95% CI for the mean difference in pre and post
# unknown population variance of difference scores
# Two-Sided 95% CI for the mean difference in pre and post
# unknown population variance of difference scores
# Paired sample design
# Two-Sided 95% CI for the mean difference in pre and post
# unknown population variance of difference scores
# analysis by group separately
.ci.mean.diff(dat2$pre, dat2$post, paired = TRUE, group = dat2$group)

# Two-Sided 95% CI for the mean difference in pre and post
# unknown population variance of difference scores
# split analysis by group
.ci.mean.diff(dat2$pre, dat2$post, paired = TRUE, split = dat2$group)

# Two-Sided 95% CI for the mean difference in pre and post
# known population standard deviation of difference scores
.ci.mean.diff(dat2$pre, dat2$post, sigma = 2, paired = TRUE)

# Two-Sided 95% CI for the mean difference in pre and post
# known population variance of difference scores
.ci.mean.diff(dat2$pre, dat2$post, sigma2 = 4, paired = TRUE)

# One-Sided 95% CI for the mean difference in pre and post
# unknown population variance of difference scores
.ci.mean.diff(dat2$pre, dat2$post, alternative = "less", paired = TRUE)

# Two-Sided 99% CI for the mean difference in pre and post
# unknown population variance of difference scores
.ci.mean.diff(dat2$pre, dat2$post, conf.level = 0.99, paired = TRUE)

# Two-Sided 95% CI for for the mean difference in pre and post
# unknown population variance of difference scores
# print results with 3 digits
.ci.mean.diff(dat2$pre, dat2$post, paired = TRUE, digits = 3)

# Two-Sided 95% CI for for the mean difference in pre and post
# unknown population variance of difference scores
# convert value 1 to NA
.ci.mean.diff(dat2$pre, dat2$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**

```r
.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


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, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 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, 1, 2, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2, 2),
                 x1 = c(3, 1, 4, 2, 5, 3, 2, 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% CI for x1
.ci.median(dat$x1)

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

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

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

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

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

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

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

# Two-Sided 95% CI 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

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

Usage
```r
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).
ci.prop

Author(s)
Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

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, 1, 0, 1, 1, 1),
                  x3 = c(1, 1, 1, 0, 1, NA, 1, NA, 0, 0, 0, 1))

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

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

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

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

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

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

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

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

# Two-Sided 95% CI 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 in a two-sample and paired-sample design for one or more variables, optionally by a grouping and/or split variable.

**Usage**

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

## Default S3 method:

```r
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'

```r
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 \( \text{cbind}(y_1, y_2, y_3) \sim \text{group} \) 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


See Also

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

Examples

dat1 <- data.frame(group1 = c(1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2),
                   group2 = c(1, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2, 2, 2),
                   group3 = c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1),
                   x1 = c(0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1),
                   x2 = c(0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1),
                   x3 = c(1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1))

#--------------------------------------
# Two-sample design
# Two-Sided 95% CI for x1 by group1
# Newcombes Hybrid Score interval
ci.prop.diff(x1 ~ group1, data = dat1)

# Two-Sided 95% CI for x1 by group1
# Wald CI
ci.prop.diff(x1 ~ group1, data = dat1, method = "wald")

# One-Sided 95% CI for x1 by group1
# Newcombes Hybrid Score interval
ci.prop.diff(x1 ~ group1, data = dat1, alternative = "less")

# Two-Sided 99% CI for x1 by group1
# Newcombes Hybrid Score interval
ci.prop.diff(x1 ~ group1, data = dat1, conf.level = 0.99)

# Two-Sided 95% CI for y1 by group1
# Newcombes Hybrid Score interval, print results with 3 digits
ci.prop.diff(x1 ~ group1, data = dat1, digits = 3)

# Two-Sided 95% CI for y1 by group1
# Newcombes Hybrid Score interval, convert value 0 to NA
ci.prop.diff(x1 ~ group1, data = dat1, as.na = 0)
# Two-Sided 95% CI for y1, y2, and y3 by group1
# Newcombes Hybrid Score interval
ci.prop.diff(cbind(x1, x2, x3) ~ group1, data = dat1)

# Two-Sided 95% CI 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 = dat1, na.omit = TRUE)

# Two-Sided 95% CI for y1, y2, and y3 by group1
# Newcombes Hybrid Score interval, analysis by group2 separately
# ci.prop.diff(cbind(x1, x2, x3) ~ group1, data = dat1, group = dat1$group2)

# Two-Sided 95% CI 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 = dat1, group = dat1$group2,
# sort.var = TRUE)

# Two-Sided 95% CI for y1, y2, and y3 by group1
# split analysis by group2
# ci.prop.diff(cbind(x1, x2, x3) ~ group1, data = dat1, split = dat1$group2)

# Two-Sided 95% CI 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 = dat1,
# group = dat1$group2, split = dat1$group3)

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

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

#--------------------------------------
# Paires-sample design

dat2 <- data.frame(pre = c(0, 1, 1, 0, 1),
   post = c(1, 1, 0, 1, 1), stringsAsFactors = FALSE)

# Two-Sided 95% CI for the mean difference in x1 and x2
# Newcombes Hybrid Score interval
# ci.prop.diff(dat2$pre, dat2$post, paired = TRUE)

# Two-Sided 95% CI for the mean difference in x1 and x2
# Wald CI
# ci.prop.diff(dat2$pre, dat2$post, method = "wald", paired = TRUE)

# One-Sided 95% CI for the mean difference in x1 and x2
# Newcombes Hybrid Score interval
# ci.prop.diff(dat2$pre, dat2$post, alternative = "less", paired = TRUE)
# Two-Sided 99% CI for the mean difference in x1 and x2
# Newcombe's Hybrid Score interval
\texttt{ci.prop.diff(dat2$pre, dat2$post, conf.level = 0.99, paired = TRUE)}

# Two-Sided 95% CI for the mean difference in x1 and x2
# Newcombe's Hybrid Score interval, print results with 3 digits
\texttt{ci.prop.diff(dat2$pre, dat2$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**

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

**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


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, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 2, 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, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 2, 2, 2),
x1 = c(3, 1, 4, 2, 5, 3, 2, 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, 6, 2, 2, 4))

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

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

# One-Sided 95% CI for x1
ci.sd(dat$x1, alternative = "less")
# Two-Sided 99% CI
ci.sd(dat$x1, conf.level = 0.99)

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

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

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

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

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

# Two-Sided 95% CI for x1, x2, and x3, split 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


See Also

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

```r
dat <- data.frame(group1 = c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2),
group2 = c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2),
x1 = c(3, 1, 4, 2, 5, 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% CI for x1
ci.var(dat$x1)

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

# One-Sided 95% CI for x1
# ci.var(dat$x1, alternative = "less")

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

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

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

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

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

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

# Two-Sided 95% CI 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)
```
Description

This function computes cluster means by default.

Usage

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

Arguments

- `x`: a numeric vector.
- `cluster`: a vector representing the nested grouping structure (i.e., group or cluster variable).
- `fun`: character string indicating the function used to compute cluster scores, default: "mean".
- `expand`: logical: if TRUE, vector of cluster 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 `cluster`.
- `check`: logical: if TRUE, argument specification is checked.

Value

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

Author(s)

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

References


See Also

- `item.scores`
- `multilevel.descript`
- `multilevel.icc`
Examples

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

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

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

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

---

**cohens.d**

*Cohen’s d*

**Description**

This function computes Cohen’s d for one-sample, two-sample (i.e., between-subject design), and paired-sample designs (i.e., within-subject design) for one or more variables, optionally by a grouping and/or split variable. In a two-sample design, the function computes the standardized mean difference by dividing the difference between means of the two groups of observations by the weighted pooled standard deviation (i.e., Cohen’s $d_s$ according to Lakens, 2013) by default. In a paired-sample design, the function computes the standardized mean difference by dividing the mean of the difference scores by the standard deviation of the difference scores (i.e., Cohen’s $d_z$ according to Lakens, 2013) by default. Note that by default Cohen’s d is computed without applying the correction factor for removing the small sample bias (i.e., Hedges’ g).

**Usage**

```r
cohens.d(x, ...)
```

```
# Default S3 method:
cohens.d(x, y = NULL, mu = 0, paired = FALSE, weighted = TRUE, cor = TRUE,
         ref = NULL, correct = 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'
cohens.d(formula, data, weighted = TRUE, cor = TRUE, ref = NULL,
         correct = 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.
mu     a numeric value indicating the reference mean.
paired logical: if TRUE, Cohen’s d for a paired-sample design is computed.
weighted logical: if TRUE (default), the weighted pooled standard deviation is used to compute the standardized mean difference between two groups of a two-sample design (i.e., paired = FALSE), while standard deviation of the difference scores is used to compute the standardized mean difference in a paired-sample design (i.e., paired = TRUE).
cor    logical: if TRUE (default), paired = TRUE, and weighted = FALSE, Cohen’s d for a paired-sample design while controlling for the correlation between the two sets of measurement is computed. Note that this argument is only used in a paired-sample design (i.e., paired = TRUE) when specifying weighted = FALSE.
ref    character string "x" or "y" for specifying the reference reference group when using the default cohens.d() function or a numeric value or character string indicating the reference group in a two-sample design using the formula cohens.d() function. The standard deviation of the reference variable or reference group is used to standardized the mean difference. Note that this argument is only used in a two-sample design (i.e., paired = FALSE).
correct logical: if TRUE, correction factor to remove positive bias in small samples is used.
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.
digits   an integer value indicating the number of decimal places to be used for displaying results.
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 two-sample design, while as.na() function is applied to pre and post in a paired-sample design.
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 giving the data values and group 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.
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.
Details

Cohen (1988, p.67) proposed to compute the standardized mean difference in a two-sample design 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 = 0 if the control group is coded with 0) to compute the standardized mean difference in a two-sample design (i.e., Glass’s Δ) 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, default). 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 overestimated 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 the function uses a gamma function for computing the correction factor, while a approximation method is used if computation based on the gamma function fails.

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 small sample correction factor is applied. Cumming and Calin-Jageman (2017, p.171) recommended to avoid the term Hedges’ g, 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 a two-sample design (i.e., paired = FALSE) Cohen’s $d_s$ is computed when using weighted = TRUE (default) and Hedges’s $g_s$ is computed when using correct = TRUE in addition. In a paired-sample design (i.e., paired = TRUE), Cohen’s $d_z$ is computed when using weighted = TRUE, default, while Cohen’s $d_{rm}$ is computed when using weighted = FALSE and cor = TRUE, default and Cohen’s $d_{av}$ is computed when using weighted = FALSE and cor = FALSE. Corresponding Hedges’ $g_z$, eqng_rm.

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


cohens.d


See Also

eta.sq, cor.cont, cor.cramer, cor.matrix, na.auxiliary

Examples

dat1 <- data.frame(
  group1 = c(1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 1, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 1),
  group2 = c(1, 2, 1, 2, 1, 1, 1, 2, 1, 2, 2, 1, 1, 1, 2, 2, 1, 1, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 1),
  group3 = c(1, 2, 1, 2, 1, 2, 2, 2, 1, 2, 2, 1, 1, 1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1),
  x1 = c(3, 2, 5, 3, 6, 3, 2, 4, 6, 5, 3, 3, 5, 4, 4, 3, 5, 3, 2, 3, 3, 6, 6, 7, 5, 6, 6, 4),
  x2 = c(4, 4, 3, 6, 4, 7, 3, 5, 3, 3, 4, 2, 3, 6, 3, 5, 2, 6, 8, 3, 2, 5, 4, 5, 3, 2, 2, 4),
  x3 = c(7, 6, 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),
  stringsAsFactors = FALSE)

#--------------------------------------
# One-sample design

# Cohen's d.z with two-sided 95% CI
# population mean = 3
cohens.d(dat1$x1, mu = 3)

# Cohen's d.z (aka Hedges' g.z) with two-sided 95% CI
# population mean = 3, with small sample correction factor
cohens.d(dat1$x1, mu = 3, correct = TRUE)

# Cohen's d.z for more than one variable with two-sided 95% CI
# population mean = 3
cohens.d(dat1[, c("x1", "x2", "x3")], mu = 3)

# Cohen's d.z with two-sided 95% CI
# population mean = 3, by group1 separately
cohens.d(dat1$x1, mu = 3, group = dat1$group1)

# Cohen's d.z for more than one variable with two-sided 95% CI
# population mean = 3, by group1 separately
cohens.d(dat1[, c("x1", "x2", "x3")], mu = 3, group = dat1$group1)
# Cohen's d.z with two-sided 95% CI
# population mean = 3, split analysis by group1
cohens.d(dat1$x1, mu = 3, split = dat1$group1)

# Cohen's d.z for more than one variable with two-sided 95% CI
# population mean = 3, split analysis by group1
cohens.d(dat1[, c("x1", "x2", "x3")], mu = 3, split = dat1$group1)

# Cohen's d.z with two-sided 95% CI
# population mean = 3, by group1 separately, split by group2
cohens.d(dat1$x1, mu = 3, group = dat1$group1, split = dat1$group2)

# Cohen's d.z for more than one variable with two-sided 95% CI
# population mean = 3, by group1 separately, split by group2
cohens.d(dat1[, c("x1", "x2", "x3")], mu = 3, group = dat1$group1,
          split = dat1$group2)

#--------------------------------------
# Two-sample design
# Cohen's d.s with two-sided 95% CI
# weighted pooled SD
cohens.d(x1 ~ group1, data = dat1)

# Cohen's d.s with two-sided 99% CI
# weighted pooled SD
cohens.d(x1 ~ group1, data = dat1, conf.level = 0.99)

# Cohen's d.s with one-sided 99% CI
# weighted pooled SD
cohens.d(x1 ~ group1, data = dat1, alternative = "greater")

# Cohen's d.s with two-sided 99% CI
# weighted pooled SD
cohens.d(x1 ~ group1, data = dat1, conf.level = 0.99)

# Cohen's d.s with one-sided 95% CI
# weighted pooled SD
cohens.d(x1 ~ group1, data = dat1, alternative = "greater")

# Cohen's d.s for more than one variable with two-sided 95% CI
# weighted pooled SD
cohens.d(cbind(x1, x2, x3) ~ group1, data = dat1)

# Cohen's d with two-sided 95% CI
# unweighted SD
cohens.d(x1 ~ group1, data = dat1, weighted = FALSE)

# Cohen's d.s (aka Hedges' g.s) with two-sided 95% CI
# weighted pooled SD, with small sample correction factor
cohens.d(x1 ~ group1, data = dat1, correct = TRUE)
# Cohen's d (aka Hedges' g) with two-sided 95% CI
# Unweighted SD, with small sample correction factor
cohens.d(x1 ~ group1, data = dat1, weighted = FALSE, correct = TRUE)

# Cohen's d (aka Glass's delta) with two-sided 95% CI
# SD of reference group 1
cohens.d(x1 ~ group1, data = dat1, ref = 1)

# Cohen's d.s with two-sided 95% CI
# weighted pooled SD, by group2 separately
cohens.d(x1 ~ group1, data = dat1, group = dat1$group2)

# Cohen's d.s for more than one variable with two-sided 95% CI
# weighted pooled SD, by group2 separately
cohens.d(cbind(x1, x2, x3) ~ group1, data = dat1, group = dat1$group2)

# Cohen's d.s with two-sided 95% CI
# weighted pooled SD, split analysis by group2
cohens.d(x1 ~ group1, data = dat1, split = dat1$group2)

# Cohen's d.s for more than one variable with two-sided 95% CI
# weighted pooled SD, split analysis by group2
cohens.d(cbind(x1, x2, x3) ~ group1, data = dat1, split = dat1$group2)

# Cohen's d.s with two-sided 95% CI
# weighted pooled SD, by group2 separately, split analysis by group3
cohens.d(x1 ~ group1, data = dat1, group = dat1$group2, split = dat1$group3)

# Cohen's d.s for more than one variable with two-sided 95% CI
# weighted pooled SD, by group2 separately, split analysis by group3
cohens.d(cbind(x1, x2, x3) ~ group1, data = dat1, group = dat1$group2, split = dat1$group3)

# Paired-sample design

# Cohen's d.z with two-sided 95% CI
# SD of the difference scores
cohens.d(dat1$x1 - dat1$x2, paired = TRUE)

# Cohen's d.z with two-sided 99% CI
# SD of the difference scores
cohens.d(dat1$x1 - dat1$x2, paired = TRUE, conf.level = 0.99)

# Cohen's d.z with one-sided 95% CI
# SD of the difference scores
cohens.d(dat1$x1 - dat1$x2, paired = TRUE, alternative = "greater")

# Cohen's d.rm with two-sided 95% CI
# controlling for the correlation between measures
cohens.d(dat1$x1, dat1$x2, paired = TRUE, weighted = FALSE)
# Cohen's d.av with two-sided 95% CI
# without controlling for the correlation between measures
cohens.d(dat1$x1, dat1$x2, paired = TRUE, weighted = FALSE, cor = FALSE)

# Cohen's d.z (aka Hedges' g.z) with two-sided 95% CI
# SD of the difference scores
cohens.d(dat1$x1, dat1$x2, paired = TRUE, correct = TRUE)

# Cohen's d.rm (aka Hedges' g.rm) with two-sided 95% CI
# controlling for the correlation between measures
cohens.d(dat1$x1, dat1$x2, paired = TRUE, weighted = FALSE, correct = TRUE)

# Cohen's d.av (aka Hedges' g.av) with two-sided 95% CI
# without controlling for the correlation between measures
cohens.d(dat1$x1, dat1$x2, paired = TRUE, weighted = FALSE, cor = FALSE,
correct = TRUE)

# Cohen's d.z with two-sided 95% CI
# SD of the difference scores, by group1 separately
cohens.d(dat1$x1, dat1$x2, paired = TRUE, group = dat1$group1)

# Cohen's d.z with two-sided 95% CI
# SD of the difference scores, split analysis by group1
cohens.d(dat1$x1, dat1$x2, paired = TRUE, split = dat1$group1)

# Cohen's d.z with two-sided 95% CI
# SD of the difference scores, by group1 separately, split analysis by group2
cohens.d(dat1$x1, dat1$x2, paired = TRUE,
group = dat1$group1, split = dat1$group2)

---

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".
**collin.diag**

- `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, `lm()` 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, \( \text{GVIF}^{\frac{1}{2d_f}} \) 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`.

**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`).

**Note**

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).

**Author(s)**

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

**References**

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, 6, 4, 5, 6, 3, 5, 5),
       x2 = c(1, 4, 3, 2, 4, 3, 5, 1, 7, 8, 7, 2),
       x3 = c(7, 3, 4, 2, 5, 6, 4, 2, 3, 5, 2, 8),
       x4 = c("a", "b", "a", "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-effects 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-effects 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-effects 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)

---

cor.cont  

**Pearson’s Contingency Coefficient**  

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

**Usage**  
cor.cont(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, \( C/C_{\text{max}} \).

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>

cor.matrix, cor.cramer, cor.phi, cor.poly, cohes.d.

References


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
cor.cont(dat[, c("x", "y")])

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

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

<table>
<thead>
<tr>
<th>cor.cramer</th>
<th>Cramer's V</th>
</tr>
</thead>
</table>

## Description

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

## Usage

```r
cor.cramer(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


See Also

cor.matrix, cor.cont, cor.phi, cor.poly, cohens.d.

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, 2, 1, 2, 3, 1, 2, 3, 2))

# Bias-corrected Cramer’s V between x and y
cor.cramer(dat[, c("x", "y")])

# Cramer’s V between x and y
# cor.cramer(dat[, c("x", "y")], correct = FALSE)

# Bias-corrected Cramer’s V matrix between x, y, and z
cor.cramer(dat[, c("x", "y", "z")])

# Cramer’s V matrix between x, y, and z
cor.cramer(dat[, c("x", "y", "z")], correct = FALSE)

cor.matrix

Correlation Matrix

Description

This function computes a correlation matrix based on Pearson product-moment correlation coefficient, Spearman’s rank-order correlation coefficient, Kendall’s Tau-b correlation coefficient, or Kendall-Stuart’s Tau-c correlation coefficient and computes significance values (p-values) for testing the hypothesis H0: \( \rho = 0 \) for all pairs of variables.

Usage

cor.matrix(x, method = c("pearson", "spearman", "kendall-b", "kendall-c"),
          na.omit = FALSE, group = NULL, sig = FALSE, alpha = 0.05,
          print = c("all", "cor", "n", "stat", "df", "p"),
          tri = c("both", "lower", "upper"),
          p.adj = c("none", "bonferroni", "holm", "hochberg", "hommel",
                    "BH", "BY", "fdr"), continuity = TRUE,
          digits = 2, p.digits = 3, as.na = NULL,
          write = 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.
na.omit  logical: if TRUE, incomplete cases are removed before conducting the analysis (i.e., listwise deletion); if FALSE (default), pairwise deletion is used.
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.
sig  logical: if TRUE, statistically significant correlation coefficients are shown in boldface on the console.
alpha  a numeric value between 0 and 1 indicating the significance level at which correlation coefficients are printed boldface when sig = TRUE.
print  a character string or character vector indicating which results to show on the console, i.e. "all" for all results, "cor" for correlation coefficients, "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. "all" for all results, "cor" for correlation coefficients, "n" for the sample sizes, and "p" for p-values.
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.
continuity  logical: if TRUE (default), continuity correction is used for testing Spearman’s rank-order correlation coefficient and Kendall’s Tau-b correlation.
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.
write  a character string for writing the results into a Excel file naming a file with or without file extension '.xlsx', e.g., "Results.xlsx" or "Results".
check  logical: if TRUE, argument specification is checked.
output  logical: if TRUE, output is shown on the console.

Details

Note that unlike the cor.test function, this function does not compute an exact p-value for Spearman’s rank-order correlation coefficient or Kendall’s Tau-b correlation coefficient, but uses the asymptotic t approximation.
Statistically significant correlation coefficients can be shown in boldface on the console when specifying `sig = TRUE`. However, this option is not supported when using R Markdown, i.e., the argument `sig` will switch to `FALSE`.

**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**


**See Also**

`write.result`, `cohens.d`, `cor.cor`, `cor.cramer`, `multilevel.icc`, `cor.phi`, `multilevel.cor`, `na.auxiliary`, `size.cor`.

**Examples**

```r
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))

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

# 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-Stuart'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
# highlight statistically significant result at alpha = 0.05
cor.matrix(dat[, c("x", "y", "z"), sig = TRUE])

# Pearson product-moment correlation coefficient matrix using pairwise deletion
```
cor.phi

# highlight statistically significant result at alpha = 0.10
cor.matrix(dat[, c("x", "y", "z")], sig = TRUE, alpha = 0.10)

# 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")], na.omit = TRUE, print = "all")

# Pearson product-moment correlation coefficient matrix using listwise deletion,
# print sample size and significance values with Bonferroni correction
# results for group "a" and "b" separately
# Pearson product-moment correlation coefficient matrix using pairwise deletion,
# print sample size and significance values with Bonferroni correction
# results for group "a" and "b" separately
# Pearson product-moment correlation coefficient matrix using pairwise deletion,
# results for group "a" and "b" separately

## Not run:
# Write Results into a Excel file
result <- cor.matrix(dat[, c("x", "y", "z")], print = "all", output = FALSE)
write.result(result, "Correlation.xlsx")
## End(Not run)

---

**cor.phi**

<table>
<thead>
<tr>
<th>Phi Coefficient</th>
</tr>
</thead>
</table>

**Description**

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

**Usage**

```
cor.phi(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


See Also

cor.matrix, cohens.d, cor.cont, cor.cramer, cor.poly.

Examples

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

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

# Adjusted phi coefficient between x1 and x2
cor.phi(dat[, c("x1", "x2")], adjust = TRUE)
# Phi coefficient matrix between x1, x2, and x3
cor.phi(dat)

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

---

**cor.poly**  
*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**

```r
cor.poly(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` 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.

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).

Note
This function is based on the polychoric() function in the psych package by William Revelle.

Author(s)
William Revelle

References

See Also
cor.matrix, cor.cont, cor.cramer, cor.phi, cohen.s

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
cor.poly(dat)

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,
       write = 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 per-
                    centages ("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.
write               a character string for writing the results into a Excel file naming a file with or
                    without file extension '.xlsx', e.g., "Results.xlsx" or "Results".
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), 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>
write.result, freq, descript, multilevel.descript, na.descript.

References

John Wiley & Sons.
Examples

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

## Not run:
# Write Results into a Excel file
crosstab(dat[, c("x1", "x2")], print = "all", write = "Crosstab.xlsx")

result <- crosstab(dat[, c("x1", "x2")], print = "all", output = FALSE)
write.result(result, "Crosstab.xlsx")
## End(Not run)

---

**descript**

**Descriptive Statistics**

**Description**

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

**Usage**

```r
descript(x, 
  print = c("all", "n", "nNA", "pNA", "m", "se.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, write = 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), `se.m` (standard error of the 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 = c("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`.
- **write**: a character string for writing the results into a Excel file naming a file with or without file extension `.xlsx`, e.g., "Results.xlsx" or "Results".
- **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


See Also
ci.mean, ci.mean.diff, ci.median, ci.prop, ci.prop.diff, ci.var, ci.sd, freq, crosstab, multilevel.descript, na.descript.

Examples

```r
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,
```

# 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)

## Not run:
# Write Results into a Excel file
result <- descript(dat[, c("x1", "x2", "x3")], output = FALSE)
write.result(result, "Descript.xlsx")

# df.duplicated

## Extract Duplicated or Unique Rows

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


See Also

df.unique, 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**

```r
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 NAs 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 ddat by the variable id df.merge(adat, bdat, cdat, ddat, by = "id")

# Do not show output on the console
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.
df.rbind

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.

Value

Returns a single data frame

Note

This function is a copy of the rbind.fill() function in the plyr package by Hadley Wickham.

Author(s)

Hadley Wickham

References

Software, 40, 1-29. https://doi.org/10.18637/jss.v040.i01
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),
df.rename

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**

```r
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

```r
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 variable 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"))
```

### Description

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

### Usage

```r
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


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**


**Examples**

```r
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"),
                  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")), stringsAsFactors = FALSE)
# 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")), stringsAsFactors = FALSE)
# 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")), stringsAsFactors = FALSE)

---

**Description**

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

**Usage**

```r
etta.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`), 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


See Also

cohens.d, cor.cont, cor.matrix, cor.cramer, cor.phi

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

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, round = 3, exclude = 15, digits = 2,
     as.na = NULL, write = 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 ("perc"), 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.
freq

val.col logical: if TRUE, values are shown in the columns, variables in the rows.
round an integer value indicating the number of decimal places to be used for rounding numeric variables.
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. It is also possible to specify exclude = FALSE to include all variables in 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.
write a character string for writing the results into a Excel file naming a file with or without file extension '.xlsx', e.g., "Results.xlsx" or "Results".
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. By default, numeric variables are rounded to three digits before computing the frequency table.

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


See Also

write.result, 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)
# Confidence Intervals for the Indirect Effect

**Description**

This function computes confidence intervals for the indirect effect based on the asymptotic normal method, distribution of the product method and the Monte Carlo method. By default, the function uses the distribution of the product method for computing the two-sided 95% asymmetric confidence intervals for the indirect effect product of coefficient estimator $\hat{a}\hat{b}$.

**Usage**

```r
indirect(a, b, se.a, se.b, print = c("all", "asymp", "dop", "mc"),
         se = c("sobel", "aroian", "goodman"), nrep = 100000,
         alternative = c("two.sided", "less", "greater"),
         seed = NULL, conf.level = 0.95, digits = 3, check = TRUE,
         output = TRUE)
```

**Arguments**

- `a` a numeric value indicating the coefficient $a$, i.e., effect of $X$ on $M$.
- `b` a numeric value indicating the coefficient $b$, i.e., effect of $M$ on $Y$ adjusted for $X$.
- `se.a` a positive numeric value indicating the standard error of $a$.
- `se.b` a positive numeric value indicating the standard error of $b$.
- `print` a character string or character vector indicating which confidence intervals (CI) to show on the console, i.e. "all" for all CIs, "asymp" for the CI based on the asymptotic normal method, "dop" (default) for the CI based on the distribution of the product method, and "mc" for the CI based on the Monte Carlo method.
- `se` a character string indicating which standard error (SE) to compute for the asymptotic normal method, i.e., "sobel" for the approximate standard error by Sobel (1982) using the multivariate delta method based on a first order Taylor series approximation, "aroian" (default) for the exact standard error by Aroian (1947) based on a first and second order Taylor series approximation, and "goodman" for the unbiased standard error by Goodman (1960).
- `nrep` an integer value indicating the number of Monte Carlo repetitions.
- `alternative` a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".
seed

conf.level

digits

check

output

Details

In statistical mediation analysis (MacKinnon & Tofighi, 2013), the indirect effect refers to the effect
of the independent variable $X$ on the outcome variable $Y$ transmitted by the mediator variable $M$.
The magnitude of the indirect effect $ab$ is quantified by the product of the coefficients $a$ (i.e.,
effect of $X$ on $M$) and the coefficient $b$ (i.e., effect of $M$ on $Y$ adjusted for $X$). In practice,
researchers are often interested in confidence limit estimation for the indirect effect. This function
offers three different methods for computing the confidence interval for the product of coefficient
estimator $\hat{a}\hat{b}$:

(1) Asymptotic normal method

In the asymptotic normal method, the standard error for the product of the coefficient estimator $\hat{a}\hat{b}$
is computed which is used to create a symmetrical confidence interval based on the $z$-value of the
standard normal ($z$) distribution assuming that the indirect effect is normally distributed. Note that
the function provides three formulas for computing the standard error by specifying the argument
se:

"sobel" Approximate standard error by Sobel (1982) using the multivariate delta method based
on a first order Taylor series approximation:

$$\sqrt{(a^2\sigma_a^2 + b^2\sigma_b^2)}$$

"aroian" Exact standard error by Aroian (1947) based on a first and second order Taylor series
approximation:

$$\sqrt{(a^2\sigma_a^2 + b^2\sigma_b^2 + \sigma_a^2\sigma_b^2)}$$

"goodman" Unbiased standard error by Goodman (1960):

$$\sqrt{(a^2\sigma_a^2 + b^2\sigma_b^2 - \sigma_a^2\sigma_b^2)}$$

Note that the unbiased standard error is often negative and is hence undefined for zero or small
effects or small sample sizes.

The asymptotic normal method is known to have low statistical power because the distribution of
the product $\hat{a}\hat{b}$ is not normally distributed. (Kisbu-Sakarya, MacKinnon, & Miocevic, 2014). In
the null case, where both random variables have mean equal to zero, the distribution is symmetric
with kurtosis of six. When the product of the means of the two random variables is nonzero, the
distribution is skewed (up to a maximum value of $\pm 1.5$) and has a excess kurtosis (up to a maximum
value of 6). However, the product approaches a normal distribution as one or both of the ratios of
the means to standard errors of each random variable get large in absolute value (MacKinnon,
Lockwood & Williams, 2004).
(2) Distribution of the product method

The distribution of the product method (MacKinnon et al., 2002) relies on an analytical approximation of the distribution of the product of two normally distributed variables. The method uses the standardized $a$ and $b$ coefficients to compute $ab$ and then uses the critical values for the distribution of the product (Meeker, Cornwell, & Aroian, 1981) to create asymmetric confidence intervals. The distribution of the product approaches the gamma distribution (Aroian, 1947). The analytical solution for the distribution of the product is provided by the Bessel function used to the solution of differential equations and is approximately proportional to the Bessel function of the second kind with a purely imaginary argument (Craig, 1936).

(3) Monte Carlo method

The Monte Carlo (MC) method (MacKinnon et al., 2004) relies on the assumption that the parameters $a$ and $b$ have a joint normal sampling distribution. Based on the parametric assumption, a sampling distribution of the product $ab$ using random samples with population values equal to the sample estimates $\hat{a}$, $\hat{b}$, $\hat{\sigma}_a$, and $\hat{\sigma}_b$ is generated. Percentiles of the sampling distribution are identified to serve as limits for a $100(1 - \alpha)$% asymmetric confidence interval about the sample $\hat{a}\hat{b}$ (Preacher & Selig, 2012). Note that parametric assumptions are invoked for $\hat{a}$ and $\hat{b}$, but no parametric assumptions are made about the distribution of $\hat{a}\hat{b}$.

Value

Returns an object of class \texttt{misti.object}, which is a list with following entries: function call (\texttt{call}), type of analysis (\texttt{type}), list with the input specified in \texttt{a b, se.a, and se.b} (\texttt{data}), specification of function arguments (\texttt{args}), and a list with the result tables (\texttt{result}).

Note

The function was adapted from the \texttt{medci()} function in the \texttt{RMediation} package by Davood Tofighi and David P. MacKinnon (2016).

Author(s)

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

References


See Also

```
multilevel.indirect
```

Examples

```
# Distribution of the Product Method
indirect(a = 0.35, b = 0.27, se.a = 0.12, se.b = 0.18)

# Monte Carlo Method
indirect(a = 0.35, b = 0.27, se.a = 0.12, se.b = 0.18, print = "mc")

# Asymptotic Normal Method
indirect(a = 0.35, b = 0.27, se.a = 0.12, se.b = 0.18, print = "asymp")
```

---

`item.alpha`  
*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

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

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` (default), 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.
- **write**: a character string for writing the results into a Excel file naming a file with or without file extension `.xlsx`, e.g., "Results.xlsx" or "Results".
- **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 been 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 MBESS 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

See Also
write.result, item.cfa, item.omega, item.reverse, item.scores

Examples
```r
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
item.alpha(dat)

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

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

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

Confirmatory Factor Analysis

Description

This function is a wrapper function for conducting confirmatory factor analysis with continuous and/or ordered-categorical indicators by calling the cfa function in the R package lavaan.

Usage

Arguments

x

a matrix or data frame. If `model = NULL`, confirmatory factor analysis based on a measurement model with one factor labeled `f` comprising all variables in the matrix or data frame is conducted. Note that the cluster variable is excluded from `x` when specifying `cluster`. If `model` is specified, the matrix or data frame needs to contain all variables used in the argument `model` and the cluster variable when specifying `cluster`.

model

a character vector specifying a measurement model with one factor, or a list of character vectors for specifying a measurement model with more than one factor, e.g., `model = c("x1", "x2", "x3", "x4")` for specifying a measurement model with one factor labeled `f` comprising four indicators, or `model = list(factor1 = c("x1", "x2", "x3", "x4"), factor2 = c("x5", "x6", "x7", "x8"))` for specifying a measurement model with two latent factors labeled `factor1` and `factor2` each comprising four indicators. Note that the name of each list element is used to label factors, i.e., all list elements need to be named, otherwise factors are labeled with "f1", "f2", "f3" and so on.

rescov

a character vector or a list of character vectors for specifying residual covariances, e.g. `rescov = c("x1", "x2")` for specifying a residual covariance between items `x1` and `x2`, or `rescov = list(c("x1", "x2"), c("x3", "x4"))` for specifying residual covariances between items `x1` and `x2`, and items `x3` and `x4`.

hierarch

logical: if `TRUE`, a second-order factor model is specified given at least three first-order factors were specified in `model`. Note that it is not possible to specify more than one second-order factor.

meanstructure

logical: if `TRUE` (default), intercept/means of observed variables means of latent variables will be added to the model. Note that `meanstructure = FALSE` is only applicable when the `missing` is listwise, pairwise, or doubly-robust.

ident

a character string indicating the method used for identifying and scaling latent variables, i.e., "marker" for the marker variable method fixing the first factor loading of each latent variable to 1, "var" for the fixed variance method fixing the variance of each latent variable to 1, or "effect" for the effects-coding method using equality constraints so that the average of the factor loading for each latent variable equals 1. By default, fixed variance method is used when `hierarch = FALSE`, whereas marker variable method is used when `hierarch = TRUE`.

parameterization

a character string indicating the method used for identifying and scaling latent variables when indicators are ordered, i.e., "delta" (default) for delta parameterization and "theta" for theta parameterization.

ordered

if `NULL` (default), all indicators of the measurement model are treated as continuous. If `TRUE`, all indicators of the measurement model are treated as ordered (ordinal). Alternatively, a character vector indicating which variables to treat as ordered (ordinal) variables can be specified.

cluster

either a character string indicating the variable name of the cluster variable in 'x' or a vector representing the nested grouping structure (i.e., group or cluster variable) for computing cluster-robust standard errors. Note that cluster-robust
standard errors are not available when treating indicators of the measurement model as ordered (ordinal).

**estimator** a character string indicating the estimator to be used (see 'Details'). By default, "MLR" is used for CFA models with continuous indicators (i.e., `ordered = FALSE`) and "WLSMV" is used for CFA model with ordered-categorical indicators (i.e., `ordered = TRUE`).

**missing** a character string indicating how to deal with missing data, i.e., "listwise" for listwise deletion, "pairwise" for pairwise deletion, "fiml" for full information maximum likelihood method, two.stage for two-stage maximum likelihood method, robust.two.stage for robust two-stage maximum likelihood method, and doubly-robust for doubly-robust method (see 'Details'). By default, "fiml" is used for CFA models with continuous indicators which are estimated by using `estimator = "MLR"`, and "pairwise" for CFA models with ordered-categorical indicators which are estimated by using `estimator = "pairwise"` by default.

**print** a character string or character vector indicating which results to show on the console, i.e., "all" for all results, "summary" for a summary of the specification of the estimation method and missing data handling in lavaan, "coverage" for the variance-covariance coverage of the data, "descript" for descriptive statistics, "fit" for model fit, "est" for parameter estimates, and "modind" for modification indices. By default, a summary of the specification, model fit, and parameter estimates are printed.

**min.value** numeric value to filter modification indices and only show modifications with a modification index value equal or higher than this minimum value. By default, modification indices equal or higher 10 is printed.

**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. Note that `as.na()` function is only applied to `x` but not to `cluster`.

**write** a character string for writing the results into a Excel file naming a file with or without file extension '.xlsx', e.g., "Results.xlsx" or "Results".

**check** logical: if TRUE, argument specification is checked.

**output** logical: if TRUE, output is shown.

**Details**

**Estimator** The R package *lavaan* provides seven estimators that affect the estimation, namely "ML", "GLS", "WLS", "DWLS", "ULS", "DLS", and "PML". All other options for the argument `estimator` combine these estimators with various standard error and chi-square test statistic computation. Note that the estimators also differ in how missing values can be dealt with (e.g., listwise deletion, pairwise deletion, or full information maximum likelihood, FIML).

- "ML": Maximum likelihood with conventional standard errors and conventional test statistic. For both complete and incomplete data using pairwise deletion or FIML.
• "MLM": Maximum likelihood parameter estimates with conventional robust standard errors and a Satorra-Bentler scaled test statistic that are robust to non-normality. For complete data only.
• "MLMV": Maximum likelihood parameter estimates with conventional robust standard errors and a mean and a variance adjusted test statistic using a scale-shifted approach that are robust to non-normality. For complete data only.
• "MLMVS": Maximum likelihood parameter estimates with conventional robust standard errors and a mean and a variance adjusted test statistic using the Satterthwaite approach that are robust to non-normality. For complete data only.
• "MLF": Maximum likelihood parameter estimates with standard errors approximated by first-order derivatives and conventional test statistic. For both complete and incomplete data using pairwise deletion or FIML.
• "MLR": Maximum likelihood parameter estimates with Huber-White robust standard errors a test statistic which is asymptotically equivalent to the Yuan-Bentler T2* test statistic that are robust to non-normality and non-independence of observed when specifying a cluster variable using the argument cluster. For both complete and incomplete data using pairwise deletion or FIML.
• "GLS": Generalized least squares parameter estimates with conventional standard errors and conventional test statistic that uses a normal-theory based weight matrix. For complete data only. and conventional chi-square test. For both complete and incomplete data.
• "WLS": Weighted least squares parameter estimates (sometimes called ADF estimation) with conventional standard errors and conventional test statistic that uses a full weight matrix. For complete data only.
• "DWLS": Diagonally weighted least squares parameter estimates which uses the diagonal of the weight matrix for estimation with conventional standard errors and conventional test statistic. For both complete and incomplete data using pairwise deletion.
• "WLSM": Diagonally weighted least squares parameter estimates which uses the diagonal of the weight matrix for estimation, but uses the full weight matrix for computing the conventional robust standard errors and a Satorra-Bentler scaled test statistic. For both complete and incomplete data using pairwise deletion.
• "WLSMV": Diagonally weighted least squares parameter estimates which uses the diagonal of the weight matrix for estimation, but uses the full weight matrix for computing the conventional robust standard errors and a mean and a variance adjusted test statistic using a scale-shifted approach. For both complete and incomplete data using pairwise deletion.
• "ULS": Unweighted least squares parameter estimates with conventional standard errors and conventional test statistic. For both complete and incomplete data using pairwise deletion.
• "ULSM": Unweighted least squares parameter estimates with conventional robust standard errors and a Satorra-Bentler scaled test statistic. For both complete and incomplete data using pairwise deletion.
• "ULSMV": Unweighted least squares parameter estimates with conventional robust standard errors and a mean and a variance adjusted test statistic using a scale-shifted approach. For both complete and incomplete data using pairwise deletion.
• "DLS": Distributionally-weighted least squares parameter estimates with conventional robust standard errors and a Satorra-Bentler scaled test statistic. For complete data only.
• "PML": Pairwise maximum likelihood parameter estimates with Huber-White robust standard errors and a mean and a variance adjusted test statistic using the Satterthwaite ap-
Missing Data The R package lavaan provides six methods for dealing with missing data:

- "listwise": Listwise deletion, i.e., all cases with missing values are removed from the data before conducting the analysis. This is only valid if the data are missing completely at random (MCAR).
- "pairwise": Pairwise deletion, i.e., each element of a variance-covariance matrix is computed using cases that have data needed for estimating that element. This is only valid if the data are missing completely at random (MCAR).
- "fiml": Full information maximum likelihood (FIML) method, i.e., likelihood is computed case by case using all available data from that case. FIML method is only applicable for following estimators: "ML", "MLF", and "MLR".
- "two.stage": Two-stage maximum likelihood estimation, i.e., sample statistics is estimated using EM algorithm in the first step. Then, these estimated sample statistics are used as input for a regular analysis. Standard errors and test statistics are adjusted correctly to reflect the two-step procedure. Two-stage method is only applicable for following estimators: "ML", "MLF", and "MLR".
- "robust.two.stage": Robust two-stage maximum likelihood estimation, i.e., two-stage maximum likelihood estimation with standard errors and a test statistic that are robust against non-normality. Robust two-stage method is only applicable for following estimators: "ML", "MLF", and "MLR".
- "doubly.robust": Doubly-robust method only applicable for pairwise maximum likelihood estimation (i.e., estimator = "PML").

Convergence and model identification checks In line with the R package lavaan, this function provides several checks for model convergence and model identification:

- Degrees of freedom: An error message is printed if the number of degrees of freedom is negative, i.e., the model is not identified.
- Model convergence: An error message is printed if the optimizer has not converged, i.e., results are most likely unreliable.
- Standard errors: An error message is printed if the standard errors could not be computed, i.e., the model might not be identified.
- Variance-covariance matrix of the estimated parameters: A warning message is printed if the variance-covariance matrix of the estimated parameters is not positive definite, i.e., the smallest eigenvalue of the matrix is smaller than zero or very close to zero.
- Negative variances of observed variables: A warning message is printed if the estimated variances of the observed variables are negative.
- Variance-covariance matrix of observed variables: A warning message is printed if the estimated variance-covariance matrix of the observed variables is not positive definite, i.e., the smallest eigenvalue of the matrix is smaller than zero or very close to zero.
- Negative variances of latent variables: A warning message is printed if the estimated variances of the latent variables are negative.
- Variance-covariance matrix of latent variables: A warning message is printed if the estimated variance-covariance matrix of the latent variables is not positive definite, i.e., the smallest eigenvalue of the matrix is smaller than zero or very close to zero.

Note that unlike the R package lavaan, the item.cfa function does not provide any results when the degrees of freedom is negative, the model has not converged, or standard errors could not be computed.
Model Fit The item.cfa function provides the chi-square test, incremental fit indices (i.e., CFI and TLI), and absolute fit indices (i.e., RMSEA, and SRMR) to evaluate overall model fit. However, different versions of the CFI, TLI, and RMSEA are provided depending on the estimator. Unlike the R package lavaan, the different versions are labeled with Standard, Ad hoc, and Robust in the output:

- "Standard": CFI, TLI, and RMSEA without any non-normality corrections. These fit measures based on the normal theory maximum likelihood test statistic are sensitive to deviations from multivariate normality of endogenous variables. Simulation studies by Brosseau-Liard et al. (2012), and Brosseau-Liard and Savalei (2014) showed that the uncorrected fit indices are affected by non-normality, especially at small and medium sample sizes (e.g., n < 500).

- "Ad hoc": Population-corrected robust CFI, TLI, and RMSEA with ad hoc non-normality corrections that simply replace the maximum likelihood test statistic with a robust test statistic (e.g., mean-adjusted chi-square). These fit indices change the population value being estimated depending on the degree of non-normality present in the data. Brosseau-Liard et al. (2012) demonstrated that the ad hoc corrected RMSEA increasingly accepts poorly fitting models as non-normality in the data increases, while the effect of the ad hoc correction on the CFI and TLI is less predictable with non-normality making fit appear worse, better, or nearly unchanged (Brosseau-Liard & Savalei, 2014).

- "Robust": Sample-corrected robust CFI, TLI, and RMSEA with non-normality corrections based on formula provided by Li and Bentler (2006) and Brosseau-Liard and Savalei (2014). These fit indices do not change the population value being estimated and can be interpreted the same way as the uncorrected fit indices when the data would have been normal.

In conclusion, the use of sample-corrected fit indices (Robust) instead of population-corrected fit indices (Ad hoc) is recommended. Note that when sample size is very small (e.g., n < 200), non-normality correction does not appear to adjust fit indices sufficiently to counteract the effect of non-normality (Brosseau-Liard & Savalei, 2014).

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 (x), specification of function arguments (args), specified model (model), fitted lavaan object (mod.fit), results of the convergence and model identification check (check), and a list with results (result).

Note

The function uses the functions cfa, lavInspect, lavTech, modindices, parameterEstimates, and standardizedsolution provided in the R package lavaan by Yves Rosseel (2012).

Author(s)

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

References

item.cfa

https://doi.org/10.1080/00273171.2014.933697


See Also

*item.alpha, item.omega, item.scores*

Examples

```r
## Not run:

Load data set "HolzingerSwineford1939" in the lavaan package
data("HolzingerSwineford1939", package = "lavaan")

#---------------------------
# Measurement model with one factor

# Specification using the argument 'x'
item.cfa(HolzingerSwineford1939[, c("x1", "x2", "x3")])

# Alternative specification using the argument 'model'
item.cfa(HolzingerSwineford1939, model = c("x1", "x2", "x3"))

# Alternative specification using the argument 'model'
item.cfa(HolzingerSwineford1939, model = list(visual = c("x1", "x2", "x3")))

#---------------------------
# Measurement model with three factors

# Specification using the argument 'model'
item.cfa(HolzingerSwineford1939,
  model = list(visual = c("x1", "x2", "x3"),
               textual = c("x4", "x5", "x6"),
               speed = c("x7", "x8", "x9")))

#---------------------------
# Residual covariances

# One residual covariance
item.cfa(HolzingerSwineford1939,
  model = list(visual = c("x1", "x2", "x3"),
               textual = c("x4", "x5", "x6"),
               speed = c("x7", "x8", "x9"),
               rescov = c("x1", "x2")))
```
# Two residual covariances
item.cfa(HolzingerSwineford1939,
    model = list(visual = c("x1", "x2", "x3"),
               textual = c("x4", "x5", "x6"),
               speed = c("x7", "x8", "x9"),
               rescov = list(c("x1", "x2"), c("x4", "x5")))

# Second-order factor model based on three first-order factors
item.cfa(HolzingerSwineford1939,
    model = list(visual = c("x1", "x2", "x3"),
               textual = c("x4", "x5", "x6"),
               speed = c("x7", "x8", "x9"),
               hierarch = TRUE)

# Measurement model with ordered-categorical indicators
item.cfa(round(HolzingerSwineford1939[, c("x4", "x5", "x6")]), ordered = TRUE)

# Cluster-robust standard errors
# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

# Specification using a variable in 'x'
item.cfa(Demo.twolevel[, c("y4", "y5", "y6", "cluster")], cluster = "cluster")

# Specification of the cluster variable in 'cluster'
item.cfa(Demo.twolevel[, c("y4", "y5", "y6")], cluster = Demo.twolevel$cluster)

# Specification using a variable in 'x'
item.cfa(Demo.twolevel, model = c("y4", "y5", "y6"), cluster = "cluster")

# Specification of the cluster variable in 'cluster'
item.cfa(Demo.twolevel, model = c("y4", "y5", "y6"), cluster = Demo.twolevel$cluster)

# Print argument
# Request all results
item.cfa(HolzingerSwineford1939[, c("x1", "x2", "x3")], print = "all")

# Request modification indices with value equal or higher than 5
item.cfa(HolzingerSwineford1939[, c("x1", "x2", "x3", "x4")],
    print = "modind", min.value = 5)

# lavaan summary of the estimated model
mod <- item.cfa(HolzingerSwineford1939[, c("x1", "x2", "x3")], output = FALSE)
```r
lavaan::summary(mod$mod.fit, standardized = TRUE, fit.measures = TRUE)
#---------------------------
# Write Results into a Excel file
item.cfa(HolzingerSwineford1939[, c("x1", "x2", "x3")], write = "CFA.xlsx")
result <- item.cfa(HolzingerSwineford1939[, c("x1", "x2", "x3")], output = FALSE)
write.result(result, "CFA.xlsx")
## End(Not run)

### 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

```r
item.omega(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, write = 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 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.
write a character string for writing the results into a Excel file naming a file with or without file extension '.xlsx', e.g., "Results.xlsx" or "Results".
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.

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 MBESS 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), fitted lavaan object (mod.fit), and list with results (result).

Note

Computation of the hierarchical and categorical omega is based on the ci.reliability() function in the MBESS package by Ken Kelley (2019).

Author(s)

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

References


See Also

write.result, item.alpha, item.cfa, item.reverse, item.scores

Examples

```r
## Not run:
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
item.omega(dat)

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

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

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

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

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

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

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

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

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

# Write Results into a Excel file
item.omega(dat, write = "Omega.xlsx")

result <- item.omega(dat, output = FALSE)
write.result(result, "Omega.xlsx")
## End(Not run)

<table>
<thead>
<tr>
<th>item.reverse</th>
<th>Reverse Code Scale Item</th>
</tr>
</thead>
</table>

**Description**

This function reverse codes an inverted item, i.e., item that is negatively worded.

**Usage**

```
item.reverse(x, min = NULL, max = NULL, keep = NULL, as.na = NULL, table = FALSE,
             check = TRUE)
```

**Arguments**

- `x` a numeric vector.
- `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>
item.scores

References


See Also

item.alpha, item.omega, rec, item.scores

Examples

dat <- data.frame(item1 = c(1, 5, 3, 1, 4, 4, 1, 5),
                  item2 = c(1, 1.3, 1.7, 2, 2.7, 3.3, 4.7, 5),
                  item3 = c(4, 2, 4, 5, 1, 3, 5, -99))

# Reverse code item1
dat$item1r <- item.reverse(dat$item1, min = 1, max = 5)

# Reverse code item2
dat$item2r <- item.reverse(dat$item2, min = 1, max = 5)

# Reverse code item3 while keeping the value -99
dat$item3r <- item.reverse(dat$item3, min = 1, max = 5, keep = -99)

# Reverse code item3 while keeping the value -99 and check recoding
dat$item3r <- item.reverse(dat$item3, min = 1, max = 5, keep = -99, table = TRUE)

item.scores

Compute Scale Scores

Description

This function computes (prorated) scale scores by averaging the (available) items that measure a single construct by default.

Usage

item.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.
item.scores

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


**See Also**

`cluster.scores, item.alpha, item.cfa, item.omega`,

**Examples**

```r
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
item.scores(dat)

# Prorated standard deviation scale scores
item.scores(dat, fun = "sd")

# Sum scale scores without proration
item.scores(dat, fun = "sum", prorated = FALSE)

# Prorated mean scale scores,
# minimum proportion of available item responses = 0.8
item.scores(dat, p.avail = 0.8)

# Prorated mean scale scores,
# minimum number of available item responses = 3
item.scores(dat, n.avail = 3)
```

---

**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


See Also

- skewness

Examples

```r
# 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)
```

multilevel.cor  
*Within-Group and Between-Group Correlation Matrix*

Description

This function is a wrapper function for computing the within-group and between-group correlation matrix by calling the sem function in the R package lavaan and provides standard errors, z test statistics, and significance values (p-values) for testing the hypothesis H0: \( \rho = 0 \) for all pairs of variables within and between groups.
Usage

```r
multilevel.cor(x, cluster, within = NULL, between = NULL, estimator = c("ML", "MLR"),
missing = c("listwise", "fiml"), sig = FALSE, alpha = 0.05,
print = c("all", "cor", "se", "stat", "p"), split = FALSE,
tri = c("both", "lower", "upper"), tri.lower = TRUE,
p.adj = c("none", "bonferroni", "holm", "hochberg", "hommel",
"BH", "BY", "fdr"), digits = 2, p.digits = 3,
as.na = NULL, write = NULL, check = TRUE, output = TRUE)
```

Arguments

- **x**: a matrix or data frame.
- **cluster**: a vector representing the nested grouping structure (i.e., group or cluster variable).
- **within**: a character vector representing variables that are measured on the within level and modeled only on the within level. Variables not mentioned in within or between are measured on the within level and will be modeled on both the within and between level.
- **between**: a character vector representing variables that are measured on the between level and modeled only on the between level. Variables not mentioned in within or between are measured on the within level and will be modeled on both the within and between level.
- **estimator**: a character string indicating the estimator to be used: "ML" (default) for maximum likelihood with conventional standard errors and "MLR" for maximum likelihood with Huber-White robust standard errors. Note that by default, full information maximum likelihood (FIML) method is used to deal with missing data when using "ML" (missing = "fiml"), whereas incomplete cases are removed listwise (i.e., missing = "listwise") when using "MLR".
- **missing**: a character string indicating how to deal with missing data, i.e., "listwise" for listwise deletion or "fiml" (default) for full information maximum likelihood (FIML) method. Note that FIML method is only available when estimator = "ML". Note that it takes longer to estimate the model when using FIML and using FIML might cause issues in model convergence, these issues might be resolved by switching to listwise deletion.
- **sig**: logical: if TRUE, statistically significant correlation coefficients are shown in boldface on the console.
- **alpha**: a numeric value between 0 and 1 indicating the significance level at which correlation coefficients are printed boldface when sig = TRUE.
- **print**: a character string or character vector indicating which results to show on the console, i.e. "all" for all results, "cor" for correlation coefficients, "se" for standard errors, "stat" for z test statistics, and "p" for p-values.
- **split**: logical: if TRUE, output table is split in within-group and between-group correlation matrix.
- **tri**: a character string indicating which triangular of the matrix to show on the console when split = TRUE, i.e., both for upper and upper for the upper triangular.
trilower logical: if TRUE (default) and split = FALSE (default), within-group correlations are shown in the lower triangular and between-group correlation are shown in 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. Note that as.na() function is only applied to x but not to cluster.

write a character string for writing the results into a Excel file naming a file with or without file extension `.xlsx`, e.g., "Results.xlsx" or "Results".

check logical: if TRUE, argument specification is checked.

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

Details

The specification of the within-group and between-group variables is in line with the syntax in Mplus. That is, the within argument is used to identify the variables in the matrix or data frame specified in x that are measured on the individual level and modeled only on the within level. They are specified to have no variance in the between part of the model. The between argument is used to identify the variables in the matrix or data frame specified in x that are measured on the cluster level and modeled only on the between level. Variables not mentioned in the arguments within or between are measured on the individual level and will be modeled on both the within and between level.

The function uses maximum likelihood estimation with conventional standard errors (estimator = "ML") which are not robust against non-normality and full information maximum likelihood (FIML) method (missing = "fiml") to deal with missing data by default. FIML method cannot be used when within-group variables have no variance within some clusters. In this cases, the function will switch to listwise deletion. Note that the current lavaan version 0.6-11 supports FIML method only for maximum likelihood estimation with conventional standard errors (estimator = "ML") in multilevel models. Maximum likelihood estimation with Huber-White robust standard errors (estimator = "MLR") uses listwise deletion to deal with missing data. When using FIML method there might be issues in model convergence, which might be resolved by switching to listwise deletion (missing = "listwise").

The lavaan package uses a quasi-Newton optimization method ("nlmnb") by default. If the optimizer does not converge, model estimation will switch to the Expectation Maximization (EM) algorithm.

Statistically significant correlation coefficients can be shown in boldface on the console when specifying sig = TRUE. However, this option is not supported when using R Markdown, i.e., the argument sig will switch to FALSE.

Adjustment method for multiple testing when specifying the argument p.adj is applied to the within-group and between-group correlation matrix separately.
**multilevel.cor**

### 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`), fitted lavaan object (`mod.fit`), and list with results (`result`).

### Note

The function uses the functions `sem`, `lavInspect`, `lavMatrixRepresentation`, `lavTech`, `parameterEstimates`, and `standardizedsolution` provided in the R package `lavaan` by Yves Rosseel (2012).

### Author(s)

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

### References


### See Also

`write.result`, `multilevel.descript`, `multilevel.icc`, `cluster.scores`

### Examples

```r
## Not run:

# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

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

# All variables modeled on both the within and between level
multilevel.cor(Demo.twolevel[, c("y1", "y2", "y3")],
               cluster = Demo.twolevel$cluster)

# Highlight statistically significant result at alpha = 0.05
multilevel.cor(Demo.twolevel[, c("y1", "y2", "y3")], sig = TRUE,
               cluster = Demo.twolevel$cluster)

# Split output table in within-group and between-group correlation matrix.
multilevel.cor(Demo.twolevel[, c("y1", "y2", "y3")],
               cluster = Demo.twolevel$cluster, split = TRUE)

# Print correlation coefficients, standard errors, z test statistics,
# and p-values
multilevel.cor(Demo.twolevel[, c("y1", "y2", "y3")],
               cluster = Demo.twolevel$cluster, print = "all")

# Print correlation coefficients and p-values
```

multilevel.cor(Demo.twolevel[, c("y1", "y2", "y3")],
   cluster = Demo.twolevel$cluster, print = c("cor", "p"),
p.adj = "bonferroni")

#---------------------------
# Variables "y1", "y2", and "y2" modeled on both the within and between level
# Variables "w1" and "w2" modeled on the cluster level
multilevel.cor(Demo.twolevel[, c("y1", "y2", "y3", "w1", "w2")],
   cluster = Demo.twolevel$cluster,
   between = c("w1", "w2"))

#---------------------------
# Variables "y1", "y2", and "y2" modeled only on the within level
# Variables "w1" and "w2" modeled on the cluster level
multilevel.cor(Demo.twolevel[, c("y1", "y2", "y3", "w1", "w2")],
   cluster = Demo.twolevel$cluster,
   within = c("y1", "y2", "y3"), between = c("w1", "w2"))

# Summary of the multilevel model used to compute the within-group
# and between-group correlation matrix
mod <- multilevel.cor(Demo.twolevel[, c("y1", "y2", "y3")],
   cluster = Demo.twolevel$cluster, output = FALSE)
lavaan::summary(mod$mod.fit, standardized = TRUE)

# Write Results into a Excel file
multilevel.cor(Demo.twolevel[, c("y1", "y2", "y3")],
   cluster = Demo.twolevel$cluster,
   write = "Multilevel_Correlation.xlsx")

result <- multilevel.cor(Demo.twolevel[, c("y1", "y2", "y3")],
   cluster = Demo.twolevel$cluster, output = FALSE)
write.result(result, "Multilevel_Correlation.xlsx")

## End(Not run)

multilevel.descript  Multilevel Descriptive Statistics

Description

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

Usage

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

x  a vector, matrix or data frame.
cluster  a vector representing the nested grouping structure (i.e., group or cluster variable).
method  a character string indicating the method used to estimate intraclass correlation coefficients, i.e., "aov" ICC estimated using the aov function, "lme4" (default) 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 (default), 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 cluster.
write  a character string for writing the results into a Excel file naming a file with or without file extension '.xlsx', e.g., "Results.xlsx" or "Results".
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


See Also

multilevel.cor, multilevel.icc, multilevel.indirect, multilevel.r2, write.result,
Examples

dat <- data.frame(id = c(1, 2, 3, 4, 5, 6, 7, 8, 9),
                  cluster = 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$x1, cluster = dat$cluster)

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

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

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

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

## Not run:
# Write Results into a Excel file
multilevel.descript(dat[, c("x1", "x2", "x3")], cluster = dat$cluster,
                    write = "Multilevel_Descript.xlsx")

result <- multilevel.descript(dat[, c("x1", "x2", "x3")], cluster = dat$cluster,
                               write = TRUE)
write.result(result, "Multilevel_Descript.xlsx")
## End(Not run)

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

Arguments

x  a vector, matrix or data frame.

cluster  a vector representing the nested grouping structure (i.e., group or cluster 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" ICC estimated using the aov function, method = "lme4" (default) ICC estimated using the lmer function in the lme4 package, method = "nlme" ICC estimated using the lme function in the nlme package. Note that if the lme4 package is not installed, method = "aov" will be used.

REML  logical: if TRUE (default), 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. Note that as.na() function is only applied to x but not to cluster.

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


See Also

multilevel.cor, multilevel.descript
Examples

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

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

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

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

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

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

---

multilevel.indirect  Confidence Interval for the Indirect Effect in a 1-1-1 Multilevel Mediation Model

Description

This function computes the confidence interval for the indirect effect in a 1-1-1 multilevel mediation model with random slopes based on the Monte Carlo method.

Usage

```r
multilevel.indirect(a, b, se.a, se.b, cov.ab = 0, cov.rand, se.cov.rand,
                      nrep = 100000, alternative = c("two.sided", "less", "greater"),
                      seed = NULL, conf.level = 0.95, digits = 3, check = TRUE, output = TRUE)
```

Arguments

- **a**: a numeric value indicating the coefficient \(a\), i.e., average effect of \(X\) on \(M\) on the cluster or between-group level.
- **b**: a numeric value indicating the coefficient \(b\), i.e., average effect of \(M\) on \(Y\) adjusted for \(X\) on the cluster or between-group level.
- **se.a**: a positive numeric value indicating the standard error of \(a\).
- **se.b**: a positive numeric value indicating the standard error of \(b\).
cov_ab a positive numeric value indicating the covariance between a and b.
cov.rand a positive numeric value indicating the covariance between the random slopes for a and b.
se.cov.rand a positive numeric value indicating the standard error of the covariance between the random slopes for a and b.
nrep an integer value indicating the number of Monte Carlo repetitions.
alternative a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".
seed a numeric value specifying the seed of the random number generator when using the Monte Carlo method.
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
check logical: if TRUE, argument specification is checked.
output logical: if TRUE, output is shown on the console.

Details

In statistical mediation analysis (MacKinnon & Tofighi, 2013), the indirect effect refers to the effect of the independent variable X on the outcome variable Y transmitted by the mediator variable M. The magnitude of the indirect effect ab is quantified by the product of the coefficient a (i.e., effect of X on M) and the coefficient b (i.e., effect of M on Y adjusted for X). However, mediation in the context of a 1-1-1 multilevel model where variables X, M, and Y are measured at level 1, the coefficients a and b can vary across level-2 units (i.e., random slope). As a result, a and b may co-vary so that the estimate of the indirect effect is no longer simply the product of the coefficients ˆab, but ˆab + τ_{a,b}, where τ_{a,b} is the level-2 covariance between the random slopes a and b. The covariance term needs to be added to ˆab only when random slopes are estimated for both a and b. Otherwise, the simple product is sufficient to quantify the indirect effect, and the indirect function can be used instead.

In practice, researchers are often interested in confidence limit estimation for the indirect effect. There are several methods for computing a confidence interval for the indirect effect in a single-level mediation model (see indirect function). The Monte Carlo (MC) method (MacKinnon et al., 2004) is a promising method in single-level mediation model which was also adapted to the multilevel mediation model (Bauer, Preacher & Gil, 2006). This method requires seven pieces of information available from the results of a multilevel mediation model:

a Coefficient a, i.e., average effect of X on M on the cluster or between-group level. In Mplus, Estimate of the random slope a under Means at the Between Level.
b Coefficient a, i.e., average effect of M on Y on the cluster or between-group level. In Mplus, Estimate of the random slope b under Means at the Between Level.
sea Standard error of a. In Mplus, S.E. of the random slope a under Means at the Between Level.
sea Standard error of a. In Mplus, S.E. of the random slope a under Means at the Between Level.
cov_ab Covariance between a and b. In Mplus, the estimated covariance matrix for the parameter estimates (i.e., asymptotic covariance matrix) need to be requested by specifying TECH3 along
with TECH1 in the OUTPUT section. In the TECHNICAL 1 OUTPUT under PARAMETER SPECIFICATION FOR BETWEEN, the numbers of the parameter for the coefficients \(a\) and \(b\) need to be identified under ALPHA to look up cov.av in the corresponding row and column in the TECHNICAL 3 OUTPUT under ESTIMATED COVARIANCE MATRIX FOR PARAMETER ESTIMATES.

**cov.rand** Covariance between the random slopes for \(a\) and \(b\). In Mplus, estimate of the covariance \(a\) WITH \(b\) at the Between Level.

**se.cov.rand** Standard error of the covariance between the random slopes for \(a\) and \(b\). In Mplus, S.E. of the covariance \(a\) WITH \(b\) at the Between Level.

Note that all pieces of information except cov.ab can be looked up in the standard result output of the multilevel mediation model. In order to specify cov.ab, the covariance matrix for the parameter estimates (i.e., asymptotic covariance matrix) is required. In practice, cov.ab will oftentimes be very small so that cov.ab may be set to 0 (i.e., default value) with negligible impact on the results.

**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 `a`, `b`, `se.a`, `se.b`, `cov.ab`, `cov.rand`, and `se.cov.rand` (data), specification of function arguments (`args`), and a list with the result of the Monte Carlo method and the result table (`result`).

**Note**

The function was adapted from the interactive web tool by Preacher and Selig (2010).

**Author(s)**

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

**References**


**See Also**

indirect
Examples

# Confidence Interval for the Indirect Effect
multilevel.indirect(a = 0.25, b = 0.20, se.a = 0.11, se.b = 0.13,
                cov.ab = 0.01, cov.rand = 0.40, se.cov.rand = 0.02)

# Save results of the Monte Carlo method
ab <- multilevel.indirect(a = 0.25, b = 0.20, se.a = 0.11, se.b = 0.13,
                cov.ab = 0.01, cov.rand = 0.40, se.cov.rand = 0.02, 
                output = FALSE)$result$ab

# Histogram of the distribution of the indirect effect
hist(ab)

multilevel.r2

R-Squared Measures for Multilevel and Linear Mixed Effects Models

Description

This function computes R-squared measures by Raudenbush and Bryk (2002), Snijders and Bosker (1994), Nakagawa and Schielzeth (2013) as extended by Johnson (2014), and Rights and Sterba (2019) for multilevel and linear mixed effects models estimated by using the lmer() function in the package lme4 or lme() function in the package nlme.

Usage

multilevel.r2(model, print = c("all", "RB", "SB", "NS", "RS"), digits = 3,
              plot = FALSE, gray = FALSE, start = 0.15, end = 0.85,
              color = c("#D55E00", "#0072B2", "#CC79A7", "#009E73", 
                        "#E69F00"),
              check = TRUE, output = TRUE)

Arguments

model a fitted model of class "lmerMod" from the lme4 package or "lme" from the nlme package.

print a character vector indicating which R-squared measures to be printed on the console, i.e., RB for measures from Raudenbush and Bryk (2002), SB for measures from Snijders and Bosker (1994), NS for measures from Nakagawa and Schielzeth (2013) as extended by Johnson (2014), and RS for measures from Rights and Sterba (2019). The default setting is print = "RS".

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

plot logical: if TRUE, bar chart showing the decomposition of scaled total, within-cluster, and between-cluster outcome variance into five (total), three (within-cluster), and two (between-cluster) proportions is drawn. Note that the ggplot2 package is required to draw the bar chart.

gray logical: if TRUE, graphical parameter to draw the bar chart in gray scale.
A number of R-squared measures for multilevel and linear mixed effects models have been developed in the methodological literature (see Rights & Sterba, 2018). Based on these measures, following measures were implemented in the current function:

**Raudenbush and Bryk (2002)** R-squared measures by Raudenbush and Bryk (2002) are based on the proportional reduction of unexplained variance when predictors are added. More specifically, variance estimates from the baseline/null model (i.e., $\sigma^2_{e|b}$ and $\sigma^2_{u0|b}$) and variance estimates from the model including predictors (i.e., $\sigma^2_{e|m}$ and $\sigma^2_{u0|m}$) are used to compute the proportional reduction in variance between baseline/null model and the complete model by:

$$R^2_{1}(RB) = \frac{\sigma^2_{e|b} - \sigma^2_{e|m}}{\sigma^2_{e|b}}$$

for the proportional reduction at level-1 (within-cluster) and by:

$$R^2_{2}(RB) = \frac{\sigma^2_{u0|b} - \sigma^2_{u0|m}}{\sigma^2_{u0|b}}$$

for the proportional reduction at level-2 (between-cluster), where $|b$ and $|m$ represent the baseline and full models, respectively (Hox et al., 2018; Roberts et al., 2010).

A major disadvantage of these measures is that adding predictors can increases rather than decreases some of the variance components and it is even possible to obtain negative values for $R^2$ with these formulas (Snijders & Bosker, 2012). According to Snijders and Bosker (1994) this can occur because the between-group variance is a function of both level-1 and level-2 variance:

$$\text{var}(\bar{Y}_j) = \sigma^2_{u0} + \frac{\sigma^2_{e}}{n_j}$$

Hence, adding a predictor (e.g., cluster-mean centered predictor) that explains proportion of the within-group variance will decrease the estimate of $\sigma^2_{e}$ and increase the estimate $\sigma^2_{u0}$ if this predictor does not explain a proportion of the between-group variance to balance out the decrease in $\sigma^2_{e}$ (LaHuis et al., 2014). Negative estimates for $R^2$ can also simply occur due to chance fluctuation in sample estimates from the two models.
Another disadvantage of these measures is that $R^2(RB)$ for the explained variance at level-2 has been shown to perform poorly in simulation studies even with $j = 200$ clusters with group cluster size of $n_j = 50$ (LaHuis et al., 2014; Rights & Sterba, 2019). Moreover, when there is missing data in the level-1 predictors, it is possible that sample sizes for the baseline and complete models differ.

Finally, it should be noted that R-squared measures by Raudenbush and Bryk (2002) are appropriate for random intercept models, but not for random intercept and slope models. For random slope models, Snijders and Bosker (2012) suggested to re-estimate the model as random intercept models with the same predictors while omitting the random slopes to compute the R-squared measures. However, the simulation study by LaHuis (2014) suggested that the R-squared measures showed an acceptable performance when there was little slope variance, but did not perform well in the presence of higher levels of slope variance.

**Snijders and Bosker (1994)** R-squared measures by Snijders and Bosker (1994) are based on the proportional reduction of mean squared prediction error and is computed using the formula:

$$R^2_1(SB) = \frac{\hat{\sigma}^2_e|m + \hat{\sigma}^2_u|m}{\hat{\sigma}^2_e|b + \hat{\sigma}^2_u|b}$$

for computing the proportional reduction of error at level-1 representing the total amount of explained variance and using the formula:

$$R^2_2(SB) = \frac{\hat{\sigma}^2_e|m/n_j + \hat{\sigma}^2_u|m}{\hat{\sigma}^2_e|b/n_j + \hat{\sigma}^2_u|b}$$

for computing the proportional reduction of error at level-2 by dividing the $\hat{\sigma}^2_e$ by the group cluster size $n_j$ or by the average cluster size for unbalanced data (Roberts et al., 2010). Note that the function uses the harmonic mean of the group sizes as recommended by Snijders and Bosker (1994). The population values of $R^2$ based on these measures cannot be negative because the interplay of level-1 and level-2 variance components is considered. However, sample estimates of $R^2$ can be negative either due to chance fluctuation when sample sizes are small or due to model misspecification (Snijders and Bosker, 2012).

When there is missing data in the level-1 predictors, it is possible that sample sizes for the baseline and complete models differ.

Similar to the R-squared measures by Raudenbush and Bryk (2002), the measures by Snijders and Bosker (1994) are appropriate for random intercept models, but not for random intercept and slope models. Accordingly, for random slope models, Snijders and Bosker (2012) suggested to re-estimate the model as random intercept models with the same predictors while omitting the random slopes to compute the R-squared measures. The simulation study by LaHuis et al. (2014) revealed that the R-squared measures showed an acceptable performance, but it should be noted that $R^2_2(SB)$ the explained variance at level-2 was not investigated in their study.

**Nakagawa and Schielzeth (2013)** R-squared measures by Nakagawa and Schielzeth (2013) are based on partitioning model-implied variance from a single fitted model and uses the variance of predicted values of $\text{var}(\hat{Y}_{ij})$ to form both the outcome variance in the denominator and the explained variance in the numerator of the formulas:

$$R^2_m(NS) = \frac{\text{var}(\hat{Y}_{ij})}{\text{var}(\hat{Y}_{ij}) + \hat{\sigma}^2_u + \hat{\sigma}^2_e}$$
for marginal total $R^2_m(\text{NS})$ and:

$$R^2_{\text{c}}(\text{NS}) = \frac{\text{var}(\hat{Y}_{ij}) + \sigma^2_{u0}}{\text{var}(\hat{Y}_{ij}) + \sigma^2_{u0} + \sigma^2_e}$$

for conditional total $R^2_{\text{c}}(\text{NS})$. In the former formula $R^2$ predicted scores are marginalized across random effects to indicate the variance explained by fixed effects and in the latter formula $R^2$ predicted scores are conditioned on random effects to indicate the variance explained by fixed and random effects (Rights and Sterba, 2019).

The advantage of these measures is that they can never become negative and that they can also be extended to generalized linear mixed effects models (GLMM) when outcome variables are not continuous (e.g., binary outcome variables). Note that currently the function does not provide $R^2$ measures for GLMMs, but these measures can be obtained using the r.squaredGLMM() function in the MuMIn package.

A disadvantage is that these measures do not allow random slopes and are restricted to the simplest random effect structure (i.e., random intercept model). In other words, these measures do not fully reflect the structure of the fitted model when using random intercept and slope models. However, Johnson (2014) extended these measures to allow random slope by taking into account the contribution of random slopes, intercept-slope covariances, and the covariance matrix of random slope to the variance in $Y_{ij}$. As a result, R-squared measures by Nakagawa and Schielzeth (2013) as extended by Johnson (2014) can be used for both random intercept, and random intercept and slope models.

The major criticism of the R-squared measures by Nakagawa and Schielzeth (2013) as extended by Johnson (2014) is that these measures do not decompose outcome variance into each of total, within-cluster, and between-cluster variance which precludes from computing level-specific $R^2$ measures. In addition, these measures do not distinguish variance attributable to level-1 versus level-2 predictors via fixed effects, and they also do not distinguish between random intercept and random slope variation (Rights and Sterba, 2019).

Rights and Sterba (2019) R-squared measures by Rights and Sterba (2019) provide an integrative framework of R-squared measures for multilevel and linear mixed effects models with random intercepts and/or slopes. Their measures are also based on partitioning model implied variance from a single fitted model, but they provide a full partitioning of the total outcome variance to one of five specific sources:

- variance attributable to level-1 predictors via fixed slopes (shorthand: variance attributable to $f1$)
- variance attributable to level-2 predictors via fixed slopes (shorthand: variance attributable to $f2$)
- variance attributable to level-1 predictors via random slope variation/ covariation (shorthand: variance attributable to $v$)
- variance attributable to cluster-specific outcome means via random intercept variation (shorthand: variance attributable to $m$)
- variance attributable to level-1 residuals

$R^2$ measures are based on the outcome variance of interest (total, within-cluster, or between-cluster) in the denominator, and the source contributing to explained variance in the numerator:

**Total $R^2$ measures** incorporate both within-cluster and between cluster variance in the denominator and quantify variance explained in an omnibus sense:
• $R^2_{t1}$: Proportion of total outcome variance explained by level-1 predictors via fixed slopes.
• $R^2_{t2}$: Proportion of total outcome variance explained by level-2 predictors via fixed slopes.
• $R^2_{tf}$: Proportion of total outcome variance explained by all predictors via fixed slopes.
• $R^2_{tv}$: Proportion of total outcome variance explained by level-1 predictors via random slope variation/covariation.
• $R^2_{tm}$: Proportion of total outcome variance explained by cluster-specific outcome means via random intercept variation.
• $R^2_{tfv}$: Proportion of total outcome variance explained by predictors via fixed slopes and random slope variation/covariation.
• $R^2_{fvm}$: Proportion of total outcome variance explained by predictors via fixed slopes and random slope variation/covariation and by cluster-specific outcome means via random intercept variation.

**Within-Cluster $R^2$ measures** incorporate only within-cluster variance in the denominator and indicate the degree to which within-cluster variance can be explained by a given model:

• $R^2_{w1}$: Proportion of within-cluster outcome variance explained by level-1 predictors via fixed slopes.
• $R^2_{wv}$: Proportion of within-cluster outcome variance explained by level-1 predictors via random slope variation/covariation.
• $R^2_{wfv}$: Proportion of within-cluster outcome variance explained by level-1 predictors via fixed slopes and random slope variation/covariation.

**Between-Cluster $R^2$ measures** incorporate only between-cluster variance in the denominator and indicate the degree to which between-cluster variance can be explained by a given model:

• $R^2_{b2}$: Proportion of between-cluster outcome variance explained by level-2 predictors via fixed slopes.
• $R^2_{bm}$: Proportion of between-cluster outcome variance explained by cluster-specific outcome means via random intercept variation.

The decomposition of the total outcome variance can be visualized in a bar chart by specifying `plot = TRUE`. The first column of the bar chart decomposes scaled total variance into five distinct proportions (i.e., $R^2_{t1}$, $R^2_{t2}$, $R^2_{tf}$, $R^2_{tm}$, and $R^2_{tfv}$), the second column decomposes scaled within-cluster variance into three distinct proportions (i.e., $R^2_{w1}$, $R^2_{wv}$, and $R^2_{wfv}$), and the third column decomposes scaled between-cluster variance into two distinct proportions (i.e., $R^2_{b2}$ and $R^2_{bm}$).

Note that the function assumes that all level-1 predictors are centered within cluster (i.e., group-mean or cluster-mean centering) as has been widely recommended (e.g., Enders & Tofighi, D., 2007; Rights et al., 2019). In fact, it does not matter whether a lower-level predictor is merely a control variable, or is quantitative or categorical (Yaremych et al., 2021), cluster-mean centering should always be used for lower-level predictors to obtain an orthogonal between-within partitioning of a lower-level predictor’s variance that directly parallels what happens to a level-1 outcome (Hoffman & Walters, 2022). In the absence of cluster-
mean-centering, however, the function provides total $R^2$ measures, but does not provide any within-cluster or between-cluster $R^2$ measures.

By default, the function only computes R-squared measures by Rights and Sterba (2019) because the other R-squared measures reflect the same population quantity provided by Rights and Sterba (2019). That is, R-squared measures $R^2_1(RB)$ and $R^2_2(RB)$ by Raudenbush and Bryk (2002) are equivalent to $R^2_{1(v)}(f_1)$ and $R^2_{1(v)}(f_2)$. R-squared measures $R^2_1(SB)$ and $R^2_2(SB)$ are equivalent to $R^2_{1(f)}$ and $R^2_{2(f)}$, and R-squared measures $R^2_{m(NS)}$ and $R^2_{b(NS)}$ by Nakagawa and Schielzeth (2013) as extended by Johnson (2014) are equivalent to $R^2_{1(f)}$ and $R^2_{2(f,v)}$ (see Rights and Sterba, Table 3).

Note that none of these measures provide an $R^2$ for the random slope variance explained by cross-level interactions, a quantity that is frequently of interest (Hoffman & Walters, 2022).

**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 `model` (`model`), specification of function arguments (`args`), and list with results (`result`).

**Note**

This function is based on the `multilevelR2()` function from the `mitml` package by Simon Grund, Alexander Robitzsch and Oliver Luedtke (2021) and calls the `r2mlm()` function in the `r2mlm` package by Mairead Shaw, Jason Rights, Sonya Sterba, and Jessica Flake.

**Author(s)**

Simon Grund, Alexander Robitzsch, Oliver Luedtke, Mairead Shaw, Jason D. Rights, Sonya K. Sterba, Jessica K. Flake, and Takuya Yanagida

**References**


multilevel.r2


See Also

multilevel.cor, multilevel.descript, multilevel.icc, multilevel.indirect

Examples

```r
## Not run:
# Load misty, lme4, nlme, and ggplot2 package
library(misty)
library(lme4)
library(nlme)
library(ggplot2)

# Load data set "Demo.twolevel" in the lavaan package
data("Demo.twolevel", package = "lavaan")

#---------------------------
# Cluster mean centering, center() from the misty package
demo.twolevel$x2.c <- center(demo.twolevel$x2, type = "CWC", cluster = demo.twolevel$cluster)

# Compute group means, cluster.scores() from the misty package
demo.twolevel$x2.b <- cluster.scores(demo.twolevel$x2, cluster = demo.twolevel$cluster)

# Estimate multilevel model using the lme4 package
mod1a <- lmer(y1 ~ x2.c + x2.b + w1 + (1 + x2.c | cluster), data = demo.twolevel, REML = FALSE, control = lmerControl(optimizer = "bobyqa"))
```
# R-squared measures according to Rights and Sterba (2019)
multi.level.r2(mod1a)

# Estimate multilevel model using the nlme package
mod1b <- lme(y1 ~ x2.c + x2.b + w1, random = ~ 1 + x2.c | cluster, data = Demo.twolevel,
  method = "ML")

# R-squared measures according to Rights and Sterba (2019)
multi.level.r2(mod1b)

# Bar chart showing the decomposition of scaled total, within-cluster,
# and between-cluster outcome variance
multi.level.r2(mod1a, plot = TRUE)

# Bar chart in gray scale
multi.level.r2(mod1a, plot = TRUE, gray = TRUE)

# Save bar chart, ggsave() from the ggplot2 package
ggsave("Proportion_of_Variance.png", dpi = 600, width = 5.5, height = 5.5)

# Estimate multilevel model without random slopes
# Note. R-squared measures by Raudenbush and Bryk (2002), and Snijders and Bosker (2012) should be computed based on the random intercept model
mod2 <- lmer(y1 ~ x2.c + x2.b + w1 + (1 | cluster), data = Demo.twolevel,
  REML = FALSE, control = lmerControl(optimizer = "bobyqa"))

# Print all available R-squared measures
multi.level.r2(mod2, print = "all")

# Draw bar chart manually
mod1a.r2 <- multi.level.r2(mod1a, output = FALSE)

# Prepare data frame for ggplot()
df <- data.frame(var = factor(rep(c("Total", "Within", "Between"), each = 5),
  level = c("Total", "Within", "Between"),
  part = factor(c("Fixed Slopes (Within)", "Fixed Slopes (Between)",
  "Slope Variation (Within)"), "Intercept Variation (Between)",
  "Residual (Within)",
  level = c("Residual (Within)"), "Intercept Variation (Between)",
  "Slope Variation (Within)"), "Fixed Slopes (Between)",
  "Fixed Slopes (Within)")),
  y = as.vector(mod1a.r2$result$rs$decomp))

# Draw bar chart in line with the default setting of multi.level.r2()
ggplot(df, aes(x = var, y = y, fill = part) +
theme_bw() +
geom_bar(stat = "identity") +
scale_fill_manual(values = c("#E69F00", "#009E73", "#CC79A7", "#0072B2", "#D55E00")) +
scale_y_continuous(name = "Proportion of Variance", breaks = seq(0, 1, by = 0.1)) +
theme(axis.title.x = element_blank(),
      axis.ticks.x = element_blank(),
      legend.title = element_blank(),
      legend.position = "bottom",
      legend.box.margin = margin(-10, 6, 6, 6)) +
guides(fill = guide_legend(nrow = 2, reverse = TRUE))

## End(Not run)

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, na.test

References

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), stringsAsFactors = FALSE)

# 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)
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 missingness.

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


See Also

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

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 dat  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,
            write = 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.
write     a character string for writing the results into a Excel file naming a file with or
         without file extension '.xlsx', e.g., "Results.xlsx" or "Results".
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 func-
tion arguments (args), and list with results (result).
**Author(s)**

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

**References**


**See Also**

`write.result`, `as.na`, `na.as`, `na.auxiliary`, `na.descript`, `na.indicator`, `na.pattern`, `na.prop`, `na.test`

**Examples**

```r
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 variance-covariance coverage
na.coverage(dat)

## Not run:
# Write Results into a Excel file
na.coverage(dat, write = "Coverage.xlsx")

result <- na.coverage(dat, output = FALSE)
write.result(result, "Coverage.xlsx")
## End(Not run)
```

**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**

```r
na.descript(x, table = FALSE, digits = 2, as.na = NULL, write = 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.
- **write**: a character string for writing the results into a Excel file naming a file with or without file extension '.xlsx', e.g., "Results.xlsx" or "Results".
- **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


See Also

write.result, as.na, na.as, na.auxiliary, na.coverage, na.indicator, na.pattern, na.prop, na.test

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)

## Not run:
# Write Results into a Excel file
na.descript(dat, table = TRUE, write = "NA_Descriptives.xlsx")
esult <- na.descript(dat, table = TRUE, output = FALSE)
write.result(result, "NA_Descriptives.xlsx")
# End(Not run)

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


See Also

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

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)

<table>
<thead>
<tr>
<th>na.pattern</th>
<th>Missing Data Pattern</th>
</tr>
</thead>
</table>

Description

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

Usage

na.pattern(x, order = FALSE, digits = 2, as.na = NULL, write = 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.
write a character string for writing the results into a Excel file naming a file with or without file extension '.xlsx', e.g., "Results.xlsx" or "Results".
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

See Also
write.result, as.na, na.as, na.auxiliary, na.coverage, na.descript, na.indicator, na.prop, na.test

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), ]

# Not run:
# Write Results into a Excel file
result <- na.pattern(dat, write = "NA_Pattern.xlsx")

result <- na.pattern(dat, output = FALSE)
write.result(result, "NA_Pattern.xlsx")

## End(Not run)

---

**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


See Also

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

Examples

```r
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)
```

---

**na.test**

*Little’s Missing Completely at Random (MCAR) Test*

Description

This function performs Little’s Missing Completely at Random (MCAR) test.

Usage

```r
na.test(x, digits = 2, p.digits = 3, as.na = NULL, check = TRUE, output = TRUE)
```
Arguments

- **x**: A matrix or data frame with incomplete data, where missing values are coded as NA.
- **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

Little (1988) proposed a multivariate test of Missing Completely at Random (MCAR) that tests for mean differences on every variable in the data set across subgroups that share the same missing data pattern by comparing the observed variable means for each pattern of missing data with the expected population means estimated using the expectation-maximization (EM) algorithm (i.e., EM maximum likelihood estimates). The test statistic is the sum of the squared standardized differences between the subsample means and the expected population means weighted by the estimated variance-covariance matrix and the number of observations within each subgroup (Enders, 2010). Under the null hypothesis that data are MCAR, the test statistic follows asymptotically a chi-square distribution with \( \sum k_j - k \) degrees of freedom, where \( k_j \) is the number of complete variables for missing data pattern \( j \), and \( k \) is the total number of variables. A statistically significant result provides evidence against MCAR.

Note that Little’s MCAR test has a number of problems (see Enders, 2010). **First**, the test does not identify the specific variables that violates MCAR, i.e., the test does not identify potential correlates of missingness (i.e., auxiliary variables). **Second**, the test is based on multivariate normality, i.e., under departure from the normality assumption the test might be unreliable unless the sample size is large and is not suitable for categorical variables. **Third**, the test investigates mean differences assuming that the missing data pattern share a common covariance matrix, i.e., the test cannot detect covariance-based deviations from MCAR stemming from a Missing at Random (MAR) or Missing Not at Random (MNAR) mechanism because MAR and MNAR mechanisms can also produce missing data subgroups with equal means. **Fourth**, simulation studies suggest that Little’s MCAR test suffers from low statistical power, particularly when the number of variables that violate MCAR is small, the relationship between the data and missingness is weak, or the data are MNAR (Thoemmes & Enders, 2007). **Fifth**, the test can only reject, but cannot prove the MCAR assumption, i.e., a statistically not significant result and failing to reject the null hypothesis of the MCAR test does not prove the null hypothesis that the data is MCAR. **Finally**, under the null hypothesis the data are actually MCAR or MNAR, while a statistically significant result indicates that missing data are MAR or MNAR, i.e., MNAR cannot be ruled out regardless of the result of the test.

This function is based on the `prelim.norm` function in the `norm` package which can handle about 30 variables. With more than 30 variables specified in the argument x, the `prelim.norm` function might run into numerical problems leading to results that are not trustworthy. In this case it is recommended to reduce the number of variables specified in the argument x. If the number of variables
cannot be reduced, it is recommended to use the LittleMCAR function in the BaylorEdPsych package which can deal with up to 50 variables. However, this package was removed from the CRAN repository and needs to be obtained from the archive along with the mvnmle which is needed for using the LittleMCAR function. Note that the mcar_test function in the naniar package is also based on the prelim.norm function which results are not trustworthy whenever the warning message In norm::prelim.norm(data) : NAs introduced by coercion to integer range 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), list with results (result).

Note

Code is adapted from the R function by Eric Stemmler: tinyurl.com/r-function-for-MCAR-test

Author(s)

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

References


See Also

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

Examples

na.test(airquality)

print.misty.object  Print misty.object object

Description

This function prints the misty.object object
Usage

```r
## S3 method for class 'misty.object'
print(x,
      print = x$args$print, tri = x$args$tri, freq = x$args$freq,
      hypo = x$args$hypo, descript = x$args$descript, effsize = x$args$effsize,
      posthoc = x$args$posthoc, 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 (crosstab() function).
- `hypo` logical: if TRUE, null and alternative hypothesis are shown on the console (test.t, test.welch, test.z function).
- `descript` logical: if TRUE, descriptive statistics are shown on the console (test.t, test.welch, test.z function).
- `effsize` logical: if TRUE, effect size measure(s) is shown on the console (test.t, test.welch, test.z function).
- `posthoc` logical: if TRUE, post hoc test for multiple comparison is shown on the console (test.welch).
- `split` logical: if TRUE, output table is split by variables when specifying more than one variable in x (freq).
- `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() and multilevel.icc() 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.
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**


**See Also**

run.mplus, write.mplus, read.sav, read.xlsx

**Examples**

```r
## 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**

```r
read.sav(file, use.value.labels = FALSE, use.missings = TRUE, formats = FALSE,
label = TRUE, labels = TRUE, missing = FALSE, widths = FALSE,
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.
formats  logical: if TRUE, variable formats are shown in an attribute for all variables.
label  logical: if TRUE (default), variable labels are shown in an attribute for all variables.
labels  logical: if TRUE (default), value labels are shown in an attribute for all variables.
missing  logical: if TRUE, value labels for user-defined missings are shown in an attribute for all variables.
widths  logical: if TRUE, widths are shown in an attribute for all variables.
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 Evan Miller

References


See Also

write.sav, read.xlsx, read.mplus

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)
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

```r
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.
read.xlsx

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
write.xlsx, 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), return a tibble
read.xlsx("data.xlsx", as.data.frame = FALSE)
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, 1o: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.

Value

Returns a numeric vector with the same length as x containing the recoded variable.
Note
This function was adapted from the recode() function in the car package by John Fox and Sanford Weisberg (2019).

Author(s)
Takuya Yanagida <takuya.yanagida@univie.ac.at>

References

See Also
item.reverse

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"))

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.

Note

This function is a copy of the runModels() function in the MplusAutomation package by Michael Hallquist and Joshua Wiley (2018).

Author(s)

Michael Hallquist

References


Examples

```r
## 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)
```
Description

This function computes \( r^{*}\text{wg}(j) \) within-group agreement index for multi-item scales as described in Lindell, Brandt and Whitney (1999).

Usage

```r
rwg.lindell(x, cluster, 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.
- **cluster**
  - a vector representing the nested grouping structure (i.e., group or cluster 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 \( \text{ranvar} \) 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 \( \text{ranvar} \) is specified.
- **z**
  - logical: if \( \text{TRUE} \), Fisher \( z \)-transformation based on the formula \( z = 0.5 \times \log((1 + r)/(1 - r)) \) is applied to the vector of \( r^{*}\text{wg}(j) \) estimates.
- **expand**
  - logical: if \( \text{TRUE} \), vector of \( r^{*}\text{wg}(j) \) estimates is expanded to match the input vector \( x \).
- **na.omit**
  - logical: if \( \text{TRUE} \), 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 \( \text{NA} \) before conducting the analysis. Note that \( \text{as.na()} \) function is only applied to \( x \), but not to \( \text{cluster} \).
- **check**
  - logical: if \( \text{TRUE} \), argument specification is checked.

Details

The \( r^{*}\text{wg}(j) \) 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 \( \sigma_u^2 = (A^2 - 1)/12 \), 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 \( r^{*}\text{wg}(j) \) allows that the mean of the item variances to be larger than the expected random variances, i.e., \( r^{*}\text{wg}(j) \) values can be negative.
Note that the \texttt{rwg.j.lindell()} function in the \texttt{multilevel} package uses listwise deletion by default, while the \texttt{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 (\texttt{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).

\textbf{Value}

Returns a numeric vector containing r*wg(j) agreement index for multi-item scales with the same length as \texttt{cluster} if \texttt{expand = TRUE} or a data frame with following entries if \texttt{expand = FALSE}:

<table>
<thead>
<tr>
<th>cluster</th>
<th>cluster identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>cluster size x</td>
</tr>
<tr>
<td>rwg.lindell</td>
<td>r*wg(j) estimate for each cluster</td>
</tr>
<tr>
<td>z.rwg.lindell</td>
<td>Fisher z-transformed r*wg(j) estimate for each cluster</td>
</tr>
</tbody>
</table>

\textbf{Author(s)}

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

\textbf{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. \textit{Learning and Instruction}, 61, 72-83. https://doi.org/10.1016/j.learninstruc.2019.01.003


\textbf{See Also}

\texttt{cluster.scores}

\textbf{Examples}

```r
dat <- data.frame(id = c(1, 2, 3, 4, 5, 6, 7, 8, 9),
                 cluster = c(1, 1, 1, 2, 2, 2, 3, 3, 3),
                 rwg.lindell)
```

x1 = c(2, 3, 2, 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")], cluster = dat$cluster, 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")], cluster = dat$cluster, ranvar = 2)

# Compute r*wg(j) for a multi-item scale with A = 5 response options
rwg.lindell(dat[, c("x1", "x2", "x3")], cluster = dat$cluster, 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")], cluster = dat$cluster, A = 5, expand = FALSE)

---

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, $\rho$.0.
- **delta**: a numeric value indicating the minimum difference to be detected, $\delta$.
- **alternative**: a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".
- **alpha**: type-I-risk, $\alpha$.
- **beta**: type-II-risk, $\beta$.
- **check**: logical: if TRUE, argument specification is checked.
- **output**: logical: if TRUE, output is shown.
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).

Value

Returns an object of class misty.object with following entries:

- **call**: function call
- **type**: type of the test (i.e., correlation coefficient)
- **args**: specification of function arguments
- **result**: list with the result, i.e., optimal sample size

Author(s)

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

References


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)
```
Usage

```r
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, \( \delta \).
- `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, \( \alpha \).
- `beta`: type-II-risk, \( \beta \).
- `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)
- `args`: specification of function arguments
- `result`: list with the result, i.e., optimal sample size

Author(s)

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

References


See Also

`size.prop`, `size.cor`

Examples

```r
#--------------------------------------
# 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**

```r
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, $\delta$.
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, $\alpha$.
beta  type-II-risk, $\beta$.
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:

<table>
<thead>
<tr>
<th>call</th>
<th>function call</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>type of the test (i.e., proportion)</td>
</tr>
<tr>
<td>args</td>
<td>specification of function arguments</td>
</tr>
<tr>
<td>result</td>
<td>list with the result, i.e., optimal sample size</td>
</tr>
</tbody>
</table>

Author(s)

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

References


See Also

size.mean, size.cor

Examples

```r
# 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)
```
# Two-sided one-sample test
# H0: pi = 0.5, H1: pi != 0.5
# alpha = 0.05, beta = 0.2, delta = 0.2
# with continuity correction

size.prop(pi = 0.5, delta = 0.2, sample = "one.sample",
            alternative = "two.sided", alpha = 0.05, beta = 0.2,
            correct = TRUE)

# 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

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 = 4, 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 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 $\beta$ can be standardized with respect to only $x$, only $y$, or both $y$ and $x$:

$$\text{Std}X(\beta_1) = \beta_1 SD(x)$$

$\text{Std}X(\beta_1)$ standardizes with respect to $x$ only and is interpreted as the change in $y$ when $x$ changes one standard deviation referred to as $SD(x)$.

$$\text{Std}Y(\beta_1) = \frac{\beta_1}{SD(x)}$$

$\text{Std}Y(\beta_1)$ 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.

$$\text{Std}YX(\beta_1) = \beta_1 \frac{SD(x)}{SD(y)}$$

$\text{Std}YX(\beta_1)$ 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 $\text{Std}YX(\beta_1)$ and the $\text{Std}Y(\beta_1)$ 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 $\beta_3$ 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 $SD(x_1x_2)$ for the interaction variable $x_1x_2$ (see Wen, Marsh & Hau, 2010). Likewise, the standardization of the slope $\beta_3$ 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 $SD(x^2)$ for the quadratic term $x^2$. 
Value

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

Author(s)

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

References


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, 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)

test.levene

Levene's Test for Homogeneity of Variance

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

Usage
test.levene(formula, data, method = c("median", "mean"),
            conf.level = 0.95, hypo = TRUE, descript = TRUE,
            plot = TRUE, violin.alpha = 0.3, violin.trim = FALSE,
            box = TRUE, box.alpha = 0.2, box.width = 0.2,
            jitter = TRUE, jitter.size = 1.25, jitter.width = 0.05, jitter.alpha = 0.2,
            gray = FALSE, start = 0.9, end = 0.4, color = NULL,
            xlab = NULL, ylab = NULL, ylim = NULL, breaks = ggplot2::waiver(),
            title = "", subtitle = "", 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 based 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.
hypo logical: if TRUE, null and alternative hypothesis are shown on the console.
descript logical: if TRUE, descriptive statistics are shown on the console.
plot logical: if TRUE, a plot showing violin plots with boxplots is drawn.
violin.alpha a numeric value indicating the opacity of the violins.
violin.trim logical: if TRUE, the tails of the violins to the range of the data is trimmed.
box logical: if TRUE (default), boxplots are drawn.
box.alpha a numeric value indicating the opacity of the boxplots.
box.width a numeric value indicating the width of the boxplots.
jitter logical: if TRUE (default), jittered data points are drawn.
jitter.size   a numeric value indicating the size aesthetic for the jittered data points.
jitter.width a numeric value indicating the amount of vertical and horizontal jitter.
jitter.alpha a numeric value indicating the opacity of the jittered data points.
gray         logical: if TRUE, the plot is drawn in gray scale.
start        a numeric value between 0 and 1, graphical parameter to specify the gray value
             at the low end of the palette.
end          a numeric value between 0 and 1, graphical parameter to specify the gray value
             at the high end of the palette.
color        a character vector, indicating the color of the violins and the boxes. By default,
             default ggplot2 colors are used.
xlab         a character string specifying the labels for the x-axis.
ylab         a character string specifying the labels for the y-axis.
ylim         a numeric vector of length two specifying limits of the limits of the y-axis.
breaks        a numeric vector specifying the points at which tick-marks are drawn at the y-
             axis.
title        a character string specifying the text for the title for the plot.
subtitle     a character string specifying the text for the subtitle for the plot.
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 interval and an object of class "anova" (result).

Author(s)

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


See Also

`aov.b`, `test.t`, `test.welch`

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
test.levene(y ~ group, data = dat)

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

# Levene's test based on the median with 99% confidence interval
test.levene(y ~ group, data = dat, conf.level = 0.99)
## Not run:
# Levene's test based on the median with 95
# plot results
test.levene(y ~ group, data = dat, plot = TRUE)

# Load ggplot2 package
library(ggplot2)

# Save plot, ggsave() from the ggplot2 package
ggsave("Levene-test.png", dpi = 600, width = 5, height = 6)

# Levene's test based on the median with 95
# extract plot
p <- test.levene(y ~ group, data = dat, output = FALSE)$plot
p

# Extract data
plotdat <- test.levene(y ~ group, data = dat, output = FALSE)$data

# Draw violin and boxplots in line with the default setting of test.levene()
ggplot(plotdat, aes(group, y, fill = group)) +
   geom_violin(alpha = 0.3, trim = FALSE) +
   geom_boxplot(alpha = 0.2, width = 0.2) +
   geom_jitter(alpha = 0.2, width = 0.05, size = 1.25) +
   theme_bw() + guides(fill = "none")

## End(Not run)
Description

This function performs one-sample, two-sample, and paired-sample t-tests and provides descriptive statistics, effect size measure, and a plot showing error bars for confidence intervals with jittered data points.

Usage

test.t(x, ...)

## Default S3 method:
test.t(x, y = NULL, mu = 0, paired = FALSE,
alternative = c("two.sided", "less", "greater"),
conf.level = 0.95, hypo = TRUE, descript = TRUE, effsize = FALSE,
weighted = TRUE, cor = TRUE, ref = NULL, correct = FALSE,
plot = FALSE, point.size = 4, error.width = 0.1,
xlab = NULL, ylab = NULL, ylim = NULL, breaks = ggplot2::waiver(),
line = TRUE, line.type = 3, line.size = 0.8,
jitter = TRUE, jitter.size = 1.25, jitter.width = 0.05, jitter.alpha = 0.1,
title = "", subtitle = "Confidence Interval",
digits = 2, p.digits = 4, as.na = NULL, check = TRUE, output = TRUE, ...)

## S3 method for class 'formula'
test.t(formula, data, alternative = c("two.sided", "less", "greater"),
conf.level = 0.95, hypo = TRUE, descript = TRUE, effsize = FALSE,
weighted = TRUE, cor = TRUE, ref = NULL, correct = FALSE,
plot = FALSE, point.size = 4, error.width = 0.1,
xlab = NULL, ylab = NULL, ylim = NULL, breaks = ggplot2::waiver(),
jitter = TRUE, jitter.size = 1.25, jitter.width = 0.05, jitter.alpha = 0.1,
title = "", subtitle = "Confidence Interval",
digits = 2, p.digits = 4, as.na = NULL, check = TRUE, output = TRUE, ...)

Arguments

x
  a numeric vector of data values.

y
  a numeric vector of data values.

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 t-test.

paired
  logical: if TRUE, paired-samples t-test is computed.

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

hypo
  logical: if TRUE, null and alternative hypothesis are shown on the console.
The `test.t` function provides several optional arguments for customizing the output:

- **`descript`**: logical; if TRUE, descriptive statistics are shown on the console.
- **`effsize`**: logical; if TRUE, effect size measure Cohen’s d is shown on the console, see `cohens.d` function.
- **`weighted`**: logical; if TRUE (default), the weighted pooled standard deviation is used to compute Cohen’s d for a two-sample design (i.e., `paired = FALSE`), while standard deviation of the difference scores is used to compute Cohen’s d for a paired-sample design (i.e., `paired = TRUE`).
- **`cor`**: logical; if TRUE (default), `paired = TRUE`, and `weighted = FALSE`, Cohen’s d for a paired-sample design while controlling for the correlation between the two sets of measurement is computed. Note that this argument is only used in a paired-sample design (i.e., `paired = TRUE`) when specifying `weighted = FALSE`.
- **`ref`**: character string “x” or “y” for specifying the reference reference group when using the default `test.t()` function or a numeric value or character string indicating the reference group in a two-sample design when using the formula `test.t()` function. The standard deviation of the reference variable or reference group is used to standardized the mean difference to compute Cohen’s d. Note that this argument is only used in a two-sample design (i.e., `paired = FALSE`).
- **`correct`**: logical; if TRUE, correction factor to remove positive bias in small samples is used.
- **`conf.level`**: a numeric value between 0 and 1 indicating the confidence level of the interval.
- **`plot`**: logical; if TRUE, a plot showing error bars for confidence intervals is drawn.
- **`point.size`**: a numeric value indicating the size aesthetic for the point representing the mean value.
- **`error.width`**: a numeric value indicating the horizontal bar width of the error bar.
- **`xlab`**: a character string specifying the labels for the x-axis.
- **`ylab`**: a character string specifying the labels for the y-axis.
- **`ylim`**: a numeric vector of length two specifying limits of the y-axis.
- **`breaks`**: a numeric vector specifying the points at which tick-marks are drawn at the y-axis.
- **`line`**: logical; if TRUE (default), a horizontal line is drawn at mu for the one-sample t-test or at 0 for the paired-sample t-test.
- **`line.type`**: an integer value or character string specifying the line type for the line representing the population mean under the null hypothesis, i.e., 0 = blank, 1 = solid, 2 = dashed, 3 = dotted, 4 = dotdash, 5 = longdash, 6 = twodash.
- **`line.size`**: a numeric value indicating the size aesthetic for the line representing the population mean under the null hypothesis.
- **`jitter`**: logical; if TRUE (default), jittered data points are drawn.
- **`jitter.size`**: a numeric value indicating the size aesthetic for the jittered data points.
- **`jitter.width`**: a numeric value indicating the amount of vertical and horizontal jitter.
- **`jitter.alpha`**: a numeric value indicating the opacity of the jittered data points.
- **`title`**: a character string specifying the text for the title for the plot.
subtitle | a character string specifying the text for the subtitle for the plot.
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 t-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 and y (data), data used to plot the results (plot.data), specification of function arguments (args), and result table (result).

Author(s)

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

References


See Also

aov.b, test.welch, test.z, test.levene, cohens.d, ci.mean.diff, ci.mean

Examples

dat1 <- 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, 6, 3, NA))

#--------------------------------------
# One-Sample Design
#--------------------------------------
# Two-sided one-sample t-test
# population mean = 3
test.t(dat1$x, mu = 3)

# One-sided one-sample t-test
# population mean = 3, population standard deviation = 1.2
test.t(dat1$x, mu = 3, alternative = "greater")

# Two-sided one-sample t-test
# population mean = 3, convert value 3 to NA
test.t(dat1$x, mu = 3, as.na = 3)

# Two-sided one-sample t-test
# population mean = 3, print Cohen's d
test.t(dat1$x, sigma = 1.2, mu = 3, effsize = TRUE)

# Two-sided one-sample t-test
# population mean = 3, print Cohen's d with small sample correction factor
test.t(dat1$x, sigma = 1.2, mu = 3, effsize = TRUE, correct = TRUE)

# Two-sided one-sample t-test
# population mean = 3,
# do not print hypotheses and descriptive statistics
test.t(dat1$x, sigma = 1.2, mu = 3, hypo = FALSE, descript = FALSE)

# Two-sided one-sample t-test
# population mean = 3,
# do not print hypotheses and descriptive statistics
# print descriptive statistics with 3 digits and p-value with 5 digits
test.t(dat1$x, mu = 3, digits = 3, p.digits = 5)

## Not run:
# Two-sided one-sample t-test
# population mean = 3, plot results
test.t(dat1$x, mu = 3, plot = TRUE)

# Load ggplot2 package
library(ggplot2)

ggsave("One-sample_t-test.png", dpi = 600, width = 3, height = 6)

# Two-sided one-sample t-test
# population mean = 3, extract plot
p <- test.t(dat1$x, mu = 3, output = FALSE)$plot
p

# Extract data
plotdat <- data.frame(x = test.t(dat1$x, mu = 3, output = FALSE)$data[[1]])

# Draw plot in line with the default setting of test.t()
ggplot(plotdat, aes(0, x)) +
  geom_point(stat = "summary", fun = "mean", size = 4) +
  stat_summary(fun.data = "mean_cl_normal", geom = "errorbar", width = 0.20) +
  scale_x_continuous(name = NULL, limits = c(-2, 2)) +
  scale_y_continuous(name = NULL) +
  geom_hline(yintercept = 3, linetype = 3, size = 0.8) +
  labs(subtitle = "Two-Sided 95")
theme_bw() + theme(plot.subtitle = element_text(hjust = 0.5),
  axis.text.x = element_blank(),
  axis.ticks.x = element_blank())
## End(Not run)
#--------------------------------------
# Two-Sample Design

# Two-sided two-sample t-test
test.t(x ~ group, data = dat1)

# One-sided two-sample t-test
# print Cohen's d with weighted pooled SD
# print Cohen's d with unweighted pooled SD
# print Cohen's d with weighted pooled SD and
# small sample correction factor
# do not print hypotheses and descriptive statistics,
# print descriptive statistics with 3 digits and p-value with 5 digits
# Load ggplot2 package
# Two-sided two-sample t-test
# do not print hypotheses and descriptive statistics,
# print descriptive statistics with 3 digits and p-value with 5 digits
# Load ggplot2 package
library(ggplot2)

# Save plot, ggsave() from the ggplot2 package
# Two-sided two-sample t-test
# extract plot
p <- test.t(x ~ group, data = dat1, output = FALSE)$plot
p

# Extract data used to plot results
plotdat <- test.t(x ~ group, data = dat1, output = FALSE)$data

# Draw plot in line with the default setting of test.t()
ggplot(plotdat, aes(factor(group), x)) +
  geom_point(stat = "summary", fun = "mean", size = 4) +
  stat_summary(fun.data = "mean_cl_normal", geom = "errorbar", width = 0.20) +
  scale_x_discrete(name = NULL) + scale_y_continuous(name = "y") +
  labs(title = "", subtitle = "Two-Sided 95")

## End(Not run)

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

# Two-sided two-sample t-test
test.t(group1, group2)

#--------------------------------------
# Paired-Sample Design

dat2 <- data.frame(pre = c(1, 3, 2, 5, 7),
  post = c(2, 2, 1, 6, 8))

# Two-sided paired-sample t-test
test.t(dat2$pre, dat2$post, paired = TRUE)

# One-sided paired-sample t-test
test.t(dat2$pre, dat2$post, paired = TRUE, alternative = "greater")

# Two-sided paired-sample t-test
# convert value 1 to NA
test.t(dat2$pre, dat2$post, as.na = 1, paired = TRUE)

# Two-sided paired-sample t-test
# print Cohen's d based on the standard deviation of the difference scores
# test.t(dat2$pre, dat2$post, paired = TRUE, effsize = TRUE)

# Two-sided paired-sample t-test
# print Cohen's d based on the standard deviation of the difference scores
# with small sample correction factor
test.t(dat2$pre, dat2$post, paired = TRUE, effsize = TRUE, correct = TRUE)

# Two-sided paired-sample t-test
# print Cohen's d controlling for the correlation between measures
test.t(dat2$pre, dat2$post, paired = TRUE, effsize = TRUE, weighted = FALSE)

# Two-sided paired-sample t-test
# print Cohen's d controlling for the correlation between measures
# with small sample correction factor
test.t(dat2$pre, dat2$post, paired = TRUE, effsize = TRUE, 
       weighted = FALSE, correct = TRUE)

# Two-sided paired-sample t-test
# print Cohen's d ignoring the correlation between measures
test.t(dat2$pre, dat2$post, paired = TRUE, effsize = TRUE, 
       weighted = FALSE, cor = FALSE)

# Two-sided paired-sample t-test
# do not print hypotheses and descriptive statistics
test.t(dat2$pre, dat2$post, paired = TRUE, hypo = FALSE, descript = FALSE)

# Two-sided paired-sample t-test
# population standard deviation of difference score = 1.2
# print descriptive statistics with 3 digits and p-value with 5 digits
test.t(dat2$pre, dat2$post, paired = TRUE, digits = 3, 
       p.digits = 5)

## Not run:
# Two-sided paired-sample t-test
# Plot results
test.t(dat2$pre, dat2$post, paired = TRUE, plot = TRUE)

# Load ggplot2 package
library(ggplot2)

# Two-sided paired-sample t-test
# Extract plot
p <- test.t(dat2$pre, dat2$post, paired = TRUE, output = FALSE)$plot
p

# Save plot, ggsave() from the ggplot2 package
ggsave("Paired-sample_t-test.png", dpi = 600, width = 3, height = 6)

# Extract data used to plot results
plotdat <- data.frame(test.t(dat2$pre, dat2$post, paired = TRUE, output = FALSE)$data)

# Difference score
plotdat$diff <- plotdat$y - plotdat$x

# Draw plot in line with the default setting of test.t()
ggplot(plotdat, aes(0, diff)) + 
geom_point(stat = "summary", fun = "mean", size = 4) + 
stat_summary(fun.data = "mean_cl_normal", geom = "errorbar", width = 0.20) + 
scale_x_discrete(name = NULL) + scale_y_continuous(name = NULL) + 
geom_hline(yintercept = 0, linetype = 3, size = 0.8) + 
labs(subtitle = "Two-Sided 95")
theme_bw() + theme(plot.subtitle = element_text(hjust = 0.5), 
axis.text.x = element_blank(), 
axis.ticks.x = element_blank())
### End(Not run)

---

#### test.welch

### Welch’s Test

**Description**

This function performs Welch’s two-sample t-test and Welch’s ANOVA including Games-Howell post hoc test for multiple comparison and provides descriptive statistics, effect size measures, and a plot showing error bars for confidence intervals with jitted data points.

**Usage**

```r
test.welch(formula, data, alternative = c("two.sided", "less", "greater"),
            posthoc = TRUE, conf.level = 0.95, hypo = TRUE, descript = TRUE,
            effsize = FALSE, weighted = FALSE, ref = NULL, correct = FALSE,
            plot = FALSE, point.size = 4, error.width = 0.1,
            xlab = NULL, ylab = NULL, ylim = NULL, breaks = ggplot2::waiver(),
            jitter = TRUE, jitter.size = 1.25, jitter.width = 0.05, jitter.alpha = 0.1,
            title = "", subtitle = "Confidence Interval",
            digits = 2, p.digits = 4, 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`.

- `alternative`: a character string specifying the alternative hypothesis, must be one of code"two.sided" (default), "greater" or "less". Note that this argument is only used when conducting Welch’s two-sample t-test.

- `posthoc`: logical: if TRUE, Games-Howell post hoc test for multiple comparison is conducted when performing Welch’s ANOVA.

- `conf.level`: a numeric value between 0 and 1 indicating the confidence level of the interval for Cohen’s d. Note that this argument is only used when conducting Welch’s two-sample t-test.

- `hypo`: logical: if TRUE, null and alternative hypothesis are shown on the console.

- `descript`: logical: if TRUE, descriptive statistics are shown on the console.

- `effsize`: logical: if TRUE, effect size measure Cohen’s d for Welch’s two-sample t-test (see `cohens.d`), $\eta^2$ and $\omega^2$ for Welch’s ANOVA and Cohen’s d for the post hoc tests are shown on the console.

- `weighted`: logical: if TRUE, the weighted pooled standard deviation is used to compute Cohen’s d.
test.welch

ref a numeric value or character string indicating the reference group. The standard
deviation of the reference group is used to standardized the mean difference to
compute Cohen’s d.
correct logical: if TRUE, correction factor to remove positive bias in small samples is
used.
plot logical: if TRUE, a plot showing error bars for confidence intervals is drawn.
point.size a numeric value indicating the size aesthetic for the point representing the mean
value.
error.width a numeric value indicating the horizontal bar width of the error bar.
xlab a character string specifying the labels for the x-axis.
ylab a character string specifying the labels for the y-axis.
ylim a numeric vector of length two specifying limits of the limits of the y-axis.
breaks a numeric vector specifying the points at which tick-marks are drawn at the y-
axis.
jitter logical: if TRUE (default), jittered data points are drawn.
jitter.size a numeric value indicating the size aesthetic for the jittered data points.
jitter.width a numeric value indicating the amount of vertical and horizontal jitter.
jitter.alpha a numeric value indicating the opacity of the jittered data points.
title a character string specifying the text for the title for the plot.
subtitle a character string specifying the text for the subtitle for the plot.
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.
... further arguments to be passed to or from methods.

Details

Note that by default Welch’s two-sample t-test and Games-Howell post hoc test reports Cohen’s
d based on the unweighted standard deviation (i.e., weighted = FALSE) when requesting an effect
size measure (i.e., effsize = TRUE) following the recommendation by Delacre et al. (2021).

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(s) (result).
Author(s)
Takuya Yanagida <takuya.yanagida@univie.ac.at>

References


See Also
test.t, test.z, test.levene, aov.b, cohens.d, ci.mean.diff, ci.mean

Examples

dat1 <- data.frame(group1 = c(1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2),
                  group2 = c(1, 1, 1, 2, 2, 2, 3, 3, 3, 3, 3, 3),
y = c(3, 1, 4, 2, 5, 3, 2, 3, 6, 6, 3, NA))

#--------------------------------------
# Two-Sample Design
# Two-sided two-sample Welch-test
test.welch(y ~ group1, data = dat1)
# One-sided two-sample Welch-test
test.welch(y ~ group1, data = dat1, alternative = "greater")
# Two-sided two-sample Welch-test
# print Cohen's d with weighted pooled SD
test.welch(y ~ group1, data = dat1, effsize = TRUE)
# Two-sided two-sample Welch-test
# print Cohen's d with unweighted pooled SD
test.welch(y ~ group1, data = dat1, effsize = TRUE, weighted = FALSE)
# Two-sided two-sample Welch-test
# print Cohen's d with weighted pooled SD and
# small sample correction factor
test.welch(y ~ group1, data = dat1, effsize = TRUE, correct = TRUE)
# Two-sided two-sample Welch-test
# print Cohen's d with SD of the reference group 1
test.welch(y ~ group1, data = dat1, effsize = TRUE, ref = 1)
# Two-sided two-sample Welch-test
# print Cohen's d with weighted pooled SD and
# small sample correction factor
test.welch(y ~ group1, data = dat1, effsize = TRUE,
correct = TRUE)
# Two-sided two-sample Welch-test
# do not print hypotheses and descriptive statistics,
test.welch(y ~ group1, data = dat1, descript = FALSE, hypo = FALSE)

# Two-sided two-sample Welch-test
# print descriptive statistics with 3 digits and p-value with 5 digits
test.welch(y ~ group1, data = dat1, digits = 3, p.digits = 5)

## Not run:
# Two-sided two-sample Welch-test
# plot results
test.welch(y ~ group1, data = dat1, plot = TRUE)

# Load ggplot2 package
library(ggplot2)

# Save plot, ggsave() from the ggplot2 package
ggsave("Two-sample_Welch-test.png", dpi = 600, width = 4, height = 6)

# Two-sided two-sample Welch-test
# extract plot
p <- test.welch(y ~ group1, data = dat1, output = FALSE)$plot

# Extract data
plotdat <- test.welch(y ~ group1, data = dat1, output = FALSE)$data

# Draw plot in line with the default setting of test.welch()
ggplot(plotdat, aes(factor(group), y)) +
  geom_point(stat = "summary", fun = "mean", size = 4) +
  stat_summary(fun.data = "mean_cl_normal", geom = "errorbar", width = 0.20) +
  scale_x_discrete(name = NULL) +
  labs(subtitle = "Two-Sided 95")

## End(Not run)

#-----------------------------
# Multiple-Sample Design

# Welch's ANOVA
test.welch(y ~ group2, data = dat1)

# Welch's ANOVA
# print eta-squared and omega-squared
test.welch(y ~ group2, data = dat1, effsize = TRUE)

# Welch's ANOVA
# do not print hypotheses and descriptive statistics,
test.welch(y ~ group2, data = dat1, descript = FALSE, hypo = FALSE)
## Not run:
# Welch's ANOVA
# plot results
test.welch(y ~ group2, data = dat1, plot = TRUE)

# Load ggplot2 package
library(ggplot2)

# Save plot, ggsave() from the ggplot2 package
ggsave("Multiple-sample_Welch-test.png", dpi = 600, width = 4.5, height = 6)

# Welch's ANOVA
# extract plot
p <- test.welch(y ~ group2, data = dat1, output = FALSE)$plot

# Extract data
plotdat <- test.welch(y ~ group2, data = dat1, output = FALSE)$data

# Draw plot in line with the default setting of test.welch()
ggplot(plotdat, aes(group, y)) +
  geom_point(stat = "summary", fun = "mean", size = 4) +
  stat_summary(fun.data = "mean_cl_normal", geom = "errorbar", width = 0.20) +
  scale_x_discrete(name = NULL) +
  labs(subtitle = "Two-Sided 95")
theme_bw() + theme(plot.subtitle = element_text(hjust = 0.5))

## End(Not run)

---

### test.z

**z-Test**

#### Description

This function performs one-sample, two-sample, and paired-sample z-tests and provides descriptive statistics, effect size measure, and a plot showing error bars for confidence intervals with jittered data points.

#### Usage

```r
test.z(x, ...
```

## Default S3 method:
test.z(x, y = NULL, sigma = NULL, sigma2 = NULL, mu = 0,
  paired = FALSE, alternative = c("two.sided", "less", "greater"),
  conf.level = 0.95, hypo = TRUE, descript = TRUE, effsize = FALSE,
  plot = FALSE, point.size = 4, error.width = 0.1,
  xlab = NULL, ylab = NULL, ylim = NULL, breaks = ggplot2::waiver(),
  line = TRUE, line.type = 3, line.size = 0.8, jitter = TRUE,
...
test.z

jitter.size = 1.25, jitter.width = 0.05, jitter.alpha = 0.1,
title = "", subtitle = "Confidence Interval",
digits = 2, p.digits = 4, as.na = NULL, check = TRUE,
output = TRUE, ...)

## S3 method for class 'formula'
test.z(formula, data, sigma = NULL, sigma2 = NULL,
alternative = c("two.sided", "less", "greater"), conf.level = 0.95,
hypo = TRUE, descript = TRUE, effsize = FALSE,
plot = FALSE, point.size = 4, error.width = 0.1,
xlab = NULL, ylab = NULL, ylim = NULL, breaks = ggplot2::waiver(),
jitter = TRUE, jitter.size = 1.25, jitter.width = 0.05, jitter.alpha = 0.1,
title = "", subtitle = "Confidence Interval",
digits = 2, p.digits = 4, 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 variances 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".
hypo logical: if TRUE, null and alternative hypothesis are shown on the console.
descript logical: if TRUE, descriptive statistics are shown on the console.
effsize logical: if TRUE, effect size measure Cohen's d is shown on the console.
conf.level a numeric value between 0 and 1 indicating the confidence level of the interval.
plot logical: if TRUE, a plot showing error bars for confidence intervals is drawn.
point.size a numeric value indicating the size aesthetic for the point representing the mean value.
error.width a numeric value indicating the horizontal bar width of the error bar.
xlab a character string specifying the labels for the x-axis.
ylab a character string specifying the labels for the y-axis.

ylim a numeric vector of length two specifying limits of the limits of the y-axis.

breaks a numeric vector specifying the points at which tick-marks are drawn at the y-axis.

line logical: if TRUE (default), a horizontal line is drawn at mu for the one-sample t-test or at 0 for the paired-sample t-test.

line.type an integer value or character string specifying the line type for the line representing the population mean under the null hypothesis, i.e., 0 = blank, 1 = solid, 2 = dashed, 3 = dotted, 4 = dotdash, 5 = longdash, 6 = twodash.

line.size a numeric value indicating the size aesthetic for the line representing the population mean under the null hypothesis.

jitter logical: if TRUE (default), jittered data points are drawn.

jitter.size a numeric value indicating the size aesthetic for the jittered data points.

jitter.width a numeric value indicating the amount of vertical and horizontal jitter.

jitter.alpha a numeric value indicating the opacity of the jittered data points.

title a character string specifying the text for the title for the plot.

subtitle a character string specifying the text for the subtitle for the plot.

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.

Details

Cohen’s d reported when argument effsize = TRUE is based on the population standard deviation specified in sigma or the square root of the population variance specified in sigma2. In a one-sample and paired-sample design, Cohen’s d is the mean of the difference scores divided by the population standard deviation of the difference scores (i.e., equivalent to Cohen’s dz according to Lakens, 2013). In a two-sample design, Cohen’s d is the difference between means of the two groups of observations divided by either the population standard deviation when assuming and specifying equal standard deviations or the unweighted pooled population standard deviation when assuming and specifying unequal standard deviations.
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


See Also

test.t, aov.b, test.welch, cohens.d, ci.mean.diff, ci.mean

Examples

dat1 <- 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))

#--------------------------------------
# One-Sample Design
#--------------------------------------
# Two-sided one-sample z-test
# population mean = 3, population standard deviation = 1.2
test.z(dat1$x, sigma = 1.2, mu = 3)

# Two-sided one-sample z-test
# population mean = 3, population variance = 1.44
test.z(dat1$x, sigma2 = 1.44, mu = 3)

# One-sided one-sample z-test
# population mean = 3, population standard deviation = 1.2
# convert value 3 to NA
test.z(dat1$x, sigma = 1.2, mu = 3, as.na = 3)

# Two-sided one-sample z-test
# population mean = 3, population standard deviation = 1.2
# print Cohen's d
test.z(dat1$x, sigma = 1.2, mu = 3, effsize = TRUE)

# Two-sided one-sample z-test
population mean = 3, population standard deviation = 1.2
do not print hypotheses and descriptive statistics
test.z(dat1$x, sigma = 1.2, mu = 3, hypo = FALSE, descript = FALSE)

# Two-sided one-sample z-test
# population mean = 3, population standard deviation = 1.2
# print descriptive statistics with 3 digits and p-value with 5 digits
test.z(dat1$x, sigma = 1.2, mu = 3, digits = 3, p.digits = 5)

## Not run:
# Two-sided one-sample z-test
# population mean = 3, population standard deviation = 1.2
# plot results
test.z(dat1$x, sigma = 1.2, mu = 3, plot = TRUE)

# Load ggplot2 package
library(ggplot2)

# Save plot, ggsave() from the ggplot2 package
ggsave("One-sample_z-test.png", dpi = 600, width = 3, height = 6)

# Two-sided one-sample z-test
# population mean = 3, population standard deviation = 1.2
# extract plot
p <- test.z(dat1$x, sigma = 1.2, mu = 3, output = FALSE)$plot

# Extract data
plotdat <- data.frame(test.z(dat1$x, sigma = 1.2, mu = 3, output = FALSE)$data[[1]])

# Extract results
result <- test.z(dat1$x, sigma = 1.2, mu = 3, output = FALSE)$result

# Draw plot in line with the default setting of test.z()

## End(Not run)

#--------------------------------------
# Two-Sample Design
# Two-sided two-sample z-test
# population standard deviation (SD) = 1.2, equal SD assumption

# population mean = 3, population standard deviation = 1.2
do not print hypotheses and descriptive statistics
test.z(dat1$x, sigma = 1.2, mu = 3, hypo = FALSE, descript = FALSE)

# Two-sided one-sample z-test
# population mean = 3, population standard deviation = 1.2
# print descriptive statistics with 3 digits and p-value with 5 digits
test.z(dat1$x, sigma = 1.2, mu = 3, digits = 3, p.digits = 5)

## Not run:
# Two-sided one-sample z-test
# population mean = 3, population standard deviation = 1.2
# plot results
test.z(dat1$x, sigma = 1.2, mu = 3, plot = TRUE)

# Load ggplot2 package
library(ggplot2)

# Save plot, ggsave() from the ggplot2 package
ggsave("One-sample_z-test.png", dpi = 600, width = 3, height = 6)

# Two-sided one-sample z-test
# population mean = 3, population standard deviation = 1.2
# extract plot
p <- test.z(dat1$x, sigma = 1.2, mu = 3, output = FALSE)$plot

# Extract data
plotdat <- data.frame(test.z(dat1$x, sigma = 1.2, mu = 3, output = FALSE)$data[[1]])

# Extract results
result <- test.z(dat1$x, sigma = 1.2, mu = 3, output = FALSE)$result

# Draw plot in line with the default setting of test.z()

## End(Not run)

#--------------------------------------
# Two-Sample Design
# Two-sided two-sample z-test
# population standard deviation (SD) = 1.2, equal SD assumption

# population mean = 3, population standard deviation = 1.2
do not print hypotheses and descriptive statistics
test.z(dat1$x, sigma = 1.2, mu = 3, hypo = FALSE, descript = FALSE)

# Two-sided one-sample z-test
# population mean = 3, population standard deviation = 1.2
# print descriptive statistics with 3 digits and p-value with 5 digits
test.z(dat1$x, sigma = 1.2, mu = 3, digits = 3, p.digits = 5)

## Not run:
# Two-sided one-sample z-test
# population mean = 3, population standard deviation = 1.2
# plot results
test.z(dat1$x, sigma = 1.2, mu = 3, plot = TRUE)

# Load ggplot2 package
library(ggplot2)

# Save plot, ggsave() from the ggplot2 package
ggsave("One-sample_z-test.png", dpi = 600, width = 3, height = 6)

# Two-sided one-sample z-test
# population mean = 3, population standard deviation = 1.2
# extract plot
p <- test.z(dat1$x, sigma = 1.2, mu = 3, output = FALSE)$plot

# Extract data
plotdat <- data.frame(test.z(dat1$x, sigma = 1.2, mu = 3, output = FALSE)$data[[1]])

# Extract results
result <- test.z(dat1$x, sigma = 1.2, mu = 3, output = FALSE)$result

# Draw plot in line with the default setting of test.z()

## End(Not run)

#--------------------------------------
# Two-Sample Design
# Two-sided two-sample z-test
# population standard deviation (SD) = 1.2, equal SD assumption
test.z(x ~ group, sigma = 1.2, data = dat1)

# Two-sided two-sample z-test
# population standard deviation (SD) = 1.2 and 1.5, unequal SD assumption
test.z(x ~ group, sigma = c(1.2, 1.5), data = dat1)

# Two-sided two-sample z-test
# population variance (Var) = 1.44 and 2.25, unequal Var assumption
test.z(x ~ group, sigma2 = c(1.44, 2.25), data = dat1)

# One-sided two-sample z-test
# population standard deviation (SD) = 1.2, equal SD assumption
# print Cohen's d
# test.z(x ~ group, sigma = 1.2, data = dat1, alternative = "greater")

test.z(x ~ group, sigma = 1.2, data = dat1, effsize = TRUE)

# Two-sided two-sample z-test
# population standard deviation (SD) = 1.2, equal SD assumption
# do not print hypotheses and descriptive statistics,
# print Cohen's d
# test.z(x ~ group, sigma = 1.2, data = dat1, descript = FALSE, hypo = FALSE)

# Two-sided two-sample z-test
# population mean = 3, population standard deviation = 1.2
# print descriptive statistics with 3 digits and p-value with 5 digits
test.z(x ~ group, sigma = 1.2, data = dat1, digits = 3, p.digits = 5)

## Not run:
# Two-sided two-sample z-test
# population standard deviation (SD) = 1.2, equal SD assumption
# plot results
test.z(x ~ group, sigma = 1.2, data = dat1, plot = TRUE)

# Load ggplot2 package
library(ggplot2)

ggsave("Two-sample_z-test.png", dpi = 600, width = 4, height = 6)

# Two-sided two-sample z-test
# population standard deviation (SD) = 1.2, equal SD assumption
# extract plot
p <- test.z(x ~ group, sigma = 1.2, data = dat1, output = FALSE)$plot
p

## End(Not run)

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

# Two-sided two-sample z-test
# population standard deviation (SD) = 1.2, equal SD assumption
test.z(group1, group2, sigma = 1.2)

#--------------------------------------
# Paired-Sample Design

dat2 <- data.frame(pre = c(1, 3, 2, 5, 7),
                   post = c(2, 2, 1, 6, 8), stringsAsFactors = FALSE)

# Two-sided paired-sample z-test
# population standard deviation of difference score = 1.2
# test.z(dat2$pre, dat2$post, sigma = 1.2, paired = TRUE)

# Two-sided paired-sample z-test
# population variance of difference score = 1.44
# test.z(dat2$pre, dat2$post, sigma2 = 1.44, paired = TRUE)

# One-sided paired-sample z-test
# population standard deviation of difference score = 1.2
# test.z(dat2$pre, dat2$post, sigma = 1.2, paired = TRUE, alternative = "greater")

# Two-sided paired-sample z-test
# population standard deviation of difference score = 1.2
# convert value 1 to NA
# test.z(dat2$pre, dat2$post, sigma = 1.2, as.na = 1, paired = TRUE)

# Two-sided paired-sample z-test
# population standard deviation of difference score = 1.2
# print Cohen’s d
# test.z(dat2$pre, dat2$post, sigma = 1.2, paired = TRUE, effsize = TRUE)

# Two-sided paired-sample z-test
# population standard deviation of difference score = 1.2
# do not print hypotheses and descriptive statistics
# test.z(dat2$pre, dat2$post, sigma = 1.2, mu = 3, paired = TRUE, hypo = FALSE, descript = FALSE)

# Two-sided paired-sample z-test
# population standard deviation of difference score = 1.2
# print descriptive statistics with 3 digits and p-value with 5 digits
# test.z(dat2$pre, dat2$post, sigma = 1.2, paired = TRUE, digits = 3, p.digits = 5)

## Not run:
# Two-sided paired-sample z-test
# population standard deviation of difference score = 1.2
# plotresults
# test.z(dat2$pre, dat2$post, sigma = 1.2, paired = TRUE, plot = TRUE)
# Load ggplot2 package
library(ggplot2)

# Save plot, ggsave() from the ggplot2 package
ggsave("Paired-sample_z-test.png", dpi = 600, width = 3, height = 6)

# Two-sided paired-sample z-test
# population standard deviation of difference score = 1.2
# extract plot
p <- test.z(dat2$pre, dat2$post, sigma = 1.2, paired = TRUE, output = FALSE)$plot
p

# Extract data
plotdat <- data.frame(test.z(dat2$pre, dat2$post, sigma = 1.2, paired = TRUE,
output = FALSE)$data)

# Difference score
plotdat$diff <- plotdat$y - plotdat$x

# Extract results
result <- test.z(dat2$pre, dat2$post, sigma = 1.2, paired = TRUE,
output = FALSE)$result

# Draw plot in line with the default setting of test.t()
ggplot(plotdat, aes(0, diff)) +
  geom_point(data = result, aes(x = 0, m.diff), size = 4) +
  geom_errorbar(data = result,
                aes(x = 0L, y = m.diff, ymin = m.low, ymax = m.upp), width = 0.2) +
  scale_x_continuous(name = NULL, limits = c(-2, 2)) +
  scale_y_continuous(name = "y") +
  geom_hline(yintercept = 0, linetype = 3, size = 0.8) +
  labs(subtitle = "Two-Sided 95")
theme_bw() + theme(plot.subtitle = element_text(hjust = 0.5),
  axis.text.x = element_blank(),
  axis.ticks.x = element_blank())

## End(Not run)

write.mplus

---

**Write Mplus Data File**

**Description**

This function writes a matrix or data frame to a tab-delimited file without variable names, a Mplus input template, and a text file with variable names. Note that only numeric variables are allowed, i.e., non-numeric variables will be removed from the data set. Missing data will be coded as a single numeric value.

**Usage**

```r
write.mplus(x, file = "Mplus_Data.dat", input = TRUE, n.var = 8,
            var = 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".

input logical: if TRUE (default), Mplus input template is written in a text file named according to the argument file with the extension _INPUT.inp.

n.var a numeric value indicating the number of variables in each line under NAMES ARE in the Mplus input template.

var logical: if TRUE, variable names are written in a text file named according to the argument file with the extension _VARNAMES.txt.

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


See Also

read.mplus, run.mplus

Examples

## Not run:

# Write Mplus Data File and a Mplus input template
write.mplus(mtcars)

# Write Mplus Data File "mtcars.dat" and a Mplus input template "mtcars_INPUT.inp", 
# missing values coded with -999, 4 variables in each line under "NAMES ARE"
# write variable names in a text file called "mtcars_VARNAMES.inp"
write.mplus(mtcars, file = "mtcars.dat", n.var = 4, var = TRUE, na = -999)

## End(Not run)
### Description

This function writes the results of a misty object (misty.object) into an Excel file.

### Usage

```r
write.result(x, file = "Results.xlsx")
```

### Arguments

- **x**: misty object (misty.object) resulting from a misty function supported by the `write.result` function (see 'Details').
- **file**: a character string naming a file with or without file extension `.xlsx`, e.g., "Results.xlsx" or "Results".

### Details

Currently the function supports result objects from the function `cor.matrix`, `crosstab`, `freq`, `item.alpha`, `item.alpha`, `item.cfa`, `item.omega`, `multilevel.cor`, `multilevel.descript`, `na.coverage`, `na.descript`, and `na.pattern`.

### Author(s)

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

### See Also

- `cor.matrix`
- `crosstab`
- `freq`
- `item.alpha`
- `item.cfa`
- `item.omega`
- `multilevel.cor`
- `multilevel.descript`
- `na.coverage`
- `na.descript`
- `na.pattern`

### Examples

```r
# Not run:
#--------------------------------------------------
# cor.matrix() function

correlation <- cor.matrix(mtcars, print = "all", output = FALSE)
write.result(correlation, "Correlation.xlsx")

#--------------------------------------------------
# crosstab() function

crosstab <- crosstab(mtcars[, c("carb", "gear")], print = "all", output = FALSE)
write.result(crosstab, "Crosstab.xlsx")
```

#--------------------------------------------------
# descript() function
result <- descript(mtcars, output = FALSE)
write.result(result, "Descript.xlsx")

#--------------------------------------
# freq() function
result <- freq(mtcars, exclude = 99, output = FALSE)
write.result(result, "Freq.xlsx")

#--------------------------------------
# item.alpha() function
result <- item.alpha(attitude, output = FALSE)
write.result(result, "Alpha.xlsx")

#--------------------------------------
# item.cfa() function
load(data("HolzingerSwineford1939", package = "lavaan")
result <- item.cfa(HolzingerSwineford1939[, c("x1", "x2", "x3")],
                   output = FALSE)
write.result(result, "CFA.xlsx")

#--------------------------------------
# item.omega() function
result <- item.omega(attitude, output = FALSE)
write.result(result, "Omega.xlsx")

#--------------------------------------
# multilevel.cor() function
load(data("Demo.twolevel", package = "lavaan")
result <- multilevel.cor(Demo.twolevel[, c("y1", "y2", "y3")],
                         cluster = Demo.twolevel$cluster, output = FALSE)
write.result(result, "Multilevel_Correlation.xlsx")

#--------------------------------------
# multilevel.descript() function
load(data("Demo.twolevel", package = "lavaan")
result <- multilevel.descript(Demo.twolevel[, c("y1", "y2", "y3")],
                               cluster = Demo.twolevel$cluster, output = FALSE)
write.result(result, "Multilevel_Descript.xlsx")
#--------------------------------------
# na.coverage() function
#--------------------------------------

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

result <- na.coverage(dat, output = FALSE)
write.result(result, "NA_Coverage.xlsx")
```

#--------------------------------------
# na.descript() function
#--------------------------------------

```r
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))

result <- na.descript(dat, table = TRUE, output = FALSE)
write.result(result, "NA_Descriptives.xlsx")
```

#--------------------------------------
# na.pattern() function
#--------------------------------------

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

result <- na.pattern(dat, output = FALSE)
write.result(result, "NA_Pattern.xlsx")
```

## 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](https://www.gnu.org/software/pspp/pspp.html)).

**Usage**

```r
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

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


### See Also

read.sav

### Examples

```r
## 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), stringsAsFactors = FALSE)

# 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 attributes 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"), stringsAsFactors = FALSE)

# Write SPSS file with variable attributes using the haven package
```

write.sav
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)

### write.xlsx

**Write Excel File**

**Description**

This function calls the `write_xlsx()` function in the *writexl* package by Jeroen Ooms to write an Excel file (.xlsx).

**Usage**

```r
write.xlsx(x, file = "Excel_Data.xlsx", col.names = TRUE, format = FALSE,
           use.zip64 = FALSE, check = TRUE)
```

**Arguments**

- `x`: a matrix, data frame or (named) list of matrices or data frames that will be written in the Excel file.
- `file`: a character string naming a file with or without file extension `.xlsx`, e.g., "My_Excel.xlsx" or "My_Excel".
- `col.names`: logical: if TRUE, column names are written at the top of the Excel sheet.
- `format`: logical: if TRUE, column names in the Excel file are centered and bold.
- `use.zip64`: logical: if TRUE, zip64 to enable support for 4GB+ Excel files is used.
- `check`: logical: if TRUE, argument specification is checked.

**Details**

This function supports strings, numbers, booleans, and dates.

**Note**

The function was adapted from the `write_xlsx()` function in the *writexl* package by Jeroen Ooms (2021).

**Author(s)**

Jeroen Ooms
References


See Also

read.xlsx

Examples

## Not run:

# Write Excel file (.xlsx)
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.xlsx(dat, file = "Excel.xlsx")

# Write Excel file with multiple sheets (.xlsx)
write.xlsx(list(cars = cars, mtcars = mtcars), file = "Excel_Sheets.xlsx")

## End(Not run)
Index

ci.mean, 5, 14, 18, 22, 25, 28, 31, 33, 58, 128, 151, 158, 163

ci.mean.diff, 5, 14, 18, 22, 25, 28, 31, 33, 58, 126

ci.median, 5, 14, 18, 21, 25, 28, 31, 33, 58, 126

ci.prop, 5, 14, 18, 22, 23, 28, 31, 33, 58, 126

ci.prop.diff, 22, 25, 26, 31, 33, 58, 126

ci.sd, 15, 18, 22, 25, 28, 30, 33, 58, 126

ci.var, 15, 18, 22, 25, 28, 31, 32, 58, 126

cluster.scores, 9, 35, 93, 97, 136

cohens.d, 4, 5, 16, 22, 25, 28, 31, 33, 58, 126, 150, 151, 158, 163

collin.diag, 42, 126

cor.cont, 39, 45, 48, 50, 52, 54, 70, 126

cor.cramer, 39, 46, 47, 50, 52, 54, 70, 126

cor.matrix, 39, 46, 48, 48, 52, 54, 70, 126, 169

cor.phi, 46, 48, 50, 51, 54, 70, 126

cor.poly, 46, 48, 52, 53, 126

cor.test, 49

crosstab, 54, 58, 71, 126, 169

descript, 15, 18, 22, 25, 28, 31, 33, 55, 57, 71, 126

df.duplicated, 59, 62, 64, 65, 67

df.merge, 60, 61, 64, 65, 67

df.rbind, 60, 62, 63, 65, 67

df.rename, 60, 62, 64, 65, 67

df.sort, 60, 62, 64, 65, 66, 126

df.unique, 60, 62, 64, 65, 67

df.unique(df.duplicated), 59

dummy.c, 9, 67

df.unique(df.duplicated), 59

dummy.c, 9, 67

df.unique(df.duplicated), 59

dummy.c, 9, 67

df.unique(df.duplicated), 59

dummy.c, 9, 67

event.test, 49

event.type, 49

freq. 55, 58, 70, 125, 126, 169

item.reverse, 9, 78, 89, 90, 132

item.scores, 9, 35, 78, 85, 89, 91, 91

kurtosis, 93, 143

multilevel.cor, 50, 94, 99, 101, 111, 169

multilevel.descript, 35, 55, 58, 71, 97, 98, 101, 111, 126, 169

multilevel.icc, 35, 50, 97, 99, 100, 111

multilevel.indirect, 76, 99, 102, 111

multilevel.r2, 99, 105, 126

na.as, 7, 113, 116–118, 120–122, 124

na.auxiliary, 7, 39, 50, 113, 114, 117, 118, 120–122, 124, 126, 126

na.coverage, 7, 113, 116, 116, 118, 120–122, 124, 126, 169

na.descript, 7, 55, 58, 71, 113, 116, 117, 117, 120–122, 124, 126, 169

na.indicator, 7, 113, 116–118, 119, 121, 122, 124

na.pattern, 7, 113, 116–118, 120, 120, 122, 124, 126, 169

na.prop, 7, 113, 116–118, 120, 121, 121, 124

na.test, 7, 113, 116–118, 120–122, 122

p.adjust, 49, 96

print.misty.object, 124

rbind, 64

read.mplus, 126, 128, 130, 168

read.sav, 127, 127, 130, 173

read.xlsx, 127, 128, 129, 175
INDEX

rec, 9, 91, 131
run.mplus, 127, 133, 168
rwg.lindell, 9, 135

size.cor, 50, 126, 137, 139, 141
size.mean, 126, 138, 138, 141
size.prop, 126, 138, 139, 140
skewness, 94, 142
std.coef, 143

test.levene, 5, 126, 146, 151, 158
test.t, 5, 15, 125, 126, 148, 149, 158, 163
test.welch, 5, 125, 126, 148, 151, 156, 163
test.z, 5, 15, 125, 126, 151, 158, 160

write.mplus, 127, 167
write.result, 50, 55, 71, 78, 89, 97, 99, 117, 118, 121, 169
write.sav, 128, 171
write.xlsx, 130, 174