Package ‘rstatix’

June 18, 2020

Type Package

Title Pipe-Friendly Framework for Basic Statistical Tests

Version 0.6.0

Description Provides a simple and intuitive pipe-friendly framework, coherent with the 'tidyverse' design philosophy, for performing basic statistical tests, including t-test, Wilcoxon test, ANOVA, Kruskal-Wallis and correlation analyses.

The output of each test is automatically transformed into a tidy data frame to facilitate visualization.

Additional functions are available for reshaping, reordering, manipulating and visualizing correlation matrix.

Functions are also included to facilitate the analysis of factorial experiments, including purely 'within-Ss' designs (repeated measures), purely 'between-Ss' designs, and mixed 'within-and-between-Ss' designs.

It's also possible to compute several effect size metrics, including "eta squared" for ANOVA, "Cohen's d" for t-test and 'Cramer V' for the association between categorical variables.

The package contains helper functions for identifying univariate and multivariate outliers, assessing normality and homogeneity of variances.

License GPL-2

LazyData TRUE

Encoding UTF-8

Depends R (>= 3.3.0)

Imports stats, utils, tidyr (>= 1.0.0), purrr, broom (>= 0.5.6), rlang (>= 0.3.1), tibble (>= 2.1.3), dplyr (>= 0.7.1), magrittr, corplot, tidyselect (>= 1.0.0), car, generics (>= 0.0.2)

Suggests knitr, rmarkdown, ggpubr, graphics, emmeans, coin, boot, testthat, spelling

URL https://rpkgs.datanovia.com/rstatix/

BugReports https://github.com/kassambara/rstatix/issues

RoxygenNote 7.1.0
Collate 'utilities.R' 'add_significance.R' 'adjust_pvalue.R'
'factorial_design.R' 'utilities_two_sample_test.R'
'anova_summary.R' 'anova_test.R' 'as_cor_mat.R' 'binom_test.R'
'box_m.R' 'chisq_test.R' 'cochran_qtest.R' 'cohens_d.R'
'cor_as_symbols.R' 'replace_triangle.R' 'pull_triangle.R'
'cor_mark_significant.R' 'cor_mat.R' 'cor_plot.R'
'cor_reorder.R' 'cor_reshape.R' 'cor_select.R' 'cor_test.R'
'counts_to_cases.R' 'cramer_v.R' 'df.R' 'doo.R' 't_test.R'
'dunn_test.R' 'emmeans_test.R' 'eta_squared.R' 'factors.R'
'fisher_test.R' 'freq_table.R' 'friedman_test.R'
'friedman_effsize.R' 'games_howell_test.R' 'get_comparisons.R'
'get_manova_table.R' 'get_mode.R' 'get_pvalue_position.R'
'get_summary_stats.R' 'get_test_label.R' 'kruskal_effsize.R'
'kruskal_test.R' 'levene_test.R' 'mahalanobis_distance.R'
'make_clean_names.R' 'menemar_test.R' 'multinom_test.R'
'outliers.R' 'p_value.R' 'prop_test.R' 'prop_trend_test.R'
'reexports.R' 'sample_n_by.R' 'shapiro_test.R' 'sign_test.R'
'tukey_hsd.R' 'utils-manova.R' 'utils-pipe.R'
'welch_anova_test.R' 'wilcoxon_effsize.R' 'wilcox_test.R'

Language en-US
NeedsCompilation no
Author Alboukadel Kassambara [aut, cre]
Maintainer Alboukadel Kassambara <alboukadel.kassambara@gmail.com>
Repository CRAN
Date/Publication 2020-06-18 20:20:02 UTC

R topics documented:

add_significance .................................................. 4
adjust_pvalue .......................................................... 5
anova_summary ........................................................ 5
anova_test ............................................................ 7
as_cor_mat ............................................................. 11
binom_test ............................................................. 12
box_m ................................................................. 14
chisq_test ............................................................. 15
cochran_qtest ........................................................ 18
cohens_d ............................................................... 19
convert_as_factor .................................................... 21
cor_as_symbols ....................................................... 23
cor_gather ............................................................. 24
cor_mark_significant ................................................ 25
cor_mat ................................................................. 26
cor_plot ............................................................... 28
cor_reorder .......................................................... 30
cor_select ............................................................ 31


R topics documented:

cor.test ................................................................. 32
counts_to_cases .................................................... 34
cramer_y ................................................................. 35
df_arrange .............................................................. 36
df_get_var_names .................................................... 37
df_group_by ........................................................... 37
df_label_both ........................................................ 38
df_nest_by ............................................................. 39
df_select ................................................................. 40
df_split_by ............................................................. 41
df_unite ................................................................. 42
doo .......................................................................... 43
dunn_test ............................................................... 44
doemmeans_test ...................................................... 46
df_squared ............................................................. 48
factorial_design ..................................................... 49
fisher_test ............................................................. 50
freq_table .............................................................. 53
friedman_effsize ..................................................... 54
friedman_test ........................................................ 56
games_howell_test .................................................. 57
getComparisons ...................................................... 58
get_mode ............................................................... 59
get_pwc_label ......................................................... 60
get_summary_stats ................................................... 63
get_y_position ......................................................... 64
identify_outliers .................................................... 67
kruskal_effsize ....................................................... 68
kruskal_test .......................................................... 70
levene_test ........................................................... 71
mahalanobis_distance ............................................... 72
make_clean_names ................................................... 73
mcnemar_test ......................................................... 74
multinom_test ........................................................ 76
prop_test ............................................................... 77
prop_trend_test ....................................................... 80
pull_triangle .......................................................... 81
p_round ................................................................. 83
replace_triangle ..................................................... 85
sample_n_by ........................................................ 86
shapiro_test .......................................................... 87
sign_test ............................................................... 88
tukey_hsd .............................................................. 90
t_test ................................................................. 92
welch_anova_test ................................................... 95
wilcox_effsize ....................................................... 96
wilcox_test ........................................................... 99

Index 103
Description

Add p-value significance symbols into a data frame.

Usage

```r
add_significance(  
data,  
p.col = NULL,  
output.col = NULL,  
cutpoints = c(0, 1e-04, 0.001, 0.01, 0.05, 1),  
symbols = c("****", "***", ",**, *, "ns")
)
```

Arguments

- **data**: a data frame containing a p-value column.
- **p.col**: column name containing p-values.
- **output.col**: the output column name to hold the adjusted p-values.
- **cutpoints**: numeric vector used for intervals.
- **symbols**: character vector, one shorter than cutpoints, used as significance symbols.

Value

a data frame

Examples

```r
# Perform pairwise comparisons and adjust p-values
ToothGrowth %>%
t_test(len ~ dose) %>%
adjust_pvalue() %>%
add_significance("p.adj")
```
**adjust_pvalue**  
*Adjust P-values for Multiple Comparisons*

**Description**
A pipe-friendly function to add an adjusted p-value column into a data frame. Supports grouped data.

**Usage**
```r
adjust_pvalue(data, p.col = NULL, output.col = NULL, method = "holm")
```

**Arguments**
- `data`: a data frame containing a p-value column
- `p.col`: column name containing p-values
- `output.col`: the output column name to hold the adjusted p-values
- `method`: method for adjusting p values (see `p.adjust`). Allowed values include "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". If you don’t want to adjust the p value (not recommended), use `p.adjust.method = "none"`.

**Value**
a data frame

**Examples**
```r
# Perform pairwise comparisons and adjust p-values
ToothGrowth %>%
  t_test(len ~ dose) %>%
  adjust_pvalue()
```

---

**anova_summary**  
*Create Nice Summary Tables of ANOVA Results*

**Description**
Create beautiful summary tables of ANOVA test results obtained from either `Anova()` or `aov()`. The results include ANOVA table, generalized effect size and some assumption checks.

**Usage**
```r
anova_summary(object, effect.size = "ges", detailed = FALSE, observed = NULL)
```
Arguments

object: an object of returned by either `Anova()` or `aov()`.

effect.size: the effect size to compute and to show in the ANOVA results. Allowed values can be either "ges" (generalized eta squared) or "pes" (partial eta squared) or both. Default is "ges".

detailed: If TRUE, returns extra information (sums of squares columns, intercept row, etc.) in the ANOVA table.

observed: Variables that are observed (i.e., measured) as compared to experimentally manipulated. The default effect size reported (generalized eta-squared) requires correct specification of the observed variables.

Value

return an object of class `anova_test` a data frame containing the ANOVA table for independent measures ANOVA. However, for repeated/mixed measures ANOVA, it is a list containing the following components are returned:

- **ANOVA**: a data frame containing ANOVA results
- **Mauchly’s Test for Sphericity**: If any within-Ss variables with more than 2 levels are present, a data frame containing the results of Mauchly’s test for Sphericity. Only reported for effects that have more than 2 levels because sphericity necessarily holds for effects with only 2 levels.
- **Sphericity Corrections**: If any within-Ss variables are present, a data frame containing the Greenhouse-Geisser and Huynh-Feldt epsilon values, and corresponding corrected p-values.

The returned object might have an attribute called `args` if you compute ANOVA using the function `anova_test()`. The attribute `args` is a list holding the arguments used to fit the ANOVA model, including: data, dv, within, between, type, model, etc.

The following abbreviations are used in the different results tables:

- DFn Degrees of Freedom in the numerator (i.e., DF effect).
- DFd Degrees of Freedom in the denominator (i.e., DF error).
- SSn Sum of Squares in the numerator (i.e., SS effect).
- SSD Sum of Squares in the denominator (i.e., SS error).
- F F-value.
- p p-value (probability of the data given the null hypothesis).
- p<.05 Highlights p-values less than the traditional alpha level of .05.
- ges Generalized Eta-Squared measure of effect size.
- GGe Greenhouse-Geisser epsilon.
- p[GGe]<.05 Highlights p-values (after correction using Greenhouse-Geisser epsilon) less than the traditional alpha level of .05.
- HFe Huynh-Feldt epsilon.
- p[HFe] p-value after correction using Huynh-Feldt epsilon.
- p[HFe]<.05 Highlights p-values (after correction using Huynh-Feldt epsilon) less than the traditional alpha level of .05.
- W Mauchly’s W statistic


Author(s)
Alboukadel Kassambara, <alboukadel.kassambara@gmail.com>

See Also

anova_test(), factorial_design()

Examples

# Load data
#:::::::::::::::::::::::::::::::::::::::
data("ToothGrowth")
df <- ToothGrowth
df$dose <- as.factor(df$dose)

# Independent measures ANOVA
#:::::::::::::::::::::::::::::::::::::::::
# Compute ANOVA and display the summary
res.anova <- Anova(lm(len ~ dose*supp, data = df))
anova_summary(res.anova)

# Display both SSn and SSd using detailed = TRUE
# Show generalized eta squared using effect.size = "ges"
anova_summary(res.anova, detailed = TRUE, effect.size = "ges")

# Show partial eta squared using effect.size = "pes"
anova_summary(res.anova, detailed = TRUE, effect.size = "pes")

# Repeated measures designs using car::Anova()
#:::::::::::::::::::::::::::::::::::::::::
# Prepare the data
df$id <- as.factor(rep(1:10, 6)) # Add individuals ids
head(df)

# Easily perform repeated measures ANOVA using the car package
# Prepare the data
design <- factorial_design(df, dv = len, wid = id, within = c(supp, dose))
res.anova <- Anova(design$model, idata = design$idata, idesign = design$idesign, type = 3)
anova_summary(res.anova)

# Repeated measures designs using stats::Aov()
#:::::::::::::::::::::::::::::::::::::::::
res.anova <- aov(len ~ dose*supp + Error(id/(supp*dose)), data = df)
anova_summary(res.anova)
Description

Provides a pipe-friendly framework to perform different types of ANOVA tests, including:

- **Independent measures ANOVA**: between-Subjects designs,
- **Repeated measures ANOVA**: within-Subjects designs
- **Mixed ANOVA**: Mixed within within- and between-Subjects designs, also known as split-plot ANOVA and
- **ANCOVA**: Analysis of Covariance.

The function is an easy to use wrapper around `Anova()` and `aov()`. It makes ANOVA computation handy in R and it's highly flexible: can support model and formula as input. Variables can be also specified as character vector using the arguments dv, wid, between, within, covariate.

The results include ANOVA table, generalized effect size and some assumption checks.

Usage

```r
anova_test(
  data,
  formula,
  dv,
  wid,
  between,
  within,
  covariate,
  type = NULL,
  effect.size = "ges",
  error = NULL,
  white.adjust = FALSE,
  observed = NULL,
  detailed = FALSE
)
```

```r
get_anova_table(x, correction = c("auto", "GG", "HF", "none"))
```

## S3 method for class 'anova_test'

```r
print(x, ...)
```

## S3 method for class 'anova_test'

```r
plot(x, ...)
```

Arguments

- **data** a data.frame or a model to be analyzed.
- **formula** a formula specifying the ANOVA model similar to `aov`. Can be of the form `y ~ group` where `y` is a numeric variable giving the data values and `group` is a factor with one or multiple levels giving the corresponding groups. For example, `formula = TP53 ~ cancer_group`.
  Examples of supported formula include:
• Between-Ss ANOVA (independent measures ANOVA): $y \sim b1*b2$
• Within-Ss ANOVA (repeated measures ANOVA): $y \sim w1*w2 + Error(id/(w1*w2))$
• Mixed ANOVA: $y \sim b1*b2*w1 + Error(id/w1)$

If the formula doesn’t contain any within vars, a linear model is directly fitted and passed to the ANOVA function. For repeated designs, the ANOVA variables are parsed from the formula.

dv (numeric) dependent variable name.

wid (factor) column name containing individuals/subjects identifier. Should be unique per individual.

between (optional) between-subject factor variables.

within (optional) within-subjects factor variables

covariate (optional) covariate names (for ANCOVA)

type the type of sums of squares for ANOVA. Allowed values are either 1, 2 or 3. type = 2 is the default because this will yield identical ANOVA results as type = 1 when data are balanced but type = 2 will additionally yield various assumption tests where appropriate. When the data are unbalanced the type = 3 is used by popular commercial softwares including SPSS.

effect.size the effect size to compute and to show in the ANOVA results. Allowed values can be either "ges" (generalized eta squared) or "pes" (partial eta squared) or both. Default is "ges".

error (optional) for a linear model, an lm model object from which the overall error sum of squares and degrees of freedom are to be calculated. Read more in Anova() documentation.

white.adjust Default is FALSE. If TRUE, heteroscedasticity correction is applied to the coefficient of covariance matrix. Used only for independent measures ANOVA.

observed Variables that are observed (i.e, measured) as compared to experimentally manipulated. The default effect size reported (generalized eta-squared) requires correct specification of the observed variables.

detailed If TRUE, returns extra information (sums of squares columns, intercept row, etc.) in the ANOVA table.

x an object of class anova_test

correction character. Used only in repeated measures ANOVA test to specify which correction of the degrees of freedom should be reported for the within-subject factors. Possible values are:

• "GG": applies Greenhouse-Geisser correction to all within-subjects factors even if the assumption of sphericity is met (i.e., Mauchly’s test is not significant, p > 0.05).
• "HF": applies Hyunh-Feldt correction to all within-subjects factors even if the assumption of sphericity is met,
• "none": returns the ANOVA table without any correction and
• "auto": apply automatically GG correction to only within-subjects factors violating the sphericity assumption (i.e., Mauchly’s test p-value is significant, p <= 0.05).

... additional arguments
Value

return an object of class anova_test a data frame containing the ANOVA table for independent measures ANOVA.

However, for repeated/mixed measures ANOVA, a list containing the following components are returned: ANOVA table, Mauchly's Test for Sphericity, Sphericity Corrections. These table are described more in the documentation of the function anova_summary().

The returned object has an attribute called args, which is a list holding the arguments used to fit the ANOVA model, including: data, dv, within, between, type, model, etc.

Functions

• anova_test: perform anova test
  • get_anova_table: extract anova table from an object of class anova_test. When within-subject factors are present, either sphericity corrected or uncorrected degrees of freedom can be reported.

Author(s)

Alboukadel Kassambara, <alboukadel.kassambara@gmail.com>

See Also

anova_summary(), factorial_design()

Examples

# Load data
#:::::::::::::::::::::::::::::::::::::::
data("ToothGrowth")
df <- ToothGrowth

# One-way ANOVA test
#:::::::::::::::::::::::::::::::::::::::
%>% anova_test(len ~ dose)

# Grouped One-way ANOVA test
#:::::::::::::::::::::::::::::::::::::::
df %>%
  group_by(supp) %>%
  anova_test(len ~ dose)

# Two-way ANOVA test
#:::::::::::::::::::::::::::::::::::::::
%>% anova_test(len ~ supp*dose)

# Two-way repeated measures ANOVA
#:::::::::::::::::::::::::::::::::::::::
df$id <- rep(1:10, 6) # Add individuals id
%>%
  Use formula
as_cor_mat

Convert a Correlation Test Data Frame into a Correlation Matrix

Description

Convert a correlation test data frame, returned by the cor_test(), into a correlation matrix format.

Usage

as_cor_mat(x)

Arguments

x an object of class cor_test.

Value

Returns a data frame containing the matrix of the correlation coefficients. The output has an attribute named "pvalue", which contains the matrix of the correlation test p-values.

See Also

cor_mat(), cor_test()
Examples

```r
# Pairwise correlation tests between variables
#:::::::::::::::::::::::::::::::::::::::::::::::
res.cor.test <- mtcars %>%
  select(mpg, disp, hp, drat, wt, qsec) %>%
  cor_test()
res.cor.test

# Convert the correlation test into a correlation matrix
#:::::::::::::::::::::::::::::::::::::::::::::::
res.cor.test %>% as_cor_mat()
```

---

### binom_test

**Exact Binomial Test**

#### Description

Performs exact binomial test and pairwise comparisons following a significant exact multinomial test. Wrapper around the R base function `link[stats]{binom.test}()` that returns a data frame as a result.

#### Usage

```r
binom_test(
  x,
  n,
  p = 0.5,
  alternative = "two.sided",
  conf.level = 0.95,
  detailed = FALSE
)
```

```r
pairwise_binom_test(
  x,
  p.adjust.method = "holm",
  alternative = "two.sided",
  conf.level = 0.95
)
```

```r
pairwise_binom_test_against_p(
  x,
  p = rep(1/length(x), length(x)),
  p.adjust.method = "holm",
  alternative = "two.sided",
  conf.level = 0.95
)
```
Arguments

x     numeric vector containing the counts.

n     number of trials; ignored if x has length 2.

p     a vector of probabilities of success. The length of p must be the same as the number of groups specified by x, and its elements must be greater than 0 and less than 1.

alternative indicates the alternative hypothesis and must be one of "two.sided", "greater" or "less". You can specify just the initial letter.

conf.level  confidence level for the returned confidence interval.

detailed   logical value. Default is FALSE. If TRUE, a detailed result is shown.

p.adjust.method method to adjust p values for multiple comparisons. Used when pairwise comparisons are performed. Allowed values include "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". If you don't want to adjust the p value (not recommended), use p.adjust.method = "none".

Value

return a data frame containing the p-value and its significance. with some the following columns:

• group, group1, group2: the categories or groups being compared.
• statistic: the number of successes.
• parameter: the number of trials.
• p: p-value of the test.
• p.adj: the adjusted p-value.
• method: the used statistical test.
• p.signif, p.adj.signif: the significance level of p-values and adjusted p-values, respectively.
• estimate: the estimated probability of success.
• alternative: a character string describing the alternative hypothesis.
• conf.low, conf.high: Lower and upper bound on a confidence interval for the probability of success.

The returned object has an attribute called args, which is a list holding the test arguments.

Functions

• binom_test: performs exact binomial test. Wrapper around the R base function binom.test that returns a dataframe as a result.
• pairwise_binom_test: performs pairwise comparisons (binomial test) following a significant exact multinomial test.
• pairwise_binom_test_against_p: performs pairwise comparisons (binomial test) following a significant exact multinomial test for given probabilities.
See Also
multinom_test

Examples

# Exact binomial test
# Data: 160 mice with cancer including 95 male and 65 female
# Q1: Does cancer affect more males than females?
binom_test(x = 95, n = 160)
# => yes, there are a significant difference

# Q2: compare the observed proportion of males
to an expected proportion (p = 3/5)
binom_test(x = 95, n = 160, p = 3/5)
# => there are no significant difference

# Multinomial test
# Data
tulip <- c(red = 81, yellow = 50, white = 27)
# Question 1: are the color equally common?
# this is a test of homogeneity
res <- multinom_test(tulip)
res
attr(res, "descriptives")

# Pairwise comparisons between groups
pairwise_binom_test(tulip, p.adjust.method = "bonferroni")

# Question 2: comparing observed to expected proportions
# this is a goodness-of-fit test
expected.p <- c(red = 0.5, yellow = 0.33, white = 0.17)
res <- multinom_test(tulip, expected.p)
res
attr(res, "descriptives")

# Pairwise comparisons against a given probabilities
pairwise_binom_test_against_p(tulip, expected.p)

box_m

Box’s M-test for Homogeneity of Covariance Matrices

Description

Performs the Box’s M-test for homogeneity of covariance matrices obtained from multivariate normal data according to one grouping variable. The test is based on the chi-square approximation.
**Usage**

```r
box_m(data, group)
```

**Arguments**

- `data`: a numeric data.frame or matrix containing `n` observations of `p` variables; it is expected that `n > p`.
- `group`: a vector of length `n` containing the class of each observation; it is usually a factor.

**Value**

A data frame containing the following components:

- `statistic`: an approximated value of the chi-square distribution.
- `parameter`: the degrees of freedom related of the test statistic in this case that it follows a Chi-square distribution.
- `p.value`: the p-value of the test.
- `method`: the character string "Box’s M-test for Homogeneity of Covariance Matrices".

**Examples**

```r
data(iris)
box_m(iris[, -5], iris[, 5])
```

---

**chisq_test**

*Chi-squared Test for Count Data*

**Description**

Performs chi-squared tests, including goodness-of-fit, homogeneity and independence tests.

**Usage**

```r
chisq_test(
  x,
  y = NULL,
  correct = TRUE,
  p = rep(1/length(x), length(x)),
  rescale.p = FALSE,
  simulate.p.value = FALSE,
  B = 2000
)
```

```r
pairwise_chisq_gof_test(x, p.adjust.method = "holm", ...)
```

```r
pairwise_chisq_test_against_p(
```
chisq_test

x,
p = rep(1/length(x), length(x)),
p.adjust.method = "holm",
... )
chisq_descriptives(res.chisq)
expected_freq(res.chisq)
observed_freq(res.chisq)
pearson_residuals(res.chisq)
std_residuals(res.chisq)

Arguments

x a numeric vector or matrix. x and y can also both be factors.
y a numeric vector; ignored if x is a matrix. If x is a factor, y should be a factor of the same length.
correct a logical indicating whether to apply continuity correction when computing the test statistic for 2 by 2 tables: one half is subtracted from all $|O - E|$ differences; however, the correction will not be bigger than the differences themselves. No correction is done if simulate.p.value = TRUE.
p a vector of probabilities of the same length of x. An error is given if any entry of p is negative.
rescale.p a logical scalar; if TRUE then p is rescaled (if necessary) to sum to 1. If rescale.p is FALSE, and p does not sum to 1, an error is given.
simulate.p.value a logical indicating whether to compute p-values by Monte Carlo simulation.
B an integer specifying the number of replicates used in the Monte Carlo test.
p.adjust.method method to adjust p values for multiple comparisons. Used when pairwise comparisons are performed. Allowed values include "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". If you don’t want to adjust the p value (not recommended), use p.adjust.method = "none".
... other arguments passed to the function {chisq_test}().
res.chisq an object of class chisq_test.

Value

return a data frame with some the following columns:

• n: the number of participants.
• group, group1, group2: the categories or groups being compared.
Chisq Test

- **statistic**: the value of Pearson’s chi-squared test statistic.
- **df**: the degrees of freedom of the approximate chi-squared distribution of the test statistic. NA if the p-value is computed by Monte Carlo simulation.
- **p**: p-value.
- **p.adj**: the adjusted p-value.
- **method**: the used statistical test.
- **p.signif, p.adj.signif**: the significance level of p-values and adjusted p-values, respectively.
- **observed**: observed counts.
- **expected**: the expected counts under the null hypothesis.

The returned object has an attribute called `args`, which is a list holding the test arguments.

**Functions**

- **chisq_test**: performs chi-square tests including goodness-of-fit, homogeneity and independence tests.
- **pairwise_chisq_gof_test**: perform pairwise comparisons between groups following a global chi-square goodness of fit test.
- **pairwise_chisq_test_against_p**: perform pairwise comparisons after a global chi-squared test for given probabilities. For each group, the observed and the expected proportions are shown. Each group is compared to the sum of all others.
- **chisq_descriptives**: returns the descriptive statistics of the chi-square test. These include, observed and expected frequencies, proportions, residuals and standardized residuals.
- **expected_freq**: returns the expected counts from the chi-square test result.
- **observed_freq**: returns the observed counts from the chi-square test result.
- **pearson_residuals**: returns the Pearson residuals, \((\text{observed} - \text{expected}) / \sqrt{\text{expected}}\).
- **std_residuals**: returns the standardized residuals

**Examples**

```r
# Chi-square goodness of fit test
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tulip <- c(red = 81, yellow = 50, white = 27)
# Q1: Are the colors equally common?
chisq_test(tulip)
pairwise_chisq_gof_test(tulip)
# Q2: comparing observed to expected proportions
chisq_test(tulip, p = c(1/2, 1/3, 1/6))
pairwise_chisq_test_against_p(tulip, p = c(0.5, 0.33, 0.17))

# Homogeneity of proportions between groups
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# Data: Titanic
xtab <- as.table(rbind(
  c(203, 118, 178, 212),
  xtab <- as.table(rbind(
  c(203, 118, 178, 212),
```

c(122, 167, 528, 673)
))
dimnames(xtab) <- list(
  Survived = c("Yes", "No"),
  Class = c("1st", "2nd", "3rd", "Crew")
)
xtab
# Chi-square test
chisq_test(xtab)
# Compare the proportion of survived between groups
pairwise_prop_test(xtab)

## cochran_qtest

### Cochran’s Q Test

**Description**

Performs the Cochran’s Q test for unreplicated randomized block design experiments with a binary response variable and paired data. This test is analogue to the friedman.test() with 0,1 coded response. It’s an extension of the McNemar Chi-squared test for comparing more than two paired proportions.

**Usage**

cochran_qtest(data, formula)

**Arguments**

- **data**: a data frame containing the variables in the formula.
- **formula**: a formula of the form `a ~ b | c`, where `a` is the outcome variable name; `b` is the within-subjects factor variables; and `c` (factor) is the column name containing individuals/subjects identifier. Should be unique per individual.

**Examples**

```r
# Generate a demo data
mydata <- data.frame(
  outcome = c(0,1,1,0,1,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1),
  treatment = gl(3,1,30,labels=LETTERS[1:3]),
  participant = gl(10,3,labels=letters[1:10])
)
mydata$outcome <- factor(
  mydata$outcome, levels = c(1, 0),
  labels = c("success", "failure")
)
# Cross-tabulation
xtabs(~outcome + treatment, mydata)
# Compare the proportion of success between treatments
```
cohens_d

cochran_qtest(mydata, outcome ~ treatment|participant)
# pairwise comparisons between groups
pairwise_mcnemar_test(mydata, outcome ~ treatment|participant)

cohens_d

Compute Cohen’s d Measure of Effect Size

Description
Compute the effect size for t-test. T-test conventional effect sizes, proposed by Cohen, are: 0.2 (small effect), 0.5 (moderate effect) and 0.8 (large effect).

Cohen’s d is calculated as the difference between means or mean minus mu divided by the estimated standardized deviation.

For independent samples t-test, there are two possibilities implemented. If the t-test did not make a homogeneity of variance assumption, (the Welch test), the variance term will mirror the Welch test, otherwise a pooled estimate is used.

If a paired samples t-test was requested, then effect size desired is based on the standard deviation of the differences.

It can also returns confidence intervals by bootstrap.

Usage
cohens_d(  
data,  
formula,  
comparisons = NULL,  
ref.group = NULL,  
paired = FALSE,  
mu = 0,  
var.equal = FALSE,  
hedges.correction = FALSE,  
ci = FALSE,  
conf.level = 0.95,  
ci.type = "perc",  
nboot = 1000  
)

Arguments

data a data.frame containing the variables in the formula.
formula a formula of the form x ~ group where x is a numeric variable giving the data values and group is a factor with one or multiple levels giving the corresponding groups. For example, formula = TP53 ~ cancer_group.
comparisons  A list of length-2 vectors specifying the groups of interest to be compared. For example to compare groups "A" vs "B" and "B" vs "C", the argument is as follow: comparisons = list(c("A","B"),c("B","C"))

ref.group  a character string specifying the reference group. If specified, for a given grouping variable, each of the group levels will be compared to the reference group (i.e. control group).
If ref.group = "all", pairwise two sample tests are performed for comparing each grouping variable levels against all (i.e. basemean).

paired  a logical indicating whether you want a paired test.

mu  theoretical mean, use for one-sample t-test. Default is 0.

var.equal  a logical variable indicating whether to treat the two variances as being equal. If TRUE then the pooled variance is used to estimate the variance otherwise the Welch (or Satterthwaite) approximation to the degrees of freedom is used. Used only for unpaired or independent samples test.

hedges.correction  logical indicating whether apply the Hedges correction by multiplying the usual value of Cohen’s d by (N-3)/(N-2.25) (for unpaired t-test) and by (n1-2)/(n1-1.25) for paired t-test; where N is the total size of the two groups being compared (N = n1 + n2).

if  If TRUE, returns confidence intervals by bootstrap. May be slow.

conf.level  The level for the confidence interval.

ci.type  The type of confidence interval to use. Can be any of "norm", "basic", "perc", or "bca". Passed to boot::boot.ci.

nboot  The number of replications to use for bootstrap.

Details
Quantification of the effect size magnitude is performed using the thresholds defined in Cohen (1992). The magnitude is assessed using the thresholds provided in (Cohen 1992), i.e. |d| < 0.2 "negligible", |d| < 0.5 "small", |d| < 0.8 "medium", otherwise "large".

Value
return a data frame with some of the following columns:

• y: the y variable used in the test.
• group1, group2: the compared groups in the pairwise tests.
• n,n1,n2: Sample counts.
• effsize: estimate of the effect size (d value).
• magnitude: magnitude of effect size.
• conf.low, conf.high: lower and upper bound of the effect size confidence interval.
References

- Navarro, Daniel. 2015. Learning Statistics with R: A Tutorial for Psychology Students and Other Beginners (Version 0.5).

Examples

```r
# One-sample t test effect size
ToothGrowth %>% cohens_d(len ~ 1, mu = 0)

# Two independent samples t-test effect size
ToothGrowth %>% cohens_d(len ~ supp, var.equal = TRUE)

# Paired samples effect size
df <- data.frame(
  id = 1:5,
  pre = c(110, 122, 101, 120, 140),
  post = c(150, 160, 110, 140, 155)
)
df <- df %>% gather(key = "treatment", value = "value", -id)
head(df)

df %>% cohens_d(value ~ treatment, paired = TRUE)
```

**convert_as_factor**

**Factors**

**Description**

Provides pipe-friendly functions to convert simultaneously multiple variables into a factor variable. Helper functions are also available to set the reference level and the levels order.

**Usage**

```r
convert_as_factor(data, ..., vars = NULL, make.valid.levels = FALSE)
set_ref_level(data, name, ref)
reorder_levels(data, name, order)
```
Arguments

data a data frame
... one unquoted expressions (or variable name) specifying the name of the variables you want to convert into factor. Alternative to the argument vars.
vars a character vector specifying the variables to convert into factor.
make.valid.levels logical. Default is FALSE. If TRUE, converts the variable to factor and add a leading character (x) if starting with a digit.
name a factor variable name. Can be unquoted. For example, use group or "group".
ref the reference level.
order a character vector specifying the order of the factor levels

Functions

- `convert_as_factor`: Convert one or multiple variables into factor.
- `set_ref_level`: Change a factor reference level or group.
- `reorder_levels`: Change the order of a factor levels

Examples

```r
# Create a demo data
df <- tibble(
  group = c("a", "a", "b", "b", "c", "c"),
  time = c("t1", "t2", "t1", "t2", "t1", "t2"),
  value = c(5, 6, 1, 3, 4, 5)
)
df

# Convert group and time into factor variable
result <- df %>% convert_as_factor(group, time)
result

# Show group levels
levels(result$group)

# Set c as the reference level (the first one)
result <- result %>%
  set_ref_level("group", ref = "c")
levels(result$group)

# Set the order of levels
result <- result %>%
  reorder_levels("group", order = c("b", "c", "a"))
levels(result$group)
```
cor_as_symbols

Replace Correlation Coefficients by Symbols

Description

Take a correlation matrix and replace the correlation coefficients by symbols according to the level of the correlation.

Usage

```r
cor_as_symbols(
  x,
  cutpoints = c(0, 0.25, 0.5, 0.75, 1),
  symbols = c(" ", ".", "*", "*")
)
```

Arguments

- `x`: a correlation matrix. Particularly, an object of class `cor_mat`.
- `cutpoints`: numeric vector used for intervals. Default values are `c(0, 0.25, 0.5, 0.75, 1)`.
- `symbols`: character vector, one shorter than `cutpoints`, used as correlation coefficient symbols. Default values are `c(" ", ".", "*", "*")`.

See Also

`cor_mat()`

Examples

```r
# Compute correlation matrix
#::::::::::::::::::::::::::::::::::::::::::::::::::
cor.mat <- mtcars %>%
  select(mpg, disp, hp, drat, wt, qsec) %>%
cor_mat()

# Replace correlation coefficient by symbols
#:::::::::::::::::::::::::::::::::::::::::::::::::
cor.mat %>%
cor_as_symbols() %>%
pull_lower_triangle()
```
**cor_gather**  
*Reshape Correlation Data*

**Description**

Reshape correlation analysis results. Key functions:

- **cor_gather()**: takes a correlation matrix and collapses (i.e. melt) it into a paired list (long format).
- **cor_spread()**: spread a long correlation data format across multiple columns. Particularly, it takes the results of **cor_test** and transforms it into a correlation matrix.

**Usage**

```r
cor_gather(data, drop.na = TRUE)
cor_spread(data, value = "cor")
```

**Arguments**

- **data**: a data frame or matrix.
- **drop.na**: logical. If TRUE, drop rows containing missing values after gathering the data.
- **value**: column name containing the value to spread.

**Functions**

- **cor_gather**: takes a correlation matrix and collapses (or melt) it into long format data frame (paired list)
- **cor_spread**: spread a long correlation data frame into wide format (correlation matrix).

**See Also**

`cor_mat()`, `cor_reorder()`

**Examples**

```r
# Data preparation
#:::::::::::::::::::::::::::::::::::::::::::::::::::::::::
mydata <- mtcars %>%
  select(mpg, disp, hp, drat, wt, qsec)
head(mydata, 3)

# Reshape a correlation matrix
#:::::::::::::::::::::::::::::::::::::::::::::::::::::::::
# Compute a correlation matrix
cor.mat <- mydata %>% cor_mat()
cor.mat
```
cor_mark_significant

# Collapse the correlation matrix into long format
# paired list data frame
long.format <- cor.mat %>% cor_gather()
long.format

# Spread a correlation data format
#::::::::::::::::::::::::::::::::::::::::::
# Spread the correlation coefficient value
long.format %>% cor_spread(value = "cor")
# Spread the p-value
long.format %>% cor_spread(value = "p")

cor_mark_significant  Add Significance Levels To a Correlation Matrix

Description

Combines correlation coefficients and significance levels in a correlation matrix data.

Usage

cor_mark_significant(
  x,
  cutpoints = c(0, 1e-04, 0.001, 0.01, 0.05, 1),
  symbols = c("****", "***", "**", "+", "")
)

Arguments

  x  an object of class cor_mat().
  cutpoints numeric vector used for intervals.
  symbols character vector, one shorter than cutpoints, used as significance symbols.

Value

  a data frame containing the lower triangular part of the correlation matrix marked by significance symbols.

Examples

  mtcars %>%
    select(mpg, disp, hp, drat, wt, qsec) %>%
    cor_mat() %>%
    cor_mark_significant()
Compute Correlation Matrix with P-values

Description
Compute correlation matrix with p-values. Numeric columns in the data are detected and automatically selected for the analysis. You can also specify variables of interest to be used in the correlation analysis.

Usage

```r
cor_mat(
data,
      ...,
      vars = NULL,
      method = "pearson",
      alternative = "two.sided",
      conf.level = 0.95
    )

cor_pmat(
data,
      ...,
      vars = NULL,
      method = "pearson",
      alternative = "two.sided",
      conf.level = 0.95
    )

cor_get_pval(x)
```

Arguments

- `data`: a data.frame containing the variables.
- `...`: One or more unquoted expressions (or variable names) separated by commas. Used to select a variable of interest.
- `vars`: a character vector containing the variable names of interest.
- `method`: a character string indicating which correlation coefficient is to be used for the test. One of "pearson", "kendall", or "spearman", can be abbreviated.
- `alternative`: indicates the alternative hypothesis and must be one of "two.sided", "greater" or "less". You can specify just the initial letter. "greater" corresponds to positive association, "less" to negative association.
- `conf.level`: confidence level for the returned confidence interval. Currently only used for the Pearson product moment correlation coefficient if there are at least 4 complete pairs of observations.
- `x`: an object of class cor_mat
cor_mat

Value

a data frame

Functions

- cor_mat: compute correlation matrix with p-values. Returns a data frame containing the
correlation coefficients. The output has an attribute named "pvalue", which
contains the matrix of the correlation test p-values.
- cor_pmat: compute the correlation matrix but returns only the p-values of the tests.
- cor_get_pval: extract a correlation matrix p-values from an object of class cor_mat().

See Also

cor_test(), cor_reorder(), cor_gather(), cor_select(), cor_as_symbols(), pull_triangle(),
replace_triangle()

Examples

# Data preparation
#:::::::::::::::::::::::::::::::::::::::::::
mydata <- mtcars %>%
  select(mpg, disp, hp, drat, wt, qsec)
head(mydata, 3)

# Compute correlation matrix
#::::::::::::::::::::::::::::::::::::::::::
# Correlation matrix between all variables
cor.mat <- mydata %>% cor_mat()
cor.mat

# Specify some variables of interest
mydata %>% cor_mat(mpg, hp, wt)

# Or remove some variables in the data
# before the analysis
mydata %>% cor_mat(-mpg, -hp)

# Significance levels
#::::::::::::::::::::::::::::::::::::::::::
cor.mat %>% cor_get_pval()

# Visualize
#::::::::::::::::::::::::::::::::::::::::::
# Insignificant correlations are marked by crosses
cor.mat %>%
cor_reorder() %>%
pull_lower_triangle() %>%
cor_plot(label = TRUE)

# Gather/collapse correlation matrix into long format
cor_plot

Visualize Correlation Matrix Using Base Plot

Description

Provide a tibble-friendly framework to visualize a correlation matrix. Wrapper around the R base function `corrplot()`. Compared to `corrplot()`, it can handle directly the output of the functions `cor_mat()` (in rstatix), `rcorr()` (in Hmisc), `correlate()` (in corrr) and `cor()` (in stats).

The p-values contained in the outputs of the functions `cor_mat()` and `rcorr()` are automatically detected and used in the visualization.

Usage

```r
cor_plot(
  cor.mat,  # the correlation matrix to visualize
  method = "circle",  # Character, the visualization method of correlation matrix to be used. Currently, it supports seven methods, named "circle" (default), "square", "ellipse", "number", "pie", "shade" and "color". See examples for details.
  type = "full",  # Character, "full" (default), "upper" or "lower", display full matrix, lower triangular or upper triangular matrix.
  palette = NULL,  # character vector containing the color palette.
  p.mat = NULL,  # matrix of p-value corresponding to the correlation matrix.
  significant.level = 0.05,  # numeric, p-value threshold. Significant correlations are marked with significant level.
  insignificant = c("cross", "blank"),  # character vector of non-significant correlation marks.
  label = FALSE,  # logical, display value labels.
  font.label = list()  # list of arguments to pass to font
)
```

Arguments

- `cor.mat`: the correlation matrix to visualize
- `method`: Character, the visualization method of correlation matrix to be used. Currently, it supports seven methods, named "circle" (default), "square", "ellipse", "number", "pie", "shade" and "color". See examples for details.
  The areas of circles or squares show the absolute value of corresponding correlation coefficients. Method "pie" and "shade" came from Michael Friendly's job (with some adjustment about the shade added on), and "ellipse" came from D.J. Murdoch and E.D. Chow's job, see in section References.
- `type`: Character, "full" (default), "upper" or "lower", display full matrix, lower triangular or upper triangular matrix.
- `palette`: character vector containing the color palette.
- `p.mat`: matrix of p-value corresponding to the correlation matrix.
significant.level

significant level, if the p-value is bigger than significant.level, then the corresponding correlation coefficient is regarded as insignificant.

insignificant

character, specialized insignificant correlation coefficients, "cross" (default), "blank". If "blank", wipe away the corresponding glyphs; if "cross", add crosses (X) on corresponding glyphs.

label

logical value. If TRUE, shows the correlation coefficient labels.

font.label

a list with one or more of the following elements: size (e.g., 1), color (e.g., "black") and style (e.g., "bold"). Used to customize the correlation coefficient labels. For example font.label = list(size = 1, color = "black", style = "bold").

See Also

  cor_as_symbols()

Examples

  # Compute correlation matrix
  #::::::::::::::::::::::::::::::::::::::::::
  cor.mat <- mtcars %>%
    select(mpg, disp, hp, drat, wt, qsec) %>%
    cor_mat()

  # Visualize correlation matrix
  #::::::::::::::::::::::::::::::::::::::::::
  # Full correlation matrix,
  # insignificant correlations are marked by crosses
  cor.mat %>% cor_plot()

  # Reorder by correlation coefficient
  # pull lower triangle and visualize
  cor.lower.tri <- cor.mat %>%
    cor_reorder() %>%
    pull_lower_triangle()
  cor.lower.tri %>% cor_plot()

  # Change visualization methods
  #::::::::::::::::::::::::::::::::::::::::::
  cor.lower.tri %>%
    cor_plot(method = "pie")
  cor.lower.tri %>%
    cor_plot(method = "color")
  cor.lower.tri %>%
    cor_plot(method = "number")

  # Show the correlation coefficient: label = TRUE
  # Blank the insignificant correlation
  #::::::::::::::::::::::::::::::::::::::::::
cor.lower.tri %>%
cor_plot(
    method = "color",
    label = TRUE,
    insignificant = "blank"
)

# Change the color palettes
#::::::::::::::::::::::::::::::::::::::::::

# Using custom color palette
# Require ggpubr: install.packages("ggpubr")
if(require("ggpubr"))(
    my.palette <- get_palette(c("red", "white", "blue"), 200)
    cor.lower.tri %>%
    cor_plot(palette = my.palette)
}

# Using RColorBrewer color palette
if(require("ggpubr"))(
    my.palette <- get_palette("PuOr", 200)
    cor.lower.tri %>%
    cor_plot(palette = my.palette)
}

---

**cor_reorder**

### Reorder Correlation Matrix

#### Description
reorder correlation matrix, according to the coefficients, using the hierarchical clustering method.

#### Usage

```r
cor_reorder(x)
```

#### Arguments

- **x**: a correlation matrix. Particularly, an object of class `cor_mat`.

#### Value

a data frame

#### See Also

`cor_mat()`, `cor_gather()`, `cor_spread()`
Examples

```r
# Compute correlation matrix
#::::::::::::::::::::::::::::::::::::::::::::::::
cor.mat <- mtcars %>%
  select(mpg, disp, hp, drat, wt, qsec) %>%
  cor_mat()

# Reorder by correlation and get p-values
#::::::::::::::::::::::::::::::::::::::::::::::
# Reorder
cor.mat %>%
  cor_reorder()
# Get p-values of the reordered cor_mat
cor.mat %>%
cor_reorder() %>%
cor_get_pval()
```

---

**cor_select**  
*Subset Correlation Matrix*

**Description**

Subset Correlation Matrix

**Usage**

```r
cor_select(x, ..., vars = NULL)
```

**Arguments**

- `x`  
a correlation matrix. Particularly, an object of class `cor_mat`.
- `...`  
One or more unquoted expressions (or variable names) separated by commas. Used to select variables of interest.
- `vars`  
a character vector containing the variable names of interest.

**Value**

a data frame

**See Also**

`cor_mat()`, `pull_triangle()`, `replace_triangle()`
Examples

```r
# Compute correlation matrix
#::::::::::::::::::::::::::::::::::::::::::
cor.mat <- mtcars %>%
  select(mpg, disp, hp, drat, wt, qsec) %>%
cor_mat()

# Subsetting correlation matrix
#::::::::::::::::::::::::::::::::::::::::::
# Select some variables of interest
cor.mat %>%
cor_select(mpg, drat, wt)

# Remove variables
cor.mat %>%
cor_select(-mpg, -wt)
```

---

**cor_test**  
*Correlation Test*

**Description**

Provides a pipe-friendly framework to perform correlation test between paired samples, using Pearson, Kendall or Spearman method. Wrapper around the function `cor.test()`.

Can also perform multiple pairwise correlation analyses between more than two variables or between two different vectors of variables. Using this function, you can also compute, for example, the correlation between one variable vs many.

**Usage**

```r
cor_test(
  data,
  ...,
  vars = NULL,
  vars2 = NULL,
  alternative = "two.sided",
  method = "pearson",
  conf.level = 0.95,
  use = "pairwise.complete.obs"
)
```

**Arguments**

- `data`: a data.frame containing the variables.
- `...`: One or more unquoted expressions (or variable names) separated by commas. Used to select a variable of interest. Alternative to the argument `vars`.
cor_test

vars optional character vector containing variable names for correlation analysis. Ignored when dot vars are specified.

- If vars is NULL, multiple pairwise correlation tests is performed between all variables in the data.
- If vars contain only one variable, a pairwise correlation analysis is performed between the specified variable vs either all the remaining numeric variables in the data or variables in vars2 (if specified).
- If vars contain two or more variables: i) if vars2 is not specified, a pairwise correlation analysis is performed between all possible combinations of variables. ii) if vars2 is specified, each element in vars is tested against all elements in vars2.

vars2 optional character vector. If specified, each element in vars is tested against all elements in vars2. Accept unquoted variable names: c(var1, var2).

alternative indicates the alternative hypothesis and must be one of "two.sided", "greater" or "less". You can specify just the initial letter. "greater" corresponds to positive association, "less" to negative association.

method a character string indicating which correlation coefficient is to be used for the test. One of "pearson", "kendall", or "spearman", can be abbreviated.

conf.level confidence level for the returned confidence interval. Currently only used for the Pearson product moment correlation coefficient if there are at least 4 complete pairs of observations.

use an optional character string giving a method for computing covariances in the presence of missing values. This must be (an abbreviation of) one of the strings "everything", "all.obs", "complete.obs", "na.or.complete", or "pairwise.complete.obs".

**Value**

return a data frame with the following columns:

- var1, var2: the variables used in the correlation test.
- cor: the correlation coefficient.
- statistic: Test statistic used to compute the p-value.
- p: p-value.
- conf.low, conf.high: Lower and upper bounds on a confidence interval.
- method: the method used to compute the statistic.

**Functions**

- cor_test: correlation test between two or more variables.

**See Also**

cor_mat(), as_cor_mat()
Examples

```r
# Correlation between the specified variable vs
# the remaining numeric variables in the data
#:::::::::::::::::::::::::::::::::::::::::
mtcars %>% cor_test(mpg)

# Correlation test between two variables
#:::::::::::::::::::::::::::::::::::::::::
mtcars %>% cor_test(wt, mpg)

# Pairwise correlation between multiple variables
#:::::::::::::::::::::::::::::::::::::::::
mtcars %>% cor_test(wt, mpg, disp)

# Grouped data
#:::::::::::::::::::::::::::::::::::::::::
iris %>%
  group_by(Species) %>%
  cor_test(Sepal.Width, Sepal.Length)

# Multiple correlation test
#:::::::::::::::::::::::::::::::::::::::::
# Correlation between one variable vs many
mtcars %>% cor_test(
  vars = "mpg",
  vars2 = c("disp", "hp", "drat")
)

# Correlation between two vectors of variables
# Each element in vars is tested against all elements in vars2
mtcars %>% cor_test(
  vars = c("mpg", "wt"),
  vars2 = c("disp", "hp", "drat")
)
```

---

### counts_to_cases

`counts_to_cases` *Convert a Table of Counts into a Data Frame of cases*

**Description**

converts a contingency table or a data frame of counts into a data frame of individual observations.

**Usage**

```r
counts_to_cases(x, count.col = "Freq")
```
Arguments

- **x**  a contingency table or a data frame
- **count.col**  the name of the column containing the counts. Default is "Freq".

Value

a data frame of cases

Examples

```r
# Create a cross-tabulation demo data
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xtab <- as.table(
  rbind(c(20, 5), c(16,9))
)
dimnames(xtab) <- list(
  before = c("non.smoker", "smoker"),
  after = c("non.smoker", "smoker")
)
xtab
# Convert into a data frame of cases
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
df <- counts_to_cases(xtab)
head(df)
```

---

**cr**am**e**r_v  

*Compute Cramer’s V*

**Description**

Compute Cramer’s V, which measures the strength of the association between categorical variables.

**Usage**

```r
cramer_v(x, y = NULL, correct = TRUE, ...)
```

**Arguments**

- **x**  a numeric vector or matrix. x and y can also both be factors.
- **y**  a numeric vector; ignored if x is a matrix. If x is a factor, y should be a factor of the same length.
- **correct**  a logical indicating whether to apply continuity correction when computing the test statistic for 2 by 2 tables: one half is subtracted from all \(|O - E|\) differences; however, the correction will not be bigger than the differences themselves. No correction is done if simulate.p.value = TRUE.
- **...**  other arguments passed to the function `chisq.test()`.
# Data preparation

```r
df <- as.table(rbind(c(762, 327, 468), c(484, 239, 477)))
dimnames(df) <- list(
  gender = c("F", "M"),
  party = c("Democrat", "Independent", "Republican")
)
df
# Compute cramer's V
cramer_v(df)
```

---

**df_arrange**

**Arrange Rows by Column Values**

### Description

Order the rows of a data frame by values of specified columns. Wrapper around the `arrange()` function. Supports standard and non standard evaluation.

### Usage

```r
df_arrange(data, ..., vars = NULL, .by_group = FALSE)
```

### Arguments

- **data**
  - a data frame
- **...**
  - One or more unquoted expressions (or variable names) separated by commas. Used to select a variable of interest. Use `desc()` to sort a variable in descending order.
- **vars**
  - a character vector containing the variable names of interest.
- **.by_group**
  - If TRUE, will sort first by grouping variable. Applies to grouped data frames only.

### Value

a data frame

### Examples

```r
df <- head(ToothGrowth)
df

# Select column using standard evaluation
df %>% df_arrange(vars = c("dose", "len"))

# Select column using non-standard evaluation
df %>% df_arrange(dose, desc(len))
```
df_get_var_names  

Get User Specified Variable Names

Description
Returns user specified variable names. Supports standard and non standard evaluation.

Usage

```r
df_get_var_names(data, ..., vars = NULL)
```

Arguments

- `data`  
a data frame
- `...`  
One or more unquoted expressions (or variable names) separated by commas. Used to select a variable of interest.
- `vars`  
a character vector containing the variable names of interest.

Value

a character vector

Examples

```r
# Non standard evaluation
ToothGrowth %>%
df_get_var_names(dose, len)

# Standard evaluation
ToothGrowth %>%
df_get_var_names(vars = c("len", "dose"))
```

df_group_by  

Group a Data Frame by One or more Variables

Description

Group a data frame by one or more variables. Supports standard and non standard evaluation.

Usage

```r
df_group_by(data, ..., vars = NULL)
```
df_label_both

Arguments

data a data frame

... One or more unquoted expressions (or variable names) separated by commas. Used to select a variable of interest.

vars a character vector containing the variable names of interest.

Examples

# Non standard evaluation
by_dose <- head(ToothGrowth) %>%
  df_group_by(dose)
by_dose

# Standard evaluation
head(ToothGrowth) %>%
  df_group_by(vars = c("dose", "supp"))

Description

Functions to label data frame rows by one or multiple grouping variables.

Usage

df_label_both(data, ..., vars = NULL, label_col = "label", sep = c("", ", ":"))

df_label_value(data, ..., vars = NULL, label_col = "label", sep = ", ", )

Arguments

data a data frame

... One or more unquoted expressions (or variable names) separated by commas. Used as grouping variables.

vars a character vector containing the grouping variables of interest.

label_col column to hold the label of the data subsets. Default column name is "label".

sep String separating labelling variables and values. Should be of length 2 in the function df_label_both(). 1) One sep is used to separate groups, for example ";"; 2) The other sep between group name and levels; for example ",".

Value

a modified data frame with a column containing row labels.
Functions

- `df_label_both`: Displays both the variable name and the factor value.
- `df_label_value`: Displays only the value of a factor.

Examples

```r
# Data preparation
df <- head(ToothGrowth)

# Labelling: Non standard evaluation
df %>%
  df_label_both(dose, supp)

# Standard evaluation
df %>%
  df_label_both(dose, supp)

# Nesting the data then label each subset by groups
ToothGrowth %>%
  df_nest_by(dose, supp) %>%
  df_label_both(supp, dose)
```

---

### df_nest_by

**Nest a Tibble By Groups**

**Description**

Nest a tibble data frame using grouping specification. Supports standard and non standard evaluation.

**Usage**

```
df_nest_by(data, ..., vars = NULL)
```

**Arguments**

- `data` a data frame
- `...` One or more unquoted expressions (or variable names) separated by commas. Used as grouping variables.
- `vars` a character vector containing the grouping variables of interest.

**Value**

A tbl with one row per unique combination of the grouping variables. The first columns are the grouping variables, followed by a list column of tibbles with matching rows of the remaining columns.
Examples

```r
# Non standard evaluation
ToothGrowth %>%
df_nest_by(dose, supp)

# Standard evaluation
ToothGrowth %>%
df_nest_by(vars = c("dose", "supp"))
```

---

**df_select**

*Select Columns in a Data Frame*

**Description**

A wrapper around the `select()` function for selection data frame columns. Supports standard and non standard evaluations. Useful to easily program with `dplyr`.

**Usage**

```r
df_select(data, ..., vars = NULL)
```

**Arguments**

- `data` a data frame
- `...` One or more unquoted expressions (or variable names) separated by commas. Used to select a variable of interest.
- `vars` a character vector containing the variable names of interest.

**Value**

a data frame

**Examples**

```r
df <- head(ToothGrowth)
df

# Select column using standard evaluation
df %>% df_select(vars = c("dose", "len"))

# Select column using non-standard evaluation
df %>% df_select(dose, len)
```
**df_split_by**  
*Split a Data Frame into Subset*

**Description**

Split a data frame by groups into subsets or data panel. Very similar to the function `df_nest_by()`. The only difference is that, it adds label to each data subset. Labels are the combination of the grouping variable levels. The column holding labels are named "label".

**Usage**

```r
df_split_by(
  data,
  ..., 
  vars = NULL, 
  label_col = "label", 
  labeller = df_label_both, 
  sep = c("", ", ", ":") 
)
```

**Arguments**

- `data`  
a data frame
- `...`  
One or more unquoted expressions (or variable names) separated by commas. Used as grouping variables.
- `vars`  
a character vector containing the grouping variables of interest.
- `label_col`  
column to hold the label of the data subsets. Default column name is "label".
- `labeller`  
a function that takes a data frame, the grouping variables, label_col and label_sep arguments, and add labels into the data frame. Example of possible values are: `df_label_both()` and `df_label_value()`.
- `sep`  
String separating labelling variables and values. Should be of length 2 in the function `df_label_both()`. 1) One sep is used to separate groups, for example ":"; 2) The other sep between group name and levels; for example ":".

**Value**

A tbl with one row per unique combination of the grouping variables. The first columns are the grouping variables, followed by a list column of tibbles with matching rows of the remaining columns, and a column named label, containing labels.

**Examples**

```r
# Split a data frame  
# :::::::::::::::::::::::::::::::::::::::::::::::::
# Create a grouped data
```
res <- ToothGrowth %>%
  df_split_by(dose, supp)
res

# Show subsets
res$data

# Add panel/subset labels
res <- ToothGrowth %>%
  df_split_by(dose, supp)
res

---

**df_unite**  
_Unite Multiple Columns into One_

**Description**

Paste together multiple columns into one. Wrapper around `unite()` that supports standard and non-standard evaluation.

**Usage**

```r
df_unite(data, col, ..., vars = NULL, sep = "_", remove = TRUE, na.rm = FALSE)
df_unite_factors(
  data,
  col,
  ..., 
  vars = NULL,
  sep = "_",
  remove = TRUE,
  na.rm = FALSE
)
```

**Arguments**

- `data` a data frame
- `col` the name of the new column as a string or a symbol.
- `...` a selection of columns. One or more unquoted expressions (or variable names) separated by commas.
- `vars` a character vector containing the column names of interest.
- `sep` Separator to use between values.
- `remove` If TRUE, remove input columns from output data frame.
- `na.rm` If TRUE, missing values will be removed prior to uniting each value.
Functions

- `df_unite`: Unite multiple columns into one.
- `df_unite_factors`: Unite factor columns. First, order factors levels then merge them into one column. The output column is a factor.

Examples

```r
# Non standard evaluation
head(ToothGrowth) %>%
  df_unite(col = "dose_supp", dose, supp)

# Standard evaluation
head(ToothGrowth) %>%
  df_unite(col = "dose_supp", vars = c("dose", "supp"))
```

---

**doo**  
*Alternative to dplyr::do for Doing Anything*

**Description**

Provides a flexible alternative to the dplyr::do() function. Technically it uses nest() + mutate() + map() to apply arbitrary computation to a grouped data frame.

The output is a data frame. If the applied function returns a data frame, then the output will be automatically unnested. Otherwise, the output includes the grouping variables and a column named ".results." (by default), which is a "list-columns" containing the results for group combinations.

**Usage**

```r
doo(data, .f, ..., result = ".results.")
```

**Arguments**

- `data` a (grouped) data frame
- `f` A function, formula, or atomic vector. For example `~t.test(len ~ supp, data = .)`.
- `...` Additional arguments passed on to .f
- `result` the column name to hold the results. Default is ".results.".

**Value**
a data frame
Examples

# Custom function
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
stat_test <- function(data, formula){
  t.test(formula, data) %>%
  tidy()
}
# Example 1: pipe-friendly stat_test().
# Two possibilities of usage are available
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# Use this
ToothGrowth %>%
  group_by(dose) %>%
  doo(~stat_test(data = ., len ~ supp))
# Or this
ToothGrowth %>%
  group_by(dose) %>%
  doo(stat_test, len ~ supp)
# Example 2: R base function t.test() (not pipe friendly)
# One possibility of usage
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
comparisons <- ToothGrowth %>%
  group_by(dose) %>%
  doo(~t.test(len ~ supp, data = .))
comparisons
comparisons$.results.
# Example 3: R base function combined with tidy()
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ToothGrowth %>%
  group_by(dose) %>%
  doo(~t.test(len ~ supp, data = .) %>% tidy())

---

dunn_test

Dunn’s Test of Multiple Comparisons

Description

Performs Dunn’s test for pairwise multiple comparisons of the ranked data. The mean rank of the different groups is compared. Used for post-hoc test following Kruskal-Wallis test.

Usage

dunn_test(data, formula, p.adjust.method = "holm", detailed = FALSE)
Arguments

- **data**: a data.frame containing the variables in the formula.
- **formula**: a formula of the form \( x \sim group \) where \( x \) is a numeric variable giving the data values and \( group \) is a factor with one or multiple levels giving the corresponding groups. For example, \( formula = TP53 \sim cancer_group \).
- **p.adjust.method**: method to adjust \( p \) values for multiple comparisons. Used when pairwise comparisons are performed. Allowed values include "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". If you don’t want to adjust the \( p \) value (not recommended), use \( p.adjust.method = "none" \).
- **detailed**: logical value. Default is FALSE. If TRUE, a detailed result is shown.

Details

DunnTest performs the post hoc pairwise multiple comparisons procedure appropriate to follow up a Kruskal-Wallis test, which is a non-parametric analog of the one-way ANOVA. The Wilcoxon rank sum test, itself a non-parametric analog of the unpaired t-test, is possibly intuitive, but inappropriate as a post hoc pairwise test, because (1) it fails to retain the dependent ranking that produced the Kruskal-Wallis test statistic, and (2) it does not incorporate the pooled variance estimate implied by the null hypothesis of the Kruskal-Wallis test.

Value

return a data frame with some of the following columns:

- \( .y. \): the \( y \) (outcome) variable used in the test.
- **group1, group2**: the compared groups in the pairwise tests.
- **n1, n2**: Sample counts.
- **estimate**: mean ranks difference.
- **statistic**: Test statistic (z-value) used to compute the \( p \)-value.
- **p**: \( p \)-value.
- **p.adj**: the adjusted \( p \)-value.
- **method**: the statistical test used to compare groups.
- **p.signif, p.adj.signif**: the significance level of \( p \)-values and adjusted \( p \)-values, respectively.

The returned object has an attribute called **args**, which is a list holding the test arguments.

References

**Examples**

```r
toothgrow %>% dunn_test(len ~ dose)

# Grouped data
ToothGrowth %>%
  group_by(supp) %>%
  dunn_test(len ~ dose)
```

**Description**

Performs pairwise comparisons between groups using the estimated marginal means. Pipe-friendly wrapper around the functions `emmeans()` + `contrast()` from the `emmeans` package, which need to be installed before using this function. This function is useful for performing post-hoc analyses following ANOVA/ANCOVA tests.

**Usage**

```r
emmeans_test(
  data,
  formula,
  covariate = NULL,
  ref.group = NULL,
  comparisons = NULL,
  p.adjust.method = "bonferroni",
  conf.level = 0.95,
  model = NULL,
  detailed = FALSE
)
```

**Arguments**

- **data**: a data.frame containing the variables in the formula.
- **formula**: a formula of the form `x ~ group` where `x` is a numeric variable giving the data values and `group` is a factor with one or multiple levels giving the corresponding groups. For example, `formula = TP53 ~ cancer_group`.
- **covariate**: (optional) covariate names (for ANCOVA).
- **ref.group**: a character string specifying the reference group. If specified, for a given grouping variable, each of the group levels will be compared to the reference group (i.e. control group).
  - If `ref.group = "all"`, pairwise two sample tests are performed for comparing each grouping variable levels against all (i.e. basemean).
comparisons  A list of length-2 vectors specifying the groups of interest to be compared. For example to compare groups "A" vs "B" and "B" vs "C", the argument is as follow: comparisons = list(c("A","B"),c("B","C"))

p.adjust.method  method to adjust p values for multiple comparisons. Used when pairwise comparisons are performed. Allowed values include "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". If you don’t want to adjust the p value (not recommended), use p.adjust.method = "none".

conf.level  confidence level of the interval.

model  a fitted-model objects such as the result of a call to `lm()`, from which the overall degrees of freedom are to be calculated.

detailed  logical value. Default is FALSE. If TRUE, a detailed result is shown.

emmeans.test  an object of class `emmeans_test`.

Value

return a data frame with some the following columns:

• .y.: the y variable used in the test.
• group1,group2: the compared groups in the pairwise tests.
• statistic: Test statistic (t.ratio) used to compute the p-value.
• df: degrees of freedom.
• p: p-value.
• p.adj: the adjusted p-value.
• method: the statistical test used to compare groups.
• p.signif,p.adj.signif: the significance level of p-values and adjusted p-values, respectively.
• estimate: estimate of the effect size, that is the difference between the two emmeans (estimated marginal means).
• conf.low,conf.high: Lower and upper bound on a confidence interval of the estimate.

The returned object has an attribute called args, which is a list holding the test arguments. It has also an attribute named "emmeans", a data frame containing the groups emmeans.

Functions

• `get_emmeans`: returns the estimated marginal means from an object of class `emmeans_test`

Examples

# Data preparation
df <- ToothGrowth
df$dose <- as.factor(df$dose)

# Pairwise comparisons
res <- df %>%


eta_squared

```r
group_by(supp) %>%
  emmeans_test(len ~ dose, p.adjust.method = "bonferroni")
res

# Display estimated marginal means
attr(res, "emmeans")

# Show details
df %>%
  group_by(supp) %>%
  emmeans_test(len ~ dose, p.adjust.method = "bonferroni", detailed = TRUE)
```

---

**Description**

Compute eta-squared and partial eta-squared for all terms in an ANOVA model.

**Usage**

```r
eta_squared(model)
```

```r
partial_eta_squared(model)
```

**Arguments**

- `model`:
  - an object of class `aov` or `anova`.

**Value**

A numeric vector with the effect size statistics

**Functions**

- `eta_squared`: compute eta squared
- `partial_eta_squared`: compute partial eta squared.

**Examples**

```r
# Data preparation
df <- ToothGrowth
df$dose <- as.factor(df$dose)

# Compute ANOVA
res.aov <- aov(len ~ supp*dose, data = df)
summary(res.aov)

# Effect size
eta_squared(res.aov)
partial_eta_squared(res.aov)
```
factorial_design  
Build Factorial Designs for ANOVA

Description

Provides helper functions to build factorial design for easily computing ANOVA using the Anova() function. This might be very useful for repeated measures ANOVA, which is hard to set up with the car package.

Usage

factorial_design(data, dv, wid, between, within, covariate)

Arguments

data  
a data frame containing the variables
dv  
(numeric) dependent variable name.
wid  
(factor) column name containing individuals/subjects identifier. Should be unique per individual.
between  
(optional) between-subject factor variables.
within  
(optional) within-subjects factor variables
covariate  
(optional) covariate names (for ANCOVA)

Value

a list with the following components:

- the specified arguments: dv, wid, between, within
- data: the original data (long format) or independent ANOVA. The wide format is returned for repeated measures ANOVA.
- idata: an optional data frame giving the levels of factors defining the intra-subject model for multivariate repeated-measures data.
- idesign: a one-sided model formula using the “data” in idata and specifying the intra-subject design.
- repeated: logical. Value is TRUE when the data is a repeated design.
- lm_formula: the formula used to build the lm model.
- lm_data: the data used to build the lm model. Can be either in a long format (i.e., the original data for independent measures ANOVA) or in a wide format (case of repeated measures ANOVA).
- model: the lm model

Author(s)

Alboukadel Kassambara, <alboukadel.kassambara@gmail.com>
See Also

\texttt{anova.test()}, \texttt{anova_summary()}

Examples

\begin{verbatim}
# Load data
#:::::::::::::::::::::::::::::::::::::::
data("ToothGrowth")
df <- ToothGrowth
head(df)

# Repeated measures designs
#:::::::::::::::::::::::::::::::::::::::::
# Prepare the data
df$id <- rep(1:10, 6) # Add individuals id
head(df)
# Build factorial designs
design <- factorial_design(df, dv = len, wid = id, within = c(supp, dose))
design
# Easily perform repeated measures ANOVA using the car package
res.anova <- Anova(design$model, idata = design$idata, idesign = design$idesign, type = 3)
summary(res.anova, multivariate = FALSE)

# Independent measures designs
#:::::::::::::::::::::::::::::::::::::::::
# Build factorial designs
df$id <- 1:nrow(df)
design <- factorial_design(df, dv = len, wid = id, between = c(supp, dose))
design
# Perform ANOVA
Anova(design$model, type = 3)
\end{verbatim}

---

\section*{fisher_test \hspace{1cm} Fisher's Exact Test for Count Data}

\section*{Description}

Performs Fisher’s exact test for testing the null of independence of rows and columns in a contingency table.

Wrappers around the R base function \texttt{fisher.test()} but have the advantage of performing pairwise and row-wise fisher tests, the post-hoc tests following a significant chi-square test of homogeneity for 2xc and rx2 contingency tables.

\section*{Usage}

\begin{verbatim}
fisher_test(
 xtab,
\end{verbatim}
workspace = 2e+05,
alternative = "two.sided",
conf.int = TRUE,
conf.level = 0.95,
simulate.p.value = FALSE,
B = 2000,
detailed = FALSE,
...
)
pairwise_fisher_test(xtab, p.adjust.method = "holm", detailed = FALSE, ...)  
row_wise_fisher_test(xtab, p.adjust.method = "holm", detailed = FALSE, ...)

Arguments

xtab a contingency table in a matrix form.
workspace an integer specifying the size of the workspace used in the network algorithm. In units of 4 bytes. Only used for non-simulated p-values larger than 2 × 2 tables. Since R version 3.5.0, this also increases the internal stack size which allows larger problems to be solved, however sometimes needing hours. In such cases, simulate.p.values=TRUE may be more reasonable.
alternative indicates the alternative hypothesis and must be one of "two.sided", "greater" or "less". You can specify just the initial letter. Only used in the 2 × 2 case.
conf.int logical indicating if a confidence interval for the odds ratio in a 2×2 table should be computed (and returned).
conf.level confidence level for the returned confidence interval. Only used in the 2 × 2 case and if conf.int = TRUE.
simulate.p.value a logical indicating whether to compute p-values by Monte Carlo simulation, in larger than 2 × 2 tables.
B an integer specifying the number of replicates used in the Monte Carlo test.
detailed logical value. Default is FALSE. If TRUE, a detailed result is shown.
... Other arguments passed to the function fisher_test().
p.adjust.method method to adjust p values for multiple comparisons. Used when pairwise comparisons are performed. Allowed values include "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". If you don’t want to adjust the p value (not recommended), use p.adjust.method = "none".

Value

return a data frame with some the following columns:

• group: the categories in the row-wise proportion tests.
• p: p-value.
• p.adj: the adjusted p-value.
• method: the used statistical test.
• p.signif,p.adj.signif: the significance level of p-values and adjusted p-values, respectively.
• estimate: an estimate of the odds ratio. Only present in the 2 by 2 case.
• alternative: a character string describing the alternative hypothesis.
• conf.low,conf.high: a confidence interval for the odds ratio. Only present in the 2 by 2 case and if argument conf.int = TRUE.

The returned object has an attribute called args, which is a list holding the test arguments.

Functions

• fisher_test: performs Fisher’s exact test for testing the null of independence of rows and columns in a contingency table with fixed marginals. Wrapper around the function fisher.test().
• pairwise_fisher_test: pairwise comparisons between proportions, a post-hoc tests following a significant Fisher’s exact test of homogeneity for 2xc design.
• row_wise_fisher_test: performs row-wise Fisher’s exact test of count data, a post-hoc tests following a significant chi-square test of homogeneity for rx2 contingency table. The test is conducted for each category (row).

Examples

# Comparing two proportions
#-------------------------------------------------------------------------------------
# Data: frequencies of smokers between two groups
xtab <- as.table(rbind(c(490, 10), c(400, 100)))
dimnames(xtab) <- list(
    group = c("grp1", "grp2"),
    smoker = c("yes", "no")
)
xtab
# compare the proportion of smokers
fisher_test(xtab, detailed = TRUE)

# Homogeneity of proportions between groups
#-------------------------------------------------------------------------------------
# H0: the proportion of smokers is similar in the four groups
# Ha: this proportion is different in at least one of the populations.
# # Data preparation
grp.size <- c(106, 113, 156, 102)
smokers <- c(50, 100, 139, 80)
no.smokers <- grp.size - smokers
xtab <- as.table(rbind(
    smokers,
    no.smokers
))
freq_table

```r
# Compare the proportions of smokers between groups
fisher_test(xtab, detailed = TRUE)
# Pairwise comparison between groups
pairwise_fisher_test(xtab)

# Pairwise proportion tests

# Data: Titanic
xtab <- as.table(rbind(
  c(122, 167, 528, 673),
  c(203, 118, 178, 212)
))
dimnames(xtab) <- list(
  Survived = c("No", "Yes"),
  Class = c("1st", "2nd", "3rd", "Crew")
)
xtab
# Compare the proportion of survived between groups
pairwise_fisher_test(xtab)

# Row-wise proportion tests

# Data: Titanic
xtab <- as.table(rbind(
  c(180, 145), c(179, 106),
  c(510, 196), c(862, 23)
))
dimnames(xtab) <- list(
  Class = c("1st", "2nd", "3rd", "Crew"),
  Gender = c("Male", "Female")
)
xtab
# Compare the proportion of males and females in each category
row_wise_fisher_test(xtab)

# A r x c table Agresti (2002, p. 57) Job Satisfaction
Job <- matrix(c(1,2,1,0, 3,3,6,1, 10,10,14,9, 6,7,12,11), 4, 4,
  dimnames = list(income = c("< 15k", "15-25k", "25-40k", "> 40k"),
                  satisfaction = c("VeryD", "LittleD", "ModerateS", "VeryS")))
fisher_test(Job)
fisher_test(Job, simulate.p.value = TRUE, B = 1e5)
```
**freq_table**  
*Compute Frequency Table*

**Description**
compute frequency table.

**Usage**
freq_table(data, ..., vars = NULL, na.rm = TRUE)

**Arguments**
- **data**: a data frame
- **...**: One or more unquoted expressions (or variable names) separated by commas. Used to specify variables of interest.
- **vars**: optional character vector containing variable names.
- **na.rm**: logical value. If TRUE (default), remove missing values in the variables used to create the frequency table.

**Value**
a data frame

**Examples**
```r
data("ToothGrowth")
ToothGrowth %>% freq_table(supp, dose)
```

**friedman_effsize**  
*Friedman Test Effect Size (Kendall’s W Value)*

**Description**
Compute the effect size estimate (referred to as w) for Friedman test: \( W = \frac{X^2}{N(K-1)} \); where \( W \) is the Kendall’s W value; \( X^2 \) is the Friedman test statistic value; \( N \) is the sample size. \( k \) is the number of measurements per subject.

The Kendall’s W coefficient assumes the value from 0 (indicating no relationship) to 1 (indicating a perfect relationship).

Kendalls uses the Cohen’s interpretation guidelines of \( 0.1 \) \(<\ 0.3 \) (small effect), \( 0.3 \leq \ 0.5 \) (moderate effect) and \( \geq 0.5 \) (large effect)

Confidence intervals are calculated by bootstrap.
friedman_effsize

Usage

friedman_effsize(
  data,
  formula,
  ci = FALSE,
  conf.level = 0.95,
  ci.type = "perc",
  nboot = 1000,
  ...
)

Arguments

data       a data.frame containing the variables in the formula.
formula    a formula of the form a ~ b | c, where a (numeric) is the dependent variable
            name; b is the within-subjects factor variables; and c (factor) is the column name
            containing individuals/subjects identifier. Should be unique per individual.
ci         If TRUE, returns confidence intervals by bootstrap. May be slow.
conf.level The level for the confidence interval.
ci.type    The type of confidence interval to use. Can be any of "norm", "basic", "perc",
            or "bca". Passed to boot::boot.ci.
nboot      The number of replications to use for bootstrap.
...         other arguments passed to the function friedman.test()

Value

return a data frame with some of the following columns:

- .y.: the y variable used in the test.
- n: Sample counts.
- effsize: estimate of the effect size.
- magnitude: magnitude of effect size.
- conf.low, conf.high: lower and upper bound of the effect size confidence interval.

References

Maciej Tomczak and Ewa Tomczak. The need to report effect size estimates revisited. An overview

Examples

# Load data
#:::::::::::::::::::::::::::::::::::::::
data("ToothGrowth")
df <- ToothGrowth %>%
  filter(supp == "VC") %>%
  mutate(id = rep(1:10, 3))
friedman_test

head(df)

# Friedman test effect size
#:::::::::::::::::::::::::::::::::::::::::
df %>% friedman_effsize(len ~ dose | id)

friedman_test  Friedman Rank Sum Test

Description

Provides a pipe-friendly framework to perform a Friedman rank sum test, which is the non-parametric alternative to the one-way repeated measures ANOVA test. Wrapper around the function friedman.test(). Read more: Friedman test in R.

Usage

friedman_test(data, formula, ...)

Arguments

data  a data.frame containing the variables in the formula.
formula  a formula of the form a ~ b | c, where a (numeric) is the dependent variable name; b is the within-subjects factor variables; and c (factor) is the column name containing individuals/subjects identifier. Should be unique per individual.
...  other arguments to be passed to the function friedman.test.

Value

return a data frame with the following columns:

• .y.: the y (dependent) variable used in the test.
• n: sample count.
• statistic: the value of Friedman's chi-squared statistic, used to compute the p-value.
• p: p-value.
• method: the statistical test used to compare groups.

Examples

# Load data
#:::::::::::::::::::::::::::::::::::::::
data("ToothGrowth")
df <- ToothGrowth %>%
  filter(supp == "VC") %>%
  mutate(id = rep(1:10, 3))
head(df)
games_howell_test

# Friedman rank sum test
#:::::::::::::::::::::::::::::::::::::::::
df %>% friedman_test(len ~ dose | id)

games_howell_test

Games Howell Post-hoc Tests

Description

Performs Games-Howell test, which is used to compare all possible combinations of group differences when the assumption of homogeneity of variances is violated. This post hoc test provides confidence intervals for the differences between group means and shows whether the differences are statistically significant.

The test is based on Welch’s degrees of freedom correction and uses Tukey’s studentized range distribution for computing the p-values. The test compares the difference between each pair of means with appropriate adjustment for the multiple testing. So there is no need to apply additional p-value corrections.

Usage

games_howell_test(data, formula, conf.level = 0.95, detailed = FALSE)

Arguments

data a data.frame containing the variables in the formula.
formula a formula of the form x ~ group where x is a numeric variable giving the data values and group is a factor with one or multiple levels giving the corresponding groups. For example, formula = TP53 ~ cancer_group.
conf.level confidence level of the interval.
detailed logical value. Default is FALSE. If TRUE, a detailed result is shown.

Details

The Games-Howell method is an improved version of the Tukey-Kramer method and is applicable in cases where the equivalence of variance assumption is violated. It is a t-test using Welch’s degrees of freedom. This method uses a strategy for controlling the type I error for the entire comparison and is known to maintain the preset significance level even when the size of the sample is different. However, the smaller the number of samples in each group, the it is more tolerant the type I error control. Thus, this method can be applied when the number of samples is six or more.
get_comparisons

Value

return a data frame with some of the following columns:

- .y.: the y (outcome) variable used in the test.
- group1,group2: the compared groups in the pairwise tests.
- n1,n2: Sample counts.
- estimate,conf.low,conf.high: mean difference and its confidence intervals.
- statistic: Test statistic (t-value) used to compute the p-value.
- df: degrees of freedom calculated using Welch’s correction.
- p.adj: adjusted p-value using Tukey’s method.
- method: the statistical test used to compare groups.
- p.adj.signif: the significance level of p-values.

The returned object has an attribute called args, which is a list holding the test arguments.

References


Examples

```r
# Simple test
ToothGrowth %>% games_howell_test(len ~ dose)

# Grouped data
ToothGrowth %>%
  group_by(supp) %>%
  games_howell_test(len ~ dose)
```

get_comparisons

Create a List of Possible Comparisons Between Groups

Description

Create a list of possible pairwise comparisons between groups. If a reference group is specified, only comparisons against reference will be kept.

Usage

```r
get_comparisons(data, variable, ref.group = NULL)
```
get_mode

Compute Mode

Description

Compute the mode in a given vector. Mode is the most frequent value.

Usage

get_mode(x)

Arguments

x a vector. Can be numeric, factor or character vector.
Examples

# Mode of numeric vector
x <- c(1:5, 6, 6, 7:10)
get_mode(x)

# Bimodal
x <- c(1:5, 6, 6, 7, 8, 9, 9, 10)
get_mode(x)

# No mode
x <- c(1, 2, 3, 4, 5)
get_mode(x)

# Nominal vector
fruits <- c(rep("orange", 10), rep("apple", 5), rep("lemon", 2))
get_mode(fruits)

get_pwc_label

Extract Label Information from Statistical Tests

Description

Extracts label information from statistical tests. Useful for labelling plots with test outputs.

Usage

general_pwc_label(stat.test, type = c("expression", "text"))

generate_test_label(
    stat.test,
    description = NULL,
    p.col = "p",
    type = c("expression", "text"),
    correction = c("auto", "GG", "HF", "none"),
    row = NULL,
    detailed = FALSE
)

create_test_label(
    statistic.text,
    statistic,
    p,
    parameter = NA,
    description = NULL,
    n = NA,
    effect.size = NA,
    effect.size.text = NA,
Arguments

stat.test  statistical test results returned by `rstatix` functions.

type  the label type. Can be one of "text" and "expression". Partial match allowed. If you want to add the label onto a ggplot, it might be useful to specify `type = "expression"`.

description  the test description used as the prefix of the label. Examples of description are "ANOVA", "Two Way ANOVA". To remove the default description, specify `description = NULL`. If missing, we'll try to guess the statistical test default description.

p.col  character specifying the column containing the p-value. Default is "p", can be "p.adj".

correction  character, considered only in the case of ANOVA test. Which sphericity correction of the degrees of freedom should be reported for the within-subject factors (repeated measures). The default is set to "GG" corresponding to the Greenhouse-Geisser correction. Possible values are "GG", "HF" (i.e., Hyunh-Feldt correction), "none" (i.e., no correction) and "auto" (apply automatically GG correction if the sphericity assumption is not for within-subject design).

row  numeric, the row index to be considered. If NULL, the last row is automatically considered for ANOVA test.

detailed  logical value. If TRUE, returns detailed label.

statistic.text  character specifying the test statistic. For example `statistic.text = "F"` (for ANOVA test); `statistic.text = "t"` (for t-test).

statistic  the numeric value of a statistic.

p  the p-value of the test.

parameter  string containing the degree of freedom (if exists). Default is NA to accommodate non-parametric tests. For example `parameter = "1,9"` (for ANOVA test. Two parameters exist: DFn and DFd); `parameter = "9"` (for t-test).

n  sample count, example: `n = 10`.

effect.size  the effect size value

effect.size.text  a character specifying the relevant effect size. For example, for Cohens d statistic, `effect.size.text = "d"`. You can also use plotmath expression as follow `quote(italic("d"))`.

Value

a text label or an expression to pass to a plotting function.
get_pwc_label

Functions

- get_pwc_label: Extract label from pairwise comparisons.
- get_test_label: Extract labels for statistical tests.
- create_test_label: Create labels from user specified test results.

Examples

```r
# Load data
#:::::::::::::::::::::::::::::::::::::::
data("ToothGrowth")
df <- ToothGrowth

# One-way ANOVA test
#:::::::::::::::::::::::::::::::::::::::::
anov <- df %>% anova_test(len ~ dose)
get_test_label(anov, detailed = TRUE, type = "text")

# Two-way ANOVA test
#:::::::::::::::::::::::::::::::::::::::::
anov <- df %>% anova_test(len ~ supp*dose)
get_test_label(anov, detailed = TRUE, type = "text",
description = "Two Way ANOVA")

# Kruskal-Wallis test
#:::::::::::::::::::::::::::::::::::::::::
kruskal<- df %>% kruskal_test(len ~ dose)
get_test_label(kruskal, detailed = TRUE, type = "text")

# Wilcoxon test
#:::::::::::::::::::::::::::::::::::::::::
# Unpaired test
wilcox <- df %>% wilcox_test(len ~ supp)
get_test_label(wilcox, detailed = TRUE, type = "text")
# Paired test
wilcox <- df %>% wilcox_test(len ~ supp, paired = TRUE)
get_test_label(wilcox, detailed = TRUE, type = "text")

# T test
#:::::::::::::::::::::::::::::::::::::::::
ttest <- df %>% t_test(len ~ dose)
get_test_label(ttest, detailed = TRUE, type = "text")

# Pairwise comparisons labels
#:::::::::::::::::::::::::::::::::::::::::
get_pwc_label(ttest, type = "text")

# Create test labels
#:::::::::::::::::::::::::::::::::::::::::
```
get_summary_stats

create_test_label(
  statistic.text = "F", statistic = 71.82,
  parameter = "4, 294",
  p = "<0.0001",
  description = "ANOVA",
  type = "text"
)

get_summary_stats
Compute Summary Statistics

Description
Compute summary statistics for one or multiple numeric variables.

Usage

get_summary_stats(
  data,
  ...,
  type = c("full", "common", "robust", "five_number", "mean_sd", "mean_se", "mean_ci",
           "median_iqr", "median_mad", "quantile", "mean", "median", "min", "max"),
  show = NULL,
  probs = seq(0, 1, 0.25)
)

Arguments

data a data frame
...
(Optional) One or more unquoted expressions (or variable names) separated by commas. Used to select a variable of interest. If no variable is specified, then the summary statistics of all numeric variables in the data frame is computed.

type type of summary statistics. Possible values include: "full", "common", "robust", "five_number", "mean_sd", "mean_se", "mean_ci", "median_iqr", "median_mad", "quantile", "mean", "median", "min", "max".

show a character vector specifying the summary statistics you want to show. Example: show = c("n", "mean", "sd"). This is used to filter the output after computation.

probs numeric vector of probabilities with values in [0,1]. Used only when type = "quantile".

Value
A data frame containing descriptive statistics, such as:

- n: the number of individuals
- min: minimum
• **max**: maximum
• **median**: median
• **mean**: mean
• **q1, q3**: the first and the third quartile, respectively.
• **iqr**: interquartile range
• **mad**: median absolute deviation (see ?MAD)
• **sd**: standard deviation of the mean
• **se**: standard error of the mean
• **ci**: 95 percent confidence interval of the mean

**Examples**

```r
# Full summary statistics
data("ToothGrowth")
ToothGrowth %>% get_summary_stats(len)

# Summary statistics of grouped data
# Show only common summary
ToothGrowth %>%
  group_by(dose, supp) %>%
  get_summary_stats(len, type = "common")

# Robust summary statistics
ToothGrowth %>% get_summary_stats(len, type = "robust")

# Five number summary statistics
ToothGrowth %>% get_summary_stats(len, type = "five_number")

# Compute only mean and sd
ToothGrowth %>% get_summary_stats(len, type = "mean_sd")

# Compute full summary statistics but show only mean, sd, median, iqr
ToothGrowth %>%
  get_summary_stats(len, show = c("mean", "sd", "median", "iqr"))
```

---

**get_y_position**

**Autocompute P-value Positions For Plotting Significance**

**Description**

Compute p-value x and y positions for plotting significance levels. Many examples are provided at:

• How to Add P-Values onto a Grouped GG PLOT using the GG PUBLISH R Package
• How to Add Adjusted P-values to a Multi-Panel GGPLOT
• How to Add P-values Generated Elsewhere to a GGPLOT
get_y_position

Usage

get_y_position(
    data,
    formula,
    fun = "max",
    ref.group = NULL,
    comparisons = NULL,
    step.increase = 0.12,
    y.trans = NULL,
    stack = FALSE
)

add_y_position(
    test,
    fun = "max",
    step.increase = 0.12,
    data = NULL,
    formula = NULL,
    ref.group = NULL,
    comparisons = NULL,
    y.trans = NULL,
    stack = FALSE
)

add_x_position(test, x = NULL, dodge = 0.8)

add_xy_position(
    test,
    x = NULL,
    dodge = 0.8,
    stack = FALSE,
    fun = "max",
    step.increase = 0.12,
    ...
)

Arguments

data  a data.frame containing the variables in the formula.
formula  a formula of the form x ~ group where x is a numeric variable giving the data values and group is a factor with one or multiple levels giving the corresponding groups. For example, formula = TP53 ~ cancer_group.
fun  summary statistics functions used to compute automatically suitable y positions of p-value labels and brackets. Possible values include: "max", "mean", "mean_sd", "mean_se", "mean_ci", "median", "median_iqr", "median_mad". For example, if fun = "max", the y positions are guessed as follow:
  • 1. Compute the maximum of each group (groups.maximum)
  • 2. Use the highest groups maximum as the first bracket y position
• 3. Add successively a step increase for remaining bracket y positions. When the main plot is a boxplot, you need the option `fun = "max"`, to have the p-value bracket displayed at the maximum point of the group.

In some situations the main plot is a line plot or a barplot showing the mean±/−error bars of the groups, where error can be SE (standard error), SD (standard deviation) or CI (confidence interval). In this case, to correctly compute the bracket y position you need the option `fun = "mean_se"`, etc.

**ref.group**

A character string specifying the reference group. If specified, for a given grouping variable, each of the group levels will be compared to the reference group (i.e. control group).

**comparisons**

A list of length-2 vectors specifying the groups of interest to be compared. For example to compare groups "A" vs "B" and "B" vs "C", the argument is as follow: `comparisons = list(c("A","B"),c("B","C"))`

**step.increase**

A numeric vector with the increase in fraction of total height for every additional comparison to minimize overlap.

**y.trans**

A function for transforming y axis scale. Value can be `log2`, `log10` and `sqrt`. Can be also any custom function that can take a numeric vector as input and returns a numeric vector, example: `y.trans = function(x){log2(x+1)}`

**stack**

A logical. If TRUE, computes y position for a stacked plot. Useful when dealing with stacked bar plots.

**test**

An object of class `rstatix_test` as returned by `t_test()`, `wilcox_test()`, `sign_test()`, `tukey_hsd()`, `dunn_test()`.

**x**

A variable on x axis.

**dodge**

A dodge width for grouped ggplot/test. Default is 0.8. Used only when x specified.

**...**

Other arguments to be passed to the function `t.test`.

**Functions**

- `get_y_position`: compute the p-value y positions
- `add_y_position`: add p-value y positions to an object of class `rstatix_test`
- `add_x_position`: compute and add p-value x positions.
- `add_xy_position`: compute and add both x and y positions.

**Examples**

```r
# Data preparation
#::::::::::::::::::::::::::::::::::::
df <- ToothGrowth
df$dose <- as.factor(df$dose)
df$group <- factor(rep(c(1, 2), 30))
head(df)

# Stat tests
#::::::::::::::::::::::::::::::::::::
stat.test <- df %>%
  t_test(len ~ dose)
```
identify_outliers

# Add the test into box plots
#:::::::::::::::::::::::::::::::::::::::::
stat.test <- stat.test %>%
  add_y_position()

if(require("ggpubr")){
  ggboxplot(df, x = "dose", y = "len") +
    stat_pvalue_manual(stat.test, label = "p.adj.signif", tip.length = 0.01)
}
Value

- `identify_outliers()`. Returns the input data frame with two additional columns: "is.outlier" and "is.extreme", which hold logical values.
- `is_outlier()` and `is_extreme()`. Returns logical vectors.

Functions

- `identify_outliers`: takes a data frame and extract rows suspected as outliers according to a numeric column. The following columns are added "is.outlier" and "is.extreme".
- `is_outlier`: detect outliers in a numeric vector. Returns logical vector.
- `is_extreme`: detect extreme points in a numeric vector. An alias of `is_outlier()`, where `coef = 3`. Returns logical vector.

Examples

```r
# Generate a demo data
set.seed(123)
demo.data <- data.frame(
sample = 1:20,
score = c(rnorm(19, mean = 5, sd = 2), 50),
gender = rep(c("Male", "Female"), each = 10)
)

# Identify outliers according to the variable score
demo.data %>%
  identify_outliers(score)

# Identify outliers by groups
demo.data %>%
  group_by(gender) %>%
  identify_outliers("score")
```

---

**kruskal_effsize**  
*Kruskal-Wallis Effect Size*

**Description**

Compute the effect size for Kruskal-Wallis test as the eta squared based on the H-statistic: \( \eta^2[H] = (H - k + 1)/(n - k) \); where \( H \) is the value obtained in the Kruskal-Wallis test; \( k \) is the number of groups; \( n \) is the total number of observations.

The eta-squared estimate assumes values from 0 to 1 and multiplied by 100 indicates the percentage of variance in the dependent variable explained by the independent variable. The interpretation values commonly in published literature are: \( 0.01 < 0.06 \) (small effect), \( 0.06 \sim 0.14 \) (moderate effect) and \( >= 0.14 \) (large effect).

Confidence intervals are calculated by bootstrap.
Usage

```
kruskal_effsize(
  data,
  formula,
  ci = FALSE,
  conf.level = 0.95,
  ci.type = "perc",
  nboot = 1000
)
```

Arguments

- **data**: a data.frame containing the variables in the formula.
- **formula**: a formula of the form `x ~ group` where `x` is a numeric variable giving the data values and `group` is a factor with one or multiple levels giving the corresponding groups. For example, `formula = TP53 ~ cancer_group`.
- **ci**: If TRUE, returns confidence intervals by bootstrap. May be slow.
- **conf.level**: The level for the confidence interval.
- **ci.type**: The type of confidence interval to use. Can be any of "norm", "basic", "perc", or "bca". Passed to `boot::boot.ci`.
- **nboot**: The number of replications to use for bootstrap.

Value

return a data frame with some of the following columns:

- `.y.`: the y variable used in the test.
- `n`: Sample counts.
- `effsize`: estimate of the effect size.
- `magnitude`: magnitude of effect size.
- `conf.low`, `conf.high`: lower and upper bound of the effect size confidence interval.

References


http://imaging.mrc-cbu.cam.ac.uk/statswiki/FAQ/effectSize

http://www.psy.gla.ac.uk/~steve/best/effect.html

Examples

```
# Load data
#:::::::::::::::::::::::::::::::::::::::::::::
data("ToothGrowth")
df <- ToothGrowth
```
# Kruskal-wallis rank sum test
#:::::::::::::::::::::::::::::::::::::::::
df %>% kruskal_effsize(len ~ dose)

# Grouped data
df %>%
  group_by(supp) %>%
  kruskal_effsize(len ~ dose)

kruskal_test Ｋｒｕｓｋａｌ-Ｗａｌｌｉｓ Ｔｅｓｔ

Description
Provides a pipe-friendly framework to perform Kruskal-Wallis rank sum test. Wrapper around the function kruskal.test().

Usage
kruskal_test(data, formula, ...)

Arguments
data  a data.frame containing the variables in the formula.
formula  a formula of the form x ~ group where x is a numeric variable giving the data values and group is a factor with one or multiple levels giving the corresponding groups. For example, formula = TP53 ~ cancer_group.
...  other arguments to be passed to the function kruskal.test.

Value
return a data frame with the following columns:
- .y.: the y variable used in the test.
- n: sample count.
- statistic: the kruskal-wallis rank sum statistic used to compute the p-value.
- p: p-value.
- method: the statistical test used to compare groups.

Examples
# Load data
data("ToothGrowth")
df <- ToothGrowth

# Kruskal-wallis rank sum test
levene_test

#:::::::::::::::::::::::::::::::::::::::::
df %>% kruskal_test(len ~ dose)

# Grouped data
df %>%
group_by(supp) %>%
kruskal_test(len ~ dose)

---

levene_test  Levene’s Test

Description

Provide a pipe-friendly framework to easily compute Levene’s test for homogeneity of variance across groups.

Wrapper around the function leveneTest(), which can additionally handles a grouped data.

Usage

levene_test(data, formula, center = median)

Arguments

data  a data frame for evaluating the formula or a model
formula  a formula
center  The name of a function to compute the center of each group; mean gives the original Levene’s test; the default, median, provides a more robust test.

Examples

# Prepare the data
data("ToothGrowth")
df <- ToothGrowth
df$dose <- as.factor(df$dose)
# Compute Levene's Test
df %>% levene_test(len ~ dose)

# Grouped data
df %>%
group_by(supp) %>%
levene_test(len ~ dose)
mahalanobis_distance  
*Compute Mahalanobis Distance and Flag Multivariate Outliers*

**Description**

Pipe-friendly wrapper around the function `mahalanobis()`, which returns the squared Mahalanobis distance of all rows in x. Compared to the base function, it automatically flags multivariate outliers.

Mahalanobis distance is a common metric used to identify multivariate outliers. The larger the value of Mahalanobis distance, the more unusual the data point (i.e., the more likely it is to be a multivariate outlier).

The distance tells us how far an observation is from the center of the cloud, taking into account the shape (covariance) of the cloud as well.

To detect outliers, the calculated Mahalanobis distance is compared against a chi-square (X^2) distribution with degrees of freedom equal to the number of dependent (outcome) variables and an alpha level of 0.001.

The threshold to declare a multivariate outlier is determined using the function `qchisq(0.999, df)` , where df is the degree of freedom (i.e., the number of dependent variable used in the computation).

**Usage**

`mahalanobis_distance(data, ...)`

**Arguments**

- `data` a data frame. Columns are variables.
- `...` One unquoted expressions (or variable name). Used to select a variable of interest. Can be also used to ignore a variable that are not needed for the computation. For example specify `-id` to ignore the id column.

**Value**

Returns the input data frame with two additional columns: 1) "mahal.dist": Mahalanobis distance values; and 2) "is.outlier": logical values specifying whether a given observation is a multivariate outlier

**Examples**

```r
# Compute mahalonobis distance and flag outliers if any
iris %>%
do(~mahalanobis_distance(.))

# Compute distance by groups and filter outliers
iris %>%
group_by(Species) %>%
```
make_clean_names

Description
Pipe-friendly function to make syntactically valid names out of character vectors.

Usage
make_clean_names(data)

Arguments
data  a data frame or vector

Value
a data frame or a vector depending on the input data

Examples

# Vector
make_clean_names(c("a and b", "a-and-b"))
make_clean_names(1:10)

# data frame
df <- data.frame(
  `a and b` = 1:4,
  `c and d` = 5:8,
  check.names = FALSE
)
df
make_clean_names(df)
McNemar’s Chi-squared Test for Count Data

Description

Performs McNemar chi-squared test to compare paired proportions.

Wrappers around the R base function `mcnemar.test()`, but provide pairwise comparisons between multiple groups

Usage

```r
mcnemar_test(x, y = NULL, correct = TRUE)
pairwise_mcnemar_test(
  data, formula,
  type = c("mcnemar", "exact"),
  correct = TRUE,
  p.adjust.method = "bonferroni"
)
```

Arguments

- `x` either a two-dimensional contingency table in matrix form, or a factor object.
- `y` a factor object; ignored if `x` is a matrix.
- `correct` a logical indicating whether to apply continuity correction when computing the test statistic.
- `data` a data frame containing the variables in the formula.
- `formula` a formula of the form `a ~ b | c`, where `a` is the outcome variable name; `b` is the within-subjects factor variables; and `c` (factor) is the column name containing individuals/subjects identifier. Should be unique per individual.
- `type` type of statistical tests used for pairwise comparisons. Allowed values are one of `c("mcnemar", "exact")`.
- `p.adjust.method` method to adjust p values for multiple comparisons. Used when pairwise comparisons are performed. Allowed values include "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". If you don’t want to adjust the p value (not recommended), use `p.adjust.method = "none"`.

Value

return a data frame with the following columns:

- `n`: the number of participants.
- `statistic`: the value of McNemar’s statistic.
mcnemar_test

- df: the degrees of freedom of the approximate chi-squared distribution of the test statistic.
- p: p-value.
- p.adj: the adjusted p-value.
- method: the used statistical test.
- p.signif: the significance level of p-values.

The returned object has an attribute called args, which is a list holding the test arguments.

Functions

- mcnemar_test: performs McNemar’s chi-squared test for comparing two paired proportions
- pairwise_mcnemar_test: performs pairwise McNemar’s chi-squared test between multiple groups. Could be used for post-hoc tests following a significant Cochran’s Q test.

Examples

```r
# Comparing two paired proportions
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# Data: frequencies of smokers before and after interventions
xtab <- as.table(rbind(c(25, 6), c(21,10)))
dimnames(xtab) <- list(before = c("non.smoker", "smoker"),
                        after = c("non.smoker", "smoker"))
xtab
# Compare the proportion of smokers
mcnemar_test(xtab)

# Comparing multiple related proportions
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# Generate a demo data
mydata <- data.frame(outcome = c(0,1,1,0,1,1,1,1,1,1,0,1,0,1,1,1,1,0,1,0,1,1,0,1,0,1,1,0,1,1,0,0,1),
                     treatment = gl(3,1,30,labels=LETTERS[1:3]),
                     participant = gl(10,3,labels=letters[1:10]))
mydata$outcome <- factor(mydata$outcome, levels = c(1, 0),
                         labels = c("success", "failure")
)
# Cross-tabulation
xtabs(~outcome + treatment, mydata)
# Compare the proportion of success between treatments
cochran_qtest(mydata, outcome ~ treatment|participant)
# pairwise comparisons between groups
```
multinom_test

pairwise_mcnemar_test(mydata, outcome ~ treatment|participant)

---

**multinom_test**

**Exact Multinomial Test**

**Description**

Performs an exact multinomial test. Alternative to the chi-square test of goodness-of-fit-test when the sample size is small.

**Usage**

```r
multinom_test(x, p = rep(1/length(x), length(x)), detailed = FALSE)
```

**Arguments**

- `x` numeric vector containing the counts.
- `p` a vector of probabilities of success. The length of `p` must be the same as the number of groups specified by `x`, and its elements must be greater than 0 and less than 1.
- `detailed` logical value. Default is FALSE. If TRUE, a detailed result is shown.

**Value**

return a data frame containing the p-value and its significance.

The **returned object has an attribute called args**, which is a list holding the test arguments.

**See Also**

`binom_test`

**Examples**

```r
# Data
tulip <- c(red = 81, yellow = 50, white = 27)

# Question 1: are the color equally common ?
#-------------------------------------------------------------
# this is a test of homogeneity
res <- multinom_test(tulip)
res

attr(res, "descriptives")

# Pairwise comparisons between groups
pairwise_binom_test(tulip, p.adjust.method = "bonferroni")
```
# Question 2: comparing observed to expected proportions
# this is a goodness-of-fit test
expected.p <- c(red = 0.5, yellow = 0.33, white = 0.17)
res <- multinom_test(tulip, expected.p)
res
attr(res, "descriptives")

# Pairwise comparisons against a given probabilities
pairwise_binom_test_against_p(tulip, expected.p)

---

**prop_test**  
*Proportion Test*

### Description

Performs proportion tests to either evaluate the homogeneity of proportions (probabilities of success) in several groups or to test that the proportions are equal to certain given values.

Wrappers around the R base function `prop.test()` but have the advantage of performing pairwise and row-wise z-test of two proportions, the post-hoc tests following a significant chi-square test of homogeneity for 2xc and rx2 contingency tables.

### Usage

```r
prop_test(
  x,
  n = NULL,
  p = NULL,
  alternative = c("two.sided", "less", "greater"),
  correct = TRUE,
  conf.level = 0.95,
  detailed = FALSE
)
```

```r
pairwise_prop_test(xtab, p.adjust.method = "holm", ...)
```

```r
row_wise_prop_test(xtab, p.adjust.method = "holm", detailed = FALSE, ...)
```

### Arguments

- `x`  
a vector of counts of successes, a one-dimensional table with two entries, or a two-dimensional table (or matrix) with 2 columns, giving the counts of successes and failures, respectively.

- `n`  
a vector of counts of trials; ignored if `x` is a matrix or a table.

- `p`  
a vector of probabilities of success. The length of `p` must be the same as the number of groups specified by `x`, and its elements must be greater than 0 and less than 1.
alternative  a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter. Only used for testing the null that a single proportion equals a given value, or that two proportions are equal; ignored otherwise.

correct  a logical indicating whether Yates’ continuity correction should be applied where possible.

conf.level  confidence level of the returned confidence interval. Must be a single number between 0 and 1. Only used when testing the null that a single proportion equals a given value, or that two proportions are equal; ignored otherwise.

detailed logical value. Default is FALSE. If TRUE, a detailed result is shown.

xtab  a cross-tabulation (or contingency table) with two columns and multiple rows (r×2 design). The columns give the counts of successes and failures respectively.

p.adjust.method  method to adjust p values for multiple comparisons. Used when pairwise comparisons are performed. Allowed values include "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". If you don’t want to adjust the p value (not recommended), use p.adjust.method = "none".

... Other arguments passed to the function prop_test().

Value

return a data frame with some the following columns:

- n: the number of participants.
- group: the categories in the row-wise proportion tests.
- statistic: the value of Pearson’s chi-squared test statistic.
- df: the degrees of freedom of the approximate chi-squared distribution of the test statistic.
- p: p-value.
- p.adj: the adjusted p-value.
- method: the used statistical test.
- p.signif, p.adj.signif: the significance level of p-values and adjusted p-values, respectively.
- estimate: a vector with the sample proportions x/n.
- estimate1, estimate2: the proportion in each of the two populations.
- alternative: a character string describing the alternative hypothesis.
- conf.low, conf.high: Lower and upper bound on a confidence interval. a confidence interval for the true proportion if there is one group, or for the difference in proportions if there are 2 groups and p is not given, or NULL otherwise. In the cases where it is not NULL, the returned confidence interval has an asymptotic confidence level as specified by conf.level, and is appropriate to the specified alternative hypothesis.

The returned object has an attribute called args, which is a list holding the test arguments.
prop_test

Functions

- **prop_test**: performs one-sample and two-samples z-test of proportions. Wrapper around the function `prop.test()`.
- **pairwise_prop_test**: pairwise comparisons between proportions, a post-hoc tests following a significant chi-square test of homogeneity for 2x2 design. Wrapper around `pairwise.prop.test()`.
- **row_wise_prop_test**: performs row-wise z-test of two proportions, a post-hoc tests following a significant chi-square test of homogeneity for r x 2 contingency table. The z-test of two proportions is calculated for each category (row).

Examples

```r
# Comparing an observed proportion to an expected proportion
prop_test(x = 95, n = 160, p = 0.5, detailed = TRUE)

# Comparing two proportions
prop_test(x = 95, n = 160, p = 0.5, detailed = TRUE)

# Data: frequencies of smokers between two groups
xtab <- as.table(rbind(c(490, 10), c(400, 100)))
dimnames(xtab) <- list(
  group = c("grp1", "grp2"),
  smoker = c("yes", "no")
)
xtab

# compare the proportion of smokers
prop_test(xtab, detailed = TRUE)

# Homogeneity of proportions between groups
prop_test(xtab, detailed = TRUE)

# H0: the proportion of smokers is similar in the four groups
# Ha: this proportion is different in at least one of the populations.

# Data preparation
grp.size <- c(106, 113, 156, 102)
smokers <- c(50, 100, 139, 80)
no.smokers <- grp.size - smokers
xtab <- as.table(rbind(
  smokers,
  no.smokers
))
dimnames(xtab) <- list(
  Smokers = c("Yes", "No"),
  Groups = c("grp1", "grp2", "grp3", "grp4")
)
xtab

# Compare the proportions of smokers between groups
prop_test(xtab, detailed = TRUE)

# Pairwise comparison between groups
pairwise_prop_test(xtab)
```
# Pairwise proportion tests
# Titanic
xtab <- as.table(rbind(
  c(122, 167, 528, 673),
  c(203, 118, 178, 212)
))
dimnames(xtab) <- list(
  Survived = c("No", "Yes"),
  Class = c("1st", "2nd", "3rd", "Crew")
)
xtab
# Compare the proportion of survived between groups
pairwise_prop_test(xtab)

# Row-wise proportion tests
# Titanic
xtab <- as.table(rbind(
  c(180, 145), c(179, 106),
  c(510, 196), c(862, 23)
))
dimnames(xtab) <- list(
  Class = c("1st", "2nd", "3rd", "Crew"),
  Gender = c("Male", "Female")
)
xtab
# Compare the proportion of males and females in each category
row_wise_prop_test(xtab)

prop_trend_test

---

**prop_trend_test**  
*Test for Trend in Proportions*

**Description**

Performs chi-squared test for trend in proportion. This test is also known as Cochran-Armitage trend test.

Wrappers around the R base function `prop.trend.test()` but returns a data frame for easy data visualization.

**Usage**

`prop_trend_test(xtab, score = NULL)`

**Arguments**

- `xtab`  
  a cross-tabulation (or contingency table) with two columns and multiple rows (rx2 design). The columns give the counts of successes and failures respectively.

- `score`  
  group score. If NULL, the default is group number.
**pull_triangle**

### Value

return a data frame with some the following columns:

- **n**: the number of participants.
- **statistic**: the value of Chi-squared trend test statistic.
- **df**: the degrees of freedom.
- **p**: p-value.
- **method**: the used statistical test.
- **p.signif**: the significance level of p-values and adjusted p-values, respectively.

The returned object has an attribute called args, which is a list holding the test arguments.

### Examples

```r
# Proportion of renal stone (calculi) across age
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# Data
xtab <- as.table(rbind(
  c(384, 536, 335),
  c(951, 869, 438)
))
dimnames(xtab) <- list(
  stone = c("yes", "no"),
  age = c("30-39", "40-49", "50-59")
)
xtab
# Compare the proportion of survived between groups
prop_trend_test(xtab)
```

---

**pull_triangle**

Pull Lower and Upper Triangular Part of a Matrix

### Description

Returns the lower or the upper triangular part of a (correlation) matrix.

### Usage

```r
pull_triangle(x, triangle = c("lower", "upper"), diagonal = FALSE)
pull_upper_triangle(x, diagonal = FALSE)
pull_lower_triangle(x, diagonal = FALSE)
```
**Arguments**

- **x**: a (correlation) matrix
- **triangle**: the triangle to pull. Allowed values are one of "upper" and "lower".
- **diagonal**: logical. Default is FALSE. If TRUE, the matrix diagonal is included.

**Value**

an object of class `cor_mat_tri`, which is a data frame

**Functions**

- `pull_triangle`: returns either the lower or upper triangular part of a matrix.
- `pull_upper_triangle`: returns an object of class `upper_tri`, which is a data frame containing the upper triangular part of a matrix.
- `pull_lower_triangle`: returns an object of class `lower_tri`, which is a data frame containing the lower triangular part of a matrix.

**See Also**

`replace_triangle()`

**Examples**

```r
# Data preparation
#::::::::::::::::::::::::::::::::::::::::::::::::::
mydata <- mtcars %>%
  select(mpg, disp, hp, drat, wt, qsec)
head(mydata, 3)

# Compute correlation matrix and pull triangles
#:::::::::::::::::::::::::::::::::::::::::::::::::
# Correlation matrix
cor.mat <- cor_mat(mydata)
cor.mat

cor.mat %>% pull_lower_triangle()

cor.mat %>% pull_upper_triangle()
```
Description

Round and format p-values. Can also mark significant p-values with stars.

Usage

p_round(x, ..., digits = 3)

p_format(
  x,
  ...,
  new.col = FALSE,
  digits = 2,
  accuracy = 1e-04,
  decimal.mark = ",",
  leading.zero = TRUE,
  trailing.zero = FALSE,
  add.p = FALSE,
  space = FALSE
)

p_mark_significant(
  x,
  ...,
  new.col = FALSE,
  cutpoints = c(0, 1e-04, 0.001, 0.01, 0.05, 1),
  symbols = c("****", "***", "**", "*", "")
)

p_detect(data, type = c("all", "p", "p.adj"))

p_names()

p_adj_names()

Arguments

x                  a numeric vector of p-values or a data frame containing a p value column. If data frame, the p-value column(s) will be automatically detected. Known p-value column names can be obtained using the functions p_names() and p_adj_names()

...                column names to manipulate in the case where x is a data frame. P value columns are automatically detected if not specified.

digits            the number of significant digits to be used.
new.col logical, used only when x is a data frame. If TRUE, add a new column to hold the results. The new column name is created by adding, to the p column, the suffix "format" (for p_format()), "signif" (for p_mark_significant()).

accuracy number to round to, that is the threshold value above which the function will replace the p-value by "<0.0xxx".

decimal.mark the character to be used to indicate the numeric decimal point.

leading.zero logical. If FALSE, remove the leading zero.

trailing.zero logical. If FALSE (default), remove the trailing extra zero.

add.p logical value. If TRUE, add "p=" before the value.

space logical. If TRUE (default) use space as separator between different elements and symbols.

cutpoints numeric vector used for intervals

symbols character vector, one shorter than cutpoints, used as significance symbols.

data a data frame

type the type of p-value to detect. Can be one of c("all","p","p.adj").

Value

a vector or a data frame containing the rounded/formatted p-values.

Functions

• p_round: round p-values
• p_format: format p-values. Add a symbol "<" for small p-values.
• p_mark_significant: mark p-values with significance levels
• p_detect: detects and returns p-value column names in a data frame.
• p_names: returns known p-value column names
• p_adj_names: returns known adjust p-value column names

Examples

# Round and format a vector of p-values
# :::::::::::::::::::::::::::::::::::::::::::::::::::
# Format
p <- c(0.5678, 0.127, 0.045, 0.011, 0.009, 0.00002, NA)
p_format(p)

# Specify the accuracy
p_format(p, accuracy = 0.01)

# Add p and remove the leading zero
p_format(p, add.p = TRUE, leading.zero = FALSE)

# Remove space before and after "=" or ",".
p_format(p, add.p = TRUE, leading.zero = FALSE, space = FALSE)
# Mark significant p-values
# ::::::::::::::::::::::::::::::::::::::::::::::::
p_mark_significant(p)

# Round, then mark significant
p %>% p_round(digits = 2) %>% p_mark_significant()

# Format, then mark significant
p %>% p_format(digits = 2) %>% p_mark_significant()

# Perform stat test, format p and mark significant
# ::::::::::::::::::::::::::::::::::::::::::::::::
ToothGrowth %>%
  group_by(dose) %>%
  t_test(len ~ supp) %>%
  p_format(digits = 2, leading.zero = FALSE) %>%
  p_mark_significant()

---

**replace_triangle**

Replace Lower and Upper Triangular Part of a Matrix

**Description**

Replace the lower or the upper triangular part of a (correlation) matrix.

**Usage**

```r
replace_triangle(x, triangle = c("lower", "upper"), by = "", diagonal = FALSE)
```

```r
replace_upper_triangle(x, by = "", diagonal = FALSE)
```

```r
replace_lower_triangle(x, by = "", diagonal = FALSE)
```

**Arguments**

- `x`: a (correlation) matrix
- `triangle`: the triangle to replace. Allowed values are one of "upper" and "lower".
- `by`: a replacement argument. Appropriate values are either "" or NA. Used to replace the upper, lower or the diagonal part of the matrix.
- `diagonal`: logical. Default is FALSE. If TRUE, the matrix diagonal is included.

**Value**

an object of class cor_mat_tri, which is a data frame
Functions

- `replace_triangle`: replaces the specified triangle by empty or NA.
- `replace_upper_triangle`: replaces the upper triangular part of a matrix. Returns an object of class `lower_tri`.
- `replace_lower_triangle`: replaces the lower triangular part of a matrix. Returns an object of class `lower_tri`.

See Also

`pull_triangle()`

Examples

```r
# Compute correlation matrix and pull triangles
#::::::::::::::::::::::::::::::::::::::::::
# Correlation matrix
cor.mat <- mtcars %>%
  select(mpg, disp, hp, drat, wt, qsec) %>
  cor_mat()
cor.mat

# Replace upper triangle by NA
#::::::::::::::::::::::::::::::::::::::::::
cor.mat %>%
  replace_upper_triangle(by = NA)

# Replace upper triangle by NA and reshape the
# correlation matrix to have unique combinations of variables
#::::::::::::::::::::::::::::::::::::::::::
cor.mat %>
  replace_upper_triangle(by = NA) %>
  cor_gather()
```

---

**sample_n_by**  
*Sample n Rows By Group From a Table*

Description

Sample n rows by group from a table using the `sample_n()` function.

Usage

`sample_n_by(data, ..., size = 1, replace = FALSE)`
**Arguments**

- `data` a data frame
- `...` Variables to group by
- `size` the number of rows to select
- `replace` with or without replacement?

**Examples**

```r
ToothGrowth %>% sample_n_by(dose, supp, size = 2)
```

---

**shapiro_test**  
*Shapiro-Wilk Normality Test*

**Description**

Provides a pipe-friendly framework to perform Shapiro-Wilk test of normality. Support grouped data and multiple variables for multivariate normality tests. Wrapper around the R base function `shapiro.test()`. Can handle grouped data. Read more: Normality Test in R.

**Usage**

```r
shapiro_test(data, ..., vars = NULL)
mshapiro_test(data)
```

**Arguments**

- `data` a data frame. Columns are variables.
- `...` One or more unquoted expressions (or variable names) separated by commas. Used to select a variable of interest.
- `vars` optional character vector containing variable names. Ignored when dot vars are specified.

**Value**

a data frame containing the value of the Shapiro-Wilk statistic and the corresponding p.value.

**Functions**

- `shapiro_test`: univariate Shapiro-Wilk normality test
- `mshapiro_test`: multivariate Shapiro-Wilk normality test. This is a modified copy of the `mshapiro.test()` function of the package mvnormtest, for internal convenience.
### Examples

- # Shapiro Wilk normality test for one variable
  ```r
  iris %>% shapiro_test(Sepal.Length)
  ```

- # Shapiro Wilk normality test for two variables
  ```r
  iris %>% shapiro_test(Sepal.Length, Petal.Width)
  ```

- # Multivariate normality test
  ```r
  mshapiro_test(iris[, 1:3])
  ```

### Description

Performs one-sample and two-sample sign tests. Read more: [Sign Test in R](#).

### Usage

```r
sign_test(
  data, formula, comparisons = NULL, ref.group = NULL, p.adjust.method = "holm", alternative = "two.sided", mu = 0, conf.level = 0.95, detailed = FALSE
)
```

```r
pairwise_sign_test(
  data, formula, comparisons = NULL, ref.group = NULL, p.adjust.method = "holm", detailed = FALSE, ...
)
```

### Arguments

- **data**: a data.frame containing the variables in the formula.
**formula**
a formula of the form \( x \sim \text{group} \) where \( x \) is a numeric variable giving the data values and \text{group} is a factor with one or multiple levels giving the corresponding groups. For example, \( \text{formula} = \text{TP53} \sim \text{treatment} \).

**comparisons**
A list of length-2 vectors specifying the groups of interest to be compared. For example to compare groups "A" vs "B" and "B" vs "C", the argument is as follow: \( \text{comparisons} = \text{list(c("A","B"),c("B","C"))} \)

**ref.group**
a character string specifying the reference group. If specified, for a given grouping variable, each of the group levels will be compared to the reference group (i.e. control group).

**p.adjust.method**
method to adjust p values for multiple comparisons. Used when pairwise comparisons are performed. Allowed values include "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". If you don’t want to adjust the p value (not recommended), use \( \text{p.adjust.method} = \text{"none"} \).

**alternative**
a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter.

**mu**
a single number representing the value of the population median specified by the null hypothesis.

**conf.level**
confidence level of the interval.

**detailed**
logical value. Default is \text{FALSE}. If \text{TRUE}, a detailed result is shown.

... other arguments passed to the function \text{sign_test}()

**Value**
return a data frame with some the following columns:

- \( .y. \): the y variable used in the test.
- group1,group2: the compared groups in the pairwise tests.
- n,n1,n2: Sample counts.
- statistic: Test statistic used to compute the p-value. That is the \( S \)-statistic (the number of positive differences between the data and the hypothesized median), with names attribute "S".
- df,parameter: degrees of freedom. Here, the total number of valid differences.
- p: p-value.
- method: the statistical test used to compare groups.
- p.signif,p.adj.signif: the significance level of p-values and adjusted p-values, respectively.
- estimate: estimate of the effect size. It corresponds to the median of the differences.
- alternative: a character string describing the alternative hypothesis.
- conf.low,conf.high: Lower and upper bound on a confidence interval of the estimate.

The returned object has an attribute called \text{args}, which is a list holding the test arguments.

**Functions**

- \text{sign_test}: Sign test
- \text{pairwise_sign_test}: performs pairwise two sample Wilcoxon test.
Note

This function is a reimplementation of the function SignTest() from the DescTools package.

Examples

# Load data
#:::::::::::::::::::::::::::::::::::::::
data("ToothGrowth")
df <- ToothGrowth

# One-sample test
#:::::::::::::::::::::::::::::::::::::::
df %>% sign_test(len ~ 1, mu = 0)

# Two-samples paired test
#:::::::::::::::::::::::::::::::::::::::
df %>% sign_test(len ~ supp)

# Compare supp levels after grouping the data by "dose"
#::::::::::::::::::::::::::::::::::::::::
df %>%
  group_by(dose) %>%
  sign_test(data =., len ~ supp) %>%
  adjust_pvalue(method = "bonferroni") %>%
  add_significance("p.adj")

# pairwise comparisons
#::::::::::::::::::::::::::::::::::::::::
# As dose contains more than two levels =>
# pairwise test is automatically performed.
df %>% sign_test(len ~ dose)

# Comparison against reference group
#::::::::::::::::::::::::::::::::::::::::
# each level is compared to the ref group
df %>% sign_test(len ~ dose, ref.group = "0.5")

---

tukey_hsd  

Tukey Honest Significant Differences

Description

Provides a pipe-friendly framework to performs Tukey post-hoc tests. Wrapper around the function TukeyHSD(). It is essentially a t-test that corrects for multiple testing.

Can handle different inputs formats: aov, lm, formula.
Usage

tukey_hsd(x, ...)

## Default S3 method:
tukey_hsd(x, ...)

## S3 method for class 'lm'
tukey_hsd(x, ...)

## S3 method for class 'data.frame'
tukey_hsd(x, formula, ...)

Arguments

x          an object of class aov, lm or data.frame containing the variables used in the formula.

...           other arguments passed to the function TukeyHSD(). These include:

  • which: A character vector listing terms in the fitted model for which the intervals should be calculated. Defaults to all the terms.
  • ordered: A logical value indicating if the levels of the factor should be ordered according to increasing average in the sample before taking differences. If ordered is true then the calculated differences in the means will all be positive. The significant differences will be those for which the lwr end point is positive.

formula      a formula of the form \( x \sim group \) where \( x \) is a numeric variable giving the data values and group is a factor with one or multiple levels giving the corresponding groups. For example, \( formula = TP53 \sim cancer\_group \).

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

Value

a tibble data frame containing the results of the different comparisons.

Methods (by class)

• default: performs tukey post-hoc test from aov() results.
• lm: performs tukey post-hoc test from lm() model.
• data.frame: performs tukey post-hoc tests using data and formula as inputs. ANOVA will be automatically performed using the function aov()

Examples

# Data preparation
df <- ToothGrowth
df$dose <- as.factor(df$dose)
# Tukey HSD from ANOVA results
aov(len ~ dose, data = df) %>% tukey_hsd()
# two-way anova with interaction
aov(len ~ dose*supp, data = df) %>% tukey_hsd()

# Tukey HSD from lm() results
lm(len ~ dose, data = df) %>% tukey_hsd()

# Tukey HSD from data frame and formula
tukey_hsd(df, len ~ dose)

# Tukey HSD using grouped data
df %>%
  group_by(supp) %>%
  tukey_hsd(len ~ dose)

---

**t_test**

**T-test**

**Description**

Provides a pipe-friendly framework to perform one and two sample t-tests. Read more: [T-test in R](https://www.r-project.org/doc/library/stats/html/t.test.html).

**Usage**

```r
R

t_test(
  data,
  formula,
  comparisons = NULL,
  ref.group = NULL,
  p.adjust.method = "holm",
  paired = FALSE,
  var.equal = FALSE,
  alternative = "two.sided",
  mu = 0,
  conf.level = 0.95,
  detailed = FALSE
)

pairwise_t_test(
  data,
  formula,
  comparisons = NULL,
  ref.group = NULL,
  p.adjust.method = "holm",
  paired = FALSE,
  pool.sd = !paired,
```

```R
```
Arguments

data

a data.frame containing the variables in the formula.

formula

a formula of the form \( x \sim group \) where \( x \) is a numeric variable giving the data values and group is a factor with one or multiple levels giving the corresponding groups. For example, formula = TP53 ~ cancer_group.

comparisons

A list of length-2 vectors specifying the groups of interest to be compared. For example to compare groups "A" vs "B" and "B" vs "C", the argument is as follow: comparisons = list(c("A", "B"), c("B", "C"))

ref.group

a character string specifying the reference group. If specified, for a given grouping variable, each of the group levels will be compared to the reference group (i.e. control group).

If ref.group = "all", pairwise two sample tests are performed for comparing each grouping variable levels against all (i.e. basemean).

p.adjust.method

method to adjust p values for multiple comparisons. Used when pairwise comparisons are performed. Allowed values include "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". If you don't want to adjust the p value (not recommended), use p.adjust.method = "none".

paired

a logical indicating whether you want a paired test.

var.equal

a logical variable indicating whether to treat the two variances as being equal. If TRUE then the pooled variance is used to estimate the variance otherwise the Welch (or Satterthwaite) approximation to the degrees of freedom is used.

alternative

a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter.

mu

a number specifying an optional parameter used to form the null hypothesis.

conf.level

confidence level of the interval.

detailed

logical value. Default is FALSE. If TRUE, a detailed result is shown.

pool.sd

logical value used in the function pairwise_t_test(). Switch to allow/disallow the use of a pooled SD.

The pool.sd = TRUE (default) calculates a common SD for all groups and uses that for all comparisons (this can be useful if some groups are small). This method does not actually call t.test, so extra arguments are ignored. Pooling does not generalize to paired tests so pool.sd and paired cannot both be TRUE.

If pool.sd = FALSE the standard two sample t-test is applied to all possible pairs of groups. This method calls the t.test(), so extra arguments, such as var.equal are accepted.

other arguments to be passed to the function t.test.
Details
- If a list of comparisons is specified, the result of the pairwise tests is filtered to keep only the comparisons of interest. The p-value is adjusted after filtering.
- For a grouped data, if pairwise test is performed, then the p-values are adjusted for each group level independently.

Value
return a data frame with some the following columns:

- .y.: the y variable used in the test.
- group1, group2: the compared groups in the pairwise tests.
- n, n1, n2: Sample counts.
- statistic: Test statistic used to compute the p-value.
- df: degrees of freedom.
- p: p-value.
- p.adj: the adjusted p-value.
- method: the statistical test used to compare groups.
- p.signif, p.adj.signif: the significance level of p-values and adjusted p-values, respectively.
- estimate: estimate of the effect size. It corresponds to the estimated mean or difference in means depending on whether it was a one-sample test or a two-sample test.
- estimate1, estimate2: show the mean values of the two groups, respectively, for independent samples t-tests.
- alternative: a character string describing the alternative hypothesis.
- conf.low, conf.high: Lower and upper bound on a confidence interval.

The returned object has an attribute called args, which is a list holding the test arguments.

Functions
- t_test: t test
- pairwise_t_test: performs pairwise two sample t-test. Wrapper around the R base function pairwise.t.test.

Examples
```r
# Load data
#:::::::::::::::::::::::::::::::::::::::
data("ToothGrowth")
df <- ToothGrowth

# One-sample test
#:::::::::::::::::::::::::::::::::::::::::
df %>% t_test(len ~ 1, mu = 0)
```
# Two-samples unpaired test
#:::::::::::::::::::::::::::::::::::::::::
df %>% t_test(len ~ supp)

# Two-samples paired test
#:::::::::::::::::::::::::::::::::::::::::
df %>% t_test (len ~ supp, paired = TRUE)

# Compare supp levels after grouping the data by "dose"
#::::::::::::::::::::::::::::::::::::::::
df %>%
  group_by(dose) %>%
  t_test(data =., len ~ supp) %>%
  adjust_pvalue(method = "bonferroni") %>%
  add_significance("p.adj")

# pairwise comparisons
#::::::::::::::::::::::::::::::::::::::::
# As dose contains more than two levels ==>  
# pairwise test is automatically performed.
df %>% t_test(len ~ dose)

# Comparison against reference group
#::::::::::::::::::::::::::::::::::::::::
# each level is compared to the ref group
df %>% t_test(len ~ dose, ref.group = "0.5")

# Comparison against all
#::::::::::::::::::::::::::::::::::::::::
df %>% t_test(len ~ dose, ref.group = "all")

---

welch_anova_test  Welch One-Way ANOVA Test

**Description**

Tests for equal means in a one-way design (not assuming equal variance). A wrapper around the base function `oneway.test()`. This is an alternative to the standard one-way ANOVA in the situation where the homogeneity of variance assumption is violated.

**Usage**

```
welch_anova_test(data, formula)
```

**Arguments**

- `data`  
a data frame containing the variables in the formula.
wilcox_effsize

formula

a formula specifying the ANOVA model similar to aov. Can be of the form y ~ group where y is a numeric variable giving the data values and group is a factor with one or multiple levels giving the corresponding groups. For example, formula = TP53 ~ cancer_group.

Value

return a data frame with the following columns:

• .y.: the y variable used in the test.
• n: sample count.
• statistic: the value of the test statistic.
• p: p-value.
• method: the statistical test used to compare groups.

Examples

# Load data
#:::::::::::::::::::::::::::::::::::::::
data("ToothGrowth")
df <- ToothGrowth
df$dose <- as.factor(df$dose)

# Welch one-way ANOVA test (not assuming equal variance)
#:::::::::::::::::::::::::::::::::::::::::
# Grouped data
#:::::::::::::::::::::::::::::::::::::::::
df %>%
group_by(supp) %>%
welch_anova_test(len ~ dose)

wilcox_effsize

Wilcoxon Effect Size

Description

Compute Wilcoxon effect size (r) for:

• one-sample test (Wilcoxon one-sample signed-rank test);
• paired two-samples test (Wilcoxon two-sample paired signed-rank test) and
• independent two-samples test (Mann-Whitney, two-sample rank-sum test).

It can also returns confidence intervals by bootstrap.

The effect size r is calculated as Z statistic divided by square root of the sample size (N) (Z/√N). The Z value is extracted from either coin::wilcoxsign_test() (case of one- or paired-samples test) or coin::wilcox_test() (case of independent two-samples test).
Note that N corresponds to total sample size for independent samples test and to total number of pairs for paired samples test.

The \( r \) value varies from 0 to close to 1. The interpretation values for \( r \) commonly in published literature and on the internet are: \( 0.10 \leq r < 0.3 \) (small effect), \( 0.30 \leq r < 0.5 \) (moderate effect) and \( r \geq 0.5 \) (large effect).

### Usage

```r
wilcox_effsize(
  data,
  formula,
  comparisons = NULL,
  ref.group = NULL,
  paired = FALSE,
  alternative = "two.sided",
  mu = 0,
  ci = FALSE,
  conf.level = 0.95,
  ci.type = "perc",
  nboot = 1000,
  ...
)
```

### Arguments

- **data**: a data.frame containing the variables in the formula.
- **formula**: a formula of the form \( x \sim group \) where \( x \) is a numeric variable giving the data values and \( group \) is a factor with one or multiple levels giving the corresponding groups. For example, `formula = TP53 ~ cancer_group`.
- **comparisons**: A list of length-2 vectors specifying the groups of interest to be compared. For example to compare groups "A" vs "B" and "B" vs "C", the argument is as follow: `comparisons = list(c("A","B"),c("B","C"))`.
- **ref.group**: a character string specifying the reference group. If specified, for a given grouping variable, each of the group levels will be compared to the reference group (i.e. control group).
  - If `ref.group = "all"`, pairwise two sample tests are performed for comparing each grouping variable levels against all (i.e. basemean).
- **paired**: a logical indicating whether you want a paired test.
- **alternative**: a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter.
- **mu**: a number specifying an optional parameter used to form the null hypothesis.
- **ci**: If TRUE, returns confidence intervals by bootstrap. May be slow.
- **conf.level**: The level for the confidence interval.
- **ci.type**: The type of confidence interval to use. Can be any of "norm", "basic", "perc", or "bca". Passed to `boot::boot.ci`.
- **nboot**: The number of replications to use for bootstrap.
Additional arguments passed to the functions `coin::wilcoxon.test` (case of one- or paired-samples test) or `coin::wilcox.test` (case of independent two-samples test).

**Value**

Return a data frame with some of the following columns:

- `.y.`: the y variable used in the test.
- `group1, group2`: the compared groups in the pairwise tests.
- `n, n1, n2`: Sample counts.
- `effsize`: estimate of the effect size (r value).
- `magnitude`: magnitude of effect size.
- `conf.low, conf.high`: lower and upper bound of the effect size confidence interval.

**References**


**Examples**

```r
if(require("coin")){

  # One-sample Wilcoxon test effect size
  ToothGrowth %>% wilcox_effsize(len ~ 1, mu = 0)

  # Independent two-samples wilcoxon effect size
  ToothGrowth %>% wilcox_effsize(len ~ supp)

  # Paired-samples wilcoxon effect size
  ToothGrowth %>% wilcox_effsize(len ~ supp, paired = TRUE)

  # Pairwise comparisons
  ToothGrowth %>% wilcox_effsize(len ~ dose)

  # Grouped data
  ToothGrowth %>%
    group_by(supp) %>%
    wilcox_effsize(len ~ dose)
}
```
Wilcoxon Tests

Description

Provides a pipe-friendly framework to perform one and two sample Wilcoxon tests. Read more: Wilcoxon in R.

Usage

wilcox_test(
  data,
  formula,
  comparisons = NULL,
  ref.group = NULL,
  p.adjust.method = "holm",
  paired = FALSE,
  exact = NULL,
  alternative = "two.sided",
  mu = 0,
  conf.level = 0.95,
  detailed = FALSE
)

pairwise_wilcox_test(
  data,
  formula,
  comparisons = NULL,
  ref.group = NULL,
  p.adjust.method = "holm",
  detailed = FALSE,
  ...
)

Arguments

data a data.frame containing the variables in the formula.
formula a formula of the form x ~ group where x is a numeric variable giving the data values and group is a factor with one or multiple levels giving the corresponding groups. For example, formula = TP53 ~ cancer_group.
comparisons A list of length-2 vectors specifying the groups of interest to be compared. For example to compare groups "A" vs "B" and "B" vs "C", the argument is as follow: comparisons = list(c("A","B"),c("B","C"))
ref.group a character string specifying the reference group. If specified, for a given grouping variable, each of the group levels will be compared to the reference group (i.e. control group).
If ref.group = "all", pairwise two sample tests are performed for comparing each grouping variable levels against all (i.e. basemean).

p.adjust.method

- method to adjust p values for multiple comparisons. Used when pairwise comparisons are performed. Allowed values include "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". If you don’t want to adjust the p value (not recommended), use p.adjust.method = "none".

paired

- a logical indicating whether you want a paired test.

exact

- a logical indicating whether an exact p-value should be computed.

alternative

- a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter.

mu

- a number specifying an optional parameter used to form the null hypothesis.

conf.level

- confidence level of the interval.

detailed

- logical value. Default is FALSE. If TRUE, a detailed result is shown.

... other arguments to be passed to the function wilcox.test.

Details

- pairwise_wilcox_test() applies the standard two sample Wilcoxon test to all possible pairs of groups. This method calls the wilcox.test(), so extra arguments are accepted.

- If a list of comparisons is specified, the result of the pairwise tests is filtered to keep only the comparisons of interest. The p-value is adjusted after filtering.

- For a grouped data, if pairwise test is performed, then the p-values are adjusted for each group level independently.

- A nonparametric confidence interval and an estimator for the pseudomedian (one-sample case) or for the difference of the location parameters x-y is computed, where x and y are the compared samples or groups. The column estimate and the confidence intervals are displayed in the test result when the option detailed = TRUE is specified in the wilcox.test() and pairwise_wilcox_test() functions. Read more about the calculation of the estimate in the details section of the R base function wilcox.test() documentation by typing ?wilcox.test in the R console.

Value

return a data frame with some of the following columns:

- .y.: the y variable used in the test.
- group1, group2: the compared groups in the pairwise tests.
- n, n1, n2: Sample counts.
- statistic: Test statistic used to compute the p-value.
- p: p-value.
- p.adj: the adjusted p-value.
- method: the statistical test used to compare groups.
- p.signif, p.adj.signif: the significance level of p-values and adjusted p-values, respectively.
• estimate: an estimate of the location parameter (Only present if argument detailed = TRUE). This corresponds to the pseudomedian (for one-sample case) or to the difference of the location parameter (for two-samples case).
  – The pseudomedian of a distribution $F$ is the median of the distribution of $(u+v)/2$, where $u$ and $v$ are independent, each with distribution $F$. If $F$ is symmetric, then the pseudomedian and median coincide.
  – Note that in the two-sample case the estimator for the difference in location parameters does not estimate the difference in medians (a common misconception) but rather the median of the difference between a sample from $x$ and a sample from $y$.
• conf.low, conf.high: a confidence interval for the location parameter. (Only present if argument conf.int = TRUE.)

The returned object has an attribute called args, which is a list holding the test arguments.

Functions

• wilcox_test: Wilcoxon test
• pairwise_wilcox_test: performs pairwise two sample Wilcoxon test.

Examples

# Load data
data("ToothGrowth")
df <- ToothGrowth

# One-sample test
df %>% wilcox_test(len ~ 1, mu = 0)

# Two-samples unpaired test
df %>% wilcox_test(len ~ supp)

# Two-samples paired test
df %>% wilcox_test(len ~ supp, paired = TRUE)

# Compare supp levels after grouping the data by "dose"
df %>%
  group_by(dose) %>%
  wilcox_test(data = ., len ~ supp) %>%
  adjust_pvalue(method = "bonferroni") %>%
  add_significance("p.adj")

# pairwise comparisons
# As dose contains more than two levels ==> # pairwise test is automatically performed.
df %>% wilcox_test(len ~ dose)

# Comparison against reference group
#:---------------------------------
# each level is compared to the ref group
# df %>% wilcox_test(len ~ dose, ref.group = "0.5")

# Comparison against all
#:---------------------------------
# df %>% wilcox_test(len ~ dose, ref.group = "all")
Index

add_significance, 4
add_x_position (get_y_position), 64
add_xy_position (get_y_position), 64
add_y_position (get_y_position), 64
adjust_pvalue, 5
Anova, 5, 6, 8, 9, 49
anova_summary, 5, 10, 50
anova_test, 6, 7, 7, 50
aov, 5, 6, 8, 91
arrange, 36
as_cor_mat, 11, 33

binom.test, 13
binom_test, 12, 76
box_m, 14

chisq.test, 35
chisq_descriptives (chisq_test), 15
chisq_test, 15
cochran_qtest, 18
cohens_d, 19
convert_as_factor, 21
cor.test, 32
cor_as_symbols, 23, 27, 29
cor_gather, 24, 27, 30
cor_get_pval (cor_mat), 26
cor_mark_significant, 25
cor_mat, 11, 23–25, 26, 28, 30, 31, 33
cor_plot, 28
cor_pmat (cor_mat), 26
cor_reorder, 24, 27, 30
cor_select, 27, 31
cor_spread, 30
cor_spread (cor_gather), 24
cor_test, 11, 24, 27, 32
corplot, 28
counts_to_cases, 34
cramer_v, 35
create_test_label (get_pwc_label), 60
desc, 36
df_arrange, 36
df_get_var_names, 37
df_group_by, 37
df_label_both, 38, 41
df_label_value, 41
df_label_value (df_label_both), 38
df_nest_by, 39, 41
df_select, 40
df_split_by, 41
df_unite, 42
df_unite_factors (df_unite), 42
doo, 43
dunn_test, 44, 66
dunn_test, 44, 66
effsize (emmeans_test), 46
emaemeans_test, 46
eta_squared, 48
expected_freq (chisq_test), 15
factorial_design, 7, 10, 49
fisher.test, 50, 52
fisher_test, 50
freq_table, 53
friedman.test, 18, 55, 56
friedman_effsize, 54
friedman_test, 56
games_howell_test, 57
get_anova_table (anova_test), 7
get_comparisons, 58
get_emmeans (emmeans_test), 46
get_mode, 59
gam getPwc_label, 60
gam pwc_label, 60
gam summary_stats, 63
gam test_label (get_pwc_label), 60
gam y_position, 64
identify_outliers, 67
is_extreme (identify_outliers), 67
is_outlier (identify_outliers), 67
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>kruskal.test</td>
<td>70</td>
</tr>
<tr>
<td>kruskal_effsize</td>
<td>68</td>
</tr>
<tr>
<td>kruskal_test</td>
<td>70</td>
</tr>
<tr>
<td>levene_test</td>
<td>71</td>
</tr>
<tr>
<td>leveneTest</td>
<td>71</td>
</tr>
<tr>
<td>mahalanobis</td>
<td>72</td>
</tr>
<tr>
<td>mahalanobis_distance</td>
<td>72</td>
</tr>
<tr>
<td>make_clean_names</td>
<td>73</td>
</tr>
<tr>
<td>mcnemar.test</td>
<td>74</td>
</tr>
<tr>
<td>mcnemar_test</td>
<td>74</td>
</tr>
<tr>
<td>mshapiro_test (shapiro_test)</td>
<td>87</td>
</tr>
<tr>
<td>multinom.test</td>
<td>14,76</td>
</tr>
<tr>
<td>observed_freq (chisq_test)</td>
<td>15</td>
</tr>
<tr>
<td>oneway.test</td>
<td>95</td>
</tr>
<tr>
<td>p.adjust</td>
<td>5</td>
</tr>
<tr>
<td>p_adj_names (p_round)</td>
<td>83</td>
</tr>
<tr>
<td>p_detect (p_round)</td>
<td>83</td>
</tr>
<tr>
<td>p_format (p_round)</td>
<td>83</td>
</tr>
<tr>
<td>p_mark_significant (p_round)</td>
<td>83</td>
</tr>
<tr>
<td>p_names (p_round)</td>
<td>83</td>
</tr>
<tr>
<td>p_round</td>
<td>83</td>
</tr>
<tr>
<td>pairwise.prop.test</td>
<td>79</td>
</tr>
<tr>
<td>pairwise.t.test</td>
<td>94</td>
</tr>
<tr>
<td>pairwise.binom.test (binom_test)</td>
<td>12</td>
</tr>
<tr>
<td>pairwise.binom_test_against_p (binom_test)</td>
<td>12</td>
</tr>
<tr>
<td>pairwise_chisq_gof_test (chisq_test)</td>
<td>15</td>
</tr>
<tr>
<td>pairwise_chisq_test_against_p (chisq_test)</td>
<td>15</td>
</tr>
<tr>
<td>pairwise_fisher_test (fisher_test)</td>
<td>50</td>
</tr>
<tr>
<td>pairwise_mcnemar_test (mcnemar_test)</td>
<td>74</td>
</tr>
<tr>
<td>pairwise_prop.test (prop_test)</td>
<td>77</td>
</tr>
<tr>
<td>pairwise_sign_test (sign_test)</td>
<td>88</td>
</tr>
<tr>
<td>pairwise_t_test (t_test)</td>
<td>92</td>
</tr>
<tr>
<td>pairwise_wilcoxon.test (wilcoxon_test)</td>
<td>99</td>
</tr>
<tr>
<td>partial_eta_squared (eta_squared)</td>
<td>48</td>
</tr>
<tr>
<td>pearson_residuals (chisq_test)</td>
<td>15</td>
</tr>
<tr>
<td>plot.anova_test (anova_test)</td>
<td>7</td>
</tr>
<tr>
<td>print.anova_test (anova_test)</td>
<td>7</td>
</tr>
<tr>
<td>prop.test</td>
<td>77,79</td>
</tr>
<tr>
<td>prop.trend.test</td>
<td>80</td>
</tr>
<tr>
<td>prop.test</td>
<td>77</td>
</tr>
<tr>
<td>pull_lower_triangle (pull_triangle)</td>
<td>81</td>
</tr>
<tr>
<td>pull_triangle</td>
<td>27,31,81,86</td>
</tr>
<tr>
<td>pull_upper_triangle (pull_triangle)</td>
<td>81</td>
</tr>
<tr>
<td>reorder_levels (convert_as_factor)</td>
<td>21</td>
</tr>
<tr>
<td>replace_lower_triangle</td>
<td>85</td>
</tr>
<tr>
<td>replace_triangle</td>
<td>27,31,82,85</td>
</tr>
<tr>
<td>replace_upper_triangle</td>
<td>85</td>
</tr>
<tr>
<td>row_wise_fisher_test (fisher_test)</td>
<td>50</td>
</tr>
<tr>
<td>row_wise_prop_test (prop_test)</td>
<td>77</td>
</tr>
<tr>
<td>sample.n</td>
<td>86</td>
</tr>
<tr>
<td>sample_n_by</td>
<td>86</td>
</tr>
<tr>
<td>select</td>
<td>40</td>
</tr>
<tr>
<td>set_ref_level (convert_as_factor)</td>
<td>21</td>
</tr>
<tr>
<td>shapiro.test</td>
<td>87</td>
</tr>
<tr>
<td>shapiro_test</td>
<td>87</td>
</tr>
<tr>
<td>sign.test</td>
<td>66,88</td>
</tr>
<tr>
<td>std_residuals (chisq_test)</td>
<td>15</td>
</tr>
<tr>
<td>t.test</td>
<td>66,93</td>
</tr>
<tr>
<td>t_test</td>
<td>66,92</td>
</tr>
<tr>
<td>tukey_hsd</td>
<td>66,90</td>
</tr>
<tr>
<td>TukeyHSD</td>
<td>90,91</td>
</tr>
<tr>
<td>unite</td>
<td>42</td>
</tr>
<tr>
<td>welch_anova_test</td>
<td>95</td>
</tr>
<tr>
<td>wilcoxon.test</td>
<td>100</td>
</tr>
<tr>
<td>wilcoxon_effsize</td>
<td>96</td>
</tr>
<tr>
<td>wilcoxon_test</td>
<td>66,99</td>
</tr>
</tbody>
</table>