Package 'neatStats'

May 6, 2022

Title Neat and Painless Statistical Reporting

Version 1.11.0

Date 2022-05-06

Description User-friendly, clear and simple statistics, primarily for publication in psychological science. The main functions are wrappers for other packages, but there are various additions as well. Every relevant step from data aggregation to reportable printed statistics is covered for basic experimental designs.

URL https://github.com/gasparl/neatstats

Depends R (>= 3.6.0)

Imports viridis, fBasics, data.table, bayestestR, pROC, MBESS, car, ez, Exact, BayesFactor, ggplot2, ggpubr, logspline, grDevices, stats, graphics

Suggests rmarkdown, knitr, rstudioapi

License BSD_2_clause + file LICENSE

Encoding UTF-8

LazyData false

RoxygenNote 7.1.2

VignetteBuilder knitr

R topics documented:

`aggr_neat` *Aggregation, descriptives*

Description

Returns aggregated values per group for given variable. Serves as argument in the `table_neat` function.

Usage

```
aggr_neat(
  dat,
  values,
  method = mean,
  group_by = NULL,
  filt = NULL,
  sep = "_",
  prefix = NULL,
  new_name = NULL,
  round_to = 2
)
```

Arguments

`dat`	A data frame (or a `data.table`, or the name of either as string).
`values`	The vector of numbers from which the statistics are to be calculated, or the name of the column in the `dat` data frame, that contains the vector.

`method` Function of string. If function, uses the `values` to calculate the returned value for the given function (e.g. means, as per default, using the `mean` function). Such a function may return a vector of results as well; see Examples. If string, one of two internal functions will be used. If the string end with `"+sd"`, e.g., `"mean+sd"`, the function preceding the `"+"` sign will be calculated along with the standard deviation, displayed in a single column, rounded as set in the `round_to` argument. (This is primarily for use in the `table_neat` function for summary tables.) If the string does not end with `"+sd"`, a ratio for the occurrences of given elements will be calculated. Multiple elements can by given as a vector of strings. The number of occurrences of these elements will be the numerator (dividend), while the entire column length (i.e., number of all elements) will be the denominator (divisor). For example, if a column contains elements `"correct"`, `"incorrent"`, `"tooslow"`, the ratio of `"correct"` to all other elements (i.e., including elements `"correct"`, `"incorrent"`, and `"tooslow"`) can be written simply as `method = "correct"`. The complementary ratio, of `"incorrent"` and `"tooslow"`, can be written as `method = "incorrent, tooslow"`. (Hint: filter to get ratios of subgroups, e.g. to include only `"correct"` and `"incorrent"` elements, and calculate their ratio; see below.)

`group_by` String, or vector of strings: the name(s) of the column(s) in the `dat` data frame, containing the vector(s) of factors by which the statistics are grouped.

`filt` An expression to filter, by column values, the entire `dat` data frame before performing the aggregation. The expression should use column names alone; see Examples.

`sep` String (underscore `"_"` by default) for separating group names (and prefix, if given).

`prefix` `NULL` (default) or string. String specifies a prefix for each group type under the `group` column.

`new_name` `NULL` (default) or string. String specifies new name for the variable to be used as column title. If `NULL`, the name will be `"aggr_value"` (or, if used with `table_neat`, the input variable name is used).

`round_to` Number of digits after the decimal point to round to, when using `"+sd"` in `method`.

Value

A `data.table` with the statistics per group, with a single column (`"aggr_group"`) indicating the grouping.

See Also

`table_neat` to create full tables using multiple variables

Examples

```
data("mtcars") # load base R example dataset

# overall means and SDs for wt (Weight)
aggr_neat(mtcars, wt)
```

```
# rename column
aggr_neat(mtcars, wt, new_name = 'weight')

# grouped by cyl (Number of cylinders)
aggr_neat(mtcars, wt, group_by = 'cyl')

# grouped by cyl and gear
aggr_neat(mtcars, wt, group_by = c('cyl', 'gear'))

# prefix for group names
aggr_neat(mtcars, wt, group_by = 'cyl', prefix = 'cyl')

# filter to only have cyl larger than  4
aggr_neat(mtcars, wt, group_by = 'cyl', filt = cyl > 4)

# filter to only have hp (Gross horsepower) smaller than  200
aggr_neat(mtcars, wt, group_by = 'cyl', filt = hp < 200)

# combine two filters above, and add prefix
aggr_neat(
    mtcars,
    wt,
    group_by = 'cyl',
    filt = (hp < 200 & cyl > 4),
    prefix = 'filtered'
)

# add SD (and round output numbers to 2)
aggr_neat(mtcars,
          wt,
          group_by = 'cyl',
          method = 'mean+sd',
          round_to = 2)

# now medians instead of means
aggr_neat(mtcars, wt, group_by = 'cyl', method = median)

# with SD
aggr_neat(mtcars,
          wt,
          group_by = 'cyl',
          method = 'median+sd',
          round_to = 1)

# overall ratio of gear 4 (Number of gears)
aggr_neat(mtcars, gear, method = '4')

# overall ratio of gear 4 and 5
aggr_neat(mtcars, gear, method = '4, 5')

# same ratio calculated per each cyl
aggr_neat(mtcars, gear, group_by = 'cyl', method = '4, 5')
```

```
# per each cyl and per vs (engine type)
aggr_neat(mtcars,
          gear,
          group_by = c('cyl', 'vs'),
          method = '4, 5')

# ratio of gear 3 per gear 3 and 5
aggr_neat(
    mtcars,
    gear,
    group_by = 'cyl',
    method = '3',
    filt = gear %in% c(3, 5)
)
```

---

anova_neat *Comparison of Multiple Means: ANOVA*

Description

Analysis of variance (ANOVA) F-test results with appropriate Welch's and epsilon corrections where applicable (unless specified otherwise), including partial eta squared effect sizes with confidence intervals (CIs), generalized eta squared, and inclusion Bayes factor based on matched models (BFs).

Usage

```
anova_neat(
  data_per_subject,
  values,
  within_ids = NULL,
  between_vars = NULL,
  ci = 0.9,
  norm_tests = "none",
  norm_plots = FALSE,
  var_tests = FALSE,
  bf_added = FALSE,
  bf_sample = 10000,
  test_title = "--- neat ANOVA ---",
  welch = TRUE,
  e_correction = NULL,
  type = 2,
  white.adjust = FALSE,
  hush = FALSE,
  plot_means = FALSE,
  ...
)
```

Arguments

`data_per_subject`
: Data frame. Should contain all values (measurements/observations) in a single row per each subject.

`values`
: Vector of strings; column name(s) in the `data_per_subject` data frame. Each column should contain a single dependent variable: thus, to test repeated (within-subject) measurements, each specified column should contain one measurement.

`within_ids`
: `NULL` (default), string, or named list. In case of no within-subject factors, leave as `NULL`. In case of a single within subject factor, a single string may be given to optionally provide custom name for the within-subject factor (note: this is a programming variable name, so it should not contain spaces, etc.); otherwise (if left `NULL`) this one within-subject factor will always just be named `"within_factor"`. In case of multiple within-subject factors, each factor must be specified as a named list element, each with a vector of strings that distinguish the levels within that factors. The column names given as `values` should always contain one (and only one) of these strings within each within-subject factor, and thus they will be assigned the appropriate level. For example, `values = 'rt_s1_neg, rt_s1_pos,rt_s2_neg, rt_s2_pos'` could have `within_ids = list( session = c('s1','s2'), valence = c('pos', 'neg')`. (Note: the strings for distinguishing must be unambiguous. E.g., for values `apple_a` and `apple_b`, do not set levels `c('a','b')`, because `'a'` is also found in `apple_b`. In this case, you could choose levels `c('_a','_b')` to make sure the values are correctly distinguished.) See also Examples.

`between_vars`
: `NULL` (default; in case of no between-subject factors) or vector of strings; column name(s) in the `data_per_subject` data frame. Each column should contain a single between-subject independent variable (representing between-subject factors).

`ci`
: Numeric; confidence level for returned CIs. (Default: `.9`; Lakens, 2014; Steiger, 2004.)

`norm_tests`
: Normality tests for the pooled ANOVA residuals (`"none"` by default, giving no tests). Any or all of the following character input is accepted (as a single string or a character vector; case-insensitive): `"W"` (Shapiro-Wilk), `"K2"` (D'Agostino), `"A2"` (Anderson-Darling), `"JB"` (Jarque-Bera); see `norm_tests`. The option `"all"` (or `TRUE`) selects all four previous tests at the same time.

`norm_plots`
: If `TRUE`, displays density, histogram, and Q-Q plots (and scatter plots for paired tests) for the pooled residuals.

`var_tests`
: Logical, `FALSE` by default. If `TRUE` (and there are any between-subject factors), runs variance equality tests via `var_tests` for all combinations of the between-subject factors within each level of within-subject factor combinations; see Details.

`bf_added`
: Logical. If `TRUE` (default), inclusion Bayes factor is calculated and displayed. (Note: with multiple factors and/or larger dataset, the calculation can take considerable time.)

`bf_sample`
: Number of samples used to estimate Bayes factor (`10000` by default).

`test_title`
: String, `"--- neat ANOVA ---"` by default. Simply displayed in printing preceding the statistics.

welch	If TRUE (default), calculates Welch's ANOVA via stats::oneway.test in case of a single factor (one-way) between-subject design. If FALSE, calculates via ez::ezANOVA in such cases too (i.e., same as in case of every other design).
e_correction	NULL (default) or one of the following strings: 'gg', 'hf', or 'none'. If set to 'gg', Greenhouse-Geisser correction is applied in case of repeated measures (regardless of violation of sphericity). If set to 'hf', Huynh-Feldt correction is applied. If set to 'none', no correction is applied. If NULL, Greenhouse-Geisser correction is applied when Mauchly's sphericity test is significant and the Greenhouse-Geisser epsilon is not larger than .75, while Huynh-Feldt correction is applied when Mauchly's sphericity test is significant and the Greenhouse-Geisser epsilon is larger than .75 (see Girden, 1992).
type	Sum of squares type specified as a number: 1, 2, or 3. Set to 2 by default (which is generally recommended, see e.g. Navarro, 2019, Chapter 16).
white.adjust	If not FALSE (default) uses a heteroscedasticity-corrected coefficient covariance matrix; the various values of the argument specify different corrections ("hc0", "hc1", "hc2", "hc3", or "hc4"). See the documentation for car::hccm for details. If set to TRUE then the "hc3" correction is selected.
hush	Logical. If TRUE, prevents printing any details to console.
plot_means	Logical (FALSE by default). If TRUE, creates plots of means by factor, by passing data and factor information to plot_neat.
...	Any additional arguments will be passed on to plot_neat.

Details

The Bayes factor (BF) is always calculated with the default rscaleFixed of 0.5 ("medium") and rscaleRandom of 1 ("nuisance"). BF supporting null hypothesis is denoted as BF01, while that supporting alternative hypothesis is denoted as BF10. When the BF is smaller than 1 (i.e., supports null hypothesis), the reciprocal is calculated (hence, BF10 = BF, but BF01 = 1/BF). When the BF is greater than or equal to 10000, scientific (exponential) form is reported for readability. (The original full BF number is available in the returned named vector as bf.)

Mauchly's sphericity test is returned for repeated measures with more than two levels. If Mauchly's test is significant, epsilon correction may be applied (see the e_correction parameter).

Variance equality tests (if var_tests is TRUE): Brown-Forsythe and Fligner-Killeen tests are performed for each within-subject factor level (or, in case of several factors, each combination) via var_tests. Note that variance testing is generally not recommended (see e.g., Zimmerman, 2004). In case of a one-way between-subject ANOVA, Welch-corrected results are reported by default, which corrects for unequal variances. In case of multiple between-subject factors, the white.adjust parameter can be set to TRUE (or "hc3") to apply "hc3" correction for unequal variances. In case of a mixed ANOVA (both between-subject and within-subject factors), if the tests are significant and the data is unbalanced (unequal group sample sizes), you should either consider the results respectively or choose a different test.

In case of multiple values (column names) that match identical levels for all factors, the given columns will be merged into a single column taking the mean value of all these columns. (This is to simplify "dropping" a within-subject factor and retesting the remaining factors.) Explicit warning messages are shown in each such case.

Value

Prints ANOVA statistics (including, for each model, F-test with partial eta squared and its CI, generalized eta squared, and BF, as specified via the corresponding parameters) in APA style. Furthermore, when assigned, returns a list with up to four elements. First, 'stat_list', a list of named vectors per each effect (main or interaction). Each vector contains the following elements: F (F value), p (p value), petas (partial eta squared), getas (generalized eta squared), epsilon (epsilon used for correction), and bf (inclusion BF; when bf_added is not FALSE). Second, a dataframe with mean and SD per each condition (i.e., per each level per each factor). Third, the ezANOVA object, named ez_anova (calculated even when oneway.test is printed). Fourth, when bf_added is not FALSE, the anovaBF object, named bf_models; including all models on which the inclusion BFs are based.

Note

The F-tests are calculated via ez::ezANOVA, including Mauchly's sphericity test. (But Welch's ANOVA is calculated in case of one-way between-subject designs via stats::oneway.test, unless the welch parameter is set to FALSE.)

Confidence intervals are calculated, using the F value, via MBESS::conf.limits.ncf, converting noncentrality parameters to partial eta squared as ncp/(ncp+ df_nom+df_denom+1) (Smithson, 2003).

Generalized eta squared is to facilitate potential subsequent meta-analytical comparisons (see Lakens, 2013).

The inclusion Bayes factor based on matched models is calculated via bayestestR::bayesfactor_inclusion, (with match_models = TRUE, and using an BayesFactor::anovaBF object for models input).

References

Girden, E. (1992). ANOVA: Repeated measures. Newbury Park, CA: Sage.

Kelley, K. (2007). Methods for the behavioral, educational, and social sciences: An R package. Behavior Research Methods, 39(4), 979-984. doi:10.3758/BF03192993

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

Lakens, D. (2014). Calculating confidence intervals for Cohen's d and eta-squared using SPSS, R, and Stata [Blog post]. Retrieved from http://daniellakens.blogspot.com/2014/06/calculating-confidence-inter
html

Mathot. S. (2017). Bayes like a Baws: Interpreting Bayesian Repeated Measures in JASP [Blog post]. Retrieved from https://www.cogsci.nl/blog/interpreting-bayesian-repeated-measures-in-jasp

McDonald, J. H. 2015. Handbook of Biological Statistics (3rd ed.). Sparky House Publishing, Baltimore, Maryland. Retrieved from http://www.biostathandbook.com

Moder, K. (2010). Alternatives to F-test in one way ANOVA in case of heterogeneity of variances (a simulation study). Psychological Test and Assessment Modeling, 52(4), 343-353.

Navarro, D. (2019). Learning Statistics with R: A Tutorial for Psychology Students and Other Beginners (Version 0.6.1). Retrieved from https://learningstatisticswithr.com/

Smithson, M. (2003). Confidence intervals. Thousand Oaks, Calif: Sage Publications.

Steiger, J. H. (2004). Beyond the F test: effect size confidence intervals and tests of close fit in the analysis of variance and contrast analysis. Psychological Methods, 9(2), 164-182. doi:10.1037/1082989X.9.2.164

Zimmerman, D. W. (2004). A note on preliminary tests of equality of variances. British Journal of Mathematical and Statistical Psychology, 57(1), 173–181. https://doi.org/10.1348/000711004849222

See Also

plot_neat, t_neat

Examples

```
# assign random data in a data frame for illustration
# (note that the 'subject' is only for illustration; since each row contains the
# data of a single subject, no additional subject id is needed)
dat_1 = data.frame(
    subject = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
    grouping1 = c(1, 1, 1, 1, 2, 2, 2, 2, 2, 2),
    grouping2 = c(1, 2, 1, 2, 2, 1, 1, 1, 2, 1),
    value_1_a = c(36.2, 45.2, 41, 24.6, 30.5, 28.2, 40.9, 45.1, 31, 16.9),
    value_2_a = c(-14.1, 58.5, -25.5, 42.2, -13, 4.4, 55.5, -28.5, 25.6, -37.1),
    value_1_b = c(83, 71, 111, 70, 92, 75, 110, 111, 110, 85),
    value_2_b = c(8.024, -14.162, 3.1, -2.1, -1.5, 0.91, 11.53, 18.37, 0.3, -0.59),
    value_1_c = c(27.4,-17.6,-32.7, 0.4, 37.2, 1.7, 18.2, 8.9, 1.9, 0.4),
    value_2_c = c(7.7, -0.8, 2.2, 14.1, 22.1, -47.7, -4.8, 8.6, 6.2, 18.2)
)
head(dat_1) # see what we have

# For example, numbers '1' and '2' in the variable names of the values can
# denote sessions in an experiment, such as '_1' for first session, and '_2 for
# second session'. The letters '_a', '_b', '_c' could denote three different
# types of techniques used within each session, to be compared to each other.
# See further below for a more verbose but more meaningful example data.

# get the between-subject effect of 'grouping1'
anova_neat(dat_1, values = 'value_1_a', between_vars = 'grouping1')

# main effects of 'grouping1', 'grouping2', and their interactions
anova_neat(dat_1,
           values = 'value_1_a',
           between_vars = c('grouping1', 'grouping2'))

# repeated measures:
# get the within-subject effect for 'value_1_a' vs. 'value_1_b'
anova_neat(dat_1, values = c('value_1_a', 'value_1_b'))

# same, but give the factor a custom variable name, and omit BF for speed
anova_neat(
    dat_1,
    values = c('value_1_a', 'value_1_b'),
    within_ids = 'a_vs_b',
    bf_added = FALSE
```

```
)
# or
anova_neat(
    dat_1,
    values = c('value_1_a', 'value_1_b'),
    within_ids = 'letters',
    bf_added = FALSE
)

# within-subject effect for 'value_1_a' vs. 'value_1_b' vs. 'value_1_c'
anova_neat(
    dat_1,
    values = c('value_1_a', 'value_1_b', 'value_1_c'),
    bf_added = FALSE
)

# within-subject main effect for 'value_1_a' vs. 'value_1_b' vs. 'value_1_c',
# between-subject main effect 'grouping1', and the interaction of these two main
# effects
anova_neat(
    dat_1,
    values = c('value_1_a', 'value_1_b', 'value_1_c'),
    between_vars = 'grouping1',
    bf_added = FALSE
)

# within-subject 'number' main effect for variables with number '1' vs. number
# '2' ('value_1_a' and 'value_1_b' vs. 'value_2_a' and 'value_2_b'), 'letter'
# main effect for variables with final letterr 'a' vs. final letter 'b'
# ('value_1_a' and 'value_2_a' vs. 'value_1_b' and 'value_2_b'), and the
# 'letter' x 'number' interaction
anova_neat(
    dat_1,
    values = c('value_1_a', 'value_2_a', 'value_1_b', 'value_2_b'),
    within_ids = list(
        letters = c('_a', '_b'),
        numbers =  c('_1', '_2')
    ),
    bf_added = FALSE
)

# same as above, but now including between-subject main effect 'grouping2' and
# its interactions
anova_neat(
    dat_1,
    values = c('value_1_a', 'value_2_a', 'value_1_b', 'value_2_b'),
    within_ids = list(
        letters = c('_a', '_b'),
        numbers =  c('_1', '_2')
    ),
    between_vars = 'grouping2',
    bf_added = FALSE
)
```

```
# same as above, but now creating a plot of means
# y_title passed add an example title (label) for the Y axis
anova_neat(
    dat_1,
    values = c('value_1_a', 'value_2_a', 'value_1_b', 'value_2_b'),
    within_ids = list(
        letters = c('_a', '_b'),
        numbers =  c('_1', '_2')
    ),
    between_vars = 'grouping2',
    bf_added = FALSE,
    plot_means = TRUE,
    y_title = 'Example Y Title'
)

# same as above, but collapsing means over the removed "numbers" factor
anova_neat(
    dat_1,
    values = c('value_1_a', 'value_2_a', 'value_1_b', 'value_2_b'),
    within_ids = list(
        letters = c('_a', '_b')
    ),
    between_vars = 'grouping2',
    bf_added = FALSE,
    plot_means = TRUE,
    y_title = 'Example Y Title'
)

# In real datasets, these could of course be more meaningful. For example, let's
# say participants rated the attractiveness of pictures with low or high levels
# of frightening and low or high levels of disgusting qualities. So there are
# four types of ratings:
# 'low disgusting, low frightening' pictures
# 'low disgusting, high frightening' pictures
# 'high disgusting, low frightening' pictures
# 'high disgusting, high frightening' pictures

# this could be meaningfully assigned e.g. as below
pic_ratings = data.frame(
    subject = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
    rating_fright_low_disgust_low = c(36.2,45.2,41,24.6,30.5,28.2,40.9,45.1,31,16.9),
   rating_fright_high_disgust_low = c(-14.1,58.5,-25.5,42.2,-13,4.4,55.5,-28.5,25.6,-37.1),
    rating_fright_low_disgust_high = c(83,71,111,70,92,75,110,111,110,85),
   rating_fright_high_disgust_high = c(8.024,-14.162,3.1,-2.1,-1.5,0.91,11.53,18.37,0.3,-0.59)
)
head(pic_ratings) # see what we have

# the same logic applies as for the examples above, but now the
# within-subject differences can be more meaningfully specified, e.g.
# 'disgust_low' vs. 'disgust_high' for levels of disgustingness, while
# 'fright_low' vs. 'fright_high' for levels of frighteningness
anova_neat(
```

```
    pic_ratings,
    values = c(
        'rating_fright_low_disgust_low',
        'rating_fright_high_disgust_low',
        'rating_fright_low_disgust_high',
        'rating_fright_high_disgust_high'
    ),
    within_ids = list(
        disgustingness = c('disgust_low', 'disgust_high'),
        frighteningness =  c('fright_low', 'fright_high')
    ),
    bf_added = FALSE
)

# the results are the same as for the analogous test for the 'dat_1' data, only
# with different names

# now let's say the ratings were done in two separate groups
pic_ratings = data.frame(
    subject = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
    group_id = c(1, 2, 1, 2, 2, 1, 1, 1, 2, 1),
    rating_fright_low_disgust_low = c(36.2,45.2,41,24.6,30.5,28.2,40.9,45.1,31,16.9),
   rating_fright_high_disgust_low = c(-14.1,58.5,-25.5,42.2,-13,4.4,55.5,-28.5,25.6,-37.1),
    rating_fright_low_disgust_high = c(83,71,111,70,92,75,110,111,110,85),
   rating_fright_high_disgust_high = c(8.024,-14.162,3.1,-2.1,-1.5,0.91,11.53,18.37,0.3,-0.59)
)

# now test the effect and interactions of 'group_id'
anova_neat(
    pic_ratings,
    values = c(
        'rating_fright_low_disgust_low',
        'rating_fright_high_disgust_low',
        'rating_fright_low_disgust_high',
        'rating_fright_high_disgust_high'
    ),
    within_ids = list(
        disgustingness = c('disgust_low', 'disgust_high'),
        frighteningness =  c('fright_low', 'fright_high')
    ),
    between_vars = 'group_id',
    bf_added = FALSE
)

# again, same results as with 'dat_1' (using 'grouping2' as group_id)
```

---

ci_from_p *CI from p value*

Description

Calculates approximate confidence interval (CI) for any given difference, based on the difference value and p value, according to Altman & Bland (2011).

Usage

```
ci_from_p(diff, p_value, ci = 0.95)
```

Arguments

`diff`	Difference (number) around which the CI is to be calculated.
`p_value`	The p value for CI calculation.
`ci`	Numeric; confidence level for the returned CIs (`.95` per default).

Value

CI limits as named numeric vector with two elements.

Note

Check the Altman & Bland (2011) paper for details! (The calculation for proportions is not implemented here.)

References

Altman, D. G., & Bland, J. M. (2011). How to obtain the confidence interval from a P value. Bmj, 343(d2090). doi:10.1136/bmj.d2090

Examples

```
# Example 1
# calculate proportion difference test
proptest_stat = prop.test(x = c(49, 40), n = c(50, 50))

# calculate proportion difference
my_diff = 49/50-40/50

# calculate approximate CI
ci_from_p(my_diff, proptest_stat$p.value)

# returned CI should be very similar to the actual CI
proptest_stat$conf.int


# Example 2
# generate random data
v1 = stats::rnorm(190, 40, 60)
v2 = stats::rnorm(170, 50, 45)

# calculate t-test
```

```
ttest_stat = stats::t.test(v1, v2)

# calculate mean difference
my_diff = mean(v1) - mean(v2)

# calculate approximate CI
ci_from_p(my_diff, ttest_stat$p.value)

# returned CI should be similar to the actual CI
ttest_stat$conf.int
```

`corr_neat` *Correlation Statistics*

Description

Pearson correlation results including confidence interval (CI) and correlation Bayes factor (BF). For non-parametric version, Spearman's rank correlation results along with corresponding rank-based BFs (as per van Doorn et al., 2020).

Usage

```
corr_neat(
  var1,
  var2,
  nonparametric = FALSE,
  ci = 0.95,
  bf_added = FALSE,
  direction = NULL,
  round_r = 3,
  for_table = FALSE,
  sb_correction = FALSE,
  hush = FALSE
)
```

Arguments

`var1`	Numeric vector; numbers of the first variable.
`var2`	Numeric vector; numbers of the second variable.
`nonparametric`	Logical (`FALSE` by default). If `TRUE`, uses nonparametric tests (Spearman's rank correlation, including BFs; see Details).
`ci`	Numeric; confidence level for the returned CI, as implemented in `cor.test`.
`bf_added`	Logical. If `TRUE` (default), Bayes factor is calculated and displayed.
`direction`	`NULL` or string; optionally specifies one-sided test: either "negative" (negative correlation expected) or "positive" (positive correlation expected). (Short forms also work, e.g. "p", "pos", "neg", etc.) If `NULL` (default), the test is two-sided.

round_r	Number to round to the correlation and its CI.
for_table	Logical. If TRUE, omits the confidence level display from the printed text.
sb_correction	Logical. If TRUE, applies Spearman-Brown correction (2 * r / (1+r)) to the correlation (including CI).
hush	Logical. If TRUE, prevents printing any details to console.

Details

The Bayes factor (BF) is calculated with the default r-scale of 1/3 for parametric test, and with the default r-scale of 1 for nonparametric test. BF supporting null hypothesis is denoted as BF01, while that supporting alternative hypothesis is denoted as BF10. When the BF is smaller than 1 (i.e., supports null hypothesis), the reciprocal is calculated (hence, BF10 = BF, but BF01 = 1/BF). When the BF is greater than or equal to 10000, scientific (exponential) form is reported for readability. (The original full BF number is available in the returned named vector as bf.)#'

Value

Prints correlation statistics (including CI and BF) in APA style. Furthermore, when assigned, returns a named vector with the following elements: r (Pearson correlation), p (p value), bf (Bayes factor).

Note

The correlation and CI is calculated via stats::cor.test.

The parametric Bayes factor is calculated via BayesFactor::correlationBF. The nonparametric (rank-based) Bayes factor is a contribution by Johnny van Doorn; the original source code is available via https://osf.io/gny35/.

References

Brown, W. (1910). Some experimental results in the correlation of mental abilities. British Journal of Psychology, 1904-1920, 3(3), 296-322. doi:10.1111/j.20448295.1910.tb00207.x

Eisinga, R., Grotenhuis, M. te, & Pelzer, B. (2013). The reliability of a two-item scale: Pearson, Cronbach, or Spearman-Brown? International Journal of Public Health, 58(4), 637-642. doi:10.1007/s0003801204163

Spearman, C. (1910). Correlation calculated from faulty data. British Journal of Psychology, 1904-1920, 3(3), 271-295. doi:10.1111/j.20448295.1910.tb00206.x

van Doorn, J., Ly, A., Marsman, M., & Wagenmakers, E.-J. (2020). Bayesian rank-based hypothesis testing for the rank sum test, the signed rank test, and Spearman's rho. Journal of Applied Statistics, 1–23. doi:10.1080/02664763.2019.1709053

See Also

t_neat

Examples

```
# assign two variables
v1 = c(11, 15, 19, 43, 53, -4, 34, 8, 33, -1, 54 )
v2 = c(4, -2, 23, 13, 32, 16, 3, 29, 37, -4, 65 )

corr_neat(v1, v2) # prints statistics

# one-sided, and omitting the "95% CI" part
corr_neat(v1, v2, direction = 'pos', for_table = TRUE)

# print statistics and assign main results
results = corr_neat(v1, v2, direction = 'pos')

results['p'] # get precise p value
```

---

dems_neat *Demographics*

Description

Prints participant count, age mean and SD, and gender ratio, from given dataset.

Usage

```
dems_neat(
  data_per_subject,
  group_by = NULL,
  gender_col = NULL,
  age_col = NULL,
  male = "male",
  female = "female",
  percent = FALSE,
  round_perc = 0,
  show_fem = NULL,
  age_range = FALSE,
  age_min = NULL,
  age_max = NULL
)
```

Arguments

`data_per_subject`

Data frame from which demographics are to be calculated. Should contain columns named as "age" and as "gender" (or, alternatively, "sex"). Alternatively, these columns can be specified via the `gender_col` and `age_col` parameters. The `age` column must contain only numbers or `NA`, while `gender` column must contain only `1` (= male) or `2` (= female), either as numbers or as strings, or `NA`. Alternatively, different gender coding can be set via the parameters `male` and `female` (but `1`/`2` will be checked for first in any case).

group_by	Optionally the name(s) of column(s) from the data frame provided as data_per_subject to group by.
gender_col	Optionally the name of column from the data frame that contains the gender (sex) information.
age_col	Optionally the name of column from the data frame that contains the age information.
male	Alternative code for male: by default, it is the string "male". Whatever string is given, its abbreviations will also be accepted (e.g. "m"). (Lettercases do not matter, e.g. Male or MALE are both evaluated same as male.)
female	Alternative code for female: by default, it is the string "female". Whatever string is given, its abbreviations will also be accepted (e.g. "fem"). (Lettercases do not matter.)
percent	Logical. If TRUE, gender ratios (and the "unknown" ratios based on NA values) are presented as percentage. If FALSE, they are presented as counts (i.e., numbers of subjects).
round_perc	Number to round to, when using percentages.
show_fem	Logical or NULL. If TRUE, the numbers of both male and female are displayed. If FALSE, only the number of males is displayed. If NULL (default), only the number of males is displayed when there are no unknown cases, but both numbers are displayed when there are any unknown cases.
age_range	Logical, FALSE by default. If TRUE, also displays age range per group (minimum and maximum ages).
age_min	If numeric given, removes all ages below (exclusive!) the given number before any age calculation.#'
age_max	If numeric given, removes all ages above (exclusive!) the given number before any age calculation.

Details

If gender_col and/or age_col are not specified, the function will first look for columns named precisely "age" and as "gender". If either is not found, the function looks for the same names but with any lettercase (e.g. "AGE" or "Gender"). If still no "gender" column is found, the function looks for "sex" column in the same manner. If no column is found for either, all related values will be counted as "unknown" (NA).

If NA values are found in either the age or gender column, the ratio (or count) of unknown cases will be displayed everywhere. Otherwise it will simply not be displayed anywhere.

Examples

```
# below is an illustrative example dataset
# (the "subject" and "measure_x" columns are not used in the function)
dat = data.frame(
    subject = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
    conditions = c('x', 'y', 'x', 'y', 'y', 'x', 'x', 'x', 'y', 'x'),
    gender = c(2, 2, 1, 2, 1, 2, 2, 2, 1, 1),
    age = c(6, 7, 8.5, 6, 5, 16.5, 17, 16, 45.8, 77),
```

```
    measure_x = c(83, 71, 111, 70, 92, 75, 110, 111, 110, 85),
    stringsAsFactors = TRUE
)

# print demographics (age and gender) per "conditions":
dems_neat(dat, group_by = 'conditions')

# replace unlikely ages with NAs
dems_neat(dat,
          group_by = 'conditions',
          age_min = 8,
          age_max = 50)

# remove only high values, and display age ranges
dems_neat(dat,
          group_by = 'conditions',
          age_max = 45,
          age_range = TRUE)

# another dataset, with some missing values
dat = data.frame(
    subject = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
    conditions = c('x', 'y', 'x', 'y', 'y', 'x', 'x', 'x', 'y', 'x'),
    gender = c(2, 2, NA, NA, 1, 1, 1, 2, NA, NA),
    age = c(6, 7, 8.5, 6, 5, 16, NA, 16, 45, 77),
    measure_x = c(83, 71, 111, 70, 92, 75, 110, 111, 110, 85),
    stringsAsFactors = TRUE
)
# again print demographics per "conditions":
dems_neat(dat, group_by = 'conditions')

# another dataset, with no "age"/"gender" columns
dat = data.frame(
    subject = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
    conditions = c('x', 'y', 'x', 'y', 'y', 'x', 'x', 'x', 'y', 'x'),
    geschlecht = c(2, 2, NA, NA, 1, 1, 1, 2, NA, NA),
    alter = c(6, 7, 8.5, 6, 5, 16, NA, 16, 45, 77),
    measure_y = c(83, 71, 111, 70, 92, 75, 110, 111, 110, 85),
    stringsAsFactors = TRUE
)

# the following will return "unknowns"
dems_neat(dat, group_by = 'conditions')

# gender column specified
dems_neat(dat, group_by = 'conditions', gender_col = 'geschlecht')

# both columns specified
dems_neat(dat,
          group_by = 'conditions',
          age_col = 'alter',
          gender_col = 'geschlecht')
```

enum *Enumerate*

Description

Aids enumeration and merging (via rbind_loop) in loops: adds numbers to a vector input, and indicates loop start for rbind_loop.

Usage

```
enum(items, hush = FALSE, enumerate = TRUE)
```

Arguments

items	The items to be enumerated in the loop.
hush	Logical. If TRUE (default), prints "Loop started." when executed.
enumerate	Logical. If TRUE (default), adds numbering to the input vector (in item pairs, see Examples). If FALSE, returns the original input.

Value

Vector with numbers added (if so set).

See Also

rbind_loop

Examples

```
my_vector = c('aa', 'bb', 'cxyz', 'last')

for (item in enum(my_vector)) {
    print(item)
}

# just to show what enum() returns
enum(my_vector)
```

`excl_neat` *Exclusion*

Description

Filters dataset by rows (normally: subjects, observations) and prints the numbers of excluded rows and remaining rows. Returns the filtered dataset and (optionally) also the excluded rows.

Usage

```
excl_neat(
  dat,
  filt,
  excluded = FALSE,
  group_by = NULL,
  sort_by = "exclusion",
  hush = FALSE
)
```

Arguments

`dat`	Data frame to be filtered.
`filt`	An expression to use for filtering, by column values, the `dat` data frame. (Only the rows for which the filter expression is `TRUE` will be kept.)
`excluded`	Logical; `FALSE` by default. If `TRUE`, the function returns not only the filtered data frame, but also a data frame containing the excluded rows. The returned object in this case will be a list with two elements: (1) the filtered data frame named `filtered`, and (2) the data frame with excluded rows named `excluded` (see Examples).
`group_by`	String, or vector of strings: the name(s) of the column(s) in the `dat` data frame, containing the vector(s) of factors by which the printed counts are grouped.
`sort_by`	String; specifies whether the printed counts should be sorted by exclusion (default; `"exclusion"` or its short forms, e.g. `"excl"`), or by the factors given for `group_by` (for this, give any other string, e.g. `"conditions"`). If `NULL` (default).
`hush`	Logical. If `TRUE`, prevents printing counts to console.

Value

A data frame with the rows for which the `filt` expression is `TRUE`, or, optionally, a list with this data frame plus a data frame with the excluded rows. At the same time, prints, by default, the count of remaining and excluded rows.

See Also

`aggr_neat`

Examples

```
data("mtcars") # load base R example dataset

# filter mtcars for mpg > 20
excl_neat(mtcars, mpg > 20)

# assign the same
mtcars_filtered = excl_neat(mtcars, mpg > 20)
# (mtcars_filtered now contains the filtered subset)

# return and assign excluded rows too
mtcars_filtered_plus_excluded = excl_neat(mtcars, mpg > 20, excluded = TRUE)

# print filtered data frame
print(mtcars_filtered_plus_excluded$filtered)

# print data frame with excluded rows
print(mtcars_filtered_plus_excluded$excluded)

# group printed count by cyl
excl_neat(mtcars, mpg > 20, group_by = 'cyl')

# sort output by grouping
excl_neat(mtcars, mpg > 20, group_by = 'cyl', sort_by = 'group')

# group by cyl amd carb
excl_neat(mtcars, mpg > 15, group_by = c('cyl', 'carb'))

# longer filter expression
excl_neat(mtcars, mpg > 15 & gear == 4, group_by = 'cyl',)
```

---

mean_ci *Confidence Interval of Mean*

Description

Calculates confidence interval of a vector of numbers.

Usage

```
mean_ci(x, distance_only = TRUE, ci = 0.95)
```

Arguments

x	Numeric vector.
distance_only	Logical. If TRUE (default), the function returns only the distance between the mean and either confidence interval limit. Otherwise returns the confidence interval (i.e., both limits).
ci	Numeric; confidence level for returned CI.

Value

Distance of limit or confidence interval (as named vector).

See Also

se, plot_neat, sd_ci

Examples

```
myvec = c(11, 15, 19, 43, 53, -4, 34, 8, 33, -1, 54 )
mean_ci( myvec, FALSE )
mean_ci( myvec, FALSE, ci = .80 )
mean_ci( myvec, ci = .80 )
```

---

mon_conv *Monitor Screen Unit Conversion*

Description

Given a specific monitor object, converts specified screen units to other specified units. The possible units to convert from and to: "cm" (centimeters), "pix" (pixels), or "deg" (degrees of visual angle).

Usage

```
mon_conv(mon_obj, value, from, to)
```

Arguments

mon_obj	Monitor object, as assigned with mon_neat.
value	Number; value (magnitude) of the given unit to convert from. (Can be vector as well.)
from	String; unit ("cm", "pix", or "deg") to convert from.
to	String; unit ("cm", "pix", or "deg") to convert to.

Value

Number (magnitude) in the given output (to) unit.

See Also

mon_neat

Examples

```
# assign monitor with 50 cm distance, screen width 52 cm and 1920 pixels
my_mon = mon_neat(distance = 50, mon_width_cm = 52, mon_width_pixel = 1920)

# convert 30.4 pixels to degrees of visual angle, for the specified monitor
mon_conv(my_mon, 30.4, 'pix', 'deg') # returns 0.9434492 (degrees)

# convert 0.94 degrees of visual angle to pixels
mon_conv(my_mon, 0.94, 'deg', 'pix') # returns 30.28885 (pixels)

# convert 10 degrees of visual angle to cm
mon_conv(my_mon, 10, from = 'deg', to = 'cm')

# convert 8.748866 cm to pixels
mon_conv(my_mon, 8.748866, from = 'cm', to = 'pix')
```

---

mon_neat *Monitor Object*

Description

Assigns a monitor object, storing distance and width parameters.

Usage

```
mon_neat(distance, mon_width_cm, mon_width_pixel)
```

Arguments

distance	Viewing distance in cm (from eyes to screen).
mon_width_cm	Monitor screen width in cm.
mon_width_pixel	Monitor screen width in pixels.

Value

A monitor object with the specified parameters, to be used in the mon_conv function.

See Also

mon_conv

Examples

```
# assign monitor with 57 cm viewing distance, screen width 52 cm and 1920 pixels
my_mon = mon_neat(distance = 57, mon_width_cm = 52, mon_width_pixel = 1920)
```

norm_tests *Normality Tests and Plots*

Description

Performs normality tests and creates related plots (histogram, density, Q-Q). This is primarily a sub-function of t_neat and anova_neat, but here it is available separately for other potential purposes.

Usage

```
norm_tests(
  var1,
  var2 = NULL,
  pair = FALSE,
  norm_tests = "all",
  alpha = 0.05,
  plots = FALSE,
  aspect_ratio = 1,
  hush = FALSE
)
```

Arguments

var1	Numeric vector; numbers of any given variable.
var2	Optional numeric vector (or NULL); numbers of a second variable.
pair	Logical; only matters if var2 is not null. In that case, if TRUE each normality test is performed for the difference values between the two variables in case of paired samples, or, if FALSE, separately for each of the two variables for unpaired samples.
norm_tests	Normality tests. Any or all of the following character input is accepted (as a single string or a character vector; case-insensitive): "W" (Shapiro-Wilk), "K2" (D'Agostino), "A2" (Anderson-Darling), "JB" (Jarque-Bera); see Notes. The option "all" (default value) selects all four previous tests at the same time.
alpha	Numeric (.05 by default), alpha level: if any p value if below this alpha level, the function returns TRUE, otherwise FALSE.
plots	Logical: if TRUE adds histogram, density, and Q-Q plots. (Note: in case of paired samples, Q-Q plots are plotted on a separate figure. In RStudio, press on "Previous plot" under "Plots" to see these Q-Q plots.)
aspect_ratio	Aspect ratio of the plots: 1 (1/1) by default. (Set to NULL for dynamic aspect ratio.)
hush	Logical. If TRUE, prevents printing any details to console.

Value

Prints normality tests, and displays plots if so specified. Returns TRUE if any of the specified tests has p value below the specified alpha, otherwise returns FALSE.

Note

Normality tests are all calculated via `fBasics::NormalityTests`, selected based on the recommendation of Lakens (2015), quoting Yap and Sim (2011, p. 2153): "If the distribution is symmetric with low kurtosis values (i.e. symmetric short-tailed distribution), then the D'Agostino and Shapiro-Wilkes tests have good power. For symmetric distribution with high sample kurtosis (symmetric long-tailed), the researcher can use the JB, Shapiro-Wilkes, or Anderson-Darling test." See urlhttps://github.com/Lakens/perfect-t-test for more details.

References

Lakens, D. (2015). The perfect t-test (version 1.0.0). Retrieved from https://github.com/Lakens/perfect-t-test. doi:10.5281/zenodo.17603

Yap, B. W., & Sim, C. H. (2011). Comparisons of various types of normality tests. Journal of Statistical Computation and Simulation, 81(12), 2141–2155. doi:10.1080/00949655.2010.520163

See Also

`t_neat`

Examples

```
norm_tests(stats::rnorm(100))
# should be normal...
```

---

path_neat *Script Path*

Description

Gives, in RStudio, the path to the script file in which it is executed.

Usage

```
path_neat(subdir = "")
```

Arguments

`subdir`	String, optional. Subdirectory relative to the script's path.

Value

Script file's path as string. If `subdir` is given, it is appended to the original path.

Examples

```
# assuming the given script is at path "C:/script_folder/"
path_neat('') # returns "C:/script_folder/"
path_neat('my_subdir/misc/') # returns "C:/script_folder/my_subdir/misc/"

# Note: the returned string can be used as argument for base::setwd()
# e.g. setwd( path_neat() ) # sets working directory to the script's path
```

---

peek_neat *Cursory Summaries and Plots per Group*

Description

Cursory summaries and plots per group.

Usage

```
peek_neat(
  dat,
  values,
  group_by = NULL,
  filt = NULL,
  sep = ", ",
  collapse = NULL,
  f_print = FALSE,
  f_plot = NULL,
  iqr_times = 3,
  round_to = 4,
  group_n = TRUE,
  ...
)
```

Arguments

dat	Data frame (or name of data frame as string).
values	String, or vector of strings: the name(s) of the column(s) in the dat data frame, containing the vector(s) of values.
group_by	String, or vector of strings: the name(s) of the column(s) in the dat data frame, containing the vector(s) of factors by which the statistics are grouped.
filt	An expression to filter, by column values, the entire dat data frame before performing the aggregation. The expression should use column names alone; see Examples.
sep	String (comma by default) for separating group names.

collapse	Decides how to handle multiple columns of values. If NULL (default), displays each column of values as separate groups. Alternatively, any function can be given using which the columns are collapsed into a single column. For example, if mean is given for this parameter, a single column will be calculated based on the means of all given values columns. (NA is always ignored.)
f_print	Printing function; see details.
f_plot	Plotting function; see details. (Provide string to skip plotting.)
iqr_times	The multiplication of IQR to calculate Tukey's fences, when using default f_print function (set to TRUE); see Details. The default is 3 to spot Tukey's "far outliers" (e.g. by comparing the fences with min and max values). (Note that the usual fences, e.g. for box plots, use 1.5).
round_to	Number of significant fractional digits to round to, when using default f_print function (set to TRUE).
group_n	Logical. If TRUE, adds sample sizes (n) per group to plots when using default f_plot (NULL).
...	Any arguments to be passed to the f_plot function.

Details

If set to TRUE, prints to console the following data (per group): mean; 95 median; quantile_1st and quantile_3rd (first and third quantiles); "Tukey's fences" as fence_low and fence_upp; minimum and maximum values (min, max); number of NAs (na). Tukey's fences are the upper and lower limits with distances of X times the IQR from the actual IQR, where X is specified via the iqr_times parameter. Returns (invisibly) the same values, unrounded, via a data frame. If alternative f_print is given, prints whatever value is returned from the given function (and attempts, if possible, to create a data frame).

Creates and displays box plot(s) (per group) by default, along with overlayed violin plot (densities proportionate to sample sizes). If alternative f_plot is given, the first argument will be the values per group, and all plots will be arranged into a single plot and displayed together. To skip plotting, just give any character as argument (e.g. "none" or just "").

Value

Data frame with the printed values (if possible).

Examples

```
data("mtcars") # load base R example dataset

# overall info for wt (Weight)
peek_neat(mtcars, 'wt', f_print = TRUE)
#'
# now groupped by cyl (Number of cylinders)
peek_neat(mtcars, 'wt', group_by = 'cyl')

# grouped by cyl and gear
peek_neat(mtcars,
          'wt',
```

```
          group_by = c('cyl', 'gear'),
          f_print = TRUE)

# filter to only have cyl larger than  4
peek_neat(mtcars, 'wt', group_by = 'cyl', filt = cyl > 4)

# without plots
peek_neat(mtcars,
          'wt',
          group_by = 'cyl',
          f_plot = "",
          f_print = TRUE)

# with histogramms etc, using plot_neat
peek_neat(mtcars, 'wt', group_by = 'cyl', f_plot = plot_neat)

# with Q-Q plots, via ggpubr
peek_neat(mtcars,
          'wt',
          group_by = 'cyl',
          f_plot = ggpubr::ggqqplot)

# skewness and kurtosis data via psych
## Not run:
info_df = peek_neat(
    mtcars,
    'wt',
    group_by = 'cyl',
    f_print = psych::describe,
    f_plot = ""
)
info_df # contains all data returns by psych::describe

## End(Not run)
```

`plot_neat` *Plots of Means and of Dispersion*

Description

Primarily for line and bar plots for factorial designs. Otherwise (if no `data_per_subject` is given) descriptive dispersion plots (histogram, density, or box plots) for a continuous variable. (For the latter, only the parameters `values`, `parts`, `part_colors`, and `binwidth` are used, the rest are ignored.)

Usage

```
plot_neat(
  data_per_subject = NULL,
```

```
  values = NULL,
  within_ids = NULL,
  between_vars = NULL,
  factor_names = NULL,
  value_names = NULL,
  y_title = NULL,
  reverse = FALSE,
  panels = NULL,
  type = "line",
  dodge = NULL,
  bar_colors = "viridis",
  line_colors = "viridis",
  row_number = 1,
  method = mean,
  eb_method = neatStats::mean_ci,
  numerics = FALSE,
  hush = FALSE,
  parts = c("h", "d", "n", "b"),
  part_colors = NULL,
  binwidth = NULL
)
```

Arguments

`data_per_subject`
: Data frame containing all values (measurements/observations for a factorial design) in a single row per each subject. Otherwise, if no data frame is given (default: `NULL`), histogram, density, or box plots will be returned for a continuous variable (numeric vector).

`values`
: For plots of means (factorial designs): vector of strings; column name(s) in the `data_per_subject` data frame. Each column should contain a single dependent variable: thus, to plot repeated (within-subject) measurements, each specified column should contain one measurement. For descriptive dispersion plots (if `data_per_subject` is `NULL`), a numeric vector is expected.

`within_ids`
: `NULL` (default), string, or named list. In case of no within-subject factors, leave as `NULL`. In case of a single within subject factor, a single string may be given to optionally provide custom name for the within-subject factor (note: this is a programming variable name, so it should not contain spaces, etc.); otherwise (if left `NULL`) this one within-subject factor will always just be named `"within_factor"`. In case of multiple within-subject factors, each factor must be specified as a named list element, each with a vector of strings that distinguish the levels within that factors. The column names given as `values` should always contain one (and only one) of these strings within each within-subject factor, and thus they will be assigned the appropriate level. For example, `values = 'rt_s1_neg, rt_s1_pos,rt_s2_neg, rt_s2_pos'` could have `within_ids = list( session = c('s1','s2'), valence = c('pos', 'neg')`. (Note: the strings for distinguishing must be unambiguous. E.g., for values `apple_a` and `apple_b`, do not set levels `c('a','b')`, because `'a'` is also found in `apple_b`.

In this case, you could choose levels `c('_a','_b')` to make sure the values are correctly distinguished.) See also Examples.

`between_vars` `NULL` (default; in case of no between-subject factors) or vector of strings; column name(s) in the `data_per_subject` data frame. Each column should contain a single between-subject independent variable (representing between-subject factors).

`factor_names` `NULL` or named vector. In a named vector, factor names (either within or between) can be given a different name for display, in a dictionary style, using original factor name as the name of a vector element, and the element's value (as string) for the new name. For example, to change a factor named `"condition"` to `"High vs. low arousal"`, the vector may be given (in this case with a single element) as `factor_names = c(condition = "High vs. low arousal")`.

`value_names` `NULL` or named vector. Same as `factor_names`, but regarding the factor values. For example, to change values `"high_a"` and `"low_a"` to `"High"` and `"Low"` for display, the vector may be given as `value_names = c(high_a = "High",low_a = "Low")`.

`y_title` `NULL` (default) or string. Optionally given title for the `y` axis.

`reverse` Logical (default: `FALSE`). If `TRUE`, reverses the default grouping of variables within the figure, or within each panel, in case of multiple panels. (The default grouping is decided automatically by given factor order, but always starting, when applicable, with within-subject factors: first factor is split to adjacent bars, or vertically aligned dots in case of line plot.)

`panels` `NULL` or string. Optionally gives the factor name by which the plot is to be split into different panels, in case of three factors. (By default, the third given factor is used.)

`type` Strong: `"line"` (default) or `"bar"`. The former gives line plot, the latter gives bar plot.

`dodge` Number. Specifies the amount by which the adjacent bars or dots 'dodge' each other (i.e., are displaced compared to each other). (Default is `0.1` for `line` plots, and `0.9` for `bar` plots.)

`bar_colors` Vector of strings, specifying colors from which all colors for any number of differing adjacent bars are interpolated. (If the number of given colors equal the number of different bars, the precise colors will correspond to each bar.) The default `'viridis'` gives a color gradient based on viridis. (In case of a single factor, the first given colors is taken.)

`line_colors` Vector of strings, specifying colors from which all colors for any number of differing vertically aligned dots and corresponding lines are interpolated. The default `'viridis'` gives a color gradient based on viridis. (In case of a single factor, the first given colors is taken.)

`row_number` Number. In case of multiple panels, the number of rows in which the panels should be arranged. For example, with the default `row_number = 1`, all panels will be displayed in one vertically aligned row.

`method` A function (default: `mean`) for the calculation of the main statistics (bar or dot heights).

eb_method — A function (default: mean_ci for 95 the calculation of the error bar size (as a single value used for both directions of the error bar). If set to NULL, no error bar is displayed.#'

numerics — If FALSE (default), returns ggplot object. If set to TRUE, returns only the numeric aggregated data per grouping factors, as specified by method and eb_method functions. If set to any string (e.g. "both"), returns the numeric aggregated data and at the same time draws the plot.

hush — Logical. If TRUE, prevents printing aggregated values.

parts — For dispersion plots only (if no data_per_subject is given). A vector of characters that specify which types of overlayed types to plot: "h" for histogram, "d" for density, "n" normally distributed density (using the mean and standard deviation of the given variable), "b" for boxplot. (All are included by default: parts = c("h", "d", "n", "b")).

part_colors — For dispersion plots only (if no data_per_subject is given). A named that can specify and thereby override default colors and alpha (transparency) of each plot type. Colors can be given by adding "c" to the plot type letter, e.g. c(hc = "blue") for blue histogram. Alpha can be given by adding "a" to the plot type letter, e.g. c(ha = 0) for completely transparent histogram. Any number may be given: e.g. a dark red transparent histogram with green boxplot would be part_colors = c(hc = "#cc0000", ha = 0.1, bc = "green").

binwidth — For dispersion plots only (if no data_per_subject is given). Binwidth for histograms. If NULL (default), Freedman–Diaconis rule is used if it produces at least 10 bins – otherwise 1bandwidth is calculated for 10 bins.

Value

By default, a ggplot plot object. (This object may be further modified or adjusted via regular ggplot methods.) If so set (numerics), aggregated values as specified by the methods.

Note

More than three factors is not allowed: it would make little sense and it would be difficult to clearly depict in a simple figure. (However, you can build an appropriate graph using ggplot directly; but you can also just divide the data to produce several three-factor plots, after which you can use e.g. ggpubr's ggarrange to easily collate the plots.)

See Also

anova_neat, mean_ci, se

Examples

```
# assign random data in a data frame for illustration
# (note that the 'subject' is only for illustration; since each row contains the
# data of a single subject, no additional subject id is needed)
dat_1 = data.frame(
    subject = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14),
    grouping1 = c(1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2),
```

```
    grouping2 = c(1, 2, 1, 2, 2, 1,2, 1,2,1, 1, 1, 2, 1),
    value_1_a = c(36.2, 45.2, 41, 24.6, 30.5, 28.2, 40.9, 45.1,
                  31, 16.9, 40.1, 42.1, 41, 12.9),
    value_2_a = c(-14.1, 58.5,-25.5, 42.2,-13, 4.4, 55.5,-28.5,
                  25.6,-37.1, 55.1,-38.5, 28.6,-34.1),
    value_1_b = c(83, 71, 111, 70, 92, 75, 110, 111, 110, 85,
                  132, 121, 151, 95),
    value_2_b = c(8.024,-14.162, 3.1,-2.1,-1.5, 0.91, 11.53,
                  18.37, 0.3,-0.59, 12.53, 13.37, 2.3,-3),
    value_1_c = c(27.4, -17.6, -32.7, 0.4, 37.2, 1.7, 18.2, 8.9,
                  1.9, 0.4, 2.7, 14.2, 3.9, 4.9),
    value_2_c = c(7.7,-0.8, 2.2, 14.1, 22.1,-47.7,-4.8, 8.6,
                  6.2, 18.2,-6.8, 5.6, 7.2, 13.2)
)
head(dat_1) # see what we have

# plot for factors 'grouping1', 'grouping2'
plot_neat(
    data_per_subject = dat_1,
    values = 'value_1_a',
    between_vars = c('grouping1', 'grouping2')
)

# same as above, but with bars and renamed factors
plot_neat(
    data_per_subject = dat_1,
    values = 'value_1_a',
    between_vars = c('grouping1', 'grouping2'),
    type = 'bar',
    factor_names = c(grouping1 = 'experimental condition', grouping2 = 'gender')
)

# same, but with different (lighter) gray scale bars
plot_neat(
    dat_1,
    values = 'value_1_a',
    between_vars = c('grouping1', 'grouping2'),
    type = 'bar',
    factor_names = c(grouping1 = 'experimental condition', grouping2 = 'gender'),
    bar_colors = c('#555555', '#BBBBBB')
)

# same, but with red and blue bars
plot_neat(
    dat_1,
    values = 'value_1_a',
    between_vars = c('grouping1', 'grouping2'),
    type = 'bar',
    factor_names = c(grouping1 = 'experimental condition', grouping2 = 'gender'),
    bar_colors = c('red', 'blue') # equals c('#FF0000', '#0000FF')
)

# within-subject factor for 'value_1_a' vs. 'value_1_b' vs. 'value_1_c'
```

```
# (automatically named 'within_factor'), between-subject factor 'grouping1'
plot_neat(
    dat_1,
    values = c('value_1_a', 'value_1_b', 'value_1_c'),
    between_vars = c('grouping1', 'grouping2')
)

# same, but panelled by 'within_factor'
plot_neat(
    dat_1,
    values = c('value_1_a', 'value_1_b', 'value_1_c'),
    between_vars = c('grouping1', 'grouping2'),
    panels = 'within_factor'
)

# same, but SE for error bars instead of (default) SD
plot_neat(
    dat_1,
    values = c('value_1_a', 'value_1_b', 'value_1_c'),
    between_vars = c('grouping1', 'grouping2'),
    panels = 'within_factor',
    eb_method = se
)

# same, but 95% CI for error bars instead of SE
# (arguably more meaningful than SEs)
plot_neat(
    dat_1,
    values = c('value_1_a', 'value_1_b', 'value_1_c'),
    between_vars = c('grouping1', 'grouping2'),
    panels = 'within_factor',
    eb_method = mean_ci
)

# same, but using medians and Median Absolute Deviations
plot_neat(
    dat_1,
    values = c('value_1_a', 'value_1_b', 'value_1_c'),
    between_vars = c('grouping1', 'grouping2'),
    panels = 'within_factor',
    method = stats::median,
    eb_method = stats::mad
)

# within-subject factor 'number' for variables with number '1' vs. number '2'
# ('value_1_a' and 'value_1_b' vs. 'value_2_a' and 'value_2_b'), factor 'letter'
# for variables with final letter 'a' vs. final letter 'b' ('value_1_a' and
# 'value_2_a' vs. 'value_1_b' and 'value_2_b')
plot_neat(
    dat_1,
    values = c('value_1_a', 'value_2_a', 'value_1_b', 'value_2_b'),
    within_ids = list(
        letters = c('_a', '_b'),
```

```
        numbers =  c('_1', '_2')
    )
)

# same as above, but now including between-subject factor 'grouping2'
plot_neat(
    dat_1,
    values = c('value_1_a', 'value_2_a', 'value_1_b', 'value_2_b'),
    within_ids = list(
        letters = c('_a', '_b'),
        numbers =  c('_1', '_2')
    ),
    between_vars = 'grouping2'
)

# same as above, but renaming factors and values for display
plot_neat(
    dat_1,
    values = c('value_1_a', 'value_2_a', 'value_1_b', 'value_2_b'),
    within_ids = list(
        letters = c('_a', '_b'),
        numbers =  c('_1', '_2')
    ),
    between_vars = 'grouping2',
    factor_names = c(numbers = 'session (first vs. second)'),
    value_names = c(
        '_1' = 'first',
        '_2' = 'second',
        '1' = 'group 1',
        '2' = 'group 2'
    )
)

# In real datasets, these could of course be more meaningful. For example, let's
# say participants rated the attractiveness of pictures with low or high levels
# of frightening and low or high levels of disgusting qualities. So there are
# four types of ratings:
# 'low disgusting, low frightening' pictures
# 'low disgusting, high frightening' pictures
# 'high disgusting, low frightening' pictures
# 'high disgusting, high frightening' pictures

# this could be meaningfully assigned e.g. as below
pic_ratings = data.frame(
    subject = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
   rating_fright_low_disgust_low = c(36.2, 45.2, 41, 24.6, 30.5, 28.2, 40.9, 45.1, 31, 16.9),
   rating_fright_high_disgust_low = c(-14.1, 58.5,-25.5, 42.2,-13, 4.4, 55.5,-28.5, 25.6,-37.1),
    rating_fright_low_disgust_high = c(83, 71, 111, 70, 92, 75, 110, 111, 110, 85),
   rating_fright_high_disgust_high = c(8.024,-14.162, 3.1,-2.1,-1.5, 0.91, 11.53, 18.37, 0.3,-0.59)
)
head(pic_ratings) # see what we have

# the same logic applies as for the examples above, but now the
```

```
# within-subject differences can be more meaningfully specified, e.g.
# 'disgust_low' vs. 'disgust_high' for levels of disgustingness, while
# 'fright_low' vs. 'fright_high' for levels of frighteningness
plot_neat(
    pic_ratings,
    values = c(
        'rating_fright_low_disgust_low',
        'rating_fright_high_disgust_low',
        'rating_fright_low_disgust_high',
        'rating_fright_high_disgust_high'
    ),
    within_ids = list(
        disgustingness = c('disgust_low', 'disgust_high'),
        frighteningness =  c('fright_low', 'fright_high')
    )
)

# now let's say the ratings were done in two separate groups
pic_ratings = data.frame(
    subject = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
    group_id = c(1, 2, 1, 2, 2, 1, 1, 1, 2, 1),
   rating_fright_low_disgust_low = c(36.2, 45.2, 41, 24.6, 30.5, 28.2, 40.9, 45.1, 31, 16.9),
   rating_fright_high_disgust_low = c(-14.1, 58.5,-25.5, 42.2,-13, 4.4, 55.5,-28.5, 25.6,-37.1),
    rating_fright_low_disgust_high = c(83, 71, 111, 70, 92, 75, 110, 111, 110, 85),
   rating_fright_high_disgust_high = c(8.024,-14.162, 3.1,-2.1,-1.5, 0.91, 11.53, 18.37, 0.3,-0.59)
)

# now include the 'group_id' factor in the plot
plot_neat(
    pic_ratings,
    values = c(
        'rating_fright_low_disgust_low',
        'rating_fright_high_disgust_low',
        'rating_fright_low_disgust_high',
        'rating_fright_high_disgust_high'
    ),
    within_ids = list(
        disgustingness = c('disgust_low', 'disgust_high'),
        frighteningness =  c('fright_low', 'fright_high')
    ),
    between_vars = 'group_id'
)


## DISPERSION PLOTS

plot_neat(values = rnorm(100))

# with smaller binwidth (hence more bins)
plot_neat(values = rnorm(100), binwidth = 0.2)

# without normal distribution line
plot_neat(values = rnorm(100), parts = c('h', 'd', 'b'))
```

```
# without histrogram
plot_neat(values = rnorm(100), parts = c('d', 'n', 'b'))

# blue density, fully opaque histogram
plot_neat(values = rnorm(100),
         part_colors = c(dc = 'blue', ha = 1))
```

`props_neat` *Difference of Two Proportions*

Description

Comparison of paired and unpaired proportions. For unpaired: Pearson's chi-squared test or unconditional exact test, including confidence interval (CI) for the proportion difference, and corresponding independent multinomial contingency table Bayes factor (BF). (Cohen's h and its CI are also calculated.) For paired tests, classical (asymptotic) McNemar test (optionally with mid-P as well), including confidence interval (CI) for the proportion difference.

Usage

```
props_neat(
  var1 = NULL,
  var2 = NULL,
  case1 = NULL,
  case2 = NULL,
  control1 = NULL,
  control2 = NULL,
  prop1 = NULL,
  prop2 = NULL,
  n1 = NULL,
  n2 = NULL,
  pair = FALSE,
  greater = NULL,
  ci = NULL,
  bf_added = FALSE,
  round_to = 3,
  exact = FALSE,
  inverse = FALSE,
  yates = FALSE,
  midp = FALSE,
  h_added = FALSE,
  for_table = FALSE,
  hush = FALSE
)
```

Arguments

`var1`	First variable containing classifications, in 'group 1', for the first proportion (see Examples). If given (strictly necessary for paired proportions), proportions will be defined using `var1` and `var2` (see Details). To distinguish classification ('cases' and 'controls'; e.g. positive outcomes vs. negative outcomes), any two specific characters (or numbers) can be used. However, more than two different elements (apart from `NA`s) will cause error.
`var2`	Second variable containing classifications in, 'group 2', for the second proportion, analogously to `var1`.
`case1`	Number of 'cases' (as opposed to 'controls'; e.g. positive outcomes vs. negative outcomes) in 'group 1'. As counterpart, either control numbers or sample sizes needs to be given (see Details).
`case2`	Number of 'cases' in 'group 2'.
`control1`	Number of 'controls' in 'group 1'. As counterpart, case numbers need to be given (see Details).
`control2`	Number of 'controls' in 'group 2'.
`prop1`	Proportion in 'group 1'. As counterpart, sample sizes need to be given (see Details).
`prop2`	Proportion in 'group 2'.
`n1`	Number; sample size of 'group 1'.
`n2`	Number; sample size of 'group 2'.
`pair`	Logical. Set `TRUE` for paired proportions (McNemar, mid-P), or `FALSE` (default) for unpaired (chi squared, or unconditional exact test). Note: paired data must be given in `var1` and `var2`.
`greater`	`NULL` or string (or number); optionally specifies one-sided exact test: either "1" (`case1/n1` proportion expected to be greater than `case2/n2` proportion) or "2" (`case2/n2` proportion expected to be greater than `case1/n1` proportion). If `NULL` (default), the test is two-sided.
`ci`	Numeric; confidence level for the returned CIs (proportion difference and Cohen's h).
`bf_added`	Logical. If `TRUE`, Bayes factor is calculated and displayed. (Always two-sided!)
`round_to`	Number `to round` to the proportion statistics (difference and CIs).
`exact`	Logical, `FALSE` by default. If `TRUE`, `unconditional exact test` is calculated and displayed, otherwise the default Pearson's `chi-squared test`.
`inverse`	Logical, `FALSE` by default. When `var1` and `var2` are given to calculate proportion from, by default the factors' frequency determines which are 'cases' and which are 'controls' (so that the latter are more frequent). If the `inverse` argument is `TRUE`, it reverses the default proportion direction.
`yates`	Logical, `FALSE` by default. If `TRUE`, Yates' continuity correction is applied to the chi-squared (unpaired) or the McNemar (paired) test. Some authors advise this correction for certain specific cases (e.g., small sample), but evidence does not seem to support this (Pembury Smith & Ruxton, 2020).

midp	Logical, FALSE by default. If TRUE, displays an additional 'mid-P' p value (using the formula by Pembury Smith & Ruxton, 2020) for McNemar's test (Fagerland et al., 2013). This provides better control for Type I error (less false positive findings) than the classical McNemar test, while it is also probably not much less robust (Pembury Smith & Ruxton, 2020).
h_added	Logical. If TRUE, Cohen's h and its CI are calculated and displayed. (FALSE by default.)
for_table	Logical. If TRUE, omits the confidence level display from the printed text.
hush	Logical. If TRUE, prevents printing any details to console.

Details

The proportion for the two groups can be given using any of the following combinations (a) two vectors (var1 and var2), (b) cases and controls, (c) cases and sample sizes, or (d) proportions and sample sizes. Whenever multiple combinations are specified, only the first parameters (as given in the function and in the previous sentence) will be taken into account.

The Bayes factor (BF), in case of unpaired samples, is always calculated with the default r-scale of 0.707. BF supporting null hypothesis is denoted as BF01, while that supporting alternative hypothesis is denoted as BF10. When the BF is smaller than 1 (i.e., supports null hypothesis), the reciprocal is calculated (hence, BF10 = BF, but BF01 = 1/BF). When the BF is greater than or equal to 10000, scientific (exponential) form is reported for readability. (The original full BF number is available in the returned named vector as bf.)

Value

Prints exact test statistics (including proportion difference with CI, and BF) in APA style. Furthermore, when assigned, returns a named vector with the following elements: z (Z), p (p value), prop_diff (raw proportion difference), h (Cohen's h), bf (Bayes factor).

Note

Barnard's unconditional exact test is calculated via Exact::exact.test ("z-pooled").

The CI for the proportion difference in case of the exact test is calculated based on the p value, as described by Altman and Bland (2011). In case of extremely large or extremely small p values, this can be biased and misleading.

The Bayes factor is calculated via BayesFactor::contingencyTableBF, with sampleType = "indepMulti", as appropriate when both sample sizes (n1 and n2) are known in advance (as it normally happens). (For details, see contingencyTableBF, or e.g. 'Chapter 17 Bayesian statistics' in Navarro, 2019.)

References

Altman, D. G., & Bland, J. M. (2011). How to obtain the confidence interval from a P value. Bmj, 343(d2090). doi:10.1136/bmj.d2090

Barnard, G. A. (1947). Significance tests for 2x2 tables. Biometrika, 34(1/2), 123-138. doi:10.1093/biomet/34.12.123

Fagerland, M. W., Lydersen, S., & Laake, P. (2013). The McNemar test for binary matched-pairs data: Mid-p and asymptotic are better than exact conditional. BMC Medical Research Methodology, 13(1), 91. doi:10.1186/147122881391

Lydersen, S., Fagerland, M. W., & Laake, P. (2009). Recommended tests for association in 2x2 tables. Statistics in medicine, 28(7), 1159-1175. doi:10.1002/sim.3531

Navarro, D. (2019). Learning statistics with R. https://learningstatisticswithr.com/

Pembury Smith, M. Q. R., & Ruxton, G. D. (2020). Effective use of the McNemar test. Behavioral Ecology and Sociobiology, 74(11), 133. doi:10.1007/s0026502002916y

Suissa, S., & Shuster, J. J. (1985). Exact unconditional sample sizes for the 2 times 2 binomial trial. Journal of the Royal Statistical Society: Series A (General), 148(4), 317-327. doi:10.2307/2981892

Examples

```
# example data
set.seed(1)
outcomes_A = sample(c(rep('x', 490), rep('y', 10)))
outcomes_B = sample(c(rep('x', 400), rep('y', 100)))

# paired proportion test (McNemar)
props_neat(var1 = outcomes_A,
           var2 = outcomes_B,
           pair = TRUE)

# unpaired chi test for the same data (two independent samples assumed)
# Yates correction applied
# cf. http://www.sthda.com/english/wiki/two-proportions-z-test-in-r
props_neat(
    var1 = outcomes_A,
    var2 = outcomes_B,
    pair = FALSE,
    yates = TRUE
)

# above data given differently for unpaired test
# (no Yates corrrection)
props_neat(
    case1 = 490,
    case2 = 400,
    control1 = 10,
    control2 = 100
)

# again differently
props_neat(
    case1 = 490,
    case2 = 400,
    n1 = 500,
    n2 = 500
)

# other example data
```

```
outcomes_A2 = c(rep(1, 707), rep(0, 212),  rep(1, 256), rep(0, 144))
outcomes_B2 = c(rep(1, 707), rep(0, 212),  rep(0, 256), rep(1, 144))

# paired test
# cf. https://www.medcalc.org/manual/mcnemartest2.php
props_neat(var1 = outcomes_A2,
           var2 = outcomes_B2,
           pair = TRUE)

# show reverse proportions (otherwise the same)
props_neat(
    var1 = outcomes_A2,
    var2 = outcomes_B2,
    pair = TRUE,
    inverse = TRUE
)


# two different sample sizes
out_chi = props_neat(
    case1 = 40,
    case2 = 70,
    n1 = 150,
    n2 = 170
)

# exact test
out_exact = props_neat(
    case1 = 40,
    case2 = 70,
    n1 = 150,
    n2 = 170,
    exact = TRUE
)

# the two p values are just tiny bit different
print(out_chi) # p 0.00638942
print(out_exact) # p 0.006481884

# one-sided test
props_neat(
    case1 = 40,
    case2 = 70,
    n1 = 150,
    n2 = 170,
    greater = '2'
)
```

---

rbind_loop *Merge by Columns in Loops*

Description

Merges rows by columns in a loop using the `enum` function. On first iteration, indicated by `enum`, initiates a new `data.table` with the data to merge as first row. On all following iterations, adds data to merge as subsequent rows (using `data.table::rbindlist`).

Usage

```
rbind_loop(merged, ..., hush = FALSE)
```

Arguments

`merged`	The name of the `data.table` for the merged data (without quotes).
`...`	Any number of data to be merged. Each argument must be one of the following: a `data.frame` (or `data.table`) with either single row or two column; a named vector or a named list (with single elements); or a single value with parameter name (e.g. date = 1989 or id = "jdoe"). Data with two columns will be transposed using first column as column names and second column as corresponding values. See Details, Examples.
`hush`	Logical. If `TRUE` (default), prints message when the data frame for merging is initiated.

Details

In each call, all data passed to the function (via `...`) will be merged into a single row, and that single row will be added to the "`merged`" data table.

See an extensive example via https://github.com/gasparl/neatstats.

See Also

`enum`

Examples

```
my_vector = c('aa', 'bb', 'cxyz', 'last')
for (elem in enum(my_vector)) {
    cat(elem, fill = TRUE)
    rbind_loop(
        merged_data, # data frame name for merging
        item = elem[2],
        number = elem[1],
        whatever = paste0('number (', elem[1], ')')
    )
}
# merged_data now contains all merged rows
print(merged_data)
# item number   whatever
# 1   aa      1 number (1)
# 2   bb      2 number (2)
# 3 cxyz      3 number (3)
```

```
# 4 last      4 number (4)


# example with other data types
for (elem in enum(my_vector)) {
    cat(elem, fill = TRUE)
    dframe1 = data.frame(item = elem[2],
                         number = elem[1])
    print(elem[1])
    asnum = as.numeric(elem[1])
    dframe2 = data.frame(
        my_cols = c('index', 'squared', 'multiple'),
        my_vals = c(elem[1], asnum ** 2, asnum * 10)
    )
    my_list = list(ls_item = elem[2], ls_num = elem[1])
    my_vec = c(v_item = elem[2], v_num = elem[1])
    rbind_loop(
        merged_data,
        dframe1, # data frame with single row
        dframe2, # data frame with two columns
        my_list, # named list
        my_vec, # named vector
        single_val = elem[2], # single element
        constant = "whatever" # other single element
    )
}

# again merged_data contains all merged rows
# (previous content, if any, were removed)
print(merged_data)


# example with differring columns
for (elem in enum(my_vector)) {
    cat(elem, fill = TRUE)
    dframe = data.frame(item = elem[2],
                        number = elem[1])
    asnum = as.numeric(elem[1])
    if (asnum %% 2 == 0) {
        dframe$sqr = asnum ** 2
    }
    rbind_loop(merged_data,
               dframe)
}

# merged_data contains all new merged rows
# with NAs where sqr was not added
print(merged_data)

# example with data.table added
library('data.table')
for (elem in enum(my_vector)) {
    cat(elem, fill = TRUE)
```

```
    dframe = data.frame(item = elem[2],
                        number = elem[1])
    asnum = as.numeric(elem[1])
    dtable = data.table(item2 = paste('DT', elem[2]),
                        number2 = asnum + 9)
    if (asnum %% 2 == 0) {
        dframe$sqr = asnum ** 2
    }
    rbind_loop(merged_data,
               dframe,
               dtable)
}

print(merged_data)

# an extensive example to show how to collect and aggregate raw data is
# available via the README file at the repository:
# https://github.com/gasparl/neatstats
```

`read_dir` *Read and Merge Files from Directory*

Description

Reads data files from any given directory as data frames and merges them into a single data frame (using `data.table::rbindlist`).

Usage

```
read_dir(
  pattern = "*[.]",
  path = ".",
  reader_function = data.table::fread,
  ...,
  subdirs = FALSE,
  filt = NULL,
  hush = FALSE
)
```

Arguments

`pattern`	Regular expression ("regex"; as string or `NULL`) for selecting files (passed to the `list.files` function). The default `NULL` means that all files at the specified path will be read in. To select, for example, a specific extension like ".txt", the pattern can be given as `"\.txt$"` (for CSV files, `"\.csv$"`, etc.). Files ending with e.g. "group2.txt" can be specified as `"group2\.txt$"`. Files starting with "exp3" can be specified as `"^exp3"`. Files starting with "exp3" AND ending with ".txt" extension can be specified as `"^exp3.*\.txt$"`. To read in a single file,

specify the full filename (e.g. "exp3_subject46_group2.txt"). (See ?regex for more details.)

path — Path to the directory from which the files should be selected and read. The default "." means the current working directory (as returned by getwd()). Either specify correct working directory in advance (see setwd, path_neat), or otherwise enter relative or full paths (e.g. "C:/research" or "/home/projects", etc.).

reader_function — A function to be used for reading the files, data.table::fread by default.

... — Any arguments to be passed on to the chosen reader_function.

subdirs — Logical (FALSE by default). If TRUE, searches files in subdirectories as well (relative to the given path).

filt — An expression to filter, by column values, each data file after it is read and before it is merged with the other data. (The expression should use column names alone; see Examples.)

hush — Logical. If FALSE (default), prints lists all data file names as they are being read (along with related warnings).

Note

This function is very similar to the readbulk::read_bulk function. One important difference however is the data.table use, which greatly speeds up the process. Another important difference is the possibility of file selection based on any regex pattern. Furthermore, this function allows pre-filtering by file (see filt). Data files could include significant amount of unnecessary data, and filtering prevents these to be merged.

See Also

data.table::rbindlist

Examples

```
# first, set current working directory
# e.g. to script's path with setwd(path_neat())

# read all text files in currect working directory
merged_df = read_dir("\\.txt$")
# merged_df now has all data

# to use utils::read.table for reading (slower than fread)
# (with some advisable options passed to it)
merged_df = read_dir(
    '\\.txt$',
    reader_function = read.table,
    header = TRUE,
    fill = TRUE,
    quote = "\"",
    stringsAsFactors = FALSE
```

```
)
```

---

ro *Neat rounding*

Description

Rounds a given number to given number of digits after the decimal point, returning it as string, with trailing zeros when applicable.

Usage

```
ro(num, round_to = 2, leading_zero = TRUE, signi = FALSE)
```

Arguments

num	Number to be rounded.
round_to	Number of fractional digits (i.e., digits after the decimal point), to round to.
leading_zero	Logical, TRUE by default. If FALSE, omits leading zero (e.g. returns ".17" instead of "0.17").
signi	Logical, FALSE by default. If TRUE, rounds to a fractional digit that allows at least the first N non-zero digits displayed in all numbers, where N is specified by the round_to parameter. (See base:formatC)

Value

Number as string: num rounded to round_to digits, with trailing zeros when applicable.

Examples

```
ro( 1.2345 ) # returns "1.23"

ro( 0.12345, 1 ) # returns "0.1"

ro( 12.3, 4 ) # returns "12.3000"

# examples with vectors
to_round = c(1000, 100, 0.1, 0.01, 0.001, 0.0001)
ro(to_round)
ro(to_round, 3)
ro(to_round, 3, leading_zero = FALSE)
ro(to_round, 3, signi = TRUE)
to_round2 = c(1230.000, 100, 0.012, 0.01, 0.123, 0.012340)
ro(to_round2, 3)
ro(to_round2, 3, signi = TRUE)
ro(to_round2, 2, signi = TRUE)
```

```
ro(to_round2, 1, signi = TRUE)
ro(to_round2, 9, signi = TRUE)
```

`roc_neat` *Difference of Two Areas Under the Curves*

Description

Comparison of two areas under the receiver operating characteristic curves (AUCs) and plotting any number of ROC curves.

Usage

```
roc_neat(
  roc1,
  roc2 = NULL,
  pair = FALSE,
  greater = NULL,
  ci = NULL,
  hush = FALSE,
  plot_rocs = FALSE,
  roc_labels = "",
  cutoff_auto = TRUE,
  cutoff_custom = NULL
)
```

Arguments

`roc1`	Receiver operating characteristic (ROC) `object`, or, for plotting only, a `list` including any number of such ROC objects.
`roc2`	Receiver operating characteristic (ROC) `object`, or, for plotting only, leave it as `NULL` (default) and provide list for the first parameter (`roc1`).
`pair`	Logical. If `TRUE`, the test is conducted for paired samples. Otherwise (default) for independent samples.
`greater`	`NULL` or string (or number); optionally specifies one-sided test: either "1" (`roc1` AUC expected to be greater than `roc2` AUC) or "2" (`roc2` AUC expected to be greater than `roc2` AUC). If `NULL` (default), the test is two-sided.
`ci`	Numeric; confidence level for the returned CIs (raw difference).
`hush`	Logical. If `TRUE`, prevents printing any details to console (and plotting).
`plot_rocs`	Logical. If `TRUE`, plots and returns ROC curves.
`roc_labels`	Optional character vector to provide legend label texts (in the order of the provided ROC objects) for the ROC plot.
`cutoff_auto`	Logical. If `TRUE` (default), optimal cutoffs on the ROC plots are displayed.

`cutoff_custom` Custom cutoff to be indicated on the plot can be given here in a `list`. The list index must exactly correspond to the index of the list index of the AUC (given in `roc1`) for which the given cutoff is intended.

Value

Prints DeLong's test results for the comparison of the two given AUCs in APA style, as well as corresponding CI for the AUC difference. Furthermore, when assigned, returns a list with `stat` (D value), `p` (p value), and, when plot is added, ROC plot.

Note

The main test statistics are calculated via `pROC::roc.test` as DeLong's test (for both paired and unpaired). The `roc_neat` function merely prints it in APA style. The CI is calculated based on the p value, as described by Altman and Bland (2011).

The ROC object may be calculated via `t_neat`, or directly with `pROC::roc`.

References

Altman, D. G., & Bland, J. M. (2011). How to obtain the confidence interval from a P value. Bmj, 343(d2090). doi:10.1136/bmj.d2090

DeLong, E. R., DeLong, D. M., & Clarke-Pearson, D. L. (1988). Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach. Biometrics, 44(3), 837-845. doi:10.2307/2531595

Robin, X., Turck, N., Hainard, A., Tiberti, N., Lisacek, F., Sanchez, J. C., & Muller, M. (2011). pROC: an open-source package for R and S+ to analyze and compare ROC curves. BMC bioinformatics, 12(1), 77. doi:10.1186/147121051277

See Also

`t_neat`

Examples

```
# calculate first AUC (from v1 and v2)
v1 = c(191, 115, 129, 43, 523,-4, 34, 28, 33,-1, 54)
v2 = c(4,-2, 23, 13, 32, 16, 3, 29, 37,-4, 65)
results1 = t_neat(v1, v2, auc_added = TRUE)

# calculate second AUC (from v3 and v4)
v3 = c(14.1, 58.5, 25.5, 42.2, 13, 4.4, 55.5, 28.5, 25.6, 37.1)
v4 = c(36.2, 45.2, 41, 24.6, 30.5, 28.2, 40.9, 45.1, 31, 16.9)
results2 = t_neat(v3, v4, auc_added = TRUE)

# one-sided comparison of the two AUCs
roc_neat(results1$roc_obj, results2$roc_obj, greater = "1")


# create a list of randomlz generated AUCs
set.seed(1)
```

```
aucs_list = list()
for (i in 1:4) {
    aucs_list[[i]] = t_neat(rnorm(50, (i-1)),
                            rnorm(50),
                            auc_added = TRUE,
                            hush = TRUE)$roc_obj
}
# depict AUCs (recognized as list)
roc_neat(aucs_list)


# with custom cutoffs depicted
roc_neat(aucs_list,
         cutoff_custom = list(0.2),
         cutoff_auto = FALSE)
roc_neat(aucs_list,
         cutoff_custom = list(.1, c(-.5, 0), NULL, c(.7, 1.6)),
         cutoff_auto = FALSE)
roc_neat(aucs_list,
         cutoff_custom = list(.6, NULL, NULL, 1.1))
```

---

sd_ci *Confidence Interval of Standard Deviation*

Description

Calculates the SD confidence interval of a vector of numbers.

Usage

```
sd_ci(x, ci = 0.95)
```

Arguments

x	Numeric vector.
ci	Numeric; confidence level for returned CI.

Value

SD confidence interval (as named vector).

See Also

mean_ci, plot_neat

Examples

```
myvec = c(11, 15, 19, 43, 53, -4, 34, 8, 33, -1, 54 )
sd_ci( myvec )
sd_ci( myvec, ci = .80 )
```

---

se *Standard Error of Mean*

Description

Simply calculates the standard error of a vector of numbers.

Usage

```
se(x)
```

Arguments

x	Numeric vector.

Value

Standard error.

See Also

mean_ci, plot_neat

Examples

```
se( c(11, 15, 19, 43, 53, -4, 34, 8, 33, -1, 54 ) )
```

---

table_neat *Table, descriptives*

Description

Creates a neat means (or similar descriptives) and standard deviations table, using aggr_neat functions as arguments. Alternatively, merges and transposes data frames into rows.

Usage

```
table_neat(
  values_list,
  group_by = NULL,
  group_per = "rows",
  to_clipboard = FALSE,
  method = "mean+sd",
  transpose = FALSE
)
```

Arguments

values_list	Data frames as returned from the aggr_neat function: variables from which the statistics for the table are to be calculated. The group_by, method, and prefix parameters are ignored when they are given in the table_neat function; see Details and also an extensive example via https://github.com/gasparl/neatstats.
group_by	String, or vector of strings: the name(s) of the column(s) in the dat given data frame, containing the vector(s) of factors by which the statistics are grouped. (Overwrites group_by in aggr_neat; see Details.)
group_per	String, "rows" or "columns". If set to "columns" (or just "c" or "col", etc.), each column contains statistics for one group. Otherwise (default), each row contains statistics for one group.
to_clipboard	Logical. If TRUE, the table is copied to the clipboard (default: FALSE).
method	Function or string; overwrites the method argument in aggr_neat when used within this function. See method in the aggr_neat function for details. Default value: "mean+sd" (to calculate means and standard deviations table).
transpose	Logical (default: FALSE) or string. If TRUE or string, ignores all other parameters (except values_list), but merges all values from given list of data frames (as returned from the aggr_neat) and transposes them into a single row, using, by default, the "aggr_group" column values for new headers (corresponding to the output of aggr_neat; see Examples). However, a string given as argument for the transpose parameter can also specify a custom column name.

Details

The values, round_to, and new_name arguments given in the aggr_neat function are always applied. However, the prefix parameter will be overwritten as NULL. If new_name in aggr_neat is NULL, the given input variable names will be used instead of "aggr_value". Furthermore, the group_by or method given in the aggr_neat function are only applied when no arguments are given in the table_neat function for the identical parameters (group_by or medians). If either parameter is given in the table_neat function, all separately given respective argument(s) in the aggr_neat function(s) are ignored.

Value

Returns a data frame with means or medians and SDs per variable and per group.

See Also

aggr_neat for more related details

Examples

```
data("mtcars") # load base R example dataset

# overall means and SDs table for disp (Displacement) and hp (Gross horsepower)
table_neat(list(aggr_neat(mtcars, disp),
                aggr_neat(mtcars, hp)))

# means and SDs table for mpg (Miles/(US) gallon), wt (Weight), and hp (Gross horsepower)
# grouped by cyl (Number of cylinders)
# each measure rounded to respective optimal number of digits
# wt renamed to weight (for the column title)
table_neat(list(
    aggr_neat(mtcars, mpg, round_to = 1),
    aggr_neat(mtcars, wt, new_name = 'weight', round_to = 2),
    aggr_neat(mtcars, hp, round_to = 0)
),
group_by = 'cyl')

# same as above, but with medians, and with groups per columns
table_neat(
    list(
        aggr_neat(mtcars, mpg, round_to = 1),
        aggr_neat(mtcars, wt, new_name = 'weight', round_to = 2),
        aggr_neat(mtcars, hp, round_to = 0)
    ),
    group_by = 'cyl',
    method = 'median+sd',
    group_per = 'columns'
)

# an extensive example to show how to collect and aggregate raw data is
# available via the README file at the repository:
# https://github.com/gasparl/neatstats
```

---

t_neat *Difference of Two Means and Area Under the Curve*

Description

Welch's t-test results including Cohen's d with confidence interval (CI), Bayes factor (BF), and area under the receiver operating characteristic curve (AUC). For non-parametric version, Wilcoxon test results (Mann–Whitney U test, aka "Wilcoxon rank-sum test", for independent samples; Wilcoxon signed-rank test for paired samples; including nonparametric "location difference estimate" (see stats::wilcox.test); along with corresponding rank-based BFs as per van Doorn et al., 2020).

Usage

```
t_neat(
  var1,
  var2,
  pair = FALSE,
  nonparametric = FALSE,
  greater = NULL,
  norm_tests = "latent",
  norm_plots = FALSE,
  ci = NULL,
  bf_added = FALSE,
  bf_rscale = sqrt(0.5),
  bf_sample = 1000,
  auc_added = FALSE,
  cutoff = NULL,
  r_added = TRUE,
  for_table = FALSE,
  test_title = NULL,
  round_descr = 2,
  round_auc = 3,
  auc_greater = "1",
  cv_rep = FALSE,
  cv_fold = 10,
  hush = FALSE,
  plots = FALSE,
  rug_size = 4,
  aspect_ratio = 1,
  y_label = "density estimate",
  x_label = "\nvalues",
  factor_name = NULL,
  var_names = c("1", "2"),
  reverse = FALSE
)
```

Arguments

var1	Numeric vector; numbers of the first variable.
var2	Numeric vector; numbers of the second variable.
pair	Logical. If TRUE, all tests (t, BF, AUC) are conducted for paired samples. If FALSE (default) for independent samples.
nonparametric	Logical (FALSE by default). If TRUE, uses nonparametric (rank-based, "Wilcoxon") t-tests (including BFs; see Notes).
greater	NULL or string (or number); optionally specifies one-sided tests (t and BF): either "1" (var1 mean expected to be greater than var2 mean) or "2" (var2 mean expected to be greater than var1 mean). If NULL (default), the test is two-sided.
norm_tests	Normality tests. Any or all of the following character input is accepted (as a single string or a character vector; case-insensitive): "W" (Shapiro-Wilk), "K2"

	(D'Agostino), "A2" (Anderson-Darling), "JB" (Jarque-Bera); see Notes. Two other options are "all" (same as TRUE; to choose all four previous tests at the same time) or "latent" (default value; prints all tests only if nonparametric is set to FALSE and any of the four tests gives a p value below .05). Each normality test is performed for the difference values between the two variables in case of paired samples, or for each of the two variables for unpaired samples. Set to "none" to disable (i.e., not to perform any normality tests).
norm_plots	If TRUE, displays density, histogram, and Q-Q plots (and scatter plots for paired tests) for each of the two variable (and differences for pairwise observations, in case of paired samples).
ci	Numeric; confidence level for returned CIs for Cohen's d and AUC.
bf_added	Logical. If TRUE (default), Bayes factor is calculated and displayed.
bf_rscale	The scale of the prior distribution (0.707 by default).
bf_sample	Number of samples used to estimate Bayes factor (1000 by default). More samples (e.g. 10000) take longer time but give more stable BF.
auc_added	Logical (FALSE by default). If TRUE, AUC is calculated and displayed. Includes TPR and TNR, i.e., true positive and true negative rates, i.e. sensitivity and specificity, using an optimal value, i.e. threshold, that provides maximal TPR and TNR. These values may be cross-validated: see cv_rep. (Note that what is designated as "positive" or "negative" depends on the scenario: this function always assumes var1 as positive and var2 as negative. If your scenario or preference differs, you can simply switch the names or values when reporting the results.)
cutoff	Numeric. Custom cutoff value for AUC TPR and TNR, also to be depicted in the plot. In case of multiple given, the first is used for calculations, but all will be depicted in the plot.
r_added	Logical. If TRUE (default), Pearson correlation is calculated and displayed in case of paired comparison.
for_table	Logical. If TRUE, omits the confidence level display from the printed text.
test_title	NULL or string. If not NULL, simply displayed in printing preceding the statistics. (Useful e.g. to distinguish several different comparisons inside a function or a for loop.)
round_descr	Number to round to the descriptive statistics (means and SDs).
round_auc	Number to round to the AUC and its CI.
auc_greater	String (or number); specifies which variable is expected to have greater values for 'cases' as opposed to 'controls': "1" (default; var1 expected to be greater for 'cases' than var2 mean) or "2" (var2 expected to be greater for 'cases' than var1). Not to be confused with one-sided tests; see Details.
cv_rep	FALSE (default), TRUE, or numeric. If TRUE or numeric, a cross-validation is performed for the calculation of TPRs and TNRs. Numeric value specifies the number of repetitions, while, if TRUE, it defaults to 100 repetitions. In each repetition, the data is divided into k random parts ("folds"; see cv_fold), and the optimal accuracy is obtained k times from a k-1 training set (var1 and var2 truncated to equal length, if needed, in each case within each repetition), and the TPR and TNR are calculated from the remaining test set (different each time).

cv_fold	Numeric. The number of folds into which the data is divided for cross-validation (default: 10).
hush	Logical. If TRUE, prevents printing any details to console.
plots	Logical (or NULL). If TRUE, creates a combined density plot (i.e., Gaussian kernel density estimates) from the two variables. Includes dashed vertical lines to indicate means of each of the two variables. If nonparametric is set to TRUE, medians are calculated for these dashed lines instead of means. When auc_added is TRUE (and the AUC is at least .5), the best threshold value for classification (maximal differentiation accuracy using Youden's index) is added to the plot as solid vertical line. (In case of multiple best thresholds with identical overall accuracy, all are added.) If NULL, same as if TRUE except that histogram is added to the background.
rug_size	Numeric (4 by default): size of the rug ticks below the density plot. Set to 0 (zero) to omit rug plotting.
aspect_ratio	Aspect ratio of the plots: 1 (1/1) by default. (Set to NULL for dynamic aspect ratio.)
y_label	String or NULL; the label for the y axis. (Default: "density estimate".)
x_label	String or NULL; the label for the x axis. (Default: "values".)
factor_name	String or NULL; factor legend title. (Default: NULL.)
var_names	A vector of two strings; the variable names to be displayed in the legend. (Default: c("1", "2").)
reverse	Logical. If TRUE, reverses the order of variable names displayed in the legend.

Details

The Bayes factor (BF) supporting null hypothesis is denoted as BF01, while that supporting alternative hypothesis is denoted as BF10. When the BF is smaller than 1 (i.e., supports null hypothesis), the reciprocal is calculated (hence, BF10 = BF, but BF01 = 1/BF). When the BF is greater than or equal to 10000, scientific (exponential) form is reported for readability. (The original full BF number is available in the returned named vector as bf.)

For simplicity, Cohen's d is reported for nonparametric tests too: you may however want to consider reporting alternative effect sizes in this case.

The original pROC::auc function, by default, always returns an AUC greater than (or equal to) .5, assuming that the prediction based on values in the expected direction work correctly at least at chance level. This however may be confusing. Consider an example where we measure the heights of persons in a specific small sample and expect that greater height predicts masculine gender. The results are, say, 169, 175, 167, 164 (cm) for one gender, and 176, 182, 179, 165 for the other. If the expectation is correct (the second, greater values are for males), the AUC is .812. However, if in this particular population females are actually taller than males, the AUC is in fact .188. To keep things clear, the t_neat function always makes an assumption about which variable is expected to be greater for correct classification ("1" by default; i.e., var1; to be specified as auc_greater = "2" for var2 to be expected as greater). For this example, if the first (smaller) variables are given as var1 for females, and second (larger), variables are given as var2 for males, we have to specify auc_greater = "2" to indicate the expectation of larger values for males. (Or, easier, just add the expected larger values as var1.)

Value

Prints t-test statistics (including Cohen's d with CI, BF, and AUC, as specified via the corresponding parameters) in APA style. Furthermore, when assigned, returns a list, that contains a named vector 'stats' with the following elements: t (t value), p (p value), d (Cohen's d), bf (Bayes factor), auc (AUC), accuracy (overall accuracy using the optimal classification threshold), and youden (Youden's index: specificity + sensitivity - 1). The latter three are NULL when auc_added is FALSE. When auc_added is TRUE, there are also two or three additional elements of the list. One is 'roc_obj', which is a roc object, to be used e.g. with the roc_neat function. Another is 'best_thresholds', which contains the best threshold value(s) for classification, along with corresponding specificity and sensitivity. The third 'cv_results' contains the results, if any, of the cross-validation of TPRs and TNRs (means per repetition). Finally, if plots is TRUE (or NULL), the plot is displayed as well as returned as a ggplot object, named t_plot.

Note

The Welch's t-test is calculated via stats::t.test.

#'Normality tests are all calculated via fBasics::NormalityTests, selected based on the recommendation of Lakens (2015), quoting Yap and Sim (2011, p. 2153): "If the distribution is symmetric with low kurtosis values (i.e. symmetric short-tailed distribution), then the D'Agostino and Shapiro-Wilkes tests have good power. For symmetric distribution with high sample kurtosis (symmetric long-tailed), the researcher can use the JB, Shapiro-Wilkes, or Anderson-Darling test." See urlhttps://github.com/Lakens/perfect-t-test for more details.

Cohen's d and its confidence interval are calculated, using the t value, via MBESS::ci.smd for independent samples (as standardized mean difference) and via MBESS::ci.sm for paired samples (as standardized mean).

The parametric Bayes factor is calculated via BayesFactor::ttestBF. The nonparametric (rank-based) Bayes factor is a contribution by Johnny van Doorn; the original source code is available via https://osf.io/gny35/.

The correlation and its CI are calculated via stats::cor.test, and is always two-sided, always with 95 percent CI. For more, use corr_neat.

The AUC and its CI are calculated via pROC::auc, and the accuracy at optimal threshold via pROC::coords (x = "best"); both using the object pROC::roc.

References

Delacre, M., Lakens, D., & Leys, C. (2017). Why psychologists should by default use Welch's t-test instead of Student's t-test. International Review of Social Psychology, 30(1). doi:10.5334/irsp.82

Kelley, K. (2007). Methods for the behavioral, educational, and social sciences: An R package. Behavior Research Methods, 39(4), 979-984. doi:10.3758/BF03192993

Lakens, D. (2015). The perfect t-test (version 1.0.0). Retrieved from https://github.com/Lakens/perfect-t-test. doi:10.5281/zenodo.17603

Robin, X., Turck, N., Hainard, A., Tiberti, N., Lisacek, F., Sanchez, J. C., & Muller, M. (2011). pROC: an open-source package for R and S+ to analyze and compare ROC curves. BMC bioinformatics, 12(1), 77. doi:10.1186/147121051277

van Doorn, J., Ly, A., Marsman, M., & Wagenmakers, E.-J. (2020). Bayesian rank-based hypothesis testing for the rank sum test, the signed rank test, and Spearman's rho. Journal of Applied Statistics, 1–23. doi:10.1080/02664763.2019.1709053

Yap, B. W., & Sim, C. H. (2011). Comparisons of various types of normality tests. Journal of Statistical Computation and Simulation, 81(12), 2141–2155. doi:10.1080/00949655.2010.520163

See Also

corr_neat, roc_neat

Examples

```
# assign two variables (numeric vectors)
v1 = c(191, 115, 129, 43, 523,-4, 34, 28, 33,-1, 54)
v2 = c(4,-2, 23, 13, 32, 16, 3, 29, 37,-4, 65)

t_neat(v1, v2) # prints results as independent samples
t_neat(v1, v2, pair = TRUE) # as paired samples (r added by default)
t_neat(v1, v2, pair = TRUE, greater = "1") # one-sided
t_neat(v1, v2, pair = TRUE, auc_added = TRUE ) # AUC included

# print results and assign returned list
results = t_neat(v1, v2, pair = TRUE)

results$stats['bf'] # get precise BF value
```

---

var_tests *Variance Equality Tests*

Description

Displayed sample sizes and SDs and performs Brown-Forsythe and Fligner-Killeen variance equality tests (tests of homogeneity of variances) per group combinations. This is primarily a subfunction of anova_neat, but here it is available separately for other potential purposes.

Usage

```
var_tests(xvar, group_by, dat = NULL, hush = FALSE, sep = ", ")
```

Arguments

xvar	Either a numeric vector (numbers of any given variable), or, if dat is given, a column name specifying the variable in the given data frame.
group_by	Either a vector of factors with which to group the xvar values, or, if dat is given, one or more column names specifying the columns in the given data frame.
dat	Either NULL or a data frame from which the respective column names should be selected for xvar and group.

hush	Logical. If TRUE, prevents printing any details to console.
sep	String (underscore "_" by default) for separating group names.

Value

Prints test results.

Note

Brown-Forsythe test (i.e., Levene's test using medians) is calculated via car::leveneTest. Fligner-Killeen test, which may be more robust (i.e., less affected by non-normal distribution), is calculated via stats::fligner.test. (See also Conover et al., 1981, p. 360.)

References

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

Conover W. J., Johnson M. E., & Johnson M. M. (1981). A comparative study of tests for homogeneity of variances, with applications to the outer continental shelf bidding data. Technometrics, 23, 351–361.

Fligner, M. A. & Killeen, T. J. (1976). Distribution-free two-sample tests for scale. 'Journal of the American Statistical Association. 71(353), 210-213.

Fox, J. & Weisberg, S. (2019) An R Companion to Applied Regression, Third Edition, Sage.

Levene, H. (1960). Robust tests for equality of variances. In I. Olkin, H. Hotelling, et al. (eds.). Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling. Stanford University Press. pp. 278–292.

See Also

anova_neat

Examples

```
data("ToothGrowth") # load base R example dataset

# the statistics of the four functions below should match
var_tests(ToothGrowth$len, ToothGrowth$supp)
var_tests('len', 'supp', ToothGrowth)
car::leveneTest(len ~ supp, data = ToothGrowth)
stats::fligner.test(len ~ supp, ToothGrowth)

# again the results below should match each other
var_tests(ToothGrowth$len,
          interaction(ToothGrowth$supp, ToothGrowth$dose))
var_tests('len', c('supp', 'dose'), ToothGrowth)
car::leveneTest(len ~ supp * as.factor(dose), data = ToothGrowth)
stats::fligner.test(len ~ interaction(supp, dose), ToothGrowth)
```

Index