Package 'neatStats'

July 18, 2019

Title An R Package for Neat and Painless Statistical Reporting

Version 0.3.1

Date 2019-07-18

Description This package focuses on 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 (>= 3.4.0)

Imports bayestestR, pROC, MBESS, ez, BayesFactor, Exact, ggplot2, grDevices, stats, graphics

Suggests rstudioapi

License BSD_2_clause + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

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,
  prefix = NULL, new_name = NULL, round_to = 2)
```

Arguments

dat	Data frame (or name of data frame 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. (Expression or string are both accepted.)
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. The elements must be specified simply divided by commas. 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.
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 frame 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)
)

# both mean and median
aggr_neat(
    mtcars,
    gear,
    group_by = 'cyl',
    method = function(v) {
        c(my_mean = mean(v), my_median = median(v))
    }
)

# mean, median, and count
aggr_neat(
    mtcars,
    gear,
    group_by = 'cyl',
    method = function(v) {
        c(mean = mean(v),
          median = median(v),
          count = length(v))
    }
)
```

---

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), 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, bf_added = TRUE,
  test_title = "--- neat ANOVA ---", welch = TRUE,
  e_correction = NULL)
```

Arguments

data_per_subject	Data frame or name of data frame as string. 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.)
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.)
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).

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.)

Levene's test is returned for between-subject design without Welch's correction, while 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). If Levene's test is significant and the data is unbalanced (unequal group sample sizes), you should either consider the results respectively or choose a different test.

Value

Prints ANOVA statistics (including, for each model, F-test with partial eta squared and its CI, and BF, as specified via the corresponding parameters) in APA style. Furthermore, when assigned, returns a list with up to three 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), epsilon (epsilon used for correction), and bf (inclusion BF; when bf_added is not FALSE). Second, the ezANOVA object, named ez_anova (calculated even when oneway.test is printed). Third, when bf_added is not FALSE, the anovaBF object, named bf_models; including all models on which the inclusion BFs are based.

Note

All F-tests are calculated via ez::ezANOVA, including Levene's test and 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).

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. (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-in
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. (2013). 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

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
)

# 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)
```

---

corr_neat *Correlation Statistics*

Description

Pearson correlation results including confidence interval (CI) and correlation Bayes factor (BF).

Usage

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

Arguments

var1	Numeric vector; numbers of the first variable.
var2	Numeric vector; numbers of the second variable.
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.

Details

The Bayes factor (BF) 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 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 Bayes factor is calculated via `BayesFactor::correlationBF`.

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, percent = TRUE,
  round_perc = 0)
```

Arguments

`data_per_subject`	Data frame from which demographics are to be calculated. Must contain columns named precisely as "age" and as "gender". The age column must contain numbers, while gender column must contain 1 (= male) and 2 (= female) only (either as numbers or as strings).
`group_by`	A vector of factors by which the statistics are grouped, typically a column from the data frame provided as `data_per_subject`.
`percent`	Logical. If `TRUE` (default), gender ratio is presented as percent of males. If `FALSE`, presented as count of males.
`round_perc`	Number `to round` to, when using percents.

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, 17, 16, 45, 77),
    measure_x = c(83, 71, 111, 70, 92, 75, 110, 111, 110, 85)
)

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

---

mean_ci *Confidence Interval of Mean limits*

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

Examples

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

plot_neat *Plots of factors*

Description

Bar and line plots for factorial designs.

Usage

```
plot_neat(data_per_subject, values, 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 = c("#333333", "#AAAAAA"),
  line_colors = c("#444444", "#000000"), row_number = 1,
  method = mean, eb_method = stats::sd)
```

Arguments

`data_per_subject`
: Data frame or name of data frame as string. 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 plot 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).

`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 c('#333333','#AAAAAA') gives a color gradient from dark gray to light gray.
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 c('#444444','#000000') gives a color gradient from dark gray to black.
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: sd) for 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.

Value

A ggplot plot object. (This object may be further modified or adjusted via regular ggplot methods.)

Note

The number of factors (within and between together) must be either two or three. Plot for a single factor would make little sense, while more than three is difficult to clearly depict in a simple plot. (In the latter case, you probably want to 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'
)
```

`props_neat` *Difference of Two Proportions*

Description

`Unconditional exact test` results for the comparison of two independent proportions, 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.

Usage

```
props_neat(case1, case2, n1, n2, greater = NULL, ci = NULL,
  bf_added = TRUE, h_added = FALSE, for_table = FALSE)
```

Arguments

`case1`	Number of 'cases' (as opposed to 'controls'; e.g. positive outcomes vs. negative outcomes) in 'group 1'.
`case2`	Number of 'cases' in 'group 2'.
`n1`	Number; sample size of 'group 1'.
`n2`	Number; sample size of 'group 2'.
`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` (default), Bayes factor is calculated and displayed. (Always two-sided.)
`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.

Details

The Bayes factor (BF) 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 CIs for the proportion difference is calculated based on the p value, as described by Altman and Bland (2011).

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

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/

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

```
props_neat(
    case1 = 35,
    case2 = 48,
    n1 = 80,
    n2 = 77,
    h_added = TRUE
)
```

```
props_neat(
    case1 = 35,
    case2 = 48,
    n1 = 80,
    n2 = 77,
    greater = "2"
)
```

`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)
```

Arguments

`num`	Number to be rounded.
`round_to`	Number of fractional digits (i.e., digits after the decimal point), to round to.

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"
```

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

Description

Comparison of two areas under the receiver operating characteristic curves (AUCs).

Usage

```
roc_neat(roc1, roc2, pair = FALSE, greater = NULL)
```

Arguments

roc1	Receiver operating characteristic (ROC) object.
roc2	Receiver operating characteristic (ROC) object.
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.

Value

Prints DeLong's test results for the comparison of the two given AUCs in APA style. Furthermore, when assigned, returns a named vector with the following two elements: stat (D value), p (p value).

Note

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

References

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")
```

script_path *Script Path*

Description

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

Usage

```
script_path(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

```
setwd( script_path() ) # sets working directory to the script's path, e.g. "C:/script_folder/"

script_path('my_subdir/misc/') # returns "C:/script_folder/my_subdir/misc/"
```

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, simply transposes data frames using first column as headers.

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.
`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 the given list of data frames (as returned from the `aggr_neat`) and then transposes them 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'
)

###

# the long example below illustrates collecting and aggregating data from
# participants, using the table_neat transpose == TRUE option

# for a more complete version of this example, see the README file at the
# repository: https://github.com/gasparl/neatstats

# you can completely ignore the following function
# it serves merely to simulate example data
next_subject = function(sub_num) {
    N = 150
    sub_dat = data.frame(
        subject_num = toString(sub_num),
        condition = sample(c('fullvision', 'colorblind'), 1),
        rt = stats::rnorm(n = N, mean = 400, sd = 150),
        response = sample(
            c(rep('correct', 9), 'incorrect', 'tooslow'),
```

```
            size = N,
            replace = TRUE
        ),
        color = sample(c('red', 'green'), size = N, replace = TRUE),
        valence = sample(
            c('positive', 'negative'),
            size = N,
            replace = TRUE
        )
    )
    if (sub_dat$condition[1] == 'fullvision') {
        green_neg = (sub_dat$color == 'green' &
                         sub_dat$valence == 'negative')
        sub_dat$rt[green_neg] = sub_dat$rt[green_neg] + stats::rnorm(
                                    n = length(sub_dat$rt[green_neg]),
                                    mean = 43,
                                    sd = 30)
        sub_dat$response[green_neg] = sample(
            c(rep('correct', 7), 'incorrect', 'tooslow'),
            size = length(sub_dat$response[green_neg]),
            replace = TRUE
        )
        red_pos = (sub_dat$color == 'red' &
                       sub_dat$valence == 'positive')
     sub_dat$rt[red_pos] = sub_dat$rt[red_pos] + stats::rnorm(n = length(sub_dat$rt[red_pos]),
                                                              mean = 37,
                                                              sd = 30)
        sub_dat$response[red_pos] = sample(
            c(rep('correct', 7), 'incorrect', 'tooslow'),
            size = length(sub_dat$response[red_pos]),
            replace = TRUE
        )
    }
    neg = (sub_dat$valence == 'negative')
    sub_dat$rt[neg] = sub_dat$rt[neg] + stats::rnorm(n = length(sub_dat$rt[neg]),
                                                     mean = 40,
                                                     sd = 25)
    sub_dat$response[neg] = sample(
        c(rep('correct', 6), 'incorrect', 'tooslow'),
        size = length(sub_dat$response[neg]),
        replace = TRUE
    )
    return(sub_dat)
}
# the variable below simulates file names
filenames = 1:5 # (just 5 sets for this example)

# in brief, each data file (as data frame) contains:
# subject_num: the number (id) of subject
# condition: "fullvision" vs "colorblind"
# rt: Response Times
# response: "correct", "incorrect", or "tooslow"
# color: "green" vs "red"
# valence: "positive" vs "negative"
# see e.g.
head(next_subject(1)) # (here the function gives the data)
```

```
# we want to aggregate this, per subject, to get the following RT values:
# rt_green_negative rt_green_positive rt_red_negative rt_red_positive
# and also Error Rates (correct/(correct+incorrect)):
# er_green_negative er_green_positive er_red_negative er_red_positive

# for this, loop through all data files
for (file_name in filenames) {
    subject_data = next_subject(file_name)
    # with real data, this would be e.g.:
    # subject_data = read.table(file_name, stringsAsFactors=F, fill=T, header=T)

    # print current file name - just to monitor the process
    cat(file_name, ' ')

    # now aggregate rt data per type
    rts = aggr_neat(
        subject_data,
        rt,
        group_by = c('color', 'valence'),
        method = mean,
        prefix = 'rt'
    )

    # same with error rates
    ers = aggr_neat(
        subject_data,
        response,
        group_by = c('color', 'valence'),
        method = 'incorrect',
        prefix = 'er',
        filt = (response %in% c('correct', 'incorrect'))
    )

    # merge and transpose here to get the subject's data in one line
    subject_line = table_neat(list(rts, ers), transpose = TRUE)

    # add the subject_id and condition to the beginning
    subject_line = data.frame(
        subject_id = subject_data$subject_num[1],
        condition = subject_data$condition[1],
        subject_line
    )

    # merge aggregated subject data
    if (!exists("subjects_merged")) {
        # if doesn't yet exist, create first line
        subjects_merged = subject_line
    } else {
        # if exists, add the next lines
        subjects_merged = rbind(subjects_merged, subject_line)
    }
}
# the data, with single rows per subject, is ready for analysis:
head(subjects_merged)
```

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).

Usage

```
t_neat(var1, var2, pair = FALSE, greater = NULL, ci = NULL,
  bf_added = TRUE, auc_added = FALSE, r_added = TRUE,
  for_table = FALSE, test_title = "Descriptives:", round_descr = 2,
  round_auc = 3, auc_greater = "1", plot_densities = FALSE,
  y_label = "density estimate", x_label = "values",
  factor_name = NULL, var_names = c("1", "2"))
```

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.
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.
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.
auc_added	Logical. If TRUE, AUC is calculated and displayed. (FALSE by default.)
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	String, "Descriptives:" by default. Simply displayed in printing preceding the descriptive 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.
plot_densities	Logical. If TRUE, creates a density plot (i.e., Gaussian kernel density estimates) from the two variables. When auc_added is TRUE (and the AUC is at least .5), the best threshold value for classification (maximal differentiation accuracy) is added to the plot as vertical line. (In case of multiple best thresholds with identical overall accuracy, all are added.)
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").)

Details

The Bayes factor (BF) 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.)

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 (accuracy using the most optimal classification threshold). The latter two is NULL when auc_added is FALSE. When auc_added is TRUE, there are also two additional elements of the list. One is 'roc_obj', which is a roc object, to be used e.g. with the roc_neat function. The other is 'best_thresholds', which contains the best threshold value(s) for classification, along with corresponding specificity and sensitivity. Finally, if plot_densities is TRUE, the plot is displayed as well as returned as a ggplot object, named density_plot.

Note

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

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 Bayes factor is calculated via BayesFactor::ttestBF.

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 most 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

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

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
```

Index