Package ‘MoffittFunctions’

October 19, 2018

Title Moffitt Biostat Functions

Version 0.1.1

Authors William (Jimmy) Fulp, Ram Thapa

Description Statistical, data processing, and annotation functions for Moffitt Biostat.

License GPL-3

LazyData true

Imports broom (¿= 0.5.0),
coin,
data.table,
dplyr,
Exact,
Hmisc,
kableExtra,
purrr,
survival,
tibble,
tidyr

Suggests knitr,
testthat

RoxygenNote 6.1.0

Encoding UTF-8

R topics documented:

`cor_test` *Correlation Test for Two Continuous Variables*

Description

A wrapper for cor.test function, except if spearman selected and ties in at least one variable, in which case this is a wrapper for coin::spreaman_test in with approximate method.

Usage

```
cor_test(x, y, method = c("pearson", "kendall", "spearman"),
  seed = 68954857, B = 10000, exact = TRUE, verbose = FALSE)
```

Arguments

`x`	numeric vector (can include NA values)
`y`	numeric vector (can include NA values)
`method`	a character string indicating which correlation coefficient is to be used for the test. One of "pearson", "kendall", or "spearman", can be abbreviated
`seed`	seed (only used if `method = "spearman"`)
`B`	number of reps (only used if `method = "spearman"`)
`exact`	Should exact method be used. Ingorned it `method = "spearman"` and ties present
`verbose`	a logical variable indicating if warnings and messages should be displayed

Details

To always get reproducible results when using approximate method we need to set seed inside of the call, and order the data

Value

spearman_test pvalue

Examples

```
set.seed(5432322)
x <- rnorm(20,0,3)
y <- x + rnorm(20,0,5)
cor_test(x,y, method = 'pearson')
cor_test(x,y, method = 'kendall')
cor_test(x,y, method = 'spearman')
```

`paste_tbl_grp` *Pasting Together Information for Two Groups*

Description

Paste together information, often statistics, from two groups. There are two predefined combinations: mean(sd) and median[min,max], but user may also paste any single measure together.

Usage

```
paste_tbl_grp(data, vars_to_paste = "all", first_name = "Group1",
  second_name = "Group2", sep_val = " vs. ", na_str_out = "---",
  alternative = c("two.sided", "less", "greater"), digits = 0,
  trailing_zeros = TRUE, keep_all = TRUE, verbose = FALSE)
```

Arguments

`data`	input dataset. User must use consistent naming throughout, **with an underscore** to separate the group names from the measures (i.e. `Group1_mean` and `Group2_mean`). There also must be two columns with column names that exactly match the input for `first_name` and `second_name` (i.e. 'Group1' and 'Group2'), which are used to form the `Comparison` variable.
`vars_to_paste`	vector of names of common measures to paste together. Can be the predefined 'median_min_max' or 'mean_sd', or any variable as long as they have matching columns for each group (i.e. Group1_MyMeasure and Group2_MyMeasure). Multiple measures can be requested. Default: "all" will run 'median_min_max' and 'mean_sd', as well as any pairs of columns in the proper format.
`first_name`	name of first group (string before '_') . Default is 'Group1'.
`second_name`	name of second group (string before '_'). Default is 'Group2'.
`sep_val`	value to be pasted between the two measures. Default is ' vs. '.
`na_str_out`	the character to replace missing values with.
`alternative`	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". Will be used to determine the character to be pasted between the group names (`Comparison` variable). Specifying "two.sided" will use the `sep_val` input.
`digits`	integer indicating the number of decimal places to round to before pasting for numeric variables. Default is 0.
`trailing_zeros`	logical indicating if trailing zeros should be included (i.e. 0.100 instead of 0.1). Note if set to TRUE output is a character vector.
`keep_all`	logical indicating if all remaining, unpasted variables in `data` should be returned with the pasted variables. Default TRUE.
`verbose`	a logical variable indicating if warnings and messages should be displayed. Default FALSE.

Details

User must use consistant naming throughout, with a underscore to separate the group names from the measures (i.e. `Group1_mean` and `Group2_mean`). There also must be columns defining the group names (i.e. `Group1` and `Group2`), which are used to form the `Comparison` variable.

`alternative` included as a parameter so the direction can easily be seen in one-sided test. If "two.sided" is selected the value to be pasted between the two group names will be set to `sep_val`, where "greater" will use " ¿ " and "less" with use " ¡ " as the pasting value.

Value

data.frame with all the pasted values requested. Each name will have '_comparison' at the end of the names (i.e. mean_comparison, median_comparison, ...)

Examples

```
# Same examples on data.table
library(data.table)
data(exampleData_BAMA)

descriptive_stats_by_group <- exampleData_BAMA[, .(
     Group1 = unique(group[group == 1]), Group2 = unique(group[group == 2]),
     Group1_n = length(magnitude[group == 1]), Group2_n = length(magnitude[group == 2]),
    Group1_mean = mean(magnitude[group == 1]), Group2_mean = mean(magnitude[group == 2]),
     Group1_sd = sd(magnitude[group == 1]), Group2_sd = sd(magnitude[group == 2]),
   Group1_median = median(magnitude[group == 1]), Group2_median = median(magnitude[group == 2]),
     Group1_min = min(magnitude[group == 1]), Group2_min = min(magnitude[group == 2]),
     Group1_max = max(magnitude[group == 1]), Group2_max = max(magnitude[group == 2])
), by = .(visitno,antigen)]

paste_tbl_grp(data = descriptive_stats_by_group, vars_to_paste = 'all',
   first_name = 'Group1', second_name = 'Group2',
   sep_val = " vs. ", digits = 0, keep_all = TRUE)

paste_tbl_grp(data = descriptive_stats_by_group, vars_to_paste = c("mean", "median_min_max"),
   alternative= "less", keep_all = FALSE)

paste_tbl_grp(data = descriptive_stats_by_group, vars_to_paste = 'all',
   first_name = 'Group1', second_name = 'Group2', sep_val = " vs. ",
   alternative = 'less', digits = 5, keep_all = FALSE)


# Same example with tidyverse (dplyr+tidyr) with some custom functions

library(dplyr)
library(tidyr)

q95_fun = function(x) quantile(x, 0.95)
N = function(x) length(x)

exampleData_BAMA %>%
 mutate(group = paste0("Group", group)) %>%
 group_by(group, visitno, antigen) %>%
 summarise_at("magnitude", funs(N, mean, sd, median, min, max, q95_fun)) %>%
 gather(variable, value, -(group:antigen)) %>% # these three chains create a wide dataset
```

```
  unite(temp, group, variable) %>%
  spread(temp, value) %>%
  mutate(Group1 = "Group 1", Group2 = "Group 2") %>%
  paste_tbl_grp()
```

`pretty_km_output` *Fancy Table Output of KM (survfit) Fit*

Description

This function takes a Kaplan-Meier model fit object (from survival::survfit) and calculate survival estimates at a specified time, and Median Survival Estimates. This can be performed on an overall KM fit or a fit including a categorical variable (strata).

Usage

```
pretty_km_output(fit, time_est = NULL, group_name = NULL,
  title_name = NULL, surv_est_prefix = "Time", surv_est_digits = 2,
  median_est_digits = 1, latex_output = FALSE)
```

Arguments

`fit`	survfit object (with or without single strata variable)
`time_est`	numerical vector of time estimates. If NULL (default) no time estimates are calculated
`group_name`	strata variable name. If NULL and strata exists then using variable
`title_name`	title to use
`surv_est_prefix`	prefix to use in survival estimate names. Default is Time (i.e. Time:5, Time:10,...)
`surv_est_digits`	number of digits to round p values for survival estimates for specified times
`median_est_digits`	number of digits to round p values for Median Survival Estimates
`latex_output`	will this table go into a latex output (making special charaters latex friendly)

Details

Currently works with multiple strata in the fit (i.e. `survfit(Surv(time, event) ~ x1 + x2)`), although level and `Group` column names may be off.

Value

A tibble with: `Name` (if provided), `Group` (if strata variable in fit), `Level` (if strata variable in fit), `Median Estimate`, `Time:X` (Survival estimates for each time provided, if any). In no strata variable tibble is one row, otherwise nrows = number of strata levels.

Examples

```
# Basic linear model example
set.seed(542542522)
ybin <- sample(0:1, 100, replace = TRUE)
ybin2 <- sample(0:1, 100, replace = TRUE)
y <- rexp(100,.1)
x1 <- factor(sample(LETTERS[1:2],100,replace = TRUE))
x2 <- factor(sample(letters[1:4],100,replace = TRUE))
my_fit <- survival::survfit(survival::Surv(y, ybin) ~ 1)
my_fit2 <- survival::survfit(survival::Surv(y, ybin) ~ x1)
my_fit3 <- survival::survfit(survival::Surv(y, ybin) ~ x2)
my_fit_y2 <- survival::survfit(survival::Surv(y, ybin2) ~ 1)

pretty_km_output(fit = my_fit3, time_est = c(5,10), title_name = 'Overall Fit')

library(dplyr)
km_info <- bind_rows(
  pretty_km_output(fit = my_fit, time_est = c(5,10),
        group_name = 'Overall', title_name = 'Overall Survival---ybin'),
  pretty_km_output(fit = my_fit2, time_est = c(5,10),
        group_name = NULL, title_name = 'Overall Survival---ybin'),
  pretty_km_output(fit = my_fit3, time_est = c(5,10),
        group_name = 'x2', title_name = 'Overall Survival---ybin'),
  pretty_km_output(fit = my_fit_y2, time_est = c(5,10),
        group_name = 'Overall', title_name = 'Overall Survival---ybin2'),
) %>% select(Name, Group, Level, everything())

library(kableExtra)
kable(km_info, escape = F, longtable = F, booktabs = TRUE, linesep = '',
     caption = 'Survival Percentage Estimates at 5 and 10 Years') %>%
  collapse_rows(1:2, row_group_label_position = 'stack', headers_to_remove = 1:2) %>%

  # Real World Examples
  data(Bladder_Cancer)
 surv_obj <- survival::Surv(Bladder_Cancer$Survival_Months, Bladder_Cancer$Vital_Status == 'Dead')
  downstage_fit <- survival::survfit(surv_obj ~ PT0N0, data = Bladder_Cancer)

  pretty_km_output(fit = downstage_fit, time_est = c(24, 60),
       surv_est_prefix = 'Month', surv_est_digits = 3)
```

`pretty_pvalues` *Round and format a vector of p-values*

Description

pretty_pvalues() takes a vector of p-values, rounds them to a specified digit amount, allows options for emphasizing p-values ¡ the defined significance level, and returns a character for missing.

Usage

```
pretty_pvalues(pvalues, digits = 3, bold = FALSE, italic = FALSE,
  background = NULL, sig_alpha = 0.05, missing_char = "---",
  include_p = FALSE, trailing_zeros = TRUE)
```

Arguments

`pvalues`	numeric vector of raw p-values to be formatted
`digits`	number of digits to round to; values with zeros past this number of digits are truncated
`bold`	TRUE or FALSE: set to TRUE to bold p-values ¡ the defined significance level
`italic`	TRUE or FALSE: set to TRUE to italicize p-values ¡ the defined significance level
`background`	highlight color for p-values ¡ the defined significance level. Default = NULL (no highlighting)
`sig_alpha`	the defined significance level. Default = 0.05
`missing_char`	character string that will replace missing values from the p-value vector. Default = ”—”
`include_p`	TRUE or FALSE: set to TRUE to print ”p = ” before each p-value
`trailing_zeros`	TRUE or FALSE: default = TRUE, p-values are formatted with trailing zeros to the defined number of digits (i.e. 0.100 instead of 0.1 if digits = 3)

Details

With this function, there are two things to be noted: Since the p-value vector formatting uses `cell_spec`, which generates raw HTML or LaTeX code, make sure you remember to put `escape = FALSE` into your kable code when generating your table. At the same time, you will need to escape special symbols manually. Additionally, `cell_spec` needs a way to know whether you want HTML or LaTeX output. You can specify it locally in the function or globally using `options(knitr.table.format = "latex")`. If you don’t provide anything, this function will output as HTML by default.

Value

Vector of transformed p-values for table output

Examples

```

pvalue_example = c(1, 0.06, 0.0005, NA, 1e-6)

pretty_pvalues(pvalue_example, background = "pink")

pretty_pvalues(pvalue_example, digits = 4, missing_char = "missing", bold = TRUE)

# How to use pretty_pvalues in reports
raw_pvals <- c(0.00000007, .05000001, NaN, NA, 0.783)
pretty_pvals <- pretty_pvalues(raw_pvals , digits = 3, background = "green", italic = TRUE, bold = TRUE)
kableExtra::kable(pretty_pvals , format = "latex", escape = FALSE, col.names = c("P-values"))
```

round_away_0 — *Rounding Using Round Away From 0 Method*

Description

round_away_0 takes a numeric vector, rounds them to a specified digit amount using the round away from 0 method for ties (i.e. 1.5). This is the SAS method for rounding.

Usage

```
round_away_0(x, digits = 0, trailing_zeros = FALSE)
```

Arguments

`x`	numeric vector (can include NA values).
`digits`	positive integer of length 1 between 0 (default) and 14, giving the amount of digits to round to.
`trailing_zeros`	logical indicating if trailing zeros should included (i.e. 0.100 instead of 0.1). Note is set to TRUE output is a character vector

Details

`round_away_0` is not designed for use at precision levels ¡= 1e-15

Value

if `trailing_zeros = TRUE` returns a character vector of rounded values with trailing zeros, otherwise returns a numeric vector of rounded values.

Examples

```
vals_to_round = c(NA,-3.5:3.5,NA)
# [1]   NA -3.5 -2.5 -1.5 -0.5  0.5  1.5  2.5  3.5   NA

# round() will round to even numbers when tied at X.5
round(vals_to_round)
# [1] NA -4 -2 -2  0  0  2  2  4 NA

# round_away_0() will round away from 0 when tied at X.5
round_away_0(vals_to_round)
# [1] NA -4 -3 -2 -1  1  2  3  4 NA

# Can force trailing zeros (will output character vector)
round_away_0(vals_to_round, digits = 2, trailing_zeros = TRUE)
```

run_km_model — *Wrapper for KM Model Output, with Log-Rank p value*

Description

This function takes a dataset, along with variables names for time and event status for KM fit, and possibly strata

Usage

```
run_km_model(strata_in = NA, model_data, time_in, event_in,
  time_est = NULL, group_name = NULL, title_name = NULL,
  surv_est_prefix = "Time", surv_est_digits = 2,
  median_est_digits = 1, p_digits = 4, latex_output = FALSE,
  sig_alpha = 0.05, background = "yellow", ...)
```

Arguments

`strata_in`	name of strata variable, or NA (default) if no strata desired
`model_data`	dataset that contains `strata_in`, `time_in`, and `event_in` variables
`time_in`	name of time variable component of outcome measure
`event_in`	name of T/F event stauts or expression resulting in T/F scalor (i.e. "Vital_Status == 'Dead'") for the name of event variable component of outcome measure. TRUE represents event (i.e. Death)
`time_est`	numerical vector of time estimates. If NULL (default) no time estimates are calculated
`group_name`	strata variable name. If NULL and strata exists then using variable
`title_name`	title to use
`surv_est_prefix`	prefix to use in survival estimate names. Default is Time (i.e. Time:5, Time:10,...)
`surv_est_digits`	number of digits to round p values for survival estimates for specified times
`median_est_digits`	number of digits to round p values for Median Survival Estimates
`p_digits`	number of digits to round p values for Log-Rank p value
`latex_output`	will this table go into a latex output (making special charaters latex friendly)
`sig_alpha`	the defined significance level. Default = 0.05
`background`	background color of significant values, or no highlighting if NULL. Default is "yellow"
`...`	other params to pass to `pretty_pvalues` (i.e. `bold` or `italic`)

Value

A tibble with: `Name` (if provided), `Group` (if strata variable in fit), `Level` (if strata variable in fit), `Time:X` (Survival estimates for each time provided), `Median Estimate`. In no strata variable tibble is one row, otherwise nrows = number of strata levels.

Examples

```
# Basic linear model example
set.seed(542542522)
ybin <- sample(0:1, 100, replace = TRUE)
ybin2 <- sample(0:1, 100, replace = TRUE)
y <- rexp(100,.1)
x1 <- factor(sample(LETTERS[1:2],100,replace = TRUE))
x2 <- factor(sample(letters[1:4],100,replace = TRUE))
my_data <- data.frame(y, ybin,ybin2, x1, x2)
Hmisc::label(my_data$x1) <- "X1 Variable"

 # Single runs
run_km_model(strata_in = 'x1', model_data = my_data,
     time_in = 'y', event_in = 'ybin == 1', time_est = NULL)
run_km_model(strata_in = 'x1', model_data = my_data,
     time_in = 'y', event_in = 'ybin == 1', time_est = c(5,10))

# Multiple runs for different variables
library(dplyr)
vars_to_run = c(NA, 'x1', 'x2')
purrr::map_dfr(vars_to_run, run_km_model, model_data = my_data,
     time_in = 'y', event_in = 'ybin == 1', time_est = NULL) %>%
   select(Group, Level, everything())

km_info <- purrr::map_dfr(vars_to_run, run_km_model, model_data = my_data, time_in = 'y',
     event_in = 'ybin == 1', time_est = c(5,10), surv_est_prefix = 'Year',
     title_name = 'Overall Survival') %>%
   select(Group, Level, everything())

km_info2 <- purrr::map_dfr(vars_to_run, run_km_model, model_data = my_data, time_in = 'y',
     event_in = 'ybin2 == 1', time_est = c(5,10), surv_est_prefix = 'Year',
     title_name = 'Cancer Specific Survival') %>%
   select(Group, Level, everything())

library(kableExtra)
options(knitr.kable.NA = '')
kable(bind_rows(km_info, km_info2), escape = F, longtable = F, booktabs = TRUE, linesep = '',
     caption = 'Survival Percentage Estimates at 5 and 10 Years') %>%
  collapse_rows(c(1:2), row_group_label_position = 'stack', headers_to_remove = 1:2)


  # Real World Example
  data(Bladder_Cancer)

  vars_to_run = c(NA, 'Gender', 'Clinical_Stage_Grouped', 'PT0N0', 'Any_Downstaging')

  purrr::map_dfr(vars_to_run, run_km_model, model_data = Bladder_Cancer,
     time_in = 'Survival_Months', event_in = 'Vital_Status == "Dead"', time_est = c(24,60),
       surv_est_prefix = 'Month', p_digits=5) %>%
   select(Group, Level, everything())
```

stat_paste — *Rounds and combines up to three numbers into table friendly presentation*

Description

Takes in up to 3 numeric values, rounds each to a specified digit amount (if numeric), and then combines them accordingly.

Usage

```
stat_paste(stat1, stat2 = NULL, stat3 = NULL, digits = 0,
  trailing_zeros = TRUE, bound_char = c("(", "[", "{", "|"),
  sep = ", ", na_str_out = "---")
```

Arguments

stat1	first statistic to be pasted.
stat2	second statistic to be pasted (optional).
stat3	third statistic to be pasted (optional).
digits	positive integer of length 1 between 0 (default) and 14, giving the amount of digits to round to.
trailing_zeros	logical indicating if trailing zeros should included (i.e. 0.100 instead of 0.1). Note is set to TRUE output is a character vector
bound_char	the character to be used between stat1 and stat2/stat3. Available options are '(' (default), '[', '{', and '—'.
sep	the string to be used between stat2 and stat3. The default is ', '.
na_str_out	the character to replace missing values with.

Details

One value provided - returns a rounded value or the missing character. Two values - returns stat1 (stat2). e.g., mean (sd) Three values - returns stat1 (stat2, stat3). e.g., estimate (95% lower, 95% upper) or median [min, max]

Currently the method does work with string variables, but of course rounding would not be relevant for strings.

Value

string of combined values

Examples

```
stat_paste(5.109293)
stat_paste(NA)
stat_paste(5.109293, 2.145)
stat_paste(5.109293, 1.7645, 8.0345)
stat_paste(NA, NA, NA)
stat_paste(5.109, "p < 0.001", digits = 3)
stat_paste(c(rep(5,5),NA),c(1:5,NA),c(1,NA,2,NA,3,NA),bound_char = '[')
```

```
library(data.table)
data(testData_BAMA)
testData_BAMA [, .(
  median_min_max = stat_paste(
     median(magnitude, na.rm = TRUE),
     min(magnitude, na.rm = TRUE),
     max(magnitude, na.rm = TRUE)
     )), by = .(antigen, visit, group)]
```

`two_samp_bin_test` *Binary (Response) Variable Compared to Binary (Group) Variable Test*

Description

Either Barnard, Fisher's, or Chi-sq test performed for unpaired data and McNemar's test for paired data

Usage

```
two_samp_bin_test(x, y, method = NA, alternative = c("two.sided",
  "less", "greater"), verbose = FALSE)
```

Arguments

`x`	numeric vector (can include NA values).
`y`	vector with only 2 levels (can include NA values unless `method = 'mcnemar'`).
`method`	what test to run ("barnard", "fisher" ,"chi.sq" , "mcnemar"). No default so user must enter one of the four selections
`alternative`	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter. Only "two.sided" available for `method = 'chi.sq' or 'mcnemar'`
`verbose`	a logical variable indicating if warnings and messages should be displayed.
`...`	parameters to pass to wilcox_test or t.test functions. For example the testing direction (`alternative`) in either call or the `var.equal` in the t.test function.

Details

For one sided tests if `y` is a factor variable the level order is respected, otherwise the levels will set to alphabetical order (i.e. if `alternative = less` then testing a ¡ b).

If `method = 'mcnemar'` assumes the first observations of the first group matches the first observation of the second group, and so on. Also if `method = 'mcnemar'` then `y` must have the same number of samples for each level.

Value

p-value for comparing x at the different levels of y.

Examples

```
set.seed(5432322)
x <- c(sample(0:1,10,replace = TRUE, prob = c(.75,.25)),
   sample(0:1,10,replace = TRUE, prob = c(.25,.75)))
y <- c(rep('a', 10), rep('b', 10))
two_samp_bin_test(x,y, method = 'barnard')
two_samp_bin_test(x,y, 'fisher')
two_samp_bin_test(x,y, 'chi.sq')
two_samp_bin_test(x,y, 'mcnemar')
```

`two_samp_cont_test` *Continuous Variable Compared to Binary Variable Test*

Description

Either Wilcox or T-Test Performed, for unpaired or paired data

Usage

```
two_samp_cont_test(x, y, method = c("wilcox", "t.test"),
  paired = FALSE, verbose = FALSE, ...)
```

Arguments

`x`	numeric vector (can include NA values).
`y`	vector with only 2 levels (can include NA values unless `paired = TRUE`).
`method`	what test to run (wilcox or t-test).
`paired`	a logical indicating whether you want a paired test.
`verbose`	a logical variable indicating if warnings and messages should be displayed.
`...`	parameters to pass to wilcox_test or t.test functions. For example the testing direction (`alternative`) in either call or the `var.equal` in the t.test function.

Details

Runs `wilcox_test()` in the coin package, with "exact" distribution and mid-ranks ties method.

For one sided tests if `y` is a factor variable the level order is respected, otherwise the levels will set to alphabetical order (i.e. if `alternative = less` then testing a ¡ b).

If `paired = TRUE` assumes the first observations of the first group matches the first observation of the second group, and so on. Also if `paired = TRUE` then `y` must have the same number of samples for each level.

Value

p-value for comparing x at the different levels of y.

Examples

```
set.seed(5432322)
x <- c(rnorm(10,0,3), rnorm(10,3,3))
y <- c(rep('a', 10), rep('b', 10))
two_samp_cont_test(x = x, y = y, method = 'wilcox', paired = FALSE)
two_samp_cont_test(x = x, y = y, method = 'wilcox', paired = TRUE)
two_samp_cont_test(x = x, y = y, method = 't', paired = FALSE)
two_samp_cont_test(x = x, y = y, method = 't', paired = TRUE)
```

Index