# Package 'MoffittFunctions'

November 14, 2018

**Title** Moffitt Functions

**Version** 0.1.2

**Authors** William (Jimmy) Fulp, Ram Thapa, Zachary Thompson

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

**License** GPL-3

**LazyData** true

**Imports** broom (¿= 0.5.0),
  car,
  coin,
  data.table,
  devtools (¿= 2.0.0),
  dplyr,
  Exact,
  Hmisc,
  kableExtra,
  knitr,
  purrr,
  sessioninfo,
  survival,
  tibble,
  tidyr

**Suggests** testthat,
  rmarkdown

**RoxygenNote** 6.1.1

**Encoding** UTF-8

**VignetteBuilder** knitr

## R topics documented:

---

MoffittFunctions-package

*Moffitt Functions*

---

### Description

Statistical, data processing, and annotation functions for data analysis. Many functions for producing fancey tables in reports

### Author(s)

**Maintainer**: Jimmy Fulp <William.Fulp@moffitt.org>

Authors:

- Ram Thapa <Ram.Thapa@moffitt.org>
- Zachary Thompson <Zachary.Thompson@moffitt.org>

---

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')
```

---

| get_full_name | *Get Full Username from ID* |
|---|---|

---

## Description

For a given ID looks up user name

## Usage

```
get_full_name(id = NULL)
```

## Arguments

id          ID to look full name up. If null (default) looks up ID of current user

## Details

If `id` null, uses system "USERNAME" variable for Windows and "USER" variable for Linux and MACs. Full Name is found in Windows via the `net` command, and via ldap search in Linux and MACs.

## Value

First and Last name associated with ID

## Examples

```
get_full_name()
```

---

get_session_info                  *Get Reproducibility Tables*

---

### Description

Creating tables used at the end of reports, for reproducibility. Most of the information is based off of sessioninfo::session_info()

### Usage

```
get_session_info()
```

### Details

Both tables usually printing with `kable()` at the end of a report

### Value

list of length two, containing dataframe of Software Session Information and dataframe of Software Package Version Information

### Examples

```
my_session_info <- get_session_info()

library(dplyr)

# Simple HTML Display
kableExtra::kable(my_session_info$platform_table, 'html',
      caption = "Reproducibility Software Session Information") %>%
      kableExtra::kable_styling()

kableExtra::kable(my_session_info$packages_table, 'html',
      caption = "Reproducibility Software Package Version Information") %>%
      kableExtra::kable_styling()


# Latex Display
kableExtra::kable(my_session_info$platform_table, 'latex', booktabs = TRUE,
      linesep = '', caption = "Reproducibility Software Session Information") %>%
      kableExtra::kable_styling(font_size = 7)

kableExtra::kable(my_session_info$packages_table, 'latex', booktabs = TRUE,
      linesep = '', caption = "Reproducibility Software Package Version Information") %>%
      kableExtra::kable_styling(font_size = 7)
```

---

| 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(Group, Level, everything())

kableExtra::kable(km_info, 'html', caption = 'Survival Percentage Estimates at 5 and 10 Years') %>%
 kableExtra::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_model_output | *Fancy Table Output of Linear, Logistic, and Cox Models* |
| --- | --- |

---

## Description

pretty_model_output() takes a Linear, Logistic, and Cox model fit object and calculate estimates, odds ratios, or hazard ratios, respectively, with confidence intervals. P values are also produced. For categorical variables with 3+ levels overall Type 3 p values are calculated, in addition to p values comparing to the first level (reference).

## Usage

```
pretty_model_output(fit, model_data, overall_p_test_stat = c("Wald",
```

```
      "LR"), title_name = NULL, conf_level = 0.95, est_digits = 3,
      p_digits = 4, latex_output = FALSE, sig_alpha = 0.05,
      background = "yellow", ...)
```

## Arguments

| | |
|---|---|
| `fit` | lm, glm, or coxph fit (currently only tested on logistic glm fit) |
| `model_data` | data.frame or tibble used to create model fits. Used for capturing variable labels, if they exist |
| `overall_p_test_stat` | |
| | "Wald" (default) or "LR"; the test.statistic to pass through to the test.statistic param in car::Anova. Ignored for lm fits. |
| `title_name` | title to use (will be repeated in first column) |
| `conf_level` | the confidence level required (default is 0.95). |
| `est_digits` | number of digits to round OR or HR to (default is 3) |
| `p_digits` | number of digits to round p values (default is 4) |
| `latex_output` | will this table go into a latex output (making special charaters latex friendly) |
| `sig_alpha` | the defined significance level for highlighting. Default = 0.05 (Only used if latex_output = TRUE) |
| `background` | background color of significant values, or no highlighting if NULL. Default is "yellow" (Only used if latex_output = TRUE) |
| `...` | other params to pass to `pretty_pvalues` (i.e. `bold` or `italic`) (Only used if latex_output = TRUE) |

## Details

Model type is determined by `fit` class, and also family if glm class. If the class is glm and binomial or quasibinomial family, then the output is designed for a Logistic model (i.e. Odd Ratios), if the class is coxph the output is designed for a Cox model (i.e. Harzard Ratios), otherwise the output is designed for a linear model or other model where normal coefficient estimates are displayed.

## Value

A tibble with: `Name` (if provided), `Variable`, `Level`, `Est/OR/HR (95% CI)`, `P Value` (for categorical variables comparing to reference), `Overall P Value` (for categorical variables with 3+ levels).

## Examples

```
# Basic linear model example
set.seed(542542522)
ybin <- sample(0:1, 100, replace = TRUE)
y <- rexp(100,.1)
x1 <- rnorm(100)
x2 <- y + rnorm(100)
x3 <- factor(sample(letters[1:4],100,replace = TRUE))
my_data <- data.frame(y, ybin, x1, x2, x3)

# Linear Regression
```

```
my_fit <- lm(y ~ x1 + x2 + x3, data = my_data)
pretty_model_output(fit = my_fit, model_data = my_data)

# Logistic Regression
my_fit <- glm(ybin ~ x1 + x2 + x3, data = my_data, family = binomial(link = "logit"))
pretty_model_output(fit = my_fit, model_data = my_data)

# Coxph Regression
my_fit <- survival::coxph(survival::Surv(y, ybin) ~ x1 + x2 + x3, data = my_data)
my_pretty_model_output <- pretty_model_output(fit = my_fit, model_data = my_data)

# Printing of Fancy table in HTML
library(dplyr)
kableExtra::kable(my_pretty_model_output, 'html', caption = 'My Table') %>%
    kableExtra::collapse_rows(c(1:2), row_group_label_position = 'stack')

# Real World Examples
data(Bladder_Cancer)
surv_obj <- survival::Surv(Bladder_Cancer$Survival_Months, Bladder_Cancer$Vital_Status == 'Dead')
my_fit <- survival::coxph(surv_obj ~ Gender + Clinical_Stage_Grouped + PT0N0, data = Bladder_Cancer)
my_output <- pretty_model_output(fit = my_fit, model_data = Bladder_Cancer)
kableExtra::kable(my_output, 'html') %>%
    kableExtra::collapse_rows(c(1:2), row_group_label_position = 'stack')
```

---

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_pretty_km_output | *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_pretty_km_output(strata_in = NA, model_data, time_in, event_in,
  event_level = NULL, time_est = NULL, group_name = NULL,
  title_name = NULL, conf_level = 0.95, 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 event status variable. If `event_level` = NULL then this must be the name of a F/T or 0/1 variable, where F or 0 are considered the censored level, respectively |
| `event_level` | event level for event status 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 |
| `conf_level` | the confidence level required (default is 0.95). |
| `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 survival model examples
set.seed(542542522)
ybin <- sample(0:1, 100, replace = TRUE)
ybin2 <- sample(0:1, 100, replace = TRUE)
ybin3 <- sample(c('Dead','Alive'), 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, ybin3, x1, x2)
Hmisc::label(my_data$x1) <- "X1 Variable"

 # Single runs
run_pretty_km_output(strata_in = 'x1', model_data = my_data,
```

```
        time_in = 'y', event_in = 'ybin', time_est = NULL)
run_pretty_km_output(strata_in = 'x1', model_data = my_data,
        time_in = 'y', event_in = 'ybin', time_est = c(5,10))
run_pretty_km_output(strata_in = 'x2', model_data = my_data,
        time_in = 'y', event_in = 'ybin3', event_level = 'Dead', 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_pretty_km_output, model_data = my_data,
        time_in = 'y', event_in = 'ybin', event_level = '0', time_est = NULL) %>%
    select(Group, Level, everything())

km_info <- purrr::map_dfr(vars_to_run, run_pretty_km_output, model_data = my_data, time_in = 'y',
     event_in = 'ybin3', event_level = 'Dead', 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_pretty_km_output, model_data = my_data, time_in = 'y',
        event_in = 'ybin2', time_est = c(5,10), surv_est_prefix = 'Year',
        title_name = 'Cancer Specific Survival') %>%
    select(Group, Level, everything())

options(knitr.kable.NA = '')
kableExtra::kable(bind_rows(km_info, km_info2), escape = F, longtable = F, booktabs = TRUE, linesep = '',
        caption = 'Survival Percentage Estimates at 5 and 10 Years') %>%
 kableExtra::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_pretty_km_output, model_data = Bladder_Cancer,
        time_in = 'Survival_Months', event_in = 'Vital_Status', event_level = 'Dead',
        time_est = c(24,60), surv_est_prefix = 'Month', p_digits=5) %>%
    select(Group, Level, everything())
```

---

run_pretty_model_output

*Wrapper for Pretty Model Output*

---

### Description

Wrapper for pretty_model_output(). This function takes a dataset, along with variables names for x (could be multiple), y, and possibly event status, for model fit.

### Usage

```
run_pretty_model_output(x_in, model_data, y_in, event_in = NULL,
  event_level = NULL, title_name = NULL, fail_if_warning = TRUE,
  conf_level = 0.95, overall_p_test_stat = c("Wald", "LR"),
```

```
    est_digits = 3, p_digits = 4, latex_output = FALSE,
    sig_alpha = 0.05, background = "yellow", verbose = FALSE, ...)
```

## Arguments

| | |
|---|---|
| x_in | name of x variables in model (can be vector of x names) |
| model_data | data.frame or tibble that contains x_in, time_in, and event_in variables |
| y_in | name of outcome measure for logistic and linear model, or name of time component in cox model |
| event_in | name of event status variable. Shouled be left NULL for logistic and linear models. If event_level = NULL then this must be the name of a F/T or 0/1 variable, where F or 0 are considered the censored level, respectively. |
| event_level | outcome variable event level for logistic model, and event status level for cox model. |
| title_name | title to use (will be repeated in first column) |
| fail_if_warning | Should program stop and give useful message if there is a warning message when running model (Default is TRUE) |
| conf_level | the confidence level required (default is 0.95). |
| overall_p_test_stat | |
| | "Wald" (default) or "LR"; the test.statistic to pass through to the test.statistic param in car::Anova. Ignored for lm fits. |
| est_digits | number of digits to round OR or HR to (default is 3) |
| p_digits | number of digits to round p values (default is 4) |
| latex_output | will this table go into a latex output (making special charaters latex friendly) |
| sig_alpha | the defined significance level for highlighting. Default = 0.05 (Only used if latex_output = TRUE) |
| background | background color of significant values, or no highlighting if NULL. Default is "yellow" (Only used if latex_output = TRUE) |
| verbose | a logical variable indicating if warnings and messages should be displayed. Default FALSE. |
| ... | other params to pass to pretty_pvalues (i.e. bold or italic) |

## Details

x_in can be single variable name, or vector of variables to include in the model. All variables must be present in the model_data dataset.

fail_if_warning variable default to TRUE because most warnings should be addressed, such as the "Loglik converged before variable XX; beta may be infinite" warning.

## Value

A tibble with: Name (if provided), Variable, Level, Est/OR/HR (95% CI), P Value (for categorical variables comparing to reference), Overall P Value (for categorical variables with 3+ levels), n/n (event).

## Examples

```
# Basic linear model example
set.seed(542542522)
ybin <- sample(0:1, 100, replace = TRUE)
ybin2 <- sample(c('Male','Female'), 100, replace = TRUE)
ybin3 <- sample(c('Dead','Alive'), 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, ybin3, x1, x2)
Hmisc::label(my_data$x1) <- "X1 Variable"

 # Single runs
run_pretty_model_output(x_in = 'x1', model_data = my_data, y_in = 'y', event_in = 'ybin')
run_pretty_model_output(x_in = 'x1', model_data = my_data, y_in = 'y',
     event_in = 'ybin3', event_level = 'Dead')
run_pretty_model_output(x_in = c('x1','x2'), model_data = my_data, y_in = 'y', event_in = 'ybin')
run_pretty_model_output(x_in = 'x2', model_data = my_data, y_in = 'ybin', event_in = NULL, verbose = T)
run_pretty_model_output(x_in = 'x2', model_data = my_data, y_in = 'y', event_in = NULL)

# Multiple runs for different variables
library(dplyr)
vars_to_run = c('x1', 'x2')
cox_models <- purrr::map_dfr(vars_to_run, run_pretty_model_output, model_data = my_data,
     y_in = 'y', event_in = 'ybin')

kableExtra::kable(cox_models, 'html', caption = 'My Table') %>%
 kableExtra::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('Gender', 'Clinical_Stage_Grouped', 'PT0N0', 'Any_Downstaging')

univariate_output <- purrr::map_dfr(vars_to_run, run_pretty_model_output, model_data = Bladder_Cancer,
      y_in = 'Survival_Months', event_in = 'Vital_Status', event_level = 'Dead')
kableExtra::kable(univariate_output, 'html') %>%
    kableExtra::collapse_rows(c(1:2), row_group_label_position = 'stack', headers_to_remove = 1:2)

multivariable_output <- run_pretty_model_output(vars_to_run, model_data = Bladder_Cancer,
      y_in = 'Survival_Months', event_in = 'Vital_Status', event_level = 'Dead')
kableExtra::kable(multivariable_output, 'html') %>%
    kableExtra::collapse_rows(c(1:2), row_group_label_position = 'stack', headers_to_remove = 1:2)
```

---

| 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 [!is.na(magnitude), .(
  median_min_max = stat_paste(
    median(magnitude), min(magnitude), max(magnitude)
    )), 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 | vector with only 2 levels (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