Package ‘MoffittFunctions’

July 20, 2020

Title Moffitt Functions

Version 1.0.0

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

License GPL-3

URL https://gitlab.moffitt.usf.edu:8000/ReproducibleResearch/MoffittFunctions

BugReports https://gitlab.moffitt.usf.edu:
8000/ReproducibleResearch/MoffittFunctions/-/issues

Imports broom (>= 0.5.0),
car,
coin,
data.table,
devtools (>= 2.0.0),
dplyr,
Exact,
haven (>= 2.0.0),
Hmisc,
kableExtra,
knitr,
purrr,
sessioninfo,
survival,
tibble,
tidyr,
compareGroups,
lme4,
tidyverse,
broom.mixed,
stringr,
coxme,
magrittr,
ggeffects,
sjmisc,
splines,
rlang

Suggests rmarkdown,
testthat

VignetteBuilder knitr

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

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Depends R (>= 3.5.0)

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: Zachary Thompson <`Zachary.Thompson@moffitt.org`>

Authors:

- Jimmy Fulp <`William.Fulp@moffitt.org`>
- Ram Thapa <`Ram.Thapa@moffitt.org`>

See Also

Useful links:

- `https://gitlab.moffitt.usf.edu:8000/ReproducibleResearch/MoffittFunctions`
- Report bugs at `https://gitlab.moffitt.usf.edu:8000/ReproducibleResearch/MoffittFunctions/-/issues`

`Bladder_Cancer` *Bladder_Cancer*

Description

A concise description of the dataset.

Usage

```
data("Bladder_Cancer")
```

Format

A data frame with 166 observations on the following 37 variables.

`PTID` a numeric vector

`Age_At_Diagnosis` a labelled

`Gender` a factor with levels `Female` `Male`

`Race` a factor with levels `White` `Black` `Other`

`Ethnicity` a factor with levels `Hispanic/Latino` `Non-Hispanic/Latino`

`Marital_Status` a factor with levels `Married/living together` `Separated/divorced` `Single (never married)` `Widowed`

`Education_Status` a factor with levels `High School or Less` `College or Some College` `Graduate/Professional Degree` `Unknown`

`Primary_Insurance` a factor with levels `Private` `Medicare` `Medicaid` `Self-paying/uninsured/other`

`Primary_Insurance_More_Cats` a factor with levels `Private` `Medicare` `Medicaid` `Self-paying/other` `Uninsured`

`Surgery_Year` a labelled

`Elix_Sum` a labelled

`Comorbidities` a labelled

`Histology` a factor with levels `Urothelial Neoplasia` `UC w/Squamous Differentiation` `UC w/Glandular Differentiation` `Micropapillary Carcinoma` `Nested Variant` `Plasmacytoid` `Sarcomatoid Carcinoma/Carcinosarcom` `Neuroendocrine Carcinoma`

`Histology_Grouped` a factor with levels `Pure Urothelial Neoplasia` `Mixed Tumors` `Variant Histology`

`Clinical_Stage` a factor with levels `Stage I (<=T1NxMx)` `Stage II (T2NxMx)` `Stage III (T3NxMx)` `Stage IV (T4NxMx)`

`Clinical_Stage_Grouped` a factor with levels `Stage I/II (<=T2NxMx)` `Stage III (T3NxMx)` `Stage IV (T4NxMx)`

`Pathologic_Stage` a factor with levels `Stage 0 (T0/Ta/isN0M0)` `Stage I (T1N0M0)` `Stage II (T2N0M0)` `Stage III (T3N0M0)` `Stage IV (T4N0-3M0-1)`

`Neoadjuvant_Chemo` a factor with levels `Yes`

`Chemo_Cat` a factor with levels `GEM/CIS`

`Adjuvant_Chemo` a labelled

`Cycles` a labelled

`Cycles_cat` a factor with levels `<3` `3` `4+`

`Cycles_cat_4group` a factor with levels `<3` `3` `4` `5+`

`Urinary_Diversion` a factor with levels `Ileal Conduit` `Pouch` `Neobladder` `Cutaneous Ureterostomy` `Sigmond`

`Urinary_Diversion_Grouped` a factor with levels `Ileal Conduit` `Pouch` `Neobladder` `Other`

`PT0N0` a factor with levels `No Completed Response` `Complete Response`

`Any_Downstaging` a factor with levels `No Downstaging` `Downstaging`

`Path_N_Stage` a factor with levels `pNX` `pN0` `pN1` `pN2` `pN3` `Missing/Other`

`Lympho_invasion` a factor with levels `No` `Yes`

`Peri_invasion` a factor with levels `No` `Yes`

`Carc_in_situ` a factor with levels `No` `Yes`

`Positive_Margins` a factor with levels `No` `Yes`

`Vital_Status` a labelled

`Cancer_Specific_Vital_Status` a labelled

`Survival_Days` a labelled

`Survival_Months` a labelled

`Survival_Years` a labelled

Details

more details than the description above

Source

Not sure

References

Don't know

Examples

```
data(Bladder_Cancer)
## maybe str(Bladder_Cancer) ; plot(Bladder_Cancer) ...
```

`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')
```

`exampleData_BAMA` *exampleData_BAMA*

Description

A description of the dataset.

Usage

```
data("exampleData_BAMA")
```

Format

A data frame with 252 observations on the following 6 variables.

`pubID`	a character vector
`group`	a numeric vector
`visitno`	a numeric vector
`antigen`	a character vector
`magnitude`	a numeric vector
`response`	a numeric vector

Details

more details than the description above

Source

Not sure

References

Don't know

Examples

```
data(exampleData_BAMA)
## maybe str(exampleData_BAMA) ; plot(exampleData_BAMA) ...
```

`exampleData_ICS` *exampleData_ICS*

Description

A description of the dataset.

Usage

```
data("exampleData_ICS")
```

Format

A data frame with 306 observations on the following 15 variables.

`pubID` a character vector

`Group` a character vector

`Visit` a numeric vector

`Stim` a character vector

`Parent` a character vector

`Population` a character vector

`Count` a numeric vector

`ParentCount` a numeric vector

`CountBG` a numeric vector

`ParentCountBG` a numeric vector

`PercentCell` a numeric vector

`PercentCellNet` a numeric vector

`response_prob` a numeric vector

`response_fdr_P` a numeric vector

`response` a numeric vector

Details

If necessary, more details than the description above

Source

Not sure

References

Don’t know

Examples

```
data(exampleData_ICS)
## maybe str(exampleData_ICS) ; plot(exampleData_ICS) ...
```

`exampleData_NAb` *exampleData_NAb*

Description

A concise description of the dataset.

Usage

```
data("exampleData_NAb")
```

Format

A data frame with 210 observations on the following 9 variables.

`pubID` a character vector

`group` a numeric vector

`visitno` a numeric vector

`celltype` a character vector

`isolate` a character vector

`titer_mod_50` a numeric vector

`titer_mod_80` a numeric vector

`response_50` a numeric vector

`response_80` a numeric vector

Details

more details than the description above

Source

Not sure

References

Don’t know

Examples

```
data(exampleData_NAb)
## maybe str(exampleData_NAb) ; plot(exampleData_NAb) ...
```

`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()
```

`PDACdata` *PDAC data set*

Description

Pancreatic cancer data set

Usage

```
data("PDACdata")
```

Format

A data frame with 29571 observations on the following 184 variables.

`puf_case_id` a character vector

`puf_facility_id` a character vector

`facility_type_cd` a numeric vector

`facility_location_cd` a numeric vector

`age` a numeric vector

`sex` a numeric vector

`race` a numeric vector

`spanish_hispanic_origin` a numeric vector

`insurance_status` a numeric vector

`med_inc_quar_00` a numeric vector

`no_hsd_quar_00` a numeric vector

`ur_cd_03` a numeric vector

`med_inc_quar_12` a numeric vector

`no_hsd_quar_12` a numeric vector

`ur_cd_13` a numeric vector

`crowfly` a numeric vector

`charlson_recode` a numeric vector

`sequence_number` a character vector

`class_of_case` a numeric vector

`year_of_diagnosis` a numeric vector

`primary_site`	a character vector
`laterality`	a numeric vector
`histology`	a numeric vector
`behavior`	a numeric vector
`grade`	a numeric vector
`diagnostic_confirmation`	a numeric vector
`tumor_size`	a numeric vector
`regional_nodes_positive`	a numeric vector
`regional_nodes_examined`	a numeric vector
`dx_staging_proc_days`	a numeric vector
`rx_summ_dxstg_proc`	a numeric vector
`tnm_clin_t`	a character vector
`tnm_clin_n`	a character vector
`tnm_clin_m`	a character vector
`tnm_clin_stage_group`	a character vector
`tnm_path_t`	a character vector
`tnm_path_n`	a character vector
`tnm_path_m`	a character vector
`tnm_path_stage_group`	a character vector
`tnm_edition_number`	a numeric vector
`analytic_stage_group`	a numeric vector
`cs_mets_at_dx`	a character vector
`cs_mets_eval`	a character vector
`cs_extension`	a character vector
`cs_tumor_sizeext_eval`	a character vector
`cs_mets_dx_bone`	a numeric vector
`cs_mets_dx_brain`	a numeric vector
`cs_mets_dx_liver`	a numeric vector
`cs_mets_dx_lung`	a numeric vector
`lymph_vascular_invasion`	a numeric vector
`cs_sitespecific_factor_1`	a numeric vector
`cs_sitespecific_factor_2`	a numeric vector
`cs_sitespecific_factor_3`	a numeric vector
`cs_sitespecific_factor_4`	a numeric vector
`cs_sitespecific_factor_5`	a numeric vector
`cs_sitespecific_factor_6`	a numeric vector
`cs_sitespecific_factor_7`	a numeric vector
`cs_sitespecific_factor_8`	a numeric vector
`cs_sitespecific_factor_9`	a numeric vector
`cs_sitespecific_factor_10`	a numeric vector

`cs_sitespecific_factor_11` a numeric vector

`cs_sitespecific_factor_12` a numeric vector

`cs_sitespecific_factor_13` a numeric vector

`cs_sitespecific_factor_14` a numeric vector

`cs_sitespecific_factor_15` a numeric vector

`cs_sitespecific_factor_16` a numeric vector

`cs_sitespecific_factor_17` a numeric vector

`cs_sitespecific_factor_18` a numeric vector

`cs_sitespecific_factor_19` a numeric vector

`cs_sitespecific_factor_20` a numeric vector

`cs_sitespecific_factor_21` a numeric vector

`cs_sitespecific_factor_22` a numeric vector

`cs_sitespecific_factor_23` a numeric vector

`cs_sitespecific_factor_24` a numeric vector

`cs_sitespecific_factor_25` a numeric vector

`cs_version_latest` a numeric vector

`dx_rx_started_days` a numeric vector

`dx_surg_started_days` a numeric vector

`dx_defsurg_started_days` a numeric vector

`rx_summ_surg_prim_site` a numeric vector

`rx_hosp_surg_appr_2010` a numeric vector

`rx_summ_surgical_margins` a numeric vector

`rx_summ_scope_reg_ln_sur` a numeric vector

`rx_summ_surg_oth_regdis` a numeric vector

`surg_discharge_days` a numeric vector

`readm_hosp_30_days` a numeric vector

`reason_for_no_surgery` a numeric vector

`dx_rad_started_days` a numeric vector

`rx_summ_radiation` a numeric vector

`rad_location_of_rx` a numeric vector

`rad_treat_vol` a numeric vector

`rad_regional_rx_modality` a numeric vector

`rad_regional_dose_cgy` a numeric vector

`rad_boost_rx_modality` a numeric vector

`rad_boost_dose_cgy` a numeric vector

`rad_num_treat_vol` a numeric vector

`rx_summ_surgrad_seq` a numeric vector

`rad_elapsed_rx_days` a numeric vector

`reason_for_no_radiation` a numeric vector

`dx_systemic_started_days` a numeric vector

dx_chemo_started_days	a numeric vector
rx_summ_chemo	a numeric vector
dx_hormone_started_days	a numeric vector
rx_summ_hormone	a numeric vector
dx_immuno_started_days	a numeric vector
rx_summ_immunotherapy	a numeric vector
rx_summ_trnsplnt_endo	a numeric vector
rx_summ_systemic_sur_seq	a numeric vector
dx_other_started_days	a numeric vector
rx_summ_other	a numeric vector
palliative_care	a numeric vector
rx_summ_treatment_status	a numeric vector
puf_30_day_mort_cd	a numeric vector
puf_90_day_mort_cd	a numeric vector
dx_lastcontact_death_months	a numeric vector
puf_vital_status	a numeric vector
rx_hosp_surg_prim_site	a numeric vector
rx_hosp_chemo	a numeric vector
rx_hosp_immunotherapy	a numeric vector
rx_hosp_hormone	a numeric vector
rx_hosp_other	a numeric vector
puf_mult_source	a numeric vector
reference_date_flag	a numeric vector
rx_summ_scope_reg_ln_2012	a numeric vector
rx_hosp_dxstg_proc	a numeric vector
palliative_care_hosp	a numeric vector
tumor_size_summary	a numeric vector
mets_at_dx_other	a logical vector
mets_at_dx_distant_ln	a logical vector
mets_at_dx_bone	a logical vector
mets_at_dx_brain	a logical vector
mets_at_dx_liver	a logical vector
mets_at_dx_lung	a logical vector
no_hsd_quar_16	a numeric vector
med_inc_quar_16	a numeric vector
medicaid_expn_code	a numeric vector
age70recode	a character vector
sex_recode	a labelled
hispanic_recode	a labelled
race_recode	a labelled

`facility_type_recode`	a labelled
`facility_academic_binary`	a character vector
`sequence_recode`	a character vector
`vital_status`	a labelled
`surgery_recode`	a labelled
`grade_recode`	a labelled
`histology_recode`	a labelled
`clinical_stage_recode`	a labelled
`hospital_count`	a numeric vector
`hospital_volume`	a numeric vector
`surgery_year`	a numeric vector
`volume_quartiles`	a numeric vector
`insurance_recode`	a labelled
`surgery_volume_recode`	a labelled
`surgery_volume_median`	a character vector
`lvi`	a labelled
`path_n_8ed`	a labelled
`path_t`	a character vector
`path_t_recode`	a character vector
`path_t_8ed`	a labelled
`margin_recode`	a labelled
`radiation_seq_recode`	a character vector
`chemo_seq_recode`	a character vector
`chemo_neoadjuvant_recode`	a labelled
`chemo_adjuvant_recode`	a labelled
`age_adjustment`	a character vector
`caci`	a numeric vector
`caci_recode`	a character vector
`caci_categorical`	a labelled
`mortality90day`	a labelled
`volume_high_low`	a labelled
`facility_type_cd0`	a numeric vector
`num_obs_by_fac`	a labelled
`surgery_year12`	a numeric vector
`volumequantiles1`	a factor with levels `(0,2.42]` `(2.42,7.42]` `(7.42,17.4]` `(17.4,79.7]`
`volumequantiles`	a labelled
`ageadjust`	a numeric vector
`CACIcheck`	a numeric vector
`caci3`	a labelled
`chemo_adjuvant_recode0`	a character vector
`chemo_neoadjuvant_recode0`	a character vector
`lvi0`	a character vector
`dx_lastcontact_death_months0`	a numeric vector
`puf_vital_status0`	a numeric vector

Examples

```
data(PDACdata)
## maybe str(PDACdata) ; plot(PDACdata) ...
```

---

percentchange — *Calculate percent change from baseline testing adding a function to MoffittFunctions package*

Description

Calculate percent change from baseline testing adding a function to MoffittFunctions package

Usage

```
percentchange(baseline, nextone)
```

Arguments

`baseline`	Baseline or starting value
`nextone`	Next value. This could be a vector of values.

---

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,
  output_type = NULL
)
```

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
`output_type`	output type, either NULL (default), "latex", or "html" (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_mixed_effects_coxme

Fancy Table Output of Mixed effects COX PH model

Description

pretty_mixed_effects_coxme() Cox mixed effects model using the coxme() function. Hazard ratios are produced 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_mixed_effects_coxme(fit, df)
```

Arguments

`fit`	glmer fit
`df`	is data.frame or tibble used to create model fits. Used for capturing variable labels, if they exist

Value

A data.frame with: `Variable`, `Level`, `HR (95% CI)`, `P Value` (for categorical variables comparing to reference), `Overall P Value` (for categorical variables with 3+ levels).

Examples

```
# Mixed effect cox ph model example - random intercepts
library(magrittr)
library(ggeffects)
library(sjmisc)
library(lme4)
library(splines)
library(coxme)
library(car)
library(survival)
library(broom.mixed)
library(rlang)
library(kableExtra)
data(PDACdata)
set.seed(542542522)
#USE THIS ONE for 3 months to 2 years (24 months)
surv_months_obj <- survival::Surv(time = PDACdata$dx_lastcontact_death_months,
                                  event = PDACdata$puf_vital_status == 0)
   cactfit  <- coxme(surv_months_obj ~
     caci3 + surgery_volume_recode +  (1|puf_facility_id),
     data = PDACdata,
                control=coxme.control(eps = 1e-04, toler.chol = .Machine$double.eps^0.75,
                            iter.max = 200, inner.iter = Quote(max(4, fit0$iter+1)),
                            sparse.calc = NULL,
                            optpar = list(method = "BFGS", control=list(reltol = 1e-3)),
                            refine.df=4, refine.detail=FALSE, refine.method="control",
                            sparse=c(50, .02),
                            varinit=c(.02, .1, .4, .8)^2, corinit = c(0, .3))  )
 cactfitcmetable <-  pretty_mixed_effects_coxme(cactfit, PDACdata)
# for PDF output
#  kable(cactfitcmetable , 'latex', escape = FALSE, longtable = T, booktabs = TRUE, linesep = '',
#  caption = 'Full Mulitvariable Mixed effects Cox Proportional-Hazards Regression Model with
#   interaction for Overall Survival2 to 10 years')
```

---

pretty_mixed_effects_glmer
Fancy Table Output of Mixed effects model from glmer

Description

pretty_mixed_effects_glmer() takes a mixed effects model from glmer() fit object and calculates estimates, odds ratios 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_mixed_effects_glmer(fit, df, rdintterm)
```

Arguments

`fit`	glmer fit
`df`	is data.frame or tibble used to create model fits. Used for capturing variable labels, if they exist
`rdintterm`	random term in the model

Value

A data.frame with: `Variable`, `Level`, `OR (95% CI)`, `P Value` (for categorical variables comparing to reference), `Overall P Value` (for categorical variables with 3+ levels).

Examples

```
# glmer fit ####
library(magrittr)
library(ggeffects)
library(sjmisc)
library(lme4)
library(splines)
library(coxme)
library(car)
library(survival)
library(broom.mixed)
library(rlang)
library(kableExtra)
data(PDACdata)
gmlmerfit <- glmer(I(PDACdata$mortality90day)=="Dead" ~   caci3
                  + surgery_volume_recode
                  + (1|puf_facility_id)
                  ,family = binomial(link = "logit"),
                  data = PDACdata )

# CALL function include fit name, data frame name and random intercept term name in quotes ####
outputtable <- pretty_mixed_effects_glmer(gmlmerfit, PDACdata, "puf_facility_id")

kableExtra::kable(outputtable, 'html', caption = 'My Table') %>%
   kableExtra::collapse_rows(c(1:2), row_group_label_position = 'stack')

# PDF output
#kable(outputtable, 'latex', escape = FALSE, longtable = T, booktabs = TRUE,
# linesep = '', caption = 'Multivariable Mixed Effects Logistic Regression
# Model Results for 90 day mortality with CACI, Volume, and Facility type')%>%
# footnote(number = c('OR are odds of death within 90 days
#  after the most definitive primary site surgery'))
```

`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,
  output_type = NULL,
  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)
`output_type`	output type, either NULL (default), "latex", or "html" (making special charaters latex friendly)
`sig_alpha`	the defined significance level for highlighting. Default = 0.05 (Only used if output_type is not NULL)
`background`	background color of significant values, or no highlighting if NULL. Default is "yellow" (Only used if output_type is not NULL)
`...`	other params to pass to `pretty_pvalues` (i.e. `bold` or `italic`) (Only used if output_type is not NULL)

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)
library(dplyr)
# 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

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_poisson_model_output *Fancy Table Output of Poisson, Logistic, and Cox Models*

Description

pretty_model_output() takes a Poisson (glm family = poisson or quasipoisson), 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_poisson_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,
  output_type = NULL,
  sig_alpha = 0.05,
  background = "yellow",
  ...
)
```

Arguments

`fit`	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)
`output_type`	output type, either NULL (default), "latex", or "html" (making special charaters latex friendly)
`sig_alpha`	the defined significance level for highlighting. Default = 0.05 (Only used if output_type is not NULL)
`background`	background color of significant values, or no highlighting if NULL. Default is "yellow" (Only used if output_type is not NULL)
`...`	other params to pass to `pretty_pvalues` (i.e. `bold` or `italic`) (Only used if output_type is not NULL)

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. Hazard Ratios), otherwise the output is designed for a Poisson model.

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)
library(dplyr)
# 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

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,
  missing_char = "---",
  include_p = FALSE,
  trailing_zeros = TRUE,
  output_type = NULL,
  bold = FALSE,
  italic = FALSE,
  background = NULL,
  sig_alpha = 0.05
)
```

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
`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)

`output_type`	output type, either NULL (default), "latex", or "html" (making special charaters latex friendly)
`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

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"))
```

pretty_pvalues_compareGroups

pretty_pvalues() wrapper for compareGroups table

Description

pretty_pvalues_compareGroups() takes a createTable object created from compareGroups::createTable(), and applies pretty_pvalues() to all p value columns.

Usage

```
pretty_pvalues_compareGroups(
  compareGroups_table_obj,
  digits = 3,
  include_p = FALSE,
  trailing_zeros = TRUE,
  output_type = NULL,
  bold = FALSE,
  italic = FALSE,
  background = NULL,
  sig_alpha = 0.05
)
```

Arguments

`compareGroups_table_obj`	createTable object created from compareGroups::createTable() function, with at least one p value column
`digits`	number of digits to round to; values with zeros past this number of digits are truncated
`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)
`output_type`	output type, either NULL (default), "latex", or "html" (making special charaters latex friendly)
`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

Details

This function is design specifically for input from compareGroups tables. Use `pretty_pvalues` directly when working with normal data.frames and other objects.

Value

createTable object with rounding, formating, and highlighting of any p value column.

Examples

```
data("Bladder_Cancer")
library(compareGroups)
cycles_formula <- as.formula(Cycles_cat ~ Age_At_Diagnosis + Gender + Elix_Sum)
cycles_compare <- compareGroups::compareGroups(cycles_formula, data = Bladder_Cancer)
cycles_table <- compareGroups::createTable(cycles_compare, digits.p = 4, show.p.mul = TRUE)
cycles_table_fancy <- pretty_pvalues_compareGroups(cycles_table, background = 'yellow')
```

```
# Printing createTable object in report
compareGroups::export2latex(cycles_table_fancy, size = 'footnotesize'
, label = 'tab:Variable-of-Interest-Table'
, caption = 'Comparing Variables to Number of Cycles'
, header.labels = c('p.overall' = 'Overall P'), landscape = FALSE)
```

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,
  output_type = NULL,
  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
`output_type`	output type, either NULL (default), "latex", or "html" (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 = FALSE
, longtable = FALSE, 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,
  output_type = NULL,
  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)
`output_type`	output type, either NULL (default), "latex", or "html" (making special charaters latex friendly)
`sig_alpha`	the defined significance level for highlighting. Default = 0.05 (Only used if output_type not NULL)
`background`	background color of significant values, or no highlighting if NULL. Default is "yellow" (Only used if output_type not NULL)
`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"
library(dplyr)
 # 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 = TRUE)
run_pretty_model_output(x_in = 'x2', model_data = my_data, y_in = 'y', event_in = NULL)

# Multiple runs for different variables

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)
```

`run_pretty_poisson_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_poisson_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,
  output_type = NULL,
  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)
`output_type`	output type, either NULL (default), "latex", or "html" (making special charaters latex friendly)
`sig_alpha`	the defined significance level for highlighting. Default = 0.05 (Only used if output_type not NULL)
`background`	background color of significant values, or no highlighting if NULL. Default is "yellow" (Only used if output_type not NULL)
`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"
library(dplyr)
 # 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 = TRUE)
run_pretty_model_output(x_in = 'x2', model_data = my_data, y_in = 'y', event_in = NULL)

# Multiple runs for different variables

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)]
```

---

testData_BAMA *testData_BAMA*

Description

A concise description of the dataset.

Usage

```
data("testData_BAMA")
```

Format

A data frame with 270 observations on the following 6 variables.

`pubID` a character vector

`group` a factor with levels `1` `2` `3`

`antigen` a character vector

`visit` a factor with levels `1` `2`

`magnitude` a numeric vector

`response` a numeric vector

Details

If necessary, more details than the __description__ above

Source

Not sure.

References

Don't know

Examples

```
data(testData_BAMA)
## maybe str(testData_BAMA) ; plot(testData_BAMA) ...
```

`testData_ICS` *testData_ICS*

Description

A concise description of the dataset.

Usage

```
data("testData_ICS")
```

Format

A data frame with 198 observations on the following 15 variables.

`pubID`	a character vector
`Group`	a factor with levels `1` `2` `3`
`Visit`	a factor with levels `1` `2`
`Stim`	a character vector
`Parent`	a character vector
`Population`	a character vector
`Count`	a numeric vector
`ParentCount`	a numeric vector
`CountBG`	a numeric vector
`ParentCountBG`	a numeric vector
`PercentCell`	a numeric vector
`PercentCellNet`	a numeric vector
`response_prob`	a numeric vector
`response_fdr_P`	a numeric vector
`response`	a numeric vector

Details

more details than the description above

Source

Not sure

References

Don't know

Examples

```
data(testData_ICS)
## maybe str(testData_ICS) ; plot(testData_ICS) ...
```

`testData_NAb` *testData_NAb*

Description

A concise description of the dataset.

Usage

```
data("testData_NAb")
```

Format

A data frame with 144 observations on the following 7 variables.

`celltype` a character vector

`pubID` a character vector

`visit` a factor with levels `1` `2`

`group` a numeric vector

`isolate` a character vector

`magnitude` a numeric vector

`response` a numeric vector

Details

more details than the description above

Source

Not sure

References

Don't know

Examples

```
data(testData_NAb)
## maybe str(testData_NAb) ; plot(testData_NAb) ...
```

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)
```