Package ‘tern’

June 27, 2023

Title  Create Common TLGs Used in Clinical Trials
Version  0.8.4
Date  2023-06-26
Description  Table, Listings, and Graphs (TLG) library for common outputs used in clinical trials.
License  Apache License 2.0
URL  https://github.com/insightsengineering/tern
BugReports  https://github.com/insightsengineering/tern/issues
Depends  R (>= 3.6), rtables (>= 0.6.1)
Imports  broom, car, checkmate, cowplot, dplyr, emmeans (>= 1.4.5),forcats (>= 1.0.0), formatters (>= 0.5.0), ggplot2 (>= 3.4.0),grid, gridExtra, gtable, labeling, lifecycle, magrittr,methods, Rdpack, rlang, scales, stats, survival (>= 3.2-13),tibble, tidy, utils
Suggests  knitr, lattice, lubridate, nestcolor (>= 0.1.1), rmarkdown,stringr, testthat (>= 3.0)
VignetteBuilder  knitr
RdMacros  lifecycle, Rdpack
Config/Needs/website  insightsengineering/nesttemplate
Config/testthat/edition  3
Encoding  UTF-8
Language  en-US
LazyData  true
RoxygenNote  7.2.3
Collate  'formatting_functions.R' 'abnormal.R' 'abnormal_by_baseline.R'
          'abnormal_by_marked.R' 'abnormal_by_worst_grade.R'
          'abnormal_by_worst_grade_worsen.R'
          'analyze_colvars_functions.R' 'analyze_functions.R'
          'analyze_vars_in_cols.R' 'argument_convention.R'
          'combination_function.R' 'summarize_variables.R'
NeedsCompilation: no

Author: Joe Zhu [aut, cre],
    Daniel Sabanés Bové [aut],
    Jana Stoilova [aut],
    Heng Wang [aut],
    François Collin [aut],
    Adrian Waddell [aut],
    Pawel Rucki [aut],
    Chendi Liao [aut],
    Jennifer Li [aut],
    F. Hoffmann-La Roche AG [cph, fnd]

Maintainer: Joe Zhu <joe.zhu@roche.com>

Repository: CRAN

Date/Publication: 2023-06-27 11:00:02 UTC
R topics documented:

term-package .................................................. 6
add_rowcounts .................................................. 7
aes_i_label .................................................. 8
analyze_colvars_functions ................................. 8
analyze_functions ........................................... 9
analyze_vars_in_cols ....................................... 10
append_varlabels ........................................... 14
arrange_grobs ............................................... 15
as_rtable .................................................... 16
combination_function ...................................... 17
combine_counts ............................................. 19
combine_groups ............................................. 20
combine_levels ............................................. 21
combine_vectors ........................................... 21
compare_variables ......................................... 22
cox_regression ............................................... 54
cox_regression_inter ...................................... 59
create_afun_compare ....................................... 63
create_afun_summary ....................................... 64
cut_quantile_bins ........................................... 65
d_count_abnormal_by_baseline ......................... 75
d_count_cumulative ......................................... 76
d_count_missed_doses ..................................... 76
d_onco_rsp_label ........................................... 77
d_pkparam .................................................... 78
d_proportion .................................................. 78
d_proportion_diff ........................................... 79
day2month ..................................................... 67
decorate_grob ............................................... 68
decorate_grob_set .......................................... 71
df_explicit_na ............................................... 72
draw_grob ..................................................... 74
d_count_abnormal_by_baseline ......................... 75
d_count_cumulative ......................................... 76
d_count_missed_doses ..................................... 76
d_onco_rsp_label ........................................... 77
d_pkparam .................................................... 78
d_proportion .................................................. 78
d_proportion_diff ........................................... 79
R topics documented:

- d_rsp_subgroups_colvars ........................................... 79
- d_survival_subgroups_colvars ..................................... 80
- d_test_proportion_diff ............................................ 81
- estimate_multinomial_rsp .......................................... 82
- estimate_proportions ............................................... 84
- explicit_na .......................................................... 87
- extract_rsp_biomarkers ............................................ 88
- extract_rsp_subgroups .............................................. 90
- extract_survival_biomarkers ...................................... 92
- extract_survival_subgroups ....................................... 94
- extreme_format ..................................................... 97
- ex_data ............................................................. 98
- fct_collapse_only .................................................. 99
- fct_discard .......................................................... 100
- fct_explicit_na_if ................................................... 101
- fit_coxreg ........................................................... 101
- fit_logistic .......................................................... 104
- fit_rsp_step ........................................................ 106
- fit_survival_step .................................................... 108
- forest_viewport ...................................................... 110
- formatting_functions ............................................... 111
- format_count_fraction ............................................. 112
- format_count_fraction_fixed_dp .................................. 113
- format_extreme_values ............................................. 113
- format_extreme_values_ci ......................................... 114
- format_fraction ..................................................... 115
- format_fraction_fixed_dp ......................................... 116
- format_fraction_threshold ....................................... 117
- format_xx ........................................................... 118
- f_conf_level ......................................................... 119
- f_pval ............................................................... 119
- get_smooths .......................................................... 120
- groups_list_to_df .................................................. 120
- g_forest ............................................................. 121
- g_km ................................................................. 125
- g_lineplot ........................................................... 130
- g_step ............................................................... 134
- g_waterfall ......................................................... 137
- h_adlb_worsen ...................................................... 138
- h_adsl_adlb_merge_using_worst_flag ............................. 140
- h_ancova ............................................................ 141
- h_append_grade_groups ............................................ 142
- h_col_indices ........................................................ 143
- h_count_cumulative ................................................ 144
- h_cox_regression .................................................... 145
- h_data_plot ........................................................ 148
- h_decompose_gg ..................................................... 149
- h_format_row ........................................................ 151
R topics documented:

h_ggkm .......................................................... 152
h_grob_coxph .................................................... 154
h_grob_median_surv ............................................ 155
h_grob_tbl_at_risk ............................................. 156
h_grob_y_annot ................................................. 158
h_g_ipp .......................................................... 159
h_km_layout ...................................................... 161
h_logistic_regression ........................................ 162
h_map_for_count_abnormal .................................... 166
h_odds_ratio ..................................................... 168
hPkparam_sort ................................................. 170
h_proportions ................................................... 170
h_prop_diff ....................................................... 173
h_response_biomarkers_subgroups ......................... 176
h_response_subgroups ......................................... 178
h_split_by_subgroups ........................................ 182
h_split_param ................................................... 183
h_stack_by_baskets ........................................... 184
h_step ........................................................... 186
h_survival_biomarkers_subgroups ......................... 188
h_survival_duration_subgroups ............................. 191
h_tab_one_biomarker ........................................... 195
h_tbl_coxph_pairwise ......................................... 196
h_tbl_median_surv .............................................. 197
h_worsen_counter .............................................. 198
h_xticks ........................................................ 199
individual_patient_plot ...................................... 200
logistic_regression_cols ..................................... 203
logistic_summary_by_flag ..................................... 203
month2day ....................................................... 204
odds_ratio ....................................................... 205
prop_diff ......................................................... 208
prune_occurrences ............................................. 212
reapply_varlabels ............................................. 215
response_biomarkers_subgroups ............................ 215
sas_na .......................................................... 217
score_occurrences ............................................. 218
split_cols_by_groups .......................................... 220
stack_grobs ...................................................... 223
stat_mean_ci ..................................................... 224
stat_mean_pval ................................................ 225
stat_median_ci ................................................... 226
strata_normal_quantile ....................................... 227
summarize_colvars ............................................. 228
summarize_functions .......................................... 230
summarize_logistic ........................................... 230
summarize_num_patients ...................................... 232
summarize_variables ......................................... 236
tern-package

Description

Package to create tables, listings and graphs to analyze clinical trials data.

Author(s)

Maintainer: Joe Zhu <joe.zhu@roche.com>

Authors:

- Daniel Sabanés Bóve <daniel.sabanes.bove@roche.com>
- Jana Stoilova <jana.stoilova@roche.com>
- Heng Wang <wang.heng@gene.com>
- Francois Collin
- Adrian Waddell <adrian.waddell@gene.com>
- Pawel Rucki <pawel.rucki@roche.com>
- Chendi Liao <chendi.liao@roche.com>
- Jennifer Li <li.jing@gene.com>

Other contributors:

- F. Hoffmann-La Roche AG [copyright holder, funder]

See Also

Useful links:

- https://github.com/insightsengineering/tern
- Report bugs at https://github.com/insightsengineering/tern/issues
add_rowcounts

Layout Creating Function to Add Row Total Counts

Description

[Stable]

This works analogously to rtables::add_colcounts() but on the rows. This function is a wrapper for rtables::summarize_row_groups().

Usage

add_rowcounts(lyt)

Arguments

lyt (layout)
input layout where analyses will be added to.

Value

A modified layout where the latest row split labels now have the row-wise total counts (i.e. without column-based subsetting) attached in parentheses.

Note

Row count values are contained in these row count rows but are not displayed so that they are not considered zero rows by default when pruning.

Examples

basic_table() %>%
  split_cols_by("ARM") %>%
  add_colcounts() %>%
  split_rows_by("RACE", split_fun = drop_split_levels) %>%
  add_rowcounts() %>%
  analyze("AGE", afun = list_wrap_x(summary), format = "xx.xx") %>%
  build_table(DM)
aesi_label  
*Labels for Adverse Event Baskets*

**Description**

[Stable]

**Usage**

```r
aesi_label(aesi, scope = NULL)
```

**Arguments**

- **aesi**  
  (character)  
  with standardized MedDRA query name (e.g. SMQzzNAM) or customized query name (e.g. CQzzNAM).

- **scope**  
  (character)  
  with scope of query (e.g. SMQzzSC).

**Value**

A string with the standard label for the AE basket.

**Examples**

```r
adae <- tern_ex_adae

# Standardized query label includes scope.
aesi_label(adae$SMQ01NAM, scope = adae$SMQ01SC)

# Customized query label.
aesi_label(adae$CQ01NAM)
```

---

**analyze_colvars_functions**

*Analyze Functions on Columns*

**Description**

These functions are wrappers of `rtables::analyze_colvars()` which apply corresponding tern statistics functions to add an analysis to a given table layout. In particular, these functions were designed to have the analysis methods split into different columns.
• **analyze_vars_in_cols**: fundamental tabulation of analysis methods onto columns. In other words, the analysis methods are defined in the column space, i.e., they become column labels. By changing the variable vector, the list of functions can be applied on different variables, with the caveat of having the same number of statistical functions.

• **tabulate_rsp_subgroups**: similarly to `analyze_vars_in_cols`, this function combines `analyze_colvars` and `summarize_row_groups` in a compact way to produce standard tables that show analysis methods as columns.

• **tabulate_survival_subgroups**: this function is very similar to the above, but it is used for other tables.

• **analyze_patients_exposure_in_cols**: based only on `analyze_colvars`. It needs `summarize_patients_exposure` to leverage nesting of label rows analysis with `rtables::summarize_row_groups`.

• **summarize_coxreg**: generally based on `rtables::summarize_row_groups`, it behaves similarly to `tabulate_*` functions described above as it is designed to provide specific standard tables that may contain nested structure with a combination of `summarize_row_groups` and `rtables::analyze_colvars`.

See Also

• **summarize_functions** for functions which are wrappers for `rtables::summarize_row_groups`.

• **analyze_functions** for functions which are wrappers for `rtables::analyze`.

---

**analyze_functions  Analyze Functions**

---

**Description**

These functions are wrappers of `rtables::analyze` which apply corresponding tern statistics functions to add an analysis to a given table layout:

• **analyze_num_patients**

• **compare_vars**

• **count_abnormal**

• **count_abnormal_by_baseline**

• **count_abnormal_by_marked**

• **count_abnormal_by_worst_grade**

• **count_cumulative**

• **count_missed_doses**

• **count_occurrences**

• **count_occurrences_by_grade**

• **count_patients_events_in_cols**

• **count_patients_with_event**

• **count_patients_with_flags**
• count_values()
• coxph_pairwise()
• estimate_incidence_rate()
• estimate_multinomial_rsp()
• estimate_odds_ratio()
• estimate_proportion()
• estimate_proportion_diff()
• summarize_ancova()
• summarize_colvars(): even if this function uses rtables::analyze_colvars(), it applies the analysis methods as different rows for one or more variables that are split into different columns. In comparison, analyze_colvars_functions leverage analyze_colvars to have the context split in rows and the analysis methods in columns.
• summarize_change()
• summarize_vars()
• surv_time()
• surv_timepoint()
• test_proportion_diff()

See Also

• summarize_functions for functions which are wrappers for rtables::summarize_row_groups().
• analyze_colvars_functions for functions that are wrappers for rtables::analyze_colvars().

---

analyze_vars_in_cols summary numeric variables in columns

Description

[Experimental]
Layout-creating function which can be used for creating column-wise summary tables. This function sets the analysis methods as column labels and is a wrapper for rtables::analyze_colvars(). It was designed principally for PK tables.

Usage

analyze_vars_in_cols(
  lyt,
  vars,
  ...,
  .stats = c("n", "mean", "sd", "se", "cv", "geom_cv"),
  .labels = c(n = "n", mean = "Mean", sd = "SD", se = "SE", cv = "CV (%)", geom_cv = "CV % Geometric Mean"),
)
analyze_vars_in_cols

    row_labels = NULL,
    do_summarize_row_groups = FALSE,
    split_col_vars = TRUE,
    .indent_mods = NULL,
    nested = TRUE,
    na_level = NULL,
    .formats = NULL
}

Arguments

lyt (layout)
input layout where analyses will be added to.

vars (character)
variable names for the primary analysis variable to be iterated over.

... additional arguments for the lower level functions.

.stats (character)
statistics to select for the table.

.labels (named character)
labels for the statistics (without indent).

row_labels (character)
as this function works in columns space, usual .labels character vector applies on
the column space. You can change the row labels by defining this parameter
to a named character vector with names corresponding to the split values. It
defaults to NULL and if it contains only one string, it will duplicate that as a
row label.

do_summarize_row_groups (flag)
defaults to FALSE and applies the analysis to the current label rows. This is a
wrapper of rtables::summarize_row_groups() and it can accept labelstr
to define row labels. This behavior is not supported as we never need to overload
row labels.

split_col_vars (flag)
defaults to TRUE and puts the analysis results onto the columns. This option
allows you to add multiple instances of this functions, also in a nested fashion,
without adding more splits. This split must happen only one time on a single
layout.

.indent_mods (named integer)
indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified
default behavior. Can be negative.

nested (flag)
whether this layout instruction be applied within the existing layout structure if
possible (TRUE, the default) or as a new top-level element (FALSE). Ignored if it
would nest a split underneath analyses, which is not allowed.

na_level (string)
string used to replace all NA or empty values in the output.
analyze_vars_in_cols

.forms (named character or list)
  formats for the statistics.

Value

A layout object suitable for passing to further layouting functions, or to \texttt{rtables::build_table()}. Adding this function to an rtable layout will summarize the given variables, arrange the output in columns, and add it to the table layout.

Note

This is an experimental implementation of \texttt{rtables::summarize_row_groups()} and \texttt{rtables::analyze_colvars()} that may be subjected to changes as rtables extends its support to more complex analysis pipelines on the column space. For the same reasons, we encourage to read the examples carefully and file issues for cases that differ from them.

Here \texttt{labelstr} behaves differently than usual. If it is not defined (default as NULL), row labels are assigned automatically to the split values in case of \texttt{rtables::analyze_colvars} (\texttt{do_summarize_row_groups} = FALSE, the default), and to the group label for \texttt{do_summarize_row_groups} = TRUE.

See Also

\texttt{summarize_vars()}, \texttt{rtables::analyze_colvars()}. 

Examples

library(dplyr)

# Data preparation
adpp <- tern_ex_adpp %>% h_pkparam_sort()

lyt <- basic_table() %>%
  split_rows_by(var = “STRATA1”, label_pos = “topleft”) %>%
  split_rows_by(  
    var = “SEX”,  
    label_pos = “topleft”,  
    child_label = “hidden”  
  ) %>% # Removes duplicated labels
  analyze_vars_in_cols(vars = “AGE”)
result <- build_table(lyt = lyt, df = adpp)
result

# By selecting just some statistics and ad-hoc labels
lyt <- basic_table() %>%
  split_rows_by(var = “ARM”, label_pos = “topleft”) %>%
  split_rows_by(  
    var = “SEX”,  
    label_pos = “topleft”,  
    child_labels = “hidden”,  
    split_fun = drop_split_levels  
  ) %>%
  analyze_vars_in_cols(
analyze_vars_in_cols

vars = "AGE",
.stats = c("n", "cv", "geom_mean"),
.labels = c(
  n = "aN",
  cv = "aCV",
  geom_mean = "aGeomMean"
)
}

result <- build_table(lyt = lyt, df = adpp)
result

# Changing row labels
lyt <- basic_table() %>%
analyze_vars_in_cols(
  vars = "AGE",
  row_labels = "some custom label"
)
result <- build_table(lyt, df = adpp)
result

# Pharmacokinetic parameters
lyt <- basic_table() %>%
  split_rows_by(
    var = "TLG_DISPLAY",
    split_label = "PK Parameter",
    label_pos = "topleft",
    child_label = "hidden"
  ) %>%
analyze_vars_in_cols(
  vars = "AVAL"
)
result <- build_table(lyt, df = adpp)
result

# Multiple calls (summarize label and analyze underneath)
lyt <- basic_table() %>%
  split_rows_by(
    var = "TLG_DISPLAY",
    split_label = "PK Parameter",
    label_pos = "topleft"
  ) %>%
analyze_vars_in_cols(
  vars = "AVAL",
  do_summarize_row_groups = TRUE # does a summarize level
) %>%
  split_rows_by("SEX",
    child_label = "hidden",
    label_pos = "topleft"
  ) %>%
analyze_vars_in_cols(
  vars = "AVAL",
  split_col_vars = FALSE # avoids re-splitting the columns
)
result <- build_table(lyt, df = adpp)
result

append_varlabels

Add Variable Labels to Top Left Corner in Table

Description

[Stable]

Helper layout creating function to just append the variable labels of a given variables vector from a
given dataset in the top left corner. If a variable label is not found then the variable name itself is
used instead. Multiple variable labels are concatenated with slashes.

Usage

append_varlabels(lyt, df, vars, indent = 0L)

Arguments

lyt  (layout)
input layout where analyses will be added to.
df   (data.frame)
data set containing all analysis variables.
vars (character)
variable names of which the labels are to be looked up in df.
indent (integer)
non-negative number of nested indent space, default to 0L which means no indent.
1L means two spaces indent, 2L means four spaces indent and so on.

Value

A modified layout with the new variable label(s) added to the top-left material.

Note

This is not an optimal implementation of course, since we are using here the data set itself during
the layout creation. When we have a more mature rtables implementation then this will also be
improved or not necessary anymore.
Examples

```r
lyt <- basic_table() %>%
  split_cols_by("ARM") %>%
  add_colcounts() %>%
  split_rows_by("SEX") %>%
  append_varlabels(DM, "SEX") %>%
  analyze("AGE", afun = mean) %>%
  append_varlabels(DM, "AGE", indent = 1)
build_table(lyt, DM)

lyt <- basic_table() %>%
  split_cols_by("ARM") %>%
  split_rows_by("SEX") %>%
  analyze("AGE", afun = mean) %>%
  append_varlabels(DM, c("SEX", "AGE"))
build_table(lyt, DM)
```

---

### arrange_grobs

<table>
<thead>
<tr>
<th>arrange_grobs</th>
<th>Arrange Multiple Grobs</th>
</tr>
</thead>
</table>

#### Description

Arrange grobs as a new grob with n*m (rows*cols) layout.

#### Usage

```r
arrange_grobs(
  ...,
  grobs = list(...),
  ncol = NULL,
  nrow = NULL,
  padding_ht = grid::unit(2, "line"),
  padding_wt = grid::unit(2, "line"),
  vp = NULL,
  gp = NULL,
  name = NULL
)
```

#### Arguments

- `...`: grobs.
- `grobs`: list of grobs.
- `ncol`: number of columns in layout.
- `nrow`: number of rows in layout.
- `padding_ht`: unit of length 1, vertical space between each grob.
as.rtable

Convert to rtable

Description

[Stable]
This is a new generic function to convert objects to rtable tables.
combination_function

Usage

as.rtable(x, ...)

## S3 method for class 'data.frame'
as.rtable(x, format = "xx.xx", ...)

Arguments

x the object which should be converted to an rtable.
...
format the format which should be used for the columns.

Value

An rtables table object. Note that the concrete class will depend on the method used.

Methods (by class)

• as.rtable(data.frame): method for converting data.frame that contain numeric columns to rtable.

Examples

x <- data.frame(
  a = 1:10,
  b = rnorm(10)
)
as.rtable(x)

combination_function  Combination Functions Class

Description

[Stable]

CombinationFunction is an S4 class which extends standard functions. These are special functions that can be combined and negated with the logical operators.

Usage

## S4 method for signature 'CombinationFunction,CombinationFunction'
e1 & e2

## S4 method for signature 'CombinationFunction,CombinationFunction'
e1 | e2

## S4 method for signature 'CombinationFunction'
!x
Arguments

- \( e_1 \) (CombinationFunction)
  - left hand side of logical operator.
- \( e_2 \) (CombinationFunction)
  - right hand side of logical operator.
- \( x \) (CombinationFunction)
  - the function which should be negated.

Value

Returns a logical value indicating whether the left hand side of the equation equals the right hand side.

Functions

- \( e_1 \& e_2 \): Logical "AND" combination of CombinationFunction functions. The resulting object is of the same class, and evaluates the two argument functions. The result is then the "AND" of the two individual results.
- \( e_1 \mid e_2 \): Logical "OR" combination of CombinationFunction functions. The resulting object is of the same class, and evaluates the two argument functions. The result is then the "OR" of the two individual results.
- \(!\) (CombinationFunction): Logical negation of CombinationFunction functions. The resulting object is of the same class, and evaluates the original function. The result is then the opposite of this result.

Examples

```r
higher <- function(a) {
  force(a)
  CombinationFunction(
    function(x) {
      x > a
    }
  )
}

lower <- function(b) {
  force(b)
  CombinationFunction(
    function(x) {
      x < b
    }
  )
}

c1 <- higher(5)
c2 <- lower(10)
c3 <- higher(5) & lower(10)
c3(7)
```
combine_counts

Combine Counts

Description
Simplifies the estimation of column counts, especially when group combination is required.

Usage
combine_counts(fct, groups_list = NULL)

Arguments

fct (factor)
the variable with levels which needs to be grouped.

groups_list (named list of character)
specifies the new group levels via the names and the levels that belong to it in the character vectors that are elements of the list.

Value
A vector of column counts.

See Also
combine_groups()

Examples

ref <- c("A: Drug X", "B: Placebo")
groups <- combine_groups(fct = DM$ARM, ref = ref)

col_counts <- combine_counts(
  fct = DM$ARM,
  groups_list = groups
)

basic_table() %>%
  split_cols_by_groups("ARM", groups) %>%
  add_colcounts() %>%
  summarize-vars("AGE") %>%
  build_table(DM, col_counts = col_counts)

ref <- "A: Drug X"
groups <- combine_groups(fct = DM$ARM, ref = ref)
col_counts <- combine_counts(
  fct = DM$ARM,
  groups_list = groups
)
```r
basic_table() %>%
  split_cols_by_groups("ARM", groups) %>%
  add_colcounts() %>%
  summarize_vars("AGE") %>%
  build_table(DM, col_counts = col_counts)
```

### combine_groups

**Reference and Treatment Group Combination**

**Description**

**[Stable]**

Facilitate the re-combination of groups divided as reference and treatment groups; it helps in arranging groups of columns in the rtables framework and teal modules.

**Usage**

```r
combine_groups(fct, ref = NULL, collapse = "/")
```

**Arguments**

- `fct` *(factor)*  
  the variable with levels which needs to be grouped.
- `ref` *(string)*  
  the reference level(s).
- `collapse` *(string)*  
  a character string to separate `fct` and `ref`.

**Value**

A list with first item `ref` (reference) and second item `trt` (treatment).

**Examples**

```r
groups <- combine_groups(
  fct = DM$ARM,
  ref = c("B: Placebo")
)

basic_table() %>%
  split_cols_by_groups("ARM", groups) %>%
  add_colcounts() %>%
  summarize_vars("AGE") %>%
  build_table(DM)
```
**combine_levels**  
*Combine Factor Levels*

**Description**

[Stable]

Combine specified old factor Levels in a single new level.

**Usage**

```r
combine_levels(x, levels, new_level = paste(levels, collapse = "/"))
```

**Arguments**

- `x`: factor
- `levels`: level names to be combined
- `new_level`: name of new level

**Value**

A factor with the new levels.

**Examples**

```r
x <- factor(letters[1:5], levels = letters[5:1])
combine_levels(x, levels = c("a", "b"))
combine_levels(x, c("e", "b"))
```

**combine_vectors**  
*Combine Two Vectors Element Wise*

**Description**

Combine Two Vectors Element Wise

**Usage**

```r
combine_vectors(x, y)
```

**Arguments**

- `x`: (vector) first vector to combine.
- `y`: (vector) second vector to combine.
Value

A list where each element combines corresponding elements of x and y.

Examples

combine_vectors(1:3, 4:6)
## S3 method for class 'factor'

```r
a_compare(x, .ref_group, .in_ref_col, denom = "n", na.rm = TRUE, ...)
```

## S3 method for class 'character'

```r
a_compare(
  x,
  .ref_group,
  .in_ref_col,
  denom = "n",
  na.rm = TRUE,
  .var,
  verbose = TRUE,
  ...
)
```

## S3 method for class 'logical'

```r
a_compare(x, .ref_group, .in_ref_col, na.rm = TRUE, denom = "n", ...)
```

### compare_vars

```r
compare_vars(  
  lyt,  
  vars,  
  var_labels = vars,  
  nested = TRUE,  
  ...,  
  na_level = NA_character_,  
  show_labels = "default",  
  table_names = vars,  
  .stats = c("n", "mean_sd", "count_fraction", "pval"),  
  .formats = NULL,  
  .labels = NULL,  
  .indent_mods = NULL
)
```

### Arguments

- `x` *(numeric)*
  vector of numbers we want to analyze.
- `.ref_group` *(data.frame or vector)*
  the data corresponding to the reference group.
- `.in_ref_col` *(logical)*
  TRUE when working with the reference level, FALSE otherwise.
- `...` (arguments passed to `s_compare()`).
- `denom` *(string)*
  choice of denominator for factor proportions, can only be `n` (number of values in this row and column intersection).
- `na.rm` *(flag)*
  whether NA values should be removed from `x` prior to analysis.
compare_variables

.. var (string)
single variable name that is passed by rtables when requested by a statistics function.

verbose (logical)
Whether warnings and messages should be printed. Mainly used to print out information about factor casting. Defaults to TRUE.

lyt (layout)
input layout where analyses will be added to.

vars (character)
variable names for the primary analysis variable to be iterated over.

var_labels (character)
character for label.

nested (flag)
whether this layout instruction be applied within the existing layout structure if possible (TRUE, the default) or as a new top-level element (FALSE). Ignored if it would nest a split underneath analyses, which is not allowed.

na_level (string)
string used to replace all NA or empty values in the output.

show_labels (string)
label visibility: one of "default", "visible" and "hidden".

table_names (character)
this can be customized in case that the same vars are analyzed multiple times, to avoid warnings from rtables.

.stats (character)
statistics to select for the table.

.formats (named character or list)
formats for the statistics.

.labels (named character)
labels for the statistics (without indent).

.indent_mods (named vector of integer)
indent modifiers for the labels. Each element of the vector should be a name-value pair with name corresponding to a statistic specified in .stats and value the indentation for that statistic's row label.

Value

- s_compare() returns output of s_summary() and comparisons versus the reference group in the form of p-values.

- a_compare() returns the corresponding list with formatted rtables::CellValue().

- compare_vars() returns a layout object suitable for passing to further layouting functions, or to rtables::build_table(). Adding this function to an rtable layout will add formatted rows containing the statistics from s_compare() to the table layout.
Functions

- `s_compare()`: S3 generic function to produce a comparison summary.
- `s_compare(numeric)`: Method for numeric class. This uses the standard t-test to calculate the p-value.
- `s_compare(factor)`: Method for factor class. This uses the chi-squared test to calculate the p-value.
- `s_compare(character)`: Method for character class. This makes an automatic conversion to factor (with a warning) and then forwards to the method for factors.
- `s_compare(logical)`: Method for logical class. A chi-squared test is used. If missing values are not removed, then they are counted as FALSE.
- `a_compare()`: Formatted analysis function which is used as afun in `compare_vars()`.
- `a_compare(numeric)`: Formatted analysis function method for numeric class.
- `a_compare(factor)`: Formatted analysis function method for factor class.
- `a_compare(character)`: Formatted analysis function method for character class.
- `a_compare(logical)`: Formatted analysis function method for logical class.
- `compare_vars()`: Layout-creating function which can take statistics function arguments and additional format arguments. This function is a wrapper for `rtables::analyze()`.

Note

- For factor variables, denom for factor proportions can only be n since the purpose is to compare proportions between columns, therefore a row-based proportion would not make sense. Proportion based on N_col would be difficult since we use counts for the chi-squared test statistic, therefore missing values should be accounted for as explicit factor levels.
- If factor variables contain NA, these NA values are excluded by default. To include NA values set `na.rm = FALSE` and missing values will be displayed as an NA level. Alternatively, an explicit factor level can be defined for NA values during pre-processing via `df_explicit_na()` - the default `na_level` ("<Missing>") will also be excluded when `na.rm` is set to TRUE.
- For character variables, automatic conversion to factor does not guarantee that the table will be generated correctly. In particular for sparse tables this very likely can fail. Therefore it is always better to manually convert character variables to factors during pre-processing.
- For `compare_vars()`, the column split must define a reference group via `ref_group` so that the comparison is well defined.

See Also

Relevant constructor function `create_afun_compare()`, and `s_summary()` which is used internally to compute a summary within `s_compare()`.

Examples

```r
# 's_compare.numeric'

## Usual case where both this and the reference group vector have more than 1 value.
s_compare(rnorm(10, 5, 1), .ref_group = rnorm(5, -5, 1), .in_ref_col = FALSE)
```
## If one group has not more than 1 value, then p-value is not calculated.
s_compare(rnorm(10, 5, 1), .ref_group = 1, .in_ref_col = FALSE)

## Empty numeric does not fail, it returns NA-filled items and no p-value.
s_compare(numeric(), .ref_group = numeric(), .in_ref_col = FALSE)

# `s_compare.factor`

## Basic usage:
x <- factor(c("a", "a", "b", "c", "a"))
y <- factor(c("a", "b", "c"))
s_compare(x = x, .ref_group = y, .in_ref_col = FALSE)

## Management of NA values.
x <- explicit_na(factor(c("a", "a", "b", "c", "a", NA, NA)))
y <- explicit_na(factor(c("a", "b", "c", NA)))
s_compare(x = x, .ref_group = y, .in_ref_col = FALSE, na.rm = TRUE)
s_compare(x = x, .ref_group = y, .in_ref_col = FALSE, na.rm = FALSE)

# `s_compare.character`

## Basic usage:
x <- c("a", "a", "b", "c", "a")
y <- c("a", "b", "c")
s_compare(x, .ref_group = y, .in_ref_col = FALSE, .var = "x", verbose = FALSE)

## Note that missing values handling can make a large difference:
x <- c("a", "a", "b", "c", "a", NA)
y <- c("a", "b", "c", rep(NA, 20))
s_compare(x,
  .ref_group = y, .in_ref_col = FALSE,  
  .var = "x", verbose = FALSE)
)s_compare(x,
  .ref_group = y, .in_ref_col = FALSE, .var = "x",  
  na.rm = FALSE, verbose = FALSE
)

# `s_compare.logical`

## Basic usage:
x <- c(TRUE, FALSE, TRUE, TRUE)
y <- c(FALSE, FALSE, TRUE)
s_compare(x, .ref_group = y, .in_ref_col = FALSE)

## Management of NA values.
x <- c(NA, TRUE, FALSE)
y <- c(NA, NA, NA, NA, FALSE)
s_compare(x, .ref_group = y, .in_ref_col = FALSE, na.rm = TRUE)
s_compare(x, .ref_group = y, .in_ref_col = FALSE, na.rm = FALSE)

# `a_compare.numeric`
a_compare(
  rnorm(10, 5, 1),
  .ref_group = rnorm(20, -5, 1),
  .in_ref_col = FALSE,
  .var = "bla"
)

# `a_compare.factor`
# We need to ungroup `count` and `count_fraction` first so that the `rtables` formatting # functions can be applied correctly.
afun <- make_afun(
  getS3method("a_compare", "factor"),
  .ungroup_stats = c("count", "count_fraction")
)
x <- factor(c("a", "a", "b", "c", "a"))
y <- factor(c("a", "a", "b", "c"))
afun(x, .ref_group = y, .in_ref_col = FALSE)

# `a_compare.character`
afun <- make_afun(
  getS3method("a_compare", "character"),
  .ungroup_stats = c("count", "count_fraction")
)
x <- c("A", "B", "A", "C")
y <- c("B", "A", "C")
afun(x, .ref_group = y, .in_ref_col = FALSE, .var = "x", verbose = FALSE)

# `a_compare.logical`
afun <- make_afun(
  getS3method("a_compare", "logical")
)
x <- c(TRUE, FALSE, FALSE, TRUE, TRUE)
y <- c(TRUE, FALSE)
afun(x, .ref_group = y, .in_ref_col = FALSE)

# `compare_vars()` in `rtables` pipelines

## Default output within a `rtables` pipeline.
lyt <- basic_table() %>%
  split_cols_by("ARMCD", ref_group = "ARM B") %>%
  compare_vars(c("AGE", "SEX"))
build_table(lyt, tern_ex_adsl)

## Select and format statistics output.
lyt <- basic_table() %>%
  split_cols_by("ARMCD", ref_group = "ARM C") %>%
  compare_vars(
    vars = "AGE",
    .stats = c("mean_sd", "pval"),
    .formats = c(mean_sd = "xx.x, xx.x"),
    .labels = c(mean_sd = "Mean, SD")
  )
build_table(lyt, df = tern_ex_adsl)
control_coxph  
*Control Function for CoxPH Model*

**Description**

[Stable]

This is an auxiliary function for controlling arguments for CoxPH model, typically used internally to specify details of CoxPH model for `s_coxph_pairwise()`. `conf_level` refers to Hazard Ratio estimation.

**Usage**

```r
control_coxph(
  pval_method = c("log-rank", "wald", "likelihood"),
  ties = c("efron", "breslow", "exact"),
  conf_level = 0.95
)
```

**Arguments**

- `pval_method` *(string)*
  p-value method for testing hazard ratio = 1. Default method is "log-rank", can also be set to "wald" or "likelihood".

- `ties` *(string)*
  specifying the method for tie handling. Default is "efron", can also be set to "breslow" or "exact". See more in `survival::coxph()`.

- `conf_level` *(proportion)*
  confidence level of the interval.

**Value**

A list of components with the same names as the arguments

---

control_coxreg  
*Controls for Cox Regression*

**Description**

[Stable]

Sets a list of parameters for Cox regression fit. Used internally.
control_incidence_rate

Usage

control_coxreg(
  pval_method = c("wald", "likelihood"),
  ties = c("exact", "efron", "breslow"),
  conf_level = 0.95,
  interaction = FALSE
)

Arguments

pval_method (string)
the method used for estimation of p.values; wald (default) or likelihood.

ties (string)
among exact (equivalent to DISCRETE in SAS), efron and breslow, see survival::coxph().
Note: there is no equivalent of SAS EXACT method in R.

conf_level (proportion)
confidence level of the interval.

interaction (flag)
if TRUE, the model includes the interaction between the studied treatment and candidate covariate. Note that for univariate models without treatment arm, and multivariate models, no interaction can be used so that this needs to be FALSE.

Value

A list of items with names corresponding to the arguments.

See Also

fit_coxreg_univar() and fit_coxreg_multivar().

Examples

control_coxreg()

Control function for incidence rate

[Stable]
This is an auxiliary function for controlling arguments for the incidence rate, used internally to specify details in s_incidence_rate().
Usage

control_incidence_rate(
    conf_level = 0.95,
    conf_type = c("normal", "normal_log", "exact", "byar"),
    input_time_unit = c("year", "day", "week", "month"),
    num_pt_year = 100,
    time_unit_input = lifecycle::deprecated(),
    time_unit_output = lifecycle::deprecated()
)

Arguments

conf_level (proportion)
    confidence level of the interval.

conf_type (string)
    normal (default), normal_log, exact, or byar for confidence interval type.

input_time_unit (string)
    day, week, month, or year (default) indicating time unit for data input.

num_pt_year (numeric)
    number of patient-years to use when calculating adverse event rates.

time_unit_input [Deprecated] Please use the input_time_unit argument instead.

time_unit_output [Deprecated] Please use the num_pt_year argument instead.

Value

A list of components with the same names as the arguments.

See Also

incidence_rate

Examples

control_incidence_rate(0.9, "exact", "month", 100)

control_lineplot_vars  Control Function for g_lineplot Function

Description

[Stable]

Default values for variables parameter in g_lineplot function. A variable’s default value can be overwritten for any variable.
control_logistic

Usage

```r
control_lineplot_vars(
  x = "AVISIT",
  y = "AVAL",
  strata = "ARM",
  paramcd = "PARAMCD",
  y_unit = "AVALU"
)
```

Arguments

- `x` (character): x variable name.
- `y` (character): y variable name.
- `strata` (character or NA): strata variable name.
- `paramcd` (character or NA): paramcd variable name.
- `y_unit` (character or NA): y_unit variable name.

Value

A named character vector of variable names.

Examples

```r
control_lineplot_vars()
control_lineplot_vars(strata = NA)
```

control_logistic

Control Function for Logistic Regression Model Fitting

Description

[Stable]

This is an auxiliary function for controlling arguments for logistic regression models. `conf_level` refers to the confidence level used for the Odds Ratio CIs.

Usage

```r
control_logistic(response_definition = "response", conf_level = 0.95)
```
control_step

Control Function for Subgroup Treatment Effect Pattern (STEP) Calculations

Description

[Stable]

This is an auxiliary function for controlling arguments for STEP calculations.

Usage

control_step(
  biomarker = NULL,
  use_percentile = TRUE,
  bandwidth,
  degree = 0L,
  num_points = 39L
)
Arguments

biomarker (numeric or NULL)
optional provision of the numeric biomarker variable, which could be used to infer bandwidth, see below.

use_percentile (flag)
if TRUE, the running windows are created according to quantiles rather than actual values, i.e. the bandwidth refers to the percentage of data covered in each window. Suggest TRUE if the biomarker variable is not uniformly distributed.

bandwidth (number or NULL)
indicating the bandwidth of each window. Depending on the argument use_percentile, it can be either the length of actual-value windows on the real biomarker scale, or percentage windows. If use_percentile = TRUE, it should be a number between 0 and 1. If NULL, treat the bandwidth to be infinity, which means only one global model will be fitted. By default, 0.25 is used for percentage windows and one quarter of the range of the biomarker variable for actual-value windows.

degree (count)
the degree of polynomial function of the biomarker as an interaction term with the treatment arm fitted at each window. If 0 (default), then the biomarker variable is not included in the model fitted in each biomarker window.

num_points (count)
the number of points at which the hazard ratios are estimated. The smallest number is 2.

Value

A list of components with the same names as the arguments, except biomarker which is just used to calculate the bandwidth in case that actual biomarker windows are requested.

Examples

# Provide biomarker values and request actual values to be used, # so that bandwidth is chosen from range.
control_step(biomarker = 1:10, use_percentile = FALSE)

# Use a global model with quadratic biomarker interaction term.
control_step(bandwidth = NULL, degree = 2)

# Reduce number of points to be used.
control_step(num_points = 10)
Description

[Stable]

Sets a list of parameters for summaries of descriptive statistics. Typically used internally to specify
details for `s_summary()`.

Usage

```r
control_summarize_vars(
  conf_level = 0.95,
  quantiles = c(0.25, 0.75),
  quantile_type = 2,
  test_mean = 0
)
```

Arguments

- `conf_level` (proportion)
  confidence level of the interval.
- `quantiles` (numeric)
  of length two to specify the quantiles to calculate.
- `quantile_type` (numeric)
  between 1 and 9 selecting quantile algorithms to be used. Default is set to
  2 as this matches the default quantile algorithm in SAS proc univariate
  set by QNTLDEF=5. This differs from R's default. See more about type in
  `stats::quantile()`.
- `test_mean` (numeric)
  to test against the mean under the null hypothesis when calculating p-value.

Value

A list of components with the same names as the arguments.

---

control_surv_time  Control Function for survfit Model for Survival Time

Description

[Stable]

This is an auxiliary function for controlling arguments for survfit model, typically used internally to specify
details of survfit model for `s_surv_time()`. conf_level refers to survival time estimation.
control_surv_timepoint

Usage

control_surv_timepoint(
  conf_level = 0.95,
  conf_type = c("plain", "log", "log-log"),
  quantiles = c(0.25, 0.75)
)

Arguments

conf_level  (proportion)
  confidence level of the interval.
conf_type    (string)
  confidence interval type. Options are "plain" (default), "log", "log-log", see
  more in survival::survfit(). Note option "none" is no longer supported.
quantiles   (numeric)
  of length two to specify the quantiles of survival time.

Value

A list of components with the same names as the arguments

Description

[Stable]

This is an auxiliary function for controlling arguments for survfit model, typically used internally
to specify details of survfit model for s_surv_timepoint(). conf_level refers to patient risk
estimation at a time point.

Usage

control_surv_timepoint(
  conf_level = 0.95,
  conf_type = c("plain", "log", "log-log")
)

Arguments

conf_level  (proportion)
  confidence level of the interval.
conf_type    (string)
  confidence interval type. Options are "plain" (default), "log", "log-log", see
  more in survival::survfit(). Note option "none" is no longer supported.
Value

A list of components with the same names as the arguments

count_occurrences  Occurrence Counts

Description

[Stable]

Functions for analyzing frequencies and fractions of occurrences for patients with occurrence data. Primary analysis variables are the dictionary terms. All occurrences are counted for total counts. Multiple occurrences within patient at the lowest term level displayed in the table are counted only once.

Usage

s_count_occurrences(
  df,
  denom = c("N_col", "n"),
  .N_col,
  .df_row,
  drop = TRUE,
  .var = "MHDECOD",
  id = "USUBJID"
)

a_count_occurrences(
  df,
  denom = c("N_col", "n"),
  .N_col,
  .df_row,
  drop = TRUE,
  .var = "MHDECOD",
  id = "USUBJID"
)

count_occurrences(
  lyt,
  vars,
  var_labels = vars,
  show_labels = "hidden",
  ...
  table_names = vars,
  .stats = "count_fraction",
  .formats = NULL,
  .labels = NULL,
Arguments

df (data.frame)
data set containing all analysis variables.

denom (string)
choice of denominator for patient proportions. Can be:
  • N_col: total number of patients in this column across rows
  • n: number of patients with any occurrences

.N_col (count)
row-wise N (row group count) for the group of observations being analyzed (i.e. with no column-based subsetting) that is passed by rtables.

.df_row (data.frame)
data frame across all of the columns for the given row split.

drop (flag)
should non appearing occurrence levels be dropped from the resulting table. Note that in that case the remaining occurrence levels in the table are sorted alphabetically.

.var (string)
single variable name that is passed by rtables when requested by a statistics function.

.id (string)
subject variable name.

.lyt (layout)
input layout where analyses will be added to.

.vars (character)
variable names for the primary analysis variable to be iterated over.

.var_labels (character)
character for label.

.show_labels (string)
label visibility: one of "default", "visible" and "hidden".

... additional arguments for the lower level functions.

.table_names (character)
this can be customized in case that the same vars are analyzed multiple times, to avoid warnings from rtables.

.stats (character)
statistics to select for the table.

.formats (named character or list)
formats for the statistics.

.labels (named character)
labels for the statistics (without indent).

.indent_mods (named integer)
indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.
Value

• `s_count_occurrences()` returns a list with:
  – `count`: list of counts with one element per occurrence.
  – `count_fraction`: list of counts and fractions with one element per occurrence.
  – `fraction`: list of numerators and denominators with one element per occurrence.

• `a_count_occurrences()` returns the corresponding list with formatted `rtables::CellValue()`.

• `count_occurrences()` returns a layout object suitable for passing to further layouting functions, or to `rtables::build_table()`. Adding this function to an `rtable` layout will add formatted rows containing the statistics from `s_count_occurrences()` to the table layout.

Functions

• `s_count_occurrences()`: Statistics function which counts number of patients that report an occurrence.

• `a_count_occurrences()`: Formatted analysis function which is used as `afun` in `count_occurrences()`.

• `count_occurrences()`: Layout-creating function which can take statistics function arguments and additional format arguments. This function is a wrapper for `rtables::analyze()`.

Note

By default, occurrences which don’t appear in a given row split are dropped from the table and the occurrences in the table are sorted alphabetically per row split. Therefore, the corresponding layout needs to use `split_fun = drop_split_levels` in the `split_rows_by` calls. Use `drop = FALSE` if you would like to show all occurrences.

Examples

def <- data.frame(  
  USUBJID = as.character(c(1, 1, 2, 4, 4, 4)),  
  MHDECOD = c("MH1", "MH2", "MH1", "MH1", "MH1", "MH3")  
  )

N_per_col <- 4L

# Count unique occurrences per subject.
s_count_occurrences(  
  df,  
  .N_col = N_per_col,  
  .df_row = df,  
  .var = "MHDECOD",  
  id = "USUBJID"  
  )

# We need to ungroup `count_fraction` first so that the `rtables` formatting  
# function `format_count_fraction()` can be applied correctly.
afun <- make_afun(a_count_occurrences, .ungroup_stats = c("count", "count_fraction", "fraction"))
afun(  
  df,
...
library(dplyr)

df <- data.frame(
  USUBJID = as.character(c(1, 1, 2, 4, 4, 4, 6, 6, 6, 7, 7, 8)),
  ARM = rep(c("A", "B"), each = 6)
)

df_adsl <- df %>%
  select(USUBJID, ARM) %>%
  unique()

# Create table layout
lyt <- basic_table() %>%
  split_cols_by("ARM") %>%
  add_colcounts() %>%
  count_occurrences(vars = "MHDECOD", .stats = c("count_fraction"))

# Apply table layout to data and produce `rtable` object
lyt %>%
  build_table(df, alt_counts_df = df_adsl) %>%
  prune_table()
count_occurrences_by_grade

`a_count_occurrences_by_grade(
  df,
  .var,
  .N_col,
  id = "USUBJID",
  grade_groups = list(),
  remove_single = TRUE,
  labelstr = ""
)`

`count_occurrences_by_grade(
  lyt,
  var,
  var_labels = var,
  show_labels = "default",
  ...
)`

`summarize_occurrences_by_grade(
  lyt,
  var,
  ...
)`

**Arguments**

- `df` *(data.frame)*
  - data set containing all analysis variables.

- `.var, var` *(string)*
  - single variable name that is passed by rtables when requested by a statistics function.
count_occurrences_by_grade

.N_col (count)
row-wise N (row group count) for the group of observations being analyzed (i.e. with no column-based subsetting) that is passed by rtables.

id (string)
subject variable name.

grade_groups (named list of character)
containing groupings of grades.

remove_single (logical)
TRUE to not include the elements of one-element grade groups in the the output list; in this case only the grade groups names will be included in the output.

labelstr (character)
label of the level of the parent split currently being summarized (must be present as second argument in Content Row Functions). See rtables::summarize_row_groups() for more information.

lyt (layout)
input layout where analyses will be added to.

var_labels (character)
labels to show in the result table.

show_labels (string)
label visibility: one of "default", "visible" and "hidden".

... additional arguments for the lower level functions.

table_names (character)
this can be customized in case that the same vars are analyzed multiple times, to avoid warnings from rtables.

.stats (character)
statistics to select for the table.

.formats (named character or list)
formats for the statistics.

.indent_mods (named integer)
indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.

.labels (named character)
labels for the statistics (without indent).

Value

- s_count_occurrences_by_grade() returns a list of counts and fractions with one element per grade level or grade level grouping.

- a_count_occurrences_by_grade() returns the corresponding list with formatted rtables::CellValue().

- count_occurrences_by_grade() returns a layout object suitable for passing to further layouting functions, or to rtables::build_table(). Adding this function to an rtable layout will add formatted rows containing the statistics from s_count_occurrences_by_grade() to the table layout.
• summarize_occurrences_by_grade() returns a layout object suitable for passing to further layouting functions, or to \texttt{rtables::build_table()}. Adding this function to an rtable layout will add formatted content rows containing the statistics from \texttt{s_count_occurrences_by_grade()} to the table layout.

Functions

• \texttt{s_count_occurrences_by_grade()}: Statistics function which counts the number of patients by highest grade.
• \texttt{a_count_occurrences_by_grade()}: Formatted analysis function which is used as \texttt{afun} in \texttt{count_occurrences_by_grade()}.
• \texttt{count_occurrences_by_grade()}: Layout-creating function which can take statistics function arguments and additional format arguments. This function is a wrapper for \texttt{rtables::analyze()}.
• \texttt{summarize_occurrences_by_grade()}: Layout-creating function which can take content function arguments and additional format arguments. This function is a wrapper for \texttt{rtables::summarize_row_groups()}.

See Also

Relevant helper function \texttt{h_append_grade_groups()}.

Examples

```r
library(dplyr)

df <- data.frame(
  USUBJID = as.character(c(1:6, 1)),
  AETOXGR = factor(c(1, 2, 3, 4, 1, 2, 3), levels = c(1:5)),
  AESEV = factor(
    x = c("MILD", "MODERATE", "SEVERE", "MILD", "MILD", "MODERATE", "SEVERE"),
    levels = c("MILD", "MODERATE", "SEVERE")
  ),
  stringsAsFactors = FALSE
)

df_ads1 <- df %>%
  select(USUBJID, ARM) %>%
  unique()

s_count_occurrences_by_grade(
  df,
  .N_col = 10L,
  .var = "AETOXGR",
  id = "USUBJID",
  grade_groups = list("ANY" = levels(df$AETOXGR))
)
```

# We need to ungroup `count_fraction` first so that the `rtables` formatting
# function `format_count_fraction()` can be applied correctly.

```r
afun <- make_afun(a_count_occurrences_by_grade, .ungroup_stats = "count_fraction")
afun(
  df,
  .N_col = 10L,
)```
# Layout creating function with custom format.
basic_table() %>%
  split_cols_by("ARM") %>%
  add_colcounts() %>%
  count_occurrences_by_grade(
    .var = "AESEV",
    .formats = c("count_fraction" = "xx.xx (xx.xx%)")
  ) %>%
  build_table(df, alt_counts_df = df_adsl)

# Define additional grade groupings.
grade_groups <- list(
  "-Any-" = c("1", "2", "3", "4", "5"),
  "Grade 1-2" = c("1", "2"),
  "Grade 3-5" = c("3", "4", "5")
)

# Layout creating function with custom format.
basic_table() %>%
  split_cols_by("ARM") %>%
  add_colcounts() %>%
  count_occurrences_by_grade(
    .var = "AESEV",
    grade_groups = grade_groups
  ) %>%
  build_table(df, alt_counts_df = df_adsl)

# Layout creating function with custom format.
basic_table() %>%
  add_colcounts() %>%
  split_rows_by("ARM", child_labels = "visible", nested = TRUE) %>%
  summarize_occurrences_by_grade(
    .var = "AESEV",
    .formats = c("count_fraction" = "xx.xx (xx.xx%)")
  ) %>%
  build_table(df, alt_counts_df = df_adsl)
count_patients_with_event

Count the Number of Patients with a Particular Event

Description

[Stable]

The primary analysis variable `.var` denotes the unique patient identifier.

Usage

s_count_patients_with_event(
  df,
  .var,
  filters,
  .N_col,
  .N_row,
  denom = c("n", "N_row", "N_col")
)

a_count_patients_with_event(
  df,
  .var,
  filters,
  .N_col,
  .N_row,
  denom = c("n", "N_row", "N_col")
)

count_patients_with_event(
  lyt,
  vars,
  ...
  table_names = vars,
  .stats = "count_fraction",
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)

Arguments

- `df` (data.frame): data set containing all analysis variables.
- `.var` (character): name of the column that contains the unique identifier.
filters (character)
a character vector specifying the column names and flag variables to be used for counting the number of unique identifiers satisfying such conditions. Multiple column names and flags are accepted in this format c("column_name1" = "flag1", "column_name2" = "flag2"). Note that only equality is being accepted as condition.

.N_col (count)
row-wise N (row group count) for the group of observations being analyzed (i.e. with no column-based subsetting) that is passed by rtables.

.N_row (count)
column-wise N (column count) for the full column that is passed by rtables.

denom (string)
choice of denominator for proportion. Options are:
  • n: number of values in this row and column intersection.
  • .N_row: total number of values in this row across columns.
  • .N_col: total number of values in this column across rows.

lyt (layout)
input layout where analyses will be added to.

cols (character)
variable names for the primary analysis variable to be iterated over.

... additional arguments for the lower level functions.

table_names (character)
this can be customized in case that the same cols are analyzed multiple times, to avoid warnings from rtables.

.stats (character)
statistics to select for the table.

.formats (named character or list)
formats for the statistics.

.labels (named character)
labels for the statistics (without indent).

.indent_mods (named integer)
indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.

Value

  • s_count_patients_with_event() returns the count and fraction of unique identifiers with the defined event.

  • a_count_patients_with_event() returns the corresponding list with formatted rtables::CellValue().

  • count_patients_with_event() returns a layout object suitable for passing to further layouting functions, or to rtables::build_table(). Adding this function to an rtable layout will add formatted rows containing the statistics from s_count_patients_with_event() to the table layout.
Functions

- `s_count_patients_with_event()`: Statistics function which counts the number of patients for which the defined event has occurred.
- `a_count_patients_with_event()`: Formatted analysis function which is used as `afun` in `count_patients_with_event()`.
- `count_patients_with_event()`: Layout-creating function which can take statistics function arguments and additional format arguments. This function is a wrapper for `rtables::analyze()`.

See Also

`count_patients_with_flags`

Examples

```r
library(dplyr)

# `s_count_patients_with_event()

s_count_patients_with_event(
  tern_ex_adae,
  .var = "SUBJID",
  filters = c("TRTEMFL" = "Y")
)

s_count_patients_with_event(
  tern_ex_adae,
  .var = "SUBJID",
  filters = c("TRTEMFL" = "Y", "AEOUT" = "FATAL")
)

s_count_patients_with_event(
  tern_ex_adae,
  .var = "SUBJID",
  filters = c("TRTEMFL" = "Y", "AEOUT" = "FATAL"),
  denom = "N_col",
  N_col = 456
)

# `a_count_patients_with_event()

a_count_patients_with_event(
  tern_ex_adae,
  .var = "SUBJID",
  filters = c("TRTEMFL" = "Y"),
  .N_col = 100,
  .N_row = 100
)

# `count_patients_with_event()

lyt <- basic_table() %>%
  split_cols_by("ARM") %>%
  add_colcounts() %>
```

count_patients_with_flags

Count the Number of Patients with Particular Flags

Description

[Stable]
The primary analysis variable .var denotes the unique patient identifier.

Usage

s_count_patients_with_flags(
    df,
    .var,
    flag_variables,
    .N_col,
    .N_row,
    denom = c("n", "N_row", "N_col")
)

a_count_patients_with_flags(

```r
count_values("STUDYID",
  values = "AB12345",
  .stats = "count",
  .labels = c(count = "Total AEs")
) %>%
count_patients_with_event("SUBJID",
  filters = c("TRTEMFL" = "Y"),
  .labels = c(count_fraction = "Total number of patients with at least one adverse event"),
  table_names = "tbl_all"
) %>%
count_patients_with_event("SUBJID",
  filters = c("TRTEMFL" = "Y", "AEOU" = "FATAL"),
  .labels = c(count_fraction = "Total number of patients with fatal AEs"),
  table_names = "tbl_fatal"
) %>%
count_patients_with_event("SUBJID",
  filters = c("TRTEMFL" = "Y", "AEOU" = "FATAL", "AEREL" = "Y"),
  .labels = c(count_fraction = "Total number of patients with related fatal AEs"),
  .indent_mods = c(count_fraction = 2L),
  table_names = "tbl_rel_fatal"
)
build_table(lyt, tern_ex_adae, alt_counts_df = tern_ex_adsl)
```
count_patients_with_flags

```r

df, 
.var,  
flag_variables,  
.N_col,  
.N_row,  
denom = c("n", "N_row", "N_col")
)

count_patients_with_flags(
  lyt,  
  var,  
  var_labels = var,  
  show_labels = "hidden",  
  ...,  
  table_names = paste0("tbl_flags_", var),  
  .stats = "count_fraction",  
  .formats = NULL,  
  .indent_mods = NULL  
)
```

**Arguments**

- **df** *(data.frame)*
  data set containing all analysis variables.

- **.var** *(character)*
  name of the column that contains the unique identifier.

- **flag_variables** *(character)*
  a character vector specifying the names of logical variables from analysis dataset used for counting the number of unique identifiers.

- **.N_col** *(count)*
  row-wise N (row group count) for the group of observations being analyzed (i.e. with no column-based subsetting) that is passed by `rtables`.

- **.N_row** *(count)*
  column-wise N (column count) for the full column that is passed by `rtables`.

- **denom** *(string)*
  choice of denominator for proportion. Options are:
  - `n`: number of values in this row and column intersection.
  - `N_row`: total number of values in this row across columns.
  - `N_col`: total number of values in this column across rows.

- **lyt** *(layout)*
  input layout where analyses will be added to.

- **var** *(string)*
  single variable name that is passed by `rtables` when requested by a statistics function.

- **var_labels** *(character)*
  character for label.
count_patients_with_flags

show_labels (string)
label visibility: one of "default", "visible" and "hidden".

... additional arguments for the lower level functions.

table_names (character)
this can be customized in case that the same vars are analyzed multiple times, to avoid warnings from rtables.

.stats (character)
statistics to select for the table.

.formats (named character or list)
formats for the statistics.

.indent_mods (named integer)
indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.

Value

- s_count_patients_with_flags() returns the count and the fraction of unique identifiers with each particular flag as a list of statistics n, count, count_fraction, and n_blq, with one element per flag.

- a_count_patients_with_flags() returns the corresponding list with formatted rtables::CellValue().

- count_patients_with_flags() returns a layout object suitable for passing to further layouting functions, or to rtables::build_table(). Adding this function to an rtable layout will add formatted rows containing the statistics from s_count_patients_with_flags() to the table layout.

Functions

- s_count_patients_with_flags(): Statistics function which counts the number of patients for which a particular flag variable is TRUE.

- a_count_patients_with_flags(): Formatted analysis function which is used as afun in count_patients_with_flags().

- count_patients_with_flags(): Layout-creating function which can take statistics function arguments and additional format arguments. This function is a wrapper for rtables::analyze().

See Also

count_patients_with_event

Examples

library(dplyr)

# 's_count_patients_with_flags()'

# Add labelled flag variables to analysis dataset.
adae <- tern_ex_adae %>%
```r
mutate(
  fl1 = TRUE,
  fl2 = TRTEMFL == "Y",
  fl3 = TRTEMFL == "Y" & AEOUT == "FATAL",
  fl4 = TRTEMFL == "Y" & AEOUT == "FATAL" & AEREL == "Y"
)

labels <- c(
  "fl1" = "Total AEs",
  "fl2" = "Total number of patients with at least one adverse event",
  "fl3" = "Total number of patients with fatal AEs",
  "fl4" = "Total number of patients with related fatal AEs"
)

formatters::var_labels(adae)[names(labels)] <- labels

s_count_patients_with_flags(
  adae,
  "SUBJID",
  flag_variables = c("fl1", "fl2", "fl3", "fl4"),
  denom = "N_col",
  .N_col = 1000
)

# We need to ungroup `count_fraction` first so that the `rtables` formatting
# function `format_count_fraction()` can be applied correctly.

# `a_count_patients_with_flags`

afun <- make_afun(a_count_patients_with_flags,
  .stats = "count_fraction",
  .ungroup_stats = "count_fraction"
)

afun(
  adae,
  .N_col = 10L,
  .N_row = 10L,
  .var = "USUBJID",
  flag_variables = c("fl1", "fl2", "fl3", "fl4"
)
)

# `count_patients_with_flags`

lyt2 <- basic_table() %>%
  split_cols_by("ARM") %>%
  add_colcounts() %>%
  count_patients_with_flags(
    "SUBJID",
    flag_variables = formatters::var_labels(adae[, c("fl1", "fl2", "fl3", "fl4")]),
    denom = "N_col"
  )

build_table(lyt2, adae, alt_counts_df = tern_ex_adsl)
```
Counting Specific Values

Description

[Stable]

We can count the occurrence of specific values in a variable of interest.

Usage

```r
s_count_values(
  x,
  values,
  na.rm = TRUE,
  .N_col,
  .N_row,
  denom = c("n", "N_row", "N_col")
)
```

```r
## S3 method for class 'character'
  s_count_values(x, values = "Y", na.rm = TRUE, ...)
```

```r
## S3 method for class 'factor'
  s_count_values(x, values = "Y", ...)
```

```r
## S3 method for class 'logical'
  s_count_values(x, values = TRUE, ...)
```

```r
a_count_values(
  x,
  values,
  na.rm = TRUE,
  .N_col,
  .N_row,
  denom = c("n", "N_row", "N_col")
)
```

```r
count_values(
  lyt,
  vars,
  values,
  ...
  ,
  table_names = vars,
  .stats = "count_fraction",
  .formats = NULL,
  .labels = c(count_fraction = paste(values, collapse = ", ")),
```
Arguments

x               (numeric)
vector of numbers we want to analyze.
values          (character)
specific values that should be counted.
na.rm           (flag)
whether NA values should be removed from x prior to analysis.
.N_col          (count)
row-wise N (row group count) for the group of observations being analyzed (i.e.
with no column-based subsetting) that is passed by rtables.
.N_row          (count)
column-wise N (column count) for the full column that is passed by rtables.
denom           (string)
choice of denominator for proportion. Options are:
  • n: number of values in this row and column intersection.
  • N_row: total number of values in this row across columns.
  • N_col: total number of values in this column across rows.
...
additional arguments for the lower level functions.
lyt             (layout)
input layout where analyses will be added to.
vars            (character)
variable names for the primary analysis variable to be iterated over.
table_names     (character)
this can be customized in case that the same vars are analyzed multiple times,
to avoid warnings from rtables.
.stats          (character)
statistics to select for the table.
.formats        (named character or list)
formats for the statistics.
.labels         (named character)
labels for the statistics (without indent).
.indent_mods    (named integer)
indent modifiers for the labels. Defaults to 0, which corresponds to the unmodi-
fied default behavior. Can be negative.

Value

• s_count_values() returns output of s_summary() for specified values of a non-numeric vari-
  able.

• a_count_values() returns the corresponding list with formatted rtables::CellValue().
- `count_values()` returns a layout object suitable for passing to further layouting functions, or to `rtables::build_table()`. Adding this function to an rtable layout will add formatted rows containing the statistics from `s_count_values()` to the table layout.

**Functions**

- `s_count_values()`: S3 generic function to count values.
- `s_count_values(character)`: Method for character class.
- `s_count_values(factor)`: Method for factor class. This makes an automatic conversion to character and then forwards to the method for characters.
- `s_count_values(logical)`: Method for logical class.
- `a_count_values()`: Formatted analysis function which is used as `afun` in `count_values()`.
- `count_values()`: Layout-creating function which can take statistics function arguments and additional format arguments. This function is a wrapper for `rtables::analyze()`.

**Note**

- For factor variables, `s_count_values` checks whether values are all included in the levels of `x` and fails otherwise.
- For `count_values()`, variable labels are shown when there is more than one element in `vars`, otherwise they are hidden.

**Examples**

```r
# `s_count_values.character`
s_count_values(x = c("a", "b", "a"), values = "a")
s_count_values(x = c("a", "b", "a", NA, NA), values = "b", na.rm = FALSE)

# `s_count_values.factor`
s_count_values(x = factor(c("a", "b", "a")), values = "a")

# `s_count_values.logical`
s_count_values(x = c(TRUE, FALSE, TRUE))

# `a_count_values`
a_count_values(x = factor(c("a", "b", "a")), values = "a", .N_col = 10, .N_row = 10)

# `count_values`

basic_table() %>%
  count_values("Species", values = "setosa") %>%
  build_table(iris)
```
Cox Proportional Hazards Regression

Description

[Stable]
Fits a Cox regression model and estimates hazard ratio to describe the effect size in a survival analysis.

Usage

s_coxreg(model_df, .stats, .which_vars = "all", .var_nms = NULL)

a_coxreg(
  df,
  labelstr,
  eff = FALSE,
  var_main = FALSE,
  multivar = FALSE,
  variables,
  at = list(),
  control = control_coxreg(),
  .spl_context,
  .stats,
  .formats,  
  .indent_mods = NULL,
  na_level = "",
  cache_env = NULL
)

summarize_coxreg(
  lyt,
  variables,
  control = control_coxreg(),
  at = list(),
  multivar = FALSE,
  common_var = "STUDYID",
  .stats = c("n", "hr", "ci", "pval", "pval_inter"),
  .formats = c(n = "xx", hr = "xx.xx", ci = "(xx.xx, xx.xx)", pval = "x.xxxx | (<0.0001)", pval_inter = "x.xxxx | (<0.0001)",
  varlabels = NULL,
  .indent_mods = NULL,
  na_level = "",
  .section_div = NA_character_
)
Arguments

model_df (data.frame) contains the resulting model fit from a fit_coxreg function with tidying applied via broom::tidy().

.stats (character) the name of statistics to be reported among:
  - n: number of observations (univariate only)
  - hr: hazard ratio
  - ci: confidence interval
  - pval: p-value of the treatment effect
  - pval_inter: p-value of the interaction effect between the treatment and the covariate (univariate only)

.which_vars (character) which rows should statistics be returned for from the given model. Defaults to "all". Other options include "var_main" for main effects, "inter" for interaction effects, and "multi_lvl" for multivariate model covariate level rows. When .which_vars is "all" specific variables can be selected by specifying .var_nms.

.var_nms (character) the term value of rows in df for which .stats should be returned. Typically this is the name of a variable. If using variable labels, var should be a vector of both the desired variable name and the variable label in that order to see all .stats related to that variable. When .which_vars is "var_main" .var_nms should be only the variable name.

df (data.frame) data set containing all analysis variables.

labelstr (character) label of the level of the parent split currently being summarized (must be present as second argument in Content Row Functions). See rtables::summarize_row_groups() for more information.

eff (flag) whether treatment effect should be calculated. Defaults to FALSE.

var_main (flag) whether main effects should be calculated. Defaults to FALSE.

multivar (flag) Defaults to FALSE. If TRUE multivariate Cox regression will run, otherwise univariate Cox regression will run.

variables (named list of string) list of additional analysis variables.

at (list of numeric) when the candidate covariate is a numeric, use at to specify the value of the covariate at which the effect should be estimated.

control (list) a list of parameters as returned by the helper function control_coxreg().
.spl_context (data.frame)
gives information about ancestor split states that is passed by rtables.
.formats (named character or list)
formats for the statistics.
.indent_mods (named integer)
indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.

na_level (string)
custom string to replace all NA values with. Defaults to "".

.cache_env (environment)
an environment object used to cache the regression model in order to avoid repeatedly fitting the same model for every row in the table. Defaults to NULL (no caching).
.lyt (layout)
input layout where analyses will be added to.

.common_var (character)
the name of a factor variable in the dataset which takes the same value for all rows. This should be created during pre-processing if no such variable currently exists.

.varlabels (list)
a named list corresponds to the names of variables found in data, passed as a named list and corresponding to time, event, arm, strata, and covariates terms. If arm is missing from variables, then only Cox model(s) including the covariates will be fitted and the corresponding effect estimates will be tabulated later.

@section_div (character)
string which should be repeated as a section divider between sections. Defaults to NA for no section divider. If a vector of two strings are given, the first will be used between treatment and covariate sections and the second between different covariates.

Details

Cox models are the most commonly used methods to estimate the magnitude of the effect in survival analysis. It assumes proportional hazards: the ratio of the hazards between groups (e.g., two arms) is constant over time. This ratio is referred to as the "hazard ratio" (HR) and is one of the most commonly reported metrics to describe the effect size in survival analysis (NEST Team, 2020).

Value

- s_coxreg() returns the selected statistic for from the Cox regression model for the selected variable(s).
- a_coxreg() returns formatted rtables::CellValue().
- summarize_coxreg() returns a layout object suitable for passing to further layouting functions, or to rtables::build_table(). Adding this function to an rtable layout will add a Cox regression table containing the chosen statistics to the table layout.
Functions

- `s_coxreg()`: Statistics function that transforms results tabulated from `fit_coxreg_univar()` or `fit_coxreg_multivar()` into a list.
- `a_coxreg()`: Analysis function which is used as `afun` in `rtables::analyze()` and `cfun` in `rtables::summarize_row_groups()` within `summarize_coxreg()`.
- `summarize_coxreg()`: Layout-creating function which creates a Cox regression summary table layout. This function is a wrapper for several `rtables` layouting functions. This function is a wrapper for `rtables::analyze_colvars()` and `rtables::summarize_row_groups()`.

See Also

- `fit_coxreg` for relevant fitting functions, `h_cox_regression` for relevant helper functions, and `tidy_coxreg` for custom tidy methods.
- `fit_coxreg_univar()` and `fit_coxreg_multivar()` which also take the variables, data, at (univariate only), and control arguments but return unformatted univariate and multivariate Cox regression models, respectively.

Examples

```r
library(survival)

# Testing dataset [survival::bladder].
set.seed(1, kind = "Mersenne-Twister")
dta_bladder <- with(
  data = bladder[bladder$enum < 5, ],
  tibble::tibble(
    TIME = stop,
    STATUS = event,
    ARM = as.factor(rx),
    COVAR1 = as.factor(enum) %>% formatters::with_label("A Covariate Label"),
    COVAR2 = factor(
      sample(as.factor(enum)),
      levels = 1:4, labels = c("F", "F", "M", "M")
    ) %>% formatters::with_label("Sex (F/M)")
  )
)
dta_bladder$AGE <- sample(20:60, size = nrow(dta_bladder), replace = TRUE)
dta_bladder$STUDYID <- factor("X")

plot(
  survfit(Surv(TIME, STATUS) ~ ARM + COVAR1, data = dta_bladder),
  lty = 2:4,
  xlab = "Months",
  col = c("blue1", "blue2", "blue3", "blue4", "red1", "red2", "red3", "red4")
)

# s_coxreg

# Univariate
ul_variables <- list(
  
```
time = "TIME", event = "STATUS", arm = "ARM", covariates = c("COVAR1", "COVAR2")
) univar_model <- fit_coxreg_univar(variables = u1_variables, data = dta_bladder) df1 <- broom::tidy(univar_model) s_coxreg(model_df = df1, .stats = "hr")

# Univariate with interactions univar_model_inter <- fit_coxreg_univar( variables = u1_variables, control = control_coxreg(interaction = TRUE), data = dta_bladder) df1_inter <- broom::tidy(univar_model_inter) s_coxreg(model_df = df1_inter, .stats = "hr", .which_vars = "inter", .var_nms = "COVAR1")

# Univariate without treatment arm - only "COVAR2" covariate effects u2_variables <- list(time = "TIME", event = "STATUS", covariates = c("COVAR1", "COVAR2")) univar_covs_model <- fit_coxreg_univar(variables = u2_variables, data = dta_bladder) df1_covs <- broom::tidy(univar_covs_model) s_coxreg(model_df = df1_covs, .stats = "hr", .var_nms = c("COVAR2", "Sex (F/M)"))

# Multivariate. m1_variables <- list( time = "TIME", event = "STATUS", arm = "ARM", covariates = c("COVAR1", "COVAR2")
) multivar_model <- fit_coxreg_multivar(variables = m1_variables, data = dta_bladder) df2 <- broom::tidy(multivar_model) s_coxreg(model_df = df2, .stats = "pval", .which_vars = "var_main", .var_nms = "COVAR1")

# Multivariate without treatment arm - only "COVAR1" main effect m2_variables <- list(time = "TIME", event = "STATUS", covariates = c("COVAR1", "COVAR2")) multivar_covs_model <- fit_coxreg_multivar(variables = m2_variables, data = dta_bladder) df2_covs <- broom::tidy(multivar_covs_model) s_coxreg(model_df = df2_covs, .stats = "hr")

a_coxreg( df = dta_bladder, labelstr = "Label 1", variables = u1_variables, .spl_context = list(value = "COVAR1"), .stats = "n", .formats = "xx"
)

a_coxreg( df = dta_bladder, labelstr = "", variables = u1_variables, .spl_context = list(value = "COVAR2"), .stats = "pval", .formats = "xx.xxxx"
```r
# summarize_coxreg
result_univar <- basic_table() %>%
  summarize_coxreg(variables = u1_variables) %>%
  build_table(dta_bladder)
result_univar

result_multivar <- basic_table() %>%
  summarize_coxreg(
    variables = m1_variables,
    multivar = TRUE,
  ) %>%
  build_table(dta_bladder)
result_multivar

result_univar_covs <- basic_table() %>%
  summarize_coxreg(
    variables = u2_variables,
  ) %>%
  build_table(dta_bladder)
result_univar_covs

result_multivar_covs <- basic_table() %>%
  summarize_coxreg(
    variables = m2_variables,
    multivar = TRUE,
    varlabels = c("Covariate 1", "Covariate 2") # custom labels
  ) %>%
  build_table(dta_bladder)
result_multivar_covs
```

---

**cox_regression_inter**  
*Cox Regression Helper: Interactions*

**Description**

[Stable]
Test and estimate the effect of a treatment in interaction with a covariate. The effect is estimated as the HR of the tested treatment for a given level of the covariate, in comparison to the treatment control.

**Usage**

```r
h_coxreg_inter_effect(x, effect, covar, mod, label, control, ...)
```

## S3 method for class 'numeric'
h_coxreg_inter_effect(x, effect, covar, mod, label, control, at, ...)

## S3 method for class 'factor'
h_coxreg_inter_effect(x, effect, covar, mod, label, control, data, ...)

## S3 method for class 'character'

h_coxreg_inter_effect(x, effect, covar, mod, label, control, data, ...)

h_coxreg_extract_interaction(effect, covar, mod, data, at, control)

h_coxreg_inter_estimations(
    variable,
    given,
    lvl_var,
    lvl_given,
    mod,
    conf_level = 0.95
)

**Arguments**

- **x** (numeric or factor)
  the values of the covariate to be tested.

- **effect** (string)
  the name of the effect to be tested and estimated.

- **covar** (string)
  the name of the covariate in the model.

- **mod** (coxph)
  a fitted Cox regression model (see `survival::coxph()`).

- **label** (string)
  the label to be returned as `term_label`.

- **control** (list)
  a list of controls as returned by `control_coxreg()`.

- **at** (list)
  a list with items named after the covariate, every item is a vector of levels at which the interaction should be estimated.

- **data** (data.frame)
  the data frame on which the model was fit.

- **variable, given** (string)
  the name of variables in interaction. We seek the estimation of the levels of variable given the levels of `given`.

- **lvl_var, lvl_given**
  (character)
  corresponding levels has given by `levels()`.
conf_level  (proportion)
confidence level of the interval.

Details
Given the cox regression investigating the effect of Arm (A, B, C; reference A) and Sex (F, M; reference Female) and the model being abbreviated: \( y \sim \text{Arm} + \text{Sex} + \text{Arm:Sex} \). The cox regression estimates the coefficients along with a variance-covariance matrix for:

- \( b_1 \) (arm b), \( b_2 \) (arm c)
- \( b_3 \) (sex m)
- \( b_4 \) (arm b: sex m), \( b_5 \) (arm c: sex m)

The estimation of the Hazard Ratio for arm C/sex M is given in reference to arm A/Sex M by \( \exp(b_2 + b_3 + b_5)/\exp(b_3) = \exp(b_2 + b_5) \). The interaction coefficient is deduced by \( b_2 + b_5 \) while the standard error is obtained as \( \sqrt{\text{Var} b_2 + \text{Var} b_5 + 2 \times \text{covariance} (b_2,b_5)} \).

Value
- \text{h_coxreg_inter_effect()}\): returns a data.frame of covariate interaction effects consisting of the following variables: effect, term, term_label, level, n, hr, lcl, ucl, pval, and pval_inter.
- \text{h_coxreg_extract_interaction()}\): returns the result of an interaction test and the estimated values. If no interaction, \text{h_coxreg_univar_extract()}\) is applied instead.
- \text{h_coxreg_inter_estimations()}\): returns a list of matrices (one per level of variable) with rows corresponding to the combinations of variable and given, with columns:
  - coef_hat: Estimation of the coefficient.
  - coef_se: Standard error of the estimation.
  - hr: Hazard ratio.
  - lcl, ucl: Lower/upper confidence limit of the hazard ratio.

Functions
- \text{h_coxreg_inter_effect()}\): S3 generic helper function to determine interaction effect.
- \text{h_coxreg_inter_effect(numeric)}\): Method for numeric class. Estimates the interaction with a numeric covariate.
- \text{h_coxreg_inter_effect(factor)}\): Method for factor class. Estimate the interaction with a factor covariate.
- \text{h_coxreg_inter_effect(character)}\): Method for character class. Estimate the interaction with a character covariate. This makes an automatic conversion to factor and then forwards to the method for factors.
- \text{h_coxreg_extract_interaction()}\): A higher level function to get the results of the interaction test and the estimated values.
- \text{h_coxreg_inter_estimations()}\): Hazard ratio estimation in interactions.
Note

- Automatic conversion of character to factor does not guarantee results can be generated correctly. It is therefore better to always pre-process the dataset such that factors are manually created from character variables before passing the dataset to `rtables::build_table()`.

Examples

```r
library(survival)

set.seed(1, kind = "Mersenne-Twister")

# Testing dataset [survival::bladder].
dta_bladder <- with(data = bladder[bladder$enum < 5, ],
data.frame(
  time = stop,
  status = event,
  armcd = as.factor(rx),
  covar1 = as.factor(enum),
  covar2 = factor(
    sample(as.factor(enum)),
    levels = 1:4,
    labels = c("F", "F", "M", "M")
  ),
)
)
labels <- c("armcd" = "ARM", "covar1" = "A Covariate Label", "covar2" = "Sex (F/M)"
formatters::var_labels(dta_bladder)[names(labels)] <- labels
dta_bladder$age <- sample(20:60, size = nrow(dta_bladder), replace = TRUE)

plot(
  survfit(Surv(time, status) ~ armcd + covar1, data = dta_bladder),
lty = 2:4,
xlab = "Months",
col = c("blue1", "blue2", "blue3", "blue4", "red1", "red2", "red3", "red4")
)

mod <- coxph(Surv(time, status) ~ armcd * covar1, data = dta_bladder)
h_coxreg_extract_interaction(
  mod = mod, effect = "armcd", covar = "covar1", data = dta_bladder,
  control = control_coxreg()
)

mod <- coxph(Surv(time, status) ~ armcd * covar1, data = dta_bladder)
result <- h_coxreg_inter_estimations(
  variable = "armcd", given = "covar1",
  lvl_var = levels(dta_bladder$armcd),
  lvl_given = levels(dta_bladder$covar1),
  mod = mod, conf_level = .95
)
result
```
create_afun_compare  Constructor Function for compare_vars()

Description

[Stable]
Constructor function which creates a combined formatted analysis function.

Usage

create_afun_compare(
  .stats = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)

Arguments

.stats (character)
statistics to select for the table.

.formats (named character or list)
formats for the statistics.

.labels (named character)
labels for the statistics (without indent).

.indent_mods (named vector of integer)
indent modifiers for the labels. Each element of the vector should be a name-value pair with name corresponding to a statistic specified in .stats and value the indentation for that statistic’s row label.

Value

Combined formatted analysis function for use in compare_vars().

Note

Since a_compare() is generic and we want customization of the formatting arguments via rtables::make_afun(), we need to create another temporary generic function, with corresponding customized methods. Then in order for the methods to be found, we need to wrap them in a combined afun. Since this is required by two layout creating functions (and possibly others in the future), we provide a constructor that does this: create_afun_compare().

See Also

compare_vars()
create_afun_summary

Examples

```r
# `create_afun_compare()` to create combined `afun`

afun <- create_afun_compare(
  .stats = c("n", "count_fraction", "mean_sd", "pval"),
  .indent_mods = c(pval = 1L)
)

lyt <- basic_table() %>%
  split_cols_by("ARMCD", ref_group = "ARM A") %>%
  analyze(
    "AGE",
    afun = afun,
    show_labels = "visible"
  )
build_table(lyt, df = tern_ex_adsl)

lyt <- basic_table() %>%
  split_cols_by("ARMCD", ref_group = "ARM A") %>%
  analyze(
    "SEX",
    afun = afun,
    show_labels = "visible"
  )
build_table(lyt, df = tern_ex_adsl)
```

create_afun_summary Constructor Function for summarize_vars() and summarize_colvars()

Description

[Stable]
Constructor function which creates a combined formatted analysis function.

Usage

`create_afun_summary(.stats, .formats, .labels, .indent_mods)`

Arguments

- `.stats` (character)
  statistics to select for the table.
- `.formats` (named character or list)
  formats for the statistics.
- `.labels` (named character)
  labels for the statistics (without indent).
cut_quantile_bins  

.cut_quantile_bins  

*.indent_mods*  
(named vector of integer)  
indent modifiers for the labels. Each element of the vector should be a name-value pair with name corresponding to a statistic specified in *stats* and value the indentation for that statistic’s row label.

**Value**

Combined formatted analysis function for use in *summarize_vars()*.

**Note**

Since *a_summary()* is generic and we want customization of the formatting arguments via *rtables::make_afun()*, we need to create another temporary generic function, with corresponding customized methods. Then in order for the methods to be found, we need to wrap them in a combined *afun*. Since this is required by two layout creating functions (and possibly others in the future), we provide a constructor that does this: *create_afun_summary()*.

**Examples**

```r
# 'create_afun_summary()' to create combined 'afun'

afun <- create_afun_summary(
  .stats = NULL,
  .formats = c(median = "xx."),
  .labels = c(median = "My median"),
  .indent_mods = c(median = 1L)
)

## Fabricated dataset.
dta_test <- data.frame(
  USUBJID = rep(1:6, each = 3),
  PARAMCD = rep("lab", 6 * 3),
  AVISIT = rep(paste0("V", 1:3), 6),
  ARM = rep(LETTERS[1:3], rep(6, 3)),
  AVAL = c(9:1, rep(NA, 9))
)

l <- basic_table() %>%
  split_cols_by(var = "ARM") %>%
  split_rows_by(var = "AVISIT") %>%
  analyze(vars = "AVAL", afun = afun)

build_table(l, df = dta_test)
```
Description

[Stable]
This cuts a numeric vector into sample quantile bins.

Usage

```r
cut_quantile_bins(
  x,
  probs = c(0.25, 0.5, 0.75),
  labels = NULL,
  type = 7,
  ordered = TRUE
)
```

Arguments

- **x** (numeric): the continuous variable values which should be cut into quantile bins. This may contain NA values, which are then not used for the quantile calculations, but included in the return vector.
- **probs** (proportion vector): the probabilities identifying the quantiles. This is a sorted vector of unique proportion values, i.e. between 0 and 1, where the boundaries 0 and 1 must not be included.
- **labels** (character): the unique labels for the quantile bins. When there are n probabilities in `probs`, then this must be n + 1 long.
- **type** (integer): type of quantiles to use, see `stats::quantile()` for details.
- **ordered** (flag): should the result be an ordered factor.

Value

A factor variable with appropriately-labeled bins as levels.

Note

Intervals are closed on the right side. That is, the first bin is the interval \([-\infty, q_1]\) where \(q_1\) is the first quantile, the second bin is then \((q_1, q_2]\), etc., and the last bin is \((q_n, +\infty]\) where \(q_n\) is the last quantile.

Examples

```r
# Default is to cut into quartile bins.
cut_quantile_bins(cars$speed)

# Use custom quantiles.
```
day2month

```r
cut_quantile_bins(cars$speed, probs = c(0.1, 0.2, 0.6, 0.88))

# Use custom labels.
cut_quantile_bins(cars$speed, labels = paste0("Q", 1:4))

# NAs are preserved in result factor.
ozone_binned <- cut_quantile_bins(airquality$Ozone)
which(is.na(ozone_binned))
# So you might want to make these explicit.
explicit_na(ozone_binned)
```

---

day2month  Conversion of Days to Months

**Description**

Conversion of Days to Months

**Usage**

day2month(x)

**Arguments**

x  (numeric)

  time in days.

**Value**

A numeric vector with the time in months.

**Examples**

```r
x <- c(403, 248, 30, 86)
day2month(x)
```
Add Titles, Footnotes, Page Number, and a Bounding Box to a Grid Grob

Description

[Stable]
This function is useful to label grid grobs (also ggplot2, and lattice plots) with title, footnote, and page numbers.

Usage

decorate_grob(
  grob, 
  titles, 
  footnotes, 
  page = "", 
  width_titles = grid::unit(1, "npc") - grid::stringWidth(page), 
  width_footnotes = grid::unit(1, "npc") - grid::stringWidth(page), 
  border = TRUE, 
  margins = grid::unit(c(1, 0, 1, 0), "lines"), 
  padding = grid::unit(rep(1, 4), "lines"), 
  outer_margins = grid::unit(c(2, 1.5, 3, 1.5), "cm"), 
  gp_titles = grid::gpar(), 
  gp_footnotes = grid::gpar(fontsize = 8), 
  name = NULL, 
  gp = grid::gpar(), 
  vp = NULL
)

Arguments

grob a grid grob object, optionally NULL if only a grob with the decoration should be shown.
titles vector of character strings. Vector elements are separated by a newline and strings are wrapped according to the page width.
footnotes vector of character string. Same rules as for titles.
page string with page numeration, if NULL then no page number is displayed.
width_titles unit object
width_footnotes unit object
border boolean, whether a a border should be drawn around the plot or not.
margins unit object of length 4
padding unit object of length 4
### decorate_grob

- **outer_margins**: unit object of length 4
- **gp_titles**: a gpar object
- **gp_footnotes**: a gpar object
- **name**: a character identifier for the grob. Used to find the grob on the display list and/or as a child of another grob.
- **gp**: A "gpar" object, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.
- **vp**: a viewport object (or NULL).

### Details

The titles and footnotes will be ragged, i.e. each title will be wrapped individually.

### Value

A grid grob (gTree).

### Examples

```r
library(grid)

titles <- c(
  "Edgar Anderson's Iris Data",
  paste("This famous (Fisher's or Anderson's) iris data set gives the measurements",
        "in centimeters of the variables sepal length and width and petal length",
        "and width, respectively, for 50 flowers from each of 3 species of iris."
  )
)

footnotes <- c(
  "The species are Iris setosa, versicolor, and virginica."
)

## empty plot
grid.newpage()

grid.draw(decorate_grob(
   NULL,
   titles = titles,
   footnotes = footnotes,
   page = "Page 4 of 10"
))
```

# grid
p <- gTree(
  children = gList(
    rectGrob(),
    xaxisGrob(),
    yaxisGrob(),
    textGrob("Sepal.Length", y = unit(-4, "lines")),
    textGrob("Petal.Length", x = unit(-3.5, "lines"), rot = 90),
    pointsGrob(iris$Sepal.Length, iris$Petal.Length, gp = gpar(col = iris$Species), pch = 16)
  ),
  vp = vpStack(plotViewport(), dataViewport(xData = iris$Sepal.Length, yData = iris$Petal.Length))
)
grid.newpage()
grid.draw(p)

grid.newpage()
grid.draw(
  decorate_grob(
    grob = p,
    titles = titles,
    footnotes = footnotes,
    page = "Page 6 of 129"
  )
)

## with ggplot2
library(ggplot2)

p_gg <- ggplot2::ggplot(iris, aes(Sepal.Length, Sepal.Width, col = Species)) +
  ggplot2::geom_point()
p_gg
p <- ggplotGrob(p_gg)
grid.newpage()
grid.draw(
  decorate_grob(
    grob = p,
    titles = titles,
    footnotes = footnotes,
    page = "Page 6 of 129"
  )
)

## with lattice
library(lattice)

xyplot(Sepal.Length ~ Petal.Length, data = iris, col = iris$Species)
p <- grid.grab()
grid.newpage()
grid.draw(
  decorate_grob(
    grob = p,
    titles = titles,
    footnotes = footnotes,
...
decorate_grob_set

Decorate Set of grobs and Add Page Numbering

Description

[Stable]
Note that this uses the decorate_grob_factory() function.

Usage

decorate_grob_set(grobs, ...)

Arguments

grobs a list of grid grobs
... arguments passed on to decorate_grob().

Value

A decorated grob.

Examples

library(ggplot2)
library(grid)
g <- with(data = iris, {
  list(
    ggplot2::ggplotGrob(
      ggplot2::ggplot(mapping = aes(Sepal.Length, Sepal.Width, col = Species)) +
      ggplot2::geom_point())
  )
})
df_explicit_na

Encode Categorical Missing Values in a Data Frame

Description

[Stable]

This is a helper function to encode missing entries across groups of categorical variables in a data frame.

Usage

df_explicit_na(
  data,
  omit_columns = NULL,
  char_as_factor = TRUE,
  logical_as_factor = FALSE,
  na_level = "<Missing>"
)
Arguments

- **data** (data.frame)
  data set.

- **omit_columns** (character)
  names of variables from data that should not be modified by this function.

- **char_as_factor** (flag)
  whether to convert character variables in data to factors.

- **logical_as_factor** (flag)
  whether to convert logical variables in data to factors.

- **na_level** (string)
  used to replace all NA or empty values inside non-omit_columns columns.

Details

Missing entries are those with NA or empty strings and will be replaced with a specified value. If factor variables include missing values, the missing value will be inserted as the last level. Similarly, in case character or logical variables should be converted to factors with the char_as_factor or logical_as_factor options, the missing values will be set as the last level.

Value

A data.frame with the chosen modifications applied.

See Also

`sas_na()` and `explicit_na()` for other missing data helper functions.

Examples

```r
my_data <- data.frame(
  u = c(TRUE, FALSE, NA, TRUE),
  v = factor(c("A", NA, NA, NA), levels = c("Z", "A")),
  w = c("A", "B", NA, "C"),
  x = c("D", "E", "F", NA),
  y = c("G", "H", "I", "),
  z = c(1, 2, 3, 4),
  stringsAsFactors = FALSE
)

# Example 1
# Encode missing values in all character or factor columns.
df_explicit_na(my_data)
# Also convert logical columns to factor columns.
df_explicit_na(my_data, logical_as_factor = TRUE)
# Encode missing values in a subset of columns.
df_explicit_na(my_data, omit_columns = c("x", "y"))

# Example 2
```

# Here we purposefully convert all 'M' values to 'NA' in the 'SEX' variable.
# After running `df_explicit_na` the 'NA' values are encoded as '<Missing>' but they are not
# included when generating 'rtables'.
adsl <- tern_ex_adsl
adsl$SEX[adsl$SEX == "M"] <- NA
adsl <- df_explicit_na(adsl)

# If you want the 'Na' values to be displayed in the table use the 'na_level' argument.
adsl <- tern_ex_adsl
adsl$SEX[adsl$SEX == "M"] <- NA
adsl <- df_explicit_na(adsl, na_level = "Missing Values")

# Example 3
# Numeric variables that have missing values are not altered. This means that any 'NA' value in
# a numeric variable will not be included in the summary statistics, nor will they be included
# in the denominator value for calculating the percent values.
adsl <- tern_ex_adsl
adsl$AGE[adsl$AGE < 30] <- NA
adsl <- df_explicit_na(adsl)

---

**draw_grob**

*Draw grob*

**Description**

*Stable*

Draw grob on device page.

**Usage**

`draw_grob(grob, newpage = TRUE, vp = NULL)`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grob</td>
<td>grid object</td>
</tr>
<tr>
<td>newpage</td>
<td>draw on a new page</td>
</tr>
<tr>
<td>vp</td>
<td>a <code>viewport()</code> object (or NULL).</td>
</tr>
</tbody>
</table>

**Value**

A grob.
Examples

```r
dlibrary(dplyr)
dlibrary(grid)

d <- rectGrob(width = grid::unit(0.5, "npc"), height = grid::unit(0.5, "npc"))
d %>% draw_grob(vp = grid::viewport(angle = 45))

num <- lapply(1:10, textGrob)
um %>%
  arrange_grobs(grobs = .) %>%
  draw_grob()
showViewport()
```

---

d_count_abnormal_by_baseline

*Description Function for s_count_abnormal_by_baseline()*

Description

[Stable]

Description function that produces the labels for `s_count_abnormal_by_baseline()`.

Usage

```r
d_count_abnormal_by_baseline(abnormal)
```

Arguments

- `abnormal` (character)
  - identifying the abnormal range level(s) in `.var`.

Value

Abnormal category labels for `s_count_abnormal_by_baseline()`.

Examples

```r
d_count_abnormal_by_baseline("LOW")
```
d_count_cumulative  Description of Cumulative Count

Description

[Stable]
This is a helper function that describes the analysis in s_count_cumulative().

Usage

d_count_cumulative(threshold, lower_tail, include_eq)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>threshold</td>
<td>(number) a cutoff value as threshold to count values of x.</td>
</tr>
<tr>
<td>lower_tail</td>
<td>(logical) whether to count lower tail, default is TRUE.</td>
</tr>
<tr>
<td>include_eq</td>
<td>(logical) whether to include value equal to the threshold in count, default is TRUE.</td>
</tr>
</tbody>
</table>

Value

Labels for s_count_cumulative().

d_count_missed_doses  Description Function that Calculates Labels for s_count_missed_doses().

Description

[Stable]

Usage

d_count_missed_doses(thresholds)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>thresholds</td>
<td>(vector of count) number of missed doses the patients at least had.</td>
</tr>
</tbody>
</table>

Value

d_count_missed_doses() returns a named character vector with the labels.
d_onco_rsp_label

See Also

s_count_missed_doses()

---

**Description**

[Stable]

Describe the oncology response in a standard way.

**Usage**

d_onco_rsp_label(x)

**Arguments**

x (character)

the standard oncology code to be described.

**Value**

Response labels.

**See Also**

estimate_multinomial_rsp()

**Examples**

d_onco_rsp_label(
  "<Missing>", "NE/Missing")
)

# Adding some values not considered in d_onco_rsp_label

d_onco_rsp_label(
  c("CR", "PR", "hello", "hi")
)
d_pkparam Generate PK reference dataset

Description

[Stable]

Usage

d_pkparam()

Value

data.frame of PK parameters

Examples

pk_reference_dataset <- d_pkparam()

d_proportion Description of the Proportion Summary

Description

[Stable]
This is a helper function that describes the analysis in s_proportion().

Usage

d_proportion(conf_level, method, long = FALSE)

Arguments

conf_level  (proportion)
  confidence level of the interval.
metho  (string)
  the method used to construct the confidence interval for proportion of successful outcomes; one of waldcc, wald, clopper-pearson, wilson, wilsone, strat_wilson, strat_wilsonc, agresti-coull or jeffreys.
long  (flag)
  whether a long or a short (default) description is required.

Value

String describing the analysis.
**Description**

[Stable]

This is an auxiliary function that describes the analysis in \texttt{s_proportion_diff}.

**Usage**

\texttt{d_proportion_diff(conf_level, method, long = FALSE)}

**Arguments**

- \texttt{conf_level} (proportion): confidence level of the interval.
- \texttt{method} (string): the method used for the confidence interval estimation.
- \texttt{long} (logical): Whether a long or a short (default) description is required.

**Value**

A string describing the analysis.

**See Also**

\texttt{prop_diff}

---

**d_rsp_subgroups_colvars**

*Labels for Column Variables in Binary Response by Subgroup Table*

**Description**

[Stable]

Internal function to check variables included in \texttt{tabulate_rsp_subgroups()} and create column labels.

**Usage**

\texttt{d_rsp_subgroups_colvars(vars, conf_level = NULL, method = NULL)}
Arguments

vars (character)
variable names for the primary analysis variable to be iterated over.

conf_level (proportion)
confidence level of the interval.

method (string)
specifies the test used to calculate the p-value for the difference between two proportions. For options, see `s_test_proportion_diff()`. Default is NULL so no test is performed.

Value

A list of variables to tabulate and their labels.

d_survival_subgroups_colvars

Labels for Column Variables in Survival Duration by Subgroup Table

Description

[Stable]

Internal function to check variables included in `tabulate_survival_subgroups()` and create column labels.

Usage

d_survival_subgroups_colvars(vars, conf_level, method, time_unit = NULL)

Arguments

vars (character)
the name of statistics to be reported among:

• n_tot_events: Total number of events per group.
• n_events: Number of events per group.
• n_tot: Total number of observations per group.
• n: Number of observations per group.
• median: Median survival time.
• hr: Hazard ratio.
• ci: Confidence interval of hazard ratio.
• pval: p-value of the effect. Note, one of the statistics n_tot and n_tot_events, as well as both hr and ci are required.

conf_level (proportion)
confidence level of the interval.
D_test_proportion_diff

Description

[Stable]

This is an auxiliary function that describes the analysis in s_test_proportion_diff.

Usage

d_test_proportion_diff(method)

Arguments

method (string)
  one of chisq, cmh, fisher, or schouten; specifies the test used to calculate the p-value.

time_unit (string)
  label with unit of median survival time. Default NULL skips displaying unit.

Value

A list of variables and their labels to tabulate.

Note

At least one of n_tot and n_tot_events must be provided in vars.
estimation of proportions per level of factor

Description

[Stable]

Estimate the proportion along with confidence interval of a proportion regarding the level of a factor.

Usage

s_length_proportion(x, .N_col, ...)

a_length_proportion(x, .N_col, ...)

estimate_multinomial_response(
    lyt,
    var,
    ...,
    show_labels = "hidden",
    table_names = var,
    .stats = "prop_ci",
    .formats = NULL,
    .labels = NULL,
    .indent_mods = NULL
)

Arguments

x (numeric)

vector of numbers we want to analyze.

.N_col (count)

row-wise N (row group count) for the group of observations being analyzed (i.e. with no column-based subsetting) that is passed by rtables.

... additional arguments for the lower level functions.

lyt (layout)

input layout where analyses will be added to.

var (string)

single variable name that is passed by rtables when requested by a statistics function.

show_labels (string)

label visibility: one of "default", "visible" and "hidden".

table_names (character)

this can be customized in case that the same vars are analyzed multiple times, to avoid warnings from rtables.
estimate_multinomial_rsp

.stats (character)
  statistics to select for the table.
.forms (named character or list)
  formats for the statistics.
.labels (named character)
  labels for the statistics (without indent).
.indent_mods (named integer)
  indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.

Value

- `s_length_proportion()` returns statistics from `s_proportion()`.
- `a_length_proportion()` returns the corresponding list with formatted `rtables::CellValue()`.
- `estimate_multinomial_response()` returns a layout object suitable for passing to further layouting functions, or to `rtables::build_table()`. Adding this function to an rtable layout will add formatted rows containing the statistics from `s_length_proportion()` to the table layout.

Functions

- `s_length_proportion()`: Statistics function which feeds the length of x as number of successes, and .N_col as total number of successes and failures into `s_proportion()`.
- `a_length_proportion()`: Formatted analysis function which is used as afun in `estimate_multinomial_response()`.
- `estimate_multinomial_response()`: Layout-creating function which can take statistics function arguments and additional format arguments. This function is a wrapper for `rtables::analyze()` and `rtables::summarize_row_groups()`.

See Also

- Relevant description function `d_onco_rsp_label()`.

Examples

```r
s_length_proportion(rep("CR", 10), .N_col = 100)
s_length_proportion(factor(character(0)), .N_col = 100)
a_length_proportion(rep("CR", 10), .N_col = 100)
a_length_proportion(factor(character(0)), .N_col = 100)
library(dplyr)

# Use of the layout creating function.
data_test <- data.frame(
  USUBJID = paste0("S", 1:12),
  ARM = factor(rep(LETTERS[1:3], each = 4)),
  AVAL = c(A = c(1, 1, 1, 1), B = c(0, 0, 1, 1), C = c(0, 0, 0, 0)))
```
```r
AVALC = factor(AVAL,
       levels = c(0, 1),
       labels = c("Complete Response (CR)", "Partial Response (PR)"
       )
)

lyt <- basic_table() %>%
  split_cols_by("ARM") %>%
  estimate_multinomial_response(var = "AVALC")

tbl <- build_table(lyt, dta_test)

html <- as_html(tbl)
html
```

---

**estimate_proportions**  
*Estimation of Proportions*

---

**Description**

**[Stable]**

Estimate the proportion of responders within a studied population.

**Usage**

```r
s_proportion(
  df,
  .var,
  conf_level = 0.95,
  method = c("waldcc", "wald", "clopper-pearson", "wilson", "wilsonc", "strat_wilson",
             "strat_wilsonc", "agresti-coull", "jeffreys"),
  weights = NULL,
  max_iterations = 50,
  variables = list(strata = NULL),
  long = FALSE
)
```

```r
a_proportion(
  df,
  .var,
  conf_level = 0.95,
  method = c("waldcc", "wald", "clopper-pearson", "wilson", "wilsonc", "strat_wilson",
             "strat_wilsonc", "agresti-coull", "jeffreys"),
```
weights = NULL,
max_iterations = 50,
variables = list(strata = NULL),
long = FALSE
}

estimate_proportion(
  lyt,
  vars,
  ..., 
  show_labels = "hidden",
  table_names = vars,
  .stats = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)

Arguments

df (logical or data.frame)
if only a logical vector is used, it indicates whether each subject is a responder or not. TRUE represents a successful outcome. If a data.frame is provided, also the strata variable names must be provided in variables as a list element with the strata strings. In the case of data.frame, the logical vector of responses must be indicated as a variable name in .var.

.var (string)
single variable name that is passed by rtables when requested by a statistics function.

cnf_level (proportion)
confidence level of the interval.

method (string)
the method used to construct the confidence interval for proportion of successful outcomes; one of waldcc, wald, clopper-pearson, wilson, wilsonc, strat_wilson, strat_wilsonc, agresti-coull or jeffreys.

weights (numeric or NULL)
weights for each level of the strata. If NULL, they are estimated using the iterative algorithm proposed in Yan and Su (2010) that minimizes the weighted squared length of the confidence interval.

max_iterations (count)
maximum number of iterations for the iterative procedure used to find estimates of optimal weights.

variables (named list of string)
list of additional analysis variables.

long (flag)
a long description is required.
lyt (layout)
input layout where analyses will be added to.

vars (character)
variable names for the primary analysis variable to be iterated over.

... other arguments are ultimately conveyed to \texttt{s\_proportion()}. 

show_labels (string)
label visibility: one of "default", "visible" and "hidden".

table_names (character)
this can be customized in case that the same \texttt{vars} are analyzed multiple times, 
to avoid warnings from \texttt{rtables}.

.stats (character)
statistics to select for the table.

.formats (named character or list)
formats for the statistics.

.labels (named character)
labels for the statistics (without indent).

.indent_mods (named integer)
indent modifiers for the labels. Defaults to 0, which corresponds to the unmodi-
fied default behavior. Can be negative.

Value

• \texttt{s\_proportion()} returns statistics \texttt{n\_prop} (n and proportion) and \texttt{prop\_ci} (proportion CI) 
for a given variable.

• \texttt{a\_proportion()} returns the corresponding list with formatted \texttt{rtables\texttt{::CellValue}}().

• \texttt{estimate\_proportion()} returns a layout object suitable for passing to further layouting 
functions, or to \texttt{rtables\texttt{::build\_table}}(). Adding this function to an \texttt{rtable} layout will 
add formatted rows containing the statistics from \texttt{s\_proportion()} to the table layout.

Functions

• \texttt{s\_proportion()}: Statistics function estimating a proportion along with its confidence inter-
val.

• \texttt{a\_proportion()}: Formatted analysis function which is used as \texttt{afun} in \texttt{estimate\_proportion()}.

• \texttt{estimate\_proportion()}: Layout-creating function which can take statistics function argu-
ments and additional format arguments. This function is a wrapper for \texttt{rtables\texttt{::analyze}}().

See Also

\texttt{h\_proportions}
Examples

# Case with only logical vector.
rsp_v <- c(1, 0, 1, 0, 1, 1, 0, 0)
s_proportion(rsp_v)

# Example for Stratified Wilson CI
nex <- 100 # Number of example rows
dta <- data.frame(
    "rsp" = sample(c(TRUE, FALSE), nex, TRUE),
    "grp" = sample(c("A", "B"), nex, TRUE),
    "f1" = sample(c("a1", "a2"), nex, TRUE),
    "f2" = sample(c("x", "y", "z"), nex, TRUE),
    stringsAsFactors = TRUE
)

s_proportion(
    df = dta,
    .var = "rsp",
    variables = list(strata = c("f1", "f2")),
    conf_level = 0.90,
    method = "strat_wilson"
)

dta_test <- data.frame(
    USUBJID = paste0("S", 1:12),
    ARM = rep(LETTERS[1:3], each = 4),
    AVAL = c(A = c(1, 1, 1, 1), B = c(0, 0, 1, 1), C = c(0, 0, 0, 0))
)

basic_table() %>%
    split_cols_by("ARM") %>%
    estimate_proportion(vars = "AVAL") %>%
    build_table(df = dta_test)

code_box(explicit_na)

**Description**

[Stable]

Substitute missing data with a string or factor level.

**Usage**

`explicit_na(x, label = "<Missing>")`
Arguments

- **x** (factor or character vector)
  values for which any missing values should be substituted.

- **label** (character)
  string that missing data should be replaced with.

Value

x with any NA values substituted by label.

Examples

```r
explicit_na(c(NA, "a", "b"))
is.na(explicit_na(c(NA, "a", "b")))

explicit_na(factor(c(NA, "a", "b")))
is.na(explicit_na(factor(c(NA, "a", "b"))))

explicit_na(sas_na(c("a", "")))
```

---

**extract_rsp_biomarkers**

Prepares Response Data Estimates for Multiple Biomarkers in a Single Data Frame

Description

[Stable]
Prepares estimates for number of responses, patients and overall response rate, as well as odds ratio estimates, confidence intervals and p-values, for multiple biomarkers across population subgroups in a single data frame. variables corresponds to the names of variables found in data, passed as a named list and requires elements rsp and biomarkers (vector of continuous biomarker variables) and optionally covariates, subgroups and strat. groups_lists optionally specifies groupings for subgroups variables.

Usage

```r
extract_rsp_biomarkers(
  variables, 
  data, 
  groups_lists = list(),
  control = control_logistic(),
  label_all = "All Patients"
)
```
extract_rsp_biomarkers

Arguments

variables (named list of string)
list of additional analysis variables.

data (data.frame)
the dataset containing the variables to summarize.

groups_lists (named list of list)
optionally contains for each subgroups variable a list, which specifies the new group levels via the names and the levels that belong to it in the character vectors that are elements of the list.

control (named list)
controls for the response definition and the confidence level produced by control_logistic().

label_all (string)
label for the total population analysis.

Value

A data.frame with columns biomarker, biomarker_label, n_tot, n_rsp, prop, or, lcl, ucl, conf_level, pval, pval_label, subgroup, var, var_label, and row_type.

Note

You can also specify a continuous variable in rsp and then use the response_definition control to convert that internally to a logical variable reflecting binary response.

See Also

h_logistic_mult_cont_df() which is used internally.

Examples

library(dplyr)
library(forcats)

adrs <- tern_ex_adrs
adrs_labels <- formatters::var_labels(adrs)

adrs_f <- adrs %>%
filter(PARAMCD == "BESRSP1") %>%
mutate(rsp = AVALC == "CR")

# Typical analysis of two continuous biomarkers 'BMRKR1' and 'AGE',
# in logistic regression models with one covariate 'RACE'. The subgroups
# are defined by the levels of 'BMRKR2'.
df <- extract_rsp_biomarkers(
  variables = list(
    rsp = "rsp",
    biomarkers = c("BMRKR1", "AGE"),
    covariates = "SEX",
    subgroups = "BMRKR2"
Here we group the levels of `BMRKR2` manually, and we add a stratification variable `STRATA1`. We also here use a continuous variable `EOSDY` which is then binarized internally (response is defined as this variable being larger than 500).

```r
df_grouped <- extract_rsp_biomarkers(
  variables = list(
    rsp = "EOSDY",
    biomarkers = c("BMRKR1", "AGE"),
    covariates = "SEX",
    subgroups = "BMRKR2",
    strat = "STRATA1"
  ),
  data = adrs_f,
  groups_lists = list(
    BMRKR2 = list(
      "low" = "LOW",
      "low/medium" = c("LOW", "MEDIUM"),
      "low/medium/high" = c("LOW", "MEDIUM", "HIGH")
    )
  ),
  control = control_logistic(
    response_definition = "I(response > 500)"
  )
)
```

---

**extract_rsp_subgroups**  *Prepares Response Data for Population Subgroups in Data Frames*

**Description**

[Stable]

Prepares response rates and odds ratios for population subgroups in data frames. Simple wrapper for `h_odds_ratio_subgroups_df()` and `h_proportion_subgroups_df()`. Result is a list of two data.frames: `prop` and `or`. `variables` corresponds to the names of variables found in data, passed as a named list and requires elements `rsp`, `arm` and optionally `subgroups` and `strat`. `groups_lists` optionally specifies groupings for `subgroups` variables.

**Usage**

```r
extract_rsp_subgroups(
  variables,
  data,
)```
extract_rsp_subgroups


groups_lists = list(),
conf_level = 0.95,
method = NULL,
label_all = "All Patients"
)

Arguments

variables (named list of string)
list of additional analysis variables.
data (data.frame)
the dataset containing the variables to summarize.
groups_lists (named list of list)
optionally contains for each subgroups variable a list, which specifies the new
group levels via the names and the levels that belong to it in the character vectors
that are elements of the list.
conf_level (proportion)
confidence level of the interval.
method (string)
specifies the test used to calculate the p-value for the difference between two
proportions. For options, see s_test_proportion_diff(). Default is NULL so
no test is performed.
label_all (string)
label for the total population analysis.

Value

A named list of two elements:

- prop: A data.frame containing columns arm, n, n_rsp, prop, subgroup, var, var_label,
and row_type.
- or: A data.frame containing columns arm, n_tot, or, lcl, ucl, conf_level, subgroup,
var, var_label, and row_type.

See Also

response_subgroups

Examples

library(dplyr)
library(forcats)

adrs <- tern_ex_adrs
adrs_labels <- formatters::var_labels(adrs)

adrs_f <- adrs %>%
  filter(PARAMCD == "BESRSPI") %>%
  filter(ARM %in% c("A: Drug X", "B: Placebo")) %>%
droplevels() %>%
  mutate(
    # Reorder levels of factor to make the placebo group the reference arm.
    ARM = fct_relevel(ARM, "B: Placebo"),
    rsp = AVALC == "CR"
  )
formatters::var_labels(adrs_f) <- c(adrs_labels, "Response")

# Unstratified analysis.
df <- extract_rsp_subgroups(
  variables = list(rsp = "rsp", arm = "ARM", subgroups = c("SEX", "BMRKR2")),
  data = adrs_f
)
df

# Stratified analysis.
df_strat <- extract_rsp_subgroups(
  variables = list(rsp = "rsp", arm = "ARM", subgroups = c("SEX", "BMRKR2"), strat = "STRATA1"),
  data = adrs_f
)
df_strat

# Grouping of the BMRKR2 levels.
df_grouped <- extract_rsp_subgroups(
  variables = list(rsp = "rsp", arm = "ARM", subgroups = c("SEX", "BMRKR2")),
  data = adrs_f,
  groups_lists = list(
    BMRKR2 = list(
      "low" = "LOW",
      "low/medium" = c("LOW", "MEDIUM"),
      "low/medium/high" = c("LOW", "MEDIUM", "HIGH")
    )
  )
)
df_grouped

---

**extract_survival_biomarkers**

*Prepares Survival Data Estimates for Multiple Biomarkers in a Single Data Frame*

**Description**

[Stable]

Prepares estimates for number of events, patients and median survival times, as well as hazard ratio estimates, confidence intervals and p-values, for multiple biomarkers across population subgroups in a single data frame. `variables` corresponds to the names of variables found in `data`, passed as a named list and requires elements `tte`, `is_event`, `biomarkers` (vector of continuous biomarker variables), and optionally `subgroups` and `strat`. `groups_lists` optionally specifies groupings for subgroups variables.
extract_survival_biomarkers

Usage

extract_survival_biomarkers(
  variables,
  data,
  groups_lists = list(),
  control = control_coxreg(),
  label_all = "All Patients"
)

Arguments

variables (named list of string)
  list of additional analysis variables.

data (data.frame)
  the dataset containing the variables to summarize.

groups_lists (named list of list)
  optionally contains for each subgroups variable a list, which specifies the new
group levels via the names and the levels that belong to it in the character vectors
  that are elements of the list.

control (list)
  a list of parameters as returned by the helper function control_coxreg().

label_all (string)
  label for the total population analysis.

Value

A data.frame with columns biomarker, biomarker_label, n_tot, n_tot_events, median, hr,
lcl, ucl, conf_level, pval, pval_label, subgroup, var, var_label, and row_type.

See Also

h_coxreg_mult_cont_df() which is used internally, tabulate_survival_biomarkers().

Examples

# Typical analysis of two continuous biomarkers `BMRKR1` and `AGE`,
# in multiple regression models containing one covariate `RACE`,
# as well as one stratification variable `STRATA1`. The subgroups
# are defined by the levels of `BMRKR2`.

library(dplyr)

adtte <- tern_ex_adtte
adtte_labels <- formatters::var_labels(adtte)

adtte_f <- adtte %>%
  filter(PARAMCD == "OS") %>%
motate(
  AVALU = as.character(AVALU),
is_event = CNSR == 0
)
labels <- c("AVALU" = adtte_labels["AVALU"], "is_event" = "Event Flag")
formatters::var_labels(adtte_f)[names(labels)] <- labels

df <- extract_survival_biomarkers(
  variables = list(
    tte = "AVAL",
    is_event = "is_event",
    biomarkers = c("BMRKR1", "AGE"),
    strata = "STRATA1",
    covariates = "SEX",
    subgroups = "BMRKR2"
  ),
  data = adtte_f
)
df

# Here we group the levels of `BMRKR2` manually.
df_grouped <- extract_survival_biomarkers(
  variables = list(
    tte = "AVAL",
    is_event = "is_event",
    biomarkers = c("BMRKR1", "AGE"),
    strata = "STRATA1",
    covariates = "SEX",
    subgroups = "BMRKR2"
  ),
  data = adtte_f,
  groups_lists = list(
    BMRKR2 = list(
      "low" = "LOW",
      "low/medium" = c("LOW", "MEDIUM"),
      "low/medium/high" = c("LOW", "MEDIUM", "HIGH")
    )
  )
)
df_grouped

---

**extract_survival_subgroups**

*Prepares Survival Data for Population Subgroups in Data Frames*

**Description**

[Stable]

Prepares estimates of median survival times and treatment hazard ratios for population subgroups in data frames. Simple wrapper for *h_survtime_subgroups_df()* and *h_coxph_subgroups_df()*.

Result is a list of two data.frames: survtime and hr. variables corresponds to the names of
variables found in data, passed as a named list and requires elements tte, is_event, arm and optionally subgroups and strat. groups_lists optionally specifies groupings for subgroups variables.

Usage

```r
extract_survival_subgroups(
  variables,
  data,
  groups_lists = list(),
  control = control_coxph(),
  label_all = "All Patients"
)
```

Arguments

- **variables** (named list of string)
  list of additional analysis variables.
- **data** (data.frame)
  the dataset containing the variables to summarize.
- **groups_lists** (named list of list)
  optionally contains for each subgroups variable a list, which specifies the new group levels via the names and the levels that belong to it in the character vectors that are elements of the list.
- **control** (list)
  parameters for comparison details, specified by using the helper function `control_coxph()`. Some possible parameter options are:
  - **pval_method** (string)
    p-value method for testing hazard ratio = 1. Default method is "log-rank" which comes from `survival::survdiff()`, can also be set to "wald" or "likelihood" (from `survival::coxph()`).  
  - **ties** (string)
    specifying the method for tie handling. Default is "efron", can also be set to "breslow" or "exact". See more in `survival::coxph()`
- **conf_level** (proportion)
  confidence level of the interval for HR.
- **label_all** (string)
  label for the total population analysis.

Value

A named list of two elements:
- **survtime**: A data.frame containing columns arm, n, n_events, median, subgroup, var, var_label, and row_type.
- **hr**: A data.frame containing columns arm, n_tot, n_tot_events, hr, lcl, ucl, conf_level, pval, pval_label, subgroup, var, var_label, and row_type.
See Also

survival_duration_subgroups

Examples

library(dplyr)
library(forcats)

adtte <- tern_ex_adtte
adtte_labels <- formatters::var_labels(adtte)

adtte_f <- adtte %>%
  filter(
    PARAMCD == "OS",
    ARM %in% c("B: Placebo", "A: Drug X"),
    SEX %in% c("M", "F")
  ) %>%
  mutate(
    # Reorder levels of ARM to display reference arm before treatment arm.
    ARM = droplevels(fct_relevel(ARM, "B: Placebo")),
    SEX = droplevels(SEX),
    AVALU = as.character(AVALU),
    is_event = CNSR == 0
  )

labels <- c(
  "ARM" = adtte_labels["ARM"],
  "SEX" = adtte_labels["SEX"],
  "AVALU" = adtte_labels["AVALU"],
  "is_event" = "Event Flag"
)
formatters::var_labels(adtte_f)[names(labels)] <- labels

df <- extract_survival_subgroups(
  variables = list(
    tte = "AVAL",
    is_event = "is_event",
    arm = "ARM", subgroups = c("SEX", "BMRKR2")
  ),
  data = adtte_f)

df

df_grouped <- extract_survival_subgroups(
  variables = list(
    tte = "AVAL",
    is_event = "is_event",
    arm = "ARM", subgroups = c("SEX", "BMRKR2")
  ),
  data = adtte_f,
  groups_lists = list(
    BMRKR2 = list(
      "low" = "LOW",
      "high" = "HIGH")
  )
)
extreme_format

Formatting Extreme Values

Description

[Stable]

rtables formatting functions that handle extreme values.

Usage

h_get_format_threshold(digits = 2L)

h_format_threshold(x, digits = 2L)

Arguments

digits

(integer)

number of decimal places to display.

x

(number)

value to format.

Details

For each input, apply a format to the specified number of digits. If the value is below a threshold, it returns "<0.01" e.g. if the number of digits is 2. If the value is above a threshold, it returns ">999.99" e.g. if the number of digits is 2. If it is zero, then returns "0.00".

Value

- h_get_format_threshold() returns a list of 2 elements: threshold, with low and high thresholds, and format_string, with thresholds formatted as strings.

- h_format_threshold() returns the given value, or if the value is not within the digit threshold the relation of the given value to the digit threshold, as a formatted string.

Functions

- h_get_format_threshold(): Internal helper function to calculate the threshold and create formatted strings used in Formatting Functions. Returns a list with elements threshold and format_string.

- h_format_threshold(): Internal helper function to apply a threshold format to a value. Creates a formatted string to be used in Formatting Functions.
See Also

Other formatting functions: format_count_fraction_fixed_dp(), format_count_fraction(), format_extreme_values_ci(), format_extreme_values(), format_fraction_fixed_dp(), format_fraction_threshold(), format_fraction(), format_xx(), formatting_functions

Examples

h_get_format_threshold(2L)

h_format_threshold(0.001)
h_format_threshold(1000)

date

---

Description

Simulated CDISC Data for Examples

Usage

tern_ex_adsl
tern_ex_adae
tern_ex_adlb
tern_ex_adpp
tern_ex_adrs
tern_ex_adtte

Format

rds (data.frame)
An object of class tbl_df (inherits from tbl, data.frame) with 200 rows and 21 columns.
An object of class tbl_df (inherits from tbl, data.frame) with 541 rows and 42 columns.
An object of class tbl_df (inherits from tbl, data.frame) with 4200 rows and 50 columns.
An object of class tbl_df (inherits from tbl, data.frame) with 522 rows and 25 columns.
An object of class tbl_df (inherits from tbl, data.frame) with 1600 rows and 29 columns.
An object of class tbl_df (inherits from tbl, data.frame) with 1000 rows and 28 columns.
Functions

- tern_ex_adsl: ADSL data
- tern_ex_adae: ADAE data
- tern_ex_adlb: ADLB data
- tern_ex_adpp: ADPP data
- tern_ex_adrs: ADRS data
- tern_ex_adtte: ADTTE data

fct_collapse_only

Collapsing of Factor Levels and Keeping Only Those New Group Levels

Description

[Stable]
This collapses levels and only keeps those new group levels, in the order provided. The returned factor has levels in the order given, with the possible missing level last (this will only be included if there are missing values).

Usage

\[ \text{fct\_collapse\_only(.f, \ldots, .na\_level = "<Missing>"\)} \]

Arguments

- \( .f \) (factor or character)
  original vector.
- \( \ldots \) (named character vectors)
  levels in each vector provided will be collapsed into the new level given by the respective name.
- \( .\text{na\_level} \) (string)
  which level to use for other levels, which should be missing in the new factor. Note that this level must not be contained in the new levels specified in \( \ldots \).

Value

A modified factor with collapsed levels. Values and levels which are not included in the given character vector input will be set to the missing level \( .\text{na\_level} \).

Note

Any existing NAs in the input vector will not be replaced by the missing level. If needed, \texttt{explicit_na()} can be called separately on the result.
fct_discard

See Also
forcats::fctCollapse(), forcats::fct_relevel() which are used internally.

Examples
fct_collapse_only(factor(c("a", "b", "c", "d")), TRT = "b", CTRL = c("c", "d"))

---

**fct_discard**

*Discard Certain Levels from a Factor*

**Description**

[Stable]
This discards the observations as well as the levels specified from a factor.

**Usage**

fct_discard(x, discard)

**Arguments**

- **x** (factor)
  the original factor.
- **discard** (character)
  which levels to discard.

**Value**

A modified factor with observations as well as levels from discard dropped.

**Examples**

fct_discard(factor(c("a", "b", "c")), "c")
**fct_explicit_na_if**  
*Insertion of Explicit Missings in a Factor*

### Description

[Stable]

This inserts explicit missings in a factor based on a condition. Additionally, existing `NA` values will be explicitly converted to given `na_level`.

### Usage

```r
fct_explicit_na_if(x, condition, na_level = "<Missing>")
```

### Arguments

- `x`  
  (factor)  
  the original factor.

- `condition`  
  (logical)  
  where to insert missings.

- `na_level`  
  (string)  
  which level to use for missings.

### Value

A modified factor with inserted and existing `NA` converted to `na_level`.

### See Also

`forcats::fct_na_value_to_level()` which is used internally.

### Examples

```r
fct_explicit_na_if(factor(c("a", "b", NA)), c(TRUE, FALSE, FALSE))
```

---

**fit_coxreg**  
*Fits for Cox Proportional Hazards Regression*

### Description

[Stable]

Fitting functions for univariate and multivariate Cox regression models.
Usage

```
fit_coxreg_univar(variables, data, at = list(), control = control_coxreg())
fit_coxreg_multivar(variables, data, control = control_coxreg())
```

Arguments

- **variables** (list)
  a named list corresponds to the names of variables found in data, passed as a
  named list and corresponding to time, event, arm, strata, and covariates
  terms. If arm is missing from variables, then only Cox model(s) including
  the covariates will be fitted and the corresponding effect estimates will be
  tabulated later.

- **data** (data.frame)
  the dataset containing the variables to fit the models.

- **at** (list of numeric)
  when the candidate covariate is a numeric, use at to specify the value of the
  covariate at which the effect should be estimated.

- **control** (list)
  a list of parameters as returned by the helper function `control_coxreg()`.

Value

- **fit_coxreg_univar()** returns a `coxreg.univar` class object which is a named list with 5
  elements:
  - mod: Cox regression models fitted by `survival::coxph()`.
  - data: The original data frame input.
  - control: The original control input.
  - vars: The variables used in the model.
  - at: Value of the covariate at which the effect should be estimated.

- **fit_coxreg_multivar()** returns a `coxreg.multivar` class object which is a named list with
  4 elements:
  - mod: Cox regression model fitted by `survival::coxph()`.
  - data: The original data frame input.
  - control: The original control input.
  - vars: The variables used in the model.

Functions

- **fit_coxreg_univar()**: Fit a series of univariate Cox regression models given the inputs.
- **fit_coxreg_multivar()**: Fit a multivariate Cox regression model.

Note

When using `fit_coxreg_univar` there should be two study arms.
See Also

`h_cox_regression` for relevant helper functions, `cox_regression`.

Examples

```r
library(survival)

set.seed(1, kind = "Mersenne-Twister")

# Testing dataset [survival::bladder].
dta_bladder <- with(
  data = bladder[bladder$enum < 5, ],
data.frame(
    time = stop,
    status = event,
    armcd = as.factor(rx),
    covar1 = as.factor(enum),
    covar2 = factor(
      sample(as.factor(enum)),
      levels = 1:4, labels = c("F", "F", "M", "M")
    ) )
)

labels <- c("armcd" = "ARM", "covar1" = "A Covariate Label", "covar2" = "Sex (F/M)"
)
formatters::var_labels(dta_bladder)[names(labels)] <- labels
dta_bladder$age <- sample(20:60, size = nrow(dta_bladder), replace = TRUE)

plot(
  survfit(Surv(time, status) ~ armcd + covar1, data = dta_bladder),
  lty = 2:4,
  xlab = "Months",
  col = c("blue1", "blue2", "blue3", "blue4", "red1", "red2", "red3", "red4")
)

# fit_coxreg_univar

## Cox regression: arm + 1 covariate.
mod1 <- fit_coxreg_univar(
  variables = list(
    time = "time", event = "status", arm = "armcd",
    covariates = "covar1"
  ),
  data = dta_bladder,
  control = control_coxreg(conf_level = 0.91)
)

## Cox regression: arm + 1 covariate + interaction, 2 candidate covariates.
mod2 <- fit_coxreg_univar(
  variables = list(
    time = "time", event = "status", arm = "armcd",
    covariates = c("covar1", "covar2")
  ),
  interaction = TRUE,
  data = dta_bladder,
  control = control_coxreg(conf_level = 0.91)
)
```

data = dta_bladder,
control = control_coxreg(conf_level = 0.91, interaction = TRUE)
)

## Cox regression: arm + 1 covariate, stratified analysis.
mod3 <- fit_coxreg_univar(
  variables = list(
    time = "time", event = "status", arm = "armcd", strata = "covar2",
    covariates = c("covar1")
  ),
  data = dta_bladder,
  control = control_coxreg(conf_level = 0.91)
)

## Cox regression: no arm, only covariates.
mod4 <- fit_coxreg_univar(
  variables = list(
    time = "time", event = "status",
    covariates = c("covar1", "covar2")
  ),
  data = dta_bladder
)

# fit_coxreg_multivar

## Cox regression: multivariate Cox regression.
multivar_model <- fit_coxreg_multivar(
  variables = list(
    time = "time", event = "status", arm = "armcd",
    covariates = c("covar1", "covar2")
  ),
  data = dta_bladder
)

# Example without treatment arm.
multivar_covs_model <- fit_coxreg_multivar(
  variables = list(
    time = "time", event = "status",
    covariates = c("covar1", "covar2")
  ),
  data = dta_bladder
)

---

**fit_logistic**  
*Fit for Logistic Regression*

**Description**

*[Stable]*

Fit a (conditional) logistic regression model.
Usage

```r
fit_logistic(
  data,
  variables = list(response = "Response", arm = "ARMCD", covariates = NULL, interaction =
    NULL, strata = NULL),
  response_definition = "response"
)
```

Arguments

- **data** *(data.frame)*
  the data frame on which the model was fit.
- **variables** *(named list of string)*
  list of additional analysis variables.
- **response_definition** *(string)*
  the definition of what an event is in terms of response. This will be used when fitting the (conditional) logistic regression model on the left hand side of the formula.

Value

A fitted logistic regression model.

Model Specification

The variables list needs to include the following elements:

- **arm**: Treatment arm variable name.
- **response**: The response arm variable name. Usually this is a 0/1 variable.
- **covariates**: This is either NULL (no covariates) or a character vector of covariate variable names.
- **interaction**: This is either NULL (no interaction) or a string of a single covariate variable name already included in covariates. Then the interaction with the treatment arm is included in the model.

Examples

```r
library(dplyr)

adrs_f <- tern_ex_adrs %>%
  filter(PARAMCD == "BESRSPI") %>%
  filter(RACE %in% c("ASIAN", "WHITE", "BLACK OR AFRICAN AMERICAN")) %>%
  mutate(
    Response = case_when(AVALC %in% c("PR", "CR") ~ 1, TRUE ~ 0),
    RACE = factor(RACE),
    SEX = factor(SEX)
  )

formatters::var_labels(adrs_f) <- c(formatters::var_labels(tern_ex_adrs), Response = "Response")
```
The Subgroup Treatment Effect Pattern (STEP) Fit for Binary (Response) Outcome

**Description**

**[Stable]**

This fits the Subgroup Treatment Effect Pattern logistic regression models for a binary (response) outcome. The treatment arm variable must have exactly 2 levels, where the first one is taken as reference and the estimated odds ratios are for the comparison of the second level vs. the first one.

The (conditional) logistic regression model which is fit is:

\[
\text{response} \sim \text{arm} \times \text{poly(biomarker, degree)} + \text{covariates} + \text{strata(strata)}
\]

where degree is specified by `control_step()`.

**Usage**

```r
fit_rsp_step(variables, data, control = c(control_step(), control_logistic()))
```

**Arguments**

- **variables** (named list of character)
  list of analysis variables: needs response, arm, biomarker, and optional covariates and strata.

- **data** (data.frame)
  the dataset containing the variables to summarize.

- **control** (named list)
  combined control list from `control_step()` and `control_logistic()`.
**Value**

A matrix of class `step`. The first part of the columns describe the subgroup intervals used for the biomarker variable, including where the center of the intervals are and their bounds. The second part of the columns contain the estimates for the treatment arm comparison.

**Note**

For the default degree 0 the biomarker variable is not included in the model.

**See Also**

`control_step()` and `control_logistic()` for the available customization options.

**Examples**

```r
# Testing dataset with just two treatment arms.
library(survival)
library(dplyr)

adrs_f <- tern_ex_adrs %>%
  filter(
    PARAMCD == "BESRSP1",
    ARM %in% c("B: Placebo", "A: Drug X")
  ) %>%
  mutate(
    # Reorder levels of ARM to have Placebo as reference arm for Odds Ratio calculations.
    ARM = droplevels(forcats::fct_relevel(ARM, "B: Placebo")),
    RSP = case_when(AVALC %in% c("PR", "CR") ~ 1, TRUE ~ 0),
    SEX = factor(SEX)
  )

variables <- list(
  arm = "ARM",
  biomarker = "BMRKR1",
  covariates = "AGE",
  response = "RSP"
)

# Fit default STEP models: Here a constant treatment effect is estimated in each subgroup.
# We use a large enough bandwidth to avoid too small subgroups and linear separation in those.
step_matrix <- fit_rsp_step(
  variables = variables,
  data = adrs_f,
  control = c(control_logistic(), control_step(bandwidth = 0.5))
)

dim(step_matrix)
head(step_matrix)

# Specify different polynomial degree for the biomarker interaction to use more flexible local
# models. Or specify different logistic regression options, including confidence level.
step_matrix2 <- fit_rsp_step(
  variables = variables,
  control = c(control_logistic(), control_logistic(conf.level = 0.95))
)

head(step_matrix2)
```


data = adrs_f,
control = c(control_logistic(conf_level = 0.9), control_step(bandwidth = 0.6, degree = 1))
)

# Use a global constant model. This is helpful as a reference for the subgroup models.
step_matrix3 <- fit_rsp_step(
  variables = variables,
  data = adrs_f,
  control = c(control_logistic(), control_step(bandwidth = NULL, num_points = 2L))
)

# It is also possible to use strata, i.e. use conditional logistic regression models.
variables2 <- list(
  arm = "ARM",
  biomarker = "BMRKR1",
  covariates = "AGE",
  response = "RSP",
  strata = c("STRATA1", "STRATA2")
)

step_matrix4 <- fit_rsp_step(
  variables = variables2,
  data = adrs_f,
  control = c(control_logistic(), control_step(bandwidth = 0.6))
)

---

fit_survival_step  

Subgroup Treatment Effect Pattern (STEP) Fit for Survival Outcome

**Description**

[Stable]

This fits the Subgroup Treatment Effect Pattern models for a survival outcome. The treatment arm variable must have exactly 2 levels, where the first one is taken as reference and the estimated hazard ratios are for the comparison of the second level vs. the first one.

The model which is fit is:

\[ \text{Surv}(\text{time}, \text{event}) \sim \text{arm} \times \text{poly(\text{biomarker}, \text{degree}) + covariates + strata(\text{strata})} \]

where degree is specified by control_step().

**Usage**

```r
fit_survival_step(
  variables,  
  data,  
  control = c(control_step(), control_coxph())
)
```
Arguments

- **variables** (named list of character)
  - list of analysis variables: needs time, event, arm, biomarker, and optional covariates and strata.

- **data** (data.frame)
  - the dataset containing the variables to summarize.

- **control** (named list)
  - combined control list from `control_step()` and `control_coxph()`.

Value

A matrix of class `step`. The first part of the columns describe the subgroup intervals used for the biomarker variable, including where the center of the intervals are and their bounds. The second part of the columns contain the estimates for the treatment arm comparison.

Note

For the default degree 0 the biomarker variable is not included in the model.

See Also

- `control_step()` and `control_coxph()` for the available customization options.

Examples

```r
# Testing dataset with just two treatment arms.
library(dplyr)

adtte_f <- tern_ex_adtte %>%
  filter(
    PARAMCD == "OS",
    ARM %in% c("B: Placebo", "A: Drug X")
  ) %>%
  mutate(
    # Reorder levels of ARM to display reference arm before treatment arm.
    ARM = droplevels(forcats::fct_relevel(ARM, "B: Placebo")),
    is_event = CNSR == 0
  )
labels <- c("ARM" = "Treatment Arm", "is_event" = "Event Flag")
formatters::var_labels(adtte_f)[names(labels)] <- labels

variables <- list(
  arm = "ARM",
  biomarker = "BMRKR1",
  covariates = c("AGE", "BMRKR2"),
  event = "is_event",
  time = "AVAL"
)

# Fit default STEP models: Here a constant treatment effect is estimated in each subgroup.
```
step_matrix <- fit_survival_step(
    variables = variables,
    data = adtte_f
)
dim(step_matrix)
head(step_matrix)

# Specify different polynomial degree for the biomarker interaction to use more flexible local
# models. Or specify different Cox regression options.
step_matrix2 <- fit_survival_step(
    variables = variables,
    data = adtte_f,
    control = c(control_coxph(conf_level = 0.9), control_step(degree = 2))
)

# Use a global model with cubic interaction and only 5 points.
step_matrix3 <- fit_survival_step(
    variables = variables,
    data = adtte_f,
    control = c(control_coxph(), control_step(bandwidth = NULL, degree = 3, num_points = 5L))
)
```

---

**forest_viewport**  
*Create a Viewport Tree for the Forest Plot*

**Description**

Create a Viewport Tree for the Forest Plot

**Usage**

```
forest_viewport(
    tbl,
    width_row_names = NULL,
    width_columns = NULL,
    width_forest = grid::unit(1, "null"),
    gap_column = grid::unit(1, "lines"),
    gap_header = grid::unit(1, "lines"),
    mat_form = NULL
)
```

**Arguments**

- `tbl` *(rtable)*
- `width_row_names` *(grid::unit)*
  - **Width of row names**
width_columns (grid::unit)
Width of column spans

width_forest (grid::unit)
Width of the forest plot

gap_column (grid::unit)
Gap width between the columns

gap_header (grid::unit)
Gap width between the header

mat_form
Matrix print form of the table

Value
A viewport tree.

Examples

library(grid)

tbl <- rtable(
    header = rheader(
        rrow("", "E", rcell("CI", colspan = 2)),
        rrow("", "A", "B", "C")
    ),
    rrow("row 1", 1, 0.8, 1.1),
    rrow("row 2", 1.4, 0.8, 1.6),
    rrow("row 3", 1.2, 0.8, 1.2)
)

v <- forest_viewport(tbl)

grid::grid.newpage()
showViewport(v)

formatting_functions Formatting Functions

Description

[Stable]
See below for the list of formatting functions created in tern to work with rtables.

Other available formats can be listed via formatters::list_valid_format_labels(). Additional custom formats can be created via the formatters::sprintf_format() function.
See Also

Other formatting functions: `extreme_format`, `format_count_fraction_fixed_dp()`, `format_count_fraction()`, `format_extreme_values_ci()`, `format_extreme_values()`, `format_fraction_fixed_dp()`, `format_fraction_threshold()`, `format_fraction()`, `format_xx()`, `formatting_functions`

---

**format_count_fraction**  Formatting Count and Fraction

**Description**

[Stable]

Formats a count together with fraction with special consideration when count is 0.

**Usage**

`format_count_fraction(x, ...)`

**Arguments**

- `x`  
  (integer)  
  vector of length 2, count and fraction.  
  ...  
  required for rtables interface.

**Value**

A string in the format `count (fraction %)`. If count is 0, the format is 0.

**See Also**

Other formatting functions: `extreme_format`, `format_count_fraction_fixed_dp()`, `format_extreme_values_ci()`, `format_extreme_values()`, `format_fraction_fixed_dp()`, `format_fraction_threshold()`, `format_fraction()`, `format_xx()`, `formatting_functions`

**Examples**

```r
format_count_fraction(x = c(2, 0.6667))
format_count_fraction(x = c(0, 0))
```
Description

[Experimental]
Formats a count together with fraction with special consideration when count is 0.

Usage

format_count_fraction_fixed_dp(x, ...)

Arguments

x (integer)
vector of length 2, count and fraction.
...
required for rtables interface.

Value

A string in the format count (fraction %). If count is 0, the format is 0.

See Also

Other formatting functions: extreme_format, format_count_fraction(), format_extreme_values_ci(),
format_extreme_values(), format_fraction_fixed_dp(), format_fraction_threshold(),
format_fraction(), format_xx(), formatting_functions

Examples

format_count_fraction_fixed_dp(x = c(2, 0.6667))
format_count_fraction_fixed_dp(x = c(2, 0.5))
format_count_fraction_fixed_dp(x = c(0, 0))

format_extreme_values

Formatting a Single Extreme Value

Description

[Stable]
Create Formatting Function for a single extreme value.
Usage

format_extreme_values(digits = 2L)

Arguments

digits (integer)
number of decimal places to display.

Value

An rtables formatting function that uses threshold digits to return a formatted extreme value.

See Also

Other formatting functions: extreme_format, format_count_fraction_fixed_dp(), format_count_fraction(), format_extreme_values_ci(), format_fraction_fixed_dp(), format_fraction_threshold(), format_fraction(), format_xx(), formatting_functions

Examples

format_fun <- format_extreme_values(2L)
format_fun(x = 0.127)
format_fun(x = Inf)
format_fun(x = 0)
format_fun(x = 0.009)

format_extreme_values_ci

Formatting Extreme Values Part of a Confidence Interval

Description

[Stable]

Formatting Function for extreme values part of a confidence interval. Values are formatted as e.g. “(xx.xx, xx.xx)” if the number of digits is 2.

Usage

format_extreme_values_ci(digits = 2L)

Arguments

digits (integer)
number of decimal places to display.
**Value**

An rtables formatting function that uses threshold digits to return a formatted extreme values confidence interval.

**See Also**

Other formatting functions: `extreme_format`, `format_count_fraction_fixed_dp()`, `format_count_fraction()`, `format_extreme_values()`, `format_fraction_fixed_dp()`, `format_fraction_threshold()`, `format_fraction()`, `format_xx()`, `formatting_functions`  

**Examples**

```r
format_fun <- format_extreme_values_ci(2L)
format_fun(x = c(0.127, Inf))
format_fun(x = c(0, 0.009))
```

---

**format_fraction**  
**Formatting Fraction and Percentage**

**Description**

[Stable]

Formats a fraction together with ratio in percent.

**Usage**

`format_fraction(x, ...)`

**Arguments**

- `x`  
  (integer)  
  with elements `num` and `denom`.  
- `...`  
  required for rtables interface.

**Value**

A string in the format `num / denom (ratio %)`. If `num` is 0, the format is `num / denom`.

**See Also**

Other formatting functions: `extreme_format`, `format_count_fraction_fixed_dp()`, `format_count_fraction()`, `format_extreme_values()`, `format_extreme_values_ci()`, `format_fraction_fixed_dp()`, `format_fraction_threshold()`, `format_fraction()`, `format_xx()`, `formatting_functions`
format_fraction_fixed_dp

Formatting Fraction and Percentage with Fixed Single Decimal Place

Description

[Stable]

Formats a fraction together with ratio in percent with fixed single decimal place. Includes trailing zero in case of whole number percentages to always keep one decimal place.

Usage

format_fraction_fixed_dp(x, ...)

Arguments

x (integer)
with elements num and denom.

... required for rtables interface.

Value

A string in the format num / denom (ratio %). If num is 0, the format is num / denom.

See Also

Other formatting functions: extreme_format, format_count_fraction_fixed_dp(), format_count_fraction(), format_extreme_values_ci(), format_extreme_values(), format_fraction_threshold(), format_fraction(), format_xx(), formatting_functions

Examples

format_fraction_fixed_dp(x = c(num = 1L, denom = 2L))
format_fraction_fixed_dp(x = c(num = 1L, denom = 4L))
format_fraction_fixed_dp(x = c(num = 0L, denom = 3L))
Description

[Stable]

Formats a fraction when the second element of the input \( x \) is the fraction. It applies a lower threshold, below which it is just stated that the fraction is smaller than that.

Usage

format_fraction_threshold(threshold)

Arguments

threshold (proportion) lower threshold.

Value

An \texttt{rtables} formatting function that takes numeric input \( x \) where the second element is the fraction that is formatted. If the fraction is above or equal to the threshold, then it is displayed in percentage. If it is positive but below the threshold, it returns, e.g. "<1" if the threshold is 0.01. If it is zero, then just "0" is returned.

See Also

Other formatting functions: \texttt{extreme_format}, \texttt{format_count_fraction_fixed_dp()}, \texttt{format_count_fraction()}, \texttt{format_extreme_values_ci()}, \texttt{format_extreme_values()}, \texttt{format_fraction_fixed_dp()}, \texttt{format_fraction()}, \texttt{format_xx()}, \texttt{formatting_functions}

Examples

```r
format_fun <- format_fraction_threshold(0.05)
format_fun(x = c(20, 0.1))
format_fun(x = c(2, 0.01))
format_fun(x = c(0, 0))
```
format_xx

Formatting: XX as Formatting Function

Description

Translate a string where x and dots are interpreted as number place holders, and others as formatting elements.

Usage

format_xx(str)

Arguments

str (string) (string)

template.

Value

An rtables formatting function.

See Also

Other formatting functions: extreme_format, format_count_fraction_fixed_dp(), format_count_fraction(), format_extreme_values_ci(), format_extreme_values(), format_fraction_fixed_dp(), format_fraction_threshold(), format_fraction(), formatting_functions

Examples

test <- list(c(1.658, 0.5761), c(1e1, 785.6))

z <- format_xx("xx (xx.x)"
apply(test, z)

z <- format_xx("xx.x - xx.x")
apply(test, z)

z <- format_xx("xx.x, incl. xx.x% NE")
apply(test, z)
f_conf_level

Utility function to create label for confidence interval

Description

[Stable]

Usage

f_conf_level(conf_level)

Arguments

conf_level  (proportion)
  confidence level of the interval.

Value

A string.

f_pval

Utility function to create label for p-value

Description

[Stable]

Usage

f_pval(test_mean)

Arguments

test_mean  (number)
  mean value to test under the null hypothesis.

Value

A string.
get_smooths

Smooth Function with Optional Grouping

Description

[Stable]

This produces loess smoothed estimates of y with Student confidence intervals.

Usage

get_smooths(df, x, y, groups = NULL, level = 0.95)

Arguments

df (data.frame)
data set containing all analysis variables.
x (character)
value with x column name.
y (character)
value with y column name.
groups (character)
vector with optional grouping variables names.
level (numeric)
level of confidence interval to use (0.95 by default).

Value

A data.frame with original x, smoothed y, ylow, and yhigh, and optional groups variables formatted as factor type.

groups_list_to_df

Convert List of Groups to Data Frame

Description

This converts a list of group levels into a data frame format which is expected by rtables::add_combo_levels().

Usage

groups_list_to_df(groups_list)

Arguments

groups_list (named list of character)
specifies the new group levels via the names and the levels that belong to it in the character vectors that are elements of the list.
**g_forest**

Value

`tibble::tibble()` in the required format.

Examples

```r
grade_groups <- list(
  "Any Grade (%)" = c("1", "2", "3", "4", "5"),
  "Grade 3-4 (%)" = c("3", "4"),
  "Grade 5 (%)" = "5"
)
groups_list_to_df(grade_groups)
```

---

**g_forest**

Create a Forest Plot based on a Table

Description

[Stable]

Usage

```r
g_forest(
  tbl,
  col_x = attr(tbl, "col_x"),
  col_ci = attr(tbl, "col_ci"),
  vline = 1,
  forest_header = attr(tbl, "forest_header"),
  xlim = c(0.1, 10),
  logx = TRUE,
  x_at = c(0.1, 1, 10),
  width_row_names = NULL,
  width_columns = NULL,
  width_forest = grid::unit(1, "null"),
  col_symbol_size = attr(tbl, "col_symbol_size"),
  col = getOption("ggplot2.discrete.colour")[1],
  draw = TRUE,
  newpage = TRUE
)
```

Arguments

- `tbl` (rtable)
- `col_x` (integer)

column index with estimator. By default tries to get this from `tbl` attribute `col_x`, otherwise needs to be manually specified.
col_ci (integer) column index with confidence intervals. By default tries to get this from tbl attribute col_ci, otherwise needs to be manually specified.

vline (number) x coordinate for vertical line, if NULL then the line is omitted.

forest_header (character, length 2) text displayed to the left and right of vline, respectively. If vline = NULL then forest_header needs to be NULL too. By default tries to get this from tbl attribute forest_header.

xlim (numeric) limits for x axis.

logx (flag) show the x-values on logarithm scale.

x_at (numeric) x-tick locations, if NULL they get automatically chosen.

width_row_names (unit) width for row names. If NULL the widths get automatically calculated. See grid::unit().

width_columns (unit) widths for the table columns. If NULL the widths get automatically calculated. See grid::unit().

width_forest (unit) width for the forest column. If NULL the widths get automatically calculated. See grid::unit().

col_symbol_size (integer) column index from tbl containing data to be used to determine relative size for estimator plot symbol. Typically, the symbol size is proportional to the sample size used to calculate the estimator. If NULL, the same symbol size is used for all subgroups. By default tries to get this from tbl attribute col_symbol_size, otherwise needs to be manually specified.

col (character) color(s).

draw (flag) whether the plot should be drawn.

newpage (flag) whether the plot should be drawn on a new page. Only considered if draw = TRUE is used.

Details

Create a forest plot from any rtables::rtable() object that has a column with a single value and a column with 2 values.
Value

gTree object containing the forest plot and table.

Examples

```r
library(dplyr)
library(forcats)
library(nestcolor)

adrs <- tern_ex_adrs
n_records <- 20
adrs_labels <- formatters::var_labels(adrs, fill = TRUE)
adrs <- adrs %>%
  filter(PARAMCD == "BESRSPI") %>%
  filter(ARM %in% c("A: Drug X", "B: Placebo")) %>%
  slice(seq_len(n_records)) %>%
  droplevels() %>%
  mutate(
    # Reorder levels of factor to make the placebo group the reference arm.
    ARM = fct_relevel(ARM, "B: Placebo"),
    rsp = AVALC == "CR"
  )
formatters::var_labels(adrs) <- c(adrs_labels, "Response")

df <- extract_rsp_subgroups(
  variables = list(rsp = "rsp", arm = "ARM", subgroups = c("SEX", "STRATA2")),
  data = adrs
)
# Full commonly used response table.

tbl <- basic_table() %>%
  tabulate_rsp_subgroups(df)
p <- g_forest(tbl)

draw_grob(p)
# Odds ratio only table.

tbl_or <- basic_table() %>%
  tabulate_rsp_subgroups(df, vars = c("n_tot", "or", "ci"))
tbl_or
p <- g_forest(
  tbl_or,
  forest_header = c("Comparison\nBetter", "Treatment\nBetter")
)

draw_grob(p)
# Survival forest plot example.

adtte <- tern_ex_adtte
# Save variable labels before data processing steps.
adtte_labels <- formatters::var_labels(adtte, fill = TRUE)
```
adtte_f <- adtte %>%
  filter(  
    PARAMCD == "OS",  
    ARM %in% c("B: Placebo", "A: Drug X"),  
    SEX %in% c("M", "F")  
  ) %>%
  mutate(  
    # Reorder levels of ARM to display reference arm before treatment arm.  
    ARM = droplevels(fct_relevel(ARM, "B: Placebo")),  
    SEX = droplevels(SEX),  
    AVALU = as.character(AVALU),  
    is_event = CNSR == 0  
  )

labels <- list("ARM" = adtte_labels["ARM"],  
               "SEX" = adtte_labels["SEX"],  
               "AVALU" = adtte_labels["AVALU"],  
               "is_event" = "Event Flag")

formatters::var_labels(adtte_f)[names(labels)] <- as.character(labels)

df <- extract_survival_subgroups(  
  variables = list(  
    tte = "AVAL",  
    is_event = "is_event",  
    arm = "ARM", subgroups = c("SEX", "BMRKR2")  
  ),  
  data = adtte_f
)

table_hr <- basic_table() %>%
  tabulate_survival_subgroups(df, time_unit = adtte_f$AVALU[1])

g_forest(table_hr)
# Works with any `rtable`

tbl <- rtable(  
  header = c("E", "CI", "N"),  
  rrow("", 1, c(.8, 1.2), 200),  
  rrow("", 1.2, c(1.1, 1.4), 50)
)

g_forest(  
  tbl = tbl,  
  col_x = 1,  
  col_ci = 2,  
  xlim = c(0.5, 2),  
  x_at = c(0.5, 1, 2),  
  col_symbol_size = 3
)

tbl <- rtable(  
  header = rheader(  
    rrow("", rcell("A", colspan = 2)),  
    rrow("", "c1", "c2")  
  ),  
  rrow("row 1", 1, c(.8, 1.2)),  
  rrow("row 2", 1.2, c(1.1, 1.4))
)
\texttt{g\_km}

\begin{verbatim}
g_forest( 
  tbl = tbl, 
  col_x = 1, 
  col_ci = 2, 
  xlim = c(0.5, 2), 
  x_at = c(0.5, 1, 2), 
  vline = 1, 
  forest_header = c("Hello", "World")
)
\end{verbatim}

---

**Kaplan-Meier Plot**

**Description**

[Stable]

From a survival model, a graphic is rendered along with tabulated annotation including the number of patient at risk at given time and the median survival per group.

**Usage**

\begin{verbatim}
g_km(
  df, 
  variables, 
  control_surv = control_surv_timepoint(), 
  col = NULL, 
  lty = NULL, 
  lwd = 0.5, 
  censor_show = TRUE, 
  pch = 3, 
  size = 2, 
  max_time = NULL, 
  xticks = NULL, 
  xlab = "Days", 
  yval = c("Survival", "Failure"), 
  ylab = paste(yval, "Probability"), 
  title = NULL, 
  footnotes = NULL, 
  draw = TRUE, 
  newpage = TRUE, 
  gp = NULL, 
  vp = NULL, 
  name = NULL, 
  font_size = 12, 
  ci_ribbon = FALSE, 
)
\end{verbatim}
ggtheme = nestcolor::theme_nest(),
annot_at_risk = TRUE,
annot_surv_med = TRUE,
annot_coxph = FALSE,
annot_stats = NULL,
annot_stats_vlines = FALSE,
control_coxph_pw = control_coxph(),
position_coxph = c(-0.03, -0.02),
position_surv_med = c(0.95, 0.9),
width_annots = list(surv_med = grid::unit(0.3, "npc"), coxph = grid::unit(0.4, "npc"))

Arguments

**df** (data.frame)
data set containing all analysis variables.

**variables** (named list)
variable names. Details are:

- **tte** (numeric)
  variable indicating time-to-event duration values.
- **is_event** (logical)
  event variable. TRUE if event, FALSE if time to event is censored.
- **arm** (factor)
  the treatment group variable.
- **strat** (character or NULL)
  variable names indicating stratification factors.

**control_surv** (list)
parameters for comparison details, specified by using the helper function `control_surv_timepoint()`. Some possible parameter options are:

- **conf_level** (proportion)
  confidence level of the interval for survival rate.
- **conf_type** (string)
  "plain" (default), "log", "log-log" for confidence interval type, see more in `survival::survfit()`. Note that the option "none" is no longer supported.

**col** (character)
lines colors. Length of a vector should be equal to number of strata from `survival::survfit()`.

**lty** (numeric)
line type. Length of a vector should be equal to number of strata from `survival::survfit()`.

**lwd** (numeric)
line width. Length of a vector should be equal to number of strata from `survival::survfit()`.

**censor_show** (flag)
whether to show censored.

**pch** (numeric, string)
value or character of points symbol to indicate censored cases.
size  (numeric) size of censored point, a class of unit.
max_time (numeric) maximum value to show on X axis. Only data values less than or up to this threshold value will be plotted (defaults to NULL).
xticks (numeric, number, or NULL) numeric vector of ticks or single number with spacing between ticks on the x axis. If NULL (default), `labeling::extended()` is used to determine an optimal tick position on the x axis.
lab   (string) label of x-axis.
yval  (string) value of y-axis. Options are Survival (default) and Failure probability.
ylab  (string) label of y-axis.
title (string) title for plot.
footnotes (string) footnotes for plot.
draw  (flag) whether the plot should be drawn.
newpage (flag) whether the plot should be drawn on a new page. Only considered if draw = TRUE is used.
gp    A ”gpar” object, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.
vp    a `viewport` object (or NULL).
name  a character identifier for the grob. Used to find the grob on the display list and/or as a child of another grob.
font_size (number) font size to be used.
ribbon (flag) draw the confidence interval around the Kaplan-Meier curve.
theme (theme) a graphical theme as provided by ggplot2 to control outlook of the Kaplan-Meier curve.
annot_at_risk (flag) compute and add the annotation table reporting the number of patient at risk matching the main grid of the Kaplan-Meier curve.
annot_surv_med (flag) compute and add the annotation table on the Kaplan-Meier curve estimating the median survival time per group.
annot_coxph (flag) add the annotation table from a `survival::coxph()` model.
annot_stats (string)
statistics annotations to add to the plot. Options are median (median survival follow-up time) and min (minimum survival follow-up time).

annot_stats_vlines
(flag)
add vertical lines corresponding to each of the statistics specified by annot_stats. If annot_stats is NULL no lines will be added.

control_coxph_pw
(list)
parameters for comparison details, specified by using the helper function control_coxph(). Some possible parameter options are:

• pval_method (string)
p-value method for testing hazard ratio = 1. Default method is "log-rank", can also be set to "wald" or "likelihood".

• ties (string)
method for tie handling. Default is "efron", can also be set to "breslow" or "exact". See more in survival::coxph()

• conf_level (proportion)
confidence level of the interval for HR.

position_coxph (numeric)
x and y positions for plotting survival::coxph() model.

position_surv_med (numeric)
x and y positions for plotting annotation table estimating median survival time per group.

width_annots (named list of units)
a named list of widths for annotation tables with names surv_med (median survival time table) and coxph (survival::coxph() model table), where each value is the width (in units) to implement when printing the annotation table.

Value
A grob of class gTree.

Examples

library(dplyr)
library(ggplot2)
library(survival)
library(grid)
library(nestcolor)

df <- tern_ex_adtte %>%
  filter(PARAMCD == "OS") %>%
  mutate(is_event = CNSR == 0)
variables <- list(tte = "AVAL", is_event = "is_event", arm = "ARMCD")
# g_km

**1. Example - basic option**

```r
res <- g_km(df = df, variables = variables)
res <- g_km(df = df, variables = variables, yval = "Failure")
res <- g_km(
  df = df,
  variables = variables,
  control_surv = control_surv_timepoint(conf_level = 0.9),
  col = c("grey25", "grey50", "grey75")
)
res <- g_km(df = df, variables = variables, ggtheme = theme_minimal())
res <- g_km(df = df, variables = variables, ggtheme = theme_minimal(), lty = 1:3)
res <- g_km(df = df, variables = variables, max = 2000)
res <- g_km(
  df = df,
  variables = variables,
  annot_stats = c("min", "median"),
  annot_stats_vlines = TRUE
)
```

**2. Example - Arrange several KM curve on a single graph device**

---

**2.1 Use case: A general graph on the top, a zoom on the bottom.**

```r
grid.newpage()
lyt <- grid.layout(nrow = 2, ncol = 1) %>%
  viewport(layout = .) %>%
  pushViewport(
    res <- g_km(
      df = df, variables = variables, newpage = FALSE, annot_surv_med = FALSE,
      vp = viewport(layout.pos.row = 1, layout.pos.col = 1)
    )
  )
res <- g_km(
  df = df, variables = variables, max = 1000, newpage = FALSE, annot_surv_med = FALSE,
  ggtheme = theme_dark(),
  vp = viewport(layout.pos.row = 2, layout.pos.col = 1)
  )
```

---

**2.1 Use case: No annotations on top, annotated graph on bottom**

```r
grid.newpage()
lyt <- grid.layout(nrow = 2, ncol = 1) %>%
  viewport(layout = .) %>%
  pushViewport(
    res <- g_km(
      df = df, variables = variables, newpage = FALSE, annot_surv_med = FALSE,
      annot_at_risk = FALSE,
      vp = viewport(layout.pos.row = 1, layout.pos.col = 1)
    )
  )
res <- g_km(
  df = df, variables = variables, max = 2000, newpage = FALSE, annot_surv_med = FALSE,
  annot_at_risk = TRUE,
  ggtheme = theme_dark(),
  )
```
g_lineplot

Line plot with the optional table

Description

[Stable]

Line plot with the optional table.

Usage

```r
g_lineplot(
  df,
  alt_counts_df = NULL,
  variables = control_lineplot_vars(),
  ```
Arguments

**df**  
(data.frame)  
data set containing all analysis variables.

**alt_counts_df**  
(data.frame or NULL)  
data set that will be used (only) to counts objects in strata.

**variables**  
(named character vector) of variable names in df data set. Details are:

- **x** (character)  
  name of x-axis variable.

- **y** (character)  
  name of y-axis variable.

- **strata** (character)  
  name of grouping variable, i.e. treatment arm. Can be NA to indicate lack of groups.

- **paramcd** (character)  
  name of the variable for parameter’s code. Used for y-axis label and plot’s subtitle. Can be NA if paramcd is not to be added to the y-axis label or subtitle.

- **y_unit** (character)  
  name of variable with units of y. Used for y-axis label and plot’s subtitle. Can be NA if y unit is not to be added to the y-axis label or subtitle.
mid

(names of the statistics that will be plotted as midpoints. All the statistics indicated in mid variable must be present in the object returned by sfun, and be of a double or numeric type vector of length one.

interval

(names of the statistics that will be plotted as intervals. All the statistics indicated in interval variable must be present in the object returned by sfun, and be of a double or numeric type vector of length two.

whiskers

(names of the interval whiskers that will be plotted. Must match the names attribute of the interval element in the list returned by sfun. It is possible to specify one whisker only, lower or upper.

table

(names of the statistics that will be displayed in the table below the plot. All the statistics indicated in table variable must be present in the object returned by sfun.

sfun

(the function to compute the values of required statistics. It must return a named list with atomic vectors. The names of the list elements refer to the names of the statistics and are used by mid, interval, table. It must be able to accept as input a vector with data for which statistics are computed.

... optional arguments to sfun.

mid_type

(character)
controls the type of the mid plot, it can be point (p), line (l), or point and line (pl).

mid_point_size

(integer or double)
controls the font size of the point for mid plot.

position

(character or call)
geom element position adjustment, either as a string, or the result of a call to a position adjustment function.

legend_title

(character string)
legend title.

legend_position

(character)
the position of the plot legend (none, left, right, bottom, top, or two-element numeric vector).

ggtheme

(theme)
a graphical theme as provided by ggplot2 to control styling of the plot.

y_lab

(character)
y-axis label. If equal to NULL, then no label will be added.

y_lab_add_paramcd

(logical)
should paramcd, i.e. unique(df[[variables["paramcd"]]]) be added to the y-axis label y_lab?
y_lab_add_unit (logical)
should y unit, i.e. unique(df[[variables["y_unit"]]]) be added to the y-axis label y_lab?

title (character)
plot title.

subtitle (character)
plot subtitle.

subtitle_add_paramcd (logical)
should paramcd, i.e. unique(df[[variables["paramcd"]]]) be added to the plot's subtitle subtitle?

subtitle_add_unit (logical)
should y unit, i.e. unique(df[[variables["y_unit"]]]) be added to the plot's subtitle subtitle?

caption (character)
optional caption below the plot.

table_format (named character or NULL)
format patterns for descriptive statistics used in the (optional) table appended to the plot. It is passed directly to the h_format_row function through the format parameter. Names of table_format must match the names of statistics returned by sfun function.

table_labels (named character or NULL)
labels for descriptive statistics used in the (optional) table appended to the plot. Names of table_labels must match the names of statistics returned by sfun function.

table_font_size (integer or double)
controls the font size of values in the table.

ewpage (logical)
should plot be drawn on new page?

col (character)
colors.

Value
A ggplot line plot (and statistics table if applicable).

Examples
library(nestcolor)

adsl <- tern_ex_adsl
adlb <- tern_ex_adlb %>% dplyr::filter(ANL01FL == "Y", PARAMCD == "ALT", AVISIT != "SCREENING")
adlb$AVISIT <- droplevels(adlb$AVISIT)
adlb <- dplyr::mutate(adlb, AVISIT = forcats::fct_reorder(AVISIT, AVISITN, min))
# Mean with CI

g_lineplot(adlb, adsl, subtitle = "Laboratory Test:"
)

# Mean with CI, no stratification

g_lineplot(adlb, variables = control_lineplot_vars(strata = NA))

# Mean, upper whisker of CI, no strata counts N

g_lineplot(
  adlb,
  whiskers = "mean_ci_upr",
  title = "Plot of Mean and Upper 95% Confidence Limit by Visit"
)

# Median with CI

g_lineplot(
  adlb,
  adsl,
  mid = "median",
  interval = "median_ci",
  whiskers = c("median_ci_lwr", "median_ci_upr"),
  title = "Plot of Median and 95% Confidence Limits by Visit"
)

# Mean, +/- SD

g_lineplot(adlb, adsl,
  interval = "mean_sdi",
  whiskers = c("mean_sdi_lwr", "mean_sdi_upr"),
  title = "Plot of Median +/- SD by Visit"
)

# Mean with CI plot with stats table

g_lineplot(adlb, adsl, table = c("n", "mean", "mean_ci"))

# Mean with CI, table and customized confidence level

g_lineplot(
  adlb,
  adsl,
  table = c("n", "mean", "mean_ci"),
  control = control_summarize_vars(conf_level = 0.80),
  title = "Plot of Mean and 80% Confidence Limits by Visit"
)

# Mean with CI, table, filtered data
adlb_f <- dplyr::filter(adlb, ARMCD != "ARM A" | AVISIT == "BASELINE")
g_lineplot(adlb_f, table = c("n", "mean"))
Description

[Stable]

Based on the STEP results, creates a ggplot graph showing the estimated HR or OR along the continuous biomarker value subgroups.

Usage

```r
g_step(
  df,
  use_percentile = "Percentile Center" %in% names(df),
  est = list(col = "blue", lty = 1),
  ci_ribbon = list(fill = getOption("ggplot2.discrete.colour")[1], alpha = 0.5),
  col = getOption("ggplot2.discrete.colour")
)
```

Arguments

- **df** (tibble): result of `tidy.step()`.
- **use_percentile** (flag): whether to use percentiles for the x axis or actual biomarker values.
- **est** (named list): col and lty settings for estimate line.
- **ci_ribbon** (named list or NULL): fill and alpha settings for the confidence interval ribbon area, or NULL to not plot a CI ribbon.
- **col** (character): colors.

Value

A ggplot STEP graph.

See Also

Custom tidy method `tidy.step()`.

Examples

```r
library(nestcolor)
library(survival)
lung$sex <- factor(lung$sex)
# Survival example.
vars <- list(
  time = "time",
  event = "status",
  arm = "sex",
```
biomarker = "age"

step_matrix <- fit_survival_step(
    variables = vars,
    data = lung,
    control = c(control_coxph(), control_step(num_points = 10, degree = 2))
)
step_data <- broom::tidy(step_matrix)

# Default plot.
g_step(step_data)

# Add the reference 1 horizontal line.
library(ggplot2)
g_step(step_data) +
  ggplot2::geom_hline(ggplot2::aes(yintercept = 1), linetype = 2)

# Use actual values instead of percentiles, different color for estimate and no CI,
# use log scale for y axis.
g_step(
    step_data,
    use_percentile = FALSE,
    est = list(col = "blue", lty = 1),
    ci_ribbon = NULL
  ) + scale_y_log10()

# Adding another curve based on additional column.
step_data$extra <- exp(step_data$"Percentile Center")
g_step(step_data) +
  ggplot2::geom_line(ggplot2::aes(y = extra), linetype = 2, color = "green")

# Response example.
vars <- list(
    response = "status",
    arm = "sex",
    biomarker = "age"
)

step_matrix <- fit_rsp_step(
    variables = vars,
    data = lung,
    control = c(
        control_logistic(response_definition = "I(response == 2)"),
        control_step()
    )
)
step_data <- broom::tidy(step_matrix)
g_step(step_data)
g_waterfall

Horizontal Waterfall Plot

Description

[Stable]

Usage

g_waterfall(
  height,
  id,
  col_var = NULL,
  col = getOption("ggplot2.discrete.colour"),
  xlab = NULL,
  ylab = NULL,
  col_legend_title = NULL,
  title = NULL
)

Arguments

height ('numeric")
  vector containing values to be plotted as the waterfall bars.
id (character)
  vector containing IDs to use as the x-axis label for the waterfall bars.
col_var (factor, character or NULL)
  categorical variable for bar coloring. NULL by default.
col (character)
  colors.
xlab (character)
  x label. Default is "ID".
ylab (character)
  y label. Default is "Value".
col_legend_title (character)
  text to be displayed as legend title.
title (character)
  text to be displayed as plot title.

Details

This basic waterfall plot visualizes a quantity height ordered by value with some markup.
Value

A ggplot waterfall plot.

Examples

```r
library(dplyr)
library(nestcolor)

g_waterfall(height = c(3, 5, -1), id = letters[1:3])

g_waterfall(
  height = c(3, 5, -1),
  id = letters[1:3],
  col_var = letters[1:3]
)

adsl_f <- tern_ex_adsl %>%
  select(USUBJID, STUDYID, ARM, ARMCD, SEX)

adrs_f <- tern_ex_adrs %>%
  filter(PARAMCD == "OVRINV") %>%
  mutate(pchg = rnorm(n(), 10, 50))

adrs_f <- head(adrs_f, 30)
adrs_f <- adrs_f[!duplicated(adrs_f$USUBJID),]
head(adrs_f)

  g_waterfall(
    height = adrs_f$pchg,
    id = adrs_f$USUBJID,
    col_var = adrs_f$AVALC
  )

  g_waterfall(
    height = adrs_f$pchg,
    id = paste("asdfsdfsdfsfd", adrs_f$USUBJID),
    col_var = adrs_f$SEX
  )

  g_waterfall(
    height = adrs_f$pchg,
    id = paste("asdfsdfsdfsfd", adrs_f$USUBJID),
    xlab = "ID",
    ylab = "Percentage Change",
    title = "Waterfall plot"
  )
```

---

**h_adlb_worsen**

*Helper Function to Prepare ADLB with Worst Labs*
**Description**

[Stable]
Helper function to prepare a df for generate the patient count shift table

**Usage**

```r
h_adlb_worsen(
  adlb,
  worst_flag_low = NULL,
  worst_flag_high = NULL,
  direction_var
)
```

**Arguments**

- `adlb` (data.frame)
  ADLB dataframe
- `worst_flag_low` (named vector)
  Worst low post-baseline lab grade flag variable
- `worst_flag_high` (named vector)
  Worst high post-baseline lab grade flag variable
- `direction_var` (string)
  Direction variable specifying the direction of the shift table of interest. Only lab records flagged by L, H or B are included in the shift table.
  - L: low direction only
  - H: high direction only
  - B: both low and high directions

**Value**

`h_adlb_worsen()` returns the adlb data.frame containing only the worst labs specified according to `worst_flag_low` or `worst_flag_high` for the direction specified according to `direction_var`. For instance, for a lab that is needed for the low direction only, only records flagged by `worst_flag_low` are selected. For a lab that is needed for both low and high directions, the worst low records are selected for the low direction, and the worst high record are selected for the high direction.

**See Also**

`abnormal_by_worst_grade_worsen`

**Examples**

```r
library(dplyr)

# The direction variable, GRADDR, is based on metadata
adlb <- tern_ex_adlb %>%
  mutate(
```
GRADDR = case_when(
  PARAMCD == "ALT" ~ "B",
  PARAMCD == "CRP" ~ "L",
  PARAMCD == "IGA" ~ "H"
)  
) %>%
filter(SAFFL == "Y" & ONTRTFL == "Y" & GRADDR != ")

df <- h_adlb_worsen(
adlb,
  worst_flag_low = c("WGRLOFL" = "Y"),
  worst_flag_high = c("WGRHIFL" = "Y"),
  direction_var = "GRADDR"
)

h_adsl_adlb_merge_using_worst_flag

Helper Function for Deriving Analysis Datasets for LBT13 and LBT14

Description

[Stable]
Helper function that merges ADSL and ADLB datasets so that missing lab test records are inserted in the output dataset.

Usage

h_adsl_adlb_merge_using_worst_flag(
adsl,
adlb,
  worst_flag = c("WGRHIFL" = "Y"),
  by_visit = FALSE,
  no_fillin_visits = c("SCREENING", "BASELINE")
)

Arguments

adsl (data.frame) ADSL dataframe.
adlb (data.frame) ADLB dataframe.
worst_flag (named vector) Worst post-baseline lab flag variable.
by_visit (logical) defaults to FALSE to generate worst grade per patient. If worst grade per patient per visit is specified for worst_flag, then by_visit should be TRUE to generate worst grade patient per visit.
no_fillin_visits
(named character)
Visits that are not considered for post-baseline worst toxicity grade. Defaults to
c(“SCREENING”, “BASELINE”).

Details
In the result data missing records will be created for the following situations:

• Patients who are present in adsl but have no lab data in adlb (both baseline and post-baseline).
• Patients who do not have any post-baseline lab values.
• Patients without any post-baseline values flagged as the worst.

Value
df containing variables shared between adlb and adsl along with variables PARAM, PARAMCD, 
ATOXGR, and BTOXGR relevant for analysis. Optionally, AVISIT are AVISITN are included when
by_visit = TRUE and no_fillin_visits = c(“SCREENING”, “BASELINE”).

Examples
# ‘h_adsl_adlb_merge_using_worst_flag’
adlb_out <- h_adsl_adlb_merge_using_worst_flag(
  tern_ex_adsl, 
  tern_ex_adlb, 
  worst_flag = c(“WGRHIFL” = “Y”) 
)

# ‘h_adsl_adlb_merge_using_worst_flag’ by visit example
adlb_out_by_visit <- h_adsl_adlb_merge_using_worst_flag(
  tern_ex_adsl, 
  tern_ex_adlb, 
  worst_flag = c(“WGRLOVFL” = “Y”), 
  by_visit = TRUE
)

----------

h_ancova

Helper Function to Return Results of a Linear Model

Description
[Stable]

Usage
h_ancova(.var, .df_row, variables, interaction_item = NULL)
Arguments

.var (string)
single variable name that is passed by rtables when requested by a statistics function.

.df_row (data.frame)
data set that includes all the variables that are called in .var and variables.

variables (named list of strings)
list of additional analysis variables, with expected elements:
  • arm (string)
group variable, for which the covariate adjusted means of multiple groups will be summarized. Specifically, the first level of arm variable is taken as the reference group.
  • covariates (character)
a vector that can contain single variable names (such as "X1"), and/or interaction terms indicated by "X1 * X2".

interaction_item (character)
name of the variable that should have interactions with arm. if the interaction is not needed, the default option is NULL.

Value

The summary of a linear model.

Examples

h_ancova(
  .var = "Sepal.Length",
  .df_row = iris,
  variables = list(arm = "Species", covariates = c("Petal.Length * Petal.Width", "Sepal.Width"))
)

h_append_grade_groups (Helper function for s_count_occurrences_by_grade())

Description

[Stable]
Helper function for s_count_occurrences_by_grade() to insert grade groupings into list with individual grade frequencies. The order of the final result follows the order of grade_groups. The elements under any-grade group (if any), i.e. the grade group equal to refs will be moved to the end. Grade groups names must be unique.

Usage

h_append_grade_groups(grade_groups, refs, remove_single = TRUE)
Arguments

grade_groups  (named list of character)
containing groupings of grades.

refs  (named list of numeric)
where each name corresponds to a reference grade level and each entry represents a count.

remove_single  (logical)
TRUE to not include the elements of one-element grade groups in the output list; in this case only the grade groups names will be included in the output.

Value

Formatted list of grade groupings.

Examples

\[
\text{h_append_grade_groups(}
\text{list(}
\quad "Any Grade" = as.character(1:5),
\quad "Grade 1-2" = c("1", "2"),
\quad "Grade 3-4" = c("3", "4")
\text{),}
\text{list("1" = 10, "2" = 20, "3" = 30, "4" = 40, "5" = 50)}
\text{)}
\text{)}
\]

\[
\text{h_append_grade_groups(}
\text{list(}
\quad "Any Grade" = as.character(5:1),
\quad "Grade A" = "5",
\quad "Grade B" = c("4", "3")
\text{),}
\text{list("1" = 10, "2" = 20, "3" = 30, "4" = 40, "5" = 50)}
\text{)}
\text{)}
\]

\[
\text{h_append_grade_groups(}
\text{list(}
\quad "Any Grade" = as.character(1:5),
\quad "Grade 1-2" = c("1", "2"),
\quad "Grade 3-4" = c("3", "4")
\text{),}
\text{list("1" = 10, "2" = 5, "3" = 0)}
\text{)}
\text{)}
\]

\[
\text{h_col_indices}
\]

Obtain Column Indices
**Description**

[Stable]

Helper function to extract column indices from a VTableTree for a given vector of column names.

**Usage**

\[ \text{h\_col\_indices}(\text{table\_tree}, \text{col\_names}) \]

**Arguments**

- **table\_tree** (VTableTree)
  table to extract the indices from.
- **col\_names** (character)
  vector of column names.

**Value**

A vector of column indices.

---

**h\_count\_cumulative**  
*Helper Function for s\_count\_cumulative()*

**Description**

[Stable]

Helper function to calculate count and fraction of \(x\) values in the lower or upper tail given a threshold.

**Usage**

\[ \text{h\_count\_cumulative}( \text{x, threshold}, \text{lower\_tail = TRUE, include\_eq = TRUE, na.rm = TRUE, .N\_col}) \]

**Arguments**

- **x** (numeric)
  vector of numbers we want to analyze.
- **threshold** (number)
  a cutoff value as threshold to count values of \(x\).
lower_tail (logical) whether to count lower tail, default is TRUE.
include_eq (logical) whether to include value equal to the threshold in count, default is TRUE.
na.rm (flag) whether NA values should be removed from x prior to analysis.
.N_col (count) denominator for fraction calculation.

Value
A named vector with items:
- count: the count of values less than, less or equal to, greater than, or greater or equal to a threshold of user specification.
- fraction: the fraction of the count.

See Also
count_cumulative

Examples
set.seed(1, kind = "Mersenne-Twister")
x <- c(sample(1:10, 10), NA)
.N_col <- length(x)
h_count_cumulative(x, 5, .N_col = .N_col)
h_count_cumulative(x, 5, lower_tail = FALSE, include_eq = FALSE, na.rm = FALSE, .N_col = .N_col)
h_count_cumulative(x, 0, lower_tail = FALSE, .N_col = .N_col)
h_count_cumulative(x, 100, lower_tail = FALSE, .N_col = .N_col)
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variables</td>
<td>(named list of string) list of additional analysis variables.</td>
</tr>
<tr>
<td>interaction</td>
<td>(flag) if TRUE, the model includes the interaction between the studied treatment and candidate covariate. Note that for univariate models without treatment arm, and multivariate models, no interaction can be used so that this needs to be FALSE.</td>
</tr>
<tr>
<td>effect</td>
<td>(string) the treatment variable.</td>
</tr>
<tr>
<td>covar</td>
<td>(string) the name of the covariate in the model.</td>
</tr>
<tr>
<td>data</td>
<td>(data.frame) the dataset containing the variables to summarize.</td>
</tr>
<tr>
<td>mod</td>
<td>(coxph) Cox regression model fitted by <code>survival::coxph()</code>.</td>
</tr>
<tr>
<td>control</td>
<td>(list) a list of controls as returned by <code>control_coxreg()</code>.</td>
</tr>
<tr>
<td>var</td>
<td>(string) single variable name that is passed by <code>rtables</code> when requested by a statistics function.</td>
</tr>
</tbody>
</table>

Value

- `h_coxreg_univar_formulas()` returns a character vector coercible into formulas (e.g. `stats::as.formula()`).
- `h_coxreg_multivar_formulas()` returns a string coercible into a formula (e.g. `stats::as.formula()`).
- `h_coxreg_univar_extract()` returns a `data.frame` with variables `effect`, `term`, `term_label`, `level`, `n`, `hr`, `lcl`, `ucl`, and `pval`.
- `h_coxreg_multivar_extract()` returns a `data.frame` with variables `pval`, `hr`, `lcl`, `ucl`, `level`, `n`, `term`, and `term_label`.

Functions

- `h_coxreg_univar_formulas()`: Helper for Cox regression formula. Creates a list of formulas. It is used internally by `fit_coxreg_univar()` for the comparison of univariate Cox regression models.
- `h_coxreg_multivar_formulas()`: Helper for multivariate Cox regression formula. Creates a formulas string. It is used internally by `fit_coxreg_multivar()` for the comparison of multivariate Cox regression models. Interactions will not be included in multivariate Cox regression model.
- `h_coxreg_univar_extract()`: Utility function to help tabulate the result of a univariate Cox regression model.
- `h_coxreg_multivar_extract()`: Tabulation of multivariate Cox regressions. Utility function to help tabulate the result of a multivariate Cox regression model for a treatment/covariate variable.
See Also

cox_regression

Examples

# `h_coxreg_univar_formulas`

## Simple formulas.

```r
h_coxreg_univar_formulas(
  variables = list(
    time = "time", event = "status", arm = "armcd", covariates = c("X", "y")
  )
)
```

## Addition of an optional strata.

```r
h_coxreg_univar_formulas(
  variables = list(
    time = "time", event = "status", arm = "armcd", covariates = c("X", "y"),
    strata = "SITE"
  ),
  interaction = TRUE
)
```

## Inclusion of the interaction term.

```r
h_coxreg_univar_formulas(
  variables = list(
    time = "time", event = "status", arm = "armcd", covariates = c("X", "y"),
    strata = "SITE"
  ),
  interaction = TRUE
)
```

## Only covariates fitted in separate models.

```r
h_coxreg_univar_formulas(
  variables = list(
    time = "time", event = "status", covariates = c("X", "y")
  )
)
```

# `h_coxreg_multivar_formula`

```r
h_coxreg_multivar_formula(
  variables = list(
    time = "AVAL", event = "event", arm = "ARMCD", covariates = c("RACE", "AGE")
  )
)
```

# Addition of an optional strata.

```r
h_coxreg_multivar_formula(
  variables = list(
    time = "AVAL", event = "event", arm = "ARMCD", covariates = c("RACE", "AGE"),
    strata = "SITE"
  )
)
# Example without treatment arm.

```r
h_coxreg_multivar_formula(
    variables = list(
        time = "AVAL", event = "event", covariates = c("RACE", "AGE"),
        strata = "SITE"
    )
)
```

```r
library(survival)
dta_simple <- data.frame(  
    time = c(5, 5, 10, 10, 5, 5, 10, 10),
    status = c(0, 0, 1, 0, 0, 1, 1, 1),
    armcd = factor(LETTERS[c(1, 1, 1, 1, 2, 2, 2, 2)], levels = c("A", "B")),
    var1 = c(45, 55, 65, 75, 55, 65, 85, 75),
    var2 = c("F", "M", "F", "M", "F", "M", "F", "U")
)
```

```r
mod <- coxph(Surv(time, status) ~ armcd + var1, data = dta_simple)
result <- h_coxreg_univar_extract(  
    effect = "armcd", covar = "armcd", mod = mod, data = dta_simple
)
```

```r
mod <- coxph(Surv(time, status) ~ armcd + var1, data = dta_simple)
result <- h_coxreg_multivar_extract(  
    var = "var1", mod = mod, data = dta_simple
)
```

---

**h_data_plot**

*Helper function: tidy survival fit*

**Description**

*Stable*

Convert the survival fit data into a data frame designed for plotting within `g_km`.

This starts from the `broom::tidy()` result, and then:

- Post-processes the `strata` column into a factor.
- Extends each stratum by an additional first row with time 0 and probability 1 so that downstream plot lines start at those coordinates.
- Adds a `censor` column.
- Filters the rows before `max_time`.
h_decompose_gg

Usage

h_data_plot(fit_km, armval = "All", max_time = NULL)

Arguments

- **fit_km** (survfit) result of `survival::survfit()`.
- **armval** (string) used as strata name when treatment arm variable only has one level. Default is "All".
- **max_time** (numeric) maximum value to show on X axis. Only data values less than or up to this threshold value will be plotted (defaults to NULL).

Value

A tibble with columns `time`, `n.risk`, `n.event`, `n.censor`, `estimate`, `std.error`, `conf.high`, `conf.low`, `strata`, and `censor`.

Examples

```r
library(dplyr)
library(survival)

# Test with multiple arms
tern_ex_adtte %>%
  filter(PARAMCD == "OS") %>%
  survfit(form = Surv(AVAL, 1 - CNSR) ~ ARMCD, data = .) %>%
  h_data_plot()

# Test with single arm
tern_ex_adtte %>%
  filter(PARAMCD == "OS", ARMCD == "ARM B") %>%
  survfit(form = Surv(AVAL, 1 - CNSR) ~ ARMCD, data = .) %>%
  h_data_plot(armval = "ARM B")
```

---

**Description**

[Stable]

The elements composing the ggplot are extracted and organized in a list.
Usage

h_decompose_gg(gg)

Arguments

gg (ggplot)

a graphic to decompose.

Value

A named list with elements:

- panel: The panel.
- yaxis: The y-axis.
- xaxis: The x-axis.
- xlab: The x-axis label.
- ylab: The y-axis label.
- guide: The legend.

Examples

library(dplyr)
library(survival)
library(grid)

fit_km <- tern_ex_adtte %>%
  filter(PARAMCD == "OS") %>%
  survfit(form = Surv(AVAL, 1 - CNSR) ~ ARMCD, data = .)
data_plot <- h_data_plot(fit_km = fit_km)
xticks <- h_xticks(data = data_plot)
gg <- h_ggkm(
data = data_plot,
yval = "Survival",
censor_show = TRUE,
xticks = xticks, xlab = "Days", ylab = "Survival Probability",
title = "tt",
footnotes = "ff"
)
g_el <- h_decompose_gg(gg)
grid::grid.newpage()
grid.rect(gp = grid::gpar(lty = 1, col = "red", fill = "gray85", lwd = 5))
grid::grid.draw(g_el$panel)

grid::grid.newpage()
grid.rect(gp = grid::gpar(lty = 1, col = "royalblue", fill = "gray85", lwd = 5))
grid::grid.draw(with(g_el, cbind(ylab, yaxis)))
**h_format_row**

*Helper function to get the right formatting in the optional table in g_lineplot.*

**Description**

[Stable]

**Usage**

```r
h_format_row(x, format, labels = NULL)
```

**Arguments**

- `x` *(named list)*  
  list of numerical values to be formatted and optionally labeled. Elements of `x` must be numeric vectors.

- `format` *(named character or NULL)*  
  format patterns for `x`. Names of the `format` must match the names of `x`. This parameter is passed directly to the `rtables::format_rcell` function through the `format` parameter.

- `labels` *(named character or NULL)*  
  optional labels for `x`. Names of the `labels` must match the names of `x`. When a label is not specified for an element of `x`, then this function tries to use label or names (in this order) attribute of that element (depending on which one exists and it is not NULL or NA or NaN). If none of these attributes are attached to a given element of `x`, then the label is automatically generated.

**Value**

A single row `data.frame` object.

**Examples**

```r
mean_ci <- c(48, 51)
x <- list(mean = 50, mean_CI = mean_ci)
format <- c(mean = "xx.x", mean_CI = "(xx.xx, xx.xx)")
labels <- c(mean = "My Mean")
h_format_row(x, format, labels)

attr(mean_ci, "label") <- "Mean 95% CI"
x <- list(mean = 50, mean_CI = mean_ci)
h_format_row(x, format, labels)
```
h_ggkm

Helper function: KM plot

Description

[Stable]
Draw the Kaplan-Meier plot using ggplot2.

Usage

h_ggkm(
  data,
  xticks = NULL,
  yval = "Survival",
  censor_show,
  xlab,
  ylab,
  title,
  footnotes = NULL,
  max_time = NULL,
  lwd = 1,
  lty = NULL,
  pch = 3,
  size = 2,
  col = NULL,
  ci_ribbon = FALSE,
  ggtheme = nestcolor::theme_nest()
)

Arguments

data (data.frame) survival data as pre-processed by h_data_plot.
xticks (numeric, number, or NULL) numeric vector of ticks or single number with spacing between ticks on the x axis. If NULL (default), labeling::extended() is used to determine an optimal tick position on the x axis.
yval (string) value of y-axis. Options are Survival (default) and Failure probability.
censor_show (flag) whether to show censored.
xlab (string) label of x-axis.
ylab (string) label of y-axis.
title (string)
title for plot.

footnotes (string)
footnotes for plot.

max_time (numeric)
maximum value to show on X axis. Only data values less than or up to this
threshold value will be plotted (defaults to NULL).

lwd (numeric)
line width. Length of a vector should be equal to number of strata from survival::survfit().

lty (numeric)
line type. Length of a vector should be equal to number of strata from survival::survfit().

pch (numeric, string)
value or character of points symbol to indicate censored cases.

size (numeric)
size of censored point, a class of unit.

col (character)
lines colors. Length of a vector should be equal to number of strata from survival::survfit().

ci_ribbon (flag)
draw the confidence interval around the Kaplan-Meier curve.

ggtheme (theme)
a graphical theme as provided by ggplot2 to control outlook of the Kaplan-
Meier curve.

Value
A ggplot object.

Examples

library(dplyr)
library(survival)

fit_km <- tern_ex_adtte %>%
  filter(PARAMCD == "OS") %>%
  survfit(form = Surv(AVAL, 1 - CNSR) ~ ARMCD, data = .)
data_plot <- h_data_plot(fit_km = fit_km)
xticks <- h_xticks(data = data_plot)
gg <- h_ggkm(
  data = data_plot,
  censor_show = TRUE,
  xticks = xticks,
  xlab = "Days",
  ylab = "Survival",
  title = "Survival"
)
**h_grob_coxph**

**Helper Function:** CoxPH Grob

**Description**

[Stable]

Grob of rtable output from `h_tbl_coxph_pairwise()`

**Usage**

```r
h_grob_coxph(
  ..., 
  x = 0, 
  y = 0, 
  width = grid::unit(0.4, "npc"), 
  ttheme = gridExtra::ttheme_default(padding = grid::unit(c(1, 0.5), "lines"), core = 
    list(bg_params = list(fill = c("grey95", "grey90"), alpha = 0.5)))
)
```

**Arguments**

- `...` arguments will be passed to `h_tbl_coxph_pairwise()`.
- `x` (numeric) a value between 0 and 1 specifying x-location.
- `y` (numeric) a value between 0 and 1 specifying y-location.
- `width` (unit) width (as a unit) to use when printing the grob.
- `ttheme` (list) see `gridExtra::ttheme_default()`.

**Value**

A grob of a table containing statistics HR, XX% CI (XX taken from control_coxph_pw), and p-value (log-rank).
Examples

```r
library(dplyr)
library(survival)
library(grid)

grid::grid.newpage()
grid.rect(gp = grid::gpar(lty = 1, col = "pink", fill = "gray85", lwd = 1))
data <- tern_ex_adtte %>%
  filter(PARAMCD == "OS") %>%
  mutate(is_event = CNSR == 0)
tbl_grob <- h_grob_coxph(
  df = data,
  variables = list(tte = "AVAL", is_event = "is_event", arm = "ARMCD"),
  control_coxph_pw = control_coxph(conf_level = 0.9), x = 0.5, y = 0.5
)
grid::grid.draw(tbl_grob)
```

---

**h_grob_median_surv**  
*Helper Function: Survival Estimation Grob*

**Description**

*Stable*

The survival fit is transformed in a grob containing a table with groups in rows characterized by N, median and 95% confidence interval.

**Usage**

```r
h_grob_median_surv(
  fit_km, 
  armval = "All", 
  x = 0.9, 
  y = 0.9, 
  width = grid::unit(0.3, "npc"),
  ttheme = gridExtra::ttheme_default()
)
```

**Arguments**

- `fit_km` *(survfit)*  
  Result of `survival::survfit()`.

- `armval` *(string)*  
  Used as strata name when treatment arm variable only has one level. Default is "All".
h_grob_tbl_at_risk

Description

[Stable]

Two graphical objects are obtained, one corresponding to row labeling and the second to the number of patient at risk.

Usage

h_grob_tbl_at_risk(data, annot_tbl, xlim)
Arguments

data (data.frame)
  survival data as pre-processed by h_data_plot.
annot_tbl (data.frame)
  annotation as prepared by survival::summary.survfit() which includes the number of patients at risk at given time points.
xlim (numeric)
  the maximum value on the x-axis (used to ensure the at risk table aligns with the KM graph).

Value

A named list of two gTree objects: at_risk and label.

Examples

library(dplyr)
library(survival)
library(grid)

fit_km <- tern_ex_adtte %>%
  filter(PARAMCD == "OS") %>%
  survfit(form = Surv(AVAL, 1 - CNSR) ~ ARMCD, data = .)

data_plot <- h_data_plot(fit_km = fit_km)

xticks <- h_xticks(data = data_plot)

gg <- h_ggkm(
  data = data_plot,
  censor_show = TRUE,
  xticks = xticks, xlab = "Days", ylab = "Survival Probability",
  title = "tt", footnotes = "ff", yval = "Survival"
)

# The annotation table reports the patient at risk for a given strata and # time (xticks).
annot_tbl <- summary(fit_km, time = xticks)
if (is.null(fit_km$strata)) {
  annot_tbl <- with(annot_tbl, data.frame(n.risk = n.risk, time = time, strata = "All"))
} else {
  strata_lst <- strsplit(sub("=" , "equals", levels(annot_tbl$strata)), "equals")
layers(annot_tbl$strata) <- matrix(unlist(strata_lst), ncol = 2, byrow = TRUE)[, 2]
  annot_tbl <- data.frame(
    n.risk = annot_tbl$n.risk,
    time = annot_tbl$time,
    strata = annot_tbl$strata
  )
}
# The annotation table is transformed into a grob.
tbl <- h_grob_tbl_at_risk(data = data_plot, annot_tbl = annot_tbl, xlim = max(xticks))

# For the representation, the layout is estimated for which the decomposition
# of the graphic element is necessary.
g_el <- h_decompose_gg(gg)
lyt <- h_km_layout(data = data_plot, g_el = g_el, title = "t", footnotes = "f")

grid::grid.newpage()
pushViewport(viewport(layout = lyt, height = .95, width = .95))
grid.rect(gp = grid::gpar(lty = 1, col = "purple", fill = "gray85", lwd = 1))
pushViewport(viewport(layout.pos.row = 4, layout.pos.col = 2))
grid.rect(gp = grid::gpar(lty = 1, col = "orange", fill = "gray85", lwd = 1))
ggrid::grid.draw(tbl$at_risk)
popViewport()
pushViewport(viewport(layout.pos.row = 4, layout.pos.col = 1))
grid.rect(gp = grid::gpar(lty = 1, col = "green3", fill = "gray85", lwd = 1))
ggrid::grid.draw(tbl$label)

h_grob_y_annot  
**Helper: Grid Object with y-axis Annotation**

**Description**

[Stable]

Build the y-axis annotation from a decomposed ggplot.

**Usage**

h_grob_y_annot(ylab, yaxis)

**Arguments**

- **ylab**  *(gtable)*
  
  the y-lab as a graphical object derived from a ggplot.

- **yaxis**  *(gtable)*
  
  the y-axis as a graphical object derived from a ggplot.

**Value**

a gTree object containing the y-axis annotation from a ggplot.
**Examples**

```r
library(dplyr)
library(survival)
library(grid)

fit_km <- tern_ex_adtte %>%
  filter(PARAMCD == "OS") %>%
  survfit(form = Surv(AVAL, 1 - CNSR) ~ ARMCD, data = .)
data_plot <- h_data_plot(fit_km = fit_km)
xticks <- h_xticks(data = data_plot)
gg <- h_ggkm(
  data = data_plot,
  censor_show = TRUE,
  xticks = xticks,
  xlab = "Days", ylab = "Survival Probability",
  title = "title", footnotes = "footnotes", yval = "Survival"
)
g_el <- h_decompose_gg(gg)

grid::grid.newpage()
pvp <- grid::plotViewport(margins = c(5, 4, 2, 20))
pushViewport(pvp)
grid::grid.draw(h_grob_y_annot(ylab = g_el$ylab, yaxis = g_el$yaxis))
grid.rect(gp = grid::gpar(lty = 1, col = "gray35", fill = NA))
```

---

**h_g_ipp**

*Helper Function To Create Simple Line Plot over Time*

**Description**

[Stable]

Function that generates a simple line plot displaying parameter trends over time.

**Usage**

```r
h_g_ipp(
  df,
  xvar,
  yvar,
  xlab,
  ylab,
  id_var,
  title = "Individual Patient Plots",
  subtitle = "",
  caption = NULL,
)```
add_baseline_hline = FALSE,
yvar_baseline = "BASE",
ggtheme = nestcolor::theme_nest(),
col = NULL
)

Arguments

df (data.frame) data set containing all analysis variables.
xvar (string) time point variable to be plotted on x-axis.
yvar (string) continuous analysis variable to be plotted on y-axis.
xlab (string) plot label for x-axis.
ylab (string) plot label for y-axis.
id_var (string) variable used as patient identifier.
title (string) title for plot.
subtitle (string) subtitle for plot.
caption (character scalar) optional caption below the plot.
add_baseline_hline (flag) adds horizontal line at baseline y-value on plot when TRUE.
yvar_baseline (string) variable with baseline values only. Ignored when add_baseline_hline is FALSE.
ggtheme (theme) optional graphical theme function as provided by ggplot2 to control outlook of plot. Use ggplot2::theme() to tweak the display.
col (character) lines colors.

Value

A ggplot line plot.

See Also

g_ipp() which uses this function.
Examples

```r
library(dplyr)
library(nestcolor)

# Select a small sample of data to plot.
adlb <- tern_ex_adlb %>%
    filter(PARAMCD == "ALT", !(AVISIT %in% c("SCREENING", "BASELINE"))) %>%
    slice(1:36)

p <- h_g_ipp(
    df = adlb,
    xvar = "AVISIT",
    yvar = "AVAL",
    xlab = "Visit",
    id_var = "USUBJID",
    ylab = "SGOT/ALT (U/L)",
    add_baseline_hline = TRUE
)
```

---

**h_km_layout**

*Helper: KM Layout*

**Description**

[Stable]

Prepares a (5 rows) x (2 cols) layout for the Kaplan-Meier curve.

**Usage**

```r
h_km_layout(data, g_el, title, footnotes, annot_at_risk = TRUE)
```

**Arguments**

- **data** *(data.frame)*
  survival data as pre-processed by `h_data_plot`.
- **g_el** *(list of gtable)*
  list as obtained by `h_decompose_gg()`.
- **title** *(string)*
  title for plot.
- **footnotes** *(string)*
  footnotes for plot.
- **annot_at_risk** *(flag)*
  compute and add the annotation table reporting the number of patient at risk matching the main grid of the Kaplan-Meier curve.
Details

The layout corresponds to a grid of two columns and five rows of unequal dimensions. Most of the
dimension are fixed, only the curve is flexible and will accommodate with the remaining free space.

- The left column gets the annotation of the ggplot (y-axis) and the names of the strata for the
  patient at risk tabulation. The main constraint is about the width of the columns which must
  allow the writing of the strata name.
- The right column receive the ggplot, the legend, the x-axis and the patient at risk table.

Value

A grid layout.

Examples

```r
library(dplyr)
library(survival)
library(grid)

fit_km <- tern_ex_adtte %>%
  filter(PARAMCD == "OS") %>%
  survfit(form = Surv(AVAL, 1 - CNSR) ~ ARMCD, data = .)
data_plot <- h_data_plot(fit_km = fit_km)
xticks <- h_xticks(data = data_plot)

gg <- h_ggkm(
data = data_plot,
censor_show = TRUE,
xticks = xticks, xlab = "Days", ylab = "Survival Probability",
title = "tt", footnotes = "ff", yval = "Survival"
)
g_el <- h_decompose_gg(gg)
lyt <- h_km_layout(data = data_plot, g_el = g_el, title = "t", footnotes = "f")
grid.show.layout(lyt)
```

h_logistic_regression  Helper Functions for Multivariate Logistic Regression

Description

[Stable]

Helper functions used in calculations for logistic regression.
Usage

h_get_interaction_vars(fit_glm)

h_interaction_coef_name(
    interaction_vars,
    first_var_with_level,
    second_var_with_level
)

h_or_cat_interaction(
    odds_ratio_var,
    interaction_var,
    fit_glm,
    conf_level = 0.95
)

h_or_cont_interaction(
    odds_ratio_var,
    interaction_var,
    fit_glm,
    at = NULL,
    conf_level = 0.95
)

h_or_interaction(
    odds_ratio_var,
    interaction_var,
    fit_glm,
    at = NULL,
    conf_level = 0.95
)

h_simple_term_labels(terms, table)

h_interaction_term_labels(terms1, terms2, table, any = FALSE)

h_glm_simple_term_extract(x, fit_glm)

h_glm_interaction_extract(x, fit_glm)

h_glm_inter_term_extract(odds_ratio_var, interaction_var, fit_glm, ...)

h_logistic_simple_terms(x, fit_glm, conf_level = 0.95)

h_logistic_inter_terms(x, fit_glm, conf_level = 0.95, at = NULL)
Arguments

fit_glm (glm)
logistic regression model fitted by \texttt{stats::glm()} with "binomial" family. Limited functionality is also available for conditional logistic regression models fitted by \texttt{survival::clogit()}, currently this is used only by \texttt{extract rsp biomarkers()}.

interaction_vars
(character of length 2)
interaction variable names.

first_var_with_level
(character of length 2)
the first variable name with the interaction level.

second_var_with_level
(character of length 2)
the second variable name with the interaction level.

odds_ratio_var (string)
the odds ratio variable.

interaction_var
(string)
the interaction variable.

conf_level (proportion)
confidence level of the interval.

at (NULL or numeric)
onoptional values for the interaction variable. Otherwise the median is used.

terms (character)
simple terms.

table (table)
table containing numbers for terms.

terms1 (character)
terms for first dimension (rows).

terms2 (character)
terms for second dimension (rows).

any (flag)
whether any of term1 and term2 can be fulfilled to count the number of patients. In that case they can only be scalar (strings).

x (string or character)
a variable or interaction term in fit_glm (depending on the helper function).

... additional arguments for the lower level functions.

Value

Vector of names of interaction variables.
Name of coefficient.
Odds ratio.
Odds ratio.
Odds ratio.
Term labels containing numbers of patients.
Term labels containing numbers of patients.
Tabulated main effect results from a logistic regression model.
Tabulated interaction term results from a logistic regression model.
A `data.frame` of tabulated interaction term results from a logistic regression model.
Tabulated statistics for the given variable(s) from the logistic regression model.
Tabulated statistics for the given variable(s) from the logistic regression model.

Functions

- `h_get_interaction_vars()`: Helper function to extract interaction variable names from a fitted model assuming only one interaction term.
- `h_interaction_coef_name()`: Helper function to get the right coefficient name from the interaction variable names and the given levels. The main value here is that the order of first and second variable is checked in the `interaction-vars` input.
- `h_or_cat_interaction()`: Helper function to calculate the odds ratio estimates for the case when both the odds ratio and the interaction variable are categorical.
- `h_or_cont_interaction()`: Helper function to calculate the odds ratio estimates for the case when either the odds ratio or the interaction variable is continuous.
- `h_or_interaction()`: Helper function to calculate the odds ratio estimates in case of an interaction. This is a wrapper for `h_or_cont_interaction()` and `h_or_cat_interaction()`.
- `h_simple_term_labels()`: Helper function to construct term labels from simple terms and the table of numbers of patients.
- `h_interaction_term_labels()`: Helper function to construct term labels from interaction terms and the table of numbers of patients.
- `h_glm_simple_term_extract()`: Helper function to tabulate the main effect results of a (conditional) logistic regression model.
- `h_glm_interaction_extract()`: Helper function to tabulate the interaction term results of a logistic regression model.
- `h_glm_inter_term_extract()`: Helper function to tabulate the interaction results of a logistic regression model. This basically is a wrapper for `h_or_interaction()` and `h_glm_simple_term_extract()` which puts the results in the right data frame format.
- `h_logistic_simple_terms()`: Helper function to tabulate the results including odds ratios and confidence intervals of simple terms.
- `h_logistic_inter_terms()`: Helper function to tabulate the results including odds ratios and confidence intervals of interaction terms.

Note

We don’t provide a function for the case when both variables are continuous because this does not arise in this table, as the treatment arm variable will always be involved and categorical.
Examples

library(dplyr)
library(broom)

adrs_f <- tern_ex_adrs %>%
  filter(PARAMCD == "BESRspi") %>%
  filter(RACE %in% c("ASIAN", "WHITE", "BLACK OR AFRICAN AMERICAN")) %>%
  mutate(
    Response = case_when(AVALC %in% c("PR", "CR") ~ 1, TRUE ~ 0),
    RACE = factor(RACE),
    SEX = factor(SEX)
  )
formatters::var_labels(adrs_f) <- c(formatters::var_labels(tern_ex_adrs), Response = "Response")

mod1 <- fit_logistic(
  data = adrs_f,
  variables = list(
    response = "Response",
    arm = "ARMCD",
    covariates = c("AGE", "RACE")
  )
)

mod2 <- fit_logistic(
  data = adrs_f,
  variables = list(
    response = "Response",
    arm = "ARMCD",
    covariates = c("AGE", "RACE"),
    interaction = "AGE"
  )
)

h_glm_simple_term_extract("AGE", mod1)

h_glm_simple_term_extract("ARMCD", mod1)

h_glm_interaction_extract("ARMCD:AGE", mod2)

h_glm_inter_term_extract("AGE", "ARMCD", mod2)

h_logistic_simple_terms("AGE", mod1)

h_logistic_inter_terms(c("RACE", "AGE", "ARMCD", "AGE:ARMCD"), mod2)

**h_map_for_count_abnormal**

*Helper Function to create a map dataframe that can be used in trim_levels_to_map split function.*
**Description**

[Stable]

Helper Function to create a map dataframe from the input dataset, which can be used as an argument in the `trim_levels_to_map` split function. Based on different method, the map is constructed differently.

**Usage**

```r
h_map_for_count_abnormal(
  df,
  variables = list(anl = "ANRIND", split_rows = c("PARAM"), range_low = "ANRLO",
                  range_high = "ANRHI"),
  abnormal = list(low = c("LOW", "LOW LOW"), high = c("HIGH", "HIGH HIGH")),
  method = c("default", "range"),
  na_level = "<Missing>"
)
```

**Arguments**

- `df` *(data.frame)*
  - data set containing all analysis variables.

- `variables` *(named list of string)*
  - list of additional analysis variables.

- `abnormal` *(named list)*
  - identifying the abnormal range level(s) in `df`. Based on the levels of abnormality of the input dataset, it can be something like `list(Low = "LOW LOW", High = "HIGH HIGH")` or `abnormal = list(Low = "LOW", High = "HIGH")`

- `method` *(string)*
  - indicates how the returned map will be constructed. Can be "default" or "range".

- `na_level` *(string)*
  - string used to replace all NA or empty values in the output.

**Value**

A map `data.frame`.

**Note**

If method is "default", the returned map will only have the abnormal directions that are observed in the `df`, and records with all normal values will be excluded to avoid error in creating layout. If method is "range", the returned map will be based on the rule that at least one observation with low range > 0 for low direction and at least one observation with high range is not missing for high direction.
Examples

```r
adlb <- df_explicit_na(tern_ex_adlb)

h_map_for_count_abnormal(
  df = adlb,
  variables = list(anl = "ANRIND", split_rows = c("LBCAT", "PARAM")),
  abnormal = list(low = c("LOW"), high = c("HIGH")),
  method = "default",
  na_level = "<Missing>"
)

df <- data.frame(
  USUBJID = c(rep("1", 4), rep("2", 4), rep("3", 4)),
  AVISIT = c(
    rep("WEEK 1", 2),
    rep("WEEK 2", 2),
    rep("WEEK 1", 2),
    rep("WEEK 2", 2),
    rep("WEEK 1", 2),
    rep("WEEK 2", 2)
  ),
  PARAM = rep(c("ALT", "CPR"), 6),
  ANRIND = c(
    "NORMAL", "NORMAL", "LOW",
    "HIGH", "LOW", "HIGH", "HIGH", rep("NORMAL", 4)
  ),
  ANRLO = rep(5, 12),
  ANRHI = rep(20, 12)
)

df$ANRIND <- factor(df$ANRIND, levels = c("LOW", "HIGH", "NORMAL"))
h_map_for_count_abnormal(
  df = df,
  variables = list(
    anl = "ANRIND",
    split_rows = c("PARAM"),
    range_low = "ANRLO",
    range_high = "ANRHI"
  ),
  abnormal = list(low = c("LOW"), high = c("HIGH")),
  method = "range",
  na_level = "<Missing>"
)
```

---

**h_odds_ratio**

*Helper Functions for Odds Ratio Estimation*

**Description**

[Stable]

Functions to calculate odds ratios in `estimate_odds_ratio()`.
Usage

or_glm(data, conf_level)

or_clogit(data, conf_level)

Arguments

data (data.frame)
data frame containing at least the variables rsp and grp, and optionally strata for or_clogit().

conf_level (proportion)
confidence level of the interval.

Value

A named list of elements or_ci and n_tot.

Functions

• or_glm(): Estimates the odds ratio based on stats::glm(). Note that there must be exactly 2 groups in data as specified by the grp variable.

• or_clogit(): estimates the odds ratio based on survival::clogit(). This is done for the whole data set including all groups, since the results are not the same as when doing pairwise comparisons between the groups.

See Also

odds_ratio

Examples

# Data with 2 groups.
data <- data.frame(
  rsp = as.logical(c(1, 1, 0, 1, 0, 0, 1, 1)),
  grp = letters[c(1, 1, 1, 2, 2, 2, 1, 2)],
  strata = letters[c(1, 2, 2, 2, 2, 2, 1, 2)],
  stringsAsFactors = TRUE
)

# Odds ratio based on glm.
or_glm(data, conf_level = 0.95)

# Data with 3 groups.
data <- data.frame(
  rsp = as.logical(c(1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0)),
  grp = letters[c(1, 1, 1, 2, 2, 2, 3, 3, 3, 3, 1, 1, 1, 2, 2, 2, 3, 3, 3, 3)],
  strata = LETTERS[c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2)],
  stringsAsFactors = TRUE
)
# Odds ratio based on stratified estimation by conditional logistic regression.
or_clogit(data, conf_level = 0.95)

## h_pkparam_sort

**Sort Data by PK PARAM Variable**

### Description

[Stable]

### Usage

```r
h_pkparam_sort(pk_data, key_var = "PARAMCD")
```

### Arguments

- `pk_data` (data.frame): Pharmacokinetics dataframe
- `key_var` (character): key variable used to merge `pk_data` and metadata created by `d_pkparam()`

### Value

A PK data.frame sorted by a PARAM variable.

### Examples

```r
library(dplyr)

adpp <- tern_ex_adpp %>% mutate(PKPARAM = factor(paste0(PARAM, " (", AVALU, ")")))
pk_ordered_data <- h_pkparam_sort(adpp)
```

## h_proportions

**Helper Functions for Calculating Proportion Confidence Intervals**

### Description

[Stable]

Functions to calculate different proportion confidence intervals for use in `estimate_proportion()`.
Usage

prop_wilson(rsp, conf_level, correct = FALSE)

prop_strat_wilson(
  rsp,
  strata,
  weights = NULL,
  conf_level = 0.95,
  max_iterations = NULL,
  correct = FALSE
)

prop_clopper_pearson(rsp, conf_level)

prop_wald(rsp, conf_level, correct = FALSE)

prop_agresti_coull(rsp, conf_level)

prop_jeffreys(rsp, conf_level)

Arguments

rsp (logical)
    whether each subject is a responder or not.

conf_level (proportion)
    confidence level of the interval.

correct (flag)
    apply continuity correction.

strata (factor)
    variable with one level per stratum and same length as rsp.

weights (numeric or NULL)
    weights for each level of the strata. If NULL, they are estimated using the iterative algorithm proposed in Yan and Su (2010) that minimizes the weighted squared length of the confidence interval.

max_iterations (count)
    maximum number of iterations for the iterative procedure used to find estimates of optimal weights.

Value

Confidence interval of a proportion.

Functions

- prop_wilson(): Calculates the Wilson interval by calling stats::prop.test(). Also referred to as Wilson score interval.
prop_strat_wilson(): Calculates the stratified Wilson confidence interval for unequal proportions as described in Yan and Su (2010)
prop_clopper_pearson(): Calculates the Clopper-Pearson interval by calling stats::binom.test(). Also referred to as the exact method.
prop_wald(): Calculates the Wald interval by following the usual textbook definition for a single proportion confidence interval using the normal approximation.
prop_agresti_coull(): Calculates the Agresti-Coull interval (created by Alan Agresti and Brent Coull) by (for 95% CI) adding two successes and two failures to the data and then using the Wald formula to construct a CI.
prop_jeffreys(): Calculates the Jeffreys interval, an equal-tailed interval based on the non-informative Jeffreys prior for a binomial proportion.

References

See Also
estimate_proportions, descriptive function d_proportion(), and helper functions strata_normal_quantile() and update_weights_strat_wilson().

Examples
rsp <- c(
  TRUE, TRUE, TRUE, TRUE, TRUE,
  TRUE, TRUE, TRUE, TRUE, TRUE,
  FALSE, FALSE, FALSE, FALSE, FALSE)
prop_wilson(rsp, conf_level = 0.9)
# Stratified Wilson confidence interval with unequal probabilities

set.seed(1)
rsp <- sample(c(TRUE, FALSE), 100, TRUE)
strata_data <- data.frame(
  "f1" = sample(c("a", "b"), 100, TRUE),
  "f2" = sample(c("x", "y", "z"), 100, TRUE),
stringsAsFactors = TRUE)
strata <- interaction(strata_data)
n_strata <- ncol(table(rsp, strata)) # Number of strata

prop_strat_wilson(
  rsp = rsp, strata = strata,
  conf_level = 0.90
)

# Not automatic setting of weights
prop_strat_wilson(
  rsp = rsp, strata = strata,
weights = rep(1 / n_strata, n_strata),
conf_level = 0.90
)

prop_clopper_pearson(rsp, conf_level = .95)

prop_wald(rsp, conf_level = 0.95)
prop_wald(rsp, conf_level = 0.95, correct = TRUE)

prop_agresti_coull(rsp, conf_level = 0.95)

prop_jeffreys(rsp, conf_level = 0.95)

## Helper Functions to Calculate Proportion Difference

Description

[Stable]

Usage

prop_diff_wald(rsp, grp, conf_level = 0.95, correct = FALSE)

prop_diff_ha(rsp, grp, conf_level)

prop_diff_nc(rsp, grp, conf_level, correct = FALSE)

prop_diff_cmh(rsp, grp, strata, conf_level = 0.95)

prop_diff_strat_nc(
  rsp,
  grp,
  strata,
  weights_method = c("cmh", "wilson_h"),
  conf_level = 0.95,
  correct = FALSE
)

Arguments

rsp (logical)
  whether each subject is a responder or not.

grp (factor)
  vector assigning observations to one out of two groups (e.g. reference and treatment group).
conf_level (proportion)
  confidence level of the interval.

correct (logical)
  whether to include the continuity correction. For further information, see stats::prop.test().

strata (factor)
  variable with one level per stratum and same length as rsp.

weights_method (string)
  weights method. Can be either "cmh" or "heuristic" and directs the way
  weights are estimated.

Value
  A named list of elements diff (proportion difference) and diff_ci (proportion difference confidence interval).

Functions
  • prop_diff_wald(): The Wald interval follows the usual textbook definition for a single proportion confidence interval using the normal approximation. It is possible to include a continuity correction for Wald’s interval.
  • prop_diff_ha(): Anderson-Hauck confidence interval.
  • prop_diff_nc(): Newcombe confidence interval. It is based on the Wilson score confidence interval for a single binomial proportion.
  • prop_diff_cmh(): Calculates the weighted difference. This is defined as the difference in response rates between the experimental treatment group and the control treatment group, adjusted for stratification factors by applying Cochran-Mantel-Haenszel (CMH) weights. For the CMH chi-squared test, use stats::mantelhaen.test().
  • prop_diff_strat_nc(): Calculates the stratified Newcombe confidence interval and difference in response rates between the experimental treatment group and the control treatment group, adjusted for stratification factors. This implementation follows closely the one proposed by Yan and Su (2010). Weights can be estimated from the heuristic proposed in prop_strat_wilson() or from CMH-derived weights (see prop_diff_cmh()).

References

See Also
  prop_diff() for implementation of these helper functions.

Examples
  # Wald confidence interval
  set.seed(2)
  rsp <- sample(c(TRUE, FALSE), replace = TRUE, size = 20)
  grp <- factor(c(rep("A", 10), rep("B", 10)))
```r
# Anderson-Hauck confidence interval
# "Mid" case: 3/4 respond in group A, 1/2 respond in group B.
rsp <- c(TRUE, FALSE, FALSE, TRUE, TRUE, TRUE)
grp <- factor(c("A", "B", "A", "B", "A"), levels = c("B", "A"))
prop_diff_wald(rsp = rsp, grp = grp, conf_level = 0.95, correct = FALSE)

# Edge case: Same proportion of response in A and B.
rsp <- c(TRUE, FALSE, TRUE, FALSE)
grp <- factor(c("A", "A", "B", "B"), levels = c("A", "B"))
prop_diff_wald(rsp = rsp, grp = grp, conf_level = 0.90)

# Newcombe confidence interval
set.seed(1)
rsp <- c(sample(c(TRUE, FALSE), size = 40, prob = c(3 / 4, 1 / 4), replace = TRUE),
sample(c(TRUE, FALSE), size = 40, prob = c(1 / 2, 1 / 2), replace = TRUE))
grp <- factor(rep(c("A", "B"), each = 40), levels = c("B", "A"))
table(rsp, grp)
prop_diff_nc(rsp = rsp, grp = grp, conf_level = 0.9)

# Cochran-Mantel-Haenszel confidence interval
set.seed(2)
rsp <- sample(c(TRUE, FALSE), 100, TRUE)
grp <- sample(c("Placebo", "Treatment"), 100, TRUE)
grp <- factor(grp, levels = c("Placebo", "Treatment"))
strata_data <- data.frame(
  "f1" = sample(c("a", "b"), 100, TRUE),
  "f2" = sample(c("x", "y", "z"), 100, TRUE),
  stringsAsFactors = TRUE)
prop_diff_cmh(rsp = rsp, grp = grp, strata = interaction(strata_data),
  conf_level = 0.9)

# Stratified Newcombe confidence interval
set.seed(2)
data_set <- data.frame(
  "rsp" = sample(c(TRUE, FALSE), 100, TRUE),
  "f1" = sample(c("a", "b"), 100, TRUE),
  "f2" = sample(c("x", "y", "z"), 100, TRUE),
  "grp" = sample(c("Placebo", "Treatment"), 100, TRUE),
  stringsAsFactors = TRUE)
prop_diff_strat_nc()
```

h_response_biomarkers_subgroups

Helper Functions for Tabulating Biomarker Effects on Binary Response by Subgroup

Description

[Stable]
Helper functions which are documented here separately to not confuse the user when reading about the user-facing functions.

Usage

h rsp to logistic variables(variables, biomarker)

h logistic mult cont df(variables, data, control = control logistic())

h_tab rsp one biomarker(df, vars, .indent mods = 0L)

Arguments

variables (named list of string)
list of additional analysis variables.

biomarker (string)
the name of the biomarker variable.

data (data.frame)
the dataset containing the variables to summarize.

control (named list)
controls for the response definition and the confidence level produced by control logistic().

df (data.frame)
results for a single biomarker, as part of what is returned by extract rsp biomarkers()
(it needs a couple of columns which are added by that high-level function relative to what is returned by h logistic mult cont df(), see the example).
vars (character)
the names of statistics to be reported among:

- n_tot: Total number of patients per group.
- n_rsp: Total number of responses per group.
- prop: Total response proportion per group.
- or: Odds ratio.
- ci: Confidence interval of odds ratio.
- pval: p-value of the effect. Note, the statistics n_tot, or and ci are required.

.indent_mods (named integer)
indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.

Value

- h_rsp_to_logistic_variables() returns a named list of elements response, arm, covariates, and strata.
- h_logistic_mult_cont_df() returns a data.frame containing estimates and statistics for the selected biomarkers.
- h_tab_rsp_one_biomarker() returns an rtables table object with the given statistics arranged in columns.

Functions

- h_rsp_to_logistic_variables(): helps with converting the "response" function variable list to the "logistic regression" variable list. The reason is that currently there is an inconsistency between the variable names accepted by extract_rsp_subgroups() and fit_logistic().
- h_logistic_mult_cont_df(): prepares estimates for number of responses, patients and overall response rate, as well as odds ratio estimates, confidence intervals and p-values, for multiple biomarkers in a given single data set. variables corresponds to names of variables found in data, passed as a named list and requires elements rsp and biomarkers (vector of continuous biomarker variables) and optionally covariates and strat.
- h_tab_rsp_one_biomarker(): prepares a single sub-table given a df_sub containing the results for a single biomarker.

Examples

library(dplyr)
library(forcats)

adrs <- tern_ex_adrs
adrs_labels <- formatters::var_labels(adrs)

adrs_f <- adrs %>%
  filter(PARAMCD == "BESRSP") %>%
  mutate(rsp = AVALC == "CR")
formatters::var_labels(adrs_f) <- c(adrs_labels, "Response")

# This is how the variable list is converted internally.
h_rsp_to_logistic_variables(
  variables = list(
    rsp = "RSP",
    covariates = c("A", "B"),
    strat = "D"
  ),
  biomarker = "AGE"
)

# For a single population, estimate separately the effects
# of two biomarkers.
df <- h_logistic_mult_cont_df(
  variables = list(
    rsp = "rsp",
    biomarkers = c("BMRKR1", "AGE"),
    covariates = "SEX"
  ),
  data = adrs_f
)

df

df

# If the data set is empty, still the corresponding rows with missings are returned.
h_coxreg_mult_cont_df(
  variables = list(
    rsp = "rsp",
    biomarkers = c("BMRKR1", "AGE"),
    covariates = "SEX",
    strat = "STRATA1"
  ),
  data = adrs_f[NULL, ]
)

# Starting from above `df`, zoom in on one biomarker and add required columns.
df1 <- df[1, ]
df1$subgroup <- "All patients"
df1$row_type <- "content"
df1$var <- "ALL"
df1$var_label <- "All patients"

h_tab_rsp_one_biomarker(
  df1,
  vars = c("n_tot", "n_rsp", "prop", "or", "ci", "pval")
)
Description

[Stable]
Helper functions that tabulate in a data frame statistics such as response rate and odds ratio for population subgroups.

Usage

h_proportion_df(rsp, arm)

h_proportion_subgroups_df(
  variables,
  data,
  groups_lists = list(),
  label_all = "All Patients"
)

h_odds_ratio_df(rsp, arm, strata_data = NULL, conf_level = 0.95, method = NULL)

h_odds_ratio_subgroups_df(
  variables,
  data,
  groups_lists = list(),
  conf_level = 0.95,
  method = NULL,
  label_all = "All Patients"
)

Arguments

rsp (logical)
whether each subject is a responder or not.

arm (factor)
the treatment group variable.

variables (named list of string)
list of additional analysis variables.

data (data.frame)
the dataset containing the variables to summarize.

groups_lists (named list of list)
optionally contains for each subgroups variable a list, which specifies the new group levels via the names and the levels that belong to it in the character vectors that are elements of the list.

label_all (string)
label for the total population analysis.

strata_data (factor, data.frame or NULL)
required if stratified analysis is performed.
conf_level  (proportion)
  confidence level of the interval.

method  (string)
specifies the test used to calculate the p-value for the difference between two proportions. For options, see s_test_proportion_diff(). Default is NULL so no test is performed.

Details
Main functionality is to prepare data for use in a layout-creating function.

Value

• h_proportion_df() returns a data.frame with columns arm, n, n_rsp, and prop.

• h_proportion_subgroups_df() returns a data.frame with columns arm, n, n_rsp, prop, subgroup, var, var_label, and row_type.

• h_odds_ratio_df() returns a data.frame with columns arm, n_tot, or, lcl, ucl, conf_level, and optionally pval and pval_label.

• h_odds_ratio_subgroups_df() returns a data.frame with columns arm, n_tot, or, lcl, ucl, conf_level, subgroup, var, var_label, and row_type.

Functions

• h_proportion_df(): helper to prepare a data frame of binary responses by arm.

• h_proportion_subgroups_df(): summarizes proportion of binary responses by arm and across subgroups in a data frame. Variables corresponds to the names of variables found in data, passed as a named list and requires elements rsp, arm and optionally subgroups. groups_lists optionally specifies groupings for subgroups variables.

• h_odds_ratio_df(): helper to prepare a data frame with estimates of the odds ratio between a treatment and a control arm.

• h_odds_ratio_subgroups_df(): summarizes estimates of the odds ratio between a treatment and a control arm across subgroups in a data frame. Variables corresponds to the names of variables found in data, passed as a named list and requires elements rsp, arm and optionally subgroups and strat. groups_lists optionally specifies groupings for subgroups variables.

Examples

library(dplyr)
library(forcats)

adrs <- tern_ex_adrs
adrs_labels <- formatters::var_labels(adrs)

adrs_f <- adrs %>%
  filter(PARAMCD == "BESRSPI") %>%
```r
define_subgroup <- function(adrs_f) {
  # Filter rows with ARM values indicating Drug X or Placebo
  filter(ARM %in% c("A: Drug X", "B: Placebo")) %>%
  droplevels() %>%
  mutate(
    # Reorder levels of factor to make the placebo group the reference arm.
    ARM = fct_relevel(ARM, "B: Placebo"),
    rsp = AVALC == "CR"
  )
}

formatters::var_labels(adrs_f) <- c(adrs_labels, "Response")

h_proportion_df(
  c(TRUE, FALSE, FALSE),
  arm = factor(c("A", "A", "B"), levels = c("A", "B"))
)

h_proportion_subgroups_df(
  variables = list(rsp = "rsp", arm = "ARM", subgroups = c("SEX", "BMRKR2")),
  data = adrs_f
)

# Define groupings for BMRKR2 levels.
variables = list(rsp = "rsp", arm = "ARM", subgroups = c("SEX", "BMRKR2")),
  data = adrs_f,
  groups_lists = list(
    BMRKR2 = list(
      "low" = "LOW",
      "low/medium" = c("LOW", "MEDIUM"),
      "low/medium/high" = c("LOW", "MEDIUM", "HIGH")
    )
  )
)

# Unstratified analysis.
odds_ratio_df(
  c(TRUE, FALSE, FALSE, TRUE),
  arm = factor(c("A", "A", "B", "B"), levels = c("A", "B"))
)

# Include p-value.
odds_ratio_df(adrs_f$rsp, adrs_f$ARM, method = "chisq")

# Stratified analysis.
odds_ratio_df(
  rsp = adrs_f$rsp,
  arm = adrs_f$ARM,
  strata_data = adrs_f[, c("STRATA1", "STRATA2")],
  method = "cmh"
)

# Unstratified analysis.
odds_ratio_subgroups_df(
  variables = list(rsp = "rsp", arm = "ARM", subgroups = c("SEX", "BMRKR2")),
  data = adrs_f
)
```

### Description

**[Stable]**

Split a dataframe into a non-nested list of subsets.

### Usage

```r
h_split_by_subgroups(data, subgroups, groups_lists = list())
```

### Arguments

- `data` *(data.frame)*: dataset to split.
subgroups (character)

names of factor variables from data used to create subsets. Unused levels not present in data are dropped. Note that the order in this vector determines the order in the downstream table.

groups_lists (named list of list)

optionally contains for each subgroups variable a list, which specifies the new group levels via the names and the levels that belong to it in the character vectors that are elements of the list.

Details

Main functionality is to prepare data for use in forest plot layouts.

Value

A list with subset data (df) and metadata about the subset (df_labels).

Examples

df <- data.frame(
  x = c(1:5),
  y = factor(c("A", "B", "A", "B", "A"), levels = c("A", "B", "C")),
  z = factor(c("C", "C", "D", "D", "D"), levels = c("D", "C"))
)
formatters::var_labels(df) <- paste("label for", names(df))

h_split_by_subgroups(
  data = df,
  subgroups = c("y", "z")
)

h_split_by_subgroups(
  data = df,
  subgroups = c("y", "z"),
  groups_lists = list(
    y = list("AB" = c("A", "B"), "C" = "C")
  )
)

---

**h_split_param**  
_Split parameters_

**Description**

**[Stable]**

It divides the data in the vector _param_ into the groups defined by _f_ based on specified values. It is relevant in _rtables_ layers so as to distribute parameters `.stats` or `.formats` into lists with items corresponding to specific analysis function.
Usage

h_split_param(param, value, f)

Arguments

param (vector)
the parameter to be split.

value (vector)
the value used to split.

f (list of vectors)
the reference to make the split

Value

A named list with the same element names as f, each containing the elements specified in .stats.

Examples

f <- list(
  surv = c("pt_at_risk", "event_free_rate", "rate_se", "rate_ci"),
  surv_diff = c("rate_diff", "rate_diff_ci", "ztest_pval")
)

.stats <- c("pt_at_risk", "rate_diff")
h_split_param(.stats, .stats, f = f)

# $surv
# [1] "pt_at_risk"
#
# $surv_diff
# [1] "rate_diff"

.formats <- c("pt_at_risk" = "xx", "event_free_rate" = "xxx")
h_split_param(.formats, names(.formats), f = f)

# $surv
# pt_at_risk event_free_rate
# "xx" "xxx"
#
# $surv_diff
# NULL
**h_stack_by_baskets**

**Description**

[Stable]

Helper Function to create a new SMQ variable in ADAE that consists of all adverse events belonging to selected Standardized/Customized queries. The new dataset will only contain records of the adverse events belonging to any of the selected baskets.

**Usage**

```r
h_stack_by_baskets(
  df,
  baskets = grep("^(SMQ|CQ).+NAM\$", names(df), value = TRUE),
  smq_varlabel = "Standardized MedDRA Query",
  keys = c("STUDYID", "USUBJID", "ASTDTM", "AEDECOD", "AESEQ"),
  aag_summary = NULL,
  na_level = "<Missing>"
)
```

**Arguments**

- **df** *(data.frame)*
  data set containing all analysis variables.

- **baskets** *(character)*
  variable names of the selected Standardized/Customized queries.

- **smq_varlabel** *(string)*
  a label for the new variable created.

- **keys** *(character)*
  names of the key variables to be returned along with the new variable created.

- **aag_summary** *(data.frame)*
  containing the SMQ baskets and the levels of interest for the final SMQ variable. This is useful when there are some levels of interest that are not observed in the df dataset. The two columns of this dataset should be named basket and basket_name.

- **na_level** *(string)*
  string used to replace all NA or empty values in the output.

**Value**

data.frame with variables in keys taken from df and new variable SMQ containing records belonging to the baskets selected via the baskets argument.

**Examples**

```r
adae <- tern_ex_adae[1:20, ] %>% df_explicit_na()

h_stack_by_baskets(df = adae)

aag <- data.frame(
  NAMVAR = c("CQ01NAM", "CQ02NAM", "SMQ01NAM", "SMQ02NAM"),
  ...)
```
h_step

Helper Functions for Subgroup Treatment Effect Pattern (STEP) Calculations

Description

[Stable]
Helper functions that are used internally for the STEP calculations.

Usage

h_step_window(x, control = control_step())

h_step_trt_effect(data, model, variables, x)

h_step_survival_formula(variables, control = control_step())
```r
h_step_survival_est(
    formula,
    data,
    variables,
    x,
    subset = rep(TRUE, nrow(data)),
    control = control_coxph()
)

h_step_rsp_formula(variables, control = c(control_step(), control_logistic()))

h_step_rsp_est(
    formula,
    data,
    variables,
    x,
    subset = rep(TRUE, nrow(data)),
    control = control_logistic()
)
```

**Arguments**

- `x` (numeric): biomarker value(s) to use (without NA).
- `control` (named list): output from `control_step()`.
- `data` (data.frame): the dataset containing the variables to summarize.
- `model` (data.frame): the regression model object.
- `variables` (named list of string): list of additional analysis variables.
- `formula` (formula): the regression model formula.
- `subset` (logical): subset vector.

**Value**

- `h_step_window()` returns a list containing the window-selection matrix `sel` and the interval information matrix `interval`.
- `h_step_trt_effect()` returns a vector with elements `est` and `se`.
- `h_step_survival_formula()` returns a model formula.
- `h_step_survival_est()` returns a matrix of number of observations `n`, events, log hazard ratio estimates `loghr`, standard error `se`, and Wald confidence interval bounds `ci_lower` and `ci_upper`. One row is included for each biomarker value in `x`. 
h_survival_biomarkers_subgroups

Description

[Stable]
Helper functions which are documented here separately to not confuse the user when reading about the user-facing functions.

Usage

h_surv_to_coxreg_variables(variables, biomarker)

h_coxreg_mult_cont_df(variables, data, control = control_coxreg())

h_tab_surv_one_biomarker(df, vars, time_unit, .indent_mods = 0L)

Arguments

variables (named list of string)
list of additional analysis variables.

biomarker (string)
the name of the biomarker variable.
data (data.frame) the dataset containing the variables to summarize.

control (list) a list of parameters as returned by the helper function control_coxreg().

df (data.frame) results for a single biomarker, as part of what is returned by extract_survival_biomarkers() (it needs a couple of columns which are added by that high-level function relative to what is returned by h_coxreg_mult_cont_df(), see the example).

df (data.frame) results for a single biomarker, as part of what is returned by extract_survival_biomarkers() (it needs a couple of columns which are added by that high-level function relative to what is returned by h_coxreg_mult_cont_df(), see the example).

vars (character) the names of statistics to be reported among:
  • n_tot_events: Total number of events per group.
  • n_tot: Total number of observations per group.
  • median: Median survival time.
  • hr: Hazard ratio.
  • ci: Confidence interval of hazard ratio.
  • pval: p-value of the effect. Note, one of the statistics n_tot and n_tot_events, as well as both hr and ci are required.

time_unit (string) label with unit of median survival time. Default NULL skips displaying unit.

.indent_mods (named integer) indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.

Value

• h_surv_to_coxreg_variables() returns a named list of elements time, event, arm, covariates, and strata.

• h_coxreg_mult_cont_df() returns a data.frame containing estimates and statistics for the selected biomarkers.

• h_tab_surv_one_biomarker() returns an rtables table object with the given statistics arranged in columns.

Functions

• h_surv_to_coxreg_variables(): helps with converting the "survival" function variable list to the "Cox regression" variable list. The reason is that currently there is an inconsistency between the variable names accepted by extract_survival_subgroups() and fit_coxreg_multivar().

• h_coxreg_mult_cont_df(): prepares estimates for number of events, patients and median survival times, as well as hazard ratio estimates, confidence intervals and p-values, for multiple biomarkers in a given single data set. variables corresponds to names of variables found in data, passed as a named list and requires elements tte, is_event, biomarkers (vector of continuous biomarker variables) and optionally subgroups and strat.

• h_tab_surv_one_biomarker(): prepares a single sub-table given a df_sub containing the results for a single biomarker.
Examples

```r
library(dplyr)
library(forcats)

adtte <- tern_ex_adtte

# Save variable labels before data processing steps.
adtte_labels <- formatters::var_labels(adtte, fill = FALSE)

adtte_f <- adtte %>%
  filter(PARAMCD == "OS") %>%
  mutate(
    AVALU = as.character(AVALU),
    is_event = CNSR == 0
  )

labels <- c("AVALU" = adtte_labels["AVALU"], "is_event" = "Event Flag")
formatters::var_labels(adtte_f)[names(labels)] <- labels

# This is how the variable list is converted internally.
h_surv_to_coxreg_variables(
  variables = list(
    tte = "AVAL", 
    is_event = "EVNT", 
    covariates = c("A", "B"), 
    strata = "D"
  ),
  biomarker = "AGE"
)

# For a single population, estimate separately the effects
# of two biomarkers.
df <- h_coxreg_mult_cont_df(
  variables = list(
    tte = "AVAL", 
    is_event = "is_event", 
    biomarkers = c("BMRKR1", "AGE"), 
    covariates = "SEX", 
    strata = c("STRATA1", "STRATA2")
  ),
  data = adtte_f
)

df

# If the data set is empty, still the corresponding rows with missings are returned.
h_coxreg_mult_cont_df(
  variables = list(
    tte = "AVAL", 
    is_event = "is_event", 
    biomarkers = c("BMRKR1", "AGE"), 
    covariates = "REGION1", 
    strata = c("STRATA1", "STRATA2")
  ),
)
data = adtte_f[NUL, ]

# Starting from above 'df', zoom in on one biomarker and add required columns.
df1 <- df[1, ]
df1$subgroup <- "All patients"
df1$row_type <- "content"
df1$var <- "ALL"
df1$var_label <- "All patients"
h_tab_surv_one_biomarker(
  df1,
  vars = c("n_tot", "n_tot_events", "median", "hr", "ci", "pval"),
  time_unit = "days"
)

---

### h_survival_duration_subgroups

**Helper Functions for Tabulating Survival Duration by Subgroup**

**Description**

[Stable]

Helper functions that tabulate in a data frame statistics such as median survival time and hazard ratio for population subgroups.

**Usage**

h_survtime_df(tte, is_event, arm)

h_survtime_subgroups_df(
  variables,
  data,
  groups_lists = list(),
  label_all = "All Patients"
)

h_coxph_df(tte, is_event, arm, strata_data = NULL, control = control_coxph())

h_coxph_subgroups_df(
  variables,
  data,
  groups_lists = list(),
  control = control_coxph(),
  label_all = "All Patients"
)

---
Arguments

tte (numeric)
contains time-to-event duration values.

is_event (logical)
TRUE if event, FALSE if time to event is censored.

arm (factor)
the treatment group variable.

variables (named list of string)
list of additional analysis variables.

data (data.frame)
the dataset containing the variables to summarize.

groups_lists (named list of list)
optionally contains for each subgroups variable a list, which specifies the new group levels via the names and the levels that belong to it in the character vectors that are elements of the list.

label_all (string)
label for the total population analysis.

strata_data (factor, data.frame or NULL)
required if stratified analysis is performed.

control (list)
parameters for comparison details, specified by using the helper function control_coxph(). Some possible parameter options are:

• pval_method (string)
p-value method for testing hazard ratio = 1. Default method is "log-rank" which comes from survival::survdiff(), can also be set to "wald" or "likelihood" (from survival::coxph()).

• ties (string)
    specifying the method for tie handling. Default is "efron", can also be set to "breslow" or "exact". See more in survival::coxph()

• conf_level (proportion)
    confidence level of the interval for HR.

Details

Main functionality is to prepare data for use in a layout-creating function.

Value

• h_survtime_df() returns a data.frame with columns arm, n, n_events, and median.

• h_survtime_subgroups_df() returns a data.frame with columns arm, n, n_events, median, subgroup, var, var_label, and row_type.

• h_coxph_df() returns a data.frame with columns arm, n_tot, n_tot_events, hr, lcl, ucl, conf_level, pval and pval_label.

• h_coxph_subgroups_df() returns a data.frame with columns arm, n_tot, n_tot_events, hr, lcl, ucl, conf_level, pval, pval_label, subgroup, var, var_label, and row_type.
Functions

- `h_survtime_df()`: helper to prepare a data frame of median survival times by arm.
- `h_survtime_subgroups_df()`: summarizes median survival times by arm and across subgroups in a data frame. Variables corresponds to the names of variables found in data, passed as a named list and requires elements `tte, is_event, arm` and optionally `subgroups`.
- `groups_lists` optionally specifies groupings for subgroups variables.
- `h_coxph_df()`: helper to prepare a data frame with estimates of treatment hazard ratio.
- `h_coxph_subgroups_df()`: summarizes estimates of the treatment hazard ratio across subgroups in a data frame. Variables corresponds to the names of variables found in data, passed as a named list and requires elements `tte, is_event, arm` and optionally `subgroups` and `strat`. `groups_lists` optionally specifies groupings for subgroups variables.

Examples

```r
library(dplyr)
library(forcats)

adtte <- tern_ex_adtte

# Save variable labels before data processing steps.
adtte_labels <- formatters::var_labels(adtte)

adtte_f <- adtte %>%
  filter(
    PARAMCD == "OS",
    ARM %in% c("B: Placebo", "A: Drug X"),
    SEX %in% c("M", "F")
  ) %>%
  mutate(
    # Reorder levels of ARM to display reference arm before treatment arm.
    ARM = droplevels(fct_relevel(ARM, "B: Placebo")),
    SEX = droplevels(SEX),
    is_event = CNSR == 0
  )

labels <- c("ARM" = adtte_labels["ARM"], "SEX" = adtte_labels["SEX"], "is_event" = "Event Flag")
formatters::var_labels(adtte_f)[names(labels)] <- labels

# Extract median survival time for one group.
h_survtime_df(
  tte = adtte_f$AVAL,
  is_event = adtte_f$is_event,
  arm = adtte_f$ARM
)

# Extract median survival time for multiple groups.
variables = list(
  tte = "AVAL",
  is_event = "is_event",
  arm = "ARM",

```

subgroups = c("SEX", "BMRKR2")
),
data = adtte_f
)

# Define groupings for BMRKR2 levels.
h_survttime_subgroups_df(
  variables = list(
    tte = "AVAL",
    is_event = "is_event",
    arm = "ARM",
    subgroups = c("SEX", "BMRKR2")
  ),
data = adtte_f,
groups_lists = list(
  BMRKR2 = list(
    "low" = "LOW",
    "low/medium" = c("LOW", "MEDIUM"),
    "low/medium/high" = c("LOW", "MEDIUM", "HIGH")
  )
)
)

# Extract hazard ratio for one group.
h_coxph_df(adtte_f$AVAL, adtte_f$is_event, adtte_f$ARM)

# Extract hazard ratio for one group with stratification factor.
h_coxph_df(adtte_f$AVAL, adtte_f$is_event, adtte_f$ARM, strata_data = adtte_f$STRATA1)

# Extract hazard ratio for multiple groups.
h_coxph_subgroups_df(
  variables = list(
    tte = "AVAL",
    is_event = "is_event",
    arm = "ARM",
    subgroups = c("SEX", "BMRKR2")
  ),
data = adtte_f
)

# Define groupings of BMRKR2 levels.
h_coxph_subgroups_df(
  variables = list(
    tte = "AVAL",
    is_event = "is_event",
    arm = "ARM",
    subgroups = c("SEX", "BMRKR2")
  ),
data = adtte_f,
groups_lists = list(
  BMRKR2 = list(
    "low" = "LOW",
    "low/medium" = c("LOW", "MEDIUM")
  )
)
h_tab_one_biomarker

"low/medium/high" = c("LOW", "MEDIUM", "HIGH")
)
)
)

# Extract hazard ratio for multiple groups with stratification factors.

h_coxph_subgroups_df(
  variables = list(
    tte = "AVAL",
    is_event = "is_event",
    arm = "ARM",
    subgroups = c("SEX", "BMRKR2"),
    strat = c("STRATA1", "STRATA2")
  ),
  data = adtte_f
)

h_tab_one_biomarker  Helper Function for Tabulation of a Single Biomarker Result

Description

[Stable]

Please see h_tab_surv_one_biomarker() and h_tab_rsp_one_biomarker(), which use this function for examples. This function is a wrapper for rtables::summarize_row_groups().

Usage

h_tab_one_biomarker(df, afuns, colvars, .indent.mods = 0L)

Arguments

df (data.frame) results for a single biomarker.

afuns (named list of function) analysis functions.

colvars (list with vars and labels) variables to tabulate and their labels.

.indent.mods (named integer) indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.

Value

An rtables table object with statistics in columns.
h_tbl_coxph_pairwise  Helper Function: Pairwise CoxPH table

Description

[Stable]
Create a data.frame of pairwise stratified or unstratified CoxPH analysis results.

Usage

h_tbl_coxph_pairwise(df, variables, control_coxph_pw = control_coxph())

Arguments

df  (data.frame)  data set containing all analysis variables.
variables  (named list)  variable names. Details are:
  • tte (numeric)  variable indicating time-to-event duration values.
  • is_event (logical)  event variable. TRUE if event, FALSE if time to event is censored.
  • arm (factor)  the treatment group variable.
  • strat (character or NULL)  variable names indicating stratification factors.

control_coxph_pw  (list)  parameters for comparison details, specified by using the helper function control_coxph(). Some possible parameter options are:
  • pval_method (string)  p-value method for testing hazard ratio = 1. Default method is "log-rank". can also be set to "wald" or "likelihood".
  • ties (string)  method for tie handling. Default is "efron", can also be set to "breslow" or "exact". See more in survival::coxph()
  • conf_level (proportion)  confidence level of the interval for HR.

Value

A data.frame containing statistics HR, XX% CI (XX taken from control_coxph_pw), and p-value (log-rank).
Examples

library(dplyr)

adtte <- tern_ex_adtte %>%
  filter(PARAMCD == "OS") %>%
  mutate(is_event = CNSR == 0)

h_tbl_coxph_pairwise(
  df = adtte,
  variables = list(tte = "AVAL", is_event = "is_event", arm = "ARM"),
  control_coxph_pw = control_coxph(conf_level = 0.9)
)

h_tbl_median_surv

Helper Function: Survival Estimations

Description

[Stable]
Transform a survival fit to a table with groups in rows characterized by N, median and confidence interval.

Usage

h_tbl_median_surv(fit_km, armval = "All")

Arguments

fit_km (survfit) result of survival::survfit().
armval (string) used as strata name when treatment arm variable only has one level. Default is "All".

Value

A summary table with statistics N, Median, and XX% CI (XX taken from fit_km).

Examples

library(dplyr)
library(survival)

adtte <- tern_ex_adtte %>% filter(PARAMCD == "OS"
fit <- survfit(
  form = Surv(AVAL, 1 - CNSR) ~ ARMCD,
  data = adtte
)
h_tbl_median_surv(fit_km = fit)

h_worsen_counter

Helper Function to Analyze Patients for
s_count_abnormal_lab_worsen_by_baseline()

Description

[Stable]

Helper function to count the number of patients and the fraction of patients according to highest post-baseline lab grade variable .var, baseline lab grade variable baseline_var, and the direction of interest specified in direction_var.

Usage

h_worsen_counter(df, id, .var, baseline_var, direction_var)

Arguments

df (data.frame)
data set containing all analysis variables.
id (string)subject variable name.
.var (string)
single variable name that is passed by rtables when requested by a statistics function.
baseline_var (string)
baseline lab grade variable
direction_var (string)
Direction variable specifying the direction of the shift table of interest. Only lab records flagged by L, H or B are included in the shift table.
  • L: low direction only
  • H: high direction only
  • B: both low and high directions

Value

h_worsen_counter() returns the counts and fraction of patients whose worst post-baseline lab grades are worse than their baseline grades, for post-baseline worst grades "1", "2", "3", "4" and "Any".
See Also

`abnormal_by_worst_grade_worsen`

Examples

```r
library(dplyr)

# The direction variable, GRADDR, is based on metadata
adlb <- tern_ex_adlb %>%
  mutate(
    GRADDR = case_when(
      PARAMCD == "ALT" ~ "B",
      PARAMCD == "CRP" ~ "L",
      PARAMCD == "IGA" ~ "H"
    )
  ) %>%
  filter(SAFFL == "Y" & ONTRTFL == "Y" & GRADDR != "")

df <- h_adlb_worsen(
  adlb,
  worst_flag_low = c("WGRLOFL" = "Y"),
  worst_flag_high = c("WGRHIFL" = "Y"),
  direction_var = "GRADDR"
)

# `h_worsen_counter`

h_worsen_counter(
  df %>% filter(PARAMCD == "CRP" & GRADDR == "Low"),
  id = "USUBJID",
  .var = "ATOXGR",
  baseline_var = "BTOXGR",
  direction_var = "GRADDR"
)
```

---

**h_xticks**

**Helper function: x tick positions**

Description

[Stable]

Calculate the positions of ticks on the x-axis. However, if xticks already exists it is kept as is. It is based on the same function ggplot2 relies on, and is required in the graphic and the patient-at-risk annotation table.

Usage

```r
h_xticks(data, xticks = NULL, max_time = NULL)
```
individual_patient_plot

Individual Patient Plots

Description

[Stable]

Line plot(s) displaying trend in patients’ parameter values over time is rendered. Patients’ individual baseline values can be added to the plot(s) as reference.
Usage

\[
g_ipp(\ 
    df, \ 
    xvar, \ 
    yvar, \ 
    xlab, \ 
    ylab, \ 
    id_var = "USUBJID", \ 
    title = "Individual Patient Plots", \ 
    subtitle = "", \ 
    caption = NULL, \ 
    add_baseline_hline = FALSE, \ 
    yvar_baseline = "BASE", \ 
    ggtheme = nestcolor\:\:theme_nest(), \ 
    plotting_choices = c("all_in_one", "split_by_max_obs", "separate_by_obs"), \ 
    max_obs_per_plot = 4, \ 
    col = NULL \ 
)
\]

Arguments

- **df** *(data.frame)*
  data set containing all analysis variables.

- **xvar** *(string)*
  time point variable to be plotted on x-axis.

- **yvar** *(string)*
  continuous analysis variable to be plotted on y-axis.

- **xlab** *(string)*
  plot label for x-axis.

- **ylab** *(string)*
  plot label for y-axis.

- **id_var** *(string)*
  variable used as patient identifier.

- **title** *(string)*
  title for plot.

- **subtitle** *(string)*
  subtitle for plot.

- **caption** *(character scalar)*
  optional caption below the plot.

- **add_baseline_hline** *(flag)*
  adds horizontal line at baseline y-value on plot when TRUE.

- **yvar_baseline** *(string)*
  variable with baseline values only. Ignored when add_baseline_hline is FALSE.
individual_patient_plot

ggtheme (theme)
optional graphical theme function as provided by ggplot2 to control outlook of
plot. Use ggplot2::theme() to tweak the display.

plotting_choices (character)
specifies options for displaying plots. Must be one of "all_in_one", "split_by_max_obs",
"separate_by_obs".

max_obs_per_plot (count)
Number of observations to be plotted on one plot. Ignored when plotting_choices
is not "separate_by_obs".

col (character)
lines colors.

Value
A ggplot object or a list of ggplot objects.

Functions

• g_ipp(): Plotting function for individual patient plots which, depending on user preference,
renders a single graphic or compiles a list of graphics that show trends in individual’s param-
eter values over time.

See Also
Relevant helper function h_g_ipp().

Examples

library(dplyr)
library(nestcolor)

# Select a small sample of data to plot.
adlb <- tern_ex_adlb %>%
  filter(PARAMCD == "ALT", !(AVISIT %in% c("SCREENING", "BASELINE"))) %>%
slice(1:36)

plot_list <- g_ipp(
  df = adlb,
  xvar = "AVISIT",
  yvar = "AVAL",
  xlab = "Visit",
  ylab = "SGOT/ALT (U/L)",
  title = "Individual Patient Plots",
  add_baseline_hline = TRUE,
  plotting_choices = "split_by_max_obs",
  max_obs_per_plot = 5
)
plot_list
logistic_regression_cols

(Logistic Regression Multivariate Column Layout Function)

Description
[Stable]
Layout-creating function which creates a multivariate column layout summarizing logistic regression results. This function is a wrapper for \texttt{rtables::split_cols_by_multivar()}.

Usage
\begin{verbatim}
logistic_regression_cols(lyt, conf_level = 0.95)
\end{verbatim}

Arguments
\begin{verbatim}
lyt (layout)
input layout where analyses will be added to.
conf_level (proportion)
confidence level of the interval.
\end{verbatim}

Value
A layout object suitable for passing to further layouting functions. Adding this function to an rtable layout will split the table into columns corresponding to statistics \texttt{df}, \texttt{estimate}, \texttt{std_error}, \texttt{odds_ratio}, \texttt{ci}, and \texttt{pvalue}.

logistic_summary_by_flag

(Logistic Regression Summary Table Constructor Function)

Description
[Stable]
Constructor for content functions to be used in \texttt{summarize_logistic()} to summarize logistic regression results. This function is a wrapper for \texttt{rtables::summarize_row_groups()}.

Usage
\begin{verbatim}
logistic_summary_by_flag(flag_var, .indent_mods = NULL)
\end{verbatim}
Arguments

flag_var (string)
variable name identifying which row should be used in this content function.

.indent_mods (named integer)
indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.

Value

A content function.

Description

[Stable]
Conversion of Months to Days. This is an approximative calculation because it considers each month as having an average of 30.4375 days.

Usage

month2day(x)

Arguments

x (numeric)
time in months.

Value

A numeric vector with the time in days.

Examples

x <- c(13.25, 8.15, 1, 2.834)
month2day(x)
odds_ratio

Odds Ratio Estimation

**Description**

[Stable]

Compares bivariate responses between two groups in terms of odds ratios along with a confidence interval.

**Usage**

```r
s_odds_ratio(
  df,
  .var,
  .ref_group,
  .in_ref_col,
  .df_row,
  variables = list(arm = NULL, strata = NULL),
  conf_level = 0.95,
  groups_list = NULL
)
```

```r
a_odds_ratio(
  df,
  .var,
  .ref_group,
  .in_ref_col,
  .df_row,
  variables = list(arm = NULL, strata = NULL),
  conf_level = 0.95,
  groups_list = NULL
)
```

```r
estimate_odds_ratio(
  lyt,
  vars,
  ...,
  show_labels = "hidden",
  table_names = vars,
  .stats = "or_ci",
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```
Arguments

- **df** (data.frame)
  - data set containing all analysis variables.
- **.var** (string)
  - single variable name that is passed by rtables when requested by a statistics function.
- **.ref_group** (data.frame or vector)
  - the data corresponding to the reference group.
- **.in_ref_col** (logical)
  - TRUE when working with the reference level, FALSE otherwise.
- **.df_row** (data.frame)
  - data frame across all of the columns for the given row split.
- **variables** (named list of string)
  - list of additional analysis variables.
- **conf_level** (proportion)
  - confidence level of the interval.
- **groups_list** (named list of character)
  - specifies the new group levels via the names and the levels that belong to it in the character vectors that are elements of the list.
- **lyt** (layout)
  - input layout where analyses will be added to.
- **vars** (character)
  - variable names for the primary analysis variable to be iterated over.
- **...**
  - arguments passed to s_odds_ratio().
- **show_labels** (string)
  - label visibility: one of "default", "visible" and "hidden".
- **table_names** (character)
  - this can be customized in case that the same vars are analyzed multiple times, to avoid warnings from rtables.
- **.stats** (character)
  - statistics to select for the table.
- **.formats** (named character or list)
  - formats for the statistics.
- **.labels** (named character)
  - labels for the statistics (without indent).
- **.indent_mods** (named integer)
  - indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.

Details

This function uses either logistic regression for unstratified analyses, or conditional logistic regression for stratified analyses. The Wald confidence interval with the specified confidence level is calculated.
odds_ratio

Value

• `s_odds_ratio()` returns a named list with the statistics or_ci (containing est, lcl, and ucl) and n_tot.

• `a_odds_ratio()` returns the corresponding list with formatted `rtables::CellValue()`.

• `estimate_odds_ratio()` returns a layout object suitable for passing to further layouting functions, or to `rtables::build_table()`. Adding this function to an rtable layout will add formatted rows containing the statistics from `s_odds_ratio()` to the table layout.

Functions

• `s_odds_ratio()`: Statistics function which estimates the odds ratio between a treatment and a control. A variables list with arm and strata variable names must be passed if a stratified analysis is required.

• `a_odds_ratio()`: Formatted analysis function which is used as afun in `estimate_odds_ratio()`.

• `estimate_odds_ratio()`: Layout-creating function which can take statistics function arguments and additional format arguments. This function is a wrapper for `rtables::analyze()`.

Note

For stratified analyses, there is currently no implementation for conditional likelihood confidence intervals, therefore the likelihood confidence interval is not yet available as an option. Besides, when rsp contains only responders or non-responders, then the result values will be NA, because no odds ratio estimation is possible.

See Also

Relevant helper function `h_odds_ratio()`.

Examples

```r
set.seed(12)
dta <- data.frame(
  rsp = sample(c(TRUE, FALSE), 100, TRUE),
  grp = factor(rep(c("A", "B"), each = 50), levels = c("B", "A")),
  strata = factor(sample(c("C", "D"), 100, TRUE))
)

# Unstratified analysis.
s_odds_ratio(
  df = subset(dta, grp == "A"),
  .var = "rsp",
  .ref_group = subset(dta, grp == "B"),
  .in_ref_col = FALSE,
  .df_row = dta
)

# Stratified analysis.
s_odds_ratio(
  df = subset(dta, strata == "C"),
  .var = "rsp",
  .ref_group = subset(dta, strata == "D"),
  .in_ref_col = FALSE,
  .df_row = dta
)
```

df = subset(dta, grp == "A"),
   .var = "rsp",
   .ref_group = subset(dta, grp == "B"),
   .in_ref_col = FALSE,
   .df_row = dta,
   variables = list(arm = "grp", strata = "strata")
)

a_odds_ratio(
   df = subset(dta, grp == "A"),
   .var = "rsp",
   .ref_group = subset(dta, grp == "B"),
   .in_ref_col = FALSE,
   .df_row = dta
)

dta <- data.frame(
   rsp = sample(c(TRUE, FALSE), 100, TRUE),
   grp = factor(rep(c("A", "B"), each = 50))
)

l <- basic_table() %>%
   split_cols_by(var = "grp", ref_group = "B") %>%
   estimate_odds_ratio(vars = "rsp")

build_table(l, df = dta)

---

prop_diff

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Stable]</td>
</tr>
</tbody>
</table>

Usage

s_proportion_diff(
   df,
   .var,
   .ref_group,
   .in_ref_col,
   variables = list(strata = NULL),
   conf_level = 0.95,
   method = c("waldcc", "wald", "cmh", "ha", "newcombe", "newcombecc", "strat_newcombe",
     "strat_newcombecc"),
   weights_method = "cmh"
)
a_proportion_diff(
  df,
  .var,
  .ref_group,
  .in_ref_col,
  variables = list(strata = NULL),
  conf_level = 0.95,
  method = c("waldcc", "wald", "cmh", "ha", "newcombe", "newcombecc", "strat_newcombe",
             "strat_newcombecc"),
  weights_method = "cmh"
)

estimate_proportion_diff(
  lyt,
  vars,
  ...,
  var_labels = vars,
  show_labels = "hidden",
  table_names = vars,
  .stats = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)

Arguments

df (data.frame)
  data set containing all analysis variables.
.var (string)
  single variable name that is passed by rtables when requested by a statistics
function.
.ref_group (data.frame or vector)
  the data corresponding to the reference group.
.in_ref_col (logical)
  TRUE when working with the reference level, FALSE otherwise.
variables (named list of string)
  list of additional analysis variables.
conf_level (proportion)
  confidence level of the interval.
method (string)
  the method used for the confidence interval estimation.
weights_method (string)
  weights method. Can be either "cmh" or "heuristic" and directs the way
weights are estimated.
lyt (layout)
  input layout where analyses will be added to.
vars  (character)
variable names for the primary analysis variable to be iterated over.

... arguments passed to s_proportion_diff().

var_labels (character)
character for label.

show_labels (string)
label visibility: one of "default", "visible" and "hidden".

table_names (character)
this can be customized in case that the same vars are analyzed multiple times,
to avoid warnings from rtables.

.stats (character)
statistics to select for the table.

.formats (named character or list)
formats for the statistics.

.labels (named character)
labels for the statistics (without indent).

.indent_mods (named integer)
indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.

Value

- s_proportion_diff() returns a named list of elements diff and diff_ci.
- a_proportion_diff() returns the corresponding list with formatted rtables::CellValue().
- estimate_proportion_diff() returns a layout object suitable for passing to further layouting functions, or to rtables::build_table(). Adding this function to an rtable layout will add formatted rows containing the statistics from s_proportion_diff() to the table layout.

Functions

- s_proportion_diff(): Statistics function estimating the difference in terms of responder proportion.
- a_proportion_diff(): Formatted analysis function which is used as afun in estimate_proportion_diff().
- estimate_proportion_diff(): Layout-creating function which can take statistics function arguments and additional format arguments. This function is a wrapper for rtables::analyze().

Note

When performing an unstratified analysis, methods "cmh", "strat_newcombe", and "strat_newcombecc" are not permitted.

See Also

d_proportion_diff()
Examples

## Summary

"Mid" case: 4/4 respond in group A, 1/2 respond in group B.

```r
# Number of example rows
nex <- 100

# Data frame
dta <- data.frame(
    "rsp" = sample(c(TRUE, FALSE), nex, TRUE),
    "grp" = sample(c("A", "B"), nex, TRUE),
    "f1" = sample(c("a1", "a2"), nex, TRUE),
    "f2" = sample(c("x", "y", "z"), nex, TRUE),
    stringsAsFactors = TRUE
)

# Proportion difference
s_proportion_diff(
    df = subset(dta, grp == "A"),
    .var = "rsp",
    .ref_group = subset(dta, grp == "B"),
    .in_ref_col = FALSE,
    conf_level = 0.90,
    method = "ha"
)

# CMH example with strata
s_proportion_diff(
    df = subset(dta, grp == "A"),
    .var = "rsp",
    .ref_group = subset(dta, grp == "B"),
    .in_ref_col = FALSE,
    variables = list(strata = c("f1", "f2")),
    conf_level = 0.90,
    method = "cmh"
)

a_proportion_diff(
    df = subset(dta, grp == "A"),
    .var = "rsp",
    .ref_group = subset(dta, grp == "B"),
    .in_ref_col = FALSE,
    conf_level = 0.90,
    method = "ha"
)

l <- basic_table() %>%
    split_cols_by(var = "grp", ref_group = "B") %>%
    estimate_proportion_diff(
        vars = "rsp",
        conf_level = 0.90,
        method = "ha"
    )

build_table(l, df = dta)
```
prune_occurrences

Occurrence Table Pruning

Description

[Stable]

Family of constructor and condition functions to flexibly prune occurrence tables. The condition functions always return whether the row result is higher than the threshold. Since they are of class \texttt{CombinationFunction()} they can be logically combined with other condition functions.

Usage

\begin{verbatim}
keep_rows(row_condition)

keep_content_rows(content_row_condition)

has_count_in_cols(atleast, ...)

has_count_in_any_col(atleast, ...)

has_fraction_in_cols(atleast, ...)

has_fraction_in_any_col(atleast, ...)

has_fractions_difference(atleast, ...)

has_counts_difference(atleast, ...)
\end{verbatim}

Arguments

\begin{verbatim}
row_condition (CombinationFunction)
condition function which works on individual analysis rows and flags whether these should be kept in the pruned table.

content_row_condition
(CombinationFunction)
condition function which works on individual first content rows of leaf tables and flags whether these leaf tables should be kept in the pruned table.

atleast (count or proportion)
threshold which should be met in order to keep the row.

... arguments for row or column access, see \texttt{rtables_access}: either col_names (character) including the names of the columns which should be used, or alternatively col_indices (integer) giving the indices directly instead.
\end{verbatim}
**Value**

- `keep_rows()` returns a pruning function that can be used with `rtables::prune_table()` to prune an `rtables` table.

- `keep_content_rows()` returns a pruning function that checks the condition on the first content row of leaf tables in the table.

- `has_count_in_cols()` returns a condition function that sums the counts in the specified column.

- `has_count_in_any_col()` returns a condition function that compares the counts in the specified columns with the threshold.

- `has_fraction_in_cols()` returns a condition function that sums the counts in the specified column, and computes the fraction by dividing by the total column counts.

- `has_fraction_in_any_col()` returns a condition function that looks at the fractions in the specified columns and checks whether any of them fulfill the threshold.

- `has_fractions_difference()` returns a condition function that extracts the fractions of each specified column, and computes the difference of the minimum and maximum.

- `has_counts_difference()` returns a condition function that extracts the counts of each specified column, and computes the difference of the minimum and maximum.

**Functions**

- `keep_rows()`: Constructor for creating pruning functions based on a row condition function. This removes all analysis rows (`TableRow`) that should be pruned, i.e., don’t fulfill the row condition. It removes the sub-tree if there are no children left.

- `keep_content_rows()`: Constructor for creating pruning functions based on a condition for the (first) content row in leaf tables. This removes all leaf tables where the first content row does not fulfill the condition. It does not check individual rows. It then proceeds recursively by removing the sub tree if there are no children left.

- `has_count_in_cols()`: Constructor for creating condition functions on total counts in the specified columns.

- `has_count_in_any_col()`: Constructor for creating condition functions on any of the counts in the specified columns satisfying a threshold.

- `has_fraction_in_cols()`: Constructor for creating condition functions on total fraction in the specified columns.

- `has_fraction_in_any_col()`: Constructor for creating condition functions on any fraction in the specified columns.

- `has_fractions_difference()`: Constructor for creating condition function that checks the difference between the fractions reported in each specified column.

- `has_counts_difference()`: Constructor for creating condition function that checks the difference between the counts reported in each specified column.
Note
Since most table specifications are worded positively, we name our constructor and condition functions positively, too. However, note that the result of `keep_rows()` says what should be pruned, to conform with the `rtables::prune_table()` interface.

Examples

```r
tab <- basic_table() %>%
  split_cols_by(“ARM”) %>%
  split_rows_by(“RACE”) %>%
  split_rows_by(“STRATA1”) %>%
  summarize_row_groups() %>%
  summarize_vars(“COUNTRY”, .stats = “count_fraction”) %>%
  build_table(DM)

# ‘keep_rows’
is_non_empty <- !CombinationFunction(all_zero_or_na)
prune_table(tab, keep_rows(is_non_empty))

# ‘keep_content_rows’
more_than_twenty <- has_count_in_cols(atleast = 20L, col_names = names(tab))
prune_table(tab, keep_content_rows(more_than_twenty))

more_than_one <- has_count_in_cols(atleast = 1L, col_names = names(tab))
prune_table(tab, keep_rows(more_than_one))

# ‘has_count_in_any_col’
any_more_than_one <- has_count_in_any_col(atleast = 1L, col_names = names(tab))
prune_table(tab, keep_rows(any_more_than_one))

# ‘has_fraction_in_cols’
more_than_five_percent <- has_fraction_in_cols(atleast = 0.05, col_names = names(tab))
prune_table(tab, keep_rows(more_than_five_percent))

# ‘has_fraction_in_any_col’
any_atleast_five_percent <- has_fraction_in_any_col(atleast = 0.05, col_names = names(tab))
prune_table(tab, keep_rows(more_than_five_percent))
```
reapply_varlabels

Description

This is a helper function that is used in tests.

Usage

reapply_varlabels(x, varlabels, ...)

Arguments

x (vector)
vector of elements that needs new labels.

varlabels (character)
vector of labels for x.

... further parameters to be added to the list.

Value

x with variable labels reapplied.

response_biomarkers_subgroups

Tabulate Biomarker Effects on Binary Response by Subgroup

Description

[Stable]
Tabulate the estimated effects of multiple continuous biomarker variables on a binary response endpoint across population subgroups.
Usage

\texttt{tabulate_rsp_biomarkers(}
\texttt{  df,}
\texttt{  vars = c("n\_tot", "n\_rsp", "prop", "or", "ci", "pval"),}
\texttt{  .indent\_mods = 0L}
\texttt{)}

Arguments

\texttt{df}  
\begin{itemize}
  \item \texttt{(data.frame)}
  \item containing all analysis variables, as returned by \texttt{extract_rsp_biomarkers()}.\end{itemize}

\texttt{vars}  
\begin{itemize}
  \item \texttt{(character)}
  \item the names of statistics to be reported among:
  \begin{itemize}
    \item \texttt{n\_tot}: Total number of patients per group.
    \item \texttt{n\_rsp}: Total number of responses per group.
    \item \texttt{prop}: Total response proportion per group.
    \item \texttt{or}: Odds ratio.
    \item \texttt{ci}: Confidence interval of odds ratio.
    \item \texttt{pval}: p-value of the effect. Note, the statistics \texttt{n\_tot}, \texttt{or} and \texttt{ci} are required.\end{itemize}\end{itemize}

\texttt{.indent\_mods}  
\begin{itemize}
  \item \texttt{(named integer)}
  \item indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.\end{itemize}

Details

These functions create a layout starting from a data frame which contains the required statistics. The tables are then typically used as input for forest plots.

Value

An \texttt{rtables} table summarizing biomarker effects on binary response by subgroup.

Note

In contrast to \texttt{tabulate_rsp_subgroups()} this tabulation function does not start from an input layout \texttt{lyt}. This is because internally the table is created by combining multiple subtables.

See Also

\texttt{h_tab_rsp_one_biomarker()} which is used internally, \texttt{extract_rsp_biomarkers()}.\n
Examples

\begin{verbatim}
library(dplyr)
library(forcats)

adrs <- tern_ex_adrs
\end{verbatim}
sas_na <- formatters::var_labels(adrs)

adr_f <- adrs %>%
  filter(PARAMCD == "BESRSP1") %>%
  mutate(rsp = AVALC == "CR")
formatters::var_labels(adr_f) <- c(adr_labels, "Response")

df <- extract_rsp_biomarkers(
  variables = list(
    rsp = "rsp",
    biomarkers = c("BMRKR1", "AGE"),
    covariates = "SEX",
    subgroups = "BMRKR2"
  ),
  data = adrs_f
)

## Table with default columns.
tabulate_rsp_biomarkers(df)

## Table with a manually chosen set of columns: leave out "pval", reorder.
tab <- tabulate_rsp_biomarkers(
  df = df,
  vars = c("n_rsp", "ci", "n_tot", "prop", "or"
)

## Finally produce the forest plot.
g_forest(tab, xlim = c(0.7, 1.4))

### sas_na

**Convert Strings to NA**

**Description**

[Stable]

SAS imports missing data as empty strings or strings with whitespaces only. This helper function can be used to convert these values to NAs.

**Usage**

`sas_na(x, empty = TRUE, whitespaces = TRUE)`

**Arguments**

- `x` (factor or character vector)
  - values for which any missing values should be substituted.
empty     (logical)
if TRUE empty strings get replaced by NA.
whitespaces (logical)
if TRUE then strings made from whitespaces only get replaced with NA.

Value
x with "" and/or whitespace-only values substituted by NA, depending on the values of empty and whitespaces.

Examples
sas_na(c("1", "", "", "", "", "b"))
sas_na(factor(c("", "", "b")))
is.na(sas_na(c("1", "", "", "", "", "b")))

score_occurrences Occurrence Table Sorting

Description

[Stable]
Functions to score occurrence table subtables and rows which can be used in the sorting of occurrence tables.

Usage
score_occurrences(table_row)
score_occurrences_cols(...)
score_occurrences_subtable(...)
score_occurrences_cont_cols(...)

Arguments

table_row   (TableRow)
an analysis row in a occurrence table.
...
... arguments for row or column access, see rtables_access: either col_names (character) including the names of the columns which should be used, or alternatively col_indices (integer) giving the indices directly instead.
Value

- `score_occurrences()` returns the sum of counts across all columns of a table row.

- `score_occurrences_cols()` returns a function that sums counts across all specified columns of a table row.

- `score_occurrences_subtable()` returns a function that sums counts in each subtable across all specified columns.

- `score_occurrences_cont_cols()` returns a function that sums counts in the first content row in specified columns.

Functions

- `score_occurrences()`: Scoring function which sums the counts across all columns. It will fail if anything else but counts are used.

- `score_occurrences_cols()`: Scoring functions can be produced by this constructor to only include specific columns in the scoring. See `h_row_counts()` for further information.

- `score_occurrences_subtable()`: Scoring functions produced by this constructor can be used on subtables: They sum up all specified column counts in the subtable. This is useful when there is no available content row summing up these counts.

- `score_occurrences_cont_cols()`: Produce score function for sorting table by summing the first content row in specified columns. Note that this is extending `rtables::cont_n_onecol()` and `rtables::cont_n_allcols()`.

See Also

- `h_row_first_values()`
- `h_row_counts()`

Examples

```r
lyt <- basic_table() %>%
  split_cols_by("ARM") %>%
  add_colcounts() %>%
  analyze_num_patients(
    vars = "USUBJID",
    .stats = c("unique"),
    .labels = c("Total number of patients with at least one event")
  )

split_rows_by("AEBODSYS", child_labels = "visible", nested = FALSE) %>%
summarize_num_patients(
  var = "USUBJID",
  .stats = c("unique", "nonunique"),
  .labels = c(
    "Total number of patients with at least one event",
    "Total number of events"
  )
) %>%
```
count_occurrences(vars = "AEDECOD")

tbl <- build_table(lyt, tern_ex_adae, alt_counts_df = tern_ex_adsl) %>%
  prune_table()

tbl_sorted <- tbl %>%
  sort_at_path(path = c("AEBODSYS", "x", "AEDECOD"), scorefun = score_occurrences)

tbl_sorted

score_cols_a_and_b <- score_occurrences_cols(col_names = c("A: Drug X", "B: Placebo"))

# Note that this here just sorts the AEDECOD inside the AEBODSYS. The AEBODSYS are not sorted.
# That would require a second pass of `sort_at_path`

tbl_sorted <- tbl %>%
  sort_at_path(path = c("AEBODSYS", "x", "AEDECOD"), scorefun = score_cols_a_and_b)

tbl_sorted

score_subtable_all <- score_occurrences_subtable(col_names = names(tbl))

# Note that this code just sorts the AEBODSYS, not the AEDECOD within AEBODSYS. That
# would require a second pass of `sort_at_path`

tbl_sorted <- tbl %>%
  sort_at_path(path = c("AEBODSYS"), scorefun = score_subtable_all, decreasing = FALSE)

tbl_sorted

---

**split_cols_by_groups**  
*Split Columns by Groups of Levels*

**Description**

[Stable]

**Usage**

`split_cols_by_groups(lyt, var, groups_list = NULL, ref_group = NULL, ...)`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lyt</td>
<td>(layout) input layout where analyses will be added to.</td>
</tr>
<tr>
<td>var</td>
<td>(string) single variable name that is passed by rtables when requested by a statistics function.</td>
</tr>
<tr>
<td>groups_list</td>
<td>(named list of character) specifies the new group levels via the names and the levels that belong to it in the character vectors that are elements of the list.</td>
</tr>
</tbody>
</table>
ref_group (data.frame or vector)
the data corresponding to the reference group.

... additional arguments to rtables::split_cols_by() in order. For instance, to control formats (format), add a joint column for all groups (incl_all).

Value
A layout object suitable for passing to further layouting functions. Adding this function to an rtable layout will add a column split including the given groups to the table layout.

See Also
rtables::split_cols_by()

Examples

# 1 - Basic use

# Without group combination `split_cols_by_groups` is
# equivalent to [rtables::split_cols_by()].
basic_table() %>%
split_cols_by_groups("ARM") %>%
add_colcounts() %>%
analyze("AGE") %>%
build_table(DM)

# Add a reference column.

basic_table() %>%
split_cols_by_groups("ARM", ref_group = "B: Placebo") %>%
add_colcounts() %>%
analyze("AGE",
afun = function(x, .ref_group, .in_ref_col) {
  if (.in_ref_col) {
    in_rows("Diff Mean" = rcell(NULL))
  } else {
    in_rows("Diff Mean" = rcell(mean(x) - mean(.ref_group), format = "xx.xx"))
  }
}) %>%
build_table(DM)

# 2 - Adding group specification

# Manual preparation of the groups.

groups <- list(
  "Arms A+B" = c("A: Drug X", "B: Placebo"),
  "Arms A+C" = c("A: Drug X", "C: Combination")
)

# Use of split_cols_by_groups without reference column.

basic_table() %>%
split_cols_by_groups("ARM", groups) %>%
add_colcounts() %>%
analyze("AGE") %>%
basic_table(DM)

# Including differentiated output in the reference column.

basic_table() %>%
split_cols_by_groups("ARM", groups_list = groups, ref_group = "Arms A+B") %>%
analyze(
  "AGE",
  afun = function(x, .ref_group, .in_ref_col) {
    if (.in_ref_col) {
      in_rows("Diff. of Averages" = rcell(NULL))
    } else {
      in_rows("Diff. of Averages" = rcell(mean(x) - mean(.ref_group), format = "xx.xx"))
    }
  }
) %>%
basic_table(DM)

# 3 - Binary list dividing factor levels into reference and treatment

# `combine_groups` defines reference and treatment.

groups <- combine_groups(
  fct = DM$ARM,
  ref = c("A: Drug X", "B: Placebo")
)
groups

# Use group definition without reference column.

basic_table() %>%
split_cols_by_groups("ARM", groups_list = groups) %>%
add_colcounts() %>%
analyze("AGE") %>%
basic_table(DM)

# Use group definition with reference column (first item of groups).

basic_table() %>%
split_cols_by_groups("ARM", groups, ref_group = names(groups)[1]) %>%
add_colcounts() %>%
analyze(
  "AGE",
  afun = function(x, .ref_group, .in_ref_col) {
    if (.in_ref_col) {
      in_rows("Diff Mean" = rcell(NULL))
    } else {
      in_rows("Diff Mean" = rcell(mean(x) - mean(.ref_group), format = "xx.xx"))
    }
  }
) %>%
basic_table(DM)
**Description**

[Stable]  
Stack grobs as a new grob with 1 column and multiple rows layout.

**Usage**

```r
stack_grobs(
  ..., 
  grobs = list(...),
  padding = grid::unit(2, "line"),
  vp = NULL,
  gp = NULL,
  name = NULL
)
```

**Arguments**

- `...`  
  grobs.
- `grobs`  
  list of grobs.
- `padding`  
  unit of length 1, space between each grob.
- `vp`  
  a `viewport()` object (or NULL).
- `gp`  
  A `gpar()` object.
- `name`  
  a character identifier for the grob.

**Value**

A grob.

**Examples**

```r
library(grid)

g1 <- circleGrob(gp = gpar(col = "blue"))
g2 <- circleGrob(gp = gpar(col = "red"))
g3 <- textGrob("TEST TEXT")
g1 <- circleGrob(gp = gpar(col = "blue"))
g2 <- circleGrob(gp = gpar(col = "red"))
g3 <- textGrob("TEST TEXT")
g1 <- circleGrob(gp = gpar(col = "blue"))
g2 <- circleGrob(gp = gpar(col = "red"))
g3 <- textGrob("TEST TEXT")
g1 <- circleGrob(gp = gpar(col = "blue"))
g2 <- circleGrob(gp = gpar(col = "red"))
g3 <- textGrob("TEST TEXT")
g1 <- circleGrob(gp = gpar(col = "blue"))
g2 <- circleGrob(gp = gpar(col = "red"))
g3 <- textGrob("TEST TEXT")
```

```r
grid.newpage()
gird.draw(stack_grobs(g1, g2, g3))
```

```
showViewport()
grid.newpage()
pushViewport(viewport(layout = grid.layout(1, 2)))
```
stat_mean_ci <- viewport(layout.pos.row = 1, layout.pos.col = 2)
grid.draw(stack_grobs(g1, g2, g3, vp = vp1, name = "test"))

showViewport()
grid.ls(grobs = TRUE, viewports = TRUE, print = FALSE)

---

**stat_mean_ci**

Confidence Interval for Mean

**Description**

[Stable]

Convenient function for calculating the mean confidence interval. It calculates the arithmetic as well as the geometric mean. It can be used as a ggplot helper function for plotting.

**Usage**

```r
stat_mean_ci(
x,  
conf_level = 0.95,  
na.rm = TRUE,  
n_min = 2,  
geom_helper = TRUE,  
geom_mean = FALSE
)
```

**Arguments**

- **x** (numeric)
  - vector of numbers we want to analyze.
- **conf_level** (proportion)
  - confidence level of the interval.
- **na.rm** (flag)
  - whether NA values should be removed from x prior to analysis.
- **n_min** (number)
  - a minimum number of non-missing x to estimate the confidence interval for mean.
- **geom_helper** (logical)
  - TRUE when output should be aligned for the use with ggplot.
- **geom_mean** (logical)
  - TRUE when the geometric mean should be calculated.

**Value**

A named vector of values mean_ci_lwr and mean_ci_upr.
Examples

```r
stat_mean_ci(sample(10), gg_helper = FALSE)

p <- ggplot2::ggplot(mtcars, ggplot2::aes(x = cyl, y = mpg)) +
  ggplot2::geom_point()

p + ggplot2::stat_summary(
  fun.data = stat_mean_ci,
  geom = "errorbar"
)

p + ggplot2::stat_summary(
  fun.data = stat_mean_ci,
  fun.args = list(conf_level = 0.5),
  geom = "errorbar"
)

p + ggplot2::stat_summary(
  fun.data = stat_mean_ci,
  fun.args = list(conf_level = 0.5, geom_mean = TRUE),
  geom = "errorbar"
)
```

---

### stat_mean_pval

*p-Value of the Mean*

**Description**

[Stable]

Convenient function for calculating the two-sided p-value of the mean.

**Usage**

```r
stat_mean_pval(x, na.rm = TRUE, n_min = 2, test_mean = 0)
```

**Arguments**

- `x` *(numeric)*
  
  Vector of numbers we want to analyze.

- `na.rm` *(flag)*
  
  Whether NA values should be removed from `x` prior to analysis.

- `n_min` *(numeric)*
  
  A minimum number of non-missing `x` to estimate the p-value of the mean.

- `test_mean` *(numeric)*
  
  Mean value to test under the null hypothesis.
Value
A p-value.

Examples

```
stat_mean_pval(sample(10))
stat_mean_pval(rnorm(10), test_mean = 0.5)
```

---

`stat_median_ci`  Confidence Interval for Median

Description

[Stable]

Convenient function for calculating the median confidence interval. It can be used as a `ggplot` helper function for plotting.

Usage

```
stat_median_ci(x, conf_level = 0.95, na.rm = TRUE, gg_helper = TRUE)
```

Arguments

- `x`  (numeric)
  vector of numbers we want to analyze.
- `conf_level`  (proportion)
  confidence level of the interval.
- `na.rm`  (flag)
  whether NA values should be removed from `x` prior to analysis.
- `gg_helper`  (logical)
  TRUE when output should be aligned for the use with `ggplot`.

Details

The function was adapted from DescTools/versions/0.99.35/source

Value

A named vector of values `median_ci_lwr` and `median_ci_upr`. 
Examples

```r
stat_median_ci(sample(10), gg_helper = FALSE)
```

```r
p <- ggplot2::ggplot(mtcars, ggplot2::aes(cyl, mpg)) +
    ggplot2::geom_point()
p + ggplot2::stat_summary(fun.data = stat_median_ci,
    geom = "errorbar"
)
```

---

**strata_normal_quantile**

*Helper Function for the Estimation of Stratified Quantiles*

**Description**

[Stable]

This function wraps the estimation of stratified percentiles when we assume the approximation for large numbers. This is necessary only in the case proportions for each strata are unequal.

**Usage**

```r
strata_normal_quantile(vars, weights, conf_level)
```

**Arguments**

- **vars** *(character)*
  - variable names for the primary analysis variable to be iterated over.

- **weights** *(numeric or NULL)*
  - weights for each level of the strata. If NULL, they are estimated using the iterative algorithm proposed in Yan and Su (2010) that minimizes the weighted squared length of the confidence interval.

- **conf_level** *(proportion)*
  - confidence level of the interval.

**Value**

Stratified quantile.

**See Also**

`prop_strat_wilson()`
Examples

```
strata_data <- table(data.frame(
  "f1" = sample(c(TRUE, FALSE), 100, TRUE),
  "f2" = sample(c("x", "y", "z"), 100, TRUE),
  stringsAsFactors = TRUE
))
ns <- colSums(strata_data)
esths <- strata_data["TRUE",] / ns
vars <- esths * (1 - esths) / ns
weights <- rep(1 / length(ns), length(ns))
strata_normal_quantile(vars, weights, 0.95)
```

summarize_colvars

Summarize Variables in Columns

Description

[Stable]

This analyze function uses the S3 generic function `s_summary()` to summarize different variables that are arranged in columns. Additional standard formatting arguments are available. It is a minimal wrapper for `rtables::analyze_colvars()`. The latter function is meant to add different analysis methods for each column variables as different rows. To have the analysis methods as column labels, please refer to `analyze_vars_in_cols()`.

Usage

```
summarize_colvars(
  lyt, 
  ..., 
  .stats = c("n", "mean_sd", "median", "range", "count_fraction"), 
  .formats = NULL, 
  .labels = NULL, 
  .indent_mods = NULL
)
```

Arguments

- `lyt` (layout) input layout where analyses will be added to.
- `...` arguments passed to `s_summary()`.
- `.stats` (character) statistics to select for the table.
- `.formats` (named character or list) formats for the statistics.
summarize_colvars

.labels (named character)
  labels for the statistics (without indent).

.indent_mods (named vector of integer)
  indent modifiers for the labels. Each element of the vector should be a
  name-value pair with name corresponding to a statistic specified in .stats
  and value the indentation for that statistic’s row label.

Value

A layout object suitable for passing to further layouting functions, or to rtables::build_table().
Adding this function to an rtable layout will summarize the given variables, arrange the output in
columns, and add it to the table layout.

See Also

rtables::split_cols_by_multivar() and analyze_colvars_functions.

Examples

dta_test <- data.frame(
  USUBJID = rep(1:6, each = 3),
  PARAMCD = rep("lab", 6 * 3),
  AVISIT = rep(paste0("V", 1:3), 6),
  ARM = rep(LETTERS[1:3], rep(6, 3)),
  AVAL = c(9:1, rep(NA, 9)),
  CHG = c(1:9, rep(NA, 9))
)

## Default output within a `rtables` pipeline.
basic_table() %>%
  split_cols_by("ARM") %>%
  split_rows_by("AVISIT") %>%
  split_cols_by_multivar(vars = c("AVAL", "CHG")) %>%
  summarize_colvars() %>%
  build_table(dta_test)

## Selection of statistics, formats and labels also work.
basic_table() %>%
  split_cols_by("ARM") %>%
  split_rows_by("AVISIT") %>%
  split_cols_by_multivar(vars = c("AVAL", "CHG")) %>%
  summarize_colvars(
    .stats = c("n", "mean_sd"),
    .formats = c("mean_sd" = "xx.x, xx.x"),
    .labels = c(n = "n", mean_sd = "Mean, SD")
  ) %>%
  build_table(dta_test)

## Use arguments interpreted by `s_summary`
basic_table() %>%
  split_cols_by("ARM") %>%
  split_rows_by("AVISIT") %>%
```r
split_cols_by_multivar(vars = c("AVAL", "CHG")) %>%
summarize_colvars(na.rm = FALSE) %>%
build_table(dta_test)
```

**summarize_functions**  
**Summarize Functions**

**Description**

These functions are wrappers for `rtables::summarize_row_groups()`, applying corresponding content functions to add summary rows to a given table layout:

- `add_rowcounts()`
- `estimate_multinomial_response()` (with `rtables::analyze()`)  
- `h_tab_one_biomarker()` (probably to deprecate)  
- `logistic_summary_by_flag()`  
- `summarize_num_patients()`  
- `summarize_occurrences_by_grade()`  
- `summarize_patients_events_in_cols()`  
- `summarize_patients_exposure_in_cols()`  
- `tabulate_rsp_subgroups()`

Additionally, the `summarize_coxreg()` function utilizes `rtables::summarize_row_groups()` (in combination with several other `rtables` functions like `rtables::analyze_colvars()`) to output a Cox regression summary table.

**See Also**

- `analyze_functions` for functions which are wrappers for `rtables::analyze()`.
- `analyze_colvars_functions` for functions that are wrappers for `rtables::analyze_colvars()`.

**summarize_logistic**  
**Multivariate Logistic Regression Table**

**Description**

[Stable]  
Layout-creating function which summarizes a logistic variable regression for binary outcome with categorical/continuous covariates in model statement. For each covariate category (if categorical) or specified values (if continuous), present degrees of freedom, regression parameter estimate and standard error (SE) relative to reference group or category. Report odds ratios for each covariate category or specified values and corresponding Wald confidence intervals as default but allow user to specify other confidence levels. Report p-value for Wald chi-square test of the null hypothesis that covariate has no effect on response in model containing all specified covariates. Allow option to include one two-way interaction and present similar output for each interaction degree of freedom.
summarize_logistic

Usage

summarize_logistic(
  lyt,  
  conf_level,  
  drop_and_remove_str = "",  
  .indent_mods = NULL
)

Arguments

lyt (layout)  
input layout where analyses will be added to.

conf_level (proportion)  
confidence level of the interval.

drop_and_remove_str  
(character)  
string to be dropped and removed.

.indent_mods (named integer)  
indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.

Value

A layout object suitable for passing to further layouting functions, or to rtables::build_table(). Adding this function to an rtable layout will add a logistic regression variable summary to the table layout.

Note

For the formula, the variable names need to be standard data.frame column names without special characters.

Examples

library(dplyr)
library(broom)

adrs_f <- tern_ex_adrs %>%
  filter(PARAMCD == "BESRSPI") %>%
  filter(RACE %in% c("ASIAN", "WHITE", "BLACK OR AFRICAN AMERICAN")) %>%
  mutate(
    Response = case_when(AVALC %in% c("PR", "CR") ~ 1, TRUE ~ 0),
    RACE = factor(RACE),
    SEX = factor(SEX)
  )
formatters::var_labels(adrs_f) <- c(formatters::var_labels(tern_ex_adrs), Response = "Response")
mod1 <- fit_logistic(
  data = adrs_f,
  variables = list(
    Response,
    SEX = factor(SUBSET = TRUE, label = "SEX"),
    RACE = factor(SUBSET = TRUE, label = "RACE")
  ),
  conf_level = 0.95,
  drop_and_remove_str = "intercept"
)
```r
mod2 <- fit_logistic(
  data = adrs_f,
  variables = list(
    response = "Response",
    arm = "ARMCD",
    covariates = c("AGE", "RACE"),
    interaction = "AGE"
  )
)

df <- tidy(mod1, conf_level = 0.99)
df2 <- tidy(mod2, conf_level = 0.99)

# flagging empty strings with "_"
df <- df_explicit_na(df, na_level = "_")
df2 <- df_explicit_na(df2, na_level = "_")

result1 <- basic_table() %>%
  summarize_logistic(
    conf_level = 0.95,
    drop_and_remove_str = "_"
  ) %>%
  build_table(df = df)
result1

result2 <- basic_table() %>%
  summarize_logistic(
    conf_level = 0.95,
    drop_and_remove_str = "_"
  ) %>%
  build_table(df = df2)
result2
```

---

**summarize_num_patients**

*Number of Patients*

**Description**

**[Stable]**

Count the number of unique and non-unique patients in a column (variable).
**Usage**

```r
s_num_patients(
  x,
  labelstr,
  .N_col,
  count_by = NULL,
  unique_count_suffix = TRUE
)
```

```r
s_num_patients_content(
  df,
  labelstr = "",
  .N_col,
  .var,
  required = NULL,
  count_by = NULL,
  unique_count_suffix = TRUE
)
```

```r
summarize_num_patients(
  lyt,
  var,
  .stats = NULL,
  .formats = NULL,
  .labels = c(unique = "Number of patients with at least one event", nonunique = "Number of events"),
  indent_mod = lifecycle::deprecated(),
  .indent.mods = 0L,
  ...
)
```

```r
analyze_num_patients(
  lyt,
  vars,
  .stats = NULL,
  .formats = NULL,
  .labels = c(unique = "Number of patients with at least one event", nonunique = "Number of events"),
  show_labels = c("default", "visible", "hidden"),
  indent_mod = lifecycle::deprecated(),
  .indent.mods = 0L,
  ...
)
```

**Arguments**

- **x** (character or factor) vector of patient IDs.
labelstr
(label)
label of the level of the parent split currently being summarized (must be present as second argument in Content Row Functions). See \texttt{rtables::summarize_row_groups()} for more information.

.N_col
(count)
row-wise N (row group count) for the group of observations being analyzed (i.e. with no column-based subsetting) that is passed by \texttt{rtables}.

count_by
(character or factor)
optional vector to be combined with \texttt{x} when counting nonunique records.

unique_count_suffix
(logical)
should "\(n\)" suffix be added to unique_count labels. Defaults to TRUE.

df
(data.frame)
data set containing all analysis variables.

.var, var
(string)
single variable name that is passed by \texttt{rtables} when requested by a statistics function.

required
(character or NULL)
optional name of a variable that is required to be non-missing.

lyt
(layout)
input layout where analyses will be added to.

.stats
(character)
statistics to select for the table.

.formats
(named character or list)
formats for the statistics.

.labels
(named character)
labels for the statistics (without indent).

indent_mod
[Deprecated] Please use the .indent_mods argument instead.

.indent_mods
(named integer)
indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.

... additional arguments for the lower level functions.

vars
(character)
variable names for the primary analysis variable to be iterated over.

show_labels
(string)
label visibility: one of "default", "visible" and "hidden".

Details
In general, functions that starts with analyze* are expected to work like \texttt{rtables::analyze()}, while functions that starts with summarize* are based upon \texttt{rtables::summarize_row_groups()}. The latter provides a value for each dividing split in the row and column space, but, being it bound to the fundamental splits, it is repeated by design in every page when pagination is involved.
Value

- s_num_patients() returns a named list of 3 statistics:
  - unique: Vector of counts and percentages.
  - nonunique: Vector of counts.
  - unique_count: Counts.

- s_num_patients_content() returns the same values as s_num_patients().

- summarize_num_patients() returns a layout object suitable for passing to further layouting functions, or to rtables::build_table(). Adding this function to an rtable layout will add formatted rows containing the statistics from s_num_patients_content() to the table layout.

- analyze_num_patients() returns a layout object suitable for passing to further layouting functions, or to rtables::build_table(). Adding this function to an rtable layout will add formatted rows containing the statistics from s_num_patients_content() to the table layout.

Functions

- s_num_patients(): Statistics function which counts the number of unique patients, the corresponding percentage taken with respect to the total number of patients, and the number of non-unique patients.
- s_num_patients_content(): Statistics function which counts the number of unique patients in a column (variable), the corresponding percentage taken with respect to the total number of patients, and the number of non-unique patients in the column.
- summarize_num_patients(): Layout-creating function which can take statistics function arguments and additional format arguments. This function is a wrapper for rtables::summarize_row_groups().
- analyze_num_patients(): Layout-creating function which can take statistics function arguments and additional format arguments. This function is a wrapper for rtables::analyze().

Note

As opposed to summarize_num_patients(), this function does not repeat the produced rows.

Examples

# Use the statistics function to count number of unique and nonunique patients.
s_num_patients(x = as.character(c(1, 1, 1, 2, 4, NA)), labelstr = "", .N_col = 6L)
s_num_patients(
  x = as.character(c(1, 1, 1, 2, 4, NA)),
  labelstr = "",
  .N_col = 6L,
  count_by = as.character(c(1, 1, 2, 1, 1, 1))
)

# Count number of unique and non-unique patients.
df <- data.frame(
  USUBJID = as.character(c(1, 2, 1, 4, NA)),
  ...
EVENT = as.character(c(10, 15, 10, 17, 8))
)
s_num_patients_content(df, .N_col = 5, .var = "USUBJID")

df_by_event <- data.frame(
    USUBJID = as.character(c(1, 2, 1, 4, NA)),
    EVENT = as.character(c(10, 15, 10, 17, 8))
)
s_num_patients_content(df_by_event, .N_col = 5, .var = "USUBJID")
s_num_patients_content(df_by_event, .N_col = 5, .var = "USUBJID", count_by = "EVENT")

df_tmp <- data.frame(
    USUBJID = as.character(c(1, 2, 1, 4, NA, 6, 6, 8, 9)),
    AGE = c(10, 15, 10, 17, 8, 11, 11, 19, 17)
)
tbl <- basic_table() %>%
    split_cols_by("ARM") %>%
    add_colcounts() %>%
    analyze_num_patients("USUBJID", .stats = c("unique")) %>%
    build_table(df_tmp)
tbl

---

**summarize_variables**

*Summarize Variables*

**Description**

[Stable]

We use the S3 generic function `s_summary()` to implement summaries for different x objects. This is used as a statistics function in combination with the `analyze` function `summarize_vars()`.

**Usage**

```r
s_summary(x, na.rm = TRUE, denom, .N_row, .N_col, .var, ...)
```

```r
## S3 method for class 'numeric'
s_summary(
    x,
    na.rm = TRUE,
    denom,
    .N_row,
    .N_col,
    .var,
    control = control_summarize_vars(),
    ...
)
```
## S3 method for class 'factor'

```r
s_summary(
  x,
  na.rm = TRUE,
  denom = c("n", "N_row", "N_col"),
  .N_row,
  .N_col,
  ...
)
```

## S3 method for class 'character'

```r
s_summary(
  x,
  na.rm = TRUE,
  denom = c("n", "N_row", "N_col"),
  .N_row,
  .N_col,
  .var,
  verbose = TRUE,
  ...
)
```

## S3 method for class 'logical'

```r
s_summary(
  x,
  na.rm = TRUE,
  denom = c("n", "N_row", "N_col"),
  .N_row,
  .N_col,
  ...
)
```

```r
a_summary(x, ..., .N_row, .N_col, .var)
```

## S3 method for class 'numeric'

```r
a_summary(
  x,
  na.rm = TRUE,
  denom,
  .N_row,
  .N_col,
  .var,
  control = control_summarize_vars(),
  ...
)
```

## S3 method for class 'factor'

```r
a_summary(
```
summarize_variables

x,
na.rm = TRUE,
denom = c("n", "N_row", "N_col"),
.N_row,
.N_col,
...
)

## S3 method for class 'character'
a_summary(
  x,
  na.rm = TRUE,
  denom = c("n", "N_row", "N_col"),
  .N_row,
  .N_col,
  .var,
  verbose = TRUE,
  ...
)

## S3 method for class 'logical'
a_summary(
  x,
  na.rm = TRUE,
  denom = c("n", "N_row", "N_col"),
  .N_row,
  .N_col,
  ...
)

summarize_vars(
  lyt,
  vars,
  var_labels = vars,
  nested = TRUE,
  ...
  na_level = NA_character_,
  show_labels = "default",
  table_names = vars,
  section_div = NA_character_,
  .stats = c("n", "mean_sd", "median", "range", "count_fraction"),
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
Arguments

x (numeric)
vector of numbers we want to analyze.

na.rm (flag)
whether NA values should be removed from x prior to analysis.

denom (string)
choice of denominator for proportion. Options are:
- \( n \): number of values in this row and column intersection.
- \( N_{\text{row}} \): total number of values in this row across columns.
- \( N_{\text{col}} \): total number of values in this column across rows.

.N_row (count)
column-wise N (column count) for the full column that is passed by rtables.

.N_col (count)
row-wise N (row group count) for the group of observations being analyzed (i.e.
with no column-based subsetting) that is passed by rtables.

.var (string)
single variable name that is passed by rtables when requested by a statistics
function.

... arguments passed to s_summary().

control (list)
parameters for descriptive statistics details, specified by using the helper function
control_summarize_vars(). Some possible parameter options are:
- conf_level (proportion)
  confidence level of the interval for mean and median.
- quantiles (numeric)
  vector of length two to specify the quantiles.
- quantile_type (numeric)
  between 1 and 9 selecting quantile algorithms to be used. See more about
type in stats::quantile().
- test_mean (numeric)
  value to test against the mean under the null hypothesis when calculating
p-value.

verbose (logical)
Defaults to TRUE, which prints out warnings and messages. It is mainly used to
print out information about factor casting.

lyt (layout)
input layout where analyses will be added to.

vars (character)
variable names for the primary analysis variable to be iterated over.

var_labels (character)
character for label.
nested (flag) whether this layout instruction be applied within the existing layout structure if possible (TRUE, the default) or as a new top-level element (FALSE). Ignored if it would nest a split underneath analyses, which is not allowed.

na_level (string) string used to replace all NA or empty values in the output.

show_labels (string) label visibility: one of "default", "visible" and "hidden".

table_names (character) this can be customized in case that the same vars are analyzed multiple times, to avoid warnings from rtables.

section_div (string) string which should be repeated as a section divider after each group defined by this split instruction, or NA_character_ (the default) for no section divider.

.stats (character) statistics to select for the table.

.formats (named character or list) formats for the statistics.

.labels (named character) labels for the statistics (without indent).

.indent_mods (named vector of integer) indent modifiers for the labels. Each element of the vector should be a name-value pair with name corresponding to a statistic specified in .stats and value the indentation for that statistic’s row label.

Value

- s_summary() returns different statistics depending on the class of x.

- If x is of class numeric, returns a list with the following named numeric items:
  - n: The length() of x.
  - sum: The sum() of x.
  - mean: The mean() of x.
  - sd: The stats::sd() of x.
  - se: The standard error of x mean, i.e.: (sd(x) / sqrt(length(x))).
  - mean_sd: The mean() and stats::sd() of x.
  - mean_se: The mean() of x and its standard error (see above).
  - mean_ci: The CI for the mean of x (from stat_mean_ci()).
  - mean_sei: The SE interval for the mean of x, i.e.: (mean() +/- stats::sd() / sqrt()).
  - mean_sdi: The SD interval for the mean of x, i.e.: (mean() +/- stats::sd()).
  - mean_pval: The two-sided p-value of the mean of x (from stat_mean_pval()).
  - median: The stats::median() of x.
  - mad: The median absolute deviation of x, i.e.: (stats::median() of xc, where xc = x - stats::median()).
- median_ci: The CI for the median of x (from `stat_median_ci()`).
- quantiles: Two sample quantiles of x (from `stats::quantile()`).
- iqr: The `stats::IQR()` of x.
- range: The `range_noinf()` of x.
- min: The `min()` of x.
- max: The `max()` of x.
- median_range: The `median()` and `range_noinf()` of x.
- cv: The coefficient of variation of x, i.e.: `(stats::sd() / mean() * 100)`.
- geom_mean: The geometric mean of x, i.e.: `(exp(mean(log(x)))`.
- geom_cv: The geometric coefficient of variation of x, i.e.: `(sqrt(exp(sd(log(x)) ^ 2) - 1) * 100)`.

- If x is of class factor or converted from character, returns a list with named numeric items:
  - n: The `length()` of x.
  - count: A list with the number of cases for each level of the factor x.
  - count_fraction: Similar to count but also includes the proportion of cases for each level of the factor x relative to the denominator, or NA if the denominator is zero.

- If x is of class logical, returns a list with named numeric items:
  - n: The `length()` of x (possibly after removing NAs).
  - count: Count of TRUE in x.
  - count_fraction: Count and proportion of TRUE in x relative to the denominator, or NA if the denominator is zero. Note that NAs in x are never counted or leading to NA here.

- `a_summary()` returns the corresponding list with formatted `rtables::CellValue()`.

- `summarize_vars()` returns a layout object suitable for passing to further layouting functions, or to `rtables::build_table()`. Adding this function to an rtable layout will add formatted rows containing the statistics from `s_summary()` to the table layout.

Functions

- `s_summary()`: S3 generic function to produces a variable summary.
- `s_summary(numeric)`: Method for numeric class.
- `s_summary(factor)`: Method for factor class.
- `s_summary(character)`: Method for character class. This makes an automatic conversion to factor (with a warning) and then forwards to the method for factors.
- `s_summary(logical)`: Method for logical class.
- `a_summary()`: Formatted analysis function which is used as `afun` in `summarize_vars()`.
- `a_summary(numeric)`: Formatted analysis function method for numeric class.
- `a_summary(factor)`: Formatted analysis function method for factor class.
- `a_summary(character)`: Formatted analysis function method for character class.
- `a_summary(logical)`: Formatted analysis function method for logical class.
- `summarize_vars()`: Layout-creating function which can take statistics function arguments and additional format arguments. This function is a wrapper for `rtables::analyze()`.
summarize_variables

Note

- If \( x \) is an empty vector, \( \text{NA} \) is returned. This is the expected feature so as to return `cell` content in `rtables` when the intersection of a column and a row delimits an empty data selection.
- When the `mean` function is applied to an empty vector, \( \text{NA} \) will be returned instead of `NaN`, the latter being standard behavior in R.
- If \( x \) is an empty factor, a list is still returned for counts with one element per factor level. If there are no levels in \( x \), the function fails.
- If factor variables contain \( \text{NA} \), these \( \text{NA} \) values are excluded by default. To include \( \text{NA} \) values set \( \text{na.rm = FALSE} \) and missing values will be displayed as an \( \text{NA} \) level. Alternatively, an explicit factor level can be defined for \( \text{NA} \) values during pre-processing via `df_explicit_na()` - the default `na_level` ("<Missing>") will also be excluded when `na.rm` is set to `TRUE`.
- Automatic conversion of character to factor does not guarantee that the table can be generated correctly. In particular for sparse tables this very likely can fail. It is therefore better to always pre-process the dataset such that factors are manually created from character variables before passing the dataset to `rtables::build_table()`.

Examples

```r
# `s_summary.numeric`

## Basic usage: empty numeric returns NA-filled items.
s_summary(numeric())

## Management of NA values.
x <- c(NA_real_, 1)
s_summary(x, na.rm = TRUE)
s_summary(x, na.rm = FALSE)

x <- c(NA_real_, 1, 2)
s_summary(x, stats = NULL)

## Benefits in `rtables` contructions:
require(rtables)
dta_test <- data.frame(
  Group = rep(LETTERS[1:3], each = 2),
  sub_group = rep(letters[1:2], each = 3),
  x = 1:6
)

## The summary obtained in with `rtables`:
basic_table() %>%
  split_cols_by(var = "Group") %>%
  split_rows_by(var = "sub_group") %>%
  analyze(vars = "x", afun = s_summary) %>%
  build_table(df = dta_test)

## By comparison with `lapply`:
X <- split(dta_test, f = with(dta_test, interaction(Group, sub_group)))
lapply(X, function(x) s_summary(x$x))
```
# `s_summary.factor`

## Basic usage:
`s_summary(factor(c("a", "a", "b", "c", "a")))`

## Management of NA values.
`x <- factor(c(NA, "Female"))`
`x <- explicit_na(x)`
`s_summary(x, na.rm = TRUE)`
`s_summary(x, na.rm = FALSE)`

## Different denominators.
`x <- factor(c("a", "a", "b", "c", "a"))`
`s_summary(x, denom = "N_row", .N_row = 10L)`
`s_summary(x, denom = "N_col", .N_col = 20L)`

# `s_summary.character`

## Basic usage:
`s_summary(c("a", "a", "b", "c", "a"), .var = "x", verbose = FALSE)`

## Management of NA values.
`x <- c(NA, TRUE, FALSE, TRUE)`
`s_summary(x, na.rm = TRUE)`
`s_summary(x, na.rm = FALSE)`

## Different denominators.
`x <- c(TRUE, FALSE, TRUE, TRUE)`
`s_summary(x, denom = "N_row", .N_row = 10L)`
`s_summary(x, denom = "N_col", .N_col = 20L)`

# `s_summary.logical`

## Basic usage:
`s_summary(c(TRUE, FALSE, TRUE, TRUE))`

# `a_summary.numeric`

`a_summary(rnorm(10), .N_col = 10, .N_row = 20, .var = "bla")`

# `a_summary.factor`

# We need to ungroup `count` and `count_fraction` first so that the rtables formatting functions can be applied correctly.
`afun <- make_afun(`
`   getS3method("a_summary", "factor"),`
`   .ungroup_stats = c("count", "count_fraction")`)
`afun(factor(c("a", "a", "b", "c", "a")), .N_row = 10, .N_col = 10)`

# `a_summary.character`
afun <- make_afun(
  getS3method("a_summary", "character"),
  .ungroup_stats = c("count", "count_fraction")
)

# 'a_summary.logical'
afun <- make_afun(
  getS3method("a_summary", "logical")
)
afun(c(TRUE, FALSE, FALSE, TRUE, TRUE), .N_row = 10, .N_col = 10)

## Fabricated dataset.
dta_test <- data.frame(
  USUBJID = rep(1:6, each = 3),
  PARAMCD = rep("lab", 6 * 3),
  AVISIT = rep(paste0("V", 1:3), 6),
  ARM = rep(LETTERS[1:3], rep(6, 3)),
  AVAL = c(9:1, rep(NA, 9))
)

## Default output within a `rtables` pipeline.
l <- basic_table() %>%
  split_cols_by(var = "ARM") %>%
  split_rows_by(var = "AVISIT") %>%
  summarize_vars(vars = "AVAL")
build_table(l, df = dta_test)

## Select and format statistics output.
l <- basic_table() %>%
  split_cols_by(var = "ARM") %>%
  split_rows_by(var = "AVISIT") %>%
  summarize_vars(
    vars = "AVAL",
    .stats = c("n", "mean_sd", "quantiles"),
    .formats = c("mean_sd" = "xx.x, xx.x"),
    .labels = c(n = "n", mean_sd = "Mean, SD", quantiles = c("Q1 - Q3"))
  )
results <- build_table(l, df = dta_test)
as_html(results)

## Use arguments interpreted by `s_summary`.
l <- basic_table() %>%
  split_cols_by(var = "ARM") %>%
  split_rows_by(var = "AVISIT") %>%
  summarize_vars(vars = "AVAL", na.rm = FALSE)
results <- build_table(l, df = dta_test)

## Handle 'NA' levels first when summarizing factors.
```r
dta_test$AVISIT <- NA_character_
dta_test <- df_explicit_na(dta_test)
l <- basic_table() %>%
  split_cols_by(var = "ARM") %>%
  summarize_vars(vars = "AVISIT", na.rm = FALSE)
results <- build_table(l, df = dta_test)
Viewer(results)
```

---

**survival_biomarkers_subgroups**

*Tabulate Biomarker Effects on Survival by Subgroup*

---

**Description**

[Stable]

Tabulate the estimated effects of multiple continuous biomarker variables across population subgroups.

**Usage**

```r
tabulate_survival_biomarkers(
  df,
  vars = c("n_tot", "n_tot_events", "median", "hr", "ci", "pval"),
  time_unit = NULL,
  .indent_mods = 0L
)
```

**Arguments**

- **df** (data.frame) containing all analysis variables, as returned by `extract_survival_biomarkers()`.
- **vars** (character) the names of statistics to be reported among:
  - `n_tot_events`: Total number of events per group.
  - `n_tot`: Total number of observations per group.
  - `median`: Median survival time.
  - `hr`: Hazard ratio.
  - `ci`: Confidence interval of hazard ratio.
  - `pval`: p-value of the effect. Note, one of the statistics `n_tot` and `n_tot_events`, as well as both `hr` and `ci` are required.
- **time_unit** (string) label with unit of median survival time. Default NULL skips displaying unit.
indent_mods (named integer)
indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.

Details
These functions create a layout starting from a data frame which contains the required statistics. The tables are then typically used as input for forest plots.

Value
An rtables table summarizing biomarker effects on survival by subgroup.

Functions
- tabulate_survival_biomarkers(): Table-creating function which creates a table summarizing biomarker effects on survival by subgroup.

Note
In contrast to tabulate_survival_subgroups() this tabulation function does not start from an input layout lyt. This is because internally the table is created by combining multiple subtables.

See Also
h_tab_surv_one_biomarker() which is used internally, extract_survival_biomarkers().

Examples
library(dplyr)
adtte <- tern_ex_adtte
# Save variable labels before data processing steps.
adtte_labels <- formatters::var_labels(adtte)

adtte_f <- adtte %>%
  filter(PARAMCD == "OS") %>%
  mutate(
    AVALU = as.character(AVALU),
    is_event = CNSR == 0
  )
labels <- c("AVALU" = adtte_labels[["AVALU"]], "is_event" = "Event Flag")
formatters::var_labels(adtte_f)[names(labels)] <- labels

df <- extract_survival_biomarkers(
  variables = list( 
    tte = "AVAL", 
    is_event = "is_event", 
    biomarkers = c("BMRKR1", "AGE"), 
    strata = "STRATA1", 
    covariates = "SEX", 
  )
)
tidy.glm

### Table with default columns.

```r
tabulate_survival_biomarkers(df)
```

### Table with a manually chosen set of columns: leave out "pval", reorder.

```r
tab <- tabulate_survival_biomarkers(
  df = df,
  vars = c("n_tot_events", "ci", "n_tot", "median", "hr"),
  time_unit = as.character(adtte_f$AVLU[1])
)
```

### Finally produce the forest plot.

```r
g_forest(tab, xlim = c(0.8, 1.2))
```

---

**tidy.glm**  
*Custom Tidy Method for Binomial GLM Results*

---

**Description**

[Stable]

Helper method (for `broom::tidy()`) to prepare a data frame from a `glm` object with binomial family.

**Usage**

```r
## S3 method for class 'glm'
tidy(x, conf_level = 0.95, at = NULL, ...)
```

**Arguments**

- `x`  
  logistic regression model fitted by `stats::glm()` with "binomial" family.

- `conf_level`  
  (proportion)  
  confidence level of the interval.

- `at`  
  (NULL or numeric)  
  optional values for the interaction variable. Otherwise the median is used.

- `...`  
  additional arguments for the lower level functions.

**Value**

A data.frame containing the tidied model.
See Also

h_logistic_regression for relevant helper functions.

Examples

library(dplyr)
library(broom)

adrs_f <- tern_ex_adrs %>%
  filter(PARAMCD == "BESRSPI") %>%
  filter(RACE %in% c("ASIAN", "WHITE", "BLACK OR AFRICAN AMERICAN")) %>%
  mutate(
    Response = case_when(AVALC %in% c("PR", "CR") ~ 1, TRUE ~ 0),
    RACE = factor(RACE),
    SEX = factor(SEX)
  )
formatters::var_labels(adrs_f) <- c(formatters::var_labels(tern_ex_adrs), Response = "Response")
mod1 <- fit_logistic(
  data = adrs_f,
  variables = list(
    response = "Response",
    arm = "ARMCD",
    covariates = c("AGE", "RACE")
  )
)
mod2 <- fit_logistic(
  data = adrs_f,
  variables = list(
    response = "Response",
    arm = "ARMCD",
    covariates = c("AGE", "RACE"),
    interaction = "AGE"
  )
)

df <- tidy(mod1, conf_level = 0.99)
df2 <- tidy(mod2, conf_level = 0.99)

---

**tidy.step**

**Custom Tidy Method for STEP Results**

**Description**

[Stable]

Tidy the STEP results into a tibble format ready for plotting.
Usage

```r
## S3 method for class 'step'
tidy(x, ...)
```

Arguments

- `x` (step matrix): results from `fit_survival_step()`.
- `...` not used here.

Value

A tibble with one row per STEP subgroup. The estimates and CIs are on the HR or OR scale, respectively. Additional attributes carry metadata also used for plotting.

See Also

g_step() which consumes the result from this function.

Examples

```r
library(survival)
lung$sex <- factor(lung$sex)
vars <- list(
  time = "time",
  event = "status",
  arm = "sex",
  biomarker = "age"
)
step_matrix <- fit_survival_step(
  variables = vars,
  data = lung,
  control = c(control_coxph(), control_step(num_points = 10, degree = 2))
)
broom::tidy(step_matrix)
```
Usage

```r
## S3 method for class 'summary.coxph'
tidy(x, ...)

## S3 method for class 'coxreg.univar'
tidy(x, ...)

## S3 method for class 'coxreg.multivar'
tidy(x, ...)
```

Arguments

- `x` 
  (list)
  Result of the Cox regression model fitted by `fit_coxreg_univar()` (for univariate models) or `fit_coxreg_multivar()` (for multivariate models).

- `...` 
  additional arguments for the lower level functions.

Value

tidy() returns:

- For `summary.coxph` objects, a `data.frame` with columns: `Pr(>|z|)`, `exp(coef)`, `exp(-coef)`, `lower .95`, `upper .95`, `level`, and `n`.
- For `coxreg.univar` objects, a `data.frame` with columns: `effect`, `term`, `term_label`, `level`, `n`, `hr`, `lcl`, `ucl`, `pval`, and `ci`.
- For `coxreg.multivar` objects, a `data.frame` with columns: `term`, `pval`, `term_label`, `hr`, `lcl`, `ucl`, `level`, and `ci`.

Functions

- tidy(summary.coxph): Custom tidy method for `survival::coxph()` summary results. Tidy the `survival::coxph()` results into a `data.frame` to extract model results.
- tidy(coxreg.univar): Custom tidy method for a univariate Cox regression. Tidy up the result of a Cox regression model fitted by `fit_coxreg_univar()`.
- tidy(coxreg.multivar): Custom tidy method for a multivariate Cox regression. Tidy up the result of a Cox regression model fitted by `fit_coxreg_multivar()`.

See Also

cox_regression

Examples

```r
library(survival)
library(broom)
set.seed(1, kind = "Mersenne-Twister")
```
tidy_coxreg

```r
dta_bladder <- with(
  data = bladder[bladder$enum < 5, ],
  data.frame(
    time = stop,
    status = event,
    armcd = as.factor(rx),
    covar1 = as.factor(enum),
    covar2 = factor(
      sample(as.factor(enum)),
      levels = 1:4, labels = c("F", "F", "M", "M")
    )
  )
)
labels <- c("armcd" = "ARM", "covar1" = "A Covariate Label", "covar2" = "Sex (F/M)"
)
formatatters::var_labels(dta_bladder)[names(labels)] <- labels
dta_bladder$age <- sample(20:60, size = nrow(dta_bladder), replace = TRUE)
formula <- "survival::Surv(time, status) ~ armcd + covar1"
msum <- summary(coxph(stats::as.formula(formula), data = dta_bladder))
tidy(msum)

## Cox regression: arm + 1 covariate.
mod1 <- fit_coxreg_univar(
  variables = list(
    time = "time", event = "status", arm = "armcd",
    covariates = "covar1"
  ),
  data = dta_bladder,
  control = control_coxreg(conf_level = 0.91)
)

## Cox regression: arm + 1 covariate + interaction, 2 candidate covariates.
mod2 <- fit_coxreg_univar(
  variables = list(
    time = "time", event = "status", arm = "armcd",
    covariates = c("covar1", "covar2")
  ),
  data = dta_bladder,
  control = control_coxreg(conf_level = 0.91, interaction = TRUE)
)

tidy(mod1)
tidy(mod2)

multivar_model <- fit_coxreg_multivar(
  variables = list(
    time = "time", event = "status", arm = "armcd",
    covariates = c("covar1", "covar2")
  ),
  data = dta_bladder
)
broom::tidy(multivar_model)
```
to_n

Replicate Entries of a Vector if Required

Description

[Stable]
Replicate entries of a vector if required.

Usage

to_n(x, n)

Arguments

x (numeric)
vector of numbers we want to analyze.

n (count)
how many entries we need.

Value

x if it has the required length already or is NULL, otherwise if it is scalar the replicated version of it with n entries.

Note

This function will fail if x is not of length n and/or is not a scalar.

to_string_matrix

Convert Table into Matrix of Strings

Description

[Stable]
Helper function to use mostly within tests. with_spaces parameter allows to test not only for content but also indentation and table structure. print_txt_to_copy instead facilitate the testing development by returning a well formatted text that needs only to be copied and pasted in the expected output.

Usage

to_string_matrix(x, with_spaces = FALSE, print_txt_to_copy = FALSE)
Arguments

x \quad \text{rtables table.}

with_spaces \quad \text{Should the tested table keep the indentation and other relevant spaces?}

print_txt_to_copy \quad \text{Utility to have a way to copy the input table directly into the expected variable instead of copying it too manually.}

Value

A matrix of strings.

\begin{center}
\begin{tabular}{ll}
univariate & \textit{Univariate Formula Special Term} \\
\end{tabular}
\end{center}

Description

[Stable]

The special term univariate indicate that the model should be fitted individually for every variable included in univariate.

Usage

univariate(x)

Arguments

x \quad A vector of variable name separated by commas.

Details

If provided alongside with pairwise specification, the model \( y \sim \text{ARM} + \text{univariate(SEX, AGE, RACE)} \) lead to the study and comparison of the models

- \( y \sim \text{ARM} \)
- \( y \sim \text{ARM} + \text{SEX} \)
- \( y \sim \text{ARM} + \text{AGE} \)
- \( y \sim \text{ARM} + \text{RACE} \)

Value

When used within a model formula, produces univariate models for each variable provided.
update_weights_strat_wilson

Helper Function for the Estimation of Weights for prop_strat_wilson

**Description**

[Stable]

This function wraps the iteration procedure that allows you to estimate the weights for each proportional strata. This assumes to minimize the weighted squared length of the confidence interval.

**Usage**

```r
description

update_weights_strat_wilson(  
  vars,  
  strata_qnorm,  
  initial_weights,  
  n_per_strata,  
  max_iterations = 50,  
  conf_level = 0.95,  
  tol = 0.001
)
```

**Arguments**

- **vars** (numeric)
  normalized proportions for each strata.
- **strata_qnorm** (numeric)
  initial estimation with identical weights of the quantiles.
- **initial_weights** (numeric)
  initial weights used to calculate strata_qnorm. This can be optimized in the future if we need to estimate better initial weights.
- **n_per_strata** (numeric)
  number of elements in each strata.
- **max_iterations** (count)
  maximum number of iterations to be tried. Convergence is always checked.
- **conf_level** (proportion)
  confidence level of the interval.
- **tol** (number)
  tolerance threshold for convergence.

**Value**

A list of 3 elements: n_it, weights, and diff_v.
See Also

For references and details see prop_strat_wilson().

Examples

vs <- c(0.011, 0.013, 0.012, 0.014, 0.017, 0.018)
sq <- 0.674
ws <- rep(1 / length(vs), length(vs))
ns <- c(22, 18, 17, 17, 14, 12)

update_weights_strat_wilson(vs, sq, ws, ns, 100, 0.95, 0.001)
Index

!, CombinationFunction-method (combination_function), 17
* datasets
  ex_data, 98
* formatting functions
  extreme_format, 97
  format_count_fraction, 112
  format_count_fraction_fixed_dp, 113
  format_extreme_values, 113
  format_extreme_values_ci, 114
  format_fraction, 115
  format_fraction_fixed_dp, 116
  format_fraction_threshold, 117
  format_xx, 118
  formatting_functions, 111
&, CombinationFunction, CombinationFunction-method (combination_function), 17
a_compare (compare_variables), 22
a_compare(), 63
a_count_occurrences (count_occurrences), 36
a_count_occurrences_by_grade (count_occurrences_by_grade), 39
a_count_patients_with_event (count_patients_with_event), 44
a_count_patients_with_flags (count_patients_with_flags), 47
a_count_values (count_values_FUNS), 51
a_coxreg (cox_regression), 54
a_length_proportion (estimate_multinomial_rsp), 82
a_odds_ratio (odds_ratio), 205
a_proportion (estimate_proportions), 84
a_proportion_diff (prop_diff), 208
a_summary (summarize_variables), 236
a_summary(), 65
abnormal_by_worst_grade_worsen, 139, 199
add_rowcounts, 7
add_rowcounts(), 230
aes_i_label, 8
analyze_colvars_functions, 8, 10, 229, 230
analyze_functions, 9, 9, 230
analyze_num_patients (summarize_num_patients), 232
analyze_num_patients(), 9
analyze_patients_exposure_in_cols(), 9
analyze_vars_in_cols(), 10
analyze_vars_in_cols(), 9, 228
append_varlabels, 14
arrange_grobs, 15
as_rtable, 16
broom::tidy(), 55, 148, 247
combination_function, 17
CombinationFunction (combination_function), 17
CombinationFunction(), 212
CombinationFunction-class (combination_function), 17
combine_counts, 19
combine_groups, 20
combine_groups(), 19
combine_levels, 21
combine_vectors, 21
compare_variables, 22
compare_vars (compare_variables), 22
compare_vars(), 9, 63
ccontrol_coxph, 28
ccontrol_coxph(), 95, 109, 128, 192, 196
ccontrolcox, 28
ccontrolcox(), 55, 60, 93, 102, 146, 189
ccontrol_incidence_rate, 29
ccontrol_lineplot_vars, 30
INDEX

fit_logistic, 104
fit_rsp_step, 106
fit_survival_step, 108
fit_survival_step(), 249
forcats::fct_collate(), 100
forcats::fct_na_value_to_level(), 101
forcats::fct_realvel(), 100
forest_viewport, 110
format_count_fraction, 98, 112, 113–118
format_count_fraction_fixed_dp, 98, 112, 113–118
format_extreme_values, 98, 112, 113, 115–118
format_extreme_values_ci, 98, 112–114, 114, 115–118
format_fraction, 98, 112–115, 116–118
format_fraction_fixed_dp, 98, 112–115, 116, 117, 118
format_fraction_threshold, 98, 112–116, 117, 118
format_xx, 98, 112–117, 118
formatters::list_valid_format_labels(), 111
formatters::sprintf_format(), 111
formatting_functions, 98, 111, 112–118
g_forest, 121
g_ipp(individual_patient_plot), 200
g_ipp(), 160
g_km, 125
g_lineplot, 130
g_step, 134
g_step(), 249
g_waterfall, 137
get_smooths, 120
gpar, 69, 127
gpar(), 16, 223
grid::unit(), 122
gridExtra::ththeme_default(), 154, 156
groups_list_to_df, 120
h_adlb_worsen, 138
h_adsl_adlb_merge_using_worst_flag, 140
h_ancova, 141
h_append_grade_groups, 142
h_append_grade_groups(), 42
h_col_indices, 143
h_count_cumulative, 144
h_cox_regression, 57, 103, 145
h_coxph_df
(h_survival_duration_subgroups), 191
h_coxph_subgroups_df
(h_survival_duration_subgroups), 191
h_coxph_subgroups_df(), 94
h_coxreg_extract_interaction
(cox_regression_inter), 59
h_coxreg_inter_effect
(cox_regression_inter), 59
h_coxreg_inter_estimations
(cox_regression_inter), 59
h_coxreg_mult_cont_df
(h_survival_biomarkers_subgroups), 188
h_coxreg_mult_cont_df(), 93, 189
h_coxreg_multivar_extract
(h_cox_regression), 145
h_coxreg_multivar_formula
(h_cox_regression), 145
h_coxreg_univar_extract
(h_cox_regression), 145
h_coxreg_univar_extract(), 61
h_coxreg_univar_formulas
(h_cox_regression), 145
h_data_plot, 148
h_decompose_gg, 149
h_format_row, 151
h_format_threshold(extreme_format), 97
h_g_ipp, 159
h_g_ipp(), 202
h_get_format_threshold
(extreme_format), 97
h_get_interaction_vars
(h_logistic_regression), 162
h_ggkm, 152
h_glm_inter_term_extract
(h_logistic_regression), 162
h_glm_interaction_extract
(h_logistic_regression), 162
h_glm_simple_term_extract
(h_logistic_regression), 162
h_glm_simple_term_extract(), 165
h_grob_coxph, 154
INDEX

h_grob_median_surv, 155
h_grob_tbl_at_risk, 156
h_grob_y_annot, 158
h_interaction_coef_name
  (h_logistic_regression), 162
h_interaction_term_labels
  (h_logistic_regression), 162
h_km_layout, 161
h_logistic_inter_terms
  (h_logistic_regression), 162
h_logistic_mult_cont_df
  (h_response_biomarkers_subgroups), 176
h_logistic_mult_cont_df(), 89, 176
h_logistic_regression, 162, 248
h_logistic_simple_terms
  (h_logistic_regression), 162
h_map_for_count_abnormal, 166
h_odds_ratio, 168
h_odds_ratio(), 207
h_odds_ratio_df (h_response_subgroups), 178
h_odds_ratio_subgroups_df
  (h_response_subgroups), 178
h_odds_ratio_subgroups_df(), 90
h_or_cat_interaction
  (h_logistic_regression), 162
h_or_cat_interaction()
  (h_logistic_regression), 162
h_or_cont_interaction
  (h_logistic_regression), 165
h_or_interaction
  (h_logistic_regression), 162
h_or_interaction()
  (h_logistic_regression), 165
h_pkparam_sort, 170
h_prop_diff, 173
h_proportion_df (h_response_subgroups), 178
h_proportion_subgroups_df
  (h_response_subgroups), 178
h_proportion_subgroups_df(), 90
h_proportions, 86, 170
h_response_biomarkers_subgroups, 176
h_response_subgroups, 178
h_row_counts(), 219
h_row_first_values(), 219
h_rsp_to_logistic_variables
  (h_response_biomarkers_subgroups), 176
h_simple_term_labels
  (h_logistic_regression), 162
h_split_by_subgroups, 182
h_split_param, 183
h_stack_by_baskets, 184
h_step, 186
h_step_rsp_est (h_step), 186
h_step_rsp_formula (h_step), 186
h_step_survival_est (h_step), 186
h_step_survival_formula (h_step), 186
h_step_trt_effect (h_step), 186
h_step_window (h_step), 186
h_surv_to_coxreg_variables
  (h_survival_biomarkers_subgroups), 188
h_survival_biomarkers_subgroups, 188
h_survival_duration_subgroups, 191
h_survtime_df
  (h_survival_duration_subgroups), 191
h_survtime_subgroups_df
  (h_survival_duration_subgroups), 191
h_survtime_subgroups_df()
  (h_survival_duration_subgroups), 94
h_tab_one_biomarker, 195
h_tab_one_biomarker(), 230
h_tab_rsp_one_biomarker
  (h_response_biomarkers_subgroups), 176
h_tab_rsp_one_biomarker(), 195, 216
h_tab_surv_one_biomarker
  (h_survival_biomarkers_subgroups), 188
h_tab_surv_one_biomarker(), 195, 246
h_tbl_coxph_pairwise, 196
h_tbl_coxph_pairwise(), 154
h_tbl_median_surv, 197
h_worsen_counter, 198
h_xticks, 199
has_count_in_any_col
  (prune_occurrences), 212
has_count_in_cols (prune_occurrences), 212
has_counts_difference
  (prune_occurrences), 212
has_fraction_in_any_col
  (prune_occurrences), 212
or_clogit(prop_diff_strat_nc, prop_diff_nc)

odds_ratio(min(), median(), max())

or_clogit(

individual_patient_plot

has_fractions_difference, has_fraction_in_cols

260

INDEX

logistic_regression_cols

levels()

prop_wald

labeling::extended()

(prop_strat_wilson, h_odds_ratio)

241,

241,

241,

h_prop_diff

h_prop_diff

h_proportions

rtables::add_combo_levels()

response_subgroups

response_biomarkers_subgroups

rtables::analyze_colvars()

rtables::CellValue()

rtables::rtable()

rtables::cont_n_onecol()

r.tables::cont_n_allcols()

rtables_access

rtables::summarize_row_groups()

s_count_values

s_count_patients_with_event

s_count_occurrences_by_grade

s_count_missed_doses

s_count_abnormal_lab_worsen_by_baseline()
s_num_patients
  (summarize_num_patients), 232
s_num_patients_content
  (summarize_num_patients), 232
s_odds_ratio(odds_ratio), 205
s_proportion(estimate_proportions), 84
s_proportion(), 78, 83, 86
s_proportion_diff(prop_diff), 208
s_summary(summarize_variables), 236
s_summary(), 24, 25, 34, 52, 228, 236
s_surv_time(), 34
s_surv_timepoint(), 35
s_test_proportion_diff(), 80, 91, 180
sas_na(), 217
sas_na(), 73
score_occurrences, 218
score_occurrences_cols
  (score_occurrences), 218
score_occurrences_cont_cols
  (score_occurrences), 218
score_occurrences_subtable
  (score_occurrences), 218
split_cols_by_groups, 220
sqrt(), 240
stack_grobs, 223
stat_mean_ci, 224
stat_mean_ci(), 240
stat_mean_pval, 225
stat_mean_pval(), 240
stat_median_ci, 226
stat_median_ci(), 241
stats::as.formula(), 146
stats::binom.test(), 172
stats::glm(), 164, 169, 247
stats::IQR(), 241
stats::mantelhaen.test(), 174
stats::median(), 240
stats::prop.test(), 171, 174
stats::quantile(), 34, 66, 239, 241
stats::sd(), 240, 241
strata_normal_quantile, 227
strata_normal_quantile(), 172
sum(), 240
summarize_ancova(), 10
summarize_change(), 10
summarize_colvars, 228
summarize_colvars(), 10, 64
summarize_coxreg(cox_regression), 54
summarize_coxreg(), 9, 230
summarize_functions, 9, 10, 230
summarize_logistic, 230
summarize_logistic(), 203
summarize_num_patients, 232
summarize_num_patients(), 230, 235
summarize_occurrences_by_grade
  (count_occurrences_by_grade), 39
summarize_occurrences_by_grade(), 230
summarize_patients_events_in_cols(), 230
summarize_patients_exposure_in_cols(), 9, 230
summarize_variables, 236
summarize_vars(summarize_variables), 236
summarize_vars(), 10, 12, 64, 65, 236
surv_time(), 10
surv_timepoint(), 10
survival::clogit(), 164, 169
survival::coxph(), 28, 29, 60, 95, 102, 127, 128, 146, 192, 196, 250
survival::summary.survfit(), 157
survival::survdiff(), 95, 192
survival::survfit(), 35, 126, 149, 153, 155, 197
survival_biomarkers_subgroups, 245
survival_duration_subgroups, 96
tabulate_rsp_biomarkers
  (response_biomarkers_subgroups), 215
tabulate_rsp_subgroups(), 9, 79, 216, 230
tabulate_survival_biomarkers
  (survival_biomarkers_subgroups), 245
tabulate_survival_biomarkers(), 93
tabulate_survival_subgroups(), 9, 80, 246
tern (tern-package), 6
tern-package, 6
tern_ex_aadae(ex_data), 98
tern_ex_adlb(ex_data), 98
tern_ex_adpp(ex_data), 98
tern_ex_adrs(ex_data), 98
tern_ex_ads1(ex_data), 98
tern_ex_adtte(ex_data), 98
test_proportion_diff(), 10
tibble::tibble(), 121
tidy(), 250
tidy.coxreg.multivar (tidy_coxreg), 249
tidy.coxreg.univar (tidy_coxreg), 249
tidy.glm, 247
 tidy.step, 248
tidy.step(), 135
tidy.summary.coxph (tidy_coxreg), 249
tidy_coxreg, 57, 249
to_n, 252
to_string_matrix, 252

univariate, 253
update_weights_strat_wilson, 254
update_weights_strat_wilson(), 172

viewport, 69, 127
viewport(), 16, 74, 223