Package 'tern'

July 17, 2026

Title Create Common TLGs Used in Clinical Trials

Version 0.9.11

Date 2026-07-15

Description Table, Listings, and Graphs (TLG) library for common outputs used in clinical trials.

License Apache License 2.0

URL https://insightsengineering.github.io/tern/, https://github.com/insightsengineering/tern/

BugReports https://github.com/insightsengineering/tern/issues

Depends R (>= 4.4.0), rtables (>= 0.6.15)

Imports broom (>= 1.0.8), car (>= 3.1-3), checkmate (>= 2.3.2), cowplot (>= 1.1.3), dplyr (>= 1.0.0), emmeans (>= 1.10.4), forcats (>= 1.0.0), formatters (>= 0.5.12), ggplot2 (>= 3.5.0), grid, gridExtra (>= 2.0.0), gtable (>= 0.3.0), labeling, lifecycle (>= 0.2.0), MASS (>= 7.3-60), methods, nestcolor (>= 0.1.1), Rdpack (>= 2.4), rlang (>= 1.1.0), scales (>= 1.2.0), stats, survival (>= 3.8-9), tibble (>= 3.2.1), tidyr (>= 0.8.3), utils

Suggests knitr (>= 1.42), lattice (>= 0.18-4), lubridate (>= 1.7.9), rmarkdown (>= 2.28), stringr (>= 1.4.1), svglite (>= 2.1.2), testthat (>= 3.3.0), withr (>= 2.0.0)

VignetteBuilder knitr, rmarkdown

RdMacros lifecycle, Rdpack

Config/Needs/verdepcheck insightsengineering/rtables, tidymodels/broom, cran/car, mllg/checkmate, wilkelab/cowplot, tidyverse/dplyr, rvlenth/emmeans, tidyverse/forcats, insightsengineering/formatters, tidyverse/ggplot2, r-lib/gtable, r-lib/lifecycle, GeoBosh/Rdpack, r-lib/rlang, r-lib/scales, therneau/survival, tidyverse/tibble, tidyverse/tidyr, yihui/knitr, deepayan/lattice, tidyverse/lubridate, insightsengineering/nestcolor, rstudio/rmarkdown, tidyverse/stringr, r-lib/svglite, r-lib/testthat, r-lib/withr

Config/Needs/website insightsengineering/nesttemplate

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 8.0.0

Collate 'formatting_functions.R' 'abnormal.R' 'abnormal_by_baseline.R'
'abnormal_by_marked.R' 'abnormal_by_worst_grade.R'
'abnormal_lab_worsen_by_baseline.R'
'analyze_colvars_functions.R' 'analyze_functions.R'
'analyze_variables.R' 'analyze_vars_in_cols.R'
'argument_convention.R' 'bland_altman.R'
'combination_function.R' 'compare_variables.R'
'control_incidence_rate.R' 'control_logistic.R'
'control_step.R' 'control_survival.R' 'count_cumulative.R'
'count_missed_doses.R' 'count_occurrences.R'
'count_occurrences_by_grade.R'
'count_patients_events_in_cols.R' 'count_patients_with_event.R'
'count_patients_with_flags.R' 'count_values.R'
'cox_regression.R' 'cox_regression_inter.R' 'coxph.R'
'd_pkparam.R' 'data.R' 'decorate_grob.R'
'desctools_binom_diff.R' 'df_explicit_na.R'
'estimate_multinomial_rsp.R' 'estimate_proportion.R'
'fit_rsp_step.R' 'fit_survival_step.R' 'g_forest.R' 'g_ipp.R'
'g_km.R' 'g_lineplot.R' 'g_step.R' 'g_waterfall.R'
'h_adsl_adlb_merge_using_worst_flag.R'
'h_biomarkers_subgroups.R' 'h_cox_regression.R'
'h_incidence_rate.R' 'h_km.R' 'h_logistic_regression.R'
'h_map_for_count_abnormal.R' 'h_pkparam_sort.R'
'h_response_biomarkers_subgroups.R' 'h_response_subgroups.R'
'h_stack_by_baskets.R' 'h_step.R'
'h_survival_biomarkers_subgroups.R'
'h_survival_duration_subgroups.R' 'imputation_rule.R'
'incidence_rate.R' 'logistic_regression.R' 'missing_data.R'
'odds_ratio.R' 'package.R' 'prop_diff.R' 'prop_diff_test.R'
'prune_occurrences.R' 'response_biomarkers_subgroups.R'
'response_subgroups.R' 'riskdiff.R' 'rtables_access.R'
'score_occurrences.R' 'split_cols_by_groups.R' 'stat.R'
'summarize_ancova.R' 'summarize_change.R' 'summarize_colvars.R'
'summarize_coxreg.R' 'summarize_functions.R'
'summarize_glm_count.R' 'summarize_num_patients.R'
'summarize_patients_exposure_in_cols.R'
'survival_biomarkers_subgroups.R' 'survival_coxph_pairwise.R'
'survival_duration_subgroups.R' 'survival_time.R'
'survival_timepoint.R' 'utils.R' 'utils_checkmate.R'
'utils_default_stats_formats_labels.R' 'utils_factor.R'
'utils_ggplot.R' 'utils_grid.R' 'utils_rtables.R'

'utils_split_funs.R'

NeedsCompilation no

Author Joe Zhu [aut, cre] (ORCID: <https://orcid.org/0000-0001-7566-2787>),
Daniel Sabanés Bové [aut],
Jana Stoilova [aut],
Davide Garolini [aut] (ORCID: <https://orcid.org/0000-0002-1445-1369>),
Emily de la Rua [aut] (ORCID: <https://orcid.org/0009-0000-8738-5561>),
Abinaya Yogasekaram [aut] (ORCID:
<https://orcid.org/0009-0005-2083-1105>),
Heng Wang [aut],
Francois Collin [aut],
Adrian Waddell [aut],
Pawel Rucki [aut],
Chendi Liao [aut],
Jennifer Li [aut],
David Munoz Tord [ctb],
F. Hoffmann-La Roche AG [cph, fnd]

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

Repository CRAN

Date/Publication 2026-07-17 06:40:02 UTC

Contents

`tern-package` *tern Package*

Description

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

Author(s)

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

Authors:

- Joe Zhu <joe.zhu@roche.com> (ORCID)
- Daniel Sabanés Bové <daniel.sabanes_bove@rconis.com>
- Jana Stoilova <stoilova@dnli.com>
- Davide Garolini <davide.garolini@roche.com> (ORCID)
- Emily de la Rua <emilydelarua@gmail.com> (ORCID)
- Abinaya Yogasekaram <ayogasek@gmail.com> (ORCID)
- Heng Wang <Heng.Wang21@gilead.com>
- Francois Collin
- Adrian Waddell <adrian.waddell@gmail.com>
- Pawel Rucki <pawel.rucki@roche.com>
- Chendi Liao
- Jennifer Li

Other contributors:

- David Munoz Tord <david.munoztord@mailbox.org> [contributor]
- F. Hoffmann-La Roche AG [copyright holder, funder]

See Also

Useful links:

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

`add_riskdiff` *Split function to configure risk difference column*

Description

[Stable]

Wrapper function for `rtables::add_combo_levels()` which configures settings for the risk difference column to be added to an `rtables` object. To add a risk difference column to a table, this function should be used as `split_fun` in calls to `rtables::split_cols_by()`, followed by setting argument `riskdiff` to `TRUE` in all following analyze function calls.

Usage

```
add_riskdiff(
  arm_x,
  arm_y,
  col_label = paste0("Risk Difference (%) (95% CI)", if (length(arm_y) > 1)
    paste0("\n", arm_x, " vs. ", arm_y)),
  pct = TRUE
)
```

Arguments

`arm_x`	(`string`) name of reference arm to use in risk difference calculations.
`arm_y`	(`character`) names of one or more arms to compare to reference arm in risk difference calculations. A new column will be added for each value of `arm_y`.
`col_label`	(`character`) labels to use when rendering the risk difference column within the table. If more than one comparison arm is specified in `arm_y`, default labels will specify which two arms are being compared (reference arm vs. comparison arm).
`pct`	(`flag`) whether output should be returned as percentages. Defaults to `TRUE`.

Value

A closure suitable for use as a split function (`split_fun`) within `rtables::split_cols_by()` when creating a table layout.

See Also

`stat_propdiff_ci()` for details on risk difference calculation.

Examples

```
adae <- tern_ex_adae
adae$AESEV <- factor(adae$AESEV)

lyt <- basic_table() |>
 split_cols_by("ARMCD", split_fun = add_riskdiff(arm_x = "ARM A", arm_y = c("ARM B", "ARM C"))) |>
  count_occurrences_by_grade(
    var = "AESEV",
    riskdiff = TRUE
  )

tbl <- build_table(lyt, df = adae)
tbl
```

---

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, alt_counts = FALSE)
```

Arguments

`lyt`	(`PreDataTableLayouts`) layout that analyses will be added to.
`alt_counts`	(`flag`) whether row counts should be taken from `alt_counts_df` (`TRUE`) or from `df` (`FALSE`). Defaults to `FALSE`.

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

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

Arguments

aesi	(character) vector with standardized MedDRA query name (e.g. SMQxxNAM) or customized query name (e.g. CQxxNAM).
scope	(character) vector with scope of query (e.g. SMQxxSC).

Value

A string with the standard label for the AE basket.

Examples

```
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 in 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 where 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_exposu` 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()`
- `analyze_vars()`
- `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()
- surv_time()
- surv_timepoint()
- test_proportion_diff()

See Also

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

analyze_variables *Analyze variables*

Description

[Stable]

The analyze function `analyze_vars()` creates a layout element to summarize one or more variables, using the S3 generic function `s_summary()` to calculate a list of summary statistics. A list of all available statistics for numeric variables can be viewed by running `get_stats("analyze_vars_numeric")` and for non-numeric variables by running `get_stats("analyze_vars_counts")`. Use the `.stats` parameter to specify the statistics to include in your output summary table. Use `compare_with_ref_group = TRUE` to compare the variable with reference groups.

Usage

```
analyze_vars(
  lyt,
  vars,
  var_labels = vars,
  na_str = default_na_str(),
  na_str_drop = "<Missing>",
  nested = TRUE,
  show_labels = "default",
  table_names = vars,
  section_div = NA_character_,
  ...,
  na_rm = TRUE,
  compare_with_ref_group = FALSE,
  .stats = c("n", "mean_sd", "median", "range", "count_fraction"),
  .stat_names = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL,
  formats_var = NULL,
  na_strs_var = NULL,
  format = NULL
)

s_summary(x, ...)

## S3 method for class 'numeric'
s_summary(x, control = control_analyze_vars(), ...)

## S3 method for class 'factor'
s_summary(x, denom = c("n", "N_col", "N_row"), ...)
```

```
## S3 method for class 'character'
s_summary(x, denom = c("n", "N_col", "N_row"), ...)

## S3 method for class 'logical'
s_summary(x, denom = c("n", "N_col", "N_row"), ...)

a_summary(
  x,
  ...,
  .stats = NULL,
  .stat_names = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```

Arguments

lyt	(PreDataTableLayouts) layout that analyses will be added to.
vars	(character) variable names for the primary analysis variable to be iterated over.
var_labels	(character) variable labels.
na_str	(string) string used to replace all NA or empty values in the output.
na_str_drop	(string) Additional NA string to be dropped from factor calculations. If NULL nothing will be removed beyond standard NA handling.
nested	(flag) whether this layout instruction should 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.
show_labels	(string) label visibility: one of "default", "visible" and "hidden".
table_names	(character) this can be customized in the 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.
...	additional arguments passed to s_summary(), including: • denom: (string) See parameter description below. • .N_row: (numeric(1)) Row-wise N (row group count) for the group of observations being analyzed (i.e. with no column-based subsetting).

- `.N_col`: (`numeric(1)`) Column-wise N (column count) for the full column being tabulated within.
- `verbose`: (`flag`) Whether additional warnings and messages should be printed. Mainly used to print out information about factor casting. Defaults to `TRUE`. Used for `character`/`factor` variables only.

`na_rm` (`flag`)
whether `NA` values should be removed from `x` prior to analysis.

`compare_with_ref_group` (`flag`)
whether comparison statistics should be analyzed instead of summary statistics (`compare_with_ref_group = TRUE` adds `pval` statistic comparing against reference group).

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

Options for numeric variables are: `'n', 'sum', 'mean', 'sd', 'se', 'mean_sd', 'mean_se', 'mean`

Options for non-numeric variables are: `'n', 'count', 'count_fraction', 'count_fraction_fixed_`

`.stat_names` (`character`)
names of the statistics that are passed directly to name single statistics (`.stats`). This option is visible when producing `rtables::as_result_df()` with `make_ard = TRUE`.

`.formats` (named `character` or `list`)
formats for the statistics. See Details in `analyze_vars` for more information on the `"auto"` setting.

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

`.indent_mods` (named `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.

`formats_var` (`NULL` or `string`)
Passed to `rtables::analyze()`. `.formats` must be `"default"` and `format` must be `NULL` when this is non-NULL.

`na_strs_var` (`string` or `NULL`)
Passed to `analyze`. `na_str` must be `NA` when this is non-NULL.

`format` (`NULL`, `list`, `string` or `function`)
Passed to `rtables::analyze()`. `.formats` must be `"default"` and `formats_var` must be `NULL` when this is non-NULL.

`x` (`numeric`)
vector of numbers we want to analyze.

`control` (`list`)
parameters for descriptive statistics details, specified by using the helper function `control_analyze_vars()`. Some possible parameter options are:

- `conf_level` (`proportion`)
  confidence level of the interval for mean and median.

- quantiles (numeric(2))
  vector of length two to specify the quantiles.
- quantile_type (numeric(1))
  between 1 and 9 selecting quantile algorithms to be used. See more about type in stats::quantile().
- test_mean (numeric(1))
  value to test against the mean under the null hypothesis when calculating p-value.

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.

Details

Automatic digit formatting: The number of digits to display can be automatically determined from the analyzed variable(s) (vars) for certain statistics by setting the statistic format to "auto" in .formats. This utilizes the format_auto() formatting function. Note that only data for the current row & variable (for all columns) will be considered (.df_row[[.var]], see rtables::additional_fun_params) and not the whole dataset.

Value

- analyze_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.
- 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 max() of x.
  - max: The min() 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().

Functions

- analyze_vars(): Layout-creating function which can take statistics function arguments and additional format arguments. This function is a wrapper for rtables::analyze().
- 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 analyze_vars() and compare_vars() and as cfun in summarize_colvars().

Note

- If x is an empty vector, NA is returned. This is the expected feature so as to return rcell 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, 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 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.
- 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().
- To use for comparison (with additional p-value statistic), parameter compare_with_ref_group must be set to TRUE.
- Ensure that either all NA values are converted to an explicit NA level or all NA values are left as is.

Examples

```
## 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))
)

# `analyze_vars()` in `rtables` pipelines
## Default output within a `rtables` pipeline.
l <- basic_table() |>
  split_cols_by(var = "ARM") |>
  split_rows_by(var = "AVISIT") |>
  analyze_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") |>
  analyze_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"))
  )

build_table(l, df = dta_test)

## Use arguments interpreted by `s_summary`.
```

```
l <- basic_table() |>
  split_cols_by(var = "ARM") |>
  split_rows_by(var = "AVISIT") |>
  analyze_vars(vars = "AVAL", na_rm = FALSE)

build_table(l, df = dta_test)

## Handle `NA` levels first when summarizing factors.
dta_test$AVISIT <- NA_character_
dta_test <- df_explicit_na(dta_test)
l <- basic_table() |>
  split_cols_by(var = "ARM") |>
  analyze_vars(vars = "AVISIT", na_rm = FALSE)

build_table(l, df = dta_test)

# auto format
dt <- data.frame("VAR" = c(0.001, 0.2, 0.0011000, 3, 4))
basic_table() |>
  analyze_vars(
    vars = "VAR",
    .stats = c("n", "mean", "mean_sd", "range"),
    .formats = c("mean_sd" = "auto", "range" = "auto")
  ) |>
  build_table(dt)

# `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)

## Benefits in `rtables` contructions:
dta_test <- data.frame(
  Group = rep(LETTERS[seq(3)], each = 2),
  sub_group = rep(letters[seq(2)], each = 3),
  x = seq(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")))

# Empty factor returns zero-filled items.
s_summary(factor(levels = c("a", "b", "c")))

## 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"), verbose = FALSE)
s_summary(c("a", "a", "b", "c", "a", ""), .var = "x", na_rm = FALSE, verbose = FALSE)

# `s_summary.logical`

## Basic usage:
s_summary(c(TRUE, FALSE, TRUE, TRUE))

# Empty factor returns zero-filled items.
s_summary(as.logical(c()))

## Management of NA values.
x <- c(NA, TRUE, FALSE)
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)

a_summary(factor(c("a", "a", "b", "c", "a")), .N_row = 10, .N_col = 10)
a_summary(
  factor(c("a", "a", "b", "c", "a")),
  .ref_group = factor(c("a", "a", "b", "c")), compare_with_ref_group = TRUE, .in_ref_col = TRUE
)
```

```
a_summary(c("A", "B", "A", "C"), .var = "x", .N_col = 10, .N_row = 10, verbose = FALSE)
a_summary(
  c("A", "B", "A", "C"),
  .ref_group = c("B", "A", "C"), .var = "x", compare_with_ref_group = TRUE, verbose = FALSE,
  .in_ref_col = FALSE
)

a_summary(c(TRUE, FALSE, FALSE, TRUE, TRUE), .N_row = 10, .N_col = 10)
a_summary(
  c(TRUE, FALSE, FALSE, TRUE, TRUE),
  .ref_group = c(TRUE, FALSE), .in_ref_col = TRUE, compare_with_ref_group = TRUE,
  .in_ref_col = FALSE
)

a_summary(rnorm(10), .N_col = 10, .N_row = 20, .var = "bla")
a_summary(rnorm(10, 5, 1),
  .ref_group = rnorm(20, -5, 1), .var = "bla", compare_with_ref_group = TRUE,
  .in_ref_col = FALSE
)
```

`analyze_vars_in_cols` *Analyze numeric variables in columns*

Description

[Experimental]

The layout-creating function `analyze_vars_in_cols()` creates a layout element to generate a column-wise analysis table.

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"),
  row_labels = NULL,
  do_summarize_row_groups = FALSE,
  split_col_vars = TRUE,
  imp_rule = NULL,
  avalcat_var = "AVALCAT1",
  cache = FALSE,
  .indent_mods = NULL,
```

```
  na_str = default_na_str(),
  nested = TRUE,
  .formats = NULL,
  .aligns = NULL
)
```

Arguments

lyt	(PreDataTableLayouts) layout that 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, usually .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.
imp_rule	(string or NULL) imputation rule setting. Defaults to NULL for no imputation rule. Can also be "1/3" to implement 1/3 imputation rule or "1/2" to implement 1/2 imputation rule. In order to use an imputation rule, the avalcat_var argument must be specified. See imputation_rule() for more details on imputation.
avalcat_var	(string) if imp_rule is not NULL, name of variable that indicates whether a row in the data corresponds to an analysis value in category "BLQ", "LTR", "<PCLLOQ", or none of the above (defaults to "AVALCAT1"). Variable must be present in the data and should match the variable used to calculate the n_blq statistic (if included in .stats).

cache	(flag) whether to store computed values in a temporary caching environment. This will speed up calculations in large tables, but should be set to FALSE if the same rtable layout is used for multiple tables with different data. Defaults to FALSE.
.indent_mods	(named integer) indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.
na_str	(string) string used to replace all NA or empty values in the output.
nested	(flag) whether this layout instruction should 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.
.formats	(named character or list) formats for the statistics. See Details in analyze_vars for more information on the "auto" setting.
.aligns	(character or NULL) alignment for table contents (not including labels). When NULL, "center" is applied. See formatters::list_valid_aligns() for a list of all currently supported alignments.

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.

Note

- This is an experimental implementation of rtables::summarize_row_groups() and rtables::analyze_colvars() that may be subjected to changes as rtables extends its support to more complex analysis pipelines in the column space. We encourage users to read the examples carefully and file issues for different use cases.
- In this function, labelstr behaves atypically. If labelstr = NULL (the default), row labels are assigned automatically as the split values if do_summarize_row_groups = FALSE (the default), and as the group label if do_summarize_row_groups = TRUE.

See Also

analyze_vars(), 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_labels = "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(
    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_labels = "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_labels = "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 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`	(`PreDataTableLayouts`) layout that 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(1)`) 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

```
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 *Arrange multiple grobs*

Description

[Deprecated]

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

Usage

```
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 grob) a list of grobs.
ncol	(integer(1)) number of columns in layout.
nrow	(integer(1)) number of rows in layout.
padding_ht	(grid::unit) unit of length 1, vertical space between each grob.
padding_wt	(grid::unit) unit of length 1, horizontal space between each grob.
vp	(viewport or NULL) a viewport() object (or NULL).
gp	(gpar) a gpar() object.
name	(string) a character identifier for the grob.

Value

A grob.

Examples

```
library(grid)


num <- lapply(1:9, textGrob)
grid::grid.newpage()
grid.draw(arrange_grobs(grobs = num, ncol = 2))

showViewport()

g1 <- circleGrob(gp = gpar(col = "blue"))
g2 <- circleGrob(gp = gpar(col = "red"))
g3 <- textGrob("TEST TEXT")
grid::grid.newpage()
grid.draw(arrange_grobs(g1, g2, g3, nrow = 2))

showViewport()

grid::grid.newpage()
grid.draw(arrange_grobs(g1, g2, g3, ncol = 3))

grid::grid.newpage()
grid::pushViewport(grid::viewport(layout = grid::grid.layout(1, 2)))
vp1 <- grid::viewport(layout.pos.row = 1, layout.pos.col = 2)
```

```
grid.draw(arrange_grobs(g1, g2, g3, ncol = 2, vp = vp1))

showViewport()
```

`as.rtable` *Convert to* `rtable`

Description

[Stable]

This is a new generic function to convert objects to `rtable` tables.

Usage

```
as.rtable(x, ...)

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

Arguments

`x`	(`data.frame`) the object which should be converted to an `rtable`.
`...`	additional arguments for methods.
`format`	(`string` or `function`) 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 a `data.frame` that contains numeric columns to `rtable`.

Examples

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

check_diff_prop_ci *Check proportion difference arguments*

Description

[Stable]

Verifies that and/or convert arguments into valid values to be used in the estimation of difference in responder proportions.

Usage

```
check_diff_prop_ci(rsp, grp, strata = NULL, conf_level, correct = NULL)
```

Arguments

rsp	(logical) vector indicating 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).
strata	(factor) variable with one level per stratum and same length as rsp.
conf_level	(proportion) confidence level of the interval.
correct	(flag) whether to include the continuity correction. For further information, see stats::prop.test().

Examples

```
# example code
## "Mid" case: 4/4 respond in group A, 1/2 respond in group B.
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
)
check_diff_prop_ci(rsp = dta[["rsp"]], grp = dta[["grp"]], conf_level = 0.95)
```

`clogit_with_tryCatch` *Wrapper function of survival::clogit*

Description

When model fitting failed, a more useful message would show.

Usage

```
clogit_with_tryCatch(formula, data, ...)
```

Arguments

`formula`	Model formula.
`data`	data frame.
`...`	further parameters to be added to survival::clogit.

Value

When model fitting is successful, an object of class "clogit".
When model fitting failed, an error message is shown.

Examples

```
## Not run:
library(dplyr)
adrs_local <- tern_ex_adrs |>
  dplyr::filter(ARMCD %in% c("ARM A", "ARM B")) |>
  dplyr::mutate(
    RSP = dplyr::case_when(AVALC %in% c("PR", "CR") ~ 1, TRUE ~ 0),
    ARMBIN = droplevels(ARMCD)
  )
dta <- adrs_local
dta <- dta[sample(nrow(dta)), ]
mod <- clogit_with_tryCatch(formula = RSP ~ ARMBIN * AGE + strata(STRATA1), data = dta)

## End(Not run)
```

combination_function *Class for* CombinationFunction

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

e1	(CombinationFunction) left hand side of logical operator.
e2	(CombinationFunction) right hand side of logical operator.
x	(CombinationFunction) the function which should be negated.

Value

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

Functions

- e1 & e2: 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.
- e1 | e2: 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 results.

Examples

```
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() |>
  analyze_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
)

basic_table() |>
  split_cols_by_groups("ARM", groups) |>
  add_colcounts() |>
  analyze_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

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

Arguments

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

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

basic_table() |>
  split_cols_by_groups("ARM", groups) |>
  add_colcounts() |>
  analyze_vars("AGE") |>
  build_table(DM)
```

---

combine_vectors *Element-wise combination of two vectors*

Description

Element-wise combination of two vectors

Usage

```
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)
```

`compare_variables` *Compare variables between groups*

Description

[Stable]

The analyze function `compare_vars()` creates a layout element to summarize and compare one or more variables, using the S3 generic function `s_summary()` to calculate a list of summary statistics. A list of all available statistics for numeric variables can be viewed by running `get_stats("analyze_vars_numeric", add_pval = TRUE)` and for non-numeric variables by running `get_stats("analyze_vars_counts", add_pval = TRUE)`. Use the `.stats` parameter to specify the statistics to include in your output summary table.

Prior to using this function in your table layout you must use `rtables::split_cols_by()` to create a column split on the variable to be used in comparisons, and specify a reference group via the `ref_group` parameter. Comparisons can be performed for each group (column) against the specified reference group by including the p-value statistic.

Usage

```
compare_vars(
  lyt,
  vars,
  var_labels = vars,
  na_str = default_na_str(),
  nested = TRUE,
  ...,
  na_rm = TRUE,
  show_labels = "default",
  table_names = vars,
  section_div = NA_character_,
  .stats = c("n", "mean_sd", "count_fraction", "pval"),
  .stat_names = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)

s_compare(x, ...)

## S3 method for class 'numeric'
s_compare(x, ...)
```

```
## S3 method for class 'factor'
s_compare(x, ...)

## S3 method for class 'character'
s_compare(x, ...)

## S3 method for class 'logical'
s_compare(x, ...)
```

Arguments

lyt	(PreDataTableLayouts) layout that analyses will be added to.
vars	(character) variable names for the primary analysis variable to be iterated over.
var_labels	(character) variable labels.
na_str	(string) string used to replace all NA or empty values in the output.
nested	(flag) whether this layout instruction should 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.
...	additional arguments passed to s_compare(), including: • denom: (string) choice of denominator. Options are c("n", "N_col", "N_row"). For factor variables, can only be "n" (number of values in this row and column intersection). • .N_row: (numeric(1)) Row-wise N (row group count) for the group of observations being analyzed (i.e. with no column-based subsetting). • .N_col: (numeric(1)) Column-wise N (column count) for the full column being tabulated within. • verbose: (flag) Whether additional warnings and messages should be printed. Mainly used to print out information about factor casting. Defaults to TRUE. Used for character/factor variables only.
na_rm	(flag) whether NA values should be removed from x prior to analysis.
show_labels	(string) label visibility: one of "default", "visible" and "hidden".
table_names	(character) this can be customized in the 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. Options for numeric variables are: 'n', 'sum', 'mean', 'sd', 'se', 'mean_sd', 'mean_se', 'mear Options for non-numeric variables are: 'n', 'count', 'count_fraction', 'count_fraction_fixed_
.stat_names	(character) names of the statistics that are passed directly to name single statistics (.stats). This option is visible when producing rtables::as_result_df() with make_ard = TRUE.
.formats	(named character or list) formats for the statistics. See Details in analyze_vars for more information on the "auto" setting.
.labels	(named character) labels for the statistics (without indent).
.indent_mods	(named 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.
x	(numeric) vector of numbers we want to analyze.

Value

- 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.

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

Functions

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

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()`.
- 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

`s_summary()` which is used internally to compute a summary within `s_compare()`, and `a_summary()` which is used (with `compare = TRUE`) as the analysis function for `compare_vars()`.

Examples

```
# `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)

# `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)
```

`control_analyze_vars` *Control function for descriptive statistics*

Description

[Stable]

Sets a list of parameters for summaries of descriptive statistics. Typically used internally to specify details for `s_summary()`. This function family is mainly used by `analyze_vars()`.

Usage

```
control_analyze_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(2)`) vector of length two to specify the quantiles to calculate.
`quantile_type`	(`numeric(1)`) number 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(1)`) number 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_annot` *Control functions for Kaplan-Meier plot annotation tables*

Description

[Stable]

Auxiliary functions for controlling arguments for formatting the annotation tables that can be added to plots generated via `g_km()`.

Usage

```
control_surv_med_annot(
  x = 0.8,
  y = 0.85,
  w = 0.32,
  h = 0.16,
  fill = TRUE,
  digits = 4
)

control_coxph_annot(
  x = 0.29,
  y = 0.51,
  w = 0.4,
  h = 0.125,
  fill = TRUE,
  ref_lbls = FALSE
)
```

Arguments

x	(proportion) x-coordinate for center of annotation table.
y	(proportion) y-coordinate for center of annotation table.
w	(proportion) relative width of the annotation table.
h	(proportion) relative height of the annotation table.
fill	(flag or character) whether the annotation table should have a background fill color. Can also be a color code to use as the background fill color. If TRUE, color code defaults to "#00000020".
digits	(integer(1)) number of significant digits for median survival time and confidence interval values. Defaults to 4.
ref_lbls	(flag) whether the reference group should be explicitly printed in labels for the annotation table. If FALSE (default), only comparison groups will be printed in the table labels.

Value

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

Functions

- `control_surv_med_annot()`: Control function for formatting the median survival time annotation table. This annotation table can be added in `g_km()` by setting `annot_surv_med=TRUE`, and can be configured using the `control_surv_med_annot()` function by setting it as the `control_annot_surv_med` argument.
- `control_coxph_annot()`: Control function for formatting the Cox-PH annotation table. This annotation table can be added in `g_km()` by setting `annot_coxph=TRUE`, and can be configured using the `control_coxph_annot()` function by setting it as the `control_annot_coxph` argument.

See Also

`g_km()`

Examples

```
control_surv_med_annot()
control_surv_med_annot(digits = 2)

control_coxph_annot()
```

---

control_coxph *Control function for Cox-PH model*

Description

[Stable]

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

Usage

```
control_coxph(
  pval_method = c("log-rank", "wald", "likelihood"),
  ties = c("efron", "breslow", "exact"),
  conf_level = 0.95,
  alternative = c("two.sided", "less", "greater")
)
```

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)
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.

alternative (string)
alternative hypothesis for the p-value test. Default is "two.sided", can also be set to "less" or "greater" for one-sided testing. Note that one-sided testing is not supported when pval_method = "likelihood".

Value

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

---

control_coxreg *Control function for Cox regression*

Description

[Stable]

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

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_incidence_rate`

Control function for incidence rate

Description

[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
)
```

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(1)`) number of patient-years to use when calculating adverse event rates.

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()`

Description

[Stable]

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

Usage

```
control_lineplot_vars(
  x = "AVISIT",
  y = "AVAL",
  group_var = "ARM",
  facet_var = NA,
  paramcd = "PARAMCD",
  y_unit = "AVALU",
  subject_var = "USUBJID"
)
```

Arguments

`x`	(`string`) x-variable name.
`y`	(`string`) y-variable name.
`group_var`	(`string` or `NA`) group variable name.
`facet_var`	(`string` or `NA`) faceting variable name.
`paramcd`	(`string` or `NA`) parameter code variable name.
`y_unit`	(`string` or `NA`) y-axis unit variable name.
`subject_var`	(`string` or `NA`) subject variable name.

Value

A named character vector of variable names.

Examples

```
control_lineplot_vars()
control_lineplot_vars(group_var = 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

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

Arguments

`response_definition`	(`string`) the definition of what an event is in terms of `response`. This will be used when fitting the logistic regression model on the left hand side of the formula. Note that the evaluated expression should result in either a logical vector or a factor with 2 levels. By default this is just `"response"` such that the original response variable is used and not modified further.
`conf_level`	(`proportion`) confidence level of the interval.

Value

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

Examples

```
# Standard options.
control_logistic()

# Modify confidence level.
control_logistic(conf_level = 0.9)

# Use a different response definition.
control_logistic(response_definition = "I(response %in% c('CR', 'PR'))")
```

`control_riskdiff` *Control function for risk difference column*

Description

[Stable]

Sets a list of parameters to use when generating a risk (proportion) difference column. Used as input to the `riskdiff` parameter of `tabulate_rsp_subgroups()` and `tabulate_survival_subgroups()`.

Usage

```
control_riskdiff(
  arm_x = NULL,
  arm_y = NULL,
  format = "xx.x (xx.x - xx.x)",
  col_label = "Risk Difference (%) (95% CI)",
  pct = TRUE
)
```

Arguments

`arm_x`	(`string`) name of reference arm to use in risk difference calculations.
`arm_y`	(`character`) names of one or more arms to compare to reference arm in risk difference calculations. A new column will be added for each value of `arm_y`.
`format`	(`string` or `function`) the format label (string) or formatting function to apply to the risk difference statistic. See the `3d` string options in `formatters::list_valid_format_labels()` for possible format strings. Defaults to `"xx.x (xx.x - xx.x)"`.
`col_label`	(`character`) labels to use when rendering the risk difference column within the table. If more than one comparison arm is specified in `arm_y`, default labels will specify which two arms are being compared (reference arm vs. comparison arm).
`pct`	(`flag`) whether output should be returned as percentages. Defaults to `TRUE`.

Value

A `list` of items with names corresponding to the arguments.

See Also

`add_riskdiff()`, `tabulate_rsp_subgroups()`, and `tabulate_survival_subgroups()`.

Examples

```
control_riskdiff()
control_riskdiff(arm_x = "ARM A", arm_y = "ARM B")
```

`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`	(`numeric(1)` 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`	(`integer(1)`) 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`	(`integer(1)`) 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)
```

---

control_surv_time *Control function for* `survfit` *models 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.

Usage

```
control_surv_time(
  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(2)`) vector of length two specifying the quantiles of survival time.

Value

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

control_surv_timepoint *Control function for* survfit *models for patients' survival rate at time points*

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 *Count occurrences*

Description

[Stable]

The analyze function count_occurrences() creates a layout element to calculate occurrence counts for patients.

This function analyzes the variable(s) supplied to vars and returns a table of occurrence counts for each unique value (or level) of the variable(s). This variable (or variables) must be non-numeric. The id variable is used to indicate unique subject identifiers (defaults to USUBJID).

If there are multiple occurrences of the same value recorded for a patient, the value is only counted once.

The summarize function summarize_occurrences() performs the same function as count_occurrences() except it creates content rows, not data rows, to summarize the current table row/column context and operates on the level of the latest row split or the root of the table if no row splits have occurred.

Usage

```
count_occurrences(
  lyt,
  vars,
  id = "USUBJID",
  drop = TRUE,
  var_labels = vars,
  show_labels = "hidden",
  riskdiff = FALSE,
  na_str = default_na_str(),
  nested = TRUE,
  ...,
  table_names = vars,
  .stats = "count_fraction_fixed_dp",
  .stat_names = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)

summarize_occurrences(
  lyt,
  var,
  id = "USUBJID",
  drop = TRUE,
  riskdiff = FALSE,
  na_str = default_na_str(),
  ...,
  .stats = "count_fraction_fixed_dp",
  .stat_names = NULL,
  .formats = NULL,
  .indent_mods = 0L,
  .labels = NULL
)

s_count_occurrences(
  df,
  .var = "MHDECOD",
  .N_col,
  .N_row,
  .df_row,
  ...,
  drop = TRUE,
  id = "USUBJID",
```

```
  denom = c("N_col", "n", "N_row")
)

a_count_occurrences(
  df,
  labelstr = "",
  ...,
  .stats = NULL,
  .stat_names = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```

Arguments

lyt	(PreDataTableLayouts) layout that analyses will be added to.
vars	(character) variable names for the primary analysis variable to be iterated over.
id	(string) subject variable name.
drop	(flag) whether non-appearing occurrence levels should be dropped from the resulting table. Note that in that case the remaining occurrence levels in the table are sorted alphabetically.
var_labels	(character) variable labels.
show_labels	(string) label visibility: one of "default", "visible" and "hidden".
riskdiff	(flag) whether a risk difference column is present. When set to TRUE, add_riskdiff() must be used as split_fun in the prior column split of the table layout, specifying which columns should be compared. See stat_propdiff_ci() for details on risk difference calculation.
na_str	(string) string used to replace all NA or empty values in the output.
nested	(flag) whether this layout instruction should 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.
...	additional arguments for the lower level functions.
table_names	(character) this can be customized in the case that the same vars are analyzed multiple times, to avoid warnings from rtables.

.stats (character)
statistics to select for the table.
Options are: 'count', 'count_fraction', 'count_fraction_fixed_dp', 'fraction'

.stat_names (character)
names of the statistics that are passed directly to name single statistics (.stats). This option is visible when producing rtables::as_result_df() with make_ard = TRUE.

.formats (named character or list)
formats for the statistics. See Details in analyze_vars for more information on the "auto" setting.

.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.

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.

.N_col (integer(1))
column-wise N (column count) for the full column being analyzed that is typically passed by rtables.

.N_row (integer(1))
row-wise N (row group count) for the group of observations being analyzed (i.e. with no column-based subsetting) that is typically passed by rtables.

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

denom (string)
choice of denominator for proportion. Options are:

- N_col: total number of patients in this column across rows.
- n: number of patients with any occurrences.
- N_row: total number of patients in this row across columns.

labelstr (string)
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.

Value

- 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.

- summarize_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 content rows containing the statistics from s_count_occurrences() to the table layout.
- 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().

Functions

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

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

```
library(dplyr)
df <- data.frame(
  USUBJID = as.character(c(
    1, 1, 2, 4, 4, 4,
    6, 6, 6, 7, 7, 8
  )),
  MHDECOD = c(
    "MH1", "MH2", "MH1", "MH1", "MH1", "MH3",
    "MH2", "MH2", "MH3", "MH1", "MH2", "MH4"
  ),
  ARM = rep(c("A", "B"), each = 6),
  SEX = c("F", "F", "M", "M", "M", "M", "F", "F", "F", "M", "M", "F")
)
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
tbl <- lyt |>
  build_table(df, alt_counts_df = df_adsl) |>
  prune_table()

tbl

# Layout creating function with custom format.
basic_table() |>
  add_colcounts() |>
  split_rows_by("SEX", child_labels = "visible") |>
  summarize_occurrences(
    var = "MHDECOD",
    .formats = c("count_fraction" = "xx.xx (xx.xx%)")
  ) |>
  build_table(df, alt_counts_df = df_adsl)

# Count unique occurrences per subject.
s_count_occurrences(
  df,
  .N_col = 4L,
  .N_row = 4L,
  .df_row = df,
  .var = "MHDECOD",
  id = "USUBJID"
)

a_count_occurrences(
  df,
  .N_col = 4L,
  .df_row = df,
  .var = "MHDECOD",
  id = "USUBJID"
)
```

---

count_occurrences_by_grade

Count occurrences by grade

Description

[Stable]

The analyze function count_occurrences_by_grade() creates a layout element to calculate occurrence counts by grade.

This function analyzes primary analysis variable `var` which indicates toxicity grades. The `id` variable is used to indicate unique subject identifiers (defaults to `USUBJID`). The user can also supply a list of custom groups of grades to analyze via the `grade_groups` parameter. The `remove_single` argument will remove single grades from the analysis so that *only* grade groups are analyzed.

If there are multiple grades recorded for one patient only the highest grade level is counted.

The summarize function `summarize_occurrences_by_grade()` performs the same function as `count_occurrences_by_grade()` except it creates content rows, not data rows, to summarize the current table row/column context and operates on the level of the latest row split or the root of the table if no row splits have occurred.

Usage

```
count_occurrences_by_grade(
  lyt,
  var,
  id = "USUBJID",
  grade_groups = list(),
  remove_single = TRUE,
  only_grade_groups = FALSE,
  var_labels = var,
  show_labels = "default",
  riskdiff = FALSE,
  na_str = default_na_str(),
  nested = TRUE,
  ...,
  table_names = var,
  .stats = "count_fraction",
  .stat_names = NULL,
  .formats = list(count_fraction = format_count_fraction_fixed_dp),
  .labels = NULL,
  .indent_mods = NULL
)

summarize_occurrences_by_grade(
  lyt,
  var,
  id = "USUBJID",
  grade_groups = list(),
  remove_single = TRUE,
  only_grade_groups = FALSE,
  riskdiff = FALSE,
  na_str = default_na_str(),
  ...,
  .stats = "count_fraction",
  .stat_names = NULL,
  .formats = list(count_fraction = format_count_fraction_fixed_dp),
  .labels = NULL,
  .indent_mods = 0L
```

```
)

s_count_occurrences_by_grade(
  df,
  labelstr = "",
  .var,
  .N_row,
  .N_col,
  ...,
  id = "USUBJID",
  grade_groups = list(),
  remove_single = TRUE,
  only_grade_groups = FALSE,
  denom = c("N_col", "n", "N_row")
)

a_count_occurrences_by_grade(
  df,
  labelstr = "",
  ...,
  .stats = NULL,
  .stat_names = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```

Arguments

lyt	(PreDataTableLayouts) layout that analyses will be added to.
id	(string) subject variable name.
grade_groups	(named list of character) list containing groupings of grades.
remove_single	(flag) 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. If only_grade_groups is set to TRUE this argument is ignored.
only_grade_groups	(flag) whether only the specified grade groups should be included, with individual grade rows removed (TRUE), or all grades and grade groups should be displayed (FALSE).
var_labels	(character) variable labels.
show_labels	(string) label visibility: one of "default", "visible" and "hidden".

riskdiff (flag)
whether a risk difference column is present. When set to TRUE, add_riskdiff() must be used as split_fun in the prior column split of the table layout, specifying which columns should be compared. See stat_propdiff_ci() for details on risk difference calculation.

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

nested (flag)
whether this layout instruction should 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.

... additional arguments for the lower level functions.

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

.stats (character)
statistics to select for the table.
Options are: 'count_fraction', 'count_fraction_fixed_dp'

.stat_names (character)
names of the statistics that are passed directly to name single statistics (.stats). This option is visible when producing rtables::as_result_df() with make_ard = TRUE.

.formats (named character or list)
formats for the statistics. See Details in analyze_vars for more information on the "auto" setting.

.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.

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

labelstr (string)
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.

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

.N_row (integer(1))
row-wise N (row group count) for the group of observations being analyzed (i.e. with no column-based subsetting) that is typically passed by rtables.

.N_col (integer(1))
column-wise N (column count) for the full column being analyzed that is typically passed by rtables.

denom (string)
choice of denominator for proportion. Options are:

- N_col: total number of patients in this column across rows.
- n: number of patients with any occurrences.
- N_row: total number of patients in this row across columns.

Value

- 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 rtables::build_table(). Adding this function to an rtable layout will add formatted content rows containing the statistics from s_count_occurrences_by_grade() to the table layout.
- 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().

Functions

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

See Also

Relevant helper function h_append_grade_groups().

Examples

```
library(dplyr)

df <- data.frame(
  USUBJID = as.character(c(1:6, 1)),
  ARM = factor(c("A", "A", "A", "B", "B", "B", "A"), levels = c("A", "B")),
  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_adsl <- df |>
  select(USUBJID, ARM) |>
  unique()

# 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")
)

basic_table() |>
  split_cols_by("ARM") |>
  add_colcounts() |>
  count_occurrences_by_grade(
    var = "AETOXGR",
    grade_groups = grade_groups,
    only_grade_groups = TRUE
  ) |>
  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)

basic_table() |>
  add_colcounts() |>
  split_rows_by("ARM", child_labels = "visible", nested = TRUE) |>
  summarize_occurrences_by_grade(
    var = "AETOXGR",
    grade_groups = grade_groups
  ) |>
  build_table(df, alt_counts_df = df_adsl)
```

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

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

`count_patients_with_event` *Count the number of patients with a particular event*

Description

[Stable]

The analyze function `count_patients_with_event()` creates a layout element to calculate patient counts for a user-specified set of events.

This function analyzes primary analysis variable `vars` which indicates unique subject identifiers. Events are defined by the user as a named vector via the `filters` argument, where each name corresponds to a variable and each value is the value(s) that that variable takes for the event.

If there are multiple records with the same event recorded for a patient, only one occurrence is counted.

Usage

```
count_patients_with_event(
  lyt,
  vars,
  filters,
  riskdiff = FALSE,
  na_str = default_na_str(),
  nested = TRUE,
  show_labels = ifelse(length(vars) > 1, "visible", "hidden"),
  ...,
  table_names = vars,
  .stats = "count_fraction",
```

```
  .stat_names = NULL,
  .formats = list(count_fraction = format_count_fraction_fixed_dp),
  .labels = NULL,
  .indent_mods = NULL
)

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

a_count_patients_with_event(
  df,
  labelstr = "",
  ...,
  .stats = NULL,
  .stat_names = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```

Arguments

lyt	(PreDataTableLayouts) layout that analyses will be added to.
vars	(character) variable names for the primary analysis variable to be iterated over.
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.
riskdiff	(flag) whether a risk difference column is present. When set to TRUE, add_riskdiff() must be used as split_fun in the prior column split of the table layout, specifying which columns should be compared. See stat_propdiff_ci() for details on risk difference calculation.
na_str	(string) string used to replace all NA or empty values in the output.
nested	(flag) whether this layout instruction should 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.

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 the case that the same vars are analyzed multiple times, to avoid warnings from rtables.

.stats (character)
statistics to select for the table.
Options are: 'n', 'count', 'count_fraction', 'count_fraction_fixed_dp', 'n_blq'

.stat_names (character)
names of the statistics that are passed directly to name single statistics (.stats). This option is visible when producing rtables::as_result_df() with make_ard = TRUE.

.formats (named character or list)
formats for the statistics. See Details in analyze_vars for more information on the "auto" setting.

.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.

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

.var (string)
name of the column that contains the unique identifier.

.N_col (integer(1))
column-wise N (column count) for the full column being analyzed that is typically passed by rtables.

.N_row (integer(1))
row-wise N (row group count) for the group of observations being analyzed (i.e. with no column-based subsetting) that is typically 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.

labelstr (string)
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.

Value

- `count_patients_with_event()` returns a layout object suitable for passing to further laying out 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.
- `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()`.

Functions

- `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()`.
- `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()`.

See Also

`count_patients_with_flags()`

Examples

```
lyt <- basic_table() |>
  split_cols_by("ARM") |>
  add_colcounts() |>
  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", "AEOUT" = "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", "AEOUT" = "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)

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

`count_patients_with_flags` *Count the number of patients with particular flags*

Description

[Stable]

The analyze function `count_patients_with_flags()` creates a layout element to calculate counts of patients for which user-specified flags are present.

This function analyzes primary analysis variable `var` which indicates unique subject identifiers. Flags variables to analyze are specified by the user via the `flag_variables` argument, and must either take value `TRUE` (flag present) or `FALSE` (flag absent) for each record.

If there are multiple records with the same flag present for a patient, only one occurrence is counted.

Usage

```
count_patients_with_flags(
  lyt,
  var,
  flag_variables,
  flag_labels = NULL,
  var_labels = var,
  show_labels = "hidden",
  riskdiff = FALSE,
  na_str = default_na_str(),
  nested = TRUE,
  ...,
  table_names = paste0("tbl_flags_", var),
  .stats = "count_fraction",
  .stat_names = NULL,
  .formats = list(count_fraction = format_count_fraction_fixed_dp),
  .indent_mods = NULL,
  .labels = NULL
)

s_count_patients_with_flags(
  df,
  .var,
  .N_col = ncol(df),
  .N_row = nrow(df),
  ...,
  flag_variables,
  flag_labels = NULL,
  denom = c("n", "N_col", "N_row")
)

a_count_patients_with_flags(
  df,
  labelstr = "",
  ...,
  .stats = NULL,
  .stat_names = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```

Arguments

lyt	(PreDataTableLayouts) layout that analyses will be added to.
var	(string) single variable name that is passed by rtables when requested by a statistics

function.

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

flag_labels (character)
vector of labels to use for flag variables. If any labels are also specified via the .labels parameter, the .labels values will take precedence and replace these labels.

var_labels (character)
variable labels.

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

riskdiff (flag)
whether a risk difference column is present. When set to TRUE, add_riskdiff() must be used as split_fun in the prior column split of the table layout, specifying which columns should be compared. See stat_propdiff_ci() for details on risk difference calculation.

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

nested (flag)
whether this layout instruction should 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.

... additional arguments for the lower level functions.

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

.stats (character)
statistics to select for the table.
Options are: 'n', 'count', 'count_fraction', 'count_fraction_fixed_dp', 'n_blq'

.stat_names (character)
names of the statistics that are passed directly to name single statistics (.stats). This option is visible when producing rtables::as_result_df() with make_ard = TRUE.

.formats (named character or list)
formats for the statistics. See Details in analyze_vars for more information on the "auto" setting.

.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).

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

`.var` (string)
name of the column that contains the unique identifier.

`.N_col` (integer(1))
column-wise N (column count) for the full column being analyzed that is typically passed by `rtables`.

`.N_row` (integer(1))
row-wise N (row group count) for the group of observations being analyzed (i.e. with no column-based subsetting) that is typically 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.

`labelstr` (string)
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.

Value

- `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.
- `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()`.

Functions

- `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()`.
- `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()`.

Note

If `flag_labels` is not specified, variables labels will be extracted from `df`. If variables are not labeled, variable names will be used instead. Alternatively, a named `vector` can be supplied to `flag_variables` such that within each name-value pair the name corresponds to the variable name and the value is the label to use for this variable.

See Also

count_patients_with_event

Examples

```
# Add labelled flag variables to analysis dataset.
adae <- tern_ex_adae |>
  dplyr::mutate(
    fl1 = TRUE |> with_label("Total AEs"),
    fl2 = (TRTEMFL == "Y") |>
      with_label("Total number of patients with at least one adverse event"),
    fl3 = (TRTEMFL == "Y" & AEOUT == "FATAL") |>
      with_label("Total number of patients with fatal AEs"),
    fl4 = (TRTEMFL == "Y" & AEOUT == "FATAL" & AEREL == "Y") |>
      with_label("Total number of patients with related fatal AEs")
  )

lyt <- basic_table() |>
  split_cols_by("ARM") |>
  add_colcounts() |>
  count_patients_with_flags(
    "SUBJID",
    flag_variables = c("fl1", "fl2", "fl3", "fl4"),
    denom = "N_col"
  )

build_table(lyt, adae, alt_counts_df = tern_ex_adsl)

# `s_count_patients_with_flags()`

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

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

`count_values` *Count specific values*

Description

[Stable]

The analyze function `count_values()` creates a layout element to calculate counts of specific values within a variable of interest.

This function analyzes one or more variables of interest supplied as a vector to `vars`. Values to count for variable(s) in `vars` can be given as a vector via the `values` argument. One row of counts will be generated for each variable.

Usage

```
count_values(
  lyt,
  vars,
  values,
  na_str = default_na_str(),
  na_rm = TRUE,
  nested = TRUE,
  ...,
  table_names = vars,
  .stats = "count_fraction",
  .stat_names = NULL,
  .formats = c(count_fraction = "xx (xx.xx%)", count = "xx"),
  .labels = c(count_fraction = paste(values, collapse = ", ")),
  .indent_mods = NULL
)

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

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

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

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

a_count_values(
  x,
  ...,
  .stats = NULL,
  .stat_names = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```

Arguments

lyt	(PreDataTableLayouts) layout that analyses will be added to.
vars	(character) variable names for the primary analysis variable to be iterated over.
values	(character) specific values that should be counted.
na_str	(string) string used to replace all NA or empty values in the output.
na_rm	(flag) whether NA values should be removed from x prior to analysis.
nested	(flag) whether this layout instruction should 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.
...	additional arguments for the lower level functions.
table_names	(character) this can be customized in the case that the same vars are analyzed multiple times, to avoid warnings from rtables.
.stats	(character) statistics to select for the table. Options are: 'n', 'count', 'count_fraction', 'count_fraction_fixed_dp', 'n_blq'
.stat_names	(character) names of the statistics that are passed directly to name single statistics (.stats). This option is visible when producing rtables::as_result_df() with make_ard = TRUE.
.formats	(named character or list) formats for the statistics. See Details in analyze_vars for more information on the "auto" setting.
.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.
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_row: total number of values in this row across columns. • N_col: total number of values in this column across rows.

Value

- `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.
- `s_count_values()` returns output of `s_summary()` for specified values of a non-numeric variable.
- `a_count_values()` returns the corresponding list with formatted `rtables::CellValue()`.

Functions

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

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

```
# `count_values`
basic_table() |>
  count_values("Species", values = "setosa") |>
  build_table(iris)

# `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)
```

cox_regression *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

```
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_str = "",
  .section_div = NA_character_
)

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_str = "",
  cache_env = NULL
)
```

Arguments

lyt	(PreDataTableLayouts) layout that analyses will be added to.
variables	(named list of string) list of additional analysis variables.
control	(list) a list of parameters as returned by the helper function control_coxreg().
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.
multivar	(flag) whether multivariate Cox regression should run (defaults to FALSE), otherwise univariate Cox regression will run.
common_var	(string) 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.
.stats	(character) the names 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)
.formats	(named character or list) formats for the statistics. See Details in analyze_vars for more information on the "auto" setting.
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.
.indent_mods	(named integer) indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.
na_str	(string) custom string to replace all NA values with. Defaults to "".
.section_div	(string or NA) 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.

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

`.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` (`string`)
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`.

`.spl_context` (`data.frame`)
gives information about ancestor split states that is passed by `rtables`.

`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).

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

- `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.
- `s_coxreg()` returns the selected statistic for from the Cox regression model for the selected variable(s).

- a_coxreg() returns formatted rtables::CellValue().

Functions

- 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().
- 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().

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

```
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")

u1_variables <- list(
  time = "TIME", event = "STATUS", arm = "ARM", covariates = c("COVAR1", "COVAR2")
)

u2_variables <- list(time = "TIME", event = "STATUS", covariates = c("COVAR1", "COVAR2"))

m1_variables <- list(
  time = "TIME", event = "STATUS", arm = "ARM", covariates = c("COVAR1", "COVAR2")
)
```

```
m2_variables <- list(time = "TIME", event = "STATUS", covariates = c("COVAR1", "COVAR2"))

# summarize_coxreg

result_univar <- basic_table() |>
  summarize_coxreg(variables = u1_variables) |>
  build_table(dta_bladder)
result_univar

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

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

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

# s_coxreg

# Univariate
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
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.
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")
s_coxreg(
  model_df = df2, .stats = "pval", .which_vars = "multi_lvl",
  .var_nms = c("COVAR1", "A Covariate Label")
)

# Multivariate without treatment arm - only "COVAR1" main effect
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"
)
```

`cox_regression_inter` *Cox regression helper function for 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

```
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().
...	see methods.
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 as 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 ~ Arm + Sex + Arm:Sex. The cox regression estimates the coefficients along with a variance-covariance matrix for:

- b1 (arm b), b2 (arm c)
- b3 (sex m)
- b4 (arm b: sex m), b5 (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(b2 + b3 + b5)/ exp(b3) = exp(b2 + b5). The interaction coefficient is deduced by b2 + b5 while the standard error is obtained as $sqrt(Var b2 + Var b5 + 2 * covariance (b2,b5))$.

Value

- `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`.
- `h_coxreg_extract_interaction()` returns the result of an interaction test and the estimated values. If no interaction, `h_coxreg_univar_extract()` is applied instead.
- `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

- `h_coxreg_inter_effect()`: S3 generic helper function to determine interaction effect.
- `h_coxreg_inter_effect(numeric)`: Method for `numeric` class. Estimates the interaction with a `numeric` covariate.
- `h_coxreg_inter_effect(factor)`: Method for `factor` class. Estimate the interaction with a `factor` covariate.
- `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.
- `h_coxreg_extract_interaction()`: A higher level function to get the results of the interaction test and the estimated values.
- `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

```
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
```

`cut_quantile_bins` *Cut numeric vector into empirical quantile bins*

Description

[Stable]

This cuts a numeric vector into sample quantile bins.

Usage

```
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`	(`numeric`) 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(1)`) type of quantiles to use, see `stats::quantile()` for details.
`ordered`	(`flag`) should the result be an ordered factor.

Value

- `cut_quantile_bins`: 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 `[-Inf, q1]` where `q1` is the first quantile, the second bin is then `(q1, q2]`, etc., and the last bin is `(qn, +Inf]` where `qn` is the last quantile.

Examples

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

# Use custom quantiles.
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(1)) time in days.

Value

A numeric vector with the time in months.

Examples

```
x <- c(403, 248, 30, 86)
day2month(x)
```

decorate_grob *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"),
  width_footnotes = grid::unit(1, "npc"),
  border = TRUE,
  padding = grid::unit(rep(1, 4), "lines"),
  margins = grid::unit(c(1, 0, 1, 0), "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	(grob) a grid grob object, optionally NULL if only a grob with the decoration should be shown.
titles	(character) titles given as a vector of strings that are each separated by a newline and wrapped according to the page width.
footnotes	(character) footnotes. Uses the same formatting rules as titles.
page	(string or NULL) page numeration. If NULL then no page number is displayed.
width_titles	(grid::unit) width of titles. Usually defined as all the available space grid::unit(1, "npc"), it is affected by the parameter outer_margins. Right margins (outer_margins[4]) need to be subtracted to the allowed width.

width_footnotes
: (grid::unit)
width of footnotes. Same default and margin correction as width_titles.

border
: (flag)
whether a border should be drawn around the plot or not.

padding
: (grid::unit)
padding. A unit object of length 4. Innermost margin between the plot (grob) and, possibly, the border of the plot. Usually expressed in 4 identical values (usually "lines"). It defaults to grid::unit(rep(1, 4), "lines").

margins
: (grid::unit)
margins. A unit object of length 4. Margins between the plot and the other elements in the list (e.g. titles, plot, and footers). This is usually expressed in 4 "lines", where the lateral ones are 0s, while top and bottom are 1s. It defaults to grid::unit(c(1, 0, 1, 0), "lines").

outer_margins
: (grid::unit)
outer margins. A unit object of length 4. It defines the general margin of the plot, considering also decorations like titles, footnotes, and page numbers. It defaults to grid::unit(c(2, 1.5, 3, 1.5), "cm").

gp_titles
: (gpar)
a gpar object. Mainly used to set different "fontsize".

gp_footnotes
: (gpar)
a gpar object. Mainly used to set different "fontsize".

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

```
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.",
  paste(
    "iris is a data frame with 150 cases (rows) and 5 variables (columns) named",
    "Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, and Species."
  )
)

## 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,
    page = "Page 6 of 129"
  )
)

# with gridExtra - no borders
library(gridExtra)
grid.newpage()
grid.draw(
  decorate_grob(
    tableGrob(
      head(mtcars)
    ),
    titles = "title",
    footnotes = "footnote",
    border = FALSE
  )
)
```

`decorate_grob_set` *Decorate set of* `grob`*s and add page numbering*

Description

[Stable]

Note that this uses the `decorate_grob_factory()` function.

Usage

```
decorate_grob_set(grobs, ...)
```

Arguments

grobs	(list of grob) 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()
    ),
    ggplot2::ggplotGrob(
      ggplot2::ggplot(mapping = aes(Sepal.Length, Petal.Length, col = Species)) +
        ggplot2::geom_point()
    ),
    ggplot2::ggplotGrob(
      ggplot2::ggplot(mapping = aes(Sepal.Length, Petal.Width, col = Species)) +
        ggplot2::geom_point()
    ),
    ggplot2::ggplotGrob(
      ggplot2::ggplot(mapping = aes(Sepal.Width, Petal.Length, col = Species)) +
        ggplot2::geom_point()
    ),
    ggplot2::ggplotGrob(
      ggplot2::ggplot(mapping = aes(Sepal.Width, Petal.Width, col = Species)) +
        ggplot2::geom_point()
    ),
    ggplot2::ggplotGrob(
      ggplot2::ggplot(mapping = aes(Petal.Length, Petal.Width, col = Species)) +
        ggplot2::geom_point()
    )
  )
})
lg <- decorate_grob_set(grobs = g, titles = "Hello\nOne\nTwo\nThree", footnotes = "")

draw_grob(lg[[1]])
draw_grob(lg[[2]])
draw_grob(lg[[6]])
```

default_na_str *Default string replacement for* NA *values*

Description

[Stable]

The default string used to represent NA values. This value is used as the default value for the na_str argument throughout the tern package, and printed in place of NA values in output tables. If not specified for each tern function by the user via the na_str argument, or in the R environment options via set_default_na_str(), then NA is used.

Usage

```
default_na_str()

set_default_na_str(na_str)
```

Arguments

na_str	(string) single string value to set in the R environment options as the default value to replace NAs. Use getOption("tern_default_na_str") to check the current value set in the R environment (defaults to NULL if not set).

Value

- default_na_str returns the current value if an R environment option has been set for "tern_default_na_str", or NA_character_ otherwise.
- set_default_na_str has no return value.

Functions

- default_na_str(): Accessor for default NA value replacement string.
- set_default_na_str(): Setter for default NA value replacement string. Sets the option "tern_default_na_str" within the R environment.

Examples

```
# Default settings
default_na_str()
getOption("tern_default_na_str")

# Set custom value
set_default_na_str("<Missing>")

# Settings after value has been set
default_na_str()
```

```
getOption("tern_default_na_str")
```

`default_stats_formats_labels` *Get default statistical methods and their associated formats, labels, and indent modifiers*

Description

[Experimental]

Utility functions to get valid statistic methods for different method groups (`.stats`) and their associated formats (`.formats`), labels (`.labels`), and indent modifiers (`.indent_mods`). This utility is used across `tern`, but some of its working principles can be seen in `analyze_vars()`. See notes to understand why this is experimental.

Usage

```
get_stats(
  method_groups = "analyze_vars_numeric",
  stats_in = NULL,
  custom_stats_in = NULL,
  add_pval = FALSE
)

get_stat_names(stat_results, stat_names_in = NULL)

get_formats_from_stats(
  stats,
  formats_in = NULL,
  levels_per_stats = NULL,
  tern_defaults = tern_default_formats
)

get_labels_from_stats(
  stats,
  labels_in = NULL,
  levels_per_stats = NULL,
  label_attr_from_stats = NULL,
  tern_defaults = tern_default_labels
)

get_indents_from_stats(
  stats,
  indents_in = NULL,
  levels_per_stats = NULL,
  tern_defaults = setNames(as.list(rep(0L, length(stats))), stats),
```

```
  row_nms = lifecycle::deprecated()
)

tern_default_stats

tern_default_formats

tern_default_labels

summary_formats(type = "numeric", include_pval = FALSE)

summary_labels(type = "numeric", include_pval = FALSE)
```

Arguments

method_groups	(character) indicates the statistical method group (tern analyze function) to retrieve default statistics for. A character vector can be used to specify more than one statistical method group.
stats_in	(character) statistics to retrieve for the selected method group. If custom statistical functions are used, stats_in needs to have them in too.
custom_stats_in	(character) custom statistics to add to the default statistics.
add_pval	(flag) should "pval" (or "pval_counts" if method_groups contains "analyze_vars_counts") be added to the statistical methods?
stat_results	(list) list of statistical results. It should be used close to the end of a statistical function. See examples for a structure with two statistical results and two groups.
stat_names_in	(character) custom modification of statistical values.
stats	(character) statistical methods to return defaults for.
formats_in	(named vector) custom formats to use instead of defaults. Can be a character vector with values from formatters::list_valid_format_labels() or custom format functions. Defaults to NULL for any rows with no value is provided. See Details.
levels_per_stats	(named list of character or NULL) named list where the name of each element is a statistic from stats and each element is the levels of a factor or character variable (or variable name), each corresponding to a single row, for which the named statistic should be calculated for. If a statistic is only calculated once (one row), the element can be either NULL or the name of the statistic. Each list element will be flattened such that the names of the list elements returned by the function have the format

statistic.level (or just statistic for statistics calculated for a single row). Defaults to NULL.

tern_defaults (list or vector)
defaults to use to fill in missing values if no user input is given. Must be of the same type as the values that are being filled in (e.g. indentation must be integers).

labels_in (named character)
custom labels to use instead of defaults. If no value is provided, the variable level (if rows correspond to levels of a variable) or statistic name will be used as label.

label_attr_from_stats
(named list)
if labels_in = NULL, then this will be used instead. It is a list of values defined in statistical functions as default labels. Values are ignored if labels_in is provided or "" values are provided.

indents_in (named integer)
custom row indent modifiers to use instead of defaults. Defaults to 0L for all values.

row_nms **[Deprecated]** Deprecation cycle started. See the levels_per_stats parameter for details.

type (string)
"numeric" or "counts".

include_pval (flag)
same as the add_pval argument in get_stats().

Format

- tern_default_stats is a named list of available statistics, with each element named for their corresponding statistical method group.
- tern_default_formats is a named vector of available default formats, with each element named for their corresponding statistic.
- tern_default_labels is a named character vector of available default labels, with each element named for their corresponding statistic.

Details

Current choices for type are counts and numeric for analyze_vars() and affect get_stats().

if formats_in is "default", instead of populating the return value with tern defaults, the return value will specify the "default" format for each element. This is useful primarily when formatting behavior should be inherited from a format specified via the format or formats_var argument to analyze.
summary_* quick get functions for labels or formats uses get_stats and get_labels_from_stats or get_formats_from_stats respectively to retrieve relevant information.

Value

- `get_stats()` returns a `character` vector of statistical methods.
- `get_stat_names()` returns a named list of `character` vectors, indicating the names of statistical outputs.
- `get_formats_from_stats()` returns a named list of formats as strings or functions.
- `get_labels_from_stats()` returns a named list of labels as strings.
- `get_indents_from_stats()` returns a named list of indentation modifiers as integers.
- `summary_formats()` returns a named `vector` of default statistic formats for the given data type.
- `summary_labels` returns a named `vector` of default statistic labels for the given data type.

Functions

- `get_stats()`: Get statistics available for a given method group (analyze function). To check available defaults see `tern::tern_default_stats` list.
- `get_stat_names()`: Get statistical *names* available for a given method group (analyze function). Please use the `s_*` functions to get the statistical names.
- `get_formats_from_stats()`: Get formats corresponding to a list of statistics. To check available defaults see list `tern::tern_default_formats`.
- `get_labels_from_stats()`: Get labels corresponding to a list of statistics. To check for available defaults see list `tern::tern_default_labels`.
- `get_indents_from_stats()`: Get row indent modifiers corresponding to a list of statistics/rows.
- `tern_default_stats`: Named list of available statistics by method group for `tern`.
- `tern_default_formats`: Named vector of default formats for `tern`.
- `tern_default_labels`: Named `character` vector of default labels for `tern`.
- `summary_formats()`: Quick function to retrieve default formats for summary statistics: `analyze_vars()` and `analyze_vars_in_cols()` principally.
- `summary_labels()`: Quick function to retrieve default labels for summary statistics. Returns labels of descriptive statistics which are understood by `rtables`. Similar to `summary_formats`.

Note

These defaults are experimental because we use the names of functions to retrieve the default statistics. This should be generalized in groups of methods according to more reasonable groupings.

Formats in `tern` and `rtables` can be functions that take in the table cell value and return a string. This is well documented in `vignette("custom_appearance", package = "rtables")`.

See Also

formatting_functions

Examples

```
# analyze_vars is numeric
num_stats <- get_stats("analyze_vars_numeric") # also the default

# Other type
cnt_stats <- get_stats("analyze_vars_counts")

# Weirdly taking the pval from count_occurrences
only_pval <- get_stats("count_occurrences", add_pval = TRUE, stats_in = "pval")

# All count_occurrences
all_cnt_occ <- get_stats("count_occurrences")

# Multiple
get_stats(c("count_occurrences", "analyze_vars_counts"))

stat_results <- list(
  "n" = list("M" = 1, "F" = 2),
  "count_fraction" = list("M" = c(1, 0.2), "F" = c(2, 0.1))
)
get_stat_names(stat_results)
get_stat_names(stat_results, list("n" = "argh"))

# Defaults formats
get_formats_from_stats(num_stats)
get_formats_from_stats(cnt_stats)
get_formats_from_stats(only_pval)
get_formats_from_stats(all_cnt_occ)

# Addition of customs
get_formats_from_stats(all_cnt_occ, formats_in = c("fraction" = c("xx")))
get_formats_from_stats(all_cnt_occ, formats_in = list("fraction" = c("xx.xx", "xx")))

# Defaults labels
get_labels_from_stats(num_stats)
get_labels_from_stats(cnt_stats)
get_labels_from_stats(only_pval)
get_labels_from_stats(all_cnt_occ)

# Addition of customs
get_labels_from_stats(all_cnt_occ, labels_in = c("fraction" = "Fraction"))
get_labels_from_stats(all_cnt_occ, labels_in = list("fraction" = c("Some more fractions")))

get_indents_from_stats(all_cnt_occ, indents_in = 3L)
get_indents_from_stats(all_cnt_occ, indents_in = list(count = 2L, count_fraction = 5L))
get_indents_from_stats(
  all_cnt_occ,
  indents_in = list(a = 2L, count.a = 1L, count.b = 5L)
)

summary_formats()
summary_formats(type = "counts", include_pval = TRUE)
```

```
summary_labels()
summary_labels(type = "counts", include_pval = TRUE)
```

`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>",
  factor_as_factor = FALSE,
  factor_level_method = c("sort_auto", "sort_radix", "data"),
  factor_level_last_pattern = NULL
)
```

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`) string used to replace all `NA` or empty values inside non-`omit_columns` columns.
`factor_as_factor`	(`flag`) whether to re-encode existing factor variables using `factor_level_method`. When `FALSE` (default), existing factor levels are preserved as-is (original behavior).

factor_level_method
: (string)
method used to order factor levels when converting character or logical variables (or existing factors when factor_as_factor = TRUE). One of:

"sort_auto" sort(unique(x)) — default R sort, locale-aware (default). Preserves the original behavior of this function.

"sort_radix" sort(unique(x), method = "radix") — byte-order (ASCII) sort. Unlike "sort_auto", this is not locale-sensitive: uppercase letters always sort before lowercase. On data where all values share the same case (e.g. all-caps ADaM variables) the two methods produce identical results.

"data" unique(x) — levels in order of first appearance in the data.

factor_level_last_pattern
: (string or NULL)
regular expression. Any factor levels matching this pattern are moved to the end (before na_level). NULL (default) disables this behaviour. Note: this parameter only takes effect when factor levels are being re-encoded (i.e. for character/logical columns with char_as_factor/logical_as_factor, or for existing factor columns with factor_as_factor = TRUE). Existing factor columns where factor_as_factor = FALSE are not affected.

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

```
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)

# Example 4: Control factor level ordering
# Use radix sort to match SAS PROC SORT behavior.
df_explicit_na(my_data, factor_level_method = "sort_radix")
# Use data order (first appearance).
df_explicit_na(my_data, factor_level_method = "data")

# Example 5: Move matching levels to the end
# Levels matching "^Other" are placed last (before na_level).
df_explicit_na(my_data, factor_level_last_pattern = "^Other")
```

---

draw_grob *Draw* grob

Description

[Deprecated]

Draw grob on device page.

Usage

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

Arguments

grob	(grob) grid object.
newpage	(flag) draw on a new page.
vp	(viewport or NULL) a viewport() object (or NULL).

Value

A grob.

Examples

```
library(dplyr)
library(grid)


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

num <- lapply(1:10, textGrob)
num |>
  (\(x) arrange_grobs(grobs = x))() |>
  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

```
d_count_abnormal_by_baseline(abnormal)
```

Arguments

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

Value

Abnormal category labels for s_count_abnormal_by_baseline().

Examples

```
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 = TRUE, include_eq = TRUE)
```

Arguments

threshold	(numeric(1)) a cutoff value as threshold to count values of x.
lower_tail	(flag) whether to count lower tail, default is TRUE.
include_eq	(flag) whether to include value equal to the threshold in count, default is TRUE.

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

thresholds	(numeric) minimum number of missed doses the patients had.

Value

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

See Also

s_count_missed_doses()

---

d_onco_rsp_label *Description of standard oncology response*

Description

[Stable]

Describe the oncology response in a standard way.

Usage

```
d_onco_rsp_label(x)
```

Arguments

x	(character) the standard oncology codes to be described.

Value

Response labels.

See Also

estimate_multinomial_rsp()

Examples

```
d_onco_rsp_label(
  c("CR", "PR", "SD", "NON CR/PD", "PD", "NE", "Missing", "<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

A `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.
`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`.
`long`	(`flag`) whether a long or a short (default) description is required.

Value

String describing the analysis.

d_proportion_diff *Description of method used for proportion comparison*

Description

[Stable]

This is an auxiliary function that describes the analysis in `s_proportion_diff()`.

Usage

```
d_proportion_diff(conf_level, method, long = FALSE)
```

Arguments

`conf_level`	(`proportion`) confidence level of the interval.
`method`	(`string`) the method used for the confidence interval estimation.
`long`	(`flag`) whether a long (`TRUE`) or a short (`FALSE`, default) description is required.

Value

A `string` describing the analysis.

See Also

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 `tabulate_rsp_subgroups()` and create column labels.

Usage

```
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 or NULL) specifies the test used to calculate the p-value for the difference between two proportions. For options, see 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 names 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.

method	(string) p-value method for testing hazard ratio = 1.
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.

---

d_test_proportion_diff *Description of the difference test between two proportions*

Description

[Stable]

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

Usage

```
d_test_proportion_diff(method, alternative = c("two.sided", "less", "greater"))
```

Arguments

method	(string) one of chisq, cmh, cmh_sato, cmh_wh, fisher, or schouten; specifies the test used to calculate the p-value.
alternative	(string) whether two.sided, or one-sided less or greater p-value should be displayed.

Value

A string describing the test from which the p-value is derived.

estimate_multinomial_rsp *Estimate proportions of each level of a variable*

Description

[Stable]

The analyze & summarize function `estimate_multinomial_response()` creates a layout element to estimate the proportion and proportion confidence interval for each level of a factor variable. The primary analysis variable, `var`, should be a factor variable, the values of which will be used as labels within the output table.

Usage

```
estimate_multinomial_response(
  lyt,
  var,
  na_str = default_na_str(),
  nested = TRUE,
  ...,
  show_labels = "hidden",
  table_names = var,
  .stats = "prop_ci",
  .stat_names = NULL,
  .formats = list(prop_ci = "(xx.xx, xx.xx)"),
  .labels = NULL,
  .indent_mods = NULL
)

s_length_proportion(x, ..., .N_col)

a_length_proportion(
  x,
  ...,
  .stats = NULL,
  .stat_names = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```

Arguments

`lyt`	(`PreDataTableLayouts`) layout that analyses will be added to.

var	(string) single variable name that is passed by rtables when requested by a statistics function.
na_str	(string) string used to replace all NA or empty values in the output.
nested	(flag) whether this layout instruction should 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.
...	additional arguments for the lower level functions.
show_labels	(string) label visibility: one of "default", "visible" and "hidden".
table_names	(character) this can be customized in the case that the same vars are analyzed multiple times, to avoid warnings from rtables.
.stats	(character) statistics to select for the table. Options are: 'n_prop', 'prop_ci'
.stat_names	(character) names of the statistics that are passed directly to name single statistics (.stats). This option is visible when producing rtables::as_result_df() with make_ard = TRUE.
.formats	(named character or list) formats for the statistics. See Details in analyze_vars for more information on the "auto" setting.
.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.
x	(numeric) vector of numbers we want to analyze.
.N_col	(integer(1)) column-wise N (column count) for the full column being analyzed that is typically passed by rtables.

Value

- 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.
- s_length_proportion() returns statistics from s_proportion().
- a_length_proportion() returns the corresponding list with formatted rtables::CellValue().

Functions

- `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()`.
- `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()`

See Also

Relevant description function `d_onco_rsp_label()`.

Examples

```
library(dplyr)

# Use of the layout creating function.
dta_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))
) |> mutate(
  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)

tbl

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)
```

`estimate_proportion` *Proportion estimation*

Description

[Stable]

The analyze function `estimate_proportion()` creates a layout element to estimate the proportion of responders within a studied population. The primary analysis variable, `vars`, indicates whether a response has occurred for each record. See the `method` parameter for options of methods to use when constructing the confidence interval of the proportion. Additionally, a stratification variable can be supplied via the `strata` element of the `variables` argument.

Usage

```
estimate_proportion(
  lyt,
  vars,
  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,
  na_str = default_na_str(),
  nested = TRUE,
  ...,
  show_labels = "hidden",
  table_names = vars,
  .stats = c("n_prop", "prop_ci"),
  .stat_names = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)

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,
  denom = c("n", "N_col", "N_row"),
  ...
)

a_proportion(
  df,
```

```
  ...,
  .stats = NULL,
  .stat_names = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```

Arguments

lyt	(PreDataTableLayouts) layout that analyses will be added to.
vars	(character) variable names for the primary analysis variable to be iterated over.
conf_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) whether a long description is required.
na_str	(string) string used to replace all NA or empty values in the output.
nested	(flag) whether this layout instruction should 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.
...	additional arguments for the lower level functions.
show_labels	(string) label visibility: one of "default", "visible" and "hidden".
table_names	(character) this can be customized in the case that the same vars are analyzed multiple times, to avoid warnings from rtables.
.stats	(character) statistics to select for the table. Options are: 'n_prop', 'prop_ci'

`.stat_names` (character)
names of the statistics that are passed directly to name single statistics (`.stats`). This option is visible when producing `rtables::as_result_df()` with `make_ard = TRUE`.

`.formats` (named character or list)
formats for the statistics. See Details in `analyze_vars` for more information on the `"auto"` setting.

`.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.

`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.

`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.

Value

- `estimate_proportion()` 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()` to the table layout.
- `s_proportion()` returns statistics `n_prop` (`n` and proportion) and `prop_ci` (proportion CI) for a given variable.
- `a_proportion()` returns the corresponding list with formatted `rtables::CellValue()`.

Functions

- `estimate_proportion()`: Layout-creating function which can take statistics function arguments and additional format arguments. This function is a wrapper for `rtables::analyze()`.
- `s_proportion()`: Statistics function estimating a proportion along with its confidence interval.
- `a_proportion()`: Formatted analysis function which is used as `afun` in `estimate_proportion()`.

See Also

h_proportions

Examples

```
dta_test <- data.frame(
  USUBJID = paste0("S", 1:12),
  ARM = rep(LETTERS[1:3], each = 4),
  AVAL = rep(LETTERS[1:3], each = 4)
) |>
  dplyr::mutate(is_rsp = AVAL == "A")

basic_table() |>
  split_cols_by("ARM") |>
  estimate_proportion(vars = "is_rsp") |>
  build_table(df = dta_test)

# 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"
)
```

---

explicit_na *Missing data*

Description

[Stable]

Substitute missing data with a string or factor level.

Usage

```
explicit_na(x, label = default_na_str(), drop_na = default_drop_na())

default_drop_na()

set_default_drop_na(drop_na)
```

Arguments

x	(factor or character) values for which any missing values should be substituted.
label	(string) string that missing data should be replaced with.
drop_na	(flag) if TRUE and x is a factor, any levels that are only label will be dropped.

Value

x with any NA values substituted by label.

- tern_default_drop_na: (flag)
  default value for drop_na argument in explicit_na().
- tern_default_drop_na has no return value.

Functions

- default_drop_na(): should NA values without a dedicated level be dropped?
- set_default_drop_na(): Setter for default NA value replacement string. Sets the option "tern_default_drop_na" within the R environment.

Examples

```
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", "")))

explicit_na(factor(levels = c(NA, "a")))
explicit_na(factor(levels = c(NA, "a")), drop_na = TRUE) # previous default
```

`extract_rsp_biomarkers` *Prepare 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 `strata`. `groups_lists` optionally specifies groupings for `subgroups` variables.

Usage

```
extract_rsp_biomarkers(
  variables,
  data,
  groups_lists = list(),
  control = control_logistic(),
  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`	(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 == "BESRSPI") |>
  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"
  ),
  data = adrs_f
)
df

# 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 750).
df_grouped <- extract_rsp_biomarkers(
  variables = list(
    rsp = "EOSDY",
    biomarkers = c("BMRKR1", "AGE"),
    covariates = "SEX",
    subgroups = "BMRKR2",
    strata = "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 > 750)"
  )
)
df_grouped
```

`extract_rsp_subgroups` *Prepare 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 `strata`. `groups_lists` optionally specifies groupings for `subgroups` variables.

Usage

```
extract_rsp_subgroups(
  variables,
  data,
  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 or NULL) specifies the test used to calculate the p-value for the difference between two proportions. For options, see 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

---

extract_survival_biomarkers

Prepare 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 strata. groups_lists optionally specifies groupings for subgroups variables.

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().

---

extract_survival_subgroups

Prepare 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 strata. groups_lists optionally specifies groupings for subgroups variables.

Usage

```
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 the null hypothesis that 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. • alternative (string) alternative hypothesis for the p-value test. Default is "two.sided", can also be set to "less" or "greater" for one-sided testing. Note that one-sided testing is not supported when pval_method = "likelihood".
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

`extreme_format` *Format 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(1)`) number of decimal places to display.
`x`	(`numeric(1)`) 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_auto()`, `format_count_fraction()`, `format_count_fraction_fixed_dp()`, `format_count_fraction_lt10()`, `format_extreme_values()`, `format_extreme_values_ci()`, `format_fraction()`, `format_fraction_fixed_dp()`, `format_fraction_threshold()`, `format_range_cens()`, `format_sigfig()`, `format_xx()`, `formatting_functions`

Examples

```
h_get_format_threshold(2L)

h_format_threshold(0.001)
h_format_threshold(1000)
```

`ex_data` *Simulated CDISC data for examples*

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

factor_utils *Factor utilities*

Description

[Stable]

A collection of utility functions for factors.

Usage

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

as_factor_keep_attributes(
  x,
  x_name = deparse(substitute(x)),
  na_level = "<Missing>",
  verbose = TRUE
)

fct_discard(x, discard)

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

fct_collapse_only(.f, ..., .na_level = "<Missing>")
```

Arguments

x	(factor) factor variable or object to convert (for as_factor_keep_attributes).
levels	(character) level names to be combined.
new_level	(string) name of new level.
x_name	(string) name of x.
na_level	(string) which level to use for missing values.
verbose	(flag) defaults to TRUE. It prints out warnings and messages.
discard	(character) levels to discard.
condition	(logical) positions at which to insert missing values.
.f	(factor or character) original vector.

`...`	(named `character`) levels in each vector provided will be collapsed into the new level given by the respective name.
`.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 `...`.

Value

- `combine_levels`: A `factor` with the new levels.
- `as_factor_keep_attributes`: A `factor` with same attributes (except class) as `x`. Does not modify `x` if already a `factor`.
- `fct_discard`: A modified `factor` with observations as well as levels from `discard` dropped.
- `fct_explicit_na_if`: A modified `factor` with inserted and existing `NA` converted to `na_level`.
- `fct_collapse_only`: 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 `.na_level`.

Functions

- `combine_levels()`: Combine specified old factor Levels in a single new level.
- `as_factor_keep_attributes()`: Converts `x` to a factor and keeps its attributes. Warns appropriately such that the user can decide whether they prefer converting to factor manually (e.g. for full control of factor levels).
- `fct_discard()`: This discards the observations as well as the levels specified from a factor.
- `fct_explicit_na_if()`: This inserts explicit missing values in a factor based on a condition. Additionally, existing `NA` values will be explicitly converted to given `na_level`.
- `fct_collapse_only()`: 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).

Note

Any existing `NA`s in the input vector will not be replaced by the missing level. If needed, `explicit_na()` can be called separately on the result.

See Also

`cut_quantile_bins()` for splitting numeric vectors into quantile bins.

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

`forcats::fct_collapse()`, `forcats::fct_relevel()` which are used internally.

Examples

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

combine_levels(x, c("e", "b"))

a_chr_with_labels <- c("a", "b", NA)
attr(a_chr_with_labels, "label") <- "A character vector with labels"
as_factor_keep_attributes(a_chr_with_labels)

fct_discard(factor(c("a", "b", "c")), "c")

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

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

`fit_coxreg` *Fitting functions 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`	(named `list`) the names of the variables found in `data`, passed as a named list and corresponding to the `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

```
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")
  ),
  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

```
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

```
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")
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"
  )
)
```

`fit_rsp_step` *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:

`response ~ arm * poly(biomarker, degree) + covariates + strata(strata)`

where `degree` is specified by `control_step()`.

Usage

```
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

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

adrs_f <- tern_ex_adrs |>
  filter(
    PARAMCD == "BESRSPI",
    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 = 0.9))
)
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,
  data = adrs_f,
 control = c(control_logistic(conf_level = 0.9), control_step(bandwidth = NULL, 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 = NULL))
)
```

`fit_survival_step` *Subgroup treatment effect pattern (STEP) fit for survival outcome*

Description

[Stable]

This fits the subgroup treatment effect pattern (STEP) 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:

`Surv(time, event) ~ arm * poly(biomarker, degree) + covariates + strata(strata)`

where degree is specified by `control_step()`.

Usage

```
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

```
# 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

[Deprecated]

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`	(`VTableTree`) `rtables` table object.
`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`	(`MatrixPrintForm`) 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

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

Details

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_auto()`, `format_count_fraction()`, `format_count_fraction_fi`
`format_count_fraction_lt10()`, `format_extreme_values()`, `format_extreme_values_ci()`,
`format_fraction()`, `format_fraction_fixed_dp()`, `format_fraction_threshold()`, `format_range_cens()`,
`format_sigfig()`, `format_xx()`

`format_auto` *Format automatically using data significant digits*

Description

[Stable]

Formatting function for the majority of default methods used in `analyze_vars()`. For non-derived values, the significant digits of data is used (e.g. range), while derived values have one more digits (measure of location and dispersion like mean, standard deviation). This function can be called internally with "auto" like, for example, `.formats = c("mean" = "auto")`. See details to see how this works with the inner function.

Usage

```
format_auto(dt_var, x_stat)
```

Arguments

`dt_var`	(`numeric`) variable data the statistics were calculated from. Used only to find significant digits. In analyze_vars this comes from `.df_row` (see rtables::additional_fun_params), and it is the row data after the above row splits. No column split is considered.
`x_stat`	(`string`) string indicating the current statistical method used.

Details

The internal function is needed to work with `rtables` default structure for format functions, i.e. `function(x, ...)`, where is x are results from statistical evaluation. It can be more than one element (e.g. for `.stats = "mean_sd"`).

Value

A string that `rtables` prints in a table cell.

See Also

Other formatting functions: `extreme_format`, `format_count_fraction()`, `format_count_fraction_fixed_dp()`, `format_count_fraction_lt10()`, `format_extreme_values()`, `format_extreme_values_ci()`, `format_fraction()`, `format_fraction_fixed_dp()`, `format_fraction_threshold()`, `format_range_cens()`, `format_sigfig()`, `format_xx()`, `formatting_functions`

Examples

```
x_todo <- c(0.001, 0.2, 0.0011000, 3, 4)
res <- c(mean(x_todo[1:3]), sd(x_todo[1:3]))

# x is the result coming into the formatting function -> res!!
format_auto(dt_var = x_todo, x_stat = "mean_sd")(x = res)
format_auto(x_todo, "range")(x = range(x_todo))
no_sc_x <- c(0.0000001, 1)
format_auto(no_sc_x, "range")(x = no_sc_x)
```

---

format_count_fraction *Format 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	(numeric(2)) vector of length 2 with count and fraction, respectively.
...	not used. 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_auto()`, `format_count_fraction_fixed_dp()`, `format_count_fraction_lt10()`, `format_extreme_values()`, `format_extreme_values_ci()`, `format_fraction()`, `format_fraction_fixed_dp()`, `format_fraction_threshold()`, `format_range_cens()`, `format_sigfig()`, `format_xx()`, `formatting_functions`

Examples

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

format_count_fraction_fixed_dp

Format count and percentage with fixed single decimal place

Description

[Experimental]

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

Usage

```
format_count_fraction_fixed_dp(x, ...)
```

Arguments

`x`	(`numeric(2)`) vector of length 2 with count and fraction, respectively.
`...`	not used. 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_auto()`, `format_count_fraction()`, `format_count_fraction_lt`
`format_extreme_values()`, `format_extreme_values_ci()`, `format_fraction()`, `format_fraction_fixed_dp()`,
`format_fraction_threshold()`, `format_range_cens()`, `format_sigfig()`, `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_count_fraction_lt10

Format count and fraction with special case for count < 10

Description

[Stable]

Formats a count together with fraction with special consideration when count is less than 10.

Usage

```
format_count_fraction_lt10(x, ...)
```

Arguments

x	(numeric(2)) vector of length 2 with count and fraction, respectively.
...	not used. Required for rtables interface.

Value

A string in the format count (fraction %). If count is less than 10, only count is printed.

See Also

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

Examples

```
format_count_fraction_lt10(x = c(275, 0.9673))
format_count_fraction_lt10(x = c(2, 0.6667))
format_count_fraction_lt10(x = c(9, 1))
```

---

format_extreme_values *Format a single extreme value*

Description

[Stable]

Create a formatting function for a single extreme value.

Usage

```
format_extreme_values(digits = 2L)
```

Arguments

digits	(integer(1)) 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_auto(), format_count_fraction(), format_count_fraction_fi
format_count_fraction_lt10(), format_extreme_values_ci(), format_fraction(), format_fraction_fixed_dp()
format_fraction_threshold(), format_range_cens(), format_sigfig(), 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

Format 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(1)`) 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_auto(), format_count_fraction(), format_count_fraction_fi
format_count_fraction_lt10(), format_extreme_values(), format_fraction(), format_fraction_fixed_dp(),
format_fraction_threshold(), format_range_cens(), format_sigfig(), format_xx(), formatting_functions

Examples

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

`format_fraction` *Format fraction and percentage*

Description

[Stable]

Formats a fraction together with ratio in percent.

Usage

```
format_fraction(x, ...)
```

Arguments

`x`	(named `integer`) vector with elements `num` and `denom`.
`...`	not used. 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_auto()`, `format_count_fraction()`, `format_count_fraction_fi`
`format_count_fraction_lt10()`, `format_extreme_values()`, `format_extreme_values_ci()`,
`format_fraction_fixed_dp()`, `format_fraction_threshold()`, `format_range_cens()`, `format_sigfig()`,
`format_xx()`, `formatting_functions`

Examples

```
format_fraction(x = c(num = 2L, denom = 3L))
format_fraction(x = c(num = 0L, denom = 3L))
```

`format_fraction_fixed_dp` *Format 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`	(named `integer`) vector with elements `num` and `denom`.
`...`	not used. 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_auto()`, `format_count_fraction()`, `format_count_fraction_fi`
`format_count_fraction_lt10()`, `format_extreme_values()`, `format_extreme_values_ci()`,
`format_fraction()`, `format_fraction_threshold()`, `format_range_cens()`, `format_sigfig()`,
`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))
```

format_fraction_threshold *Format fraction with lower threshold*

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 `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: `extreme_format`, `format_auto()`, `format_count_fraction()`, `format_count_fraction_fi`
`format_count_fraction_lt10()`, `format_extreme_values()`, `format_extreme_values_ci()`,
`format_fraction()`, `format_fraction_fixed_dp()`, `format_range_cens()`, `format_sigfig()`,
`format_xx()`, `formatting_functions`

Examples

```
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_range_cens *Format range with censoring indicators*

Description

[Experimental]

Formats a survival time range where the minimum and/or maximum may be a censored observation. A + suffix is appended to a bound when the corresponding censoring flag is TRUE.

Usage

```
format_range_cens(digits = 1L)
```

Arguments

digits	(integer(1)) number of decimal places to display. Defaults to 1L.

Value

An rtables formatting function that takes a numeric(4) vector of the form c(min, max, lower_censored, upper_censored), where lower_censored and upper_censored are 0/1 (or FALSE/TRUE) flags, and returns a string in the format "min to max", with + appended to min and/or max when the corresponding censoring flag is non-zero.

See Also

Other formatting functions: extreme_format, format_auto(), format_count_fraction(), format_count_fraction_fi
format_count_fraction_lt10(), format_extreme_values(), format_extreme_values_ci(),
format_fraction(), format_fraction_fixed_dp(), format_fraction_threshold(), format_sigfig(),
format_xx(), formatting_functions

Examples

```
fmt <- format_range_cens(1L)
fmt(c(1.23, 9.87, 1, 0))
fmt(c(1.23, 9.87, 0, 0))
```

format_sigfig *Format numeric values by significant figures*

Description

Format numeric values to print with a specified number of significant figures.

Usage

```
format_sigfig(sigfig, format = "xx", num_fmt = "fg")
```

Arguments

sigfig	(integer(1)) number of significant figures to display.
format	(string) the format label (string) to apply when printing the value. Decimal places in string are ignored in favor of formatting by significant figures. Formats options are: "xx", "xx / xx", "(xx, xx)", "xx - xx", and "xx (xx)".
num_fmt	(string) numeric format modifiers to apply to the value. Defaults to "fg" for standard significant figures formatting - fixed (non-scientific notation) format ("f") and sigfig equal to number of significant figures instead of decimal places ("g"). See the formatC() format argument for more options.

Value

An rtables formatting function.

See Also

Other formatting functions: extreme_format, format_auto(), format_count_fraction(), format_count_fraction_fi
format_count_fraction_lt10(), format_extreme_values(), format_extreme_values_ci(),
format_fraction(), format_fraction_fixed_dp(), format_fraction_threshold(), format_range_cens(),
format_xx(), formatting_functions

Examples

```
fmt_3sf <- format_sigfig(3)
fmt_3sf(1.658)
fmt_3sf(1e1)

fmt_5sf <- format_sigfig(5)
fmt_5sf(0.57)
fmt_5sf(0.000025645)
```

format_xx *Format XX as a 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) template.

Value

An rtables formatting function.

See Also

Other formatting functions: extreme_format, format_auto(), format_count_fraction(), format_count_fraction_fi
format_count_fraction_lt10(), format_extreme_values(), format_extreme_values_ci(),
format_fraction(), format_fraction_fixed_dp(), format_fraction_threshold(), format_range_cens(),
format_sigfig(), formatting_functions

Examples

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

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

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

z <- format_xx("xx.x, incl. xx.x% NE")
sapply(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	(numeric(1)) mean value to test under the null hypothesis.

Value

A string.

get_covariates *Utility function to return a named list of covariate names*

Description

[Stable]

Usage

```
get_covariates(covariates)
```

Arguments

covariates	(character) a vector that can contain single variable names (such as "X1"), and/or interaction terms indicated by "X1 * X2".

Value

A named list of character vector.

Examples

```
get_covariates(c("a * b", "c"))
```

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	(string) x column name.
y	(string) y column name.
groups	(character or NULL) vector with optional grouping variables names.
level	(proportion) 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 a 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.

Value

A tibble in the required format.

Examples

```
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_bland_altman *Bland-Altman plot*

Description

[Experimental]

Graphing function that produces a Bland-Altman plot.

Usage

```
g_bland_altman(x, y, conf_level = 0.95)
```

Arguments

x	(numeric) vector of numbers we want to analyze.
y	(numeric) vector of numbers we want to analyze, to be compared with x.
conf_level	(proportion) confidence level of the interval.

Value

A ggplot Bland-Altman plot.

Examples

```
x <- seq(1, 60, 5)
y <- seq(5, 50, 4)

g_bland_altman(x = x, y = y, conf_level = 0.9)
```

g_forest *Create a forest plot from an* rtable

Description

[Stable]

Usage

```
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 = lifecycle::deprecated(),
  width_columns = NULL,
  width_forest = lifecycle::deprecated(),
  lbl_col_padding = 0,
  rel_width_forest = 0.25,
  font_size = 12,
  col_symbol_size = attr(tbl, "col_symbol_size"),
  col = getOption("ggplot2.discrete.colour")[1],
  ggtheme = NULL,
  as_list = FALSE,
  gp = lifecycle::deprecated(),
  draw = lifecycle::deprecated(),
  newpage = lifecycle::deprecated()
)
```

Arguments

tbl	(VTableTree) rtables table with at least one column with a single value and one column with 2 values.
col_x	(integer(1) or NULL) column index with estimator. By default tries to get this from tbl attribute col_x, otherwise needs to be manually specified. If NULL, points will be excluded from forest plot.
col_ci	(integer(1) or NULL) column index with confidence intervals. By default tries to get this from tbl attribute col_ci, otherwise needs to be manually specified. If NULL, lines will be excluded from forest plot.
vline	(numeric(1) or NULL) x coordinate for vertical line, if NULL then the line is omitted.
forest_header	(character(2)) text displayed to the left and right of vline, respectively. If vline = NULL then forest_header is not printed. By default tries to get this from tbl attribute forest_header. If NULL, defaults will be extracted from the table if possible, and set to "Comparison\nBetter" and "Treatment\nBetter" if not.
xlim	(numeric(2)) limits for x axis.

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

`x_at` (numeric)
x-tick locations, if NULL, `x_at` is set to `vline` and both `xlim` values.

`width_row_names`
[Deprecated] Please use the `lbl_col_padding` argument instead.

`width_columns` (numeric)
a vector of column widths. Each element's position in `colwidths` corresponds to the column of `tbl` in the same position. If NULL, column widths are calculated according to maximum number of characters per column.

`width_forest` **[Deprecated]** Please use the `rel_width_forest` argument instead.

`lbl_col_padding`
(numeric)
additional padding to use when calculating spacing between the first (label) column and the second column of `tbl`. If `colwidths` is specified, the width of the first column becomes `colwidths[1] + lbl_col_padding`. Defaults to 0.

`rel_width_forest`
(proportion)
proportion of total width to allocate to the forest plot. Relative width of table is then `1 - rel_width_forest`. If `as_list` = TRUE, this parameter is ignored.

`font_size` (numeric(1))
font size.

`col_symbol_size`
(numeric or NULL)
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).

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

`as_list` (flag)
whether the two ggplot objects should be returned as a list. If TRUE, a named list with two elements, `table` and `plot`, will be returned. If FALSE (default) the table and forest plot are printed side-by-side via `cowplot::plot_grid()`.

`gp` **[Deprecated]** `g_forest` is now generated as a ggplot object. This argument is no longer used.

`draw` **[Deprecated]** `g_forest` is now generated as a ggplot object. This argument is no longer used.

`newpage` **[Deprecated]** `g_forest` is now generated as a ggplot object. This argument is no longer used.

Details

Given a rtables::rtable() object with at least one column with a single value and one column with 2 values, converts table to a ggplot2::ggplot() object and generates an accompanying forest plot. The table and forest plot are printed side-by-side.

Value

ggplot forest plot and table.

Examples

```
library(dplyr)
library(forcats)

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)
g_forest(tbl)

# Odds ratio only table.

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

# 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))
```

```
)
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")
)
```

---

g_ipp *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 (string)
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.

plotting_choices
(string)
specifies options for displaying plots. Must be one of "all_in_one", "split_by_max_obs", or "separate_by_obs".

max_obs_per_plot
(integer(1))
number of observations to be plotted on one plot. Ignored if plotting_choices is not "separate_by_obs".

col (character)
line 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 parameter values over time.

See Also

Relevant helper function h_g_ipp().

Examples

```
library(dplyr)

# 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
```

---

g_km *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

```
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"),
```

```
  ylim = NULL,
  title = NULL,
  footnotes = NULL,
  font_size = 10,
  ci_ribbon = FALSE,
  annot_at_risk = TRUE,
  annot_at_risk_title = TRUE,
  annot_surv_med = TRUE,
  annot_coxph = FALSE,
  annot_stats = NULL,
  annot_stats_vlines = FALSE,
  control_coxph_pw = control_coxph(),
  ref_group_coxph = NULL,
  control_annot_surv_med = control_surv_med_annot(),
  control_annot_coxph = control_coxph_annot(),
  legend_pos = NULL,
  rel_height_plot = 0.75,
  ggtheme = NULL,
  as_list = FALSE,
  draw = lifecycle::deprecated(),
  newpage = lifecycle::deprecated(),
  gp = lifecycle::deprecated(),
  vp = lifecycle::deprecated(),
  name = lifecycle::deprecated(),
  annot_coxph_ref_lbls = lifecycle::deprecated(),
  position_coxph = lifecycle::deprecated(),
  position_surv_med = lifecycle::deprecated(),
  width_annots = lifecycle::deprecated()
)
```

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.
- `strata` (`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. If a vector is given, its length should be equal to the number of strata from survival::survfit().

lwd (numeric)
line width. If a vector is given, its length should be equal to the number of strata from survival::survfit().

censor_show (flag)
whether to show censored observations.

pch (string)
name of symbol or character to use as point symbol to indicate censored cases.

size (numeric(1))
size of censored point symbols.

max_time (numeric(1))
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 or NULL)
numeric vector of tick positions or a single number with spacing between ticks on the x-axis. If NULL (default), labeling::extended() is used to determine optimal tick positions on the x-axis.

xlab (string)
x-axis label.

yval (string)
type of plot, to be plotted on the y-axis. Options are Survival (default) and Failure probability.

ylab (string)
y-axis label.

ylim (numeric(2))
vector containing lower and upper limits for the y-axis, respectively. If NULL (default), the default scale range is used.

title (string)
plot title.

footnotes (string)
plot footnotes.

font_size (numeric(1))
font size to use for all text.

ci_ribbon (flag)
whether the confidence interval should be drawn around 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_at_risk_title
(flag)
whether the "Patients at Risk" title should be added above the annot_at_risk table. Has no effect if annot_at_risk is FALSE. Defaults to TRUE.

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)
whether to add the annotation table from a survival::coxph() model.

annot_stats (string or NULL)
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 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.

ref_group_coxph
(string or NULL)
level of arm variable to use as reference group in calculations for annot_coxph table. If NULL (default), uses the first level of the arm variable.

control_annot_surv_med
(list)
parameters to control the position and size of the annotation table added to the plot when annot_surv_med = TRUE, specified using the control_surv_med_annot() function. Parameter options are: x, y, w, h, and fill. See control_surv_med_annot() for details.

control_annot_coxph
(list)
parameters to control the position and size of the annotation table added to the

plot when annot_coxph = TRUE, specified using the control_coxph_annot() function. Parameter options are: x, y, w, h, fill, and ref_lbls. See control_coxph_annot() for details.

legend_pos (numeric(2) or NULL)
vector containing x- and y-coordinates, respectively, for the legend position relative to the KM plot area. If NULL (default), the legend is positioned in the bottom right corner of the plot, or the middle right of the plot if needed to prevent overlapping.

rel_height_plot (proportion)
proportion of total figure height to allocate to the Kaplan-Meier plot. Relative height of patients at risk table is then 1 - rel_height_plot. If annot_at_risk = FALSE or as_list = TRUE, this parameter is ignored.

ggtheme (theme)
a graphical theme as provided by ggplot2 to format the Kaplan-Meier plot.

as_list (flag)
whether the two ggplot objects should be returned as a list when annot_at_risk = TRUE. If TRUE, a named list with two elements, plot and table, will be returned. If FALSE (default) the patients at risk table is printed below the plot via cowplot::plot_grid().

draw **[Deprecated]** This function no longer generates grob objects.

newpage **[Deprecated]** This function no longer generates grob objects.

gp **[Deprecated]** This function no longer generates grob objects.

vp **[Deprecated]** This function no longer generates grob objects.

name **[Deprecated]** This function no longer generates grob objects.

annot_coxph_ref_lbls
[Deprecated] Please use the ref_lbls element of control_annot_coxph instead.

position_coxph **[Deprecated]** Please use the x and y elements of control_annot_coxph instead.

position_surv_med
[Deprecated] Please use the x and y elements of control_annot_surv_med instead.

width_annots **[Deprecated]** Please use the w element of control_annot_surv_med (for surv_med) and control_annot_coxph (for coxph)."

Value

A ggplot Kaplan-Meier plot and (optionally) summary table.

Examples

```
library(dplyr)

df <- tern_ex_adtte |>
  filter(PARAMCD == "OS") |>
```

```
  mutate(is_event = CNSR == 0)
variables <- list(tte = "AVAL", is_event = "is_event", arm = "ARMCD")

# Basic examples
g_km(df = df, variables = variables)
g_km(df = df, variables = variables, yval = "Failure")

# Examples with customization parameters applied
g_km(
  df = df,
  variables = variables,
  control_surv = control_surv_timepoint(conf_level = 0.9),
  col = c("grey25", "grey50", "grey75"),
  annot_at_risk_title = FALSE,
  lty = 1:3,
  font_size = 8
)
g_km(
  df = df,
  variables = variables,
  annot_stats = c("min", "median"),
  annot_stats_vlines = TRUE,
  max_time = 3000,
  ggtheme = ggplot2::theme_minimal()
)

# Example with pairwise Cox-PH analysis annotation table, adjusted annotation tables
g_km(
  df = df, variables = variables,
  annot_coxph = TRUE,
  control_coxph = control_coxph(pval_method = "wald", ties = "exact", conf_level = 0.99),
  control_annot_coxph = control_coxph_annot(x = 0.26, w = 0.35),
  control_annot_surv_med = control_surv_med_annot(x = 0.8, y = 0.9, w = 0.35)
)
```

`g_lineplot` *Line plot with optional table*

Description

[Stable]

Line plot with optional table.

Usage

```
g_lineplot(
  df,
  alt_counts_df = NULL,
```

```
  variables = control_lineplot_vars(),
  mid = "mean",
  interval = "mean_ci",
  whiskers = c("mean_ci_lwr", "mean_ci_upr"),
  table = NULL,
  sfun = s_summary,
  ...,
  mid_type = "pl",
  mid_point_size = 2,
  position = ggplot2::position_dodge(width = 0.4),
  legend_title = NULL,
  legend_position = "bottom",
  ggtheme = nestcolor::theme_nest(),
  xticks = NULL,
  xlim = NULL,
  ylim = NULL,
  x_lab = obj_label(df[[variables[["x"]]]]),
  y_lab = NULL,
  y_lab_add_paramcd = TRUE,
  y_lab_add_unit = TRUE,
  title = "Plot of Mean and 95% Confidence Limits by Visit",
  subtitle = "",
  subtitle_add_paramcd = TRUE,
  subtitle_add_unit = TRUE,
  caption = NULL,
  table_format = NULL,
  table_labels = NULL,
  table_font_size = 3,
  errorbar_width = 0.45,
  newpage = lifecycle::deprecated(),
  col = NULL,
  linetype = NULL,
  rel_height_plot = 0.5,
  as_list = FALSE
)
```

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 groups for stratification.
variables	(named character) vector of variable names in df which should include: • x (string) name of x-axis variable. • y (string) name of y-axis variable.

- group_var (string or NULL)
  name of grouping variable (or strata), i.e. treatment arm. Can be NA to indicate lack of groups.
- subject_var (string or NULL)
  name of subject variable. Only applies if group_var is not NULL.
- paramcd (string or NA)
  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 (string or NA)
  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.
- facet_var (string or NA)
  name of the secondary grouping variable used for plot faceting, i.e. treatment arm. Can be NA to indicate lack of groups.

mid (character or NULL)
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 (character or NULL)
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. Set interval = NULL if intervals should not be added to the plot.

whiskers (character)
names of the interval whiskers that will be plotted. Names must match names of the list element interval that will be returned by sfun (e.g. mean_ci_lwr element of sfun(x)[["mean_ci"]]). It is possible to specify one whisker only, or to suppress all whiskers by setting interval = NULL.

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

mid_point_size (numeric(1))
font size of the mid plot points.

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 (string)
legend title.

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

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

xticks (numeric or NULL)
numeric vector of tick positions or a single number with spacing between ticks on the x-axis, for use when variables$x is numeric. If NULL (default), labeling::extended() is used to determine optimal tick positions on the x-axis. If variables$x is not numeric, this argument is ignored.

xlim (numeric(2))
vector containing lower and upper limits for the x-axis, respectively. If NULL (default), the default scale range is used.

ylim (numeric(2))
vector containing lower and upper limits for the y-axis, respectively. If NULL (default), the default scale range is used.

x_lab (string or NULL)
x-axis label. If NULL then no label will be added.

y_lab (string or NULL)
y-axis label. If NULL then no label will be added.

y_lab_add_paramcd
(flag)
whether paramcd, i.e. unique(df[[variables["paramcd"]]]) should be added to the y-axis label (y_lab).

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

title (string)
plot title.

subtitle (string)
plot subtitle.

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

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

caption (string)
optional caption below the plot.

table_format (named vector or NULL)
custom formats for descriptive statistics used instead of defaults 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. Can be a character vector with values from formatters::list_valid_format_labels() or custom format functions.

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 (numeric(1))
font size of the text in the table.

errorbar_width (numeric(1))
width of the error bars.

newpage **[Deprecated]** not used.

col (character)
color(s). See ?ggplot2::aes_colour_fill_alpha for example values.

linetype (character)
line type(s). See ?ggplot2::aes_linetype_size_shape for example values.

rel_height_plot (proportion)
proportion of total figure height to allocate to the line plot. Relative height of annotation table is then 1 - rel_height_plot. If table = NULL, this parameter is ignored.

as_list (flag)
whether the two ggplot objects should be returned as a list when table is not NULL. If TRUE, a named list with two elements, plot and table, will be returned. If FALSE (default) the annotation table is printed below the plot via cowplot::plot_grid().

Value

A ggplot line plot (and statistics table if applicable).

Examples

```
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 with group_var
g_lineplot(adlb, variables = control_lineplot_vars(group_var = NA))

# Mean, upper whisker of CI, no group_var(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_analyze_vars(conf_level = 0.80),
  title = "Plot of Mean and 80% Confidence Limits by Visit"
)

# Mean with CI, table with customized formats/labels
g_lineplot(
  adlb,
  adsl,
  table = c("n", "mean", "mean_ci"),
  table_format = list(
    mean = function(x, ...) {
      ifelse(x < 20, round_fmt(x, digits = 3), round_fmt(x, digits = 2))
    },
    mean_ci = "(xx.xxx, xx.xxx)"
  ),
  table_labels = list(
    mean = "mean",
```

```
    mean_ci = "95% CI"
  )
)

# Mean with CI, table, filtered data
adlb_f <- dplyr::filter(adlb, ARMCD != "ARM A" | AVISIT == "BASELINE")
g_lineplot(adlb_f, table = c("n", "mean"))
```

`g_step` *Create a STEP graph*

Description

[Stable]

Based on the STEP results, creates a `ggplot` graph showing the estimated HR or OR along the continuous biomarker value subgroups.

Usage

```
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`) color(s).

Value

A `ggplot` STEP graph.

See Also

Custom tidy method `tidy.step()`.

Examples

```
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]

This basic waterfall plot visualizes a quantity `height` ordered by value with some markup.

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 identifiers 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`) color(s).
`xlab`	(`string`) x label. Default is `"ID"`.
`ylab`	(`string`) y label. Default is `"Value"`.

col_legend_title
: (string)
text to be displayed as legend title.

title
: (string)
text to be displayed as plot title.

Value

A ggplot waterfall plot.

Examples

```
library(dplyr)

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$pchg,
  id = paste("asdfdsfdsfsd", adrs_f$USUBJID),
  col_var = adrs_f$SEX
)

g_waterfall(
  height = adrs_f$pchg,
  id = paste("asdfdsfdsfsd", adrs_f$USUBJID),
  xlab = "ID",
  ylab = "Percentage Change",
  title = "Waterfall plot"
)
```

h_adlb_abnormal_by_worst_grade *Helper function to prepare ADLB for* `count_abnormal_by_worst_grade()`

Description

[Stable]

Helper function to prepare an ADLB data frame to be used as input in `count_abnormal_by_worst_grade()`. The following pre-processing steps are applied:

1. `adlb` is filtered on variable `avisit` to only include post-baseline visits.
2. `adlb` is filtered on variables `worst_flag_low` and `worst_flag_high` so that only worst grades (in either direction) are included.
3. From the standard lab grade variable `atoxgr`, the following two variables are derived and added to `adlb`:

- A grade direction variable (e.g. `GRADE_DIR`). The variable takes value `"HIGH"` when `atoxgr > 0`, `"LOW"` when `atoxgr < 0`, and `"ZERO"` otherwise.
- A toxicity grade variable (e.g. `GRADE_ANL`) where all negative values from `atoxgr` are replaced by their absolute values.

1. Unused factor levels are dropped from `adlb` via `droplevels()`.

Usage

```
h_adlb_abnormal_by_worst_grade(
  adlb,
  atoxgr = "ATOXGR",
  avisit = "AVISIT",
  worst_flag_low = "WGRLOFL",
  worst_flag_high = "WGRHIFL"
)
```

Arguments

`adlb`	(`data.frame`) ADLB data frame.
`atoxgr`	(`string`) name of the analysis toxicity grade variable. This must be a `factor` variable.
`avisit`	(`string`) name of the analysis visit variable.

worst_flag_low (string)
: name of the worst low lab grade flag variable. This variable is set to "Y" when indicating records of worst low lab grades.

worst_flag_high
: (string)
name of the worst high lab grade flag variable. This variable is set to "Y" when indicating records of worst high lab grades.

Value

h_adlb_abnormal_by_worst_grade() returns the adlb data frame with two new variables: GRADE_DIR and GRADE_ANL.

See Also

abnormal_by_worst_grade

Examples

```
h_adlb_abnormal_by_worst_grade(tern_ex_adlb) |>
  dplyr::select(ATOXGR, GRADE_DIR, GRADE_ANL) |>
  head(10)
```

---

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

```
h_adlb_worsen(
  adlb,
  worst_flag_low = NULL,
  worst_flag_high = NULL,
  direction_var
)
```

Arguments

adlb
: (data.frame)
ADLB data frame.

worst_flag_low (named vector)
: worst low post-baseline lab grade flag variable. See how this is implemented in the following examples.

worst_flag_high
: (named vector)
worst high post-baseline lab grade flag variable. See how this is implemented in the following examples.

direction_var
: (string)
name of the 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_lab_worsen_by_baseline

Examples

```
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 select laboratory tables*

Description

[Stable]

Helper function that merges ADSL and ADLB datasets so that missing lab test records are inserted in the output dataset. Remember that `na_level` must match the needed pre-processing done with `df_explicit_na()` to have the desired output.

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 data frame.
`adlb`	(`data.frame`) ADLB data frame.
`worst_flag`	(named `character`) worst post-baseline lab flag variable. See how this is implemented in the following examples.
`by_visit`	(`flag`) 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,
  weights_emmeans = 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 string)
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
(string or NULL)
name of the variable that should have interactions with arm. if the interaction is not needed, the default option is NULL.

weights_emmeans
(string or NULL)
argument from emmeans::emmeans()

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,
  only_grade_groups = FALSE
)
```

Arguments

grade_groups	(named list of character) list containing groupings of grades.
refs	(named list of numeric) named list where each name corresponds to a reference grade level and each entry represents a count.
remove_single	(flag) 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. If only_grade_groups is set to TRUE this argument is ignored.
only_grade_groups	(flag) whether only the specified grade groups should be included, with individual grade rows removed (TRUE), or all grades and grade groups should be displayed (FALSE).

Value

Formatted list of grade groupings.

Examples

```
h_append_grade_groups(
  list(
    "Any Grade" = as.character(1:5),
    "Grade 1-2" = c("1", "2"),
    "Grade 3-4" = c("3", "4")
  ),
  list("1" = 10, "2" = 20, "3" = 30, "4" = 40, "5" = 50)
)

h_append_grade_groups(
  list(
    "Any Grade" = as.character(5:1),
    "Grade A" = "5",
    "Grade B" = c("4", "3")
  ),
  list("1" = 10, "2" = 20, "3" = 30, "4" = 40, "5" = 50)
)

h_append_grade_groups(
  list(
    "Any Grade" = as.character(1:5),
    "Grade 1-2" = c("1", "2"),
    "Grade 3-4" = c("3", "4")
  ),
  list("1" = 10, "2" = 5, "3" = 0)
)
```

h_biomarkers_subgroups *Helper functions for tabulation of a single biomarker result*

Description

[Deprecated]

Usage

```
h_tab_one_biomarker(
  df,
  afuns,
  colvars,
  na_str = default_na_str(),
  ...,
  .stats = NULL,
  .stat_names = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)

h_tab_rsp_one_biomarker(
  df,
  vars,
  na_str = default_na_str(),
  .indent_mods = 0L,
  ...
)

h_tab_surv_one_biomarker(
  df,
  vars,
  time_unit,
  na_str = default_na_str(),
  .indent_mods = 0L,
  ...
)
```

Arguments

df	(data.frame) results for a single biomarker. For h_tab_rsp_one_biomarker(), the results returned by extract_rsp_biomarkers(). For h_tab_surv_one_biomarker(), the results returned by extract_survival_biomarkers().

afuns	(named list of function) analysis functions.
colvars	(named list) named list with elements vars (variables to tabulate) and labels (their labels).
na_str	(string) string used to replace all NA or empty values in the output.
...	additional arguments for the lower level functions.
.stats	(character) statistics to select for the table.
.stat_names	(character) names of the statistics that are passed directly to name single statistics (.stats). This option is visible when producing rtables::as_result_df() with make_ard = TRUE.
.formats	(named character or list) formats for the statistics. See Details in analyze_vars for more information on the "auto" setting.
.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.
vars	(character) variable names for the primary analysis variable to be iterated over.
time_unit	(string) label with unit of median survival time. Default NULL skips displaying unit.

Value

An rtables table object with statistics in columns.

Functions

- h_tab_one_biomarker(): Helper function to calculate statistics in columns for one biomarker.
- h_tab_rsp_one_biomarker(): Helper function that prepares a single response sub-table given the results for a single biomarker.
- h_tab_surv_one_biomarker(): Helper function that prepares a single survival sub-table given 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 == "BESRSPI") |>
  mutate(rsp = AVALC == "CR")
formatters::var_labels(adrs_f) <- c(adrs_labels, "Response")

# For a single population, separately estimate the effects of two biomarkers.
df <- h_logistic_mult_cont_df(
  variables = list(
    rsp = "rsp",
    biomarkers = c("BMRKR1", "AGE"),
    covariates = "SEX"
  ),
  data = adrs_f
)

# 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")
)

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

# For a single population, separately estimate 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
)
```

```
# 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_col_indices *Obtain column indices*

Description

[Stable]

Helper function to extract column indices from a `VTableTree` for a given vector of column names.

Usage

```
h_col_indices(table_tree, col_names)
```

Arguments

`table_tree`	(`VTableTree`) `rtables` table object 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

```
h_count_cumulative(
  x,
  threshold,
  lower_tail = TRUE,
  include_eq = TRUE,
  na_rm = TRUE,
  denom
)
```

Arguments

x	(numeric) vector of numbers we want to analyze.
threshold	(numeric(1)) a cutoff value as threshold to count values of x.
lower_tail	(flag) whether to count lower tail, default is TRUE.
include_eq	(flag) 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.
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.

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, denom = .N_col)
```

```
h_count_cumulative(x, 5, lower_tail = FALSE, include_eq = FALSE, na_rm = FALSE, denom = .N_col)
h_count_cumulative(x, 0, lower_tail = FALSE, denom = .N_col)
h_count_cumulative(x, 100, lower_tail = FALSE, denom = .N_col)
```

`h_cox_regression` *Helper functions for Cox proportional hazards regression*

Description

[Stable]

Helper functions used in `fit_coxreg_univar()` and `fit_coxreg_multivar()`.

Usage

```
h_coxreg_univar_formulas(variables, interaction = FALSE)

h_coxreg_multivar_formula(variables)

h_coxreg_univar_extract(effect, covar, data, mod, control = control_coxreg())

h_coxreg_multivar_extract(var, data, mod, control = control_coxreg())
```

Arguments

`variables`	(named `list` of `string`) list of additional analysis variables.
`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`.
`effect`	(`string`) the treatment variable.
`covar`	(`string`) the name of the covariate in the model.
`data`	(`data.frame`) the dataset containing the variables to summarize.
`mod`	(`coxph`) Cox regression model fitted by `survival::coxph()`.
`control`	(`list`) a list of controls as returned by `control_coxreg()`.
`var`	(`string`) single variable name that is passed by `rtables` when requested by a statistics function.

Value

- h_coxreg_univar_formulas() returns a character vector coercible into formulas (e.g stats::as.formula()).
- h_coxreg_multivar_formula() 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_formula(): 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.
h_coxreg_univar_formulas(
  variables = list(
    time = "time", event = "status", arm = "armcd", covariates = c("X", "y")
  )
)

## Addition of an optional strata.
h_coxreg_univar_formulas(
  variables = list(
    time = "time", event = "status", arm = "armcd", covariates = c("X", "y"),
    strata = "SITE"
  )
)

## Inclusion of the interaction term.
```

```
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.
h_coxreg_univar_formulas(
  variables = list(
    time = "time", event = "status", covariates = c("X", "y")
  )
)

# `h_coxreg_multivar_formula`

h_coxreg_multivar_formula(
  variables = list(
    time = "AVAL", event = "event", arm = "ARMCD", covariates = c("RACE", "AGE")
  )
)

# Addition of an optional strata.
h_coxreg_multivar_formula(
  variables = list(
    time = "AVAL", event = "event", arm = "ARMCD", covariates = c("RACE", "AGE"),
    strata = "SITE"
  )
)

# Example without treatment arm.
h_coxreg_multivar_formula(
  variables = list(
    time = "AVAL", event = "event", covariates = c("RACE", "AGE"),
    strata = "SITE"
  )
)

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")
)
mod <- coxph(Surv(time, status) ~ armcd + var1, data = dta_simple)
result <- h_coxreg_univar_extract(
  effect = "armcd", covar = "armcd", mod = mod, data = dta_simple
)
result
```

```
mod <- coxph(Surv(time, status) ~ armcd + var1, data = dta_simple)
result <- h_coxreg_multivar_extract(
  var = "var1", mod = mod, data = dta_simple
)
result
```

`h_data_plot` *Helper function to tidy survival fit data*

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`.

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(1)`) 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

```
library(dplyr)
library(survival)

# Test with multiple arms
tern_ex_adtte |>
  filter(PARAMCD == "OS") |>
  (\(x) survfit(formula = Surv(AVAL, 1 - CNSR) ~ ARMCD, data = x))() |>
  h_data_plot()

# Test with single arm
tern_ex_adtte |>
  filter(PARAMCD == "OS", ARMCD == "ARM B") |>
  (\(x) survfit(formula = Surv(AVAL, 1 - CNSR) ~ ARMCD, data = x))() |>
  h_data_plot(armval = "ARM B")
```

---

h_decompose_gg ggplot *decomposition*

Description

[Deprecated]

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") |>
  (\(x) survfit(formula = Surv(AVAL, 1 - CNSR) ~ ARMCD, data = x))()
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 format the optional* g_lineplot *table*

Description

[Stable]

Usage

```
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

```
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 to create a KM plot*

Description

[Deprecated]

Draw the Kaplan-Meier plot using ggplot2.

Usage

```
h_ggkm(
  data,
  xticks = NULL,
  yval = "Survival",
  censor_show,
  xlab,
  ylab,
  ylim = NULL,
  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 or NULL) numeric vector of tick positions or a single number with spacing between ticks on the x-axis. If NULL (default), labeling::extended() is used to determine optimal tick positions on the x-axis.
yval	(string) type of plot, to be plotted on the y-axis. Options are Survival (default) and Failure probability.
censor_show	(flag) whether to show censored observations.
xlab	(string) x-axis label.
ylab	(string) y-axis label.
ylim	(numeric(2)) vector containing lower and upper limits for the y-axis, respectively. If NULL (default), the default scale range is used.
title	(string) plot title.
footnotes	(string) plot footnotes.
max_time	(numeric(1)) 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. If a vector is given, its length should be equal to the number of strata from survival::survfit().
lty	(numeric) line type. If a vector is given, its length should be equal to the number of strata from survival::survfit().
pch	(string) name of symbol or character to use as point symbol to indicate censored cases.
size	(numeric(1)) size of censored point symbols.

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

ci_ribbon (flag)
whether the confidence interval should be drawn around the Kaplan-Meier curve.

ggtheme (theme)
a graphical theme as provided by ggplot2 to format the Kaplan-Meier plot.

Value

A ggplot object.

Examples

```
library(dplyr)
library(survival)

fit_km <- tern_ex_adtte |>
  filter(PARAMCD == "OS") |>
  (\(x) survfit(formula = Surv(AVAL, 1 - CNSR) ~ ARMCD, data = x))()
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",
  yval = "Survival",
  ylab = "Survival Probability",
  title = "Survival"
)
gg
```

---

h_grob_coxph *Helper function to create Cox-PH grobs*

Description

[Deprecated]

Grob of rtable output from h_tbl_coxph_pairwise()

Usage

```
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 to pass to h_tbl_coxph_pairwise().
x	(proportion) a value between 0 and 1 specifying x-location.
y	(proportion) a value between 0 and 1 specifying y-location.
width	(grid::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

```
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 to create survival estimation grobs*

Description

[Deprecated]

The survival fit is transformed in a grob containing a table with groups in rows characterized by N, median and 95% confidence interval.

Usage

```
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".
x	(proportion) a value between 0 and 1 specifying x-location.
y	(proportion) a value between 0 and 1 specifying y-location.
width	(grid::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 N, Median, and XX% CI (XX taken from fit_km).

Examples

```
library(dplyr)
library(survival)
library(grid)
```

```
grid::grid.newpage()
grid.rect(gp = grid::gpar(lty = 1, col = "pink", fill = "gray85", lwd = 1))
tern_ex_adtte |>
  filter(PARAMCD == "OS") |>
  (\(x) survfit(formula = Surv(AVAL, 1 - CNSR) ~ ARMCD, data = x))() |>
  h_grob_median_surv() |>
  grid::grid.draw()
```

`h_grob_tbl_at_risk` *Helper function to create patient-at-risk grobs*

Description

[Deprecated]

Two graphical objects are obtained, one corresponding to row labeling and the second to the table of numbers of patients at risk. If `title` = `TRUE`, a third object corresponding to the table title is also obtained.

Usage

```
h_grob_tbl_at_risk(data, annot_tbl, xlim, title = TRUE)
```

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(1)`) the maximum value on the x-axis (used to ensure the at risk table aligns with the KM graph).
`title`	(`flag`) whether the "Patients at Risk" title should be added above the `annot_at_risk` table. Has no effect if `annot_at_risk` is `FALSE`. Defaults to `TRUE`.

Value

A named `list` of two `gTree` objects if `title` = `FALSE`: `at_risk` and `label`, or three `gTree` objects if `title` = `TRUE`: `at_risk`, `label`, and `title`.

Examples

```
library(dplyr)
library(survival)
library(grid)

fit_km <- tern_ex_adtte |>
  filter(PARAMCD == "OS") |>
  (\(x) survfit(formula = Surv(AVAL, 1 - CNSR) ~ ARMCD, data = x))()

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
# times (`xticks`).
annot_tbl <- summary(fit_km, times = 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")
  levels(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 = 3:4, layout.pos.col = 2))
grid.rect(gp = grid::gpar(lty = 1, col = "orange", fill = "gray85", lwd = 1))
grid::grid.draw(tbl$at_risk)
popViewport()
pushViewport(viewport(layout.pos.row = 3:4, layout.pos.col = 1))
grid.rect(gp = grid::gpar(lty = 1, col = "green3", fill = "gray85", lwd = 1))
```

```
grid::grid.draw(tbl$label)
```

---

h_grob_y_annot *Helper function to create grid object with y-axis annotation*

Description

[Deprecated]

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

```
library(dplyr)
library(survival)
library(grid)

fit_km <- tern_ex_adtte |>
  filter(PARAMCD == "OS") |>
  (\(x) survfit(formula = Surv(AVAL, 1 - CNSR) ~ ARMCD, data = x))()
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

```
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` (`string`)
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`)
line colors.

Value

A `ggplot` line plot.

See Also

`g_ipp()` which uses this function.

Examples

```
library(dplyr)

# 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
)
p
```

`h_incidence_rate` *Helper functions for incidence rate*

Description

[Stable]

Usage

```
h_incidence_rate(person_years, n_events, control = control_incidence_rate())

h_incidence_rate_normal(person_years, n_events, alpha = 0.05)

h_incidence_rate_normal_log(person_years, n_events, alpha = 0.05)

h_incidence_rate_exact(person_years, n_events, alpha = 0.05)

h_incidence_rate_byar(person_years, n_events, alpha = 0.05)
```

Arguments

`person_years`	(`numeric(1)`) total person-years at risk.
`n_events`	(`integer(1)`) number of events observed.
`control`	(`list`) parameters for estimation details, specified by using the helper function `control_incidence_rate()`. Possible parameter options are: • `conf_level`: (`proportion`) confidence level for the estimated incidence rate. • `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`) time unit for desired output (in person-years).
`alpha`	(`numeric(1)`) two-sided alpha-level for confidence interval.

Value

Estimated incidence rate, `rate`, and associated confidence interval, `rate_ci`.

Functions

- `h_incidence_rate()`: Helper function to estimate the incidence rate and associated confidence interval.
- `h_incidence_rate_normal()`: Helper function to estimate the incidence rate and associated confidence interval based on the normal approximation for the incidence rate. Unit is one person-year.
- `h_incidence_rate_normal_log()`: Helper function to estimate the incidence rate and associated confidence interval based on the normal approximation for the logarithm of the incidence rate. Unit is one person-year.
- `h_incidence_rate_exact()`: Helper function to estimate the incidence rate and associated exact confidence interval. Unit is one person-year.
- `h_incidence_rate_byar()`: Helper function to estimate the incidence rate and associated Byar's confidence interval. Unit is one person-year.

See Also

incidence_rate

Examples

```
h_incidence_rate(200, 2)
h_incidence_rate(200, 2, control_incidence_rate(conf_type = "exact", num_pt_year = 100))

h_incidence_rate_normal(200, 2)

h_incidence_rate_normal_log(200, 2)

h_incidence_rate_exact(200, 2)

h_incidence_rate_byar(200, 2)
```

---

h_km_layout *Helper function to prepare a KM layout*

Description

[Deprecated]

Prepares a (5 rows) x (2 cols) layout for the Kaplan-Meier curve.

Usage

```
h_km_layout(
  data,
  g_el,
  title,
```

```
  footnotes,
  annot_at_risk = TRUE,
  annot_at_risk_title = 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) plot title.
footnotes	(string) plot footnotes.
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_at_risk_title	(flag) whether the "Patients at Risk" title should be added above the annot_at_risk table. Has no effect if annot_at_risk is FALSE. Defaults to TRUE.

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

```
library(dplyr)
library(survival)
library(grid)

fit_km <- tern_ex_adtte |>
  filter(PARAMCD == "OS") |>
  (\(x) survfit(formula = Surv(AVAL, 1 - CNSR) ~ ARMCD, data = x))()
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 `stats::glm()` with "binomial" family. Limited functionality is also available for conditional logistic regression models fitted by `survival::clogit()`, currently this is used only by `extract_rsp_biomarkers()`.
`interaction_vars`	(`character(2)`) interaction variable names.
`first_var_with_level`	(`character(2)`) the first variable name with the interaction level.
`second_var_with_level`	(`character(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`	(`numeric` or `NULL`) optional 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	(character) a variable or interaction term in fit_glm (depending on the helper function used).
...	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 data frame for* `trim_levels_to_map()`

Description

[Stable]

Helper function to create a map data frame 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

```
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_str = "<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_str` (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

```
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_str = "<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", "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_str = "<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, method = "exact")
```

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.
`method`	(`string`) whether to use the correct (`"exact"`) calculation in the conditional likelihood or one of the approximations. See `survival::clogit()` for details.

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, 1, 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, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 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 pharmacokinetic data by* PARAM *variable*

Description

[Stable]

Usage

```
h_pkparam_sort(pk_data, key_var = "PARAMCD")
```

Arguments

pk_data	(data.frame) pharmacokinetic data frame.
key_var	(string) key variable used to merge pk_data and metadata created by d_pkparam().

Value

A pharmacokinetic data.frame sorted by a PARAM variable.

Examples

```
library(dplyr)

adpp <- tern_ex_adpp |> mutate(PKPARAM = factor(paste0(PARAM, " (", AVALU, ")")))
pk_ordered_data <- h_pkparam_sort(adpp)
```

---

h_ppmeans *Function to return the estimated means using predicted probabilities*

Description

For each arm level, the predicted mean rate is calculated using the fitted model object, with `newdata` set to the result of `stats::model.frame`, a reconstructed data or the original data, depending on the object formula (coming from the fit). The confidence interval is derived using the `conf_level` parameter.

Usage

```
h_ppmeans(obj, .df_row, arm, conf_level)
```

Arguments

`obj`	(`glm.fit`) fitted model object used to derive the mean rate estimates in each treatment arm.
`.df_row`	(`data.frame`) dataset that includes all the variables that are called in `.var` and `variables`.
`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.
`conf_level`	(`proportion`) value used to derive the confidence interval for the rate.

Value

- `h_ppmeans()` returns the estimated means.

See Also

`summarize_glm_count()`.

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, n = length(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, n = length(rsp), conf_level)

prop_wald(rsp, n = length(rsp), conf_level, correct = FALSE)

prop_agresti_coull(rsp, n = length(rsp), conf_level)

prop_jeffreys(rsp, n = length(rsp), conf_level)
```

Arguments

`rsp`	(`logical`) vector indicating whether each subject is a responder or not.
`n`	(`count`) number of participants (if `denom = "N_col"`) or the number of responders (if `denom = "n"`, the default).
`conf_level`	(`proportion`) confidence level of the interval.
`correct`	(`flag`) whether to 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. Constructed (for 95% CI) by 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

Yan X, Su XG (2010). "Stratified Wilson and Newcombe Confidence Intervals for Multiple Binomial Proportions." *Stat. Biopharm. Res.*, **2**(3), 329–335.

See Also

estimate_proportion, descriptive function `d_proportion()`, and helper functions `strata_normal_quantile()` and `update_weights_strat_wilson()`.

Examples

```
rsp <- c(
  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)
```

---

h_prop_diff_test *Helper functions to test proportion differences*

Description

Helper functions to implement various tests on the difference between two proportions.

Usage

```
prop_chisq(tbl, alternative = c("two.sided", "less", "greater"))

prop_cmh(
  ary,
  alternative = c("two.sided", "less", "greater"),
  diff_se = c("standard", "sato"),
  transform = c("none", "wilson_hilferty")
)

prop_schouten(tbl, alternative = c("two.sided", "less", "greater"))

prop_fisher(tbl, alternative = c("two.sided", "less", "greater"))
```

Arguments

tbl	(matrix) matrix with two groups in rows and the binary response (TRUE/FALSE) in columns.
alternative	(string) whether two.sided, or one-sided less or greater p-value should be displayed.
ary	(array, 3 dimensions) array with two groups in rows, the binary response (TRUE/FALSE) in columns, and the strata in the third dimension.
diff_se	(string) either standard or sato; specifies whether to use the Sato variance estimator to calculate the chi-squared statistic.
transform	(string) either none or wilson_hilferty; specifies whether to apply the Wilson-Hilferty transformation of the chi-squared statistic.

Value

A p-value.

Functions

- prop_chisq(): Performs Chi-Squared test. Internally calls stats::prop.test().
- prop_cmh(): Performs stratified Cochran-Mantel-Haenszel test, using stats::mantelhaen.test() internally. Note that strata with less than two observations are automatically discarded.
- prop_schouten(): Performs the Chi-Squared test with Schouten correction.
- prop_fisher(): Performs the Fisher's exact test. Internally calls stats::fisher.test().

See Also

prop_diff_test() for implementation of these helper functions.

Schouten correction is based upon Schouten et al. (1980).

Examples

```
# Chi-Squared test
tbl <- matrix(
  c(13, 7, 8, 12),
  nrow = 2,
  byrow = TRUE,
  dimnames = list(group = c("A", "B"), response = c("TRUE", "FALSE"))
)

prop_chisq(tbl)

# Cochran-Mantel-Haenszel test with two strata
ary <- array(
  c(12, 8, 8, 12, 10, 10, 6, 14),
```

```
  dim = c(2, 2, 2),
  dimnames = list(
    group = c("A", "B"),
    response = c("TRUE", "FALSE"),
    strata = c("Low", "High")
  )
)

prop_cmh(ary)

# Chi-Squared test with Schouten correction
tbl <- matrix(
  c(13, 7, 8, 12),
  nrow = 2,
  byrow = TRUE,
  dimnames = list(group = c("A", "B"), response = c("TRUE", "FALSE"))
)

prop_schouten(tbl)

# Fisher's exact test
tbl <- matrix(
  c(13, 7, 8, 12),
  nrow = 2,
  byrow = TRUE,
  dimnames = list(group = c("A", "B"), response = c("TRUE", "FALSE"))
)

prop_fisher(tbl)
```

`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())
```

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().

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.

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 strata.

Examples

```
library(dplyr)
library(forcats)

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

adrs_f <- adrs |>
  filter(PARAMCD == "BESRSPI") |>
  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"),
    strata = "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

# 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",
    strata = "STRATA1"
  ),
  data = adrs_f[NULL, ]
)
```

---

h_response_subgroups *Helper functions for tabulating binary response by subgroup*

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) vector indicating 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 or NULL) specifies the test used to calculate the p-value for the difference between two proportions. For options, see 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 strata. 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") |>
  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.
h_proportion_subgroups_df(
  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")
    )
  )
)

# Unstratatified analysis.
h_odds_ratio_df(
  c(TRUE, FALSE, FALSE, TRUE),
  arm = factor(c("A", "A", "B", "B"), levels = c("A", "B"))
)

# Include p-value.
h_odds_ratio_df(adrs_f$rsp, adrs_f$ARM, method = "chisq")

# Stratatified analysis.
h_odds_ratio_df(
  rsp = adrs_f$rsp,
  arm = adrs_f$ARM,
  strata_data = adrs_f[, c("STRATA1", "STRATA2")],
  method = "cmh"
)

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

# Stratified analysis.
h_odds_ratio_subgroups_df(
  variables = list(
    rsp = "rsp",
    arm = "ARM",
    subgroups = c("SEX", "BMRKR2"),
    strata = c("STRATA1", "STRATA2")
  ),
  data = adrs_f
)

# Define groupings of BMRKR2 levels.
h_odds_ratio_subgroups_df(
  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")
    )
  )
)
```

`h_split_by_subgroups` *Split data frame by subgroups*

Description

[Stable]

Split a data frame into a non-nested list of subsets.

Usage

```
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

[Deprecated]

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`) 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 *Helper function to create a new SMQ variable in ADAE by stacking SMQ and/or CQ records.*

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. Remember that `na_str` must match the needed pre-processing done with `df_explicit_na()` to have the desired output.

Usage

```
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_str = "<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_str	(string) string used to replace all NA or empty values in the output.

Value

A 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

```
adae <- tern_ex_adae[1:20, ] |> df_explicit_na()
h_stack_by_baskets(df = adae)

aag <- data.frame(
  NAMVAR = c("CQ01NAM", "CQ02NAM", "SMQ01NAM", "SMQ02NAM"),
  REFNAME = c(
    "D.2.1.5.3/A.1.1.1.1 aesi", "X.9.9.9.9/Y.8.8.8.8 aesi",
    "C.1.1.1.3/B.2.2.3.1 aesi", "C.1.1.1.3/B.3.3.3.3 aesi"
  ),
  SCOPE = c("", "", "BROAD", "BROAD"),
  stringsAsFactors = FALSE
)

basket_name <- character(nrow(aag))
cq_pos <- grep("^(CQ).+NAM$", aag$NAMVAR)
smq_pos <- grep("^(SMQ).+NAM$", aag$NAMVAR)
basket_name[cq_pos] <- aag$REFNAME[cq_pos]
basket_name[smq_pos] <- paste0(
  aag$REFNAME[smq_pos], "(", aag$SCOPE[smq_pos], ")"
)

aag_summary <- data.frame(
  basket = aag$NAMVAR,
  basket_name = basket_name,
  stringsAsFactors = TRUE
```

```
)

result <- h_stack_by_baskets(df = adae, aag_summary = aag_summary)
all(levels(aag_summary$basket_name) %in% levels(result$SMQ))

h_stack_by_baskets(
  df = adae,
  aag_summary = NULL,
  keys = c("STUDYID", "USUBJID", "AEDECOD", "ARM"),
  baskets = "SMQ01NAM"
)
```

---

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())

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	(coxph or glm) 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_step_rsp_formula() returns a model formula.
- h_step_rsp_est() returns a matrix of number of observations n, log odds ratio estimates logor, standard error se, and Wald confidence interval bounds ci_lower and ci_upper. One row is included for each biomarker value in x.

Functions

- h_step_window(): Creates the windows for STEP, based on the control settings provided.
- h_step_trt_effect(): Calculates the estimated treatment effect estimate on the linear predictor scale and corresponding standard error from a STEP model fitted on data given variables specification, for a single biomarker value x. This works for both coxph and glm models, i.e. for calculating log hazard ratio or log odds ratio estimates.
- h_step_survival_formula(): Builds the model formula used in survival STEP calculations.
- h_step_survival_est(): Estimates the model with formula built based on variables in data for a given subset and control parameters for the Cox regression.
- h_step_rsp_formula(): Builds the model formula used in response STEP calculations.
- h_step_rsp_est(): Estimates the model with formula built based on variables in data for a given subset and control parameters for the logistic regression.

h_survival_biomarkers_subgroups *Helper functions for tabulating biomarker effects on survival 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_surv_to_coxreg_variables(variables, biomarker)

h_coxreg_mult_cont_df(variables, data, control = control_coxreg())
```

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().

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.

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 strata.

Examples

```
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[NULL, ]
)
```

`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`) vector of time-to-event duration values.
`is_event`	(`flag`) `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 the null hypothesis that 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.
- `alternative` (`string`)
  alternative hypothesis for the p-value test. Default is `"two.sided"`, can also be set to `"less"` or `"greater"` for one-sided testing. Note that one-sided testing is not supported when `pval_method = "likelihood"`.

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 strata. groups_lists optionally specifies groupings for subgroups variables.

Examples

```
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.
h_survtime_subgroups_df(
  variables = list(
    tte = "AVAL",
    is_event = "is_event",
    arm = "ARM",
```

```
    subgroups = c("SEX", "BMRKR2")
  ),
  data = adtte_f
)

# Define groupings for BMRKR2 levels.
h_survtime_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"),
```

```
      "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"),
    strata = c("STRATA1", "STRATA2")
  ),
  data = adtte_f
)
```

`h_tbl_coxph_pairwise` *Helper function for generating a pairwise Cox-PH table*

Description

[Stable]

Create a `data.frame` of pairwise stratified or unstratified Cox-PH analysis results.

Usage

```
h_tbl_coxph_pairwise(
  df,
  variables,
  ref_group_coxph = NULL,
  control_coxph_pw = control_coxph(),
  annot_coxph_ref_lbls = FALSE
)
```

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.

- strata (character or NULL)
  variable names indicating stratification factors.

ref_group_coxph
: (string or NULL)
level of arm variable to use as reference group in calculations for annot_coxph table. If NULL (default), uses the first level of the arm variable.

control_coxph_pw
: (list)
parameters for comparison details, specified 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.

annot_coxph_ref_lbls
: (flag)
whether the reference group should be explicitly printed in labels for the annot_coxph table. If FALSE (default), only comparison groups will be printed in annot_coxph table labels.

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 for 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", digits = 4)
```

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"`.
`digits`	(`integer(1)`) number of significant digits for median and CI values. Defaults to `4`.

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(
  formula = Surv(AVAL, 1 - CNSR) ~ ARMCD,
  data = adtte
)
h_tbl_median_surv(fit_km = fit)
h_tbl_median_surv(fit_km = fit, digits = 2)
```

`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`) name of the baseline lab grade variable.
`direction_var`	(`string`) name of the 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

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_lab_worsen_by_baseline

Examples

```
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 to calculate 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

```
h_xticks(data, xticks = NULL, max_time = NULL)
```

Arguments

`data`	(`data.frame`) survival data as pre-processed by `h_data_plot`.

xticks	(numeric or NULL) numeric vector of tick positions or a single number with spacing between ticks on the x-axis. If NULL (default), labeling::extended() is used to determine optimal tick positions on the x-axis.
max_time	(numeric(1)) 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 vector of positions to use for x-axis ticks on a ggplot object.

Examples

```
library(dplyr)
library(survival)

data <- tern_ex_adtte |>
  filter(PARAMCD == "OS") |>
  (\(x) survfit(formula = Surv(AVAL, 1 - CNSR) ~ ARMCD, data = x))() |>
  h_data_plot()

h_xticks(data)
h_xticks(data, xticks = seq(0, 3000, 500))
h_xticks(data, xticks = 500)
h_xticks(data, xticks = 500, max_time = 6000)
h_xticks(data, xticks = c(0, 500), max_time = 300)
h_xticks(data, xticks = 500, max_time = 300)
```

---

imputation_rule *Apply 1/3 or 1/2 imputation rule to data*

Description

[Stable]

Usage

```
imputation_rule(
  df,
  x_stats,
  stat,
  imp_rule,
  post = FALSE,
  avalcat_var = "AVALCAT1"
)
```

Arguments

df	(data.frame) data set containing all analysis variables.
x_stats	(named list) a named list of statistics, typically the results of s_summary().
stat	(string) statistic to return the value/NA level of according to the imputation rule applied.
imp_rule	(string) imputation rule setting. Set to "1/3" to implement 1/3 imputation rule or "1/2" to implement 1/2 imputation rule.
post	(flag) whether the data corresponds to a post-dose time-point (defaults to FALSE). This parameter is only used when imp_rule is set to "1/3".
avalcat_var	(string) name of variable that indicates whether a row in df corresponds to an analysis value in category "BLQ", "LTR", "<PCLLOQ", or none of the above (defaults to "AVALCAT1"). Variable avalcat_var must be present in df.

Value

A list containing statistic value (val) and NA level (na_str) that should be displayed according to the specified imputation rule.

See Also

analyze_vars_in_cols() where this function can be implemented by setting the imp_rule argument.

Examples

```
set.seed(1)
df <- data.frame(
  AVAL = runif(50, 0, 1),
  AVALCAT1 = sample(c(1, "BLQ"), 50, replace = TRUE)
)
x_stats <- s_summary(df$AVAL)
imputation_rule(df, x_stats, "max", "1/3")
imputation_rule(df, x_stats, "geom_mean", "1/3")
imputation_rule(df, x_stats, "mean", "1/2")
```

labels_or_names *Labels or names of list elements*

Description

Helper function for working with nested statistic function results which typically don't have labels but names that we can use.

Usage

```
labels_or_names(x)
```

Arguments

x	(list) a list.

Value

A character vector with the labels or names for the list elements.

Examples

```
x <- data.frame(
  a = 1:10,
  b = rnorm(10)
)
labels_or_names(x)
var_labels(x) <- c(b = "Label for b", a = NA)
labels_or_names(x)
```

labels_use_control *Update labels according to control specifications*

Description

[Stable]

Given a list of statistic labels and and a list of control parameters, updates labels with a relevant control specification. For example, if control has element conf_level set to 0.9, the default label for statistic mean_ci will be updated to "Mean 90% CI". Any labels that are supplied via labels_custom will not be updated regardless of control.

Usage

```
labels_use_control(labels_default, control, labels_custom = NULL)
```

Arguments

`labels_default`	(named `character`) a named vector of statistic labels to modify according to the control specifications. Labels that are explicitly defined in `labels_custom` will not be affected.
`control`	(named `list`) list of control parameters to apply to adjust default labels.
`labels_custom`	(named `character`) named vector of labels that are customized by the user and should not be affected by `control`.

Value

A named character vector of labels with control specifications applied to relevant labels.

Examples

```
control <- list(conf_level = 0.80, quantiles = c(0.1, 0.83), test_mean = 0.57)
get_labels_from_stats(c("mean_ci", "quantiles", "mean_pval")) |>
  labels_use_control(control = control)
```

`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 `rtables::split_cols_by_multivar()`.

Usage

```
logistic_regression_cols(lyt, conf_level = 0.95)
```

Arguments

`lyt`	(`PreDataTableLayouts`) layout that analyses will be added to.
`conf_level`	(`proportion`) confidence level of the interval.

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 `df`, `estimate`, `std_error`, `odds_ratio`, `ci`, and `pvalue`.

logistic_summary_by_flag *Logistic regression summary table*

Description

[Stable]

Constructor for content functions to be used in `summarize_logistic()` to summarize logistic regression results. This function is a wrapper for `rtables::summarize_row_groups()`.

Usage

```
logistic_summary_by_flag(
  flag_var,
  na_str = default_na_str(),
  .indent_mods = NULL
)
```

Arguments

`flag_var`	(`string`) variable name identifying which row should be used in this content function.
`na_str`	(`string`) string used to replace all `NA` or empty values in the output.
`.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.

month2day *Conversion of months to days*

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(1)) 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]

The analyze function estimate_odds_ratio() creates a layout element to compare bivariate responses between two groups by estimating an odds ratio and its confidence interval.

The primary analysis variable specified by vars is the group variable. Additional variables can be included in the analysis via the variables argument, which accepts arm, an arm variable, and strata, a stratification variable. If more than two arm levels are present, they can be combined into two groups using the groups_list argument.

Usage

```
estimate_odds_ratio(
  lyt,
  vars,
  variables = list(arm = NULL, strata = NULL),
  conf_level = 0.95,
  groups_list = NULL,
  method = "exact",
  na_str = default_na_str(),
  nested = TRUE,
  ...,
  table_names = vars,
  show_labels = "hidden",
  var_labels = vars,
  .stats = "or_ci",
  .stat_names = NULL,
  .formats = NULL,
  .labels = NULL,
```

```
  .indent_mods = NULL
)

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,
  method = "exact",
  ...
)

a_odds_ratio(
  df,
  ...,
  .stats = NULL,
  .stat_names = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```

Arguments

lyt	(PreDataTableLayouts) layout that analyses will be added to.
vars	(character) variable names for the primary analysis variable to be iterated over.
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.
method	(string) whether to use the correct ("exact") calculation in the conditional likelihood or one of the approximations. See survival::clogit() for details.
na_str	(string) string used to replace all NA or empty values in the output.
nested	(flag) whether this layout instruction should 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.
...	additional arguments to rtables::split_cols_by() in order. For instance, to control formats (format), add a joint column for all groups (incl_all).
table_names	(character) this can be customized in the case that the same vars are analyzed multiple times, to avoid warnings from rtables.
show_labels	(string) label visibility: one of "default", "visible" and "hidden".
var_labels	(character) variable labels.
.stats	(character) statistics to select for the table. Options are: 'or_ci', 'n_tot'
.stat_names	(character) names of the statistics that are passed directly to name single statistics (.stats). This option is visible when producing rtables::as_result_df() with make_ard = TRUE.
.formats	(named character or list) formats for the statistics. See Details in analyze_vars for more information on the "auto" setting.
.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.
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	(flag) 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.

Value

- 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.

- `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()`.

Functions

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

Note

- This function uses logistic regression for unstratified analyses, and conditional logistic regression for stratified analyses. The Wald confidence interval is calculated with the specified confidence level.
- For stratified analyses, there is currently no implementation for conditional likelihood confidence intervals, therefore the likelihood confidence interval is not available as an option.
- When `vars` contains only responders or non-responders no odds ratio estimation is possible so the returned values will be `NA`.

See Also

Relevant helper function `h_odds_ratio()`.

Examples

```
set.seed(12)
dta <- data.frame(
  rsp = sample(c(TRUE, FALSE), 100, TRUE),
  grp = factor(rep(c("A", "B"), each = 50), levels = c("A", "B")),
  strata = factor(sample(c("C", "D"), 100, TRUE))
)

l <- basic_table() |>
  split_cols_by(var = "grp", ref_group = "B") |>
  estimate_odds_ratio(vars = "rsp")

build_table(l, df = dta)

# 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, 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
)
```

---

prop_diff *Proportion difference estimation*

Description

[Stable]

The analysis function `estimate_proportion_diff()` creates a layout element to estimate the difference in proportion of responders within a studied population. The primary analysis variable, `vars`, is a logical variable indicating whether a response has occurred for each record. See the `method` parameter for options of methods to use when constructing the confidence interval of the proportion difference. A stratification variable can be supplied via the `strata` element of the `variables` argument.

Usage

```
estimate_proportion_diff(
  lyt,
  vars,
  variables = list(strata = NULL),
  conf_level = 0.95,
  method = c("waldcc", "wald", "cmh", "cmh_sato", "cmh_mn", "ha", "newcombe",
    "newcombecc", "strat_newcombe", "strat_newcombecc", "uncond_exact_diff"),
  weights_method = "cmh",
  var_labels = vars,
  na_str = default_na_str(),
  nested = TRUE,
  show_labels = "hidden",
```

```
  table_names = vars,
  section_div = NA_character_,
  ...,
  na_rm = TRUE,
  .stats = c("diff", "diff_ci"),
  .stat_names = NULL,
  .formats = c(diff = "xx.x", diff_ci = "(xx.x, xx.x)", se_diff = "xx.x"),
  .labels = NULL,
  .indent_mods = c(diff = 0L, diff_ci = 1L, se_diff = 1L)
)

s_proportion_diff(
  df,
  .var,
  .ref_group,
  .in_ref_col,
  variables = list(strata = NULL),
  conf_level = 0.95,
  method = c("waldcc", "wald", "cmh", "cmh_sato", "cmh_mn", "ha", "newcombe",
    "newcombecc", "strat_newcombe", "strat_newcombecc", "uncond_exact_diff"),
  weights_method = "cmh",
  ...
)

a_proportion_diff(
  df,
  ...,
  .stats = NULL,
  .stat_names = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```

Arguments

lyt	(PreDataTableLayouts) layout that analyses will be added to.
vars	(character) variable names for the primary analysis variable to be iterated over.
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.
var_labels	(character) variable labels.
na_str	(string) string used to replace all NA or empty values in the output.
nested	(flag) whether this layout instruction should 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.
show_labels	(string) label visibility: one of "default", "visible" and "hidden".
table_names	(character) this can be customized in the 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.
...	additional arguments for the lower level functions.
na_rm	(flag) whether NA values should be removed from x prior to analysis.
.stats	(character) statistics to select for the table. Options are: 'diff', 'diff_ci'
.stat_names	(character) names of the statistics that are passed directly to name single statistics (.stats). This option is visible when producing rtables::as_result_df() with make_ard = TRUE.
.formats	(named character or list) formats for the statistics. See Details in analyze_vars for more information on the "auto" setting.
.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.
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	(flag) TRUE when working with the reference level, FALSE otherwise.

Details

The possible methods are:

- "waldcc": Wald confidence interval with continuity correction (Agresti and Coull 1998).
- "wald": Wald confidence interval without continuity correction (Agresti and Coull 1998).
- "cmh": Cochran-Mantel-Haenszel (CMH) confidence interval (Mantel and Haenszel 1959).
- "cmh_sato": CMH confidence interval with Sato variance estimator (Sato et al. 1989).
- "cmh_mn": CMH confidence interval with Miettinen and Nurminen confidence interval (Miettinen and Nurminen 1985).
- "ha": Anderson-Hauck confidence interval (Hauck and Anderson 1986).
- "newcombe": Newcombe confidence interval without continuity correction (Newcombe 1998).
- "newcombecc": Newcombe confidence interval with continuity correction (Newcombe 1998).
- "strat_newcombe": Stratified Newcombe confidence interval without continuity correction (Yan and Su 2010).
- "strat_newcombecc": Stratified Newcombe confidence interval with continuity correction (Yan and Su 2010).
- "uncond_exact_diff": Unconditional exact confidence interval for the difference in proportions (Santner and Snell 1980).

Value

- 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.
- s_proportion_diff() returns a named list of elements diff and diff_ci. Depending on the method used, also the standard error of the difference se_diff is returned.
- a_proportion_diff() returns the corresponding list with formatted rtables::CellValue().

Functions

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

Note

When performing an unstratified analysis, methods "cmh", "cmh_sato", "strat_newcombe", and "strat_newcombecc" are not permitted. For stratified analysis, method "uncond_exact_diff" is not permitted.

References

Agresti A, Coull BA (1998). "Approximate is Better than "Exact" for Interval Estimation of Binomial Proportions." *The American Statistician*, **52**(2), 119–126. doi:10.1080/00031305.1998.10480550.

Hauck WW, Anderson S (1986). "A Comparison of Large-Sample Confidence Interval Methods for the Difference of Two Binomial Probabilities." *The American Statistician*, **40**(4), 318–322. doi:10.2307/2684618.

Mantel N, Haenszel W (1959). "Statistical aspects of the analysis of data from retrospective studies of disease." *Journal of the National Cancer Institute*, **22**(4), 719–748.

Miettinen OS, Nurminen M (1985). "Comparative analysis of two rates." *Statistics in Medicine*, **4**(2), 213–226. doi:10.1002/sim.4780040211.

Newcombe RG (1998). "Interval estimation for the difference between independent proportions: comparison of eleven methods." *Statistics in Medicine*, **17**(8), 873-890. doi:10.1002/(SICI)1097-0258(19980430)17:8<873::AIDSIM779>3.0.CO;2I.

Santner TJ, Snell MK (1980). "Small-Sample Confidence Intervals for p1 - p2 and p1/p2 in 2 x 2 Contingency Tables." *Journal of the American Statistical Association*, **75**(370), 386–394. doi:10.1080/01621459.1980.10477482.

Sato T, Greenland S, Robins JM (1989). "On the variance estimator for the Mantel-Haenszel Risk Difference." *Biometrics*, **45**(4), 1323–1324. http://www.jstor.org/stable/2531784.

Yan X, Su XG (2010). "Stratified Wilson and Newcombe Confidence Intervals for Multiple Binomial Proportions." *Stat. Biopharm. Res.*, **2**(3), 329–335.

See Also

d_proportion_diff()

Examples

```
## "Mid" case: 4/4 respond in group A, 1/2 respond in group B.
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
)

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)

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"),
  .stats = c("diff"),
  .var = "rsp",
  .ref_group = subset(dta, grp == "B"),
  .in_ref_col = FALSE,
  conf_level = 0.90,
  method = "ha"
)
```

`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 `CombinationFunction()` they can be logically combined with other condition functions.

Usage

```
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, ...)
```

Arguments

`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`	(`numeric(1)`) threshold which should be met in order to keep the row.
`...`	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

- `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

```
tab <- basic_table() |>
  split_cols_by("ARM") |>
  split_rows_by("RACE") |>
  split_rows_by("STRATA1") |>
  summarize_row_groups() |>
  analyze_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(any_atleast_five_percent))



# `has_fractions_difference`
more_than_five_percent_diff <- has_fractions_difference(atleast = 0.05, col_names = names(tab))
prune_table(tab, keep_rows(more_than_five_percent_diff))



more_than_one_diff <- has_counts_difference(atleast = 1L, col_names = names(tab))
prune_table(tab, keep_rows(more_than_one_diff))
```

`range_noinf` *Re-implemented* `range()` *default S3 method for numerical objects*

Description

[Stable]

This function returns `c(NA, NA)` instead of `c(-Inf, Inf)` for zero-length data without any warnings.

Usage

```
range_noinf(x, na.rm = FALSE, finite = FALSE)
```

Arguments

`x`	(`numeric`) a sequence of numbers for which the range is computed.
`na.rm`	(`flag`) flag indicating if `NA` should be omitted.
`finite`	(`flag`) flag indicating if non-finite elements should be removed.

Value

A 2-element vector of class `numeric`.

Examples

```
x <- rnorm(20, 1)
range_noinf(x, na.rm = TRUE)
range_noinf(rep(NA, 20), na.rm = TRUE)
range(rep(NA, 20), na.rm = TRUE)
```

`reapply_varlabels` *Reapply variable labels*

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]

The `tabulate_rsp_biomarkers()` function creates a layout element to tabulate the estimated biomarker effects on a binary response endpoint across subgroups, returning statistics including response rate and odds ratio for each population subgroup. The table is created from `df`, a list of data frames returned by `extract_rsp_biomarkers()`, with the statistics to include specified via the `vars` parameter.

A forest plot can be created from the resulting table using the `g_forest()` function.

Usage

```
tabulate_rsp_biomarkers(
  df,
  vars = c("n_tot", "n_rsp", "prop", "or", "ci", "pval"),
  na_str = default_na_str(),
  ...,
  .stat_names = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```

Arguments

`df`	(`data.frame`) containing all analysis variables, as returned by `extract_rsp_biomarkers()`.
`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.
`na_str`	(`string`) string used to replace all `NA` or empty values in the output.

`...`	additional arguments for the lower level functions.
`.stat_names`	(`character`) names of the statistics that are passed directly to name single statistics (`.stats`). This option is visible when producing `rtables::as_result_df()` with `make_ard = TRUE`.
`.formats`	(named `character` or `list`) formats for the statistics. See Details in `analyze_vars` for more information on the `"auto"` setting.
`.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

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 binary response by subgroup.

Note

In contrast to `tabulate_rsp_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

`extract_rsp_biomarkers()`

Examples

```
library(dplyr)
library(forcats)

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

adrs_f <- adrs |>
  filter(PARAMCD == "BESRSPI") |>
  mutate(rsp = AVALC == "CR")
formatters::var_labels(adrs_f) <- c(adrs_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))
```

`rtable2gg` *Convert* `rtable` *objects to* `ggplot` *objects*

Description

[Experimental]

Given a `rtables::rtable()` object, performs basic conversion to a `ggplot2::ggplot()` object built using functions from the `ggplot2` package. Any table titles and/or footnotes are ignored.

Usage

```
rtable2gg(tbl, fontsize = 12, colwidths = NULL, lbl_col_padding = 0)
```

Arguments

`tbl`	(`VTableTree`) `rtables` table object.
`fontsize`	(`numeric(1)`) font size.
`colwidths`	(`numeric` or `NULL`) a vector of column widths. Each element's position in `colwidths` corresponds to the column of `tbl` in the same position. If `NULL`, column widths are calculated according to maximum number of characters per column.
`lbl_col_padding`	(`numeric`) additional padding to use when calculating spacing between the first (label) column and the second column of `tbl`. If `colwidths` is specified, the width of the first column becomes `colwidths[1] + lbl_col_padding`. Defaults to 0.

Value

A ggplot object.

Examples

```
dta <- data.frame(
  ARM     = rep(LETTERS[1:3], rep(6, 3)),
  AVISIT  = rep(paste0("V", 1:3), 6),
  AVAL    = c(9:1, rep(NA, 9))
)

lyt <- basic_table() |>
  split_cols_by(var = "ARM") |>
  split_rows_by(var = "AVISIT") |>
  analyze_vars(vars = "AVAL")

tbl <- build_table(lyt, df = dta)

rtable2gg(tbl)

rtable2gg(tbl, fontsize = 15, colwidths = c(2, 1, 1, 1))
```

---

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) values for which any missing values should be substituted.
empty	(flag) if TRUE, empty strings get replaced by NA.
whitespaces	(flag) if TRUE, strings made from only whitespaces 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()`: Produces a 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

```
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", "*", "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", "*", "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

`lyt`	(`PreDataTableLayouts`) layout that analyses will be added to.
`var`	(`string`) single variable name that is passed by `rtables` when requested by a statistics function.
`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.
`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") |>
  build_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"))
      }
    }
  ) |>
  build_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") |>
  build_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"))
      }
    }
  ) |>
  build_table(DM)
```

---

stack_grobs *Stack multiple grobs*

Description

[Deprecated]

Stack grobs as a new grob with 1 column and multiple rows layout.

Usage

```
stack_grobs(
  ...,
  grobs = list(...),
  padding = grid::unit(2, "line"),
  vp = NULL,
  gp = NULL,
  name = NULL
)
```

Arguments

...	grobs.
grobs	(list of grob) a list of grobs.
padding	(grid::unit) unit of length 1, space between each grob.
vp	(viewport or NULL) a viewport() object (or NULL).
gp	(gpar) a gpar() object.
name	(string) a character identifier for the grob.

Value

A grob.

Examples

```
library(grid)

g1 <- circleGrob(gp = gpar(col = "blue"))
g2 <- circleGrob(gp = gpar(col = "red"))
g3 <- textGrob("TEST TEXT")
grid.newpage()
grid.draw(stack_grobs(g1, g2, g3))

showViewport()

grid.newpage()
pushViewport(viewport(layout = grid.layout(1, 2)))
vp1 <- 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

```
stat_mean_ci(
  x,
  conf_level = 0.95,
  na.rm = TRUE,
  n_min = 2,
  gg_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`	(`numeric(1)`) a minimum number of non-missing `x` to estimate the confidence interval for mean.
`gg_helper`	(`flag`) whether output should be aligned for use with `ggplot`s.
`geom_mean`	(`flag`) whether the geometric mean should be calculated.

Value

A named `vector` of values `mean_ci_lwr` and `mean_ci_upr`.

Examples

```
stat_mean_ci(sample(10), gg_helper = FALSE)

p <- ggplot2::ggplot(mtcars, ggplot2::aes(cyl, 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

```
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(1)`) a minimum number of non-missing `x` to estimate the p-value of the mean.
`test_mean`	(`numeric(1)`) 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`	(`flag`) whether output should be aligned for use with `ggplot`s.

Details

This function was adapted from `DescTools/versions/0.99.35/source`

Value

A named `vector` of values `median_ci_lwr` and `median_ci_upr`.

Examples

```
stat_median_ci(sample(10), gg_helper = FALSE)

p <- ggplot2::ggplot(mtcars, ggplot2::aes(cyl, mpg)) +
  ggplot2::geom_point()
p + ggplot2::stat_summary(
  fun.data = stat_median_ci,
  geom = "errorbar"
)
```

`stat_propdiff_ci` *Proportion difference and confidence interval*

Description

[Stable]

Function for calculating the proportion (or risk) difference and confidence interval between arm X (reference group) and arm Y. Risk difference is calculated by subtracting cumulative incidence in arm Y from cumulative incidence in arm X.

Usage

```
stat_propdiff_ci(
  x,
  y,
  N_x,
  N_y,
  list_names = NULL,
  conf_level = 0.95,
  pct = TRUE
)
```

Arguments

`x`	(`list` of `integer`) list of number of occurrences in arm X (reference group).
`y`	(`list` of `integer`) list of number of occurrences in arm Y. Must be of equal length to `x`.
`N_x`	(`numeric(1)`) total number of records in arm X.
`N_y`	(`numeric(1)`) total number of records in arm Y.
`list_names`	(`character`) names of each variable/level corresponding to pair of proportions in `x` and `y`. Must be of equal length to `x` and `y`.
`conf_level`	(`proportion`) confidence level of the interval.
`pct`	(`flag`) whether output should be returned as percentages. Defaults to `TRUE`.

Value

List of proportion differences and CIs corresponding to each pair of number of occurrences in `x` and `y`. Each list element consists of 3 statistics: proportion difference, CI lower bound, and CI upper bound.

See Also

Split function `add_riskdiff()` which, when used as `split_fun` within `rtables::split_cols_by()` with `riskdiff` argument is set to `TRUE` in subsequent analyze functions, adds a column containing proportion (risk) difference to an `rtables` layout.

Examples

```
stat_propdiff_ci(
  x = list(0.375), y = list(0.01), N_x = 5, N_y = 5, list_names = "x", conf_level = 0.9
)

stat_propdiff_ci(
  x = list(0.5, 0.75, 1), y = list(0.25, 0.05, 0.5), N_x = 10, N_y = 20, pct = FALSE
)
```

---

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

```
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)
ests <- strata_data["TRUE", ] / ns
vars <- ests * (1 - ests) / ns
weights <- rep(1 / length(ns), length(ns))

strata_normal_quantile(vars, weights, 0.95)
```

---

summarize_colvars *Summarize variables in columns*

Description

[Stable]

The analyze function `summarize_colvars()` uses the statistics function `s_summary()` to analyze variables that are arranged in columns. The variables to analyze should be specified in the table layout via column splits (see `rtables::split_cols_by()` and `rtables::split_cols_by_multivar()`) prior to using `summarize_colvars()`.

The function is a minimal wrapper for `rtables::analyze_colvars()`, a function typically used to apply different analysis methods in rows for each column variable. To use the analysis methods as column labels, please refer to the `analyze_vars_in_cols()` function.

Usage

```
summarize_colvars(
  lyt,
  na_str = default_na_str(),
  ...,
  .stats = c("n", "mean_sd", "median", "range", "count_fraction"),
  .stat_names = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```

Arguments

`lyt`	(`PreDataTableLayouts`) layout that analyses will be added to.
`na_str`	(`string`) string used to replace all `NA` or empty values in the output.

`...`	arguments passed to `s_summary()`.
`.stats`	(`character`) statistics to select for the table.
`.stat_names`	(`character`) names of the statistics that are passed directly to name single statistics (`.stats`). This option is visible when producing `rtables::as_result_df()` with `make_ard = TRUE`.
`.formats`	(named `character` or `list`) formats for the statistics. See Details in `analyze_vars` for more information on the `"auto"` setting.
`.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") |>
  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 `tern` content functions to add summary rows to a given table layout:

Details

- `add_rowcounts()`
- `estimate_multinomial_response()` (with `rtables::analyze()`)
- `logistic_summary_by_flag()`
- `summarize_num_patients()`
- `summarize_occurrences()`
- `summarize_occurrences_by_grade()`
- `summarize_patients_events_in_cols()`
- `summarize_patients_exposure_in_cols()`

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.

Usage

```
summarize_logistic(
  lyt,
  conf_level,
  drop_and_remove_str = "",
  .indent_mods = NULL
)
```

Arguments

`lyt`	(`PreDataTableLayouts`) layout that analyses will be added to.
`conf_level`	(`proportion`) confidence level of the interval.
`drop_and_remove_str`	(`string`) 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 = "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)

# 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
```

`survival_biomarkers_subgroups` *Tabulate biomarker effects on survival by subgroup*

Description

[Stable]

The `tabulate_survival_biomarkers()` function creates a layout element to tabulate the estimated effects of multiple continuous biomarker variables on survival across subgroups, returning statistics including median survival time and hazard ratio for each population subgroup. The table is created from `df`, a list of data frames returned by `extract_survival_biomarkers()`, with the statistics to include specified via the `vars` parameter.

A forest plot can be created from the resulting table using the `g_forest()` function.

Usage

```
tabulate_survival_biomarkers(
  df,
  vars = c("n_tot", "n_tot_events", "median", "hr", "ci", "pval"),
  groups_lists = list(),
  control = control_coxreg(),
  label_all = lifecycle::deprecated(),
  time_unit = NULL,
  na_str = default_na_str(),
  ...,
  .stat_names = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```

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.

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 **[Deprecated]**
please assign the label_all parameter within the extract_survival_biomarkers() function when creating df.

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

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

... additional arguments for the lower level functions.

.stat_names (character)
names of the statistics that are passed directly to name single statistics (.stats). This option is visible when producing rtables::as_result_df() with make_ard = TRUE.

.formats (named character or list)
formats for the statistics. See Details in analyze_vars for more information on the "auto" setting.

.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

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

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

# 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`.

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

## Table with default columns.
tabulate_survival_biomarkers(df)

## Table with a manually chosen set of columns: leave out "pval", reorder.
tab <- tabulate_survival_biomarkers(
  df = df,
  vars = c("n_tot_events", "ci", "n_tot", "median", "hr"),
  time_unit = as.character(adtte_f$AVALU[1])
)

## Finally produce the forest plot.

g_forest(tab, xlim = c(0.8, 1.2))
```

`survival_time` *Survival time analysis*

Description

[Stable]

The analyze function `surv_time()` creates a layout element to analyze survival time by calculating survival time median, median confidence interval, quantiles, and range (for all, censored, or event patients). The primary analysis variable `vars` is the time variable and the secondary analysis variable `is_event` indicates whether or not an event has occurred.

Usage

```
surv_time(
  lyt,
  vars,
  is_event,
  control = control_surv_time(),
  ref_fn_censor = TRUE,
  na_str = default_na_str(),
  nested = TRUE,
  ...,
  var_labels = "Time to Event",
  show_labels = "visible",
```

```
  table_names = vars,
  .stats = c("median", "median_ci", "quantiles", "range"),
  .stat_names = NULL,
  .formats = list(median_ci = "(xx.x, xx.x)", quantiles = "xx.x, xx.x", range =
    "xx.x to xx.x", quantiles_lower = "xx.x (xx.x - xx.x)", quantiles_upper =
    "xx.x (xx.x - xx.x)", median_ci_3d = "xx.x (xx.x - xx.x)"),
  .labels = list(median_ci = "95% CI", range = "Range"),
  .indent_mods = list(median_ci = 1L)
)

s_surv_time(df, .var, ..., is_event, control = control_surv_time())

a_surv_time(
  df,
  labelstr = "",
  ...,
  .stats = NULL,
  .stat_names = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```

Arguments

lyt	(PreDataTableLayouts) layout that analyses will be added to.
vars	(character) variable names for the primary analysis variable to be iterated over.
is_event	(flag) TRUE if event, FALSE if time to event is censored.
control	(list) parameters for comparison details, specified by using the helper function control_surv_time(). Some possible parameter options are: • conf_level (proportion) confidence level of the interval for survival time. • conf_type (string) confidence interval type. Options are "plain" (default), "log", or "log-log", see more in survival::survfit(). Note option "none" is not supported. • quantiles (numeric) vector of length two to specify the quantiles of survival time.
ref_fn_censor	(flag) whether referential footnotes indicating censored observations should be printed when the range statistic is included.
na_str	(string) string used to replace all NA or empty values in the output.

nested	(flag) whether this layout instruction should 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.
...	additional arguments for the lower level functions.
var_labels	(character) variable labels.
show_labels	(string) label visibility: one of "default", "visible" and "hidden".
table_names	(character) this can be customized in the case that the same vars are analyzed multiple times, to avoid warnings from rtables.
.stats	(character) statistics to select for the table. Options are: 'median', 'median_ci', 'median_ci_3d', 'quantiles', 'quantiles_lower', 'qua
.stat_names	(character) names of the statistics that are passed directly to name single statistics (.stats). This option is visible when producing rtables::as_result_df() with make_ard = TRUE.
.formats	(named character or list) formats for the statistics. See Details in analyze_vars for more information on the "auto" setting.
.labels	(named character) labels for the statistics (without indent).
.indent_mods	(named 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.
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.
labelstr	(string) 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.

Value

- surv_time() 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_surv_time() to the table layout.

- s_surv_time() returns the statistics:

  - median: Median survival time.
  - median_ci: Confidence interval for median time.
  - median_ci_3d: Median with confidence interval for median time.
  - quantiles: Survival time for two specified quantiles.
  - quantiles_lower: quantile with confidence interval for the first specified quantile.
  - quantiles_upper: quantile with confidence interval for the second specified quantile.
  - range_censor: Survival time range for censored observations.
  - range_event: Survival time range for observations with events.
  - range: Survival time range for all observations.
  - range_with_cens_info: Survival time range for all observations, with + suffix on censored bounds.

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

Functions

- surv_time(): Layout-creating function which can take statistics function arguments and additional format arguments. This function is a wrapper for rtables::analyze().
- s_surv_time(): Statistics function which analyzes survival times.
- a_surv_time(): Formatted analysis function which is used as afun in surv_time().

Examples

```
library(dplyr)

adtte_f <- tern_ex_adtte |>
  filter(PARAMCD == "OS") |>
  mutate(
    AVAL = day2month(AVAL),
    is_event = CNSR == 0
  )
df <- adtte_f |> filter(ARMCD == "ARM A")

basic_table() |>
  split_cols_by(var = "ARMCD") |>
  add_colcounts() |>
  surv_time(
    vars = "AVAL",
    var_labels = "Survival Time (Months)",
    is_event = "is_event",
    control = control_surv_time(conf_level = 0.9, conf_type = "log-log")
  ) |>
  build_table(df = adtte_f)

a_surv_time(
  df,
  .df_row = df,
  .var = "AVAL",
  is_event = "is_event"
```

```
)
```

`s_bland_altman` *Bland-Altman analysis*

Description

[Experimental]

Statistics function that uses the Bland-Altman method to assess the agreement between two numerical vectors and calculates a variety of statistics.

Usage

```
s_bland_altman(x, y, conf_level = 0.95)
```

Arguments

`x`	(`numeric`) vector of numbers we want to analyze.
`y`	(`numeric`) vector of numbers we want to analyze, to be compared with `x`.
`conf_level`	(`proportion`) confidence level of the interval.

Value

A named list of the following elements:

- `df`
- `difference_mean`
- `ci_mean`
- `difference_sd`
- `difference_se`
- `upper_agreement_limit`
- `lower_agreement_limit`
- `agreement_limit_se`
- `upper_agreement_limit_ci`
- `lower_agreement_limit_ci`
- `t_value`
- `n`

Examples

```
x <- seq(1, 60, 5)
y <- seq(5, 50, 4)

s_bland_altman(x, y, conf_level = 0.9)
```

`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

```
## S3 method for class 'glm'
tidy(x, conf_level = 0.95, at = NULL, ...)
```

Arguments

`x`	(`glm`) logistic regression model fitted by `stats::glm()` with "binomial" family.
`conf_level`	(`proportion`) confidence level of the interval.
`at`	(`numeric` or `NULL`) 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

```
## S3 method for class 'step'
tidy(x, ...)
```

Arguments

`x`	(`matrix`) results from `fit_survival_step()`.
`...`	not used.

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

```
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)
```

---

tidy_coxreg *Custom tidy methods for Cox regression*

Description

[Stable]

Usage

```
## 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

broom::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

```
library(survival)
library(broom)

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

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)

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	(integer(1)) number of entries that are needed.

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_spacesparameter 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,
  widths = NULL,
  max_width = NULL,
  hsep = formatters::default_hsep(),
  with_spaces = TRUE,
  print_txt_to_copy = FALSE
)
```

Arguments

x	(VTableTree) rtables table object.
widths	(numeric or NULL) Proposed widths for the columns of x. The expected length of this numeric vector can be retrieved with ncol(x) + 1 as the column of row names must also be considered.

max_width (integer(1), string or NULL)
width that title and footer (including footnotes) materials should be word-wrapped to. If NULL, it is set to the current print width of the session (getOption("width")). If set to "auto", the width of the table (plus any table inset) is used. Parameter is ignored if tf_wrap = FALSE.

hsep (string)
character to repeat to create header/body separator line. If NULL, the object value will be used. If " ", an empty separator will be printed. See default_hsep() for more information.

with_spaces (flag)
whether the tested table should keep the indentation and other relevant spaces.

print_txt_to_copy (flag)
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. If print_txt_to_copy = TRUE the well formatted printout of the table will be printed to console, ready to be copied as a expected value.

Examples

```
tbl <- basic_table() |>
  split_rows_by("SEX") |>
  split_cols_by("ARM") |>
  analyze("AGE") |>
  build_table(tern_ex_adsl)

to_string_matrix(tbl, widths = ceiling(propose_column_widths(tbl) / 2))
```

---

univariate *Univariate formula special term*

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	(character) a vector of variable names separated by commas.

Details

If provided alongside with pairwise specification, the model `y ~ ARM + univariate(SEX, AGE, RACE)` lead to the study and comparison of the models

- `y ~ ARM`
- `y ~ ARM + SEX`
- `y ~ ARM + AGE`
- `y ~ ARM + 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

```
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(1)) 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	(integer(1)) maximum number of iterations to be tried. Convergence is always checked.
conf_level	(proportion) confidence level of the interval.
tol	(numeric(1)) 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)
```

---

utils_split_funs *Custom split functions*

Description

[Stable]

Collection of useful functions that are expanding on the core list of functions provided by rtables. See rtables::custom_split_funs and rtables::make_split_fun() for more information on how to make a custom split function. All these functions work with rtables::split_rows_by() argument split_fun to modify the way the split happens. For other split functions, consider consulting rtables::split_funcs.

Usage

```
ref_group_position(position = "first")

level_order(order)
```

Arguments

position	(string or integer) position to use for the reference group facet. Can be "first", "last", or a specific position.
order	(character or numeric) vector of ordering indices for the split facets.

Value

- ref_group_position() returns an utility function that puts the reference group as first, last or at a certain position and needs to be assigned to split_fun.
- level_order() returns an utility function that changes the original levels' order, depending on input order and split levels.

Functions

- ref_group_position(): Split function to place reference group facet at a specific position during post-processing stage.
- level_order(): Split function to change level order based on an integer vector or a character vector that represent the split variable's factor levels.

See Also

rtables::make_split_fun()

Examples

```
library(dplyr)

dat <- data.frame(
  x = factor(letters[1:5], levels = letters[5:1]),
  y = 1:5
)

# With rtables layout functions
basic_table() |>
  split_cols_by("x", ref_group = "c", split_fun = ref_group_position("last")) |>
  analyze("y") |>
  build_table(dat)

# With tern layout funcitons
adtte_f <- tern_ex_adtte |>
  filter(PARAMCD == "OS") |>
```

```
  mutate(
    AVAL = day2month(AVAL),
    is_event = CNSR == 0
  )

basic_table() |>
 split_cols_by(var = "ARMCD", ref_group = "ARM B", split_fun = ref_group_position("first")) |>
  add_colcounts() |>
  surv_time(
    vars = "AVAL",
    var_labels = "Survival Time (Months)",
    is_event = "is_event",
  ) |>
  build_table(df = adtte_f)

basic_table() |>
  split_cols_by(var = "ARMCD", ref_group = "ARM B", split_fun = ref_group_position(2)) |>
  add_colcounts() |>
  surv_time(
    vars = "AVAL",
    var_labels = "Survival Time (Months)",
    is_event = "is_event",
  ) |>
  build_table(df = adtte_f)

# level_order --------
# Even if default would bring ref_group first, the original order puts it last
basic_table() |>
  split_cols_by("Species", split_fun = level_order(c(1, 3, 2))) |>
  analyze("Sepal.Length") |>
  build_table(iris)

# character vector
new_order <- level_order(levels(iris$Species)[c(1, 3, 2)])
basic_table() |>
  split_cols_by("Species", ref_group = "virginica", split_fun = new_order) |>
  analyze("Sepal.Length") |>
  build_table(iris)
```

Index