Package: crane (via r-universe)

June 3, 2026

Title Supplements the 'gtsummary' Package for Pharmaceutical Reporting

Version 0.3.2

Description Tables summarizing clinical trial results are often complex and require detailed tailoring prior to submission to a health authority. The 'crane' package supplements the functionality of the 'gtsummary' package for creating these often highly bespoke tables in the pharmaceutical industry.

License Apache License 2.0

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

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

Depends gtsummary (>= 2.5.1), R (>= 4.2)

Imports broom (>= 1.0.8), cards (>= 0.8.0), cardx (>= 0.3.3), cli (>= 3.6.4), cowplot (>= 1.2.0), dplyr (>= 1.2.0), flextable (>= 0.9.7), ggplot2 (>= 4.0.0), glue (>= 1.8.0), gt (>= 0.11.1), labeling, lifecycle, rlang (>= 1.1.5), scales, scales (>= 1.3.0), survival (>= 3.6-4), tibble, tidyr (>= 1.3.0)

Suggests broom.helpers (>= 1.20.0), coin (>= 1.4.3), emmeans, labelled, magick, mmrm, parameters, pharmaverseadam, testthat (>= 3.0.0), tidyselect, withr (>= 3.0.1), yaml

Config/Needs/check hms

Config/Needs/website rmarkdown, yaml

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

Language en-US

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 8.0.0

Config/pak/sysreqs libcairo2-dev cmake libfontconfig1-dev libfreetype6-dev libfribidi-dev make libharfbuzz-dev libicu-dev libjpeg-dev libpng-dev libtiff-dev libuv1-dev libwebp-dev libxml2-dev libssl-dev libnode-dev

Repository https://insightsengineering.r-universe.dev

Date/Publication 2026-06-03 10:30:23 UTC

RemoteUrl https://github.com/insightsengineering/crane

RemoteRef v0.3.2

RemoteSha e6e535d371ce75219e83858b20ae45f1d8646c32

Contents

`add_blank_rows` *Add Blank Row*

Description

Add a blank row below each variable group defined by `variables` or below each specified `row_numbers`. A blank row will not be added to the bottom of the table.

NOTE: For HTML flextable output (which includes the RStudio IDE Viewer), the blank rows do not render. But they will appear when the table is rendered to Word.

Usage

```
add_blank_rows(x, variables = NULL, row_numbers = NULL, variable_level = NULL)
```

Arguments

`x` (`gtsummary`)
a 'gtsummary' table. The table must include a column named `'variable'` in `x$table_body`.

`variables`, `row_numbers`, `variable_level`
(`tidy-select` or `integer`)

- `variables`: When a table contains variable summaries, use this argument to add blank rows below the specified variable block.
- `row_numbers`: Add blank rows after each row number specified.
- `variable_level`: A single column name in `x$table_body` and blank rows will be added after each unique level.

Value

updated 'gtsummary' table.

Examples

```
# Example 1 ----------------------------------
# Default to every variable used
trial |>
  tbl_roche_summary(
    by = trt,
    include = c(age, marker, grade),
    nonmissing = "always"
  ) |>
  add_blank_rows(variables = everything())

# Example 2 ----------------------------------
trial |>
```

```
  tbl_roche_summary(
    by = trt,
    include = c(age, marker, grade),
    nonmissing = "always"
  ) |>
  add_blank_rows(variables = age)
```

`add_forest` *Add a Forest Plot Column to a gtsummary Table*

Description

This function adds a forest plot column to a gtsummary table, typically produced by `tbl_roche_subgroups()`. The forest plot visualizes estimates and confidence intervals for each subgroup in the table. The function supports rendering with either the `gt` or `flextable` engines, making it suitable for different outputs.

Usage

```
add_forest(
  x,
  estimate = starts_with("estimate"),
  conf_low = starts_with("conf.low"),
  conf_high = starts_with("conf.high"),
  pvalue = starts_with("p.value"),
  after = starts_with("p.value"),
  header_spaces = 20,
  table_engine = c("flextable", "gt")
)
```

Arguments

`x`	(`gtsummary`) A gtsummary table with estimates and confidence intervals in the table body. Usually produced by `tbl_roche_subgroups()`.
`estimate`	(`tidy-select`) Estimate column name.
`conf_low`	(`tidy-select`) Confidence interval lower bound column name.
`conf_high`	(`tidy-select`) Confidence interval upper bound column name.
`pvalue`	(`tidy-select`, optional) P-value column name. Point sizes in the forest plot will be scaled according to p-value (smaller p-values = smaller points). `NULL` to turn this off.
`after`	(`tidy-select`) Column name after which the forest plot column will be added. Default is after the p-value column.

header_spaces (integer)
Spaces to add to the forest plot header to visually separate the two treatment areas (`trt A\n Better` and `trt B\nBetter`). It is suggested to modify manually this variable if the treatment names are long, with `add_forest(..., header_spaces = 5)` or `flextable::set_header_labels(ggplot = "*")`.

table_engine (character)
Table rendering engine to use. Default is "flextable".

Details

Both gt and flextable outputs could produce issues in line continuity between rows if there are wrapping in the statistical cells.

Value

a gt table or flextable object with an added forest plot column.

Examples

```
# Simple example ------------------------------------------------------------
trial |>
  select(age, marker, grade, response) |>
  tbl_uvregression(
    y = response,
    method = glm,
    method.args = list(family = binomial),
    exponentiate = TRUE,
    hide_n = TRUE
  ) |>
  modify_column_merge(
    pattern = "{estimate} (95% CI {ci}; {p.value})",
    rows = !is.na(estimate)
  ) |>
  modify_header(estimate = "**Odds Ratio**") |>
  add_forest(table_engine = "gt")

# Realistic example ---------------------------------------------------------

if (requireNamespace("broom.helpers", quietly = TRUE)) {
  trial |>
    tbl_roche_subgroups(
      rsp = "response",
      by = "trt",
      subgroups = c("grade"),
      ~ glm(response ~ trt, data = .x) |>
        gtsummary::tbl_regression(
          show_single_row = trt,
          exponentiate = TRUE # , tidy_fun = broom.helpers::tidy_parameters
        )
    ) |>
    add_forest(pvalue = starts_with("p.value"), table_engine = "flextable") |>
    flextable::set_header_labels(ggplot = "---------")
```

```
}
```

`add_hierarchical_count_row` *Add row with counts*

Description

Typically used to add a row with overall AE counts to a table that primarily displays AE rates.

Usage

```
add_hierarchical_count_row(
  x,
  label = "Overall total number of events",
  .before = NULL,
  .after = NULL,
  data_preprocess = identity
)
```

Arguments

`x`	(`gtsummary`) a gtsummary table
`label`	(`string`) label for the new row
`.before, .after`	(`integer`) Row index where to add the new row. Default is after last row.
`data_preprocess`	(`function` or `formula`) a function that is applied to `x$inputs$data` before the total row counts are tabulated. Default is `identity`. Tidyverse formula shortcut notation for the function is accepted. See `rlang::as_function()` for details.

Value

gtsummary table

Examples

```
# Example 1 ----------------------------------
cards::ADAE |>
  # subset the data for a shorter example table
  dplyr::slice(1:10) |>
  tbl_hierarchical(
    by = "TRTA",
    variables = AEDECOD,
```

```
  denominator = cards::ADSL,
  id = "USUBJID",
  overall_row = TRUE
) |>
add_hierarchical_count_row(.after = 1L)
```

`adjust_stat_columns_wrap` *Adjust Statistics Columns Wrapping*

Description

Toggles standard whitespace to non-breaking spaces (`\u00A0`) dynamically in all visible statistics columns of a `gtsummary` table's body (or vice versa). This forces the layout engine to keep statistics (e.g., "12.5 (95%)") on a single line when protected. Column headers and labels remain unaffected. For the rare cases when protecting creates ugly squashed label column protection can be reversed using the same function.

Usage

```
adjust_stat_columns_wrap(tbl, mode = c("protect", "unprotect"))
```

Arguments

`tbl`	(`gtsummary`) A `gtsummary` object.
`mode`	(`character(1)`) Either `"protect"` (replaces whitespace with non-breaking spaces) or `"unprotect"` (replaces non-breaking spaces with standard spaces). Defaults to `"protect"`.

Value

A modified `gtsummary` object.

Examples

```
tbl <- gtsummary::tbl_summary(
  trial,
  by = trt,
  include = c(age, grade)
)
adjust_stat_columns_wrap(tbl, "protect")
```

annotate_gg_km *Annotate Kaplan-Meier Plot*

Description

These functions provide capabilities to annotate Kaplan-Meier plots (gg_km()) with additional summary tables, including median survival times, numbers at risk, and cox proportional hazards results. The annotations are added using the cowplot package for flexible placement.

Usage

```
annotate_riskdf(
  gg_plt,
  fit_km,
  title = "Patients at Risk:",
  rel_height_plot = 0.75,
  xlab = "Days",
  ...
)

annotate_surv_med(
  gg_plt,
  fit_km,
  table_position = c(x = 0.8, y = 0.85, w = 0.32, h = 0.16),
  ...
)

annotate_coxph(
  gg_plt,
  coxph_tbl,
  table_position = c(x = 0.29, y = 0.51, w = 0.4, h = 0.125),
  ...
)
```

Arguments

gg_plt	(ggplot2 or cowplot) The primary plot object of the Kaplan-Meier plot. Note: While floating tables accept cowplot objects, aligned tables (like the risk table) require a pure ggplot2 object.
fit_km	(survfit) A fitted Kaplan-Meier object of class survfit (from the survival package). This object contains the necessary survival data used to calculate and generate the content displayed in the annotation table.
title	(character) A single string value indicating whether to include a title above the table. Defaults to "Patients at Risk:". If NULL, no title is added.

`rel_height_plot` (numeric)
A single numeric value defining the **relative height** of the main Kaplan-Meier plot area compared to the 'at-risk' table. This value should be between 0 and 1, where a value closer to 1 gives the main plot more vertical space. Defaults to `0.75`.

`xlab` (character)
A single character string for the **x-axis label** on the 'at-risk' table. This typically represents time (e.g., "Days").

`...` Additional arguments passed to the control list for the annotation box. These arguments override the default values.
Accepted arguments include:

- `fill` (`logical`): Whether the annotation box should have a background fill. Default is `TRUE`.
- `font_size` (`numeric`): Base font size for the text inside the annotation box. Default is `10`.

`table_position` (numeric)
A named numeric vector `c(x, y, w, h)` defining the position and size of the floating table. `x` and `y` are the coordinates (0 to 1), while `w` and `h` represent width and height (0 to 1). Defaults vary by function.

`coxph_tbl` (`data.frame`)
A data frame containing the pre-calculated Cox-PH results, derived using function `get_cox_pairwise_df()`.

Value

The function `annotate_riskdf` returns a `cowplot` object combining the KM plot and the 'Numbers at Risk' table.

The function `annotate_surv_med` returns a `cowplot` object with the median survival table annotation added.

The function `annotate_coxph` returns a `cowplot` object with the Cox-PH table annotation added.

Functions

- `annotate_riskdf()`: The function `annotate_riskdf` adds a "Numbers at Risk" table below a Kaplan-Meier plot using `patchwork`.
  Note: For this specific function, `gg_plt` must be a pure `ggplot2` object (not a combined `cowplot` object) because it requires exact X-axis extraction.
- `annotate_surv_med()`: The `annotate_surv_med` function adds a median survival time summary table as an annotation box.
- `annotate_coxph()`: The function `annotate_coxph()` adds a Cox Proportional Hazards summary table as an annotation box.

See Also

`gg_km()`, `process_survfit()`, and `get_cox_pairwise_df()` for related functionalities.

Examples

```
# Preparing the Kaplan-Meier Plot
library(survival)
use_lung <- lung
use_lung$arm <- factor(sample(c("A", "B", "C"), nrow(use_lung), replace = TRUE))
use_lung$status <- use_lung$status - 1 # Convert status to 0/1
use_lung <- na.omit(use_lung)

formula <- Surv(time, status) ~ arm
fit_kmg01 <- survfit(formula, use_lung)
surv_plot_data <- process_survfit(fit_kmg01)

plt_kmg01 <- gg_km(surv_plot_data)

# Annotate Plot with Numbers at Risk Table
annotate_riskdf(plt_kmg01, fit_kmg01)

# Change order of y-axis (arm)
use_lung2 <- use_lung
use_lung2$arm <- factor(use_lung2$arm, levels = c("C", "B", "A"))
fit_kmg01 <- survival::survfit(formula, use_lung2)
annotate_riskdf(plt_kmg01, fit_kmg01) # rerun gg_km to change legend order

# Annotate Kaplan-Meier Plot with Median Survival Table
annotate_surv_med(plt_kmg01, fit_kmg01)

# Annotate Kaplan-Meier Plot with Cox-PH Table
coxph_tbl <- get_cox_pairwise_df(
  formula,
  data = use_lung, arm = "arm", ref_group = "A"
)
result <- annotate_coxph(plt_kmg01, coxph_tbl)

# Extract original plots from any annotated result
attr(result, "plotlist")$main
```

---

annotate_lineplot_df *Annotate Line Plot with Summary Table*

Description

These functions provide capabilities to annotate lineplot (`gg_lineplot()`) with additional summary statistics table. The annotations are added using the `cowplot` package for flexible placement.

Usage

```
annotate_lineplot_df(
  gg_plt,
```

```
  data,
  x = NULL,
  y = NULL,
  group = NULL,
  summary_stats = c("n", "mean", "sd"),
  digits = NULL,
  rel_height_plot = 0.75
)
```

Arguments

gg_plt	(ggplot2) The base line plot generated by gg_lineplot().
data	(data.frame) The raw data used to generate the statistics.
x, y, group	(string or NULL) Optional column names as strings. If NULL (default), the function automatically extracts these from the gg_plt mapping.
summary_stats	(character) Vector of statistics to include. Defaults to c("n", "mean", "sd").
digits	(numeric, list, or formula) Optional specification for the number of decimal places for the summary statistics. Can be a single integer (e.g., 2), a vector of integers matching the statistics (e.g., c(0, 2, 2)), or a gtsummary style formula. Defaults to NULL (uses gtsummary default auto-formatting).
rel_height_plot	(numeric) Relative height of the plot vs the table. Defaults to 0.75.

Value

A cowplot object.

See Also

gg_lineplot() for related functionalities.

Examples

```
# 1. Create a mock dataset
set.seed(123)
mock_adlb <- data.frame(
  ARM = rep(c("Treatment A", "Treatment B"), each = 30),
  AVISIT = rep(c(0, 4, 8), 20),
  AVAL = rnorm(60, mean = 10, sd = 2)
)

# 2. Generate the base line plot
p_base <- gg_lineplot(
```

```
  data = mock_adlb,
  x = AVISIT,
  y = AVAL,
  group = ARM
)

# 3. Annotate with default stats (auto-extracts variables from p_base)
annotate_lineplot_df(gg_plt = p_base, data = mock_adlb)

# 4. Annotate with custom statistics and exactly 2 decimal places
annotate_lineplot_df(
  gg_plt = p_base,
  data = mock_adlb,
  summary_stats = c("n", "median", "iqr"),
  digits = c(0, 2, 2)
)
```

`annotate_pkc_df` *Annotate PK Plot with Summary Table*

Description

These functions provide capabilities to annotate Pharmacokinetics plot (`gg_pkc_lineplot()`) with additional summary statistics table. The annotations are added using the `cowplot` package for flexible placement.

Usage

```
annotate_pkc_df(
  gg_plt,
  data,
  time_var = NULL,
  analyte_var = NULL,
  group = NULL,
  summary_stats = c("n", "mean", "sd"),
  digits = NULL,
  text_size = 3.5,
  rel_height_plot = 0.75
)
```

Arguments

`gg_plt`	(ggplot2) The PK plot.
`data`	(`data.frame`) The raw or summarized dataset used to generate the table.

`time_var`, `analyte_var`, `group`
: (tidy-select)
Optional. If `NULL` (default), the function automatically extracts these from the `gg_plt` mapping.

`summary_stats`
: (`character`)
A vector of statistics to include. Defaults to `c("n", "mean", "sd")`.

`digits`
: (`numeric`, `list`, or `formula`)
Optional specification for the number of decimal places for the summary statistics. Can be a single integer (e.g., `2`), a vector of integers matching the statistics (e.g., `c(0, 2, 2)`), or a `gtsummary` style formula. Defaults to `NULL` (uses `gtsummary` default auto-formatting).

`text_size`
: (`numeric`)
The font size for the table text. Defaults to `3.5`.

`rel_height_plot`
: (`numeric`)
Relative height of the plot vs the table. Defaults to `0.75`.

Value

A ggplot2 object: a plot with a table at the bottom.

See Also

`gg_pkc_lineplot()` for related functionalities.

Examples

```
# Prepare PK Data using the built-in Theoph dataset
df_pk <- Theoph
df_pk$Time_Nominal <- round(df_pk$Time)
# Filter to specific timepoints to keep the table clean
df_pk <- df_pk[df_pk$Time_Nominal %in% c(0, 2, 4, 8, 24), ]
# Create a mock treatment group based on Dose
df_pk$Dose_Group <- ifelse(df_pk$Dose > 4.5, "High Dose", "Low Dose")

# Create the Base Plot using actual Theoph column names
p_pk <- gg_pkc_lineplot(
  data = df_pk,
  time_var = Time_Nominal,
  analyte_var = conc,
  group = Dose_Group,
  stat = "mean",
  variability = "sd",
  log_y = FALSE
)

# Annotate the Plot (Auto-detects variables from aesthetic mapping)
annotate_pkc_df(
  data = df_pk,
  gg_plt = p_pk
```

```
)

# Annotate with specific statistics, explicit variable names, and explicit digits
annotate_pkc_df(
  data = df_pk,
  gg_plt = p_pk,
  time_var = "Time_Nominal",
  analyte_var = "conc",
  group = "Dose_Group",
  summary_stats = c("n", "median", "iqr"),
  digits = c(0, 2, 2)
)
```

`ard_tabulate_abnormal_by_baseline` *ARD Tabulate Abnormality by Baseline Status*

Description

This function creates an Analysis Results Data (ARD) object counting participants with abnormal assessments, stratified by their baseline status. For each abnormality (e.g., "Low", "High"), it calculates statistics for three tiers:

1. Patients **Not Abnormal** at baseline.
2. Patients **Abnormal** at baseline.
3. **Total** (all patients with a post-baseline assessment).

Usage

```
ard_tabulate_abnormal_by_baseline(
  data,
  postbaseline,
  baseline,
  abnormal,
  id = "USUBJID",
  by = NULL,
  strata = NULL
)
```

Arguments

`data`	(`data.frame`) A data frame containing the clinical results.
`postbaseline`	(tidy-select) Column name of the post-baseline reference range indicator (e.g., `ANRIND`).
`baseline`	(tidy-select) Column name of the baseline reference range indicator (e.g., `BNRIND`).

abnormal	(list) A named list of abnormalities (e.g., list(Low = c("LOW", "LOW LOW"))). The name is used as the level label, and the vector contains the values in postbaseline and baseline that define the abnormality.
id	(tidy-select) Column name for the subject identifier. Defaults to "USUBJID".
by	(tidy-select) Optional column names for grouping variables (e.g., TRTA).
strata	(tidy-select) Optional column names for additional stratification.

Value

An ARD data frame of class 'card'.

Examples

```
# Example usage with ADLB-like data
adlb <- cards::ADLB
ard_tabulate_abnormal_by_baseline(
  data = adlb,
  postbaseline = LBNRIND,
  baseline = BNRIND,
  abnormal = list(Low = "LOW", High = "HIGH"),
  by = TRTA
) %>% tbl_ard_summary(by = TRTA)
```

---

df_add_poolings *Add custom pooling to an ADaM dataset list - see* **DISCLAIMER**

Description

DISCLAIMER: *this is a risky function. Please consider using* tbl_with_pools() *instead.* This function allows you to create new pooled groups in your ADaM datasets based on specified arm values. You can choose to keep the original unpooled rows or not. **Important Note**: If you choose to keep the original rows and also add a pool that includes all patients (using the "all" keyword), you will end up with duplicate rows in your dataset. This can lead to incorrect patient counts if you later add a total column. Use this option with caution and ensure that you do not add a standard total column later to avoid double-counting.

Usage

```
df_add_poolings(adam_db, pools, arm_var = "TRT01A", keep_original = TRUE)
```

Arguments

adam_db	(list) List of ADaM datasets containing at least the adsl data frame.
pools	(list) Named list of custom pools. Values can be character vectors of arm names, logical expressions wrapped in rlang::expr(), or the keyword "all" to include all patients.
arm_var	(character) String of the arm variable to evaluate and overwrite.
keep_original	(logical) Whether to keep the original unpooled rows. Default is TRUE.

Value

Updated list of ADaM datasets.

See Also

tbl_with_pools() for a safer alternative that creates pooled summaries without modifying the underlying datasets.

Examples

```
# Create a minimal dummy adam_db
adsl <- data.frame(
  USUBJID = c("001", "002", "003", "004", "005"),
  TRT01A = c("Drug A", "Drug A", "Drug B", "Drug C", "Drug C"),
  FLAG = c("Y", "N", "Y", "N", "Y"),
  stringsAsFactors = FALSE
)
adam_db <- list(adsl = adsl)

# Define the requested pools
my_pools <- list(
  "Drugs A and B" = c("Drug A", "Drug B"),
  "All Patients"  = "all"
)

# Example A: Safe pooling (keep_original = FALSE, no "all" pool) -----------------
safe_pools <- list("Drugs A and B" = c("Drug A", "Drug B"))
adam_db_safe <- df_add_poolings(adam_db, pools = safe_pools, keep_original = FALSE)
print(adam_db_safe$adsl)

# Example B: Triggering the warnings (keep_original = TRUE and "all" pool) -------
# This will throw two warnings: one for duplicates, one for the "all" pool.
adam_db_warnings <- df_add_poolings(adam_db, pools = my_pools, keep_original = TRUE)
print(adam_db_warnings$adsl)

# Example C: Complex pooling using logical expressions ---------------------------
complex_pools <- list(
```

```
  "Flagged Patients" = rlang::expr(FLAG == "Y"),
  "Drug A Flagged"   = rlang::expr(TRT01A == "Drug A" & FLAG == "Y")
)

adam_db_complex <- df_add_poolings(adam_db, pools = complex_pools, keep_original = FALSE)
print(adam_db_complex$adsl)


# Example D: Use yaml to define the pools config and run the function ------------
# Creating Dummy Data
adex <- data.frame(
  USUBJID = c("001", "002", "003", "004"),
  AEDECOD = c("Headache", "Nausea", "Fatigue", "Dizziness"),
  stringsAsFactors = FALSE
)
adam_db <- list(adsl = adsl, adex = adex, adsl2 = adsl)

# Define the config as a standard R list
config_to_write <- list(
  df_add_poolings_config = list(
    keep_original = FALSE,
    arm_var = "TRT01A",
    pools = list(
      "Drug A + B" = c("Drug A", "Drug B"),
      "Drug C + B" = c("Drug C", "Drug B"),
      "All Patients" = "all"
    )
  )
)

# Write it to a file (using a temp file for this example)
yaml_path <- tempfile(fileext = ".yaml")
yaml::write_yaml(config_to_write, yaml_path)

# Print out what the physical YAML file looks like
cat("--- Contents of the generated YAML file ---\n")
cat(readLines(yaml_path), sep = "\n")
cat("-------------------------------------------\n\n")

# Read the YAML file back into R
arg_specs <- yaml::read_yaml(yaml_path)

# Extract just the poolings config block
pool_args <- arg_specs$df_add_poolings_config

# Run the function
if (!is.null(pool_args)) {
  adam_db_pooled <- df_add_poolings(
    adam_db       = adam_db,
    pools         = pool_args$pools,
    arm_var       = pool_args$arm_var,
    keep_original = pool_args$keep_original
  )
```

```
}

# View the result
adam_db_pooled$adsl2
```

`get_cox_pairwise_df` *Generate Table of Pairwise Cox-PH and Log-Rank Results*

Description

This function performs pairwise comparisons of treatment arms using the **Cox Proportional Hazards model** and calculates the corresponding **log-rank p-value**. Each comparison tests a non-reference group against a specified reference group.

Usage

```
get_cox_pairwise_df(
  model_formula,
  data,
  arm,
  ref_group = NULL,
  ties = c("exact", "efron", "breslow"),
  test = c("log-rank", "gehan-breslow", "tarone", "peto", "prentice",
    "fleming-harrington", "likelihood-ratio"),
  ...
)
```

Arguments

`model_formula`	(`formula`) A `formula` object specifying the survival model, typically in the form `Surv(time, status) ~ arm + covariates`.
`data`	(`data.frame`) A `data.frame` containing the survival data, including time, status, and the arm variable.
`arm`	(`character`) A single character string specifying the name of the column in `data` that contains the grouping/treatment **arm variable**. This column **must be a factor** for correct stratification and comparison.
`ref_group`	(`character` or `NULL`) A single character string specifying the level of the `arm` variable to be used as the **reference group** for all pairwise comparisons. If `NULL` (the default), the **first unique level** of the `arm` column is automatically selected as the reference group.
`ties`	(`character`) A string specifying the method for tie handling in the Cox model. Must be one of "exact", "efron", or "breslow". Default is "exact".

`test` (character)
A string specifying the type of test to compute the p-value. Must be one of "log-rank", "gehan-breslow" (wilcoxon), "tarone", "peto", "prentice" (modified peto), "fleming-harrington", or "likelihood-ratio".

`...` Additional arguments passed to `survival::coxph()` and `summary()` two arguments are supported: `conf.int` (default `0.95`) to adjust the CI level; `robust = TRUE` to compute robust standard errors;

Details

The function iterates through each non-reference arm, subsets the data to the current arm and the reference arm, and then:

- Fits a Cox model using `survival::coxph()`.
- Computes a p-value, which dispatches to `coin::logrank_test()` for weighted log-rank variants or to a nested `survival::coxph()` LRT for the likelihood-ratio test.

Value

A `data.frame` with one row per comparison arm (stored as rownames). The columns are:

- `HR`: The Hazard Ratio formatted to two decimal places.
- `conf.int` (default `0.95`): Adjusts the confidence interval level. **Note:** Changing this value dynamically updates the corresponding column name in the output (e.g., passing `0.99` renames the column to `"99% CI"`).
- `p-value (<test>)`: The p-value from the selected `test`, where `<test>` is the title-cased test name (e.g., `"p-value (log-rank)"`).

Note

When `robust = TRUE` is specified, the Hazard Ratio and Confidence Intervals are computed using robust sandwich standard errors. However, the p-values across all tests (including the likelihood-ratio test) are calculated using standard, non-robust model variances.

See Also

`annotate_gg_km()`, `gg_km()`, `survival::coxph()`, `coin::logrank_test()`.

Examples

```
# Example data setup (assuming 'time' is event time, 'status'
# is event indicator (1=event), and 'arm' is the treatment group)
# for data handling
library(dplyr)
library(survival)
# Prepare data in a modern dplyr-friendly way
surv_data <- lung |>
  mutate(
    arm = factor(sample(c("A", "B", "C"), n(), replace = TRUE)),
    status = status - 1 # Convert status to 0/1
```

```
  ) |>
  filter(if_all(everything(), ~ !is.na(.)))

formula <- Surv(time, status) ~ arm

# Example 1: Default usage (ties = "exact", test = "log-rank")
results_default <- get_cox_pairwise_df(
  model_formula = formula,
  data = surv_data,
  arm = "arm",
  ref_group = "A"
)
print(results_default)

# Example 2: Using Breslow ties and the Gehan-Breslow test
results_wilcoxon <- get_cox_pairwise_df(
  model_formula = formula,
  data = surv_data,
  arm = "arm",
  ref_group = "A",
  ties = "breslow",
  test = "gehan-breslow"
)
print(results_wilcoxon)

# Example 3: Using Efron ties and the Likelihood-Ratio test
results_lr <- get_cox_pairwise_df(
  model_formula = formula,
  data = surv_data,
  arm = "arm",
  ref_group = "A",
  ties = "efron",
  test = "likelihood-ratio"
)
print(results_lr)
```

---

gg_km *Kaplan-Meier Plot*

Description

This set of functions facilitates the creation of Kaplan-Meier survival plots using `ggplot2`. Use `process_survfit()` to prepare the survival data from a fitted `survfit` object, and then `gg_km()` to generate the Kaplan-Meier plot with various customization options. Additional functions like `annot_surv_med()`, `annot_cox_ph()`, and `annotate_riskdf()` allow for adding summary tables and annotations to the plot.

Usage

```
process_survfit(fit_km, strata_levels = "All", max_time = NULL)

gg_km(
  surv_plot_data,
  lty = NULL,
  lwd = 0.5,
  censor_show = TRUE,
  size = 2,
  max_time = NULL,
  xticks = NULL,
  yval = c("Survival", "Failure"),
  ylim = NULL,
  font_size = 10,
  legend_pos = NULL
)
```

Arguments

fit_km	A fitted Kaplan-Meier object of class survfit.
strata_levels	(string) A single character string used as the strata level if the input fit_km object has no strata (e.g., "All").
max_time	(numeric) A single numeric value defining the **maximum time point** to display on the x-axis.
surv_plot_data	(data.frame) A data frame containing the pre-processed survival data, ready for plotting. This data should be equivalent to the output of process_survfit.
lty	(numeric or NULL) A numeric vector of **line types** (e.g., 1 for solid, 2 for dashed) for the survival curves, or NULL for ggplot2 defaults. The length should match the number of arms/groups.
lwd	(numeric) A single numeric value specifying the **line width** for the survival curves.
censor_show	(logical) A single logical value indicating whether to display **censoring marks** on the plot. Defaults to TRUE.
size	(numeric) A single numeric value specifying the **size** of the censoring marks.
xticks	(numeric or NULL) A numeric vector of explicit **x-axis tick positions**, or a single numeric value representing the **interval** between ticks, or NULL for automatic ggplot2 scaling.
yval	(character) A single character string, either "Survival" or "Failure" to plot the corresponding probability.

ylim	(numeric) A **numeric vector of length 2** defining the lower and upper limits of the y-axis (e.g., c(0, 1)).
font_size	(numeric) A single numeric value specifying the **base font size** for the plot theme elements.
legend_pos	(numeric or NULL) A **numeric vector of length 2** defining the **legend position** as (x, y) coordinates relative to the plot area (ranging from 0 to 1), or NULL for automatic placement.

Details

Data setup assumes "time" is event time, "status" is event indicator (1 represents an event), while "arm" is the treatment group.

Value

The function process_survfit returns a data frame containing the survival curve steps, confidence intervals, and censoring info.

The function gg_km returns a ggplot2 object of the KM plot.

Functions

- process_survfit(): takes a fitted survfit object and processes it into a data frame suitable for plotting a Kaplan-Meier curve with ggplot2. Time zero is also added to the data.
- gg_km(): creates a Kaplan-Meier survival curve, with support for various customizations like censoring marks, Confidence Intervals (CIs), and axis control.

Examples

```
# Data preparation for KM plot
library(survival)
use_lung <- lung
use_lung$arm <- factor(sample(c("A", "B", "C"), nrow(use_lung), replace = TRUE))
use_lung$status <- use_lung$status - 1 # Convert status to 0/1
use_lung <- na.omit(use_lung)

# Fit Kaplan-Meier model
formula <- Surv(time, status) ~ arm
fit_kmg01 <- survfit(formula, use_lung)

# Process survfit data for plotting
surv_plot_data <- process_survfit(fit_kmg01)
head(surv_plot_data)

# Example of making the KM plot
plt_kmg01 <- gg_km(surv_plot_data)

# Confidence Interval as Ribbon
plt_kmg01 +
  ggplot2::geom_ribbon(alpha = 0.3, lty = 0, na.rm = TRUE)
```

```
# Adding Title and Footnotes
plt_kmg01 +
  ggplot2::labs(title = "title", caption = "footnotes")

# Changing xlab and ylab
plt_kmg01 +
  ggplot2::xlab("Another Day") +
  ggplot2::ylab("THE Survival Probability")
```

`gg_lineplot` *Generate a Summary Line Plot from Raw Data*

Description

Calculates summary statistics inline using `ggplot2::stat_summary()`, generating a line plot directly from raw data. Supports configurable central tendencies and dispersion metrics.

Usage

```
gg_lineplot(
  data,
  x,
  y,
  group = NULL,
  stat = c("mean", "median"),
  variability = c("ci", "sd", "se", "iqr", "none"),
  conf_level = 0.95
)
```

Arguments

`data`	(`data.frame`) The raw data frame (e.g., ADaM dataset).
`x`	(tidy-select) Column name for the x-axis timepoints (e.g., `AVISIT`).
`y`	(tidy-select) Column name for the continuous variable to summarize (e.g., `AVAL`).
`group`	(tidy-select) Optional column name for the grouping/treatment variable.
`stat`	(`string`) Primary summary statistic: `"mean"` or `"median"`. Default is `"mean"`.
`variability`	(`string`) Variability measure: `"sd"`, `"se"`, `"ci"`, `"iqr"`, or `"none"`. Default is `"ci"`.
`conf_level`	(`numeric`) Confidence level for error bars when `variability = "ci"` (default: `0.95`).

Value

A ggplot object of class crane_gg_line.

See Also

annotate_lineplot_df() for related functionalities.

Examples

```
set.seed(123)
mock_adlb <- data.frame(
  ARM = rep(c("Treatment A", "Treatment B"), each = 30),
  AVISIT = rep(c(0, 4, 8), 20),
  AVAL = rnorm(60, mean = 10, sd = 2)
)

# 1. Default Plot: Mean with 95% Confidence Intervals
gg_lineplot(
  data = mock_adlb,
  x = AVISIT,
  y = AVAL,
  group = ARM
)

# 2. Median with Interquartile Range (IQR)
gg_lineplot(
  data = mock_adlb,
  x = AVISIT,
  y = AVAL,
  group = ARM,
  stat = "median",
  variability = "iqr"
)

# 3. Ungrouped data with Mean and Standard Deviation +
# Change legend position to top and add horizontal reference line
gg_lineplot(
  data = mock_adlb,
  x = AVISIT,
  y = AVAL,
  group = ARM,
  stat = "mean",
  variability = "sd"
) +
  ggplot2::theme(legend.position = "top") +
  ggplot2::geom_hline(
    yintercept = 30,
    linetype = "dashed",
    color = "gray50"
  )
```

`gg_mmrm_lineplot` *Create MMRM Line Plot*

Description

Generates a standardized line plot for Mixed-Effect Repeated Measures Model (MMRM) analysis. It displays adjusted means (change from baseline) along with either 95% Confidence Intervals (CI) or Standard Errors (SE) over time.

Usage

```
gg_mmrm_lineplot(
  mmrm_df,
  arm,
  visit,
  error_bar = c("ci", "se"),
  dodge_width = 0.15,
  hline = 0,
  legend_pos = c(0.02, 0.02)
)
```

Arguments

`mmrm_df`	(`data.frame`) A tidy data frame containing the MMRM results. Usually the output of `get_mmrm_results()`.
`arm`	(`string`) The column in `mmrm_df` that identifies the treatment arms.
`visit`	(`string`) The column in `mmrm_df` that identifies the visits.
`error_bar`	(`string`) Type of error bars to display. Either `"ci"` (95% Confidence Interval) or `"se"` (Standard Error). Default is `"ci"`.
`dodge_width`	(`numeric`) The amount to jitter the x-axis points to prevent overlapping error bars. Default is `0.15`.
`hline`	(`numeric` or `NULL`) The y-intercept for a horizontal reference line. Set to `NULL` to remove. Default is `0`.
`legend_pos`	(`numeric` vector or `string`) The position of the legend. To place it inside the plot on the bottom left, use `c(0.02, 0.02)`. Can also be standard string positions like `"bottom"`, `"right"`, `"none"`.

Value

A `ggplot` object.

See Also

`get_mmrm_results()` to get the MMRM results, and `tbl_mmrm()` for a summary table of the MMRM results.

Examples

```
library(mmrm)
fv_dt <- mmrm::fev_data |>
  dplyr::mutate(
    ARMCD = factor(ARMCD)
  )
# Fit an MMRM model using the FEV data
fit_mmrm <- mmrm::mmrm(
  formula = FEV1 ~ RACE + SEX + ARMCD * AVISIT + us(AVISIT | USUBJID),
  data = fv_dt
)
mmrm_results <- get_mmrm_results(fit_mmrm, arm = "ARMCD", visit = "AVISIT", conf_level = 0.95)

# Create a plot with SE bars, jittered by 0.2, legend inside bottom-left
my_plot <- gg_mmrm_lineplot(
  mmrm_df = mmrm_results,
  arm = "ARMCD",
  visit = "AVISIT",
  error_bar = "se", # Switch to "ci" for Confidence Intervals
  dodge_width = 0.2, # Jitters the x-axis to prevent overlap
  legend_pos = c(0.02, 0.02)
)

print(my_plot)
```

---

gg_pkc_lineplot *Plot Pharmacokinetic Concentration Time Profile*

Description

Creates a standard Pharmacokinetic (PK) concentration-time profile plot. This function wraps `ggplot2` calls to consistently format PK profiles, handling log transformations, various summary statistics, and variability measures.

Usage

```
gg_pkc_lineplot(
  data,
  time_var,
  analyte_var,
  group,
  stat = c("mean", "median"),
  variability = c("sd", "se", "ci", "iqr", "none"),
```

```
  conf_level = 0.95,
  log_y = TRUE,
  lloq = NA_real_
)
```

Arguments

`data`	(`data.frame`) The dataset containing PK data.
`time_var`	(tidy-select) The time variable (x-axis).
`analyte_var`	(tidy-select) The concentration/analyte variable (y-axis).
`group`	(tidy-select) The grouping/treatment variable.
`stat`	(`string`) Primary summary statistic: `"mean"` or `"median"`. Default is `"mean"`.
`variability`	(`string`) Variability measure: `"sd"`, `"se"`, `"ci"`, `"iqr"`, or `"none"`. Default is `"sd"`.
`conf_level`	(`numeric`) Confidence level for error bars when `variability = "ci"` (default: `0.95`).
`log_y`	(`logical`) Whether to apply log10 scale to the y-axis. Default is `TRUE`.
`lloq`	(`numeric` or `NULL`) Lower Limit of Quantification. Default is `NA_real_`.

Value

A `ggplot` object.

See Also

`annotate_pkc_df()` for related functionalities.

Examples

```
# Prepare PK Data using the built-in Theoph dataset
df_pk <- Theoph
df_pk$Time_Nominal <- round(df_pk$Time)
# Filter to specific timepoints to keep the table clean
df_pk <- df_pk[df_pk$Time_Nominal %in% c(0, 2, 4, 8, 24), ]
# Create a mock treatment group based on Dose
df_pk$Dose_Group <- ifelse(df_pk$Dose > 4.5, "High Dose", "Low Dose")

# Linear Scale Example (Baseline 0 is included)
gg_pkc_lineplot(
  data = df_pk,
  time_var = Time_Nominal,
```

```
  analyte_var = conc,
  group = Dose_Group,
  stat = "mean",
  variability = "sd",
  log_y = FALSE
)

# Log Scale Example (Filter out 0s first to avoid log(0) warnings)
df_pk |>
  dplyr::filter(conc > 0) |>
  gg_pkc_lineplot(
    time_var = Time_Nominal,
    analyte_var = conc,
    group = Dose_Group,
    stat = "mean",
    variability = "se",
    log_y = TRUE,
    lloq = 2.0
  )

# Title, subtitle, axes labels and legend position customization
gg_pkc_lineplot(
  data = df_pk,
  time_var = Time_Nominal,
  analyte_var = conc,
  group = Dose_Group,
  stat = "mean",
  variability = "sd",
  log_y = FALSE
) +
  ggplot2::labs(
    x = "Nominal time (hr)",
    y = "Concentration (ng/mL)",
    title = "Title",
    subtitle = "Subtitle"
  ) +
  ggplot2::theme(
    legend.position = "top"
  )
```

`label_roche` *Formatting percent and p-values*

Description

- `label_roche_pvalue()` returns formatted p-values.
- `label_roche_percent()` returns formatted percent values. This function only formats percentages between 0 and 1.

- `label_roche_ratio()` returns formatted ratios with values below and above a threshold being returned as `< 0.1` and `> 999.9`, for example, when `digits=1`.
- `label_roche_number()` returns formatted numbers.

Usage

```
style_roche_pvalue(
  x,
  big.mark = ifelse(decimal.mark == ",", " ", ","),
  decimal.mark = getOption("OutDec"),
  ...
)

label_roche_pvalue(
  big.mark = ifelse(decimal.mark == ",", " ", ","),
  decimal.mark = getOption("OutDec"),
  ...
)

style_roche_percent(
  x,
  digits = 1,
  prefix = "",
  suffix = "",
  scale = 100,
  big.mark = ifelse(decimal.mark == ",", " ", ","),
  decimal.mark = getOption("OutDec"),
  ...
)

label_roche_percent(
  digits = 1,
  suffix = "",
  scale = 100,
  big.mark = ifelse(decimal.mark == ",", " ", ","),
  decimal.mark = getOption("OutDec"),
  ...
)

style_roche_ratio(
  x,
  digits = 2,
  prefix = "",
  suffix = "",
  scale = 1,
  big.mark = ifelse(decimal.mark == ",", " ", ","),
  decimal.mark = getOption("OutDec"),
  ...
```

```
)

label_roche_ratio(
  digits = 2,
  prefix = "",
  suffix = "",
  scale = 1,
  big.mark = ifelse(decimal.mark == ",", " ", ","),
  decimal.mark = getOption("OutDec"),
  ...
)

style_roche_number(
  x,
  digits = 0,
  big.mark = ifelse(decimal.mark == ",", " ", ","),
  decimal.mark = getOption("OutDec"),
  scale = 1,
  prefix = "",
  suffix = "",
  na = "NE",
  inf = "NE",
  nan = "NE",
  ...
)

label_roche_number(
  digits = 0,
  big.mark = ifelse(decimal.mark == ",", " ", ","),
  decimal.mark = getOption("OutDec"),
  scale = 1,
  prefix = "",
  suffix = "",
  na = "NE",
  inf = "NE",
  nan = "NE",
  ...
)
```

Arguments

x	(numeric) Numeric vector
big.mark	(string) Character used between every 3 digits to separate hundreds/thousands/millions/etc. Default is ",", except when decimal.mark = "," when the default is a space.
decimal.mark	(string) The character to be used to indicate the numeric decimal point. Default is "."

	or `getOption("OutDec")`
`...`	Arguments passed on to `base::format()`
`digits`	(non-negative `integer`) Integer or vector of integers specifying the number of decimals to round `x`. When vector is passed, each integer is mapped 1:1 to the numeric values in `x`
`prefix`	(`string`) Additional text to display before the number.
`suffix`	(`string`) Additional text to display after the number.
`scale`	(scalar `numeric`) A scaling factor: `x` will be multiplied by scale before formatting.
`na`, `inf`, `nan`	(`NA`/`string`) scalar to replace `NA`, infinite, and `NaN` values with. Default is `"NE"` for arguments `na`, `inf`, and `nan` argument.

Value

A character vector of rounded p-values

Examples

```
# p-value formatting
x <- c(0.0000001, 0.123456)

style_roche_pvalue(x)
label_roche_pvalue()(x)

# percent formatting
x <- c(0.0008, 0.9998)

style_roche_percent(x)
label_roche_percent()(x)

# ratio formatting
x <- c(0.0008, 0.8234, 2.123, 1000)

style_roche_ratio(x)
label_roche_ratio()(x)

# number formatting
x <- c(0.0008, 0.8234, 2.123, 1000, NA, Inf, -Inf)

style_roche_number(x)
label_roche_number()(x)
```

modify_header_rm_md *Remove Markdown Syntax from Header*

Description

Remove markdown syntax (e.g. double star for bold, underscore for italic, etc) from the headers and spanning headers of a gtsummary table.

Usage

```
modify_header_rm_md(x, md = "bold", type = "star")
```

Arguments

x	(gtsummary) A gtsummary table
md	(character) Must be one or more of 'bold' and 'italic'. Default is 'bold'.
type	(character) Must be one or more of 'star' and 'underscore'. Default is 'star'.

Value

gtsummary table

Examples

```
tbl_roche_summary(
  data = cards::ADSL,
  include = AGE,
  by = ARM,
  nonmissing = "always"
) |>
  modify_header_rm_md()
```

modify_zero_recode *Zero Count Recode*

Description

Removes the percentage from cells with zero counts. Handles both regular spaces and non-breaking spaces (\u00A0, the HTML   equivalent) that some formatting engines insert.

```
0 (0.0%)      -->  0
0 (0%)        -->  0
0 (NA%)       -->  0
0 / nn (0%)   -->  0 / nn
0/nn (0.0%)   -->  0/nn
0 / 0 (NA%)   -->  0 / 0
```

Usage

```
modify_zero_recode(x)
```

Arguments

x	(gtsummary) a gtsummary table

Details

The function is a wrapper for gtsummary::modify_post_fmt_fun().

```
gtsummary::modify_post_fmt_fun(
  x,
  fmt_fun = \(x) {
    dplyr::case_when(
      # convert "0 (0%)" OR "0 (0.0%)" OR 0 (NA%) to "0"
    str_detect(x, "^0\\s\\((?:0(?:\\.0)?|NA)%\\)$") ~ str_remove(x, pattern = "\\s\\((?:0(?:\\.0)?|NA
    # convert "0 / nn (0%)" OR "0/nn (0.0%)" OR 0/0 (NA%) to "0 / nn" OR "0/nn" OR "0/0"
    str_detect(x, pattern = "^(0 ?/) ?\\d+[^()]* \\((?:0(?:\\.0)?|NA)%\\)$") ~ str_remove(x, pattern =
      .default = x
    )
  },
  columns = gtsummary::all_stat_cols()
)
```

Value

a gtsummary table

Examples

```
trial |>
  dplyr::mutate(trt = factor(trt, levels = c("Drug A", "Drug B", "Drug C"))) |>
  tbl_summary(include = trt) |>
  modify_zero_recode()
```

`reverse_ci` *Reverse Confidence Interval*

Description

Negates and swaps confidence interval bounds. Takes a CI string in the format "(lower%, upper%)" and returns "(-upper%, -lower%)". This is useful when `gtsummary::add_difference_row()` computes `reference - arm` but you need `arm - reference`.

Usage

```
reverse_ci(x)
```

Arguments

`x`	(`character`) A character vector of confidence interval strings in format "(lower%, upper%)".

Value

A character vector with negated and swapped CI bounds.

See Also

`reverse_rate_difference()` for reversing rate difference values.

Examples

```
# Basic usage - negates values and swaps order
reverse_ci(c("(2.5%, 10.0%)", "(-5.0%, 3.0%)"))

# Handles NA and empty strings
reverse_ci(c("(1.0%, 5.0%)", NA, ""))

# Handles negative bounds
reverse_ci("(-8.0%, -2.0%)")

# Example: Reversing direction in a gtsummary table
# When add_difference_row() computes "reference - arm" but you need "arm - reference"
library(gtsummary)

tbl <- trial |>
  tbl_summary(
    by = trt,
    include = response,
    missing = "no"
  ) |>
  add_difference_row(
    include = response,
```

```
    reference = "Drug A",
    statistic = response ~ c("{estimate}", "({conf.low}, {conf.high})"),
    estimate_fun = response ~ label_style_number(digits = 1, scale = 100, suffix = "%")
  )

# Reverse the direction using modify_table_body
tbl |>
  modify_table_body(
    ~ .x |>
      dplyr::mutate(
        dplyr::across(
          dplyr::starts_with("stat_"),
          ~ ifelse(variable == "response-row_difference" & label == "Rate Difference",
            reverse_rate_difference(.x), .x
          )
        )
      ) |>
      dplyr::mutate(
        dplyr::across(
          dplyr::starts_with("stat_"),
          ~ ifelse(variable == "response-row_difference" &
            label == "(CI Lower Bound, CI Upper Bound)",
          reverse_ci(.x), .x
          )
        )
      )
  )
```

`reverse_rate_difference` *Reverse Rate Difference*

Description

Negates numeric rate difference values while preserving any suffix (e.g., "%"). This is useful when `gtsummary::add_difference_row()` computes `reference - arm` but you need `arm - reference`.

Usage

```
reverse_rate_difference(x)
```

Arguments

`x`	(`character`) A character vector of rate difference values, possibly with suffixes like "%".

Value

A character vector with negated numeric values.

See Also

Usually used together with `reverse_ci()` for reversing confidence intervals; see examples there for usage with `gtsummary::modify_table_body()`.

Examples

```
# Basic usage with percentage suffix
reverse_rate_difference(c("5.0%", "-3.2%", "0.0%"))

# Handles NA and empty strings
reverse_rate_difference(c("2.5%", NA, "", "-1.0%"))

# Works with values without suffix
reverse_rate_difference(c("10.0", "-5.5"))
```

`tbl_baseline_chg` *Change from Baseline*

Description

Typical use is tabulating changes from baseline measurement of an Analysis Variable.

Usage

```
tbl_baseline_chg(
  data,
  baseline_level,
  denominator,
  by = NULL,
  digits = NULL,
  statistic = gtsummary::all_continuous() ~ c("{mean} ({sd})", "{median}",
    "{min} - {max}"),
  id = "USUBJID",
  visit = "AVISIT",
  visit_number = "AVISITN",
  analysis_variable = "AVAL",
  change_variable = "CHG"
)

## S3 method for class 'tbl_baseline_chg'
add_overall(
  x,
  last = FALSE,
  col_label = "All Participants  \n(N = {style_roche_number(n)})",
  ...
)
```

Arguments

data	(data.frame) A data frame.
baseline_level	(string) String identifying baseline level in the visit variable.
denominator	(data.frame) Data set used to compute the header counts (typically ADSL).
by	(tidy-select) A single column from data. Summary statistics will be stratified by this variable. Default is NULL.
digits	(formula-list-selector) Specifies how summary statistics are rounded. Values may be either integer(s) or function(s). If not specified, default formatting is assigned via assign_summary_digits(). See below for details.
statistic	(formula) Formula or list of formulas specifying the summary statistics to display. Default is all_continuous() ~ c("{mean} ({sd})","{median}","{min} - {max}")
id	(tidy-select) String identifying the unique subjects. Default is 'USUBJID'.
visit	(tidy-select) String for the visit variable. Default is 'AVISIT'. If there are more than one entry for each visit and subject, only the first row is kept.
visit_number	(tidy-select) String identifying the visit or analysis sequence number. Default is 'AVISITN'.
analysis_variable	(tidy-select) String identifying the analysis values. Default is 'AVAL'.
change_variable	(tidy-select) String identifying the change from baseline values. Default is 'CHG'.
x	(tbl_summary, tbl_svysummary, tbl_continuous, tbl_custom_summary) A stratified 'gtsummary' table
last	(scalar logical) Logical indicator to display overall column last in table. Default is FALSE, which will display overall column first.
col_label	(string) String indicating the column label. Default is "**Overall** \nN = {style_number(N)}"
...	These dots are for future extensions and must be empty.

Value

A gtsummary table.

Examples

```
theme_gtsummary_roche()

df <- cards::ADLB |>
  dplyr::mutate(AVISIT = trimws(AVISIT)) |>
  dplyr::filter(
    AVISIT != "End of Treatment",
    PARAMCD %in% c("SODIUM", "K")
  )

tbl_baseline_chg(
  data = df |> dplyr::filter(PARAMCD == "SODIUM"),
  baseline_level = "Baseline",
  by = "TRTA",
  denominator = cards::ADSL,
  statistic = everything() ~ c("{mean} ({sd})", "{median} ({p25}, {p75})")
)

tbl_baseline_chg(
  data = df |> dplyr::filter(PARAMCD == "K"),
  baseline_level = "Baseline",
  by = "TRTA",
  denominator = cards::ADSL
) |>
  add_overall(last = TRUE, col_label = "All Participants")

# Split by PARAM
tbl_strata(
  data = df,
  strata = PARAMCD,
  .tbl_fun = ~ tbl_baseline_chg(
    data = .x,
    baseline_level = "Baseline",
    by = "TRTA",
    denominator = cards::ADSL
  ),
  .combine_with = "tbl_stack",
  .combine_args = list(group_header = NULL, quiet = TRUE)
) |>
  tbl_split_by_rows(variable_level = ends_with("lbl"))
```

---

tbl_coxph *Pairwise Cox Proportional Hazards Table*

Description

Generates a gtsummary table from the pairwise comparison results created by `get_cox_pairwise_df()`. The table splits the results by comparison arms, presenting the p-value, Hazard Ratio, and 95% Confidence Interval in a stacked layout where statistics form the rows of the table.

Usage

```
tbl_coxph(pairwise_df)
```

Arguments

pairwise_df	(data.frame) The results data.frame output generated by get_cox_pairwise_df(). Must contain rownames (comparison arms) and at least one statistic column: "HR", "95% CI", or "p-value (...)".

Value

A gtsummary object with additional class tbl_coxph.

See Also

get_cox_pairwise_df().

Examples

```
# Setup sample survival data
library(survival)
surv_data <- lung |>
  dplyr::mutate(
    arm = factor(sample(c("A", "B", "C"), dplyr::n(), replace = TRUE)),
    status = status - 1
  ) |>
  dplyr::filter(dplyr::if_all(dplyr::everything(), ~ !is.na(.)))

formula <- Surv(time, status) ~ arm

# Generate the pairwise statistics data.frame
pairwise_results <- get_cox_pairwise_df(
  model_formula = formula,
  data = surv_data,
  arm = "arm",
  ref_group = "A"
)

# Example 1: Full table
tbl_coxph(pairwise_df = pairwise_results)

# Example 2: Table with only HR and CI (p-value removed)
pairwise_no_pval <- pairwise_results[, c("HR", "95% CI"), drop = FALSE]
tbl_coxph(pairwise_df = pairwise_no_pval)

# Example 3: Table with only p-values
pairwise_only_pval <- pairwise_results[, 3, drop = FALSE]
tbl_coxph(pairwise_df = pairwise_only_pval)

# Example 4: Customize p-value precision
```

```
# Pre-format the p-value column as character before passing to tbl_coxph().
# Character values are displayed as-is (no further formatting applied).
pval_col <- grep("p-value", names(pairwise_results), value = TRUE)
custom <- pairwise_results
custom[[pval_col]] <- ifelse(
  custom[[pval_col]] < 0.001, "<0.001",
  sprintf("%.3f", custom[[pval_col]])
)
tbl_coxph(pairwise_df = custom)
```

`tbl_hierarchical_incidence_rate` *Hierarchical Exposure-Adjusted Incidence Rates*

Description

A wrapper function for `gtsummary::tbl_hierarchical()` to calculate exposure-adjusted incidence rates of adverse events (or other clinical events) across a hierarchy.

The function calculates the incidence rate per specified person-time dynamically. For subjects experiencing an event, Person-Years is calculated from `start_date` to `event_date`. For subjects without an event, it is calculated from `start_date` to `end_date`.

Usage

```
tbl_hierarchical_incidence_rate(
  data,
  denominator,
  variables,
  by = NULL,
  id = "USUBJID",
  start_date = "TRTSDT",
  end_date = "TRTEDT",
  event_date = "AESTDTC",
  event_type = c("first_event", "all"),
  n_person_time = 100,
  unit_label = "years",
  conf.level = 0.95,
  conf.type = "normal",
  digits = 2,
  label = NULL
)
```

Arguments

`data`	(`data.frame`) a data frame.

`denominator` (`data.frame`, `integer`)
used to define the denominator and enhance the output. The argument is required for `tbl_hierarchical()` and optional for `tbl_hierarchical_count()`. The `denominator` argument must be specified when `id` is used to calculate event rates.

`variables` (tidy-select)
A character vector or tidy-selector of hierarchical columns in `data` (e.g., system organ class and preferred term).

`by` (tidy-select)
A single column name in `data` to stratify the summary table by (e.g., treatment arm).

`id` (tidy-select)
argument used to subset `data` to identify rows in `data` to calculate event rates in `tbl_hierarchical()`.

`start_date` (tidy-select)
A column name in `denominator` specifying the treatment start date.

`end_date` (tidy-select)
A column name in `denominator` specifying the treatment end date or follow-up cutoff.

`event_date` (tidy-select)
A column name in `data` specifying the onset date of the event.

`event_type` (`string`)
Type of the events to be counted. Can be `"first_event"` or `"all"`. Default is `"first_event"`.

`n_person_time` (`numeric(1)`)
A numeric scalar multiplier to scale the incidence rate. Defaults to `100`.

`unit_label` (`string`)
Label for the unit of estimated person-time output. Defaults to `"years"`. Known abbreviations ("years", "months", "weeks", "days") are parsed into standard acronyms (e.g., "PY"). Custom strings will be formatted to Title Case and prefixed with "Person-" (e.g., `"decades"` becomes `"Person-Decades"`).

`conf.level` (`numeric(1)`)
Confidence level for the calculated interval. Default is `0.95`. Passed directly to `cardx::ard_incidence_rate()`.

`conf.type` (`string`)
Confidence interval type. Default is `"normal"`. Passed directly to `cardx::ard_incidence_rate()`.

`digits` (`numeric(1)`)
An integer specifying the number of decimal places to round the incidence estimates, confidence intervals, and person-years to. Defaults to `2`.

`label` (formula-list-selector)
used to override default labels in hierarchical table, e.g. `list(AESOC = "System Organ Class")`. The default for each variable is the column label attribute, `attr(., 'label')`. If no label has been set, the column name is used.

Value

a gtsummary table of class "tbl_hierarchical_incidence_rate".

Examples

```
# Dummy denominator dataset with treatment start and end dates
adsl <- data.frame(
  USUBJID = paste0("PT", sprintf("%02d", 1:5)),
  ARM = c("Treatment", "Treatment", "Placebo", "Placebo", "Placebo"),
  TRTSDT = as.Date(rep("2023-01-01", 5)),
  TRTEDT = as.Date(c(
    "2023-12-31", "2023-06-30", "2023-12-31", "2023-08-15", "2023-10-31"
  ))
)

# Dummy AE dataset with onset dates (subset to first occurrences)
adae <- data.frame(
  USUBJID = c("PT02", "PT05", "PT05"),
  AESOC = c("Cardiac", "Nervous", "Cardiac"),
  ARM = c("Treatment", "Placebo", "Placebo"),
  AEDECOD = c("Tachycardia", "Headache", "Palpitations"),
  AESTDTC = c("2023-04-15", "2023-09-10", "2023-10-01")
)

# Build the hierarchical incidence rate table
tbl_hierarchical_incidence_rate(
  data = adae,
  denominator = adsl,
  variables = c(AESOC, AEDECOD),
  by = ARM,
  start_date = TRTSDT,
  end_date = TRTEDT,
  event_date = AESTDTC,
  n_person_time = 100,
  unit_label = "years",
  label = list(
    AESOC = "MedDRA System Organ Class",
    AEDECOD = "MedDRA Preferred Term",
    "..ard_hierarchical_overall.." = "All Adverse Events"
  )
)
```

`tbl_hierarchical_rate_and_count`

Hierarchical Rates and Counts

Description

A mix of adverse event rates (from `gtsummary::tbl_hierarchical()`) and counts (from `gtsummary::tbl_hierarchical_`
The function produces additional summary rows for the higher level nesting variables providing both rates and counts.

When a hierarchical summary is filtered, the summary rows no longer provide useful/consistent information. When creating a filtered summary, use `gtsummary::tbl_hierarchical()` or `gtsummary::tbl_hierarchical_` directly, followed by a call to `gtsummary::filter_hierarchical()`.

Usage

```
tbl_hierarchical_rate_and_count(
  data,
  variables,
  denominator,
  by = NULL,
  id = "USUBJID",
  label = NULL,
  digits = NULL,
  sort = NULL,
 label_overall_rate = "Total number of participants with at least one adverse event",
  label_overall_count = "Overall total number of events",
  label_rate = "Total number of participants with at least one adverse event",
  label_count = "Total number of events"
)

## S3 method for class 'tbl_hierarchical_rate_and_count'
add_overall(
  x,
  last = FALSE,
  col_label = "All Participants  \n(N = {style_roche_number(N)})",
  ...
)
```

Arguments

`data`	(`data.frame`) a data frame.
`variables`	(tidy-select) Hierarchical variables to summarize. Must be 2 or 3 variables. Typical inputs are `c(AEBODSYS, AEDECOD)` for an SOC/AE summary or `c(AEBODSYS, AEHLT, AEDECOD)` for an SOC/HLT/AE summary. Variables must be specified in the nesting order.
`denominator`	(`data.frame`, `integer`) used to define the denominator and enhance the output. The argument is required for `tbl_hierarchical()` and optional for `tbl_hierarchical_count()`. The `denominator` argument must be specified when `id` is used to calculate event rates.

by (tidy-select)
a single column from data. Summary statistics will be stratified by this variable. Default is NULL.

id (tidy-select)
argument used to subset data to identify rows in data to calculate event rates in tbl_hierarchical().

label (formula-list-selector)
used to override default labels in hierarchical table, e.g. list(AESOC = "System Organ Class"). The default for each variable is the column label attribute, attr(., 'label'). If no label has been set, the column name is used.

digits (formula-list-selector)
Specifies how summary statistics are rounded. Values may be either integer(s) or function(s). If a theme is applied, the digits specifications of the theme is applied.

sort *Optional* arguments passed to gtsummary::sort_hierarchical(sort).

label_overall_rate
(string)
String for the overall rate summary. Default is "Total number of participants with at least one adverse event".

label_overall_count
(string)
String for the overall count summary. Default is "Overall total number of events".

label_rate (string)
String for the rate summary. Default is "Overall total number of events". "Total number of participants with at least one adverse event".

label_count (string)
String for the overall count summary. Default is "Total number of events".

x (tbl_hierarchical_rate_and_count)
a stratified 'tbl_hierarchical_rate_and_count' table

last (scalar logical)
Logical indicator to display overall column last in table. Default is FALSE, which will display overall column first.

col_label (string)
String indicating the column label. Default is "**Overall** \nN = {style_number(N)}"

... These dots are for future extensions and must be empty.

Details

Factor levels and zero-rows:

When the first variable in variables is a factor, the function respects its levels and emits zero-rows for any levels not observed in the data. Each unobserved level gets a header row (with NA stats), a rate summary row ("0"), and a count summary row ("0").

This is useful when the set of expected categories is predefined but some may not be present in the data. To exclude categories with no observations, drop unused levels before calling the function (e.g., droplevels()).

Ensures consistent output structure for empty datasets, removing the need for manual workarounds in reporting templates.

Value

a gtsummary table

Examples

```
# Example 1 ----------------------------------
cards::ADAE[c(1, 2, 3, 8, 16), ] |>
  tbl_hierarchical_rate_and_count(
    variables = c(AEBODSYS, AEDECOD),
    denominator = cards::ADSL,
    by = TRTA
  ) |>
  add_overall(last = TRUE)

# Example 2: factor with unobserved levels ----------------------------------
# Adding an unobserved SOC level produces zero-rows automatically
cards::ADAE[c(1, 2, 3, 8, 16), ] |>
  dplyr::mutate(
    AEBODSYS = factor(AEBODSYS, levels = c(unique(AEBODSYS), "UNOBSERVED SOC"))
  ) |>
  tbl_hierarchical_rate_and_count(
    variables = c(AEBODSYS, AEDECOD),
    denominator = cards::ADSL,
    by = TRTA
  )
```

---

tbl_hierarchical_rate_by_grade

AE Rates by Highest Toxicity Grade

Description

A wrapper function for `gtsummary::tbl_hierarchical()` to calculate rates of highest toxicity grades with the options to add rows for grade groups and additional summary sections at each variable level.

Only the highest grade level recorded for each subject will be analyzed. Prior to running the function, ensure that the toxicity grade variable (`grade`) is a factor variable, with factor levels ordered lowest to highest.

Grades will appear in rows in the order of the factor levels given, with each grade group appearing prior to the first level in its group.

Usage

```
tbl_hierarchical_rate_by_grade(
  data,
  variables,
  denominator,
  by = NULL,
  id = "USUBJID",
  include_overall = everything(),
  statistic = everything() ~ "{n} ({p}%)",
  label = NULL,
  digits = NULL,
  sort = "alphanumeric",
  filter = NULL,
  grade_groups = list(),
  grades_exclude = NULL,
  keep_zero_rows = FALSE
)

## S3 method for class 'tbl_hierarchical_rate_by_grade'
add_overall(
  x,
  last = FALSE,
  col_label = "**Overall**  \nN = {style_number(N)}",
  statistic = NULL,
  digits = NULL,
  ...
)

add_grade_column(x)
```

Arguments

data	(data.frame) a data frame.
variables	(tidy-select) A character vector or tidy-selector of 3 columns in data specifying a system organ class variable, an adverse event terms variable, and a toxicity grade level variable, respectively.
denominator	(data.frame, integer) used to define the denominator and enhance the output. The argument is required for tbl_hierarchical() and optional for tbl_hierarchical_count(). The denominator argument must be specified when id is used to calculate event rates.
by	(tidy-select) a single column from data. Summary statistics will be stratified by this variable. Default is NULL.

`id` (tidy-select)
argument used to subset `data` to identify rows in `data` to calculate event rates in `tbl_hierarchical()`.

`include_overall` (tidy-select)
Variables from `variables` for which an overall section at that hierarchy level should be computed. An overall section at the SOC variable level will have label `"- Any adverse events -"`. An overall section at the AE term variable level will have label `"- Overall -"`. If the grade level variable is included it has no effect. The default is `everything()`.

`statistic` (formula-list-selector)
used to specify the summary statistics to display for all variables in `tbl_hierarchical()`. The default is `everything() ~ "{n} ({p})"`.

`label` (formula-list-selector)
used to override default labels in hierarchical table, e.g. `list(AESOC = "System Organ Class")`. The default for each variable is the column label attribute, `attr(., 'label')`. If no label has been set, the column name is used.

`digits` (formula-list-selector)
specifies how summary statistics are rounded. Values may be either integer(s) or function(s). If not specified, default formatting is assigned via `label_style_number()` for statistics `n` and `N`, and `label_style_percent(digits=1)` for statistic `p`.

`sort` (formula-list-selector, string)
a named list, a list of formulas, a single formula where the list element is a named list of functions (or the RHS of a formula), or a string specifying the types of sorting to perform at each hierarchy level. If the sort method for any variable is not specified then the method will default to `"descending"`. If a single unnamed string is supplied it is applied to all hierarchy levels. For each variable, the value specified must be one of:

- `"alphanumeric"` - at the specified hierarchy level, groups are ordered alphanumerically (i.e. A to Z) by `variable_level` text.
- `"descending"` - at the specified hierarchy level, count sums are calculated for each row and rows are sorted in descending order by sum. If `sort` is `"descending"` for a given variable and `n` is included in `statistic` for the variable then `n` is used to calculate row sums, otherwise `p` is used. If neither `n` nor `p` are present in `x` for the variable, an error will occur.

Defaults to `everything() ~ "descending"`.

`filter` (expression)
An expression that is used to filter rows of the table. Filter will be applied to the second variable (adverse event terms) specified via `variables`. See the Details section below for more information.

`grade_groups` (named list)
A named list of grade groups for which rates should be calculated. Grade groups must be mutually exclusive, i.e. each grade cannot be assigned to more than one grade group. Each grade group must be specified in the list as a character vector of the grades included in the grade group, named with the corresponding name of the grade group, e.g. `"Grade 1-2" = c("1", "2")`.

`grades_exclude` (character)
A vector of grades to omit individual rows for when printing the table. These grades will still be used when computing overall totals and grade group totals. For example, to avoid duplication, if a grade group is defined as `"Grade 5" = "5"`, the individual rows corresponding to grade 5 can be excluded by setting `grades_exclude = "5"`.

`keep_zero_rows` (logical)
Whether rows containing zero rates across all columns should be kept. If `FALSE`, this filter will be applied prior to any filters specified via the `filter` argument which may still remove these rows. Defaults to `FALSE`.

`x` (gtsummary)
A gtsummary table produced by `tbl_hierarchical_rate_by_grade()`, or a merged table (e.g., from `tbl_with_pools()`) where the underlying tables were produced by `tbl_hierarchical_rate_by_grade()`.

`last` (scalar `logical`)
Logical indicator to display overall column last in table. Default is `FALSE`, which will display overall column first.

`col_label` (`string`)
String indicating the column label. Default is `"**Overall** \nN = {style_number(N)}"`

`...` These dots are for future extensions and must be empty.

Details

This function returns a structurally pristine table where the `label` column retains unique grade text (e.g., "1", "2", "Grade 1-2"). This preserves row uniqueness required by `gtsummary::tbl_merge()` and `tbl_with_pools()`. To apply visual formatting (grade column, label blanking, header styling), pipe the result through `add_grade_column()` **after** any merging operations.

When using the `filter` argument, the filter will be applied to the second variable from `variables`, i.e. the adverse event terms variable. If an AE does not meet the filtering criteria, the AE overall row as well as all grade and grade group rows within an AE section will be excluded from the table. Filtering out AEs does not exclude the records corresponding to these filtered out rows from being included in rate calculations for overall sections. If all AEs for a given SOC have been filtered out, the SOC will be excluded from the table. If all AEs are filtered out and the SOC variable is included in `include_overall` the `- Any adverse events -` section will still be kept.

See `gtsummary::filter_hierarchical()` for more details and examples.

`add_grade_column()`:

Post-processing function that applies visual formatting to tables generated by `tbl_hierarchical_rate_by_grade()`. Must be called **after** any merging (e.g., via `tbl_with_pools()`) to avoid Cartesian join explosions caused by blanking the `label` column prior to merge.

The function extracts metadata injected by `tbl_hierarchical_rate_by_grade()` via `x$custom_info` (standalone tables) or the first sub-table's `custom_info` (merged tables). If no metadata is found, the function aborts with an informative error.

`add_grade_column()` only works on tables produced by `tbl_hierarchical_rate_by_grade()` — it reads the `custom_info` metadata that function stores. They share a help page because they are designed to be used together: build the table first, optionally merge with `gtsummary::tbl_merge()` or `tbl_with_pools()`, then call `add_grade_column()` as the final step.

Value

a gtsummary table of class "tbl_hierarchical_rate_by_grade".

Examples

```
theme_gtsummary_roche()
ADSL <- cards::ADSL
ADAE_subset <- cards::ADAE |>
  dplyr::filter(
    AESOC %in% unique(cards::ADAE$AESOC)[1:5],
    AETERM %in% unique(cards::ADAE$AETERM)[1:10]
  )

grade_groups <- list(
  "Grade 1-2" = c("1", "2"),
  "Grade 3-4" = c("3", "4"),
  "Grade 5" = "5"
)

# Example 1 ----------------------------------
tbl_hierarchical_rate_by_grade(
  ADAE_subset,
  variables = c(AEBODSYS, AEDECOD, AETOXGR),
  denominator = ADSL,
  by = TRTA,
  label = list(
    AEBODSYS = "MedDRA System Organ Class",
    AEDECOD = "MedDRA Preferred Term",
    AETOXGR = "Grade"
  ),
  grade_groups = grade_groups,
  grades_exclude = "5"
) |>
  add_grade_column()

# Example 2 ----------------------------------
# Filter: Keep AEs with an overall prevalence of greater than 10%
tbl_hierarchical_rate_by_grade(
  ADAE_subset,
  variables = c(AEBODSYS, AEDECOD, AETOXGR),
  denominator = ADSL,
  by = TRTA,
  grade_groups = list("Grades 1-2" = c("1", "2"), "Grades 3-5" = c("3", "4", "5")),
  filter = sum(n) / sum(N) > 0.10
) |>
  add_overall(last = TRUE) |>
  add_grade_column()
```

tbl_listing *Create listings from a data frame*

Description

This function creates a listing from a data frame. Common uses rely on few pre-processing steps, such as ensuring unique values in key columns or split by rows or columns. They are described in the note section.

Usage

```
tbl_listing(
  data,
  split_by_rows = list(),
  split_by_columns = list(),
  add_blank_rows = list()
)

remove_duplicate_keys(x, keys = NULL, value = NA)
```

Arguments

`data` (`data.frame`)
a data frame containing the data to be displayed in the listing.

`split_by_rows`, `split_by_columns`, `add_blank_rows`
(named `list`)

- `split_by_rows`: Named list of arguments that are passed to `gtsummary::tbl_split_by_rows()`.
- `split_by_columns`: Named list of arguments that are passed to `gtsummary::tbl_split_by_colum`
- `add_blank_rows`: Named list of arguments that are passed to `crane::add_blank_rows()`. `add_blank_rows()` is applied after table splitting and applied to each table individually.

Variable names passed in these named lists must be character vectors; tidyselect/unquoted syntax is not accepted.

`x` (`tbl_listing` or `list`)
a `tbl_listing` object or a list of `tbl_listing` objects.

`keys` (tidy-select)
columns to highlight for duplicate values. If `NULL`, nothing is done.

`value` (`string`)
string to use for blank values. Defaults to `NA`. It should not be changed.

Value

A table listing of class "tbl_listing".

Note

Common pre-processing steps for the data frame that may be common:

- Unique values - this should be enforced in pre-processing by users.
- NA values - they are not printed by default in {gtsummary}. You can make them explicit if they need to be displayed in the listing. See example 3.
- Sorting key columns and moving them to the front. See the examples pre-processing.

Splitting the listing:

- Split by rows - you can split the data frame by rows by using split_by_rows parameter. You can use the same parameters used in gtsummary::tbl_split_by_rows(). See example 4.
- Split by columns - you can split the data frame by columns by using split_by_columns parameter. Use the same parameters from gtsummary::tbl_split_by_rows(). See example 5.

Examples

```
# Load the trial dataset
trial_data <- trial |>
  dplyr::select(trt, age, marker, stage) |>
  dplyr::filter(stage %in% c("T2", "T3")) |>
  dplyr::slice_head(n = 2, by = c(trt, stage)) |> # downsampling
  dplyr::arrange(trt, stage) |> # key columns should be sorted
  dplyr::relocate(trt, stage) # key columns should be first

# Example 1 ---------------------------------
out <- tbl_listing(trial_data)
out
out |> remove_duplicate_keys(keys = "trt")

# Example 2 ---------------------------------
# make NAs explicit
trial_data_na <- trial_data |>
  mutate(across(everything(), ~ tidyr::replace_na(labelled::to_character(.), "-")))
tbl_listing(trial_data_na)

# Example 3 ---------------------------------
# Add blank rows for first key column
lst <- tbl_listing(trial_data_na, add_blank_rows = list(variable_level = "trt"))
lst

# Can add them also manually in post-processing
lst |> add_blank_rows(row_numbers = seq(2))

# Example 4 ---------------------------------
# Split by rows
list_lst <- tbl_listing(trial_data, split_by_rows = list(row_numbers = c(2, 3, 4)))
list_lst[[2]]

# Example 5 ---------------------------------
# Split by columns
```

```
show_header_names(lst)
grps <- list(c("trt", "stage", "age"), c("trt", "stage", "marker"))
list_lst <- tbl_listing(trial_data, split_by_columns = list(groups = grps))
list_lst[[2]]

# Example 6 --------------------------------
# Split by rows and columns
list_lst <- tbl_listing(trial_data,
  split_by_rows = list(row_numbers = c(2, 3, 4)), split_by_columns = list(groups = grps)
)
length(list_lst) # 8 tables are flatten out
list_lst[[2]]

# Example 7 --------------------------------
# Hide duplicate columns in post-processing
out <- list_lst |>
  remove_duplicate_keys(keys = c("trt", "stage"))
out[[2]]
```

`tbl_mmrm` *Get and display MMRM Results in a Formatted Table*

Description

These functions take a fitted MMRM model object and creates a formatted table, following the style of the MMRM template. It combines baseline summary statistics (if available) with the MMRM results, presenting them in a clear and organized manner.

Usage

```
get_mmrm_results(fit_mmrm, arm, visit, conf_level = 0.95)

tbl_mmrm(
  mmrm_df,
  base_df = NULL,
  arm,
  visit,
  baseline_aval = NULL,
  digits = c(2, 3, 4)
)
```

Arguments

`fit_mmrm`	(`mmrm` model object) A fitted MMRM model object, typically created using the `mmrm` function from the `mmrm` package. This object should contain the necessary information to extract adjusted means, differences, confidence intervals, and p-values for the specified visits and arms.

arm (string)
The column in mmrm_df and base_df that identifies the treatment arms. This will be used to divide the results in columns. First value is reference.

visit (string)
The column in mmrm_df and base_df that identifies the visits. This will be used to stratify the results in rows.

conf_level (numeric)
The confidence level to use when calculating confidence intervals for the adjusted means and differences. Default is 0.95 for 95% confidence intervals.

mmrm_df (data.frame)
A tidy data frame containing the MMRM results. This should include columns for the visit, arm, adjusted means, differences, confidence intervals, and p-values. The data frame should be structured in a way that allows for stratification by visit and arm. Usually the output of get_mmrm_results().

base_df (data.frame)
A data frame containing baseline measurements. This should include columns for the visit, arm, and baseline values. The function will summarize this data to provide baseline statistics in the final table. If base_df is not provided or does not contain any rows, the function will simply omit the baseline summary section from the final table.

baseline_aval (tidy-select)
The column in base_df that contains the baseline values to be summarized. Unused if base_df is not provided or does not contain any rows.

digits (numeric)
A numeric vector of length 3 specifying the number of decimal places for: 1) Estimates/CIs, 2) Standard Errors, and 3) P-values. Default is c(2, 3, 4).

Value

A data.frame containing the estimated marginal means (adjusted means) and contrasts (differences in adjusted means) for each visit and arm, along with their standard errors, confidence intervals, degrees of freedom, and sample sizes. This data frame is structured to facilitate the creation of a formatted table using tbl_mmrm().

tbl_mmrm returns a 'gtsummary' table object.

See Also

gg_mmrm_lineplot() for visualizing MMRM results.

Examples

```
library(mmrm)
fv_dt <- mmrm::fev_data |>
  dplyr::mutate(
    ARMCD = sprintf(
      "%s\n(N = %d)", ARMCD,
      table(mmrm::fev_data$ARMCD)[ARMCD]
    ),
```

```
    ARMCD = factor(ARMCD)
  )


# Fit an MMRM model using the FEV data
fit_mmrm <- mmrm::mmrm(
  # us -> unstructured cov structure
  formula = FEV1 ~ RACE + SEX + ARMCD * AVISIT + us(AVISIT | USUBJID),
  data = fv_dt
)
mmrm_results <- get_mmrm_results(fit_mmrm, arm = "ARMCD", visit = "AVISIT", conf_level = 0.95)


tbl_mmrm(
  mmrm_results,
  fv_dt |> dplyr::mutate(AVISIT = "Baseline"),
  arm = "ARMCD", visit = "AVISIT", baseline_aval = "FEV1"
)
```

`tbl_null_report` *Creates null report*

Description

This function creates a null report for tables or listings without any statistics.

Usage

```
tbl_null_report(
  label = "No observations met the reporting criteria for this output."
)
```

Arguments

`label`	(`string`) label to display in the header of the null report. It defaults to "No observations met the reporting criteria for this output."

Value

A gtsummary object of class tbl_null_report.

Examples

```
tbl_null_report(label = "No data available for the selected criteria.")
```

`tbl_rmpt` *Risk Management Plan Table (RMPT)*

Description

Creates a summary table showing participant counts and person-time exposure across categories of exposure duration. The table displays both:

- Number and percentage of participants in each exposure duration category
- Total person-time (sum of exposure durations) for each category

By default, the table does not stratify by treatment arms. Please refer to the RMP Best Practice documents for guidance.

Total person-time is computed by summing up the exposure duration (e.g., `AVAL`) across all participants within each category. The unit can be days, months or years depending on the use-case.

Usage

```
tbl_rmpt(
  data,
  variable,
  aval,
  by = NULL,
  id = "USUBJID",
  denominator,
  label = "Duration of exposure"
)

## S3 method for class 'tbl_rmpt'
add_overall(
  x,
  last = FALSE,
  col_label = "All Participants  \n(N = {style_roche_number(n)})",
  ...
)
```

Arguments

`data`	(`data.frame`) A data frame containing the exposure data. Typically ADEX dataset filtered to one row per subject (e.g., `PARAMCD == 'TDURD'` and `PARCAT1 == 'OVERALL'`).
`variable`	(tidy-select) Categorical variable defining exposure duration categories (e.g., `AVAL_CAT`). This variable should be a factor with ordered levels.
`aval`	(tidy-select) Continuous variable containing exposure duration values (e.g., `AVAL`). Used to calculate person-time by summing across participants.

by	(tidy-select) Variable to report results by. Typically the treatment arm (e.g., TRT01A or TRTA). Default is NULL for unstratified analysis.
id	(tidy-select) String identifying the unique subjects. Default is 'USUBJID'.
denominator	(data.frame) Data set used to compute the header counts and percentages (typically ADSL). Should contain columns for id and by variables.
label	(string) Label for the exposure duration variable that appears in the table header. Default is "Duration of exposure".
x	(tbl_rmpt) Object of class 'tbl_rmpt'.
last	(scalar logical) Logical indicator to display overall column last in table. Default is FALSE, which will display overall column first.
col_label	(string) String indicating the column label for overall column. Default is "All Participants \n(N = {style_roche_number(n)})".
...	These dots are for future extensions and must be empty.

Value

A gtsummary table.

Examples

```
# Create example exposure data

df_adsl <- pharmaverseadam::adsl |> dplyr::filter(SAFFL == "Y")
df_adex <- pharmaverseadam::adex |>
  dplyr::filter(PARAMCD == "TDURD", PARCAT1 == "OVERALL", SAFFL == "Y") |>
  dplyr::mutate(
    AVAL_MONTH = AVAL / 30.4375,
    AVAL_CAT = factor(
      dplyr::case_when(
        AVAL_MONTH < 1 ~ "< 1 month",
        AVAL_MONTH >= 1 & AVAL_MONTH < 3 ~ "1 to <3 months",
        AVAL_MONTH >= 3 & AVAL_MONTH < 6 ~ "3 to <6 months",
        TRUE ~ ">=6 months"
      ),
      levels = c("< 1 month", "1 to <3 months", "3 to <6 months", ">=6 months")
    )
  ) |>
  dplyr::select(
    USUBJID,
    SEX,
    ETHNIC,
    RACE,
```

```
    AGEGR1,
    AVAL,
    AVAL_MONTH,
    AVAL_CAT,
    TRT01A
  )
# Example 1 ---------------------------------
# Create basic RMPT table
tbl_rmpt(
  data = df_adex,
  variable = AVAL_CAT,
  aval = AVAL,
  by = TRT01A,
  denominator = df_adsl
)

# Example 2 ---------------------------------
# Add overall column at the end

tbl_rmpt(
  data = df_adex,
  variable = AVAL_CAT,
  aval = AVAL,
  by = TRT01A,
  denominator = df_adsl
) |>
  add_overall(last = TRUE)

# Example 3 ---------------------------------
# RMPT table for other variables (age group and sex), add label
tbl_rmpt(
  data = df_adex,
  variable = AGEGR1,
  aval = AVAL,
  by = SEX,
  denominator = df_adsl,
  label = "Treatment Exposure Duration"
)
```

`tbl_roche_subgroups` *Subgroup Analyses*

Description

Function adapted from `gtforester::tbl_subgroups()`.

Usage

```
tbl_roche_subgroups(data, rsp, by, subgroups, .tbl_fun, time_to_event = NULL)
```

Arguments

data	(data.frame, survey.design) a data frame or survey object
rsp	(tidy-select) Variable to use in responder rate calculations.
by	(tidy-select) Variable to make comparison between groups.
subgroups	(tidy-select) Variables to perform stratified analyses for.
.tbl_fun	(function) A function or formula. If a *function*, it is used as is. If a formula, e.g. ~ .x %>% tbl_summary() %>% add_p(), it is converted to a function. The stratified data frame is passed to this function.
time_to_event	(tidy-select) Variable to use in time-to-event analyses. If specified, the mid table shows n (subjects), Events (from rsp), and Median per arm instead of responder rates. The total column shows Total Events instead of Total n.

Value

a 'gtsummary' table

Examples

```
set.seed(1)

# prepare sample data
df_adtte <- data.frame(
  time = rexp(100, rate = 0.1),
  status = sample(c(0, 1), 100, replace = TRUE),
  arm = sample(c("Arm A", "Arm B"), 100, replace = TRUE),
  grade = sample(c("I", "II"), 100, replace = TRUE),
  strata = sample(c("1", "2"), 100, replace = TRUE)
) |>
  mutate(arm = relevel(factor(arm), ref = "Arm A")) # Set Reference

# logistic regression ------------------------------------------------------
df_adtte |>
  tbl_roche_subgroups(
    rsp = "status",
    by = "arm",
    subgroups = c("grade"),
    .tbl_fun =
      ~ glm(status ~ arm, data = .x) |>
        tbl_regression(
          show_single_row = arm,
          exponentiate = TRUE # , tidy_fun = broom.helpers::tidy_parameters
        )
  ) |>
  modify_header(starts_with("estimate") ~ "**Odds Ratio**")
```

```
# coxph regression ------------------------------------------------------------
# please use browser() inside .tbl_fun to check if the coxph model throws an error
# and use tryCatch to modify the input/output accordingly

df_adtte |>
  tbl_roche_subgroups(
    rsp = status,
    by = arm,
   time_to_event = time, # Specify time variable for time-to-event analyses (different mid table)
    subgroups = c(grade, strata),
    ~ survival::coxph( # Please use coxph for time-to-event analyses
      survival::Surv(time, status) ~ arm,
      data = .x,
      ties = "exact"
    ) |> # Exact Ties
      tbl_regression(
        show_single_row = arm,
        exponentiate = TRUE # Get Hazard Ratios
      )
  ) |>
  modify_header(starts_with("estimate") ~ "**Hazard Ratio**")
```

`tbl_roche_summary` *Roche Summary Table*

Description

This is a thin wrapper of `gtsummary::tbl_summary()` with the following differences:

- Default summary type for continuous variables is `'continuous2'`.
- Number of non-missing observations, when requested, is added for each variable and placed on the row under the variable label/header.
- The `tbl_summary(missing*)` arguments have been renamed to `tbl_roche_summary(nonmissing*)` with updated default values.
- The default footnotes from `tbl_summary()` are removed.
- Cells with `"0 (0.0%)"` are converted to `"0"` with `gtsummary::modify_post_fmt_fun()`.

Usage

```
tbl_roche_summary(
  data,
  by = NULL,
  label = NULL,
  statistic = list(gtsummary::all_continuous() ~ c("{mean} ({sd})", "{median}",
```

```
    "{min} - {max}"), gtsummary::all_categorical() ~ "{n} ({p}%)"),
  digits = NULL,
  type = NULL,
  value = NULL,
  nonmissing = c("no", "always", "ifany"),
  nonmissing_text = "n",
  nonmissing_stat = "{N_nonmiss}",
  sort = gtsummary::all_categorical(FALSE) ~ "alphanumeric",
  percent = c("column", "row", "cell"),
  include = everything()
)
```

Arguments

`data` (`data.frame`)
A data frame.

`by` (tidy-select)
A single column from `data`. Summary statistics will be stratified by this variable. Default is `NULL`.

`label` (formula-list-selector)
Used to override default labels in summary table, e.g. `list(age = "Age, years")`. The default for each variable is the column label attribute, `attr(., 'label')`. If no label has been set, the column name is used.

`statistic` (formula-list-selector)
Specifies summary statistics to display for each variable. The default is `list(all_continuous() ~ "{median} ({p25}, {p75})", all_categorical() ~ "{n} ({p}%)")`. See below for details.

`digits` (formula-list-selector)
Specifies how summary statistics are rounded. Values may be either integer(s) or function(s). If not specified, default formatting is assigned via `assign_summary_digits()`. See below for details.

`type` (formula-list-selector)
Specifies the summary type. Accepted value are `c("continuous", "continuous2", "categorical", "dichotomous")`. If not specified, default type is assigned via `assign_summary_type()`. See below for details.

`value` (formula-list-selector)
Specifies the level of a variable to display on a single row. The gtsummary type selectors, e.g. `all_dichotomous()`, cannot be used with this argument. Default is `NULL`. See below for details.

`nonmissing`, `nonmissing_text`, `nonmissing_stat`
Arguments dictating how and if missing values are presented:

- `nonmissing`: must be one of `c("always", "ifany", "no")`
- `nonmissing_text`: string indicating text shown on non-missing row. Default is `"n"`
- `nonmissing_stat`: statistic to show on non-missing row. Default is `"{N_nonmiss}"`. Possible values are `N_nonmiss`, `N_miss`, `N_obs`, `p_nonmiss` `p_miss`.

sort	(formula-list-selector) Specifies sorting to perform for categorical variables. Values must be one of c("alphanumeric", "frequency"). Default is all_categorical(FALSE) ~ "alphanumeric".
percent	(string) Indicates the type of percentage to return. Must be one of c("column", "row", "cell"). Default is "column". In rarer cases, you may need to define/override the typical denominators. In these cases, pass an integer or a data frame. Refer to the ?cards::ard_tabulate(denominator) help file for details. When a data frame is passed, this data frame is used to calculate header counts.
include	(tidy-select) Variables to include in the summary table. Default is everything().

Value

a 'gtsummary' table

Examples

```
# Example 1 ----------------------------------
trial |>
  tbl_roche_summary(
    by = trt,
    include = c(age, grade),
    nonmissing = "always"
  ) |>
  add_overall()
```

---

tbl_shift *Shift Table*

Description

Typical use is tabulating post-baseline measurement stratified by the baseline measurement.

Usage

```
tbl_shift(
  data,
  variable,
  strata = NULL,
  by = NULL,
  data_header = NULL,
  strata_location = c("new_column", "header"),
  strata_label = "{strata}",
  header = "{level}  \nN = {n}",
```

```
  label = NULL,
  nonmissing = "always",
  nonmissing_text = "Total",
  ...
)

## S3 method for class 'tbl_shift'
add_overall(
  x,
  col_label = "All Participants  \n(N = {style_roche_number(n)})",
  last = FALSE,
  ...
)
```

Arguments

`data`	(`data.frame`) A data frame.
`variable`	(tidy-select) Variable to tabulate. Typically the post-baseline grade.
`strata`	(tidy-select) Stratifying variable. Typically the baseline grade.
`by`	(tidy-select) Variable to report results by. Typical value is the treatment arm.
`data_header`	(`data.frame`) Data frame used to calculate the Ns in the table header. Only include the columns needed to merge with `data`: these are typically the `'USUBJID'` and the treatment arm only, e.g `ADSL[c("USUBJID", "ARM")]`.
`strata_location`	(`string`) Specifies the location where the individual stratum levels will be printed. Must be one of `c("new_column", "header")`. `"new_column"`: stratum labels are placed in a new column to the left of the tabulated results. `"header"`: stratum labels are placed in a header row above the tabulations.
`strata_label`	(`string`) A glue-string that inserts stratum level. Default is `'{strata}'`, and `{n}` is also available to insert.
`header`	(`string`) String that is passed to `gtsummary::modify_header(all_stat_cols() ~ header)`.
`label`	(formula-list-selector) Used to specify the labels for the `strata` and `variable` columns. Default is to use the column label attribute.
`nonmissing, nonmissing_text, ...`	Argument passed to `tbl_roche_summary()`. See details below for call details to `tbl_roche_summary()`.
`x`	(`tbl_shift`) Object of class `'tbl_shift'`.

col_label (string)
String indicating the column label. Default is "All Participants \nN = {style_roche_number(n)}"

last (scalar logical)
Logical indicator to display overall column last in table. Default is FALSE, which will display overall column first.

Details

Broadly, this function is a wrapper for chunk below with some additional calls to gtsummary::modify_*() function to update the table's headers, indentation, column alignment, etc.

```
gtsummary::tbl_strata2(
  data = data,
  strata = strata,
   ~ tbl_roche_summary(.x, include = variable, by = by)
)
```

Value

a 'gtsummary' table

Examples

```
library(dplyr, warn.conflicts = FALSE)

# subsetting ADLB on one PARAM, and the highest grade
adlb <- pharmaverseadam::adlb |>
 dplyr::select("USUBJID", "TRT01A", "PARAM", "PARAMCD", "ATOXGRH", "BTOXGRH", "VISITNUM") |>
  mutate(TRT01A = factor(TRT01A)) |>
  dplyr::filter(PARAMCD %in% c("CHOLES", "GLUC")) |>
  slice_max(by = c(USUBJID, PARAMCD), order_by = ATOXGRH, n = 1L, with_ties = FALSE) |>
  labelled::set_variable_labels(
    BTOXGRH = "Baseline  \nNCI-CTCAE Grade",
    ATOXGRH = "Post-baseline  \nNCI-CTCAE Grade"
  )
adsl <- pharmaverseadam::adsl[c("USUBJID", "TRT01A")] |>
  dplyr::filter(TRT01A != "Screen Failure")

# Example 1 ---------------------------------
# tabulate baseline grade by worst grade
tbl_shift(
  data = dplyr::filter(adlb, PARAMCD %in% "CHOLES"),
  strata = BTOXGRH,
  variable = ATOXGRH,
  by = TRT01A,
  data_header = adsl
)

# Example 2 ---------------------------------
# same as Ex1, but with the stratifying variable levels in header rows
adlb |>
```

```
  dplyr::filter(PARAMCD %in% "CHOLES") |>
  labelled::set_variable_labels(
    BTOXGRH = "Baseline NCI-CTCAE Grade",
    ATOXGRH = "Post-baseline NCI-CTCAE Grade"
  ) |>
  tbl_shift(
    data = ,
    strata = BTOXGRH,
    variable = ATOXGRH,
    strata_location = "header",
    by = TRT01A,
    data_header = adsl
  )

# Example 3 ---------------------------------
# same as Ex2, but with two labs
adlb |>
  labelled::set_variable_labels(
    BTOXGRH = "Baseline NCI-CTCAE Grade",
    ATOXGRH = "Post-baseline NCI-CTCAE Grade"
  ) |>
  tbl_strata_nested_stack(
    strata = PARAM,
    ~ .x |>
      tbl_shift(
        strata = BTOXGRH,
        variable = ATOXGRH,
        strata_location = "header",
        by = TRT01A,
        data_header = adsl
      )
  ) |>
  # Update header with Lab header and indentation (the '\U00A0' character adds whitespace)
  modify_header(
    label = "Lab  \n\U00A0\U00A0\U00A0\U00A0
            Baseline NCI-CTCAE Grade  \n\U00A0\U00A0\U00A0\U00A0\U00A0\U00A0\U00A0\U00A0\U00A0
             Post-baseline NCI-CTCAE Grade"
  )

# Example 4 ---------------------------------
# Include the treatment variable in a new column
filter(adlb, PARAMCD %in% "CHOLES") |>
  right_join(
    pharmaverseadam::adsl[c("USUBJID", "TRT01A")] |>
      dplyr::filter(TRT01A != "Screen Failure"),
    by = c("USUBJID", "TRT01A")
  ) |>
  tbl_shift(
    strata = TRT01A,
    variable = BTOXGRH,
    by = ATOXGRH,
    header = "{level}",
    strata_label = "{strata}, N={n}",
```

```
  label = list(TRT01A = "Actual Treatment"),
  percent = "cell",
  nonmissing = "no"
) |>
modify_spanning_header(all_stat_cols() ~ "Worst Post-baseline NCI-CTCAE Grade")
```

`tbl_survfit_quantiles` *Survival Quantiles*

Description

Create a gtsummary table with Kaplan-Meier estimated survival quantiles. If you must further customize the way these results are presented, see the Details section below for the full details.

Usage

```
tbl_survfit_quantiles(
  data,
  y = "survival::Surv(time = AVAL, event = 1 - CNSR, type = 'right', origin = 0)",
  by = NULL,
  header = "Time to event",
  estimate_fun = label_roche_number(digits = 1, na = "NE"),
  method.args = list(conf.int = 0.95, conf.type = "plain")
)

## S3 method for class 'tbl_survfit_quantiles'
add_overall(
  x,
  last = FALSE,
  col_label = "All Participants  \nN = {style_roche_number(N)}",
  ...
)
```

Arguments

`data`	(`data.frame`) A data frame
`y`	(`string` or `expression`) A string or expression with the survival outcome, e.g. `survival::Surv(time, status)`. The default value is `survival::Surv(time = AVAL, event = 1 - CNSR, type = "right", origin = 0)`.
`by`	(tidy-select) A single column from `data`. Summary statistics will be stratified by this variable. Default is `NULL`, which returns results for the unstratified model.
`header`	(`string`) String for the header of the survival quantile chunks. Default is `"Time to event"`.

estimate_fun	(function) Function used to round and format the estimates in the table. Default is label_roche_number(digits = 1).
method.args	(named list) Named list of arguments that will be passed to survival::survfit(). Note that this list may contain non-standard evaluation components, and must be handled similarly to tidyselect inputs by using rlang's embrace operator {{ . }} or !!enquo() when programming with this function.
x	(tbl_survfit_quantiles) A stratified 'tbl_survfit_quantiles' object.
last	(scalar logical) Logical indicator to display overall column last in table. Default is FALSE, which will display overall column first.
col_label	(string) String indicating the column label. Default is "**Overall** \nN = {style_number(N)}"
...	These dots are for future extensions and must be empty.

Value

a gtsummary table

ARD-first

This function is a helper for creating a common summary. But if you need to modify the appearance of this table, you may need to build it from ARDs.

Here's the general outline for creating this table directly from ARDs.

1. Create an ARD of survival quantiles using cardx::ard_survival_survfit().
2. Construct an ARD of the minimum and maximum survival times using cards::ard_summary().
3. Combine the ARDs and build summary table with gtsummary::tbl_ard_summary().

```
# get the survival quantiles with 95% CI
ard_surv_quantiles <-
  cardx::ard_survival_survfit(
    x = cards::ADTTE,
   y = survival::Surv(time = AVAL, event = 1 - CNSR, type = 'right', origin = 0),
    variables = "TRTA",
    probs = c(0.25, 0.50, 0.75)
  ) |>
  # modify the shape of the ARD to look like a
  # 'continuous' result to feed into `tbl_ard_summary()`
  dplyr::mutate(
    stat_name = paste0(.data$stat_name, 100 * unlist(.data$variable_level)),
    variable_level = list(NULL)
  )

# get the min/max followup time
```

```
ard_surv_min_max <-
  cards::ard_summary(
    data = cards::ADTTE,
    variables = AVAL,
    by = "TRTA",
    statistic = everything() ~ cards::continuous_summary_fns(c("min", "max"))
  )

# stack the ARDs and pass them to `tbl_ard_summary()`
cards::bind_ard(
  ard_surv_quantiles,
  ard_surv_min_max
) |>
  tbl_ard_summary(
    by = "TRTA",
    type = list(prob = "continuous2", AVAL = "continuous"),
    statistic = list(
    prob = c("{estimate50}", "({conf.low50}, {conf.high50})", "{estimate25}, {estimate75}"),
      AVAL = "{min} to {max}"
    ),
    label = list(
      prob = "Time to event",
      AVAL = "Range"
    )
  ) |>
  # directly modify the labels in the table to match spec
  modify_table_body(
    ~ .x |>
      dplyr::mutate(
        label = dplyr::case_when(
          .data$label == "Survival Probability" ~ "Median",
          .data$label == "(CI Lower Bound, CI Upper Bound)" ~ "95% CI",
       .data$label == "Survival Probability, Survival Probability" ~ "25% and 75%-ile",
          .default = .data$label
        )
      )
  ) |>
  # update indentation to match spec
  modify_indent(columns = "label", rows = label == "95% CI", indent = 8L) |>
  modify_indent(columns = "label", rows = .data$label == "Range", indent = 4L) |>
  # remove default footnotes
  remove_footnote_header(columns = all_stat_cols())
```

Examples

```
# Example 1 ----------------------------------
tbl_survfit_quantiles(
  data = cards::ADTTE,
  by = "TRTA",
```

```
  estimate_fun = label_roche_number(digits = 1, na = "NE")
) |>
  add_overall(last = TRUE, col_label = "**All Participants**  \nN = {n}")

# Example 2: unstratified analysis -----------
tbl_survfit_quantiles(data = cards::ADTTE)
```

---

tbl_survfit_times *Survival Times*

Description

Create a gtsummary table with Kaplan-Meier estimated survival estimates and specified times.

Usage

```
tbl_survfit_times(
  data,
  times,
  y = "survival::Surv(time = AVAL, event = 1 - CNSR, type = 'right', origin = 0)",
  by = NULL,
  label = "Time {time}",
  statistic = c("{n.risk}", "{estimate}", "({conf.low}, {conf.high})"),
  estimate_fun = label_roche_number(digits = 1, scale = 100),
  method.args = list(conf.int = 0.95, conf.type = "plain")
)

## S3 method for class 'tbl_survfit_times'
add_difference_row(
  x,
  reference,
  statistic = c("{estimate}", "({conf.low}, {conf.high})", "{p.value}"),
  conf.level = 0.95,
  pvalue_fun = label_roche_pvalue(),
  estimate_fun = label_roche_number(digits = 2, scale = 100),
  ...
)

## S3 method for class 'tbl_survfit_times'
add_overall(
  x,
  last = FALSE,
  col_label = "All Participants  \nN = {style_roche_number(N)}",
  ...
)
```

Arguments

data	(data.frame) A data frame
times	(numeric) a vector of times for which to return survival probabilities.
y	(string or expression) A string or expression with the survival outcome, e.g. survival::Surv(time, status). The default value is survival::Surv(time = AVAL, event = 1 - CNSR, type = "right", origin = 0).
by	(tidy-select) A single column from data. Summary statistics will be stratified by this variable. Default is NULL, which returns results for the unstratified model.
label	(string) Label to appear in the header row. Default is "Time {time}", where the glue syntax injects the time estimate into the label.
statistic	(character) Character vector of the statistics to report. May use any of the following statistics: c(n.risk, estimate, std.error, conf.low, conf.high), Default is c("{n.risk}", "{estimate}", "({conf.low}, {conf.high})") Statistics available to include when using add_difference_row() are: "estimate", "std.error", "statistic", "conf.low", "conf.high", "p.value".
estimate_fun	(function) Function used to style/round the c(estimate, conf.low, conf.high) statistics.
method.args	(named list) Named list of arguments that will be passed to survival::survfit(). Note that this list may contain non-standard evaluation components, and must be handled similarly to tidyselect inputs by using rlang's embrace operator {{ . }} or !!enquo() when programming with this function.
x	(tbl_survfit_times) A stratified 'tbl_survfit_times' object
reference	(string) Value of the tbl_survfit_times(by) variable value that is the reference for each of the difference calculations. For factors, use the character level. The reference column will appear as the leftmost column in the table.
conf.level	(numeric) a scalar in the interval (0, 1) indicating the confidence level. Default is 0.95
pvalue_fun	(function) Function to round and format the p.value statistic. Default is label_roche_pvalue(). The function must have a numeric vector input, and return a string that is the rounded/formatted p-value (e.g. pvalue_fun = label_style_pvalue(digits = 3)).
...	These dots are for future extensions and must be empty.

`last` (scalar `logical`)
Logical indicator to display overall column last in table. Default is `FALSE`, which will display overall column first.

`col_label` (`string`)
String indicating the column label. Default is `"**Overall** \nN = {style_number(N)}"`

Details

When the `statistic` argument is modified, the statistic labels will likely also need to be updated. To change the label, call the `modify_table_body()` function to directly update the underlying `x$table_body` data frame.

Value

a gtsummary table

Methods (by generic)

- `add_difference_row(tbl_survfit_times)`: Adds survival differences between groups as additional rows to tables created by `tbl_survfit_times()`.

  Difference statistics are calculated using `cardx::ard_survival_survfit_diff()` for all `tbl_survfit_times(times)` variable values, using `survfit` formula:

  ```
  survival::survfit(y ~ by, data = data)
  ```

  where `y`, `by` and `data` are the inputs of the same names to the `tbl_survfit_times()` object `x`.

  Pairwise differences are calculated relative to the specified `by` variable's specified reference level.

Examples

```
# Example 1 ----------------------------------
tbl_survfit_times(
  data = cards::ADTTE,
  by = "TRTA",
  times = c(30, 60),
  label = "Day {time}"
) |>
  add_overall()
# Example 2 - Survival Differences -----------
tbl_survfit_times(
  data = cards::ADTTE,
  by = "TRTA",
  times = c(30, 60),
  label = "Day {time}"
) |>
  add_difference_row(reference = "Placebo")
```

`tbl_with_pools` *Create a gtsummary table with overlapping pooled columns*

Description

Generates a gtsummary table that includes both the original treatment arms and user-defined pooled treatment arms. It does this by creating individual tables for the original data and each pool, then merging them together seamlessly using `gtsummary::tbl_merge()`. This approach keeps the underlying dataset rows completely unique, preserving standard ADaM integrity while bypassing the need for complex pre-processing.

Usage

```
tbl_with_pools(
  data,
  pools,
  by = "TRT01A",
  denominator = NULL,
  keep_original = TRUE,
  .tbl_fun,
  ...
)
```

Arguments

`data`	(`data.frame`) The main analysis dataset (e.g., `adae`).
`pools`	(`list`) Named list of custom pools. Values can be character vectors of arm names, logical expressions wrapped in `rlang::expr()`, or the keyword `"all"` to include a total column.
`by`	(`character`) The treatment arm variable name.
`denominator`	(`data.frame` or `NULL`) The denominator dataset (e.g., `adsl`). Leave as NULL for functions that do not use a denominator (like `tbl_summary`). Provide the dataset for functions that require it (like `tbl_hierarchical_rate_and_count`).
`keep_original`	(`logical`) Keep the unpooled treatment arms. Default is `TRUE`.
`.tbl_fun`	(`function`) The gtsummary function to apply (e.g., `tbl_summary`).
`...`	Additional arguments passed directly to `.tbl_fun`.

Value

A merged `gtsummary` object of class `tbl_merge` and `tbl_with_pools`.

See Also

df_add_poolings() for a more dangerous pre-processing approach that modifies the underlying datasets to create pooled rows (not recommended).

Examples

```
# Create minimal dummy ADaM data
adsl <- data.frame(
  USUBJID = c("001", "002", "003", "004", "005"),
  TRT01A = c("Drug A", "Drug A", "Drug B", "Drug C", "Drug C"),
  AGE = c(45, 50, 60, 65, 55),
  FLAG = c("Y", "N", "Y", "N", "Y"),
  stringsAsFactors = FALSE
)

adae <- data.frame(
  USUBJID = c("001", "001", "002", "004", "005"),
  TRT01A = c("Drug A", "Drug A", "Drug A", "Drug C", "Drug C"),
  AEBODSYS = c("SOC1", "SOC1", "SOC2", "SOC1", "SOC2"),
  AEDECOD = c("PT1", "PT2", "PT3", "PT1", "PT4"),
  FLAG = c("Y", "Y", "N", "N", "Y"),
  stringsAsFactors = FALSE
)

# Define the requested pools
my_pools <- list(
  "Drugs A and B" = c("Drug A", "Drug B"),
  "All Patients"  = "all"
)

# Example A: Safe pooling with standard gtsummary (no denominator) ---------------
safe_pools <- list("Drugs A and B" = c("Drug A", "Drug B"))

tbl_safe <- tbl_with_pools(
  data = adsl,
  pools = safe_pools,
  by = "TRT01A",
  denominator = NULL,
  keep_original = FALSE,
  .tbl_fun = tbl_summary,
  include = AGE
)
tbl_safe

# Example B: Triggering the skipped pool warning ---------------------------------
# This throws a warning because 'Drug Z' has zero patients in the denominator
warning_pools <- list("Drug Z Pool" = c("Drug Z"))

tbl_warning <- tbl_with_pools(
  data = adae,
  pools = warning_pools,
  by = "TRT01A",
```

```
  keep_original = TRUE,
  .tbl_fun = tbl_summary,
  include = AEBODSYS
)
tbl_warning

# Example C: Complex pooling using logical expressions ---------------------------
complex_pools <- list(
  "Flagged Patients" = rlang::expr(FLAG == "Y"),
  "Drug A Flagged"   = rlang::expr(TRT01A == "Drug A" & FLAG == "Y")
)

tbl_complex <- tbl_with_pools(
  data = adsl,
  pools = complex_pools,
  by = "TRT01A",
  denominator = NULL,
  keep_original = FALSE,
  .tbl_fun = tbl_summary,
  include = AGE
)
tbl_complex


# Example D: Use yaml to define the pools config and run the function ------------

# Define the config as a standard R list
config_to_write <- list(
  tbl_with_pools_config = list(
    keep_original = FALSE,
    arm_var = "TRT01A",
    pools = list(
      "Drug A + B"   = c("Drug A", "Drug B"),
      "Drug C + B"   = c("Drug C", "Drug B"),
      "All Patients" = "all"
    )
  )
)

# Write it to a file (using a temp file for this example)
yaml_path <- tempfile(fileext = ".yaml")
yaml::write_yaml(config_to_write, yaml_path)

# Print out what the physical YAML file looks like
cat("--- Contents of the generated YAML file ---\n")
cat(readLines(yaml_path), sep = "\n")
cat("-------------------------------------------\n\n")

# Read the YAML file back into R
arg_specs <- yaml::read_yaml(yaml_path)

# Extract just the poolings config block
pool_args <- arg_specs$tbl_with_pools_config
```

```
# Run the function using tbl_hierarchical_rate_and_count
if (!is.null(pool_args)) {
  tbl_pooled_yaml <- tbl_with_pools(
    data = adae,
    pools = pool_args$pools,
    by = pool_args$arm_var,
    denominator = adsl,
    keep_original = pool_args$keep_original,
    .tbl_fun = tbl_hierarchical_rate_and_count,
    variables = c(AEBODSYS, AEDECOD)
  )
  tbl_pooled_yaml
}
```

`theme_gtsummary_roche` *Roche Theme*

Description

A gtsummary theme for Roche tables

- flextable- and gt-printed tables are styled with reduced padding and font size.
- Uses `label_roche_pvalue()` as the default formatting function for all p-values.
- Uses `label_roche_percent()` as the default formatting function for all percent values.
- Font size defaults are 8 points for all the table by the footers that are 7 points.
- Border defaults to `flextable::fp_border_default(width = 0.5)`.
- The `add_overall(col_label)` default value has been updated.
- The results from `gtsummary::tbl_hierarchical()` and `gtsummary::tbl_hierarchical_count()` are now post-processed with `gtsummary::remove_footnote_header()`, `crane::modify_zero_recode()`, and `crane::modify_header_rm_md()`.

Usage

```
theme_gtsummary_roche(
  font_size = NULL,
 print_engine = c("flextable", "gt", "kable", "kable_extra", "huxtable", "tibble"),
  set_theme = TRUE
)
```

Arguments

`font_size`	(scalar `numeric`) Numeric font size for compact theme. Default is 13 for gt tables, and 8 for all other output types

print_engine	String indicating the print method. Must be one of "gt", "kable", "kable_extra", "flextable", "tibble"
set_theme	(scalar logical) Logical indicating whether to set the theme. Default is TRUE. When FALSE the named list of theme elements is returned invisibly

Value

theme list

Examples

```
theme_gtsummary_roche()

tbl_roche_summary(
  trial,
  by = trt,
  include = c(age, grade),
  nonmissing = "always"
)

reset_gtsummary_theme()
```

Index