

Package: junco (via r-universe)

October 16, 2026

Title Create Common Tables and Listings Used in Clinical Trials

Version 0.1.7009

Date 2026-10-09

Description Structure and formatting requirements for clinical trial table and listing outputs vary between pharmaceutical companies. 'junco' provides additional tooling for use alongside the 'rtables', 'rlistings' and 'tern' packages when creating table and listing outputs. While motivated by the specifics of Johnson and Johnson Clinical and Statistical Programming's table and listing shells, 'junco' provides functionality that is general and reusable. Major features include a) alternative and extended statistical analyses beyond what 'tern' supports for use in standard safety and efficacy tables, b) a robust production-grade Rich Text Format (RTF) and DOCX exporter for tables, listings and graphs, c) structural support for spanning column headers and risk difference columns in tables, and d) robust font-aware automatic column width algorithms for both listings and tables.

License Apache License (≥ 2)

URL <https://github.com/johnsonandjohnson/junco>,
<https://johnsonandjohnson.github.io/junco/>

BugReports <https://github.com/johnsonandjohnson/junco/issues>

Depends R (≥ 4.4), formatters ($\geq 0.5.13$), rtables ($\geq 0.6.17$)

Imports tidytlg ($\geq 0.12.0$), tern ($\geq 0.9.13$), rlistings ($\geq 0.2.13$),
checkmate ($\geq 2.1.0$), rbmi ($\geq 1.6.0$), broom, methods,
generics, stats, survival, tibble, utils, emmeans, mrmr,
vcdExtra ($\geq 0.8.7$), rtables.officer ($\geq 0.1.2$), flextable ($\geq 0.9.11$), officer, xml2, ggplot2, assertthat

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

Suggests knitr, rmarkdown, forcats ($\geq 1.0.0$), testthat ($\geq 3.0.0$),
 mockery, mvtnorm, parallel, readxl, rlang, tidyr,
 pharmaverseadamjnj ($\geq 0.0.5$), multcomp, lifecycle, dplyr,
 ratesci ($\geq 1.1.0$), random.cdisc.data

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

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

Repository <https://johnsonandjohnson.r-universe.dev>

Date/Publication 2026-10-16 08:04:23 UTC

RemoteUrl <https://github.com/johnsonandjohnson/junco>

RemoteRef HEAD

RemoteSha f723458abfcad1c300e970a25640d614f22286c1

Contents

a_combo_prop_diff_pval_mf	5
a_cond_proportion_j	9
a_coxph_hr	13
a_freq_combos_j	14
a_freq_j	18
a_freq_resp_var_j	30
a_freq_subcol_j	32
a_kaplan_meier	36
a_maxlev	37
a_proportion_ci_factor	39
a_proportion_ci_logical	40
a_relative_risk	41
a_sum_ratio_j	44
a_summarize_ancova_j	46
a_summarize_aval_chg_diff_j	48
a_summarize_ex_j	53
a_summary_j	57
a_two_tier	59
ac_blank_line	63
analyze_values	64
bspt_pruner	64
build_formula	66
c_proportion_logical	67
categorize_pval	68
check_wrap_nobreak	69
cmhrms	70
cmp_cfun	72

cmp_post_fun	73
column_stats	74
cond_rm_facets	75
count and fraction related formatting functions	78
count_pruner	80
create_colspan_map	81
create_colspan_var	83
cur_col_split_path_utils	85
diff_means	87
do_exclude_split	90
event_free	91
export_as_docx_j	93
export_graph_as_docx	97
export_TLG_as_docx	98
factor_by_order	104
find_missing_chg_after_avisit	105
fit_ancova	106
fit_mmrn_j	107
fmt_spec_d	110
fmt_utils	114
get_mmrn_lsmeans	117
get_ref_info	118
get_titles_from_file	120
get_visit_levels	121
grouped_cols_w_diffs	121
h_odds_ratio	125
inches_to_spaces	127
insert_blank_line	127
jj_complex_scorefun	128
jjcs_num_formats	132
jjcsformat_fraction	132
jjcsformat_xx	134
keep_non_null_rows	136
leftside	137
listing_column_widths	138
lsmeans_wide_first_split_fun_fct	139
make_combo_splitfun	141
make_multicomp_splfun	142
make_rbmi_cluster	145
odds_ratio	145
par_lapply	149
pool_rubin_scalar	150
pool_z_stat	151
prepend_label_cell	151
prop_diff	152
prop_diff_test	154
prop_post_fun	156
prop_ratio_cmh	157

prop_table_afun	158
proportion_diff_mf	159
rbmi_analyse	163
rbmi_ancova	165
rbmi_ancova_single	166
rbmi_mmrn	168
rbmi_mmrn_single_info	170
rbmi_pool	170
real_add_overall_facet	171
remove_col_count	172
remove_rows	173
resp_multiple_imputation	174
resp01_a_comp_stat_factor	175
resp01_a_comp_stat_logical	176
resp01_acfun	178
resp01_counts_cfun	180
resp01_split_fun_fct	181
response_by_var	182
rightside	184
rm_levels	185
rm_other_facets_fact	185
s_ancova_j	186
s_proportion_factor	188
s_proportion_logical	189
s_summary_diff	190
safe_prune_table	191
set_titles	193
shift_tbl_col_struct	193
some_v_all_col_struct	194
stats_in_cols_setup	196
string_to_title	197
summarize_coxreg_multivar	198
summarize_mmrn	199
summarize_row_counts	202
summary_subset	203
tabulate_lsmeans	206
tabulate_rbmi	208
test_proportion_diff_mf	210
theme_docx_default_j	215
tt_to_flextable_j	216
tt_to_tbldf	219
tt_to_tlgtrf	220

a_combo_prop_diff_pval_mf

Combined Proportion, Risk Difference, and Risk Difference Test Analysis Function with Method Selection Based on the Mantel-Fleiss Criterion

Description

Performs a combined analysis of proportions, treatment-vs-control risk differences, and tests of risk differences.

The method used for the risk difference and risk difference test is selected based on the Mantel-Fleiss criterion. If the Mantel-Fleiss (MF) criterion is satisfied, the Cochran-Mantel-Haenszel (CMH) method with Sato variance is used for both the risk difference and the corresponding statistical test.

If the Mantel-Fleiss criterion is not satisfied, an exact method is used. Specifically, the exact unconditional method is used for estimation of the risk difference, and Fisher's exact test is used for the hypothesis test.

Usage

```
a_combo_prop_diff_pval_mf(
  df,
  .var,
  .df_row,
  .spl_context,
  val,
  colpaths,
  id = "USUBJID",
  trt_var,
  ctrl_group,
  variables = list(strata = NULL),
  mf_threshold = NULL,
  conf_level = 0.95,
  alternative = c("two.sided", "less", "greater"),
  exact_footnote = rtables::RefFootnote("Exact Inference", 1L, "+"),
  .stats = NULL,
  .formats = NULL,
  ...
)
```

Arguments

df	(data.frame) data set containing all analysis variables.
.var	(character(1)) Name of a response variable in df and .df_row. The variable must be a factor.

<code>.df_row</code>	(<code>data.frame</code>) Data set containing all analysis variables across all columns for the given row split. It is required for risk difference and risk difference test calculations to obtain the reference data.
<code>.spl_context</code>	(<code>data.frame</code>) gives information about ancestor split states that is passed by <code>rtables</code> .
<code>val</code>	(<code>character(1)</code>) The value used to identify the response of interest. An observation is considered a response of interest when <code>df[[.var]] == val</code> (and <code>.df_row[[.var]] == val</code> for the reference group). Since <code>df[[.var]]</code> and <code>.df_row[[.var]]</code> must be factors, <code>val</code> must be a character value matching one of their levels.
<code>colpaths</code>	(<code>list(3)</code>) A named list containing elements defining the column paths corresponding to the three types of analysis. Supported names are "prop", "diff", and "pval". Each element must contain the column path used to identify the corresponding column(s). Each column path must contain at least two elements and have an even length. For example, valid column paths include <code>c("ARM", "Placebo")</code> or <code>c("ARM", "*")</code> . See <code>rtables::col_paths()</code> for details on column path specification.
<code>id</code>	(<code>string</code>) subject variable name.
<code>trt_var</code>	(<code>character(1)</code>) The name of the treatment variable. This variable is used to identify the control group in <code>.df_row</code> . It is used only for risk difference and risk difference test calculations.
<code>ctrl_group</code>	(<code>character(1)</code>) The value of <code>trt_var</code> identifying the control treatment group. It is used only for risk difference and risk difference test calculations.
<code>variables</code>	(<code>list</code> or <code>NULL</code>) Optional stratification variables used by the Cochran-Mantel-Haenszel analysis. These are specified as <code>list(strata = <strata_vars>)</code> , where <code><strata_vars></code> is a character vector containing column names in <code>df</code> . All specified strata variables must be factors. Strata variables are used only for risk difference and risk difference test calculations.
<code>mf_threshold</code>	(<code>numeric(1)</code> or <code>NULL</code>) Threshold for the Mantel-Fleiss criterion. If <code>NULL</code> , the default from <code>tern::mantel_fleiss_crit()</code> is used.
<code>conf_level</code>	(<code>proportion</code>) confidence level of the interval.
<code>alternative</code>	(<code>string</code>) whether two.sided, or one-sided less or greater p-value should be displayed.
<code>exact_footnote</code>	(<code>character(1)</code> or <code>RefFootnote</code>) Footnote used to indicate that the unconditional exact method was actually used for the analysis.

<code>.stats</code>	(list) A named list of at most three elements specifying the statistic(s) to calculate for each type of analysis. Supported names are "prop", "diff", and "pval". The supported values for each analysis type are determined by the <code>.stats</code> argument of <code>a_freq_j()</code>, <code>a_proportion_diff_mf()</code>, and <code>a_test_proportion_diff_mf()</code>, respectively. If a statistic is not supplied for any of the three analysis types, the following defaults are used: • <code>prop = "count_unique_denom_fraction"</code> • <code>diff = "diff_est_ci"</code> • <code>pval = "pval"</code>
<code>.formats</code>	(list) A named list of at most three elements specifying the formats for the statistics specified in <code>.stats</code>. Supported names are "prop", "diff", and "pval". The format specified for each analysis type is passed to the corresponding analysis function.
<code>...</code>	Additional arguments reserved for compatibility with the <code>rtables</code> analysis function interface.

Details

The analysis function determines which type of result to calculate based on the current column path specified by `colpaths`.

- "prop": calculates and displays the proportion for the current analysis facet.
- "diff": calculates the treatment-vs-control proportion difference.
- "pval": calculates the treatment-vs-control p-value.

For proportion columns, `a_freq_j()` is used. For risk difference and p-value columns, the control group is identified from `.df_row` using `trt_var` and `ctrl_group`, and the corresponding `a_proportion_diff_mf()` or `a_test_proportion_diff_mf()` function is called.

The current column is identified using `in_column()` against each element of `colpaths` list. If the current column does not correspond to any of the configured `prop`, `diff`, or `pval` paths, an empty `rtables::rcell()`.

The function first determines the current row label from `.spl_context`. If the current split represents the root of the analysis, the label "Full analysis set" is used; otherwise, the value of the leaf split is used.

Value

Returns the corresponding `CellValue` or `RowsVerticalSection` object with formatted results.

Author(s)

WW

See Also

[a_freq_j\(\)](#), [a_proportion_diff_mf\(\)](#), [a_test_proportion_diff_mf\(\)](#),

Examples

```
library(dplyr)

adrs_f <- formatters::ex_adrs |>
  filter(
    ARM %in% c("A: Drug X", "B: Placebo"),
    PARAMCD == "BESRSPI"
  ) |>
  droplevels() |>
  mutate(
    rsp = factor(ifelse(AVALC == "CR", "Y", "N")),
    ARM_FOR_RISCOLS = ARM # Required for Risk Difference and p-value columns
  )

riskcols_combodf <- tribble(
  ~valname, ~label, ~levelcombo, ~exargs,
  "diff", "Risk Difference", "A: Drug X", list(),
  "pval", "p-value", "B: Placebo", list()
)

col_split_fun <- add_combo_levels(
  riskcols_combodf,
  keep_levels = riskcols_combodf$valname
)

# Additional arguments for a_combo_prop_diff_pval_mf().
extra_args <- list(
  colpaths = list(
    prop = c("ARM", "*"),
    diff = c("ARM_FOR_RISCOLS", "diff"),
    pval = c("ARM_FOR_RISCOLS", "pval")
  ),
  id = "SUBJID",
  trt_var = "ARM",
  ctrl_group = "B: Placebo",
  val = "Y",
  variables = list(strata = "BMRKR2"),
  .stats = list(diff = "diff")
)

lyt <- basic_table(top_level_section_div = " ") |>
  split_cols_by("ARM") |>
  split_cols_by(
    "ARM_FOR_RISCOLS",
    split_fun = col_split_fun,
    nested = FALSE
  ) |>
  analyze(
```

```

      "rsp",
      afun = a_combo_prop_diff_pval_mf,
      extra_args = extra_args
    ) |>
    split_rows_by(
      var = "SEX",
      split_label = "Gender",
      child_labels = "hidden",
      label_pos = "visible"
    ) |>
    analyze(
      vars = "rsp",
      afun = a_combo_prop_diff_pval_mf,
      extra_args = extra_args,
      show_labels = "hidden"
    )

tbl <- build_table(lyt, adrs_f)
tbl

```

a_cond_proportion_j	<i>Conditional proportion with adaptive CI (exact vs. Wald)</i>
---------------------	---

Description

[Experimental]

Usage

```

a_cond_proportion_j(
  df,
  .var,
  ...,
  .stats = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL,
  .df_row = NULL
)

```

```

s_cond_proportion_j(
  df,
  .var,
  conf_level = 0.95,
  long = FALSE,
  na.rm = FALSE,
  num_limit = 0,
  denom_limit = 10,

```

```

    denom = c("n", "N_row"),
    method_scope = c("cell", "row"),
    .df_row = NULL
  )

  d_cond_proportion_j(
    conf_level,
    long = FALSE,
    num_limit = NULL,
    denom_limit = NULL,
    method = NULL,
    method_denom = NULL,
    method_rsp = NULL
  )

```

Arguments

<code>df</code>	(logical or <code>data.frame</code>) if only a logical vector is used, it indicates whether each subject is a responder or not. TRUE represents a successful outcome. If a <code>data.frame</code> is provided, the logical vector of responses must be indicated as a variable name in <code>.var</code> .
<code>.var</code>	(string)
<code>...</code>	additional arguments for the lower level functions.
<code>.stats</code>	(character) statistics to select for the table.
<code>.formats</code>	(named character or list) formats for the statistics. See Details in <code>analyze_vars</code> for more information on the 'auto' setting.
<code>.labels</code>	(named character) labels for the statistics (without indent).
<code>.indent_mods</code>	(named integer) indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.
<code>.df_row</code>	(<code>data.frame</code>) data for the current row across all columns, automatically supplied by <code>rtables</code> .
<code>conf_level</code>	(numeric)
<code>long</code>	(flag) whether a long description is required.
<code>na.rm</code>	(flag) whether NA responses should be removed before analysis. If FALSE and NA values are present, an error is raised. Note that missing values are also removed from the row-wise counts which is relevant when <code>method_scope = "row"</code> or <code>denom = "N_row"</code> .
<code>num_limit</code>	(int) numerator limit to trigger the exact method.

denom_limit	(int) denominator limit to trigger the exact method.
denom	(character) denominator to use for percentage and CI computation: "n" (default, number of observed records), or "N_row" (number of observations in the row).
method_scope	(string) select the CI method using counts from the current cell ("cell") or all columns in the current row ("row"). See details.
method	(string or NULL) selected CI method to show in the label. NULL retains the combined method description.
method_denom	(int) denominator used for the selected CI method.
method_rsp	(int) numerator used for the selected CI method.

Details

The analysis function `a_cond_proportion_j()` is used to create a layout element that estimates a response proportion with confidence interval, automatically selecting between exact and Wald methods based on observed counts and user-defined limits.

The statistics function mirrors `tern::s_proportion()` usage and output but removes the method argument and decides internally between "clopper-pearson" and "wald".

The exact method is used when any of these are true: (a) the number of responders is `num_limit` (or less), (b) all subjects except `num_limit` (or less) have observed response, or (c) the observed group size is less than `denom_limit`.

Depending on the `method_scope` choice, this decision is either taken based on the response data for an individual cell (`method_scope = "cell"`), or based on the response data for the entire row (`method_scope = "row"`):

- With `method_scope = "cell"`, the counts used for the decision are:
 - numerator: the number of responders in the current cell;
 - denominator: the total number of observations in the cell (using `df`) if `denom = "n"`, or across the row (using `.df_row`) if `denom = "N_row"`.
- With `method_scope = "row"`, the counts used for the decision are:
 - numerator: the total number of responders across the row (using `.df_row`);
 - denominator: the total number of observations across the row (from `.df_row`).

CI computation follows `tern::s_proportion()` conventions: helper functions are called with `n = denom`, so when `denom = "N_row"`, this row-level denominator is used for the interval.

Value

- `a_cond_proportion_j()` returns the corresponding list with formatted `rtables::CellValue()`.
- `s_cond_proportion_j()` returns statistics `n_prop` (n responders and proportion) and `prop_ci` (proportion CI), formatted consistently with `tern::s_proportion()`.
- `d_cond_proportion_j()` returns a string describing the analysis label.

Functions

- `a_cond_proportion_j()`: Formatted analysis function which is used as `afun` for conditional proportion with adaptive CI selection (exact vs. Wald).
- `s_cond_proportion_j()`: Statistics function estimating a proportion along with its confidence interval, with adaptive method selection.
- `d_cond_proportion_j()`: Description helper function for the conditional proportion summary label.

Examples

```
nex <- 100
dta <- data.frame(
  "rsp" = sample(c(TRUE, FALSE), nex, TRUE),
  "grp" = sample(c("A", "B"), nex, TRUE),
  "f1" = sample(c("a1", "a2"), nex, TRUE),
  stringsAsFactors = TRUE
)

l <- basic_table() |>
  split_cols_by(var = "grp") |>
  analyze(
    vars = "rsp",
    afun = a_cond_proportion_j,
    extra_args = list(
      conf_level = 0.90,
      num_limit = 0,
      denom_limit = 10
    )
  )

build_table(l, df = dta)

# Logical vector input
rsp_v <- c(TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, FALSE)
s_cond_proportion_j(rsp_v)

# Data frame input
dta <- data.frame(rsp = c(TRUE, TRUE, FALSE, TRUE, FALSE, NA))
s_cond_proportion_j(dta, .var = "rsp", na.rm = TRUE)

# Using method_scope = "row" (requires .df_row in ...)
df_row <- data.frame(rsp = rep(rsp_v, 2))
s_cond_proportion_j(
  dta, .var = "rsp", method_scope = "row",
  .df_row = df_row, na.rm = TRUE
)

d_cond_proportion_j(conf_level = 0.90, long = FALSE, num_limit = 0, denom_limit = 10)

# With dedicated row-wise method:
d_cond_proportion_j(
```

```
conf_level = 0.90, long = TRUE, num_limit = 0, denom_limit = 10,
method = "wald", method_denom = 10, method_rsp = 8)
```

a_coxph_hr

a_coxph_hr

Description

`a_coxph_hr` **[Deprecated]** Use `tern::a_coxph_pairwise()` instead, and use `tern::coxph_pairwise()` as the layout-level wrapper.

Usage

```
a_coxph_hr(
  df,
  .var,
  ref_path,
  .spl_context,
  ...,
  .stats = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```

Arguments

<code>df</code>	(data.frame) input data frame.
<code>.var</code>	(string) name of the response variable.
<code>ref_path</code>	(character) path to the reference group.
<code>.spl_context</code>	(environment) split context environment.
<code>...</code>	Additional arguments passed to the statistics function.
<code>.stats</code>	(character) statistics to calculate.
<code>.formats</code>	(list) formats for the statistics.
<code>.labels</code>	(list) labels for the statistics.
<code>.indent_mods</code>	(list) indentation modifications for the statistics.

a_freq_combos_j	<i>Analysis function count and percentage in column design controlled by combosdf</i>
-----------------	---

Description

Analysis function count and percentage in column design controlled by combosdf

Usage

```
a_freq_combos_j(
  df,
  labelstr = NULL,
  .var = NA,
  val = NULL,
  combosdf = NULL,
  do_not_filter = NULL,
  filter_var = NULL,
  flag_var = NULL,
  .df_row,
  .spl_context,
  .N_col,
  id = "USUBJID",
  denom = c("N_col", "n_df", "n_altdf", "n_rowdf", "n_parentdf"),
  label = NULL,
  label_fstr = NULL,
  label_map = NULL,
  .alt_df_full = NULL,
  denom_by = NULL,
  .stats = "count_unique_denom_fraction",
  .formats = NULL,
  .labels_n = NULL,
  .indent_mods = NULL,
  na_str = rep("NA", 3)
)
```

Arguments

df	(data.frame) data set containing all analysis variables.
labelstr	(character) label of the level of the parent split currently being summarized (must be present as second argument in Content Row Functions). See rtables::summarize_row_groups() for more information.
.var	(string) single variable name that is passed by rtables when requested by a statistics function.

val	(character or NULL) When NULL, all levels of the incoming variable (variable used in the analyze call) will be considered. When a single string, only that current level/value of the incoming variable will be considered. When multiple levels, only those levels/values of the incoming variable will be considered. When no values are observed (eg zero row input df), a row with row-label No data to report will be included in the table.
combosdf	The df which provides the mapping of column facets to produce cumulative counts for .N_col. In the cell facet, these cumulative records must then be removed from the numerator, which can be done via the filter_var parameter to avoid unwanted counting of events.
do_not_filter	A vector of facets (i.e., column headers), identifying headers for which no filtering of records should occur. That is, the numerator should contain cumulative counts. Generally, this will be used for a "Total" column, or something similar.
filter_var	The variable which identifies the records to count in the numerator for any given column. Generally, this will contain text matching the column header for the column associated with a given record.
flag_var	Variable which identifies the occurrence (or first occurrence) of an event. The flag variable is expected to have a value of "Y" identifying that the event should be counted, or NA otherwise.
.df_row	(data.frame) data frame across all of the columns for the given row split.
.spl_context	(data.frame) gives information about ancestor split states that is passed by rtables.
.N_col	(integer) column-wise N (column count) for the full column being analyzed that is typically passed by rtables.
id	(string) subject variable name.
denom	(string) One of

- **N_col** Column count,
- **n_df** Number of patients (based upon the main input dataframe df),
- **n_altdf** Number of patients from the secondary dataframe (.alt_df_full), Note that argument denom_by will perform a row-split on the .alt_df_full dataframe.
It is a requirement that variables specified in denom_by are part of the row split specifications.

- **n_rowdf** Number of patients from the current row-level dataframe (`.row_df` from the rtables splitting machinery).
- **n_parentdf** Number of patients from a higher row-level split than the current split.
This higher row-level split is specified in the argument `denom_by`.

label (string)
When `val` has length 1, the row label to be shown on the output can be specified using this argument.
When `val` is a character vector, the `label_map` argument can be specified to control the row-labels.

label_fstr (string)
a `sprintf` style format string. It can contain up to one `"%s"`, which takes the current split value and generates the row/column label.
It will be combined with the `labelstr` argument, when utilizing this function as a `cfun` in a `summarize_row_groups` call.
It is recommended not to utilize this argument for other purposes. The `label` argument could be used instead (if `val` is a single string)

label_map (data.frame)
A mapping data frame used to translate levels of the analysis variable(s) into the row labels displayed in the table, optionally conditioned on the nearest row-split.

Four types of mappings are supported:

1. Single-variable mapping. Column names: `level`, `label`.
2. Multi-variable mapping. Column names: `var`, `level`, `label`.
3. Conditional-on-row-split mapping. Column names: `<split_var>`, `level`, `label`.
4. Conditional-on-row-split multi-variable mapping. Column names: `<split_var>`, `var`, `level`, `label`.

Here, `<split_var>` is a placeholder for the name of the variable used for the nearest row-split (e.g. `SEX`, `PARAMCD`).

Recommendation: ensure input variable(s) are of type factor, to avoid the error "got a label map that doesn't provide labels for all values".

See examples for more details.

.alt_df_full (dataframe)
Denominator dataset for fraction and relative risk calculations.
this argument gets populated by the rtables split machinery (see [rtables::additional_fun_params](#)).

denom_by (character)
Variables from row-split to be used in the denominator derivation.
This controls both `denom = "n_parentdf"` and `denom = "n_altdf"`.
When `denom = "n_altdf"`, the denominator is derived from `.alt_df_full` in combination with `denom_by` argument

.stats	(character) statistics to select for the table.
.formats	(named 'character' or 'list') formats for the statistics.
.labels_n	(named character) String to control row labels for the 'n'-statistics. Only useful when more than one 'n'-statistic is requested (rare situations only).
.indent_mods	(named integer) indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.
na_str	(string) string used to replace all NA or empty values in the output.

Value

list of requested statistics with formatted `rtables::CellValue()`.

See Also

[a_freq_j\(\)](#) for extensive `label_map` usage examples.

Examples

```
library(dplyr)
ADSL <- ex_adsl |> select(USUBJID, ARM, EOSSTT, EOSDT, EOSDY, TRTSDTM)

cutoffd <- as.Date("2023-09-24")

ADSL <- ADSL |>
  mutate(
    TRTDURY = case_when(
      !is.na(EOSDY) ~ EOSDY,
      TRUE ~ as.integer(cutoffd - as.Date(TRTSDTM) + 1)
    )
  ) |>
  mutate(ACAT1 = case_when(
    TRTDURY < 183 ~ "0-6 Months",
    TRTDURY < 366 ~ "6-12 Months",
    TRUE ~ "+12 Months"
  )) |>
  mutate(ACAT1 = factor(ACAT1, levels = c("0-6 Months", "6-12 Months", "+12 Months")))

ADAE <- ex_adae |> select(USUBJID, ARM, AEBODSYS, AEDECOD, ASTDY)

ADAE <- ADAE |>
  mutate(TRTEMFL = "Y") |>
  mutate(ACAT1 = case_when(
    ASTDY < 183 ~ "0-6 Months",
```

```

    ASTDY < 366 ~ "6-12 Months",
    TRUE ~ "+12 Months"
  )) |>
  mutate(ACAT1 = factor(ACAT1, levels = c("0-6 Months", "6-12 Months", "+12 Months")))

  combodf <- tribble(
    ~valname, ~label, ~levelcombo, ~exargs,
    "Tot", "Total", c("0-6 Months", "6-12 Months", "+12 Months"), list(),
    "A_0-6 Months", "0-6 Months", c("0-6 Months", "6-12 Months", "+12 Months"), list(),
    "B_6-12 Months", "6-12 Months", c("6-12 Months", "+12 Months"), list(),
    "C_+12 Months", "+12 Months", c("+12 Months"), list()
  )

  lyt <- basic_table(show_colcounts = TRUE) |>
    split_cols_by("ARM") |>
    split_cols_by("ACAT1",
      split_fun = rtables::add_combo_levels(
        combosdf = combodf,
        trim = FALSE, keep_levels = combodf$valname
      )
    ) |>
    analyze("RTEMFL",
      show_labels = "hidden",
      afun = a_freq_combos_j,
      extra_args = list(
        val = "Y",
        label = "Subjects with >= 1 AE",
        combosdf = combodf,
        filter_var = "ACAT1",
        do_not_filter = "Tot"
      )
    )

  result <- build_table(lyt, df = ADAE, alt_counts_df = ADSL)

  result

```

a_freq_j

*Analysis/statistical function for count and percentage in core columns
and (optional) relative risk columns*

Description

Analysis/statistical function for count and percentage in core columns and (optional) relative risk columns

Usage

```

s_freq_j(
  df,
  .var,
  .df_row,
  val = NULL,
  drop_levels = FALSE,
  excl_levels = NULL,
  alt_df,
  parent_df,
  id = "USUBJID",
  denom = c("n_df", "n_altdf", "N_col", "n_rowdf", "n_parentdf"),
  .N_col,
  countsource = c("df", "altdf", "altdf_subset")
)

a_freq_j(
  df,
  labelstr = NULL,
  .var = NA,
  val = NULL,
  drop_levels = FALSE,
  excl_levels = NULL,
  new_levels = NULL,
  new_levels_after = FALSE,
  addstr2levs = NULL,
  .df_row,
  .spl_context,
  .N_col,
  id = "USUBJID",
  denom = c("N_col", "n_df", "n_altdf", "N_colgroup", "n_rowdf", "n_parentdf"),
  riskdiff = TRUE,
  ref_path = NULL,
  variables = list(strata = NULL),
  conf_level = 0.95,
  method = c("wald", "waldcc", "cmh", "ha", "newcombe", "newcombecc", "strat_newcombe",
    "strat_newcombecc", "cmh_sato", "cmh_mn", "uncond_exact_diff"),
  weights_method = "cmh",
  label = NULL,
  label_fstr = NULL,
  label_map = NULL,
  .alt_df_full = NULL,
  denom_by = NULL,
  .stats = "count_unique_denom_fraction",
  .formats = NULL,
  .indent_mods = NULL,
  na_str = rep("NA", 3),
  .labels_n = NULL,

```

```

    extrablankline = FALSE,
    extrablanklineafter = NULL,
    restr_columns = NULL,
    colgroup = NULL,
    countsource = c("df", "altdf", "altdf_subset")
  )

  a_freq_j_with_exclude(
    df,
    labelstr = NULL,
    exclude_levels,
    .var = NA,
    .spl_context,
    .df_row,
    .N_col,
    .alt_df_full = NULL,
    .stats = "count_unique_denom_fraction",
    .formats = NULL,
    .indent_mods = NULL,
    .labels_n = NULL,
    ...
  )

```

Arguments

df	(data.frame) data set containing all analysis variables.
.var	(character(1)) Name of a categorical analysis variable in df. The variable must be a factor.
.df_row	(data.frame) data frame across all of the columns for the given row split.
val	(character or NULL) When NULL, all levels of the incoming variable (variable used in the analyze call) will be considered. When a single string, only that current level/value of the incoming variable will be considered. When multiple levels, only those levels/values of the incoming variable will be considered. When no values are observed (eg zero row input df), a row with row-label No data to report will be included in the table.
drop_levels	(logical) If TRUE non-observed levels (based upon .df_row) will not be included. Cannot be used together with val.
excl_levels	(character or NULL) When NULL, no levels of the incoming variable (variable used in the analyze call) will be excluded.

	When multiple levels, those levels/values of the incoming variable will be excluded. Cannot be used together with val.
alt_df	(dataframe) Will be derived based upon alt_df_full and denom_by within a_freq_j.
parent_df	(dataframe) Will be derived within a_freq_j based upon the input dataframe that goes into build_table (df) and denom_by. It is a data frame in the higher row-space than the current input df (which underwent row-splitting by the rtables splitting machinery).
id	(string) subject variable name.
denom	(string) See Details.
.N_col	(integer) column-wise N (column count) for the full column being analyzed that is typically passed by rtables.
countsource	Either df, altdf, or altdf_subset. When altdf the counts will be based upon the alternative dataframe alt_df. When altdf_subset the counts will be based upon alt_df but first restricted to the levels/values of the current row split for .var (or to val when provided). This is useful for subgroup processing, to present counts of subjects in a subgroup from the alternative dataframe.
labelstr	An argument to ensure this function can be used as a cfun in a summarize_row_groups call. It is recommended not to utilize this argument for other purposes. The label argument could be used instead (if val is a single string) An another approach could be to utilize the label_map argument to control the row labels of the incoming analysis variable.
new_levels	(list(2) or NULL) List of length 2. First element : names of the new levels Second element: list with values of the new levels.
new_levels_after	(logical) If TRUE new levels will be added after last level.
addstr2levs	string, if not NULL will be appended to the rowlabel for that level, eg to add ",n (percent)" at the end of the rowlabels
.spl_context	(data.frame) gives information about ancestor split states that is passed by rtables.
riskdiff	(logical) When TRUE, risk difference calculations will be performed and presented (if required risk difference column splits are included). When FALSE, risk difference columns will remain blank (if required risk difference column splits are included).

ref_path	(string) Column path specifications for the control group for the relative risk derivation.
variables	Will be passed onto the risk difference function (internal function <code>s_risk_diff_val_j</code>), which is based upon <code>tern::s_proportion_diff()</code> . See <code>?tern::s_proportion_diff</code> for details.
conf_level	(proportion) confidence level of the interval.
method	Will be passed onto the risk difference function (internal function <code>s_risk_diff_val_j</code>).
weights_method	Will be passed onto the risk difference function (internal function <code>s_risk_diff_val_j</code>).

label	(string) When <code>val</code> has length 1, the row label to be shown on the output can be specified using this argument. When <code>val</code> is a character vector, the <code>label_map</code> argument can be specified to control the row-labels.
-------	---

label_fstr	(string) a <code>sprintf</code> style format string. It can contain up to one <code>"%s"</code> , which takes the current split value and generates the row/column label. It will be combined with the <code>labelstr</code> argument, when utilizing this function as a <code>cfun</code> in a <code>summarize_row_groups</code> call. It is recommended not to utilize this argument for other purposes. The <code>label</code> argument could be used instead (if <code>val</code> is a single string)
------------	--

label_map	(data.frame) A mapping data frame used to translate levels of the analysis variable(s) into the row labels displayed in the table, optionally conditioned on the nearest row-split.
-----------	--

Four types of mappings are supported:

1. Single-variable mapping. Column names: `level`, `label`.
2. Multi-variable mapping. Column names: `var`, `level`, `label`.
3. Conditional-on-row-split mapping. Column names: `<split_var>`, `level`, `label`.
4. Conditional-on-row-split multi-variable mapping. Column names: `<split_var>`, `var`, `level`, `label`.

Here, `<split_var>` is a placeholder for the name of the variable used for the nearest row-split (e.g. `SEX`, `PARAMCD`).

Recommendation: ensure input variable(s) are of type factor, to avoid the error "got a label map that doesn't provide labels for all values".

See examples for more details.

.alt_df_full	(dataframe) Denominator dataset for fraction and relative risk calculations. this argument gets populated by the <code>rtables</code> split machinery (see <code>rtables::additional_fun_params</code>).
--------------	---

denom_by	(character) Variables from row-split to be used in the denominator derivation. This controls both denom = "n_parentdf" and denom = "n_altdf". When denom = "n_altdf", the denominator is derived from .alt_df_full in combination with denom_by argument
.stats	(character) Statistics to include in the table. May contain one or more of: "count", "count_unique", "count_unique_fraction", "count_unique_denom_fraction", "unique_fraction", "n_df", "n_altdf", "n_rowdf", "n_parentdf", "denom". See Value for the full list of available statistics.
.formats	(named 'character' or 'list') formats for the statistics.
.indent_mods	(named integer) indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.
na_str	(string) string used to replace all NA or empty values in the output.
.labels_n	(named character) String to control row labels for the 'n'-statistics. Only useful when more than one 'n'-statistic is requested (rare situations only).
extrablankline	(logical) When TRUE, an extra blank line will be added after the last value. Avoid using this in template scripts, use section_div = " " instead (once PR for rtables is available)
extrablanklineafter	(string) When the row-label matches the string, an extra blank line will be added after that value.
restr_columns	character If not NULL, columns not defined in restr_columns will be blanked out.
colgroup	The name of the column group variable that is used as source for denominator calculation. Required to be specified when denom = "N_colgroup".
exclude_levels	(list) A named list where names correspond to split variables and values are vectors of levels to exclude.
...	additional arguments for the lower level functions.

Details

denom controls the denominator used to calculate proportions/percents. It must be one of

- **N_col** Column count,

- **n_df** Number of patients (based upon the main input dataframe df),
- **n_altdf** Number of patients from the secondary dataframe (.alt_df_full),
Note that argument denom_by will perform a row-split on the .alt_df_full dataframe.
It is a requirement that variables specified in denom_by are part of the row split specifications.
- **N_colgroup** Number of patients from the column group variable (note that this is based upon the input .alt_df_full dataframe).
Note that the argument colgroup (column variable) needs to be provided, as it cannot be retrieved directly from the column layout definition.
- **n_rowdf** Number of patients from the current row-level dataframe (.row_df from the rtables splitting machinery).
- **n_parentdf** Number of patients from a higher row-level split than the current split.
This higher row-level split is specified in the argument denom_by.

Value

- **s_freq_j**: returns a list of following statistics
 - n_df
 - n_rowdf
 - n_parentdf
 - n_altdf
 - denom
 - count
 - count_unique
 - count_unique_fraction
 - count_unique_denom_fraction
 - unique_fraction
- **a_freq_j**: returns a list of requested statistics with formatted rtables::CellValue().
Within the relative risk difference columns, the following stats are blanked out:
 - any of the n-statistics (n_df, n_altdf, n_parentdf, n_rowdf, denom)
 - count
 - count_unique

For the others (count_unique_fraction, count_unique_denom_fraction, unique_fraction), the statistic is replaced by the relative risk difference + confidence interval.

Functions

- **a_freq_j_with_exclude()**: Wrapper for the afun which can exclude row split levels from producing the analysis. These have to be specified in the exclude_levels argument, see ?do_exclude_split for details.

Examples

```

library(dplyr)

adsl <- ex_adsl |> select("USUBJID", "SEX", "ARM")
adae <- ex_adae |> select("USUBJID", "AEBODSYS", "AEDECOD")
adae[["TRTEMFL"]] <- "Y"

trtvar <- "ARM"
ctrl_grp <- "B: Placebo"
adsl$colspan_trt <- factor(ifelse(adsl[[trtvar]] == ctrl_grp, " ", "Active Study Agent"),
  levels = c("Active Study Agent", " ")
)

adsl$rrisk_header <- "Risk Difference (%) (95% CI)"
adsl$rrisk_label <- paste(adsl[[trtvar]], paste("vs", ctrl_grp))

adae <- adae |> left_join(adsl)

colspan_trt_map <- create_colspan_map(adsl,
  non_active_grp = "B: Placebo",
  non_active_grp_span_lbl = " ",
  active_grp_span_lbl = "Active Study Agent",
  colspan_var = "colspan_trt",
  trt_var = trtvar
)

ref_path <- c("colspan_trt", " ", trtvar, ctrl_grp)

lyt <- basic_table(show_colcounts = TRUE) |>
  split_cols_by("colspan_trt", split_fun = trim_levels_to_map(map = colspan_trt_map)) |>
  split_cols_by(trtvar) |>
  split_cols_by("rrisk_header", nested = FALSE) |>
  split_cols_by(trtvar, labels_var = "rrisk_label", split_fun = remove_split_levels(ctrl_grp))

lyt1 <- lyt |>
  analyze("TRTEMFL",
    show_labels = "hidden",
    afun = a_freq_j,
    extra_args = list(
      method = "wald",
      .stats = c("count_unique_denom_fraction"),
      ref_path = ref_path
    )
  )

result1 <- build_table(lyt1, adae, alt_counts_df = adsl)

result1

x_drug_x <- list(length(unique(subset(adae, adae[[trtvar]] == "A: Drug X")[[USUBJID]])))
N_x_drug_x <- length(unique(subset(adsl, adsl[[trtvar]] == "A: Drug X")[[USUBJID]]))
y_placebo <- list(length(unique(subset(adae, adae[[trtvar]] == ctrl_grp)[[USUBJID]])))

```

```

N_y_placebo <- length(unique(subset(adsl, adsl[[trtvar]] == ctrl_grp)[["USUBJID"]]))

tern::stat_propdiff_ci(
  x = x_drug_x,
  N_x = N_x_drug_x,
  y = y_placebo,
  N_y = N_y_placebo
)

x_combo <- list(length(unique(subset(adae, adae[[trtvar]] == "C: Combination")[["USUBJID"]]))))
N_x_combo <- length(unique(subset(adsl, adsl[[trtvar]] == "C: Combination")[["USUBJID"]]))

tern::stat_propdiff_ci(
  x = x_combo,
  N_x = N_x_combo,
  y = y_placebo,
  N_y = N_y_placebo
)

extra_args_rr <- list(
  denom = "n_altdf",
  denom_by = "SEX",
  riskdiff = FALSE,
  .stats = c("count_unique")
)

extra_args_rr2 <- list(
  denom = "n_altdf",
  denom_by = "SEX",
  riskdiff = TRUE,
  ref_path = ref_path,
  method = "wald",
  .stats = c("count_unique_denom_fraction"),
  na_str = rep("NA", 3)
)

lyt2 <- basic_table(
  top_level_section_div = " ",
  colcount_format = "N=xx"
) |>
  split_cols_by("colspan_trt", split_fun = trim_levels_to_map(map = colspan_trt_map)) |>
  split_cols_by(trtvar, show_colcounts = TRUE) |>
  split_cols_by("rrisk_header", nested = FALSE) |>
  split_cols_by(trtvar,
    labels_var = "rrisk_label", split_fun = remove_split_levels("B: Placebo"),
    show_colcounts = FALSE
  ) |>
  split_rows_by("SEX", split_fun = drop_split_levels) |>
  summarize_row_groups("SEX",
    cfun = a_freq_j,
    extra_args = append(extra_args_rr, list(label_fstr = "Gender: %s"))
  ) |>

```

```

split_rows_by("TRTEMFL",
  split_fun = keep_split_levels("Y"),
  indent_mod = -1L,
  section_div = c(" ")
) |>
summarize_row_groups("TRTEMFL",
  cfun = a_freq_j,
  extra_args = append(extra_args_rr2, list(
    label =
      "Subjects with >=1 AE", extrablankline = TRUE
  ))
) |>
split_rows_by("AEBODSYS",
  split_label = "System Organ Class",
  split_fun = trim_levels_in_group("AEDECOD"),
  label_pos = "topleft",
  section_div = c(" "),
  nested = TRUE
) |>
summarize_row_groups("AEBODSYS",
  cfun = a_freq_j,
  extra_args = extra_args_rr2
) |>
analyze("AEDECOD",
  afun = a_freq_j,
  extra_args = extra_args_rr2
)

result2 <- build_table(lyt2, adae, alt_counts_df = adsl)

# -----
# Examples using the label_map argument
# -----

adsl <- ex_adsl |> select("USUBJID", "SEX", "ARM")
adae <- ex_adae |> select("USUBJID", "AEBODSYS", "AEDECOD")
adae[["TRTEMFL"]] <- "Y"

trtvar <- "ARM"
adae <- adae |> left_join(adsl)

# -----
# 1. Simple mapping.
# Remaps factor levels (value) to custom new labels (label).
# -----

label_map_simple <- data.frame(
  value = c("M", "F", "UNDIFFERENTIATED", "U"),
  label = c("Male", "Female", "Undifferentiated", "Unknown")
)

lyt_lmap1 <- basic_table(show_colcounts = TRUE) |>
  split_cols_by(trtvar) |>

```

```

analyze("SEX",
  afun = a_freq_j,
  extra_args = list(
    .stats = "count_unique_fraction",
    label_map = label_map_simple
  )
)

result_lmap1 <- build_table(lyt_lmap1, adsl, alt_counts_df = adsl)
result_lmap1

# -----
# 2. Multi-variable mapping.
# Useful in situations where multiple Y/N variables can be analyzed in one
# [rtabls::analyze()] call.
# -----

multi_vars <- c("DTH30FL", "DTHA30FL", "DTHB30FL")
adslx <- pharmaverseadamjnj::adsl
# ensure the variables are factor with levels Y/N - and variable label is kept
adslx <- adslx |>
  mutate(across(all_of(multi_vars), \(x) structure(
    factor(x, levels = c("Y", "N")),
    label = attr(x, "label")
  )))

map_multi <- data.frame(
  var = multi_vars,
  value = "Y",
  label = c(
    "Death <=30D of Last Trt Flag",
    "Death >30D from Last Trt Flag",
    "Death <=30D of First Trt Flag"
  )
)

lyt_lmap1 <- basic_table(show_colcounts = TRUE) |>
  split_cols_by("ARM") |>
  analyze(multi_vars,
    afun = a_freq_j,
    extra_args = list(
      .stats = "count_unique_denom_fraction",
      label_map = map_multi,
      val = "Y"
    ),
    show_labels = "hidden"
  )

result_lmap2 <- build_table(lyt_lmap1, adslx)
result_lmap2

# -----
# 3. Conditional-on-row-split mapping.

```

```

# -----

map_rowsplit <- data.frame(
  SEX = c("M", "F"),
  value = c("Y", "Y"),
  label = c("Male subjects with >=1 TE AE", "Female subjects with >=1 TE AE")
)

adsl <- ex_adsl |> select("USUBJID", "ARM", "SEX")
adae <- ex_adae |> select("USUBJID", "AEBODSYS", "AEDECOD")
adae[["TRTEMFL"]] <- "Y"
adae <- adae |> left_join(adsl)

lyt_lmap3 <- basic_table(show_colcounts = TRUE) |>
  split_cols_by("ARM") |>
  split_rows_by("SEX", split_fun = keep_split_levels(c("F", "M")), child_labels = "hidden") |>
  analyze("TRTEMFL",
    afun = a_freq_j,
    extra_args = list(
      .stats = "count_unique_denom_fraction",
      label_map = map_rowsplit,
      val = "Y"
    ),
    show_labels = "hidden"
  )

result_lmap3 <- build_table(lyt_lmap3, adae, alt_counts_df = adsl)
result_lmap3

# -----
# 4. Conditional-on-row-split multi-variable mapping.
# -----

adsl_jnj <- pharmaverseadamjnj::adsl
advs_jnj <- pharmaverseadamjnj::advs

multi_vars <- c("CRIT1FL", "CRIT2FL", "CRIT3FL")

map_multi_rowsplit <- data.frame(
  PARAMCD = c(rep("DIABP", 3), rep("SYSBP", 3)),
  var = rep(multi_vars, 2),
  value = rep("Y", 6),
  label = c(
    "<50 mmHg and with >20 mmHg decrease from baseline",
    ">105 mmHg and with >30 mmHg increase from baseline",
    "Diastolic blood pressure<60",
    "<90 mmHg and with >30 mmHg decrease from baseline",
    ">180 mmHg and with >40 mmHg increase from baseline",
    "Systolic blood pressure<90"
  )
)

lyt_lmap4 <- basic_table(show_colcounts = TRUE) |>
  split_cols_by("TRT01A") |>

```

```

split_rows_by("PARAMCD", split_fun = keep_split_levels(c("DIABP", "SYSBP"))) |>
analyze(multi_vars,
  afun = a_freq_j,
  extra_args = list(
    .stats = "count_unique_denom_fraction",
    label_map = map_multi_rowsplit,
    val = "Y"
  ),
  show_labels = "hidden"
)

result_lmap4 <- build_table(lyt_lmap4, advs_jnj, alt_counts_df = adsl_jnj)
result_lmap4

```

a_freq_resp_var_j	<i>Analysis Function for Response Variables</i>
-------------------	---

Description

This function calculates counts and percentages for response variables (Y/N values), with optional risk difference calculations.

Usage

```

a_freq_resp_var_j(
  df,
  .var,
  .df_row,
  .N_col,
  .spl_context,
  resp_var = NULL,
  id = "USUBJID",
  drop_levels = FALSE,
  riskdiff = TRUE,
  ref_path = NULL,
  variables = list(strata = NULL),
  conf_level = formals(s_proportion_diff)$conf_level,
  method = c("wald", "waldcc", "cmh", "ha", "newcombe", "newcombecc", "strat_newcombe",
    "strat_newcombecc", "cmh_sato", "cmh_mn", "uncond_exact_diff"),
  weights_method = formals(s_proportion_diff)$weights_method,
  .formats = NULL,
  na_str = rep("NA", 3),
  legacy = FALSE
)

```

Arguments

df	(data.frame) data set containing all analysis variables.
.var	(string) variable name that is passed by rtables.
.df_row	(data.frame) data frame across all of the columns for the given row split.
.N_col	(integer) column-wise N (column count) for the full column being analyzed.
.spl_context	(data.frame) gives information about ancestor split states.
resp_var	(string) response variable name containing Y/N values.
id	(string) subject variable name.
drop_levels	(logical) if TRUE, non-observed levels will not be included.
riskdiff	(logical) if TRUE, risk difference calculations will be performed.
ref_path	(string) column path specifications for the control group.
variables	(list) variables to include in the analysis.
conf_level	(proportion) confidence level of the interval.
method	(character) method for calculating confidence intervals.
weights_method	(character) method for calculating weights.
.formats	(character or NULL) formats to apply to the statistics. If NULL, default formats will be used.
na_str	(character) string to use for NA values. Defaults to rep("NA", 3).
legacy	(logical(1)) To allow for backward compatibility default of used format.

Value

Formatted analysis function which is used as `afun` in `analyze_vars()` and as `cfun` in `summarize_row_groups()`.

Examples

```
library(dplyr)
ADSL <- ex_adsl |> select(USUBJID, ARM, SEX)

ADAE <- ex_adae |> select(USUBJID, ARM, SEX, AEBODSYS, AEDECOD)

ADAE <- ADAE |>
  mutate(TRTEMFL = "Y")

lyt <- basic_table(show_colcounts = TRUE) |>
  split_cols_by("ARM") |>
  analyze("SEX",
    show_labels = "visible",
    afun = a_freq_resp_var_j,
    extra_args = list(resp_var = "TRTEMFL", riskdiff = FALSE)
  )

result <- build_table(lyt, df = ADAE, alt_counts_df = ADSL)

result
```

a_freq_subcol_j	<i>Analysis function count and percentage with extra column-subsetting in selected columns (controlled by subcol_* arguments)</i>
-----------------	---

Description

Analysis function count and percentage with extra column-subsetting in selected columns (controlled by subcol_* arguments)

Usage

```
a_freq_subcol_j(
  df,
  labelstr = NULL,
  .var = NA,
  val = NULL,
  subcol_split = NULL,
  subcol_var = NULL,
  subcol_val = NULL,
  .df_row,
  .spl_context,
  .N_col,
  id = "USUBJID",
  denom = c("N_col", "n_df", "n_altdf", "n_rowdf", "n_parentdf"),
  label = NULL,
  label_fstr = NULL,
  label_map = NULL,
```

```

    .alt_df_full = NULL,
    denom_by = NULL,
    .stats = c("count_unique_denom_fraction"),
    .formats = NULL,
    .labels_n = NULL,
    .indent_mods = NULL,
    na_str = rep("NA", 3)
  )

```

Arguments

df	(data.frame) data set containing all analysis variables.
labelstr	(character) label of the level of the parent split currently being summarized (must be present as second argument in Content Row Functions). See rtables::summarize_row_groups() for more information.
.var	(string) single variable name that is passed by rtables when requested by a statistics function.
val	(character or NULL) When NULL, all levels of the incoming variable (variable used in the analyze call) will be considered. When a single string, only that current level/value of the incoming variable will be considered. When multiple levels, only those levels/values of the incoming variable will be considered. When no values are observed (eg zero row input df), a row with row-label No data to report will be included in the table.
subcol_split	(string) text to search colid to determine whether further subsetting should be performed.
subcol_var	(string) name of variable containing to be searched for the text identified in subcol_val argument.
subcol_val	(string) value to use to perform further data sub-setting.
.df_row	(data.frame) data frame across all of the columns for the given row split.
.spl_context	(data.frame) gives information about ancestor split states that is passed by rtables.
.N_col	(integer) column-wise N (column count) for the full column being analyzed that is typically passed by rtables.
id	(string) subject variable name.

denom	(string) One of • N_col Column count, • n_df Number of patients (based upon the main input dataframe df), • n_altdf Number of patients from the secondary dataframe (.alt_df_full), Note that argument denom_by will perform a row-split on the .alt_df_full dataframe. It is a requirement that variables specified in denom_by are part of the row split specifications. • n_rowdf Number of patients from the current row-level dataframe (.row_df from the rtables splitting machinery). • n_parentdf Number of patients from a higher row-level split than the current split. This higher row-level split is specified in the argument denom_by.
label	(string) When val has length 1, the row label to be shown on the output can be specified using this argument. When val is a character vector, the label_map argument can be specified to control the row-labels.
label_fstr	(string) a sprintf style format string. It can contain up to one "%s", which takes the current split value and generates the row/column label. It will be combined with the labelstr argument, when utilizing this function as a cfun in a summarize_row_groups call. It is recommended not to utilize this argument for other purposes. The label argument could be used instead (if val is a single string)
label_map	(data.frame) A mapping data frame used to translate levels of the analysis variable(s) into the row labels displayed in the table, optionally conditioned on the nearest row-split. Four types of mappings are supported: 1. Single-variable mapping. Column names: level, label. 2. Multi-variable mapping. Column names: var, level, label. 3. Conditional-on-row-split mapping. Column names: <split_var>, level, label. 4. Conditional-on-row-split multi-variable mapping. Column names: <split_var>, var, level, label. Here, <split_var> is a placeholder for the name of the variable used for the nearest row-split (e.g. SEX, PARAMCD).

Recommendation: ensure input variable(s) are of type factor, to avoid the error "got a label map that doesn't provide labels for all values".

See examples for more details.

.alt_df_full	(dataframe) Denominator dataset for fraction and relative risk calculations. this argument gets populated by the rtables split machinery (see rtables::additional_fun_params).
denom_by	(character) Variables from row-split to be used in the denominator derivation. This controls both denom = "n_parentdf" and denom = "n_altdf". When denom = "n_altdf", the denominator is derived from .alt_df_full in combination with denom_by argument
.stats	(character) statistics to select for the table.
.formats	(named 'character' or 'list') formats for the statistics.
.labels_n	(named character) String to control row labels for the 'n'-statistics. Only useful when more than one 'n'-statistic is requested (rare situations only).
.indent_mods	(named integer) indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.
na_str	(string) string used to replace all NA or empty values in the output.

Value

list of requested statistics with formatted `rtables::CellValue()`.

See Also

[a_freq_j\(\)](#) for extensive label_map usage examples.

Examples

```
library(dplyr)

ADSL <- ex_ads1 |>
  select(USUBJID, ARM)

ADSL$COLSPAN_REL <- "AEs"

ADAE <- ex_adae |>
  select(USUBJID, ARM, AEDECOD, AREL)

ADAE <- ADAE |>
  mutate(
```

```

    AEREL = case_when(
      AREL == "Y" ~ "RELATED",
      AREL == "N" ~ "NOT RELATED"
    ),
    AEREL = factor(AEREL),
    COLSPAN_REL = "AEs"
  )

  combodf <- tribble(
    ~valname, ~label, ~levelcombo, ~exargs,
    "RELATED", "Related AEs", c("AEs"), list()
  )

  lyt <- basic_table(show_colcounts = TRUE) |>
    split_cols_by("COLSPAN_REL", split_fun = rtables::add_combo_levels(combodf, trim = TRUE)) |>
    split_cols_by("ARM") |>
    analyze("AEDECOD",
      afun = a_freq_subcol_j,
      extra_args = list(
        subcol_split = "RELATED",
        subcol_var = "AEREL",
        subcol_val = "RELATED"
      )
    )

  result <- build_table(lyt, ADAE, alt_counts_df = ADSL)

  result

```

a_kaplan_meier

a_kaplan_meier

Description

[Deprecated] Use `tern::a_surv_time()` instead.

Usage

```

a_kaplan_meier(
  df,
  .var,
  ...,
  .stats = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)

```

Arguments

df	(data.frame) input data.
.var	(string) variable name.
...	additional arguments (ignored).
.stats	(character) statistics to compute (ignored).
.formats	(list) formats for statistics (ignored).
.labels	(list) labels for statistics (ignored).
.indent_mods	(list) indentation modifications (ignored).

a_maxlev	<i>Calculate Count and Percentage of the Maximum Level of an Ordered Factor per Subject.</i>
----------	--

Description

A formatted analysis function used as an afun in [analyze](#) and as a cfun in [summarize_row_groups](#).

It computes count and proportion statistics for the maximum level of an ordered factor, `df[[.var]]`, for each unique subject in `df[[id]]`. Specifically, for each subject, the function identifies the highest level of `df[[.var]]`, producing one value per subject. Then, if `any_level = TRUE`, the function reports the total number of maximum values, excluding those specified in `any_level_exclude`. Otherwise, it tabulates the frequency of each maximum level across all subjects.

This function is particularly useful for identifying the maximum severity of adverse events in a treatment sequence, where the most severe event experienced by a subject is used for reporting.

Usage

```
a_maxlev(
  df,
  labelstr = NULL,
  .var,
  .spl_context,
  id = "USUBJID",
  .alt_df_full = NULL,
  any_level = FALSE,
  any_level_exclude = "Missing",
  ...
)
```

Arguments

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

labelstr	(character) label of the level of the parent split currently being summarized (must be present as second argument in Content Row Functions). See <code>rtables::summarize_row_groups()</code> for more information.
.var	(string) single variable name that is passed by <code>rtables</code> when requested by a statistics function.
.spl_context	(data.frame) gives information about ancestor split states that is passed by <code>rtables</code> .
id	(string) subject variable name.
.alt_df_full	(dataframe) A dataset used to compute the denominator for proportions. This is required when the same subject appears multiple times in the dataset due to treatment sequences. <code>colnames(.alt_df_full)</code> must be a superset of <code>id</code> . This argument gets populated by the <code>rtables</code> split machinery (see <code>rtables::additional_fun_params</code>).
any_level	(flag) Should be set to TRUE when the function is used as a cfun.
any_level_exclude	(character) Applicable only when <code>any_level = TRUE</code> . Specifies levels of <code>df[[.var]]</code> to exclude from the statistic (default = "Missing").
...	additional arguments for the lower level functions.

Details

For each unique subject, only the maximum level of the ordered factor `df[[.var]]` is included in the final count and percentage statistics.

Value

A `RowsVerticalSection` object.

Note

The denominator for proportions is computed using the `denom_df` argument. This serves as a temporary workaround until the next version of `rtables` is released, which will support `.alt_count_df` for use in `afun/cfun`.

Author(s)

WW

Examples

```
treatments <- factor(c("a", "b", "c"))
ae_severities <- c("Missing", "Mild", "Moderate", "Severe")
ae_severities <- ordered(ae_severities, levels = ae_severities)
```

```

my_adae <- data.frame(
  ID = c(1, 1, 1, 2, 2, 3, 3, 3, 3, 4),
  TRT = factor(c("a", "b", "b", "b", "c", "c", "a", "c", "b", "b")),
  AESEV = ae_severities[c(4L, 1L, 2L, 1L, 2L, 1L, 2L, 3L, 1L, 2L)]
)
my_adsl <- data.frame(
  ID = rep(1:5, each = 3),
  TRT = factor(rep(c("a", "b", "c"), times = 5))
)

aesevall_spf <- make_combo_splitfun(
  nm = "AESEV_ALL",
  label = "Any AE",
  levels = NULL,
)

lyt <- basic_table() |>
  split_cols_by("TRT") |>
  add_overall_col("Total") |>
  split_rows_by("AESEV", split_fun = aesevall_spf) |>
  summarize_row_groups(
    "AESEV",
    cfun = a_maxlev,
    extra_args = list(id = "ID", any_level = TRUE)
  ) |>
  analyze(
    "AESEV",
    afun = a_maxlev,
    extra_args = list(id = "ID")
  )
build_table(lyt, my_adae, alt_counts_df = my_adsl)

```

a_proportion_ci_factor

Formatted Analysis Function For Proportion Confidence Interval for Factor

Description

Formatted Analysis Function For Proportion Confidence Interval for Factor.

Usage

```
a_proportion_ci_factor(df, .var, ...)
```

Arguments

df (data.frame)
including factor .var.

`.var` (string)
name of the factor variable.

... see [a_proportion_ci_logical\(\)](#) for additionally required arguments.

Value

The `rtables::rcell()` result.

Examples

```
a_proportion_ci_factor(
  df = DM,
  .var = "SEX",
  .alt_df = DM,
  conf_level = 0.95,
  formats = list(prop_ci = jjcsformat_xx("xx.x%, xx.x%")),
  method = "clopper-pearson"
)
```

`a_proportion_ci_logical`

Formatted Analysis Function For Proportion Confidence Interval for Logical

Description

Formatted Analysis Function For Proportion Confidence Interval for Logical.

Usage

```
a_proportion_ci_logical(x, .alt_df, conf_level, method, formats)
```

Arguments

`x` (logical)
including binary response values.

`.alt_df` (data.frame)
alternative data frame used for denominator calculation.

`conf_level` (numeric)
confidence level for the confidence interval.

`method` (string)
please see [tern::s_proportion\(\)](#) for possible methods.

`formats` (list)
including element `prop_ci` with the required format. Note that the value is in percent already.

Value

The `rtables::rcell()` result.

Examples

```
a_proportion_ci_logical(
  x = DM$SEX == "F",
  .alt_df = DM,
  conf_level = 0.95,
  formats = list(prop_ci = jjcsformat_xx("xx.xx% - xx.xx%")),
  method = "wald"
)
```

a_relative_risk	<i>Relative risk estimation</i>
-----------------	---------------------------------

Description

The analysis function `a_relative_risk()` is used to create a layout element to estimate the relative risk for response within a studied population. Only the CMH method is available currently. The primary analysis variable, `vars`, is a logical variable indicating whether a response has occurred for each record. A stratification variable must be supplied via the `strata` element of the `variables` argument.

Usage

```
a_relative_risk(
  df,
  .var,
  ref_path,
  .spl_context,
  ...,
  .stats = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)

s_relative_risk(
  df,
  .var,
  .ref_group,
  .in_ref_col,
  variables = list(strata = NULL),
  conf_level = 0.95,
  method = "cmh",
  weights_method = "cmh"
)
```

Arguments

<code>df</code>	(<code>data.frame</code>) input data frame.
<code>.var</code>	(<code>string</code>) name of the response variable.
<code>ref_path</code>	(<code>character</code>) path to the reference group.
<code>.spl_context</code>	(<code>environment</code>) split context environment.
<code>...</code>	Additional arguments passed to the statistics function.
<code>.stats</code>	(<code>character</code>) statistics to calculate.
<code>.formats</code>	(<code>list</code>) formats for the statistics.
<code>.labels</code>	(<code>list</code>) labels for the statistics.
<code>.indent_mods</code>	(<code>list</code>) indentation modifications for the statistics.
<code>.ref_group</code>	(<code>data.frame</code>) reference group data frame.
<code>.in_ref_col</code>	(<code>logical</code>) whether the current column is the reference column.
<code>variables</code>	(<code>list</code>) list with strata variable names.
<code>conf_level</code>	(<code>numeric</code>) confidence level for the confidence interval.
<code>method</code>	(<code>string</code>) method to use for relative risk calculation.
<code>weights_method</code>	(<code>string</code>) method to use for weights calculation in stratified analysis.

Details

The variance of the CMH relative risk estimate is calculated using the Greenland and Robins (1985) variance estimation.

Value

- `a_relative_risk()` returns the corresponding list with formatted `rtables::CellValue()`.
- `s_relative_risk()` returns a named list of elements `rel_risk_ci` and `pval`.

Functions

- `a_relative_risk()`: Formatted analysis function which is used as `afun`. Note that the `junco` specific `ref_path` and `.spl_context` arguments are used for reference column information.
- `s_relative_risk()`: Statistics function estimating the relative risk for response.

Note

This has been adapted from the `odds_ratio` functions in the `tern` package.

Examples

```
nex <- 100
dta <- data.frame(
  "rsp" = sample(c(TRUE, FALSE), nex, TRUE),
  "grp" = sample(c("A", "B"), nex, TRUE),
  "f1" = sample(c("a1", "a2"), nex, TRUE),
  "f2" = sample(c("x", "y", "z"), nex, TRUE),
  stringsAsFactors = TRUE
)

l <- basic_table() |>
  split_cols_by(var = "grp") |>
  analyze(
    vars = "rsp",
    afun = a_relative_risk,
    extra_args = list(
      conf_level = 0.90,
      variables = list(strata = "f1"),
      ref_path = c("grp", "B")
    )
  )

build_table(l, df = dta)
nex <- 100
dta <- data.frame(
  "rsp" = sample(c(TRUE, FALSE), nex, TRUE),
  "grp" = sample(c("A", "B"), nex, TRUE),
  "f1" = sample(c("a1", "a2"), nex, TRUE),
  "f2" = sample(c("x", "y", "z"), nex, TRUE),
  stringsAsFactors = TRUE
)

s_relative_risk(
  df = subset(dta, grp == "A"),
  .var = "rsp",
  .ref_group = subset(dta, grp == "B"),
  .in_ref_col = FALSE,
  variables = list(strata = c("f1", "f2")),
  conf_level = 0.90
)
```

a_sum_ratio_j

*Analysis Function: Sum and Ratio of Sums***Description****[Stable]**

Computes one or more of the following statistics for use with `rtables::analyze()`:

- "sum": `sum(.var)`
- "sum_unique": `sum(.var)` after keeping unique rows by `id_var`
- "ratio": `sum(.var) / sum(denom_by)`
- "ratio_unique": same ratio after keeping unique rows by `id_var`

Both `.var` and `denom_by` columns must be numeric (integer, double, or logical coerced to numeric).

Usage

```
a_sum_ratio_j(
  df,
  .var,
  .stats = "sum",
  denom_by = NULL,
  id_var = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL,
  na_str = NULL,
  ...
)
```

Arguments

<code>df</code>	(<code>data.frame</code>) data set containing all analysis variables.
<code>.var</code>	(<code>string</code>) single variable name that is passed by <code>rtables</code> when requested by a statistics function.
<code>.stats</code>	(<code>character</code>) one or more of "sum", "sum_unique", "ratio", "ratio_unique". Default: "sum".
<code>denom_by</code>	(<code>string</code> or <code>NULL</code>) denominator column; required for "ratio" and "ratio_unique".
<code>id_var</code>	(<code>string</code> or <code>NULL</code>) subject-id column for unique row selection; required for "sum_unique" and "ratio_unique".

.formats	(named character or list) formats for the statistics. See Details in analyze_vars for more information on the 'auto' setting.
.labels	(named character) labels for the statistics (without indent).
.indent_mods	(named integer or NULL) indent modifiers for the row labels. Defaults to NULL (no modification).
na_str	(character or NULL) string(s) used to replace NA values in the formatted output. Defaults to NULL (no replacement).
...	additional arguments for the lower level functions.

Value

A RowsVerticalSection for use by rtables.

Author(s)

VR

Examples

```
df <- data.frame(
  USUBJID = c("S01", "S01", "S02", "S03"),
  ARM     = factor(c("A", "A", "A", "B")),
  EVENTS  = c(1, 1, 0, 1),
  DAYS    = c(10, 10, 20, 15)
)

# All 4 stats in a layout (S01 duplicate tests unique logic)
lyt <- basic_table() |>
  split_cols_by("ARM") |>
  analyze(
    "EVENTS",
    afun = a_sum_ratio_j,
    extra_args = list(
      .stats = c("sum", "sum_unique", "ratio", "ratio_unique"),
      denom_by = "DAYS",
      id_var = "USUBJID"
    )
  )

build_table(lyt, df)
```

a_summarize_ancova_j *ANCOVA Summary Function*

Description

Combination of [tern::s_summary](#), and ANCOVA based estimates for mean and diff between columns, based on ANCOVA function `s_ancova_j`.

Usage

```
a_summarize_ancova_j(
  df,
  .var,
  .df_row,
  ref_path,
  .spl_context,
  ...,
  .stats = c("n", "mean_sd", "median", "range", "quantiles", "lsmean_se", "lsmean_ci",
    "lsmean_diff_with_ci", "pval"),
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)

s_summarize_ancova_j(df, .var, .df_row, .ref_group, .in_ref_col, ...)
```

Arguments

<code>df</code>	(data.frame) data set containing all analysis variables.
<code>.var</code>	(string) single variable name that is passed by <code>rtables</code> when requested by a statistics function.
<code>.df_row</code>	(data.frame) data set that includes all the variables that are called in <code>.var</code> and variables.
<code>ref_path</code>	(character) path to the reference group.
<code>.spl_context</code>	(environment) split context environment.
<code>...</code>	Additional arguments passed to <code>s_ancova_j</code> .
<code>.stats</code>	(character) statistics to calculate.
<code>.formats</code>	(list) formats for the statistics.

.labels	(list) labels for the statistics.
.indent_mods	(list) indentation modifications for the statistics.
.ref_group	(data.frame or vector) the data corresponding to the reference group.
.in_ref_col	(flag) TRUE when working with the reference level, FALSE otherwise.

Details

Combination of [tern::s_summary](#), and ANCOVA based estimates for mean and diff between columns, based on ANCOVA function `s_ancova_j`

Value

- `a_summarize_ancova_j()` returns the corresponding list with formatted [rtables::CellValue\(\)](#).

returns the statistics from `tern::s_summary(x)`, appended with a new statistics based upon ANCOVA

Functions

- `a_summarize_ancova_j()`: Formatted analysis function which is used as `afun`. Note that the `junco` specific `ref_path` and `.spl_context` arguments are used for reference column information.

See Also

Other Inclusion of ANCOVA Functions: [a_summarize_aval_chg_diff_j\(\)](#), [s_ancova_j\(\)](#)

Examples

```
basic_table() |>
  split_cols_by("Species") |>
  add_colcounts() |>
  analyze(
    vars = "Petal.Length",
    afun = a_summarize_ancova_j,
    show_labels = "hidden",
    na_str = tern::default_na_str(),
    table_names = "unadj",
    var_labels = "Unadjusted comparison",
    extra_args = list(
      variables = list(arm = "Species", covariates = NULL),
      conf_level = 0.95,
      .labels = c(lsmean = "Mean", lsmean_diff = "Difference in Means"),
      ref_path = c("Species", "setosa")
    )
  ) |>
  analyze(
```

```

vars = "Petal.Length",
afun = a_summarize_ancova_j,
show_labels = "hidden",
na_str = tern::default_na_str(),
table_names = "adj",
var_labels = "Adjusted comparison (covariates: Sepal.Length and Sepal.Width)",
extra_args = list(
  variables = list(
    arm = "Species",
    covariates = c("Sepal.Length", "Sepal.Width")
  ),
  conf_level = 0.95,
  ref_path = c("Species", "setosa")
)
) |>
build_table(iris)

library(dplyr)
library(tern)

df <- iris |> filter(Species == "virginica")
.df_row <- iris
.var <- "Petal.Length"
variables <- list(arm = "Species", covariates = "Sepal.Length * Sepal.Width")
.ref_group <- iris |> filter(Species == "setosa")
conf_level <- 0.95
s_summarize_ancova_j(
  df,
  .var = .var,
  .df_row = .df_row,
  variables = variables,
  .ref_group = .ref_group,
  .in_ref_col = FALSE,
  conf_level = conf_level
)

```

a_summarize_aval_chg_diff_j

Analysis function 3-column presentation

Description

Analysis functions to produce a 1-row summary presented in a 3-column layout in the columns (column 1 = N, column 2 = Value, column 3 = Change).

In the difference columns, only 1 column will be presented : difference + CI

When ancova = TRUE, the presented statistics will be based on ANCOVA method (s_summarize_ancova_j). mean and ci (both for Value (column 2) and CHG (column 3)) using statistic lsmean_ci

mean and ci for the difference column are based on same ANCOVA model using statistic lsmean_diff_with_ci

When ancova = FALSE, descriptive statistics will be used instead.

In the difference column, the 2-sample t-test will be used.

Usage

```

a_summarize_aval_chg_diff_j(
  df,
  .df_row,
  .spl_context,
  ancova = FALSE,
  comp_btw_group = TRUE,
  ref_path = NULL,
  .N_col,
  denom = c("N", ".N_col"),
  indatavar = NULL,
  d = 0,
  id = "USUBJID",
  interaction_y = FALSE,
  interaction_item = NULL,
  conf_level = 0.95,
  variables = list(arm = "TRT01A", covariates = NULL),
  format_na_str = "",
  .stats = list(col1 = "count_denom_frac", col23 = "mean_ci_3d", coldiff =
    "mean_diff_with_ci"),
  .formats = list(col1 = NULL, col23 = "xx.dx (xx.dx, xx.dx)", coldiff =
    "xx.dx (xx.dx, xx.dx)"),
  .formats_fun = list(col1 = jjcsformat_count_denom_fraction, col23 = jjcsformat_xx,
    coldiff = jjcsformat_xx),
  multivars = c("AVAL", "AVAL", "CHG"),
  weights_emmeans = NULL,
  method_combo = c("contrasts", "collapse"),
  weights_combo = NULL
)

```

Arguments

df	(data.frame) data set containing all analysis variables.
.df_row	(data.frame) data frame across all of the columns for the given row split.
.spl_context	(data.frame) gives information about ancestor split states that is passed by rtables.
ancova	(logical) If FALSE, only descriptive methods will be used. If TRUE, ANCOVA methods will be used for each of the columns : AVAL, CHG, DIFF.
comp_btw_group	(logical) If TRUE, comparison between groups will be performed. When ancova = FALSE, the estimate of between group difference (on CHG) will be based upon a two-sample t-test.

	When <code>ancova = TRUE</code> , the same ANCOVA model will be used for the estimate of between group difference (on CHG).
<code>ref_path</code>	(character) global reference group specification, see <code>get_ref_info()</code> .
<code>.N_col</code>	(integer) column-wise N (column count) for the full column being analyzed that is typically passed by <code>rtables</code> .
<code>denom</code>	(string) choice of denominator for proportions. Options are: • <code>N</code>: number of records in this column/row split. There is no check in place that the current split only has one record per subject. Users should be careful with this. • <code>.N_col</code>: number of records in this column intersection (based on <code>alt_counts_df</code> dataset) (when <code>alt_counts_df</code> is a single record per subjects, this will match number of subjects)
<code>indatavar</code>	(string) If not null, variable name to extra subset incoming <code>df</code> to non-missing values of this variable.
<code>d</code>	(default = 1) choice of Decimal precision. Note that one extra precision will be added, as means are presented. Options are: • <code>numerical(1)</code> • variable name containing information on the precision, this variable should be available on input dataset. The content of this variable should then be an integer.
<code>id</code>	(string) subject variable name.
<code>interaction_y</code>	(character) Will be passed onto the tern function <code>s_ancova</code> , when <code>ancova = TRUE</code> .
<code>interaction_item</code>	(character) Will be passed onto the tern function <code>s_ancova</code> , when <code>ancova = TRUE</code> .
<code>conf_level</code>	(proportion) Confidence level of the interval
<code>variables</code>	(named list of strings) list of additional analysis variables, with expected elements: • <code>arm</code> (string) group variable, for which the covariate adjusted means of multiple groups will be summarized. Specifically, the first level of <code>arm</code> variable is taken as the reference group. • <code>covariates</code> (character) a vector that can contain single variable names (such as <code>'X1'</code>), and/or interaction terms indicated by <code>'X1 * X2'</code>.

format_na_str	(string)
.stats	(named list) column statistics to select for the table. The following column names are to be used: col1, col23, coldiff. For col1, the following stats can be specified. For col23, only mean_ci_3d is available. When ancova = TRUE these are LS Means, otherwise, arithmetic means. For coldiff, only mean_diff_with_ci is available. When ancova = TRUE these are LS difference in means, otherwise, difference in means based upon 2-sample t-test.
.formats	(named list) formats for the column statistics. xx.d style formats can be used.
.formats_fun	(named list) formatting functions for the column statistics, to be applied after the conversion of xx.d style to the appropriate precision.
multivars	(string(3)) Variables names to use in 3-col layout.
weights_emmeans	(string) argument from <code>emmeans::emmeans()</code>, "counterfactual" by default.
method_combo	(string) Method for derivations in combined column. • contrast Derivations for the combined level are done through contrasts from the original model (using weights per weights_combo specifications). • collapse The ancova model for the combined group will be performed with group levels that contribute to the combination collapsed into a single combined level. For more information see the vignette ANCOVA with Combined Treatment Groups.
weights_combo	(string) Weights for the contrasts of the combined levels. • equal 1/(number of levels from arm variable included in the combination) • proportional, proportional_marginal weight for each level included in the combination is proportional to number of observations in that level The difference between proportional and proportional_marginal is only relevant when the model includes an interaction between arm and other factor variable (interaction_item). proportional_marginal interprets proportional over all levels of interaction_item, ie, the same weights will be used for all levels of interaction_item. For proportional the weights will be derived within the requested level (interaction_y) for interaction_item.

Details

See Description

Value

A function that can be used in an analyze function call

See Also

Other Inclusion of ANCOVA Functions: [a_summarize_ancova_j\(\)](#), [s_ancova_j\(\)](#)

Examples

```
library(dplyr)

ADEG <- data.frame(
  STUDYID = c(
    "DUMMY", "DUMMY", "DUMMY", "DUMMY", "DUMMY",
    "DUMMY", "DUMMY", "DUMMY", "DUMMY", "DUMMY"
  ),
  USUBJID = c(
    "XXXXX01", "XXXXX02", "XXXXX03", "XXXXX04", "XXXXX05",
    "XXXXX06", "XXXXX07", "XXXXX08", "XXXXX09", "XXXXX10"
  ),
  TRT01A = c(
    "ARMA", "ARMA", "ARMA", "ARMA", "ARMA", "Placebo",
    "Placebo", "Placebo", "ARMA", "ARMA"
  ),
  PARAM = c("BP", "BP", "BP", "BP", "BP", "BP", "BP", "BP", "BP", "BP"),
  AVISIT = c(
    "Visit 1", "Visit 1", "Visit 1", "Visit 1", "Visit 1",
    "Visit 1", "Visit 1", "Visit 1", "Visit 1", "Visit 1"
  ),
  AVAL = c(56, 78, 67, 87, 88, 93, 39, 87, 65, 55),
  CHG = c(2, 3, -1, 9, -2, 0, 6, -2, 5, 2)
)

ADEG <- ADEG |>
  mutate(
    TRT01A = as.factor(TRT01A),
    STUDYID = as.factor(STUDYID)
  )

ADEG$colspan_trt <- factor(ifelse(ADEG$TRT01A == "Placebo", " ", "Active Study Agent"),
  levels = c("Active Study Agent", " ")
)

ADEG$rrisk_header <- "Risk Difference (%) (95% CI)"
ADEG$rrisk_label <- paste(ADEG$TRT01A, paste("vs", "Placebo"))

colspan_trt_map <- create_colspan_map(ADEG,
  non_active_grp = "Placebo",
  non_active_grp_span_lbl = " ",
  active_grp_span_lbl = "Active Study Agent",
  colspan_var = "colspan_trt",
  trt_var = "TRT01A"
)
```

```

ref_path <- c("colspan_trt", " ", "TRT01A", "Placebo")

lyt <- basic_table() |>
  split_cols_by(
    "colspan_trt",
    split_fun = trim_levels_to_map(map = colspan_trt_map)
  ) |>
  split_cols_by("TRT01A") |>
  split_rows_by(
    "PARAM",
    label_pos = "topleft",
    split_label = "Blood Pressure",
    section_div = " ",
    split_fun = drop_split_levels
  ) |>
  split_rows_by(
    "AVISIT",
    label_pos = "topleft",
    split_label = "Study Visit",
    split_fun = drop_split_levels,
    child_labels = "hidden"
  ) |>
  split_cols_by_multivar(
    c("AVAL", "AVAL", "CHG"),
    varlabels = c("n/N (%)", "Mean (CI)", "CFB (CI)")
  ) |>
  split_cols_by("rrisk_header", nested = FALSE) |>
  split_cols_by(
    "TRT01A",
    split_fun = remove_split_levels("Placebo"),
    labels_var = "risk_label"
  ) |>
  split_cols_by_multivar(c("CHG"), varlabels = c(" ")) |>
  analyze("STUDYID",
    afun = a_summarize_aval_chg_diff_j,
    extra_args = list(
      format_na_str = "-", d = 0,
      ref_path = ref_path, variables = list(arm = "TRT01A", covariates = NULL)
    )
  )

result <- build_table(lyt, ADEG)

result

```

a_summarize_ex_j

*Tabulation for Exposure Tables***Description**

A function to create the appropriate statistics needed for exposure table

Usage

```

s_summarize_ex_j(
  df,
  .var,
  .df_row,
  .spl_context,
  comp_btw_group = TRUE,
  ref_path = NULL,
  ancova = FALSE,
  interaction_y,
  interaction_item,
  conf_level,
  daysconv,
  variables
)

a_summarize_ex_j(
  df,
  .var,
  .df_row,
  .spl_context,
  comp_btw_group = TRUE,
  ref_path = NULL,
  ancova = FALSE,
  interaction_y = FALSE,
  interaction_item = NULL,
  conf_level = 0.95,
  variables,
  .stats = c("mean_sd", "median", "range", "quantiles", "total_subject_years"),
  .formats = c(diff_mean_est_ci = jjcsformat_xx("xx.xx (xx.xx, xx.xx)")),
  .labels = c(quantiles = "Interquartile range"),
  .indent_mods = NULL,
  na_str = rep("NA", 3),
  daysconv = 1
)

```

Arguments

<code>df</code>	(data.frame) data set containing all analysis variables.
<code>.var</code>	(string) single variable name that is passed by <code>rtables</code> when requested by a statistics function.
<code>.df_row</code>	(data.frame) data frame across all of the columns for the given row split.
<code>.spl_context</code>	(data.frame) gives information about ancestor split states that is passed by <code>rtables</code> .

comp_btw_group	(logical) If TRUE, comparison between groups will be performed. When ancova = FALSE, the estimate of between group difference (on CHG) will be based upon two-sample t-test. When ancova = TRUE, the same ANCOVA model will be used for the estimate of between group difference (on CHG).
ref_path	(character) global reference group specification, see get_ref_info() .
ancova	(logical) If FALSE, only descriptive methods will be used. If TRUE, ANCOVA methods will be used for each of the columns : AVAL, CHG, DIFF.
interaction_y	(character) Will be passed onto the tern function s_ancova, when ancova = TRUE.
interaction_item	(character) Will be passed onto the tern function s_ancova, when ancova = TRUE.
conf_level	(proportion) Confidence level of the interval
daysconv	(numeric) conversion required to get the values into days (i.e 1 if original PARAMCD unit is days, 30.4375 if original PARAMCD unit is in months)
variables	(named list of strings) list of additional analysis variables, with expected elements: • arm (string) group variable, for which the covariate adjusted means of multiple groups will be summarized. Specifically, the first level of arm variable is taken as the reference group. • covariates (character) a vector that can contain single variable names (such as 'X1'), and/or interaction terms indicated by 'X1 * X2'.
.stats	(character) statistics to select for the table.
.formats	(named character or list) formats for the statistics. See Details in analyze_vars for more information on the 'auto' setting.
.labels	(named character) labels for the statistics (without indent).
.indent_mods	(named integer) indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.
na_str	(string) string used to replace all NA or empty values in the output.

Details

Creates statistics needed for standard exposure table. This includes differences and 95% CI and total treatment years. This is designed to be used as an analysis (afun in analyze) function.

Value

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

Functions

- `s_summarize_ex_j()`: Statistics function needed for the exposure tables.
- `a_summarize_ex_j()`: Formatted analysis function which is used as afun.

Examples

```
library(dplyr)
ADEX <- ex_adsl |> select(USUBJID, ARM, TRTSDTM, EOSSTT, EOSDY)

trtvar <- "ARM"
ctrl_grp <- "B: Placebo"
cutoffd <- as.Date("2023-09-24")

ADEX <- ADEX |>
  create_colspan_var(
    non_active_grp      = ctrl_grp,
    non_active_grp_span_lbl = " ",
    active_grp_span_lbl  = "Active Study Agent",
    colspan_var          = "colspan_trt",
    trt_var              = trtvar
  ) |>
  mutate(
    diff_header = "Difference in Means (95% CI)",
    diff_label = paste(!rlang::sym(trtvar), "vs", ctrl_grp),
    TRTDURY = case_when(
      !is.na(EOSDY) ~ EOSDY,
      TRUE ~ as.integer(cutoffd - as.Date(TRTSDTM) + 1)
    )
  )

colspan_trt_map <- create_colspan_map(ADEX,
  non_active_grp = ctrl_grp,
  non_active_grp_span_lbl = " ",
  active_grp_span_lbl = "Active Study Agent",
  colspan_var = "colspan_trt",
  trt_var = trtvar
)

ref_path <- c("colspan_trt", "", trtvar, ctrl_grp)

lyt <- basic_table() |>
  split_cols_by(
```

```

      "colspan_trt",
      split_fun = trim_levels_to_map(map = colspan_trt_map)
    ) |>
    split_cols_by(trtvar) |>
    split_cols_by("diff_header", nested = FALSE) |>
    split_cols_by(
      trtvar,
      split_fun = remove_split_levels(ctrl_grp),
      labels_var = "diff_label"
    ) |>
    analyze("EOSDY",
      afun = a_summarize_ex_j, var_labels = "Duration of treatment (Days)",
      show_labels = "visible",
      indent_mod = 0L,
      extra_args = list(
        daysconv = 1,
        ref_path = ref_path,
        variables = list(arm = trtvar, covariates = NULL),
        ancova = TRUE,
        comp_btw_group = TRUE
      )
    )

result <- build_table(lyt, ADEX, alt_counts_df = ADEX)
result

```

a_summary_j

*Wrappers Around tern::a_summary() with Row Split Exclusion***Description****[Stable]**

These wrappers around `tern::a_summary()` allow selected row split levels to be excluded from producing the analysis.

Usage

```
a_summary_j_with_exclude(df, .var, exclude_levels, .spl_context, ...)
```

Arguments

df	(data.frame) data set containing all analysis variables.
.var	(string) single variable name that is passed by rtables when requested by a statistics function.
exclude_levels	(list) A named list where names correspond to split variables and values are vectors of levels to exclude.

```
.spl_context    (data.frame)
                 gives information about ancestor split states that is passed by rtables.
...             Additional arguments passed to tern::a_summary(), excluding the x argument, which is constructed internally from df[[.var]].
```

Details

`tern::a_summary()` is a formatted analysis function that computes summary statistics for a single variable. In typical use within the `rtables` framework, the analysis variable is first extracted from a data frame and then passed to `tern::a_summary()` as a standalone vector.

`a_summary_j_with_exclude()` extends this behavior by:

- accepting a data frame (`df`) as input,
- checking whether the current row split path should be excluded,
- returning `NULL` for excluded row split levels,
- extracting the analysis variable `.var` for all other row split levels,
- and passing the resulting vector to `tern::a_summary()`.

Value

Returns the same type of object as `tern::a_summary()`, or `NULL` when the current row split path is excluded by `exclude_levels`.

Functions

- `a_summary_j_with_exclude()`: Analysis function which can exclude row split levels from producing the analysis. These have to be specified in the `exclude_levels` argument, see `?do_exclude_split` for details.

See Also

`tern::a_summary()`, `do_exclude_split()`

Examples

```
df <- data.frame(
  USUBJID = 1:8,
  AVISIT = factor(rep(c("Baseline", "Week 1"), 4)),
  ARM = factor(rep(c("A", "B"), each = 4)),
  AVAL = c(1.2, 3, -1.5, 0.8, 3.5, 2, -1.1, 0.3)
)

lyt <- basic_table() |>
  split_cols_by("ARM") |>
  split_rows_by("AVISIT") |>
  analyze(
    "AVAL",
    afun = a_summary_j_with_exclude,
    extra_args = list(exclude_levels = list(AVISIT = "Baseline"))
  )
```

```

    )

    build_table(lyt, df)

```

a_two_tier

Two Tier Analysis Function

Description

`a_two_tier()`: The analysis function used as an `afun` in [analyze](#). This function simulates a final additional level of nesting with a traditional `analyze` call inside it.

This makes it possible to create what *appear to be* group summary or content rows and to *optionally or conditionally* generate one or more "detail" rows underneath it.

For example, in a disposition table, one might want counts for completed and ongoing patients with no further detail underneath, but a breakdown of specific reasons beneath the count of patients who discontinued treatment.

`a_three_tier()`: Extends `a_two_tier()` by adding a third level of nesting: a group summary outer row, mid-level detail rows (one per level of `inner_var1`), and innermost detail rows (one per level of `inner_var2`) nested beneath each mid-level row.

Usage

```

a_two_tier(
  df,
  labelstr = NULL,
  .var,
  .N_col,
  .df_row,
  inner_var,
  drill_down_levs,
  .spl_context,
  use_all_levels = FALSE,
  grp_fun,
  detail_fun,
  .alt_df_full = NULL,
  ...
)

```

```

a_three_tier(
  df,
  labelstr = NULL,
  .var,
  .N_col,
  .df_row,
  inner_var1,
  inner_var2,

```

```

    drill_down_levs,
    .spl_context,
    use_all_levels = FALSE,
    grp_fun,
    detail_fun,
    .alt_df_full = NULL,
    ...
  )

```

Arguments

<code>df</code>	(data.frame) data set containing all analysis variables.
<code>labelstr</code>	(character) label of the level of the parent split currently being summarized (must be present as second argument in Content Row Functions). See rtables::summarize_row_groups() for more information.
<code>.var</code>	(string) single variable name that is passed by <code>rtables</code> when requested by a statistics function.
<code>.N_col</code>	(integer) column-wise N (column count) for the full column being analyzed that is typically passed by <code>rtables</code> .
<code>.df_row</code>	(data.frame) data frame across all of the columns for the given row split.
<code>inner_var</code>	(string) single variable name to use when generating the detail rows.
<code>drill_down_levs</code>	(character) the level(s) of <code>.var</code> under which detail rows should be generated.
<code>.spl_context</code>	(data.frame) gives information about ancestor split states that is passed by <code>rtables</code> .
<code>use_all_levels</code>	(flag) controls which factor levels will be present for <code>inner_var</code> (both in <code>df/x</code> and in <code>.df_row</code>) when calling <code>detail_fun</code> . If TRUE, all levels (those present on the factor <code>.df_row[[inner_var]]</code> , *regardless if the level is observed in the row group or not) will be present when creating detail rows. Otherwise (the default), only the levels observed <i>anywhere in the row group, i.e., within .df_row</i> will be present.
<code>grp_fun</code>	(function) analysis function to be used when generating the "group summary" outer rows.
<code>detail_fun</code>	(function) analysis function to be used when generating "detail" inner rows.
<code>.alt_df_full</code>	(dataframe) denominator dataset for fraction and relative risk calculations. this argument gets populated by the <code>rtables</code> split machinery (see rtables::additional_fun_params).

...	additional arguments passed directly to grp_fun and detail_fun.
inner_var1	(string) single variable name to use when generating the mid-level (second tier) detail rows.
inner_var2	(string) single variable name to use when generating the innermost (third tier) detail rows, nested under each level of inner_var1.

Details

a_two_tier(): Both the analysis variable and inner_var must be factors. Detail rows are differentiated by having an indent mod of one, causing them to hang indented under their corresponding group row.

a_three_tier(): The analysis variable, inner_var1, and inner_var2 must all be factors. Mid-level rows are indented by one and innermost rows by two, relative to the group summary row.

Value

a_two_tier(): A RowsVerticalSection object including both the group row and all detail rows, if applicable, for the facet.

a_three_tier(): A RowsVerticalSection object including the group row, mid-level detail rows, and innermost detail rows, if applicable, for the facet.

Note

In its current form, this function may not retain any formatting or labeling instructions added by grp_fun or detail_fun, and it will override any .indent_mods values specified by them. This behavior may change in future versions.

Author(s)

GB, WW.

Examples

```
# Example 1

lyt_obs_levels <- basic_table() |>
  split_cols_by("ARM") |>
  split_rows_by("EOSSTT", child_labels = "hidden") |>
  analyze("EOSSTT",
    afun = a_two_tier,
    extra_args = list(
      grp_fun = simple_analysis,
      detail_fun = simple_analysis,
      inner_var = "DCSREAS",
      drill_down_levs = "DISCONTINUED"
    )
  )
```

```

tbl <- build_table(lyt_obs_levels, ex_adsl)
tbl

lyt_all_levels <- basic_table() |>
  split_cols_by("ARM") |>
  split_rows_by("EOSSTT", child_labels = "hidden") |>
  analyze("EOSSTT",
    afun = a_two_tier,
    extra_args = list(
      grp_fun = simple_analysis,
      detail_fun = simple_analysis,
      inner_var = "DCSREAS",
      drill_down_levs = "DISCONTINUED",
      use_all_levels = TRUE
    )
  )

adsl_subset <- subset(ex_adsl, DCSREAS != "ADVERSE EVENT")
levels(adsl_subset$DCSREAS)

tbl_all_levels <- build_table(lyt_all_levels, adsl_subset)
tbl_all_levels

tbl_obs_levels <- build_table(lyt_obs_levels, adsl_subset)
tbl_obs_levels

# Example 2

library(dplyr)

trtvar <- "ARM"
ctrl_grp <- "B: Placebo"

adsl <- ex_adsl |> select(c("USUBJID", "STRATA1", "EOSSTT", "DCSREAS", all_of(trtvar)))
adsl$colspan_trt <- factor(
  ifelse(adsl[[trtvar]] == ctrl_grp, " ", "Active Study Agent"),
  levels = c("Active Study Agent", " ")
)
adsl$rrisk_header <- "Risk Difference (%) (95% CI)"
adsl$rrisk_label <- paste(adsl[[trtvar]], paste("vs", ctrl_grp))

colspan_trt_map <- create_colspan_map(
  df = adsl,
  non_active_grp = ctrl_grp,
  non_active_grp_span_lbl = " ",
  active_grp_span_lbl = "Active Study Agent",
  colspan_var = "colspan_trt",
  trt_var = trtvar
)

a_freq_j_args <- list(
  .stats = "count_unique_fraction",
  denom = "n_altdf",

```

```

    ref_path = c("colspan_trt", " ", trtvar, ctrl_grp)
  )

  two_tier_args <- list(
    grp_fun = a_freq_j,
    detail_fun = a_freq_j,
    inner_var = "DCSREAS",
    drill_down_levs = "DISCONTINUED"
  )

  lyt_rrisk <- basic_table() |>
    split_cols_by("colspan_trt", split_fun = trim_levels_to_map(map = colspan_trt_map)) |>
    split_cols_by(trtvar) |>
    split_cols_by("rrisk_header", nested = FALSE) |>
    split_cols_by(trtvar, labels_var = "rrisk_label", split_fun = remove_split_levels(ctrl_grp)) |>
    split_rows_by("STRATA1") |>
    split_rows_by("EOSSTT", child_labels = "hidden") |>
    analyze("EOSSTT", afun = a_two_tier, extra_args = c(two_tier_args, a_freq_j_args))

  adsl_subset <- subset(
    adsl,
    EOSSTT != "COMPLETED" & (is.na(DCSREAS) | DCSREAS != "PROTOCOL VIOLATION")
  )

  tbl_rrisk <- build_table(lyt_rrisk, adsl_subset, alt_counts_df = adsl_subset)
  tbl_rrisk

```

ac_blank_line

*Analysis and Content Summary Function Producing Blank Line***Description**

Analysis and Content Summary Function Producing Blank Line

Usage

```
ac_blank_line(df, labelstr = "")
```

Arguments

df	(data.frame) data set containing all analysis variables.
labelstr	(character) label of the level of the parent split currently being summarized (must be present as second argument in Content Row Functions). See rtables::summarize_row_groups() for more information.

analyze_values	<i>Shortcut Layout Function for Standard Continuous Variable Analysis</i>
----------------	---

Description

Shortcut Layout Function for Standard Continuous Variable Analysis

Usage

```
analyze_values(lyt, vars, ..., formats)
```

Arguments

lyt	(layout) input layout where analyses will be added to.
vars	(character) variable names for the primary analysis variable to be iterated over.
...	additional arguments for the lower level functions.
formats	(list) formats including mean_sd, median and range specifications.

Value

Modified layout.

bspt_pruner	<i>Pruning Function for pruning based on a fraction and/or a difference from the control arm</i>
-------------	--

Description

This is a pruning constructor function which identifies records to be pruned based on the the fraction from the percentages. In addition to just looking at a fraction within an arm, this function also allows further flexibility to also prune based on a comparison versus the control arm.

Usage

```
bspt_pruner(
  fraction = 0.05,
  keeprowtext = "Analysis set: Safety",
  reg_expr = FALSE,
  control = NULL,
  diff_from_control = NULL,
  only_more_often = TRUE,
  cols = c("TRT01A")
)
```

Arguments

<code>fraction</code>	(proportion) Fraction threshold. Function will keep all records strictly greater than this threshold.
<code>keeprowtext</code>	(character) Row to be excluded from pruning.
<code>reg_expr</code>	(logical) Apply keeprowtext as a regular expression (grepl with fixed = TRUE)
<code>control</code>	(character) Control Group
<code>diff_from_control</code>	(numeric) Difference from control threshold.
<code>only_more_often</code>	(logical) TRUE: Only consider when column pct is more often than control. FALSE: Also select a row where column pct is less often than control and abs(diff) above threshold
<code>cols</code>	(character) Column path.

Value

Function that can be utilized as pruning function in `prune_table`.

Examples

```
ADSL <- data.frame(
  USUBJID = c(
    "XXXXX01", "XXXXX02", "XXXXX03", "XXXXX04", "XXXXX05",
    "XXXXX06", "XXXXX07", "XXXXX08", "XXXXX09", "XXXXX10"
  ),
  TRT01P = c(
    "ARMA", "ARMB", "ARMA", "ARMB", "ARMB",
    "Placebo", "Placebo", "Placebo", "ARMA", "ARMB"
  ),
  FASFL = c("Y", "Y", "Y", "Y", "N", "Y", "Y", "Y", "Y", "Y"),
  SAFFL = c("N", "N", "N", "N", "N", "N", "N", "N", "N", "N"),
  PKFL = c("N", "N", "N", "N", "N", "N", "N", "N", "N", "N")
)

ADSL <- ADSL |>
  dplyr::mutate(TRT01P = as.factor(TRT01P)) |>
  dplyr::mutate(SAFFL = factor(SAFFL, c("Y", "N"))) |>
  dplyr::mutate(PKFL = factor(PKFL, c("Y", "N")))

lyt <- basic_table() |>
  split_cols_by("TRT01P") |>
  add_overall_col("Total") |>
```

```

split_rows_by(
  "FASFL",
  split_fun = drop_and_remove_levels("N"),
  child_labels = "hidden"
) |>
analyze("FASFL",
  var_labels = "Analysis set:",
  afun = a_freq_j,
  show_labels = "visible",
  extra_args = list(label = "Full", .stats = "count_unique_fraction")
) |>
split_rows_by(
  "SAFFL",
  split_fun = remove_split_levels("N"),
  child_labels = "hidden"
) |>
analyze("SAFFL",
  var_labels = "Analysis set:",
  afun = a_freq_j,
  show_labels = "visible",
  extra_args = list(label = "Safety", .stats = "count_unique_fraction")
) |>
split_rows_by(
  "PKFL",
  split_fun = remove_split_levels("N"),
  child_labels = "hidden"
) |>
analyze("PKFL",
  var_labels = "Analysis set:",
  afun = a_freq_j,
  show_labels = "visible",
  extra_args = list(label = "PK", .stats = "count_unique_fraction")
)

result <- build_table(lyt, ADSL)

result

result <- prune_table(
  result,
  prune_func = bspt_pruner(
    fraction = 0.05,
    keeprowtext = "Safety",
    cols = c("Total")
  )
)

result

```

Description

This builds the model formula which is used inside `fit_mmrj()` and provided to `mmrm::mmrm()` internally. It can be instructive to look at the resulting formula directly sometimes. In particular, if a subgroup variable is included in `vars`, then the formula will include the interaction of subgroup with `arm` and `visit`.

Usage

```
build_formula(
  vars,
  cor_struct = c("unstructured", "toeplitz", "heterogeneous toeplitz", "ante-dependence",
    "heterogeneous ante-dependence", "auto-regressive", "heterogeneous auto-regressive",
    "compound symmetry", "heterogeneous compound symmetry")
)
```

Arguments

<code>vars</code>	(list) variables to use in the model.
<code>cor_struct</code>	(string) specify the covariance structure to use.

Value

Formula to use in `mmrm::mmrm()`.

Examples

```
vars <- list(
  response = "AVAL",
  covariates = c("RACE", "SEX"),
  id = "USUBJID",
  arm = "ARMCD",
  visit = "AVISIT",
  subgroup = "REGION"
)
build_formula(vars, "auto-regressive")
build_formula(vars)
```

`c_proportion_logical` *c_function for proportion of TRUE in logical vector*

Description

A simple statistics function which prepares the numbers with percentages in the required format, for use in a split content row. The denominator here is from the column N. Note that we don't use here `.alt_df` because that might not have required row split variables available.

Usage

```
c_proportion_logical(x, labelstr, label_fstr, format, .N_col)
```

Arguments

x	(logical) binary variable we want to analyze.
labelstr	(string) label string.
label_fstr	(string) format string for the label.
format	(character or list) format for the statistics.
.N_col	(numeric) number of columns.

Value

The `rtables::in_rows()` result with the proportion statistics.

See Also

`s_proportion_logical()` for the related statistics function.

categorize_pval	<i>Categorize p-values</i>
-----------------	----------------------------

Description

Assigns each p-value to a category defined by a named list of lower and upper bounds.

Usage

```
categorize_pval(p, pvalcat)
```

Arguments

p	(numeric) A vector of p-values to categorize. Missing values are permitted.
pvalcat	(named list) A non-empty named list of p-value category bounds. See <code>h_normalize_pvalcat()</code> for the required structure.

Details

Categories are evaluated in the order supplied by pvalcat. Bounds are lower-inclusive and upper-exclusive, except that the upper bound of the final category is inclusive. P-values that do not fall in a category, including missing values, are returned as NA_character_.

Value

A character vector of category labels, with one element per value in p.

Examples

```
pvalcat <- list(
  "<0.001" = c(0, 0.001),
  "0.001 to <0.05" = c(0.001, 0.05),
  ">=0.05" = c(0.05, 1)
)

categorize_pval(c(0, 0.001, 0.049, 0.05, 1, NA), pvalcat)
```

check_wrap_nobreak	<i>Check Word Wrapping</i>
--------------------	----------------------------

Description

Check a set of column widths for word-breaking wrap behavior.

Usage

```
check_wrap_nobreak(tt, colwidths, fontspec)
```

Arguments

tt	(TableTree) TableTree object
colwidths	(numeric) Column widths (in numbers of spaces under fontspec)
fontspec	(font_spec) Font specification object

Value

TRUE if the wrap is able to be done without breaking words, FALSE if wordbreaking is required to apply colwidths.

cmhrms*Cochran-Mantel-Haenszel Row Mean Scores test*

Description

See https://psiaims.github.io/CAMIS/Comp/r-sas_cmh.html for a general comparison overview between R and SAS.

Usage

```
a_cmhrms_j(  
  df,  
  .var,  
  ref_path,  
  .spl_context,  
  .ref_group,  
  .in_ref_col,  
  .df_row,  
  ...,  
  variables,  
  collapse_combo = TRUE,  
  .stats = NULL,  
  .formats = NULL,  
  .indent_mods = NULL,  
  .labels = NULL  
)  
  
s_cmhrms_j(  
  df,  
  .var,  
  .ref_group,  
  .in_ref_col,  
  ...,  
  .df_row,  
  variables,  
  collapse_combo = FALSE  
)  
  
a_cmhrms_j_with_exclude(  
  df,  
  exclude_levels,  
  .var,  
  .spl_context,  
  .ref_group,  
  .in_ref_col,  
  .df_row,  
  ...,
```

```

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

```

Arguments

<code>df</code>	(data.frame) data set containing all analysis variables.
<code>.var</code>	(string) single variable name that is passed by <code>rtables</code> when requested by a statistics function.
<code>ref_path</code>	(character) global reference group specification, see get_ref_info() .
<code>.spl_context</code>	(data.frame) gives information about ancestor split states that is passed by <code>rtables</code> .
<code>.ref_group</code>	(data.frame or vector) the data corresponding to the reference group.
<code>.in_ref_col</code>	(logical) TRUE when working with the reference level, FALSE otherwise.
<code>.df_row</code>	(data.frame) data frame across all of the columns for the given row split.
<code>...</code>	additional arguments for the lower level functions.
<code>variables</code>	(list) list with arm and strata variable names.
<code>collapse_combo</code>	(logical) If TRUE, multiple arm levels from <code>df</code> will be combined into 1 level.
<code>.stats</code>	(character) statistics to select for the table.
<code>.formats</code>	(named character or list) formats for the statistics. See Details in <code>analyze_vars</code> for more information on the 'auto' setting.
<code>.indent_mods</code>	(named integer) indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.
<code>.labels</code>	(named character) labels for the statistics (without indent).
<code>exclude_levels</code>	(list) A named list where names correspond to split variables and values are vectors of levels to exclude.

Value

- `s_cmhrms_j` a single element list containing the p-value from row mean score test.
- `a_cmhrms_j` a `VerticalRowsSection` object (single row).

Functions

- `a_cmhrms_j()`: Formatted analysis function which is used as `afun`.
- `s_cmhrms_j()`: Statistics function for the calculation of the p-value based upon the row mean scores test.
- `a_cmhrms_j_with_exclude()`: Wrapper for the `afun` which can exclude row split levels from producing the analysis. These have to be specified in the `exclude_levels` argument, see `?do_exclude_split` for details.

cmp_cfun

Summary Analysis Function for Compliance Columns

Description

A simple statistics function which prepares the numbers with percentages in the required format, for use in a split content row. The denominator here is from the expected visits column.

Usage

```
cmp_cfun(df, labelstr, .spl_context, variables, formats)
```

Arguments

<code>df</code>	(<code>data.frame</code>) data set containing all analysis variables.
<code>labelstr</code>	(<code>character</code>) label of the level of the parent split currently being summarized (must be present as second argument in Content Row Functions). See <code>rtables::summarize_row_groups()</code> for more information.
<code>.spl_context</code>	(<code>data.frame</code>) gives information about ancestor split states that is passed by <code>rtables</code> .
<code>variables</code>	(<code>list</code>) with variable names of logical columns for expected, received and missing visits.
<code>formats</code>	(<code>list</code>) with the <code>count_percent</code> format to use for the received and missing visits columns.

Value

The `rtables::in_rows()` result with the counts and proportion statistics.

See Also

`cmp_post_fun()` for the corresponding split function.

`cmp_post_fun`*Split Function for Compliance Columns*

Description

Here we just split into 3 columns for expected, received and missing visits.

Usage

```
cmp_post_fun(ret, spl, fulldf, .spl_context)
```

```
cmp_split_fun(df, spl, vals = NULL, labels = NULL, trim = FALSE, .spl_context)
```

Arguments

<code>ret</code>	(list) result from previous split function steps.
<code>spl</code>	(split) split object.
<code>fulldf</code>	(data.frame) full data frame.
<code>.spl_context</code>	(data.frame) gives information about ancestor split states that is passed by <code>rtables</code> .
<code>df</code>	(data.frame) data set containing all analysis variables.
<code>vals</code>	(character) values to use for the split.
<code>labels</code>	(named character) labels for the statistics (without indent).
<code>trim</code>	(logical) whether to trim the values.

Value

a split function for use with [rtables::split_rows_by](#) when creating proportion-based tables with compliance columns.

See Also

[rtables::make_split_fun\(\)](#) describing the requirements for this kind of post-processing function.

column_stats

*Statistics within the column space***Description**

A function used for obtaining statistics within the columns of your table. Used in change from baseline tables. This takes the visit names as its row labels.

Usage

```
column_stats(
  df,
  .var,
  .spl_context,
  .formats = NULL,
  formats_var = NULL,
  exclude_visits = c("Baseline (DB)"),
  var_names = c("AVAL", "CHG", "BASE")
)
```

Arguments

df	(data.frame or tibble) dataset.
.var	(string) single variable name that is passed by rtables
.spl_context	(data.frame) gives information about ancestor split states.
.formats	(named character or list) formats for the statistics. When set to "default" formats_var argument will be used for format setup.
formats_var	(string or NULL) NULL (the default) or the name of the list column containing named lists of default formats to use. These will not override .formats (other than "default").
exclude_visits	Visits to exclude for the second/third part of the columns ("BASE", "CHG").
var_names	(character vector) Vector of variable names to use instead of the default AVAL, CHG, BASE. By default, the function expects these specific variable names in your data, but you can customize them to match your dataset's column names.

Value

An analysis function (for use with [rtables::analyze](#)) implementing the specified statistics. Typically used in a layout that has been setup by [stats_in_cols_setup\(\)](#)

Examples

```

# example code
library(dplyr)
advs <- ex_advs |>
  filter(AVISIT %in% toupper(c("Screening", "Baseline", "Week 1 Day 8", "Week 2 Day 15"))) |>
  mutate(AVISIT = droplevels(AVISIT)) |>
  filter(PARAMCD %in% c("DIABP", "SYSBP"))
advs_fmt <- tribble(
  ~PARAMCD, ~fmt_d,
  "DIABP", list(
    mean = "xx.xx", sd = "xx.xxx", se = "xx.xxx", median = "xx.xx",
    min = "xx.x", max = "xx.x", mean_sd = "xx.xx (xx.xxx)"
  ),
  "SYSBP", list(
    mean = "xx.x", sd = "xx.xx", se = "xx.xx", median = "xx.x",
    min = "xx.", max = "xx.", mean_sd = "xx.x (xx.xx)"
  )
)

advs <- left_join(advs, advs_fmt)

mysplitfun <- make_split_fun(
  post = list(stats_in_cols_setup())
)

lyt <- basic_table() |>
  split_cols_by_multivar(
    c("AVAL", "BASE", "CHG"),
    varlabels = c("Visit Values", " ", "Change from Baseline")
  ) |>
  split_cols_by("STUDYID", split_fun = mysplitfun) |>
  split_rows_by("PARAM", split_fun = drop_split_levels) |>
  split_rows_by("ARMCD", labels_var = "ARM") |>
  analyze("AVISIT", afun = column_stats, extra_args = list(
    exclude_visits = toupper(c("Baseline", "Screening")),
    .formats = "default",
    formats_var = "fmt_d"
  ))
result <- build_table(lyt, advs, round_type = "sas")
head(result, 20)

```

Description

Conditional Removal of Facets

Usage

```
cond_rm_facets(
  facets = NULL,
  facets_regex = NULL,
  ancestor_pos = 1,
  split = NULL,
  split_regex = NULL,
  value = NULL,
  value_regex = NULL,
  keep_matches = FALSE
)
```

Arguments

facets	(character or NULL) Vector of facet names to be removed if condition(s) are met
facets_regex	(character) Regular expression to identify facet names to be removed if condition(s) are met.
ancestor_pos	(numeric) Row in <code>spl_context</code> to check the condition within. E.g., 1 represents the first split, 2 represents the second split nested within the first, etc. NA specifies that the conditions should be checked at all split levels. Negative integers indicate position counting back from the current one, e.g., -1 indicates the direct parent (most recent split before this one). Negative and positive/NA positions cannot be mixed.
split	(character or NULL) If specified, name of the split at position <code>ancestor_pos</code> must be identical to this value for the removal condition to be met.
split_regex	(character or NULL) If specified, a regular expression the name of the split at position <code>ancestor_pos</code> must match for the removal condition to be met. Cannot be specified at the same time as <code>split</code> .
value	(character or NULL) If specified, split (facet) value at position <code>ancestor_pos</code> must be identical to this value for removal condition to be met.
value_regex	(character or NULL) If specified, a regular expression the value of the split at position <code>ancestor_pos</code> must match for the removal condition to be met. Cannot be specified at the same time as <code>value</code> .
keep_matches	(logical) Given the specified condition is met, should the facets removed be those matching <code>facets/facets_regex</code> (FALSE, the default), or those <i>not</i> matching (TRUE).

Details

Facet removal occurs when the specified condition(s) on the split(s) and or value(s) are met within at least one of the `split_context` rows indicated by `ancestor_pos`; otherwise the set of facets is

returned unchanged.

If facet removal is performed, either *all* facets which match `facets` (or `facets_regex` will be removed (the default `keep_matches == FALSE` case), or all *non-matching* facets will be removed (when `keep_matches_only == TRUE`).

Value

A function suitable for use in `make_split_fun`'s `post` argument which encodes the specified condition.

Note

A degenerate table is likely to be returned if all facets are removed.

Examples

```
rm_a_from_placebo <- cond_rm_facets(
  facets = "A",
  ancestor_pos = NA,
  value_regex = "Placeb",
  split = "ARM"
)
mysplit <- make_split_fun(post = list(rm_a_from_placebo))

lyt <- basic_table() |>
  split_cols_by("ARM") |>
  split_cols_by("STRATA1", split_fun = mysplit) |>
  analyze("AGE", mean, format = "xx.x")
build_table(lyt, ex_adsl)

rm_bc_from_combo <- cond_rm_facets(
  facets = c("B", "C"),
  ancestor_pos = -1,
  value_regex = "Combi"
)
mysplit2 <- make_split_fun(post = list(rm_bc_from_combo))

lyt2 <- basic_table() |>
  split_cols_by("ARM") |>
  split_cols_by("STRATA1", split_fun = mysplit2) |>
  analyze("AGE", mean, format = "xx.x")
tbl2 <- build_table(lyt2, ex_adsl)
tbl2

rm_bc_from_combo2 <- cond_rm_facets(
  facets_regex = "^A$",
  ancestor_pos = -1,
  value_regex = "Combi",
  keep_matches = TRUE
)
mysplit3 <- make_split_fun(post = list(rm_bc_from_combo2))
```

```

lyt3 <- basic_table() |>
  split_cols_by("ARM") |>
  split_cols_by("STRATA1", split_fun = mysplit3) |>
  analyze("AGE", mean, format = "xx.x")
tbl3 <- build_table(lyt3, ex_ads1)

stopifnot(identical(cell_values(tbl2), cell_values(tbl3)))

```

count and fraction related formatting functions

Formatting functions for count and fraction, and for count denominator and fraction values

Description

Formats a count together with fraction (and/or denominator) with special consideration when count is 0, or fraction is 1.

See also: [tern::format_count_fraction_fixed_dp\(\)](#)

Usage

```

jjcsformat_cnt_den_fract_fct(
  d = 1,
  type = c("count_fraction", "count_denom_fraction", "fraction_count_denom"),
  verbose = FALSE,
  legacy = FALSE
)

jjcsformat_count_fraction(x, round_type = valid_round_type, output, ...)

jjcsformat_count_denom_fraction(x, round_type = valid_round_type, output, ...)

jjcsformat_count_denom_fraction_legacy(
  x,
  round_type = valid_round_type,
  output,
  ...
)

jjcsformat_fraction_count_denom(x, round_type = valid_round_type, output, ...)

jjcsformat_fraction_count_denom_legacy(
  x,
  round_type = valid_round_type,
  output,
  ...
)

```

Arguments

d	(numeric(1)) Number of digits to round fraction to (default = 1)
type	(character(1)) One of count_fraction, count_denom_fraction, fraction_count_denom, to specify the type of format the function will represent.
verbose	(logical) Whether to print verbose output
legacy	(logical(1)) To allow for backward compatibility behavior
x	(numeric vector) Vector with elements num and fraction or num, denom and fraction.
round_type	(character(1)) the type of rounding to perform. See formatters::format_value() for more details.
output	(string) output type. See formatters::format_value() for more details.
...	Additional arguments passed to other methods.

Value

A formatting function to format input into string in the format count / denom (ratio percent).
If count is 0, the format is 0. If fraction is >0.99, the format is count / denom (>99.9 percent)

See Also

Other JJCS formatting functions: [jjcsformat_fraction\(\)](#), [jjcsformat_xx\(\)](#)

Examples

```
jjcsformat_count_fraction(c(7, 0.7))
jjcsformat_count_fraction(c(70000, 70000 / 70001))
jjcsformat_count_fraction(c(235, 235 / 235))
fmt <- jjcsformat_cnt_den_fract_fct(type = "count_fraction", d = 2)
fmt(c(23, 23 / 235))

jjcsformat_count_denom_fraction(c(7, 10, 0.7))
jjcsformat_count_denom_fraction(c(70000, 70001, 70000 / 70001))
jjcsformat_count_denom_fraction(c(235, 235, 235 / 235))
fmt <- jjcsformat_cnt_den_fract_fct(type = "count_denom_fraction", d = 2)
fmt(c(23, 235, 23 / 235))
jjcsformat_count_denom_fraction_legacy(c(0, 235, 0 / 235))

jjcsformat_fraction_count_denom(c(7, 10, 0.7))
jjcsformat_fraction_count_denom(c(70000, 70001, 70000 / 70001))
jjcsformat_fraction_count_denom(c(235, 235, 235 / 235))
fmt <- jjcsformat_cnt_den_fract_fct(type = "fraction_count_denom", d = 2)
fmt(c(23, 235, 23 / 235))
jjcsformat_fraction_count_denom_legacy(c(0, 235, 0 / 235))
```

count_pruner	<i>Count Pruner</i>
--------------	---------------------

Description

This is a pruning constructor function which identifies records to be pruned based on the count (assumed to be the first statistic displayed when a compound statistic (e.g., ## / ## (XX.X percent) is presented).

Usage

```
count_pruner(  
  count = 0,  
  cat_include = NULL,  
  cat_exclude = NULL,  
  cols = c("TRT01A")  
)
```

Arguments

count	(numeric) count threshold. Function will keep all records strictly greater than this threshold.
cat_include	(character) Category to be considered for pruning
cat_exclude	(character) Category to be excluded from pruning
cols	(character) column path (character or integer (column indices))

Value

Function that can be utilized as pruning function in prune_table.

Examples

```
ADSL <- data.frame(  
  USUBJID = c(  
    "XXXXX01", "XXXXX02", "XXXXX03", "XXXXX04", "XXXXX05",  
    "XXXXX06", "XXXXX07", "XXXXX08", "XXXXX09", "XXXXX10"  
  ),  
  TRT01P = factor(  
    c(  
      "ARMA", "ARMB", "ARMA", "ARMB", "ARMB",  
      "Placebo", "Placebo", "Placebo", "ARMA", "ARMB"  
    )  
  ),  
)
```

```

FASFL = c("Y", "Y", "Y", "Y", "N", "Y", "Y", "Y", "Y", "Y"),
SAFFL = c("N", "N", "N", "N", "N", "N", "N", "N", "N", "N"),
PKFL = c("N", "N", "N", "N", "N", "N", "N", "N", "N", "N")
)

lyt <- basic_table() |>
  split_cols_by("TRT01P") |>
  add_overall_col("Total") |>
  analyze("FASFL",
    var_labels = "Analysis set:",
    afun = a_freq_j,
    extra_args = list(label = "Full", val = "Y"),
    show_labels = "visible"
  ) |>
  analyze("SAFFL",
    var_labels = "Analysis set:",
    afun = a_freq_j,
    extra_args = list(label = "Safety", val = "Y"),
    show_labels = "visible"
  ) |>
  analyze("PKFL",
    var_labels = "Analysis set:",
    afun = a_freq_j,
    extra_args = list(label = "PK", val = "Y"),
    show_labels = "visible"
  )

result <- build_table(lyt, ADSL)

result

result <- prune_table(
  result,
  prune_func = count_pruner(cat_exclude = c("Safety"), cols = "Total")
)

result

```

create_colspan_map

*Creation of Column Spanning Mapping Dataframe***Description**

A function used for creating a data frame containing the map that is compatible with the rtables split function `rtables::trim_levels_to_map()`.

Usage

```

create_colspan_map(
  df,

```

```

non_active_grp = c("Placebo"),
non_active_grp_span_lbl = " ",
active_grp_span_lbl = "Active Study Agent",
colspan_var = "colspan_trt",
trt_var = "TRT01A",
active_first = TRUE
)

```

Arguments

<code>df</code>	(data.frame) The name of the data frame in which the spanning variable is to be appended to
<code>non_active_grp</code>	(character) The value(s) of the treatments that represent the non-active or comparator treatment groups default value = <code>c('Placebo')</code>
<code>non_active_grp_span_lbl</code>	(character) The assigned value of the spanning variable for the non-active or comparator treatment groups default value = <code>" "</code>
<code>active_grp_span_lbl</code>	(character) The assigned value of the spanning variable for the active treatment group(s) default value = <code>'Active Study Agent'</code>
<code>colspan_var</code>	(character) The desired name of the newly created spanning variable default value = <code>'colspan_trt'</code>
<code>trt_var</code>	(character) The name of the treatment variable that is used to determine which spanning treatment group value to apply. default value = <code>'TRT01A'</code>
<code>active_first</code>	(logical) whether the active columns come first.

Details

This function creates a data frame containing the map that is compatible with the `rtables` split function `rtables::trim_levels_to_map()`. The levels of the specified `trt_var` variable will be stored within the `trt_var` variable and the `colspan_var` variable will contain the corresponding spanning header value for each treatment group.

Value

A data frame that contains the map to be used with the `rtables` split function `rtables::trim_levels_to_map()`.

See Also

`rtables::add_combo_levels()` for creating split functions that utilize combination levels.

Examples

```
library(tibble)

df <- tribble(
  ~TRT01A,
  "Placebo",
  "Active 1",
  "Active 2"
)

df$TRT01A <- factor(df$TRT01A, levels = c("Placebo", "Active 1", "Active 2"))

colspan_map <- create_colspan_map(
  df = df,
  non_active_grp = c("Placebo"),
  non_active_grp_span_lbl = " ",
  active_grp_span_lbl = "Active Study Agent",
  colspan_var = "colspan_trt",
  trt_var = "TRT01A"
)

colspan_map
```

create_colspan_var	<i>Creation of Column Spanning Variables</i>
--------------------	--

Description

A function used for creating a spanning variable for treatment groups.

Usage

```
create_colspan_var(
  df,
  non_active_grp = c("Placebo"),
  non_active_grp_span_lbl = " ",
  active_grp_span_lbl = "Active Study Agent",
  colspan_var = "colspan_trt",
  trt_var = "TRT01A"
)
```

Arguments

df	(data.frame) The name of the data frame in which the spanning variable is to be appended to
non_active_grp	(character) The value(s) of the treatments that represent the non-active or comparator treatment groups default value = c('Placebo')

```

non_active_grp_span_lbl
    (character)
    The assigned value of the spanning variable for the non-active or comparator
    treatment groups default value = ""

active_grp_span_lbl
    (character)
    The assigned value of the spanning variable for the active treatment group(s)
    default value = 'Active Study Agent'

colspan_var    (character)
    The desired name of the newly created spanning variable default value = 'colspan_trt'

trt_var        (character)
    The name of the treatment variable that is used to determine which spanning
    treatment group value to apply. default value = 'TRT01A'

```

Details

This function creates a spanning variable for treatment groups that is intended to be used within the column space.

Value

A data frame that contains the new variable as specified in `colspan_var`.

Examples

```

library(tibble)

df <- tribble(
  ~TRT01A,
  "Placebo",
  "Active 1",
  "Active 2"
)

df$TRT01A <- factor(df$TRT01A, levels = c("Placebo", "Active 1", "Active 2"))

colspan_var <- create_colspan_var(
  df = df,
  non_active_grp = c("Placebo"),
  non_active_grp_span_lbl = " ",
  active_grp_span_lbl = "Active Treatment",
  colspan_var = "colspan_trt",
  trt_var = "TRT01A"
)

colspan_var

```

`cur_col_split_path_utils`*Utilities for the Current Column Split Path*

Description

[Stable]

These helper functions are intended for use in [rtables](#) custom analysis functions that depend on the current column split context.

Usage

```
cur_col_split_path(.spl_context)
```

```
in_column(col_path, .spl_context)
```

Arguments

- | | |
|---------------------------|---|
| <code>.spl_context</code> | (data.frame)
gives information about ancestor split states that is passed by <code>rtables</code> . |
| <code>col_path</code> | (character or NULL)
Column path specified as alternating split variable names and split values. Must have at least two elements and an even length, e.g. <code>c("ARM", "Placebo")</code> or <code>c("ARM", "*", "desc_stat", "N")</code> ; see rtables::col_paths() .
The special value <code>"*"</code> can be used as a wildcard to match any split variable name or value.
NULL can be used to indicate that no path is specified, in which case the function returns FALSE, regardless of <code>.spl_context</code> . |

Value

- `cur_col_split_path()` returns a character vector containing the current column split path extracted from `.spl_context`, interleaved as `c(var1, val1, var2, val2, ...)`.
- `in_column()` returns a single logical value indicating whether the current column split matches the specified `col_path`.

Functions

- `cur_col_split_path()`: Get the current column split path.
- `in_column()`: Determine whether a given column path matches the current column split path.

Author(s)

WW

See Also

- `rtables::col_paths()` for obtaining column paths from a built table.

Examples

```
.spl_context_1 <- data.frame(
  cur_col_split = I(list(c("ARM"))),
  cur_col_split_val = I(list(c("Placebo")))
)

.spl_context_2 <- data.frame(
  cur_col_split = I(list(c("ARM", "desc_stat"))),
  cur_col_split_val = I(list(c("Placebo", "N")))
)

cur_col_split_path(.spl_context_1)
cur_col_split_path(.spl_context_2)

in_column(c("ARM", "Placebo"), .spl_context_1)
in_column(c("ARM", "*"), .spl_context_1)
in_column(c("ARM", "X"), .spl_context_1)

in_column(c("ARM", "*", "desc_stat", "N"), .spl_context_2)
in_column(c("ARM", "*", "desc_stat", "Mean"), .spl_context_2)

# Use in an rtables custom analysis function.

data <- formatters::ex_adsl
data$desc_stat <- "phantom"

# analysis function
a_N_mean <- function(x, .spl_context) {
  if (in_column(c("ARM", "*", "desc_stat", "N"), .spl_context)) {
    rcell(length(x))
  } else if (in_column(c("ARM", "*", "desc_stat", "Mean"), .spl_context)) {
    rcell(mean(x), format = "xx.x")
  } else {
    rcell(NULL)
  }
}

combodf <- data.frame(
  valname = c("N", "Mean"),
  label = c("N", "Mean"),
  levelcombo = I(list(select_all_levels, select_all_levels)),
  exargs = I(list(list(), list()))
)

lyt <- basic_table() |>
  split_cols_by("ARM") |>
  split_cols_by(
    "desc_stat",
```

```

    split_fun = add_combo_levels(combodf, keep_levels = c("N", "Mean"))
  ) |>
  analyze("AGE", afun = a_N_mean)

tbl <- build_table(lyt, data)
tbl

```

diff_means

Difference in Means

Description

[Stable]

Computes estimates and associated inferential statistics for the estimand defined as the difference in population means between two distributions.

The following quantities are available:

diff_means_n1 Sample size of Group 1.

diff_means_n2 Sample size of Group 2.

diff_means_est Point estimate of the difference in population means.

diff_means_se Standard error of the estimator of the difference in population means.

diff_means_est_se The point estimate and the standard error.

diff_means_ci Confidence interval for the difference in population means.

diff_means_est_ci The point estimate and the confidence interval.

Usage

```

s_diff_means(df1, df2, .var, paired = FALSE, paired_by = NULL, ...)

a_diff_means(
  df,
  .var,
  .spl_context,
  ref_path = NULL,
  ...,
  .stats = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)

```

Arguments

df1	(data.frame) Dataset for the first sample.
df2	(data.frame) Dataset for the second sample.
.var	(character(1)) Name of the column in the dataset(s) containing observed sample values.
paired	(logical(1)) Whether the samples are paired.
paired_by	(character or NULL) Column name(s) in df1 and df2 used to match observations between datasets. Required when paired = TRUE and must uniquely identify each pair in both datasets.
...	Additional named arguments passed to s_diff_means() and safe_t_test().
df	(data.frame) Dataset for the first sample.
.spl_context	(data.frame) gives information about ancestor split states that is passed by rtables.
ref_path	(character) Global reference group specification; see get_ref_info() . This specifies the column path for the second sample. Specifically, the second sample is .ref_group[[.var]], where .ref_group is computed by get_ref_info() using .spl_context and ref_path. This is a workaround for an rtables limitation in supporting .ref_group / .in_ref_col in the presence of stacked headers. Normally, .ref_group and .in_ref_col are provided by the rtables tabulation framework when the current column split is within the reference variable.
.stats	(character) Statistics to include in the table. Available options are: 'diff_means_n1', 'diff_means_n2', 'diff_means_est', 'diff_means_se'. If NULL, all statistics are included.
.formats	(named character or list) formats for the statistics. See Details in analyze_vars for more information on the 'auto' setting.
.labels	(named character) labels for the statistics (without indent).
.indent_mods	(named integer) indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.

Details

Supports both non-paired and paired samples. For non-paired samples, the estimator is defined as the difference between sample means. For paired samples, the estimator is defined as the mean of within-pair differences.

Inferential statistics, including standard errors and confidence intervals, are obtained using a t-distribution framework via the internal `safe_t_test()` function.

It supports only numeric variables. Only complete (non-missing) observations are used in statistical computations. Non-finite values (i.e., `Inf`, `-Inf`) are not allowed.

Value

- `s_diff_means()` returns a named list containing the quantities described in the Description section.
- `a_diff_means()` returns formatted `rtables::in_rows()` result with the quantities requested in `.stats`.

Functions

- `s_diff_means()`: Statistical function that computes estimates of the difference in means. The first sample is taken from `df1[[.var]]` and the second from `df2[[.var]]`.
If `paired = FALSE`, missing values are removed separately from each sample prior to statistical computations. If `paired = TRUE`, observations are matched using `paired_by` prior to statistical analysis, and only complete pairs are used.
Data extraction, alignment, and missing-value handling are delegated to the internal `extract_vectors()` function, which prepares cleaned numeric vectors for both paired and unpaired settings.
Inferential statistics are then computed using `safe_t_test()` applied to the processed vectors.
- `a_diff_means()`: Formatted analysis function which is used as a fun in `rtables::analyze()`. The sample from the first distribution is `df[[.var]]`, and the sample from the second distribution is `.ref_group[[.var]]`, where `.ref_group` is computed by `get_ref_info()` using `.spl_context` and `ref_path`.
If `ref_path = NULL`, then `a_diff_means()` returns an `rtables::in_rows()` result with an empty `formatted_cell` for every statistic specified in `.stats`.

Author(s)

WW

See Also

`safe_t_test()`, `extract_vectors()`

Examples

```
## Statistics function:

df1 <- data.frame(id = c("A", "B", "C", "D"), value = 1:4)
df2 <- data.frame(id = c("A", "C", "D", "E", "F"), value = c(3:1, NA, 16))

# Unpaired samples.
s_diff_means(df1, df2, "value", conf.level = .8)

# Paired samples.
```

```

s_diff_means(df1, df2, "value", paired = TRUE, paired_by = "id")

## Analysis function:

lyt <- basic_table() |>
  split_cols_by("ARM") |>
  split_rows_by(
    "PARAMCD",
    labels_var = "PARAM",
    split_fun = keep_split_levels(c("SYSBP", "DIABP"))
  ) |>
  split_rows_by(
    "AVISIT",
    split_fun = keep_split_levels(c("BASELINE", "WEEK 1 DAY 8"))
  ) |>
  analyze(
    "AVAL",
    afun = tern::a_summary,
    extra_args = list(.stats = c("n", "mean")),
    show_labels = "hidden"
  ) |>
  analyze(
    "AVAL",
    afun = a_diff_means,
    table_names = "aval_diff",
    extra_args = list(
      .stats = "diff_means_est_ci",
      ref_path = c("ARM", "B: Placebo")
    ),
    show_labels = "hidden"
  )

tbl <- build_table(lyt, formatters::ex_advs)
tbl

```

do_exclude_split

Predicate to Check if Split Should be Excluded

Description

Predicate to Check if Split Should be Excluded

Usage

```
do_exclude_split(exclude_levels, .spl_context)
```

Arguments

- `exclude_levels` (list)
A named list where names correspond to split variables and values are vectors of levels to exclude.
- `.spl_context` (data.frame)
gives information about ancestor split states that is passed by `rtables`.

Value

TRUE if the current split context matches any of the exclude levels, FALSE otherwise.

Examples

```
do_exclude_split(
  exclude_levels = list(AVISIT = "Baseline"),
  .spl_context = data.frame(
    split = c("AVISIT", "ARM"),
    value = c("Week 4", "Placebo")
  )
)
do_exclude_split(
  exclude_levels = list(AVISIT = "Baseline"),
  .spl_context = data.frame(
    split = c("AVISIT", "ARM"),
    value = c("Baseline", "Placebo")
  )
)
```

event_free

Workaround statistics function to time point survival estimate with CI

Description

This is a workaround for `tern::s_surv_timepoint()`, which adds a statistic containing the time point specific survival estimate together with the confidence interval.

Usage

```
a_event_free(
  df,
  .var,
  ...,
  .stats = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```

```
s_event_free(
  df,
  .var,
  time_point,
  time_unit,
  is_event,
  percent = FALSE,
  control = control_surv_timepoint()
)
```

Arguments

<code>df</code>	(data.frame) data set containing all analysis variables.
<code>.var</code>	(string) single variable name that is passed by <code>rtables</code> when requested by a statistics function.
<code>...</code>	additional arguments for the lower level functions.
<code>.stats</code>	(character) statistics to select for the table.
<code>.formats</code>	(named character or list) formats for the statistics. See Details in <code>analyze_vars</code> for more information on the 'auto' setting.
<code>.labels</code>	(named character) labels for the statistics (without indent).
<code>.indent_mods</code>	(named integer) indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.
<code>time_point</code>	(numeric) time point at which to estimate survival.
<code>time_unit</code>	(string) unit of time for the time point.
<code>is_event</code>	(character) variable name storing Logical values: TRUE if event, FALSE if time to event is censored.
<code>percent</code>	(flag) whether to return in percent or not.
<code>control</code>	(list) relevant list of control options.

Value

- `s_event_free` returns a list as returned by the `tern::s_surv_timepoint()` with an additional three-dimensional statistic `event_free_ci` which combines the `event_free_rate` and `rate_ci` statistics.
- `a_event_free` is analogous to `tern::a_surv_timepoint` but with the additional three-dimensional statistic described above available via `.stats`.

Functions

- `a_event_free()`: Formatted analysis function which is used as `afun`.
- `s_event_free()`: Statistics function which works like `tern::s_surv_timepoint()`, the difference is that it returns the additional statistic `event_free_ci`.

Examples

```
adtte_f <- tern::tern_ex_adtte |>
  dplyr::filter(PARAMCD == "OS") |>
  dplyr::mutate(
    AVAL = tern::day2month(AVAL),
    is_event = CNSR == 0
  )

basic_table() |>
  split_cols_by(var = "ARMCD") |>
  analyze(
    vars = "AVAL",
    afun = a_event_free,
    show_labels = "hidden",
    na_str = tern::default_na_str(),
    extra_args = list(
      time_unit = "week",
      time_point = 3,
      is_event = "is_event"
    )
  ) |>
  build_table(df = adtte_f)
adtte_f <- tern::tern_ex_adtte |>
  dplyr::filter(PARAMCD == "OS") |>
  dplyr::mutate(
    AVAL = tern::day2month(AVAL),
    is_event = CNSR == 0
  )

s_event_free(
  df = adtte_f,
  .var = "AVAL",
  time_point = 6,
  is_event = "is_event",
  time_unit = "month"
)
```

Description

This function is based on `rtables.officer::export_as_docx()`.

Usage

```

export_as_docx_j(
  tt,
  tblid = NULL,
  output_dir = NULL,
  theme = theme_docx_default_j(font = "Times New Roman", font_size = 9L, bold = NULL),
  add_page_break = FALSE,
  titles_as_header = TRUE,
  integrate_footers = TRUE,
  section_properties = officer::prop_section(page_size = officer::page_size(width = 11,
    height = 8.5, orient = orientation), page_margins = officer::page_mar(bottom = 1, top
    = 1, right = 1, left = 1, gutter = 0, footer = 1, header = 1)),
  doc_metadata = NULL,
  template_file = NULL,
  orientation = "portrait",
  paginate = tlgtype == "Table",
  nosplitin = list(row = character(), col = character()),
  string_map = default_str_map,
  markup_df_docx = dps_markup_df_docx,
  combined_docx = FALSE,
  tlgtype = tlg_type(tt),
  col_gap = ifelse(tlgtype == "Listing", 0.5, 3),
  pagenum = ifelse(tlgtype == "Listing", TRUE, FALSE),
  round_type = formatters::obj_round_type(tt),
  alignments = list(),
  border = flextable::fp_border_default(width = 0.875, color = "black"),
  border_mat = make_header_bordmat(obj = tt),
  watermark = NULL,
  export_csv = FALSE,
  output_csv_directory = NULL,
  markup_df = dps_markup_df,
  validate = TRUE,
  ...
)

```

Arguments

<code>tt</code>	(TableTree or listing_df) the object to export.
<code>tblid</code>	(character) output ID to be displayed in the title and last line of footer. When exporting, it will also be used as the output filename. If NULL, a temp file will be created, its dirname will replace argument <code>output_dir</code> , and its basename will replace argument <code>tblid</code> . (optional) Default = NULL.
<code>output_dir</code>	(character) a directory path to save the docx. If NULL, a temp file will be created, its dirname will replace argument <code>output_dir</code> ,

	and its basename will replace argument tblid. (optional) Default = NULL.
theme	(function factory) the theme to apply to the flextable. (optional) Default = <code>theme_docx_default_j()</code> . See <code>theme_docx_default_j()</code> or <code>rtables.officer::theme_docx_default()</code> for more details.
add_page_break	(logical) (optional) Default = FALSE.
titles_as_header	(logical) (optional) Default = TRUE.
integrate_footers	(logical) (optional) Default = TRUE.
section_properties	(prop_section) (optional) A "prop_section" object containing information about page size, orientation, margins, etc. See <code>officer::prop_section()</code> for more details. No need to be specified by end user.
doc_metadata	(list of string) any value that can be used as metadata by <code>officer::set_doc_properties()</code> . Important text values are title, subject, creator, and description, while created is a date object. (optional) Default = NULL.
template_file	(character) Template file that officer will use as a starting point for the final document. Document attaches the table and uses the defaults defined in the template file. Paragraph styles are inherited from this file. If NULL, this function will use an internal template. (optional) Default = NULL.
orientation	(character) one of: "portrait", "landscape". (optional) Default = "portrait".
paginate	(logical) (optional) Default = TRUE for TableTree and FALSE otherwise.
nosplitin	(list) path elements whose children should not be paginated within if it can be avoided. The list should have the format <code>list(row=, col=)</code> . E.g., <code>list(col="TRT01A")</code> means don't split within treatment arms unless all the associated columns don't fit on a single page. (optional) Default = <code>list(row = character(), col = character())</code> .
string_map	(tibble) (optional) Default = <code>default_str_map</code> .
markup_df_docx	(tibble) (optional) Default = <code>dps_markup_df_docx</code> .

combined_docx	(logical) whether to also export an "allparts" docx version. Only applies when exporting a Table or Listing. (optional) Default = FALSE.
tlgtype	(character) (optional) No need to be specified by end user.
col_gap	(numeric) (optional) Default = 3 (Tables) or 0.5 (Listings).
pagenum	(logical) whether to display page numbers. Only applicable to listings (i.e. for tables and figures this argument is ignored). (optional) Default = TRUE for Listings and FALSE otherwise.
round_type	("iec" or "sas") the type of rounding to perform. iec, the default, performs rounding compliant with IEC 60559, while sas performs nearest-value rounding consistent with rounding within SAS. See <code>[formatters::format_value()]</code> for more details.
alignments	(list) list of named lists. Vectorized. Used to specify individual column or cell alignments. Each named list contains row, col, and value. (optional) Default = <code>list()</code> .
border	(fp_border) border to use. Default = <code>flextable::fp_border_default(width = 0.875, color = "black")</code> .
border_mat	(matrix) a $m \times k$ matrix where m is the number of columns of the input Table/Listing and k is the number of lines the header takes up. See tidytlg::add_bottom_borders for what the matrix should contain. Users should only specify this when the default behavior does not meet their needs.
watermark	(character) the watermark (text) to display in the output docx file. If NULL, no watermark will be displayed. (optional) Default = NULL.
export_csv	(logical(1)) Whether to export the object as a csv representation. Default = FALSE.
output_csv_directory	(character(1)) the directory to export the csv. Default = NULL. Only used if <code>export_csv = TRUE</code> . If NULL or attempting to export in a non-existent directory, the csv will be exported in the same directory as the .docx file.
markup_df	(data.frame) Data frame containing markup information. Only used if <code>export_csv = TRUE</code> .
validate	(logical(1)) Whether to validate the table structure using <code>rtables::validate_table_struct()</code> . Defaults to TRUE. If FALSE, a message will be displayed when validation fails.
...	other parameters.

Note

This function may be removed from junco in the future if the functionality is merged into `rtables.officer`.

For more information, refer to the vignette `table_and_listing_customizations` (`browseVignettes("junco")`)

export_graph_as_docx *export_graph_as_docx*

Description

Export graph in DOCX format.

Usage

```
export_graph_as_docx(
  g = NULL,
  plotnames = NULL,
  tblid = NULL,
  output_dir = NULL,
  title = NULL,
  footers = NULL,
  orientation = "portrait",
  plotwidth = 8,
  plotheight = 5.51,
  units = c("in", "cm", "mm", "px")[1],
  border = flextable::fp_border_default(width = 0.875, color = "black"),
  watermark = NULL
)
```

Arguments

<code>g</code>	(<code>ggplot2</code>) a <code>ggplot2</code> object, or a list of them, to export. At least one of <code>g</code> or <code>plotnames</code> must be provided. If both are provided, <code>g</code> precedes and <code>plotnames</code> will be ignored. (optional) Default = <code>NULL</code> .
<code>plotnames</code>	(<code>list</code>) a file path, or a list of them, to previously saved <code>.png</code> files. These will be opened and exported in the output file. At least <code>g</code> (of class <code>ggplot2</code>) or <code>plotnames</code> must be provided. If both are provided, <code>g</code> precedes and <code>plotnames</code> will be ignored. (optional) Default = <code>NULL</code> .
<code>tblid</code>	(<code>character</code>) output ID to be displayed in the title and last line of footer. When exporting, it will also be used as the output filename. If <code>NULL</code> , a temp file will be created, its dirname will replace argument <code>output_dir</code> , and its basename will replace argument <code>tblid</code> . (optional) Default = <code>NULL</code> .

output_dir	(character) a directory path to save the docx. If NULL, a temp file will be created, its dirname will replace argument output_dir, and its basename will replace argument tblid. (optional) Default = NULL.
title	(character) character, or list of them, with the titles to be displayed. (optional) Default = NULL.
footers	(character) a list of footers to be displayed. (optional) Default = NULL.
orientation	(character) one of: "portrait", "landscape". (optional) Default = "portrait".
plotwidth	(numeric) plot size in units expressed by the units argument. If not supplied, uses the size of the current graphics device. (optional) Default = 8.
plotheight	(numeric) plot size in units expressed by the units argument. If not supplied, uses the size of the current graphics device. (optional) Default = 5.51.
units	(character) one of the following units in which the plotwidth and plotheight arguments are expressed: "in", "cm", "mm" or "px". (optional) Default = "in".
border	(fp_border) border to use. Default = <code>flectable::fp_border_default(width = 0.875, color = "black")</code> .
watermark	(character) the watermark (text) to display in the output docx file. If NULL, no watermark will be displayed. (optional) Default = NULL.

Note

This function may be removed from junco in the future if the functionality is merged into `rtables.officer`.

For more information, refer to the vignette `table_and_listing_customizations` (`browseVignettes("junco")`)

export_TLG_as_docx	<i>Export a TLG (Table, Listing, Graph) to .docx format</i>
--------------------	---

Description

Export a TLG (Table, Listing, Graph) to .docx format

Usage

```

export_TLG_as_docx(
  obj = NULL,
  tblid = NULL,
  output_dir = NULL,
  theme = theme_docx_default_j(font = "Times New Roman", font_size = 9L, bold = NULL),
  add_page_break = FALSE,
  titles_as_header = TRUE,
  integrate_footers = TRUE,
  section_properties = officer::prop_section(page_size = officer::page_size(width = 11,
    height = 8.5, orient = orientation), page_margins = officer::page_mar(bottom = 1, top
    = 1, right = 1, left = 1, gutter = 0, footer = 1, header = 1)),
  doc_metadata = NULL,
  template_file = NULL,
  orientation = "portrait",
  paginate = tlgtype == "Table",
  nosplitin = list(row = character(), col = character()),
  string_map = default_str_map,
  markup_df_docx = dps_markup_df_docx,
  combined_docx = FALSE,
  tlgtype = ifelse(is.null(obj), "Figure", tlg_type(obj)),
  col_gap = ifelse(tlgtype == "Listing", 0.5, 3),
  pagenum = ifelse(tlgtype == "Listing", TRUE, FALSE),
  round_type = ifelse(tlgtype %in% c("Table", "Listing"),
    formatters::obj_round_type(obj), "iec"),
  alignments = list(),
  border = flextable::fp_border_default(width = 0.875, color = "black"),
  border_mat = NULL,
  export_csv = FALSE,
  output_csv_directory = NULL,
  markup_df = dps_markup_df,
  validate = TRUE,
  watermark = NULL,
  plotnames = NULL,
  title = NULL,
  footers = NULL,
  plotwidth = 8,
  plotheight = 5.51,
  units = c("in", "cm", "mm", "px")[1],
  ...
)

```

Arguments

<code>obj</code>	(TableTree, listing_df or ggplot2) the object to export.
<code>tblid</code>	(character) output ID to be displayed in the title and last line of footer. When exporting, it

	will also be used as the output filename. If NULL, a temp file will be created, its dirname will replace argument <code>output_dir</code>, and its basename will replace argument <code>tblid</code>. (optional) Default = NULL.
<code>output_dir</code>	(character) a directory path to save the docx. If NULL, a temp file will be created, its dirname will replace argument <code>output_dir</code>, and its basename will replace argument <code>tblid</code>. (optional) Default = NULL.
<code>theme</code>	(function factory) the theme to apply to the flextable. (optional) Default = <code>theme_docx_default_j()</code>. See <code>theme_docx_default_j()</code> or <code>rtables.officer::theme_docx_default()</code> for more details.
<code>add_page_break</code>	(logical) (optional) Default = FALSE.
<code>titles_as_header</code>	(logical) (optional) Default = TRUE.
<code>integrate_footers</code>	(logical) (optional) Default = TRUE.
<code>section_properties</code>	(prop_section) (optional) A "prop_section" object containing information about page size, orientation, margins, etc. See <code>officer::prop_section()</code> for more details. No need to be specified by end user.
<code>doc_metadata</code>	(list of string) any value that can be used as metadata by <code>officer::set_doc_properties()</code>. Important text values are title, subject, creator, and description, while created is a date object. (optional) Default = NULL.
<code>template_file</code>	(character) Template file that officer will use as a starting point for the final document. Document attaches the table and uses the defaults defined in the template file. Paragraph styles are inherited from this file. If NULL, this function will use an internal template. (optional) Default = NULL.
<code>orientation</code>	(character) one of: "portrait", "landscape". (optional) Default = "portrait".
<code>paginate</code>	(logical) (optional) Default = TRUE for TableTree and FALSE otherwise.
<code>nosplitin</code>	(list) path elements whose children should not be paginated within if it can be avoided. The list should have the format <code>list(row=, col=)</code>.

	E.g., <code>list(col="TRT01A")</code> means don't split within treatment arms unless all the associated columns don't fit on a single page. (optional) Default = <code>list(row = character(), col = character())</code>.
<code>string_map</code>	(tibble) (optional) Default = <code>default_str_map</code>.
<code>markup_df_docx</code>	(tibble) (optional) Default = <code>dps_markup_df_docx</code>.
<code>combined_docx</code>	(logical) whether to also export an "allparts" docx version. Only applies when exporting a Table or Listing. (optional) Default = <code>FALSE</code>.
<code>tlgtype</code>	(character) (optional) No need to be specified by end user.
<code>col_gap</code>	(numeric) (optional) Default = 3 (Tables) or 0.5 (Listings).
<code>pagenum</code>	(logical) whether to display page numbers. Only applicable to listings (i.e. for tables and figures this argument is ignored). (optional) Default = <code>TRUE</code> for Listings and <code>FALSE</code> otherwise.
<code>round_type</code>	("iec" or "sas") the type of rounding to perform. iec, the default, performs rounding compliant with IEC 60559, while sas performs nearest-value rounding consistent with rounding within SAS. See <code>[formatters::format_value()]</code> for more details.
<code>alignments</code>	(list) list of named lists. Vectorized. Used to specify individual column or cell alignments. Each named list contains row, col, and value. (optional) Default = <code>list()</code>.
<code>border</code>	(fp_border) border to use. Default = <code>flextable::fp_border_default(width = 0.875, color = "black")</code>.
<code>border_mat</code>	(matrix) a $m \times k$ matrix where m is the number of columns of the input Table/Listing and k is the number of lines the header takes up. See tidytlg::add_bottom_borders for what the matrix should contain. Users should only specify this when the default behavior does not meet their needs.
<code>export_csv</code>	(logical(1)) Whether to export the object as a csv representation. Default = <code>FALSE</code>.
<code>output_csv_directory</code>	(character(1)) the directory to export the csv. Default = <code>NULL</code>. Only used if <code>export_csv = TRUE</code>. If <code>NULL</code> or attempting to export in a non-existent directory, the csv will be exported in the same directory as the .docx file.
<code>markup_df</code>	(data.frame) Data frame containing markup information. Only used if <code>export_csv = TRUE</code>.

validate	(logical(1)) Whether to validate the table structure using <code>rtables::validate_table_struct()</code> . Defaults to TRUE. If FALSE, a message will be displayed when validation fails.
watermark	(character) the watermark (text) to display in the output docx file. If NULL, no watermark will be displayed. (optional) Default = NULL.
plotnames	(character) a file path, or a list of them, to previously saved .png files. These will be opened and exported in the output file. When exporting a Graph, at least <code>obj</code> (of class <code>ggplot2</code>) or <code>plotnames</code> must be provided. If both are provided, <code>obj</code> precedes and <code>plotnames</code> will be ignored. (optional) Default = NULL.
title	(character) character, or list of them, with the titles to be displayed. (optional) Default = NULL.
footers	(character) a list of footers to be displayed. (optional) Default = NULL.
plotwidth	(numeric) plot size in units expressed by the <code>units</code> argument. If not supplied, uses the size of the current graphics device. (optional) Default = 8.
plotheight	(numeric) plot size in units expressed by the <code>units</code> argument. If not supplied, uses the size of the current graphics device. (optional) Default = 5.51.
units	(character) one of the following units in which the <code>plotwidth</code> and <code>plotheight</code> arguments are expressed: "in", "cm", "mm" or "px". (optional) Default = "in".
...	other parameters.

Note

This function may be removed from `junco` in the future if the functionality is merged into `rtables.officer`.

For more information, refer to the vignette `table_and_listing_customizations` (`browseVignettes("junco")`)

Examples

```
adsl <- ex_adsl
adae <- ex_adae
extra_args_1 <- list(
  .stats = c("count_unique_denom_fraction")
)
lyt1 <- basic_table(show_colcounts = TRUE) |>
  split_cols_by("ARM") |>
  analyze(
```

```

    vars = "COUNTRY",
    afun = a_freq_j,
    extra_args = extra_args_1
  )
tbl1 <- build_table(lyt1, adsl)
tab_titles <- list(
  "title" = "This is the main Title",
  "subtitles" = NULL,
  "main_footer" = c(
    "footer 1",
    "footer 2"
  ),
  "prov_footer" = NULL)
tbl1b <- set_titles(tbl1, tab_titles)

export_TLG_as_docx(
  obj = tbl1b,
  tblid = "test",
  output_dir = tempdir(),
  theme = theme_docx_default_j(), add_page_break = FALSE,
  titles_as_header = TRUE, integrate_footers = TRUE,
  section_properties = officer::prop_section(
    page_size = officer::page_size(width = 11, height = 8.5, orient = "portrait"),
    page_margins = officer::page_mar(
      bottom = 1,
      top = 1,
      right = 1,
      left = 1,
      gutter = 0,
      footer = 1,
      header = 1)
  ),
  doc_metadata = NULL,
  template_file = NULL,
  orientation = "portrait",
  paginate = FALSE,
  nosplitin = list(
    row = character(),
    col = character()
  ),
  string_map = default_str_map,
  markup_df_docx = junco::dps_markup_df_docx,
  combined_docx = FALSE,
  tlgttype = "Table",
  col_gap = 3,
  pagenum = FALSE,
  round_type = "iec",
  alignments = list(),
  border = flextable::fp_border_default(width = 0.875, color = "black"),
  border_mat = NULL,
  watermark = NULL,
  plotnames = NULL,
  title = NULL,

```

```

    footers = NULL,
    plotwidth = 8,
    plotheight = 5.51,
    units = "in"
  )

```

factor_by_order

Create a factor with levels ordered by a separate ordering vector

Description

[Stable]

Converts a character vector or factor into a factor where the level order is determined by a second integer-like vector containing the corresponding order values.

Usage

```
factor_by_order(x, y, ordered = FALSE)
```

Arguments

x	(character or factor) A vector to be converted to a factor.
y	(integerish) A vector defining the order of levels in x. Must have the same length as x.
ordered	(logical(1)) Indicates whether the result should be an ordered factor. Defaults to FALSE.

Details

The values in x and y must define a bijection between the unique non-missing values: each unique value in x must correspond to exactly one unique value in y, and vice versa. Missing values are handled separately: NA values in x and y must occur at the same positions. Missing values are not included as factor levels.

If x is already a factor containing unobserved (unused) levels, these unobserved levels will be dropped in the resulting factor.

Value

A factor created from x, with levels ordered according to y. Attributes of x other than class and levels are preserved. If ordered = TRUE, an ordered factor is returned. If x is empty or consists entirely of missing values, a factor with no levels is returned.

Author(s)

WW

Examples

```

factor_by_order(c("A", "A", "B"), c(1, 1, 2))

factor_by_order(c("A", "A", "B"), c(1, 1, 2), ordered = TRUE)

factor_by_order(c("A", "A", "B"), c(2, 2, 1))

factor_by_order(c("A", "A", "B", NA), c(1, 1, 2, NA))

## Not run:
factor_by_order(c("A", "A", "B", NA), c(1, 2, 2, 4))

## End(Not run)

```

```
find_missing_chg_after_avisit
```

Helper for Finding AVISIT after which CHG are all Missing

Description

Helper for Finding AVISIT after which CHG are all Missing.

Usage

```
find_missing_chg_after_avisit(df)
```

Arguments

df (data.frame)
with CHG and AVISIT variables.

Value

A string with either the factor level after which AVISIT is all missing, or NA.

Examples

```

df <- data.frame(
  AVISIT = factor(c(1, 2, 3, 4, 5)),
  CHG = c(5, NA, NA, NA, 3)
)
find_missing_chg_after_avisit(df)

df2 <- data.frame(
  AVISIT = factor(c(1, 2, 3, 4, 5)),
  CHG = c(5, NA, 3, NA, NA)
)
find_missing_chg_after_avisit(df2)

```

```
df3 <- data.frame(
  AVISIT = factor(c(1, 2, 3, 4, 5)),
  CHG = c(NA, NA, NA, NA, NA)
)
find_missing_chg_after_avisit(df3)
```

fit_ancova	ANCOVA Analysis
------------	-----------------

Description

Performs the ANCOVA analysis, separately for each visit.

Usage

```
fit_ancova(
  vars = list(response = "AVAL", covariates = c(), arm = "ARM", visit = "AVISIT", id =
    "USUBJID"),
  data,
  conf_level = 0.95,
  weights_emmeans = "proportional"
)
```

Arguments

vars	(named list of string or character) specifying the variables in the ANCOVA analysis. The following elements need to be included as character vectors and match corresponding columns in data: • response: the response variable. • covariates: the additional covariate terms (might also include interactions). • id: the subject ID variable (not really needed for the computations but for internal logistics). • arm: the treatment group variable (factor). • visit: the visit variable (factor). Note that the arm variable is by default included in the model, thus should not be part of covariates.
data	(data.frame) with all the variables specified in vars. Records with missing values in any independent variables will be excluded.
conf_level	(proportion) confidence level of the interval.
weights_emmeans	(string) argument from <code>emmeans::emmeans()</code> , 'counterfactual' by default.

Value

A `tern_model` object which is a list with model results:

- `fit`: A list with a fitted `stats::lm()` result for each visit.
- `mse`: Mean squared error, i.e. variance estimate, for each visit.
- `df`: Degrees of freedom for the variance estimate for each visit.
- `lsmeans`: This is a list with data frames estimates and contrasts. The attribute `weights` save the settings used (`weights_emmeans`).
- `vars`: The variable list.
- `labels`: Corresponding list with variable labels extracted from data.
- `ref_level`: The reference level for the arm variable, which is always the first level.
- `treatment_levels`: The treatment levels for the arm variable.
- `conf_level`: The confidence level which was used to construct the `lsmeans` confidence intervals.

Examples

```
library(mmrn)

fit <- fit_ancova(
  vars = list(
    response = "FEV1",
    covariates = c("RACE", "SEX"),
    arm = "ARMCD",
    id = "USUBJID",
    visit = "AVISIT"
  ),
  data = fev_data,
  conf_level = 0.9,
  weights_emmeans = "equal"
)
```

fit_mmrn_j

MMRM Analysis

Description

Does the MMRM analysis. Multiple other functions can be called on the result to produce tables and graphs.

Usage

```
fit_mmrn_j(
  vars = list(response = "AVAL", covariates = c(), id = "USUBJID", arm = "ARM", visit =
    "AVISIT", subgroup = NULL),
  data,
  conf_level = 0.95,
  cor_struct = "unstructured",
  weights_emmeans = "counterfactual",
  averages_emmeans = list(),
  mult_adj_emmeans = c("none", "dunnett", "step-down-dunnett"),
  ...
)
```

Arguments

vars	(named list of string or character) specifying the variables in the MMRM. The following elements need to be included as character vectors and match corresponding columns in data: • response: the response variable. • covariates: the additional covariate terms (might also include interactions). • id: the subject ID variable. • arm: the treatment group variable (factor). • visit: the visit variable (factor). • weights: optional weights variable (if NULL or omitted then no weights will be used). Note that the main effects and interaction of arm and visit are by default included in the model.
data	(data.frame) with all the variables specified in vars. Records with missing values in any independent variables will be excluded.
conf_level	(proportion) confidence level of the interval.
cor_struct	(string) specifying the covariance structure, defaults to 'unstructured'. See the details.
weights_emmeans	(string) argument from <code>emmeans::emmeans()</code> , 'counterfactual' by default.
averages_emmeans	(list) optional named list of visit levels which should be averaged and reported along side the single visits.
mult_adj_emmeans	(string) whether to multiplicity adjust LS means contrast p-values and confidence inter-

vals within visits when there are more than 2 treatment arms. Note that this cannot be combined with `averages_emmeans`. Either "none", "dunnett" or "step-down-dunnett".

... additional arguments for `mmrm::mmrm()`, in particular `reml` and options listed in `mmrm::mmrm_control()`.

Details

Multiple different degree of freedom adjustments are available via the method argument for `mmrm::mmrm()`. In addition, covariance matrix adjustments are available via `vcov`. Please see `mmrm::mmrm_control()` for details and additional useful options.

For the covariance structure (`cor_struct`), the user can choose among the following options.

- `unstructured`: Unstructured covariance matrix. This is the most flexible choice and default. If there are T visits, then $T * (T+1) / 2$ variance parameters are used.
- `toeplitz`: Homogeneous Toeplitz covariance matrix, which uses T variance parameters.
- `heterogeneous toeplitz`: Heterogeneous Toeplitz covariance matrix, which uses $2 * T - 1$ variance parameters.
- `ante-dependence`: Homogeneous Ante-Dependence covariance matrix, which uses T variance parameters.
- `heterogeneous ante-dependence`: Heterogeneous Ante-Dependence covariance matrix, which uses $2 * T - 1$ variance parameters.
- `auto-regressive`: Homogeneous Auto-Regressive (order 1) covariance matrix, which uses 2 variance parameters.
- `heterogeneous auto-regressive`: Heterogeneous Auto-Regressive (order 1) covariance matrix, which uses $T + 1$ variance parameters.
- `compound symmetry`: Homogeneous Compound Symmetry covariance matrix, which uses 2 variance parameters.
- `heterogeneous compound symmetry`: Heterogeneous Compound Symmetry covariance matrix, which uses $T + 1$ variance parameters.

Value

A `tern_model` object which is a list with model results:

- `fit`: The `mmrm` object which was fitted to the data. Note that via `mmrm::component(fit, 'optimizer')` the finally used optimization algorithm can be obtained, which can be useful for refitting the model later on.
- `cov_estimate`: The matrix with the covariance matrix estimate.
- `diagnostics`: A list with model diagnostic statistics (REML criterion, AIC, corrected AIC, BIC).
- `lsmeans`: This is a list with data frames estimates and contrasts. The attributes `averages`, `weights` and `mult_adj` save the settings used (`averages_emmeans`, `weights_emmeans` and `mult_adj_emmeans`).
- `vars`: The variable list.
- `labels`: Corresponding list with variable labels extracted from data.

- `cor_struct`: input.
- `ref_level`: The reference level for the arm variable, which is always the first level.
- `treatment_levels`: The treatment levels for the arm variable.
- `conf_level`: The confidence level which was used to construct the `lsmeans` confidence intervals.
- `additional`: List with any additional inputs passed via `...`

Note

This function has the `_j` suffix to distinguish it from `mmrm::fit_mmrm()`. It is modified from the `tern.mmrm` package. The new features are:

- the `mult_adj_emmeans` argument and its functionality
- the `subgroup` variable in `vars` and its functionality

These could later be contributed upstream in `tern.mmrm`.

Examples

```
mmrm_results <- fit_mmrm_j(
  vars = list(
    response = "FEV1",
    covariates = c("RACE", "SEX"),
    id = "USUBJID",
    arm = "ARMCD",
    visit = "AVISIT"
  ),
  data = mmrm::fev_data,
  cor_struct = "unstructured",
  weights_emmeans = "equal",
  averages_emmeans = list(
    "VIS1+2" = c("VIS1", "VIS2")
  )
)
```

fmt_spec_d

Utilities for processing d-style string formatting specifications (eg "xx.dx")

Description

[Experimental]

Usage

```
fmt_spec_single_d(
  d = 1,
  stats_in = NULL,
  fmt_d_def = junco_def_d_all,
  fmt_d_in = NULL
)
```

```
fmt_spec_df_d(
  df_d,
  d_column = "d",
  fmt_column = "fmt_d",
  stats_in = NULL,
  fmt_d_def = junco_def_d_all,
  fmt_d_in = NULL
)
```

```
fmt_spec_var_d(
  var_d,
  stats_in = NULL,
  fmt_d_def = junco_def_d_all,
  fmt_d_in = NULL
)
```

```
junco_def_d_all
```

```
junco_def_d_alt1
```

```
junco_def_d_alt2
```

Arguments

d	integer(1) Value of d for the translation of "xx.dx" into "xx.xx" style (example when d = 1)
stats_in	(character) Set of statistics to restrict to.
fmt_d_def	Named vector with default format specifications where d-style format is allowed.
fmt_d_in	Named vector with format specifications where d-style format is allowed. Formats defined here will take higher precedence over formats in fmt_d_def.
df_d	Dataframe in which the d-column specification is available.
d_column	Name of column within dataframe df_d that contains required integer values for d.
fmt_column	Name of column that will be added to dataframe that will contain resulting format specifications.
var_d	Named vector of d specifications. Names are to be considered as variables to be analyzed in further processing.

Format

- `junco_def_d_all` Used as default input vector for argument `fmt_d_def` in functions `fmt_spec_single_d()`, `fmt_spec_df_d()` and `fmt_spec_var_d()` (corresponds to the named vector of with default available formats `junco_default_formats` with `d = 1`)
- `junco_def_d_alt1` Alternative vector of d-style specification defaults.
- `junco_def_d_alt2` Another alternative vector of d-style specification defaults.

Details

Format specifications in d-style notation (such as "xx.dx") will be translated to string based formats (or wrapped with `jjcsformat_xx()`), eg "xx.dx" will be translated into "xx.xx", when `d = 1`.

Functions

- `fmt_spec_single_d()`: Function to convert a named list using d-style string format specification into a named list with valid formatters formatting specification
- `fmt_spec_df_d()`: Function to convert an input column with d-style formatting specification into a new column with valid formatters formatting specification.
The new column can be passed onto argument `formats_var` in `rtables::analyze()` function.
- `fmt_spec_var_d()`: Function to convert a named list of variables with d-style formatting specification into a named list with valid formatters formatting specification.
The result can be used as input for argument `format` in `rtables::analyze()` call.
- `junco_def_d_all`: Named vector of default format specification using d-style notation.
- `junco_def_d_alt1`: Alternative vector of d-style specification defaults.
- `junco_def_d_alt2`: Another alternative vector of d-style specification defaults.

Examples

```
junco_def_d <- c(
  "mean" = "xx.dx",
  "mean_sd" = "xx.dx (xx.dxx)",
  "mean_se" = "xx.dx (xx.dxx)",
  "range" = "(xx.d, xx.d)"
)
myfmts <- fmt_spec_single_d(
  d = 1,
  stats_in = c("mean", "mean_sd"),
  fmt_d_def = junco_def_d,
  fmt_d_in = NULL
)
myfmts

# example for fmt_spec_df_d ----
library(dplyr)
df_d <- tribble(
  ~PARAMCD, ~d,
```

```

    "DIABP", 2L,
    "PULSE", 3L,
    "RESP", 1L
  )
yy <- fmt_spec_df_d(df_d,
  d_column = "d",
  fmt_column = "fmt_d",
  stats_in = NULL,
  fmt_d_def = junco_def_d,
  fmt_d_in = NULL
)

df <- ex_advs |>
  filter(PARAMCD %in% c("DIABP", "PULSE", "RESP")) |>
  filter(AVISIT %in% c("BASELINE", "WEEK 1 DAY 8", "WEEK 2 DAY 15"))
df <- df |>
  left_join(yy)

lyt <- basic_table() |>
  split_cols_by("ARMCD") |>
  split_rows_by("PARAMCD", split_fun = drop_split_levels) |>
  split_rows_by("AVISIT", split_fun = drop_split_levels) |>
  analyze(
    vars = "AVAL",
    afun = tern::a_summary,
    extra_args = list(
      .stats = c("n", "mean_se", "range"),
      .formats = "default"
    ),
    formats_var = "fmt_d"
  )

rslt <- build_table(lyt, df, alt_counts_df = ex_adsl)
rslt

# example for fmt_spec_var_d ----
var <- c("AGE" = 0, "BMI" = 1, "BMRKR1" = 2)
vars_fmt <- fmt_spec_var_d(var,
  stats_in = NULL,
  fmt_d_def = junco_def_d,
  fmt_d_in = NULL
)
vars_fmt
lyt <- basic_table() |>
  split_cols_by("ARMCD") |>
  analyze(
    vars = c("AGE", "BMRKR1"),
    afun = tern::a_summary,
    extra_args = list(
      .stats = c("n", "mean_sd", "range"),
      .formats = "default"
    ),
  ),

```

```

      format = vars_fmt,
      section_div = " "
    )

    rslt <- build_table(lyt, ex_adsl)
    rslt

```

 fmt_utils

Utilities for reviewing format specification objects

Description

[Experimental]

Usage

```
get_fmt_details(myfmts, recursive = FALSE, as_tibble = TRUE)
```

```
compare_fmt_specs(fmt1, fmt2)
```

Arguments

myfmts	A named list of either formatting specifications (ie stats with its associated formats) or a list of variables with formatting specifications.
recursive	logical(1) Set to TRUE when requesting details from named list of variables with specifications. Set to FALSE when requesting details from named list of statistics with format specifications.
as_tibble	logical(1) When TRUE, convert the information into a tibble for ease of reviewing.
fmt1	A named list of formatting specifications (ie stats with its associated format)
fmt2	Another named list of formatting specifications (ie stats with its associated format), to compare against fmt1

Details

fmt_utils general details

Functions

- `get_fmt_details()`: A utility function to collect information on formatting specification in list or tibble format.

The following information is collected:

- var (variable name): Only applicable if the format specification is a named list with formattings specs (eg `list("AGE" = c("mean" = "xx", "se" = "xx.x"), "BMI" = c("mean" = "xx.x", "se" = "xx.xx"))`), eg processed via `fmt_spec_var_d` function.
- stat (name of statistic)

- fun_fact : name of function factory that has produced the format specification (if spec is from formatting function, blank if spec is string).
Only the following known formatting functions are supported:
 - * jjcsformat_xx
 - * jjcsformat_cnt_den_fract_fct,
 - * jjcsformat_pval_fct,
 - * jjcsformat_range_fct,
 - * tern::format_xx,
 - * tern::format_extreme_values,
 - * tern::format_extreme_values_ci,
 - * tern::format_count_fraction_fixed_dp,
 - * tern::format_count_fraction,
 - * tern::format_fraction_fixed_dp,
 - * format_sigfig_j
- str: string from specification, or str from formatting function factory (jjcsformat_xx, ...)
- str_formatters: (logical) Is the str value a valid format string as in ([list_valid_format_labels\(\)](#))?
- type: when formatting function factory jjcsformat_cnt_den_fract_fct is used
- alpha: when formatting function factory jjcsformat_pval_fct is used
- message: Message when formatting function is unknown, or input string was not valid format string as in ([list_valid_format_labels\(\)](#))
- fmt_package: In which package was the formatting specification defined.
- is_fun: (logical) Is the formatting specification a function?
- round_type: (logical) Does the formatting support round_type argument/behavior?
- fun: Formatting string or body of the formatting function.

Only when one of the formats provided is based upon formatting function factory format_sigfig_j setup:

- d: value used for sigfig
 - whole_integer: value used for whole_integer
 - drop0trailing: value used for drop0trailing
 - zero_threshold: value used for zero_threshold
- compare_fmt_specs(): A utility to report findings in differences between 2 formatting specification vectors. It returns a list object with the following information:
 - spec1 : For statistics with a different specification, tibble with specification details from fmt1.
 - spec2 : For statistics with a different specification, tibble with specification details from fmt2. The tibbles in spec1 and spec2 contain information as described in [get_fmt_details\(\)](#).
 - msg: Message on how many stats have a difference in specification.
 - comm_stats: Vector of stats in common between fmt1 and fmt2.
 - diff_stats: Vector of stats with any difference identified.

Examples

```

# Example for get_fmt_details --
junco_def_d <- c(
  "mean" = "xx.dx",
  "mean_sd" = "xx.dx (xx.dxx)",
  "range" = "(xx.d, xx.d)"
)

myfmts <- fmt_spec_single_d(
  d = 1,
  stats_in = c("mean", "mean_sd"),
  fmt_d_def = junco_def_d,
  fmt_d_in = NULL
)

fmt_details <- get_fmt_details(myfmts, as_tibble = TRUE)

myfmts2 <- fmt_spec_single_d(
  d = 3,
  stats_in = c("mean", "mean_sd"),
  fmt_d_def = junco_def_d,
  fmt_d_in = NULL
)
fmt_details2 <- get_fmt_details(myfmts2, as_tibble = TRUE)
fmt_details2

# Example for get_fmt_details using format specs for variables --
# processed with fmt_spec_var_d
var <- c("AGE" = 0, "BMI" = 1, "BMRKR1" = 2)
vars_fmt <- fmt_spec_var_d(var,
  stats_in = NULL,
  fmt_d_def = junco_def_d,
  fmt_d_in = NULL
)
fmt_details2 <- get_fmt_details(vars_fmt, as_tibble = TRUE, recursive = TRUE)

# Example for compare_fmt_specs --
test1_target <- junco_default_formats
# junco_def_d_all has been constructed with d = 1 as reference
# compare against junco_default_formats
test1_d <- fmt_spec_single_d(
  d = 1,
  stats_in = NULL,
  fmt_d_def = junco_def_d_all,
  fmt_d_in = NULL
)

check1 <- compare_fmt_specs(test1_d, test1_target)
check1

# more differences when using other d

```

```
test2_d <- fmt_spec_single_d(
  d = 2,
  stats_in = NULL,
  fmt_d_def = junco_def_d_all,
  fmt_d_in = NULL
)

check2 <- compare_fmt_specs(test2_d, test1_target)
```

get_mmrmlsmeans

*Extract Least Square Means from MMRM***Description**

Extracts the least square means from an MMRM fit.

Usage

```
get_mmrmlsmeans(
  fit,
  vars,
  conf_level,
  weights,
  averages = list(),
  mult_adj = c("none", "dunnett", "step-down-dunnett")
)
```

Arguments

fit	(mmrm) result of <code>mmrm::mmrm()</code> .
vars	(named list of string or character) specifying the variables in the MMRM. The following elements need to be included as character vectors and match corresponding columns in data: • response: the response variable. • covariates: the additional covariate terms (might also include interactions). • id: the subject ID variable. • arm: the treatment group variable (factor). • visit: the visit variable (factor). • weights: optional weights variable (if NULL or omitted then no weights will be used).

Note that the main effects and interaction of arm and visit are by default included in the model.

conf_level	(proportion) confidence level of the interval.
weights	(string) type of weights to be used for the least square means, see emmeans::emmeans() for details.
averages	(list) named list of visit levels which should be averaged and reported along side the single visits.
mult_adj	(string) multiplicity adjustment within visits for the contrasts.

Value

A list with data frames estimates and contrasts. The attributes averages, weights and mult_adj save the settings used.

Note

This is modified from `tern.mmrn` and has the additional `mult_adj` argument.

get_ref_info	<i>Obtain reference group information from split context.</i>
--------------	---

Description**[Stable]**

`get_ref_info()` identifies a reference group defined by a column-split path and returns both the reference-group data and an indicator of whether the current column is the reference column. It is intended for use inside custom `rtables` analysis functions.

The reference group is specified using `ref_path`, which consists of alternating column-split variable names and its corresponding levels. For example, `c("SEX", "F", "ARM", "Placebo")` specifies the column-split path where SEX is "F" and ARM is "Placebo".

Usage

```
get_ref_info(ref_path, .spl_context, .var = NULL)
```

Arguments

ref_path	(character) Reference group specification as an <code>rtables</code> colpath; see Details.
.spl_context	(data.frame) Ancestor split-state information passed by <code>rtables</code> .
.var	(character(1)) The variable being analyzed; see rtables::additional_fun_params . If supplied, the corresponding column is extracted from the reference-group data. If NULL, the complete reference-group data frame is returned.

Value

A list with the following elements:

- `in_ref_col` (logical(1) or NULL) indicates whether the current column matches the reference path. This corresponds to `.in_ref_col` in [rtables::additional_fun_params](#).
- `ref_group` (data.frame, vector, or NULL) contains the observations belonging to the reference group. If `.var` is NULL, the complete data frame is returned; otherwise, the column specified by `.var` is returned. This corresponds to `.ref_group` in [rtables::additional_fun_params](#).

If the reference path is not present in the current column-split hierarchy, both elements are NULL.

Examples

```
dm <- DM
dm$colspan_trt <- factor(
  ifelse(dm$ARM == "B: Placebo", " ", "Active Study Agent"),
  levels = c("Active Study Agent", " ")
)
colspan_trt_map <- create_colspan_map(
  dm,
  non_active_grp = "B: Placebo",
  non_active_grp_span_lbl = " ",
  active_grp_span_lbl = "Active Study Agent",
  colspan_var = "colspan_trt",
  trt_var = "ARM"
)

# A standard analysis function which uses a reference group.
standard_afun <- function(x, .ref_group, .in_ref_col) {
  diff_means <- if (isFALSE(.in_ref_col)) {
    mean(x) - mean(.ref_group)
  } else {
    NULL
  }
  in_rows(
    m = rcell(mean(x), label = "Mean"),
    dm = rcell(diff_means, label = "Difference in Means vs Placebo"),
    .formats = "xx.xx"
  )
}

# The custom analysis function which can work with a global reference group.
result_afun <- function(x, ref_path, .spl_context, .var) {
  ref <- get_ref_info(ref_path, .spl_context, .var)
  standard_afun(x, .ref_group = ref$ref_group, .in_ref_col = ref$in_ref_col)
}

ref_path <- c("colspan_trt", " ", "ARM", "B: Placebo")

lyt <- basic_table() |>
  split_cols_by("colspan_trt", split_fun = trim_levels_to_map(colspan_trt_map)) |>
  split_cols_by("ARM") |>
```

```

add_overall_col("Total") |>
analyze("AGE", afun = result_afun, extra_args = list(ref_path = ref_path))

build_table(lyt, dm)

```

get_titles_from_file *Get Titles/Footers For Table From Sources*

Description

Retrieves the titles and footnotes for a given table from a CSV/XLSX file or a data.frame.

Usage

```

get_titles_from_file(
  id,
  file = .find_titles_file(input_path),
  input_path = ".",
  title_df = .read_titles_file(file)
)

```

Arguments

id	(character(1)) The identifier for the table of interest.
file	(character(1)) A path to CSV or xlsx file containing title and footer information for one or more outputs. See Details. Ignored if title_df is specified.
input_path	(character(1)) A path to look for titles.csv/titles.xlsx. Ignored if file or title_df is specified.
title_df	(data.frame) A data.frame containing titles and footers for one or more outputs. See Details.

Details

Retrieves the titles for a given output id (see below) and outputs a list containing the title and footnote objects supported by rtables. Both titles.csv and titles.xlsx (*if readxl is installed*) files are supported, with titles.csv being checked first.

Data is expected to have `TABLE ID`, `IDENTIFIER`, and `TEXT` columns, where `IDENTIFIER` has the value `TITLE` for a title and `FOOT\*` for footer materials where `\*` is a positive integer. `TEXT` contains the value of the title/footer to be applied.

Value

List object containing: title, subtitles, main_footer, prov_footer for the table of interest. Note: the subtitles and prov_footer are currently set to NULL. Suitable for use with [set_titles\(\)](#).

get_visit_levels	<i>Get Visit Levels in Order Defined by Numeric Version</i>
------------------	---

Description

Get Visit Levels in Order Defined by Numeric Version

Usage

```
get_visit_levels(visit_cat, visit_n)
```

Arguments

- visit_cat (character)
the categorical version.
- visit_n (numeric)
the numeric version.

Value

The unique visit levels in the order defined by the numeric version.

Examples

```
get_visit_levels(  
  visit_cat = c("Week 1", "Week 11", "Week 2"),  
  visit_n = c(1, 5, 2)  
)
```

grouped_cols_w_diffs	<i>Standard Column Structure With Grouped Treatments and Difference Columns</i>
----------------------	---

Description

Standard Column Structure With Grouped Treatments and Difference Columns

Usage

```
grouped_cols_w_diffs(  
  lyt,  
  colspan_trt_map,  
  combo_map_df = NULL,  
  comp_map = NULL,  
  diff_cols = TRUE,  
  diffs_label = "Risk Differences",
```

```

    .main_pre = list(),
    .main_post = list(),
    .rr_pre = list(),
    .rr_post = list()
  )

  grouped_cols_w_subgrps(
    lyt,
    colspan_trt_map = NULL,
    combo_map_df = NULL,
    trtvar = names(colspan_trt_map)[2],
    subgrp_var = NULL,
    subgrp_lbl = subgrp_var,
    total_facet = TRUE,
    total_lbl = "Total",
    .pre = list(),
    .post = list()
  )

```

Arguments

<code>lyt</code>	(PreDataTableLayouts). The layout to modify. This should virtually always be the object returned by <code>basic_table</code> .
<code>colspan_trt_map</code>	(data.frame). The spanning label map for the main columns, as given by <code>create_colspan_map</code> .
<code>combo_map_df</code>	(data.frame or NULL). A combination data frame as defined by add_combo_levels() with an additional <code>is_control</code> column indicating whether the virtual level will act as a reference (TRUE) or active (FALSE) group.
<code>comp_map</code>	(data.frame or NULL). A data.frame with columns "active", "comparator", "active_is_combo" and "comparator_is_combo", or NULL indicating the default comparison behavior (See Details).
<code>diff_cols</code>	(logical(1)). Whether the risk difference column structure should be included (TRUE, the default) or not (FALSE).
<code>diffs_label</code>	(character(1)). The spanning label for the risk difference section of columns
<code>.main_pre</code>	(list of functions). Passed to rtables::make_split_fun() as pre for treatment split in main structure.
<code>.main_post</code>	(list of functions). Passed to rtables::make_split_fun() as post for treatment split in main structure.
<code>.rr_pre</code>	(list of functions). Passed to make_multicomp_splfun() as .pre for risk difference faceting.
<code>.rr_post</code>	(list of functions). Passed to make_multicomp_splfun() as .post for risk difference faceting.
<code>trtvar</code>	(character(1) or NULL) the treatment variable to split by. Defaults to the treatment variable in <code>colspan_trt_map</code> .

subgrp_var	(character(1) or NULL) the name of the subgroup variable to split by within the trtvar split
subgrp_lbl	(character(1) or NULL) the spanning label to place over the subgroups, if different than subgrp_var
total_facet	(logical(1)) should an overall facet be prepended to the partition defined by subgrp_var? Defaults to TRUE.
total_lbl	(character(1)) Label to be used for the overall facet, if applicable. Defaults to "Total"; ignored if total_facet is FALSE.
.pre	(list of functions) Passed to <code>rtables::make_split_fun()</code> as pre for treatment splitting.
.post	(list of functions) Passed to <code>rtables::make_split_fun()</code> as post for treatment splitting after the standard column-structure processing.

Details

This function combines multiple `rtables` column splitting instructions with customized split functions to create a column structure with treatment columns for each treatment arm (optionally including combination arms), grouped by active and non-active, with risk difference columns comparing active arm(s) against one or more non-active controls. It is intended for use in layouts that will use `a_freq_j()` or similar junco-style analysis functions which support risk difference columns and accept a `ref_path` argument.

It is equivalent to the following sequence of layout instructions:

1. splitting on a colspan labeling variable with `rtables::trim_levels_to_map()` as the split function;
2. splitting on treatment;
3. adding a (non-nested) overall column acting as the risk difference spanning label; and finally
4. splitting on treatment using `make_multicomp_splfun()` as the split function

In addition, it supports:

- comparison against multiple control groups (as specified by `colspan_trt_map` and/or `comp_map`),
- virtual combination-levels as active an/or control "treatments" (via `combo_map_df`),
- full control of which comparisons are performed, and their order (via `comp_map`).

If combination levels are declared via `combo_map_df` but none appear in `colspan_trt_map`, all combinations will be added to the appropriate group within the map based on `combo_map_df$is_control` (assumed to be FALSE if the column is missing), with a warning.

If some combination levels *do* appear in `combo_map_df` but others do not, a warning will be thrown but the missing combination levels will *not* be added to the treatment map.

By default (when `comp_map` is NULL), all active treatments, including active combinations, will be compared against all control groups.

The risk difference section of the structure is declared using `make_multicomp_splfun()`. Reference paths are inferred automatically from `colspan_trt_map` (after combination levels have been added if necessary).

For the purposes of `pathin` in the resulting structure, `diffs_label` will be both the split name and split value of the parent containing the individual risk difference columns.

`grouped_cols_w_subgrps` creates a hierarchical column structure that splits by `trtvar`, underneath which is a spanning label over a split with a Total column along with columns for each level of `subgrp_var`.

Value

lyt updated with the specified main and risk difference column structures added

See Also

Other `std_col_struct`: `make_multicomp_splfun()`, `shift_tbl_col_struct()`

Examples

```
colspan_var <- create_colspan_var(
  data.frame(TRT01A = factor(c("Placebo", "Active 1", "Active 2"))),
  non_active_grp = "Placebo",
  non_active_grp_span_lbl = "Control",
  active_grp_span_lbl = "Active Treatment",
  colspan_var = "colspan_trt",
  trt_var = "TRT01A"
)
colspan_trt_map <- create_colspan_map(
  colspan_var,
  non_active_grp = "Placebo",
  non_active_grp_span_lbl = "Control",
  active_grp_span_lbl = "Active Treatment",
  colspan_var = "colspan_trt",
  trt_var = "TRT01A"
)

lyt <- basic_table() |>
  grouped_cols_w_diffs(colspan_trt_map) |>
  analyze("TRT01A", afun = function(x, ...) length(x))

build_table(lyt, colspan_var)
dat <- create_colspan_var(
  data.frame(
    TRT01A = factor(rep(c("Placebo", "Active 1"), each = 2)),
    SEX = factor(rep(c("Female", "Male"), 2))
  ),
  non_active_grp = "Placebo",
  non_active_grp_span_lbl = "Control",
  active_grp_span_lbl = "Active Treatment",
  colspan_var = "colspan_trt",
  trt_var = "TRT01A"
```

```

)
colspan_trt_map <- create_colspan_map(
  dat,
  non_active_grp = "Placebo",
  non_active_grp_span_lbl = "Control",
  active_grp_span_lbl = "Active Treatment",
  colspan_var = "colspan_trt",
  trt_var = "TRT01A"
)

lyt <- basic_table() |>
  grouped_cols_w_subgrps(
    colspan_trt_map,
    subgrp_var = "SEX",
    subgrp_lbl = "SUB_*"
  ) |>
  analyze("TRT01A", afun = function(x, ...) length(x))

build_table(lyt, dat)

```

h_odds_ratio

*Helper functions for odds ratio estimation***Description****[Stable]**

Functions to calculate odds ratios in [s_odds_ratio_j\(\)](#).

Usage

```

or_glm_j(data, conf_level)

or_clogit_j(data, conf_level, method = "exact")

or_cmh(data, conf_level)

```

Arguments

data	(data.frame) data frame containing at least the variables <code>rsp</code> and <code>grp</code> , and optionally <code>strata</code> for or_clogit_j() .
conf_level	(numeric) confidence level for the confidence interval.
method	(string) whether to use the correct ('exact') calculation in the conditional likelihood or one of the approximations, or the CMH method. See survival::clogit() for details.

Value

A named list of elements `or_ci`, `n_tot` and `pval`.

Functions

- `or_glm_j()`: Estimates the odds ratio based on `stats::glm()`. Note that there must be exactly 2 groups in data as specified by the `grp` variable.
- `or_clogit_j()`: Estimates the odds ratio based on `survival::clogit()`. This is done for the whole data set including all groups, since the results are not the same as when doing pairwise comparisons between the groups.
- `or_cmh()`: Estimates the odds ratio based on CMH. Note that there must be exactly 2 groups in data as specified by the `grp` variable.

See Also

[odds_ratio](#)

Examples

```
data <- data.frame(
  rsp = as.logical(c(1, 1, 0, 1, 0, 0, 1, 1)),
  grp = letters[c(1, 1, 1, 2, 2, 2, 1, 2)],
  strata = letters[c(1, 2, 1, 2, 2, 2, 1, 2)],
  stringsAsFactors = TRUE
)

or_glm_j(data, conf_level = 0.95)

data <- data.frame(
  rsp = as.logical(c(1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0)),
  grp = letters[c(1, 1, 1, 2, 2, 2, 3, 3, 3, 3, 1, 1, 1, 2, 2, 2, 3, 3, 3, 3)],
  strata = LETTERS[c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2)],
  stringsAsFactors = TRUE
)

or_clogit_j(data, conf_level = 0.95)

set.seed(123)
data <- data.frame(
  rsp = as.logical(rbinom(n = 40, size = 1, prob = 0.5)),
  grp = letters[sample(1:2, size = 40, replace = TRUE)],
  strata = LETTERS[sample(1:2, size = 40, replace = TRUE)],
  stringsAsFactors = TRUE
)

or_cmh(data, conf_level = 0.95)
```

inches_to_spaces	<i>Conversion of inches to spaces.</i>
------------------	--

Description

Conversion of inches to spaces.

Usage

```
inches_to_spaces(ins, fontspec, raw = FALSE, tol = sqrt(.Machine$double.eps))
```

Arguments

<code>ins</code>	(numeric) Vector of widths in inches.
<code>fontspec</code>	(font_spec) The font specification to use.
<code>raw</code>	(logical(1)) Should the answer be returned unrounded (TRUE), or rounded to the nearest reasonable value (FALSE, the default).
<code>tol</code>	(numeric(1)) The numeric tolerance. Values between an integer n , and $n+tol$ will be returned as n , rather than $n+1$, if <code>raw == FALSE</code> . Ignored when <code>raw</code> is TRUE.

Value

The number of either fractional (`raw = TRUE`) or whole (`raw = FALSE`) spaces that will fit within `ins` inches in the specified font.

insert_blank_line	<i>Insertion of Blank Lines in a Layout</i>
-------------------	---

Description

This is a hack for `rtables` in order to be able to add row gaps, i.e. blank lines. In particular, by default this function needs to maintain a global state for avoiding duplicate table names. The global state variable is hidden by using a dot in front of its name. However, this likely won't work with parallelisation across multiple threads and also causes non-reproducibility of the resulting `rtables` object. Therefore also a custom table name can be used.

Usage

```
insert_blank_line(lyt, table_names = NULL)
```

Arguments

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

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

Value

The modified layout now including a blank line after the current row content.

Examples

```
ADSL <- ex_adsl

lyt <- basic_table() |>
  split_cols_by("ARM") |>
  split_rows_by("STRATA1") |>
  analyze(vars = "AGE", afun = function(x) {
    in_rows(
      "Mean (sd)" = rcell(c(mean(x), sd(x)), format = "xx.xx (xx.xx)")
    )
  }) |>
  insert_blank_line() |>
  analyze(vars = "AGE", table_names = "AGE_Range", afun = function(x) {
    in_rows(
      "Range" = rcell(range(x), format = "xx.xx - xx.xx")
    )
  })
build_table(lyt, ADSL)
```

jj_complex_scorefun *Complex Scoring Function*

Description

A function used for sorting AE tables (and others) as required.

Usage

```
jj_complex_scorefun(
  spanningheadercolvar = "colspan_trt",
  usefirstcol = FALSE,
  colpath = NULL,
  firstcat = NULL,
  lastcat = NULL,
  cellvalue_index = 1
)
```

Arguments

spanningheadercolvar	(character) Name of spanning header variable that defines the active treatment columns. If you do not have an active treatment spanning header column then user can define this as NA.
usefirstcol	(logical) This allows you to just use the first column of the table to sort on.
colpath	(character) Name of column path that is needed to sort by (default=NULL). This overrides other arguments if specified (except firstcat and lastcat which will be applied if requested on this colpath).
firstcat	(logical) If you wish to put any category at the top of the list despite any n's, user can specify it here.
lastcat	(logical) If you wish to put any category at the bottom of the list despite any n's, user can specify it here.
cellvalue_index	(integer) Index of the cell value to perform the scoring on.

Details

This sort function sorts as follows:

- Takes all the columns from a specified spanning column header (default= colspan_trt) and sorts by the last treatment column within this.
- If no spanning column header variable exists (e.g you have only one active treatment arm and have decided to remove the spanning header from your layout), it will sort by the first treatment column in your table.

This function is not really designed for tables that have sub-columns. However, if users wish to override any default sorting behavior, they can simply specify their own colpath to use for sorting on (default = NULL)

Value

A function which can be used as a score function (scorefun in sort_at_path).

Examples

```
library(dplyr)
ADAE <- data.frame(
  USUBJID = c(
    "XXXXX01", "XXXXX02", "XXXXX03", "XXXXX04", "XXXXX05",
    "XXXXX06", "XXXXX07", "XXXXX08", "XXXXX09", "XXXXX10"
  ),
```

```

AEBODSYS = c(
  "SOC 1", "SOC 2", "SOC 1", "SOC 2", "SOC 2",
  "SOC 2", "SOC 2", "SOC 1", "SOC 2", "SOC 1"
),
AEDECOD = c(
  "Coded Term 2", "Coded Term 1", "Coded Term 3", "Coded Term 4",
  "Coded Term 4", "Coded Term 4", "Coded Term 5", "Coded Term 3",
  "Coded Term 1", "Coded Term 2"
),
TRT01A = c(
  "ARMA", "ARMB", "ARMA", "ARMB", "ARMB",
  "Placebo", "Placebo", "Placebo", "ARMA", "ARMB"
),
TRTEMFL = c("Y", "Y", "N", "Y", "Y", "Y", "Y", "N", "Y", "Y")
)

ADAE <- ADAE |>
  dplyr::mutate(TRT01A = as.factor(TRT01A))

ADAE$colspan_trt <- factor(ifelse(ADAE$TRT01A == "Placebo", " ", "Active Study Agent"),
  levels = c("Active Study Agent", " ")
)

ADAE$rrisk_header <- "Risk Difference (%) (95% CI)"
ADAE$rrisk_label <- paste(ADAE$TRT01A, paste("vs", "Placebo"))

colspan_trt_map <- create_colspan_map(ADAE,
  non_active_grp = "Placebo",
  non_active_grp_span_lbl = " ",
  active_grp_span_lbl = "Active Study Agent",
  colspan_var = "colspan_trt",
  trt_var = "TRT01A"
)

ref_path <- c("colspan_trt", " ", "TRT01A", "Placebo")

ADSL <- unique(ADAE |> select(USUBJID, "colspan_trt", "rrisk_header", "rrisk_label", "TRT01A"))

lyt <- basic_table() |>
  split_cols_by(
    "colspan_trt",
    split_fun = trim_levels_to_map(map = colspan_trt_map)
  ) |>
  split_cols_by("TRT01A") |>
  split_cols_by("rrisk_header", nested = FALSE) |>
  split_cols_by(
    "TRT01A",
    labels_var = "rrisk_label",
    split_fun = remove_split_levels("Placebo")
  ) |>
  analyze(
    "TRTEMFL",
    a_freq_j,

```

```

      show_labels = "hidden",
      extra_args = list(
        method = "wald",
        label = "Subjects with >=1 AE",
        ref_path = ref_path,
        .stats = "count_unique_fraction"
      )
    ) |>
    split_rows_by("AEBODSYS",
      split_label = "System Organ Class",
      split_fun = trim_levels_in_group("AEDECOD"),
      label_pos = "topleft",
      section_div = c(" "),
      nested = FALSE
    ) |>
    summarize_row_groups(
      "AEBODSYS",
      cfun = a_freq_j,
      extra_args = list(
        method = "wald",
        ref_path = ref_path,
        .stats = "count_unique_fraction"
      )
    ) |>
    analyze(
      "AEDECOD",
      afun = a_freq_j,
      extra_args = list(
        method = "wald",
        ref_path = ref_path,
        .stats = "count_unique_fraction"
      )
    )
  )

result <- build_table(lyt, ADAE, alt_counts_df = ADSL)

result

result <- sort_at_path(
  result,
  c("root", "AEBODSYS"),
  scorefun = jj_complex_scorefun()
)

result <- sort_at_path(
  result,
  c("root", "AEBODSYS", "*", "AEDECOD"),
  scorefun = jj_complex_scorefun()
)

result

```

jjcs_num_formats *Numeric Formatting Function*

Description

Formatting setter for selected numerical statistics.

Usage

```
jjcs_num_formats(d, cap = 4)
```

Arguments

d	(numeric) precision of individual values
cap	(numeric) cap to numerical precision (d > cap – will use precision as if cap was specified as precision)

Value

list:

- `fmt` : named vector with formatting function (`jjcsformat_xx`) for numerical stats: range, median, mean_sd, sd
- `spec` : named vector with formatting specifications for numerical stats: range, median, mean_sd, sd

Examples

```
P1_precision <- jjcs_num_formats(d = 0)$fmt
jjcs_num_formats(2)$fmt
jjcs_num_formats(2)$spec
```

jjcsformat_fraction *Formatting function for percentage values*

Description**[Experimental]**

Formats fractional values as percentages with special consideration for values that round to 0 or 100, as well as missing values. The input is expected to be a fraction, which is multiplied by 100 internally before formatting. Missing values are formatted as "-".

Usage

```
jjcsformat_fraction(  
  x,  
  round_type = valid_round_type,  
  output = c("ascii", "html"),  
  d = 1,  
  add_pct = FALSE,  
  verbose = FALSE,  
  ...  
)
```

Arguments

x	(numeric(1)) Fraction to format as a percentage (e.g. 0.15 represents 15%. NA is formatted as "_").
round_type	(character(1)) The type of rounding to perform. See formatters::format_value() for more details.
output	(character(1)) Output type. See formatters::format_value() for more details.
d	(numeric(1)) Number of digits to round the percentage to.
add_pct	(logical(1)) Whether to append the percentage symbol ("%") to the formatted value.
verbose	(logical(1)) Whether to print verbose output.
...	Additional arguments passed to other methods.

Value

A formatted string.

Author(s)

WW

See Also

Other JJCS formatting functions: [count and fraction related formatting functions](#), [jjcsformat_xx\(\)](#)

Examples

```
jjcsformat_fraction(0.15)  
jjcsformat_fraction(0.15, d = 2)  
jjcsformat_fraction(0.15, d = 2, verbose = TRUE)
```

jjcsformat_xx

*Utility for specifying custom formats***Description**

Utility for specifying custom formats that can be used as a format in `formatters::format_value`

A function factory to generate formatting functions for p-value formatting that support rounding close to the significance level specified.

A function factory to generate formatting functions for range formatting that includes information about the censoring of survival times.

Usage

```
jjcsformat_xx(
  str,
  na_str = na_str_dflt,
  na_str_dflt = "NE",
  replace_na_dflt = TRUE
)

jjcsformat_pval_fct(alpha = 0.05)

jjcsformat_range_fct(str, censor_char = "+")
```

Arguments

<code>str</code>	(string) the format specifying the number of digits to be used, for the range values, e.g. "xx.xx".
<code>na_str</code>	String for NA values.
<code>na_str_dflt</code>	Character to represent NA value
<code>replace_na_dflt</code>	logical(1). Should an na_string of "NA" within the formatters framework be overridden by na_str_default? Defaults to TRUE, as a way to have a different default na string behavior from the base formatters framework.
<code>alpha</code>	(numeric) the significance level to account for during rounding.
<code>censor_char</code>	(string) the character (of length 1) to be appended to min or max

Value

Either a supported format string, or a formatting function that can be used as format in `formatters::format_value`

The p-value in the standard format. If count is 0, the format is 0. If it is smaller than 0.001, then <0.001, if it is larger than 0.999, then >0.999 is returned. Otherwise, 3 digits are used. In the

special case that rounding from below would make the string equal to the specified alpha, then a higher number of digits is used to be able to still see the difference. For example, 0.0048 is not rounded to 0.005 but stays at 0.0048 if `alpha = 0.005` is set.

A function that formats a numeric vector with 4 elements:

- minimum
- maximum
- censored minimum? (1 if censored, 0 if event)
- censored maximum? (1 if censored, 0 if event) The range along with the censoring information is returned as a string with the specified numeric format as (min, max), and the `censor_char` is appended to min or max if these have been censored.

See Also

Other JJCS formatting functions: [count and fraction related formatting functions](#), `jjcsformat_fraction()`

Examples

```
value <- c(1.65, 8.645)
fmt <- jjcsformat_xx("xx.x")
is.function(fmt)
fmt
format_value(value[1], fmt, round_type = "sas")
format_value(value[1], fmt, round_type = "iec")
if (is.function(fmt)) fmt(value[1])

fmt2 <- jjcsformat_xx("xx.x (xx.xxx)")
is.function(fmt2)
value <- c(1.65, 8.645)
format_value(value, fmt2, round_type = "sas")
format_value(value, fmt2, round_type = "iec")
# only possible when resulting format is a function
if (is.function(fmt2)) fmt2(value, round_type = "sas")

value <- c(1.65, NA)
format_value(value, fmt2, round_type = "iec", na_str = c("ne1", "ne2"))
if (is.function(fmt2)) fmt2(value, round_type = "iec", na_str = c("ne1", "ne2"))
my_pval_format <- jjcsformat_pval_fct(0.005)
my_pval_format(0.2802359)
my_pval_format(0.0048)
my_pval_format(0.00499)
my_pval_format(0.004999999)
my_pval_format(0.0051)
my_pval_format(0.0009)
my_pval_format(0.9991)

my_range_format <- jjcsformat_range_fct("xx.xx")
my_range_format(c(0.35235, 99.2342, 1, 0))
my_range_format(c(0.35235, 99.2342, 0, 1))
my_range_format(c(0.35235, 99.2342, 0, 0))
my_range_format(c(0.35235, 99.2342, 1, 1))
```

```
my_range_format <- jjcsformat_range_fct("xx.xx", censor_char = "*")
my_range_format(c(0.35235, 99.2342, 1, 1))
```

keep_non_null_rows	<i>Pruning Function to accommodate removal of completely NULL rows within a table</i>
--------------------	---

Description

Condition function on individual analysis rows. Flag as FALSE when all columns are NULL, as then the row should not be kept. To be utilized as a row_condition in function tern::keep_rows

Usage

```
keep_non_null_rows(tr)
```

Arguments

tr (TableTree)
The TableTree object to prune.

Value

A function that can be utilized as a row_condition in the tern::keep_rows function.

Examples

```
library(dplyr)

ADSL <- data.frame(
  USUBJID = c(
    "XXXXXX01", "XXXXXX02", "XXXXXX03", "XXXXXX04", "XXXXXX05",
    "XXXXXX06", "XXXXXX07", "XXXXXX08", "XXXXXX09", "XXXXXX10"
  ),
  TRT01P = c(
    "ARMA", "ARMB", "ARMA", "ARMB", "ARMB", "Placebo",
    "Placebo", "Placebo", "ARMA", "ARMB"
  ),
  AGE = c(34, 56, 75, 81, 45, 75, 48, 19, 32, 31),
  SAFFL = c("N", "N", "N", "N", "N", "N", "N", "N", "N", "N"),
  PKFL = c("N", "N", "N", "N", "N", "N", "N", "N", "N", "N")
)

ADSL <- ADSL |>
  mutate(TRT01P = as.factor(TRT01P))

create_blank_line <- function(x) {
  list(
    "Mean" = rcell(mean(x), format = "xx.x"),
    " " = rcell(NULL),
  )
}
```

```

      "Max" = rcell(max(x))
    )
  }

  lyt <- basic_table() |>
    split_cols_by("TRT01P") |>
    analyze("AGE", afun = create_blank_line)

  result <- build_table(lyt, ADSL)

  result
  result <- prune_table(result, prune_func = tern::keep_rows(keep_non_null_rows))

  result

```

leftside

Extract the left-hand side of a formula

Description

Extract the left-hand side of a formula

Usage

```
leftside(x)
```

Arguments

x (formula)
A two-sided formula, e.g., $y \sim x1 + x2$.

Value

(character(1)) The name of the left-hand side of the formula.

Examples

```
leftside(y ~ x)
```

listing_column_widths *Define Column Widths*

Description

def_colwidths uses heuristics to determine suitable column widths given a table or listing, and a font.

Usage

```
listing_column_widths(
  mpf,
  incl_header = TRUE,
  col_gap = 0.5,
  pg_width_ins = 8.88,
  fontspec = font_spec("Times", 8, 1.2),
  verbose = FALSE
)

def_colwidths(
  tt,
  fontspec,
  label_width_ins = 2,
  col_gap = ifelse(type == "Listing", 0.5, 3),
  type = tlg_type(tt)
)
```

Arguments

mpf	(listing_df or MatrixPrintForm derived thereof) The listing calculate column widths for.
incl_header	(logical(1)) Should the constraint to not break up individual words be extended to words in the column labels? Defaults to TRUE
col_gap	Column gap in spaces. Defaults to .5 for listings and 3 for tables.
pg_width_ins	(numeric(1)) Number of inches in width for <i>the portion of the page the listing will be printed to</i> . Defaults to 8.88 which corresponds to landscape orientation on a standard page after margins.
fontspec	Font specification
verbose	(logical(1)) Should additional information messages be displayed during the calculation of the column widths? Defaults to FALSE.
tt	input TableTree

label_width_ins	Label Width in Inches.
type	Type of the TableTree, used to determine column width calculation method.

Details

Listings are assumed to be rendered landscape on standard A1 paper, such that all columns are rendered on one page. Tables are allowed to be horizontally paginated, and column widths are determined based only on required word wrapping. See the Automatic Column Widths vignette for a detailed discussion of the algorithms used.

Value

- listing_column_widths: a vector of column widths suitable to use in tt_to_tlgtrtf and other exporters.
- def_colwidths: a vector of column widths (including the label row pseudo-column in the table case) suitable for use rendering tt in the specified font.

lsmeans_wide_first_split_fun_fct

Layout Generating Function for LS Means Wide Table Layouts

Description

Layout Generating Function for LS Means Wide Table Layouts

Usage

```
lsmeans_wide_first_split_fun_fct(include_variance)

lsmeans_wide_second_split_fun_fct(pval_sided, conf_level, include_pval)

lsmeans_wide_cfun(
  df,
  labelstr,
  .spl_context,
  variables,
  ref_level,
  treatment_levels,
  pval_sided = c("2", "1", "-1"),
  conf_level,
  formats
)

summarize_lsmeans_wide(
  lyt,
```

```

variables,
ref_level,
treatment_levels,
conf_level,
pval_sided = "2",
include_variance = TRUE,
include_pval = TRUE,
formats = list(lsmean = jjcsformat_xx("xx.x"), mse = jjcsformat_xx("xx.x"), df =
  jjcsformat_xx("xx."), lsmean_diff = jjcsformat_xx("xx.x"), se =
  jjcsformat_xx("xx.xx"), ci = jjcsformat_xx("(xx.xx, xx.xx)"), pval =
  jjcsformat_pval_fct(0))
)

```

Arguments

include_variance	(flag) whether to include the variance statistics (M.S. error and d.f.).
pval_sided	(string) either '2' for two-sided or '1' for 1-sided with greater than control or '-1' for 1-sided with smaller than control alternative hypothesis.
conf_level	(proportion) confidence level of the interval.
include_pval	(flag) whether to include the p-value column.
df	(data.frame) data set containing all analysis variables.
labelstr	(character) label of the level of the parent split currently being summarized (must be present as second argument in Content Row Functions). See rtables::summarize_row_groups() for more information.
.spl_context	(data.frame) gives information about ancestor split states that is passed by rtables.
variables	(list) see fit_ancova() for required variable specifications.
ref_level	(string) the reference level of the treatment arm variable.
treatment_levels	(character) the non-reference levels of the treatment arm variable.
formats	(list) including lsmean, mse, df, lsmean_diff, se, ci, pval formats.
lyt	(layout) empty layout, i.e. result of rtables::basic_table()

Details

The functions `lsmeans_wide_first_split_fun_fct()`, `lsmeans_wide_second_split_fun_fct()` and `lsmeans_wide_cfun()` are also exported and can be used directly when the layout is slightly different (e.g. contains additional subgroup row split).

Value

Modified layout.

Examples

```
variables <- list(
  response = "FEV1",
  covariates = c("RACE", "SEX"),
  arm = "ARMCD",
  id = "USUBJID",
  visit = "AVISIT"
)
fit <- fit_ancova(
  vars = variables,
  data = mrmr::fev_data,
  conf_level = 0.9,
  weights_emmeans = "equal"
)
anl <- broom::tidy(fit)
basic_table() |>
  summarize_lsmeans_wide(
    variables = variables,
    ref_level = fit$ref_level,
    treatment_levels = fit$treatment_levels,
    pval_sided = "2",
    conf_level = 0.8
  ) |>
  build_table(df = anl)
```

make_combo_splitfun *Split Function Helper*

Description

A function which aids the construction for users to create their own split function for combined columns.

Usage

```
make_combo_splitfun(nm, label = nm, levels = NULL, rm_other_facets = TRUE)
```

Arguments

nm	(character) Name/virtual 'value' for the new facet.
label	(character) Label for the new facet.
levels	(character or NULL) The levels to combine into the new facet, or NULL, indicating the facet should include all incoming data.
rm_other_facets	(logical) Should facets other than the newly created one be removed. Defaults to TRUE.

Value

Function usable directly as a split function.

Examples

```
aesevall_splf <- make_combo_splitfun(nm = "AESEV_ALL", label = "Any AE", levels = NULL)
```

make_multicomp_splfun *Make Multi-comparator Split Function*

Description

Create a custom splitting function suitable for creating risk difference columns against one or more comparators. This is used within `col_struct_w_risk_diffs`.

Usage

```
make_multicomp_splfun(
  colspan_trt_map,
  combo_levels_map = NULL,
  comp_level_map = NULL,
  .pre = list(),
  .post = list()
)

make_dflt_comp_map(df, spl_var, ref_lvls, combo_map)
```

Arguments

colspan_trt_map	(data.frame) A data.frame defining the active and non-active groups of treatment arms, including combination arms defined in <code>combo_levels_map</code> , as returned by <code>create_colspan_map()</code> .
-----------------	--

combo_levels_map	(data.frame or NULL) NULL (the default) or a data.frame indicating combination levels to added to some or all blocks of comparisons. See Details.
comp_level_map	(data.frame or NULL) A data.frame with columns active and comparator indicating which risk difference comparisons to include in the column structure, or NULL (the default), indicating all active vs non-active pairwise comparisons as defined colspan_trt_map treatment groupings.
.pre	(list) A list of additional preprocessing functions to be provided to make_split_fun. Defaults to list().
.post	(list) A list of additional post-processing functions to be provided to make_split_fun <i>after</i> those which provide this function's primary multi-comparator functionality. Defaults to list()
df	(data.frame) Data used to derive the available levels for the splitting variable spl_var and to construct the default comparison map when comp_level_map is not provided.
spl_var	(character(1)) Name of the splitting variable (typically the treatment variable) whose levels define active and comparator groups.
ref_lvls	(character) The level names to be treated as reference (control) groups. Comparisons will be formed against these levels.
combo_map	(data.frame) A combination levels map (i.e., the value passed to combo_levels_map) used to include virtual combination levels in the default comparison map.

Details

This split function is intended to create a set of risk difference or similar columns. As such it will automatically exclude the facet for each comparator level (e.g., Placebo vs Placebo) as determined by the last element of each element of comp_level_paths.

Further control of facets is provided by comp_level_map. If NULL (the default), all non-control/reference groups will be compared pairwise with all control/reference groups as defined by the grouping in colspan_trt_map.

If specified, comp_level_map must be a data.frame (including tbl_df) with three columns:

- active - (character) the value to be compared to a reference level,
- comparator - (character) the level that should be compared against, and
- active_is_combo - (logical) is the level specified in active a virtual combination level.
- comparator_is_combo - (logical) is the level specified in comparator a virtual combination level.

If a `data.frame` with only the active and comparator columns is given for `comp_level_map`, `active_is_combo` and `comparator_is_combo` are inferred from `colspan_trt_map`.

If any rows of `comp_level_map` have `active_is_combo == TRUE` or `comparator_is_combo`, the relevant values in those rows *must* also appear in `combo_levels_map` with the correct level for `comp_level` (or the `select_all_levels` sentinel value which indicates inclusion for all comparators).

If specified, `combo_levels_map` must be a `data.frame` (including `tbl_df`) with the following columns:

- `valname` - (character) The name(s) for the combination level(s),
- `label` - (character) the label(s) for the combination level(s),
- `levelcombo` - (list of character) the levels of the split variable to be combined, or `select_all_levels` for all levels,
- `exargs` - (list) the `extra_args` values for each combo level. If not present this will be assumed to be `list()` for all combo levels.
- `compare_against` - (list of character) Optional. The reference level(s) the combo level should be compared against, or `select_all_levels` for inclusion against all comparators.
- `is_control` - (logical) Optional. Is this combination level going to be used as a reference level (must appear as the last element in one of `comp_level_paths` if so).

When specifying `combo_levels_map` if the `compare_against` column is omitted, comparison against all reference levels will be performed for all combination levels. If `is_control` is omitted, it will be assumed as `FALSE` for all combination levels.

Order of combination levels when multiple are present for a single comparator, as well as their position relative to non-combination comparisons, is determined by row order in `combo_levels_map`.

Labels and names of comparison columns involving combination levels will be automatically computed in the form of "<combo level name/label> vs <ref group name>". Note currently ref group *name* is always used as it needs to be inferable from `colspan_trt_map`.

The comparator reference path is calculated based on `colspan_trt_map` and then added as `ref_path` to the `extra_args` associated with generated facet. As such, analysis (or content) functions used underneath a split using the generated split function must accept either `ref_path` or ...

Value

A split function suitable for use in both `split_rows_by` and `split_cols_by`.

For `make_dflt_comp_map()`, a `data.frame` with columns `active`, `comparator`, `active_is_combo`, and `comparator_is_combo`, listing the comparisons to keep and whether either side is a virtual combination level.

Note

It is not currently possible to use a virtual combination level as a comparator/reference group. If you need this functionality please contact the maintainers by filing an issue at <https://github.com/johnsonandjohnson/junco/issues>

See Also

Other `std_col_struct`: [grouped_cols_w_diffs\(\)](#), [shift_tbl_col_struct\(\)](#)

make_rbmi_cluster	Create a rbmi ready cluster
-------------------	-----------------------------

Description

[Deprecated] Use rbmi::make_rbmi_cluster() instead.

Usage

make_rbmi_cluster(cluster_or_cores = 1, objects = NULL, packages = NULL)

Arguments

- cluster_or_cores (integer or cluster object)
Number of parallel processes to use or an existing cluster to make use of
- objects (list)
A named list of objects to export into the sub-processes
- packages (character vector)
A character vector of libraries to load in the sub-processes

Value

See rbmi::make_rbmi_cluster().

odds_ratio	Odds ratio estimation
------------	-----------------------

Description

[Stable] A set of functions for Odds-Ratio (OR) calculation.

Usage

```
a_odds_ratio_j(  
  df,  
  .var,  
  .df_row,  
  ref_path,  
  .spl_context,  
  ...,  
  .stats = NULL,  
  .formats = NULL,  
  .labels = NULL,  
  .indent_mods = NULL
```

```

)

s_odds_ratio_j(
  df,
  .var,
  .ref_group,
  .in_ref_col,
  .df_row,
  variables = list(arm = NULL, strata = NULL),
  conf_level = 0.95,
  groups_list = NULL,
  na_if_no_events = TRUE,
  method = c("exact", "approximate", "efron", "breslow", "cmh")
)

```

Arguments

<code>df</code>	(data.frame) input data frame.
<code>.var</code>	(string) name of the response variable.
<code>.df_row</code>	(data.frame) data frame containing all rows.
<code>ref_path</code>	(character) path to the reference group.
<code>.spl_context</code>	(environment) split context environment.
<code>...</code>	Additional arguments passed to the statistics function.
<code>.stats</code>	(character) statistics to calculate.
<code>.formats</code>	(list) formats for the statistics.
<code>.labels</code>	(list) labels for the statistics.
<code>.indent_mods</code>	(list) indentation modifications for the statistics.
<code>.ref_group</code>	(data.frame) reference group data frame.
<code>.in_ref_col</code>	(logical) whether the current column is the reference column.
<code>variables</code>	(list) list with arm and strata variable names.
<code>conf_level</code>	(numeric) confidence level for the confidence interval.

groups_list (list)
list of groups for combination.

na_if_no_events (flag)
whether the point estimate should be NA if there are no events in one arm. The p-value and confidence interval will still be computed.

method (string)
whether to use the correct ('exact') calculation in the conditional likelihood or one of the approximations, or the CMH method. See [survival::clogit\(\)](#) for details.

Value

- `a_odds_ratio_j()` returns the corresponding list with formatted [rtables::CellValue\(\)](#).
- `s_odds_ratio_j()` returns a named list with the statistics `or_ci` (containing `est`, `lcl`, and `ucl`), `pval` and `n_tot`.

Functions

- `a_odds_ratio_j()`: Formatted analysis function which is used as `afun`. Note that the junco specific `ref_path` and `.spl_context` arguments are used for reference column information.
- `s_odds_ratio_j()`: Statistics function which estimates the odds ratio between a treatment and a control. A `variables` list with `arm` and `strata` variable names must be passed if a stratified analysis is required.

Note

The `a_odds_ratio_j()` and `s_odds_ratio_j()` functions have the `_j` suffix to distinguish them from [tern::a_odds_ratio\(\)](#) and [tern::s_odds_ratio\(\)](#), respectively. These functions differ as follows:

- Additional `method = 'cmh'` option is provided to calculate the Cochran-Mantel-Haenszel estimate.
- The p-value is returned as an additional statistic.

Once these updates are contributed back to `tern`, they can later be replaced by the `tern` versions.

Examples

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

a_odds_ratio_j(
  df = subset(dta, grp == "A"),
  .var = "rsp",
```

```

    ref_path = c("grp", "B"),
    .spl_context = data.frame(
      cur_col_split = I(list("grp")),
      cur_col_split_val = I(list(c(grp = "A"))),
      full_parent_df = I(list(dta))
    ),
    .df_row = dta
  )

l1 <- basic_table() |>
  split_cols_by(var = "grp") |>
  analyze(
    "rsp",
    afun = a_odds_ratio_j,
    show_labels = "hidden",
    extra_args = list(
      ref_path = c("grp", "B"),
      .stats = c("or_ci", "pval")
    )
  )

build_table(l1, df = dta)

l2 <- basic_table() |>
  split_cols_by(var = "grp") |>
  analyze(
    "rsp",
    afun = a_odds_ratio_j,
    show_labels = "hidden",
    extra_args = list(
      variables = list(arm = "grp", strata = "strata"),
      method = "cmh",
      ref_path = c("grp", "A"),
      .stats = c("or_ci", "pval")
    )
  )

build_table(l2, df = dta)
s_odds_ratio_j(
  df = subset(dta, grp == "A"),
  .var = "rsp",
  .ref_group = subset(dta, grp == "B"),
  .in_ref_col = FALSE,
  .df_row = dta
)

s_odds_ratio_j(
  df = subset(dta, grp == "A"),
  .var = "rsp",
  .ref_group = subset(dta, grp == "B"),
  .in_ref_col = FALSE,
  .df_row = dta,

```

```

    variables = list(arm = "grp", strata = "strata")
  )

  s_odds_ratio_j(
    df = subset(dta, grp == "A"),
    method = "cmh",
    .var = "rsp",
    .ref_group = subset(dta, grp == "B"),
    .in_ref_col = FALSE,
    .df_row = dta,
    variables = list(arm = "grp", strata = c("strata"))
  )

```

par_lapply

Parallelise Lapply

Description

[Deprecated] This internal helper is no longer exported. Use `rbmi::analyse()` directly.

Usage

```
par_lapply(cl, fun, x, ...)
```

Arguments

cl	(cluster object) Cluster created by parallel::makeCluster() or NULL
fun	(functions) Function to be run
x	(object) Object to be looped over
...	Extra arguments passed to fun

Value

list of results of calling fun on elements of x.

pool_rubin_scalar	<i>Pool a scalar estimate across imputations</i>
-------------------	--

Description

Applies Rubin's rules to combine scalar estimates and their within-imputation variance estimates.

Usage

```
pool_rubin_scalar(q, u)
```

Arguments

q	(numeric) Vector of scalar estimates, with one value for each imputed dataset.
u	(numeric) Vector of within-imputation variance estimates corresponding to q.

Details

For m imputations, the pooled estimate is the mean of q . The pooled variance combines the mean within-imputation variance with the between-imputation variance, using the usual $(1 + 1 / m)$ multiplier. The returned standard error is the square root of the pooled variance.

Value

A named list with the pooled estimate (est), standard error (se), variance (var), and number of imputations (m).

See Also

[pool_z_stat\(\)](#)

Examples

```
pool_rubin_scalar(  
  q = c(0.20, 0.25, 0.15),  
  u = c(0.01, 0.016, 0.012)  
)
```

pool_z_stat	<i>Pool z statistics across imputations</i>
-------------	---

Description

Combines z statistics from multiple imputed datasets and calculates a two-sided p-value for the pooled statistic.

Usage

```
pool_z_stat(z_stat_vals)
```

Arguments

`z_stat_vals` (numeric) Vector of z statistics, with one value for each imputed dataset.

Details

The within-imputation variance of a z statistic is approximately 1. The between-imputation variance is pooled using the same $(1 + 1 / m)$ multiplier as `pool_rubin_scalar()`. The returned p-value is calculated from the standard normal distribution.

Value

A named list with the pooled z statistic (z) and its two-sided p-value (p).

See Also

[pool_rubin_scalar\(\)](#)

Examples

```
pool_z_stat(c(1.1, 1.3, 1.2))
```

prepend_label_cell	<i>Prepend a Label Row to a RowsVerticalSection Object</i>
--------------------	--

Description

[Stable]

Prepends a label row to a RowsVerticalSection object.

RowsVerticalSection objects are returned by analysis functions used within the **rtables** framework and are typically created with `rtables::in_rows()`.

This function is typically used to introduce section headers (e.g., "Descriptive Statistics") in tabular reports and other reporting outputs.

Usage

```
prepend_label_cell(x, label = "", label_indent_mod = 0L)
```

Arguments

x (RowsVerticalSection)
Analysis result object.

label (character(1))
Label to insert as the first row.

label_indent_mod (integer(1))
Indentation modifier applied to the label row.

Value

A RowsVerticalSection object with a label row prepended.

Note

The `row_na_strs` and `row_footnotes` attributes of the RowsVerticalSection object are not preserved. Since an additional row is added, be aware of recycling rules; that is, the length of all attributes of `x` should be equal to `length(x)`.

Examples

```
rvs <- rtables::in_rows(
  Mean = rtables::rcell(5),
  Range = rtables::rcell(c(1, 8))
)

prepend_label_cell(
  rvs,
  "Descriptive Statistics",
  label_indent_mod = 1L
)
```

prop_diff

Proportion difference estimation

Description

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

Usage

```
a_proportion_diff_j(
  df,
  .var,
  ref_path,
  .spl_context,
  ...,
  conf_level = 0.95,
  label = paste0("% Difference (", f_conf_level(conf_level), ")"),
  .stats = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```

Arguments

df	(data.frame) input data frame.
.var	(string) name of the response variable.
ref_path	(character) path to the reference group.
.spl_context	(environment) split context environment.
...	Additional arguments passed to the statistics function.
conf_level	(numeric) confidence level for the confidence interval.
label	(string) label for the diff_est_ci statistic in a_proportion_diff_j() . Defaults to "% Difference (conf_level CI)". Can be overridden by the caller.
.stats	(character) statistics to calculate.
.formats	(list) formats for the statistics.
.labels	(list) labels for the statistics.
.indent_mods	(list) indentation modifications for the statistics.

Value

- `a_proportion_diff_j()` returns the corresponding list with formatted [rtables::CellValue\(\)](#).

Functions

- `a_proportion_diff_j()`: Formatted analysis function which is used as a fun in `estimate_proportion_diff()`.

Note

The `a_proportion_diff_j()` function has the `_j` suffix to distinguish it from `tern::a_proportion_diff()`. It is a thin wrapper around `tern::s_proportion_diff()` with the following additions:

- The `diff_est_ci` statistic (returned natively by `tern` since `tern#1523`) is relabelled with a shorter junco-style label ("`% Difference (conf_level CI)`").
- `ref_path` needs to be provided as extra argument to specify the control group column.

Examples

```
nex <- 100
dta <- data.frame(
  "rsp" = sample(c(TRUE, FALSE), nex, TRUE),
  "grp" = sample(c("A", "B"), nex, TRUE),
  "f1" = sample(c("a1", "a2"), nex, TRUE),
  "f2" = sample(c("x", "y", "z"), nex, TRUE),
  stringsAsFactors = TRUE
)

l <- basic_table() |>
  split_cols_by(var = "grp") |>
  analyze(
    vars = "rsp",
    afun = a_proportion_diff_j,
    show_labels = "hidden",
    na_str = tern::default_na_str(),
    extra_args = list(
      conf_level = 0.9,
      method = "ha",
      ref_path = c("grp", "B")
    )
  )

build_table(l, df = dta)
```

prop_diff_test

Difference test for two proportions

Description**[Stable]**

The analysis function `a_test_proportion_diff()` can be used to create a layout element to test the difference between two proportions. The primary analysis variable, `vars`, indicates whether a response has occurred for each record. See the `method` parameter for options of methods to use to calculate the p-value. Additionally, a stratification variable can be supplied via the `strata` element of the `variables` argument. The argument `alternative` specifies the direction of the alternative hypothesis.

Usage

```
a_test_proportion_diff(
  df,
  .var,
  ref_path,
  .spl_context,
  ...,
  .stats = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```

Arguments

<code>df</code>	(data.frame) data set containing all analysis variables.
<code>.var</code>	(string) single variable name that is passed by <code>rtables</code> when requested by a statistics function.
<code>ref_path</code>	(character) global reference group specification, see get_ref_info() .
<code>.spl_context</code>	(data.frame) gives information about ancestor split states that is passed by <code>rtables</code> .
<code>...</code>	Additional arguments passed to tern::s_test_proportion_diff() , including: <code>method</code> (string) one of <code>chisq</code>, <code>cmh</code>, <code>cmh_wh</code>, <code>fisher</code> or <code>schouten</code>; specifies the test used to calculate the p-value.
<code>.stats</code>	(character) statistics to select for the table.
<code>.formats</code>	(named character or list) formats for the statistics. See Details in <code>analyze_vars</code> for more information on the 'auto' setting.
<code>.labels</code>	(named character) labels for the statistics (without indent).
<code>.indent_mods</code>	(named integer) indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.

Value

- `a_test_proportion_diff()` returns the corresponding list with formatted [rtables::CellValue\(\)](#).

Functions

- `a_test_proportion_diff()`: Formatted analysis function which is used as `afun`

Note

This function has been forked from the tern package. Additional features are:

- Additional ref_path argument for flexible reference column path specification.

Examples

```
dta <- data.frame(
  rsp = sample(c(TRUE, FALSE), 100, TRUE),
  grp = factor(rep(c("A", "B"), each = 50)),
  strata = factor(rep(c("V", "W", "X", "Y", "Z"), each = 20))
)

l <- basic_table() |>
  split_cols_by(var = "grp") |>
  analyze(
    vars = "rsp",
    afun = a_test_proportion_diff,
    show_labels = "hidden",
    extra_args = list(
      method = "cmh",
      variables = list(strata = "strata"),
      ref_path = c("grp", "B")
    )
  )

build_table(l, df = dta)
```

prop_post_fun

Split Function for Proportion Analysis Columns

Description

Here we just split into 3 columns n, % and Cum %.

Usage

```
prop_post_fun(ret, spl, fullldf, .spl_context)
```

```
prop_split_fun(df, spl, vals = NULL, labels = NULL, trim = FALSE, .spl_context)
```

Arguments

ret	(list) return value from the previous split function.
spl	(list) split information.

fulldf	(data.frame) full data frame.
.spl_context	(environment) split context environment.
df	A data frame that contains all analysis variables.
vals	A character vector that contains values to use for the split.
labels	A character vector that contains labels for the statistics (without indent).
trim	A single logical that indicates whether to trim the values.

Value

a split function for use in [rtables::split_rows_by](#).

See Also

[rtables::make_split_fun\(\)](#) describing the requirements for this kind of post-processing function.

prop_ratio_cmh	<i>Relative Risk CMH Statistic</i>
----------------	------------------------------------

Description

Calculates the relative risk which is defined as the ratio between the response rates between the experimental treatment group and the control treatment group, adjusted for stratification factors by applying Cochran-Mantel-Haenszel (CMH) weights.

Usage

```
prop_ratio_cmh(rsp, grp, strata, conf_level = 0.95)
```

Arguments

rsp	(logical) whether each subject is a responder or not.
grp	(factor) defining the groups.
strata	(factor) variable with one level per stratum and same length as rsp.
conf_level	(proportion) confidence level of the interval.

Value

A list with elements `rel_risk_ci` and `pval`.

Examples

```
set.seed(2)
rsp <- sample(c(TRUE, FALSE), 100, TRUE)
grp <- sample(c("Placebo", "Treatment"), 100, TRUE)
grp <- factor(grp, levels = c("Placebo", "Treatment"))
strata_data <- data.frame(
  "f1" = sample(c("a", "b"), 100, TRUE),
  "f2" = sample(c("x", "y", "z"), 100, TRUE),
  stringsAsFactors = TRUE
)

prop_ratio_cmh(
  rsp = rsp, grp = grp, strata = interaction(strata_data),
  conf_level = 0.90
)
```

prop_table_afun	<i>Formatted Analysis Function for Proportion Analysis</i>
-----------------	--

Description

This function applies to a factor x when a column split was prepared with [prop_split_fun\(\)](#) before.

Usage

```
prop_table_afun(x, .spl_context, formats, add_total_level = FALSE)
```

Arguments

- x (factor)
factor variable to analyze.
- .spl_context (environment)
split context environment.
- formats (list)
formats for the statistics.
- add_total_level (flag)
whether to add a total level.

Details

In the column named n, the counts of the categories as well as an optional Total count will be shown. In the column named percent, the percentages of the categories will be shown, with an optional blank entry for Total. In the column named cum_percent, the cumulative percentages will be shown instead.

Value

A VerticalRowsSection as returned by [rtables::in_rows](#).

proportion_diff_mf	<i>Proportion Difference with Method Selection Based on the Mantel-Fleiss Criterion</i>
--------------------	---

Description**[Stable]**

Performs an adaptive analysis of the difference in response proportions between a non-reference group and a reference group for stratified data, with the method selected based on the Mantel-Fleiss (MF) criterion.

When the Mantel-Fleiss criterion is satisfied, a Cochran-Mantel-Haenszel (CMH) analysis is performed using the method specified by `mf_method`. When the criterion is not satisfied, an unconditional exact method is used instead.

Usage

```
a_proportion_diff_mf(
  df,
  .var,
  .in_ref_col = NULL,
  .ref_group = NULL,
  val = TRUE,
  variables = list(strata = NULL),
  na.rm = FALSE,
  conf_level = 0.95,
  mf_method = c("cmh", "cmh_sato", "cmh_mn"),
  mf_threshold = NULL,
  ...,
  .stats = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL,
  exact_footnote = rtables::RefFootnote(note = tern::d_proportion_diff(method =
    "uncond_exact_diff", method_only = TRUE), index = 1L, symbol = "+")
)

s_proportion_diff_mf(
  df,
  .var,
  .in_ref_col = NULL,
  .ref_group = NULL,
  val = TRUE,
  variables = list(strata = NULL),
```

```

na.rm = FALSE,
conf_level = 0.95,
mf_method = c("cmh", "cmh_sato", "cmh_mn"),
mf_threshold = NULL
)

```

Arguments

<code>df</code>	(data.frame) The data containing all analysis variables for the non-reference group.
<code>.var</code>	(character(1)) The column name in <code>df</code> (and, if supplied, <code>.ref_group</code>) specifying the response variable. The response is converted to a logical vector by comparing its values with <code>val</code> . <code>df[[var]]</code> (and <code>.ref_group[[var]]</code> , if supplied) must be an atomic vector as defined by <code>checkmate::check_atomic_vector()</code> , of one of the following types: logical, integer, numeric, or character.
<code>.in_ref_col</code>	(logical(1) or NULL) Indicates whether the function is being evaluated for the reference column. Set to TRUE for the reference column and to FALSE or NULL otherwise. For the statistics function, TRUE or NULL results in empty statistics, while for the analysis function, the result is a RowsVerticalSection with a NULL value for all selected statistics.
<code>.ref_group</code>	(data.frame or NULL) The data corresponding to the reference group.
<code>val</code>	(logical(1) or integer(1) or numeric(1) or character(1)) The value used to identify the response of interest. An observation is considered a response of interest when <code>df[[.var]] == val</code> (and, if the the reference group is supplied, <code>.ref_group[[.var]] == val</code>). If <code>df[[.var]]</code> is a factor, <code>val</code> must be a character value matching one of its levels. Otherwise, <code>val</code> must have the same class as <code>df[[.var]]</code> .
<code>variables</code>	(list or NULL) Optional strata variables specified as <code>list(strata = <strata_vars>)</code> , where <code><strata_vars></code> is a character vector containing column names in <code>df</code> (and, if supplied, <code>.ref_group</code>). All specified strata variables must be factors.
<code>na.rm</code>	(logical(1)) Whether to remove incomplete rows from <code>df</code> (and, if supplied, <code>.ref_group</code>) before constructing the response, group, and strata vectors. Completeness is assessed only for <code>.var</code> and the strata columns (if supplied). If <code>na.rm = TRUE</code> , rows containing missing values in any of these columns are removed. If <code>na.rm = FALSE</code> , the function fails if any of these columns contain missing values, rather than returning results containing NA values. This argument is passed unchanged to the <code>complete_cases</code> argument of tern::h_prepare_rsp_table() .
<code>conf_level</code>	(proportion) confidence level of the interval.
<code>mf_method</code>	(character(1)) The method used for estimating the difference in response proportions and its

	confidence interval when strata are provided and the Mantel-Fleiss criterion is satisfied. Available choices are: "cmh", "cmh_sato", and "cmh_mn". See Details for more information.
mf_threshold	(numeric(1) or NULL) Threshold for the Mantel-Fleiss criterion. If NULL, the default from <code>tern::mantel_fleiss_crit()</code> is used.
...	additional arguments for the lower level functions.
.stats	(character) Statistics to select. Available choices are "diff", "diff_ci", and "diff_est_ci". This parameter affects only the formatted analysis function and has no effect on the statistics function.
.formats	(named character or list) Formats for the statistics.
.labels	(named character) labels for the statistics (without indent).
.indent_mods	(named integer) indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.
exact_footnote	(character(1) or RefFootnote) Footnote used to indicate that the unconditional exact method was actually used for the analysis.

Details

The data are prepared by `tern::h_prepare_rsp_table()`, which constructs the response vector, group indicator, optional strata variable, and the corresponding 2 x 2 contingency table(s). When multiple strata variables are supplied, the cross-combinations of their levels define the analysis strata.

The response variable is converted to a binary response according to `val`. The analysis compares the response proportion in the non-reference group with that in the reference group. The difference is calculated as the non-reference group minus the reference group.

When strata are provided and the Mantel-Fleiss criterion is satisfied, the selected CMH method `tern::prop_diff_cmh()` is used to obtain the point estimate and confidence interval for the difference in response proportions. The available Cochran-Mantel-Haenszel-type methods for estimating the difference in response proportions and its confidence interval are:

- "cmh": CMH method.
- "cmh_sato": CMH method using the Sato variance estimator.
- "cmh_mn": CMH method using the Miettinen-Nurminen method.

See `tern::prop_diff_cmh()` for more details and references.

When strata are not specified, or when strata are specified but the Mantel-Fleiss criterion is not satisfied or cannot be evaluated (e.g., when all strata contain no observed values), the unconditional exact method `tern::prop_diff_uncond_exact()` is used to obtain the point estimate and confidence interval.

The Mantel-Fleiss criterion is evaluated using `tern::mantel_fleiss_crit()`.

The statistics function `s_proportion_diff_mf()` follows the usage and output of `tern::s_proportion_diff()` but removes the method argument and selects internally between a Cochran-Mantel-Haenszel-type method and an unconditional exact method based on the Mantel-Fleiss criterion.

Value

- `a_proportion_diff_mf()` returns the corresponding `RowsVerticalSection` object with formatted results.
- `s_proportion_diff_mf()` returns a list containing:
 - `diff` (`numeric(1)` **or** `numeric(0)`) Point estimate of the difference in response proportions, or `numeric(0)` when the function is evaluated for the reference column (`.in_ref_col` is `NULL` or `.in_ref_col` is `TRUE`).
 - `diff_ci` (`numeric(2)` **or** `numeric(0)`) Confidence interval for the difference in response proportions, or `numeric(0)` when the function is evaluated for the reference column (`.in_ref_col` is `NULL` or `.in_ref_col` is `TRUE`).
 - `diff_est_ci` (`numeric(3)` **or** `numeric(0)`) Point estimate and confidence interval for the difference in response proportions, or `numeric(0)` when the function is evaluated for the reference column (`.in_ref_col` is `NULL` or `.in_ref_col` is `TRUE`).
 - `executed_method` (`character(1)`) Name of the method actually used for the analysis. Takes the value `NA_character_` when the function is evaluated for the reference column (`.in_ref_col` is `NULL` or `.in_ref_col` is `TRUE`). Otherwise, identifies the method used: `"cmh"`, `"cmh_sato"`, `"cmh_mn"`, or `"uncond_exact_diff"`. Formatted analysis functions can use this value to determine whether an appropriate footnote should be added.

Functions

- `a_proportion_diff_mf()`: Formatted analysis function used as `afun` for the difference in response proportions with method selection based on the Mantel-Fleiss criterion.
- `s_proportion_diff_mf()`: Statistics function estimating the difference in response proportions and its confidence interval. When strata variables are provided and the Mantel-Fleiss criterion is satisfied, the CMH method `tern::prop_diff_cmh()` is used, with the `diff_se` argument determined by `mf_method`. Otherwise, i.e., when strata variables are not provided, or when strata variables are provided but the Mantel-Fleiss criterion is not satisfied or cannot be evaluated, the unconditional exact method `tern::prop_diff_uncond_exact()` is used.

Author(s)

WW

See Also

`cond_proportion_j()`

Examples

```
n <- 100 # Number of observations.
set.seed(123)
dta <- data.frame(
  "rsp" = sample(c("Y", "N"), n, TRUE),
  "grp" = c(
    sample(c("A", "Placebo"), round(n - 0.1 * n), TRUE),
    rep("C", round(0.1 * n))
  ),
  "f1" = sample(c("a1", "a2"), n, TRUE),
  "f2" = sample(c("x", "y", "z"), n, TRUE),
  stringsAsFactors = TRUE
)

lyt <- basic_table() |>
  split_cols_by(var = "grp", ref_group = "Placebo") |>
  analyze(
    vars = "rsp",
    afun = a_proportion_diff_mf,
    extra_args = list(
      val = "Y",
      variables = list(strata = c("f1", "f2")),
      conf_level = 0.90,
      mf_method = "cmh_mn",
      .stats = "diff_est_ci"
    )
  )

build_table(lyt, df = dta)

prop_d <- s_proportion_diff_mf(
  df = subset(dta, grp == "A"),
  .var = "rsp",
  .ref_group = subset(dta, grp == "Placebo"),
  .in_ref_col = FALSE,
  val = "Y",
  variables = list(strata = c("f1", "f2")),
  conf_level = 0.90,
  mf_method = "cmh_mn"
)
prop_d
```

Description

This function takes multiple imputed datasets (as generated by the `impute()` function from the `rbmi` package) and runs an analysis function on each of them.

[Deprecated] Use `rbmi::analyse()` instead.

Usage

```
rbmi_analyse(
  imputations,
  fun = rbmi_ancova,
  delta = NULL,
  ...,
  cluster_or_cores = 1,
  .validate = TRUE
)
```

Arguments

<code>imputations</code>	An imputations object as created by the <code>impute()</code> function from the <code>rbmi</code> package.
<code>fun</code>	An analysis function to be applied to each imputed dataset. See details.
<code>delta</code>	A <code>data.frame</code> containing the delta transformation to be applied to the imputed datasets prior to running <code>fun</code> . See details.
<code>...</code>	Additional arguments passed onto <code>fun</code> .
<code>cluster_or_cores</code>	(numeric or cluster object) The number of parallel processes to use when running this function. Can also be a cluster object created by make_rbmi_cluster() . See the parallelisation section below.
<code>.validate</code>	(logical) Should imputations be checked to ensure it conforms to the required format (default = TRUE) ? Can gain a small performance increase if this is set to FALSE when analysing a large number of samples.

Details

This function works by performing the following steps:

1. Extract a dataset from the `imputations` object.
2. Apply any delta adjustments as specified by the `delta` argument.
3. Run the analysis function `fun` on the dataset.
4. Repeat steps 1-3 across all of the datasets inside the `imputations` object.
5. Collect and return all of the analysis results.

The analysis function `fun` must take a `data.frame` as its first argument. All other options to [rbmi_analyse\(\)](#) are passed onto `fun`

Value

An analysis object, as defined by `rbmi`, representing the desired analysis applied to each of the imputed datasets in `imputations`.

Description

Performs an analysis of covariance between two groups returning the estimated "treatment effect" (i.e. the contrast between the two treatment groups) and the least square means estimates in each group.

Usage

```
rbmi_ancova(
  data,
  vars,
  visits = NULL,
  weights = c("counterfactual", "equal", "proportional_em", "proportional")
)
```

Arguments

<code>data</code>	A <code>data.frame</code> containing the data to be used in the model.
<code>vars</code>	A <code>vars</code> object as generated by the <code>set_vars()</code> function from the <code>rbmi</code> package. Only the <code>group</code> , <code>visit</code> , <code>outcome</code> and <code>covariates</code> elements are required. See details.
<code>visits</code>	An optional character vector specifying which visits to fit the ancova model at. If <code>NULL</code> , a separate ancova model will be fit to the outcomes for each visit (as determined by <code>unique(data[[vars\$visit]])</code>). See details.
<code>weights</code>	Character, either "counterfactual" (default), "equal", "proportional_em" or "proportional". Specifies the weighting strategy to be used when calculating the <code>lsmeans</code> . See the weighting section for more details.

Details

The function works as follows:

1. Select the first value from `visits`.
2. Subset the data to only the observations that occurred on this visit.
3. Fit a linear model as `vars$outcome ~ vars$group + vars$covariates`.
4. Extract the "treatment effect" & least square means for each treatment group.
5. Repeat points 2-3 for all other values in `visits`.

If no value for `visits` is provided then it will be set to `unique(data[[vars$visit]])`.

In order to meet the formatting standards set by `rbmi_analyse()` the results will be collapsed into a single list suffixed by the visit name, e.g.:

```
list(
  var_visit_1 = list(est = ...),
  trt_B_visit_1 = list(est = ...),
  lsm_A_visit_1 = list(est = ...),
  lsm_B_visit_1 = list(est = ...),
  var_visit_2 = list(est = ...),
  trt_B_visit_2 = list(est = ...),
  lsm_A_visit_2 = list(est = ...),
  lsm_B_visit_2 = list(est = ...),
  ...
)
```

Please note that "trt" refers to the treatment effects, and "lsm" refers to the least square mean results. In the above example `vars$group` has two factor levels A and B. The new "var" refers to the model estimated variance of the residuals.

If you want to include interaction terms in your model this can be done by providing them to the `covariates` argument of the `set_vars()` function from the `rbmi` package e.g. `set_vars(covariates = c("sex*age"))`.

Value

a list of variance (`var_*`), treatment effect (`trt_*`), and least square mean (`lsm_*`) estimates for each visit, organized as described in Details above.

Note

These functions have the `rbmi_` prefix to distinguish them from the corresponding `rbmi` package functions, from which they were copied from. Additional features here include:

- Support for more than two treatment groups.
- Variance estimates are returned.

See Also

[rbmi_analyse\(\)](#)

[stats::lm\(\)](#)

The `set_vars()` function from the `rbmi` package

rbmi_ancova_single

Implements an Analysis of Covariance (ANCOVA)

Description

Performance analysis of covariance. See [rbmi_ancova\(\)](#) for full details.

Usage

```
rbmi_ancova_single(  
  data,  
  outcome,  
  group,  
  covariates,  
  weights = c("counterfactual", "equal", "proportional_em", "proportional")  
)
```

Arguments

<code>data</code>	A <code>data.frame</code> containing the data to be used in the model.
<code>outcome</code>	string, the name of the outcome variable in <code>data</code> .
<code>group</code>	string, the name of the group variable in <code>data</code> .
<code>covariates</code>	character vector containing the name of any additional covariates to be included in the model as well as any interaction terms.
<code>weights</code>	Character, either "counterfactual" (default), "equal", "proportional_em" or "proportional". Specifies the weighting strategy to be used when calculating the lsmeans. See the weighting section for more details.

Details

- `group` must be a factor variable with only 2 levels.
- `outcome` must be a continuous numeric variable.

Value

a list containing `var` with variance estimates as well as `trt_*` and `lsm_*` entries. See [rbmi_ancova\(\)](#) for full details.

See Also

[rbmi_ancova\(\)](#)

Examples

```
iris2 <- iris[iris$Species %in% c("versicolor", "virginica"), ]  
iris2$Species <- factor(iris2$Species)  
rbmi_ancova_single(iris2, "Sepal.Length", "Species", c("Petal.Length * Petal.Width"))
```

rbmi_mmrn

*MMRM Analysis for Imputed Datasets***Description**

Performs an MMRM for two or more groups returning the estimated 'treatment effect' (i.e. the contrast between treatment groups and the control group) and the least square means estimates in each group.

Usage

```
rbmi_mmrn(
  data,
  vars,
  cov_struct = c("us", "toep", "cs", "ar1"),
  visits = NULL,
  weights = c("counterfactual", "equal"),
  ...
)
```

Arguments

data	(data.frame) containing the data to be used in the model.
vars	(vars) list as generated by the <code>set_vars()</code> function from the <code>rbmi</code> package. Only the <code>subjid</code> , <code>group</code> , <code>visit</code> , <code>outcome</code> and <code>covariates</code> elements are required. See details.
cov_struct	(string) the covariance structure to use. Note that the same covariance structure is assumed for all treatment groups.
visits	(NULL or character) An optional character vector specifying which visits to fit the MMRM at. If NULL, the MMRM model will be fit to the whole dataset.
weights	(string) the weighting strategy to be used when calculating the least square means, either 'counterfactual' or 'equal'.
...	additional arguments passed to <code>mmrm::mmrm()</code> , in particular <code>method</code> and <code>vcov</code> to control the degrees of freedom and variance-covariance adjustment methods as well as <code>reml</code> decide between REML and ML estimation.

Details

The function works as follows:

1. Optionally select the subset of the data corresponding to 'visits.

2. Fit an MMRM as `vars$outcome ~ vars$group + vars$visit + vars$covariates` with the specified covariance structure for visits within subjects.
3. Extract the 'treatment effect' & least square means for each treatment group vs the control group.

In order to meet the formatting standards set by the `analyse()` function from the `rbmi` package, the results will be collapsed into a single list suffixed by the visit name, e.g.:

```
list(
  var_B_visit_1 = list(est = ...),
  trt_B_visit_1 = list(est = ...),
  lsm_A_visit_1 = list(est = ...),
  lsm_B_visit_1 = list(est = ...),
  var_B_visit_2 = list(est = ...),
  trt_B_visit_2 = list(est = ...),
  lsm_A_visit_2 = list(est = ...),
  lsm_B_visit_2 = list(est = ...),
  ...
)
```

Please note that 'trt' refers to the treatment effects, and 'lsm' refers to the least square mean results. In the above example `vars$group` has two factor levels A and B. The new 'var' refers to the model estimated variance of the residuals at the given visit, together with the degrees of freedom (which is treatment group specific).

If you want to include additional interaction terms in your model this can be done by providing them to the `covariates` argument of the `set_vars()` function from the `rbmi` package e.g. `set_vars(covariates = c('sex*age'))`.

Value

a list of variance (`var_*`), treatment effect (`trt_*`), and least square mean (`lsm_*`) estimates for each visit, organized as described in Details above.

Note

The group and visit interaction `group:visit` is not included by default in the model, therefore please add that to `covariates` manually if you want to include it. This will make sense in most cases.

See Also

`rbmi_analyse()`

`mmrm::mmrm()`

The `set_vars()` function from the `rbmi` package

rbmi_mmrn_single_info	<i>Extract Single Visit Information from a Fitted MMRM for Multiple Imputation Analysis</i>
-----------------------	---

Description

Extracts relevant estimates from a given fitted MMRM. See [rbmi_mmrn\(\)](#) for full details.

Usage

```
rbmi_mmrn_single_info(fit, visit_level, visit, group, weights)
```

Arguments

fit	(mmrn) the fitted MMRM.
visit_level	(string) the visit level to extract information for.
visit	(string) the name of the visit variable.
group	(string) the name of the group variable.
weights	(string) the weighting strategy to be used when calculating the least square means, either 'counterfactual' or 'equal'.

Value

a list with trt_*, var_* and lsm_* elements. See [rbmi_mmrn](#) for full details.

See Also

[rbmi_mmrn\(\)](#)

rbmi_pool	<i>Pool analysis results obtained from the imputed datasets</i>
-----------	---

Description

Pool analysis results obtained from the imputed datasets

Usage

```
rbmi_pool(
  results,
  conf.level = 0.95,
  alternative = c("two.sided", "less", "greater"),
  type = c("percentile", "normal")
)
```

Arguments

<code>results</code>	an analysis object created by <code>analyse()</code> .
<code>conf.level</code>	confidence level of the returned confidence interval. Must be a single number between 0 and 1. Default is 0.95.
<code>alternative</code>	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".
<code>type</code>	a character string of either "percentile" (default) or "normal". Determines what method should be used to calculate the bootstrap confidence intervals. See details. Only used if <code>method_condmean(type = "bootstrap")</code> was specified in the original call to <code>draws()</code> .

Details

This has been forked from the `rbmi` package, mainly to support in addition the pooling of variance estimates. See `pool()` for more details.

Value

A list of class `pool`.

`real_add_overall_facet`

Add Overall Facet

Description

A function to help add an overall facet to your tables.

Usage

```
real_add_overall_facet(name, label)
```

Arguments

<code>name</code>	(character) Name/virtual 'value' for the new facet.
<code>label</code>	(character) Label for the new facet.

Value

Function usable directly as a split function.

Note

Current add_overall_facet is bugged. Can be used directly after it's fixed <https://github.com/pharmaverse/rtables/issues/768>

Examples

```
splfun <- make_split_fun(post = list(real_add_overall_facet("Total", "Total")))
```

remove_col_count	<i>Removal of Unwanted Column Counts</i>
------------------	--

Description

Remove the N=xx column headers for specified span_label_var columns - default is 'rrisk_header'.

Usage

```
remove_col_count(obj, span_label_var = "rrisk_header")
```

Arguments

obj	(TableTree) TableTree object.
span_label_var	(character) The spanning header text variable value for which column headers will be removed from.

Details

This works for only the lowest level of column splitting (since colcounts is used).

Value

TableTree object with column counts in specified columns removed.

remove_rows	<i>Pruning function to remove specific rows of a table regardless of counts</i>
-------------	---

Description

This function will remove all rows of a table based on the row text provided by the user.

Usage

```
remove_rows(removerowtext = NULL, reg_expr = FALSE)
```

Arguments

removerowtext	(character)	Define a text string for which any row with row text will be removed.
reg_expr	(logical)	Apply removerowtext as a regular expression (grepl with fixed = TRUE)

Value

Function that can be utilized as pruning function in prune_table.

Examples

```
ADSL <- data.frame(
  USUBJID = c(
    "XXXXX01", "XXXXX02", "XXXXX03", "XXXXX04", "XXXXX05",
    "XXXXX06", "XXXXX07", "XXXXX08", "XXXXX09", "XXXXX10"
  ),
  TRT01P = c(
    "ARMA", "ARMB", "ARMA", "ARMB", "ARMB", "Placebo",
    "Placebo", "Placebo", "ARMA", "ARMB"
  ),
  Category = c(
    "Cat 1", "Cat 2", "Cat 1", "Unknown", "Cat 2",
    "Cat 1", "Unknown", "Cat 1", "Cat 2", "Cat 1"
  ),
  SAFFL = c("N", "N", "N", "N", "N", "N", "N", "N", "N", "N"),
  PKFL = c("N", "N", "N", "N", "N", "N", "N", "N", "N", "N")
)

ADSL <- ADSL |>
  dplyr::mutate(TRT01P = as.factor(TRT01P))

lyt <- basic_table() |>
  split_cols_by("TRT01P") |>
  analyze(
    "Category",
```

```

      afun = a_freq_j,
      extra_args = list(.stats = "count_unique_fraction")
    )

result <- build_table(lyt, ADSL)

result

result <- prune_table(result, prune_func = remove_rows(removerowtext = "Unknown"))

result

```

resp_multiple_imputation

Calculate results across binary-imputation scenarios

Description

Runs `h_impute_analyze_resp()` for each paired control and treatment response probability, pools estimates across imputations using Rubin's rules, and pools Wilson-Hilferty z statistics to obtain a p-value. A scenario where both probabilities are zero or one is deterministic and is therefore analyzed once rather than repeatedly imputed.

Usage

```

resp_multiple_imputation(
  dat,
  p_ctrl,
  p_trt,
  trtvar,
  ctrlab,
  trtlab,
  respvar,
  stratvar,
  n_imputations,
  pvalcat
)

```

Arguments

<code>dat</code>	(data.frame) Analysis data containing treatment, response, and stratification variables.
<code>p_ctrl</code>	(numeric) Control-group response probabilities, one for each scenario.
<code>p_trt</code>	(numeric) Treatment-group response probabilities, paired with <code>p_ctrl</code> by position.
<code>trtvar</code>	(string) Name of the treatment variable in <code>dat</code> .
<code>ctrlab</code>	(string) Value of <code>trtvar</code> identifying the control group.

trtlab	(string) Value of trtvar identifying the treatment group.
respvar	(string) Name of a logical binary response variable in dat.
stratvar	(character) Names of one or more stratification variables in dat.
n_imputations	(count) Number of imputations for non-deterministic scenarios.
pvalcat	(named list) P-value categories passed to <code>categorize_pval()</code> .

Value

A tibble with one row per scenario. `rd` and `rd_se` are the pooled risk-difference estimate and standard error; `p_value` and `p_cat` are the pooled p-value and its category. `marker_flag` and `effect_label` contain "*" when a deterministic scenario or a single-imputation p-value fallback was used.

Examples

```
dat <- data.frame(
  arm = rep(c("Control", "Treatment"), each = 4),
  response = c(TRUE, FALSE, NA, TRUE, TRUE, FALSE, NA, FALSE),
  strata = rep(c("A", "B"), 4)
)
pvalcat <- list("<0.05" = c(0, 0.05), ">=0.05" = c(0.05, 1))

set.seed(123)
resp_multiple_imputation(
  dat,
  p_ctrl = c(0, 0.25),
  p_trt = c(1, 0.75),
  trtvar = "arm",
  ctrlab = "Control",
  trtlab = "Treatment",
  respvar = "response",
  stratvar = "strata",
  n_imputations = 5,
  pvalcat = pvalcat
)
```

resp01_a_comp_stat_factor

Formatted Analysis Function for Comparative Statistic in Response Tables (RESP01)

Description

This function applies to a factor column called `.var` from `df`.

Usage

```
resp01_a_comp_stat_factor(df, .var, include, ...)
```

Arguments

<code>df</code>	(data.frame) data set containing all analysis variables.
<code>.var</code>	(string) single variable name that is passed by <code>rtables</code> when requested by a statistics function.
<code>include</code>	(character) for which factor levels to include the comparison statistic results.
<code>...</code>	see <code>resp01_a_comp_stat_logical()</code> for additional required arguments.

Value

The formatted result as `rtables::rcell()`.

Examples

```
dm <- droplevels(subset(formatters::DM, SEX %in% c("F", "M")))

resp01_a_comp_stat_factor(
  dm,
  .var = "COUNTRY",
  conf_level = 0.9,
  include = c("USA", "CHN"),
  arm = "SEX",
  strata = "RACE",
  stat = "comp_stat_ci",
  methods = list(comp_stat_ci = "or_cmh"),
  formats = list(
    comp_stat_ci = jjcsformat_xx("xx.xx (xx.xx - xx.xx)"),
    pval = jjcsformat_pval_fct(0.05)
  )
)
```

`resp01_a_comp_stat_logical`

Formatted Analysis Function for Comparative Statistic in Response Tables (RESP01)

Description

This function applies to a logical column called `.var` from `df`. The response proportion is compared between the treatment arms identified by column `arm`.

Usage

```
resp01_a_comp_stat_logical(
  df,
  .var,
  conf_level,
  include,
  arm,
  strata,
  formats,
  methods,
  stat = c("comp_stat_ci", "pval")
)
```

Arguments

<code>df</code>	(<code>data.frame</code>) data set containing all analysis variables.
<code>.var</code>	(<code>string</code>) single variable name that is passed by <code>rtables</code> when requested by a statistics function.
<code>conf_level</code>	(<code>proportion</code>) confidence level of the interval.
<code>include</code>	(<code>flag</code>) whether to include the results for this variable.
<code>arm</code>	(<code>string</code>) column name in the data frame that identifies the treatment arms.
<code>strata</code>	(<code>character</code> or <code>NULL</code>) variable names indicating stratification factors.
<code>formats</code>	(<code>list</code>) containing formats for <code>comp_stat_ci</code> and <code>pval</code> .
<code>methods</code>	(<code>list</code>) containing methods for comparative statistics. The element <code>comp_stat_ci</code> can be <code>'rr'</code> (relative risk), <code>'or_cmh'</code> (odds ratio with CMH estimation and p-value) or <code>'or_logistic'</code> (odds ratio estimated by conditional or standard logistic regression). The element <code>pval</code> can be <code>'fisher'</code> (Fisher's exact test) or <code>'chisq'</code> (chi-square test), only used when using unstratified analyses with <code>'or_logistic'</code> .
<code>stat</code>	(<code>string</code>) the statistic to return, either <code>comp_stat_ci</code> or <code>pval</code> .

Value

The formatted result as `rtables::rcell()`.

See Also

`resp01_a_comp_stat_factor()` for the factor equivalent.

Examples

```
dm <- droplevels(subset(formatters::DM, SEX %in% c("F", "M")))
dm$RESP <- as.logical(sample(c(TRUE, FALSE), size = nrow(DM), replace = TRUE))

resp01_a_comp_stat_logical(
  dm,
  .var = "RESP",
  conf_level = 0.9,
  include = TRUE,
  arm = "SEX",
  strata = "RACE",
  stat = "comp_stat_ci",
  methods = list(comp_stat_ci = "or_cmh"),
  formats = list(
    comp_stat_ci = jjcsformat_xx("xx.xx (xx.xx - xx.xx)"),
    pval = jjcsformat_pval_fct(0.05)
  )
)
```

resp01_acfun

Formatted Analysis and Content Summary Function for Response Tables (RESP01)

Description

This function applies to both factor and logical columns called `.var` from `df`. Depending on the position in the split, it returns the right formatted results for the RESP01 and related layouts.

Usage

```
resp01_acfun(
  df,
  labelstr = NULL,
  label = NULL,
  .var,
  .spl_context,
  include_comp,
  .alt_df,
  conf_level,
  arm,
  strata,
  formats,
  methods
)
```

Arguments

<code>df</code>	(data.frame) data set containing all analysis variables.
<code>labelstr</code>	(character) label of the level of the parent split currently being summarized (must be present as second argument in Content Row Functions). See <code>rtables::summarize_row_groups()</code> for more information.
<code>label</code>	(string) only for logicals, which label to use. (For factors, the labels are the factor levels.)
<code>.var</code>	(string) single variable name that is passed by <code>rtables</code> when requested by a statistics function.
<code>.spl_context</code>	(data.frame) gives information about ancestor split states that is passed by <code>rtables</code> .
<code>include_comp</code>	(character or flag) whether to include comparative statistic results, either character for factors or flag for logicals.
<code>.alt_df</code>	(data.frame) alternative data frame used for denominator calculation.
<code>conf_level</code>	(proportion) confidence level of the interval.
<code>arm</code>	(string) column name in the data frame that identifies the treatment arms.
<code>strata</code>	(character or NULL) variable names indicating stratification factors.
<code>formats</code>	(list) containing formats for <code>prop_ci</code> , <code>comp_stat_ci</code> and <code>pval</code> .
<code>methods</code>	(list) containing methods for comparative statistics. The element <code>comp_stat_ci</code> can be 'rr' (relative risk), 'or_cmh' (odds ratio with CMH estimation and p-value) or 'or_logistic' (odds ratio estimated by conditional or standard logistic regression). The element <code>pval</code> can be 'fisher' (Fisher's exact test) or 'chisq' (chi-square test), only used when using unstratified analyses with 'or_logistic'. The element <code>prop_ci</code> specifies the method for proportion confidence interval calculation.

Value

The formatted result as `rtables::in_rows()` result.

Examples

```
fake_spl_context <- data.frame(
  cur_col_split_val = I(list(c(ARM = "A: Drug X", count_prop = "count_prop")))
)
```

```

dm <- droplevels(subset(DM, SEX %in% c("F", "M")))
resp01_acfun(
  dm,
  .alt_df = dm,
  .var = "COUNTRY",
  .spl_context = fake_spl_context,
  conf_level = 0.9,
  include_comp = c("USA", "CHN"),
  arm = "SEX",
  strata = "RACE",
  methods = list(
    comp_stat_ci = "or_cmh",
    pval = "",
    prop_ci = "wald"
  ),
  formats = list(
    prop_ci = jjcsformat_xx("xx.% - xx.%"),
    comp_stat_ci = jjcsformat_xx("xx.xx (xx.xx - xx.xx)"),
    pval = jjcsformat_pval_fct(0.05)
  )
)
fake_spl_context2 <- data.frame(
  cur_col_split_val = I(list(c(ARM = "Overall", comp_stat_ci = "comp_stat_ci")))
)
resp01_acfun(
  dm,
  .alt_df = dm,
  .var = "COUNTRY",
  .spl_context = fake_spl_context2,
  conf_level = 0.9,
  include_comp = c("USA", "CHN"),
  arm = "SEX",
  strata = "RACE",
  methods = list(
    comp_stat_ci = "or_cmh",
    pval = "",
    prop_ci = "wald"
  ),
  formats = list(
    prop_ci = jjcsformat_xx("xx.% - xx.%"),
    comp_stat_ci = jjcsformat_xx("xx.xx (xx.xx - xx.xx)"),
    pval = jjcsformat_pval_fct(0.05)
  )
)

```

Description

Content Row Function for Counts of Subgroups in Response Tables (RESP01)

Usage

```
resp01_counts_cfun(df, labelstr, .spl_context, .alt_df, label_fstr)
```

Arguments

- df (data.frame)
data set containing all analysis variables.
- labelstr (character)
label of the level of the parent split currently being summarized (must be present as second argument in Content Row Functions). See `rtables::summarize_row_groups()` for more information.
- .spl_context (data.frame)
gives information about ancestor split states that is passed by `rtables`.
- .alt_df (data.frame)
alternative data frame used for denominator calculation.
- label_fstr (string)
format string for the label.

Value

The correct `rtables::in_rows()` result.

Examples

```
fake_spl_context <- data.frame(  
  cur_col_split_val = I(list(c(ARM = "A: Drug X", count_prop = "count_prop")))  
)  
resp01_counts_cfun(  
  df = DM,  
  labelstr = "Blue",  
  .spl_context = fake_spl_context,  
  .alt_df = DM,  
  label_fstr = "Color: %s"  
)
```

resp01_split_fun_fct	<i>Split Function Factory for the Response Tables (RESP01)</i>
----------------------	--

Description

The main purpose here is to have a column dependent split into either comparative statistic (relative risk or odds ratio with p-value) in the 'Overall' column, and count proportions and corresponding confidence intervals in the other treatment arm columns.

Usage

```
resp01_split_fun_fct(method = c("rr", "or_logistic", "or_cmh"), conf_level)
```

Arguments

```
method      (string)
              which method to use for the comparative statistics.

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

Value

A split function for use in the response table RESP01 and similar ones.

See Also

[rtables::make_split_fun\(\)](#) describing the requirements for this kind of post-processing function.

Examples

```
split_fun <- resp01_split_fun_fct(
  method = "or_cmh",
  conf_level = 0.95
)
```

response_by_var	<i>Count denom fraction statistic</i>
-----------------	---------------------------------------

Description

Derives the count_denom_fraction statistic (i.e., 'xx /xx (xx.x percent)')
Summarizes the number of unique subjects with a response = 'Y' for a given variable (e.g. TRTEMFL) within each category of another variable (e.g., SEX). Note that the denominator is derived using input df, in order to have these aligned with alt_source_df, it is expected that df includes all subjects.

Usage

```
response_by_var(
  df,
  labelstr = NULL,
  .var,
  .N_col,
  resp_var = NULL,
  id = "USUBJID",
  .format = jjcsformat_count_denom_fraction,
  ...
)
```

Arguments

<code>df</code>	(data.frame) Name of dataframe being analyzed.
<code>labelstr</code>	(character vector) Custom label for the variable being analyzed.
<code>.var</code>	(character) Name of the variable being analyzed. Records with non-missing values will be counted in the denominator.
<code>.N_col</code>	(numeric) The total for the current column.
<code>resp_var</code>	(character) Name of variable, for which, records with a value of 'Y' will be counted in the numerator.
<code>id</code>	(character) Name of column in df which will have patient identifiers
<code>.format</code>	(character) Format for the count/denominator/fraction output.
<code>...</code>	Additional arguments passed to the function.

Details

This is an analysis function for use within `analyze`. Arguments `df`, `.var` will be populated automatically by `rtables` during the tabulation process.

Value

a `RowsVerticalSection` for use by the internal tabulation machinery of `rtables`

Examples

```
library(dplyr)

ADAE <- data.frame(
  USUBJID = c(
    "XXXXX01", "XXXXX02", "XXXXX03", "XXXXX04", "XXXXX05",
    "XXXXX06", "XXXXX07", "XXXXX08", "XXXXX09", "XXXXX10"
  ),
  SEX_DECODE = c(
    "Female", "Female", "Male", "Female", "Male",
    "Female", "Male", "Female", "Male", "Female"
  ),
  TRT01A = c(
    "ARMA", "ARMB", "ARMA", "ARMB", "ARMB",
    "Placebo", "Placebo", "Placebo", "ARMA", "ARMB"
  ),
  TRTEMFL = c("Y", "Y", "N", "Y", "Y", "Y", "Y", "N", "Y", "Y")
)
```

```

ADAE <- ADAE |>
  mutate(
    TRT01A = as.factor(TRT01A),
    SEX_DECODE = as.factor(SEX_DECODE)
  )

lyt <- basic_table() |>
  split_cols_by("TRT01A") |>
  analyze(
    vars = "SEX_DECODE",
    var_labels = "Sex, n/Ns (%)",
    show_labels = "visible",
    afun = response_by_var,
    extra_args = list(resp_var = "TRTEMFL"),
    nested = FALSE
  )

result <- build_table(lyt, ADAE)

result

```

rightside

Extract the right-hand side of a formula

Description

Extract the right-hand side of a formula

Usage

```
rightside(x)
```

Arguments

x (formula)
A two-sided formula, e.g., $y \sim x1 + x2$.

Value

(character(1)) The right-hand side of the formula.

Examples

```
rightside(y ~ x1 + x2)
```

rm_levels

Removal of Levels

Description

Custom function for removing level inside pre step in make_split_fun.

Usage

```
rm_levels(excl)
```

Arguments

excl (character)
Choose which level(s) to remove

Value

A function implementing pre-processing split behavior (for use in make_split_fun(pre =) which removes the levels in excl from the data before facets are generated.

rm_other_facets_fact *rm_other_facets_fact*

Description

```
rm_other_facets_fact
```

Usage

```
rm_other_facets_fact(nm)
```

Arguments

nm character. names of facets to keep. all other facets will be removed

Value

a function suitable for use within the post portion make_split_fun

s_ancova_j

*Junco Extended ANCOVA Function***Description**

Extension to `tern::s_ancova`, 3 extra statistics are returned:

- `lsmean_se`: Marginal mean and estimated SE in the group.
- `lsmean_ci`: Marginal mean and associated confidence interval in the group.
- `lsmean_diff_with_ci`: Difference in mean and associated confidence level in one combined statistic. In addition, the LS mean weights can be specified. In addition, also a `NULL.ref_group` can be specified, the `lsmean_diff` related estimates will be returned as NA.

Usage

```
s_ancova_j(
  df,
  .var,
  .df_row,
  variables,
  .ref_group,
  .in_ref_col,
  conf_level,
  interaction_y = FALSE,
  interaction_item = NULL,
  weights_emmeans = "counterfactual",
  method_combo = c("contrasts", "collapse"),
  weights_combo = NULL
)
```

Arguments

- | | |
|------------------------|--|
| <code>df</code> | (<code>data.frame</code>)
data set containing all analysis variables. |
| <code>.var</code> | (<code>string</code>)
single variable name that is passed by <code>rtables</code> when requested by a statistics function. |
| <code>.df_row</code> | (<code>data.frame</code>)
data set that includes all the variables that are called in <code>.var</code> and <code>variables</code> . |
| <code>variables</code> | (<code>named list of string</code>)
list of additional analysis variables, with expected elements: <ul style="list-style-type: none"> • <code>arm</code> (<code>string</code>)
group variable, for which the covariate adjusted means of multiple groups will be summarized. Specifically, the first level of <code>arm</code> variable is taken as the reference group. |

	covariates (character) a vector that can contain single variable names (such as "X1"), and/or interaction terms indicated by "X1 * X2".
.ref_group	(data.frame or vector) the data corresponding to the reference group.
.in_ref_col	(flag) TRUE when working with the reference level, FALSE otherwise.
conf_level	(proportion) confidence level of the interval.
interaction_y	(string or flag) a selected item inside of the interaction_item variable which will be used to select the specific ANCOVA results. if the interaction is not needed, the default option is FALSE.
interaction_item	(string or NULL) name of the variable that should have interactions with arm. if the interaction is not needed, the default option is NULL.
weights_emmeans	(string) argument from <code>emmeans::emmeans()</code> , "counterfactual" by default.
method_combo	(string) Method for derivations in combined column. - contrast Derivations for the combined level are done through contrasts from the original model (using weights per weights_combo specifications). - collapse The ancova model for the combined group will be performed with group levels that contribute to the combination collapsed into a single combined level. For more information see the vignette ANCOVA with Combined Treatment Groups.
weights_combo	(string) Weights for the contrasts of the combined levels. - equal 1/(number of levels from arm variable included in the combination) - proportional, proportional_marginal weight for each level included in the combination is proportional to number of observations in that level The difference between proportional and proportional_marginal is only relevant when the model includes an interaction between arm and other factor variable (interaction_item). proportional_marginal interprets proportional over all levels of interaction_item, ie, the same weights will be used for all levels of interaction_item. For proportional the weights will be derived within the requested level (interaction_y) for interaction_item.

Value

Returns a named list of 8 statistics (3 extra compared to `tern::s_ancova()`).

See Also

Other Inclusion of ANCOVA Functions: [a_summarize_ancova_j\(\)](#), [a_summarize_aval_chg_diff_j\(\)](#)

Examples

```
library(dplyr)
library(tern)

df <- iris |> filter(Species == "virginica")
.df_row <- iris
.var <- "Petal.Length"
variables <- list(arm = "Species", covariates = "Sepal.Length * Sepal.Width")
.ref_group <- iris |> filter(Species == "setosa")
conf_level <- 0.95
s_ancova_j(df, .var, .df_row, variables, .ref_group, .in_ref_col = FALSE, conf_level)
```

s_proportion_factor	<i>s_function for proportion of factor levels</i>
---------------------	---

Description

A simple statistics function which prepares the numbers with percentages in the required format. The denominator here is from the alternative counts data set in the given row and column split.

If a total row is shown, then here just the total number is shown (without 100%).

Usage

```
s_proportion_factor(
  x,
  .alt_df,
  use_alt_counts = TRUE,
  show_total = c("none", "top", "bottom"),
  total_label = "Total"
)
```

Arguments

x	(factor) categorical variable we want to analyze.
.alt_df	(data.frame) alternative data frame used for denominator calculation.
use_alt_counts	(flag) whether the .alt_df should be used for the total, i.e. the denominator. If not, then the number of non-missing values in x is used.
show_total	(string) show the total level optionally on the top or in the bottom of the factor levels.
total_label	(string) which label to use for the optional total level.

Value

The `rtables::in_rows()` result with the proportion statistics.

See Also

`s_proportion_logical()` for tabulating logical x.

s_proportion_logical	<i>s_function for proportion of TRUE in logical vector</i>
----------------------	--

Description

A simple statistics function which prepares the numbers with percentages in the required format. The denominator here is from the alternative counts data set in the given row and column split.

Usage

```
s_proportion_logical(x, label = "Responders", .alt_df)
```

Arguments

x	(logical) binary variable we want to analyze.
label	(string) label to use.
.alt_df	(data.frame) alternative data frame used for denominator calculation.

Value

The `rtables::in_rows()` result with the proportion statistics.

See Also

`s_proportion_factor()` for tabulating factor x.

s_summary_diff	<i>Descriptive Statistics for Univariate Data with Optional Reference Comparison</i>
----------------	--

Description

[Experimental]

Computes descriptive statistics for a single variable `df[[.var]]` using `tern::s_summary()`, which dispatches type-specific methods depending on the S3 class of the input (e.g., character, factor, logical, numeric).

Optionally, it computes estimates and associated inferential statistics for the difference in population means between two underlying distributions, based on observed samples `df[[.var]]` and `.ref_group[[.var]]` using `s_diff_means()`. These statistics are applicable only to numeric data.

Usage

```
s_summary_diff(
  df,
  .var,
  .stats = NULL,
  .ref_group = NULL,
  .in_ref_col = FALSE,
  control = tern::control_analyze_vars(),
  ...
)
```

Arguments

<code>df</code>	(data.frame) data set containing all analysis variables.
<code>.var</code>	(character(1)) Name of the column in <code>df</code> containing the values for which statistics are computed. The variable type is handled by the corresponding methods of <code>tern::s_summary()</code> . If difference-in-means statistics are requested, <code>df[[.var]]</code> must be numeric.
<code>.stats</code>	(character or NULL) Names of statistics to be computed. For numerical data, the names of supported statistics are listed via <code>tern::get_stats()</code> for statistics for a single variable and <code>junco_get_stats("diff_means")</code> for difference-in-means statistics. If NULL, all available statistics for numerical data are computed.
<code>.ref_group</code>	(data.frame or vector) the data corresponding to the reference group.
<code>.in_ref_col</code>	(logical) TRUE when working with the reference level, FALSE otherwise.

control (list)
List of control options passed to `tern::s_summary()`. If difference-in-means statistics are requested, `control$conf_level` specifies the confidence level of the interval.

... Additional named arguments passed to `s_diff_means()`, except `df1`, `df2`, `.var`, and `conf.level`, which are already passed.

Value

A named list with the requested statistics.

Author(s)

WW

Examples

```
df <- data.frame(
  USUBJID = c("X01", "X02", "X03", "X04", "X05"),
  TRT01A = rep("ARM_A", 5),
  PARAMCD = rep("SYSBP", 5),
  AVISIT = rep("Visit 1", 5),
  CHG = c(4, 1, -1, 9, -2)
)
df

rg <- data.frame(
  USUBJID = c("X06", "X07", "X08", "X09", "X10"),
  TRT01A = rep("Placebo", 5),
  PARAMCD = rep("SYSBP", 5),
  AVISIT = rep("Visit 1", 5),
  CHG = c(-2, 6, -2, 5, 2)
)
rg

.stats <- c("n", "mean_sd", "diff_means_n1", "diff_means_n2", "diff_means_est_ci")

# With reference group.
s_summary_diff(df, "CHG", .stats, rg)

# Using df as reference.
s_summary_diff(df, "CHG", .stats, df, .in_ref_col = TRUE)
```

safe_prune_table

Safely Prune Table With Empty Table Message If Needed

Description

Safely Prune Table With Empty Table Message If Needed

Usage

```
safe_prune_table(
  tt,
  prune_func = prune_empty_level,
  stop_depth = NA,
  empty_msg = " - No Data To Display - ",
  spancols = FALSE
)
```

Arguments

<code>tt</code>	(TableTree or related class) a TableTree object representing a populated table.
<code>prune_func</code>	(function) a function to be called on each subtree which returns TRUE if the entire subtree should be removed.
<code>stop_depth</code>	(numeric(1)) the depth after which subtrees should not be checked for pruning. Defaults to NA which indicates pruning should happen at all levels.
<code>empty_msg</code>	(character(1)) The message to place in the table if no rows were left after pruning
<code>spancols</code>	(logical(1)) Should <code>empty_msg</code> be spanned across the table's columns (TRUE) or placed in the rows row label (FALSE). Defaults to FALSE currently.

Value

`tt` pruned based on the arguments, or, if pruning would remove all rows, a TableTree with the same column structure, and one row containing the empty message spanning all columns.

Examples

```
prfun <- function(tt) TRUE

lyt <- basic_table() |>
  split_cols_by("ARM") |>
  split_cols_by("STRATA1") |>
  split_rows_by("SEX") |>
  analyze("AGE")
tbl <- build_table(lyt, ex_adsl)

safe_prune_table(tbl, prfun)
```

set_titles	<i>Set Output Titles</i>
------------	--------------------------

Description

Retrieves titles and footnotes from the list specified in the titles argument and appends them to the TableTree specified in the obj argument.

Usage

```
set_titles(obj, titles)
```

Arguments

- obj (TableTree)
The TableTree to which the titles and footnotes will be appended.
- titles (list)
The list object containing the titles and footnotes to be appended.

Value

The TableTree object specified in the obj argument, with titles and footnotes appended.

shift_tbl_col_struct	<i>Create Column Structure for a J&J-Style Shift Table</i>
----------------------	--

Description

Create Column Structure for a J&J-Style Shift Table

Usage

```
shift_tbl_col_struct(lyt, var, span_lbl = "Baseline", .outer_spl_var = var)
```

Arguments

- lyt (PreDataTableLayouts)
The layout, typically as returned directly from basic_table.
- var (character(1))
The variable defining the partition to create a shift table for.
- span_lbl (character(1))
The spanning label for the shift portion of the column structure. Defaults to "Baseline".
- .outer_spl_var (character(1))
The variable to "split on" when creating the spanning labels. Must exist in the dataset(s) used in build_table but its values are irrelevant.

Details

This function creates a column structure akin to the following (with Grade 1 through Grade 5 representing the levels of var):

			Baseline				
N	Grade 1	Grade 2	Grade 3	Grade 4	Grade 5	Total	

Value

lyt updated with the specified shift table column structure.

See Also

Other std_col_struct: [grouped_cols_w_diffs\(\)](#), [make_multicomp_splfun\(\)](#)

Examples

```
shift_data <- data.frame(
  BASE = factor(c("Grade 1", "Grade 2", "Grade 3")),
  CHG = c("Improved", "Stable", "Worsened")
)

lyt <- basic_table() |>
  shift_tbl_col_struct("BASE") |>
  analyze("CHG", afun = function(x, ...) length(x))

build_table(lyt, shift_data)
```

<code>some_v_all_col_struct</code>	<i>Standard All vs Some (e.g. Related AEs) column structure</i>
------------------------------------	---

Description

Standard All vs Some (e.g. Related AEs) column structure

Usage

```
some_v_all_col_struct(
  lyt,
  colspan_trt_map = NULL,
  combo_map_df = NULL,
  trtvar = names(colspan_trt_map)[2],
  subgrp_var = NULL,
  subgrp_lbl = subgrp_var,
  all_lbl,
  subgrp_lvls,
  .pre = list(),
  .post = list()
)
```

Arguments

lyt	(PreDataTableLayouts). The layout to modify. This should virtually always be the object returned by <code>basic_table</code> .
colspan_trt_map	(data.frame). The spanning label map for the main columns, as given by <code>create_colspan_map</code> .
combo_map_df	(data.frame or NULL). A combination data frame as defined by <code>add_combo_levels()</code> with an additional <code>is_control</code> column indicating whether the virtual level will act as a reference (TRUE) or active (FALSE) group.
trtvar	(character(1) or NULL) the treatment variable to split by. Defaults to the treatment variable in <code>colspan_trt_map</code> .
subgrp_var	(character(1) or NULL) the name of the subgroup variable to split by within the <code>trtvar</code> split
subgrp_lbl	(character(1)) The label to put above the combination level representing <code>subgrp_lvls</code>
all_lbl	(character(1)) The label to put above the "All" column
subgrp_lvls	(character) All level(s) to be included in the <code>subgrp_lbl</code> column.
.pre	(list of functions) Passed to <code>rtables::make_split_fun()</code> as <code>pre</code> for treatment splitting.
.post	(list of functions) Passed to <code>rtables::make_split_fun()</code> as <code>post</code> for treatment splitting after the standard column-structure processing.

Details

This column structure generating function is for comparing a single portion of the data (as represented by level(s) of `subgrp_var`) against the full data, comparison of AE counts to treatment-related AE counts being a motivating example.

Examples

```
library(junco)
dat <- create_colspan_var(
  data.frame(
    TRT01A = factor(rep(c("Placebo", "Active 1", "Active 2"), each = 5)),
    GRADE = factor(rep(paste("Grade ", 1:5), 3))
  ),
  non_active_grp = "Placebo",
  non_active_grp_span_lbl = "Control",
  active_grp_span_lbl = "Active Treatment",
  colspan_var = "colspan_trt",
  trt_var = "TRT01A"
)
colspan_trt_map <- create_colspan_map(
  dat,
```

```

    non_active_grp = "Placebo",
    non_active_grp_span_lbl = "Control",
    active_grp_span_lbl = "Active Treatment",
    colspan_var = "colspan_trt",
    trt_var = "TRT01A"
  )

  lyt <- basic_table() |>
    some_v_all_col_struct(
      colspan_trt_map,
      subgrp_var = "GRADE",
      subgrp_lvls = c("Grade 4", "Grade 5"),
      subgrp_lbl = "High Grade",
      all_lbl = "All Grades")

  build_table(lyt, dat)

```

stats_in_cols_setup	<i>Function factory to be used as Post-processing split function for setup of stats in column structure</i>
---------------------	---

Description

This helper is designed to be used in the post argument of `rtables::make_split_fun()` to expand a column facet (e.g. AVAL/BASE/CHG) into the specific statistics to be analyzed for each subfacet. It returns a split result instructing rtables which values/labels/subsets to create.

Typical usage:

- construct a split function like `mysplitfun <- rtables::make_split_fun(post = list(stats_in_cols_setup()))`. Then use `mysplitfun` in your table layout where you split columns by a variable whose levels are one of "AVAL", "BASE", or "CHG" and want to analyze different statistics for each.
- column layout setup from `rtables::split_cols_by_multivar()`, followed by `split_cols_by` using `stats_in_cols_setup` in conjunction with an analyze call with `a_fun = column_stats()`

See examples for typical usage.

Usage

```

stats_in_cols_setup(
  stats_list = list(AVAL = c("n", "mean", "sd", "se", "median", "min", "max"), BASE =
    c("mean_sd"), CHG = c("n", "mean", "sd", "se", "median", "min", "max")),
  statlbls_list = c(median = "Med", min = "Min", max = "Max", mean_sd = "Base Mean (SD)",
    n = "N")
)

```

Arguments

- `stats_list` (list) Named list with statistics to present for each variable. Any statistic from `tern::s_summary` can be selected.
- `statlbls_list` “ Named vector with labels to use for the statistic in column header. If NULL, or a statistic name is not included, the default label for the statistic will be used.

Value

A function that can be used inside `rtables::make_split_result()` statistics for the current column level.

See Also

`rtables::make_split_fun()`, `rtables::make_split_result()`.

Examples

```
# example code
library(dplyr)
adv<= ex_adv<= |>
  filter(AVISIT %in% toupper(c("Screening", "Baseline", "Week 1 Day 8", "Week 2 Day 15"))) |>
  mutate(AVISIT = droplevels(AVISIT))

mysplitfun <- make_split_fun(
  post = list(stats_in_cols_setup())
)
lyt <- basic_table() |>
  split_cols_by_multivar(
    c("AVAL", "BASE", "CHG"),
    varlabels = c("Visit Values", " ", "Change from Baseline")
  ) |>
  split_cols_by("STUDYID", split_fun = mysplitfun) |>
  split_rows_by("PARAM") |>
  split_rows_by("ARMCD", labels_var = "ARM") |>
  analyze("AVISIT",
    afun = column_stats,
    extra_args = list(exclude_visits = toupper(c("Baseline", "Screening")))
  )
result <- build_table(lyt, advs, round_type = "sas")
head(result, 20)
```

string_to_title

*Title Case Conversion***Description****[Stable]**

Convert strings to title case (first letter of each word capitalized), with the option to keep specified words, such as conjunctions, lowercase.

Usage

```
string_to_title(x, lowercase_words = NULL)
```

Arguments

x (character or factor)
Input string.

lowercase_words (character or NULL)
Words that should remain lowercase when converting to title case. Matching is case-insensitive. Specified words are only converted to lowercase when they are not the first word in the string. The first word is always capitalized, even if it is included in lowercase_words.

Value

x converted to title case, with words specified in lowercase_words kept lowercase.

Examples

```
x <- c("THIS IS an eXaMple", "statement TO CAPItaliZe")
string_to_title(x)
string_to_title(x, c("is", "an", "to"))

x <- factor(
  c("OPTIMAL DOSE", "UNDERDOSE"),
  levels = c("OPTIMAL DOSE", "UNDERDOSE", "OVERDOSE")
)
string_to_title(x)
```

```
summarize_coxreg_multivar
```

Layout Generating Function for TEFOS03 and Related Cox Regression Layouts

Description

Layout Generating Function for TEFOS03 and Related Cox Regression Layouts

Usage

```
summarize_coxreg_multivar(
  lyt,
  var,
  variables,
  control = control_coxreg(),
  formats = list(coef_se = jjcsformat_xx("xx.xx (xx.xx)"), hr_est =
```

```

      jjcsformat_xx("xx.xx"), hr_ci = jjcsformat_xx("(xx.xx, xx.xx)"), pval =
      jjcsformat_pval_fct(0))
)

```

Arguments

lyt	(layout) input layout where analyses will be added to.
var	(string) any variable from the data, because this is not used.
variables	(named list of string) list of additional analysis variables.
control	(list) relevant list of control options.
formats	(named character or list) formats for the statistics. See Details in analyze_vars for more information on the 'auto' setting.

Value

lyt modified to add the desired cox regression table section.

Examples

```

anl <- tern::tern_ex_adtte |>
  dplyr::mutate(EVENT = 1 - CNSR)

variables <- list(
  time = "AVAL",
  event = "EVENT",
  arm = "ARM",
  covariates = c("SEX", "AGE")
)

basic_table() |>
  summarize_coxreg_multivar(
    var = "STUDYID",
    variables = variables
  ) |>
  build_table(df = anl)

```

Description**[Stable]**

These functions can be used to produce tables for MMRM results, within tables which are split by arms and visits. This is helpful when higher-level row splits are needed (e.g. splits by parameter or subgroup).

Usage

```
s_summarize_mmrn(
  df,
  .var,
  variables,
  ref_levels,
  .spl_context,
  alternative = c("two.sided", "less", "greater"),
  show_relative = c("reduction", "increase"),
  ...
)

a_summarize_mmrn(
  df,
  .var,
  .spl_context,
  ...,
  .stats = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)

a_summarize_mmrn_with_exclude(
  df,
  .var,
  exclude_levels,
  .spl_context,
  ...,
  .stats = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```

Arguments

<code>df</code>	(data.frame) data set containing all analysis variables.
<code>.var</code>	(string) single variable name that is passed by <code>rtables</code> when requested by a statistics

	function.
variables	(named list of string) list of additional analysis variables.
ref_levels	(list) with visit and arm reference levels.
.spl_context	(data.frame) gives information about ancestor split states that is passed by rtables.
alternative	(string) whether two.sided, or one-sided less or greater p-value should be displayed.
show_relative	(string) should the 'reduction' (control - treatment, default) or the 'increase' (treatment - control) be shown for the relative change from baseline?
...	eventually passed to <code>fit_mmrn_j()</code> via <code>h_summarize_mmrn()</code> .
.stats	(character) statistics to select for the table.
.formats	(named character or list) formats for the statistics. See Details in <code>analyze_vars</code> for more information on the 'auto' setting.
.labels	(named character) labels for the statistics (without indent).
.indent_mods	(named integer) indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.
exclude_levels	(list) A named list where names correspond to split variables and values are vectors of levels to exclude.

Value

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

Functions

- `s_summarize_mmrn()`: Statistics function which is extracting estimates, not including any results when in the reference visit, and only showing LS mean estimates when in the reference arm and not in reference visit. It uses `s_lsmeans()` for the final processing.
- `a_summarize_mmrn()`: Formatted analysis function which is used as `afun`.
- `a_summarize_mmrn_with_exclude()`: Wrapper for the `afun` which can exclude row split levels from producing the analysis. These have to be specified in the `exclude_levels` argument, see `?do_exclude_split` for details.

Examples

```
set.seed(123)
longdat <- data.frame(
```

```

ID = rep(DM$ID, 5),
AVAL = c(
  rep(0, nrow(DM)),
  rnorm(n = nrow(DM) * 4)
),
VISIT = factor(rep(paste0("V", 0:4), each = nrow(DM)))
) |>
dplyr::inner_join(DM, by = "ID")

basic_table() |>
split_rows_by("VISIT") |>
split_cols_by("ARM") |>
analyze(
  vars = "AVAL",
  afun = a_summarize_mmrn,
  na_str = tern::default_na_str(),
  show_labels = "hidden",
  extra_args = list(
    variables = list(
      covariates = c("AGE"),
      id = "ID",
      arm = "ARM",
      visit = "VISIT"
    ),
    conf_level = 0.9,
    cor_struct = "toeplitz",
    ref_levels = list(VISIT = "V0", ARM = "B: Placebo")
  )
) |>
build_table(longdat) |>
prune_table(all_zero)

```

summarize_row_counts *Layout Creating Function Adding Row Counts*

Description

This is a simple wrapper of `rtables::summarize_row_groups()` and the main additional value is that we can choose whether we want to use the alternative (usually ADSL) data set for the counts (default) or use the original data set.

Usage

```
summarize_row_counts(lyt, label_fstr = "%s", alt_counts = TRUE)
```

Arguments

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

`label_fstr` (string)
a sprintf style format string. It can contain up to one %s which takes the current split value and generates the row label.

`alt_counts` (flag)
whether row counts should be taken from `alt_counts_df` (TRUE) or from `df` (FALSE).

Value

A modified layout where the latest row split now has a row group summaries (as created by [rtab-les::summarize_row_groups](#) for the counts).

Examples

```
basic_table() |>
  split_cols_by("ARM") |>
  add_colcounts() |>
  split_rows_by("RACE", split_fun = drop_split_levels) |>
  summarize_row_counts(label_fstr = "RACE value - %s") |>
  analyze("AGE", afun = list_wrap_x(summary), format = "xx.xx") |>
  build_table(DM, alt_counts_df = rbind(DM, DM))
```

summary_subset	<i>Wrappers Around <code>tern::a_summary()</code> with Optional Subsetting and Labels</i>
----------------	---

Description

[Stable]

These wrappers around [tern::a_summary\(\)](#) allow optional row subsetting using a logical expression evaluated in the context of the input data frame.

`a_summary_subset()` is an [rtab-les](#) analysis function (see [rtab-les::analyze\(\)](#)), while `c_summary_subset_label()` is a content row function (see [rtab-les::summarize_row_groups\(\)](#)) that may optionally append a label to the resulting content row.

Usage

```
a_summary_subset(df, .var, filter_expr, ...)
```

```
c_summary_subset_label(
  df,
  labelstr,
  .var,
  filter_expr,
  label = "",
  label_indent_mod = 0L,
  ...
)
```

Arguments

<code>df</code>	(data.frame) data set containing all analysis variables.
<code>.var</code>	(string) single variable name that is passed by <code>rtables</code> when requested by a statistics function.
<code>filter_expr</code>	(expression) A single logical expression wrapped in <code>expression(...)</code> used to subset rows of <code>df</code> before summarization. Exactly one expression is allowed. The expression is evaluated using <code>eval()</code> in an environment containing the columns of <code>df</code> , meaning that all column names are available as variables. In the current implementation, variables used in the expression can only be column names of <code>df</code> . That is, the following is not supported: <code>min_age <- 45; a_summary_subset(df, "AVAL", expression(AGE > min_age))</code> , given that <code>AVAL</code> is a column in <code>df</code> and <code>min_age</code> is not a column in <code>df</code> . The expression must evaluate to a logical vector whose length is exactly equal to <code>nrow(df)</code> . No recycling is performed. In particular, scalar logical values such as <code>expression(TRUE)</code> or <code>expression(FALSE)</code> are not allowed when <code>nrow(df) > 1</code> .
<code>...</code>	Additional arguments passed to <code>tern::a_summary()</code> , excluding the <code>x</code> argument, which is constructed internally from <code>df[[.var]]</code> after applying the filter.
<code>labelstr</code>	(character) label of the level of the parent split currently being summarized (must be present as second argument in Content Row Functions). See <code>rtables::summarize_row_groups()</code> for more information.
<code>label</code>	(character(1)) Label to insert as the first row.
<code>label_indent_mod</code>	(integer(1)) Indentation modifier applied to the label row.

Details

`tern::a_summary()` is a formatted analysis function that computes summary statistics for a single variable. In typical use within the `rtables` framework, the analysis variable is first extracted from a data frame and then passed to `tern::a_summary()` as a standalone vector.

The `a_summary_subset()` and `c_summary_subset_label()` wrappers extend this behavior by:

- accepting a data frame (`df`) as input,
- optionally subsetting rows using a logical expression,
- extracting the analysis variable `.var` after subsetting,
- and passing the resulting vector to `tern::a_summary()`.

Value

Returns the same type of object as `tern::a_summary()`.

Functions

- `a_summary_subset()`: Analysis function with optional subsetting.
- `c_summary_subset_label()`: Content row function with optional subsetting and an optional label prepended to the resulting content row using `prepend_label_cell()`.

Author(s)

WW

See Also[tern::a_summary\(\)](#)**Examples**

```
## Analysis function:

df <- data.frame(
  ID = 1:8,
  AGE = c(34, NA, 52, 45, 50, NA, 41, 29),
  AVAL = c(1.2, 3, -1.5, 0.8, 3.5, 2, -1.1, 0.3)
)
df

stats <- c("n", "sum")

tern::a_summary(df$AVAL, .stats = stats)

a_summary_subset(df, "AVAL", expression(!is.na(AGE)), .stats = stats)
a_summary_subset(df, "AVAL", expression(AGE > 30 & AVAL > 0), .stats = stats)

## Content row function:

df <- data.frame(
  USUBJID = rep(1:6, each = 2),
  AVISIT = rep(c("Baseline", "Day 1"), 6),
  AVAL = c(1, 3, 2, 9, 13, 19, 15, 23, 43, 56, 24, 32),
  ABLFL = rep(c(TRUE, FALSE), 6),
  BASE = rep(c(1, 2, 13, 15, 43, 24), each = 2),
  CHG = c(0, 2, 0, 7, 0, 6, 0, 8, 0, 13, 0, 8)
)
df

c_summary_subset_label(
  df = df,
  .var = "BASE",
  filter_expr = expression(ABLFL),
  label = "Baseline",
  .stats = c("n", "mean_sd")
)
```

tabulate_lsmeans	<i>Tabulation of Least Square Means Results</i>
------------------	---

Description**[Stable]**

These functions can be used to produce tables from LS means, e.g. from `fit_mmrj()` or `fit_ancova()`.

Usage

```
## S3 method for class 'tern_model'
tidy(x, ...)

s_lsmeans(
  df,
  .in_ref_col,
  alternative = c("two.sided", "less", "greater"),
  show_relative = c("reduction", "increase")
)

a_lsmeans(
  df,
  ref_path,
  .spl_context,
  ...,
  .stats = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```

Arguments

<code>x</code>	(numeric) vector of numbers we want to analyze.
<code>...</code>	additional arguments for the lower level functions.
<code>df</code>	(data.frame) data set containing all analysis variables.
<code>.in_ref_col</code>	(logical) TRUE when working with the reference level, FALSE otherwise.
<code>alternative</code>	(string) whether two.sided, or one-sided less or greater p-value should be displayed.
<code>show_relative</code>	(string) should the 'reduction' (control - treatment, default) or the 'increase' (treatment - control) be shown for the relative change from baseline?

<code>ref_path</code>	(character) global reference group specification, see get_ref_info() .
<code>.spl_context</code>	(data.frame) gives information about ancestor split states that is passed by <code>rtables</code> .
<code>.stats</code>	(character) statistics to select for the table.
<code>.formats</code>	(named character or list) formats for the statistics. See Details in <code>analyze_vars</code> for more information on the 'auto' setting.
<code>.labels</code>	(named character) labels for the statistics (without indent).
<code>.indent_mods</code>	(named integer) indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.

Value

- For `s_lsmeans`, a list containing the same statistics returned by `tern.mmrm::s_mmrm_lsmeans`, with the additional `diff_mean_est_ci` three-dimensional statistic.
- For `a_lsmeans`, a `VerticalRowsSection` as returned by [rtables::in_rows](#).

Functions

- `tidy(tern_model)`: Helper method (for [broom::tidy\(\)](#)) to prepare a `data.frame` from an `tern_model` object containing the least-squares means and contrasts.
- `s_lsmeans()`: Statistics function which is extracting estimates from a tidied least-squares means data frame.
- `a_lsmeans()`: Formatted Analysis function to be used as `afun`

Note

These functions have been forked from the `tern.mmrm` package. Additional features are:

- Additional `ref_path` argument for `tern.mmrm::summarize_lsmeans()`.
- The function is more general in that it also works for LS means results from ANCOVA
- Additional statistic `diff_mean_est_ci` is returned
- P-value sidedness can be chosen

Examples

```
result <- fit_mmrm_j(
  vars = list(
    response = "FEV1",
    covariates = c("RACE", "SEX"),
    id = "USUBJID",
    arm = "ARMCD",
    visit = "AVISIT"
```

```

    ),
    data = mrmr::fev_data,
    cor_struct = "unstructured",
    weights_emmeans = "equal"
  )

df <- broom::tidy(result)

s_lsmeans(df[8, ], .in_ref_col = FALSE)
s_lsmeans(df[8, ], .in_ref_col = FALSE, alternative = "greater", show_relative = "increase")

dat_adsl <- mrmr::fev_data |>
  dplyr::select(USUBJID, ARMCD) |>
  unique()

basic_table() |>
  split_cols_by("ARMCD") |>
  add_colcounts() |>
  split_rows_by("AVISIT") |>
  analyze(
    "AVISIT",
    afun = a_lsmeans,
    show_labels = "hidden",
    na_str = tern::default_na_str(),
    extra_args = list(
      .stats = c(
        "n",
        "adj_mean_se",
        "adj_mean_ci",
        "diff_mean_se",
        "diff_mean_ci"
      ),
      .labels = c(
        adj_mean_se = "Adj. LS Mean (Std. Error)",
        adj_mean_ci = "95% CI",
        diff_mean_ci = "95% CI"
      ),
      .formats = c(adj_mean_se = jjcsformat_xx("xx.x (xx.xx)")),
      alternative = "greater",
      ref_path = c("ARMCD", result$ref_level)
    )
  ) |>
  build_table(
    df = broom::tidy(result),
    alt_counts_df = dat_adsl
  )

```

Description**[Stable]**

These functions can be used to produce tables from RBMI.

Usage

```
h_tidy_pool(x, visit_name, group_names)

s_rbm_lsmmeans(df, .in_ref_col, show_relative = c("reduction", "increase"))

a_rbm_lsmmeans(
  df,
  ref_path,
  .spl_context,
  ...,
  .stats = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL
)
```

Arguments

x	(list) is a list of pooled object from rbmi analysis results. This list includes analysis results, confidence level, hypothesis testing type.
visit_name	(string) single visit level.
group_names	(character) group levels.
df	(data.frame) input with LS means results.
.in_ref_col	(flag) whether reference column is specified.
show_relative	(string) 'reduction' if (control - treatment, default) or 'increase' (treatment - control) of relative change from baseline?
ref_path	(character) global reference group specification, see get_ref_info() .
.spl_context	(data.frame) gives information about ancestor split states that is passed by rtables.
...	additional arguments for the lower level functions.
.stats	(character) statistics to select for the table.

<code>.formats</code>	(named character or list) formats for the statistics. See Details in <code>analyze_vars</code> for more information on the 'auto' setting.
<code>.labels</code>	(named character) labels for the statistics (without indent).
<code>.indent_mods</code>	(named integer) indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.

Value

- `h_tidy_pool()` returns a `data.frame` with results of pooled analysis for a single visit.
- `s_rbmi_lsmeans()` returns a list of statistics extracted from a tidied LS means data frame.

Functions

- `h_tidy_pool()`: Helper function to produce data frame with results of pool for a single visit.
- `s_rbmi_lsmeans()`: Statistics function which is extracting estimates from a tidied RBMI results data frame.
- `a_rbmi_lsmeans()`: Formatted Analysis function which is used as `afun`.

Note

These functions have been forked from `tern.rbmi`. Additional features are:

- Additional `ref_path` argument.
- Extraction of variance statistics in the `tidy()` method.
- Adapted to `rbmi` forked functions update with more than two treatment groups.

test_proportion_diff_mf

Test of Proportion Difference with Method Selection Based on the Mantel-Fleiss Criterion

Description

[Stable]

Tests the difference in response proportions between a non-reference group and a reference group for stratified data. The test method is selected based on the Mantel-Fleiss (MF) criterion.

When the Mantel-Fleiss criterion is satisfied, a Cochran-Mantel-Haenszel (CMH)-type test is performed using the method specified by `mf_method`. When the criterion is not satisfied, Fisher's exact test is used instead.

Usage

```

a_test_proportion_diff_mf(
  df,
  .var,
  .in_ref_col = NULL,
  .ref_group = NULL,
  val = TRUE,
  variables = list(strata = NULL),
  na.rm = FALSE,
  alternative = c("two.sided", "less", "greater"),
  mf_method = c("cmh", "cmh_sato", "cmh_wh"),
  mf_threshold = NULL,
  ...,
  .stats = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL,
  exact_footnote = rtables::RefFootnote(note = tern::d_test_proportion_diff(method =
    "fisher", method_only = TRUE), index = 1L, symbol = "+")
)

s_test_proportion_diff_mf(
  df,
  .var,
  .in_ref_col = NULL,
  .ref_group = NULL,
  val = TRUE,
  variables = list(strata = NULL),
  na.rm = FALSE,
  alternative = c("two.sided", "less", "greater"),
  mf_method = c("cmh", "cmh_sato", "cmh_wh"),
  mf_threshold = NULL
)

```

Arguments

<code>df</code>	(data.frame) The data containing all analysis variables for the non-reference group.
<code>.var</code>	(character(1)) The column name in <code>df</code> (and, if supplied, <code>.ref_group</code>) specifying the response variable. The response is converted to a logical vector by comparing its values with <code>val</code> . <code>df[[var]]</code> (and <code>.ref_group[[var]]</code> , if supplied) must be an atomic vector as defined by <code>checkmate::check_atomic_vector()</code> , of one of the following types: logical, integer, numeric, or character.
<code>.in_ref_col</code>	(logical(1) or NULL) Indicates whether the function is being evaluated for the reference column. Set to TRUE for the reference column and to FALSE or NULL otherwise. For the statistics function, TRUE or NULL results in empty statistics, while for the analysis

	function, the result is a RowsVerticalSection with a NULL value for all selected statistics.
.ref_group	(data.frame or NULL) The data corresponding to the reference group.
val	(logical(1) or integer(1) or numeric(1) or character(1)) The value used to identify the response of interest. An observation is considered a response of interest when <code>df[[.var]] == val</code> (and, if the reference group is supplied, <code>.ref_group[[.var]] == val</code>). If <code>df[[.var]]</code> is a factor, <code>val</code> must be a character value matching one of its levels. Otherwise, <code>val</code> must have the same class as <code>df[[.var]]</code> .
variables	(list or NULL) Optional strata variables specified as <code>list(strata = <strata_vars>)</code> , where <code><strata_vars></code> is a character vector containing column names in <code>df</code> (and, if supplied, <code>.ref_group</code>). All specified strata variables must be factors.
na.rm	(logical(1)) Whether to remove incomplete rows from <code>df</code> (and, if supplied, <code>.ref_group</code>) before constructing the response, group, and strata vectors. Completeness is assessed only for <code>.var</code> and the strata columns (if supplied). If <code>na.rm = TRUE</code> , rows containing missing values in any of these columns are removed. If <code>na.rm = FALSE</code> , the function fails if any of these columns contain missing values, rather than returning results containing NA values. This argument is passed unchanged to the <code>complete_cases</code> argument of tern::h_prepare_rsp_table() .
alternative	(string) whether two.sided, or one-sided less or greater p-value should be displayed.
mf_method	(character(1)) The method used for testing the difference in response proportions when strata are provided and the Mantel-Fleiss criterion is satisfied. Available choices are "cmh", "cmh_sato", and "cmh_wh". See Details for more information.
mf_threshold	(numeric(1) or NULL) Threshold for the Mantel-Fleiss criterion. If NULL, the default from tern::mantel_fleiss_crit() is used.
...	additional arguments for the lower level functions.
.stats	(character) Statistics to select. The only available choice is "pval". This parameter affects only the formatted analysis function and has no effect on the statistics function.
.formats	(named character or list) Formats for the statistics.
.labels	(named character) labels for the statistics (without indent).
.indent_mods	(named integer) indent modifiers for the labels. Defaults to 0, which corresponds to the unmodified default behavior. Can be negative.
exact_footnote	(character(1) or RefFootnote) Footnote used to indicate that the unconditional exact method was actually used for the analysis.

Details

The data are prepared by `tern::h_prepare_rsp_table()`, which constructs the response vector, group indicator, optional strata variable, and the corresponding 2 x 2 contingency table(s). When multiple strata variables are supplied, the cross-combinations of their levels define the analysis strata.

The response variable is converted to a binary response according to `val`. The analysis compares the response proportion in the non-reference group with that in the reference group. The difference is calculated as the non-reference group minus the reference group.

When strata are provided and the Mantel-Fleiss criterion is satisfied, the selected CMH test `tern::prop_cmh()` is used to obtain the p-value for the test of the difference in response proportions. The available Cochran-Mantel-Haenszel tests are:

- "cmh": Stratified Cochran-Mantel-Haenszel test.
- "cmh_sato": Stratified CMH test using the Sato variance estimator.
- "cmh_wh": Stratified CMH test using the Wilson-Hilferty transformation.

See `tern::prop_cmh()` for more details and references.

When strata are not specified, or when strata are specified but the Mantel-Fleiss criterion is not satisfied or cannot be evaluated (e.g., when none of the strata contain any observed values), Fisher's exact test `tern::prop_fisher()` is used instead.

The Mantel-Fleiss criterion is evaluated using `tern::mantel_fleiss_crit()`.

The statistics function `s_test_proportion_diff_mf()` follows the usage and output of `tern::s_test_proportion_diff()` but removes the `method` argument and selects internally between a Cochran-Mantel-Haenszel-type test and Fisher's exact test based on the Mantel-Fleiss criterion.

Value

- `a_test_proportion_diff_mf()` returns the corresponding `RowsVerticalSection` object with formatted results.
- `s_test_proportion_diff_mf()` returns a list containing:
 - `pval` (`numeric(1)` or `numeric(0)`) P-value from the test, or `numeric(0)` when the function is evaluated for the reference column (`.in_ref_col` is `NULL` or `.in_ref_col` is `TRUE`).
 - `executed_method` (`character(1)`) Name of the test actually used for the analysis. Takes the value `NA_character_` when the function is evaluated for the reference column (`.in_ref_col` is `NULL` or `.in_ref_col` is `TRUE`). Otherwise, identifies the method used: "cmh", "cmh_sato", "cmh_wh", or "fisher". Formatted analysis functions can use this value to determine whether an appropriate footnote should be added.

Functions

- `a_test_proportion_diff_mf()`: Formatted analysis function used as `afun` to test the difference in response proportions, with method selection based on the Mantel-Fleiss criterion.
- `s_test_proportion_diff_mf()`: Statistics function testing the difference in response proportions. When strata variables are provided and the Mantel-Fleiss criterion is satisfied, the CMH test `tern::prop_cmh()` is used, with the values of the `diff_se` and `transform` arguments determined by `mf_method`. Otherwise, i.e., when strata variables are not provided, or

when strata variables are provided but the Mantel-Fleiss criterion is not satisfied or cannot be evaluated, Fisher's exact test `tern::prop_fisher()` is used.

Author(s)

WW

See Also

[proportion_diff_mf](#), [cond_proportion_j\(\)](#)

Examples

```
n <- 100 # Number of observations.
set.seed(123)
dta <- data.frame(
  "rsp" = sample(c("Y", "N"), n, TRUE),
  "grp" = c(
    sample(c("A", "Placebo"), round(n - 0.1 * n), TRUE),
    rep("C", round(0.1 * n))
  ),
  "f1" = sample(c("a1", "a2"), n, TRUE),
  "f2" = sample(c("x", "y", "z"), n, TRUE),
  stringsAsFactors = TRUE
)

lyt <- basic_table() |>
  split_cols_by(var = "grp", ref_group = "Placebo") |>
  analyze(
    vars = "rsp",
    afun = a_test_proportion_diff_mf,
    extra_args = list(
      val = "Y",
      variables = list(strata = c("f1", "f2"))
    )
  )

build_table(lyt, df = dta)

test_prop_d <- s_test_proportion_diff_mf(
  df = subset(dta, grp == "A"),
  .var = "rsp",
  .ref_group = subset(dta, grp == "Placebo"),
  .in_ref_col = FALSE,
  val = "Y",
  variables = list(strata = c("f1", "f2"))
)
test_prop_d
```

 theme_docx_default_j *Obtain the default theme for the docx*

Description

This function is based on `rtables.officer::theme_docx_default()`.

Usage

```
theme_docx_default_j(
  font = "Times New Roman",
  font_size = 9L,
  cell_margins = c(0, 0, 0, 0),
  bold = c("header", "content_rows", "label_rows", "top_left"),
  bold_manual = NULL,
  border = flextable::fp_border_default(width = 0.875, color = "black")
)
```

Arguments

font	(string) (optional) Default = "Times New Roman".
font_size	(integer(1)) (optional) Default = 9.
cell_margins	(numeric(1) or numeric(4)) a numeric or a vector of four numbers indicating c("left", "right", "top", "bottom"). (optional) Default = c(0, 0, 0, 0).
bold	(character) parts of the table text that should be in bold. Can be any combination of c("header", "content_rows", "label_rows", "top_left"). The first one renders all column names bold (not topleft content). The second and third option use <code>formatters::make_row_df()</code> to render content or/and label rows as bold. (optional) Default = c("header", "content_rows", "label_rows", "top_left").
bold_manual	(named list) list of index lists. Accepted groupings/names are c("header", "body"). See examples in <code>rtables.officer::theme_docx_default()</code> . (optional) Default = NULL.
border	(fp_border) border to use. Default = <code>flextable::fp_border_default(width = 0.875, color = "black")</code> .

Value

A function that applies the given theme to a flextable.

Note

This function may be removed from `junco` in the future if the functionality is merged into `rtables.officer`.

For more information, refer to the vignette `table_and_listing_customizations` (`browseVignettes("junco")`)

<code>tt_to_flextable_j</code>	<i>Convert a <code>TableTree</code> or a <code>listing_df</code> object to a flextable</i>
--------------------------------	--

Description

This function is based on `rtables.officer::tt_to_flextable()`.

Usage

```
tt_to_flextable_j(
  tt,
  tblid = NULL,
  theme = theme_docx_default_j(font = "Times New Roman", font_size = 9L, bold = NULL),
  border = flextable::fp_border_default(width = 0.875, color = "black"),
  titles_as_header = TRUE,
  bold_titles = TRUE,
  integrate_footers = TRUE,
  counts_in_newline = FALSE,
  paginate = tlg_type(tt) == "Table",
  fontspec = formatters::font_spec("Times", 9L, 1.2),
  colwidths = NULL,
  label_width_ins = 2,
  total_page_width = pg_width_by_orient(orientation == "landscape"),
  orientation = "portrait",
  nosplitin = list(row = character(), col = character()),
  string_map = default_str_map,
  markup_df_docx = dps_markup_df_docx,
  reduce_first_col_indentation = FALSE,
  tlgtype = tlg_type(tt),
  col_gap = ifelse(tlgtype == "Listing", 0.5, 3),
  round_type = formatters::obj_round_type(tt),
  alignments = list(),
  border_mat = make_header_bordmat(obj = tt),
  validate = TRUE,
  ...
)
```

Arguments

<code>tt</code>	(<code>TableTree</code> or <code>listing_df</code>) the object to convert to flextable.
-----------------	--

tblid	(character) output ID to be displayed in the title and last line of footer. When exporting, it will also be used as the output filename. If NULL, a temp file will be created, its dirname will replace argument output_dir, and its basename will replace argument tblid. (optional) Default = NULL.
theme	(function factory) the theme to apply to the flextable. (optional) Default = <code>theme_docx_default_j()</code> . See <code>theme_docx_default_j()</code> or <code>rtables.officer::theme_docx_default()</code> for more details.
border	(fp_border) border to use. Default = <code>flextable::fp_border_default(width = 0.875, color = "black")</code> .
titles_as_header	(logical) (optional) Default = TRUE.
bold_titles	(logical) (optional) Default = TRUE.
integrate_footers	(logical) (optional) Default = TRUE.
counts_in_newline	(logical) (optional) Default = FALSE.
paginate	(logical) (optional) Default = TRUE for TableTree and FALSE otherwise.
fontspec	(font_spec) font specification object.
colwidths	(numeric) column widths for the table. (optional) Default = NULL.
label_width_ins	(numeric) label width in inches. (optional) Default = 2.
total_page_width	(numeric) no need to be specified by end user. (optional) Default = 6.38 ("portrait") or 8.88 ("landscape").
orientation	(character) one of: "portrait", "landscape". (optional) Default = "portrait".
nosplitin	(list) path elements whose children should not be paginated within if it can be avoided.

	The list should have the format <code>list(row=, col=)</code>. E.g., <code>list(col="TRT01A")</code> means don't split within treatment arms unless all the associated columns don't fit on a single page. (optional) Default = <code>list(row = character(), col = character())</code>.
<code>string_map</code>	(tibble) (optional) Default = <code>default_str_map</code>.
<code>markup_df_docx</code>	(tibble) (optional) Default = <code>dps_markup_df_docx</code>.
<code>reduce_first_col_indentation</code>	(logical) whether to reduce by 1 the indentation if we have vertical pagination. No need to be specified by the end user. (optional) Default = <code>FALSE</code>.
<code>tlgtype</code>	(character) (optional) No need to be specified by end user.
<code>col_gap</code>	(numeric) (optional) Default = 3 (Tables) or 0.5 (Listings).
<code>round_type</code>	("iec" or "sas") the type of rounding to perform. iec, the default, performs rounding compliant with IEC 60559, while sas performs nearest-value rounding consistent with rounding within SAS. See <code>[formatters::format_value()]</code> for more details.
<code>alignments</code>	(list) list of named lists. Vectorized. Used to specify individual column or cell alignments. Each named list contains row, col, and value. (optional) Default = <code>list()</code>.
<code>border_mat</code>	(matrix) a $m \times k$ matrix where m is the number of columns of the input Table/Listing and k is the number of lines the header takes up. See tidytlg::add_bottom_borders for what the matrix should contain. Users should only specify this when the default behavior does not meet their needs.
<code>validate</code>	(logical(1)) Whether to validate the table structure using <code>rtables::validate_table_struct()</code>. Defaults to <code>TRUE</code>. If <code>FALSE</code>, a message will be displayed when validation fails.
<code>...</code>	other parameters.

Value

a flextable object.

Note

This function may be removed from `junco` in the future if the functionality is merged into `rtables.officer`.

For more information, refer to the vignette `table_and_listing_customizations` (`browseVignettes("junco")`)

Note

The following features are not implemented in `flextable`, and as a result they will only be visible when exporting to docx using `export_TLG_as_docx()`:

- watermark
- hanging indents
- caption style
- page numbering

tt_to_tbldf	Create TableTree as DataFrame via gentlg
-------------	--

Description

Create TableTree as DataFrame via gentlg

Usage

```
tt_to_tbldf(  
  tt,  
  fontspec = font_spec("Times", 9L, 1),  
  string_map = default_str_map,  
  markup_df = dps_markup_df,  
  round_type = obj_round_type(tt),  
  validate = TRUE  
)
```

Arguments

tt	(TableTree) TableTree object to convert to a data frame
fontspec	(font_spec) Font specification object
string_map	(list) Unicode mapping for special characters
markup_df	(data.frame) Data frame containing markup information
round_type	(character(1)) the type of rounding to perform. See <code>formatters::format_value()</code> for more details.
validate	logical(1). Whether to validate the table structure using <code>rtables::validate_table_struct()</code> . Defaults to TRUE. If FALSE, a message will be displayed instead of stopping with an error when validation fails.

Value

tt represented as a tbl data.frame suitable for passing to `tidytlg::gentlg` via the `huxme` argument.

tt_to_tlgtrtf	<i>TableTree to .rtf Conversion</i>
---------------	-------------------------------------

Description

A function to convert TableTree to .rtf

Usage

```
tt_to_tlgtrtf(  
  tt,  
  file = NULL,  
  orientation = c("portrait", "landscape"),  
  colwidths = def_colwidths(tt, fontspec, col_gap = col_gap, label_width_ins =  
    label_width_ins, type = tlgtype),  
  label_width_ins = 2,  
  watermark = NULL,  
  pagenum = ifelse(tlgtype == "Listing", TRUE, FALSE),  
  fontspec = font_spec("Times", 9L, 1.2),  
  pg_width = pg_width_by_orient(orientation == "landscape"),  
  margins = c(0, 0, 0, 0),  
  paginate = tlg_type(tt) == "Table",  
  col_gap = ifelse(tlgtype == "Listing", 0.5, 3),  
  nosplitin = list(row = character(), col = character()),  
  verbose = FALSE,  
  tlgtype = tlg_type(tt),  
  string_map = default_str_map,  
  markup_df = dps_markup_df,  
  combined_rtf = FALSE,  
  one_table = TRUE,  
  border_mat = make_header_bordmat(obj = tt),  
  round_type = obj_round_type(tt),  
  alignments = list(),  
  validate = TRUE,  
  export_csv = FALSE,  
  output_csv_directory = NULL,  
  ...  
)
```

Arguments

tt	(TableTree) TableTree object to convert to RTF
file	(character(1)) File to create, including path, but excluding .rtf extension.
orientation	(character) Orientation of the output ("portrait" or "landscape")

colwidths	(numeric vector) Column widths for the table
label_width_ins	(numeric) Label width in inches
watermark	(optional) String containing the desired watermark for RTF outputs. Vectorized.
pagenum	(logical) Whether to add page numbers to the output. Only applicable to listings (i.e. it is ignored for tables and figures).
fontspec	(font_spec) Font specification object
pg_width	(numeric) Page width in inches
margins	(numeric vector) Margins in inches (top, right, bottom, left)
paginate	(logical) Whether to paginate the output
col_gap	(numeric) Column gap in spaces
nosplitin	(list) list(row=, col=). Path elements whose children should not be paginated within if it can be avoided. e.g., list(col="TRT01A") means don't split within treatment arms unless all the associated columns don't fit on a single page.
verbose	(logical) Whether to print verbose output
tlgtype	(character) Type of the output (Table, Listing, or Figure)
string_map	(data.frame) Unicode mapping for special characters
markup_df	(data.frame) Data frame containing markup information
combined_rtf	(logical(1)) In the case where the result is broken up into multiple parts due to width, should a combined rtf file also be created. Defaults to FALSE.
one_table	(logical(1)) If tt is a (non-MatrixPrintForm) list, should the parts be added to the rtf within a single table (TRUE, the default) or as separate tables. End users will not generally need to set this.
border_mat	(matrix) A $m \times k$ matrix where m is the number of columns of <code>tt</code> and k is the number of lines the header takes up. See tidytlg::add_bottom_borders for what the matrix should contain. Users should only specify this when the default behavior does not meet their needs.

round_type	(character(1)) the type of rounding to perform. See <code>formatters::format_value()</code> for more details.
alignments	(list) List of named lists. Vectorized. (Default = <code>list()</code>) Used to specify individual column or cell alignments. Each named list contains row, col, and value, which are passed to <code>huxtable::set_align()</code> to set the alignments.
validate	(logical(1)) Whether to validate the table structure using <code>rtables::validate_table_struct()</code> . Defaults to TRUE. If FALSE, a message will be displayed when validation fails.
export_csv	(logical(1)) Whether to export the object as a csv representation. Default = FALSE.
output_csv_directory	(character(1)) the directory to export the csv. Default = NULL. Ignored if <code>export_csv = FALSE</code> . If NULL or attempting to export in a non-existent directory, the csv will be exported in the same directory as the .rtf file.
...	Additional arguments passed to <code>gentlg</code>

Details

This function aids in converting the `rtables` `TableTree` into the desired .rtf file.

Value

If `file` is non-NULL, this is called for the side-effect of writing one or more RTF files. Otherwise, returns a list of `huxtable` objects.

Note

`file` should always include path. Path will be extracted and passed separately to `gentlg`.

When `one_table` is FALSE, only the width of the row label pseudocolumn can be directly controlled due to a limitation in `tidytlg::gentlg`. The proportion of the full page that the first value in `colwidths` would take up is preserved and all other columns equally split the remaining available width. This will cause, e.g., the elements within the `allparts` rtf generated when `combined_rtf` is TRUE to differ visually from the content of the individual part rtfs.

Index

* Inclusion of ANCOVA Functions

a_summarize_ancova_j, 46
a_summarize_aval_chg_diff_j, 48
s_ancova_j, 186

* JJCS formatting functions

count and fraction related
formatting functions, 78
jjcsformat_fraction, 132
jjcsformat_xx, 134

* std_col_struct

grouped_cols_w_diffs, 121
make_multicomp_splfun, 142
shift_tbl_col_struct, 193

a_cmhrms_j (cmhrms), 70
a_cmhrms_j_with_exclude (cmhrms), 70
a_combo_prop_diff_pval_mf, 5
a_cond_proportion_j, 9
a_cond_proportion_j(), 11
a_coxph_hr, 13
a_diff_means (diff_means), 87
a_event_free (event_free), 91
a_freq_combos_j, 14
a_freq_j, 18
a_freq_j(), 7, 8, 17, 35, 123
a_freq_j_with_exclude (a_freq_j), 18
a_freq_resp_var_j, 30
a_freq_subcol_j, 32
a_kaplan_meier, 36
a_lsmeans (tabulate_lsmeans), 206
a_maxlev, 37
a_odds_ratio_j (odds_ratio), 145
a_proportion_ci_factor, 39
a_proportion_ci_logical, 40
a_proportion_ci_logical(), 40
a_proportion_diff_j (prop_diff), 152
a_proportion_diff_j(), 152–154
a_proportion_diff_mf
(proportion_diff_mf), 159
a_proportion_diff_mf(), 7, 8

a_rbmi_lsmeans (tabulate_rbmi), 208
a_relative_risk, 41
a_relative_risk(), 41
a_sum_ratio_j, 44
a_summarize_ancova_j, 46
a_summarize_ancova_j(), 52, 188
a_summarize_aval_chg_diff_j, 48
a_summarize_aval_chg_diff_j(), 47, 188
a_summarize_ex_j, 53
a_summarize_mmr (summarize_mmr), 199
a_summarize_mmr_with_exclude
(summarize_mmr), 199
a_summary_j, 57
a_summary_j_with_exclude (a_summary_j),
57
a_summary_subset (summary_subset), 203
a_test_proportion_diff
(prop_diff_test), 154
a_test_proportion_diff(), 154
a_test_proportion_diff_mf
(test_proportion_diff_mf), 210
a_test_proportion_diff_mf(), 7, 8
a_three_tier (a_two_tier), 59
a_two_tier, 59
ac_blank_line, 63
add_combo_levels(), 122, 195
analyze, 37, 59
analyze_values, 64

broom::tidy(), 207
bspt_pruner, 64
build_formula, 66

c_proportion_logical, 67
c_summary_subset_label
(summary_subset), 203
categorize_pval, 68
categorize_pval(), 175
check_wrap_nobreak, 69
cmhrms, 70

- cmp_cfun, 72
- cmp_post_fun, 73
- cmp_post_fun(), 72
- cmp_split_fun(cmp_post_fun), 73
- column_stats, 74
- column_stats(), 196
- compare_fmt_specs(fmt_utils), 114
- cond_proportion_j
 - (a_cond_proportion_j), 9
- cond_proportion_j(), 162, 214
- cond_rm_facets, 75
- count and fraction related formatting
 - functions, 78
- count_pruner, 80
- create_colspan_map, 81
- create_colspan_map(), 142
- create_colspan_var, 83
- cur_col_split_path
 - (cur_col_split_path_utils), 85
- cur_col_split_path_utils, 85
- d_cond_proportion_j
 - (a_cond_proportion_j), 9
- def_colwidths(listing_column_widths), 138
- diff_means, 87
- do_exclude_split, 90
- do_exclude_split(), 58
- emmeans::emmeans(), 51, 106, 108, 118, 187
- eval(), 204
- event_free, 91
- export_as_docx_j, 93
- export_graph_as_docx, 97
- export_TLG_as_docx, 98
- export_TLG_as_docx(), 219
- factor_by_order, 104
- find_missing_chg_after_avisit, 105
- fit_ancova, 106
- fit_ancova(), 140, 206
- fit_mmrj, 107
- fit_mmrj(), 67, 201, 206
- fmt_spec_d, 110
- fmt_spec_df_d(fmt_spec_d), 110
- fmt_spec_single_d(fmt_spec_d), 110
- fmt_spec_var_d(fmt_spec_d), 110
- fmt_utils, 114
- formatters::format_value(), 79, 133, 219, 222
- formatters::make_row_df(), 215
- get_fmt_details(fmt_utils), 114
- get_mmrj_lsmeans, 117
- get_ref_info, 118
- get_ref_info(), 50, 55, 71, 88, 89, 155, 207, 209
- get_titles_from_file, 120
- get_visit_levels, 121
- grouped_cols_w_diffs, 121
- grouped_cols_w_diffs(), 144, 194
- grouped_cols_w_subgrps
 - (grouped_cols_w_diffs), 121
- h_normalize_pvalcat(), 68
- h_odds_ratio, 125
- h_summarize_mmrj(), 201
- h_tidy_pool(tabulate_rbmi), 208
- huxtable::set_align(), 222
- in_column(cur_col_split_path_utils), 85
- in_column(), 7
- inches_to_spaces, 127
- insert_blank_line, 127
- jjcs_complex_scorefun, 128
- jjcs_num_formats, 132
- jjcsformat_cnt_den_fract_fct(count and fraction related formatting functions), 78
- jjcsformat_count_denom_fraction(count and fraction related formatting functions), 78
- jjcsformat_count_denom_fraction_legacy(count and fraction related formatting functions), 78
- jjcsformat_count_fraction(count and fraction related formatting functions), 78
- jjcsformat_fraction, 132
- jjcsformat_fraction(), 79, 135
- jjcsformat_fraction_count_denom(count and fraction related formatting functions), 78
- jjcsformat_fraction_count_denom_legacy(count and fraction related formatting functions), 78

- `jjcsformat_pval_fct` (`jjcsformat_xx`), 134
- `jjcsformat_range_fct` (`jjcsformat_xx`), 134
- `jjcsformat_xx`, 134
- `jjcsformat_xx()`, 79, 112, 133
- `junco_def_d_all` (`fmt_spec_d`), 110
- `junco_def_d_alt1` (`fmt_spec_d`), 110
- `junco_def_d_alt2` (`fmt_spec_d`), 110
- `junco_default_formats`, 112
-
- `keep_non_null_rows`, 136
-
- `leftside`, 137
- `list_valid_format_labels()`, 115
- `listing_column_widths`, 138
- `lsmeans_wide_cfun`
 - (`lsmeans_wide_first_split_fun_fct`), 139
- `lsmeans_wide_cfun()`, 141
- `lsmeans_wide_first_split_fun_fct`, 139
- `lsmeans_wide_first_split_fun_fct()`, 141
- `lsmeans_wide_second_split_fun_fct`
 - (`lsmeans_wide_first_split_fun_fct`), 139
- `lsmeans_wide_second_split_fun_fct()`, 141
-
- `make_combo_splitfun`, 141
- `make_dflt_comp_map`
 - (`make_multicomp_splfun`), 142
- `make_multicomp_splfun`, 142
- `make_multicomp_splfun()`, 122–124, 194
- `make_rbmi_cluster`, 145
- `make_rbmi_cluster()`, 164
- `mmrm::fit_mmrn()`, 110
- `mmrm::mmrm()`, 67, 109, 117, 168, 169
- `mmrm::mmrm_control()`, 109
-
- `odds_ratio`, 126, 145
- `officer::prop_section()`, 95, 100
- `officer::set_doc_properties()`, 95, 100
- `or_clogit_j` (`h_odds_ratio`), 125
- `or_clogit_j()`, 125
- `or_cmh` (`h_odds_ratio`), 125
- `or_glm_j` (`h_odds_ratio`), 125
-
- `par_lapply`, 149
- `parallel::makeCluster()`, 149
-
- `pool_rubin_scalar`, 150
- `pool_rubin_scalar()`, 151
- `pool_z_stat`, 151
- `pool_z_stat()`, 150
- `prepend_label_cell`, 151
- `prepend_label_cell()`, 205
- `prop_diff`, 152
- `prop_diff_test`, 154
- `prop_post_fun`, 156
- `prop_ratio_cmh`, 157
- `prop_split_fun` (`prop_post_fun`), 156
- `prop_split_fun()`, 158
- `prop_table_afun`, 158
- `proportion_diff_mf`, 159, 214
-
- `rbmi_analyse`, 163
- `rbmi_analyse()`, 164–166, 169
- `rbmi_ancova`, 165
- `rbmi_ancova()`, 166, 167
- `rbmi_ancova_single`, 166
- `rbmi_mmrn`, 168, 170
- `rbmi_mmrn()`, 170
- `rbmi_mmrn_single_info`, 170
- `rbmi_pool`, 170
- `real_add_overall_facet`, 171
- `relative_risk` (`a_relative_risk`), 41
- `remove_col_count`, 172
- `remove_rows`, 173
- `resp01_a_comp_stat_factor`, 175
- `resp01_a_comp_stat_factor()`, 177
- `resp01_a_comp_stat_logical`, 176
- `resp01_a_comp_stat_logical()`, 176
- `resp01_acfun`, 178
- `resp01_counts_cfun`, 180
- `resp01_split_fun_fct`, 181
- `resp_multiple_imputation`, 174
- `response_by_var`, 182
- `rightside`, 184
- `rm_levels`, 185
- `rm_other_facets_fact`, 185
- `rtables`, 58, 85, 88, 203, 204
- `rtables.officer::export_as_docx()`, 93
- `rtables.officer::theme_docx_default()`, 95, 100, 215, 217
- `rtables.officer::tt_to_flexitable()`, 216
- `rtables::add_combo_levels()`, 82
- `rtables::additional_fun_params`, 16, 22, 35, 38, 60, 118, 119

- `rtables::analyze`, 74
- `rtables::analyze()`, 44, 89, 112, 203
- `rtables::basic_table()`, 140
- `rtables::CellValue()`, 11, 42, 47, 56, 147, 153, 155, 201
- `rtables::col_paths()`, 6, 85, 86
- `rtables::in_rows`, 159, 207
- `rtables::in_rows()`, 68, 72, 89, 151, 179, 181, 189
- `rtables::make_split_fun()`, 73, 122, 123, 157, 182, 195–197
- `rtables::make_split_result()`, 197
- `rtables::rcell()`, 7, 40, 41, 176, 177
- `rtables::split_cols_by_multivar()`, 196
- `rtables::split_rows_by`, 73, 157
- `rtables::summarize_row_groups`, 203
- `rtables::summarize_row_groups()`, 14, 33, 38, 60, 63, 72, 140, 179, 181, 202–204
- `rtables::trim_levels_to_map()`, 81, 82, 123

- `s_ancova_j`, 186
- `s_ancova_j()`, 47, 52
- `s_cmhrms_j` (`cmhrms`), 70
- `s_cond_proportion_j` (`a_cond_proportion_j`), 9
- `s_diff_means` (`diff_means`), 87
- `s_diff_means()`, 190, 191
- `s_event_free` (`event_free`), 91
- `s_freq_j` (`a_freq_j`), 18
- `s_lsmeans` (`tabulate_lsmeans`), 206
- `s_lsmeans()`, 201
- `s_odds_ratio_j` (`odds_ratio`), 145
- `s_odds_ratio_j()`, 125
- `s_proportion_diff_mf` (`proportion_diff_mf`), 159
- `s_proportion_factor`, 188
- `s_proportion_factor()`, 189
- `s_proportion_logical`, 189
- `s_proportion_logical()`, 68, 189
- `s_rbmi_lsmeans` (`tabulate_rbmi`), 208
- `s_relative_risk` (`a_relative_risk`), 41
- `s_summarize_ancova_j` (`a_summarize_ancova_j`), 46
- `s_summarize_ex_j` (`a_summarize_ex_j`), 53
- `s_summarize_mmrn` (`summarize_mmrn`), 199
- `s_summary_diff`, 190

- `s_test_proportion_diff_mf` (`test_proportion_diff_mf`), 210
- `safe_prune_table`, 191
- `set_titles`, 193
- `set_titles()`, 120
- `shift_tbl_col_struct`, 193
- `shift_tbl_col_struct()`, 124, 144
- `some_v_all_col_struct`, 194
- `stats::glm()`, 126
- `stats::lm()`, 107, 166
- `stats_in_cols_setup`, 196
- `stats_in_cols_setup()`, 74
- `string_to_title`, 197
- `summarize_coxreg_multivar`, 198
- `summarize_lsmeans_wide` (`lsmeans_wide_first_split_fun_fct`), 139
- `summarize_mmrn`, 199
- `summarize_row_counts`, 202
- `summarize_row_groups`, 37
- `summary_subset`, 203
- `survival::clogit()`, 125, 126, 147

- `tabulate_lsmeans`, 206
- `tabulate_rbmi`, 208
- `tern::a_odds_ratio()`, 147
- `tern::a_proportion_diff()`, 154
- `tern::a_summary()`, 57, 58, 203–205
- `tern::a_surv_time()`, 36
- `tern::a_surv_timepoint`, 92
- `tern::format_count_fraction_fixed_dp()`, 78
- `tern::get_stats()`, 190
- `tern::h_prepare_rsp_table()`, 160, 161, 212, 213
- `tern::mantel_fleiss_crit()`, 6, 161, 162, 212, 213
- `tern::prop_cmh()`, 213
- `tern::prop_diff_cmh()`, 161, 162
- `tern::prop_diff_uncond_exact()`, 161, 162
- `tern::prop_fisher()`, 213, 214
- `tern::s_odds_ratio()`, 147
- `tern::s_proportion()`, 11, 40
- `tern::s_proportion_diff()`, 22, 162
- `tern::s_summary`, 46, 47
- `tern::s_summary()`, 190, 191
- `tern::s_surv_timepoint()`, 91–93

tern::s_test_proportion_diff(), [155](#),
[213](#)
test_proportion_diff_mf, [210](#)
theme_docx_default_j, [215](#)
theme_docx_default_j(), [95](#), [100](#), [217](#)
tidy.tern_model (tabulate_lsmeans), [206](#)
tidytlg::add_bottom_borders, [96](#), [101](#),
[218](#), [221](#)
tidytlg::gentlg, [219](#)
tt_to_flextable_j, [216](#)
tt_to_tbldf, [219](#)
tt_to_tlgtrf, [220](#)