Package 'dtlg'

May 8, 2026

Title A Performance-Focused Package for Clinical Trial Tables

Version 0.1.0

Description Create high-performance clinical reporting tables (TLGs) from ADaM-like inputs. The package provides a consistent, programmatic API to generate common tables such as demographics, adverse event incidence, and laboratory summaries, using 'data.table' for fast aggregation over large populations. Functions support flexible target-variable selection, stratification by treatment, and customizable summary statistics, and return tidy, machine-readable results ready to render with downstream table/formatting packages in analysis pipelines.

License MIT + file LICENSE

Imports data.table (>= 1.18.0), vctrs

Suggests bench, dplyr, kableExtra, rmarkdown, random.cdisc.data, rtables, tern, testthat (>= 3.0.0), tidyr, withr

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Config/testthat/edition 3

Config/Needs/website rmarkdown, ascentsoftware/ascentdown

LazyData true

URL https://AscentSoftware.github.io/dtlg/

BugReports https://github.com/AscentSoftware/dtlg/issues

NeedsCompilation no

Author Max Ebenezer-Brown [aut],
Max Norman [aut],
Xinye Li [aut],
Anja Peebles-Brown [aut],
Ashley Baldry [aut],
Ramiro Magno [aut, cre]

Maintainer Ramiro Magno <ramiro.morgado@acuityanalytics.com>

Repository CRAN

Date/Publication 2026-04-29 09:10:07 UTC

Contents

---

adae *Adverse Event Analysis Dataset example dataset*

Description

adae is a re-export of the random.cdisc.data::cadae dataset, included in `{dtlg}` for function usage illustration and testing.

Usage

```
adae
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 1934 rows and 92 columns.

Examples

```
adae
```

`adlb` *ADaM Basic Data Structure (BDS) example dataset*

Description

adlb is a re-export of the random.cdisc.data::cadlb dataset, included in `{dtlg}` for function usage illustration and testing.

Usage

```
adlb
```

Format

An object of class `data.table` (inherits from `data.frame`) with 8400 rows and 102 columns.

Examples

```
adlb
```

`adsl` *Subject-Level Analysis Dataset (ADSL) example dataset*

Description

adsl is a re-export of the random.cdisc.data::cadsl dataset, included in `{dtlg}` for function usage illustration and testing.

Usage

```
adsl
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 400 rows and 55 columns.

Examples

```
adsl
```

aesi *Adverse Events of Special Interest (AESI) example dataset*

Description

aesi is a modified version of the random.cdisc.data::cadae dataset, filtered to include only analysis-flagged records (`ANL01FL == "Y"`) and extended with binary indicator variables corresponding to adverse events of special interest (AESIs).

Usage

```
aesi
```

Format

A `data.frame` with a subset of rows from `cadae` and additional derived columns including:

FATAL Logical flag for fatal AEs (`AESDTH == "Y"`).

SEV Logical flag for severe AEs (`AESEV == "SEVERE"`).

SER Logical flag for serious AEs (`AESER == "Y"`).

SERWD Serious AE leading to withdrawal (`AESER == "Y" & AEACN == "DRUG WITHDRAWN"`).

SERDSM Serious AE leading to dose modification/interruption.

RELSER Serious and related AE.

WD AE leading to withdrawal.

DSM AE leading to dose modification/interruption.

REL Related AE.

RELWD Related AE leading to withdrawal.

RELDSM Related AE leading to dose modification/interruption.

Details

These derived flags include seriousness, severity, fatality, relatedness, and treatment consequence (e.g., dose modification or withdrawal), and are used to illustrate key safety summaries in clinical reporting.

Each derived variable is labeled using `with_label()` for compatibility with tabulation functions.

This dataset is included in `{dtlg}` to support function testing, usage examples, and reproducible safety analyses.

See Also

random.cdisc.data::cadae, `multi_event_true()`

Examples

```
aesi
```

AET01_table *Generate Core Safety Tables for Clinical Study Reports*

Description

AET01_table() produces and combines the main safety summary tables typically found in Section 14.3.1 of a Clinical Study Report (CSR). It calculates patient counts and event totals for deaths, AE-related withdrawals, total AEs, and adverse events of special interest (AESIs).

Usage

```
AET01_table(
  adsl,
  adae,
  patient_var,
  treat_var,
  aesi_vars,
  aesi_heading = "Total number of patients with at least one",
  indent = "  "
)
```

Arguments

adsl	A subject-level dataset (typically ADaM ADSL).
adae	A dataset of adverse events, preprocessed with AESI flags.
patient_var	A string indicating the subject identifier variable (e.g., "USUBJID").
treat_var	A string indicating the treatment arm variable (e.g., "ARM").
aesi_vars	A character vector of binary AESI flags in adae.
aesi_heading	Optional character string used as a heading in the AESI block.
indent	A string used to indent AESI row labels (default is 2 spaces).

Value

A merged data.table summarising the main safety outcomes.

Examples

```
AET01_summary <- AET01_table(
  adsl = adsl,
  adae = aesi,
  patient_var = "USUBJID",
  treat_var = "ARM",
  aesi_vars = c("FATAL", "SER", "SERWD", "SERDSM", "RELSER",
                "WD", "DSM", "REL", "RELWD", "RELDSM", "SEV")
)
print(AET01_summary)
```

AET02_table *Create AET02-style AE summary table*

Description

Summarises adverse events in a format similar to the AET02 table from a CSR, showing total AE counts, patients with AEs, and a breakdown by System Organ Class (SOC) and Preferred Term (PT).

Usage

```
AET02_table(
  adsl,
  adae,
  patient,
  treat,
  target = "AEDECOD",
  rows_by = "AEBODSYS",
  indent = nbsp(n = 4L)
)
```

Arguments

adsl	Subject-level dataset.
adae	Adverse event dataset.
patient	Unique subject identifier variable.
treat	Treatment arm variable.
target	Preferred term variable for grouping (default: "AEDECOD").
rows_by	Higher-level term for nesting (default: "AEBODSYS").
indent	Character or string to indent nested rows (default: 4 non-breaking spaces).

Value

A merged data.table containing AE summary.

Examples

```
# Create a AET02 table
AET02_table(
  adsl = adsl,
  adae = aesi,
  patient = "USUBJID",
  treat = "ARM",
  target = "AEDECOD",
  rows_by = "AEBODSYS",
  indent = "  "
)
```

as_dtlg_table *Convert a TableTree to a dtlg table*

Description

`as_dtlg_table()` reformats a TableTree object into a format close to that of dtlg's `data.table`.

Usage

```
as_dtlg_table(tt, .label_col = "stats")
```

Arguments

`tt`	A TableTree object. Typically obtained with `tern_summary_table()`.
`.label_col`	Label for stats' column.

Value

A data.table.

Examples

```
vars <- c('AGE', 'RACE', 'ETHNIC', 'BMRKR1')
var_labels <- c("Age (yr)", "Race", "Ethnicity", "Continuous Level Biomarker 1")

# Summary statistics table split by ARM with custom labels.
(tt <- tern_summary_table(
  adsl,
  target = vars,
  treat = 'ARM',
  target_name = var_labels
))

# Format as a dtlg table
as_dtlg_table(tt)
```

calc_counts *Calculate counts of a categorical variable*

Description

`calc_counts()` counts observations of a categorical variable (`target`) by another (`treat`) and reports summary statistics in clinical trial reporting format.

Usage

```
calc_counts(
  dt,
  target,
  target_name = target,
  treat,
  indent = nbsp(n = 4L),
  .total_dt = NULL,
  pct_dec = 1
)
```

Arguments

dt	A data.frame containing, at least, the variables indicated in target and treat.
target	Target variable passed as a string for which summary statistics are to be calculated.
target_name	Heading for the target variable as a string. Defaults to target.
treat	A string indicating the grouping variable, e.g. the variable specifying the treatment population.
indent	A string to be used as indentation of summary statistics labels. Defaults to four HTML non-breaking spaces ( ).
.total_dt	Separate table from dt from which to derive total counts per group.
pct_dec	This argument is ignored, and is only kept for backward compatibility reasons.

Value

A list containing a data.table formatted as follows:

- First column is named stats and contains the target variable name indicated in target in the first row. Subsequent rows contain the levels of target.
- Other columns are for the levels of the grouping variable (treat).
- All columns are of character type.

This table is structured for easy integration with Shiny output widgets.

Examples

```
calc_counts(dt = adsl, "RACE", treat = "ARM", indent = "  ")[[1]]
```

calc_desc *Calculate descriptive summary statistics for a numeric variable*

Description

calc_desc() summarises a numeric variable (target) by another (treat) and reports summary statistics in clinical trial reporting format. The following statistics are calculated for target, per group, i.e. by variable treat levels:

- n: number of observations
- Mean (SD): mean and standard deviation of target
- Median: median of target
- Min, Max: minimum and maximum of target
- Missing: number of missing target values

Usage

```
calc_desc(
  dt,
  target,
  target_name = target,
  treat,
  indent = nbsp(n = 4L),
  pct_dec = 1,
  inc_missing = TRUE
)
```

Arguments

dt	A data.frame containing, at least, the variables indicated in target and treat.
target	Target variable passed as a string for which summary statistics are to be calculated.
target_name	Heading for the target variable as a string. Defaults to target.
treat	A string indicating the grouping variable, e.g. the variable specifying the treatment population.
indent	A string to be used as indentation of summary statistics labels. Defaults to four HTML non-breaking spaces ( ).
pct_dec	Decimal places for reported figures.
inc_missing	Toggle for including the "Missing" row: TRUE (default) The Missing row is always displayed NA The Missing row is only displayed if any missing values are present FALSE The Missing row is never included in the table

Value

A list containing a `data.table` formatted as follows:

- First column is named `stats` and contains the target variable name indicated in `target` in the first row. Subsequent rows contain the summarised statistics labels.
- Other columns are for the levels of the grouping variable (`treat`).
- All columns are of character type.

This table is structured for easy integration with Shiny output widgets.

Examples

```
# Calculate summary statistics for the age of the subjects in each region.
calc_desc(dt = adsl, "AGE", treat = "REGION1")[[1]]

# Calculate summary statistics for biomarker 1 in each of the three arms
# (`ARM`).
calc_desc(dt = adsl, "BMRKR1", treat = "ARM")[[1]]
```

`calc_stats` *Calculate summary statistics for a variable*

Description

`calc_stats()` calculates summary statistics for a variable on groups. This is a generic function; note that it dispatches based on the class of `target` (second argument), not `dt` (first argument).

Usage

```
calc_stats(
  dt,
  target,
  treat,
  ...,
  target_name = target,
  indent = nbsp(n = 4L),
  .total_dt = NULL,
  pct_dec = 1
)
```

Arguments

`dt`	A `data.frame` containing, at least, the variables indicated in `target` and `treat`.
`target`	Target variable passed as a string for which summary statistics are to be calculated.

treat	A string indicating the grouping variable, e.g. the variable specifying the treatment population.
...	Additional arguments passed to other methods
target_name	Heading for the target variable as a string. Defaults to target.
indent	A string to be used as indentation of summary statistics labels. Defaults to four HTML non-breaking spaces ( ).
.total_dt	Separate table from dt from which to derive total counts per group.
pct_dec	Decimal places for reported figures.

Value

A data.table of summary statistics. The format depends on the type of the target variable:

- If the target variable is categorical, i.e. type character, factor or logical then the output is that of calc_counts().
- If the target variable is numeric, then the output is that of calc_desc().

Examples

```
# Calculate summary statistics of a numeric variable, e.g. `AGE`.
calc_stats(dt = adsl, "AGE", treat = "ARM")[[1]]

# Calculate summary statistics of a categorical variable, e.g. `SEX`.
calc_stats(dt = adsl, "SEX", treat = "ARM")[[1]]
```

---

cross_tab_to_obsv_tab *Convert a contingency table to a long-format observation-level data frame*

Description

cross_tab_to_obsv_tab() expands a contingency table or matrix of counts into a long-format data frame where each row represents one observation. The output contains one column per dimension of the input, with repeated rows according to the frequency counts.

Usage

```
cross_tab_to_obsv_tab(cross_tab, strings_as_factors = TRUE)
```

Arguments

cross_tab	A two-way or multi-way contingency table (matrix or table) with named dimnames. Each combination of factor levels is assumed to represent a count of occurrences.
strings_as_factors	Should character columns in the output be converted to factors?

Value

A `data.frame` in long format with one row per implied observation and one column per dimension of the input table.

Examples

```
dim_names <- list(Sex = c("Male", "Female"),
                  Response = c("Yes", "No"))
cross_tab <- matrix(c(2, 1, 3, 4), nrow = 2, dimnames = dim_names)
cross_tab_to_obsv_tab(cross_tab)
```

`dt_copy_semantics` *Get or set data.table copy semantics*

Description

These functions control how `maybe_copy_dt()` decides whether to return a `data.table` by reference (in place) or by value (as a deep copy).

Usage

```
dt_copy_semantics()

set_dt_copy_semantics(dt_copy_semantics = c("reference", "value"))
```

Arguments

`dt_copy_semantics`
: Character string. Either `"reference"` or `"value"`.

Details

The copy semantics are stored in the global option `dtlg_dt_copy_semantics`. The option can take two values:

- `"reference"` (default): inputs are treated with reference semantics.
  - If the input is already a `data.table`, it is returned unchanged and aliases are preserved.
  - If the input is a `data.frame`, it is converted to a `data.table` in place via `data.table::setDT()`, mutating the caller's object.
- `"value"`: inputs are treated with value semantics.
  - The input is converted to a `data.table` (if necessary) and a deep copy is returned, leaving the original unchanged.

Value

- `dt_copy_semantics()` returns the current semantics as a string, `"reference"` or `"value"`.
- `set_dt_copy_semantics()` sets the semantics, returning the previous semantics invisibly.

See Also

maybe_copy_dt()

Examples

```
# Get current semantics (defaults to "reference")
dt_copy_semantics()

# Switch to value semantics
old <- set_dt_copy_semantics("value")
dt_copy_semantics()

# Restore previous semantics
set_dt_copy_semantics(old)
```

---

event_count *Count events*

Description

event_count() counts events defined by predicate expressions passed in .filters.

Usage

```
event_count(
  dt,
  patient,
  treat,
  label,
  .filters = NULL,
  .total_dt = dt,
  pct_dec = 1
)
```

Arguments

dt	A data.frame containing, at least, the variables indicated in target and treat.
patient	A string indicating the subject identifying variable.
treat	A string indicating the grouping variable, e.g. the variable specifying the treatment population.
label	A string to be used as label in the output reporting table. This should be a text descriptive of the event being counted.
.filters	Predicate expressions identifying events in dt. Argument should be passed as a character vector of expressions to be evaluated in the frame of dt.
.total_dt	Separate table from dt from which to derive total counts per group.
pct_dec	This argument is ignored, and is only kept for backward compatibility reasons.

Value

A one-element list, where the element is a `data.table`.

Examples

```
# Count deaths per arm.
event_count(
  adsl,
  patient = "USUBJID",
  treat = "ARM",
  label = "Total number of deaths",
  .filters = "DTHFL == 'Y'"
)[[1]]

# Count patients withdraw from study due to an adverse event.
withdrawn_lbl <- "Total number of patients withdrawn from study due to an AE"
event_count(
  adsl,
  patient = "USUBJID",
  treat = "ARM",
  label = withdrawn_lbl,
  .filters = "DCSREAS == 'ADVERSE EVENT'"
)[[1]]

# Count patients with at least one adverse event.
# NB: When `.filters` is `NULL` (i.e., omitted), all records in `dt` are used
# for counting events.
event_count(
  adae,
  patient = "USUBJID",
  treat = "ARM",
  label = "Total number of patients with at least one AE",
  .filters = "ANL01FL == 'Y'",
  .total_dt = adsl
)[[1]]
```

---

event_count_by *Summarise adverse events by arm and other grouping variables*

Description

`event_count_by()` creates a tabular summary of adverse events grouped by a higher-level classification variable (e.g., system organ class), and counts both the number of events and the number of unique patients per treatment arm.

Usage

```
event_count_by(
  dt,
  patient,
  treat,
  rows_by,
  target,
  .total_dt = dt,
  indent = nbsp(n = 4L),
  pct_dec = 1
)
```

Arguments

dt	A data.frame or data.table containing the adverse event data and patient-level identifiers.
patient	A string giving the name of the patient identifier variable (e.g., "USUBJID").
treat	A string giving the name of the treatment arm variable (e.g., "ARM").
rows_by	A string giving the name of the grouping variable (e.g., "AEBODSYS" for body system).
target	A string giving the name of the variable to report within each group (e.g., "AEDECOD" for preferred term).
.total_dt	A data.frame or data.table containing the denominator population. Defaults to dt.
indent	A string used to indent row labels (e.g., " " or nbsp(n = 4L)).
pct_dec	Integer. Number of decimal places to show in percentages. Defaults to 1.

Value

A data.table with the following structure:

- One row per combination of rows_by and target
- One row per group total (Total number of events)
- One row per patient-level total (Total number of patients with at least one event)

Columns include:

- stats: character column with labels
- one column per level of the treat variable, formatted as "n (x%)"

See Also

event_count(), calc_stats(), total_events()

Examples

```
event_count_by(
  dt = adae[adae$ANL01FL == "Y"],
  patient = 'USUBJID',
  treat = 'ARM',
  rows_by = 'AEBODSYS',
  target = 'AEDECOD',
  .total_dt = adsl,
  indent = ' '
)
```

---

label *Retrieve the label of an object*

Description

`label()` gets the attached label to an object.

Usage

```
label(x)
```

Arguments

x	An R object.

Value

The label attribute (string) associated with object passed in `x` or `NULL` if the label attribute does not exist.

See Also

`with_label()`

Examples

```
label(1)
label(with_label(1, "my label"))
```

maybe_copy_dt *Return a data.table by reference or by value*

Description

`maybe_copy_dt()` returns its input as a `data.table`, with behaviour controlled by the global copy semantics option `dt_copy_semantics()`.

Usage

```
maybe_copy_dt(x)
```

Arguments

`x`	A `data.table` or `data.frame`.

Details

- If the semantics are `"reference"` (default):
  - If `x` is already a `data.table`, it is returned unchanged. Aliasing holds, so mutations with `:=` will affect both input and output.
  - If `x` is a `data.frame`, it is converted to a `data.table` in place via `data.table::setDT()`, mutating the caller's object. The returned object is a `data.table` with the same contents. For efficiency, the column vectors are reused without a deep copy.
- If the semantics are `"value"`:
  - `x` is converted to a `data.table` (if necessary) and a deep copy is returned. Mutating the result does not affect the input.

Value

A `data.table`. Whether the return value aliases the input depends on the semantics:

- `"reference"`: input is mutated in place, aliasing guaranteed if `x` is already a `data.table`.
- `"value"`: a fresh copy is returned, independent of the input.

See Also

`dt_copy_semantics()`, `set_dt_copy_semantics()`

Examples

```
# Default: reference semantics
df <- data.frame(a = 1:3)
out <- maybe_copy_dt(df)
data.table::is.data.table(df) # TRUE, converted in place

# Switch to value semantics
```

```
old <- set_dt_copy_semantics("value")
dt <- data.table::data.table(a = 1:3)
out2 <- maybe_copy_dt(dt)
out2[, b := 99L]
"b" %in% names(dt)  # FALSE, original unchanged

# Restore previous semantics
set_dt_copy_semantics(old)
```

`merge_table_lists` *Merge a list of list-wrapped data.tables into one data.table*

Description

This function is typically used to combine multiple reporting tables, each produced by `event_count()`, `total_events()`, or `multi_event_true()`, into a single summary table. These intermediate tables are often returned as one-element lists containing a `data.table`.

Usage

```
merge_table_lists(dt_l)
```

Arguments

`dt_l`	A list of one-element lists, where each element is a list containing a single `data.table`.

Details

This helper unwraps and merges them, row-wise, to produce a consolidated safety report table — commonly used in clinical study reports or data monitoring reviews.

Value

A single merged `data.table`, row-bound from all input tables.

Examples

```
# Count deaths by treatment arm
death_table <- event_count(
  adsl,
  patient = "USUBJID",
  treat = "ARM",
  label = "Total number of deaths",
  .filters = "DTHFL == 'Y'"
)

# Count study withdrawals due to adverse events
```

```
withdrawal_table <- event_count(
  adsl,
  patient = "USUBJID",
  treat = "ARM",
  label = "Total number of patients withdrawn from study due to an AE",
  .filters = "DCSREAS == 'ADVERSE EVENT'"
)

# Count patients with at least one adverse event
patients_with_ae_table <- event_count(
  adae,
  patient = "USUBJID",
  treat = "ARM",
  label = "Total number of patients with at least one AE"
)

# Count total number of adverse events (not patients)
total_ae_events_table <- total_events(
  dt = adae,
  treat = "ARM",
  label = "Total number of AEs"
)

# Summarise AESIs (e.g., serious, related, severe, etc.)
aesi_vars <- c("FATAL", "SER", "SERWD", "SERDSM", "RELSER",
               "WD", "DSM", "REL", "RELWD", "RELDSM", "SEV")

aesi_table <- multi_event_true(
  dt = aesi,
  event_vars = aesi_vars,
  patient = "USUBJID",
  treat = "ARM",
  heading = "Total number of patients with at least one",
  .total_dt = adsl,
  indent = "  "
)

# Combine all safety tables into a single summary table
safety_summary <- merge_table_lists(list(
  patients_with_ae_table,
  total_ae_events_table,
  death_table,
  withdrawal_table,
  aesi_table
))

safety_summary
```

`multi_event_true` *Summarise multiple AESI-like events per treatment arm*

Description

multi_event_true() generates a summary table showing the number and percentage of patients with at least one event across multiple binary indicator variables (e.g., flags for adverse events of special interest).

Usage

```
multi_event_true(
  dt,
  event_vars,
  patient,
  treat,
  heading,
  label = NULL,
  .total_dt = NULL,
  indent = nbsp(n = 4L),
  pct_dec = 1
)
```

Arguments

dt	A data.frame or data.table containing the binary event flags and subject-level data.
event_vars	A character vector of column names (binary flags) to summarise.
patient	A string giving the name of the variable that uniquely identifies each patient (e.g., "USUBJID").
treat	A string giving the name of the treatment variable (e.g., "ARM").
heading	A string to be shown as the first row in the output, usually a summary descriptor such as "Total number of patients with at least one".
label	Optional. A character vector of the same length as event_vars giving human-readable labels for the output table rows. If NULL, labels are extracted from the label attribute of each variable, or fall back to the variable name.
.total_dt	A data.frame or data.table containing the total analysis population (denominator). If NULL, dt is used as the denominator.
indent	A string to indent the row labels (e.g., " " or nbsp(n = 4L) for non-breaking spaces).
pct_dec	An integer indicating how many decimal places to show in percentages (default is 1).

Details

Each event is counted only once per patient. This function is typically used for summarising *Adverse Events of Special Interest* (AESIs) or other derived flags (e.g., SER, FATAL, RELDSM) that are binary (TRUE/FALSE).

Value

A one-element list containing a `data.table` with one row per event plus one header row. The first column is `"stats"` (row labels), and subsequent columns are one per treatment arm, with values in `"n (x%)"` format.

See Also

`event_count()`, `total_events()`

Examples

```
aesi_vars <- c(
  "FATAL", "SER", "SERWD", "SERDSM", "RELSER",
  "WD", "DSM", "REL", "RELWD", "RELDSM", "SEV"
)

heading <- "Total number of patients with at least one AE"

multi_event_true(
  dt = aesi,
  event_vars = aesi_vars,
  patient = "USUBJID",
  treat = "ARM",
  heading = heading,
  .total_dt = adsl,
  indent = "  "
)[[1]]
```

---

print_dtlg *Print a* `dtlg` *table*

Description

A convenience wrapper around `print()` for printing `dtlg` tables with consistent formatting options.

Usage

```
print_dtlg(
  dt,
  row.names = FALSE,
  trunc.cols = TRUE,
  class = FALSE,
  nrows = Inf,
  justify = "left"
)
```

Arguments

dt	A dtlg table, typically a data.frame or data.table.
row.names	If TRUE, row indices will be printed alongside x.
trunc.cols	If TRUE, only the columns that can be printed in the console without wrapping the columns to new lines will be printed (similar to tibbles).
class	If TRUE, the resulting output will include above each column its storage class (or a self-evident abbreviation thereof). When combined with col.names="auto" and tables >20 rows, classes will also appear at the bottom.
nrows	The number of rows which will be printed before truncation is enforced.
justify	String. Column alignment; one of "left", "right", "centre", or "none". Defaults to "left".

Value

Invisibly returns the printed object.

Examples

```
calc_stats(dt = adsl, "AGE", treat = "ARM", indent = "  ")[[1]] |>
  print_dtlg()
```

---

round_pct *Rounded percentage*

Description

round_pct() returns the rounded percentages of x values.

Usage

```
round_pct(x, digits = 1L, method = c("round", "round_sum"))
```

Arguments

x	A numeric vector of non-negative values for which you want percentages to be determined and rounded. Missing values (NA) are ignored.
digits	The number of decimal places to round to. Default is 0 (integer rounding).
method	Rounding method: "round" that uses R's base round() or "round_sum" that uses dtlg::round_sum.

Value

A numeric vector of the same length as x with rounded percentages.

Examples

```
x <- c(1 / 3, 1 / 3, 1 / 3)

# Default method ensures precise rounding but total might not be 100%.
round_pct(x = x)
sum(round_pct(x = x))

# You can trade off rounding precision for precision on the total with the
# method `"round_sum"`.
round_pct(x = x, method = "round_sum")
sum(round_pct(x = x, method = "round_sum"))

# Vary the number of decimal places, e.g. increase to three.
round_pct(x = x, digits = 3, method = "round_sum")

# Missing values are ignored.
x <- c(1, 2, NA)
round_pct(x = x, digits = 3)
```

---

round_sum *Rounds numbers while preserving the total sum*

Description

`round_sum()` rounds a numeric vector of non-negative values to a specified number of decimal places while ensuring that the sum of the rounded value remains as close as possible to the original total.

Usage

```
round_sum(x, digits = 0L)
```

Arguments

`x`	A numeric vector of non-negative values that you want to round. Missing values (`NA`) are ignored.
`digits`	The number of decimal places to round to. Default is `0` (integer rounding).

Value

A numeric vector of the same length as `x`, with values rounded in such a way that the total sum is preserved.

Examples

```
# Rounds to integers, preserving the sum of 100.
x <- c(33.3333, 33.3333, 33.3334)
(y <- round_sum(x))
identical(sum(x), sum(y))

# Rounds to integers, preserving the sum of 1002.
x <- c(100.5, 200.25, 300.75, 400.5)
(y <- round_sum(x))
identical(sum(x), sum(y))

# Rounds to one decimal place, preserving the total sum.
x <- c(12.345, 67.890, 19.765)
(y <- round_sum(x))
identical(sum(x), sum(y))
```

---

summary_table *Summary Table*

Description

summary_table() summarises clinical variables into a report table using data.table as backend.

Usage

```
summary_table(
  dt,
  target,
  treat,
  target_name = target,
  indent = nbsp(n = 4L),
  .total_dt = dt,
  pct_dec = 1,
  treat_order = NULL,
  skip_absent = TRUE,
  inc_missing = TRUE
)
```

Arguments

dt	A data.frame containing, at least, the variables indicated in target and treat.
target	Target variable passed as a string for which summary statistics are to be calculated.
treat	A string indicating the grouping variable, e.g. the variable specifying the treatment population.
target_name	Heading for the target variable as a string. Defaults to target.

indent	A string to be used as indentation of summary statistics labels. Defaults to four HTML non-breaking spaces ( ).
.total_dt	Separate table from dt from which to derive total counts per group.
pct_dec	Decimal places for reported figures.
treat_order	Customise the column order of the output table.
skip_absent	Whether to ignore variables passed in treat_order that are absent from dt. Default is TRUE; FALSE will throw an error in case there are missing variables.
inc_missing	If any of the target variables are numeric, then used as a toggle to determine whether or not "Missing" appears in the summary stats: TRUE (default) The Missing row is always displayed NA The Missing row is only displayed if any missing values are present FALSE The Missing row is never included in the table

Value

A data.table of summary statistics. The format depends on the type of the target variable:

- If the target variable is categorical, i.e. type character, factor or logical then the output is that of calc_counts().
- If the target variable is numeric, then the output is that of calc_desc().

See Also

tern_summary_table()

Examples

```
dmg_vars <- c("AGE", "RACE", "ETHNIC")
dmg_var_lbls <- c("Age (yr)", "Race", "Ethnicity")

# Demographics table (DMT01)
summary_table(
  adsl,
  target = dmg_vars,
  treat = 'ARM',
  target_name = dmg_var_lbls
)

# Demographics table (DMT01) with continuous variable (e.g., BMRKR1)
summary_table(
  adsl,
  target = c(dmg_vars, "BMRKR1"),
  treat = 'ARM',
  target_name = c(dmg_var_lbls, "Biomarker 1")
)
```

summary_table_by *Create a summary table using multiple rows for grouping on one target column*

Description

Create a summary table using multiple rows for grouping on one target column

Usage

```
summary_table_by(
  dt,
  target,
  treat,
  rows_by,
  indent = nbsp(n = 4L),
  .total_dt = dt,
  pct_dec = 1,
  treat_order = NULL,
  skip_absent = TRUE,
  sep = "."
)
```

Arguments

dt	A data.frame containing, at least, the variables indicated in target and treat.
target	Target variable passed as a string for which summary statistics are to be calculated.
treat	A string indicating the grouping variable, e.g. the variable specifying the treatment population.
rows_by	string, grouping variable to split events by.
indent	A string to be used as indentation of summary statistics labels. Defaults to four HTML non-breaking spaces ( ).
.total_dt	Separate table from dt from which to derive total counts per group.
pct_dec	Decimal places for reported figures.
treat_order	Customise the column order of the output table.
skip_absent	Whether to ignore variables passed in treat_order that are absent from dt. Default is TRUE; FALSE will throw an error in case there are missing variables.
sep	character string to separate the terms

Value

The same output as summary_table() except that folded by variables indicated in rows_by.

Examples

```
summary_table_by(
  adlb,
  target = "AVAL",
  treat = "ARM",
  rows_by = c("PARAM","AVISIT")
)
```

`summary_table_by_targets` *Create a summary table using multiple rows for grouping on two target column ideal for creating change from baseline tables*

Description

Create a summary table using multiple rows for grouping on two target column ideal for creating change from baseline tables

Usage

```
summary_table_by_targets(
  dt,
  target,
  treat,
  rows_by,
  indent = nbsp(n = 4L),
  .total_dt = NULL,
  pct_dec = 1,
  treat_order = NULL,
  skip_absent = TRUE,
  sep = "."
)
```

Arguments

`dt`	table to perform function on
`target`	vector of column names desired to obtain information on
`treat`	string of treatment variable used for splitting / grouping data
`rows_by`	string, grouping variable to split events by.
`indent`	indent to be used for display and formatting purposes
`.total_dt`	optional table for total counts to be derived
`pct_dec`	decimal places for percentages
`treat_order`	customise the column order of output table
`skip_absent`	Logical, default `TRUE`. Passed to `data.table::setcolorder`, if `treat_order` includes columns not present in `dt`:

- TRUE will silently ignore them
- FALSE will throw an error

sep character string to separate the terms

Value

data.table

Examples

```
adlb <- random.cdisc.data::cadlb |>
  dplyr::filter(AVISIT != "SCREENING")

labs <- summary_table_by_targets(
  dt = adlb,
  target = c("AVAL", "CHG"),
  treat = "ARM",
  rows_by = c("PARAM", "AVISIT"),
  indent = "  "
)
```

---

tern_AET01_table *Generate Core Safety Tables (CSR Section 14.3.1) using tern/rtables*

Description

tern_AET01_table() produces a consolidated safety summary table using rtables and tern. It mirrors the output and interface of AET01_table(), generating standard adverse event summaries (e.g. death, withdrawal, AESIs) for Clinical Study Reports (CSR) Section 14.3.1.

Usage

```
tern_AET01_table(
  adsl,
  adae,
  patient_var,
  treat_var,
  aesi_vars,
  aesi_heading = "Total number of patients with at least one",
  indent = "  "
)
```

Arguments

adsl	A subject-level dataset (typically ADaM ADSL).
adae	A dataset of adverse events, preprocessed with AESI flags.
patient_var	A string indicating the subject identifier variable (e.g., "USUBJID").

treat_var	A string indicating the treatment arm variable (e.g., "ARM").
aesi_vars	A character vector of binary AESI flags in adae.
aesi_heading	Ignored (included for interface compatibility).
indent	Ignored (included for interface compatibility).

Details

The function returns a single formatted rtables table summarising core safety endpoints by treatment arm.

Value

A TableTree object from the rtables package.

Examples

```
tern_AET01_table(
  adsl = adsl,
  adae = aesi,
  patient_var = "USUBJID",
  treat_var = "ARM",
  aesi_vars = c("FATAL", "SER", "SERWD", "SERDSM", "RELSER",
                "WD", "DSM", "REL", "RELWD", "RELDSM", "SEV")
)
```

---

tern_AET02_table *Generate AET02-style AE summary using* tern *and* rtables

---

Description

This function builds a System Organ Class (SOC) and Preferred Term (PT) adverse event summary table, following the AET02 CSR format, using the tern and rtables packages.

Usage

```
tern_AET02_table(
  adsl,
  adae,
  patient,
  treat,
  target = "AEDECOD",
  rows_by = "AEBODSYS",
  indent = "  "
)
```

Arguments

adsl	Subject-level dataset.
adae	Adverse event dataset.
patient	Unique subject identifier variable.
treat	Treatment arm variable.
target	Preferred term variable (default: "AEDECOD").
rows_by	Higher-level nesting term (default: "AEBODSYS").
indent	Ignored (included for compatibility).

Value

A TableTree object with AE summary by SOC/PT.

See Also

AET02_table()

---

tern_summary_table *Create a clinical reporting table with tern/rtables*

Description

tern_summary_table() is a convenience wrapper around {rtables} and {tern} commands to generate a clinical reporting summary statistics tables whilst using a similar interface as summary_table(). This can be helpful for side by side comparisons of the two functions.

Usage

```
tern_summary_table(dt, target, treat, target_name = target)
```

Arguments

dt	A data.frame containing, at least, the variables indicated in target and treat.
target	Target variable passed as a string for which summary statistics are to be calculated.
treat	A string indicating the grouping variable, e.g. the variable specifying the treatment population.
target_name	Heading for the target variable as a string. Defaults to target.

Value

A data.table of summary statistics. The format depends on the type of the target variable:

- If the target variable is categorical, i.e. type character, factor or logical then the output is that of calc_counts().
- If the target variable is numeric, then the output is that of calc_desc().

See Also

summary_table()

Examples

```
dmg_vars <- c("AGE", "RACE", "ETHNIC")
dmg_var_lbls <- c("Age (yr)", "Race", "Ethnicity")

# Demographics table (DMT01)
tern_summary_table(
  adsl,
  target = dmg_vars,
  treat = 'ARM',
  target_name = dmg_var_lbls
)

# Demographics table (DMT01) with continuous variable (e.g., BMRKR1)
tern_summary_table(
  adsl,
  target = c(dmg_vars, "BMRKR1"),
  treat = 'ARM',
  target_name = c(dmg_var_lbls, "Biomarker 1")
)
```

---

total_events *Count total events*

Description

total_events() counts the number of observations in dt in each group defined by treat levels. Counts are returned in wide format, i.e. one column per level in treat.

Usage

```
total_events(dt, treat, label)
```

Arguments

dt	A data.frame containing, at least, the variable indicated in treat.
treat	A string indicating the grouping variable, e.g. the variable specifying the treatment population.
label	A string to be used as label in the output reporting table. This should be a text descriptive of the event being counted.

Value

A list wrapping a one-row data.table of 1 + n variables, where n is the number of levels in treat. First variable is stats, character type, whose value is the argument passed in as label. Following variables are of integer type and provide the counts.

Examples

```
# In the absence of pre-filtering, `total_events()`, actually, just counts
# observations in `dt`.
total_events(dt = adsl, treat = "ARM", label = "Subjects")[[1]]

# If `dt` is pre-filtered, e.g. with a condition matching an event, then
# `total_events()` can be used to (effectively) count events.
total_events(dt = adsl[adsl$DTHFL == "Y"], treat = "ARM", label = "Deaths")[[1]]

# Another example using the complement predicate condition.
total_events(dt = adsl[adsl$DTHFL == "N"], treat = "ARM", label = "Lives")[[1]]
```

---

with_label *Add a label attribute to an object*

Description

Add a label attribute to an object

Usage

```
with_label(x, label)
```

Arguments

x	An R object.
label	A label provided as a single string.

Value

x labeled by label.

See Also

label()

Examples

```
label(1)
label(with_label(1, "my label"))
```

Index