Package ‘StatsTFLValR’

December 29, 2025

Type Package

Title Utilities for Validation of Clinical Trial Datasets and Outputs

Version 1.0.0

Description Provides utility functions for validation and quality control of
clinical trial datasets and outputs. The package supports dataset loading,
metadata inspection, frequency and summary calculations, table-ready
aggregations, and compare-style dataset review similar to SAS PROC COMPARE.
Functions are designed to support reproducible execution, transparent review,
and independent verification of statistical programming results.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.2.0)

Imports dplyr,
tidyr,
tibble,
rlang,
haven,
readxl,
tidyselect,
purrr,
arsenal,
data.table

Suggests knitr,
rmarkdown,
testthat (>= 3.0.0),
gt,
gtsummary,
withr

Config/testthat/edition 3

LazyData true

URL `https://github.com/kalsem/StatsTFLValR`

BugReports `https://github.com/kalsem/StatsTFLValR/issues`

Contents

`ATCbyDrug` *Fully Nested ATC2 → ATC4 → Drug (CMDECOD) Table by Treatment (wide)*

Description

Builds a three-level nested summary table of concomitant medications (or similar data), grouped as **ATC2 → ATC4 → Drug (CMDECOD)**, with counts and percentages by treatment arm. Outputs a **wide** data frame where each treatment column contains `n (pct)`.

Two indent modes are supported for the display label column `stat`:

- **RTF mode (default):** If `atc4_spaces` and `cmdecod_spaces` are both `NULL`, and `rtf_safe = TRUE`, `stat` will include the provided RTF indent strings (`atc4_rtf`, `cmdecod_rtf`) before the label text.
- **SAS blanks mode:** If `atc4_spaces` or `cmdecod_spaces` is provided (non-`NULL`), `stat` will use **only blank spaces** (no RTF codes) as visual indents (SAS-style), regardless of `rtf_safe`.

Sorting can be controlled by `sort_by`:

- `"count"` (default): within each level, sort descending by counts for the column `n__<trtan_coln>` (e.g., `n__21`), then alphabetically.
- `"alpha"`: alphabetical ascending order at each level.

Rows where **all three levels** are `"UNCODED"` (case-insensitive) are pushed to the very end of the table (after all other rows), preserving the nested order.

Usage

```
ATCbyDrug(
  indata,
  dmdata,
  group_vars,
  trtan_coln,
  rtf_safe = TRUE,
  sort_by = c("count", "alpha"),
  atc4_spaces = NULL,
  cmdecod_spaces = NULL,
  atc4_rtf = "(*ESC*)R/RTF\"\\li180 \"",
  cmdecod_rtf = "(*ESC*)R/RTF\"\\li360 \""
)
```

Arguments

`indata`	A data frame containing medication/event records. Must include: `USUBJID` and the variables named in `group_vars`.
`dmdata`	A data frame with one row per subject (for denominators). Must include `USUBJID` and the main treatment grouping variable (first element of `group_vars`).
`group_vars`	Character vector of length 4 specifying, in order: `c(main_group, atc2, atc4, meddecod)`. • `main_group` = treatment/grouping variable used for columns (e.g., `"TRTAN"`). • `atc2`, `atc4`, `meddecod` = the three nested display levels.
`trtan_coln`	Character scalar giving the **column-level** of interest used for count-based sorting, i.e., the suffix in `n__<trtan_coln>`. Example: `"21"` makes the function look for `n__21` to drive `"count"` sorting.
`rtf_safe`	Logical; if `TRUE`, RTF strings will be used in `stat` **when** both `atc4_spaces` and `cmdecod_spaces` are `NULL`. If either spaces argument is provided, `stat` will **not** include RTF strings.
`sort_by`	One of `c("count","alpha")`. See Details.
`atc4_spaces, cmdecod_spaces`	`NULL` or non-negative integer specifying the number of **blank spaces** to prepend for `ATC4` and `Drug (CMDECOD)` labels in `stat`. If either is non-`NULL`, the function uses **SAS blanks mode** (no RTF codes).
`atc4_rtf, cmdecod_rtf`	Character RTF indent strings used **only** when *both* `atc4_spaces` and `cmdecod_spaces` are `NULL` and `rtf_safe = TRUE`. Defaults: `(*ESC*)R/RTF"\\li180 "` for `ATC4`, `(*ESC*)R/RTF"\\li360 "` for `Drug (CMDECOD)`.

Details

Denominator (`N`) is computed from `dmdata` as distinct `USUBJID` per `main_group`. For each level (ATC2, ATC4 within ATC2, Drug/CMDECOD within ATC4), the function computes distinct-subject counts by `main_group`, the percentage w.r.t. `N`, and forms `"n (pct)"`. The wide result has:

- `stat` = display label with indent (RTF or blanks, depending on mode).
- `trt<value>` columns (e.g., `trt21`, `trt22`, ...): `"n (pct)"` per treatment value.
- `n__<value>` columns mirroring raw counts (useful for custom sorting or QC).
- Ordering columns: `sec_ord`, `psec_ord`, `sort_ord` (help keep nested order).

Indent modes:

- *RTF mode*: Use when you want RTF control words in the output for direct RTF rendering. Do **not** set `atc4_spaces`/`cmdecod_spaces`; keep `rtf_safe = TRUE`.
- *SAS blanks mode*: Provide `atc4_spaces` and/or `cmdecod_spaces` to indent using blanks only (friendly for plain-text outputs or RTF pipelines that inject formatting later).

UNCODED handling: Rows are considered UNCODED **only if** all three of `ATC2`, `ATC4`, and `Drug (CMDECOD)` equal `"UNCODED"` (case-insensitive, leading/trailing space ignored). Such rows are assigned to the end of the table after sorting.

Value

A tibble with nested rows containing:

- `stat` (indented label),
- treatment columns `trt*` (string `"n (pct)"`),
- raw-count columns `n__*`,
- helper ordering columns (`sec_ord`, `psec_ord`, `sort_ord`).

`freq_by` *Frequency Table by Group (wide): n (%) with flexible ordering and formats*

Description

`freq_by()` produces a one-level frequency table by treatment (wide layout) where each row is a category of `last_group` (e.g., a bucketed lab value), and each treatment column shows **n (%)** using distinct subject counts.

New: If `fmt` is **not provided** (`NULL`), labels are derived from the **unique values present in** `data[[last_group]]` (post `na_to_code` mapping, if used).

It supports:

- **SAS-style rounding** (`use_sas_round = TRUE`) for the percent.
- Format mapping via either a **named vector** or a **tibble/data.frame** with columns `value` (codes) and `raw` (labels).
- **Ordering** by the **numeric value** of `last_group` found in the data, or optionally the **union** of format + data codes (`include_all_fmt_levels`).
- Counting **NA** under a chosen code/label using `na_to_code` (e.g., code `"4" = "MISSING"`).
- Auto-detecting the subject ID column when `id_var` is not provided.

Usage

```
freq_by(
  data,
  denom_data = NULL,
  main_group,
  last_group,
  label,
  sec_ord,
  fmt = NULL,
  use_sas_round = FALSE,
  indent = 2,
  id_var = "USUBJID",
  include_all_fmt_levels = TRUE,
  na_to_code = NULL
)
```

Arguments

`data`	A data frame containing at least `main_group`, `last_group`, and an ID column.
`denom_data`	Optional data frame used to derive denominators (N per treatment). Defaults to `data`.
`main_group`	Character scalar. The treatment or grouping variable name (columns in output), e.g., `"TRTAN"`.
`last_group`	Character scalar. The categorical **code** variable to tabulate (rows). Numeric or character are both accepted; converted to character for display/ordering.
`label`	Character scalar. A header row displayed on top (unindented).
`sec_ord`	Integer scalar carried through for downstream table sorting.
`fmt`	Optional. Either: • a **named character vector** like `c("1"="<1","2"="1-<4",...)` (names = codes, values = labels), or • a **data.frame/tibble** with columns `value` (codes) and `raw` (labels), or • a **string** naming an object (in parent frame) that resolves to either of the above. If `NULL` (default), labels are derived from unique values of `data[[last_group]]`.
`use_sas_round`	Logical; if `TRUE`, percent is rounded with SAS-compatible "round halves away from zero" via `sas_round()`. Default `FALSE`.
`indent`	Integer number of **leading spaces** applied to all category rows (the first `label` row is not indented). Default `2`.
`id_var`	Character; the subject identifier column. If not found in `data`, the function tries common alternatives (e.g., `USUBJID`, `SUBJID`, etc.).
`include_all_fmt_levels`	Logical; if `TRUE` (default), the row order is built from the **union of format codes and data codes** (numeric sort). When `fmt = NULL`, this effectively reduces to observed data codes only.
`na_to_code`	Optional character scalar (e.g., `"4"`). If supplied, NA values in `last_group` are **counted under that code** before tabulation.

Details

- Counting uses `n_distinct(id_var)` within each `(main_group, last_group)` cell.
- Percent is `100 * n / N` where `N` = distinct subjects in `denom_data` by `main_group`.
- When `fmt = NULL`, both **codes** and **labels** are taken from the observed values of `last_group` (after applying `na_to_code` mapping), ordered numerically where possible.
- Output treatment columns are normalized to `trtXX` if original names start with digits.
- Missing treatment arms are added as `"0"`.

Value

A tibble with:

- `stat` (character), `sort_ord` (integer), `sec_ord` (integer),
- One column per treatment arm (e.g., `trt1`, `trt2`, ...), with `"n (pct)"` or `"0"`.

Examples

```
# --- Minimal toy example: Age group, Sex, Ethnic by treatment ---
set.seed(1)

toy_adsl <- tibble::tibble(
  USUBJID = sprintf("ID%03d", 1:60),
  TRTAN   = sample(c(1, 2), size = 60, replace = TRUE),
  AGE     = sample(18:85, size = 60, replace = TRUE),
  SEX     = sample(c("Male", "Female"), size = 60, replace = TRUE),
  ETHNIC  = sample(
    c("Hispanic or Latino",
      "Not Hispanic or Latino",
      "Unknown",
      NA_character_),
    size = 60, replace = TRUE
  )
) |>
  dplyr::mutate(
    AGEGR1 = dplyr::case_when(
      AGE < 65            ~ "<65 years",
      AGE >= 65 & AGE < 75 ~ "65-<75 years",
      AGE >= 75           ~ ">=75 years"
    )
  )

# Denominator dataset typically comes from ADSL
toy_dm <- toy_adsl |>
  dplyr::select(USUBJID, TRTAN)

# --- 1) Age group by treatment (auto fmt from data) ---
freq_by(
  data       = toy_adsl,
  denom_data = toy_dm,
  main_group = "TRTAN",
  last_group = "AGEGR1",
  label      = "Age group, n (%)",
  sec_ord    = 1,
  fmt        = NULL,        # derive labels from AGEGR1 values
  na_to_code = NULL         # AGEGR1 NA (if any) will be dropped
)

# --- 2) Sex by treatment, mapping NA to a code (fmt still auto) ---
freq_by(
  data       = toy_adsl,
  denom_data = toy_dm,
  main_group = "TRTAN",
  last_group = "SEX",
  label      = "Sex, n (%)",
  sec_ord    = 2,
  fmt        = NULL,        # auto labels: "Female", "Male", "99"
  na_to_code = "99"         # missing SEX counted under "99"
)

# --- 3) Ethnic category with explicit fmt (named vector) ---
fmt_ethnic <- c(
  "Hispanic or Latino"         = "Hispanic or Latino",
  "Not Hispanic or Latino"     = "Not Hispanic or Latino",
  "Unknown"                    = "Unknown",
  "99"                         = "Missing"
)

freq_by(
  data       = toy_adsl,
  denom_data = toy_dm,
  main_group = "TRTAN",
  last_group = "ETHNIC",
  label      = "Ethnic group, n (%)",
  sec_ord    = 3,
  fmt        = fmt_ethnic,
  include_all_fmt_levels = TRUE,  # show all levels from fmt_ethnic
  na_to_code = "99"               # NA ETHNIC mapped to code "99"
)
```

`freq_by_line` *One-Line Frequency Summary by Treatment Group*

Description

Generates a single-row frequency summary table across treatment groups, reporting counts and percentages of subjects meeting a filter condition.

Usage

```
freq_by_line(data, id_var, trt_var, filter_expr, label, denom_data = NULL)
```

Arguments

`data`	A data.frame containing subject-level data.
`id_var`	Unquoted subject ID variable (e.g., `USUBJID`).
`trt_var`	Unquoted treatment variable (e.g., `TRT01P`).
`filter_expr`	A logical filter expression (unquoted), e.g., `SAFFL == "Y" & AGE >= 65`.
`label`	Character string for the row label in the output (e.g., `"SAF population"`).
`denom_data`	Optional. A data.frame used to calculate denominators per treatment group. Defaults to `data`.

Details

This function calculates the number and percentage of unique subjects per treatment group (`trt_var`) satisfying a given filter condition (`filter_expr`). The result is formatted as `"n (pct)"` and returned in a single-row tibble, labeled by the provided `label`. An optional denominator dataset (`denom_data`) can be specified to override the default denominator population (used to calculate percentages).

Useful for producing compact summary rows (e.g., "SAF Population", "Subjects >= 65") in clinical tables.

Value

A one-row tibble containing "n (pct)" summaries per treatment group.

Examples

```
set.seed(123)
adsl <- data.frame(
  USUBJID = paste0("SUBJ", 1:100),
  TRT01P = sample(c("0", "54", "100"), 100, replace = TRUE),
  SAFFL = sample(c("Y", "N"), 100, replace = TRUE),
  AGE = sample(18:80, 100, replace = TRUE)
)

# Standard use with internal denominator
freq_by_line(adsl, USUBJID, TRT01P, SAFFL == "Y", label = "SAF population")

# Use SAF subset as denominator
saf <- adsl[adsl$SAFFL == "Y", ]
freq_by_line(
  adsl, USUBJID, TRT01P,
  AGE >= 65,
  label = "Age >=65 in SAF",
  denom_data = saf
)
```

---

generate_compare_report

Compare DEV vs VAL datasets (PROC COMPARE-style) with robust file detection

---

Description

`generate_compare_report()` compares a **developer (DEV)** dataset and a **validation (VAL)** dataset for a given `domain` and produces outputs similar to SAS `PROC COMPARE`.

This function is intended for ADaM/SDTM/TFL validation workflows and supports:

- **Directory-driven inputs**: DEV and VAL locations are provided via `dev_dir` and `val_dir`.
- **Case-insensitive domain matching**: `domain = "ADAE"` will match files like `adae.*`.
- **VAL prefix flexibility**: resolves `prefix_val` variants such as `v_`, `v-`, and `v` (no separator).
- **Automatic extension detection** for DEV and VAL files: `.sas7bdat`, `.xpt`, `.csv`, `.rds`.
- **Optional filtering** using `filter_expr` prior to comparison.
- **Optional PROC COMPARE-style CSV** output with `BASE`, `COMPARE`, and `DIF` triplets.
- **Optional LST-like report** using `arsenal::comparedf()` for summarized differences.

Usage

```
generate_compare_report(
  domain,
  dev_dir,
  val_dir,
  by_vars = c("STUDYID", "USUBJID"),
  vars_to_check = NULL,
  report_dir = getwd(),
  prefix_val = "v_",
  max_print = 50,
  write_csv = FALSE,
  run_comparedf = TRUE,
  filter_expr = NULL,
  study_id = NULL,
  author = NULL
)
```

Arguments

`domain`	Character scalar domain name (e.g., `"adsl"`, `"adae"`, `"rt-ae-sum"`). Matching is case-insensitive.
`dev_dir`	DEV dataset directory path.
`val_dir`	VAL dataset directory path.
`by_vars`	Character vector of key variables used to match records (e.g., `c("STUDYID","USUBJID")` or `c("STUDYID","USUBJID","AESEQ")`).
`vars_to_check`	Optional character vector of variables to compare. If `NULL`, compares all common variables (excluding key handling remains as per implementation).
`report_dir`	Output directory for report files. Created if missing.
`prefix_val`	Character prefix for validation datasets (default `"v_"`). The resolver also supports variants like `v-` and `v` (no separator).
`max_print`	Maximum number of lines printed in the `.lst` report for summaries/diffs.
`write_csv`	Logical; if `TRUE`, writes PROC COMPARE-style CSV to `report_dir` as `compare_<domain>.csv`.
`run_comparedf`	Logical; if `TRUE`, uses `arsenal::comparedf()` to generate a `.lst` report.
`filter_expr`	Optional filter expression **string** evaluated within each dataset (e.g., `"SAFFL == 'Y' & TRTEMFL == 'Y'"`).
`study_id`	Optional study identifier included in the `.lst` header.
`author`	Optional author name included in the `.lst` header.

Details

File resolution rules:

The function looks for exactly one matching domain file per directory:

- DEV: `<domain>.<ext>`
- VAL: `<prefix><domain>.<ext>` where `<prefix>` is `prefix_val` plus common variants supporting underscore/hyphen/no-separator forms (e.g., `v_`, `v-`, `v`).

Supported extensions (priority order) are: `sas7bdat`, `xpt`, `csv`, `rds`.

If multiple matches exist for the same domain in a directory (e.g., `adae.csv` and `adae.xpt`), the function stops with an **ambiguous match** error to prevent accidental comparisons.

PROC COMPARE-style CSV behavior:

When `write_csv` = `TRUE`, the output includes:

- `_TYPE_` with values `BASE`, `COMPARE`, `DIF`
- `_OBS_` sequence within each BY key
- For numeric variables, `DIF` = `DEV - VAL`
- For Date variables, `DIF` is **integer day difference** (`as.integer(DEV - VAL)`)
- For POSIXct variables, `DIF` is **seconds difference** (`as.numeric(DEV - VAL)`)
- For other types, `DIF` is a character mask (`X` indicates difference)

Value

Invisibly returns a list with:

- `only_in_dev`: rows present only in DEV (set-difference result)
- `only_in_val`: rows present only in VAL (set-difference result)
- `comparedf`: `arsenal::comparedf` object (or `NULL` if `run_comparedf` = `FALSE`)

See Also

`comparedf`, `fsetdiff`, `fintersect`

Examples

```
## Not run:
# ------------------------------------------------------------
# Example 1: Basic comparison (auto-detect CSV files)
# DEV: adae.csv
# VAL: v-adae.csv
generate_compare_report(
  domain    = "adae",
  dev_dir   = "C:/R-Packages/adam/dev",
  val_dir   = "C:/R-Packages/adam/val",
  by_vars   = c("STUDYID","USUBJID","AESEQ"),
  write_csv = TRUE
)

# ------------------------------------------------------------
# Example 2: Case-insensitive domain name ("ADAE" matches adae.*)
generate_compare_report(
  domain  = "ADAE",
  dev_dir = "C:/R-Packages/adam/dev",
  val_dir = "C:/R-Packages/adam/val",
  by_vars = c("STUDYID","USUBJID","AESEQ")
)

# ------------------------------------------------------------
# Example 3: Mixed formats (DEV .sas7bdat, VAL .xpt)
# DEV: adsl.sas7bdat
# VAL: v_adsl.xpt (or v-adsl.xpt)
generate_compare_report(
  domain  = "adsl",
  dev_dir = "D:/study/dev/adam",
  val_dir = "D:/study/val/sasout",
  by_vars = c("STUDYID","USUBJID")
```

```
)

# ------------------------------------------------------------
# Example 4: Hyphenated domain (e.g., TFL artifacts)
# DEV: rt-ae-sum.csv
# VAL: v-rt-ae-sum.csv
generate_compare_report(
  domain    = "rt-ae-sum",
  dev_dir   = "C:/R-Packages/adam/dev",
  val_dir   = "C:/R-Packages/adam/val",
  by_vars   = c("STUDYID","USUBJID","AESEQ"),
  write_csv = TRUE
)

# ------------------------------------------------------------
# Example 5: Filtered comparison (subset both datasets)
generate_compare_report(
  domain      = "adae",
  dev_dir     = "C:/adam/dev",
  val_dir     = "C:/adam/val",
  by_vars     = c("STUDYID","USUBJID","AESEQ"),
  filter_expr = "SAFFL == 'Y' & TRTEMFL == 'Y'",
  write_csv   = TRUE
)

# ------------------------------------------------------------
# Example 6: Restrict comparison to specific variables only
generate_compare_report(
  domain        = "adae",
  dev_dir       = "C:/adam/dev",
  val_dir       = "C:/adam/val",
  by_vars       = c("STUDYID","USUBJID","AESEQ"),
  vars_to_check = c("AETOXGR", "AEREL", "ASTDT", "AENDT"),
  write_csv     = TRUE
)

# ------------------------------------------------------------
# Example 7: CSV only (skip comparedf / LST report)
generate_compare_report(
  domain        = "adlb",
  dev_dir       = "C:/adam/dev",
  val_dir       = "C:/adam/val",
  by_vars       = c("STUDYID","USUBJID","PARAMCD","AVISITN"),
  write_csv     = TRUE,
  run_comparedf = FALSE
)

# ------------------------------------------------------------
# Example 8: Compare only key alignment (minimal variable selection)
# Useful when you only want to detect missing/extra keys quickly.
generate_compare_report(
  domain        = "adsl",
  dev_dir       = "C:/adam/dev",
  val_dir       = "C:/adam/val",
  by_vars       = c("STUDYID","USUBJID"),
  vars_to_check = c("USUBJID"),   # effectively only key coverage + minimal compare
  run_comparedf = FALSE,
  write_csv     = TRUE
)

# ---------------------------------------------------------------
# Example 9: Ambiguous file error (intentional safety stop)
# DEV folder contains both adae.csv and adae.xpt -> stop
generate_compare_report(
  domain  = "adae",
  dev_dir = "C:/adam/dev",
  val_dir = "C:/adam/val",
  by_vars = c("STUDYID","USUBJID","AESEQ")
)

## End(Not run)
```

`get_column_info` *Extract Column Metadata from a Data Frame*

Description

Inspects a data frame and returns a summary of metadata for each column, including column name, label, format, class/type, missingness, uniqueness, and (optionally) SAS-style display for Date variables (e.g., DATE9 -> 09JUL2012).

Usage

```
get_column_info(
  df,
  include_attributes = TRUE,
  exclude_attributes = c("class", "row.names"),
  label_attr = c("label", "var.label", "labelled", "Label"),
  format_attr = c("format", "format.sas", "Format", "displayWidth"),
  compute_ranges = TRUE,
  sas_date_display = TRUE
)
```

Arguments

`df`	A data.frame or tibble. The input dataset whose column metadata should be extracted.
`include_attributes`	Logical. If TRUE, includes a list-column of full attributes (after exclusions).
`exclude_attributes`	Character vector of attribute names to drop from the attributes list.
`label_attr`	Character vector of attribute names to check (in order) for a label.
`format_attr`	Character vector of attribute names to check (in order) for a format.
`compute_ranges`	Logical. If TRUE, computes min/max for numeric and date/datetime types.
`sas_date_display`	Logical. If TRUE, adds SAS-style display columns for Date/POSIXct.

Value

A tibble with one row per column and metadata fields.

- **column**: Column name
- **label**: Label attribute (if present)
- **format**: Format attribute (if present; e.g., DATE9.)
- **class**: Class(es)
- **typeof**: Underlying storage type
- **n**: Total length
- **n_missing**: Number of NAs
- **n_unique**: Number of unique values
- **min_raw/max_raw**: Min/max as raw values (Date/numeric)
- **min_disp/max_disp**: Min/max as display strings (SAS-like for dates when enabled)
- **sample_disp**: First non-missing value as display string (SAS-like for dates when enabled)
- **attribute_names**: Comma-separated attribute names (after exclusions)
- **attributes**: List column of attributes (optional)

Examples

```
## Not run:
  df <- haven::read_sas("adae.sas7bdat")
  out <- get_column_info(df)
  print(out)

## End(Not run)
```

---

get_data — *Load Data Files of Various Formats into Global Environment*

Description

Loads one or more data files from a given directory into the global environment. Supports multiple file types commonly used in clinical trials: `.sas7bdat`, `.xpt`, `.csv`, `.xls`, and `.xlsx`.

Usage

```
get_data(dir, file_names = NULL)
```

Arguments

`dir`	Character. Path to the directory containing data files.
`file_names`	Character vector. Optional base names (with or without extensions) to load; if NULL, loads all supported files from the directory.

Details

Automatically detects file extensions and assigns each dataset using its base file name (e.g., "adsl.xpt" becomes adsl).

If multiple files with the same base name but different extensions exist (e.g., adsl.csv and adsl.sas7bdat), the function will stop and report the duplicates to avoid ambiguity. This validation applies both when reading all files and when a list of files is specified.

Value

Invisibly returns the single data frame if one file is loaded, or NULL if multiple. In all cases, assigns each loaded data frame to the global environment using its base name.

Examples

```
## Not run:

  # ------------------------------------------------------------------
  # 1) Initialize directory structure (Windows example)
  # ------------------------------------------------------------------

  # Use proper slashes in Windows paths:
  adam_ds <- "C:/R-Packages/adam/"
  # OR: adam_ds <- "C:\R-Packages\adam\"


  # ------------------------------------------------------------------
  # 2) Load a single dataset from ADaM
  # ------------------------------------------------------------------

  adae <- get_data(adam_ds, "adae")


  # ------------------------------------------------------------------
  # 3) Load multiple datasets by base name
  # ------------------------------------------------------------------

  get_data(adam_ds, c("adsl", "adae"))


  # ------------------------------------------------------------------
  # 4) Load all supported datasets from a directory
  # ------------------------------------------------------------------

  get_data(adam_ds)


  # ------------------------------------------------------------------
  # 5) Example: duplicate base names trigger an error
  # ------------------------------------------------------------------

  # If your folder contains files like:
  #   adlb.sas7bdat
  #   ADLB.xpt
  #   admh.sas7bdat
  #   ADMH.xpt
  #
  # Then the following will throw:
  #
  #   Error in get_data(adam_ds):
  #     Multiple files found with the same base name (different extensions):
  #       - adlb.sas7bdat
  #       - ADLB.xpt
  #       - admh.sas7bdat
  #       - ADMH.xpt
  #     Please resolve file name conflicts or use only one extension per base name.
  #
  # get_data(adam_ds)


## End(Not run)
```

---

mean_by *Summary Table: Mean and Related Statistics by Group*

Description

This function calculates common summary statistics (N, Mean, SD, Median, Q1, Q3, Min, Max) for a numeric variable, grouped by a treatment or category variable. It supports optional **SAS-style rounding** (round half away from zero) and formats the results for table-ready display. Missing treatment groups are automatically added with zero values.

Usage

```
mean_by(
  data,
  group_var,
  uniq_var,
  label,
  sec_ord,
  precision_override = NULL,
  indent = 3,
  use_sas_round = FALSE,
  id_var = "USUBJID"
)
```

Arguments

`data`	A data frame or tibble containing the input data.
`group_var`	The grouping variable (e.g., treatment arm). Can be unquoted (tidy evaluation) or a string.
`uniq_var`	The numeric variable to summarise. Can be unquoted (tidy evaluation) or a string.
`label`	Character string: table section label for the output (e.g., `"BMI (WEIGHT [KG]/ HEIGHT [M2])"`).
`sec_ord`	Integer: section order value (for downstream table ordering).
`precision_override`	Optional integer to manually set decimal precision; if `NULL`, the function infers precision from the data.

`indent`	Integer: number of leading spaces in statistic labels (default = 3).
`use_sas_round`	Logical: if `TRUE`, applies SAS-compatible rounding (round half away from zero). Default is `FALSE`.
`id_var`	Character: name of subject ID variable (default = `"USUBJID"`). If not found, function attempts to auto-detect common ID variable names.

Details

The function:

1. Auto-detects precision if `precision_override` is `NULL`.
2. Calculates N, mean, SD, quartiles, min, max.
3. Applies SAS-style rounding if `use_sas_round = TRUE`.
4. Converts statistics into a display format suitable for RTF or text output.
5. Ensures all treatment columns appear in output, filling missing ones with `"0"`.

SAS-style rounding logic: Values exactly halfway between two increments are rounded away from zero (e.g., `1.25` → `1.3`, `-1.25` → `-1.3` with 1 decimal place).

Value

A tibble with the following columns:

- `stats` : internal statistic code (`n1`, `mn`, `sd`, etc.)
- `stat` : display label (`" N"`, `" MEAN"`, etc.)
- `sort_ord` : row ordering number
- `sec_ord` : section ordering number (from input)
- Treatment columns (`trt1`, `trt2`, ...): formatted values per treatment group

Examples

```
library(dplyr)

# Example 1: Basic usage with inferred precision
df <- tibble::tibble(
  USUBJID = rep(1:6, each = 1),
  TRTAN   = c(1, 1, 2, 2, 3, 3),
  BMIBL   = c(25.1, 26.3, 24.8, NA, 23.4, 27.6)
)
mean_by(
  data          = df,
  group_var     = TRTAN,
  uniq_var      = BMIBL,
  label         = "BMI (kg/m^2)",
  sec_ord       = 1
)

# Example 2: Forcing precision to 2 decimal places
mean_by(
  data          = df,
  group_var     = TRTAN,
  uniq_var      = BMIBL,
  label         = "BMI (kg/m^2)",
  sec_ord       = 1,
  precision_override = 2
)

# Example 3: Using SAS-style rounding
mean_by(
  data          = df,
  group_var     = TRTAN,
  uniq_var      = BMIBL,
  label         = "BMI (kg/m^2)",
  sec_ord       = 1,
  use_sas_round = TRUE
)

# Example 4: Missing treatment group automatically filled
df2 <- tibble::tibble(
  USUBJID = c(1, 2, 3, 4),
  TRTAN   = c(1, 1, 3, 3),
  BMIBL   = c(25.1, 26.3, 23.4, 27.6)
)
mean_by(
  data      = df2,
  group_var = TRTAN,
  uniq_var  = BMIBL,
  label     = "BMI (kg/m^2)",
  sec_ord   = 1
)
```

`sas_round` *SAS-Compatible Rounding*

Description

Performs rounding in the same manner as SAS, where values exactly halfway between two integers are always rounded away from zero. This differs from R's default rounding (IEC 60559), which rounds to the nearest even number ("bankers' rounding").

Usage

```
sas_round(x, digits = 0)
```

Arguments

`x`	A numeric vector to be rounded.
`digits`	Integer indicating the number of decimal places to round to. Default is 0.

Details

In SAS, values like 1.5 or -2.5 are rounded to 2 and -3 respectively. This function emulates that behavior by manually adjusting and checking the fractional component of the value before applying rounding.

Value

A numeric vector with values rounded using SAS-compatible logic.

Examples

```
# Default rounding (digits = 0)
sas_round(c(1.5, 2.5, 3.5, -1.5, -2.5, -3.5))
# [1]  2  3  4 -2 -3 -4

# Round to 1 decimal place
sas_round(c(1.25, 1.35, -1.25, -1.35), digits = 1)
# [1]  1.3  1.4 -1.3 -1.4

# Round to 2 decimal places
sas_round(c(1.235, 1.245, -1.235, -1.245), digits = 2)
# [1]  1.24  1.25 -1.24 -1.25

# Round to 3 decimal places
sas_round(c(1.2345, 1.2355), digits = 3)
# [1] 1.235 1.236

# Round to 4 decimal places
sas_round(c(1.23445, 1.23455), digits = 4)
# [1] 1.2345 1.2346

# Round to 5 decimal places
sas_round(c(1.234445, 1.234455), digits = 5)
# [1] 1.23445 1.23446
```

`SOCbyPT` *SOC → PT summary by treatment (wide), with optional BY-grouping, SOC totals, UNCODED positioning, BY-specific Big-N, and optional Big-N printing*

Description

Build a System Organ Class (SOC) → Preferred Term (PT) summary by treatment in a wide layout suitable for clinical TLFs. Optionally stratify the display by a BY variable from the AE dataset, order BY groups by a separate key, add TOTAL rows, control UNCODED placement, and optionally calculate percentages using BY-specific denominators.

Usage

```
SOCbyPT(
  indata,
  dmdata,
  pop_data = NULL,
  group_vars,
  trtan_coln,
  by_var = NULL,
  by_sort_var = NULL,
  by_sort_numeric = TRUE,
  id_var = "USUBJID",
  rtf_safe = TRUE,
  indent_str = "(*ESC*)R/RTF\"\\li360 \"",
  use_sas_round = FALSE,
  header_blank = FALSE,
  soc_totals = FALSE,
  total_label = "TOTAL SUBJECTS WITH AN EVENT",
  uncoded_position = c("count", "last"),
  bigN_by = NULL,
  print_bigN = FALSE
)
```

Arguments

`indata`	AE-like input with at least: subject id, SOC, PT, and the main treatment column. If BY is used, `by_var` (and `by_sort_var` if different) must exist in `indata`.
`dmdata`	Working denominator dataset (e.g., filtered ADSL) with at least: subject id and the main treatment column. If `bigN_by = "YES"` and BY is used, `dmdata` must also contain `by_var` to compute BY-specific denominators.
`pop_data`	Master population dataset (e.g., full ADSL) used to define the set/order of treatment arms. If `NULL`, defaults to `dmdata`.
`group_vars`	Character vector of length 3: `c(main_treatment, SOC, PT)`.
`trtan_coln`	Treatment level value (e.g., `"12"` or `12`) that drives sorting (descending count, then alpha).
`by_var`	Optional BY column name (quoted or unquoted) from `indata` used to split the table into groups.
`by_sort_var`	Optional column (quoted or unquoted) used to order BY groups. Defaults to `by_var`.
`by_sort_numeric`	If `TRUE`, BY groups ordered by `as.numeric(by_sort_var)`; else lexicographic.
`id_var`	Subject identifier column name. Default `"USUBJID"`.
`rtf_safe`	If `TRUE`, PT labels are prefixed by `indent_str`. Default `TRUE`.
`indent_str`	Prefix added to PT labels when `rtf_safe = TRUE`.
`use_sas_round`	If `TRUE`, use SAS-style rounding (ties away from zero). Default `FALSE`.
`header_blank`	If `TRUE`, blank treatment cells on SOC header rows (TOTAL rows remain populated). Default `FALSE`.
`soc_totals`	If `TRUE`, SOC header rows are retained/populated (default behavior). Included for API parity.
`total_label`	Label for TOTAL row(s). Default `"TOTAL SUBJECTS WITH AN EVENT"`.
`uncoded_position`	Where to place UNCODED: `"count"` (default behavior by counts) or `"last"` (push to bottom).
`bigN_by`	Flag controlling denominator behavior when BY is used: • `NULL` / `"NO"` (default): denominators are by treatment only (not stratified by BY) • `"YES"`: denominators are by BY × treatment (requires `by_var` in `dmdata`)
`print_bigN`	If `TRUE`, prints denominators (Big-N) used for percent calculations to console/log.

Value

A tibble with columns:

- `stat`
- `trt*` treatment columns
- `sort_ord`, `sec_ord`
- `by_var`, `by_sort_var` (when BY used)

`SOCbyPT_Grade` *SOC → PT summary by treatment with Grade split (wide)*

Description

Summarises AEs by **System Organ Class (SOC)** → **Preferred Term (PT)** per treatment arm and splits each arm into **Grade** buckets (1–5 + NOT REPORTED). The table includes a first **TOTAL SUBJECTS WITH AN EVENT** row, optional SOC subtotal rows, and RTF-safe indenting for PT lines. The SOC/PT block order can be driven by a reference arm (e.g., `TRTAN = 12`) and **a specific grade** via `sort_grade` (default 5).

Usage

```
SOCbyPT_Grade(
  indata,
  dmdata,
  pop_data = NULL,
  group_vars,
  trtan_coln,
  grade_num = "AETOXGRN",
  grade_char = NULL,
  by_var = NULL,
  by_sort_var = NULL,
  by_sort_numeric = TRUE,
  bigN_by = NULL,
  print_bigN = FALSE,
  id_var = "USUBJID",
  rtf_safe = TRUE,
  indent_str = "(*ESC*)R/RTF\"\\li360 \"",
  use_sas_round = FALSE,
  header_blank = TRUE,
  soc_totals = FALSE,
  total_label = "TOTAL SUBJECTS WITH AN EVENT",
  nr_char_values = c("NOT REPORTED", "NOT_REPORTED", "NOTREPORTED", "NOT REPRTED", "NR",
    "N", "NA"),
  sort_grade = 5,
  debug = FALSE,
  uncoded_position = c("count", "last")
)
```

Arguments

`indata`	`data.frame`. AE-like data containing `USUBJID`, treatment, SOC, PT, and Grade variables.
`dmdata`	`data.frame`. ADSL-like data containing denominators per arm (must include `USUBJID` and the same treatment column as in `indata`).
`pop_data`	`data.frame` or `NULL`. Optional master population for arm Ns (defaults to `dmdata`).
`group_vars`	Character vector of length 3: `c(main_trt, soc, pt)`. Example: `c("TRTAN","AEBODSYS","AEDECOD`
`trtan_coln`	Character or numeric. The **reference treatment code** used for ordering SOC/PT blocks (e.g., `"12"`).
`grade_num`	Character. Name of numeric grade column (default `"AETOXGRN"`). Values 1–5 are treated as valid grades; others are ignored in numeric logic.
`grade_char`	Character or `NULL`. Optional character grade column name (e.g., `"AETOCGR"`/`"AETOXGR"`). If `NULL`, the function auto-detects `"AETOCGR"` then `"AETOXGR"` if present.
`by_var`	Character or `NULL`. Optional BY variable (from AE dataset) to generate stratified outputs and sort independently per stratum.
`by_sort_var`	Character or `NULL`. Optional helper column to order BY strata; defaults to `by_var` when `NULL`.
`by_sort_numeric`	Logical. If `TRUE` (default), order BY strata by `as.numeric(by_sort_var)`, else use character order.
`bigN_by`	Flag controlling denominator behavior when BY is used: • `NULL` / `"NO"` (default): denominators are by treatment only (not stratified by BY) • `"YES"`: denominators are by BY × treatment (requires `by_var` in `dmdata` or `pop_data`)
`print_bigN`	If `TRUE`, prints denominators (Big-N) used for percent calculations to console/log.
`id_var`	Character. Subject ID column (default `"USUBJID"`).
`rtf_safe`	Logical. If `TRUE` (default), prefix PT rows with `indent_str`.
`indent_str`	Character. The RTF literal for indentation of PT lines (default `(*ESC*)R/RTF\"\\li360 \"`).
`use_sas_round`	Logical. If `TRUE`, use SAS-style rounding for percentages; else base R `round()`.
`header_blank`	Logical. If `TRUE` (default) and `soc_totals = FALSE`, grade columns on SOC header rows are blanked.
`soc_totals`	Logical. If `TRUE`, include SOC subtotal rows using the same grade logic as PT rows.
`total_label`	Character. Label for the top row (default `"TOTAL SUBJECTS WITH AN EVENT"`).
`nr_char_values`	Character vector. Values in `grade_char` that are considered "Not Reported". Default includes multiple NR encodings.
`sort_grade`	Integer or character. **Grade used for ordering** within the reference arm (default 5). Use `"NOT REPORTED"` (or any synonym in `nr_char_values`) to sort by NR instead.
`debug`	Logical. If `TRUE`, prints debug summaries.
`uncoded_position`	Character. One of `c("count","last")`. Controls the placement of the UNCODED block: `"count"` = position by counts (default); `"last"` = force `SOC == "UNCODED"` to the end (per BY stratum) and `PT == "UNCODED"` last within that SOC.

Value

A tibble with columns:

- `stat`
- For each treatment and each grade bucket: `TRT<trt>_GRADE1`, . . . , `TRT<trt>_GRADE5`, `TRT<trt>_NOT_REPORTED`
- `sort_ord`, `sec_ord`

Key features

- **Grades from numeric and/or character sources**: Uses `grade_num` (1–5). If a character grade column exists (e.g., `"AETOCGR"`/`"AETOXGR"`), it is cleaned and mapped, with values in `nr_char_values` treated as *Not Reported*.
- **NR logic**: (a) For PT rows, a subject contributes the **max numeric grade** among 1–5 (NR ignored). (b) For the top **TOTAL** row, if any PT for the subject is **NR-only** (no numeric grade), the subject contributes to **NOT REPORTED**; otherwise to their **max numeric grade**.
- **Ordering**: Within SOC/PT, order is determined using counts from the reference arm `trtan_coln` filtered to `sort_grade` (fallback = all grades).
- **BY support**: Optional `by_var` (from AE) adds strata with optional `by_sort_var` to control strata ordering (numeric or character).
- **SOC totals**: `soc_totals = TRUE` adds a SOC subtotal row (max-grade logic).
- **Denominators**: Ns are computed from `dmdata` (or `pop_data`, if provided).
- **Big N behavior with BY**: controlled by `bigN_by` (TRT-only vs BY×TRT).
- **RTF-safe indent**: PT `stat` values can be indented using `indent_str`.
- **SAS-style rounding**: Percentages can follow SAS “round half away from zero” via `use_sas_round = TRUE`.
- **UNCODED placement**: `uncoded_position = c("count","last")`. With `"last"`, the block where `SOC == "UNCODED"` is forced to the very end (per BY stratum), and within that SOC the `PT == "UNCODED"` line is forced last. Detection is case-insensitive and robust to extra spaces/non-breaking spaces.

Index