Package ‘tidytlg’

June 15, 2024

Title  Create TLGs using the ‘tidyverse’

Version  0.1.5

Description  Generate tables, listings, and graphs (TLG) using ‘tidyverse.’
Tables can be created functionally, using a standard TLG process, or by
specifying table and column metadata to create generic analysis summaries.
The 'envsetup' package can also be leveraged to create environments for table
creation.

License  Apache License 2.0

URL  https://pharmaverse.github.io/tidytlg/main/,
https://github.com/pharmaverse/tidytlg

BugReports  https://github.com/pharmaverse/tidytlg/issues

Encoding  UTF-8

LazyData  true

RoxygenNote  7.3.1

Suggests  testthat (>= 2.1.0), knitr (>= 1.23), rmarkdown (>= 2.10),
renv (>= 0.13.2), shiny (>= 1.3.2), kableExtra (>= 1.3.4),
haven (>= 2.4.1), usethis (>= 1.6.3), withr (>= 2.3.0)

Imports  cli (>= 3.6.0), dplyr (>= 1.1.0), tibble (>= 2.1.3), magrittr
(>= 1.5), rlang (>= 0.4.10), tidyr (>= 1.0.0), stats (>=
3.6.0), stringr (>= 1.4.0), forcats (>= 0.5.1), purrr (>=
0.3.4), huxtable (>= 5.1.0), assertthat (>= 0.2.1), glue (>=
1.4.2), crayon (>= 1.4.1), methods, readxl (>= 1.3.1),
cellranger (>= 1.1.0), png (>= 0.1.7), ggplot2 (>= 3.3.2),
rstudioapi (>= 0.13)

Depends  R (>= 3.6.0)

VignetteBuilder  knitr

Config/testthat/edition  3

NeedsCompilation  no

Author  Nicholas Masel [aut],
        Steven Haesendonckx [aut],
add_bottom_borders

Description

Adds bottom borders to a huxtable

Usage

add_bottom_borders(ht, border_matrix = no_borders(ht), transform_fns = list())

Arguments

- **ht**: A huxtable object
- **border_matrix**: (optional) matrix A matrix indicating where to add the bottom borders. If NULL, then no borders are added.
- **transform_fns**: (optional) list of function A list of functions applied to the border_matrix. The functions have to accept two arguments:
  1. The huxtable.
  2. The border_matrix with dimensions matching huxtable.
    The functions in the list are applied sequentially to border_matrix.

Details

Adds bottom borders to a huxtable based on a matrix indicating where the borders should be put.

This function is responsible for adding bottom borders to a huxtable object. It supports borders spanning multiple columns and borders that are under neighbouring, single cells (or merged cells), but separate (see examples).

This feature has limitations. Mainly, it does not support both versions of the borders (continuous and separate) on the same line. In such a case, the borders in the resulting RTF look misaligned.

Value

A huxtable with added borders.

border_matrix details

You mark where the bottom borders should go in the table by passing a matrix. The matrix has to have the same number of columns as the passed huxtable and the number of rows lower by one than the passed huxtable. Each cell in border_matrix corresponds to a cell in huxtable (starting from the first row).

Internally, the function adds the first row of 0s to border_matrix before the execution. At that point, border_matrix's dimensions match ht's dimensions.

Table:
A border matrix:

\[
\begin{pmatrix}
1 & 1 \\
0 & 0
\end{pmatrix}
\]

The above border matrix puts a bottom border across the entire first row and no borders in the second row.

A border matrix:

\[
\begin{pmatrix}
1 & 2 \\
0 & 0
\end{pmatrix}
\]

The above border matrix puts one border under the first cell in the first row; and another border (separate from the first one) under the second cell in the first row. The second row stays without any borders.

**Functions transforming the border matrix**

The below functions can be passed to `gentlg()`’s `border_fns` argument to modify how `gentlg` renders the borders under the cells.

Border functions:

- `no_borders()`
- `spanning_borders()`
- `col_borders()`
- `single_border()`
- `row_border()`

`border_fns` will accept your own, custom functions as long as they adhere to the format. All the functions passed to `border_fns` need to accept two arguments:

- the first - the printed `huxtable` object,
- the second - a border matrix.

They also must return a matrix interpreted the same way as `border_matrix` passed to `add_bottom_borders` or `gentlg()`.
Examples

```r
border_matrix <- matrix(c(1, 1, 2, 0, 1, 1, 0, 0, 0), nrow = 3, ncol = 3)
ht <- huxtable::as_huxtable(
  data.frame(a = c(1, 2, 3), b = c("a", "b", "c"), c = c(TRUE, FALSE, TRUE))
)

# By default adds no borders
add_bottom_borders(ht, border_matrix)

# Adds spanning borders under cells with text in the second row
add_bottom_borders(ht, transform_fns = list(spanning_borders(2)))

# Adds spanning borders under cells with text in the second row and a border
# under a cell in row 3 and column 3
add_bottom_borders(ht, transform_fns = list(spanning_borders(2), single_border(3, 3)))

final <- data.frame(
  label = c("Overall", "Safety Analysis Set", "Any Adverse event\(^{a}\)", 
            "- Serious Adverse Event"),
  Drug_A = c("", "40", "10 (25\%)", "0"),
  Drug_B = c("", "40", "10 (25\%)", "0"),
  anbr = c(1, 2, 3, 4),
  roworder = c(1, 1, 1, 1),
  boldme = c(1, 0, 0, 0),
  newrows = c(0, 0, 1, 0),
  indentme = c(0, 0, 0, 1),
  newpage = c(0, 0, 0, 0)
)

# Add spanning bottom borders under the cells in the first row
gentlg(
  huxme = final,
  wcol = c(0.70, 0.15, 0.15),
  file = "TSFAEX",
  colheader = c("", "Drug A", "Drug B"),
  title = "This is Amazing Demonstration 1",
  footers = c(
    "Note: For demonstrative purposes only",
    "\(^{a}\) Subjects are counted once for any given event."),
  border_fns = list(no_borders, spanning_borders(1)))
)

# Tables with no bottom borders
gentlg(
  huxme = final,
  wcol = c(0.70, 0.15, 0.15),
  file = "TSFAEX",
  colheader = c("", "Drug A", "Drug B"),
  title = "This is Amazing Demonstration 1",
  footers = c(
    "Note: For demonstrative purposes only",
    "\(^{a}\) Subjects are counted once for any given event."),
)
border_fns = list(no_borders)
)

# Tables with a border under cell in the 3nd row and 3rd column,
# and borders under cells in the first row

gentlg(
  huxme = final,
  wcol = c(0.70, 0.15, 0.15),
  file = "TSFAEX",
  colheader = c("", "Drug A", "Drug B"),
  title = "This is Amazing Demonstration 1",
  footers = c(
    "Note: For demonstrative purposes only",
    "(\super a) Subjects are counted once for any given event."
  ),
  border_fns = list(no_borders, spanning_borders(1), single_border(3, 3))
)

# We discourage, but you can pass the border matrix directly
mat <- matrix(rep(0, 8 * 3), ncol = 3, nrow = 8)
mat[3, 3] <- 1

gentlg(
  huxme = final,
  wcol = c(0.70, 0.15, 0.15),
  file = "TSFAEX",
  colheader = c("", "Drug A", "Drug B"),
  title = "This is Amazing Demonstration 1",
  footers = c(
    "Note: For demonstrative purposes only",
    "(\super a) Subjects are counted once for any given event."
  ),
  bottom_borders = mat, # The same as a single border under 3nd row and 3rd column
  border_fns = list()
)

# clean up.
file.remove("tsfaex.rtf")

---

## add_format

Add the formatting variables of `indentme`, `newrows`, `newpage`, and `roworder` to the results dataframe

### Description

Add the formatting variables of `indentme`, `newrows`, `newpage`, and `roworder` to the results dataframe

### Usage

```r
add_format(df, tableby = NULL, groupby = NULL, .keep = FALSE)
```
add_indent

Arguments

df  (required) dataframe of results and must contain the anbr variable
tableby  (optional) character vector containing table by variables
groupby  (optional) character vector containing group by variables
.keep  (optional) should tableby and groupby variables be kept in the final dataframe. (default = FALSE)

Value
dataframe with the formatting variables indentme, newrows, newpage, and roworder added

Examples

df <- tibble::tibble(row_type =
c("TABLE_BY_HEADER", "HEADER", "BY_HEADER1", "N", "VALUE", "COUNTS", "UNIVAR", "NESTED", "NESTED"),
nested_level = c(NA, NA, NA, NA, NA, NA, NA, 1, 2),
group_level = c(0, 0, 0, 0, 0, 0, 0, 0, 0),
label = c(NA, NA, NA, NA, NA, "N", NA, NA, NA),
by = c(NA, NA, NA, NA, NA, NA, NA, NA, NA),
tableby = c(NA, NA, NA, NA, NA, NA, NA, NA, NA),
anbr = c(1:9))
add_format(df)

add_indent

Add indentation variable to the results dataframe

Description

Add the indentme variable to your results data. This drives the number of indents for the row label text (e.g. 0, 1, 2, etc.).

Usage

add_indent(df)

Arguments

df  dataframe of results that contains row_type and label and the optional nested_level and group_level variables.
Details

The `group_level` variable, which is added to the results dataframe by `freq()` and `univar()` calls, is needed to define indentation when by variables are used for summary.

The `nested_level` variable, which is added to the results dataframe by `nested_freq()`, is needed to define indentation for each level of nesting.

Both of these are added to the default indentation which is driven by `row_type`.

```
<table>
<thead>
<tr>
<th>row_type</th>
<th>default indentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_BY_HEADER</td>
<td>0</td>
</tr>
<tr>
<td>BY_HEADER[1-9]</td>
<td>0</td>
</tr>
<tr>
<td>HEADER</td>
<td>0</td>
</tr>
<tr>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>VALUE</td>
<td>2</td>
</tr>
<tr>
<td>NESTED</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Value

dataframe with the `indentme` variable added.

Examples

df <- tibble::tibble(row_type = c("TABLE_BY_HEADER", "HEADER", "BY_HEADER[1-9]", "N", "VALUE", "COUNTS", "UNIVAR", "NESTED", "NESTED"), nested_level = c(NA, NA, NA, NA, NA, NA, NA, 1, 2), group_level = c(0, 0, 0, 0, 0, 0, 0, 0, 0), label = c(NA, NA, NA, NA, NA, "N", NA, NA, NA), by = c(NA, NA, NA, NA, NA, "N", NA, NA, NA), tableby = c(NA, NA, NA, NA, NA, NA, NA, NA, NA))
add_indent(df)

add_newrows

Add the newrows variable to the results dataframe.

Description

The newrows variable is used by `gentlg()` to define when to add a blank row to the output. Data will be grouped by `anbr` and the variables passed into the `tableby` and `groupby` parameters. `newrows` will be set to 1 for the first record in each group, except for the first row in the data. The first row will always be set to 0.

Usage

```
add_newrows(df, tableby = NULL, groupby = NULL)
```
add_newrows

Arguments

df dataframe of results. must contain the anbr variable that is added by add_format()
tableby character vector containing table by variables used to generate the results
groupby character vector containing group by variables used to generate the results

Value
dataframe with the variable newrows and roworder added. newrows is used by gentlg to insert line breaks.

Examples

# Example showing how newrows is set to one for each new anbr except
# the first
tbl <-
structure(
  list(rowvar = c("RANDFL", "AGE", "AGE", "AGE", "AGE", "AGE"),
anbr = c(1L, 2L, 2L, 2L, 2L, 2L),
label = c("Analysis set: Subjects Randomized", "Age (Years)", "N",
"Mean (SD)", "Range", "IQ Range"),
row_type = c("COUNT", "UNIVAR", "UNIVAR", "UNIVAR", "UNIVAR", "UNIVAR")
),
row.names = c(NA,-6L),
class = c("tbl_df", "tbl", "data.frame")
)
add_newrows(tbl)

# Example of use when you have results summarized by one or more variables
tbl2 <- tibble::tribble(
~anbr, ~SEX, ~ETHNIC, ~label, ~row_type,
 01, 'F', NA, 'Sex : F', "TABLE_BY_HEADER",
 01, 'F', NA, '<65', "VALUE",
 01, 'F', NA, '65-80', "VALUE",
 01, 'F', NA, '>80', "VALUE",
 01, 'M', NA, 'Sex : M', "TABLE_BY_HEADER",
 01, 'M', NA, '<65', "VALUE",
 01, 'M', NA, '65-80', "VALUE",
 01, 'M', NA, '>80', "VALUE"
)
add_newrows(tbl2, tableby = "SEX")

tbl3 <- tibble::tribble(
~anbr, ~SEX, ~ETHNIC, ~label, ~row_type,
 01, 'F', NA, 'Sex : F', "TABLE_BY_HEADER",
 01, 'F', 'HISPANIC OR LATINO', 'HISPANIC OR LATINO', "BY_HEADER1",
 01, 'F', 'HISPANIC OR LATINO', '<65', "VALUE",
 01, 'F', 'HISPANIC OR LATINO', '>80', "VALUE",
 01, 'F', 'NOT HISPANIC OR LATINO', 'NOT HISPANIC OR LATINO', "BY_HEADER1",
)
add_newrows(tbl3, tableby = "SEX", groupby = "ETHNIC")

bind_table

Bind a set of tidytlg tables together with formatting variables

Description

bind_table combines analysis results with formatting variables (indentme, newrows, newpage) based on by variables (tablebyvar, rowbyvar), such that appropriate formatting (indentation, line break, page break) can be applied in creating the output. It can also attach the column metadata attribute, which will be automatically used in gentlg for creating output.

Usage

bind_table(
  ..., 
  colvar = NULL, 
  tablebyvar = NULL, 
  rowbyvar = NULL, 
  prefix = NULL, 
  add_count = FALSE, 
  add_format = TRUE, 
  column_metadata_file = NULL, 
  column_metadata = NULL, 
  tbltype = NULL
)

Arguments

... (required) a set of tidytlg tables to bind together

colvar (required) treatment variable within df to use to summarize. Required if add_count is TRUE.

tablebyvar (optional) repeat entire table by variable within df

rowbyvar (optional) any rowbyvar values used to create the table
bind_table

prefix (optional) text to prefix the values of tablebyvar with
add_count (optional) Should a count be included in the tablebyvar? (default = TRUE)
add_format (optional) Should format be added to the output table? This is done using the
             add_format function. (default = TRUE)
column_metadata_file (optional) An excel file for column_metadata. Does not change the behavior
             of the function binds the column metadata for gentlg. If a column_metadata
dataframe is passed in too, this is ignored.
column_metadata (optional) A dataframe containing the column metadata. This will be used in
             place of column_metadata_file.
tbltype (optional) A value used to subset the column_metadata_file.

Value

The tidytlg tables bound together reflecting the tablebyvars used

Examples

library(magrittr)

# bind tables together
t1 <- cdisc_ads1 %>%
  freq(colvar = "TRT01PN",
       rowvar = "ITTFL",
       statlist = statlist("n"),
       subset = ITTFL == "Y",
       rowtext = "Analysis set: ITT")

t2 <- cdisc_ads1 %>%
  univar(colvar = "TRT01PN",
         rowvar = "AGE",
         decimal = 0,
         row_header = "Age, years")

bind_table(t1, t2)

# bind tables together w/by groups
t1 <- cdisc_ads1 %>%
  freq(colvar = "TRT01PN",
       rowvar = "ITTFL",
       rowbyvar = "SEX",
       statlist = statlist("n"),
       subset = ITTFL == "Y",
       rowtext = "Analysis set: ITT")

t2 <- cdisc_ads1 %>%
  univar(colvar = "TRT01PN",
         rowvar = "AGE",
         rowbyvar = "SEX",
bind_table(t1, t2, rowbyvar = "SEX")

# bind tables together w/table by groups

t1 <- cdisc_adsl %>%
  freq(colvar = "TRT01PN",
       rowvar = "ITTFL",
       tablebyvar = "SEX",
       statlist = statlist("n"),
       subset = ITTFL == "Y",
       rowtext = "Analysis set: ITT")

  t2 <- cdisc_adsl %>%
  univar(colvar = "TRT01PN",
         rowvar = "AGE",
         tablebyvar = "SEX",
         decimal = 0,
         row_header = "Age, years")

bind_table(t1, t2, tablebyvar = "SEX")

# w/prefix
bind_table(t1, t2, tablebyvar = "SEX", prefix = "Gender: ")

# w/counts
bind_table(t1, t2, tablebyvar = "SEX", add_count = TRUE, colvar = "TRT01PN")

---

cdisc adae

ADAE data created from subsetting the CDISC ADAE dataset

Description
ADAE data created from subsetting the CDISC ADAE dataset

Usage
cdisc_adae

Format
A data frame with 84 rows and 55 variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STUDYID</td>
<td>Study Identifier</td>
</tr>
<tr>
<td>SITEID</td>
<td>Study Site Identifier</td>
</tr>
<tr>
<td>USUBJID</td>
<td>Unique Subject Identifier</td>
</tr>
<tr>
<td>SUBJID</td>
<td>Subject Identifier for the Study</td>
</tr>
<tr>
<td>TRTA</td>
<td>Actual Treatment</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRTAN</td>
<td>Actual Treatment (N)</td>
</tr>
<tr>
<td>AGE</td>
<td>Age</td>
</tr>
<tr>
<td>AGEGR1</td>
<td>Pooled Age Group 1</td>
</tr>
<tr>
<td>AGEGR1N</td>
<td>Pooled Age Group 1 (N)</td>
</tr>
<tr>
<td>RACE</td>
<td>Race</td>
</tr>
<tr>
<td>RACEN</td>
<td>Race (N)</td>
</tr>
<tr>
<td>SEX</td>
<td>Sex</td>
</tr>
<tr>
<td>SAFFL</td>
<td>Safety Population Flag</td>
</tr>
<tr>
<td>TRTSDT</td>
<td>Date of First Exposure to Treatment</td>
</tr>
<tr>
<td>TRTEDT</td>
<td>Date of Last Exposure to Treatment</td>
</tr>
<tr>
<td>ASTDT</td>
<td>Analysis Start Date</td>
</tr>
<tr>
<td>ASTDTDF</td>
<td>Analysis Start Date Imputation Flag</td>
</tr>
<tr>
<td>ASTDY</td>
<td>Analysis Start Relative Day</td>
</tr>
<tr>
<td>AENDT</td>
<td>Analysis End Date</td>
</tr>
<tr>
<td>AENDY</td>
<td>Analysis End Relative Day</td>
</tr>
<tr>
<td>ADURN</td>
<td>AE Duration (N)</td>
</tr>
<tr>
<td>ADURU</td>
<td>AE Duration Units</td>
</tr>
<tr>
<td>AETERM</td>
<td>Reported Term for the Adverse Event</td>
</tr>
<tr>
<td>AEELT</td>
<td>Lowest Level Term</td>
</tr>
<tr>
<td>AEELTCD</td>
<td>Lowest Level Term Code</td>
</tr>
<tr>
<td>AEEDECOD</td>
<td>Dictionary-Derived Term</td>
</tr>
<tr>
<td>AEPTCD</td>
<td>Preferred Term Code</td>
</tr>
<tr>
<td>AEHLT</td>
<td>High Level Term</td>
</tr>
<tr>
<td>AEHLTCD</td>
<td>High Level Term Code</td>
</tr>
<tr>
<td>AEHLGT</td>
<td>High Level Group Term</td>
</tr>
<tr>
<td>AEHLGTCRD</td>
<td>High Level Group Term Code</td>
</tr>
<tr>
<td>AEBODSYS</td>
<td>Body System or Organ Class</td>
</tr>
<tr>
<td>AESOC</td>
<td>Primary System Organ Class</td>
</tr>
<tr>
<td>AESOCCD</td>
<td>Primary System Organ Class Code</td>
</tr>
<tr>
<td>AESEV</td>
<td>Severity/Intensity</td>
</tr>
<tr>
<td>AESER</td>
<td>Serious Event</td>
</tr>
<tr>
<td>AESCAN</td>
<td>Involves Cancer</td>
</tr>
<tr>
<td>AESCONG</td>
<td>Congenital Anomaly or Birth Defect</td>
</tr>
<tr>
<td>AESDISAB</td>
<td>Persist or Signif Disability/Incapacity</td>
</tr>
<tr>
<td>AESDTH</td>
<td>Results in Death</td>
</tr>
<tr>
<td>AESHOSP</td>
<td>Requires or Prolongs Hospitalization</td>
</tr>
<tr>
<td>AESLIFE</td>
<td>Is Life Threatening</td>
</tr>
</tbody>
</table>
AESOD  Occurred with Overdose
AEREL  Causality
AEACN  Action Taken with Study Treatment
AEOUT  Outcome of Adverse Event
ASEQ  Sequence Number
TRTEMFL  Treatment Emergent Analysis Flag
AOCCFL  1st Occurrence of Any AE Flag
AOCCSFL  1st Occurrence of SOC Flag
AOCCPFL  1st Occurrence of Preferred Term Flag
AOCC02FL  1st Occurrence 02 Flag for Serious
AOCC03FL  1st Occurrence 03 Flag for Serious SOC
AOCC04FL  1st Occurrence 04 Flag for Serious PT
CQ01NAM  Customized Query 01 Name
AOCC01FL  1st Occurrence 01 Flag for CQ01

Source

CDISC SDTM/ADAM Pilot Project.

Description

ADLB data created from subsetting the CDISC ADLB dataset

Usage

cdisc_adlb

Format

A data frame with 2154 rows and 46 variables:

STUDYID  Study Identifier
SUBJID  Subject Identifier for the Study
USUBJID  Unique Subject Identifier
TRTA  Actual Treatment
TRTAN  Actual Treatment (N)
TRTSDT  Date of First Exposure to Treatment
TRTEDT  Date of Last Exposure to Treatment
AGE  Age
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGEGR1</td>
<td>Pooled Age Group 1</td>
</tr>
<tr>
<td>AGEGR1N</td>
<td>Pooled Age Group 1 (N)</td>
</tr>
<tr>
<td>RACE</td>
<td>Race</td>
</tr>
<tr>
<td>RACEN</td>
<td>Race (N)</td>
</tr>
<tr>
<td>SEX</td>
<td>Sex</td>
</tr>
<tr>
<td>COMP24FL</td>
<td>Completers of Week 24 Population Flag</td>
</tr>
<tr>
<td>DSRAEFL</td>
<td>Discontinued due to AE?</td>
</tr>
<tr>
<td>SAFFL</td>
<td>Safety Population Flag</td>
</tr>
<tr>
<td>AVISIT</td>
<td>Analysis Visit</td>
</tr>
<tr>
<td>AVISITN</td>
<td>Analysis Visit (N)</td>
</tr>
<tr>
<td>ADY</td>
<td>Analysis Relative Day</td>
</tr>
<tr>
<td>ADT</td>
<td>Analysis Date</td>
</tr>
<tr>
<td>VISIT</td>
<td>Visit Name</td>
</tr>
<tr>
<td>VISITNUM</td>
<td>Visit Number</td>
</tr>
<tr>
<td>PARAM</td>
<td>Parameter</td>
</tr>
<tr>
<td>PARAMCD</td>
<td>Parameter Code</td>
</tr>
<tr>
<td>PARAMN</td>
<td>Parameter (N)</td>
</tr>
<tr>
<td>PARCAT1</td>
<td>Parameter Category 1</td>
</tr>
<tr>
<td>A1LO</td>
<td>Analysis Range 1 Lower Limit</td>
</tr>
<tr>
<td>A1HI</td>
<td>Analysis Range 1 Upper Limit</td>
</tr>
<tr>
<td>R2A1LO</td>
<td>Ratio to Analysis Range 1 Lower Limit</td>
</tr>
<tr>
<td>R2A1HI</td>
<td>Ratio to Analysis Range 1 Upper Limit</td>
</tr>
<tr>
<td>BR2A1LO</td>
<td>Base Ratio to Analysis Range 1 Lower Limit</td>
</tr>
<tr>
<td>BR2A1HI</td>
<td>Base Ratio to Analysis Range 1 Upper Limit</td>
</tr>
<tr>
<td>ANL01FL</td>
<td>Analysis 01 - Special Interest Flag</td>
</tr>
<tr>
<td>ALBTRVAL</td>
<td>Amount Threshold Range</td>
</tr>
<tr>
<td>ANRIND</td>
<td>Analysis Reference Range Indicator</td>
</tr>
<tr>
<td>BNRIND</td>
<td>Baseline Reference Range Indicator</td>
</tr>
<tr>
<td>ABLFL</td>
<td>Baseline Record Flag</td>
</tr>
<tr>
<td>AENTMTFL</td>
<td>Last value in treatment visit</td>
</tr>
<tr>
<td>LBSEQ</td>
<td>Sequence Number</td>
</tr>
<tr>
<td>LBNRIND</td>
<td>Reference Range Indicator</td>
</tr>
<tr>
<td>LBSTRESN</td>
<td>Numeric Result/Finding in Standard Units</td>
</tr>
</tbody>
</table>

**Source**

CDISC SDTM/ADAM Pilot Project.
**cdisc adsl**

*ADSL data created from subsetting the CDISC ADSL with 15 subjects (5 subjects in each arm)*

**Description**

ADSL data created from subsetting the CDISC ADSL with 15 subjects (5 subjects in each arm)

**Usage**

`cdisc_adsl`

**Format**

A data frame with 15 rows and 49 variables:

- **STUDYID**: Study Identifier
- **USUBJID**: Unique Subject Identifier
- **SUBJID**: Subject Identifier for the Study
- **SITEID**: Study Site Identifier
- **SITEGR1**: Pooled Site Group 1
- **ARM**: Description of Planned Arm
- **TRT01P**: Planned Treatment for Period 01
- **TRT01PN**: Planned Treatment for Period 01 (N)
- **TRT01A**: Actual Treatment for Period 01
- **TRT01AN**: Actual Treatment for Period 01 (N)
- **TRTSDT**: Date of First Exposure to Treatment
- **TRTEDT**: Date of Last Exposure to Treatment
- **TRTDUR**: Duration of Treatment (days)
- **AVGDD**: Avg Daily Dose (as planned)
- **CUMDOSE**: Cumulative Dose (as planned)
- **AGE**: Age
- **AGEGR1**: Pooled Age Group 1
- **AGEGR1N**: Pooled Age Group 1 (N)
- **AGEU**: Age Units
- **RACE**: Race
- **RACEN**: Race (N)
- **SEX**: Sex
- **ETHNIC**: Ethnicity
- **SAFFL**: Safety Population Flag
ITTFL  Intent-To-Treat Population Flag
EFFFL  Efficacy Population Flag
COMP8FL  Completers of Week 8 Population Flag
COMP16FL  Completers of Week 16 Population Flag
COMP24FL  Completers of Week 24 Population Flag
DISCONFL  Did the Subject Discontinue the Study?
DSRAEFL  Discontinued due to AE?
DTHFL  Subject Died?
BMIBL  Baseline BMI (kg/m^2)
BMIBLGR1  Pooled Baseline BMI Group 1
HEIGHTBL  Baseline Height (cm)
WEIGHTBL  Baseline Weight (kg)
EDUCLVL  Years of Education
DISONSDT  Date of Onset of Disease
DURDIS  Duration of Disease (Months)
DURDSGR1  Pooled Disease Duration Group 1
VISIT1DT  Date of Visit 1
RFSTDTC  Subject Reference Start Date/Time
RFENDTC  Subject Reference End Date/Time
VISNUMEN  End of Trt Visit (Vis 12 or Early Term.)
RFENDT  Date of Discontinuation/Completion
DCDECOD  Standardized Disposition Term
EOSSTT  End of Study Status
DCREASCD  Reason for Discontinuation
MMSETOT  MMSE Total

Source
CDISC SDTM/ADAM Pilot Project.
ADVS data created from subsetting the CDISC ADVS dataset

Description

ADVS data created from subsetting the CDISC ADVS dataset

Usage

cdisc_advs

Format

A data frame with 1938 rows and 35 variables:

- **STUDYID** Study Identifier
- **SITEID** Study Site Identifier
- **USUBJID** Unique Subject Identifier
- **AGE** Age
- **AGEGR1** Pooled Age Group 1
- **AGEGR1N** Pooled Age Group 1 (N)
- **RACE** Race
- **RACEN** Race (N)
- **SEX** Sex
- **SAFFL** Safety Population Flag
- **TRTSDT** Date of First Exposure to Treatment
- **TRTEDT** Date of Last Exposure to Treatment
- **TRTP** Planned Treatment
- **TRTPN** Planned Treatment (N)
- **TRTA** Actual Treatment
- **TRTAN** Actual Treatment (N)
- **PARAMCD** Parameter Code
- **PARAM** Parameter
- **PARAMN** Parameter (N)
- **ADT** Analysis Date
- **ADY** Analysis Relative Day
- **ATPTN** Analysis Timepoint (N)
- **ATPT** Analysis Timepoint
- **AVISIT** Analysis Visit
- **AVISITN** Analysis Visit (N)
**char2factor**

<table>
<thead>
<tr>
<th>AVAL</th>
<th>Analysis Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE</td>
<td>Baseline Value</td>
</tr>
<tr>
<td>BASETYPE</td>
<td>Baseline Value</td>
</tr>
<tr>
<td>CHG</td>
<td>Change from Baseline</td>
</tr>
<tr>
<td>PCHG</td>
<td>Percent Change from Baseline</td>
</tr>
<tr>
<td>VISITNUM</td>
<td>Visit Number</td>
</tr>
<tr>
<td>VISIT</td>
<td>Visit Name</td>
</tr>
<tr>
<td>VSSEQ</td>
<td>Sequence Number</td>
</tr>
<tr>
<td>ANL01FL</td>
<td>Analysis 01 - Special Interest Flag</td>
</tr>
<tr>
<td>ABLFL</td>
<td>Baseline Record Flag</td>
</tr>
</tbody>
</table>

**Source**

CDISC SDTM/ADAM Pilot Project.

---

| char2factor | Convert character variable to a factor based off it’s numeric variable counterpart. |

**Description**

Convert character variable to a factor based off it’s numeric variable counterpart.

**Usage**

`char2factor(df, c_var, n_var)`

**Arguments**

- `df` : data frame.
- `c_var` : character variable within the data frame.
- `n_var` : numeric variable counter part within the data frame to control the levels.

**Value**

A factor.
Examples

df <- tibble::tribble(
  ~TRT01P, ~TRT01PN,
  "Placebo", 1,
  "Low Dose", 2,
  "High Dose", 3
 )

# alphabetical order
dplyr::arrange(df, TRT01P)

# change to factor with char2factor
df$TRT01P <- char2factor(df, "TRT01P", "TRT01PN")

# factor order
dplyr::arrange(df, TRT01P)

---

column_metadata Metadata describing table column layouts

Description

This is used by tlgsetup to prepare you input data to support the desired column layout.

Usage

column_metadata

Format

A data frame with one row per column for each table type and 6 variables:

- **tbltype** identifier used to group a table column layout
- **coldef** distinct variable values used, typically numeric and typically a treatment/main effect variable, think TRT01PN
- **decode** decode of coldef that will display as a column header in the table
- **span1** spanning header to display across multiple columns
- **span2** spanning header to display across multiple columns, second level
- **span3** spanning header to display across multiple columns, third level
**colBorders**  
*Adds borders under cells in a column*

**Description**  
Adds borders under cells in a column

**Usage**  
`colBorders(col, rows)`

**Arguments**  
- `col`: numeric the column of the table  
- `rows`: numeric the range of rows to include

**See Also**  
Other border functions: `noBorders()`, `rowBorder()`, `singleBorder()`, `spanningBorders()`

**freq**  
*Frequency counts and percentages*

**Description**  
Frequency counts and percentages for a variable by treatment and/or group.

**Usage**  
```r  
freq(  
  df,  
  denom_df = df,  
  colvar = NULL,  
  tablebyvar = NULL,  
  rowvar = NULL,  
  rowbyvar = NULL,  
  statlist =getOption("tidytlg.freq.statlist.default"),  
  decimal = 1,  
  nested = FALSE,  
  cutoff = NULL,  
  cutoff_stat = "pct",  
  subset = TRUE,  
  descending_by = NULL,  
  display_missing = FALSE,  
  rowtext = NULL,  
)  
```
row_header = NULL,
.kept = TRUE,
.ord = FALSE,
pad = TRUE,
...)

Arguments

df (required) dataframe containing records to summarize by treatment
denom_df (optional) dataframe used for population based denominators (default = df)
colvar (required) treatment variable within df to use to summarize
tablebyvar (optional) repeat entire table by variable within df
rowvar (required) character vector of variables to summarize within the dataframe
rowbyvar (optional) repeat rowvar by variable within df
statlist (optional) statlist object of stats to keep of length 1 or 2 specifying list of statistics and format desired (e.g statlist(c("N", "n (x.x \ (x.x)")))
decimal (optional) decimal precision root level default (default = 1)
nested (optional) INTERNAL USE ONLY. The default should not be changed. Switch on when this function is called by nested_freq() so we will not include the by variables as part of the group denominators (default = FALSE)
cutoff (optional) percentage cutoff threshold. This can be passed as a numeric cutoff, in that case any rows with greater than or equal to that cutoff will be preserved, others will be dropped. To specify a single column to define the cutoff logic, pass a character value of the form <colName> >= <value> and only that column will be used.
cutoff_stat (optional) The value to cutoff by, n or pct. (default = 'pct'). Can be done with multiple columns by adding & or | ex. col1 >= val1 & col2 >= val2
subset (optional) An R expression that will be passed to a dplyr::filter() function to subset the data.frame. This is performed on the numerator before any other derivations. Denominators must be preprocessed and passed through using denom_df.
descending_by (optional) The column or columns to sort descending counts. Can also provide a named list to do ascending order ex. c("VarName1" = "asc", "VarName2" = "desc") would sort by VarName1 in ascending order and VarName2 in descending order. In case of a tie in count or descending_by not provided, the columns will be sorted alphabetically.
display_missing (optional) Should the "missing" values be displayed? If missing values are displayed, denominators will include missing values. (default = FALSE)
rowtext (optional) A character vector used to rename the label column. If named, names will give the new level and values will be the replaced value. If unnamed, and the table has only one row, the rowtext will rename the label of the row. If the rowtext is unnamed, the table has no rows, and there is a subset, the table will be populated with zeros and the label will be the only row.
freq

row_header (optional) A character vector to be added to the table.
 kep (optional) Should the rowbyvar and tablebyvar be output in the table. If FALSE, rowbyvar will still be output in the label column. (default = TRUE)
.ord Should the ordering columns be output with the table? This is useful if a table needs to be merged or reordered in any way after build.
 pad (optional) A boolean that controls if levels with zero records should be included in the final table. (default = TRUE)
... (optional) Named arguments to be included as columns on the table.

Value
A dataframe of results

Sorting a 'freq' table
By default, a frequency table is sorted based on the factor level of the rowvar variable. If the rowvar variable isn’t a factor, it will be sorted alphabetically. This behavior can be modified in two ways, the first is the char2factor() function that offers an interface for releveling a variable based on a numeric variable, like VISITN. The second is based on the descending_by argument which will sort based on counts on a variable.

Examples

adsl <- data.frame(
  USUBJID = c("DEMO-101", "DEMO-102", "DEMO-103"),
  RACE = c("WHITE", "BLACK", "ASIAN"),
  SEX = c("F", "M", "F"),
  colnbr = factor(c("Placebo", "Low", "High"))
)

# Unique subject count of a single variable
freq(adsl
  ,colvar = "colnbr"
  ,rowvar = "RACE"
  ,statlist = statlist("n"))

# Unique subject count and percent of a single variable
freq(adsl
  ,colvar = "colnbr"
  ,rowvar = "RACE"
  ,statlist = statlist(c("N", "n (x.x%)")))

# Unique subject count of a variable by another variable
freq(adsl
  ,colvar = "colnbr"
  ,rowvar = "RACE"
  ,rowbyvar = "SEX"
  ,statlist = statlist("n"))

# Unique subject count of a variable by another variable using colvar and
# group to define the denominator
freq(adsl,
colvar = "colnbr",
rowvar = "RACE",
rowbyvar = "SEX",
statlist = statlist("n (x.x%)", denoms_by = c("colnbr", "SEX")))

# Cut records where count meets threshold for any column
freq(cdisc_adsl,
rowvar = "ETHNIC",
colvar = "TRT01P",
statlist = statlist("n (x.x%)")
cutoff = "5",
cutoff_stat = "n")

data <- tibble::tribble(
  ~anbr, ~func, ~df, ~rowvar, ~statlist, ~colvar,
  1, "freq", "cdisc_adsl", "ETHNIC", statlist("n"), "TRT01PN")
generate_results(data,
column_metadata = column_metadata,
tbltype = "type1")

data <- tibble::tribble(
  ~anbr, ~func, ~df, ~rowvar, ~statlist, ~colvar,
  "1", "freq", "cdisc_adsl", "ETHNIC", statlist(c("N", "n (x.x%)")), "TRT01PN")
generate_results(data,
column_metadata = column_metadata,
tbltype = "type1")

data <- tibble::tribble(
  "1", "freq", "cdisc_adsl", "ETHNIC",
  statlist = statlist("n (x.x%)"), colvar = "TRT01PN", cutoff = 5,
cutoff_stat = "n")
generate_results(data,
generate_results

column_metadata = column_metadata,
.tbltype = "type1"

# Cut records where count meets threshold for a specific column
table_metadata <- tibble::tibble(
anbr= 1, func = "freq", df = "cdisc_adsl", rowvar = "ETHNIC",
statlist = statlist("n (x.x%)"), colvar = "TRT01PN",
cutoff = 'col1 >= 3', cutoff_stat = "n"
)

generate_results(table_metadata,
    column_metadata = column_metadata,
    tbltype = "type1")

generate_results

Generate Results using Table and Column Metadata

Description

Generate Results using Table and Column Metadata

Usage

generate_results(
    table_metadata,
    column_metadata_file = NULL,
    column_metadata = NULL,
    env = parent.frame(),
    tbltype = NULL,
    add_count = FALSE
)

Arguments

table_metadata dataframe containing table metadata (see ?table_metadata for details)
column_metadata_file An excel file with the data for column_metadata. The file is read in with readxl::read_excel(). Should not be used with column_metadata argument. Results in a dataframe containing the column metadata that is passed to tlgsetup (see tlgsetup() for details). If a column_metadata dataframe is passed in too, this is ignored.
column_metadata A dataframe containing the column metadata. This will be used in place of column_metadata_file.
env environment to find dataframe specified in the table metadata (defaults to parent environment)
tbltype If used, this will be used to subset the column_metadata based on the tbltype column.
add_count Passed to bind_table() should counts be added for tablebyvars?
Value
dataframe of results

gentlg

Output a tidy table

Description
Generate and output a huxtable with desired properties. During this function call, the huxtable can be written to an RTF or displayed in HTML. gentlg is vectorized; see parameter descriptions to learn for which arguments.

Usage
gentlg(
  huxme = NULL,
  tlf = "Table",
  format = "rtf",
  colspan = NULL,
  idvars = NULL,
  plotnames = NULL,
  plotwidth = NULL,
  plotheight = NULL,
  wcol = 0.45,
  orientation = "portrait",
  opath = ".",
  title_file = NULL,
  file = NULL,
  title = NULL,
  footers = NULL,
  print.hux = TRUE,
  watermark = NULL,
  colheader = NULL,
  pagenum = FALSE,
  bottom_borders = "old_format",
  border_fns = list()
)

Arguments

huxme  (optional) For tables and listings, a list of input dataframes containing all columns of interest. For graphs, either NULL or a list of ggplot objects. Vectorized.

tlf  (optional) String, representing the output choice. Choices are "Table" "Listing" "Figure". Abbreviations are allowed eg "T" for Table. Strings can be either upper- or lowercase. Vectorized. (Default = "Table")
format (optional) String, representing the output format. Choices are "rtf" and "html". Strings can be either upper- or lowercase. (Default = "rtf")

colspan (optional) A list of character vectors representing the spanning headers to be used for the table or listing. The first vector represents the top spanning header, etc. Each vector should have a length equal to the number of columns in the output data frame. A spanning header is identified through the use of the same column name in adjacent elements. Vectorized.

idvars (optional) Character vector defining the columns of a listing where repeated values should be removed recursively. If NULL then all column names are used in the algorithm. If NA, then the listing remains as is.

plotnames (optional) Character vector containing the names of the png files, with their extension to be incorporated for figure outputs. The png files need to be located in the path defined by the parameter opath.

plotwidth (optional) Numerical value that indicates the plot width in cm for figure outputs. (Default = 6)

plotheight (optional) Numerical value that indicates the plot height in cm for figure outputs. (Default = 5)

wcol (optional) Can be a single numerical value that represents the width of the first column or a vector, specifying the lengths of all columns in the final table or listing.

When a single numerical value is used, this will be taken as the column width for the first column. The other columns will be equally spaced across the remainder of the available space. Alternatively, a vector can be used to represent the widths of all columns in the final output. The order of the arguments needs to correspond to the order of the columns in the huxme dataset, that are not part of the formatting algorithms (eg anbr, roworder, newrow, indentme, boldme, by_value, by_order). The sum of the widths in the vector needs to be less or equal to one. When 'format'="HTML" wcol can take only one value, the width of the first column. (Default = 0.45)

orientation (optional) String: "portrait" or "landscape". (Default = "portrait")

opath (optional) File path pointing to the output files (including .png files for graphs). (Default = ".")

title_file An Excel file that will be read in with readxl::read_excel() to be used as the title and footers argument. The use of title or footers will override the values passed by this argument. The file should be either an xls or xlsx file with the columns 'TABLE ID', 'IDENTIFIER', and 'TEXT'. The file will be read in, subset to where the tblid matches the tlf argument, and identifiers with 'title' or 'footnote' will be used to populate the table.

file (required) String. Output identifier. File name will be adjusted to be lowercase and have - and _ removed, this will not affect table title.

title (required) String. Title of the output. Vectorized.

footers (optional) Character vector, containing strings of footnotes to be included. Vectorized.

print.hux (optional) Logical, indicating whether the output should be printed to RTF ('format' = "rtf") or displayed as HTML ('format' = "HTML"). (Default = TRUE)
Note that RTF is written using `quick_rtf_jnj()` function and that the HTML is displayed via the `huxtable::print_html` function.

**watermark**

(optional) String containing the desired watermark for RTF outputs. Vectorized.

**colheader**

(optional) Character vector that contains the column labels for a table or listing. Default uses the column labels of `huxme`. Vectorized.

**pagenum**

(optional) Logical. When true page numbers are added on the right side of the footer section in the format page x/y. Vectorized. (Default = FALSE)

**bottom_borders**

(optional) Matrix or "old_format". A matrix indicating where to add the bottom borders. Vectorized. See `add_bottom_borders()` for more information. If "old_format", then borders are added to the colspan and colheader rows. (Default = "old_format").

**border_fns**

(optional) List. A list of functions that transform the matrix passed to `bottom_borders`. Vectorized. See `add_bottom_borders()` for more information.

**Value**

A list of formatted huxtables with desired properties for output to an RTF/HTML

**Huxme Details**

For tables and listings, formatting of the output can be dictated through the formatting columns (`newrows`, `indentme`, `boldme`, `newpage`), present in the input dataframe. The final huxtable will display all columns of the input dataframe, except any recognized formatting/sorting columns. For tables, the algorithm uses the column label as first column. The remaining columns are treated as summary columns. For graphs, you can pass a ggplot object directly into `huxme` and `gentlg` will save a png with with `ggplot2::ggsave()` and output an rtf.

**Author(s)**

Steven Haesendonckx shaesen2@its.jnj.com

Pelagia Alexandra Papadopoulou ppapadop@its.jnj.com

**References**

https://github.com/hughjonesd/huxtable

**Examples**

```r
final <- data.frame(
  label = c(
    "Overall", "Safety Analysis Set",
    "Any Adverse event{\super a}", "- Serious Adverse Event"
  ),
  Drug_A = c("", "40", "10 (25%)", "0"),
  Drug_B = c("", "40", "10 (25%)", "0"),
  anbr = c(1, 2, 3, 4),
  roworder = c(1, 1, 1, 1),
  boldme = c(1, 0, 0, 0),
  newrows = c(0, 0, 1, 0),
)```
# Produce output in rtf format

gentlg(
    huxme = final,
    wcol = c(0.70, 0.15, 0.15),
    file = "TSFAEX",
    title = "This is Amazing Demonstration 1",
    footers = c(
        "Note: For demonstrative purposes only",
        "{\super a} Subjects are counted once for any given event."
    )
)

# Pass in column headers instead of using variable name

gentlg(
    huxme = final,
    wcol = c(0.70, 0.15, 0.15),
    file = "TSFAEX",
    colheader = c("", "Drug A", "Drug B"),
    title = "This is Amazing Demonstration 1",
    footers = c(
        "Note: For demonstrative purposes only",
        "{\super a} Subjects are counted once for any given event."
    )
)

# Add spanning bottom borders under the cells in the second row

gentlg(
    huxme = final,
    wcol = c(0.70, 0.15, 0.15),
    file = "TSFAEX",
    colheader = c("", "Drug A", "Drug B"),
    title = "This is Amazing Demonstration 1",
    footers = c(
        "Note: For demonstrative purposes only",
        "{\super a} Subjects are counted once for any given event."
    ),
    border_fns = list(spanning_borders(2))
)

# Use a watermark

gentlg(
    huxme = final,
    wcol = c(0.70, 0.15, 0.15),
    file = "TSFAEX",
    colheader = c("", "Drug A", "Drug B"),
    title = "This is Amazing Demonstration 1",
    footers = c(
        "Note: For demonstrative purposes only",
        "{\super a} Subjects are counted once for any given event."
    )
)
nested_freq

Generate nested count/percent for two or three levels

Description

This will call \texttt{freq()} multiple times and combine the levels together. This is useful for adverse event and concomitant mediations.

Usage

\begin{verbatim}
nested_freq(
  df,
  denom_df = df,
  colvar = NULL,
  tablebyvar = NULL,
  rowvar = NULL,
  rowbyvar = NULL,
  statlist = getOption("tidytlg.nested_freq.statlist.default"),
  decimal = 1,
  cutoff = NULL,
  cutoff_stat = "pct",
  subset = TRUE,
  descending_by = NULL,
  display_missing = FALSE,
\end{verbatim}
Arguments

df (required) dataframe containing the two levels to summarize
denom_df (optional) dataframe containing records to use as the denominator (default = df)
colvar (required) treatment variable within df to use to summarize
tablebyvar (optional) repeat entire table by variable within df.
rowvar (required) nested levels separated by a star, for example AEBODSYS*AEDECOD, this can handle up to three levels.
rowbyvar (optional) repeat rowvar by variable within df
statlist (optional) count/percent type to return (default = "n (x.x)")
decimal (optional) decimal precision root level (default = 1)
cutoff (optional) numeric value used to cut the data to a percentage threshold, if any column meets the threshold the entire record is kept.
cutoff_stat (optional) The value to cutoff by, n or pct. (default = 'pct')
subset (optional) An R expression that will be passed to a dplyr::filter() function to subset the data.frame
descending_by (optional) The column or columns to sort descending values by. Can also provide a named list to do ascending order. ex. c("VarName1" = "asc", "VarName2" = "desc") would sort by VarName1 in ascending order and VarName2 in descending order. If not provided, the columns will be sorted alphabetically.
display_missing (optional) Should the "missing" values be displayed? (default = FALSE)
rowtext (optional) A character vector used to rename the label column. If named, names will give the new level and values will be the replaced value. If unnamed, and the table has only one row, the rowtext will rename the label of the row.
row_header (optional) A character vector to be added to the table.
.keep (optional) Should the rowbyvar and tablebyvar be output in the table. If FALSE, rowbyvar will still be output in the label column. (default = TRUE)
.ord Should the ordering columns be output with the table? This is useful if a table needs to be merged or reordered in any way after build.
... (optional) Named arguments to be included as columns on the table.
Value

A dataframe of nested results by colvar and optional tablebyvar. There are a few additional variable sets added to support multiple requirements.

The level variables (level1_, level2_, level3_) will carry down the counts for each level to every record. This allows for easy sorting of nested groups.

The header variables (header1, header2, header3) will flag the header for each level to ensure each level header is sorted to the top of the level.

The n variables ("n_") provide a numeric variable containing frequency for each colvar. This can be used to sort and filter records.

The pct variables ("pct_") provide a numeric variable containing percentages for each colvar. This can be used to sort and filter records.

Examples

```r
adae <- data.frame(
  SITEID = c("100", "100", "100", "200", "200", "200"),
  USUBJID = c("Demo1-101", "Demo1-102", "Demo1-103",
              "Demo1-104", "Demo1-105", "Demo1-106"),
  AEBODSYS = c("Cardiac disorders", "Cardiac disorders",
               "Respiratory, thoracic and mediastinal disorders",
               "Infections and infestations",
               "Skin and subcutaneous tissue disorders",
               "Infections and infestations"),
  AEDECOD = c("Arrhythmia supraventricular", "Cardiac failure",
              "Chronic obstructive pulmonary disease", "Pneumonia",
              "Pustular psoriasis", "Upper respiratory tract infection"),
 colnbr = structure(
    c(1L, 2L, 3L, 1L, 2L, 3L),
    .Label = c("Active", "Placebo", "Comparator"),
    class = "factor"
  )
)
# Frequency and percent for two levels of nesting
nested_freq(adae,
  colvar = "colnbr",
  rowvar = "AEBODSYS*AEDECOD",
  statlist = statlist("n (x.x%)"))

# Frequency and percent for three levels of nesting (for illustrative # purpose)
nested_freq(adae,
  colvar = "colnbr",
  rowvar = "SITEID*AEBODSYS*AEDECOD",
  statlist = statlist("n (x.x%)"))
```
# Cut records where pct meets threshold for any column
nested_freq(cdisc_adae,
  colvar = "TRTA",
  rowvar = "AEBODSYS*AEDECOD",
  statlist = statlist("n (x.x%)", distinct = TRUE),
  cutoff = 2,
  cutoff_stat = "n")

# Cut records where pct meets threshold for a specific column
nested_freq(cdisc_adae,
  rowvar = "AEBODSYS*AEDECOD",
  colvar = "TRTAN",
  statlist = statlist("n (x.x%)", distinct = TRUE),
  cutoff = "54 >= 2",
  cutoff_stat = "n")

# Frequency and percent for two levels of nesting and sort by descending
# active
nested_freq(adae,
  colvar = "colnbr",
  rowvar = "AEBODSYS*AEDECOD",
  statlist = statlist("n (x.x%)"),
  descending = "Active")

# Below illustrates how make the same calls to nested_freq() as above, using
# table and # column metadata along with generate_results().

column_metadata <- tibble::tribble(
  ~tbltype, ~coldef, ~decode,
  "type1", "1", "Placebo",
  "type1", "2", "Low",
  "type1", "3", "High")

table_metadata <- tibble::tribble(
  ~anbr, ~func, ~df, ~rowvar, ~tbltype, ~colvar, ~statlist,
  "1", "nested_freq", "cdisc_adae", "AEBODSYS*AEDECOD", "type1", "TRTP",
  statlist("n (x.x%)"))

generate_results(table_metadata,
  column_metadata_file = tidytlg_metadata(path))

# Frequency and percent for three levels of nesting (for illustrative purpose)
table_metadata <- tibble::tribble(
  ~anbr, ~func, ~df, ~rowvar, ~tbltype, ~colvar, ~statlist,
  "1", "nested_freq", "cdisc_adae", "SITEID*AEBODSYS*AEDECOD","type1",
  "TRTP", statlist("n (x.x%)"))

# Commented out because it takes too long
# generate_results(table_metadata, column_metadata)

# Cut records where pct meets threshold for a any column
column_metadata <- tibble::tribble(  
  ~tbltype, ~coldef, ~decode,  
  "type2", "1", "Placebo",  
  "type2", "2", "Active"
)
table_metadata <- tibble::tibble(  
  anbr = "1", func = "nested_freq", df = "cdisc_adae",  
  rowvar = "AEBODSYS*AEDECOD",  
  tbltype = "type2", colvar = "TRTP", statlist = statlist("n (x.x%)"),  
  dotdotdot = "cutoff = 5"
)

# generate_results(table_metadata,  
# column_metadata_file = tidytlg_metadata(path)

# Cut records where pct meets threshold for a specific column
table_metadata <- tibble::tibble(  
  anbr = "1", func = "nested_freq", df = "cdisc_adae",  
  rowvar = "AEBODSYS*AEDECOD",  
  tbltype = "type2", colvar = "TRTP", statlist = statlist("n (x.x%)"),  
  dotdotdot = "cutoff = 'col1 >= 5'"
)

# generate_results(table_metadata,  
# column_metadata_file = tidytlg_metadata(path)

# Frequency and percent for two levels of nesting and sort by descending col1
table_metadata <- tibble::tibble(  
  anbr = "1", func = "nested_freq", df = "cdisc_adae",  
  rowvar = "AEBODSYS*AEDECOD",  
  tbltype = "type2", colvar = "TRTP", statlist = statlist("n (x.x%)"),  
  dotdotdot = "descending = 'col1'"
)

# generate_results(table_metadata,  
# column_metadata_file = tidytlg_metadata(path)

---

no_borders

Removes all borders from the table

**Description**

Removes all borders from the table

**Usage**

no_borders(ht, matrix = NULL)
replace_na_with_blank

Arguments

- `ht`: huxtable object.
- `matrix`: matrix of bottom borders. Ignored. Included for the sake of compatibility with the interface of all border mutating functions.

See Also

Other border_functions: `col_borders()`, `row_border()`, `single_border()`, `spanning_borders()`

---

**replace_na_with_blank**  
*Replace NA with ""*

---

Description

Used to swap in "" for by variables so the headers sort correctly to the top

Usage

`replace_na_with_blank(x)`

Arguments

- `x`: variable to check for NA and replace with "".

Value

`x` with NA's replaced with "". Factors will add "" as the first level.

Examples

- `replace_na_with_blank(c("a", "b", NA))`
- `replace_na_with_blank(factor(c("a", "b", NA), levels = c("a", "b")))`
**rmdpstitle**

*Get Titles and Footnotes for all TLGs or one specific TLG*

**Description**

Get Titles and Footnotes for all TLGs or one specific TLG

**Usage**

```r
rmdpstitle(
  df,
  tblid = NULL,
  idvar = "tblid",
  identifier = "identifier",
  text = "text"
)
```

**Arguments**

- `df` dataframe with three variables; table name, row identifier (TITLE or FOOTNOTE), and title/footnote text to display
- `tblid` character vector containing the table id, optional, used to subset `df` to a specific table (defaults to `tblid`)
- `idvar` character vector containing the variable in `df` that contains your table id
- `identifier` character vector containing the variable name in `df` that contains your record identifier (defaults to "identifier")
- `text` character vector containing the variable name in `df` that contains your title and footnote text (defaults to "text")

**Value**

list of length two, the first element contains the titles as a tibble and the second contains the footnotes as a list

**Examples**

```r
tblid <- "TSIDEM01"

titles <- tibble::tribble(
  ~tblid, ~identifier, ~text,
  "TSIDEM01", "TITLE", "Demographics Example",
  "TSIDEM01", "FOOTNOTE1", "Example footnote."
)

title_foot <- rmdpstitle(titles, tblid)

title_foot[[1]]

title_foot[[2]]
```
Description

roundSAS is an alternative rounding function, ensuring that decimals equal or bigger than 5 are rounded upwards to the nearest number and returned as character vector.

Usage

roundSAS(x, digits = 0, as_char = FALSE, na_char = NULL)

Arguments

- **x**: Numeric vector.
- **digits**: An integer specifying the number of decimal places to be displayed after rounding. Default is 0.
- **as_char**: logical value indicating conversion of rounded numerical vector to character vector; default is FALSE.
- **na_char**: A character string indicating missing value; if not specified, "NA" is created.

Details

At the midpoint of a decimal place (e.g. 0.5, 1.5), the round function in R rounds to the nearest even number (i.e. 0.5 is rounded to 0; 1.5 is rounded to 2), whereas SAS rounds to the nearest number (i.e. 0.5 is rounded to 1; 1.5 is rounded to 2). The roundSAS function is an alternative rounding function for R that ensures rounding to the nearest number, as done in SAS. roundSAS comes from this Stack Overflow post https://stackoverflow.com/questions/12688717/round-up-from-5

Value

character vector of rounded values

Examples

```r
### input data vector with midpoint decimals
x <- c(-2.5, -1.5, -0.5, 0.5, 1.5, 2.5)
### rounds to integer
roundSAS(x, digits = 0)

### input data vector with a missing value
y <- c(8.65, 8.75, NA, 9.85, 9.95)
### rounds to tenths and label the missing value with "NE"
roundSAS(y, digits = 1, as_char = TRUE, na_char = "NE")
```
**single_border**

---

**row_border**

*Adds a continuous bottom border under a row*

**Description**

Adds a continuous bottom border under a row

**Usage**

`row_border(row)`

**Arguments**

- `row` numeric
  
  the row of the table

**See Also**

Other border functions: `col_borders()`, `no_borders()`, `single_border()`, `spanning_borders()`

---

**single_border**

*Adds a border under a cell*

**Description**

Adds a border under a cell

**Usage**

`single_border(row, col)`

**Arguments**

- `row` numeric
  
  the row of the cell
- `col` numeric
  
  the column of the cell

**See Also**

Other border functions: `col_borders()`, `no_borders()`, `row_border()`, `spanning_borders()`
**spanning_borders**

Adds borders under cells in a row, excluding the first column.

**Description**

Adds borders under cells that are not empty in a given row, omitting the first column of the row. The borders do not touch each other - they are separate.

**Usage**

```r
spanning_borders(row, cols = c(-1))
```

**Arguments**

- `row` numeric
  the row of the table
- `cols` numeric
  the columns of the row to consider

**See Also**

Other border_functions: `col_borders()`, `no_borders()`, `row_border()`, `single_border()`

---

**spanning_headers**

Spanning headers for outputs

**Description**

This will create the list object to be passed to gent1g(). You can create as many spanning headers as you like, just add variables prefixed with span to the column metadata.

**Usage**

```r
spanning_headers(column_metadata)
```

**Arguments**

- `column_metadata` dataframe
  containing the column metadata that is passed to tlgsetup() (see tlgsetup() for details)

**Value**

List of character vectors containing column headers for an output.
Examples

column_metadata <-
tibble::tribble(
  ~tbltype, ~coldef, ~decode, ~span1,
  "type1", "0", "Placebo", "",
  "type1", "54", "Low Dose", "Xanomeline",
  "type1", "81", "High Dose", "Xanomeline",
  "type1", "54+81", "Total Xanomeline", ""
)

spanning_headers(column_metadata)

---

statlist

Create a statlist interface for a table

Description

The statlist is the interface for the presentation of data in a tidytlg table.

Usage

statlist(stats, ...)

Arguments

stats (required) A character vector of statistics to display in the table.
... (optional) Additional configuration for stats. See sections below for allowable arguments.

Value

A statlist object that can be passed in the ’statlist’ argument of freq, nested_freq, or univar.

Statlists for freq() and nested_freq()

freq() statlists can be composed of n(count), N(denominator), and x.x(percentage, formatted with or without a percent sign). Denominators will include missing values if the ’display_missing’ argument is TRUE, otherwise they will be excluded. They can be arranged in the following ways:

- n
- n/N
- n (x.x)
- n (x.x%)
- n/N (x.x)
- n/N (x.x%)

The following other configurations are supported:
• denoms_by - Controls what groupings of variables should define the denominator. Variables should be passed as a quoted vector
• distinct - A boolean value. Should the numerator reflect distinct USUBJIDs or event counts. Defaults to TRUE which captures distinct subjects.
• distinct_by - A character value used to select the variable that should be used to "distinct" the freq tables. Defaults to USUBJID.
• zero_denom - The string to display when there are no records found in an entire denominator group. Defaults to "-".
• zero_n - The string to display when there are no records found for a numerator. Defaults to "0".

Statlists for univar statlists
• N
• SUM
• MEAN
• GeoMEAN
• SD
• SE
• CV
• GSD
• GSE
• MEANSD
• MEANSE
• MEDIAN
• MIN
• MAX
• RANGE
• Q1
• Q3
• IQRANGE
• MEDRANGE
• MEDIQRANGE
• MEAN_CI
• GeoMEAN_CI

where GeoMEAN: Geometric Mean, CV: Coefficient of Variation, GSD: Geometric Std. Dev., GSE: Geometric Std. Error, MEAN_CI: Mean (95% C.I.), GeoMEAN_CI: Geometric Mean (95% C.I.). In calculating geometric statistics, if there are zero values in the inputs, zero values will be excluded before calculating geometric statistics.
Examples

freq(
  mtcars,
  colvar = "gear",
  rowvar = "cyl",
  rowbyvar = "am",
  statlist = statlist("n/N (x.x)",
                     distinct = FALSE,
                     denoms_by = c("gear", "am"),
                     zero_denom = "_0_")
)

---

**table_metadata**

Metadata describing the data, functions and arguments needed to produce your results.

---

**Description**

Metadata describing the data, functions and arguments needed to produce your results.

**Usage**

table_metadata

**Format**

A data frame with one row per function call and 16 variables:

- **func** name of the function you wish to call
- **df** data frame to pass to the function call
- **subset** filter df records, this is passed directly to filter, ex. "AESER == 'Y'"
- **rowvar** variable being summarized that will pass to the function call
- **rowtext** row label text to display in the table
- **row_header** header text to display above row summary
- **statlist** list of statistics in the analysis, see individual functions for what is available per function (eg. "N, n (x.x)")
- **colvar** variable used to determine the columns of the table
- **decimal** decimal precision
- **rowbyvar** repeat rowvar summary by this variable/s, comma separated for multiple (eg. "ETHNIC, AGEGR1")
- **tablebyvar** repeat the entire table summary by this variable/s, comma separated for multiple (eg. "ETHNIC, AGEGR1")
- **denom_df** used to set denominators if df does not contain all required records
tidytlg_titles  

**Description**

Helper functions for returning files used in gentlg

**Usage**

```r
tidytlg_titles(path)
tidytlg_metadata(path)
```

**Arguments**

- `path`  
  Working directory of the project

**Value**

A character vector to the requested file.

---

tlgsetup  

**Description**

tlgsetup is useful for pre-processing total columns and columns composed of other columns. tlgsetup is called internally by `generate_results()` and can be run manually for custom tables.

**Usage**

```r
tlgsetup(
  df,
  var,
  column_metadata_file = NULL,
  column_metadata = NULL,
  tbltype = NULL
)
```
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>df</td>
<td>dataframe of records for analysis</td>
</tr>
<tr>
<td>var</td>
<td>character vector that identifies the numeric column/treatment variable</td>
</tr>
<tr>
<td>column_metadata_file</td>
<td>A file containing the column metadata. Read in with readxl::read_excel(). If a column_metadata dataframe is passed in too, this is ignored.</td>
</tr>
<tr>
<td>column_metadata</td>
<td>A dataframe containing the column metadata. This will be used in place of column_metadata_file.</td>
</tr>
<tr>
<td>tbltype</td>
<td>A value used to subset the column_metadata, both this and the file requirements are needed to bind the data to the table.</td>
</tr>
</tbody>
</table>

Value
dataframe with observations added to support the column type as well as the factor variable colnbr which is used as our new column summary variable. Regardless of if a coldef exists in data, the column will exist in the table.

Examples

df <-
tibble::tribble(  
  ~TRT01AN, ~USUBJID,  
  0, "A",  
  54, "B",  
  81, "C"  )

tlgsetup(df, "TRT01AN", column_metadata = column_metadata)

# Using a dataframe of column metadata
column_metadata <-
tibble::tribble(  
  ~tbltype, ~coldef, ~decode, ~span1,  
  "type1", "0", "Placebo", "",  
  "type1", "54", "Low Dose", "Xanomeline",  
  "type1", "81", "High Dose", "Xanomeline",  
  "type1", "54+81", "Total Xanomeline", ""  )

tlgsetup(df, "TRT01AN", column_metadata = column_metadata)

univar     Descriptive statistics

Description

Univariate statistics for a variables by treatment and/or group.
Usage

univar(
  df,
  colvar = NULL,
  tablebyvar = NULL,
  rowvar = NULL,
  rowbyvar = NULL,
  statlist = getOption("tidytlg.univar.statlist.default"),
  decimal = 1,
  precisionby = NULL,
  precisionon = NULL,
  wide = FALSE,
  alpha = 0.05,
  rowtext = NULL,
  row_header = NULL,
  .keep = TRUE,
  .ord = FALSE,
  ...
)

Arguments

df  (required) dataframe containing records to summarize by treatment
colvar  (required) character vector of the treatment variable within the dataframe
tablebyvar  (optional) repeat entire table by variable within df
rowvar  (required) character vector of variable to summarize within the dataframe
rowbyvar  (optional) repeat rowvar by variable within df
statlist  (optional) statlist object of stats to keep (default = statlist(c("N", "MEANSD", "MEDIAN", "RANGE", "IQRANGE")))
decimal  (optional) decimal precision root level, when using precisionby this will be used as the base decimal cap (default = 1)
precisionby  (optional) vector of by variable(s) to use when calculating parameter based precision
precisionon  (optional) variable to use when calculating parameter based precision. If precisionby is specified but not precisionon this will default to rowvar
wide  (optional) logical indicating to convert labels to column and columns to labels (default = FALSE)
alpha  (optional) alpha level for 2-sided confidence interval (default = 0.05)
rowtext  (optional) A text string to replace the label value on the table. Useful for tables with a single row.
row_header  (optional) A row to add as a header for the table.
.keep  (optional) Should the rowbyvar and tablebyvar be output in the table. If FALSE, rowbyvar will still be output in the label column. (default = TRUE)
.ord  Should the ordering columns be output with the table? This is useful if a table needs to be merged or reordered in any way after build.
...  (optional) Named arguments to be included as columns on the table.
Value
dataframe of results

Examples

```r
adsl <-
structure(
  list(
    USUBJID = c("DEMO-101", "DEMO-102", "DEMO-103", "DEMO-104",
               "DEMO-105", "DEMO-106"),
    AGE = c(59, 51, 57, 65, 21, 80),
    SEX = c("F", "M", "F", "M", "F", "M"),
    WEIGHTBL = c(83.6, 75, 84, 90, 65, 70),
    colnbr = structure(
      c(1L, 3L, 2L, 2L, 3L, 1L),
      .Label = c("Placebo", "Low", "High"),
      class = "factor"
    ),
    row.names = c(NA, 6L),
    class = "data.frame"
  ),
)

# N, Mean(SD), Median, Range, IQ Range for a rowvar by colvar
univar(adsl
  ,colvar = "colnbr"
  ,rowvar = "AGE"
)

# N and Mean for a rowvar by colvar
univar(adsl
  ,colvar = "colnbr"
  ,rowvar = "AGE"
  ,statlist = statlist(c("N", "MEAN")))

# N and Mean for a rowvar by colvar and a by variable
univar(adsl
  ,colvar = "colnbr"
  ,rowvar = "AGE"
  ,rowbyvar = "SEX"
  ,statlist = statlist(c("N", "MEAN")))

# Below illustrates how make the same calls to univar() as above, using table
# and column metadata # along with generate_results().

column_metadata <- tibble::tribble(
  ~tbltype, ~coldef, ~decode,
  "type1", "0", "Placebo",
  "type1", "54", "Low",
  "type1", "81", "High"
)

# N, Mean(SD), Median, Range, IQ Range for a rowvar by colvar
```
```r
table_metadata <- tibble::tribble(
  ~anbr, ~func, ~df, ~rowvar, ~tbltype, ~colvar,
  "1", "univar", "cdisc_adae", "AGE", "type1", "TRTA"
)

generate_results(table_metadata, column_metadata = column_metadata,
  tbltype = "type1")

# N and Mean for a rowvar by colvar

table_metadata <- tibble::tribble(
  ~anbr, ~func, ~df, ~rowvar, ~tbltype, ~colvar, ~statlist,
  "1", "univar", "cdisc_adae", "AGE", "type1", "TRTA",
  statlist(c("N","MEAN"))
)

generate_results(table_metadata, column_metadata = column_metadata,
  tbltype = "type1")

# N and Mean for a rowvar by colvar and a by variable

table_metadata <- tibble::tribble(
  ~anbr, ~func, ~df, ~rowvar, ~tbltype, ~colvar, ~statlist, ~by,
  "1", "univar", "cdisc_adae", "AGE", "type1", "TRTA",
  statlist(c("N","MEAN")), "SEX"
)

generate_results(table_metadata, column_metadata = column_metadata,
  tbltype = "type1")
```
Index

* CDISC
  cdisc adae, 12
cdisc adlb, 14
cdisc adsl, 16
cdisc advs, 18

* adae
cdisc adae, 12

* adlb
cdisc adlb, 14

* adsl
cdisc adsl, 16

* advs
cdisc advs, 18

* border_functions
  col_borders, 21
  no_borders, 34
  row_border, 38
  single_border, 38
  spanning_borders, 39

* datasets
  cdisc adae, 12
cdisc adlb, 14
cdisc adsl, 16
cdisc advs, 18
column_metadata, 20
table_metadata, 42

add_bottom_borders, 3
add_bottom_borders(), 28
add_format, 6
add_indent, 7
add_newrows, 8
bind_table, 10

cdisc adae, 12
cdisc adlb, 14
cdisc adsl, 16
cdisc advs, 18
cdisc adae (cdisc adae), 12
cdisc adlb (cdisc adlb), 14
cdisc adsl (cdisc adsl), 16
cdisc advs (cdisc advs), 18
char2factor, 19
col_borders, 21, 35, 38, 39
col_borders(), 4
column_metadata, 20
freq, 21
generate_results, 25
gentlg, 26
gentlg(), 4
nested_freq, 30
no_borders, 21, 34, 38, 39
no_borders(), 4
replace_na_with_blank, 35
rmdpstyl, 36
roundSAS, 37
row_border, 21, 35, 38, 39
row_border(), 4
single_border, 21, 35, 38, 39
single_border(), 4
spanning_borders, 21, 35, 38, 39
spanning_borders(), 4
spanning_headers, 39
statlist, 40

table_metadata, 42
tidytlg_metadata (tidytlg_titles), 43
tidytlg_titles, 43
tlgest, 43

univar, 44