Paper OS02

Tidytlg: An R Package for Clinical Reporting Using Tidyverse

Sheng-Wei Wang, Janssen R&D, Raritan, NJ, USA
Eli Miller, Atorus Research, Mountain Lakes, NJ, USA

ABSTRACT

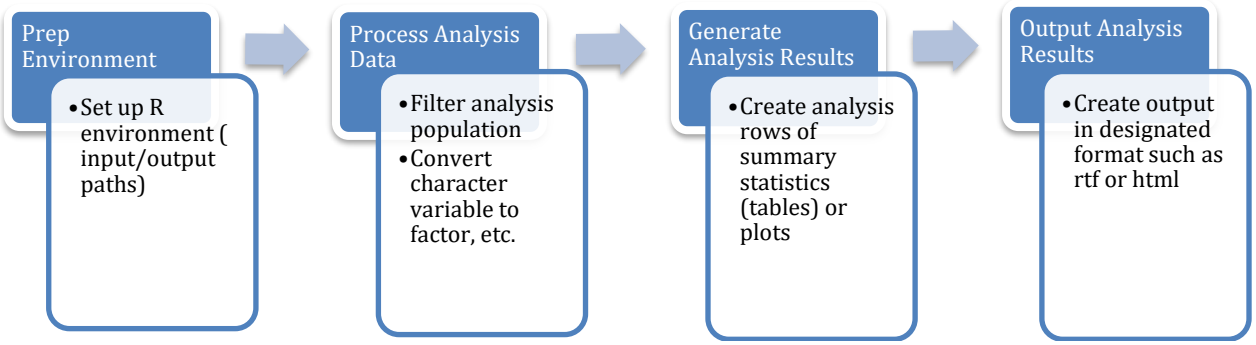
Tidytlg provides a framework for creating tables, listings, and graphs (TLGs) for clinical study reports. The TLG programming workflow consists of: (1) set up R environment, (2) process the analysis datasets (e.g., filter data, convert character variable to factor, etc.), (3) generate analysis results by creating analysis rows of summary statistics (for tables) or plots (for graphs), and (4) output analysis results in designated format such as rtf or html. Tidytlg offers a suite of analysis functions to summarize descriptive statistics (univariate statistics and counts (percentages)) for table creation and a function to convert analysis results to rtf/html outputs. For graphic output, tidytlg can integrate plot objects created by ggplot2 or a png file with titles and footnotes to produce rtf/html output. Examples of creating typical demographic, adverse events, laboratory tables as well as listing and graph using tidytlg are illustrated in this paper.

INTRODUCTION

Open-source technologies have been rapidly growing in recent years, and more and more pharmaceutical companies are starting to use R as an alternative to SAS or additional software for clinical trial reporting. To ease the transition for programmers from SAS to R, we developed the tidytlg package [1] for R to create the tables, listings, and graphs (TLG) for clinical study reports.

Tidytlg is developed using tidyverse [2] as a backbone and offers a suite of analysis functions to summarize descriptive statistics (univariate statistics and counts (percentages)) for table creation and a function to convert analysis results to RTF/HTML outputs. Tidytlg can integrate plot objects created by ggplot2 or a png file with titles and footnotes to produce RTF/HTML output for graphic work. Tidytlg function design aims to strike a balance between our internal SAS macro suites and R to reduce the learning curve of SAS programmers to begin adopting R for clinical trial reporting.

TLG PROGRAMMING WORKFLOW



The programming workflow of creating TLG is illustrated in the above flowchart. For setting up the R environment, users can define the path objects of input and output folders to be used consistently for all TLG programs within a study. The analysis datasets and other required inputs such as the file of storing titles and footnotes for each TLG are placed in the input folder, while the output folder will be used to store the output files. Please check out the envsetup package [3], which can be used to set up the R environment for TLG programming.

Tidytlg provides three core analysis functions for generating analysis results of descriptive summary statistics:

- **univar**: generate univariate statistics for numeric variables
- **freq**: generate counts and percentages for character variables
- **nested_freq**: summarize counts and percentages in a nested structure (counts within counts) such as body system by preferred terms of Adverse Events data

After creating the analysis results, the following two functions are used to format and output the analysis results to rtf or html files.

- **bind_table**: combine analysis results with formatting variables
- **gentlg**: convert analysis results to rtf or html files

KEY TIDYTLG ANALYSIS FUNCTION ARGUMENTS

The three core analysis functions have consistent convention of function arguments as following:

- **df**: analysis dataframe containing records to summarize
- **colvar**: column variable within **df** to summarize (such as the treatment variable)
- **rowvar**: analysis variable within **df** for producing summary statistics
- **statlist**: object of statistics list to display (e.g., statlist(c("N", "MEANSD", "MEDIAN", "RANGE", "IQRANGE"))
- **row_header**: text string to add as a header for the row summary
- **rowtext**: text string to display for a single row summary
- **rowbyvar**: variable within **df** to repeat the **rowvar** analysis
- **tablebyvar**: variable within **df** to repeat entire table analysis

EXAMPLES OF CREATING TLG

EAXMPLE 1: DEMOGRAPHIC TABLE

The first example illustrates the step-by-step process of building the R script of creating typical demographic output. This example uses the built-in dataset, cdisc_adsl, as the analysis dataframe for creating analysis results.

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

# Note cdisc_adsl is built into the package for use
ittpop <- cdisc_adsl %>%
  filter(ITTFL == "Y")

# frequency of Intend-to-Treat patients by planned treatment
tbl1 <- freq(ittpop,
             rowvar = "ITTFL",
             statlist = statlist("n"),
             colvar = "TRT01P",
             rowtext = "Analysis Set:  Intend-to-Treat Population",
             subset = ITTFL == "Y")

# N, MEAN (SD), MEDIAN, RANGE, IQ Range of age by planned treatment
tbl2 <- univar(ittpop,
               rowvar = "AGE",
               colvar = "TRT01P",
               row_header = "Age (Years)")

# frequency of gender by planned treatment
tbl3 <- freq(ittpop,
             rowvar = "SEX",
             statlist = statlist(c("N", "n (x.x%)")),
             colvar = "TRT01P",
             row_header = "Gender, n(%)")

# combine results together
tbl <- bind_table(tbl1, tbl2, tbl3)
```

After loading the required packages (dplyr and tidytlg) and processing the analysis dataframe, analysis results are generated sequentially by the following function calls:

- The first function call using **freq** creates a one row summary of analysis set population counts by treatment group.
- The second function call using **univar** creates multi-row summary of univariate statistics (N, Mean (SD), Median, Range, and IQ (interquartile) Range) for the age variable with the row header of 'Age (years)'.
- The third function call using **freq** creates the categorical counts and percentages of the sex variable with the row header of 'Gender, n(%)'.
- The fourth function call using **bind_table** combines the analysis results from above three function calls (tbl1, tbl2, tbl3) with the formatting variables of indentme (number of indentation), newrows (inserting a blank line: 1=yes, 0=no), and newpage (starting a newpage: 1=yes, 0=no).

	label	Placebo	Xanomeline High Dose	Xanomeline Low Dose	row_type	anbr	indentme	roworder	newrows	newpage
1	Analysis Set: Intend-to-Treat Population	5	5	5	HEADER	1	0	1	0	0
2	Age (Years)				HEADER	2	0	1	1	0
3	N	5	5	5	N	2	1	2	0	0
4	Mean (SD)	69.60 (14.398)	72.20 (9.230)	75.60 (6.731)	VALUE	2	2	3	0	0
5	Median	64.00	75.00	74.00	VALUE	2	2	4	0	0
6	Range	(52.0; 85.0)	(57.0; 81.0)	(68.0; 84.0)	VALUE	2	2	5	0	0
7	IQ range	(63.00; 84.00)	(71.00; 77.00)	(71.00; 81.00)	VALUE	2	2	6	0	0
8	Gender, n(%)				HEADER	3	0	1	1	0
9	N	5	5	5	N	3	1	2	0	0
10	F	2 (40.0%)	3 (60.0%)	1 (20.0%)	VALUE	3	2	3	0	0
11	M	3 (60.0%)	2 (40.0%)	4 (80.0%)	VALUE	3	2	4	0	0

The combined analysis results are stored in the tbl dataframe as shown above, which is then fed into the **gentlg** function to produce the rtf output shown below.

```
# Create rtf output ----------------------------------------------------
gentlg(huxme       = tbl ,
       orientation = "portrait",
       file        = "TDEMO01",
       title       = "Summary of demographics",
       footers     = "Note: SD: standard deviation; IQ: interquartile",
       colspan     = list(c("", "", "Xanomeline", "Xanomeline")),
       colheader   = c("", "Placebo", "High", "Low"),
       wcol        = .30)
```

The **gentlg** function uses the arguments below to control the appearance of the rtf output:

- orientation: portrait or landscape
- title and footers for specifying table title and footnotes
- colheader: column headers
- colspan: column spanning header

TDEMO01: Summary of demographics

		Xanomeline	
	Placebo	High	Low
Analysis Set: Intend-to-Treat Population	5	5	5
Age (Years)			
N	5	5	5
Mean (SD)	69.60 (14.398)	72.20 (9.230)	75.60 (6.731)
Median	64.00	75.00	74.00
Range	(52.0; 85.0)	(57.0; 81.0)	(68.0; 84.0)
IQ range	(63.00; 84.00)	(71.00; 77.00)	(71.00; 81.00)
Gender, n(%)			
N	5	5	5
F	2 (40.0%)	3 (60.0%)	1 (20.0%)
M	3 (60.0%)	2 (40.0%)	4 (80.0%)

Note: SD: standard deviation; IQ: interquartile

[tdemo01.rtf][C:/R/phuse2023_tidytlg/tdemo01.R] 26JAN2023, 14:57

For html output, users need to add two arguments in the **gentlg** call: format = “HTML”, print.hux = FALSE.

EXAMPLE 2: ADVERSE EVENT SUMMARY TABLE

This example presents the summary table of counting subjects with AEs by body system and preferred term.

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

# filter treatment emergent AE records for safety population
adae <- cdisc_adae %>%
  filter(SAFFL == "Y", TRTEMFL == "Y") %>%
  # filter one body system for illustration
  filter(AEBODSYS == "GENERAL DISORDERS AND ADMINISTRATION SITE CONDITIONS") %>%
  # convert AEBODSYS and AEDECOD to sentence case
  mutate(AEBODSYS = str_to_sentence(AEBODSYS), AEDECOD = str_to_sentence(AEDECOD)) %>%
  rename(TRT01AN = TRTAN)

# filter safety population
safetypop <- cdisc_adsl %>%
  filter(SAFFL == "Y")

# frequency of safety population
tbl1 <- freq(safetypop,
             rowvar = "SAFFL",
             statlist = statlist("n"),
             colvar = "TRT01AN",
             rowtext = "Analysis Set: Safety Population",
             subset = SAFFL == "Y")

# nested frequency of AEBODSYS by AEDECOD
tbl2 <- nested_freq(adae,
                    denom_df = safetypop,
                    colvar = "TRT01AN",
                    rowvar = "AEBODSYS*AEDECOD",
                    statlist = statlist("n (x.x%)"),
                    descending_by = "81",
                    row_header = "System organ class \\\n Preferred term")

# combine results together
tbl <- bind_table(tbl1, tbl2) %>%
  select(-AEBODSYS)
```

The built_in dataset, cdisc_adae, is used as the analysis dataframe that is further processed by filtering the safety population and treatment emergent AE records. For illustration purpose, the AE records with one body system are selected for creating the summary table. The key highlight of this table is the 2nd function call using **nested_freq** for summarizing nested counts and percentages:

- Use the denom_df argument to specify the denominator data from ADSL, since ADAE data only include subjects with AEs and not every subject in the analysis population has AEs.
- Use rowvar = “AEBODSYS*AEDECOD” to indicate the nested structure of summary counts and percentages
- Use the descending_by argument to specify the column for sorting the counts of body system and preferred terms within each body system in descending order.
- Additional arguments of cutoff and cutoff_stat is available to filter out low frequency of AEDECOD terms for reporting. Please see our package vignette for further details [4].

```
# Create rtf output ------------------------------------------------------------
gentlg(huxme       = tbl ,
       orientation = "portrait",
       file        = "TAE01",
       title       = "Summary of treatment-emergent adverse events",
       footers     = "Note: subjects are counted only once for any given event",
       colspan     = list(c("", "", "Xanomeline", "Xanomeline")),
       colheader   = c("", "Placebo", "High", "Low"),
       wcol        = .50)
```

The rtf output created by the gentlg call is shown below. The sorting of the nested counts is based on the 3rd column (TRT01AN=81 that is Xanomeline low dose group), which is specified in the descending_by argument of the **nested_freq** call.

TAE01: Summary of treatment-emergent adverse events

		Xanomeline	
	Placebo	High	Low
Analysis Set: Safety Population	5	5	5
System organ class			
Preferred term			
General disorders and administration site conditions	2 (40.0%)	3 (60.0%)	5 (100.0%)
Application site pruritus	1 (20.0%)	2 (40.0%)	5 (100.0%)
Application site erythema	1 (20.0%)	1 (20.0%)	4 (80.0%)
Fatigue	0	1 (20.0%)	2 (40.0%)
Application site irritation	0	1 (20.0%)	1 (20.0%)
Application site pain	0	0	1 (20.0%)
Application site vesicles	0	1 (20.0%)	1 (20.0%)
Application site dermatitis	0	1 (20.0%)	0
Application site urticaria	0	1 (20.0%)	0
Pyrexia	1 (20.0%)	0	0

Note: subjects are counted only once for any given event

[tae01.rtf][C:/R/phuse2023_tidytlg/tae01.R] 23JAN2023, 16:40

EXAMPLE 3: LABORATORY SUMMARY TABLE

The lab measurements are often collected across different time points of a clinical study. The lab summary table will then need to summarize the descriptive statistics across different time points.

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

# filter safety population records
adlb <- cdisc_adlb %>%
  filter(SAFFL == "Y") %>%
  # filter ALT and AST records for illustration
  filter(PARAMCD %in% c("ALT","AST")) %>%
  filter(AVISIT %in% c("Baseline", "Week 4", "Week 8", "Week 12"))

# convert AVISIT to factor based on AVISITN
adlb$AVISIT <- char2factor(adlb, "AVISIT", "AVISITN")

# call univar to summarize descriptive stats by PARAM and AVISIT
tbl1 <- univar(adlb,
               colvar = "TRTAN",
               tablebyvar = "PARAM",
               rowvar = "AVAL",
               rowbyvar = "AVISIT",
               statlist = statlist(c("N","MEANSD","MEDIAN","RANGE")))

# bind analysis results with formatting variables
tbl <- bind_table(tbl1, tablebyvar = "PARAM", rowbyvar = "AVISIT")
```

```
# Create rtf output ------------------------------------------------------------
gentlg(huxme       = tbl,
       orientation = "portrait",
       file        = "TLB01",
       title       = "Summary of descriptive statistics for lab values",
       footers     = "Note: SD: standard deviation",
       colspan     = list(c("", "", "Xanomeline", "Xanomeline")),
       colheader   = c("", "Placebo", "High", "Low"),
       wcol        = .30)
```

This example uses the built-in cdisc_adlb dataset, which is further processed by selecting two lab parameters (ALT and AST, both are liver function measurements) for illustration. The analysis time points focus on the visits of baseline, week 4, week 8, and week 12. The analysis visit (AVISIT) variable is converted from character to factor based on its numeric counterpart of AVISITN. Therefore, the summary results by visit can be displayed in the correct order. The key highlight of this table is the **univar** function call with the by-processing variables (rowbyvar and tablebyvar) to summarize univariate descriptive statistics of the lab data:

- Use rowbyvar = “AVISIT” to repeat descriptive analysis for each visit within a parameter
- Use tablebyvar = “PARAM” to repeat the entire by visit analysis for each parameter.

The analysis results obtained by the univar call is then passed into the bind_table call with the same arguments of rowbyvar and tablebyvar used in the univar call. This step is to create the appropriate formatting variables (indentme, newrows, and newpage) when by-processing variables are involved in creating analysis summary.

The snapshot of the rtf output for ALT is shown below to illustrate the output layout, where univariate descriptive statistics are summarized across four visits. The entire by-visit analysis is repeated for AST (not shown below due to page limit).

TLB01: Summary of descriptive statistics for lab values

	Placebo	Xanomeline High	Xanomeline Low
Alanine Aminotransferase (U/L)			
Baseline			
N	5	5	5
Mean (SD)	20.40 (5.273)	18.80 (3.114)	22.60 (8.019)
Median	22.00	18.00	23.00
Range	(15.0; 27.0)	(16.0; 23.0)	(15.0; 34.0)
Week 4			
N	5	4	5
Mean (SD)	21.80 (9.230)	16.75 (4.500)	24.40 (7.537)
Median	18.00	15.50	21.00
Range	(15.0; 38.0)	(13.0; 23.0)	(16.0; 35.0)
Week 8			
N	3	2	4
Mean (SD)	15.67 (6.028)	17.00 (1.414)	26.00 (7.703)
Median	15.00	17.00	27.50
Range	(10.0; 22.0)	(16.0; 18.0)	(16.0; 33.0)
Week 12			
N	3	1	4
Mean (SD)	18.67 (7.371)	14.00 (-)	26.00 (10.231)
Median	16.00	14.00	24.50
Range	(13.0; 27.0)	(14.0; 14.0)	(16.0; 39.0)

EXAMPLE 4: LISTING OF ADVERSE EVENTS

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

# process AE data for listing
tbl <- cdisc_adae %>%
  filter(SAFFL == "Y" & TRTEMFL == "Y") %>%
  mutate(SAEFL = if_else(AESER == "Y", "Yes", "No"),
         DTHFL = if_else(AEOUT == "FATAL", "Yes", "No"),
         AETERM = str_to_sentence(AETERM)) %>%
  # filter records of 2 subjects for illustration
  filter(USUBJID %in% c("01-701-1015", "01-701-1023")) %>%
  # select columns to display in listing
  select(TRTA, USUBJID, ASTDY, AETERM, SAEFL, DTHFL)

# Create rtf output ------------------------------------------------
gentlg(huxme       = tbl,
       tlf         = "l",
       orientation = "landscape",
       file        = "LSAE01",
       title       = "Listing of Adverse Events",
       idvars      = c("TRTA", "USUBJID"),
       wcol        = 0.10,
       colheader   = c("Treatment Group",
                       "Subject ID",
                       "Study Day of AE",
                       "Verbatim Term",
                       "Serious",
                       "Fatal"))
```

For creating the listing output, users just need to prepare the data and assign it to the tbl dataframe. In this example, the cdisc_adae data is processed by:

- filtering treatment emergent AE records for the safety population
- converting the serious AE flag into categories of yes and no
- deriving the fatal flag based on the AE outcome
- converting the AE verbatim term to sentence case for display
- selecting the variables to display for listing.

For illustration purpose, only two subjects are included for the AE listing shown below. In the **gentlg** call, key highlights are:

- Use tlf = "l" to create listing
- Use idvars argument to remove repeated values. In this example, the argument of idvars = c("TRTA","USUBJID") enables the removal of repeated values of treatment group and subject ID in the listing output shown below.

LSAE01: Listing of Adverse Events

Treatment Group	Subject ID	Study Day of AE	Verbatim Term	Serious	Fatal
Placebo	01-701-1015	2	Application site erythema	No	No
		2	Application site pruritus	No	No
		8	Diarrhoea	No	No
	01-701-1023	3	Erythema	No	No
		3	Erythema	No	No
		22	Atrioventricular block second degree	No	No
		3	Erythema	No	No

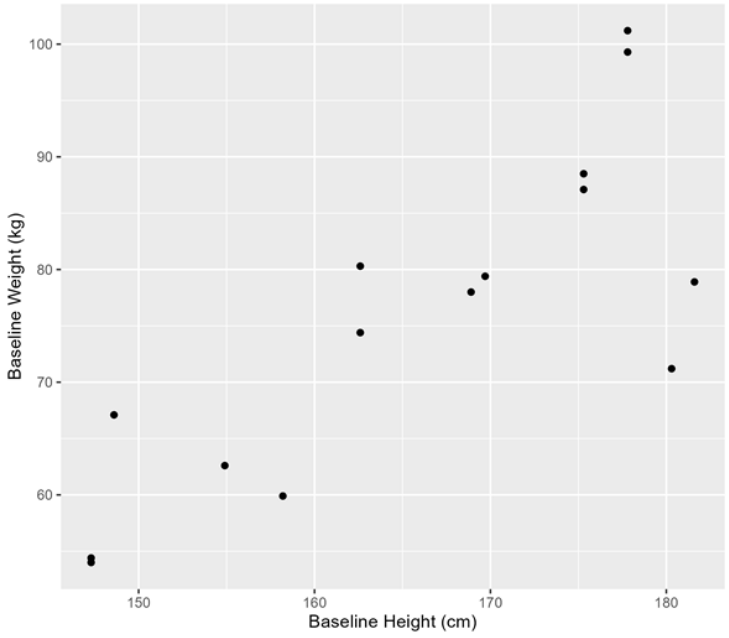
[lsae01.rtf][C:/R/phuse2023_tidytlg/lsae01.R] 25JAN2023, 15:18

EXAMPLE 5: GRAPH OUTPUT

```
library(dplyr, warn.conflicts = FALSE)
library(ggplot2)
library(tidytlg)

# process data
adsl <- cdisc_adsl %>%
  filter(ITTFL == "Y")

# generate plot
plot <- ggplot(data = adsl, aes(x = HEIGHTBL, y = WEIGHTBL)) +
  geom_point() +
  labs(x = "Baseline Height (cm)",
       y = "Baseline Weight (kg)")

# Create rtf output ----------------------------------------------------
gentlg(huxme       = plot,
       tlf         = "f",
       orientation = "landscape",
       file        = "Graph01",
       title       = "Scatter plot of baseline body weight and height")
```

For generating graph output, users need to create a png file or a plot object produced by ggplot2, which is then fed into the **gentlg** function to integrate with title and footnote to produce rtf/html output. In this example, a scatter plot object of baseline weight versus baseline height in the cdisc_adsl dataset is created by ggplot2. Then, this plot object is directly used by the **gentlg** function for producing the rtf output shown below. Users need to use tlf = 'f' in the gentlg call for producing graph output. For using png file to create graph output, users need to include the arguments of plotnames, plotwidth, and plotheight. Please visit the get started page of the tidytlg documentation for further details [5].

Graph01: Scatter plot of baseline body weight and height



[graph01.rtf][C:/R/phuse2023_tidytlg/graph01.R] 25JAN2023, 16:13

METADATA-DRIVEN TABLE CREATION

In addition to the functional method of building table output step-by-step in the above examples, table creation can also be implemented using the metadata method that utilizes column and table metadata to create a generic script for each output.

COLUMN METADATA

Column metadata defines the column structure of the table layout and includes the following variables:

- tbltype: identifier used to group a table column layout
- coldef: distinct variable values used, typically numeric and typically a treatment 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 (the lowest level)
- span2: spanning header to display across multiple columns, second level
- span3: spanning header to display across multiple columns, third level

Different types of column layouts identified by different table type (tbltype) can be stored in an excel file called column_metadata.xlsx. Please see a snapshot of column_metadata.xlsx below. Within each tbltype, the column definition (coldef) variable defines the order of the column based on the column variable used for creating the output.

	A	B	C	D	E	F
1	**tbltype**	**coldef**	**decode**	**span1**	**span2**	**span3**
2	type1	0	Placebo			
3	type1	54	Low Dose	Xanomeline		
4	type1	81	High Dose	Xanomeline		
5	type2	0	Placebo			
6	type2	54	Low Dose	Xanomeline		
7	type2	81	High Dose	Xanomeline		
8	type2	54+81	Combined	Xanomeline		
9	type3	0	Placebo			
10	type3	54	Low Dose	Xanomeline		
11	type3	81	High Dose	Xanomeline		
12	type3	54+81	Combined	Xanomeline		
13	type3	0+54+81	Total			

For tbltype = “type1”, there are 3 records for defining the column structure:

- the 1st column represents the treatment group of TRT01PN = 0 with the column header of Placebo
- the second and third columns represent the Low Dose and High Dose groups respectively with the spanning header of Xanomeline.

Users can also include the column that is derived from combination of individual columns. For example, the tbltype of type3 include the 4th column of combined Low Dose and High Dose as well as the 5th column of total group. To include the combined columns in the analysis output, users need to use the **tlgsetup** function along with the column metadata in the processing data step. Please see our package vignette [6] for further details.

TABLE METADATA

Table metadata is a data frame describing the data, functions and arguments needed to produce your table results. The table metadata shown below can be used to create the same table output as the example 1. Each row in the table metadata describes how a tbl chunk will be created by the function defined in the func column. The rest of the columns defines the arguments (i.e., df, colvar, rowvar, rowtext, row_header, statlist, subset) that will be passed into the function. Once table metadata is defined, users just need to call the **generate_results** function with the column metadata define in the column_metadata_file and tbltype arguments to create the tbl dataframe.

```
# define table metadata
table_metadata <- tibble::tribble(
  ~func,      ~df, ~rowvar,      ~rowtext,       ~row_header, ~statlist,    ~subset,
  "freq",  "adsl", "ITTFL","Analysis set: ITT",           NA, statlist("n"),"ITTFL == 'Y'",
  "univar","adsl", "AGE",                  NA,"Age (Years)",           NA,           NA,
  "freq",  "adsl", "SEX",                  NA,     "Gender", statlist(c("N", "n (x.x%)")),NA
) %>%
  mutate(colvar  = "TRT01PN")

# Generate results
tbl <- generate_results(table_metadata,
        column_metadata_file = system.file("extdata/column_metadata.xlsx", package = "tidytlg"),
        tbltype = "type1")
```

Please see the tbl dataframe below, which has the same analysis results of the demographic table in the example 1. The column variables (col1. col2, col3) reflect the column order defined in the column metadata. When this tbl dataframe is passed into the **gentlg** call, the corresponding column headers (decode) and spanning header (span1) defined in the column metadata will be populated in the rtf output.

	label	col1	col2	col3	row_type	func	anbr	indentme	roworder	newrows
1	Analysis set: ITT	5	5	5	HEADER	freq	1	0	1	0
2	Age (Years)				HEADER	univar	2	0	1	1
3	N	5	5	5	N	univar	2	1	2	0
4	Mean (SD)	69.60 (14.398)	75.60 (6.731)	72.20 (9.230)	VALUE	univar	2	2	3	0
5	Median	64.00	74.00	75.00	VALUE	univar	2	2	4	0
6	Range	(52.0; 85.0)	(68.0; 84.0)	(57.0; 81.0)	VALUE	univar	2	2	5	0
7	IQ range	(63.00; 84.00)	(71.00; 81.00)	(71.00; 77.00)	VALUE	univar	2	2	6	0
8	Gender				HEADER	freq	3	0	1	1
9	N	5	5	5	N	freq	3	1	2	0
10	Male	3 (60.0%)	4 (80.0%)	2 (40.0%)	VALUE	freq	3	2	3	0
11	Female	2 (40.0%)	1 (20.0%)	3 (60.0%)	VALUE	freq	3	2	4	0

MISSING DATA TREATMENT

The analysis datasets used for creating TLG outputs are often generated by SAS and imported into R using the **read_sas** function from the haven package [8]. Missing values in character variable have different representations between SAS and R: a blank (`""`) in SAS, and the symbol of **NA** (not available) in R. When SAS datasets are

imported into R, the character missing values are not converted to **NA** and they are kept as is (i.e. blank (`""`)). This will create confusion in the downstream analysis results. It is recommended to perform further data manipulation of converting blank (`""`) to NA in analysis datasets before creating TLGs.

If blank (`""`) is not converted to NA, it will be treated as another string character and the analysis results of calling the **freq** function will include the category of blank (`""`). Please see an example below, where we manipulate the cdisc_adsl data to resemble the SAS missing of blank (`""`) in the RACE variable for 2 subjects.

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

# manipulate the RACE var in cdisc_adsl to resemble SAS missing of ""
adsl2 <- cdisc_adsl %>%
  mutate(RACE = case_when(USUBJID %in% c("01-701-1015", "01-701-1023") ~ "",
                          TRUE ~ RACE))

# call freq to summarize RACE category to see the impact of ""
tbl2 <- freq(adsl2,
             rowvar = "RACE",
             statlist = statlist(c("N", "n (x.x%)")),
             colvar = "TRT01P",
             row_header = "Race, n(%)")
```

After calling the freq function, the blank value is included as one of the RACE categories in the summary counts and percentages.

label	Placebo	Xanomeline High Dose	Xanomeline Low Dose
Race, n(%)			
N	5	5	5
	2 (40.0%)	0	0
WHITE	3 (60.0%)	5 (100.0%)	5 (100.0%)

If we perform further data manipulation of converting blank (`""`) to **NA**, the default behavior of the freq function call is to remove the missing values and only the non-missing values are summarized as shown below.

```
# convert the blank "" value in RACE var into NA
adsl3 <- adsl2 %>%
  mutate(RACE = case_when(RACE == "" ~ NA_character_,
                          TRUE ~ .data$RACE))

# call freq to summarize RACE category to see how NA is treated by default
tbl3a <- freq(adsl3,
             rowvar = "RACE",
             statlist = statlist(c("N", "n (x.x%)")),
             colvar = "TRT01P",
             row_header = "Race, n(%)")
```

label	Placebo	Xanomeline High Dose	Xanomeline Low Dose
Race, n(%)			
N	3	5	5
WHITE	3 (100.0%)	5 (100.0%)	5 (100.0%)

However, sometimes the analysis specification may require showing the missing counts for character variables. We have designed the freq function to accommodate this requirement with the display_missing argument. For displaying the missing counts of the analysis variable, please set the display_missing = TRUE in the **freq** call (the default is FALSE).

```
# call freq to summarize RACE category by turning on display_missing
tbl3b <- freq(adsl3,
              rowvar = "RACE",
              statlist = statlist(c("N", "n (x.x%)")),
              colvar = "TRT01P",
              row_header = "Race, n(%)",
              display_missing = TRUE)
```

label	Placebo	Xanomeline High Dose	Xanomeline Low Dose
Race, n(%)			
N	5	5	5
WHITE	3 (60.0%)	5 (100.0%)	5 (100.0%)
Missing	2 (40.0%)	0	0

CONCLUSION

We have provided the overview of the tidytlg package and illustrated the use cases of creating typical demographic, adverse events, laboratory tables as well as listing and graph using tidytlg. The design goal of developing tidytlg was to increase the adoption of R for clinical reporting by reducing the learning curve from SAS to R. The TLG programming workflow can be implemented with tidytlg by: (1) the functional method, which is intuitive and easy to learn, or (2) the metadata method that is efficient for generic table creation. Tidytlg has been open sourced in the Pharmaverse GitHub organization[1], and we welcome your feedback on bug fixes, functionality improvements, and contributions to improving this package.

The intended use of the tidytlg package is to perform safety analysis, since only the analysis functions of creating descriptive summary statistics are developed in this package. For efficacy analysis, users can use the hybrid approach by: (1) developing the open code of using the required statistical R package to obtain the analysis results (e.g., use the survival package to analyze the time to event data), and (2) perform data wrangling to transform or

reformat the analysis result into the tbl structure, which can be passed into the **gentlg** function for creating rtf or html outputs. Please see our package vignette of manipulating tbl data frame [7] for further details of the tbl structure.

REFERENCES

1. Masel N, Haesendonckx S, Alexandra Papadopoulou P, Wang S, Miller E, Kosiba N, Ceney A (2023). tidytlg: Create TLGs using the Tidyverse. R package version 0.0.1, https://github.com/pharmaverse/tidytlg.
2. Wickham H, Averick M, Bryan J, Chang W, McGowan L, François R, et al. Welcome to the tidyverse. J Open Source Softw. 2019 Nov 21;4(43):1686, https://www.tidyverse.org/packages/.
3. Masel N, Stackhouse M, Ceney A (2022). envsetup: Support the Setup of the R Environment for Clinical Trial Programming Workflows. R package version 0.0.1, https://github.com/pharmaverse/envsetup.
4. Tidytlg package vignette: Nested Frequency Analysis, https://pharmaverse.github.io/tidytlg/main/articles/freq.html#nested-frequency-analysis.
5. Tidytlg package vignette: Get Started, https://pharmaverse.github.io/tidytlg/main/articles/tidytlg.html.
6. Tidytlg package vignette: Column Metadata & How tidytlg Sets Up Your Data Using tlgsetup, https://pharmaverse.github.io/tidytlg/main/articles/tlgsetup.html.
7. Tidytlg package vignette: Manipulating tbl data frame, https://pharmaverse.github.io/tidytlg/main/articles/tbl_manipulation.html.
8. Wickham H, Miller E, Smith D (2022). haven: Import and Export 'SPSS', 'Stata' and 'SAS' Files. https://haven.tidyverse.org.

ACKNOWLEDGMENTS

We would like to acknowledge Nicholas Masel for your guidance, support from Sumesh Kalappurakal, and thank to Aidan Ceney, Nathan Kosiba, Steven Haesendonckx, Alexandra Papadopoulou for all your contributions to the tidytlg package.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Sheng-Wei Wang	Eli Miller
Janssen R&D	Atorus Research
Raritan, NJ 08869, USA	Mountain Lakes, NJ 07046
Email: swang69@its.jnj.com	Email: Eli.Miller@atorusresearch.com