Paper PP12

Summary Outputs Made Simple: A Mirror View for SAS Programmers Exploring R

Venkata Koteswara Rao Rayala, Merck Specialities Private Limited, Hyderabad, India

ABSTRACT

SAS is widely trusted for creating summary outputs in clinical research, supporting regulatory submissions with precision, efficiency, and reliability. As R becomes increasingly common in the pharmaceutical industry, SAS programmers are encouraged to understand how similar outputs can be generated using R. This poster presents a mirror image view of how summary outputs can be produced in R through simple, practical approaches that are easy to implement. Designed for SAS programmers with no prior R experience, the poster provides clear, easy-to-follow tips that highlight the similarities in logic, structure, and syntax between SAS and R. Real-world examples illustrate how to create consistent, accurate outputs across both tools without a steep learning curve. The goal is to help programmers confidently apply their existing knowledge to R, enabling seamless collaboration in dual-programming environments while ensuring regulatory-quality deliverables that meet industry standards.

INTRODUCTION

The preparation of clear, consistent, and reproducible summary outputs is a critical step in clinical research and regulatory submissions. Traditionally, SAS has been the dominant tool for generating these outputs, valued for its precision, efficiency, and compliance-ready workflows. However, with the growing adoption of R across the pharmaceutical industry, there is increasing interest in leveraging R's flexibility and open-source ecosystem to produce equivalent deliverables that meet regulatory standards.

The rtables package in R provides a powerful framework for building complex, hierarchical tables that align with regulatory expectations. It enables programmers to create demographic, efficacy, and safety summaries through intuitive functions that closely parallel SAS logic. By combining layout design, customizable analysis functions, and direct export to submission-ready formats, rtables bridges exploratory analysis and regulatory-quality reporting. This paper demonstrates how rtables can be applied to generate summary outputs in clinical research, offering SAS programmers practical, easy-to-follow approaches that highlight similarities in structure and syntax while ensuring accuracy, traceability, and compliance.

BASICS

Purpose: rtables builds complex, hierarchical tables for clinical trial data.

- **Workflow**:

1. basic_table() → start the layout.
2. split_cols_by ("ARM") → split columns by treatment arms (e.g., placebo vs drug).
3. split_cols_by ("STRATA1") → split rows by strata (e.g., baseline strata).
4. analyze("AGE") → apply analysis (mean, median, etc.).
5. build_table (adsl) → render with your dataset.

```
Ex. analyze ("AGE", afun = median)
analyze ("AGE", afun = function(x) {
list (
Mean     = mean (x, na.rm = TRUE),
Median   = median (x, na.rm = TRUE),
SD       = sd (x, na.rm = TRUE),
Variance = var (x, na.rm = TRUE),
Minimum  = min(x, na.rm = TRUE),
Maximum = max (x, na.rm = TRUE),
```

```
Range    = diff (range (x, na.rm = TRUE)),
Q1       = quantile (x, 0.25, na.rm = TRUE),
Q3       = quantile (x, 0.75, na.rm = TRUE)
)
}
```

LAYOUT & TABULATION

- **Analysis functions**:
  1. analyze () → summarize one variable.
  2. analyze_colvars () → analyze across multiple columns.
  3. summarize_row_groups () → summarize grouped rows.
- **Modifiers**:
  1. add_colcounts () → adds N per arm.
  2. add_overall_col("TOTAL") → adds a total column.
  3. append_topleft () → adds descriptive labels.

Clinical context: Useful for **baseline characteristics** tables, where regulators expect totals and clear labelling.

```
tbl <- basic_table(show_colcounts = TRUE) |>   # <-- add_colcounts()
 split_cols_by("ARM") |>
 add_overall_col("TOTAL") |>              # <-- add_overall_col()
 split_rows_by("RACE") |>
 summarize_row_groups() |>
 analyze("AGE", var_labels = "Age (years)", afun = mean) |>
 analyze_colvars(c("AGE", "BMI"), afun = mean) |>
 append_topleft("Demographics & Baseline Characteristics") |>  # <-- append_topleft()
 build_table(adsl)
```

CUSTOMIZATION OPTIONS

1. **afun/cfun**: custom analysis functions (e.g., mean, median, %).
2. **var_labels**: descriptive labels (“Age (yrs)” instead of “AGE”).
3. **format**: control numeric precision (xx.xx).
4. **na_str**: define how missing values appear.
5. **inclNAs**: include/exclude missing records.
6. **indent_mod**: indent rows for hierarchy.
7. **section_div**: add dividers between groups.
8. **Clinical context**: Ensures **clarity and compliance** — regulators want consistent formatting and explicit handling of missing data.

```
tbl <- basic_table(show_colcounts = TRUE) |>        # add_colcounts(): show N in column headers
 split_cols_by("ARM") |>                    # split columns by treatment arm
 add_overall_col("TOTAL") |>                  # add_overall_col(): adds a total column
 split_rows_by("RACE", section_div = "-----") |>    # split rows by RACE, add divider between groups
 summarize_row_groups() |>                    # summarize_row_groups(): counts & percentages per race
 analyze("AGE",                          # analyze(): summarize one variable (AGE)
       var_labels = "Age (yrs)",             # var_labels: descriptive label instead of "AGE"
       afun = function(x) {                 # afun: custom analysis function
        list(
          Mean   = mean(x, na.rm = TRUE),       # mean with missing values removed
          Median = median(x, na.rm = TRUE),     # median
          SD     = sd(x, na.rm = TRUE)          # standard deviation
        )
       },
       format = "xx.xx",                  # format: numeric precision (2 decimals)
       na_str = "Missing",                 # na_str: how missing values appear
       inclNAs = FALSE,                    # inclNAs: exclude missing records
       indent_mod = 2                      # indent_mod: indent rows for hierarchy
```

```
) |>
analyze_colvars(c("AGE", "BMI"),              # analyze_colvars(): analyze multiple variables side by side
          afun = mean,                  # mean for AGE and BMI
          var_labels = c("Age (yrs)", "BMI (kg/m2)"), # custom labels
          format = "xx.xx") |>          # format: 2 decimals
append_topleft("Demographics & Baseline Characteristics") |>   # append_topleft(): label in top-left corner
build_table(adsl)                       # build_table(): render table with ADSL dataset
```

SIMPLE TABULATION

The qtable() function in the rtables package provides a fast and convenient way to generate exploratory summaries of clinical trial data. Designed for efficiency, it automatically produces descriptive statistics for numeric variables (mean, median, standard deviation, minimum, maximum, and counts) and frequency distributions for categorical variables

```
# Quick summary of one numeric variable
qtable(adsl$AGE)              # Summarizes AGE → mean, median, SD, min, max

# Quick summary of multiple numeric variables
qtable(adsl$AGE, adsl$BMI)        # Summarizes AGE and BMI side by side with default stats

# Quick summary of a categorical variable
qtable(adsl$SEX)              # Summarizes SEX → counts and percentages per category

# Quick summary grouped by treatment arm
qtable(adsl$AGE, adsl$ARM)        # Summarizes AGE within each ARM (e.g., ARM X vs ARM Y)
```

	Placebo	Treatment A	Treatment B
Mean	50.2	49.8	51.1
SD	17.5	18.2	16.9
Min	21	20	22
Median	50	49	52
Max	80	79	78
N	34	33	33

TITLES & FOOTERS

- Functions:
  1. main_title () → table title.
  2. subtitles() → add context.
  3. main_footer() → footnotes.
  4. prov_footer() → provenance (data source).
  5. fnotes_at_path() → targeted footnotes for specific cells.

Clinical context: Regulatory tables must have **titles, footnotes, and provenance** for traceability.

```
library(rtables)

tbl <- basic_table(show_colcounts = TRUE) |>          # add_colcounts(): show N in column headers
 split_cols_by("ARM") |>                  # split columns by treatment arm
 add_overall_col("TOTAL") |>                 # add_overall_col(): adds a total column
 split_rows_by("RACE") |>                 # split rows by RACE
 summarize_row_groups() |>                    # summarize_row_groups(): counts & percentages per race
 analyze("AGE", var_labels = "Age (yrs)", afun = mean) |>  # analyze(): summarize AGE with mean
 append_topleft("Demographics") |>               # append_topleft(): label in top-left corner
 main_title("Table 5.1: Demographics Summary") |>      # main_title(): main table title
 subtitles("Safety Population (N=82)") |>          # subtitles(): add context under title
 main_footer("Note: SD = Standard Deviation") |>       # main_footer(): general footnote
 prov_footer("Source: ADSL dataset, generated on 20-Jan-2026") |> # prov_footer(): provenance (data source)
 build_table(adsl)                        # build_table(): render table with ADSL dataset
```

```
# Add targeted footnote for a specific cell
fnotes_at_path(tbl,
        rowpath = c("RACE", "Asian"),          # row path: Race → Asian
        colpath = "ARM X") <- "Includes subjects from India and China" # targeted footnote
```

SPLIT FUNCTIONS

1. **Row splits**: split_rows_by(), split_rows_by_multivar(), split_rows_by_cuts(), etc.
2. **Column splits**: split_cols_by(), split_cols_by_multivar(), etc.
3. **Custom split functions**: remove, reorder, drop levels.

Clinical context: For example, split by **biomarker levels** or **quartiles of lab values**.

split_rows_by()	Split rows by a single variable
split_rows_by_multivar()	Split rows by multiple variables side-by-side
split_rows_by_cuts()	Split rows by numeric ranges
split_rows_by_cutfun()	Split rows using a custom cut function
split_rows_by_quartiles()	Split rows into quartiles

split_cols_by()	Split columns by a single variable
split_cols_by_multivar()	Split columns by multiple variables
split_cols_by_cuts()	Split columns by numeric ranges
split_cols_by_cutfun()	Split columns using a custom cut function
split_cols_by_quartiles()	Split columns into quartiles

Split Modifiers

remove_split_levels()	Remove specific levels from split
keep_split_levels()	Keep only specified levels
drop_split_levels()	Drop levels but retain metadata
drop_and_remove_levels()	Drop and fully remove levels
add_overall_level()	Add an “Overall” summary row
add_combo_levels()	Combine multiple levels into one
reorder_split_levels()	Reorder levels manually
trim_levels_in_group()	Remove unused levels from a group
trim_levels_to_map()	Trim levels based on mapping

```
tbl <- basic_table(show_colcounts = TRUE) |>          # show N in column headers
  split_cols_by("ARM") |>                    # column split: treatment arms (ARM X, ARM Y)
  add_overall_col("TOTAL") |>                  # add overall total column
  split_rows_by("RACE") |>                   # row split: by RACE categories
  summarize_row_groups() |>                    # summarize counts & percentages for RACE
  split_rows_by_multivar(c("AGE", "BMI")) |>          # row split: multiple variables side by side
  analyze_colvars(c("AGE", "BMI"), afun = mean) |>      # analyze AGE and BMI with mean
  split_rows_by_cuts("AGE", cuts = c(20, 30, 40, 50)) |>   # row split: AGE grouped into ranges (20–29, 30–39, etc.)
  summarize_row_groups() |>                    # summarize counts & percentages for AGE ranges
```

```
 split_rows_by("RACE", remove_levels = "Unknown") |>       # custom split: remove "Unknown" category
 split_rows_by("RACE", reorder_levels = c("White", "Asian")) |> # custom split: reorder categories (White first)
 split_rows_by("RACE", drop_levels = list("Asian+Other" = c("Asian", "Other"))) |> # custom split: collapse Asian+Other
 build_table(adsl)                                  # build table with ADSL dataset
```

SORTING & PRUNING

1. **Sorting**: sort_at_path() → reorder rows by counts or criteria.
2. **Pruning**: prune_table() → remove rows with all zeros, NA, or low observations.

Clinical context: Ensures **clean tables** — regulators don't want cluttered outputs with empty rows.

cont_n_allcols()	Sort by count across all columns
cont_n_oncecol()	Sort by count in at least one column

prune_empty_level()	Remove empty levels
prune_zeros_only()	Remove rows with only zeros
content_all_zeros_na()	Remove rows with all zeros or NAs
low_obs_pruner()	Remove rows with low counts

```
library(rtables)

tbl <- basic_table(show_colcounts = TRUE) |>              # show N in column headers
 split_cols_by("ARM") |>                          # column split: treatment arms (ARM X, ARM Y)
 add_overall_col("TOTAL") |>                        # add overall total column
 split_rows_by("RACE") |>                         # row split: by RACE categories
 summarize_row_groups() |>                          # summarize counts & percentages for RACE
 sort_at_path(rowpath = "RACE", by = "count", decreasing = TRUE) |> # sorting: reorder rows by count (largest first)
 split_rows_by("AESEV") |>                        # row split: by Adverse Event Severity
 summarize_row_groups() |>                          # summarize counts & percentages for AE severity
 prune_table() |>                                 # pruning: remove rows with all 0s or NA
 build_table(adsl)                                # build table with ADSL dataset
```

ACCESS & MODIFY

- **Accessors**:
  1. head(), tail() → preview.
  2. cell_values(), value_at() → extract values.
- **Modifiers**:
  1. tbl[x,y] <- rcell(...) → replace cell.
  2. rbind(), cbind_rtables() → combine tables.

Clinical context: Useful for **post-processing** or combining outputs across multiple analyses.

```
# Build a simple demographics table
tbl <- basic_table(show_colcounts = TRUE) |>              # show N in column headers
 split_cols_by("ARM") |>                          # split columns by treatment arm
 add_overall_col("TOTAL") |>                        # add overall total column
 split_rows_by("RACE") |>                         # row split: by RACE categories
 summarize_row_groups() |>                          # summarize counts & percentages
 build_table(adsl)                                # build table with ADSL dataset

# --- Accessors ---
head(tbl)                                   # preview first rows of the table
tail(tbl)                                  # preview last rows of the table

head(tbl, n = 3) # preview the first 3 rows
```

```
tail(tbl, n = 2) # preview the last 2 rows


cell_values(tbl, rowpath = c("RACE", "Asian"), colpath = "ARM X") # extract all values for Asian in ARM X
value_at(tbl, rowpath = c("RACE", "Asian"), colpath = "ARM X")    # extract single value (e.g., count)

# --- Modifiers ---
tbl[2, 2] <- rcell(99, format = "xx")                  # replace cell at row 2, col 2 with value 99

tbl1 <- basic_table() |> split_rows_by("RACE") |> summarize_row_groups() |> build_table(adsl)
tbl2 <- basic_table() |> split_rows_by("SEX")  |> summarize_row_groups() |> build_table(adsl)

rbind(tbl1, tbl2)                                  # combine tables vertically (stack rows)
cbind_rtables(tbl1, tbl2)                          # combine tables horizontally (side by side)
```

RENDERING

1. **Console output**: ASCII tables.
2. **Conversion**: as_html(), tt_to_flextable().
3. **Pagination**: paginate_table() for printable layouts.
4. **Export**: export_as_docx(), export_as_pdf(), export_as_rtf(), export_as_tsv().

Clinical context: Direct export to **submission-ready formats** (PDF, RTF, DOCX) is critical for regulatory filings.

```
library(rtables)

# Build a simple demographics table
tbl <- basic_table(show_colcounts = TRUE) |>          # show N in column headers
  split_cols_by("ARM") |>                             # split columns by treatment arm
  add_overall_col("TOTAL") |>                         # add overall total column
  split_rows_by("RACE") |>                            # row split: by RACE categories
  summarize_row_groups() |>                           # summarize counts & percentages
  build_table(adsl)                                   # build table with ADSL dataset

# --- Rendering Options ---

tbl                                                   # 1. Console output: ASCII table in R console

as_html(tbl)                                          # 2. Conversion: render table as HTML
tt_to_flextable(tbl)                                  # 2. Conversion: convert to flextable for Word/PowerPoint

paginate_table(tbl, page_size = 40)                   # 3. Pagination: split table into pages of 40 rows each

export_as_docx(tbl, file = "demographics.docx")       # 4. Export: save table as Word document
export_as_pdf(tbl, file = "demographics.pdf")         # 4. Export: save table as PDF
export_as_rtf(tbl, file = "demographics.rtf")         # 4. Export: save table as RTF (regulatory standard)
export_as_tsv(tbl, file = "demographics.tsv")         # 4. Export: save table as tab-separated values
```

ADDITIONAL FUNCTIONS

1. **add_overall_level()** → add overall summary row.
2. **add_combo_levels()** → combine levels.
3. **trim_levels_in_group()** → trim unused levels.

Clinical context: Helps tailor tables to **regulatory expectations** (e.g., overall, AE counts, combined strata).

```
library(rtables)

# Build a demographics table with additional functions
tbl <- basic_table(show_colcounts = TRUE) |>
  split_cols_by("ARM") |>                             # split columns by treatment arm
```

```
 add_overall_col("TOTAL") |>                    # add overall total column
 split_rows_by("RACE", add_overall_level = TRUE) |>    # 1. add_overall_level(): adds an "Overall" summary row
 split_rows_by("RACE",
          add_combo_levels = list("Asian+Other" = c("Asian", "Other"))) |> # 2. add_combo_levels(): combine Asian + Other into one row
 split_rows_by("RACE", trim_levels_in_group = TRUE) |>    # 3. trim_levels_in_group(): remove unused categories (e.g., Unknown if empty)
 summarize_row_groups() |>                    # summarize counts & percentages
 build_table(adsl)                    # build table with ADSL dataset
```

CONCLUSION

The **rtables framework** offers a comprehensive, reproducible, and regulator-aligned approach to clinical trial reporting by enabling the construction of complex hierarchical tables with customizable analyses, clear formatting, and explicit handling of missing data. Its workflow—from basic setup and exploratory tabulation to advanced customization, splitting, sorting, pruning, and rendering—ensures that outputs are both scientifically rigorous and submission-ready. By integrating features such as titles, footnotes, provenance, and flexible export options, rtables bridges exploratory analysis with regulatory compliance, empowering clinical programmers to deliver transparent, auditable, and well-structured tables that enhance the credibility of trial evidence and support reliable scientific communication.

REFERENCES:

1. TLGs
2. https://insightsengineering.github.io/tlg-catalog/
3. https://pharmasug.org/proceedings/2025/OS/PharmaSUG-2025-OS-128.pdf?utm_source=chatgpt.com
4. Demographic Table – pharmaverse examples
5. https://cran.r-project.org/web/packages/pharmaRTF/index.html
6. CRAN: Package admiral

CONTACT INFORMATION

Author Name: Venkata Koteswara Rao Rayala

Company: Merck Specialities Private Limited

Address: Hyderabad

Work Phone:

Email: venkata-koteswara-rao.rayala@merckgroup.com

Website: