

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

Venkata Koteswara Rao Rayala

Merck Specialities Private Limited

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.

Basics:

The rtables R package is designed to create and display complex tables with R. Every rtable layout is constructed starting with basic_table and is rendered using build_table.

- basic_table()** → start the layout.
- split_cols_by ("ARM")** → split columns by treatment arms (e.g., placebo vs drug).
- split_rows_by ("STRATA1")** → split rows by strata (e.g., baseline strata).
- analyze("AGE")** → apply analysis (mean, median, etc.).
- 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)  
  )  
})
```

```
tbl_a <- basic_table() |>  
  split_cols_by("ARM") |>  
  split_rows_by("STRATA1") |>  
  analyze("AGE") |>  
  build_table(adsl)
```

TABLE OUTPUT

	ARM X	ARM Y

A		
Mean	33.32	35.86
B		
Mean	33.65	38.00

Split Functions

Split functions are used to add, remove, or transform the levels of the variable used in a split.

Row splits

```
split_rows_by()  
split_rows_by_multivar()  
split_rows_by_cuts()  
split_rows_by_cutfun()  
split_rows_by_quartiles()
```

```
remove_split_levels()  
keep_split_levels()  
drop_split_levels()  
drop_and_remove_levels()
```

Column splits

```
split_cols_by()  
split_cols_by_multivar()  
split_cols_by_cuts()  
split_cols_by_cutfun()  
split_cols_by_quartiles()
```

```
basic_table() |>  
  split_cols_by(  
    "ARM",  
    split_fun = remove_split_levels(c("ARM Y"))  
  ) |>  
  split_rows_by(  
    "STRATA1",  
    split_fun = reorder_split_levels(c("B", "A"))  
  ) |>  
  analyze("AGE") |>  
  build_table(adsl)
```

Table output

	ARM X		ARM Y

B	Mean	33.65	
A	Mean	33.32	

Sorting & Pruning Sorting functions are used to reorder table rows according to a given criteria function. Pruning functions are used to remove table rows according to a given criteria.

SORT FUNCTION

sort_at_path() → reorder rows by counts or criteria.

```
cont_n_allcols()
```

Sort by count across all columns

```
cont_n_oncecol()
```

Sort by count in at least one column

PRUNE FUNCTION

prune_table() → remove rows with all zeros, NA, or low observations.

```
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

```
tbl <- basic_table() |>  
  split_cols_by("ARM") |>  
  split_rows_by("STRATA2") |>  
  summarize_row_groups() |>  
  build_table(adsl)
```

```
tbl |>  
  sort_at_path(  
    "STRATA2",  
    scorefun = cont_n_allcols  
  ) |>  
  prune_table()
```

	ARM X	ARM Y

X	19 (45.2%)	19 (47.5%)
Y	23 (54.8%)	21 (52.5%)
Z	0 (0.0%)	0 (0.0%)

	ARM X	ARM Y

Y	23 (54.8%)	21 (52.5%)
X	19 (45.2%)	19 (47.5%)

ACCESS & MODIFY

Accessors:

head(), tail() → preview.

cell_values(), value_at() → extract values.

Modifiers:

tbl[x,y] <- rcell(...) → replace cell.

rbind(), cbind_rtables() → combine tables.

--- 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)

EXPORT



tbl

Console output: ASCII table in R console

as_html(tbl)

Conversion: render table as HTML

tt_to_flextable(tbl)

Conversion: convert to flextable for Word/PowerPoint

export_as_docx(tbl, "tbl.docx")

export_as_pdf(tbl, "tbl.pdf")

export_as_rtf(tbl, "tbl.rtf")

export_as_tsv(tbl, "tbl.tsv")

export_as_txt(tbl, "tbl.txt")

paginate_table(tbl, page_size = 40)

Pagination: split table into pages of 40 rows each

export_as_docx(tbl, file = "demographics.docx")

Export: save table as Word document

export_as_pdf(tbl, file = "demographics.pdf")

Export: save table as PDF

export_as_rtf(tbl, file = "demographics.rtf")

Export: save table as RTF (regulatory standard)

export_as_tsv(tbl, file = "demographics.tsv")

Export: save table as tab-separated values

CONCLUSION

Future of Clinical Programming

SAS: trusted, stable, validated for regulatory submissions.

R: 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.

Together: dual use ensures accuracy, reproducibility, and compliance while blending innovation with reliability.

REFERENCES

1. [TLGs](#)

2. <https://insightsengineering.github.io/tlg-catalog/>

3. https://pharmasug.org/proceedings/2025/QS/PharmaSUG-2025-QS-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](#)

Layout & Tabulation

Analyze & summarize Functions

analyze () → summarize one variable.

analyze_colvars () → analyze across multiple columns.

summarize_row_groups () → summarize grouped rows.

Layout modifiers

add_colcounts () → adds N per arm.

add_overall_col("TOTAL") → adds a total column.

append_topleft () → adds descriptive labels.

Customized table code

```
basic_table(show_colcounts = TRUE) |>  
  split_cols_by("ARM") |>  
  add_overall_col("TOTAL") |>  
  split_rows_by("BMRKR2",  
    split_label = "Biomarker 2 Level",  
    label_pos = "topleft") |>  
  summarize_row_groups() |>  
  analyze("AGE", var_labels = "Age (yrs)",  
    afun = mean, format = "xx.x") |>  
  analyze("STRATA1", var_labels = "Stratif.  
Term",  
    afun = function(x, _N_col) lapply(  
      table(x),  
      function(xi) rcell(  
        xi * c(1, 1 / _N_col),  
        format = "xx (xx.xx%)"  
      ))) |>  
  append_topleft(" Attribute") |>  
  build_table(adsl)
```

Customized table output

Biomarker 2 Level Attribute	ARM X (N=42)	ARM Y (N=40)	TOTAL (N=82)

LOW	22 (52.4%)	25 (62.5%)	47 (57.3%)
Age (yrs) mean	33.5	36.4	35.1
Stratif. Term			
A	9 (21.43%)	12 (30.00%)	21 (25.61%)
B	13 (30.95%)	13 (32.50%)	26 (31.71%)
HIGH	20 (47.6%)	15 (37.5%)	35 (42.7%)
Age (yrs) mean	33.5	37.7	35.3
Stratif. Term			
A	10 (23.81%)	9 (22.50%)	19 (23.17%)
B	10 (23.81%)	6 (15.00%)	16 (19.51%)

CUSTOMIZATION OPTIONS

afun/cfun: custom analysis functions (e.g., mean, median, %).

format: control numeric precision (xx.xx).

inclNAs: include/exclude missing records.

section_div: add dividers between groups.

var_labels: descriptive labels ("Age (yrs)" instead of "AGE").

na_str: define how missing values appear.

indent_mod: indent rows for hierarchy.

Titles & Footers

CODE

```
main_title(tbl_a) <- "My Title"  
subtitles(tbl_a) <- c("A subtitle")  
main_footer(tbl_a) <- c("A footnote")  
prov_footer(tbl_a) <- c("A provenance footer")  
fnotes_at_path(tbl_a,  
  rowpath = c("STRATA1", "A", "AGE", "Mean"),  
  colpath = c("ARM", "ARM X")  
) <- "Mean age for arm X"
```

Table output

My Title

A subtitle

ARM X ARM Y

A Mean 33.32 (1) 35.86

B Mean 33.65 38.00

(1) - Mean age for arm X

A footnote

A provenance footer