SESUG Paper 15-2026

Hands-On Reporting in R with the *reporter* Package

David J. Bosak, Archytas Clinical Solutions

ABSTRACT

The **reporter** R package has recently seen a surge of interest by companies looking for a PROC REPORT and ODS type reporting system for R. This package fits the bill. The **reporter** package can create almost any kind of report: demographics, adverse events, patient profiles, figures, listings, survival analysis, etc. The package supports multiple levels of spanning headers, page bys, nested stubs, page breaking, page wrapping, and much more. And it can output the results in five different file formats: TXT, HTML, DOCX, RTF, and PDF. The **reporter** package is by far the most comprehensive reporting system in R. This hands-on workshop will take you from the basics to some of the most advanced features of the package. By the end of the workshop, you will be able to create reports in R with ease and confidence. The workshop is being presented by David J. Bosak, the creator of the package.

INTRODUCTION

Many organizations are engaged in pilot studies to reproduce SAS reports in R. The **reporter** R package can help you with this task. The **reporter** package was designed to give you similar capabilities as PROC REPORT and ODS. This package will allow you to reproduce SAS reports with a high degree of fidelity. The package allows you to:

- Generate tables, figures, and listings
- Write in RTF, PDF, TXT, DOCX and HTML file formats
- Has many options for placement and justification of titles and footnotes
- Supports page break, page wrap, and page by functionality
- Allows mixing of figures, text, and tables on the same page

The purpose of this paper is to introduce you to the most basic features of the package. Full documentation is available at https://reporter.r-sassy.org/index.html.

SAMPLE DATA

The following sample data will be used for most of the examples below:

```
# Create sample data
dat <- read.table(header = TRUE, text = '
VAR      LABEL            A             B
Age          N            9            10
Age    MEANSTD  "13.3 (1.00)" "13.3 (1.89)"
Age     MEDIAN           13            13
Age       Q1Q3    13.0-14.0     12.0-15.0
Age     MINMAX        12-15         11-16
Height       N            9            10
Height MEANSTD  "62.1 (3.88)" "62.5 (6.26)"
Height  MEDIAN        62.50         64.55
Height    Q1Q3    59.8-63.5     57.5-66.5
Height  MINMAX    56.5-69.0     51.3-72.0
Weight       N            9            10
Weight MEANSTD "95.9 (12.36)" "103.7 (29.49)"
Weight  MEDIAN        98.00        105.75
Weight    Q1Q3   84.0-102.5    85.0-128.0
Weight  MINMAX   83.0-112.5    50.5-150.0
Sex          N            9            10
```

```
Sex           F     "5 (55.6%)"   "4 (40.0%)"
Sex           M     "4 (44.4%)"   "6 (60.0%)"')
```

SIMPLE REPORT

Creating a report with the **reporter** package involves three steps:

- Create report content (a table, text, or plot)
- Create a report object and attach the content
- Write out the report in the desired output format

Here is an example that shows how to create a minimal report using the sample data from above:

```
library(reporter)

# Create table
tbl <- create_table(dat) |>
  titles("Listing 1.0", "Basic Listing") |>
  footnotes("* SESUG, 2026")

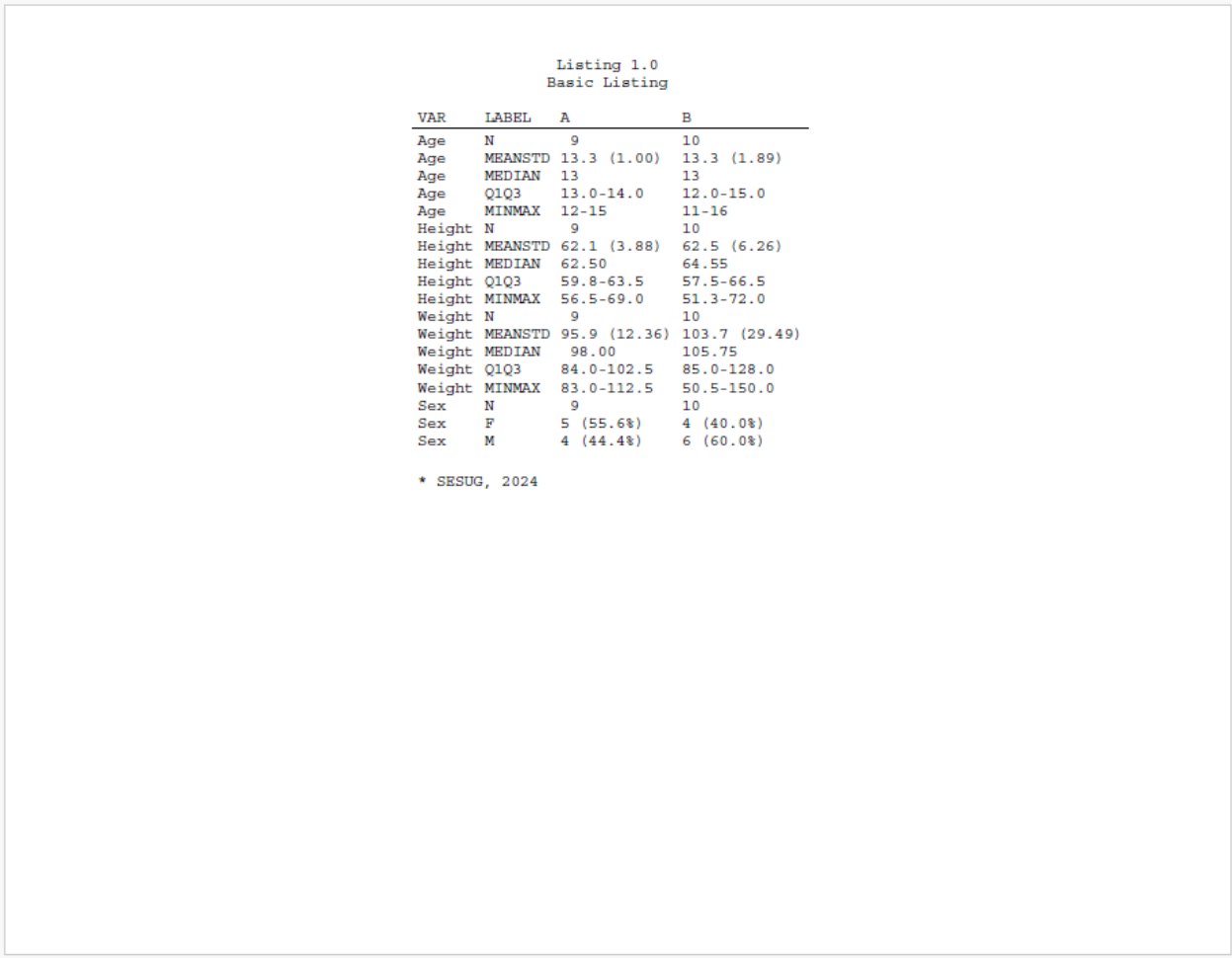
# Create the report
rpt <- create_report("./report/example1",
                     font = "Courier",
                     output_type = "PDF") |>
  add_content(tbl)

# Write the report
write_report(rpt)
```

In the above example, notice the following:

- The input data is passed to the `create_table()` function. Any calculations should be done prior to reporting. The reporting functions will not perform any calculations.
- The table object `tbl` is then added to the report object `rpt` via the `add_content()` function. You can add one or more pieces of content to a report.
- You can change to a different file format by changing the value passed to the *output_type* parameter of `create_report()`.
- The `write_report()` function renders the report in the desired output type and writes it to the file system. The file format can also be specified on the *output_type* parameter of `write_report()`.

The above code will produce the following PDF report. Observe that the columns and column widths are all determined automatically based on the data. Also, the orientation, font size, and margins all have sensible defaults.



Listing 1.0
Basic Listing

VAR	LABEL	A	B
Age	N	9	10
Age	MEANSTD	13.3 (1.00)	13.3 (1.89)
Age	MEDIAN	13	13
Age	Q1Q3	13.0-14.0	12.0-15.0
Age	MINMAX	12-15	11-16
Height	N	9	10
Height	MEANSTD	62.1 (3.88)	62.5 (6.26)
Height	MEDIAN	62.50	64.55
Height	Q1Q3	59.8-63.5	57.5-66.5
Height	MINMAX	56.5-69.0	51.3-72.0
Weight	N	9	10
Weight	MEANSTD	95.9 (12.36)	103.7 (29.49)
Weight	MEDIAN	98.00	105.75
Weight	Q1Q3	84.0-102.5	85.0-128.0
Weight	MINMAX	83.0-112.5	50.5-150.0
Sex	N	9	10
Sex	F	5 (55.6%)	4 (40.0%)
Sex	M	4 (44.4%)	6 (60.0%)

* SESUG, 2024

Figure 1: Simple Report Example

PAGE HEADERS AND FOOTERS

Now let's add page headers and footers to the report. Page headers are specified using the function `page_header()`. Page footers are added using the function `page_footer()`. The objects produced by these functions are attached to the report object. Normally, you will add these functions to the `create_report()` pipeline, as follows:

```
# Create table
tbl <- create_table(dat) |>
  titles("Listing 1.0", "Basic Listing")

# Create the report
rpt <- create_report("./report/example2", output_type = "PDF",
                     font = "Arial") |>
  page_header("Sponsor: My Company", "Study: Students") |>
  add_content(tbl) |>
  footnotes("* SESUG, 2026", footer = TRUE) |>
  page_footer(Sys.time(), "Confidential", "Page [pg] of [tpg]")

# Write the report
write_report(rpt)
```

Notice the following in the above example:

- The page header is split into two columns, left and right, with two corresponding parameters. To add multiple lines to the page header, pass the desired lines as a vector of strings.
- The page footer is split into three columns: *left*, *center*, and *right*.
- There are special tokens to generate page numbers. The token `[pg]` generates the page number, while the token `[tpg]` is the total number of pages. These tokens can be placed anywhere in the page header, page footer, or footnotes.
- The font can be changed using the *font* parameter of `create_report()`. Available fonts are "Courier", "Arial", and "Times".

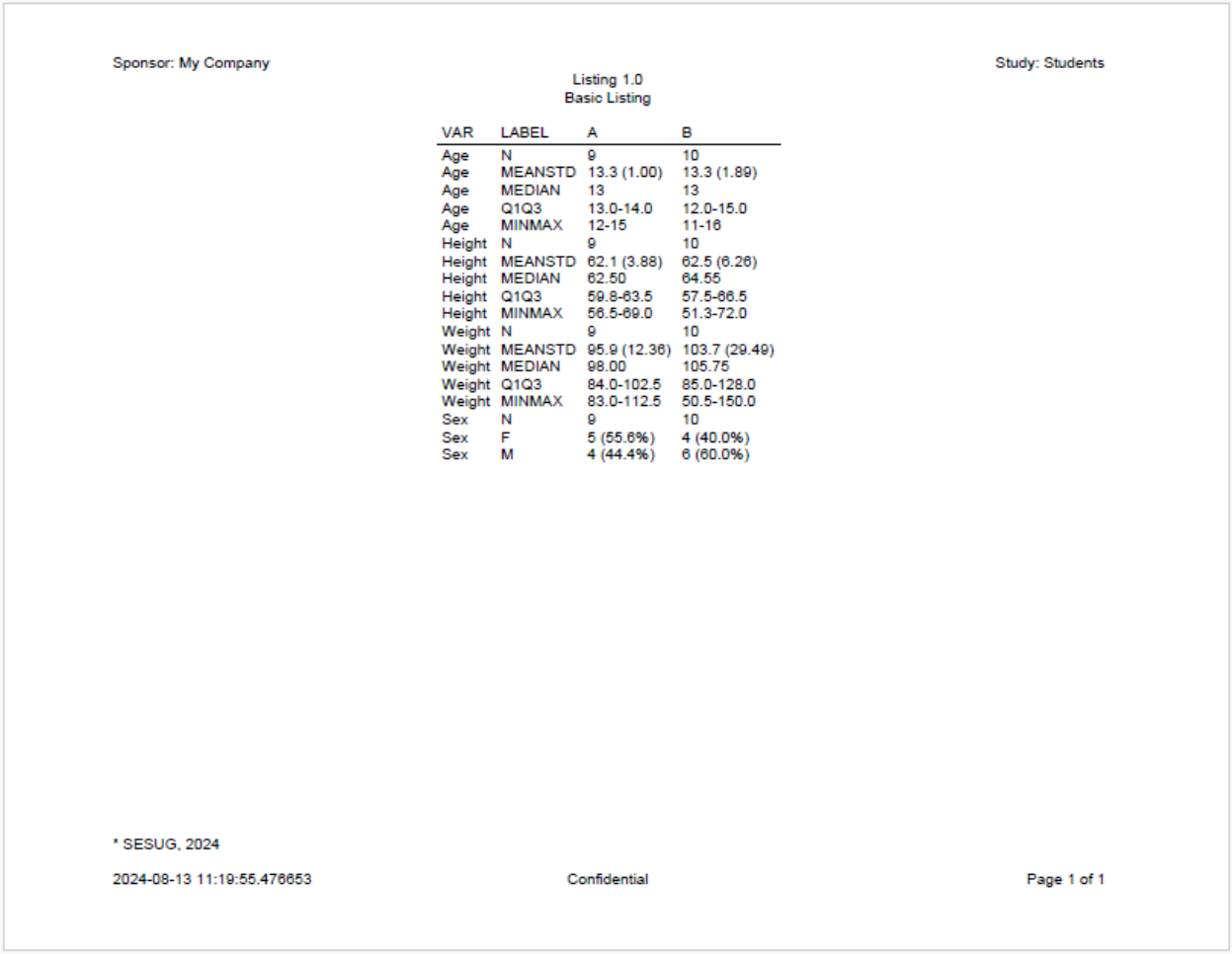


Figure 2: Page Header and Footer

TITLE AND FOOTNOTE LOCATION

In the above example, also notice that the footnotes have been attached to the report object. Titles and footnotes can be attached either to the table object or the report object. When the titles or footnotes are attached to the report, they will be the same for the entire report. When the titles or footnotes are attached to the table, they will only persist for that table. This feature is useful when you have multiple pieces of content in the report, and wish to have different titles and footnotes for each piece of content.

Additionally, the titles and footnotes have different features available depending on which object they are attached to. Attaching to the report allows you to place the titles or footnotes in the document header or footer. The above example shows you how to place the footnote in the document footer. Some organizations prefer this style of footnote.

One last comment regarding titles and footnotes is that you can have multiple title and footnote functions. For instance, if you wanted one footnote directly under the table, and one footnote in the document footer, you can do it. You can do it by attaching one footnote function to the table object. Then add the second footnote function to the report object, setting the *footer* parameter to TRUE.

You may also have multiple title or footnote functions attached to the same object. Multiple title and footnotes functions provides you with considerable flexibility in how titles and footnotes are organized.

For more information about titles and footnotes, see the **reporter** documentation.

BORDERS

Many organizations have particular requirements regarding table borders. The **reporter** package allows you to recreate the most common table border configurations.

Border parameters are usually available on the object you want to apply borders to. For instance, the `create_table()` function has a *borders* parameter that accepts a vector of border keywords. Available border keywords are: top, bottom, left, right, all, and outside. Any border keywords passed to this parameter will generate the desired borders on the table object.

The title and footnote objects have their own *borders* parameter. These borders will apply to the corresponding object.

Let's start with a simple top and bottom border on our table:

```
# Create table
tbl <- create_table(dat, borders = c("top", "bottom"),
                    first_row_blank = TRUE) |>
  titles("Listing 1.0", "Table with Borders", bold = TRUE)

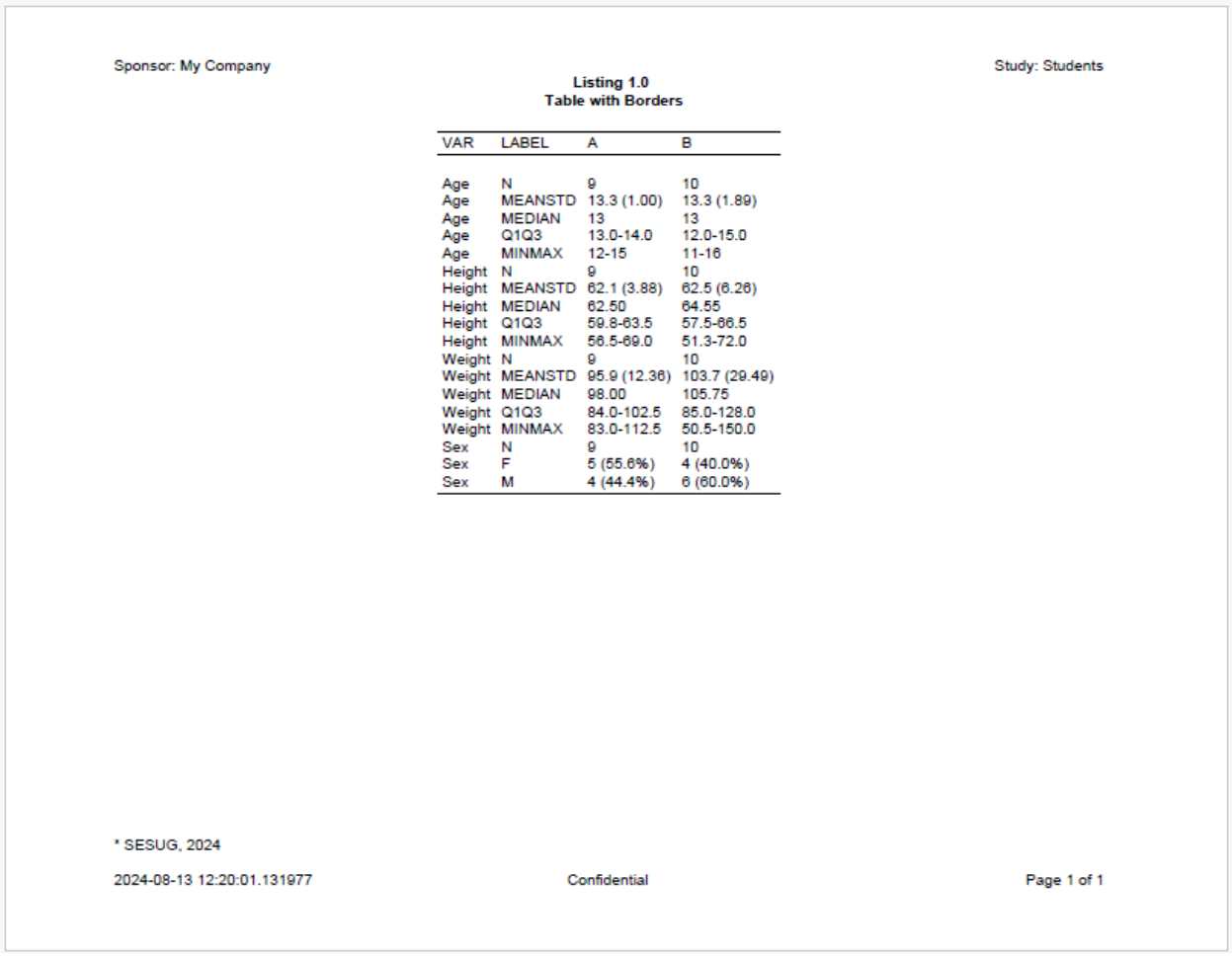
# Create the report
rpt <- create_report("./report/example3", output_type = "PDF",
                     font = "Arial") |>
  page_header("Sponsor: My Company", "Study: Students") |>
  add_content(tbl)  |>
  footnotes("* SESUG, 2026", footer = TRUE, blank_row = "none") |>
  page_footer(Sys.time(), "Confidential", "Page [pg] of [tpg]")

# Write the report
write_report(rpt)
```

Note the following in the above example:

- Border keywords are passed to the *borders* parameter as a vector.
- The *first_row_blank* parameter will add a blank row below the table header.
- Titles can be bolded using the *bold* parameter on the `titles()` function. The `titles()` function also has a *font_size* parameter to control the font size.

Here is the table with desired borders:



Sponsor: My Company

Study: Students

Listing 1.0
Table with Borders

VAR	LABEL	A	B
Age	N	9	10
Age	MEANSTD	13.3 (1.00)	13.3 (1.89)
Age	MEDIAN	13	13
Age	Q1Q3	13.0-14.0	12.0-15.0
Age	MINMAX	12-15	11-16
Height	N	9	10
Height	MEANSTD	62.1 (3.88)	62.5 (6.26)
Height	MEDIAN	62.50	64.55
Height	Q1Q3	59.8-63.5	57.5-66.5
Height	MINMAX	56.5-69.0	51.3-72.0
Weight	N	9	10
Weight	MEANSTD	95.9 (12.36)	103.7 (29.49)
Weight	MEDIAN	98.00	105.75
Weight	Q1Q3	84.0-102.5	85.0-128.0
Weight	MINMAX	83.0-112.5	50.5-150.0
Sex	N	9	10
Sex	F	5 (55.6%)	4 (40.0%)
Sex	M	4 (44.4%)	6 (60.0%)

* SESUG, 2024

2024-08-13 12:20:01.131977

Confidential

Page 1 of 1

Figure 3: Table Borders

COLUMN DEFINITIONS

The above examples are still using the automatic column definitions. The **reporter** package permits you to override the automatic definitions and control them yourself. You can control the column definitions using the `define()` function.

The `define()` function is similar to the DEFINE statement in PROC REPORT. It allows you to add a label, set the column width, specify column alignment, and assign many other attributes of a column.

You can also set defaults that are different from the automatic defaults, using the `column_defaults()` function. This function allows you to set defaults for some or all columns on the table. Note that any value specified on the `define()` function will override that from `column_defaults()`.

The below example also shows how to reformat the row labels. Notice that on the above examples, the row labels (“Age”, “Height”, “Weight”, and “Sex”) are repeated on every row. These labels can be made nicer by eliminating the duplication, and mapping the variable name to a proper label.

To perform the mapping, we will create vector lookups. The lookups are then assigned to the *format* parameter of the `define()` function for the `VAR` and `LABEL` columns. The combination of these changes will produce nice row labels for our report.

Let’s take a look at the code:

```
# Vector lookup for VAR
varfmt <- c(Age = "Age (yrs)",
            Height = "Height (inches)",
            Weight = "Weight (lbs)",
            Sex = "Biological Sex")

# Vector lookup for LABEL
labelfmt <- c(N = "n", MEANSTD = "Mean (STD)", MEDIAN = "Median",
              Q1Q3 = "Q1-Q3", MINMAX = "Min-Max",
              "F" = "Female", M = "Male")

# Create table
tbl <- create_table(dat, borders = c("top", "bottom"),
                    first_row_blank = TRUE) |>
  titles("Table 1.0", "Column Definitions", bold = TRUE) |>
  column_defaults(width = 1, align = "center") |>
  define(VAR, label = "Variable", blank_after = TRUE, dedupe = TRUE,
         format = varfmt, width = 1.5, align = "left") |>
  define(LABEL, label = "", align = "left", format = labelfmt) |>
  define(A, label = "Group A", n = 9) |>
  define(B, label = "Group B", n = 10)

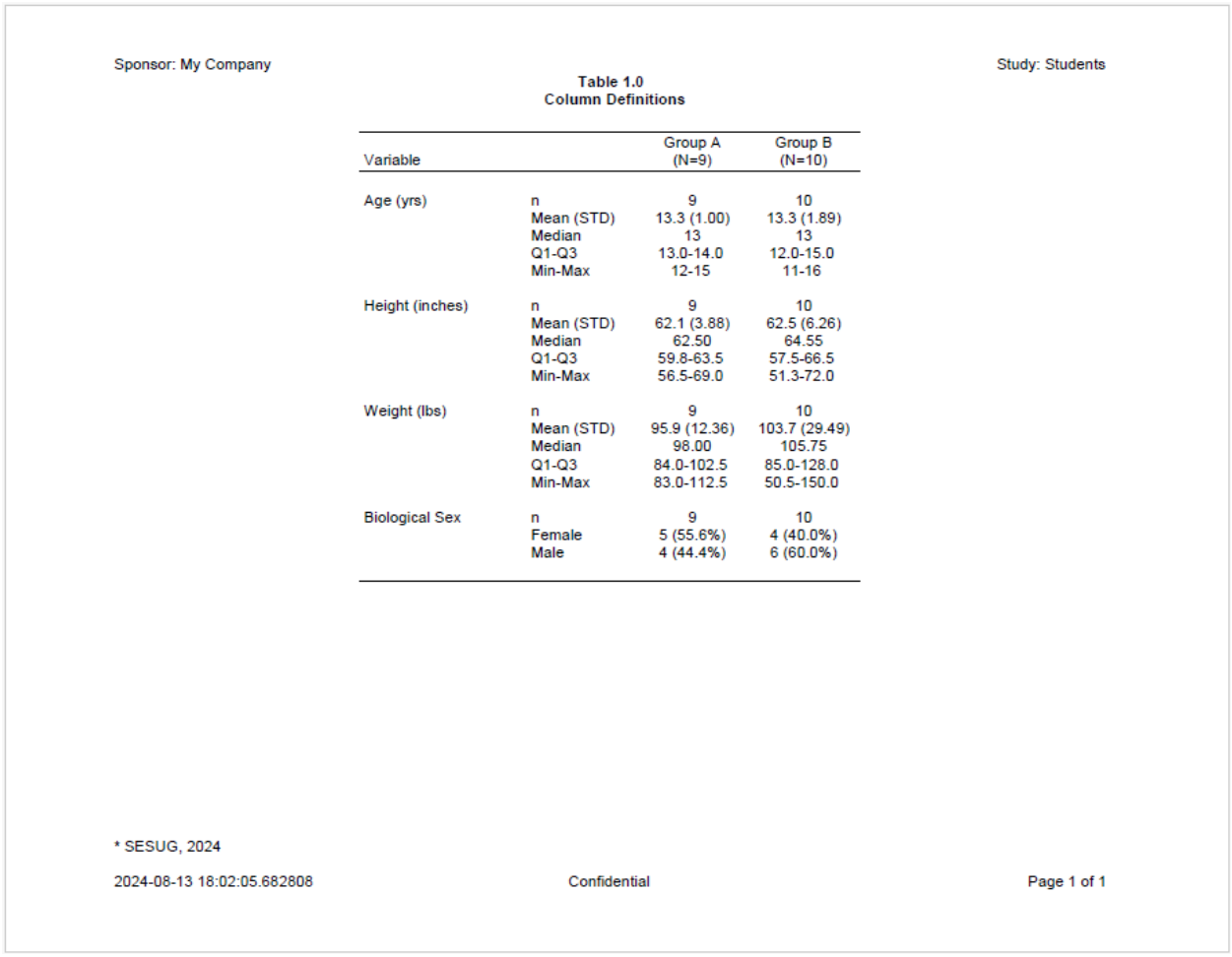

# Create report and add content
rpt <- create_report("./report/example4", output_type = "PDF",
                     font = "Arial") |>
  page_header(left = "Sponsor: My Company", right = "Study: Students") |>
  add_content(tbl) |>
  footnotes("* SESUG, 2026") |>
  page_footer(left = Sys.time(),
              center = "Confidential",
              right = "Page [pg] of [tpg]")

# Write out report
write_report(rpt)
```

Observe the following in the above code sample:

- Vector lookups are created for both the VAR and LABEL columns, and then assigned to the *format* parameter for the respective columns. You may also create a user-defined format for this mapping. See the **fmtr** package for additional information.
- The `column_defaults()` function set the defaults to 1 inch and “center” alignment for all columns.
- The `define()` function can be specified for any or all columns.
- The column definition for the VAR variable sets the *blank_after* parameter to TRUE. This value will create a blank row every time the VAR value changes.
- The VAR definition also set the *dedupe* parameter to TRUE. This parameter eliminates duplicate values from the VAR column.
- You can control formatting on a column by assigning a format to the *format* parameter.
- Header labels can be assigned with the *label* parameter.
- The “N = “ population counts can be assigned with the `n` parameter. This feature is built into the `define()` function.

Here is the generated report:



Sponsor: My Company

Study: Students

Table 1.0
Column Definitions

Variable		Group A (N=9)	Group B (N=10)
Age (yrs)	n	9	10
	Mean (STD)	13.3 (1.00)	13.3 (1.89)
	Median	13	13
	Q1-Q3	13.0-14.0	12.0-15.0
	Min-Max	12-15	11-16
Height (inches)	n	9	10
	Mean (STD)	62.1 (3.88)	62.5 (6.26)
	Median	62.50	64.55
	Q1-Q3	59.8-63.5	57.5-66.5
	Min-Max	56.5-69.0	51.3-72.0
Weight (lbs)	n	9	10
	Mean (STD)	95.9 (12.36)	103.7 (29.49)
	Median	98.00	105.75
	Q1-Q3	84.0-102.5	85.0-128.0
	Min-Max	83.0-112.5	50.5-150.0
Biological Sex	n	9	10
	Female	5 (55.6%)	4 (40.0%)
	Male	4 (44.4%)	6 (60.0%)

* SESUG, 2024

2024-08-13 18:02:05.682808

Confidential

Page 1 of 1

Figure 4: Column Definitions

Now our sample table is looking pretty good! Many more samples can be found on the **sassy** report gallery. You can use these examples to help you learn more about how the **reporter** package works, and understand its capabilities. The report gallery can be found here: https://r-sassy.org/gallery.html#gallery.

FIGURE

The last report we will examine is a figure. To create a figure, we will create a plot using the **ggplot2** package. Then add the plot to the report. But first, let's create some sample data.

SAMPLE DATA

Below is the sample data we will use for the figure:

```
# Create sample data
cls <- read.table(header = TRUE, text = '
Name    Sex Age Height Weight  Group
Alfred    M  14   69.0  112.5      A
Alice     F  13   56.5   84.0      A
Barbara   F  13   65.3   98.0      A
Carol     F  14   62.8  102.5      A
Henry     M  14   63.5  102.5      A
```

```
James    M  12   57.3   83.0      A
Jane     F  12   59.8   84.5      A
Janet    F  15   62.5  112.5      A
Jeffrey  M  13   62.5   84.0      A
John     M  12   59.0   99.5      B
Joyce    F  11   51.3   50.5      B
Judy     F  14   64.3   90.0      B
Louise   F  12   56.3   77.0      B
Mary     F  15   66.5  112.0      B
Philip   M  16   72.0  150.0      B
Robert   M  12   64.8  128.0      B
Ronald   M  15   67.0  133.0      B
Thomas   M  11   57.5   85.0      B
William  M  15   66.5  112.0      B')
```

CREATE PLOT

To create a figure, we first need to create a plot. To do this, we will use the popular **ggplot2** package. The **ggplot2** package has abundant capabilities, and many functions. Discussion of this package is beyond the scope of this paper. But, generally speaking, you will create the plot using standard **ggplot2** syntax, and store the resulting object in a variable. That variable can then be used to create a plot for the report.

Here is how to create a plot using **ggplot2**. The plot object is stored in the variable `p`:

```
library(ggplot2)

# Calculate statistics
agemn <- mean(cls$Age)
agesd <- sd(cls$Age)

# Create ggplot object
p <- ggplot(cls, aes(Age)) +
  geom_histogram(aes(y = after_stat(density)), bins = 6,
                 color="darkblue", fill="#CAD5E5",
                 show.legend = FALSE) +
  theme_light() +
  stat_function(fun = dnorm,
                args = list(mean = agemn,
                            sd = agesd),
                aes(col = "#1b98e0"),
                linewidth = 1) +
  geom_density(adjust = 1.75, linewidth = 1, aes(col = "red"),
               show.legend = FALSE) +
  geom_vline(aes(xintercept=agemn),
             color="darkgray", linetype="dashed", linewidth=1) +

  scale_color_manual("Legend", values = c("#1b98e0", "red"),
                     labels = c("normal", "kernel")) +
  theme(text = element_text(size = 7),
        panel.border = element_rect(colour = "black", fill=NA,
                                    linewidth=1))
```

ADD PLOT TO REPORT

In the **reporter** package, creating a figure is almost exactly the same as creating a table or listing. You will first create the report content, then add the content to the report:

```
# Create plot content
plt <- create_plot(p, height = 4, width = 7) |>
  titles("Figure 1.0", "Age Distribution",
         bold = TRUE) |>
  footnotes("* SESUG, 2026")

# Create report object
rpt <- create_report("./report/figure1", output_type = "PDF",
                     font = "Arial") |>
  page_header(left = "Sponsor: My Company", right = "Study: Students") |>
  add_content(plt) |>

  page_footer(left = Sys.time(),
              center = "Confidential",
              right = "Page [pg] of [tpg]")

# Write report to file
res <- write_report(rpt)
```

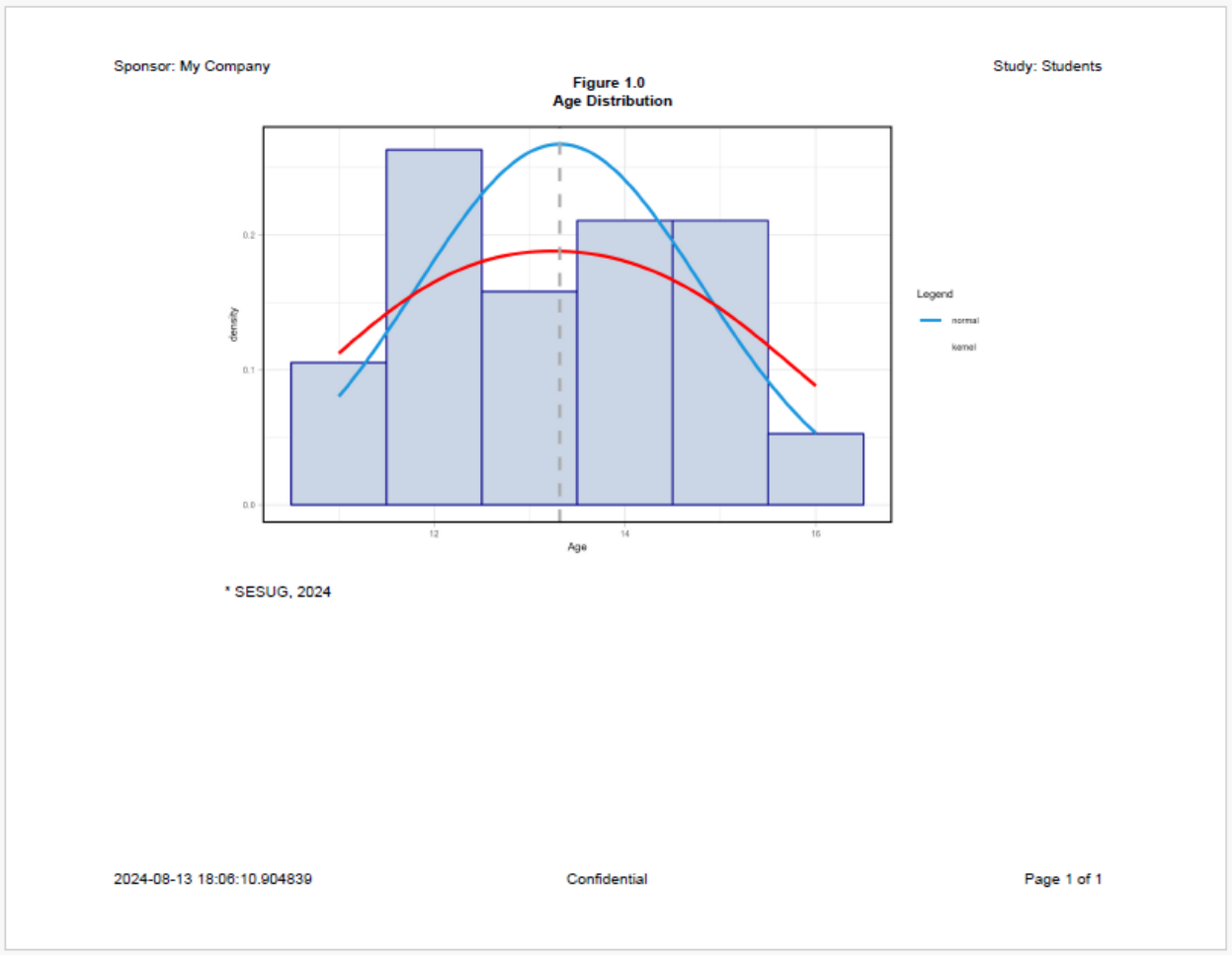


Figure 5: Report with Figure

Comments on above example:

- Note that the overall steps shown above are the same as the previous examples. Except this time, we used the `create_plot()` function instead of `create_table()`. The plot object `p` is passed on the first parameter of `create_plot()`. The `create_plot()` function also requires a *height* and *width* for the rendered plot.
- Everything else concerning the report is the same. You can add page headers and footers, titles and footnotes, etc. just like with the table.

CONCLUSION

The **reporter** package allows you to create with R almost any report that you can create with SAS. The **reporter** package was designed specifically to have similar capabilities and a similar conceptual framework. It has a variety of options to control table layout, titles and footnotes, and column definitions. It can create tables, listings, and figures. It also allows you to export into five different file formats. The package is easy to use, and produces results that are nearly identical to SAS. SAS programmers will find this package much more satisfying to use compared to the alternatives.

REFERENCES

Bosak D (2026). *Getting Started with Reporter*, Accessed October 12, 2026. https://reporter.r-sassy.org/articles/reporter.html

Bosak D (2023). *An Overview of the SASSY System*, WUSS Paper 185-2023, https://r-sassy.org/papers/WUSS_2023_Paper_185.pdf

ACKNOWLEDGMENTS

During development of the **reporter** package, several people have provided ideas and encouragement. I'd like to thank Kevin Kramer, Duong Tran, and Raphael Huang for their contributions.

CONTACT INFORMATION

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

David J. Bosak
Archytas Clinical Solutions
dbosak01@gmail.com
www.r-sassy.org

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.