Paper PP22

%CallR

Duong Tran, Elderbrook Solutions GmbH

ABSTRACT

The integration of software applications within workflows is both desirable and often essential, as it enables the combined strengths of these tools to increase productivity and gives better quality results. When it comes to leveraging R from within SAS, the licensed SAS/IML module is the commonly recognised solution. However, there exists a cost-free alternative. This technique was first introduced by Hollands[1] in the paper "SAS to R to SAS" and has since been developed and explored by other contributors[3]. This presentation seeks to expand on this body of work by providing additional insights and demonstrating practical applications through %CallR, a SAS macro designed to facilitate seamless interaction with R.

Key words: systask, x, %sysexec, BATCH, R.EXE

INTRODUCTION

Executing R codes from SAS with Proc IML to exchange data and returning the results to the SAS screen is well documented[4]. A less well-known free alternative[1] is running R codes in batch (non-interactive) mode with this "**R.exe CMD BATCH [options] infile [outfile]**" command and this is key to the %CallR implementation. Like SAS procedures, this method can return results in text and graphics to the SAS screen or save them to a file. Results can also be in Text, RTF, PDF and HTML formats. These possibilities enable SAS users to leverage R functionalities within their SAS workflow as if they are like any other SAS procedures.

METHODS

Running R Codes in Batch from SAS

SAS provides these **systask**, **x**, **%sysexec** commands as well as the **pipe** method to run any executable file, while R can run in batch mode. Together, these methods allow writing and executing R codes on the fly within SAS. Results from R can return to the screen or further processed by SAS.

```
data _null_; file "r_pgm.r"; <R statements>; put <R statements>; run;

1. filename proc_r pipe "R.exe CMD BATCH --vanilla -quiet r_pgm.r r_pgm.rlg";
   data _null_; infile proc_r; run;

2. systask command "R.exe CMD BATCH --vanilla --quiet r_pgm.r r_pgm.rlg" wait;

3. x "R.exe CMD BATCH --vanilla --quiet r_pgm.r r_pgm.rlg";

4. %sysexec R.exe CMD BATCH --vanilla --quiet r_pgm.r r_pgm.rlg;
```

Save and Sending R Graphics to the Screen

Again, this is possible, using SAS to display the saved R graphics after executing R codes as shown in this code snippet from Hollands[1]. An alternative is to use proc report instead of the put statement used in the Hollands example.

```
ods html file='I:\Phuse2025\report.html' style=minimal graph='I:\Phuse2025';

data _null_;
  file print;
  put "<img src='I:\Phuse2025\RRisk.png' border='0'>";
run;
```

```
ods escapechar='^';
ods rtf file='I:\Phuse2025\report.rtf' style=minimal;

data _null_;
  file print;
  put "^s={preimage='I:\Phuse2025\RRisk.png'}";
run;

ods _all_ close; ods listing;

data callR_image;
  image="RRisk.png";
run;

ods rtf path="I:\Phuse2025" file= "report.rtf" style=styles.monospace;

proc report data=callR_image nowindows style(report)=[outputwidth=100%];
  column image;
  define image / " " display style={just=center cellwidth=100%};
  compute image; call define(…);
    image = " ";
  endcomp;
run;
```

Generate R Codes with SAS on the Fly

SAS can be used to generate R codes dynamically on the fly, save them to a file, then source (execute) it by %callR. Alternatively, this can be done inside the %callR call and even using the SAS macro language is allowed, as shown in the example in 'SAS Enterprise Guide' section. This makes %callR highly flexible for interleaving the two languages.

```
data _null_;
  file "I:\Phuse2025\code.r";
  put "library (utils)";
  put "df <- read.table(header = TRUE, text = '";
  put %sysfunc(quote(package  description));
  put %sysfunc(quote("tidyverse:" "R Packages for Data Science"));
  put %sysfunc(quote("sassy:" "R Packages to Make R Easier for Everyone"));
  put %sysfunc(quote("pharmaverse:" "R packages for Clinical Trial Reporting"));
  put "')";
run;
%callR(library(haven);
  source("I:/Phuse2025/code.r"); df;
  rconsoleto = PGM,
  srconsoleto = rresult1.rlg
);
```

INTERFACE

Being able to execute **R.EXE** in batch mode is one possible way that SAS can interface with R, and there are several published versions[3] of this implementation already. Here is the %callR interface. Refer to the examples below for its usage.

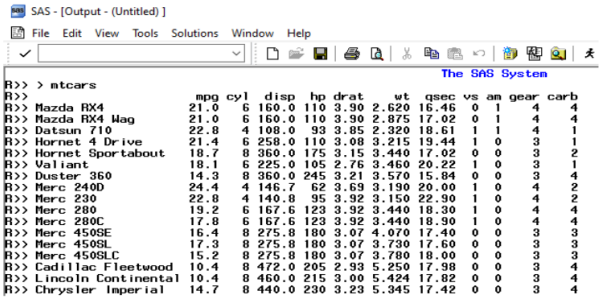
```
%callR(pgmx                    /* R codes                                          */
      ,rpath       =           /* R.exe path                                       */
      ,rvarname    =           /* a variable name to store a value returned from R */
      ,rconsoleto  = PGM       /* the R console goes to PGM | LOG                  */
      ,srconsoleto =           /* save the R console to this file                  */
      ,displayG    =           /* display the graph                                */
      ,displayGF   = rtf       /* display graph format RTF | PDF | HTML            */
      ,justG       = center    /* justify the displayed graph center|left|right    */
      ,df2sas7bdat =           /* export R data frame to SAS dataset               */
      ,temploc     =           /* temporary storage folder                         */
      ,savegto     =           /* save graph to this location                      */
      ,savegname   =           /* saved graph name                                 */
      )
```

SCREENS DISPLAY AND SAVED

Text Returns to Screen from R Console

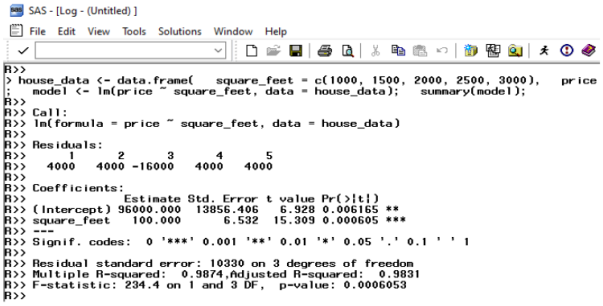
Interactivity with data is a core aspect of statistical analysis and data science, so the immediate results returned to the screen after code execution are absolutely necessary. Hence, %CallR was built with this important feature that captures text results from the R console and re-displays them to either the SAS LOG or OUTPUT screen. Two examples below demonstrate this. The text results shown on the screens can also be saved by SAS with **proc printto** as per usual.

%CallR(mtcars)




```
proc printto new file="I:\Phuse2025\mtcars.lst";
run;
%callR(mtcars, rconsoleto=out)
proc printto;
run;
```

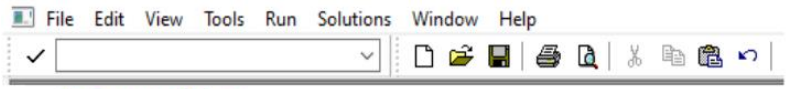
%CallR(house_data, ... , rconsoleto=log)



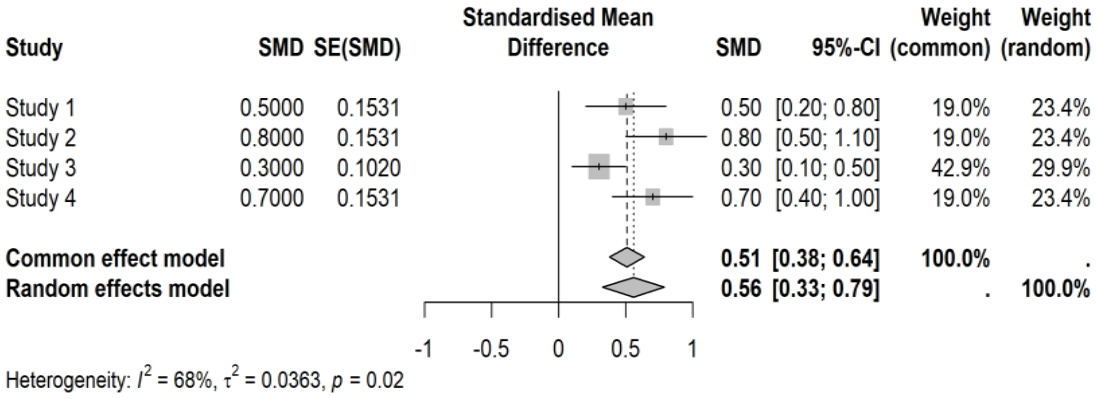

```
proc printto new log="I:\Phuse2025\linreg.log"; run;
%callR(house_data <- data.frame(
  square_feet = c(1000, 1500, 2000, 2500, 3000),
  price = c(200000, 250000, 280000, 350000, 400000));
  model <- lm(price ~ square_feet, data = house_data);
  summary(model);
 ,rconsoleto  = LOG);
proc printto; run;
```

Graphics from R

For R graphics, %CallR has the **displayF** option to automatically screen-display them in the chosen HTML, PDF and RTF formats, provided that a Web browser, Adobe and Microsoft Word are installed. There is also the **saveto** option to save them to a file.

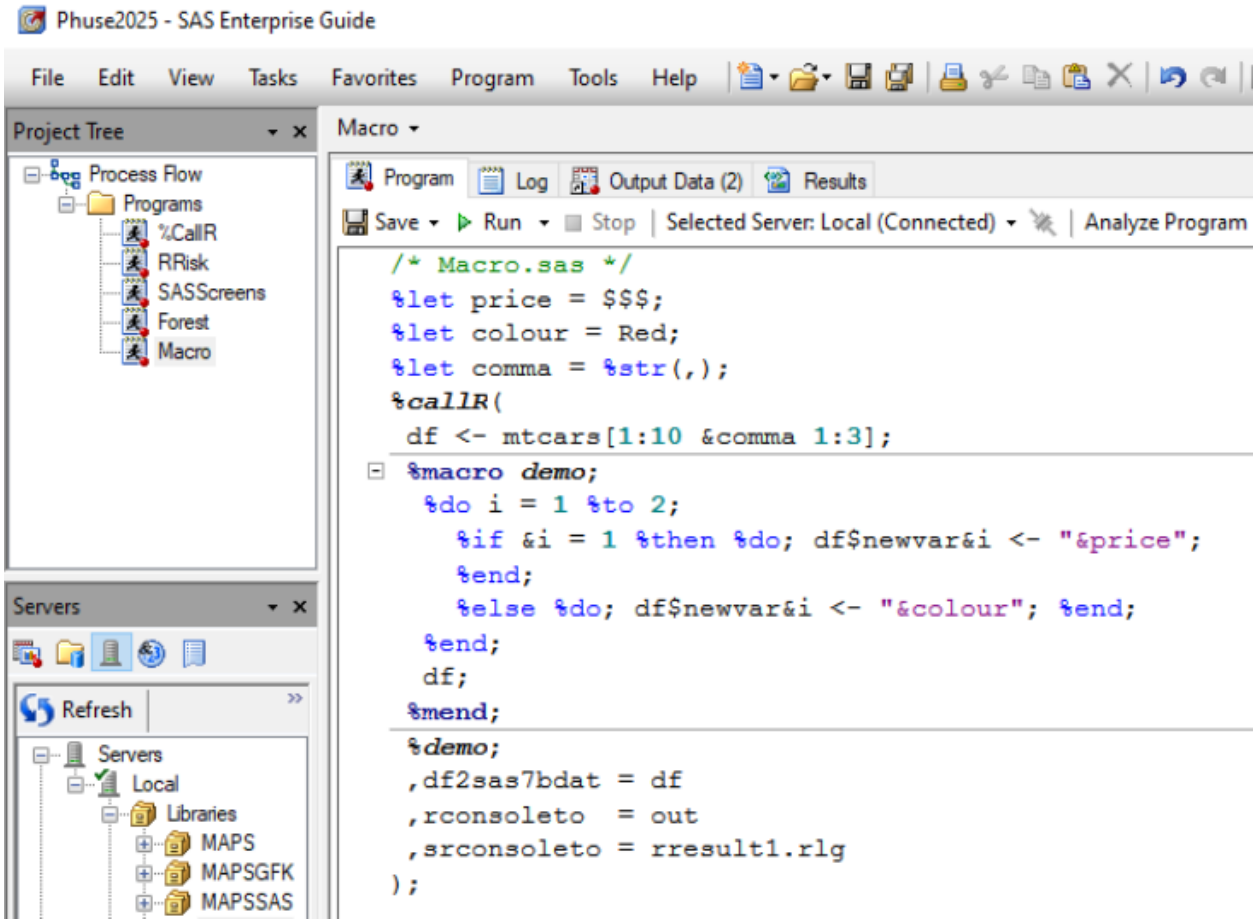


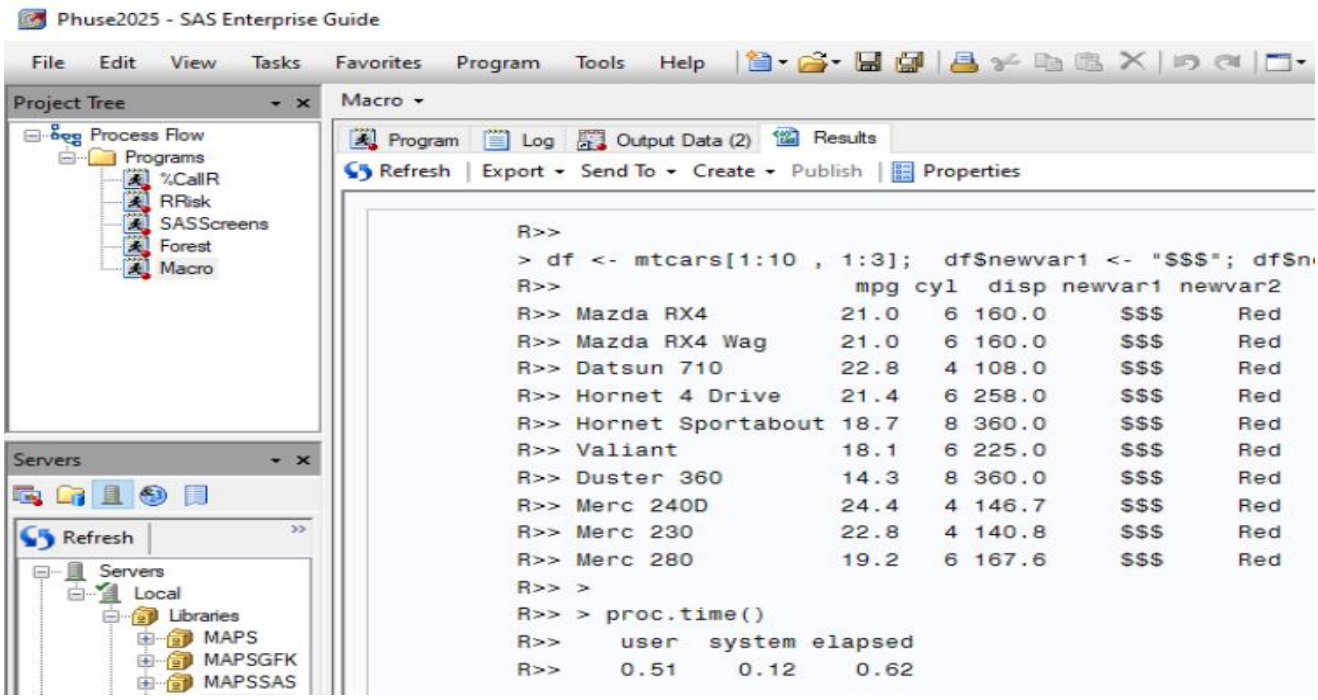

```
/* Forest.SAS */
%callR(
library(meta);

study <- c("Study 1", "Study 2", "Study 3", "Study 4");
effect <- c(0.5, 0.8, 0.3, 0.7);
lower_ci <- c(0.2, 0.5, 0.1, 0.4);
upper_ci <- c(0.8, 1.1, 0.5, 1.0);

meta_analysis <- metagen(TE = effect, lower = lower_ci,
upper = upper_ci, studlab = study, sm = "SMD");

png(filename="I:/Phuse2025/Forest.png", bg="transparent",
  width = 10, height = 3, units="in", res=300);
  forest(meta_analysis);
dev.off();

,rconsoleto  = PGM
,srconsoleto = rresult1.rlg
,displayG    = I:\Phuse2025\Forest.png
,displayGF   = rtf
);
```



SAS Enterprise Guide

SAS Enterprise Guide is a powerful, point-and-click Windows interface that provides access to the functionality of SAS. It has a modern Integrated Development Environment (IDE) for editing and running SAS programs. Again, %CallR works seamlessly via this IDE, as demonstrated by this purposeful example. That is interleaving R code with the SAS **macro language**. The implementation of this concept makes the flow of generating of R scripts more natural and dynamic.

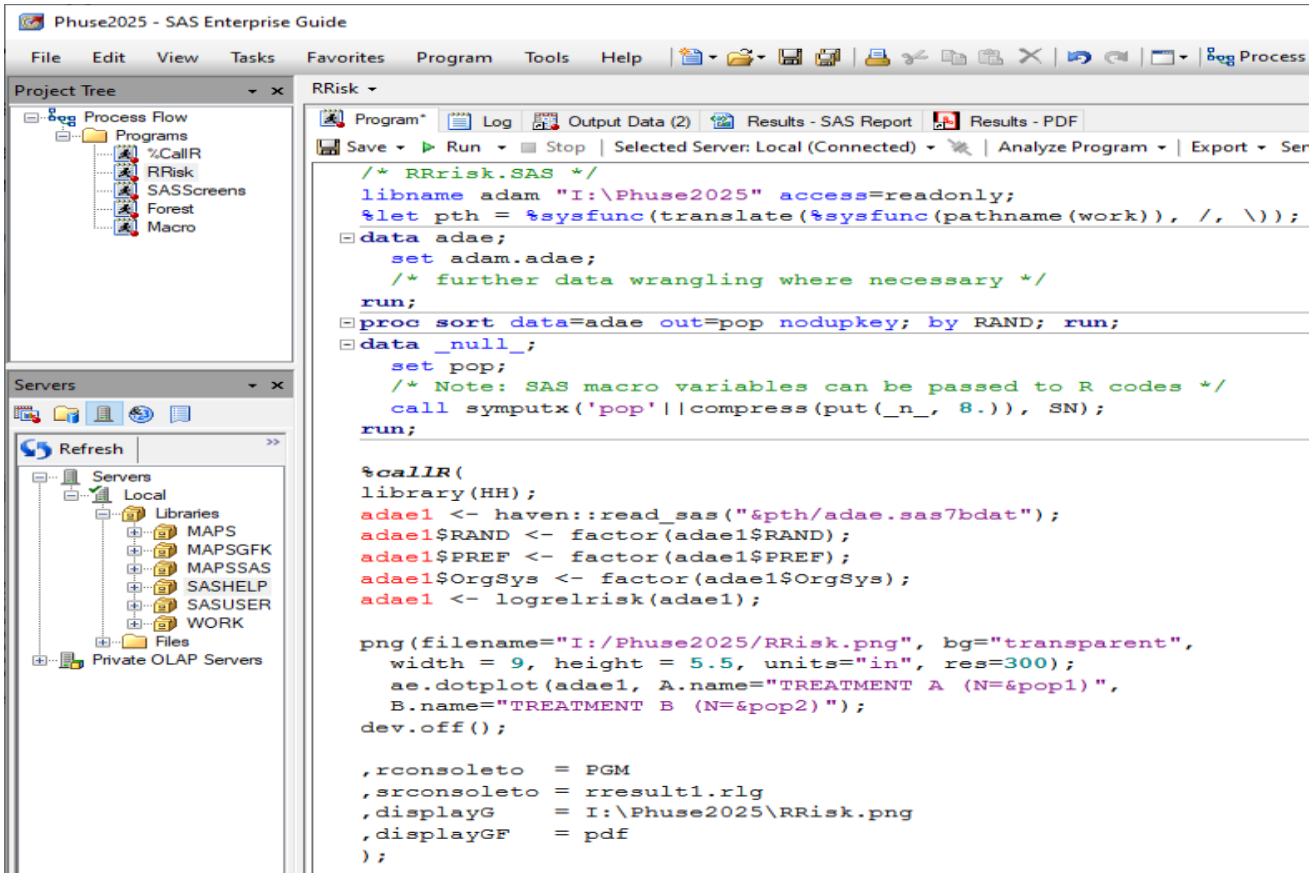


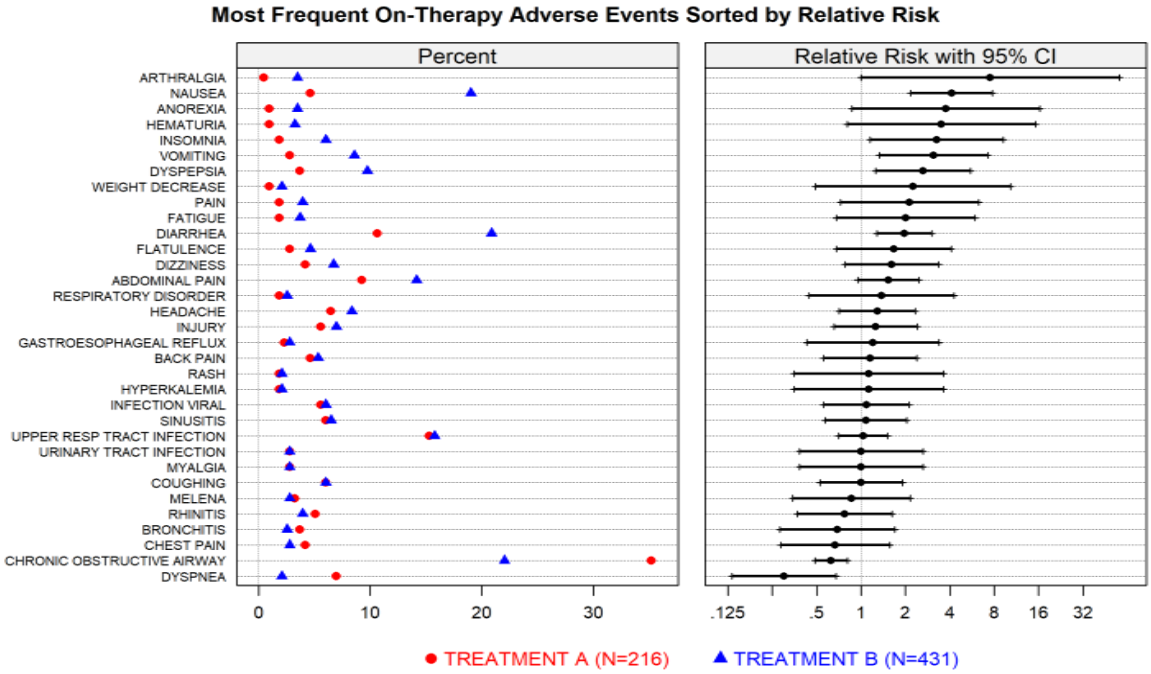


APPLICATIONS

Utilising R Packages in SAS

There are many functionalities from R packages that are not readily available in SAS. For example, these few lines of code generate a relative risk plot. So, using %callR, you can quickly customise this and alike into a library of SAS macros.





Here are a few more packages and there are many others that will add value to your SAS workflow.

- **ggplot2** - a powerful and flexible data visualisation package based on the Grammar of Graphics.
- **forestploter** - makes it easy to create publication-ready forest plots with great flexibility and customisation
- **gtsummary** - create publication-ready summary and analytic tables with minimal code

Define SAS Macro Functions with R Codes

In SAS, macro functions can include data steps and procedures. The DOSUBL function facilitates this powerful feature. A surprising finding is that DOSUBBL also allows the usage of other languages. Hence, we can utilize many existing R functions to quickly turn them into SAS macro functions and this is demonstrated here with just a few lines of code. Note, instead of using these ready-made R functions, you can, of course, write your own with R codes or even mix them with SAS codes.

Example 1

```
%macro Rfuns(fun);
%let rc=%sysfunc(dosubl(%nrstr(
  %callR(
    comment <- "you can also customise your R code function here";
        &fun;
        ,rconsoleto = OUT
        ,rvarname   = rvalue);
)));
&rvalue
%mend;
%put %Rfuns(mean(c(pi, 3.14))); * you can use already made R functions;
R>> [1] 3.140796

%let set1 = c('some', 'random', 'few', 'words');
%let set2 = c('some', 'many', 'none', 'few');
%put %Rfuns(base::intersect(&set1, &set2));
R>> [1] "some" "few"

%put %Rfuns(t.test(mtcars$mpg, mu = 0)$p.value);
R>> [1]1.526151e-18 1.526151e-18

%put %Rfuns(paste(base::list.files(R.home()), collapse = ' '));
R>> [1] "bin CHANGES COPYING doc etc include library MD5 module"
```

Example 2

It is quite common to iterate through a list of files in a folder for some particular task. For example, convert them into a different format etc. SAS does not have a function for this. We can write a SAS macro for this as shown here by using the **list.files** R function.

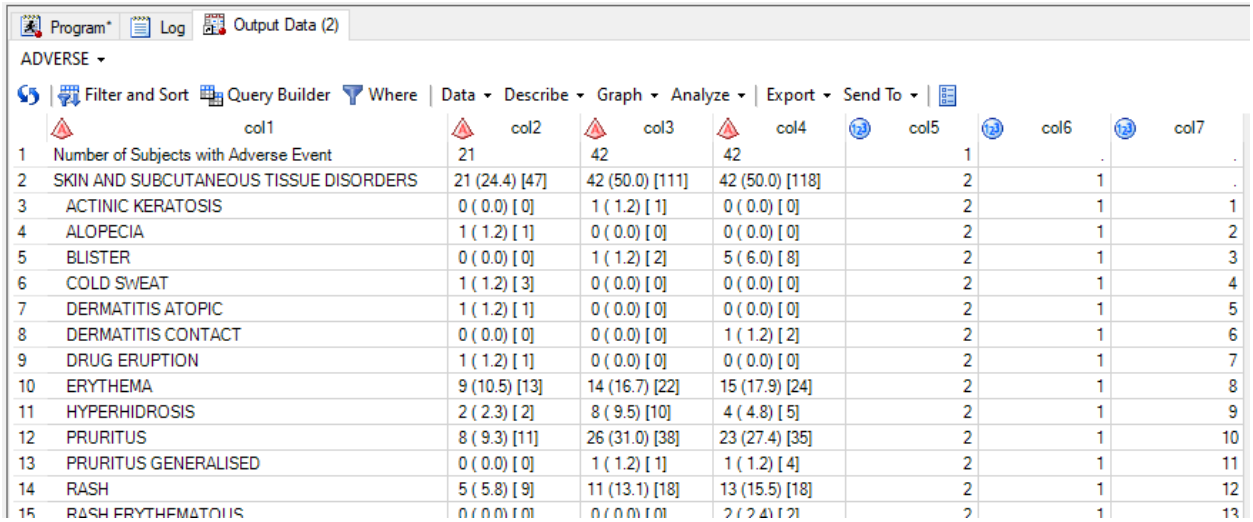
```
%macro getfilesR(
       path     =          /* path of the directory                              */
      ,pattern  =          /* regular expression for the files names             */
      ,delim    = %str(, ) /* delimier for files names                           */
      ,rmethod  = WORD     /* WORD | COUNT return (list | count) of files names */
      );

  %local rvalue;
  %let rvalue = _ERROR_;
  %let rc=%sysfunc(dosubl(%nrstr(
    options NOQUOTELENMAX;
    %callR(

     fnames <- list.files(path="&path", pattern = "&pattern", ignore.case = TRUE);
     fnames <- paste0(fnames, collapse = "&delim");
     fnames;

     ,rconsoleto = LOG
     ,rvarname   = getfiles_rvalue
    );

  )));

  /* note: you can use SAS codes outside of %callR per usual */
  %if %sysfunc(upcase(&rmethod)) = WORD %then %do;
    %let rvalue = &getfiles_rvalue;
  %end;
  %else %if %sysfunc(upcase(&rmethod)) = COUNT %then %do;
    %let rvalue = %sysfunc(countw(&getfiles_rvalue, "&delim"));
  %end;

  &rvalue

%mend;

/*
%let files = %getfilesR(path=I:/Phuse2025/rtf, pattern=\\.rtf$, delim=%str( # ),
rmethod=WORD);

%put LIST OF FILES : &files;

%put NUMBER OF FILES: %getfilesR(path=I:/Phuse2025/rtf/, pattern=\\.rtf$, delim=%str(
# ), rmethod=COUNT);

data test;
  files  = &files;
  nfiles = %getfilesR(path=I:/Phuse2025/rtf, pattern=\\.rtf$, delim=%str( # ),
  rmethod=COUNT);
run;
*/
```

Using R to Validate TFLs

Double programming is a common practice for validating tables, figures and listings (TFLs) in a clinical trial study report. To improve the efficiency of this process, we can harness the already existing functions from R packages such as {gtSummary}, {cards}, {Tplyr}, {tidytlg} and {tern} into our workflow, providing an advantageous alternative to writing all code in SAS from scratch and example 3 below demonstrates this with {Tplyr}.

Example 3

```
%callR(
library(Tplyr);
library(dplyr);

adsl <- tplyr_adsl %>%
  mutate(TRTA = TRT01A);

adae <- tplyr_adae %>%
  select(-SAFFL);

aetab <- tplyr_table(adae, TRTA) %>%
  set_pop_data(adsl) %>%
  set_pop_treat_var(TRTA) %>%
  set_pop_where(SAFFL == "Y") %>%
  set_distinct_by(USUBJID) %>%

  add_layer(
    group_count("Number of Subjects with Adverse Event") %>%
      set_nest_count(TRUE)  %>%
      set_format_strings(f_str("xx", distinct_n))
  ) %>%

  add_layer(
    group_count(vars(AEBODSYS, AEDECOD)) %>%
      set_nest_count(TRUE) %>%
      set_format_strings(f_str("xx (xx.x) [xx]", distinct_n, distinct_pct, n))
  ) %>%

build();

names(aetab) <- c("col1", "col2", "col3", "col4", "col5", "col6", "col7");
haven::write_xpt(aetab, "I:/Phuse2025/adverse.xpt", version = 5);
,rconsoleto = LOG);

/* read the xpt file generated by R into SAS dataset */
libname xptfile xport "I:/Phuse2025/adverse.xpt";
data adverse;
  set xptfile.adverse;
  /* further data wrangling to the table below before comparing to the production
     result etc */
run;
```



Program* | Log | Output Data (2)

ADVERSE

Filter and Sort | Query Builder | Where | Data | Describe | Graph | Analyze | Export | Send To

	col1	col2	col3	col4	col5	col6	col7
1	Number of Subjects with Adverse Event	21	42	42	1	.	.
2	SKIN AND SUBCUTANEOUS TISSUE DISORDERS	21 (24.4) [47]	42 (50.0) [111]	42 (50.0) [118]	2	1	.
3	ACTINIC KERATOSIS	0 (0.0) [0]	1 (1.2) [1]	0 (0.0) [0]	2	1	1
4	ALOPECIA	1 (1.2) [1]	0 (0.0) [0]	0 (0.0) [0]	2	1	2
5	BLISTER	0 (0.0) [0]	1 (1.2) [2]	5 (6.0) [8]	2	1	3
6	COLD SWEAT	1 (1.2) [3]	0 (0.0) [0]	0 (0.0) [0]	2	1	4
7	DERMATITIS ATOPIC	1 (1.2) [1]	0 (0.0) [0]	0 (0.0) [0]	2	1	5
8	DERMATITIS CONTACT	0 (0.0) [0]	0 (0.0) [0]	1 (1.2) [2]	2	1	6
9	DRUG ERUPTION	1 (1.2) [1]	0 (0.0) [0]	0 (0.0) [0]	2	1	7
10	ERYTHEMA	9 (10.5) [13]	14 (16.7) [22]	15 (17.9) [24]	2	1	8
11	HYPERHIDROSIS	2 (2.3) [2]	8 (9.5) [10]	4 (4.8) [5]	2	1	9
12	PRURITUS	8 (9.3) [11]	26 (31.0) [38]	23 (27.4) [35]	2	1	10
13	PRURITUS GENERALISED	0 (0.0) [0]	1 (1.2) [1]	1 (1.2) [4]	2	1	11
14	RASH	5 (5.8) [9]	11 (13.1) [18]	13 (15.5) [18]	2	1	12
15	RASH ERYTHEMATOUS	0 (0.0) [0]	0 (0.0) [0]	2 (2.4) [2]	2	1	13

CONCLUSION

SAS and R are powerful tools for data analysis and visualisations, though their paradigms are different. By leveraging their strengths together, it can, and will, boost productivity. This paper presents the key features and functionalities of %CallR. The selected examples show how the two languages are seamlessly interfaced. For further examples, refer to the appendix.

ACKNOWLEDGEMENTS

I would like to thank Chris Dixon from amgen.com, Yen Phan from codlad.com and Gerry Downey from bluebirdbio.com for their reviews and comments on this paper.

REFERENCES

[1] https://www.lexjansen.com/phuse/2005/cc/cc03.pdf
[2] https://www.jstatsoft.org/article/view/v046c02
[3] https://www.lexjansen.com/sesug/2016/AD-118_Final_PDF.pdf
[4] https://rpubs.com/dtran01/blog4]

CONTACT INFORMATION

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

Author Name: Duong Tran
Company: Elderbrook Solutions GmbH
Address: Sky Tower, Borsigstr. 4, D-74321 Bietigheim-Bissingen
Tel: +49 71423392919
info@elderbrook.de
Email: duong.tran@ext.elderbrooksolutions; trand000@aol.com
Website: https://elderbrook.de/

APPENDIX

We have already seen a few applications of R functionalities seamlessly interfacing within a SAS workflow via %CallR. Now let us see a few more examples.

Example 4 – Parsing an RTF Table

Rich text format (RTF) is often preferred for clinical reporting tables and listings and one may wish to extract its contents and compare them to a validation dataset as a validation process. The **read_rtf** function in the {striprtf} package makes it very easy to parse RTF codes and convert the RTF table into a dataset.

```
/* RTFParser.sas */
%macro RTFParser(ifile  =           /* input RTF file       */
                ,odset  = odset     /* output dataset       */
                ,delim  = @         /* delimiter            */
                ,tfname = tfname    /* temporary file name  */
                );

%let swpath = %sysfunc(pathname(work));
%let rwpath = %sysfunc(translate(&swpath, %str(/), %str(\)));

*** calling SAS to R interface (%callR) ***;
%callR(
    read_rtf_df <- striprtf::read_rtf("&ifile"
    ,row_start = ""
    ,row_end = ""
    ,cell_end = " &delim "
    );

    read_rtf_df <- as.data.frame(read_rtf_df);
    names(read_rtf_df) <- "ALLCOLS";
    RTFParser_dset <- dplyr::mutate_if(read_rtf_df, is.factor, as.character);
    haven::write_xpt(RTFParser_dset, "&rwpath/&tfname..xpt", version = 5);
  );

libname xptfile xport "&rwpath\&tfname..xpt";
data &tfname;
  set xptfile.&tfname;
run;

proc sql noprint;
  select max(countw(ALLCOLS, "&delim") - 1) into: maxcol
  from &tfname
  ;
quit;

%let maxcol = %sysfunc(compress(&maxcol));

data &odset;
  set &tfname;
  array cols $200 c1-c&maxcol;
  do c = 1 to &maxcol;
    cols(c) = scan(ALLCOLS, c, "&delim");
  end;
  keep c1-c&maxcol;
run;

%mend;


/*
%RTFParser(ifile = I:/Phuse2025/l_demo2.rtf
          ,odset = odset
          ,delim = #
          );
*/
```

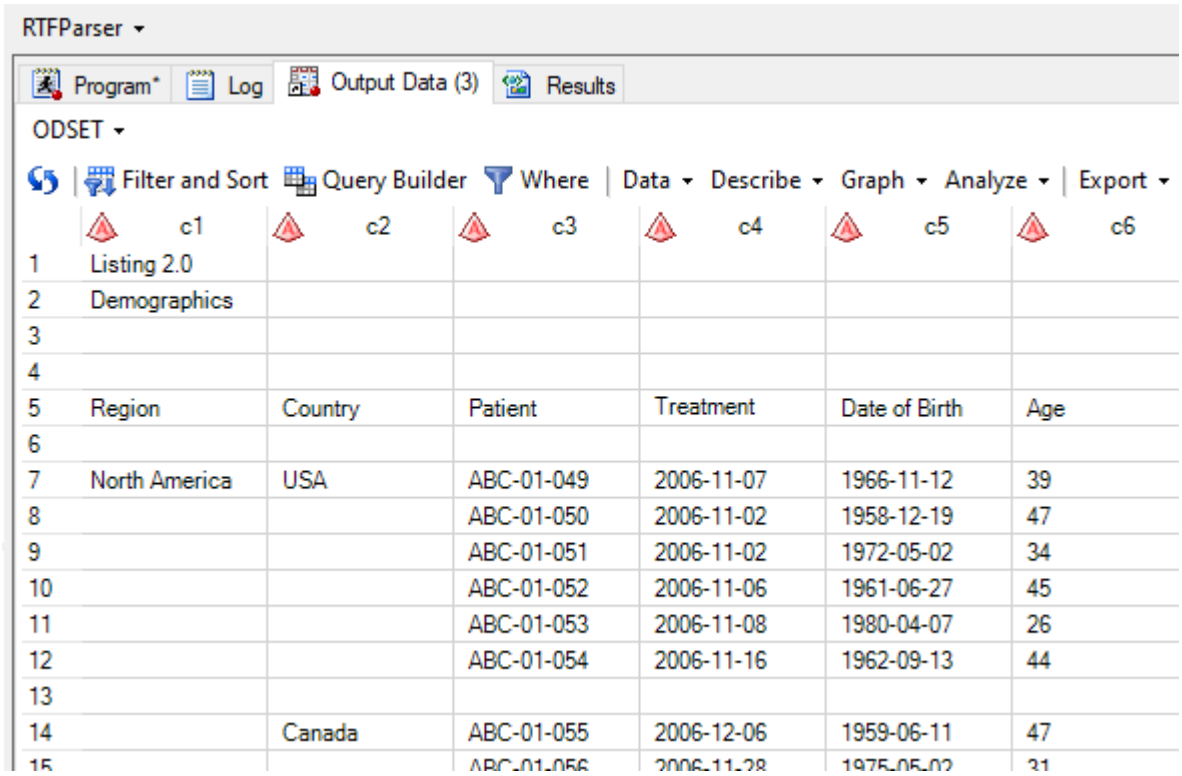
The RTF table with data to be extracted:

Listing 2.0

Demographics

Region	Country	Patient	Treatment Date	Date of Birth	Age
North America	USA	ABC-01-049	2006-11-07	1966-11-12	39
		ABC-01-050	2006-11-02	1958-12-19	47
		ABC-01-051	2006-11-02	1972-05-02	34
		ABC-01-052	2006-11-06	1961-06-27	45
		ABC-01-053	2006-11-08	1980-04-07	26
		ABC-01-054	2006-11-16	1962-09-13	44
	Canada	ABC-01-055	2006-12-06	1959-06-11	47
		ABC-01-056	2006-11-28	1975-05-02	31
		ABC-01-113	2006-12-05	1932-02-08	74
		ABC-01-114	2006-12-14	1934-07-09	72

Dataset of the extracted RTF table:

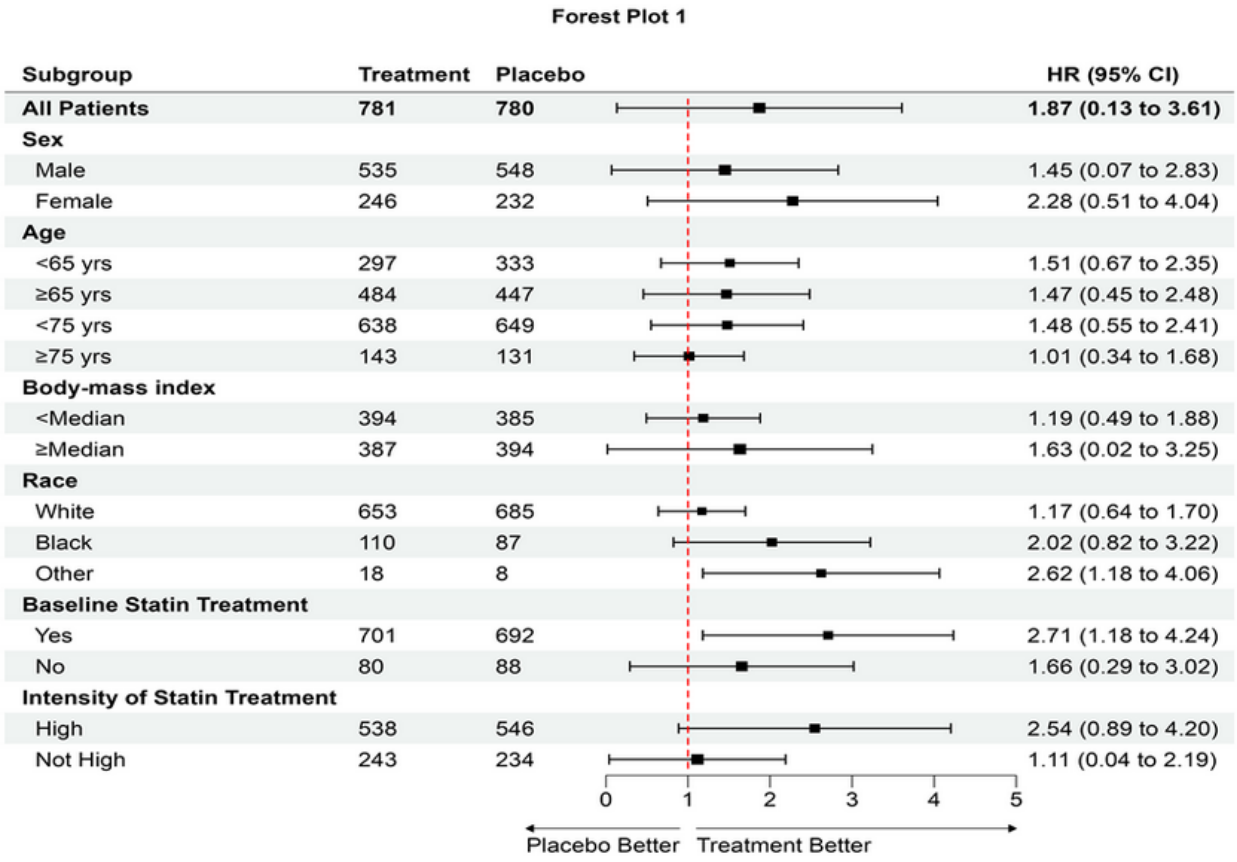


	c1	c2	c3	c4	c5	c6
1	Listing 2.0					
2	Demographics					
3						
4						
5	Region	Country	Patient	Treatment	Date of Birth	Age
6						
7	North America	USA	ABC-01-049	2006-11-07	1966-11-12	39
8			ABC-01-050	2006-11-02	1958-12-19	47
9			ABC-01-051	2006-11-02	1972-05-02	34
10			ABC-01-052	2006-11-06	1961-06-27	45
11			ABC-01-053	2006-11-08	1980-04-07	26
12			ABC-01-054	2006-11-16	1962-09-13	44
13						
14		Canada	ABC-01-055	2006-12-06	1959-06-11	47
15			ABC-01-056	2006-11-28	1975-05-02	31

Example 5 – Forest Plot

Forest plots like the one below are time-consuming to produce using the SAS SG Procedures. With the {forestploter} package, this is quick and straightforward. That is, by just feeding in the results data and setting a few parameters to the **forest** function, you will have this nice-looking plot.

```
data result;
  <Derrive results in SAS>
run;


%callR(
rdat <- haven::read_sas("result.sas7bdat");

p <- forest(rdat [,c(1:3, 20:21)],
            est = rdat $est,
            lower = rdat $low,
            upper = rdat $hi,
            sizes = rdat $se,
            ci_column = 4,
            ref_line = 1,
            arrow_lab = c("Placebo Better", "Treatment Better"),
            xlim = c(0, 5),
            ticks_at = c(0, 1, 2, 3, 4,5));

etc…

ggsave( "forest.png", plot = p, device = "png", path   = "I:/Phuse2025");

,displayG = I:\Phuse2025\Forest_Plot.png
);
```



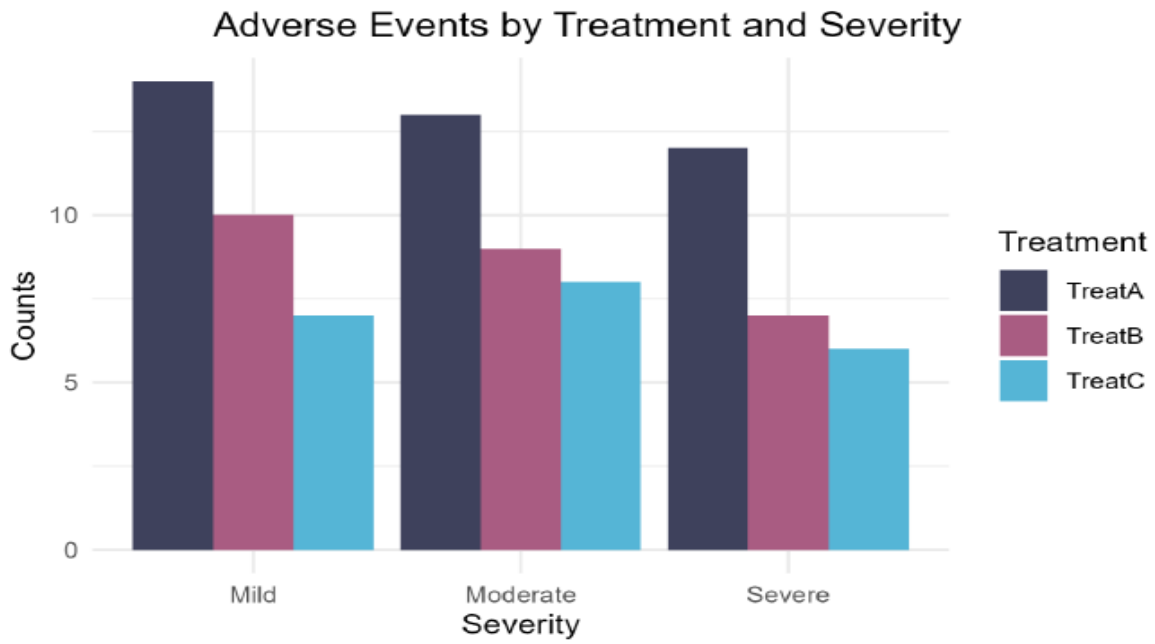
Example 6 – Interaction with Excel
Interaction with Excel from SAS is a common task, and SAS offers multiple methods – **proc export/import**, **output delivery system** and the **libname engine**. However, there are R packages, for example {openxlsx} that are more versatile, and you may wish to adopt instead, into your SAS workflow.

```
libname ads "I:\Phuse2025";
data ads.adae;
input Severity $ TreatA TreatB TreatC;
cards;
Mild     14 10 7
Moderate 13 9  8
Severe   12 7 6
;
run;

%callR(
library(openxlsx); library(ggplot2); library(tidyr);

%let pth = I:/Phuse2025;
adae <- haven::read_sas("&pth/adae.sas7bdat");
adae_long <- pivot_longer(adae, cols = -Severity, names_to = "Treatment",
values_to = "Counts");
adae_long$Treatment <- factor(adae_long$Treatment, levels = c("TreatA", "TreatB",
"TreatC"));
adae_long$Severity <- factor(adae_long$Severity, levels = c("Mild", "Moderate",
"Severe"));

p <- ggplot(adae_long, aes(x = Severity, y = Counts, fill = Treatment)) +
  geom_bar(stat = "identity", position = "dodge") +
  scale_fill_manual(values = c("TreatA" = "#3e415c", "TreatB" = "#a95c82",
"TreatC" = "#55b5d6")) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5), plot.title.position = "plot") +
  labs(title = "Adverse Events by Treatment and Severity", x = "Severity",
y = "Counts");

ggsave("&pth/ae_chart.png", plot = p, width = 5.5, height = 3.5);

wb <- createWorkbook();
addWorksheet(wb, "Report");

titleStyle <- createStyle(fontSize = 14, fontColour = "#FFFFFF", fgFill = "#4F81BD",
halign = "CENTER", textDecoration = "Bold");
mergeCells(wb, sheet = "Report", cols = 1:4, rows = 1);
writeData(wb, sheet = "Report", x = "Adverse Events by Treatment and Severity",
startCol = 1, startRow = 1);
addStyle(wb, sheet = "Report", style = titleStyle, rows = 1, cols = 1:4,
gridExpand = TRUE);
centerStyle <- createStyle(halign = "LEFT");
addStyle(wb, sheet = "Report", style = centerStyle, rows = 3:(3 + nrow(adae)),
cols = 2:4, gridExpand = TRUE);
addStyle(wb, sheet = "Report", style = centerStyle, rows = 3, cols = 1:4,
gridExpand = TRUE);
boldHeaderStyle <- createStyle(textDecoration = "bold");
addStyle(wb, sheet = "Report", style = boldHeaderStyle, rows = 3, cols = 1:4,
gridExpand = TRUE);
textboxStyle <- createStyle(
  fontSize = 11,
  fontColour = "#000000",
  fgFill = "#FFF2CC",
  halign = "LEFT",
  valign = "CENTER",
  border = "TopBottomLeftRight",
  wrapText = TRUE
);

mergeCells(wb, sheet = "Report", cols = 1:4, rows = 30:33);
```

```
writeData(wb, sheet = "Report", x = "Note: This report was generated by the {openxlsx}
R package. It has many customisable functionalities, such as, style cells, adding
filters and freeze panes, inserting images and plots, merging cells, adding data
validation and conditional formatting, support for formulas, multiple sheet
management.", startCol = 1, startRow = 30);

addStyle(wb, sheet = "Report", style = textboxStyle, rows = 30:33, cols = 1:4,
gridExpand = TRUE);

setColWidths(wb, sheet = "Report", cols = 1:4, widths = c(15, 20, 20, 20));
setRowHeights(wb, sheet = "Report", rows = 30:33, heights = 20);

writeData(wb, sheet = "Report", x = adae, startCol = 1, startRow = 3);
insertImage(wb, sheet = "Report", file = "&pth/ae_chart.png", startRow = 10,
startCol = 1, width = 6, height = 4);

saveWorkbook(wb, "&pth/Ae_Report_with_Chart.xlsx", overwrite = TRUE);

,rconsoleto  = PGM
,temploc     = D:\duong
,srconsoleto = rresult1.rlg
);
```

Adverse Events by Treatment and Severity

Severity	TreatA	TreatB	TreatC
Mild	14	10	7
Moderate	13	9	8
Severe	12	7	6



Note: This report was generated by the {openxlsx} R package. It has many customisable functionalities, such as, style cells, adding filters and freeze panes, inserting images and plots, merging cells, adding data validation and conditional formatting, support for formulas, multiple sheet management.