

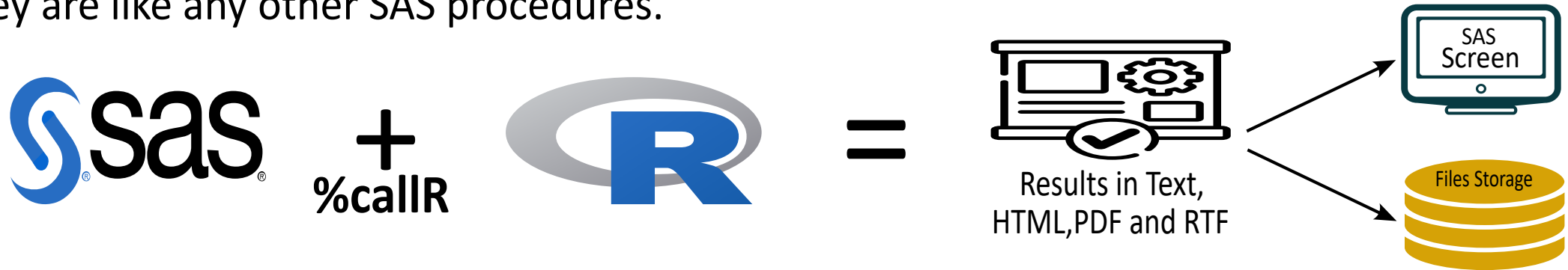
Abstract

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

3.1 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;
```

- filename proc_r pipe "R.exe CMD BATCH --vanilla -quiet r_pgm.r r_pgm.rlg";
data _null_; infile proc_r; run;
- systask command "R.exe CMD BATCH --vanilla --quiet r_pgm.r r_pgm.rlg" wait;
- x "R.exe CMD BATCH --vanilla --quiet r_pgm.r r_pgm.rlg";
- %sysexec R.exe CMD BATCH --vanilla --quiet r_pgm.r r_pgm.rlg;

3.2 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;
```

3.3 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 section 5.3. This makes %callR highly flexible for interleaving the two languages.

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 in section 5 and 6 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

5.1 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)  
SAS - [Output - (Untitled)]  
File Edit View Tools Solutions Window Help  
The SAS System  
R>> mtcars  
R>> Mazda RX4  
R>> Mazda RX4 Wag  
R>> Datsun 710  
R>> Hornet 4 Drive  
R>> Hornet Sportabout  
R>> Valiant  
R>> Duster 360  
R>> Merc 240D  
R>> Merc 230  
R>> Merc 280  
R>> Merc 280C  
R>> Merc 450SE  
R>> Merc 450SL  
R>> Merc 450SLC  
R>> Cadillac Fleetwood  
R>> Lincoln Continental  
R>> Chrysler Imperial  
R>> nmpg cyl disp hp drat wt qsec vs am gear carb  
R>> 21.0 6 160.0 110 3.90 2.820 16.46 0 1 4 4  
R>> 21.0 6 160.0 110 3.90 2.875 17.02 0 1 4 4  
R>> 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1  
R>> 21.4 6 258.0 110 3.08 3.215 19.44 1 0 3 1  
R>> 18.7 8 360.0 175 3.15 3.440 17.92 0 0 3 2  
R>> 18.1 6 225.0 105 2.76 3.460 20.22 1 0 3 1  
R>> 14.3 8 360.0 245 3.21 3.570 15.84 0 0 3 4  
R>> 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 4  
R>> 22.8 4 140.8 95 3.92 3.150 22.30 1 0 4 2  
R>> 19.2 6 167.6 123 3.92 3.440 18.30 1 0 4 4  
R>> 17.8 6 167.6 123 3.92 3.440 18.30 1 0 4 4  
R>> 16.4 8 275.8 180 3.07 4.070 17.40 0 0 3 3  
R>> 17.3 8 275.8 180 3.07 4.070 17.40 0 0 3 3  
R>> 16.4 8 275.8 180 3.07 4.070 17.40 0 0 3 3  
R>> 15.2 8 275.8 180 3.07 4.070 17.40 0 0 3 3  
R>> 10.4 8 472.0 265 2.30 5.250 17.98 0 0 3 4  
R>> 10.4 8 460.0 215 3.00 5.424 17.92 0 0 3 4  
R>> 14.7 8 440.0 230 3.23 5.345 17.42 0 0 3 4
```

```
%CallR(house_data, ... , rconsoleto=log)  
SAS - [Log - (Untitled)]  
File Edit View Tools Solutions Window Help  
R>> house_data <- data.frame( square_feet = c(1000, 1500, 2000, 2500, 3000), price =  
R>> model <- lm(price ~ square_feet, data = house_data); summary(model);  
R>> Call:  
R>> lmFormula ~ price ~ square_feet, data = house_data)  
R>> Residuals:  
R>> 1 2 3 4 5  
R>> 4000 4000 -10000 4000 4000  
R>> Coefficients:  
R>> Estimate Std. Error t value Pr(>|t|)  
R>> (Intercept) 96000.000 13856.406 6.928 0.000155 **  
R>> square_feet 100.000 6.532 15.309 0.000000 ***  
R>> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
R>> Residual standard error: 19330 on 3 degrees of freedom  
R>> Multiple R-squared: 0.9874, Adjusted R-squared: 0.9891  
R>> F-statistic: 234.4 on 1 and 3 DF, p-value: 0.000653
```

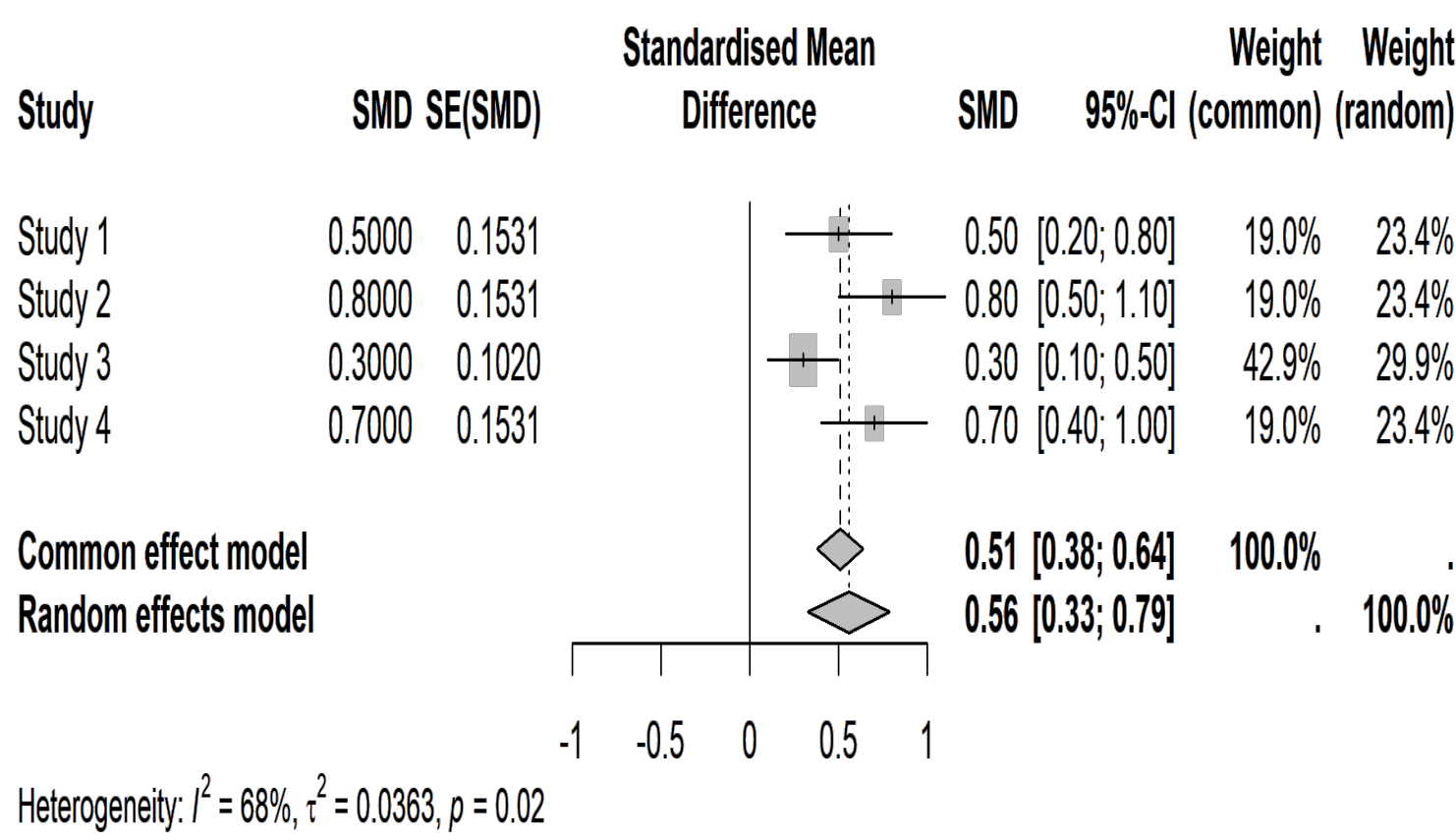
```
proc printto new file="I:\Phuse2025\mtcars.lst";  
run;  
%callR(mtcars, rconsoleto=out)  
proc printto;  
run;
```

```
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;
```

5.2 Graphics from R

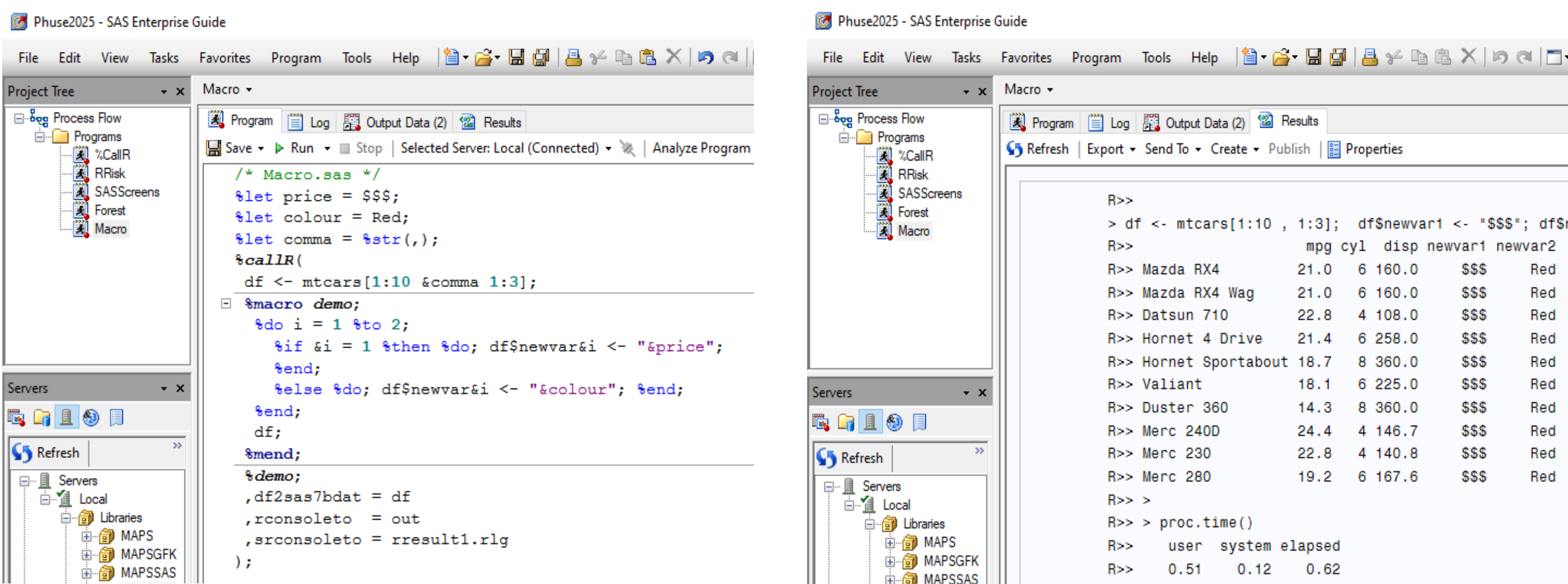
```
File Edit View Tools Run Solutions Window Help  
/* Forest.SAS */  
%callR(  
library(meta);  
  
study <- c("Study 1", "Study 2", "Study 3", "Study 4");  
effect <- c(0.5, 0.8, 0.3, 0.7);  
lower_ci <- c(0.2, 0.5, 0.1, 0.4);  
upper_ci <- c(0.8, 1.1, 0.5, 1.0);  
  
meta_analysis <- metagen(TE = 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  
);
```

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.



5.3 SAS Enterprise Guide

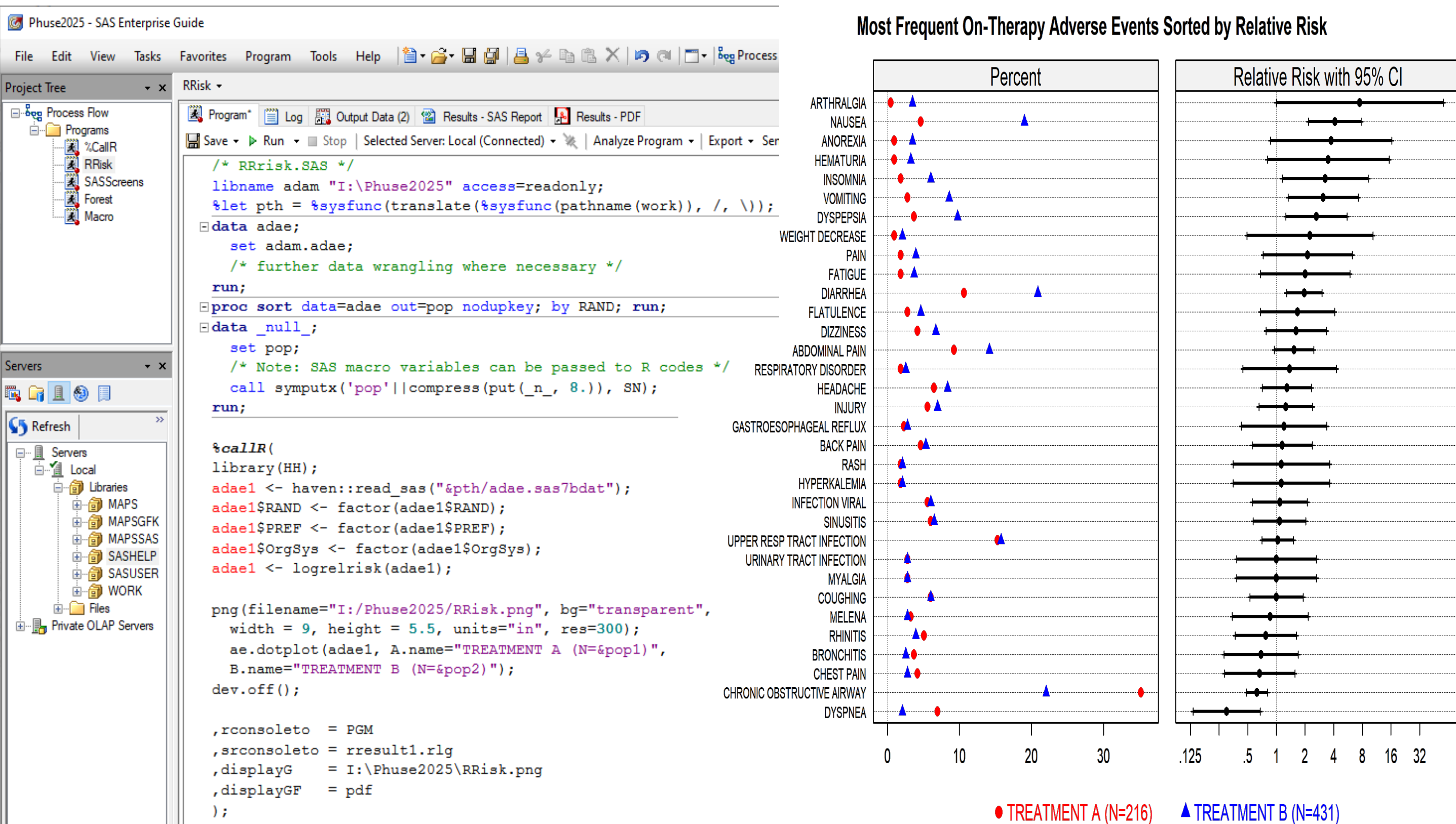
SAS Enterprise Guide is a powerful, point-and-click Windows interface that provides access to the functionality of SAS. It has a modern Integrated Development Environment (IDE) for editing and running SAS programs. Again, %CallR works seamlessly via this IDE, as demonstrated by this purposeful example.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



Applications

6.1 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.

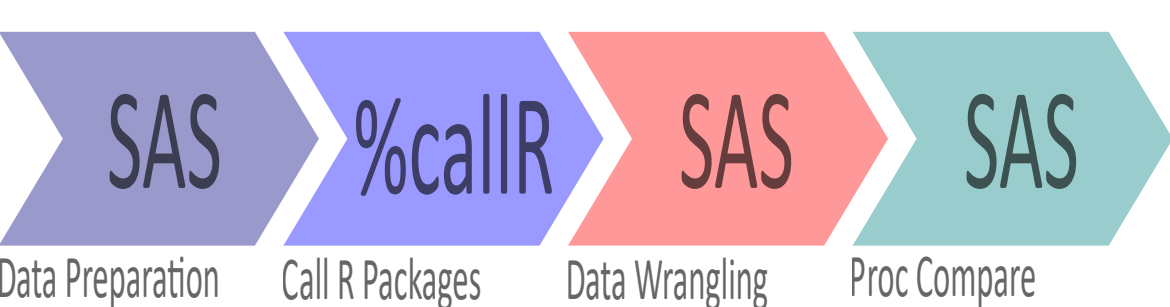
- ggplot2 - a powerful and flexible data visualisation package based on the Grammar of Graphics.
- forestploter - makes it easy to create publication-ready forest plots with great flexibility and customisation
- gtsummary - create publication-ready summary and analytic tables with minimal code

6.2 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 DOSUBL 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.

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

6.3 Using R to Validate TFLs



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 poster 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 accompanying paper.

Acknowledgments

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

References

- <https://www.lexjansen.com/phuse/2005/cc/cc03.pdf>
- <https://www.jstatsoft.org/article/view/V046c02>
- https://www.lexjansen.com/sesug/2016/AD-118_Final_PDF.pdf
- <https://rpubs.com/dtran01/blog4>