PharmaSUG 2022 - Paper AP-193

R syntax for SAS programmers

Max Cherny, GlaxoSmithKline

ABSTRACT

Making the transition from SAS programming to R programming can be quite challenging, especially for SAS programmers who have never coded in an object-oriented language. However, learning to code in R does not have to be difficult for those already fluent in SAS. Both languages provide similar functionality, with the main difference being the syntax required to accomplish the same objective. This paper provides a side-by-side comparison of R and SAS syntax, using examples of a clinical programmer's typical SAS code adjacent to the corresponding R syntax. These easy and reusable examples can function either as stand-alone programs or as starting points for further acquisition of R fluency for the SAS programmer.

INTRODUCTION

As more pharmaceutical companies are starting to use R, it is becoming important for SAS users to become familiar with R to stay competitive. Moving from SAS to R can be challenging as these are very different programming languages. R is an object-oriented language, and R syntax may seem rather complex to a SAS programmer used to the relative simplicity of SAS language. However, despite the differences, R and SAS were designed for the same purpose of statistical analysis. As a result, R shares many similar functionalities with SAS. This paper will provide contrast and examples of some common R and SAS syntax.

STARTING R

Starting R is easy. The first step is to download R from r-project.org and install it on a local machine. R runs on most common systems such as Windows or UNIX. In addition to installing R, it is recommended to install the free R integrated development environment called RStudio from rstudio.com. R studio is a user-friendly tool for developing various R programs. There are five major areas (tabs) in R during each session. The workspace is the area containing all active objects. The console section is where a programmer can enter programming commands and view the output. The history tab displays previously used commands. Files are displayed in the files tab. The plots tab contains graphs and the packages lists all installed packages.

BASICS OF R SYNTAX

Everything in R is an object. The most fundamental types of R objects are *data types* and *functions*. The most basic data types are *numbers, matrices, vectors, factors, lists* and *data frames*. While these data types all play important roles, the most important and familiar type to a SAS user is the data frame. The data frame is the collection of columns and rows, and it is very similar to the concept of a data set in SAS. Almost all examples of R syntax in this paper use a data frame. Various values in R are referred to as variables and are assigned using <- operator. R variables are similar to macro variables in SAS, although they can be declared and manipulated much easier. For example, a *Height<- 6* statement declares the Height variable and assigns a value of 6 to it.

Most SAS tasks are performed within data steps, procedures and macros. That is not the case with R. R uses functions to perform specific tasks that can be used anywhere. R programmers can also define their own functions when necessary.

BASE R AND TIDYVERSE

The R language is made up of packages. The basic version of R contains Base R, which is what is included with default installation of R. Other packages are used to provide additional functionality. As of 2020, there are almost 10,000 packages. The *install.package* command must be executed to install an R package.

The proliferation of various packages can make R even more complex. In recent years, RStudio's chief scientist Hadley Wickham developed a package called Tidyverse to address some of the shortcomings of R. The primary purpose behind Tidyverse is to streamline how R analyzes and processes the data. One of the principles behind Tidyverse is the use of data frames as the primary data types for R. SAS programmers will find Tidyverse syntax more familiar than Base R syntax. Tidyverse has recently became a very popular and commonly used package. Therefore, most code examples in this paper use Tidyverse syntax instead of base R. At GSK Biostatistics, we are prioritizing code that uses the Tidyverse over other code (e.g., base R) that can achieve the same outcome.

Use *install.packages("tidyverse")* command to install Tidyverse and *library(tidyverse)* to load the package.

CREATING A DATA FRAME

A data frame is the R equivalent of a SAS data set. The function to create a data frame is called *data.frame* in base R or *tibble* in Tidyverse. Each column is created by using the *c* function, and observations are separated by a comma. The R code below creates a data frame called *vitals*. The corresponding code to create the same dataset in SAS is shown as well.

SAS	R
data vitals; *input subjid visit dia sys wt;* *datalines;* *1 1 67 113 170* *1 2 70 114 160* *;* *run;*	*vitals<-tibble(* *subjid=c(1, 1),* *visit=c(1, 2),* *dia=c(67, 70),* *sys=c(113, 114),* *wt=c(170, 160)* *)*

subjid	visit	dia	sys	wt
1	1	67	113	170
1	2	70	114	160

Table 1. vitals data frame

VIEWING A DATA FRAME

print(vitals) command can be used to print out the contents of the data frame. It is very similar to PROC PRINT in SAS. The *view(vitals)* command allows the user to view the data frame in the way that SAS VIEWTABLE does:

COPYING A DATA FRAME

<- command can be used to copy a data frame. The equivalent of <- function in R is SET in SAS.

SAS	R
data vitals2; set vitals; run;	*vitals2<- vitals*

Deleting a dataset can be performed with the *rm* function.

CALCULATING A NEW VARIABLE

There are many ways a variable can be created in a data frame. I recommend using the *mutate* function of the Tidyverse package. The *mutate* function is one of the core functions of the Tidyverse package because it provides an efficient approach to making changes to a data frame. *mutate* is similar to a SAS data step in that it reads in and outputs a data set, but does this in one statement. The example below shows how to calculate a variable representing weight in kilograms.

SAS	R
data vitals2; set vitals; wt_kg=wt/2.2; unit="kg"; run;	*vitals2<-mutate(vitals, wt_kg=wt/2.2, unit="kg")*

subjid	visit	dia	sys	wt	wt_kg	unit
1	1	67	113	170	77.27273	kg
1	2	70	114	160	72.72727	kg

Table 2. vitals data frame with a derived variable

SORTING A DATA FRAME

Sorting a dataset in R can be done by using the *order* function in BASE R or the *arrange* function in Tidyverse. Using the *order* function requires the user to specifically reference the data frame with *$* notation, while, the *arrange* function makes it possible to reference the data frame directly. The example below will sort the demography data frame by subject id.

subjid	age
2	67
4	54
3	76

1	75

Table 3. DEMODATA dataset

SAS	R
proc sort data=demodata *out= demodata2;* *by subjid;* *run;*	*demodata2<-arrange(* *demodata,* *subjid)*

USING CONDITIONAL STATEMENTS

The R syntax for conditional statements such as *if* and *else* are similar to SAS syntax. The code below creates new category variables using conditional statements and the *cut* function. The *cut* function uses *labels* statement to assign character results to the newly derived column. *Inf* represents positive infinity.

SAS	R
data vitals2; *set vitals;* *length sys_flag $15;* *if sys > 113 then sys_flag="High";* *else sys_flag="Low or Normal";* *run;*	*vitals2<-mutate(vitals,* *sys_flag= cut (sys, c(-Inf, 113, Inf),* *labels= c("Low or Normal","High"))* *)*

subjid	visit	dia	sys	wt	sys_flag
1	1	67	113	170	Low or Normal
1	2	70	114	160	High

Table 4. VITALS data frame with a category

SUBSETTING A DATA FRAME

Just like in SAS, there are many ways to subset a table in R. One approach is to use the *filter* function. The example below demonstrates how to select vital sign records for visit 2. Note the "== " operator in R is different from "=" operator in SAS.

SAS	R
data vitals2; *set vitals;* *where visit=2;* *run;*	*vitals2<-filter (vitals,* *visit==2)*

subjid	visit	dia	sys	wt
1	2	70	114	160

Table 5. vitals data frame filtered by visit=2

DROPPING A VARIABLE

There are many ways to drop columns in both SAS and R. One such way in R is to add "–" sign before the variable and use the *select* function. The example below creates a new data frame called vitals2 without sys column.

SAS	R
data vitals2; *set vitals;* *drop sys;* *run;*	*vitals2<-select (vitals, -sys)*

LABELING A VARIABLE

There is no easy way to label a column in R like there is in SAS. However, several R packages exist which provide labeling functions. One such package is *labelled*. This package's function *var_label* can be used to label columns in a data frame.

SAS	R
data vitals2; *set vitals;* *label subjid="Subject ID"* *visit="Visit number"* *dia="Diastolic BP"* *sys="Systolic BP"* *wt="Weight";* *run;*	*install.packages("labelled")* *library("labelled")* *vitals2<-vitals* *var_label(vitals2) <- list(subjid="Subject ID",* *visit="Visit number",* *dia="Diastolic BP",* *sys="Systolic BP",* *wt="Weight")*

subjid Subject ID	visit Visit number	dia Diastolic BP	sys Systolic BP	wt Weight
1	1	67	113	170
1	2	70	114	160

Table 6. vitals data frame with labelled columns

CONVERTING DIFFERENT TYPES OF VARIABLES

Both character and numeric variables can sometimes be used together in SAS. For example, a character variable set to "1" can be added to a numeric variable of 2 to produce 3. However, attempting to do the same in R will produce an error. Therefore, functions *as.numeric and as.character* must be used to convert numeric and character variables. Let's assume we have the following data frame consisting of numeric and character variables representing numbers:

char	num
1	9999
222	44

Table 7. TEST dataset

The following code displays how to use convert a character variable CHAR to a numeric variable CHAR_TO_NUM and a numeric variable NUM to a character variable NUM_TO_CHAR.

SAS	R
data test2; *set test;* *char_to_num=input(char, 8.);* *num_to_char=put(num, 8. -L);* *run;*	*test2<-mutate(test,* *char_to_num=as.numeric(char),* *num_to_char=as.character(num))*

char	num	char_to_num	num_to_char
1	9999	1	9999
222	44	222	44

Table 8. TEST2 dataset

COMBINING DATASETS

PROC APPEND or SET commands can be used to stack datasets in SAS. The same task can be accomplished using the *bind_rows* function in R. The code below shows how to combine demo1 and demo2 data frames together to form a demo3 data frame:

subjid	age
1	60
2	55

Table 9. DEMO1 data frame

subjid	age
3	20
4	30

Table 10. DEMO2 data frame

SAS	R

data demo3; *set demo1* *demo2;* *run;*	*demo3<-bind_rows(demo1, demo2)*

subjid	age
1	60
2	55
3	20
4	30

Table 11. DEMO3 data frame

Merging data frames with R syntax can be done with the *merge* function in BASE R. However, Tidyverse uses a series of *join* functions where the name of the function represents the type of join, which is similar to SQL syntax. For example, if we wanted to merge vitals data frame with demo1 and only keep those subjects which are in both tables, the *inner_join* function can be used as follows:

SAS	R
data vitals2; *merge demo1 (in=a)* *vitals (in=b);* *by subjid; if a and b;* *run;*	*vitals2<-inner_join(demo1,* *vitals,by="subjid")*

subjid	visit	dia	sys	wt	age
1	1	67	113	170	60
1	2	70	114	160	60

Table 12. VITALS2 data frame

STRING MANIPULATIONS WITH R SYNTAX

Like SAS, R and Tidyverse have a very extensive set of powerful functions to manipulate and analyze text strings. The example below shows how to combine a character variable containing a drug treatment arm with the numeric variable representing the number of subjects in the treatment arm.

trtgrp	trt_counts
Drug A	50
Drug B	55

Table 13. RESULTS data frame

To combine strings together, the *str_c* function is used. This function is very similar to the SAS CAT function.

SAS	R
data results2; *set results;* *length trt_text $14;* *trt_text=cat(trtgrp, ' (N=', trt_counts, ')');* *run;*	*results2<-mutate(results,* *trt_text=str_c(trtgrp,* *sep=" ",* *'(N=',trt_counts, ')'))*

trtgrp	trt_counts	trt_text
Drug A	50	Drug A (N=50)
Drug B	55	Drug B (N=55)

Table 14. RESULTS2 data frame

TRANSPOSING A DATA FRAME

PROC TRANSPOSE is the procedure for transposing datasets in SAS. It can be used to perform many sorts of transposing. However, there are two separate functions in TIDYVERSE to perform transpose from long to wide and from wide to long formats. The function to transpose from long to wide format is the *pivot_wider* function.

subjid	visit	dia
1	1	77
1	2	76
2	1	82
2	2	80

Table 15. VITALS_DIA data frame

The following code transforms the diastolic blood pressure column into two columns per each visit.

SAS	R
proc transpose data=vitals_dia out=tr_dia (drop=_name_) prefix=visit ; *by subjid;* *var dia;* *run;*	*tr_dia<-pivot_wider(vitals_dia,* *names_from=visit,* *names_prefix = "visit:",* *values_from=c("dia"))*

subjid	visit:1	visit:2
1	77	76
2	82	80

Table 16. TR_DIA data frame

The *pivot_longer* function is used to perform the opposite role. In the following example, we have columns for DBP and SBP. They will be combined into one parameter using *names_to* parameter.

subjid	visit	dia	sys
1	1	67	113
1	2	70	114

Table 17. VITALS_DIA_SYS data frame

SAS	R
proc transpose data=vitals_dia_sys *out=tr_vs* *(rename=(_name_=paramcd col1=aval)) ;* *by subjid visit;* *run;*	*tr_vs<-pivot_longer(vitals_dia_sys,* *cols=c(dia,sys),* *names_to = "paramcd")*

subjid	visit	paramcd	aval
1	1	dia	67
1	1	sys	113
1	2	dia	70
1	2	sys	114

Table 18. TR_VS data frame

REMOVING DUPLICATE RECORDS

Removing duplicates is a very common activity in SAS programming. It is usually done in SAS using the NODUPKEY option in a SORT procedure. The *distinct* function in R was designed specifically for the same purpose. If *.keep_all* is TRUE, then all variables are kept in the dataset.

SAS	R
proc sort data= vitals_dia_sys *nodupkey out=no_dups;* *by subjid;* *run;*	*no_dups<-distinct(vitals_dia_sys,* *subjid,* *.keep_all=TRUE)*

subjid	visit	dia	sys
1	1	67	113

Table 19. VITALS_DIA_SYS data frame without duplicate SUBJID

USING PIPES IN R

The pipe operator in R is used to chain various functions together so that the output of one function is the input into the next. The syntax for pipe is *%>%*. The R code below is used to sort a data frame by subject and then derive the wt variable in kilograms using pipes to chain the functions together. The same SAS syntax requires using a separate procedure and a data step.

SAS	R
data vitals_wt; *input subjid wt;* *datalines;* *13 170* *5 160* *7 200* *;* *run;* *proc sort data=vitals_wt out=vitals2;* *by subjid;* *run;* *data vitals2;* *set vitals2;* *wt_kg=wt/2.2;* *unit='kg';* *run;*	*vitals_wt<-tibble(* *subjid=c(13, 5, 7),* *wt=c(170, 160, 200)* *)* *vitals2<-vitals_wt %>%* *# sort data* *arrange(subjid) %>%* *# derive weight in kilograms* *mutate(wt_kg=wt/2.2,unit="kg")*

SUMMARIZING STATISTICS

R is a very powerful software designed for statistical analysis. There are many functions in R similar to PROC SUMMARY or PROC MEANS. One such function in Tidyverse is *summarize*. For example, *summarize (vitals, max(dia))* can be used to calculate the maximum value of the dia variable in the vitals data frame. The *group_by function* must be used in order to use *summarize* on a grouped variable. The following program calculates the maximum value of the vitals.dia variable by subject and stores the results in a new data frame.

SAS	R
proc summary data=vitals_dia noprint ; *var dia;* *output out=max_dia* *(keep=subjid max) max=max;* *by subjid;* *run;*	*max_dia<-group_by(vitals_dia, subjid) %>%* *summarize(max=max(dia))*

subjid	max
1	77
2	82

Table 20. MAX_DIA data frame

COMPARING DATASETS

The *all.equal* function in R can be used to compare data frames.

subjid	age
1	60
2	55

Table 21. DEMO1 data frame

subjid	age
1	61
2	55

Table 22. DEMO2 data frame

SAS	R
proc compare base=demo1 compare=demo2; *run;*	*all.equal(demo1, demo2)*

The result of *all.equal* statement will be displayed in the console window:

```
> all.equal(demo1, demo2)
[1] "Rows in x but not y: 2, 1. Rows in y but not x: 1, 2. "
>
```

As of 2020, there is also a package called *diffdf* that can be used to compare frames.

MACROS

The R equivalent of SAS macros are user-defined functions. The example below demonstrates the use of user-defined R functions to stack two data frames:

SAS (macro)	R (function)
%macro combine_two_datasets(data1, data2); *data alldata;* *set &data1* *&data2;* *run;* *%mend;* *%combine_two_datasets(demo1, demo2);*	*combine_two_datasets<-function (data1, data2)* *{* *alldata<<-rbind(data1, data2)* *}* *combine_two_datasets(demo1, demo2)*

CONCLUSION

This paper outlines how I started to learn to program in R, by comparing what I knew how to program in SAS and discovering how the equivalent can be programmed in R. The next step in my journey of learning R would be to attempt to recreate more complex SAS programs in R. For example to create an R program which re-creates a typical ADaM dataset SAS program. This would necessitate learning more complex data manipulations with R/Tidyverse techniques to create efficient programs.

Both R and SAS programming languages are very powerful tools for statistical data analysis. The Tidyverse package makes R more similar to SAS and easier to use. Despite differences in language syntax, most of the statistical programmer's typical SAS code can be easily converted into R code.

ACKNOWLEDGMENTS

Author would like to thank Michael Rimler, Zuzana Rehm, Jessica Mandel, David Wade, Eric Simms and Andy Nicholls for their help with this paper.

CONTACT INFORMATION

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

Max Cherny

GlaxoSmithKline

Email: max.2.cherny@gsk.com