Paper BB90

A Beginner's Babblefish: Basic Skills for Translation Between R and SAS®

Sarah Woodruff, Westat, Rockville, MD

ABSTRACT

SAS professionals invest time and energy in improving their fluency with the broad range of capabilities SAS software has to offer. However, the computer programming field is not limited to SAS alone and it behooves the professional to be well rounded in his or her skill sets. One of the most interesting contenders in the field of analytics is the open source R software. Due to its range of applications and the fact that it is free, more organizations are considering how to incorporate it into their operations and many people are already seeing its use incorporated into project requirements. As such, it is now common to need to move code between R and SAS, a process which is not inherently seamless.

This paper serves as a basic tutorial on some of the most critical functions in R and shows their parallel in SAS to aid in the translation process between the two software packages. A brief history of R is covered followed by information on the basic structure and syntax of the language. This is followed by the foundational skill involved in importing data and establishing R data sets. Next, some common reporting and graphing strategies are explored with additional coverage on creating data sets that can be saved, as well as how to export files in various formats. By having the R and SAS code together in the same place, this tutorial serves as a reference that a beginner can follow to gain confidence and familiarity when moving between the two.

Keywords: R, SYNTAX, IMPORT, GRAPH, REPORT, EXPORT

INTRODUCTION

Why should a SAS programmer be concerned with the R programming language? The capabilities that R incorporates are covered and exceeded by SAS products. However, the fact that R can be acquired at no cost and can be added to by its user community confers real advantage. Given the overlap in functionality, it behooves SAS programmers to be familiar with R as its popularity is rising. While its limitations do not put it in a position to displace SAS, programmers are likely to be called upon to work with both or may inherit projects that began in R and will need to repurpose those tasks into SAS.

HISTORY OF R

The development of R began at the University of Auckland in the early 1990s with Ross Ihaka and Robert Gentleman. Both were involved in teaching statistical computing and looking for improved tools for the Macintosh computers being used by their students. While they had an appreciation for the capabilities of the S programming language that had already been developed, they also saw its limitations and were looking to expand its scope. This goal of improving on S spawned the writing of an interpreter program in 1993. That code consisted of about 1000 lines written in C and ended up becoming the foundation of R.

By 1995, R was robust enough to be of interest to a larger community of users and in June of that year, it was placed on the FTP servers of the GNU Project as part of the Free Software Foundation. A limited mailing list was established for users to discuss R and to report on bugs, but the use of this communication quickly outstripped the University of Auckland's ability to support it. In March of 1996, the mailing lists were moved to ETH Zurich in Switzerland at the behest of a statistical computing scientist, Martin Mächler; his interest in R was growing and he wanted to play a larger role in supporting its ongoing development. Being able to expand user interaction and contribution meant that the progress of R could be greatly accelerated.

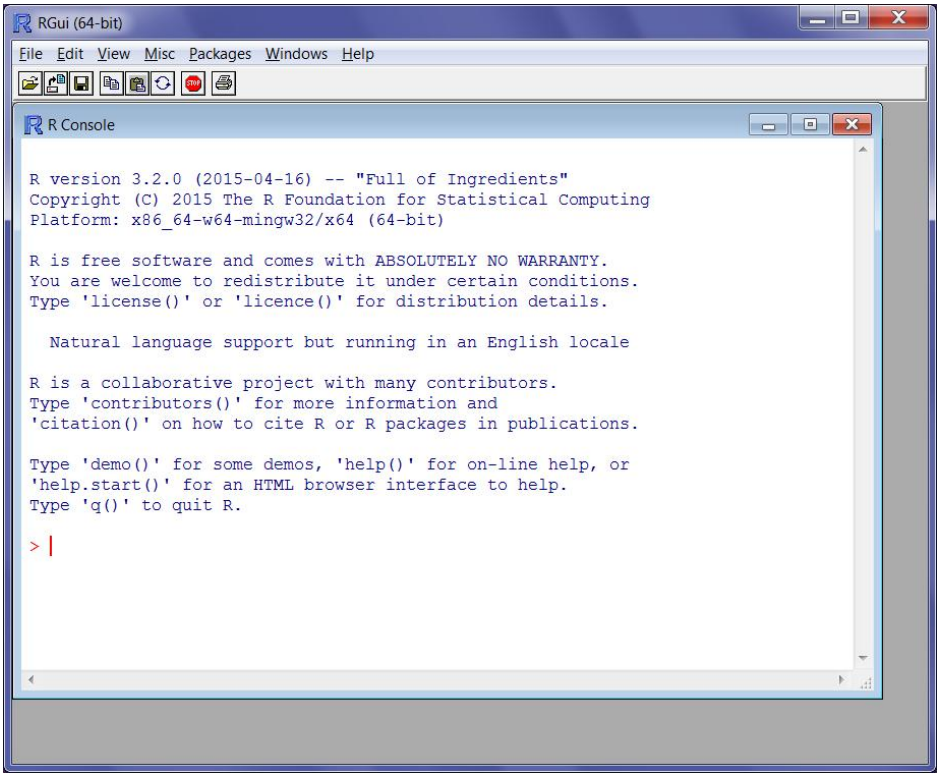
Making such steady progress was possible because the user community was not only discussing R, but also making suggestions for improvements, providing bug fixes and even supplying code for new or expanded features. These combined efforts meant that by February of 2000, R was considered to be ready for full production use and became widely available. In 2004 the first useR! Conference was held and continues to be an annual event.

Online, R has coalesced at CRAN (the Comprehensive R Archive Network). At the site, current versions of the package for Linux, Mac and Windows are available along with prior iterations, user documentation and compiled

manuals. The site serves as the formal starting point for any user of R; from there, many other online forums exist, some exclusively devoted to R, such as R-Forge, and others with certain areas devoted to R, such as StackOverflow. The DIY nature of R's development means that support for users is less formalized than with other commercial software, but remains readily available.

BASICS OF R SYNTAX AND OBJECTS

R is an interpreted language that runs interactively so code is converted to machine instructions as it is encountered. This can cause R to run more slowly in comparison to other languages; however, the flexibility that this provides is a practical advantage. Invoking a session of R yields an opening console in which each interactive line begins with >



Variables in R are referred to as objects. They can be established as numeric or character and can be invoked outside of procedures or creations of format data sets. Numeric variables are set as follows:

```
> NUMBERX = 6   # assign value
> NUMBERX       # call object to display value
[1] 6
```

The first thing of note in this simple example is that variables names are case sensitive, an important detail for those who are accustomed to the lack of case-sensitivity in SAS. If you were to call the variable NUMBERX with a different mix of cases, R gives an error message that it cannot be found:

```
> NumberX       # call object with different use of case
Error: object 'NumberX' not found
```

The second thing of note is how comments are being included. R allows for commenting by including a hash, #, at the start of each line of the comment. Unlike SAS, R does not currently have a native syntax for creating comment blocks or for commenting out multiple lines at one time. If you are working in a text editor that allows for column block selection, the insertion of multiple # is facilitated. Alternatively, many of the editors specific to R have short cut commands in which a block of text can be selected and then the command can be run to comment out the selected text.

Character variables require, well, a few more characters in order to be set:

```
> CharacterX = "SESUG"  # assign value
> CharacterX            # call object to display value
[1] "SESUG"
```

Note that character values need to be enclosed in quote marks in order to be assigned; double or single quote marks can be used to achieve this, but the case-sensitive properties of the variable are the same as if it were numeric. Setting a character string as the value without use of quote marks will lead to an "object not found" error.

Converting between character and numeric values can be done using basic functions, as.numeric and as.character, as follows:

```
> NumberCharacter = "2015"        # establishing numeric value as a character string
> NumberCharacter                 # call object to display value, quote marks indicate character class
[1] "2015"
```

Once a character value has been established, it can then be converted to a numeric using as.numeric.

```
> CharacterNumber = as.numeric(NumberCharacter)  # convert character value to numeric
> CharacterNumber                                # call object to display value, lack of quote marks indicate numeric class
[1] 2015
```

A numeric value can be converted in similar fashion to a character value using as.character.

```
> CharacterReconvert = as.character(CharacterNumber)  # convert numeric value to character
> CharacterReconvert                                  # call object to display value, quote marks indicate character class
[1] "2015"
```

Though the display of a variable value gives visual indication of the data type, this can also be checked directly using a call for class on the variable:

```
> class(NumberCharacter)          # check type of object and display result
[1] "character"
```

(Data) Frame, (Data) Set….(and How They) Match in R

Creating Data

When creating a data set in SAS, an INPUT statement can generate variables and then DATALINES can establish the values of those variables. Here is a simple example:

```
DATA LabResult ;
   INPUT PID Sex $ Test1 Test2 $ ;
   DATALINES;
10045  F  156 POS
10048  M  35 NEG
<more lines of data>
;
RUN;
```

The parallel process in R generates the values of objects as arrays and combines them into a data frame, R's equivalent of SAS' data set. The arrays are used to establish the data type as well as the values. Here is the same sample data generated in R:

```
> PID = c(10045, 10048)              # set values for object PID
> Sex = c("F", "M")                  # set values for object Sex; character data is indicated via use of quote marks
> Test1 = c(156, 35)                 # set values for object Test1
> Test2 = c("POS", "NEG")            # set values for object Test2; character data is indicated via use of quote marks
> LabResult = data.frame(PID, Sex, Test1, Test2)  # generate data frame called LabResult to combine all four objects
> LabResult                          # call data frame LabResult to be printed
   PID Sex Test1 Test2
1 10045   F   156   POS
2 10048   M    35   NEG
```

Saving Data

To save a data frame to a particular location, invoke the SAVE command as follows:

```
> save(LabResult,file="C:/Users/Woodruff_S/LabResult_Initial.Rdata")
# saves data frame as LabResult_Initial in specified location
```

Saving all data frames in a given session follows a similar command path, only with the name of a particular data frame left out of the command:

```
> save(file="C:/Users/Woodruff_S/LabResult_Complete.Rdata")
# saves all data frames from a session at specified location
```

Reading Data

When looking to read an established SAS data set into a program, SAS needs an LIBNAME to establish location and from there a DATA step or PROC can utilize the data. Here is a simple example:

```
LIBNAME userdata "C:\Users\Woodruff_S" ; /* establishes data location */

DATA LabResult_Female ;  /* outputs requested records */
  SET userdata.LabResult ;  /* accesses complete data set */
  WHERE Sex in ('F') ;  /* specifies conditions to limit records */
RUN ;
```

In R, it is the LOAD command which brings in data based on an indicated network or computer location. Utilizing the above sample data, a single data frame could be accessed and the SUBSET command can be used to create the same selection of records as in the prior example in SAS:

```
> load("c:/Users/Woodruff_S/LabResult_Initial.Rdata")# accesses single saved data frame

> LabResult_Female <- subset(LabResult, Sex == "F")
# specifies condition to limit records and outputs new data set with requested records
```

Or if all data frames from a given session have been saved, then access to the complete set of those data frames can be established with similar syntax and functioning much like a LIBNAME statement in SAS:

```
> load("C:/Users/Woodruff_S/LabResult_Complete.Rdata")      # accesses complete set of data frames from session
```

IMPORTING DATA

The flexibility to import data from a variety of sources is part of what makes SAS so capable and user-friendly. Understanding the parallel syntax in R is an important step in making the software useful in routine work life.

Excel

The LIBNAME engine can read Excel files directly into SAS, though it is typically good practice to include the EXCEL keyword in the libname statement:

```
LIBNAME LabResult EXCEL "C:\Users\Woodruff_S\ExternalData.xlsx" ;
```

However, using PROC IMPORT in SAS provides more nuanced control over how the data arrive and makes it easier to manipulate the information once it is in place. This example gives basic descriptions about what each of the proc components can do, but more details are available in the online documentation.

```
PROC IMPORT OUT = LabResult
      DATAFILE = "C:\Users\Woodruff_S\ExternalData.xlsx "
      DBMS = EXCEL REPLACE /* use of REPLACE is optional */ ;
      RANGE = "Sheet1$" /* use appropriate sheet name, leave $ in place */ ;
      GETNAMES = YES /* use values in first line as variable name */ ;
      MIXED = YES /* evaluate data type -- default is NO */ ;
      SCANTEXT = YES /* only relevant for character data */ ;
      USEDATE = YES /* default format is DATE9. */ ;
      SCANTIME=YES ;
RUN ;
```

Just as in SAS, R has multiple ways in which Excel files can be accessed and converted into usable data frames. However, the most useful approaches typically requiring accessing additional packages to make this work. For instance, the XLConnect package is Java-based and functions well across platforms. Invoking it allows the user to then read in a complete workbook using the loadWorkbook function and then to specify a particular sheet to generate an individual data frame. Calling the individual sheet also allows for the first row to specify the names of the objects (or variables) by using the header=TRUE option as part of the statement.

```
> library(XLConnect)            # load XLConnect package

> ExternalLabResult = loadWorkbook("C:/Users/Woodruff_S/ExternalData.xlsx")
# specify location of desired workbook

> LabResult = readWorksheet(ExternalLabResult, sheet="Sheet1", header=TRUE)
# create separate data frame out of specific sheet where objects are named according to first row of sheet
```

Another option is the use of the GData package. GData requires the use of Perl libraries and having access to those may be more work for Windows users, though they are typically already present on Macs and for those using Linux. A given data frame can then be generated with one read.xls statement that allows the sheet to be specified while also indicating whether the header row contains the object names.

```
> library(gdata)    # load GData package

> LabResult = read.xls ("C:/Users/Woodruff_S/ExternalData.xlsx "), sheet = 1, header = TRUE)
# specify location of complete workbook, indicate desired sheet for data frame, set headers as object names
```

CSV

Bringing data into SAS from CSV is very similar to bringing in data from Excel and similar LIBNAME engine or PROC IMPORT tools can be utilized.

In R, reading in a CSV file is more straightforward than reading in Excel data. Users should be clear that the expectation when reading a CSV file into R is that the first row contains the names of the objects to be established in the data frame. Such an import would look like this:

```
> LabResult = read.csv("C:/Users/Woodruff_S/ExternalData.csv")  # read CSV file and create data frame
```

SPSS

Importing data from SPSS into SAS utilizes a simpler variation on PROC IMPORT where the file in SPSS form is specified and the output data set is established.

```
PROC IMPORT datafile = "C:\Users\Woodruff_S\ExternalData.sav"
            out = LabResult ;
RUN ;
```

For R, this process requires an additional step to be taken in the native SPSS environment. The data must be saved in transport format which means that the file should have a .POR extension. Once in this format, the file can then be read into R, but like handling Excel data, this importation requires an additional package, HMISC. The value labels can be turned into the object names by utilizing the use.value.labels option at the end of the statement.

```
> library(Hmisc)   # load HMISC package

> LabResult <- spss.get("C:/Users/Woodruff_S/ExternalData.por", use.value.labels=TRUE)
 # read SPSS file and create data frame where value labels become objects
```

Between SAS and R

Importing data from R into SAS requires an additional step, much like the preparation to bring an SPSS file into R. To accomplish this, the data frame in R needs to be written to an output file with the .DBF extension. This requires the use of the FOREIGN package to also enable the SAS-writing functionality of the write.dbf statement.

```
> library(foreign)  # load FOREIGN package

> write.dbf(LabResult, "C:/Users/Woodruff_S/ExternalData.dbf")
# specifies data frame to be written and names file with .DBF extension
```

Then SAS' PROC IMPORT can take over and the task looks very much like bringing data in from SPSS.

```
PROC IMPORT datafile = "C:\Users\Woodruff_S\ExternalData.dbf"
      out = LabResult DBMS = dbf ;
RUN ;
```

Similar minor acrobatics are necessary to move data from SAS to R. First, in SAS, the data set to be moved must be prepared in the transport format and thus will have a .XPT extension.

```
LIBNAME userdata xport "C:\Users\Woodruff_S\ExternalData.xpt" ;
/* establishes data export location and format */

DATA userdata.LabResult ;
      SET LabResult ;  /* specifies data set to be written out in transport format */
RUN ;
```

Then pulling the .XPT file into R once again requires the use of the HMISC package.

```
> library(Hmisc)            # load HMISC package

> LabResult <- sasxport.get("C:/Users/Woodruff_S/ExternalData.xpt")  # read XPT file and create data frame
```

REPORTING ON DATA

Once data are moved into R or SAS, the goal is then to be able to analyze it. Often, an analysis begins with an exploration of descriptive statistics. In R, a single line of code can start this process by yielding the means, medians, 25th quartiles, 75th quartiles, minimums and maximums for numeric objects in a given data frame as well as

frequencies for values of character objects. Running the SUMMARY command on the sample data set LabResult from above yield the following results:

```
> summary(LabResult)

      PID        Sex       Test1        Test2
 Min.   :10045   F:1   Min.   : 35.00   NEG:1
 1st Qu.:10046   M:1   1st Qu.: 65.25   POS:1
 Median :10046         Median : 95.50
 Mean   :10046         Mean   : 95.50
 3rd Qu.:10047         3rd Qu.:125.75
 Max.   :10048         Max.   :156.00
>
```

Note that PID was a numeric object in this case, but that obtaining descriptive statistics about a subject identified is typically not meaningful. Though R has strong analytic tools, there is still no substitute for understanding the data being examined by those tools.

Similar output is possible through a variety of means in SAS, but the most straightforward is likely to be PROC MEANS. The desired statistics needs to be specified in the main statement of the proc, but it will only analyze numeric variables and will not give frequencies for the values of those of type character.

```
PROC MEANS data = LabResult min P25 median mean P75 max ;
/* specify list of descriptive statistics to be generated */
RUN ;
```

The standard HTML output would produce a table in the following format:

The MEANS Procedure

Variable	Minimum	25th Pctl	Median	Mean	75th Pctl	Maximum
PID	10045.00	10045.00	10046.50	10046.50	10048.00	10048.00
Test1	35.0000000	35.0000000	95.5000000	95.5000000	156.0000000	156.0000000

It is easy in SAS to restrict PROC MEANS to only numeric variables where the descriptive statistics are meaningful by use of a VAR statement. In this next example, the analysis was limited to only the variable TEST1 and thus PID was not evaluated.

```
PROC MEANS data = LabResult min P25 median mean P75 max ;
        var Test1 ;
RUN ;
```

The output remains in the same format, but only has the results for TEST1.

The MEANS Procedure

Analysis Variable : Test1

Minimum	25th Pctl	Median	Mean	75th Pctl	Maximum
35.0000000	35.0000000	95.5000000	95.5000000	156.0000000	156.0000000

Getting more granular details, including the values of extreme observations can be done using the HMISC package in R combined with the describe function. This will yield the number of observations, the number of unique values, the mean of the values, the 5th, 10th, 25th, 50th, 75th, 90th and 95th percentiles as well as the five lowest and five highest observations.

```
> library(Hmisc)            # invoke Hmisc package

> describe(LabResult_T1) # run DESCRIBE command to yield more granular percentiles and extreme observations
# FOR THIS EXAMPLE, LABRESULT_T1 REPRESENTS A TOTAL OF 10 VALUES FOR TEST1
```

Running this function yields the following output:

```
LabResult_T1

 1  Variables      10  Observations
---------------------------------------------------------------------------------------------------
Test1
      n missing  unique    Info    Mean     .05     .10     .25     .50     .75     .90     .95
     10       0      10       1    89.2    5.15    8.30   21.50   59.00  141.00  185.10  235.05

           2  9 17 35 53 65 96 156 174 285
Frequency  1  1  1  1  1  1  1   1   1   1
%         10 10 10 10 10 10 10  10  10  10
---------------------------------------------------------------------------------------------------
>
```

Particularly if the focus is on getting extreme values, the use of PROC UNIVARIATE is an easy way to yield this type of information in SAS. The additional percentiles are available in the output of this proc too along with more in-depth measures, like skewness and Kurtosis

```
PROC UNIVARIATE data = LabResult_T1 ;
      var Test1 ; /* limiting analysis to only relevant variable */
RUN ;
```

The following standard HTML output is produced by this proc:

The UNIVARIATE Procedure
Variable: Test1

Moments

N	10	**Sum Weights**	10
Mean	89.2	**Sum Observations**	892
Std Deviation	90.7497169	**Variance**	8235.51111
Skewness	1.23369606	**Kurtosis**	1.06191249
Uncorrected SS	153686	**Corrected SS**	74119.6
Coeff Variation	101.737351	**Std Error Mean**	28.6975802

Basic Statistical Measures

Location		Variability	
Mean	89.20000	**Std Deviation**	90.74972
Median	59.00000	**Variance**	8236
Mode	.	**Range**	283.00000
		Interquartile Range	139.00000

Tests for Location: Mu0=0

Test	Statistic	p Value

Tests for Location: Mu0=0

Test		Statistic		p Value
Student's t	**t**	3.108276	**Pr > |t|**	0.0126
Sign	**M**	5	**Pr >= |M|**	0.0020
Signed Rank	**S**	27.5	**Pr >= |S|**	0.0020

Quantiles (Definition 5)

Quantile	Estimate
100% Max	285.0
99%	285.0
95%	285.0
90%	229.5
75% Q3	156.0
50% Median	59.0
25% Q1	17.0
10%	5.5
5%	2.0
1%	2.0
0% Min	2.0

Extreme Observations

Lowest		Highest	
Value	**Obs**	**Value**	**Obs**
2	10	65	5
9	6	96	8
17	3	156	1
35	2	174	7
53	9	285	4

In order to derive some of the more in-depth statistics, like Kurtosis, R requires the use of the PSYCH package, but can then once again run the DESCRIBE function to derive many of the desired values.

```
> library(psych)              # invoke PSYCH package

> describe(LabResult_T1)  # run DESCRIBE command to yield more in-depth statistics
```

The output from R provides the following:

```
vars  n mean    sd median trimmed  mad min max range skew kurtosis   se
Test1    1 10 89.2 90.75     59   75.62 68.2   2 285   283 0.89    -0.53 28.7
```

GRAPHING DATA

A recognized strength of R is its ability to easily and interactively create graphs which can then be extensively customized and output in a variety of formats. Like SAS, R has some built-in data and one of these frames, MTCARS, is used here for demonstration purposes in both types of software. MTCARS contains information on vehicles which have been part of Motor Trend's road tests and includes metrics such as number of cylinders, gross horsepower, weight, miles per gallon, type of transmission, amongst others.

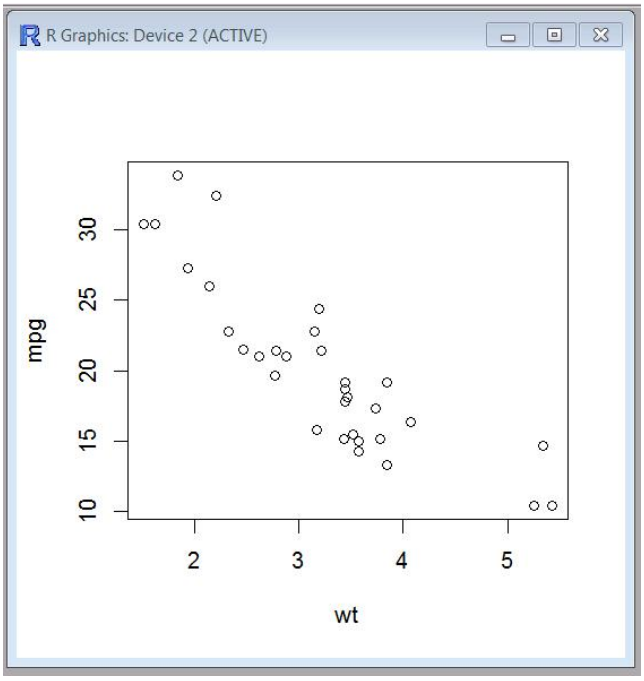
The first step for graphing in R is to run the ATTACH function:

```
> attach(mtcars)
```

Then the PLOT function is invoked with the variables that are to appear on the graph. For this example, weight and miles per gallon are to be included.

```
> plot(wt, mpg)
```

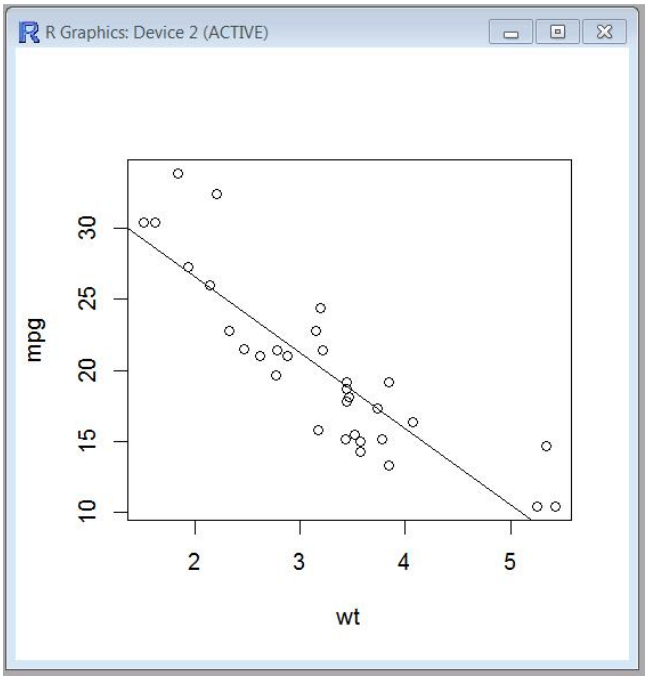
This automatically opens an additional window in which the graph is already being generated and can be seen by the programmer. With only two lines of code, the graph already looks like this:



An additional line of code will then add a line of regression for miles per gallon on weight

```
> abline(lm(mpg~wt))
```

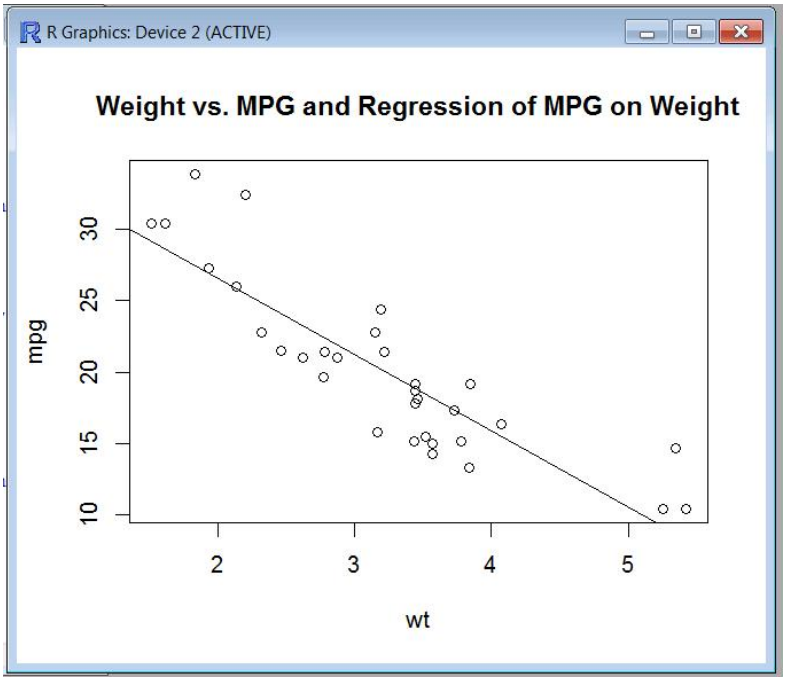
Which then interactively adds the line for the programmer to see:



Adding customization, like a descriptive title, also requires a single command and is then also immediately visible. Running this:

```
> title("Weight vs. MPG and Regression of MPG on Weight")
```

Yields this further improved graph:



Saving graphs is equally straightforward. If the graphing window is selected, it can be done from File -> Save As

where the options are then Metafile, Postscript, PDF, PNG, BMP, TIFF and JPEG, the latter giving the option for three different levels of image quality. Otherwise, graphs can be saved programmatically with an additional line of code that specifies not only the format of the file, but also its name and location. For example, if this graph were to be saved as a PDF, the code could be:

```
> pdf("C:/Users/Woodruff_S/ExternalData/MTCARS_GRAPH.pdf")
```

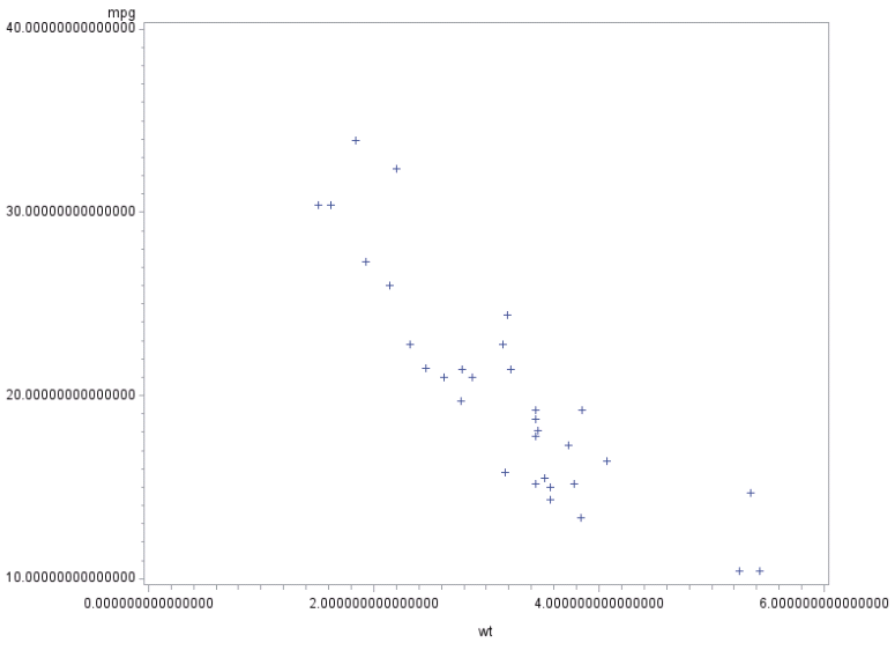
Other file formats follow a similar pattern as indicated:

- win.metafile("<location and name>") for Windows Metafile
- postscript("<location and name>") for Postscript
- png("<location and name>") for PNG
- bmp("<location and name>") for Bitmap
- jpeg("<location and name>") for JPEG
- tiff("<location and name>") for TIFF

While graphing capabilities are very strong in SAS, determining how to carry them out can be much more complicated. PROC GPLOT is likely to be the easiest starting point for graphing, but even so it requires a greater understanding of syntax to produce a basic plot.

```
PROC GPLOT data = mtcars ; /* specifies data set */
   plot mpg*wt /* specifies variables */
       / haxis=axis1 vaxis=axis2 /* provides for labels on axes */;
RUN ;
QUIT ;
```

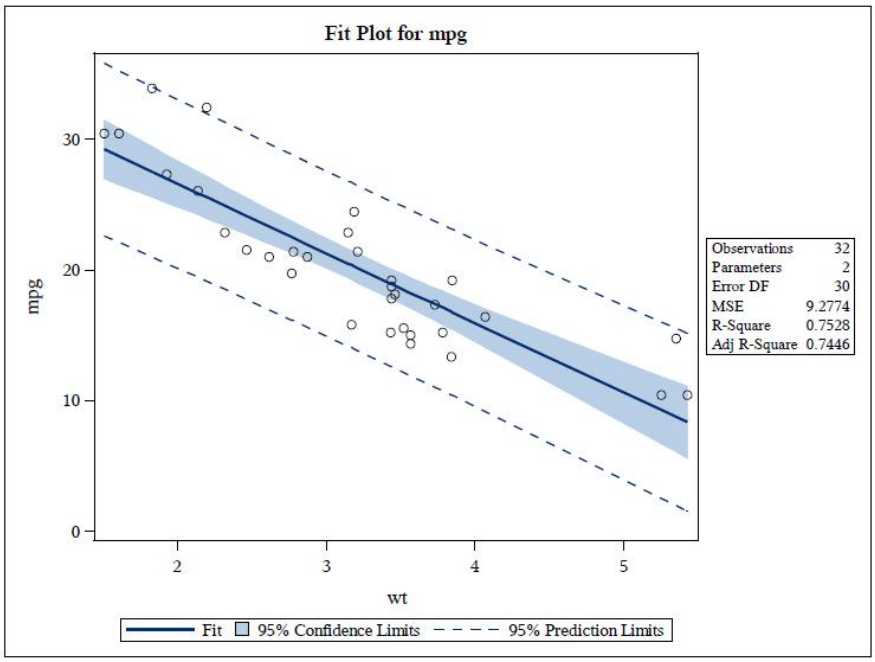
Running the above PROC GPLOT code produces the following graph:



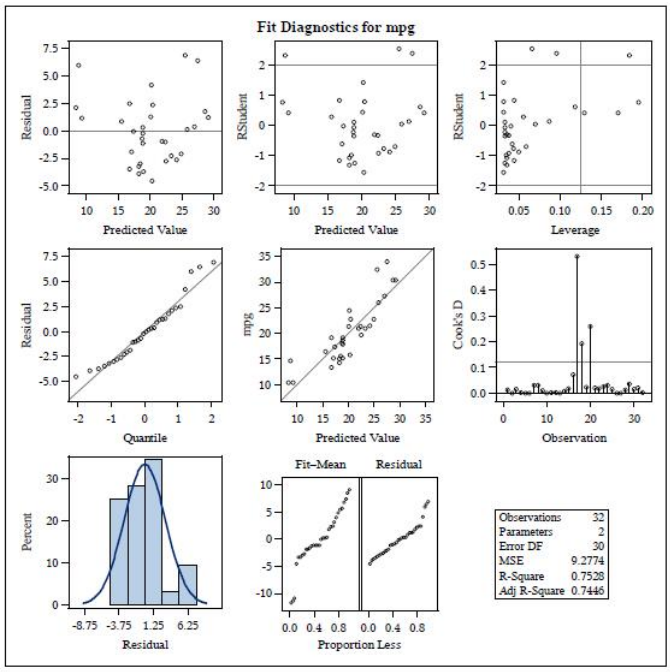
To easily incorporate a line of regression in a SAS graph, PROC REG is simpler than sticking with PROC GPLOT, but a programmer must be familiar with both to know to explore the options. Titles can be included as an inherent part of either of these procs and will lead to constructive labels on the graph

```
PROC REG data=mtcars;
     model mpg = wt ; /* specifies variables for plot and regression relationship */
     title "Weight vs. MPG and Regression of MPG on Weight" ;
RUN ;
QUIT ;
```

The use of PROC REG will yield a bubble scatter plot with the fitted line of regression, but the default also includes confidence and prediction limits. While such additional information may often be desirable, it then requires more insight into the specifics of the proc to get a graph that is simpler. The potential advantage to R is that a programmer can build in pieces of the graph as they are required or, if in the position to inherit a program, can see how the graph came to be in it final incarnation.



Outside of this particular fit plot, PROC REG also yields analysis of variance, parameter estimates, a plot of the residuals for the dependent variable in the regression relationship and a bevy of other graphs demonstrating various measures on the quality of fit of the dependent variable. As can be seen in the following sample, this information is potentially useful, but may also be overwhelming and requires additional work to have it not appear in output if it is not desired:



EXPORTING DATA

Being able to get data back out of both SAS and R in multiple formats is just as important as being able to get it into the various software packages in the first place. Having well-rounded export skills makes a programmer more flexible, particularly when a project requires the use of multiple types of software through its trajectory.

Excel

The LIBNAME engine can be used for exporting SAS data sets to Excel almost as easily as it can be used to import them. The LIBNAME engine can read Excel files directly into SAS, though it is typically good practice to include the EXCEL keyword in the libname statement:

```
LIBNAME userdata "C:\Users\Woodruff_S\ExternalData.xlsx";
/* establishes data export location and format */

DATA userdata.LabResult ;
      set LabResult ;  /* specifies data set to be written out in Excel format */
RUN ;
```

Just as with importing data, using PROC EXPORT can provide more flexibility and nuance to the creation of the output Excel file. Though the SHEET option is technically not required, it should be viewed as mandatory in the creation of meaningful spreadsheet output.

```
PROC EXPORT DATA = LabResult
            OUTFILE = "C:\Users\Woodruff_S\ExternalData.xlsx"
            /* extensions like XLS or XLSM can also be used */
            DBMS = EXCEL LABEL replace ;
            /* use REPLACE with caution */
            SHEET = "Initial";
            NEWFILE = YES ;
RUN ;
```

In R, writing out to Excel requires the XLSReadWrite package, but once that is in place, the code can be even more straightforward than when importing.

```
> library(xlsx)      # load the functionality of the XLSReadWrite package

> write.xlsx(LabResult, "C:/Users/Woodruff_S/ExternalData.xlsx")
# specify data frame to be exported along with the location and name of the Excel file being created
```

CSV

As with importing data, similar LIBNAME engine constructs and the more detail-conscious approaches of PROC EXPORT can be used to take data from SAS to out to a CSV file.

Writing out to CSV is just as straightforward as it was reading into one. The data frame to be read out needs to be specified as does the location and name of the CSV being created.

```
> write.csv(LabResult, "C:/Users/Woodruff_S/ExternalData.csv") # read data frame and create CSV
```

SPSS

Exporting data from SAS into SPSS looks much like importing it in that PROC EXPORT needs to specify the data set to be output the SPSS-specific .DTA format.

```
PROC EXPORT data = LabResult
                    outfile= "C:\Users\Woodruff_S\ExternalData.dta" ;
RUN ;
```

Getting R to export data into SPSS also requires the software to write its own toolkit to take with it when SPSS will then read in the data. The FOREIGN package must once again be invoked and the data frame must be generated as a TXT file. A .SPS file of the same name is also generated and the package (in this case SPSS) is also specified. This generates not only the data that will be read in by SPSS, but also the directions that the software will need in order to do so.

```
> library(foreign)  # load FOREIGN package

> write.foreign
(LabResult, " C:/Users/Woodruff_S/ExternalData.txt", "C:/Users/Woodruff_S/ExternalData.sps",   package="SPSS")
# designates data frame to be sent to SPSS, the SPSS program that will read data and the particular package
```

Between SAS and R

The export of data between these two software packages was addressed in the import section above because to import into either, a programmer inherently needs to export in a format that can be read by the other.

CONCLUSION

The diversity and strength of SAS' capabilities serve as the foundation of professional working lives, but the scope of available analytical software continues to expand, with R being a serious contender in the field. R's origins as a user-developed system that was then opened up to the wider community for both further expansion and maintenance makes it a profound and unique experiment. From the hobbyist up through the professional developer, R's lack of hierarchy gives all who wish to be involved in its infrastructure the chance to do so.

SAS programmers should consider expanding their skill sets to include such open-sourced technologies. Certain projects already require tools that are available to all, regardless of budget, and as the user base for R continues to broaden, it is likely that other projects will want to move as seamlessly as possible between softwares like SAS and R. This paper was designed to provide context for any SAS programmer looking to deconstruct R code they may have encountered or to serve as an introduction to this software which can then be enhance through experimentation and further study. The references here represent a tiny fraction of what is available, but can serve as a strong starting point for anyone who is looking to bridge the translation gap.

A NOTE ON PACAKGES

All available R packages, including those that have been archived or do not currently have someone actively maintaining them can be found in the R package repository (http://cran.r-project.org/web/packages/). This location also includes basic directions on installation and guidance for more in-depth help.

REFERENCES

- CRAN. "The Comprehensive R Archive Network." Available at http://cran.r-project.org/.
- Gillespie, C.S. "Installing R Packages." R-bloggers. 2010. Available at http://www.r-bloggers.com/installing-r-packages/.
- GM-RAM. "Exporting Data From R to Text Files." Statistical Modeling with R. 2015. Available at http://www.wekaleamstudios.co.uk/posts/exporting-data-from-r-to-text-files/.
- Heaton, Ed. "So, Your Data are in Excel!" Proceedings of SUGI 31. 2006. Available at http://www2.sas.com/proceedings/sugi31/020-31.pdf.
- IDRE. "How can I subset a data set?". Frequently Asked Questions about R. 2015. Available at http://www.ats.ucla.edu/stat/r/faq/subset_R.htm.
- Ihaka, Ross. "Genesis." R: Past and Future History. 1998. Available at http://cran.r-project.org/doc/html/interface98-paper/paper_1.html.
- Kabacoff, Robert. "Axes and Text." Quick-R. 2014. Available at http://www.statmethods.net/advgraphs/axes.html.
- Kabacoff, Robert. "Creating A Graph" Quick-R. 2014. Available at http://www.statmethods.net/graphs/creating.html.
- Kabacoff, Robert. "Descriptive Statistics." Quick-R. 2014. Available at http://www.statmethods.net/stats/descriptives.html.
- Kabacoff, Robert. "Exporting Data." Quick-R. 2014. Available at http://www.statmethods.net/input/exportingdata.html.
- Kabacoff, Robert. "Importing Data." Quick-R. 2014. Available at http://www.statmethods.net/input/importingdata.html.
- Kleinman, Ken. "Get Data From R Into SAS." R-Bloggers. 2009. Available at http://www.r-bloggers.com/example-7-10-get-data-from-r-into-sas/.
- Muenchen, Bob. "Data Export" r4stats. Available at http://r4stats.com/examples/data-export/.
- Ohio State Univerisity. "Understanding Data Frames." Available at http://facweb.knowlton.ohio-state.edu/pviton/courses2/crp87105/data-frame.html.
- Pennsylvania State University. "Numeric Vectors." R Documentation. Available at http://www.astrostatistics.psu.edu/su07/R/html/base/html/numeric.html.
- SAS. "IMPORT Procedure." Base SAS 9.3 Procedures Guide, Second Edition. 2014. Available at http://support.sas.com/documentation/cdl/en/proc/65145/HTML/default/viewer.htm#n18jyszn33umngn14czw2qfw7thc.htm.
- SAS. "Scatter plot of groups with an overall regression line." Knowledge Base/Samples & SAS Notes. Available at http://support.sas.com/kb/42/864.html.
- Sommacal, Nicole. "Read Excel Files from R." R-bloggers. 2013. Available at http://www.r-bloggers.com/read-excel-files-from-r/.

- Winters, Ralph. "Excellent Ways of Exporting SAS Data to Excel." Proceedings of NESUG 17. 2004. Available at http://www.nesug.org/proceedings/nesug04/io/io09.pdf.
- Yau, Chi. "Character." R Tutorial. 2015. Available at http://www.r-tutor.com/r-introduction/basic-data-types/character.
- Yau, Chi. "Data Frame." R Tutorial. 2015. Available at http://www.r-tutor.com/r-introduction/data-frame.
- Yau, Chi. "Data Import." R Tutorial. 2015. Available at http://www.r-tutor.com/r-introduction/data-frame/data-import.

ACKNOWLEDGMENTS

I would like to thank Rick Mitchell for encouraging me to continue to write and being a SAS conference inspiration. I would like to thank Michael Raithel at Westat for his patience and editorial support. I would like to thank Westat for their institutional support of my participation in both local and regional SAS users' groups.

CONTACT INFORMATION

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

Sarah Woodruff
Westat
1600 Research Boulevard, WB 401
Rockville, MD 20850
240-314-7562
SarahWoodruff@westat.com

DISCLAIMERS

The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of Westat.