Paper 183-2019

Python and R made easy for the SAS® Programmer

Janet J. Li, Pfizer Inc.; Varaprasad Ilapogu, Ephicacy Consulting Group

ABSTRACT

Many of the day-to-day tasks and responsibilities of the statistical programmer of a pharmaceutical research and development group or contract research organization (CRO) involved include importing/exporting data, deriving variables and creating analysis data sets, and creating clinical study report (CSR) materials such as tables, listings, and figures (TLFs). These outputs are used for submission to regulatory agencies and are generally programmed in SAS®. There is a growing movement and acceptance towards using Python and R in the Clinical SAS® programming world. In our paper, we explore tips and tricks on how to use Python and R in conjunction with SAS® to more effectively and efficiently deliver statistical programming outputs. We intend to provide examples of common SAS® procedures and syntax that are used in the creation of analysis datasets and TLFs and translate them to Python and R to help the beginner Python and/or R programmer familiarize themselves with the alternative way of programming these statistical outputs.

INTRODUCTION

Many of the day-to-day tasks and responsibilities of the statistical programmer in the pharmaceutical industry include importing/exporting data, deriving variables and creating analysis data sets, and creating clinical study report (CSR) materials such as tables, listings, and figures (TLFs). These outputs are used for submission to regulatory agencies and are generally programmed in SAS®.

There is a growing movement to integrate artificial intelligence (AI) capabilities, such as machine learning and natural language processing, into the clinical programming world. SAS® programmers are encouraged to pursue relevant education and training to adapt to the anticipated evolution of the clinical SAS® world. Two very useful tools for SAS® programmers wishing to expand their AI knowledge are Python and R. Python is a high-level, open source general-purpose language; R software is a free open source software for statistical computing and graphics. Since clinical SAS® programmers come from diverse backgrounds and may not necessarily have a background in computer science, learning a new language may seem like a difficult task.

In this paper, we hope to draw parallels in the syntax of the languages to help the clinical SAS® programmer familiarize themselves with an alternative way of programming and allay their concerns related to learning a new language.

COMMONLY USED PROCEDURES AND SYTNAX

We provide examples of SAS® procedures and syntax commonly used by the clinical programmer and the equivalent Python and R code below.

Most of the functions within Python and R referenced below are available as part of the respective software's base libraries. For Python, some functions referenced are found within the *Pandas* library, which is a library for data manipulation and analysis. To use functions within this library, the user will need to import *Pandas* (`import pandas as pd`).

BASIC RELATIONAL TABLE

SAS® emphasizes the procedure or program flow, in which the emphasis is on the order of the steps that the program performs. In contrast, Python and R are both object-oriented programming (OOP) languages, where the emphasis is placed on the data in the program and on the operations that are performed on that data. A relationship table between the different environments is provided below.

SAS	Python	R

`SAS Dataset`	`Series`
	`Dataframe`
		`Array`
		`Dataframe`
		`Matrix`

`Observation`	`Row`	`Row`
`Variable`	`Column`	`Column`

COMMENTS AND DOCUMENTATION

Documentation of programming is crucial for the clinical SAS programmer. Comments in SAS start with an asterisk, *, and end with a semi-colon, ;. Alternatively, the SAS® programmer can start embed comments within a /* and a */. Comments in Python and R start with the hash character, # , and extend to the end of the physical line. A comment may appear at the start of a line or following whitespace or code, but not within a string literal.

SAS	Python	R

`*this is a comment;`
`/*this is a comment*/`
	`#this is a comment`	`#this is a comment`

IMPORTING DATA

The clinical programmer receives raw data in various file formats and imports it into SAS®. The raw data source can be a clinical data management system, external laboratory data, data found in Microsoft Office files, or CDISC model data. PROC IMPORT can be used for this step in SAS®. *Pandas's read_csv()* function can be used in Python, whereas *read.table()* function within R base library can be used to import the file. The below examples consider a scenario in which the raw data (*mydata.csv*) is in a comma-separated values (CSV) format. In all three examples the raw data file will be read into the respective system as a dataset or as an object named *d2*. This *d2* dataset is assigned with the <*out* = > option with SAS®, with the equal sign (*d2* =) in Python, and with a left arrow (*d2* <-) in R.

SAS	**Python**	**R**

`proc import datafile =`
`   "C:\mydata.csv"`
`   out=d2`
`   dbms=csv replace;`
`run;`
	`import pandas as pd`
	
	`d2 =`
	`pd.read_csv("C:\mydata.csv")`
		`d2 <- read.table(`
		`   "C:\mydata.csv",`
		`   header=TRUE,`
		`   sep=",",`
		`   row.names="id")`

PRINTING DATA

PROC PRINT prints a listing of the values of some or all the variables in a SAS dataset. The *var* statement within PROC PRINT allows the user to assign the set of variables to print. In the below example, variable *var1* is printed. Similarly, the *print()* functions in Python and R outputs the given input to the console window.

SAS	**Python**	**R**

`proc print data = d1;`
`run;`
	`print(d1)`	`print(d1)`

`proc print data = d1;`
`   var var1;`
`run;`
	`print(d1.var1)`	`Print(d1$var1)`

FUNCTIONS

In the clinical programming world, functions are helpful in analyzing and processing data. Examples of how to call functions in SAS, Python, and R are listed below.

SAS	**Python**	**R**

`var2 = upcase(var1)`
`var2 = lowcase(var1)`
`var2 = propcase(var1)`
`var2 = left(var1)`
 	
`var2 = trim(var1)`
 	
`var2 = strip(var1)`
	`var2 = var1.upper()`
	`var2 = var1.lower()`
	`var2 = var1.title()`
	`var2 = var1.lstrip()`
	
	`var2 = var1.rstrip()`
	
	`var2 = var1.strip()`
		`var2 <- toupper(var1)`
		`var2 <- tolower(var1)`
		
		`var2 <- trimws(var1,`
		`        “l”)`
		`var2 <- trimws(var1,`
		`        “r”)`
		`var2 <- trimws(var1)`

`put(numeric_var1, best.)`	`str(numeric_var1)`	`toString(numeric_var1)`

CONCATENATION

The clinical programmer often needs to combine one or more values in SAS® to create new data fields for reporting. SAS, python, and R have the capability to allow the user to combine variables and values. The SAS concatenation operator is < II >, the Python concatenation operator is < + >, and character (or string) values can be combined with the *paste()* function in R.

SAS	Python	R

x = “hello”
y = “world”
 	
x || " " || y
 	
>> “hello world”
	x = “hello”
	y = “world”
	
	x + y
	
	>> “hello world”
		x = “hello”
		y = “world”
		
		paste(x, y, sep = " ")
		
		>> “hello world”

COUNTS

It is common for the clinical programmer to determine the frequency or number of occurrences for values found within each variable of a dataset. The SAS procedure PROC FREQ produces frequency counts and cross tabulation tables and can produce statistics to analyze relationships among the variables. The equivalent *value_counts* and *Pandas’s crosstab()* functions in Python and the *table()* and *prop.table()* functions in R can be used to produce such frequency counts and tables.

SAS	Python	R

proc freq data=d1;
   tables var1*var2;
run;
	x.value_counts(dropna =
	   False).sort_index()
	
	pd.crosstab(df.var1,
	   df.var2).apply(lambda
	   r: r/r.sum(), axis =
	   1)
		table(d1$var1, d1$var2)
		
		prop.table(table(d1$var1,
		   d1$var2))
		prop.table(table(d1$var1,
		   d1$var2), 1)
		prop.table(table(d1$var1,
		   d1$var2), 2)

DATA MANIPULATION

An important responsibility of the clinical SAS programmer is to transform the “raw” clinical data into more useful analysis-ready data. Transformation of this raw data may include the addition of variables that need to be defined, defining study populations (such as intent-to-treat or safety populations), referencing medical dictionaries, and merging two or more datasets. The following section details common data manipulation and transformation tasks that the clinical programmer performs.

Subsetting Data

The clinical programmer is oftentimes interested in a small portion of the dataset. Subsetting data allows the programmer to specify exactly which observations should be written to a dataset. In SAS, this can be done using there *where* statement as demonstrated in the example below. In Python and R, the indexing operator, or square brackets (`[ ]`), can be used to accomplish this task.

SAS	Python	R
`data d2;`	`d2 = [d1.x == "hello"]`	`d2 <- d1[ which( x ==`
`   set d1;`		`   “hello”)]`
`   where x = 'hello';`		
`run;`		

Merging Data

Merges and joins can be performed with the DATA step in SAS. The *merge()* functions within *Pandas’s* Python library and within R can be used to perform left, right, inner, and outer joins.

SAS	Python	R
`data d3;`	`#left join`	`#left join`
`   merge d1(in=a)`	`d3 = pd.merge(d1,`	`d3 <- merge(x=d1, y=d2,`
`d2(in=b);`	`   d2, on = 'var1',`	`   by = "var1",`
`   by var1;`	`   how = 'left')`	`   all.x=TRUE)`
`/*left join*/ if a;`		
`/*right join*/ if b;`	`#right join`	`#right join`
`/*inner join*/ if a and b;`	`d3 = pd.merge(d1,`	`d3 <- merge(x=d1, y=d2,`
`/*full merge is default*/`	`   d2, on = 'var1',`	`   by = "var1",`
`run;`	`   how = 'right')`	`   all.y=TRUE)`
	`#inner join`	`#inner join`
	`d3 = pd.merge(d1,`	`d3 <- merge(x=d1, y=d2,`
	`   d2, on = 'var1',`	`   by = "var1")`
	`   how = 'inner')`	
		`#outer join`
	`#outer join`	`d3 <- merge(x=d1, y=d2,`
	`d3 = pd.merge(d1,`	`   by = "var1",`
	`   d2, on = 'var1',`	`   all=TRUE)`
	`   how = 'outer')`	

Sorting Data

Datasets sometimes need to be ordered by the values of one or more character or numeric variables before certain tasks or procedures can be performed. The SORT procedure in SAS replaces the original or input dataset. In the example below, the PROC SORT orders the *d1* dataset, so that it is sorted by variable *var1* and then variable *var2.* The *sort()* function in Python and the *order()* function in R can be used to perform this task.

SAS	Python	R
`proc sort data=d1; by var1 var2…; run;`	`df.sort(['var1', 'var2'], ascending = True)`	`d1[order(d1$var1, d1$var2)]`

Dropping and Keeping Variables

The SAS® programmer can use the KEEP or DROP statement within the DATA step to let SAS know which variables to write to an output dataset. To perform this task in Python, the *loc()* or *drop()* functions can be used. The indexing operator, square brackets (`[ ]`), or the *subset()* function can be used to keep or drop variables in R. In the examples below, *d1* is the input dataset and *d2* is the output dataset.

SAS	Python	R
`data d2; set d1; keep var1 var2….; run;`	`d2 = d1.loc([:, ['var1', 'var2'])`	`d2 <- d1[c("var1", "var")]`
`data d2; set d1; drop var3 var4…; run;`	`d2 = d1.drop(['var1', 'var2'], axis = 1)`	`d2 = subset(d1, select = -c("var1","var2"))`

CONCLUSION

There is a growing movement and acceptance towards using Python and R in the Clinical SAS® programming world. The examples of common SAS® syntax and code, such as those related to documentation, importing data, transforming data, and their translation to Python and R provided above exemplify the similarities in the syntax of the languages. The authors of this paper hope that in drawing parallels in the syntax of the languages, the initial apprehension associated with learning a new language can be assuaged. Furthermore, we hope that this serves as a starting point for the SAS® programmer to adapt to the anticipated changing landscape of the clinical programming world.

CONTACT INFORMATION

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

Janet J. Li
Pfizer Inc.
janet.li@pfizer.com