PharmaSUG China 2022 - Paper AT-129

The interaction between SAS and R, python, linux

Kaiping Yang, BeiGene

ABSTRACT

The interaction between SAS and R, Python, Linux and other open source languages is a topic worth exploring and applying.

R is a free software environment for statistical computing and graphics; Python is a programming language that allows you to work efficiently and integrate different systems easily; Linux is a free and open source Unix-like operating system.

The interaction between SAS ,R, Python and Linux can improve the efficiency of data analysis, statistical graphics, automation and other aspects in the pharmaceutical field. This article introduces the interaction technology between SAS version 9.4 and R 3.6.0 and Python 3.7.4 on Linux operating system.

R integrates cutting-edge statistical analysis methods and powerful statistical graphing capabilities through open source packages.

SAS can execute R scripts within a built-in procedure which named PROC IML in SAS environments. But PROC IML doesn't support macro variables and macro procedures when execute R scripts. This paper will introduce how to get over this issue by using a macro procedure %MS_R. The secondary development of SAS macro based on %MS_R will also be introduced which includes the application of vroom, readxl, tidyverse and other packages. We will focus on solving the problem on importing csv/excel file by SAS in this paper. .

Python is excellent in machine learning, automation, and software development, and Linux has become the backbone of the work. You can even develop desktop applications to manage SAS applications through Python and Linux.

SAS can also use the X statement and filename pipe statement to call python and Linux scripts. On the other hand, Python and Linux can also call SAS to complete automated operations to improve efficiency. This article will also give a briefly introduction on how SAS interacts with Python and Linux.

INTRODUCTION

The interaction between SAS, R, Python and Linux can improve the efficiency of data analysis, statistical graphics, automation and other aspects in the pharmaceutical field. This article introduces the interaction technology between SAS version 9.4 and R 3.6.0 and Python 3.7.4 on Linux operating system.

The article structure of this article is as follows:

1. SAS interaction with R

The main introduction is sas through proc iml procedure call R for data processing and analysis, and returning the result dataset to the SAS dataset. Encapsulates proc iml to call R as macro procedures %ms_r and %proc_r, extending functionality. Leverage %ms_r secondary development %ms_r_import enhance SAS import data with R. Use the R package SASmarkdown to complete the operation of interacting with SAS in R

2.SAS interaction with Python

The main introduction is that SAS completes the interaction with Python through the data step and the filename pipe statement. SAS interacts with Python through the PROC FCMP process. Install SAS l call python procedures as macro procedures %ms_r and %proc _r, extending functionality 。

3.SAS uses the X statement and fiveleme pipe to invoke Linux sh scripts, and Linux to invoke SAS programs.

THE INTERACTION BETWEEN SAS AND R

R is a free language and environment for statistical computing and graphing. Like the SAS/IML language, the R language has features suitable for developers of statistical algorithms: the ability to manipulate matrices and vectors, a large number of built-in functions for computing statistics, and writing user-defined functions. There are also a large number of user-contributed packages in R that implement specialized calculations.

SAS CALLS R THROUGH THE PROC IML PROCEDURE

SAS calls R through the proc iml procedure for data processing and analysis, and returns the result dataset to the SAS dataset. The Table 1 lists the R versions supported by SAS, this article interacts with SAS 9.4 and R 3.6.0.

SAS Version	PROC IML	SAS/IML Studio	Release Date	R Versions
9.4	12.3	12.3	Jul 2013	2.13.0 - 3.0.1
9.4M1	13.1	13.1	Dec 2013	2.13.0 - 3.2.5
9.4M2	13.2	13.2	Aug 2014	2.13.0 - 3.2.5
9.4M3	14.1	14.1	Aug 2015	2.13.0 - 3.2.5
9.4M4	14.2	14.2	Nov 2016	2.13.0 - 3.6.3
9.4M5	14.3	14.3	Sep 2017	2.13.0 - 3.6.3*
9.4M6	15.1	15.1	Nov 2018	2.13.0 - 3.6.3*
Hot Fix*	14.3-15.2		Feb 2021	R 4.0.x
9.4M7	15.2	N/A	Aug 2020	4.0.0* - 4.2.0
9.4M8	15.3	N/A	Aug 2022 (Tent)	4.0.0 - 4.2.0

Table 1. Compatibility with R Releases

Starting with SAS/IML 9.22, calling R is available in PROC IML. This chapter shows you how to use SUBMIT and ENDSUBMIT statements to call R functions from PROC IML.

In order to invoke the R software, you must first install R on the same computer that is running the SAS software.

The RLANG system option determines whether you have permission to call R from the SAS system. You can determine the value of the RLANG option by submitting the following SAS statement:

```
proc options option=RLANG;
run;
```

The result is one of the following statements in the SAS log:

```
SAS (r) Proprietary Software Release 9.4  TS1M4

 RLANG                Enables SAS to execute R language
```

If the SAS log contains this statement, you can call R from the SAS system.

```
NORLANG   Do not support access to R language interfaces
```

If the SAS log contains this statement, you do not have permission to call R from the SAS system.

The RLANG option can only be changed at SAS startup. In order to invoke R, you must start the SAS system with the -RLANG option. For security reasons, some system administrators configure sas systems to boot with the -NORLANG option.

Submit the R declaration

Using the SUBMIT statement, the R option is added: SUBMIT / R. All statements in the program between the SUBMIT statement and the next ENDSUBMIT statement are sent to R for execution. The ENDSUBMIT statement must appear on a single line.

Use a SUBMIT statement with the R option in the following program to send R a statement that calculates the product of a matrix and vector.

```
proc iml;
submit / R;
rx <- matrix( 1:3, nrow=1)
rm <- matrix( 1:9, nrow=3, byrow=TRUE)
rq <- rm %*% t(rx)
print(rq)
endsubmit;
quit;
```

The printout of R is automatically taken to the SAS Output window, as shown in Output 1.

```
       [,1]
[1,]     14
[2,]     32
[3,]     50
```

Output 1. Output from a product of a matrix and vector

You can use SUBMIT/R to invoke R multiple times in a single SAS/IML program, and any R variables created during earlier calls can be used for subsequent calls. Similarly, if you load a package, the package remains loaded until the PROC IML exits.

Call the R package from PROC IML

If you have an R package installed, you can call library(*package*) from inside the SUBMIT block to load the package. You can then call the functions in the package. If you are not installing the R package in the default folder, you need to run the following code:

```
.libPaths(<your path of R package>)
```

```
library(package)
```

Transferring Data between SAS and R Software

Table 2 summarizes the subroutines for transferring data between SAS and R software. It is important to note that the result of the R analysis can be a complex structure. In order to transfer an R object to SAS, the object must be cast to a data frame.

Subroutine	SAS	R
ExportDataSetToR	SAS data set	R data frame
ExportMatrixToR	SAS/IML matrix	R matrix
ExportTableToR	SAS/IML table	R data frame

ImportDataSetFromR	SAS data set	R expression
ImportMatrixFromR	SAS/IML matrix	R expression
ImportTableFromR	SAS/IML table	R expression

Table 2. Transferring Data between SAS and R Software

As a simple example, the following program transfers a data set from SasHelp Libref to an R data frame named DF. Then, the program submits some R statements, displays the variable names of df, generates new variables BMI, lagWeight, leadWeight in DF and outputs DF to sas workspace, naming it Class.

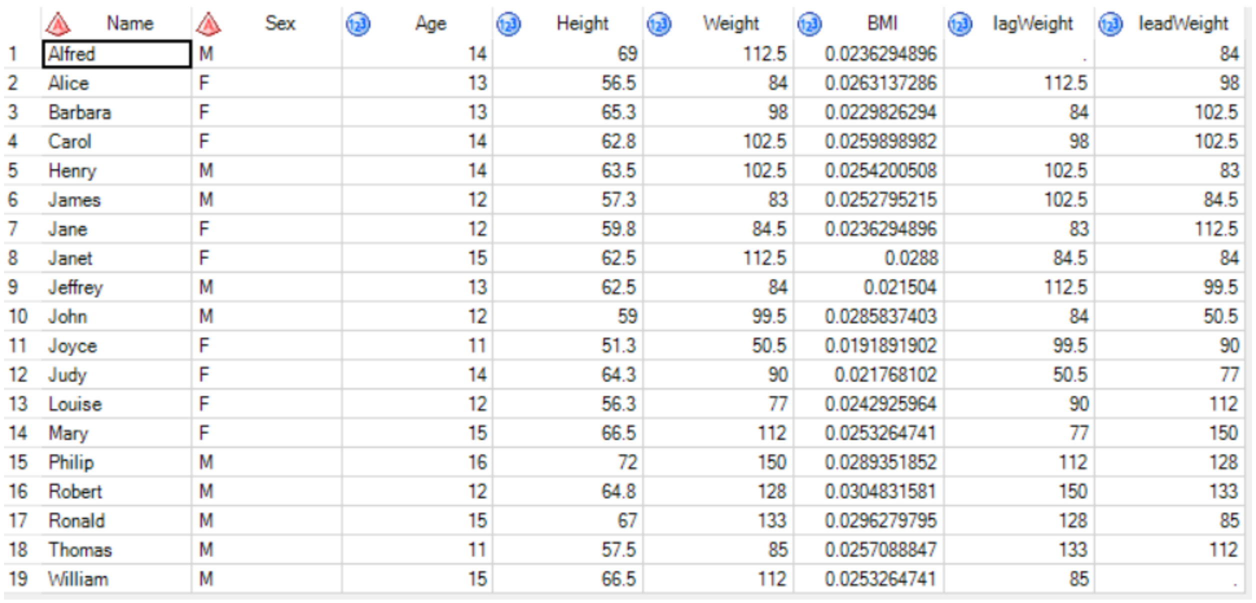
```
proc iml;
call ExportDataSetToR("Sashelp.Class", "df" );
submit / R;
names(df)
.libPaths(<your path of R package>)
library(tidyverse)
df = df %>%
       mutate(BMI = Weight / (Height ^ 2),
              lagWeight=lag(Weight),
              leadWeight=lead(Weight))
endsubmit;
call ImportDataSetFromR("work.Class", "df" );
quit;
```

The R function names produces the output shown in Output 2.

```
[1] "Name"    "Sex"    "Age    ""Height"    "Weight"
```

Output 2. Output from R function names

Go to the dataset in SAS Class as shown in the Figure 1:



	Name	Sex	Age	Height	Weight	BMI	lagWeight	leadWeight
1	Alfred	M	14	69	112.5	0.0236294896	.	84
2	Alice	F	13	56.5	84	0.0263137286	112.5	98
3	Barbara	F	13	65.3	98	0.0229826294	84	102.5
4	Carol	F	14	62.8	102.5	0.0259898982	98	102.5
5	Henry	M	14	63.5	102.5	0.0254200508	102.5	83
6	James	M	12	57.3	83	0.0252795215	102.5	84.5
7	Jane	F	12	59.8	84.5	0.0236294896	83	112.5
8	Janet	F	15	62.5	112.5	0.0288	84.5	84
9	Jeffrey	M	13	62.5	84	0.021504	112.5	99.5
10	John	M	12	59	99.5	0.0285837403	84	50.5
11	Joyce	F	11	51.3	50.5	0.0191891902	99.5	90
12	Judy	F	14	64.3	90	0.021768102	50.5	77
13	Louise	F	12	56.3	77	0.0242925964	90	112
14	Mary	F	15	66.5	112	0.0253264741	77	150
15	Philip	M	16	72	150	0.0289351852	112	128
16	Robert	M	12	64.8	128	0.0304831581	150	133
17	Ronald	M	15	67	133	0.0296279795	128	85
18	Thomas	M	11	57.5	85	0.0257088847	133	112
19	William	M	15	66.5	112	0.0253264741	85	.

Figure 1. Generate a dataset of BMI results

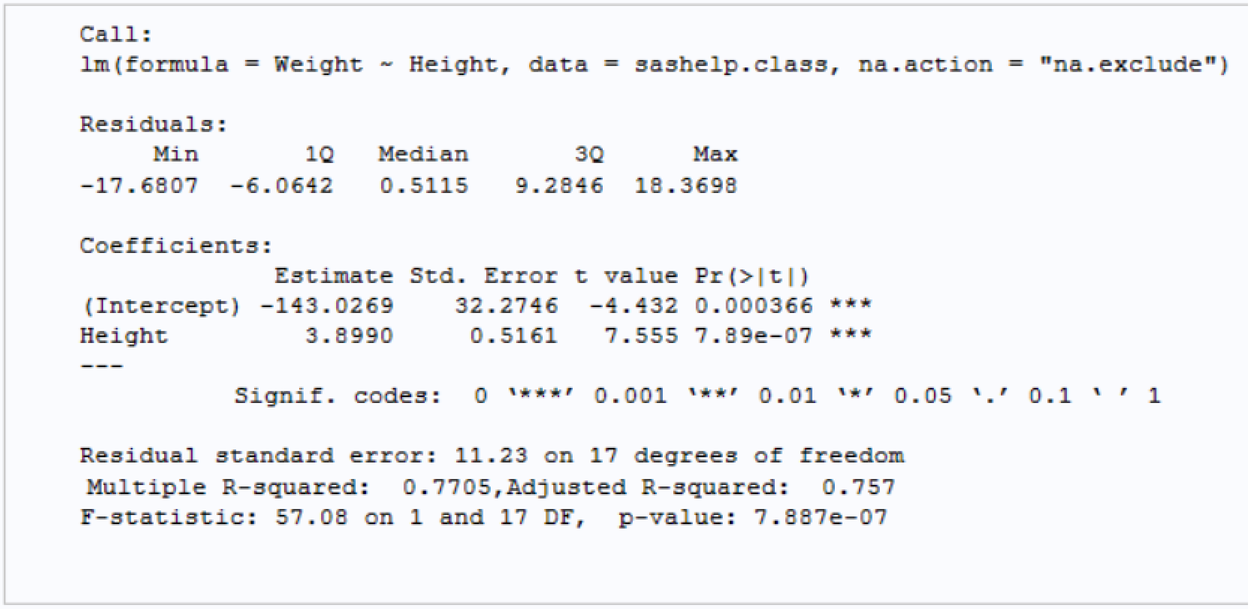
Note that the final data set Class is the variable BMI and lagWeight, which can be easily generated in SAS. However, lead function in tidyverse package does not have corresponding function in SAS, so leadWeight variable cannot be directly generated in SAS. Therefore, we can enhance SAS data processing capability with the famous tidyverse package in R.

Use R to statistical analyze data in a SAS dataset

You can use the R language's rich statistical functions or statistical packages for statistical analysis. For example, you can create a R data frame named Class and model a Weight variable with the following statement:

```
proc iml;
call ExportDataSetToR("Sashelp.Class", "Class");
submit / R;
Model <- lm(Weight ~ Height, data=Class, na.action="na.exclude")
endsubmit;
quit;
```

The result of linear regression is shown in the Figure 2:



```
Call:
lm(formula = Weight ~ Height, data = sashelp.class, na.action = "na.exclude")

Residuals:
     Min       1Q   Median       3Q      Max
-17.6807  -6.0642   0.5115   9.2846  18.3698

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -143.0269    32.2746  -4.432 0.000366 ***
Height         3.8990     0.5161   7.555 7.89e-07 ***
---
           Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 11.23 on 17 degrees of freedom
 Multiple R-squared:  0.7705,Adjusted R-squared:  0.757
F-statistic: 57.08 on 1 and 17 DF,  p-value: 7.887e-07
```

Figure 2. Caption for linear regression

Call the R graph from PROC IML

R creates a graph in a separate window that, by default, appears on the same computer that is running R. If you are running PROC IML and R locally on your desktop or laptop, you can display R graphics. However, if you are running client software that is connected to a remote SAS server running PROC IML and R, R graphics may be disabled.

MACRO PROCEDURE: %MS_R

When we wanted to encapsulate the R code into a macro procedure to implement multiple calls, we found that SAS would tell us Submit block cannot be directly placed in a macro.

If we encapsulate the following macro process, we will get the error prompt shown in Output 3:

```
%macro test();
proc iml;
YVar = "Weight";
XVar = "Height";
call ExportDataSetToR("Sashelp.Class", "df");
```

```
submit XVar YVar / R;
Model <- lm(&YVar ~ &XVar, data=df)
summary (Model)
endsubmit;
quit;
%mend;
%test();
```

```
ERROR: Submit block cannot be directly placed in a macro. Instead, place
the submit block into a file first and then use %include to include the
file within a macro definition.
```

Output 3. Output from encapsulating the R code into a macro procedure

Based on the error prompt, we can encapsulate the procedure of proc iml calling R into a macro procedure:

First pass in the R code via the macro parameter code.

Secondly, export the file Rcode.sas containing the following contents through the data step:

```
submit / R;
< R code >
endsubmit;
```

Finally, the proc iml procedure step is passed in via `%include "path/Rcode.sas"`.

After adding features such as incoming outgoing SAS7bdat datasets, whether to load python libraries and what python libraries to load, display drawing results, etc., and optimizing the related outputs, we encapsulate a new macro process that is more extensible %ms_r：

```
%ms_r(indata=%str(),
      outdata=%str(),
      library=TRUE,
      libPaths=%str(<your path of R library>),
      code=);
```

The specific parameters are meanings as follows

- **indata**: optional, input datasets such as indata=dataset1 dataset2
- **outdata**: optional, output datasets such as outdata=dataset1 dataset2
- **library**:
  optional, TRUE/FLASE T/F, whether to import frequently used R packages
  required, the R packages you want to import such as library=tidyverse vroom
- **libPaths**: optional, sets the library trees within which packages are looked. default is your path of R library
- **code**: required, you need to encapsulate R code with a function like %nrbquote() or % NRSTR(), And separate the end of each complete R code sentence with a semicolon

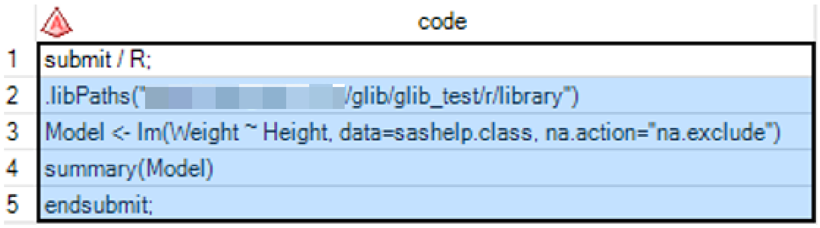
%ms_r has the following enhancements:

More powerful ability to pass sas macro variables into R

We can pass in any macro variable to code without SAS getting an error. In fact, the macro variable is resolved in SAS before R is passed in. Let's rewrite %test with %ms_r:

```
%let YVar=Weight;
%let XVar=Height;
%ms_r(
    indata=sashelp.class,
    library=FALSE,
    code=%nrbquote(
    Model <- lm(&YVar ~ &XVar, data=sashelp.class);
    summary(Model)
));
```

%ms_r will default to dataset to check if the code entered into R is correct as shown in Figure 3 and the result is shown in Figure 2:



	code
1	submit / R;
2	.libPaths("[redacted]/glib/glib_test/r/library")
3	Model <- lm(Weight ~ Height, data=sashelp.class, na.action="na.exclude")
4	summary(Model)
5	endsubmit;

Figure 3. Dataset of R code for %ms_r

Print the result of R package gglot2 drawing onto the SAS page

If SAS EG is used, then the drawing of R will not be displayed. So we can use the SAS interface to display the drawing program in R saved png, png, jpg format.

```
%ms_r(library=ggplot2,code=%nrbquote(
p <- ggplot(data= mtcars, aes(x = wt, y = mpg)) +
geom_point();
ggsave("../glib/glib_test/cp/tools/uat/plot.png",
      p,width = 10, height = 10, units = "cm");
));
```

Figure 4 shows the result of printing the R output picture to the SAS interface:

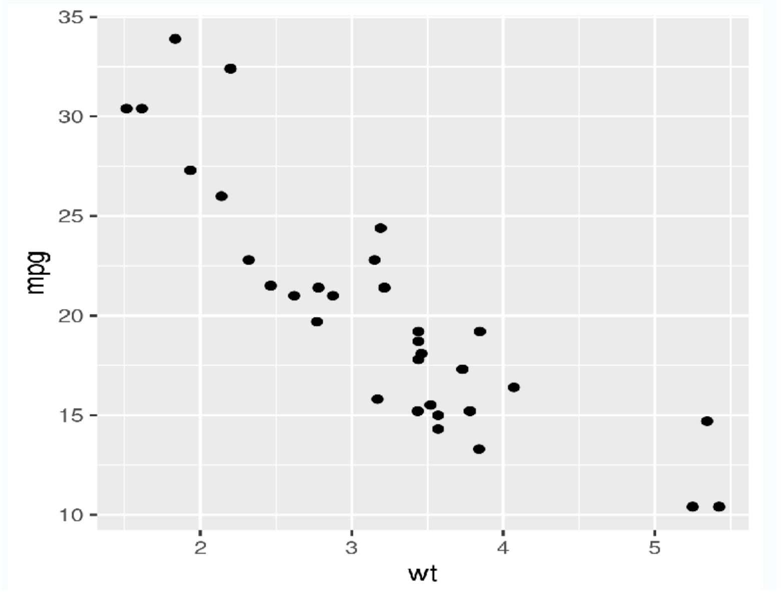


Figure 4. Print the result of R drawing onto the SAS page

Be encapsulated into other macro processes and used R for secondary development

When we want to use the many packages in R to do things that SAS can't easily do, we can encapsulate the process of % ms_r calling R in other macro procedures to facilitate repeated calls. An example of %ms_r_import will be covered in detail in case study.

MACRO PROCESS: %PROC_R

We can take the cards4/datalines4 statement to pass in the R code in the form of a row of data:

```
data _null_;
   file "../glib/glib_test/cp/tools/uat/test.R";
   infile cards4 length=len;
   input code $varying8192. len;
   _infile_ = resolve(_infile_);
   put _infile_;
cards4;
hello_world < - function(){
    print("hello world")
}
hello_world()
;;;;
run;
```

Note:

Cards4 can allow typing; But it has to be used ;;;; As the end of the data row. Cards4 statements cannot be encapsulated into macro procedures because row statements of data are always run before macro procedures.

$varying8192. is a string format in variable-length format.

The resolve function allows parsing macro variables in cards4.

Although the cards4 statement cannot be encapsulated in a macro procedure, we can encapsulate the SAS call R procedure in the form of two macro procedures sandwiched between the cards4 sentence:

```
%proc_r();
cards4;
hello_world < - function(){
print("hello world")
}
hello_world()
;;;;
%quit_r();
```

A simple example of %proc_r() and %quit_r() is as follows:

```
%macro proc_r();
data front_code;
   length code packages $20000.;
   code="submit / R;" ; output;
   %end;
run;

data body_code;
   infile cards4 length=len;
   input code $varying8192. len;
   code=dequote(resolve(quote(code)));
```

```
%mend;

%macro quit_r();
data ms_code;
   set front_code body_code end=eof;
   output;
   if eof then do; code="endsubmit;" ; output; end;
run;

%let saswork=%sysfunc(pathname(work));

data _null_;
   set ms_code;
   file "&saswork./Rcode.sas";
   put code;
run;


proc iml;
%include "&saswork./Rcode.sas";
quit;

proc delete data=front_code body_code; run;
%mend;
```

Of course, we can mimic %ms_r to build the more complex %proc_r() and %quit_r() where %proc_r() is as follows:

```
%proc_r(indata=%str(),
        outdata=%str(),
        library=TRUE,
        libPaths=%str(<your path of r library>).
);
```

The parameter settings are roughly the same as %ms_r, except that the code parameter is passed in as cards4:

- code: required, You need to encapsulate R code sandwiched between "cards4;/datalines4;" and ";;;;" , such as:
  ```
  cards4;
  print("hello world")
  ;;;;
  ```

The advantage of %proc_r() and %quit_r()is that it is more convenient to type the R code, the disadvantage is that %proc_r() and %quit_r() cannot be encapsulated in other macro procedures (because the cards4 statement would be encapsulated in the macro procedure resulting in an error).

CASE STUDY: %MS_R_IMPORT ENHANCE SAS IMPORT DATA WITH R

We encounter various pain points when importing csv/excel files using SAS in actual work, for example: SAS uses roc import to import csv files, because SAS needs to obtain the length of character variables, it needs to add guessingrows=max parameters. If you specify guessingrows=1000 but SAS is importing large files (such as 1 million rows of data), the guessingrows=max parameter will cause the data to not be imported successfully. However, changing to a fixed value such as guessingrows=1000 may cause character variables to be truncated. SAS is slower when importing large excel files using proc import SAS is trickier when importing non-standard csv/excel files.

In the R packages vroom, readxl, openxlsx, etc. have high import efficiency and rich functions when importing csv/excel files. We can take advantage of %ms_r integrate these features into the new macro process %ms_r_import to supplement the shortcomings of S AS importing csv/excel files.

```
%ms_r_import(outdata =,
             datafile = %str(),
             file_path = %str(),
             in_file = %str(),
             delim = NULL,
             sheetname = NULL,
             first_line = 1,
             modify=N,
             warning = T,
             clean=Y);
```

The meaning of the specific parameters is as follows:

- **outdata**: required, output dataset.
- **datafile**: required, data file. The file name part can be passed in a regular expression.
- **file_path**: required, file folder.
- **in_file**: required, file name. Can be passed in a regular expression.
- **delim**: required, one or more characters used to delimit fields within a csv file. If NULL the delimiter is guessed
- **sheetname**: required, sheet name of excel. The default is NULL, and the program automatically selects the first sheet.
  You can pass in either the name of the sheet or the number of the sheet,such as: sheetname=test or sheetname=1
- **first_line**: required, what line to start reading data from csv/excel. By default, it starts from the first line.
  For csv,you can pass in numbers such as: first_line=5
  For excel,you can pass in numbers or pass in letters and numbers,such as: first_line=5 or first_line=C5
- **modify**: required, whether to automatically find the first row and column for csv/excel. The default is N
- **warning**: suppress warning and message from R, default is T/TRUE.
- **clean**: optional, whether to clear The intermediate dataset,default is Y.

%ms_r_import has the following enhancements:

Import a csv file with an unknown delimiter, or specify a delimiter

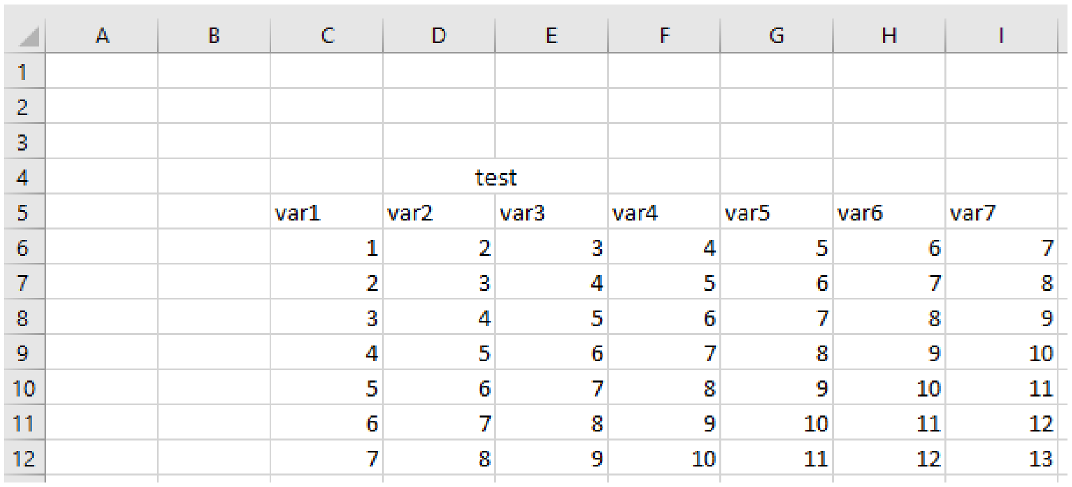
When importing a csv file with tab key as the delimiter, %ms_r_import call the functions vroom(), vroom() in the R package vroom The delimiter of the csv file is automatically recognized, or the delimiter can be specified via the macro parameter delim=%str(\t):

```
%ms_r_import(datafile=%str(../glib/glib_test/cp/tools/uat/test_tab.csv),
             outdata=test);
%ms_r_import(datafile=%str(../glib/glib_test/cp/tools/uat/test_tab.csv),
             outdata=test,delim=%str(\t));
```

Specify excel file data in several sheets and first row cells

When importing an excel file, %ms_r_import will call the function read_excel() in R package readxl first, read_excel() allows the first sheet in excel to be specified in numerical form, and allows reading from a specified row and column.

Suppose test_modify.xlsx file data variable names start from cell C5, the first sheet name is unknown. The data content is shown in the Figure 5:



	A	B	C	D	E	F	G	H	I
1									
2									
3									
4				test					
5			var1	var2	var3	var4	var5	var6	var7
6			1	2	3	4	5	6	7
7			2	3	4	5	6	7	8
8			3	4	5	6	7	8	9
9			4	5	6	7	8	9	10
10			5	6	7	8	9	10	11
11			6	7	8	9	10	11	12
12			7	8	9	10	11	12	13

Figure 5. data content of test_modify.xlsx

We read the data by specifying the macro parameter sheetname=1, first_line=C5:

```
%ms_r_import(datafile=%str(../glib/glib_test/cp/tools/uat/test_modify.xlsx),
             outdata=test,sheetname=1,first_line=C5);
```

Automatically finds the first row and first column of data in csv/excel files

By using the functions vroom() and read_excel() in R and user-defined functions, we can achieve the first row and first column of data in the csv/excel file by setting the macro parameter modeify=y. The files test_modify.xlsx above and the test_modify.csv files converted can be read with mode=y.

```
%ms_r_import(datafile=%str(../glib/glib_test/cp/tools/uat/test_modify.csv),
             outdata=test,modify=y);
%ms_r_import(datafile=%str(../glib/glib_test/cp/tools/uat/test_modify.xlsx),
             outdata=test,modify=y);
```

Support regular expression search for the latest date file that matches the keyword

%ms_r_import performs a regular match on the last level of file names in the macro parameter datafile and returns the latest date file that satisfies the regular match (looking only for csv and xlsx files).

Suppose we need to import csv or xlsx files with the latest date in the file name that contains the QueryDetail character:

```
%ms_r_import(datafile=%str(../glib/glib_test/cp/tools/uat/.*QueryDetail.*),
             outdata=test);
%ms_r_import(datafile=%str(../glib/glib_test/cp/tools/uat/QueryDetail),outdata=test);
```

%ms_r_import prints the final found file name to the SAS output as shown in Output 4:

```
The last file name is:
/glib/glib_test/cp/tools/uat/XXX-XXXX-XXX_QueryDetailReport_20220701.xlsx
```

Output 4. Output from last file name of %ms_r_import

We can also specify the file name suffix to csv:

```
%ms_r_import(datafile=%str(../glib/glib_test/cp/tools/uat/.*QueryDetail.*.csv),
             outdata=test);
```

If the file name is QueryDetail.xlsx, you can pass in ^QueryDetail.xlsx$:

```
%ms_r_import(datafile=%str(../glib/glib_test/cp/tools/uat/^QueryDetail.xlsx$),
             outdata=test);
```

If you want to find the latest file in the folder, you can enter it *:

```
%ms_r_import(datafile=%str(../glib/glib_test/cp/tools/uat/.*),outdata=test);
```

Import big data

R package vroom and readxl import large csv and excel extremely fast.

There are 1675762 rows and 27 columns of big_data.csv. Import data using %ms_r_import:

```
%let time1=%sysfunc(time());
%ms_r_import(datafile=%str(../glib/glib_test/cp/tools/uat/big_data.csv),
             outdata=test);
%let time2=%sysfunc(time());
data _null_;
R=&time2-&time1;
put "time of R import: " R "s";
run;
```

%ms_r_import takes 39.99 seconds as shown in Output 5, in fact, the vroom package reads the file into R in only 0.35 seconds, S AS's subroutines ImportDataSetFromR take a long time, But this avoids the problem of character variable length, and the cost is worth it.

```
time of R import: 39.993641853 s
```

Output 5. Output from importing big_data.csv

There are 279,599 rows, 42 columns of big_data.xlsx. Import data using %ms_r_import and proc import:

```
%let time1=%sysfunc(time());
%ms_r_import(datafile=%str(../glib/glib_test/cp/tools/uat/ big_data.xlsx),
             outdata=test);
%let time2=%sysfunc(time());
proc import out= data
   datafile= "../glib/glib_test/cp/tools/uat/big_data.xlsx"
   dbms=xlsx replace;
   getnames=yes;
run;
%let time3=%sysfunc(time());
data _null_;
   R=&time2-&time1;
   SAS=&time3-&time2;
   diff=SAS-R;
   put "time of R import: " R "s";
   put "time of SAS import: " SAS "s";
   put "diff of time: " diff "s";
```

```
run;
```

%ms_r_import 2 3.98 seconds to import data and 112.54 seconds for SAS as shown in Output 6. %ms_r_import greatly improves the efficiency of importing excel files. In fact, the readxl package takes only 8.698 seconds to read the file into R, and the subroutines of S AS, ImportDataSetFromR, take a long time.

```
time of R import: 23.98407793 s
time of SAS import: 112.541435 s
diff of time: 88.557357073 s
```

Output 6. Output from importing big_data.xlsx

R CALLS SAS

We use the R package SASmarkdown to complete the operation of interacting with SAS in R. Suppose you already have a basic understanding of Markdown for document formatting, Rmarkdown for including executable code in documents, and SAS for writing code.

After setting up the relevant configuration with SASmarkdown, we open the script in rmd format in Rstudio, create a code block and run SAS. The code is up:

~~~
```{r example1, engine="sas", engine.path=saspath, engine.opts=sasopts, comment=""}
proc means data=sashelp.class(keep = age);
run;
```
~~~

The specific operation is as follows (refer to SAS Using R Markdown):

Set up your SAS engine configuration.

It depends on your operating system, SAS version, and whether you have SAS installed in the default location. This code captures SAS in Linux:

```
require(SASmarkdown)
saspath < - "sas"
sasopts <- "-nosplash -ls 75"
```

Set the SAS code block.

A simple block of code in Rmarkdown might look like this:

~~~
```{r engine="sas", engine.path=saspath, engine.opts=sasopts, comment=""}
proc means data=sashelp.class(keep = age);
run;
```
~~~

In documentation, this will result in Output 7:

```
proc means data=sashelp.class (keep = age);
run;
                            The MEANS Procedure

                         Analysis Variable : Age

     N             Mean          Std Dev          Minimum          Maximum
    ------------------------------------------------------------------
    19       13.3157895        1.4926722       11.0000000       16.0000000
    ------------------------------------------------------------------
```

Output 7. Output from R calls SAS

Rerun with HTML output.

This block of code changes the output of the previous code block to HTML by setting engine=" sashtml":

~~~
```{r engine="sashtml", engine.path=saspath, engine.opts=sasopts, comment=""}
proc means data=sashelp.class(keep = age);
run;
```
~~~

The output in HTML format is shown in the Table 3:

Analysis Variable : Age				
N	**Mean**	**Std Dev**	**Minimum**	**Maximum**
19	13.3157895	1.4926722	11.0000000	16.0000000

Table 3. Analysis Variable Age of sashelp.class

Displays the SAS log file.

We can repeat the block of code using engine="saslog" to display sas logs instead of SAS code.

~~~
```{r engine="saslog", engine.path=saspath, engine.opts=sasopts, comment=""}
proc means data=sashelp.class(keep = age);
run;
```
~~~

In documentation, this will result in Output 8:

```
1          proc means data=sashelp.class (keep = age);
2          run;

NOTE: There were 19 observations read from the data set SASHELP. CLASS.
NOTE: The PROCEDURE MEANS printed page 1.
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.14 seconds
      cpu time            0.10 seconds

                            The MEANS Procedure

                         Analysis Variable : Age

     N             Mean          Std Dev          Minimum          Maximum
    ------------------------------------------------------------------
    19       13.3157895        1.4926722       11.0000000       16.0000000
    ------------------------------------------------------------------
```

Output 8. Output from R calls SAS using engine="saslog"

THE INTERACTION BETWEEN SAS AND PYTHON

Python is excellent in machine learning, automation, and software development, and Linux has become the backbone of the work. You can even develop desktop applications to manage SAS applications through Python and Linux.

SAS CALLS PYTHON

Run python script by using FILENAME fileref PIPE 'UNIX-command'

We can create a python script via the data step of SAS:

```
data _null_;
   file "../glib/glib_test/cp/tools/uat/hello_world.py" lrecl=30000;
   put "def hello_world():";
   put "    print('hello world')";
   put "hello_world()";
run;
```

Typing code using put statements can be a bit tedious, so we can take cards4/datalines4 statements to pass in python code in the form of rows of data

```
data _null_;
   file "../glib/glib_test/cp/tools/uat/hello_world.py";
   infile cards4 length=len;
   input code $varying8192. len;
   _infile_ = resolve(_infile_);
   put _infile_;
cards4;
def hello_world():
print("hello world")
hello_world()
;;;;
run;
```

Note:

Cards4 can allow typing; But it has to be used ;;;; As the end of the data row. Cards4 statements cannot be encapsulated into macro procedures because row statements of data are always run before macro procedures.

$varying8192. is a string format in variable-length format.

The Resolve function allows parsing macro variables in cards4.

We use the **FILENAME fileref PIPE 'UNIX-command'** ;

The syntax runs the created python script and prints the result of the script run into the sas log via the data step:

```
filename test pipe "python3 ../glib/glib_test/cp/tools/uat/hello_world.py";
data _null_;
   infile test;
   input;
   put _infile_;
run;
```

The output of SAS logs is shown in the Output 9:

```
NOTE: The infile TEST is:
      Pipe command="python3 ../build/glib/glib_test/cp/tools/uat/test.py"

hello world
```

Output 9. Output from SAS logs for python script

Python Function in PROC FCMP

PROC FCMP's Python objects enable you to embed and import Python functions into SAS programs. Python code is not converted to SAS code. Instead, the Python code runs in the Python interpreter of your choice and returns the results to the SAS. With minor modifications to your Python code, you can run Python functions from SAS and easily program in both languages at the same time.

Python objects require environment variables to be set before they can be used in a SAS environment. If the environment variables have not been set, or if they are not set correctly, SAS returns an error when you publish python code.

This SAS code is an example of using PROC FCMP program that uses Python objects

```
proc fcmp;
   declare object py(python);
   submit into py;
   def MyPyFunc(arg1):
       "Output: MyKey"
       myvar = arg1 * 10
   endsubmit;
   rc = py.append("return myvar");
   rc = py.publish();
   rc = py.call("MyPyFunc", 5);
   a = py.results["MyKey"];
   put a=;
run;
```

The preceding statement produces the result as shown in Output 10:

```
MyResult=50
```

Output 10. Output from Python Function in PROC FCMP

MACRO PROCEDURE: %MS _PY

We can encapsulate sass calling python's procedure as a macro procedure

First pass in the python code through the macro parameters;

Secondly, export the python code file containing < python code > through the data step;

Finally, run the python script via the fileme pipe statement and print the run result via the data step.

After adding features such as incoming outgoing SAS7bdat datasets, whether to load python libraries and what python libraries to load, display drawing results, etc., and optimizing the related outputs, we encapsulate a new macro process that is more extensible %ms_py：

```
%ms_py(indata=%str(),
       outdata=%str(),
       library=TRUE,
       libPaths=%str(<your path of python library>),
       code=);
```

The specific parameters are meanings as follows

- **indata**: optional, input datasets such as indata=dataset1 dataset2
- **outdata**: optional, output datasets such as outdata=dataset1 dataset2
- **library**:
  optional, TRUE/FLASE T/F, whether to import frequently used python packages
  required, the python packages you want to import
- **libPaths**: optional, sets the library trees within which packages are looked. default is your path of python library
- **code**: required, you need to encapsulate python code with a function like %nrbquote() or % NRSTR(), And separate the end of each complete python code sentence with a semicolon, such as

```
            code= %nrbquote(
          "def hello_world():";
          "     print('hello world')";
          "hello_world()";
            )
```

Note that because python is sensitive to indentation, the code parameter here uses a line of strings separated by semicolons.

MACRO PROCESS: %PROC_PY

Although the cards4 statement cannot be encapsulated in a macro procedure, we can encapsulate the SAS call python procedure in the form of two macro procedures sandwiched between the cards4 statement:

```
%proc_py();
cards4;
def hello_world():
    print("hello world")
hello_world()
;;;;
%quit_py();
```

A simple example of %proc_py() and %quit_py() is as follows:

```
%macro proc_py();
data _null_;
   file "../build/glib/glib_test/cp/tools/uat/test.py";
   infile cards4 length=len;
   input code $varying8192. len;
   _infile_ = resolve(_infile_);
   put _infile_;
run;
%mend;

%macro quit_py();
filename test pipe "python3 ../build/glib/glib_test/cp/tools/uat/test.py";
data _null_;
```

```
    infile test;
    input;
    put _infile_;
run;
%mend;
```

Of course, we can imitate %ms_py to build the more complex %proc_py() and %quit_py(), where %proc_py() are as follows:

```
%proc_py(indata=%str(),
          outdata=%str(),
          library=TRUE,
          libPaths=%str(<your path of python library>).
          );
```

The parameter settings are roughly the same as %ms_py, except that the code parameter is passed in as cards4:

- code: required, You need to encapsulate R code sandwiched between "cards4;/datalines4;" and ";;;;" , such as:
  ```
  cards4;
  print("hello world")
  ;;;;
  ```

The %proc_py() and %quit_py() has the advantage that it is more convenient to type the Python code, the disadvantage is that proc_py() and %quit_py() cannot be encapsulated in other macro procedures (because The cards4 statement is encapsulated in the macro procedure resulting in an error report).

PYTHON CALLS SAS

SAS officially developed the Python package SASpy to complete the python call SAS function. The interface is designed to enable programmers to interact with SAS using Python syntax and constructs. This interface makes SAS an analytics engine or "calculator" for data analysis. In its simplest form, it is a transcoder that takes Python commands and converts them into SAS language statements. Run these statements and then return the results to Python for display or access. The installation configuration process is detailed in the official document SASPy — saspy 4.3.0 documentation (sassoftware.github.io).

This is the interface module for sass systems. It connects to SAS 9.4 or later and enables Python programmers to leverage their licensed SAS infrastructure through Python 3.7.4

Initial import

Assume that you have completed the installation and configuration. The following code completes the initial import:

```
import saspy
import pandas as pd
```

Start a SAS session

In the following code, we start a SAS session named with the default configuration. Each SAS session is a connection to a separate SAS instance. The cfgname parameter specifies the configuration definition (sascfg_personal.py) used to connect to the SAS.

If sascfg_personal.py only one connection definition, you do not need to specify the cfgname parameter. If the file has multiple connection definitions and you do not specify a connection definition to use with cfgname, you will be prompted to use that connection.

After the connection is established and the SAS session is started, a note is as shown in Output 11.

```
sas = saspy.SASsession(cfgname='default')
```

```
SAS Connection established.  Subprocess id is 28207
```

Output 11. Output from saspy.SASsession

Load the data into the SAS

Data can be easily loaded from many sources. The following example shows the most commonly used methods. In each case, it is a SASdata object that represents the SAS dataset.

CSV

In the following example, Python can access a CSV file. Read the CSV file into the data frame.

```
hr_pd = pd.read_csv("./HR_comma_sep.csv")
```

Pandas DataFrame

In the following example, Python reads the data frame and creates a SAS dataset in the SAS session.

```
hr = sas.df2sd(hr_pd)
```

In the following example, Python converts sas dataset to pandas dataframe.

```
hrdf2 = sas.sd2df(hr.table)
```

Existing SAS datasets

In the following example, Python cannot access any data files. Existing SAS datasets that are accessible to SAS sessions are associated with objects.

```
hr = sas.sasdata('hr', 'mylibref')
hr = sas.sasdata('hr') # if hr.sas7bdat is in your 'work' or 'user' library
```

Submit SAS code directly from a Python session

The following example demonstrates the common Python methods provided by this module.

If you encounter a situation where you need to submit SAS statements directly to the SAS system, you can use any of the three submit* methods to do so. The following example creates a side-by-side panel diagram to compare employees who have left their jobs with employees who are still working in the company based on their business unit, median performance rating, and satisfaction level.

The submit method returns a dictionary with two keys: LOG and LST. You can print() the LOG and sas.HTML() the LST. Or you can use the submitLST() method or the submitLOG() methods which automatically render the respective output for you.

```
c = sas.submitLST("""
proc means data=sashelp.class(keep = age);
""")
```

SAS INTERACTS WITH LINUX

Linux is a free and open source Unix-like operating system. SAS can invoke Linux's sh scripts through x statements and the filename pipe. Linux, as an operating system, naturally makes it easy to call SAS software.

SAS CALLS LINUX

X statement

The X Statement enables to run an operating system command or a Windows application from within a SAS session.

The syntax of an X statement is very simple:

X <‘command’> ;

‘command’: specifies an operating environment command that is enclosed in quotation marks.

We can call a Linux directive directly from X statement, such as the cd directive to switch folders:

```
x "cd ../glib/glib_test/cp/tools/uat"
```

You can also use X Statement to call the Linux sh script:

```
x "sh ../glib/glib_test/cp/tools/uat/hello_world.sh";
```

In fact, filename pipe directives run R and py scripts are essentially SAS calling R and py script via Linux directives.

```
x "Rscript ../glib/glib_test/cp/tools/uat/hello_world.R";
x "python3 ../glib/glib_test/cp/tools/uat/hello_world.py";
```

FILENAME: fileref PIPE 'UNIX-command'

Under UNIX, you can use the FILENAME statement to assign filerefs not only to external files and I/O devices, but also to a pipe.

Here is the syntax of the FILENAME statement:

FILENAME: fileref **PIPE** 'UNIX-command' ;

- **Fileref**: the name by which you reference the pipe from SAS.
- **PIPE**: identifies the device-type as a UNIX pipe.
- **‘UNIX-command’**: the name of a UNIX command, executable program, or shell script to which you want to route output or from which you want to read input. The commands must be enclosed in either double or single quotation marks.

The following code creates the sh script with the data step and prints the result to SAS with the filename PIPE statement and the data step:

```
data _null_;
  file "../glib/glib_test/cp/tools/uat/hello_world.sh" lrecl=30000;
  put "echo hello word";
run;
filename test pipe "sh ../glib/glib_test/cp/tools/uat/hello_world.sh";
data _null_;
  infile test;
  input;
  put _infile_;
```

```
run;
```

In fact, filename pipe directives run R and py scripts are essentially SAS calling R and py script via Linux directives:

```
filename test pipe "Rscript ../glib/glib_test/cp/tools/uat/hello_world.R";
filename test pipe "python3 ../glib/glib_test/cp/tools/uat/hello_world.py";
```

LINUX CALLS SAS

As an operating system, Linux makes it easy to call SAS programs.

Use the cd command to open the folder where the program is located:

```
cd ../glib/glib_test/cp/tools/uat
```

Use the ls directive to view programs in a folder:

```
ls
```

Run the program with the sasen *program name* or sasu8 *program name* directive:

```
sasen hello_world.sas
sasu8 hello_world.sas
```

CONCLUSION

The interaction between SAS and R, Python, Linux and other open source languages is a topic worth exploring and applying.

R is a free software environment for statistical computing and graphics; Python is a programming language that allows you to work efficiently and integrate different systems easily; Linux is a free and open source Unix-like operating system.

R integrates cutting-edge statistical analysis methods and powerful statistical graphing capabilities through open source packages.

SAS can execute R scripts within PROC IML in SAS environments. But PROC IML doesn't support macro variables and macro procedures when execute R scripts. This paper introduces how to get over this issue by using a macro procedure %ms_r. The secondary development of SAS macro based on %ms_r is also introduced which includes the application of vroom, readxl, tidyverse and other packages. We focus on solving the problem on importing csv/excel file by %ms_r_import.

Python is excellent in machine learning, automation, and software development, and Linux has become the backbone of the work. You can even develop desktop applications to manage SAS applications through Python and Linux.

SAS can also use the X statement and filename pipe statement to call python and Linux scripts. On the other hand, Python and Linux can also call SAS to complete automated operations to improve efficiency. This article also gives a briefly introduction on how SAS interacts with Python and Linux.

All in all, SAS calls R and Python to greatly enhance SAS data processing, statistical analysis, statistical graph and other functions. R and Python can also use SAS mature modules to enhance their data analysis capabilities. As an operating system, Linux provides the medium for SAS interaction with R and Python.

REFERENCES

[1] SAS/IML(R) 9.22 User's Guide:
https://support.sas.com/documentation/cdl/en/imlug/63541/HTML/default/viewer.htm#r_toc.htm

[2] %PROC_R: A SAS Macro that Enables Native R Programming in the Base SAS Environment: %PROC_R: A SAS Macro that Enables Native R Programming in the Base SAS Environment | Journal of Statistical Software (jstatsoft.org)

[3] SAS Using R Markdown:
https://www.ssc.wisc.edu/~hemken/SASworkshops/Markdown/SASmarkdown.html

[4] Using PROC FCMP Python Objects:
https://documentation.sas.com/doc/en/pgmsascdc/v_006/lecompobjref/p18qp136f91aaqn1h54v3b6pkant.htm#:~:text=%20Python%20Function%20Workflow%20in%20PROC%20FCMP%20,method%20to%20read%20Python%20source%20code...%20More%20

[5] SASpy documentation: SASPy — saspy 4.3.0 documentation (sassoftware.github.io)

[6] X Statement: SAS(R) 9.2 Language Reference: Dictionary, Fourth Edition

[7] Reading from and Writing to UNIX Commands (PIPE):
https://support.sas.com/documentation/cdl/en/hostunx/61879/HTML/default/viewer.htm#pipe.htm

CONTACT INFORMATION

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

Name:	Kaiping Yang
Enterprise:	BeiGene
Phone:	13588214900
E-mail:	kaiping.yang@beigene.com