Paper 3260-2019

Interaction between SAS® and Python for Data Handling and Visualization

Yohei Takanami, Takeda Pharmaceuticals

ABSTRACT

For drug development, SAS is the most powerful tool for analyzing data and producing tables, figures, and listings (TLF) that are incorporated into a statistical analysis report as a part of Clinical Study Report (CSR) in clinical trials. On the other hand, in recent years, programming tools such as Python and R have been growing up and are used in the data science industry, especially for academic research. For this reason, improvement in productivity and efficiency gain can be realized with the combination and interaction among these tools. In this paper, basic data handling and visualization techniques in clinical trials with SAS and Python, including pandas and SASPy modules that enable Python users to access SAS datasets and use other SAS functionalities, are introduced.

INTRODUCTION

SAS is fully validated software for data handling, visualization and analysis and it has been utilized for long periods of time as a de facto standard in drug development to report the statistical analysis results in clinical trials. Therefore, basically SAS is used for the formal analysis report to make an important decision. On the other hand, although Python is a free software, there are tremendous functionalities that can be utilized in broader areas. In addition, Python provides useful modules to enable users to access and handle SAS datasets and utilize SAS modules from Python via SASPy modules (Nakajima 2018). These functionalities are very useful for users to learn and utilize both the functionalities of SAS and Python to analyze the data more efficiently. Especially for Python users who are not familiar with or have to learn SAS code, SASPy modules are powerful tool that automatically generate and execute native SAS code via Jupyter Notebook bundled with Anaconda environment.

This paper is mainly focused on the basic functionalities, interaction and their differences between SAS and Python, therefore the advanced skills or functionalities for data handling and visualization are not covered. However, these introductions are useful for Python users who are not familiar with SAS code and vice versa. As shown in Figure 1, in this paper, it is assumed that the following versions of analysis environment are available on local PC (Windows 64 bit):

- Windows PC SAS 9.4M3
  - SAS code is executed in SAS environment
  - BASE SAS and SAS/STAT are available
- Anaconda 5.3.1 (Python 3.7)
  - Python code is executed in Jupyter Notebook
  - SASPy modules are executed in SAS session in Jupyter Notebook

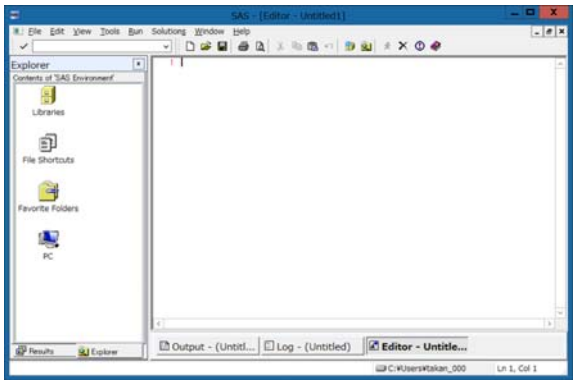
SAS 9.4	Python (Jupyter Notebook with Anaconda)
	

Figure 1. SAS and Python (Jupyter Notebook in Anaconda) Environment

Table 1 shows the basic data handling and visualization modules of SAS and Python. Although only SAS dataset format is used in SAS, there are multiple data formats used in Python such as Dataframe in Pandas module and Array in Numpy module.

	SAS	Python
Data Format	SAS dataset (Array data can be used in the DATA Step as a part of dataset)	Dataframe (Pandas module), Array (Numpy module)
Data Handling	DATA Step (e.g. MERGE statement) and PROC step (e.g. SORT procedure, TRANSPOSE procedure)	Pandas (e.g. DataFrame method, merge method,), Numpy (e.g. arange method)
Data Visualization	PROC step for Graphics Procedure (e.g. SGPLOT, SGPANEL)	Matplotlib (e.g. plot, hist, scatter), Pandas (e.g. plot), Seaborn (e.g. regplot)

Table 1. Basic SAS and Python Modules for Data Handling and Visualization

In addition to the basic modules used for data handling and visualization in SAS and Python, Python SASPy modules to realize interactive process between them are introduced in a later chapter.

DATA HANDLING

In SAS, mainly data are manipulated and analyzed in SAS dataset format. On the other hand, in Python, there are some data formats used for data handling and visualization. The Dataframe format that corresponds to the SAS dataset in terms of data structure is mainly used in this paper.

READ SAS DATASET IN PYTHON

Although various kinds of data format (e.g. Excel, Text) can be imported and used in SAS and Python, it is more convenient for users to utilize the SAS dataset directly in Python in

terms of the interaction between them. Python can read SAS datasets with Pandas modules that enable users to handle these data in Dataframe format. For example, the following Python code simply reads a SAS dataset, test.sas7bdat, and converts it to the Dataframe format with the read_sas method in Pandas module:

```
import pandas as pd
sasdt = pd.read_sas(r'C:\test\test.sas7bdat')
```

The test.sas7bdat is a simple dataset that includes only one row with three numeric variables, x, y and z.

	x	y	z
1	1	1	1

Figure 2. SAS Dataset "Test"

Table 2 shows a Python code and output in Jupyter Notebook. After converting SAS dataset to Dataframe format, Pandas modules can handle it without any SAS modules. Columns in Dataframe correspond to variables in SAS dataset.

In:	`# import the pandas modules`
	`import pandas as pd`
	`# Convert a SAS dataset 'test' to a Dataframe 'sasdt'`
	`sasdt = pd.read_sas(r'C:\test\test.sas7bdat')`
	`print(sasdt)`

Out:	`  x  y  z` `0  1.0  1.0  1.0`

Table 2. Conversion of SAS Dataset to Dataframe in Python

On the other hand, a Dataframe can be converted to a SAS dataset with the dataframe2sasdata() method in SASPy that is introduced in a later chapter:

```
# Export Dataframe to SAS dataset
import saspy
# Create SAS session
sas = saspy.SASsession()
# Create SAS library
sas.saslib('test', path="C:/test")
# Convert Dataframe to SAS dataset
sas.dataframe2sasdata(df=sasdt, table='test2', libref='test')
```

SAS library "test" that is used for storing a SAS dataset "test2" is created using the sas.saslib method and a SAS dataset "test2.sas7bdat" is actually created in "C:/test" folder as shown in Figure 3.

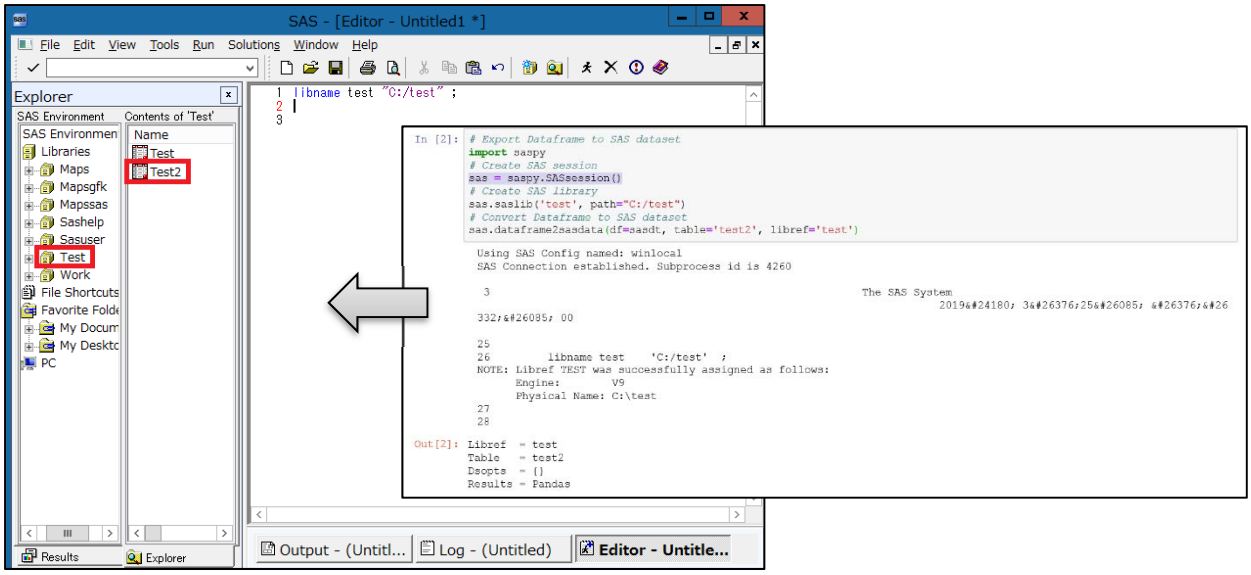


Figure 3. SAS Dataset "Test2" Converted from a Dataframe

DATA MANUPILATION IN SAS AND PYTHON

As shown in Table 1, for data handling, mainly the DATA step is used in SAS and Pandas and Numpy modules are used in Python. In this section, some major modules and techniques for data manipulation are introduced in SAS and Python:

- Creation of SAS dataset and Dataframe/Array
- Handling of rows and columns

Creation of SAS Dataset and Dataframe/Array

Table 3 shows the data creation with simple SAS and Python codes:

- SAS: Numeric and character variables are defined in the INPUT statement and data are listed in the CARDS statement. The PRINT procedure outputs the dataset "data1".
- Python: Pandas modules are imported and the DataFrame method is used to create a Dataframe and the print method is used to output the Dataframe "data1".

SAS Dataset	Python Dataframe

`data data1 ;`
`  input a b $ ;`
`cards;`
`1 xxx`
`2 yyy`
`; run ;`
`proc print data=data1 ; run ;`
	`# Dataframe with numeric and character variables`
	`import pandas as pd`
	`data1 = pd.DataFrame([[1,'xxx'],[2,'yyy']],`
	`  columns=['a', 'b'])`
	`print(data1)`

Output	
Obs a b 1 1 xxx 2 2 yyy	a b 0 1 xxx 1 2 yyy

Table 3. Creation of SAS dataset in SAS and Dataframe in Python

In Python, it should be noted that the row numbers are presented with data as shown in Table 3 where the number begins with 0. This rule is applied to the element of data such as Pandas Dataframe and Numpy Array. For example, data1.loc[1,'a'] extracts 2, the value of the 2nd row of column 'a' in the Dataframe data1.

As shown in Table 4, a SAS dataset and a Dataframe can be created more efficiently with other functionalities:

- In SAS, the DO statement is used to generate consecutive values
- In Python, firstly the array data are created with the arange method followed by the conversion to a Dataframe with the DataFrame and T methods. The T method transposes the Dataframe after combining col1 and col2 array data.

SAS Dataset creation	Dataframe and Array in Python

`data data2 ;`
`  do a = 1 to 3 ;`
`  b = a*2 ;`
`  output ;`
`  end ;`
`run ;`
`proc print data=data2 ; run ;`
	`import pandas as pd`
	`import numpy as np`
	`# Create Array with Numpy module`
	`col1 = np.arange(1,4,1) # 1 to 3 by 1`
	`col2 = col1*2`
	`# Convert Array to Dataframe`
	`data2 = pd.DataFrame([col1,col2]).T`
	`data2.columns=['a','b']`
	`print(data2)`

Output	
Obs a b 1 1 2 2 2 4 3 3 6	a b 0 1 2 1 2 4 2 3 6

Table 4. Creation of SAS Dataset, Dataframe and Array

Handling of rows and columns

Granted that a SAS dataset or Dataframe is successfully created, data transformation may be needed prior to the data visualization or analysis process. The following data handling techniques are introduced here:

- Addition and Extraction of Data
- Concatenation of SAS Datasets/Dataframe
- Handling of Missing Data

Addition and Extraction of Data

The following example shows the addition of new variables/columns to SAS dataset/ Dataframe with simple manipulation.

SAS Dataset creation	Dataframe and Array in Python

data data2 ;
  set data2 ;
  c = a + b ; *--- New variable ;
run ;
proc print data=data2 ; run ;
	# New column
	data2['c']=data2['a']+data2['b']
	print(data2)

Output	
Obs a b c 1 1 2 3 2 2 4 6 3 3 6 9	a b c 0 1 2 3 1 2 4 6 2 3 6 9

Table 5. Addition of New Variables/Columns

As shown in Table 6. Rows/records that meet specific conditions ("a" equals 2 or 3) can be extracted with logical operators in SAS and Python, respectively.

SAS Dataset creation	Dataframe and Array in Python

data data2_ex ;
  set data2 ;
  where a=2 or a=3 ;
run ;
proc print data=data2_ex ;
run ;
	# Extract the records where a=2 or 3
	data2_ex=data2[(data2.a==2) | (data2.a==3)]
	print(data2_ex)

Output	
Obs a b c 1 2 4 6 2 3 6 9	a b c 1 2 4 6 2 3 6 9

Table 6. Extraction of Rows/Records

Basic logical and arithmetic operators of SAS and Python are shown in Table 7. The DO statement and the 'for' operator are used to iterate specific programming logic in SAS and Python, respectively. Most of the basic arithmetic operators are similar between them.

SAS Operators	Python Operators

*--- DO and IF statement ;
do i = 1 to 3 ;
  if i = 1 then y = 1 ;
  else if i = 2 then y = 2 ;
  else y = 3 ;
end ;
	# for and if operators
	x = [1,2,3]
	for i in x:
	  if i == 1:
	  print('i=',1)
	  elif i == 2:
	  print('i=',2)
	  else:
	  print('i=',3)

*--- DO statement with decimal numbers ;
do i = 10 to 11 by 0.1 ;
  output ;
end ;
	# for operators with decimal numbers
	for x in range(10, 12, 1):
	  for y in [0.1*z for z in range(10)]:
	  x1 = round(x + y,1)

*--- Arithmetic operators ;
data xxx ;
  x1 = 13 ;
  x2 = x1+3 ;
  x3 = x1-3 ;
  x4 = x1*3 ;
  x5 = round(x1/3, .001) ;
  x6 = int(x1/3) ;
  x7 = mod(x1,3) ;
  x8 = x1**3 ;
run ;
	# Arithmetic operators
	
	x1 = 13
	x2 = x1+3
	x3 = x1-3
	x4 = x1*3
	x5 = round(x1/3, 3)
	x6 = x1//3 # divmod(x1,3) returns (4, 1)
	x7 = x1%3
	x8 = x1**3

Results: 13 16 10 39 4.333 4 1 2197	

Table 7. Basic Logical and Arithmetic Operators in SAS and Python

Concatenation of SAS Dataset/Dataframe

SAS and Python have various kinds of functionalities to concatenate SAS datasets and Dataframes, respectively. In this section, the concat and the merge methods in Pandas modules that correspond to the SET and the MERGE statements in SAS are introduced:

- The SET statement and the MERGE statement in SAS are basically used to combine the dataset in vertical and horizontal manner, respectively.
- The concat method with the "axis" option (1: Horizontal, 0: Vertical) and the merge method with the "on" and "how" options in Pandas modules are used to combine Dataframes in both vertical and horizontal ways.

As shown in Table 8, the missing values (dot (.) in SAS numeric variables, NaN in Python numeric columns) are generated if there are no data correspond to that in another dataset/Dataframe.

SAS Dataset concatenation (Horizontal)

```
data data3 ;
   input d e f ;
cards;
1 2 3
4 5 6
7 8 9
;
run ;
*--- Merge the datasets ;
data data4 ;
   merge data2 data3 ;
run ;
proc print ; run ;
```

Dataframe concatenation (Horizontal)

```
data3 = pd.DataFrame(                    \
        np.arange(1,10,1).reshape(3,3), \
        columns=['d','e','f']
        )
print(data3)

# Horizontal concatenation with axis=1
data4 = pd.concat([data2,data3],axis=1)
print(data4))
```

Output

Obs	a	b	c	d	e	f
1	1	2	3	1	2	3
2	2	4	6	4	5	6
3	3	6	9	7	8	9

```
   d  e  f
0  1  2  3
1  4  5  6
2  7  8  9

   a  b  c  d  e  f
0  1  2  3  1  2  3
1  2  4  6  4  5  6
2  3  6  9  7  8  9
```

SAS Dataset concatenation (Vertical)

```
*--- Vertical concatenation ;
data data5 ;
   set data2 data3 ;
run ;
proc print ; run ;
```

Dataframe concatenation (Vertical)

```
# Vertical concatenation with axis=0
data5 = pd.concat([data2,data3],axis=0)
print(data5)
```

Output

Obs	a	b	c	d	e	f
1	1	2	3	.	.	.
2	2	4	6	.	.	.
3	3	6	9	.	.	.
4	.	.	.	1	2	3
5	.	.	.	4	5	6
6	.	.	.	7	8	9

```
     a    b    c    d    e    f
0  1.0  2.0  3.0  NaN  NaN  NaN
1  2.0  4.0  6.0  NaN  NaN  NaN
2  3.0  6.0  9.0  NaN  NaN  NaN
0  NaN  NaN  NaN  1.0  2.0  3.0
1  NaN  NaN  NaN  4.0  5.0  6.0
2  NaN  NaN  NaN  7.0  8.0  9.0
```

Table 8. Simple Concatenation of Dataset and Dataframe

The MERGE statement in SAS is very useful and frequently used to combine SAS datasets with key variables such as subject ID in clinical trials. There is the merge method with how='outer' option in Pandas modules that realizes the similar functionalities to the MERGE statement in SAS. The missing values are generated on the records where the key variable does not match each other.

SAS Dataset concatenation (MERGE)	Dataframe concatenation (merge)

*--- Rename and Merge with key ;
data data3_r ;
set data3 ;
rename d = a ; *--- key ;
run ;
data data6 ;
merge data2 data3_r ;
by a ;
run ;
proc print ; run ;
	# Rename and Merge with key
	data3_r = data3.rename(\
	index=str,columns={'d':'a'} \
	)
	data6=pd.merge(\
	data2,data3_r,on='a',how='outer' \
	)
	print(data6)

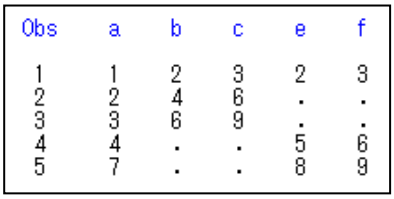
Output	
	a b c e f 0 1 2.0 3.0 2.0 3.0 1 2 4.0 6.0 NaN NaN 2 3 6.0 9.0 NaN NaN 3 4 NaN NaN 5.0 6.0 4 7 NaN NaN 8.0 9.0

Table 9. Merge SAS Datasets/Dataframes with Key Variables

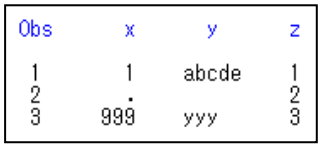
Handling of Missing data

Missing data is possibly included in database of clinical trials due to issues related to data collection such as subject withdrawal, no data entry via electronic devices and pre-specified data entry rules. In such cases, missing data imputation would be needed before the data analysis (e.g. partial date, Last observation carried forward (LOCF)). Some of the basic functionalities for handling of missing data are introduced in this section:

- SAS: Missing value of numeric and character variable is dot (.) and null character (‘’), respectively
- Python: Missing value of numeric and character column is NaN and None, respectively. The fillna method imputes missing values with specified ones (e.g. 999, 'yyy') for each column.

Handling of Missing Data in SAS	Handling of Missing Data in Python

data Missing ;
x = 1 ; y = "abcde" ; z = 1 ;
output ;
*--- Replaced by missing values ;
call missing(x) ; *--- x = . ;
call missing(y) ; *--- y = "" ;
z = 2 ;
output ;
*--- Impute missing values ;
if missing(x) then x = 999 ;
if missing(y) then y = 'yyy' ;
z = 3 ;
output ;
run ;
	import pandas as pd
	import numpy as np
	
	# Insert missing values
	m1=pd.DataFrame({'x':[1],'y':['abc'],'z':[1]})
	m2=pd.DataFrame({'x':np.nan,'y':None,'z':[2]})
	m3=pd.DataFrame({'x':np.nan,'y':None,'z':[3]})
	
	# Impute missing values
	m3 = miss3.fillna({'x':999, 'y':'yyy'})
	
	# concatenate data vertically
	miss = pd.concat([miss1,miss2,miss3],axis=0)
	print(miss)

Handling of Missing Data in SAS	Handling of Missing Data in Python
	

Obs	x	y	z
1	1	abcde	1
2	.		2
3	999	yyy	3

```
      x      y  z
0   1.0  abcde  1
0   NaN   None  2
0 999.0    yyy  3
```

Table 10. Handling of Missing Data in SAS and Python

DATA VISUALIZATION

SAS and Python have many graphics functionalities for data visualization. In this chapter, some of the most commonly used graphs in clinical trials are introduced using the SGPLOT procedure in SAS and the Pandas/Matplotlib/Seaborn modules in Python:

- Mean and SD plot over time
- Histogram
- Scatter plot with regression line
- Bar chart

TEST DATA

A SAS dataset ADEFF that conforms to the CDISC ADaM standard format is used for data visualization in this section. The CDISC standard is required format of clinical study data for electronic data submission to regulatory agency (i.e. FDA, PMDA) for New Drug Application (NDA). See the CDISC web site (https://www.cdisc.org/standards/foundational/adam) for further details of data structure and rules of ADaM standard.

SAS Dataset	ADEFF	
Description	• Efficacy dataset to populate Laboratory Test results to diagnose the healing of Disease A in Study-XXX • Two treatment arms: Drug A and Drug B • 8 weeks treatment period	
Variables and Contents	TRTP/TRTPN	Treatment group code and description (Drug A or Drug B)
	PARAM/PARAMCD	Parameter code and abbreviation for Laboratory Test results (PARAMCD: "TESTRES") and Binary data (PARAMCD: "HEAL")
	AVAL/AVALC	Numeric variable of Laboratory Test results (Continuous values) and Binary data (Disease A is healed: 1 or unhealed: 2)
	AVISIT/AVISITN:	Analysis Visits (e.g. Baseline, Week 8) in the study
	FASFL:	'Y' indicates the Full Analysis Set that is used for main analysis (the condition the FASFL equals 'Y' is omitted in the following examples)

Image

	USUBJID	TRTP	TRTPN	FASFL	PARAM	PARAMCD	AVAL	AVISIT	AVISITN
1	PROD-XXX/STUDY-XXX/9001001	Drug A	1	Y	Healing of Disease A	HEAL	2	Week 2	2
2	PROD-XXX/STUDY-XXX/9001001	Drug A	1	Y	Test Results	TESTRES	38.3	Week 2	2
3	PROD-XXX/STUDY-XXX/9001001	Drug A	1	Y	Healing of Disease A	HEAL	1	Week 4	3
4	PROD-XXX/STUDY-XXX/9001001	Drug A	1	Y	Test Results	TESTRES	16.2	Week 4	3
5	PROD-XXX/STUDY-XXX/9001001	Drug A	1	Y	Healing of Disease A	HEAL	1	Week 8	4
6	PROD-XXX/STUDY-XXX/9001001	Drug A	1	Y	Test Results	TESTRES	27	Week 8	4
7	PROD-XXX/STUDY-XXX/9001001	Drug A	1	Y	Test Results	TESTRES	35.3	Baseline	1
8	PROD-XXX/STUDY-XXX/9001002	Drug A	1	Y	Healing of Disease A	HEAL	2	Week 2	2
9	PROD-XXX/STUDY-XXX/9001002	Drug A	1	Y	Test Results	TESTRES	29.2	Week 2	2
10	PROD-XXX/STUDY-XXX/9001002	Drug A	1	Y	Healing of Disease A	HEAL	1	Week 4	3
11	PROD-XXX/STUDY-XXX/9001002	Drug A	1	Y	Test Results	TESTRES	13	Week 4	3
12	PROD-XXX/STUDY-XXX/9001002	Drug A	1	Y	Healing of Disease A	HEAL	1	Week 8	4
13	PROD-XXX/STUDY-XXX/9001002	Drug A	1	Y	Test Results	TESTRES	13	Week 8	4
14	PROD-XXX/STUDY-XXX/9001002	Drug A	1	Y	Test Results	TESTRES	28.9	Baseline	1
15	PROD-XXX/STUDY-XXX/9001003	Drug B	2	Y	Healing of Disease A	HEAL	2	Week 2	2
16	PROD-XXX/STUDY-XXX/9001003	Drug B	2	Y	Test Results	TESTRES	46.5	Week 2	2
17	PROD-XXX/STUDY-XXX/9001003	Drug B	2	Y	Healing of Disease A	HEAL	1	Week 4	3
18	PROD-XXX/STUDY-XXX/9001003	Drug B	2	Y	Test Results	TESTRES	16.4	Week 4	3
19	PROD-XXX/STUDY-XXX/9001003	Drug B	2	Y	Healing of Disease A	HEAL	1	Week 8	4
20	PROD-XXX/STUDY-XXX/9001003	Drug B	2	Y	Test Results	TESTRES	19.3	Week 8	4

Table 11. SAS Dataset ADEFF

MEAN AND SD PLOT

Mean and SD plot for Laboratory Test results can be produced using the SGPLOT procedure and Pandas plot method in SAS and Python, respectively. Assuming that the Dataframe adeff2 is created after reading SAS dataset ADEFF (stored in "C:\test" folder) prior to the graph creation in Python.

```
#read SAS dataset
import pandas as pd
adeff2 = pd.read_sas('C:\test\adeff.sas7bdat')
```

- SAS: the VLINE statement with RESPONSE, GROUP, STAT and LIMITSTAT options are executed to calculate the mean and the standard deviation by study visits and treatment groups and generates the Mean and SD plot. The MARKERS/MAKERATTRS and the LINEATTRS options control the symbol and line pattern of the plot. The XAXIS/YAXIS and the KEYLEGEND statements control the appearance of each axis and legend, respectively.
- Python: the mean and the std methods in Pandas modules are used to calculate the mean and the standard deviation. The plot method with the yerr option generates the mean and sd plot. The fmt option controls the appearance of symbol and line. The xticks/yticks and the legend methods control the appearance of each axis and legend, respectively.

SAS with PROC SGPLOT

```
*--- Format for Study Visit ;
proc format ;
   value _VITF 1 = 'Baseline' 2 = 'Week 2' 3 = 'Week 4' 4 = 'Week 8' ;
run ;
title "Test Results" ;
proc sgplot data=ADEFF ;
   where PARAMCD = 'TESTRES' ;
   vline AVISITN / response=AVAL group=TRTP stat=mean limitstat=stddev limits=both
                   markers markerattrs=(symbol=circlefilled) lineattrs=(pattern=1);
   xaxis label='Study Visit' ;
   yaxis display=(nolabel) ;
   keylegend / title="Treatment group" position=topright location=inside down=2 ;
   format AVISITN _VITF. ;
run ;
```

Python with Pandas plot

```
import matplotlib as mpl; import matplotlib.pyplot as plt
import numpy as np; import pandas as pd
fig, ax = plt.subplots(figsize=(8,5))
# Calculate SD
yerror = adeff2[adeff2.PARAMCD=='TESTRES']            \
         .groupby(['AVISITN','TRTP'])['AVAL'].std().unstack()
# Calculate Mean by visits, treatment groups and output plot with error bar of SD
adeff2[adeff2.PARAMCD=='TESTRES']                     \
.groupby(['AVISITN','TRTP'])['AVAL'].mean().unstack() \
.plot(yerr=yerror,ax=ax,fmt='-o',capsize=3)
# Ticks, legend, label and title
plt.yticks([10,20,30,40])
plt.xticks(np.arange(1,5,1), ('Baseline','Week 2','Week 4', 'Week 8'))
plt.legend(title='Treatment group',loc='upper right',frameon=False,ncol=1)
plt.xlabel('Study Visit')
plt.title('Test Results')
```

Output (SAS and Python)

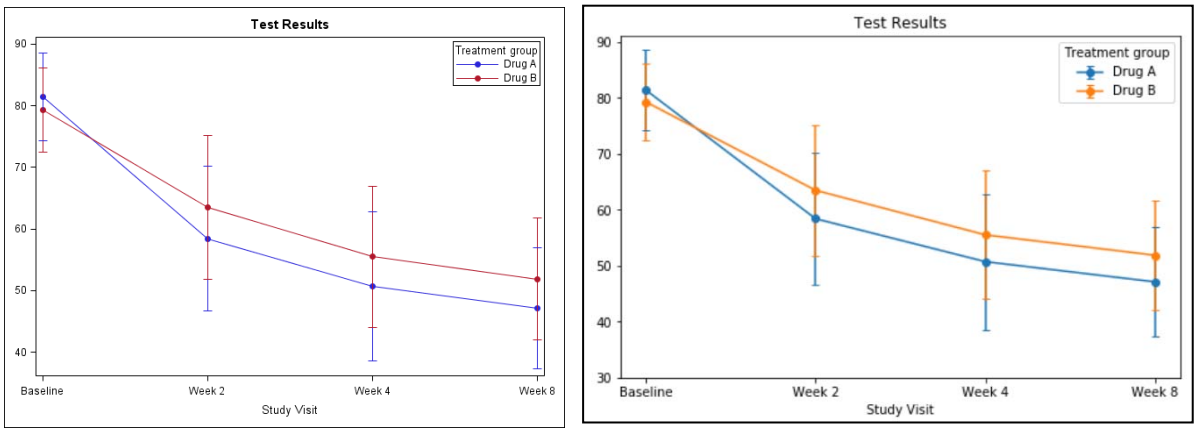


Figure 4. Mean and SD Plot Created by SAS and Python

HISTOGRAM

Histogram is produced with the PROC SGPLOT in SAS as well as the Mean and SD plot. In Python, there are many functionalities in the Matplotlib modules and the hist method is used for the creation of Histogram. In this section, a Histogram is produced to make sure the distribution of the laboratory test results at week 8 for each treatment group.

- SAS: the HISTOGRAM statement with the GROUP option to generate a histogram by treatment groups. The number of bins can be specified with the NBINS option. The TRANSPARENCY and NOOUTLINE options adjust the transparency of histogram and control the appearance of outline, respectively.
- Python: the hist method is used to create a Histogram. The range and the bins options control the data range and the number of bins. The alpha option controls the transparency of histogram. The style.use method enables users to utilize the high quality graphics style such as 'ggplot' and 'classic'. All the available styles in graphics can be output with "print(plt.style.available)".

SAS with PROC SGPLOT

```
title 'Distribution of Test Results';
proc sgplot data=ADEFF ;
   where PARAMCD = 'TESTRES' and AVISITN = 4 ;
   histogram AVAL / group=TRTP name='a' transparency=0.6
                    nooutline nbins=14 scale=count ;
   keylegend 'a' / location=inside position=topright across=1 noborder ;
   yaxis label='Percentage' grid ;
   xaxis display=(nolabel) ;
run;
```

Python with Matplotlib Hist

```
plt.style.use('ggplot') # High quality graphics style is available
fig, ax = plt.subplots(figsize=(7,5))
# Create Hisotgrams by treatment groups
plt.hist(adeff2['AVAL'][(adeff2.PARAMCD=='TESTRES') & (adeff2.TRTPN==1) & \
                        (adeff2.AVISITN==4)],                            \
         range=(20,90),bins=14, color='b', alpha=0.5)
plt.hist(adeff2['AVAL'][(adeff2.PARAMCD=='TESTRES') & (adeff2.TRTPN==2) & \
                        (adeff2.AVISITN==4)],                             \
         range=(20,90),bins=14, color='r', alpha=0.5)
plt.legend(['Drug A','Drug B'],title='Treatment group',loc='upper right', \
           frameon=True,ncol=1)
plt.title('Distribution of Test Results')
```

Output (SAS and Python)

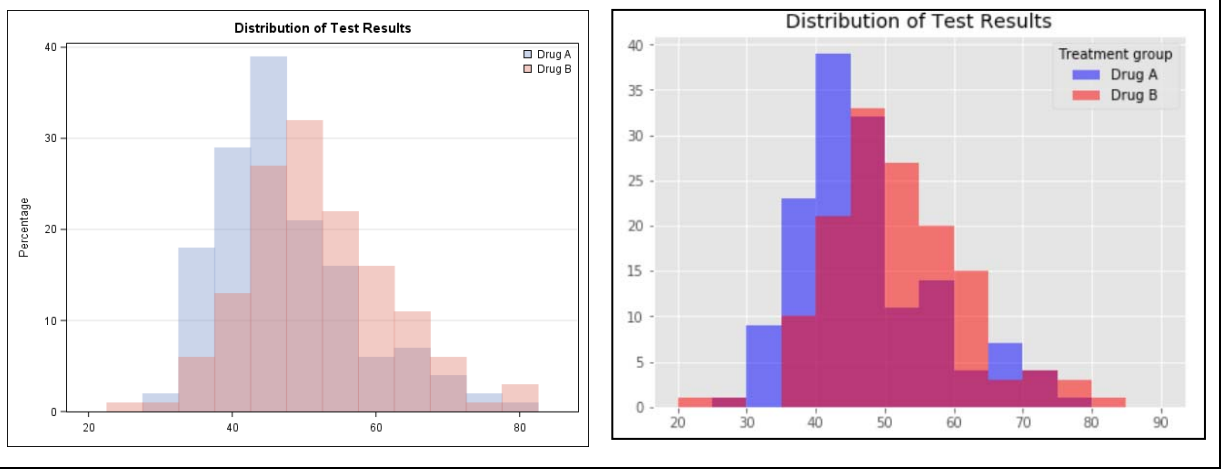


Figure 5. Histogram Created by SAS and Python

SCATTER PLOT WITH REGRESSION LINE

Scatter plot and regression line are produced using the PROC SGPLOT in SAS and the regplot method of the Seaborn modules in Python. Before the creation of graphs, ADEFF should be transformed into dataset _SCATTER in order to have two variables for baseline and post-baseline data using the MERGE statement in SAS. On the other hand, the baseline and post-baseline data can be extracted directly in the regplot method in Python.

- SAS: X-axis and Y-axis variables are specified in the REG statement with the GROUP option to generate a scatter plot with regression line by treatment groups. The CLM option generates the

confidence limits for the mean predicted values.

- Python: the regplot method generates a scatter plot with regression line and the confidence limits.

SAS with PROC SGPLOT

```
data _Scatter ; *--- Create two variables, BASE and POST ;
   merge ADEFF(where=(PARAMCD = 'TESTRES' and AVISITN = 1)
               rename=(AVAL=BASE) keep=USUBJID AVAL PARAMCD AVISITN TRTP TRTPN)
         ADEFF(where=(PARAMCD = 'TESTRES' and AVISITN = 4)
               rename=(AVAL=POST) keep=USUBJID AVAL PARAMCD AVISITN) ;
   by USUBJID ;
   keep USUBJID TRTP TRTPN BASE POST ;
run ;
proc sgplot data=_Scatter ; *--- Scatter plot with Regression line ;
   reg x=base y=post / group=TRTP lineattrs=(pattern=1) clm name='a'
                      markerattrs=(symbol=circlefilled) ;
   xaxis label='Baseline' ;
   yaxis label='Week 8' ;
   keylegend 'a' / title='' noborder location=inside position=topright ;
run ;
```

Python with Seaborn Regplot

```
import seaborn as sns
plt.style.use('ggplot')
fig=plt.figure(figsize=(5,3))
sns.regplot(x=adeff2['AVAL'][(adeff2.PARAMCD=='TESTRES') & (adeff2.TRTPN==1) \
                             & (adeff2.AVISITN==1)],                          \
            y=adeff2['AVAL'][(adeff2.PARAMCD=='TESTRES') & (adeff2.TRTPN==1) \
                             & (adeff2.AVISITN==4)],                          \
            data=adeff2, color='b')
sns.regplot(x=adeff2['AVAL'][(adeff2.PARAMCD=='TESTRES') & (adeff2.TRTPN==2) \
                             & (adeff2.AVISITN==1)],                          \
            y=adeff2['AVAL'][(adeff2.PARAMCD=='TESTRES') & (adeff2.TRTPN==2) \
                             & (adeff2.AVISITN==4)],                          \
            data=adeff2, color='r')
plt.legend(['Drug A','Drug B'],loc='upper right',frameon=False,ncol=2)
plt.xlabel('Baseline'); plt.ylabel('Week 8'); plt.yticks(np.arange(20,110,10))
```

Output (SAS and Python)

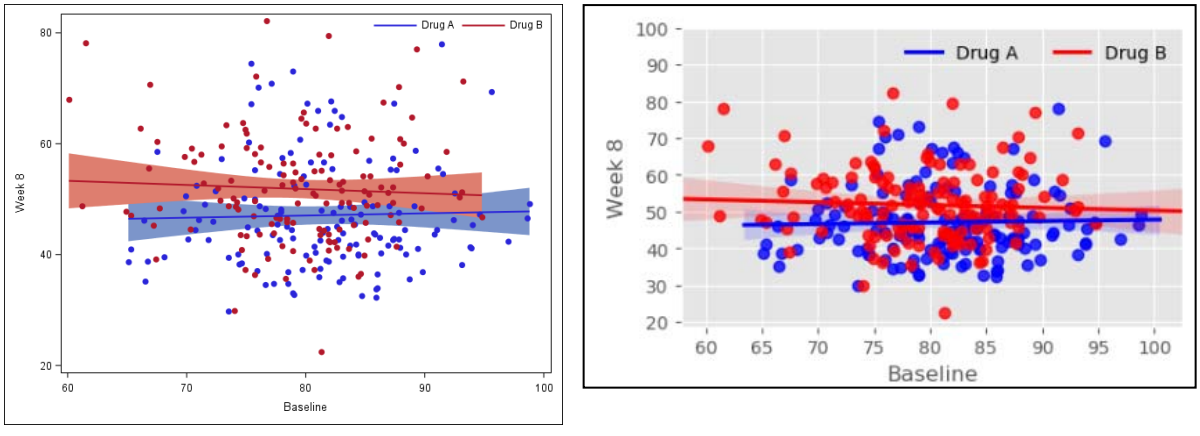


Figure 6. Scatter Plot with Regression Line Created by SAS and Python

BAR CHART

A Bar chart for the healing rate of the Disease A by study visits is created using the PROC SGPLOT and the bar method in the Matplotlib modules. The percentages of the subjects with the healing of Disease A should be calculated prior to the creation of graphs with the FREQ procedure and the count method of Pandas modules in SAS and Python, respectively.

SAS with PROC SGPLOT

```
ods output crosstabfreqs=_freq(where=(AVAL=1 and _TYPE_='111')) ;
proc freq data=ADEFF ;
   where PARAMCD = 'HEAL' ;
   table AVISITN*TRTP*AVAL / nocol nopercent ; *--- Calculate the healing rate ;
run ; ods output close ;
title 'Healing rate (%) of Disease A' ;
proc sgplot data=_freq ; *--- Create Bar chart with VBAR statement ;
   vbar AVISITN / response=RowPercent group=TRTP groupdisplay=cluster
                  clusterwidth=0.7 barwidth=0.9 name='a' ;
   keylegend 'a' / title='' location=inside position=topleft noborder ;
   yaxis values=(0 to 100 by 10) display=(nolabel) ; xaxis display=(nolabel) ;
  format AVISITN _VITF. ; run ;
```

Python with Matplotlib Bar

```
# Calculate the healing rate by Study visits and treatment group
nume  = adeff2[(adeff2.PARAMCD=='HEAL') & (adeff2.AVAL == 1)] \
        .groupby(['AVISITN','TRTP'])['AVAL'].count()
denom = adeff2[adeff2.PARAMCD=='HEAL'].groupby(['AVISITN','TRTP'])['AVAL'].count()
rate  = 100*pd.DataFrame(nume/denom,columns=['AVAL'])
rate.reset_index(level=['AVISITN','TRTP'],inplace=True)
# Create Bar chart with bar method
plt.bar(rate['AVISITN'][rate.TRTP=='Drug A']-0.16, \
        rate['AVAL'][rate.TRTP=='Drug A'], width=0.3,color='b',linewidth=1)
plt.bar(rate['AVISITN'][rate.TRTP=='Drug B']+0.16, \
        rate['AVAL'][rate.TRTP=='Drug B'], width=0.3,color='r',linewidth=1)
plt.xticks(np.arange(2,5,1), ('Week 2','Week 4', 'Week 8'))
plt.yticks(np.arange(0,110,10)); plt.title('Healing rate (%) of Disease A')
plt.legend(['Drug A','Drug B'],loc='upper left',frameon=False,ncol=2)
```

Output (SAS and Python)

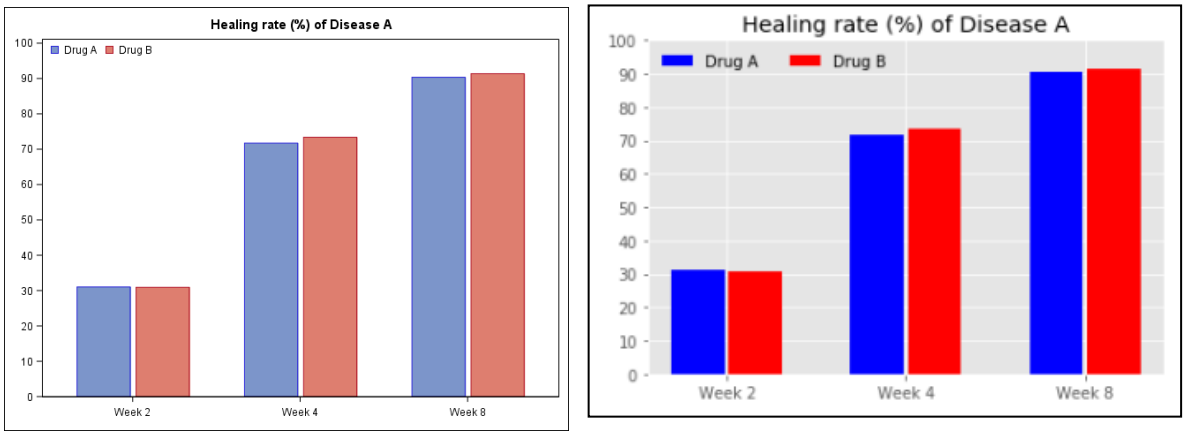


Figure 7. Bar Chart Created by SAS and Python

INTERACTION BETWEEN SAS AND PYTHON WITH SASPY MODULE

In this chapter, the SASPy modules are introduced including the setup in Anaconda environment. As mentioned above, SAS 9.4 and Anaconda 5.3.1 should be installed in local PC prior to the setup of the SASPy module. With the SASPy modules, SAS modules can be executed with simple codes in Python environment. In addition, the 'teach_me_SAS' method that outputs the actual SAS codes is also introduced. This method is very helpful for Python users to learn the SAS coding rules in Python environment.

SETUP OF SASPY MODULE IN PYTHON

Firstly users should install the SASPy and the sas_kernel modules with Anaconda Prompt as follows:

```
e.g., conda install saspy, pip install sas_kernel
```

Users also need to add a path of SAS-installed folder to the Environment Variables in the System Properties setting dialog in local PC. It should be noted that the path depends on the environment in the local PC setting.

```
e.g., C:\Program Files\SASHome\SASFoundation\9.4\core\sasext
```

The sascfg.py, a configuration file of the Saspy modules, should be modified based on the default setting in local PC. The location of sascfg.py can be output with the sascfg method in the SASPy modules (the file path also depends on local PC environment).

In:	import saspy saspy.sascfg
Out:	<module 'saspy.sascfg' from 'C:\\Users\\<*user*>\\AppData\\Local\\Continuum\\anaconda3\\lib\\ site-packages\\saspy\\sascfg.py'>

Table 12. Example of Location of sascfg.py File

The sascfg.py should be modified in the following contents:

- The parameter of the SAS_config_names should be changed to 'winlocal'.
- The setting of ‘winlocal’ (e.g. ‘java’ path, ‘encoding’ to be ‘utf-8’) should be modified based on the local environment
- The file paths for .jar files should be modified based on the local Java setting.

Figure 8. Image of Modification of sascfg.py file

See the web site (https://sassoftware.github.io/saspy/install.html) for more details of installation and configuration of the SASPy modules. Once the setup process is successfully completed, the SASPy modules become available and executable in Python environment.

EXECUTION OF SASPY MODULES IN PYTHON

After the setup of the SASPy modules, a SAS session can be created with the SASsession method as shown in Table 13 in Jupyter Notebook. The SASPy modules can be executed as well as other packages and modules in Python until the SAS session is closed.

In:	`import saspy`
	`sas=saspy.SASsession()`
	`print(sas)`

Out:	`Using SAS Config named: winlocal`
	`SAS Connection established. Subprocess id is` *`xxxxx`*
	`Access Method = IOM`
	`SAS Config name = winlocal`
	`WORK Path = /`
	`SAS Version = 9.04.01xxxxxxxxxxxx`
	`SASPy Version = 2.3.0`
	`Teach me SAS = False`
	`Batch = False`
	`Results = Pandas`
	`SAS Session Encoding = UTF-8`
	`Python Encoding value = utf-8`

Table 13. Create SAS Session in SASPy modules

A SAS session is terminated with the _endsas() method.

In:	`sas._endsas()`
Out:	`SAS Connection terminated. Subprocess id was xxxx`

Table 14. Termination of SAS Session in Python

Some of the basic functionalities of SASPy modules are introduced in this section:

- Read SAS dataset and Output SAS dataset information
- Summary statistics and Statistical procedure
- Creation of graphs
- Output actual SAS code with ‘teach_me_SAS’ method
- Submit actual SAS code to SAS session

Read SAS dataset and Output SAS dataset information

With the sasdata method, a SAS dataset can be imported to Python environment as the sasdata object. There are many test SAS datasets stored in 'sashelp' library in SAS environment and the CLASS dataset is one of them. In Table 15, the CLASS dataset is

displayed with the head method where the SAS PRINT procedure is executed in SAS session.

In:
```
dt = sas.sasdata('class','sashelp')
dt.head()
```

Out:

	Name	Sex	Age	Height	Weight
0	Alfred	M	14	69.0	112.5
1	Alice	F	13	56.5	84.0
2	Barbara	F	13	65.3	98.0
3	Carol	F	14	62.8	102.5
4	Henry	M	14	63.5	102.5

Table 15. Read and Output SAS Dataset with Sasdata and Head Methods of SASPy

SAS dataset CARS is also one of the datasets stored in SAS library 'sashelp'. As shown in Table 16, the information of the dataset can be obtained such as variable name, label, type and the number of missing values by using the info method,.

In:
```
cars=sas.sasdata('cars','sashelp')
cars.info()
```

Out:

	Type	Variable	N	Nmiss
0	char	Make	428	0
1	char	Model	428	0
2	char	Type	428	0
3	char	Origin	428	0
4	char	DriveTrain	428	0
5	numeric	MSRP	428	0
6	numeric	Invoice	428	0
7	numeric	EngineSize	428	0
8	numeric	Cylinders	428	2
9	numeric	Horsepower	428	0
10	numeric	MPG_City	428	0
11	numeric	MPG_Highway	428	0
12	numeric	Weight	428	0
13	numeric	Wheelbase	428	0
14	numeric	Length	428	0

Table 16. SAS Dataset Information with Info Method

Summary Statistics and Statistical Procedure

The MEANS procedure calculates summary statistics for numeric variables in a SAS dataset. SASPy provides the describe and the means method to execute the MEANS procedure in a SAS session and output the results in the Jupyter Notebook. In Table 17, the SAS dataset ADEFF is read and the summary statistics is calculated for the laboratory test results at week 8 by treatment groups. The results are stored in the res1 and the res2 as Pandas Dataframe format.

In:

```
sas.saslib('test', path='C:/test')
adeff=sas.sasdata('adeff','test')
# Execution of the MEANS procedure in SAS
res1=adeff.where('TRTPN=1 and AVISITN=4 and PARAMCD="TESTRES"').means()
res2=adeff.where('TRTPN=2 and AVISITN=4 and PARAMCD="TESTRES"').means()
res1_2=res1[res1.Variable=='AVAL'].set_index('Variable')
res2_2=res2[res1.Variable=='AVAL'].set_index('Variable')
# Output the results
print('Drug A \n',res1_2[['N', 'Mean', 'StdDev', 'Min', 'Median', 'Max']])
print('Drug B \n',res2_2[['N', 'Mean', 'StdDev', 'Min', 'Median', 'Max']])
```

Out:

```
Drug A
              N        Mean     StdDev   Min  Median  Max
Variable
AVAL      145  47.102759  9.824042  29.8      45   78
Drug B
              N        Mean     StdDev   Min  Median   Max
Variable
AVAL      139  51.865468  9.876592  22.5    50.9  82.2
```

Table 17. Summary Statistics with the MEANS Procedure in SAS Session

In addition to the MEANS procedure, some of the SAS statistical procedures are available in SASPy. For example, the TTEST procedure is executable with the ttest method. In Table 18, two-sample t-test is performed for the laboratory test results at week 8 for the comparison between treatment groups. The results are stored in the ttest as Pandas Dataframe format. The TTEST procedure produces several outputs and 'TTEST' is one of them that stores the results of the t-test such as t-value and p-value.

In:

```
# Create an object of sasstat
stat=sas.sasstat()
adeff_w8=adeff.where('AVISITN=4 and PARAMCD="TESTRES"')
# Execute the TTEST procedure with ttest meshod in SASPy
ttest = stat.ttest(data=adeff_w8,var='AVAL',cls='TRTP')
# Output the result of 2-sample t-test
ttest.TTESTS
```

Out:

	Variable	Method	Variances	tValue	DF	Probt
0	AVAL	Pooled	Equal	-4.073420	282.000000	0.00006
1	AVAL	Satterthwaite	Unequal	-4.072959	281.358593	0.00006

Table 18. T-test with the TTEST Procedure in SAS Session

Creation of Graphs

The SAS graphics procedures such as the SGPLOT procedure are available in SASPy. In Figure 9, some examples of graphs produced by the PROC SGPLOT in SASPy are introduced. The CLASS dataset is used for the creation of these graphs. It should be noted that only the basic options of the PROC SGPLOT are available in SASPy modules at this point.

`# Histogram`
`dt=sas.sasdata('class','sashelp')`
`dt.hist(var='age',title='Histogram')`
	`# Bar chart`
	`dt.bar(var='SEX')`

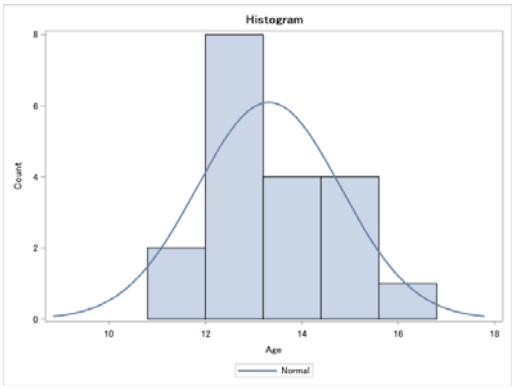
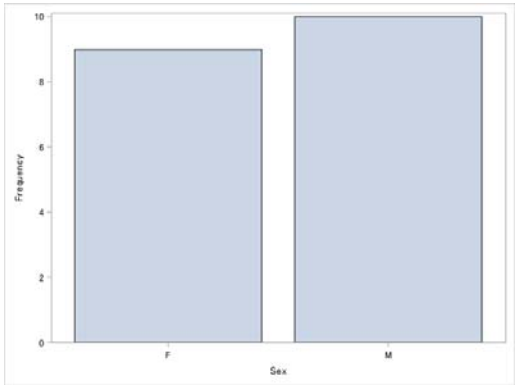
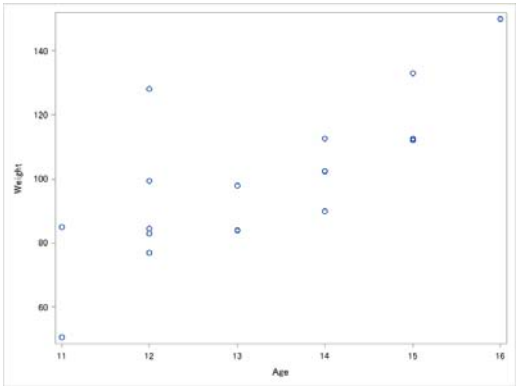
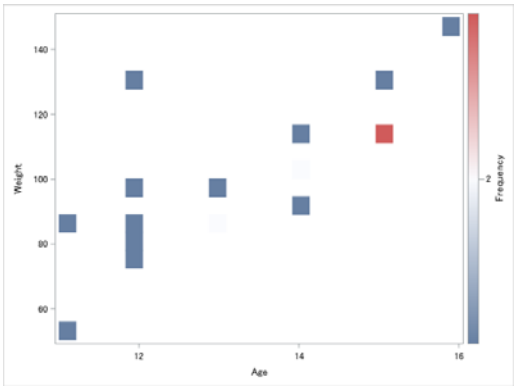
	
`# Scatter plot` `dt.scatter(x='age',y='weight')`	`# Heatmap` `dt.heatmap(x='age',y='weight')`
	

Figure 9. Graphics Methods of SASPy to Execute SAS Graphics Procedures

Output SAS Code with 'teach_me_SAS' Method

The 'teach_me_SAS' method outputs SAS codes executed in a SAS session, therefore this functionality is useful for Python users who are not familiar with the SAS coding rules. Before the execution of the methods of SASPy modules, "teach_me_SAS('True')" should be run (the default is set to 'False'). It needs to be noted that the programs are not actually executed during 'True' of this method. SAS codes generated by the teach_me_SAS method can be executed in SAS environment without any modifications.

In:	dt=sas.sasdata('class','sashelp')
	# Generate and Output SAS codes in a SAS session
	sas.teach_me_SAS('True')
	dt.head(3)
	dt.means()
	dt.hist(var='age',title='Histogram')
	dt.scatter(x='age',y='weight')
	dt.heatmap(x='age',y='weight')
	dt.bar(var='SEX')

Out:	proc print data=sashelp.class(obs=3);run;
	proc means data=sashelp.class stackodsoutput n nmiss median mean std min p25 p50 p75 max;run;
	proc sgplot data=sashelp.class;
	histogram age / scale=count;
	title "Histogram";
	density age;
	run;
	title;
	proc sgplot data=sashelp.class;
	scatter x=age y=weight;
	run;
	title;
	proc sgplot data=sashelp.class ;
	heatmap x=age y=weight;;
	run;
	title;
	proc sgplot data=sashelp.class;
	vbar SEX;
	run;
	title;

Table 19. Output SAS Codes with 'teach_me_SAS' Method

Submit Actual SAS Codes to SAS Session

The '%%SAS', so-called 'magic', enables users to submit actual SAS codes to a SAS session in SASPy. For example, users can execute the SAS code that is generated by the teach_me_SAS method to check if the same results can be obtained compared with the methods of SASPy modules. Figure 10 shows the execution of SAS code of the PRINT and the SGPLOT procedures that were obtained by teach_me_SAS method in Table 19.

```
%%SAS
*** %%SAS magic to submit SAS code to
SAS session ;
proc print data=sashelp.class(obs=3 );
run;
proc sgplot data=sashelp.class;
   histogram age / scale=count;
   title "Histogram";
   density age;
run;
title;
```

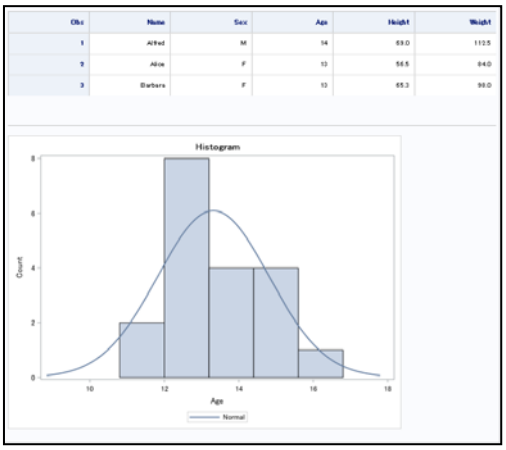


Figure 10. The %%SAS Magic for Execution of SAS Code in Python

CONCLUSION

SAS is still the most powerful software to analyze the data in clinical trials, but the interaction with other programming tools and the utilization of useful functionalities of them enable users to gain more efficiency and streamline the analysis process. Especially the Python is very useful for SAS users because there are tremendous modules in Python to easily exchange and utilize the SAS datasets and other formats for data handling and visualization. On the other hand, for Python users who are not familiar with native SAS code, the SASPy modules are very helpful to learn the basic SAS coding rules and easily execute the SAS modules with simple Python codes. In addition, it can be expected that more useful modules and methods will be enhanced and implemented in both SAS and Python in terms of the interaction between them. Therefore, as seen in this paper, both SAS and Python users will be able to obtain more knowledge, expand their abilities and gain more efficiency in daily works while utilizing the rapidly evolving functionalities of SAS and Python such as SASPy.

REFERENCES

Yuichi Nakajima. 2018. “Utilization of Python in clinical study by SASPy.” *PhUSE EU connect 2018*.

RECOMMENDED READING

- *Pandas (https://pandas.pydata.org/pandas-docs/stable/index.html)*
- *Matplotlib (https://matplotlib.org/)*
- *SASPy (https://sassoftware.github.io/saspy/)*
- *SAS Documentation (https://support.sas.com/en/documentation/all-products-documentation.html)*
- *CDISC ADaM Standard (https://www.cdisc.org/standards/foundational/adam)*

CONTACT INFORMATION

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

Yohei Takanami
youhei.takanami@takeda.com