Paper 3191-2019

Utiization of Python in clinical study by SASPy

Yuichi Nakajima, Novartis Pharma K.K

ABSTRACT

Python is the one of the most popular programming languages in recent years. It is now getting used in machine learning and AI. Big advantage of Python is that plenty of Python libraries to implement various analysis and it can be a "one stop shop" for programming. Although SAS® is and will be the most powerful analytical tool in clinical study, Python will expand reporting activity such as data aggregation and visualization in addition to SAS, and also it can be potential advancement of SAS programmer's career.

SASPy is the new ticket to Python for SAS programmers. It is known as Python Package library to the SAS system and enables to start SAS session in Python. This means Python can support the activities from SAS data handling to reporting.

This paper describes basic usage of SASPy and introduction of tips for handling SAS dataset. Then, discuss possible reporting activities using Python in clinical study.

INTRODUCTION

When SAS programmers try to use Python in your daily work, the first thing that comes to mind is SAS Viya™. SAS Viya is known as cloud analytic platform to solve various business needs and it is the one of the best analytic environment to use Python for SAS programmer. However unless your company decides to implement SAS Viya with spending a large amount of costs, it may not be a feasible option for the people who starts Python programming from scratch.

SASPy is the module, which provides Python Application Programming Interfaces (APIs) to the SAS system. By using SASPy, Python can establish SAS session and run analytics from Python.

PRE-REQUIREMENTS

If you have already installed SAS to your laptop, now you are ready to use SASPy. Additional a few steps will enable you to handle SAS datasets by Python. Here's a list for pre-requirements.

1. SAS 9.4 or higher
2. Anaconda distribution (Jupyter notebook, Python 3.X or higher, etc)
3. SASPy-2.4.3 (As of March 2019, v2.4.3 is the latest version)

Anaconda is the distribution package developed for implementation Python analytic library such as NumPy, Matplolib, Pandas and so on. The advantage of using Anaconda is that plenty of analytic libraries. If you start with single Python program, every time you need to install every libraries that you need. In other words, anaconda will provide a generic analytic environment instead of those a laborious process.

Jupyter notebook is the editor application whose kernel is the IPython. IPython kernel becomes a bridge between Jupyter notebook and Python so that Python can be effective in Jupyter notebook as a programming language. This IPython provides Magic Commands and this will be the best combination between Python and SAS. This will be explained in later section.

SASPY INSTALLATION PROCESS

SASPy installation will be the most complicated steps of utilization SASPy. Here is the example to install SASPy with local Windows PC SAS. All steps other than 3 can be completed in Anaconda Prompt only. Anaconda Prompt will be available on your PC after anaconda installation.

1. Make sure to ready above pre requirements respectively.
2. After download SASPy into your laptop, open Anaconda Prompt and enter `pip install SASPy` for install SASPy.
3. SASPy-2.4.3 (As of March 2019, v2.4.3 is the latest version) Now you need to update SAS configuration files (sascfg.py) to establish corresponding SAS session. If you can't find location of file, type `saspy.SAScfg` after importing SASPy in step 4.

- Update sascfg.py

  Location can be found by `saspy.SAScfg` in Python (Type `python` in Anaconda prompt then python is started.) When you update sascfg.py, first you can copy file and rename it as sascfg_personal.py. Then you can update below three points.

  - SAS_config_names to 'winlocal', e.g.

    SAS_config_names=['winlocal']

  - SAS session to specify java.exe file in sascfg_personal.py, e.g

    winlocal = {'java':'C:\ProgramData\Oracle\Java\javapath\java.exe',

    'encoding' : 'windows-1252',

    'classpath' : cpW}

  - Windows client class path. Make sure below links are corresponded to your own links.

    cpW = "C:\\Program Files\\SASHome\\SASDeploymentManager\\9.4\\products\\deploywiz__94472__prt__xx__sp0__1\\deploywiz\\sas.svc.connection.jar"

    cpW += ";C:\\Program Files\\SASHome\\SASDeploymentManager\\9.4\\products\\deploywiz__94472__prt__xx__sp0__1\\deploywiz\\log4j.jar"

    cpW += ";C:\\Program Files\\SASHome\\SASDeploymentManager\\9.4\\products\\deploywiz__94472__prt__xx__sp0__1\\deploywiz\\sas.security.sspi.jar"

    cpW += ";C:\\Program Files\\SASHome\\SASDeploymentManager\\9.4\\products\\deploywiz__94472__prt__xx__sp0__1\\deploywiz\\sas.core.jar"

    cpW += ";C:\\ProgramData\\Anaconda3\\Lib\\site-packages\\saspy\\java\\saspyiom.jar"

- Add system PATH environment variable for "C:\Program Files\SASHome\SASFoundation\9.4\core\sasext". (In fact it is a location of sspiauth.dll and it depends on your PC environment.)

4. Then import SASPy into your Python session by `import saspy` in python, check SAS connection is correctly established by `sas=saspy.SASsession(cfgname='winlocal')` with no error. If subprocess is displayed, SASPy installation is successfully completed.

Note that this example is only focusing on SASPy installation to PC SAS. Configuration update process depends on OS (Windows or Unix), and what to connect either local or server SAS.

Recently SAS announced SAS University Edition has implemented a functionality of Python by using SASPy. Only for learning purpose, you can use SAS University Edition and it will not request any additional steps to start.

DATA HANDLING CHOICES IN PYTHON

When SAS programmer thinks about data handling using Python, Pandas which is Python Package providing efficient data handling process would be one of possible option. Pandas data structures are called “Series” for single dimension like vector and “DataFrame” for two dimensions data like matrix.

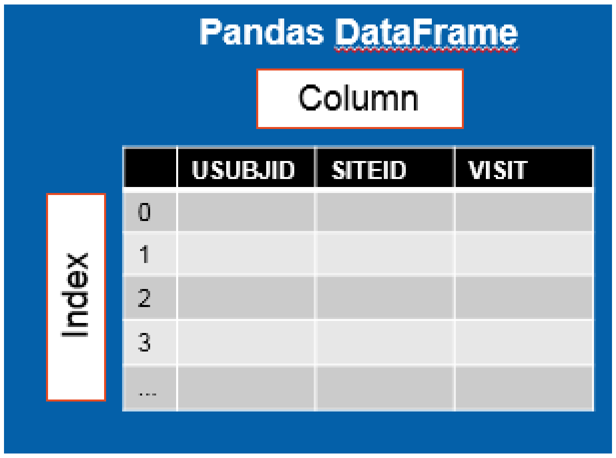


Figure 1. Image of Pandas DataFrame

Pandas can read directly both sas7bdat and xpt format and convert to Pandas DataFrame. This is the simplest way to handle SAS data in Python. On the other hands, SASPy is capable to handle SAS datasets without conversion to DataFrame. This means there are several ways to process SAS dataset in python. In fact, at least three types of process, Jupyter magic, SASPy API and Pandas DataFrame, can be choices to get the same result in Python although data format is different. Thus, depending on your purpose, you can choose the best way among them. Here is a brief comparison of those selections. Actual example will be shown in later section.

Choices	**Description**	**Example code of data sorting in Jupyter notebook cell** **<>: SAS dataset, []: Pandas DataFrame**

Jupyter magic	Magic commands is utility command such provided by IPython. Set “%%SAS” on the top of cell, then SAS code can be effective within that python cell.	`%%SAS`
		
		`libname temp ’xxxxxx’;`
		
		`proc sort data = temp.<AAAA> out = work.<BBBB>;`
		
		`  by USUBJID descending AESEV;`
		
		`run;`

SASPy API	SASPy can setup a SAS session and run analytics from Python.	`sas.saslib(’temp’, path=’xxxxxx’)`
		
		`<AAAA>=sas.sasdata(’<AAAA>’, libref=’temp’)`
		
		`<BBBB>=<AAAA>.sort(by=’USUBJID DESCENDING AESEV’, out=’WORK’)`

Choices	Description	Example code of data sorting in Jupyter notebook cell <>: SAS dataset, []: Pandas DataFrame
Pandas DataFrame	Pandas is a third-party package to handle one dimension data (Vector: Series) and 2 dimension data (Matrix: DataFrame) with Pandas analytic functions. Set “import Pandas” to use Pandas first.	`[BBBB]=<AAAA>.sort_values(by=['USUBJID', 'AESEV'], ascending=[True, False])`

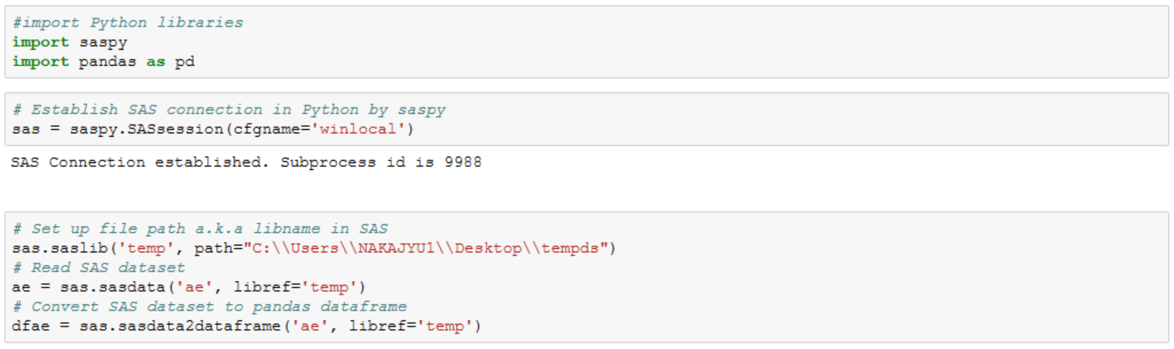
Table 1. Choices of data sorting step in Python

This paper mainly focus on using Pandas DataFrame because Pandas is very basic and popular Python library to process input data regardless its data formats. For the comparison to SAS programming, the summary of differences between Python and SAS in basic data process technics can be found in the backup section.

DATA HANDLING AND REPOTING

Now let us get started with data reporting in Python using SAS dataset. The goal of this section is to understand how to start SAS session and to create a basic summary table with CDISC standardized dataset.

Here is the overview process before data reporting. As the first step, import 2 python libraries, SASPy and Pandas. Secondary, establish SAS session to read SAS data with SASPy API, SASsession(). Note that SAS dataset can be directly obtained in Python as a SAS dataset by SASPy API, sasdata(). Then, convert SAS dataset to Pandas DataFrame by SASPy API, sasdata2dataframe().



```
#import Python libraries
import saspy
import pandas as pd
```

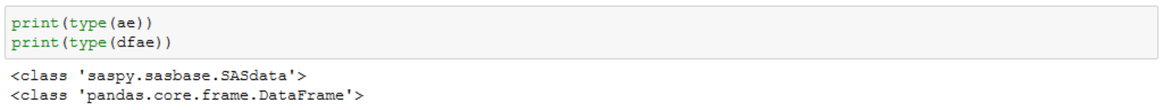
```
# Establish SAS connection in Python by saspy
sas = saspy.SASsession(cfgname='winlocal')
```

```
SAS Connection established. Subprocess id is 9988
```

```
# Set up file path a.k.a libname in SAS
sas.saslib('temp', path="C:\\Users\\NAKAJYU1\\Desktop\\tempds")
# Read SAS dataset
ae = sas.sasdata('ae', libref='temp')
# Convert SAS dataset to pandas dataframe
dfae = sas.sasdata2dataframe('ae', libref='temp')
```

Display 1. General step to create SAS session in Jupyter notebook

Now you will see the data type of each element, “ae” is SAS data and “dfae” is DataFrame. Those data type can be obtained by type() function.



```
print(type(ae))
print(type(dfae))
```

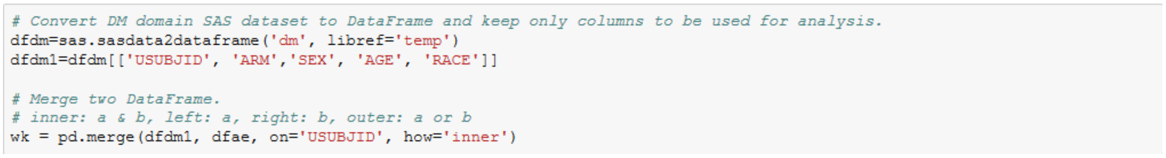
```
<class 'saspy.sasbase.SASdata'>
<class 'pandas.core.frame.DataFrame'>
```

Display 2. Result of type function

COMMON TABLES IN CLINICAL STUDY

If you are success to establish SAS session, now start to create DataFrame to be used for reporting. This section will show a few summary tables which are commonly used in clinical study with simple Python code.

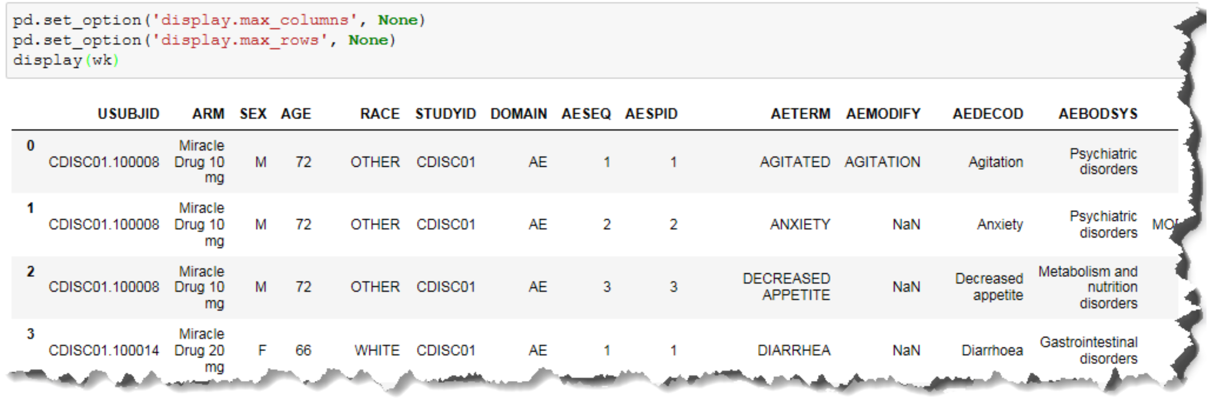
Firstly, DM and AE domain are merged to create AE summary table by treatment arm (ARM). The first DM is converted to Pandas DataFrame and keep only columns to be used. Then, create merged DataFrame wk from two DataFrame dfdm1 and dfae with merge() function e.g. `wk = pd.merge(dfdm1, dfae, on='USUBJID', how='inner')`. Merge function has several options how to merge, such as inner, right, left and outer join.



```
# Convert DM domain SAS dataset to DataFrame and keep only columns to be used for analysis.
dfdm=sas.sasdata2dataframe('dm', libref='temp')
dfdm1=dfdm[['USUBJID', 'ARM','SEX', 'AGE', 'RACE']]

# Merge two DataFrame.
# inner: a & b, left: a, right: b, outer: a or b
wk = pd.merge(dfdm1, dfae, on='USUBJID', how='inner')
```

Display 3. Convert SAS dataset to DataFrame and merge example

Here is a merged DataFrame. Default setting won't show every column so that below two set_option() functions are recommended to use. Otherwise some columns will be shown as "...". To display contents of DataFrame display() function is one of option.

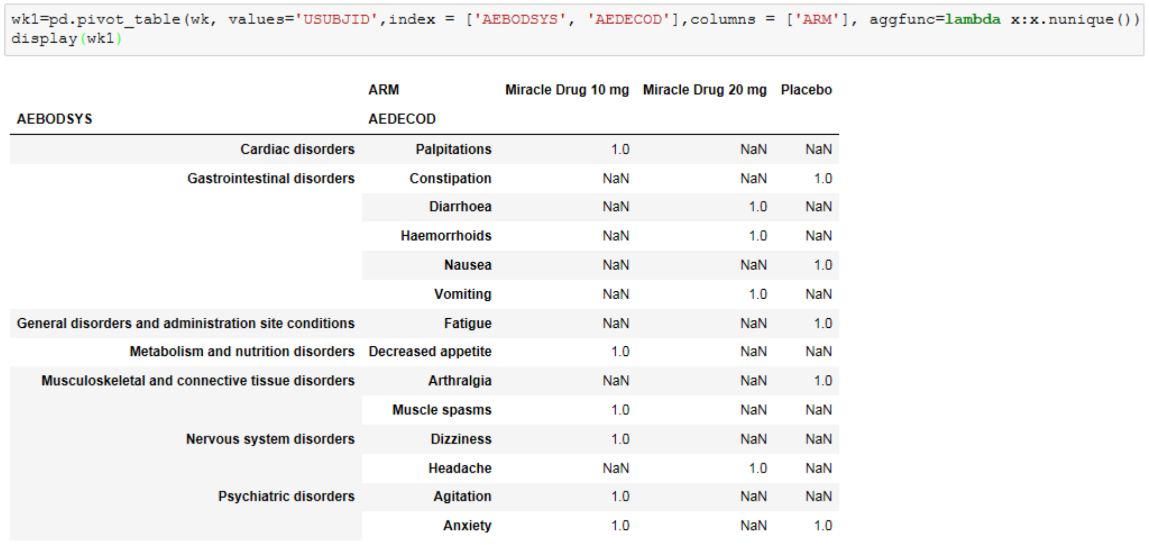


```
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
display(wk)
```

	USUBJID	ARM	SEX	AGE	RACE	STUDYID	DOMAIN	AESEQ	AESPID	AETERM	AEMODIFY	AEDECOD	AEBODSYS	
0	CDISC01.100008	Miracle Drug 10 mg	M	72	OTHER	CDISC01	AE	1	1	AGITATED	AGITATION	Agitation	Psychiatric disorders	
1	CDISC01.100008	Miracle Drug 10 mg	M	72	OTHER	CDISC01	AE	2	2	ANXIETY	NaN	Anxiety	Psychiatric disorders	MO
2	CDISC01.100008	Miracle Drug 10 mg	M	72	OTHER	CDISC01	AE	3	3	DECREASED APPETITE	NaN	Decreased appetite	Metabolism and nutrition disorders	
3	CDISC01.100014	Miracle Drug 20 mg	F	66	WHITE	CDISC01	AE	1	1	DIARRHEA	NaN	Diarrhoea	Gastrointestinal disorders	

Display 4. Example of display DataFrame in Jupyter notebook

With simple code of Pandas pivot_table() function will show Adverse Event (AE) summary table by treatment arm can be displayed. This pivot_table() has several functionalities for summary table by setting aggfunc option. This example is counting records in values (e.g. USUBJID) without duplicated records. If data has no duplicated record, simply set aggfunc = count. In addition to python function like count function, actually argument for aggfunc can be selected from numpy function such as np.mean(), np.sum(). Return value of pivot_table() is Pandas DataFrame, that means index are AEBODSYS and AEDECOD, columns are ARM corresponding values, Miracle Drug 10 mg, Miracle Drug 20 mg and Placebo.

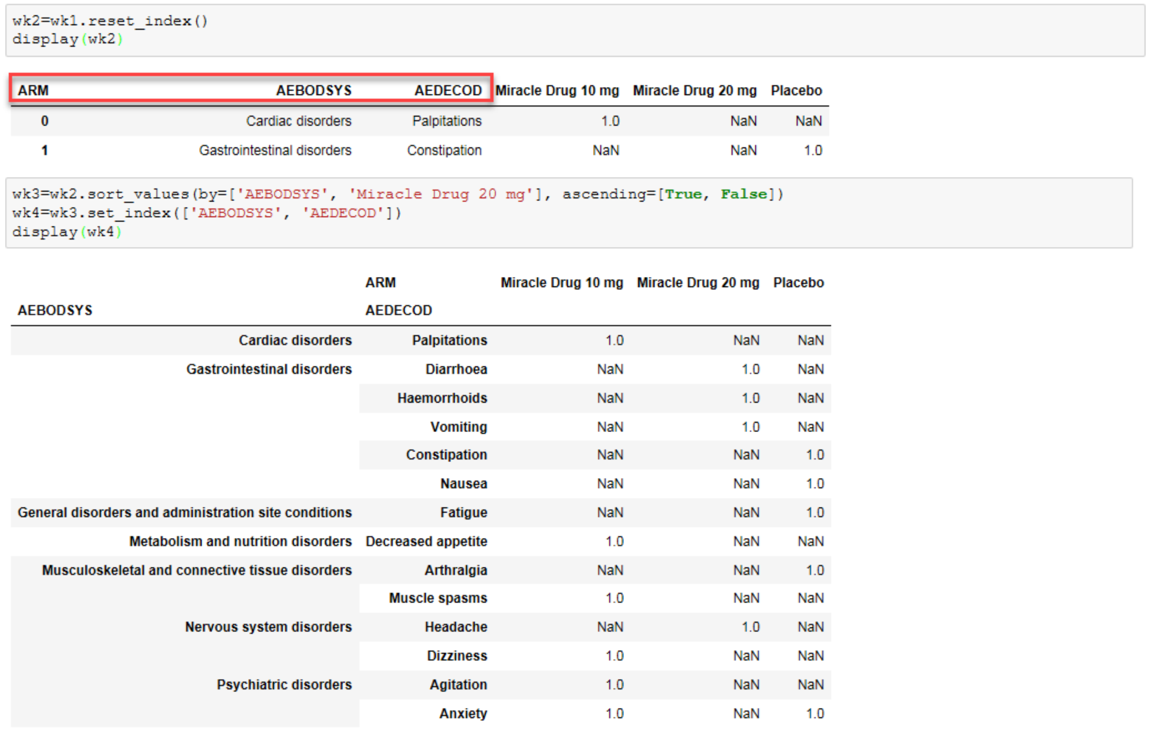


```
wk1=pd.pivot_table(wk, values='USUBJID',index = ['AEBODSYS', 'AEDECOD'],columns = ['ARM'], aggfunc=lambda x:x.nunique())
display(wk1)
```

	ARM	Miracle Drug 10 mg	Miracle Drug 20 mg	Placebo
AEBODSYS	AEDECOD			
Cardiac disorders	Palpitations	1.0	NaN	NaN
Gastrointestinal disorders	Constipation	NaN	NaN	1.0
	Diarrhoea	NaN	1.0	NaN
	Haemorrhoids	NaN	1.0	NaN
	Nausea	NaN	NaN	1.0
	Vomiting	NaN	1.0	NaN
General disorders and administration site conditions	Fatigue	NaN	NaN	1.0
Metabolism and nutrition disorders	Decreased appetite	1.0	NaN	NaN
Musculoskeletal and connective tissue disorders	Arthralgia	NaN	NaN	1.0
	Muscle spasms	1.0	NaN	NaN
Nervous system disorders	Dizziness	1.0	NaN	NaN
	Headache	NaN	1.0	NaN
Psychiatric disorders	Agitation	1.0	NaN	NaN
	Anxiety	1.0	NaN	1.0

Display 5. Example of AE summary table by SOC and PT

If you would like to sort in alphabetically in AEBODSYS and frequency in high dose, you need to remove index first with Pandas reset_index() function. Now You see ARM is changed from index to column. Then sort column with Pandas sort_values() function and back to index by set_index() function.



```
wk2=wk1.reset_index()
display(wk2)
```

ARM	AEBODSYS	AEDECOD	Miracle Drug 10 mg	Miracle Drug 20 mg	Placebo
0	Cardiac disorders	Palpitations	1.0	NaN	NaN
1	Gastrointestinal disorders	Constipation	NaN	NaN	1.0

```
wk3=wk2.sort_values(by=['AEBODSYS', 'Miracle Drug 20 mg'], ascending=[True, False])
wk4=wk3.set_index(['AEBODSYS', 'AEDECOD'])
display(wk4)
```

	ARM	Miracle Drug 10 mg	Miracle Drug 20 mg	Placebo
AEBODSYS	AEDECOD			
Cardiac disorders	Palpitations	1.0	NaN	NaN
Gastrointestinal disorders	Diarrhoea	NaN	1.0	NaN
	Haemorrhoids	NaN	1.0	NaN
	Vomiting	NaN	1.0	NaN
	Constipation	NaN	NaN	1.0
	Nausea	NaN	NaN	1.0
General disorders and administration site conditions	Fatigue	NaN	NaN	1.0
Metabolism and nutrition disorders	Decreased appetite	1.0	NaN	NaN
Musculoskeletal and connective tissue disorders	Arthralgia	NaN	NaN	1.0
	Muscle spasms	1.0	NaN	NaN
Nervous system disorders	Headache	NaN	1.0	NaN
	Dizziness	1.0	NaN	NaN
Psychiatric disorders	Agitation	1.0	NaN	NaN
	Anxiety	1.0	NaN	1.0

Display 6. Change sorting order in column

Note that "NaN" in results can be replaced by zero if fill_value=0 option is applied in pivot_table function. Adding percentage is also important for this type of output. As there is no one step calculation of percentage in Pandas library, to get percentage of each ARM, total number of subject should be counted first. Then function that can calculate percentage is defined and apply to each column in DF.

ARM	Miracle Drug 10 mg	Miracle Drug 20 mg	Placebo
USUBJID	2	1	1

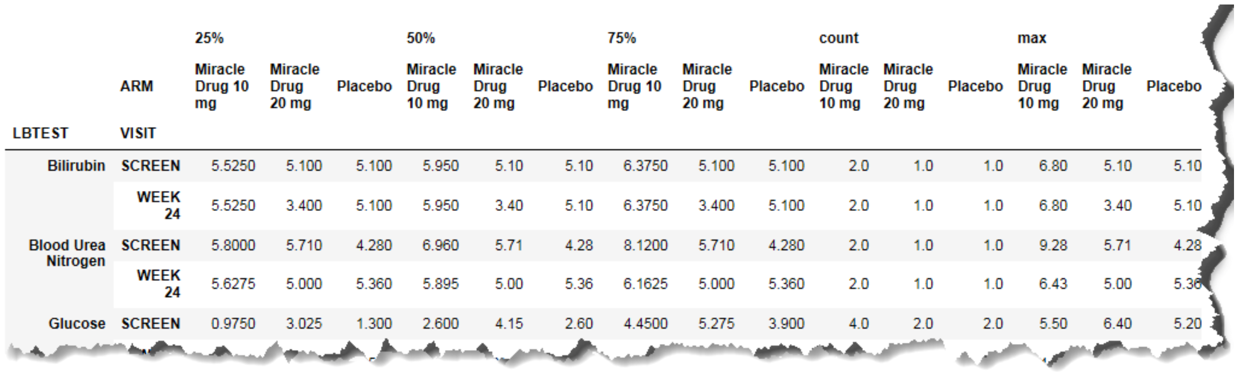
```
def divf(a, b):
    return str(a) + ' (' + str(a * 100 / b) + ')'
```

```
wk1['Miracle Drug 10 mg']=wk1['Miracle Drug 10 mg'].apply(divf, b=div.loc['USUBJID', 'Miracle Drug 10 mg'])
wk1['Miracle Drug 20 mg']=wk1['Miracle Drug 20 mg'].apply(divf, b=div.loc['USUBJID', 'Miracle Drug 20 mg'])
wk1['Placebo']=wk1['Placebo'].apply(divf, b=div.loc['USUBJID', 'Placebo'])
display(wk1)
```

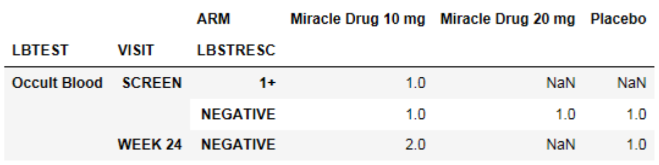
	ARM	Miracle Drug 10 mg	Miracle Drug 20 mg	Placebo
AEBODSYS	AEDECOD			
Cardiac disorders	Palpitations	1 (50.0)	0 (0.0)	0 (0.0)
Gastrointestinal disorders	Constipation	0 (0.0)	0 (0.0)	1 (100.0)
	Diarrhoea	0 (0.0)	1 (100.0)	0 (0.0)
	Haemorrhoids	0 (0.0)	1 (100.0)	0 (0.0)
	Nausea	0 (0.0)	0 (0.0)	1 (100.0)
	Vomiting	0 (0.0)	1 (100.0)	0 (0.0)
General disorders and administration site conditions	Fatigue	0 (0.0)	0 (0.0)	1 (100.0)
Metabolism and nutrition disorders	Decreased appetite	1 (50.0)	0 (0.0)	0 (0.0)
Musculoskeletal and connective tissue disorders	Arthralgia	0 (0.0)	0 (0.0)	1 (100.0)
	Muscle spasms	1 (50.0)	0 (0.0)	0 (0.0)

Display 7. Example to add percentage of each ARM

Secondary, if you would like to check summary statistics from LB domain, Pandas pivot_table() function will also provide you a result for both continuous and categorical values. "describe" and "count" in aggfunc option will give descriptive statistics and simple count table respectively as below. For example, descriptive statistics is obtained by `rslt1 = pd.pivot_table(wk, values='LBSTRESN'), index=['LBTEST', 'VISIT'], columns = ['ARM'], aggfunc='describe').`



		25%			50%			75%			count			max		
	ARM	Miracle Drug 10 mg	Miracle Drug 20 mg	Placebo	Miracle Drug 10 mg	Miracle Drug 20 mg	Placebo	Miracle Drug 10 mg	Miracle Drug 20 mg	Placebo	Miracle Drug 10 mg	Miracle Drug 20 mg	Placebo	Miracle Drug 10 mg	Miracle Drug 20 mg	Placebo
LBTEST	VISIT															
Bilirubin	SCREEN	5.5250	5.100	5.100	5.950	5.10	5.10	6.3750	5.100	5.100	2.0	1.0	1.0	6.80	5.10	5.10
	WEEK 24	5.5250	3.400	5.100	5.950	3.40	5.10	6.3750	3.400	5.100	2.0	1.0	1.0	6.80	3.40	5.10
Blood Urea Nitrogen	SCREEN	5.8000	5.710	4.280	6.960	5.71	4.28	8.1200	5.710	4.280	2.0	1.0	1.0	9.28	5.71	4.28
	WEEK 24	5.6275	5.000	5.360	5.895	5.00	5.36	6.1625	5.000	5.360	2.0	1.0	1.0	6.43	5.00	5.36
Glucose	SCREEN	0.9750	3.025	1.300	2.600	4.15	2.60	4.4500	5.275	3.900	4.0	2.0	2.0	5.50	6.40	5.20



		ARM	Miracle Drug 10 mg	Miracle Drug 20 mg	Placebo
LBTEST	VISIT	LBSTRESC			
Occult Blood	SCREEN	1+	1.0	NaN	NaN
		NEGATIVE	1.0	1.0	1.0
	WEEK 24	NEGATIVE	2.0	NaN	1.0

Display 8. Summary table of descriptive statistics and simple counting

Finally, when those results are available in Python, you can export them to html (Use DF.to_html('result01.html')).

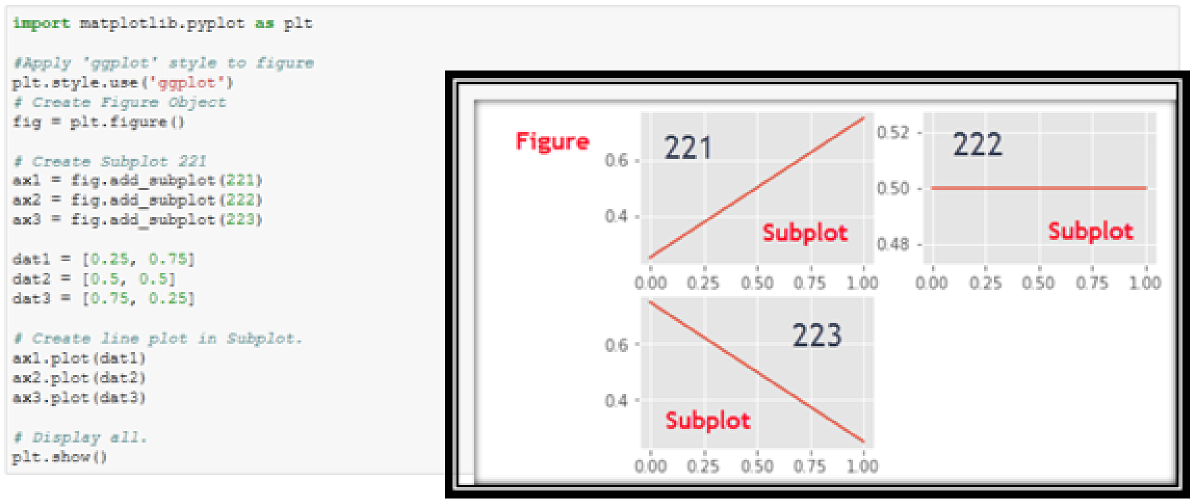
Thus, Python can produce simple summary result easily with simple code. Of course Python is possible to create a TFL shells for clinical study report but as you can easily imagine it will be difficult in terms of computer system validation. Oh the other hand, Python can be used for quick data review for data manager and also be considered as a tool for acceptance checks for outsourced deliverables. In addition, Python programming can be started with no additional system cost.

DATA VISUALIZATION IN PYTHON

This section will explain the data visualization in python and focus on using Matplotlib.pyplot, which is a Python plotting library. Actually Pandas has implemented a plot method called Pandas Plot as known as a simple wrapper around Matplotlib. The reason to use Matplotlib is because Matplotlib is able to export an output in several formats (png, gif, pdf, mp4, ...), to set the detail figure setting like axis, title, legend, and to find an information easily from several website thanks to huge number of Matplotlib users.

GETTING STARTED WITH MATPLOTLIB

After importing Matplotlib, generate 1) Figure and 2) Subplot object first. Figure is the plotting area to locate Subplot and Subplot is the plotting area to display plot. At least, one Subplot must be included in Figure.



```
import matplotlib.pyplot as plt

#Apply 'ggplot' style to figure
plt.style.use('ggplot')
# Create Figure Object
fig = plt.figure()

# Create Subplot 221
ax1 = fig.add_subplot(221)
ax2 = fig.add_subplot(222)
ax3 = fig.add_subplot(223)

dat1 = [0.25, 0.75]
dat2 = [0.5, 0.5]
dat3 = [0.75, 0.25]

# Create line plot in Subplot.
ax1.plot(dat1)
ax2.plot(dat2)
ax3.plot(dat3)

# Display all.
plt.show()
```

Figure 2. Concept of Figure and Subplot

PLOT EXAMPLE FOR CLINICAL STUDY

Now, we will more focus on the plot to be referred in clinical study reporting. The first example is a mean plot with SD. To get summary statistics, save mean and SD as Pandas

DataFrame by Pandas describe() function. Then, prepare Pandas Series which contains each mean and SD by Pandas loc() function.

```
#Calcurate summary statistic per ARM, AVISITN
sum=wk1.groupby(['TRT01P_a', 'AVISITN'])['AVAL'].describe()

#Get mean and std into pandas Series.
mean1=sum.loc['DRUG X', 'mean']
mean2=sum.loc['DRUG Y', 'mean']
mean3=sum.loc['Placebo', 'mean']
std1=sum.loc['DRUG X', 'std']
std2=sum.loc['DRUG Y', 'std']
std3=sum.loc['Placebo', 'std']
```

Display 9. Extract Mean and SD columns from DataFrame

Note that wk1 is the ADaM BDS structured dummy data in Pandas DF and “sum” is a Pandas DF structure whose index are [TRT01_P, AVISITN] and columns are [count, mean, std]. mean1, 2, 3 and std1, 2, 3 are Pandas Series whose index is AVISITN and column is each statistic (mean or SD).

```
plt.style.use('ggplot')

# Create figure object and subplot
fig=plt.figure(figsize=(20,10))
ax = fig.add_subplot(111)

# Define subplot setting
ax.plot(mean1.index-0.5, mean1, color='r', label='DRUG X')
ax.plot(mean2.index, mean2, color='g', label='DRUG Y')
ax.plot(mean3.index+0.5, mean3, color='b', label='Placebo')
ax.legend(loc="upper left")

# Define array for x-axis label setting
vis_num   = np.array([0, 1, 2, 7, 14, 28, 56])
vis_order = np.array(["Baseline", "Day 1", "Day 2", "Week 1", "Week 2", "Week 4", "Week 8"])
# x-axis ticks
ax.set_xticks(vis_num)
# x-axis labels
ax.set_xticklabels(vis_order, rotation=90)

# Define error bar
ax.errorbar(mean1.index-0.5, mean1, yerr=std1, fmt='ro', ecolor='r', capsize=4)
ax.errorbar(mean1.index, mean2, yerr=std2, fmt='ro',ecolor='g', capsize=4)
ax.errorbar(mean1.index+0.5, mean3, yerr=std3, fmt='ro',ecolor='b', capsize=4)

#Figure setting
plt.title('SBP (mmHg), Mean with SD')
plt.xlabel('Analysis Visit')
plt.ylabel('SBP (mmHg)')

plt.show()
```

Display 10. Example of Mean with SD plot by Matplotlib

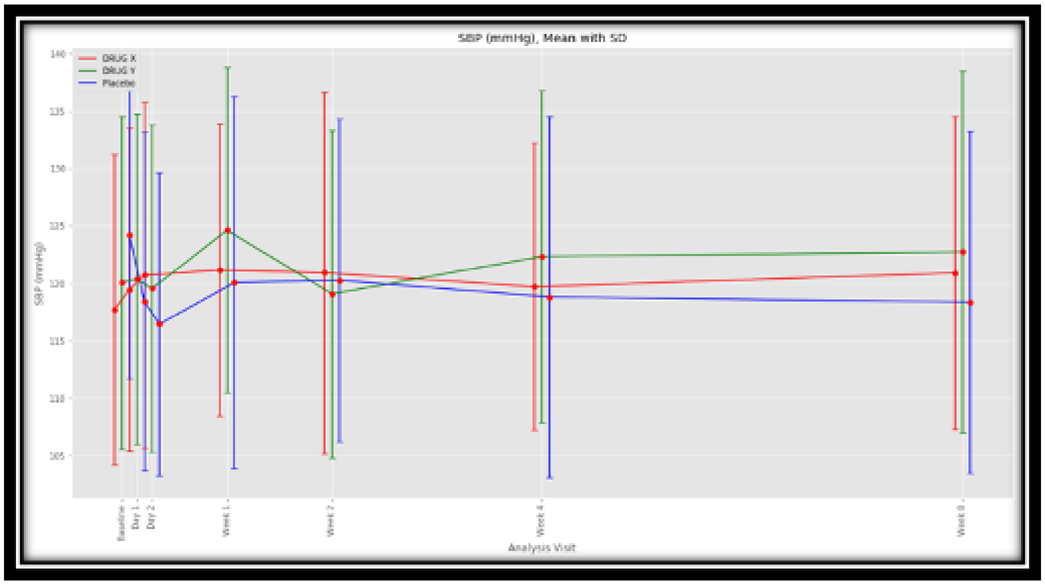
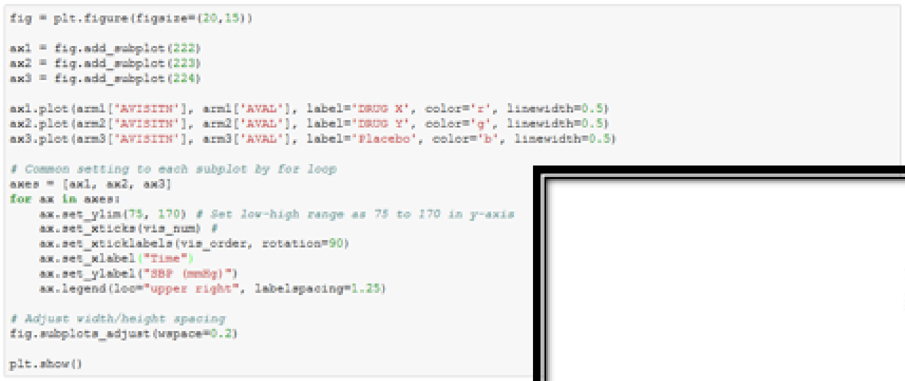


Figure 3. Mean with SD plot

The point here is x-axis labels. The set_xticklabels() is the function to display x-axis labels. If we use AVISIT in set_xticklabels(), labels will be displayed alphabetically so that it is not a sequential order. To aboid this problem, you need to prepare the label list to be displaied which is corresponding to actual AVISIT values.

Another example is a patient level plot as known as "spaghetti plot". As there will be 3 treatment groups in dummy data, 3 subplot will be required. Note that most of subplot setting are equal, it can be processed by for loop with subset list `axes = [ax1, ax2, ax3]`. For loop, for ax in axes: , will do each process per subplot read by the list [ax1, ax2, ax3].



```
fig = plt.figure(figsize=(20,15))

ax1 = fig.add_subplot(222)
ax2 = fig.add_subplot(223)
ax3 = fig.add_subplot(224)

ax1.plot(arm1['AVISITN'], arm1['AVAL'], label='DRUG X', color='r', linewidth=0.5)
ax2.plot(arm2['AVISITN'], arm2['AVAL'], label='DRUG Y', color='g', linewidth=0.5)
ax3.plot(arm3['AVISITN'], arm3['AVAL'], label='Placebo', color='b', linewidth=0.5)

# Common setting to each subplot by for loop
axes = [ax1, ax2, ax3]
for ax in axes:
    ax.set_ylim(75, 170) # Set low-high range as 75 to 170 in y-axis
    ax.set_xticks(vis_num) #
    ax.set_xticklabels(vis_order, rotation=90)
    ax.set_xlabel("Time")
    ax.set_ylabel("SBP (mmHg)")
    ax.legend(loc="upper right", labelspacing=1.25)

# Adjust width/height spacing
fig.subplots_adjust(wspace=0.2)

plt.show()
```

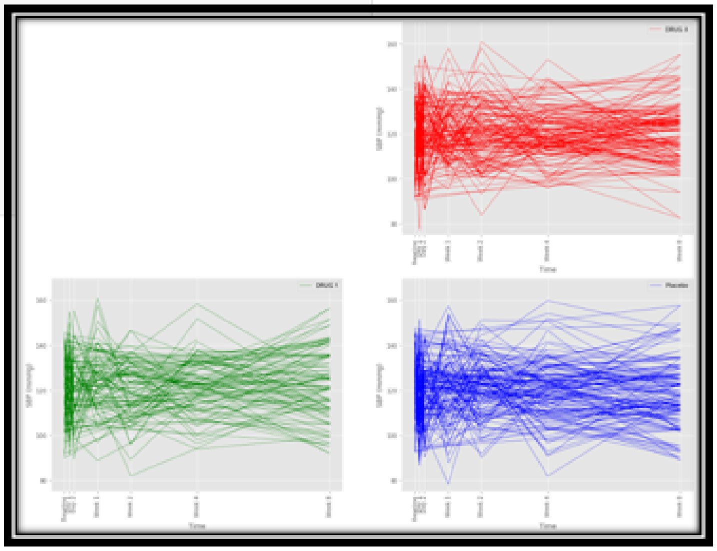


Figure 4. Example code and figures of Spaghetti plot

Actually above those examples might not be a simple code as it was tried to approach the general plots in clinical study. Here are some quick examples for other type of plots with simple code. It would be sufficient if you need only quick data check and review.

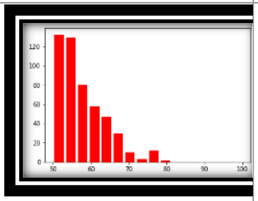
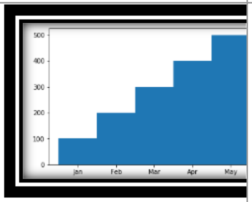
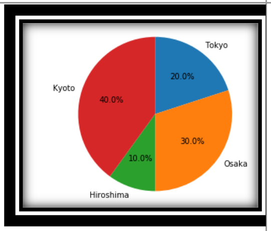
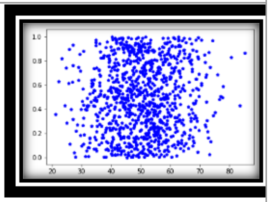
Type of plot	**Histgrams**	**Bar charts**	**Pie charts**	**Scatter plot**
Matplotlib function	hist()	bar()	pie()	scatter()
Description	Compute and draw the histogram of x.	The bars are positioned at x with the given alignment. Their dimensions are given by width and height.	Make a pie chart of array x. The fractional area of each wedge is given by x/sum(x).	A scatter plot of y vs x with varying marker size and/or color.
Python code	>>> plt.hist(x, bins=16, range=(50, 100), rwidth=0.8, color='red')	>>> plt.bar(x1, x2, width=1.0, linewidth=3, align='center', tick_label=['Jan', 'Feb', 'Mar', 'Apr', 'May'])	>>> plt.pie(x3, labels=['Tokyo', 'Osaka', 'Hiroshima', 'Kyoto'], counterclock=False, startangle=90, autopct="%1.1f%%") >>> plt.axis('equal')	>>> plt.scatter(x, y, s=15, c='blue', marker='*', linewidth='2')
Example				

Table 2. Python plot gallery

UTLIZATION PYTHON IN CLINICAL STUDY

As described in previous sections, Python is possible to be used in clinical study reporting regarding data review, acceptance check and data visualization. With combination of SAS and Python in Jupyter notebook, more efficient program development will be possible. For example, when you create AE table for CSR in Jupyter notebook, you can use proc report by SAS code in Jupyter magic then the results can be checked by Pandas pivot_table() by yourself. In other words, when SAS session can be established in Jupyter notebook, it can be an enhanced editor which is capable to use both SAS and Python. Of course, Python can be used for validation purpose as far as main program will be created by SAS. If analysis datasets like ADaM is well prepared and created, validation code could be also simple. This might reduce total cost and time of validation programming because of multiple choices in validation such as programming languages, type of tables and skill level of programmers.

Obviously, those activities can be completed by SAS only. Now, let us consider about Python advantage programming at which SAS is not good.

In Novartis, annotated CRF (acrf.pdf) is generated by semi-automated process. Annotation database in both central and project level are well structured and controlled, and easy to carry out annotation to new studies. However, there is the only thing that programmer has to do manually. That is creation of the simple excel file which contains two columns, CRF ID and page number of where that CRF ID is located. In process, SAS program will read excel file and link annotation database and blank study CRF.

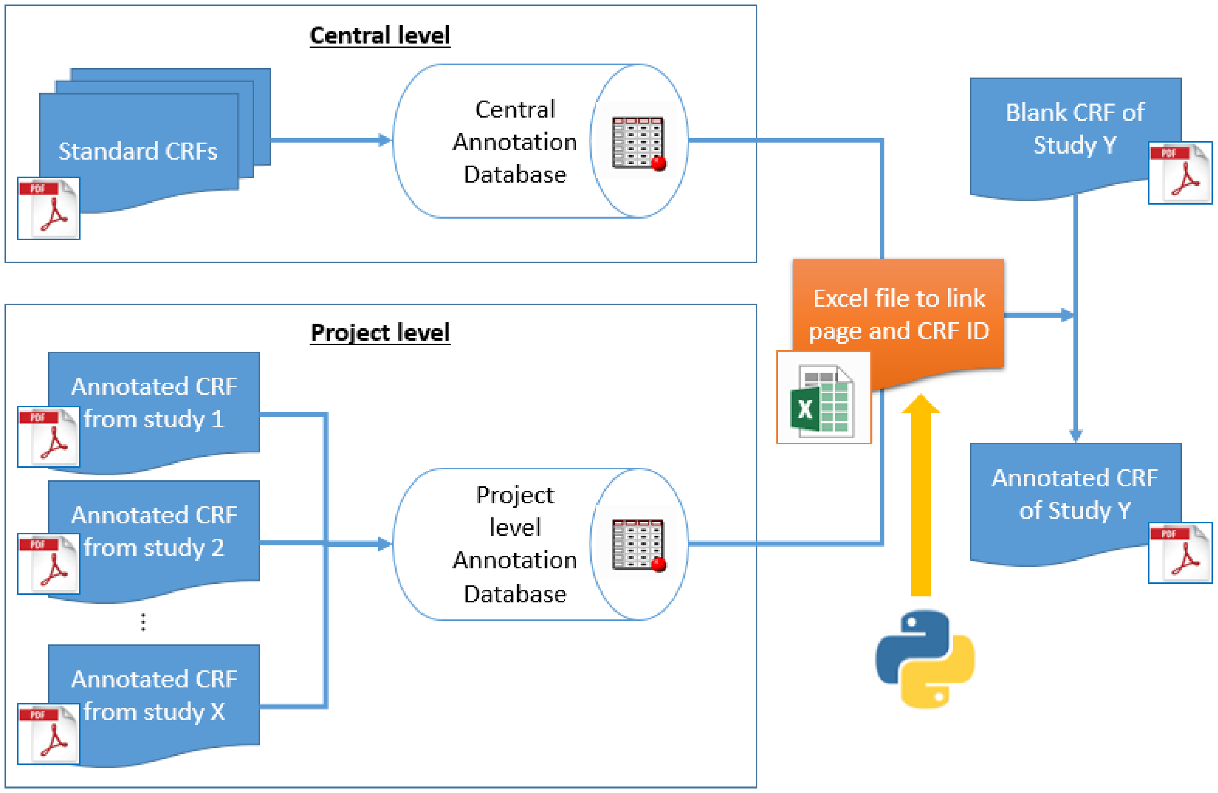
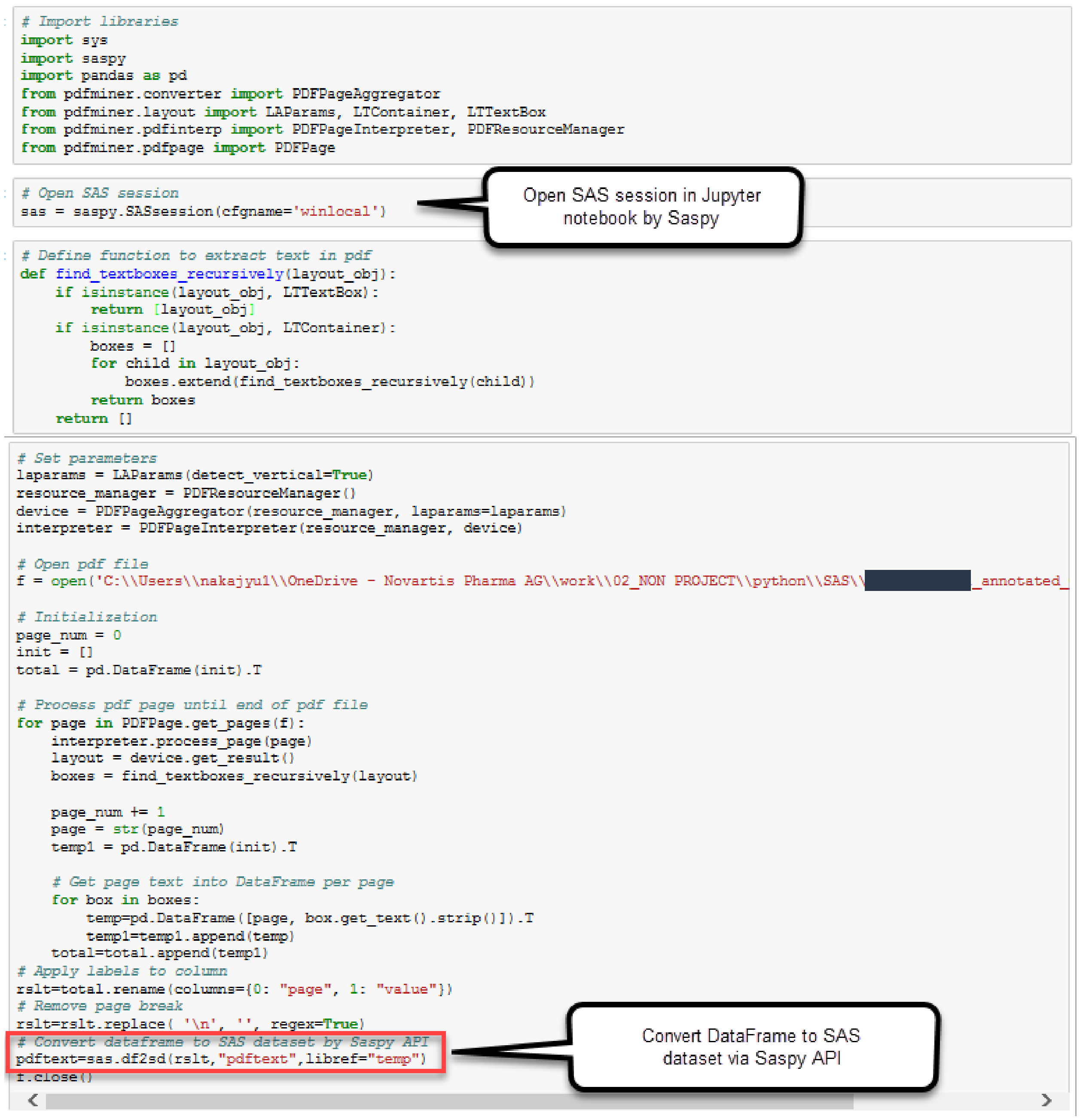

Central level
Standard CRFs
Central Annotation Database
Blank CRF of Study Y
Excel file to link page and CRF ID
Project level
Annotated CRF from study 1
Annotated CRF from study 2
Annotated CRF from study X
Project level Annotation Database
Annotated CRF of Study Y

Figure 5. Process overview of creation of aCRF in Novartis

What Python does in this section is to extract every text including CRF ID from each page and extract corresponded page number from PDF file, then, merge with annotation database and create the excel file to be required. This process is executed in only Jupyter notebook by combination of SASPy API and Jupyter magic command.

In order to achieve this idea, third vendor library pdfminer has to be installed. Please note that detail information about pdfminer is not explained in this paper because the key message is the process of data extraction from PDF to SAS dataset by using SASPy done in Jupyter notebook only. Here is an example code.


Open SAS session in Jupyter notebook by Saspy
Convert DataFrame to SAS dataset via Saspy API

```
# Import libraries
import sys
import saspy
import pandas as pd
from pdfminer.converter import PDFPageAggregator
from pdfminer.layout import LAParams, LTContainer, LTTextBox
from pdfminer.pdfinterp import PDFPageInterpreter, PDFResourceManager
from pdfminer.pdfpage import PDFPage
```

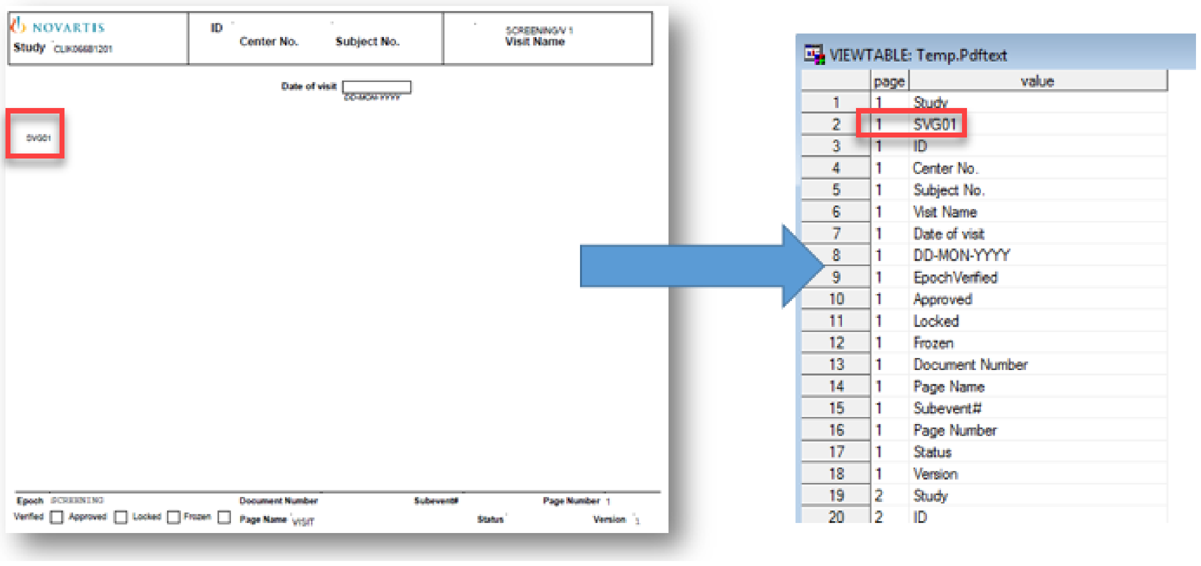
```
# Open SAS session
sas = saspy.SASsession(cfgname='winlocal')
```

```
# Define function to extract text in pdf
def find_textboxes_recursively(layout_obj):
    if isinstance(layout_obj, LTTextBox):
        return [layout_obj]
    if isinstance(layout_obj, LTContainer):
        boxes = []
        for child in layout_obj:
            boxes.extend(find_textboxes_recursively(child))
        return boxes
    return []
```

```
# Set parameters
laparams = LAParams(detect_vertical=True)
resource_manager = PDFResourceManager()
device = PDFPageAggregator(resource_manager, laparams=laparams)
interpreter = PDFPageInterpreter(resource_manager, device)

# Open pdf file
f = open('C:\\Users\\nakajyu1\\OneDrive - Novartis Pharma AG\\work\\02_NON PROJECT\\python\\SAS\\[redacted]_annotated_

# Initialization
page_num = 0
init = []
total = pd.DataFrame(init).T

# Process pdf page until end of pdf file
for page in PDFPage.get_pages(f):
    interpreter.process_page(page)
    layout = device.get_result()
    boxes = find_textboxes_recursively(layout)

    page_num += 1
    page = str(page_num)
    temp1 = pd.DataFrame(init).T

    # Get page text into DataFrame per page
    for box in boxes:
        temp=pd.DataFrame([page, box.get_text().strip()]).T
        temp1=temp1.append(temp)
    total=total.append(temp1)
# Apply labels to column
rslt=total.rename(columns={0: "page", 1: "value"})
# Remove page break
rslt=rslt.replace( '\n', '', regex=True)
# Convert dataframe to SAS dataset by Saspy API
pdftext=sas.df2sd(rslt,"pdftext",libref="temp")
f.close()
```

Display 11. Example code of extract PDF comment box to DataFrame

After running the above command in Jupyter, SAS dataset with all page numbers and values will be generated from blank crf. This format is exactly the same format which annotation tool requires in excel format.



	page	value
1	1	Study
2	1	SVG01
3	1	ID
4	1	Center No.
5	1	Subject No.
6	1	Visit Name
7	1	Date of visit
8	1	DD-MON-YYYY
9	1	EpochVerified
10	1	Approved
11	1	Locked
12	1	Frozen
13	1	Document Number
14	1	Page Name
15	1	Subevent#
16	1	Page Number
17	1	Status
18	1	Version
19	2	Study
20	2	ID

Figure 6. Text information in CRF are extracted to SAS dataset

As you can see in the below figure, when SAS session is established in Jupyter notebook, %%SAS will enable to run SAS code directly in Jupyter notebook cell. After the above SAS dataset is created, it will be merged to central annotation database and keep only effective crfid in database.

```
# Jupyter magic command, run SAS command in cell
%%SAS
libname temp "C:\Users\nakajyu1\OneDrive - Novartis Pharma AG\work\02_NON PROJECT\python\SAS";

proc sort data = temp.pdftext out = wk1 nodupkey;
  by value page;
run;
data wk2;
  set wk1;
  length crfid $30.;
  pagenum = input(page, best.);
  crfid = strip(value);
run;
proc sql noprint ;
  select max ( pagenum ) into :maxn from wk2 ;
quit ;
%let maxn = &maxn ;
proc sort data = temp.central_annotation_database out = wk3 nodupkey;
  by crfid;
run;

data wk4;
  merge wk2(in=a) wk3(in=b);
  by crfid;
  if a and b;
  keep pagenum crfid;
  proc sort;
    by pagenum crfid;
run;
data dummy;
  do pagenum = 1 to &maxn.;
  output;end;
run;
data temp.rslt;
  merge dummy wk4;
  by pagenum;
run;
```

Display 12. Example of Jupyter magic to run SAS code

Finally completed excel file (only if every crfid are available in annotation database) will be created from final dataset (temp.rslt) above.

CONCLUSION

As described, SASPy is able to establish SAS session in Jupyter notebook via SASPy, and Python can provide summary tables and figures for data review from SAS dataset in clinical study. This also means Python can be possibly used not only for acceptance check to vendor deliverables, but also for a part of output validation process. In addition, Python will provide several possibilities by plenty of useful libraries such as reading PDF file.

There is no doubt SAS is the most popular software in clinical study data analysis. However, acquiring Python programming skill additionally will enable SAS programmer to expand the capability and opportunity of daily work. As described in previous section, SAS is good at data handling and Python can read PDF file. Both programming languages have each advantage and disadvantage respectively. With the combination of SAS and Python via SASPy, their disadvantage can be covered each other. If you are the experienced SAS programmer, it is never too late to start Python.

REFERENCES

"pandas." https://pandas.pydata.org/

"matplotlib." https://matplotlib.org/

"SASPy." https://sassoftware.github.io/saspy/

"Pdfminer." https://media.readthedocs.org/pdf/pdfminer-docs/latest/pdfminer-docs.pdf

CONTACT INFORMATION

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

Yuichi Nakajima
yuichi.nakajima@novartis.com