PharmaSUG SDE Japan

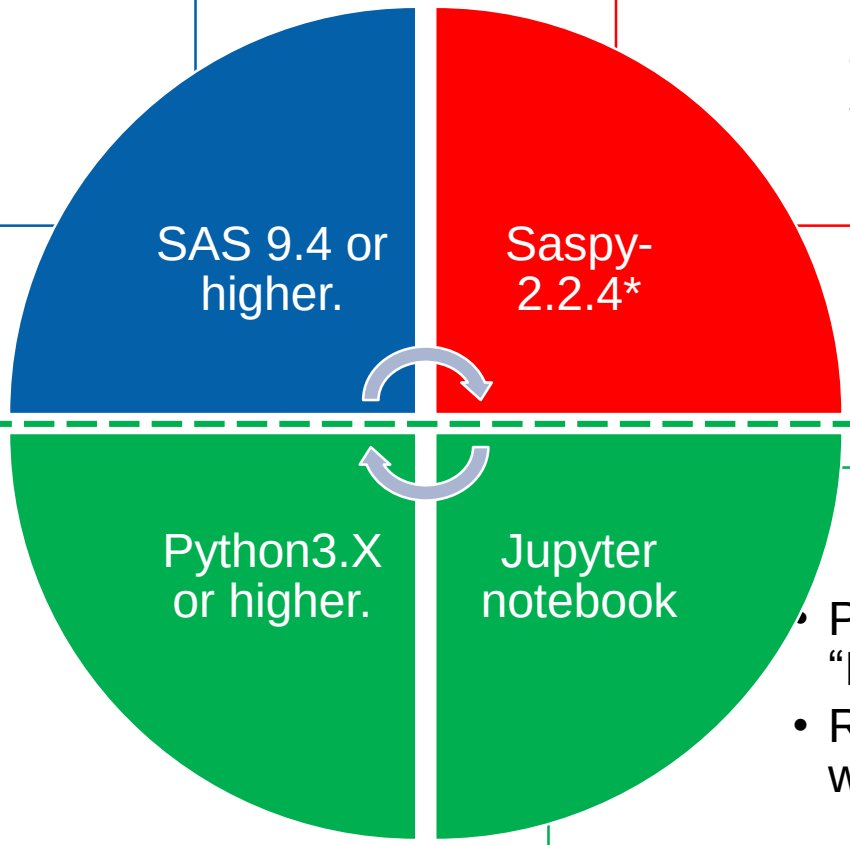
Data Visualization by Python using SAS dataset: Data from Pandas to Matplotlib

Yuichi Nakajima, Principal Programmer,
Novartis
September 4, 2018

NOVARTIS

Pre-requirement

- Focus on "**Windows PC SAS" connection.**
- See reference for other connection type.

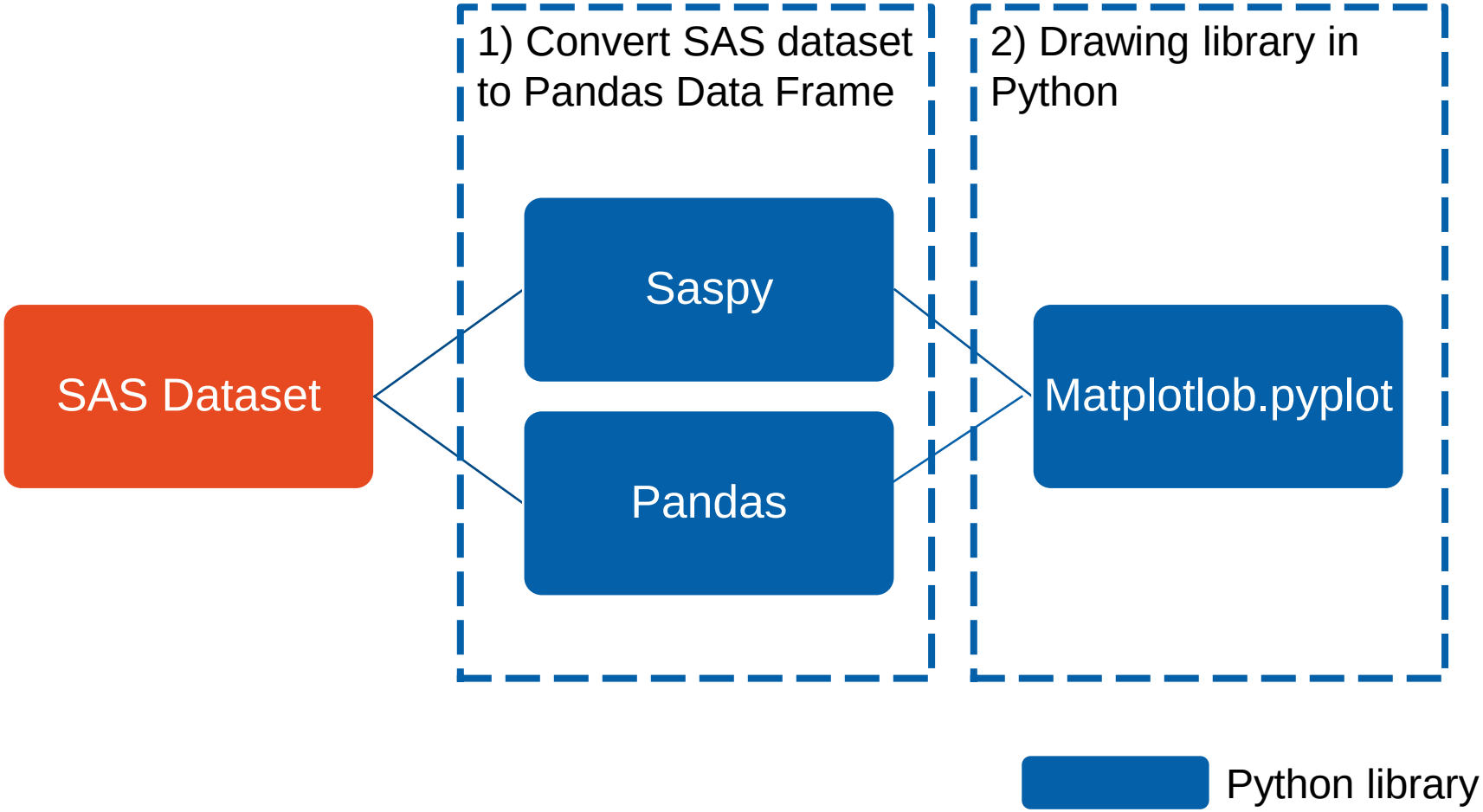


- As of July 2018, v2.2.4 is the latest version.

- Previously called "IPython Notebook".
- Run Python on the web browse.

Available from **Anaconda distribution**

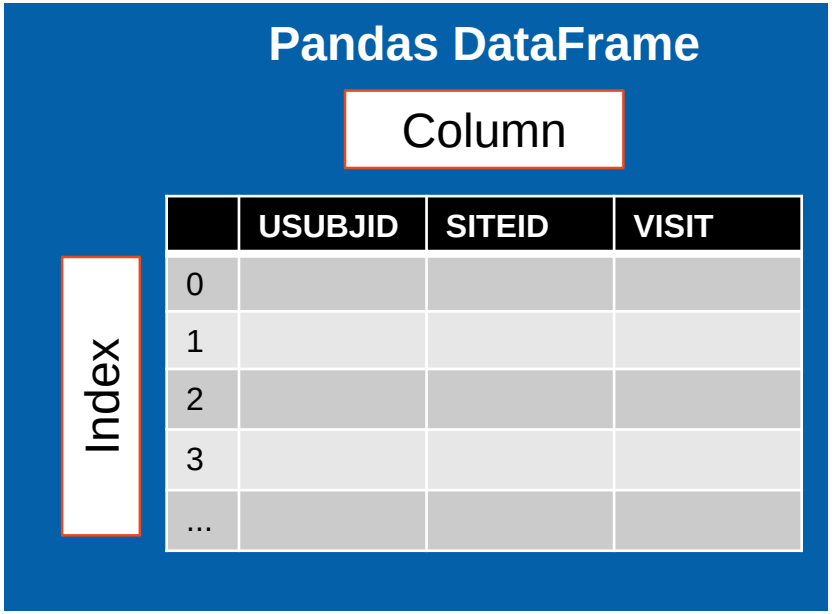
Overview process



1. Access to SAS datasets

- There will be 3 possible way to handle SAS data in Jupyter notebook.
  - **Saspy API (Please refer to SAS User group 2018 Poster)**
  - Jupyter Magic %%SAS
  - **Pandas DataFrame(DF)**

- "Pandas" is the Python Package providing efficient data handling process. Pandas data structures are called "Series" for single dimension like vector and "Dataframe" for two dimensions with "Index" and "Column".



1. Access to SAS datasets

- Import necessary library in Jupyter notebook.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import saspy
```

- Access to SAS datasets (sas7bdat or xpt) and convert to Pandas DF.

1. Use **Pandas** to read SAS dataset (both xpt and sas7bdat are acceptable).

```
# "%cd" is one of magic command.
%cd C:\Users\NAKAJYU1\Desktop\tempds
adsl = pd.read_sas('adsldmy.sas7bdat', format='sas7bdat', encoding="utf-8")
```

2. **Saspy API** to read SAS dataset as sas7bdat. Then covert to Pandas DF.

```
# Create libname by Saspy API
sas.saslib('temp', path="C:\\Users\\NAKAJYU1\\Desktop\\tempds")
# Read SAS datasets in .sas7bdat
advs = sas.sasdata('advsdmy', libref='temp')
```

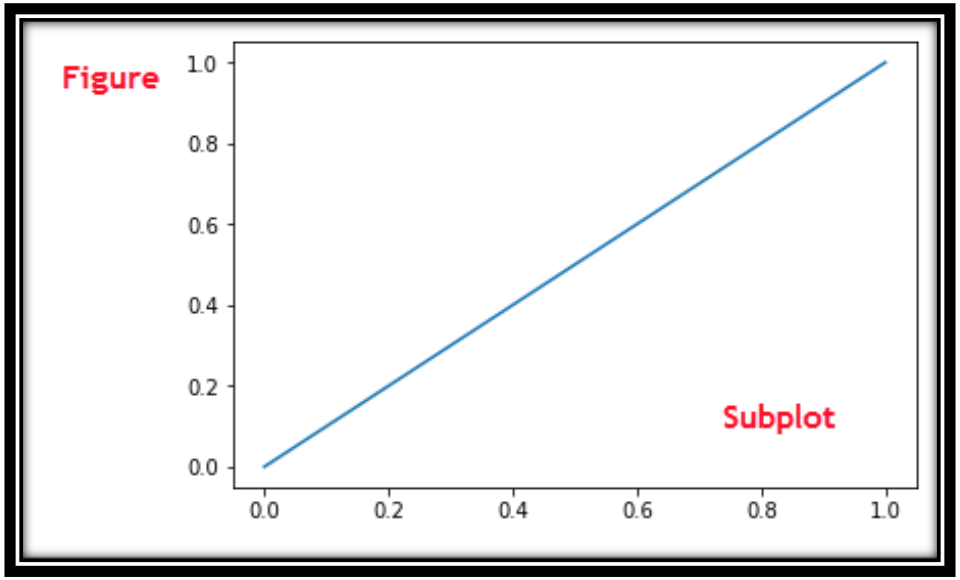
Recommended to use Saspy to avoid character set issue

```
# Convert sas dataset to DF
advsdf = sas.sasdata2dataframe('advsdmy', libref='temp')
```

2. Data Visualization
- Get started -

- In order to plot data by Matplotlib, first generate 1)**figure** and 2)**sub plot**. At least one sub plot must be created.

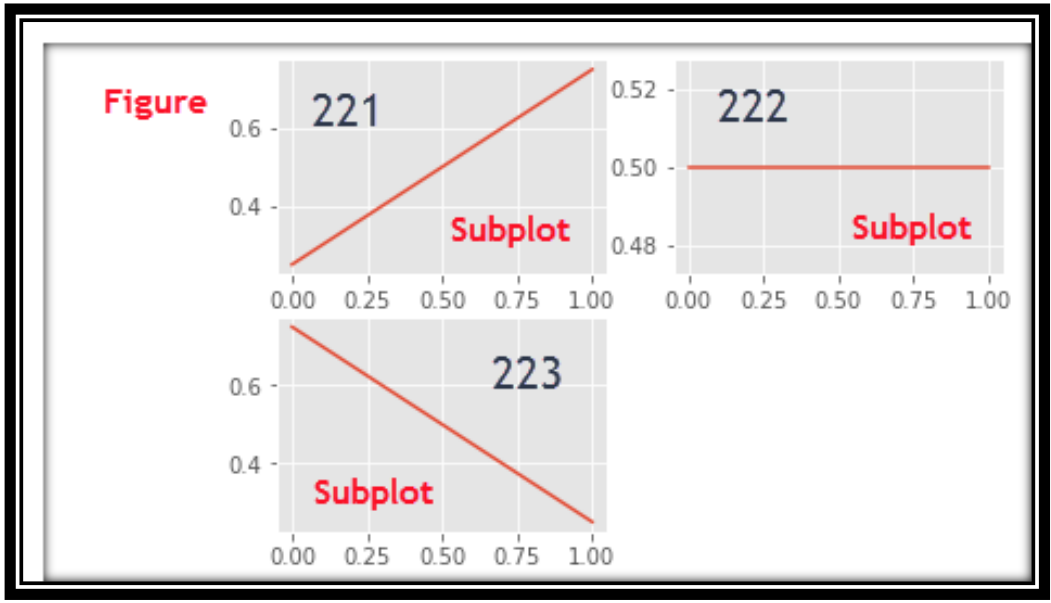
```
# 1) Call figure instance
fig = plt.figure()
# 2) Call subplot
ax = fig.add_subplot(111)

dat = [0, 1]
# Line plot by plot function
ax.plot(dat)

# Display with show function
plt.show()
```



2. Data Visualization
- Get started -

```
#Apply 'ggplot' style to figure
plt.style.use('ggplot')
fig = plt.figure()

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]

ax1.plot(dat1)
ax2.plot(dat2)
ax3.plot(dat3)

plt.show()
```

- Here's an example to show 3 subplots. Applied *'ggplot'* style(added grid line)



2. Data Visualization
- Line Plot 1 / mean with SD plot -

- Prepare summary statistic from data(DF) . “wk1” is a dummy data with pandas DF which is following ADaM BDS structure.

```
#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']
```

sum: Pandas DF
mean1-3: Pandas Series
std1-3: Pandas Series

```
print(sum)
```

		count	mean	std	min	25%	\
TRT01P_a	AVISITN						
DRUG X	0	75.0	117.689374	13.549857	90.298443	109.910094	
	1	74.0	119.439286	14.046465	77.085242	112.187640	
	2	75.0	120.684247	15.039663	86.064269	111.507457	
	7	75.0	121.133844	12.722854	95.318836	111.736148	
	14	74.0	120.9012				
	28	74.0	119.6835				
	56	75.0	120.8798				
DRUG Y	0	52.0	120.0401				
	1	54.0	120.3531				
	2	54.0	119.5331				
	7	54.0	124.5945				

Index:
[TRT01_P, AVISITN]
Column:
[count, mean, std, ...]

```
print(mean1)
```

```
AVISITN
0      117.689374
1      119.439286
2      120.684247
7      121.133844
14     120.901240
28     119.683569
56     120.879800
Name: mean, dtype:
```

Index:
AVISITN
Column:
mean1

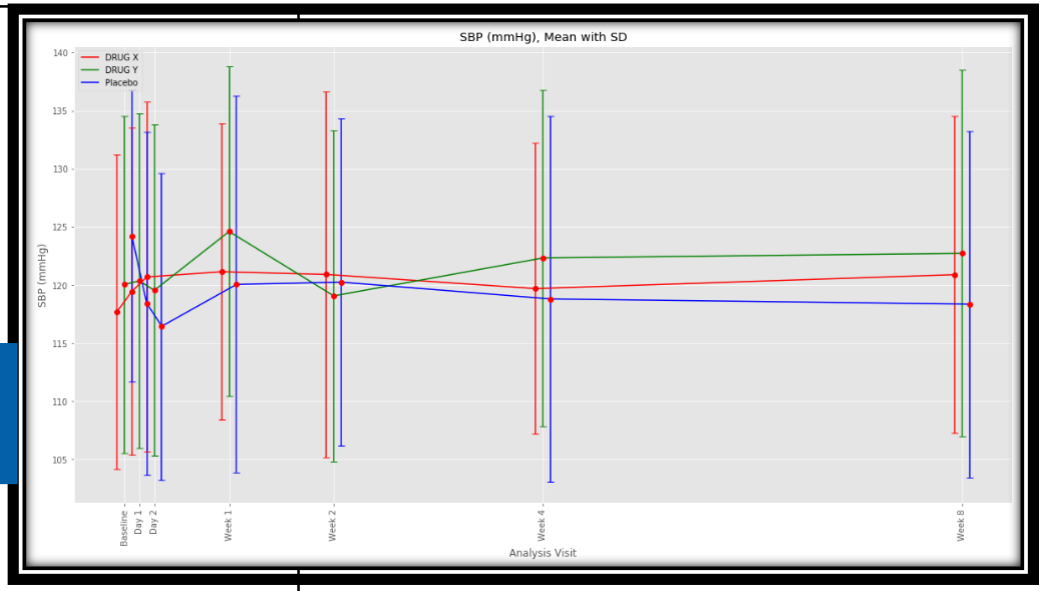
2. Data Visualization
- Line Plot 1 / mean with SD plot -

```
# 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"])
```

```
plt.style.use('ggplot')
fig=plt.figure(figsize=(20,10))
ax = fig.add_subplot(111)

#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')
#Show legend on upper left.
ax.legend(loc="upper left")

#Apply label ticks and labels
ax.set_xticks(vis_num)
ax.set_xticklabels(vis_order, rotation=90)
#Set errorbar by errorbar function
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)')
#Display plot
plt.show()
```

One subplot example

x: AVISITN as index
y: mean



NOVARTIS

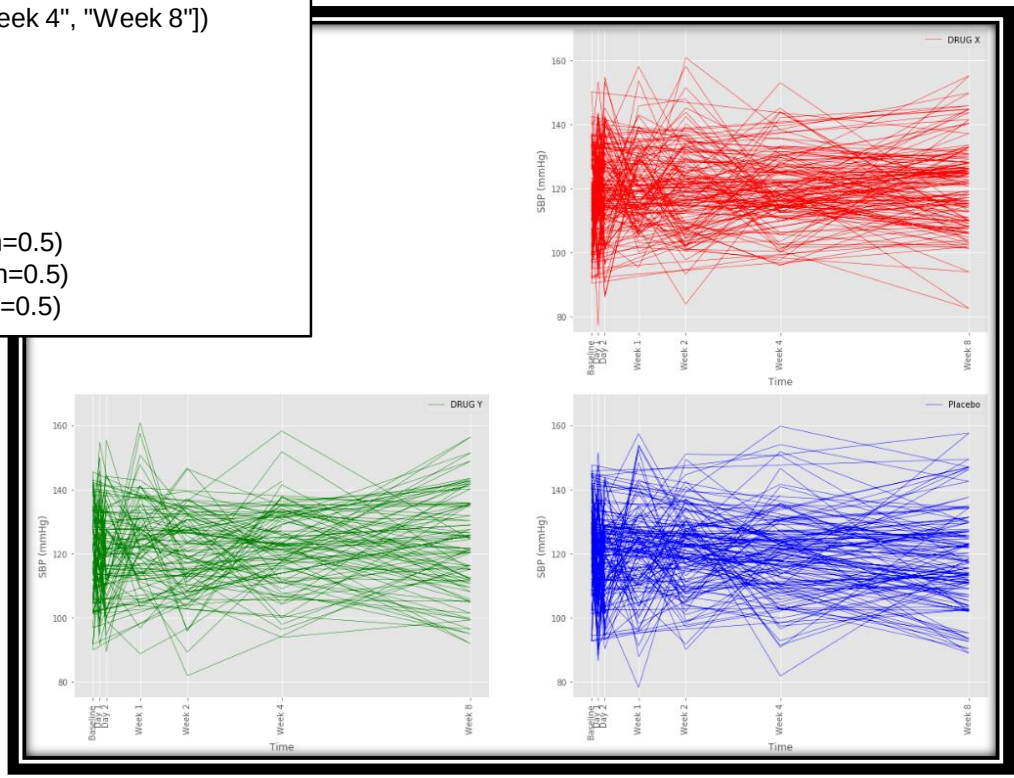
2. Data Visualization
- Line Plot 2 / Patient level plot -

```
# Pre-define DF
wk1 = wk[(wk['PARAMCD']=='STSBPSI') & (wk['AVISITN'] < 199)]
arm1 = wk1.loc[wk1['TRT01P_a']=='DRUG X']
arm2 = wk1.loc[wk1['TRT01P_a']=='DRUG Y']
arm3 = wk1.loc[wk1['TRT01P_a']=='Placebo']
# 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"])
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)
```

Three subplot example

```
# Common setting to each subplot
axes = [ax1, ax2, ax3]
for ax in axes:
    ax.set_ylim(75, 170)
    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()
```

Each plot setting can be done in for loop



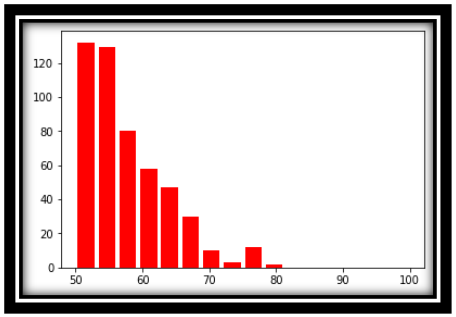
NOVARTIS

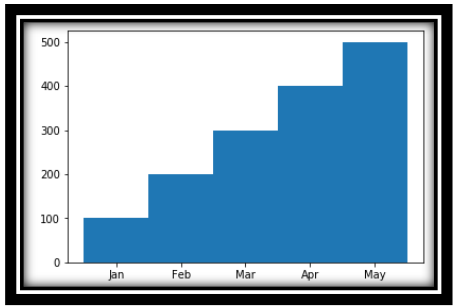
2. Data Visualization - Other plots -

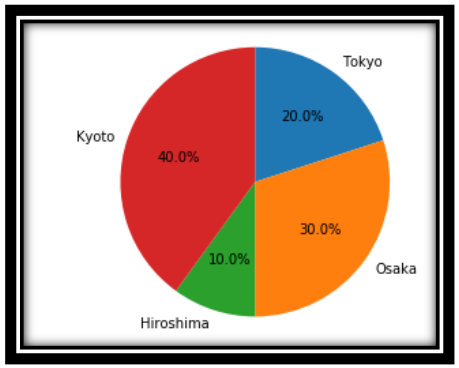
Plot type	Function	Description	Quick examples

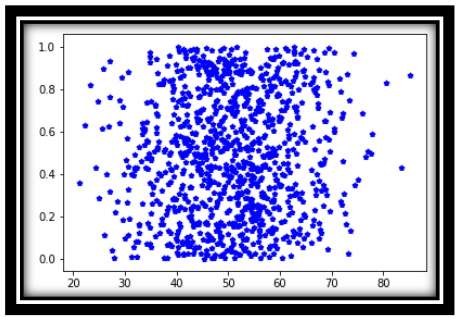
To try below examples, run following code first.
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> x1 = np.array([1, 2, 3, 4, 5])
>>> x2 = np.array([100, 200, 300, 400, 500])
>>> x = np.random.normal(50, 10, 1000)
>>> y = np.random.rand(1000)			

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









Summary

- Pandas and Matplotlib enables you to make an easy data visualization from standardized SAS datasets like CDISC.
- Using Saspy will make a first step to start Python for a SAS programmer. Thus combination of several language with data handling choices (Saspy, Jupyter magic and Pandas), you may find process improvement in your daily work.
- References
  - Matplotlib manual released 2.2.3 https://matplotlib.org/Matplotlib.pdf