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

Yuichi Nakajima, Principal Programmer, Novartis
September 4, 2018
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.

SAS 9.4 or higher.

Saspy-2.2.4*

Python3.X or higher.

Jupyter notebook

Available from Anaconda distribution
Overview process

1) Convert SAS dataset to Pandas Data Frame

2) Drawing library in Python

SAS Dataset

Saspy

Pandas

Matplotlib.pyplot

Python library
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”.

<table>
<thead>
<tr>
<th>USUBJID</th>
<th>SITEID</th>
<th>VISIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. Access to SAS datasets

• Import necessary library in Jupyter notebook.

```python
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).

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

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

```python
# 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')

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

Recommended to use Saspy to avoid character set issue.
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.

```python
# 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 -

```python
# 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.

```python
# Calculate 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']
```

Index: [TRT01_P, AVISITN]
Column: [count, mean, std, ...]
2. Data Visualization
- Line Plot 1 / mean with SD plot -

```python
# 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(mean2.index, mean2, yerr=std2, fmt='ro', ecolor='g', capsize=4)
ax.errorbar(mean3.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()
```
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)

# Adjust width/height spacing
fig.subplots_adjust(wspace=0.2)
plt.show()
# 2. Data Visualization
- Other plots -

<table>
<thead>
<tr>
<th>Plot type</th>
<th>Function</th>
<th>Description</th>
<th>Quick examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Histograms</strong></td>
<td>hist()</td>
<td>Compute and draw the histogram of x.</td>
<td>&gt;&gt;&gt; plt.hist(x, bins=16, range=(50, 100), rwidth=0.8, color='red')</td>
</tr>
<tr>
<td><strong>Bar charts</strong></td>
<td>bar()</td>
<td>The bars are positioned at x with the given align ment. Their dimensions are given by width and height.</td>
<td>&gt;&gt;&gt; plt.bar(x1, x2, width=1.0, linewidth=3, align='center', tick_label=['Jan', 'Feb', 'Mar', 'Apr', 'May'])</td>
</tr>
<tr>
<td><strong>Pie charts</strong></td>
<td>pie()</td>
<td>Make a pie chart of array x. The fractional area of each wedge is given by x/sum(x).</td>
<td>&gt;&gt;&gt; plt.pie(x3, labels=['Tokyo', 'Osaka', 'Hiroshima', 'Kyoto'], countercclock=False, startangle=90, autopct=&quot;%1.1f%&quot;)</td>
</tr>
<tr>
<td><strong>Scatter plot</strong></td>
<td>scatter()</td>
<td>A scatter plot of y vs x with varying marker size and/or color.</td>
<td>&gt;&gt;&gt; plt.scatter(x, y, s=15, c='blue', marker='*', linewidth='2')</td>
</tr>
</tbody>
</table>

To try below examples, run following code first.

```python
>>> 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)
```
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