IE 555 – Programming for Analytics

Module #3 – Data Visualization (Plotting)

This module provides an overview of plotting data in Python. Please take the time to investigate these sources for more information, examples, and alternative approaches:

- http://matplotlib.org/users/pyplot_tutorial.html
- "Python Data Science Handbook" by Jake VanderPlas
- "Python for Data Analysis" by Wes McKinney

1 Intro to Plotting with `matplotlib`

We'll create different types of `matplotlib` plots, and will interact with Python in four different contexts:

1. Terminal
2. IPython
3. Python script (.py file)
4. Jupyter notebook

1.1 From the Terminal

First, open a terminal window and start Python (by issuing the `python3` command).

Then, type the following lines:

A simple plot using Python terminal commands

```
import matplotlib.pyplot as plt

x = [1, 3, 4, 7]
y = [2, 5, 1, 6]

plt.plot(x, y)
plt.show()
```

Note that once you issue the `plt.show()` command, you can't issue any other commands until you close the figure window.

1.2 From IPython

First, open IPython (typically via Anaconda Navigator).

IPython features "magic commands" which are both shortcuts and enable some useful functionality. For more information about magic commands, type `%magic` from within an IPython session. Type `%lsmagic` to see a list of available magic commands.

Creating a basic plot in IPython

```
In [1]: %matplotlib
Using matplotlib backend: Qt5Agg

In [2]: import matplotlib.pyplot as plt

In [3]: x = [1, 3, 4, 7]

In [4]: y = [2, 5, 1, 6]

In [5]: plt.plot(x, y)
Out[5]: [<matplotlib.lines.Line2D at 0x11fb7a320>]

In [6]: plt.plot(x, y, 'o')
Out[6]: [<matplotlib.lines.Line2D at 0x11cb87828>]
```

The `%matplotlib` magic command has two key benefits:

1. We don't need to use `plt.show()`, and
2. We can add more `plt.plot()` functions interactively to customize the plot.
    - However, if your `plt.plot()` modifies an existing element of a plot, you'll need to issue the `plt.draw()` command.

For more information about the `%matplotlib` magic command, type `%matplotlib?` from within an IPython session.

1.3 From a Python Script

Save these lines in a file named "myplot.py"

```
import matplotlib.pyplot as plt

x = [1, 3, 4, 7]
y = [2, 5, 1, 6]

plt.plot(x, y)
plt.show()
```

Note that `plt.show()` should only appear once in a particular script (typically at the very end of the script).

The advantage of using a script versus using python from a terminal window is that you can save your script (and edit and re-run it later).

1.4 From a Jupyter Notebook

Using Anaconda Navigator, start a Jupyter Notebook session. Then, issue the following commands:

A simple plot in a Jupyter notebook

```
%matplotlib notebook

import matplotlib.pyplot as plt

x = [1, 3, 4, 7]
y = [2, 5, 1, 6]

plt.plot(x, y)
```

The `%matplotlib notebook` line is a magic command. It allows interactive plots within the notebook. This is the same functionality as the simple `%matplotlib` magic command in IPython.

Alternatively, you could use the `%matplotlib inline` command, which produces static images in the notebook.

Note that we do not need the `plt.show()` command in the Jupyter notebook.

1.5 How to find your version of `matplotlib`

From within a Python session (either in the terminal, IPython, or a Jupyter notebook), type the following two lines to find the installed version of `matplotlib`:

Finding matplotlib version

```
import matplotlib
matplotlib.__version__        # double underscores
```

2 Basic Plots with Lines and/or Markers

Let's look at ways we can customize a plot with lines and markers. In this section we're going to look at the average high/low monthly temperatures in Buffalo and Miami, as summarized in Table 1.

Table 1: Average monthly temperatures (in degrees Fahrenheit) for Buffalo and Miami.

Month	**Jan**	**Feb**	**Mar**	**Apr**	**May**	**Jun**	**Jul**	**Aug**	**Sep**	**Oct**	**Nov**	**Dec**
Buf High	31	31	41	52	63	73	79	78	70	58	48	35
Buf Low	20	19	27	38	49	59	66	64	57	45	37	26
Mia High	73	74	75	78	81	84	85	86	85	82	78	76
Mia Low	65	66	69	73	76	79	80	81	80	77	71	69

A code snippet for Jupyter Notebooks to get us started

```
# Use a "magic command" to specify how we want our plots
# displayed.  Choose one of the following two options:
# %matplotlib notebook  # (dynamic/interactive)
# %matplotlib inline    # (static)
%matplotlib inline

import matplotlib.pyplot as plt

y = [31, 31, 41, 52, 63, 73, 79, 78, 70, 58, 48, 35]
x = list(range(1,13))

plt.plot(x, y)
```

2.1 Use markers for the data points

Our first plot simply showed line segments. What if we wanted a scatter plot instead? Let's test out these commands individually:

Using markers

```
plt.plot(x, y, 'o')
plt.plot(x, y, '+')
plt.plot(x, y, '^')
plt.plot(x, y, 'v')
plt.plot(x, y, 's')
```

2.2 Setting marker colors

We can also change the color of the markers:

Colorizing markers

```
plt.plot(x, y, 'rD')          # red diamonds
plt.plot(x, y, 'gH')          # green hexagons
plt.plot(x, y, 'yo')          # yellow dots
```

2.3 Change the size of the markers

Sometimes we want our markers to be sized differently from the default size. This is easy to do:

Changing marker size

```
plt.plot(x, y, 'rD', markersize=12)
plt.plot(x, y, 'rD', ms=12)
```

2.4 Draw the marker with a different outline color

Our markers can have outline and fill colors that are different.

Diamond markers filled with red and outlined in blue

```
plt.plot(x, y, 'rD', ms=12, markeredgecolor='blue')
plt.plot(x, y, 'rD', ms=12, mec='blue')
plt.plot(x, y, 'rD', ms=12, markeredgecolor='blue',
    markeredgewidth=5)
plt.plot(x, y, 'rD', ms=12, mec='blue', mew=5)
```

2.5 More marker customization

Marker Fill Styles:
https://matplotlib.org/gallery/lines_bars_and_markers/marker_fillstyle_reference.html#sphx-glr-gallery-lines-bars-and-markers-marker-fillstyle-reference-py

Marker Reference:
https://matplotlib.org/gallery/lines_bars_and_markers/marker_reference.html#sphx-glr-gallery-lines-bars-and-markers-marker-reference-py

2.6 Color the line

We'll now include lines connecting our markers. By default, the line and markers will be the same color. But we can customize this.

Setting line and marker colors

```
# Markers and line are the same color (red):
plt.plot(x, y, 'rD-', ms=12)

# Green hexagon markers and a red line:
plt.plot(x, y, 'H-', ms=12, color='red', mfc='green')
```

2.7 Customizing the line style

Customized line styles, using dots as markers

```
plt.plot(x, y, 'o--')
plt.plot(x, y, 'o', linestyle='dashed')
plt.plot(x, y, 'o', ls='dashed')
plt.plot(x, y, ls='dashed', marker='o')
plt.plot(x, y, 'o-')
plt.plot(x, y, 'o--')
plt.plot(x, y, 'o-.')
plt.plot(x, y, 'o:')
```

For more information, see:
https://matplotlib.org/gallery/lines_bars_and_markers/line_styles_reference.html#sphx-glr-gallery-lines-bars-and-markers-line-styles-reference-py

2.8 Custom hex colors

A small collection of basic colors are provided. To really customize your plot you can use "hex" colors. More information can be found here:
http://www.w3schools.com/colors/colors_picker.asp

A pink dashed line with dot markers

```
plt.plot(x, y, 'o--', color='#ff8080')
```

2.9 Add a title to our plot

A rather nondescript plot title

```
plt.title('Plot Title')
```

2.10 Add axis labels

Adding axis labels

```
80 plt.ylabel('Temperature [$^\circ$F]')
81 plt.xlabel('Month')
```

Note that we've used LaTeX formatting to show the degree symbol.

2.11 Replace month numbers with names

More descriptive tick labels

```
82 months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', \
83     'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
84 plt.xticks(x, months)
```

2.12 Add a second data series and a legend

Let's add the Miami high temperatures and a legend.

Very obvious axis labels

```
85 miami_high = [73, 74, 75, 78, 81, 84, 85, 86, 85, 82, 78, 76]
86
87 plt.plot(x, y, ls='dashed', marker='o', label='Buffalo')
88 plt.plot(x, miami_high, 'd', label='Miami')
89 plt.legend()
```

2.13 Add grid lines

Add grid lines

```
90 plt.grid()
```

2.14 Change the range of our figure

By default, the axes will be scaled to exactly the minimum and maximum values of what we're plotting. This has the effect of "cutting off" any data points on the extremes. To fix this issue, we'll expand the range of our figure. This may take some trial-and-error to get it right, depending on the scale of your data.

Increase the x- and y-scales

```
91 # plt.axis([xmin, xmax, ymin, ymax])
92 plt.axis([min(x)-1, max(x)+1, min(y)-0.5, max(y)+0.5])
```

3 Scatter Plots

What if we want our data points to be differently sized? The basic `plot` command doesn't allow this (all markers will be the same size and color). Suppose we have demand information for Widgets, as shown in Table 2. We might want to scale the markers such that cities with larger demand have larger markers.

Table 2: Widget demand and location data.

City	Demand	x	y
Buffalo	5,000	50	30
Dallas	9,000	25	15
Los Angeles	10,000	5	5
St. Louis	3,000	32	20

3.1 A basic scatter plot

Creating a scatter plot

```
import matplotlib.pyplot as plt

city = ['Buffalo', 'Dallas', 'Los Angeles', 'St. Louis']
x = [50, 25, 5, 32]
y = [30, 15, 5, 20]
demand = [5000, 9000, 10000, 3000]

plt.title('Widget Demand')

plt.scatter(x, y, s = demand)
```

Details on "scatter" may be found here: http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.scatter

3.2 Changing colors based on demand

Suppose we want our demand bubbles to be colored differently, based on the magnitude of demand at each location.

Changing colors and including transparency

```
colors = []
for i in range(0, len(demand)):
    colors.append(demand[i]/float(max(demand)))

plt.scatter(x, y, s=demand, c=colors, alpha=0.5)
```

3.3 Adding text labels

It's difficult to tell which bubble goes with which city. Let's label our bubbles with the name of each city.

To do this, we'll need to loop over all of the cities to add individual text labels.

Adding text labels

```
for i in range(0,len(city)):
    myx = x[i]
    myy = y[i]
    mycity = city[i]

    plt.text(myx, myy, mycity, color="red", fontsize=12)
```

3.4 Formatting text labels

Suppose we want our text labels to be centered in the bubbles. This is easy to do by editing the "`plt.text(...)`" command we used previously:

Centering text labels

```
    plt.text(myx, myy, mycity, color="red", fontsize=12,
   horizontalalignment='center', verticalalignment='center')
```

4 Bar Charts

Let's return to our weather example and plot Buffalo's monthly high temperatures in a bar chart.

A code snippet for a basic bar chart

```
import matplotlib.pyplot as plt

y = [31, 31, 41, 52, 63, 73, 79, 78, 70, 58, 48, 35 ]
x = list(range(1,13))
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', \
    'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

plt.bar(x, y, tick_label=months, width=0.8)
# plt.show()
```

More info: https://matplotlib.org/api/_as_gen/matplotlib.pyplot.bar.html

5 Histograms

A code snippet for a basic bar chart

```
102 import matplotlib.pyplot as plt
103
104 y = [31, 31, 41, 52, 63, 73, 79, 78, 70, 58, 48, 35 ]
105 x = list(range(1,13))
106
107 # Try the following individually:
108 # n, bins, patches = plt.hist(y, bins=3, density=False,
      facecolor='g', alpha=0.75)
109 # n, bins, patches = plt.hist(y, bins=range(0,110,10), density
      =False, facecolor='g', alpha=0.75)
110
111 print(n)
112 print(bins)
```

More info: https://matplotlib.org/api/_as_gen/matplotlib.pyplot.hist.html

6 Saving Plots as Images

If you've created a spectacular plot, you probably want to share it with others. The best way to do this is to save the plot as an image file.

Saving plots as .png or .pdf

```
1 plt.savefig('my_plot.png')
2 plt.savefig('my_plot.pdf')
```

See what happens if you zoom in on the .png image. Does it get pixelated? What happens with the .pdf version?

You might notice that the above commands leave lots of extra whitespace around the plot. This can be removed as follows:

Reducing whitespace from plots

```
3 plt.savefig('my_plot_trimmed.png', bbox_inches='tight')
4 plt.savefig('my_plot_trimmed.pdf', bbox_inches='tight')
```

7 Creating Multiple Plots

Suppose you want to create one figure containing multiple plots (i.e., "subplots"). We'll need to use "axes" within `matplotlib` to do this.

For example, let's create two separate line plots (with markers) showing the monthly high and low temperatures in Buffalo.

Creating subplots

```
%matplotlib inline

import matplotlib.pyplot as plt

# Our Data:
x = list(range(1,13))
y_high = [31, 31, 41, 52, 63, 73, 79, 78, 70, 58, 48, 35]
y_low  = [20, 19, 27, 38, 49, 59, 66, 64, 57, 45, 37, 26]

# Labels for the months:
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', \
    'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

# Create two subplots sharing the y axis
fig, (ax1, ax2) = plt.subplots(2, sharey=True)

# Add a title for the entire figure:
fig.suptitle('Buffalo Monthly Temperatures')

# Plot the high temperatures in the top subplot:
ax1.plot(x, y_high, 'ko-')

# Add a title for the top subplot, and a label for its y-axis:
ax1.set(title='An unnecessary label for this subplot', ylabel=
   'High [$^\circ$F]')

# Remove the tick marks for the x-axis in the top subplot:
ax1.set_xticks([])

# Plot the low temperatures in the bottom subplot:
ax2.plot(x, y_low, 'r.-')

# Add a label for the x-axis in the bottom subplot, and
# provide a label for the bottom y-axis:
ax2.set(xlabel='Month', ylabel='Low [$^\circ$F]')

# Set the tickmarks in the bottom subplot to show each x value
   :
```

```
ax2.set_xticks(x)

# Replace the numeric x-axis values with our months:
ax2.set_xticklabels(months)
```

More examples: https://matplotlib.org/gallery/subplots_axes_and_figures/subplots_demo.html

8 Other Examples

Some of these use `numpy`, which we will study in the next module.

8.1 Error Bars

Basic error bars

```
import matplotlib.pyplot as plt

y = [31, 31, 41, 52, 63, 73, 79, 78, 70, 58, 48, 35 ]
x = list(range(1,13))
errors = [3, 9, 8, 2, 7, 6, 3, 7, 5, 4, 8, 6]

plt.errorbar(x, y, yerr=errors, fmt='.k')
```

Customized error bars

```
plt.errorbar(x, y, yerr=errors, fmt='*', color='green', ecolor
    ='blue', elinewidth=2, ms='12')
```

In IPython or Jupyter Notebook, type `plt.errorbar?` for documentation.

8.2 Box-and-Whisker Plots

This example comes from https://matplotlib.org/gallery/pyplots/boxplot_demo_pyplot.html#sphx-glr-gallery-pyplots-boxplot-demo-pyplot-py

A simple boxplot

```
import numpy as np
import matplotlib.pyplot as plt

# Fixing random state for reproducibility
np.random.seed(19680801)

# fake up some data
spread = np.random.rand(50) * 100
```

```
center = np.ones(25) * 50
flier_high = np.random.rand(10) * 100 + 100
flier_low = np.random.rand(10) * -100
data = np.concatenate((spread, center, flier_high, flier_low))

fig1, ax1 = plt.subplots()
ax1.set_title('Basic Plot')
ax1.boxplot(data)
```

8.3 3D Plots

This example comes from https://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html

A 3D line plot

```
%matplotlib notebook

import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt

mpl.rcParams['legend.fontsize'] = 10

fig = plt.figure()
ax = fig.gca(projection='3d')
theta = np.linspace(-4 * np.pi, 4 * np.pi, 100)
z = np.linspace(-2, 2, 100)
r = z**2 + 1
x = r * np.sin(theta)
y = r * np.cos(theta)
ax.plot(x, y, z, label='parametric curve')
ax.legend()
```

8.4 `matplotlib` Styles

Setting `matplotlib` styles

```
%matplotlib inline

import matplotlib.pyplot as plt

y = [31, 31, 41, 52, 63, 73, 79, 78, 70, 58, 48, 35]
x = list(range(1,13))


```

```
# Try each of the following individually
# with plt.style.context('classic'):
#   plt.plot(x, y)

# with plt.style.context('dark_background'):
#   plt.plot(x, y)

# with plt.style.context('grayscale'):
#   plt.plot(x, y)

# import seaborn
# plt.plot(x, y)
```

8.5 Seaborn

Seaborn is an alternative to `matplotlib`. You may find that it produces plots that are more visually appealing.

Please explore https://seaborn.pydata.org/examples/index.html for Seaborn examples. I'm particularly fond of violin plots as an alternative to box-and-whisker plots.

9 More Info

Check out these links for some interesting applications of `matplotlib`:

- Gallery – https://matplotlib.org/gallery/
- Tutorials – https://matplotlib.org/tutorials/