Getting Started with Matplotlib

We need matplotlib.pyplot for plotting.

```
import matplotlib.pyplot as plt
import pandas as pd
```

About the Data

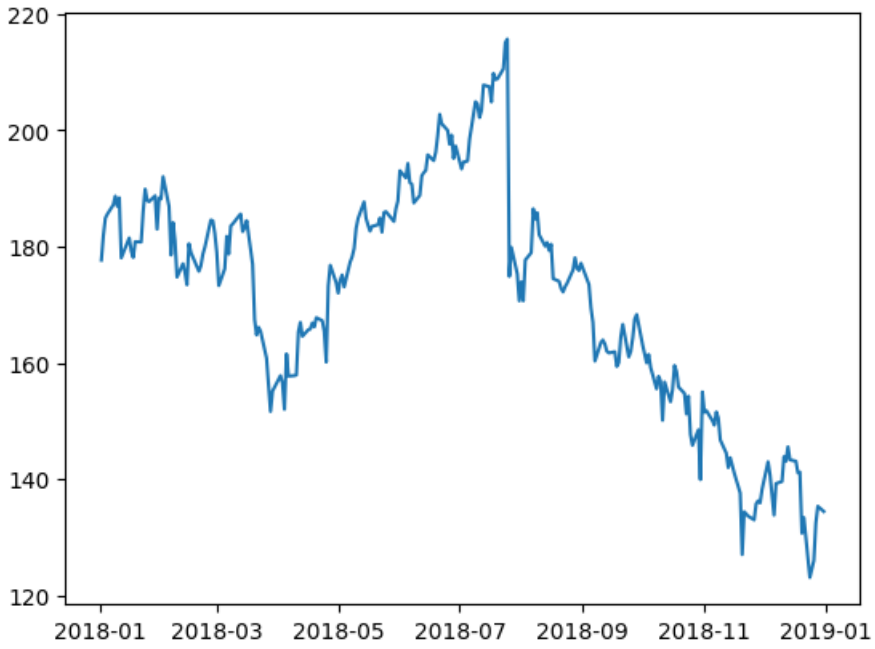
In this notebook, we will be working with 2 datasets:

- Facebook's stock price throughout 2018 (obtained using the stock_analysis package)
- Earthquake data from September 18, 2018 - October 13, 2018 (obtained from the US Geological Survey (USGS) using the USGS API)

Plotting lines

```
fb = pd.read_csv(
    'data/fb_stock_prices_2018.csv', index_col='date', parse_dates=True #Inserting the dates in a column
)

plt.plot(fb.index, fb.open)
plt.show() #Showing the output of the graph
```



Since we are working in a Jupyter notebook, we can use the magic command %matplotlib inline once and not have to call plt.show() for each plot.

```
%matplotlib inline #The %matplotlib inline will automatically show the plot without using th plt.show()
import matplotlib.pyplot as plt
```

```
import pandas as pd

fb = pd.read_csv(
    'data/fb_stock_prices_2018.csv', index_col='date', parse_dates=True
)
plt.plot(fb.index, fb.open)
```

Scatter plots

We can pass in a string specifying the style of the plot. This is of the form '[color][marker][linestyle]'. For example, we can make a black dashed line with 'k--' or a red scatter plot with 'ro':

```
# The scatter plots will show an output of the plot with just dots and no lines.
plt.plot('high', 'low', 'ro', data=fb.head(20))
```

Histograms

```
# Histogram is a graph that shows the frequency of numerical data using rectangles.
quakes = pd.read_csv('data/earthquakes-1.csv')

# The plt.hist will show the output of the plot into Histogram format
plt.hist(quakes.query('magType == "ml"').mag)
```

Bin size matters

Notice how our assumptions of the distribution of the data can change based on the number of bins (look at the drop between the two highest peaks on the righthand plot):

```
x = quakes.query('magType == "ml"').mag
fig, axes = plt.subplots(1, 2, figsize=(10, 3))
for ax, bins in zip(axes, [7, 35]):
    ax.hist(x, bins=bins)
    ax.set_title(f'bins paramd: {bins}')
```

Plot components

Figure

Top-level object that holds the other plot components.

```
fig = plt.figure()
```

Axes

Individual plots contained within the Figure.

Creating Subplots

Simple specify the number of rows and columns to create:

```
fig, axes = plt.subplots(1, 2)
```

As an alternative to using plt.subplots() we can add the Axes to the Figure on our own. This allows for some more complex layouts, such as picture in picture:

```
fig = plt.figure(figsize=(3,3))
outside = fig.add_axes([0.1, 0.1, 0.9, 0.9])
```

```
inside = fig.add_axes([0.7, 0.7, 0.25, 0.25])
```

Creating Plot Layouts with gridspec

we can create subplots with varying sizes as well:

```
fig = plt.figure(figsize=(8, 8))
gs = fig.add_gridspec(3, 3)
top_left = fig.add_subplot(gs[0, 0])
mid_left = fig.add_subplot(gs[1, 0])
top_right = fig.add_subplot(gs[:2, 1:])
bottom = fig.add_subplot(gs[2,:])
```

Saving plots

Use plt.savefig() to save the last created plot. To save a specific Figure object, use its savefig() method.

```
fig.savefig('empty.png') #Used to save the created plot
```

Cleaning up

It's important to close resources when we are done with them. We use plt.close() to do so. if we pass in nothing, it will close the last plot, but we can pass the specific **Figure** to close or say **'all'** to close all **Figure** objects that are open. Let's close all the **Figure** objects that are open with plt.close():

```
plt.close('all') #Closing all Figure objects that are open
```

Additional plotting options

Specifying figure size Just pass the **figsize** parameter to plt.figure(). it's a tuple of (width, height):

```
fig = plt.figure(figsize=(10,4))
```

This can be specified when creating subplots as well:

```
fig, axes = plt.subplots(1, 2, figsize=(10, 4))
```

rcParams

A small subset of all the available plot setting (shuffling to get a good variation of options):

```
import random
import matplotlib as mpl

rcparams_list = list(mpl.rcParams.keys())
random.seed(20) # make this repeatable
random.shuffle(rcparams_list)
sorted(rcparams_list[:20])
```

```
['animation.convert_args',
 'axes.edgecolor',
 'axes.formatter.use_locale',
 'axes.spines.right',
 'boxplot.meanprops.markersize',
 'boxplot.showfliers',
 'keymap.home',
 'lines.markerfacecolor',
 'lines.scale_dashes',
 'mathtext.rm',
 'patch.force_edgecolor',
 'savefig.facecolor',
 'svg.fonttype',
 'text.hinting_factor',
 'xtick.alignment',
 'xtick.minor.top',
 'xtick.minor.width',
 'ytick.left',
 'ytick.major.left',
 'ytick.minor.width']
```

We can check the current default **figsize** using **rcParams**:

```
mpl.rcParams['figure.figsize'] #Checking the default figure size using rcParams:
```

```
[6.4, 4.8]
```

We can also update this value to change the default (until the kernel is restarted):

```
mpl.rcParams['figure.figsize'] = (300, 10)
mpl.rcParams['figure.figsize']
```

```
[300.0, 10.0]
```

Use **rcdefaults()** to restore the defaults:

```
mpl.rcdefaults()
mpl.rcParams['figure.figsize']
```

```
[6.4, 4.8]
```

This can also be done via **pyplot**:

```
plt.rc('figure', figsize=(20, 20)) # change figsize default to (20, 20)
plt.rcdefaults() # reset the default
```