⌄ Getting Started with Matplotlib

Name: Cuadra, Audrick Zander G.

Section: CPE22S3

Date: March 30, 2024

Submitted to: Engr. Roman Richard

We need `matplotlib.pyplot t` for plotting.

```
1 import matplotlib.pyplot as mpl
2 import pandas as p
```

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

```
1 fb = p.read_csv(
2     "/content/fb_stock_prices_2018.csv", index_col='date', parse_dates=True
3 )
4
5 mpl.plot(fb.index, fb.open)
6 mpl.show()
```
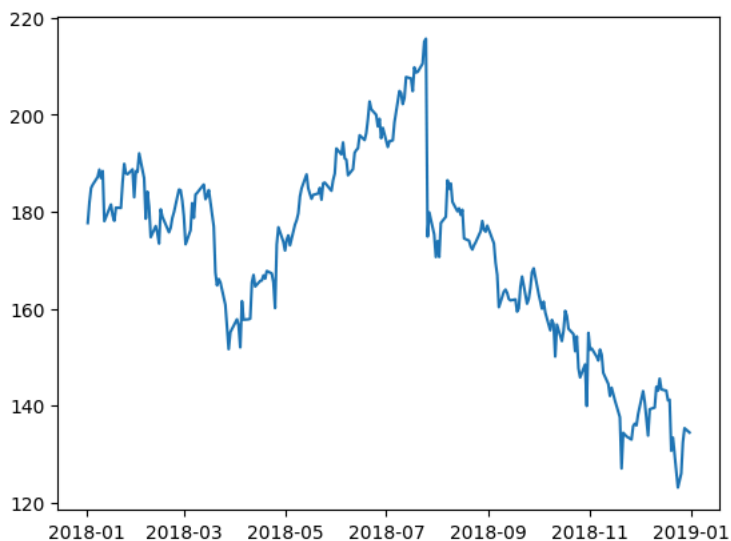

Since we are working in a Jupyter notebook, we can use the magic command `%matplotlib inline` once and not have to call `plt.show()` for each plot.

```
1 %matplotlib inline
2 import matplotlib.pyplot as mpl
3 import pandas as p
4
5 fb = p.read_csv(
6     "/content/fb_stock_prices_2018.csv", index_col='date', parse_dates=True
7 )
8 mpl.plot(fb.index, fb.open)
```
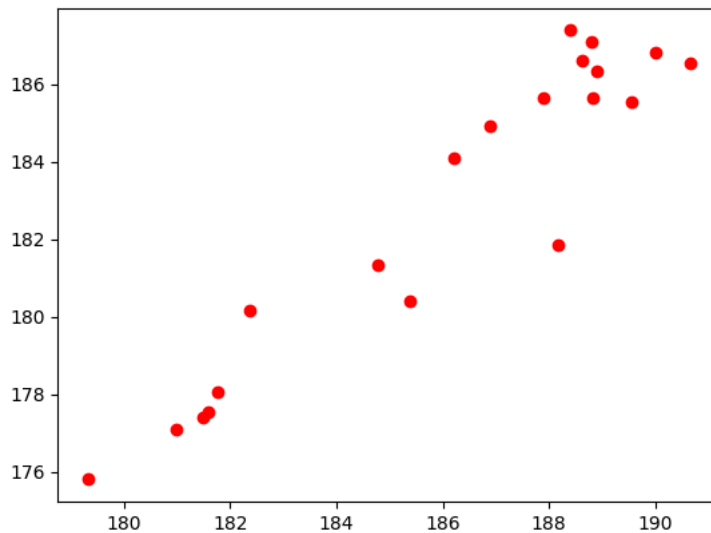
```
[<matplotlib.lines.Line2D at 0x7ea8435373a0>]
```


Scatter Plot

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'` :

```
1 mpl.plot('high', 'low', 'ro', data=fb.head(20))
```
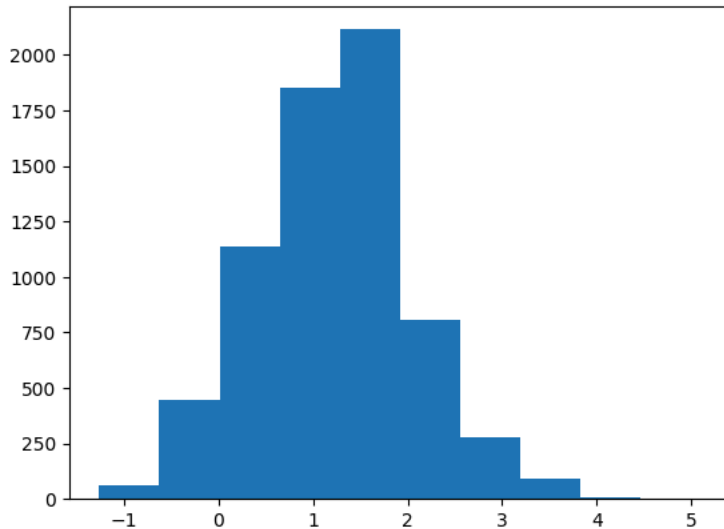
```
[<matplotlib.lines.Line2D at 0x7ea84140c880>]
```


Histograms

```
1 quakes = p.read_csv("/content/earthquakes.csv")
2 mpl.hist(quakes.query('magType == "ml"').mag)
```

```
(array([6.400e+01, 4.450e+02, 1.137e+03, 1.853e+03, 2.114e+03, 8.070e+02,
        2.800e+02, 9.200e+01, 9.000e+00, 2.000e+00]),
 array([-1.26 , -0.624,  0.012,  0.648,  1.284,  1.92 ,  2.556,  3.192,
         3.828,  4.464,  5.1  ]),
 <BarContainer object of 10 artists>)
```


⌄ 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):

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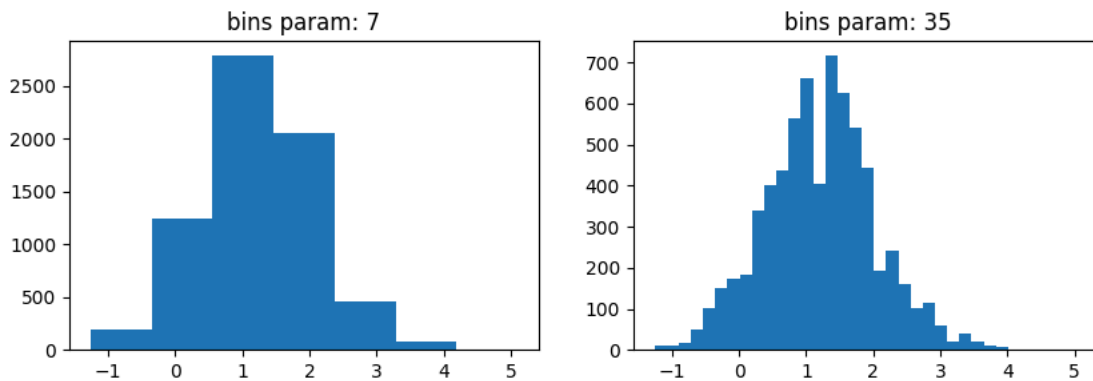

⌄ Plot components

Figure

Top-level object that holds the other plot components.

```
1 fig = mpl.figure()
```
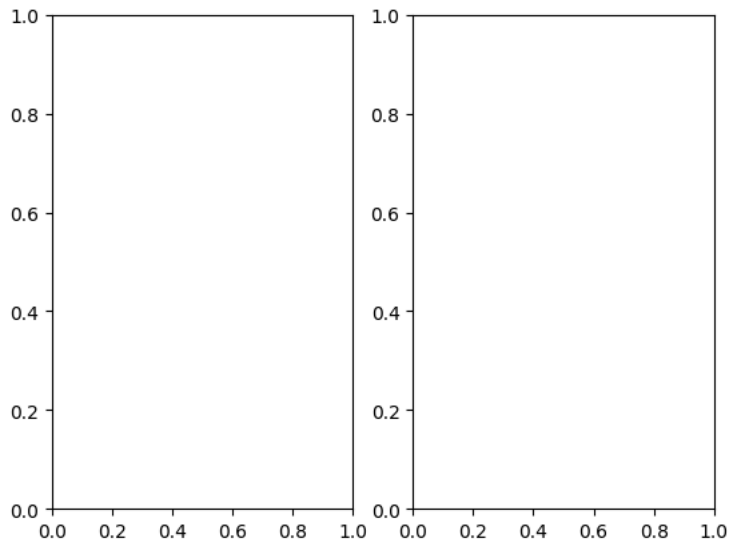
```
<Figure size 640x480 with 0 Axes>
```

Axes

Individual plots contained within the `Figure`.

Creating subplots

Simply specify the number of rows and columns to create:

```
1 fig, axes = mpl.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:
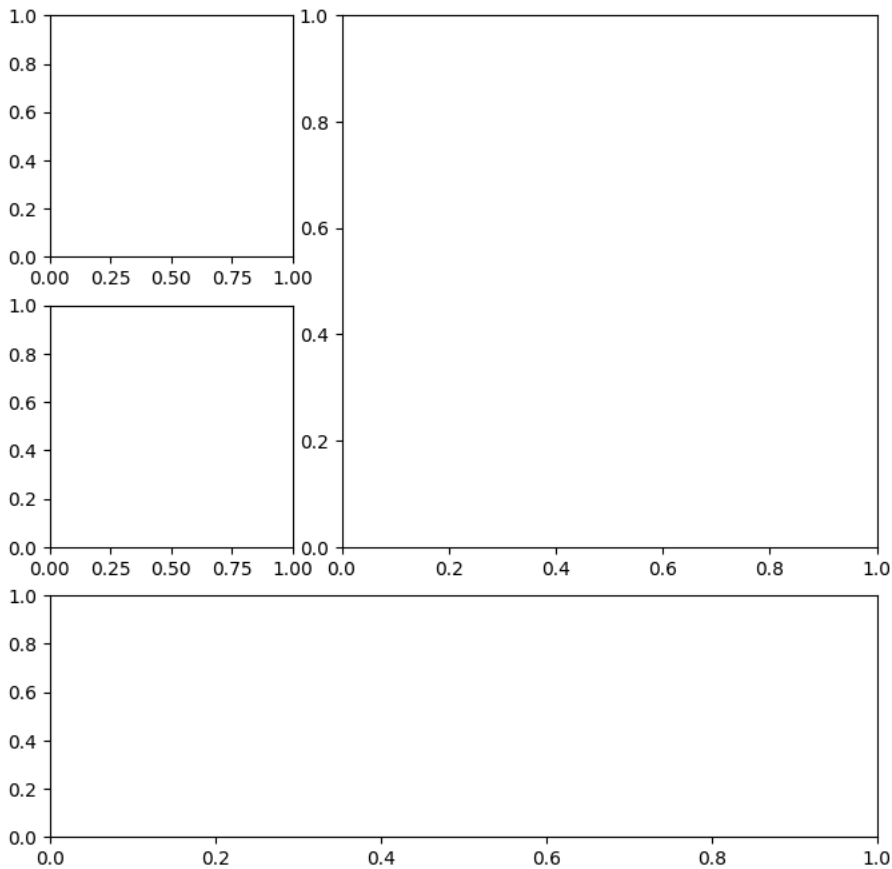
```
1 fig = mpl.figure(figsize=(3, 3))
2 outside = fig.add_axes([0.1, 0.1, 0.9, 0.9])
3 inside = fig.add_axes([0.7, 0.7, 0.25, 0.25])
```
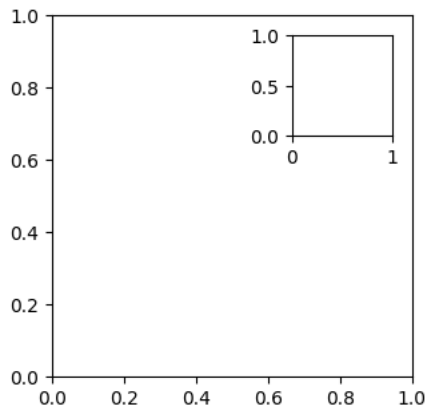

Creating Plot Layouts with gridspec

We can create subplots with varying sizes as well:

```
1 fig = mpl.figure(figsize=(8, 8))
2 gs = fig.add_gridspec(3, 3)
3 top_left = fig.add_subplot(gs[0, 0])
4 mid_left = fig.add_subplot(gs[1, 0])
5 top_right = fig.add_subplot(gs[:2, 1:])
6 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.

```
1 fig.savefig('empty.png')
```

⌄ 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()`:

```
1 mpl.close('all')
```
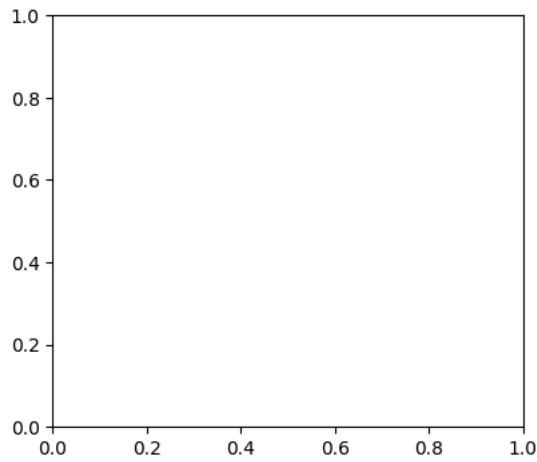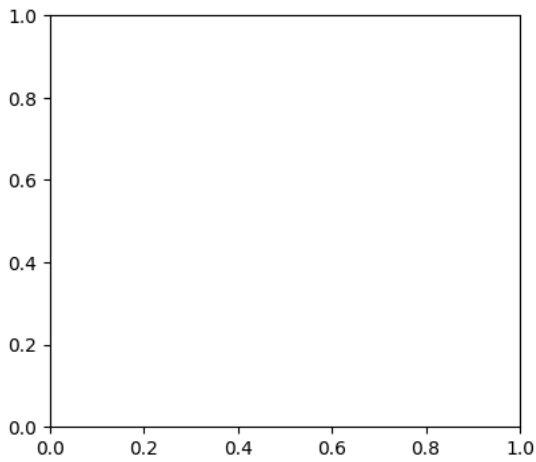
⌄ Additional plotting options

Specifying figure size

Just pass the `figsize` parameter to `plt.figure()`. It's a tuple of (width, height):

```
1 fig = mpl.figure(figsize=(10, 4))
```

```
<Figure size 1000x400 with 0 Axes>
```

This can be specified when creating subplots as well:

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

rcParams

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

```
1 import random
2 import matplotlib as mtb
3
4 rcparams_list = list(mtb.rcParams.keys())
5 random.seed(20) # make this repeatable
6 random.shuffle(rcparams_list)
7 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`:

```
1 mpl.rcParams['figure.figsize']
```

```
[6.4, 4.8]
```

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

```
1 mtb.rcParams['figure.figsize'] = (300, 10)
2 mtb.rcParams['figure.figsize']
```

```
[300.0, 10.0]
```

Use `rcdefaults()` to restore the defaults:

```
1 mtb.rcdefaults()
2 mtb.rcParams['figure.figsize']
```

        [6.4, 4.8]

This can also be done via `pyplot`:

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