Week 3

This week's assignments:

1. Watch this video: `https://learning.oreilly.com/videos/beginning-python/9781786468994/9781786468994-video7_1`

2. Go to **this** link and watch the following:
    a. Basic Matplotlib graph
    b. Labels, titles, and window buttons
    c. Legends
    d. Histogram
    e. Scatter plot
    f. Modifying labels and adding a grid
    g. Customizing ticks

5.3 Matplotlib

Matplotlib is typically used to make graphs. The graphs you make in Matplotlib are very powerful because you can customize them to your exact specifications. For those of you who have made graphs in Excel, you know that customizing those graphs beyond the default settings can be difficult and confusing. With Matplotlib, creating and customizing graphs is much easier. Some students, after learning Matplotlib, begin to use Python in other classes just for the ease of creating graphs with it. In this section, we will go over how to create different types of graphs and how to customize them.

5.3.1 Importing Matplotlib

Since Matplotlib is a package, we need to *import* it into Python. Do this at the top of your Jupyter Notebook:

```
1 import matplotlib.pyplot as plt
```

matplotlib.pyplot is the name of the package, and plt is the nickname we give it. Take a brief look at the code in the sections below, starting with section 5.3.2. Notice all the new commands in this section have the syntax, *plt.command()*. This is because the command comes from the package matplotlib, which we have named plt.

Let's say we gave matplotlib a different nickname, like so:

```
1 import matplotlib.pyplot as mat
```

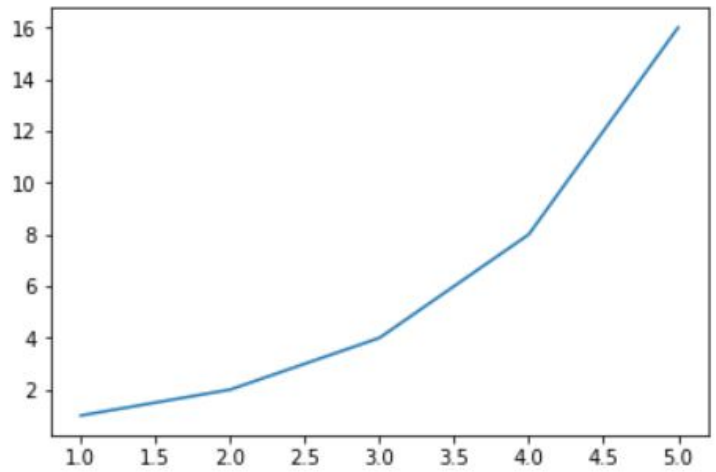
Since we named the package *mat*, all the commands you see below would have the syntax, *mat.command()*. Keep this in mind as you go through the homework in this course. If you name your Matplotlib package something besides plt, make sure that you change the commands that use the package as well.

5.3.2 Line Plots

Line plots are really simple to do. First determine the x-axis and what you want on the y-axis. Now you will use two simple lines of code. Let's say you've created two arrays, one labeled x, and the other labeled y. You want to plot these arrays on a graph with x on the x-axis and y on the y-axis.

```
1 x = np.array([1, 2, 3, 4, 5])
2 y = np.array([1, 2, 4, 8, 16])
3 plt.plot(x, y)
4 plt.show()
```

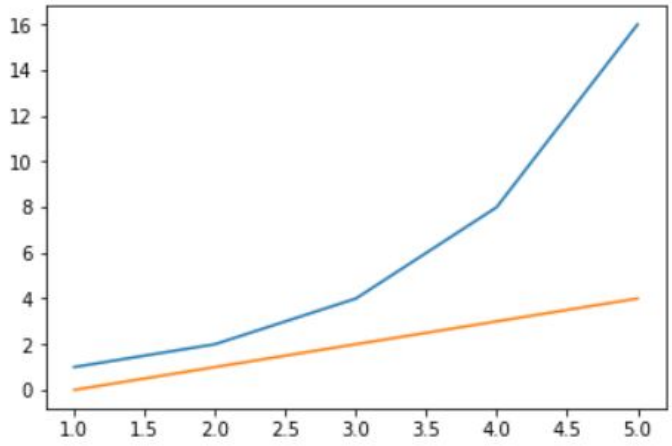
The command *plt.plot()* creates the plot. Notice the variable you want on the x-axis is listed first inside the parenthesis, and the variable that you want on the y-axis is listed after a comma. The command *plt.show()* is what makes your graph appear on the output screen. When you run the code above, you should get a graph that looks like this:



5.3.3 Multiple Lines on a Line Plot

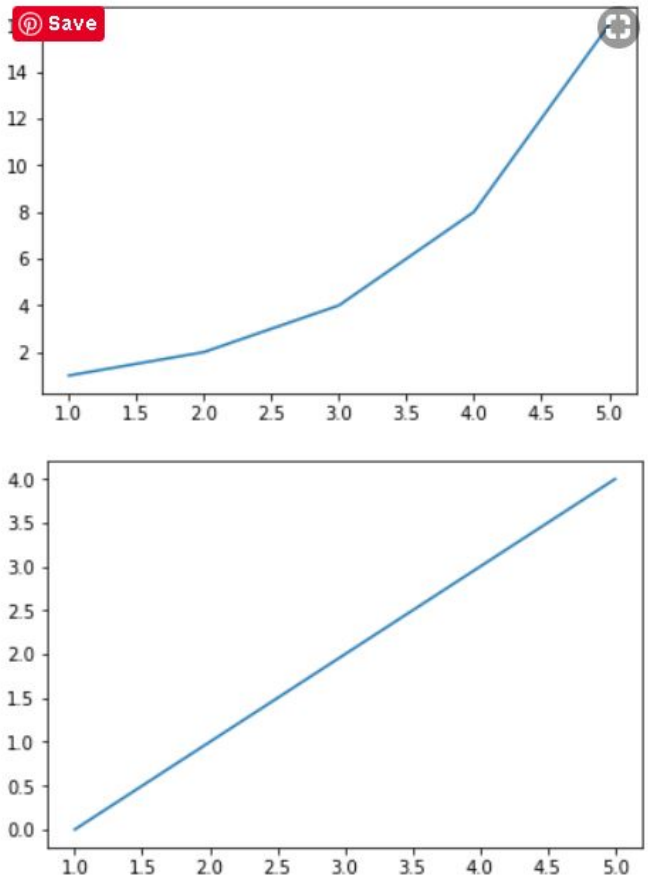
If you want a line plot graph with multiple lines, then you would do something like this:

```
x = np.array([1, 2, 3, 4, 5])
y = np.array([1, 2, 4, 8, 16])
z = np.array([0, 1, 2, 3, 4])
plt.plot(x, y)
plt.plot(x, z)
plt.show()
```



Notice, when I plotted the two lines, I didn't put the *plt.show()* command until the very end. This is because the *plt.show()* command signals Python that you are done working on that graph and are ready to move on. If you put *plt.show()* in between the two *plt.plot()* commands, then you will get something like this:

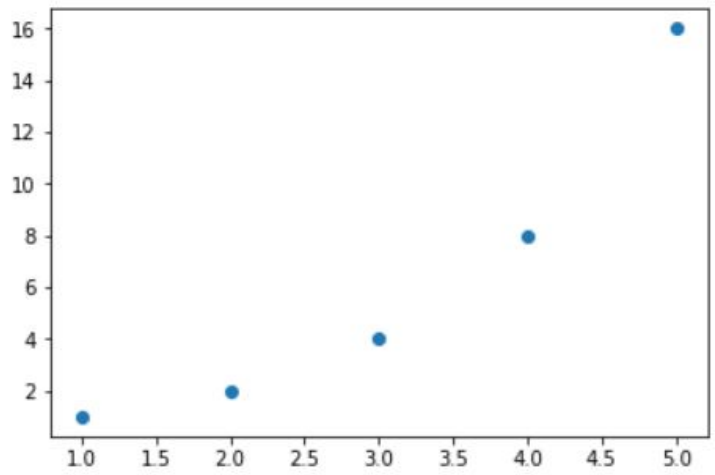
```
x = np.array([1, 2, 3, 4, 5])
y = np.array([1, 2, 4, 8, 16])
z = np.array([0, 1, 2, 3, 4])
plt.plot(x, y)
plt.show()
plt.plot(x, z)
plt.show()
```



5.3.4 Scatter Plots

You can create a scatter plot the same way that you create a line plot, but with a slightly different command. Instead of plt.plot, you use plt.scatter:

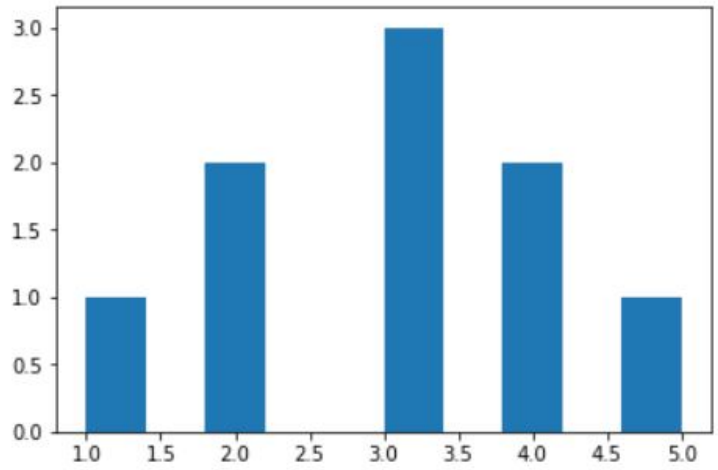
```
1 x = ([1, 2, 3, 4, 5])
2 y = ([1, 2, 4, 8, 16])
3 plt.scatter(x, y)
4 plt.show()
5
```



5.3.5 Histograms

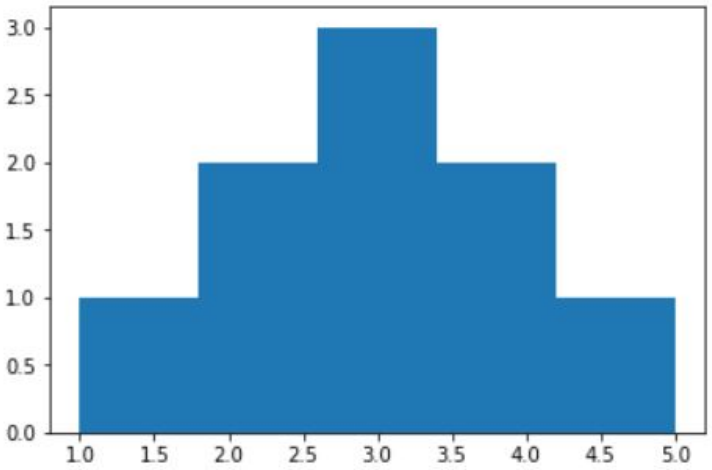
Histograms are also created the same way as the other two plots, but you combine the x and y plots under the same function, a. The y-axis function (also seen as b) is not needed because it plots the frequency of the event. To create a histogram, use plt.hist.

```
1 a = ([1, 2, 2, 3, 3, 3, 4, 4, 5])
2 plt.hist(a)
3 plt.show()
```



You've probably noticed the graph above looks a little bit weird. This is because Python automatically creates histograms with 10 bars. In the example above, there is a space for 1, 1.5, 2, 2.5, and so forth. Since the array doesn't contain any decimals, it shows up as gaps on the histogram. To fix this, change the number of bars there are in your histogram. Specify the number of bins, or sections, that you want in your histogram. If you want to separate you code into 5 sections, then you would do so like this:

```
plt.hist(a, bins = 5)
plt.show()
```

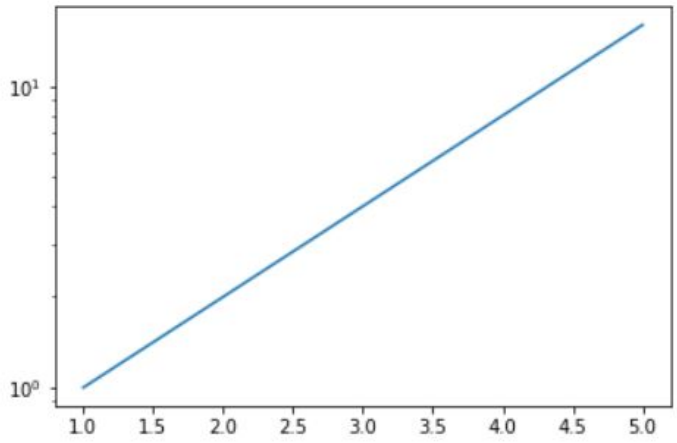


Now there are no gaps in the histogram because the histogram does not have any extra bins.

5.3.6 Customization

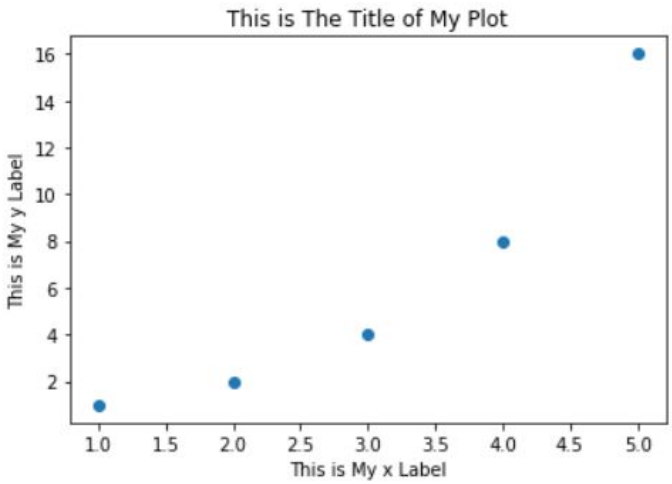
In between plt.plot and plt.show, you can insert a number of style formats to make your graph more readable. A few examples are listed below.
Logarithmic Scale A logarithmic scale is usually a nonlinear scale used to analyze a large range of quantities. In this example, the equation does not create a nonlinear graph, but this function certainly does allow you to create one. You can change your graph to logarithmic scale by adding plt.yscale to your code:

```
plt.plot(x, y)
plt.yscale('log')
plt.show()
```



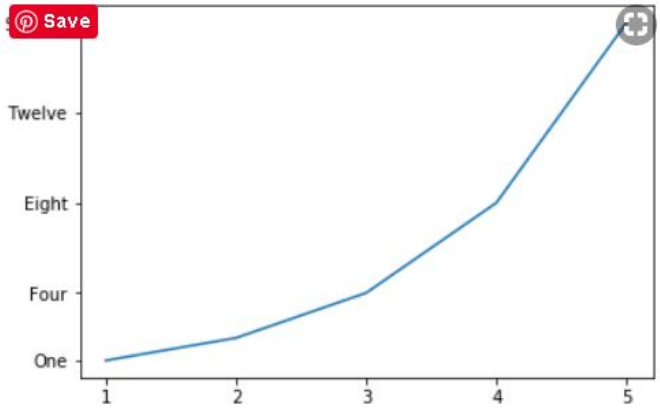
Labels You can label the x-axis, y-axis, and the title of the graph by using plt.xlabel, plt.ylabel, and plt.title respectively. For example:

```
plt.scatter(x, y)
plt.xlabel('This is My x Label')
plt.ylabel('This is My y Label')
plt.title('This is The Title of My Plot')
plt.show()
```



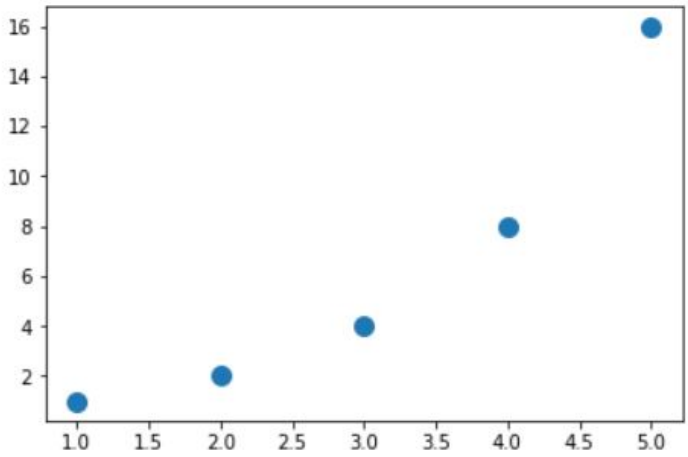
Ticks Tick marks provide reference points along the axes. You can change the amount of space between ticks and what each tick is called in your graphs. To change them, you need to know what numbers you want to mark and what you want each mark to say. For example:

```
#This is the formatting:
#plt.xticks(tickvalue, ticklabel)
plt.plot(x, y)
plt.xticks([1, 2, 3, 4, 5], ['1', '2', '3', '4', '5'])
plt.yticks([1, 4, 8, 12, 16], ['One', 'Four', 'Eight','
    Twelve', 'Sixteen'])
plt.show()
```



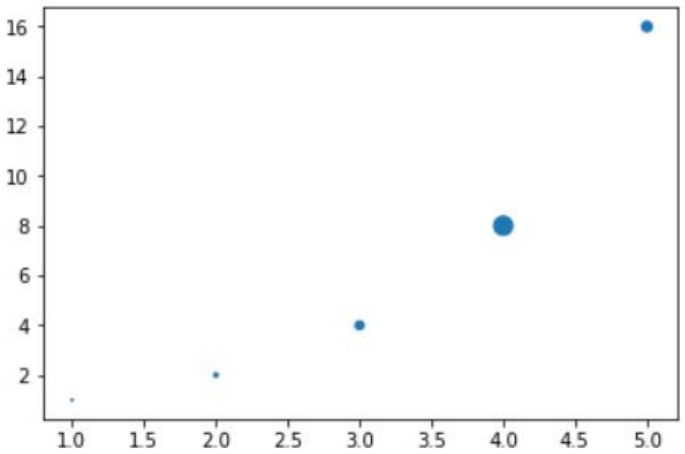
Sizes You can also change the sizes of the scatter plot points within the plt.scatter command:

```
plt.scatter(x, y, s= 100)
plt.show()
```



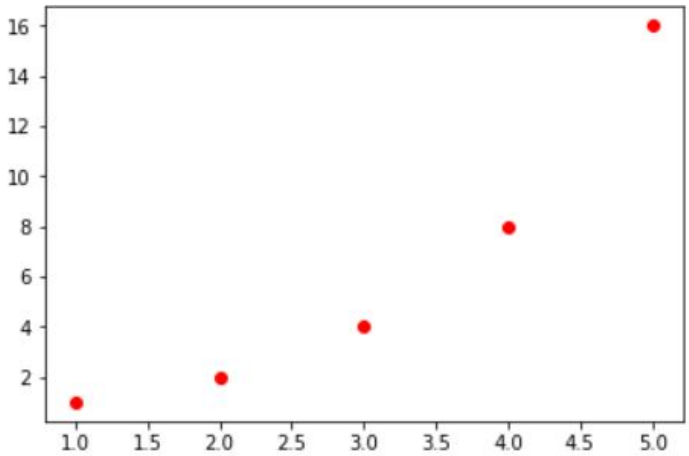
You can even create an array and put that in the size section if you want each point to be a different size. Just be sure that your x, y, and size arrays are all the same length.

```
size_array = np.array([1, 5, 20, 100, 30])
plt.scatter(x, y, s= size_array)
plt.show()
```



Color Change the color of your plot by adding the c= command to your plt.plot command. You can find a list of colors and their code in Python by Googling "Python colors."

```
plt.scatter(x, y, c= 'red')
plt.show()
```



Datetime In this course, you may want to have the date on the x or y-axis. To learn about Datetime, read this article.

https://docs.scipy.org/doc/numpy/reference/arrays.datetime.html