5

Data Visualization

Lesson Objectives:

By the end of this lesson, you will be able to:

* This
* This
* This

Introduction

Data visualizations are powerful tools allowing the user to digest large amounts of data very quickly. Plots come in a variety of shapes, sizes, and styles. In business, line plots and bar graphs are very common to display trends over time and compare metrics across groups, respectively. Statisticians, on the other hand, may be more interested in checking correlations between variables using a scatterplot or correlation heat map. They may also use histograms to check the distribution of a variable or boxplots to check for outliers. In politics, pie charts are widely used for comparing the total data between or among categories. Data visualizations can be very intricate and creative, being limited only by one’s imagination. The Python library Matplotlib (https://matplotlib.org/index.html) is a well-documented, 2-dimensional plotting library which can be used to create a variety of powerful data visualizations and aims to “...make easy things easy and hard things possible” ([https://matplotlib.org/index.html](https://matplotlib.org/3.0.3/index.html)). On the Matplotlib home page (<https://matplotlib.org/index.html>) are instructions for installation (https://matplotlib.org/users/installing.html) as well as the documentation. Additionally, Matplotlib provides a very comprehensive gallery of examples with the corresponding code.

For example, if we are wanting to create a bar chart and we want to see some of the examples that have already been completed we would begin by clicking on the *examples* link from the Matplotlib home page.



Figure 5.1. Matplotlib home page (https://matplotlib.org/index.html)

In the gallery, we can see hundreds of examples, but if we want to see an example of a horizontal bar chart, we can select the *Horizontal bar chart* example.

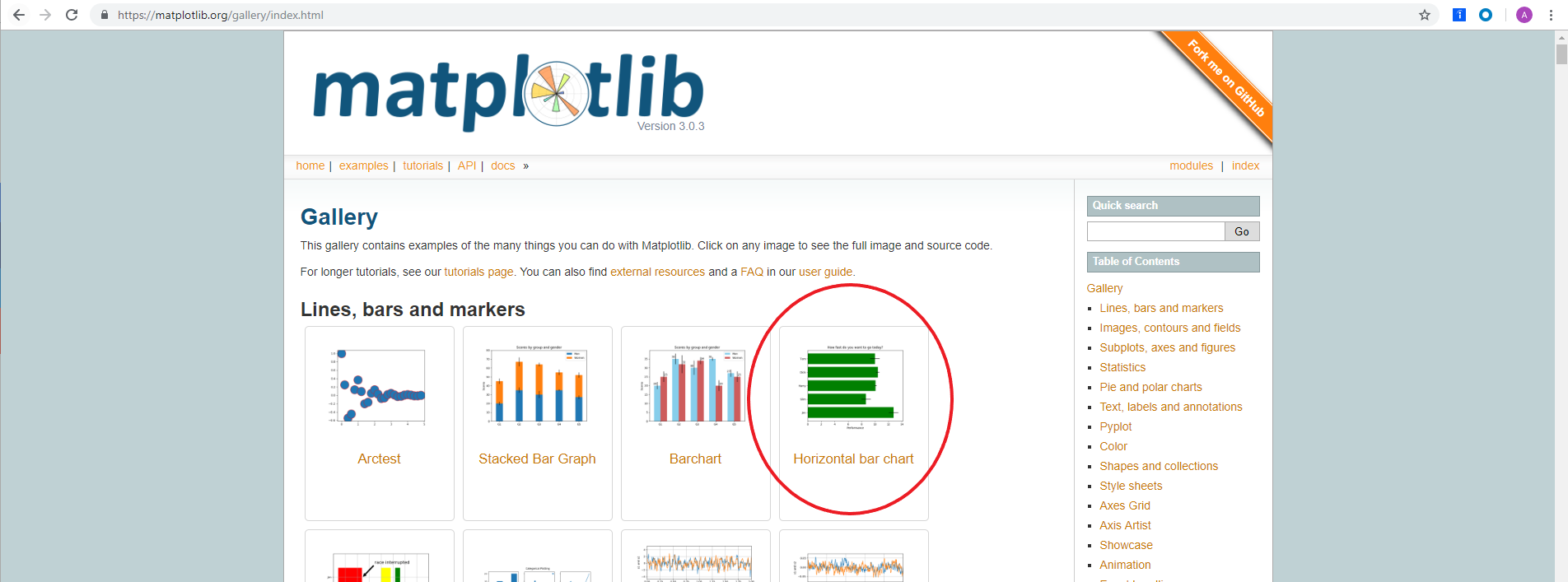


Figure 5.2. Matplotlib gallery (https://matplotlib.org/gallery/index.html)

This brings to a page displaying a picture of the chart as well as the code used to construct the plot which can be downloaded as a Python file or a Jupyter Notebook.

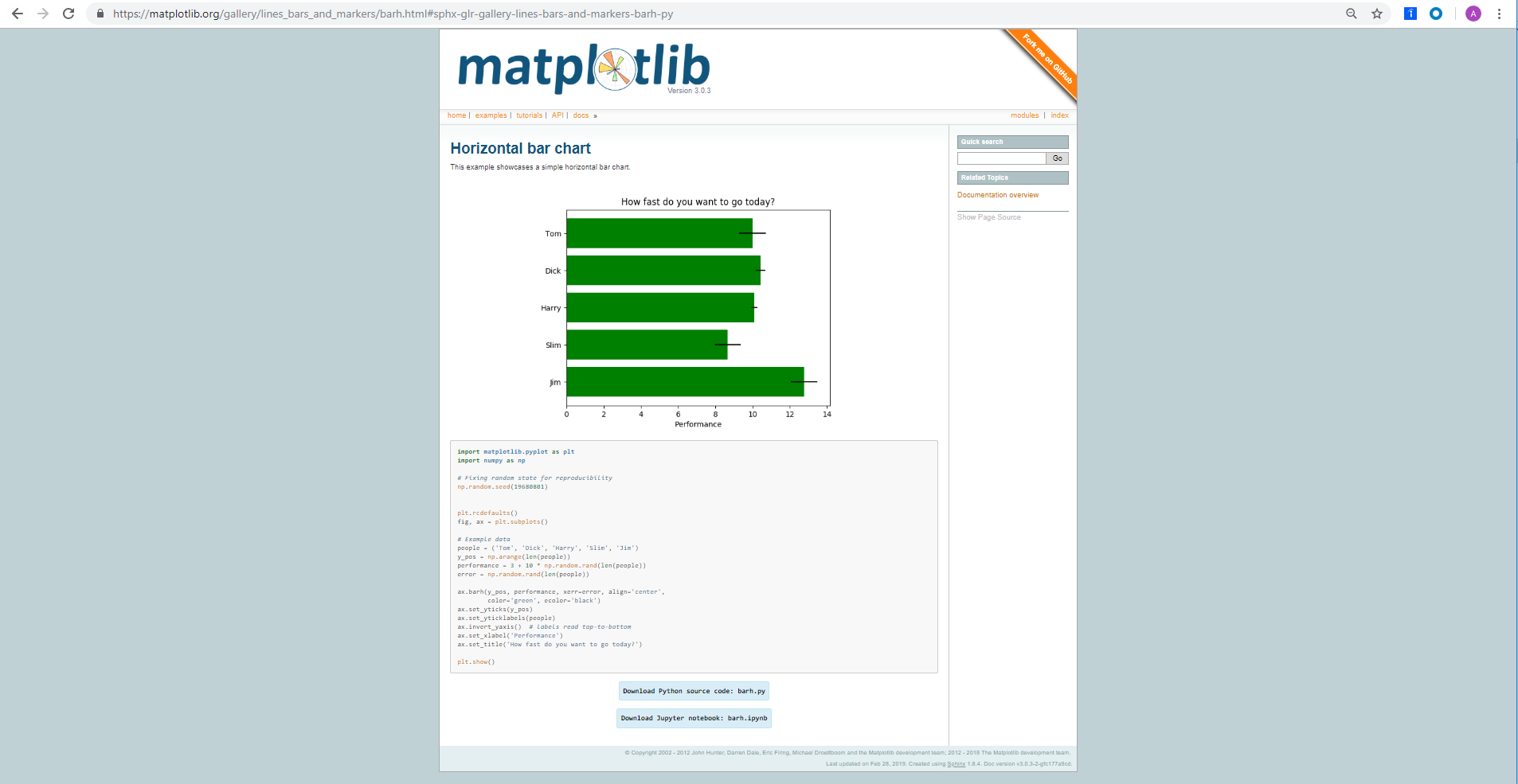


Figure 5.3. Matplotlib horizontal bar chart example (https://matplotlib.org/gallery/lines\_bars\_and\_markers/barh.html#sphx-glr-gallery-lines-bars-and-markers-barh-py)

There are two methods in which plots are created using Matplotlib, the functional method and the object-oriented method.

Functional Method

In the functional method.......

Exercise 1: Line Plot

To learn matplotlib, we will begin with learning how to make a line plot and progress from there.

1. Import matplotlib using import matplotlib.pyplot as plt.



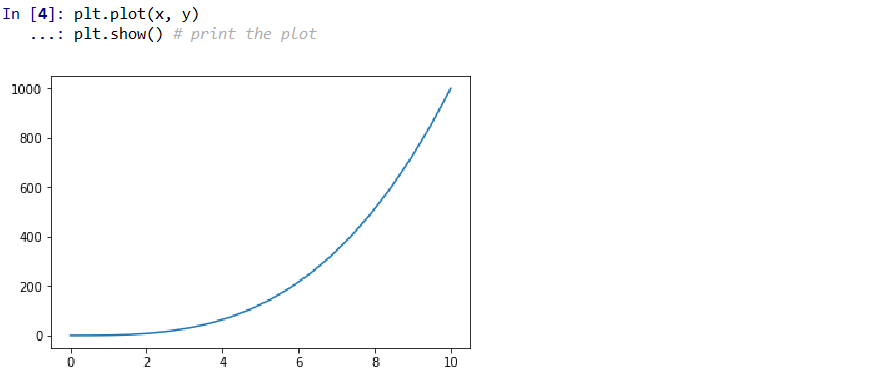
1. Create an array and save it as the object x. By first importing numpy, we are able to create an array of numbers ranging 0 to 10 in 20 evenly spaced values using x = np.linspace(0, 10, 20).



1. Create an array and save it as object y. The snippet of code below cubes the values of x and saves it to the array, y.



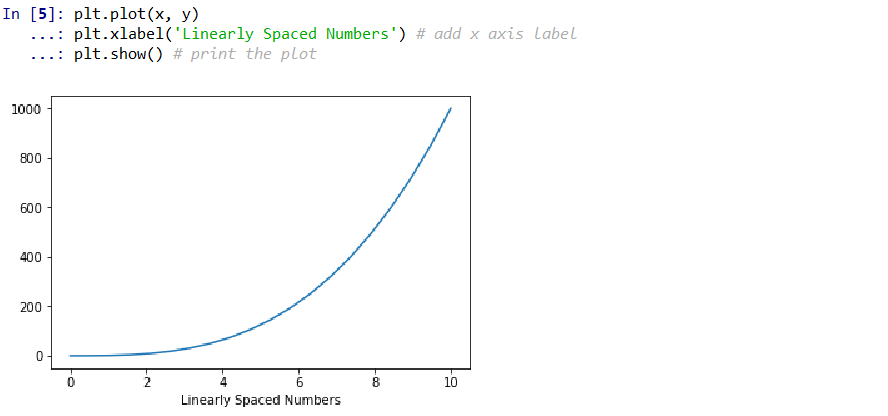
1. Create the plot using plt.plot(x, y) and print it to the console using plt.show().



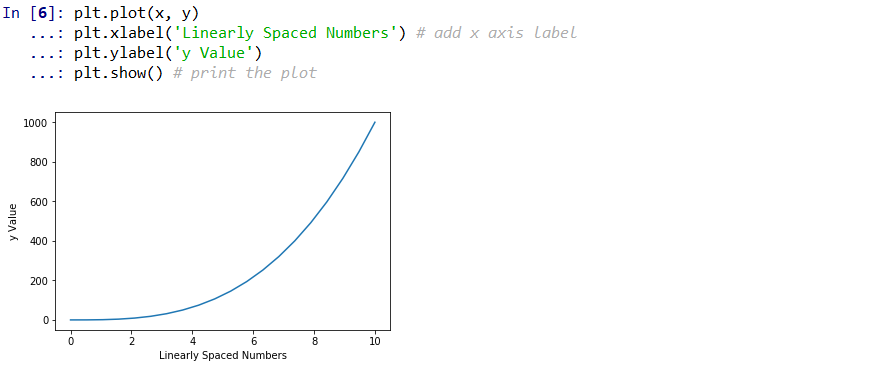
***Note***

Here, we have a plot which displays x on the x-axis and cubed values of x on the y-axis. Matplotlib makes it straightforward to add styling and labels to plots.

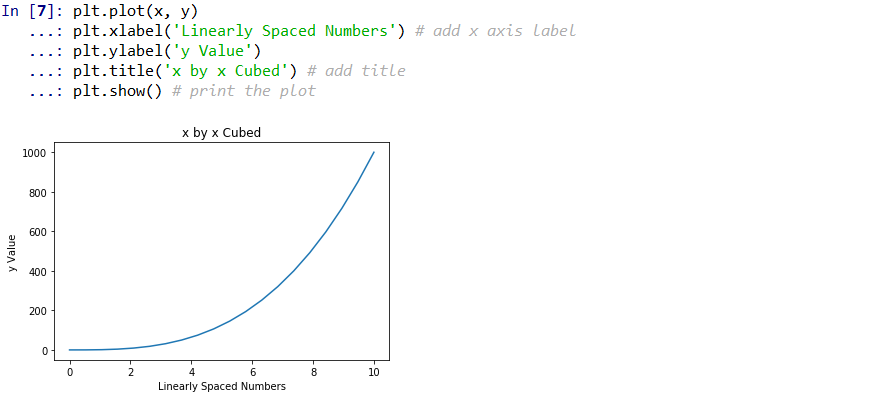
1. Add an x-axis label that reads ‘Linearly Spaced Numbers’ using plt.xlabel(‘Linearly Spaced Numbers’)



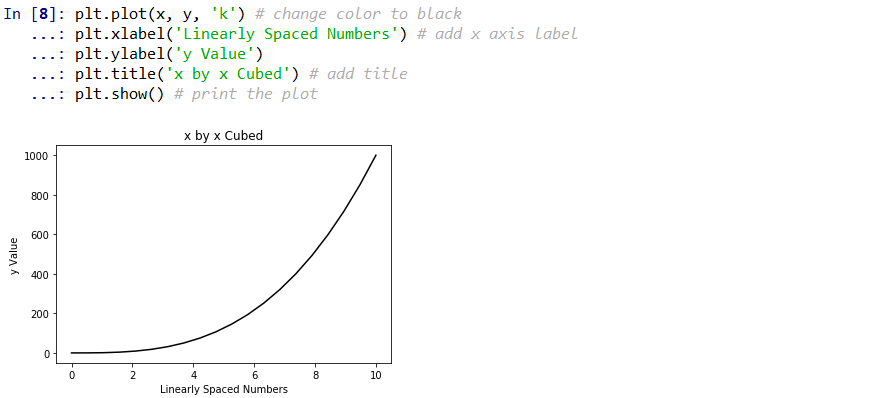
1. Add a y-axis label that reads ‘Cubed Values of x’ using plt.ylabel(‘Cubed Values of x’).



1. Add a title that reads ‘x by x cubed’ using plt.title(‘x by x Cubed’).



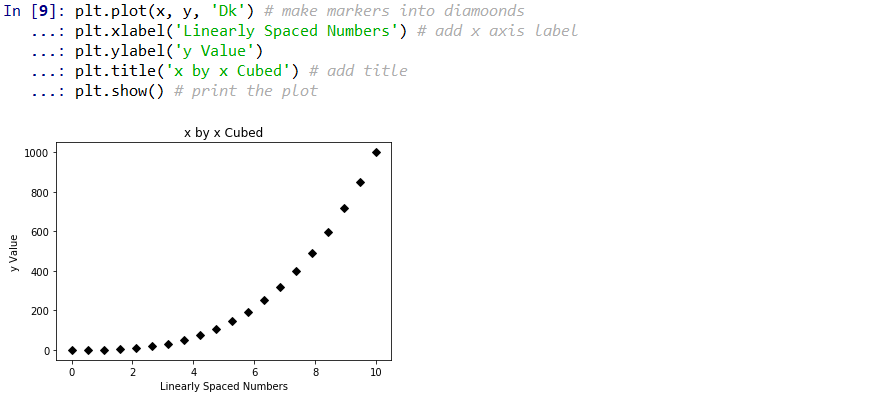
1. To change the line color specify a color argument in the plt.plot() function using plt.plot(x, y, ‘k’).



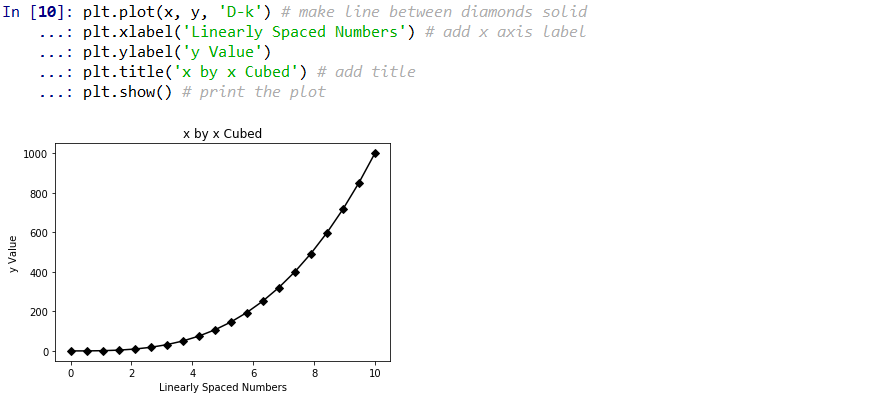
***Note***

We specified the color of the line to be black using the ‘k’ argument. For a full list of color arguments see <https://matplotlib.org/2.1.1/api/_as_gen/matplotlib.pyplot.plot.html>.

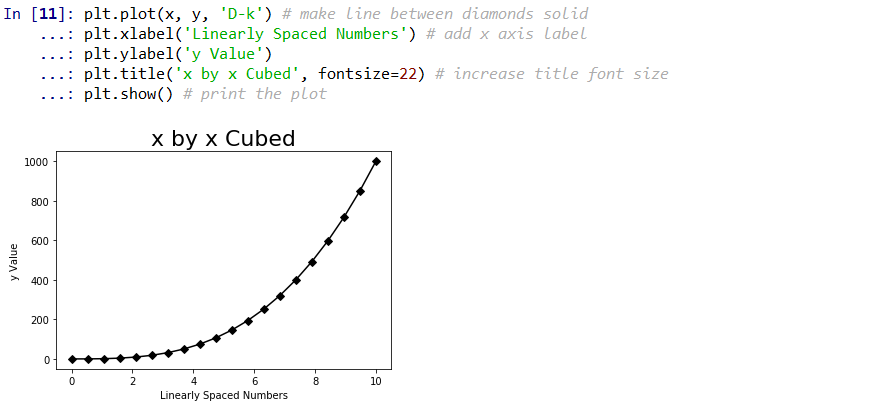
1. To make the line characters into a diamond use a character argument combined with the color character



1. We lost the solid line, so we will connect the diamonds with a solid line by placing ‘-’ between ‘D’ and ‘k’ using ‘D-k’.



1. Increase the font size of the title using the fontsize argument in the plt.title() function.



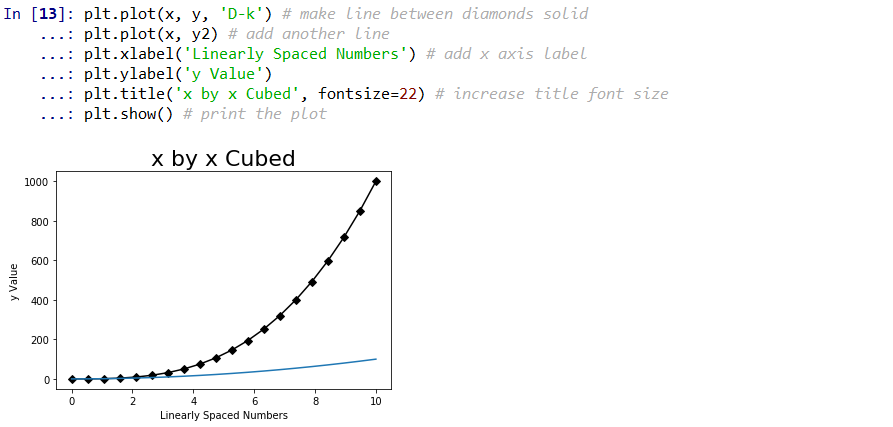
***Note***

The fontsize argument can be used in the xlabel() and xlabel() functions as well.

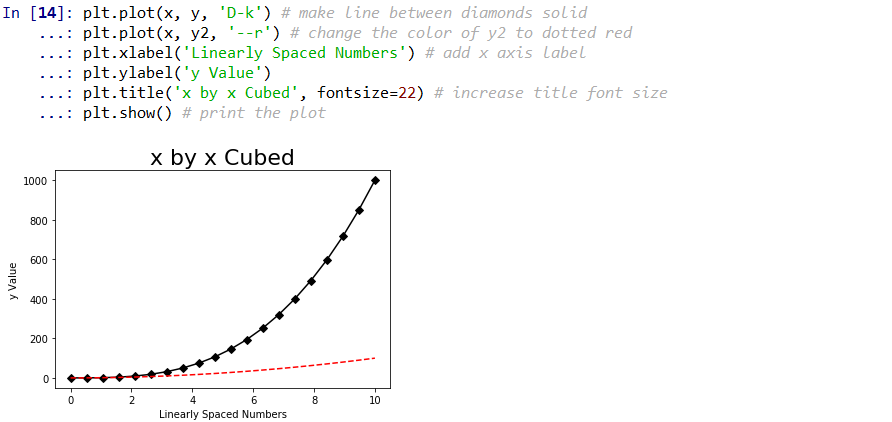
1. Add another line to the plot by simply specifying another plt.plot() function. For this exercise we will be displaying x-squared in another line. First, we must create another y object much as we did the first y object, but this time squaring x rather than cubing x.



1. Now, plot y2 in the same plot at y using plt.plot(x, y2).



1. Change the color of y2 to dotted red line using plt.plot(x, y2, ‘--r’) and adjust the title accordingly.



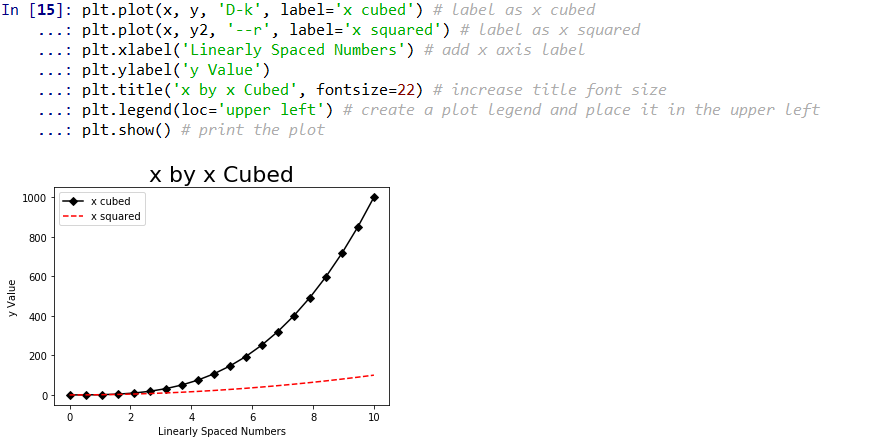
***Note***

At first glance, it is not intuitive as to what this plot is showing. We need to include a more descriptive title and a legend.

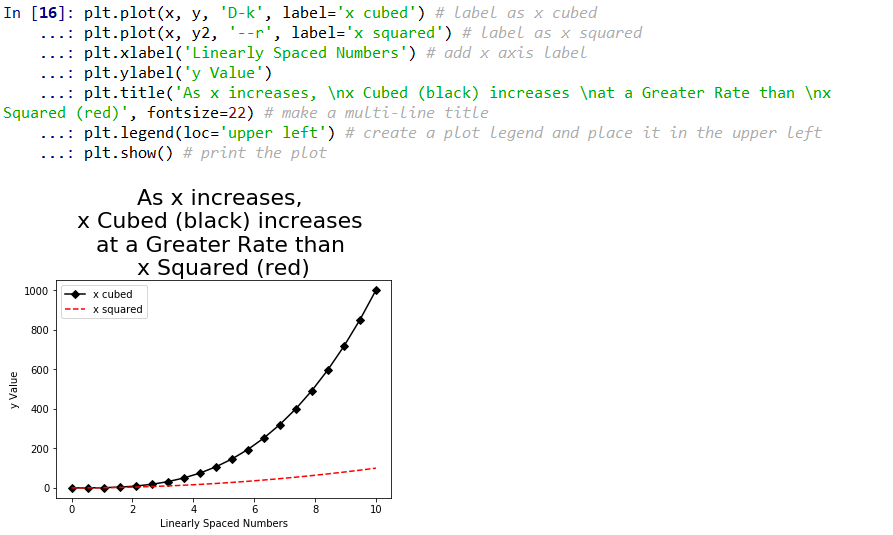
1. To create a legend we must first create labels for our lines using the label argument inside the plt.plot() functions. To label y as ‘x cubed’ use plt.plot(x, y, 'D-k', label='x cubed'). To label y2 as ‘x squared’ use plt.plot(x, y2, '--r', label='x squared'). Then, use plt.legend(loc=’upper left’) to specify the location for the legend.

***Note***

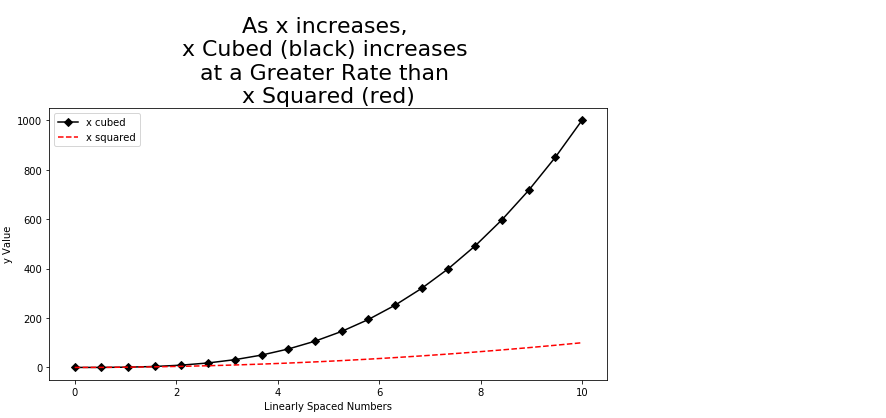
To view other arguments for legend locations see <https://matplotlib.org/api/_as_gen/matplotlib.pyplot.legend.html>.



1. It is helpful to the reader if the plot can be summarized in the title. In this plot, I want to communicate to the reader that, as x increases, x cubed increases at a greater rate than x squared. Sometimes, titles may get a little longer than we want. To break a line into new lines we use ‘\n’ at the beginning of a new line within our string. Thus, using plt.title('As x increases, \nx Cubed (black) increases \nat a Greater Rate than \nx Squared (red)', fontsize=22), we can create the title displayed below.



1. To change the dimensions of our plot we will need to add the function plt.figure(figsize=(10,5)) to the top of our plt functions. The figsize argument of 10 and 5 specify the width and height, respectively.



Exercise 2: Bar Plot

Now, that we have the basics of creating a line plot in matplotlib, we will explore how to create another very common type of plot, the bar plot. While line charts are most powerful when presenting trends, bar plots strive in displaying the differences between, or among, groups. Thus, in this example, we will be displaying Sales Revenue by Item type.

1. Import the matplotlib package using import matplotlib.pyplot as plt.



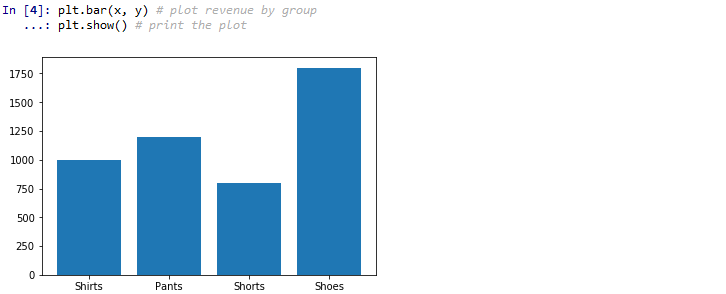
1. Create a list of item types and save it as x using x = ['Shirts', 'Pants','Shorts','Shoes'].



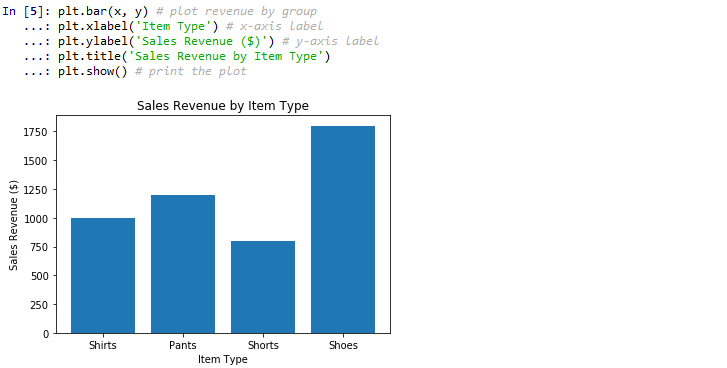
1. Create a list of Sales Revenue and save it as y using y = [1000, 1200, 800, 1800].



1. Create a bar plot using plt.bar(x, y) and print the plot using plt.show().



1. Add a title, an x label, and y label as we did in Exercise 1.



***Note***

While this title explains what the plot is displaying, it does not tell the reader what is going on in the plot. Next, we will go over how to make a programmatic title explaining to the reader what is happening in the plot.

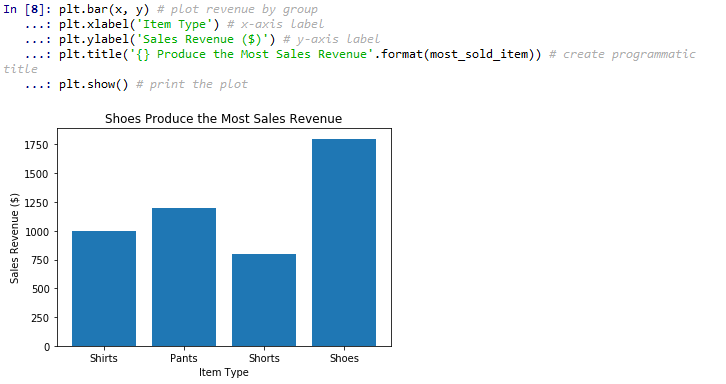
1. We are going to create a title that will change to the data which is plotted. For this example, it will read “Shoes Produce the Most Sales Revenue.” First, we will find the index of the maximum value in y and save it as the object index\_of\_max\_y using index\_of\_max\_y = y.index(max(y)).



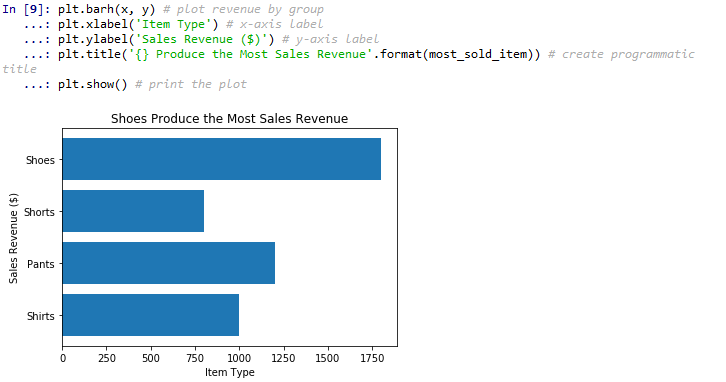
1. Save the item in list x with index equaling index\_of\_max\_y to the object most\_sold\_item using most\_sold\_item = x[index\_of\_max\_y].



1. Make the title programmatic using plt.title('{} Produce the Most Sales Revenue'.format(most\_sold\_item)). This code places the value for most\_sold\_item in place of {}.



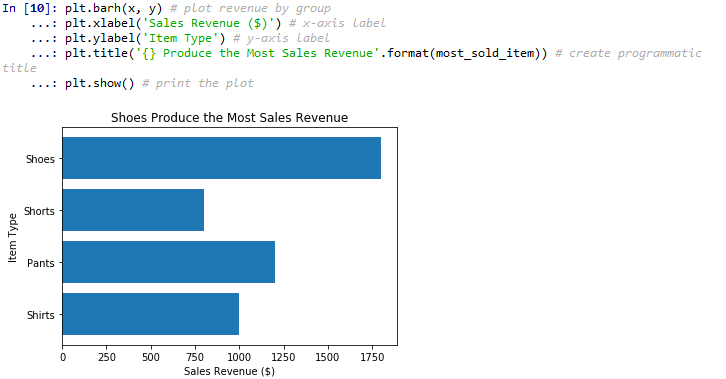
1. Turn the plot into a horizontal bar plot by preplacing plt.bar(x, y) with plt.barh(x, y).



***Note***

Remember, when a bar plot is transformed from vertical to horizontal that the x and y axes need to be switched.

1. Switch x and y labels from plt.xlabel('Item Type') and plt.ylabel('Sales Revenue ($)'), respectively, to plt.xlabel('Sales Revenue ($)') and plt.ylabel('Item Type'), respectively.



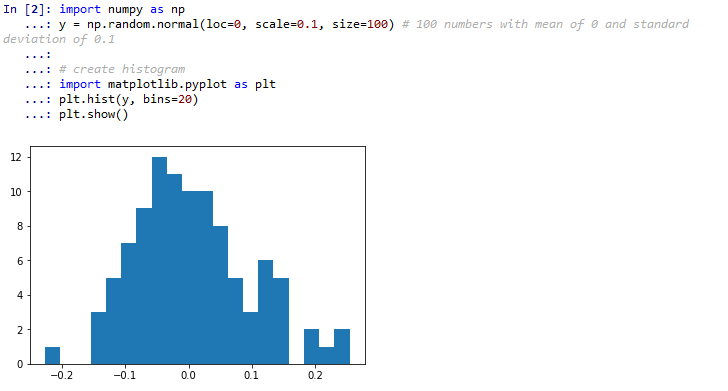
Exercise 3: Histogram

In statistics, it is essential to be aware of the distribution of continuous variables. To display the distribution we will use a histogram. Histograms display the frequency by the bin.

1. To demonstrate the creation of a histogram we must first generate an array of normally distributed values and save them as y. First, use import numpy as np. Then, using numpy’s random.normal function, use y = np.random.normal(loc=0, scale=0.1, size=100). This generates 100 normally distributed numbers with a mean of 0 and a standard deviation of 0.1.



1. With matplotlib imported, create the histogram using plt.hist(y, bins=20). The ‘bins’ argument determines the extent of the histogram’s granularity. Thus, the range of y is divided into 20 evenly spaced buckets, or bins.



***Note***

Since we covered styling arguments in great detail in Exercises 2 and 3, we will skip this portion and go right into making the title tell us what we need to know.

When we are looking at a histogram, we are determining whether or not the distribution is normal. There is a separate test for normality termed the Shapiro-Wilk test. The null hypothesis for the Shapiro-Wilk test is that the data is normally distributed. Thus, a p-value < 0.05 indicates a non-normal distribution while a p-value > 0.05 indicates a normal distribution.

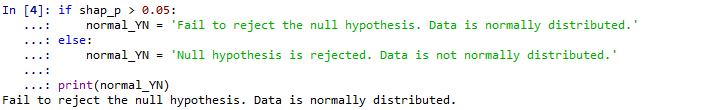
1. To return the W test statistic and the corresponding p-value from the Shapiro-Wilk test we must import the shapiro function using from scipy.stats import shapiro. The shapiro function returns a tuple containing two values. The first is the W statistic while the second is the p-value. Thus, we can use tuple unpacking to save the W statistic and the p-value into the objects shap\_w and shap\_p, respectively, using shap\_w, shap\_p = shapiro(y).



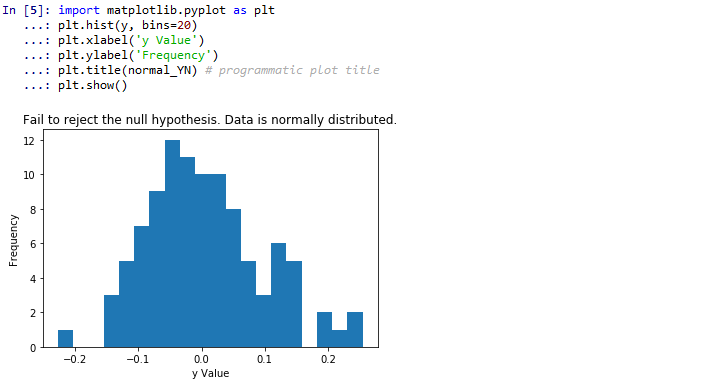
***Note***

The p-value is > 0.05, so, according to the Shapiro-Wilk test, y is normally distributed. We will set up some logic to make this decision programmatic.

1. We will use an if-else statement that assigns the string 'Fail to reject the null hypothesis. Data is normally distributed' to the object normal\_YN if shap\_p > 0.05 and assigns the string 'Null hypothesis is rejected. Data is not normally distributed' is shap\_p > 0.05.



1. Assign normal\_YN to our plot using plt.title(normal\_YN).



Object-Oriented Method Using Subplots

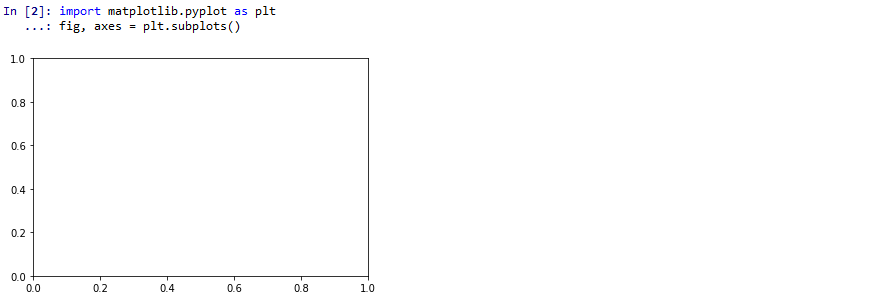
However, it is much more common to see the use of subplots in Matplotlib plots than to see the functional method. In the object-oriented method we create figure objects which act as an empty canvas and then we add a set of axes to it. We will demonstrate how this works by plotting the same x and y objects as we did in Exercise 1.

Exercise 4: Single Line Plot Using Subplots

1. Save x as an array ranging from 0 to 10 in 20 linearly spaced steps using x = np.linspace(0, 10, 20). Then, save y as x cubed using y = x\*\*3.



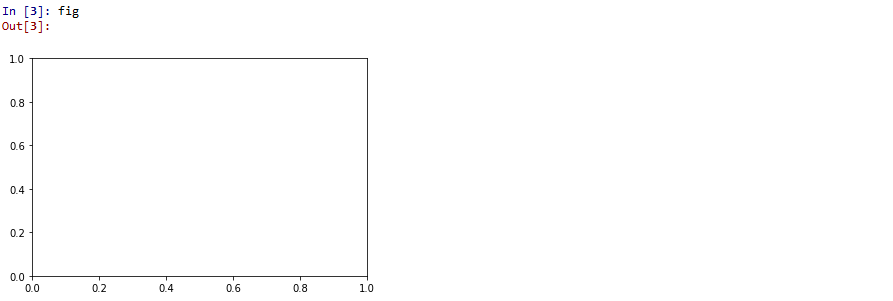
1. Import matplotlib using import matplotlib.pyplot as plt. Use tuple unpacking to create a figure and a set of axes using fig, axes = plt.subplots().



***Note***

By default, matplotlib creates a figure with one set of axes. To create multiple plots we will provide arguments inside the subplots() function.

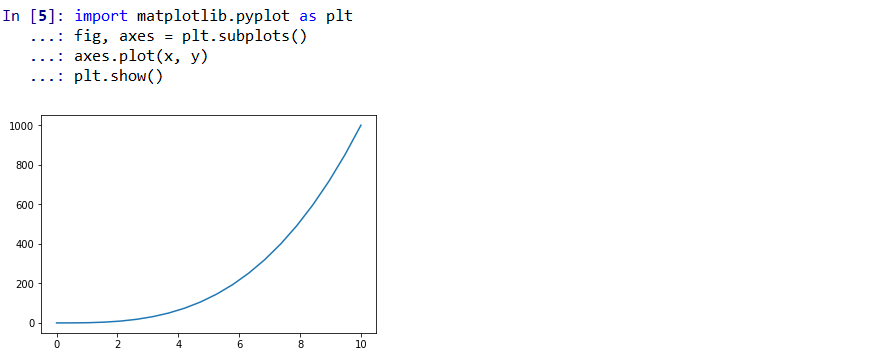
1. If we now just run fig we can see the axes that were created.



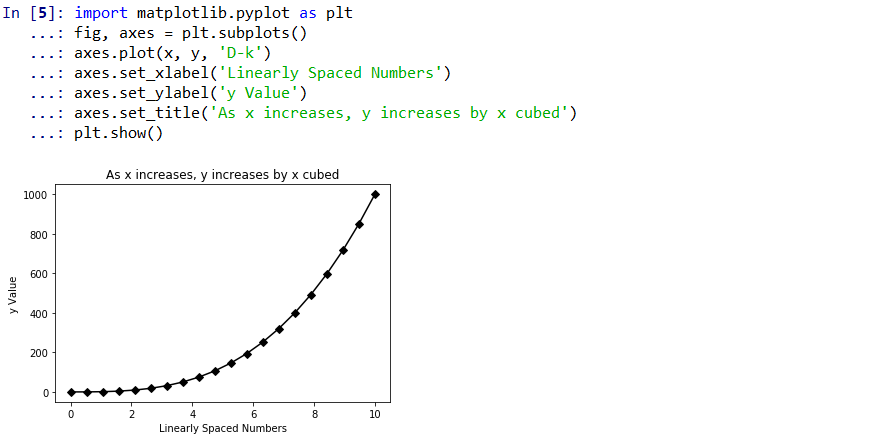
1. If we run axes we get:



1. To plot y by x (i.e., x squared), we use axes.plot(x, y) and show the plot using plt.show().



1. Style the plot much the same as in Exercise 1. Set the x-axis label to ‘Linearly Spaced Numbers’ using axes.set\_xlabel('Linearly Spaced Numbers'). To set the y-axis to ‘y Value’ use axes.set\_ylabel('y Value'). To set the title to ‘As x increases, y increases by x cubed’ use axes.set\_title('As x increases, y increases by x cubed'). Lastly, to change the line color and markers use the argument ‘D-k’ in the axes.plot() function.



***Note***

We created a plot very similar to the first plot in Exercise 1. However, power of subplots is the ability to specify multiple axes in the same figure.

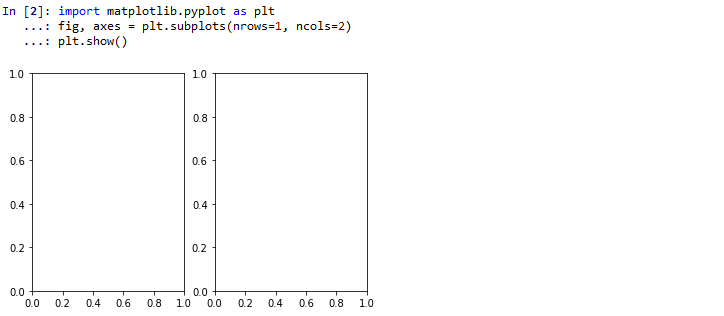
Exercise 4: Multiple Line Plots Using Subplots

In this exercise, we will plot the same lines as in Exercise 1, but we will plot them on two subplots.

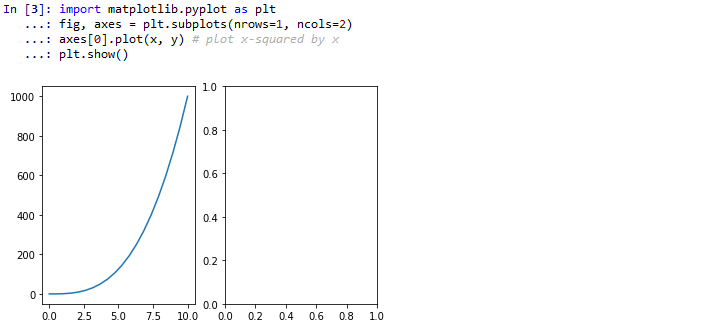
After, x, y, and y2 have been created (for more details, see Exercise 1):



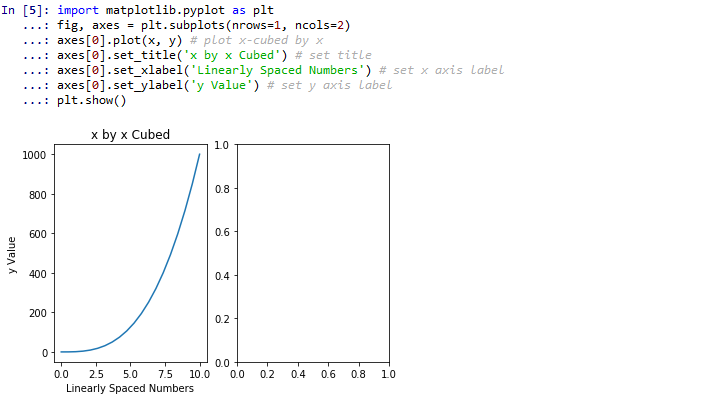
1. Import matplotlib using import matplotlib.pyplot as plt. Create a figure with 2 axes (I.e., plots) that are side-by-side (I.e., 1 row with 2 columns) using fig, axes = plt.subplots(nrows=1, ncols=2).



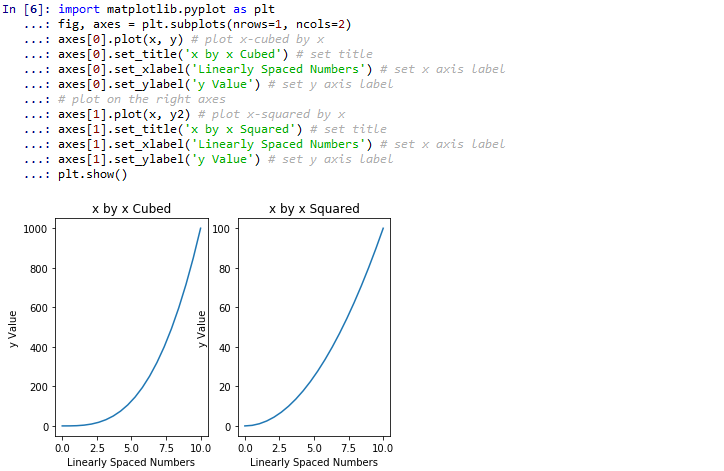
1. To access the plot on the left refer to it as axes[0]. To access the plot on the right refer to it as axes[1]. On the left axes, plot y by x using axes[0].plot(x, y).



1. Style it the same way we did in Exercise 3 by adding a title, x label, and y label.



1. On the right axes, plot y by x using axes[1].plot(x, y2) and use the same styling as in step 5.



***Note***

We have successfully created two subplots. However, it looks like the y-axis of the plot on the right is overlapping onto the left plot.

1. Fix the overlapping of the plots using plt.tight\_layout().

