Matplotlib Tutorial: Learn basics of Python’s powerful Plotting library

towardsdatascience.com/matplotlib-tutorial-learn-basics-of-pythons-powerful-plotting-library-b5d1b8f67596

Killol Govani 3 February 2019

Killol Govani
Feb 3 ★

Photo by rawpixel on Unsplash

What is Matplotlib

To make necessary statistical inferences, it becomes necessary to visualize your data and Matplotlib is one such solution for the Python users. It is a very powerful plotting library useful for those working with Python and NumPy. The most used module of Matplotib is Pyplot which provides an interface like MATLAB but instead, it uses Python and it is open source.

Installing Matplotlib

To install Matplotlib on your local machine, open Python command prompt and type following commands:

```
python -m pip install -U pip
python -m pip install -U matplotlib
```

I am assuming that you wish to foray into the world of data science and machine learning and hence I suggest you download the anaconda package distribution from here. It installs python, Jupyter notebook and other important python libraries including Matplotlib, Numpy, Pandas, scikit-learn. Anaconda supports Windows, MacOS and Linux. To quickly get started with Matplotlib without installing anything on your local machine, check out Google Colab. It provides the Jupyter Notebooks hosted on the cloud for free which are associated with your Google Drive account and it comes with all the important packages pre-installed. You can also run your code on GPU which helps in faster computation though we don't need GPU computation for this tutorial. To quickly get started with Google Colab, check out this amazing article.

General Concepts

A Matplotlib figure can be categorized into several parts as below:

Figure: It is a whole figure which may contain one or more than one axes (plots). You can think of a **Figure** as a canvas which contains plots.

Axes: It is what we generally think of as a plot. A **Figure** can contain many Axes. It contains two or three (in the case of 3D) **Axis** objects. Each Axes has a title, an x-label and a y-label.

Axis: They are the number line like objects and take care of generating the graph limits.

Artist: Everything which one can see on the figure is an artist like `Text` objects, `Line2D` objects, `collection` objects. Most Artists are tied to Axes.

Getting Started with Pyplot

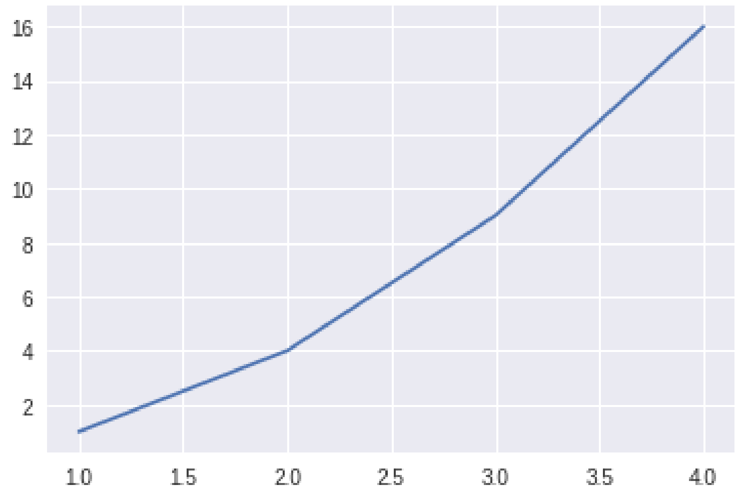
Pyplot is a module of Matplotlib which provides simple functions to add plot elements like lines, images, text, etc. to the current axes in the current figure.

Make a simple plot

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

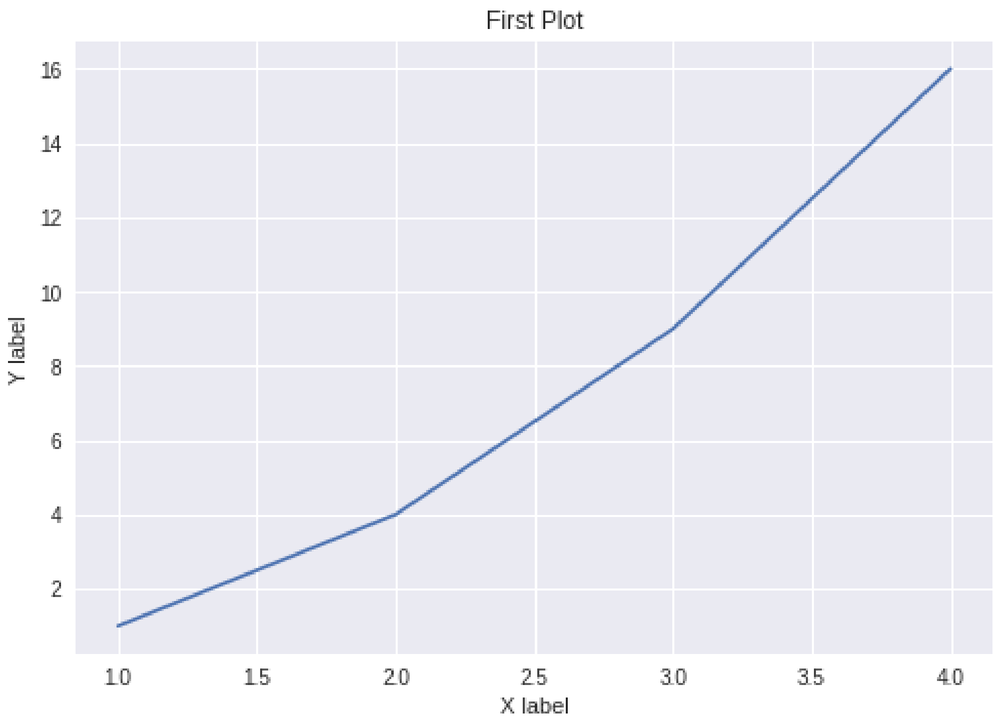
Here we import Matplotlib's Pyplot module and Numpy library as most of the data that we will be working with will be in the form of arrays only.

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

plt.plot([1,2,3,4],[1,4,9,16])
plt.show()
```



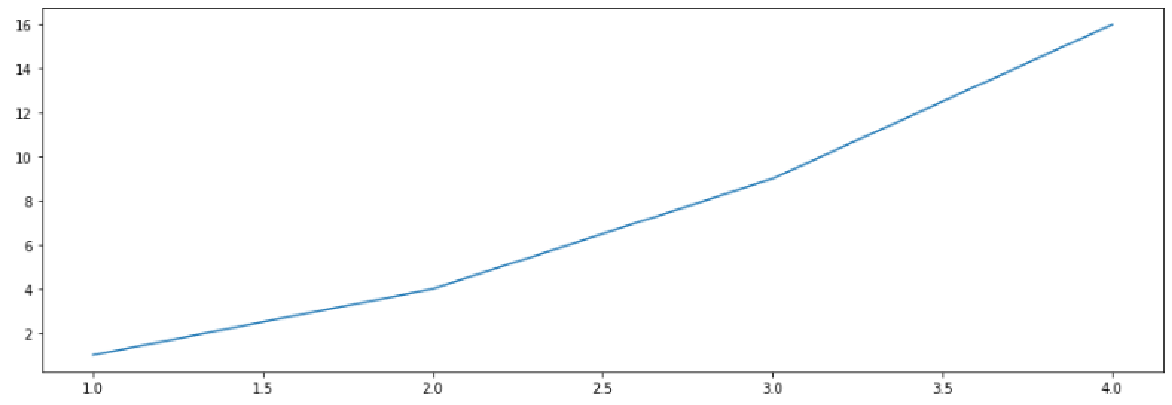
We pass two arrays as our input arguments to Pyplot's `plot()` method and use `show()` method to invoke the required plot. Here note that the first array appears on the x-axis and second array appears on the y-axis of the plot. Now that our first plot is ready, let us add the title, and name x-axis and y-axis using methods `title()`, `xlabel()` and `ylabel()` respectively.

```
plt.plot([1,2,3,4],[1,4,9,16])
plt.title("First Plot")
plt.xlabel("X label")
plt.ylabel("Y label")
plt.show()
```



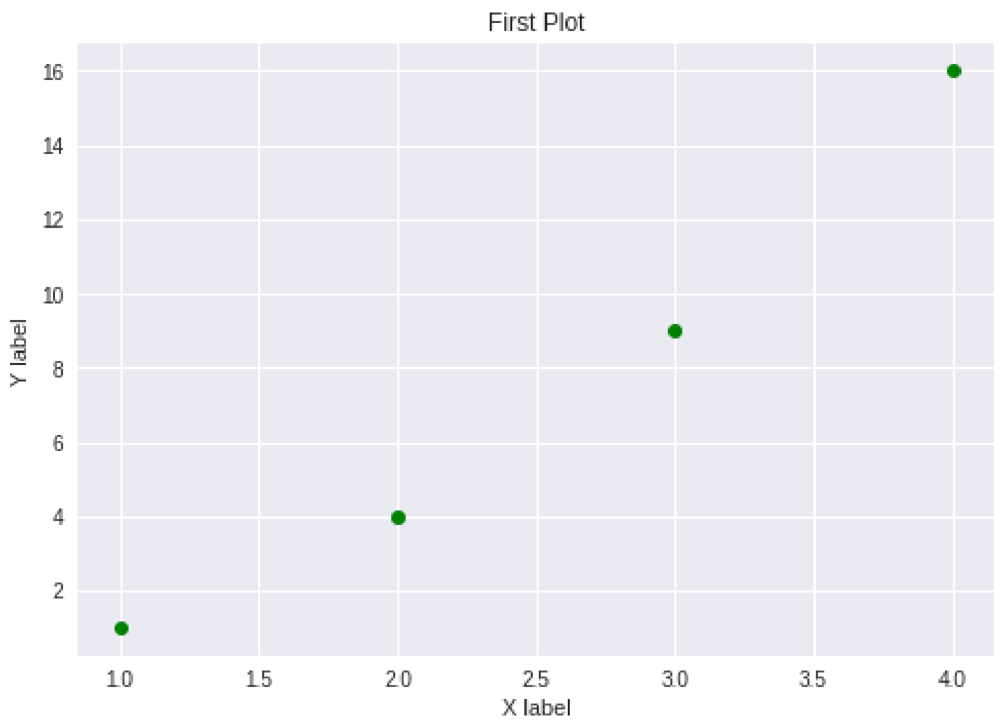
We can also specify the size of the figure using method `figure()` and passing the values as a tuple of the length of rows and columns to the argument `figsize`

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

plt.figure(figsize=(15,5))
plt.plot([1,2,3,4],[1,4,9,16])
plt.show()
```



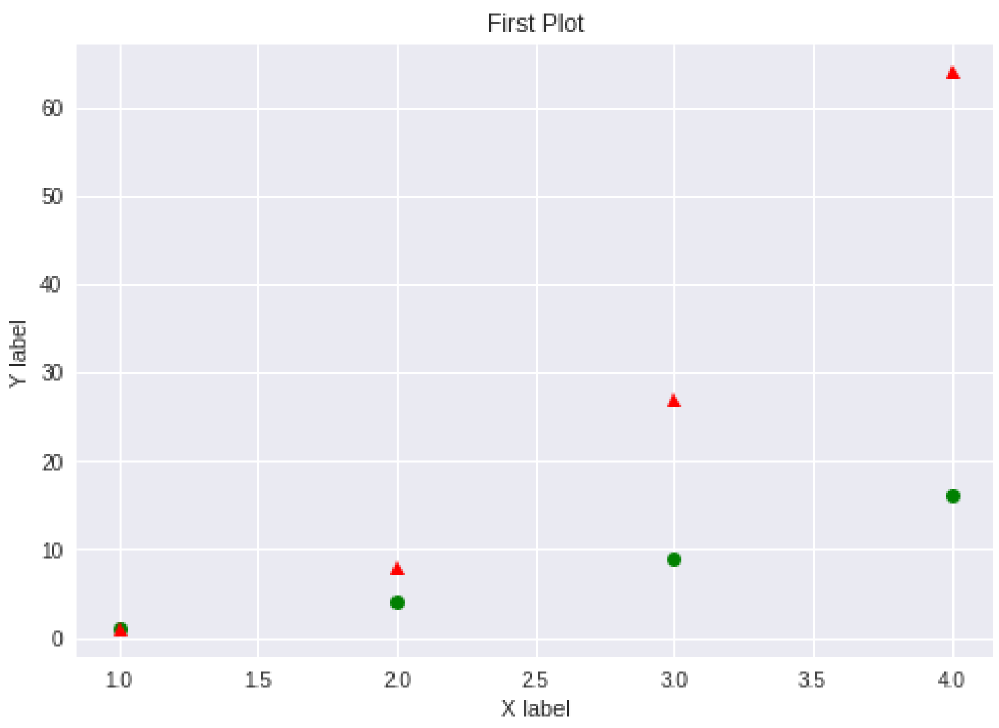
With every X and Y argument, you can also pass an optional third argument in the form of a string which indicates the colour and line type of the plot. The default format is **b-** which means a solid blue line. In the figure below we use **go** which means green circles. Likewise, we can make many such combinations to format our plot.

```
plt.plot([1,2,3,4],[1,4,9,16],"go")
plt.title("First Plot")
plt.xlabel("X label")
plt.ylabel("Y label")
plt.show()
```



We can also plot multiple sets of data by passing in multiple sets of arguments of X and Y axis in the `plot()` method as shown.

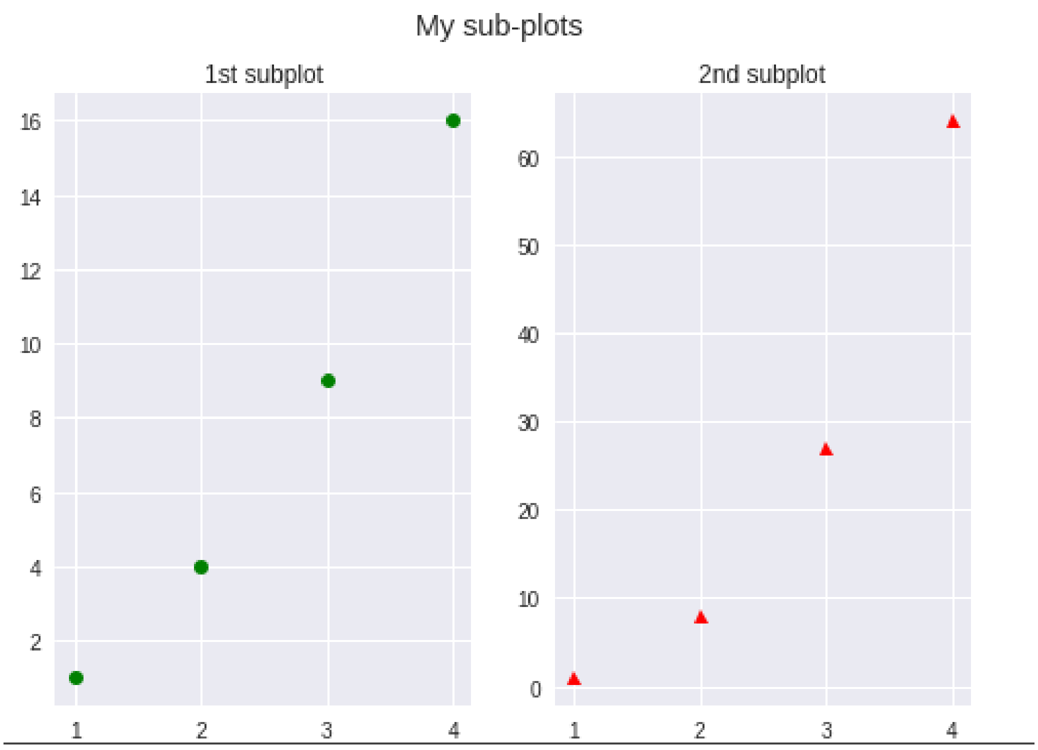
```
x = np.arange(1,5)
y = x**3
plt.plot([1,2,3,4],[1,4,9,16],'go', x, y,'r^')
plt.title("First Plot")
plt.xlabel("X label")
plt.ylabel("Y label")
plt.show()
```



Multiple plots in one figure:

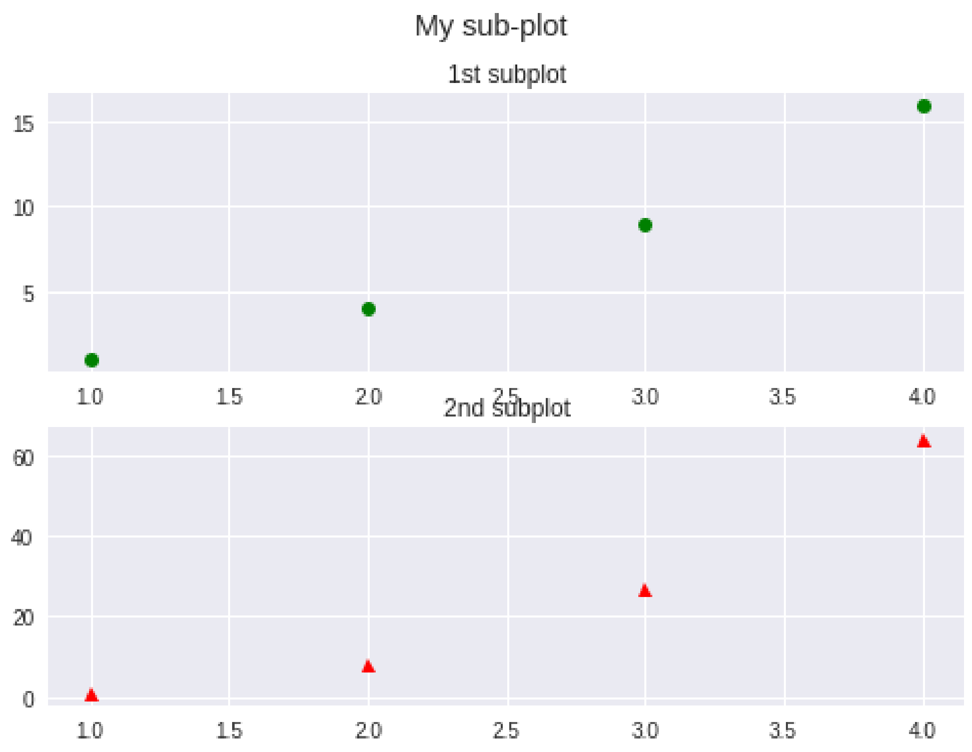
We can use `subplot()` method to add more than one plots in one figure. In the image below, we used this method to separate two graphs which we plotted on the same axes in the previous example. The `subplot()` method takes three arguments: they are `nrows`, `ncols` and `index`. They indicate the number of rows, number of columns and the index number of the sub-plot. For instance, in our example, we want to create two sub-plots in one figure such that it comes in one row and in two columns and hence we pass arguments `(1,2,1)` and `(1,2,2)` in the `subplot()` method. Note that we have separately used `title()`method for both the subplots. We use `suptitle()` method to make a centralized title for the figure.

```
plt.subplot(1,2,1)
plt.plot([1,2,3,4],[1,4,9,16],"go")
plt.title("1st subplot")

plt.subplot(1,2,2)
plt.plot(x,y,"r^")
plt.title("2nd subplot")

plt.suptitle("My sub-plots")
plt.show()
```



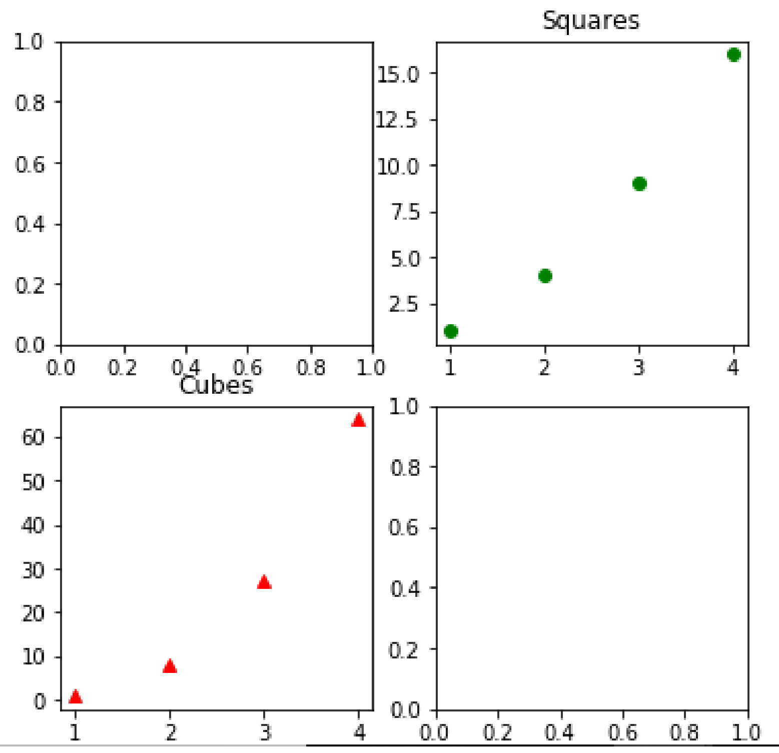
If we want our sub-plots in two rows and single column, we can pass arguments `(2,1,1)` and `(2,1,2)`

```
plt.subplot(2,1,1)
plt.plot([1,2,3,4],[1,4,9,16],'go')
plt.title('1st subplot')

plt.subplot(2,1,2)
plt.plot(x,y,'r^')
plt.title('2nd subplot')

plt.suptitle("My sub-plot")
plt.show()
```



The above way of creating subplots becomes a bit tedious when we want many subplots in our figure. A more convenient way is to use `subpltots()` method. Notice the difference of **'s'** in both the methods. This method takes two arguments `nrows` and `ncols` as number of rows and number of columns respectively. This method creates two objects: `figure` and `axes` which we store in variables fig and ax which can be used to change the figure and axes level attributes respectively. Note that these variable names are chosen arbitrarily.

```
x = np.arange(1,5)
y = x**3
fig, ax = plt.subplots(nrows=2,ncols=2,figsize=(6,6))
ax[0,1].plot([1,2,3,4],[1,4,9,16],'go')
ax[1,0].plot(x,y,'r^')
ax[0,1].set_title("Squares")
ax[1,0].set_title("Cubes")
plt.show()
```

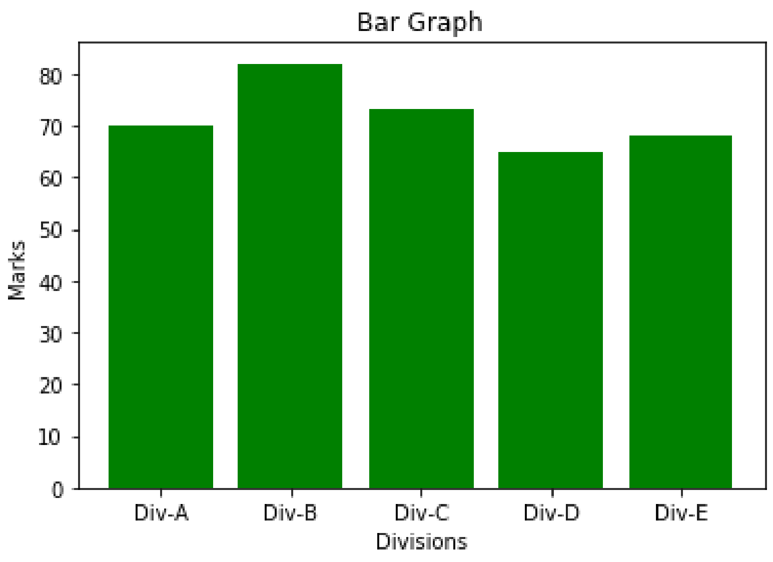


Creating different types of graphs with Pyplot

1) Bar Graphs

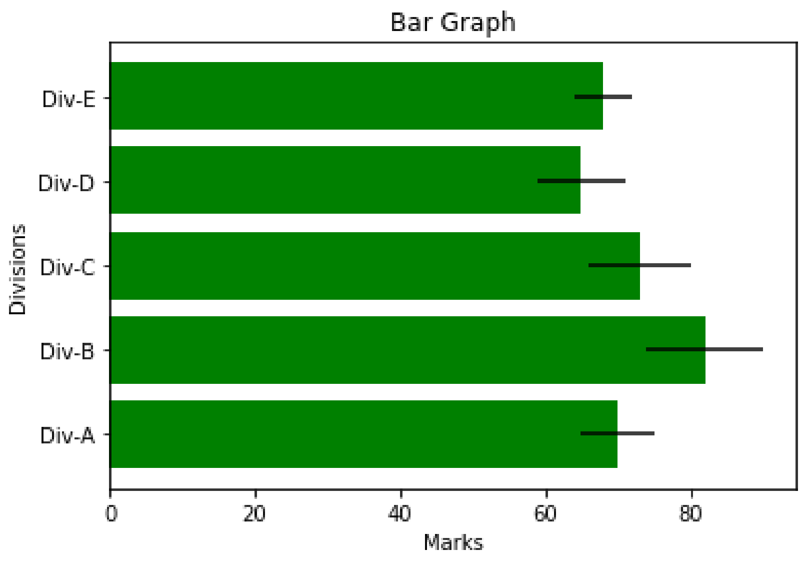
Bar graphs are one of the most common types of graphs and are used to show data associated with the categorical variables. Pyplot provides a method `bar()` to make bar graphs which take arguments: categorical variables, their values and color (if you want to specify any).

```
divisions = ["Div-A", "Div-B", "Div-C", "Div-D", "Div-E"]
division_average_marks = [70, 82, 73, 65, 68]

plt.bar(divisions, division_average_marks, color='green')
plt.title("Bar Graph")
plt.xlabel("Divisions")
plt.ylabel("Marks")
plt.show()
```



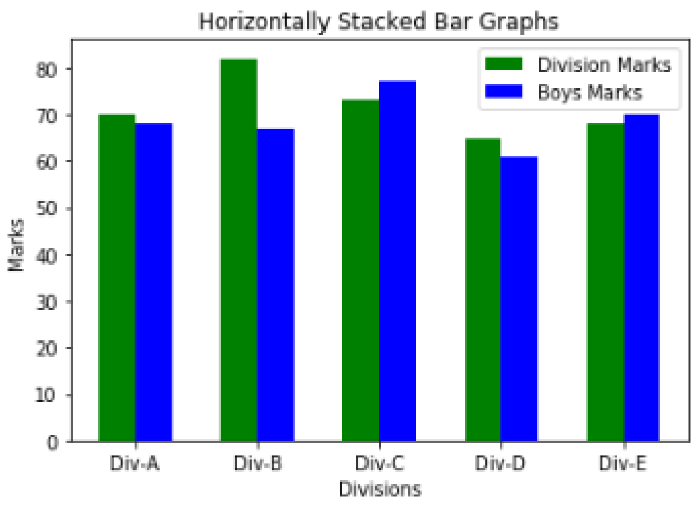
To make horizontal bar graphs use method `barh()` Also we can pass an argument (with its value) `xerr` or `yerr` (in case of the above vertical bar graphs) to depict the variance in our data as follows:

```
divisions = ["Div-A", "Div-B", "Div-C", "Div-D", "Div-E"]
division_average_marks = [70, 82, 73, 65, 68]
variance = [5,8,7,6,4]

plt.barh(divisions, division_average_marks, xerr=variance, color='green')
plt.title("Bar Graph")
plt.xlabel("Marks")
plt.ylabel("Divisions")
plt.show()
```



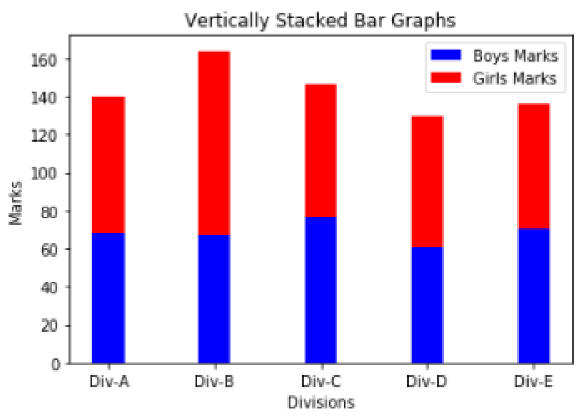
To create horizontally stacked bar graphs we use the `bar()` method twice and pass the arguments where we mention the index and width of our bar graphs in order to horizontally stack them together. Also, notice the use of two other methods `legend()` which is used to show the legend of the graph and `xticks()` to label our x-axis based on the position of our bars.

```
divisions = ["Div-A", "Div-B", "Div-C", "Div-D", "Div-E"]
division_average_marks = [70, 82, 73, 65, 68]
boys_average_marks = [68, 67, 77, 61, 70]

index = np.arange(5)
width = 0.30

plt.bar(index, division_average_marks, width, color='green',label='Division Marks')
plt.bar(index+width, boys_average_marks, width, color='blue',label='Boys Marks')
plt.title("Horizontally Stacked Bar Graphs")

plt.ylabel("Marks")
plt.xlabel("Divisions")
plt.xticks(index+ width/2, divisions)

plt.legend(loc='best')
plt.show()
```



Similarly, to vertically stack the bar graphs together, we can use an argument `bottom` and mention the bar graph which we want to stack below as its value.

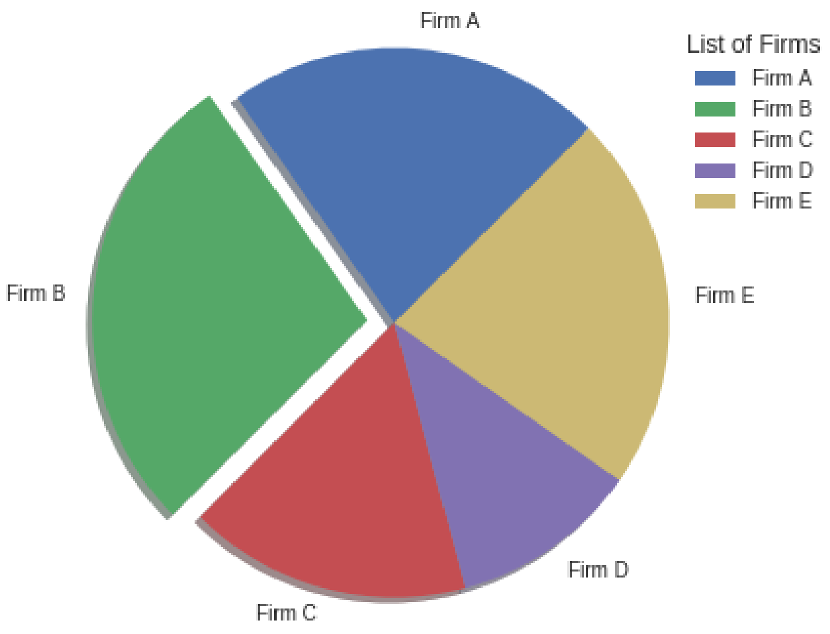
```
divisions = ["Div-A", "Div-B", "Div-C", "Div-D", "Div-E"]
boys_average_marks = [68, 67, 77, 61, 70]
girls_average_marks = [72, 97, 69, 69, 66]

index = np.arange(5)
width = 0.30

plt.bar(index, boys_average_marks, width, color="blue", label="Boys Marks")
plt.bar(index, girls_average_marks, width, color="red", label="Girls Marks", bottom=boys_average_marks)

plt.title("Vertically Stacked Bar Graphs")
plt.xlabel("Divisions")
plt.ylabel("Marks")
plt.xticks(index, divisions)

plt.legend(loc='best')
plt.show()
```



2) Pie Charts

One more basic type of chart is a Pie chart which can be made using the method `pie()` We can also pass in arguments to customize our Pie chart to show shadow, explode a part of it, tilt it at an angle as follows:

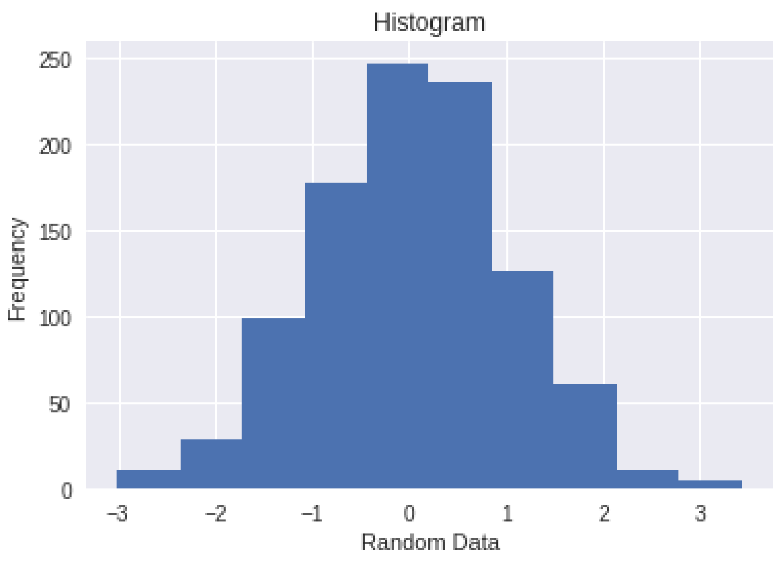
```
firms = ["Firm A", "Firm B", "Firm C", "Firm D","Firm E"]
market_share = [20, 25, 15, 10, 20]
Explode = [0,0.1,0,0,0]
plt.pie(market_share,explode=Explode,labels=firms,shadow=True,startangle=45)
plt.axis('equal')
plt.legend(title="List of Firms")
plt.show()
```



3) Histogram

Histograms are a very common type of plots when we are looking at data like height and weight, stock prices, waiting time for a customer, etc which are continuous in nature. Histogram’s data is plotted within a range against its frequency. Histograms are very commonly occurring graphs in probability and statistics and form the basis for various distributions like the normal -distribution, t-distribution, etc. In the following example, we generate a random continuous data of 1000 entries and plot it against its frequency with the data divided into 10 equal strata. We have used NumPy’s `random.randn()` method which generates data with the properties of a standard normal distribution i.e. mean = 0 and standard deviation = 1, and hence the histogram looks like a normal distribution curve.

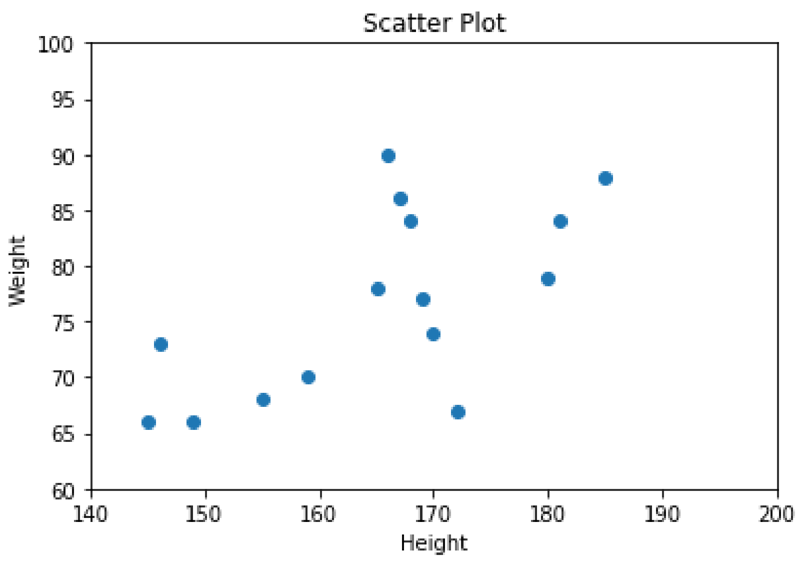
```
x = np.random.randn(1000)

plt.title("Histogram")
plt.xlabel("Random Data")
plt.ylabel("Frequency")
plt.hist(x,10)
plt.show()
```



4)Scatter Plots and 3-D plotting

Scatter plots are widely used graphs, especially they come in handy in visualizing a problem of regression. In the following example, we feed in arbitrarily created data of height and weight and plot them against each other. We used `xlim()` and `ylim()` methods to set the limits of X-axis and Y-axis respectively.

```
height = np.array([167,170,149,165,155,180,166,146,
                   159,185,145,168,172,181,169])
weight = np.array([86,74,66,78,68,79,90,73,
                   70,88,66,84,67,84,77])
plt.xlim(140,200)
plt.ylim(60,100)
plt.scatter(height,weight)
plt.title("Scatter Plot")
plt.xlabel("Height")
plt.ylabel("Weight")
plt.show()
```

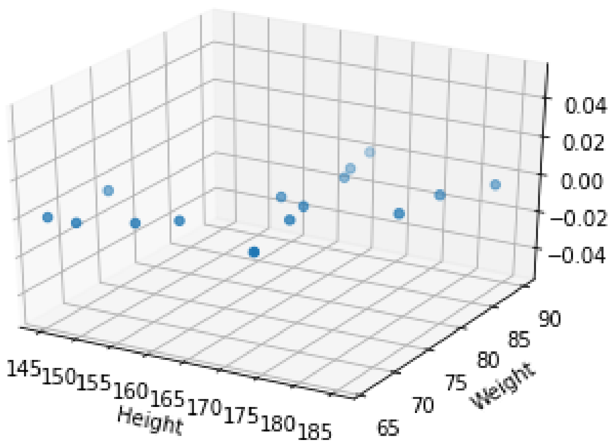


The above scatter can also be visualized in three dimensions. To use this functionality, we first import the module `mplot3d` as follows:

```
from mpl_toolkits import mplot3d
```

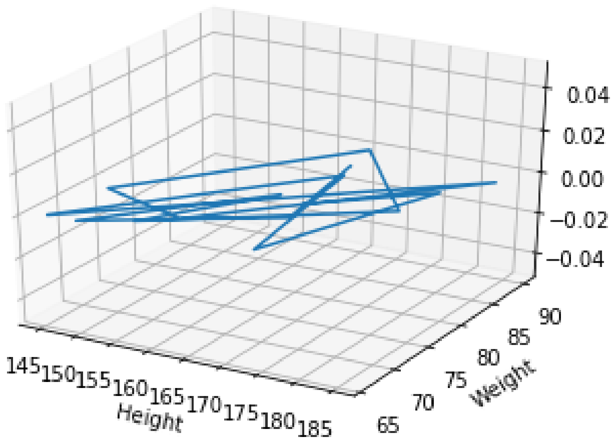
Once the module is imported, a three-dimensional axes is created by passing the keyword `projection='3d'` to the `axes()` method of Pyplot module. Once the object instance is created, we pass our arguments height and weight to `scatter3D()` method.

```
ax = plt.axes(projection='3d')
ax.scatter3D(height,weight)
ax.set_xlabel("Height")
ax.set_ylabel("Weight")
plt.show()
```



We can also create 3-D graphs of other types like line graph, surface, wireframes, contours, etc. The above example in the form of a simple line graph is as follows: Here instead of `scatter3D()` we use method `plot3D()`

```
ax = plt.axes(projection='3d')
ax.plot3D(height,weight)
ax.set_xlabel("Height")
ax.set_ylabel("Weight")
plt.show()
```



Summary

Hope this article was useful to you. If you liked this article please express your appreciation. Before we end the article here is the list of all the methods as they appeared.

- plot(x-axis values, y-axis values) — plots a simple line graph with x-axis values against y-axis values
- show() — displays the graph
- title(“string”) — set the title of the plot as specified by the string
- xlabel(“string”) — set the label for x-axis as specified by the string
- ylabel(“string”) — set the label for y-axis as specified by the string
- figure() — used to control a figure level attributes
- subplot(nrows, ncols, index) — Add a subplot to the current figure
- suptitle(“string”) — It adds a common title to the figure specified by the string
- subplots(nrows, ncols, figsize) — a convenient way to create subplots, in a single call. It returns a tuple of a figure and number of axes.
- set_title(“string”) — an axes level method used to set the title of subplots in a figure
- bar(categorical variables, values, color) — used to create vertical bar graphs
- barh(categorical variables, values, color) — used to create horizontal bar graphs
- legend(loc) — used to make legend of the graph
- xticks(index, categorical variables) — Get or set the current tick locations and labels of the x-axis
- pie(value, categorical variables) — used to create a pie chart
- hist(values, number of bins) — used to create a histogram
- xlim(start value, end value) — used to set the limit of values of the x-axis
- ylim(start value, end value) — used to set the limit of values of the y-axis

- scatter(x-axis values, y-axis values) — plots a scatter plot with x-axis values against y-axis values
- axes() — adds an axes to the current figure
- set_xlabel(“string”) — axes level method used to set the x-label of the plot specified as a string
- set_ylabel(“string”) — axes level method used to set the y-label of the plot specified as a string
- scatter3D(x-axis values, y-axis values) — plots a three-dimensional scatter plot with x-axis values against y-axis values
- plot3D(x-axis values, y-axis values) — plots a three-dimensional line graph with x-axis values against y-axis values