MatPlotLib:

Matplotlib is a plotting library for Python.

It is used along with NumPy to provide an environment that is an effective open source alternative for MatLab.

Matplotlib module was first written by John D. Hunter

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],[10,20,30,40])
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],[10,20,30,40])
plt.title("Sales Plot")
plt.xlabel("Month")
plt.ylabel("Sales")
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

```
plt.figure(figsize=(15,5))
plt.plot([1,2,3,4],[10,20,30,40])
plt.show()
```

Instead of the linear graph, the values can be displayed discretely by adding a format string to the plot() function.

-	Solid line style
--	Dashed line style
-.	Dash-dot line style
:	Dotted line style
.	Point marker
,	Pixel marker
o	Circle marker
v	Triangle_down marker

^	Triangle_up marker
<	Triangle_left marker
>	Triangle_right marker
1	Tri_down marker
2	Tri_up marker
3	Tri_left marker
4	Tri_right marker
s	Square marker
p	Pentagon marker
*	Star marker
h	Hexagon1 marker
H	Hexagon2 marker
+	Plus marker
x	X marker
D	Diamond marker
d	Thin_diamond marker
|	Vline marker
_	Hline marker

The following color abbreviations are also defined.

Character	Color
b	Blue
g	Green

r	Red
c	Cyan
m	Magenta
y	Yellow
k	Black
w	White

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.

Likewise, we can make many such combinations to format our plot.

```
plt.plot([1,2,3,4],[10,20,30,40],"go")
plt.title("Sales Plot")
plt.xlabel("Month")
plt.ylabel("Sales")
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

```
x = np.arange(1,5)
y = x**5
```

```
plt.plot([1,2,3,4],[10,20,30,40],"go",x,y,"r^")
plt.title("Sales Plot")
plt.xlabel("Month")
plt.ylabel("Sales")
plt.show()
```

Sine Wave Plot

==========

```
# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)
plt.title("sine wave form")
# Plot the points using matplotlib
plt.plot(x, y)
plt.show()
```

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).

```
month = ["Jan","Feb","Mar","Apr","May"]
sales = [50,60,55,75,65]
plt.bar(month,sales,color="green")
plt.title("Bar Graph")
plt.xlabel("Months")
plt.ylabel("Sales")
plt.show()
```

To make horizontal bar graphs use method barh()

```
plt.barh(month,sales,color="green")
plt.title("Bar Graph")
plt.xlabel("Months")
plt.ylabel("Sales")
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.

```
month = ["Jan","Feb","Mar","Apr","May"]
HYDsales = [50,60,55,75,65]
BNGsales = [55,50,70,60,80]
```

```
index = np.arange(5)
width = 0.30
plt.bar(index,HYDsales,width,color="green",label="Hyd Sales")
plt.bar(index+width,BNGsales,width,color="blue",label="Bang Sales")
plt.title("Horizontally Stacked Bar Graphs")
plt.xlabel("Months")
plt.ylabel("Sales")
plt.xticks(index+ width/2, month)
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.

```
plt.bar(index,HYDsales,width,color="blue",label="Hyd Sales")
plt.bar(index,BNGsales,width,color="red",label="Bang Sales",bottom=HYDsales)
plt.title("Vertically Stacked Bar Graphs")
plt.xlabel("Months")
plt.ylabel("Sales")
plt.xticks(index, month)
plt.legend(loc="best")
plt.show()
```

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:

```
month = ["Jan","Feb","Mar","Apr","May"]
sales = [50,60,55,75,65]
e = [0,0.1,0,0,0]
plt.pie(sales,explode=e,labels=month,shadow=True,startangle=45)
plt.axis("equal")
plt.legend(title="Sales Graph")
plt.show()
```


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()
```


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([130,135,120,145,165,175,180,145,150,160,140,170,165,135,1
55])

weight = np.array([60,70,65,75,80,90,95,85,70,78,88,72,68,77,88])

plt.xlim(100,200)

plt.ylim(50,100)

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

```
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()
```


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],[10,20,30,40],"go")
plt.title("1st subplot")
plt.subplot(1,2,2)
x = np.arange(1,5)
y = x**5
plt.plot(x,y,"r^")
plt.title("2nd subplot")
plt.suptitle("Sales 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],[10,20,30,40],"go")
plt.title("1st subplot")
plt.subplot(2,1,2)
```

```
x = np.arange(1,5)
y = x**5
plt.plot(x,y,"r^")
plt.title("2nd subplot")
plt.suptitle("Sales Plots")
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**5
fig, ax = plt.subplots(nrows=2,ncols=2,figsize=(6,6))
ax[0,1].plot([1,2,3,4],[10,20,30,40],"go")
ax[1,0].plot(x,y,'r^')
ax[0,1].set_title("Squares")
ax[1,0].set_title("Cubes")
```

```
plt.show()
```

Summary:

========

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