Matplotlib in Python:

A wide variety of tools exist for visualization. “matplotlib” is a Python library that helps in visualizing and analyzing the data for better understanding. Which is widely used for producing simple bar charts, line charts, and scatterplots by using matplotlib.pyplot module. The pyplot maintains an internal state in which you build up a visualization step by step. For saving the graphs use savefig() and to display use show() functions.

Most of the *matplotlib* utilities lies under the *pyplot* submodule, and are usually imported under the *plt* alias:

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

Functions in *matplotlib*

plot() - to create the plot

show() – to display the created plots

xlabel() - to label the x-axis

ylabel() – to label the y-axis

title() - to give the title to the graph

xticks() - to decide how the markings are to be made on the x-axis

yticks() - to decide how the markings are to be made on the y-axis

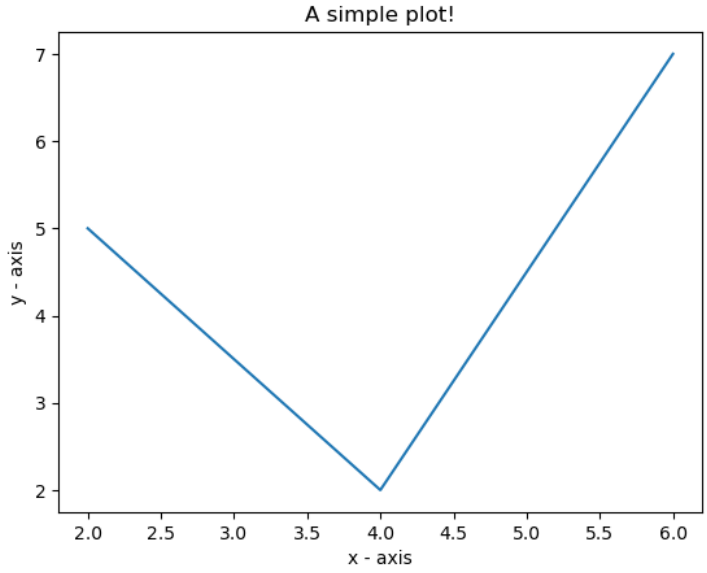
Creating a Simple Plot

Program 1 :

```
# importing the required module
import matplotlib.pyplot as plt

# x axis values
x = [2,4,6]
# corresponding y axis values
y = [5,2,7]

# plotting the points
plt.plot(x, y)

# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')

# giving a title of the graph
plt.title('A simple plot!')
```

```
# function to show the plot
plt.show()
```

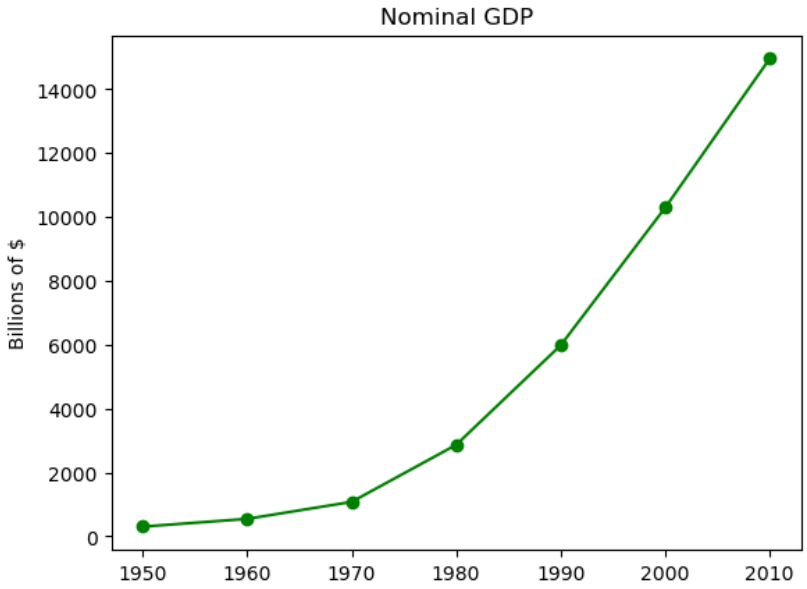
Output:



Program 2 :

```
import matplotlib.pyplot as plt
years = [1950, 1960, 1970, 1980, 1990, 2000, 2010]
gdp = [300.2, 543.3, 1075.9, 2862.5, 5979.6, 10289.7, 14958.3]
# create a line chart, years on x-axis, gdp on y-axis
plt.plot(years, gdp, color='green', marker='o',
                        linestyle='solid')
# add a title
plt.title("Nominal GDP")
# add a label to the y-axis
plt.ylabel("Billions of $")
plt.show()
```

Output:



Types of Plots:

Most commonly used Plots come under Matplotlib are:

1. Bar Chart
2. Histogram
3. Frequency polygon
4. Quartiles
5. Box-plot
6. Scatter Plot
7. Heat maps

1) Bar Chart:

A bar chart is a graph that represents the category of data with rectangular bars. A bar chart describes the comparisons between the discrete categories. One of the axis ofthe plot represents the specific categories being compared, while the other axis represents the measured values corresponding to those categories. *bar()* function used to plot Bar chart.

Syntax:

```
bar(x, height, width, bottom, align)
```

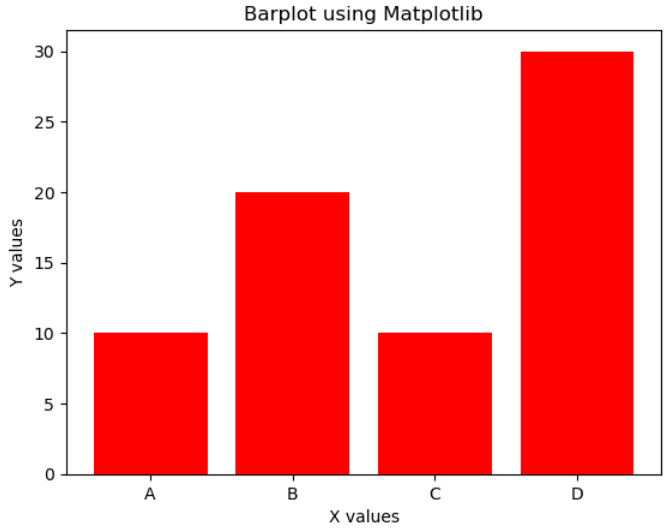
The function creates a bar plot bounded with a rectangle depending on the given parameters.

x	x coordinates of the bars.
height	scalar or sequence of scalars representing height(s) of the bars.
width	Optional parameter - the width(s) of the bars default 0.8
bottom	Optional parameter - the y coordinate(s) of the bars default None.
align	Optional parameter - {‘center’, ‘edge’}, default ‘center’

Program 1:

```
import matplotlib.pyplot as plt
 x = ['A', 'B', 'C', 'D']
y1 = [10, 20, 10, 30]
plt.bar(x, y1, color='r')
plt.xlabel("X values")
plt.ylabel("Y values")
plt.title("Barplot using Matplotlib")
plt.show()
```

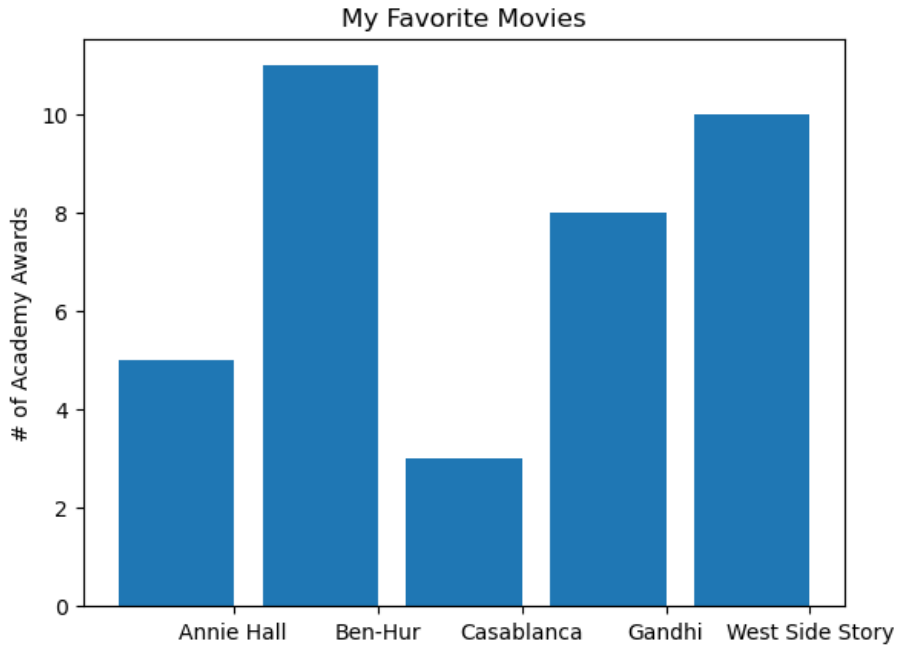
Output:



Program 2:

```
import matplotlib.pyplot as plt
movies = ["Annie Hall", "Ben-Hur", "Casablanca", "Gandhi",
"West Side Story"]
num_oscars = [5, 11, 3, 8, 10]
# bars are by default width 0.8, so we'll add 0.1 to the
#left coordinates so that each bar is centered
xs = [i + 0.1 for i, _ in enumerate(movies)]
# plot bars with left x-coordinates [xs],
# heights [num_oscars]
plt.bar(xs, num_oscars)
plt.ylabel("# of Academy Awards")
plt.title("My Favorite Movies")
# label x-axis with movie names at bar centers
plt.xticks([i + 0.5 for i, _ in enumerate(movies)], movies)
plt.show()
```

Output:



2) Histogram :

- A histogram is an accurate representation of the distribution of numerical data. It is a kind of bar graph. A **histogram** plot shows the frequency distribution (shape) of a set of continuous data, each bar in histogram represents the height of the number of values present in that range. In Python **hist()** function is used to plot a histogram.

Syntax:

```
hist (x, bins, range, align, color, label)
```

x : This parameter is the sequence of data.

bins : optional parameter and it contains the integer or sequence or string.

range : optional parameter and it the lower and upper range of the bins.

align : optional parameter and it controls how the histogram is plotted. {'left', 'mid', 'right'}

color : optional parameter and it is a color spec

label : This parameter is an optional parameter and it is a string to match datasets.

To construct a histogram, follow these steps :

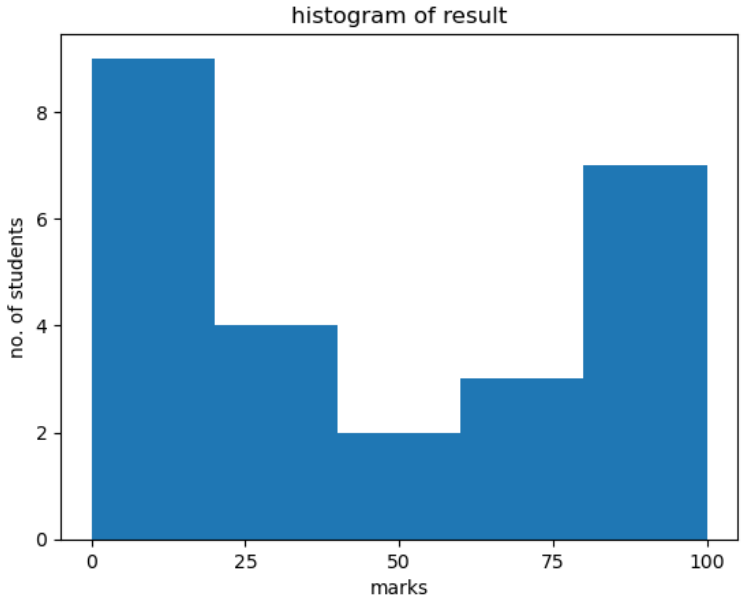
1) **Bin** the range of values.
2) Divide the entire range of values into a series of intervals.
3) Count how many values fall into each interval.

Program:

```
from matplotlib import pyplot as plt
import numpy as np
# Creating dataset
```

```
a = np.array([5,5,6,7,8,10,15,15,18,20,25,27,30,50,55,65,65,68,
88, 89,90,92,94,96,99])
plt.hist(a, bins = [0, 20, 40, 60, 80, 100])
plt.title("histogram of result")
plt.xticks([0,25,50,75,100])
plt.xlabel('marks')
plt.ylabel('no. of students')

plt.show()
```

Output:

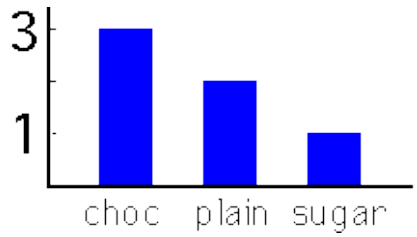


Differences between Bar chart and Histogram

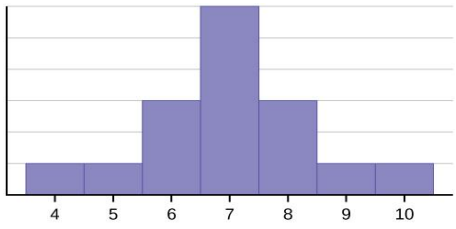
Histogram is similar to bar chart but the difference is

1) the type of data that is presented
2) the way they are drawn.

- Bar graphs are usually used to display "categorical data", that is data that fits into categories.
- Histograms on the other hand are usually used to present "continuous data", that is data that represents measured quantity where the numbers can take on any value in a certain range. Ex: A good example is weight.



Bar chart



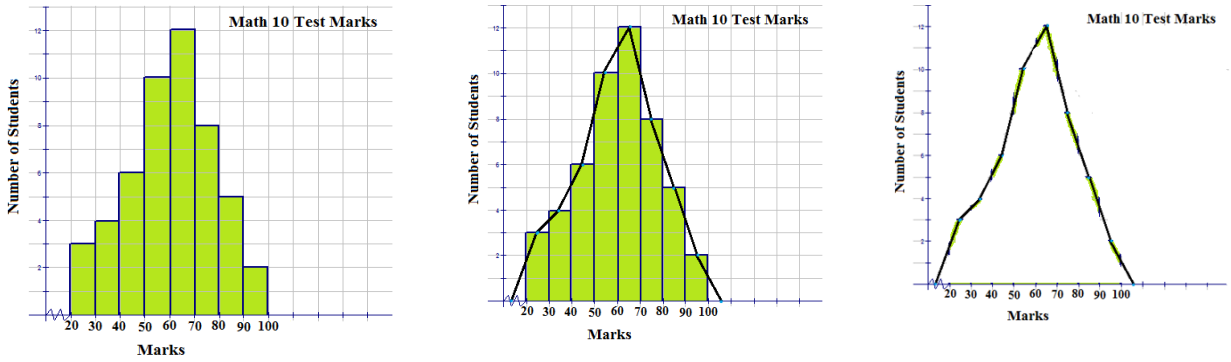
Histogram

3) Frequency polygon

A **frequency polygon** shows the overall distribution of a data set. It looks like a line graph–but the points on the graph can be plotted using data from a **histogram** or a **frequency table**.

Frequency polygons are especially useful for **comparing two data sets**

Frequency polygon plotted using data from a **histogram**



Frequency polygon plotted using data from the **Frequency table**

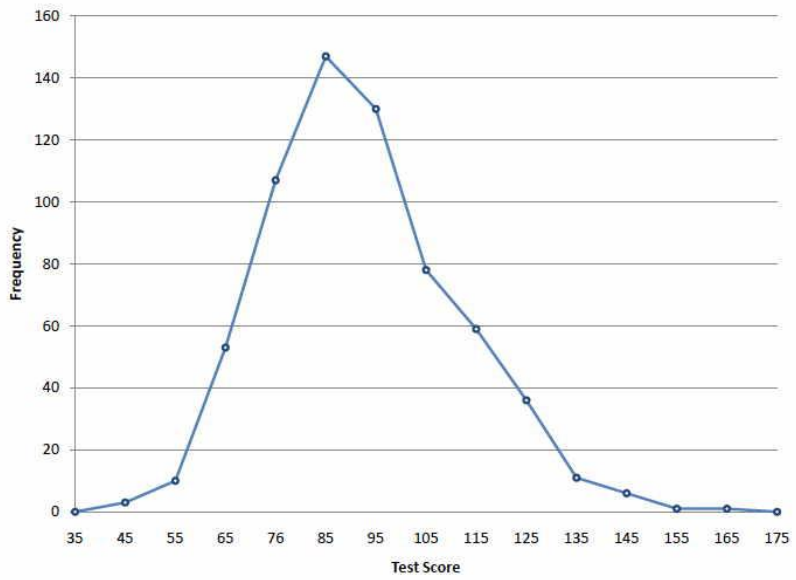
- Frequency polygons are very similar to histograms, except histograms have bars, and frequency polygons have dots and lines connecting the frequencies of each class interval.
- To create a frequency polygon, start just as for histograms, by choosing a class interval.
- Then draw an X-axis representing the values of the scores in your data.
- Mark the middle of each class interval with a tick mark, and label it with the middle value represented by the class.
- Draw the Y-axis to indicate the frequency of each class. Place a point in the middle of each class interval at the height corresponding to its frequency.
- Finally, connect the points. You should include one class interval below the lowest value in your data and one above the highest value.
- The graph will then touch the X-axis on both sides

Ex: 642 psychology test scores from frequency table

Lower Limit	**Upper Limit**	Middle of Class Interval (Round off)	**Count**	Cumulative Count
29.5	39.5	35	0	0
39.5	49.5	45	3	3

49.5	59.5	55	10	13
59.5	69.5	65	53	66
69.5	79.5	75	107	173
79.5	89.5	85	147	320
89.5	99.5	95	130	450
99.5	109.5	105	78	528
109.5	119.5	115	59	587
119.5	129.5	125	36	623
129.5	139.5	135	11	634
139.5	149.5	145	6	640
149.5	159.5	155	1	641
159.5	169.5	165	1	642
169.5	179.5	175	0	642

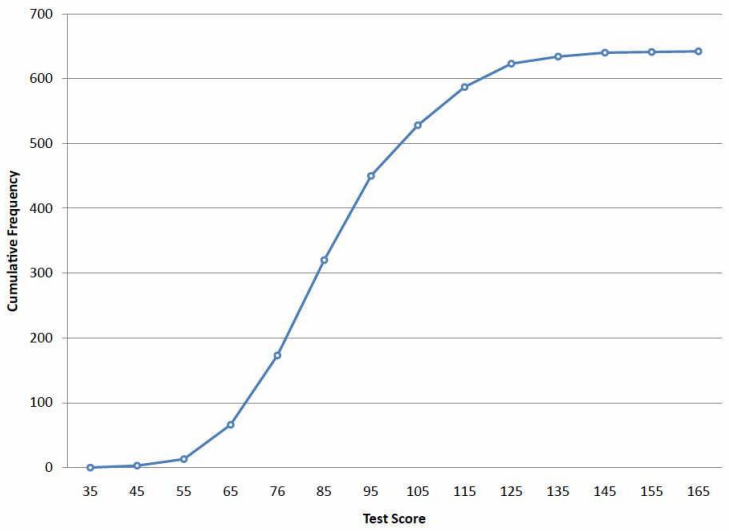
- choosing a *class interval*. Then draw an X-axis representing the values of the scores in your data. Mark the middle of each class interval with a tick mark, and label it with the middle value represented by the class.



Frequency polygon for the psychology test scores

- It is easy to discern the shape of the distribution from the above Figure. Most of the scores are between 65 and 115. It is clear that the distribution is not symmetric in as much as good scores (to the right) trail off more gradually than poor scores (to the left). The distribution is *skewed*.

Cumulative Frequency polygon

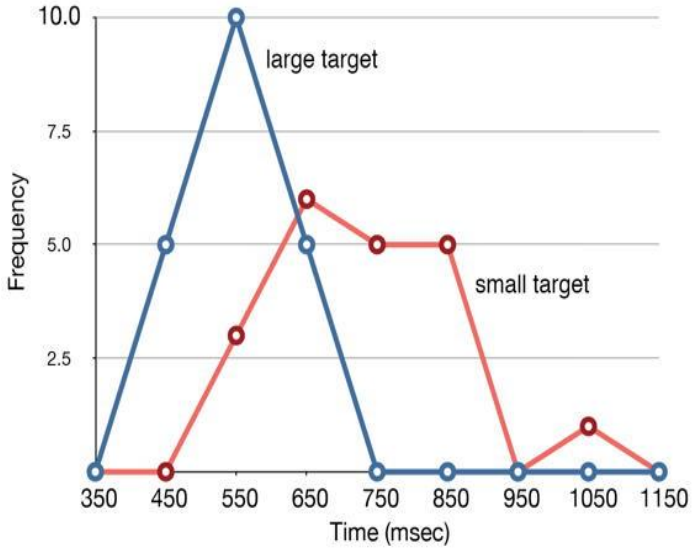


Cumulative frequency polygon for the psychology test scores

- A *cumulative frequency polygon* for the same test scores is shown in the above Figure. The graph is the same as before except that the Y value for each point is the number of students in the corresponding class interval plus all numbers in lower intervals(cumulative count).

Uses of Frequency polygon

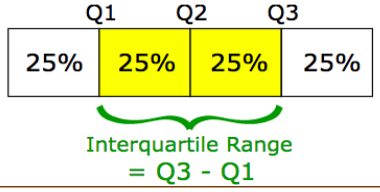
Frequency polygons are useful for comparing distributions. This is achieved by overlaying the frequency polygons drawn for different data sets.



4) Quartiles

Quartiles are measures of variation, which describes how spread out the data is. Quartiles are type of quantiles. These are values that separate the data into four equal parts.

Quartiles are the values that divide the data in to four equal parts (quarters)

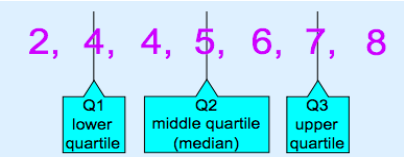


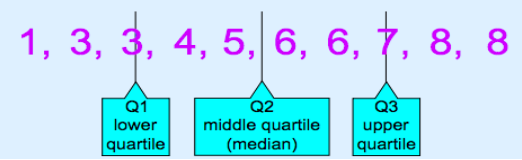
- The **first quartile**, or **lower quartile**, is the value that cuts off the first 25% of the data when it is sorted in ascending order.
- The **second quartile**, or **median**, is the value that cuts off the first 50%.
- The **third quartile**, or **upper quartile**, is the value that cuts off the first 75%.

Procedure :

Arrange the data in ascending order then start by finding the median or middle value. The median splits the data values into halves. Again find the middle value of these two halves.

Example:





With Python use the NumPy library quantile() method to find the quartiles

Syntax:

```
quantile (x,q)
```

x : This parameter is the sequence of data.

q : List of quartiles 0.25 for Q1, 0.5 for Q2, 0.75 for Q3 and 0 for minimum and 1 for maximum values

Program:

```
import numpy
values = [13,21,21,40,42,48,55,72]
x = numpy.quantile(values, [0,0.25,0.5,0.75,1])
print(x)
```

Output:

```
[13.   21.   41.   49.75 72.  ]
```

Here Q0(Minimum) = 13, Q1(First Quartile) = 21, Q2(Second Quartile) = 41, Q3(Third Quartile) = 49.75, Q4(Maximum) = 72

Program: only first quartile

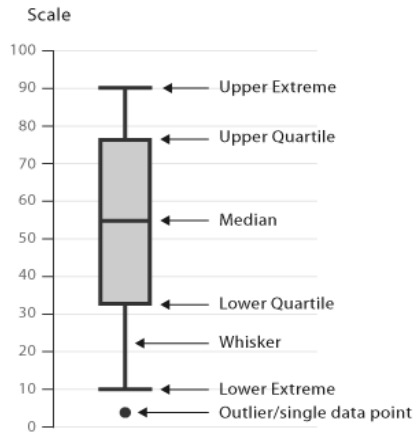
```
import numpy
values = [13,21,21,40,42,48,55,72]
x = numpy.quantile(values, [.25])
print(x)
```

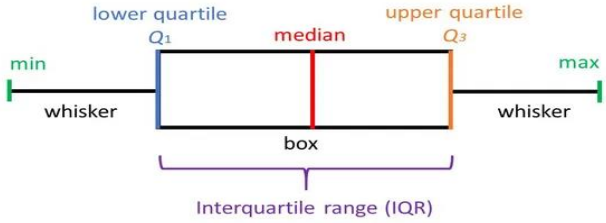
Output:

```
[21.]
```

5) Box-plot

A **Box Plot** is also known as **Whisker plot** is created to display the summary of the set of data values having properties like minimum, first quartile, median, third quartile and maximum. In the box plot, a box is created from the first quartile to the third quartile, a vertical line is also there which goes through the box at the median. Here x-axis denotes the data to be plotted while the y-axis shows the frequency distribution.





The *matplotlib.pyplot* module of *matplotlib* library provides *boxplot( )* function

Syntax:

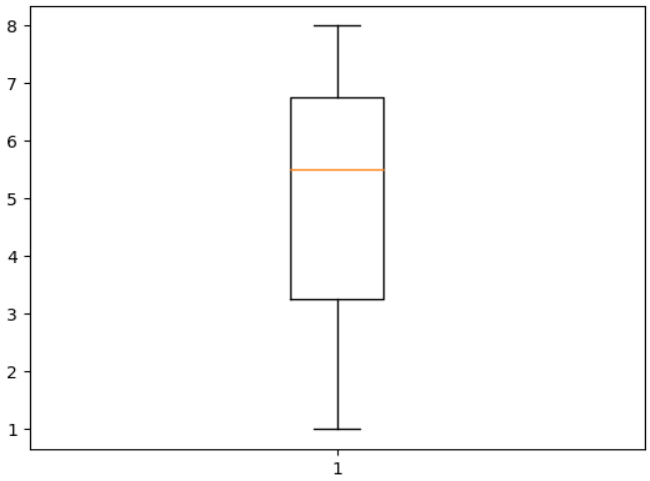
boxplot (data, notch, vert, widths)

data	array or sequence of array to be plotted
notch	optional parameter accepts boolean values
vert	optional parameter accepts boolean values false and true for horizontal and vertical plot respectively
widths	optional parameter accepts array and sets the width of boxes

Program:

```
import matplotlib.pyplot as plt

# Creating dataset
data = [1,3,3,4,5,6,6,7,8,8]

# Creating plot
plt.boxplot(data)

# show plot
plt.show()
```

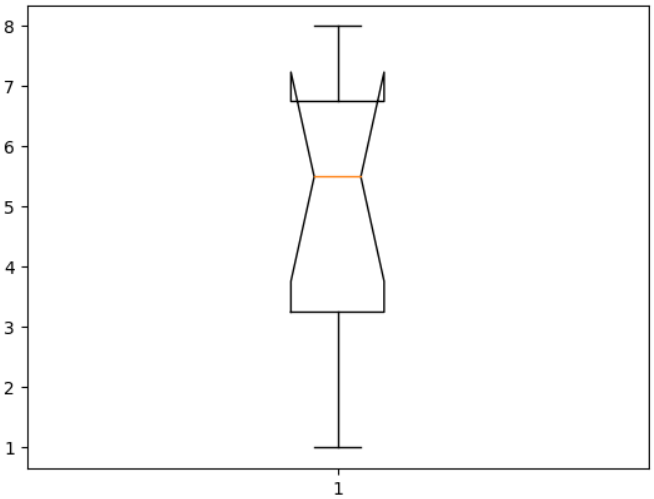
Output:



Program:

```
import matplotlib.pyplot as plt

# Creating dataset
data = [1,3,3,4,5,6,6,7,8,8]

# Creating plot
plt.boxplot(data, notch=True)

# show plot
plt.show()
```

Output:



Uses of Boxplots

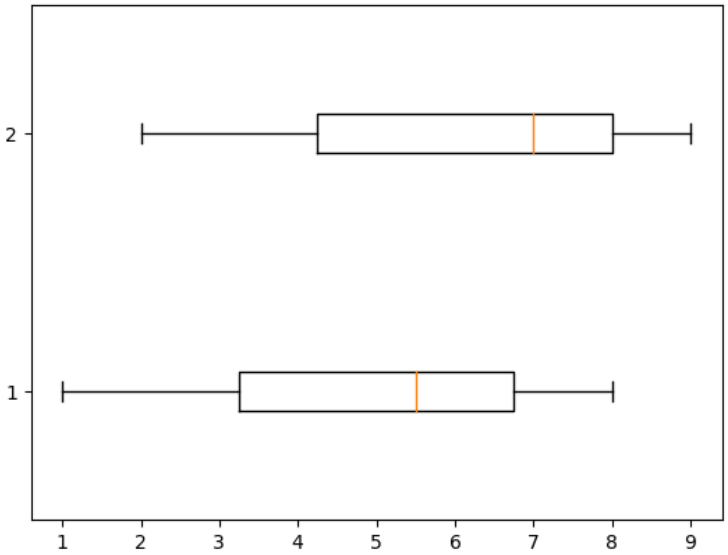
1) To compare and analyses two data sets
2) To identify outliers

1) To compare and analyse two data sets

Program:

```
import matplotlib.pyplot as plt

# Creating dataset
data1 = [1,3,3,4,5,6,6,7,8,8]
data2 = [2,4,4,5,7,7,8,8,8,9]

data = [data1, data2]

# Creating plot
plt.boxplot(data, vert = False)

# show plot
plt.show()
```

Output:



2) To Identify outliers

Outliers : In a box plot, an outlier is an a data point that is located outside the whiskers of the box plot. That is 1.5 times interquartile range above the upper quartile or below the lower quartile

outlier = (Q1 - 1.5 * IQR) or (Q3 + 1.5 * IQR)

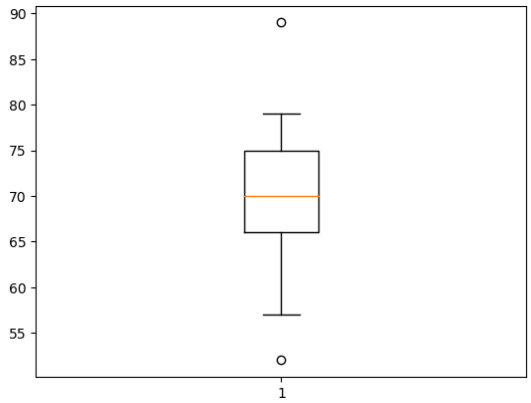
Here IQR – Inter quartile Range = Q3-Q1

Program:

```
# with outliers
import matplotlib.pyplot as plt

# Creating dataset
data = [52, 57, 57, 58, 63, 66, 66, 67, 67, 68, 69, 70, 70,
```

```
70, 70, 72, 73, 75, 75, 76, 76, 78, 79, 89]
# Creating plot
plt.boxplot(data)
# show plot
plt.show()
```

Output:



The round circle represents outliers. In this case Maximum and minimum are outliers.

6) Scatter Plot:

A scatterplot is used for visualizing the relationship between two paired sets of data. Scatter plots are widely used to represent relation among variables and how change in one affects the other.

- Data are displayed as a collection of points, each having the value of one variable determining the position on the horizontal axis and the value of the other variable determining the position on the vertical axis.
- A scatter plot can suggest various kinds of correlations between variables with a certain confidence interval.

  Ex: weight and height, weight would be on y axis and height would be on the x axis.
- Correlations may be positive (rising), negative (falling), or null (uncorrelated).
- If the pattern of dots slopes from lower left to upper right, it indicates a positive correlation between the variables being studied. If the pattern of dots slopes from upper left to lower right, it indicates a negative correlation.
- A line of best fit (alternatively called 'trendline') can be drawn in order to study the relationship between the variables.
- The scatter() method in the matplotlib library is usedto draw a scatter plot. Scatter plots are widely used to represent relation among variables and how change in one affects the other.

Syntax:

```
scatter(x_axis_data, y_axis_data, s, c, marker,
            linewidths, edgecolors)
```

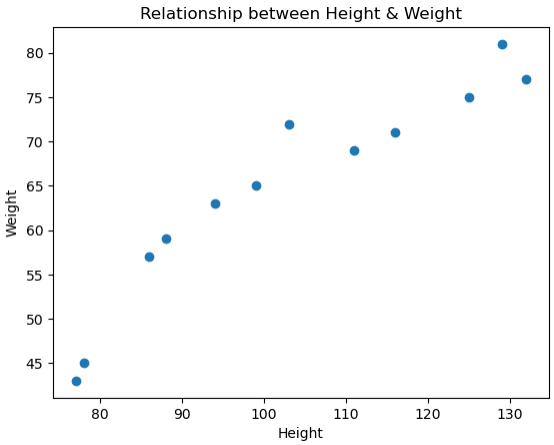
x_axis_data- An array containing x-axis data
y_axis_data- An array containing y-axis data

s- optional, marker size (can be scalar or array of size equal to size of x or y)
c- optional, color of sequence of colors for markers
marker- optional, marker style
linewidths- optional, width of marker border
edgecolor- optional**.** marker border color

Program:

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

height = np.array([99,86,88,111,132,103,125,94,78,77,129,116])
weight = np.array([65,57,59,69,77,72,75,63,45,43,81,71])

plt.scatter(height, weight)

plt.xlabel('Height')
plt.ylabel('Weight')
plt.title('Relationship between Height & Weight')
plt.show()
```

Output:

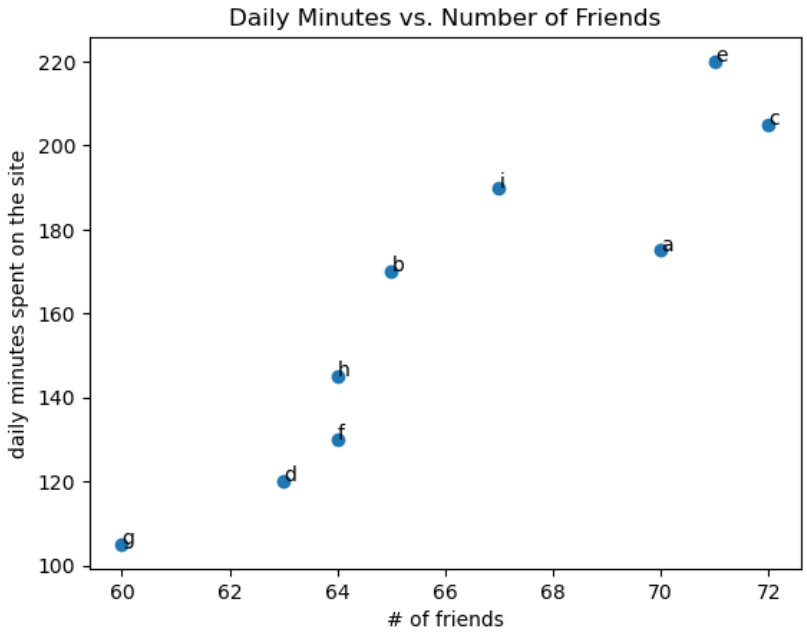


Program:

illustrates the relationship between the number of friends the users have and the number of minutes they spend on the site every day:

```
import matplotlib.pyplot as plt

friends = [ 70, 65, 72, 63, 71, 64, 60, 64, 67]
minutes = [175, 170, 205, 120, 220, 130, 105, 145, 190]
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']

plt.scatter(friends, minutes)

# label each point
for label, f_count, m_count in zip(labels, friends, minutes):
    plt.annotate(label, xy=(f_count, m_count))

plt.title("Daily Minutes vs. Number of Friends")
plt.xlabel("# of friends")
plt.ylabel("daily minutes spent on the site")

plt.show()
```

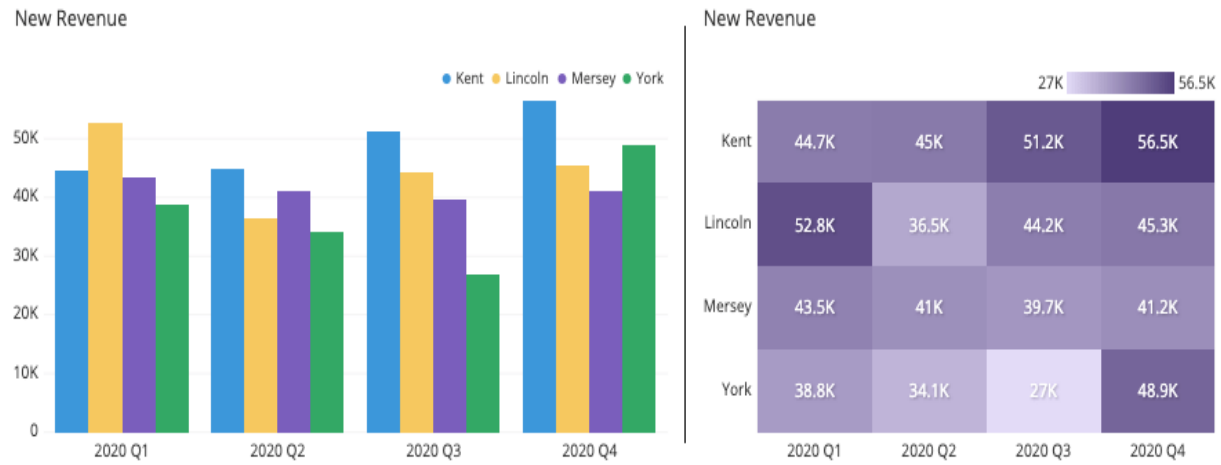
Output:



7) Heat Maps:

A heat map is a graphical representation of data where the individual values contained in a matrix are represented as colors. i.e a colored representation of a matrix. It is a great tool for visualizing data across the surface. It is used to see the correlation between columns of a dataset where we can use a darker color for columns having a high correlation.

Ex: Below figure shows the analysis of the data by using bar charts and heat maps. It is easy to understand the relationship between data by using heatmaps.



Matplotlib Heatmaps can be created using functions such as ***imshow()***

Syntax:

```
imshow(X, cmap, alpha)
```

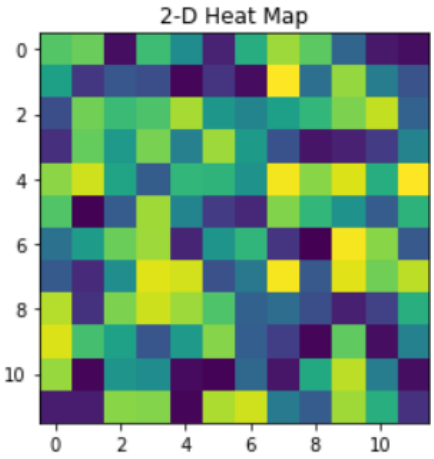
X :- this is input data matrix which is to be displayed
cmap :- Optional parameter - Colormap use to display the heatmap,default Viridis colour palette
alpha :- Optional parameter - it specifies the transpiracy of the heatmap

Method 1: Using *matplotlib.pyplot.imshow()* Function

Program:

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

data = np.random.random(( 12 , 12 ))
plt.imshow( data)

plt.title( "2-D Heat Map" )
plt.show()
```

Output:



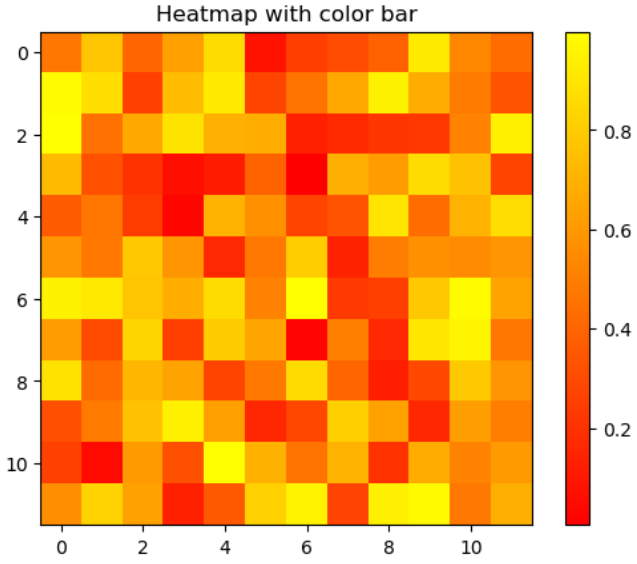
Adding Colorbar to Heatmap Using Matplotlib:

Add a colorbar to the heatmap using plt.colorbar(). colorbar shows the weight of color relatively between a certain range.

Program:

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

data = np.random.random((12, 12))
plt.imshow(data, cmap='autumn')

# Add colorbar
plt.colorbar()

plt.title("Heatmap with color bar")
plt.show()
```

Output:



Method 2: Using Seaborn Library

Seaborn library has a function called seaborn.heatmap() to create heatmaps

Program:

```
import numpy as np
import seaborn as sns
import matplotlib.pylab as plt

data_set = np.random.rand( 10 , 10 )
ax = sns.heatmap( data_set , linewidth = 0.5 , cmap = 'coolwarm' )
plt.title( "2-D Heat Map" )

plt.show()
```

Output: