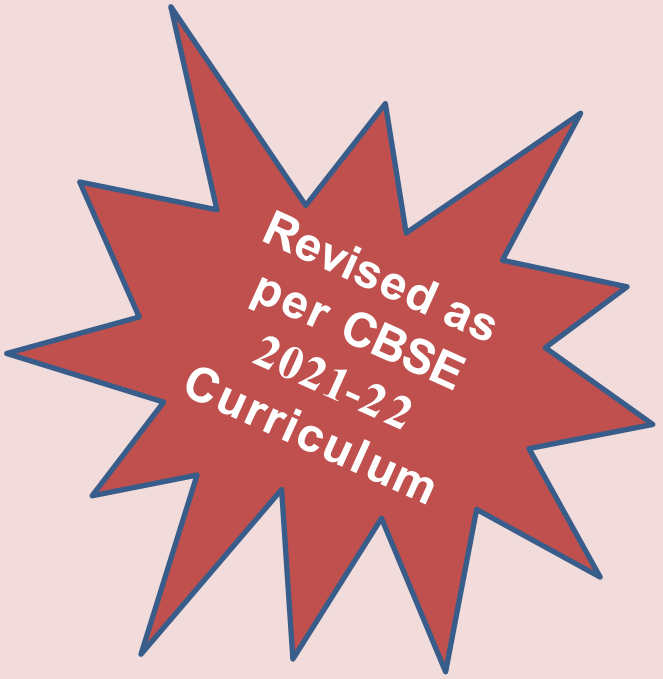


Data Visualization

Expected Learning Outcome:

CBSE Syllabus (2021-22) Covered in this presentation:

- **Data Visualization**: Purpose of plotting, drawing and saving of plots using Matplotlib (line plot, bar graph, histogram, pie chart, frequency polygon, box plot and scatter plot).
- Customizing plots: color, style (dashed, dotted), width; adding label, title, and legend in plots.

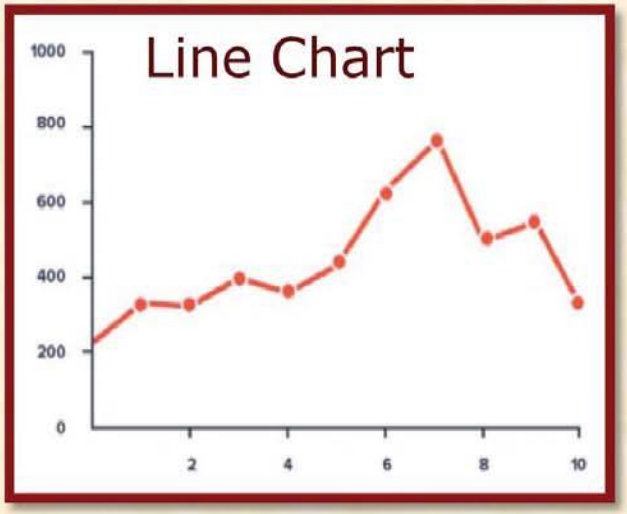
In this presentation you will learn about Data Visualization (plotting of various types of graphs) using Matplotlib library.

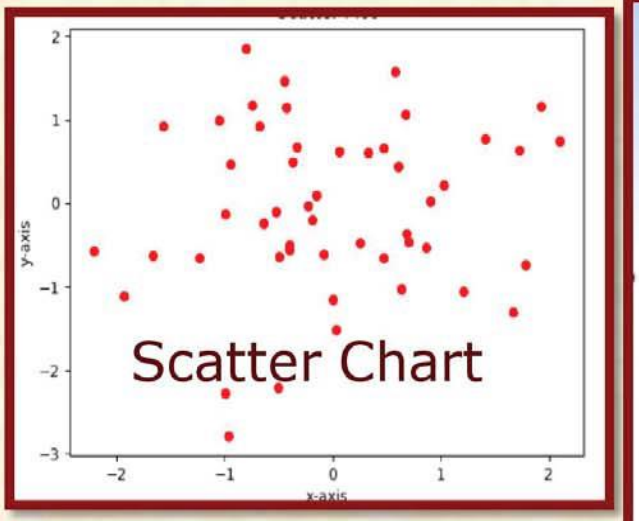
- Concepts of Data Visualization and its need.
- How to use Matplotlib for plotting/drawing.
- Understa nding of different types of Graphs and plotting of various types of graphs (line plot, bar graph, histogram, pie chart, frequency polygon, box plot and scatter plot) using Matplotlib on data set.
- Customizing plots : color, style (dashed, dotted), width; adding label, title, and legend in plots.

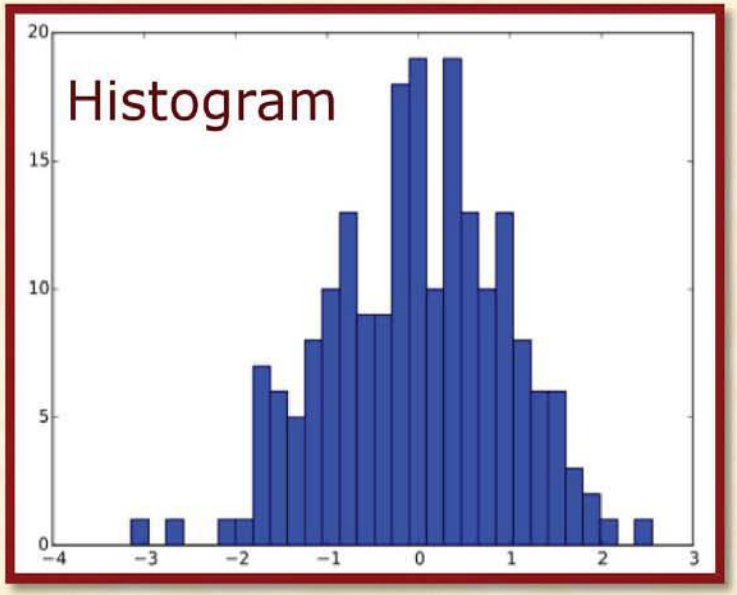
Concept & purpose of Data Visualization

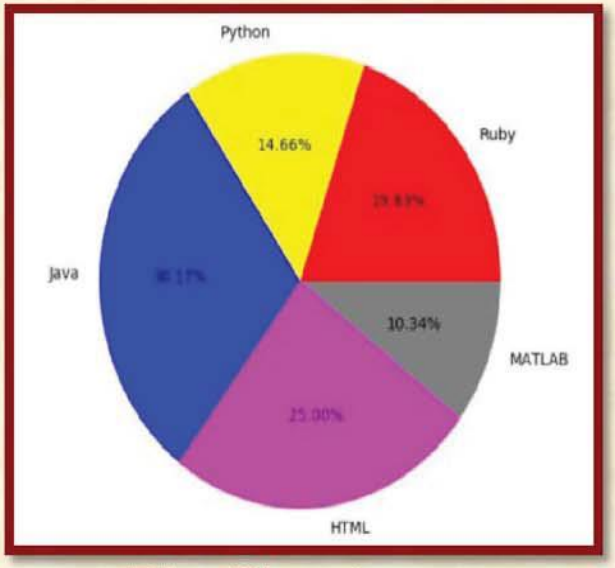
- Data visualisation means graphical or pictorial representation of the data using some pictorial tools like graph, chart, etc. Since it is well known fact that image presentation is more effective that textual representation.
- Visualisation also helps to effectively communicate information to users. In real life, we interact with Traffic symbols, Atlas or map book, speedometer of a vehicles etc. which are pictorial representation of facts.
- Visualisation of data is effectively used in fields like health, finance, science, mathematics, engineering etc.
- In Pandas, we have learned so many data analysis functions which can be applied on series or dataframe. These analysis can be used as conclusions to make better decisions. In such cases, visualisation helps in better understanding of results of the analysis in pictorial way.

Types of charts (plots)

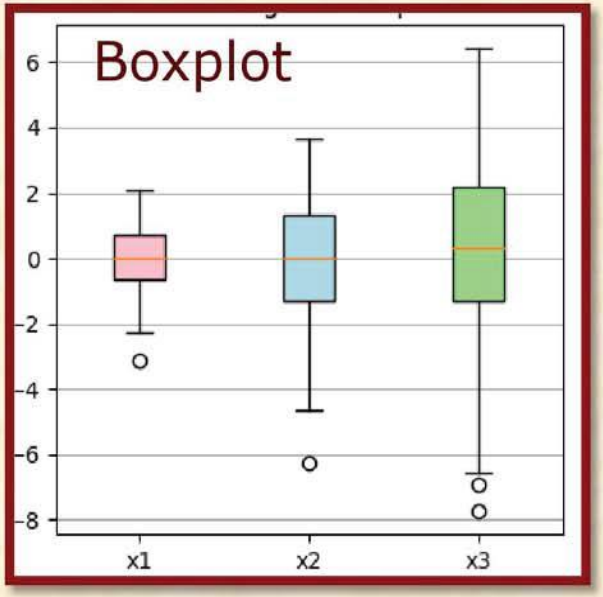






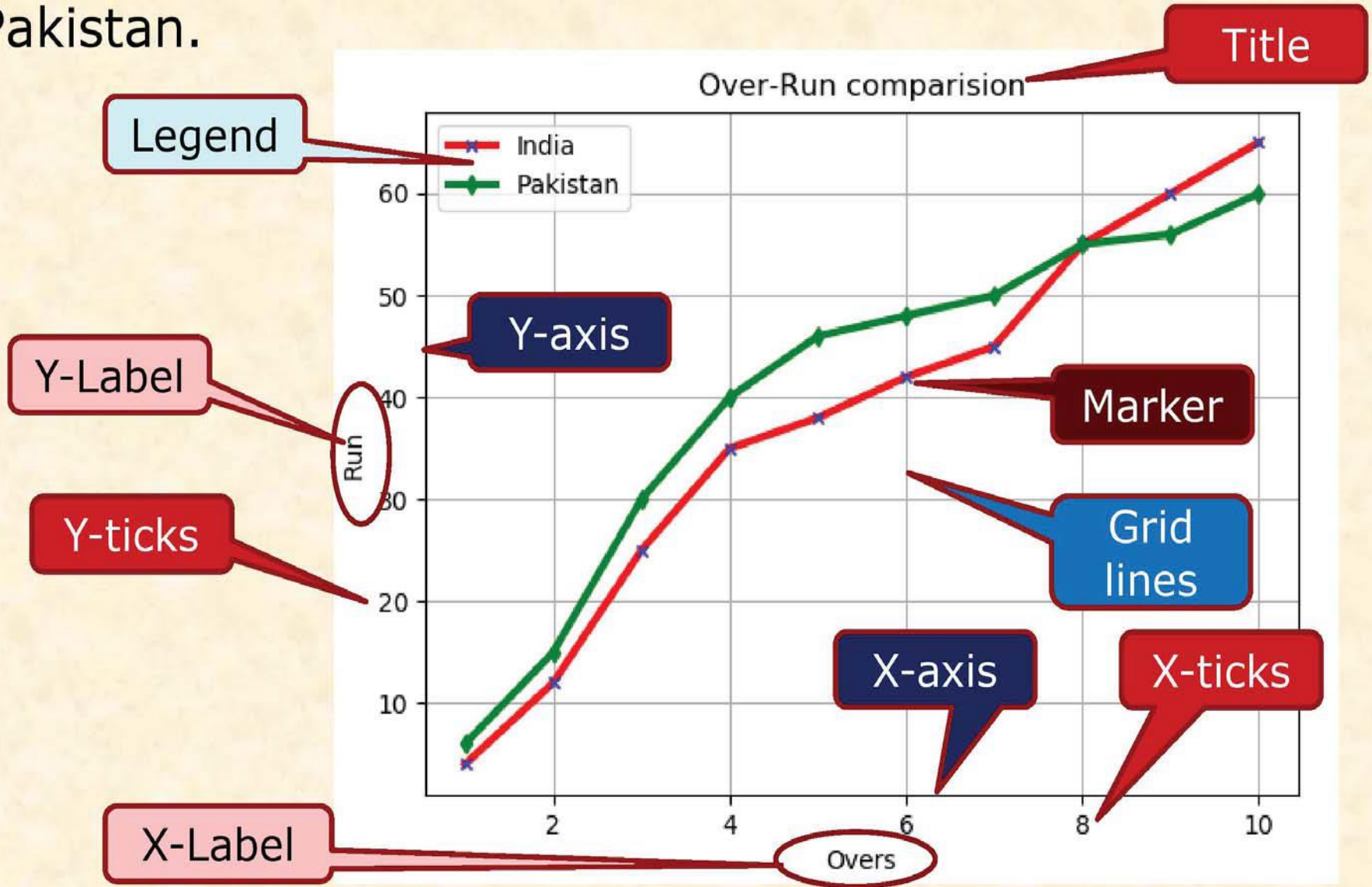


Pie Chart



Elements of plots/charts

Consider the following **line chart** depicting comparison between Overs (x-axis) and Runs (y-axis) of India and Pakistan.



Role of Matplotlib Library in Data Visualization

- The **Matplotlib** library of Python, is used for creating static, animated, and interactive 2D-plots in Python. It can be installed using the following pip command from the command prompt: **pip install matplotlib**
- The **pyplot** being a collection of methods (module), works as an interface to Matplotlib library. So, for plotting using Matplotlib, we need to import its Pyplot module using the following command: **import matplotlib.pyplot as plt**

Here, **plt** is an alias or an alternative name for matplotlib.pyplot object. You can use any other name (alias) also. Once object name (alias) has been defined, you can use functions for plotting.

Example: **plt.plot(a,b)** **or** **matplotlib.pyplot.plot(a,b)**

Pyplot methods for plotting

Pyplot offers the following functions, which can be used to plot various types of graphs or charts.

Chart type	Function	purpose
Line chart	plot()	Visualize information as a series of data point (markers) connected by line.
Scatter chart	scatter()	Similar to line chart, in which data points are not connected with line.
Bar chart	bar()/barh()	Visualize data with rectangular bars with height/length proportional to value.
Pie chart	pie()	A circle shaped chart which depicts numerical proportional of data as arc.
Histogram	hist()	Visualize the number of data points (frequency) that lie within range of values.
Box plot	boxplot()	Visual representation of five statistical summery for given data.

Pyplot methods for customizing plots

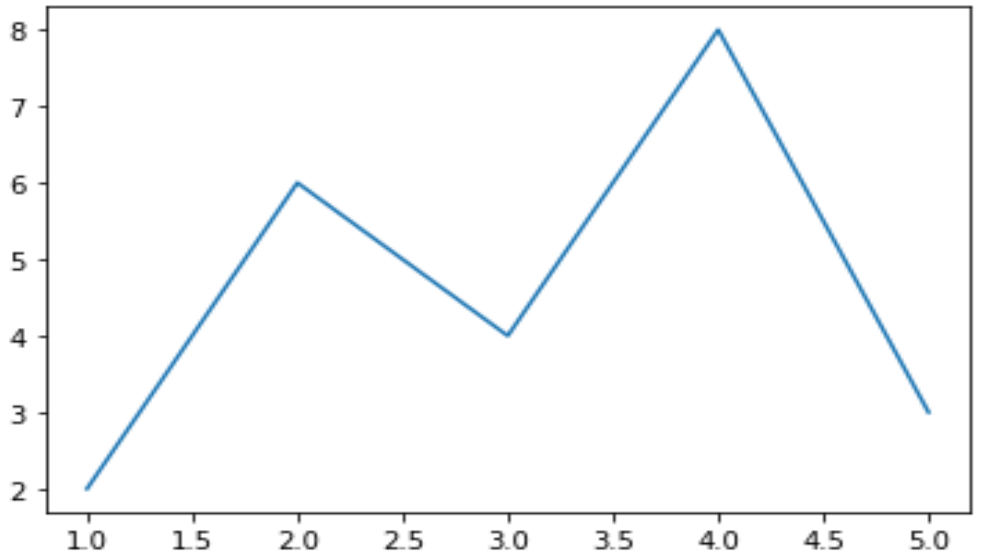
Pyplot also offers some additional functions to customize plot and settings etc.

Functions	purpose
grid()	Configures Gridlines in the graph/plot.
legend()	Displays legend of the axis for identification.
savefig()	Saves the plot as image/pdf file types.
show()	Displays the plot/graph.
title()	Defines the title for the plot/graph.
xlabel()	Sets the label for x-axis.
ylabel()	Sets the label for y-axis.
xticks()	Sets the tick location and label on x-axis
yticks()	Sets the tick location and label on y-axis

Line Chart: plot()

❖ **plot(): Creates line chart for given x and y values.** Pyplot's plot() function is used to generate line chart for given values for x axis and y-axis. Data for X and Y axis may list, series, column of Data Frame.

```
#  creating simplest line chart
#  through List of values
import matplotlib.pyplot as plt
xdata= [1,2,3,4,5]
ydata= [2,6,4,8,3]
#  plotting graph
plt.plot(xdata,ydata)
#  Displaying Graph
plt.show()
```



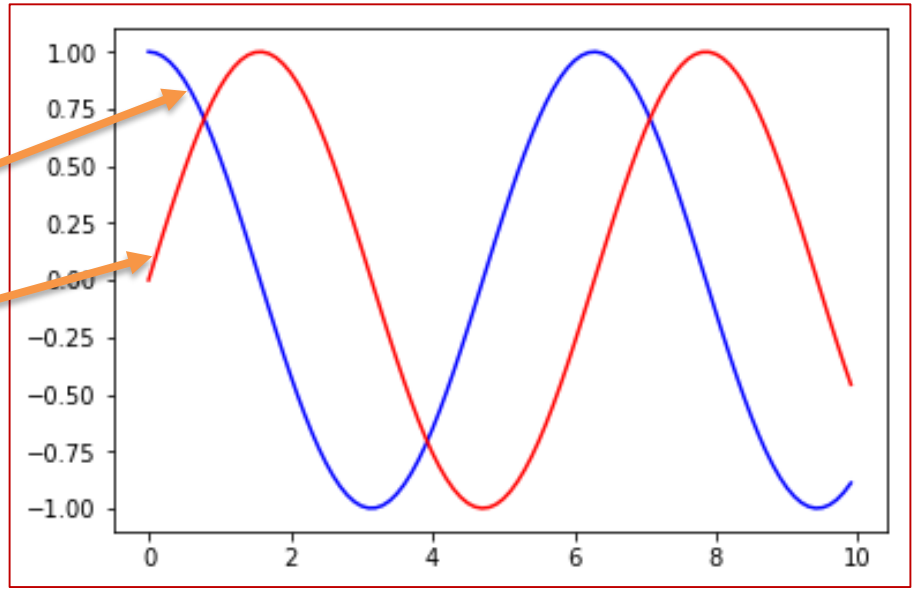

```
#creating line chart through DataFrame
import pandas as pd
import matplotlib.pyplot as plt
dct={'X':[1,2,3,4,5],'y':[2,6,4,8,3]}
df=pd.DataFrame(dct)
plt.plot(df['X'],df['y'])
plt.show()
```

Data Frame

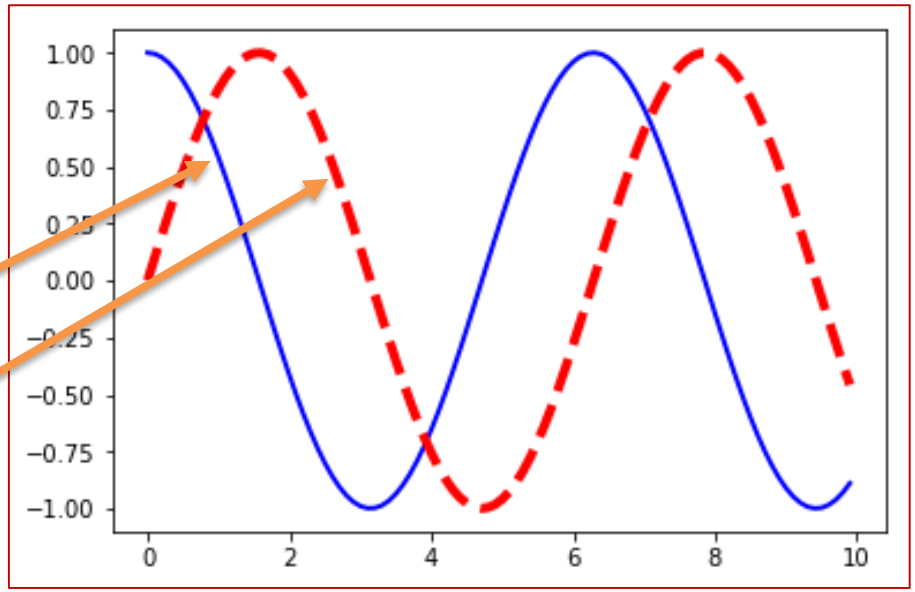
	X	Y
0	1	2
1	2	6
2	3	4
3	4	8
4	5	3

Line Chart: plot()

```
import matplotlib.pyplot as  plt
import numpy as np
x=np.arange(0.,10,0.1)
a=np.cos(x)
b=np.sin(x)
plt.plot(x,a,'b')
plt.plot(x,b,'r')
plt.show()
```




```
import matplotlib.pyplot as  plt
import numpy as np
x=np.arange(0.,10,0.1)
a=np.cos(x)
b=np.sin(x)
plt.plot(x,a,'b',linewidth=2)
plt.plot(x,b,'r',linewidth=4,linestyle='—')
plt.show()
```



Customizing Line Chart: plot( )

Pyplot's plot() function offers various setting to customizing plot. Some common setting are - line colour, line width, line style and marker types etc.

<pyplot obj>.plot([X-Value,] <Y-Value> [,Line / Marker-colorcode] [,linewidth=<n>] [,linestyle=<style code>] [,marker=<marker style>] [, markersize =< n>] [,markeredgecolor = <colorcode>] [,label = <text>])

X-Val ue : Dataset for X-Axis. Default value is [O..N-1], if not given values for X Axis. Dataset may be **1-D Array, List, Series or Column of Dataf rame.**
Y-Value: Dataset for Y-Axis. Number of values for X-Axis and Y-Axis should be equal.
Line/Marker-Color: Defines color code for line and Marker symbol.
Linewidth=<n>: A number indicating thickness of line.
Linestyle= <style>: Line style may be **solid, dashed, dashdot or dotted** as per given code.
Marker=<style> : Defines Marker style code.
Markersize=<n> : Defines size of marker in number.
Markeredgecolor=<color code>: Defines color for marker edge.
Label=<text>: Defines label text for legend.

Customizing Line Chart: plot( )

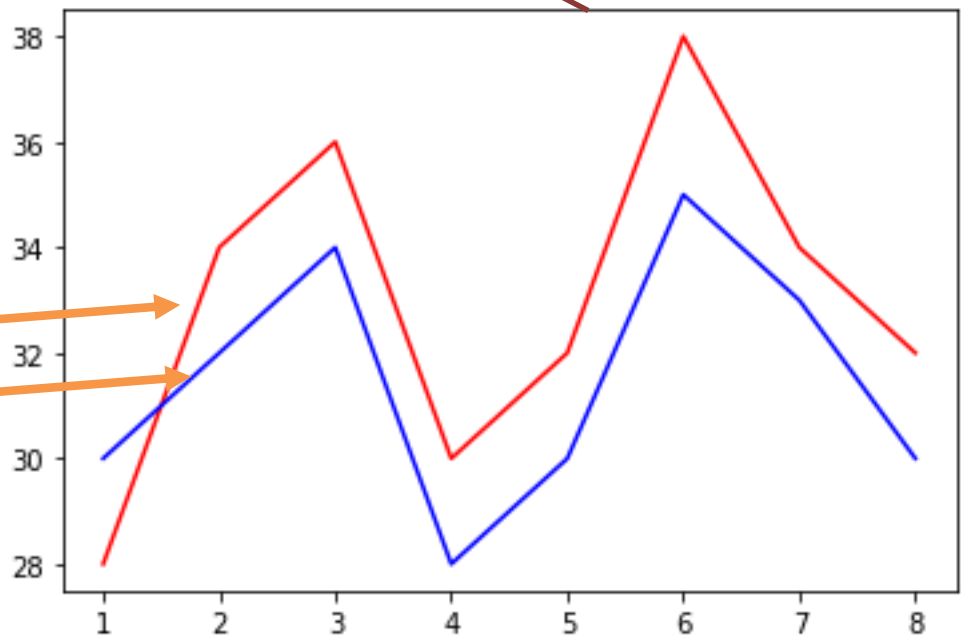
❖ Color code for lines/ Marker/ Marker edge:

Code	Color	Code	Color
b	Blue	y	Yellow
g	Green	k	Black
r	Red	c	Cyan
m	Magenta	w	White

These color code and color text can be used as line color and marker color.

You can plot multiple charts on different data set on same graphs before calling show() method.

```
# Week and average temp graph import
matplotlib.pyplot as plt w=
[1,2,3,4,S,6,7,8]
t=[28,34,36,30,32,38,34,32]
h=[30,32,34,28,30,35,33,30]
# plotting graph
plt.plot(w,t,'r')
plt.plot(w,h,'b')
# Displaying Graph plt
.show()
```



Customizing Line Chart: plot()

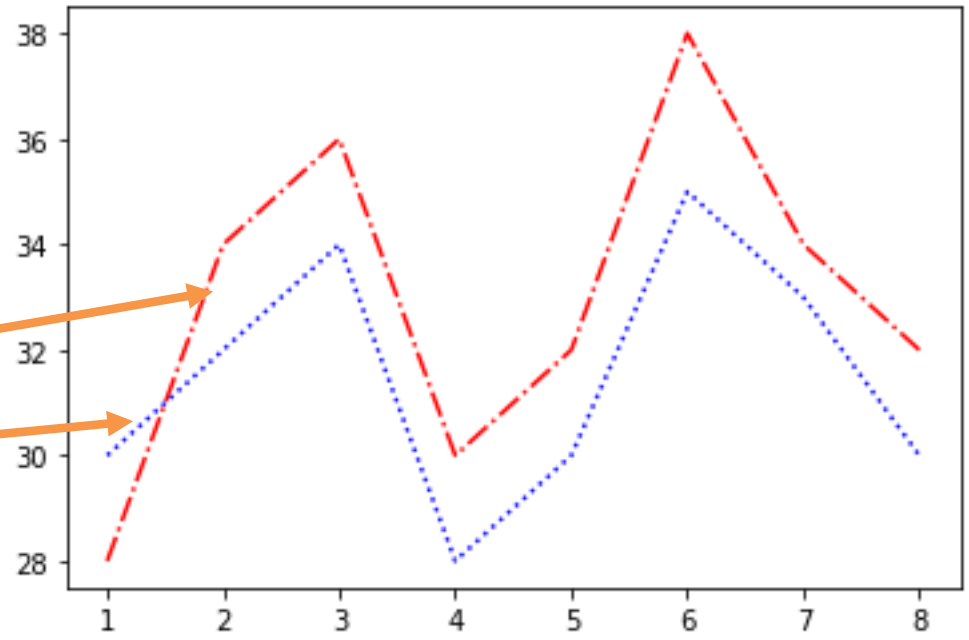
❖ Line Style for lines:

The following Style code or Keywords can be used as line style in plot() function.

You can use line style code or keyword with linestyle option.

Code	Keyword	Remarks
-	solid	Solid line
--	dashed	Dashed line
-.	dashdot	Dash-dot-dash line
:	dotted	Dotted line

```
#  Week and average temp graph
import matplotlib.pyplot as plt
w= [1,2,3,4,5,6,7,8]
t= [28,34,36,30,32,38,34,32]
h= [30,32,34,28,30,35,33,30]
# plotting graph
plt.plot(w,t,'r', linestyle= '-.' )
plt.plot(w,h, 'b',linestyle= ':')
# Displaying Graph
plt.show()
```



❖ Commonly used Marker Types:

Changing the marker Type, Size and Colour

The data points being plotted on a graph/char are called markers. To change marker type, its size and color, we can give following additional optional arguments in plot() function:

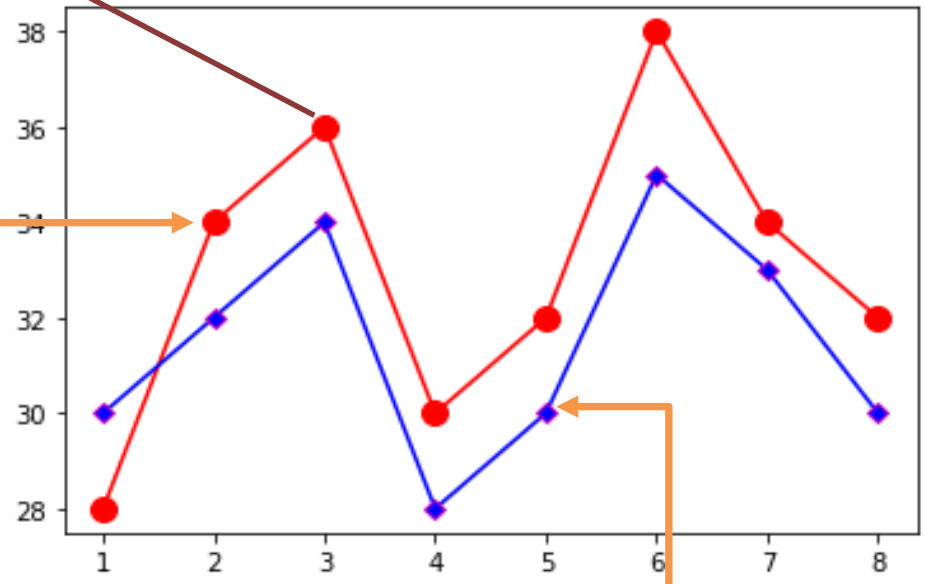
marker=<validmarker type>, markersize=<in points>, markeredgecolor=<valid color>

Marker Types for Plotting:-

Marker	Description	Marker	Description	Marker	Description
'.'	point	's'	square	'3'	tri_left
','	pixel	'p'	pentagon	'4'	tri_right
'o'	circle	'*'	star	'v'	triangle_down
'+'	plus	'h'	hexagon1	'^'	triangle_up
'x'	x marker	'H'	hexagon2	'<'	triangle_left
'D'	diamond	'1'	tri_down	'>'	triangle_right
'd'	thin_diamond	'2'	tri_up	'|', '-'	vline, hline

Customizing Line Chart: plot( )

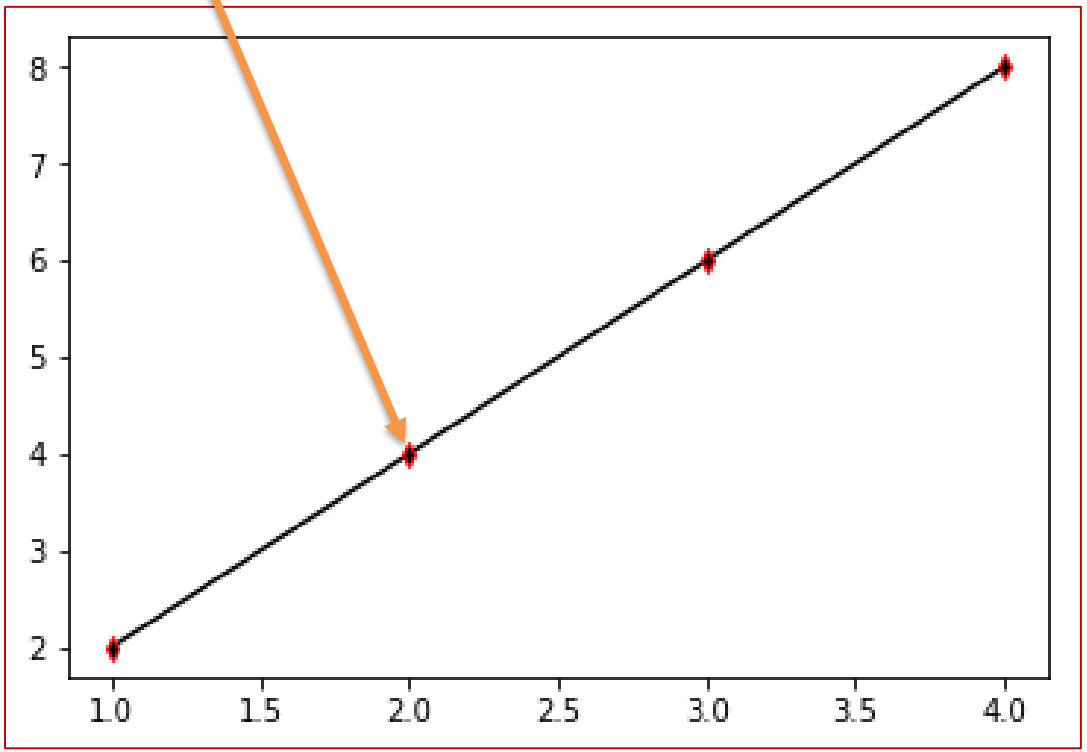
You can also define Marker type with line code color. Ex.'r+' i.e. Red line with + marker. But this will create Scatter graph




```
import matplotlib.pyplot as plt
w= [1,2,3,4,5,6,7,8]
t= [28,34,36,30,32,38,34,32]
h= [30,32,34,28,30,35,33,30]
plt.plot(w,t,'r', marker='o', markersize=10)
plt.plot(w,h, 'b', marker='D', markeredgecolor='m')
plt.show()
```

Customizing Line Chart: plot( )

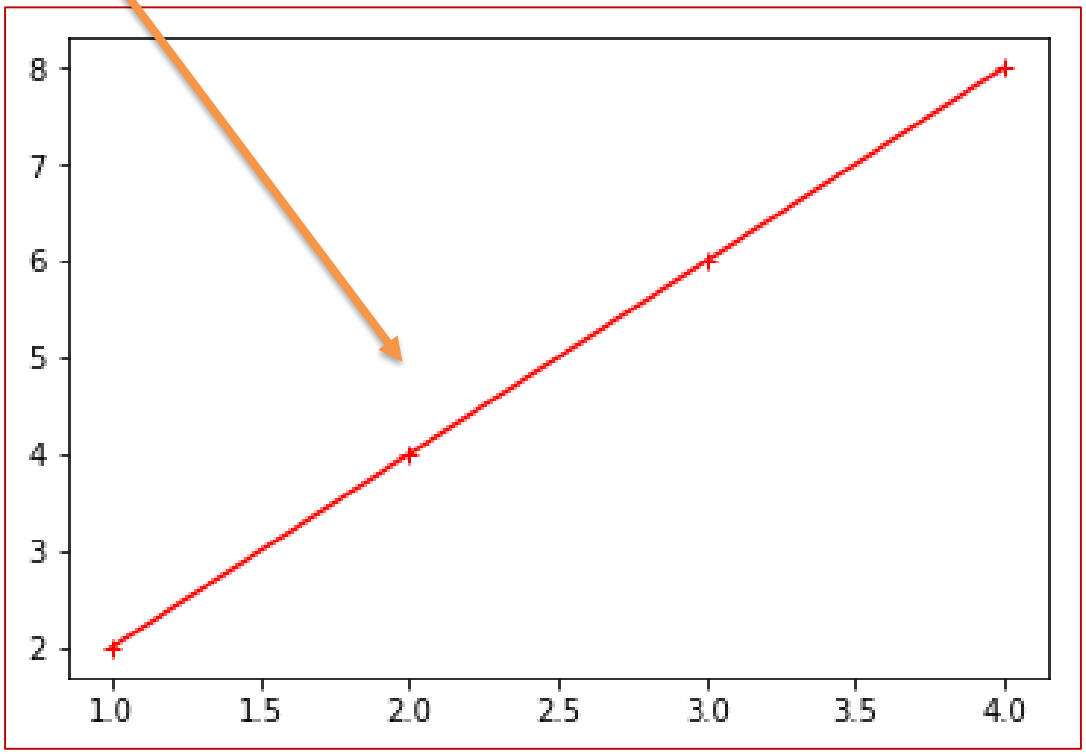
```
import matplotlib.pyplot as plt
p=[1,2,3,4]
q=[2,4,6,8]
plt.plot(p,q,'k',marker='d',markersize=5,markeredgecolor='r')
plt.show()
```



Line color is black (color 'k') Marker Type = small diamond ('d') Marker Size=5 points Marker Color='red'.

Customizing Line Chart: plot( )

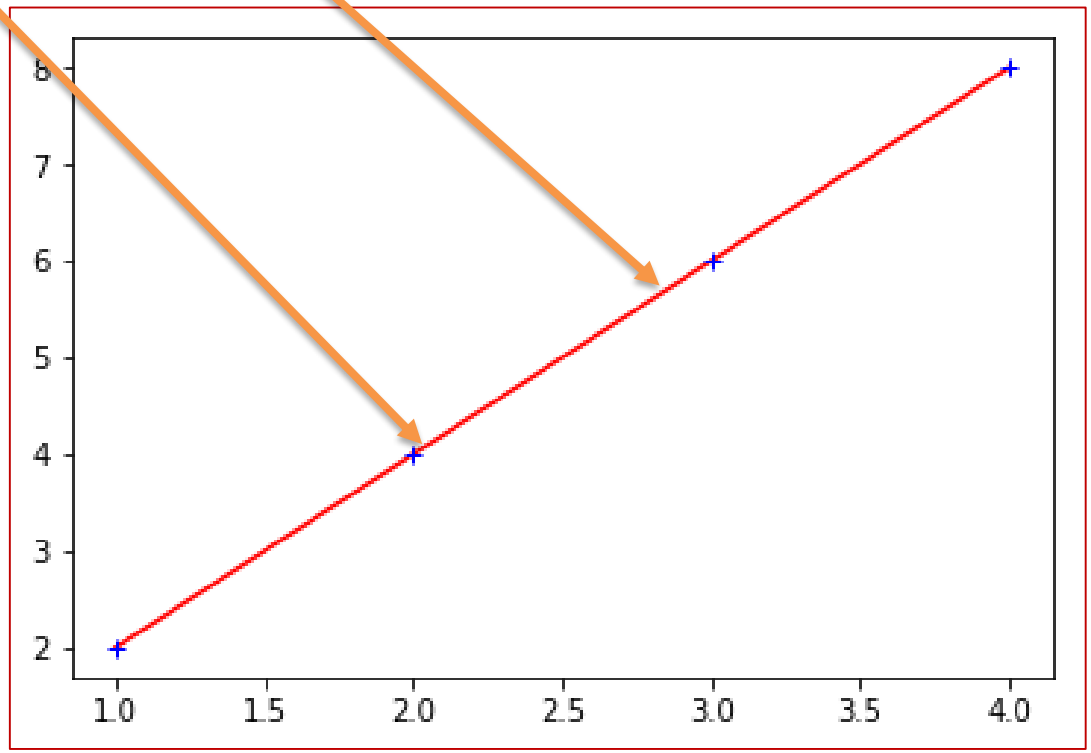
```
import matplotlib.pyplot as plt
p=[1,2,3,4]
q=[2,4,6,8]
plt.plot(p,q,'r+',linestyle='solid')
plt.show()
```



Line color and marker style combined so marker takes same color as line and style as '+'.

Customizing Line Chart: plot( )

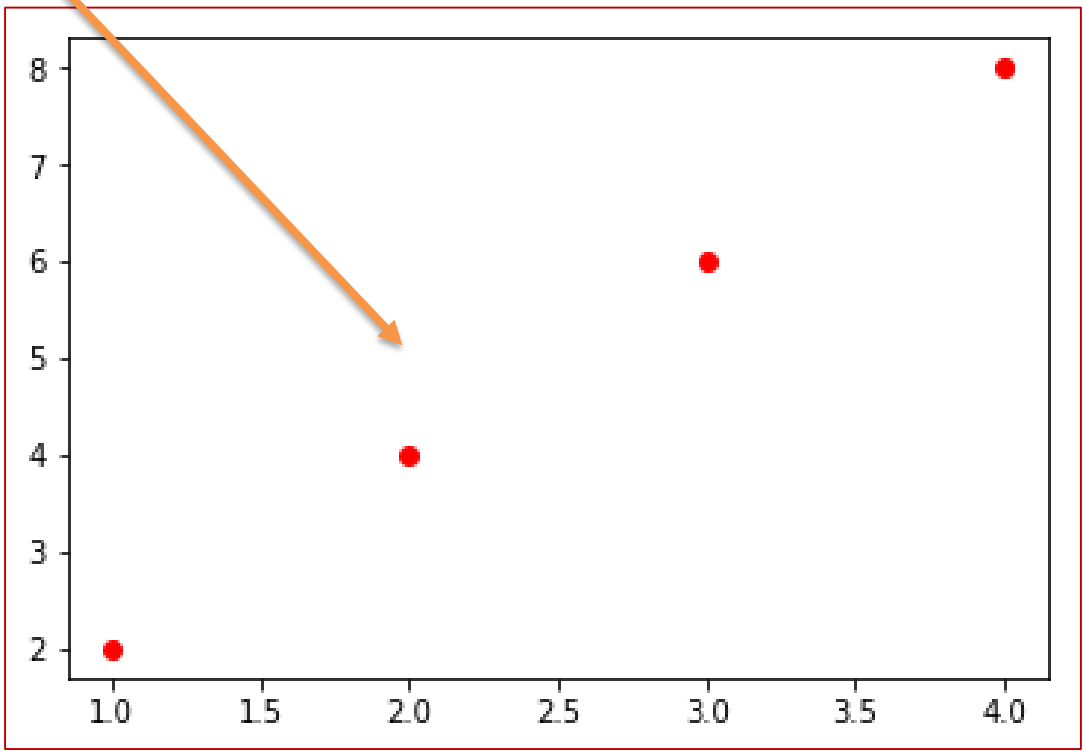
```
import matplotlib.pyplot as plt
p=[1,2,3,4]
q=[2,4,6,8]
plt.plot(p,q,'r+',linestyle='solid', markeredgecolor='b')
plt.show()
```



Line color and marker style combined. Marker color separately specified.

Customizing Line Chart: plot( )

```
import matplotlib.pyplot as plt
p=[1,2,3,4]
q=[2,4,6,8]
plt.plot(p,q,'ro')
plt.show()
```

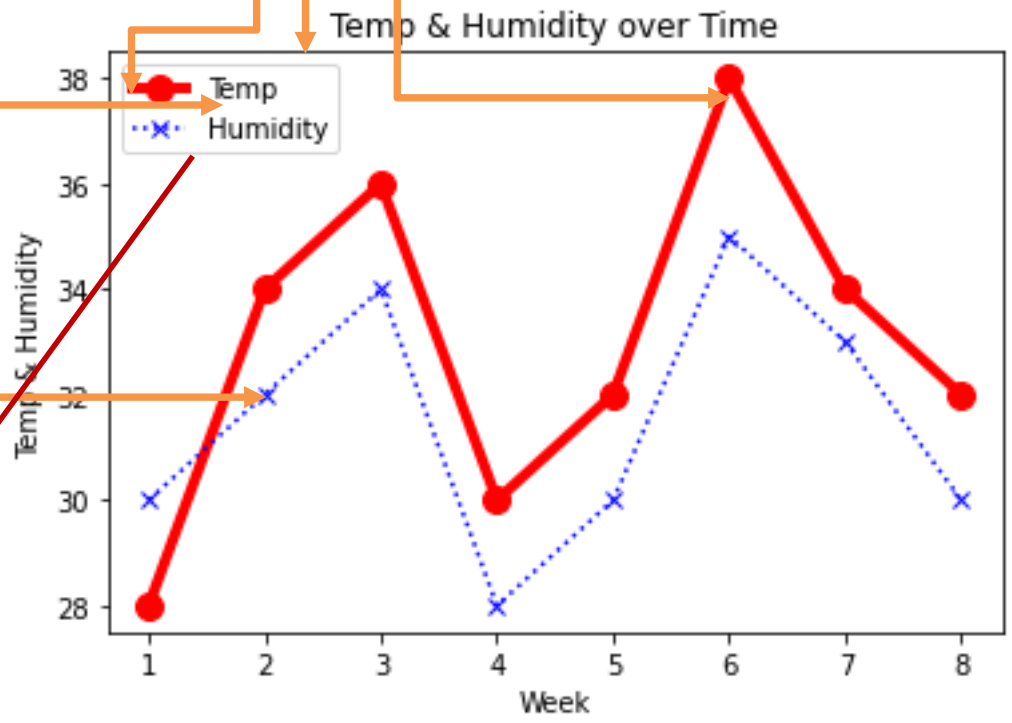


If we do not specify the line style argument separately along with linecolor and markerstyle string (e.g. 'r+' or 'bo' etc.). Python will only plot the markers and not the line.

Customizing Line Chart: plot()

❖ Example of line chart:

```
import matplotlib.pyplot as plt
w= [1,2,3,4,5,6,7,8]
t= [28,34,36,30,32,38,34,32]
h= [30,32,34,28,30,35,33,30]
# plotting graph
plt.plot(w,t,'r', linewidth=4, marker='o', markersize=10, label="Temp")
plt.plot(w,h,'b',linestyle=':',marker='x', label="Humidity")
#setting graph
plt.xlabel("Week")
plt.ylabel("Temp & Humidity")
plt.title("Temp & Humidity over Time")
plt.legend(loc="upper left")
# Displaying graph
plt.show()
```



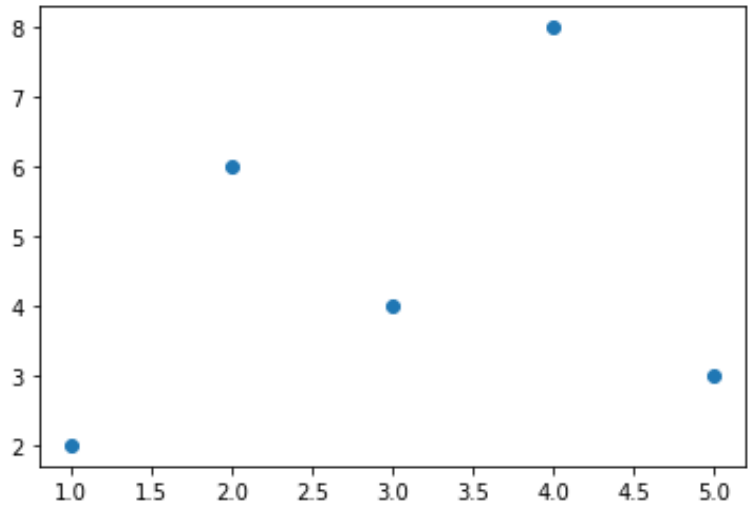
If you want to display legend with plot, then define text for legend with label option in plot() function.

Scatter Chart: scatter()

❖ scatter (): Creates scatter chart for given x and y values.

Scatter chart is chart which show s only data points on graph i.e. points are not connected with line like line graph. Pyplot's scatter() function is used to generate scatter chart for given values for x-axis and y-axis. Data for X and Y axis may list, series, column of DataFrame.

```
#  creating simplest Scatter chart
#  through List of values
import matplotlib.pyplot as plt
xdata= [1,2,3,4,5]
ydata= [2,6,4,8,3]
# plotting graph
plt.scatter(xdata,ydata)
plt.show()
```




```
#Creating line chart through Dataframe
import pandas as pd
import matplotlib.pyplot as plt
dct={'X':[1,2,3,4,5],'y':[2,6,4,8,3]}
df=pd.DataFrame(dct)
plt.scatter(df['X'],df['y'])
plt.show()
```

	x	y
0	1	2
1	2	6
2	3	4
3	4	8
4	5	3

You can also define columns of data frame

Customizing Scatter Chart: scatter()

Pyplot's scatter() function offers various setting to customize scatter chart like marker types, color and size etc. You can also define separate color and size for each marker points too by providing size and color list.

```
<pyplot obj>.scatter(<X-Value>,<Y-Value> [,marker =<marker style>] [ s= <size(s)>]
[c=<color code(s)>]  [label=<text>])
```

X-Value: Dataset for X-Axis. Dataset may be 1-D Array, List, Series or Column of Data frame.

Y-Value: Dataset for Y-Axis. Number of values for X-Axis and Y-Axis should be equal.

Marker =<style>: Defines Marker style code.

S=<size(s)>:Defines size of marker points in number. You can also define a list of sizes for each points, if required.

C= <color code(s)>: Defines color for markers. You can also define a list of separate color code for each point

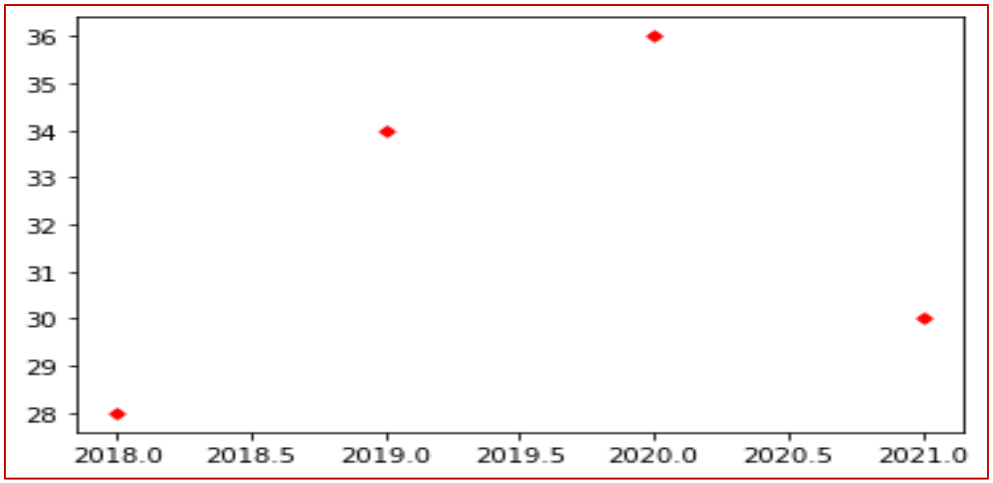
Label=<text>: Defines label text for legend.

Color code and Marker style characters can be used as per plot() method

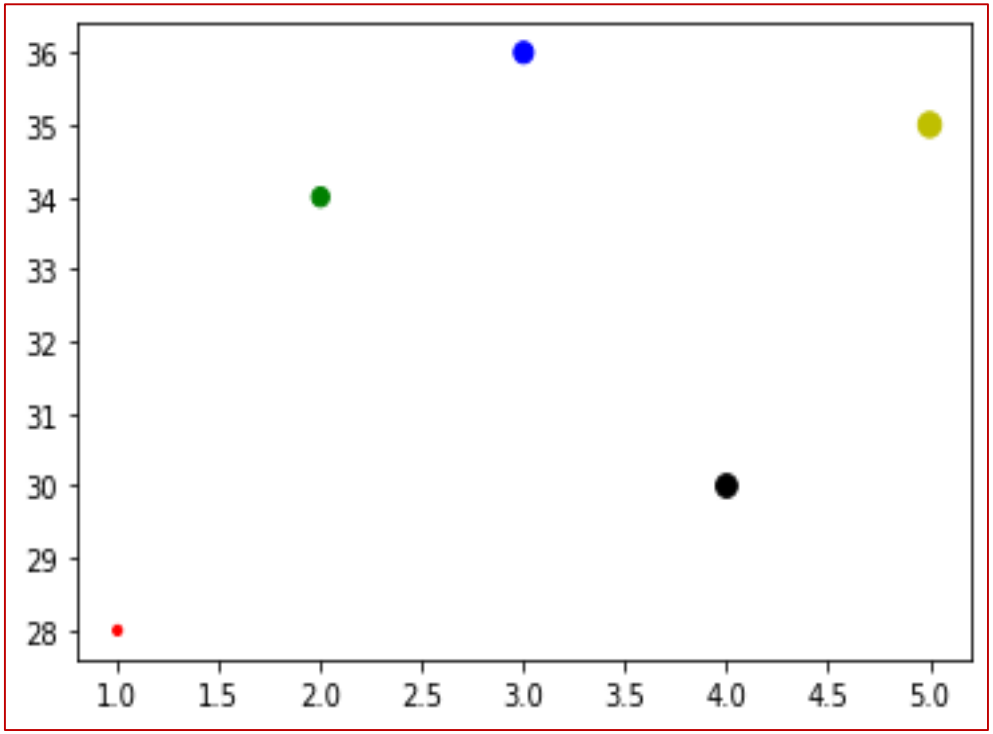
Customizing Scatter Chart: scatter ()

❖ Example of scatter chart:

```
#Scatter chart for year & pass students
import matplotlib.pyplot as plt
year=[2018,2019,2020,2021]
pas= [28,34,36,30]
#plotting graph
plt.scatter(year,pas,marker="D",s=15,c='r')
plt.show()
```




```
#Scatter chart with different size and color for
markers
import matplotlib.pyplot as plt
w=[1,2,3,4,5]
t=[28,34,36,30,35]
#plotting graph
plt.scatter(w,t,marker='o',
        s=[10,40,50,60,70],
        c=['r','g','b','k','y'])
#Displaying graph
plt.show()
```

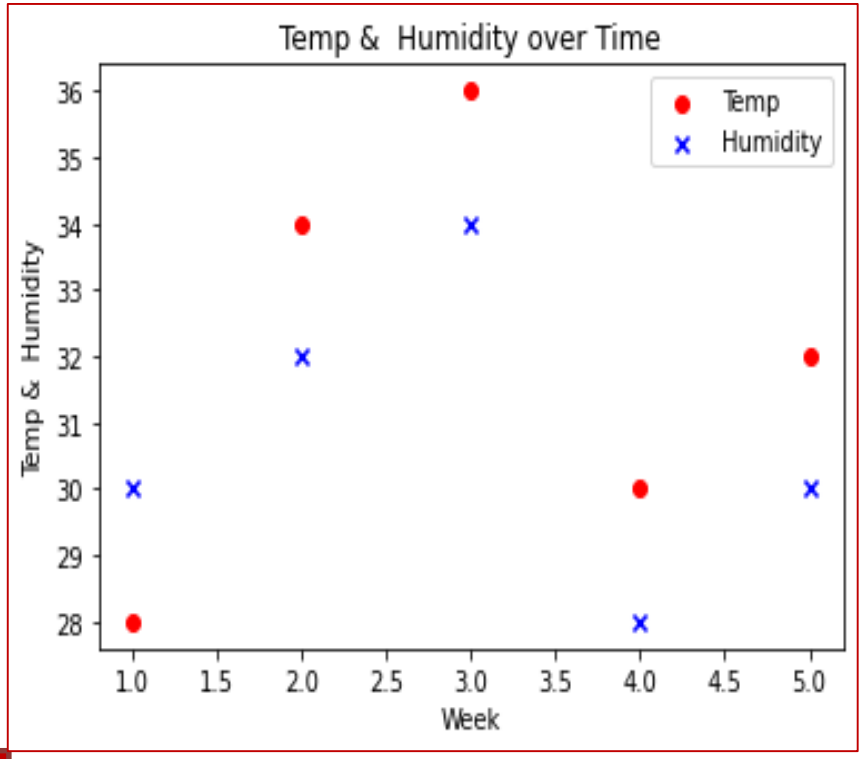


Customizing Scatter Chart: scatter()

- Example of scatter chart:

Scatter graph depicting relation in week, temperature and humidity data

```
import matplotlib .pyplot as plt
w= [1,2,3,4,5]
t= [28,34,36,30,32]
h= [30,32,34,28,30]
#  plotting graph
plt .scatter(w,t, marker='o',c='r',label="Temp")
plt .scatter(w,h,marker='x',c='b', label="Humidity")
plt .xlabel("Week")
plt .ylabel("Temp &  Humidity")
plt.title("Temp &  Humidity over Time")
plt.legend(loc="upper right")
plt.show()
```



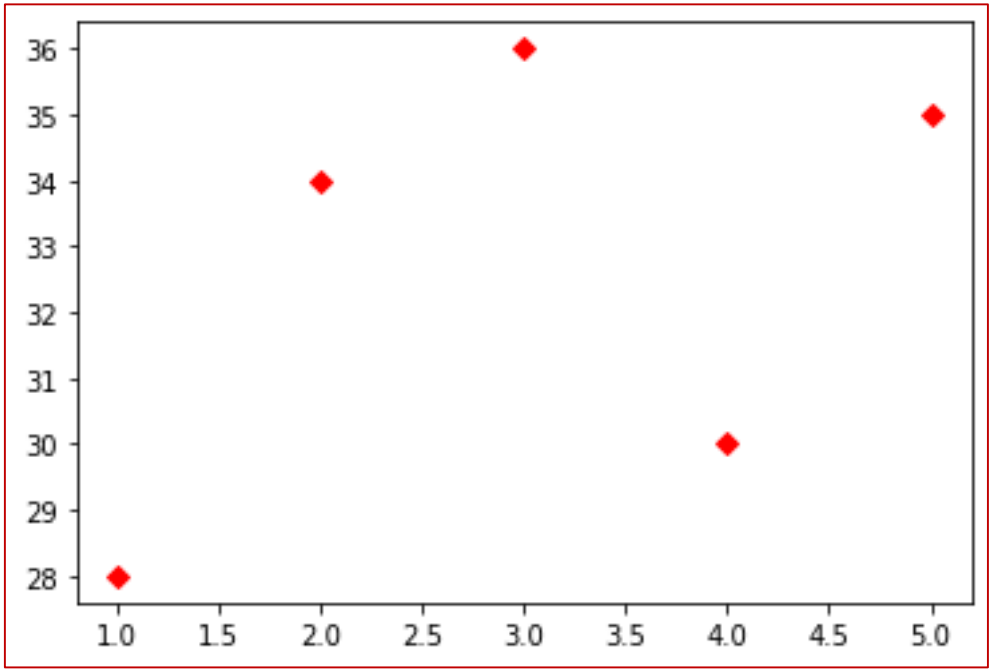
If you want to display legend with plot, then define text for legend with label option in scatter() function.

Creating Scatter Chart through plot()

As we ca n see that Scatter chart is similar to line chart except that in line chart all marker points are connected through a line. So, if we disconnect all marker points than we can get scatter chart through plot() function too.

In plot () function, if we give **marker type with line color code, then it create scatter graph**. For example, 'rd' will give red color diamond shaped marker points.

```
#creating Scatter chart with plot()
import matplotlib.pyplot as plt
w=[1,2,3,4,5]
t=[28,34,36,30,35]
#plotting graph
plt.plot(w,t,'rD')
#Displaying graph
plt.show()
```



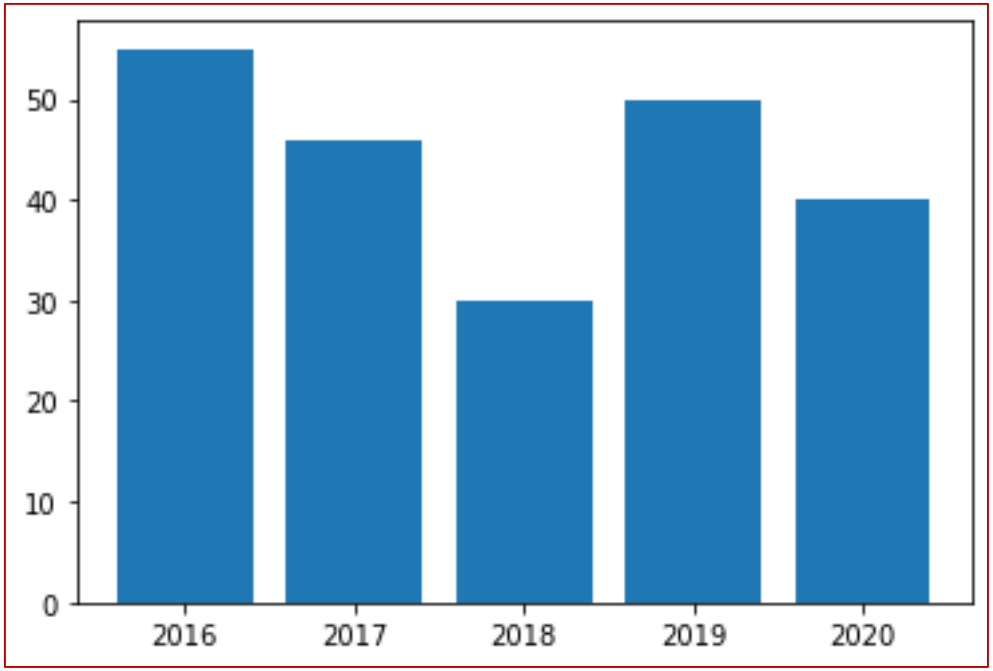
Bar chart: bar()

❖ bar() : Creates bar graph for given x and y values.

Bar chart is a graphical display of data through colored bars of different height/length. It can be drawn horizontally or vertically. Pyplot's barh() function can be used to plot horizontal bar graph.

By default bar() creates bar of 0.8 width in random color.

```
#creating simple bar graph
#through List of values
import matplotlib.pyplot as plt
x= [2016,2017,2018,2019,2020]
y= [55,46,30,50,40]
#  plotting graph
plt.bar(x,y)
#Displaying graph
plt.show()
```



Customizing Bar Chart: bar()

Pyplot's bar() function offers various setting to customize bar chart like width and color of bars. You can also define separate color and width size for each bar by providing width and color list.

```
<pyplotobj>.bar(<X-Value>,<Y-Value>[,width=<size(s)>]  [color=
<color code(s)>] [label=<text>])
```

X-Value: Dataset for X-Axis. Dataset may be 1-D Array, List, Series or Column of Data frame.

Y-Value: Dataset for Y-Axis. Number of values for X-Axis and Y-Axis should be equal.

width=<size(s) > : Defines width of bars in number. You can also define a list of sizes for each bar, if required.

color =<color code(s)>: Defines color for bars. You can also define a list of separate color code for each bar.

Label= <text>: Defines label text for legend.

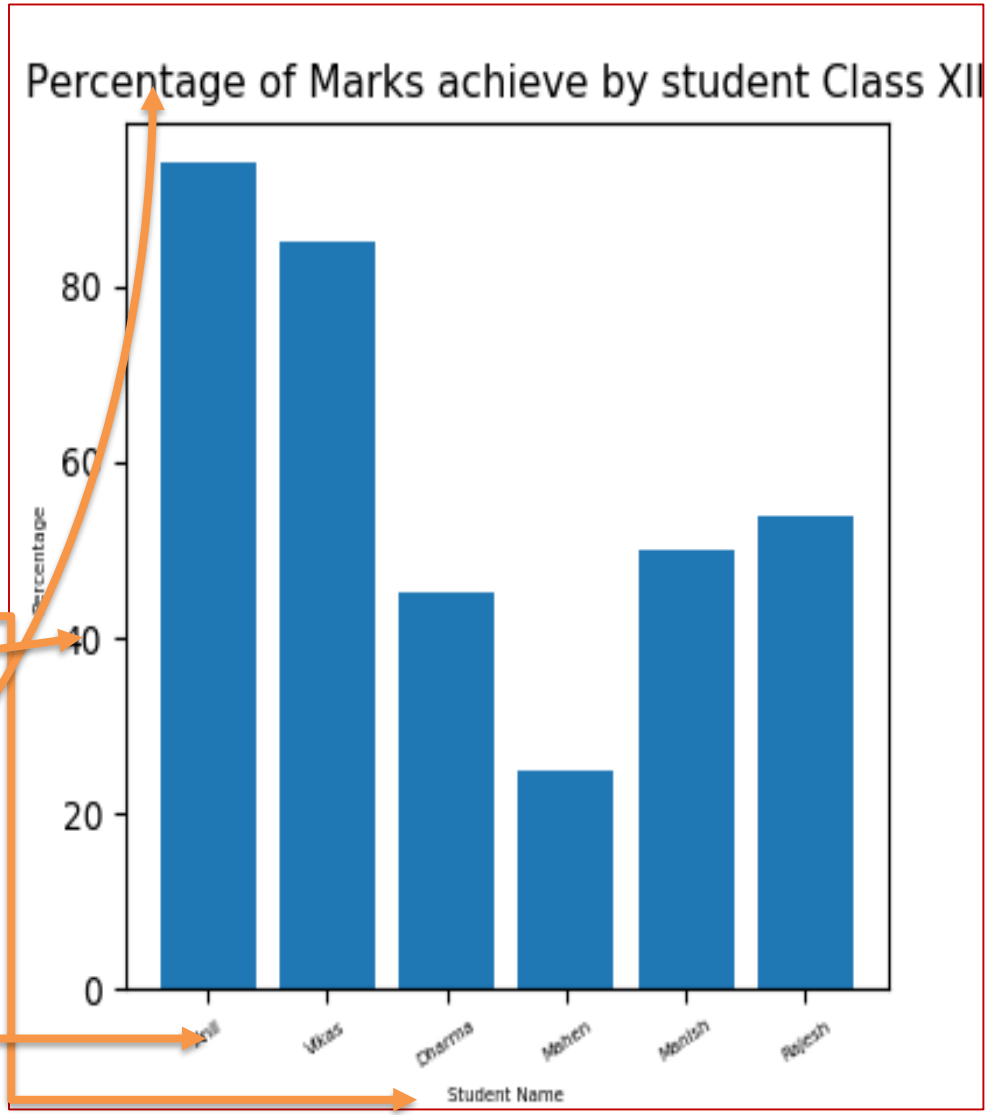
Color code characters can be used as per plot() method

Customizing Bar Chart: bar()

Plot bar graphs

e.g program

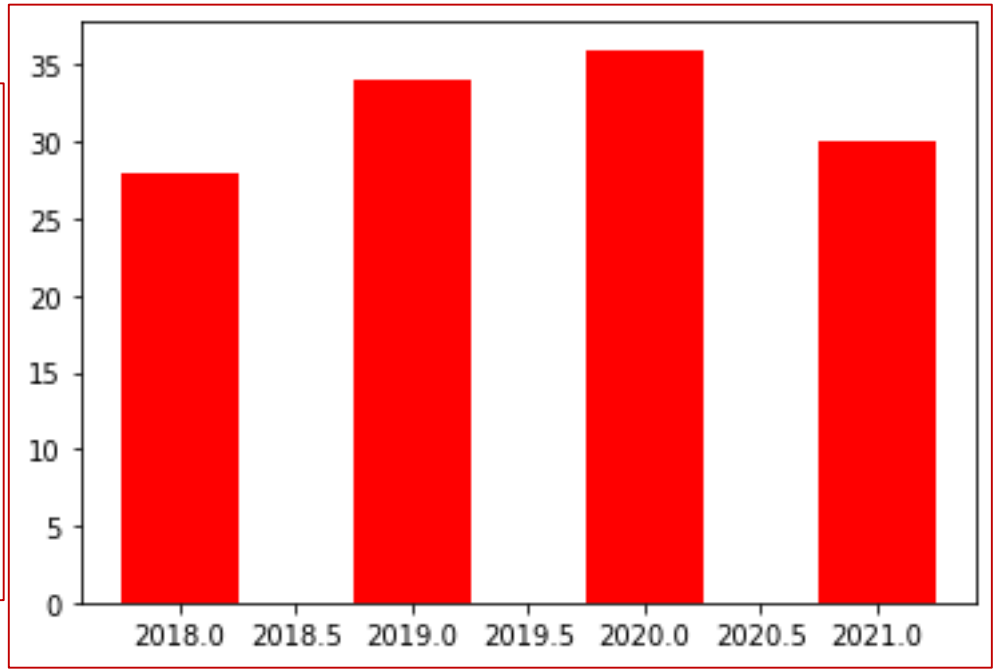
```
import matplotlib.pyplot as plt
import numpy as np
label = ['Anil', 'Vikas', 'Dharma', 'Mahen',
'Manish', 'Rajesh']
per = [94,85,45,25,50,54]
index = np.arange(len(label))
plt.bar(index, per)
plt.xlabel('Student Name', fontsize=5)
plt.ylabel('Percentage', fontsize=5)
plt.xticks(index, label, fontsize=5,
rotation=30)
plt.title('Percentage of Marks achieve by
student Class XII')
plt.show()
#Note – use barh () for horizontal bars
```



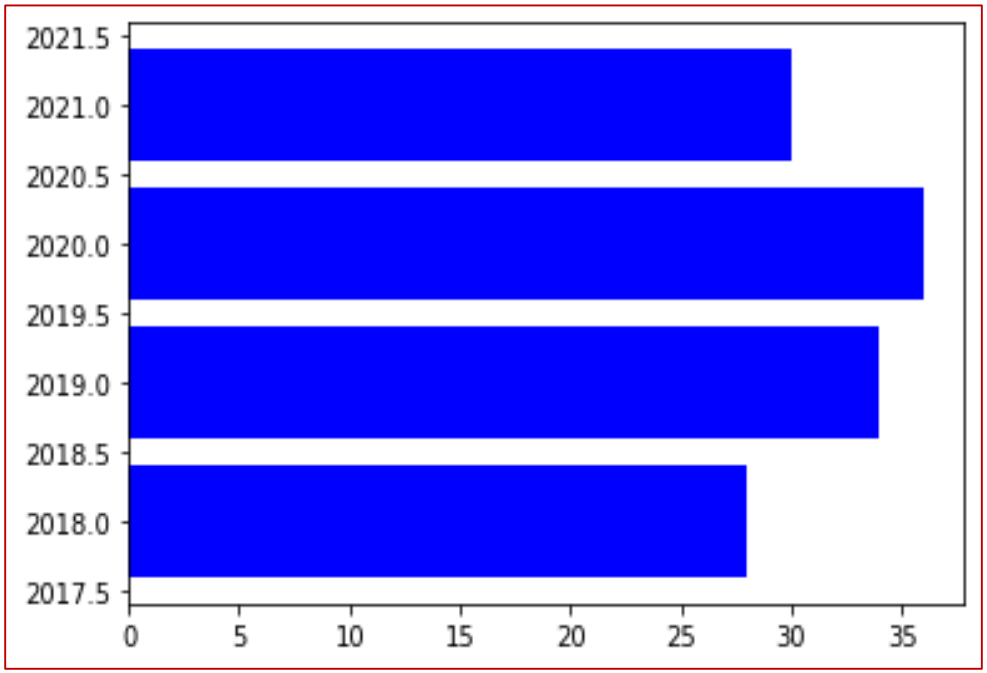
Customizing bar Chart: bar()

❖ Example of bar chart:

```
# bar chart for year & pass % students
import matplotlib.pyplot as plt
year=[2018,2019,2020,2021]
pas= [28,34,36,30]
#  plotting vertical graph
plt.bar(year,pas, width=0.5, color='r')
plt.show()
```




```
# bar chart for year & pass % students
import matplotlib.pyplot as plt
year=[2018,2019,2020,2021]
pas= [28,34,36,30]
#  plotting vertical graph
plt.barh(year,pas, color='b')
plt.show()
```

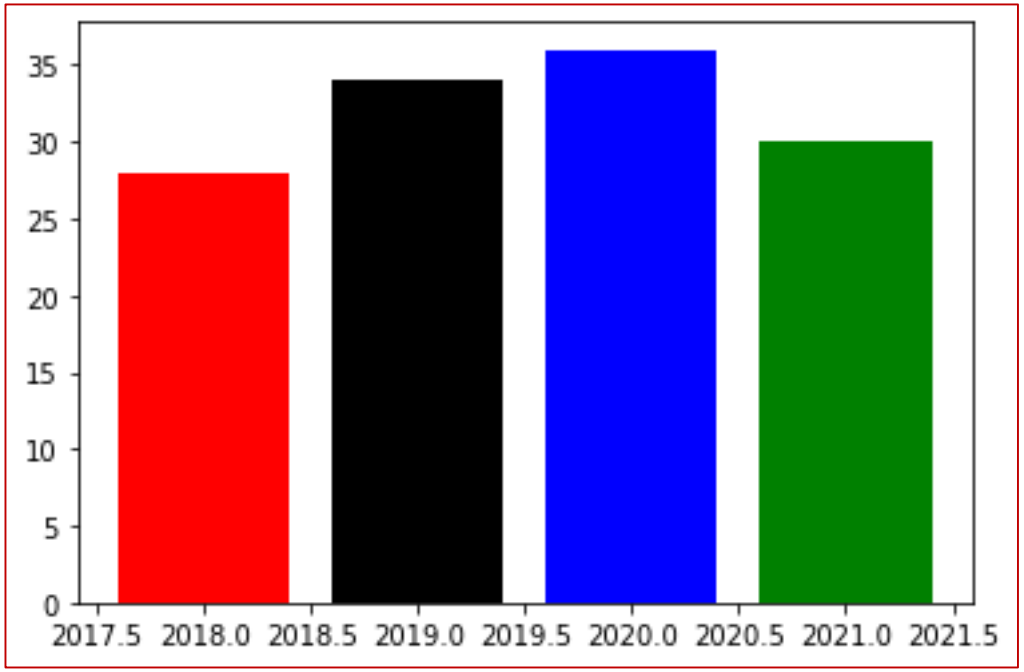


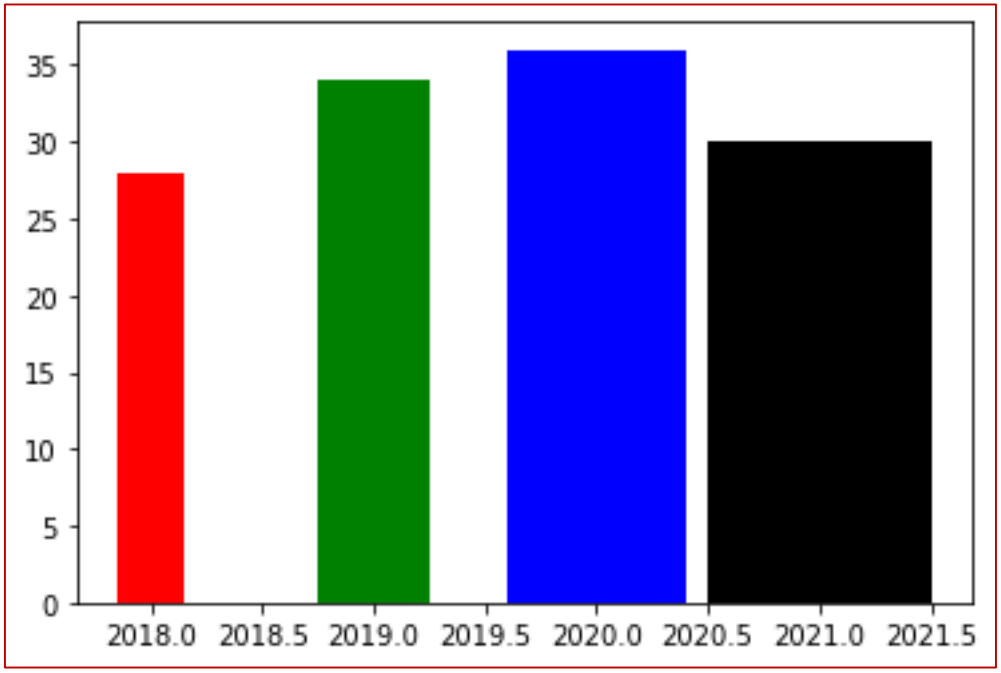
Note that barh() uses **height** parameter in place of **width.**

Customizing bar Chart: bar()

❖ Graph with varying color and width

```
# bar chart with varying colors
import matplotlib.pyplot as plt
year= [2018,2019,2020,2021]
pas= [28,34,36,30]
# plotting graph
plt.bar(year,pas, color= ['r','k','b','g'])
plt .show()
```





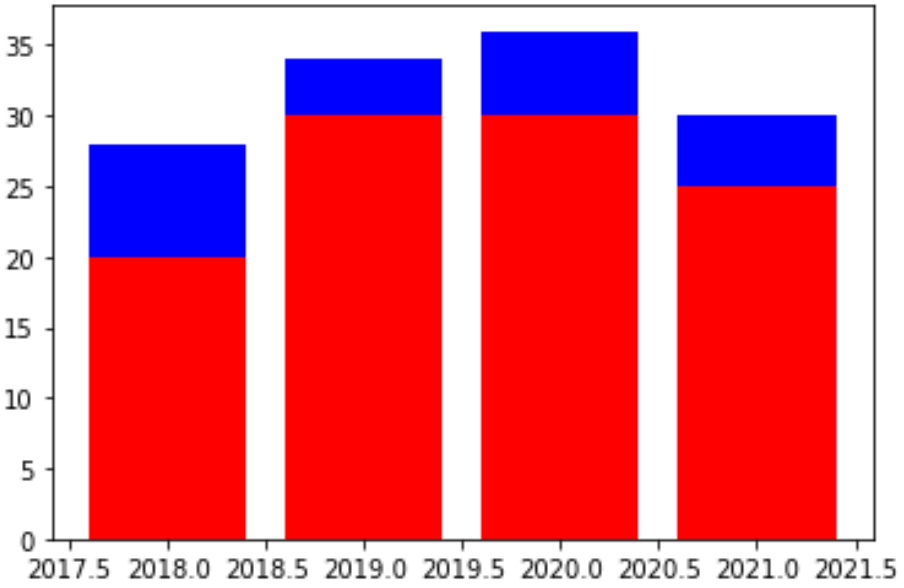

```
#  bar chart with varying color and size
import matplotlib.pyplot as plt
year= [2018,2019,2020,2021]
pas= [28,34,36,30]
#  plotting graph
plt.bar(year,pas,width= [0.3,0.5,0.8,1],
        color=['r','g','b','k'])
plt.show()
```

Customizing bar Chart: bar()

- Creating multiple bar Graph

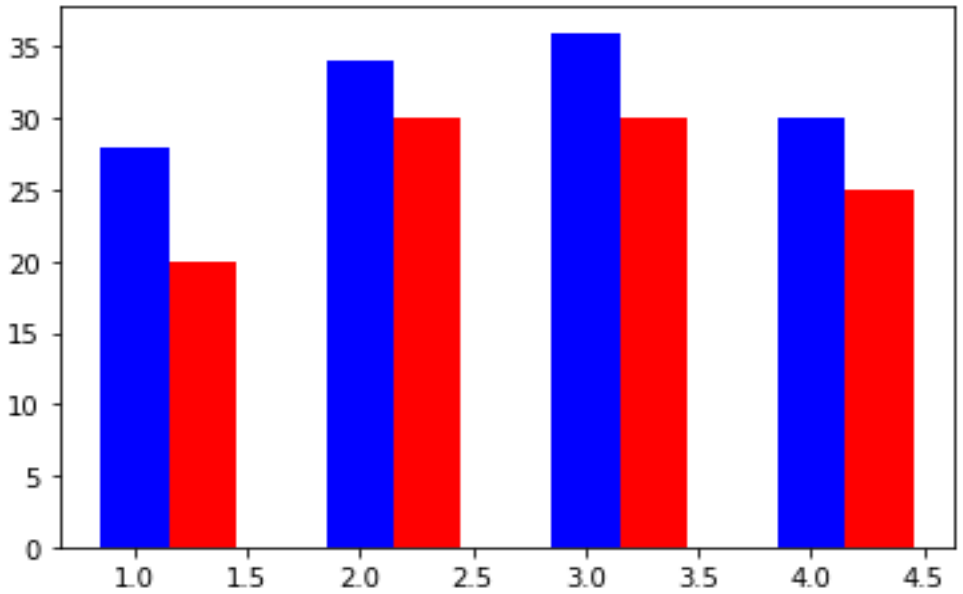
Same X value will produce stacked bar

```
# bar chart with multiple bar
import matplotlib.pyplot as plt
year= [2018,2019,2020,2021]
male= [28,34,36,30]
female= [20,30,30,25]
plt.bar(year,male, color='b')
plt.bar(year,female, color='r')
plt.show()
```



X-value is adjusted with size of bars

```
# bar chart with multiple bar
import matplotlib.pyplot as plt
import numpy as np
x=np.array([1,2,3,4])
male=[28,34,36,30]
female=[20,30,30,25]
plt.bar(x, male, width=0.3,color='b')
plt.bar(x+0.3, female, width=0.3,color='r')
plt.show()
```



Customizing bar Chart: bar()

Setting Xlimits and Ylimits

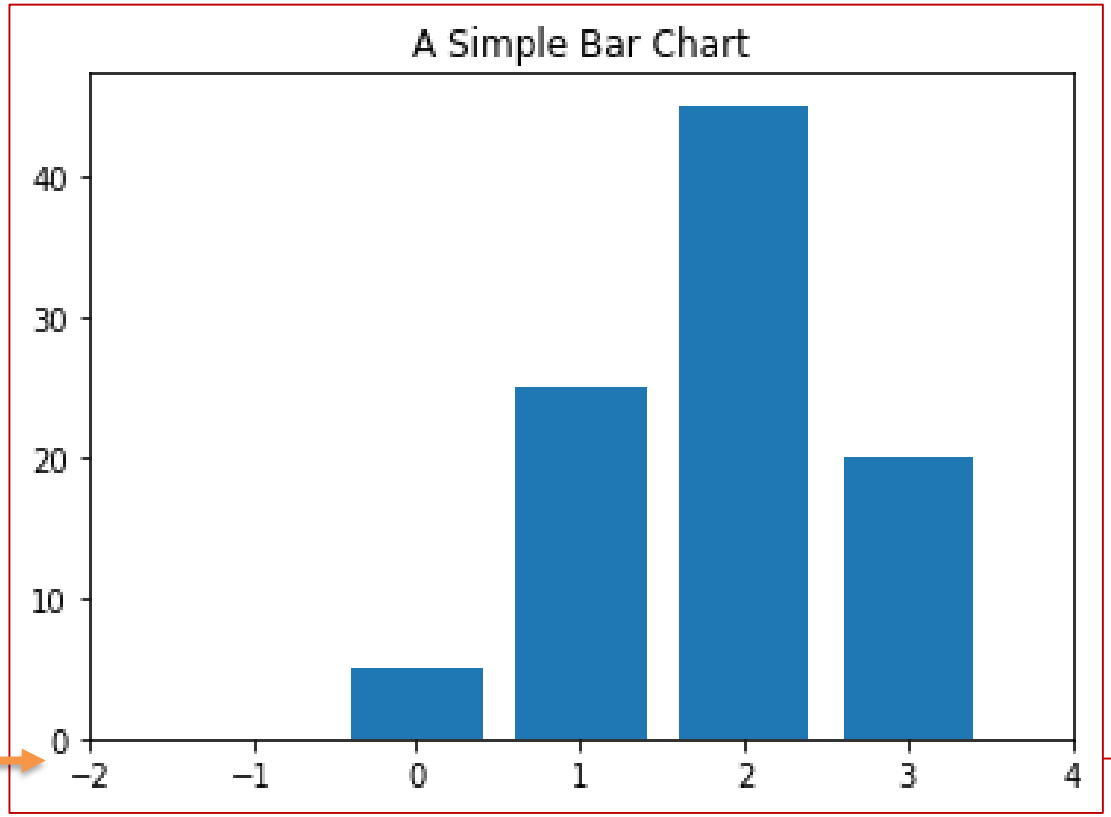
With the help of xlim() and ylim() functions to set limits for X-axis and Y-axis respectively:-

Syntax:

pyplot.xlim(<xmin>,<xmax>)

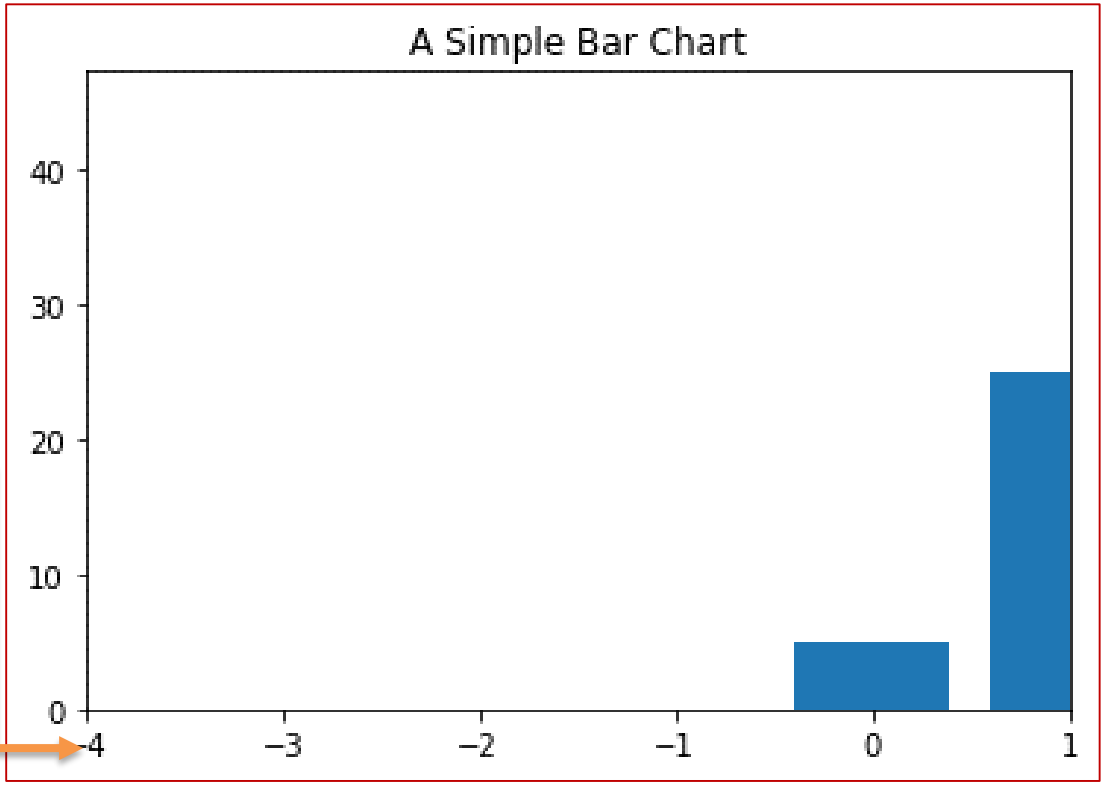
pyplot.ylim(<ymin>,<ymax>)

```
import matplotlib.pyplot as plt
import numpy as np
x=np.arange(4)
y=[5.,25.,45.,20.]
plt.xlim(-2.0,4.0)
plt.bar(x,y)
plt.title("A Simple Bar Chart")
plt.show()
```

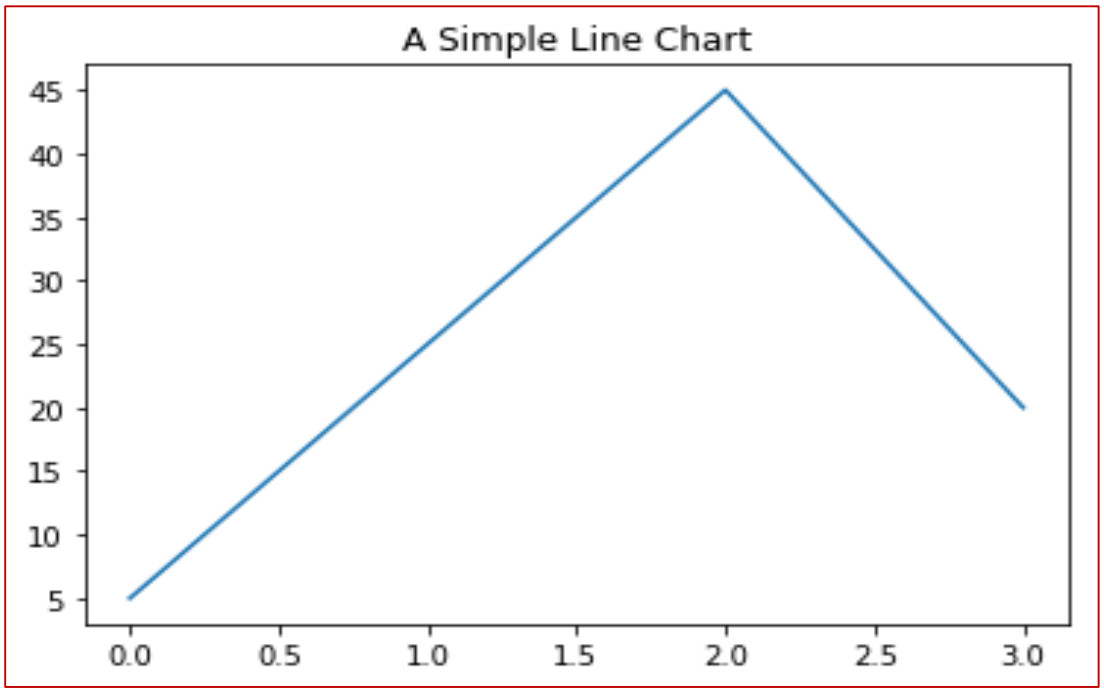


Customizing bar Chart: bar()

```
import matplotlib.pyplot as plt
import numpy as np
x=np.arange(4)
y=[5.,25.,45.,20.]
plt.xlim(-4.0,1.0)
plt.bar(x,y)
plt.title("A Simple Bar Chart")
plt.show()
```

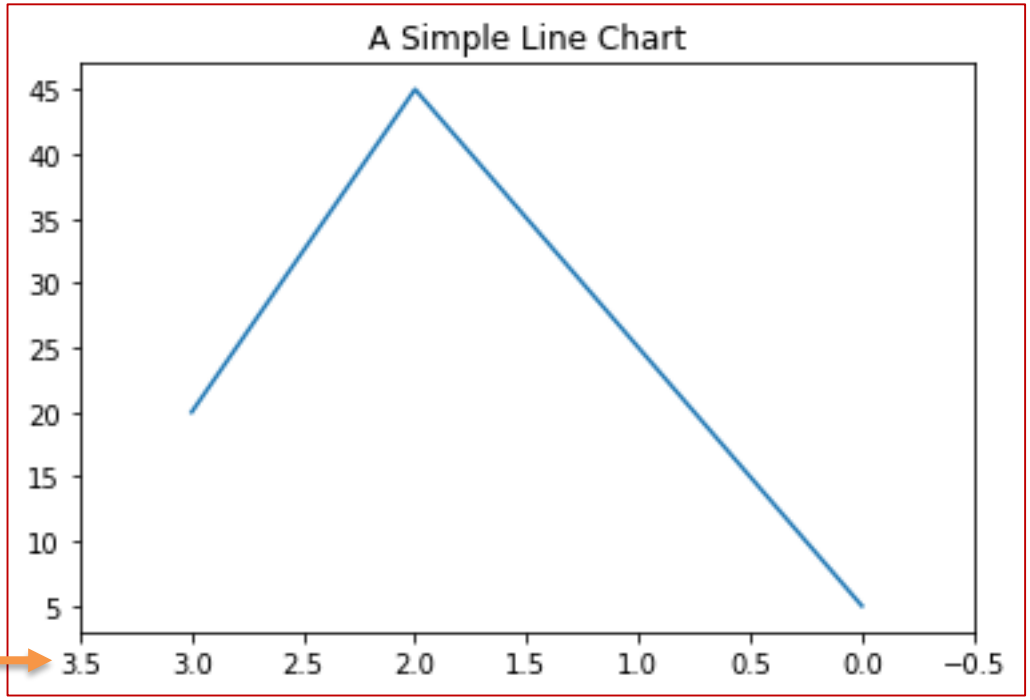



```
import matplotlib.pyplot as plt
x=[0,1,2,3]
y=[5.,25.,45.,20.]
plt.plot(x,y)
plt.title("A Simple Line Chart")
plt.show()
```

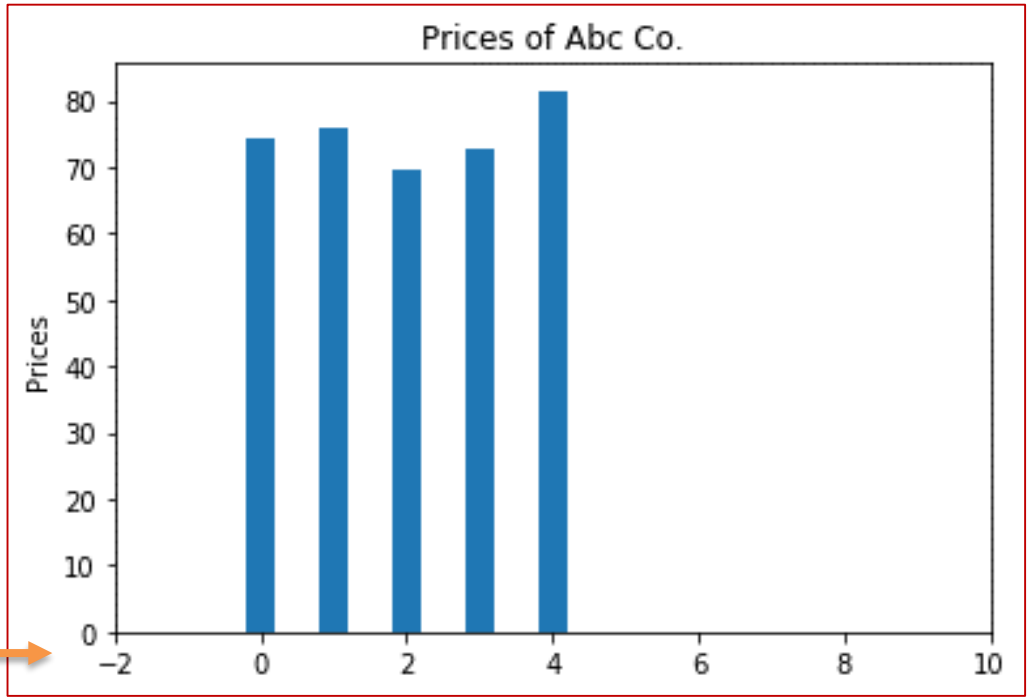


Customizing bar Chart: bar()

```
import matplotlib.pyplot as plt
x=[0,1,2,3]
y=[5.,25.,45.,20.]
plt.xlim(3.5,-0.5)
plt.plot(x,y)
plt.title("A Simple Line Chart")
plt.show()
```




```
import matplotlib.pyplot as plt
pr=[74.25,76.06,69.5,72.55,81.5]
plt.bar(range(len(pr)),pr,width=0.4)
plt.xlim(-2,10)
plt.title("Prices of Abc Co.")
plt.ylabel("Prices")
plt.show()
```



Customizing bar Chart: bar()

Setting Ticks for Axes:

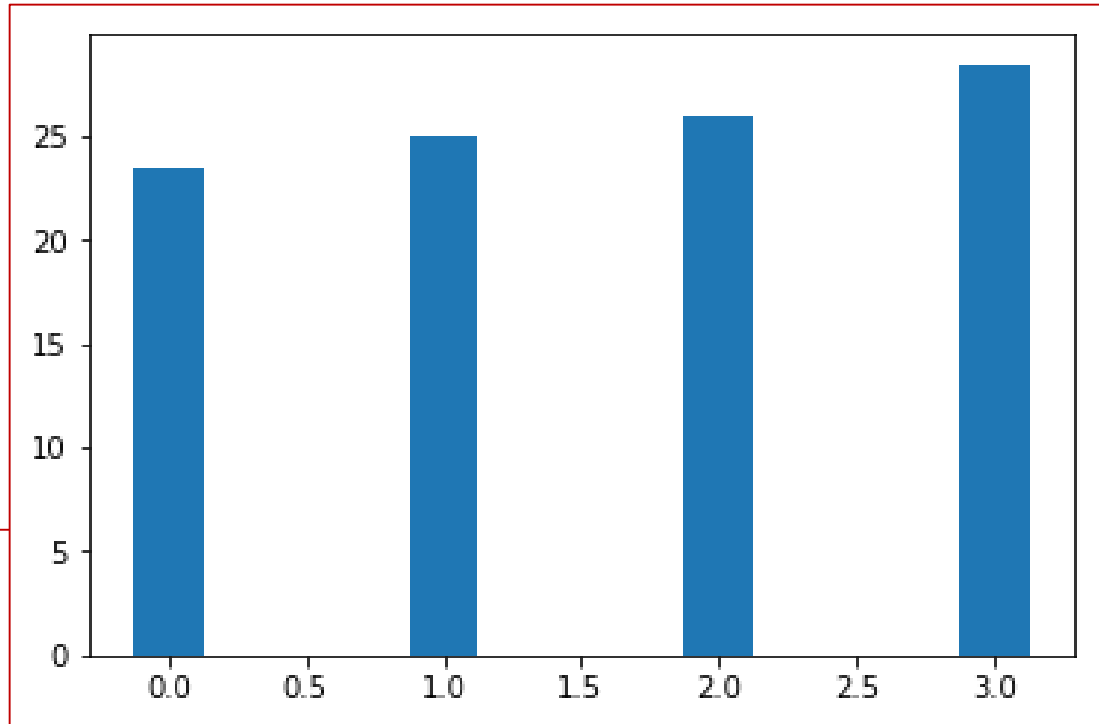
By default PyPlot will automatically decide which data points will have ticks on the axes, but we can also decide which data points will have tick marks on X and Y axes:-

Syntax:

pyplot.xticks(<sequence containing tick data points>,
<Optional sequence containing tick label>)

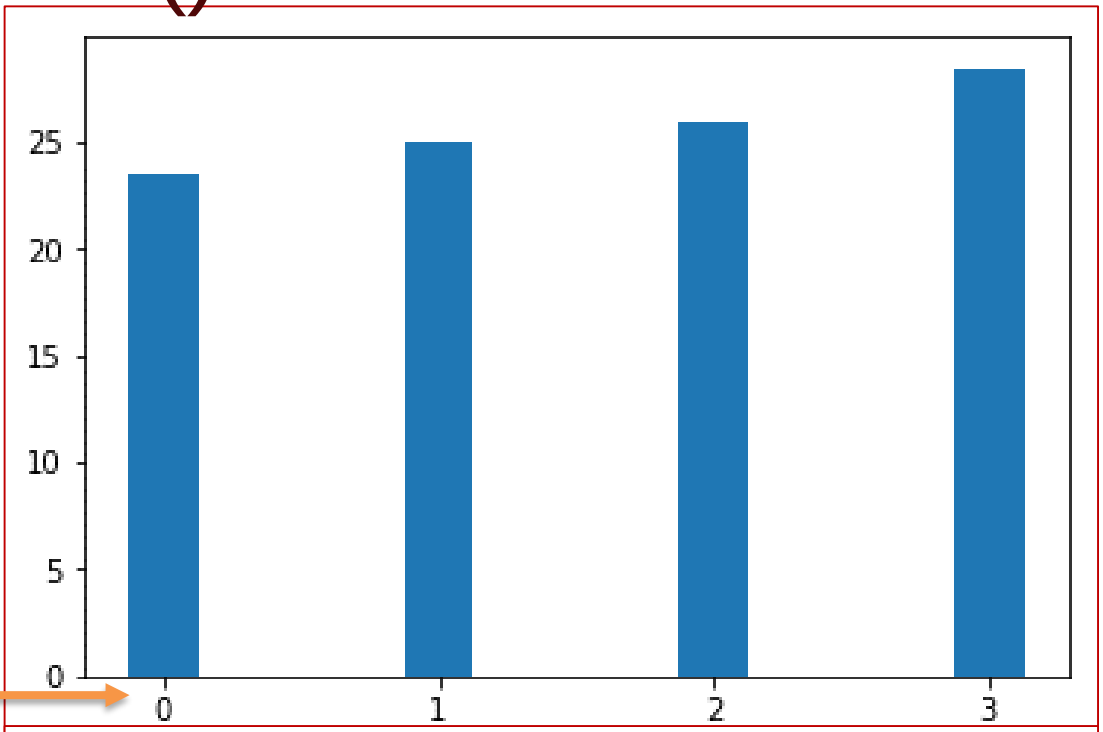
pyplot.yticks(<sequence containing tick data points>,
<Optional sequence containing tick label>)

```
import matplotlib.pyplot as plt
x=range(4)
y=[23.5,25,26,28.5]
plt.bar(x,y,width=0.25)
plt.show()
```

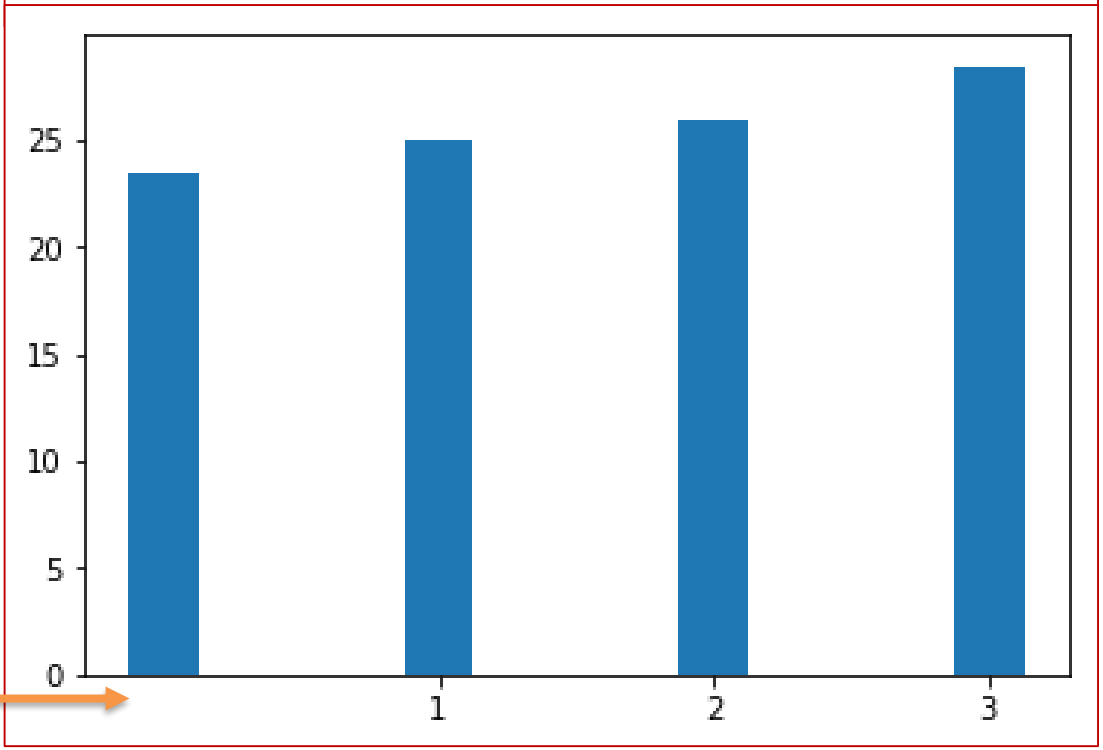


Customizing bar Chart: bar()

```
import matplotlib.pyplot as plt
x=range(4)
y=[23.5,25,26,28.5]
plt.xticks([0,1,2,3])
plt.bar(x,y,width=0.25)
plt.show()
```




```
import matplotlib.pyplot as plt
x=range(4)
y=[23.5,25,26,28.5]
plt.xticks([1,2,3,4])
plt.bar(x,y,width=0.25)
plt.show()
```



Histogram chart:

❖ What is Histogram?

A histogram is graphical visualization of the distribution of numerical Data in terms of frequency count. It represent frequencies as bars based on non overlapping ranges called bins.

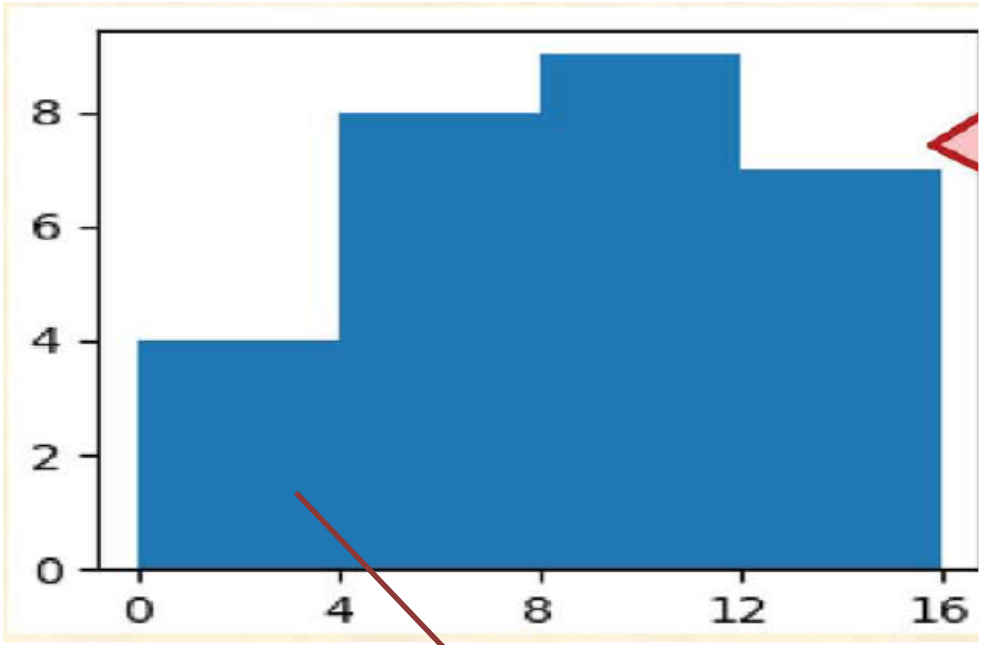
Consider the following collection of age of 28 students of a school in different age-group.

[2,4,5,6,2,6,8,10,12,12,11,10,4,3,2,5,7,9,11,12,13,14,15,14,10,9,7,9]

Suppose, above data set is classified in the following age groups called bins.

Age group	No.of students
0-4	4
4-8	8
8-12	9
12-16	7

Total 28



Upper age is exclusive in the range except last range.

Bar representing frequency count for each range/Bin

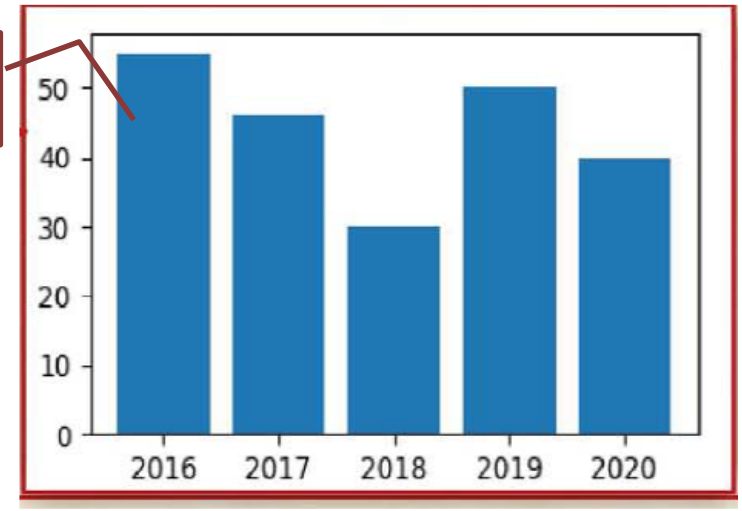
Histogram chart:

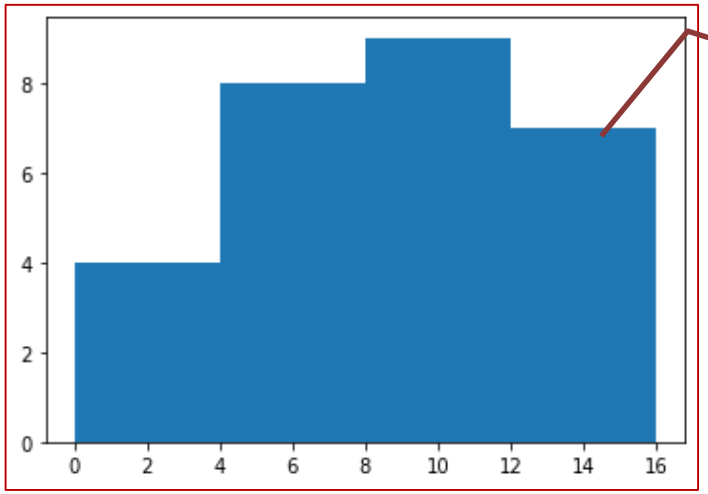
❖ How Histogram is different from Bar chart?

Visually, histogram is much similar to Bar graph but histogram represent different analysis than Bar chart. Some main differences are as follows-

Bar Chart	Histogram
Bar graph is a representation of data that uses bars to compare **categorical data** values.	Histogram is representation, that displays distribution (frequency) of **quantitative data** through bars.
In Bar chart, bars are not closed i.e. not continuous.	In histogram, bars are close to each other, since, it uses continuous ranges.
Data values are considered as individual entities.	Data values are grouped together, as per defined ranges.

Bar Chart





Histogram

Histogram chart: hist()

❖ hist( ): Creates histogram for given data set

The hist() function of Pyplot creates histogram (frequency count chart) **Bins** can be considered as consecutive, non-overlapping intervals of ranges. If can be a number or range with lower and upper limit.

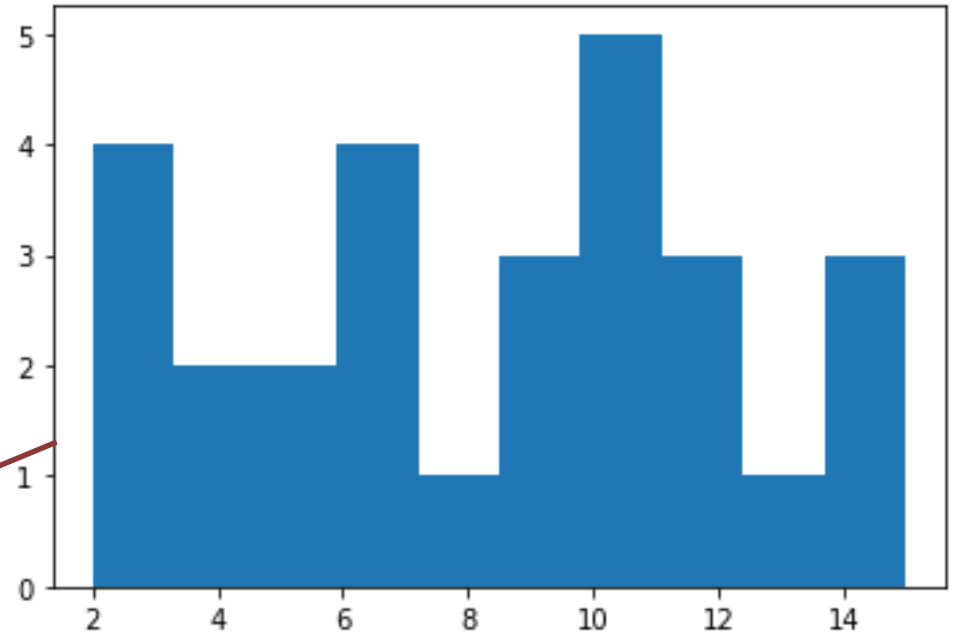
For example, if bin is 5 then it shows that all data to classified and counted in 5 equal groups.

But if it is given as list of numbers, then it shows ranges. bins=[0,4,8,12, 16] shows 0-3,4-7,8-11,12-16 ranges.

By default it uses 10 bins with auto calculated ranges. However you can defines number of bins or ranges as per need.

```
# creating Histogram (List of values)
import matplotlib.pyplot as plt
data= [2,4,5,6,2,6,8,10,12,12,11,
      10,4,3,2,5,7,9,11,12,13,14,
      15,14,10,9,7,9]
#plotting histogram
plt.hist(data)
#Displaying graph
plt.show()
```

Histogram with default bins 10



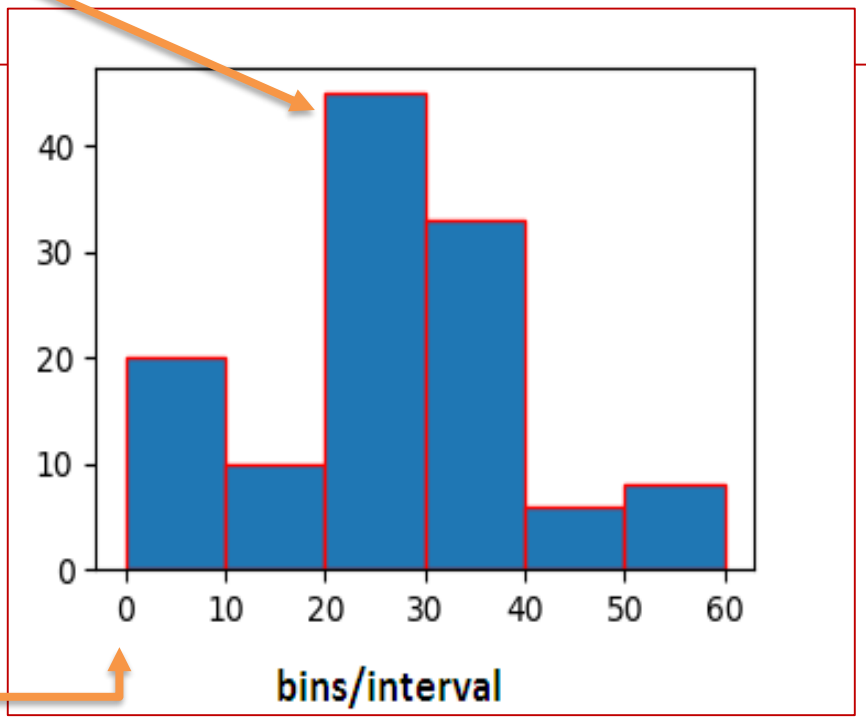
Histogram chart: hist()

Matplotlib –Histogram

Histogram in Python –

```
import matplotlib.pyplot as plt
data = [1,11,21,31,41]
plt.hist([5,15,25,35,45, 55], bins=[0,10,20,30,40,50, 60],
weights=[20,10,45,33,6,8],
edgecolor="red")
plt.show()
```

#first argument of hist() method is position (x,y Coordinate) of weight, where weight is to be displayed. No of coordinates must match with No of weight otherwise error will generate
#Second argument is interval
#Third argument is weight for bars



Customizing Histogram Chart: hist()

Pyplot's hist() function offers options to customize histogram like bins or ranges, histogram types and orientation etc.

```
<pyplot obj>.hist(< Data-Values > [, bins = <Number/ list of numbers>]
[histtype= <type> ] [cumulative = <True/ False>] [orientation=
<'horizontal'|'vertical'>])
```

Data-Value: Dataset for the chart. Dataset may be **1-D Array, List, Series or Column of Dataf rame.**

bins= <number(s)> : Defines the number or bins (in number) and range if given as list of numbers.

histtype= <type>: Defines the type of plot. Type may be bar, barstacked, step and stepfilled. Default is bar type. The barstacked is used for multiple data sets in which one is stacked on top of other.

cumulative= <T/ F>: Creates cumulative graph if True. Default is false.

orientation = <'horizontal'|'vertical'>: Defines orientation of plot as horizontal bars or vertical bars.

You can use xticks() method to show defined ranges on X-Axis.

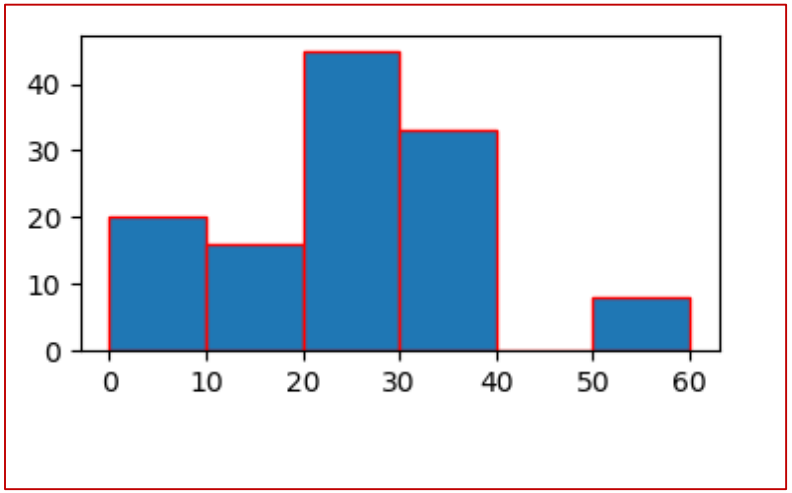
Customizing Histogram Chart: hist()

Histogram in Python –

For better understading we develop the same program with minor change .

```
import numpy as np
import matplotlib.pyplot as
plt
data = [1,11,21,31,41]
plt.hist([5,15,25,35,15, 55], bins=[0,10,20,30,40,50, 60],
weights=[20,10,45,33,6,8],edgecolor="red")
plt.show()
```

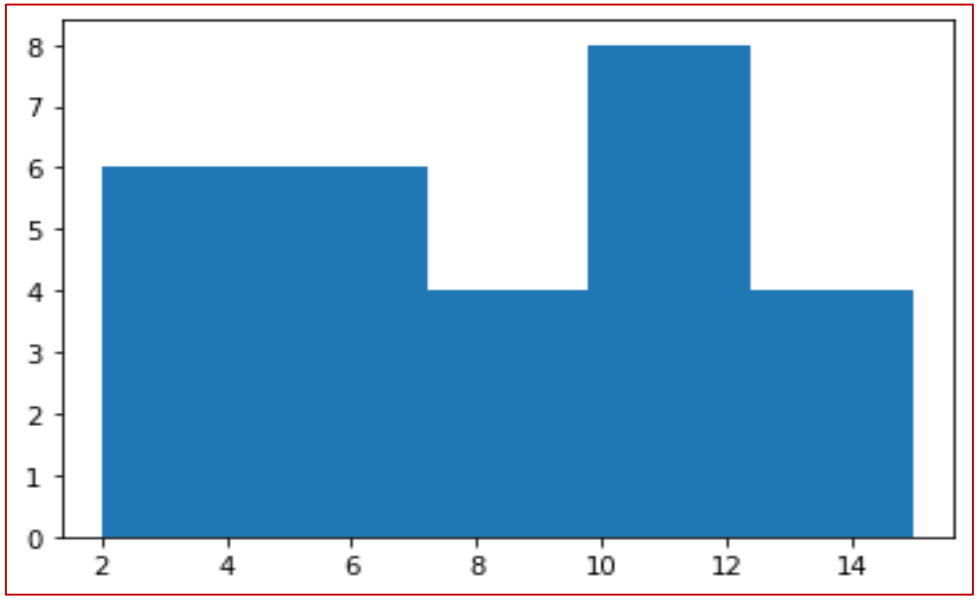
at interval (bin) 40 to 50 no bar because we have not mentioned position from 40 to 50 in first argument(list) of hist method. Where as in interval 10 to 20 width is being Displayed as 16 (10+6 both weights are added) because 15 is twice In first argument.

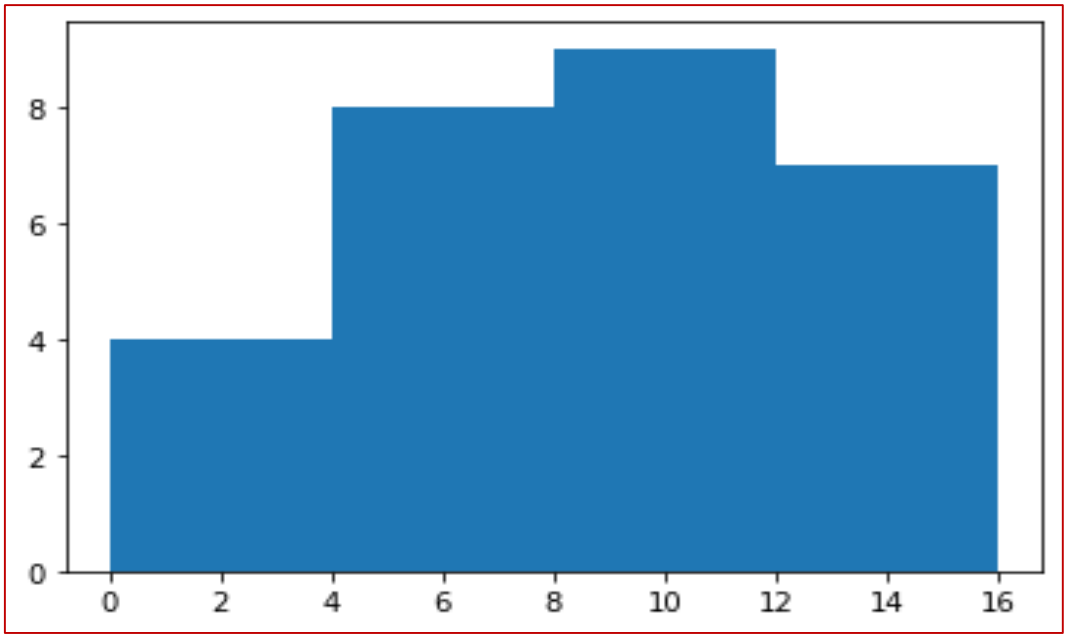


Customizing Histogram Chart: hist()

- Example of Histogram chart

```
#  Hitogram for age of students
import matplotlib.pyplot as plt
data=[2,4,5,6,2,6,8,10,12,12,11,
      10,4,3,2,5,7,9,11,12,13,14,
      15,14,10,9,7,9]
#  histogram with 5 bins
plt.hist(data, bins=5)
plt.show()
```





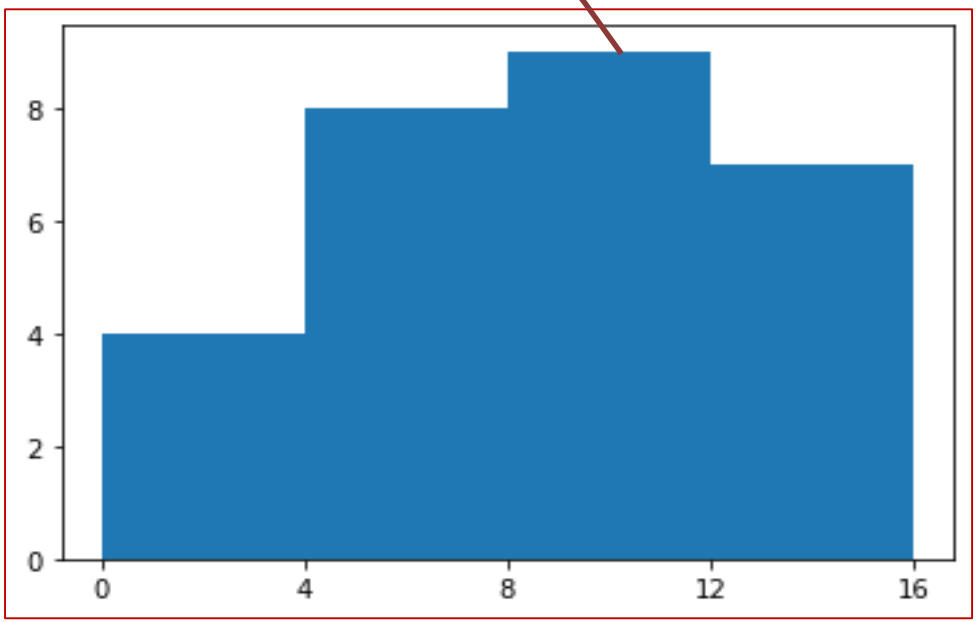

```
# Hitogram for age of students
import matplotlib.pyplot as plt
data= [2,4,5,6,2,6,8,10,12,12,11,
       10,4,3,2,5,7,9,11,12,13,14,
       15,14,10,9,7,9]
#  histogram with 4 ranges
plt.hist(data, bins= [0,4,8,12,16])
plt.show()
```

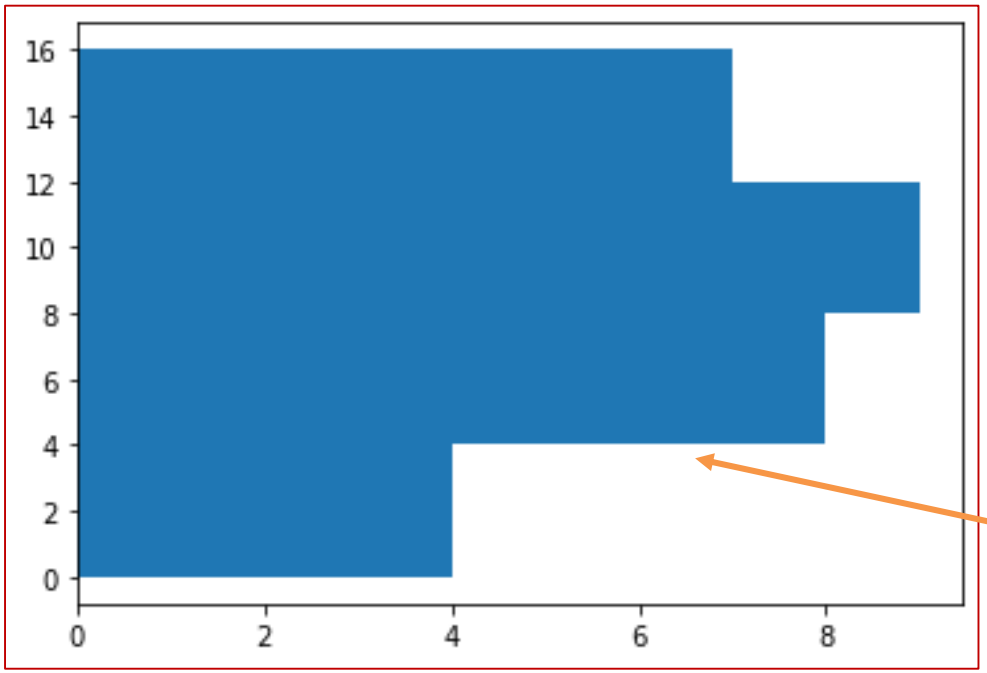
Customizing Histogram Chart: hist()

- Example of Histogram chart:

```
#  Hitogram for age of students
import matplotlib.pyplot as plt
data= [2,4,5,6,2,6,8,10,12,12,11,
       10,4,3,2,5,7,9,11,12,13,14,
       15,14,10,9,7,9]
#  histogram with 4 ranges and x-ticks
plt.hist(data,bins= [0,4,8,12,16])
plt.xticks([0,4,8,12,16])
plt.show()
```

Histogram with x-ticks as defined range



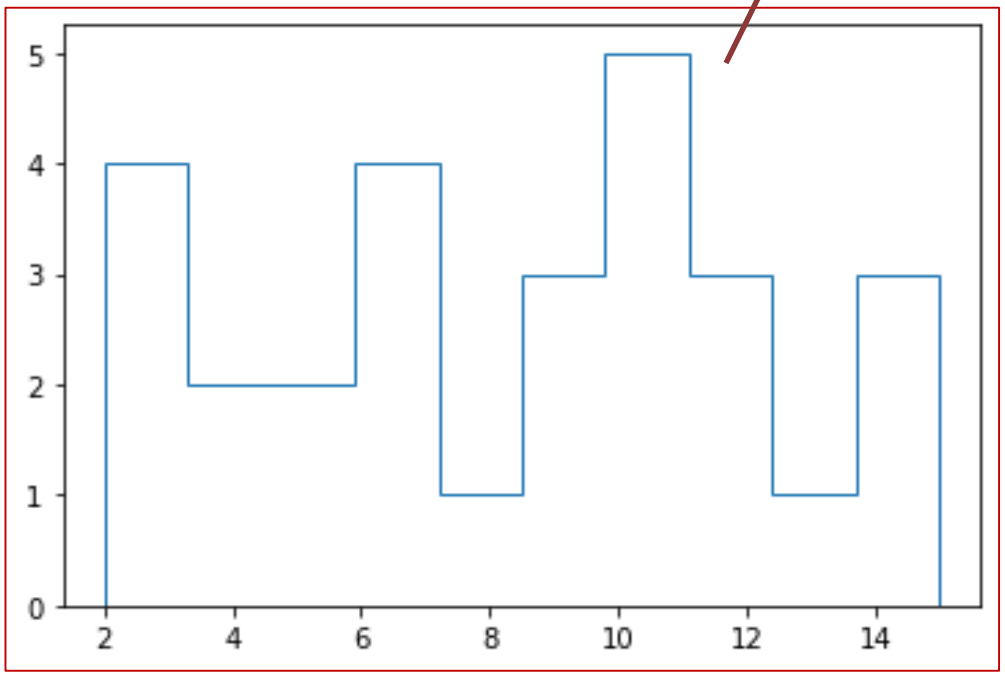



```
#  Hitogram for age of students
import matplotlib.pyplot as plt
data= [2,4,5,6,2,6,8,10,12,12,11,
       10,4,3,2,5,7,9,11,12,13,14,
       15,14,10,9,7,9]
#  histogram with horizontal orientation
plt.hist(data,bins= [0,4,8,12,16],
         orientation='horizontal')
plt.show()
```

Customizing Histogram Chart: hist()

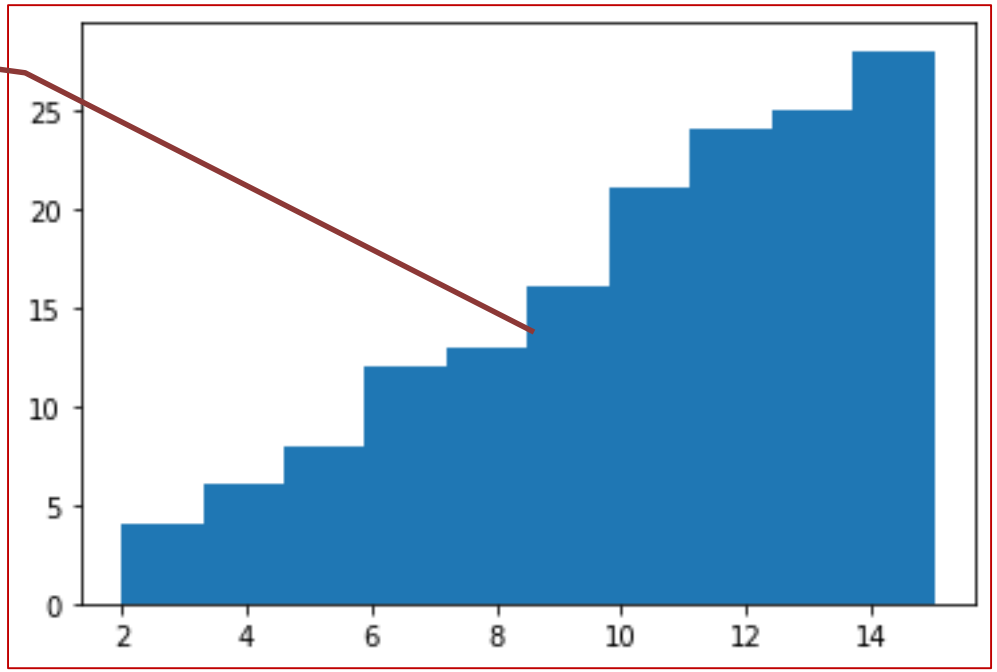
Step type Histogram

```
#  Step Hitogram for age of students
import matplotlib.pyplot as plt
data=[2,4,5,6,2,6,8,10,12,12,11,
      10,4,3,2,5,7,9,11,12,13,14,
      15,14,10,9,7,9]
plt.hist(data,histtype='step')
plt.show()
```



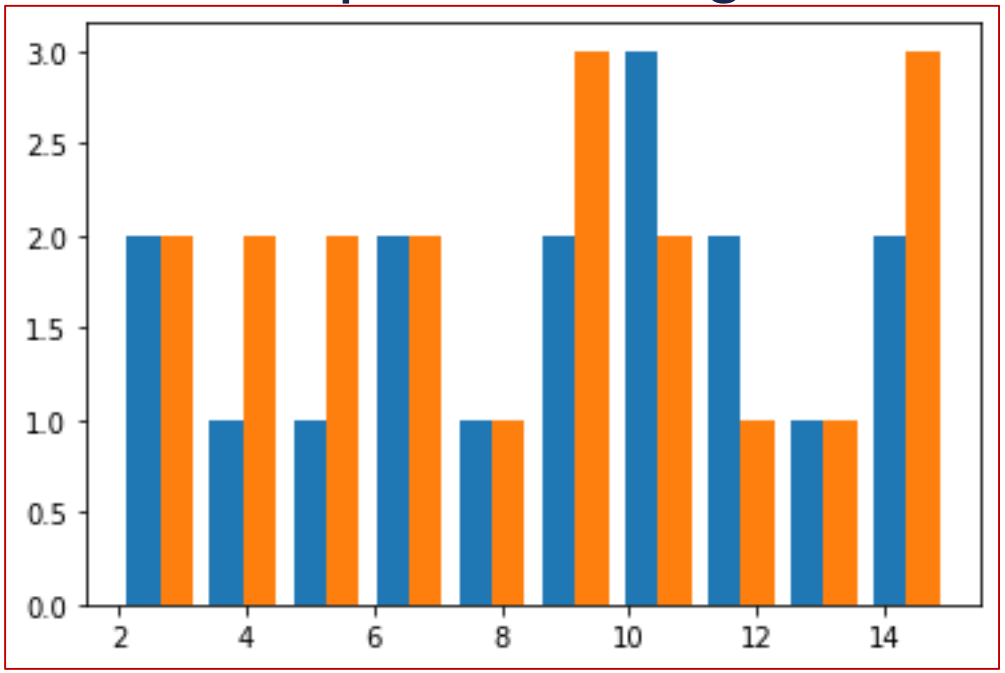
Cumulative Histogram

```
# Cumulative Hitogram for age of
students
import matplotlib.pyplot as plt
data= [2,4,5,6,2,6,8,10,12,12,11,
       10,4,3,2,5,7,9,11,12,13,14,
       15,14,10,9,7,9]
plt.hist(data, cumulative='True')
plt.show()
```



Customizing Histogram Chart: hist()

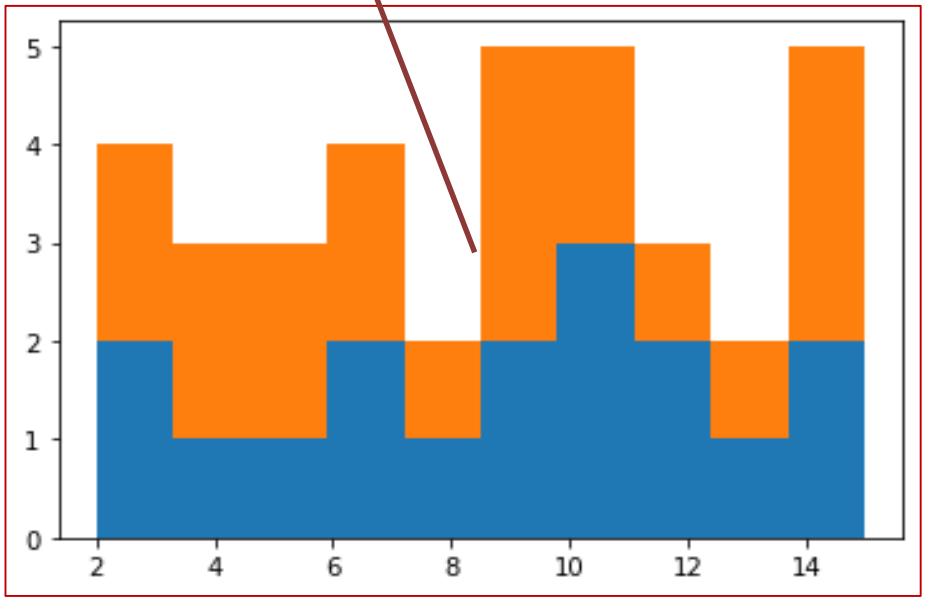
- Example of Histogram chart:




```
# Hitogram for multiple Data set
import matplotlib .pyplot as plt
data1= [2,4,5,6,2,6,8,9,10,12,12,
        9,11,10,13,14,15]
data2= [4,3,2,4,5,7,9,8,11,12,13,
        14,15,14,10,5,9,7,9]
plt.hist([data1,data2], histtype='bar')
plt.show()
```

Stacked bar type Histogram on two data sets.

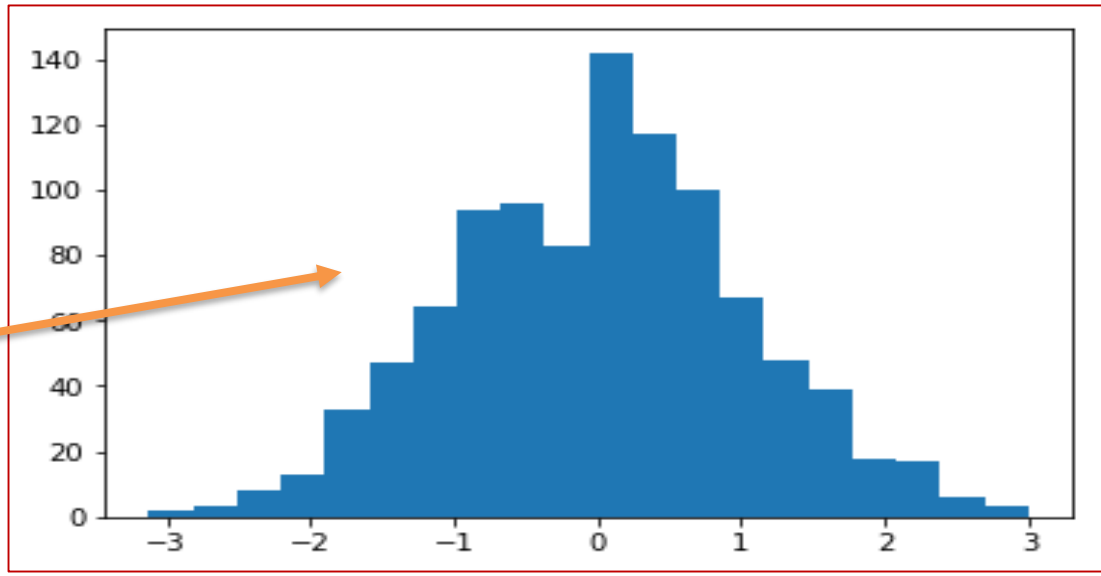
```
#  Hitogram for age of students
import matplotlib.pyplot as plt
data1= [2,4,5,6,2,6,8,9,10,12,
        12,9,11,10,13,14,15]
data2=[4,3,2,4,5,7,9,8,11,12,
        13,14,15,14,10,5,9,7,9]
plt.hist([data1,data2],histtype='barstacked')
plt .show()
```



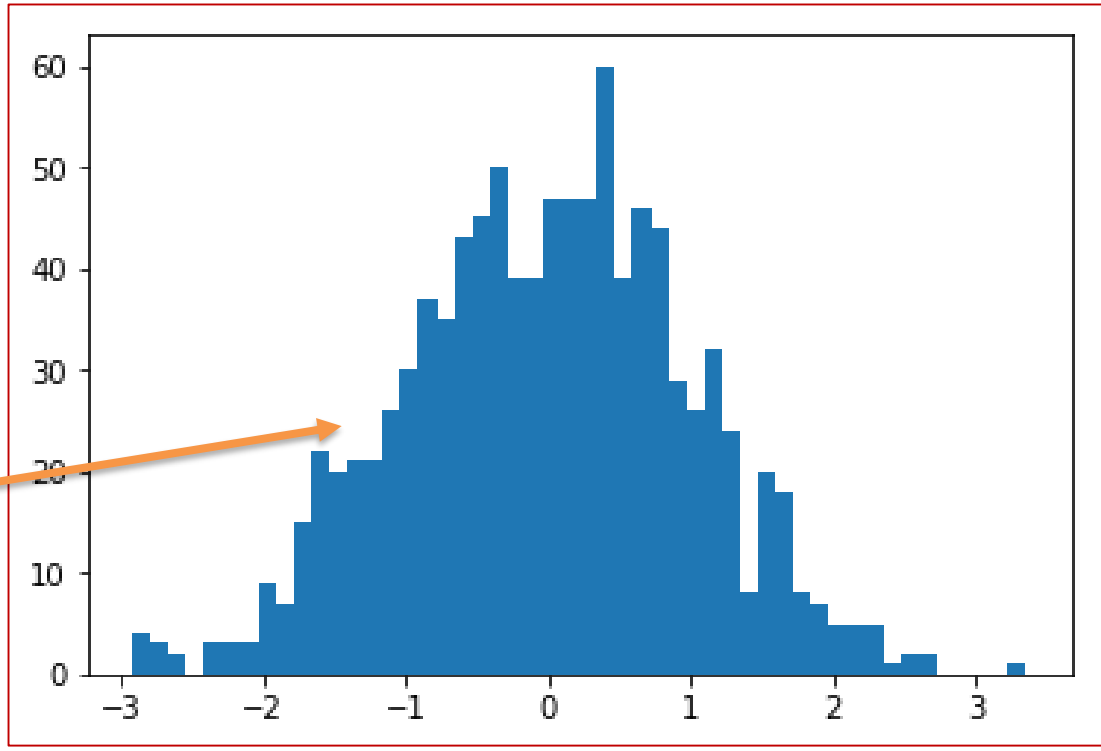
Customizing Histogram Chart: hist()

- Example of Histogram chart:

```
import numpy as np
import matplotlib.pyplot as plt
x=np.random.randn(1000)
plt.hist(x, bins=20)
plt.show()
```



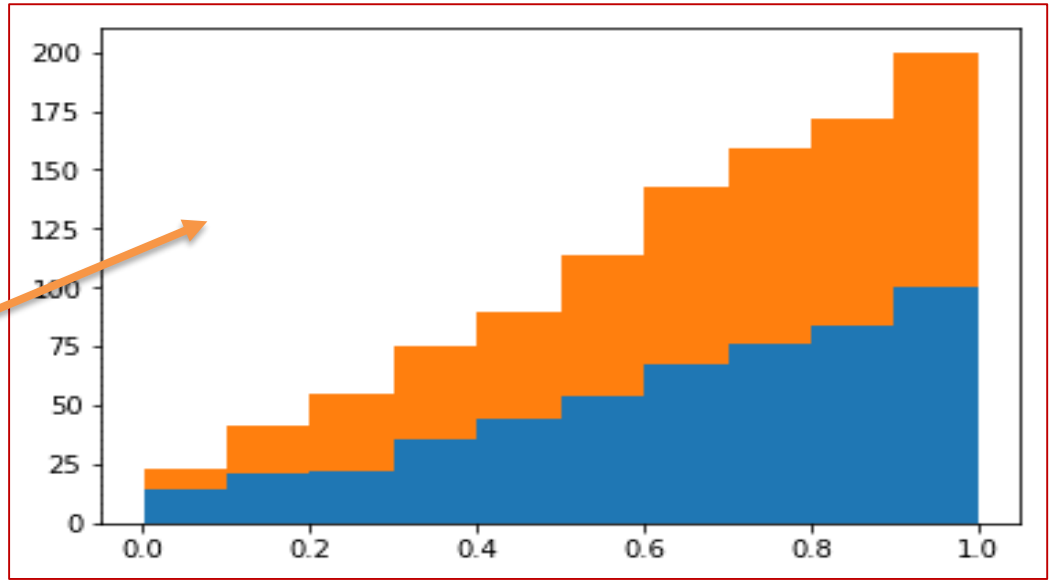

```
import numpy as np
import matplotlib.pyplot as plt
x=np.random.randn(1000)
plt.hist(x, bins=50)
plt.show()
```



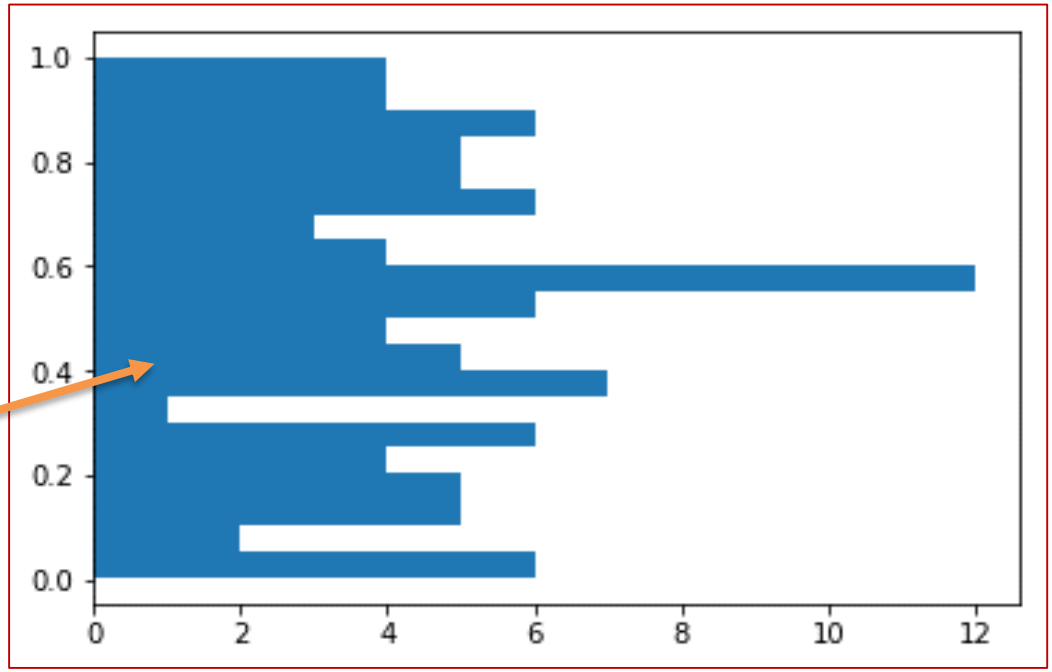
Customizing Histogram Chart: hist()

- Example of Histogram chart:

```
import numpy as np
import matplotlib.pyplot as plt
x=np.random.rand(100)
y=np.random.rand(100)
plt.hist([x,
y],histtype='barstacked'
,cumulative=True)
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt
x=np.random.rand(100)
plt.hist(x,bins=20,
orientation='horizontal')
plt.show()
```



Frequency Polygon chart:

❖ **Frequency Polygon Chart:**

Frequency Polygon is also a frequency distribution chart in which mid point of each range or interval is marked and connected with a line to show comparison of two or more distributions on same axis.

Basically it is extension of Histogram, in which additional line is plotted to connect mid point of each frequency bar. So, Frequency polygon can be visualize as Histogram and Line chart .

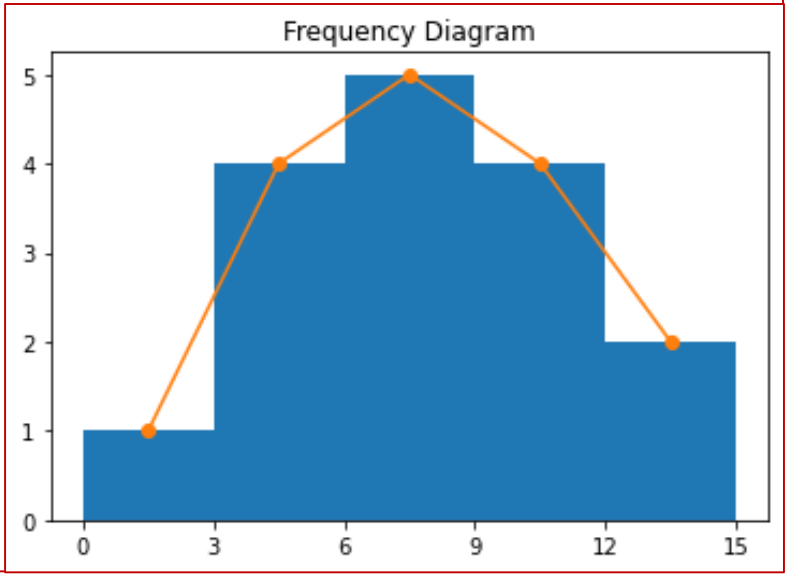
Pyplot does not provide function / method for Frequency polygon, but we can plot frequency polygon by following steps-

1. Plot Histogram on data set.
2. Mark mid points of each bar representing frequency of intervals or bins.
3. Draw line to connect all mid points.
4. You can also connect first and last mid data points with first and last data points on x-scale.

Frequency Polygon chart:

❖ Frequency Polygon Chart- Example:

```
# creating simple pie chart
import matplotlib.pyplot as plt
import numpy as np
data= [2,3,4,5,4,6,7,6,6,6,9,10,12,11, 11,12]
# Plot Histogram and extract (frequency) and edges (bins) on f and edges variable
f,edges,c = plt.hist(data, bins=[0,3,6,9,12,15])
#calculate the midpoint for each bar
mid = 0.5*(edges[1:]+ edges[:-1])
#Plot line chart using mid points with marker
plt.plot(mid,f,marker= 'o')
plt.xticks ([0,3,6,9,12,15])
plt.title('Frequency Diagram')
plt.show()
```



If we trace the values of f, edges and mid variable, then we will get the following array of values.

```
print (f)
print (edges)
print (edges[1:])
print (edges[:-1])
print(mid)
```

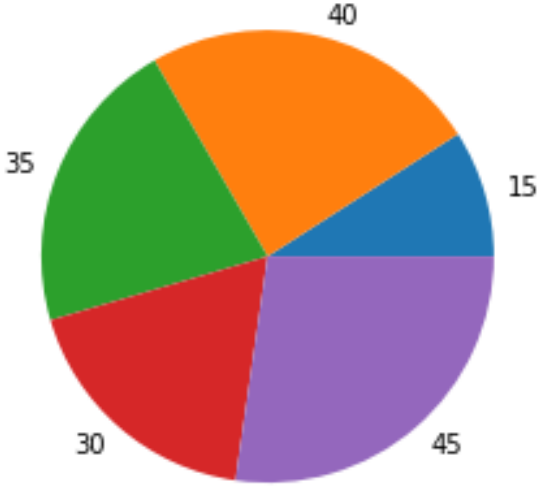
```
[   1.   4.   5 .   4.   2 .   ]
[   0    3    6    9    12   15   ]
[   3    6    9    12   15   ]
[   0    3    6   9     12   ]
[   1.5  4 .5  7.5   10.5  13.5 ]
```

Pie chart: pie()

❖ **pie( ): Creates pie chart for given data set.**

Pie chart is a circle chart in which area of whole circle is divided into sectors or slices to represent a part of the whole in percentage (%). Pie chart takes single data range only i.e. it shows the share of individual elements of given data range in respect to whole.

```
# creating simple pie chart
import matplotlib.pyplot as plt
data= [15,40,35,30,45]
#  plotting graph
plt.pie(data)
plt .show()
```




```
#  creating simple pie chart with labels
import matplotlib.pyplot as plt
data= [15,40,35,30,45]
plt.pie(data, labels=data)
plt.show()
```

Customizing Pie Chart: pie()

Pyplot's pie() function offers options to customize pie cha rt like Labels, colors and display format for data over sectors.

```
<pyplot obj>.pie(< Data-Value> [, labels =<list of labels for sectors)>] [colors= <color code(s)>] [explode=<explode sequence>] [autopct=<format string>])
```

Data-Value : Dataset for chart. Dataset may be 1-D Array, List, Series or Column of DataFrame.

labels=<list of labels> : Defines texts to be displayed for each sector or partition. Number of labels should be equal to number of data elements. color =<color code(s)>: Defines color for sectors. You should define a list of separate color code for each segment.

explode=<explode sequence>: You may pullout sectors to highlight data . Define a list with 0 or distance in number for sector to be explode.

autopct=<format string>: Defines format string for data labels as "%<width>d" or "%<width>.<precision>f"

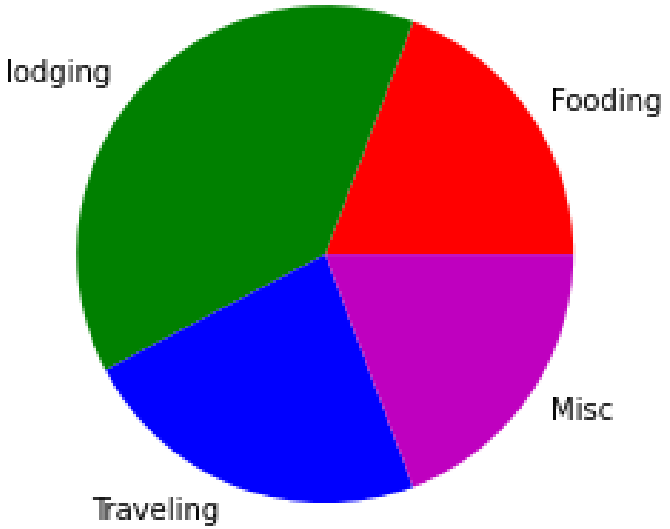
Example: "%5d": defines 5 width integer number.

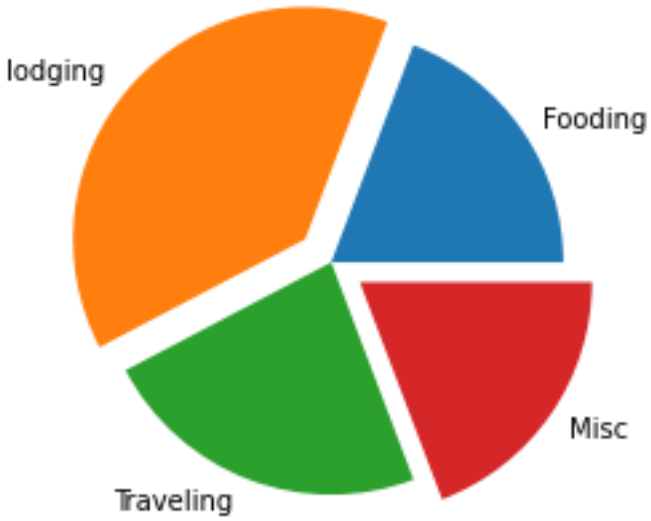
Example: "%6.2f: defines 6 digit number with 2 decimal place

Customizing pie Chart: pie( )

❖ Example of pie chart:

```
# pie chart for expenses by person on tour
import matplotlib.pyplot as plt
exp= [2500,5000,3000,2500]
head= ['Fooding ','lodging ','Traveling
','Misc']
# plotting graph
plt.pie(exp,labels=head,
colors= ['r','g','b','m' ])
plt.show()
```



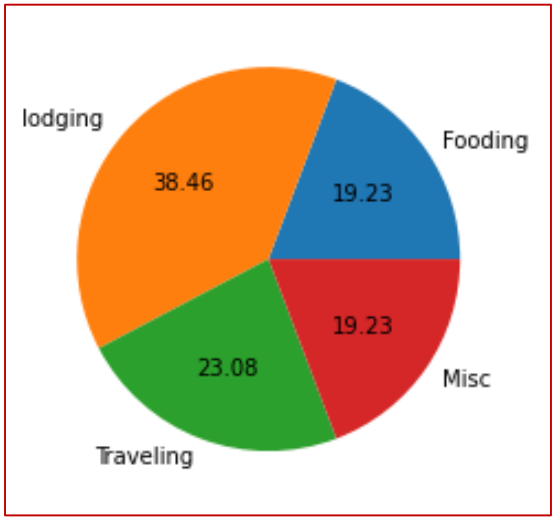


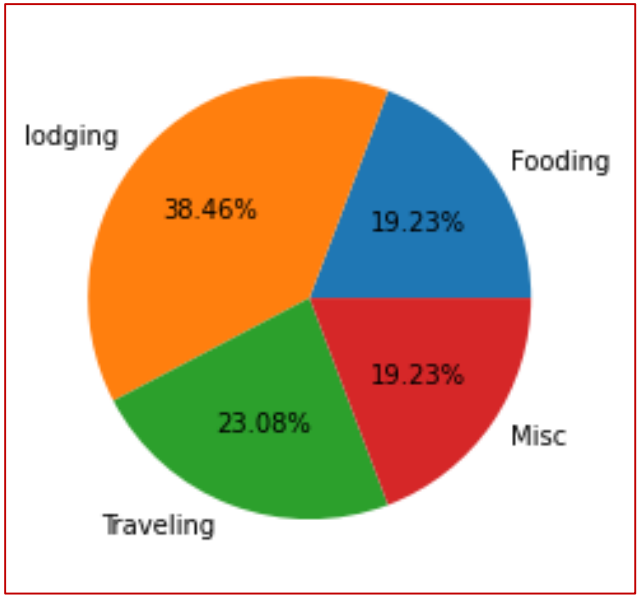

```
#  pie chart for expenses by person on tour
import matplotlib.pyplot as plt
exp= [2500,5000,3000,2500]
head = ['Fooding ','lodging ','Traveling ','Misc']
#  plotting graph
plt.pie(exp, labels=head, explode=[0,0.15,0,0.15])
plt.show()
```

Customizing pie Chart: pie()

❖ Example of pie chart:

```
#  pie chart for expenses by person on tour
import matplotlib.pyplot as plt
exp= [2500,5000,3000,2500]
head= ['Fooding ','lodging ','Traveling','Misc']
#  plotting graph
plt.pie(exp,labels=head, autopct="%5.2f" )
plt.show()
```





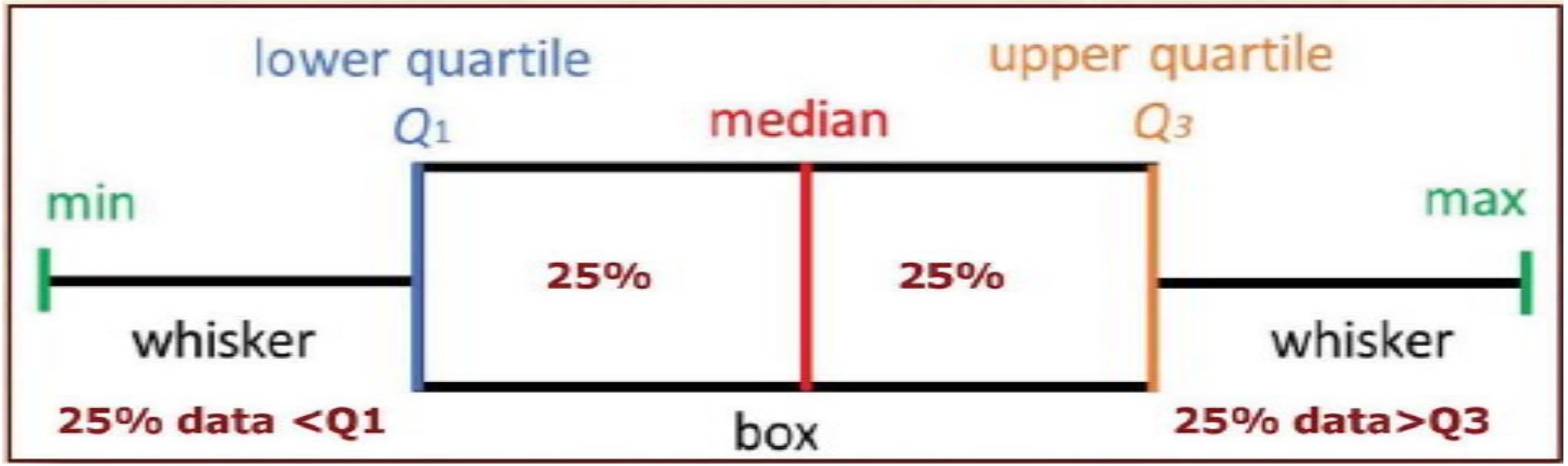

```
# pie chart for expenses by person on tour
import matplotlib.pyplot as plt
exp= [2500,5000,3000,2500]
head= ['Fooding ','lodging ','Traveling ','Misc']
# plotting graph
plt .pie(exp, labels=head, autopct="%5.2f%%“)
plt .show()
```

You can also suffix % sign with data labels by adding %% In format string with autopct.

Box Plot chart: boxplot()

- boxplot(): Creates descri ptive graph with 5 descriptions The Box Plot chart is a presentation of five descriptive indicators, which comprises of the following-

1. The minimum range of Data set (values)- min()
2. The maximum range of Data set (values)- max()
3. The median value of data set -Q2- median()
4. The upper quartile- Q3
5. The lower quartile- Ql



Customizing Box Plot: boxplot()

Pyplot's boxplot() function offers various setting to customize box plot chart like orientation, notch and labels etc.

<pyplot obj>.boxplot(<Data-Value> [, notch = <True/ False>] [vert = <True / False>] [meanline = <True/False>] [showbox = <True/ False>] [showmeans = <True/False>] [patch_artist = <True/False>] [labels = <True/ False>])

Data-Value: Dataset for chart. Dataset may be 1-D Array, List, Series or Column of Dataf rame. Multiple data set may given in list.

notch=<True/False>: Produces notched box plot, if True. Otherwise create simple box plot. Default is False.

vert=<True/False>: Produces vertical box plot, if True. Default is True.

meanline=<True/ False> : Shows mean line in box, if set True.

showbox=< True/ False> : Shows box, if set True. Default is True.

showmeans=<True/ False> : Shows arithmetical mean, if set True.

patch_artist= <True/ False> : Fills the box with color, if set True.

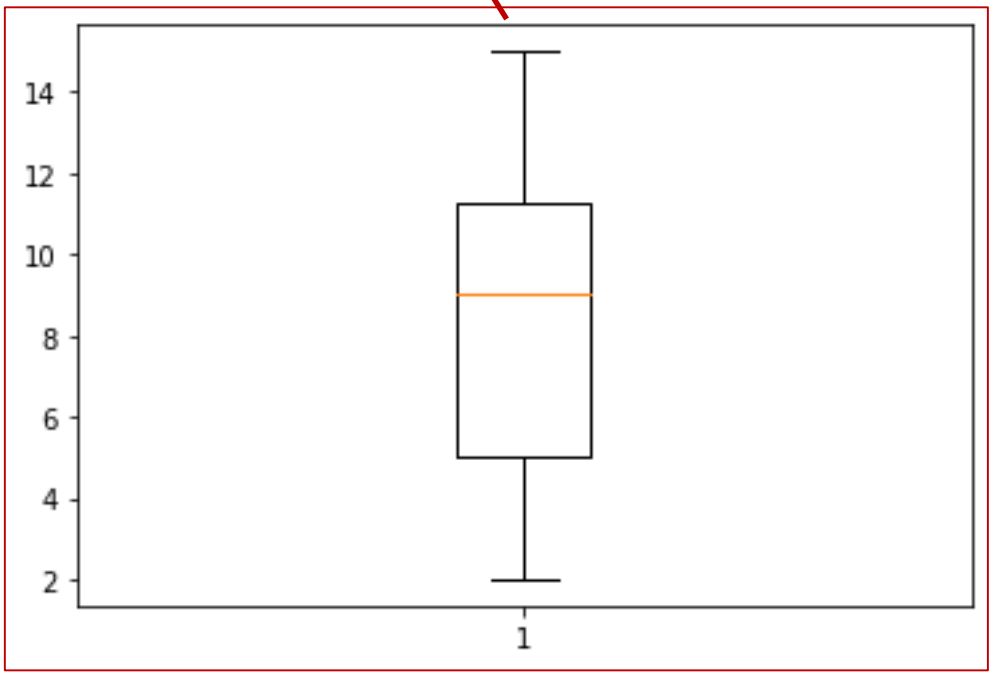
labels=<list of labels> : Defines labels to be displayed for each boxplot. Used when multiple boxplots are being plotted on multiple data set.

Customizing pie Box Plot: boxplot()

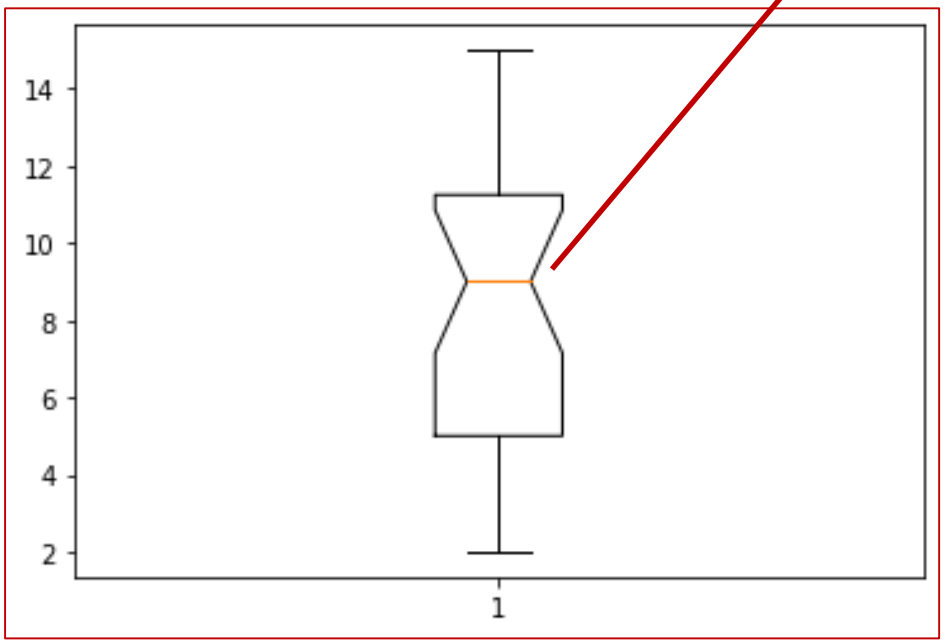
❖ Example of box plot chart:

```
#  Simple Box Plot
import matplotlib .pyplot as plt
data=
[2,4,5,6,2,6,8,10,12,12,11,10,4,3,2,5,
      7,9,11,12,13,14,15,14,10,9,7,9]
#  plot boxplot
plt.boxplot(data)
plt.show()
```

Simple Box plot



Notched Box Plot

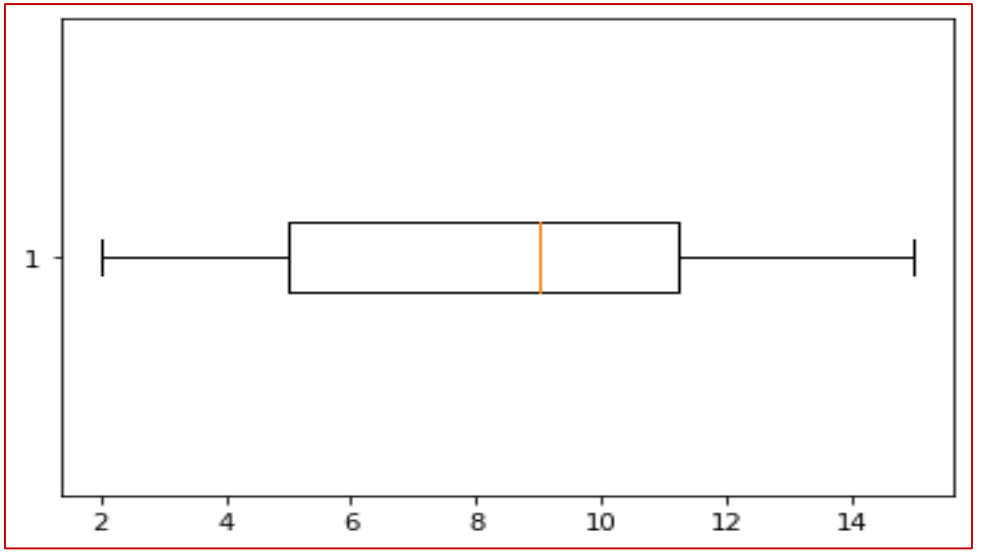


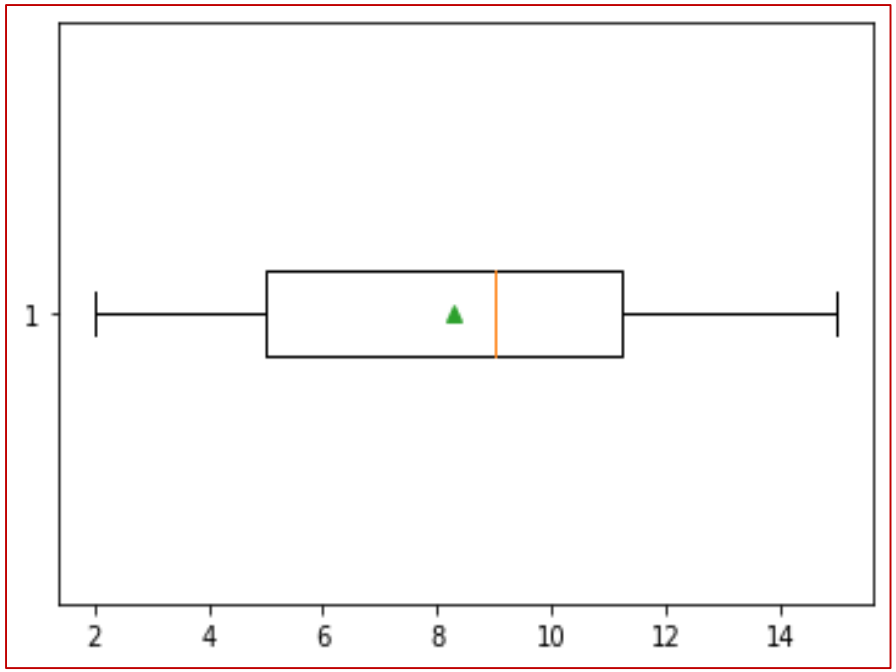

```
#  Notched Box Plot
import matplotlib.pyplot as plt
data=
[2,4,5,6,2,6,8,10,12,12,11,10,4,3,2,5,
      7,9,11,12,13,14,15,14,10,9,7,9]
# plot boxplot
plt.boxplot(data, notch=True)
plt.show()
```

Customizing pie Box Plot: boxplot()

❖ **Example of box plot chart:**

```
#  Vertical Box Plot
import matplotlib .pyplot as plt
data= [2,4,5,6,2,6,8,10,12,12,11,10,4,3,2,5,
7,9,11,12,13,14,15,14,10,9,7,9]
#  plot boxplot
plt.boxplot(data, vert=False)
plt.show()
```





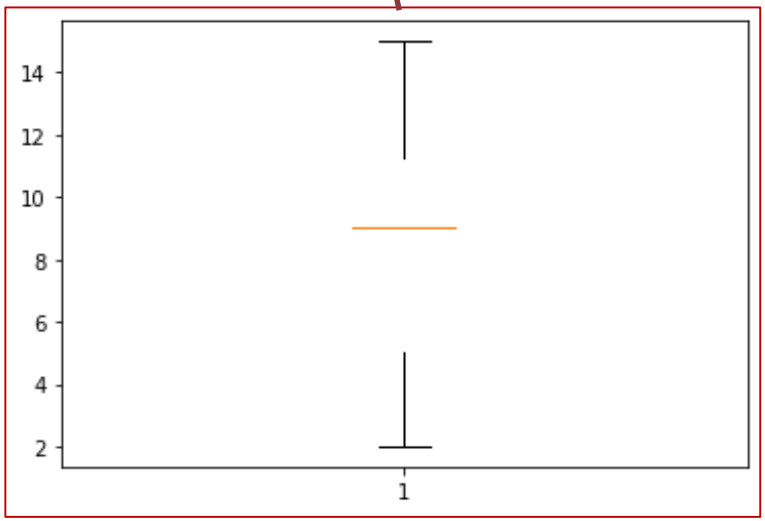

```
#  vertical box plot with mean
import matplotlib.pyplot as plt
data= [2,4,5,6,2,6,8,10,12,12,11,10,4,3,2,5,
      7,9,11,12,13,14,15,14,10,9,7,9]
#  plot boxplot
plt .boxplot(data,vert=False, showmeans=True)
plt .show()
```

Customizing pie Box Plot: boxplot()

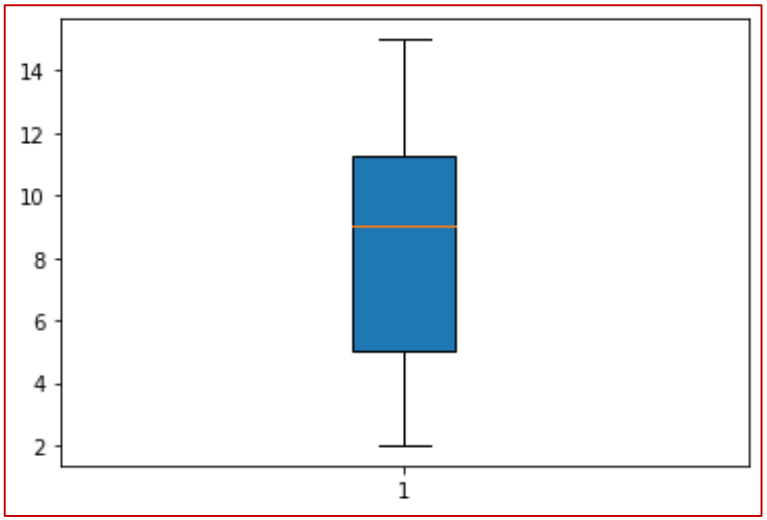
❖Example of box plot chart:

```
#  Box Plot without box
import matplotlib.pyplot as plt
data= [2,4,5,6,2,6,8,10,12,12,11,10,4,3,2,5,
          7,9,11,12,13,14,15,14,10,9,7,9]
#  plot boxplot
plt.boxplot(data, showbox=False)
plt.show()
```

Box plot with no box



Box plot with filled colour

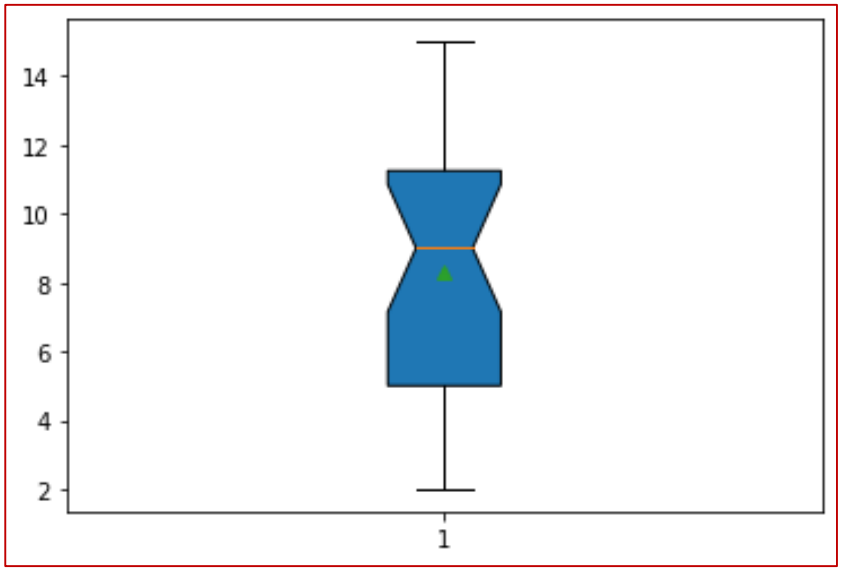



```
# Filled box plot
import matplotlib.pyplot as plt
data= [2,4,5,6,2,6,8,10,12,12,11,10,4,3,2,5,
          7,9,11,12,13,14,15,14,10,9,7,9]
#  plot boxplot
plt .boxplot(data, patch_artist=True)
plt .show()
```

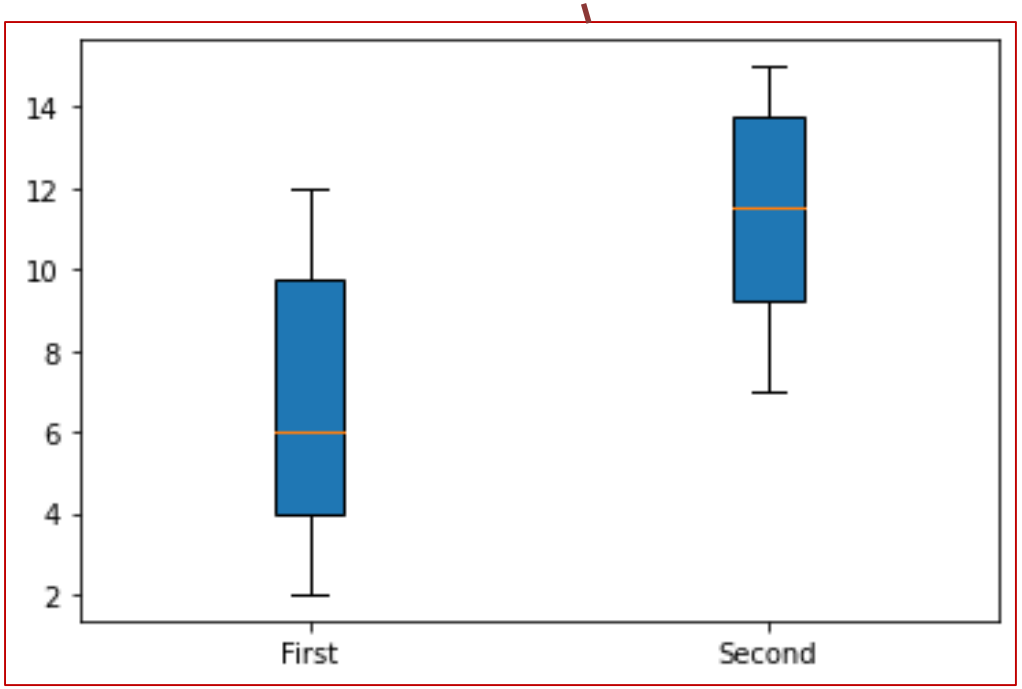
Customizing pie Box Plot: boxplot()

❖ Example of box plot chart:

```
#  Box Plot with notch and mean
import matplotlib.pyplot as plt
data= [2,4,5,6,2,6,8,10,12,12,11,10,4,3,2,5,
      7,9,11,12,13,14,15,14,10,9,7,9]
#  plot boxplot
plt .boxplot(data, notch=True ,showmeans=True,
patch_artist=True)
plt .show()
```



Box plot on multiple data sets




```
# Boxplot on Multiple data set.
import matplotlib.pyplot as plt
data1= [2,4,5,6,2,6,8,10,12,12,11,
       10,4,3,2,5,7,9]
data2= [11,12,13,14,15,14,10,9,7,9]
plt.boxplot([data1,data2],
patch_artist=True,
labels=["First","Second"])
plt.show()
```

Plotting charts using Pandas plot() method -

Alternative way of plotting.

- We have used Pyplot interface of Matplotlib library of Python to plot various types of charts. These methods are capable to handle various types of data sets like Numpy array, list, series and Data Frame etc.
- For plotting charts using numerical data of Series and Data Frame, you can also use Pandas plot() method a lternatively to plot various type of charts.
- Pandas plot() function is capable to plot various charts on Series and Data Frame only. The Pandas Plot() function is simple to use and automatically adds legend to the plots.

```
<Series/ OF obj>.plot([x =<X-axis data >] [y= <Y-Axis Data>] , <Kind=
'line' / 'bar' / 'barh' / 'hist' / 'box' / 'pie' / 'scatter'>
```

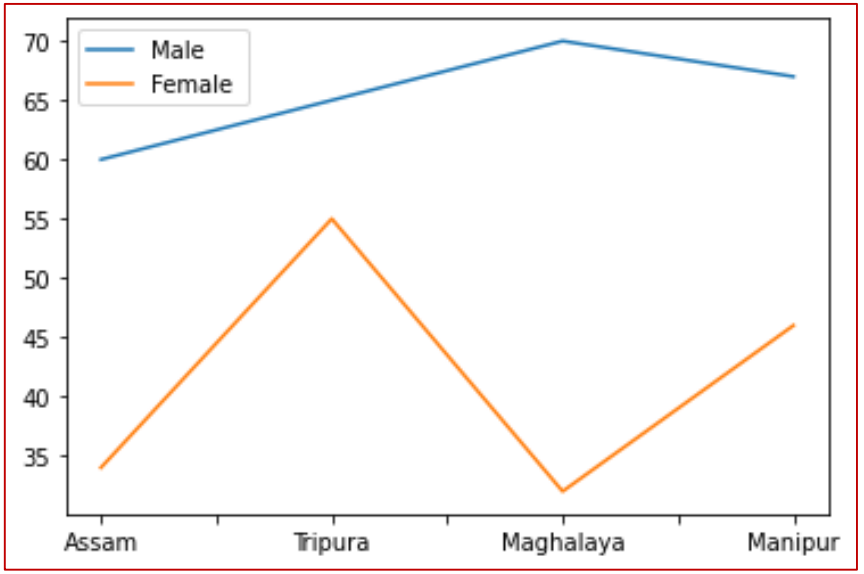
x=X-axis data : Specify DataFrame column to be used for X-axis. y=Y-axis data: Specify DataFrame column to be used for Y-axis.

Kind=<Graph type>: Specify which graph to be plotted. Default chart type is line chart, if no type is given.

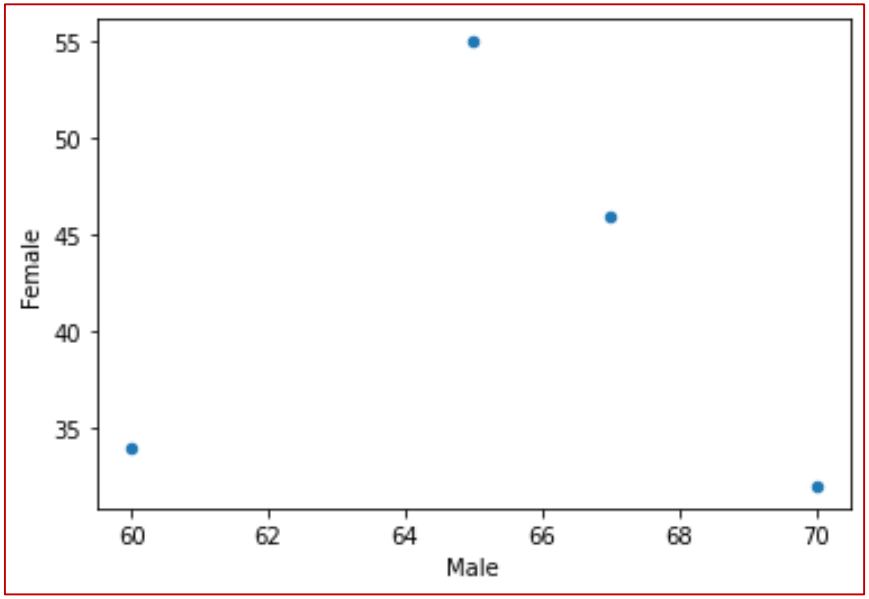
Pandas Plot() Method:

❖ Line chart using Pandas plot() method:

```
import pandas as pd
import matplotlib.pyplot as plt
dct={'Male ':[60,65,70,67],
    'Female ':[34,55,32,46]}
df=pd.DataFrame(dct,index= ['Assam','Tripura',
                            'Maghalaya','Manipur'])
df.plot(kind='line')
plt.show()
```



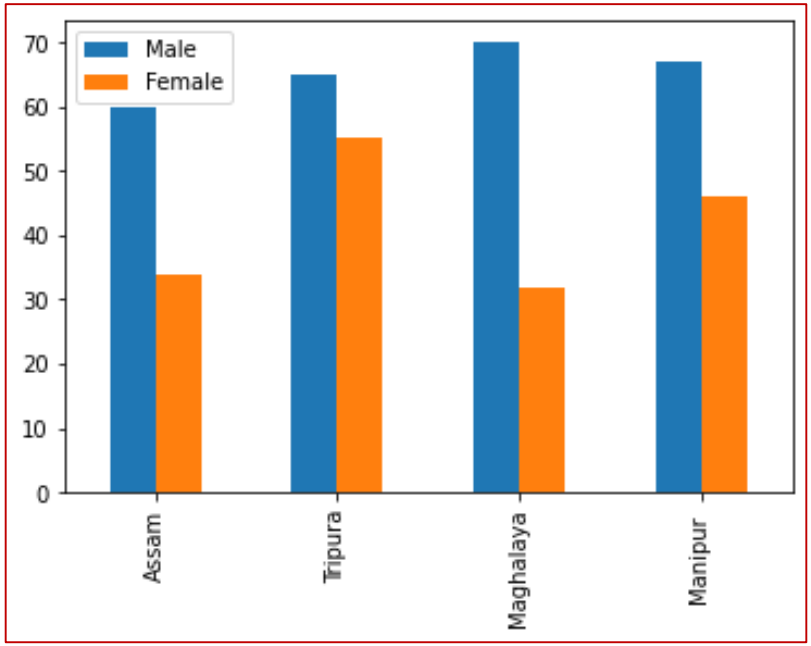
❖ Scatter chart using Pandas plot() method:




```
import pandas as pd
import matplotlib.pyplot as plt
dct={'Male':[60,65,70,67],
    'Female':[34,55,32,46]}
df=pd.DataFrame(dct,index= ['Assam','Tripura','
Maghalaya','Manipur' ])
df.plot(x='Male',y='Female', kind='scatter')
plt .show()
```

Pandas Plot() Method:

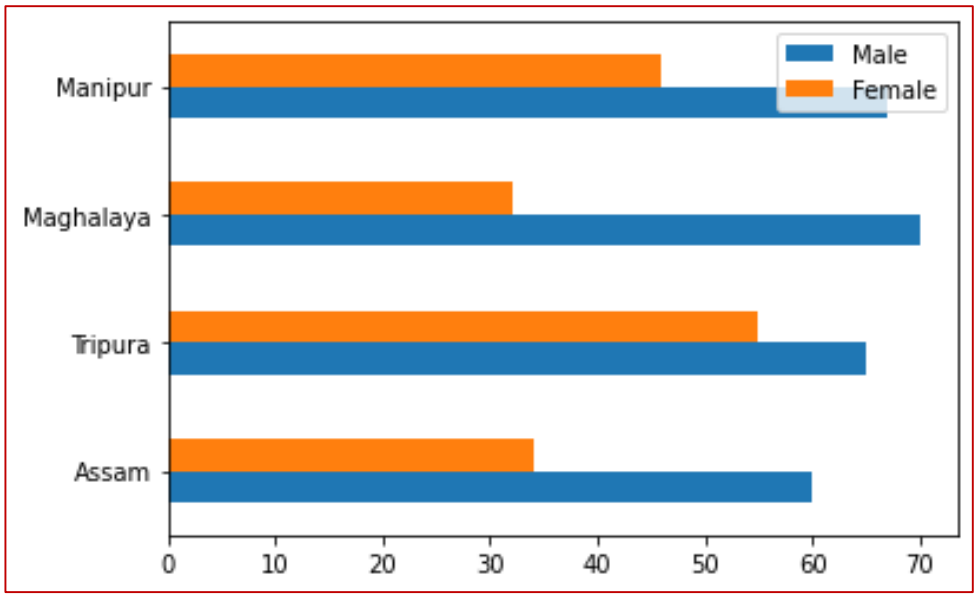
❖ Bar chart using Pandas plot() method:




```
import pandas as pd
import matplotlib.pyplot as plt
dct={'Male ':[60,65,70,67],
    'Female':[34,55,32,46]}
df=pd.DataFrame(dct,index= ['Assam','Tripura',
                            'Maghalaya' ,'Manipur ']) 
df.plot(kind='bar')
plt.show()
```

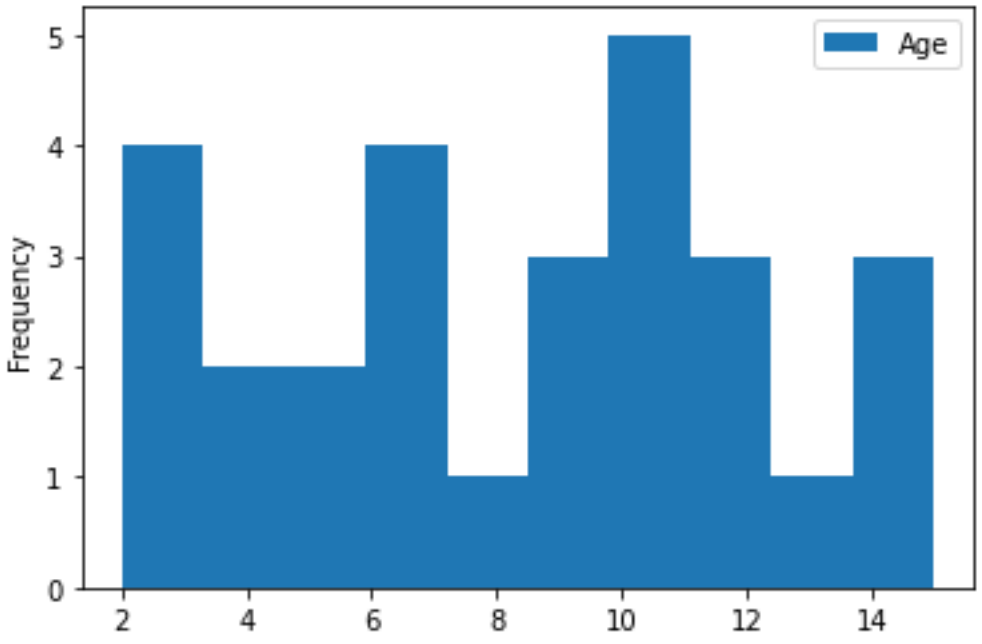
❖ Horizontal Bar chart using Pandas plot() method:

```
import pandas as pd
import matplotlib.pyplot as plt
dct={'Male':[60,65,70,67],
    'Female':[34,55,32,46]}
df=pd.DataFrame(dct,index= ['Assam’,
            'Tripura','Maghalaya','Manipur' ])
df.plot(kind='barh')
plt.show()
```



Pandas Plot() Method:

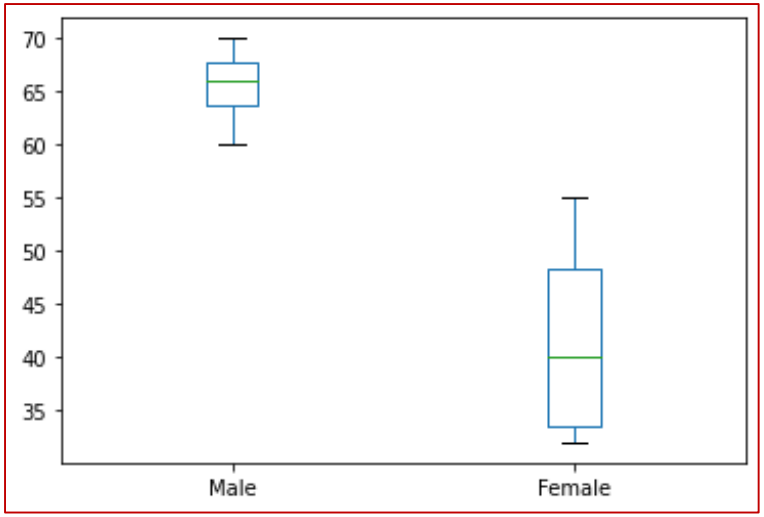
❖ Histogram chart using Pandas plot() method:




```
import pandas as pd
import matplotlib.pyplot as plt
data=[2,4,5,6,2,6,8,10,12,12,11,
      10,4,3,2,5,7,9,11,12,13,14,
      15,14,10,9,7,9]
df=pd.DataFrame(data,columns=['Age'])
df.plot(kind='hist')
plt .show()
```

❖ Boxplot chart using Pandas plot() method:

```
import pandas as pd
import matplotlib.pyplot as plt
dct={'Male':[60,65,70,67], 'Female':[34,55,32,46]}
df=pd.DataFrame (dct,index= ['Assam',
'Tripura','Maghalaya','Manipur' ])
df.plot(kind='box')
plt.show()
```



Other operations on Plotting:

- Setting X-axis limit and Y-axis limit:

You can specify minimum and maximum limit of scales of X-axis and Y-axis using xlim() and ylim() functions.

< pypl ot obj>.xlim (< m in >,< max >)

< pypl ot obj>.ylim (< m in >,< max >)

Ex : plt.xlim(2, 10) plt.ylim(S,50)

- Setting X-axis and Y-axis ticks:

You can also define draw points (ticks) on x-axis and y-axis .

<pyplot obj>.xticks(<list of numbers>[,<tick labels>])

<pyplot obj>.yticks(<list numbers>[,<tick lables>])

Example: plt.xticks([2,4,6,8,10])

- Saving Graphs:

You can a graph in various formats like .jpg or .pdf etc. using pyplot's savefig() method.

<pyplot obj>.savfig(<filename with path>)

Ex : plt.savfig("graph1.png") or plt.savfig("c :\\myfile\graph1.png")