CHAPTER 3 – PLOTTING WITH PYPLOT – I (BAR GRAPHS & SCATTER PLOT)

What is Data Visualization?

- It refers to the graphical or visual representation of information and data using visual elements like charts, graphs, and maps etc.
- Helpful in decision making.
- It unveils pattern, trends, outliers, correlations etc. in the data, and thereby helps decision makers understand the meaning of data to drive business decisions.

Using PyPlot of Matplotlib Library

- The matplotlib is a Python library that provides many interfaces and functionality for 2D-graphics. In short, matplotlib is a high quality plotting library of Python.
- PyPlot is a collection of methods within matplotlib which allows user to **construct 2D plots** easily and interactively.

Importing PyPlot

- In order to use pyplot methods on your computers, we need to import it by issuing one of the following commands:

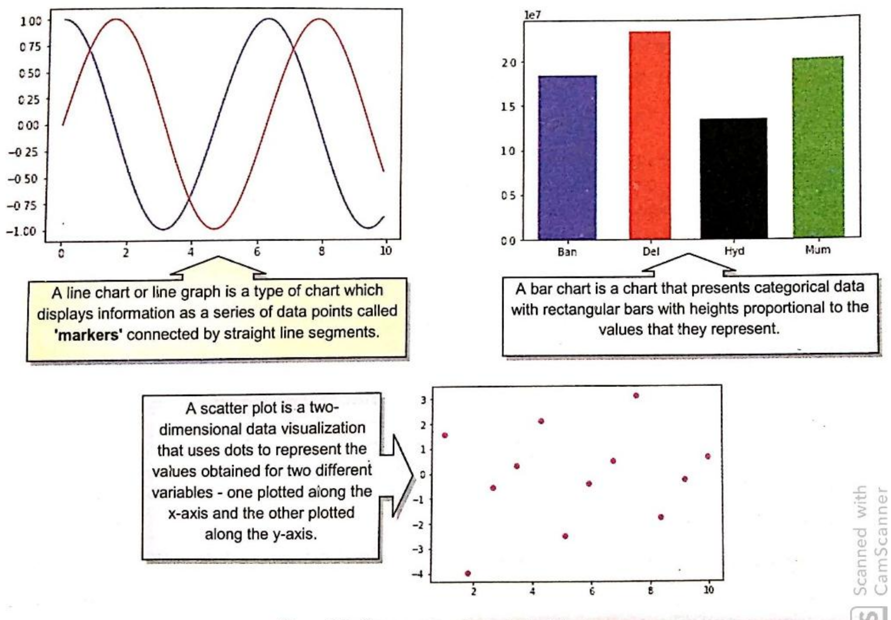
- With the *first command* above, you will need to issue every pyplot command as per following syntax:

```
matplotlib.pyplot.<command>
```

- But with the *second command* above, you have provided **pl** as the shorthand for **matplotlib.pyplot** and thus now you can invoke PyPlot's methods as this:

```
pl.plot(X , Y)
```

Commonly used chart types


A line chart or line graph is a type of chart which displays information as a series of data points called **'markers'** connected by straight line segments.

A bar chart is a chart that presents categorical data with rectangular bars with heights proportional to the values that they represent.

A scatter plot is a two-dimensional data visualization that uses dots to represent the values obtained for two different variables - one plotted along the x-axis and the other plotted along the y-axis.

Figure 3.1 Some commonly used chart types.

Line chart using plot( ) function

- A Line chart or line graph is a type of chart which displays information as a series of data points called 'markers' connected by straight line segments.
- The PyPlot interface offers plot( ) function for creating a line graph.
- **E.g.**

```
import matplotlib.pyplot as pl
a=[1,2,3,4]
b=[2,4,6,8]
c=[1,4,9,16]
pl.plot(a,b)
pl.xlabel("Some Values")
pl.ylabel("Doubled Values")
pl.show()
```

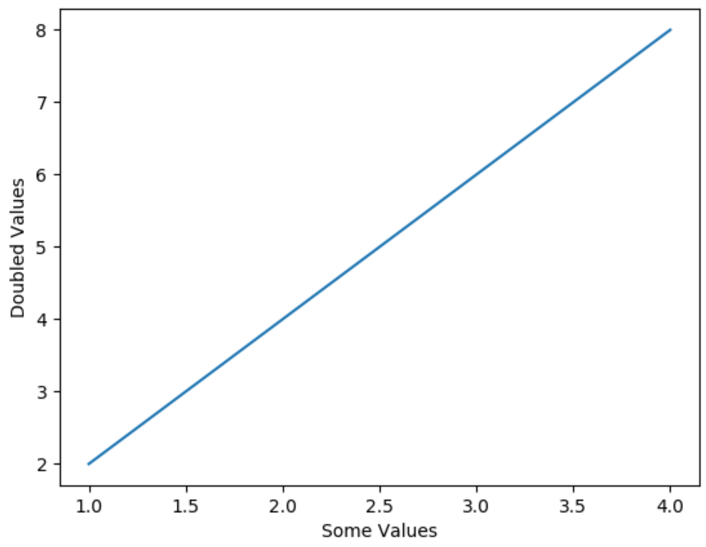
The import statement is to be given just once

List b containing **values as double** of values in **list a**

List c containing **values as squares** of values in **list a**

show( ) method is used to display plot as per given specification

Output:



- You can set x-axis' and y-axis' labels using functions xlabel( ) and ylabel( ) respectively, i.e.:

  <matplotlib.pyplot or its alias> . xlabel(<string>)

  and

  <matplotlib.pyplot or its alias> . ylabel(<string>)

Applying Various Settings in plot( ) Function

The plot( ) function allows you to specify multiple settings for your chart/graph such as:

- color(line color/marker color)
- marker type
- marker size , etc.

Changing Line Color

<matplotlib>.plot(<data1>,<data2> , **<color code>**)

Different color code

character	color	character	color	character	color
‘b’	blue	‘m’	magenta	‘c’	cyan
‘g’	green	‘y’	yellow	‘w’	white
‘r’	red	‘k’	black		

Changing Line Style

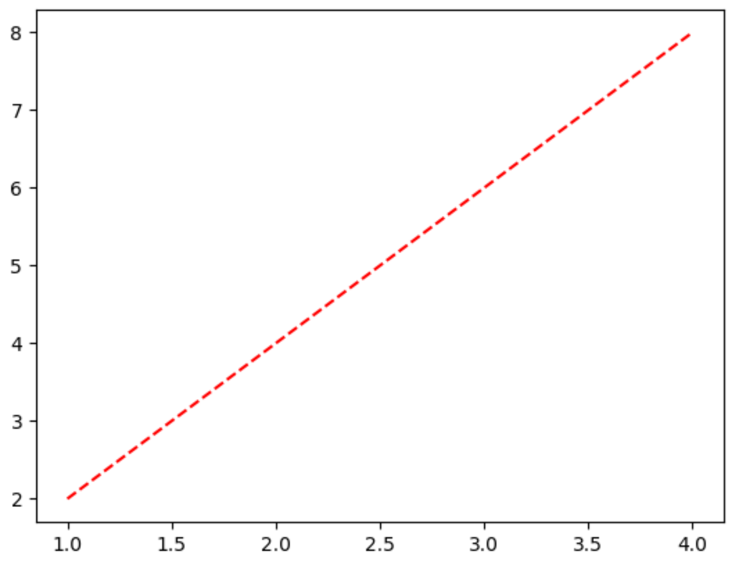
<matplotlib>.plot(<data1>, <data2> , **<linestyle>**)

linestyle or ls = [‘Solid’ | ‘dashed’ , ‘dashdot’ , ‘dotted’]

e.g.

```
import matplotlib.pyplot as plt
a=[1,2,3,4]
b=[2,4,6,8]
c=[1,4,9,16]
plt.plot(a,b,'r',linestyle='dashed')
plt.show()
```

Output:

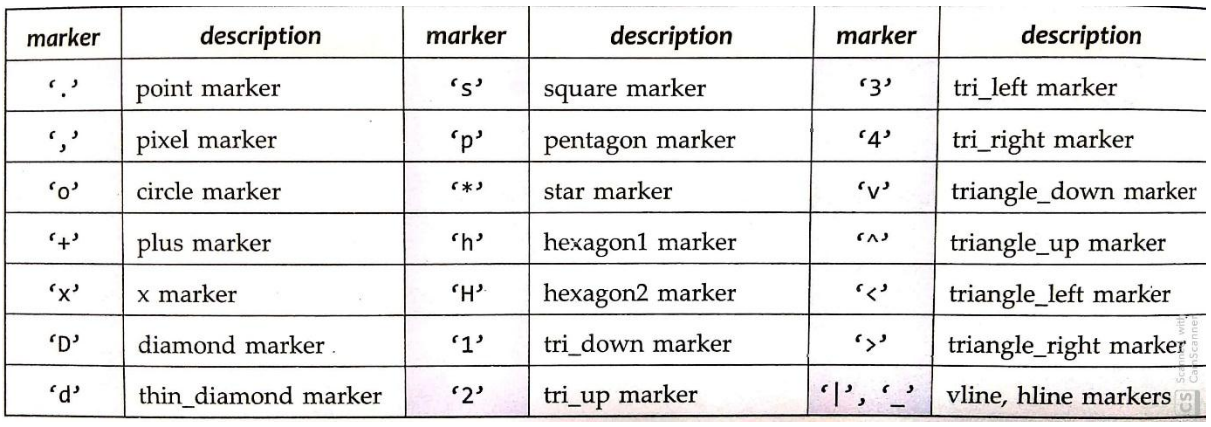


Changing Marker Type, Size and Color

- data points being plotted are called **markers.** To change market type, its size and color, following arguments can be used in plot( ) function:

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

Marker Type for Plotting



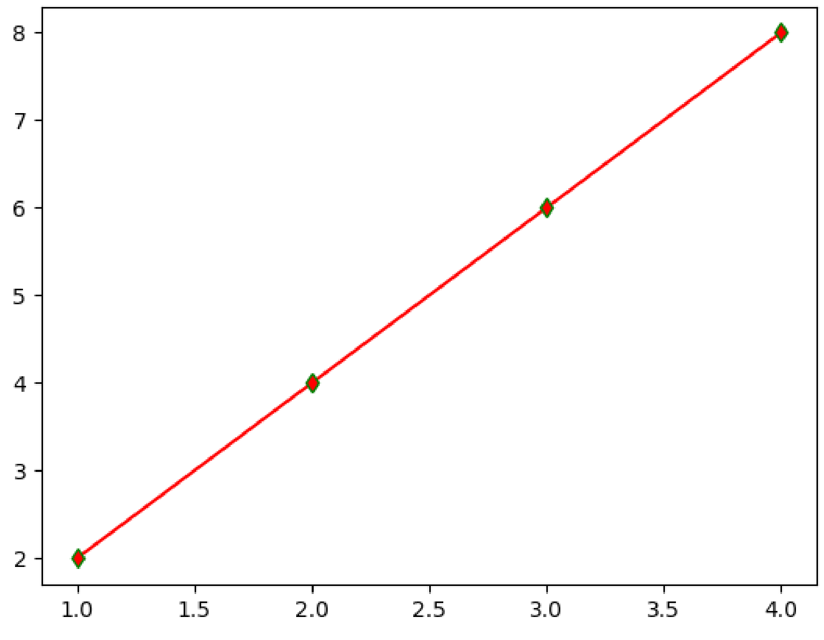
marker	description	marker	description	marker	description
'.'	point marker	's'	square marker	'3'	tri_left marker
','	pixel marker	'p'	pentagon marker	'4'	tri_right marker
'o'	circle marker	'*'	star marker	'v'	triangle_down marker
'+'	plus marker	'h'	hexagon1 marker	'^'	triangle_up marker
'x'	x marker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

For example:

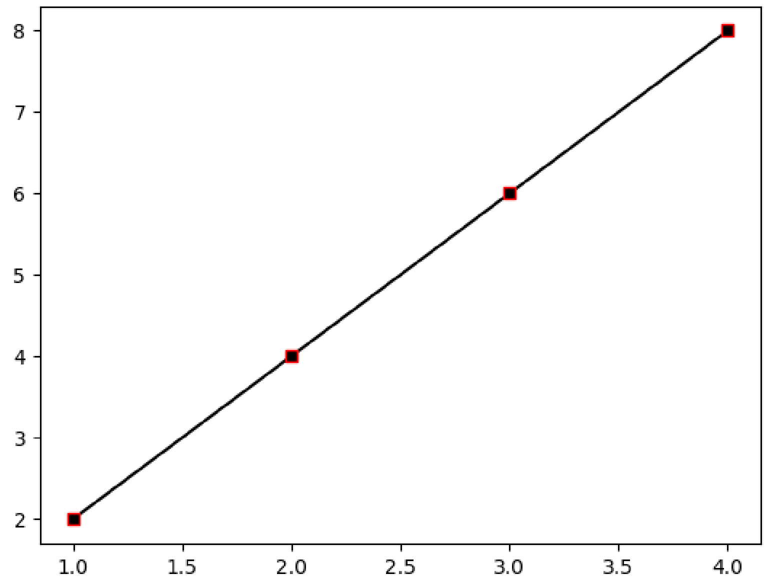
```
import matplotlib.pyplot as plt
a=[1,2,3,4]
b=[2,4,6,8]
c=[1,4,9,16]
plt.plot(a,b,'r',marker='d',markersize=6,markeredgecolor='green')  #plot1
plt.show()
plt.plot(a,b,'k',linestyle='solid',marker='s',markersize=6,markeredgecolor='red')  #plot2
plt.show()
plt.plot(a,b,'r+',linestyle='solid',markersize=6,markeredgecolor='green') #plot3
plt.show()
```

Output:

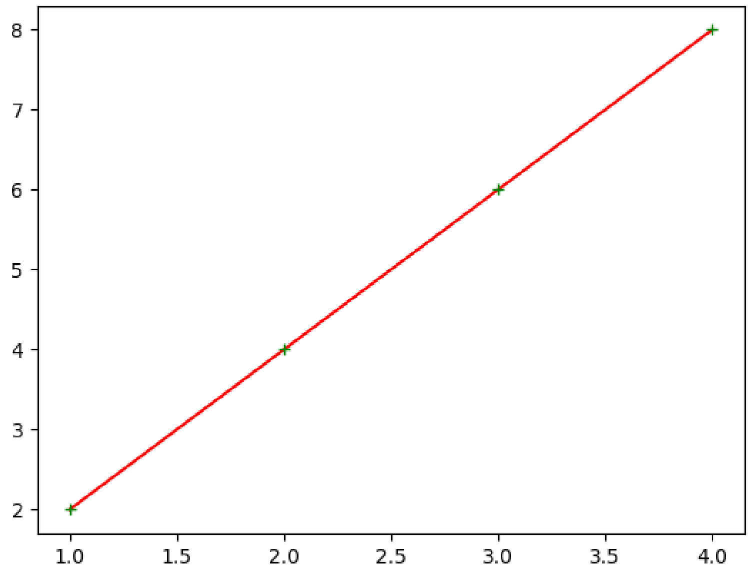
#plot1



#plot2



#plot3



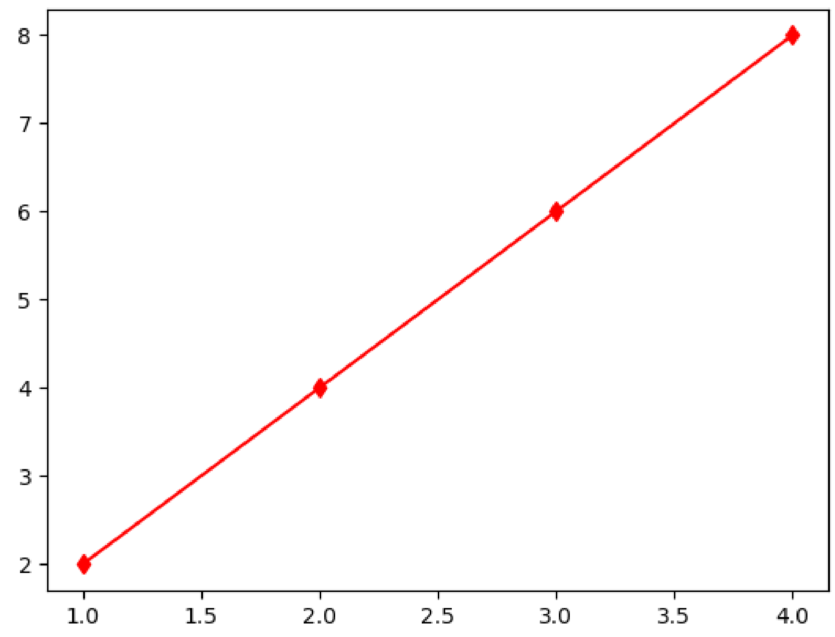
–

** when you do not specify **markeredgecolor** separately in **plot( )** , the marker takes the same color as the line.

E.g.

```
import matplotlib.pyplot as plt
a=[1,2,3,4]
b=[2,4,6,8]
plt.plot(a,b,'r',marker='d',markersize=6)
plt.show()
```

Output:

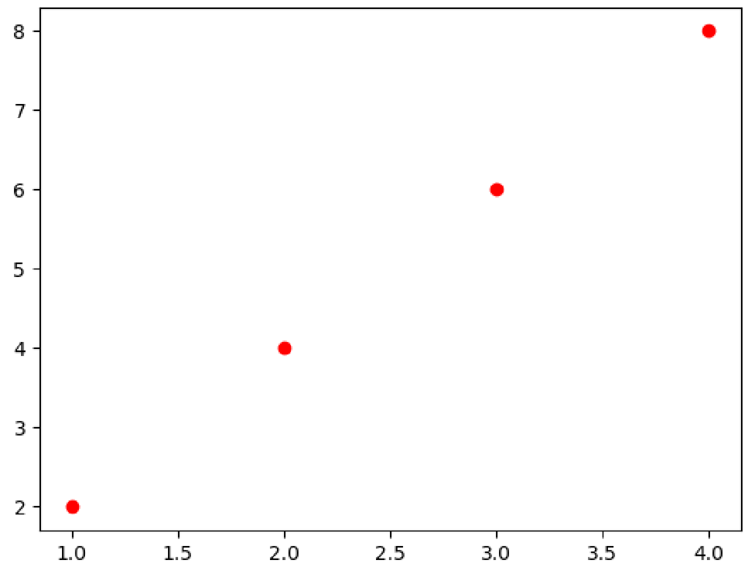


** Also, if you do not specify the **linestyle** separately along with linecolor & markerstyle-combination-string(e.g. 'r+' above), Python will only plot the markers and not the line.

E.g.

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

Output:



Creating Scatter Chart

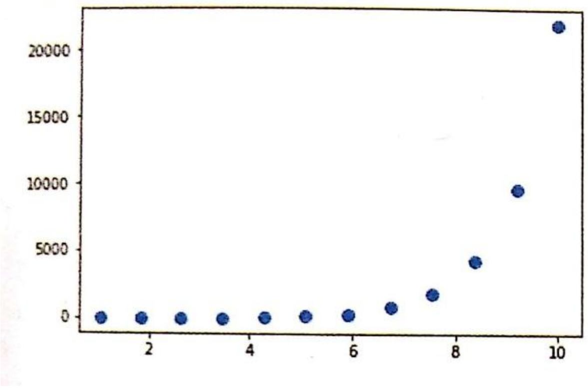
- It is a graph of plotted points on two axes that show the relationship between two sets of data.
- The scatter charts can be created through two functions of pyplot library:
  1. plot( ) function
  2. scatter( ) function

<u>Scatter charts using plot( ) function</u>

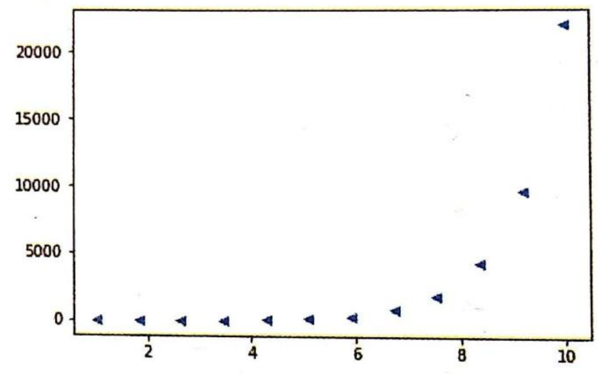
- If you specify the **linecolor** and **markerstyle** (e.g. “r+” or “bo” etc.) without the **linestyle** argument, then the plot created resembles a scatter chart as only the datapoints are plotted now.

e.g.

```
In [36]: pl.plot(a1, a4, "o", markersize = 8)
Out[36]: [<matplotlib.lines.Line2D at 0x7968e90>]
```



```
In [37]: pl.plot(a1, a4, "<")
Out[37]: [<matplotlib.lines.Line2D at 0x86aec70>]
```



Example 3.1:

Example 3.1 *Create an array in the range 1 to 20 with values 1.25 apart. Another array contains the log values of the elements in first array.*

(*a*) *Create a plot of first vs second array ; specify the x-axis (containing first array's values) title as* **'Random Values'** *and y-axis title as* **'Logarithm Values'**.

(*b*) *Create a third array that stores the COS values of first array and then plot both the second and third arrays vs first array. The Cos values should be plotted with a* **dashdotted line.**

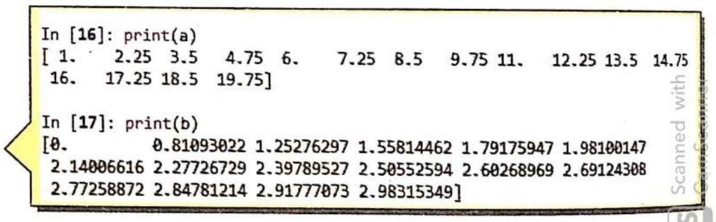
(*c*) *Change the marker type as a circle with blue color in second array.*

(*d*) *Create scatter chart as this : second array data points as* **blue small diamonds**, *third array data points as* **black circles.**

Note : Show commands and their results.

Solution.

```
import numpy as np
import matplotlib.pyplot as plt
a = np.arange(1, 20, 1.25)
b = np.log(a)
```

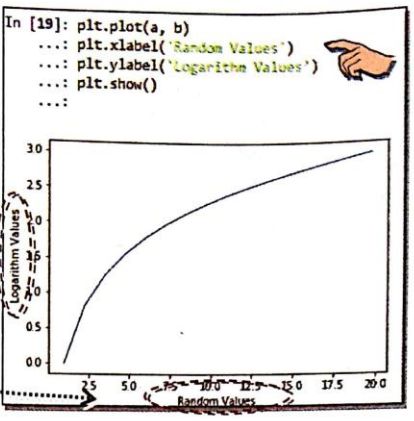


```
In [16]: print(a)
[ 1.     2.25  3.5   4.75  6.     7.25  8.5   9.75 11.    12.25 13.5  14.75
 16.    17.25 18.5   19.75]

In [17]: print(b)
[0.         0.81093022 1.25276297 1.55814462 1.79175947 1.98100147
 2.14006616 2.27726729 2.39789527 2.50552594 2.60268969 2.69124308
 2.77258872 2.84781214 2.91777073 2.98315349]
```

(*a*)

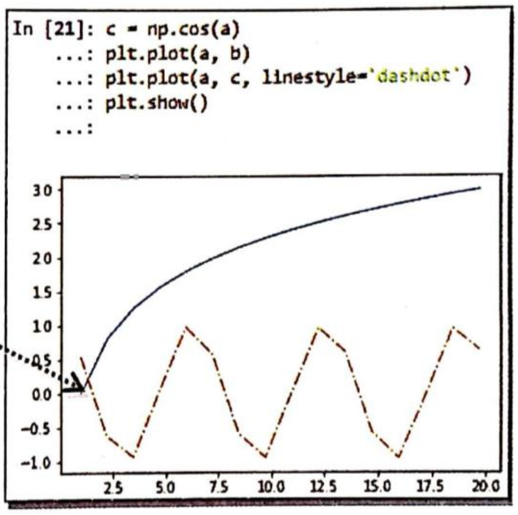
```
plt.plot(a, b)
plt.xlabel('Random Values')
plt.ylabel('Logarithm Values')
plt.show()
```



```
In [19]: plt.plot(a, b)
    ...: plt.xlabel('Random Values')
    ...: plt.ylabel('Logarithm Values')
    ...: plt.show()
    ...:
```

(*b*)

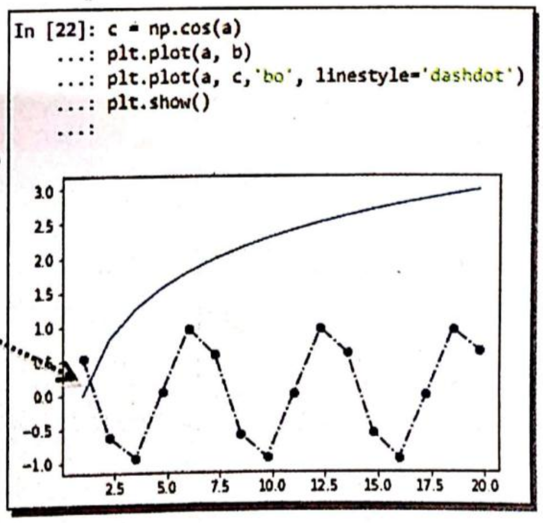
```
c = np.cos(a)
plt.plot(a, b)
plt.plot(a, c, linestyle = 'dashdot')
plt.show()
```



```
In [21]: c = np.cos(a)
    ...: plt.plot(a, b)
    ...: plt.plot(a, c, linestyle='dashdot')
    ...: plt.show()
    ...:
```

(*c*)

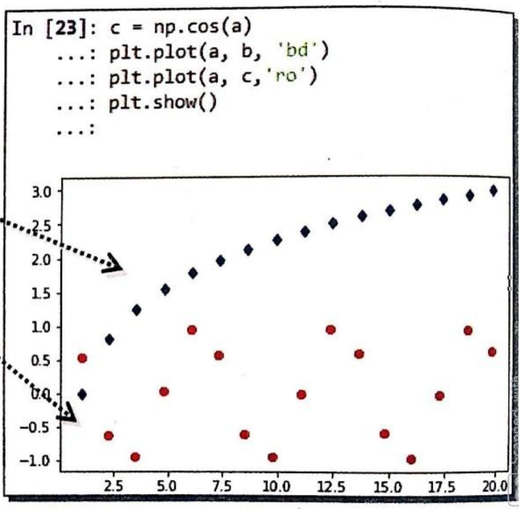
```
c = np.cos(a)
plt.plot(a, b)
plt.plot(a, c, 'bo', linestyle = 'dashdot')
plt.show()
```



```
In [22]: c = np.cos(a)
    ...: plt.plot(a, b)
    ...: plt.plot(a, c,'bo', linestyle='dashdot')
    ...: plt.show()
    ...:
```

(*d*)

```
c = np.cos(a)
plt.plot(a, b, 'bd')
plt.plot(a, c, 'ro')
plt.show()
```



```
In [23]: c = np.cos(a)
    ...: plt.plot(a, b, 'bd')
    ...: plt.plot(a, c,'ro')
    ...: plt.show()
    ...:
```

Scatter Charts using scatter Function ( )

- This function can be used as:

  matplotlib.pyplot.scatter(<array1>, <array2>)

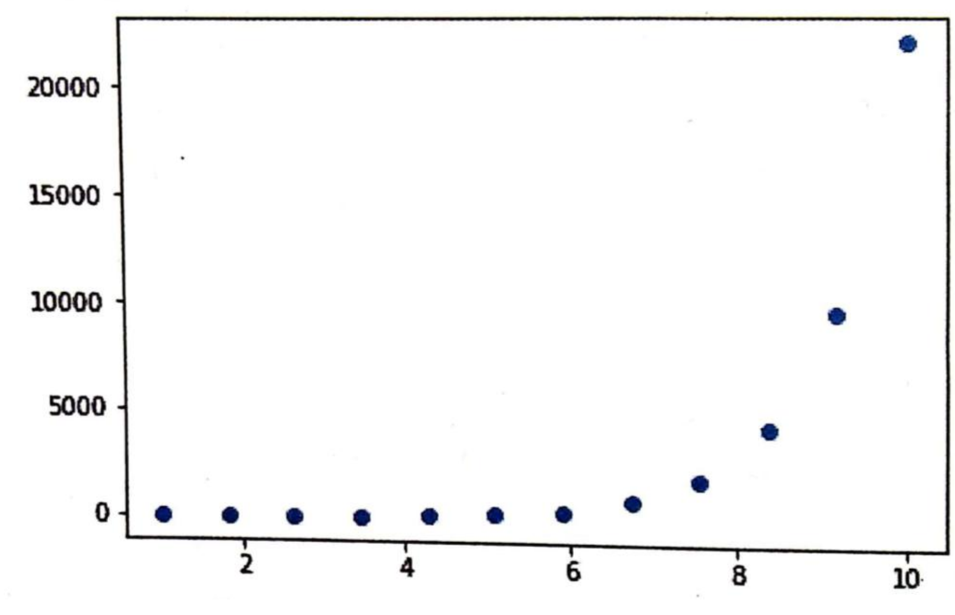
  or

  <pyplot aliasname>.scatter(<array1>, <array2>)

e.g.

```
import matplotlib.pyplot as pl
#a1 and a4 are two ndarrays
pl.scatter(a1 , a4)
```

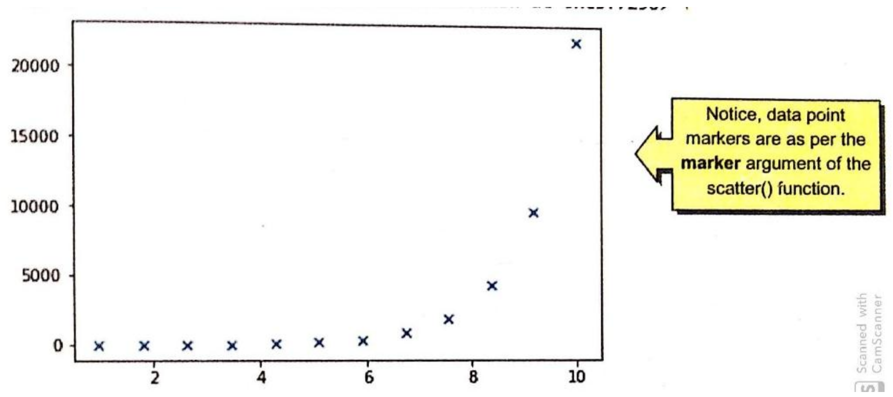
Output:



Specifying marker type and size

Using **marker** argument marker type can be specified and using argument **s** , size can be marker size can be specified. **E.g.**

pl.scatter(a1 , a4, marker = “x”, s = 5)



<u>Specifying color of the markers</u>

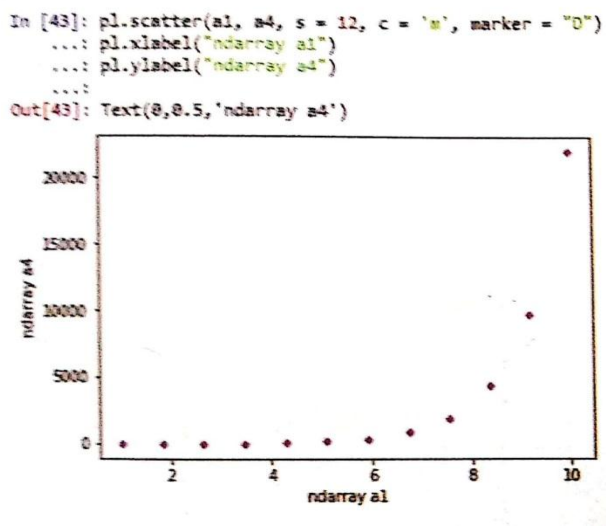
Using argument **c ,** you can specify the color of the markers.

<u>Detailed syntax of scatter( ) function</u>

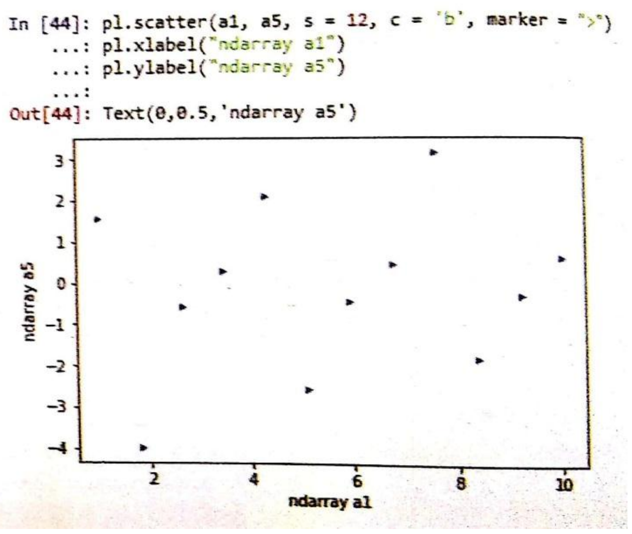
matplotlib.pyplot.scatter(<array1>, <array2>,**s**=None , **c**=None , **marker**=None)

where

s – The marker size in points
c – marker color
marker – marker style



```
In [43]: pl.scatter(a1, a4, s = 12, c = 'm', marker = "D")
    ...: pl.xlabel("ndarray a1")
    ...: pl.ylabel("ndarray a4")
    ...:
Out[43]: Text(0,0.5,'ndarray a4')
```

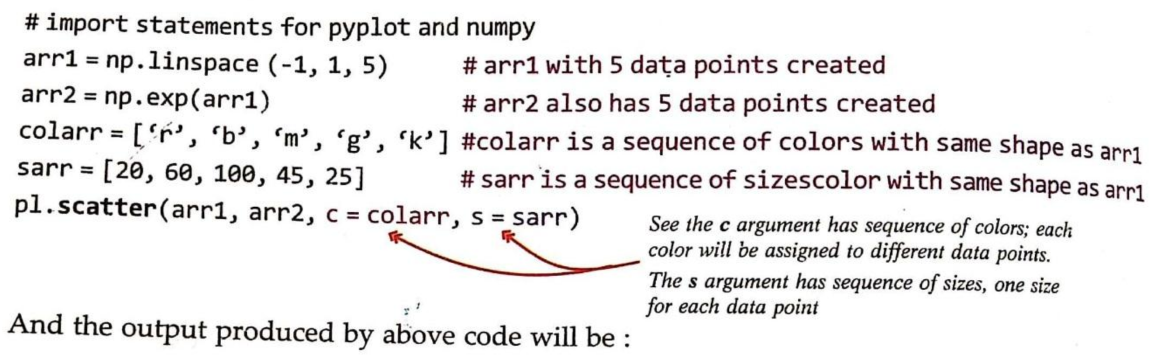


```
In [44]: pl.scatter(a1, a5, s = 12, c = 'b', marker = ">")
    ...: pl.xlabel("ndarray a1")
    ...: pl.ylabel("ndarray a5")
    ...:
Out[44]: Text(0,0.5,'ndarray a5')
```

<u>Specifying varying colors and sizes for data points</u>

Scatter function allows you to specify different sizes and color and size for the data points. For this purpose, you need to specify an array of colors having the same shape as arrays being plotted as the value of **c argument** and an array of sizes having the same shape as arrays being plotted as the value of **s argument.**

E.g.

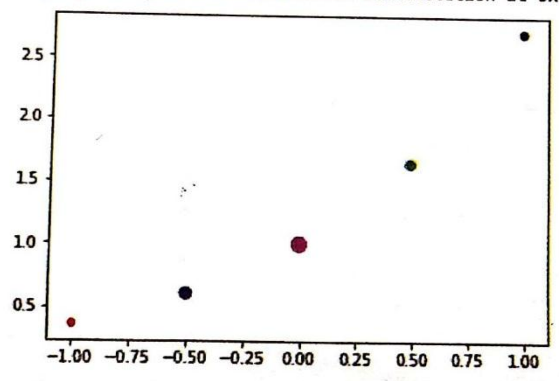


```
# import statements for pyplot and numpy
arr1 = np.linspace (-1, 1, 5)          # arr1 with 5 data points created
arr2 = np.exp(arr1)                    # arr2 also has 5 data points created
colarr = ['r', 'b', 'm', 'g', 'k']    #colarr is a sequence of colors with same shape as arr1
sarr = [20, 60, 100, 45, 25]           # sarr is a sequence of sizescolor with same shape as arr1
pl.scatter(arr1, arr2, c = colarr, s = sarr)
```

*See the **c** argument has sequence of colors; each color will be assigned to different data points.*
*The **s** argument has sequence of sizes, one size for each data point*

And the output produced by above code will be :

```
In [67]: pl.scatter(arr1, arr2, c = colarr, s = sarr)
Out[67]: <matplotlib.collections.PathCollection at 0xc85a930>
```



Creating Bar Charts

- A Bar Graph/ Chart is a graphical display of data using bars of different heights.
- Pyplot offers **bar( )** function to create a bar chart where you can specify the sequences for *x-axis* and corresponding sequence to be plotted on *y-axis .*

  e.g.
  ```
  import matplotlib.pyplot as plt
  a=[1,2,3,4]
  b=[2,4,6,8]
  c=[1,4,9,16]
  plt.bar(a,b)
  plt.xlabel("values")
  plt.ylabel("Doubles")
  plt.show()
  plt.bar(a,c)
  plt.xlabel("values")
  plt.ylabel("Squares")
  plt.show()
  ```

- If you want to specify *x-axis label* and *y-axis label,* then you need to give commands:
  ```
  matplotlib.pyplot.xlabel(<label string>)
  matplotlib.pyplot.ylabel(<label string>)
  ```

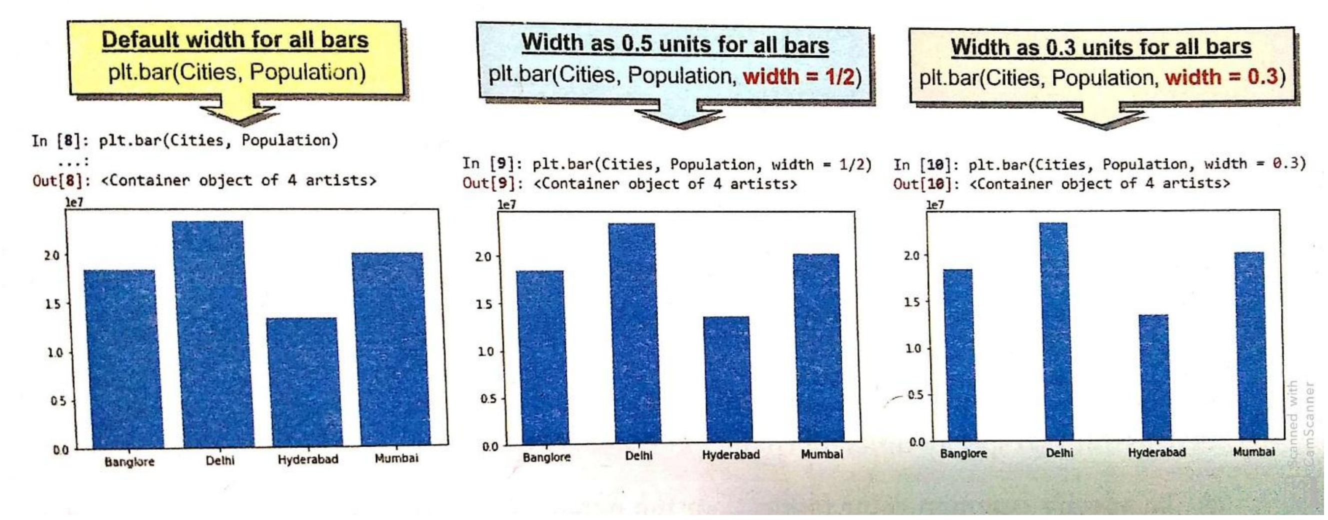
Changing Widths of the Bars in a Bar Chart

- By default, bar chart draws bars with equal widths. (default width is 0.8 units)
- Bar width can be changed in following 2 manners:

1. **To specify common width(other than the default width) for all bars,** you can specify **width** argument having a **scalar float value** in the bar( ) function, i.e.

   <matplotlib.pyplot>.bar(x sequence , y sequence , width = <float value>)

e.g.


Default width for all bars
plt.bar(Cities, Population)
Width as 0.5 units for all bars
plt.bar(Cities, Population, width = 1/2)
Width as 0.3 units for all bars
plt.bar(Cities, Population, width = 0.3)
1e7
Banglore
Delhi
Hyderabad
Mumbai

```
In [8]: plt.bar(Cities, Population)
   ...:
Out[8]: <Container object of 4 artists>
```

```
In [9]: plt.bar(Cities, Population, width = 1/2)
Out[9]: <Container object of 4 artists>
```

```
In [10]: plt.bar(Cities, Population, width = 0.3)
Out[10]: <Container object of 4 artists>
```

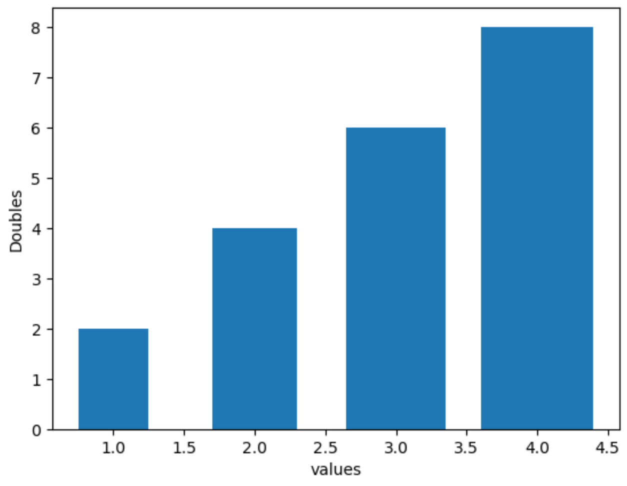
2. **To specify different widths for different bars of a bar chart,** you can specify **width** argument having a sequence (list or tuple) containing widths for each of the bars, in the **bar( )** function, i.e.

   <matplotlib.pyplot>.bar(x sequence , y sequence , width = <width values sequence>)

e.g.

```
import matplotlib.pyplot as plt
a=[1,2,3,4]
b=[2,4,6,8]
c=[1,4,9,16]
plt.bar(a,b, width = [0.5 , 0.6 , 0.7 , 0.8])
plt.xlabel("values")
plt.ylabel("Doubles")
plt.show( )
```

Output:



Note: the **width** sequence must have widths for all bars(i.e., its length must match the length of data sequences being plotted) otherwise Python will report an error [**valueError : *shape mismatch error***]

Changing Colors of the Bars in a Bar Chart

Color of the Bars can be changed in 2 ways:

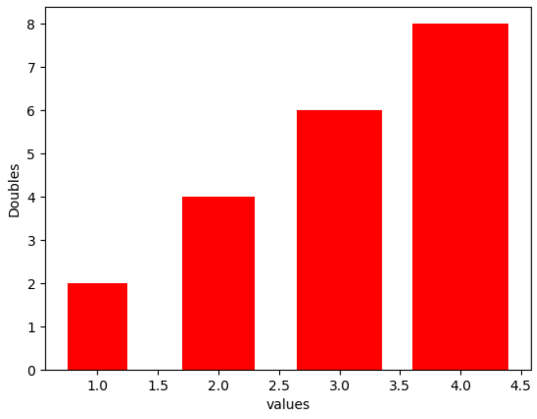
(i) To specify the common color(other than default color) for all bars, you can specify **color** argument having a **valid color code/name** in the bar( ) function:

<matplotlib.pyplot>.bar(x sequence , y sequence ,color = <color code/name>)

e.g.

```
import matplotlib.pyplot as plt
a=[1,2,3,4]
b=[2,4,6,8]
c=[1,4,9,16]
plt.bar(a,b, width = [0.5 , 0.6 , 0.7 , 0.8], color=’red’)
plt.xlabel("values")
plt.ylabel("Doubles")
plt.show( )
```

Output:



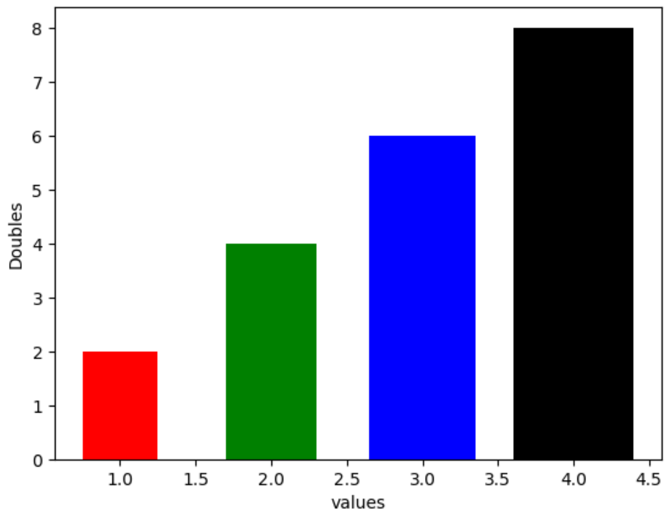
(ii) To specify different colors for different bars of a bar chart, you can specify ***color*** argument having a sequence(list or tuple) containing colors for each of the bars, in the bar( ) function:

<matplotlib.pyplot>.bar(x sequence , y sequence ,color = <color code sequence >)

e.g.

```
import matplotlib.pyplot as plt
a=[1,2,3,4]
b=[2,4,6,8]
c=[1,4,9,16]
plt.bar(a,b, width = [0.5 , 0.6 , 0.7 , 0.8], color=['red', 'g' , 'b' , 'black'])
plt.xlabel("values")
plt.ylabel("Doubles")
plt.show( )
```

Output:



Note: the **color** sequence must have color for all bars(i.e., its length must match the length of data sequences being plotted) otherwise Python will report an error [**valueError :** ***shape mismatch error***]

Creating Multiple Bars chart

Say we want to plot ranges , A = [2 , 4 , 6 , 8] and B =[2.8 , 3.5 , 6.5 , 7.7]

Steps:

1. ***Deciding X points and thickness***. Say, we want the thickness of each bar as 0.35, then **for the first range, X point will be X** and **for the second range,** the X will shift by first bar's thickness, i.e. X+0.35.

2. ***Deciding colors.***Say we want *red* color for the first range and blue color for the second range.

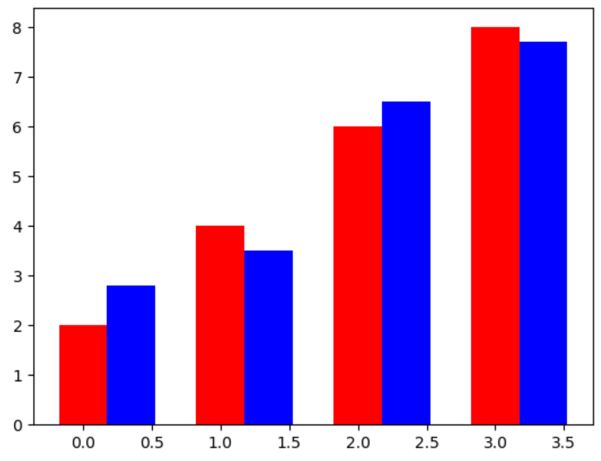
3. The **width** argument will take value as 0.35 in this case.

4. Plot using multiple bar( ) functions.

E.g.

```
import matplotlib.pyplot as plt
import numpy as np
A=[2,4,6,8]
B=[2.8,3.5,6.5,7.7]
X=np.arange(len(A))
plt.bar(X,A,color="red",width=0.35)
plt.bar(X+0.35,B,color="blue",width=0.35)
plt.show()
```

Output:



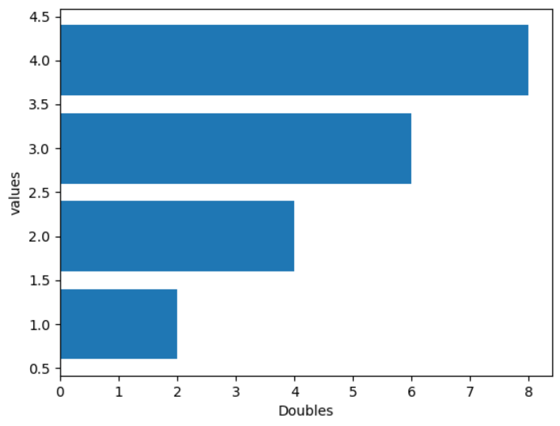
Creating a Horizontal Bar Chart

- To create a horizontal bar chart, you need to use **barh( )** function, in place of bar.

e.g.

```
import matplotlib.pyplot as plt
a=[1,2,3,4]
b=[2,4,6,8]
plt.barh(a,b)
plt.ylabel("values")
plt.xlabel("Doubles")
plt.show()
```

Output:



Anatomy of a Chart

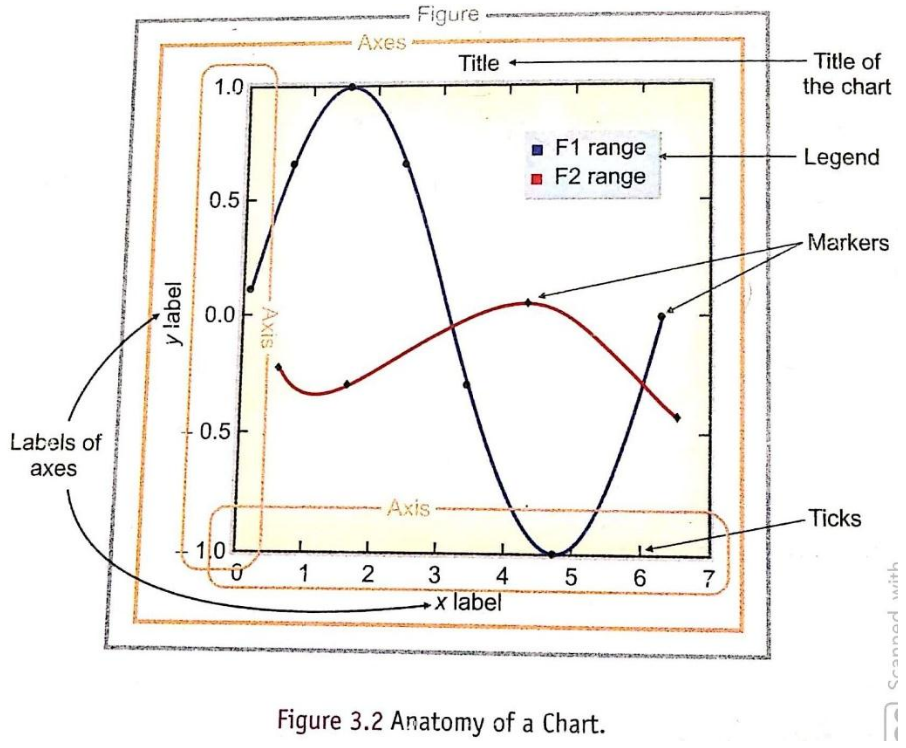


Figure 3.2 Anatomy of a Chart.

- **Figure :** Pyplot by default plots every chart into an area called *Figure.* A figure contains other elements of the plot in it.
- **Axes :** It defined the area on which actual plot(line or bar or graph etc.) will appear. Axes have properties like **label , limits** and **tick marks** on them.
  There are two axes in a plot : (i) **x-axis,** the horizontal axis , (ii) **y-axis,** the vertical axis.

  Axis label – it defines the name for an axis. It is individually defined for X-axis and Y-axis each.
  Limits – These define the range of values and number of values marked on X-axis and Y-axis.
  Tick_Marks – The tick marks are individual points marked on the X-axis or Y-axis.

- **Title:** This is the text that appears on the top of the plot. It defines what the chart is about.
- **Legends :** These are the different colors that identify different sets of data plotted on the plot. The legends are shown in a corner of the plot.

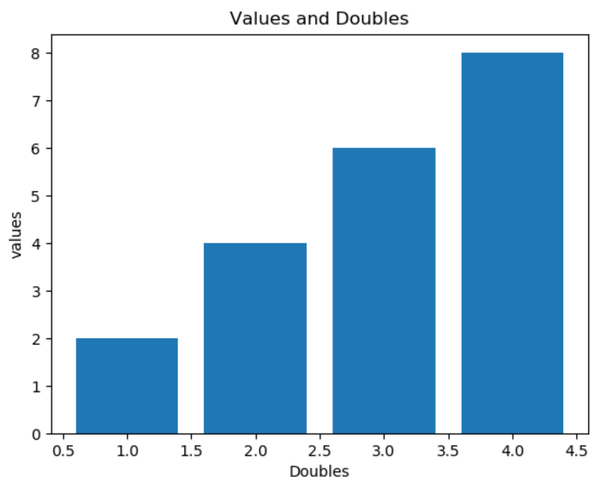
Adding a Title

- The syntax to add a title in a plot is :
  <matplotlib.pyplot>.title(<title string>)

  e.g.

```
import matplotlib.pyplot as plt
a=[1,2,3,4]
b=[2,4,6,8]
plt.bar(a,b)
plt.title("Values and Doubles")
plt.ylabel("values")
plt.xlabel("Doubles")
plt.show()
```

Output:



Setting Limits and Ticks

(i) Setting Xlimits and Ylimits

- Both **xlim( )** and **ylim( )** are used as per following format:

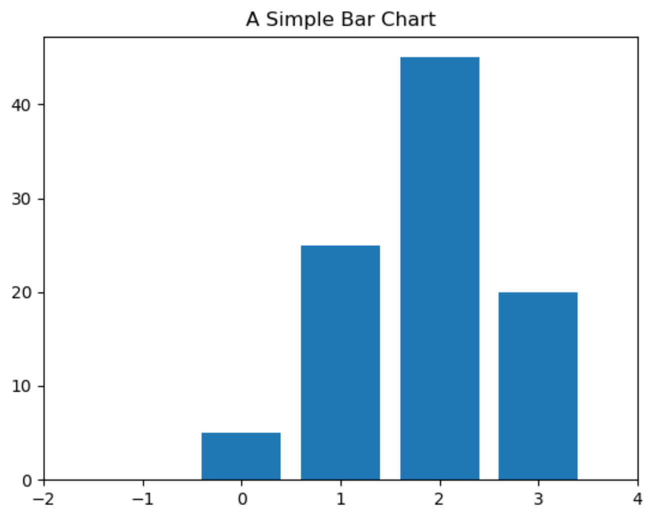
  <matplotlib.pyplot>.xlim(<xmin> , <xmax>)

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

- **E.g.**

```
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()
```

Output:

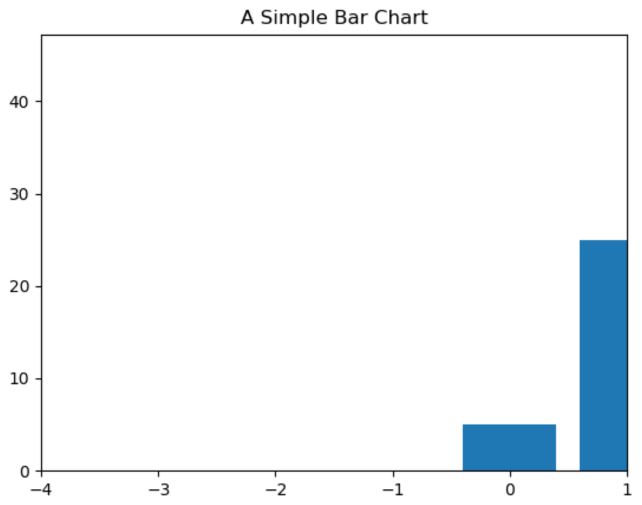


Note: While setting up the limits for axes, you must keep in mind that only the data that falls into the limits of X and Y-axes will be plotted; rest of the data will not show in the plot.

E.g.

```
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()
```

<u>Output:</u>



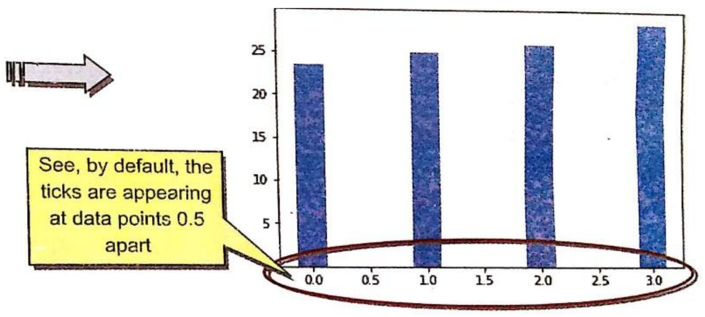
Note: If you do not specify X or Y limits, PyPlot will automatically decide the limits for X and Y-axes as per the values being plotted.

(ii) <u>Setting Ticks for Axes</u>

- To set own tick marks:
  - ➢ For X-axis , you can use **xticks( )** function as per format:
    xticks(<sequence containing tick data points>, [optional sequence containing tick labels])
  - ➢ For Y-axis , you can use **yticks( )** function as per format:
    yticks(<sequence containing tick data points>, [optional sequence containing tick labels])

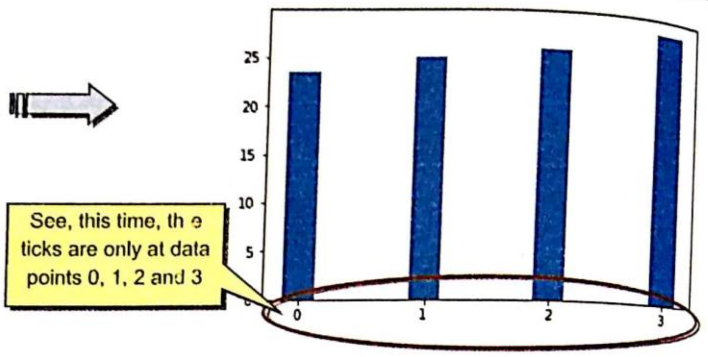
e.g.

```
q = range(4)
s = [23.5, 25, 26, 28.5]
plt.bar(q, s, width = 0.25)
```



If you want that tick marks should appear only at data points 0, 1, 2 and 3, you will need to give code as :

```
q = range(4)
s = [23.5, 25, 26, 28.5]
plt.xticks([0,1, 2, 3])
plt.bar(q, s, width = 0.25)
```

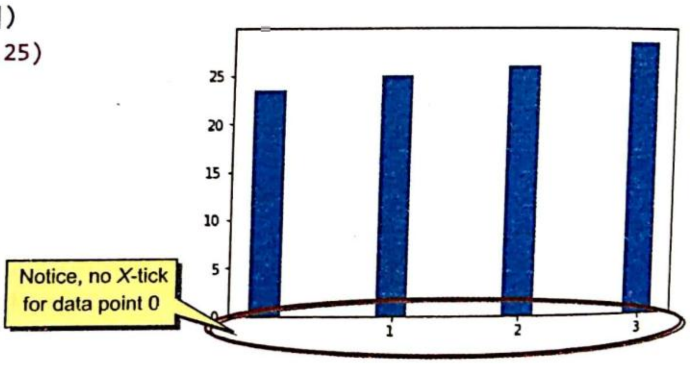


Now carefully go through the following code (it is very similar to the code above) :

```
q = range(4)
s = [23.5, 25, 26, 28.5]
plt.xticks([1, 2, 3, 4])
plt.bar(q,s, width = 0.25)
```

Can you figure out how the resultant plot will look like ?

Hmm. Well, you guessed it right – this time *X*-ticks will appear at data points 1, 2, 3.. but not on data point 0, *i.e.*, as :



Adding Legend

- When we plot multiple ranges on a single plot, it becomes necessary that legends are specified.
- **Two step process:**
  (i) In the plotting functions like plot( ) , bar( ) etc., give a specific label to data range using argument **label.**
  (ii) Add legend to the plot using legend( ) as per format:
  <matplotlib.pyplot>.legend(loc = <position number or string>)

  The **loc** argument can either take values 1, 2, 3, 4 signifying the position strings 'upper right','upper left','lower left','lower right' respectively. Default position is 'upper right' or 1.

e.g.

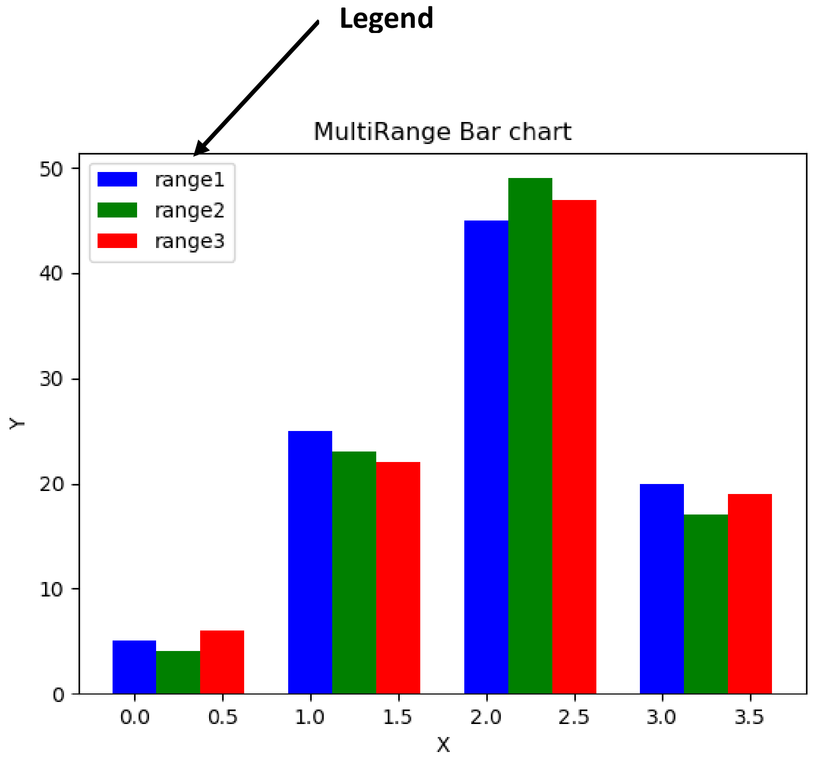
```
import matplotlib.pyplot as plt
import numpy as np
val=[[5.,25.,45.,20.],[4.,23.,49.,17.],[6.,22.,47.,19.]]
x=np.arange(4)

#step1: specify label for each range being plotted using label
plt.bar(x+0.00,val[0],color='b',width=0.25,label='range1')
plt.bar(x+0.25,val[1],color='g',width=0.25,label='range2')
plt.bar(x+0.50,val[2],color='r',width=0.25,label='range3')

#step2:add legend,i.e.
plt.legend(loc='upper left')

plt.title("MultiRange Bar chart")
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

Output:



Saving a Figure

- If you want to save a plot created using pyplot functions for later use or for keeping records, you can use **savefig( )** to save the plot.
- You can use the pyplot's savefig( ) as per format:

  <matplotlib.pyplot>.savefig(<string with filename and path>)

 *you can save figures in popular formats like ***.pdf* , *.png* ,** etc.

e.g. plt.savefig("multibar.pdf")   # save the plot in current directory
 plt.savefig("c:\\data\\multibar.pdf") #save the plot at the given path

****

Creating Histogram with PyPlot

- A histogram is a summarisation tool for discrete or continuous data.
- It provides visual interpretation of numerical data by showing the number of data points that fall within a specified range of values (called **bins)**.
- It is similar to a vertical bar graph. However, a histogram, unlike a vertical bar graph, shows no gaps between the bars.

Histogram using hist( ) Function

- The syntax for using hist( ) function of pyplot is:

Matplotlib.pyplot.hist(x , bins=None, Cumulative = False , histtype='bar' , align='mid', orientation = 'vertical')

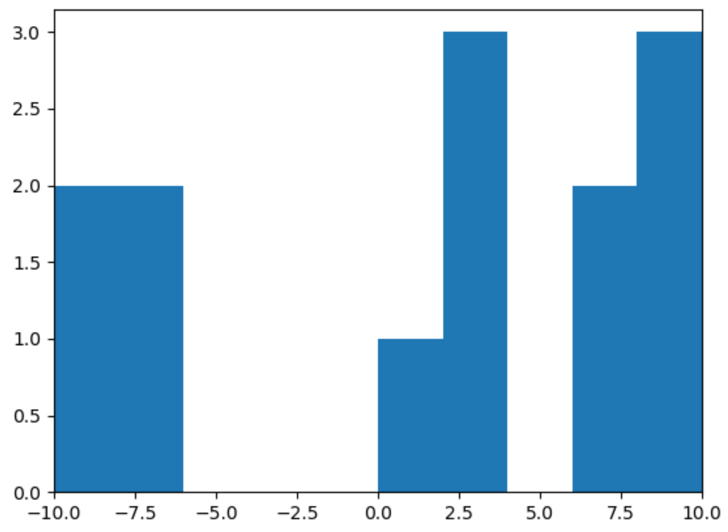
Parameters:

1. x - array or sequence to be plotted on histogram.
2. bins – integer ,optional ; If an integer is given, bins+1 bin edges are calculated and returned.
3. cumulative – bool , optional ; If *True,* then a histogram is computed where each bin gives the counts in that bin plus all bins for smaller values. Default value is *False.*
4. histtype – {'bar' , 'barstacked' , 'step' , 'stepfilled' } , optional ; the type of histogram to draw. Default is 'bar'.
5. orientation – {'horizontal' , 'vertical'}, optional ; If 'horizontal' , barh will be used for bar-type histograms.

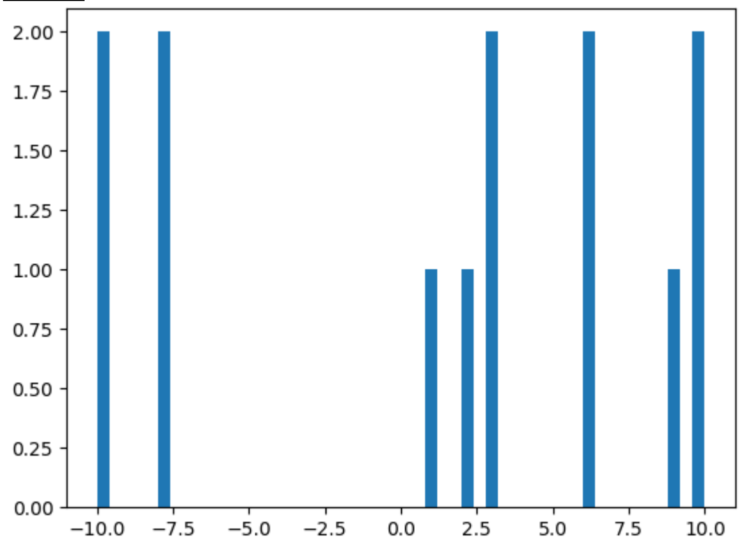
e.g.1.

```
import matplotlib.pyplot as plt
import numpy as np
x=[-10,-8,3,6,9,10,-10,2,1,-8,3,6,10]
plt.xlim(-10,10)
plt.hist(x)
plt.show()          #hist1
plt.hist(x,bins=50)
plt.show()          #hist2
plt.hist(x,bins=100)
plt.show()          #hist3
```

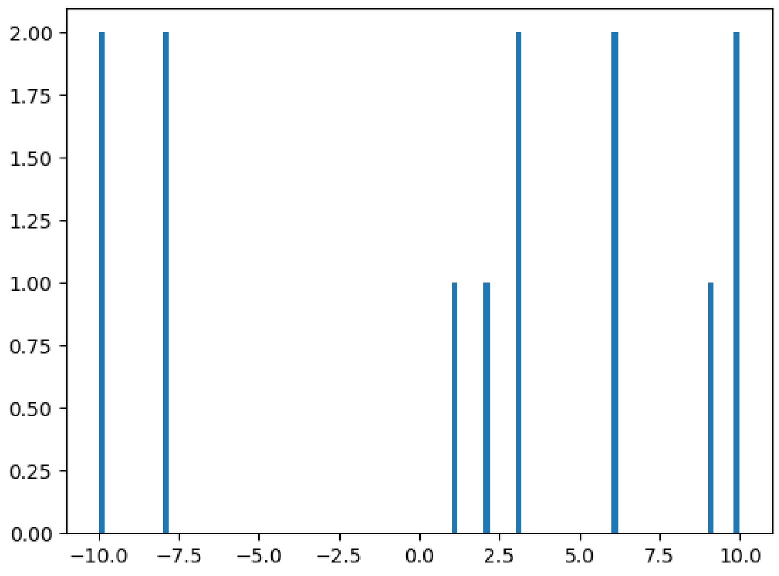
#hist1



#hist2



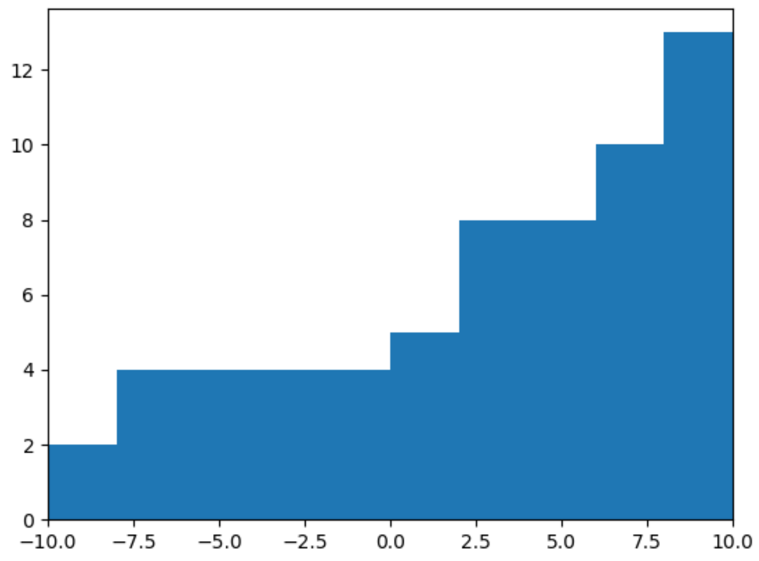
#hist3



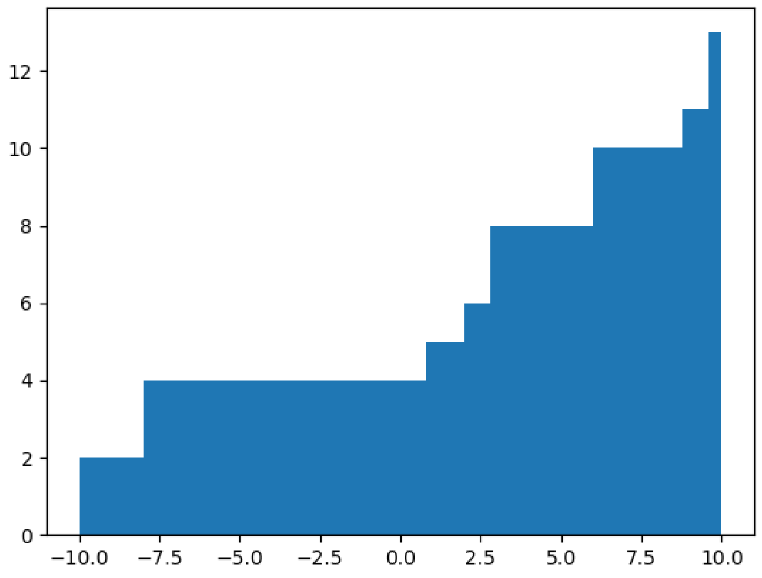
e.g.2

```
import matplotlib.pyplot as plt
import numpy as np
x=[-10,-8,3,6,9,10,-10,2,1,-8,3,6,10]
plt.xlim(-10,10)
plt.hist(x,cumulative=True)
plt.show()          #hist1
plt.hist(x,bins=50,cumulative=True)
plt.show()          #hist2
plt.hist(x,bins=100,cumulative=True)
plt.show()          #hist3
```

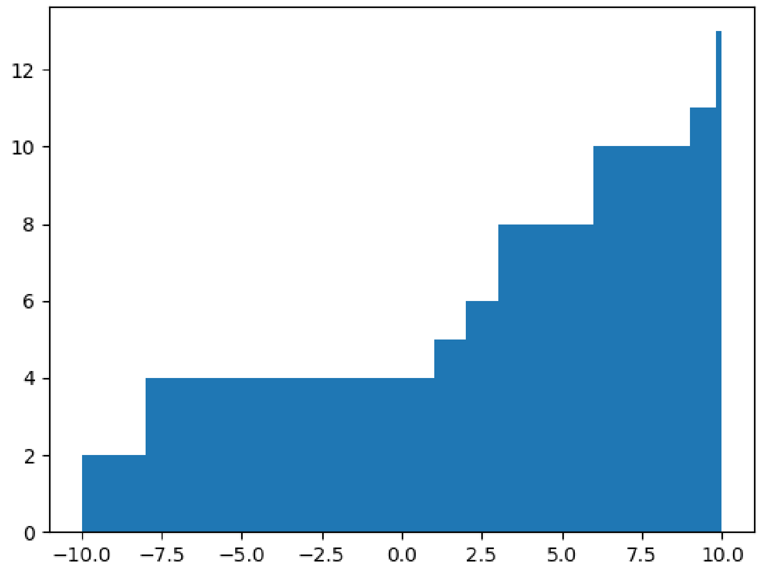
#hist1



#hist2



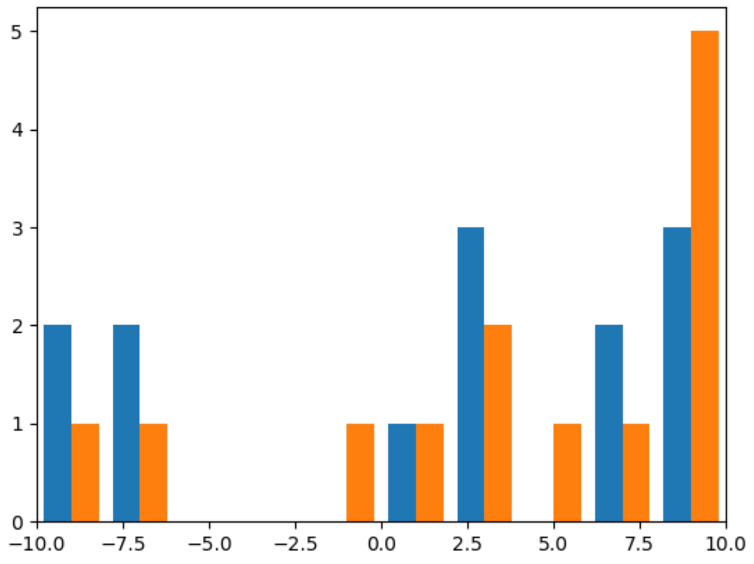
#hist3



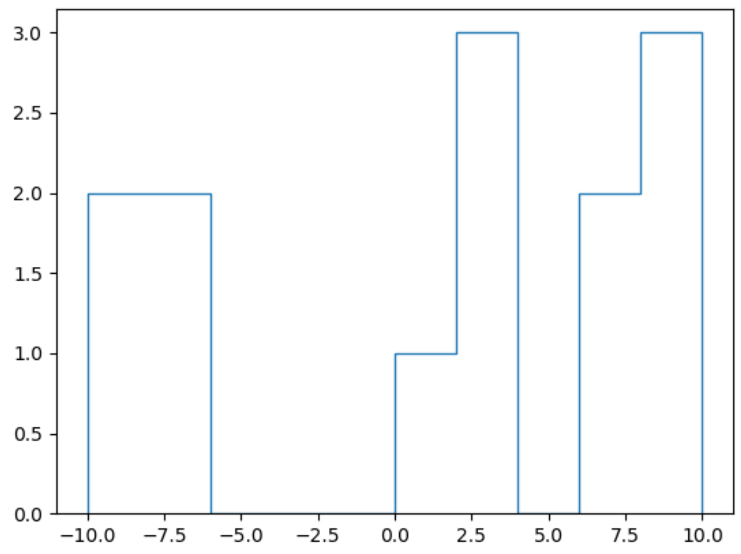
e.g.3

```
import matplotlib.pyplot as plt
import numpy as np
x=[-10,-8,3,6,9,10,-10,2,1,-8,3,6,10]
y=[8,10,9,2,1,5,-10,-2,3,-8,7,9,10]
plt.xlim(-10,10)
plt.hist([x,y])     #plotting two histogram - plot1
plt.show()
plt.hist(x,histtype='step')          #plot2
plt.show()
plt.hist(x,orientation='horizontal') #plot3
plt.show()
```

#plotting two histogram - plot1

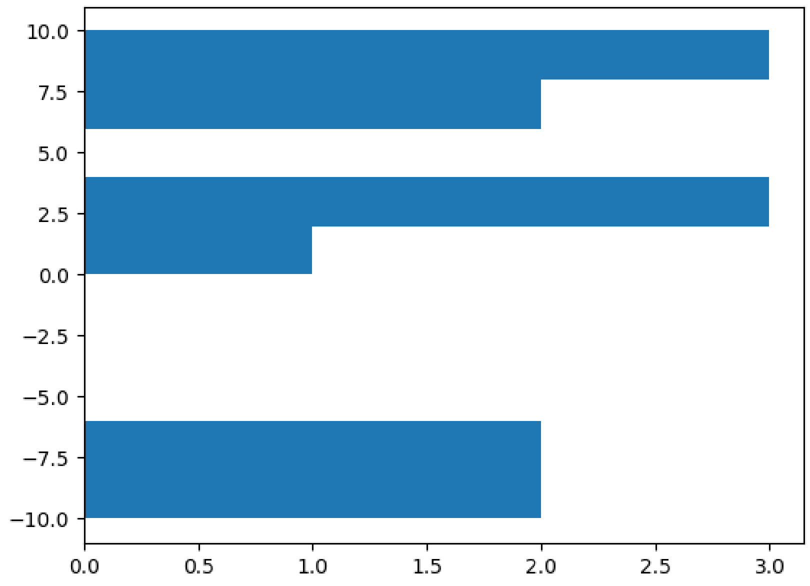


#plot2



3.0
2.5
2.0
1.5
1.0
0.5
0.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

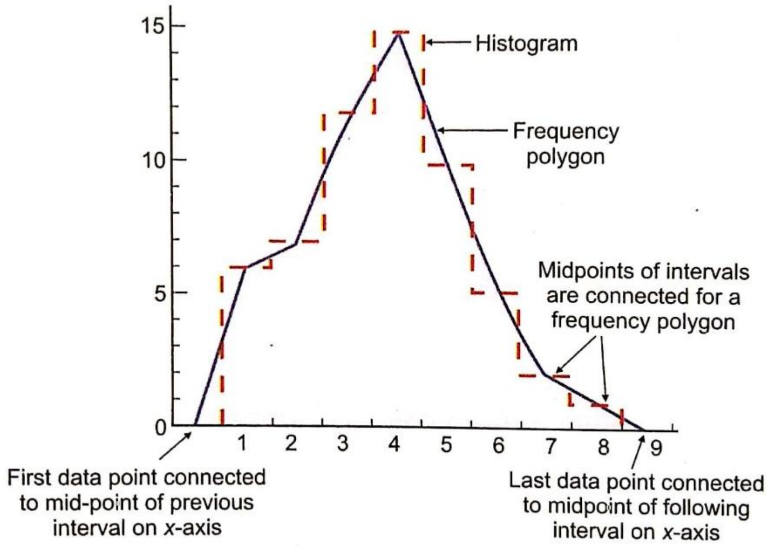
#plot3



10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0

Creating Frequency Polygons

- A frequency polygon is a type of frequency distribution graph.
- In a frequency polygon, the number of observations is marked with a single point at the midpoint of an interval. A straight line then connects each set of points.
- Frequency polygons make it easy to compare two or more distributions on the same set of axes.



15
10
5
0
1 2 3 4 5 6 7 8 9
Histogram
Frequency polygon
Midpoints of intervals are connected for a frequency polygon
First data point connected to mid-point of previous interval on x-axis
Last data point connected to midpoint of following interval on x-axis

Figure 4.2 Frequency polygon

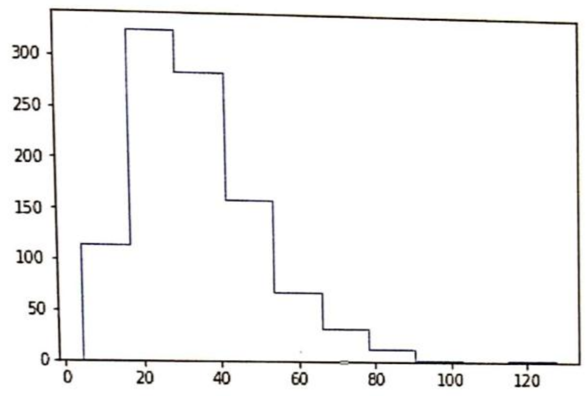
- Python's **pyplot** module of **matplotlib** provides no separate function for creating frequency polygon. Therefore, to create a frequency polygon, what you can do is:

(i) Plot a histogram from the data.
(ii) Mark a single point at the midpoint of an interval/bin.
(iii) Draw a straight lines to connect the adjacent points.
(iv) Connect first data point to the midpoint of previous interval on x-axis.
(v) Connect last data point to the midpoint of following interval on x-axis.

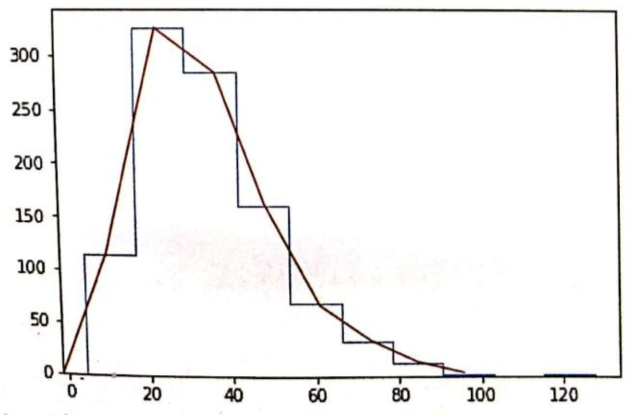
e.g.

(i) Create a step type histogram from the data. We have a Series namely **com** that stores some 1000 values. Plotting a step histogram from the same.

```
pl.hist(com, bins = 10, histtype = 'step')
```



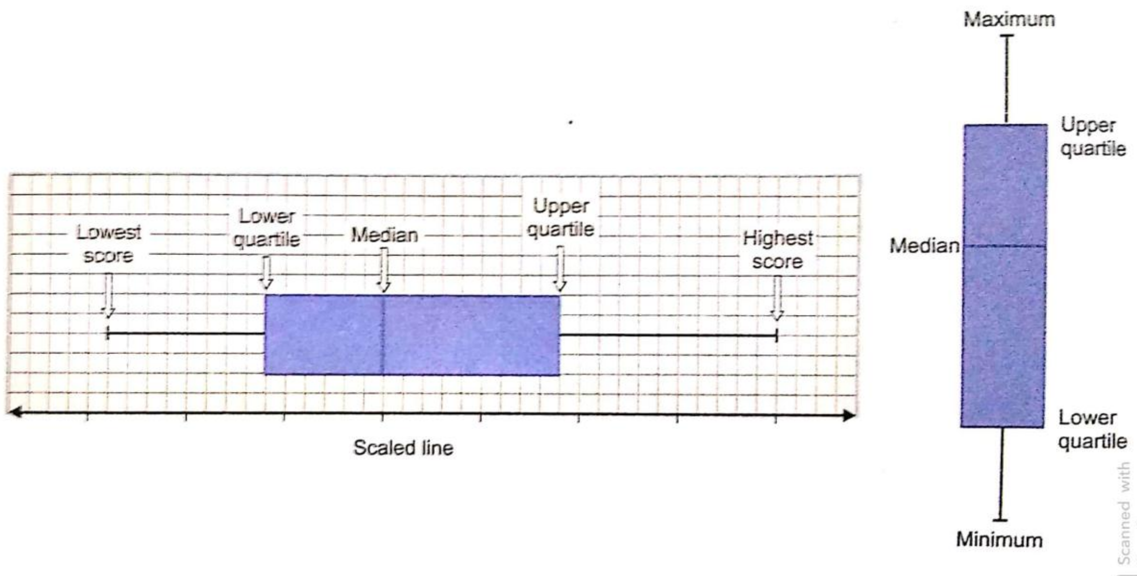
(*ii*) Join midpoints of each set of adjacent bins to create frequency polygon.



(*iii*) We have desired frequency polygon ready (*see* above)

Creating Box Plots

- A box plot uses ***five important numbers of a data range***: the *extremes* (the highest and the lowest numbers), the *median*, and the *upper and lower quartiles*, making up the five number summary.
- A box plot is used to show the range and middle half of ranked data. Ranked data is numerical data such as *numbers* etc. The middle half of the data is represented by the box. The highest and lowest scores are joined to the box by straight lines. The regions above the upper quartile and below the lower quartile each contain 25% of the data.



(*a*) Anatomy of a box plot.

- boxplot( ) method is used to create boxplots. The syntax is:

matplotlib.pyplot.boxplot(x , notch =None , vert=None , meanline = None, showmeans = None,
showbox = None)

Parameters:

x – Array or a sequence of vectors.

notch – bool,optional(False) ; If True, will produce a notched box plot. Otherwise, a rectangular boxplot is produced.

vert – bool, optional(True) ; If True (default), makes the boxes vertical. If false, everything is drawn horizontally.

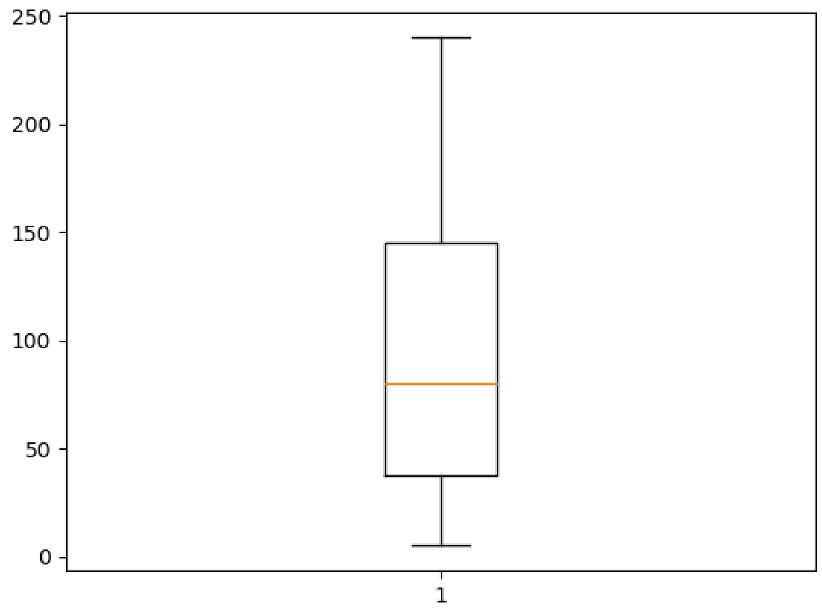
showbox – bool,optional(True) ; show the central box.

showmeans – bool, optional (False) ; Show the arithmetic means.

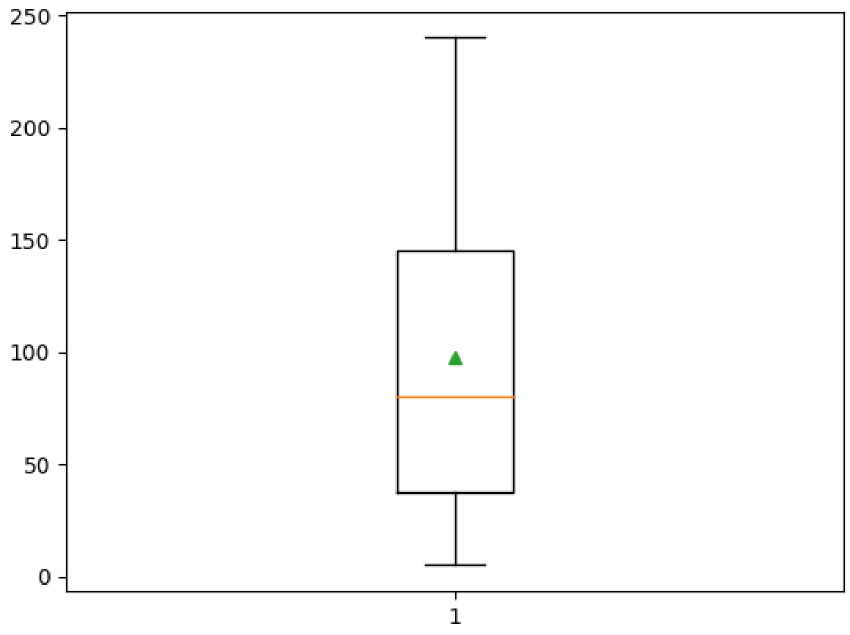
E.g

```
import matplotlib.pyplot as plt
import numpy as np
ary=[5,20,30,45,60,80,100,140,150,200,240]
plt.boxplot(ary)  #simple boxplot
plt.show()
plt.boxplot(ary,showmeans=True) #boxplot with mean
plt.show()
plt.boxplot(ary,showmeans=True,notch=True) #notched boxplot
plt.show()
plt.boxplot(ary,showbox=False) #boxplot without central box
plt.show()
```

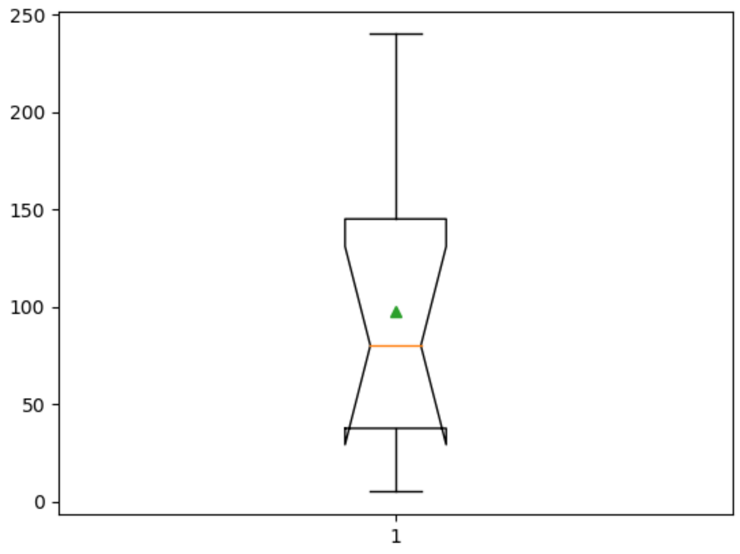
#simple boxplot



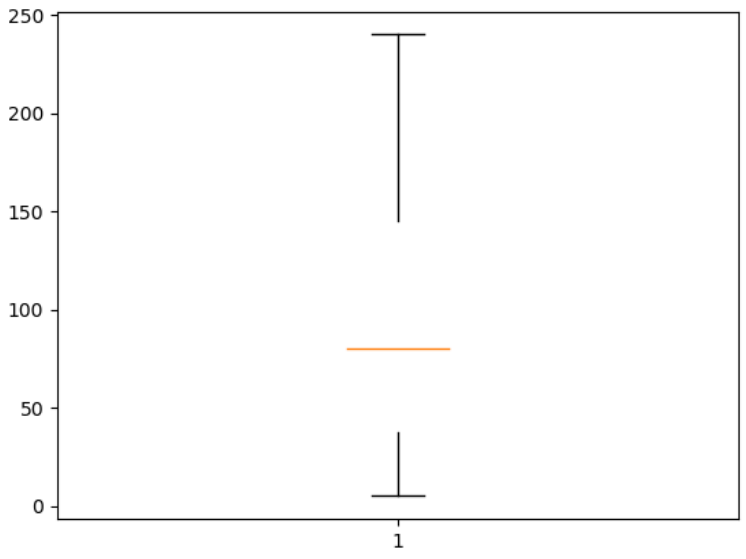
#boxplot with mean



#notched boxplot



#boxplot without central box



****