Page 1 Data Visualisation: - Data visualisation basically refers to the graphical or visual representation of information and data using visual elements like charts, graphs and maps etc.

Using Pyplot of MATPLOT LIBRARY:-

Pyplot is a collection of methods within **MATPLOT library** (of Python) which allows user to construct 2D plots easily and interactively.

INSTALLING AND IMPORTING MATPLOT LIBRARY

*** At first you have to open Command Prompt Then write this code –**
pip install matplotlib
*** After doing this process, Now you have to import MATPLOT library.**
For example: -
>>> import matplotlib.pyplot
or
>>> import matplotlib.pyplot as pl

• *Please note that if you have installed Python through Anaconda installer then Numpy, SciPy, Pandas, Matplotlib etc. Are by default installed with python, otherwise you need to install*

Using Numpy Arrays: -

NumPy ("Numerical Python" or "Numeric Python") is an open source module of python that offers functions and routines for fast mathematical computation on arrays and matrices. In order to use NumPy, you must import in your module by using a statement like:

import numpy as n

• A **Numpy** array is simply a grid that contains values of the same / homogeneous type (same type).

Example:

import **numpy** as np
list = [1,2,3,4]
al = np.array(list)
print(al)

• You can check type of a NumPy array

Example:

>>> type (al)
<class 'numpy.ndarray'>

• You can check data type of a NumPy arrays elements using <arrayname>.dtype

Example:

>>> al.dtype
dtype('int32')

Some useful way of creating NumPy Arrays:

(I) **Creating arrays with a numerical range using arange ():-** The arange () creates a NumPy arrays with evenly spaced values within a specified numerical range. It is used as:

<Array name>=numpy.arange (start, stop, step, dtype)

• The dtype specific the datatype for the NumPy array.

Example:

>>> import numpy as np
>>> ar = np.arange(1,7,2,np.float32)
>>> ar
array([1., 3., 5.], dtype=float32)

(II) **Creating arrays with a numerical range using linspace() :-** Sometimes, you need evenly spaced elements between two given limits. For this purpose, NumPy provides linspace () function to which you provide, the start value, the end value and number of elements to be generated for the ndarray. The syntax for using linspace () function is:

<Array name>=numpy.linspace (start, stop, <number of values to be generated>)

Example

>>> al = np.linspace(2.5,5,6)
>>> al
array([2.5, 3. , 3.5, 4. , 4.5, 5.])

CREATING CHARTS WITH MATPLOTLIB LIBRARY'S PYPLOT INTERFACE

As stated earlier make sure to **import matplotlib.pyplot** library interface and other required libraries before you start using any of their functions.

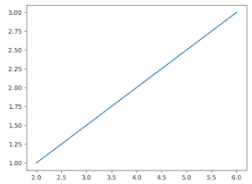
Line Chart: -

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

• The PyPlot interface offers **plot()** function for creating a line graph.

Example:

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



Output: -

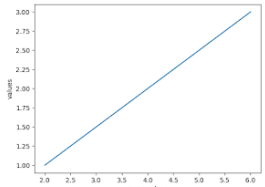
• You can set x- axis and y-axis label using functions xlabel() and ylable() respectively.

Syntax:

<matplotlib.pyplot or its alias>. xlabel ("label name in string form")
<matplotlib.pyplot or its alias>. ylabel ("label name in string form")

Example

```
import matplotlib.pyplot as pl
a = [1, 2, 3]
b = [2, 4, 6]
pl.plot(b, a)
pl.xlabel("some values")
pl.ylabel("values")
pl.show()
```



Output: -

Changing Line Style, Line Width and Line Color in a Line chart:-

To change line Color: -

You can specify the color code next to data being plotting in plot () function as shown below:

Syntax:

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

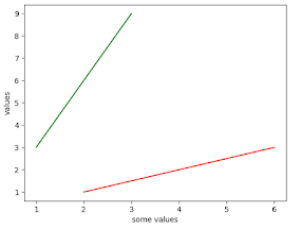
Some different Colors codes: -

Character	Color
'r'	red
'b'	blue
'g'	green
'm'	magenta
'y'	yellow
'k'	black
'c'	cyan
'w'	White

• In addition, you can also specify Colors in many other ways, including full **Color names** ('red', 'green' ,etc.), hex string ('**#008000**')

For example:

```
import matplotlib.pyplot as pl
a = [1, 2, 3]
b = [2, 4, 6]
c= [3, 6, 9]
pl.plot(b, a, "r")
pl.plot(a, c, "g")
pl.xlabel("some values")
pl.ylabel("values")
pl.show()
```



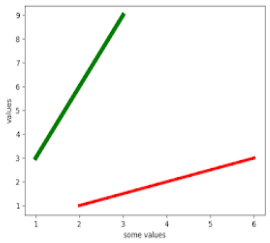
Output: -

To change Line Width: -

page 2 You can give addition argument in plot() as linewidth =<width>

For example:

```
import matplotlib.pyplot as pl
a = [1, 2, 3]
b = [2, 4, 6]
c = [3, 6, 9]
pl.plot (b, a, "r", linewidth = 4)
pl.plot (a, c, "g", linewidth = 6)
pl.xlabel ("some values")
pl.ylabel ("values")
pl.show( )
```



Output: -

To change the Line style: -

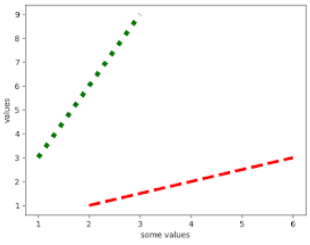
You can add following additional optional argument in plot() function:

linestyle = ['solid' | 'dashed', 'dashdot', 'dotted']

For example:

```
import matplotlib.pyplot as pl
a = [1, 2, 3]
b = [2, 4, 6]
c = [3, 6, 9]
pl.plot (b, a, "r", linewidth = 4, linestyle = '--')
pl.plot (a, c, "g", linewidth = 6, linestyle = 'dotted')
pl.xlabel ("some values")
pl.ylabel ("values")
pl.show( )
```

Output: -



To change Marker Type, Size and Color: -

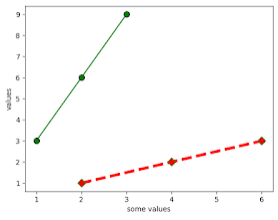
To change marker type, its size and color, you can give following additional optional arguments in plot() function.

marker = <valid marker type >, marker size =<in points>, markercolor =<valid color>

Marker type for plotting

For example:

```
import matplotlib.pyplot as pl
a = [1, 2, 3]
b = [2, 4, 6]
c = [3, 6, 9]
pl.plot (b, a, "r", linewidth = 4, linestyle = '--', marker = 'D', markersize = 8, markeredgecolor = 'g')
pl.plot (a, c, "g", marker = 'o', markersize = 8, markeredgecolor = 'k')
pl.xlabel ("some values")
pl.ylabel ("values")
pl.show( )
```

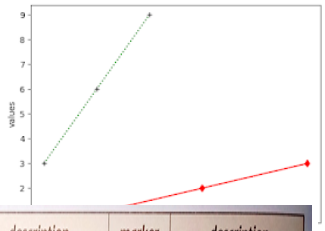


Output: -

• You can combine the **marker type with color code**.

For example

```
import matplotlib.pyplot as pl
a = [1, 2, 3]
b = [2, 4, 6]
c = [3, 6, 9]
pl.plot (b, a, "rd", linestyle = 'solid')
pl.plot (a, c, "g+", linestyle = ':', markeredgecolor = 'k')
pl.xlabel ("some values")
pl.ylabel ("values")
pl.show()
```



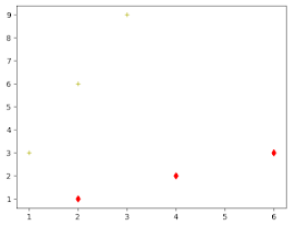
marker	description	marker	description	marker	description
'.'	point marker	's'	square marker	'3'	tri_left marker
','	pixel marker	'p'	pentagon marker	'4'	tri_right marker
'o'	circle marker	'*'	star marker	'v'	triangle_down marker
'+'	plus marker	'h'	hexagon1 marker	'^'	triangle_up marker
'x'	x marker	'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

Output: -

• When you do not specify markeredgecolor separately in plot(), the marker takes the same color as the line. Also, if you do not specify the linestyle separately along with linecolor-and-markerstyle-combination-string (eg, 'r+'), Python will only the markers and not the line. To get the line, specify linestyle argument

For example:

```
import matplotlib.pyplot as pl
a = [1, 2, 3]
b = [2, 4, 6]
c = [3, 6, 9]
pl.plot (b, a, "rd")
pl.plot (a, c, "g+", markeredgecolor = 'y')
pl.show( )
```



Output: -

• **So full syntax of plot () as:-**

plot (<data1>,<data2>,<color code and marker type>,<linewidth>,<linestyle>,<marker type>,<markersize>,<markeredgecolor>)

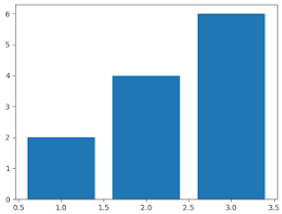
Bar Chart: -

A bar graph / chart is a graphical display of data using bars of different heights.

• A bar chart can be drawn vertically or horizontally using bars of different heights/widths PyPlot offers bar () function to create a bar chart.

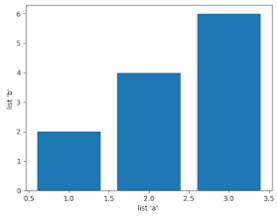
For example:

```
import matplotlib.pyplot as p
a = [1, 2, 3]
b = [2, 4, 6]
pl.bar (a, b)
```



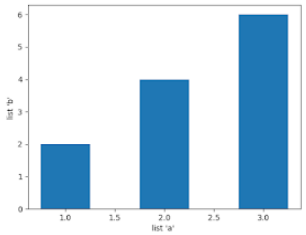
page 3pl.show( )

• If you want to specify x-axis label and y-axis label, then you need to give command like:
matplotlib.pyplot.xlabel ("label name in string form")
matplotlib.pyplot.ylabel ("label name in string form")
For example:
import matplotlib.pyplot as pl
a = [1, 2, 3]
b = [2, 4, 6]
pl.bar (a, b)
pl.xlabel("list 'a' ")
pl.ylabel("list 'b' ")
pl.show( )

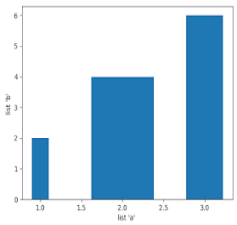


Changing Widths of the Bars in Bar Chart:

(I) **To specify common width for all bars:**
<matplotlib.pyplot>.bar (<x-sequence>, <y-sequence> width=<float value>)
For example:
import matplotlib.pyplot as p
a = [1, 2, 3]
b = [2, 4, 6]
pl.bar (a, b, width = 0.5)
pl.xlabel("list 'a' ")
pl.ylabel("list 'b' ")
pl.show( )

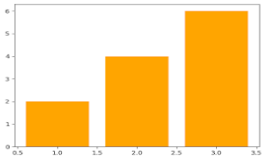


(II) **To specify different widths for different bars of chart:**
You can specify width argument having a **sequence** (such as lists or tuples) containing widths for each of the bars in the bar() function as,
<matplotlib.pyplot>.bar (<x- sequence>, <y- sequence>, width= <width values sequence>)
• *Please note that the width sequence must have widths for all the bars otherwise Python will report an error.*
For example:
import matplotlib.pyplot as pl
a = [1, 2, 3]
b = [2, 4, 6]
pl.bar (a, b, width = [0.21, 0.76, 0.45])
pl.xlabel("list 'a' ")
pl.ylabel("list 'b' ")
pl.show( )

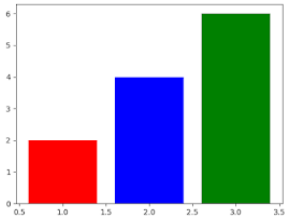


Changing Colors of the Bars in a Bar Chart:

(I) **To specify common Color for all bars: -**
You can specify color argument having a valid color code /name in the bar() function.
Syntax:-
<matplotlib.pyplot>.bar (<x- sequence>, <y- sequence>, color = <color code/ name>)
For example:
import matplotlib.pyplot as pl
a = [1, 2, 3]
b = [2, 4, 6]
pl.bar (a, b, color = 'orange')
pl.show( )

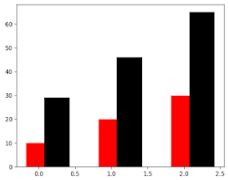


(II) **To specify different Colors for different bars of a bar chart: -**
You can specify color argument having a **sequence** (such as lists or tuples) containing Colors for each of the bars, in the bar() function.
Syntax:-
<matplotlib.pyplot>.bar (<x- sequence>, <y- sequence>, color = <color name/ codes sequence>)
• *Please note that the color sequence must have color for all bars otherwise Python will report an error.*
For example:
import matplotlib.pyplot as pl
a = [1, 2, 3]
b = [2, 4, 6]
pl.bar (a, b, color = ['r', 'b', 'g'])
pl.show( )



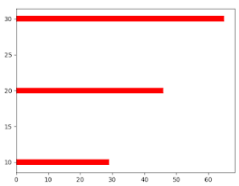
Creating Multiple Bars Chart:-

As such PyPlot does not provide a specific function for this, But you can always create one exploiting the width and color argument of bar().
Let's see how:-
(I) Decide number of X points.
(II) Decide thickness of each bar and accordingly adjust X points on x-axis.
(III) Give different color to different data range.
(IV) The width argument remains the same for all ranges being plotted.
(V) Plot using bar() for each range separately.
For example:
import matplotlib.pyplot as pl
import numpy as np
a = [10, 20, 30]
b = [29, 46, 65]
x = np.arange(3)
pl.bar (x, a,color = 'r' , width = 0.35)
pl.bar (x+0.25, b, color= 'k', width = 0.35)
pl.show( )



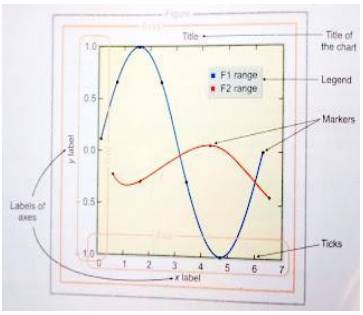
Creating a Horizontal Bar Chart:-

To create a horizontal bar chart, you need use **barh()** function (bar horizontal), in place of bar().
• The label that you gave to x-axis in an bar() will become y-axis label in barh() and **vice versa.**
For example:
import matplotlib.pyplot as pl
a = [10, 20, 30]
b = [29, 46, 65]
pl.barh (a, b, color = 'r')
pl.show( )



CUSTOMIZING THE PLOT:-

Page 4 Anatomy of a Chart: -



Let us know about various parts 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: -** The axes define the area (mostly rectangular in shape for simple plots) 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 tor 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.

Setting X & Y Limits: -

When you specify and X and Y range for plotting, PyPlot automatically tries to find best fitting range for X-axis and Y-axis depending on the data being plotted. But sometimes, you may need to have own limits specified for X and Y axes. For this, you can use xlim() and ylim() functions to set limits for axis and y-axis respectively.

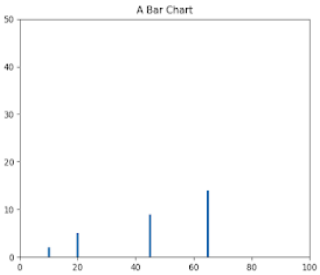
Both xlim() and ylim() are used as per following format:

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

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

For example:

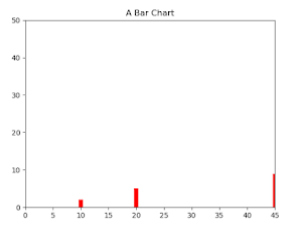
```
import matplotlib.pyplot as pl
pl.title("A Bar Chart")
a = [10, 20, 45, 65]
b = [2, 5, 9, 14]
pl.xlim(0,100)
pl.ylim(0,50)
pl.bar(a , b)
pl.show( )
```



• If you have set X-axis or Y-axis limits which are not compatible with the data values being plotted, you may either get incomplete plot or the data being plotted is not visible at all.

For example:

```
import matplotlib.pyplot as p
pl.title("A Bar Chart")
a = [10, 20, 45, 65]
b = [2, 5, 9, 14]
pl.xlim(0,45)
pl.ylim(0,50)
pl.bar(a , b, color = 'r')
pl.show( )
```



Setting Ticks for Axes: -

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

To set own tick marks:

• For X-axis, you can use xticks() function as per format :

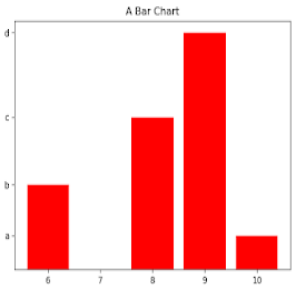
xticks (<sequence containing tick data points>, [<Optional sequence containing tick labels>])

• For Y axis, you can use yticks() function as per format:

yticks(<sequence containing tick data points>, [<Optional sequence containing tick labels>])

For example:

```
import matplotlib.pyplot as pl
pl.title("A Bar Chart")
a = [10, 6, 8, 9]
b = [2, 5, 9, 14]
pl.bar(a , b, color = 'r')
pl.yticks(b,['a', 'b', 'c', 'd'])
pl.show( )
```



Adding Legends:-

When we plot multiple ranges on a single plot, it becomes necessary that legends are specified. Recall that a legend is a color or mark linked to a specific data range plotted. To plot a legend, you need to do two things:

(I)In the plotting functions like plot( ), bar( ) etc., give a specific label to data range using argument **label**.

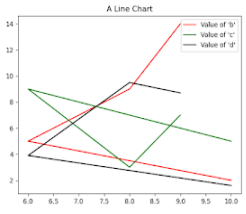
(ii) Add legend to the plot using **legend()** as per format:

<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.

For example:

```
import matplotlib.pyplot as pl
pl.title("A Line Chart")
a = [10, 6, 8, 9]
b = [2, 5, 9, 14]
c = [5, 9, 3, 7]
d = [1.6, 3.9, 9.5, 8.7]
pl.plot(a , b, color = 'r', label = "Value
pl.plot(a , c, color = 'g', label = "Value of 'c'")
pl.plot(a , d, color = 'k', label = "Value of 'd'")
pl.legend(loc= 'upper right')
pl.show( )
```



Saving a Figure: -

If you want to save a plot created using pyplot functions for later use or for keeping records,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, .eps etc. Specify the format as file extension

• Also, while specifying the path, use double slashes to suppress special meaning of single slash character.

For example:

```
plt.savefig ("graph.pdf") # save the plot in current directory
plt.savefig ("c:\\data\\graph.png") # save the plot at the given path
```