Matpotlib

- This document is all about Matplotlib , basic data visualization using matplotlib tool of python programming language. I have discussed Matplotlib object hierarchy , various plot types with Matplotlib and customization technique associated with Matplotlib.

Table of Contents

1. Introduction

When we want to convey some information to others , there are several ways to do so. The process of conveying the information with the help of plots and graphics is called data visualization . The plots and graphics take numerical data as input and display

output in the form of charts , figures and tablets . it helps to analyze and visualize the data clearly and make concreate decisions. It makes complex data more accesible and understandable. The goal of data visualization is to communicate innformation in a clear and efficient manner.

In this project , I shed some light on Matplolib , Which is the basic data visualization tool of python programming language. Python has diffrent data visualization tools available which are suitable for different purposes.

3. Introduction to Matplotlib

Matplotlib is the basic ploting library of python programming language . it is the most prominent tool among python visualization packages. Matplotlib is highly efficient in performing wide range of. tasks . it can produce publication quality figures in a variety of formats. it can export visualization to all of the common formats like PDF , SVG , JPG , PNG , BMP and GIF . It can create popular visualization types - line plot , scatter plot , histogram , bar chart , error charts , pie chart , box plot , and many more types of plot . Matplotlib also supports 3d plotting .Many python libraries are built on top of matplotlib .For example , Pandas and seaborn are built on matplotlib .They allow to access Matplotlib's methods with less code.

The project Matplotlib was started by john hunter in 2002 . Matplotlib was originally started to visualize to visualize Electrocortocography(ECoG) data of epilepsy patients during post doctoral research in neurobiology . the open source tool matplotlib emerged as the most widely used plotting library for python programming language . it was used for data visualization during landing of the phoenix spacecraft in 2008.

4. Import Matplotlib

Before , we need to actually start using Maplotlib, we need to import it . We can import Matplotlib as follows:-

Import matplotlib

Most of the time, we have to work with **pyplot** interface of Matplolib . So , I will import **pyplot** interface of Matplotlib as follows:-

import matplotlib.pyplot

To make things even simpler , we will use shorthand for matplotlib imports as follows:-

import matplotlib.pyplot as plt

In [6]:
```
# Import dependencies

import numpy as np

import pandas as pd
```

In [7]:
```
import matplotlib.pyplot as plt
```

5. Displaying Plots in Matplotlib

Viewing the Matplotlib plot is context based . the best usage of matplotlib differs depending on how we are using it .There are three applicable contexts for viewing the plots . the Three applicable contexts are using plotting from a script , Plotting from an ipython shell or plotting from a Jupyter notebook.

Plotting from a script

If we are using Matplotlib from within a script , then the plt.show() command is of great use . it starts an event loop , looks for all currrently active figure objects, and opens one or more interactive windows that display the figure or figures.

The plt.show() command should be used only once per python session . It should be used only at the end of the script . Multiple plt.show() commands can lead to unpredictable results and should mostly be avoided.

Plotting from an ipython shell

We can use Matplotlib interactively within an ipython shell . ipython works well with Matplotlib if we specify Matplotlib mode . to enable this mode , we can use the %matplotlib magic command after starting ipython . Any plt plot command will cause a figure window to open and further commands can be run to update the plot.

Plotting from a jupyter notebook

The jupyter notebook (formerly known as the ipython notebook) is a data analysis and visualization tool that provides multiple tools under one roof . It provides code execution graphical plots , rich text and media display , mathematics formula and much more facilities into a single executable document .

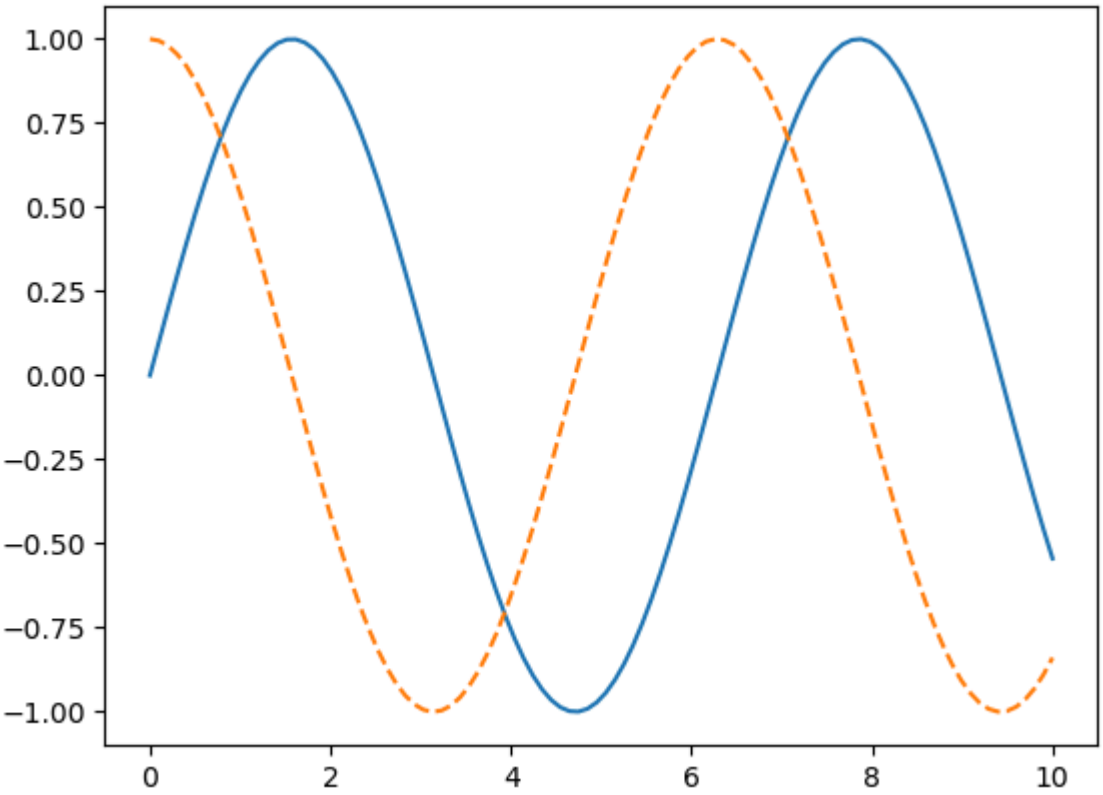
interactive plottting within a jupyter notebook can be done with the %matplotlib command . There are two possible options to work with graphics in jupyter notebook . These are as follows : -

- **%matplotlib notebook** -This command will produce interactive plots embedded within the notebook.
- **% matplotlib inline -** It will output static images of the plot embedded in the notebook .

After this command (it needs to be done only once per kernel per session), any cell within the notebook that creates a plot will embed a PNG image of the graphic.

In [9]:
```
%matplotlib inline

x1 = np.linspace(0,10,100) # this linspace() creates the values within 0 to 10 w

# crate a plot figure
```

```
fig = plt.figure()

plt.plot(x1, np.sin(x1), '-')

plt.plot(x1 , np.cos(x1) , '--')

plt.show()
```



Matplotlib API Overview

Matplotlib has two APIs to work with . A MATLAB -style state based interface and a more powerful object oriented (OO) interface . The Former MALAB style state based interface is called **pyplot interface** and the latter is called **Object -Oriented** interface.

There is a third interface also called **pylab** interface . it merges pyplot (for plotting) and Numpy (for mathematical functions) together in an environment closer to matlab this is considered bad practise nowadays . so the use of `pylab` is strongly discouraged and hence , I will not discuss it any further .

Matlpotlib.Pyplot is stateful because the undelying engine keeps track of the current figure and plotting area information and plotting funtions change that information .To make it clearer , we did not use any object references during our plotting we just issued a pyplot command , and the changes appeared in the figure .

we can get a refereance to the current figure and axs using the following commands -

plt.gcf() # get current figure

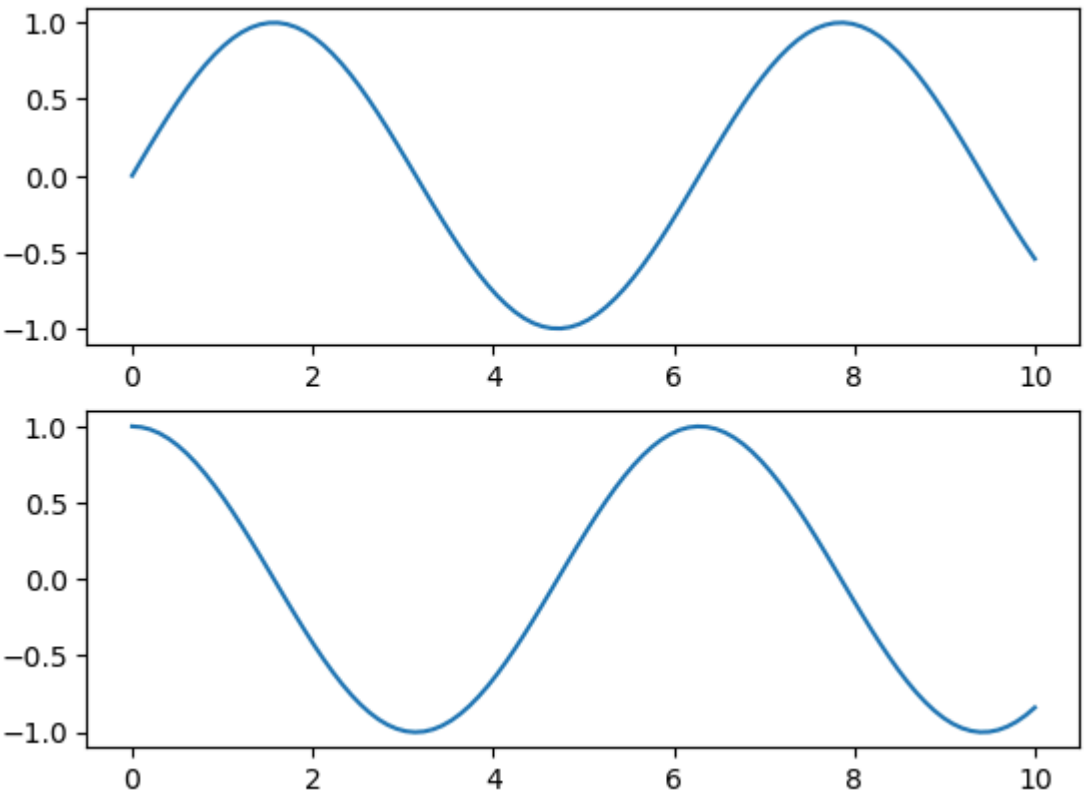
plt.gca() # getr current axes

Matplotlib.pyplot is a collection of commands and functions that make matplotlib behave like matlab (for plotting) The MATLAB - style tools are contained in the pyplot (plt) interface .

This is really helpful fo interactive plotting , because we can issue a command and see the result immediately . BUt it is not suitable for more complicated cases. for these case we have anotheer interface called object oriented interface , described later

the following code produces sine and cosine curves using pyplot api.

In [11]:
```
#create a plot figure

plt.figure()

# create the first of two panels and set current axis

plt.subplot(2,1,1) # rows , columns , panel number)
plt.plot(x1, np.sin(x1))
# create the second of two panels and set current axis
plt.subplot(2,1,2) #rows ,columns , panel number)
plt.plot(x1,np.cos(x1));
```

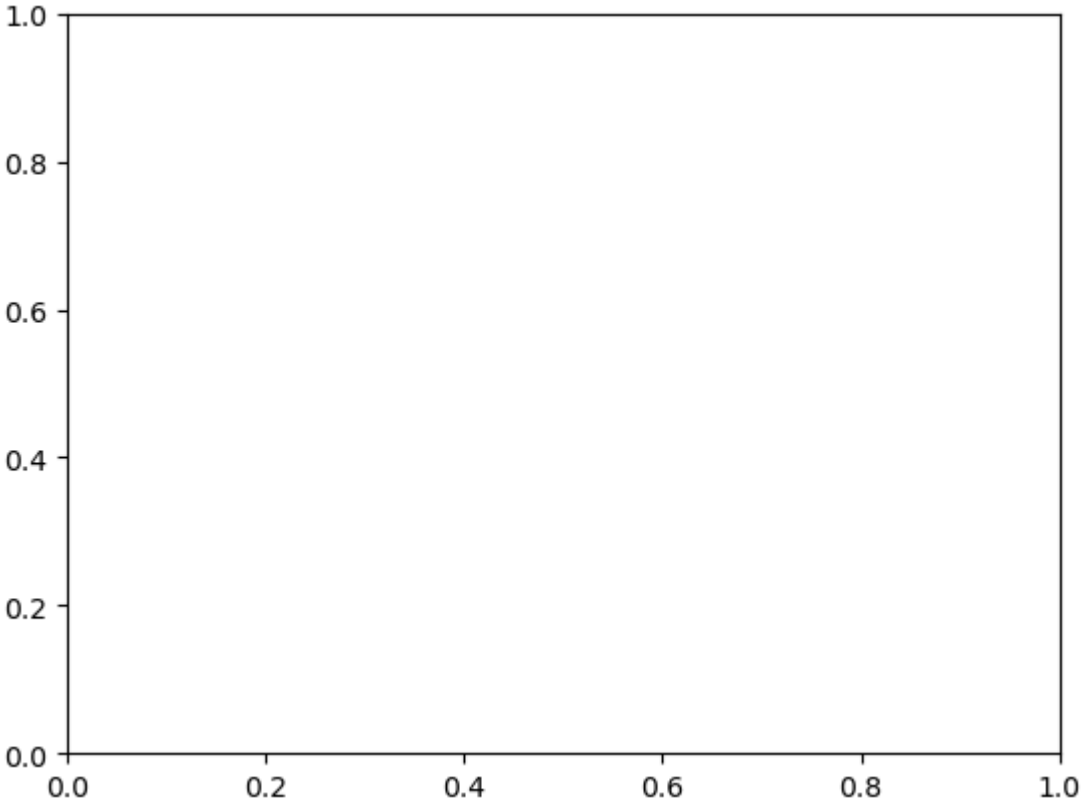


In [12]:
```
# get current figure information

print(plt.gcf())
```

```
Figure(640x480)
<Figure size 640x480 with 0 Axes>
```

In [13]:
```
print(plt.gca())
```

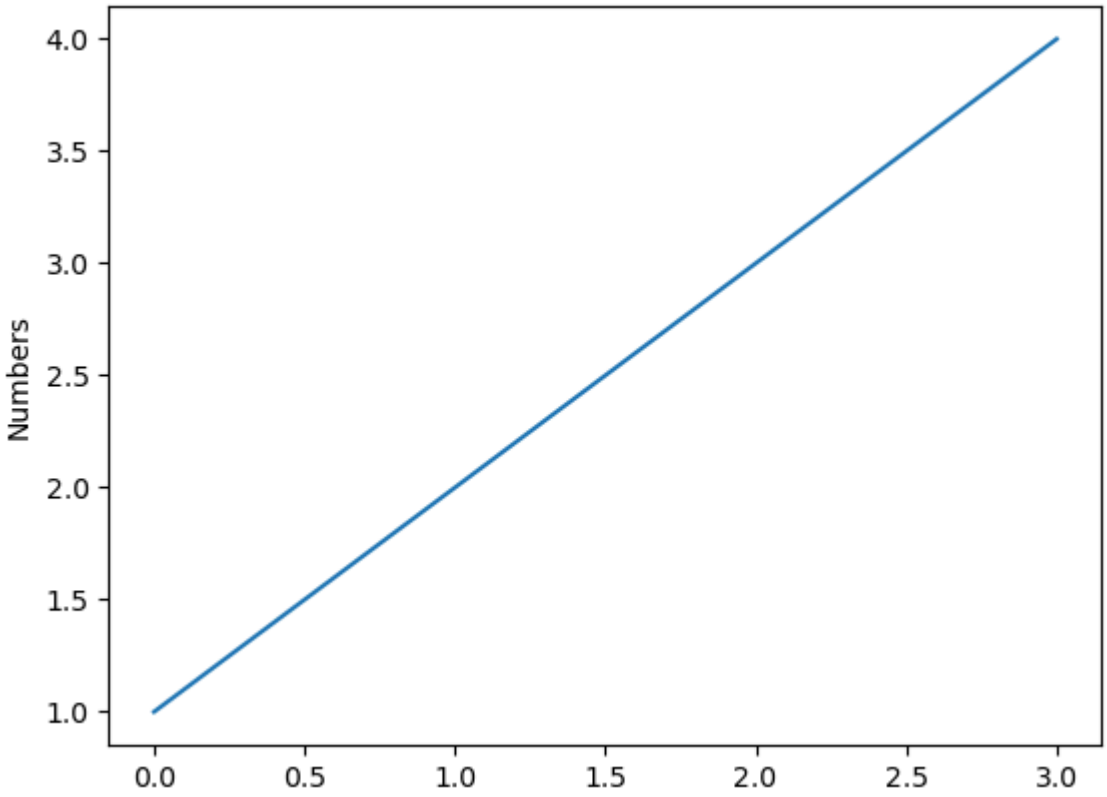
```
Axes(0.125,0.11;0.775x0.77)
```



Visualization with Pyplot

Generating visualization with pyplot is very easy. The x - axis values ranges from 0-3 and the y - axis from 1-4 if we provide a single list or array to the plot() command matplotlib assumes it is a sequence of y values , and automatically generates the x values . Since python ranges start with 0 the default x vector has the same length as y but start with 0. Hence the x data are[0,1,2,3] and y data are[1,2,3,4].

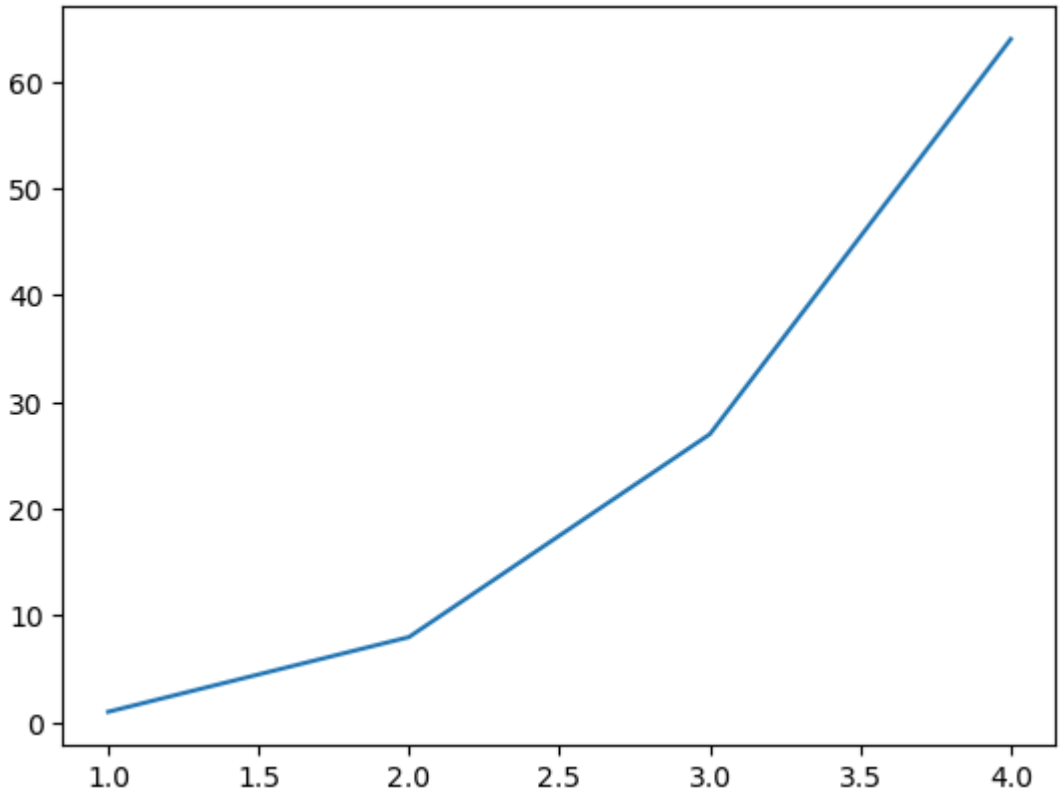
In [15]:
```
plt.plot([1,2,3,4])
plt.ylabel('Numbers')
plt.show()
```



plot() - A versatile command

plot() is a versatile commnd . it will take an arbitrary number of arguments . for example , to plot x versus y , we can issue the following command : -

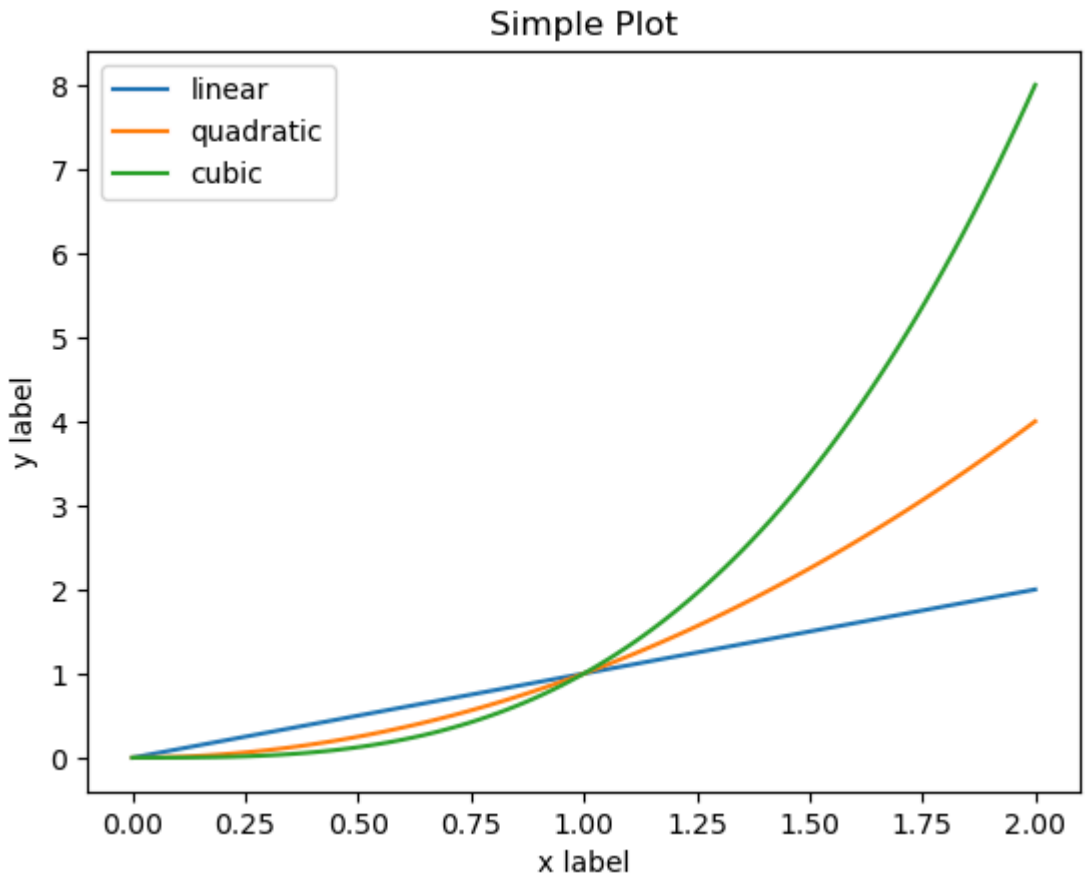
In [17]:
```
import matplotlib.pyplot as plt

plt.plot([1,2,3,4],[1,8,27,64])
plt.show()
```



State Machine Interface

Pyplot provides the state - machine interfface to the underlying object - oriented plotting library . the state machine implicitly and automatically creates figures and axes to achieve the desired plot . for example:

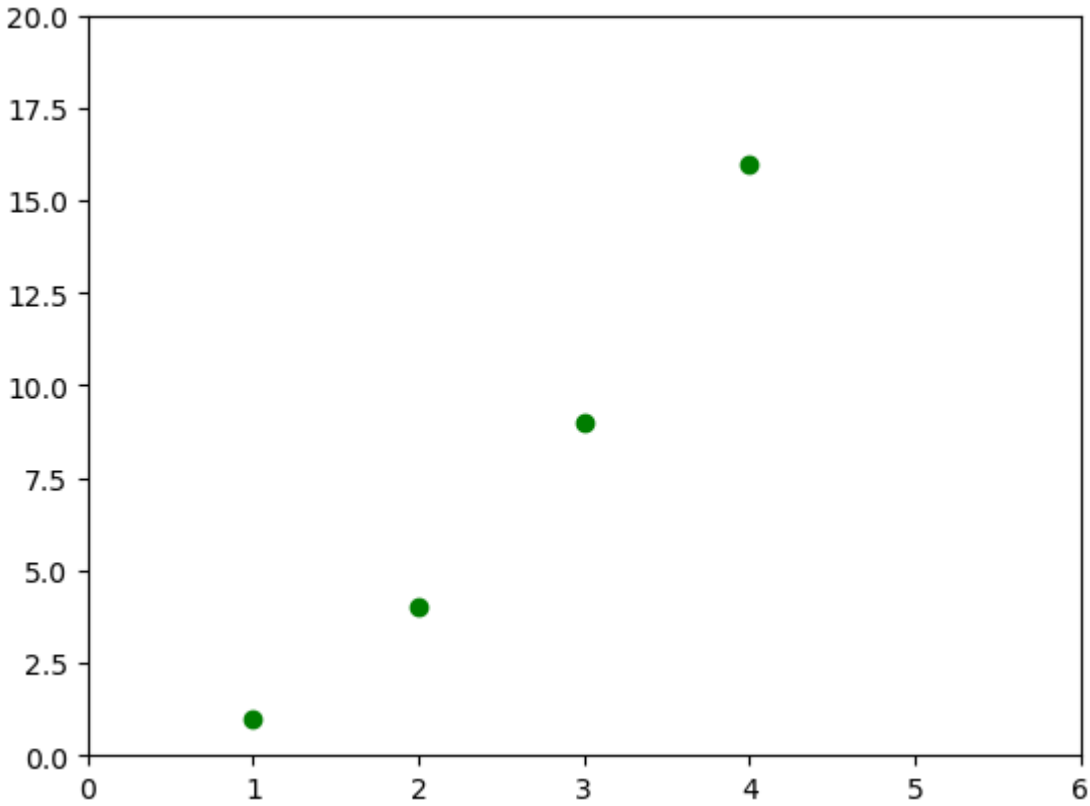
In [19]:
```
x = np.linspace(0,2,100)

plt.plot(x,x,label = 'linear')
plt.plot(x,x**2, label = 'quadratic')
plt.plot(x,x**3,label = 'cubic' )

plt.xlabel('x label')

plt.ylabel('y label')

plt.title("Simple Plot")

plt.legend()

plt.show()
```



Formatting the style of plot

for every x,y pair of arguments , there is an optional third argument which is the format string that indicates the color and line type of the plot. the letters and symbols of the format string are form matlab. we can concatenate a color string with a line style string the default format string is 'b-' , which is a solid blue line. for example , to plot the above line with red circles , we would issue the following commands :-

In [21]:
```
plt.plot([1,2,3,4],[1,4,9,16],'go')

plt.axis([0,6,0,20])

plt.show()
```

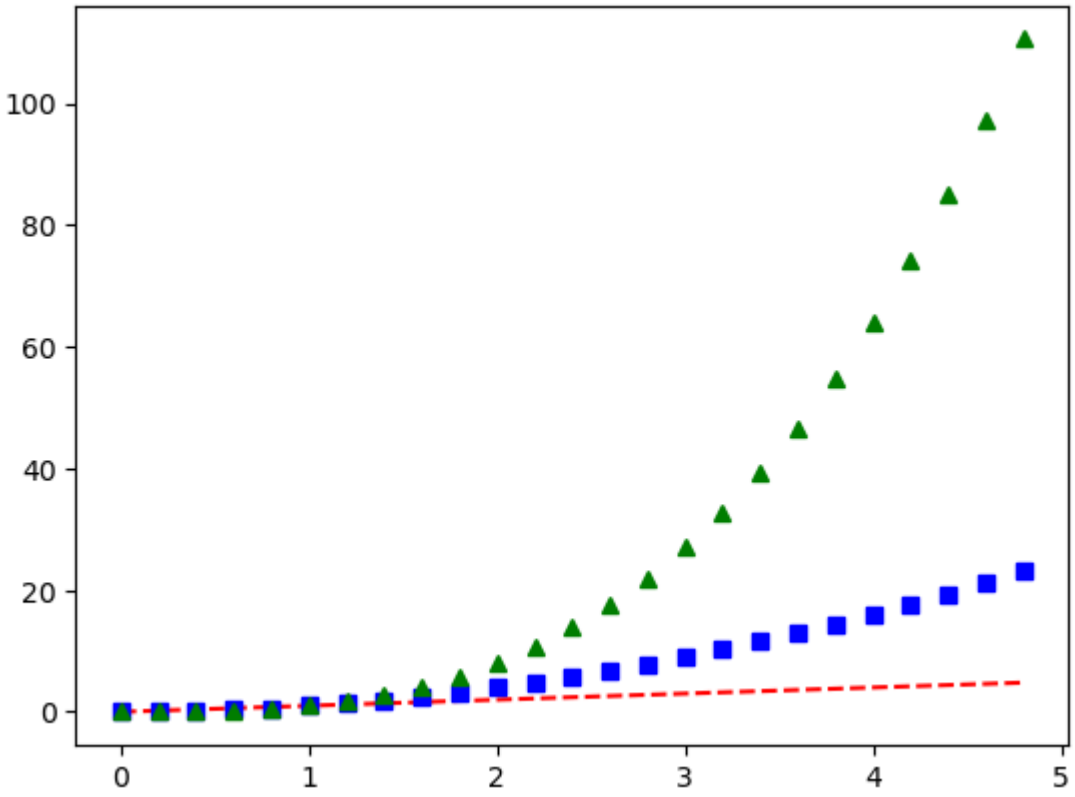


The **axis()** command in the exaple above takes a list of [xmin,xmax,ymin,max] and specifies the viewport of the axes.

Working with Numpy arrays

Generally , we have to work woth numpy arrays . All sequences are converted to Numpy arrays internally. The below example illustrates plotting several lines with different format style in one command using arrays.

In [23]:
```
t = np.arange(0.,5.,0.2)

# red dashes , blue squares and green triangles

plt.plot(t,t,'r--',t,t**2,'bs' , t, t**3,'g^')
plt.show()
```



Object-Oriented API

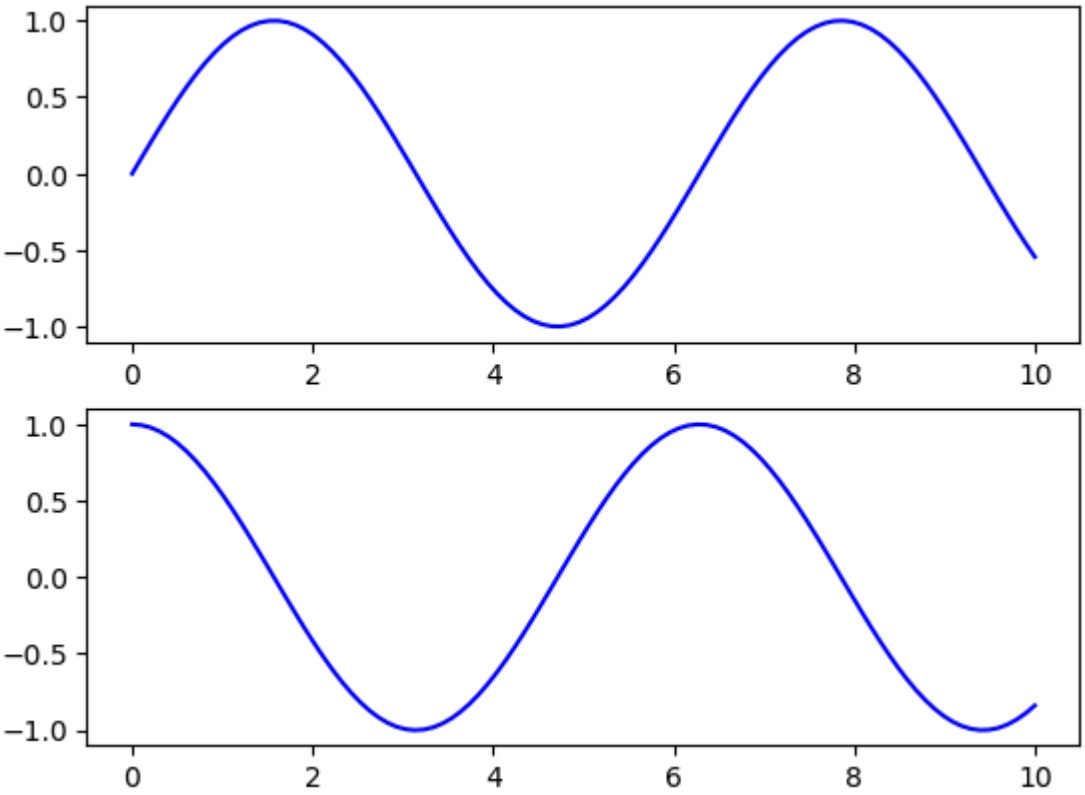
The object Oriented Api is available for more complex plotting situations . it allows us to exercise more comntrol over the figure . In Pyplot API, we depend on some notion of an "active" figure or axes. But , in the object-oriented api the plotting function are methods of explicit figure and axes objects.

Figure is the top level container for all the plot elements. we can think of the figure object as a box like container containing one or more axes.

the axes represents an individual plot. The axes object contain smaller objects such as axis,tick marks , lines , legends , title and text-boxes.

The following code produces sine and cosine curves using object oriented Api.

In [25]:
```
# first create grid of plots
# ax will be an array of two axes objects

fig, ax = plt.subplots(2)

# call plot() method on the appropriate object

ax[0].plot(x1, np.sin(x1), 'b-')
ax[1].plot(x1,np.cos(x1), 'b-');
```



Objects and Reference

The main idea with the **Object Oriented API** is to have objects that one can apply functions and actions on. The real advantage of this approach becomes apparent when more than one figure is created or when a figure contains more than one subplot.

We create a reference to the figure instance in the fig variable . Then we create a new axis instance axes using the add_axes method in the figure class intance fig as follows:-

In [27]:
```
fig = plt.figure()

x2 = np.linspace(0,5,10)

y2 = x2**2

axes = fig.add_axes([0.1,0.1,0.8,0.8])

axes.plot(x2,y2,'r')

axes.set_xlabel('x2')
axes.set_ylabel('y2')
axes.set_title('title');
```

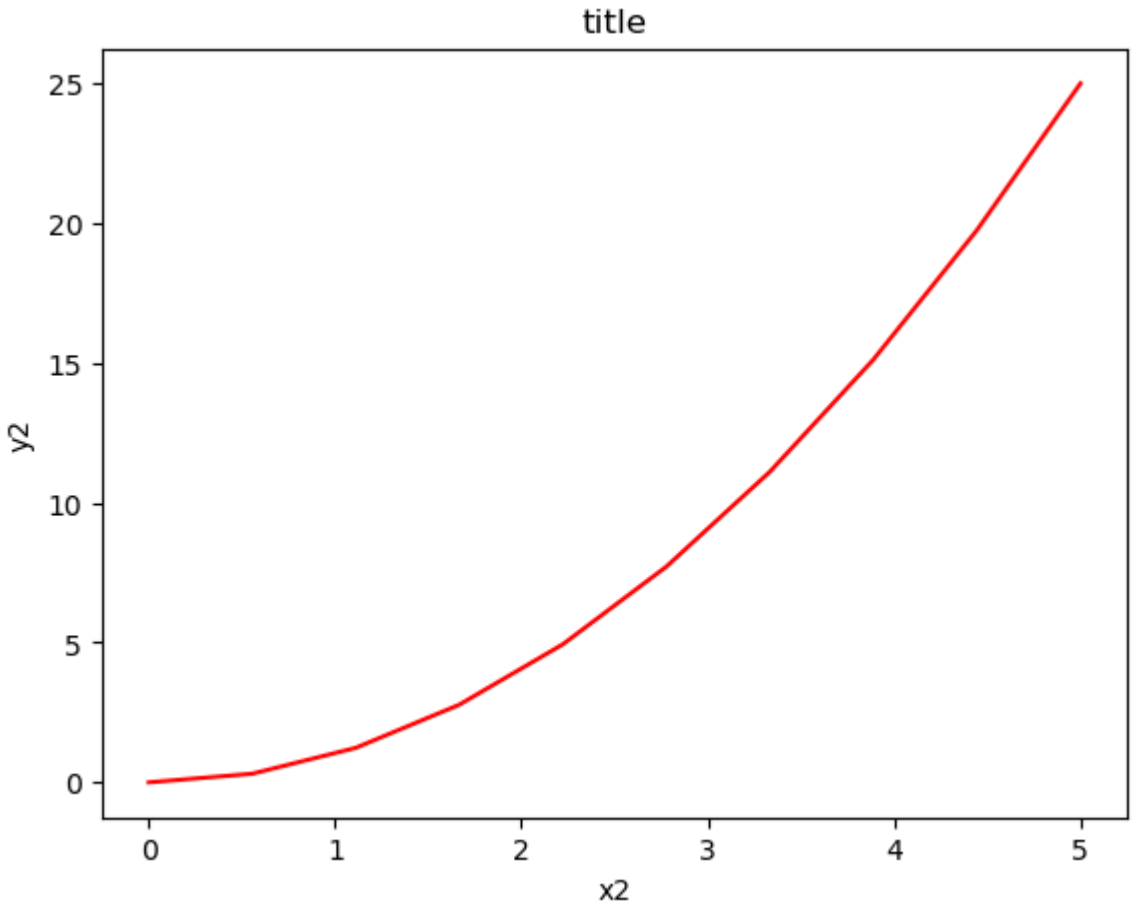


Figure and axes

I start by creating a figure and an axes. A figure and axes can be created as follows:

```
fig = plt.figure()
```

```
ax = plt.axes()
```

In matplotlib , the figure (an intance of the class plt.figure) is a single container that contains all the objects representing axes, graphics , text and labels .The axes (an instance of the class plt.axes) is a bounding box with ticks and labels . it will contain the plot elements that make up the visualization . I have used the variable name fig to refer to a figure instance, and ax to refer to an axes instance or group of axes instances.

In [29]:
```
fig = plt.figure()
ax = plt.axis()
```

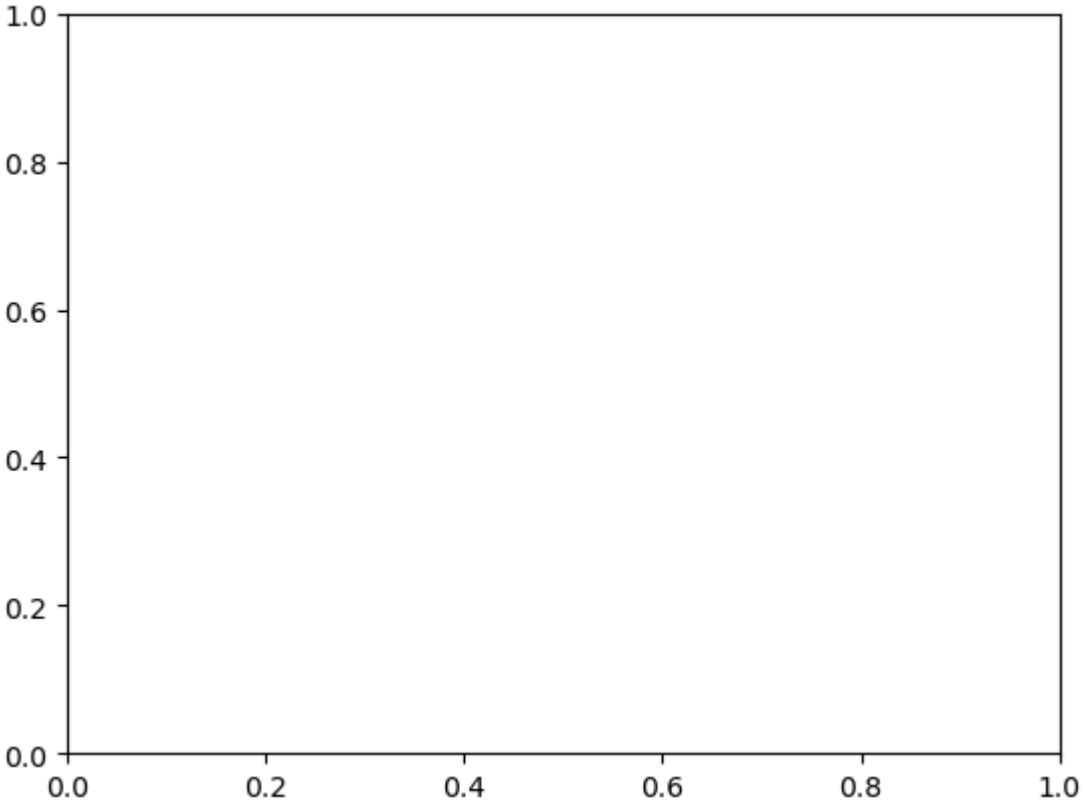


figure and Subplots

Plots in Matplotlib reside within a figure object . As described earlier , we can create a new figure with plt.figure() as follows:-

fig = plt.figure ()

now i create one or more subplots using fig.add_subplots() as follows:-

ax1 = fig.add_subplots(2,2,1)

The above command means that there are four plots in total (2 * 2 = 4). I select the first of four subplots (numbered from 1)

I create the next three subplots using the fig.add_subplots() commnds as follows:-

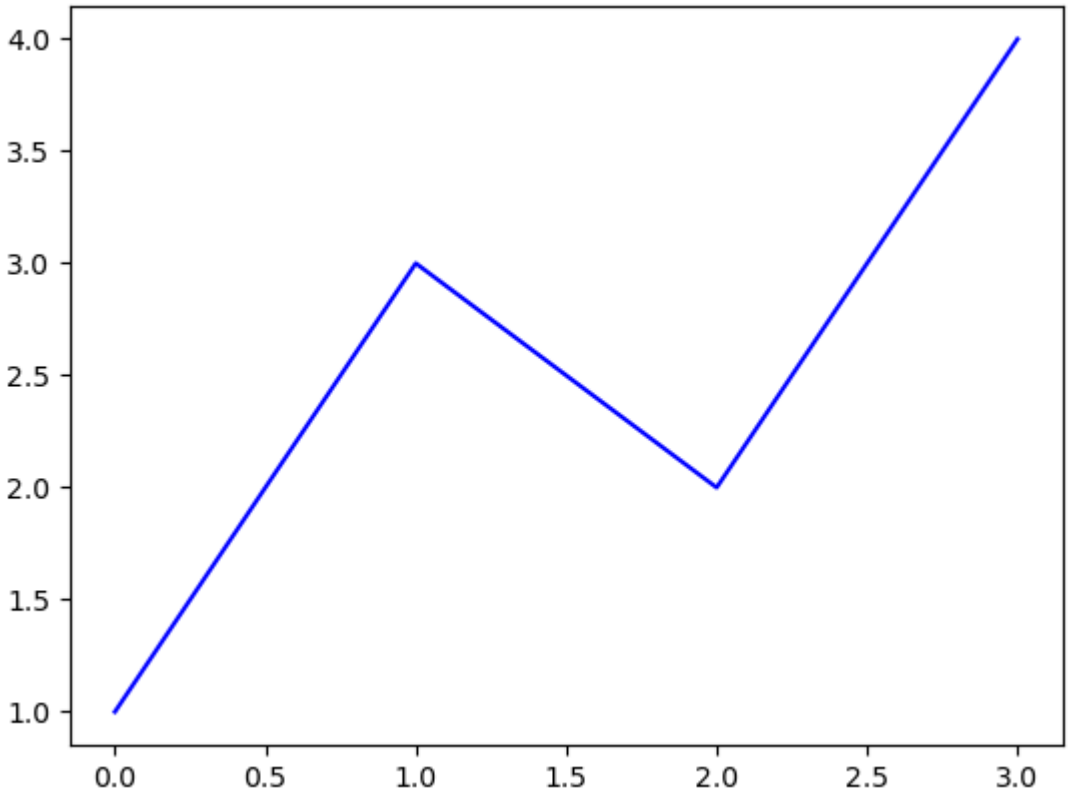
ax2 = fig.add_subplot(2,2,2) ax3 = fig.add_subplot(2,2,3) ax4 = fig.add_subplot(2,2,4)

The above command result in creation of subplots. The diagrammatic representation of subplots are as follow:-

11.First plot with Matplotlib

Now , i will start producing plots. Here is the first example;-

In [32]:
```
plt.plot([1,3,2,4],'b-')
plt.show()
```



`plt.plot([1,3,2,4],'b-')

this code line is the actual plotting command . Only a list of values has been plotted that represent the vertical coordinates of the points to be plotted .Matplotlib will use an implicit horizontal values list, form o (the first value) to N-1 (where n is the number of items in the list).

Specify both Lists

Also we can explicitly specify both the lists as follows :-

x3 = range(6)

```
plt.plot(x3 , [xi**2 for xi in x3])
```

```
plt.show()
```

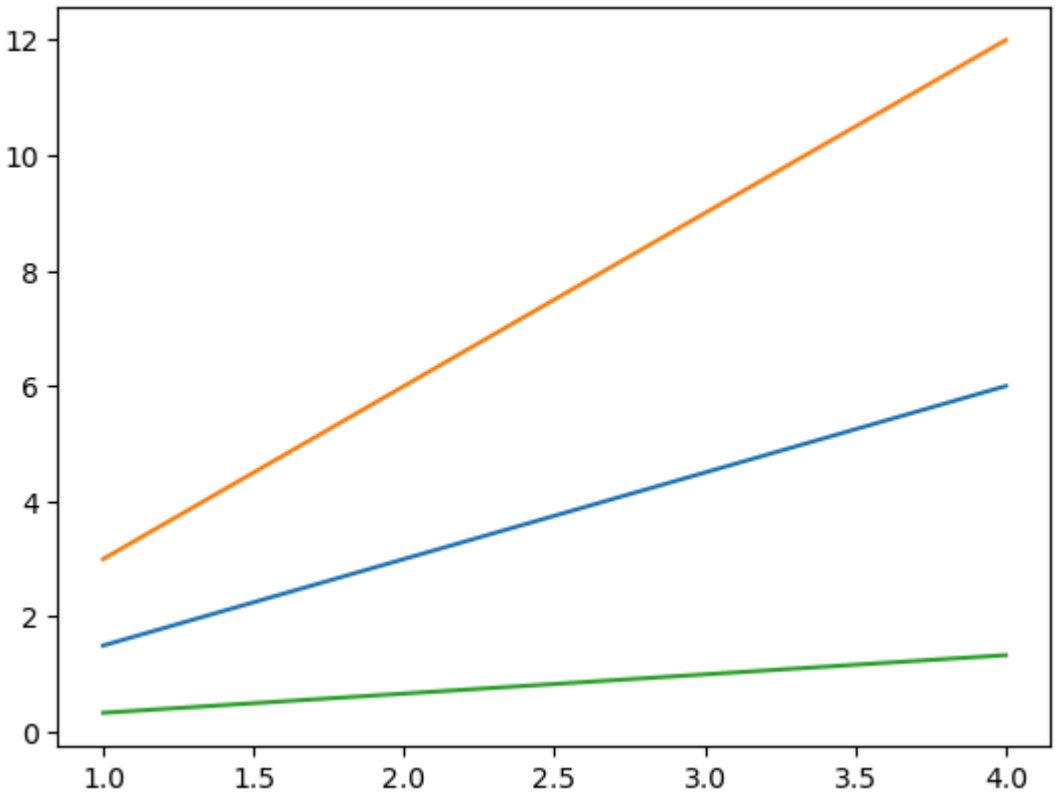
12.Multiline Plots

Multiline Plots mean plotting more than one plot on the same figure. we can plot more than one plot in the same figure.

it can be achieved by plotting all the lines before calling show(). It can be done as follows:-

In [61]:
```
x4 = range(1,5)

plt.plot(x4,[xi*1.5 for xi in x4])
plt.plot(x4,[xi*3 for xi in x4])
```

```
plt.plot(x4, [xi/3.0 for xi in x4])

plt.show()
```



13. Parts of a plot

There are different parts of a plot . These are title , legend , grid , axis and labels etc. These are denoted in the following figure:-

14. Saving the plot

we can save the figures in a wide variety of formats . we can save them using the savefig() command as follows:-

```
fig.savefig('fig1.png)
```

We can explore the contents of the file using the Ipython Image object.

```
from Ipython.display import Image
```

Image('fig1.png')

In savefig() command , the file format is inferred from the extension of the given filename .Depending on the backend, many different file formats are available. The list of supported file types can be found by using the get_supported_filetypes() method of the figure canvas object as follows:-

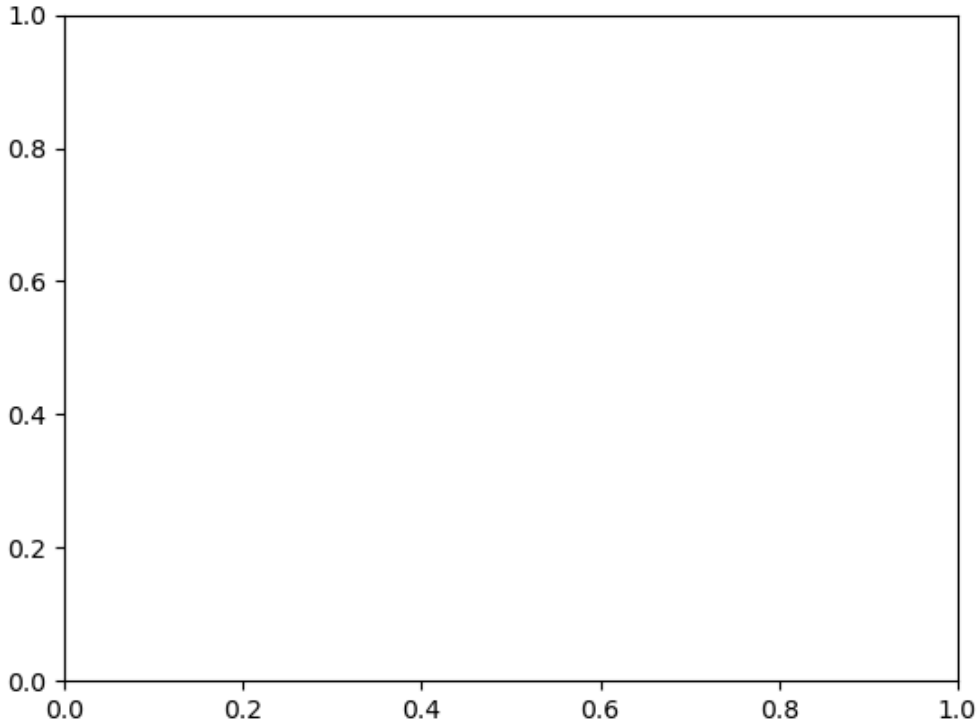
```
fig.canvas.get_supported_filetypes()
```

In [67]:
```
# saving the figure

fig.savefig('plot1.png')
```

In [75]:
```
from IPython.display import Image
```

In [77]:
```
Image('plot1.png')
```

Out[77]:



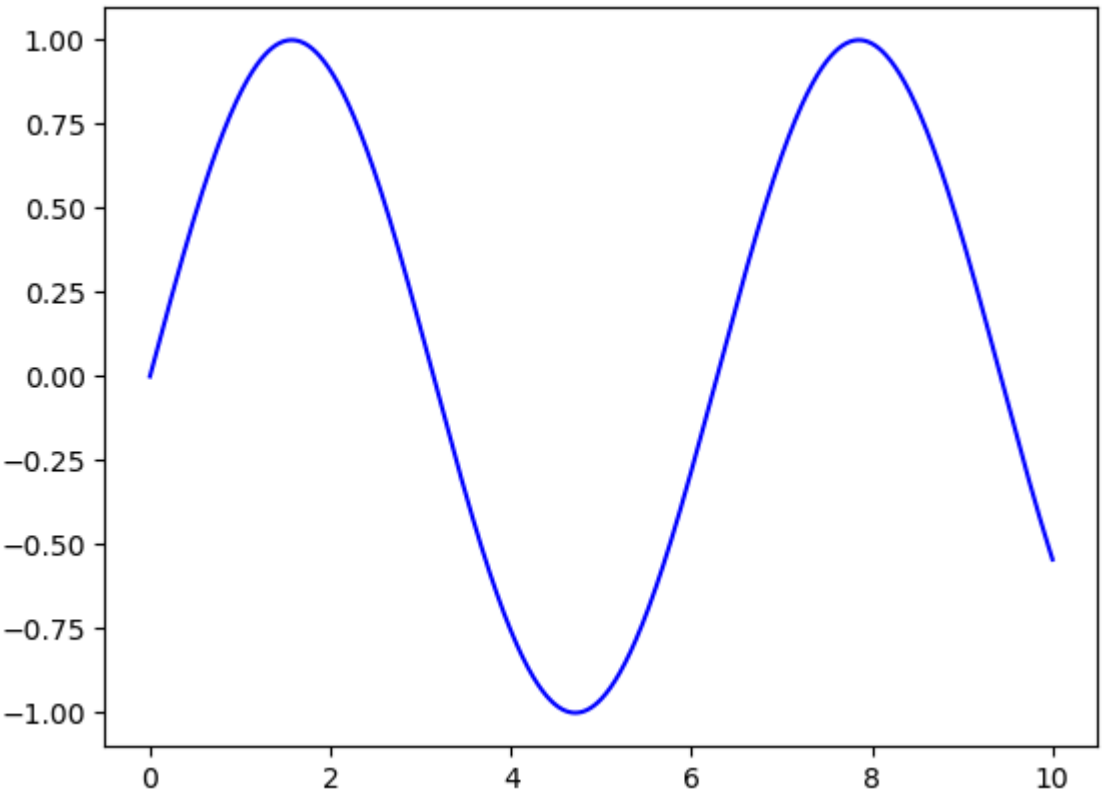
In [79]:
```
# Explore supported file formats

fig.canvas.get_supported_filetypes()
```

Out[79]:
```
{'eps': 'Encapsulated Postscript',
 'jpg': 'Joint Photographic Experts Group',
 'jpeg': 'Joint Photographic Experts Group',
 'pdf': 'Portable Document Format',
 'pgf': 'PGF code for LaTeX',
 'png': 'Portable Network Graphics',
 'ps': 'Postscript',
 'raw': 'Raw RGBA bitmap',
 'rgba': 'Raw RGBA bitmap',
 'svg': 'Scalable Vector Graphics',
 'svgz': 'Scalable Vector Graphics',
 'tif': 'Tagged Image File Format',
 'tiff': 'Tagged Image File Format',
 'webp': 'WebP Image Format'}
```

15. Line Plot

we can use the following commands to draw the simple sinusoid line plot:-

In [82]:
```
fig = plt.figure()

ax = plt.axes()

# declare a variable x5

x5 = np.linspace(0,10,1000)

# plot the sinusoid function

ax.plot(x5,np.sin(x5),'b-');
```



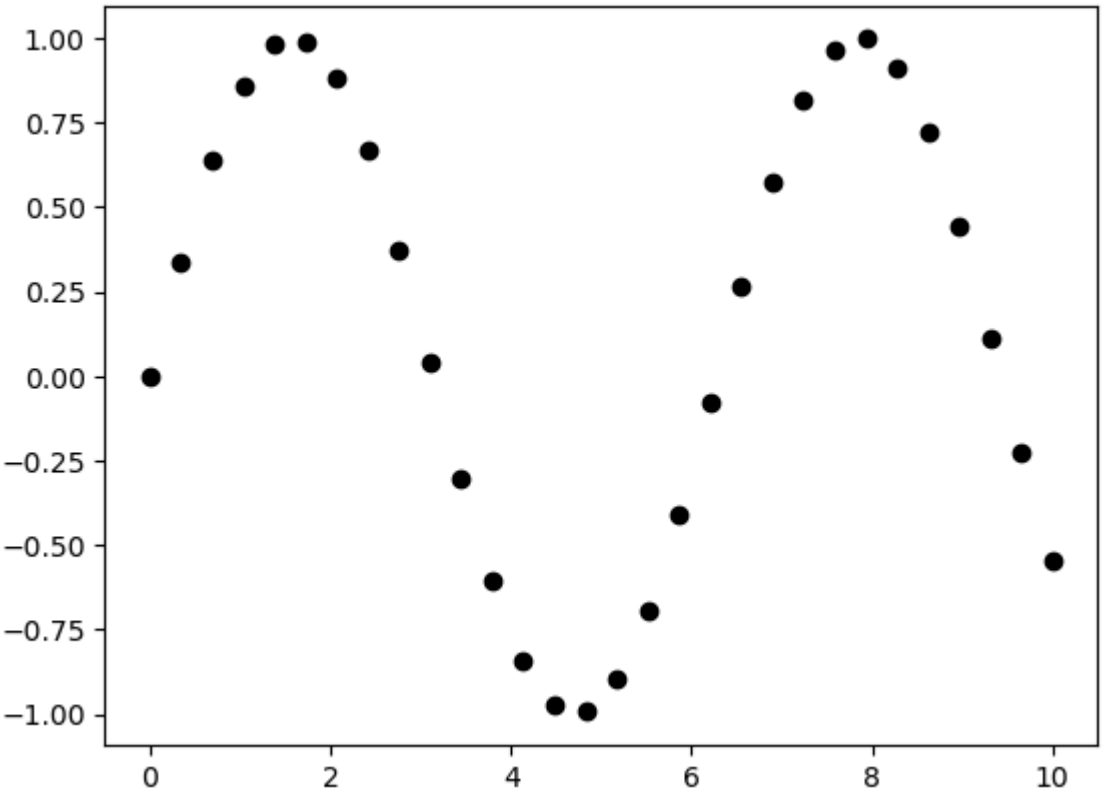
16. Scatter Plot

Another commonly used plot type is the scatter plpot. Here the points are represented individually with a dot or a circle.

Scatter plot with plt.plot()

we have used plt.plot/ax.plot to produce line plots . we can use the same functions to produce the scatter plots as follows:-

In [93]:
```
x7 = np.linspace(0,10,30)

y7 = np.sin(x7)

plt.plot(x7,y7,'o', color = 'black');
```

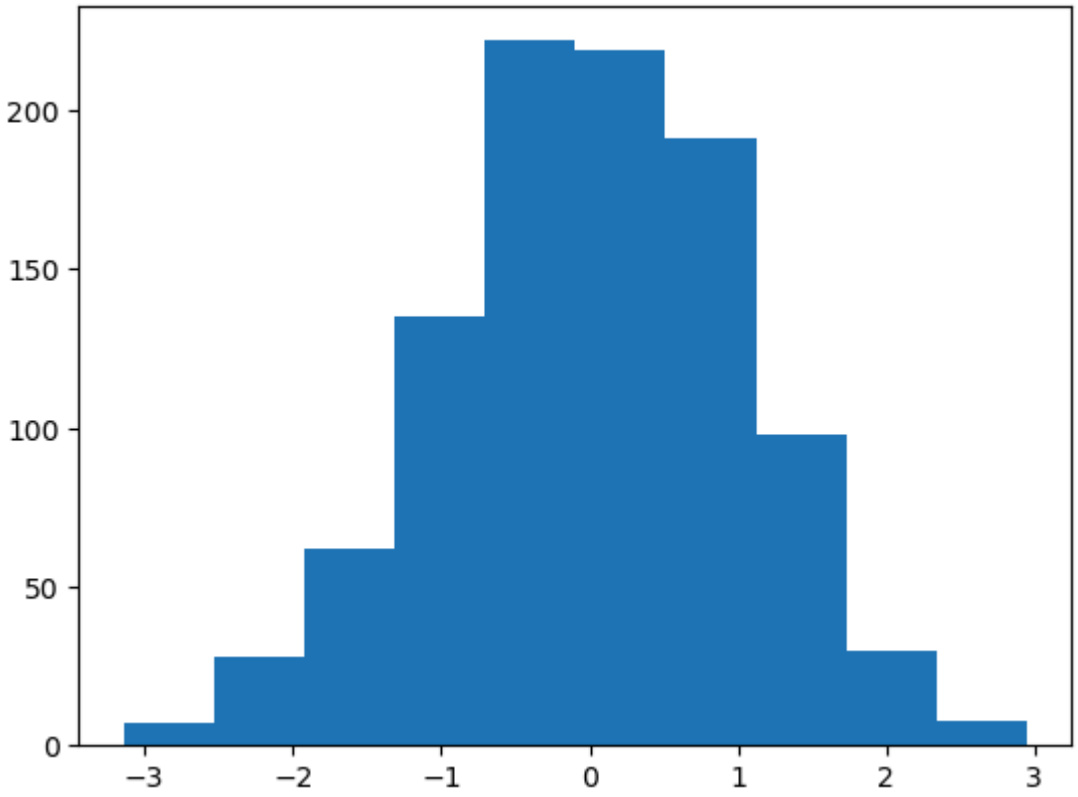


17. Histogram

Histogram charts are a graphical display of frequencies. They are represented as bars. They show what portion of the dataset falls into each category , usually specified as non-overlappng intervals . These categories are called bins .

The plt.hist() function can be used to plot a simple histogram as follows:-

In [96]:
```
data1 = np.random.randn(1000)
plt.hist(data1);
```

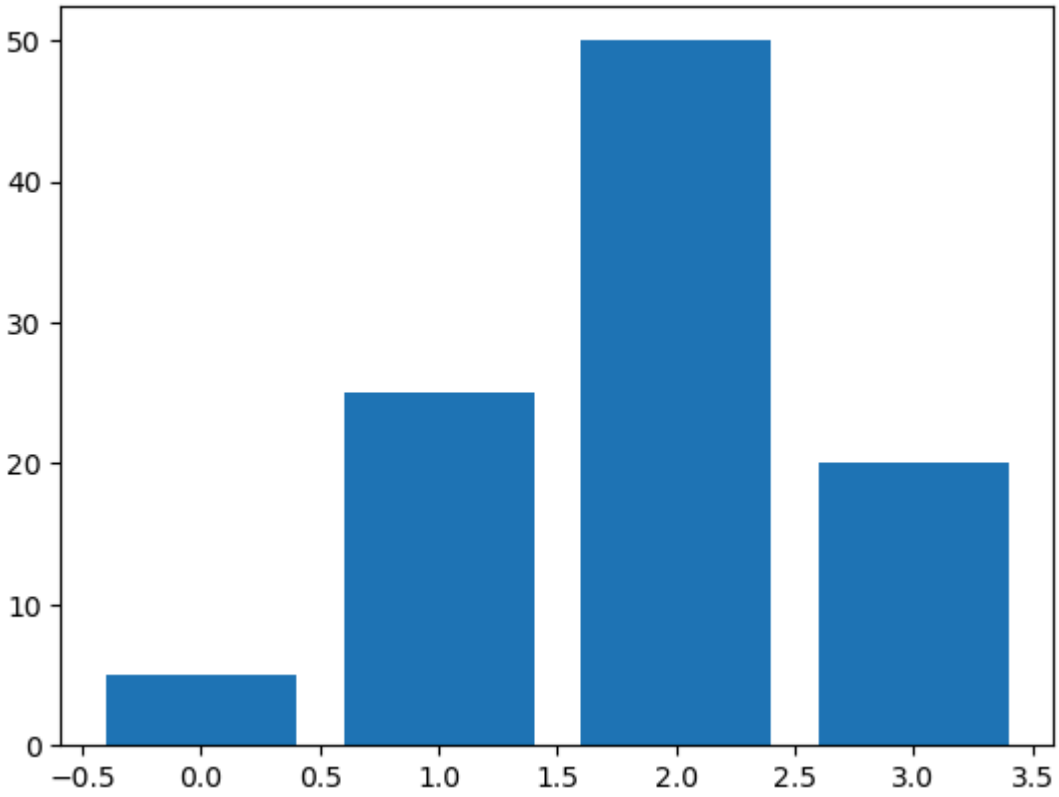


18. Bar Chart

Bar charts display rectangular bars either in vertical or horizontal form. Their length is proportional to the values they represent .They are used to compare two or more values.

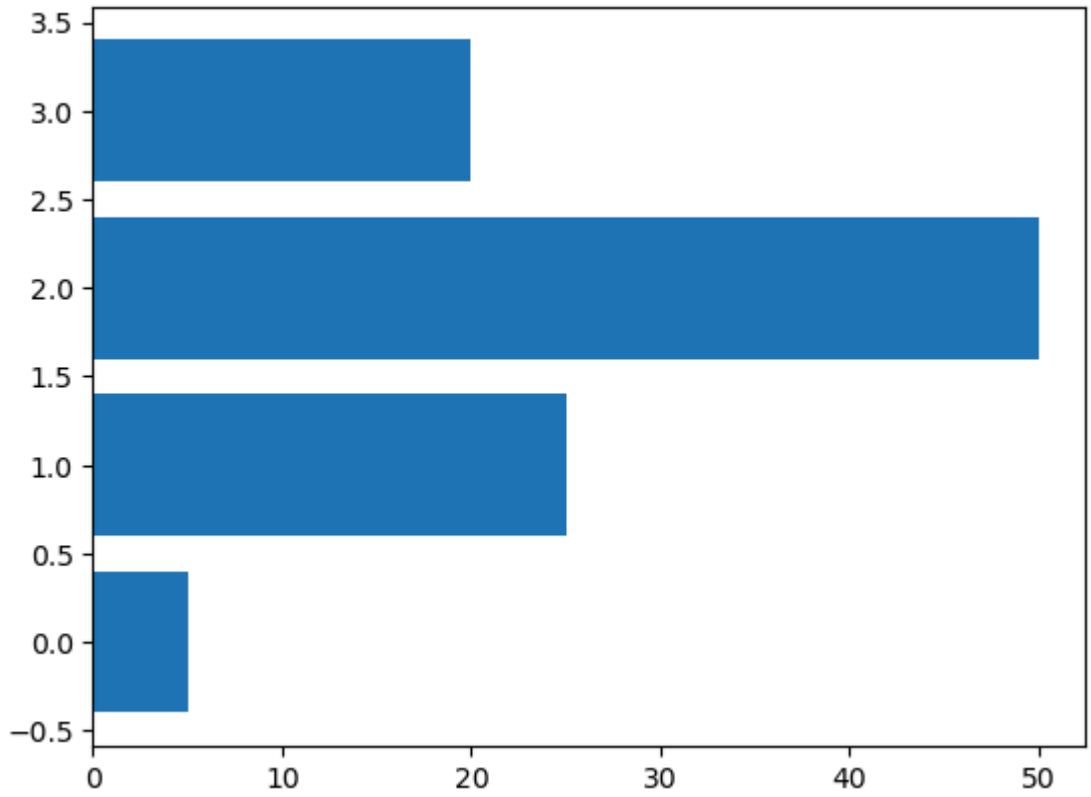
we can plot a bar using plt.bar() function. we can plot a bar chart as follows:-

In [99]:
```
data2 = (5.,25.,50.,20.)

plt.bar(range(len(data2)),data2)

plt.show()
```



In [101…
```
data2 = [5,25,50,20]
plt.barh(range(len(data2)),data2)
```

Out[101… `<BarContainer object of 4 artists>`

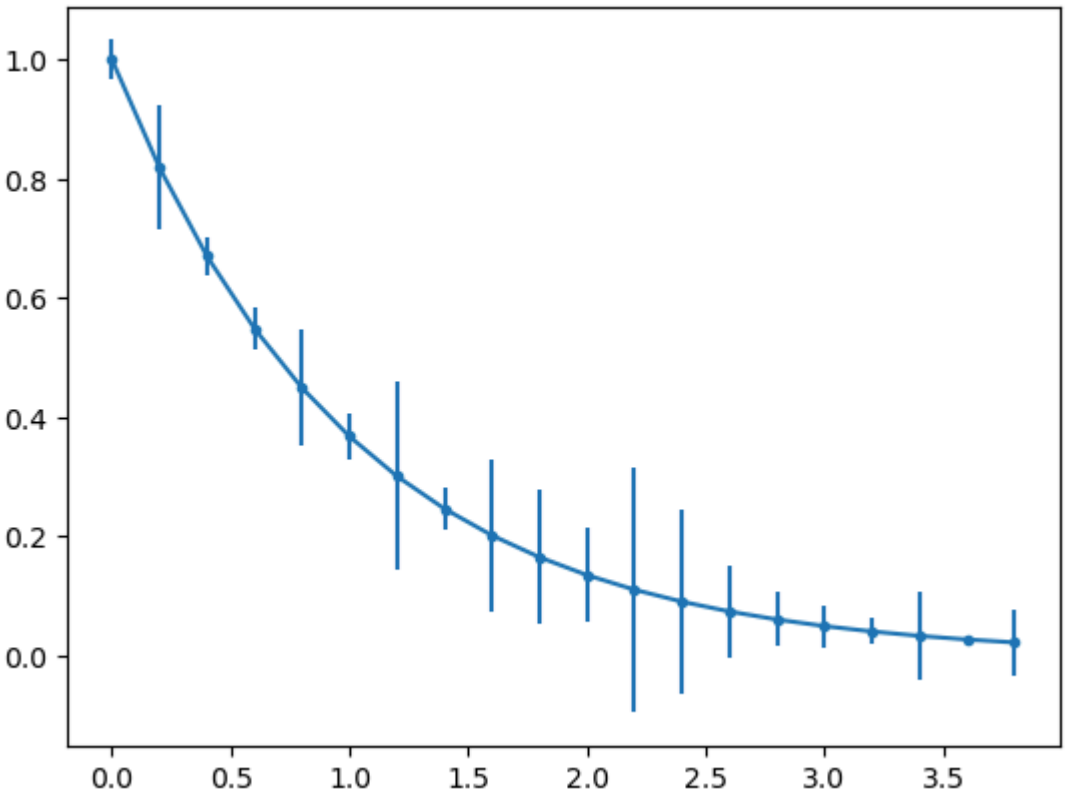


Error Bar Chart

IN experimental design , the measurements lack perfect precision. So we have to repeat the measurements . It results in obtaining a set of values .The representatuon of the distribution of data values us done y plotting a single data point (knows as mean value of dataset) and an error bar to represent the overall distribution of the data

we can use matplotlib's errorbar() function to represetn the distribution of data values.It can done as follows:-

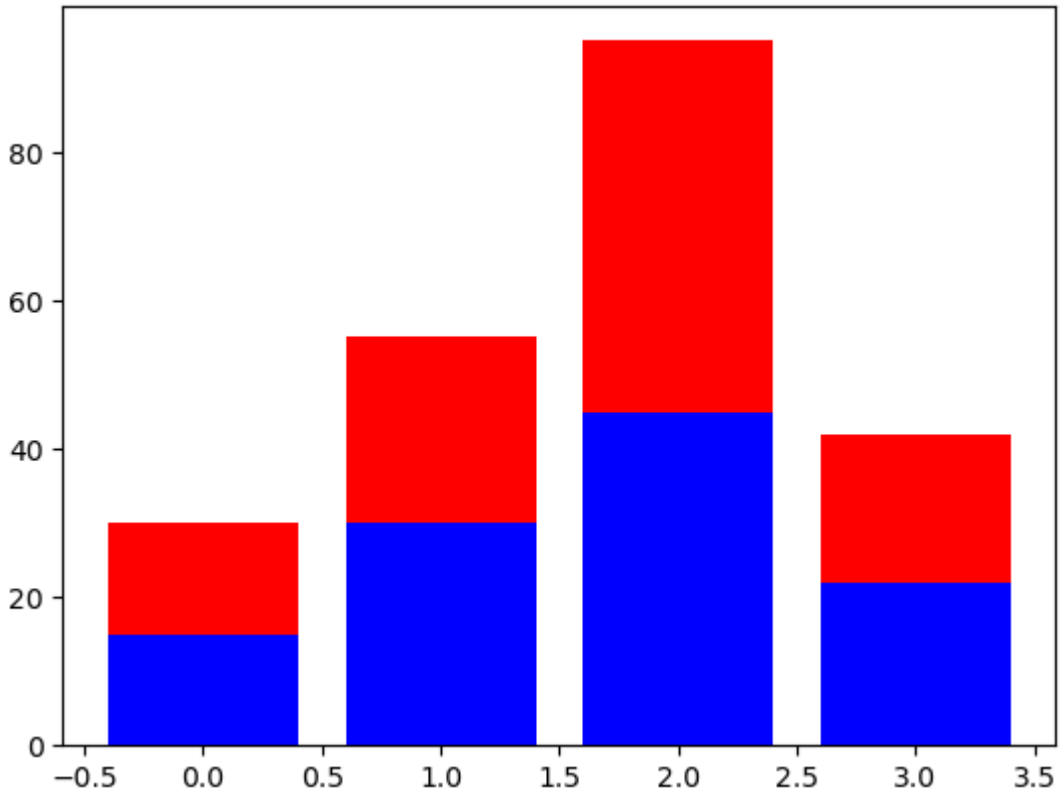
In [104…
```
x9 = np.arange(0,4,0.2)

y9 = np.exp(-x9)

e1 = 0.1 * np.abs(np.random.randn(len(y9)))

plt.errorbar(x9, y9, yerr = e1 , fmt = '.-')
plt.show()
```



21 .Stacked Bar Chart

we can draw stacked bar chart by using a special parameter called **bottom** from the plt.bar() function . It can be done as follows:-

In [111…
```
a = [15,30,45,22]
b = [15,25,50,20]

z2 = range(4)

plt.bar(z2 , a, color = 'b')
plt.bar(z2 , b, color = 'r', bottom = a)
plt.show()
```



The optional bottom parameter of the plt.bar() function allows us to specify a starting position for a bar . Instead of running from zero to a value , it will go from the bottom to value. The first call to plt.bar() plots the blue bars . The second call to plt.bar() plots the red bars, with the bottom of the red bars being at the top of the blue bars.

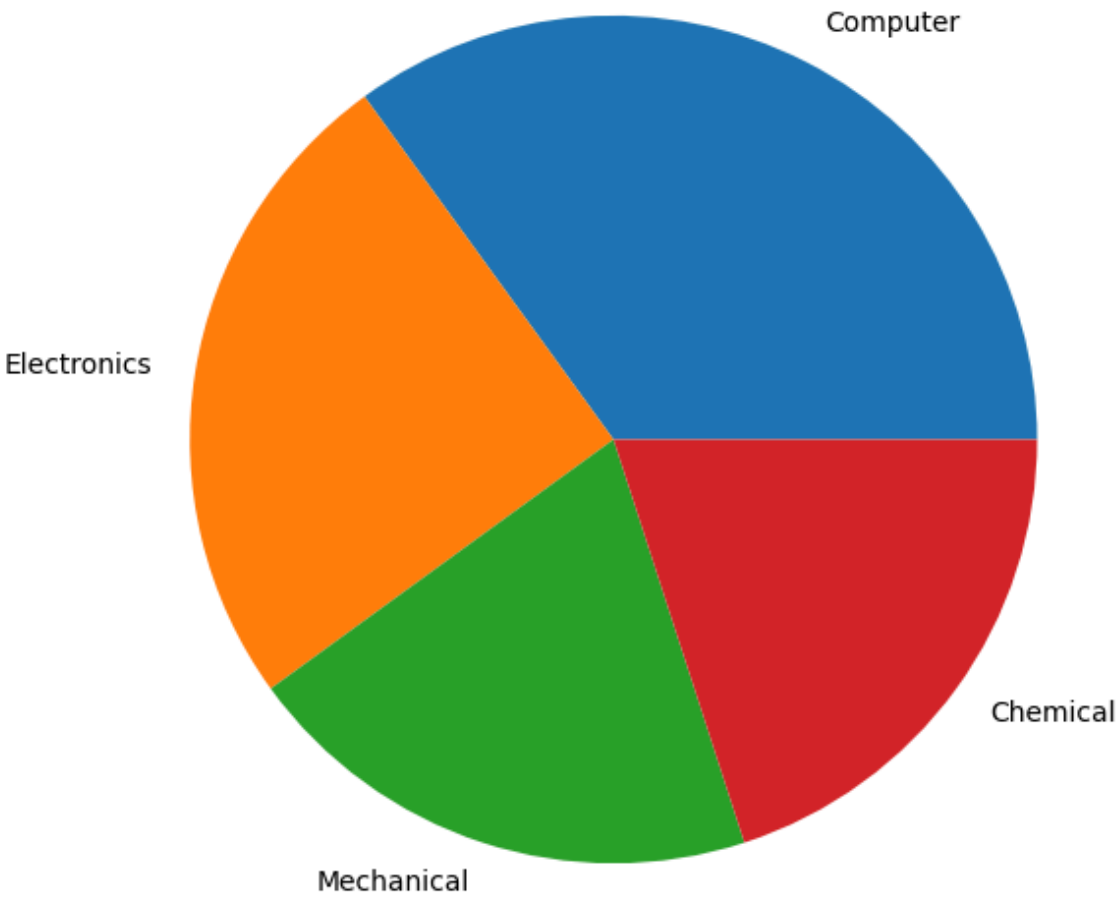
22. Pie Chart

Pie charts are circular representations, divided into sectors .The sectors are also called wedges . The arc length of each sector is proportional to the quantity we are describing . It is an effective way to represent information when we are interested mainly in comparing the wedge against the whole pie , instead of wedges against each other.

Matplolib provides the pie() function to plot pie charts from an array X . wedges are created proportionally , so that each value x of array X generates a wedge Proportional to x\sum(X).

In [119…

```
plt.figure(figsize = (7,7))
x10 = [35,25,20,20]
labels = ['Computer', 'Electronics','Mechanical','Chemical']
plt.pie(x10,labels =labels);

plt.show()
```



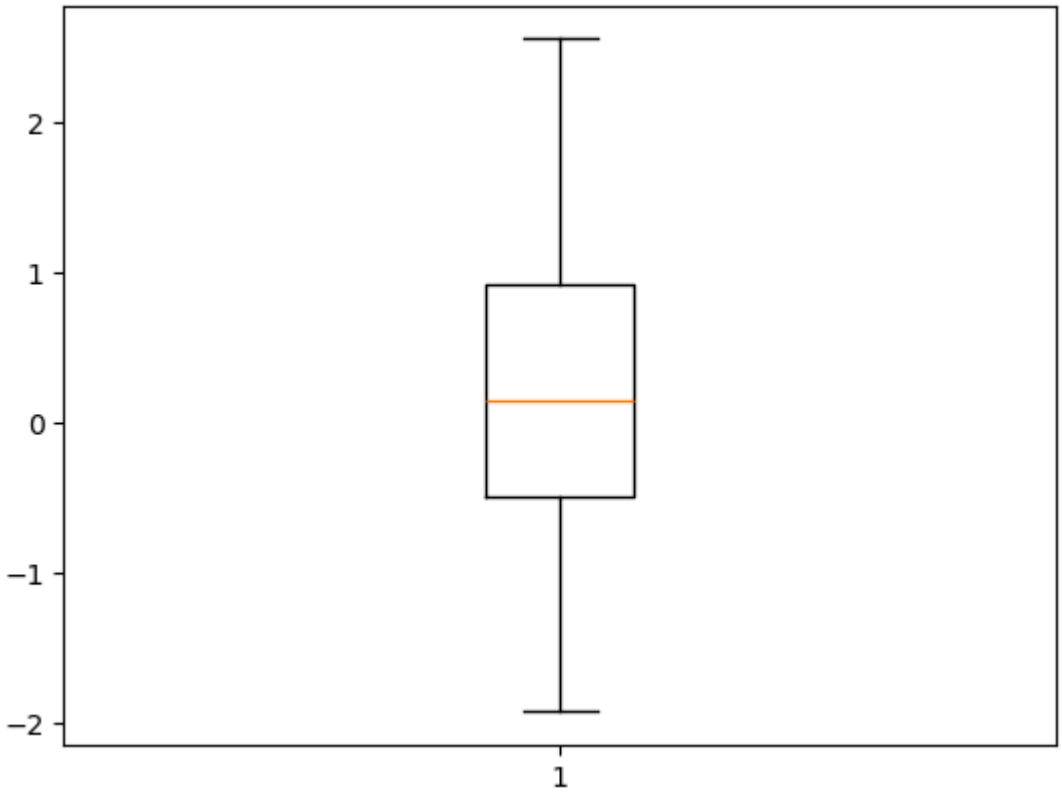
23.Boxplot

Boxplot allows us to compare distributions of values by showing the median , quartiles, maximum and minimum of a set of values.

we can plot a boxplot with the boxplot() function as follows:-

In [122…

```
data3 = np.random.randn(100)

plt.boxplot(data3)

plt.show();
```



The boxplot() function takes a set of values and computes the mean , median and other statistical quantities . The following points describes the preceeding boxplot;

- the red bar is the median of the distribution.
- The Blue box includes 50 percent of the data from the lower quartile to the upper quartile thus the box is centered on the median of the data.
- The lower whisker extends to the lowest value within 1.5 IQR from the lower quartile.
- The upper whisker extends to the highest value within 1.5 IQR from the upper Quartile .
- values further from the whiskers are shown with a cross marker.

24 . Area Chart

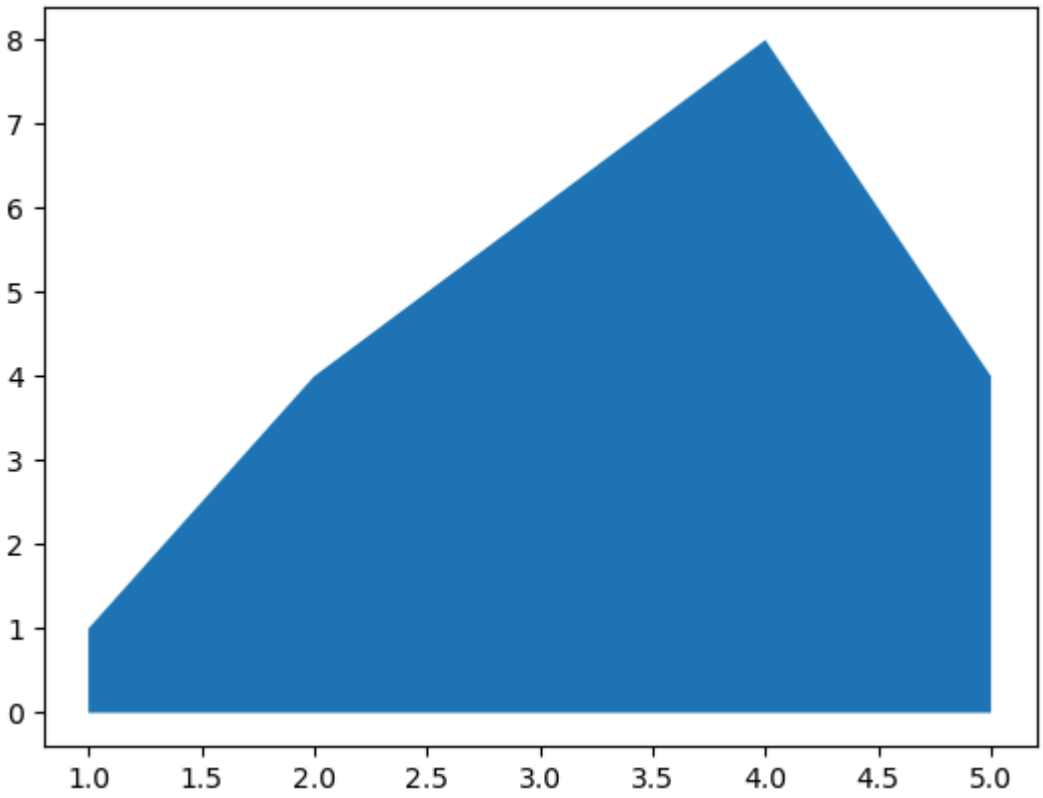
An Area chart is very similar to a line chart . The area between the x-axis and the line is filled in with color or shading . It represents the evolution of numerical variable following another numerical variable.

we can create an area chart as follows;-

In [125…

```
# Create some data

x12 = range(1,6)
y12 = [1,4,6,8,4]

# Area plot
```

```
plt.fill_between(x12 , y12)
plt.show()
```



I have created a basic Area chart . I could also use the stackplot function to crate the area chart as follows;-

plt.stackplot(x12,y12)

The fill_between() function is more convenient for future customization.

25.Contour Plot

Contour Plots are useful to display three dimensional data in two dimensions using contours or color-coded regions .Contour lines are also known as level lines or isolines contour lines for a function of two variables are curves where the function has constant values. They have specific names beginning with iso-according to the nature of the variables being mapped.

There are lot of applications of contour lines in several fields such as meteorology (for temperature , pressure ,rain , wind speed) , geography , magnetism, engineeringm social sciences and so on.

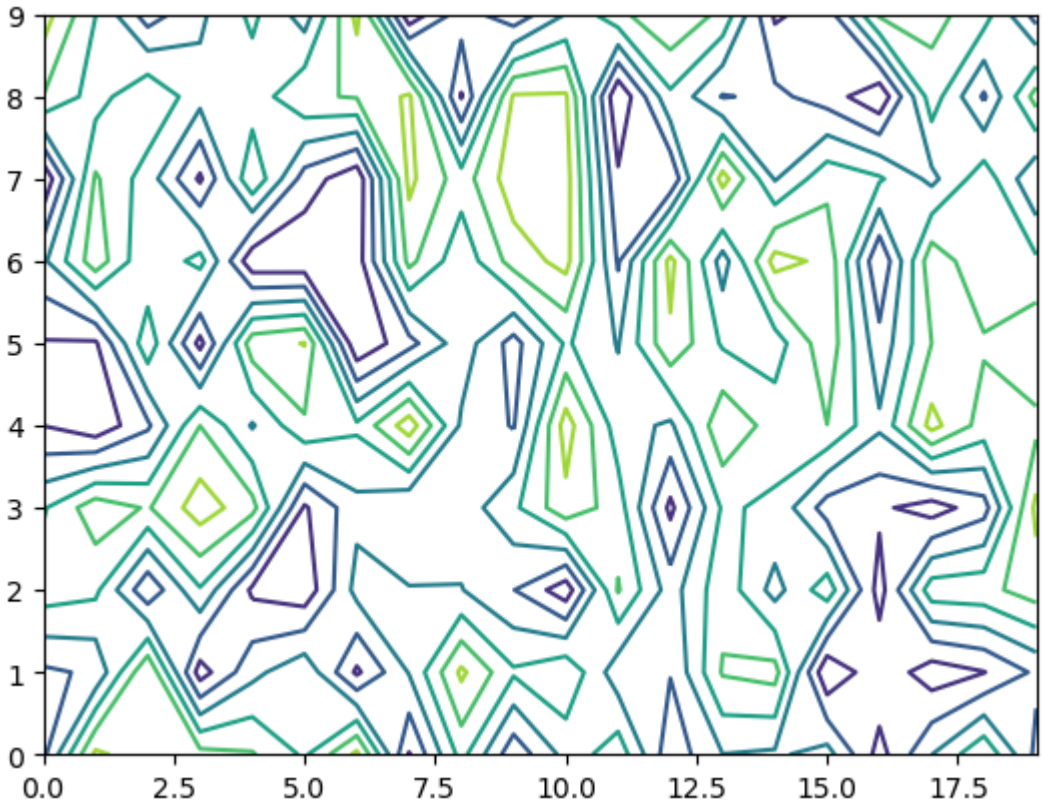
The density of the lines indicates the slope of the function . The gradient of the function is always perpendicular to the contour line . when the lines are close together ,the length of the gradient is large and the variation is steep.

A Contour plot can be created with the plt.contour () function as follows;-

In [128…

```
# create a matrix
```

```
matrix1 = np.random.rand(10,20)

cp = plt.contour(matrix1)
plt.show()
```



The **Contour()** function draws contour lines. It takes a 2d array as input . Here, it is a matrix of 10 x 20 random elements.

The number of level lines to draw is chosen automatically , but we can also specify it as an additional parameter,N.

```
plt.contour(matrix, N)
```

26. Styles with Matplotlib Plots

The Matplotlib version 1.4 which released in august 2014 added a very convenient style module . it includes a number of new default stylesheets, as well as the ability to - create and package own styles

we can view the list of all available styles by the following command.

```
print(plt.styles.available)
```

In [135…

```
# view list of all available styles

print(plt.style.available)
```

```
['Solarize_Light2', '_classic_test_patch', '_mpl-gallery', '_mpl-gallery-nogrid',
'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'graysc
ale', 'seaborn-v0_8', 'seaborn-v0_8-bright', 'seaborn-v0_8-colorblind', 'seaborn-
v0_8-dark', 'seaborn-v0_8-dark-palette', 'seaborn-v0_8-darkgrid', 'seaborn-v0_8-d
eep', 'seaborn-v0_8-muted', 'seaborn-v0_8-notebook', 'seaborn-v0_8-paper', 'seabo
rn-v0_8-pastel', 'seaborn-v0_8-poster', 'seaborn-v0_8-talk', 'seaborn-v0_8-tick
s', 'seaborn-v0_8-white', 'seaborn-v0_8-whitegrid', 'tableau-colorblind10']
```

we can set the styles for matplotlib plots as follows:-

plt.style.use('seaborn-bright')

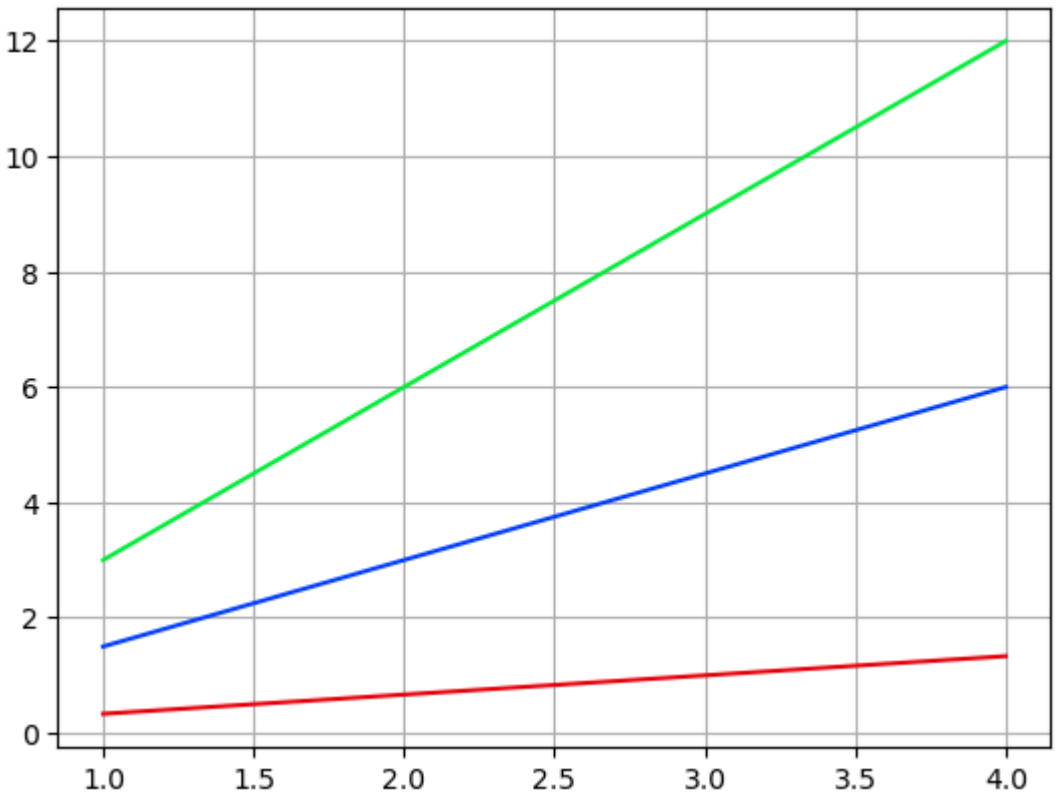
In [142…
```
plt.style.use('seaborn-v0_8-bright')
```

I have set the **seaborn-bright** style for plots. So the plot uses the seaborn bright matplotlib style for plots

27. Adding a grid

In some cases , the background of a plot was completely blank. We can get more information , if there is a reference system in the plot . The reference system would improve the comprehension of the plot . An example of the reference system is adding a grid . we can add a grid to the plot.by calling the grid() function it takes one parameter a boolean value , to enable(if True) or disable (if false) the grid.

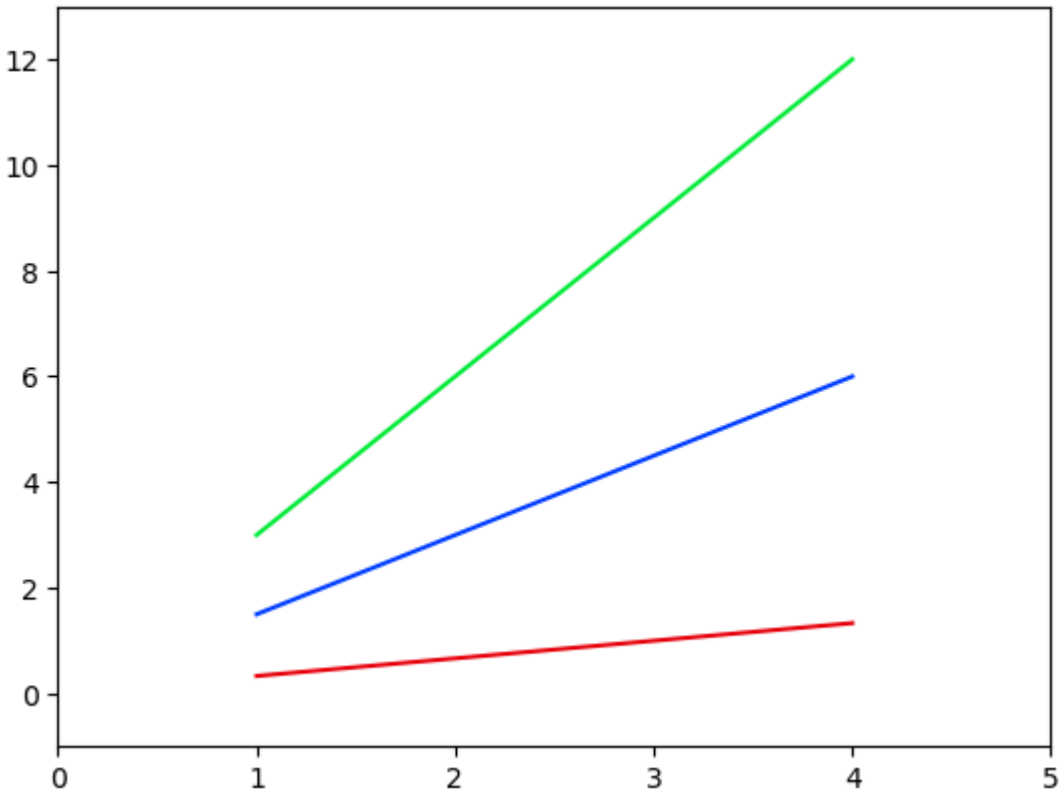
In [145…
```
x15 = np.arange(1,5)
plt.plot(x15,x15 * 1.5, x15, x15 *3.0 ,x15,x15/3.0)

plt.grid(True)

plt.show()
```



28. Handling axes

Matplotlib automatically sets the limits of the plot to precisely contain the plotted datasets. Sometimes , we want to set the axes limits ourself. We can set the axes limits with the axis() function as follows :-

In [148…
```
x15 = np.arange(1,5)

plt.plot(x15 , x15*1.5,x15, x15 *3.0 , x15 , x15/3.0)

plt.axis()

plt.axis([0,5,-1,13])
plt.show()
```



we can see that we now have more space around the lines

if we execute axis() without patameters it returns the actual axis limits .

we can set parameters to axis() by a list of four values.

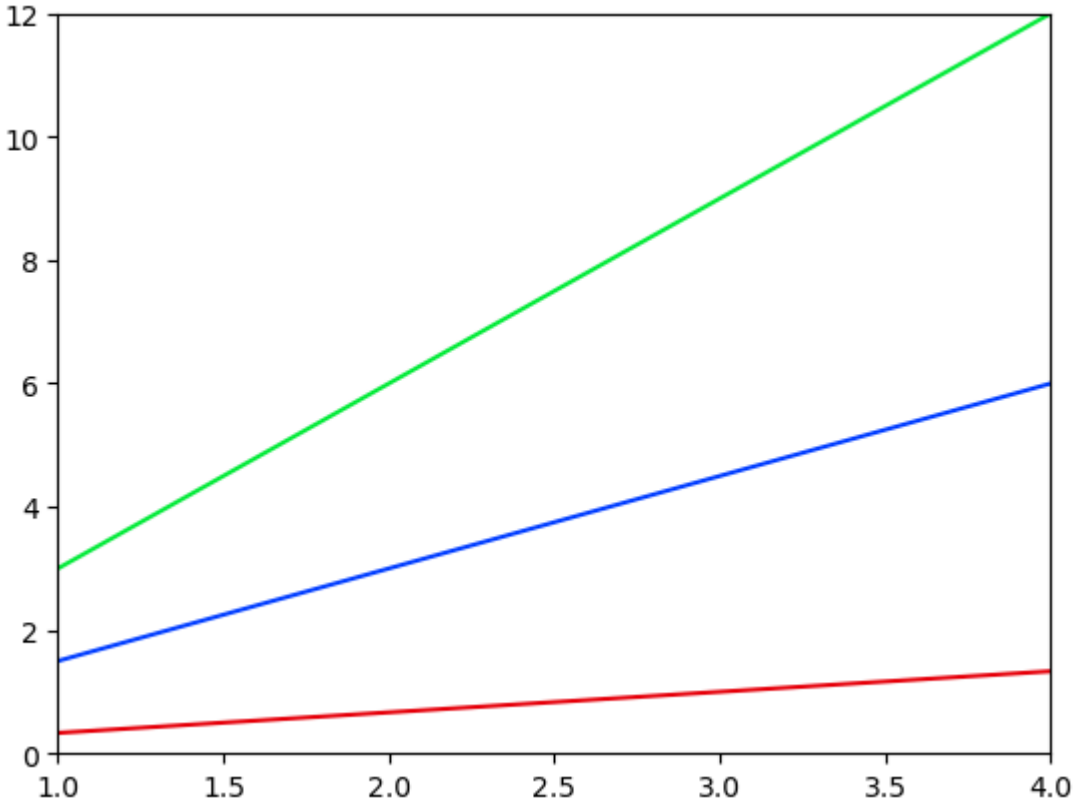
The list of four values are the keyword arguments [xmin,xmax,ymin,ymax] allows the minimum and maximum limits for X and Y axis respectively.

we can control the limits for each axis separately using the xlim() and ylim() functions this can be done as follows:-

In [151…
```
x15 = np.arange(1,5)
plt.plot(x15,x15*1.5,x15,x15*3.0,x15,x15/3.0)

plt.xlim([1.0,4.0])
```

```
plt.ylim([0.0,12.0])
```

Out[151… (0.0, 12.0)



29. Handling X and Y ticks

vertical and horizontal ticks are those little segments on the axes , coupled with axes labels , used to give a reference system on the graph so , they form the origin and the grid lines.

Matplotlib provides two basic functions to manage them - xticks() and yticks() .

Executing with no arguments the tick function returns the current ticks location and the labels corresponding to each of them.

we can pass arguments(in the form of lists) to the ticks functions. The arguments are :-

1.Location of the ticks

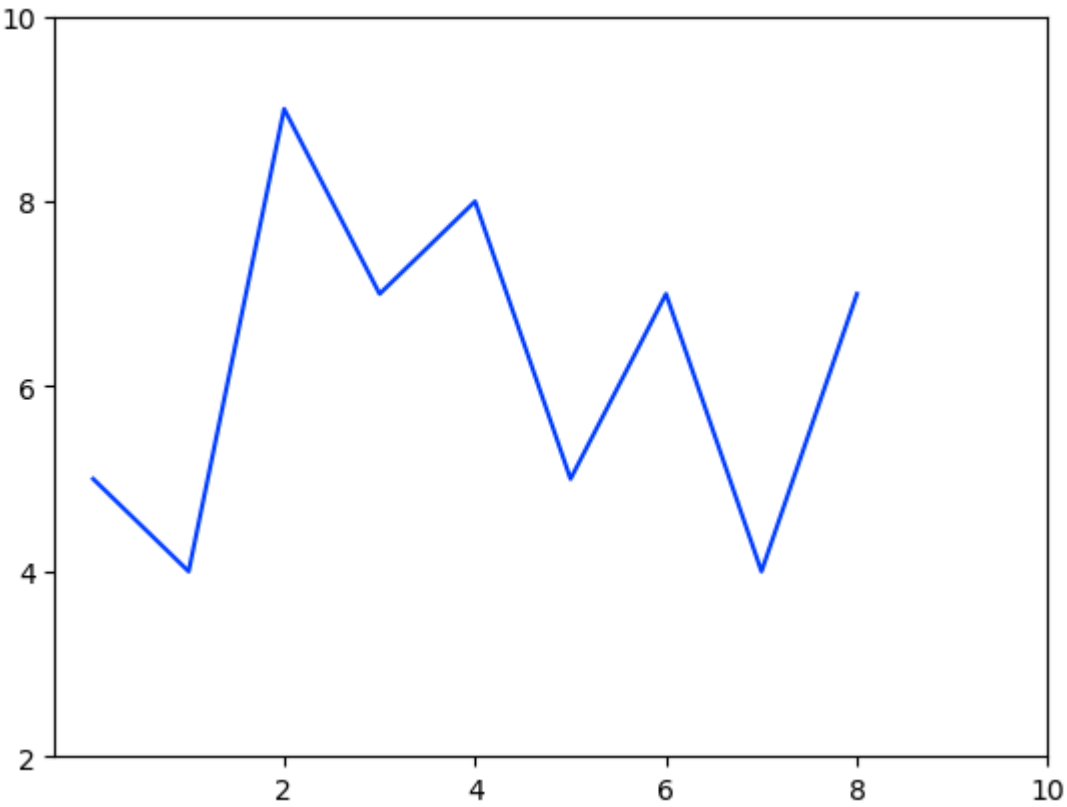
2. Labels to draw at these locations.

we can demonstrate the usage of the ticks functions in the code snippet below:-

In [154…

```
u = [5,4,9,7,8,5,7,4,7]

plt.plot(u)

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

plt.yticks([2,4,6,8,10])
```

```
plt.show()
```

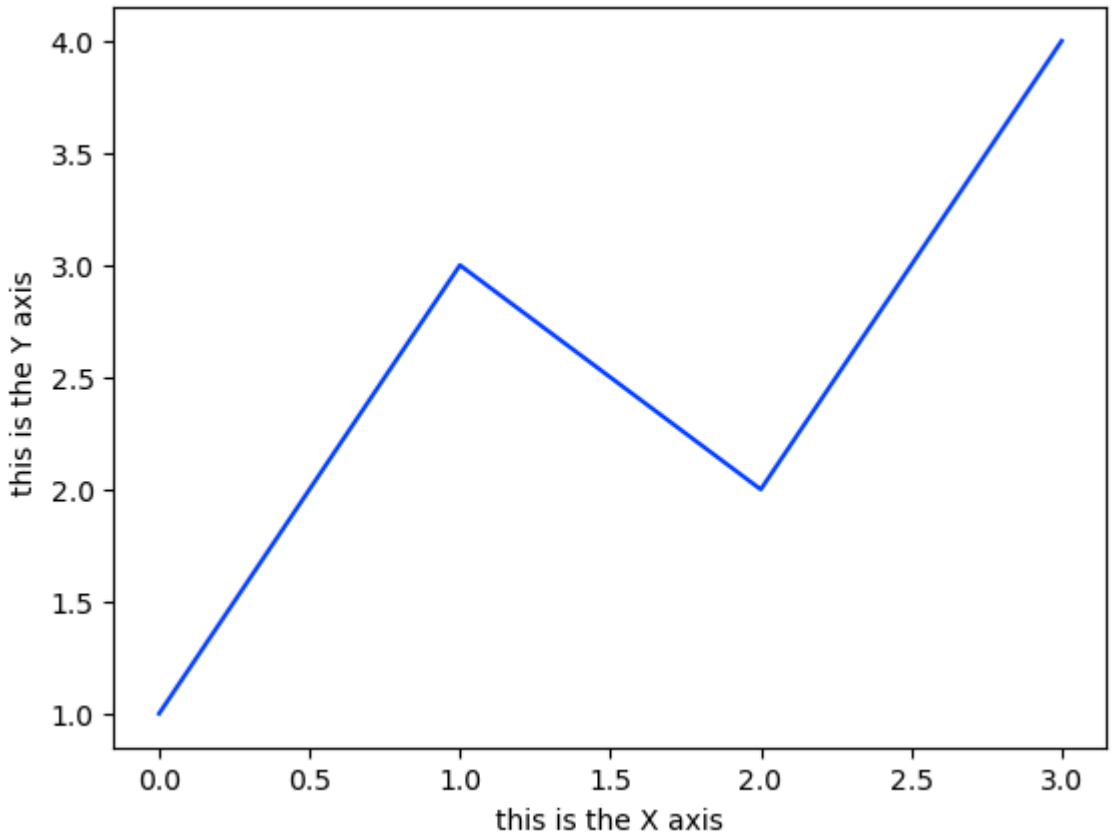


30.Adding labels

Another important piece of information to add to a plot is the axes labels, since they specify the type of data we are plotting.

In [157…

```
plt.plot([1,3,2,4])

plt.xlabel("this is the X axis")

plt.ylabel("this is the Y axis ")

plt.show()
```

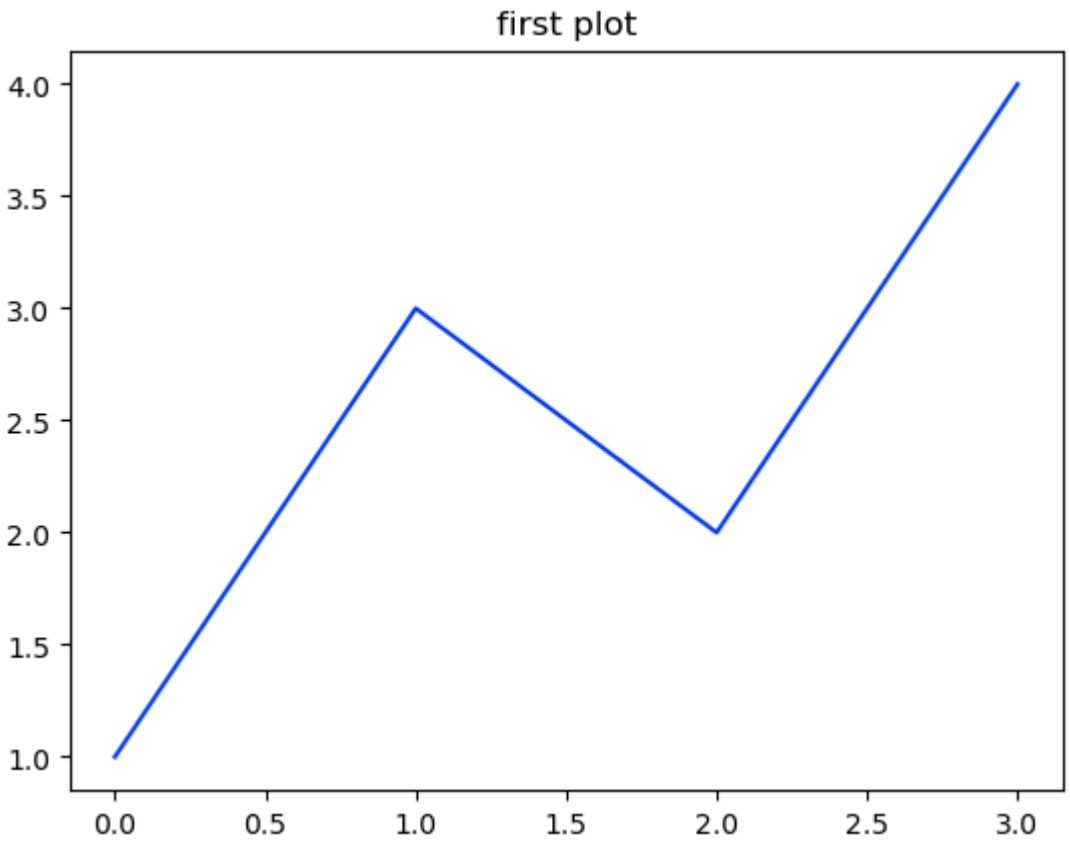


31. Adding a title

The title of a plot describes about the plot . Matplotlib provides a simple function title() to add a title to an image.

In [160…

```
plt.plot([1,3,2,4])
plt.title('first plot')

plt.show()
```



The above plots displays the output of the previous code . the title first plot is displayed on top of the plot.

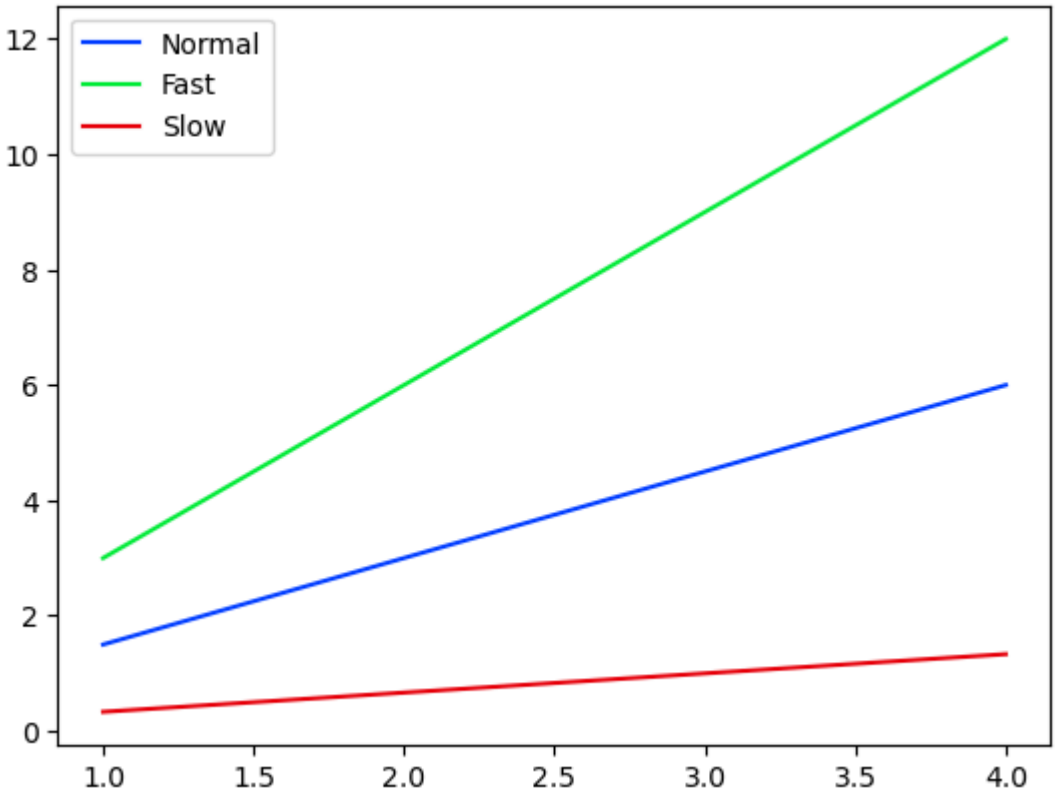
32. Adding a legend

legends are used to describe what each line or curve means in the plot.

legends for curves in a figure can be added in two ways. one method is to use the legend method of the axis object and pass a list\tuple of texts as follows:-

In [166…

```
x15 = np.arange(1,5)

fig ,ax = plt.subplots()

ax.plot(x15 , x15*1.5)
ax.plot(x15, x15*3.0)
ax.plot(x15, x15/3.0)

ax.legend(['Normal' , 'Fast' , 'Slow']);
```



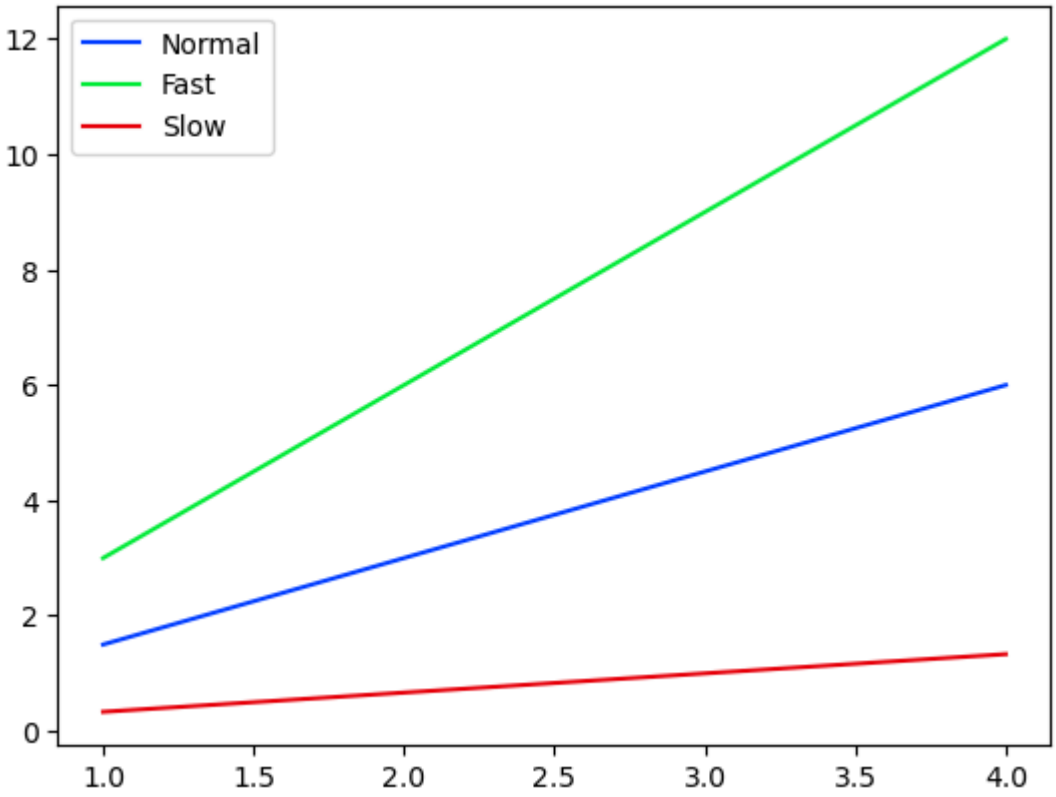
The above method follows the matlab Api . it is prone to errors and unflexible if curves are added to or removed from the plot . it resulted in a wrongly labelled curve.

A better method is to use the label keyword argument when plots are added to the figure . Then we use the legend method without arguments to add the legend to the figure.

The advantage of this method is that if curves are added or removed from the figure , the legend is automatically updated accordingly . It can be achieved by executing the code below.

In [169…

```
x15 = np.arange(1,5)
fig,ax = plt.subplots()

ax.plot(x15, x15*1.5,label = "Normal")

ax.plot(x15 , x15*3.0 ,label ='Fast')

ax.plot(x15,x15/3.0 , label = "Slow")

ax.legend();
```



The legend function takes an optional keyword argument loc . It specifies the location of the legend to be drawn . The loc takes numerical codes for the various pplaces the legend can be drawn . The most common loc values are as follows:-

ax.legend(loc = 0) # let matplotlib decide the optimal location

ax.legend(loc =1) # upper right corner

ax.legend(loc = 2) # upper left corner

ax.legend(loc = 3) # lower left corner

ax.legend(loc = 4) # lower right corner

ax.legend(loc = 5) # right

ax.legend(loc = 6) center left

ax.legend(loc = 7) center right

ax.legend(loc = 8) lower center

ax.legend(loc = 9) upper center

ax.legend(loc = 10) #center

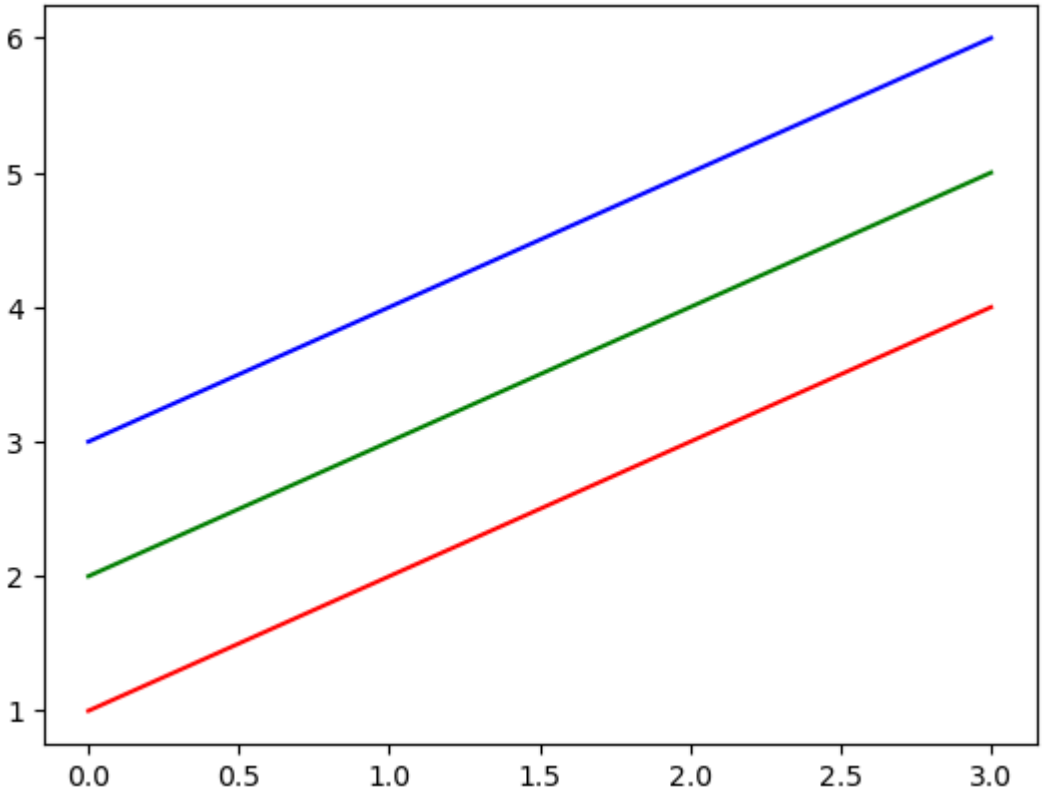
33. Control colors

we can draw different lines or curves in a plot with different colours . In the code below ,

we specify colour as the last argument to draw red blue and green lines.

In [174…

```
x16 = np.arange(1,5)

plt.plot(x16 , 'r')

plt.plot(x16+1, 'g')
plt.plot(x16+2 , 'b')
plt.show()
```



The colour names and color abbreviations are given in the following table

Colour abbreviation Colour name

b blue

c cyan

g green

k black

m magenta

r red

w white

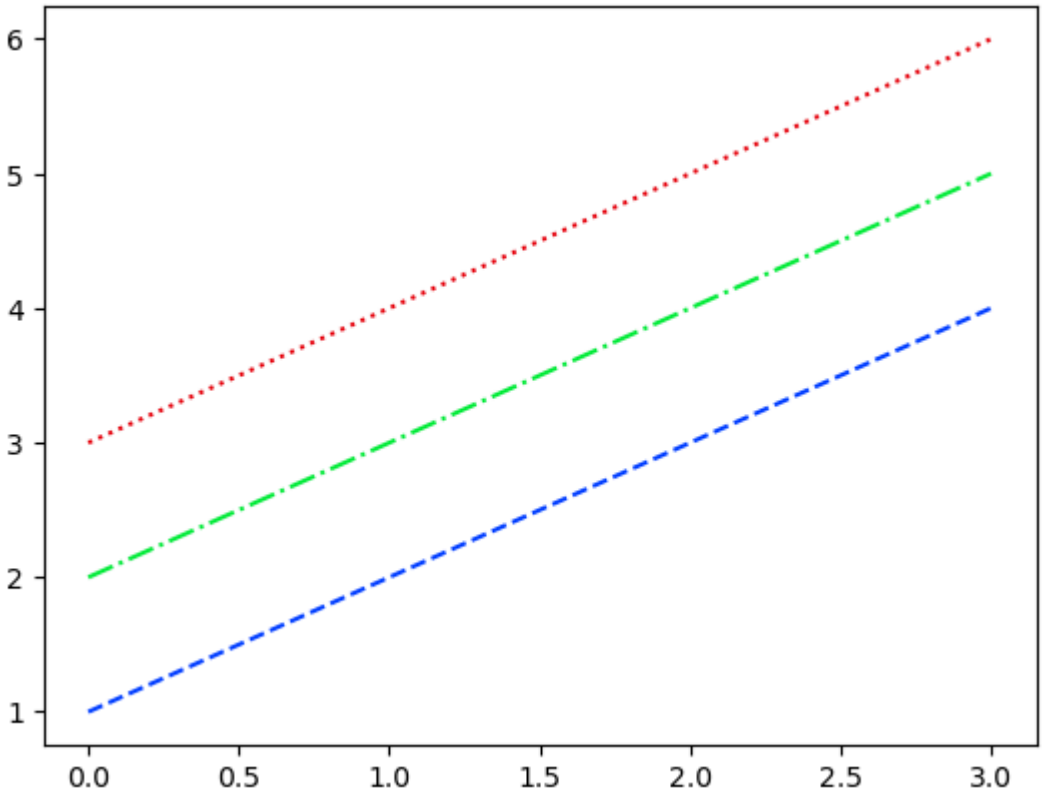
y yellow

34.Control line styles

Matplotlib provides us different line styles option to draw curves or plots in the code below , I use different line styles to draw different plots.

In [178…

```
x16 = np.arange(1,5)

plt.plot(x16,'--',x16+1,'-.',x16+2,':')
plt.show()
```



The above code snippet generates a blue dashed line , a green dash dotted line and a red dotted line .

All the available line styles are available in the following table:

Style abbreviation Style

- solid line

-- dashed line

-. dash-dot line

: dotted line

Now, we can see the default format string for a single line plot is 'b-'.

Summary

In this project, I discuss Matplotlib (the basic plotting library in Python) and throw some light on various charts and customization techniques associated with it.

In particular, I discuss Matplotlib object hierarchy, Matplotlib architecture, Pyplot and Object-Oriented architecture. I also discuss subplots which is very important tool to create graphics in Matplotlib.

Then, I discuss various types of plots like line plot, scatter plot, histogram, bar chart, pie chart, box plot, area chart and contour plot.

Finally, I discuss various customization techniques. I discuss how to customize the graphics with styles. I discuss how to add a grid and how to handle axes and ticks. I discuss how to add labels, title and legend. I discuss how to customize the charts with colours and line styles.

In [ ]: