Matplotlib

This project is all about Matplotlib, the basic data visualization tool of Python programming language. I have discussed Matplotlib object hierarchy, various plot types with Matplotlib and customization techniques associated with Matplotlib.

This project is divided into various sections based on contents which are listed below:-

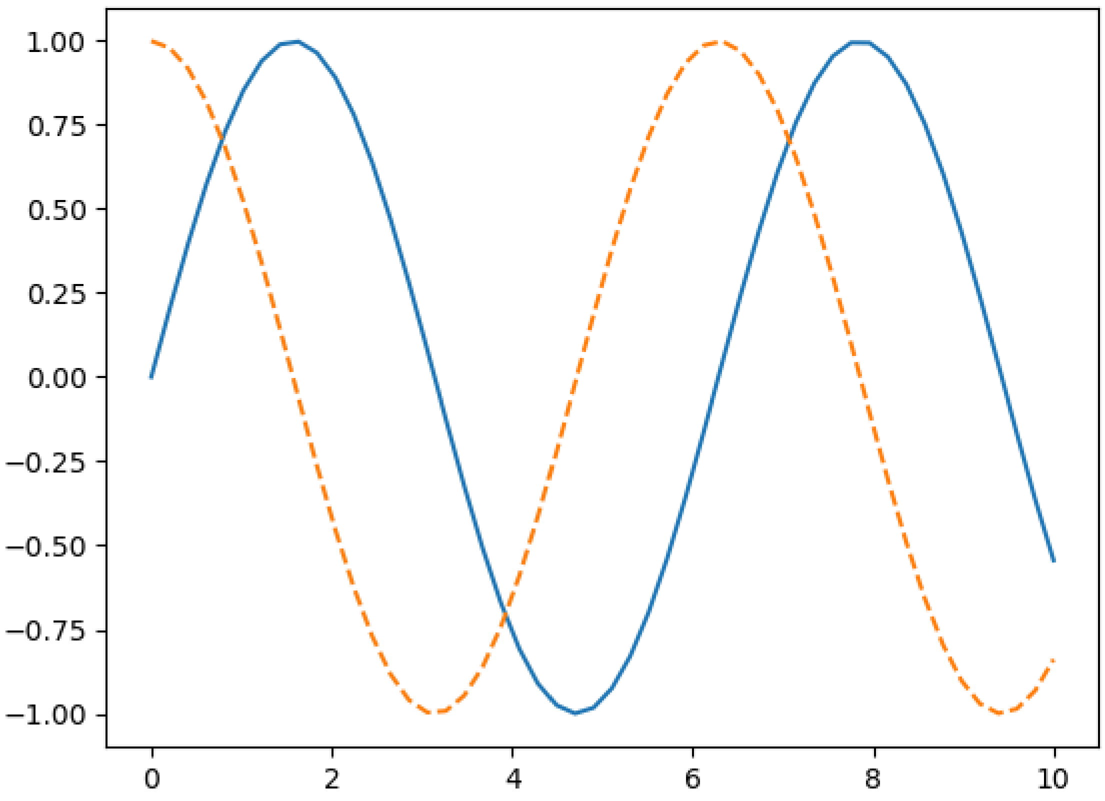
Table of contents

1. Import Matplotlib

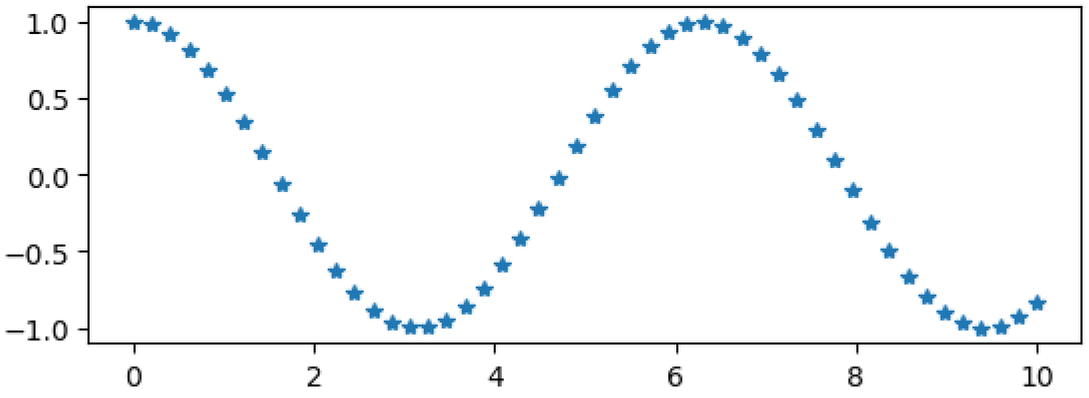
In [1]:
```
# Import dependencies
import numpy as np
import pandas as pd
```

In [2]:
```
# Import matplotlib
import matplotlib.pyplot as plt
```

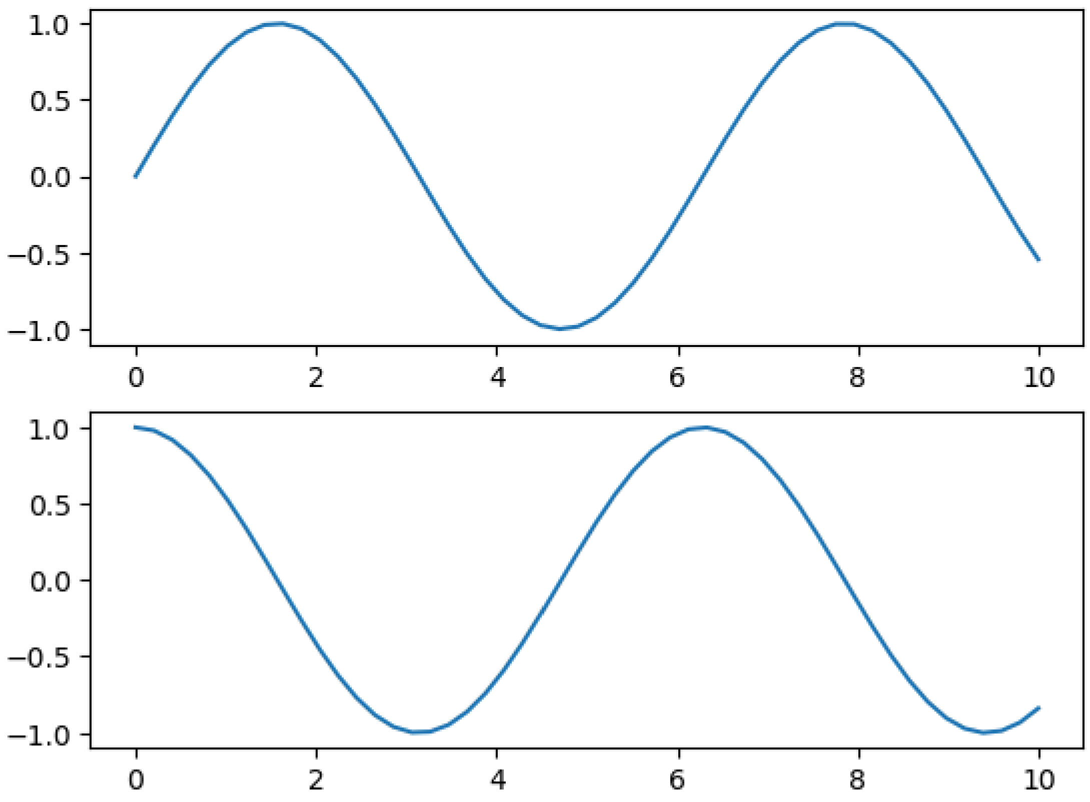
In [3]:
```
%matplotlib inline
x1 = np.linspace(0,10,50)

# create a plot figure()
#fig = plt.figure()

plt.plot(x1, np.sin(x1), '-')
plt.plot(x1,np.cos(x1), '--')
#plt.plot(x1,np.tan(x1), '--')
plt.show()
```



In [4]:
```
# create the first of two panels and set current axis
plt.subplot(2, 1, 1)   # (rows, columns, panel number)
plt.plot(x1, np.cos(x1), '*')
plt.show()
```



In [5]:
```
# create a plot figure
plt.figure()

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

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

plt.show()
plt.show()
```

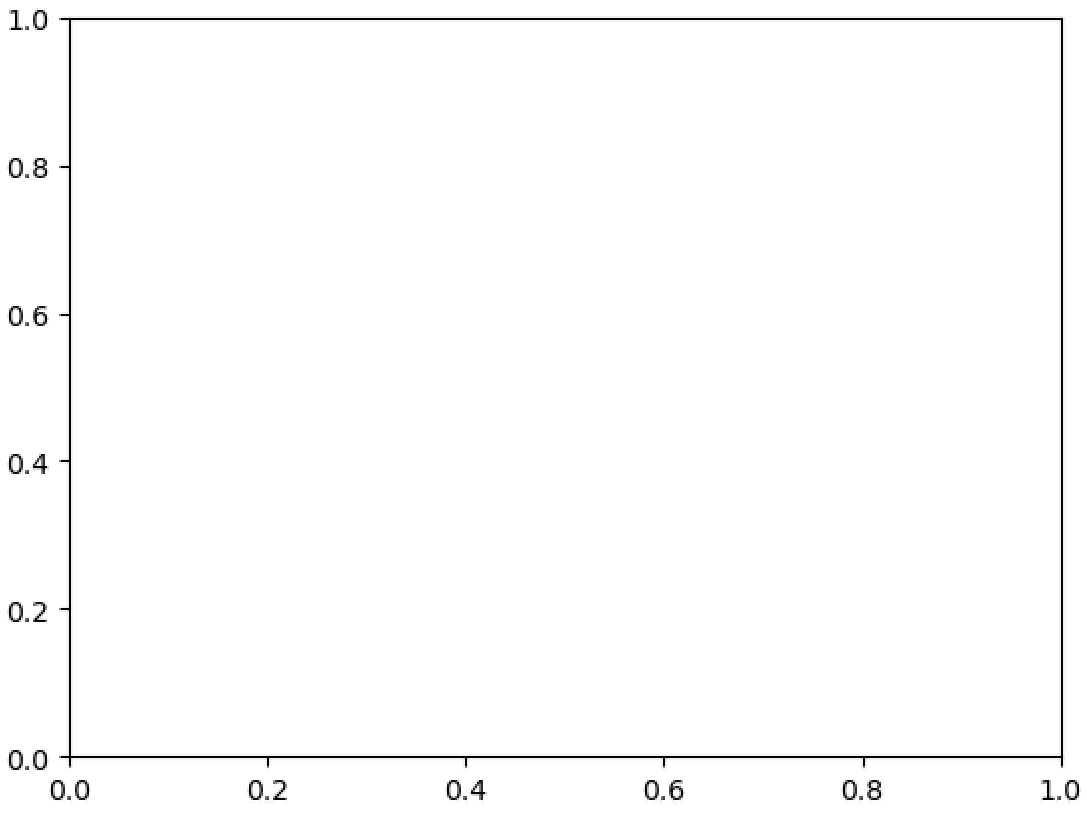


```
In [6]: # get current figure information

        print(plt.gcf())
```

```
Figure(640x480)
```

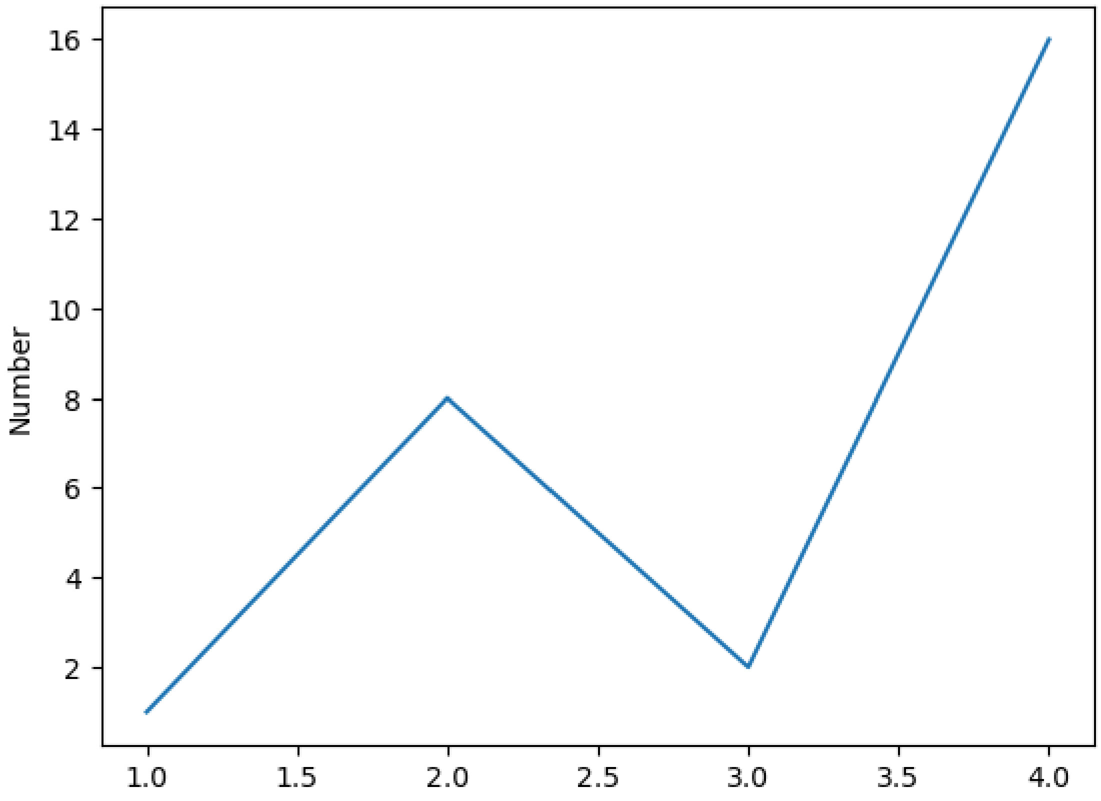
```
In [7]: # get current axis information

        print(plt.gca())
        plt.show()
```

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



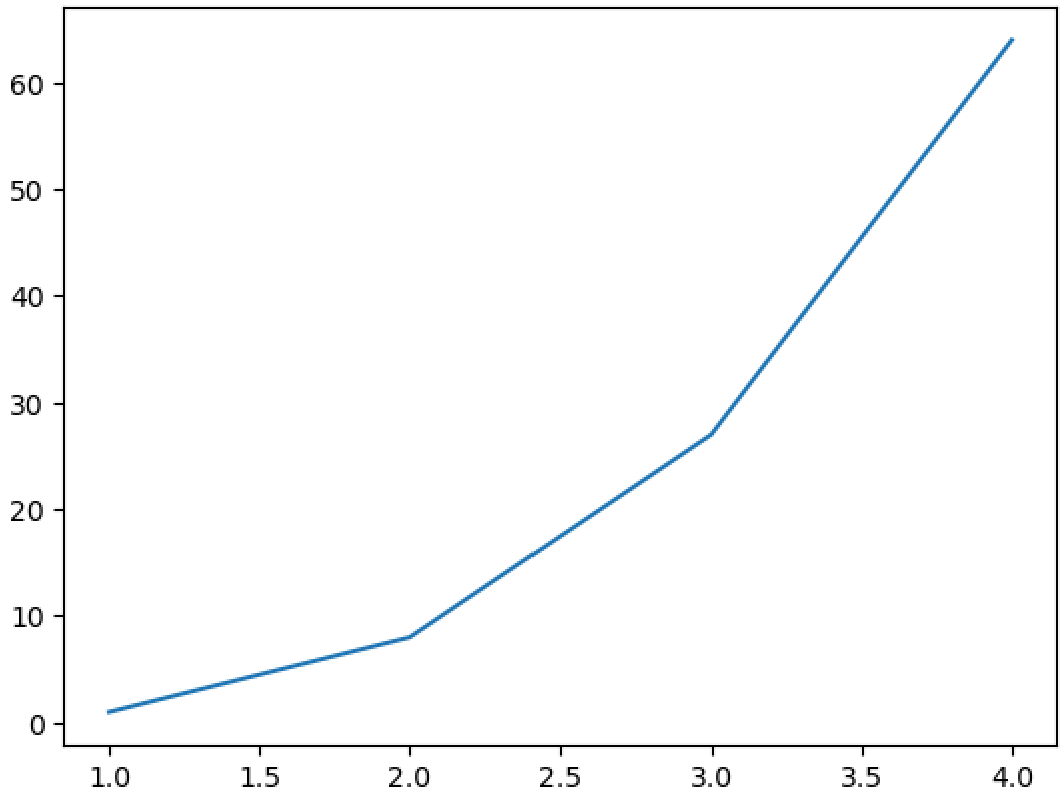
2. visualization with pyplot

In [8]:
```
plt.plot([1,2,3,4], [1,8,2,16])
plt.ylabel('Number')
plt.show()
```



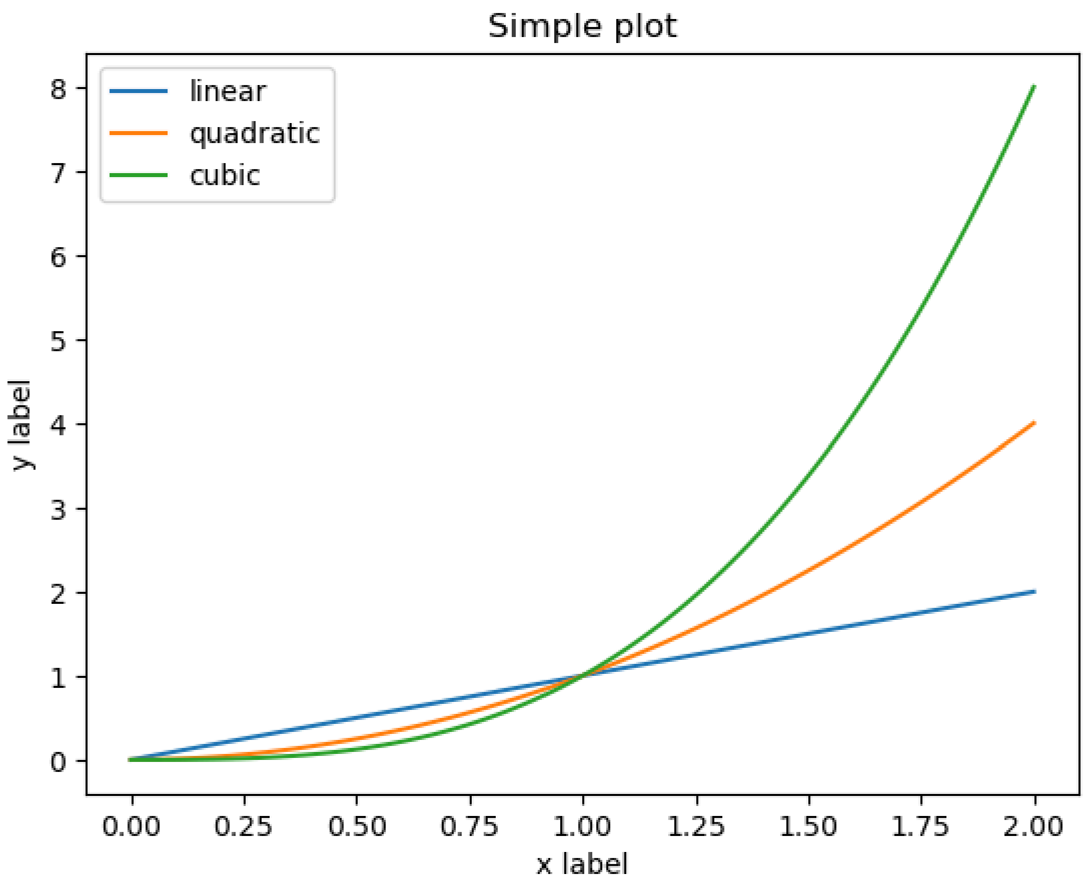
3.Plot() - A versatile command

In [9]:
```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4, ], [1, 8, 27, 64])
plt.show()
```



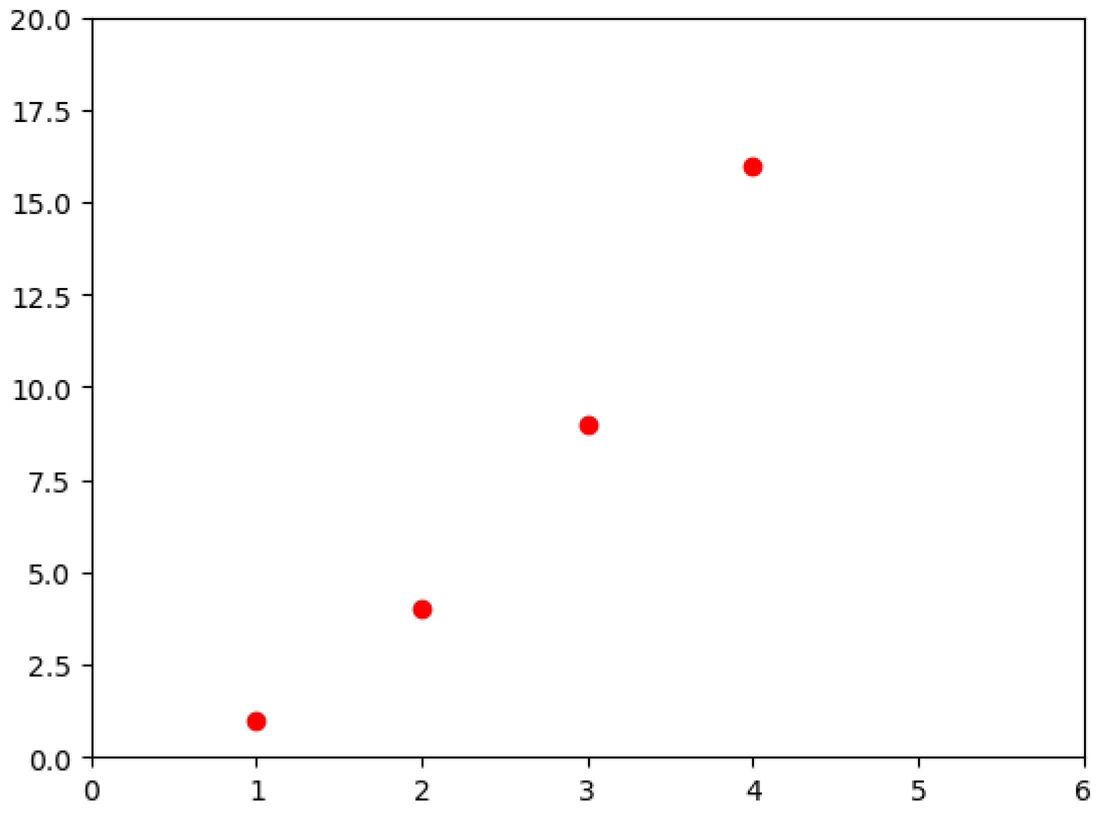
4.State-machine interface

In [10]:
```
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()
```



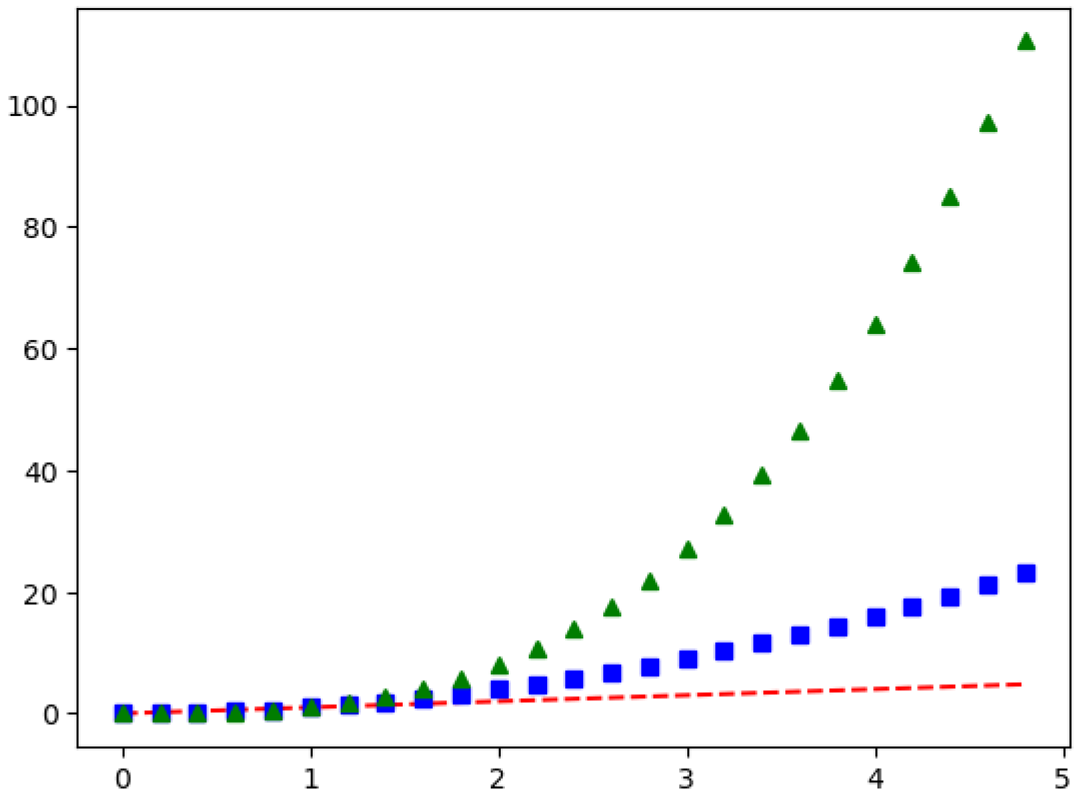
5.Formatting the style of plot

```
In [11]: plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
         plt.axis([0, 6, 0, 20])
         plt.show()
```



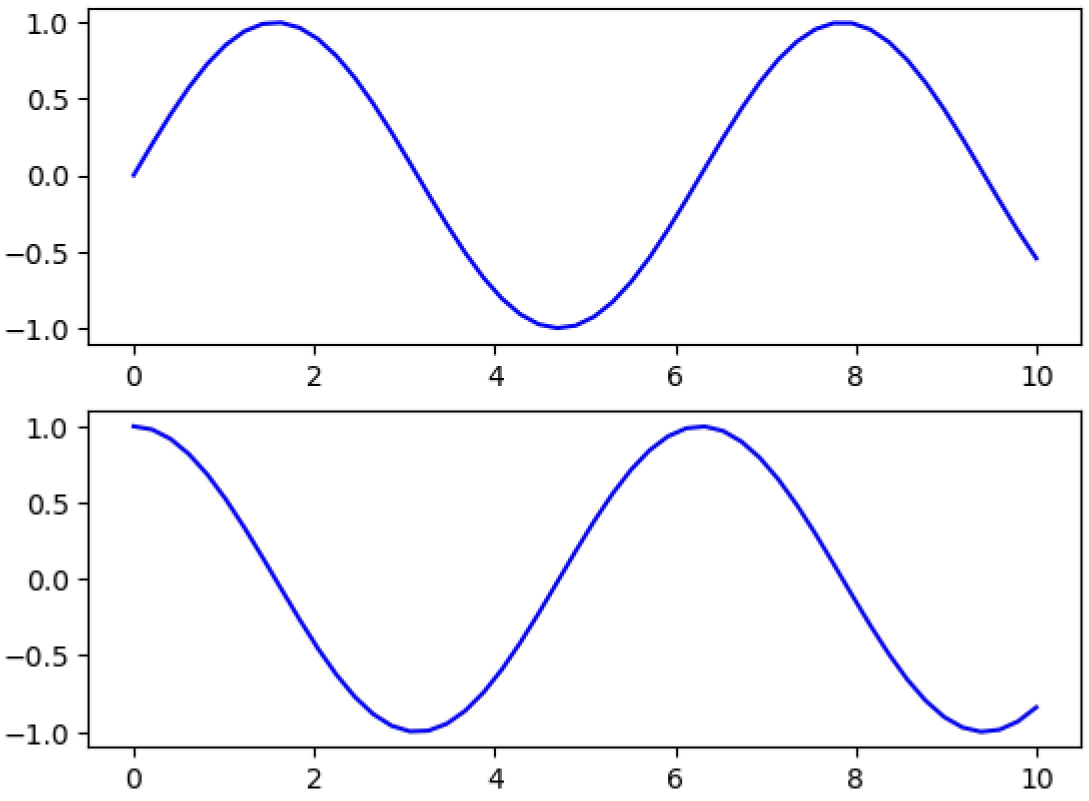
6.Working with numpy arrays

In [12]:
```
# evenly sampled time at 200ms intervals
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()
```



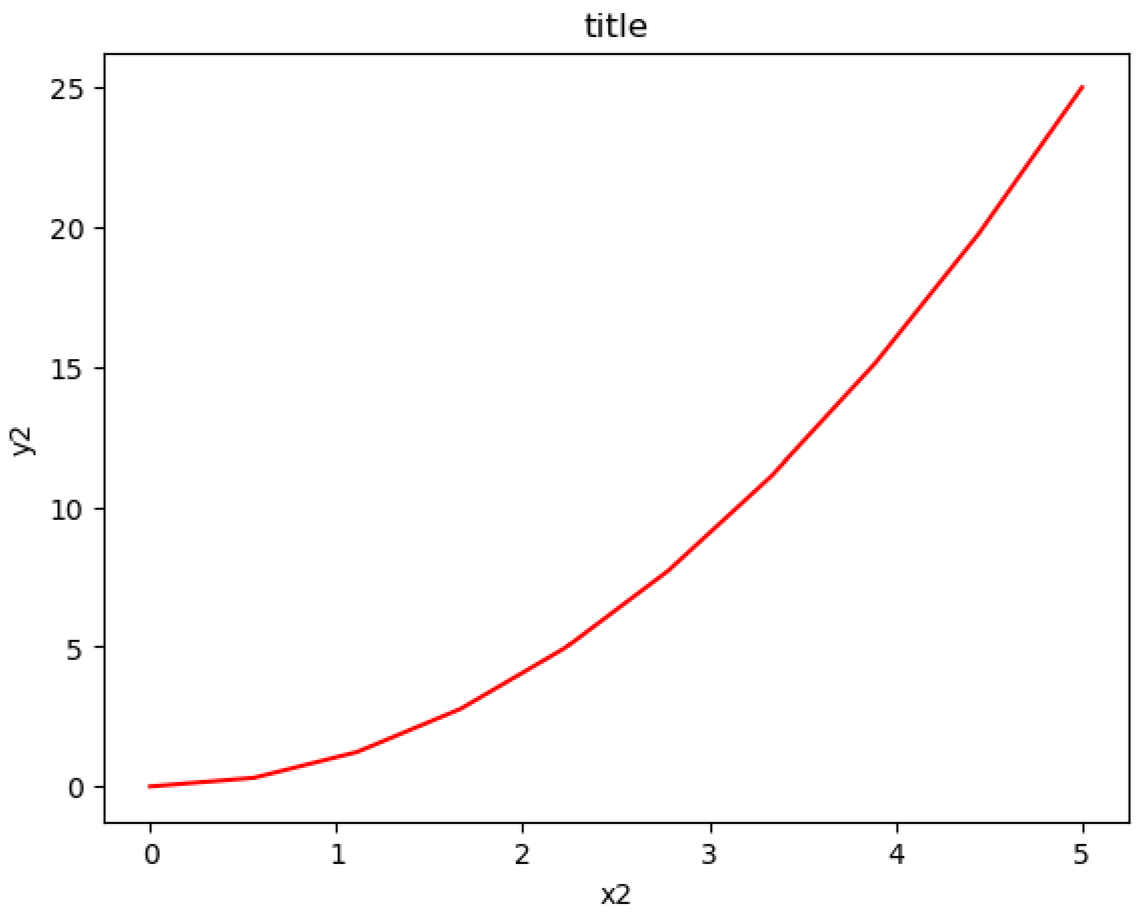
7.Object-oriented API

In [13]:
```
# First creat a 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-');
plt.show()
```



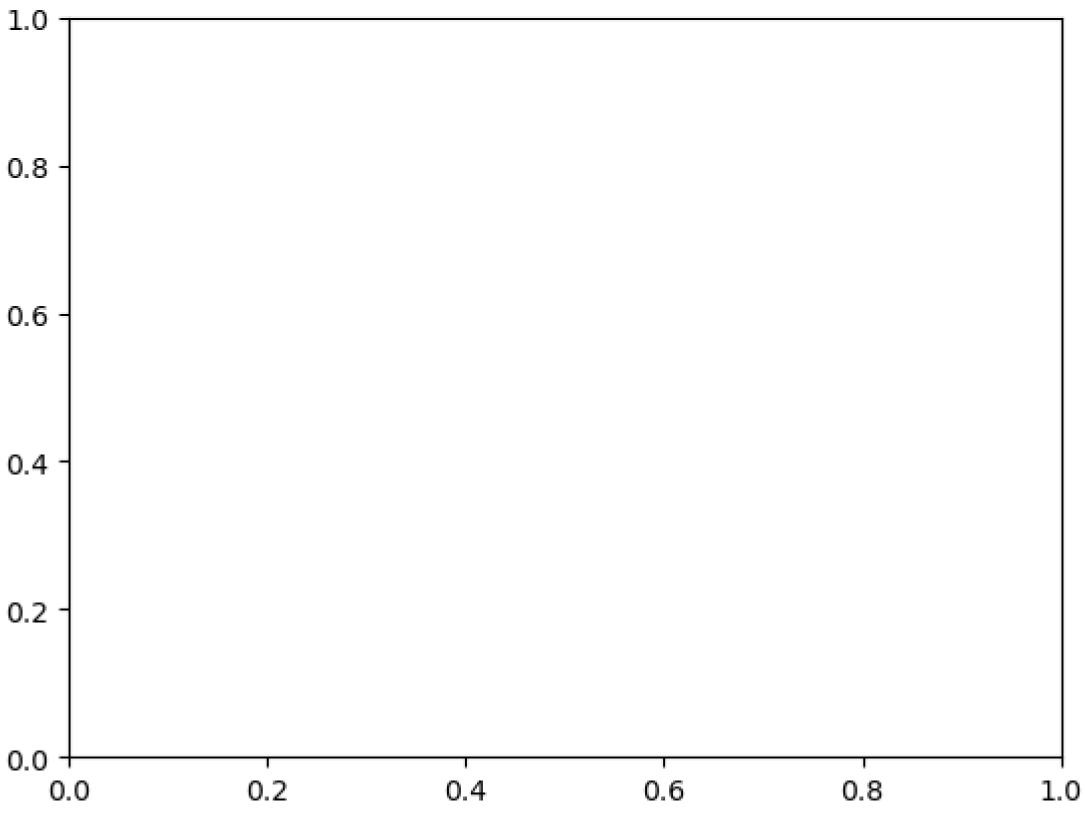
8. Objects and reference

```
In [14]: 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');
         plt.show()
```



9.Figure and axes

In [15]:
```
fig = plt.figure()

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

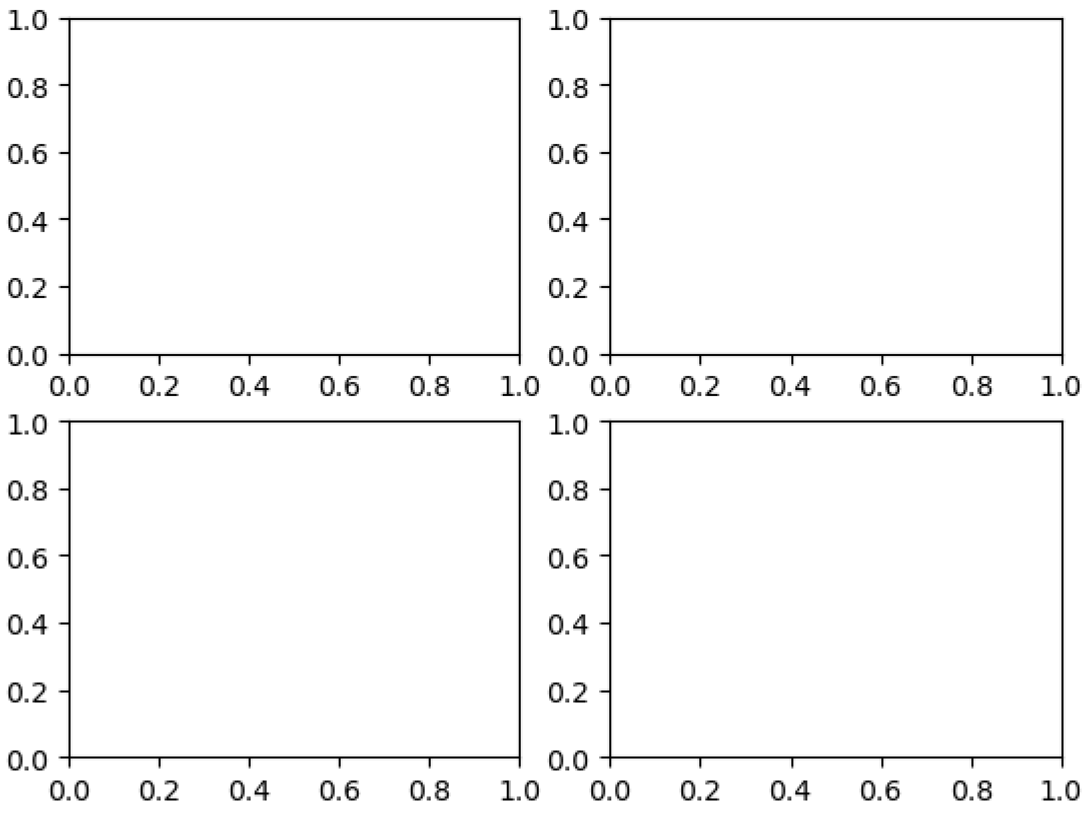


10.Figure and subplots

In [16]:
```
fig = plt.figure()
```

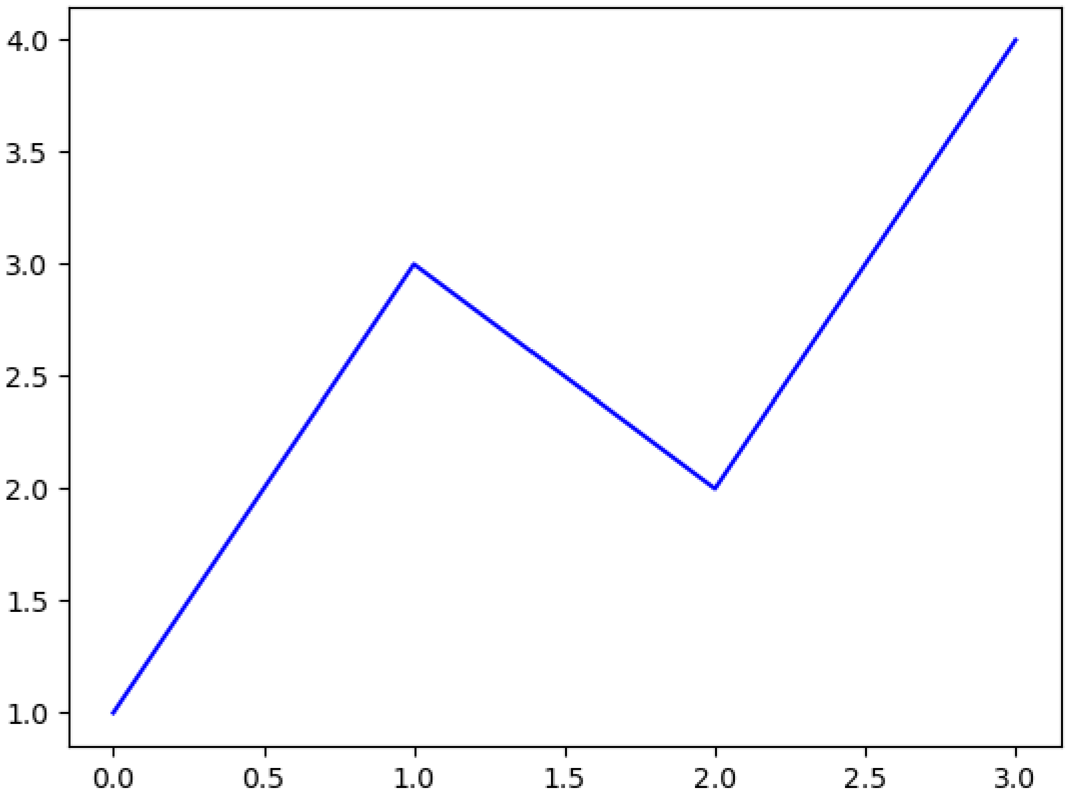
In [17]:
```
ax1 = fig.add_subplot(2, 2, 1)
```

In [18]:
```
ax2 = fig.add_subplot(2, 2, 2)

ax3 = fig.add_subplot(2, 2, 3)

ax4 = fig.add_subplot(2, 2, 4)

plt.show()
```



First plot with matplotlib

In [19]:
```
plt.plot([1, 3, 2, 4], 'b-')
plt.show()
```



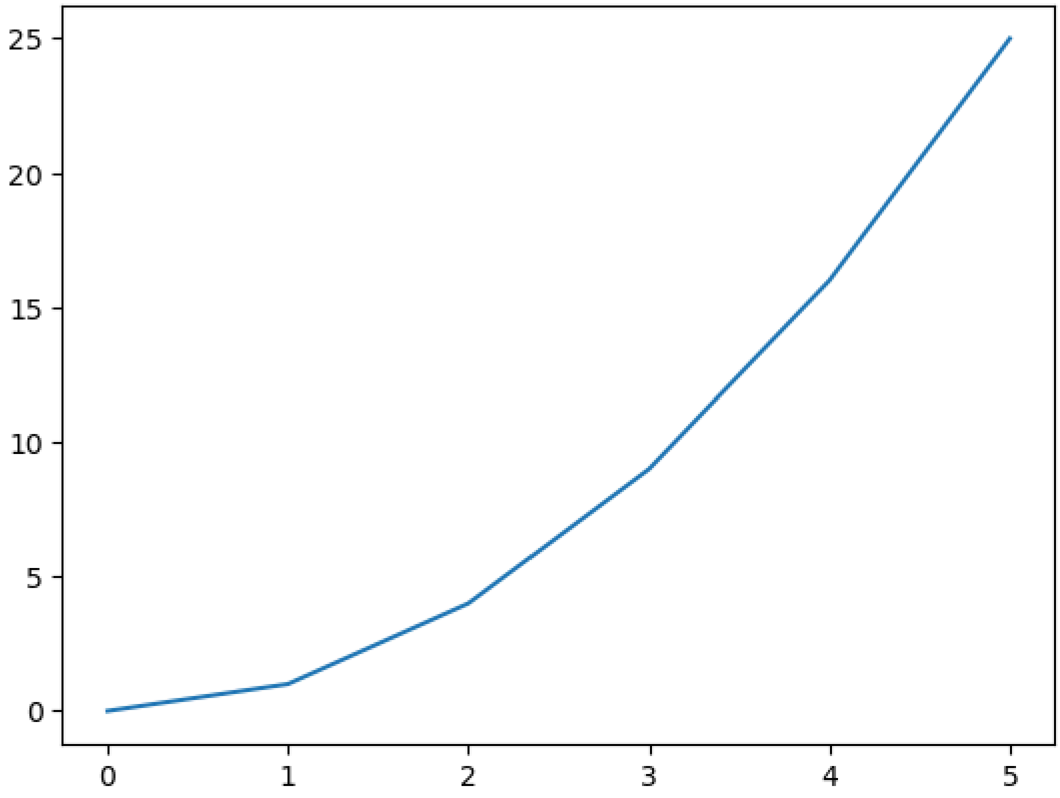
11.Specify both list

```
In [20]: x3 = range(6)
```

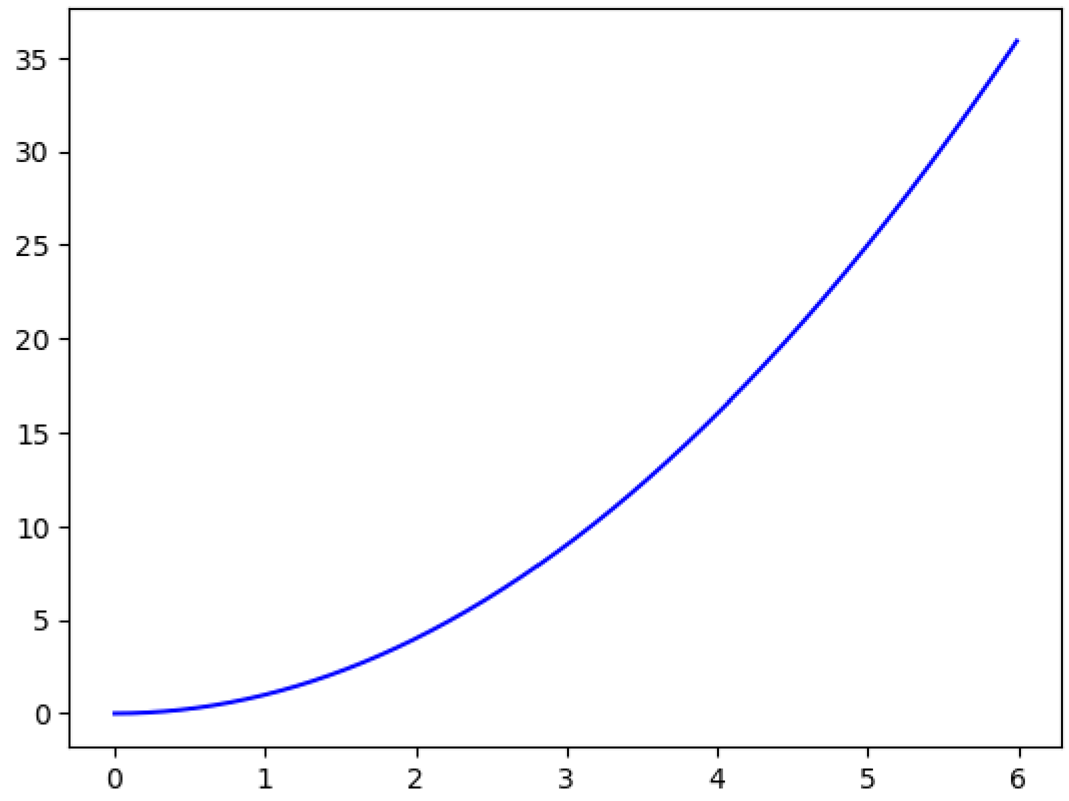
```
In [21]: plt.plot(x3, [xi**2 for xi in x3])
```

```
Out[21]: [<matplotlib.lines.Line2D at 0x21f361f0050>]
```

```
In [22]: plt.show()
```

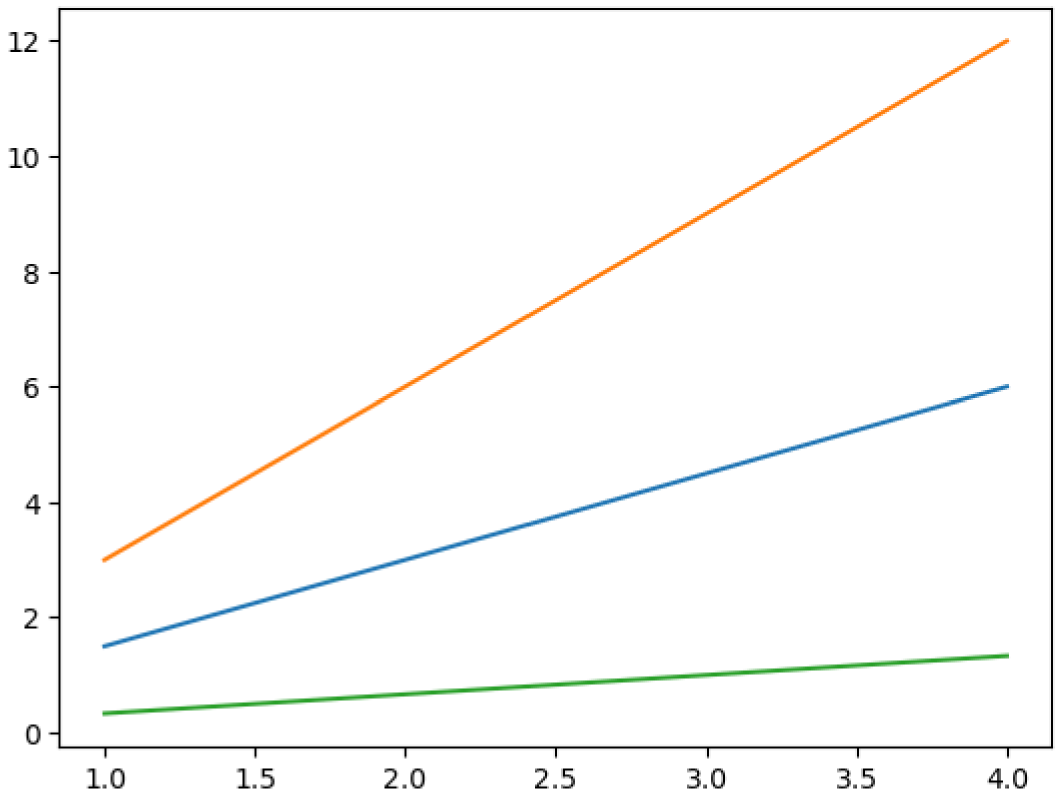


```
In [23]: x3 = np.arange(0.0, 6.0, 0.01)

         plt.plot(x3, [xi**2 for xi in x3], 'b-')

         plt.show()
```



12.Multiline plots

In [24]:
```
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

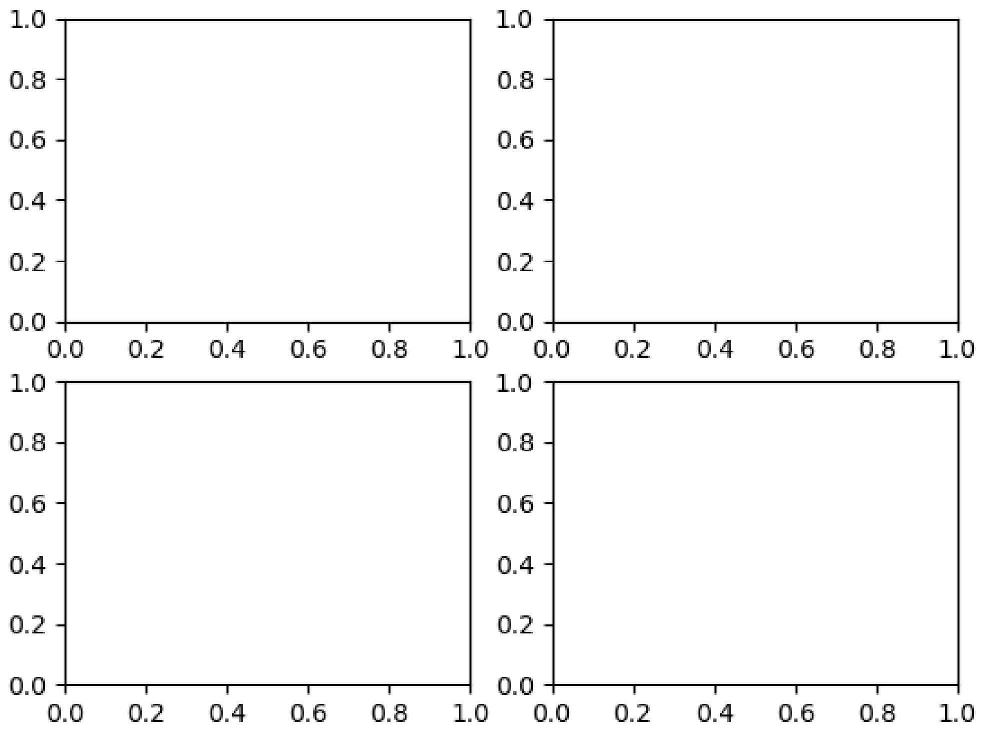
There are different parts of a plot. These are title, legend, grid, axis and labels etc. These are denoted in the following figure:-

14. saving the plot

In [25]:
```
# saving the figure
fig.savefig('plot1.png')
```

In [26]:
```
# Explore the contents of figure

from IPython.display import Image

Image('plot1.png')
```

Out[26]:



Explore supported file formats

In [27]:

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

Out[27]:

```
{'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

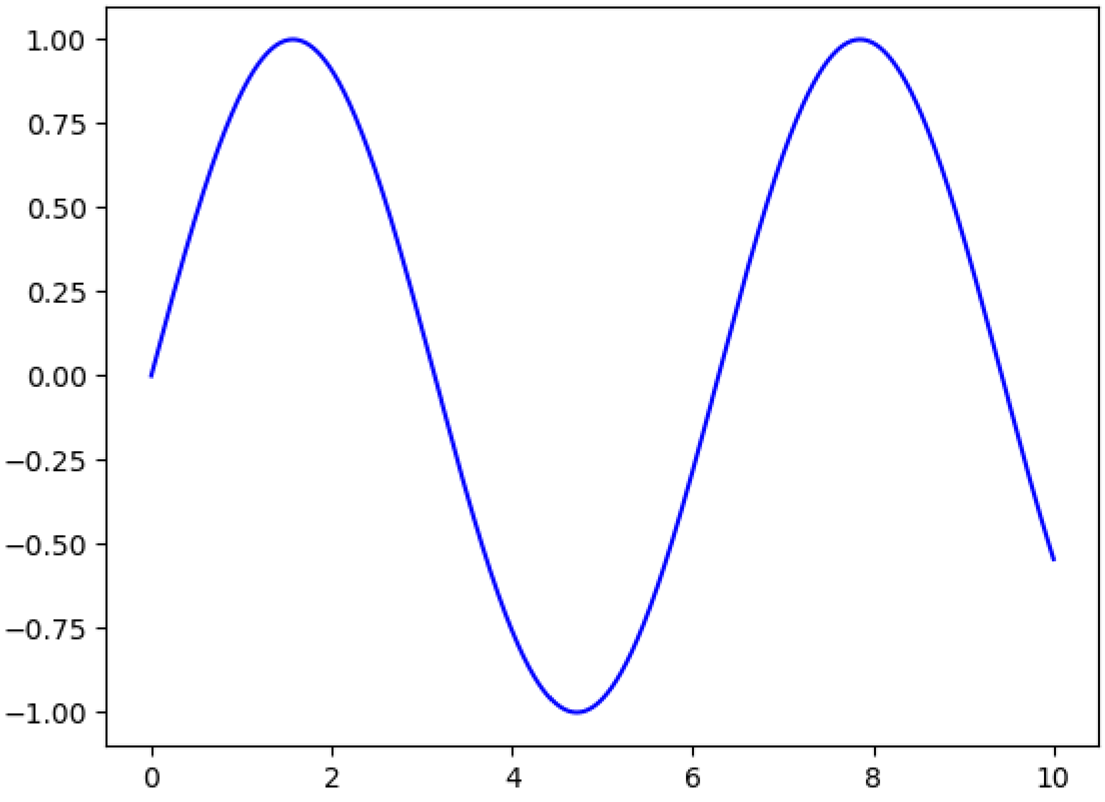
- We can use the following commands to draw the simple sinusoid line plot:-

In [28]:

```
# create figure and axes first
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-');
```

In [29]:
```
plt.show()
```



16 Scatter Plot

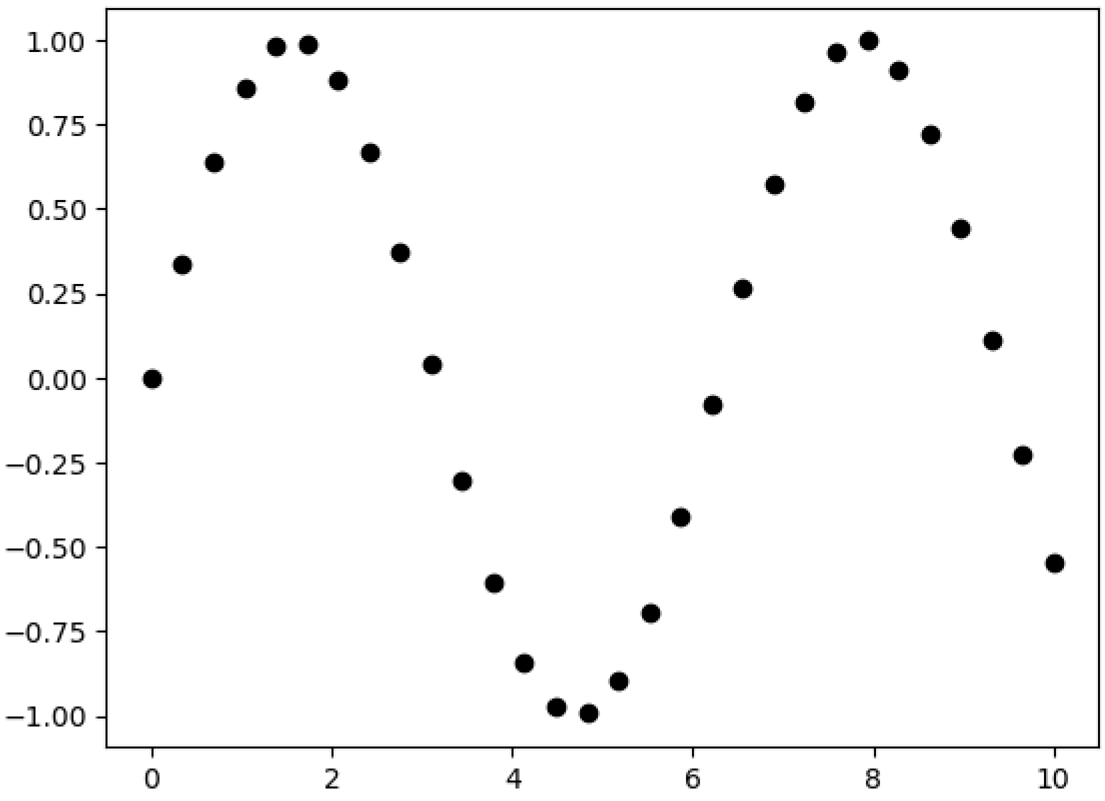
- Another commonly used plot type is the scatter plot.Here the Points are represented individually with a dot or a circle.

Scatter Plot with plt.plot()

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

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

y7 = np.sin(x7)

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

In [31]:
```
plt.show()
```



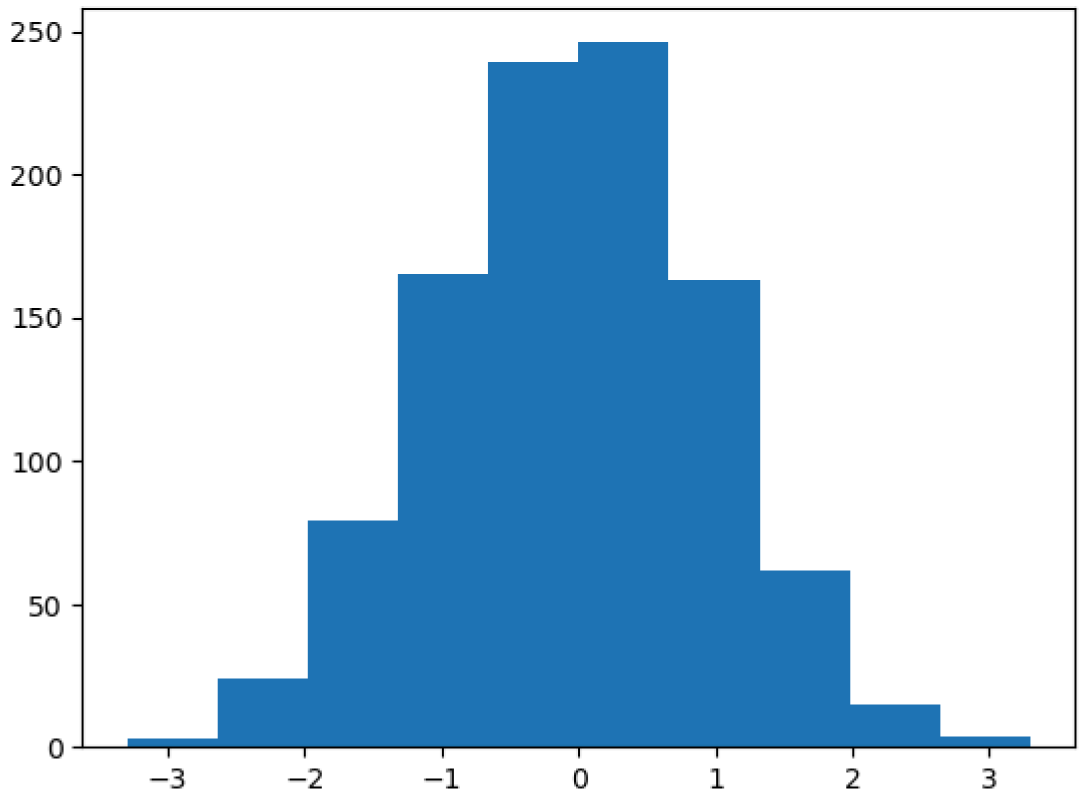
17. Histogram

- 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-overlapping intervals. These categories are called bins.

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

In [32]:
```
data1 = np.random.randn(1000)

plt.hist(data1);
```

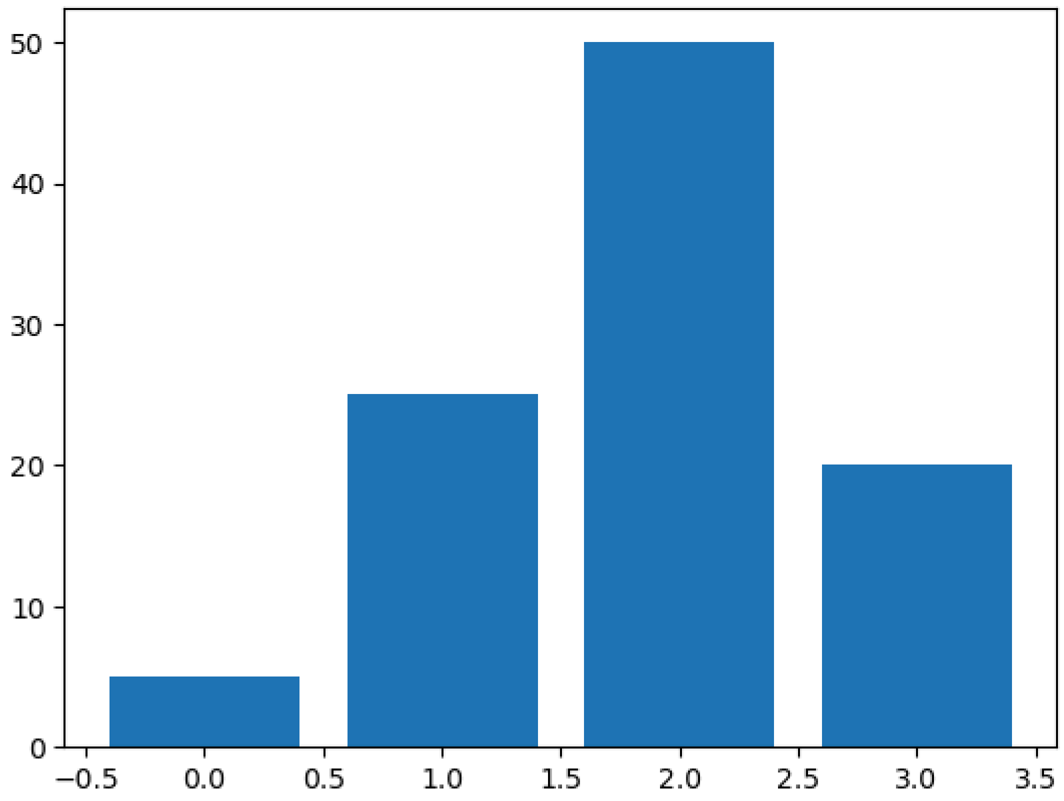
In [33]:
```
plt.show()
```



18. Bar Chart

- Bar chart display rectanglar bars either in verctical or horizantal form.Their length is proportional to the values they represent. They are used to compare two or more values.
- We can plot a bar chat using plt.bar() function. We can plot a bar chart as follows:-

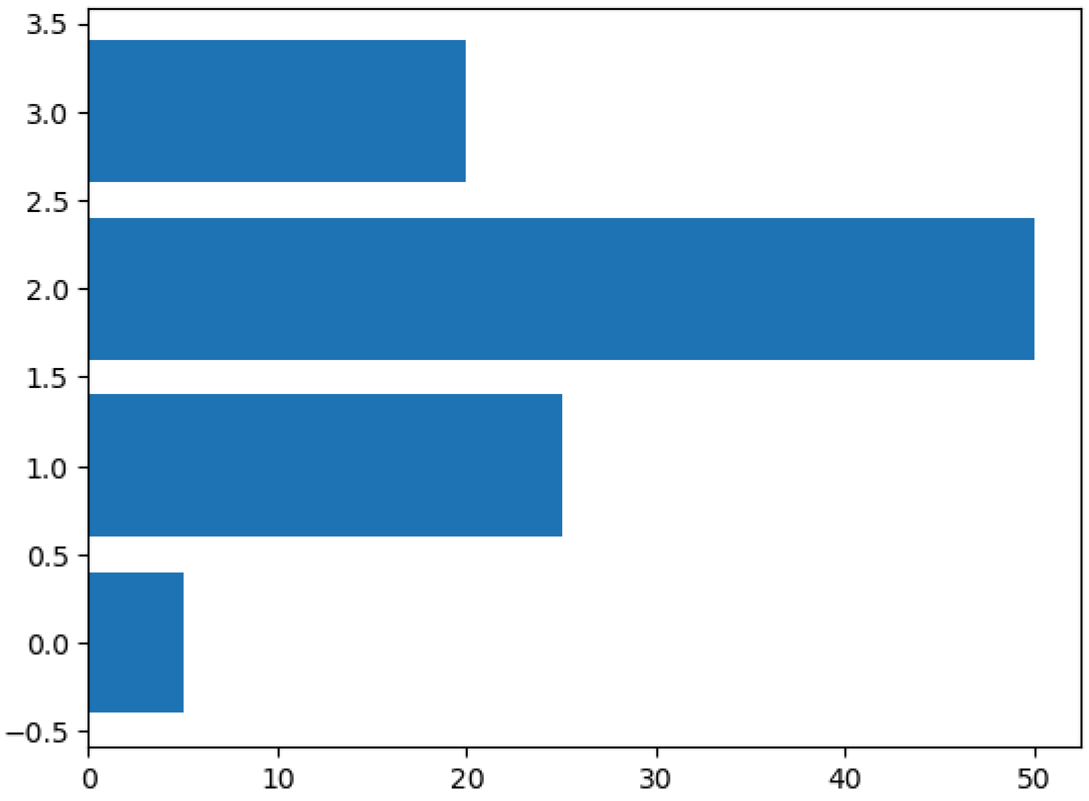
```
In [34]: data2 = [5. , 25., 50., 20.]

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

         plt.show()
```



19.Horizontal Bar Chart

- We can produce Horizantal Bar Chart using the plt.barh()function.It is the strict equivalent of plt.bar() function.

In [35]:
```
data2 = [5.,25.,50.,20.]
plt.barh(range(len(data2)),data2)
plt.show()
```

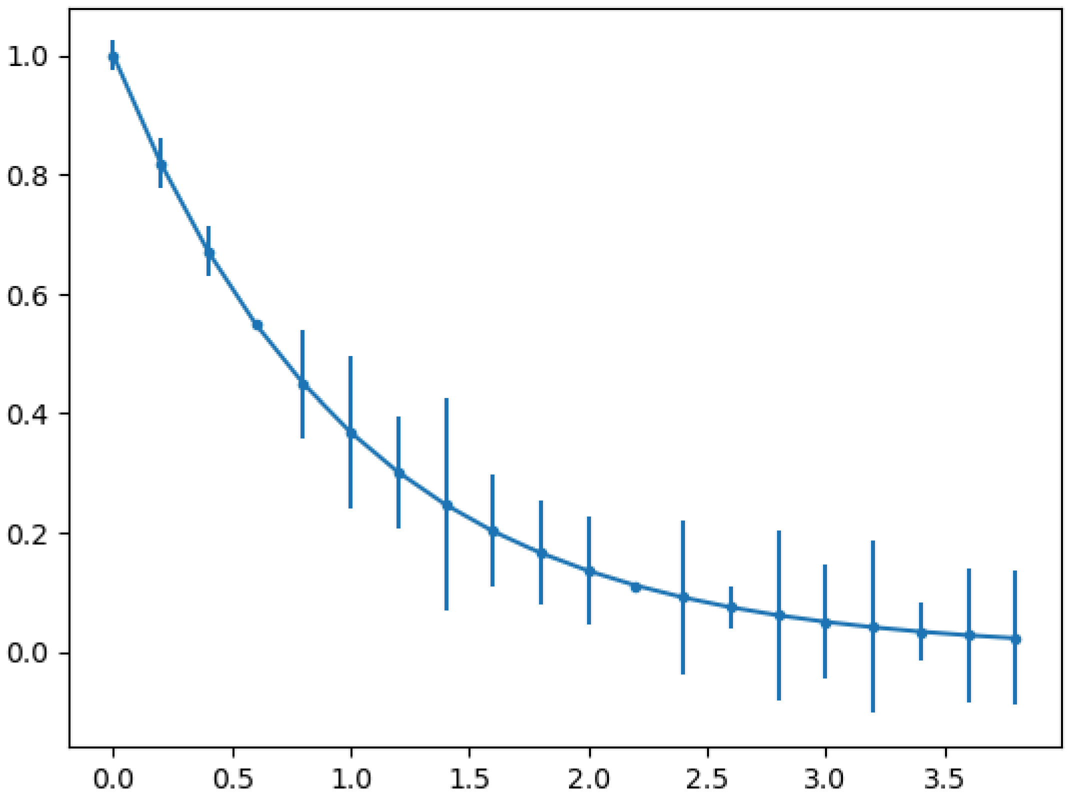


20.Error Bar Chart

In experimental design, the measurements lack perfect perfect precision.So,we have to repeat the measurements.It results in obtaining a set of values.The representation of data values is done by plotting a single data point (known as mean value of dataset) and an error bar to represent the ovarall distribution of data.

- We can use Matplotlib's errorbar() function to represent the distribution of data values. It can be done as follows:-

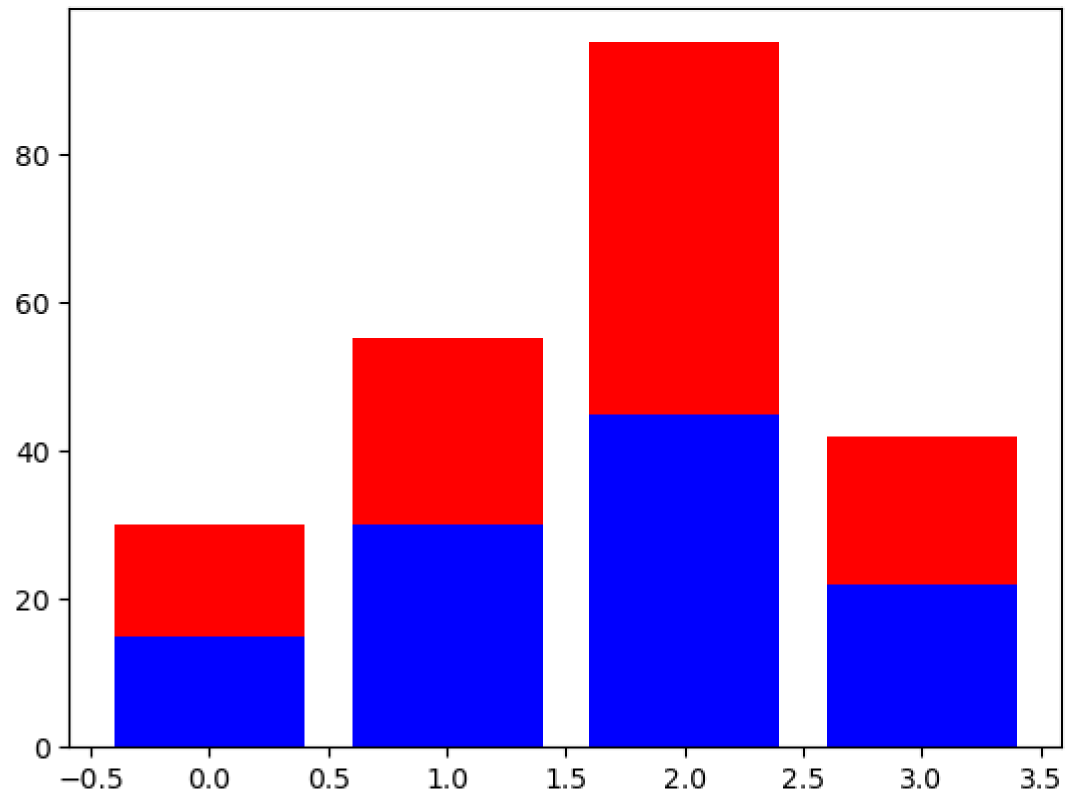
In [36]:
```
x9 = np.arange(0, 4, 0.2)

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 [37]:
```
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()
```



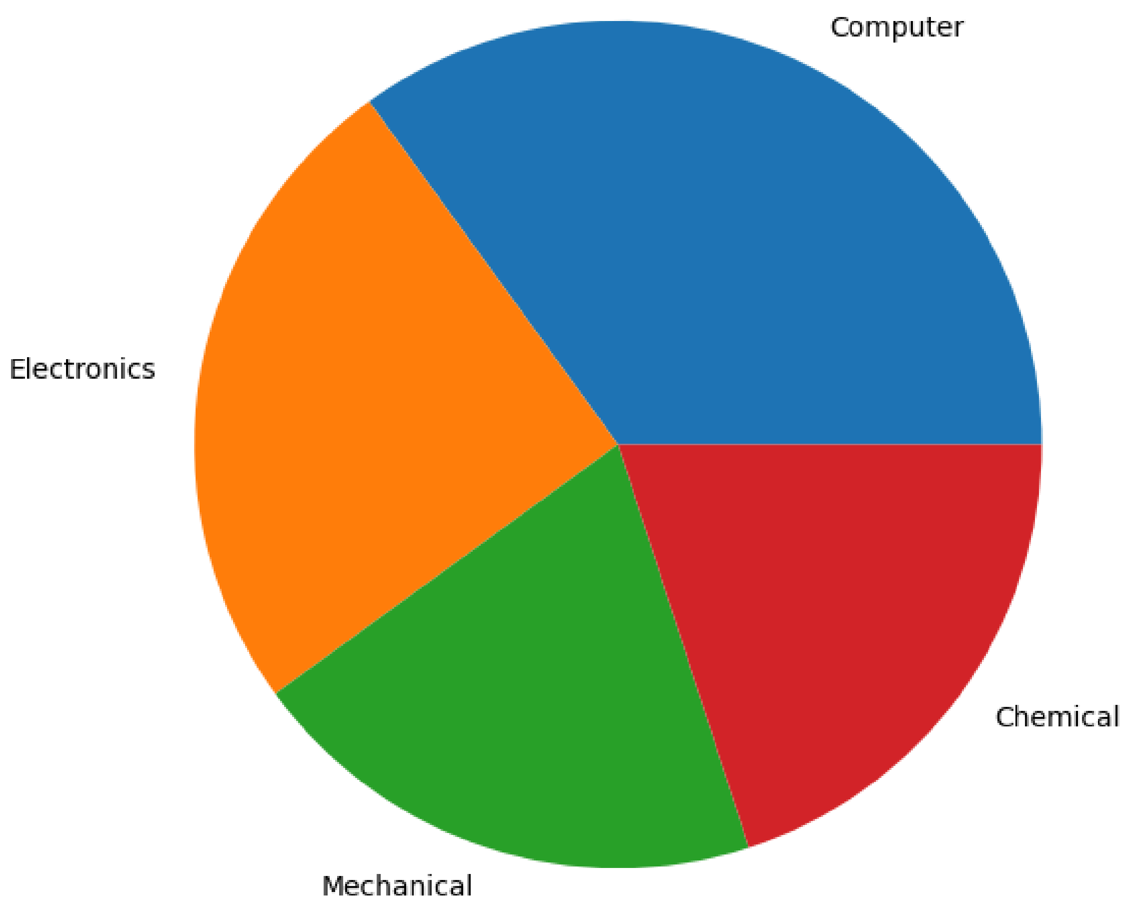
The optional bottom parameter of the plt.bar() function allows us to specify a starting position for a bar. Instead of running from zero to a value, it will go from the bottom to value. The first call to plt.bar() plots the blue bars. The second call to plt.bar() plots the red bars, with the bottom of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

Matplotlib 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).

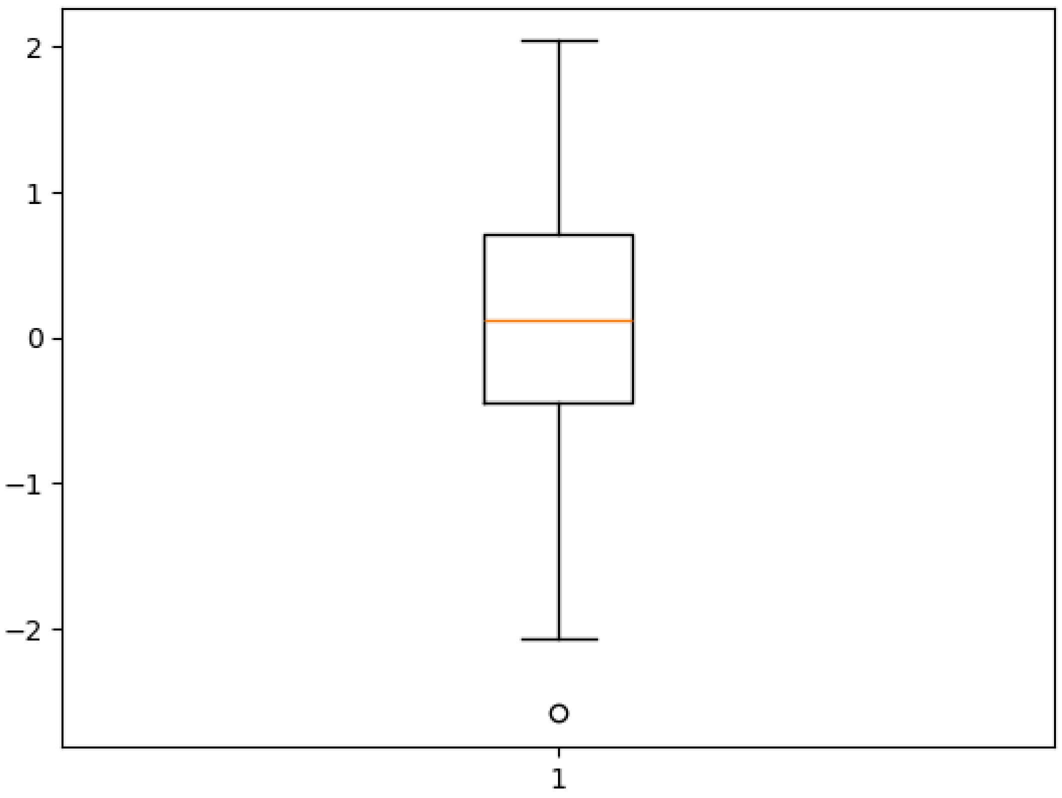
```
In [38]: plt.figure(figsize=(7,7))
         x10 = [35,25,20,20]
         labels =  ['Computer','Electronics','Mechanical','Chemical']
         plt.pie(x10,labels=labels);
         plt.show()
```



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 [39]:
```
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 describe 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.

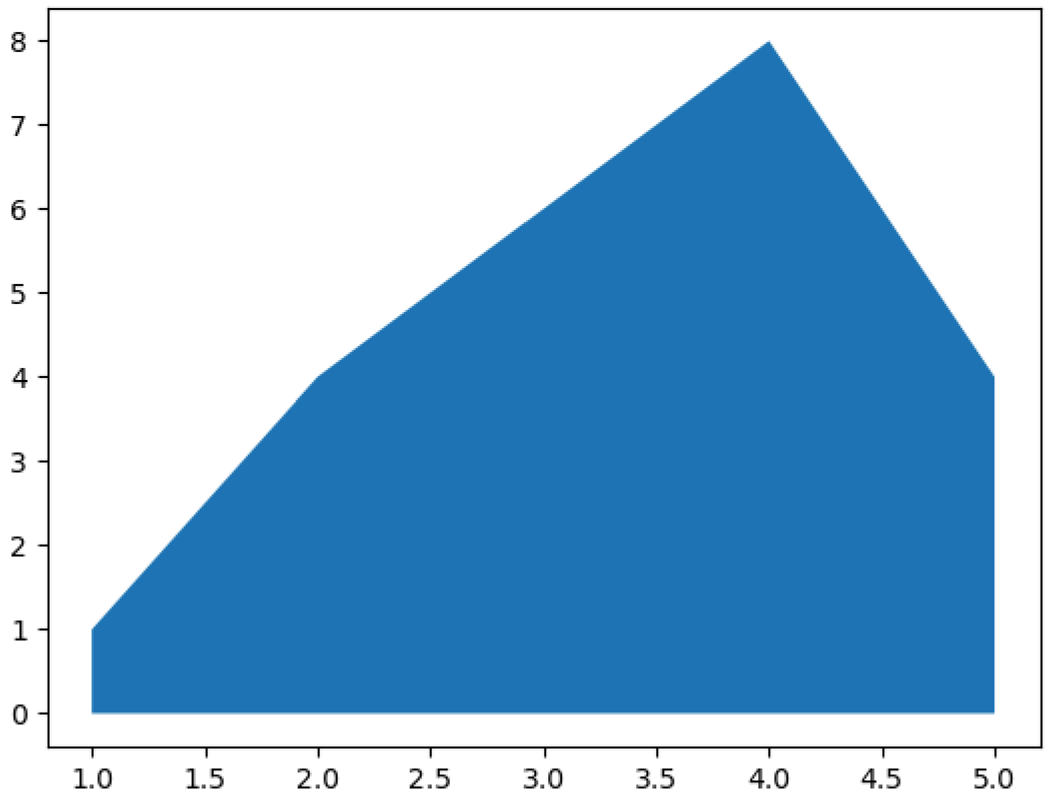
• Values further from the whiskers are shown with a cross marker.

24. Area Chart

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

We can create an Area chart as follows:-

In [40]:
```
# create some data
x12 = range(1,6)
y12 = [1,4,6,8,4]

# Area plot
plt.fill_between(x12,y12)
plt.show()
```



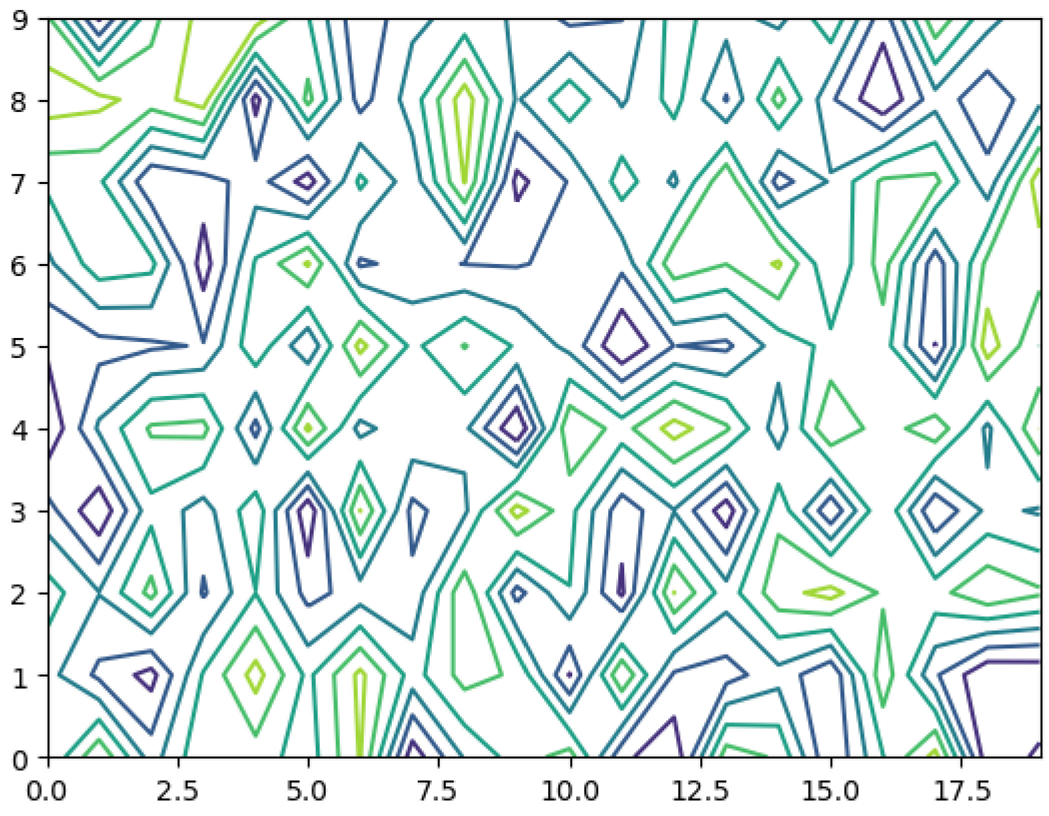
- I have created a basic Area chart. I could also use the stackplot function create the Area Chart as follows:-
- plt.stackplot(x12,y12)
- The fill_between() function is more conveninent for future customization.

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

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

In [41]:
```
# 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 was 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.style.availabe)

In [42]:
```
# 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', 'petroff10', 'seaborn-v0_8', 'seaborn-v0_8-bright', 'seaborn-v0_8-colorblin
d', 'seaborn-v0_8-dark', 'seaborn-v0_8-dark-palette', 'seaborn-v0_8-darkgrid', 's
eaborn-v0_8-deep', 'seaborn-v0_8-muted', 'seaborn-v0_8-notebook', 'seaborn-v0_8-p
aper', 'seaborn-v0_8-pastel', 'seaborn-v0_8-poster', 'seaborn-v0_8-talk', 'seabor
n-v0_8-ticks', 'seaborn-v0_8-white', 'seaborn-v0_8-whitegrid', 'tableau-colorblin
d10']
```

We can set the Styles for Matplotlib plots as follows:-

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

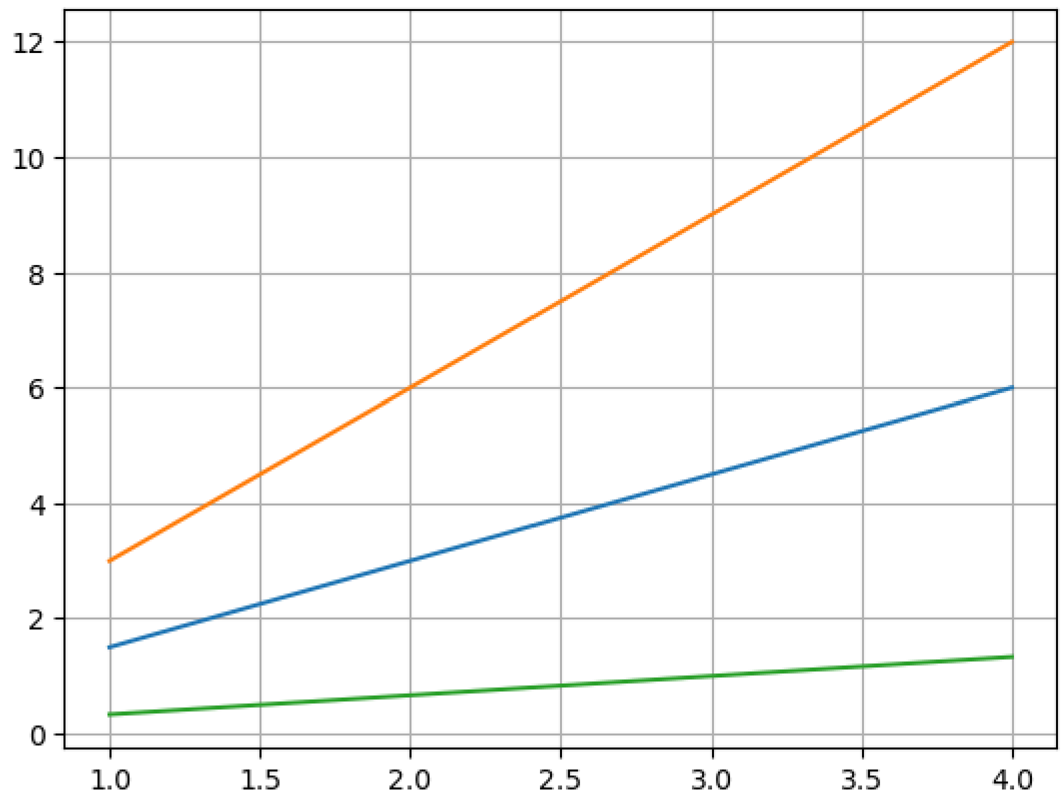
# set styles for plots plt.style.use('seaborn-bright')

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.

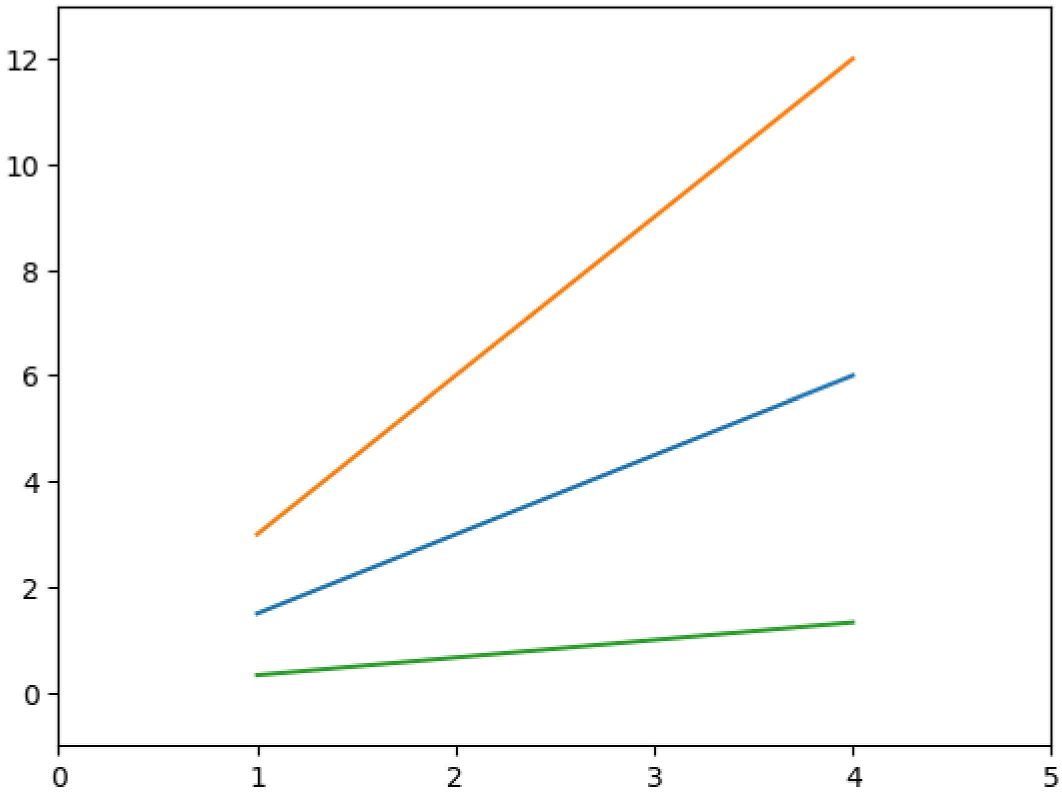
```
In [43]: x15 = np.arange(1,5)

         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 [44]:
```
x15 = np.arange(1,5)

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

plt.axis() # shows the current axis limits values

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 parameters, it returns the actual axis limits.

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

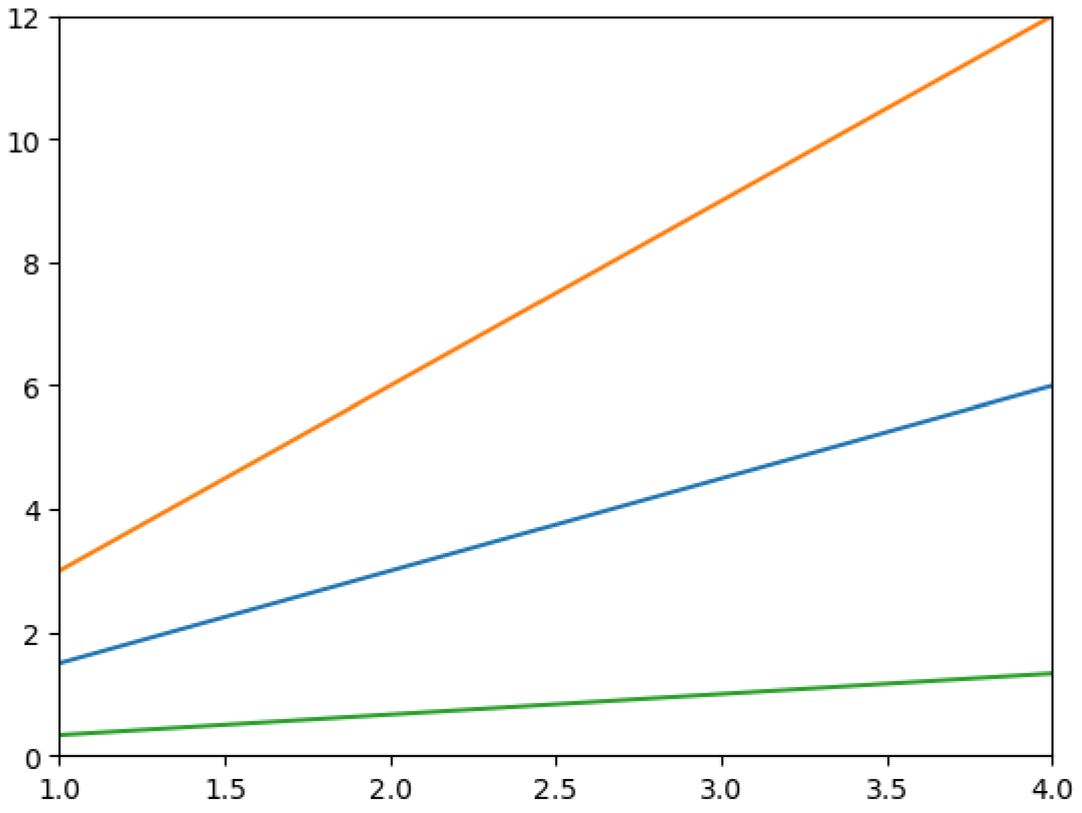
The list of four values are the keyword arguments [xmin, xmax, ymin, ymax] allows the minimum and maximum limits for X and Y axis respectively.

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

In [45]:
```
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[45]:
```
(0.0, 12.0)
```

In [46]:
```
plt.show()
```



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().

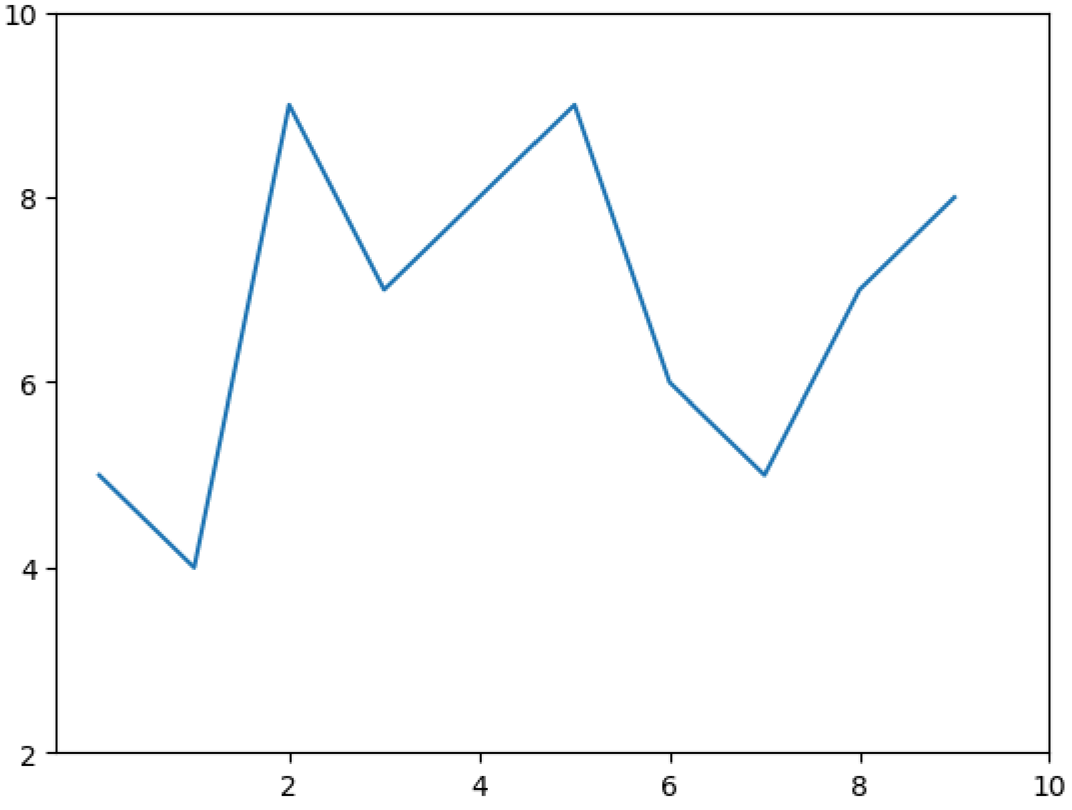
Executing with no arguments, the tick function returns the current ticks' locations and the labels corresponding to each of them.

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

1. Locations of the ticks
2. Labels to draw at these locations.

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

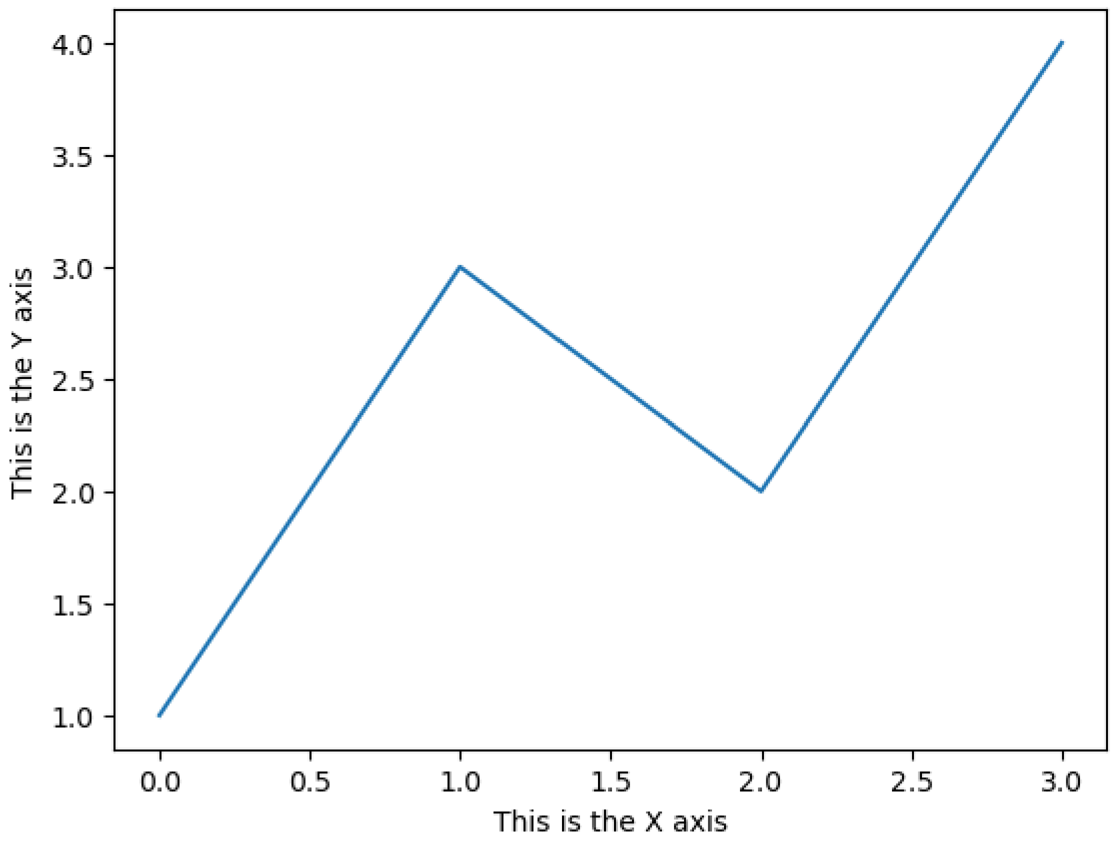
In [47]:
```
u = [5,4,9,7,8,9,6,5,7,8]

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 a plot is the axes labels,since they specify the type of data we are plotting.

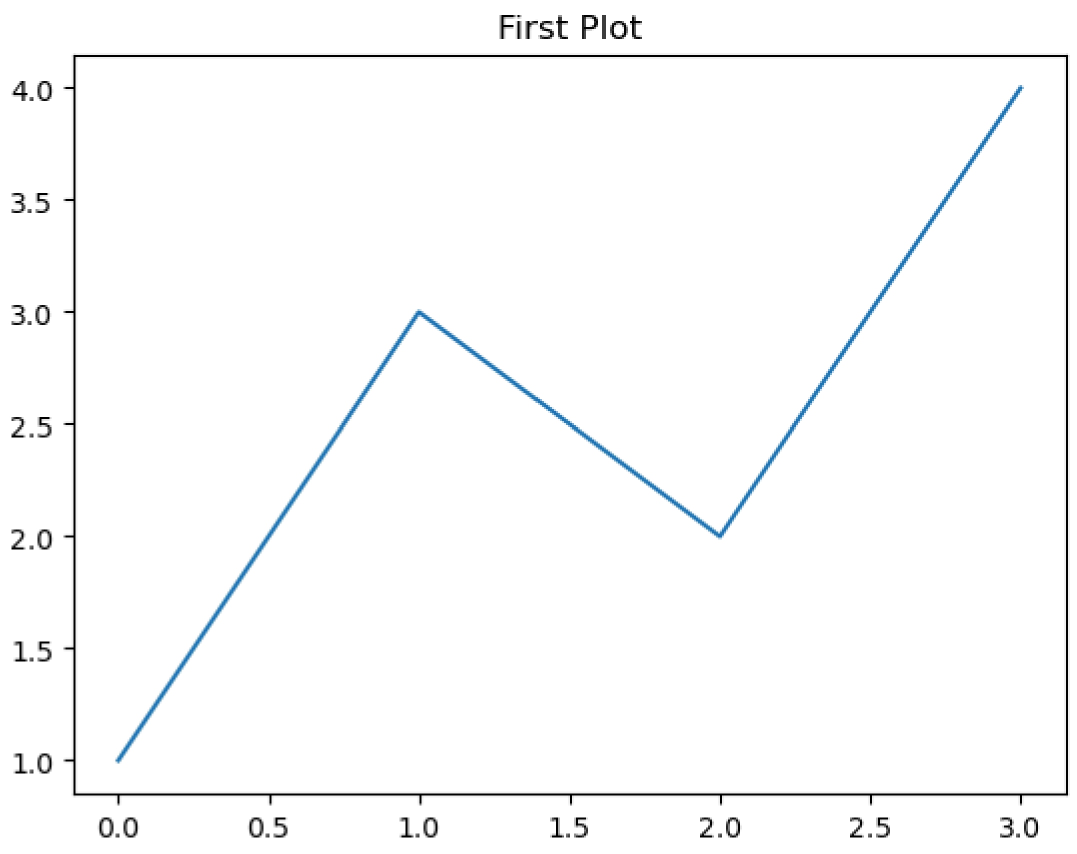
In [48]:
```
plt.plot([1,3,2,4])

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 [49]:
```
plt.plot([1,3,2,4])

plt.title('First Plot')

plt.show()
```



The above plot displays the output of the previous code.The title First Plot is displayed on top of the plot.

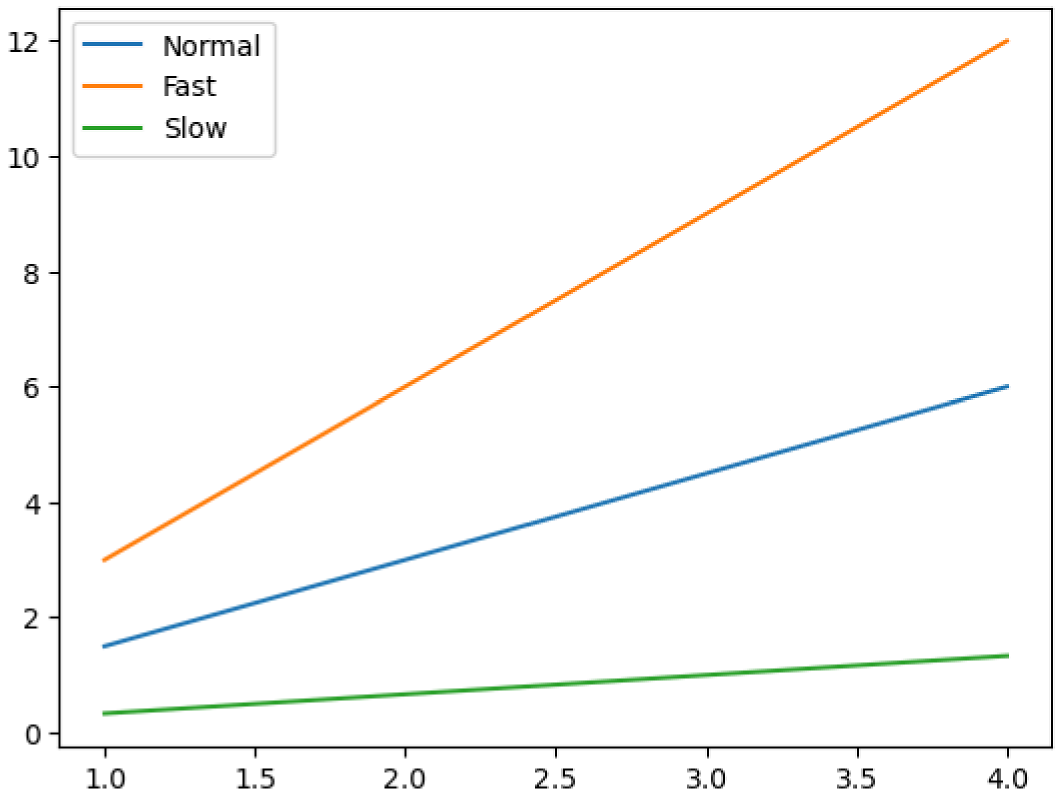
32.Adding a legend

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

legends for curves in 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 legend texts as follows:-

In [50]:
```
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']);
```

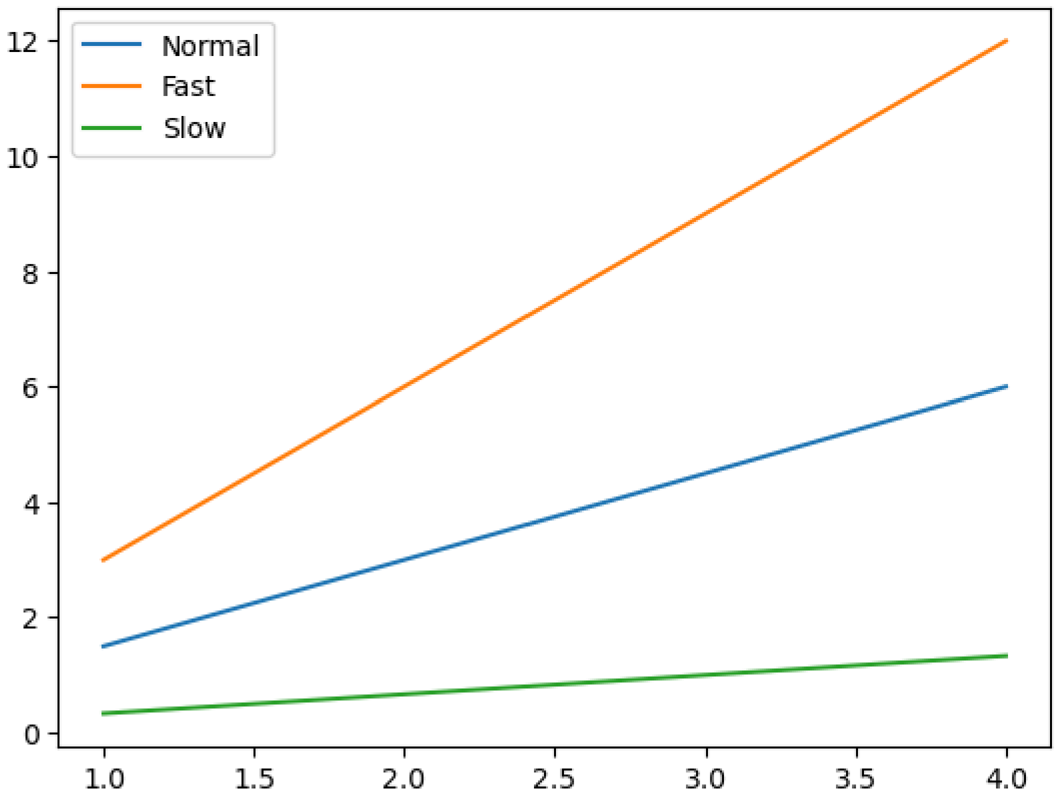
In [51]:
```
plt.show()
```



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 [52]:
```
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();
```

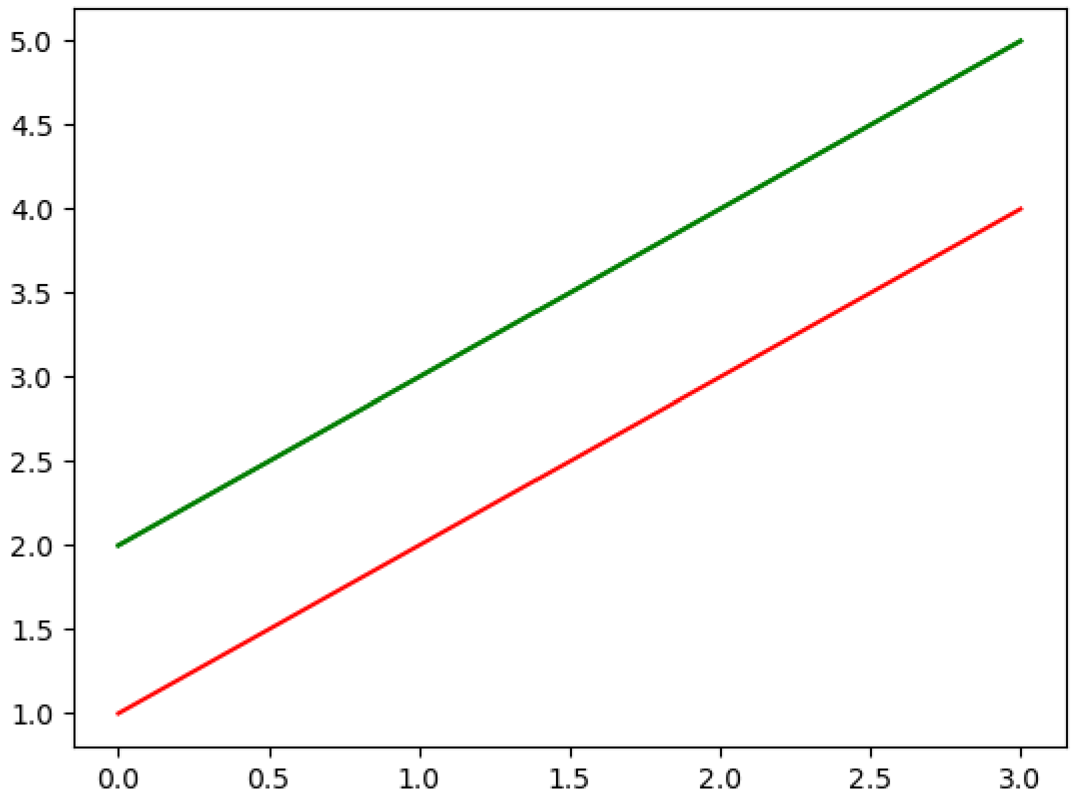
In [53]:
```
plt.show()
```



33. Control colours

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.

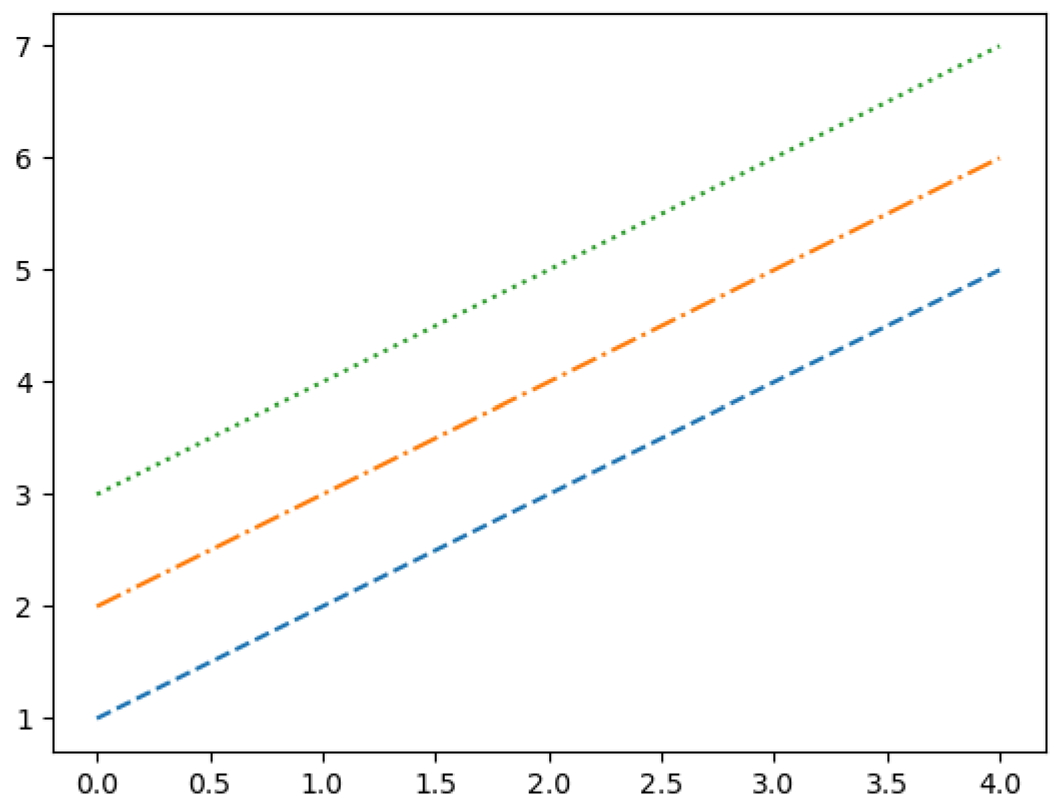
In [54]:
```
x16 = np.arange(1,5)
plt.plot(x16,'r')
plt.plot(x16+1,'g')
plt.plot(x16+1,'g')

plt.show()
```



34. Control line styles

Matplotlib provides us different line style options to draw curves or plots. In the code below, I use different line styles to draw different plots.

In [55]:
```
x16 = np.arange(1,6)
plt.plot(x16,'--',x16+1,'-.',x16+2,':')
plt.show()
```


7
6
5
4
3
2
1
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0