Matplotlib

- 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 tables.

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

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

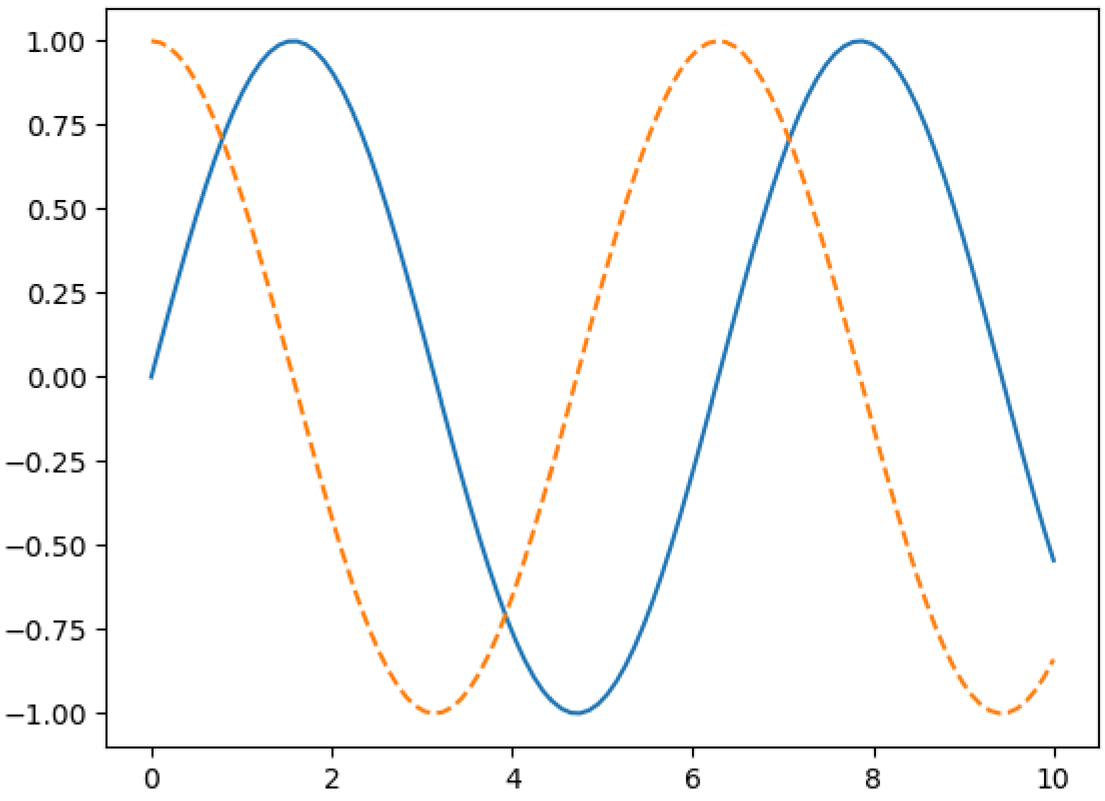
displaying plots in matplotlib

In [ ]:
```
# %matplotlib - magic command
# %matplotlib notebook - this command will produce interactive plots emmbedded w
%matplotlib inline
# it will output static images of plot embedded in the notebook
```

Display plots in matplotlib

In [9]:
```
x1 = np.linspace(0,10,100)
# create a plot figure
fig = plt.figure()

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

Out[9]:
```
[<matplotlib.lines.Line2D at 0x26d9df7a0f0>]
```

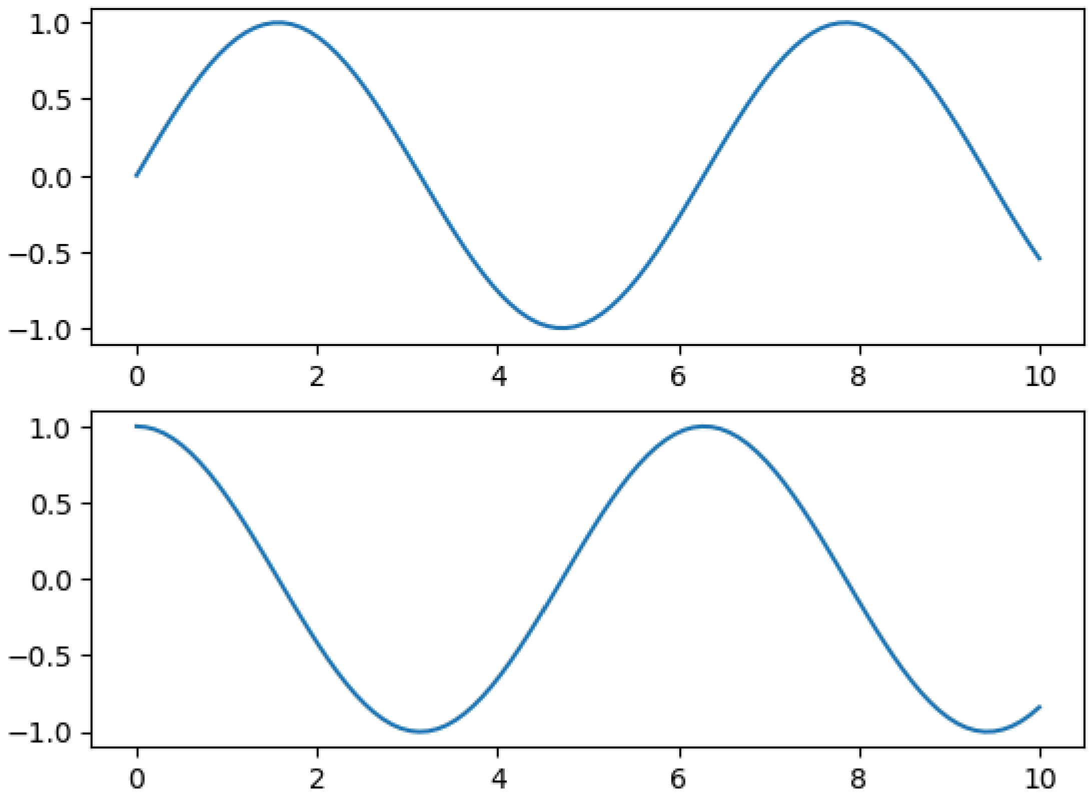


pyplot API

In [ ]:
```
# matplotlib.pyplot is a collection of functions that make matplotlib act like m
```

In [10]:
```
# the following code produces sine and cosine curves using pyplot API
plt.figure()

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

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

Out[10]:
```
[<matplotlib.lines.Line2D at 0x26d9f3b1a60>]
```

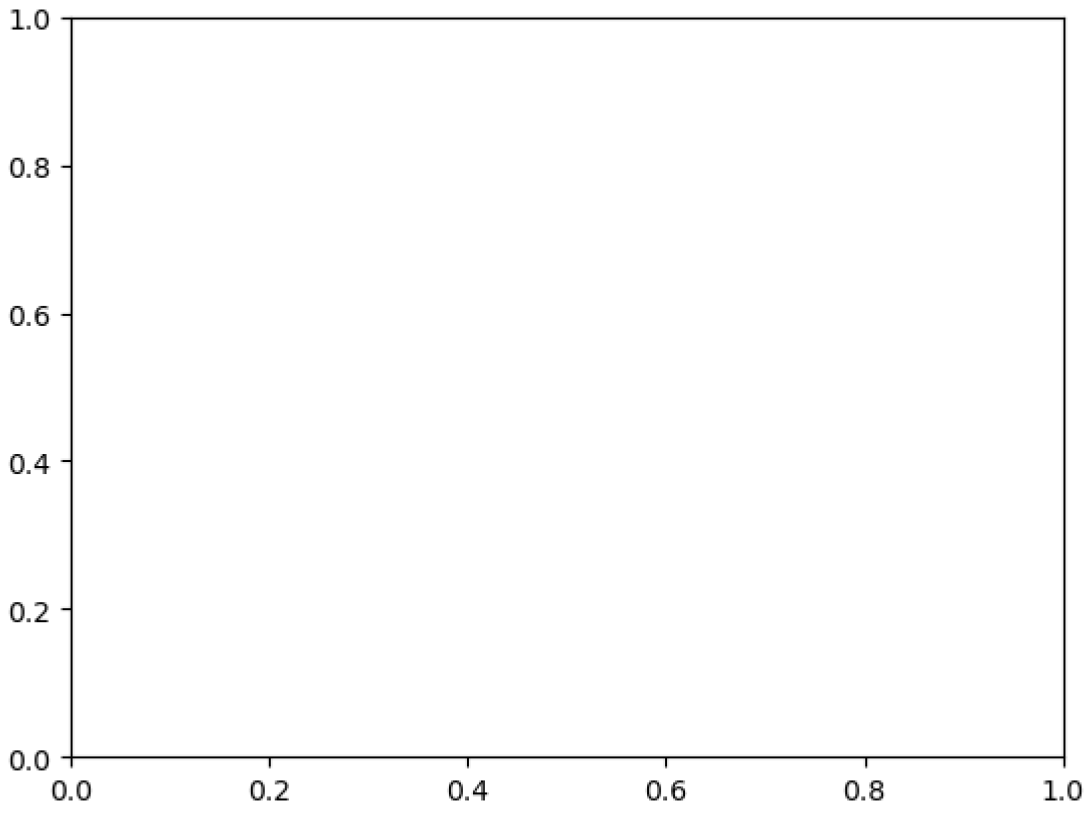


In [11]:
```
# get current figure information
print(plt.gcf()) # gcf=>get current figure
```

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

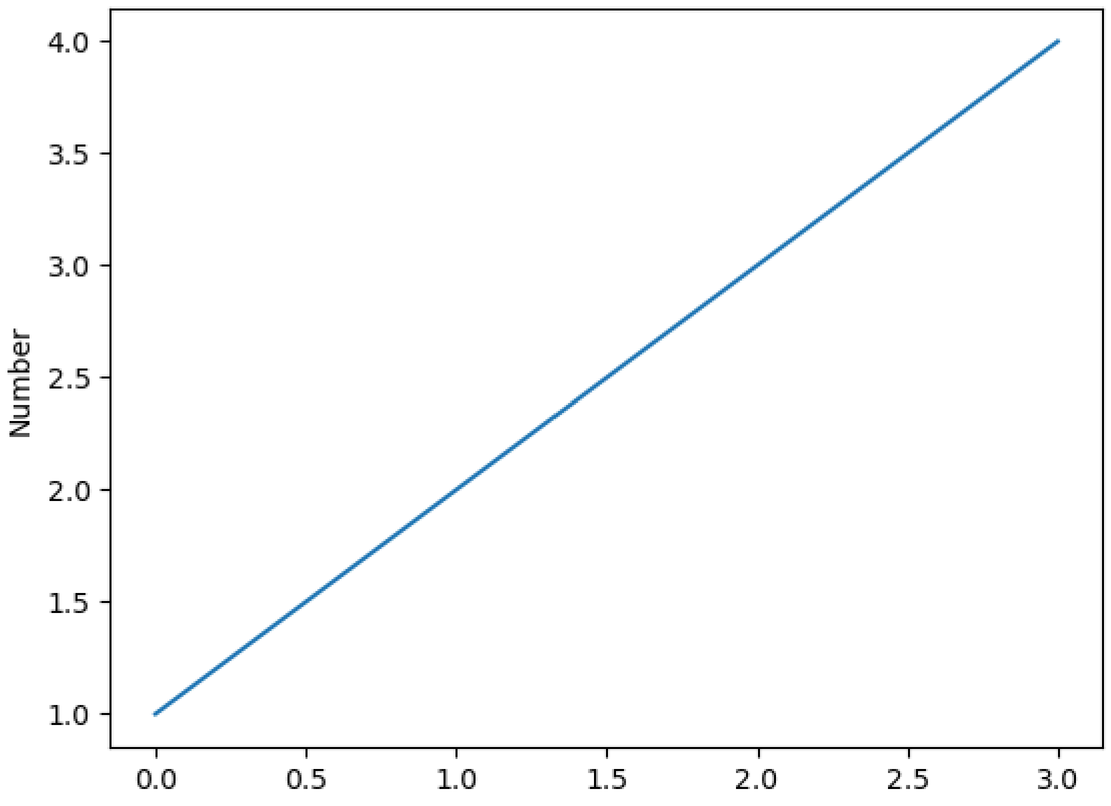
In [12]:
```
# get current axis information
print(plt.gca())# gca=>get current faxis
```

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

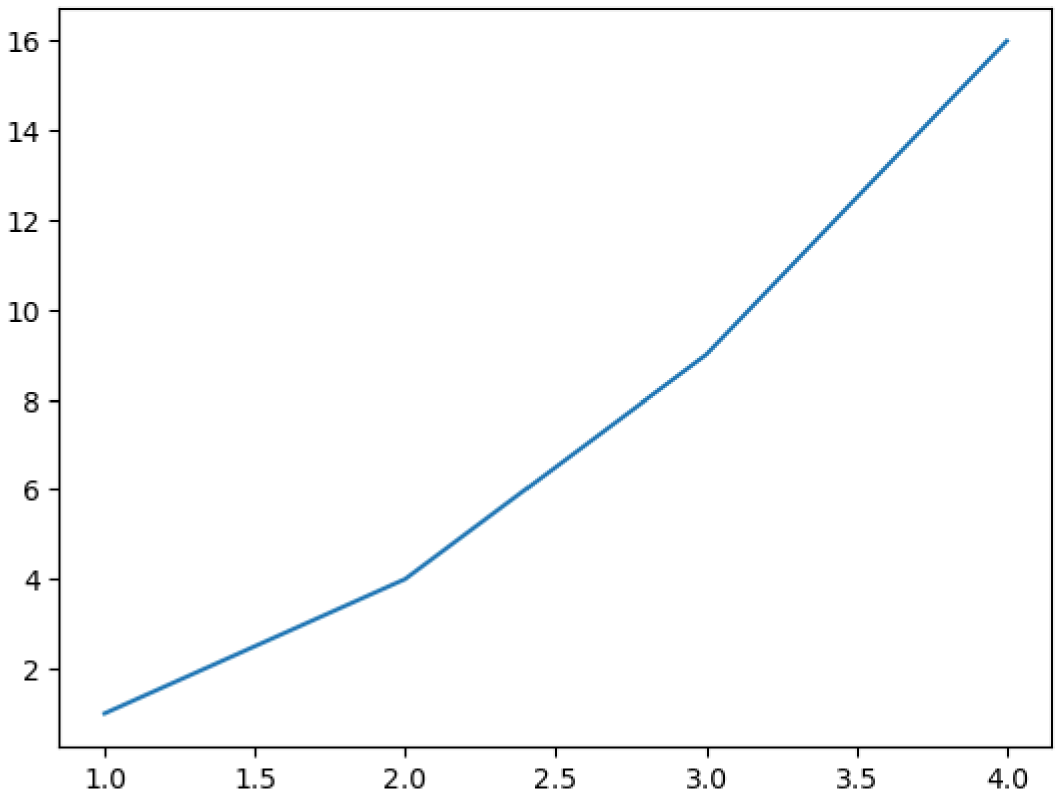


Visualization with pyplot

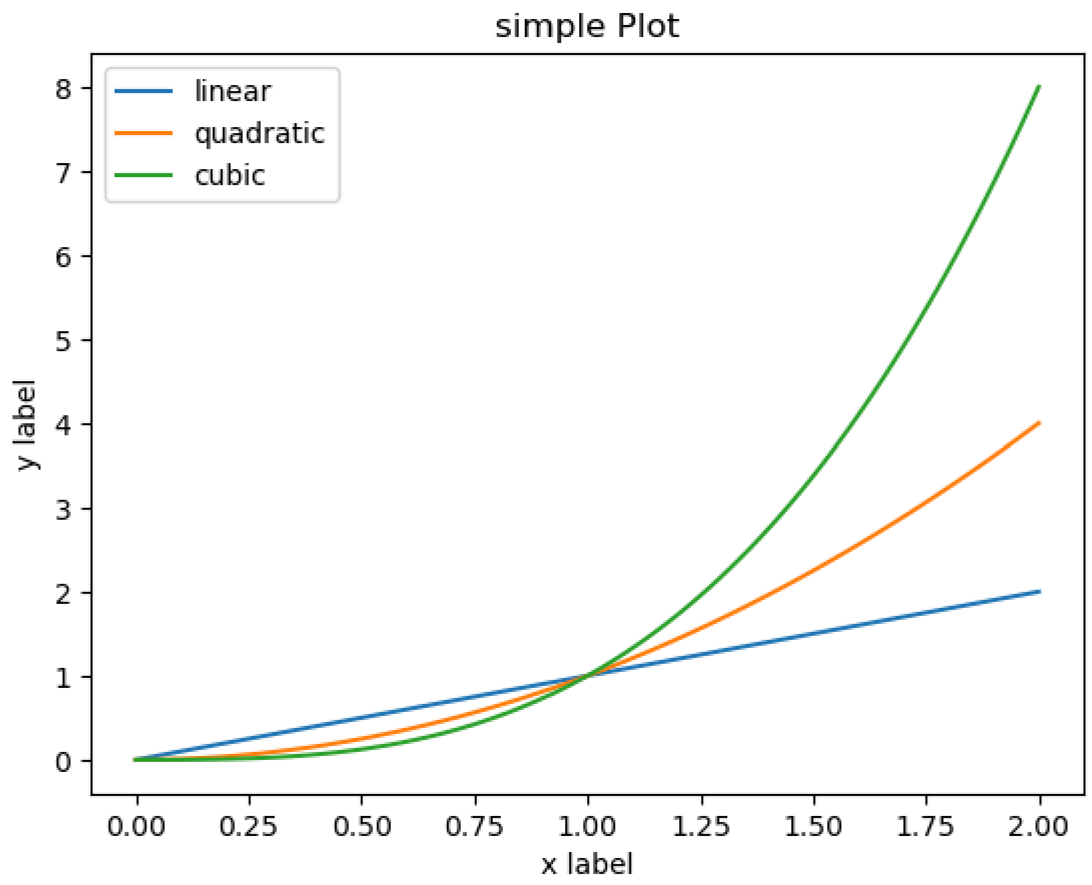
```
In [13]: plt.plot([1,2,3,4])# considers the list as values of y-axis, x-axis values are a
         plt.ylabel('Number')
         plt.show()
```



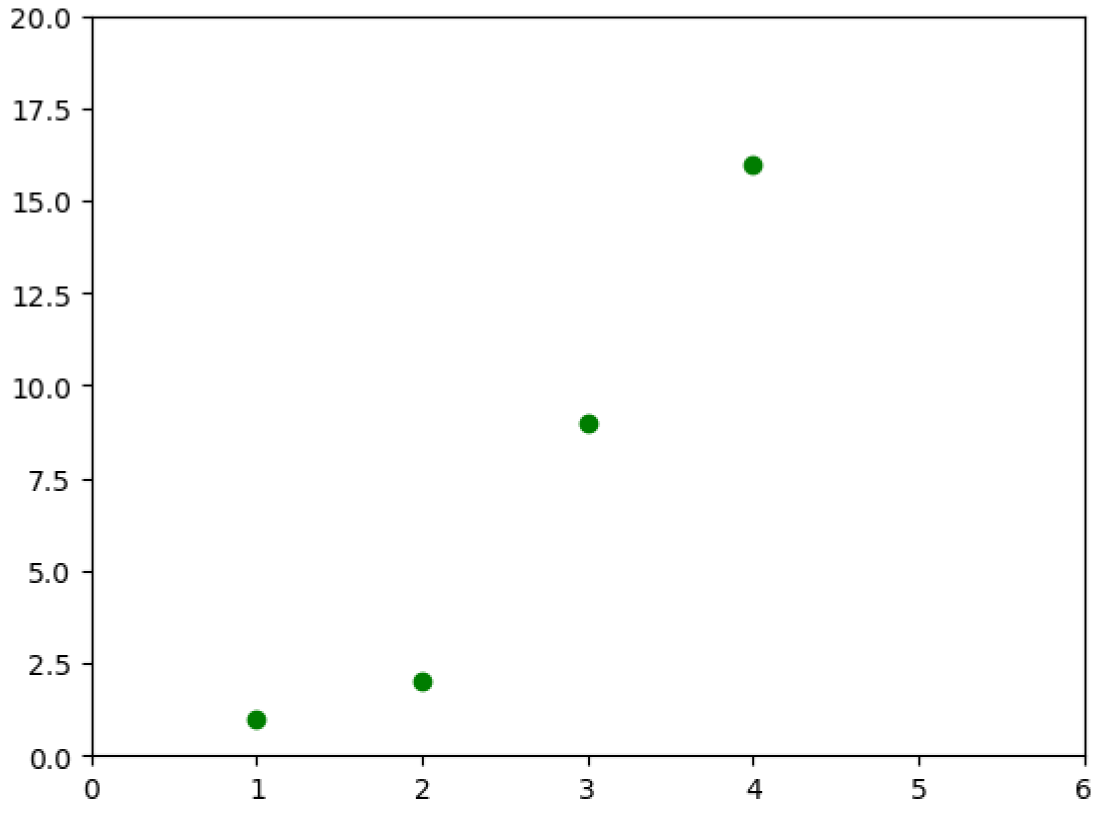
```
In [14]: # plot()- it is a versatile command, takes arbitary number of arguments, to plot
         plt.plot([1,2,3,4],[1,4,9,16])

Out[14]: [<matplotlib.lines.Line2D at 0x26d9f62ae70>]
```



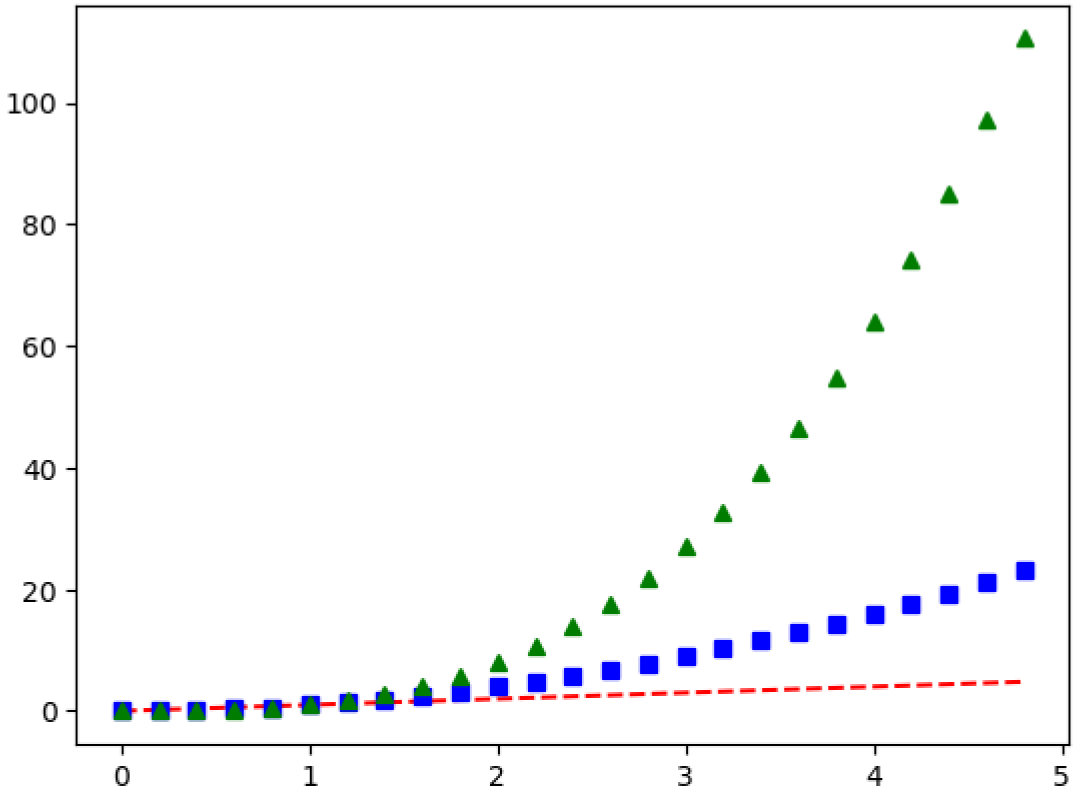
In [16]:
```
# pyplot provides state machine interface
# state machine implicitly and automatically creates figures and axes to achieve

# Example
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()
```



In [18]:
```
# formatting the style of plot
# there is a optional argument format string, indicates color and line type of p

plt.plot([1,2,3,4],[1,2,9,16],'go')
plt.axis([0,6,0,20]) # 0-xmin,6-xmax,0-ymin,20-ymax # axis command specifies vie
plt.show()
```

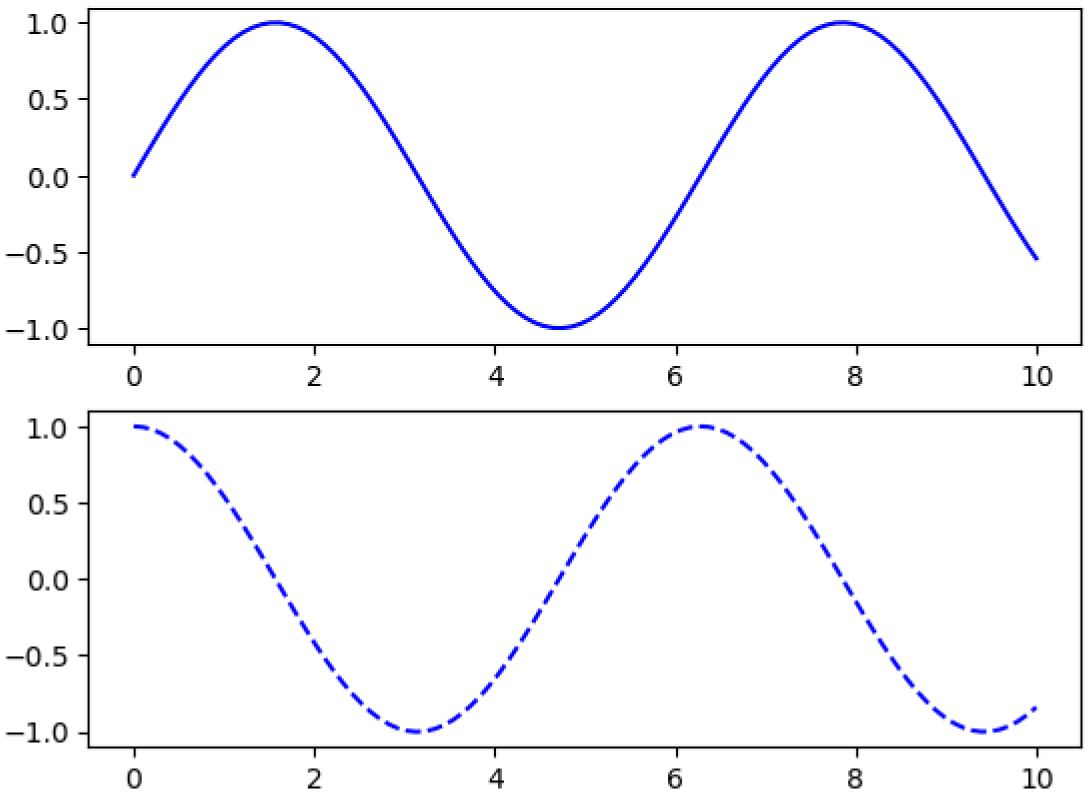


In [20]:
```
# Working with numpy arrays
# sequences converted into numpy arrays
# below illustrates ploting several lines with different format style in onr com

# evenly sampled time at 200 intervals
t = np.arange(0.,5.,0.2)

# red dashes, blue squares, green triangles
plt.plot(t,t,'r--',t,t**2,'bs',t,t**3,'g^')
plt.show()
```

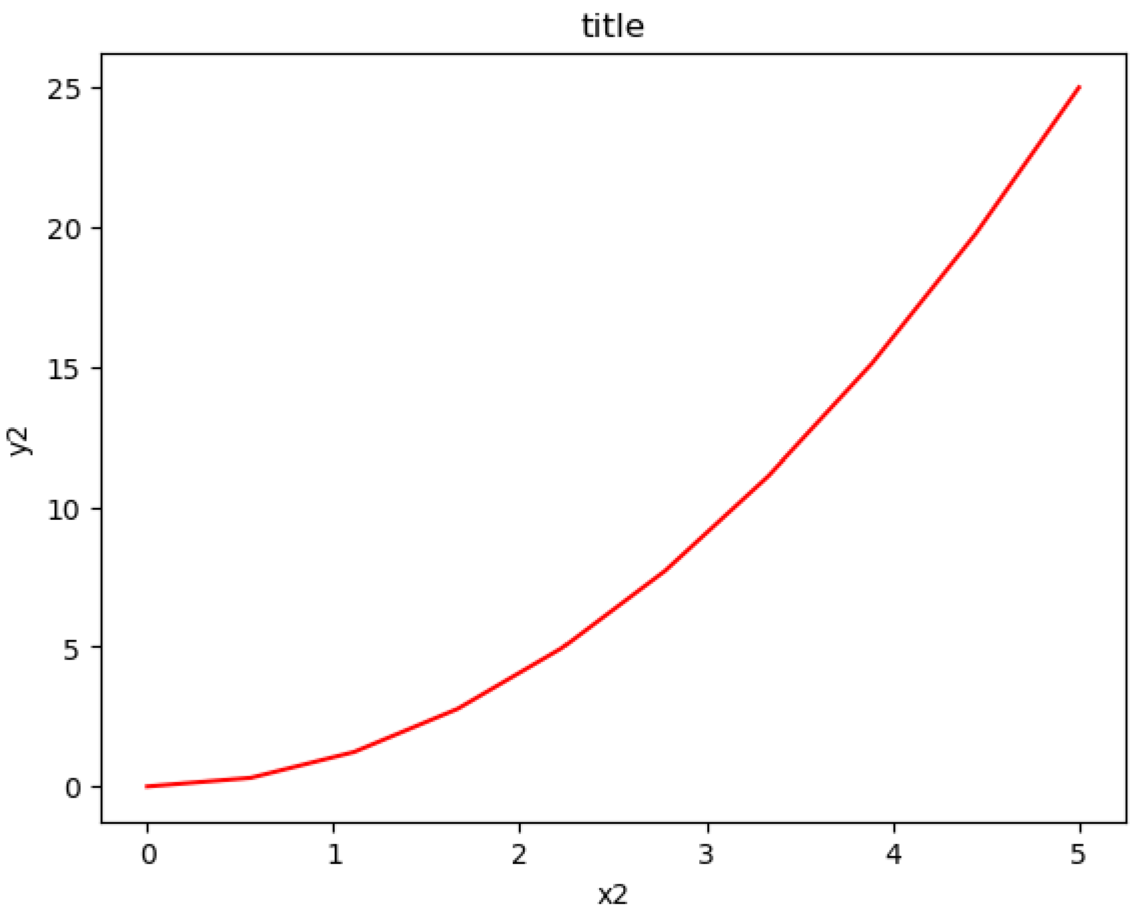


In [21]:
```
#For complex , exercise more control over figures
# figure is top level container for all plot elements- contains one or more axis
# axis - represents each individual plot

# first create a grid of plots
# ax- array of two axis 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--')
```

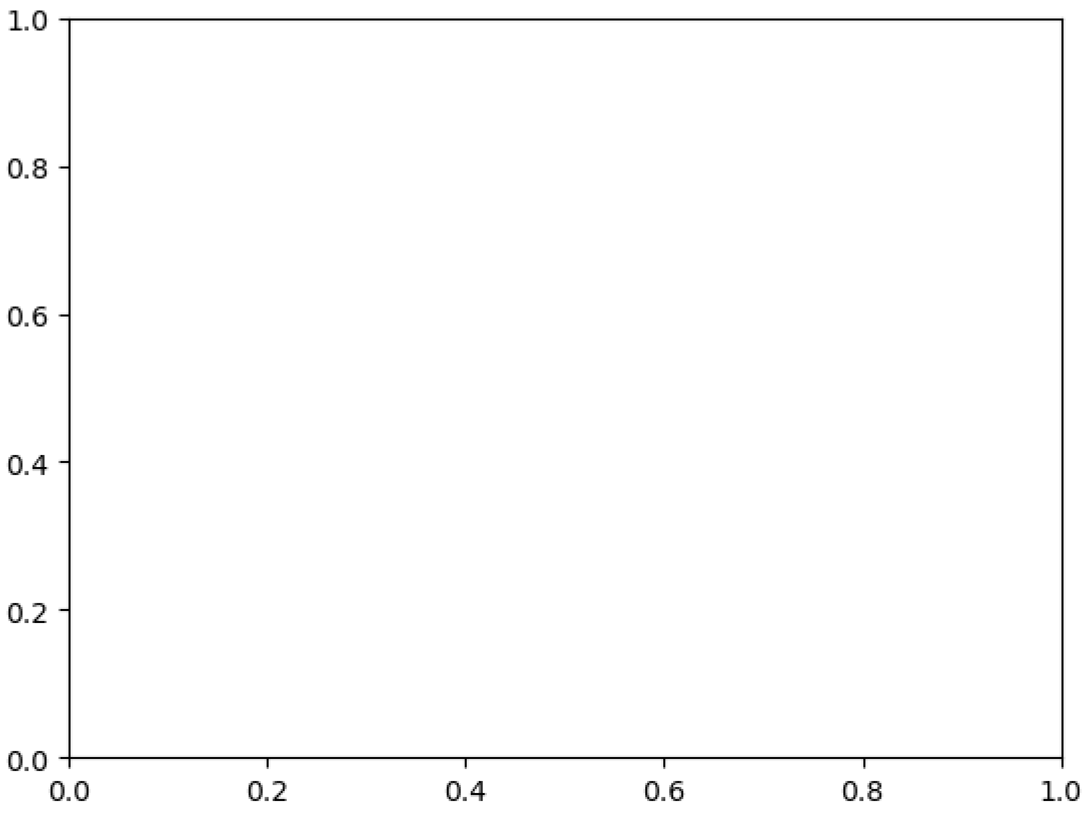
Out[21]:
```
[<matplotlib.lines.Line2D at 0x26d9f7717f0>]
```



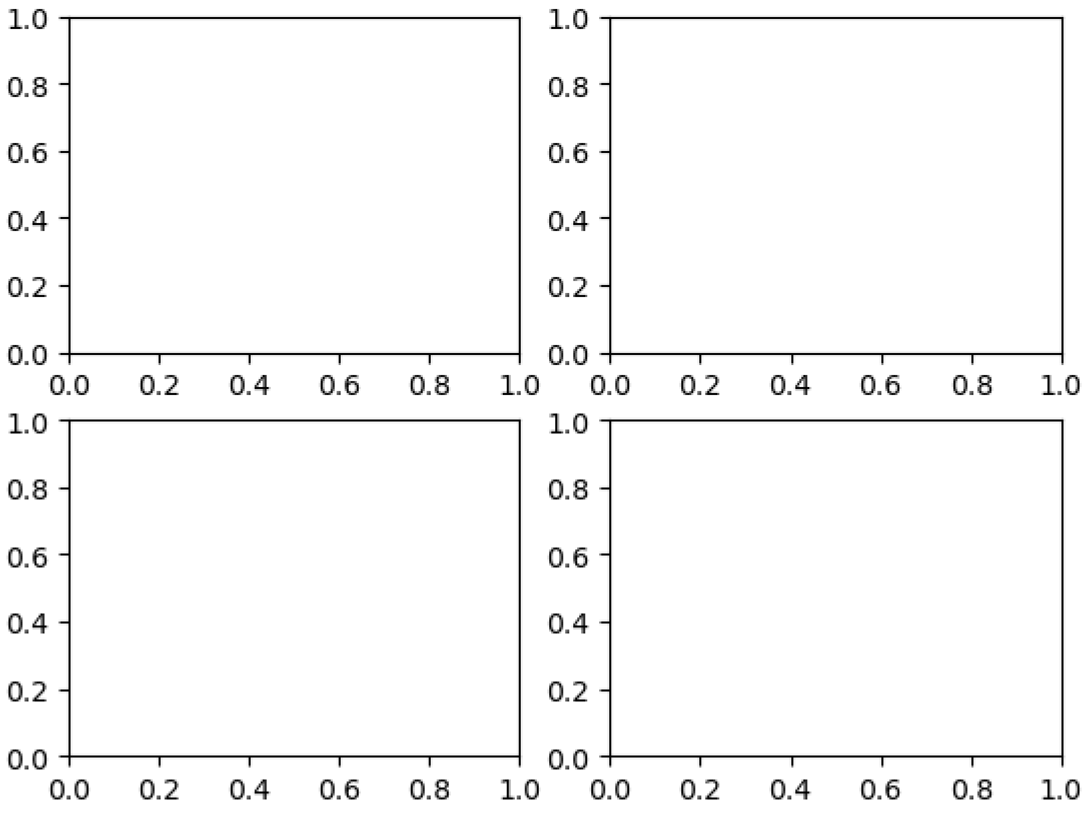
```
In [22]: # objective of OO-API is to have objects to apply functions and actions on, more

         fig=plt.figure() # creates reference to the figure instance

         x2=np.linspace(0,5,10)
         y2=x2**2

         axes=fig.add_axes([0.1,0.1,0.8,0.8]) # create new axis istance using add_axes me

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

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

Out[22]: Text(0.5, 1.0, 'title')
```



In [25]:
```
# creating figure and axis

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

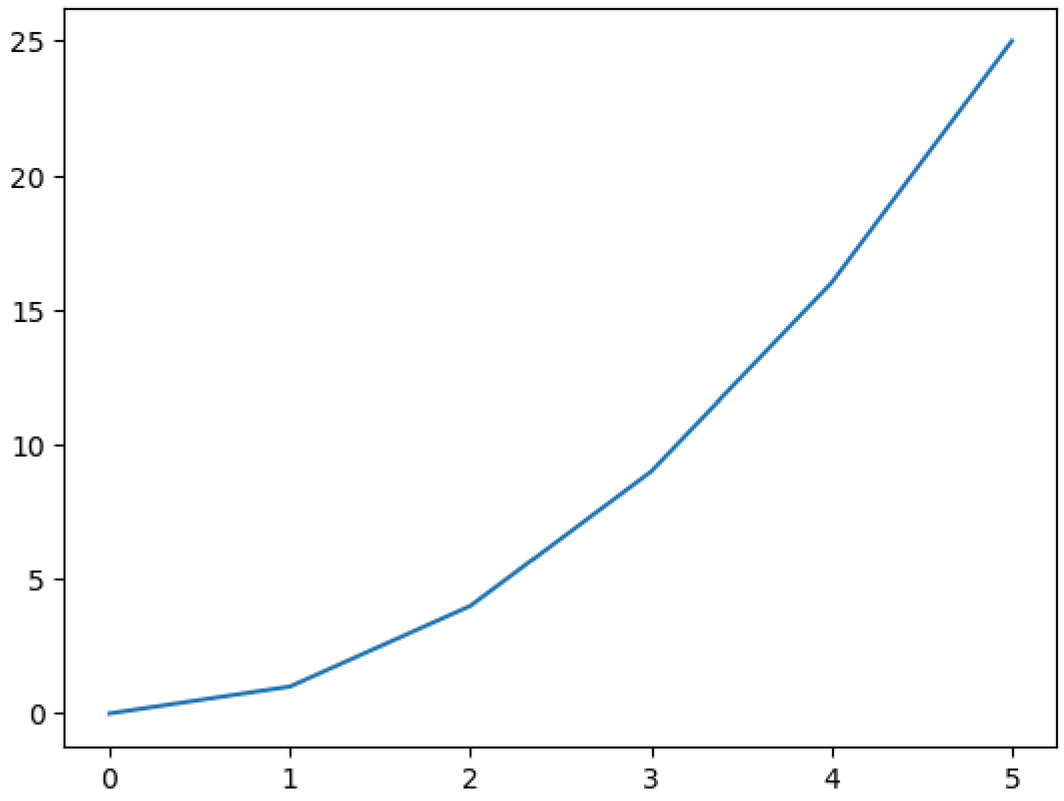


In [26]:
```
# Figure and subplots
# plots in mlt reside within a fig obj
fig=plt.figure()

#now create subplot using gig.add_subplot() method
ax1=fig.add_subplot(2,2,1) # here the in first two arguments 2=> no.of rows , 2=
ax2=fig.add_subplot(2,2,2) # i.e., we can create only four subplots
ax3=fig.add_subplot(2,2,3)
ax4=fig.add_subplot(2,2,4) # added four subplots
```

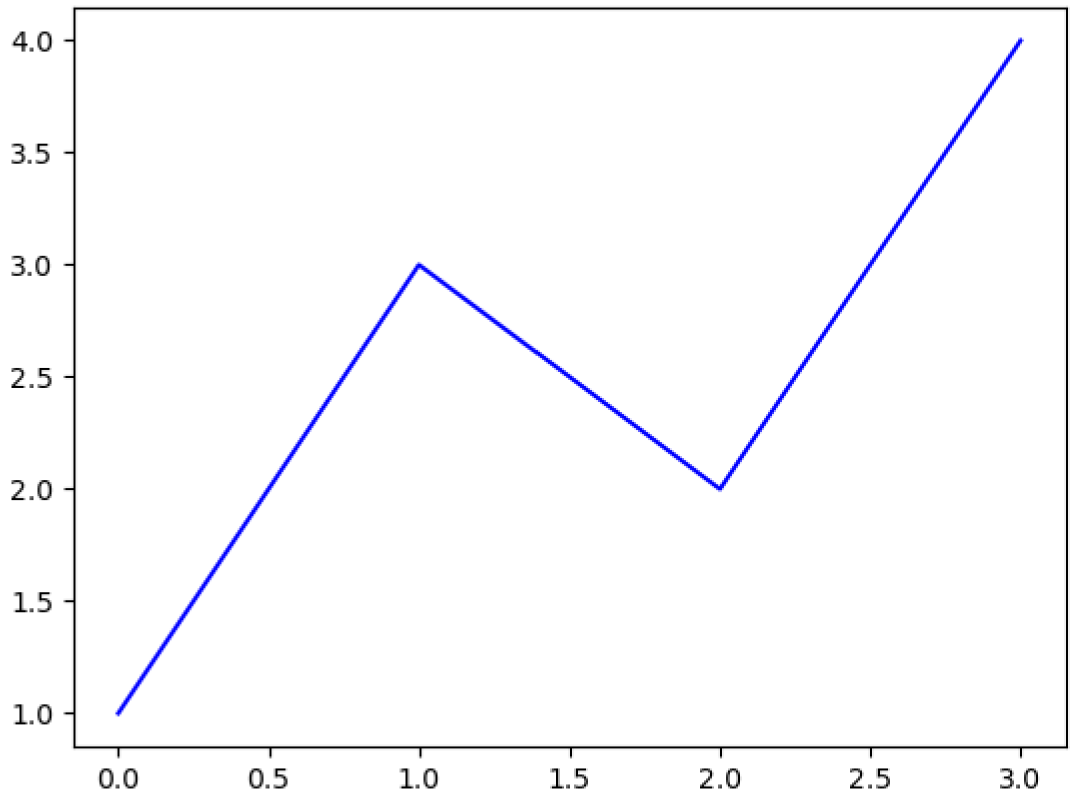


In [ ]:
```
# The difference in plots ways as list and array
```

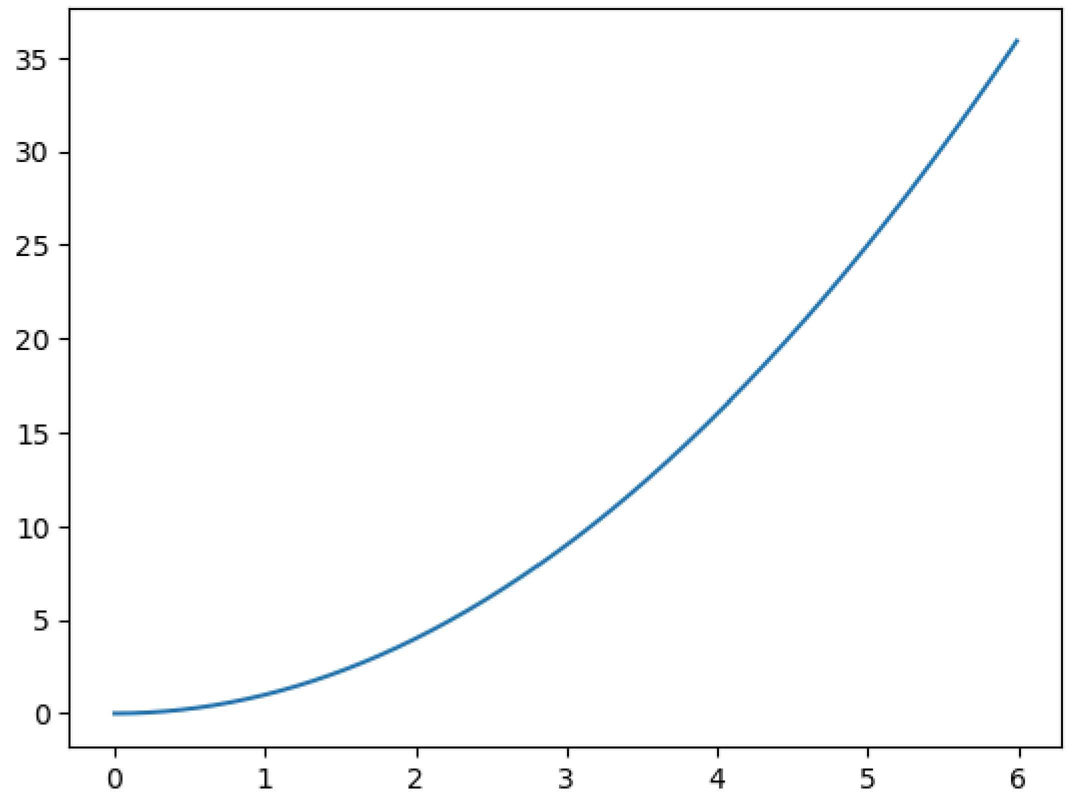
In [27]:
```
x3=range(6)
plt.plot(x3,[xi**2 for xi in x3])
plt.show()
```



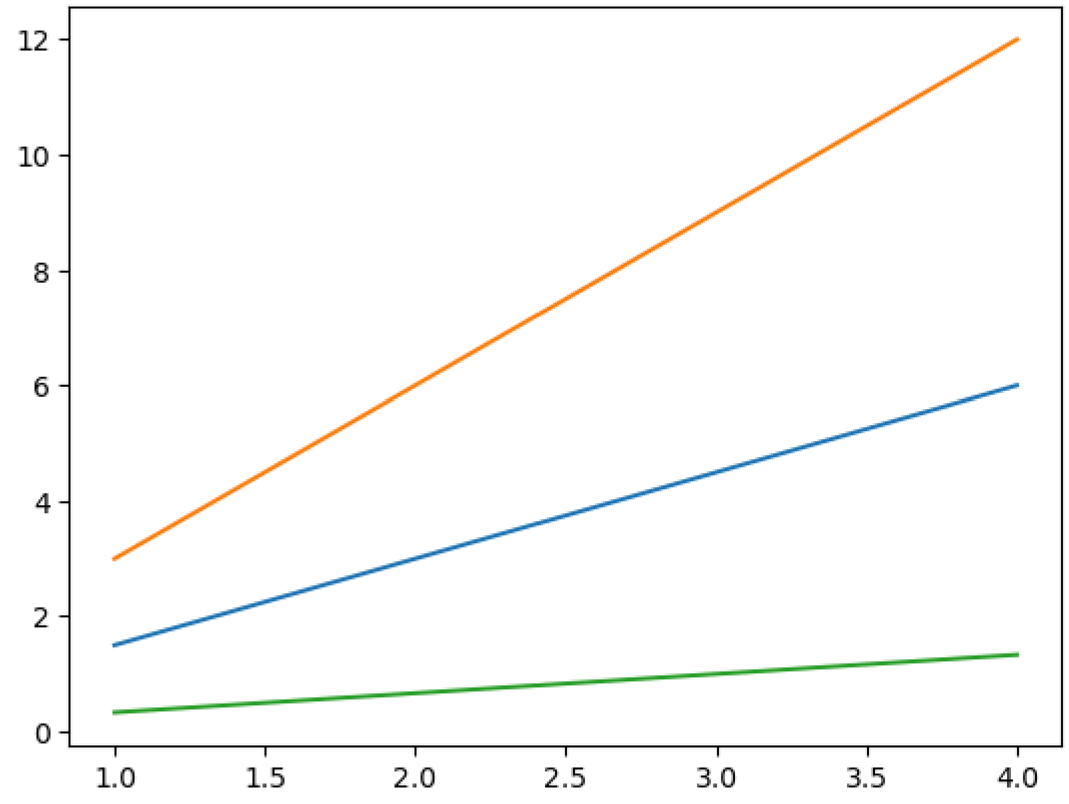
In [28]:
```
# first plot with matplotlib
plt.plot([1,3,2,4],'b-')
plt.show()
```



In [29]:
```
x3=np.arange(0.,6.,0.01)
plt.plot(x3,[xi**2 for xi in x3])
plt.show()
```



In [30]:
```
#multiline plots
# calling more than one plot in same figure, can be achieved by calling show() m
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()
```



Saving the plot

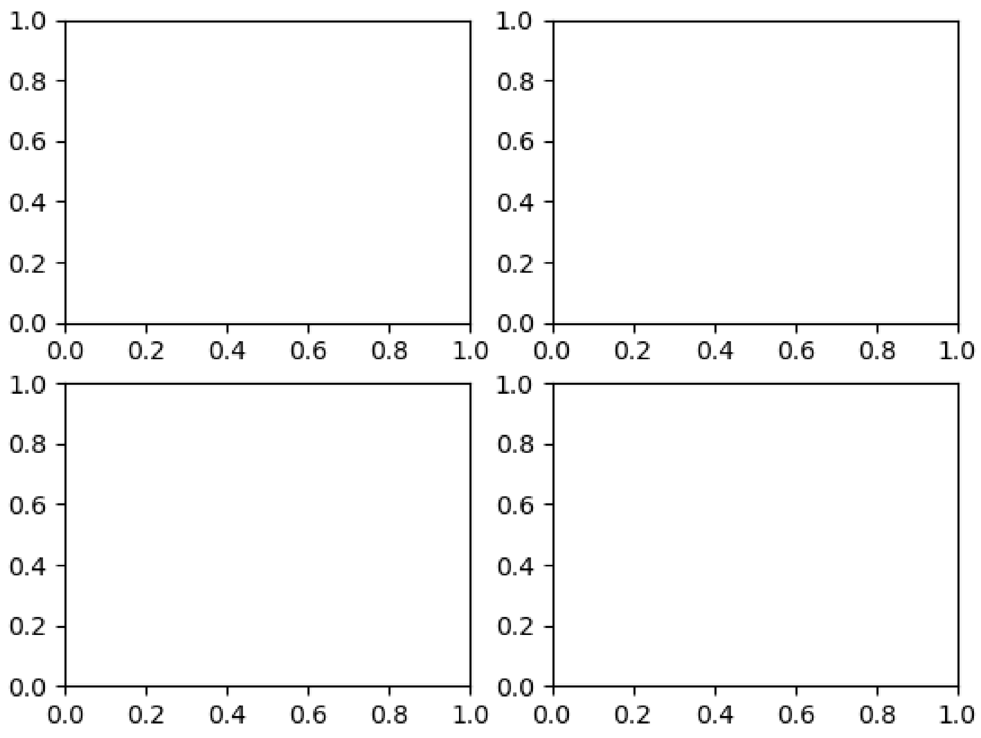
In [31]:
```
# the list of file types in which we can save the plots can be known using the f
fig.canvas.get_supported_filetypes()
```

Out[31]:
```
{'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'}
```

In [32]:
```
# saving a figure
fig.savefig('plot1.png')
```

In [33]:
```
# explore contents of figure
from IPython.display import Image
Image('plot1.png')
```

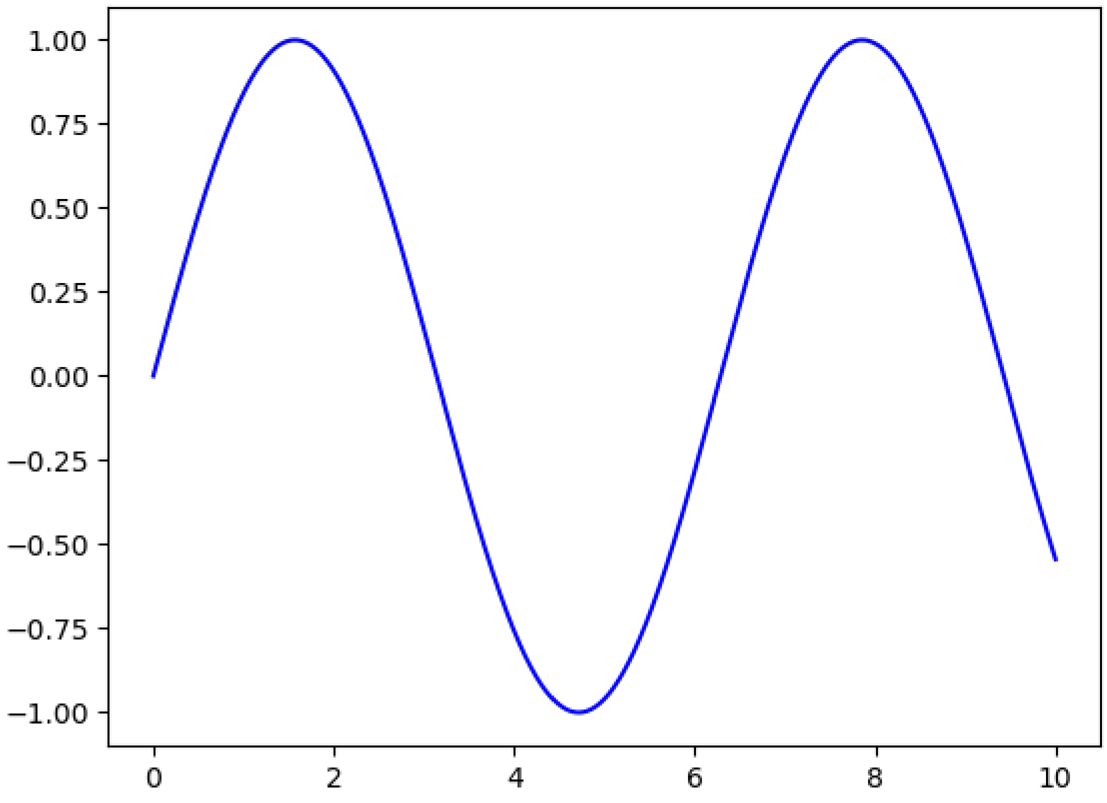
Out[33]:



Line Plot

In [34]:
```
# create a 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-')
```

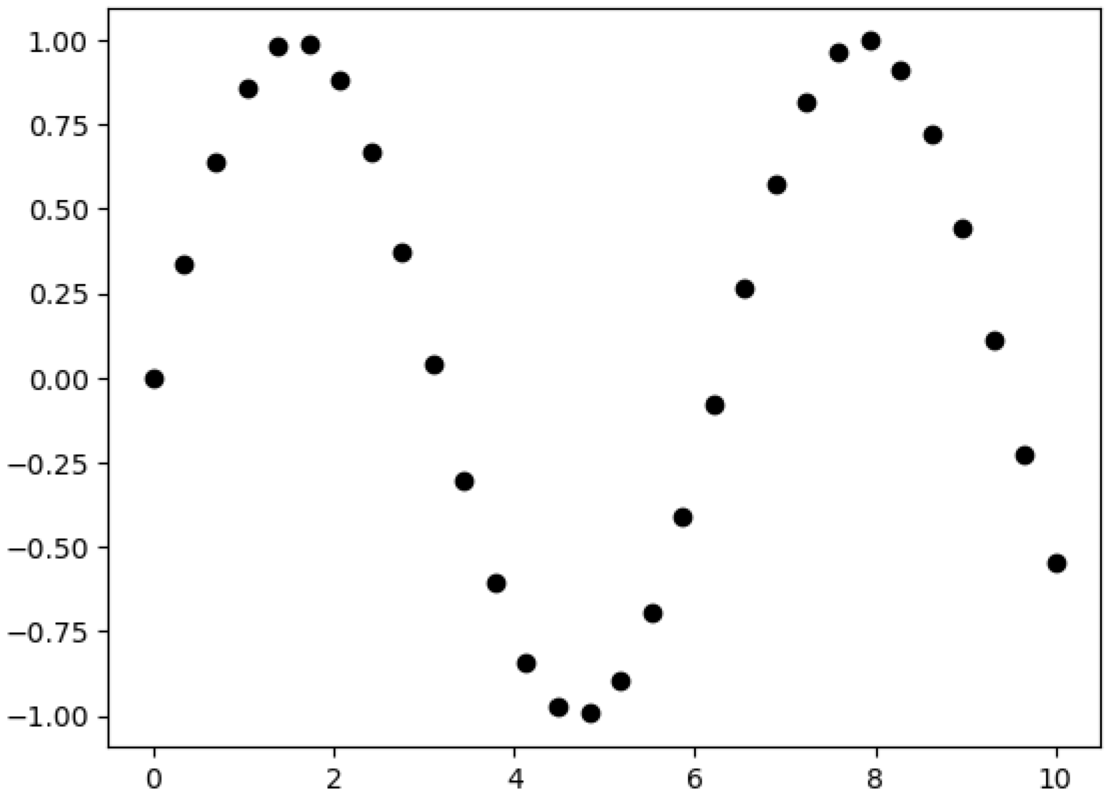
Out[34]:
```
[<matplotlib.lines.Line2D at 0x26d9ec3e630>]
```



Scatter plot

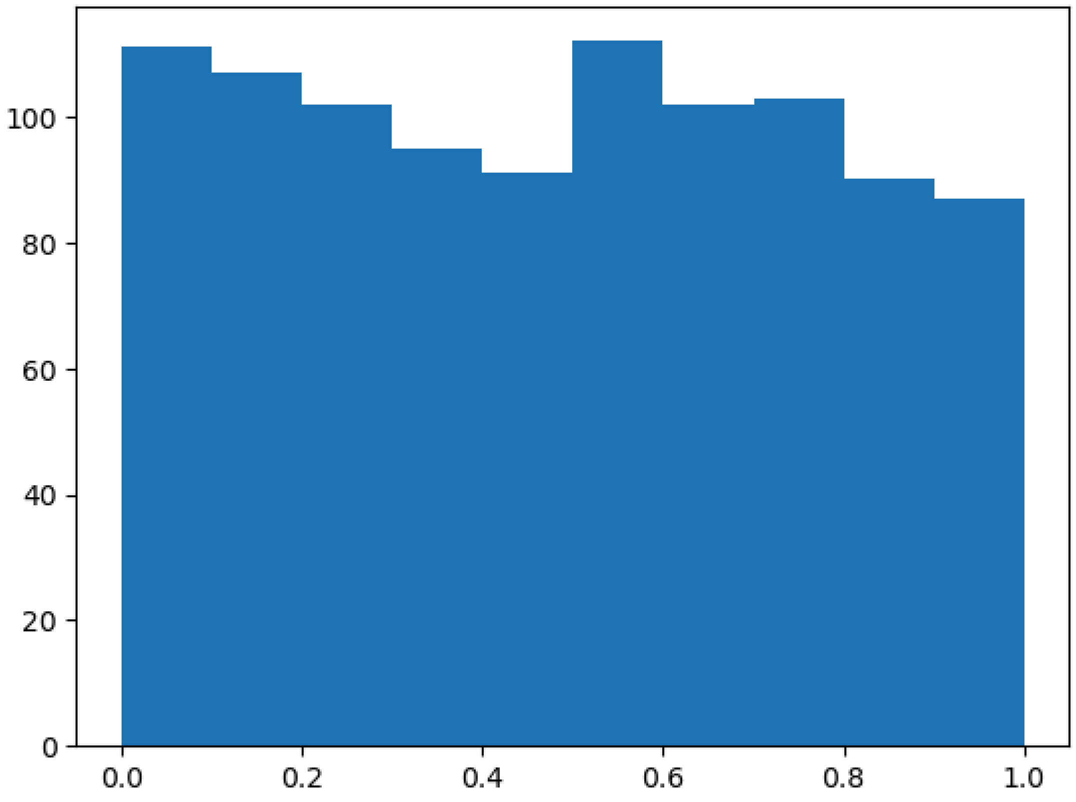
In [36]:
```
# make use of same functions as line plot

x7=np.linspace(0,10,30) # 0=> start, 10=> stop, 30=> no.of numbers to be generat
y7=np.sin(x7)
plt.plot(x7,y7,'o',color='k')
```

Out[36]:
```
[<matplotlib.lines.Line2D at 0x26da0b1c440>]
```



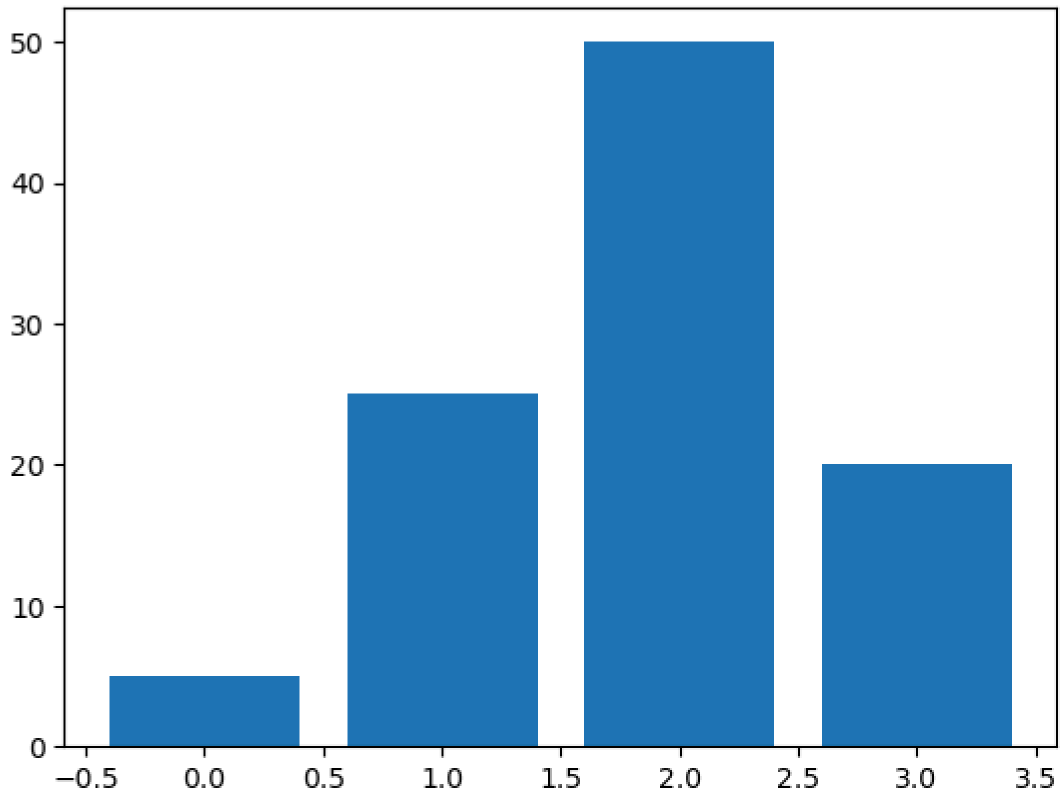
Histogram

```
In [37]: # plt.hist() - used to plot simple histogram
         data1=np.random.rand(1000)
         plt.hist(data1);
```



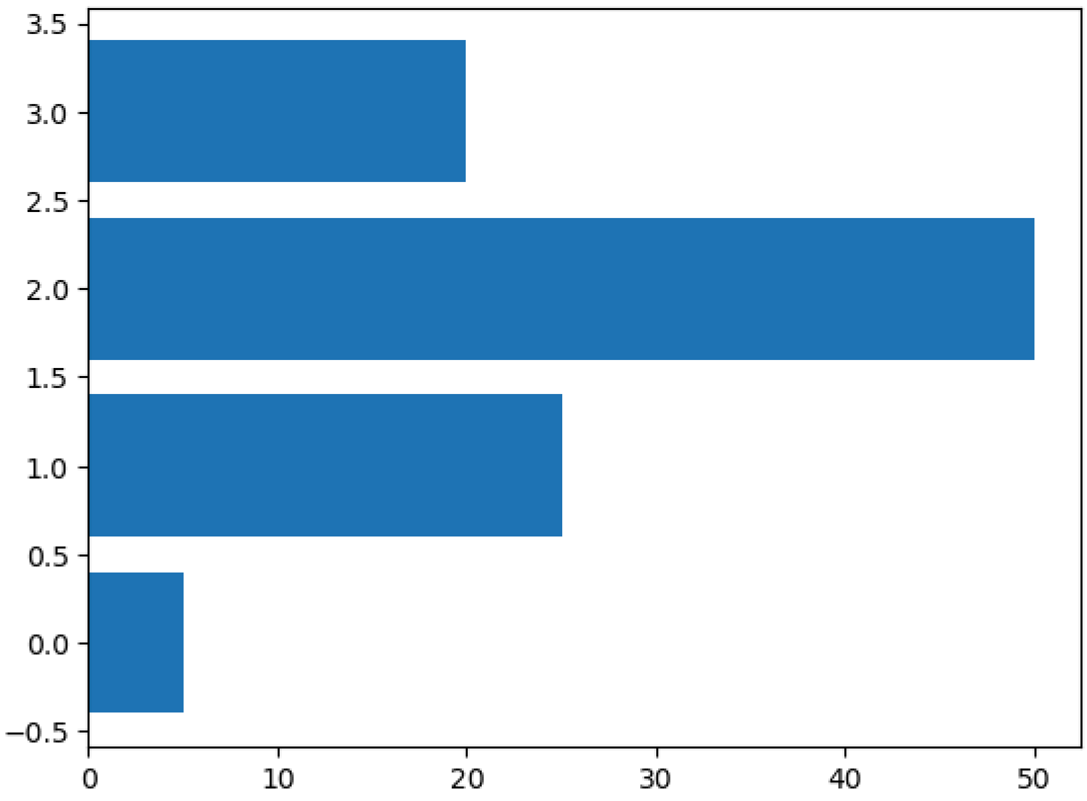
Bar Chart

In [38]:
```
# used to compare two or more values
# plot bar chart using plt.bar

data2=[5.,25.,50.,20.]
plt.bar(range(len(data2)),data2)
plt.show()
```



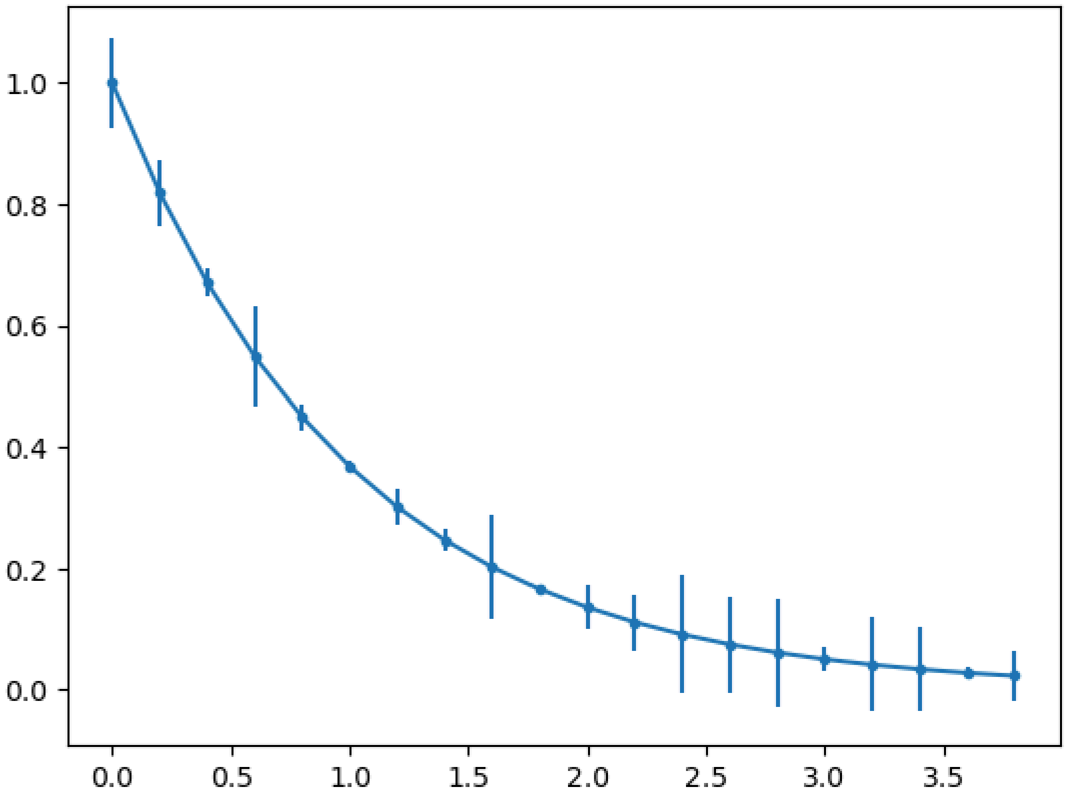
Horizontal Bar Chart

In [39]:
```
#plt.barh() - used to plot horizontal bar chart

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



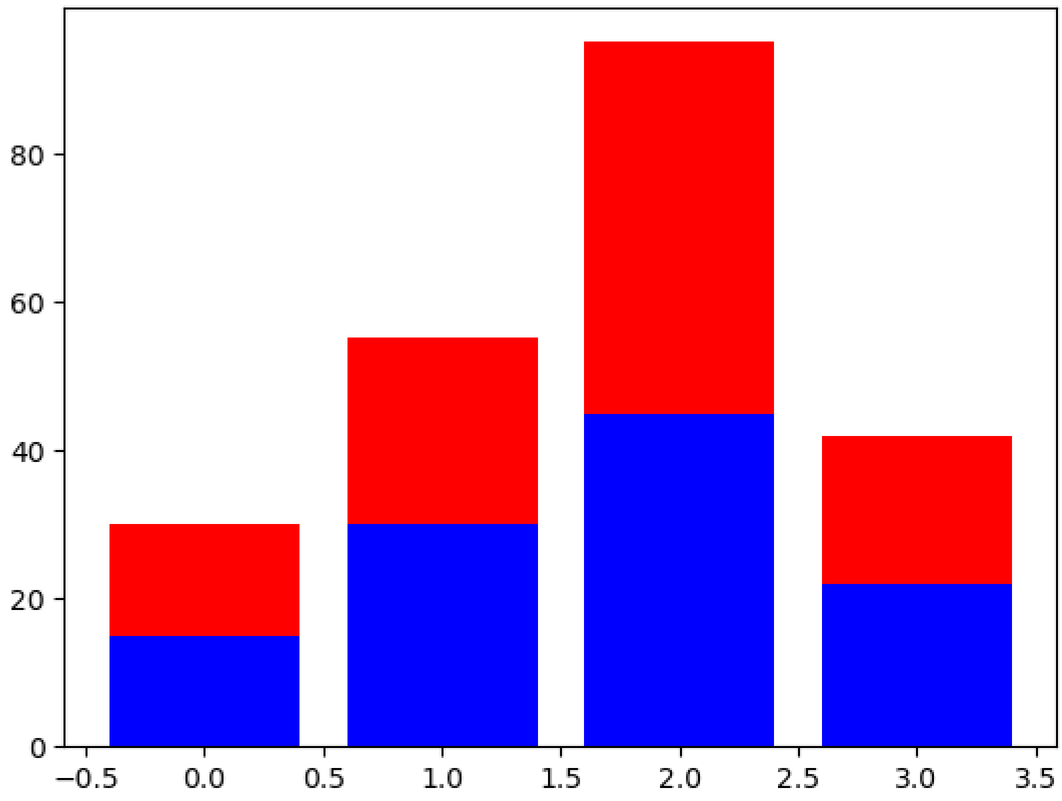
Error bar Chart

In [40]:
```
x9=np.arange(0,4,0.2)
y9=np.exp(-x9)
e1=0.1*np.abs(np.random.rand(len(y9)))
plt.errorbar(x9,y9,yerr=e1,fmt='.-') # error distribution
plt.show();
```



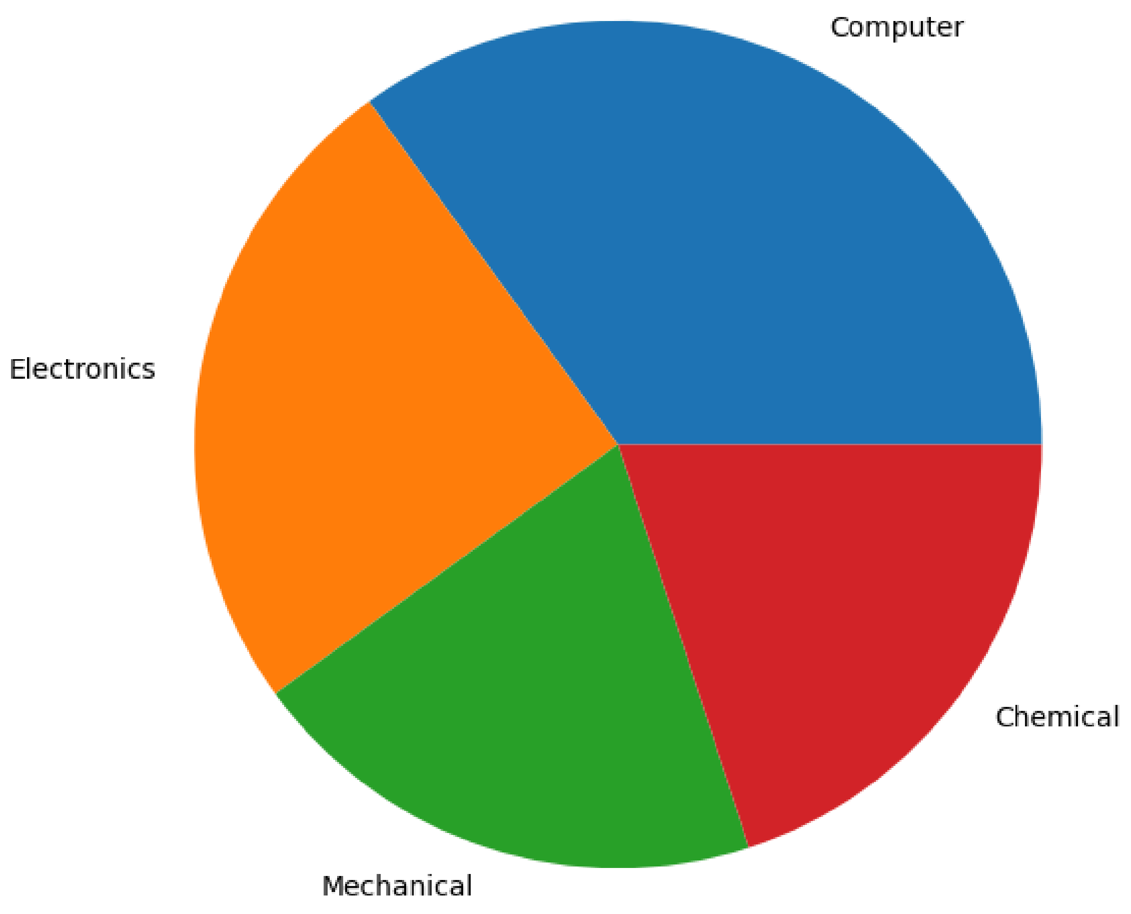
Statcked bar Chart

In [41]:
```
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) # bottom important for staced plotting
plt.show()
```



Pie Chart

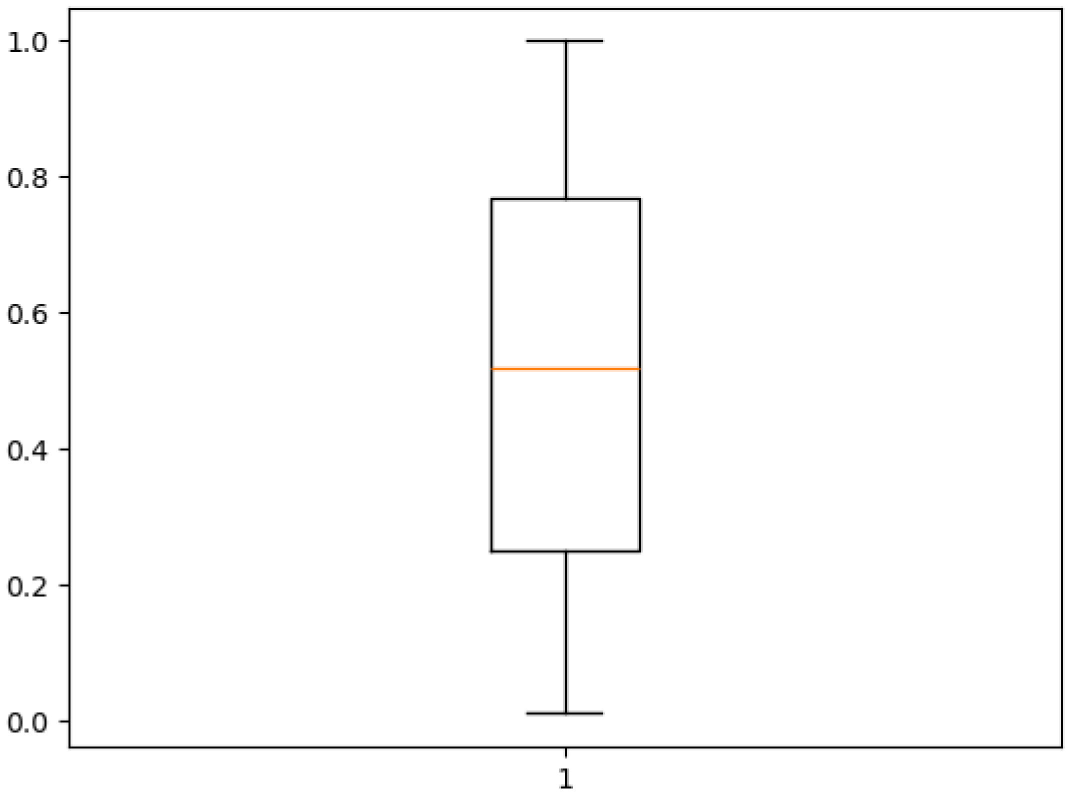
In [43]:
```
# pie function used to plot pie from an array x, wedges are created proprtionall
plt.figure(figsize=(7,7))
x10=[35,25,20,20]
labels=['Computer','Electronics','Mechanical','Chemical']
plt.pie(x10,labels=labels);
plt.show()
```



Box Plot

```
In [ ]: Allows us to compare distribution of values by showing quartiles,median,max,min
```

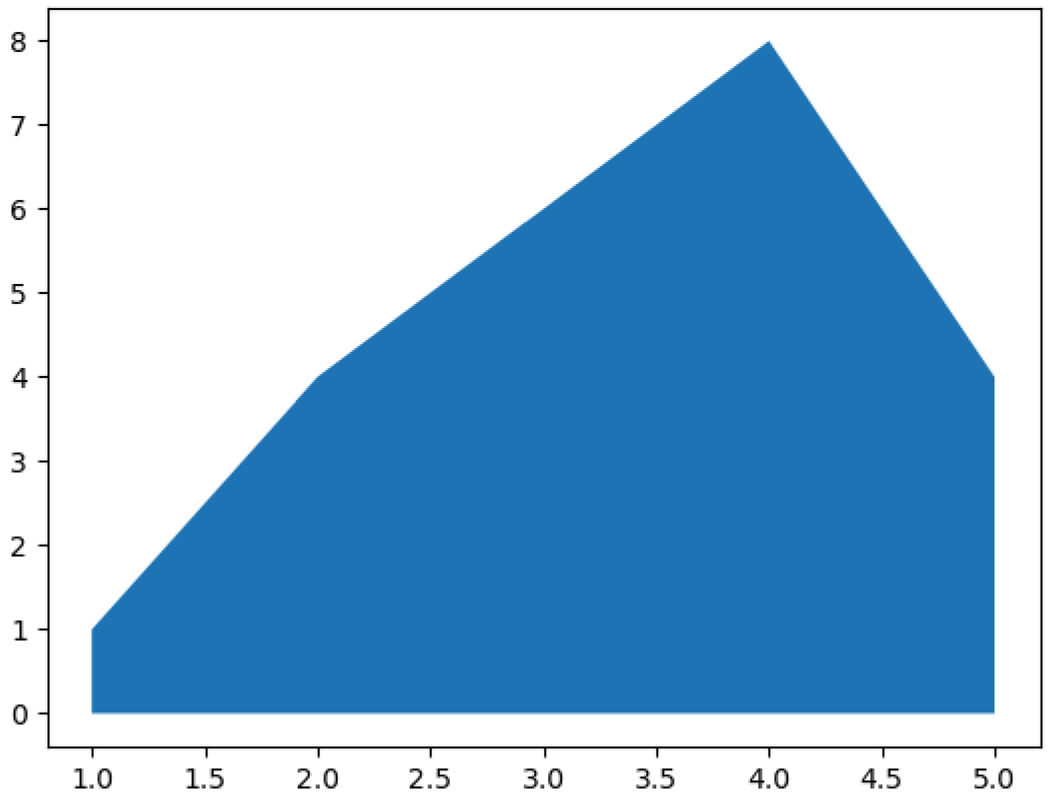
```
In [44]: data3=np.random.rand(100)
         plt.boxplot(data3)
         plt.show();
```



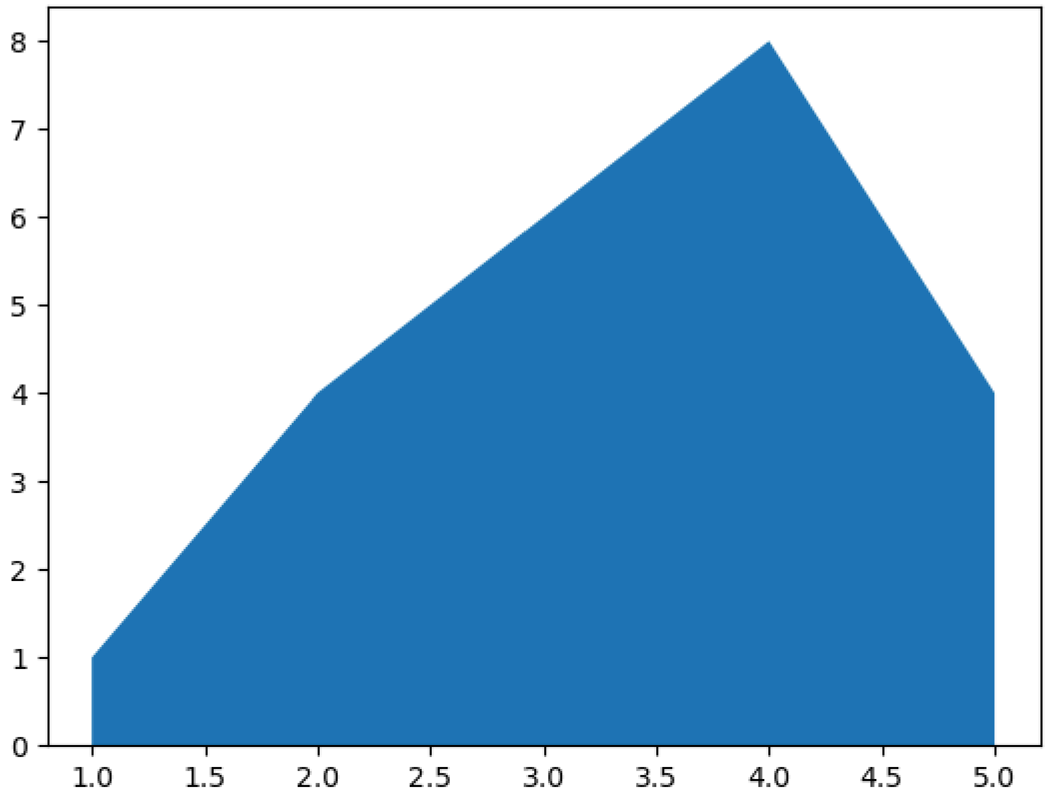
Area Chart

Similar to Line Chart

```
In [45]: x12=range(1,6)
         y12=[1,4,6,8,4] # data creation

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



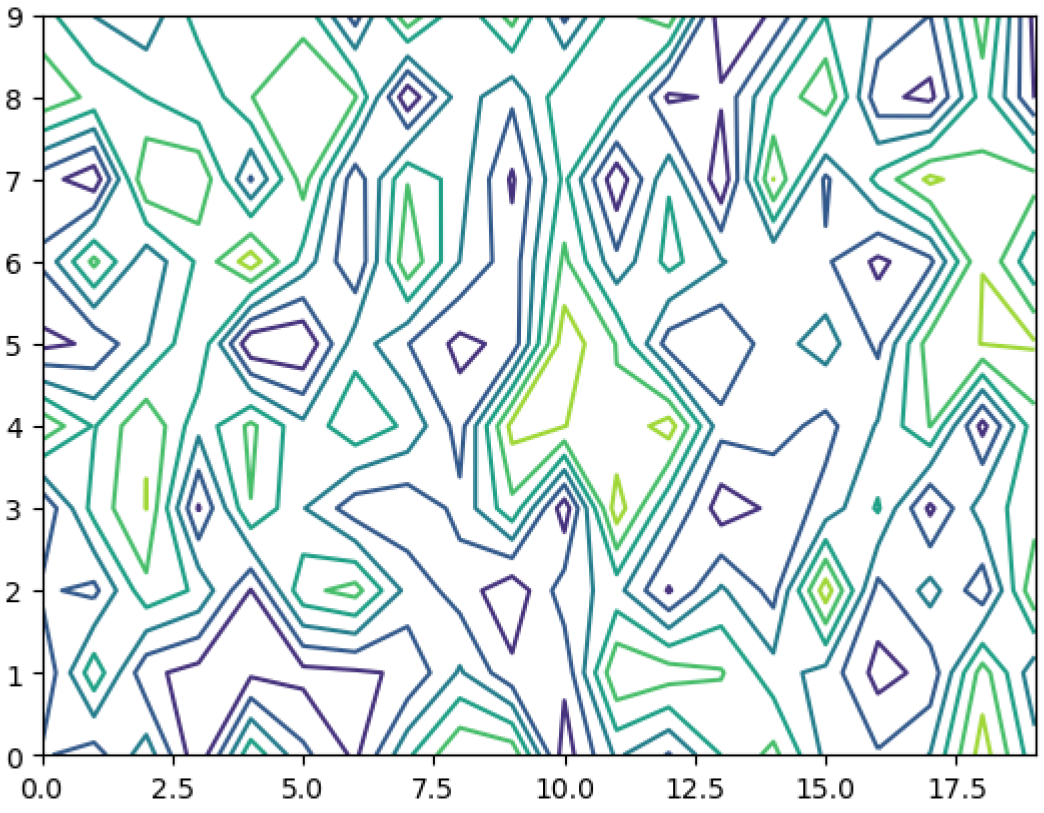
In [46]:
```
x12=range(1,6)
y12=[1,4,6,8,4] # data creation

# area plot
plt.stackplot(x12,y12) # can also be used byt fill_between is more efficient
plt.show();
```



used to display 3-d data in 2-d, also know as level lines or isolines

- several applications in metrology
- density of line= slope, gradient is perpendicular to contour lines

In [47]:
```
# create a matrix
matrix1=np.random.rand(10,20)
cp=plt.contour(matrix1)
plt.show();
```



Styles in Matplotlib

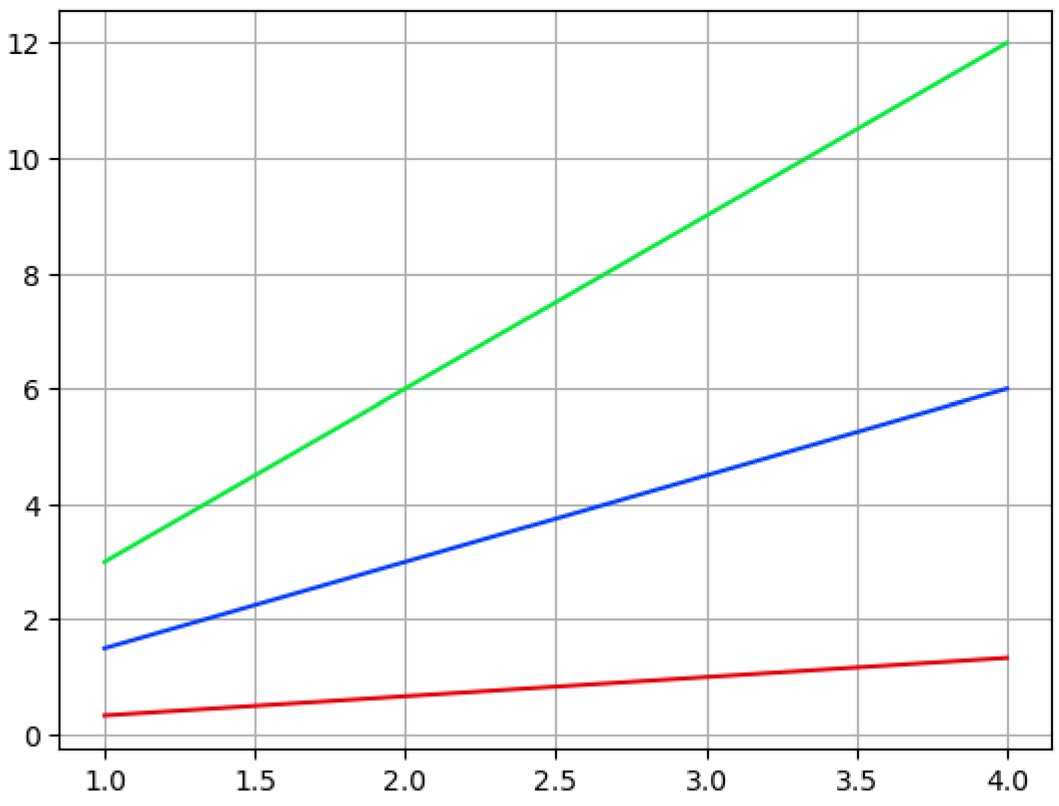
In [48]:
```
# view list of al styles availabel
print(plt.style.available)
```

```
['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']
```

In [49]:
```
# set style for plots
plt.style.use('seaborn-v0_8-bright')
```

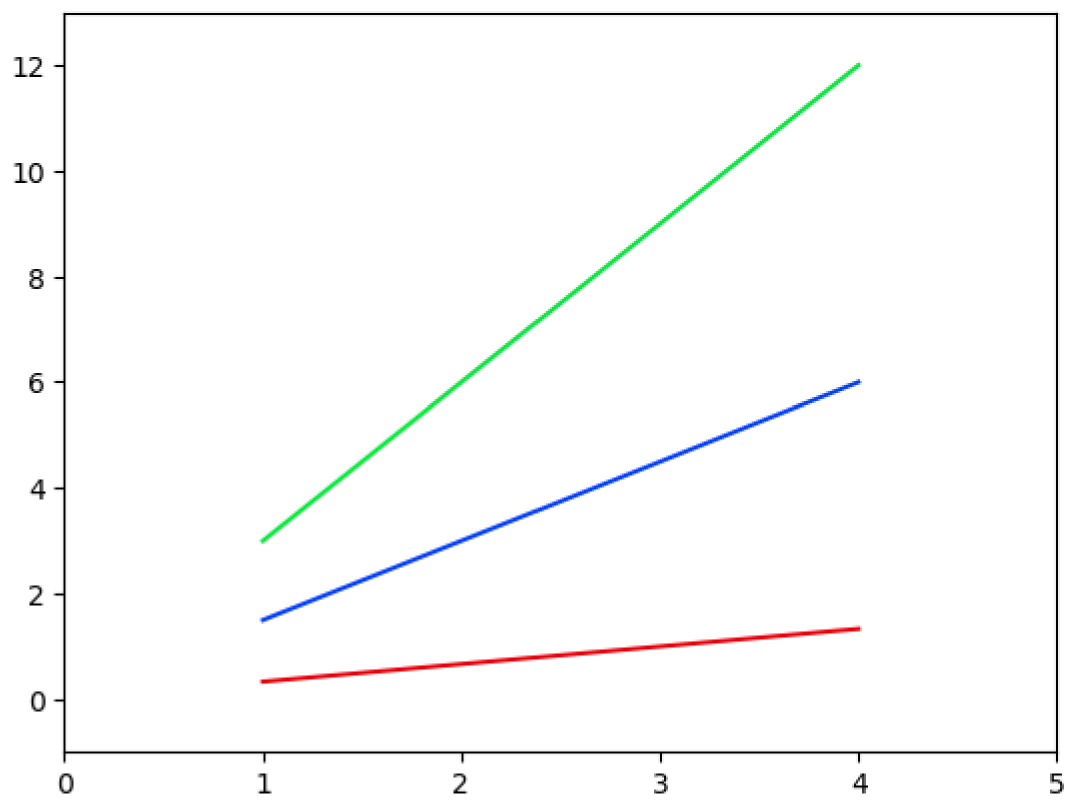
Adding a grid

In [51]:
```
x15=np.arange(1,5)
plt.plot(x15,x15*1.5,x15,x15*3.0,x15,x15/3.0)
plt.grid(True) # Takes only one arg i.e., T/F
plt.show()
```



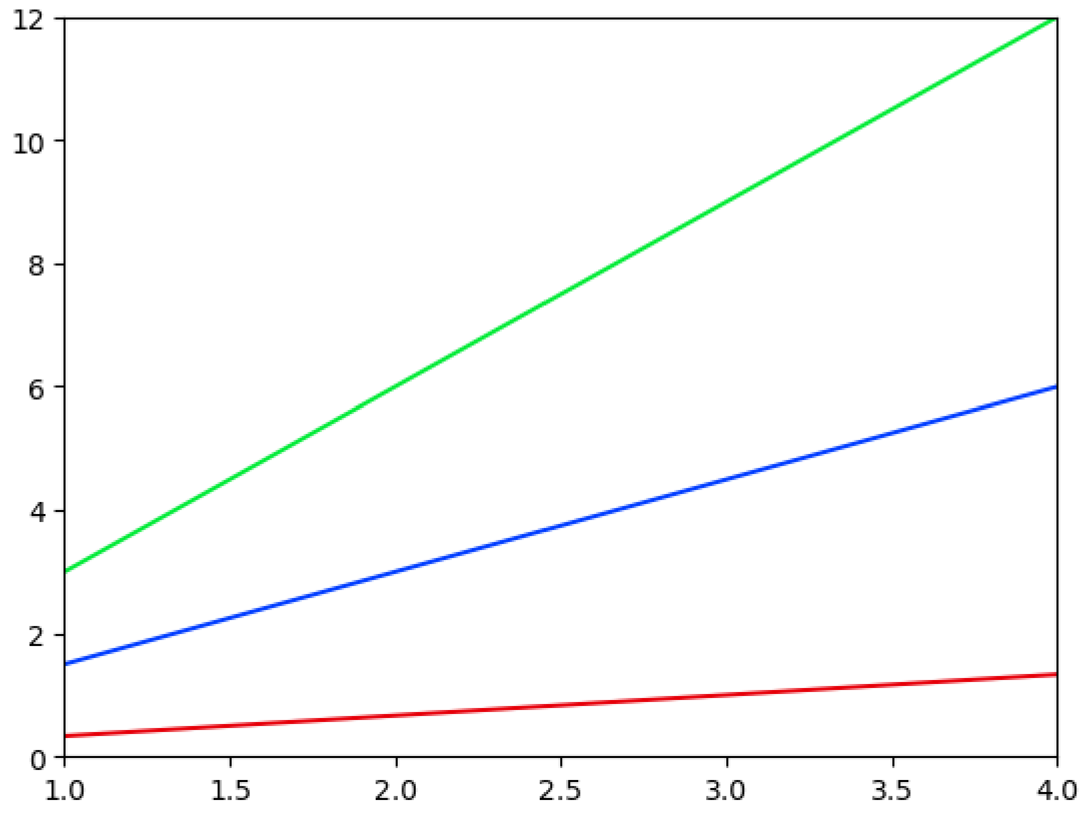
Handling Axes

In [53]:
```
x15=np.arange(1,5)
plt.plot(x15,x15*1.5,x15,x15*3.0,x15,x15/3.0)
plt.axis() # shows current axis limit values
plt.axis([0,5,-1,13])
plt.show();
```



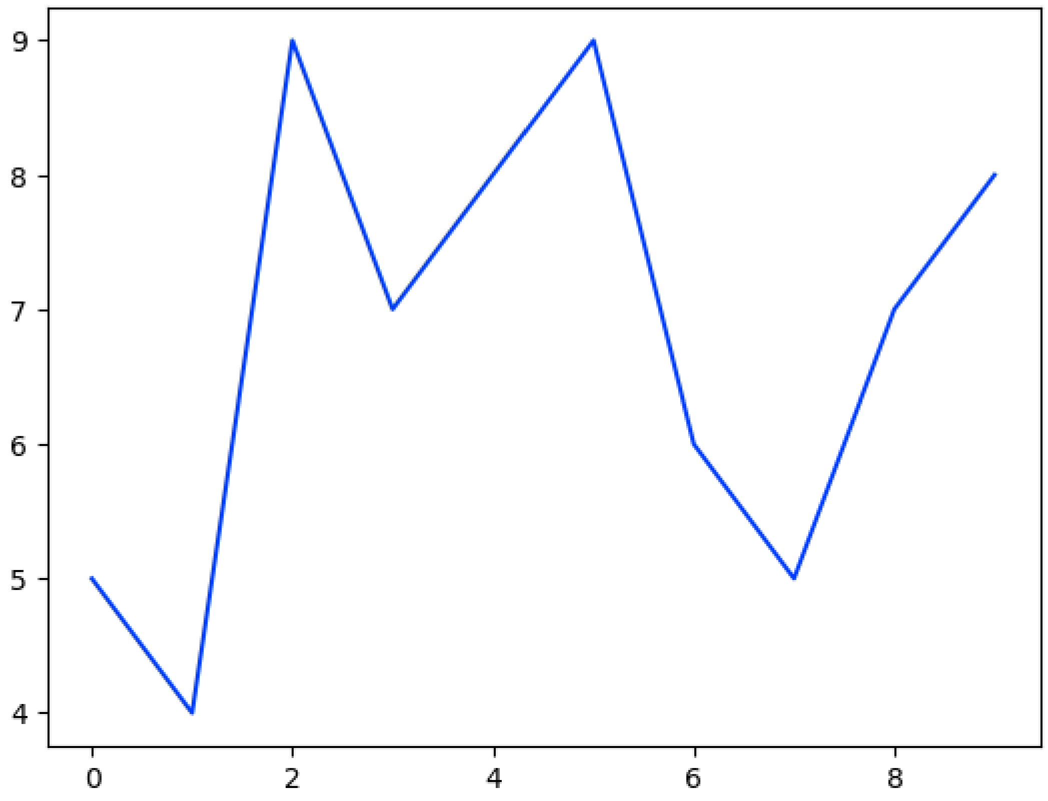
In [54]:
```
x15=np.arange(1,5)
plt.plot(x15,x15*1.5,x15,x15*3.0,x15,x15/3.0)
plt.xlim([1.0,4.0])
plt.ylim(0.0,12.0)
```

Out[54]:
```
(0.0, 12.0)
```

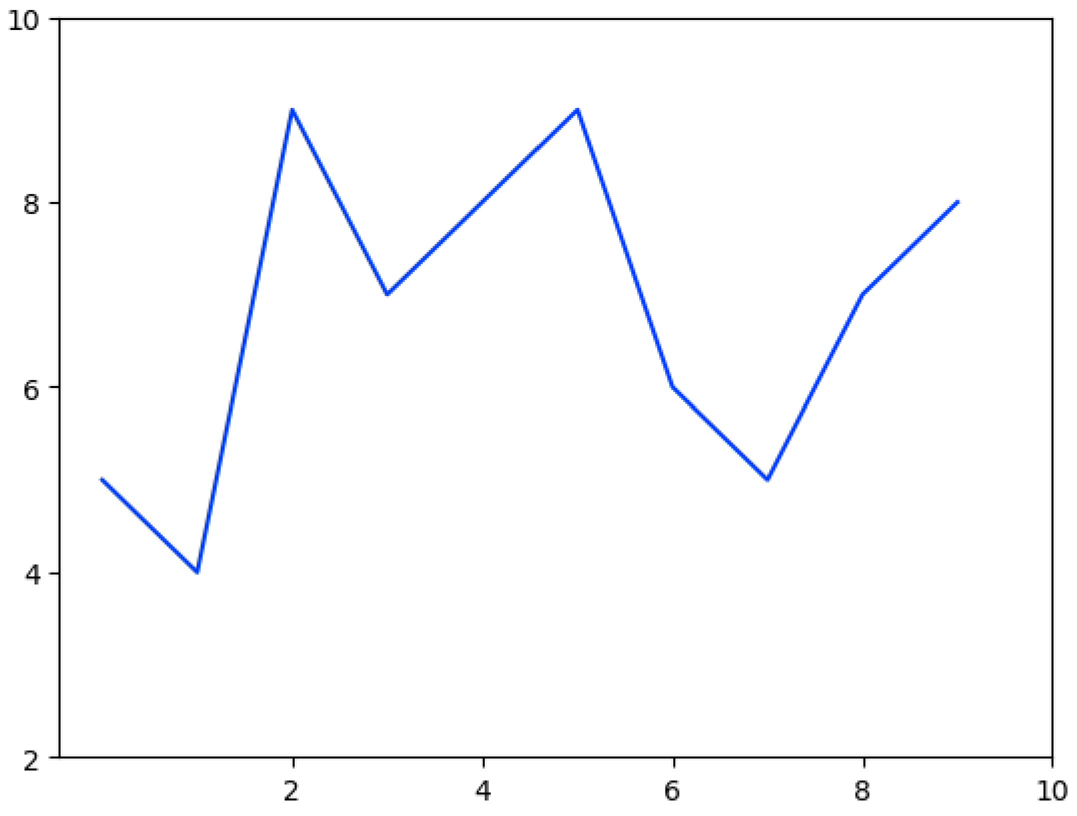


Handling x and y ticks

```
In [55]: # without x and y ticks
         u=[5,4,9,7,8,9,6,5,7,8]
         plt.plot(u)
         plt.show();
```

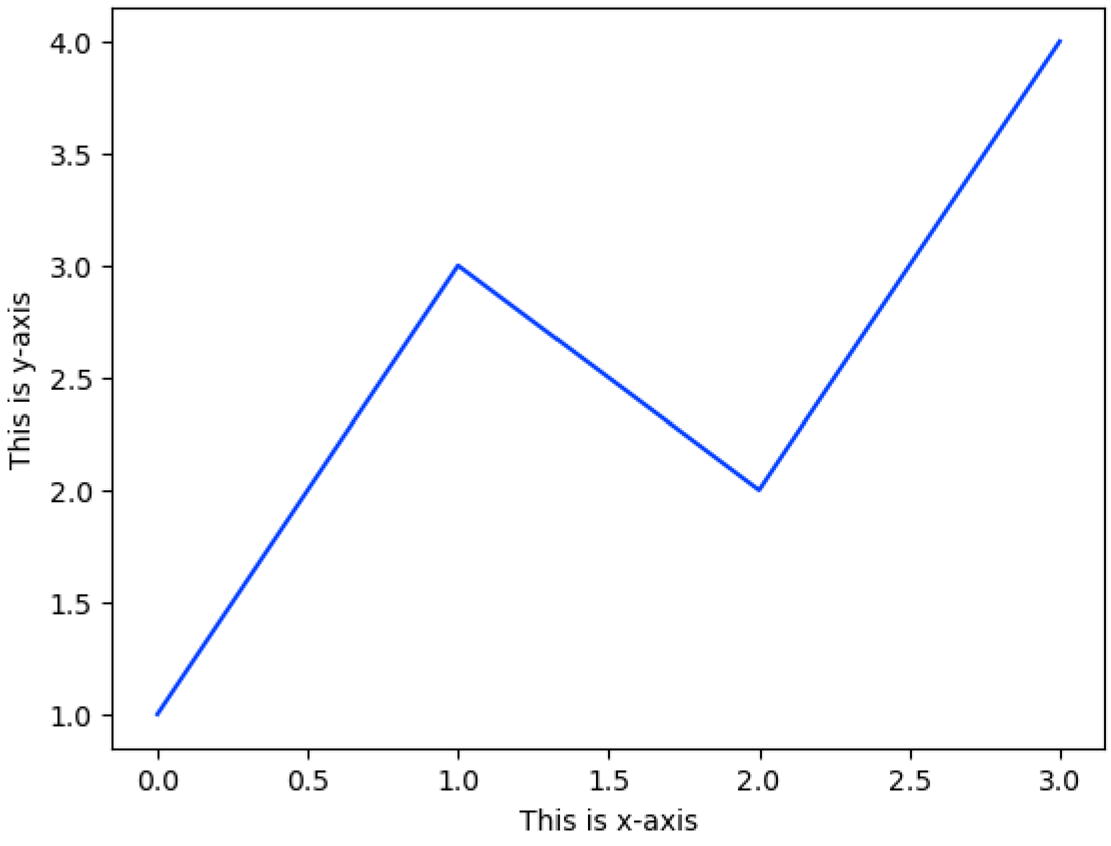


```
In [56]: # with x and y ticks
         u=[5,4,9,7,8,9,6,5,7,8]
         plt.plot(u)
         plt.xticks([2,4,6,8,10])
         plt.yticks([2,4,6,8,10])
         plt.show();
```



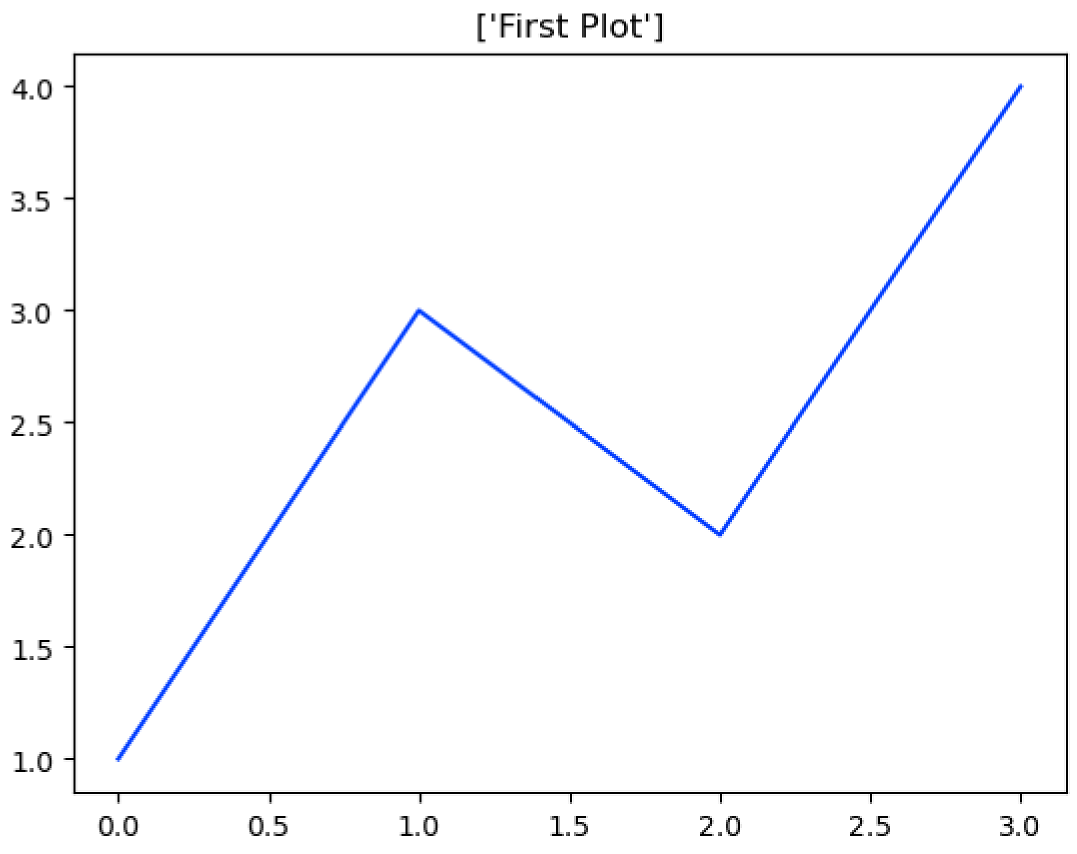
Adding Labels

```
In [58]: plt.plot([1,3,2,4])
         plt.xlabel('This is x-axis')
         plt.ylabel('This is y-axis')
         plt.show()
```



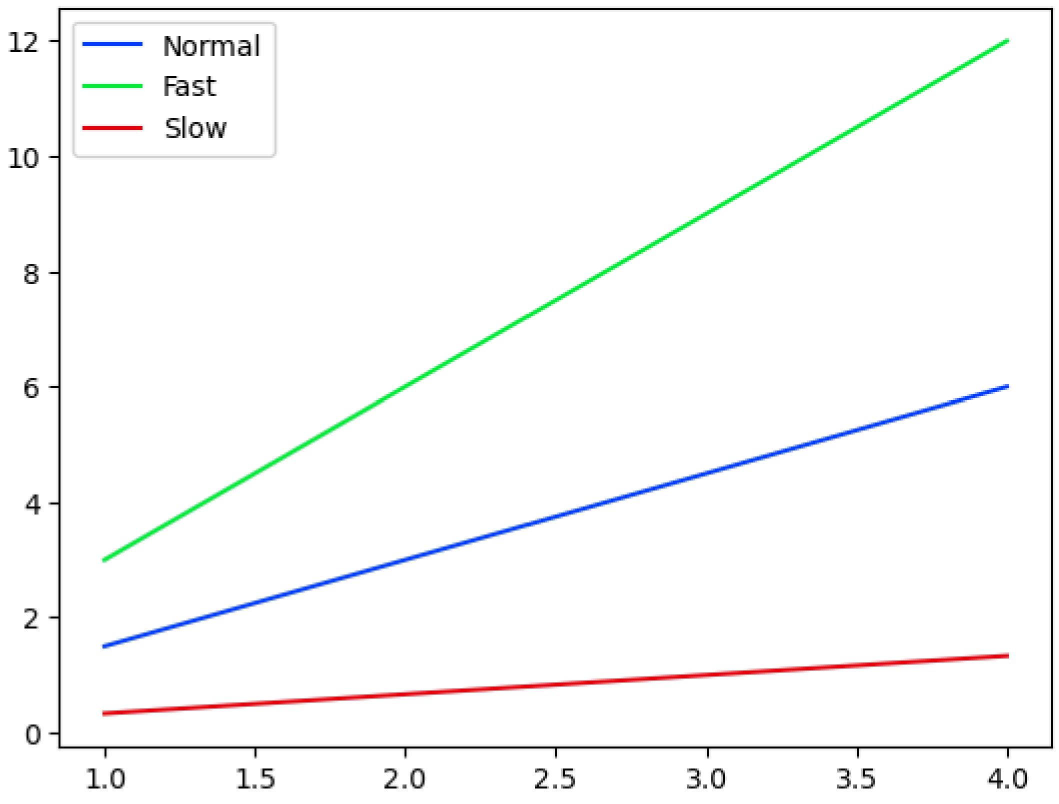
Adding a title

In [59]:
```
plt.plot([1,3,2,4])
plt.title(['First Plot'])
plt.show();
```



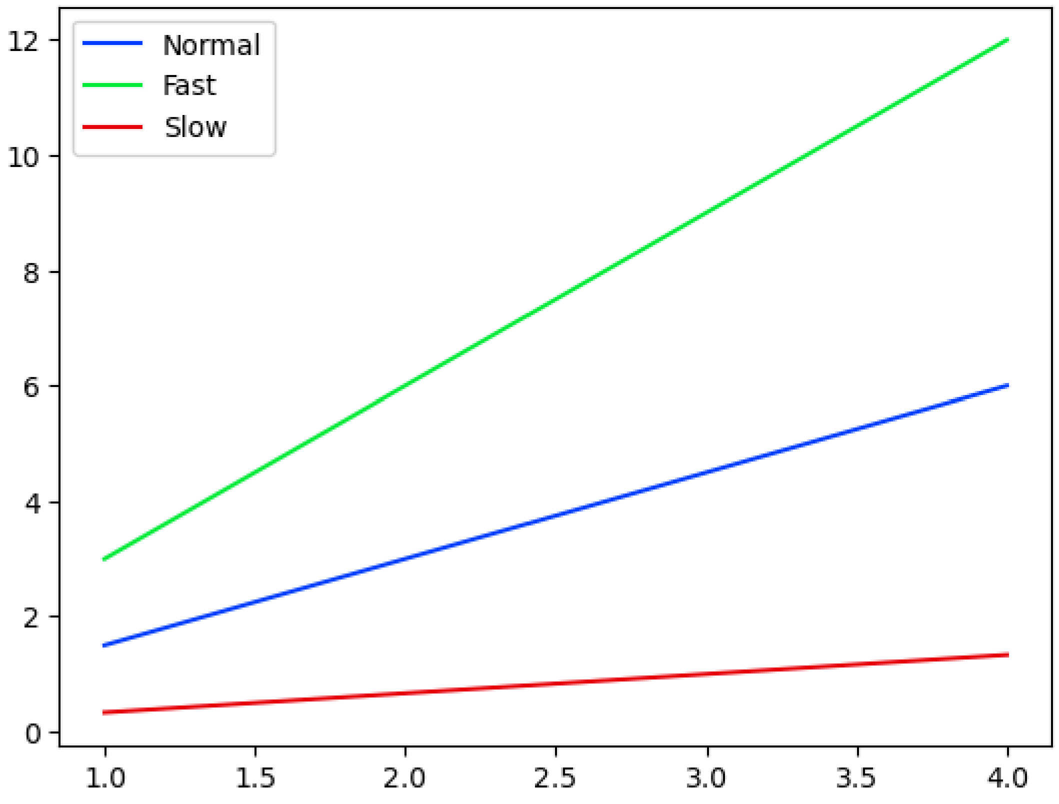
Adding Legend

In [60]:
```
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'])
```



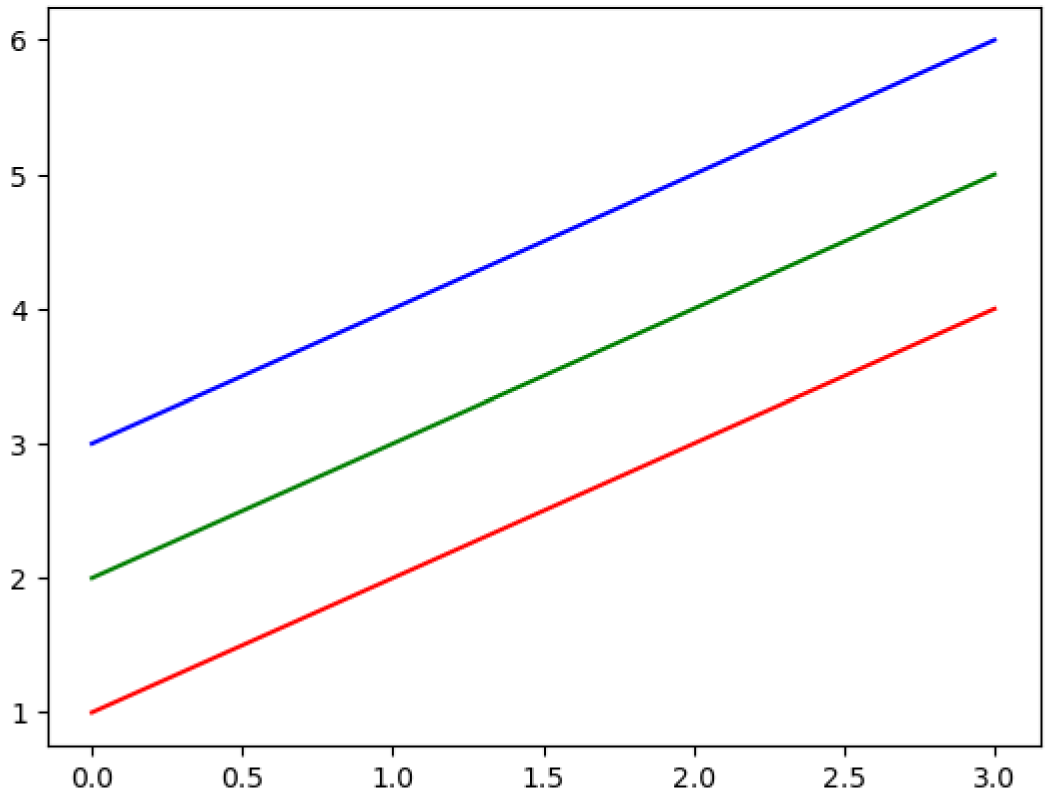
In [61]:
```
x15=np.arange(1,5)
fig,ax=plt.subplots()
ax.plot(x15,x15*1.5,label='Normal')
ax.plot(x15,x15*3.0,label='Fast')
ax.plot(x15,x15/3.0,label='Slow')
ax.legend()
```

Out[61]:
```
<matplotlib.legend.Legend at 0x26da0aff530>
```



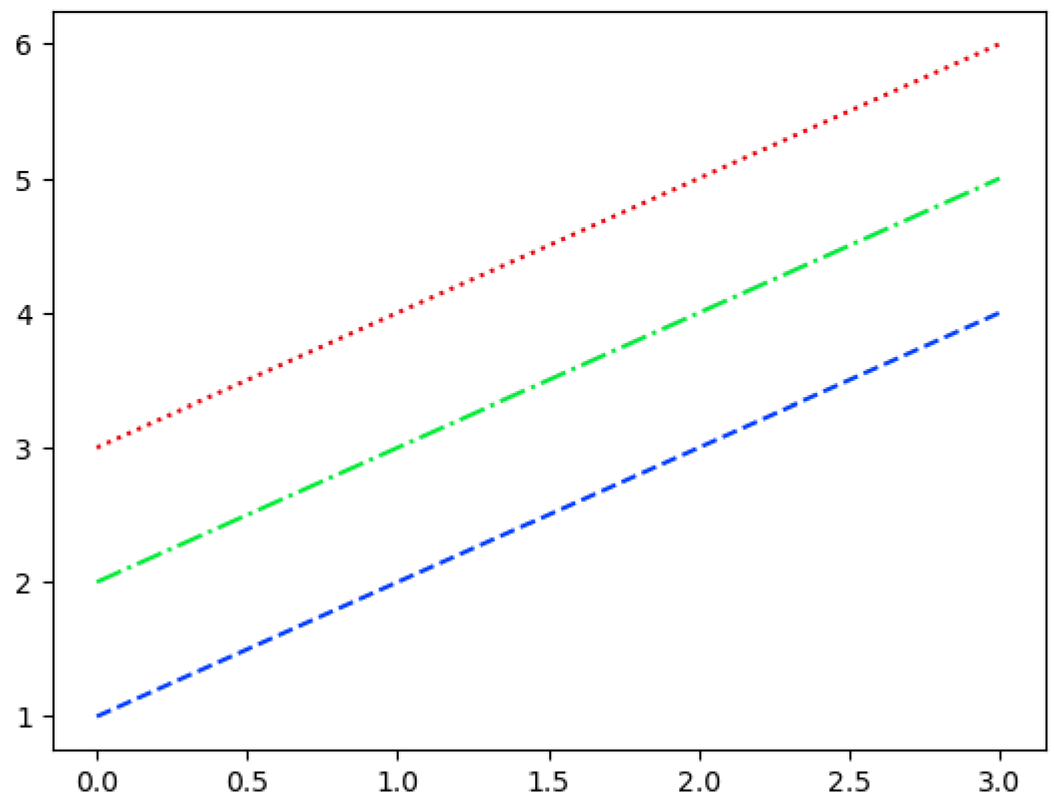
Control Colors

In [63]:
```
x16=np.arange(1,5)
plt.plot(x16,'r')
plt.plot(x16+1,'g')
plt.plot(x16+2,'b')
plt.show();
```



Control Line Styles

In [64]:
```
x16=np.arange(1,5)
plt.plot(x16,'--',x16+1,'-.',x16+2,':')
plt.show();
```



THE END