Introduction to Matplotlib

Matplotlib – Key Points

Basic plotting library in Python.

Most prominent tool for data visualization.

Produces publication-quality figures.

Supports multiple output formats: PDF, SVG, JPG, PNG, BMP, GIF.

Can create various plots:

Line plot

Scatter plot

Histogram

Bar chart

Error chart

Pie chart

Box plot

Many more

Supports 3D plotting.

Highly efficient for a wide range of tasks.

Base for other libraries like Pandas and Seaborn.

Pandas/Seaborn allow using Matplotlib features with less code.

In [1]:
```
# import matplotlib
import matplotlib
```

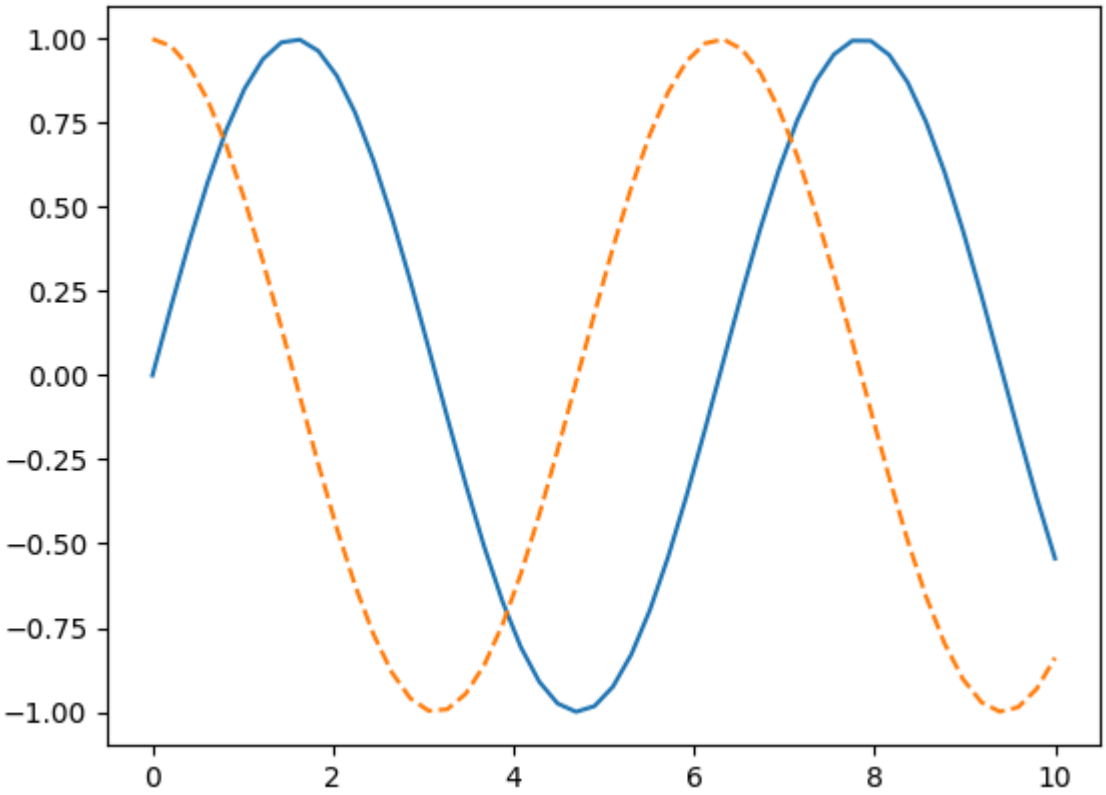
In [2]:
```
import matplotlib.pyplot as plt #pyplot is a interface
```

In [3]:
```
import numpy as np #import bum
import pandas as pd
```

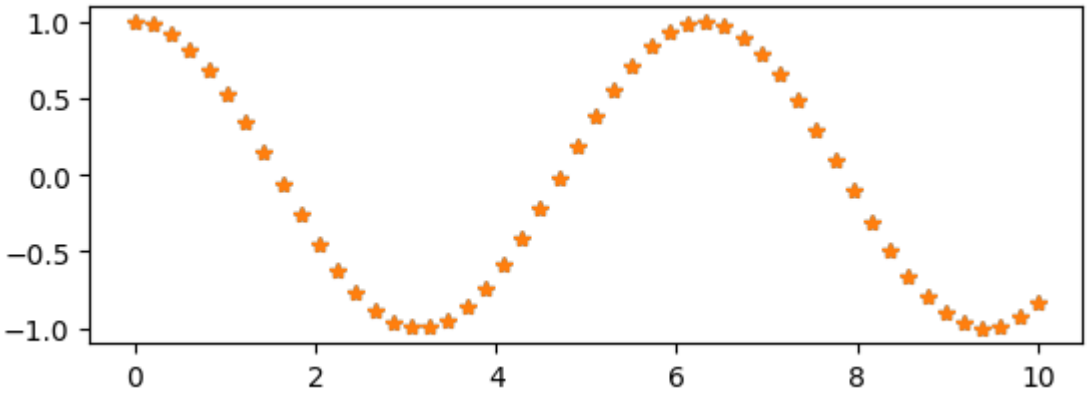
Displaying Plots in Matplotlib

In [5]:
```
#%matplotlib inline - It will output static images of the plot embedded in the n
%matplotlib inline
x1 = np.linspace(0, 10, 50)  #Creates an array of 50 evenly spaced values betwee
```

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

plt.plot(x1, np.sin(x1), '-')#Plots the sine curve ('-' for solid line).
plt.plot(x1, np.cos(x1), '--')#plots the cosine curve ('--' for dashed line).
#plt.plot(x1, np.tan(x1), '--')
plt.show()
```

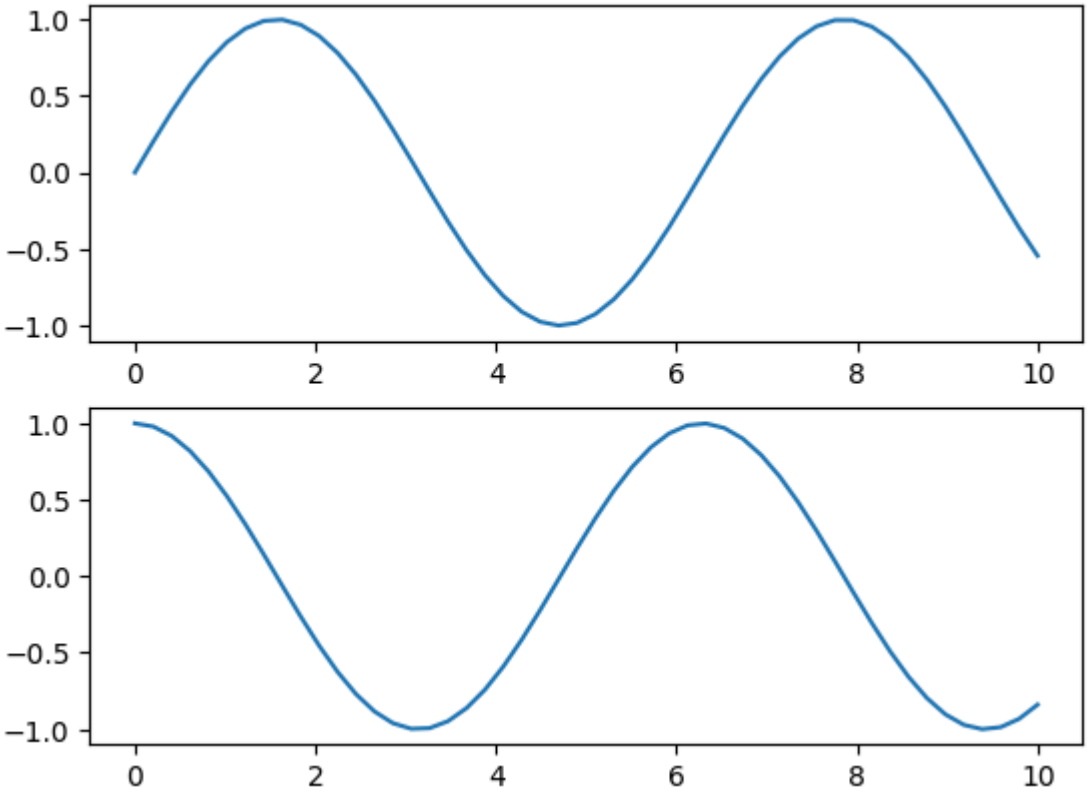


In [7]:
```
# create the first of two panels and set current axis
plt.subplot(2, 1, 1)   # (rows, columns, panel number)
plt.plot(x1, np.cos(x1), '*')
plt.show()
```



In [8]:
```
# 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))
plt.show()
```

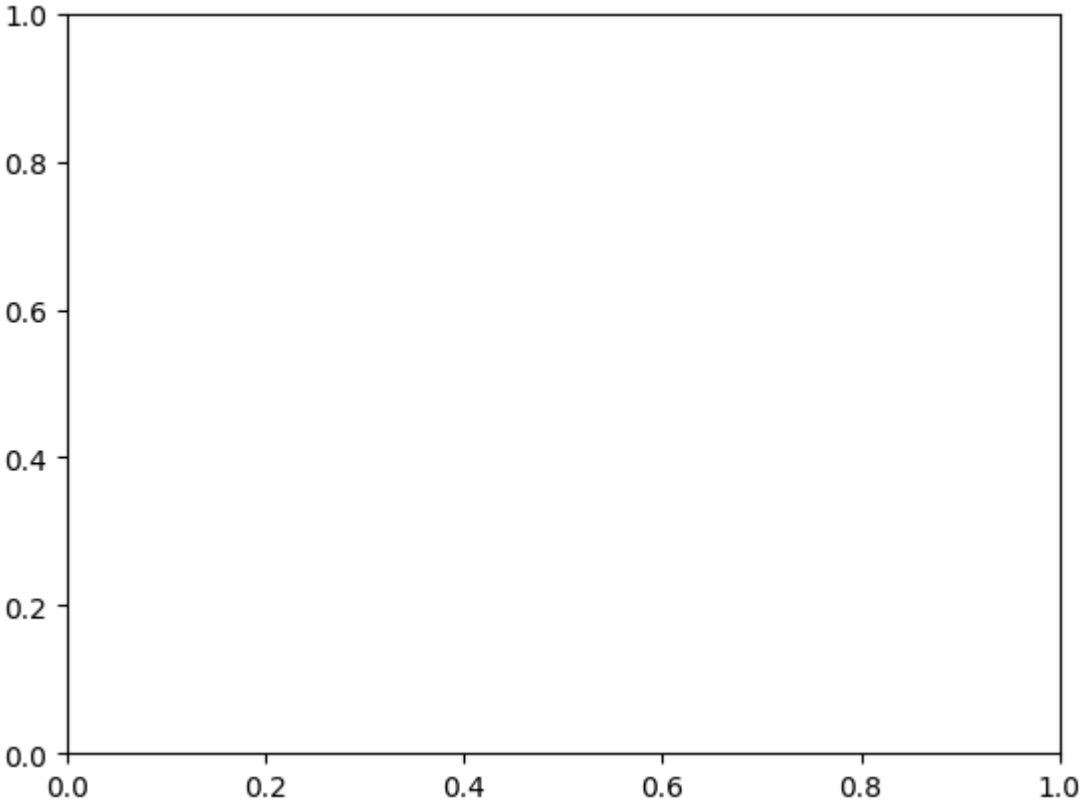


In [9]:
```
# get current figure information

print(plt.gcf())
```

```
Figure(640x480)
```

In [11]:
```
# get current axis information

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

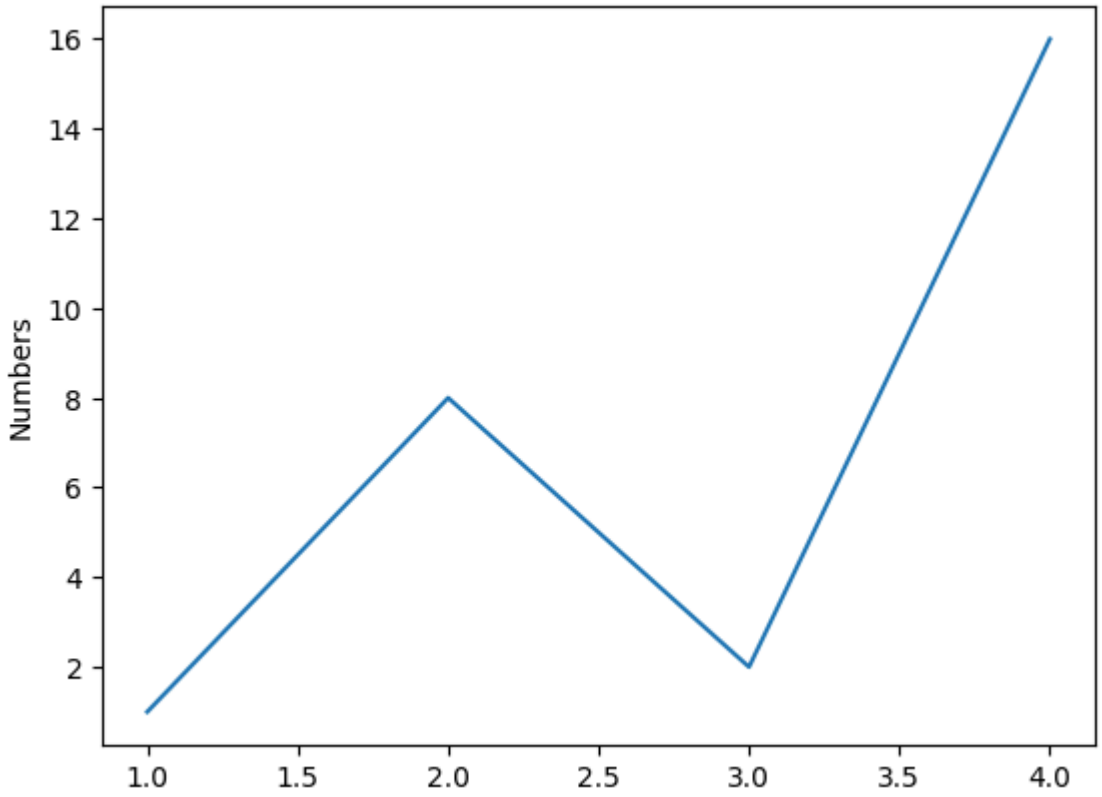
```
Axes(0.125,0.11;0.775x0.77)
```



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 starts with 0. Hence the x data are [0,1,2,3] and y data are [1,2,3,4].

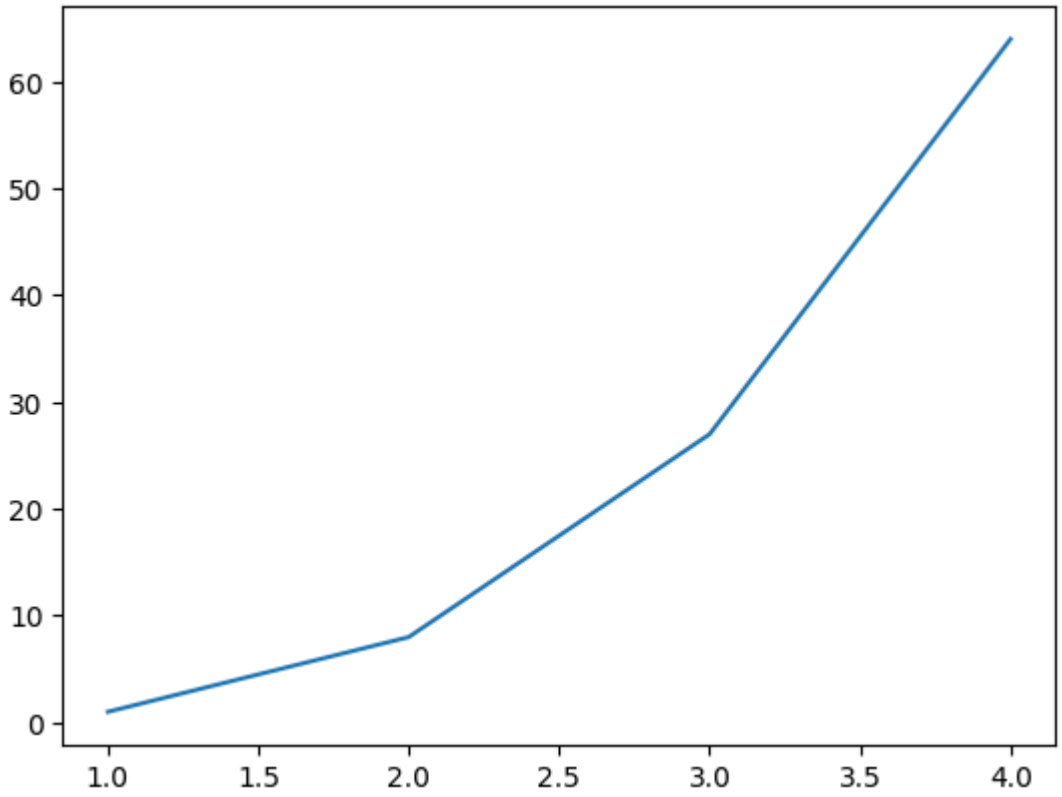
In [14]:
```
plt.plot([1,2,3,4], [1,8,2,16])  #Plots points (1,1), (2,8), (3,2), (4,16) conne

                                 #First list → x values, second list → y val
plt.ylabel('Numbers')  #Sets the y-axis label to "Numbers".
plt.show()
```



plot() - A versatile command

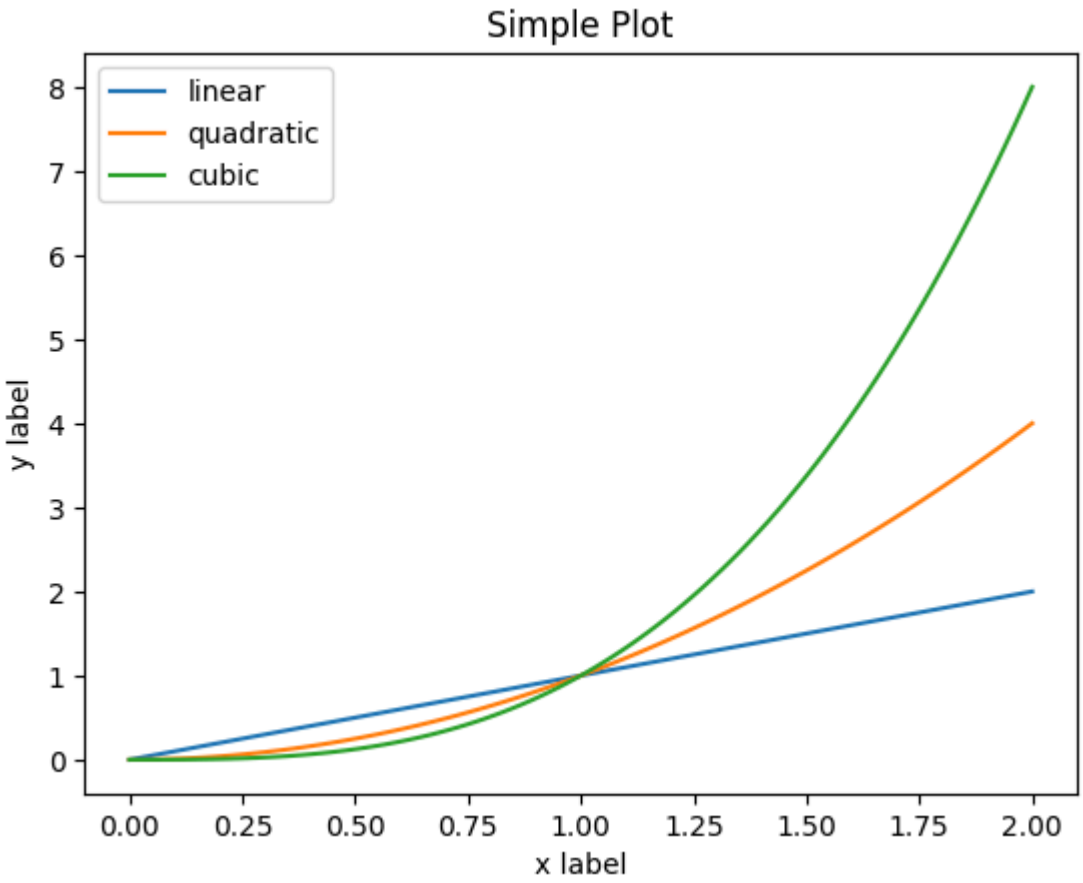
plot() is a versatile command. It will take an arbitrary number of arguments. For example, to plot x versus y, we can issue the following command:-

In [15]:
```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], [1, 8, 27, 64])
plt.show()
```



In [16]:
```
#State-machine interface
#Pyplot provides the state-machine interface to the underlying object-oriented p
#The state-machine implicitly and automatically creates figures and axes to achi
```

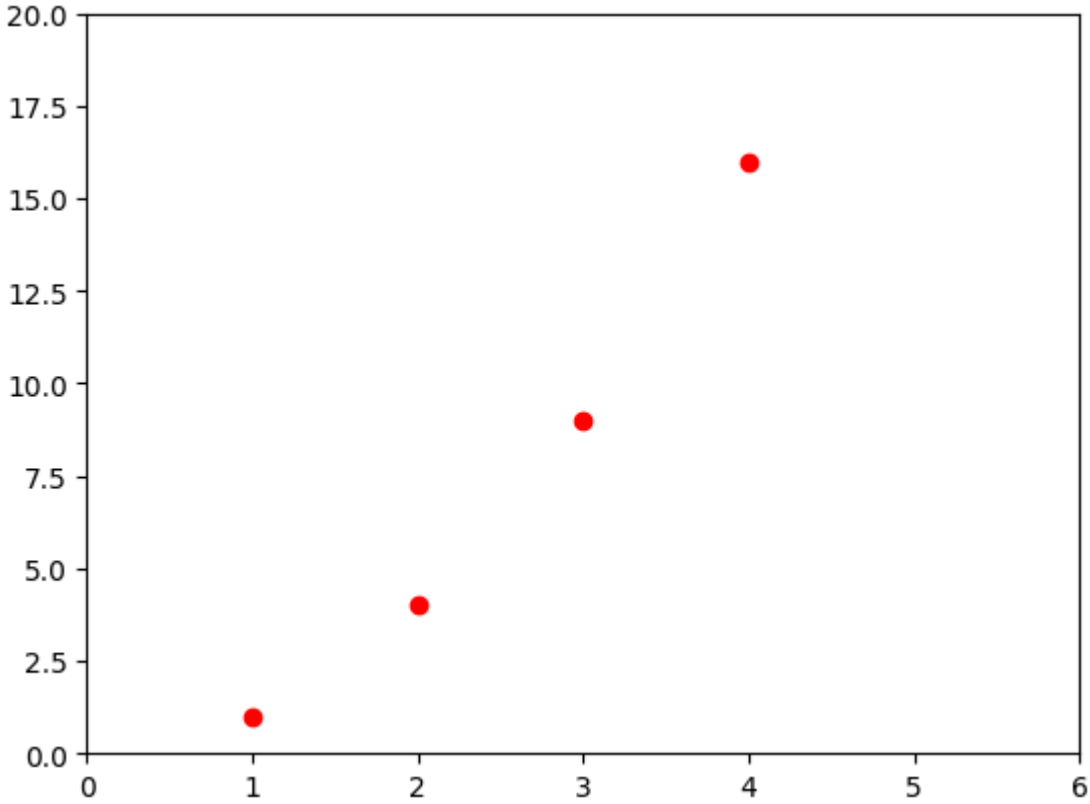
In [17]:
```
x = np.linspace(0, 2, 100)  #Creates an array of 100 evenly spaced values betwee

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

plt.xlabel('x label')  #Labels the x-axis and y-axis.
plt.ylabel('y label')

plt.title("Simple Plot") #title of plot is

plt.legend() #Shows the legend (uses label= values from plt.plot()).

plt.show()
```



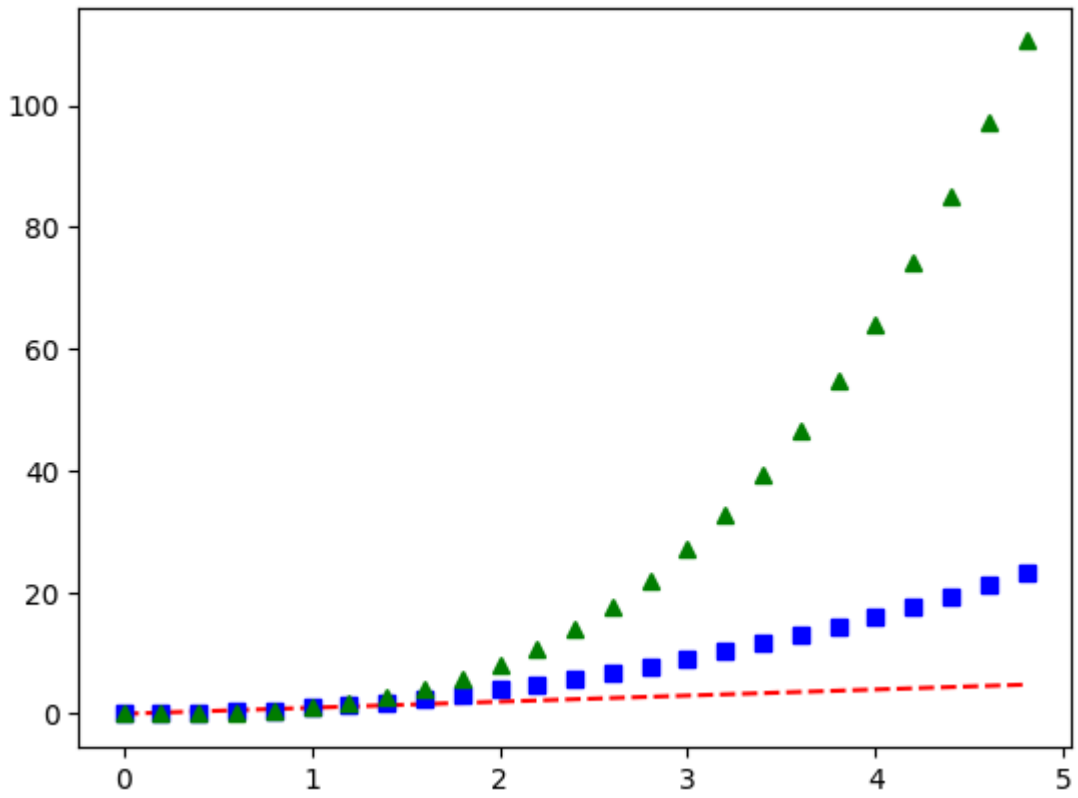
Formatting the style of plot

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 from 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 command:-

In [18]:
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro') #plot points are 1,1),(2,4),(3,9),(4
#This creates a red dots plot instead of lines.
plt.axis([0, 6, 0, 20]) #x-axis: 0 to 6 and y-axis: 0 to 20  and
plt.show()
```



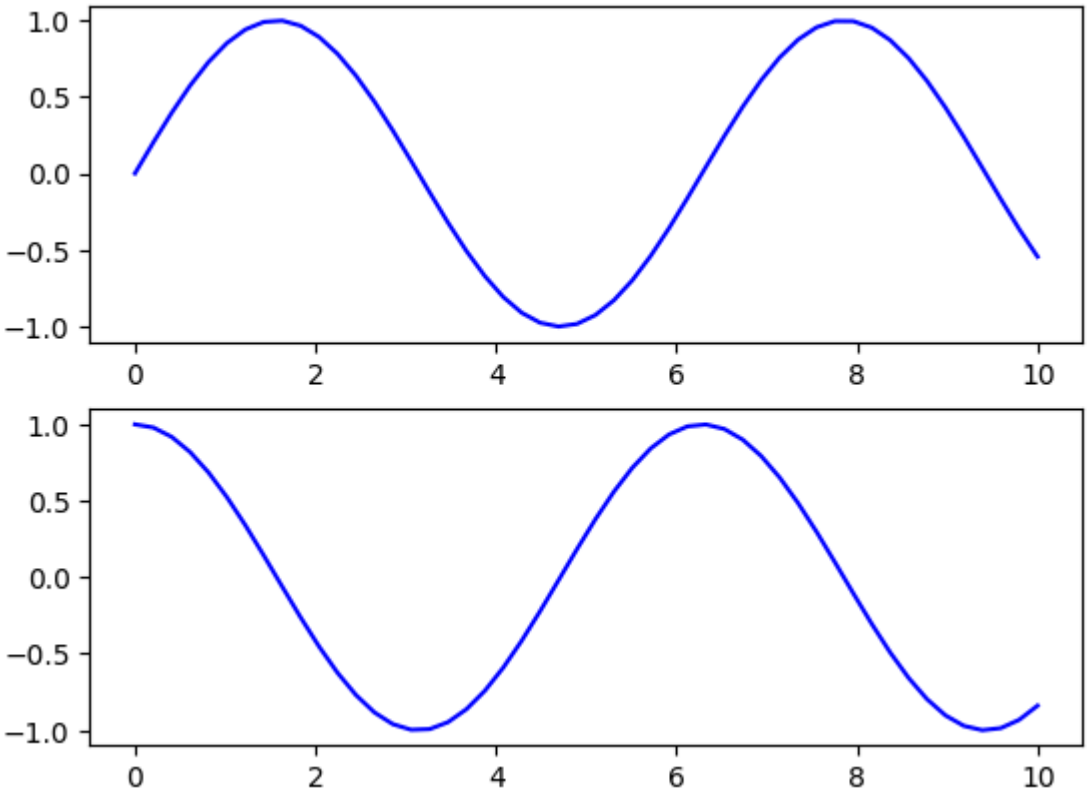
In [19]:
```
# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2) #Creates values from 0 to 5 with a step of 0.2 second

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^') #'r--' → Red dashed line (y
#'g^' → Green triangle markers (y = t³)
plt.show()
```



Object-Oriented API

The Object-Oriented (OO) API in Matplotlib is used for more complex plotting where greater control over the figure is required. Unlike the Pyplot API, which relies on an active figure or axes, the OO API works with explicit Figure and Axes objects. A Figure is the top-level container that holds all plot elements, acting like a canvas that can contain one or more Axes. Each Axes represents an individual plot within the figure and contains smaller elements such as axis lines, tick marks, plotted data lines, legends, titles, and text boxes. By calling plotting methods directly on Axes objects, the OO API provides a clear, structured, and flexible way to create and customize plots.

In [22]:
```
# First create a grid of plots
# ax will be an array of two Axes objects
fig, ax = plt.subplots(2) #Creates a Figure (fig) and an array of two Axes (ax)
#ax[0] → top plot, ax[1] → bottom plot.


# Call plot() method on the appropriate object
ax[0].plot(x1, np.sin(x1), 'b-') #Plots sin(x1) on the first subplot (ax[0]).'b-
ax[1].plot(x1, np.cos(x1), 'b-');
plt.show()
```

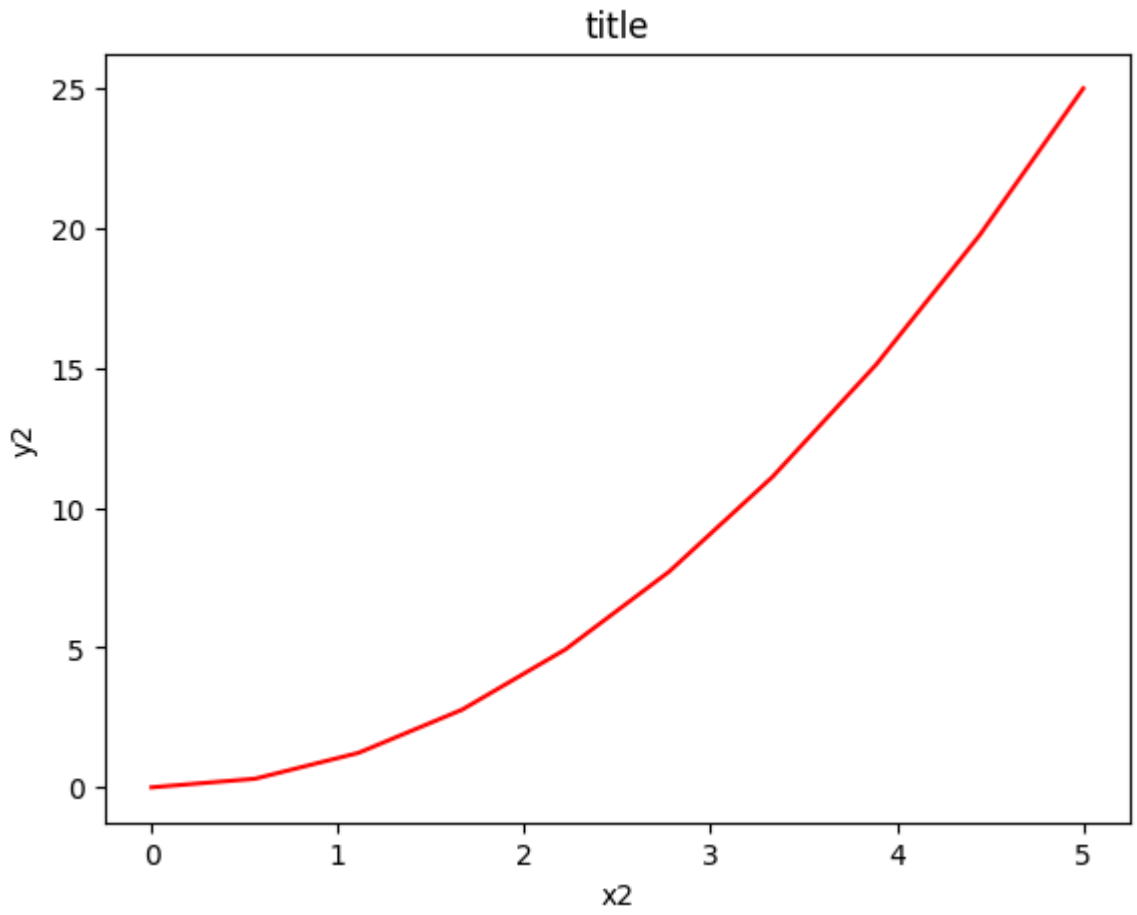


Objects and Reference

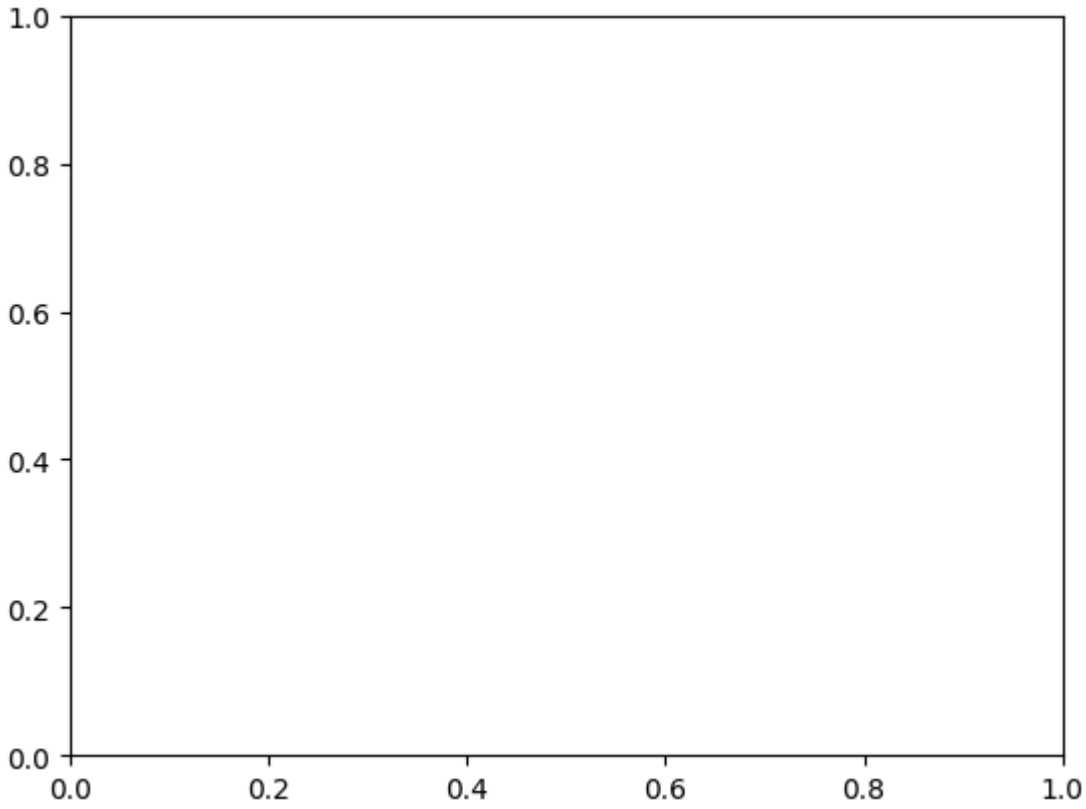
The main idea with the Object Oriented API is to have objects that one can apply functions and actions on. The real advantage of this approach becomes apparent when more than one figure is created or when a figure contains more than one subplot.

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

In [27]:
```
fig = plt.figure() #Creates a new empty Figure.

x2 = np.linspace(0, 5, 10) #Generates 10 evenly spaced values between 0 and 5.
y2 = x2 ** 2 #calculates y =x*2

axes = fig.add_axes([0.1, 0.1, 0.8, 0.8]) #Manually adds Axes to the Figure. The
# the figure size. (0.1, 0.1) → position of bottom-left corner.(0.8, 0.8) → widt

axes.plot(x2, y2, 'r')  # plot a curse as y=x*2 in red colour

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

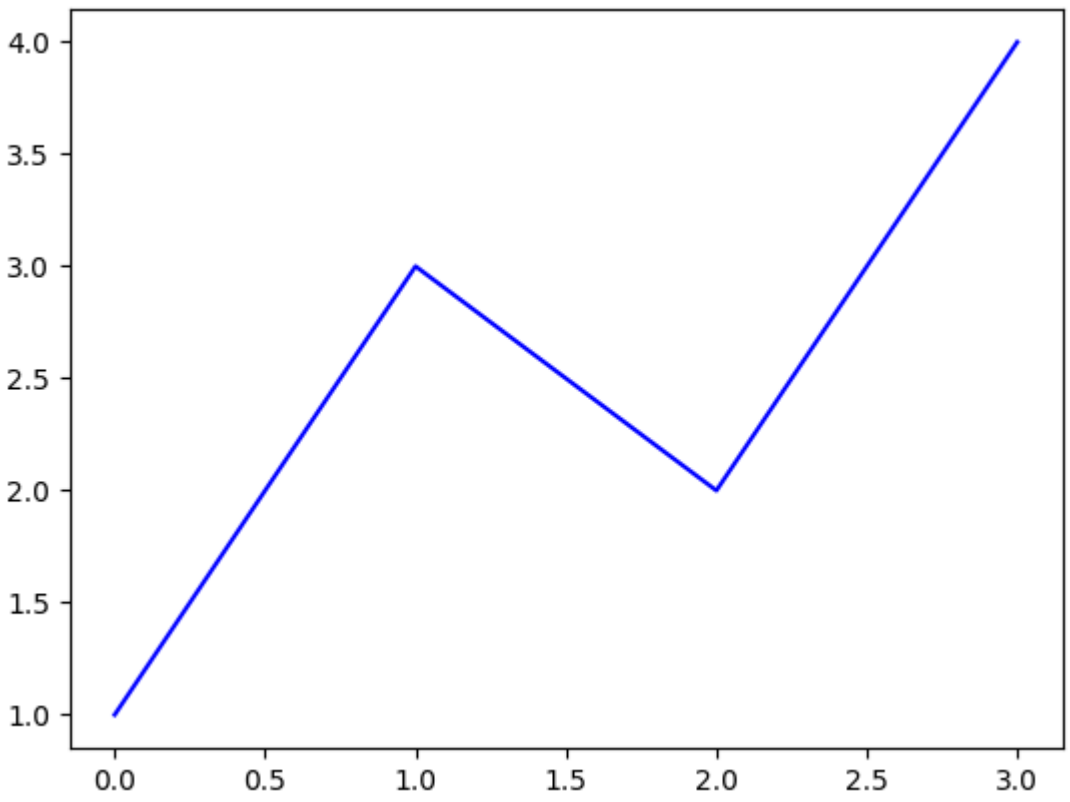


In [30]:
```
fig = plt.figure()
#Creates an empty Figure object (the main container for plots).
ax = plt.axes()#Creates a default Axes object inside the figure.
plt.show()
```

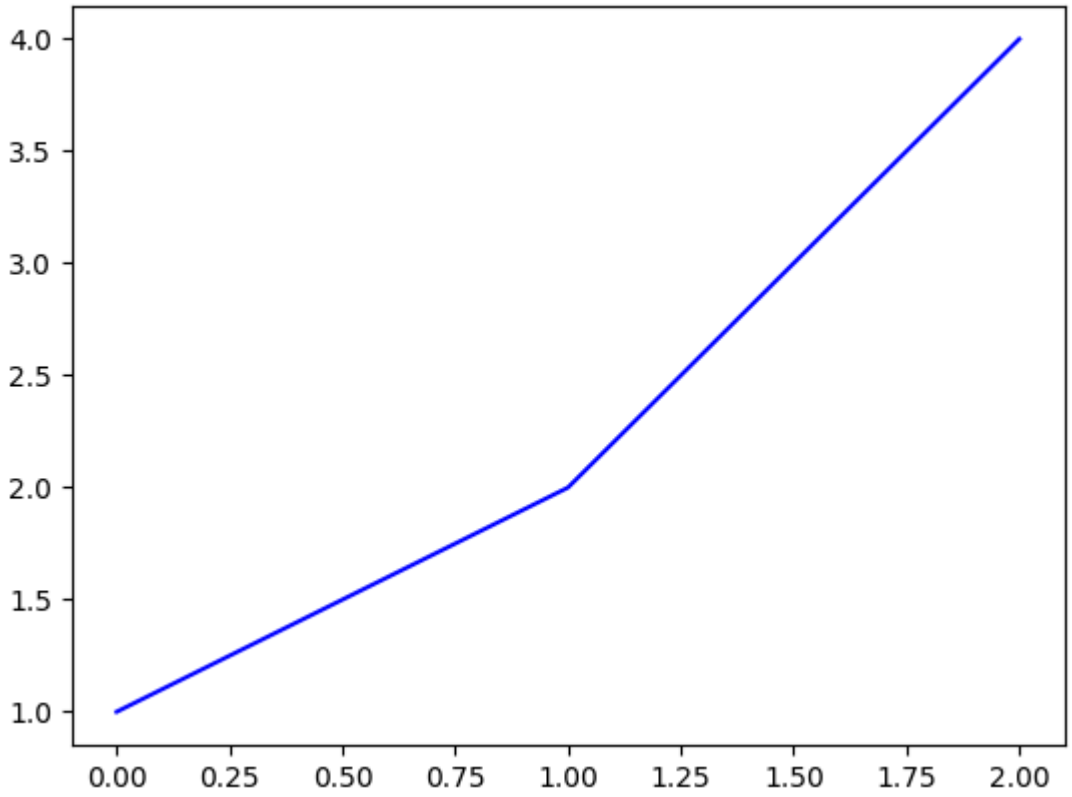


First plot with Matplotlib

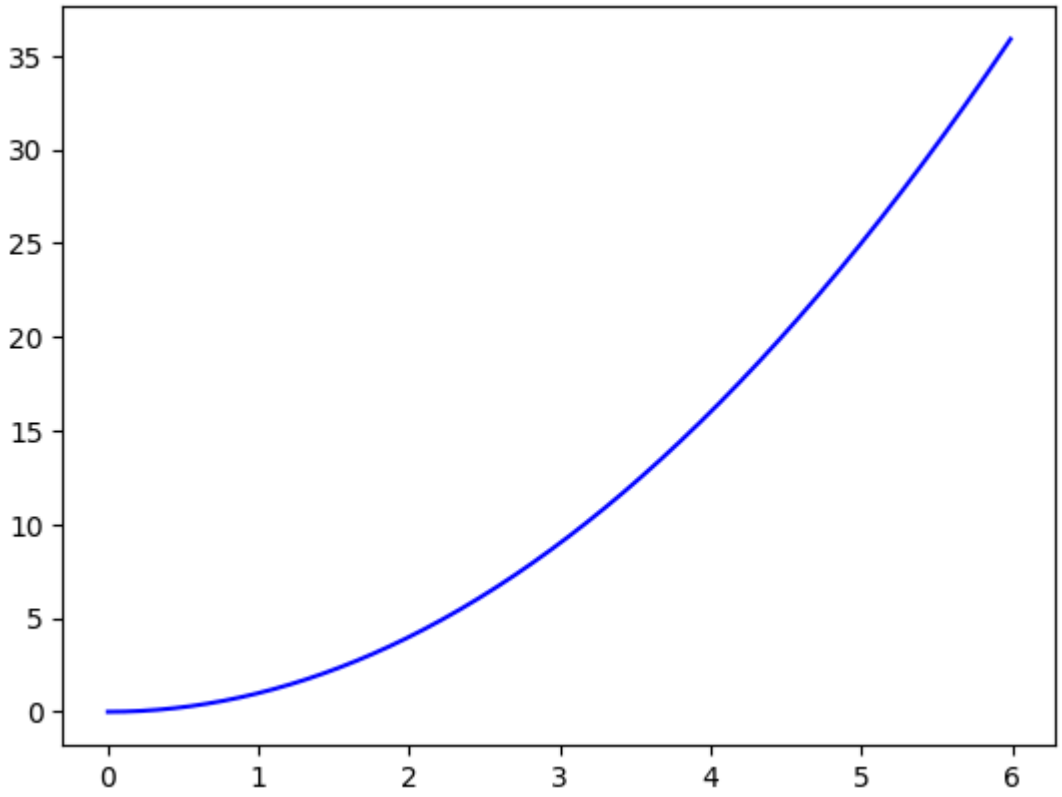
In [32]:
```
plt.plot([1, 3, 2, 4], 'b-')#Plots the values [1, 3, 2, 4] on the y-axis.The x-a

#- → solid line

plt.show( )
```



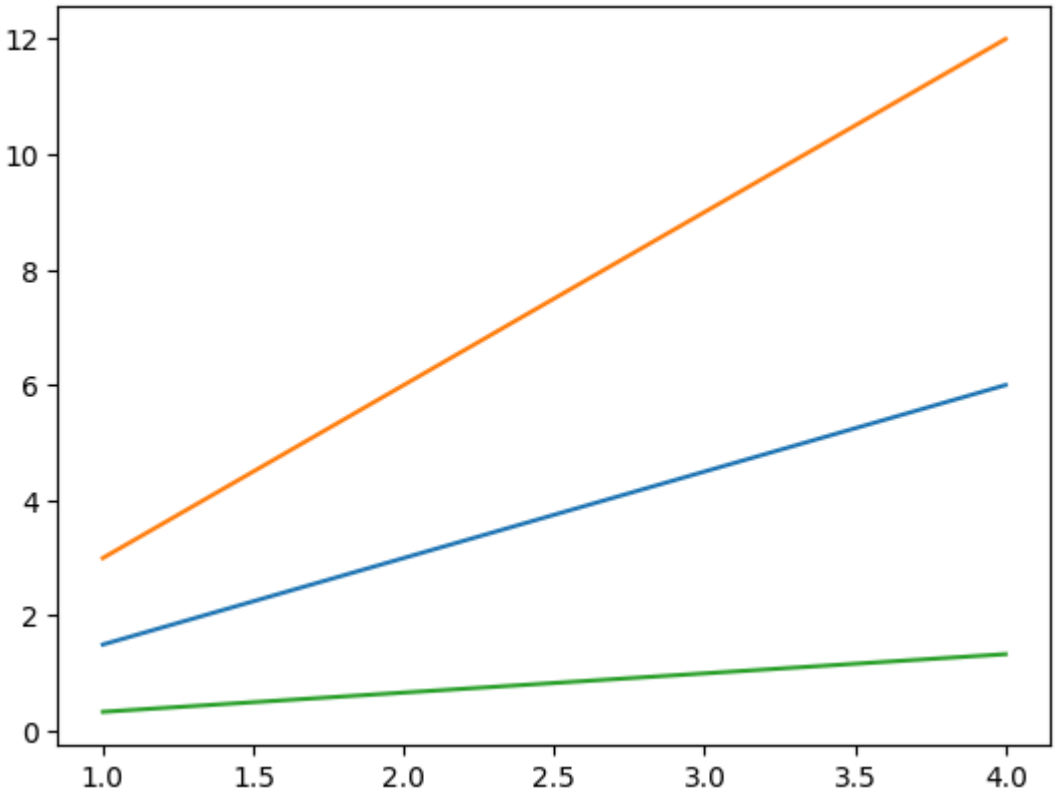
In [33]:
```
plt.plot([1, 2, 4], 'b-')

plt.show( )
```



In [34]:
```
x3 = np.arange(0.0, 6.0, 0.01)

plt.plot(x3, [xi**2 for xi in x3], 'b-') #Calculates y=x*2 for each x3 value usi

#'b-' → blue solid line.

plt.show()
```



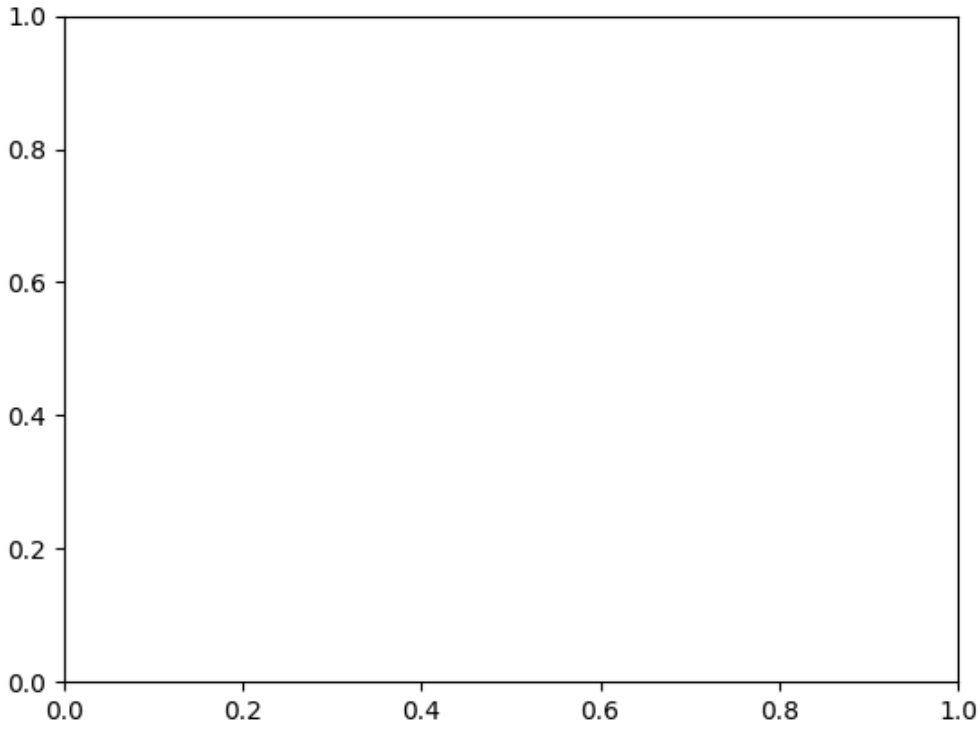
In [35]:
```
#Multiline Plots
x4 = range(1, 5)

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()
```



In [36]:
```
# Saving the figure
fig.savefig('plot1.png')
```

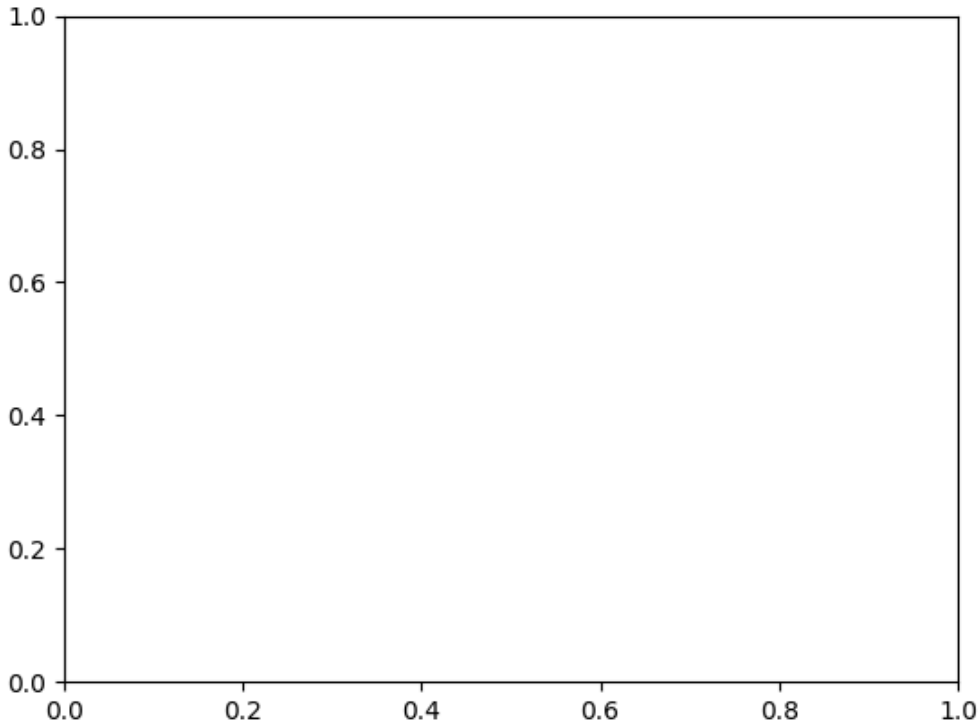
In [37]:
```
# Explore the contents of figure
from IPython.display import Image
Image('plot1.png')
```

Out[37]:



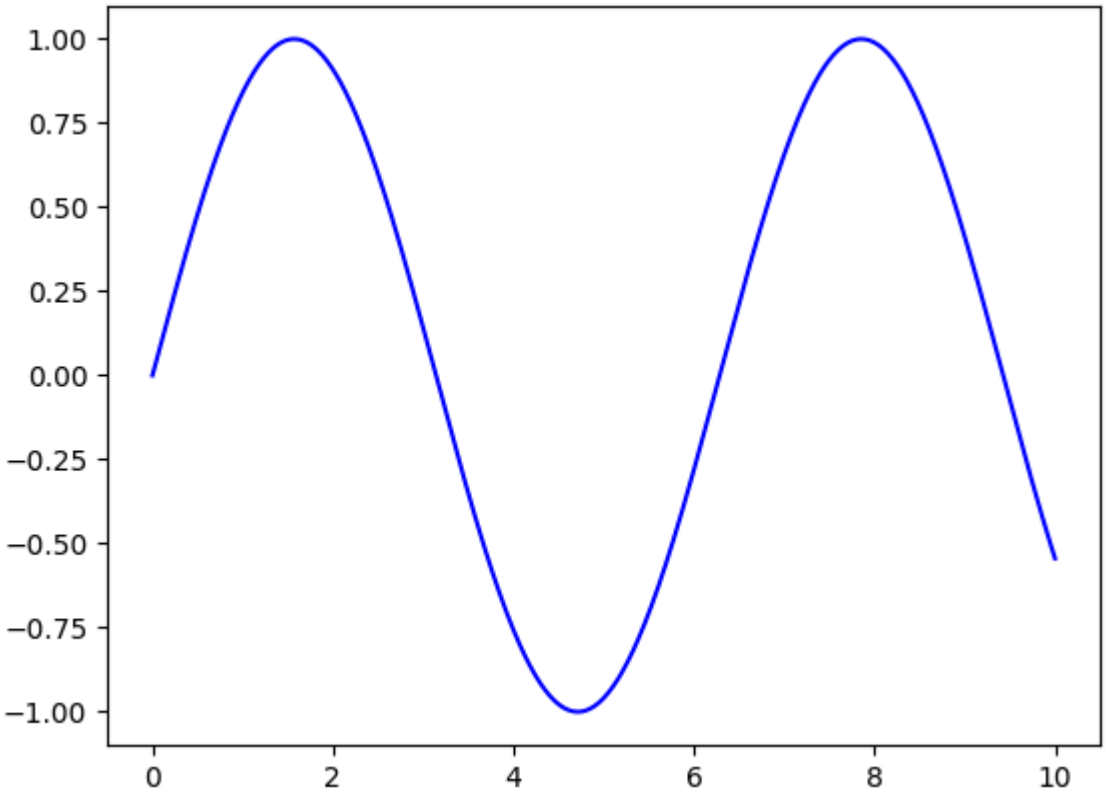
In [38]:
```
# Explore the contents of figure
from IPython.display import Image
Image('plot1.png')
```

Out[38]:

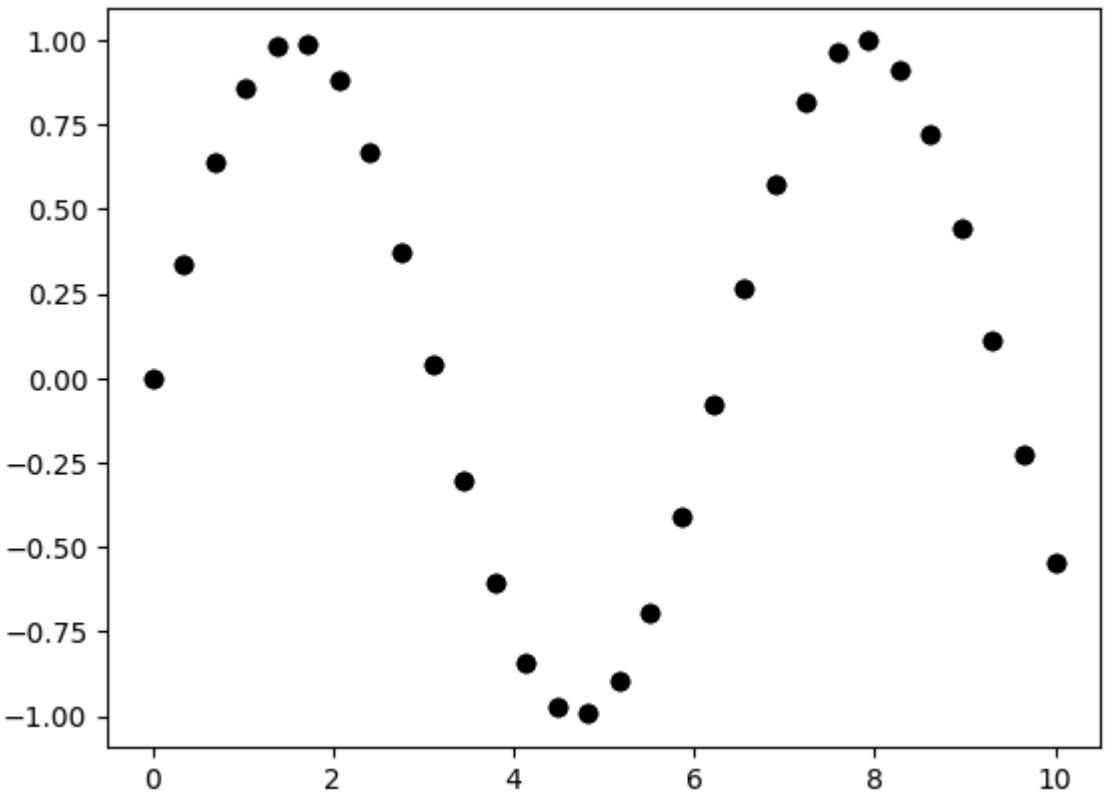


Line Plot

In [41]:
```
# 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-')
plt.show()
```



In [44]:
```
#Scatter Plot with plt.plot()
x7 = np.linspace(0, 10, 30)  #x7 = np.linspace(0, 10, 30)
y7 = np.sin(x7) #calculates the sine of each x7 value.

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

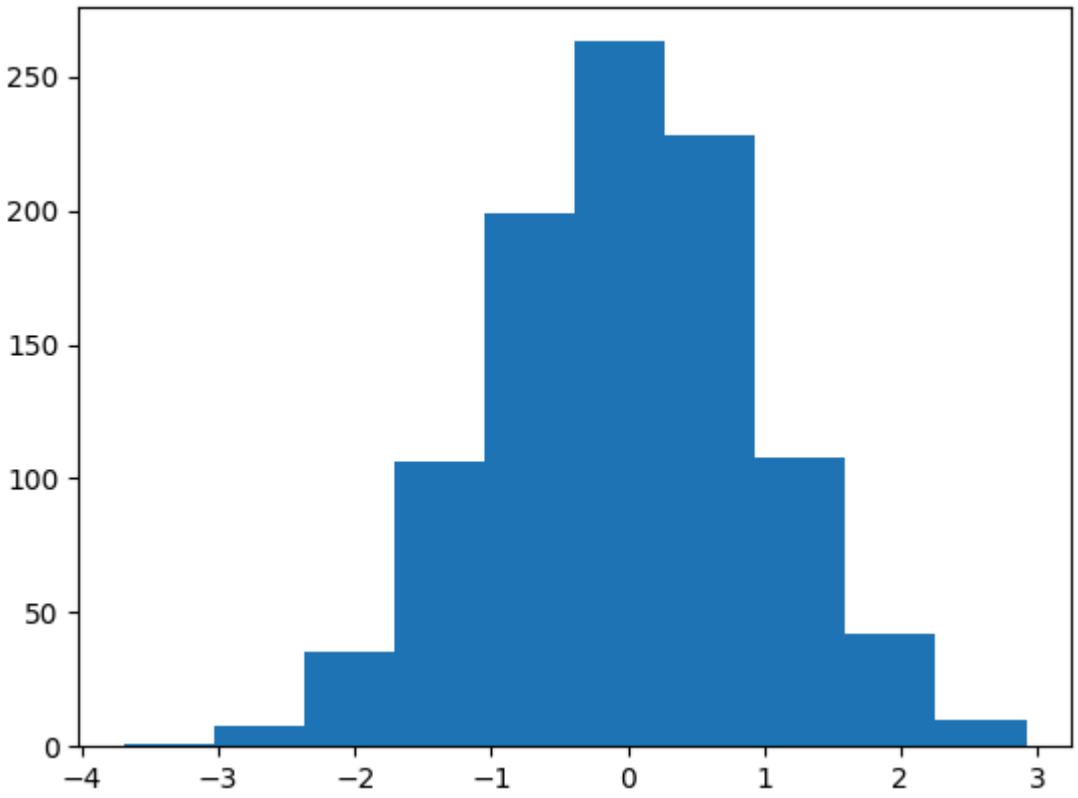


Histogram

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 [47]:
```
data1 = np.random.randn(1000)  #Generates 1000 random numbers from a standard no

plt.hist(data1); #Creates a histogram showing how the data is distributed  and ;
plt.show()
```

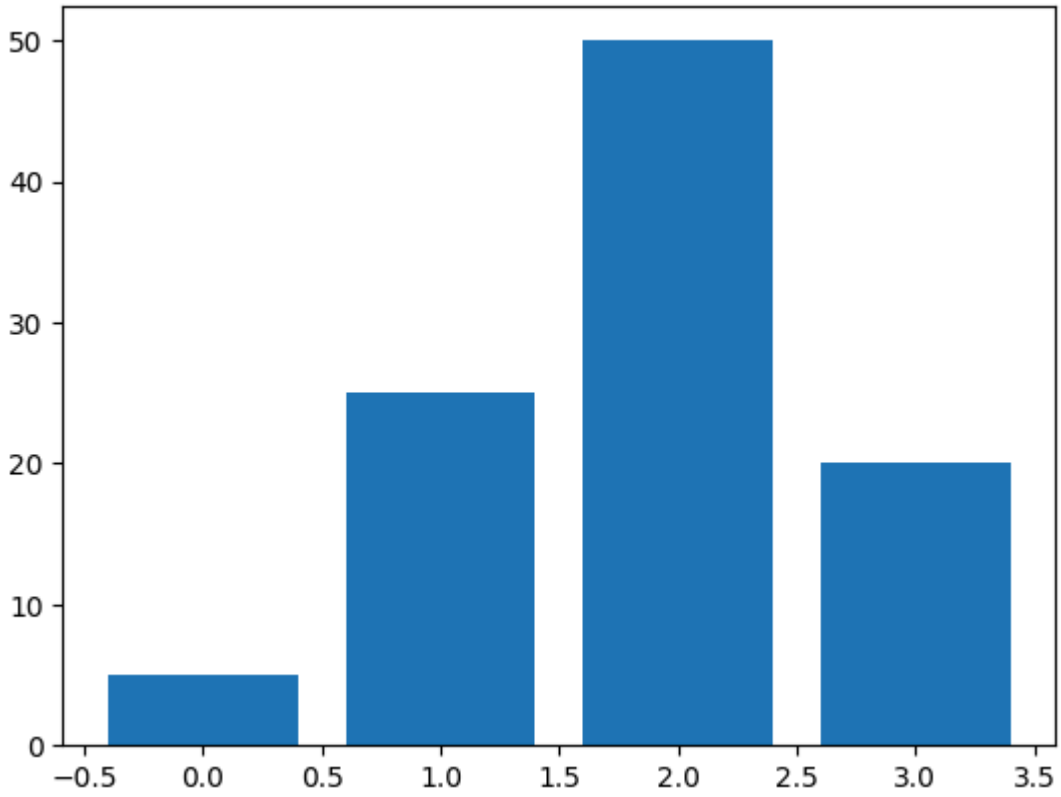


Bar Chart

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

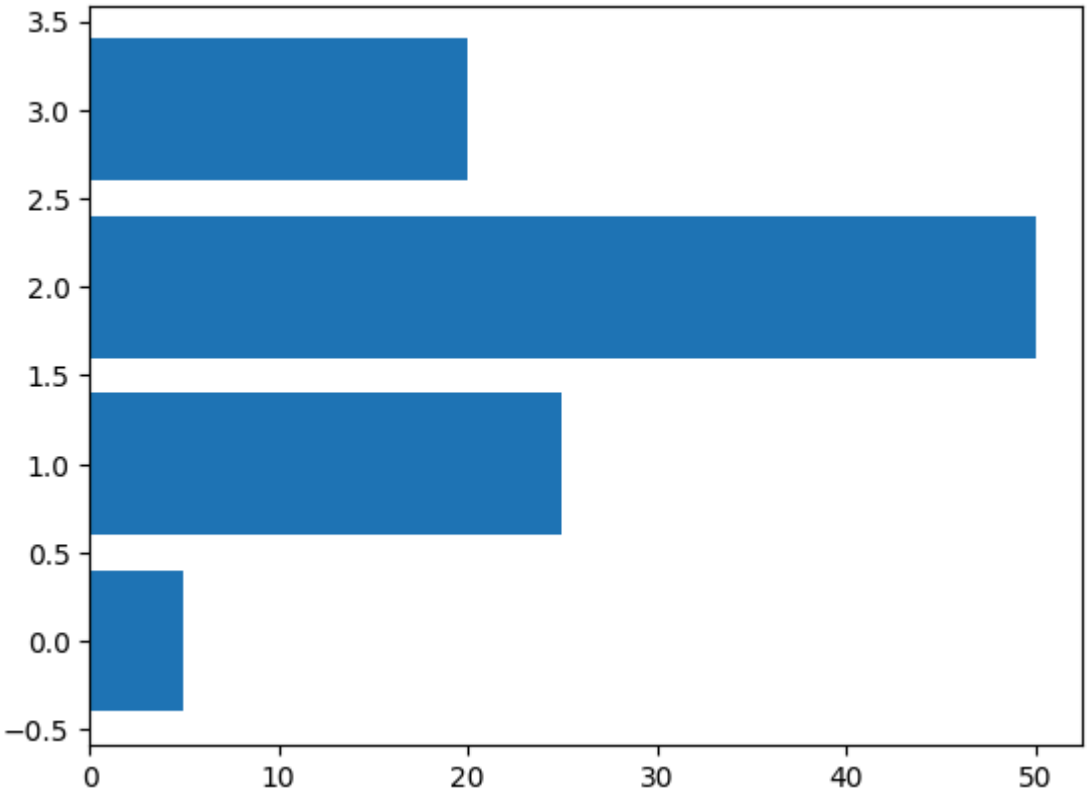
In [49]:
```
data2 = [5. , 25. , 50. , 20.]  #Data values for the heights of the bars.

plt.bar(range(len(data2)), data2) #range(len(data2)) → generates X positions [0,

plt.show()
```



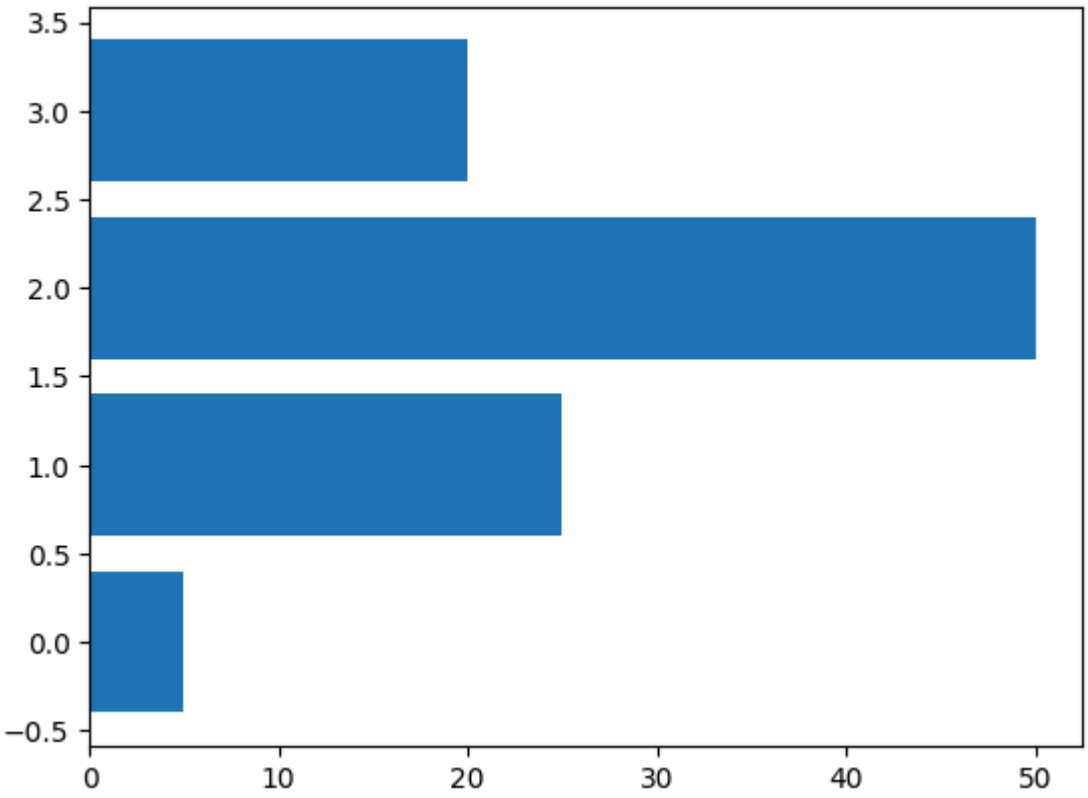
Horizontal Bar Chart

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

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

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

plt.show()
```



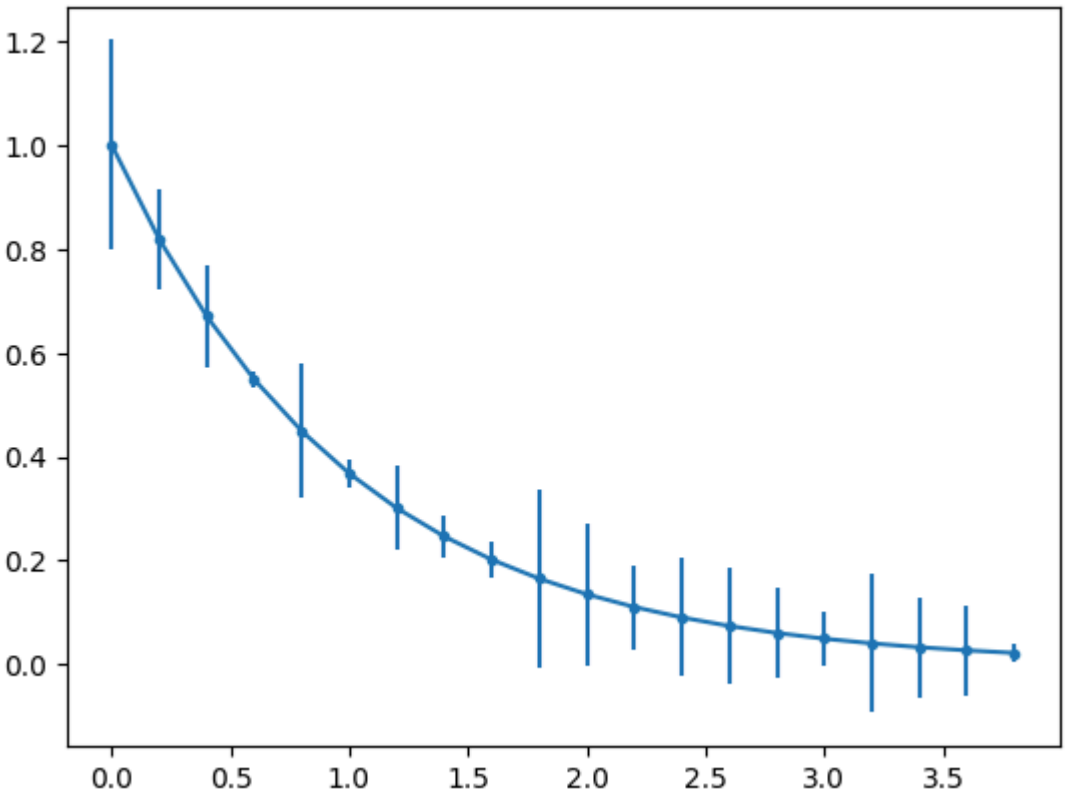
In [51]:
```
data2 = [5 , 25 , 50 , 20]

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

plt.show()
```



Error Bar Chart

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

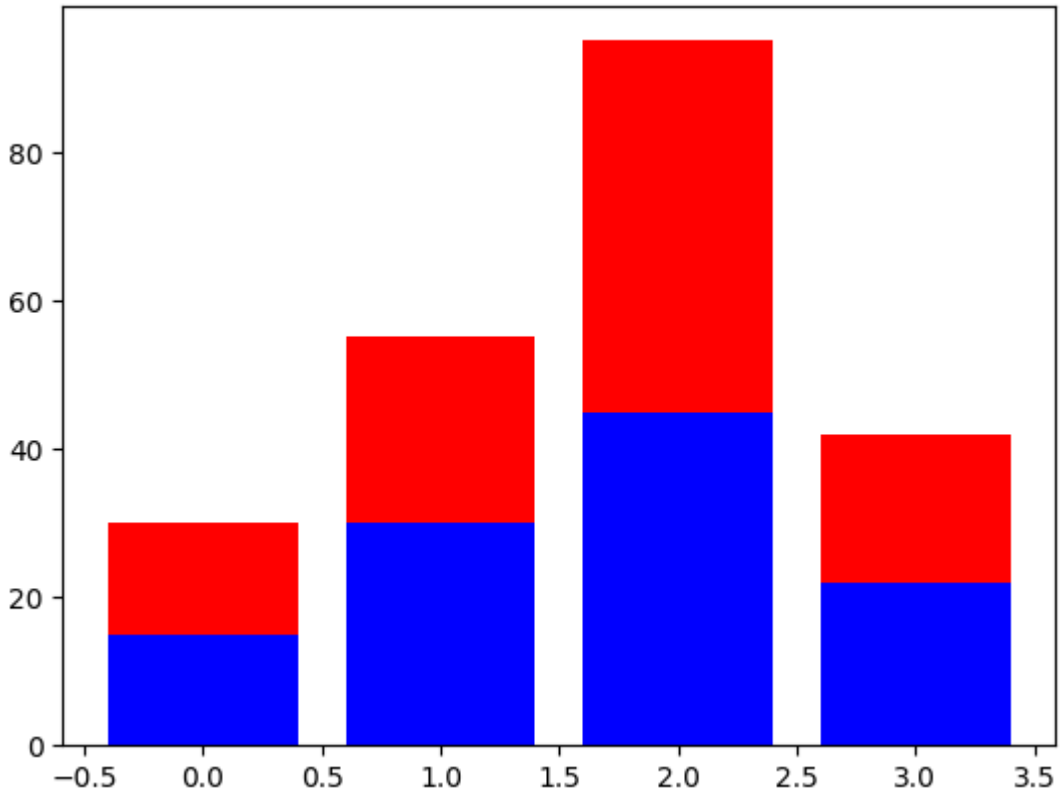
In [52]:
```
x9 = np.arange(0, 4, 0.2)# Creates x values from 0 to 4 in steps of 0.
y9 = np.exp(-x9) #Calculates exponential decay for each x value
e1 = 0.1 * np.abs(np.random.randn(len(y9)))#Generates random positive error valu
plt.errorbar(x9, y9, yerr = e1, fmt = '.-') #Plots y9 vs x9 with error bars (±e1
plt.show();
```



Stacked Bar Chart

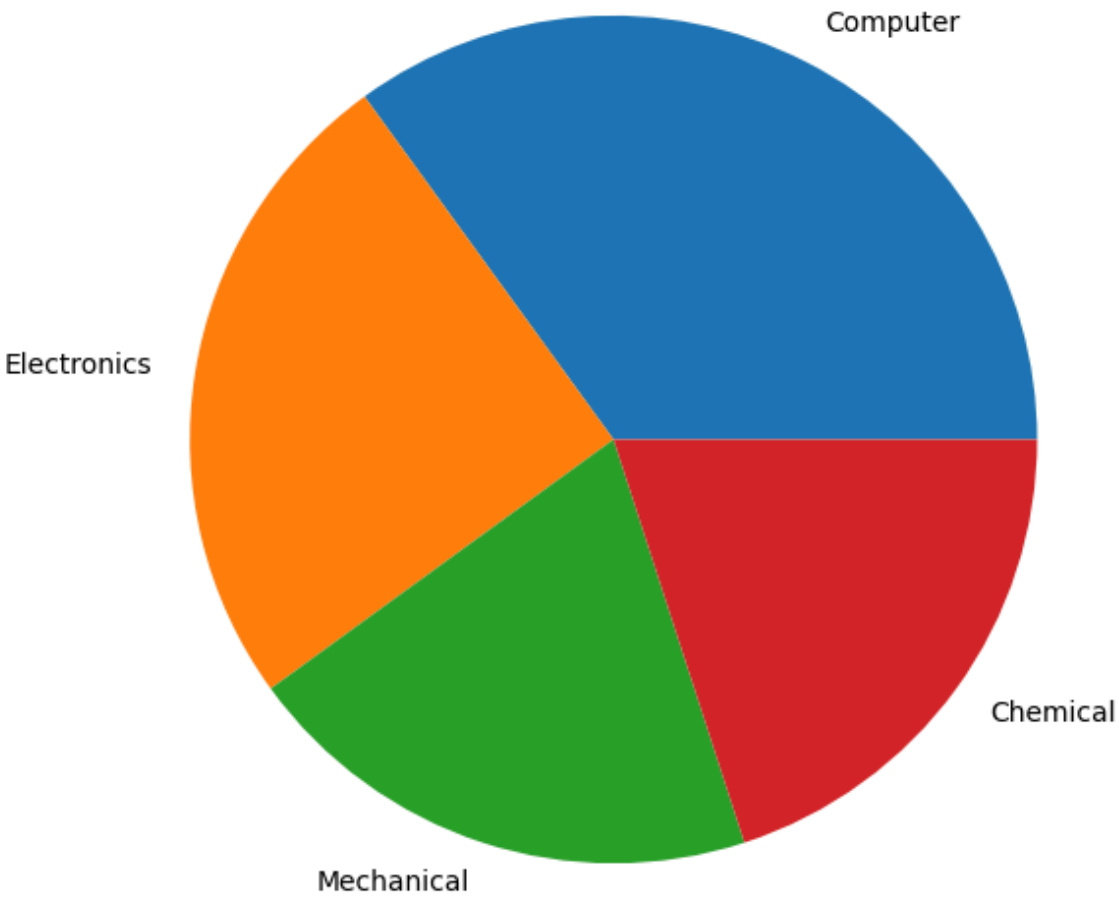
We can draw stacked bar chart by using a special parameter called bottom from the plt.bar() function. It can be done as follows:-

In [54]:
```
A = [15., 30., 45., 22.] #  First set of values
B = [15., 25., 50., 20.] # Second set of values
z2 = range(4) # Positions of the bars (0, 1, 2, 3)
plt.bar(z2, A, color = 'b') # Positions of the bars (0, 1, 2, 3)
plt.bar(z2, B, color = 'r', bottom = A) # Second set of bars (red), stacked on t
plt.show()
```



Pie Chart

In [56]:
```
plt.figure(figsize=(7,7)) #Sets the figure size to 7x7 inches, making it a squar
x10 = [35, 25, 20, 20] #Contains the values for each category.
labels = ['Computer', 'Electronics', 'Mechanical', 'Chemical'] #Names for each s
plt.pie(x10, labels=labels); #Draws the pie chart with the given values and labe
plt.show()
```

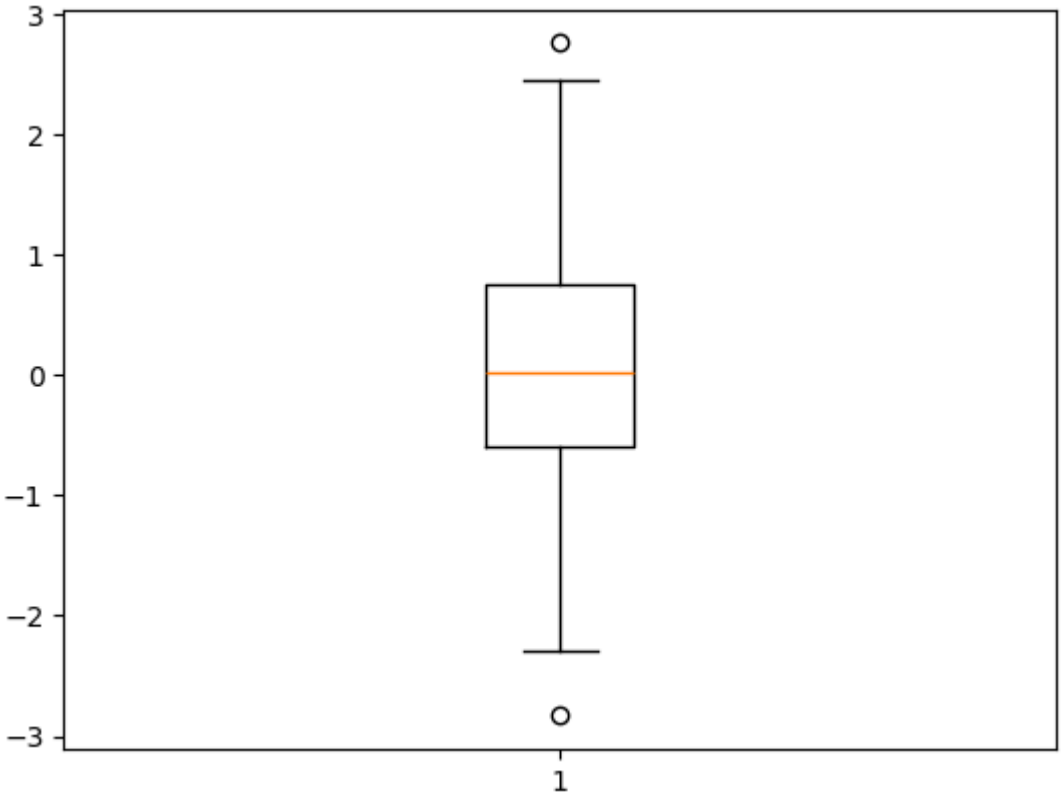


Boxplot

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 [59]:
```
data3 = np.random.randn(100) #creates 100 random numbers from a normal distribut

plt.boxplot(data3)

plt.show();
```

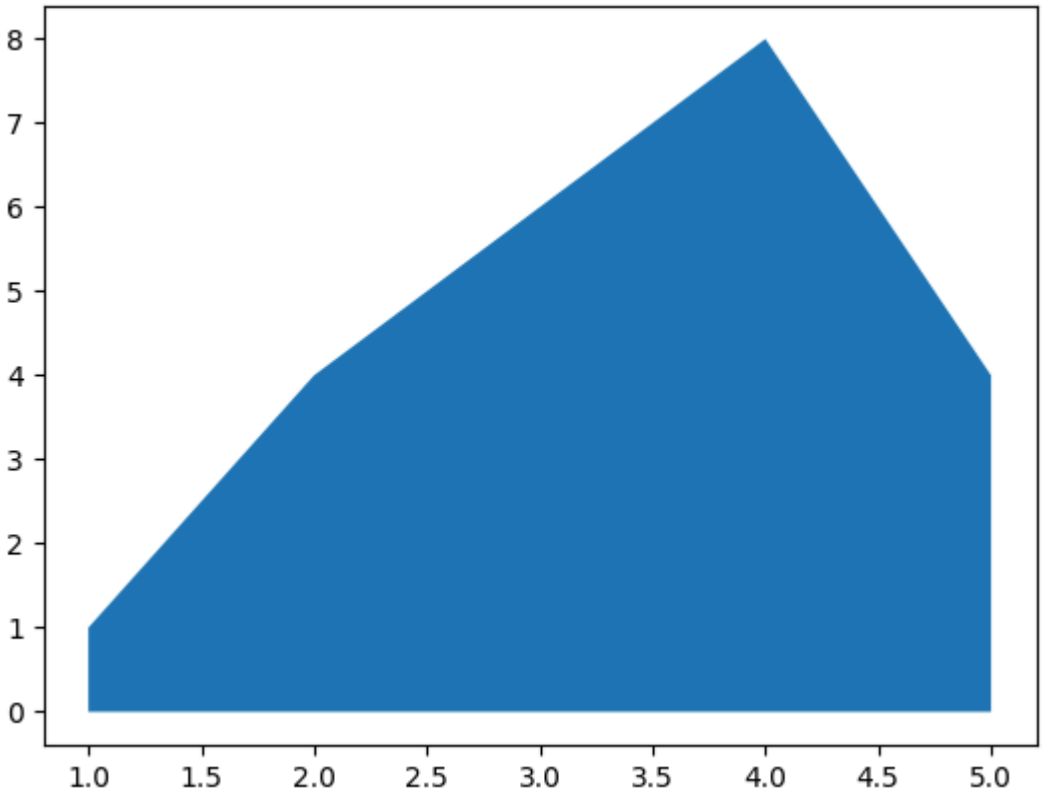


Area Chart

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 a numerical variable following another numerical variable.

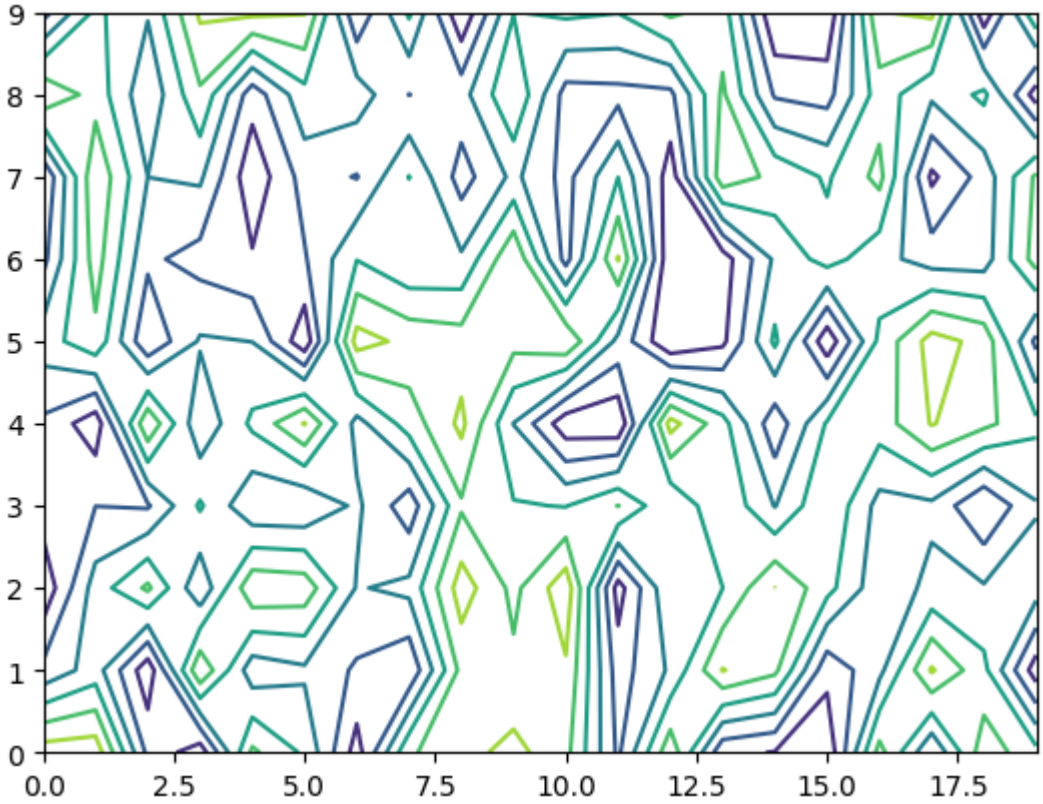
We can create an Area Chart as follows:-

In [60]:
```
# Create some data
x12 = range(1, 6)
y12 = [1, 4, 6, 8, 4]
# Area plot
plt.fill_between(x12, y12) #shades the area between the line defined by (x12, y1
plt.show()
```



Contour Plot

In [63]:
```
# Create a matrix
matrix1 = np.random.rand(10, 20) #creates a 2D array of random values between 0

cp = plt.contour(matrix1)#draws contour lines (like elevation lines on a map) co
plt.show()
```



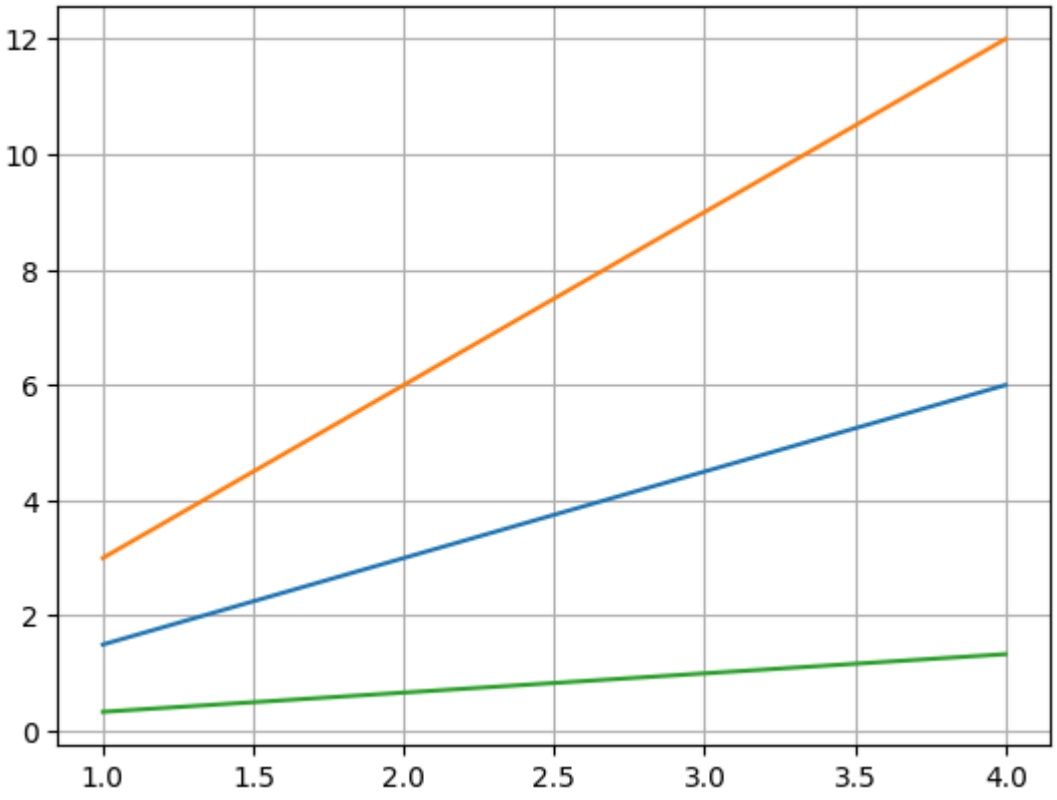
Styles with Matplotlib Plots

In [64]:
```
# 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']
```

Adding a grid

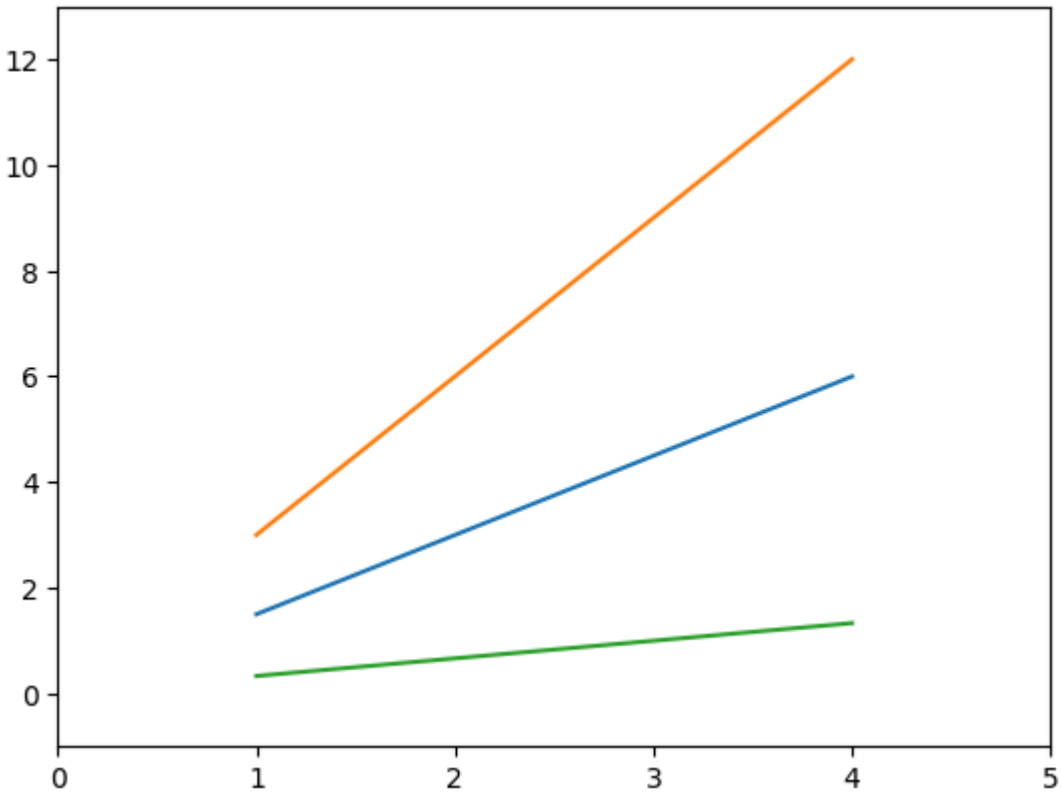
In [66]:
```
x15 = np.arange(1, 5)
plt.plot(x15, x15*1.5, x15, x15*3.0, x15, x15/3.0) #x15 vs x15 * 1.5 → slope 1.5
plt.grid(True)
plt.show()
```



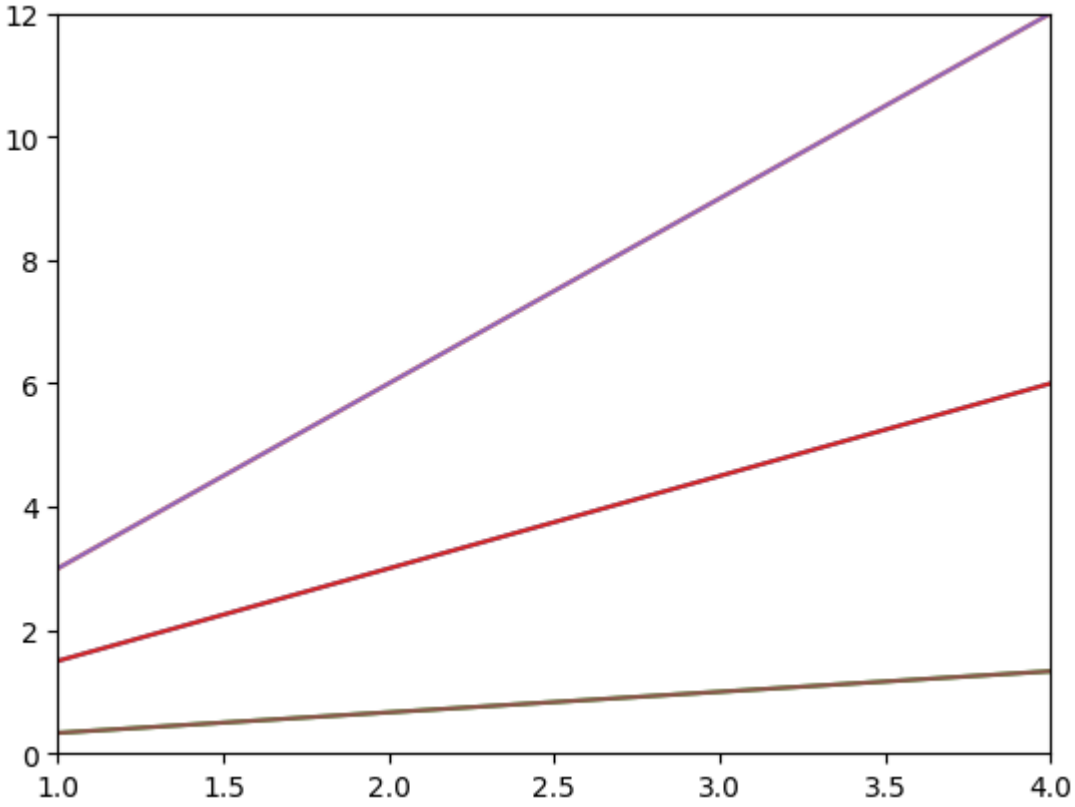
Handling axes

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 [68]:
```
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]) #  # Manually set limits: x from 0 to 5, y from -1 to 1
plt.show()
```

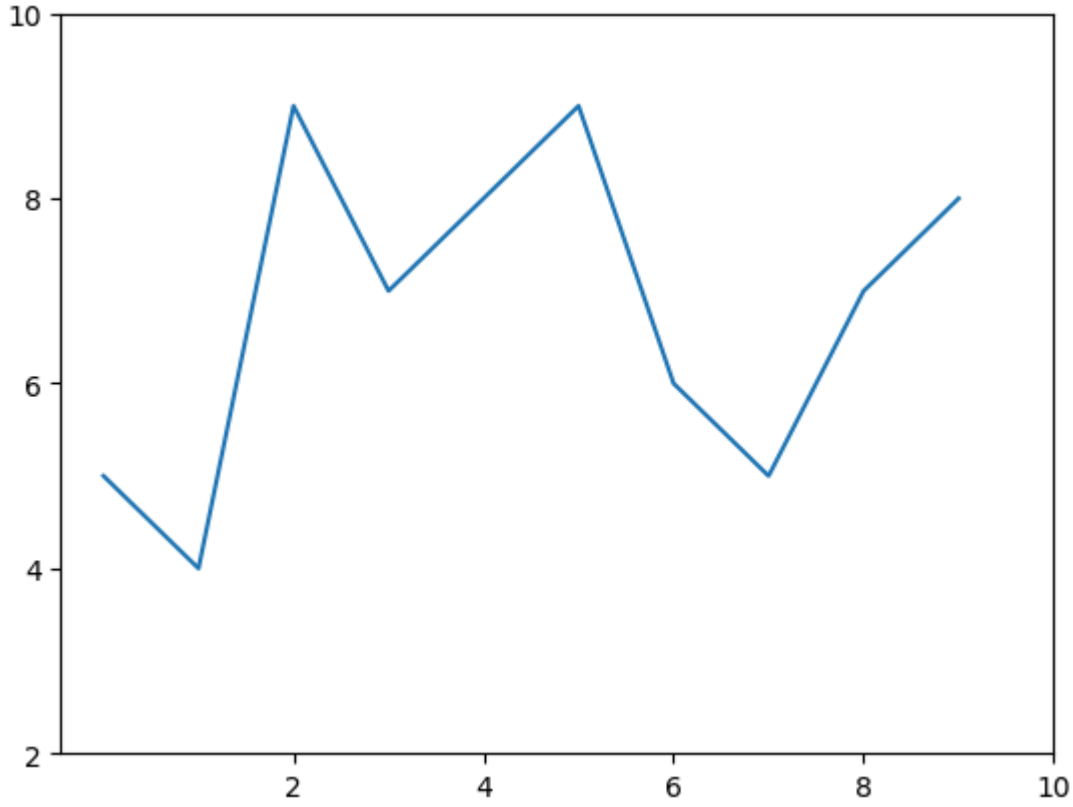


In [70]:
```
x15 = np.arange(1, 5)

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])
plt.show()
```



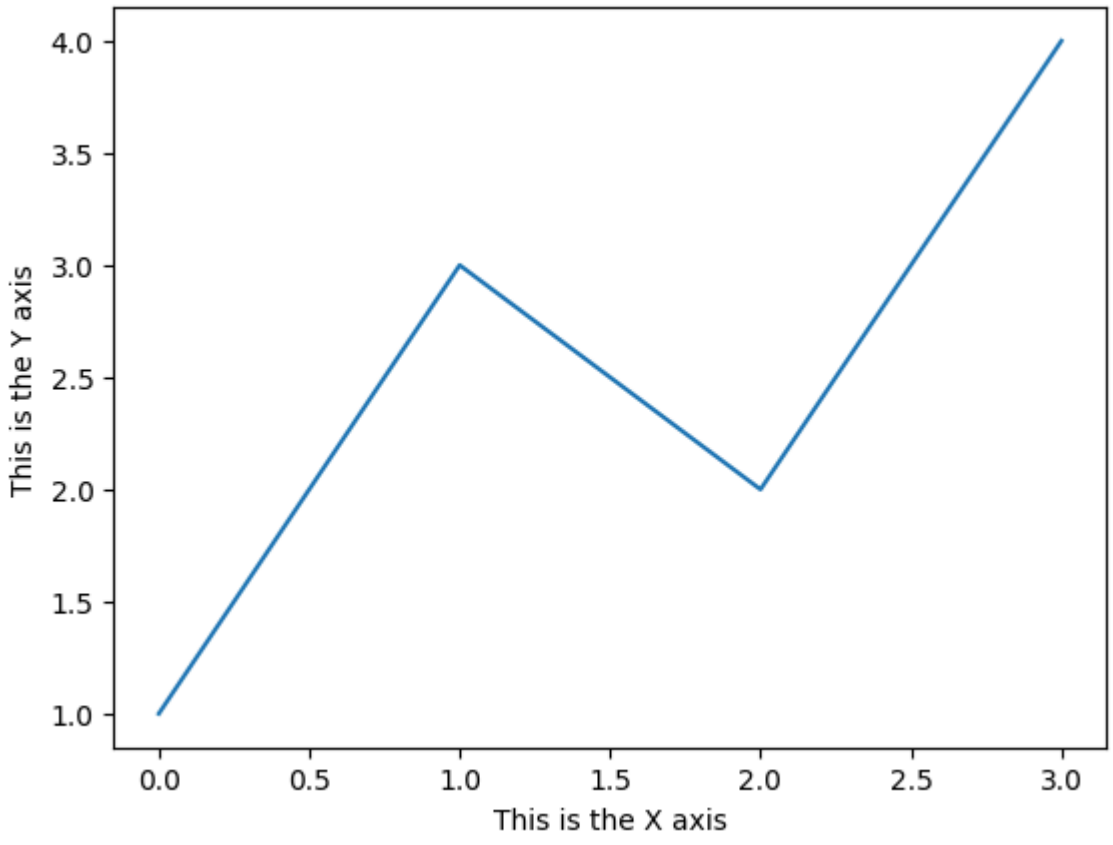
Handling X and Y ticks

In [73]:
```
u = [5, 4, 9, 7, 8, 9, 6, 5, 7, 8]  #lots the values of u against their index (0

plt.plot(u)

plt.xticks([2, 4, 6, 8, 10]) #Forces the x-axis to display ticks only at these p
#indexes go from 0 to 9, so tick 10 won't actually appear.
plt.yticks([2, 4, 6, 8, 10]) #Forces y-axis to show only these tick values.

plt.show()
```



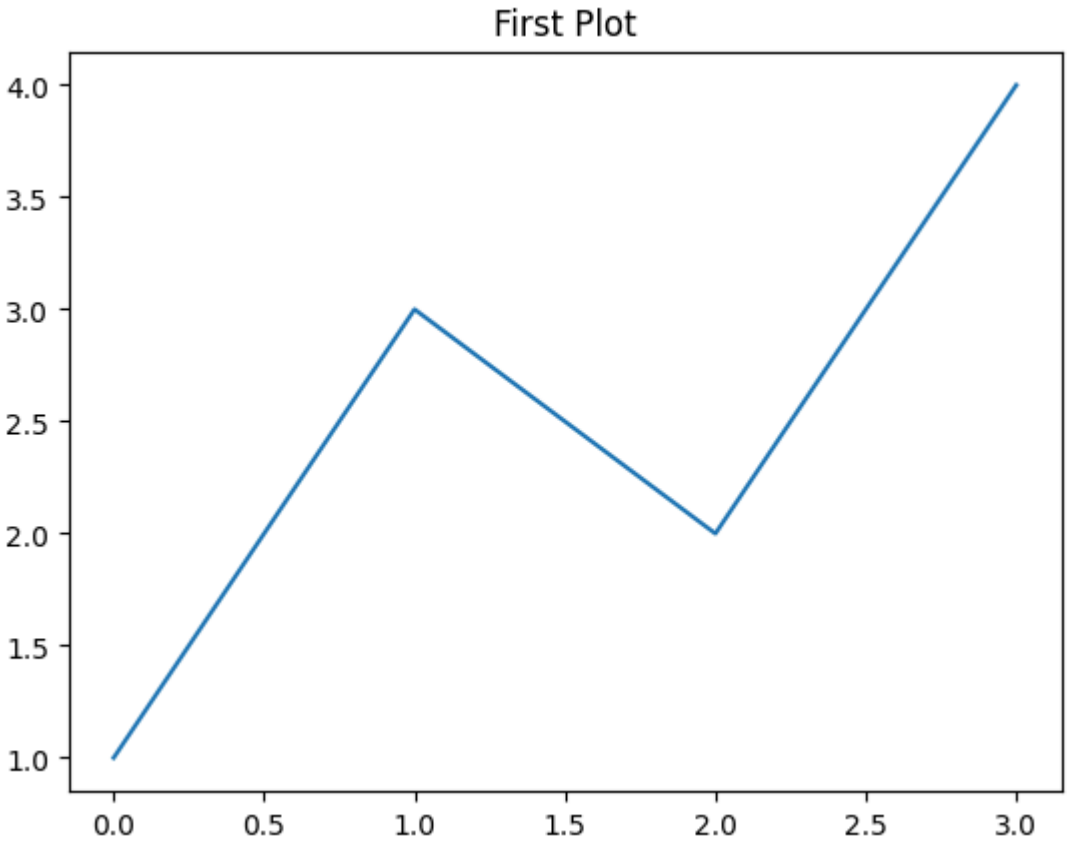
Adding labels

In [74]:
```
plt.plot([1, 3, 2, 4])
plt.xlabel('This is the X axis')
plt.ylabel('This is the Y axis')
plt.show()
```



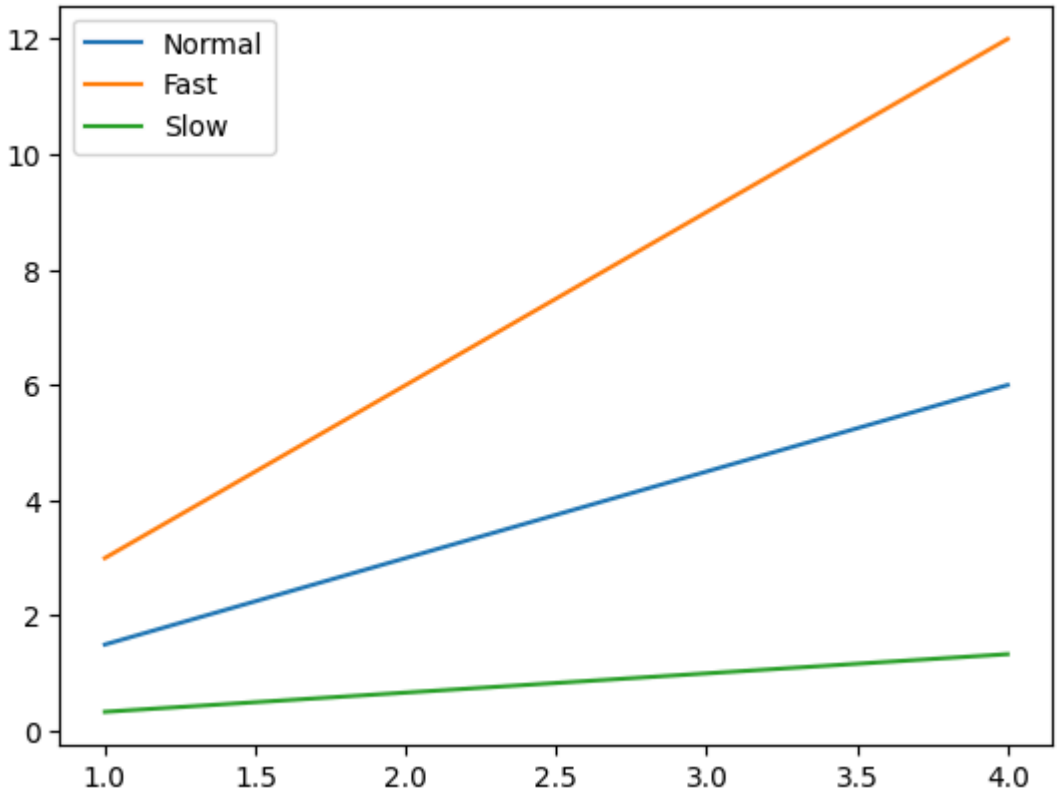
Adding a title

In [75]:
```
plt.plot([1, 3, 2, 4])
plt.title('First Plot')
plt.show()
```

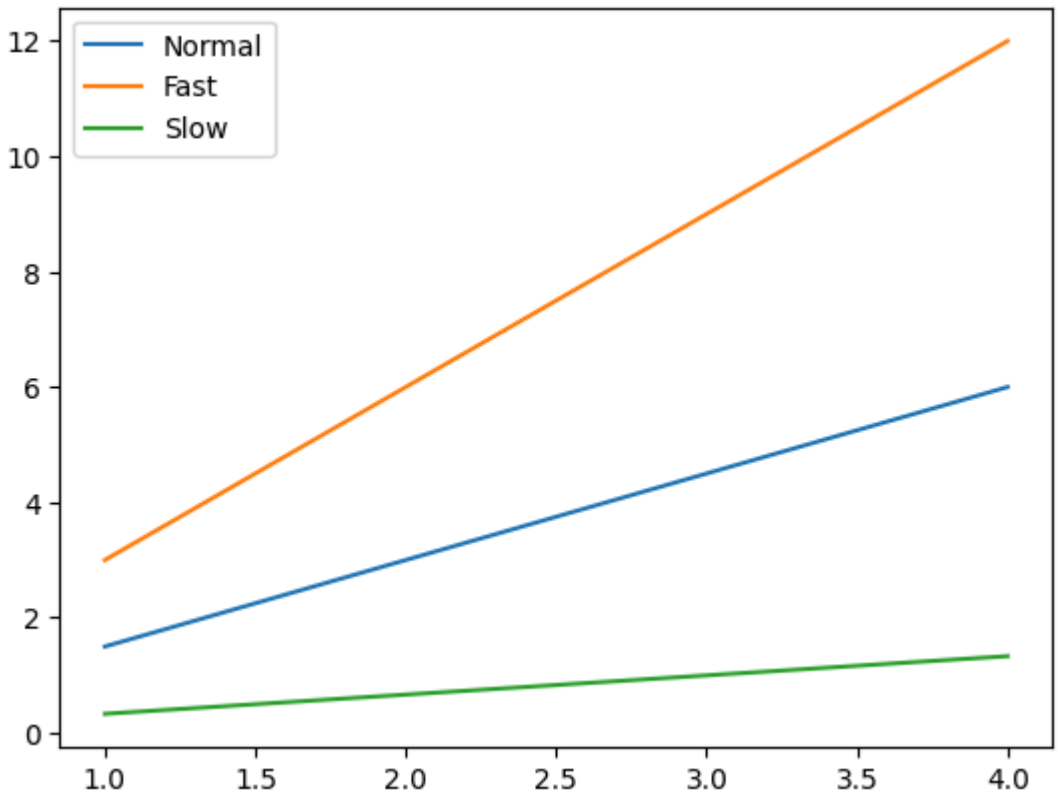


Adding a legend

In [78]:
```
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']);
plt.show()
```



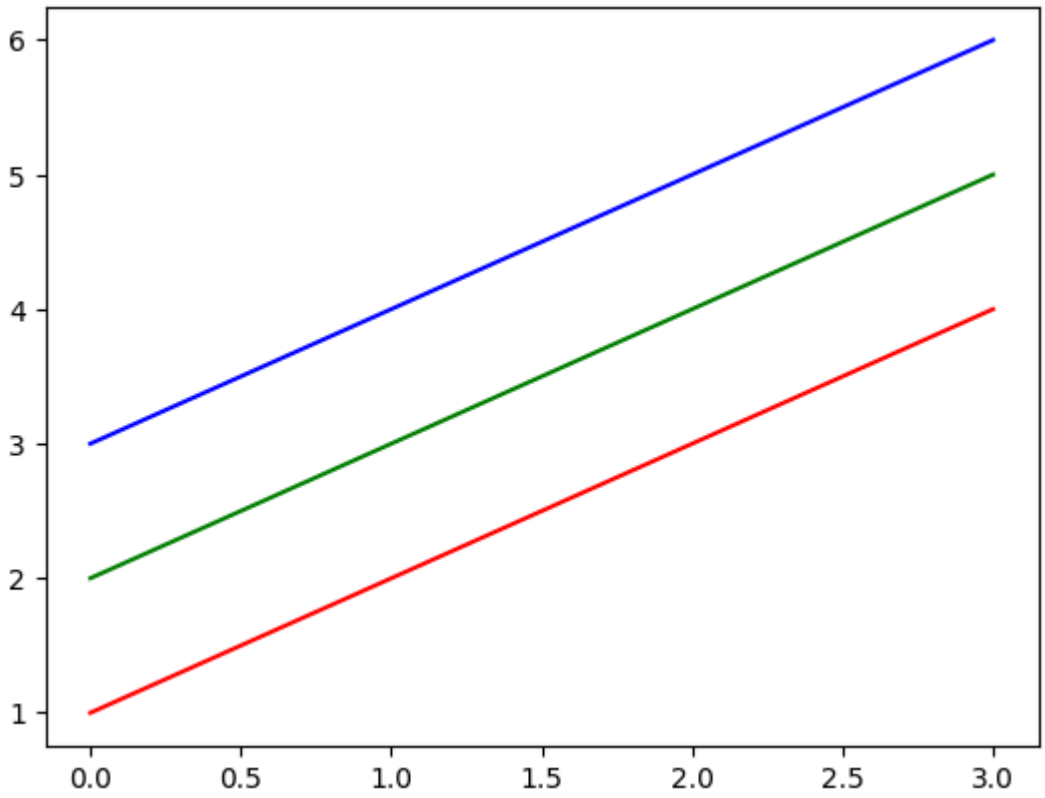
In [85]:
```
#Another method
x15 = np.arange(1, 5)
fig, ax = plt.subplots()
ax.plot(x15, x15*1.5, label='Normal')
ax.plot(x15, x15*3.0, label='Fast')
ax.plot(x15, x15/3.0, label='Slow')
ax.legend();
plt.show()
```



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 places 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 colours We can draw different lines or c

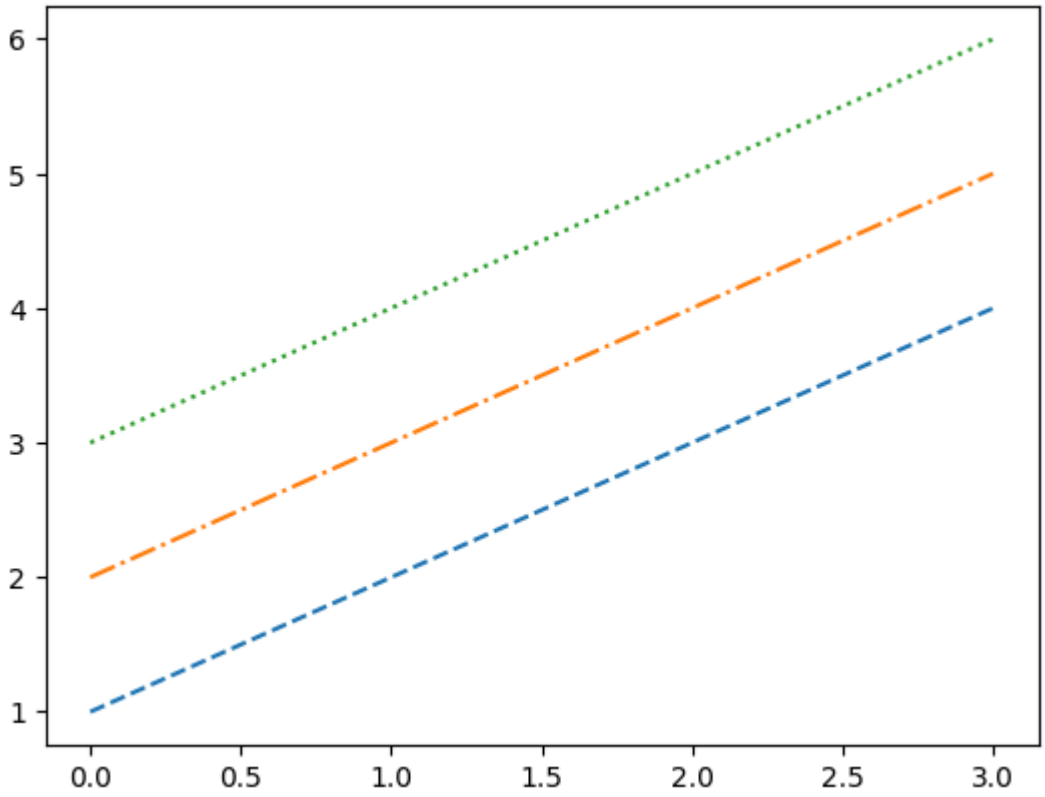
Control colours

In [86]:
```
x16 = np.arange(1, 5)
plt.plot(x16, 'r') #red line ('r') with values [1, 2, 3, 4]
plt.plot(x16+1, 'g') #reen line ('g') with values [2, 3, 4, 5]
plt.plot(x16+2, 'b') #blue line ('b') with values [3, 4, 5, 6]
plt.show()
```



Control line styles

In [87]:
```
x16 = np.arange(1, 5)
plt.plot(x16, '--', x16+1, '-.', x16+2, ':')
plt.show()
```



In [ ]:

In [ ]: