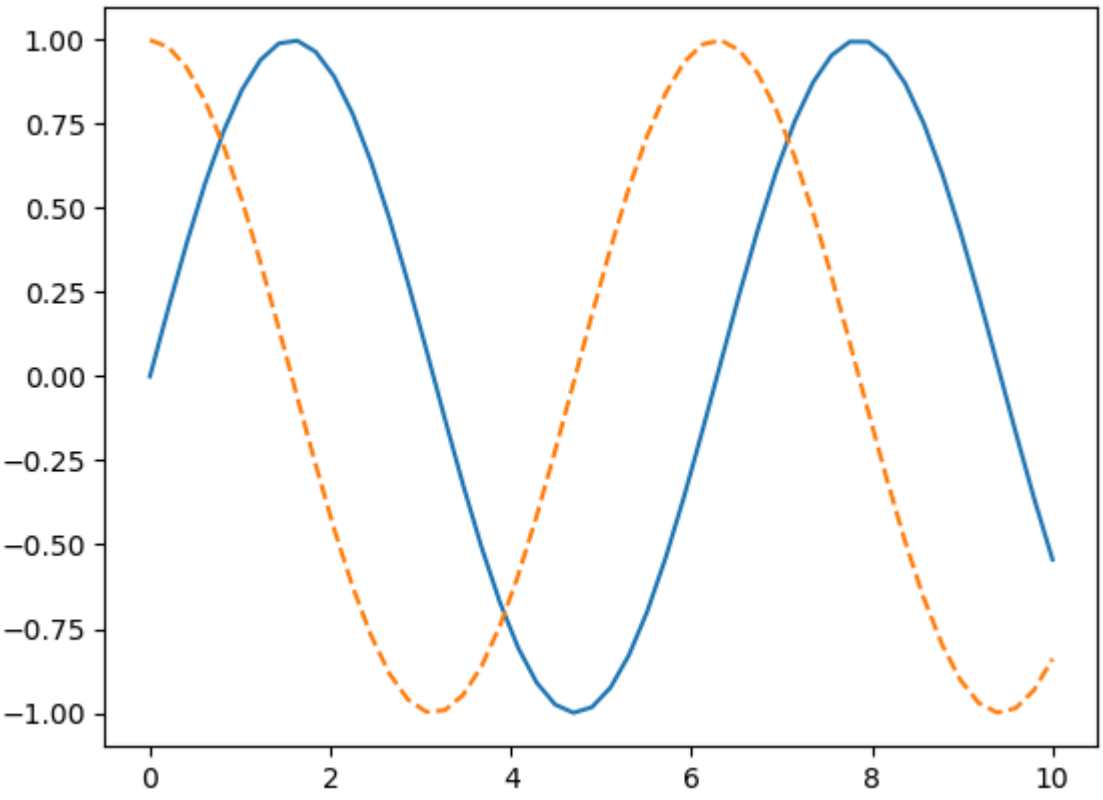
Matplotlib

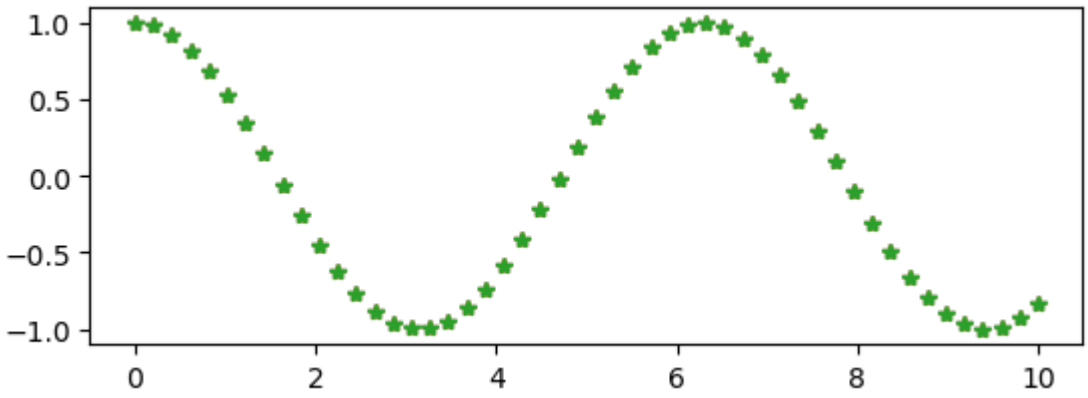
In [2]:
```
# Import dependencies
import numpy as np
import pandas as pd
```

In [3]:
```
# Import Matplotlib
import matplotlib.pyplot as plt
```

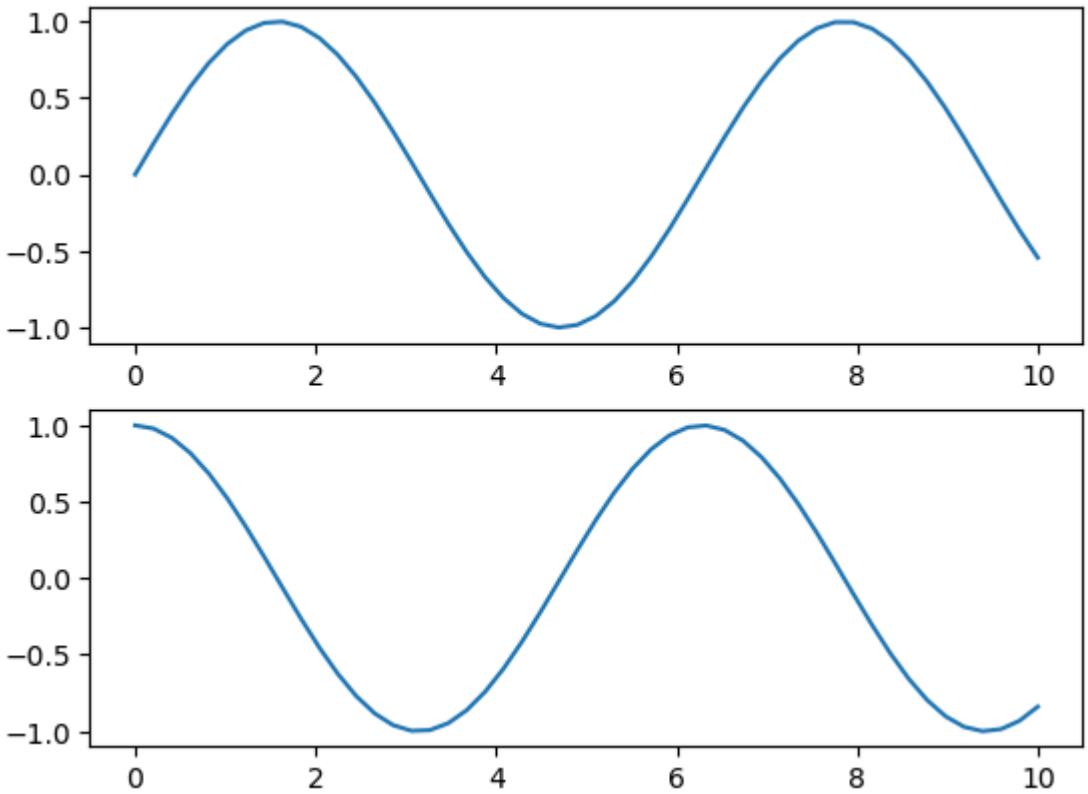
In [4]:
```
%matplotlib inline
x1 = np.linspace(0, 10, 50)

# 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 [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()
```



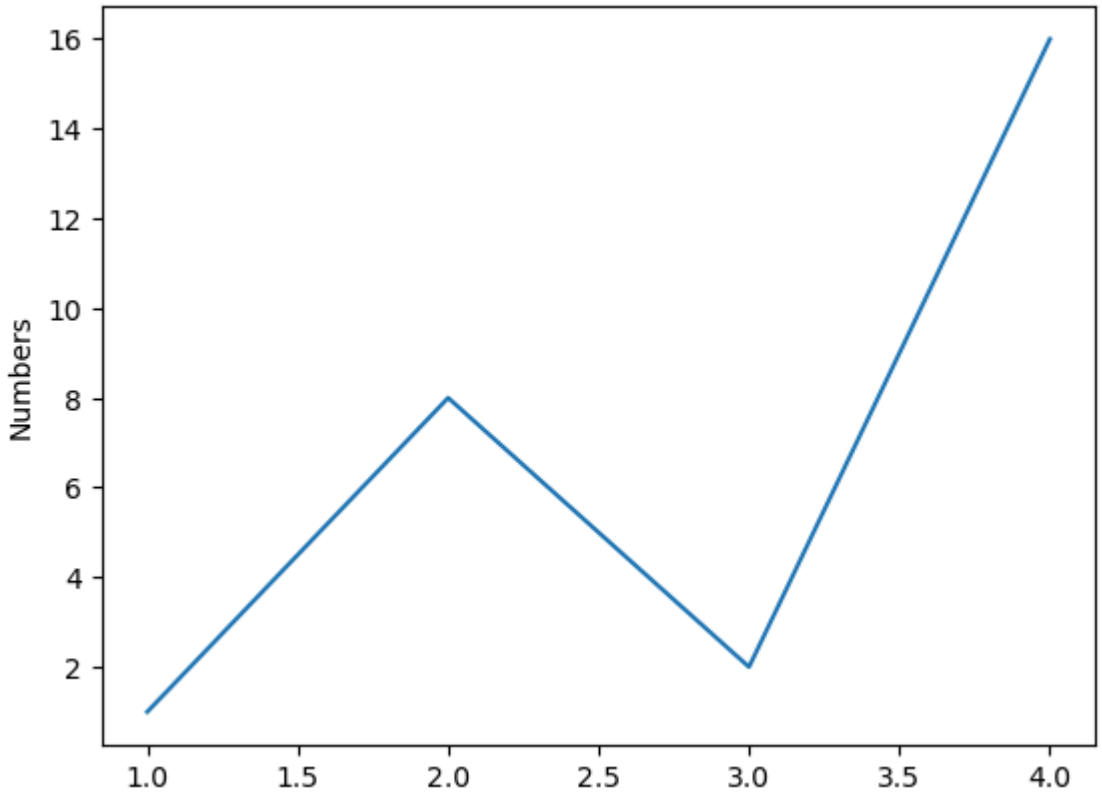
In [13]:
```
# get current figure information
print(plt.gcf())
```

```
Figure(640x480)
```

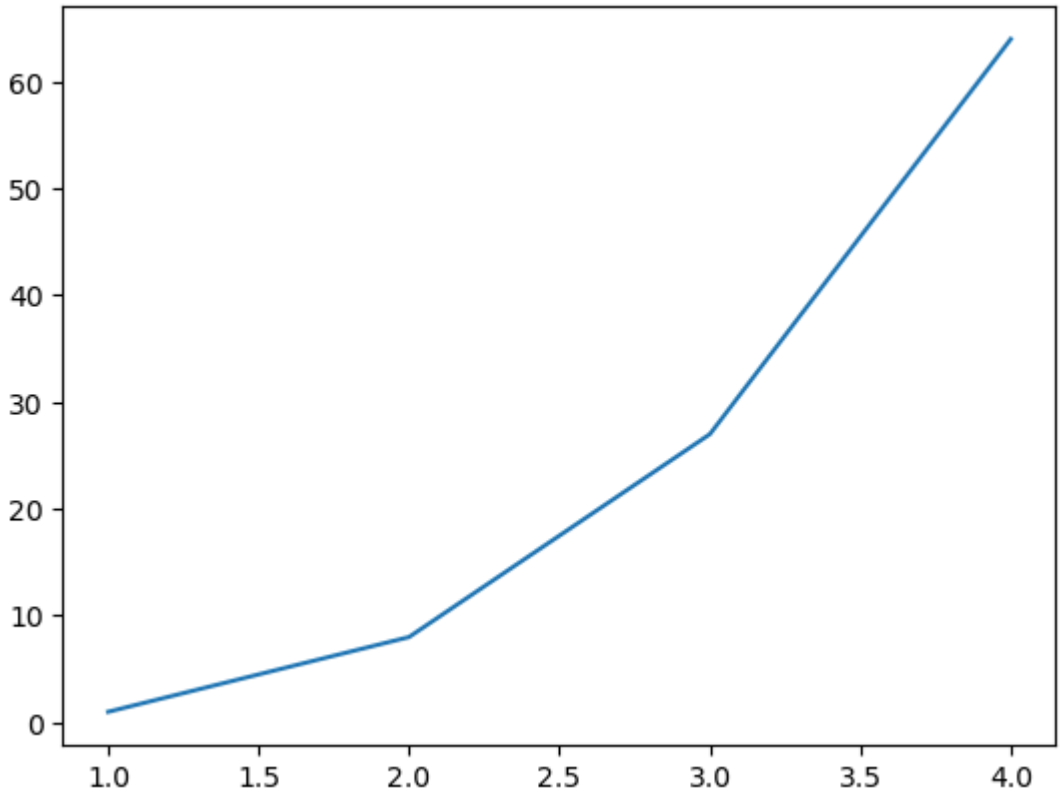
In [14]:
```
# get current axis information
print(plt.gca())
```

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

In [11]:
```
plt.plot([1,2,3,4], [1,8,2,16])
plt.ylabel('Numbers')
plt.show()
```



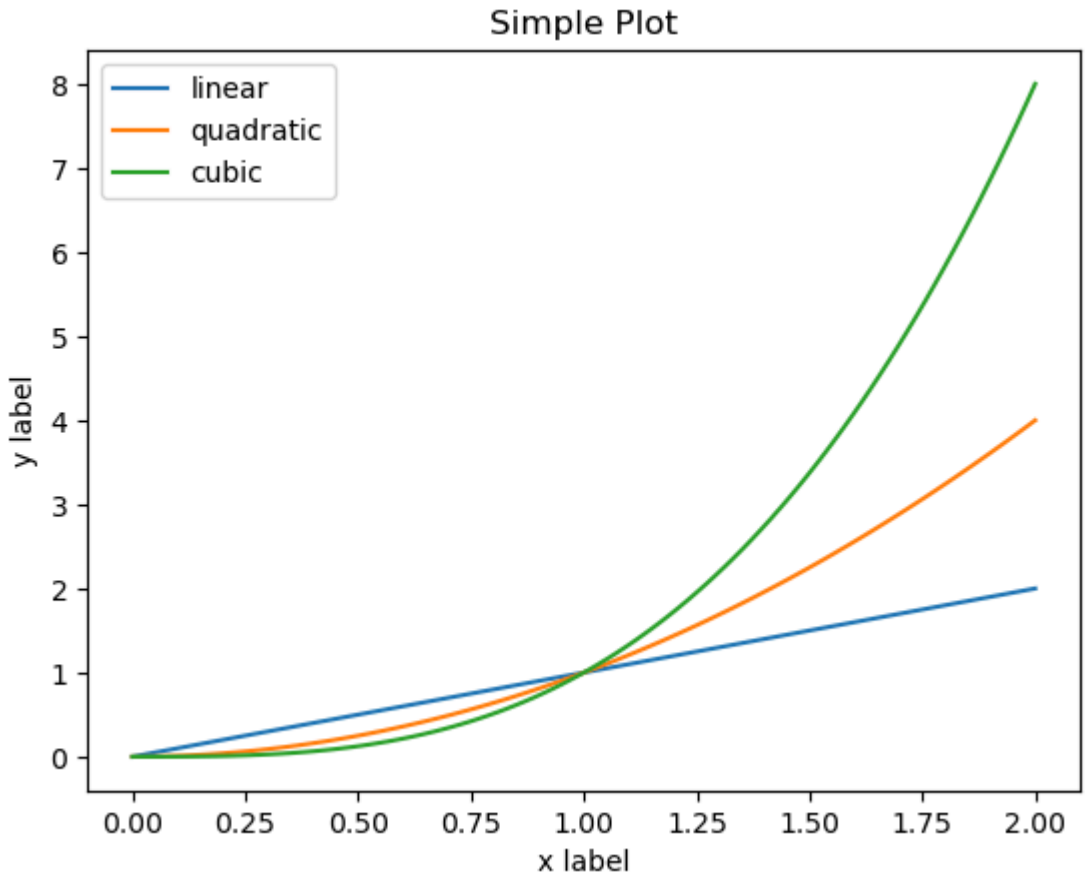
In [15]:
```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], [1, 8, 27, 64])
plt.show()
```



State-machine interface

Pyplot provides the state-machine interface 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 [16]:
```
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()
```

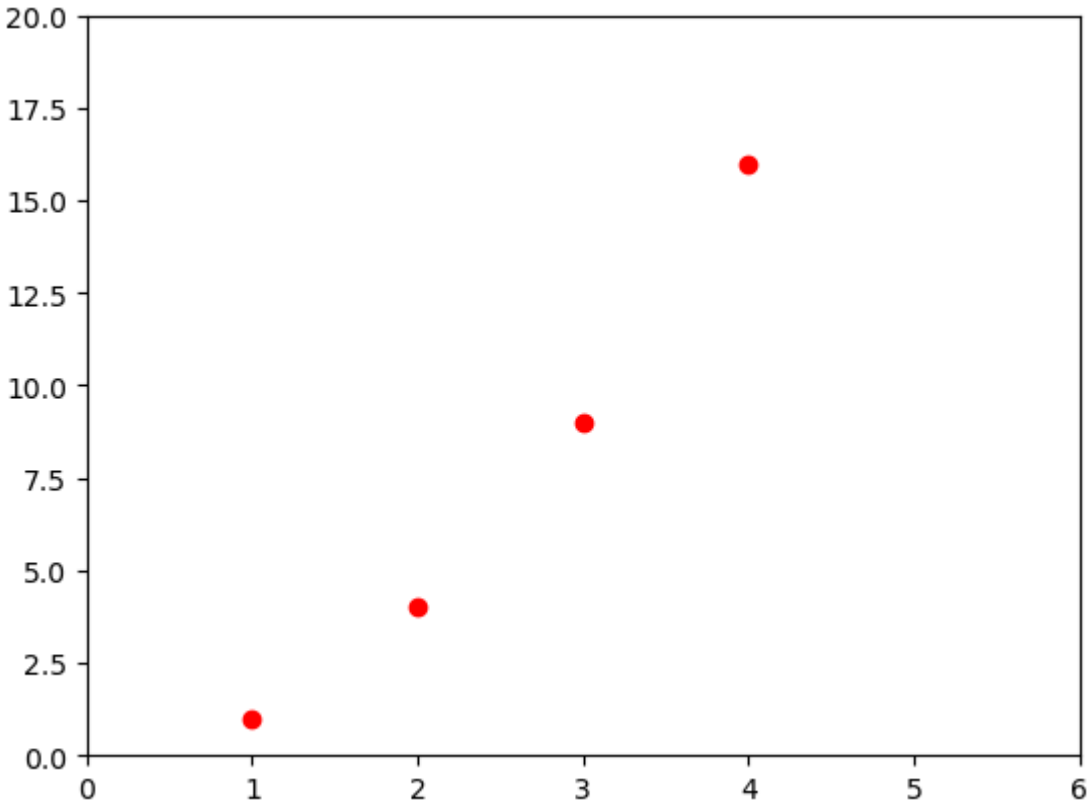


ormatting the style of plot

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:-

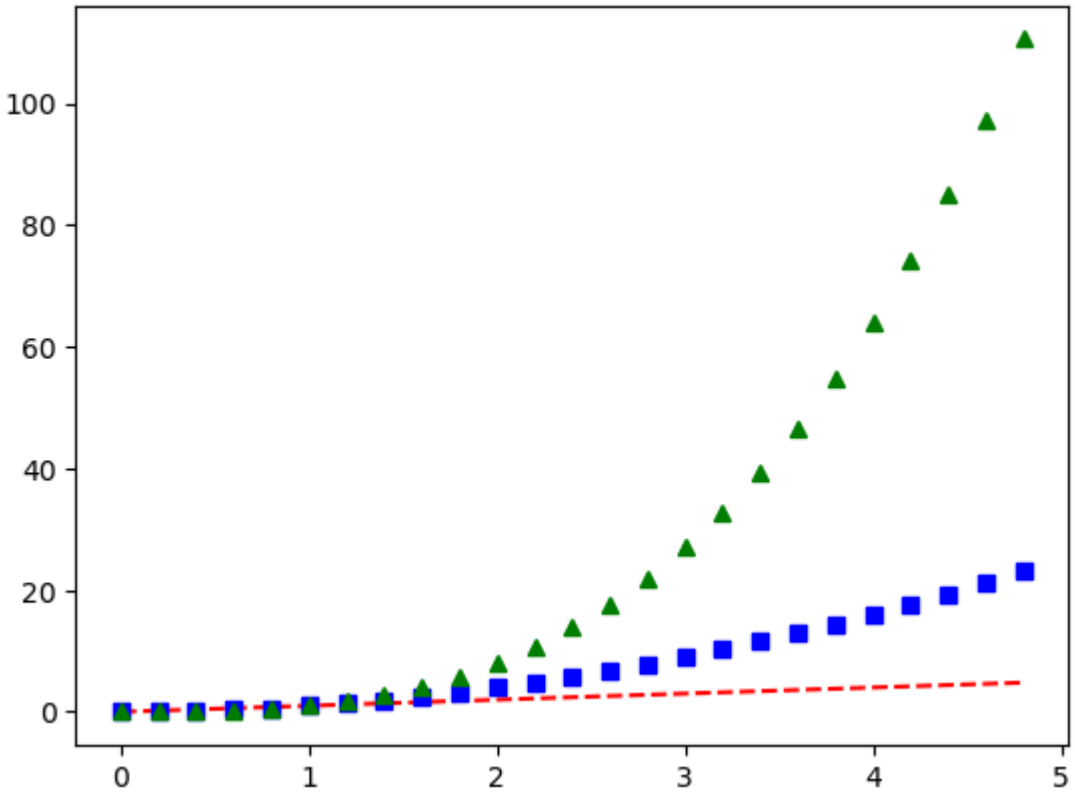
In [17]:
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```



Working with NumPy arrays

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

In [18]:
```
# 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()
```



9. Object-Oriented API

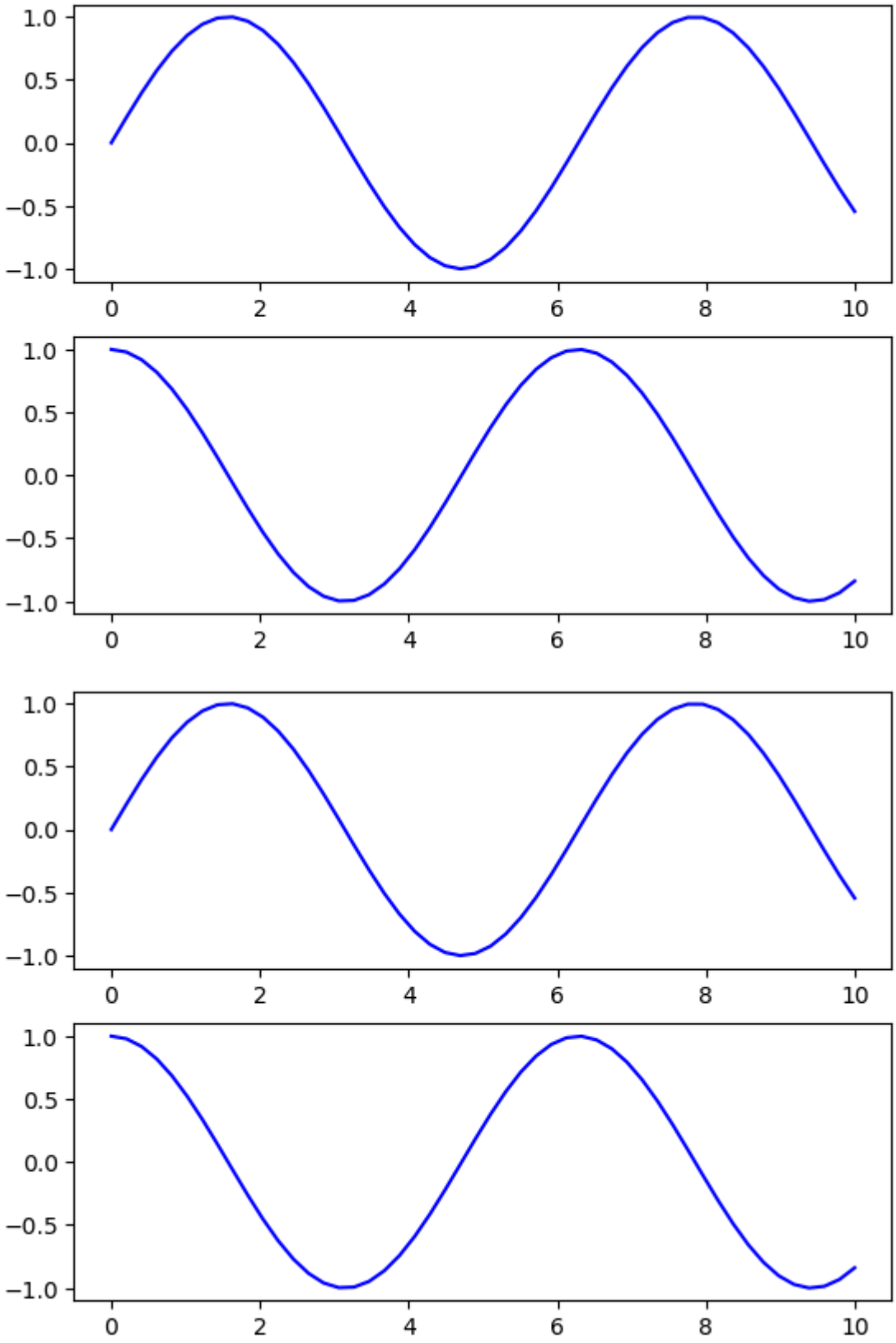
The Object-Oriented API is available for more complex plotting situations. It allows us to exercise more control over the figure. In Pyplot API, we depend on some notion of an "active" figure or axes. But, in the Object-Oriented API the plotting functions are methods of explicit Figure and Axes objects.

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 represent 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 [20]:
```
# First create 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()
```

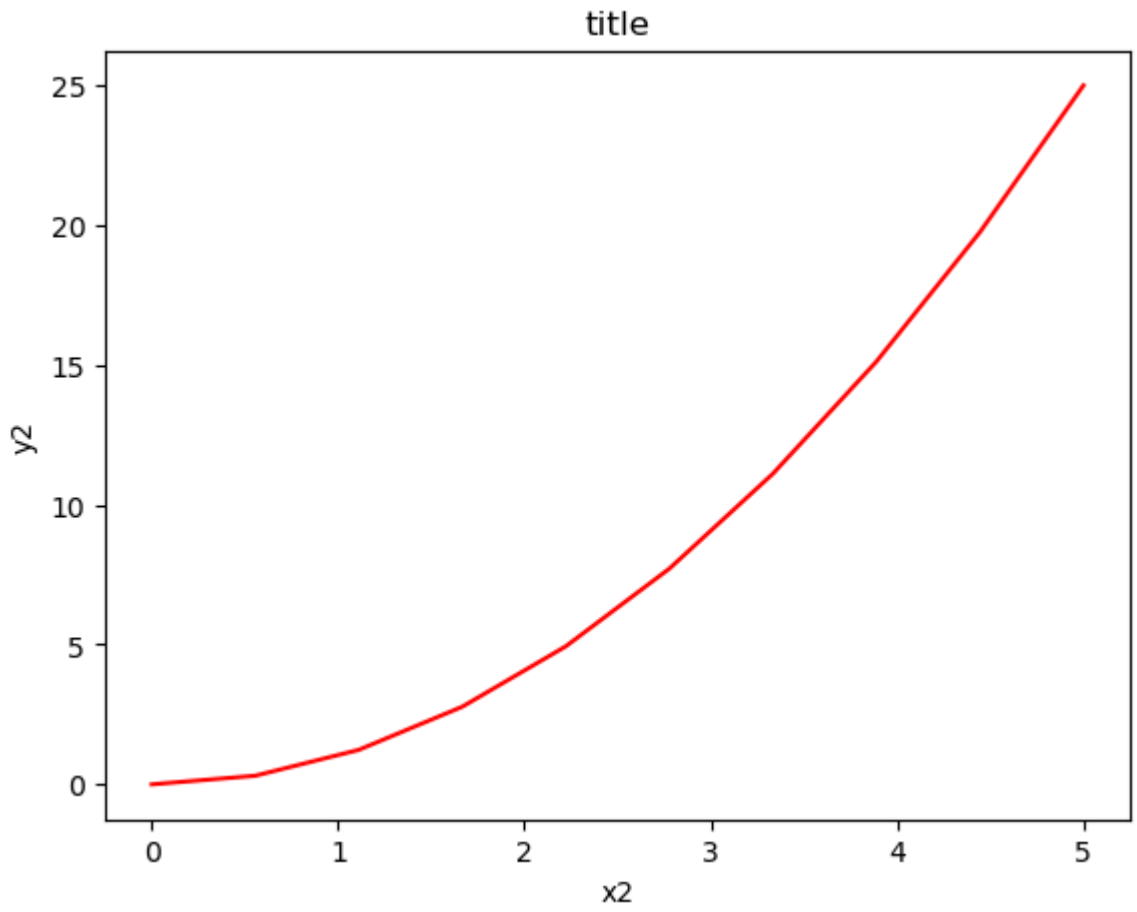


Objects and Reference

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 [22]:
```
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()
```



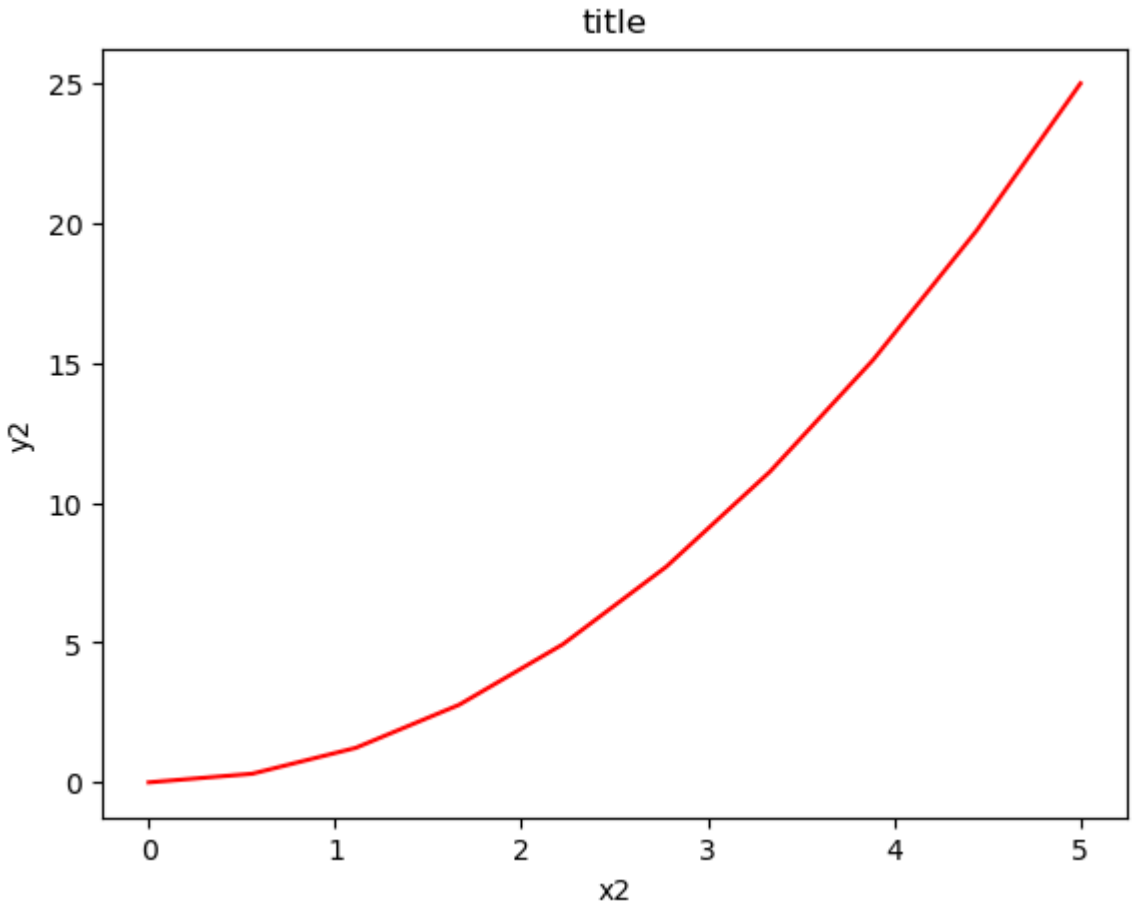


Figure and Axes

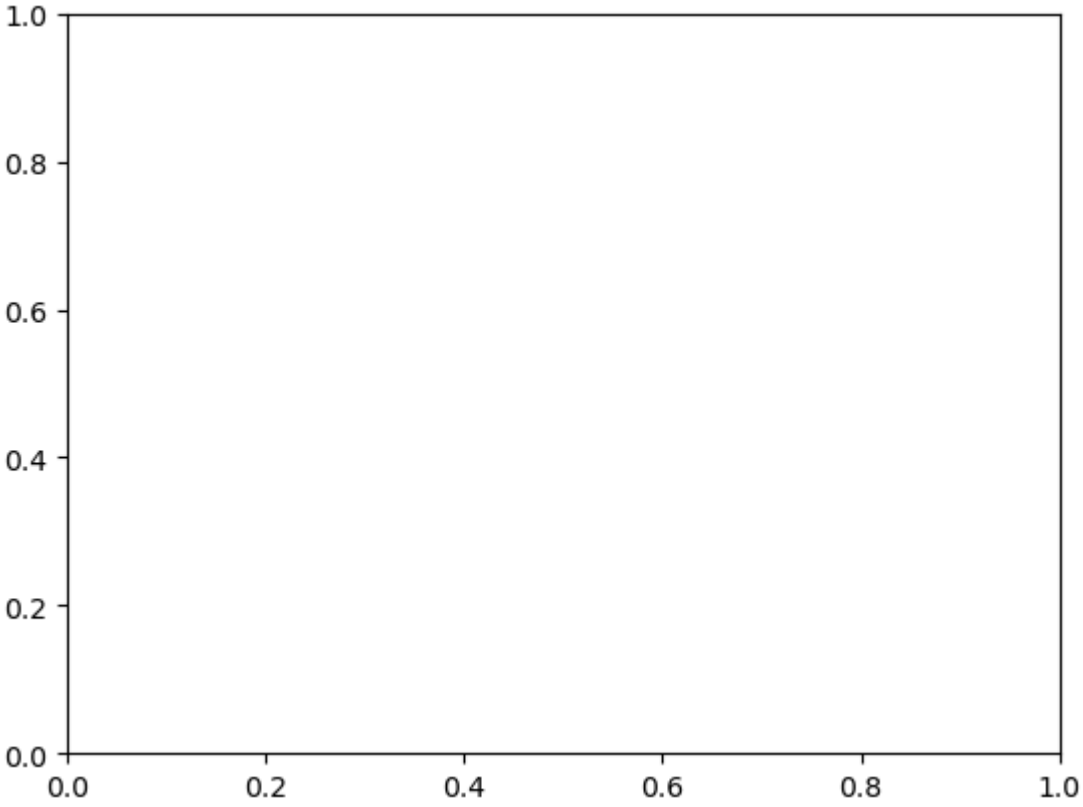
I start by creating a figure and an axes. A figure and axes can be created as follows:

fig = plt.figure()

ax = plt.axes()

In Matplotlib, the figure (an instance 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 [23]:
```
fig = plt.figure()

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



10. 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_subplot() as follows:-

ax1 = fig.add_subplot(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_subplot() commands as follows:-

ax2 = fig.add_subplot(2, 2, 2)

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

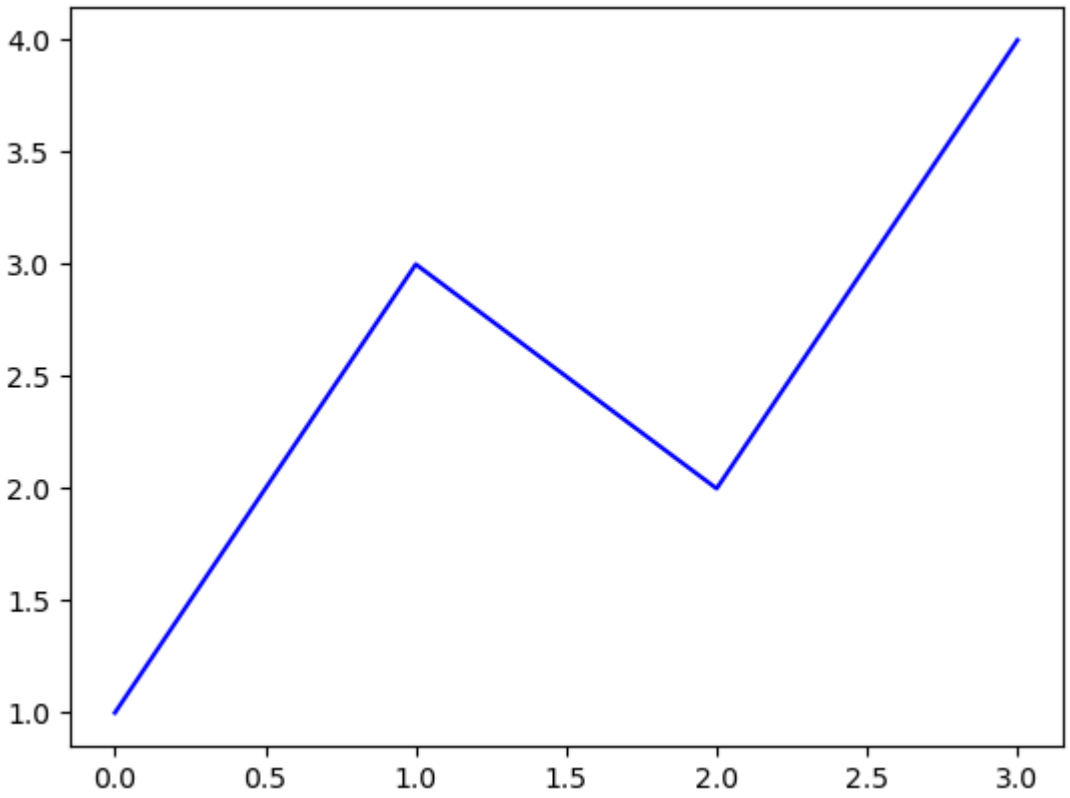
ax4 = fig.add_subplot(2, 2, 4)

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

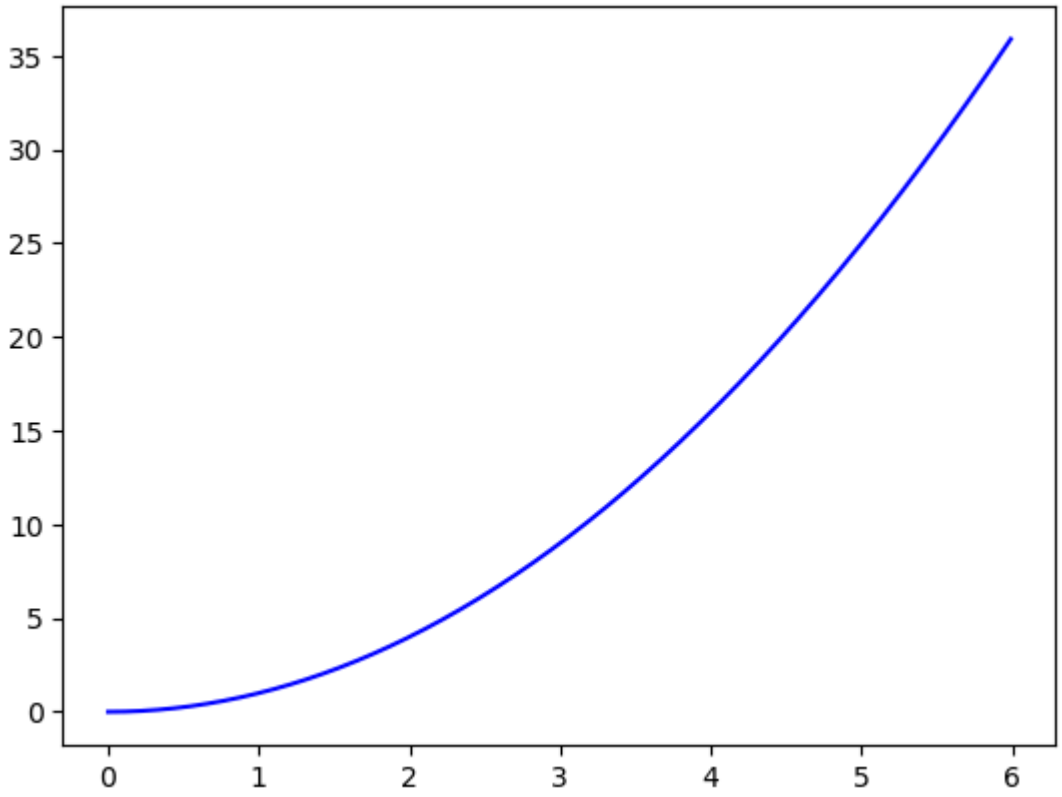
1. First plot with Matplotlib

Now, I will start producing plots. Here is the first example:-

In [25]:
```
plt.plot([1, 3, 2, 4], 'b-')

plt.show( )
```



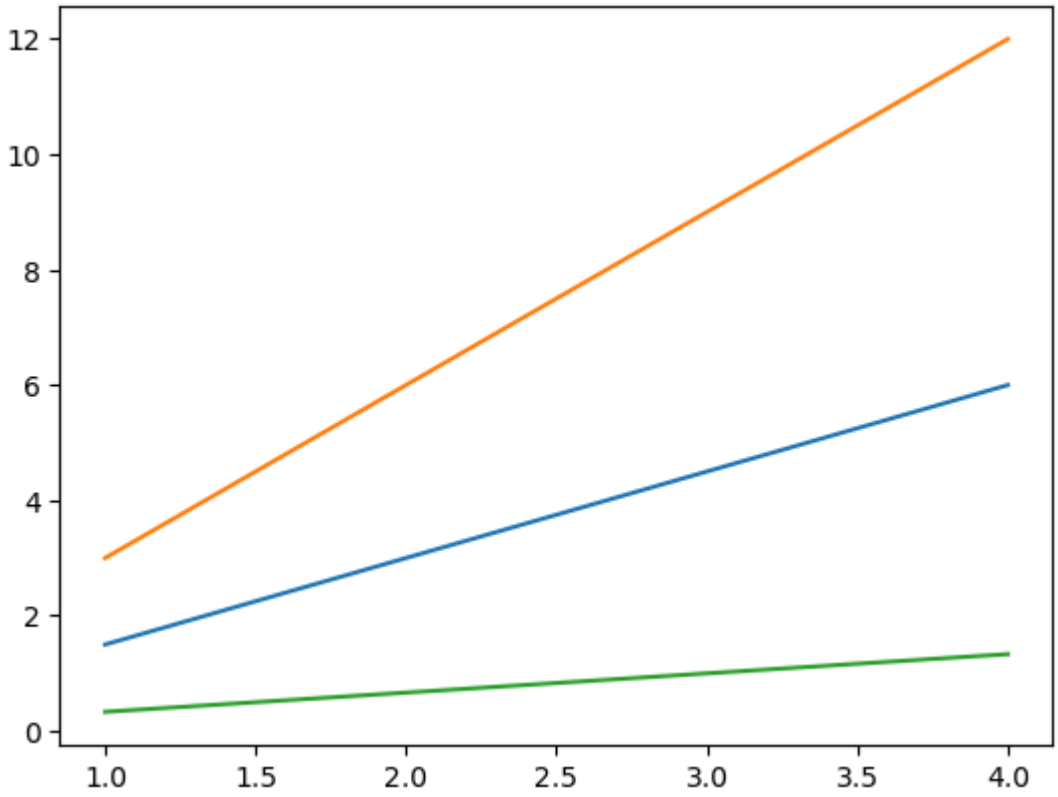
In [26]:
```
x3 = np.arange(0.0, 6.0, 0.01)

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

plt.show()
```



12. Multiline Plots

Multiline Plots mean plotting more than one plot on the same figure. We can plot more than one plot on the same figure. It can be achieved by plotting all the lines before calling show(). It can be done as follows:-

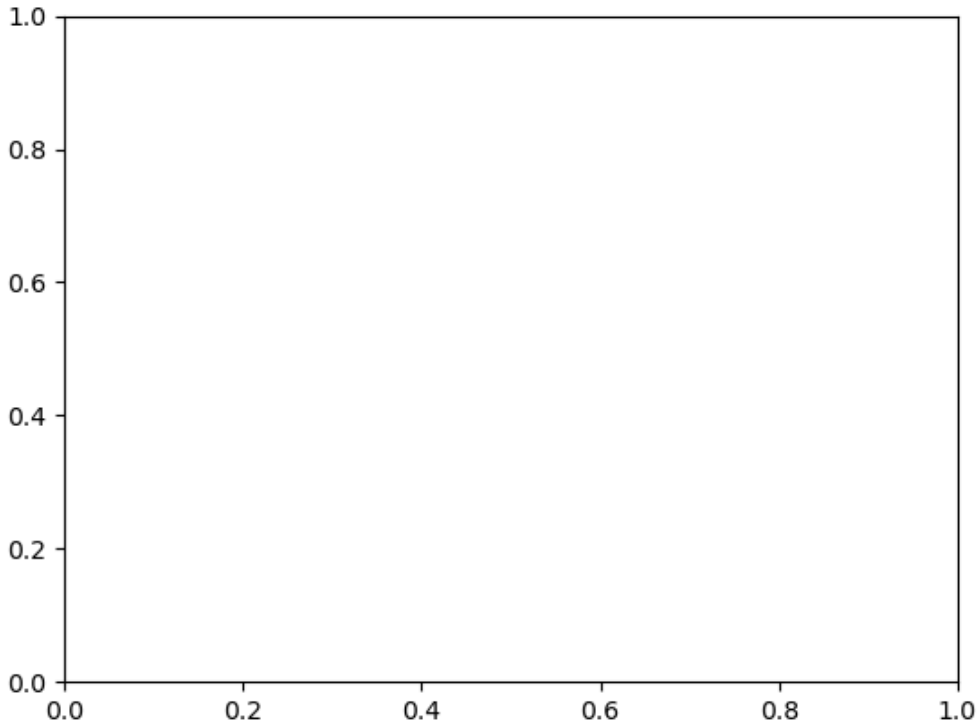
In [27]:
```
x4 = range(1, 5)

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 [28]:
```
# Saving the figure

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

In [29]:
```
# Explore the contents of figure

from IPython.display import Image

Image('plot1.png')
```

Out[29]:



In [30]:
```
# Explore supported file formats


fig.canvas.get_supported_filetypes()
```

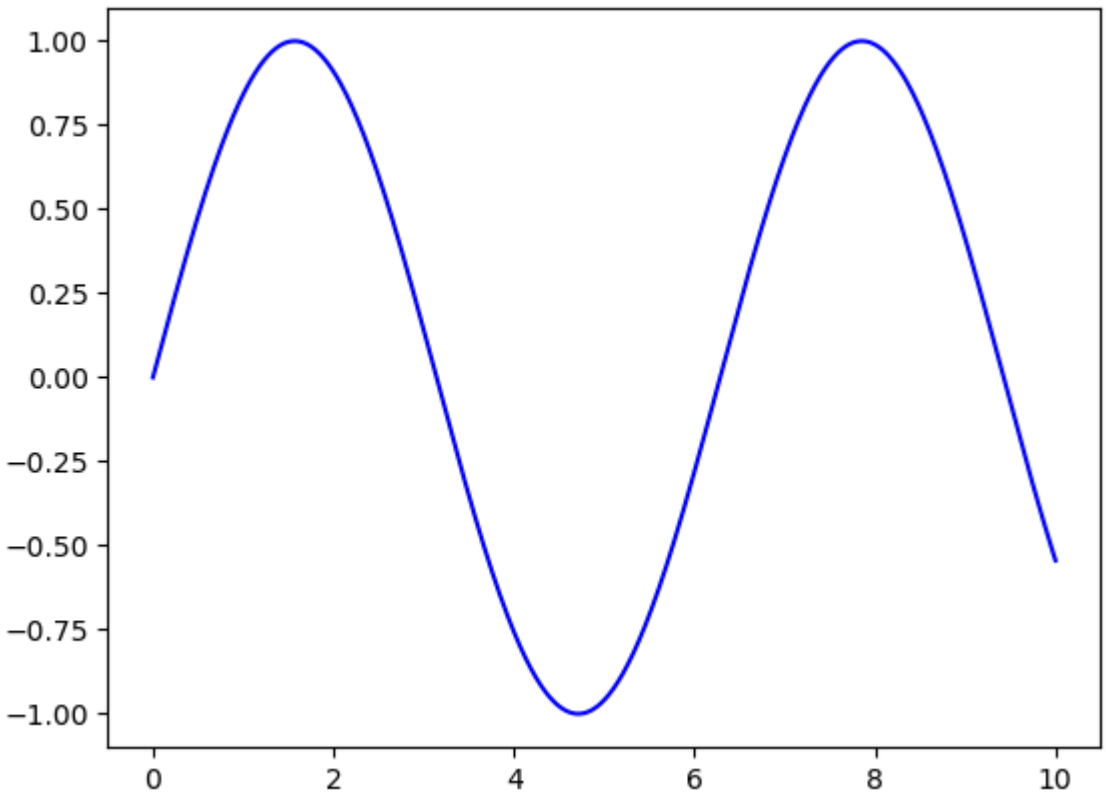
Out[30]:
```
{'eps': 'Encapsulated Postscript',
 'jpg': 'Joint Photographic Experts Group',
 'jpeg': 'Joint Photographic Experts Group',
 'pdf': 'Portable Document Format',
 'pgf': 'PGF code for LaTeX',
 'png': 'Portable Network Graphics',
 'ps': 'Postscript',
 'raw': 'Raw RGBA bitmap',
 'rgba': 'Raw RGBA bitmap',
 'svg': 'Scalable Vector Graphics',
 'svgz': 'Scalable Vector Graphics',
 'tif': 'Tagged Image File Format',
 'tiff': 'Tagged Image File Format',
 'webp': 'WebP Image Format'}
```

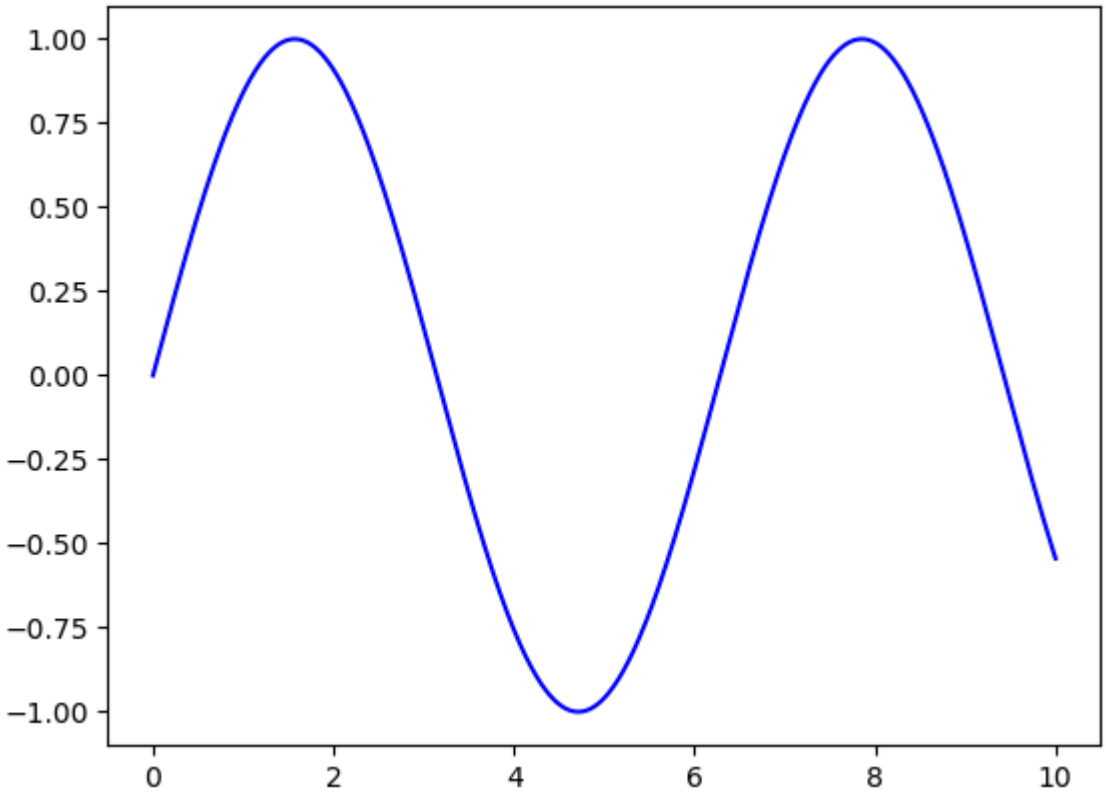
15. Line Plot

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

In [32]:
```
# Create figure and axes first
fig = plt.figure()

ax = plt.axes()

# Declare a variable x5
x5 = np.linspace(0, 10, 1000)
```

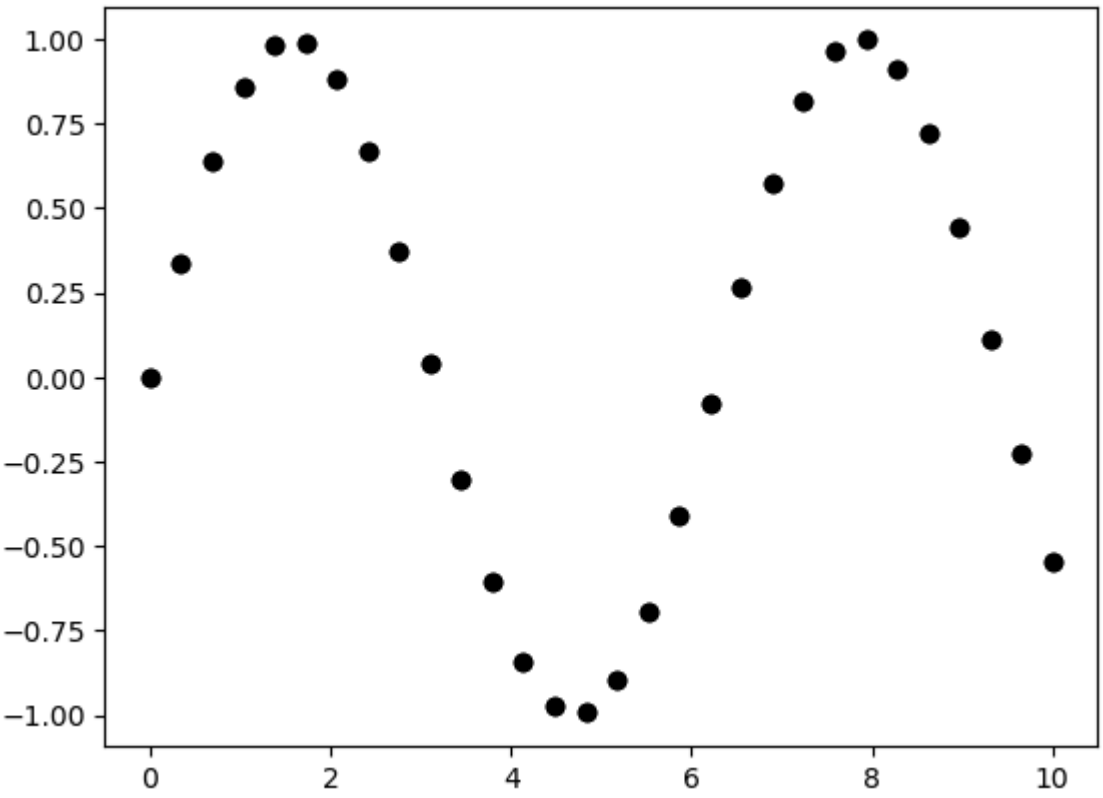
```
# Plot the sinusoid function
ax.plot(x5, np.sin(x5), 'b-');
plt.show()
```





In [34]:
```
x7 = np.linspace(0, 10, 30)

y7 = np.sin(x7)
```

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

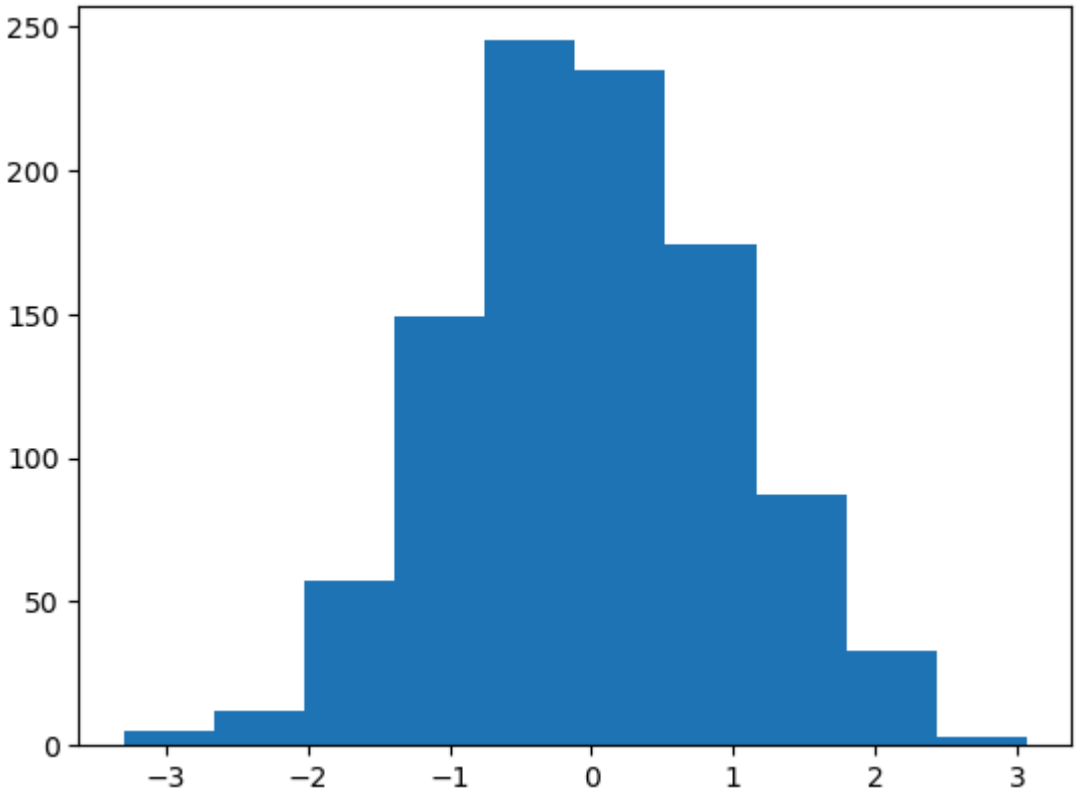


17. Histogram

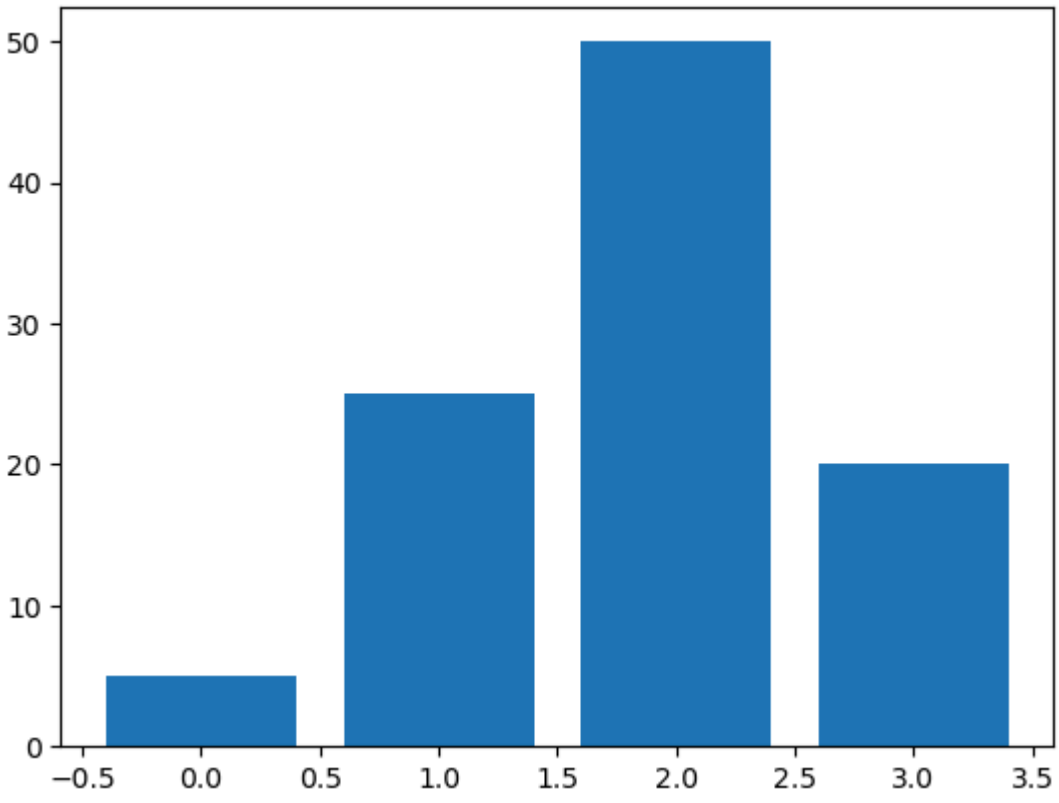
Histogram charts are a graphical display of frequencies. They are represented as bars. They show what portion of the dataset falls into each category, usually specified as non-overlapping intervals. These categories are called bins.

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

In [35]:
```
data1 = np.random.randn(1000)

plt.hist(data1);
plt.show()
```

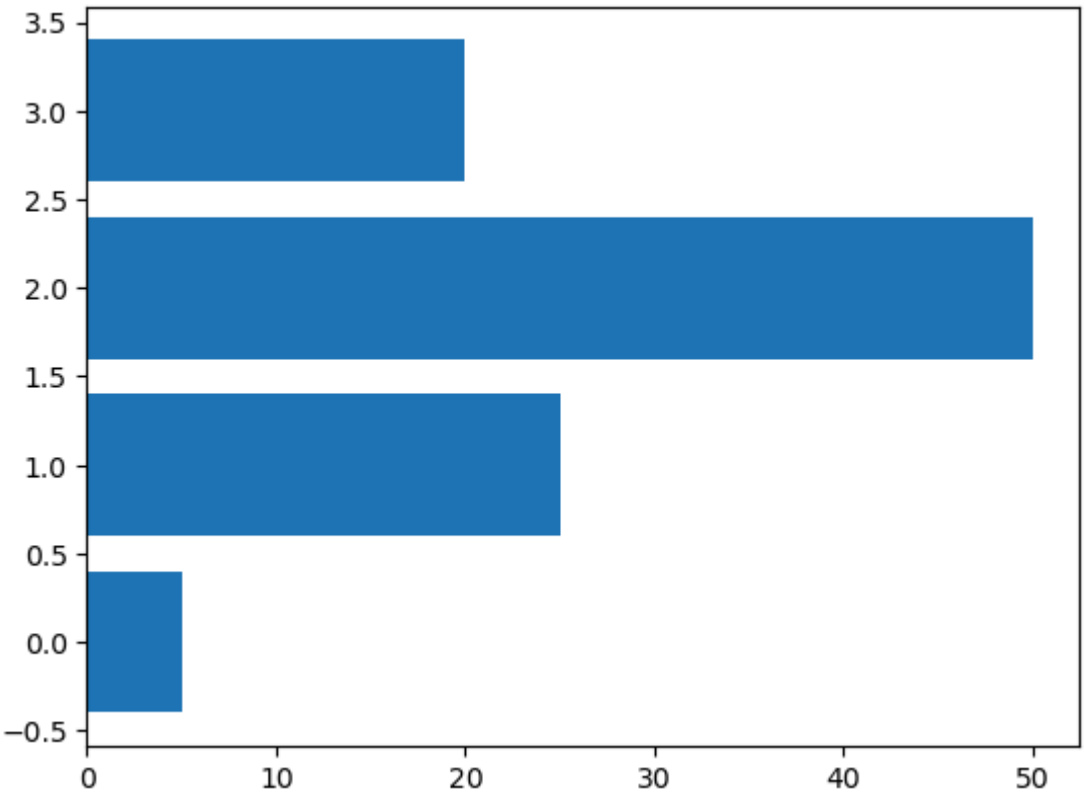


In [36]:
```
data2 = [5. , 25. , 50. , 20.]

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

plt.show()
```



In [37]:
```
data2 = [5. , 25. , 50. , 20.]

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

```
plt.show()
```

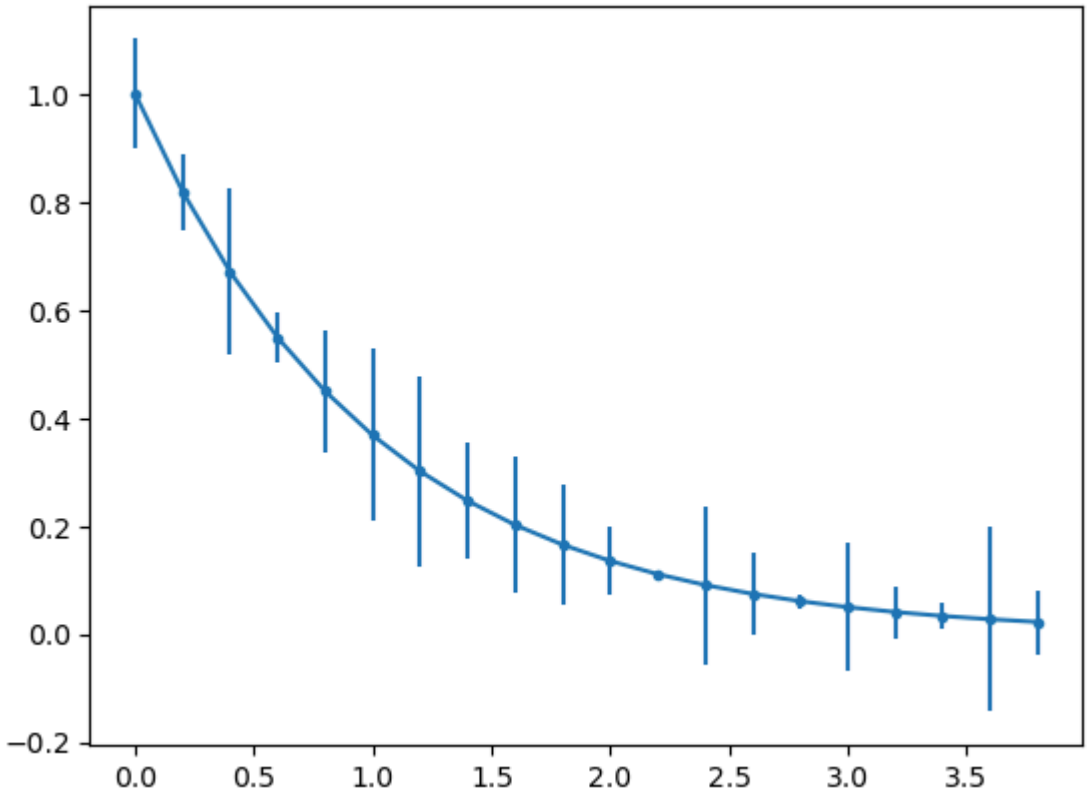


20. Error Bar Chart

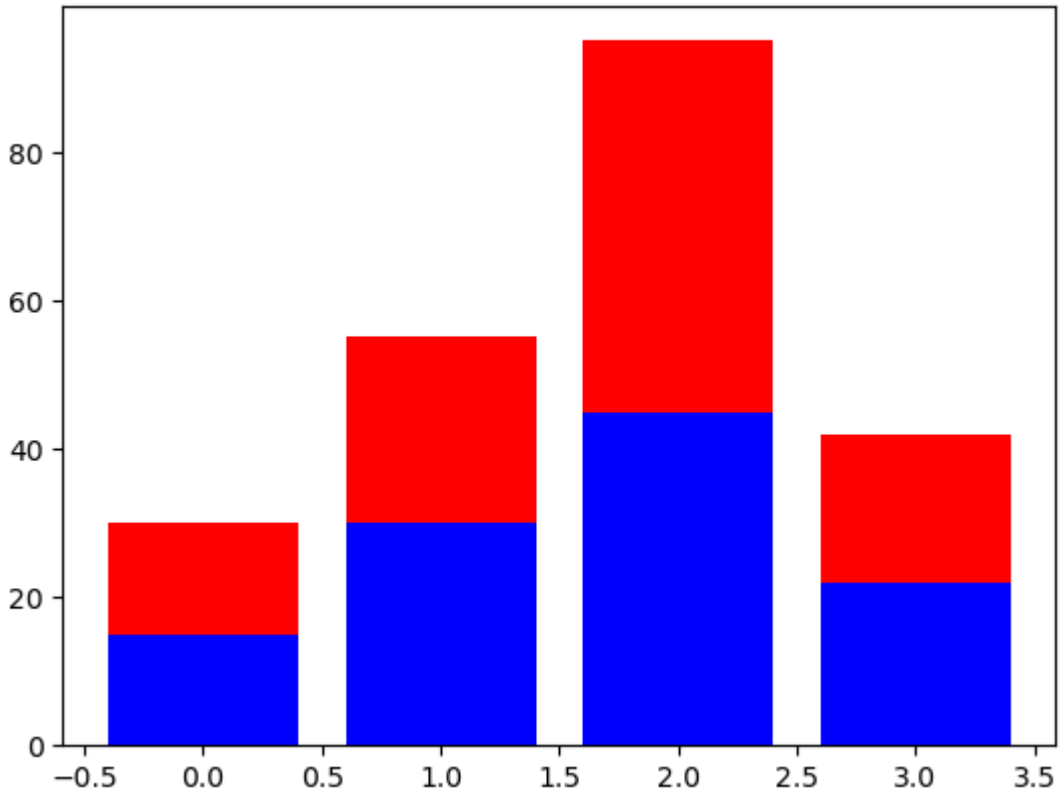
In experimental design, the measurements lack perfect precision. So, we have to repeat the measurements. It results in obtaining a set of values. The representation of the distribution of data values is done by plotting a single data point (known as mean value of dataset) and an error bar to represent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

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

In [38]:
```
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();
```



In [39]:
```
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()
```

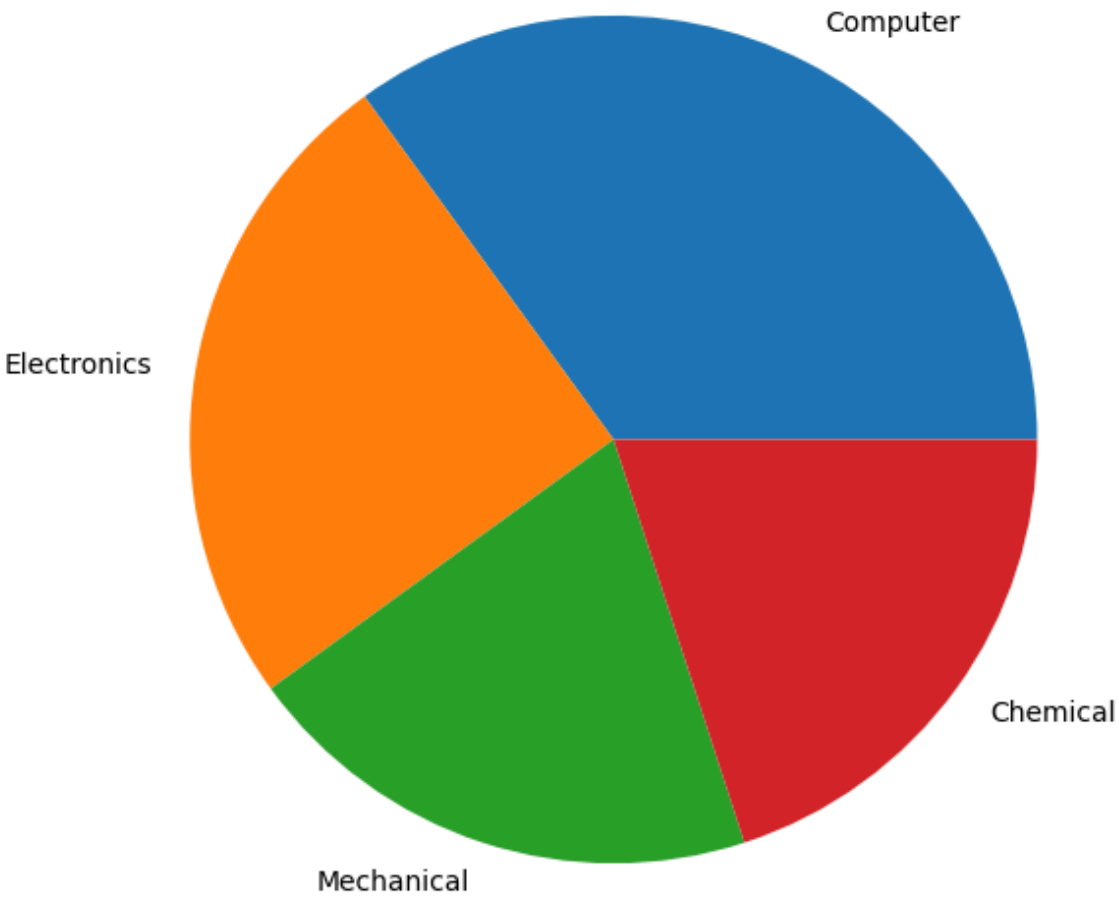


22. Pie Chart

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 [40]:
```
plt.figure(figsize=(7,7))

x10 = [35, 25, 20, 20]

labels = ['Computer', 'Electronics', 'Mechanical', 'Chemical']

plt.pie(x10, labels=labels);

plt.show()
```

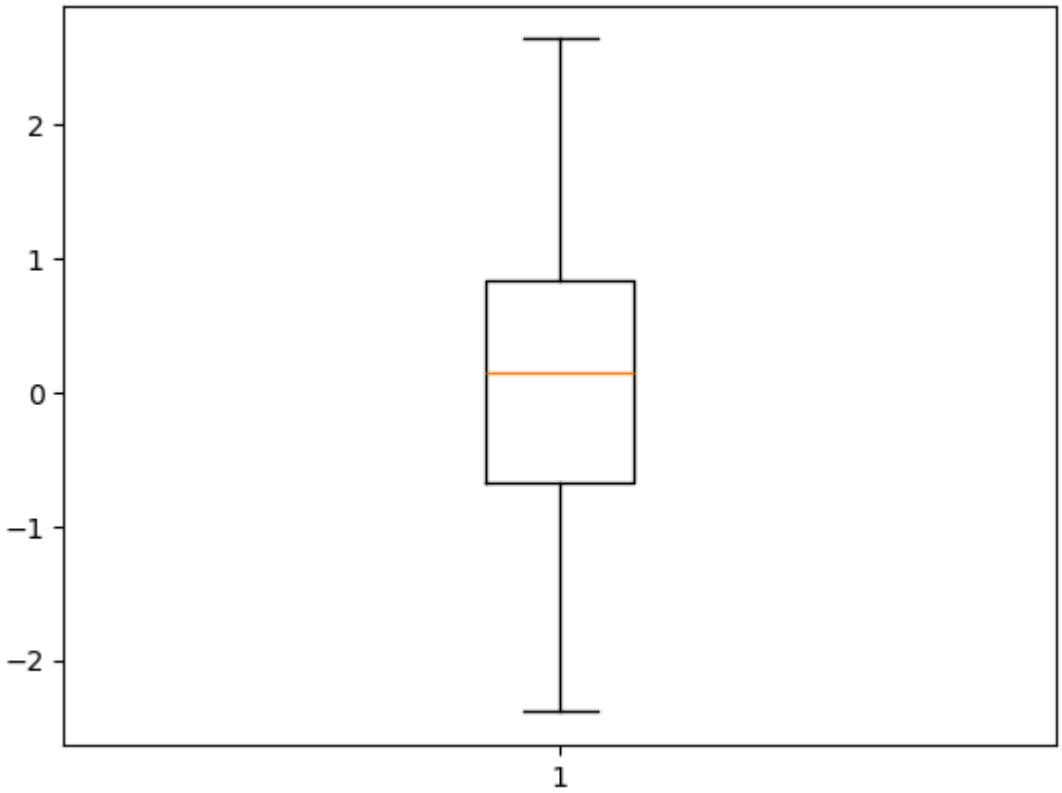


23. Boxplot

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 [41]:
```
data3 = np.random.randn(100)

plt.boxplot(data3)

plt.show();
```

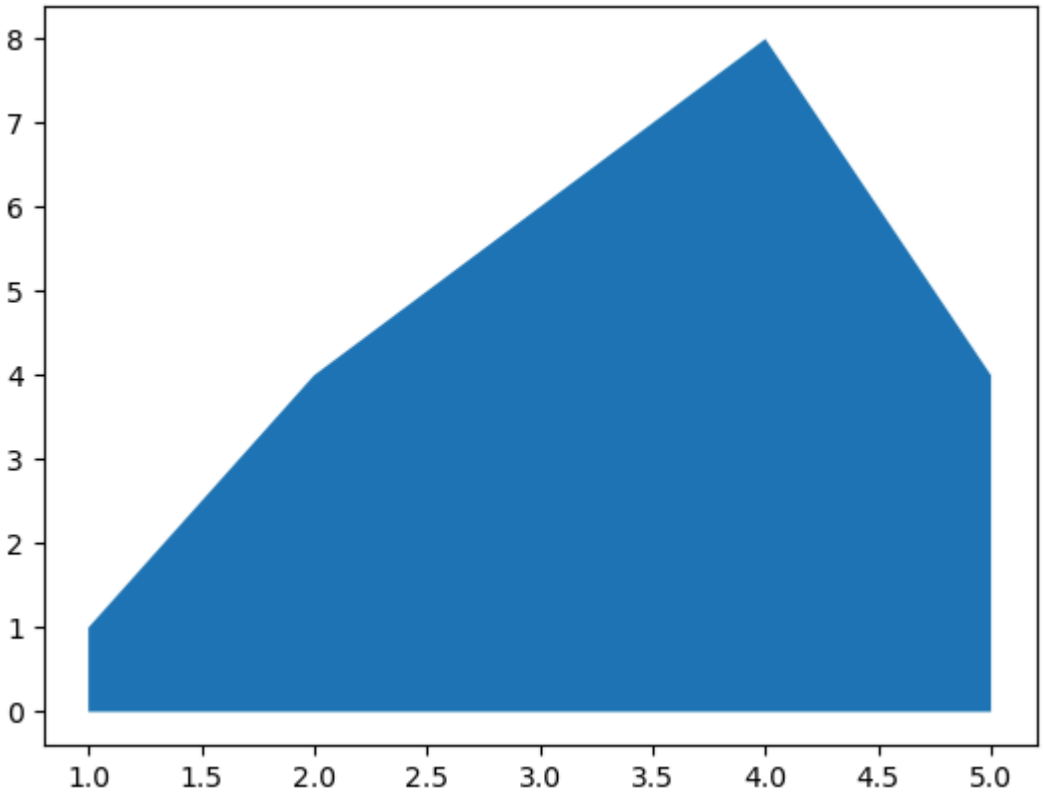


24. Area Chart

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 a numerical variable following another numerical variable.

We can create an Area Chart as follows:-

In [42]:
```
# Create some data
x12 = range(1, 6)
y12 = [1, 4, 6, 8, 4]

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



25. Contour Plot

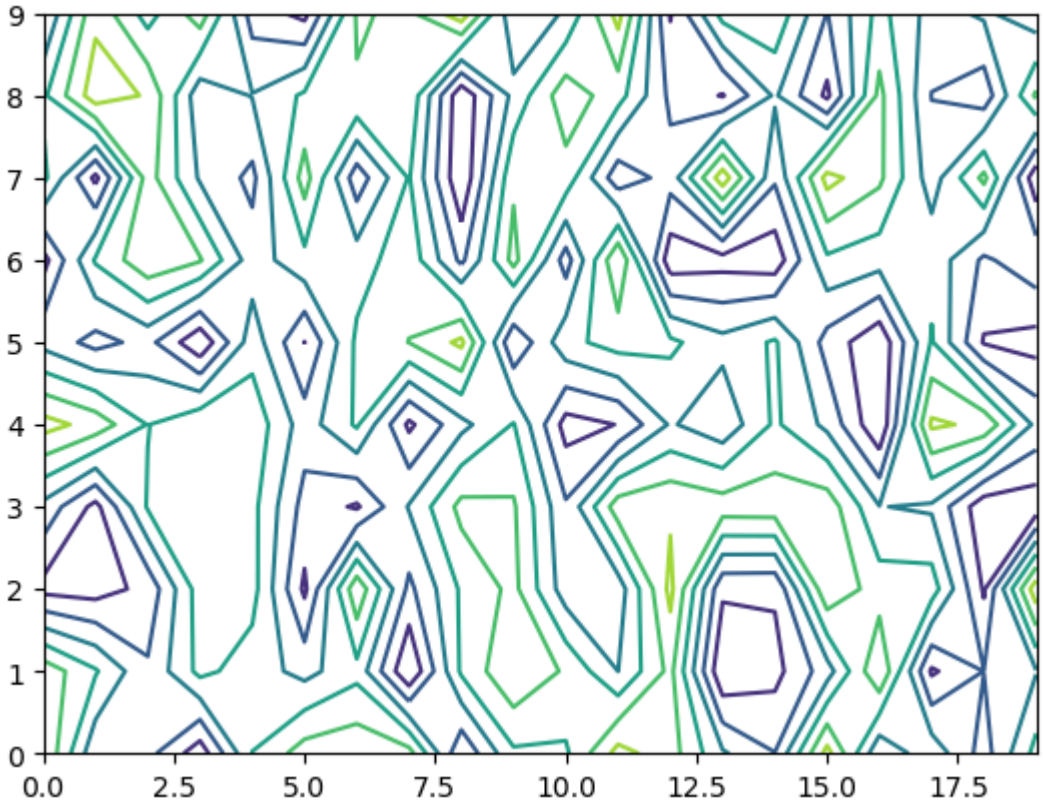
Contour plots are useful to display three-dimensional data in two dimensions using contours or color-coded regions. Contour lines are also known as level lines or isolines. Contour lines for a function of two variables are curves where the function has constant values. They have specific names beginning 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, engineering, 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 lines. 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 [43]:
```
# Create a matrix
matrix1 = np.random.rand(10, 20)

cp = plt.contour(matrix1)

plt.show()
```



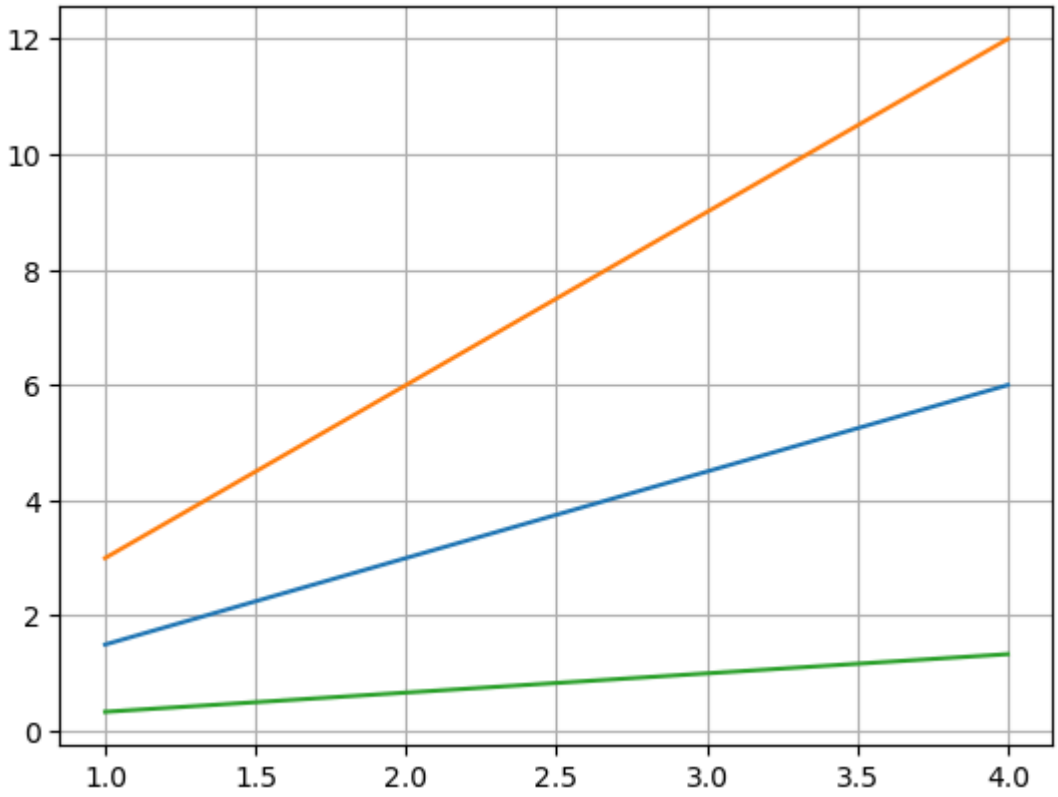
In [44]:
```
# View list of all available styles

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']
```

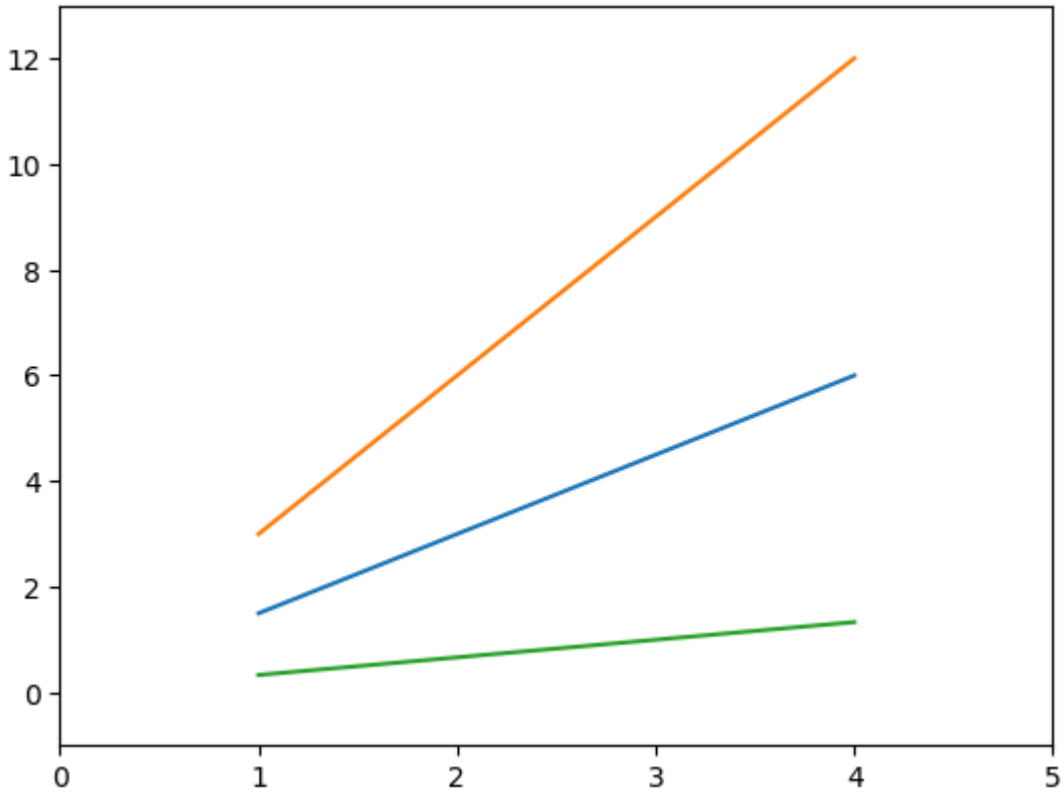
27. Adding a grid

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.

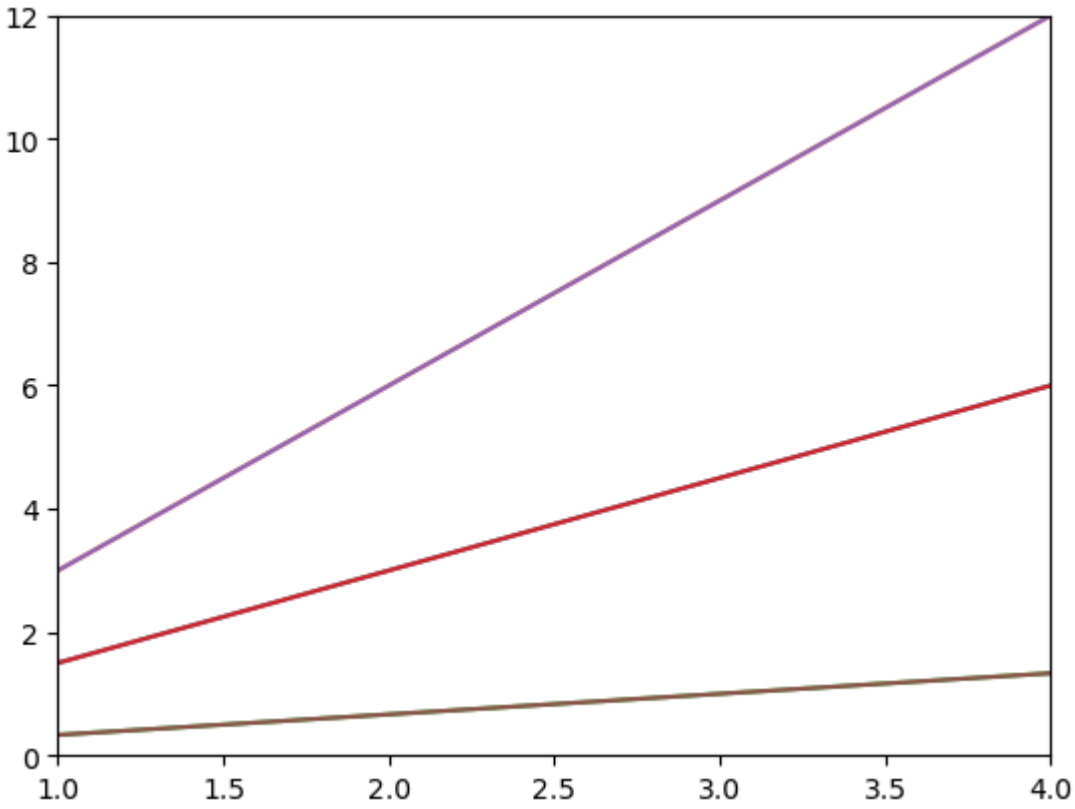
In [47]:
```
x15 = np.arange(1, 5)

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

plt.grid(True)

plt.show()
```



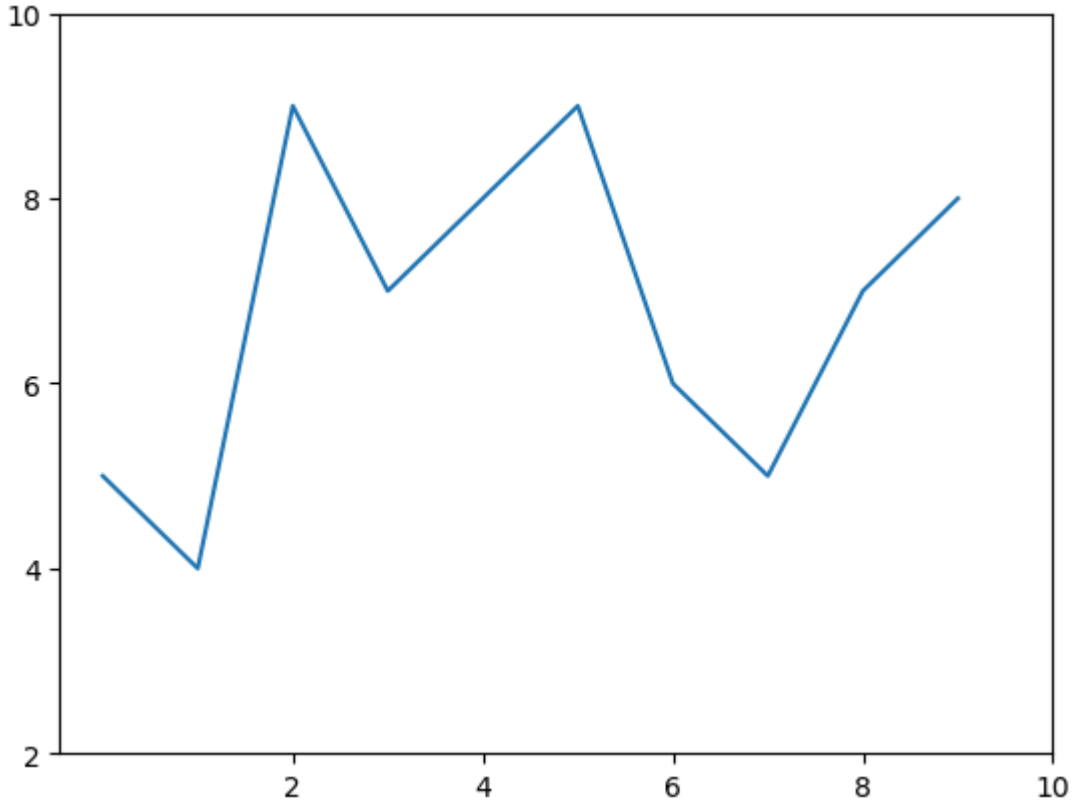
In [49]:
```
x15 = np.arange(1, 5)

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()
```



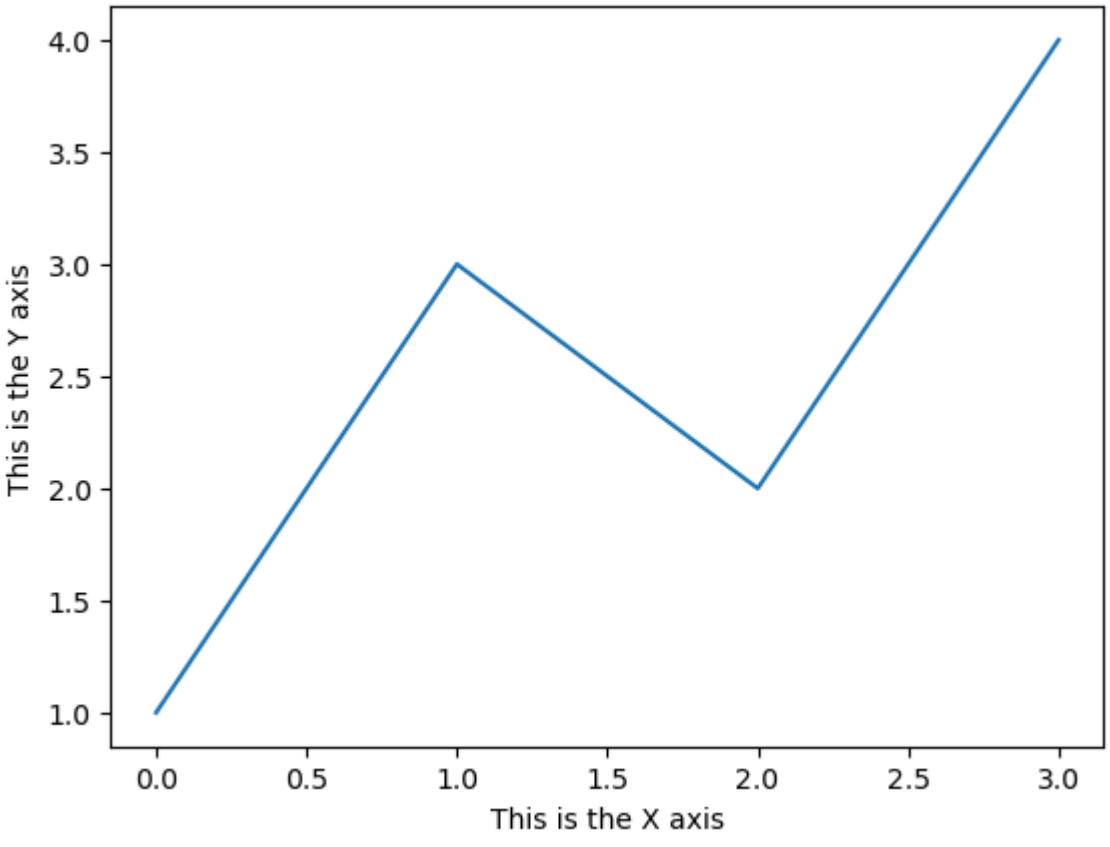
In [51]:
```
x15 = np.arange(1, 5)

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()
```



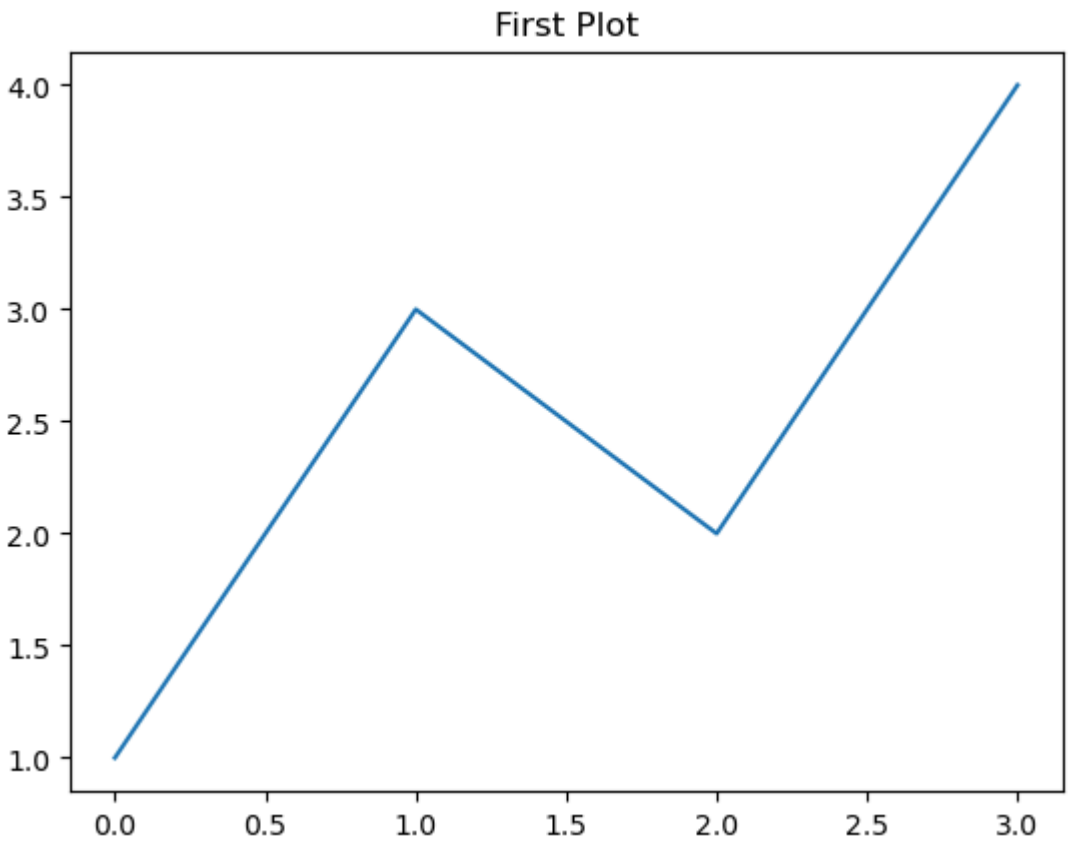
In [52]:
```
u = [5, 4, 9, 7, 8, 9, 6, 5, 7, 8]

plt.plot(u)

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

plt.show()
```



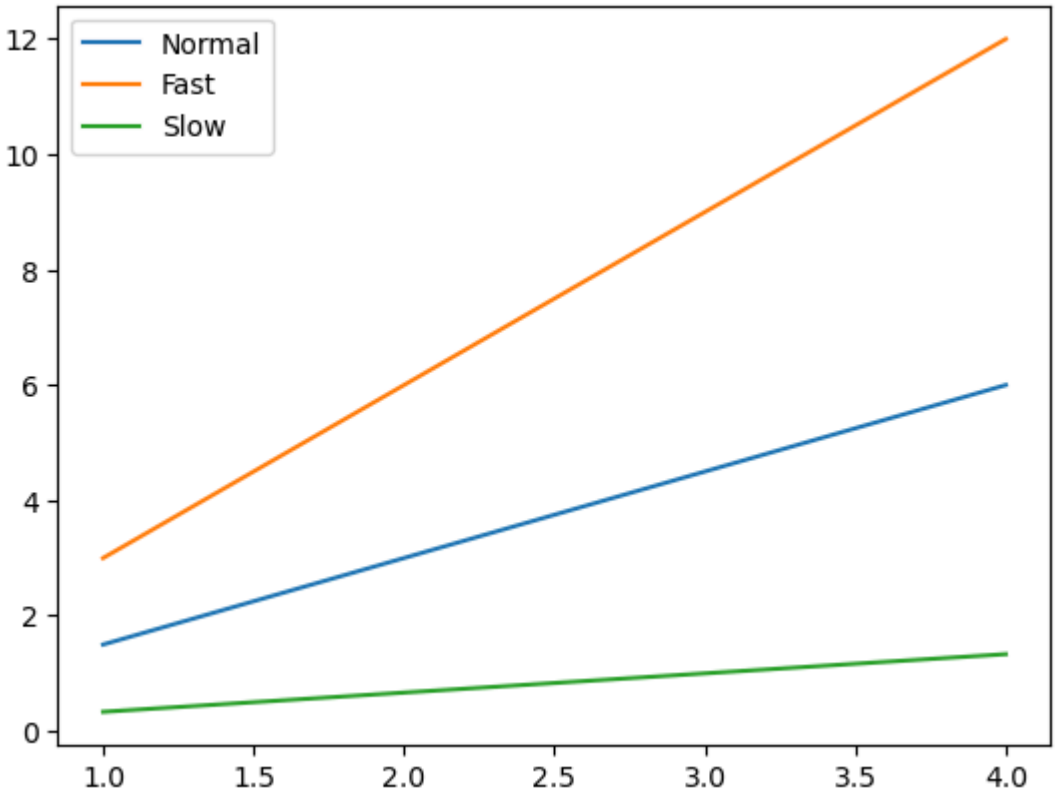
In [53]:
```
plt.plot([1, 3, 2, 4])

plt.xlabel('This is the X axis')

plt.ylabel('This is the Y axis')

plt.show()
```

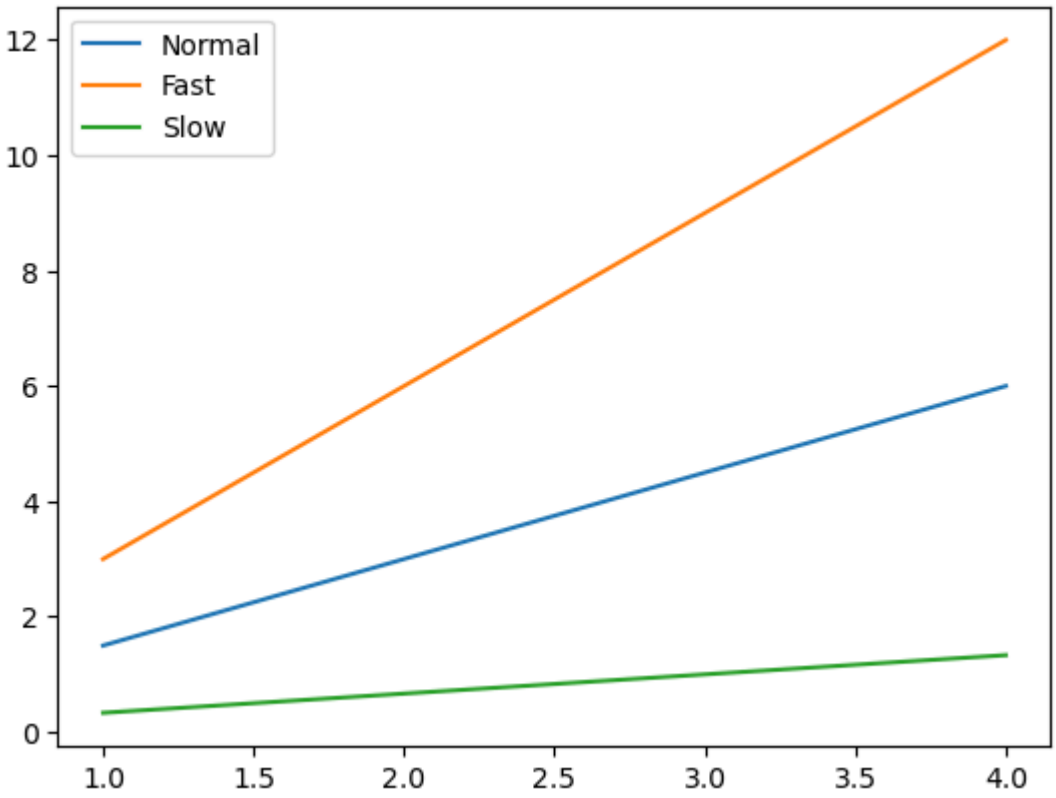


In [54]:
```
plt.plot([1, 3, 2, 4])

plt.title('First Plot')

plt.show()
```

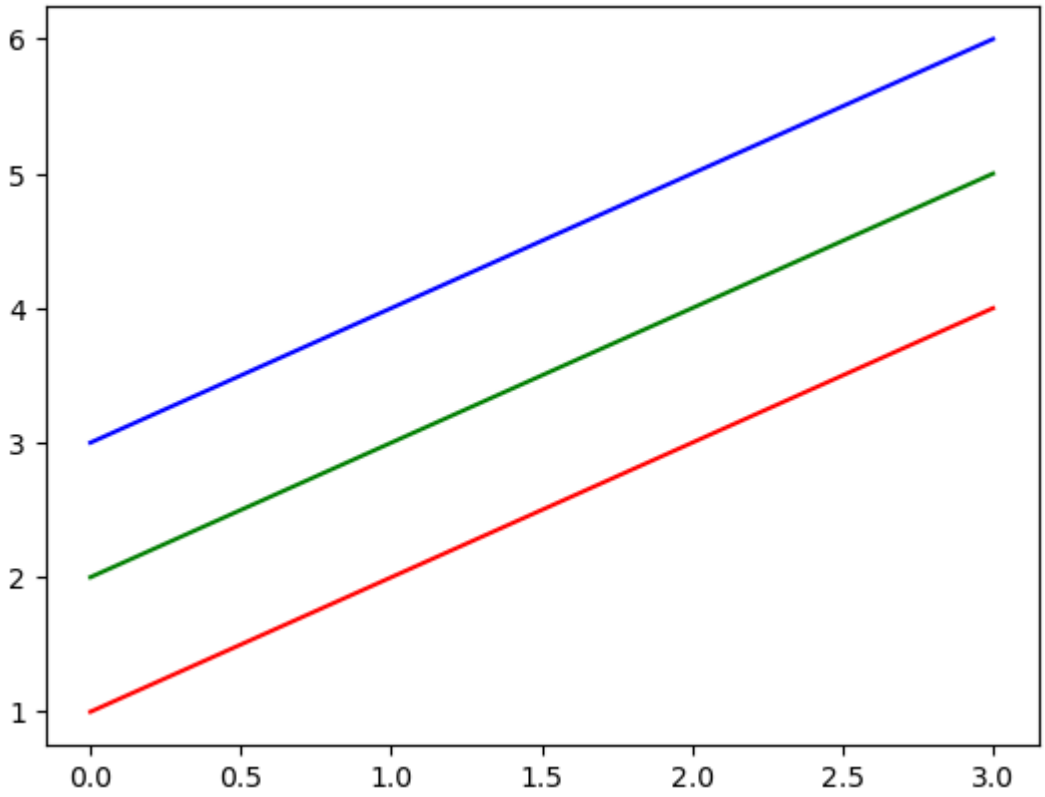


In [56]:
```
x15 = np.arange(1, 5)

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()
```

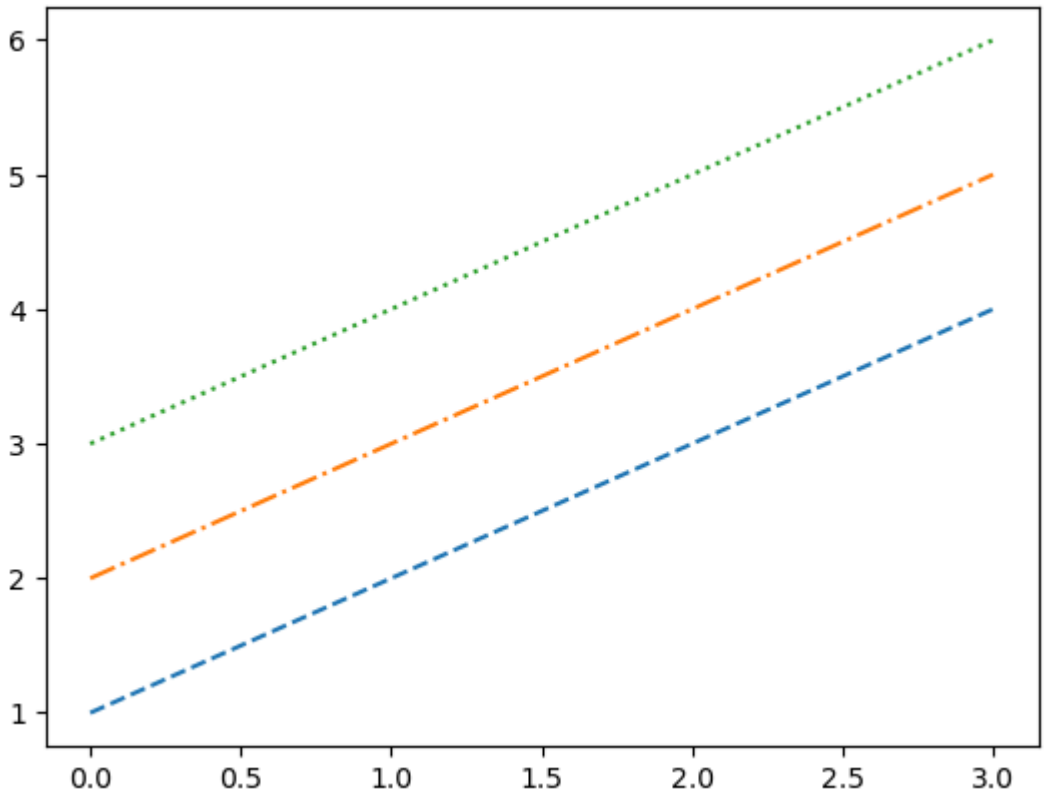




In [57]:
```
x16 = np.arange(1, 5)

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

plt.show()
```



In [58]:
```
x16 = np.arange(1, 5)

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

plt.show()
```



In [ ]:
```

```