Data Visualization with Matplotlib in Numpy/Pandas.

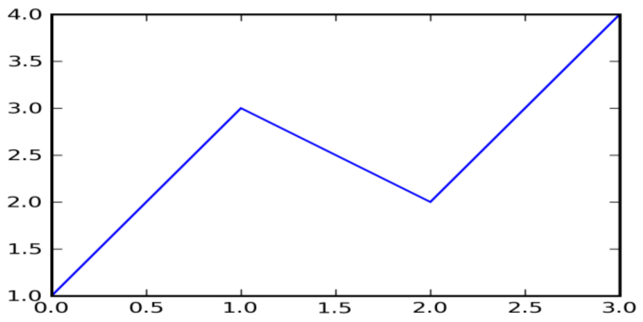
By Malay Mitra

First plots with Matplotlib

One of the strong points of Matplotlib is how quickly we can start producing plots out of the box. Here is our first example:

Example -1

```
import matplotlib.pyplot as plt
plt.plot([1, 3, 2, 4])
plt.show()
```



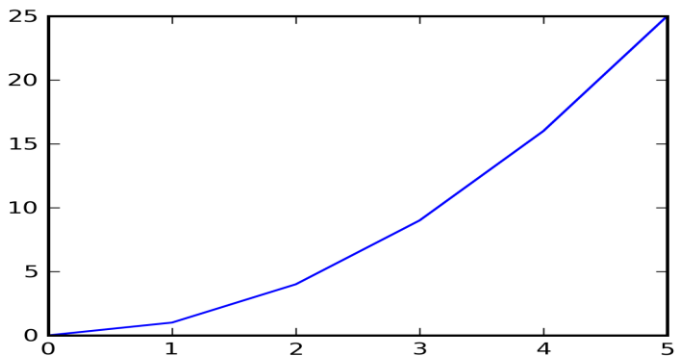
Note : We have specified only a list of values that represent the vertical coordinates of the points to be plotted. Matplotlib will use an implicit horizontal values list, from 0 (the first value) to N-1 (where N is the number of items in the list). Vertical values represent the Y-axis while the horizontal values are the X-axis, and what we do is called "to plot Y against X".

We can also explicitly specify both the lists:

Example – 2

```
 import matplotlib.pyplot as plt
 x = range(6)
plt.plot(x, [xi**2 for xi in x])
plt.show()
```

This script results in the following screenshot :

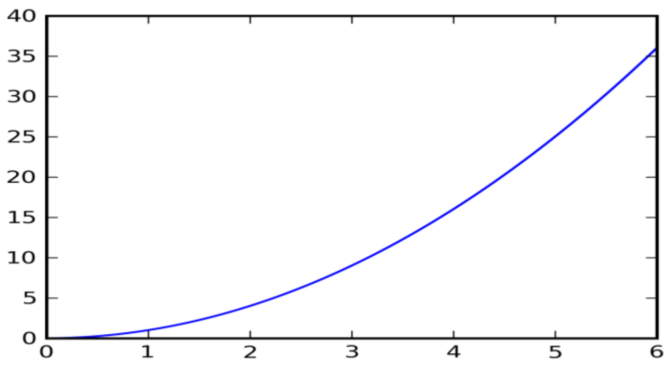


The line shown here has several edges. To have a smoother parabola. So, we can start using the NumPy

So we can use arange() to generate a finer range:
Example – 3

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(0.0, 6.0, 0.01)
plt.plot(x, [x**2 for x in x])
plt.show()
```

Above program generates following graph :

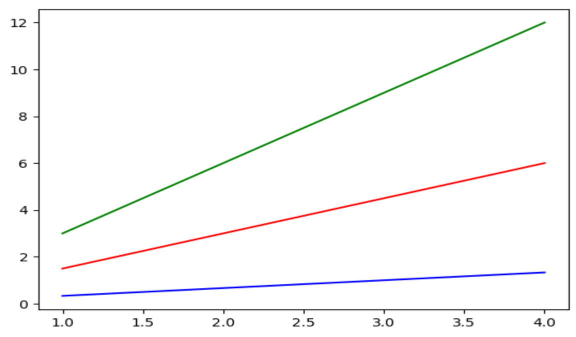


Multiline plots

We can plot more than one line on the same figure. Check the below code and screenshot

Example – 4

```
import matplotlib.pyplot as plt
x = range(1, 5)
plt.plot(x, [xi*1.5 for xi in x],'r')    # red
plt.plot(x, [xi*3.0 for xi in x],'g')   # green
plt.plot(x, [xi/3.0 for xi in x],'b')   # blue
plt.show()
```



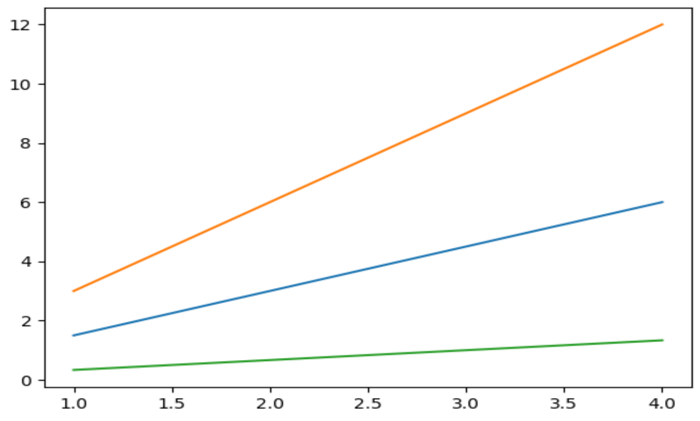
Screenshot

We can plot multiline figures by passing the X and Y values list in a single plot() call:

Example – 5

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1, 5)
plt.plot(x, x*1.5, x, x*3.0, x, x/3.0)                    # Three plots in same call of plot() function
plt.show()
```

Screenshot :



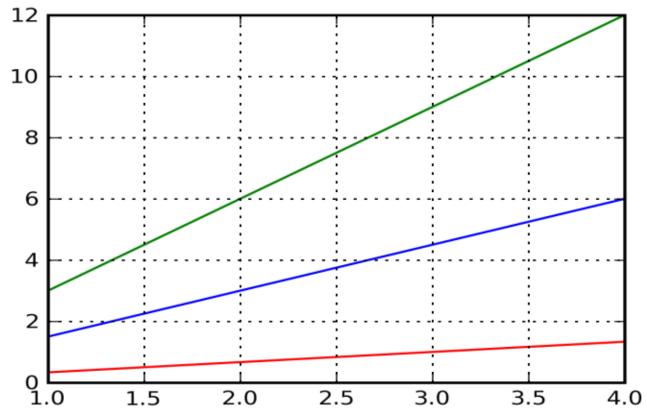
<u>Grid, axes, labels, titles</u>

Now we can add some features to plots to control them better.

<u>Grid</u>

Example – 6

```
    import matplotlib.pyplot as plt
    import numpy as np
    x = np.arange(1, 5)
    plt.plot(x, x*1.5, x, x*3.0, x, x/3.0)
    plt.grid(True)                    # Adding grid to the graph
    plt.show()
```



Screenshot :

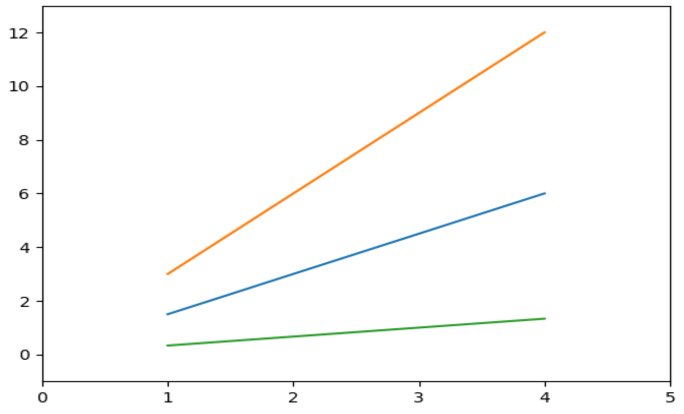
<u>Handling Axes</u>

We have noticed that Matplotlib automatically sets the limits of the figure to precisely contain the plotted data sets. We can set the axes limits as per our choice (defining the scale of the chart).

Example – 7

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1, 5)
plt.plot(x, x*1.5, x, x*3.0, x, x/3.0)
plt.axis()                  # shows the current axis limits values as (xmin, xmax, ymin, ymax)
plt.axis([0, 5, -1, 13])    # set new axes limits as (xmin,  xmax,  ymin, ymax)
plt.show()
```

Screenshot :



The whole set of four values of keyword arguments [xmin, xmax, ymin, ymax], helps us to specify the minimum and maximum limits respectively for the X-axis and the Y-axis at the same time. We can also use the specific keyword arguments, for example:

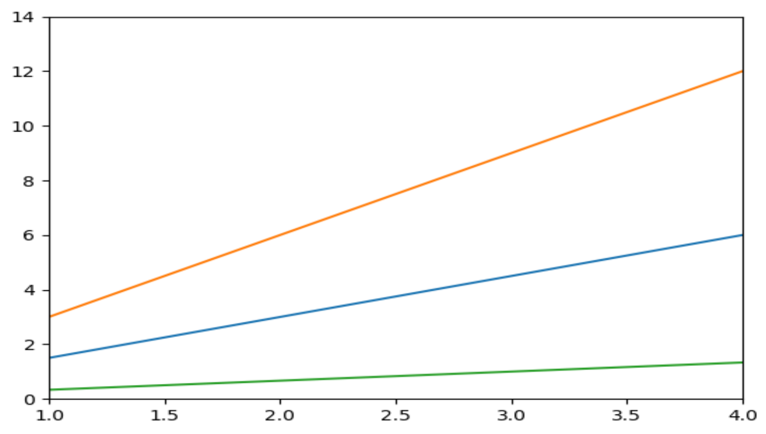
```
plt.axis(xmin=NNN, ymax=NNN)        # Specify xmin and ymax
```

We can also control the limits for each axis separately using plt.xlim() and plt.ylim() functions.

Example – 8

```
import matplotlib.pyplot as plt
import numpy as np
x=np.arange(1, 5)
plt.plot(x, x*1.5, x, x*3.0, x, x/3.0)
plt.xlim(1,4)                  # Setting X-axis limit from 1 to 4
plt.ylim(0,14)                 # Setting Y-axis limit from 0 to 14
plt.show()
```

Screenshot :



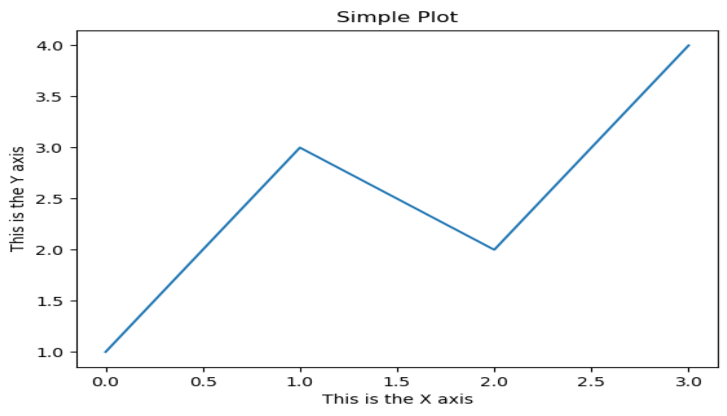
<u>Adding labels and Title</u>

To add axes labels to a plot is the axes labels, since they usually specify what kind of data we are plotting.

Example – 9

```
import matplotlib.pyplot as plt
plt.plot([1, 3, 2, 4])
plt.xlabel('This is the X axis')
plt.ylabel('This is the Y axis')
plt.title('Simple plot')
plt.show()
```

Screenshot :



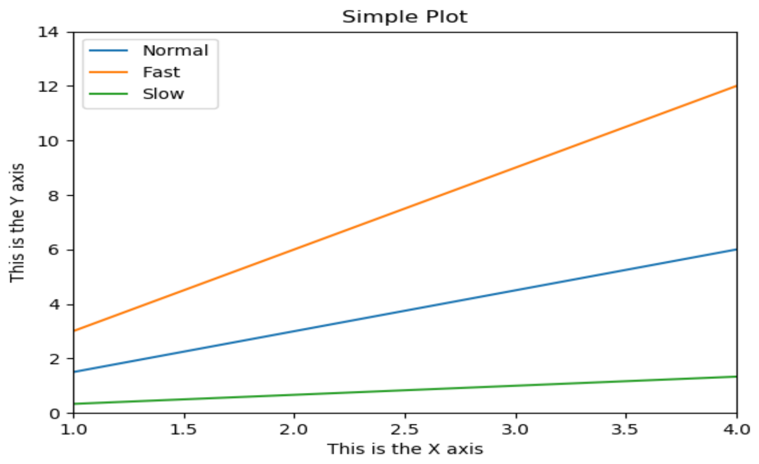
<u>Adding a legend</u>

Legends are used to explain what each line means in the current figure. We consider the multiline plot example again, and extend it with a legend format:

Example – 10

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1, 5)
plt.plot(x, x*1.5, label='Normal')
plt.plot(x, x*3.0, label='Fast')
plt.plot(x, x/3.0, label='Slow')
plt.xlabel('This is the X axis')
plt.ylabel('This is the Y axis')
plt.xlim(1,4)
plt.ylim(0,14)
plt.title('Simple Plot')
plt.legend()                    # Setting legends
plt.show()
```

Screenshot :



We added an extra keyword argument called 'label' to the plot() call. This keyword argument provides the information required to build the text of the legend

Alternatively we can mention as

```
plt.plot(x, x*1.5)
plt.plot(x, x*3.0)
plt.plot(x, x/3.0)
plt.legend(['Normal', 'Fast', 'Slow'])
```

But we have to remember the order of lines plotted because the legend() parameters are matched with the lines as they are plotted.
We can also specify the locations of the Legends. Some of them can be:

String	**Code**
best	0
upper right	1
upper left	2
lower right	4
right	5

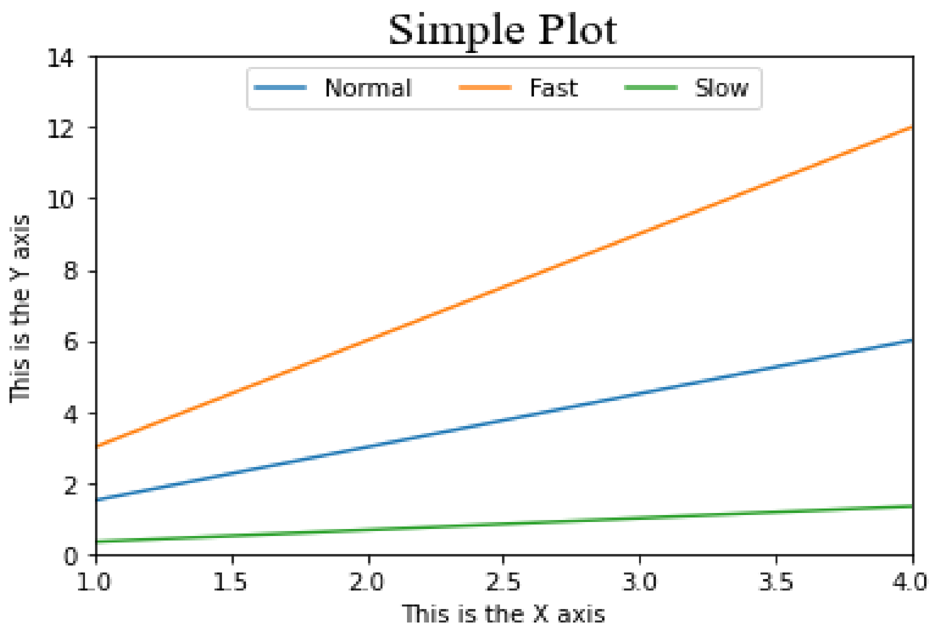
Syntax: *plt.legend(loc = 'upper right')*

<u>With more parameters on title and legend</u>

Example - 11

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1, 5)
plt.plot(x, x*1.5, label='Normal')
plt.plot(x, x*3.0, label='Fast')
plt.plot(x, x/3.0, label='Slow')
plt.xlabel('This is the X axis')
plt.ylabel('This is the Y axis')
plt.xlim(1,4)
plt.ylim(0,14)
plt.title('Simple Plot', fontsize=20, fontname ='Times New Roman')
plt.legend(loc='upper center', ncol=3)      # Parameters for legends. ncol is number of columns
plt.show()
```

Screenshot:



Saving plots to a file

Saving a plot to a file is an easy task. The following command saves the plot to a PNG file on disk.

```
plt.savefig('plot123.png')          # Saves the plot on disk as a PNG file
```

- **Markers and line styles**

By default, Matplotlib draws markers as a single dot and lines as straight lines. There are situations where we can change the marker style to clearly identify them in the plot and the line style so that the line appears dashed or something as per command.

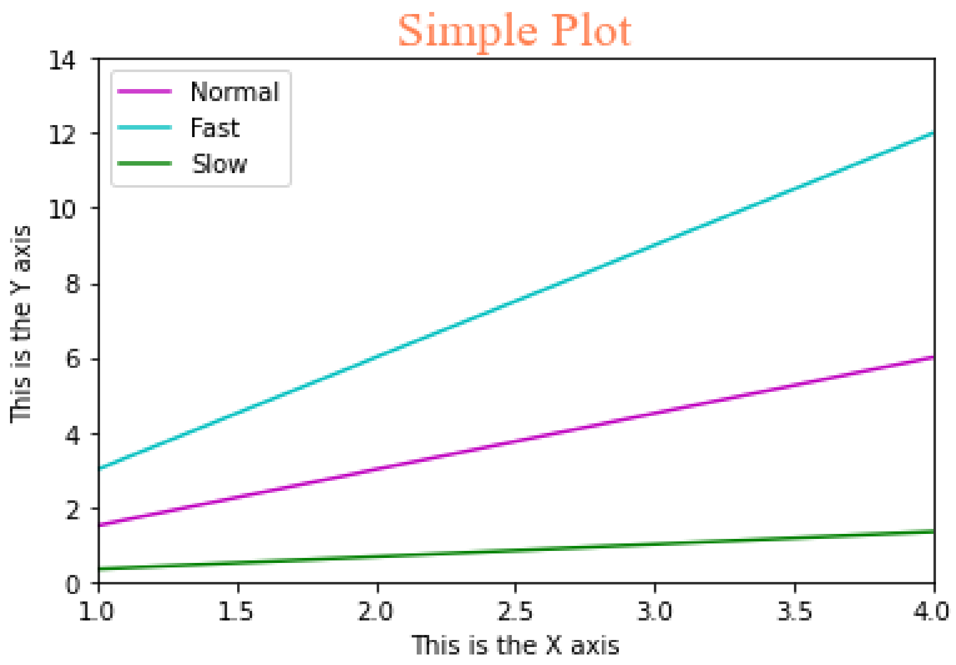
There are three levels of customization as mentioned below:

• Colors
• Line styles
• Marker styles

Example – 12

```
# Colors with Matplotlib
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1, 5)
plt.plot(x, x*1.5, color='m', label='Normal')                # Magenta
plt.plot(x, x*3.0, color='c', label='Fast')                   # Cyan
plt.plot(x, x/3.0, color='g', label='Slow')                   # Green
plt.xlabel('This is the X axis')
plt.ylabel('This is the Y axis')
plt.xlim(1,4)
plt.ylim(0,14)
plt.title('Simple Plot', fontsize=20, color='coral', fontname ='Times New Roman')
plt.legend()
plt.show()
```

Screenshot:



List of Color codes used with Matplotlib :

Color abbreviation	**Color Name**
b	**blue**
c	**cyan**
g	**green**
k	**black**
m	**magenta**
r	**red**
w	**white**
y	**yellow**

Example – 13

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1, 5)
plt.plot(x, x*1.5, 'm', x, x*3.0, 'c', x, x/3.0, 'g')     # 'm' – Magenta, 'c' – Cyan, 'g' - Green
plt.xlabel('This is the X axis')
plt.ylabel('This is the Y axis')
plt.legend(['Normal', 'Fast', 'Slow'])
plt.title('Simple Plot')
plt.show()
```

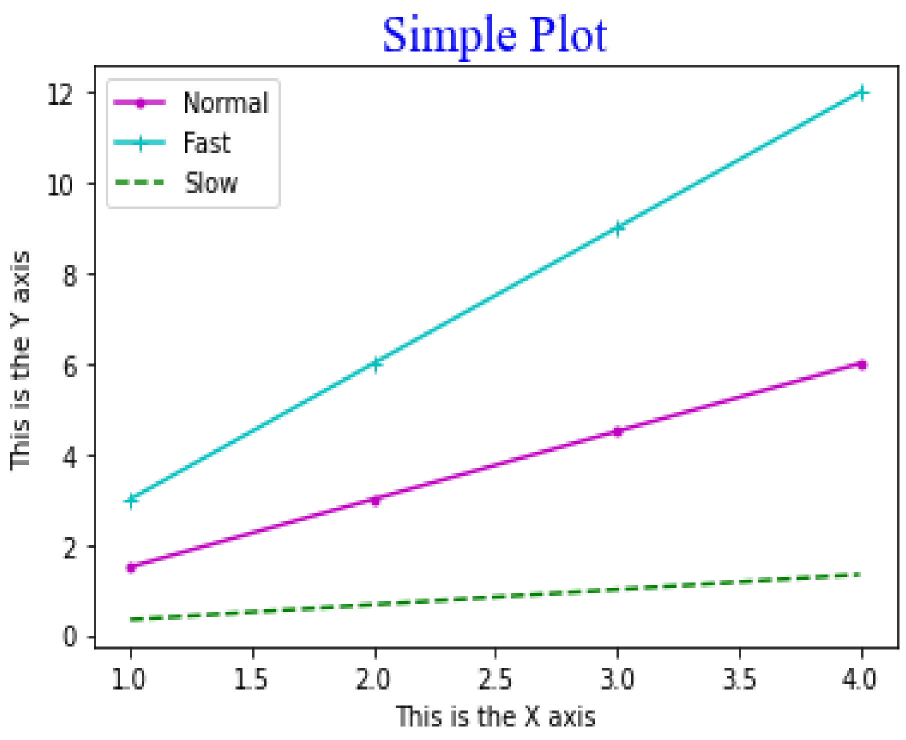
- **<u>Control line styles</u>**

Matplotlib allows us to use different line styles, for example:

Example - 14

```
 # Control line styles
# Matplotlib allows us to use different line styles, for example:
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1, 5)
plt.plot(x, x*1.5, 'm.-', x, x*3.0, 'c+-', x, x/3.0, 'g--')  # magenta dot-dashed, cyan +- , green dashed line
plt.xlabel('This is the X axis')
plt.ylabel('This is the Y axis')
plt.legend(['Normal', 'Fast', 'Slow'])
plt.title('Simple Plot', fontsize=20, color='blue', fontname ='Times New Roman')
plt.show()
```

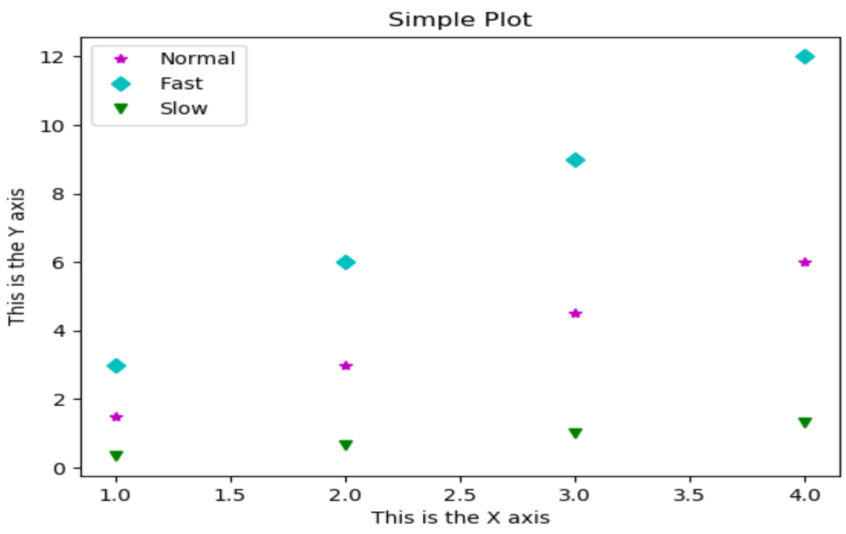
This code generates a magenta dot-dashed line, a cyan +- line, and a green dashed line, as shown in the next screenshot:



- <u>**Control marker styles**</u>

Example – 15

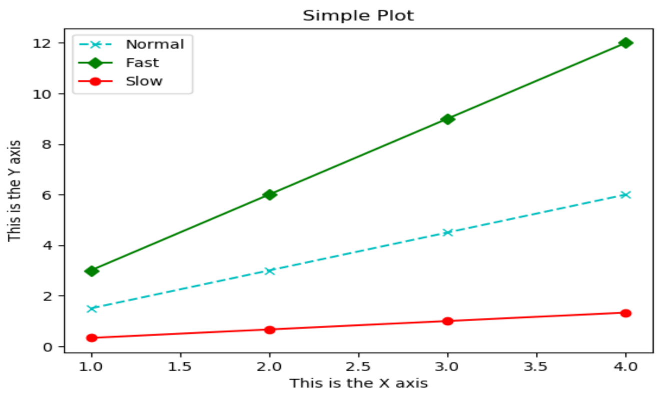
```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1, 5)
# Magenta/ '*', Cyan/Diamond,Green/Traingle   . 'D' stands for diamond, 'v' stands for triangle
plt.plot(x, x*1.5, 'm*', x, x*3.0, 'cD', x, x/3.0, 'gv')
plt.xlabel('This is the X axis')
plt.ylabel('This is the Y axis')
plt.legend(['Normal', 'Fast', 'Slow'])
plt.title('Simple Plot')
plt.show()
```



Screenshot:

Example – 16

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1, 5)
plt.plot(x, x*1.5, 'cx--', x, x*3.0, 'gD-', x, x/3.0, 'ro-')
plt.xlabel('This is the X axis')
plt.ylabel('This is the Y axis')
plt.legend(['Normal', 'Fast', 'Slow'])
plt.title('Simple Plot')
plt.show()
```



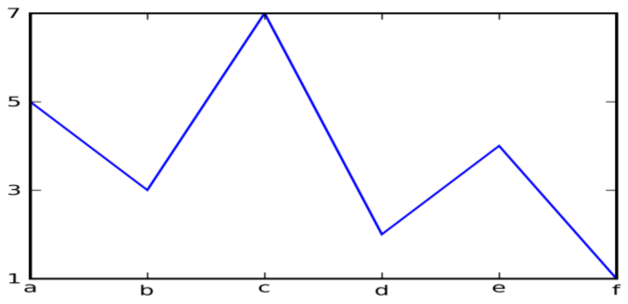
- **Handling X and Y ticks**

Vertical and horizontal ticks are those little segments on the axes, usually coupled with axes labels, used to give a reference system on the graph

Example – 17

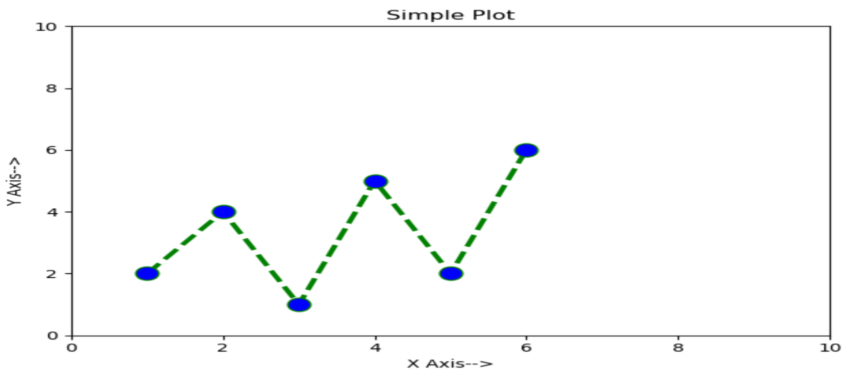
```
import matplotlib.pyplot as plt
x = [5, 3, 7, 2, 4, 1]
plt.plot(x)
plt.xticks(range(len(x)), ['a', 'b', 'c', 'd', 'e', 'f'])
plt.yticks(range(1, 8, 2))
plt.show()
```

Screenshot :



Example - 18

```
# Customization of plots
import matplotlib.pyplot as plt
x = [ 1, 2, 3, 4, 5, 6]                #X-axis values
y = [2, 4, 1, 5, 2, 6]                 # Y-axis values
plt.xlim(0, 10)
plt.ylim(0, 10)
# Now plot the chart
plt.plot(x, y, color='green', linestyle='dashed', linewidth=3, marker='o', markerfacecolor='blue',
markersize=12)
# Title / labels
plt.title('Simple Plot')
plt.xlabel('X Axis-->')
plt.ylabel('Y Axis-->')
plt.tight_layout()      # To fit the chart within display
plt.show()
```



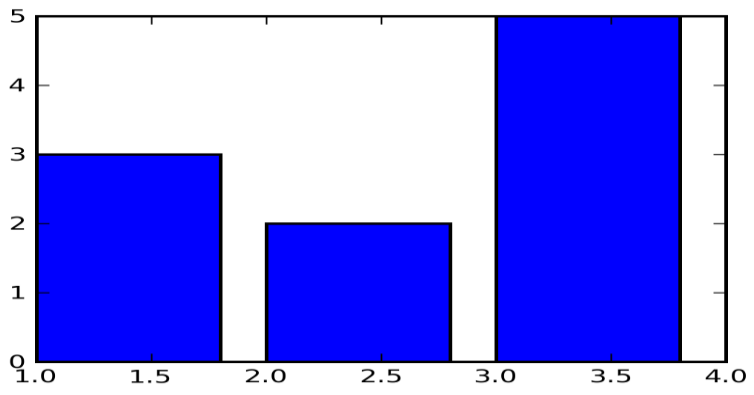
Screenshot :

- **Bar charts**
  Bar charts display rectangular bars (which can be either vertical or horizontal) with their length proportional to the values they represent.

  Example -19

```
import matplotlib.pyplot as plt
plt.bar([1, 2, 3], [3, 2, 5]);          # 1,2,3 are on x axis.  3,2.5 are heights of the bars
plt.show()
```

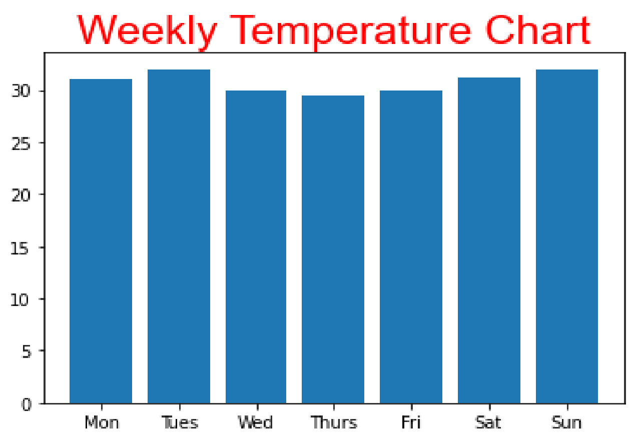
Screenshot :



Matplotlib sets, the bar width to 0.8 by default. We can change the width as per our requirements. As usual, everything is scaled and auto-adjusted to perfectly fit the figure area. Use 'barh' method to draw horizontal bar chart.

Example – 20

```
# Weekly Temperature Bar Chart
import matplotlib.pyplot as plt
temp = [31, 32, 30, 29.5, 30, 31.2, 32]
days = ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun']
plt.title('Weekly Temperature Chart', color='r', fontsize=25, fontname='Arial')
plt.bar(days, temp)
plt.show()
```



- **Text inside figure**

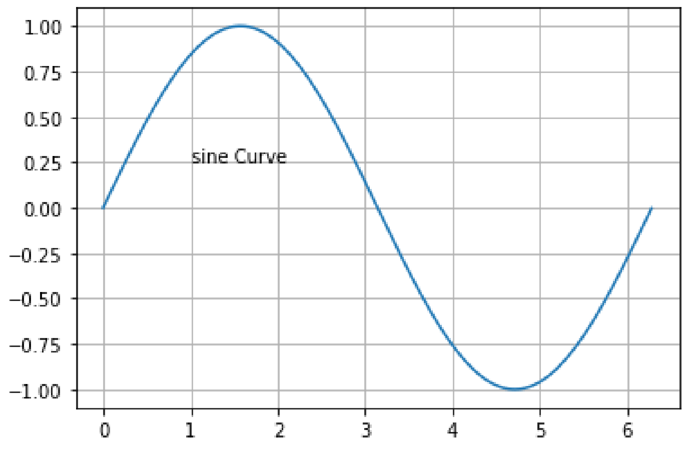
  We have already used xlabel(), ylabel(), and title() to add text around the figure, but we can do something more, namely, add text inside the figure. The text() function does that—writes a string (text) at an arbitrary position (specified by (x,y)):

  plt.text(x, y, text)

  Example - 21

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(0, 2*np.pi, .01)     # 0 to 2*Pi
y = np.sin(x)
plt.plot(x, y)
plt.text(1.0, 0.25, 'sine Curve')
plt.grid()
plt.show()
```

Screenshot :



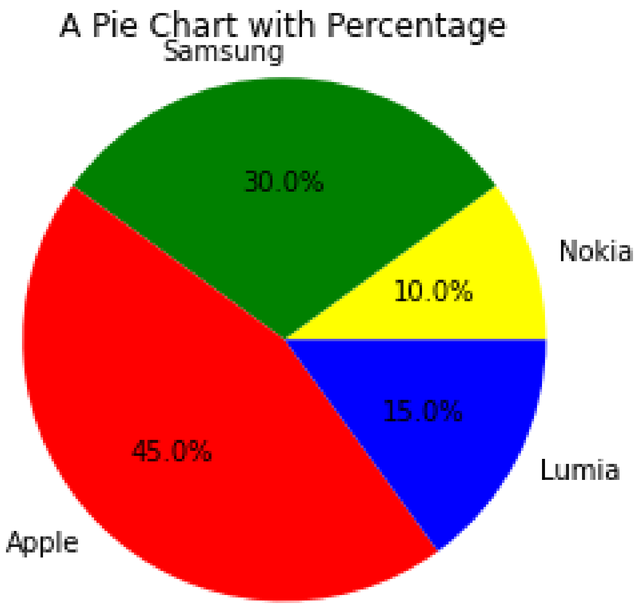
- **Pie Chart**

  Example – 22

```
# Pie chart
# I chose the percentages (their sum is 100), but you can use any value.
# pie() function will calculate the percentage occupied by each value.

import matplotlib.pyplot as plt
%matplotlib inline

labels = ['Nokia','Samsung','Apple','Lumia']   # Labels for each input value
values = [10,30,45,15]
colors = ['yellow','green','red','blue']   # Colors for each input value
plt.pie(values,labels=labels,colors=colors, autopct='%1.1f%%')
plt.title('A Pie Chart with Percentage')
plt.axis('equal')   # to draw the pie chart in a spherical way, we used the axis()
```


A Pie Chart with Percentage
Samsung
30.0%
Nokia
10.0%
15.0%
Lumia
45.0%
Apple