CSCE 5300

Data Visualization

Matplotlib

Why Data Visualization

“A picture is worth a thousand words”

- Data visualization is the first step of analysis work.
- It gives intuitive understanding of data.
- Helps you to see data in certain meaningful patterns.
- Visual representations enhances the human cognitive process.

Benefits of Data Visualization

- Data visualization allow users to see several different perspectives of data
- Data visualization makes it possible to interpret vast amounts of data.
- It offers ability to note expectations in data
- Exploring trends within a database through visualization by letting analysts navigate through data and visually orient themselves to the patterns in the data

Matplotlib

- Matplotlib is an amazing visualization library in Python for 2D plots of arrays
- Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack.
- It was introduced by John Hunter in the year 2002.
- One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals.
- Matplotlib consists of several plots like line, bar, scatter, histogram etc

Importing matplotlib

- from matplotlib import pyplot as plt *or*
- import matplotlib.pyplot as plt

Basic plots in Matplotlib

- Matplotlib comes with a wide variety of plots.
- Plots helps to understand trends, patterns, and to make correlations.
- They’re typically instruments for reasoning about quantitative information.

Line plot

```
# importing matplotlib module
from matplotlib import pyplot as plt

# x-axis values
x = [5, 2, 9, 4, 7]

# Y-axis values
y = [10, 5, 8, 4, 2]

# Function to plot
plt.plot(x,y)

# function to show the plot
plt.show()
```

Bar plot

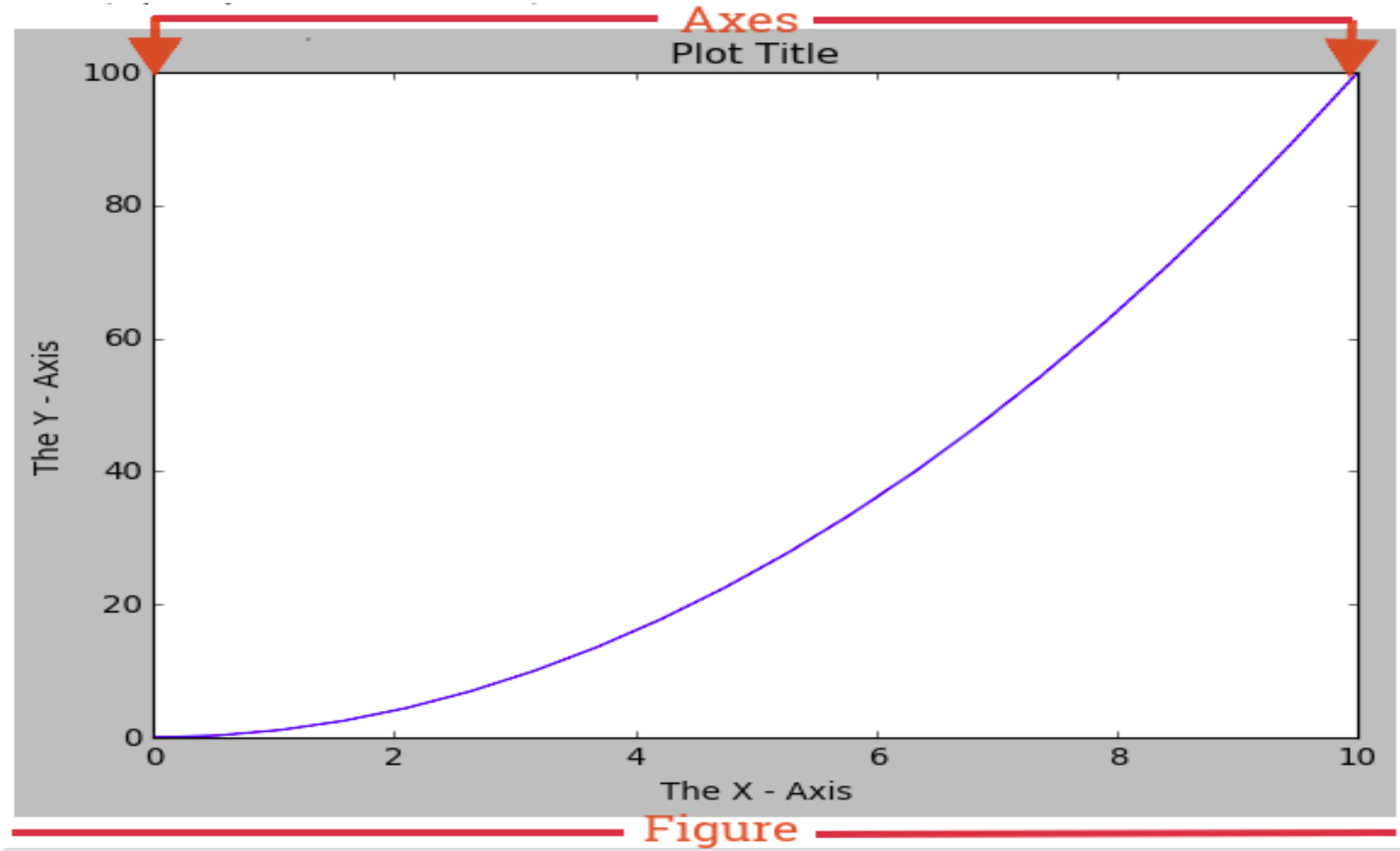
```
# importing matplotlib module
from matplotlib import pyplot as plt

# x-axis values
x = [5, 2, 9, 4, 7]

# Y-axis values
y = [10, 5, 8, 4, 2]

# Function to plot the bar
plt.bar(x,y)

# function to show the plot
plt.show()
```

Histogram

```
#importing matplotlib module
From matplotlib import pyplot as plt

#Y-axis values
y= [10, 5,8, 4, 2]

#Function to plot histogram
plt.hist(y)

#Function to show the plot
plt.show()
```

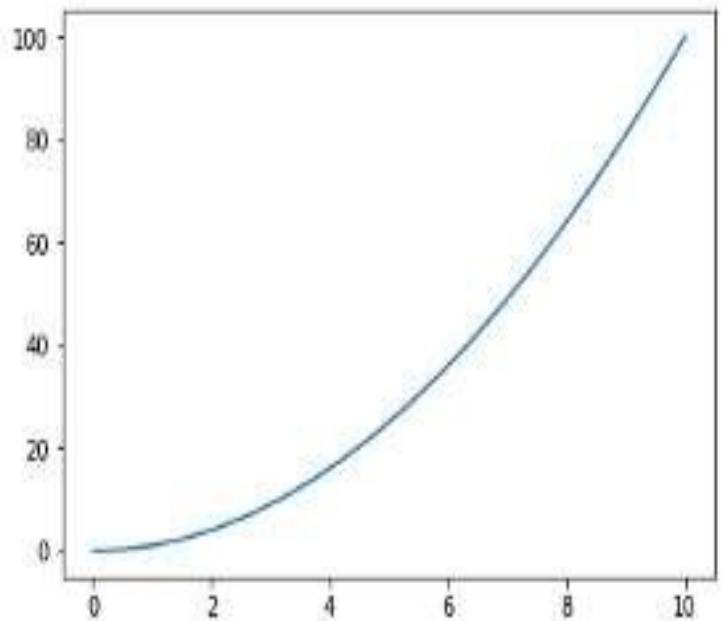
Scatter Plot :

```
# importing matplotlib module
from matplotlib import  pyplot as plt

# x-axis values
x = [5, 2, 9, 4, 7]

# Y-axis values
y = [10, 5, 8, 4, 2]

# Function to plot scatter
plt.scatter(x, y)

# function to show the plot
plt.show()
```


Axes
Plot Title
100
80
60
40
20
0
The Y - Axis
0
2
4
6
8
10
The X - Axis
Figure

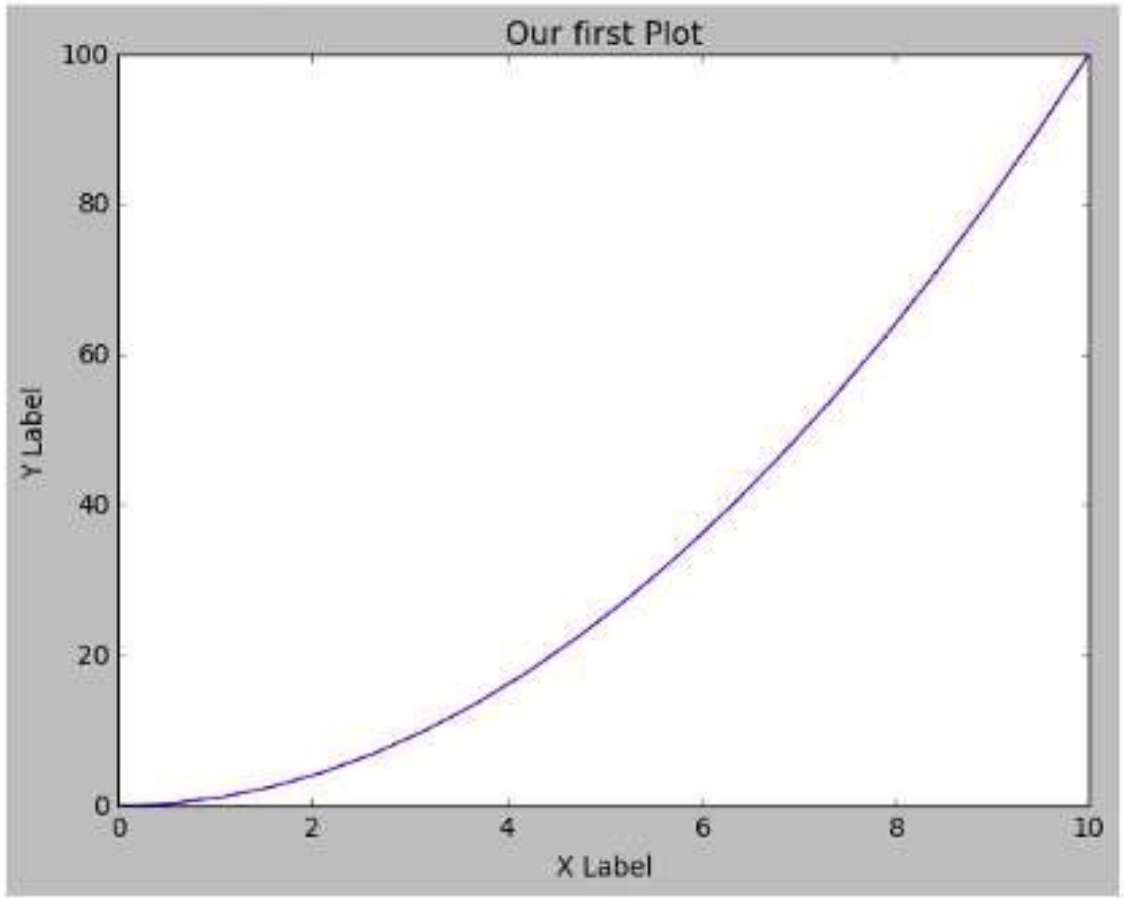
Functional Approach:

```
In [3]: import numpy as np
        x = np.linspace(0, 10, 20) #Generate 20 datapoints between 0 and 10
        y = x**2                   #Generate array 'y' from square of 'x'

In [4]: plt.plot(x,y)

Out[4]: [<matplotlib.lines.Line2D at 0x7f5a339fcd30>]
```



In [12]:
```
plt.plot(x,y)
plt.title('Our first Plot')
plt.xlabel('X Label')
plt.ylabel('Y Label')
```

Out[12]:
```
Text(0,0.5,'Y Label')
```



Matplotlib allows us easily create multi-plots on the same figure using the .subplot() method. This .subplot() method takes in three parameters, namely:

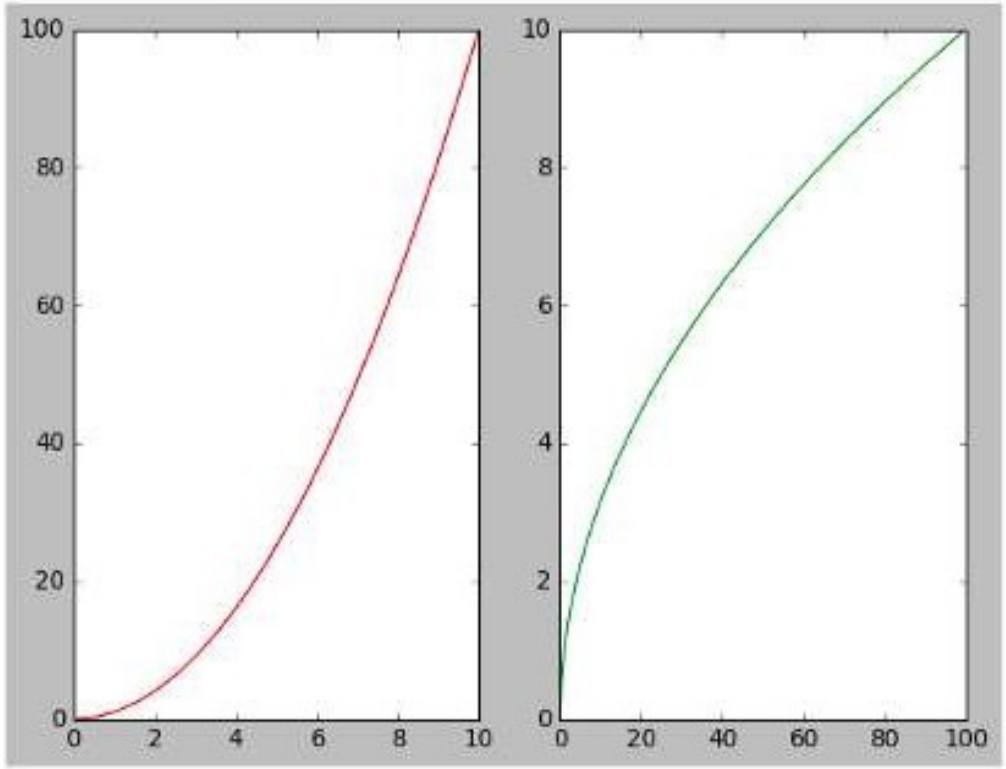
nrows: the number of

rows ncols: the number of columns

plot_number : refers to a specific plot in the Figure.

Using .subplot() we will create a two plots on the same canvas:

In [13]:
```
# plt.subplot(nrows, ncols, plot_number)
plt.subplot(1, 2, 1)
plt.plot(x, y, 'red') # More on color options later

plt.subplot(1,2,2)
plt.plot(y, x, 'green');
```



Object oriented Interface

This is the best way to create plots.
The idea here is to create Figure objects and call methods off it. Let's create a blank Figure using the .figure() method

```
In [14]: fig = plt.figure()

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

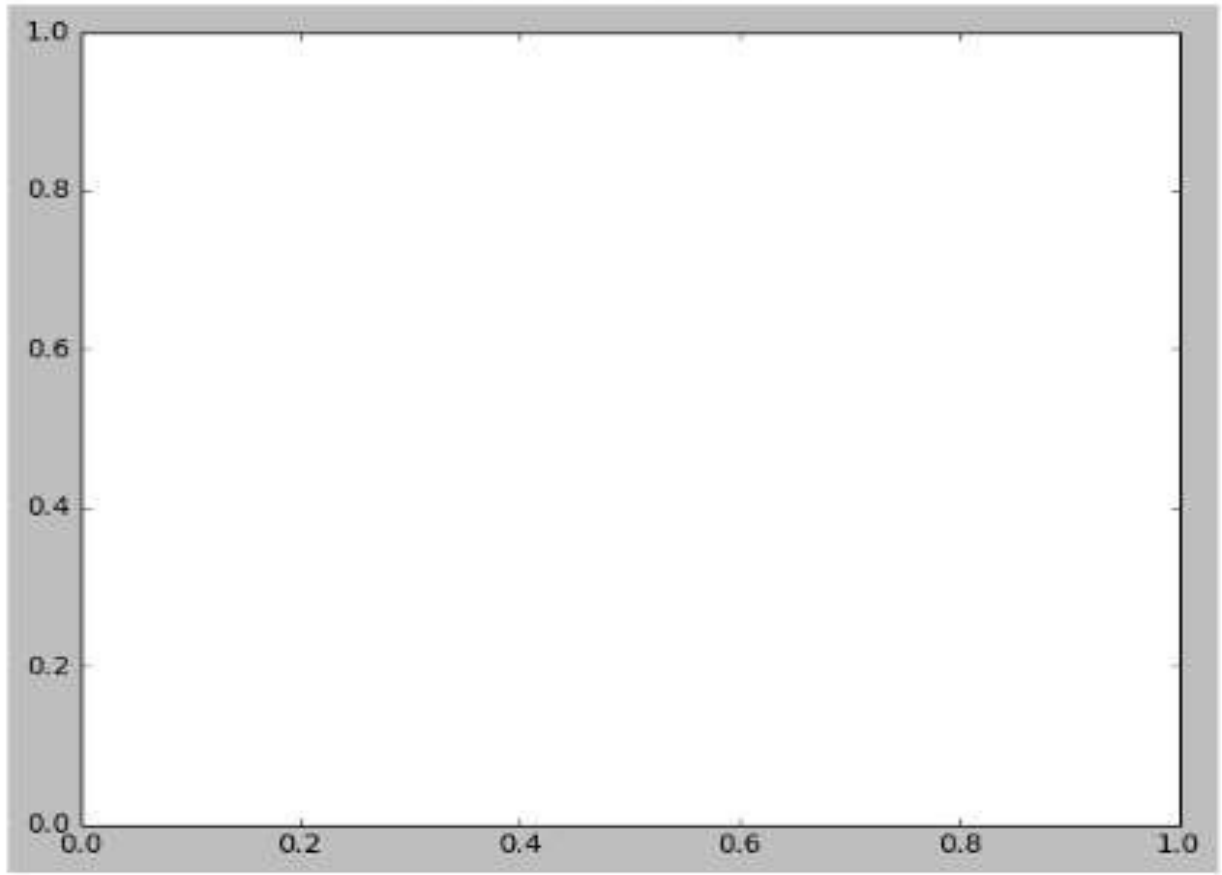
Next step

- Now we need to add a set of axes

.add_axes()

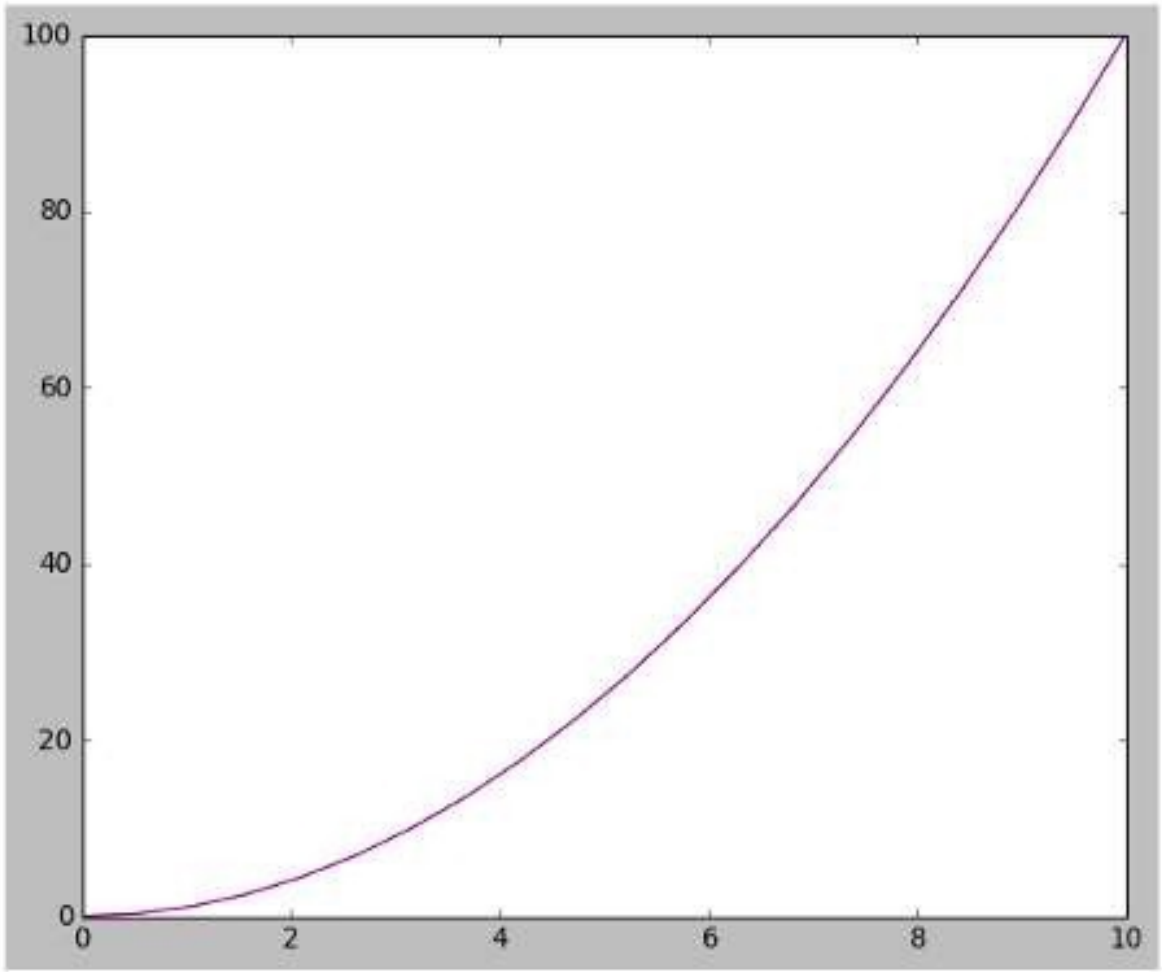
(left, bottom, width, and height)

In [17]:
```
fig = plt.figure()

ax = fig.add_axes([0.1, 0.2, 0.8, 0.9])
```



In [18]:
```
fig = plt.figure()
ax = fig.add_axes([0.1, 0.2, 0.8, 0.9])

ax.plot(x,y, 'purple')
```

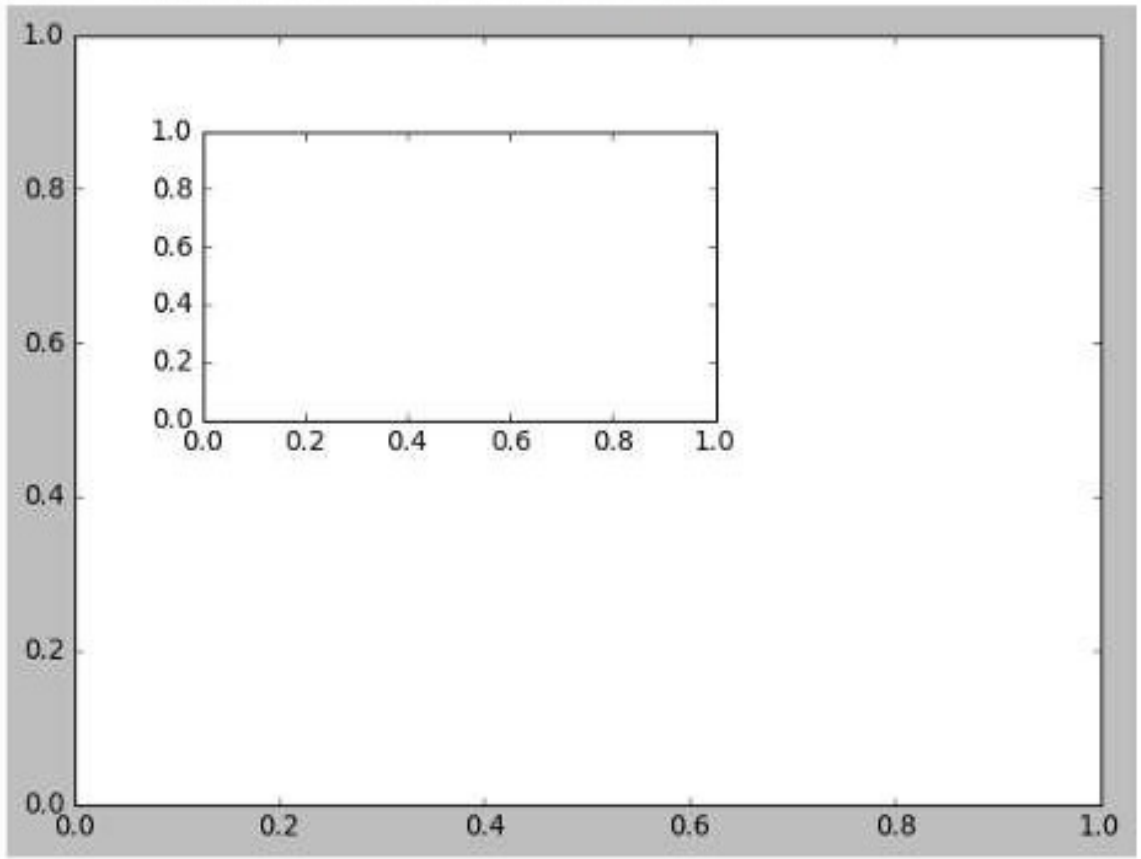
Out[18]:
```
[<matplotlib.lines.Line2D at 0x7f00630b6208>]
```



In [22]:
```
fig = plt.figure()
ax = fig.add_axes([0.1, 0.2, 0.8, 0.9])

ax.plot(x,y, 'purple')
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_title('Our First Plot using Object Oriented Approach')
```

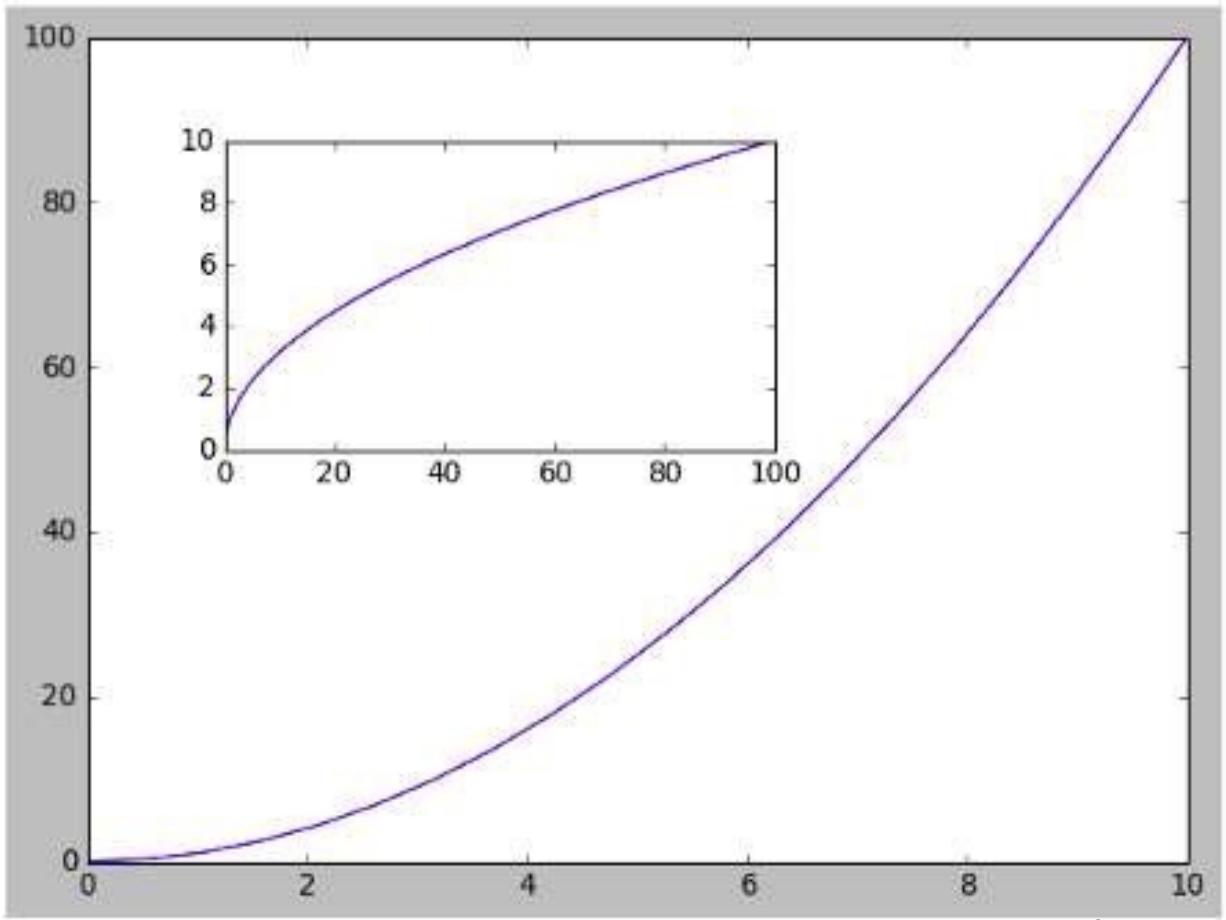
Something interesting ,figure in figure

```
In [24]: fig = plt.figure()

         axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8])
         axes2 = fig.add_axes([0.2, 0.5, 0.4, 0.3])
```



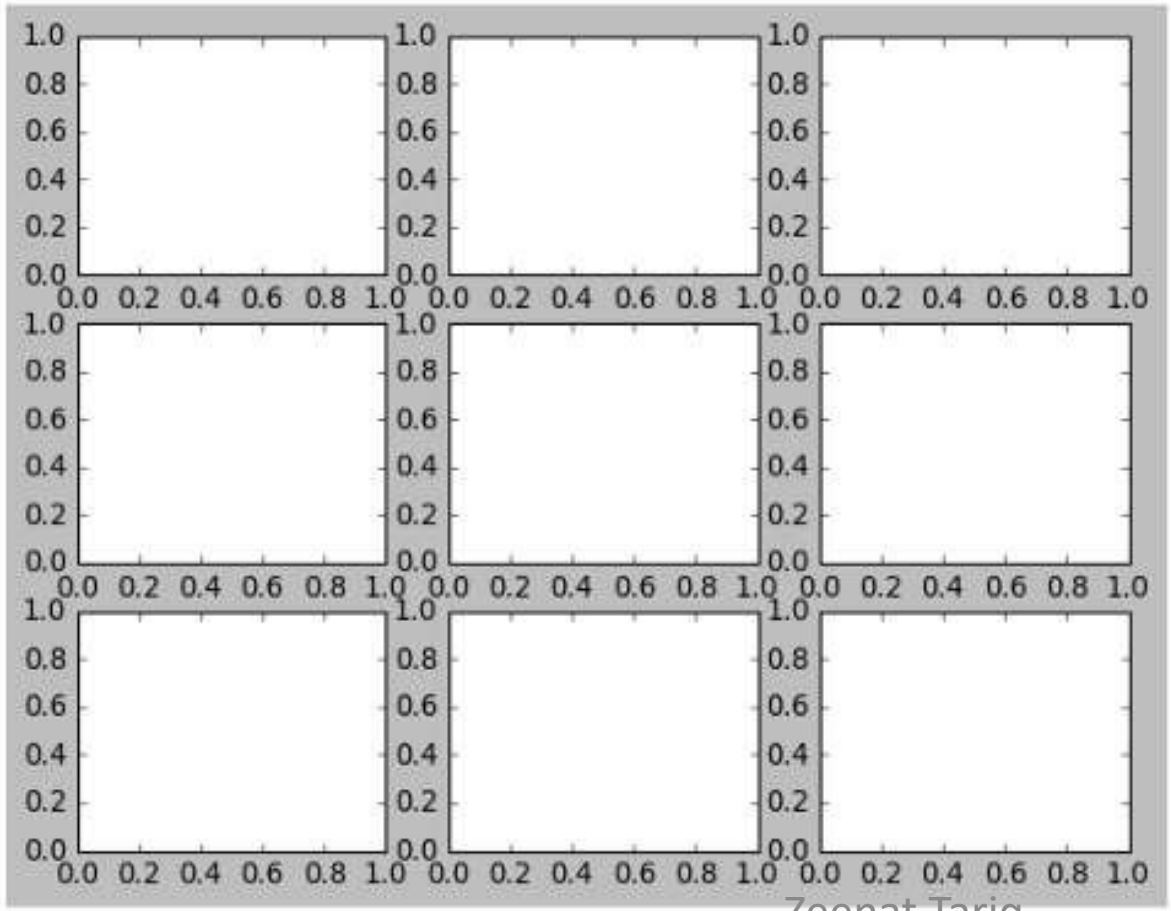
In [26]:
```
fig = plt.figure()

axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8])
axes2 = fig.add_axes([0.2, 0.5, 0.4, 0.3])

axes1.plot(x,y)
axes2.plot(y,x)
```

Out[26]: [<matplotlib.lines.Line2D at 0x7f00630991d0>]



Wecan create a matrix of subplot for example 3*3

```
In [30]: # Empty canvas of 3 by 3 subplots
         fig, axes = plt.subplots(nrows=3, ncols=3)
```



Add, plot.tight_layout

The only difference between plt.figure() and plt.subplots() is that plt.subplots() automatically does what the .add_axes() method of .figure() will do for you based off the number of rows and columns you specify.

In [34]:
```
# Empty canvas of 3 by 3 subplots
fig, ax = plt.subplots(nrows=3, ncols=3)

ax[0,1].plot(x,y)
ax[1,2].plot(y,x)

plt.tight_layout()
```

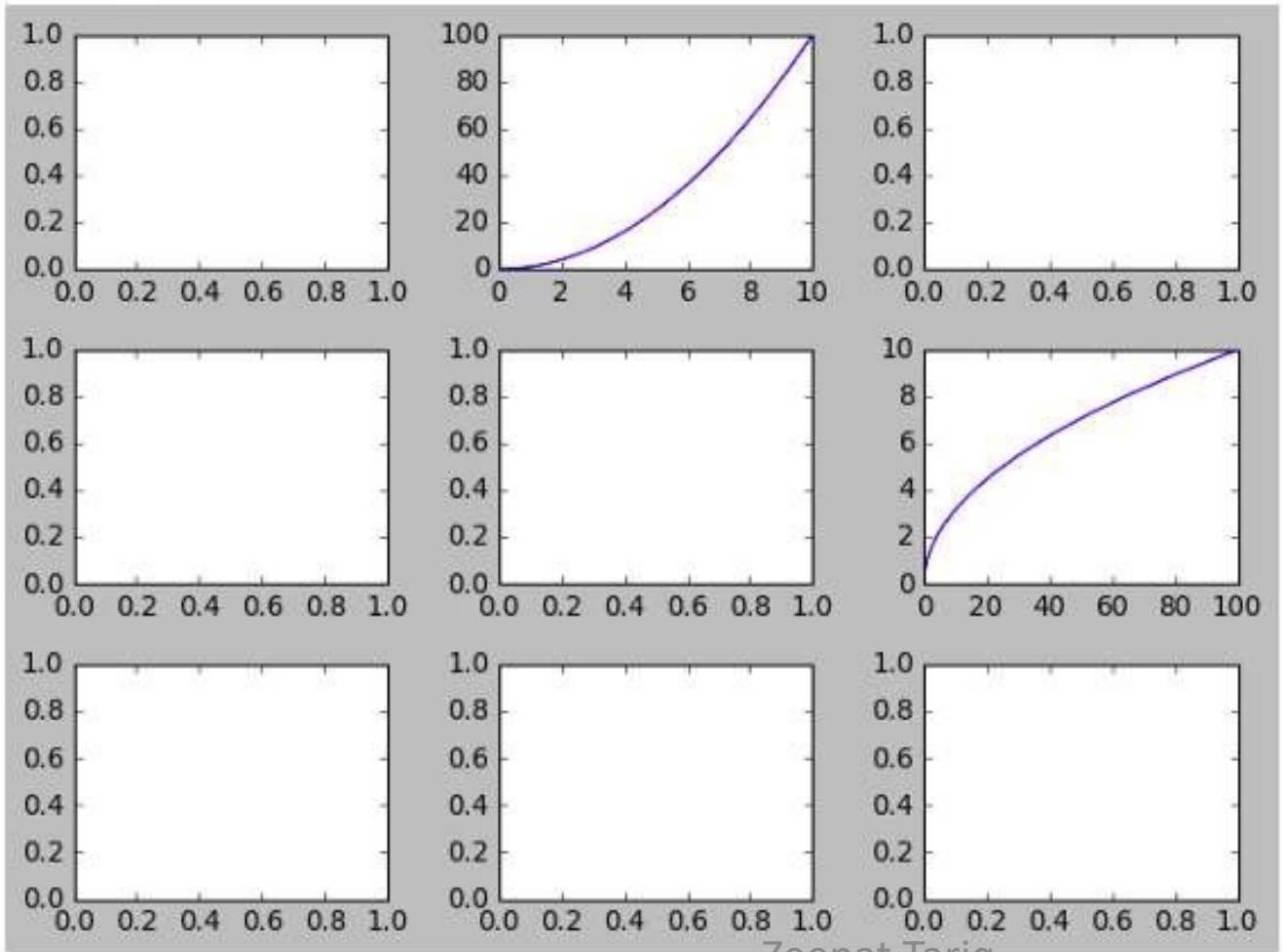
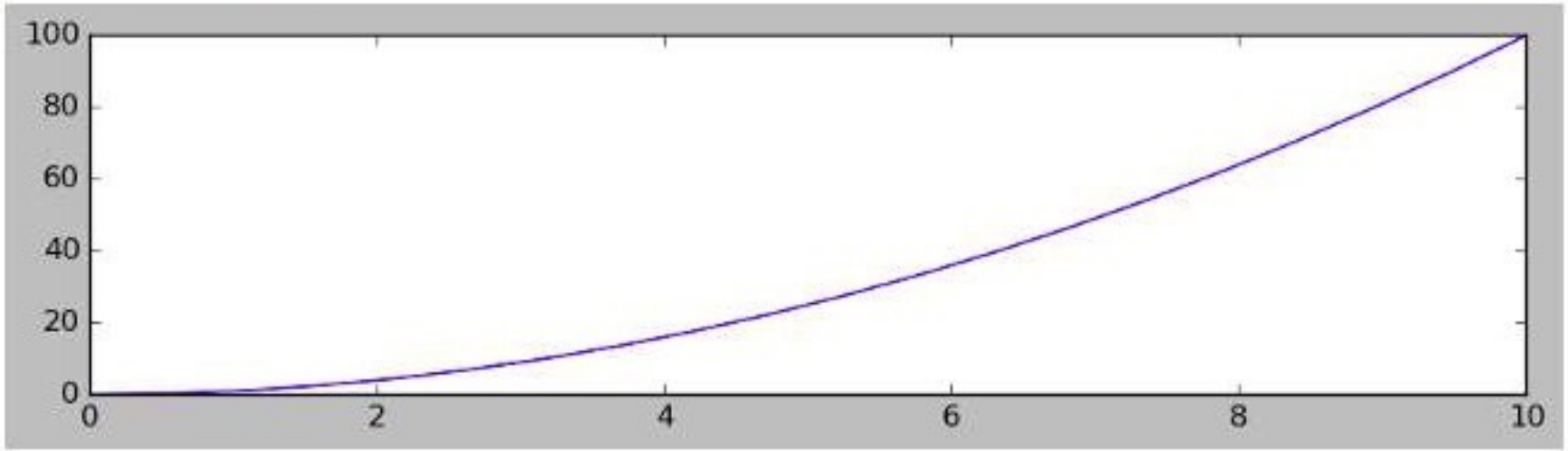


Figure size, aspect ratio, and DPI IN FIGURE

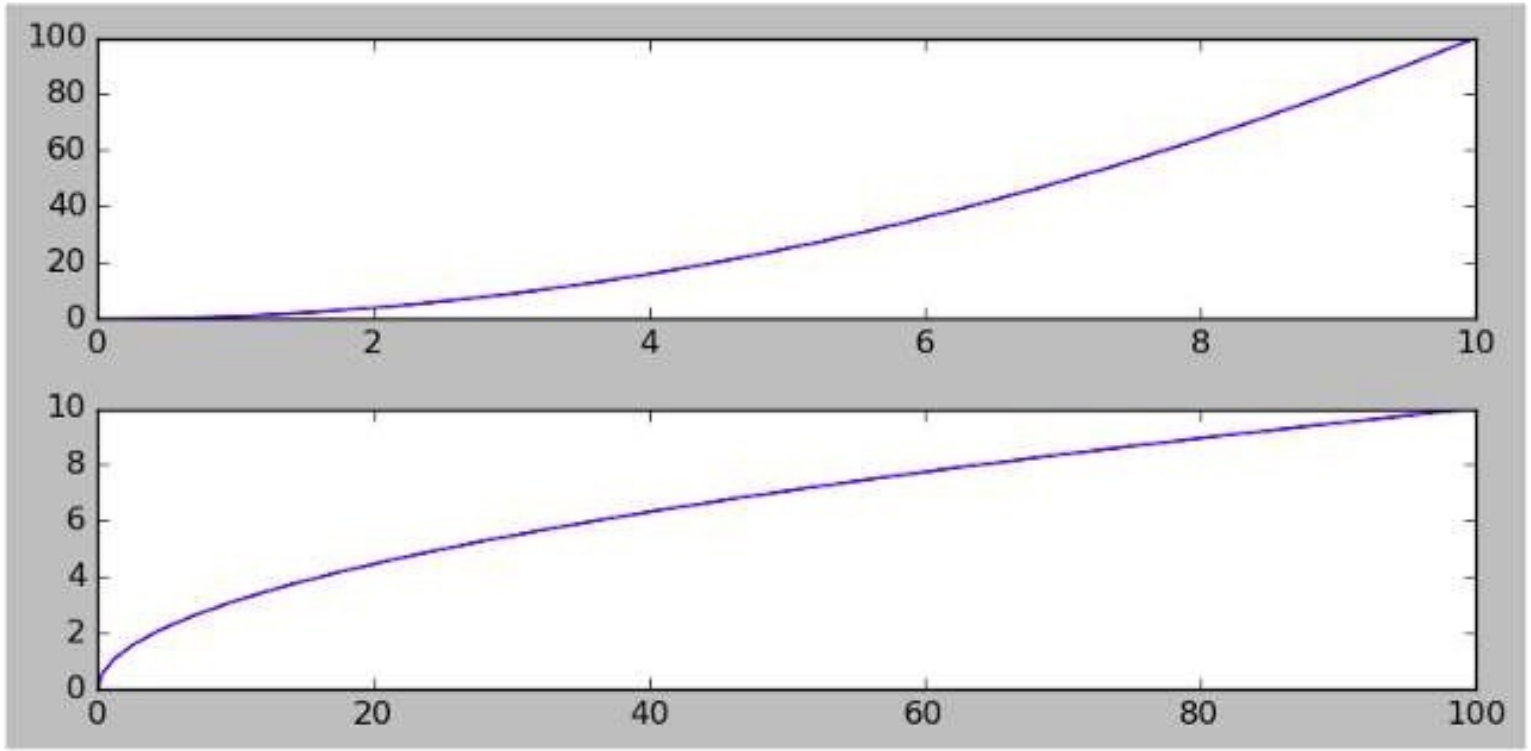
```
In [39]: fig = plt.figure(figsize=(8,2), dpi = 100)

         ax = fig.add_axes([0,0,1,1])
         ax.plot(x,y)
```

```
Out[39]: [<matplotlib.lines.Line2D at 0x7f00624b9a20>]
```



IN SUBPLOTS

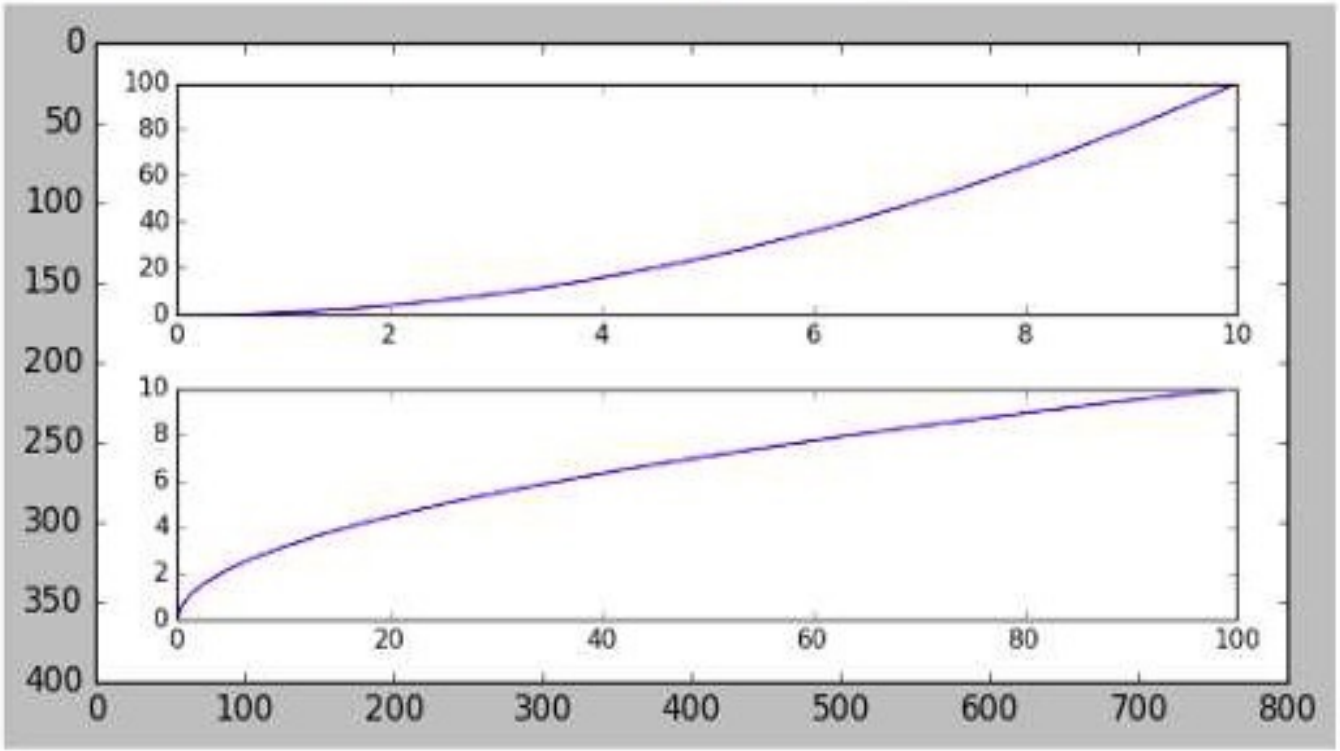
```
In [44]: fig, ax = plt.subplots(nrows=2, ncols=1,figsize=(8,4), dpi = 100)

         ax[0].plot(x,y)
         ax[1].plot(y,x)

         plt.tight_layout()
```



SAVE PICTURE ALSO VIA
FIG.SAVEFIG('NAME.PNG')

```
In [52]: import matplotlib.image as mpimg

         plt.imshow(mpimg.imread('my_figure.png'))
```

```
Out[52]: <matplotlib.image.AxesImage at 0x7f00629afc88>
```

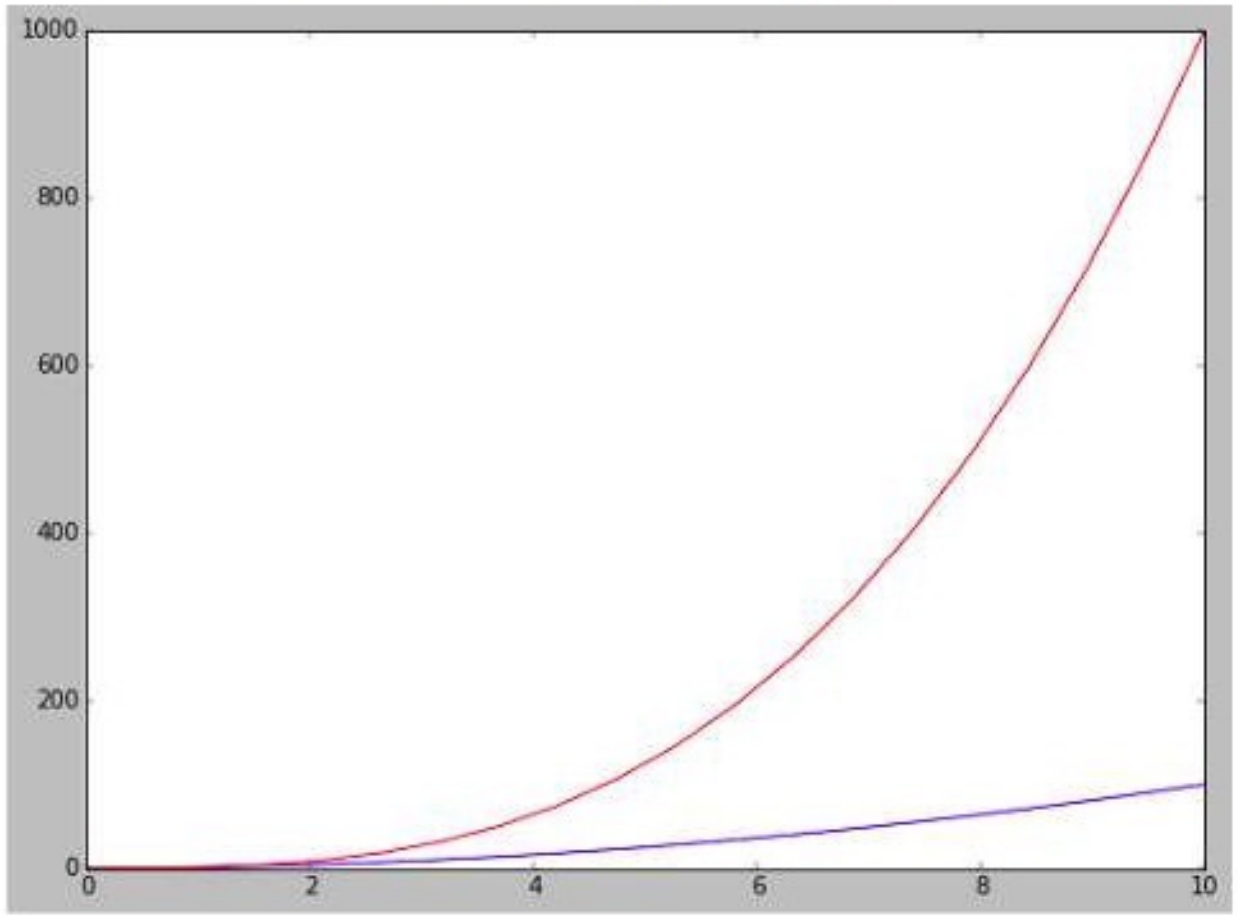


Legends

- Legends allows us to distinguish between plots. With Legends, you can use label texts to identify or differentiate one plot from another. For example, say we have a figure having two plots like below

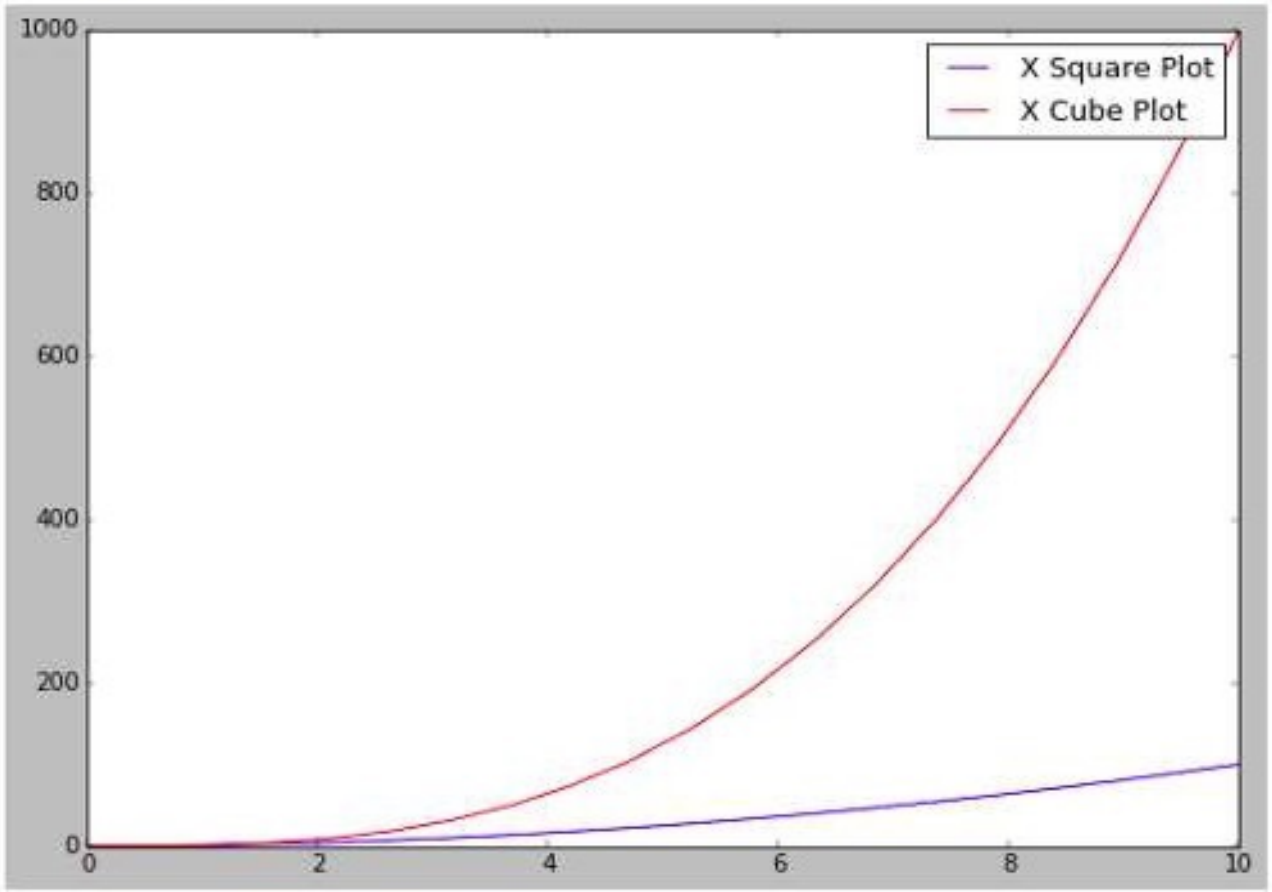
In [74]:
```
fig = plt.figure(figsize=(8,6), dpi = 60)
ax = fig.add_axes([0,0,1,1])
ax.plot(x,x**2)
ax.plot(x,x**3, 'red')
```

Out[74]:
```
[<matplotlib.lines.Line2D at 0x7f0062338f60>]
```



In [75]:
```
fig = plt.figure(figsize=(8,6), dpi = 60)
ax = fig.add_axes([0,0,1,1])
ax.plot(x,x**2, label="X Square Plot")
ax.plot(x,x**3, 'red', label='X Cube Plot')

ax.legend()
```

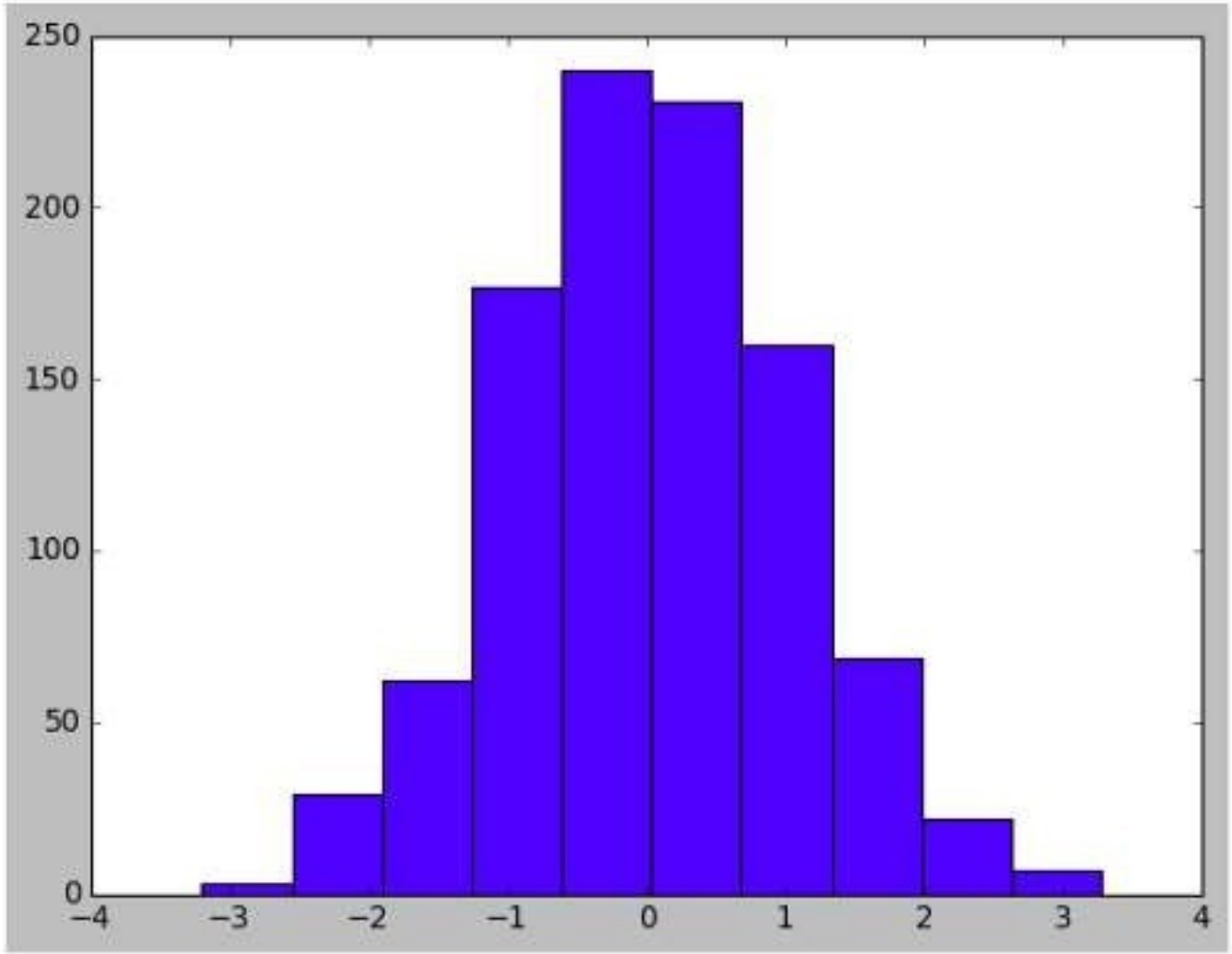
Out[75]: <matplotlib.legend.Legend at 0x7f0061e0ee80>



Plot Types

- Histogram
- Helps us understand the distribution of numeric value in a way that you can not do with mean, median and mode.

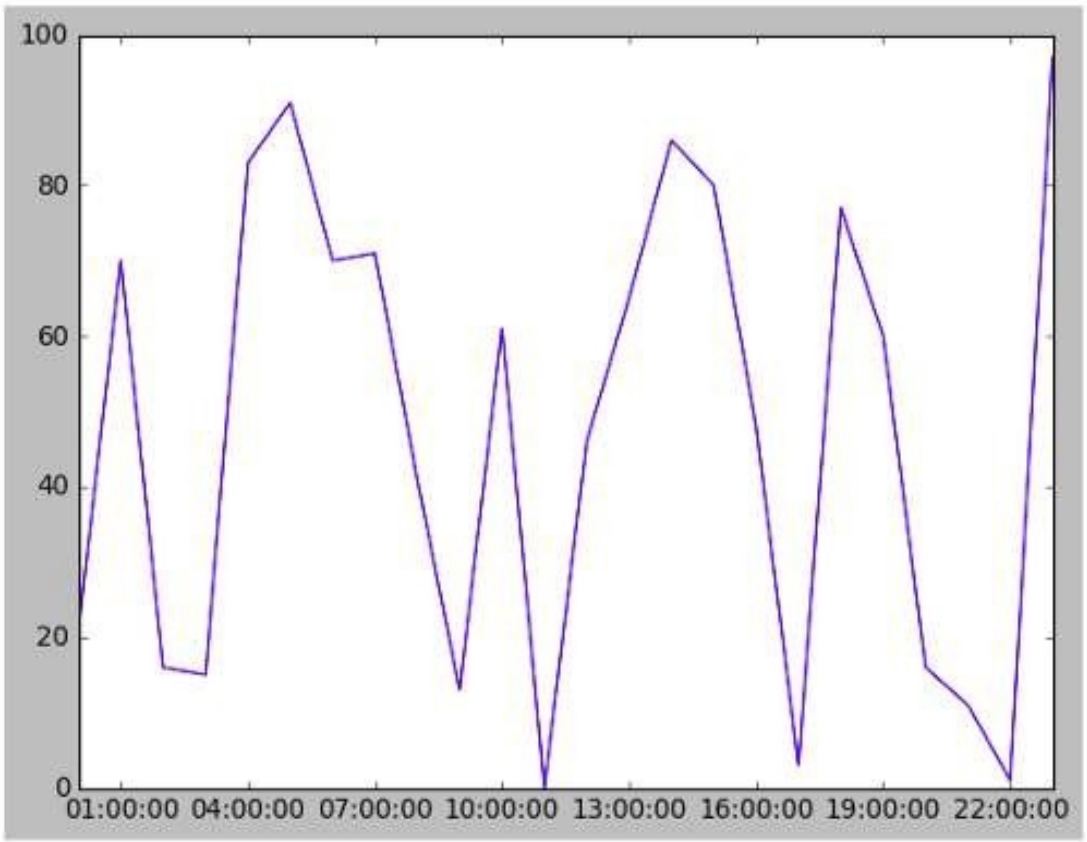
In [90]:
```
x = np.random.randn(1000)
plt.hist(x);
```



Time series (LinePlot)

- A chart that shows a trend over a period of time
- It allows you to test various hypotheses under certain conditions, like what happens different days of the week or between different times of the day

In [227]:
```
import matplotlib.pyplot as plt
import datetime
import numpy as np

x = np.array([datetime.datetime(2018, 9, 28, i, 0) for i in range(24)])
y = np.random.randint(100, size=x.shape)

plt.plot(x,y)
plt.show()
```



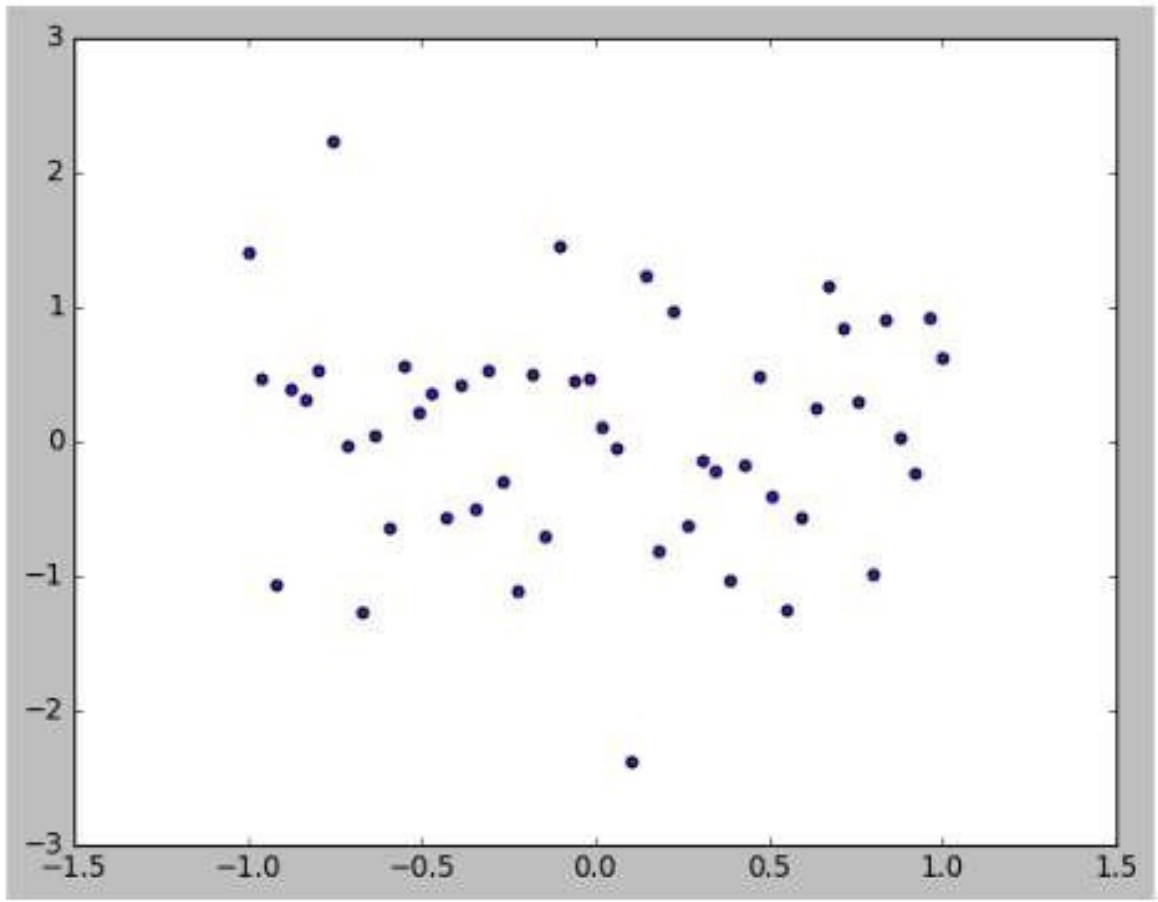
Scatter plots

- offer a convenient way to visualize how two numeric values are related in your data.
- It helps in understanding relationships between multiple variables.
- Using .scatter() method, we can create a scatter plot:

EXAMPLE

In [214]:
```
fig, ax = plt.subplots()
x = np.linspace(-1, 1, 50)
y = np.random.randn(50)
ax.scatter(x, y)
```

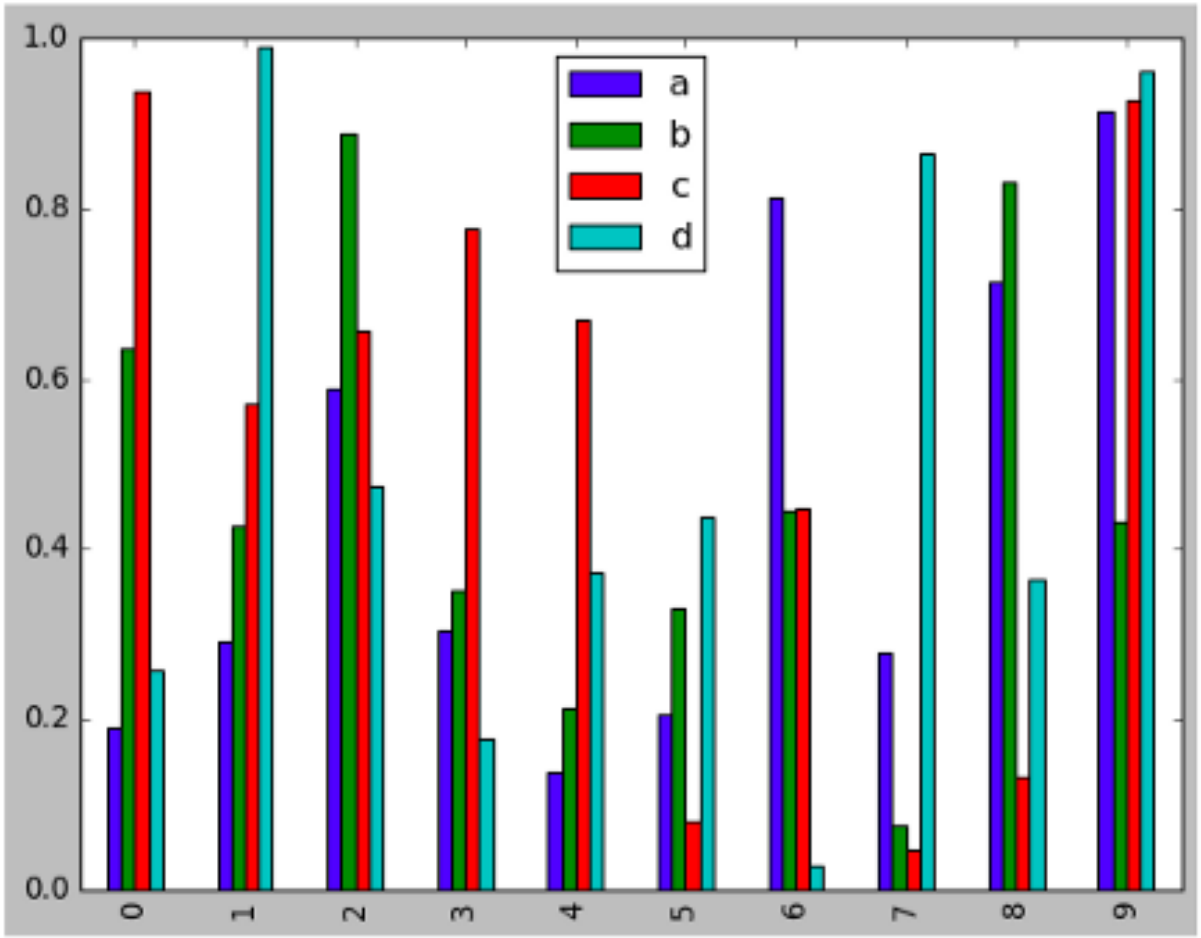
Out[214]: <matplotlib.collections.PathCollection at 0x7f004eaa30f0>



Bar graphs

- are convenient for comparing numeric values of several groups. Using .bar() method, we can create a bar graph:

In [216]:
```
my_df = pd.DataFrame(np.random.rand(10, 4), columns=['a', 'b', 'c', 'd'])

my_df.plot.bar();
```



References

- https://www.simplifiedpython.net/data-visualization-python-tutorial/
- https://matplotlib.org/
- https://mode.com/blog/python-data-visualization-libraries/
- https://towardsdatascience.com/top-6-python-libraries-for-visualization-which-one-to-use-fe43381cd658