Data Visualization

- □ What is data visualization?

  Data visualization is the presentation of data in a pictorial or graphical format.

- □ Why we need data visualization?

  We need visualization because human brain process information easily when it is in pictorial or graphical from.

  Data visualization allows us to quickly interpret the data and adjust different variables to see their effect.

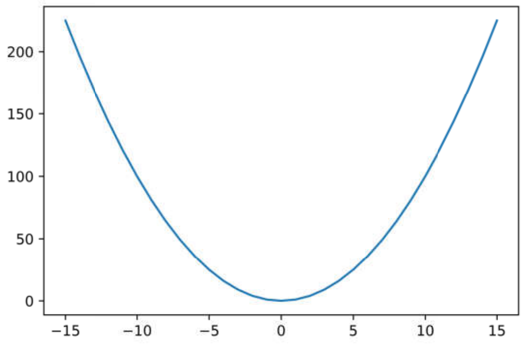
Introduction to Matplotlib

- □ Matplotlib is a python visualization/plotting library.
- □ It was introduced by John Hunter in the year 2002.
- □ Matplotlib comes with a wide variety of plots e.g. line plot, bar plot, scatter plot, histogram plot etc.
- □ Plots helps to understand trends, patterns, and to make correlations in data.
- □ pyplot is matplotlib's plotting framework. It consist command style functions that makes matplotlib work like MATLAB.
- □ We can import pyplot framework in following ways:

```python
import matplotlib.pyplot as plt
# or
from matplotlib import pyplot as plt
```

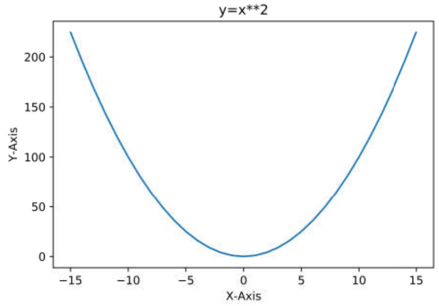
Line Plot

```python
import matplotlib.pyplot as plt
X = range(-15,16)
Y = [x**2 for x in X]
plt.plot(X, Y)
plt.show()
```

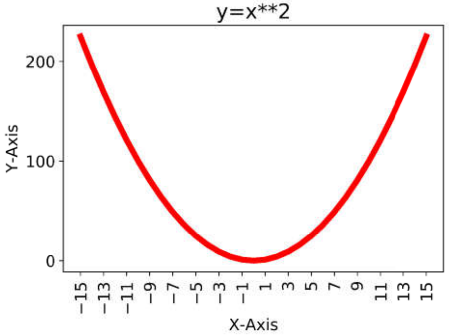


Adding title and label to graph

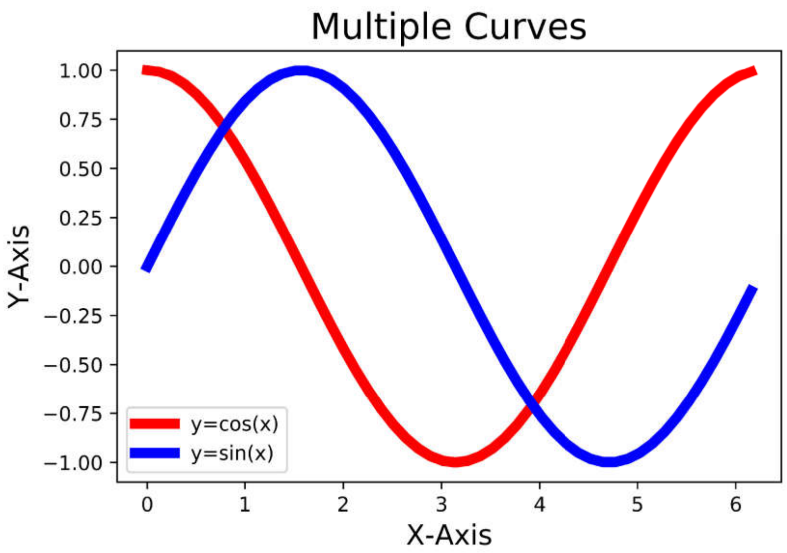
```python
import matplotlib.pyplot as plt
X = range(-15,16)
Y = [x**2 for x in X]
plt.plot(X, Y)
plt.title('y=x**2')
plt.xlable('X-Axis')
plt.ylable('Y-Axis')
plt.show()
```



```
import matplotlib.pyplot as plt
X = range(-15,16)
Y = [x**2 for x in X]
plt.plot(X, Y, 'r', linewidth=5.2)
plt.title('y=x**2', fontsize=18)
plt.xlabel('X-Axis', fontsize=14)
plt.ylabel('Y-Axis', fontsize=14)
plt.xticks(range(-15,16,2), fontsize=14, rotation=90)
plt.yticks([0, 100,  200], fontsize=14, rotation=0)
plt.show()
```



Multiple Curves Plotting



Multiple Curves Plotting

```
import math
import matplotlib.pyplot as plt
T = range(50)
X = [(2*math.pi * t)/len(T) for t in T]
Y1 = [math.cos(i) for i in X]
Y2 = [math.sin(i) for i in X]
plt.plot(X, Y1, 'r', linewidth=5.2, label = 'y=cos(x)')
plt.plot(X, Y2, 'b', linewidth=5.2, label = 'y=sin(x)')
plt.title('Multiple Curves', fontsize=18)
plt.xlabel('X-Axis', fontsize=14)
plt.ylabel('Y-Axis', fontsize=14)
plt.legend()
plt.show()
```

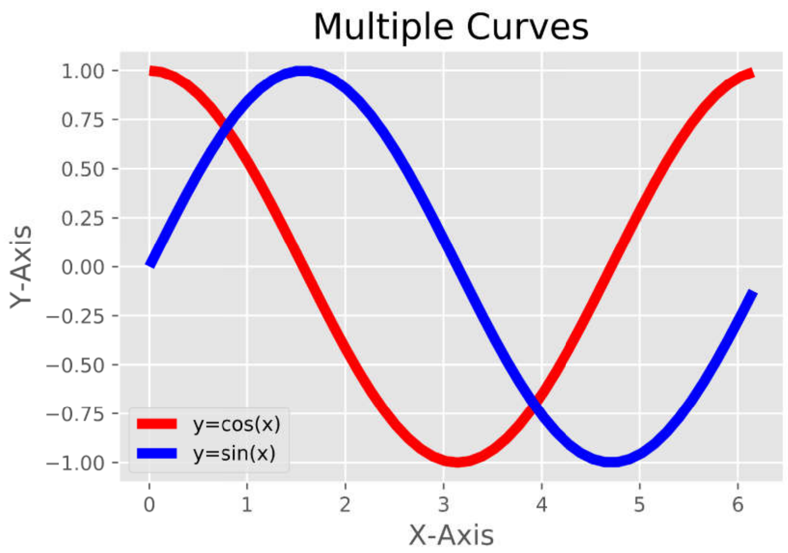
Adding Style To Graph

- □ Matplotlib offers about 26 different types of plotting style sheets.
- □ We can check the available plotting style sheets using statement

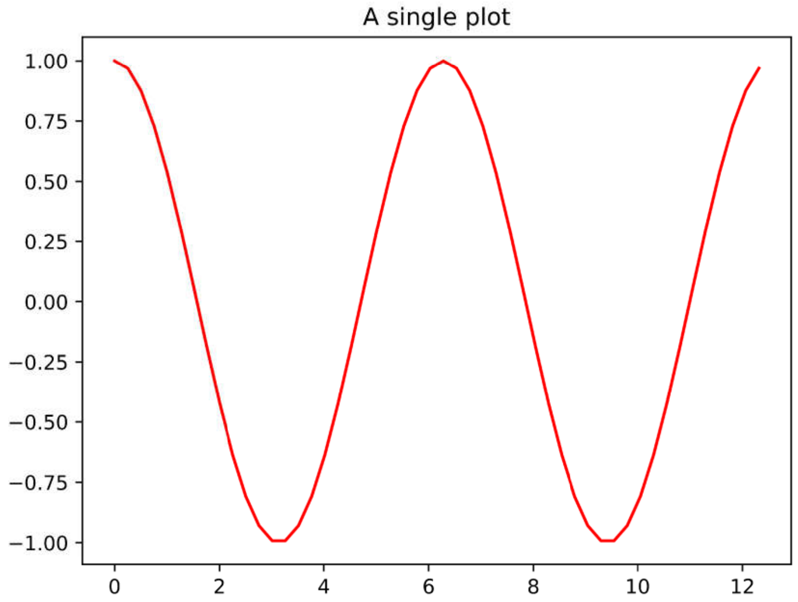
  print(plt.style.available)

```
import math
import matplotlib.pyplot as plt
plt.style.use('ggplot')
T = range(50)
X = [(2 * math.pi * t)/len(T) for t in T]
Y1 = [math.cos(i) for i in X]
Y2 = [math.sin(i) for i in X]
plt.plot(X, Y1, 'r', linewidth=5.2, label = 'y=cos(x)')
plt.plot(X, Y2, 'b', linewidth=5.2, label = 'y=sin(x)')
plt.title('Multiple Curves', fontsize=18)
plt.xlabel('X-Axis', fontsize=14)
plt.ylabel('Y-Axis', fontsize=14)
plt.legend()
plt.show()
```

Adding Style To Graph



A figure with just one subplot



A figure with just one subplot

- □ pyplot.subplots() is used in python to create multiple plots in one plot.
- □ pyplot.subplots() without arguments returns a Figure and a single Axes.

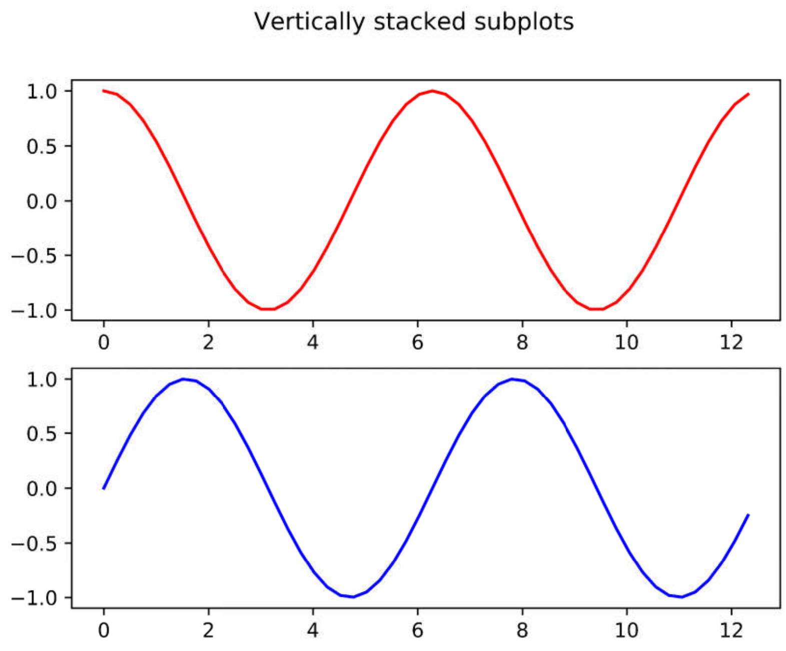
```
import math
import matplotlib.pyplot as plt
T = range(50)
X = [(4 * math.pi * t)/len(T) for t in T]
Y1 = [math.cos(i) for i in X]

fig, ax = plt.subplots()
ax.plot(X, Y1,'r')
ax.set_title('A single plot')
plt.show()
```

Stacking subplots in one direction

- □ The first two optional arguments of pyplot.subplots define the number of rows and columns of the subplot grid.
- □ When stacking in one direction only, the returned axs is a 1D numpy array containing the list of created Axes.

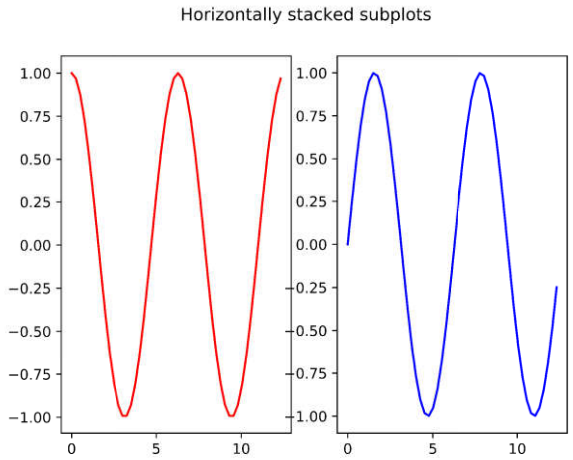
```
import math
import matplotlib.pyplot as plt
T = range(50)
X = [(4*math.pi * t)/len(T) for t in T]
Y1 = [math.cos(i) for i in X]
Y2 = [math.sin(i) for i in X]

fig, axs = plt.subplots(2)
fig.suptitle('Vertically stacked subplots')
axs[0].plot(X, Y1, 'r')
axs[1].plot(X, Y2, 'b')
plt.show()
```

A figure with just one subplot



Stacking subplots in one direction

```
fig, axs = plt.subplots(1,2)
fig.suptitle('Horizontally stacked subplots')
axs[0].plot(X, Y1, color = 'r')
axs[1].plot(X, Y2, color = 'b')
plt.show()
```

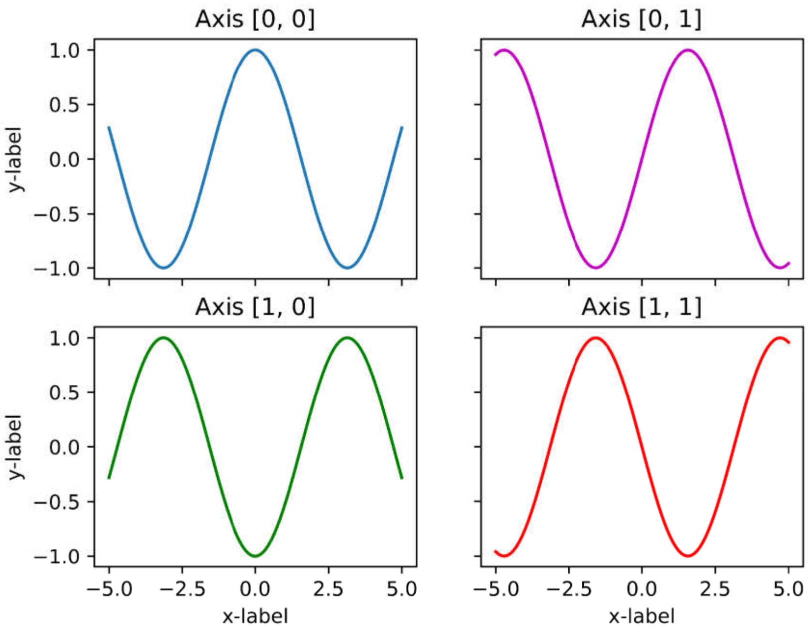


Stacking subplots in two directions

- □ When stacking in two directions, the returned axs is a 2D NumPy array.

```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(-5,5,100) # (start,stop,points)
Y1 = np.array([np.cos(x) for x in X])
Y2 = np.array([np.sin(x) for x in X])
fig, axs = plt.subplots(2, 2)
axs[0, 0].plot(X, Y1)
axs[0, 0].set_title('Axis [0, 0]')
axs[0, 1].plot(X, Y2, 'm')
axs[0, 1].set_title('Axis [0, 1]')
axs[1, 0].plot(X, -Y1, 'g')
axs[1, 0].set_title('Axis [1, 0]')
axs[1, 1].plot(X, -Y2, 'r')
axs[1, 1].set_title('Axis [1, 1]')

for ax in axs.flat:
    ax.set(xlabel='x-label', ylabel='y-label')
# Hide x labels and tick labels for top plots and y ticks for right plots.
for ax in axs.flat:
    ax.label_outer()
```

Stacking subplots in two directions



Sharing axes

- □ By default, each Axes is scaled individually. Thus, if the ranges are different the tick values of the subplots do not align.
- □ Use of sharex and sharey allows aligning of horizontal and vertical axis.

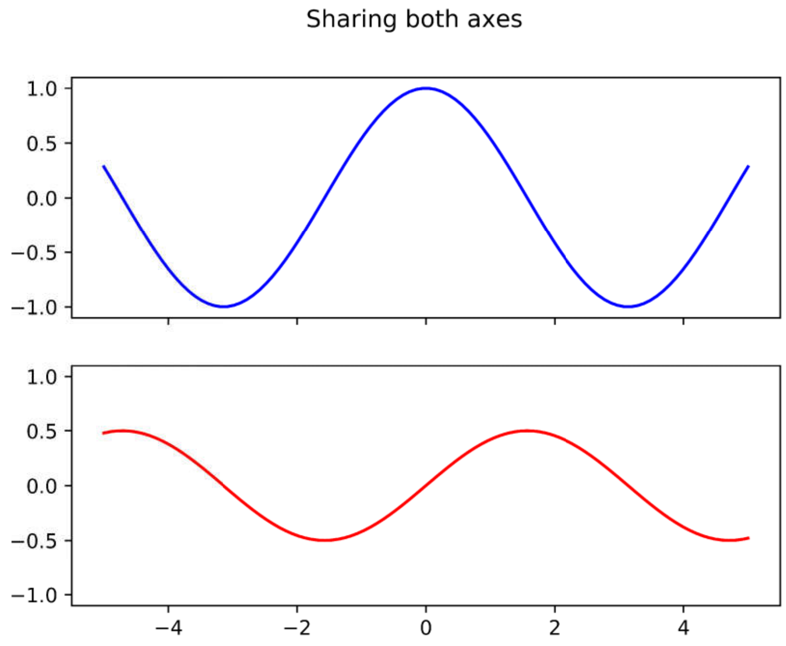
```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(-5,5,100) # (start,stop,points)
Y1 = np.array([np.cos(x) for x in X])
Y2 = np.array([np.sin(x) for x in X])
fig, axs = plt.subplots(2, sharex=True, sharey=True)
fig.suptitle('Sharing both axes')
axs[0].plot(X, Y1,'b')
axs[1].plot(X, 0.5*Y2, 'r')
plt.show()
```

Sharing both axes

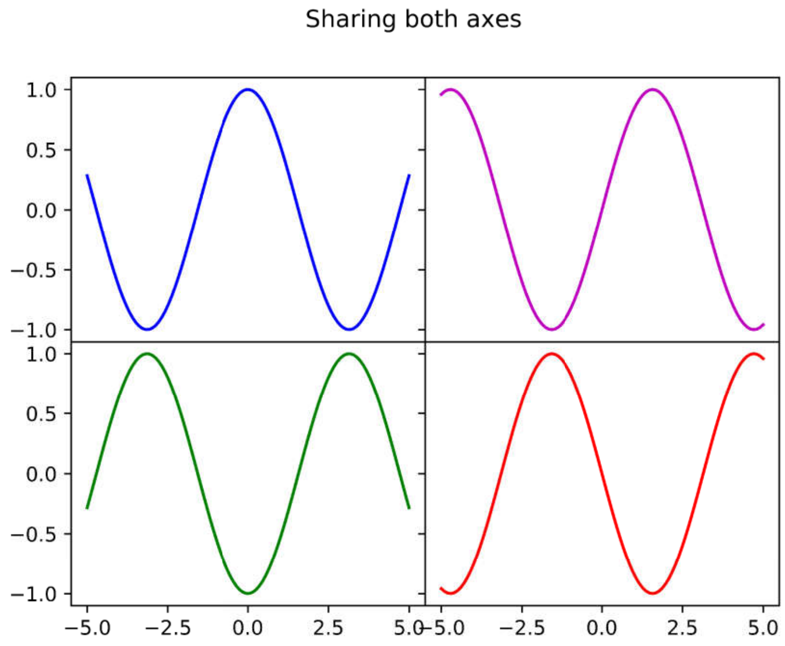
- □ We can reduce the height between vertical and horizontal subplots using gridspec_kw={'hspace': 0, 'wspace': 0}, where optional 'hspace' denotes horizontal space and optional 'wspace' vertical space.

```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(-5,5,100) # (start,stop,points)
Y1 = np.array([np.cos(x) for x in X])
Y2 = np.array([np.sin(x) for x in X])
fig, axs = plt.subplots(2, 2, sharex=True, sharey=True,
    gridspec_kw={'hspace': 0, 'wspace': 0})
fig.suptitle('Sharing both axes')
axs[0, 0].plot(X, Y1,'b')
axs[0, 1].plot(X, Y2, 'm')
axs[1, 0].plot(X, -Y1, 'g')
axs[1, 1].plot(X, -Y2, 'r')
# Hide x labels and tick labels for top plots and y ticks
    for right plots.
for ax in axs.flat:
    ax.label_outer()
plt.show()
```

Sharing both axes



Sharing both axes



Scatter Plot

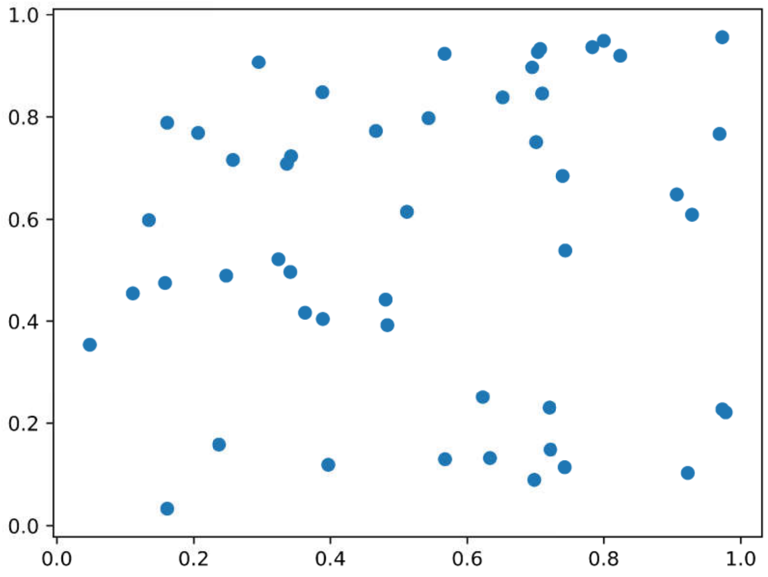
- □ Scatter plot shows the data as collection of points.
- □ The position of a point depends on its two-dimensional value, where each value is a position on either the horizontal or vertical dimension.

```
import numpy as np
import matplotlib.pyplot as plt
# Fixing random state for reproducibility
np.random.seed(19680801)
N = 50
x = np.random.rand(N)
y = np.random.rand(N)
plt.scatter(x, y)
plt.show()
```

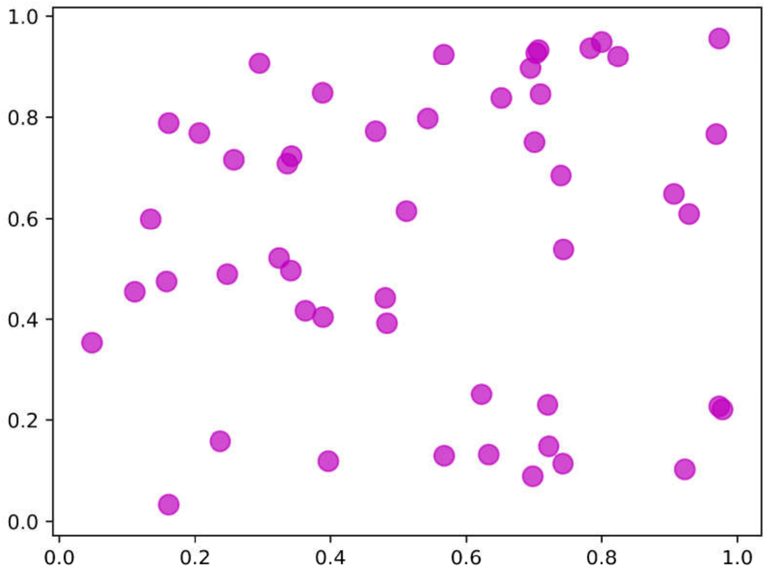
Scatter Plot

```
import numpy as np
import matplotlib.pyplot as plt
# Fixing random state for reproducibility
np.random.seed(19680801)
N = 50
x = np.random.rand(N)
y = np.random.rand(N)
plt.scatter(x, y, marker = 'o', s = 100, c = 'm', alpha =
    0.7)
# s represent area of the marker,  c for colors and alpha
    is blending value, between 0 (transparent) and 1
    (opaque)
plt.show()
```

Scatter Plot



Scatter Plot



Scatter Plot

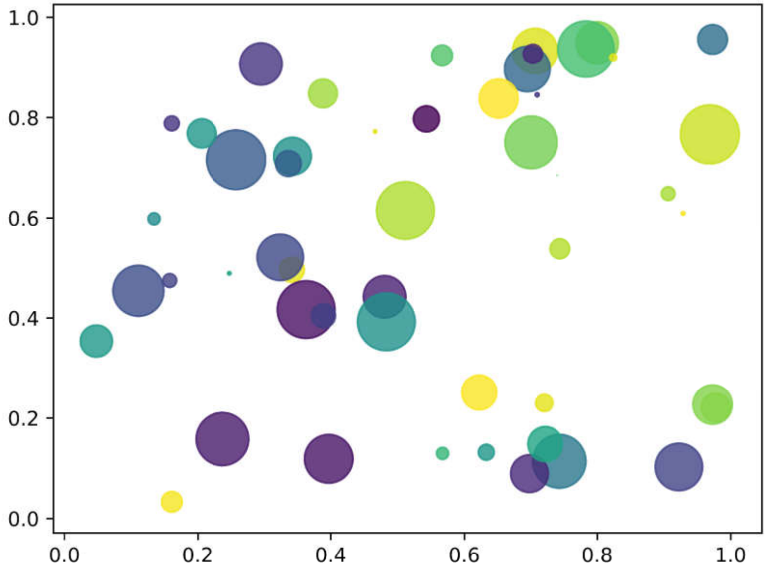
```
import numpy as np
import matplotlib.pyplot as plt
# Fixing random state for reproducibility
np.random.seed(19680801)
N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = (30 * np.random.rand(N))**2  # 0 to 15 point radii
    circles
plt.scatter(x, y, marker = 'o', s = area, c = colors, alpha
    = 0.8)
# s represent area of the marker,  c for colors and alpha
    is blending value, between 0 (transparent) and 1
    (opaque)
plt.show()
```

Bar Chart

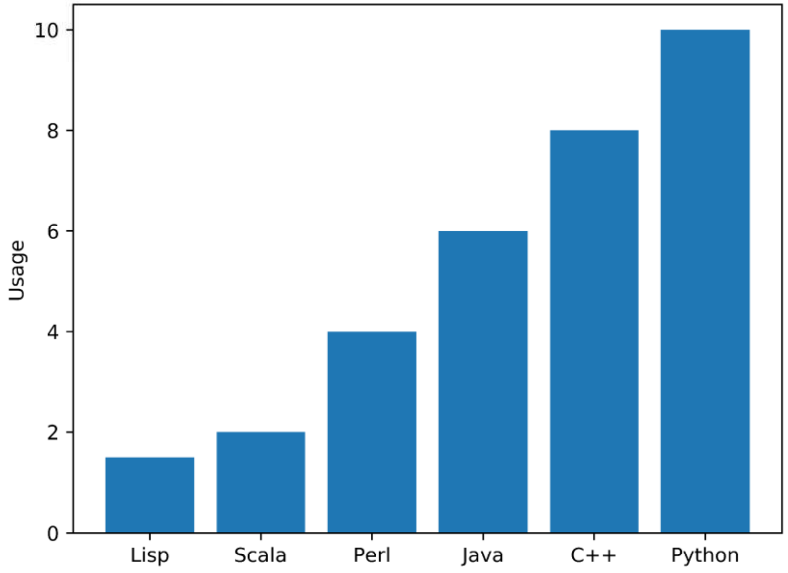
- □ A bar chart is a graph with rectangular bars.
- □ A bar chart is used to show the relationship between numerical value and categorical variable.

```
import matplotlib.pyplot as plt
import numpy as np

objects = ('Lisp', 'Scala', 'Perl', 'Java', 'C++',
    'Python')
y_pos = np.arange(len(objects))
Usage = [1.5, 2, 4, 6, 8, 10]
plt.bar(y_pos, Usage,  align='center', alpha=1)
plt.xticks(y_pos, objects)
plt.ylabel('Usage')
plt.show()
```

Scatter Plot



Bar Chart



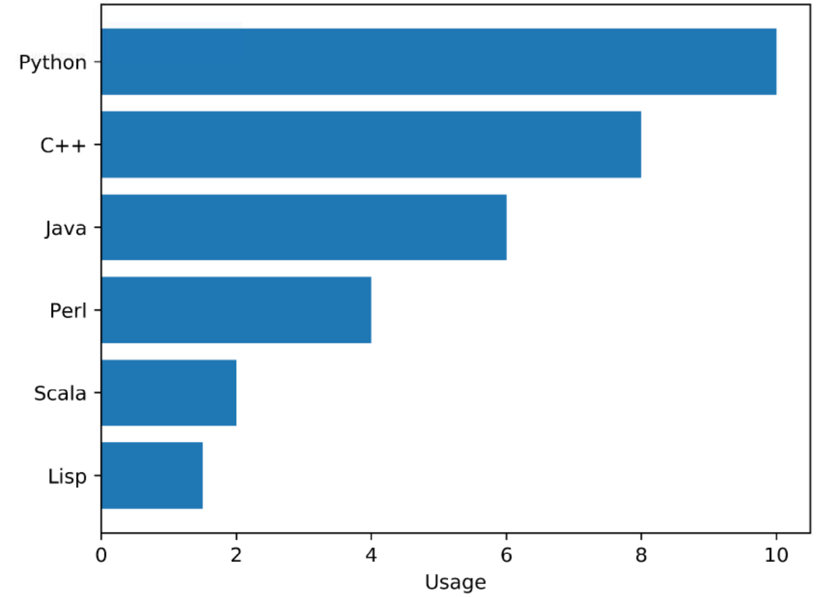
Horizontal Bars

```
import matplotlib.pyplot as plt
import numpy as np

objects = ('Lisp', 'Scala', 'Perl', 'Java', 'C++',
    'Python')
y_pos = np.arange(len(objects))
Usage = [1.5, 2, 4, 6, 8, 10]

plt.barh(y_pos, Usage,  align='center', alpha=1)
plt.yticks(y_pos, objects)
plt.xlabel('Usage')
plt.show()
```

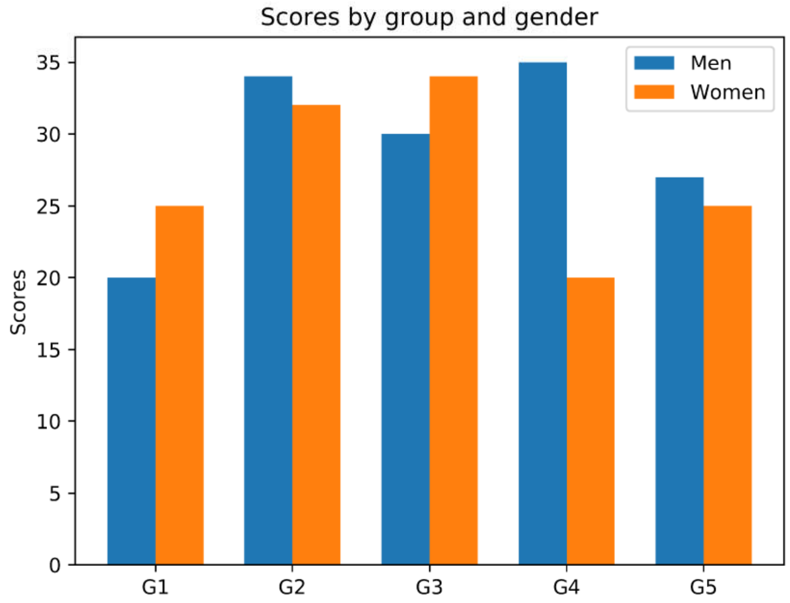
Multiple Bar Plot

```
import matplotlib.pyplot as plt
import numpy as np

labels = ['G1', 'G2', 'G3', 'G4', 'G5']
Men_means = [20, 34, 30, 35, 27]
Women_means = [25, 32, 34, 20, 25]
X = np.arange(len(labels))  # the label locations
Width = 0.35  # the width of the bars

fig, ax = plt.subplots()
ax.bar(X - Width/2, Men_means, Width, label='Men')
ax.bar(X + Width/2, Women_means, Width, label='Women')
ax.set_ylabel('Scores')
ax.set_title('Scores by group and gender')
ax.set_xticks(X)
ax.set_xticklabels(labels)
ax.legend()
plt.show()
```

Horizontal Bars



Multiple Bar Plot



Stacked Bar Plot

```python
import matplotlib.pyplot as plt
import numpy as np

labels = ['G1', 'G2', 'G3', 'G4', 'G5']
Men_means = [20, 35, 30, 35, 27]
Women_means = [25, 32, 34, 20, 25]
X = np.arange(len(labels))  # the label locations
Width = 0.35  # the width of the bars

fig, ax = plt.subplots()
ax.bar(labels, Men_means, Width, label='Men')
ax.bar(labels, Women_means, Width, bottom = Men_means,
    label='Women')
ax.set_ylabel('Scores')
ax.set_title('Scores by group and gender')
ax.legend()
plt.show()
```

Histogram

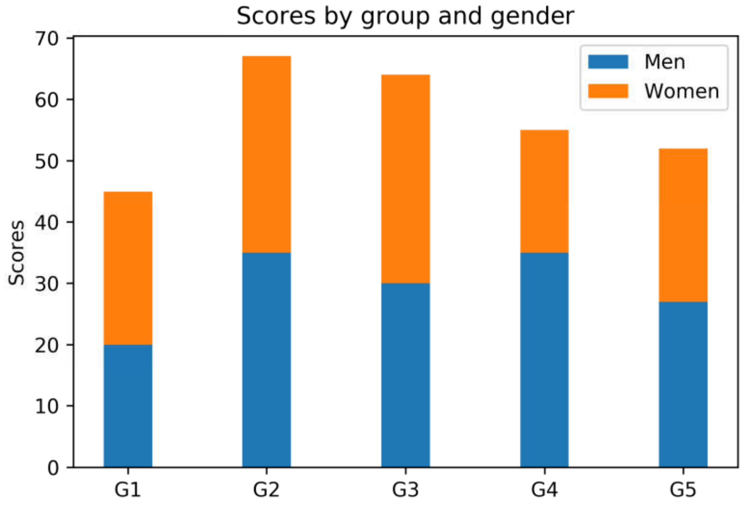
□ A histogram is used to represents the frequency distribution of the numerical data.

□ Usually a histogram has bins, where every bin has a minimum and maximum value.

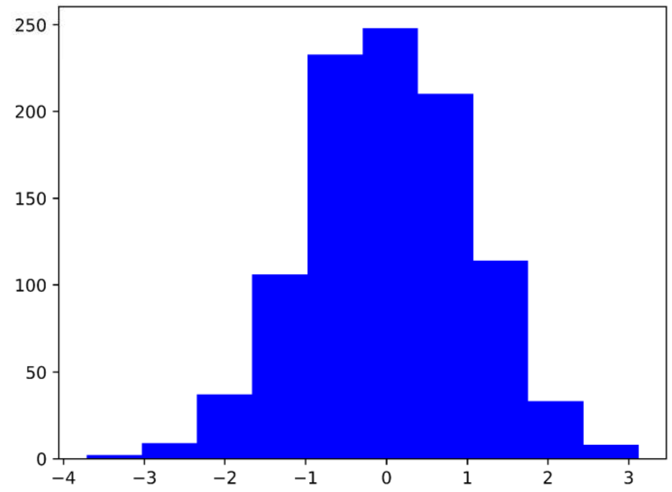
□ The difference between histogram and bar chart is that, histogram presents numerical data whereas bar graph shows categorical data.

□ The histogram is drawn in such a way that there is no gap between the bars.

Stacked Bar Plot



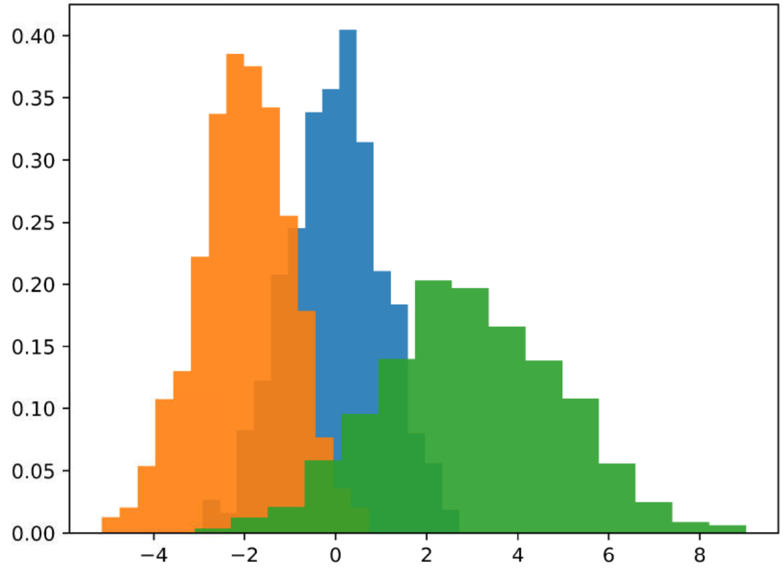
```python
import numpy as np
import matplotlib.pyplot as plt
data = np.random.randn(1000)
plt.hist(data, bins=10, color='blue')
plt.show()
```



Histogram

```
import numpy as np
import matplotlib.pyplot as plt

X1 = np.random.normal(0, 1, 1000)
X2 = np.random.normal(-2, 1, 1000)
X3 = np.random.normal(3, 2, 1000)

kwargs = dict(alpha=0.9, normed=True, bins=15)
plt.hist(X1, **kwargs)
plt.hist(X2, **kwargs)
plt.hist(X3, **kwargs)
plt.show()
```

Histogram

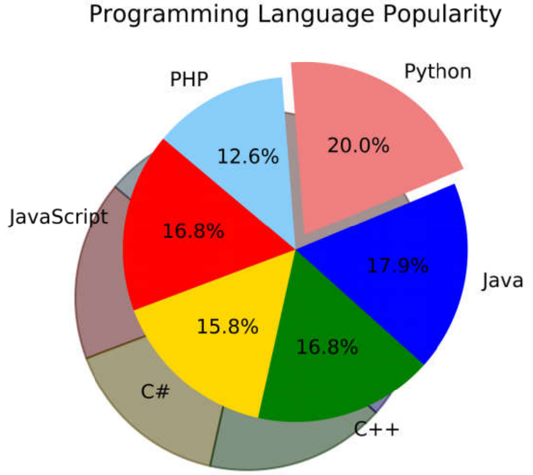


Pie Chart

- □ A Pie Chart is a circular statistical graphic which is divided into slices to illustrate numerical proportion.
- □ In a pie chart, the arc length of each slice (and consequently its central angle and area), is proportional to the quantity it represents.

```
import matplotlib.pyplot as plt
Languages = ('JavaScript', 'C#', 'C++', 'Java', 'Python',
    'PHP')
Popuratity = [8, 7.5, 8, 8.5, 9.5, 6]
colors = ['red', 'gold', 'green', 'blue', 'lightcoral',
    'lightskyblue']
# explode 5th slice
explode = (0, 0, 0, 0, 0.1, 0)
plt.pie(Popuratity, explode=explode, labels=Languages,
    colors=colors, autopct='%1.1f%%', shadow=True,
    startangle=140)
plt.title("Programming Language Popularity")
plt.show()
```

Pie Chart



Contour Plot

- □ A contour plot of function of two variable is a curve along which the function has constant value.
- □ Contour plot is a cross section of three dimension graph of the function $f(x, y)$ and plane parallel to x,y-plane.
- □ numpy.linspace
  - □ linspace() is a numpy function which returns evenly spaced numbers over a specified interval.
  - □ The endpoint of the interval can optionally be excluded.
  - □ Syntax:

```
numpy.linspace(start, stop, num=50, endpoint=True,
    retstep=False, dtype=None)
```

  - □ start(optional) : starting value of interval. By default start = 0.
  - □ stop : end value of interval.

numpy.meshgrid

- □ meshgrid function is used to create a rectangular grid out of an array of x values and an array of y values.
- □ meshgrid is very useful to evaluate functions on a grid.

```
import numpy as np
x = np.linspace(5.0, 8.0, num=5)
y = np.linspace(2.0, 3.0, num = 3)
X,Y = np.meshgrid(x,y)
>>>X
array([[5.  , 5.75, 6.5 , 7.25, 8.  ],
       [5.  , 5.75, 6.5 , 7.25, 8.  ],
       [5.  , 5.75, 6.5 , 7.25, 8.  ]])
>>>Y
array([[2. , 2. , 2. , 2. , 2. ],
       [2.5, 2.5, 2.5, 2.5, 2.5],
       [3. , 3. , 3. , 3. , 3. ]])
```

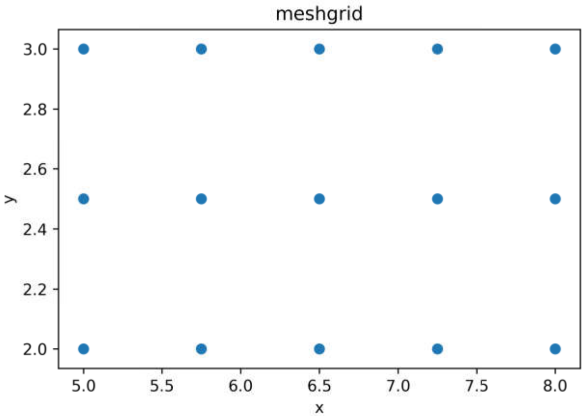
Contour Plot

- □ num (int, optional) : No. of samples to generate. Default value is 50.
- □ endpoint (bool, optional) : If True, stop is the last sample. Otherwise, it is not included. Default value is True
- □ retstep (bool, optional) : If True, return (samples, step). By default retstep = False.
- □ dtype (dtype, optional) : type of output array. If dtype is not given, infer the data type from the other input arguments.

```
>>>import numpy as np
>>>np.linspace(5.0, 8.0, num=5)
array([5.  , 5.75, 6.5 , 7.25, 8.  ])
>>>np.linspace(5.0, 8.0, num=5, endpoint = False)
array([5. , 5.6, 6.2, 6.8, 7.4])
>>>np.linspace(5.0, 8.0, num=5, retstep = True)
(array([5.  , 5.75, 6.5 , 7.25, 8.  ]), 0.75)
```

meshgrid visualization

```
>>>import matplotlib.pyplot as plt
>>>plt.scatter(X,Y)
>>>plt.xlabel('x')
>>>plt.ylabel('y')
>>>plt.title('meshgrid')
>>>plt.show()
```

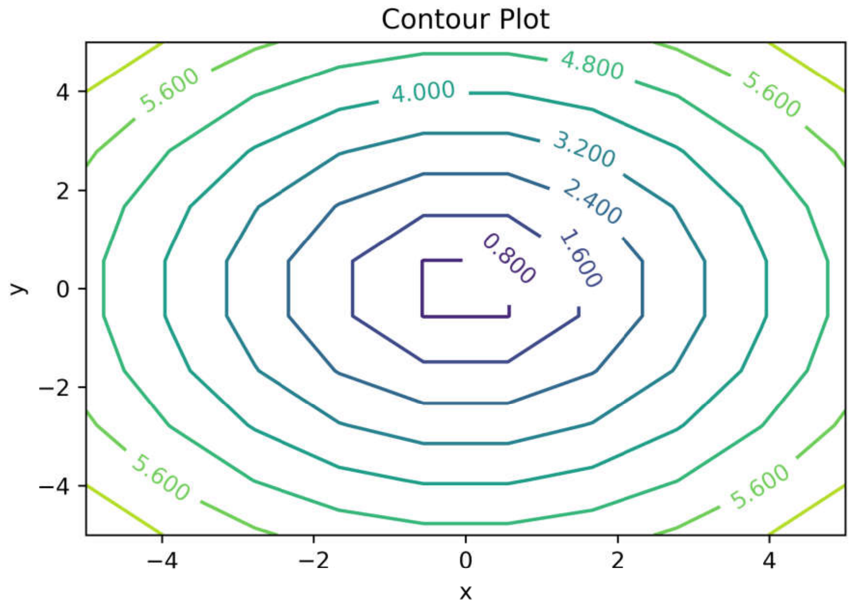


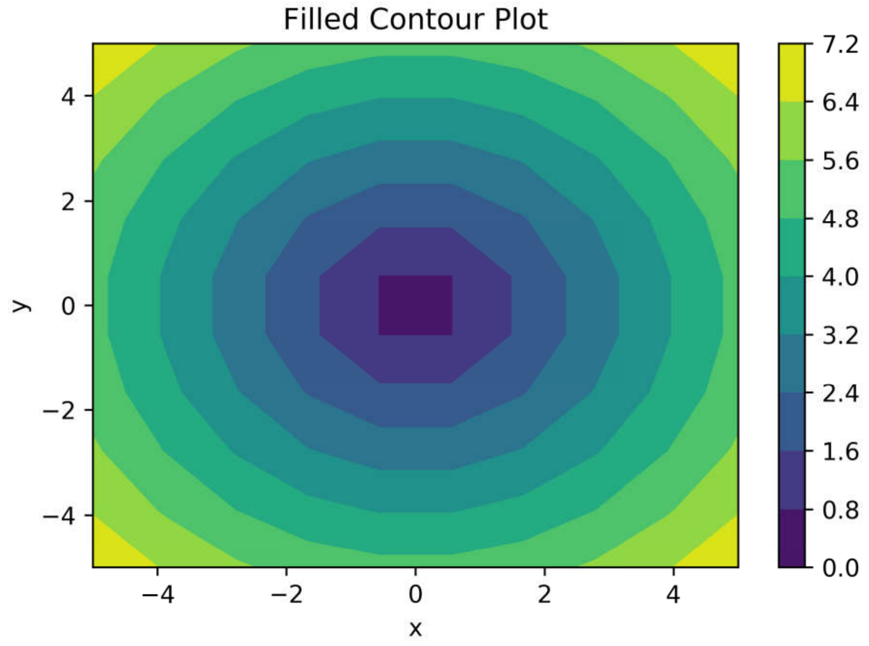
Contour Plot

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-5.0, 5.0, num=10)
y = np.linspace(-5.0, 5.0, num=10)
X,Y = np.meshgrid(x,y)
Z = np.sqrt(X**2 + Y**2)

#countour plots
cp = plt.contour(X, Y, Z)

#plot contour labels
plt.clabel(cp, inline=True,fontsize=10)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Contour Plot')
plt.show()
```

Contour Plot

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-5.0, 5.0, num=10)
y = np.linspace(-5.0, 5.0, num=10)
X,Y = np.meshgrid(x,y)
Z = np.sqrt(X**2 + Y**2)

#filled countour plots
cp = plt.contourf(X, Y, Z)

# Add a colorbar to a plot
plt.colorbar(cp)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Filled Contour Plot')
plt.show()
```





Example of Plotting Data From Text File

□ 'MarksDetail.txt' is space separated file contains marks detail of students, and is as follow:

Student_Name/Subjects	Math	Physics	Chemistry
Daniel	65	70	80
Adam	60	75	65
John	75	80	70
Robert	70	60	78

□ For given data, using bar plot, we can do two types of comparison

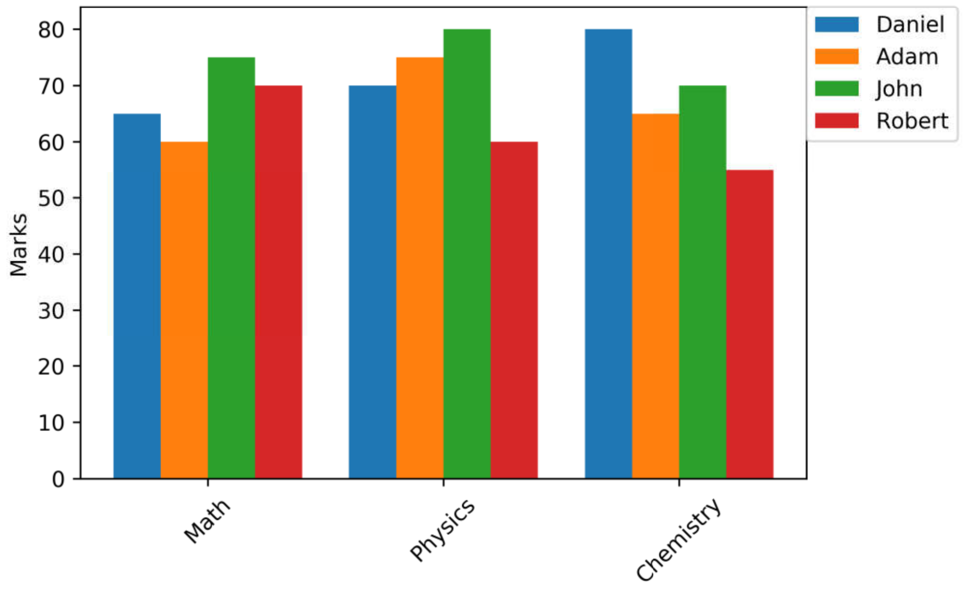
- □ Comparison of students based on subject wise marks.
- □ Comparison of subject marks for different students.

Reading Data From Text File

```
import numpy as np
import matplotlib.pyplot as plt
with open('MarksDetail.txt') as f:
    Subjects = f.readline().split()[1:]
    Name = []
    Marks =[]
    while True:
        row = f.readline().split()
        if row == []:
            break
        Name.append(row[0])
        Marks.append(list(map(float, row[1:])))
Marks = np.array(Marks)
```

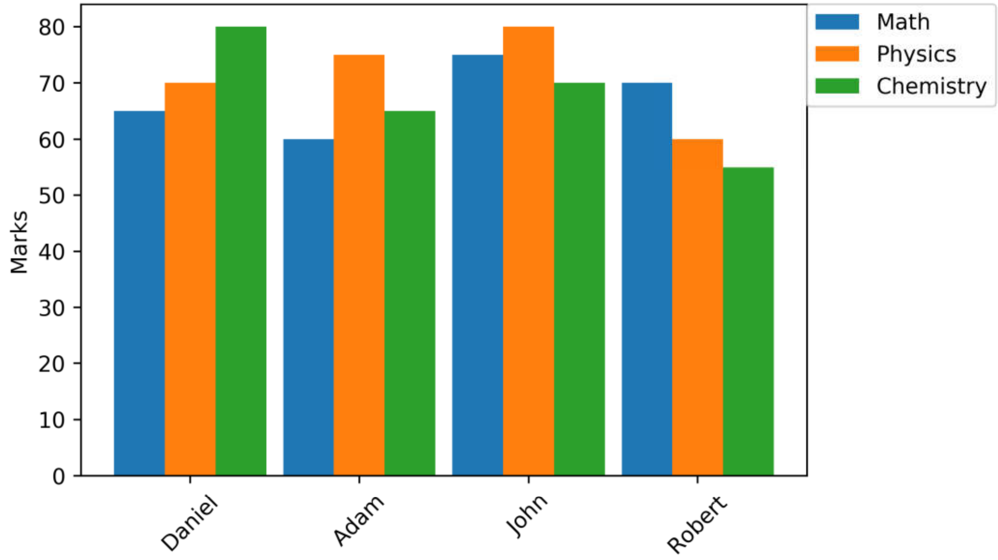
Comparison of students based on subject wise marks.

```
width = 0.20
x = np.arange(len(Subjects))
plt.figure()
plt.bar(x - 3/2*width, Marks[0,:], width, label = Name[0])
plt.bar(x - 1/2*width, Marks[1,:], width, label = Name[1])
plt.bar(x + 1/2*width, Marks[2,:], width, label = Name[2])
plt.bar(x + 3/2*width, Marks[3,:], width, label = Name[3])
plt.xticks(x, Subjects, rotation = 45)
plt.ylabel('Marks')
plt.legend(bbox_to_anchor=(1, 1), loc='upper left',
    borderaxespad=0.)
plt.show()
```



Comparison of subject marks for different students.

```
width = 0.3
x = np.arange(len(Name))
plt.figure()
plt.bar(x - width, Marks[:,0], width, label = Subjects[0])
plt.bar(x,  Marks[:,1], width, label = Subjects[1])
plt.bar(x + width, Marks[:,2], width, label = Subjects[2])

plt.xticks(x, Name, rotation = 45)
plt.ylabel('Marks')
plt.legend(bbox_to_anchor=(1, 1), loc='upper left',
    borderaxespad=0.)
plt.savefig('MarksDetail2.png', dpi = 600,bbox_inches =
    "tight")
plt.show()
```


Marks
Daniel
Adam
John
Robert
Math
Physics
Chemistry