AlmaBetter

MATPLOTLIB FOR PYTHON

Pitaya Smoothie light
Annotation
X-label
Pitaya Smoothie dark
Annotation
X-label
Pacoty
Annotation
X-label

Matplotlib

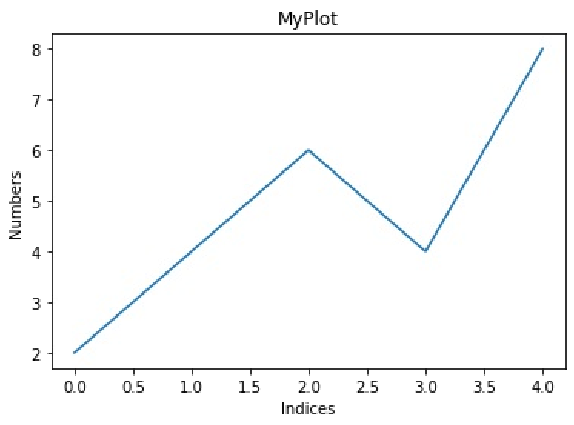
matplotlib.pyplot is a collectionof command style functions that make matplotlib work ike MATLAB.

Each pyplot function makes some change to a figure: e.g, creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

In [ ]:
```
import matplotlib.pyplot as plt #importing matplotlib
```

In [ ]:
```
plt.plot([2,4,6,4,8]) # taking random number
plt.ylabel("Numbers")
plt.xlabel("Indices")
plt.title("MyPlot")
plt.show()
```



If you provide a single list or array to the plot() command. matplotlib assumes it is a sequence of y values, and automatically generates the x valus for you. Since python ranges start with 0, the default x vector has the same lenght as y but starts with 0. Hance the x data are [0,1,2,3].

In [ ]:
```

```

plot x versus y

To plot Y versus X using matplotlib, call plot() function on matplotlib.pyplot object.

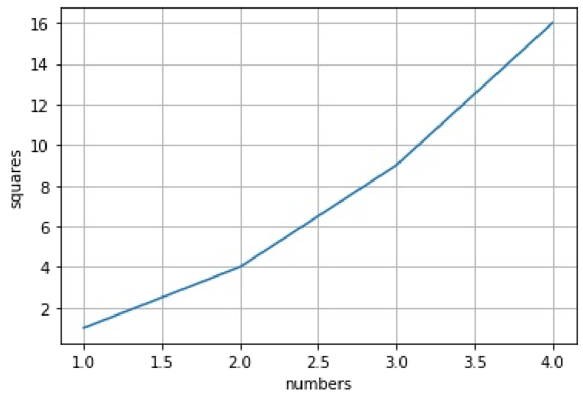
The definition of plot() function is

plot(*args, scalex=True, scaley=True, data=None, **kwargs) The following are some of the sample plot() function calls with different set of parameters.

plot(x, y) plot([x], y, [fmt], *, data=None, **kwargs) plot([x], y, [fmt], [x2], y2, [fmt2], ..., kwargs)**

In [ ]:

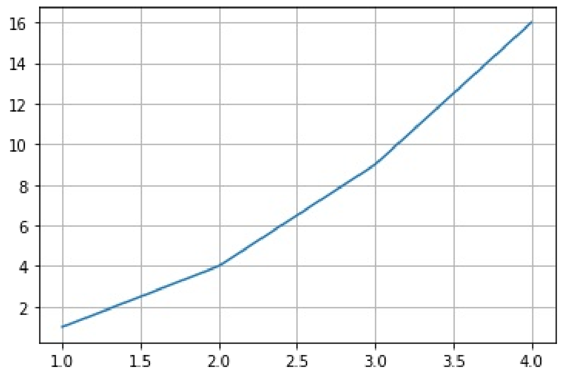
```
plt.plot([1,2,3,4],[1,4,9,16]) #[1,2,3,4]=x & [1,49,16]=y
plt.ylabel("squares")
plt.xlabel("numbers")
plt.grid()# grid on
plt.show()
```



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.

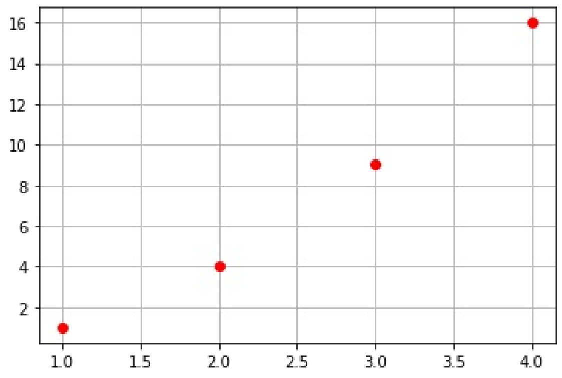
In [ ]:

```
plt.plot([1,2,3,4],[1,4,9,16])
plt.grid()
plt.show()
```



In [ ]:

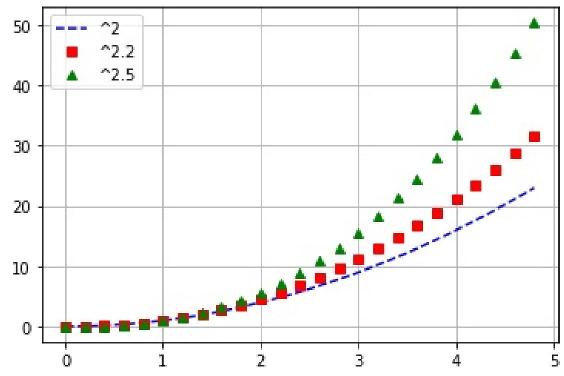
```
#Now can you observe some changes with respect to above one
plt.plot([1,2,3,4],[1,4,9,16],'ro')
plt.grid()
plt.show()
```



If matplotlib were limited to working with lists, it would be fairly useless for numeric processing. generally, you will use numpy arrays. In fact, all sequences are converted to numpy arrays internally.

In [ ]:

```
import numpy as np #for your reference we use numpy for mathematical operations
t = np.arange(0., 5., 0.2)

# Blue dashes, red squares and green traingles
plt.plot(t, t**2, 'b--', label = '^2') # 'rs', 'g^'   # Blue dash line.
plt.plot(t, t**2.2, 'rs', label = '^2.2')   # red squres (rs)
plt.plot(t, t**2.5, 'g^', label = '^2.5')   # green traingle (g^)
plt.grid()
plt.legend()  #  add legend based on line labels
plt.show()
```



In [ ]:

Controlling line properties

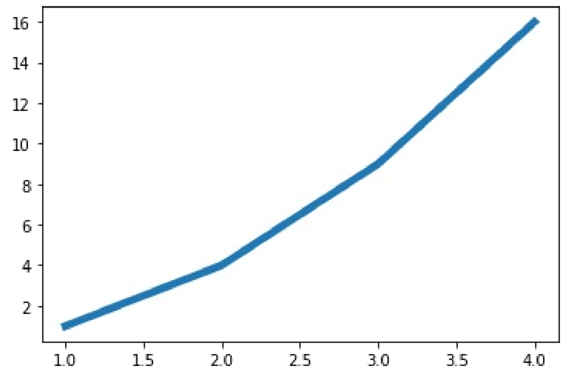
There are many properties of a line that can be set, such as the color, dashes, and several others.

Using keyword arguments

We can use arguments within the plot function to set the property of the line.

In [ ]:

```
x = [1,2,3,4]
y = [1,4,9,16]
plt.plot(x,y,linewidth = 5.0) # change the value of linewidth=5.0 to any other value to observe the changes
# plt.grid()
plt.show()
```

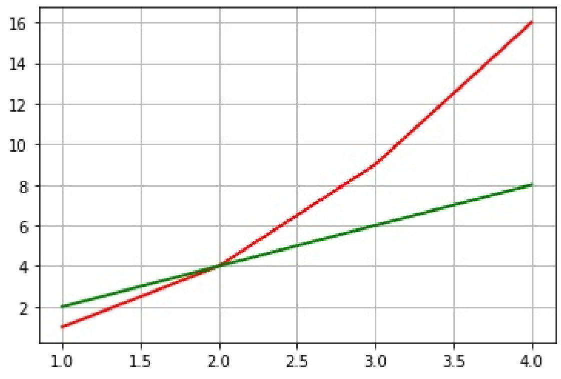


use the step()

In [ ]:

```
x1 = [1,2,3,4]
y1 = [1,4,9,16]
x2 = [1,2,3,4]
y2 = [2,4,6,8]
lines = plt.plot(x1, y1, x2, y2)

# Use keyword args
plt.setp(lines[0], color='r', linewidth=2.0) #lines[0]=x1 & Y1 ,lines[1]=x2 & y2
plt.setp(lines[1], color='g', linewidth=2.0)

# or MATLAB style string value pairs
# plt.setp(lines[1], 'color' 'g', 'linewidth', 2.0)
```

```
plt.grid()
```

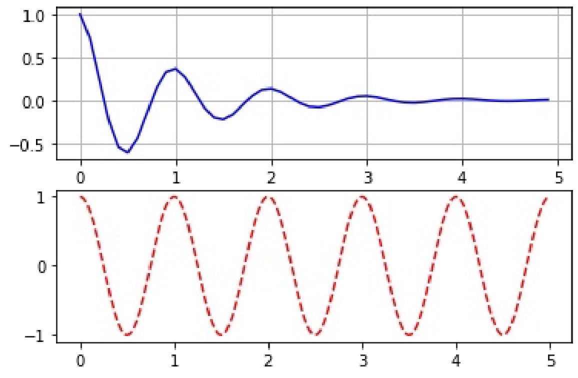


Working with multiple figures & axes

Matplotlib uses the concept of a current figure and current axes. Figures are identified via a figure number that is passed to figure. The figure with the given number is set as current figure. Additionally, if no figure with the number exists, a new one is created.

In [ ]:

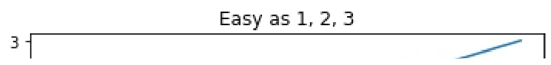
```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure(1)
# The subplot() command specifies numrows, numcols,
# fignum where fignum ranges from 1 to numrows*numcols.

plt.subplot(211)  # two rows and one columns and figure one(211 means)
plt.grid()
plt.plot(t1, f(t1), 'b-')

plt.subplot(212) # two rows and one columns and figure two(212 means)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```

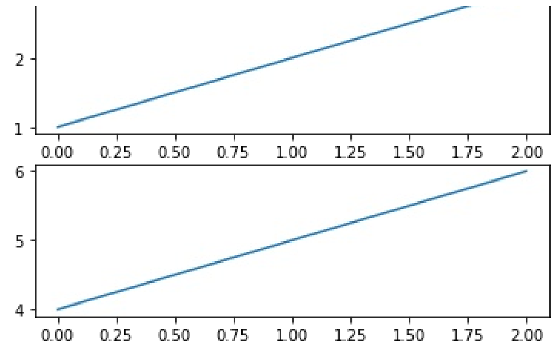
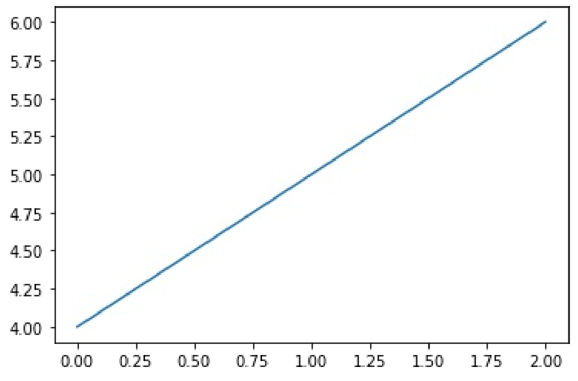


In [ ]:

```
plt.figure(1)                # the first figure
plt.subplot(211)             # the first subplot in the first figure
plt.plot([1, 2, 3])
plt.subplot(212)             # the second subplot in the first figure
plt.plot([4, 5, 6])

plt.figure(2)                 # a second figure
plt.plot([4, 5, 6])           # creates a subplot(111) by default

plt.figure(1)                 # Figure 1 current; cubplot(212) still current
plt.subplot(211)              # make subplot (211) in figure 1 current
plt.title("Easy as 1, 2, 3")  # subplot 211 title
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:11: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance.  In a future version, a new instance will always be created and returned.  Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.
  # This is added back by InteractiveShellApp.init_path()
```



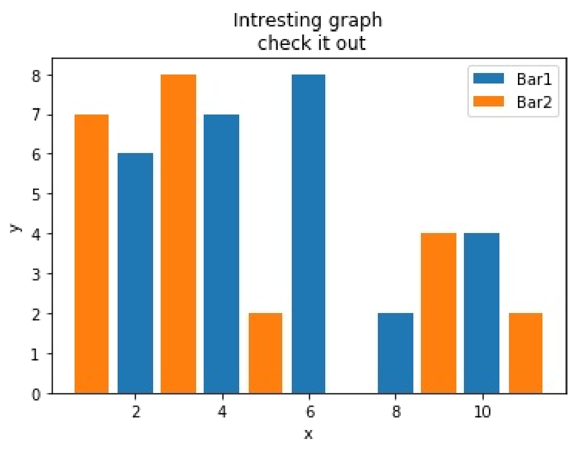
In [ ]:
```

```

Bar Chart and Histrogram

In [ ]:
```
import matplotlib.pyplot as plt
x = [2,4,6,8,10]
y = [6,7,8,2,4]
```

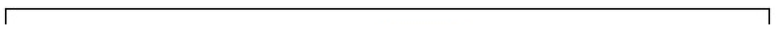
In [ ]:
```
x2 = [1,3,5,9,11]
y2 = [7,8,2,4,2]
```

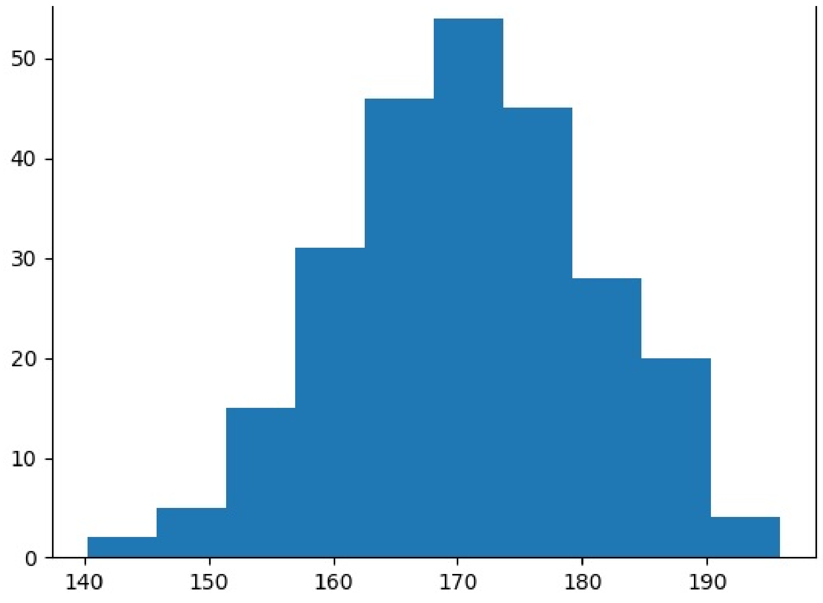
In [ ]:
```
plt.bar(x,y,label='Bar1')
plt.bar(x2,y2,label='Bar2')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Intresting graph\n check it out')
plt.legend()
plt.show()
```



In [ ]:
```

```

histogram plot





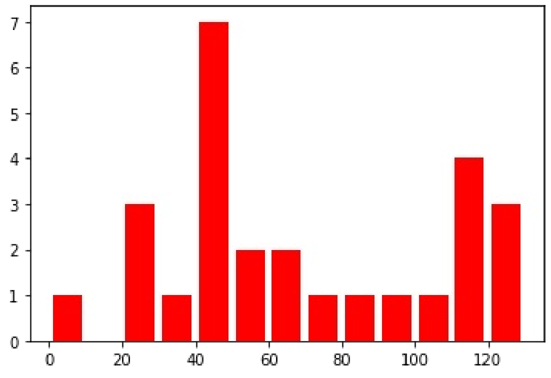
In [ ]:
```
population_ages = [22,55,62,45,21,22,34,42,42,4,99,102,110,120,121,130,111,115,112,80,75,65,54,44,43,42,48]
```

In [ ]:
```
ids = [x for x in range (len(population_ages))]
```

In [ ]:
```
# plt.bar(ids,population_ages)
bins = [0,10,20,30,40,50,60,70,80,90,100,110,120,130]
```

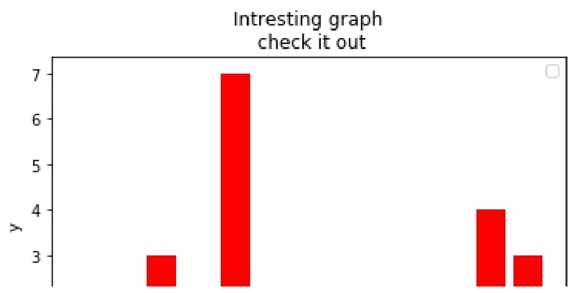
In [ ]:
```
plt.hist(population_ages, bins, histtype='bar',rwidth=0.8,color='red')
```

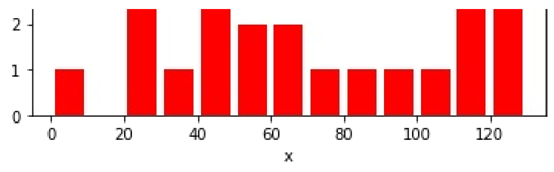
Out[ ]:
```
(array([1., 0., 3., 1., 7., 2., 2., 1., 1., 1., 1., 4., 3.]),
 array([  0,  10,  20,  30,  40,  50,  60,  70,  80,  90, 100, 110, 120,
        130]),
 <a list of 13 Patch objects>)
```



In [ ]:
```
plt.hist(population_ages, bins, histtype='bar',rwidth=0.8,color='red')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Intresting graph\n check it out')
plt.legend()
plt.show()
```

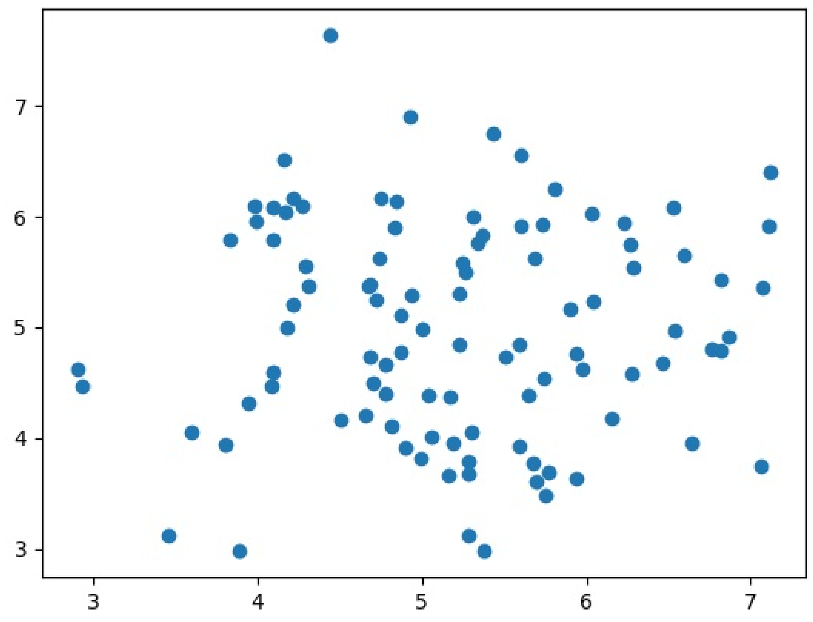
```
No handles with labels found to put in legend.
```





```
In [ ]:
```

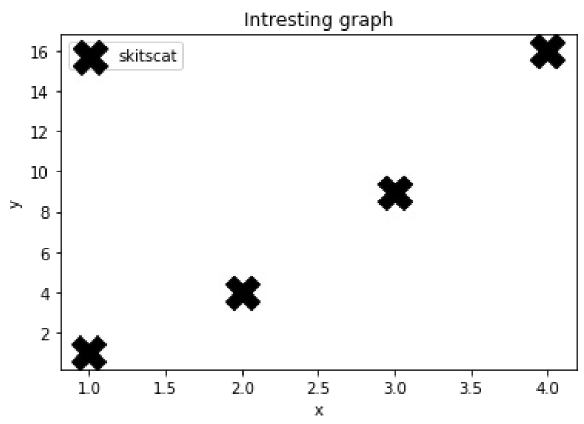
Scatter plot



```
In [ ]: import matplotlib.pyplot as plt
```

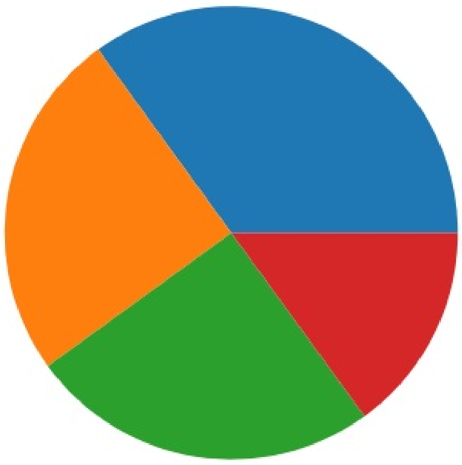
```
In [ ]: x = [1,2,3,4,5,6,7,8]
        y = [5,2,4,2,1,4,5,2]
```

```
In [ ]: plt.scatter(x,y,label='skitscat',color='k',marker='X', s=500) # play with value of marker='X', s=500
        plt.xlabel('x')
        plt.ylabel('y')
        plt.title('Intresting graph')
        plt.legend()
        plt.show()
```



```
In [ ]:
```

Pie Chart

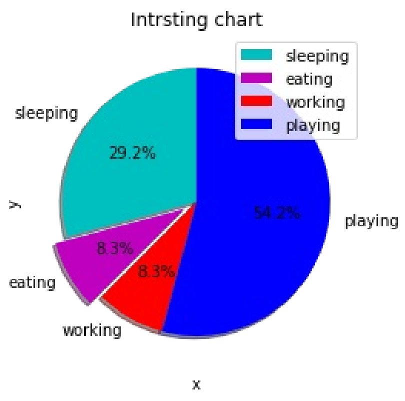


```
In [ ]: import matplotlib.pyplot as plt
```

```
In [ ]: days =      [1,2,3,4,5]
        sleeping =  [6,7,8,11,7]
        eating =    [2,3,4,3,2]
        working =   [7,8,7,2,2,]
        playing =   [8,5,7,8,13]
```

```
In [ ]: slices = [7,2,2,13] #slicing means we are converting our pie chart into 4 parts
        activities = ['sleeping','eating','working','playing']
        cols = ['c','m','r','b']
```

```
In [ ]: plt.pie(slices, labels=activities, colors=cols, startangle=90,shadow=True, explode=(0,0.1,0,0), autopct='%1.1f%%'
        plt.xlabel('x')
        plt.ylabel('y')
        plt.title('Intrsting chart')
        plt.legend()
        plt.show()
```



```
In [ ]:
```

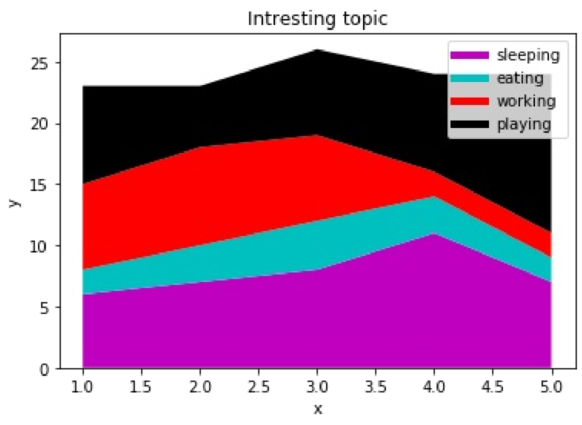
Stack Plot

```
In [ ]: import matplotlib.pyplot as plt
```

```
In [ ]: days =      [1,2,3,4,5]
        sleeping =  [6,7,8,11,7]
        eating =    [2,3,4,3,2]
        working =   [7,8,7,2,2,]
```

```
playing =   [8,5,7,8,13]
```

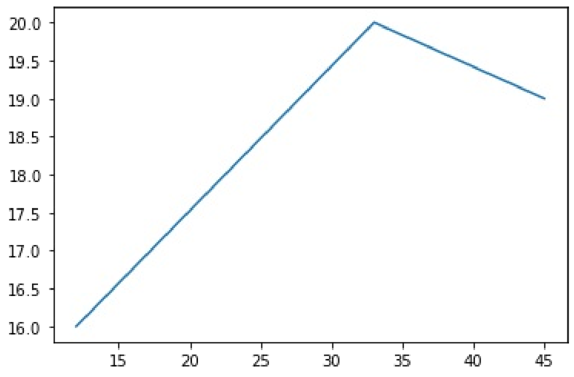
In [ ]:
```
plt.plot([],[],color='m',label='sleeping',linewidth=5)
plt.plot([],[],color='c',label='eating',linewidth=5)
plt.plot([],[],color='r',label='working',linewidth=5)
plt.plot([],[],color='k',label='playing',linewidth=5)
plt.stackplot(days,sleeping,eating,working,playing, colors=['m','c','r','k'])
plt.xlabel('x')
plt.ylabel('y')
plt.title('Intresting topic')
plt.legend()
plt.show()
```



In [ ]:
```
# Let's work More
```

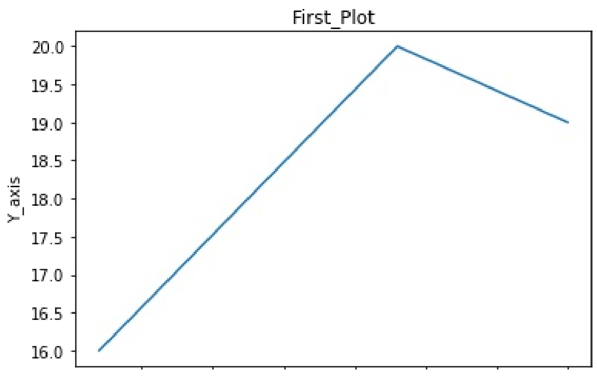
In [ ]:
```
import matplotlib.pyplot as plt
```

In [ ]:
```
plt.plot([12,33,45],[16,20,19])
plt.show()
```



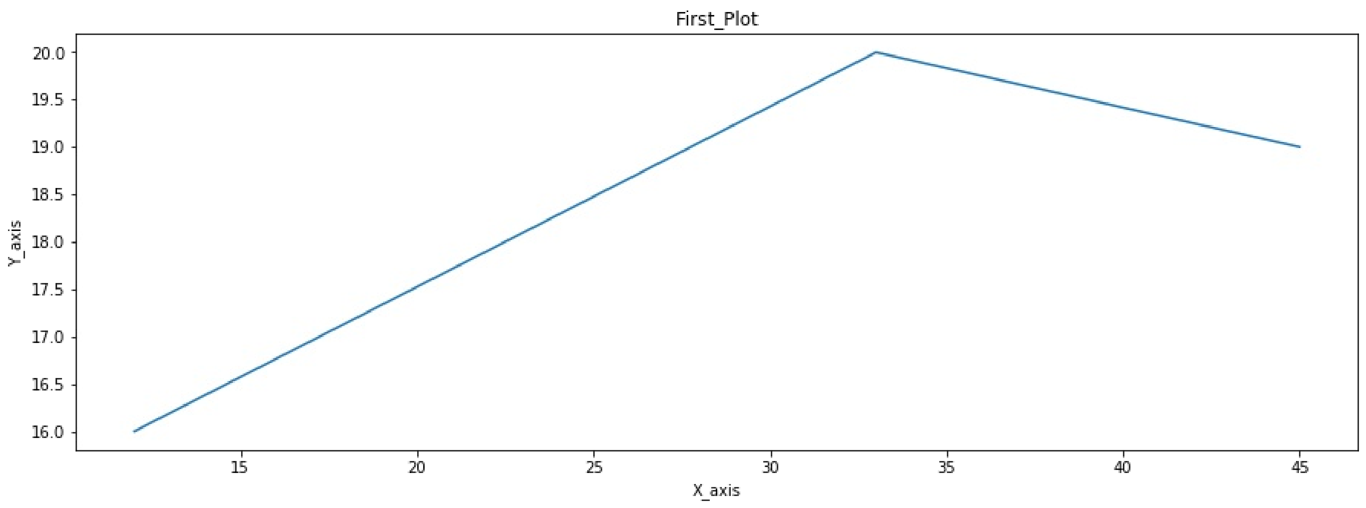
In [ ]:
```
# # Next we will give our plot a title as well as a label for the x-axis and the y-axis.
```

In [ ]:
```
plt.plot([12,33,45],[16,20,19])
plt.title('First_Plot')
plt.xlabel('X_axis')
plt.ylabel('Y_axis')
plt.show()
```



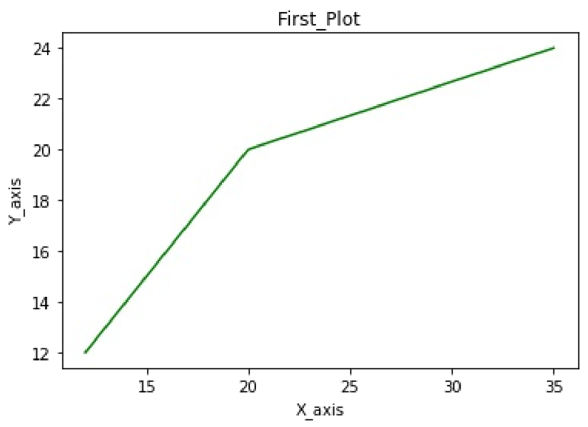
In [ ]:
```python
# We can also change the size of our graph.
```

In [ ]:
```python
plt.figure( figsize = ( 15 , 5 ) )
plt.plot([12,33,45],[16,20,19])
plt.title('First_Plot')
plt.xlabel('X_axis')
plt.ylabel('Y_axis')
plt.show()
```



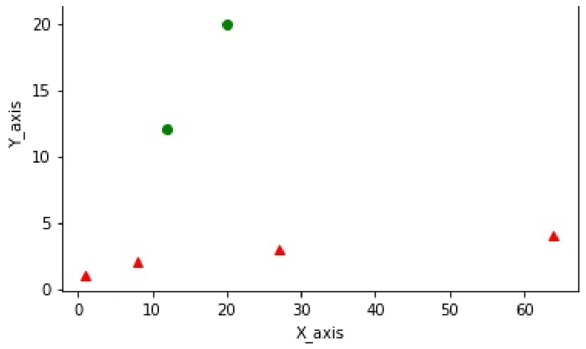
In [ ]:
```python
#  Note that the default color and appearance for our plotted points is a blue line.
# We can pass a third argument that allows us to change the color of our line and/or turn our line into points.
#‘g’ for green ‘r’ for red. ‘go’ for green dots and ‘ro’ for red dots.
```

In [ ]:
```python
plt.plot( [ 12 , 20 , 35 ] , [ 12 , 20 , 24 ] , "g" )# red = r, go= green dots, ro= red dots etc
plt.title('First_Plot')
plt.xlabel('X_axis')
plt.ylabel('Y_axis')
plt.show()
```



In [ ]:
```python
# It is also possible to plot multiple sets of data within the same plot
# by passing multiple arguments within the plot( ) method
```

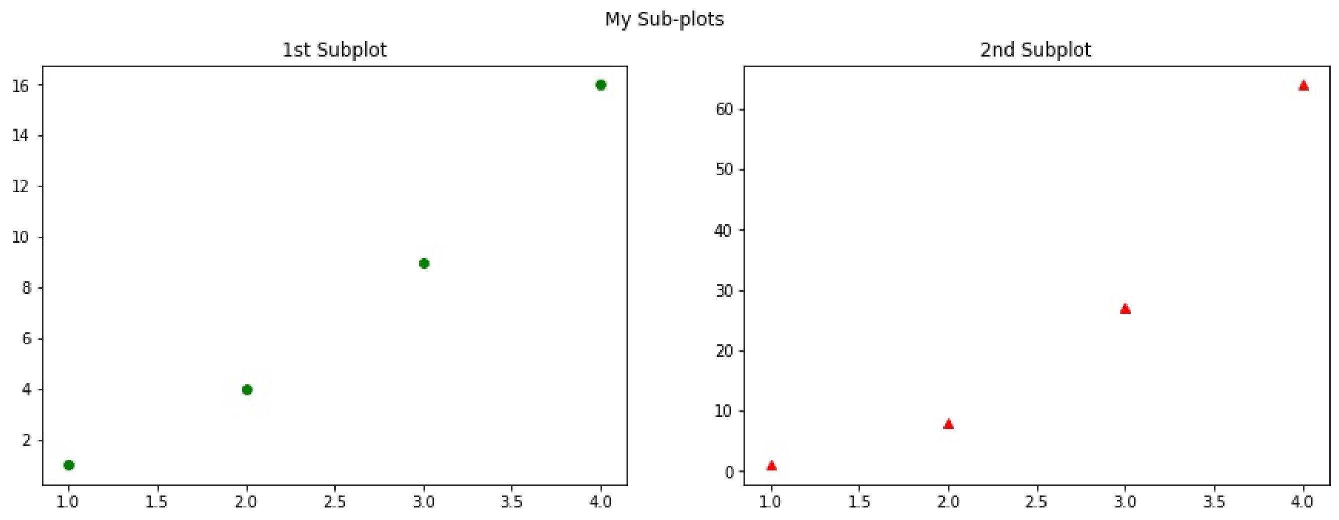
In [ ]:
```python
x = np.arange( 1 , 5 )
y = x**3
plt.plot( [ 12 , 20 , 35 ] , [ 12 , 20 , 24 ] , "go" , y, x , "r^" )
plt.title('First_Plot')
plt.xlabel('X_axis')
plt.ylabel('Y_axis')
plt.show()
```



```
In [ ]:  # Now let’s create multiple plots in one figure.
         #This can be done by using the subplot( ) method. The subplot( ) method takes three arguments nrows(), ncols() ar
```

```
In [ ]:  plt.figure( figsize = ( 15 , 5 ) )
         plt.subplot( 1 , 2 , 1 ) #— — — the 1 , 2 , 1 indicates 1 row 2 column position 1
         plt.plot( [ 1 , 2 , 3 , 4 ] , [ 1 , 4 , 9 , 16 ] ,"go" )
         plt.title( "1st Subplot" )
         plt.subplot( 1 , 2 , 2 ) #— — — the 1 , 2 , 1 indicates 1 row 2 column position 1
         plt.plot( x , y , "r^" )
         plt.title( "2nd Subplot" )
         plt.suptitle( "My Sub-plots" )
```

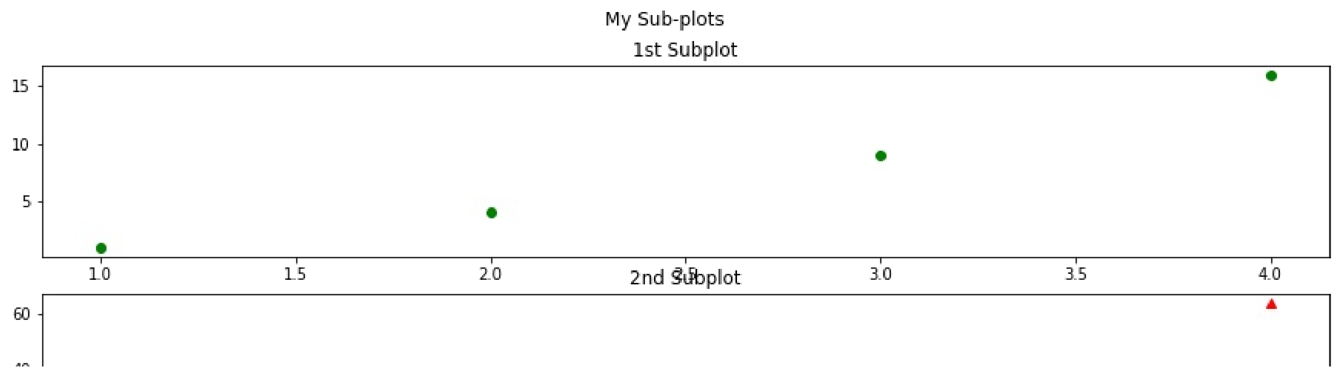
```
Out[ ]:  Text(0.5, 0.98, 'My Sub-plots')
```

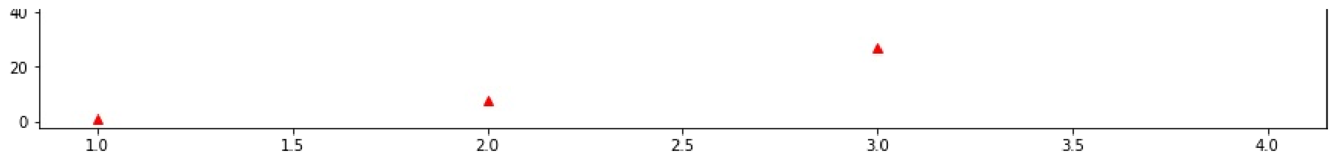


```
In [ ]:  # If we prefer to have our plots in a vertical row all we have to do is change the
         # plt.subplot () argument to (2 , 1 ,1) and (2 , 1 , 2). 2 rows, 1 column, position 1 or position 2.
```

```
In [ ]:  plt.figure( figsize = ( 15 , 5 ) )
         plt.subplot( 2 , 1 , 1 ) #— — — the 1 , 2 , 1 indicates 1 row 2 column position 1
         plt.plot( [ 1 , 2 , 3 , 4 ] , [ 1 , 4 , 9 , 16 ] ,"go" )
         plt.title( "1st Subplot" )
         plt.subplot( 2 , 1 , 2 ) #— — — the 1 , 2 , 1 indicates 1 row 2 column position 1
         plt.plot( x , y , "r^" )
         plt.title( "2nd Subplot" )
         plt.suptitle( "My Sub-plots" )
```

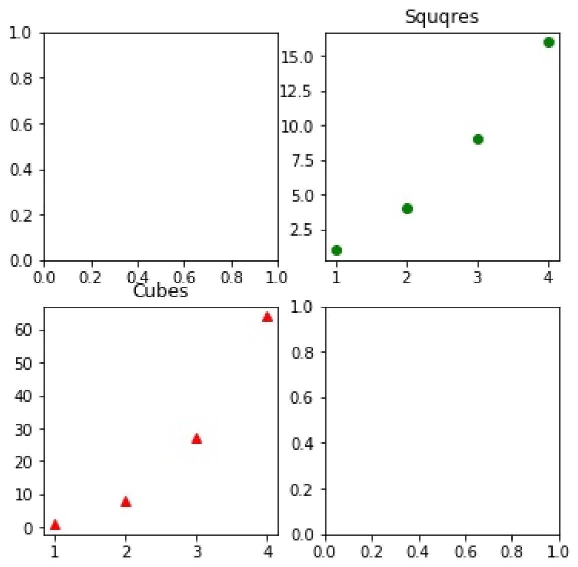
```
Out[ ]:  Text(0.5, 0.98, 'My Sub-plots')
```





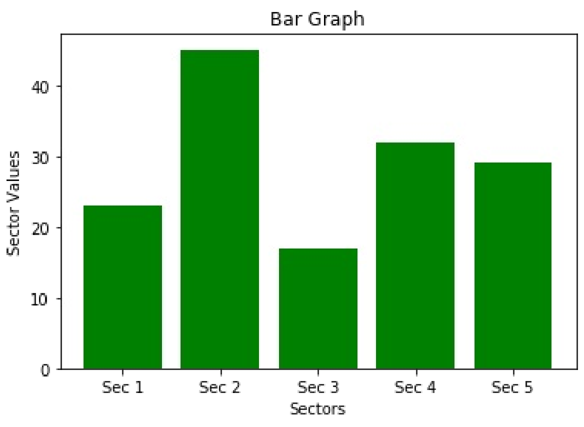
In [ ]:
```
# Creating two subplots is fairly easy but when we try and
# create more subplots the method above can become tedious. In order to overcome this we can use
# the plt.subplots( ) method. This method creates two objects ( figure and axis ) and allows us to assign
# them arbitrary variable names to ( fig & ax ). We then assign the variables to the subplot method that
# indicates the number of rows, the number of columns and the figure size. ( this designates how many plots we wa
```

In [ ]:
```
x = np.arange( 1 , 5 )
y = x**3
fig, ax = plt.subplots( nrows = 2 , ncols = 2 , figsize = ( 6 , 6 ) )
ax[ 0 , 1 ].plot( [ 1 , 2 , 3 , 4 ] , [ 1 , 4 , 9 , 16 ] ,"go" )
ax[ 1 , 0 ].plot( x , y , "r^" )
ax[ 0 , 1 ].set_title( "Squqres" )
ax[ 1 , 0 ].set_title( "Cubes" )
plt.show( )
```



In [ ]:
```
# The most common type of graph is the bar graph because of its ease in
# viewing categorical data. Bar graphs are fairly easy to construct and only require a few arguments
```

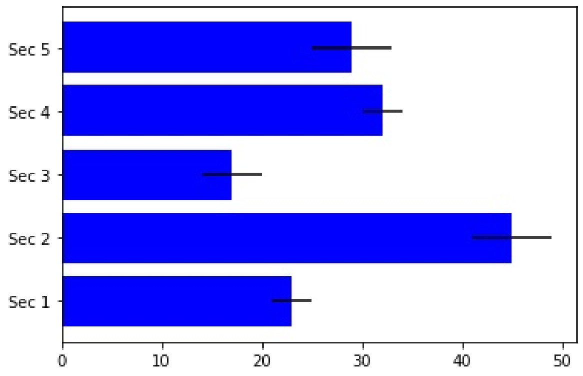
In [ ]:
```
sectors = [ "Sec 1" , "Sec 2" , "Sec 3" , "Sec 4" , "Sec 5" ]
sector_values = [ 23 , 45 , 17 , 32 , 29 ]
plt.bar( sectors , sector_values , color = "green") # — — — plt.barh for horizontal graph
plt.title( "Bar Graph" )
plt.xlabel( "Sectors" )
plt.ylabel( "Sector Values" )
plt.show()
```



In [ ]:
```
# Making the bar graph horizontal is as easy as plt.barh( ).
# Let’s add one more attribute to our graphs in order to depict the amount of variance.
```

In [ ]:
```
varience = [2,4,3,2,4]
plt.barh( sectors , sector_values , xerr = varience , color = 'blue')
```

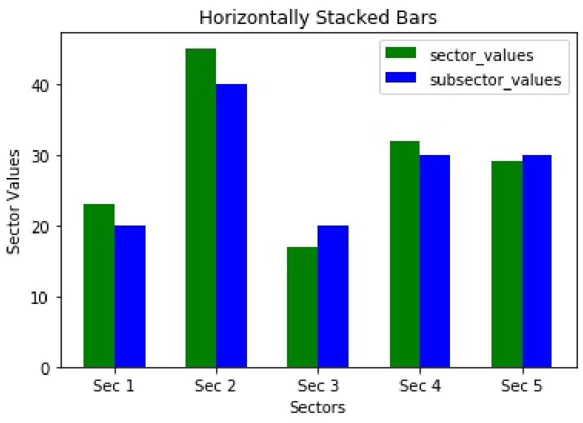
Out[ ]:
```
<BarContainer object of 5 artists>
```



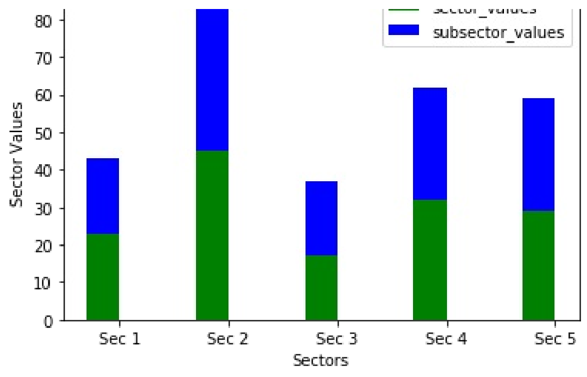
In [ ]:
```
# The xerr= allows us to indicate the amount of variance per sector value. If need be yerr= is also an option
```

In [ ]:
```
# Next we will create a stacked bar graph.
# It may appear that there is a lot of code for this graph but try your best
# to go through it slowly and remember all the steps we took while creating every graph until now
```

In [ ]:
```
sectors = ['Sec 1','Sec 2','Sec 3','Sec 4','Sec 5']
sector_values = [ 23 , 45 , 17 , 32 , 29 ]
subsector_values = [ 20 , 40 , 20 , 30 , 30 ]
index = np.arange(5)
width = 0.30
plt.bar(index, sector_values, width, color = 'green', label = 'sector_values')
plt.bar(index + width, subsector_values,width, color = 'blue', label = 'subsector_values')
plt.title('Horizontally Stacked Bars')
plt.xlabel('Sectors')
plt.ylabel('Sector Values')
plt.xticks(index + width/2 , sectors)
plt.legend(loc = 'best')
plt.show()
```

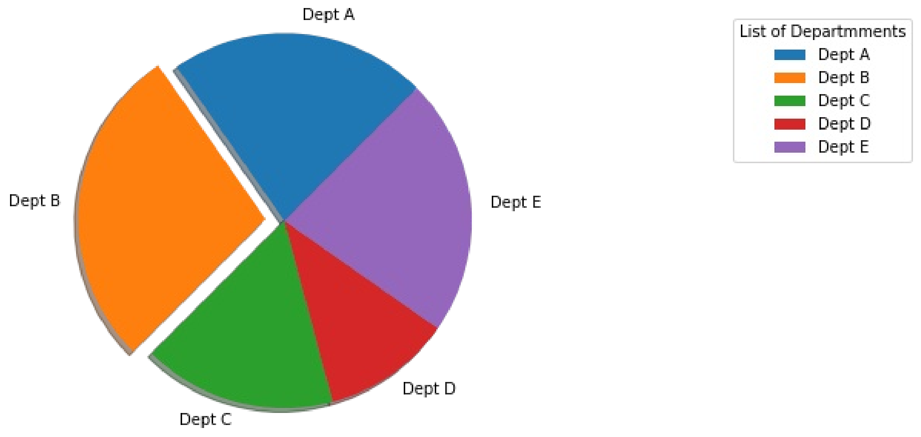


In [ ]:
```
# Without making much modification to our code we can stack our bar graphs one atop the other by indicating,
sectors = ['Sec 1','Sec 2','Sec 3','Sec 4','Sec 5']
sector_values = [ 23 , 45 , 17 , 32 , 29 ]
subsector_values = [ 20 , 40 , 20 , 30 , 30 ]
index = np.arange( 5 )
plt.bar( index , sector_values , width , color = 'green' , label = 'sector_values' )
plt.bar( index , subsector_values , width , color = 'blue' , label = 'subsector_values' , bottom = sector_values
plt.title('Horizontally Stacked Bars')
plt.xlabel('Sectors')
plt.ylabel('Sector Values')
plt.xticks(index + width/2 , sectors)
plt.legend(loc = 'best')
plt.show()
```



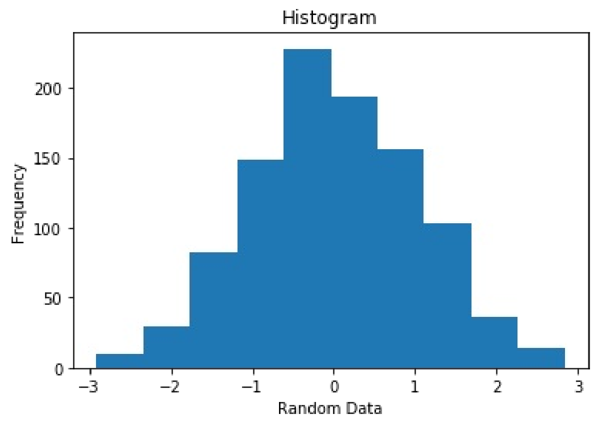
In [ ]:
```
# Next let’s create a pie chart. This is done easily by using the pie( ) method.
```

In [ ]:
```
plt.figure( figsize=( 15 , 5 ) )
hospital_dept = [ 'Dept A' , 'Dept B' , 'Dept C' , 'Dept D' , 'Dept E' ]
dept_share = [ 20 , 25 , 15 , 10 , 20 ]
Explode = [ 0 , 0.1 , 0 , 0 , 0 ]# — — Explodes the Orange Section of Our Plot
plt.pie( dept_share , explode = Explode , labels = hospital_dept , shadow ='True' , startangle= 45 )
plt.axis( 'equal' )
plt.legend( title ="List of Departmments" , loc="upper right" )
plt.show( )
```



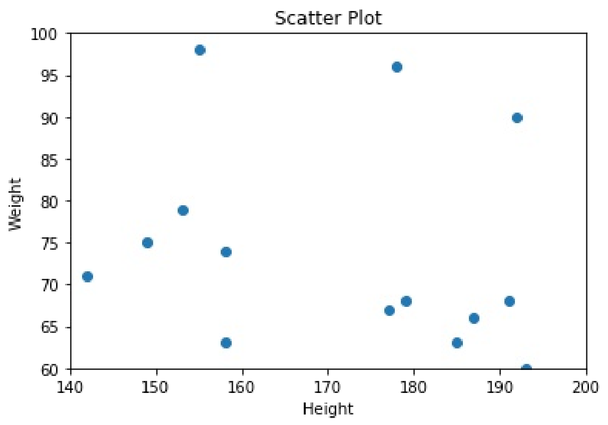
In [ ]:
```
# Histograms are used to plot the frequency of score occurrences in a continuous dataset
#that have been divided into classes called bins.
# In order to create our dataset we are going to use the numpy function np.random.randn.
```

In [ ]:
```
x = np.random.randn( 1000 )
plt.title( "Histogram" )
plt.xlabel( "Random Data" )
plt.ylabel( "Frequency" )
plt.hist( x , 10 ) #— — — plots our randomly generated x values into 10 bins.
plt.show()
```



In [ ]:
```
# Scatter plots are vert useful when dealing with a regression problem.
# In order to create our scatter plot we are going to create an arbitrary set of height
# and weight data and plot them against each other.
```

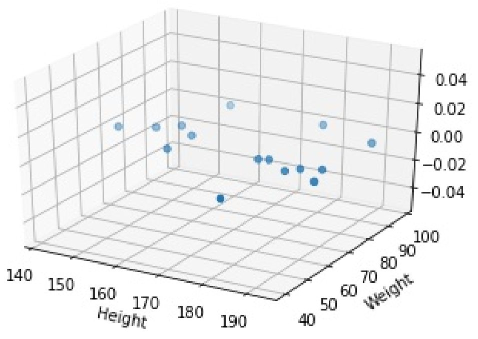
In [ ]:
```
height = np.array ( [ 192 , 142 , 187 , 149 , 153 , 193 , 155 , 178 , 191 , 177 , 182 , 179 , 185 , 158 , 158 ] )
weight = np.array ( [ 90 , 71 , 66 , 75 , 79 , 60 , 98 , 96 , 68 , 67 , 40 , 68 , 63, 74 , 63 ] )
plt.xlim( 140 , 200 )
plt.ylim( 60 , 100 )
plt.scatter( height , weight )
plt.title( 'Scatter Plot' )
plt.xlabel( 'Height' )
plt.ylabel( 'Weight' )
plt.show( )
```



In [ ]:
```
# This same scatterplot can also be visualized in 3D
```

In [ ]:
```
from mpl_toolkits import mplot3d
```

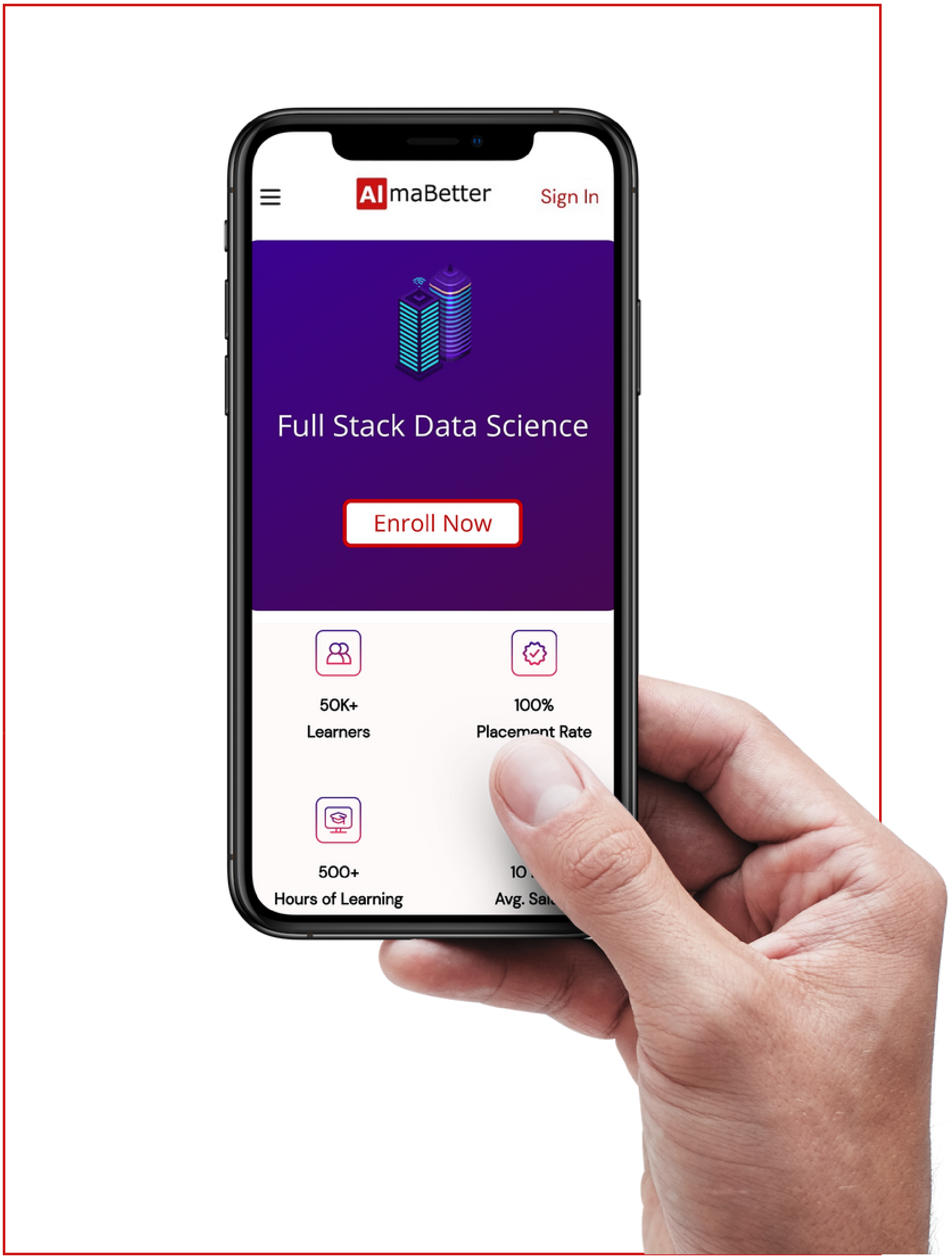
In [ ]:
```
ax = plt.axes( projection = '3d')
ax.scatter3D( height , weight )
ax.set_xlabel( 'Height' )
ax.set_ylabel( 'Weight' )
plt.show( )
```



Happy Learning

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



If you're looking to get into **Data Science**, then **AlmaBetter** is the best place to start your journey.

Join our **Full Stack Data Science Program** and become a job-ready Data Science and Analytics professional in 30 weeks.