Numpy Syntaxes

In [1]:
```
import numpy as np
```

1.Creating Numpy Array 1d and 2d

In [2]:
```
list1 = [2,4,6,8,10]
```

In [3]:
```
list1
```

Out[3]:
```
[2, 4, 6, 8, 10]
```

In [4]:
```
arr1= np.array(list1)
```

In [5]:
```
arr1
```

Out[5]:
```
array([ 2,  4,  6,  8, 10])
```

In [6]:
```
type(list1)
```

Out[6]:
```
list
```

In [7]:
```
type(arr1)
```

Out[7]:
```
numpy.ndarray
```

In [8]:
```
#main use of numpy operation is to use vector operation
arr1 + 2
```

Out[8]:
```
array([ 4,  6,  8, 10, 12])
```

In [9]:
```
list2 = [[0,6,9],[2,3,4]]
```

In [10]:
```
list2
```

Out[10]:
```
[[0, 6, 9], [2, 3, 4]]
```

In [11]:
```
arr2 = np.array(list2)
```

In [12]:
```
arr2
```

Out[12]:
```
array([[0, 6, 9],
       [2, 3, 4]])
```

In [13]:
```
type(arr2)
```

Out[13]:
```
numpy.ndarray
```

In [14]:
```
arr3 = np.array(arr2,dtype="float")
```

In [15]:
```
arr3
```
Out[15]:
```
array([[0., 6., 9.],
       [2., 3., 4.]])
```

In [16]:
```
arr4 = np.array(list2,dtype="float")
```

In [17]:
```
arr4
```
Out[17]:
```
array([[0., 6., 9.],
       [2., 3., 4.]])
```

In [18]:
```
arr5 = np.array(arr3,dtype="bool")
arr5
```
Out[18]:
```
array([[False,  True,  True],
       [ True,  True,  True]])
```

In [19]:
```
#converting float array to int
arr6 = np.array(arr4,dtype="int")
arr6
```
Out[19]:
```
array([[0, 6, 9],
       [2, 3, 4]])
```

In [20]:
```
#converting float array to int
arr6 = arr4.astype('int')
arr6
```
Out[20]:
```
array([[0, 6, 9],
       [2, 3, 4]])
```

In [21]:
```
#creating object array
list7 = [2,5,6.7,True,'g',"sai"]
arr7 = np.array(list7,dtype="object")
arr7
```
Out[21]:
```
array([2, 5, 6.7, True, 'g', 'sai'], dtype=object)
```

In [22]:
```
#converting numpy array to list
list8 = arr7.tolist()
list8
```
Out[22]:
```
[2, 5, 6.7, True, 'g', 'sai']
```

2.Array Dimensions

In [23]:
```
arr8 = arr2.astype('float')
arr8
```
Out[23]:
```
array([[0., 6., 9.],
       [2., 3., 4.]])
```

In [24]:
```
#shape of array
arr8.shape
```
Out[24]:
```
(2, 3)
```

In [25]:
```
#size of array
arr8.size
```
Out[25]:
```
6
```

In [26]:
```
#dtype of array
arr8.dtype
```
Out[26]:
```
dtype('float64')
```

In [27]:
```
#dimensions of an array
arr8.ndim
```
Out[27]:
```
2
```

3.Reversing Rows & Columns

In [28]:
```
arr8
```
Out[28]:
```
array([[0., 6., 9.],
       [2., 3., 4.]])
```

In [29]:
```
#reversing rows
arr8[::-1]
```
Out[29]:
```
array([[2., 3., 4.],
       [0., 6., 9.]])
```

In [30]:
```
#reversing rows and columns
arr8[::-1,::-1]
```
Out[30]:
```
array([[4., 3., 2.],
       [9., 6., 0.]])
```

In [31]:
```
#reversing columns
arr8[:,::-1]
```
Out[31]:
```
array([[9., 6., 0.],
       [4., 3., 2.]])
```

4.Specific Element Extraction

In [32]:
```
arr8
```
Out[32]:
```
array([[0., 6., 9.],
       [2., 3., 4.]])
```

In [33]:
```python
arr8[0, :]
```
Out[33]:
```
array([0., 6., 9.])
```

In [34]:
```python
arr8[:1,:]
```
Out[34]:
```
array([[0., 6., 9.]])
```

In [35]:
```python
arr8[:-1,:]
```
Out[35]:
```
array([[0., 6., 9.]])
```

In [36]:
```python
arr8[:,:-1]
```
Out[36]:
```
array([[0., 6.],
       [2., 3.]])
```

In [37]:
```python
arr8[:,-1]
```
Out[37]:
```
array([9., 4.])
```

In [38]:
```python
arr8[:,1:3]
```
Out[38]:
```
array([[6., 9.],
       [3., 4.]])
```

In [39]:
```python
# : everything
# :-1 everything except last
# :x everyting  from 0 to x-1
# x specific row x or column x
# x:y slicing from x to y-1
```

5.Basic Statistics

In [40]:
```python
arr8
```
Out[40]:
```
array([[0., 6., 9.],
       [2., 3., 4.]])
```

In [41]:
```python
arr8.min()
```
Out[41]:
```
0.0
```

In [42]:
```python
arr8.max()
```
Out[42]:
```
9.0
```

In [43]:
```python
arr8.sum()
```
Out[43]:
```
24.0
```

In [44]:
```python
#mean
arr8.mean()
```

Out[44]:
```
4.0
```

In [45]:
```python
#mean
np.average(arr8)
```

Out[45]:
```
4.0
```

In [46]:
```python
#median
np.median(arr8)
```

Out[46]:
```
3.5
```

In [47]:
```python
#mode
from scipy import stats as s
s.mode(arr8)
```

Out[47]:
```
ModeResult(mode=array([[0., 3., 4.]]), count=array([[1, 1, 1]]))
```

In [48]:
```python
arr8.var()
```

Out[48]:
```
8.333333333333334
```

In [49]:
```python
arr8.std()
```

Out[49]:
```
2.886751345948129
```

6.Reshaping and Flattening

In [50]:
```python
arr8
```

Out[50]:
```
array([[0., 6., 9.],
       [2., 3., 4.]])
```

In [51]:
```python
arr8.shape
```

Out[51]:
```
(2, 3)
```

In [52]:
```python
arr8.reshape(1,6)
```

Out[52]:
```
array([[0., 6., 9., 2., 3., 4.]])
```

In [53]:
```python
arr8.reshape(6,1)
```

Out[53]:
```
array([[0.],
       [6.],
       [9.],
       [2.],
       [3.],
       [4.]])
```

In [54]:
```
arr8.reshape(3,2)
```
Out[54]:
```
array([[0., 6.],
       [9., 2.],
       [3., 4.]])
```

In [55]:
```
arr9 = arr8.flatten()
```

In [56]:
```
arr9.ndim
```
Out[56]:
```
1
```

7.Random Arrays and Sequences

In [57]:
```
np.arange(10)
```
Out[57]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [58]:
```
np.arange(2,10)
```
Out[58]:
```
array([2, 3, 4, 5, 6, 7, 8, 9])
```

In [59]:
```
# np.arange(start, end+1 , step_size)
np.arange(2,10,3)
```
Out[59]:
```
array([2, 5, 8])
```

In [60]:
```
#array in descending order
np.arange(10,0,-1)
```
Out[60]:
```
array([10,  9,  8,  7,  6,  5,  4,  3,  2,  1])
```

In [61]:
```
#it automatically caluculates stepsize by averaging
np.linspace(1,10,5)
```
Out[61]:
```
array([ 1.  ,  3.25,  5.5 ,  7.75, 10.  ])
```

In [62]:
```
#creating arrays with 0 as default value
#np.zeros([i,j,...])
np.zeros([2,2])
```
Out[62]:
```
array([[0., 0.],
       [0., 0.]])
```

8.Unique items and Count

In [63]:
```
arr10 = [[2,4,6,8,10],
         [4,6,8,10,12],
         [6,8,10,12,14],
         [8,10,12,14,16],
         [10,12,14,16,18]]
arr10
```

Out[63]:
```
[[2, 4, 6, 8, 10],
 [4, 6, 8, 10, 12],
 [6, 8, 10, 12, 14],
 [8, 10, 12, 14, 16],
 [10, 12, 14, 16, 18]]
```

In [64]:
```
arr11 = np.array(arr10)
arr11
```

Out[64]:
```
array([[ 2,  4,  6,  8, 10],
       [ 4,  6,  8, 10, 12],
       [ 6,  8, 10, 12, 14],
       [ 8, 10, 12, 14, 16],
       [10, 12, 14, 16, 18]])
```

In [65]:
```
u_val,count= np.unique(arr11,return_counts=True)
```

In [66]:
```
u_val
```

Out[66]:
```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18])
```

In [67]:
```
count
```

Out[67]:
```
array([1, 2, 3, 4, 5, 4, 3, 2, 1], dtype=int64)
```

In [68]:
```
u_val= np.unique(arr10)
```

In [69]:
```
u_val
```

Out[69]:
```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18])
```

Pandas Syntaxes

In [70]:
```
import pandas as pd
import numpy as np
```

1.Create DataFrame

In [71]:
```
data = {
    'roll_no' : [1,2,3,4,5],
    'course_id' : ['a','b','c','b','a'],
    'marks' : [85,89,85,74,96],
}
data
```

Out[71]:
```
{'roll_no': [1, 2, 3, 4, 5],
 'course_id': ['a', 'b', 'c', 'b', 'a'],
 'marks': [85, 89, 85, 74, 96]}
```

In [72]:
```
df = pd.DataFrame(data)
df
```

Out[72]:

	roll_no	course_id	marks
0	1	a	85
1	2	b	89
2	3	c	85
3	4	b	74
4	5	a	96

2.Setting Index

In [73]:
```
df1 = pd.DataFrame(data,index=['p','q','r','s','t'])
df1
```

Out[73]:

	roll_no	course_id	marks
p	1	a	85
q	2	b	89
r	3	c	85
s	4	b	74
t	5	a	96

3.Extracting Info

In [74]:
```
#Row wise extraction
df1.loc['s']
```

Out[74]:
```
roll_no       4
course_id     b
marks        74
Name: s, dtype: object
```

In [75]:
```
#column wise extraction
# df1.iloc[ri:rj , ci:cj]
df1.iloc[:,-1]
```

Out[75]:
```
p    85
q    89
r    85
s    74
t    96
Name: marks, dtype: int64
```

In [76]:
```
df1.iloc[:]
```

Out[76]:

	roll_no	course_id	marks
p	1	a	85
q	2	b	89
r	3	c	85
s	4	b	74
t	5	a	96

In [77]:
```
df1.iloc[0:3]
```

Out[77]:

	roll_no	course_id	marks
p	1	a	85
q	2	b	89
r	3	c	85

In [78]:
```
df1.iloc[:,2]
```

Out[78]:
```
p    85
q    89
r    85
s    74
t    96
Name: marks, dtype: int64
```

In [79]:
```
df1.iloc[:,1:3]
```

Out[79]:

	course_id	marks
p	a	85
q	b	89
r	c	85
s	b	74
t	a	96

In [80]:
```
df1.iloc[:,[0,2]]
```
Out[80]:

	roll_no	marks
p	1	85
q	2	89
r	3	85
s	4	74
t	5	96

In [81]:
```
df1.iloc[[1,3],:]
```
Out[81]:

	roll_no	course_id	marks
q	2	b	89
s	4	b	74

In [82]:
```
df1.iloc[[1,4],[0,2]]
```
Out[82]:

	roll_no	marks
q	2	89
t	5	96

In [83]:
```
df1.iloc[4,2]
```
Out[83]: 96

4.Loading Data from CSV or Excel Sheet

In [84]:
```
df = pd.read_csv('C:\\Users\\G.SAI KRISHNA\\Desktop\\ML_Projects\\ML_GFG\\iris_dataset.csv
```

In [85]:
```
df
```

Out[85]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [86]:
```
df.head()
```

Out[86]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [87]:
```
df.tail()
```

Out[87]:

	sepal_length	sepal_width	petal_length	petal_width
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

In [88]:
```
df.head(15)
```

Out[88]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4
6	4.6	3.4	1.4	0.3
7	5.0	3.4	1.5	0.2
8	4.4	2.9	1.4	0.2
9	4.9	3.1	1.5	0.1
10	5.4	3.7	1.5	0.2
11	4.8	3.4	1.6	0.2
12	4.8	3.0	1.4	0.1
13	4.3	3.0	1.1	0.1
14	5.8	4.0	1.2	0.2

5.Data Info

In [89]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
dtypes: float64(4)
memory usage: 4.8 KB
```

6.Data Description

In [90]:
```
df.describe()
```

Out[90]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

7.Data Selection

In [91]:
```
df['sepal_length'][0:5]
```

Out[91]:
```
0    5.1
1    4.9
2    4.7
3    4.6
4    5.0
Name: sepal_length, dtype: float64
```

In [92]:
```
df.iloc[0:5,0]
```

Out[92]:
```
0    5.1
1    4.9
2    4.7
3    4.6
4    5.0
Name: sepal_length, dtype: float64
```

In [93]:
```
df[['sepal_length','sepal_width']][:5]
```

Out[93]:

	sepal_length	sepal_width
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2
3	4.6	3.1
4	5.0	3.6

In [94]:
```
df.iloc[:5,0:2]
```
Out[94]:

	sepal_length	sepal_width
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2
3	4.6	3.1
4	5.0	3.6

8.Missing Values - Null Values

In [95]:
```
data = {
    'roll_no' : [1,2,3,4,5],
    'course_id' : ['a','b','c','b','a'],
    'marks' : [np.nan,89,85,74,96],
}
data
```
Out[95]:
```
{'roll_no': [1, 2, 3, 4, 5],
 'course_id': ['a', 'b', 'c', 'b', 'a'],
 'marks': [nan, 89, 85, 74, 96]}
```

In [96]:
```
df3 = pd.DataFrame(data)
df3
```
Out[96]:

	roll_no	course_id	marks
0	1	a	NaN
1	2	b	89.0
2	3	c	85.0
3	4	b	74.0
4	5	a	96.0

In [97]:
```
#Check whether a cell is null or not
df3.isnull()
```
Out[97]:

	roll_no	course_id	marks
0	False	False	True
1	False	False	False
2	False	False	False
3	False	False	False
4	False	False	False

In [98]:
```
#Count of null values in a column
df3.isnull().sum()
```

Out[98]:
```
roll_no      0
course_id    0
marks        1
dtype: int64
```

In [99]:
```
#Filling null values with a default value
df4 = df3.fillna(0)
df4
```

Out[99]:

	roll_no	course_id	marks
0	1	a	0.0
1	2	b	89.0
2	3	c	85.0
3	4	b	74.0
4	5	a	96.0

In [100]:
```
#Dropping null values(Entire row will be deleted)
df5=df3.dropna()
df5
```

Out[100]:

	roll_no	course_id	marks
1	2	b	89.0
2	3	c	85.0
3	4	b	74.0
4	5	a	96.0

In [101]:
```
df5 = df3.dropna(axis=0)
df5
```

Out[101]:

	roll_no	course_id	marks
1	2	b	89.0
2	3	c	85.0
3	4	b	74.0
4	5	a	96.0

In [102]:
```
df5 = df3.dropna(axis=1)
df5
```

Out[102]:

	roll_no	course_id
0	1	a
1	2	b
2	3	c
3	4	b
4	5	a

In [103]:
```
#Creating data with not null values
df6 = pd.notnull(df3["marks"])
df6
```

Out[103]:
```
0    False
1     True
2     True
3     True
4     True
Name: marks, dtype: bool
```

9.Statistics

In [104]:
```
data = {
    'roll_no' : [1,2,3,4,5],
    'course_id' : ['a','b','c','b','a'],
    'marks' : [76,89,85,74,96],
}
data
```

Out[104]:
```
{'roll_no': [1, 2, 3, 4, 5],
 'course_id': ['a', 'b', 'c', 'b', 'a'],
 'marks': [76, 89, 85, 74, 96]}
```

In [105]:
```
df7 = pd.DataFrame(data)
df7
```

Out[105]:

	roll_no	course_id	marks
0	1	a	76
1	2	b	89
2	3	c	85
3	4	b	74
4	5	a	96

In [106]:
```
#Sum
df7["marks"].sum()
```
Out[106]:
```
420
```
In [107]:
```
#Average
df7["marks"].mean()
```
Out[107]:
```
84.0
```
In [108]:
```
#Cumulative Sum
df7["marks"].cumsum()
```
Out[108]:
```
0     76
1    165
2    250
3    324
4    420
Name: marks, dtype: int64
```
In [109]:
```
#Count
df7["marks"].count()
```
Out[109]:
```
5
```
In [110]:
```
#Minimum
df7["marks"].min()
```
Out[110]:
```
74
```
In [111]:
```
#Maximum
df7["marks"].max()
```
Out[111]:
```
96
```
In [112]:
```
#Variance
df7["marks"].var()
```
Out[112]:
```
83.5
```
In [113]:
```
#Standard Deviation
df7["marks"].std()
```
Out[113]:
```
9.137833441248533
```
In [114]:
```
#Correlation
df7.corr()
```
Out[114]:

	roll_no	marks
roll_no	1.00000	0.43258
marks	0.43258	1.00000

Matplotlib Graphs

In [115]:

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

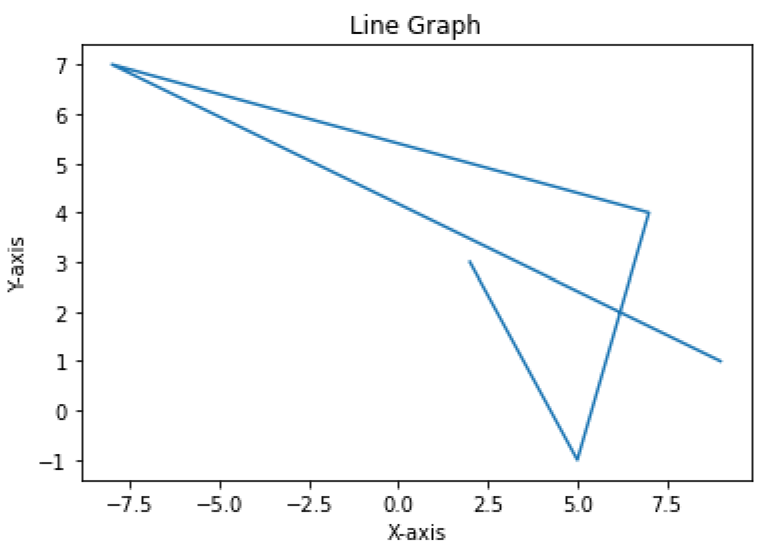
1.Line Plot

In [116]:

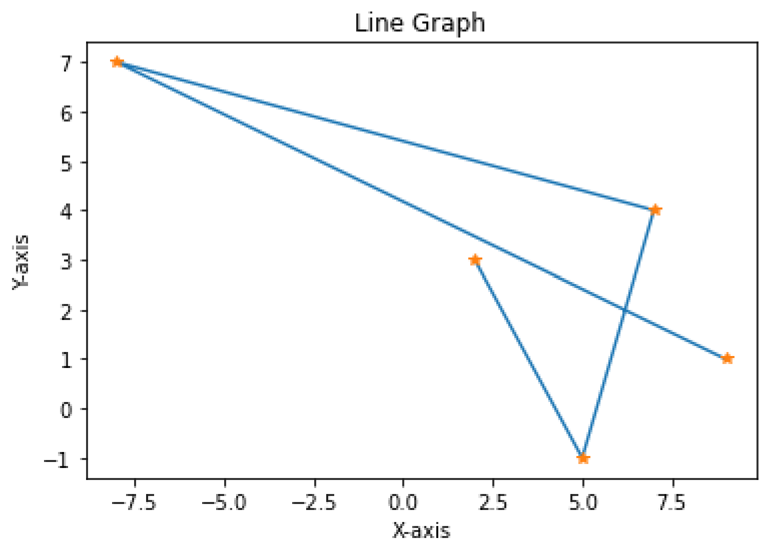
```
x = [2,5,7,-8,9]
y = [3,-1,4,7,1]
plt.plot(x,y)
plt.title("Line Graph")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show
```

Out[116]:

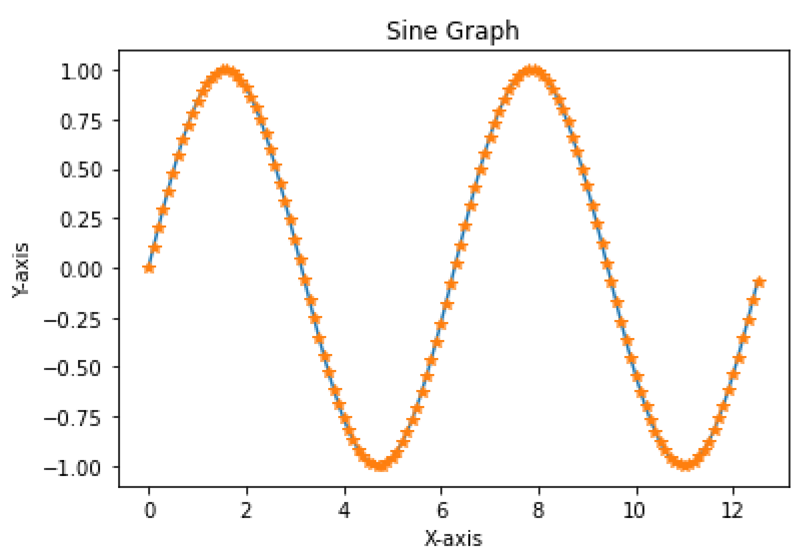
```
<function matplotlib.pyplot.show(*args, **kw)>
```



In [117]:
```
x = [2,5,7,-8,9]
y = [3,-1,4,7,1]
plt.plot(x,y)
plt.plot(x,y,"*")
plt.title("Line Graph")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show
```
Out[117]: <function matplotlib.pyplot.show(*args, **kw)>

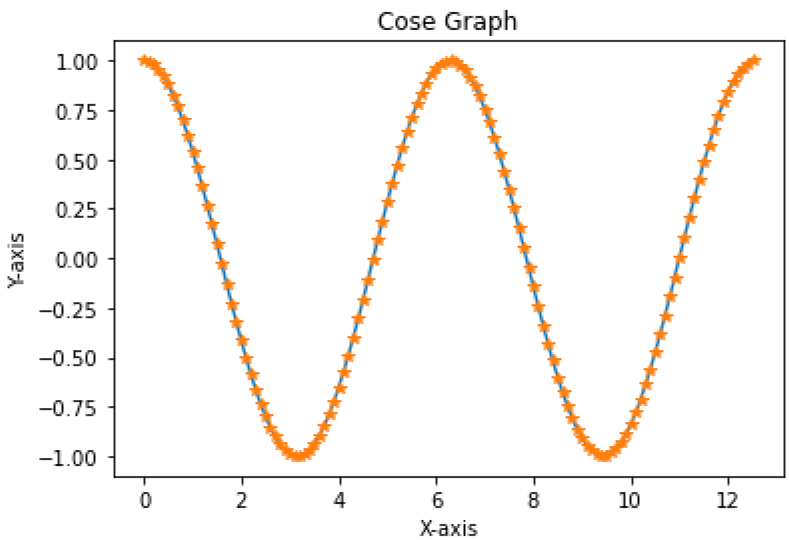


In [118]:
```
x = np.arange(0,4*np.pi,0.1)
y = np.sin(x)
plt.plot(x,y)
plt.plot(x,y,"*")
plt.title("Sine Graph")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show
```
Out[118]: <function matplotlib.pyplot.show(*args, **kw)>



In [119]:
```
x = np.arange(0,4*np.pi,0.1)
y = np.cos(x)
plt.plot(x,y)
plt.plot(x,y,"*")
plt.title("Cose Graph")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show
```

Out[119]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



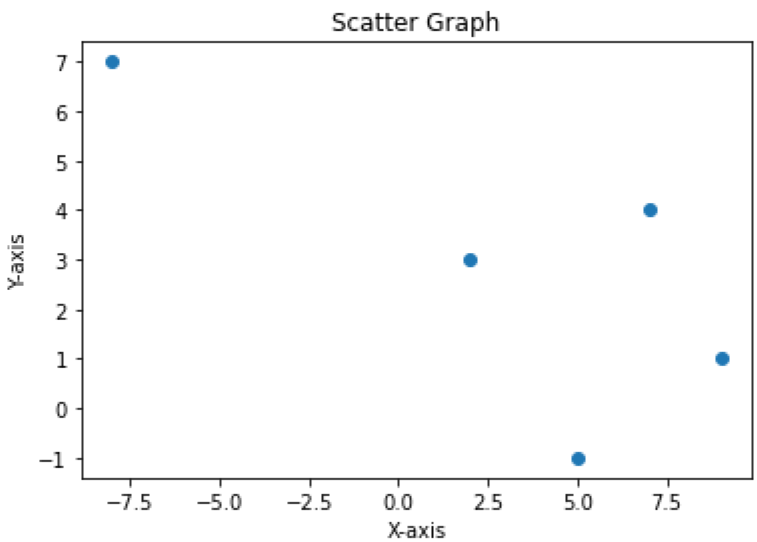
2.Scatter Graph

In [120]:

```
x = [2,5,7,-8,9]
y = [3,-1,4,7,1]
plt.scatter(x,y)
plt.title("Scatter Graph")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show
```

Out[120]:

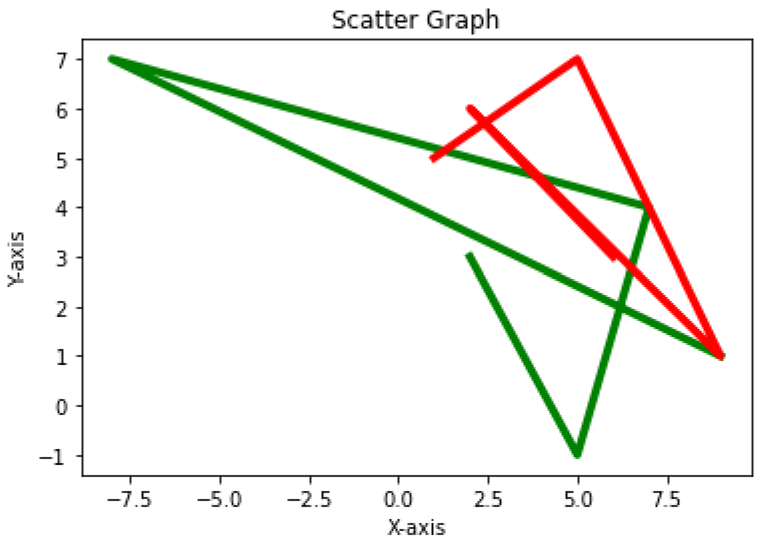
```
<function matplotlib.pyplot.show(*args, **kw)>
```



3.Compare Two Graphs

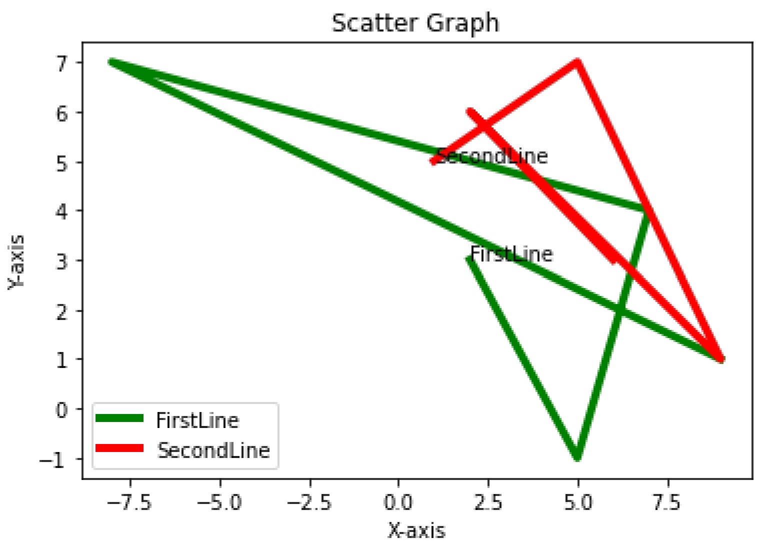
In [121]:
```
x1 = [2,5,7,-8,9]
y1 = [3,-1,4,7,1]

x2 = [1,5,9,2,6]
y2 = [5,7,1,6,3]

plt.plot(x1,y1,color='g',linewidth=4)
plt.plot(x2,y2,color='r',linewidth=4)
plt.title("Scatter Graph")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show
```

Out[121]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



In [122]:
```
#Annotating the graph
x1 = [2,5,7,-8,9]
y1 = [3,-1,4,7,1]

x2 = [1,5,9,2,6]
y2 = [5,7,1,6,3]

plt.plot(x1,y1,color='g',linewidth=4)
plt.plot(x2,y2,color='r',linewidth=4)
plt.title("Scatter Graph")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")

plt.annotate(xy=[2,3],s='FirstLine')
plt.annotate(xy=[1,5],s='SecondLine')

plt.legend(['FirstLine','SecondLine'],loc=3)

plt.show
```

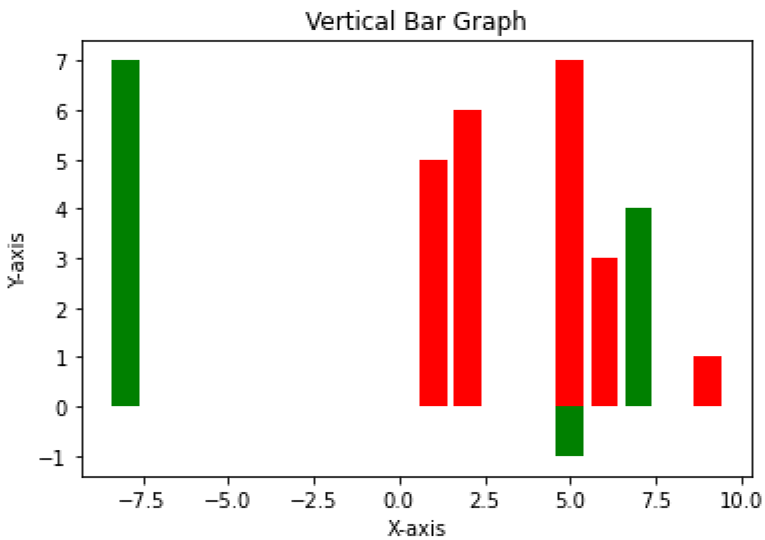
Out[122]: <function matplotlib.pyplot.show(*args, **kw)>



4.Vertical Bar Graph

In [123]:
```
x1 = [2,5,7,-8,9]
y1 = [3,-1,4,7,1]

x2 = [1,5,9,2,6]
y2 = [5,7,1,6,3]
plt.bar(x1,y1,color='g',linewidth=4)
plt.bar(x2,y2,color='r',linewidth=4)
plt.title("Vertical Bar Graph")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show
```

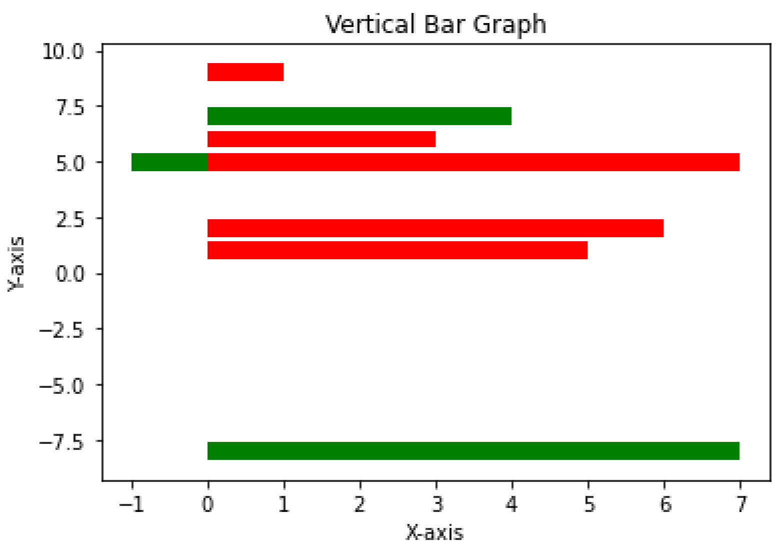
Out[123]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



5.Horizontal Bar Graph

In [124]:
```
x1 = [2,5,7,-8,9]
y1 = [3,-1,4,7,1]

x2 = [1,5,9,2,6]
y2 = [5,7,1,6,3]
plt.barh(x1,y1,color='g',linewidth=4)
plt.barh(x2,y2,color='r',linewidth=4)
plt.title("Vertical Bar Graph")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show
```

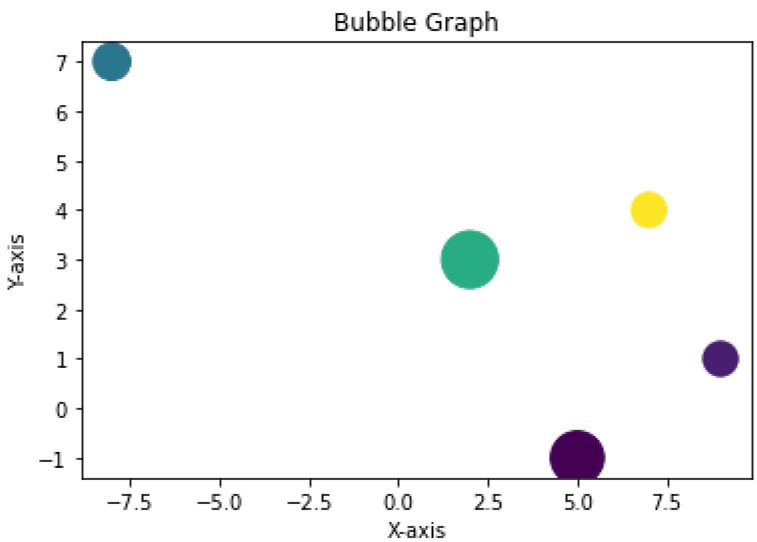
Out[124]: `<function matplotlib.pyplot.show(*args, **kw)>`



6.Bubble Graph

In [125]:
```
x = [2,5,7,-8,9]
y = [3,-1,4,7,1]
plt.scatter(x,y,s=np.random.rand(5)*1000,c=np.random.rand(5))
plt.title("Bubble Graph")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show
```

Out[125]: <function matplotlib.pyplot.show(*args, **kw)>



7.Pie Chart

In [126]:
```
x = [2,5,7,8,9]
lb = ['a','b','c','d','e']
ex = [0,0.1,0.2,0,0]
plt.pie(x,labels=lb,explode=ex,shadow=True)
plt.show
```

Out[126]: <function matplotlib.pyplot.show(*args, **kw)>

Handling Categorical Data

In [127]:
```
import pandas as pd
import numpy as np
```

In [128]:
```
data=pd.read_csv("C:\\Users\\G.SAI KRISHNA\\Desktop\\ML_Projects\\ML_GFG\\employee_data.csv
data
```

Out[128]:

	Emploee_ID	Gender	Remarks
0	45	Male	Nice
1	78	Female	Good
2	56	Female	Great
3	12	Male	Great
4	7	Female	Nice
5	68	Female	Great
6	23	Male	Good
7	45	Female	Nice
8	89	Male	Great
9	75	Female	Nice
10	47	Female	Good
11	62	Male	Nice

In [129]:
```
data.count()
```

Out[129]:
```
Emploee_ID    12
Gender        12
Remarks       12
dtype: int64
```

1.Checking labels in the column

In [130]:
```
data["Remarks"].unique()
```

Out[130]:
```
array(['Nice', 'Good', 'Great'], dtype=object)
```

In [131]:
```
data["Gender"].unique()
```

Out[131]:
```
array(['Male', 'Female'], dtype=object)
```

2.Checking count of each label

In [132]:
```
data["Remarks"].value_counts()
```

Out[132]:
```
Nice     5
Great    4
Good     3
Name: Remarks, dtype: int64
```

Approach 1 - Label Encoding

Label Encoder encodes labels with values between 0 and number_of_classes-1

In [133]:
```
from sklearn.preprocessing import LabelEncoder
label_encoder_X = LabelEncoder()
```

In [134]:
```
label_encoded_data = label_encoder_X.fit_transform(data['Remarks'])
```

In [135]:
```
label_encoded_data = pd.DataFrame(data = label_encoded_data,columns=['Remarks'])
label_encoded_data
```

Out[135]:

	Remarks
0	2
1	0
2	1
3	1
4	2
5	1
6	0
7	2
8	1
9	2
10	0
11	2

In [136]:
```
label_encoded_data['Remarks'].unique()
```

Out[136]:
```
array([2, 0, 1])
```

Approach 2 - One Hot Encoding

Binary Labelling with 1 at position where the label is present else 0

In [137]:
```
#Binary Transformation
one_hot_encoded_data = pd.get_dummies(data,columns=['Remarks','Gender'])
one_hot_encoded_data
```

Out[137]:

	Emploee_ID	Remarks_Good	Remarks_Great	Remarks_Nice	Gender_Female	Gender_Male
0	45	0	0	1	0	1
1	78	1	0	0	1	0
2	56	0	1	0	1	0
3	12	0	1	0	0	1
4	7	0	0	1	1	0
5	68	0	1	0	1	0
6	23	1	0	0	0	1
7	45	0	0	1	1	0
8	89	0	1	0	0	1
9	75	0	0	1	1	0
10	47	1	0	0	1	0
11	62	0	0	1	0	1

In [138]:
```
data
```

Out[138]:

	Emploee_ID	Gender	Remarks
0	45	Male	Nice
1	78	Female	Good
2	56	Female	Great
3	12	Male	Great
4	7	Female	Nice
5	68	Female	Great
6	23	Male	Good
7	45	Female	Nice
8	89	Male	Great
9	75	Female	Nice
10	47	Female	Good
11	62	Male	Nice

Data Scaling

In [139]:
```
data=pd.read_csv("C:\\Users\\G.SAI KRISHNA\\Desktop\\ML_Projects\\ML_GFG\\housing.csv")
```

In [140]:
```
data.head(10)
```

Out[140]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_incor
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.32
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.30
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.25
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.64
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.84
5	-122.25	37.85	52.0	919.0	213.0	413.0	193.0	4.03
6	-122.25	37.84	52.0	2535.0	489.0	1094.0	514.0	3.65
7	-122.25	37.84	52.0	3104.0	687.0	1157.0	647.0	3.12
8	-122.26	37.84	42.0	2555.0	665.0	1206.0	595.0	2.08
9	-122.25	37.84	52.0	3549.0	707.0	1551.0	714.0	3.69

In [141]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [142]:
```
#scaler = MinMaxScaler()
#scaler.fit(data)
```

As we can observe the scaler is giving error so we need to preprocess the data first

Data Preprocessing : Drop Categorical Data,Drop rows with null values

In [143]:
```
data = data.drop(data.columns[[9]],axis=1)
data = data.dropna(axis=0)
```

In [144]:
```
scaler = MinMaxScaler()
scaler.fit(data)
```

Out[144]:
```
MinMaxScaler()
```

In [145]:
```
scaled_data = pd.DataFrame(data = scaler.transform(data),columns=data.columns,index=data.in
scaled_data
```

Out[145]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median
0	0.211155	0.567481	0.784314	0.022331	0.019863	0.008941	0.020556	
1	0.212151	0.565356	0.392157	0.180503	0.171477	0.067210	0.186976	
2	0.210159	0.564293	1.000000	0.037260	0.029330	0.013818	0.028943	
3	0.209163	0.564293	1.000000	0.032352	0.036313	0.015555	0.035849	
4	0.209163	0.564293	1.000000	0.041330	0.043296	0.015752	0.042427	
...	...	...	...	...	...	...	...	
20635	0.324701	0.737513	0.470588	0.042296	0.057883	0.023599	0.054103	
20636	0.312749	0.738576	0.333333	0.017676	0.023122	0.009894	0.018582	
20637	0.311753	0.732200	0.313725	0.057277	0.075109	0.028140	0.071041	
20638	0.301793	0.732200	0.333333	0.047256	0.063315	0.020684	0.057227	
20639	0.309761	0.725824	0.294118	0.070782	0.095438	0.038790	0.086992	

20433 rows × 9 columns

In [146]:
```
scaled_data.describe()
```

Out[146]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	househ
count	20433.000000	20433.000000	20433.000000	20433.000000	20433.000000	20433.000000	20433.00
mean	0.476027	0.328716	0.541825	0.067005	0.083313	0.039854	0.08
std	0.199560	0.227030	0.246898	0.055579	0.065392	0.031761	0.06
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
25%	0.253984	0.147715	0.333333	0.036828	0.045779	0.021974	0.04
50%	0.583665	0.182784	0.549020	0.054046	0.067349	0.032596	0.06
75%	0.631474	0.550478	0.705882	0.079887	0.100248	0.048180	0.09
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00

Data Splitting

In [147]:
```
data=pd.read_csv("C:\\Users\\G.SAI KRISHNA\\Desktop\\ML_Projects\\ML_GFG\\housing.csv")
data.head()
```

Out[147]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_incor
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.32
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.30
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.25
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.64
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.84

In [148]:
```
data.describe()
```

Out[148]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	househ
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.00
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.53
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.32
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.00
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.00
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.00
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.00
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.00

In [149]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

In [150]:
```
data['total_bedrooms'] = data['total_bedrooms'].fillna(data['total_bedrooms'].mean())
```

In [151]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20640 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

In [152]:
```
data.shape
```

Out[152]:
```
(20640, 10)
```

Approach 1 - Manual splitting through Slicing

Splitting the data to input and output data

In [153]:
```
x = data.drop('median_house_value',axis=1)
x.head()
```

Out[153]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_incor
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.32
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.30
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.25
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.64
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.84

In [154]:
```
y = data['median_house_value']
y.head()
```

Out[154]:
```
0    452600.0
1    358500.0
2    352100.0
3    341300.0
4    342200.0
Name: median_house_value, dtype: float64
```

In [155]:
```
x.shape
```
Out[155]:
```
(20640, 9)
```
In [156]:
```
y.shape
```
Out[156]:
```
(20640,)
```
Further split the input and output into train data and test data

In [157]:
```
x_train = x.iloc[0:20000, :]
x_test = x.iloc[20000:,:]
```
In [158]:
```
y_train = y.iloc[0:20000]
y_test = y.iloc[20000:]
```
Approach 2 - Splitting data using sklearn

Everything is automated

In [159]:
```
from sklearn.model_selection import train_test_split
```
In [160]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2)
```
Handling Missing Values

In [161]:
```
data = pd.read_csv("C:\\Users\\G.SAI KRISHNA\\Desktop\\ML_Projects\\ML_GFG\\melb_data.csv"
data.head()
```
Out[161]:

	Unnamed: 0	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	...	Bathr
0	1	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	2.5	...	
1	2	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	2.5	...	
2	4	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	2.5	...	
3	5	Abbotsford	40 Federation La	3	h	850000.0	PI	Biggin	4/03/2017	2.5	...	
4	6	Abbotsford	55a Park St	4	h	1600000.0	VB	Nelson	4/06/2016	2.5	...	

5 rows × 22 columns

In [162]:
```python
data.describe()
```

Out[162]:

	Unnamed: 0	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom
count	18396.000000	18396.000000	1.839600e+04	18395.000000	18395.000000	14927.000000	14925.000000
mean	11826.787073	2.935040	1.056697e+06	10.389986	3107.140147	2.913043	1.538492
std	6800.710448	0.958202	6.419217e+05	6.009050	95.000995	0.964641	0.689311
min	1.000000	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000
25%	5936.750000	2.000000	6.330000e+05	6.300000	3046.000000	2.000000	1.000000
50%	11820.500000	3.000000	8.800000e+05	9.700000	3085.000000	3.000000	1.000000
75%	17734.250000	3.000000	1.302000e+06	13.300000	3149.000000	3.000000	2.000000
max	23546.000000	12.000000	9.000000e+06	48.100000	3978.000000	20.000000	8.000000

In [163]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18396 entries, 0 to 18395
Data columns (total 22 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Unnamed: 0     18396 non-null  int64  
 1   Suburb         18396 non-null  object 
 2   Address        18396 non-null  object 
 3   Rooms          18396 non-null  int64  
 4   Type           18396 non-null  object 
 5   Price          18396 non-null  float64
 6   Method         18396 non-null  object 
 7   SellerG        18396 non-null  object 
 8   Date           18396 non-null  object 
 9   Distance       18395 non-null  float64
 10  Postcode       18395 non-null  float64
 11  Bedroom2       14927 non-null  float64
 12  Bathroom       14925 non-null  float64
 13  Car            14820 non-null  float64
 14  Landsize       13603 non-null  float64
 15  BuildingArea   7762 non-null   float64
 16  YearBuilt      8958 non-null   float64
 17  CouncilArea    12233 non-null  object 
 18  Lattitude      15064 non-null  float64
 19  Longtitude     15064 non-null  float64
 20  Regionname     18395 non-null  object 
 21  Propertycount  18395 non-null  float64
dtypes: float64(12), int64(2), object(8)
memory usage: 3.1+ MB
```

Approach 1 - Dropping Missing Values Column wise

In [164]:
```
temp = data.dropna(axis=1)
temp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18396 entries, 0 to 18395
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  18396 non-null  int64  
 1   Suburb      18396 non-null  object 
 2   Address     18396 non-null  object 
 3   Rooms       18396 non-null  int64  
 4   Type        18396 non-null  object 
 5   Price       18396 non-null  float64
 6   Method      18396 non-null  object 
 7   SellerG     18396 non-null  object 
 8   Date        18396 non-null  object 
dtypes: float64(1), int64(2), object(6)
memory usage: 1.3+ MB
```

Approach 2 - Dropping Missing Values Row wise

In [165]:
```
temp = data.dropna(axis=0)
temp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6196 entries, 1 to 15395
Data columns (total 22 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Unnamed: 0     6196 non-null   int64  
 1   Suburb         6196 non-null   object 
 2   Address        6196 non-null   object 
 3   Rooms          6196 non-null   int64  
 4   Type           6196 non-null   object 
 5   Price          6196 non-null   float64
 6   Method         6196 non-null   object 
 7   SellerG        6196 non-null   object 
 8   Date           6196 non-null   object 
 9   Distance       6196 non-null   float64
 10  Postcode       6196 non-null   float64
 11  Bedroom2       6196 non-null   float64
 12  Bathroom       6196 non-null   float64
 13  Car            6196 non-null   float64
 14  Landsize       6196 non-null   float64
 15  BuildingArea   6196 non-null   float64
 16  YearBuilt      6196 non-null   float64
 17  CouncilArea    6196 non-null   object 
 18  Lattitude      6196 non-null   float64
 19  Longtitude     6196 non-null   float64
 20  Regionname     6196 non-null   object 
 21  Propertycount  6196 non-null   float64
dtypes: float64(12), int64(2), object(8)
memory usage: 1.1+ MB
```

Approach 3 - Using fillna method

In [166]:
```python
cols=['Bathroom','Car','Landsize','BuildingArea']
data[cols].head(10)
```

Out[166]:

	Bathroom	Car	Landsize	BuildingArea
0	1.0	1.0	202.0	NaN
1	1.0	0.0	156.0	79.0
2	2.0	0.0	134.0	150.0
3	2.0	1.0	94.0	NaN
4	1.0	2.0	120.0	142.0
5	1.0	0.0	181.0	NaN
6	2.0	0.0	245.0	210.0
7	1.0	2.0	256.0	107.0
8	NaN	NaN	NaN	NaN
9	NaN	NaN	NaN	NaN

In [167]:
```python
temp = data[cols].fillna(value = 999)
temp.head(10)
```

Out[167]:

	Bathroom	Car	Landsize	BuildingArea
0	1.0	1.0	202.0	999.0
1	1.0	0.0	156.0	79.0
2	2.0	0.0	134.0	150.0
3	2.0	1.0	94.0	999.0
4	1.0	2.0	120.0	142.0
5	1.0	0.0	181.0	999.0
6	2.0	0.0	245.0	210.0
7	1.0	2.0	256.0	107.0
8	999.0	999.0	999.0	999.0
9	999.0	999.0	999.0	999.0

In [168]:
```python
temp.mean()
```

Out[168]:
```
Bathroom        189.741846
Car             195.497173
Landsize        672.986736
BuildingArea    641.288179
dtype: float64
```

In [169]:
```
temp = data[cols].fillna(value = data[cols].mean())
temp.head(10)
```

Out[169]:

	Bathroom	Car	Landsize	BuildingArea
0	1.000000	1.00000	202.000000	151.220219
1	1.000000	0.00000	156.000000	79.000000
2	2.000000	0.00000	134.000000	150.000000
3	2.000000	1.00000	94.000000	151.220219
4	1.000000	2.00000	120.000000	142.000000
5	1.000000	0.00000	181.000000	151.220219
6	2.000000	0.00000	245.000000	210.000000
7	1.000000	2.00000	256.000000	107.000000
8	1.538492	1.61552	558.116371	151.220219
9	1.538492	1.61552	558.116371	151.220219

In [170]:
```
temp.mean()
```

Out[170]:
```
Bathroom          1.538492
Car               1.615520
Landsize        558.116371
BuildingArea    151.220219
dtype: float64
```

Approach 4 - Using Imputer

Strategy to handle missing values can be :

- Mean
- Median
- most_frequent

In [171]:
```
from sklearn.impute import SimpleImputer
```

In [172]:
```
imputer_mean = SimpleImputer(missing_values=np.nan,strategy="mean")
imputer_median = SimpleImputer(missing_values=np.nan,strategy="median")
imputer_mode = SimpleImputer(missing_values=np.nan,strategy="most_frequent")
```

In [173]:
```
imputer_data_mean = imputer_mean.fit_transform(data[cols])
imputer_data_median = imputer_median.fit_transform(data[cols])
imputer_data_mode = imputer_mode.fit_transform(data[cols])
```

In [174]:
```
df_mean = pd.DataFrame(data=imputer_data_mean,columns=cols)
df_mean.head(10)
```

Out[174]:

	Bathroom	Car	Landsize	BuildingArea
0	1.000000	1.00000	202.000000	151.220219
1	1.000000	0.00000	156.000000	79.000000
2	2.000000	0.00000	134.000000	150.000000
3	2.000000	1.00000	94.000000	151.220219
4	1.000000	2.00000	120.000000	142.000000
5	1.000000	0.00000	181.000000	151.220219
6	2.000000	0.00000	245.000000	210.000000
7	1.000000	2.00000	256.000000	107.000000
8	1.538492	1.61552	558.116371	151.220219
9	1.538492	1.61552	558.116371	151.220219

In [175]:
```
df_median = pd.DataFrame(data=imputer_data_median,columns=cols)
df_median.head(10)
```

Out[175]:

	Bathroom	Car	Landsize	BuildingArea
0	1.0	1.0	202.0	126.0
1	1.0	0.0	156.0	79.0
2	2.0	0.0	134.0	150.0
3	2.0	1.0	94.0	126.0
4	1.0	2.0	120.0	142.0
5	1.0	0.0	181.0	126.0
6	2.0	0.0	245.0	210.0
7	1.0	2.0	256.0	107.0
8	1.0	2.0	440.0	126.0
9	1.0	2.0	440.0	126.0

In [176]:
```
df_mode = pd.DataFrame(data=imputer_data_mode,columns=cols)
df_mode.head(10)
```

Out[176]:

	Bathroom	Car	Landsize	BuildingArea
0	1.0	1.0	202.0	120.0
1	1.0	0.0	156.0	79.0
2	2.0	0.0	134.0	150.0
3	2.0	1.0	94.0	120.0
4	1.0	2.0	120.0	142.0
5	1.0	0.0	181.0	120.0
6	2.0	0.0	245.0	210.0
7	1.0	2.0	256.0	107.0
8	1.0	2.0	0.0	120.0
9	1.0	2.0	0.0	120.0