In [9]:
```
import numpy as np
```

1.Creating a numpy array

In [36]:
```
#array from a python list
a=[2,4,6,8,10]
b=np.array(a)
```

In [37]:
```
b
```

Out[37]:
```
array([ 2,  4,  6,  8, 10])
```

In [38]:
```
a
```

Out[38]:
```
[2, 4, 6, 8, 10]
```

In [39]:
```
type(a)
```

Out[39]:
```
list
```

In [40]:
```
type(b)
```

Out[40]:
```
numpy.ndarray
```

In [42]:
```
list1=a
arr1=b
```

In [76]:
```
 list1+1
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-76-3611460f1106> in <module>
----> 1 list1+1

TypeError: can only concatenate list (not "int") to list
```

In [77]:
```
arr1+1
```

Out[77]:
```
array([ 3,  5,  7,  9, 11])
```

In [78]:
```
list2=[[9,0,4,6],
       [4,6,2,1]]
arr2=np.array(list2)
```

In [79]:
```
arr2
```

Out[79]:
```
array([[9, 0, 4, 6],
       [4, 6, 2, 1]])
```

In [80]:
```
#float array
arr2=np.array(list2,dtype='float')
```

In [81]:
```
arr2
```

Out[81]:
```
array([[9., 0., 4., 6.],
       [4., 6., 2., 1.]])
```

In [82]:
```
arr3=arr2.astype('int')
```

In [83]:
```
arr3
```

Out[83]:
```
array([[9, 0, 4, 6],
```

Out[83]:
```
array([[9, 0, 4, 6],
       [4, 6, 2, 1]])
```

In [84]:
```
arr4=np.array(list2,dtype='bool')
```

In [85]:
```
arr4
```

Out[85]:
```
array([[ True, False,  True,  True],
       [ True,  True,  True,  True]])
```

In [86]:
```
list2
```

Out[86]:
```
[[9, 0, 4, 6], [4, 6, 2, 1]]
```

In [87]:
```
 arr5=np.array([2,3,2.0,'y'],dtype='object' )
```

In [88]:
```
arr5
```

Out[88]:
```
array([2, 3, 2.0, 'y'], dtype=object)
```

In [89]:
```
#to convert arr to list
list3=arr5.tolist()
```

In [90]:
```
list3
```

Out[90]:
```
[2, 3, 2.0, 'y']
```

In [91]:
```
type(list3)
```

Out[91]:
```
list
```

In [92]:
```

a
```

Out[92]:
```
[2, 4, 6, 8, 10]
```

L2 ARRAY DIMENSION

In [94]:
```
arr2
```

Out[94]:
```
array([[9., 0., 4., 6.],
       [4., 6., 2., 1.]])
```

In [95]:
```
#shape of array
arr2.shape
```

Out[95]:
```
(2, 4)
```

In [96]:
```
#no. of elements
arr2.size
```

Out[96]:
```
8
```

In [97]:
```
#type
arr2.dtype
```

Out[97]:
```
dtype('float64')
```

In [98]:
```
 arr2.ndim
```

Out[98]:
```
2
```

L3 REVERSING ROWS AND COLUMNS

In [100]:
```
arr2
```

Out[100]:
```
array([[9., 0., 4., 6.],
       [4., 6., 2., 1.]])
```

In [101]:
```
#reverse rows
arr2[::-1]
```

Out[101]:
```
array([[4., 6., 2., 1.],
       [9., 0., 4., 6.]])
```

In [102]:
```
arr2[::-1,::-1]
```

Out[102]:
```
array([[1., 2., 6., 4.],
       [6., 4., 0., 9.]])
```

L4 SPECIFIC ELEMENT EXTRACTION

In [104]:
```
arr2
```

Out[104]:
```
array([[9., 0., 4., 6.],
       [4., 6., 2., 1.]])
```

In [105]:
```
arr2[0,:]
```

Out[105]:
```
array([9., 0., 4., 6.])
```

In [106]:
```
arr2[:1,:]
```

Out[106]:
```
array([[9., 0., 4., 6.]])
```

In [107]:
```
arr2[:2,:]
```

Out[107]:
```
array([[9., 0., 4., 6.],
       [4., 6., 2., 1.]])
```

In [108]:
```
arr2[:-1,:]
```

Out[108]:
```
array([[9., 0., 4., 6.]])
```

In [108]:
```
arr2[:-1,:]
```

Out[108]:
```
array([[9., 0., 4., 6.]])
```

In [109]:
```
#last column
arr2[:,:3]
```

Out[109]:
```
array([[9., 0., 4.],
       [4., 6., 2.]])
```

In [110]:
```
 arr2[:,:-1]
```

Out[110]:
```
array([[9., 0., 4.],
       [4., 6., 2.]])
```

In [111]:
```
arr2[:1,:3] #1,2
```

Out[111]:
```
array([[9., 0., 4.]])
```

I5 BASIC STATISTICS

In [112]:
```
 arr2
```

Out[112]:
```
array([[9., 0., 4., 6.],
       [4., 6., 2., 1.]])
```

In [113]:
```
arr2.min()
```

Out[113]:
```
0.0
```

In [114]:
```
arr2.max()
```

Out[114]:
```
9.0
```

In [115]:
```
arr2.sum()
```

Out[115]:
```
32.0
```

In [116]:
```
arr2.mean()
```

Out[116]:
```
4.0
```

In [118]:
```
np.median(arr2)
```

Out[118]:
```
4.0
```

In [120]:
```
np.average(arr2)
```

Out[120]:
```
4.0
```

In [121]:
```
#variance
np.var(arr2)
```

Out[121]:
```
7.75
```

I-6 RESHAPING AND FLATENNING

In [122]:
```
arr2
```

Out[122]:
```
array([[9., 0., 4., 6.],
       [4., 6., 2., 1.]])
```

In [123]:
```
arr2.shape
```

Out[123]:
```
(2, 4)
```

In [126]:
```
arr2.reshape(4,2)
```

Out[126]:
```
array([[9., 0.],
       [4., 6.],
       [4., 6.],
       [2., 1.]])
```

In [127]:
```
arr2.reshape(1,8)
```

Out[127]:
```
array([[9., 0., 4., 6., 4., 6., 2., 1.]])
```

In [128]:
```
arr2.reshape(8,1)
```

Out[128]:
```
array([[9.],
       [0.],
       [4.],
       [6.],
       [4.],
       [6.],
       [2.],
       [1.]])
```

In [129]:
```
arr2.reshape(2,2)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-129-7e35622d1c82> in <module>
----> 1 arr2.reshape(2,2)

ValueError: cannot reshape array of size 8 into shape (2,2)
```

In [131]:
```
#single dimension
f1=arr2.flatten()
```

In [133]:
```
f1.ndim
```

Out[133]:
```
1
```

L7 RANDOM ARRAYS AND SEQUENCES

In [135]:
```
np.arange(10)
```

Out[135]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [136]:
```
np.arange(2,10)
```

Out[136]:
```
array([2, 3, 4, 5, 6, 7, 8, 9])
```

In [137]:
```
np.arange(0,10,2)
```

Out[137]:
```
array([0, 2, 4, 6, 8])
```

In [138]:
```
np.arange(10,0)
```

Out[138]:
```
array([], dtype=int32)
```

In [139]:
```
#starting from 10 not 0 in descending order

np.arange(10,0,-1)
```

Out[139]:
```
array([10,  9,  8,  7,  6,  5,  4,  3,  2,  1])
```

In [141]:
```
#mention start,stop,total no of elements
#equal spaces

np.linspace(1,10,3)
```

Out[141]:
```
array([ 1. ,  5.5, 10. ])
```

In [142]:
```
np.linspace(1,10,3)
```

Out[142]:
```
array([ 1.  ,  3.25,  5.5 ,  7.75, 10.  ])
```

In [143]:
```
np.linspace(1,10,4)
```

Out[143]:
```
array([ 1.,  4.,  7., 10.])
```

In [145]:
```
np.zeros([2,3,4])
```

Out[145]:
```
array([[[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]],

       [[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]]])
```

I-8 UNIQUE ITEMS AND COUNT

In [155]:
```
arr=[[1,4,5,2,2,5],
[4,4,1,7,4,5]]
```

In [156]:
```
arr
```

Out[156]:
```
[[1, 4, 5, 2, 2, 5], [4, 4, 1, 7, 4, 5]]
```

In [157]:
```
u_val, count= np.unique(arr, return_counts=True)
```

In [158]:
```
u_val
```

Out[158]:
```
array([1, 2, 4, 5, 7])
```

In [159]:
```
  count
```

```
In [159]: count
```

Out[159]: array([2, 2, 4, 3, 1], dtype=int64)

2.PANDAS TUTORIAL I-9

In [ ]:

In [1]:
```
#LOADING LIBRARY
import pandas as pd
import numpy as np
```

In [2]:
```
#1.create Dataframe
data={
    'roll_no':[3,2,7,11],
    'ppr_id':[34,21,10,11],
    'marks':[30,23,17,27]
}
```

In [3]:
```
data
```

Out[3]:
```
{'roll_no': [3, 2, 7, 11],
 'ppr_id': [34, 21, 10, 11],
 'marks': [30, 23, 17, 27]}
```

In [4]:
```
df1=pd.DataFrame(data)
```

In [5]:
```
df1
```

Out[5]:

	roll_no	ppr_id	marks
0	3	34	30
1	2	21	23
2	7	10	17

Out[5]:

	roll_no	ppr_id	marks
0	3	34	30
1	2	21	23
2	7	10	17
3	11	11	27

In [6]:
```
#2 Setting index

df2=pd.DataFrame(data,index=['ab','ef','xy','uv'])
```

In [7]:
```
df2
```

Out[7]:

	roll_no	ppr_id	marks
ab	3	34	30
ef	2	21	23
xy	7	10	17
uv	11	11	27

In [8]:
```
#3 Extracting info
df2.loc['xy']
```

Out[8]:
```
roll_no     7
ppr_id     10
marks      17
Name: xy, dtype: int64
```

In [9]:
```
#3 Extracting info(col based,want last column data values)
df2.iloc[:,-1]
```

Out[9]:
```
ab    30
ef    23
xy    17
uv    27
Name: marks, dtype: int64
```

In [11]:
```
df2.iloc[0:2,2:3] #intersection of rollno and 2 column it returns
```

Out[11]:

	marks
ab	30
ef	23

I-10 working on csv file

In [ ]:
```
#1loading data
#csv file download link- https://www.kaggle.com/datasets/saurabh00007/iriscsv?resource=download
```

In [17]:
```
import pandas as pd
```

In [28]:
```
df = pd.read_csv('C:\\Users\\kriti\\Downloads\\archive\\Iris.csv')
```

In [29]:
```
df.head()
#print first 5 entries
```

Out[29]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [30]:
```
#for more entries
df.head(10)
```

Out[30]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

In [31]:
```
df.info
#shows info
```

Out[31]:
```
<bound method DataFrame.info of       Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
```

Out[31]:
```
<bound method DataFrame.info of       Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2      3            4.7           3.2            1.3           0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
..   ...            ...           ...            ...           ...
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
148  149            6.2           3.4            5.4           2.3
149  150            5.9           3.0            5.1           1.8

            Species
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              ...
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]>
```

In [32]:
```
#data description
df.describe()
```

```
df.describe()
```

Out[32]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [33]:
```
#data selection
```

In [39]:
```
df['SepalWidthCm'][:5]
# print first 5 columns of sepal_width
```

Out[39]:
```
0    3.5
1    3.0
2    3.2
3    3.1
4    3.6
Name: SepalWidthCm, dtype: float64
```

In [41]:
```
df[['SepalWidthCm']][:5]
#prints in form of data frame
```

Out[41]:

	SepalWidthCm
0	3.5
1	3.0
2	3.2
3	3.1
4	3.6

In [42]:
```
#two columns
df[['SepalWidthCm','PetalWidthCm']].head()
```

Out[42]:

	SepalWidthCm	PetalWidthCm
0	3.5	0.2
1	3.0	0.2
2	3.2	0.2
3	3.1	0.2
4	3.6	0.2

In [37]:
```
#till 3 but excluding 3
df.iloc[:10,1:3]
```

Out[37]:

	SepalLengthCm	SepalWidthCm
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2
3	4.6	3.1

3	4.6	3.1
4	5.0	3.6
5	5.4	3.9
6	4.6	3.4
7	5.0	3.4
8	4.4	2.9
9	4.9	3.1

In [38]:
```
df.iloc[:10,[1,3]]
```

Out[38]:

	SepalLengthCm	PetalLengthCm
0	5.1	1.4
1	4.9	1.4
2	4.7	1.3
3	4.6	1.5
4	5.0	1.4
5	5.4	1.7
6	4.6	1.4
7	5.0	1.5
8	4.4	1.4
9	4.9	1.5

5.Missing values I-11

5.Missing values I-11

In [45]:
```
import numpy as np
data={
    'roll_no':[3,2,7,11],
    'ppr_id':[34,21,10,11],
    'marks':[np.nan,23,17,27]
}
#np.nan is null value in python
```

In [46]:
```
df1=pd.DataFrame(data)
```

In [47]:
```
df1
```

Out[47]:

	roll_no	ppr_id	marks
0	3	34	NaN
1	2	21	23.0
2	7	10	17.0
3	11	11	27.0

In [52]:
```
# to check prrescence of null value in data set
df1.isnull()
```

Out[52]:

	roll_no	ppr_id	marks
0	False	False	True
1	False	False	False
2	False	False	False
3	False	False	False

In [53]:
```
df1.isnull().sum()
```

Out[53]:
```
roll_no    0
ppr_id     0
marks      1
dtype: int64
```

In [55]:
```
#fillna()   to get rid of null values
df2 = df1.fillna(1)
```

In [56]:
```
df2
#gets filled with 1
```

Out[56]:

	roll_no	ppr_id	marks
0	3	34	1.0
1	2	21	23.0
2	7	10	17.0
3	11	11	27.0

In [ ]:
```
#dropping NULL values
```

In [57]:
```
a=df1.dropna() # drop entire row
```

In [58]:
```
df1
```

Out[58]:

	roll_no	ppr_id	marks
0	3	34	NaN
1	2	21	23.0
2	7	10	17.0
3	11	11	27.0

In [59]:
```
a
```

Out[59]:

	roll_no	ppr_id	marks
1	2	21	23.0
2	7	10	17.0
3	11	11	27.0

In [60]:
```
#drow rows
```

In [62]:
```
a=df1.dropna(axis=0)
a
```

Out[62]:

	roll_no	ppr_id	marks
1	2	21	23.0
2	7	10	17.0
3	11	11	27.0

In [63]:
```
# drop column
a=df1.dropna(axis=1)
a
```

Out[63]:

	roll_no	ppr_id
0	3	34
1	2	21
2	7	10
3	11	11

In [64]:
```
# creating a data with non NULL value
a=pd.notnull(df1["marks"])
a
```

Out[64]:
```
0    False
1     True
2     True
3     True
Name: marks, dtype: bool
```

In [65]:
```
df1[a]
```

Out[65]:

	roll_no	ppr_id	marks
1	2	21	23.0
2	7	10	17.0
3	11	11	27.0

In [ ]:

6 Statistics I12

In [76]:
```
import numpy as np
data={
    'roll_no':[3,2,7,11],
    'ppr_id':[23,np.nan,10,11],
    'marks':[30,22,17,27]
}
```

In [77]:
```
data
```

Out[77]:
```
{'roll_no': [3, 2, 7, 11],
 'ppr_id': [23, nan, 10, 11],
 'marks': [30, 22, 17, 27]}
```

In [78]:
```
df1=pd.DataFrame(data)
```

In [79]:
```
df1
```

Out[79]:

	roll_no	ppr_id	marks
0	3	23.0	30
1	2	NaN	22
2	7	10.0	17
3	11	11.0	27

In [80]:
```
#calculate sum of particular column
```

In [81]:
```
df1['marks'].sum()
```

Out[81]: 96

In [82]:
```
#average marks
df1['marks'].mean()
```

Out[82]: 24.0

In [83]:
```
df1['marks'].cumsum()
#cummulativ sum
```

Out[83]:
```
0    30
1    52
2    69
3    96
Name: marks, dtype: int64
```

In [84]:
```
df1['marks'].count()
```

Out[84]: 4

In [85]:
```
df1['marks'].min()
```

Out[85]: 17

In [86]:
```
df1['marks'].max()
```

Out[86]: 30

In [87]:
```
df1['marks'].var()
```

Out[87]: 32.666666666666664

In [88]:
```
df1['marks'].std()
```

Out[88]: 5.715476066494082

In [86]:
```
df1['marks'].max()
```

Out[86]: 30

In [87]:
```
df1['marks'].var()
```

Out[87]: 32.666666666666664

In [88]:
```
df1['marks'].std()
```

Out[88]: 5.715476066494082

In [90]:
```
df1.corr()
#correlation
# +ve correlation means one is increasing other will also increase directly prortional
```

Out[90]:

	roll_no	ppr_id	marks
roll_no	1.000000	-0.829396	-0.028360
ppr_id	-0.829396	1.000000	0.727698
marks	-0.028360	0.727698	1.000000

matplotlib l13

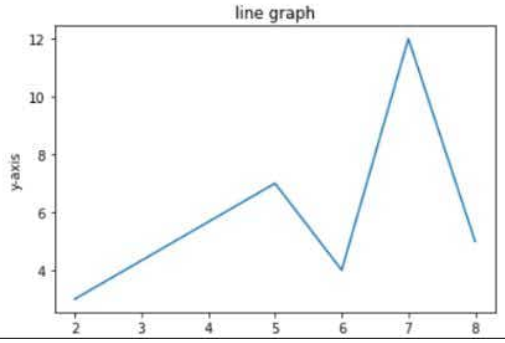
In [1]:
```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [ ]:
```
#Line Plot
```

In [2]:
```
x=[2,5,6,7,8]
y=[3,7,4,12,5]
```

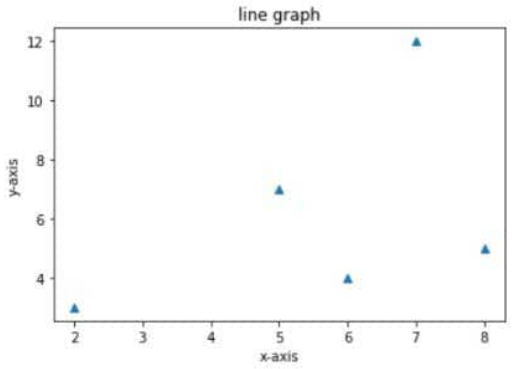
In [6]:
```
plt.plot(x,y)
plt.title("line graph")
plt.ylabel('y-axis')
plt.xlabel('x-axis')
plt.show
```

Out[6]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



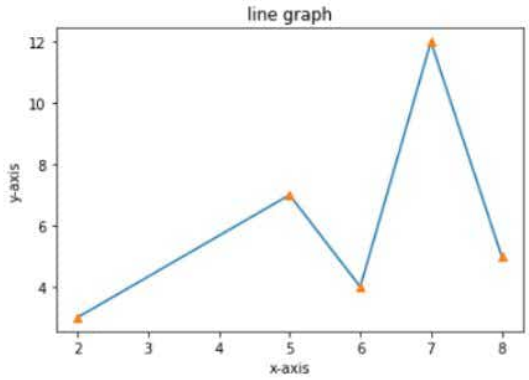
In [7]:
```
plt.plot(x,y,'^')
plt.title("line graph")
plt.ylabel('y-axis')
plt.xlabel('x-axis')
plt.show
```

Out[7]: <function matplotlib.pyplot.show(close=None, block=None)>



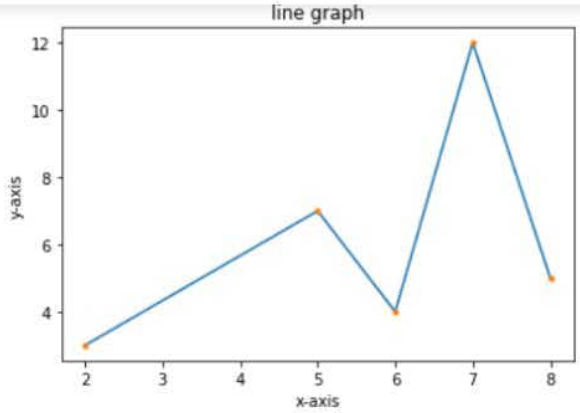
In [8]:
```
plt.plot(x,y)
plt.plot(x,y,'^')
plt.title("line graph")
plt.ylabel('y-axis')
plt.xlabel('x-axis')
plt.show
```

Out[8]: <function matplotlib.pyplot.show(close=None, block=None)>



In [9]:
```
plt.plot(x,y)
plt.plot(x,y,'.')
plt.title("line graph")
plt.ylabel('y-axis')
plt.xlabel('x-axis')
plt.show
```

Out[9]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [ ]:
```
#line grPH IN INPUT AND OUTPUT ..GIVE START END STEP SIZE    0TO 3 PI WITH STEP SIZE 0.1
```

In [15]:
```
x = np.arange(0,3*np.pi,0.1)
y=np.sin(x)
x
```

Out[15]:
```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2,
       1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5,
       2.6, 2.7, 2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8,
       3.9, 4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5. , 5.1,
       5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6. , 6.1, 6.2, 6.3, 6.4,
       6.5, 6.6, 6.7, 6.8, 6.9, 7. , 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7,
       7.8, 7.9, 8. , 8.1, 8.2, 8.3, 8.4, 8.5, 8.6, 8.7, 8.8, 8.9, 9. ,
       9.1, 9.2, 9.3, 9.4])
```

In [16]:
```
y
```

Out[16]:
```
array([ 0.        ,  0.09983342,  0.19866933,  0.29552021,  0.38941834,
        0.47942554,  0.56464247,  0.64421769,  0.71735609,  0.78332691,
        0.84147098,  0.89120736,  0.93203909,  0.96355819,  0.98544973,
```

```
5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6. , 6.1, 6.2, 6.3, 6.4,
6.5, 6.6, 6.7, 6.8, 6.9, 7. , 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7,
7.8, 7.9, 8. , 8.1, 8.2, 8.3, 8.4, 8.5, 8.6, 8.7, 8.8, 8.9, 9. ,
9.1, 9.2, 9.3, 9.4])
```

In [16]:
```
y
```

Out[16]:
```
array([ 0.        ,  0.09983342,  0.19866933,  0.29552021,  0.38941834,
        0.47942554,  0.56464247,  0.64421769,  0.71735609,  0.78332691,
        0.84147098,  0.89120736,  0.93203909,  0.96355819,  0.98544973,
        0.99749499,  0.9995736 ,  0.99166481,  0.97384763,  0.94630009,
        0.90929743,  0.86320937,  0.8084964 ,  0.74570521,  0.67546318,
        0.59847214,  0.51550137,  0.42737988,  0.33498815,  0.23924933,
        0.14112001,  0.04158066, -0.05837414, -0.15774569, -0.2555411 ,
       -0.35078323, -0.44252044, -0.52983614, -0.61185789, -0.68776616,
       -0.7568025 , -0.81827711, -0.87157577, -0.91616594, -0.95160207,
       -0.97753012, -0.993691  , -0.99992326, -0.99616461, -0.98245261,
       -0.95892427, -0.92581468, -0.88345466, -0.83226744, -0.77276449,
       -0.70554033, -0.63126664, -0.55068554, -0.46460218, -0.37387666,
       -0.2794155 , -0.1821625 , -0.0830894 ,  0.0168139 ,  0.1165492 ,
        0.21511999,  0.31154136,  0.40484992,  0.49411335,  0.57843976,
        0.6569866 ,  0.72896904,  0.79366786,  0.85043662,  0.8987081 ,
        0.93799998,  0.96791967,  0.98816823,  0.99854335,  0.99894134,
        0.98935825,  0.96988981,  0.94073056,  0.90217183,  0.85459891,
        0.79848711,  0.7343971 ,  0.66296923,  0.58491719,  0.50102086,
        0.41211849,  0.31909836,  0.22288991,  0.12445442,  0.02477543])
```

In [21]:
```
plt.plot(x,y)
plt.plot(x,y,'^')
plt.title("sin graph")
plt.ylabel('y-axis')
plt.xlabel('x-axis')
plt.show
```

Out[21]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```

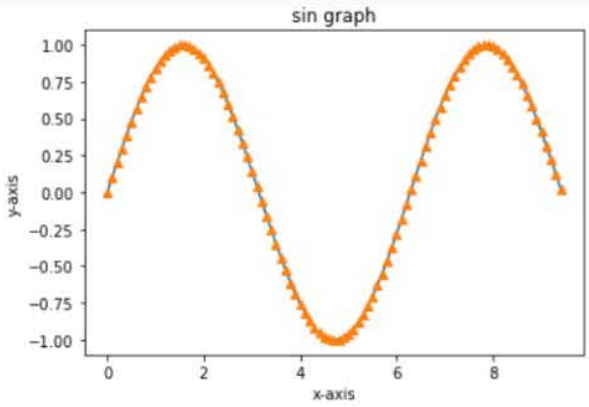
```
5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6. , 6.1, 6.2, 6.3, 6.4,
6.5, 6.6, 6.7, 6.8, 6.9, 7. , 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7,
7.8, 7.9, 8. , 8.1, 8.2, 8.3, 8.4, 8.5, 8.6, 8.7, 8.8, 8.9, 9. ,
9.1, 9.2, 9.3, 9.4])
```

In [16]:
```
y
```

Out[16]:
```
array([ 0.        ,  0.09983342,  0.19866933,  0.29552021,  0.38941834,
        0.47942554,  0.56464247,  0.64421769,  0.71735609,  0.78332691,
        0.84147098,  0.89120736,  0.93203909,  0.96355819,  0.98544973,
        0.99749499,  0.9995736 ,  0.99166481,  0.97384763,  0.94630009,
        0.90929743,  0.86320937,  0.8084964 ,  0.74570521,  0.67546318,
        0.59847214,  0.51550137,  0.42737988,  0.33498815,  0.23924933,
        0.14112001,  0.04158066, -0.05837414, -0.15774569, -0.2555411 ,
       -0.35078323, -0.44252044, -0.52983614, -0.61185789, -0.68776616,
       -0.7568025 , -0.81827711, -0.87157577, -0.91616594, -0.95160207,
       -0.97753012, -0.993691  , -0.99992326, -0.99616461, -0.98245261,
       -0.95892427, -0.92581468, -0.88345466, -0.83226744, -0.77276449,
       -0.70554033, -0.63126664, -0.55068554, -0.46460218, -0.37387666,
       -0.2794155 , -0.1821625 , -0.0830894 ,  0.0168139 ,  0.1165492 ,
        0.21511999,  0.31154136,  0.40484992,  0.49411335,  0.57843976,
        0.6569866 ,  0.72896904,  0.79366786,  0.85043662,  0.8987081 ,
        0.93799998,  0.96791967,  0.98816823,  0.99854335,  0.99894134,
        0.98935825,  0.96988981,  0.94073056,  0.90217183,  0.85459891,
        0.79848711,  0.7343971 ,  0.66296923,  0.58491719,  0.50102086,
        0.41211849,  0.31909836,  0.22288991,  0.12445442,  0.02477543])
```

In [21]:
```
plt.plot(x,y)
plt.plot(x,y,'^')
plt.title("sin graph")
plt.ylabel('y-axis')
plt.xlabel('x-axis')
plt.show
```

Out[21]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



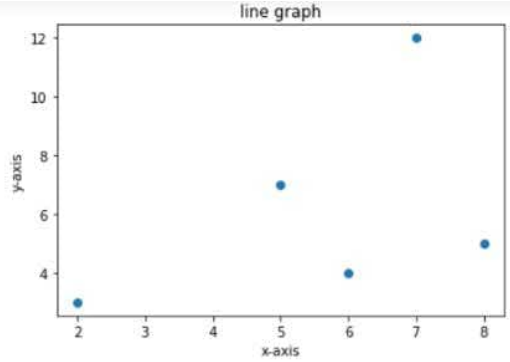
In [ ]:
```
#scatter plot
```

In [24]:
```
x=[2,5,6,7,8]
y=[3,7,4,12,5]
```

In [27]:
```
plt.scatter(x,y)

plt.title("line graph")
plt.ylabel('y-axis')
plt.xlabel('x-axis')
plt.show
```

Out[27]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



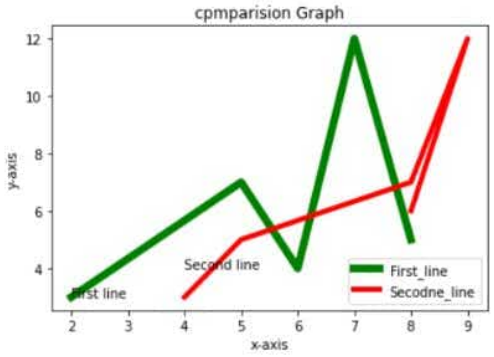
In [ ]:
```
#comparision
```

In [29]:
```
x=[2,5,6,7,8]
y=[3,7,4,12,5]

x2=[4,5,8,9,8]
y2=[3,5,7,12,6]
```

In [36]:
```
plt.plot(x,y,color='g',linewidth=6)
plt.plot(x2,y2,color='r',linewidth=4)
plt.title('comparision Graph')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
#marke line plots
#annotate in s is changed to text
plt.annotate(xy=[2,3],text='First line')
plt.annotate(xy=[4,4],text='Second line')
#the first line places the annotation at the point (2, 3),
#and the second line places it at (4, 4).
plt.legend(['First_line','Secodne_line'],loc=4)
plt.show
```

Out[36]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [ ]:
```
#vertical Bar Graph
```

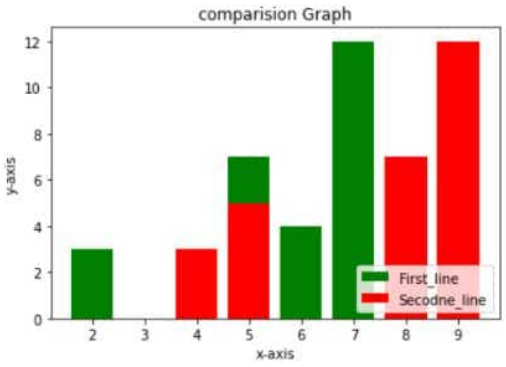
In [37]:
```
x=[2,5,6,7,8]
y=[3,7,4,12,5]

x2=[4,5,8,9,8]
y2=[3,5,7,12,6]

plt.bar(x,y,color='g',linewidth=6)
plt.bar(x2,y2,color='r',linewidth=4)
plt.title('comparision Graph')
plt.xlabel('x-axis')
plt.ylabel('y-axis')

plt.legend(['First_line','Secodne_line'],loc=4)
plt.show
```

Out[37]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



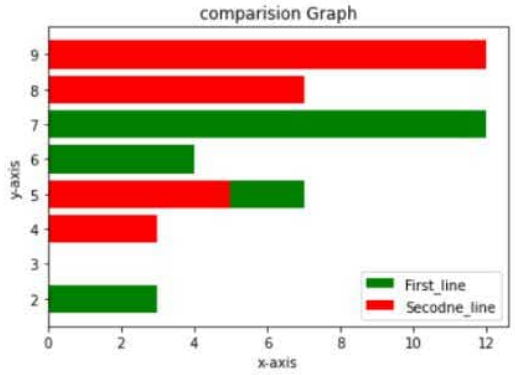
In [38]:
```
x=[2,5,6,7,8]
y=[3,7,4,12,5]

x2=[4,5,8,9,8]
y2=[3,5,7,12,6]

plt.barh(x,y,color='g',linewidth=6)
plt.barh(x2,y2,color='r',linewidth=4)
plt.title('comparision Graph')
plt.xlabel('x-axis')
plt.ylabel('y-axis')

plt.legend(['First_line','Secodne_line'],loc=4)
plt.show
```

Out[38]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [ ]:

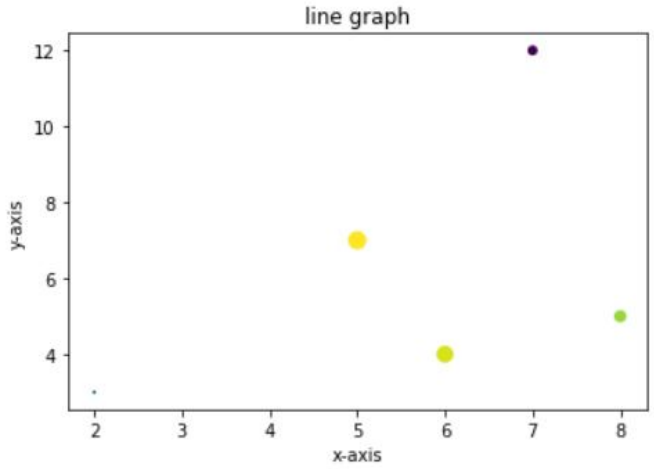
BUBBLE GRAPH L 15

In [ ]:
```
x=[2,5,6,7,8]
y=[3,7,4,12,5]

x2=[4,5,8,9,8]
y2=[3,5,7,12,6]
```

In [46]:
```
plt.scatter(x,y,s=np.random.rand(5)*100,c=np.random.rand(5))
#randomly initolize size and color
plt.title("line graph")
plt.ylabel('y-axis')
plt.xlabel('x-axis')
plt.show
```

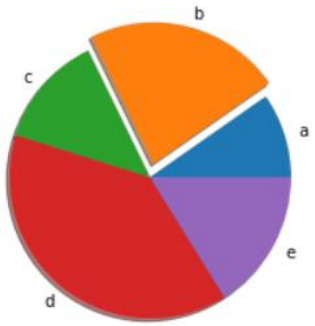
Out[46]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [ ]:
```
#pie chart
```

In [52]:
```
data=[3,7,4,12,5]
ex=(0,0.1,0,0,0)
#explode gives gaps
lb=['a','b','c','d','e']
plt.pie(data,labels=lb,explode=ex,shadow=True)
plt.show
```

Out[52]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Categorical Data I 16

In [54]:
```
#IMPORTING LIBRARY
import pandas as pd
import numpy as np
```

In [58]:
```
data=pd.read_csv('C:\\Users\\kriti\\Downloads\\tableConvert.com_fm4d25.csv')
```

In [65]:
```
data.head()
```

Out[65]:

	Unnamed: 0	EMPLOYEE_ID	GENDER	REMARKS	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9
0	1.0	48.0	MALE	NICE	NaN	NaN	NaN	NaN	NaN	NaN
1	2.0	78.0	FEMALE	GOOD	NaN	NaN	NaN	NaN	NaN	NaN
2	3.0	56.0	FEMALE	GREAT	NaN	NaN	NaN	NaN	NaN	NaN
3	4.0	12.0	MALE	GREAT	NaN	NaN	NaN	NaN	NaN	NaN
4	5.0	7.0	FEMALE	NICE	NaN	NaN	NaN	NaN	NaN	NaN

In [66]:
```
data.iloc[:5,1:4]
#added by me to show then actual table in class
```

Out[66]:

	EMPLOYEE_ID	GENDER	REMARKS
0	48.0	MALE	NICE
1	78.0	FEMALE	GOOD
2	56.0	FEMALE	GREAT
3	12.0	MALE	GREAT
4	7.0	FEMALE	NICE

In [67]:
```
#checking the label in the column
```

In [69]:
```
data['GENDER'].unique()
#to check different unique category og gender
```

Out[69]:
```
array(['MALE', 'FEMALE', nan], dtype=object)
```

In [70]:
```
data['REMARKS'].unique()
```

Out[70]:
```
array(['NICE', 'GOOD', 'GREAT', nan], dtype=object)
```

In [71]:
```
#checking count of each label
```

In [72]:
```
data['GENDER'].value_counts()
```

Out[72]:
```
FEMALE    3
MALE      2
Name: GENDER, dtype: int64
```

In [73]:
```
data['REMARKS'].value_counts()
```

Out[73]:
```
GREAT    2
NICE     2
GOOD     1
Name: REMARKS, dtype: int64
```

In [ ]:
```
#MATHEMATICAL DATA TO NUMERIC FORM
```

APPROCH 1- LABEL ENCODING

In [74]:
```
# Label Encoder encodes labels with value between 0 and number_of_class-1
```

In [79]:
```
from sklearn.preprocessing import LabelEncoder
label_encoder_X= LabelEncoder()
```

In [80]:
```
label_encoded_data=label_encoder_X.fit_transform(data['REMARKS'])
```

In [81]:
```
label_encoded_data=pd.DataFrame(data=label_encoded_data,columns=['REMARKS'])
#good ,nice ,great is converted into numbers
```

In [82]:
```
label_encoded_data
```

Out[82]:

	REMARKS
0	2
1	0
2	1
3	1
4	2
5	3
6	3
7	3
8	3

In [83]:
```
label_encoded_data['REMARKS'].unique()
```

Out[83]:
```
array([2, 0, 1, 3])
```

APPROACH 2-ONE HOT ENCODING

In [85]:
```
one_hot_encoded_data=pd.get_dummies(data,columns=['REMARKS','GENDER'])
```

In [86]:
```
one_hot_encoded_data
```

Out[86]:

YEE_ID	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	REMARKS_GOOD	REMARKS_GREAT	REMARKS_NICE	GENDER_FEMALE	GENDER
48.0	NaN	NaN	NaN	NaN	NaN	NaN	0	0	1	0	
78.0	NaN	NaN	NaN	NaN	NaN	NaN	1	0	0	1	
56.0	NaN	NaN	NaN	NaN	NaN	NaN	0	1	0	1	
12.0	NaN	NaN	NaN	NaN	NaN	NaN	0	1	0	0	
7.0	NaN	NaN	NaN	NaN	NaN	NaN	0	0	1	1	
NaN	NaN	NaN	NaN	NaN	NaN	NaN	0	0	0	0	
NaN	NaN	NaN	NaN	NaN	NaN	NaN	0	0	0	0	
NaN	NaN	NaN	NaN	NaN	NaN	NaN	0	0	0	0	
NaN	NaN	NaN	NaN	NaN	NaN	NaN	0	0	0	0	

In [87]:
```
data
```

Out[87]:

	Unnamed: 0	EMPLOYEE_ID	GENDER	REMARKS	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9
0	1.0	48.0	MALE	NICE	NaN	NaN	NaN	NaN	NaN	NaN
1	2.0	78.0	FEMALE	GOOD	NaN	NaN	NaN	NaN	NaN	NaN
2	3.0	56.0	FEMALE	GREAT	NaN	NaN	NaN	NaN	NaN	NaN
3	4.0	12.0	MALE	GREAT	NaN	NaN	NaN	NaN	NaN	NaN
4	5.0	7.0	FEMALE	NICE	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [88]:
```
#prints one in good position,0for female and 1 male
```

DATA SCALING L 17

In [ ]:
```
#predicting
```

In [1]:
```
#min -max Normalization-

#data scaling method to remove the bisaed

#this technique re-scales a feature or observation
#value with distribution value between 0 and 1

#X new=Xi-min(X) \max(x)-min(X)
```

DATA SCALING L 18

In [91]:
```
import pandas as pd
import numpy as np
```

In [92]:
```
data=pd.read_csv('C:\\Users\\kriti\\Downloads\\archive (1)\\housing.csv')
```

In [93]:
```
data.head(10)
```

Out[93]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY
5	-122.25	37.85	52.0	919.0	213.0	413.0	193.0	4.0368	269700.0	NEAR BAY
6	-122.25	37.84	52.0	2535.0	489.0	1094.0	514.0	3.6591	299200.0	NEAR BAY
7	-122.25	37.84	52.0	3104.0	687.0	1157.0	647.0	3.1200	241400.0	NEAR BAY
8	-122.26	37.84	42.0	2555.0	665.0	1206.0	595.0	2.0804	226700.0	NEAR BAY
9	-122.25	37.84	52.0	3549.0	707.0	1551.0	714.0	3.6912	261100.0	NEAR BAY

In [94]:
```
#scaling Data with MinMaxScalerr
```

In [95]:
```
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
import numpy as np
```

In [96]:
```
scaler=MinMaxScaler()
scaler.fit(data)
#error we ned to preprocess data before feeding
#it to MinMaxScaler remove null values etc
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-96-ef78f0619a74> in <module>
      1 scaler=MinMaxScaler()
----> 2 scaler.fit(data)
      3 #error we ned to preprocess data before feeding
      4 #it to MinMaxScaler remove null values etc
```

Note- (these steps are peformed to show the scaling part only)

In [7]:
```
data = data.drop(data.columns[[9]], axis = 1)
```

In [8]:
```
data = data.dropna(axis =0)
data.head(10)
```

Out[8]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0
5	-122.25	37.85	52.0	919.0	213.0	413.0	193.0	4.0368	269700.0
6	-122.25	37.84	52.0	2535.0	489.0	1094.0	514.0	3.6591	299200.0
7	-122.25	37.84	52.0	3104.0	687.0	1157.0	647.0	3.1200	241400.0
8	-122.26	37.84	42.0	2555.0	665.0	1206.0	595.0	2.0804	226700.0
9	-122.25	37.84	52.0	3549.0	707.0	1551.0	714.0	3.6912	261100.0

Now using scaling part

Now using scaling part

In [9]:
```
scaler = MinMaxScaler()
scaler.fit(data)
```

Out[9]:
```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

In [10]:
```
scaled_data = pd.DataFrame(data = scaler.transform(data), columns = data.columns, index = data.index)
```

In [11]:
```
scaled_data.head(10)
```

Out[11]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	0.211155	0.567481	0.784314	0.022331	0.019863	0.008941	0.020556	0.539668	0.902266
1	0.212151	0.565356	0.392157	0.180503	0.171477	0.067210	0.186976	0.538027	0.708247
2	0.210159	0.564293	1.000000	0.037260	0.029330	0.013818	0.028943	0.466028	0.695051
3	0.209163	0.564293	1.000000	0.032352	0.036313	0.015555	0.035849	0.354699	0.672783
4	0.209163	0.564293	1.000000	0.041330	0.043296	0.015752	0.042427	0.230776	0.674638
5	0.209163	0.564293	1.000000	0.023323	0.032899	0.011491	0.031574	0.243921	0.525155
6	0.209163	0.563231	1.000000	0.064423	0.075729	0.030578	0.084361	0.217873	0.585979
7	0.209163	0.563231	1.000000	0.078895	0.106456	0.032344	0.106233	0.180694	0.466804
8	0.208167	0.563231	0.803922	0.064932	0.103042	0.033717	0.097681	0.108998	0.436495
9	0.209163	0.563231	1.000000	0.090213	0.109559	0.043387	0.117250	0.220087	0.507423

DATA SPLITTING L19

In [104]:
```
#DATA
   #INPUT
   #OUTPUT
```

In [105]:
```
# IT C X TRAIN (INPUT) IT TRAINS USING THIS
#AND Y TRAIN (OUPUT) AND X TEST (WILL TEST ,
#PREDICT ACCORDING TO THE GIVEN VALUES) AND Y TEST
```

DATA SPLITTING L20

In [106]:
```
#IMPPORTING LIBRARIES

import pandas as pd
import numpy as np
```

In [108]:
```
#loding data
data=pd.read_csv('C:\\Users\\kriti\\Downloads\\archive (1)\\housing.csv')
```

In [109]:
```
data.head()
```

Out[109]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

In [110]:
```
data.describe()
```

Out[110]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

In [111]:
```
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 [112]:
```
data=data.fillna(value=537.870553)
#filling null values witl mean of column total_bedrooms
```

In [113]:
```
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 [114]:
```
data.shape
```

Out[114]:
```
(20640, 10)
```

APPROACH 1- MANUAL SPLITTING WITH SLICING

SPLITTING DATA TO INTUO AND OUTPUT DATA

In [115]:
```
x=data.drop('median_house_value',axis=1)
```

In [116]:
```
x.head()
```

Out[116]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	NEAR BAY

In [118]:
```
y=data['median_house_value']
```

In [119]:
```
y.head()
```

Out[119]:
```
0    452600.0
1    358500.0
2    352100.0
3    341300.0
4    342200.0
Name: median_house_value, dtype: float64
```

In [121]:
```
x.shape
```

Out[121]:
```
(20640, 9)
```

In [122]:
```
y.shape
```

Out[122]:
```
(20640,)
```

In [131]:
```
x_train=x.loc[0:20000,:]
x_test=x.loc[200000:, :]
y_train=y.loc[0:20000]
y_test=y.loc[200000:]
#single column
```

In [133]:
```
x.shape
```

Out[133]: (20640, 9)

In [134]:
```
y.shape
```

Out[134]: (20640,)

In [135]:
```
x_test.shape
```

Out[135]: (0, 9)

In [137]:
```
y_test.shape
```

Out[137]: (0,)

SPLITTING DATA USING SKLEARN

In [138]:
```
#RANDOMLY SELECTS ENTRY
```

In [140]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20)
#RANDOMLY SELECTS ENTRY 20% test
```

In [141]:
```
x_test.shape
```

Out[141]: (4128, 9)

In [142]:
```
y_test.shape
```

Out[142]: (4128,)

In [143]:
```
x.shape
```

Out[143]:
```
(20640, 9)
```

In [144]:
```
y.shape
```

Out[144]:
```
(20640,)
```

HANDLING MISSING VALUES L21

In [146]:
```
#IMPORTING LIBRARIES
import pandas as pd
import numpy as np
```

In [ ]:
```
#LOADING DATASET
```

In [147]:
```
data=pd.read_csv('C:\\Users\\kriti\\Downloads\\melb_data.csv\\melb_data.csv')
```

In [148]:
```
data.head(10)
```

Out[148]:

	Unnamed: 0	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	...	Bathroom	Car	Landsize	BuildingArea	YearBuilt	Co
0	1	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	2.5	...	1.0	1.0	202.0	NaN	NaN	
1	2	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	2.5	...	1.0	0.0	156.0	79.0	1900.0	
2	4	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	2.5	...	2.0	0.0	134.0	150.0	1900.0	
3	5	Abbotsford	40 Federation La	3	h	850000.0	PI	Biggin	4/03/2017	2.5	...	2.0	1.0	94.0	NaN	NaN	

4	6	Abbotsford	55a Park St	4	h	1600000.0	VB	Nelson	4/06/2016	2.5	...	1.0	2.0	120.0	142.0	2014.0
5	10	Abbotsford	129 Charles St	2	h	941000.0	S	Jellis	7/05/2016	2.5	...	1.0	0.0	181.0	NaN	NaN
6	11	Abbotsford	124 Yarra St	3	h	1876000.0	S	Nelson	7/05/2016	2.5	...	2.0	0.0	245.0	210.0	1910.0
7	14	Abbotsford	98 Charles St	2	h	1636000.0	S	Nelson	8/10/2016	2.5	...	1.0	2.0	256.0	107.0	1890.0
8	15	Abbotsford	217 Langridge St	3	h	1000000.0	S	Jellis	8/10/2016	2.5	...	NaN	NaN	NaN	NaN	NaN
9	16	Abbotsford	18a Mollison St	2	t	745000.0	S	Jellis	8/10/2016	2.5	...	NaN	NaN	NaN	NaN	NaN

10 rows × 22 columns

In [150]:
```
#data description
data.describe()
```

Out[150]:

	Unnamed: 0	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearB
count	18396.000000	18396.000000	1.839600e+04	18395.000000	18395.000000	14927.000000	14925.000000	14820.000000	13603.000000	7762.000000	8958.000
mean	11826.787073	2.935040	1.056697e+06	10.389986	3107.140147	2.913043	1.538492	1.615520	558.116371	151.220219	1965.879
std	6800.710448	0.958202	6.419217e+05	6.009050	95.000995	0.964641	0.689311	0.955916	3987.326586	519.188596	37.013
min	1.000000	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1196.000
25%	5936.750000	2.000000	6.330000e+05	6.300000	3046.000000	2.000000	1.000000	1.000000	176.500000	93.000000	1950.000
50%	11820.500000	3.000000	8.800000e+05	9.700000	3085.000000	3.000000	1.000000	2.000000	440.000000	126.000000	1970.000
75%	17734.250000	3.000000	1.302000e+06	13.300000	3149.000000	3.000000	2.000000	2.000000	651.000000	174.000000	2000.000
max	23546.000000	12.000000	9.000000e+06	48.100000	3978.000000	20.000000	8.000000	10.000000	433014.000000	44515.000000	2018.000

In [151]:
```
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
```

APPOACH1; DROPPING MISSING VALUES COLUMN_WISE

In [153]:
```
a=data.dropna(axis=1)
```

In [155]:
```
a.head()
```

Out[155]:

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

APPROACH 2: DROPPING MISSING VALUES ROW_WISE

In [157]:
```
a=data.dropna(axis=0)
```

In [158]:
```
a.head()
```

Out[158]:

	Unnamed: 0	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	...	Bathroom	Car	Landsize	BuildingArea	YearBuilt	Cou
1	2	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	2.5	...	1.0	0.0	156.0	79.0	1900.0	
2	4	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	2.5	...	2.0	0.0	134.0	150.0	1900.0	
4	6	Abbotsford	55a Park St	4	h	1600000.0	VB	Nelson	4/06/2016	2.5	...	1.0	2.0	120.0	142.0	2014.0	
6	11	Abbotsford	124 Yarra St	3	h	1876000.0	S	Nelson	7/05/2016	2.5	...	2.0	0.0	245.0	210.0	1910.0	

7	14	Abbotsford	98 Charles St	2	h	1636000.0	S	Nelson	8/10/2016	2.5	...	1.0	2.0	256.0	107.0	1890.0

5 rows × 22 columns

In [159]:
```
a.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 [161]:
```
cols=['Bathroom','Car','Landsize','BuildingArea']
```

In [163]:
```
data[cols].head(10)
```

Out[163]:

	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 [164]:
```
a=data[cols].fillna(value=999)
```

In [165]:
```
a.head(10)
```

Out[165]:

	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 [167]:
```
data[cols].describe()
```

Out[167]:

	Bathroom	Car	Landsize	BuildingArea
count	14925.000000	14820.000000	13603.000000	7762.000000
mean	1.538492	1.615520	558.116371	151.220219
std	0.689311	0.955916	3987.326586	519.188596
min	0.000000	0.000000	0.000000	0.000000
25%	1.000000	1.000000	176.500000	93.000000
50%	1.000000	2.000000	440.000000	126.000000
75%	2.000000	2.000000	651.000000	174.000000
max	8.000000	10.000000	433014.000000	44515.000000

APPROACH 4 -using Imputer

In [168]:
```
#stratergy to handle missing values can be
#mean
#median
#most_frequent
```

In [175]:
```
#from sklearn.preprocessing import Imputer
from sklearn.impute import SimpleImputer
```

In [177]:
```
imputer_mean = SimpleImputer(strategy='mean', missing_values=np.nan)
```

In [181]:
```
imputer_median = SimpleImputer(strategy='median', missing_values=np.nan)
```

In [182]:
```
#imputer_data_mean=imputer_mean.fit_transformI(data[cols])
imputer_data_mean = imputer_mean.fit_transform(data[cols])
```

In [183]:
```
imputer_data_median = imputer_median.fit_transform(data[cols])
```

In [184]:
```
imputer_data_mean=pd.DataFrame(data=imputer_data_mean,columns=cols)
```

In [185]:
```
imputer_data_median=pd.DataFrame(data=imputer_data_median,columns=cols)
```

In [189]:
```
imputer_data_mean
```

Out[189]:

	Bathroom	Car	Landsize	BuildingArea
0	1.0	1.0	202.000000	151.220219
1	1.0	0.0	156.000000	79.000000
2	2.0	0.0	134.000000	150.000000
3	2.0	1.0	94.000000	151.220219
4	1.0	2.0	120.000000	142.000000
...	...	...	...	...
18391	2.0	1.0	558.116371	89.000000
18392	1.0	5.0	866.000000	157.000000

18391	2.0	1.0	558.116371	89.000000
18392	1.0	5.0	866.000000	157.000000
18393	3.0	2.0	558.116371	151.220219
18394	1.0	1.0	362.000000	112.000000
18395	2.0	2.0	558.116371	139.000000

18396 rows × 4 columns

In [186]:
```
data[cols].describe()
```

Out[186]:

	Bathroom	Car	Landsize	BuildingArea
count	14925.000000	14820.000000	13603.000000	7762.000000
mean	1.538492	1.615520	558.116371	151.220219
std	0.689311	0.955916	3987.326586	519.188596
min	0.000000	0.000000	0.000000	0.000000
25%	1.000000	1.000000	176.500000	93.000000
50%	1.000000	2.000000	440.000000	126.000000
75%	2.000000	2.000000	651.000000	174.000000
max	8.000000	10.000000	433014.000000	44515.000000

In [187]:
```
imputer_data_mean.describe()
```

Out[187]:

	Bathroom	Car	Landsize	BuildingArea
count	18396.000000	18396.000000	18396.000000	18396.000000
mean	1.538492	1.615520	558.116371	151.220219
std	0.620880	0.857984	3428.730081	337.236125
min	0.000000	0.000000	0.000000	0.000000

In [187]:
```
imputer_data_mean.describe()
```

Out[187]:

	Bathroom	Car	Landsize	BuildingArea
count	18396.000000	18396.000000	18396.000000	18396.000000
mean	1.538492	1.615520	558.116371	151.220219
std	0.620880	0.857984	3428.730081	337.236125
min	0.000000	0.000000	0.000000	0.000000
25%	1.000000	1.000000	250.750000	140.000000
50%	1.538492	1.615520	558.116371	151.220219
75%	2.000000	2.000000	596.000000	151.220219
max	8.000000	10.000000	433014.000000	44515.000000

In [188]:
```
imputer_data_median.describe()
```

Out[188]:

	Bathroom	Car	Landsize	BuildingArea
count	18396.000000	18396.000000	18396.000000	18396.000000
mean	1.436888	1.690259	527.341650	136.641408
std	0.655656	0.871371	3429.122049	337.466074
min	0.000000	0.000000	0.000000	0.000000
25%	1.000000	1.000000	250.750000	126.000000
50%	1.000000	2.000000	440.000000	126.000000
75%	2.000000	2.000000	596.000000	126.000000
max	8.000000	10.000000	433014.000000	44515.000000