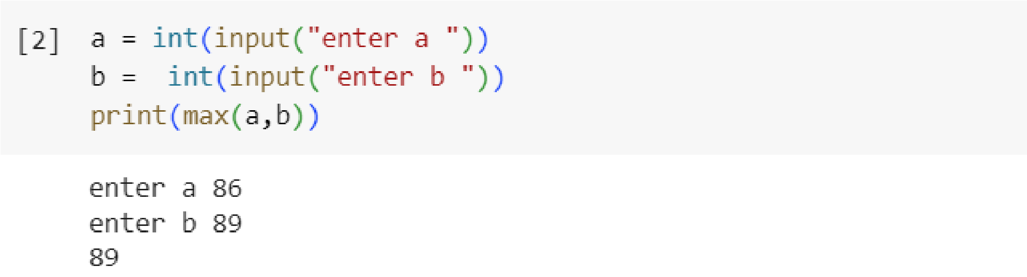
Program-1

Objective: To get input from user and perform numerical operations (MAX, MIN, AVG, SUM, SQRT, ROUND) using python.

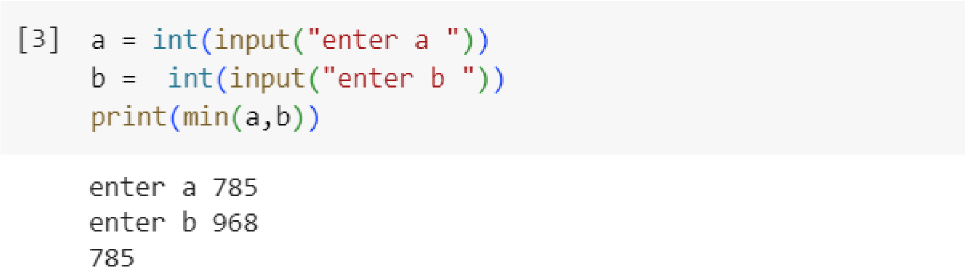
MAX:



```
[2]  a = int(input("enter a "))
     b =  int(input("enter b "))
     print(max(a,b))
```

```
enter a 86
enter b 89
89
```

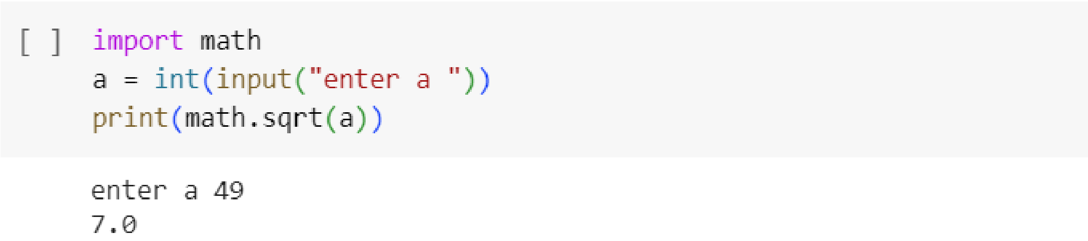
MIN:



```
[3]  a = int(input("enter a "))
     b =  int(input("enter b "))
     print(min(a,b))
```

```
enter a 785
enter b 968
785
```

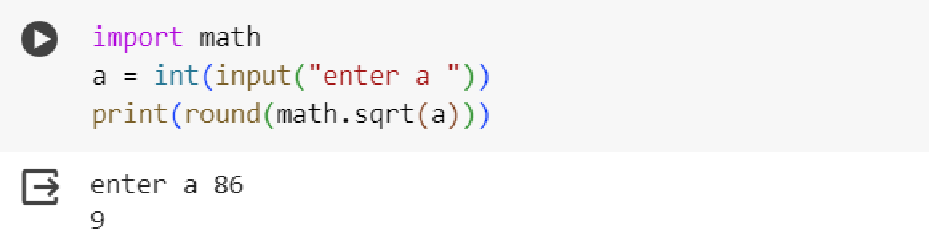
SQRT:



```
[ ]  import math
     a = int(input("enter a "))
     print(math.sqrt(a))
```

```
enter a 49
7.0
```

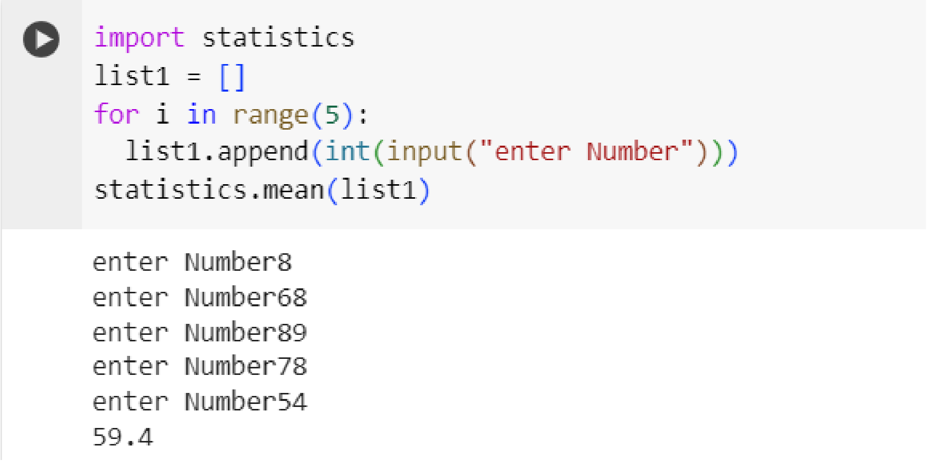
ROUND:



```
import math
a = int(input("enter a "))
print(round(math.sqrt(a)))
```

```
enter a 86
9
```

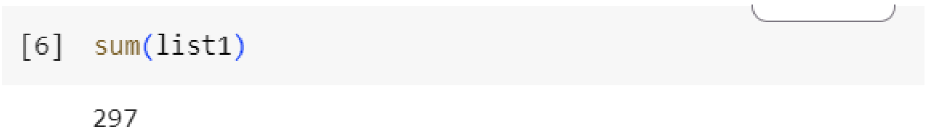
MEAN:



```
import statistics
list1 = []
for i in range(5):
  list1.append(int(input("enter Number")))
statistics.mean(list1)
```

```
enter Number8
enter Number68
enter Number89
enter Number78
enter Number54
59.4
```

SUM:



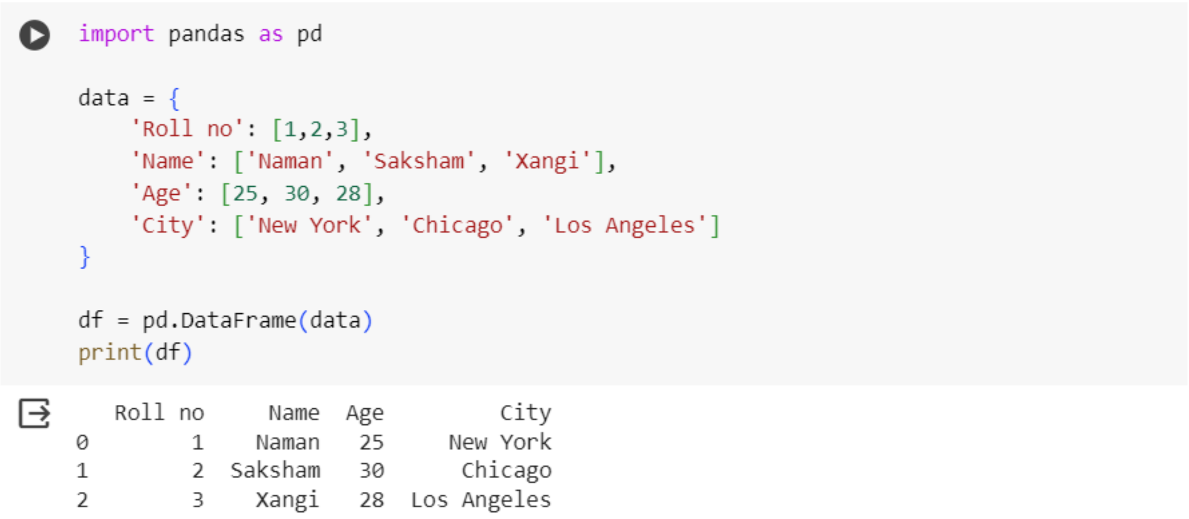
```
[6] sum(list1)
```

```
297
```

Program-2

Objective: To perform data import/export (.CSV, .XLSX, .TXT) operations using dataframes in python.

Creating Dataframes using Pandas Library :

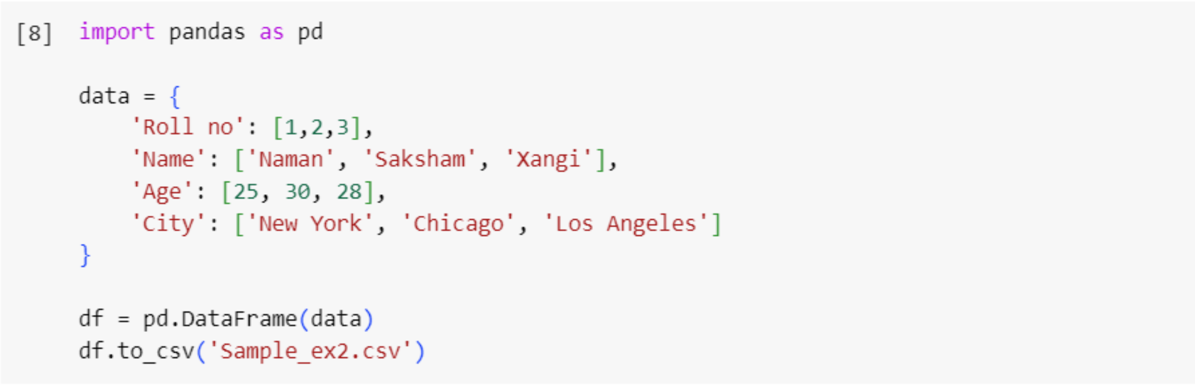


```
import pandas as pd

data = {
    'Roll no': [1,2,3],
    'Name': ['Naman', 'Saksham', 'Xangi'],
    'Age': [25, 30, 28],
    'City': ['New York', 'Chicago', 'Los Angeles']
}

df = pd.DataFrame(data)
print(df)
```

```
   Roll no     Name  Age         City
0        1    Naman   25     New York
1        2  Saksham   30      Chicago
2        3    Xangi   28  Los Angeles
```

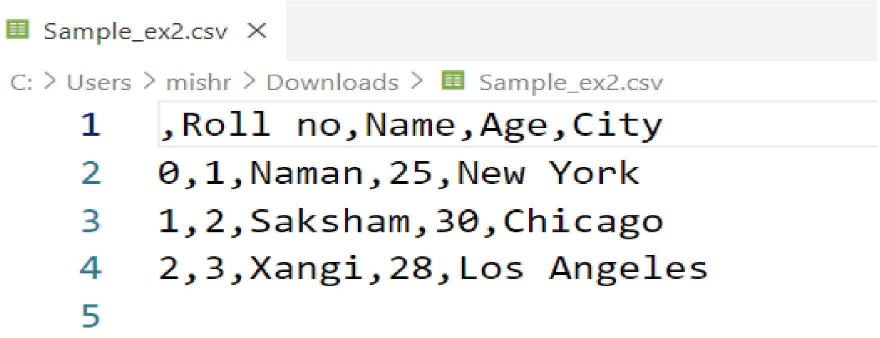
1 .CSV FILES

EXPORT CSV:



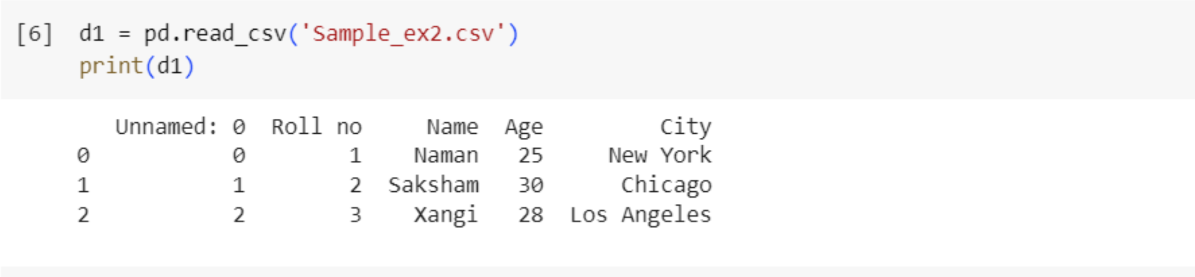
```
[8] import pandas as pd

    data = {
        'Roll no': [1,2,3],
        'Name': ['Naman', 'Saksham', 'Xangi'],
        'Age': [25, 30, 28],
        'City': ['New York', 'Chicago', 'Los Angeles']
    }

    df = pd.DataFrame(data)
    df.to_csv('Sample_ex2.csv')
```

CSV FILE:



```
,Roll no,Name,Age,City
0,1,Naman,25,New York
1,2,Saksham,30,Chicago
2,3,Xangi,28,Los Angeles

```

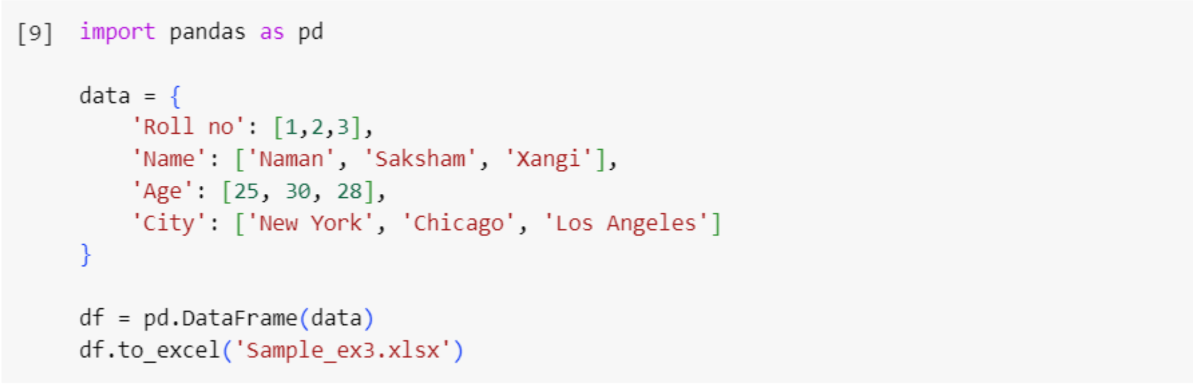
IMPORT CSV:



```
[6]  d1 = pd.read_csv('Sample_ex2.csv')
     print(d1)

        Unnamed: 0  Roll no     Name  Age         City
     0           0        1    Naman   25     New York
     1           1        2  Saksham   30      Chicago
     2           2        3    Xangi   28  Los Angeles
```

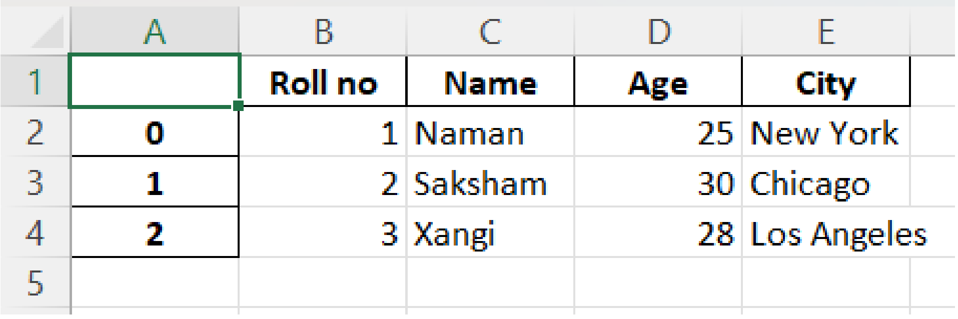
2. EXCEL FILES

EXPORT EXCEL:



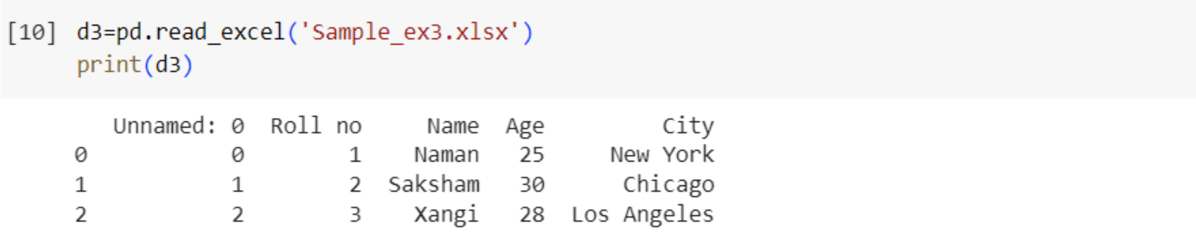
```
[9]  import pandas as pd

     data = {
         'Roll no': [1,2,3],
         'Name': ['Naman', 'Saksham', 'Xangi'],
         'Age': [25, 30, 28],
         'City': ['New York', 'Chicago', 'Los Angeles']
     }

     df = pd.DataFrame(data)
     df.to_excel('Sample_ex3.xlsx')
```

EXCEL FILE:



	A	B	C	D	E
1		Roll no	Name	Age	City
2	0	1	Naman	25	New York
3	1	2	Saksham	30	Chicago
4	2	3	Xangi	28	Los Angeles
5					

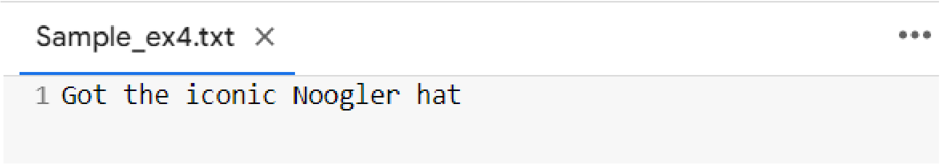
IMPORT EXCEL:



```
[10] d3=pd.read_excel('Sample_ex3.xlsx')
     print(d3)

        Unnamed: 0  Roll no     Name  Age         City
     0           0        1    Naman   25     New York
     1           1        2  Saksham   30      Chicago
     2           2        3    Xangi   28  Los Angeles
```

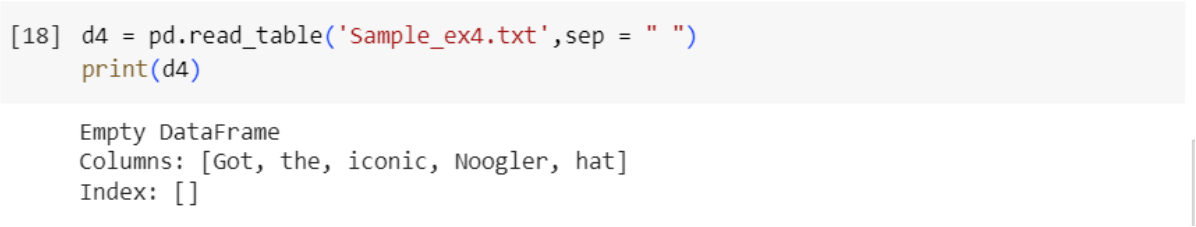
3. TEXT FILES

.TXT FILE:



```
Got the iconic Noogler hat
```

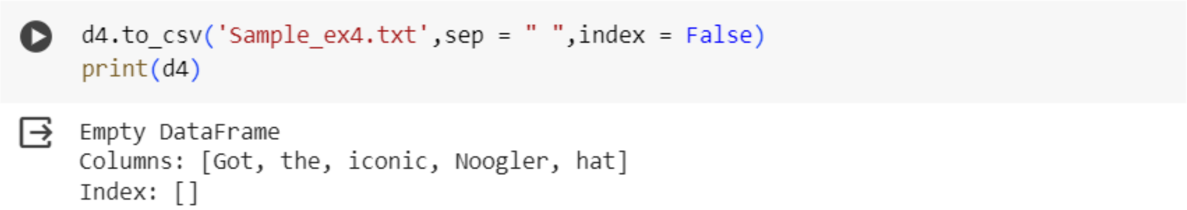
IMPORT TXT:



```
[18] d4 = pd.read_table('Sample_ex4.txt',sep = " ")
     print(d4)
```

```
Empty DataFrame
Columns: [Got, the, iconic, Noogler, hat]
Index: []
```

EXPORT TXT:



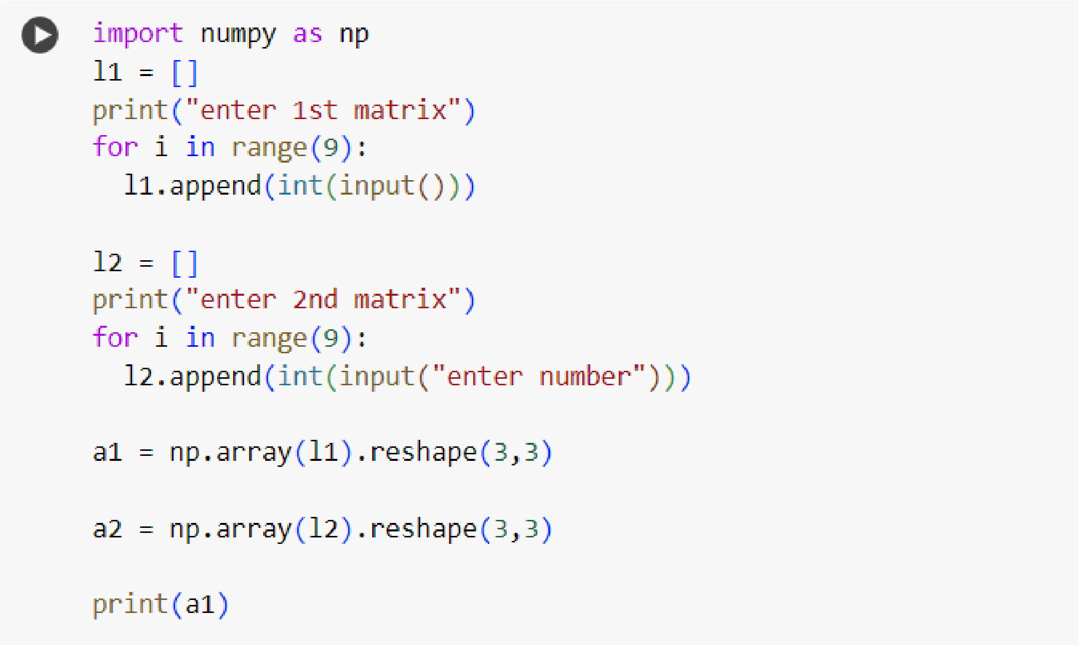
```
d4.to_csv('Sample_ex4.txt',sep = " ",index = False)
print(d4)
```

```
Empty DataFrame
Columns: [Got, the, iconic, Noogler, hat]
Index: []
```

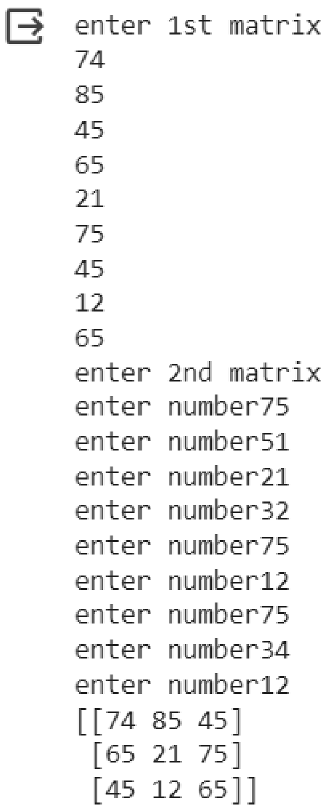
Program-3

Objective: To get input matrix from user and perform Matrix addition, Subtraction , Multiplication, inverse Transpose using python.

INPUT MATRIX:

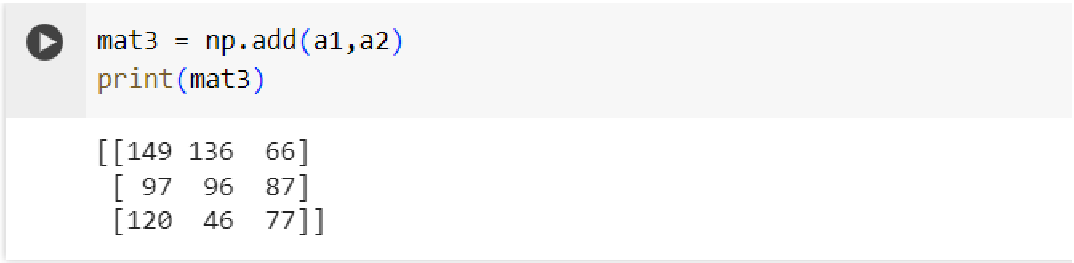


```
import numpy as np
l1 = []
print("enter 1st matrix")
for i in range(9):
  l1.append(int(input()))

l2 = []
print("enter 2nd matrix")
for i in range(9):
  l2.append(int(input("enter number")))

a1 = np.array(l1).reshape(3,3)

a2 = np.array(l2).reshape(3,3)

print(a1)
```



```
enter 1st matrix
74
85
45
65
21
75
45
12
65
enter 2nd matrix
enter number75
enter number51
enter number21
enter number32
enter number75
enter number12
enter number75
enter number34
enter number12
[[74 85 45]
 [65 21 75]
 [45 12 65]]
```

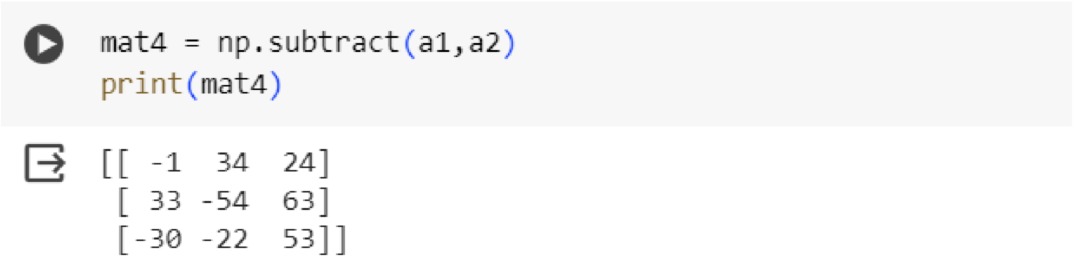
ADDITION:



```
mat3 = np.add(a1,a2)
print(mat3)
```

```
[[149 136  66]
 [ 97  96  87]
 [120  46  77]]
```

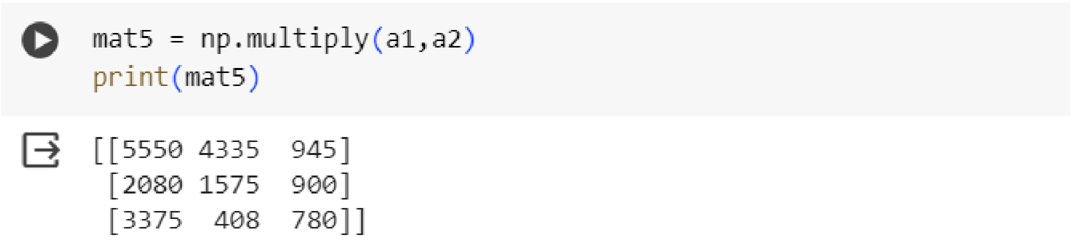
SUBTRACTION:



```
mat4 = np.subtract(a1,a2)
print(mat4)
```

```
[[ -1  34  24]
 [ 33 -54  63]
 [-30 -22  53]]
```

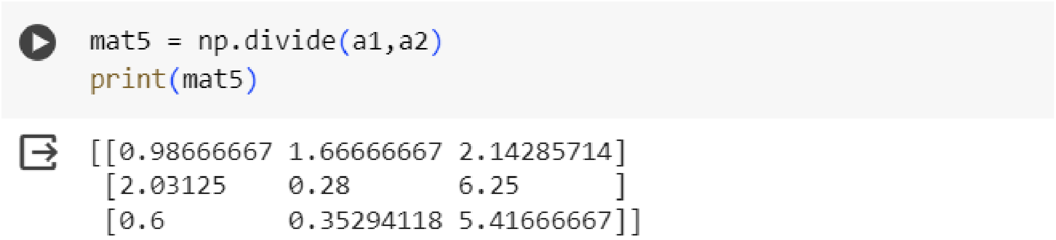
MULTIPLICATION:



```
mat5 = np.multiply(a1,a2)
print(mat5)
```

```
[[5550 4335  945]
 [2080 1575  900]
 [3375  408  780]]
```

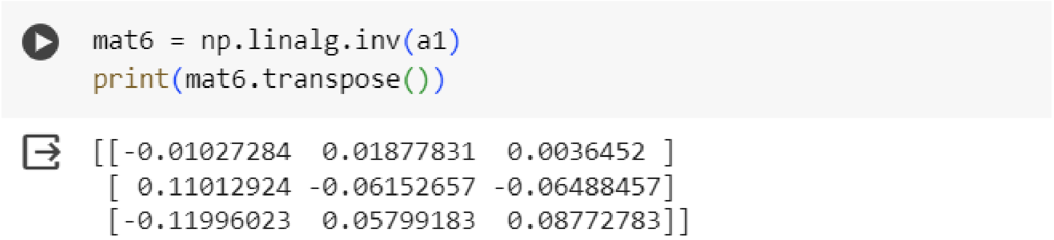
DIVISION:



```
mat5 = np.divide(a1,a2)
print(mat5)
```

```
[[0.98666667 1.66666667 2.14285714]
 [2.03125    0.28       6.25      ]
 [0.6        0.35294118 5.41666667]]
```

INVERSE TRANSPOSE:



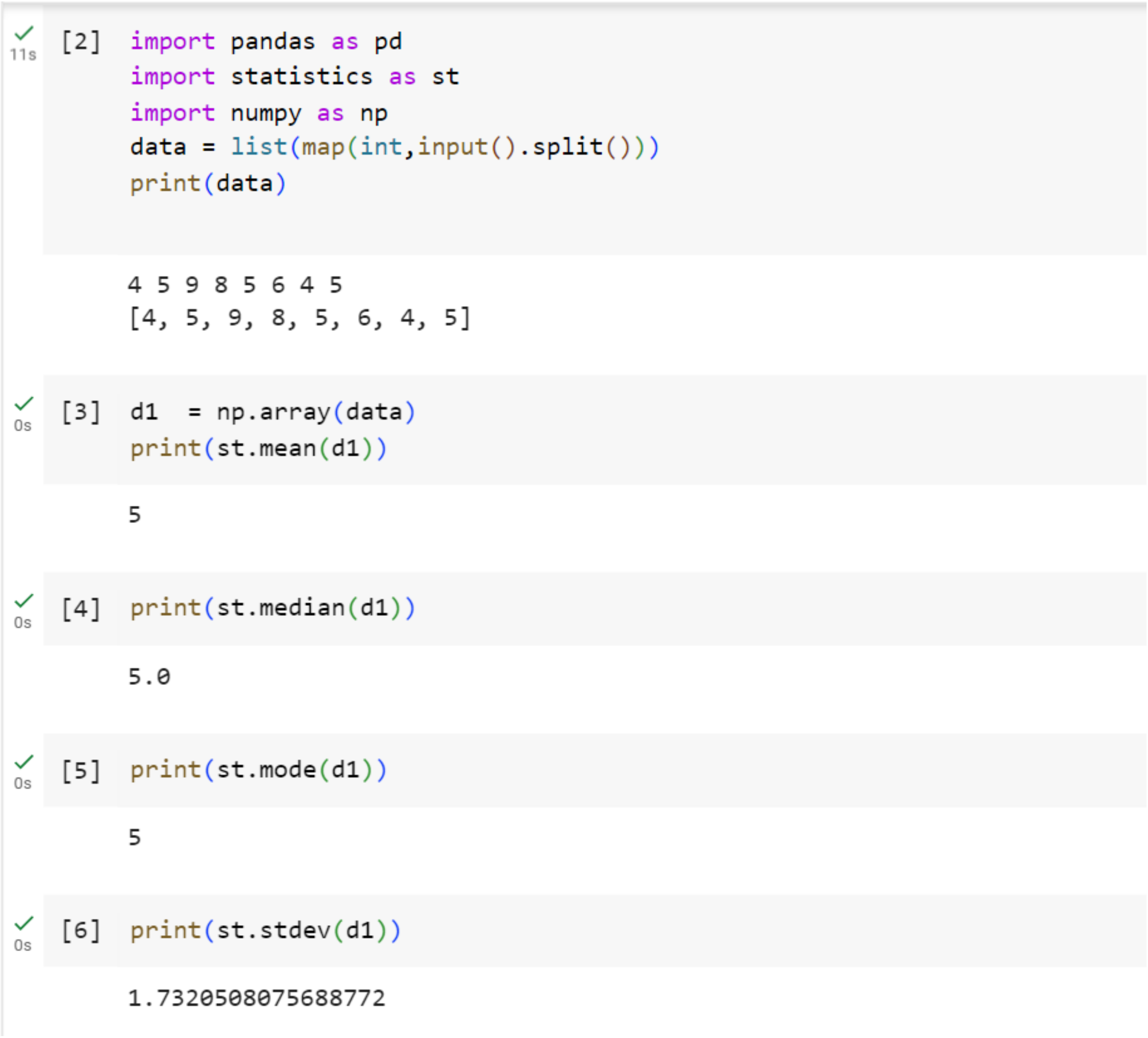
```
mat6 = np.linalg.inv(a1)
print(mat6.transpose())
```

```
[[-0.01027284  0.01877831  0.0036452 ]
 [ 0.11012924 -0.06152657 -0.06488457]
 [-0.11996023  0.05799183  0.08772783]]
```

Program-4

Objective: To perform statistical operations (Mean, Median, Mode, Standard Deviation)

OPERATIONS ON ARRAY:



```
import pandas as pd
import statistics as st
import numpy as np
data = list(map(int,input().split()))
print(data)
```

```
4 5 9 8 5 6 4 5
[4, 5, 9, 8, 5, 6, 4, 5]
```

```
d1  = np.array(data)
print(st.mean(d1))
```

```
5
```

```
print(st.median(d1))
```

```
5.0
```

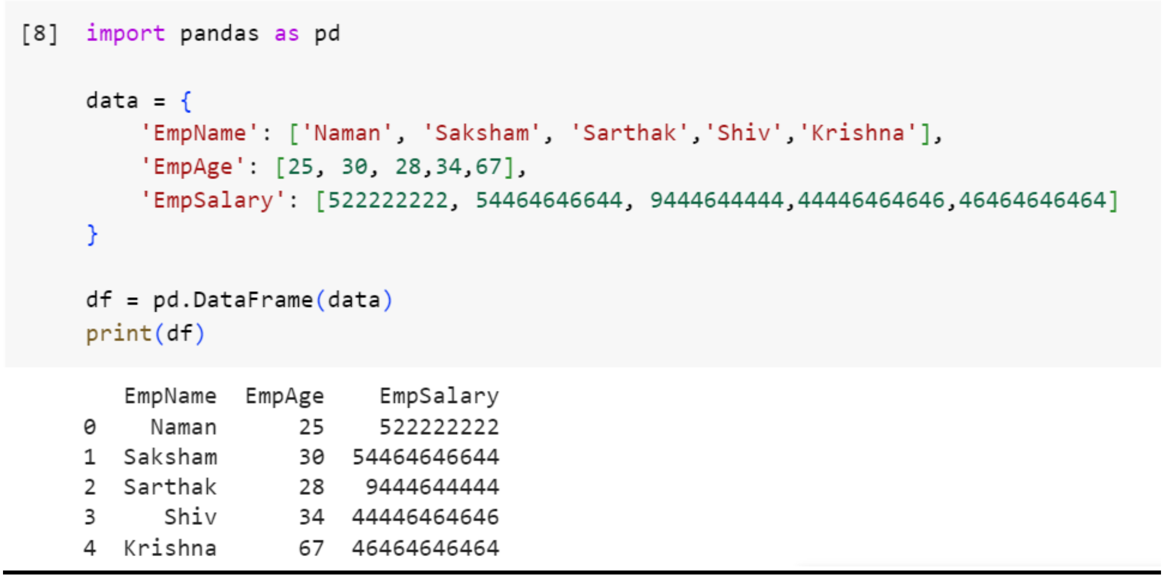
```
print(st.mode(d1))
```

```
5
```

```
print(st.stdev(d1))
```

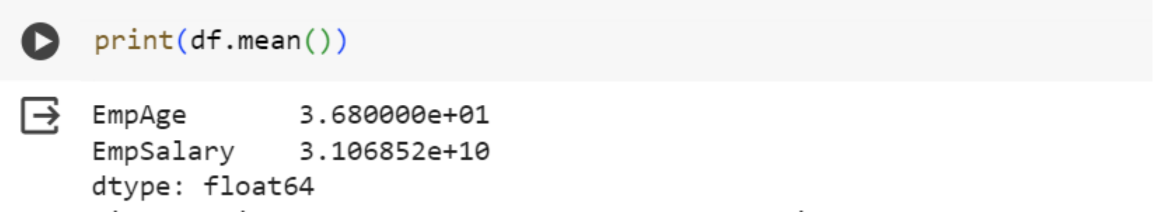
```
1.7320508075688772
```

OPERATIONS ON DATAFRAMES:



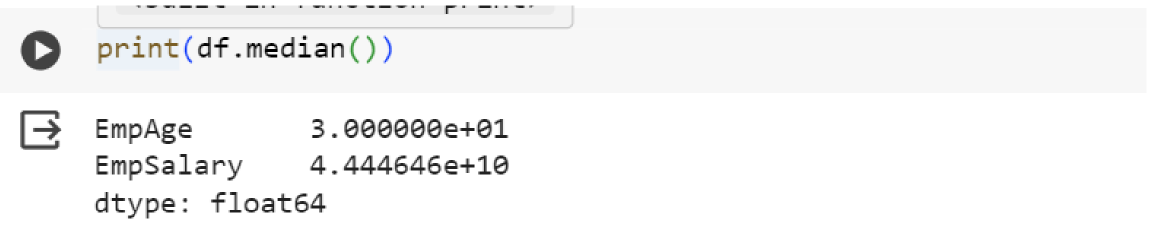
```
[8]  import pandas as pd

     data = {
         'EmpName': ['Naman', 'Saksham', 'Sarthak','Shiv','Krishna'],
         'EmpAge': [25, 30, 28,34,67],
         'EmpSalary': [522222222, 54464646644, 9444644444,44446464646,46464646464]
     }

     df = pd.DataFrame(data)
     print(df)
```

```
   EmpName  EmpAge    EmpSalary
0    Naman      25    522222222
1  Saksham      30  54464646644
2  Sarthak      28   9444644444
3     Shiv      34  44446464646
4  Krishna      67  46464646464
```



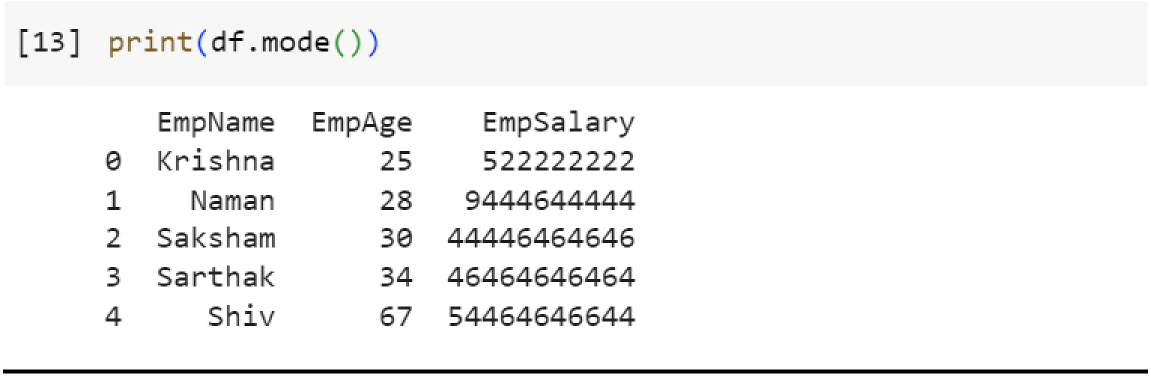
```
print(df.mean())
```

```
EmpAge       3.680000e+01
EmpSalary    3.106852e+10
dtype: float64
```



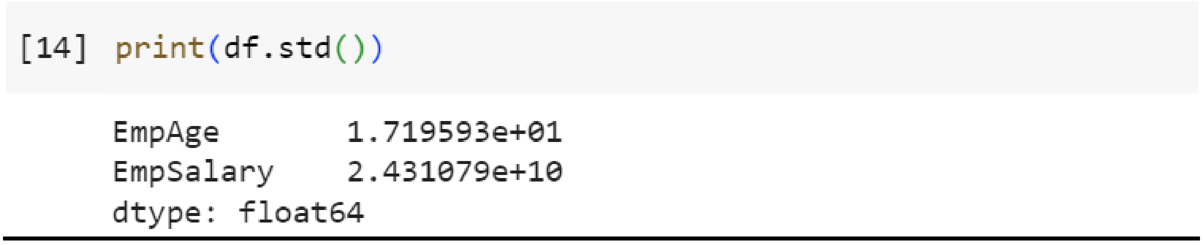
```
print(df.median())
```

```
EmpAge       3.000000e+01
EmpSalary    4.444646e+10
dtype: float64
```



```
[13] print(df.mode())
```

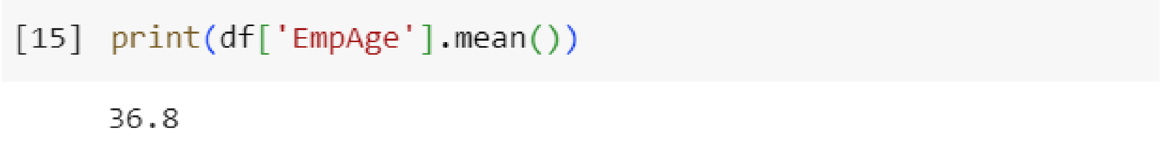
```
   EmpName  EmpAge    EmpSalary
0  Krishna      25    522222222
1    Naman      28   9444644444
2  Saksham      30  44446464646
3  Sarthak      34  46464646464
4     Shiv      67  54464646644
```



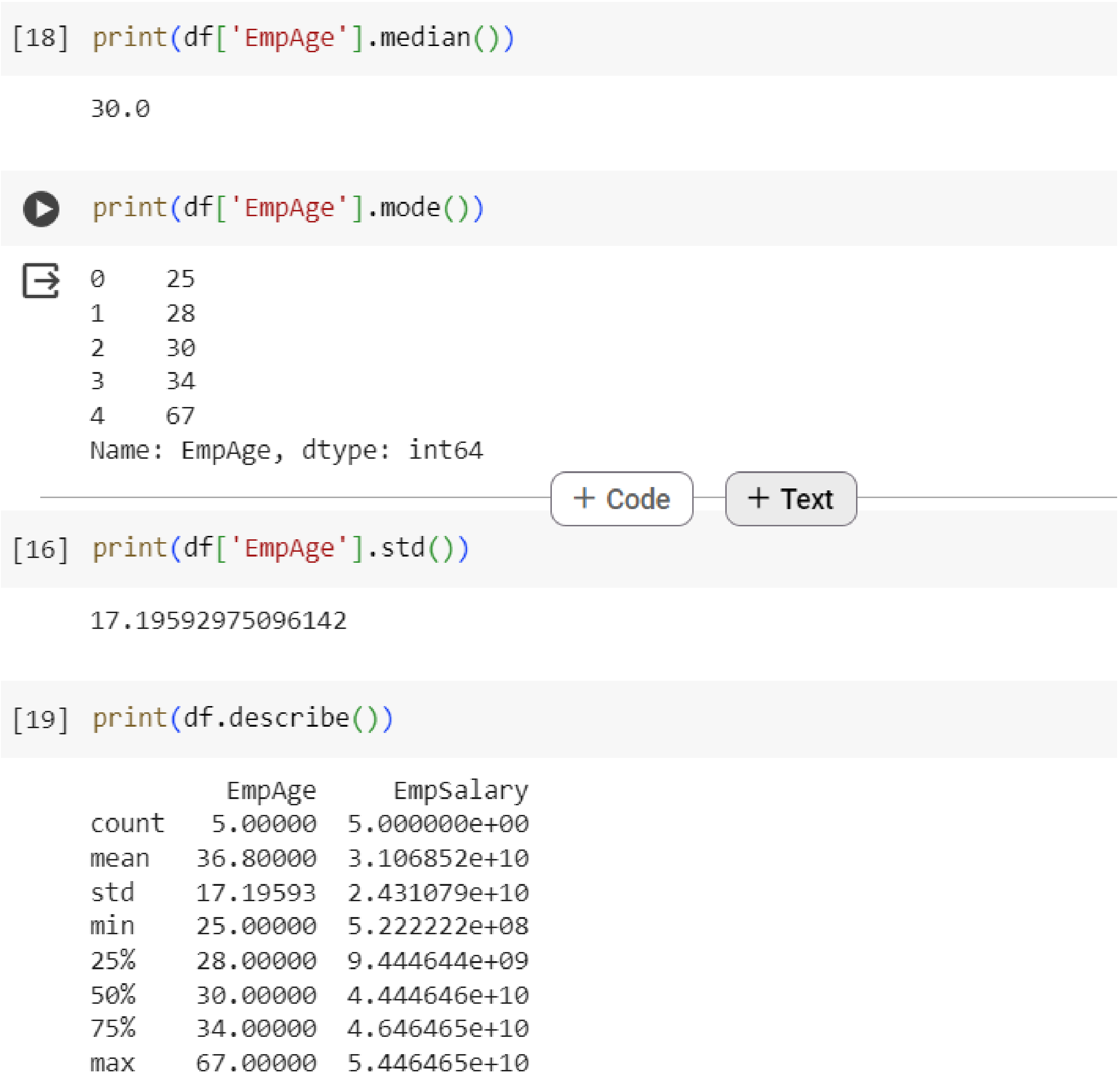
```
[14] print(df.std())
```

```
EmpAge       1.719593e+01
EmpSalary    2.431079e+10
dtype: float64
```

OPERATIONS ON SELECTED COLUMN ON DATAFRAMES:



```
[15] print(df['EmpAge'].mean())

     36.8
```



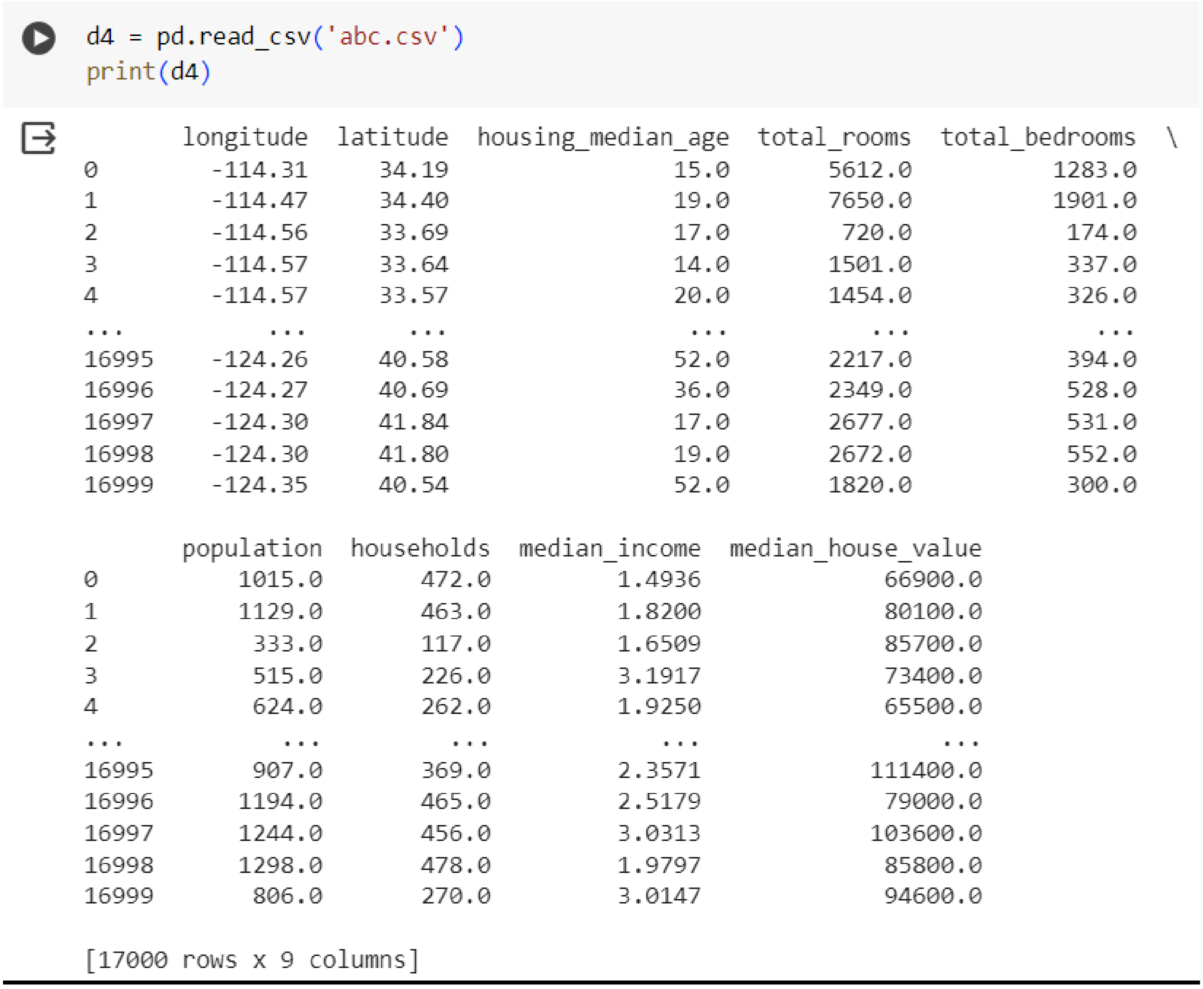
```
[18] print(df['EmpAge'].median())

     30.0
```

```
print(df['EmpAge'].mode())

0    25
1    28
2    30
3    34
4    67
Name: EmpAge, dtype: int64
```

```
[16] print(df['EmpAge'].std())

     17.19592975096142
```

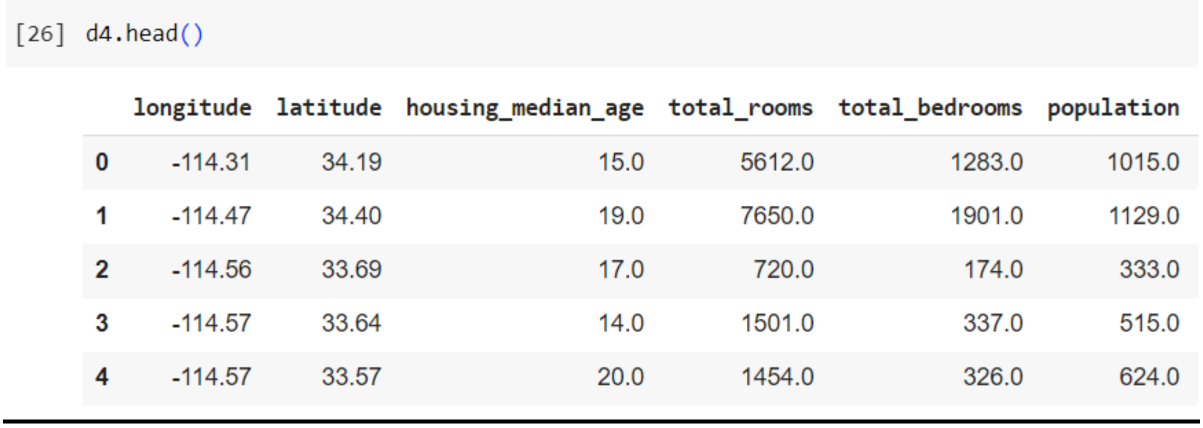
```
[19] print(df.describe())

            EmpAge     EmpSalary
     count   5.00000  5.000000e+00
     mean   36.80000  3.106852e+10
     std    17.19593  2.431079e+10
     min    25.00000  5.222222e+08
     25%    28.00000  9.444644e+09
     50%    30.00000  4.444646e+10
     75%    34.00000  4.646465e+10
     max    67.00000  5.446465e+10
```

OPERATIONS ON .CSV FILE:



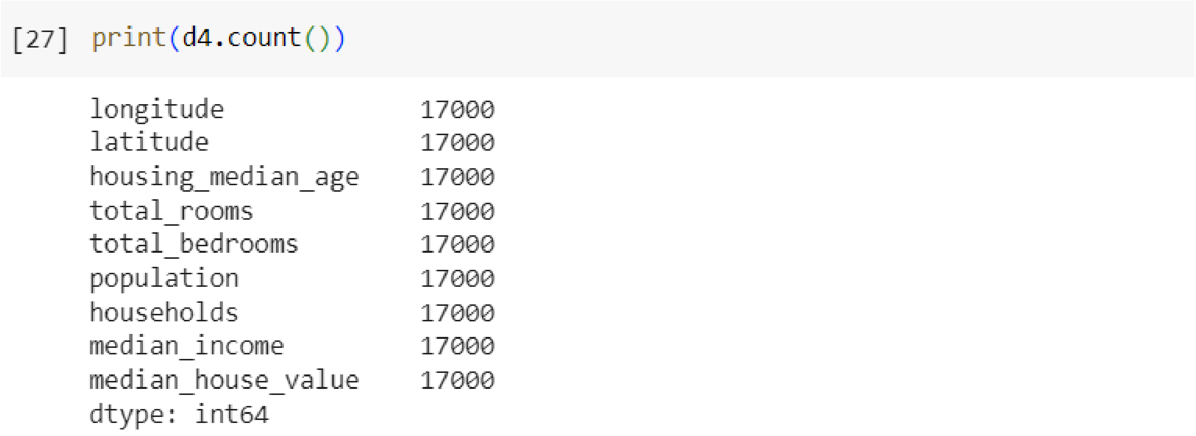
```
d4 = pd.read_csv('abc.csv')
print(d4)
```

```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0        -114.31     34.19                15.0       5612.0          1283.0
1        -114.47     34.40                19.0       7650.0          1901.0
2        -114.56     33.69                17.0        720.0           174.0
3        -114.57     33.64                14.0       1501.0           337.0
4        -114.57     33.57                20.0       1454.0           326.0
...          ...       ...                 ...          ...             ...
16995    -124.26     40.58                52.0       2217.0           394.0
16996    -124.27     40.69                36.0       2349.0           528.0
16997    -124.30     41.84                17.0       2677.0           531.0
16998    -124.30     41.80                19.0       2672.0           552.0
16999    -124.35     40.54                52.0       1820.0           300.0

       population  households  median_income  median_house_value
0          1015.0       472.0         1.4936             66900.0
1          1129.0       463.0         1.8200             80100.0
2           333.0       117.0         1.6509             85700.0
3           515.0       226.0         3.1917             73400.0
4           624.0       262.0         1.9250             65500.0
...           ...         ...            ...                 ...
16995       907.0       369.0         2.3571            111400.0
16996      1194.0       465.0         2.5179             79000.0
16997      1244.0       456.0         3.0313            103600.0
16998      1298.0       478.0         1.9797             85800.0
16999       806.0       270.0         3.0147             94600.0

[17000 rows x 9 columns]
```

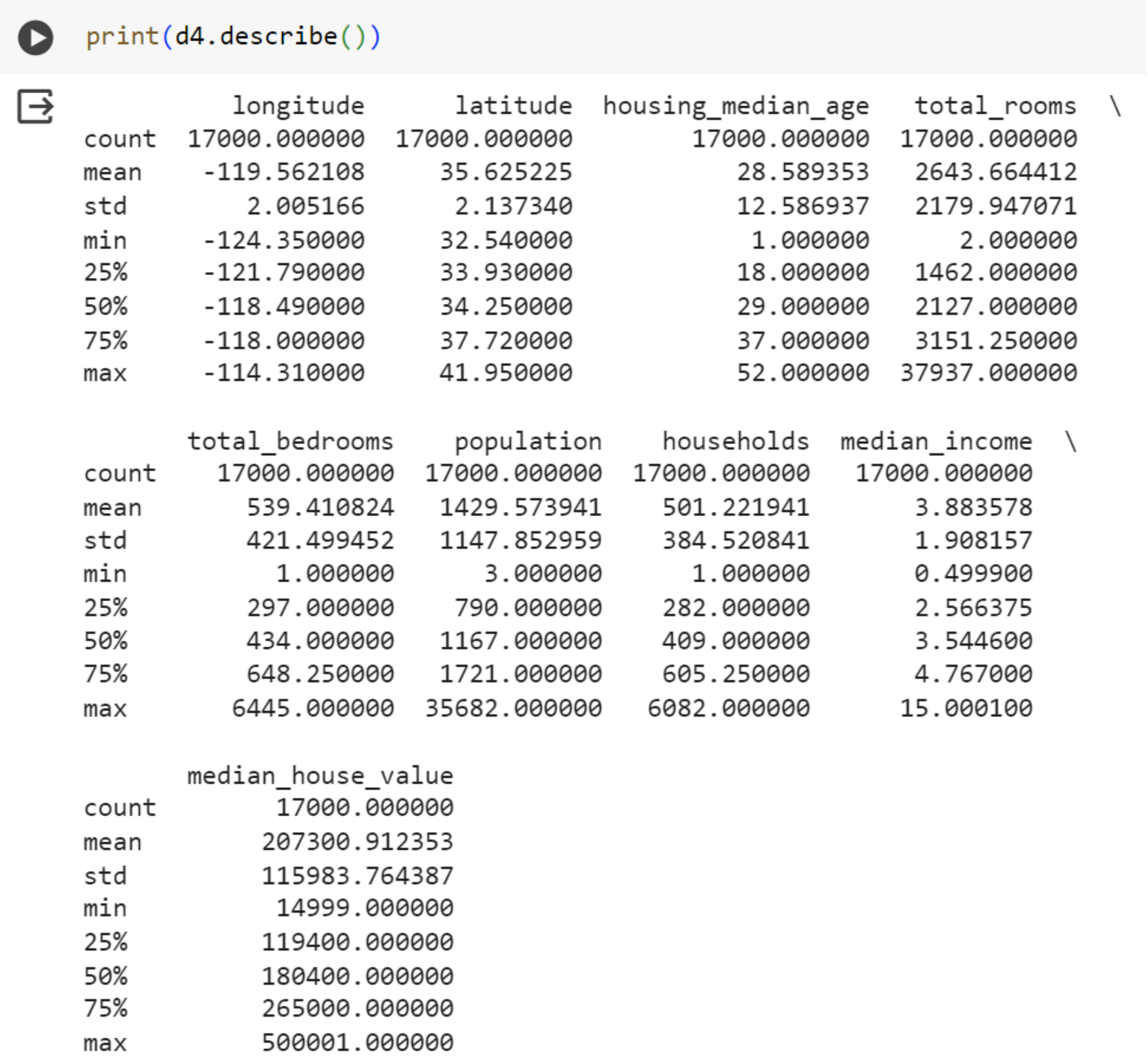


```
[26] d4.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0
1	-114.47	34.40	19.0	7650.0	1901.0	1129.0
2	-114.56	33.69	17.0	720.0	174.0	333.0
3	-114.57	33.64	14.0	1501.0	337.0	515.0
4	-114.57	33.57	20.0	1454.0	326.0	624.0



```
[27] print(d4.count())

     longitude             17000
     latitude              17000
     housing_median_age    17000
     total_rooms           17000
     total_bedrooms        17000
     population            17000
     households            17000
     median_income         17000
     median_house_value    17000
     dtype: int64
```



```
print(d4.describe())

         longitude      latitude  housing_median_age   total_rooms  \
count  17000.000000  17000.000000        17000.000000  17000.000000
mean    -119.562108     35.625225           28.589353   2643.664412
std        2.005166      2.137340           12.586937   2179.947071
min     -124.350000     32.540000            1.000000      2.000000
25%     -121.790000     33.930000           18.000000   1462.000000
50%     -118.490000     34.250000           29.000000   2127.000000
75%     -118.000000     37.720000           37.000000   3151.250000
max     -114.310000     41.950000           52.000000  37937.000000

       total_bedrooms    population    households  median_income  \
count    17000.000000  17000.000000  17000.000000   17000.000000
mean       539.410824   1429.573941    501.221941       3.883578
std        421.499452   1147.852959    384.520841       1.908157
min          1.000000      3.000000      1.000000       0.499900
25%        297.000000    790.000000    282.000000       2.566375
50%        434.000000   1167.000000    409.000000       3.544600
75%        648.250000   1721.000000    605.250000       4.767000
max       6445.000000  35682.000000   6082.000000      15.000100

       median_house_value
count        17000.000000
mean        207300.912353
std         115983.764387
min          14999.000000
25%         119400.000000
50%         180400.000000
75%         265000.000000
max         500001.000000
```

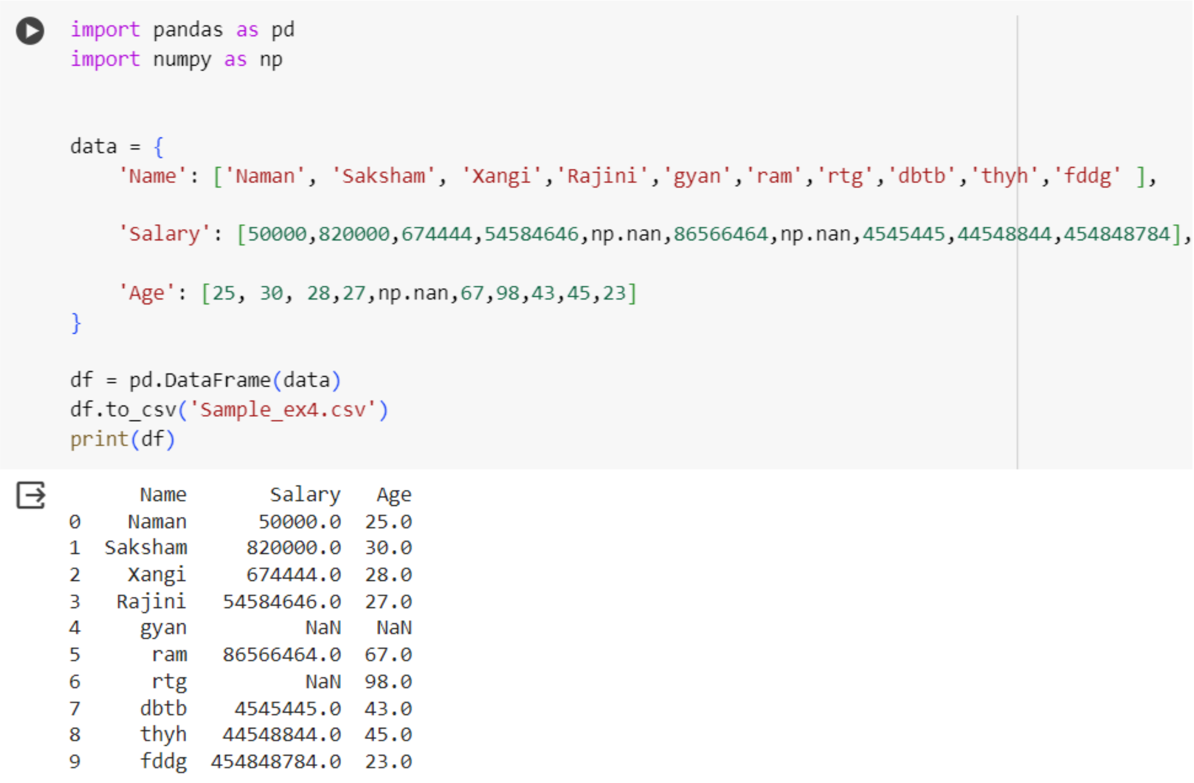
Program-5

Objective: To perform data preprocessing operation:

1) Handling Missing Data
2) Min- Max Normalisation

Handling Missing data

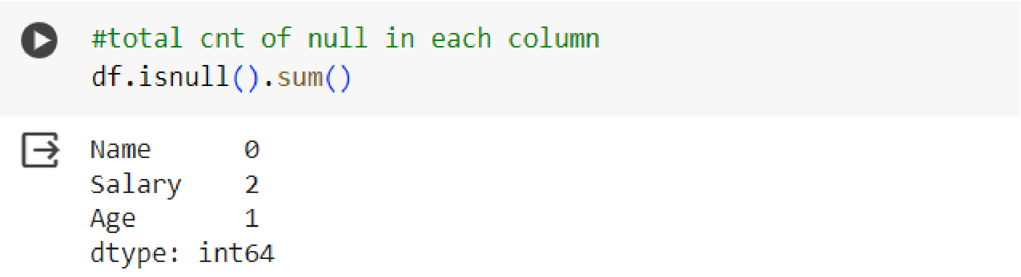
1. Creating DataFrames



```
import pandas as pd
import numpy as np


data = {
    'Name': ['Naman', 'Saksham', 'Xangi','Rajini','gyan','ram','rtg','dbtb','thyh','fddg' ],

    'Salary': [50000,820000,674444,54584646,np.nan,86566464,np.nan,4545445,44548844,454848784],

    'Age': [25, 30, 28,27,np.nan,67,98,43,45,23]
}

df = pd.DataFrame(data)
df.to_csv('Sample_ex4.csv')
print(df)
```

```
      Name       Salary   Age
0    Naman      50000.0  25.0
1  Saksham     820000.0  30.0
2    Xangi     674444.0  28.0
3   Rajini   54584646.0  27.0
4     gyan          NaN   NaN
5      ram   86566464.0  67.0
6      rtg          NaN  98.0
7     dbtb    4545445.0  43.0
8     thyh   44548844.0  45.0
9     fddg  454848784.0  23.0
```

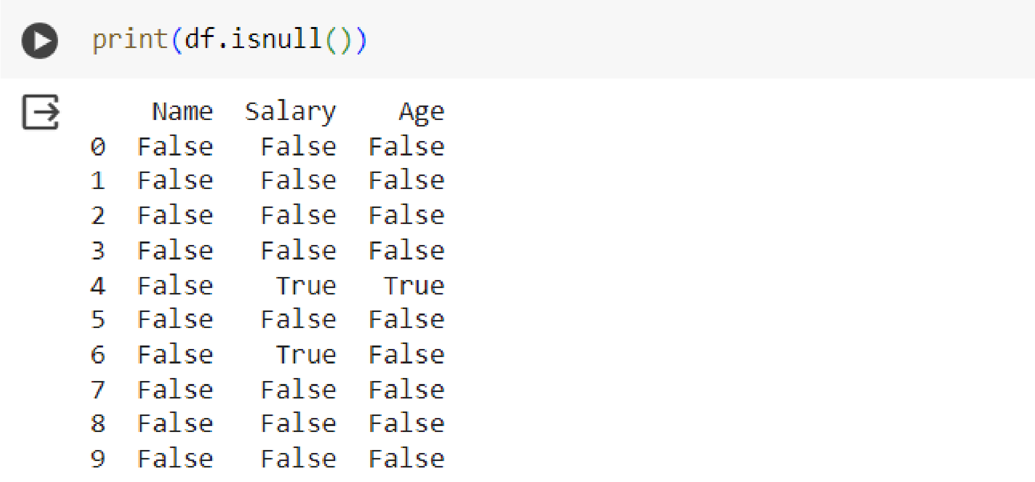
2. Total count of NULL in each column



```
#total cnt of null in each column
df.isnull().sum()
```

```
Name      0
Salary    2
Age       1
dtype: int64
```

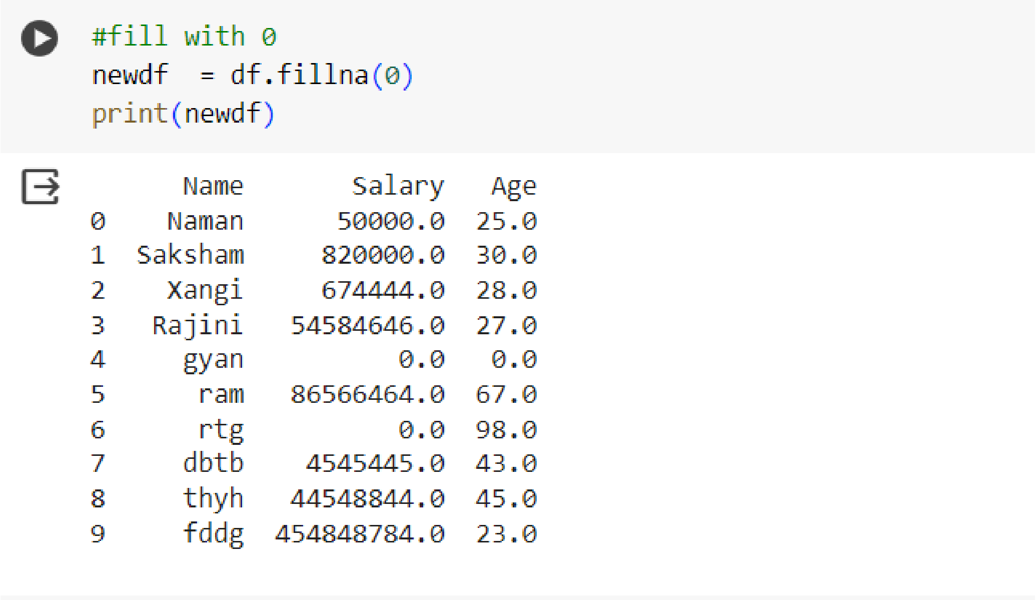
3. Printing whether data value is NULL or Not



```
print(df.isnull())
```

```
    Name  Salary    Age
0  False   False  False
1  False   False  False
2  False   False  False
3  False   False  False
4  False    True   True
5  False   False  False
6  False    True  False
7  False   False  False
8  False   False  False
9  False   False  False
```

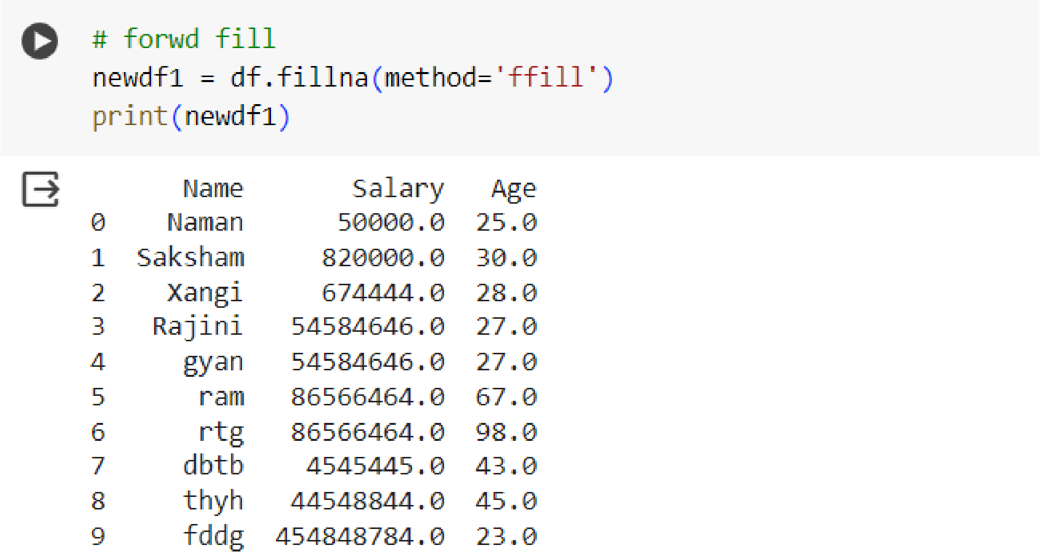
4. Fill Missing values with 0



```
#fill with 0
newdf  = df.fillna(0)
print(newdf)
```

```
      Name       Salary   Age
0    Naman      50000.0  25.0
1  Saksham     820000.0  30.0
2    Xangi     674444.0  28.0
3   Rajini   54584646.0  27.0
4     gyan          0.0   0.0
5      ram   86566464.0  67.0
6      rtg          0.0  98.0
7     dbtb    4545445.0  43.0
8     thyh   44548844.0  45.0
9     fddg  454848784.0  23.0
```

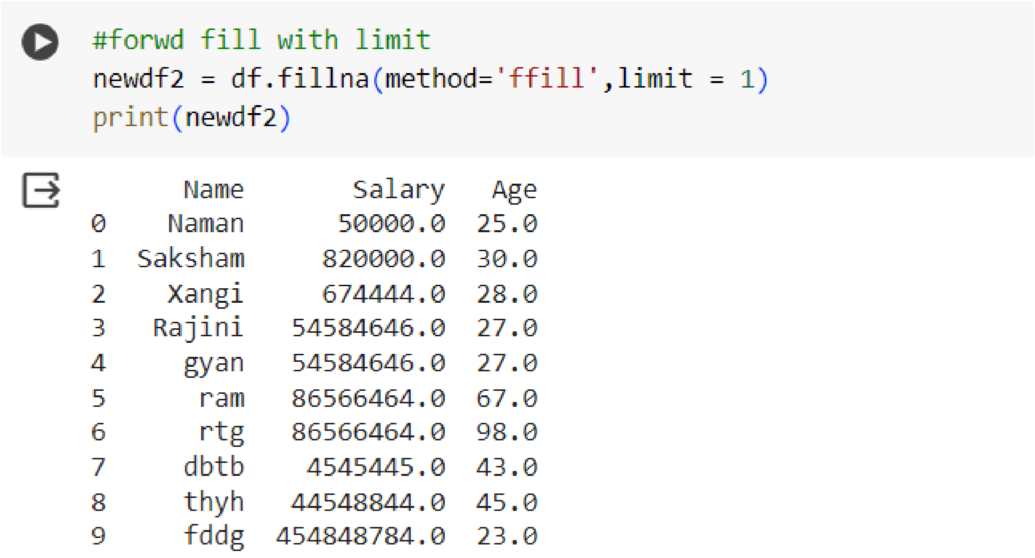
5. Forword Filling missing values



```
# forwd fill
newdf1 = df.fillna(method='ffill')
print(newdf1)
```

```
      Name       Salary   Age
0    Naman      50000.0  25.0
1  Saksham     820000.0  30.0
2    Xangi     674444.0  28.0
3   Rajini   54584646.0  27.0
4     gyan   54584646.0  27.0
5      ram   86566464.0  67.0
6      rtg   86566464.0  98.0
7     dbtb    4545445.0  43.0
8     thyh   44548844.0  45.0
9     fddg  454848784.0  23.0
```

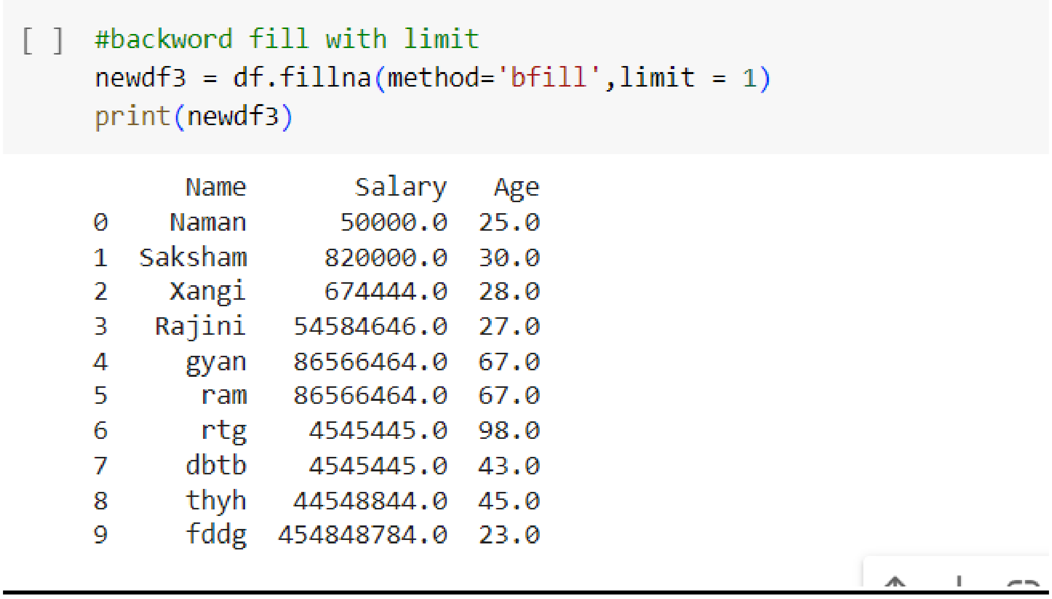
6. Forword Filling missing values with limit



```
#forwd fill with limit
newdf2 = df.fillna(method='ffill',limit = 1)
print(newdf2)
```

```
      Name       Salary   Age
0    Naman      50000.0  25.0
1  Saksham     820000.0  30.0
2    Xangi     674444.0  28.0
3   Rajini   54584646.0  27.0
4     gyan   54584646.0  27.0
5      ram   86566464.0  67.0
6      rtg   86566464.0  98.0
7     dbtb    4545445.0  43.0
8     thyh   44548844.0  45.0
9     fddg  454848784.0  23.0
```

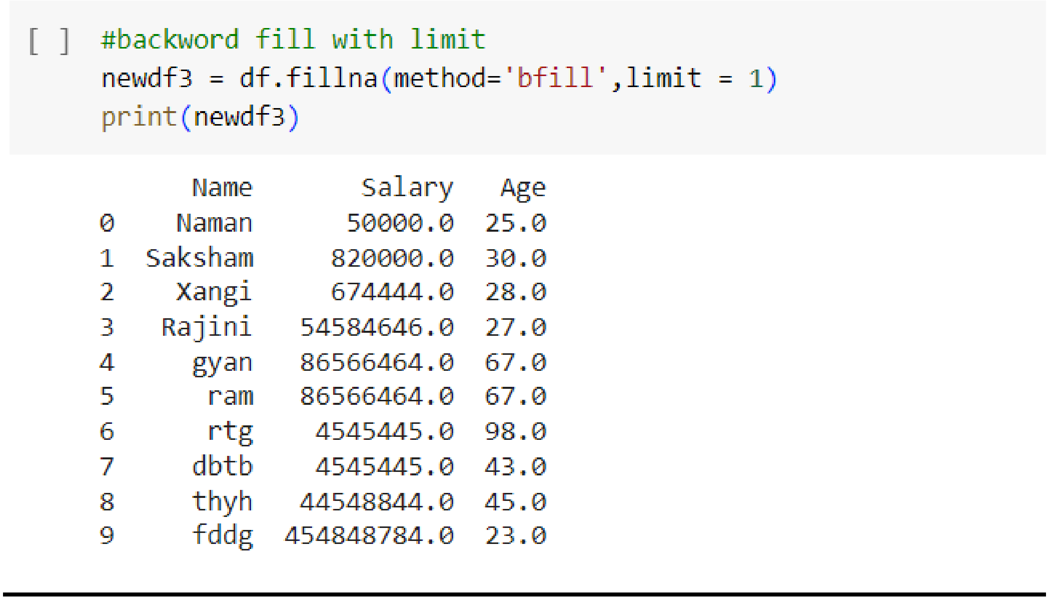
7. Backword Filling missing values



```
[ ]  #backword fill with limit
     newdf3 = df.fillna(method='bfill',limit = 1)
     print(newdf3)
```

```
         Name       Salary   Age
0       Naman      50000.0  25.0
1     Saksham     820000.0  30.0
2       Xangi     674444.0  28.0
3      Rajini   54584646.0  27.0
4        gyan   86566464.0  67.0
5         ram   86566464.0  67.0
6         rtg    4545445.0  98.0
7        dbtb    4545445.0  43.0
8        thyh   44548844.0  45.0
9        fddg  454848784.0  23.0
```

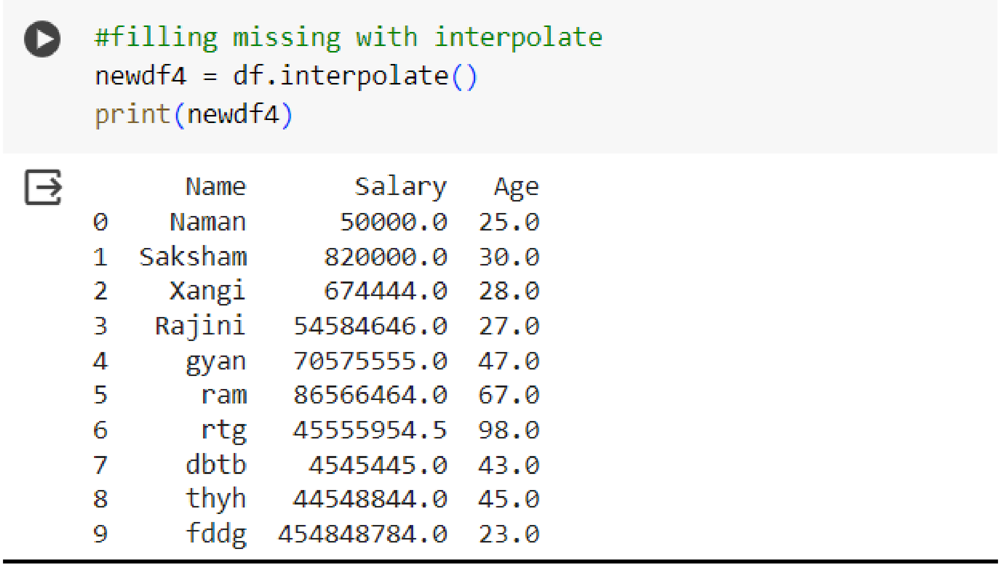
8. Backword Filling missing values with limit



```
[ ]  #backword fill with limit
     newdf3 = df.fillna(method='bfill',limit = 1)
     print(newdf3)
```

```
         Name       Salary   Age
0       Naman      50000.0  25.0
1     Saksham     820000.0  30.0
2       Xangi     674444.0  28.0
3      Rajini   54584646.0  27.0
4        gyan   86566464.0  67.0
5         ram   86566464.0  67.0
6         rtg    4545445.0  98.0
7        dbtb    4545445.0  43.0
8        thyh   44548844.0  45.0
9        fddg  454848784.0  23.0
```

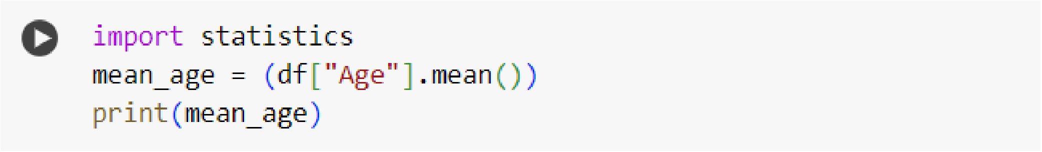
9. Filling Missing values with Interpolate



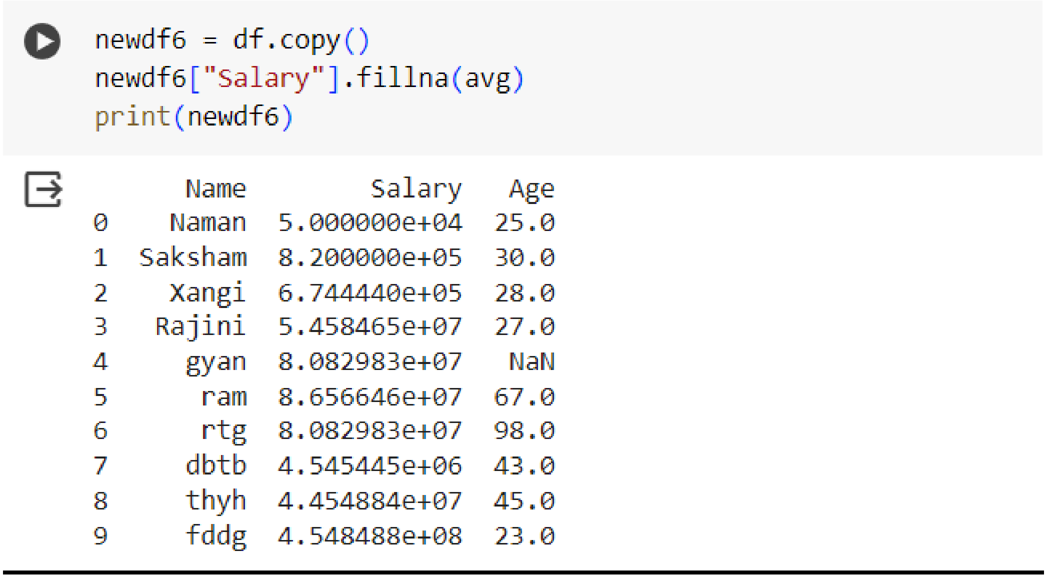
```
#filling missing with interpolate
newdf4 = df.interpolate()
print(newdf4)
```

```
      Name       Salary   Age
0    Naman      50000.0  25.0
1  Saksham     820000.0  30.0
2    Xangi     674444.0  28.0
3   Rajini   54584646.0  27.0
4     gyan   70575555.0  47.0
5      ram   86566464.0  67.0
6      rtg   45555954.5  98.0
7     dbtb    4545445.0  43.0
8     thyh   44548844.0  45.0
9     fddg  454848784.0  23.0
```

10. Filling Missing values with mean



```
import statistics
mean_age = (df["Age"].mean())
print(mean_age)
```

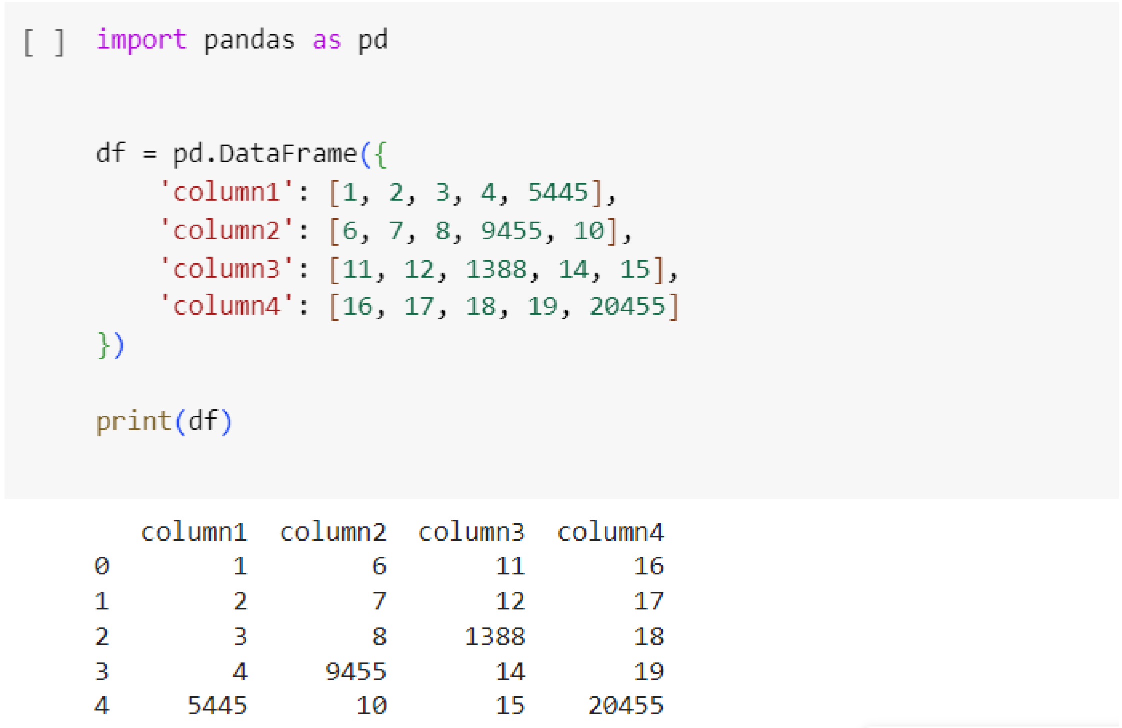


```
newdf6 = df.copy()
newdf6["Salary"].fillna(avg)
print(newdf6)
```

```
      Name        Salary   Age
0    Naman  5.000000e+04  25.0
1  Saksham  8.200000e+05  30.0
2    Xangi  6.744440e+05  28.0
3   Rajini  5.458465e+07  27.0
4     gyan  8.082983e+07   NaN
5      ram  8.656646e+07  67.0
6      rtg  8.082983e+07  98.0
7     dbtb  4.545445e+06  43.0
8     thyh  4.454884e+07  45.0
9     fddg  4.548488e+08  23.0
```

MIN- MAX Normalisation

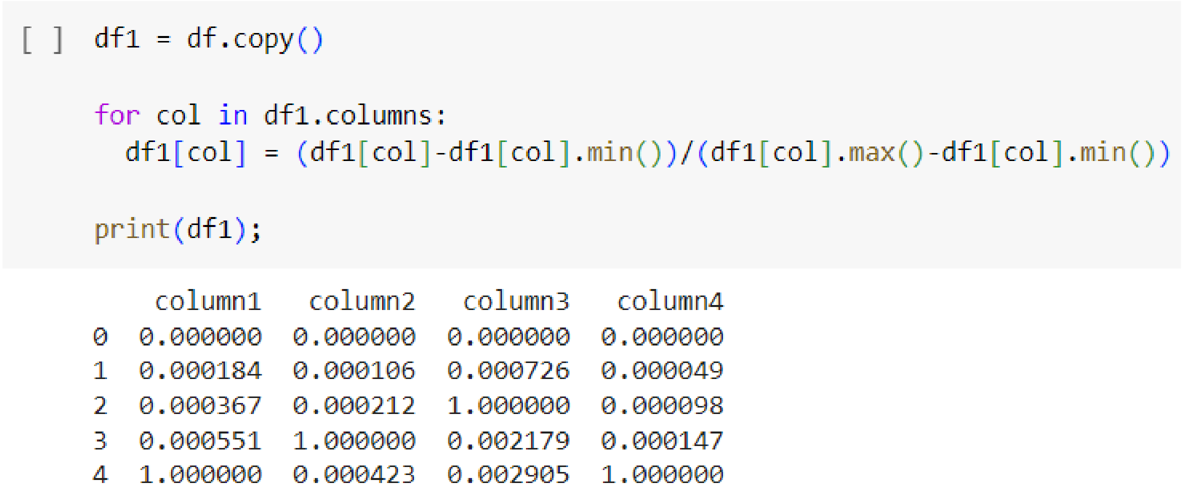
1. Creating dataframes



```
[ ]  import pandas as pd


     df = pd.DataFrame({
         'column1': [1, 2, 3, 4, 5445],
         'column2': [6, 7, 8, 9455, 10],
         'column3': [11, 12, 1388, 14, 15],
         'column4': [16, 17, 18, 19, 20455]
     })

     print(df)
```

```
   column1  column2  column3  column4
0        1        6       11       16
1        2        7       12       17
2        3        8     1388       18
3        4     9455       14       19
4     5445       10       15    20455
```

2. MIN- MAX Normalisation



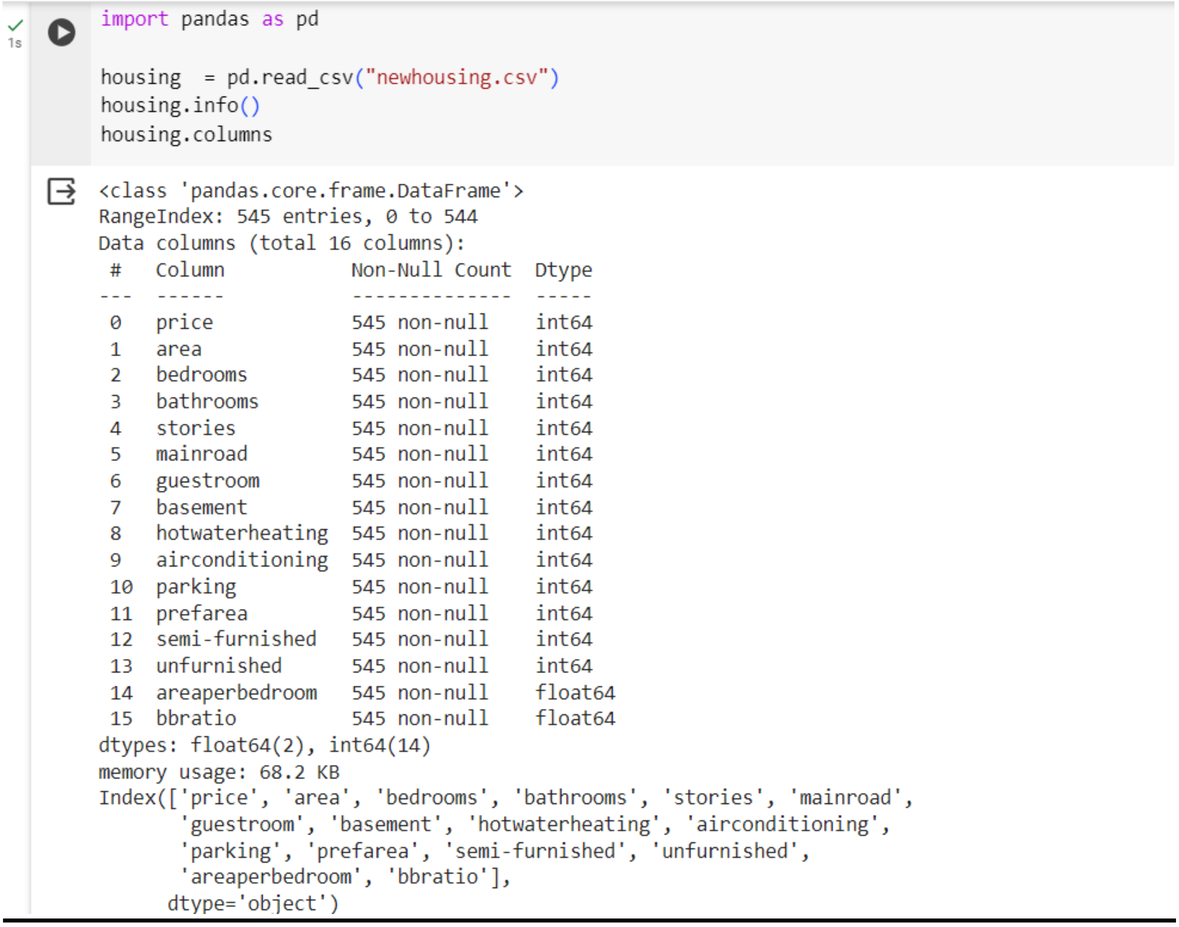
```
[ ]  df1 = df.copy()

     for col in df1.columns:
       df1[col] = (df1[col]-df1[col].min())/(df1[col].max()-df1[col].min())

     print(df1);
```

```
    column1   column2   column3   column4
0  0.000000  0.000000  0.000000  0.000000
1  0.000184  0.000106  0.000726  0.000049
2  0.000367  0.000212  1.000000  0.000098
3  0.000551  1.000000  0.002179  0.000147
4  1.000000  0.000423  0.002905  1.000000
```

Program-6

Objective: To perform dimensionality Reduction operation using PCA for Housing Data set.

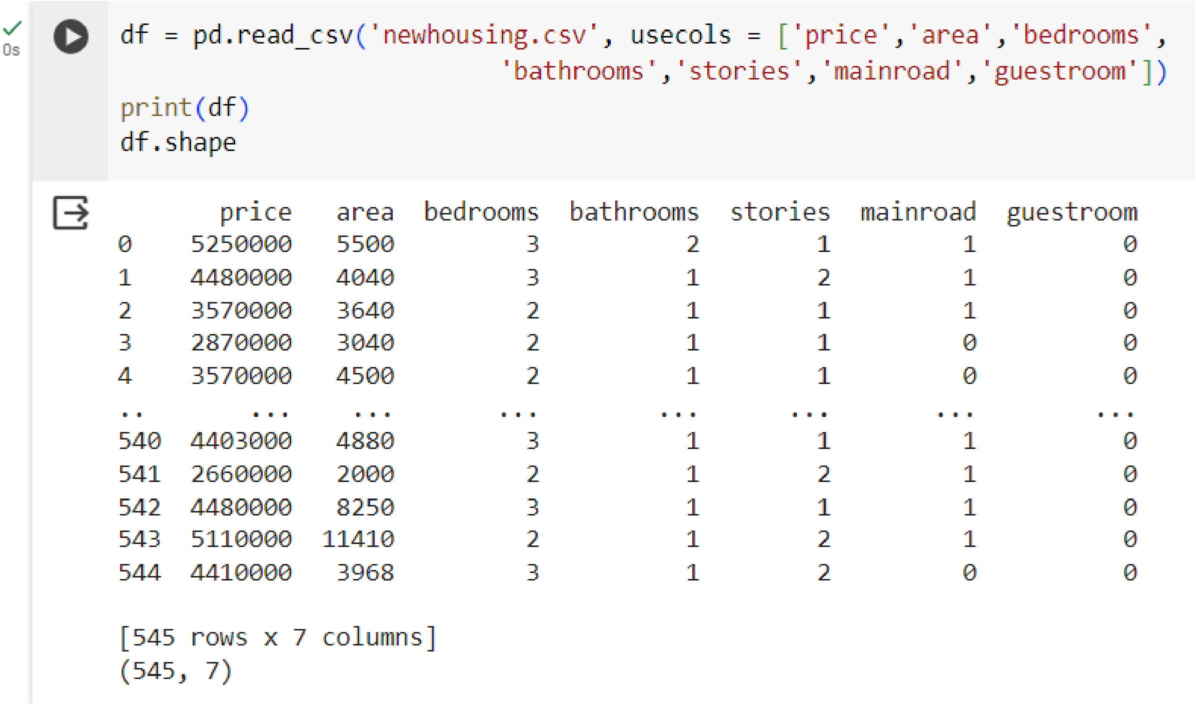
1. Importing housing.csv file



```
import pandas as pd

housing  = pd.read_csv("newhousing.csv")
housing.info()
housing.columns
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   price            545 non-null    int64  
 1   area             545 non-null    int64  
 2   bedrooms         545 non-null    int64  
 3   bathrooms        545 non-null    int64  
 4   stories          545 non-null    int64  
 5   mainroad         545 non-null    int64  
 6   guestroom        545 non-null    int64  
 7   basement         545 non-null    int64  
 8   hotwaterheating  545 non-null    int64  
 9   airconditioning  545 non-null    int64  
 10  parking          545 non-null    int64  
 11  prefarea         545 non-null    int64  
 12  semi-furnished   545 non-null    int64  
 13  unfurnished      545 non-null    int64  
 14  areaperbedroom   545 non-null    float64
 15  bbratio          545 non-null    float64
dtypes: float64(2), int64(14)
memory usage: 68.2 KB
Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
       'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
       'parking', 'prefarea', 'semi-furnished', 'unfurnished',
       'areaperbedroom', 'bbratio'],
      dtype='object')
```

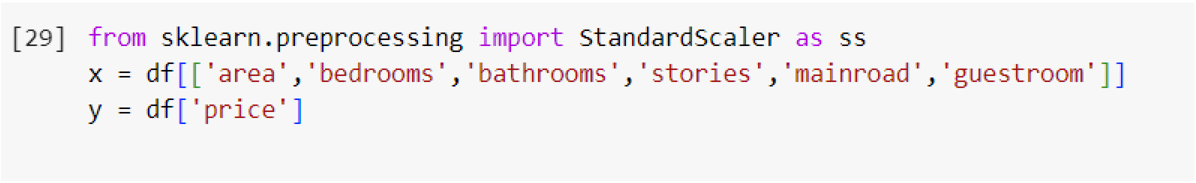
2. Resizing the DataFrame



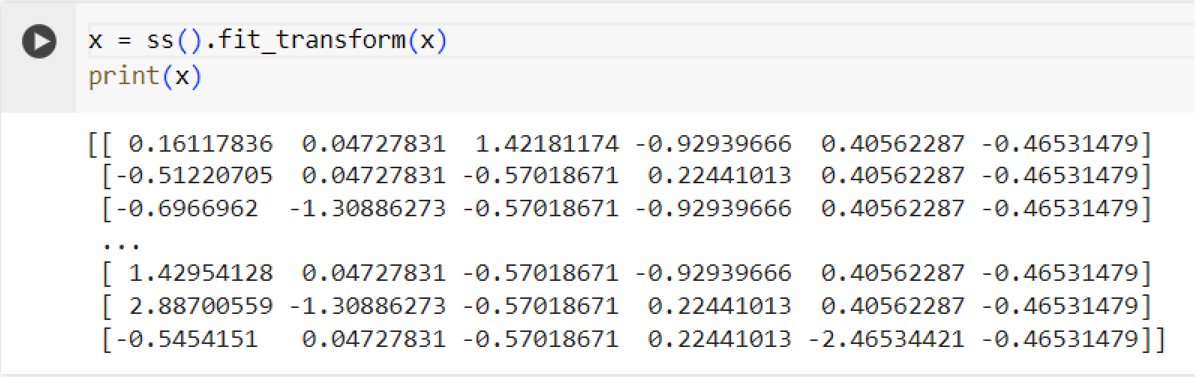
```
df = pd.read_csv('newhousing.csv', usecols = ['price','area','bedrooms',
                          'bathrooms','stories','mainroad','guestroom'])
print(df)
df.shape
```

```
       price   area  bedrooms  bathrooms  stories  mainroad  guestroom
0    5250000   5500         3          2        1         1          0
1    4480000   4040         3          1        2         1          0
2    3570000   3640         2          1        1         1          0
3    2870000   3040         2          1        1         0          0
4    3570000   4500         2          1        1         0          0
..       ...    ...       ...        ...      ...       ...        ...
540  4403000   4880         3          1        1         1          0
541  2660000   2000         2          1        2         1          0
542  4480000   8250         3          1        1         1          0
543  5110000  11410         2          1        2         1          0
544  4410000   3968         3          1        2         0          0

[545 rows x 7 columns]
(545, 7)
```

3. Perform Scaler Transformation on Data



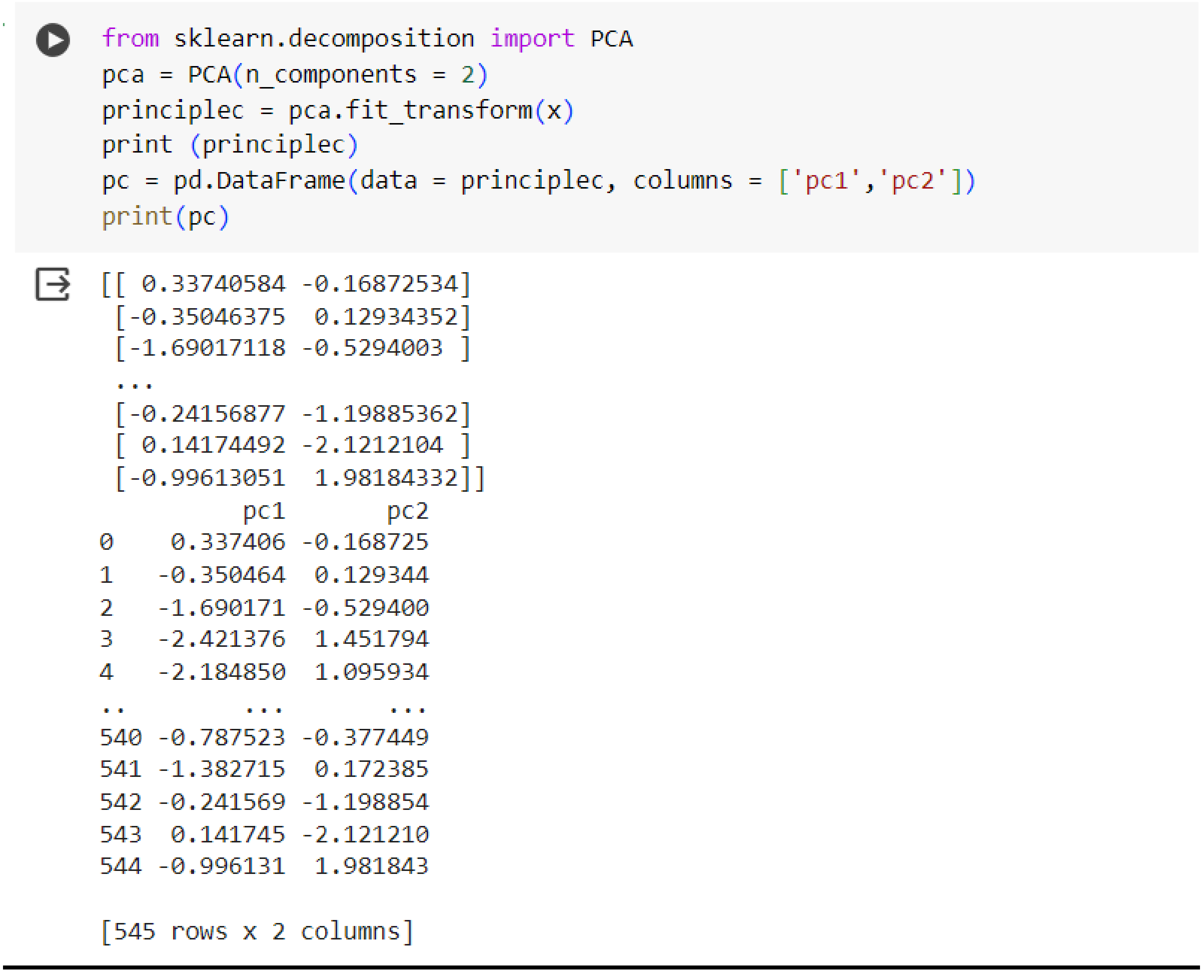
```
[29] from sklearn.preprocessing import StandardScaler as ss
     x = df[['area','bedrooms','bathrooms','stories','mainroad','guestroom']]
     y = df['price']
```



```
x = ss().fit_transform(x)
print(x)
```

```
[[ 0.16117836  0.04727831  1.42181174 -0.92939666  0.40562287 -0.46531479]
 [-0.51220705  0.04727831 -0.57018671  0.22441013  0.40562287 -0.46531479]
 [-0.6966962  -1.30886273 -0.57018671 -0.92939666  0.40562287 -0.46531479]
 ...
 [ 1.42954128  0.04727831 -0.57018671 -0.92939666  0.40562287 -0.46531479]
 [ 2.88700559 -1.30886273 -0.57018671  0.22441013  0.40562287 -0.46531479]
 [-0.5454151   0.04727831 -0.57018671  0.22441013 -2.46534421 -0.46531479]]
```

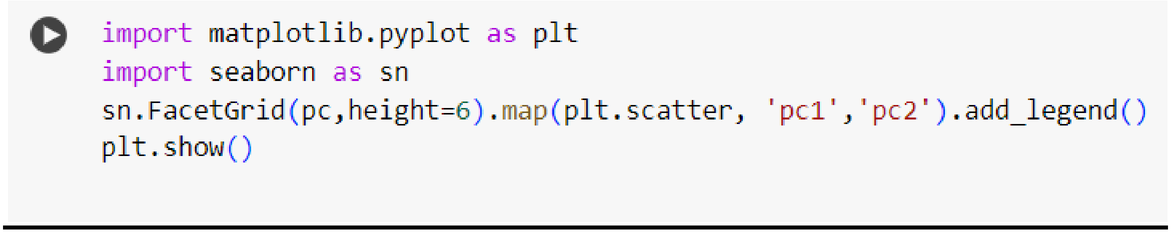
4. Compute Principle Component Analysis



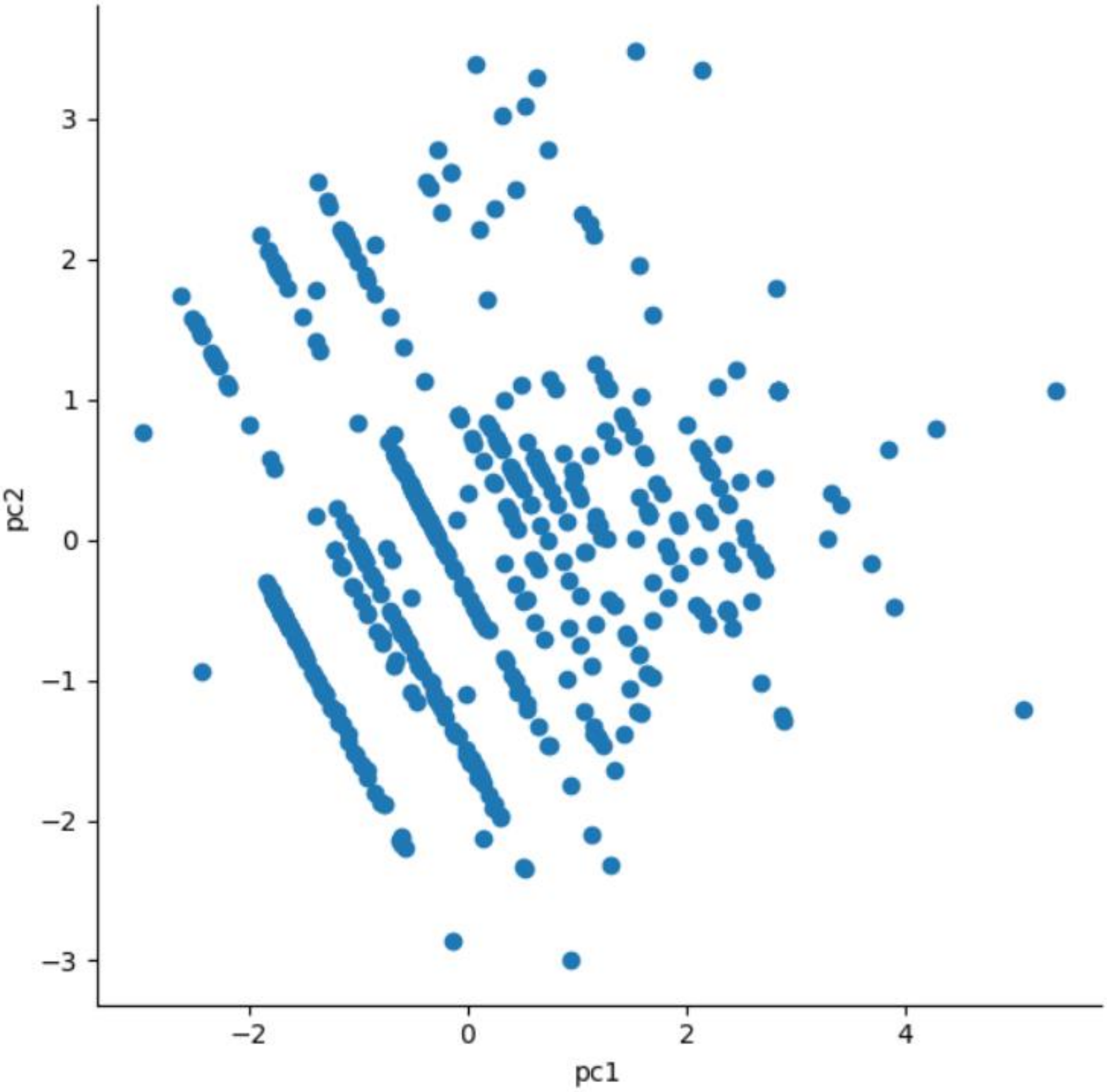
```
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
principlec = pca.fit_transform(x)
print (principlec)
pc = pd.DataFrame(data = principlec, columns = ['pc1','pc2'])
print(pc)
```

```
[[ 0.33740584 -0.16872534]
 [-0.35046375  0.12934352]
 [-1.69017118 -0.5294003 ]
 ...
 [-0.24156877 -1.19885362]
 [ 0.14174492 -2.1212104 ]
 [-0.99613051  1.98184332]]
          pc1       pc2
0    0.337406 -0.168725
1   -0.350464  0.129344
2   -1.690171 -0.529400
3   -2.421376  1.451794
4   -2.184850  1.095934
..        ...       ...
540 -0.787523 -0.377449
541 -1.382715  0.172385
542 -0.241569 -1.198854
543  0.141745 -2.121210
544 -0.996131  1.981843

[545 rows x 2 columns]
```

5. Plotting graph for PCA



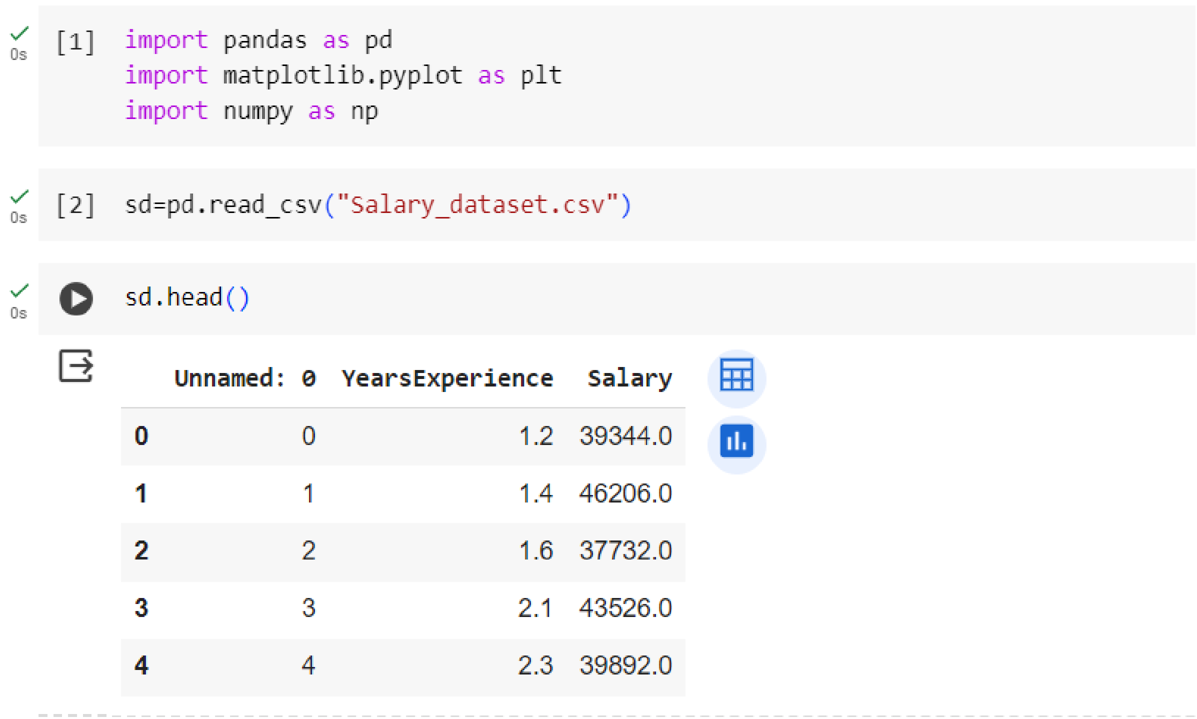
```
import matplotlib.pyplot as plt
import seaborn as sn
sn.FacetGrid(pc,height=6).map(plt.scatter, 'pc1','pc2').add_legend()
plt.show()
```



Program-7

Objective: To perform Simple Linear Regression using Python.

1. IMPORT SALARY DATASET CSV FILE AND PRINT 5 COLUMNS:



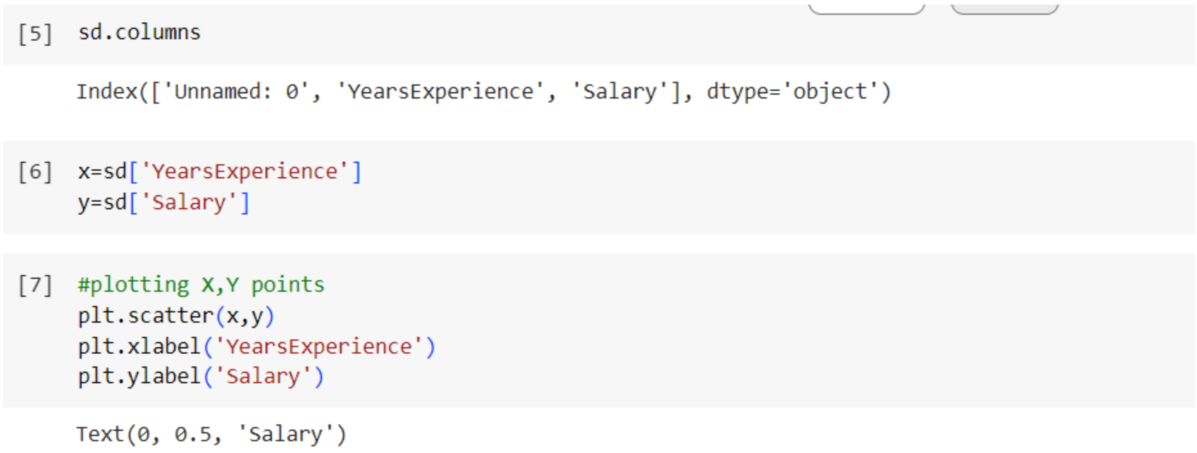
```
[1] import pandas as pd
    import matplotlib.pyplot as plt
    import numpy as np
```

```
[2] sd=pd.read_csv("Salary_dataset.csv")
```

```
sd.head()
```

	Unnamed: 0	YearsExperience	Salary
0	0	1.2	39344.0
1	1	1.4	46206.0
2	2	1.6	37732.0
3	3	2.1	43526.0
4	4	2.3	39892.0

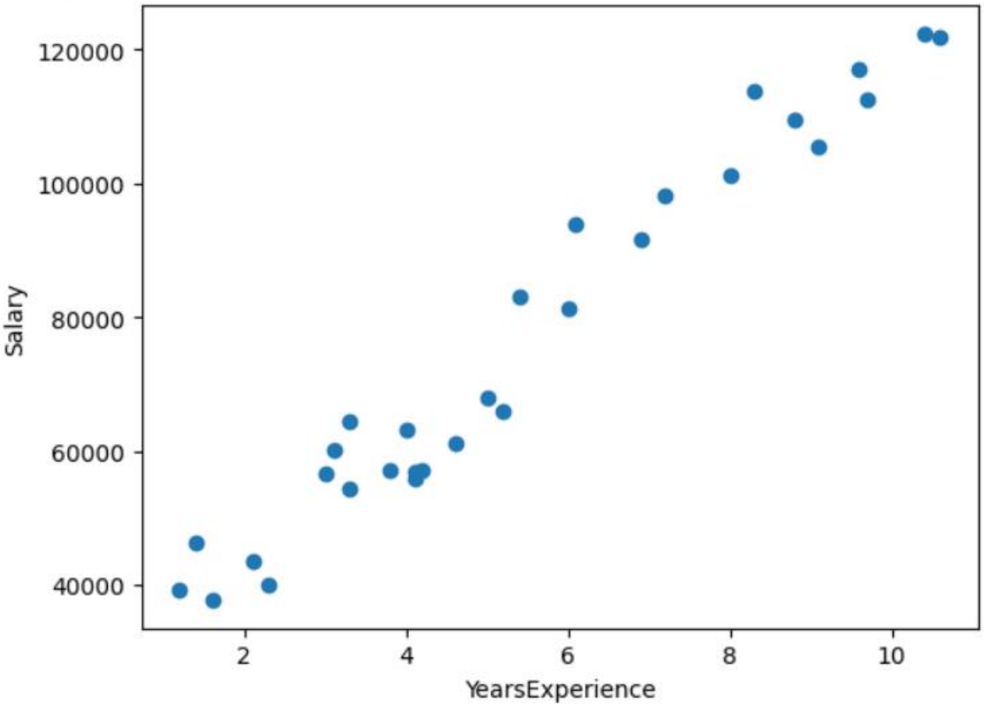
2. PLOTTING GRAPH BETWEEN TWO VARIABLES(YearsExperience and Salary)



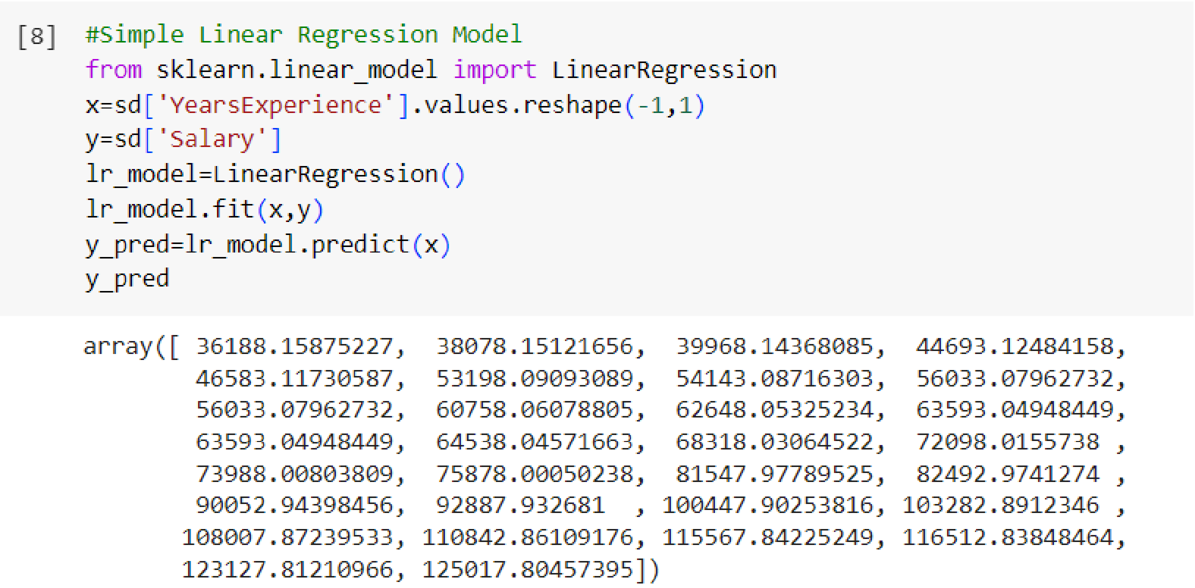
```
[5] sd.columns

    Index(['Unnamed: 0', 'YearsExperience', 'Salary'], dtype='object')
```

```
[6] x=sd['YearsExperience']
    y=sd['Salary']
```

```
[7] #plotting X,Y points
    plt.scatter(x,y)
    plt.xlabel('YearsExperience')
    plt.ylabel('Salary')

    Text(0, 0.5, 'Salary')
Text(0, 0.5, 'Salary')
```



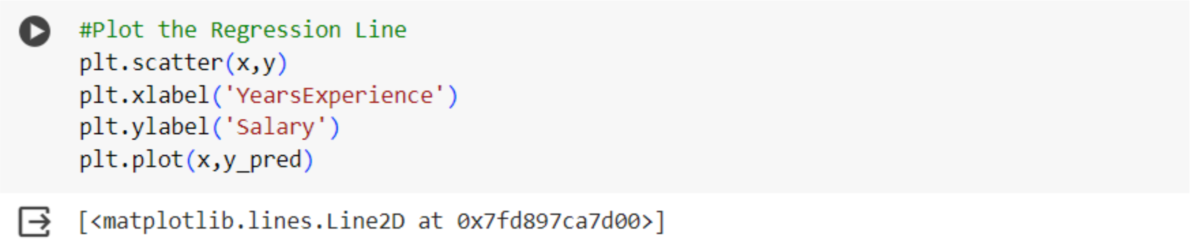
3. BUILDING SIMPLE LINEAR REGRESSION MODEL



```
[8]  #Simple Linear Regression Model
     from sklearn.linear_model import LinearRegression
     x=sd['YearsExperience'].values.reshape(-1,1)
     y=sd['Salary']
     lr_model=LinearRegression()
     lr_model.fit(x,y)
     y_pred=lr_model.predict(x)
     y_pred
```

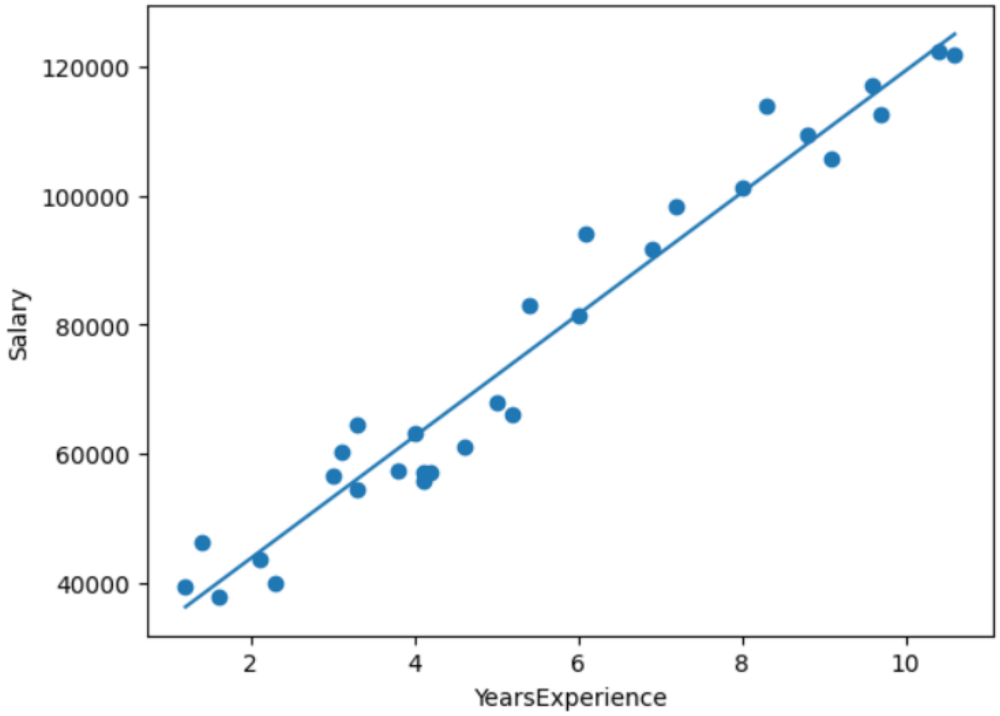
```
array([ 36188.15875227,  38078.15121656,  39968.14368085,  44693.12484158,
        46583.11730587,  53198.09093089,  54143.08716303,  56033.07962732,
        56033.07962732,  60758.06078805,  62648.05325234,  63593.04948449,
        63593.04948449,  64538.04571663,  68318.03064522,  72098.0155738 ,
        73988.00803809,  75878.00050238,  81547.97789525,  82492.9741274 ,
        90052.94398456,  92887.932681  , 100447.90253816, 103282.8912346 ,
       108007.87239533, 110842.86109176, 115567.84225249, 116512.83848464,
       123127.81210966, 125017.80457395])
```

4. PLOTTING GRAPH FOR THE REGRESSION LINE

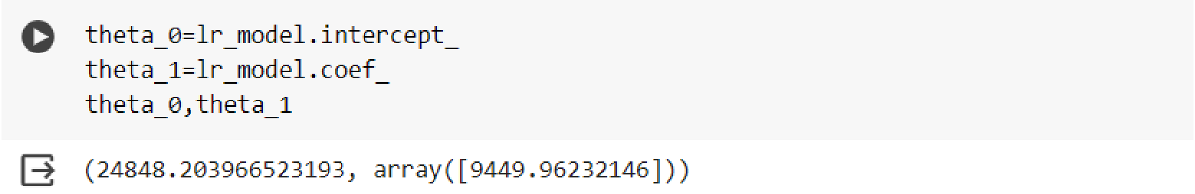


```
#Plot the Regression Line
plt.scatter(x,y)
plt.xlabel('YearsExperience')
plt.ylabel('Salary')
plt.plot(x,y_pred)
```

```
[<matplotlib.lines.Line2D at 0x7fd897ca7d00>]
```



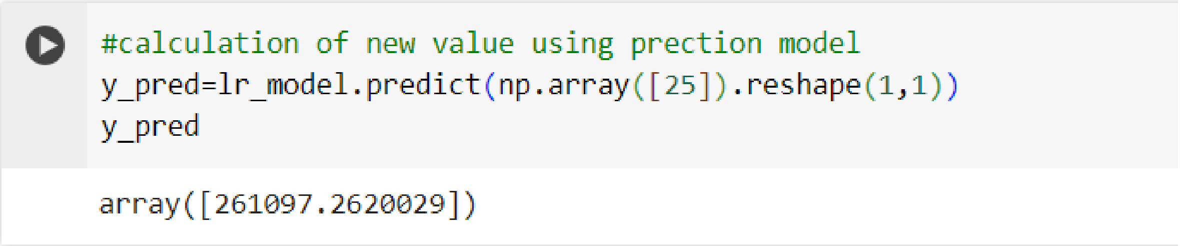
5. COMPUTE INTERCEPT AND COEFFICIENT OF REGRESSION LINE



```
theta_0=lr_model.intercept_
theta_1=lr_model.coef_
theta_0,theta_1
```

```
(24848.203966523193, array([9449.96232146]))
```

6. PREDICT NEW VALUE USING REGRESSION MODEL



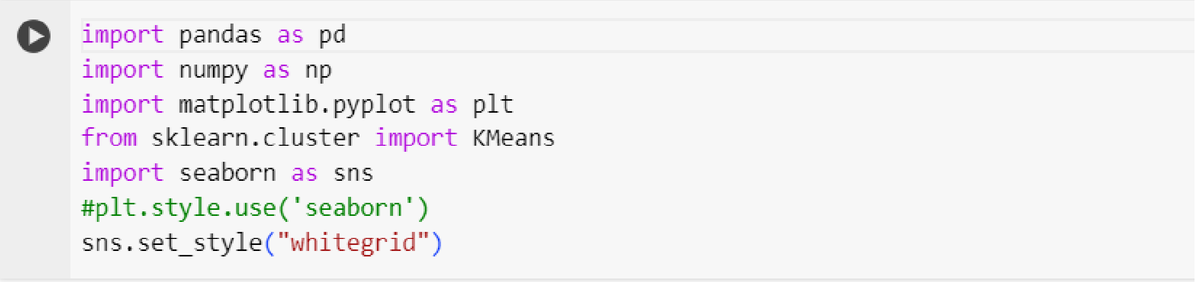
```
#calculation of new value using prection model
y_pred=lr_model.predict(np.array([25]).reshape(1,1))
y_pred
```

```
array([261097.2620029])
```

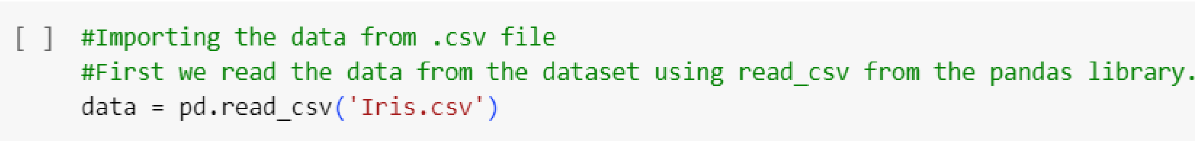
Program-8

Objective: To perform Kmeans clustering on Iris Dataset.

1. IMPORT REQUIRED LIBRARIES AND IRIS DATASET:

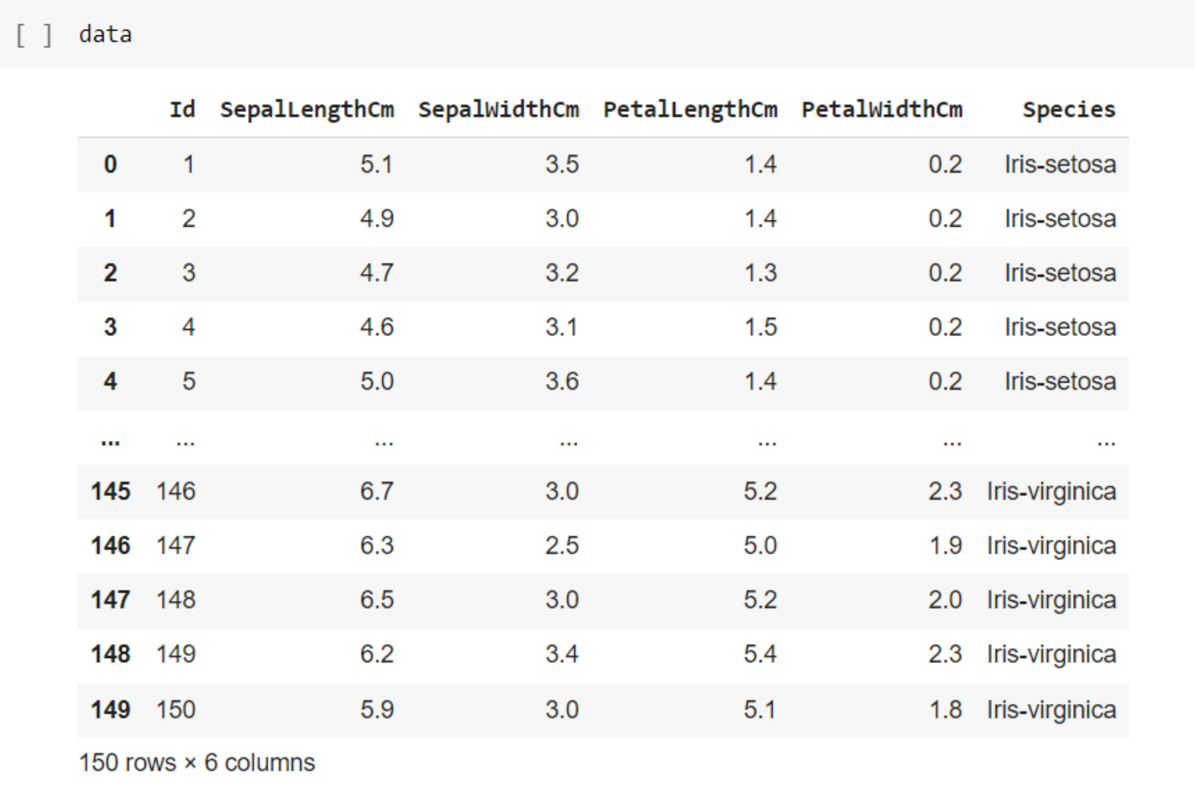


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import seaborn as sns
#plt.style.use('seaborn')
sns.set_style("whitegrid")
```



```
[ ]  #Importing the data from .csv file
     #First we read the data from the dataset using read_csv from the pandas library.
     data = pd.read_csv('Iris.csv')
```

2. PRINTING THE DATASET

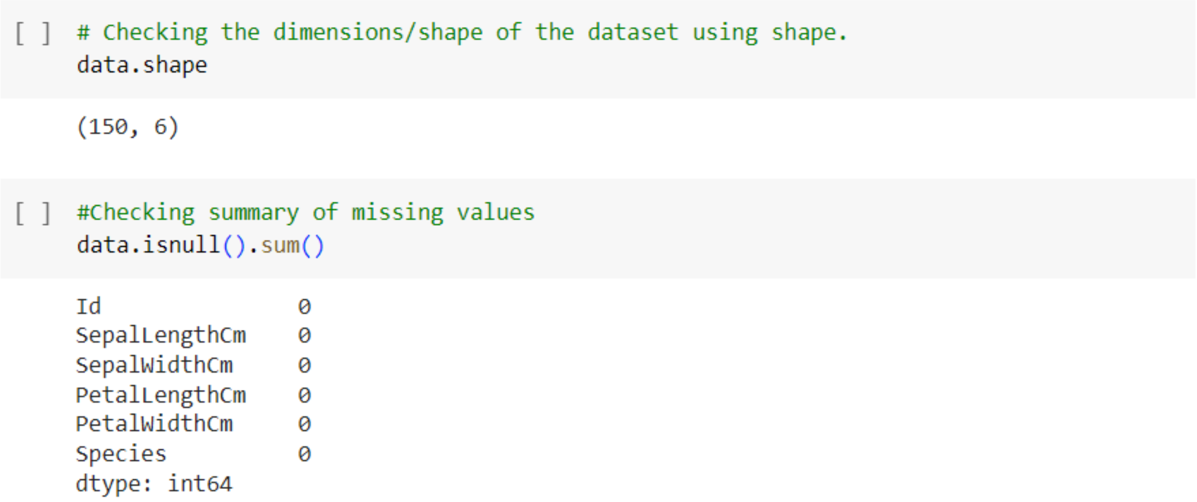


```
[ ]  data
```

	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
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

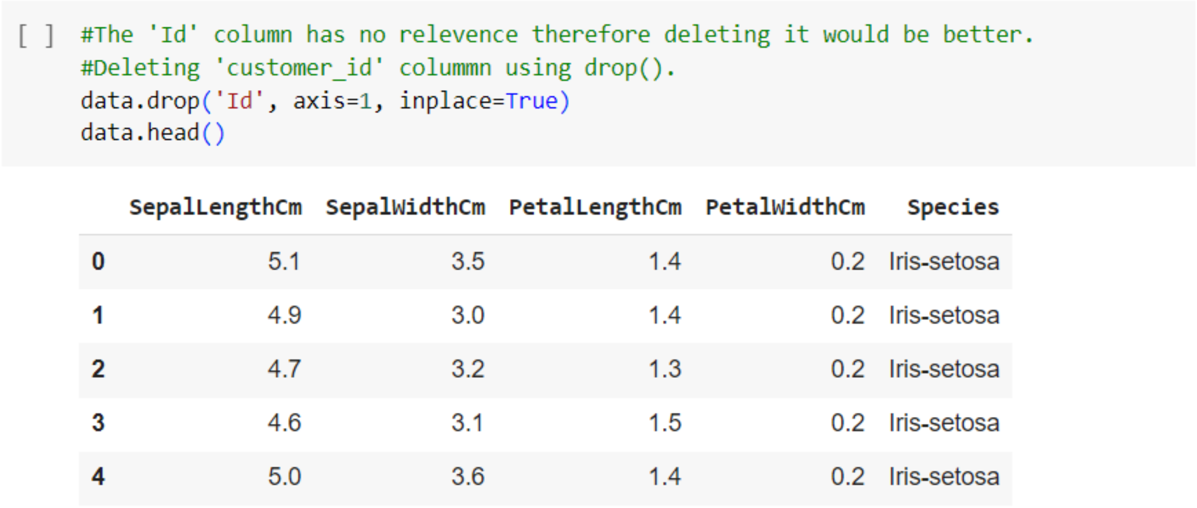
3. CHECKING THE DIMENSIONS AND MISSING VALUES IN DATASET



```
[ ]  # Checking the dimensions/shape of the dataset using shape.
     data.shape

     (150, 6)
```

```
[ ]  #Checking summary of missing values
     data.isnull().sum()

     Id               0
     SepalLengthCm    0
     SepalWidthCm     0
     PetalLengthCm    0
     PetalWidthCm     0
     Species          0
     dtype: int64
```

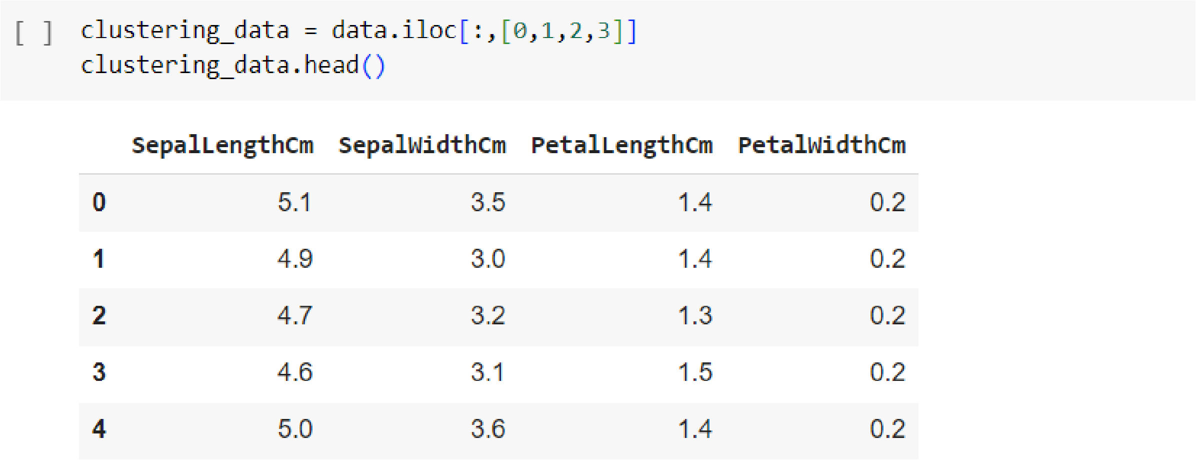
4. DELETING COLUMN ‘ID’



```
[ ]  #The 'Id' column has no relevence therefore deleting it would be better.
     #Deleting 'customer_id' colummn using drop().
     data.drop('Id', axis=1, inplace=True)
     data.head()
```

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

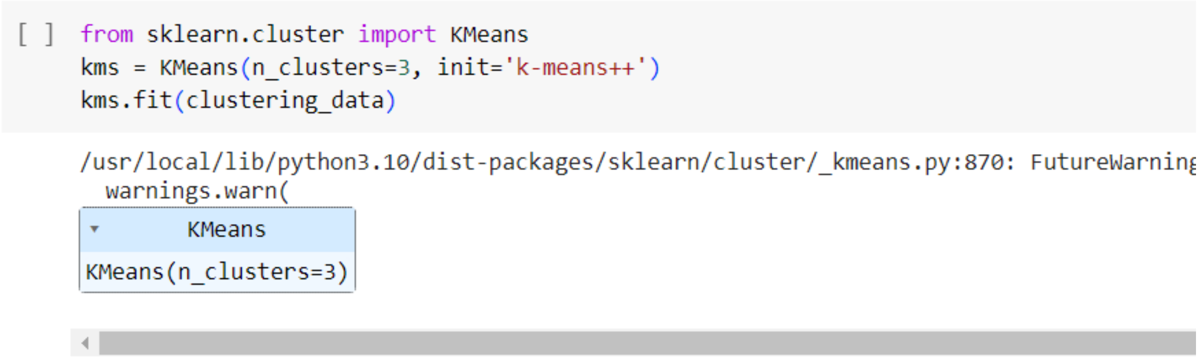
5. MAKING CLUSTERS OF DATA



```
[ ]  clustering_data = data.iloc[:,[0,1,2,3]]
     clustering_data.head()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
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

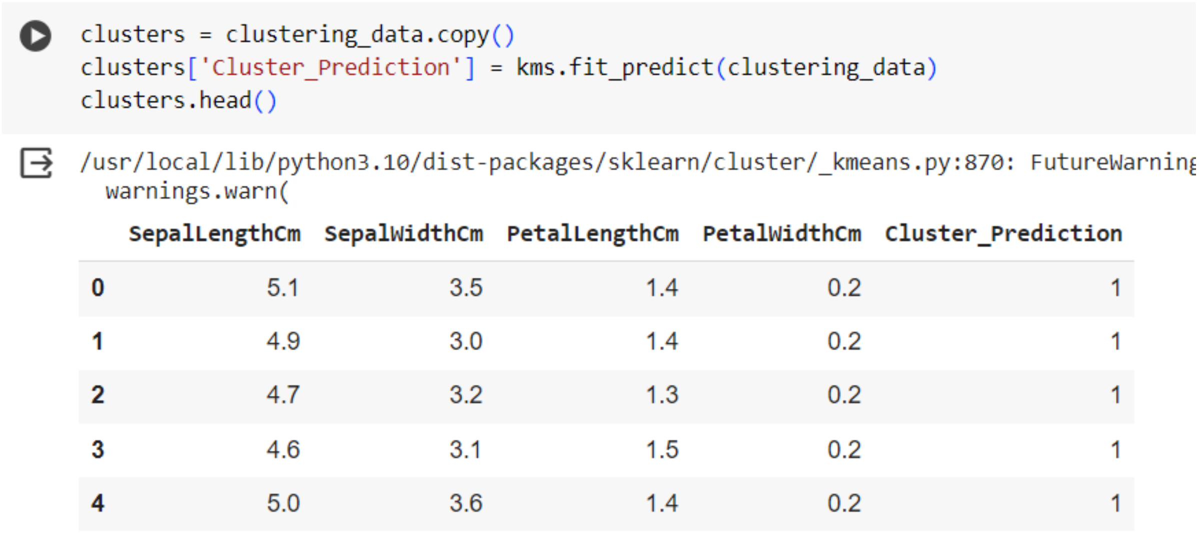
6. FITTING DATASETS IN CLUSTERS AND PREDICTING CLUSTERS OF DATA



```
[ ]  from sklearn.cluster import KMeans
     kms = KMeans(n_clusters=3, init='k-means++')
     kms.fit(clustering_data)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
```

```
KMeans(n_clusters=3)
```

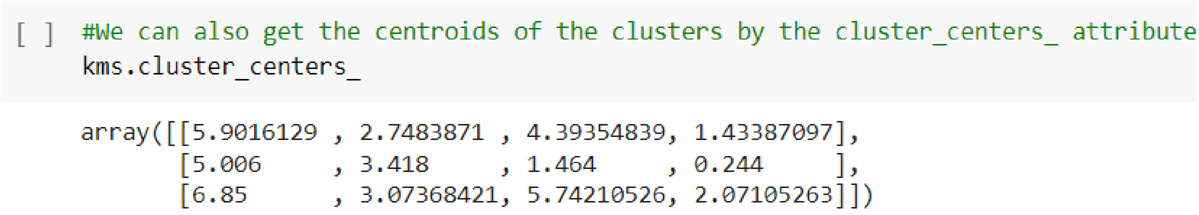


```
clusters = clustering_data.copy()
clusters['Cluster_Prediction'] = kms.fit_predict(clustering_data)
clusters.head()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Cluster_Prediction
0	5.1	3.5	1.4	0.2	1
1	4.9	3.0	1.4	0.2	1
2	4.7	3.2	1.3	0.2	1
3	4.6	3.1	1.5	0.2	1
4	5.0	3.6	1.4	0.2	1

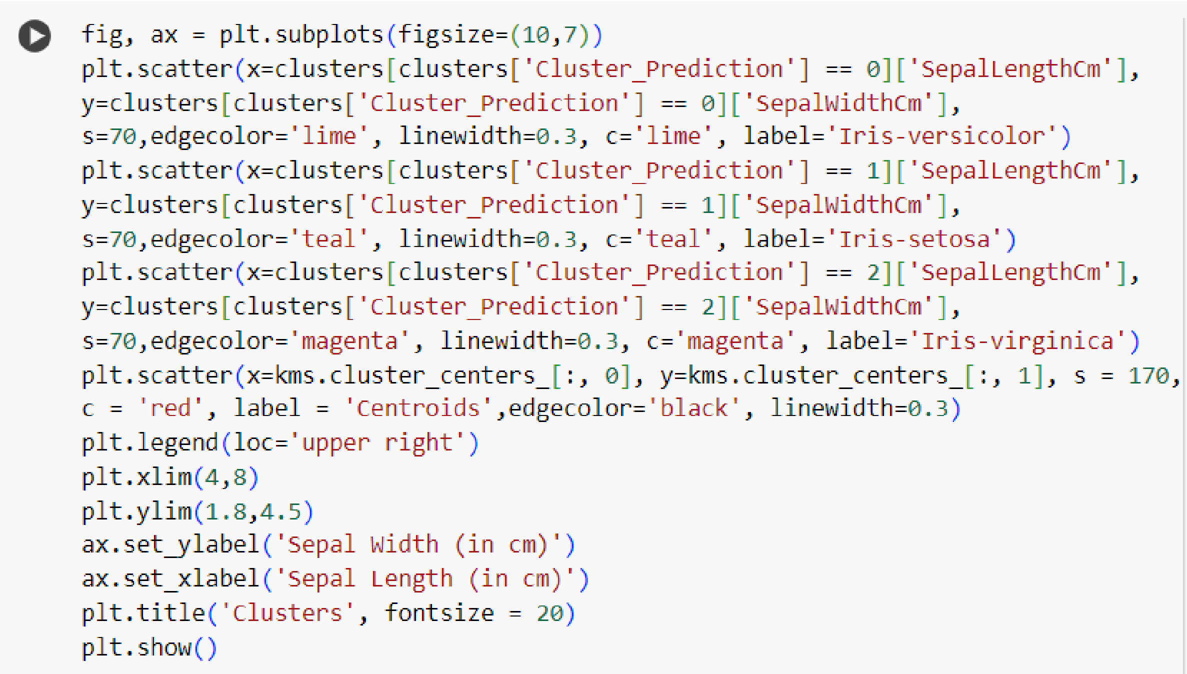
7. FINDING CENTROID OF CLUSTERS



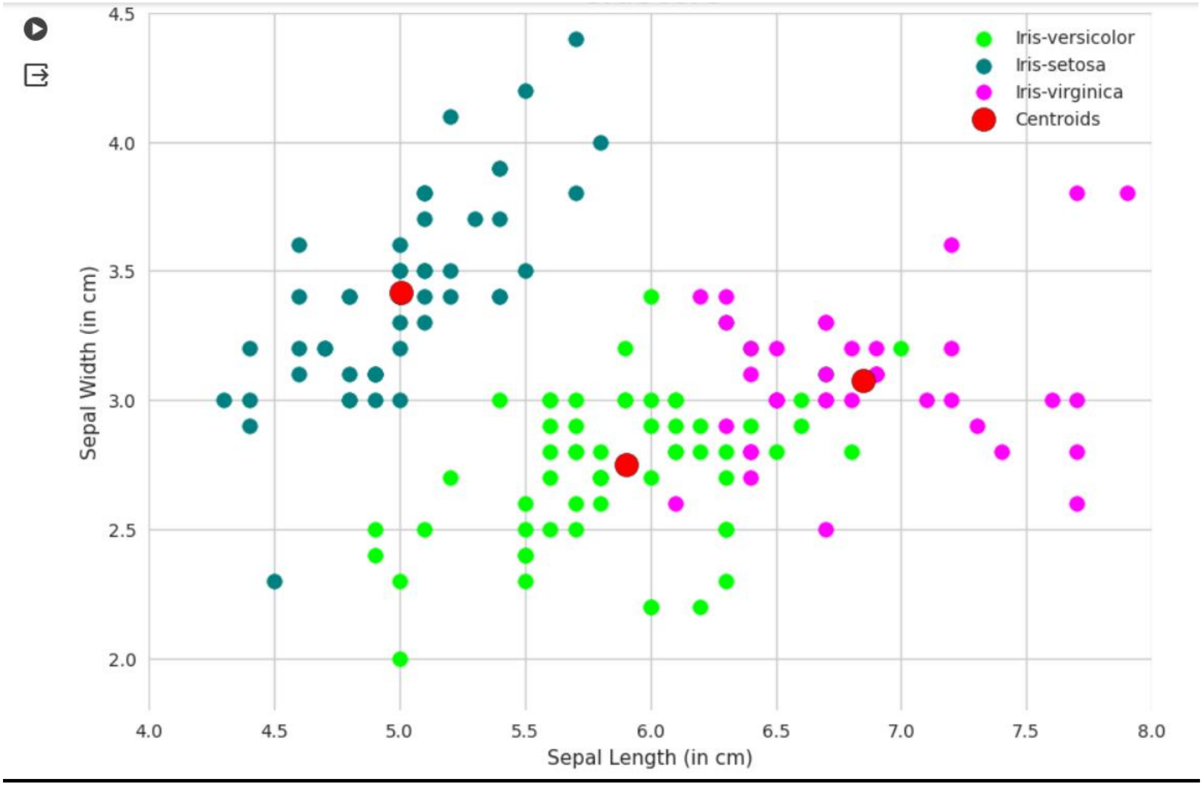
```
#We can also get the centroids of the clusters by the cluster_centers_ attribute
kms.cluster_centers_
```

```
array([[5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
       [5.006     , 3.418     , 1.464     , 0.244     ],
       [6.85      , 3.07368421, 5.74210526, 2.07105263]])
```

8. PLOTTING GRAPH FOR NEW CLUSTERED DATA



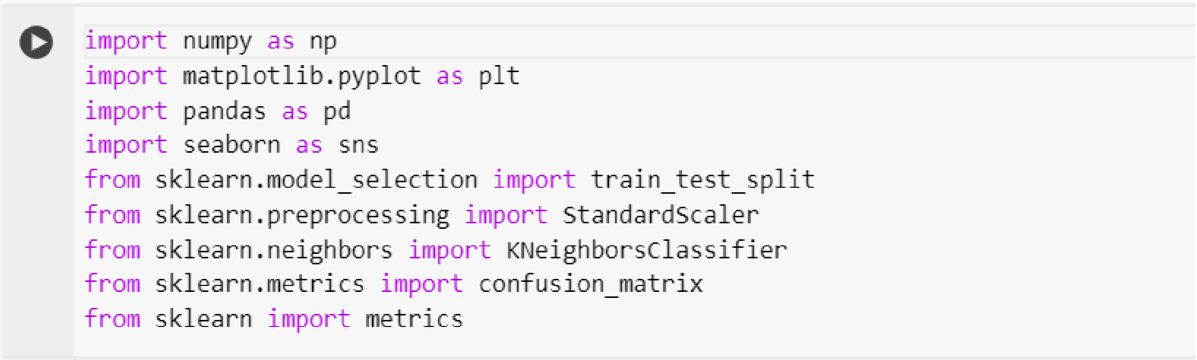
```
fig, ax = plt.subplots(figsize=(10,7))
plt.scatter(x=clusters[clusters['Cluster_Prediction'] == 0]['SepalLengthCm'],
y=clusters[clusters['Cluster_Prediction'] == 0]['SepalWidthCm'],
s=70,edgecolor='lime', linewidth=0.3, c='lime', label='Iris-versicolor')
plt.scatter(x=clusters[clusters['Cluster_Prediction'] == 1]['SepalLengthCm'],
y=clusters[clusters['Cluster_Prediction'] == 1]['SepalWidthCm'],
s=70,edgecolor='teal', linewidth=0.3, c='teal', label='Iris-setosa')
plt.scatter(x=clusters[clusters['Cluster_Prediction'] == 2]['SepalLengthCm'],
y=clusters[clusters['Cluster_Prediction'] == 2]['SepalWidthCm'],
s=70,edgecolor='magenta', linewidth=0.3, c='magenta', label='Iris-virginica')
plt.scatter(x=kms.cluster_centers_[:, 0], y=kms.cluster_centers_[:, 1], s = 170,
c = 'red', label = 'Centroids',edgecolor='black', linewidth=0.3)
plt.legend(loc='upper right')
plt.xlim(4,8)
plt.ylim(1.8,4.5)
ax.set_ylabel('Sepal Width (in cm)')
ax.set_xlabel('Sepal Length (in cm)')
plt.title('Clusters', fontsize = 20)
plt.show()
```


Iris-versicolor
Iris-setosa
Iris-virginica
Centroids
Sepal Width (in cm)
Sepal Length (in cm)
4.5
4.0
3.5
3.0
2.5
2.0
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

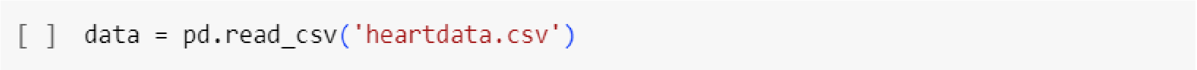
Program-9

Objective: To perform KNN model for heart disease prediction.

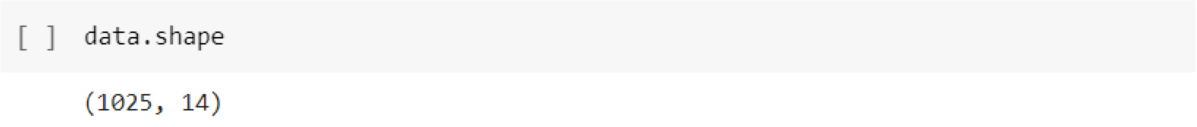
1. IMPORT REQUIRED LIBRARIES AND HEART DISEASE DATASET:



```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn import metrics
```

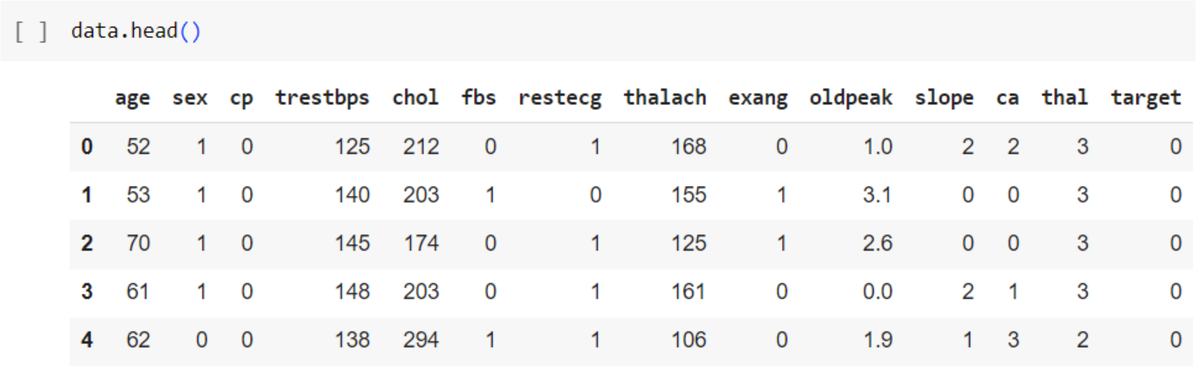


```
[ ] data = pd.read_csv('heartdata.csv')
```



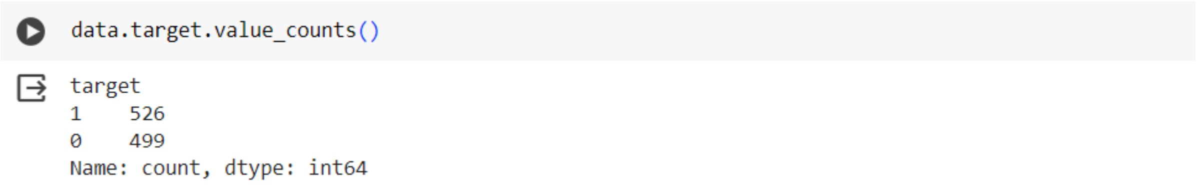
```
[ ] data.shape

(1025, 14)
```

2. PRINTING TOP 5 VALUES OF DATASET AND COUNTING TARGET VALUES



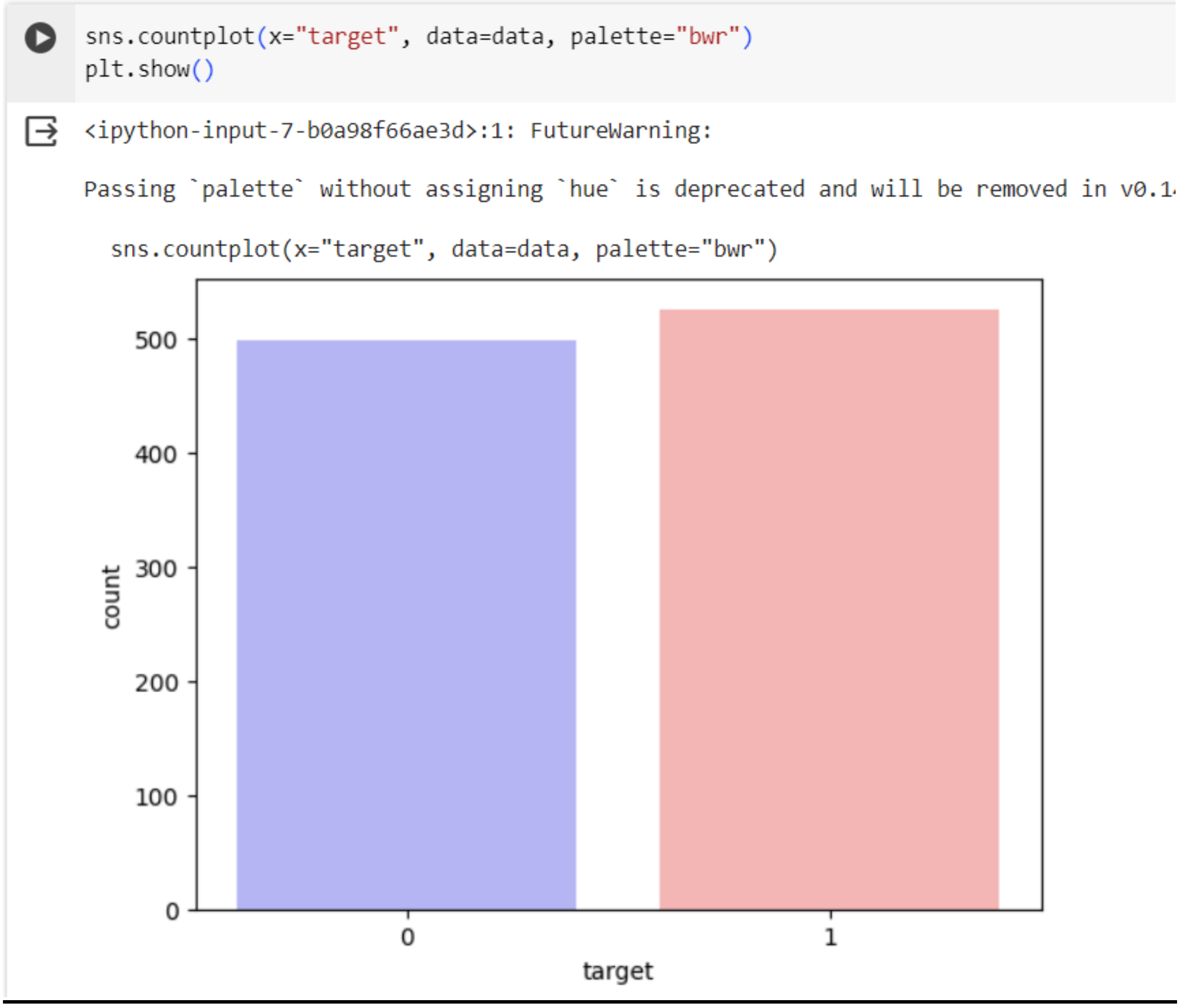
```
[ ] data.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0



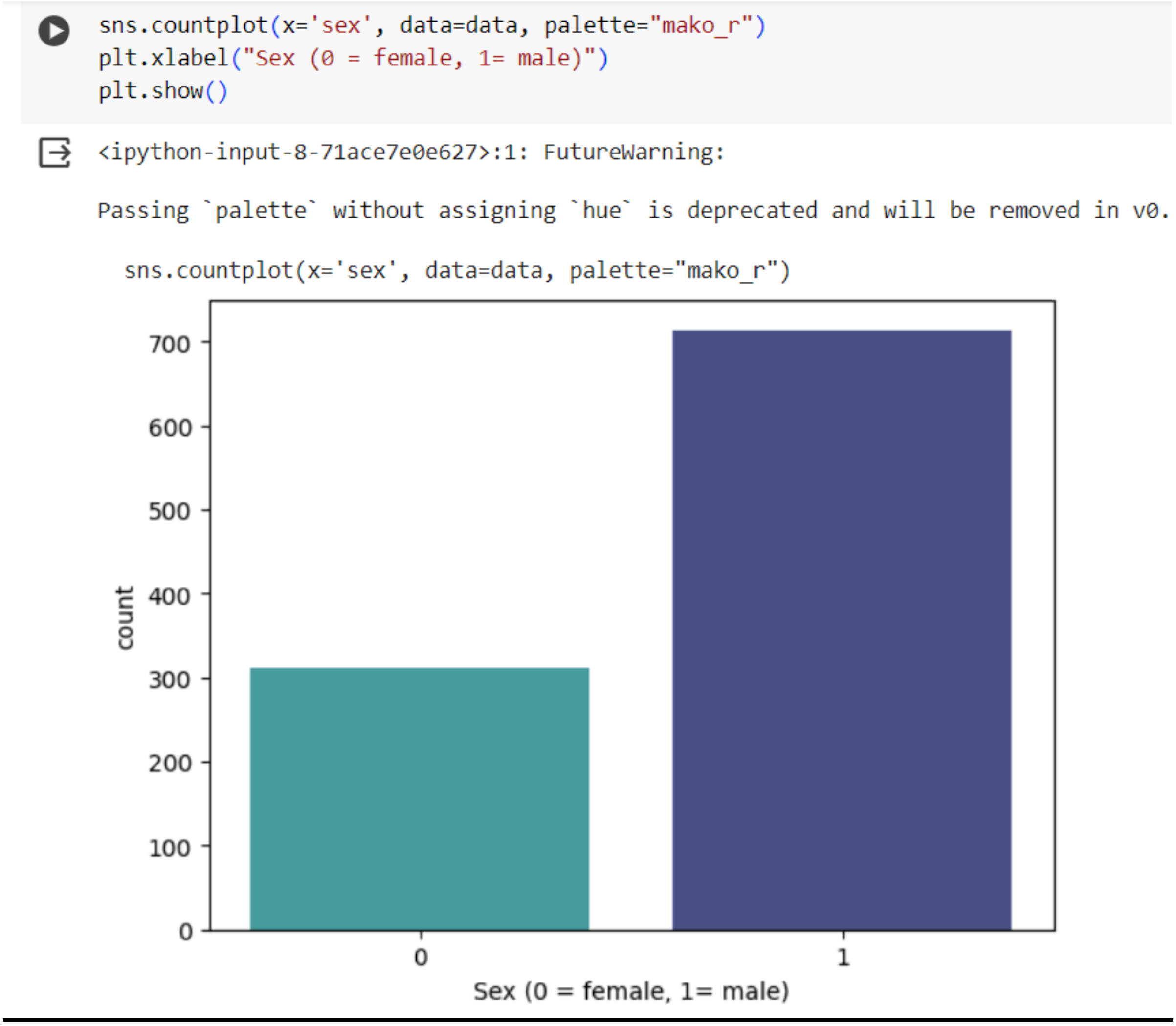
```
data.target.value_counts()

target
1    526
0    499
Name: count, dtype: int64
```

3. PLOTTING GRAPH FOR TARGET



```
sns.countplot(x="target", data=data, palette="bwr")
plt.show()
```

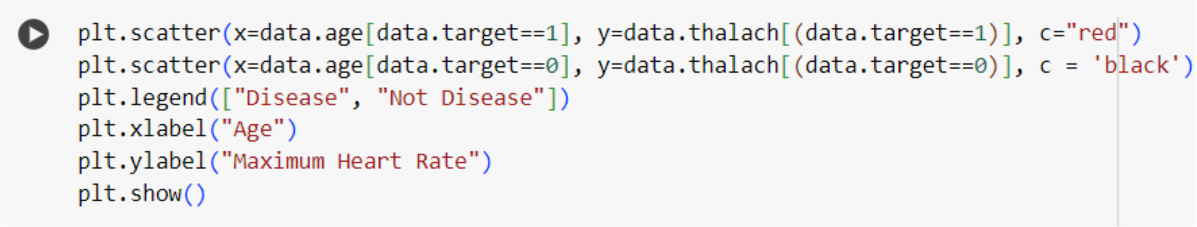
```
<ipython-input-7-b0a98f66ae3d>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1

  sns.countplot(x="target", data=data, palette="bwr")
```



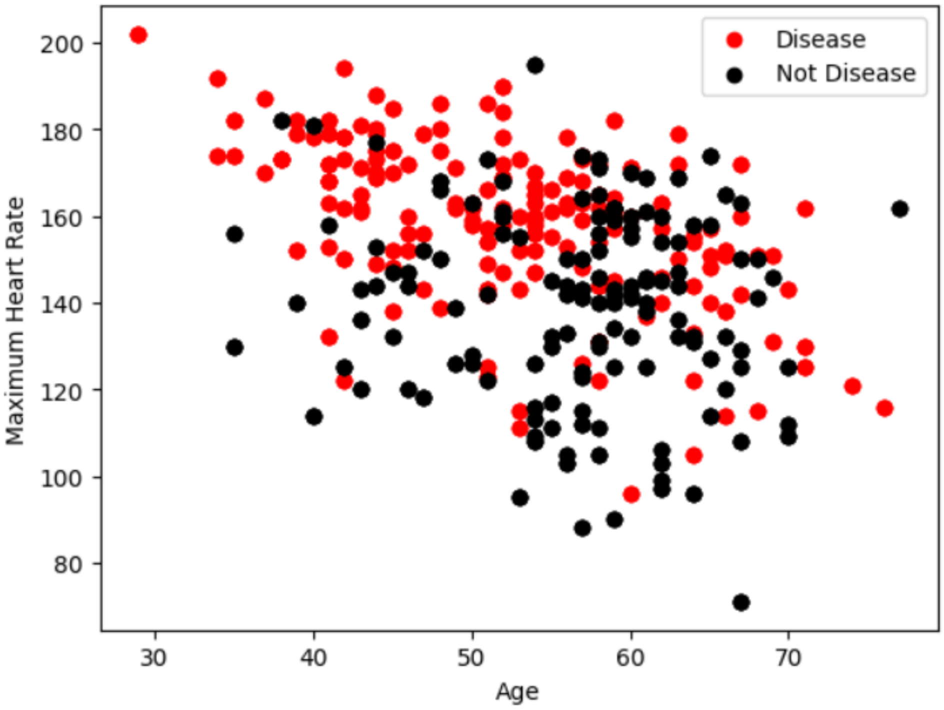
```
sns.countplot(x='sex', data=data, palette="mako_r")
plt.xlabel("Sex (0 = female, 1= male)")
plt.show()
```

```
<ipython-input-8-71ace7e0e627>:1: FutureWarning: 

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.

  sns.countplot(x='sex', data=data, palette="mako_r")
```

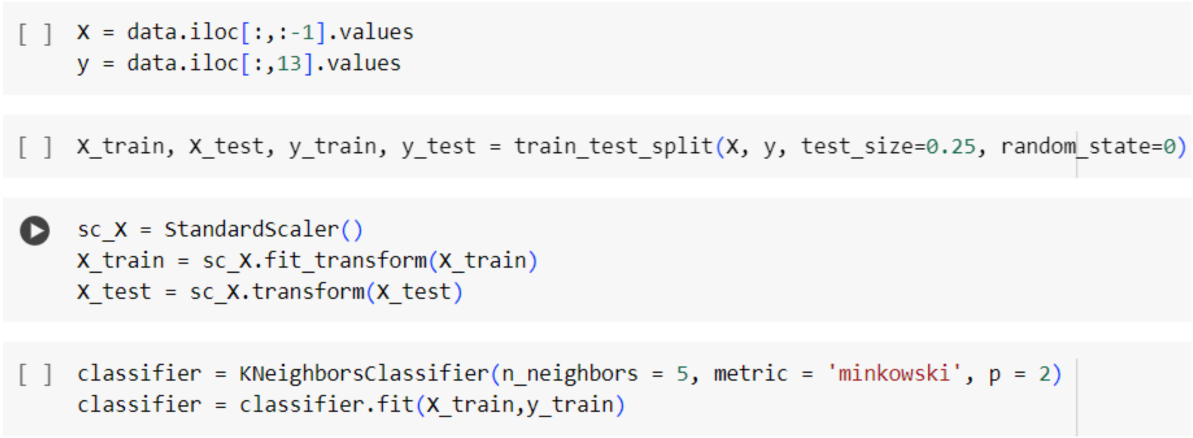
4. PLOTTING DATASET AGAINST AGE



```
plt.scatter(x=data.age[data.target==1], y=data.thalach[(data.target==1)], c="red")
plt.scatter(x=data.age[data.target==0], y=data.thalach[(data.target==0)], c = 'black')
plt.legend(["Disease", "Not Disease"])
plt.xlabel("Age")
plt.ylabel("Maximum Heart Rate")
plt.show()
```



5. PERFORM KNN BY SPLITTING TO TRAIN AND TEST



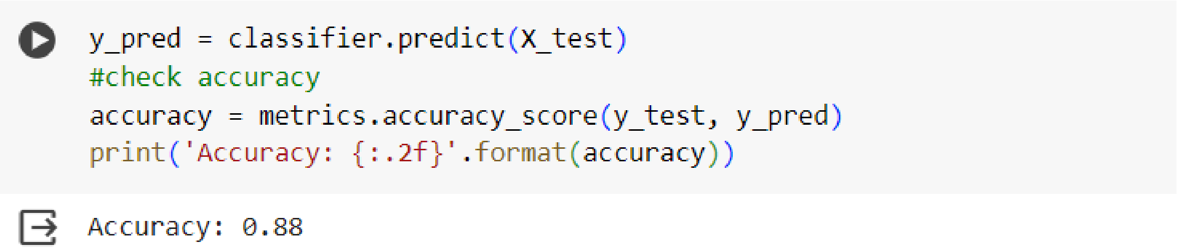
```
X = data.iloc[:,:-1].values
y = data.iloc[:,13].values
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
```

```
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

```
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier = classifier.fit(X_train,y_train)
```

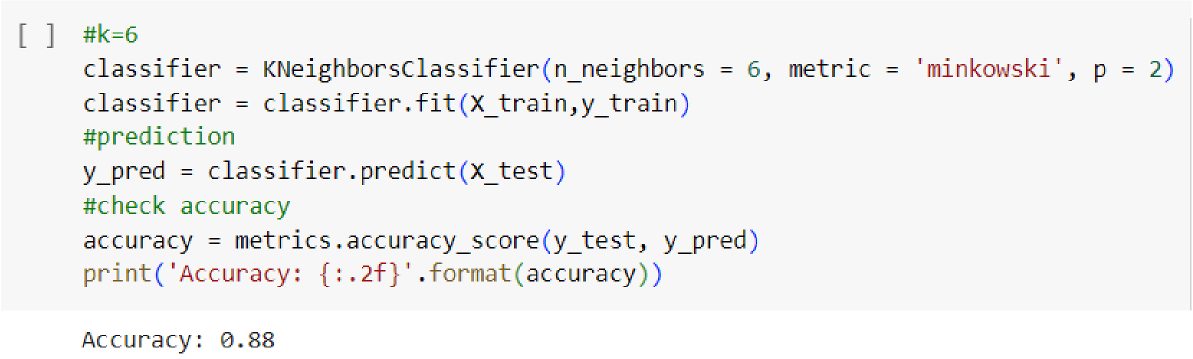
6. CHECKING ACCURACY



```
y_pred = classifier.predict(X_test)
#check accuracy
accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy: {:.2f}'.format(accuracy))
```

```
Accuracy: 0.88
```

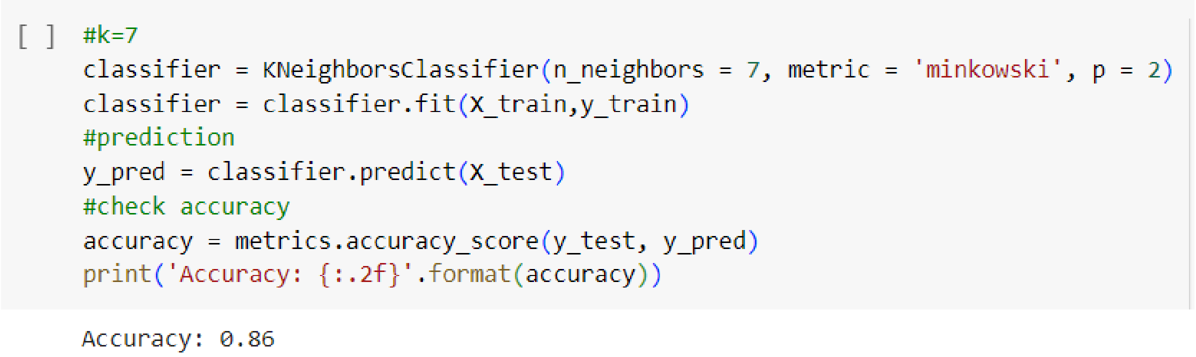
7. CHECKING ACCURACY FOR K = 6



```
#k=6
classifier = KNeighborsClassifier(n_neighbors = 6, metric = 'minkowski', p = 2)
classifier = classifier.fit(X_train,y_train)
#prediction
y_pred = classifier.predict(X_test)
#check accuracy
accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy: {:.2f}'.format(accuracy))
```

```
Accuracy: 0.88
```

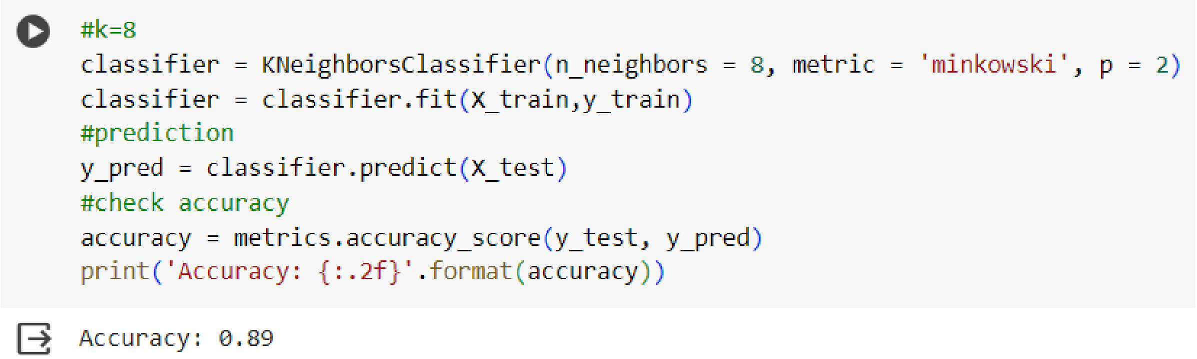
8. CHECKING ACCURACY FOR K = 7



```
#k=7
classifier = KNeighborsClassifier(n_neighbors = 7, metric = 'minkowski', p = 2)
classifier = classifier.fit(X_train,y_train)
#prediction
y_pred = classifier.predict(X_test)
#check accuracy
accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy: {:.2f}'.format(accuracy))
```

```
Accuracy: 0.86
```

9. CHECKING ACCURACY FOR K = 8



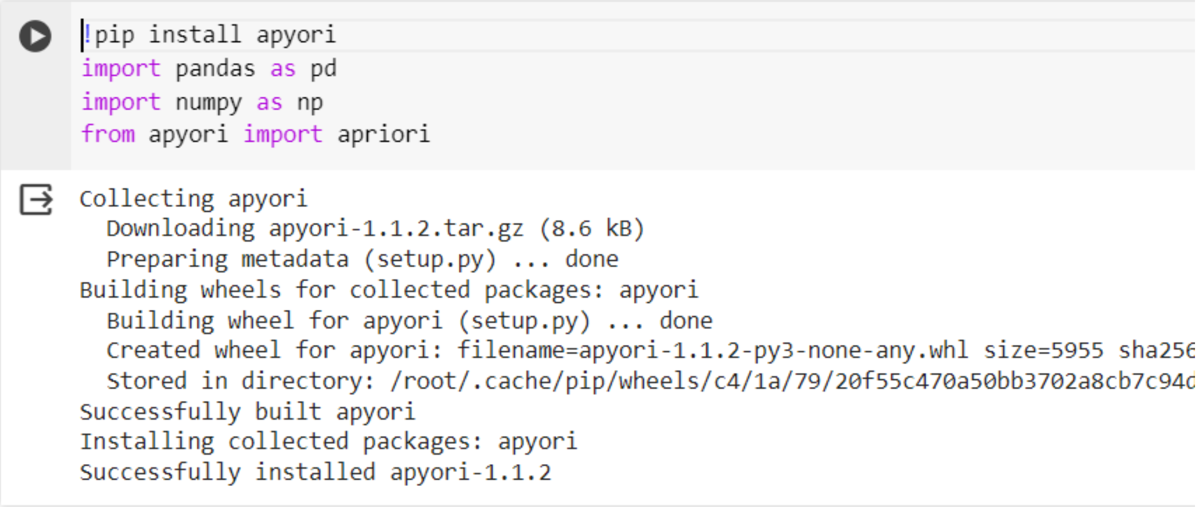
```
#k=8
classifier = KNeighborsClassifier(n_neighbors = 8, metric = 'minkowski', p = 2)
classifier = classifier.fit(X_train,y_train)
#prediction
y_pred = classifier.predict(X_test)
#check accuracy
accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy: {:.2f}'.format(accuracy))
```

```
Accuracy: 0.89
```

Program-10

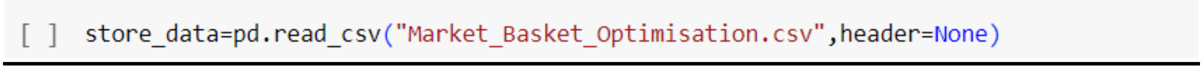
Objective: To perform Apriori Algorithm on Market dataset.

1. IMPORT REQUIRED LIBRARIES AND MARKET DATASET:



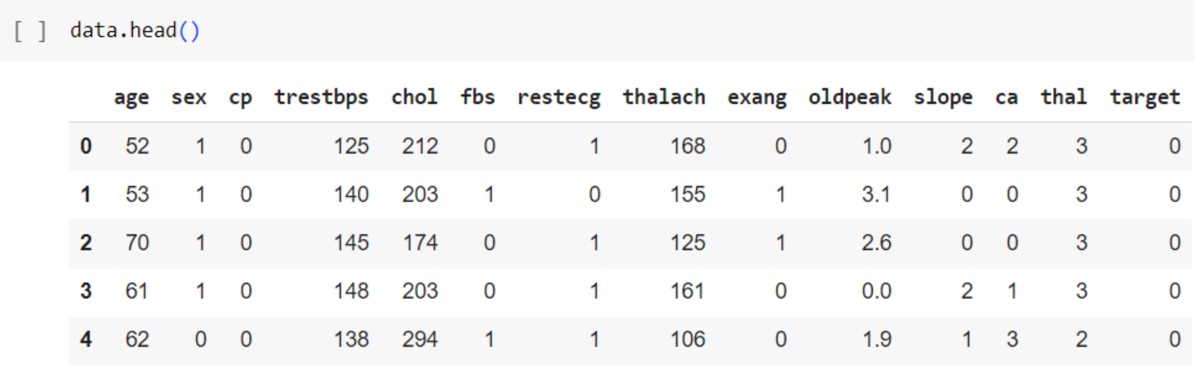
```
!pip install apyori
import pandas as pd
import numpy as np
from apyori import apriori
```

```
Collecting apyori
  Downloading apyori-1.1.2.tar.gz (8.6 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: apyori
  Building wheel for apyori (setup.py) ... done
  Created wheel for apyori: filename=apyori-1.1.2-py3-none-any.whl size=5955 sha256
  Stored in directory: /root/.cache/pip/wheels/c4/1a/79/20f55c470a50bb3702a8cb7c94c
Successfully built apyori
Installing collected packages: apyori
Successfully installed apyori-1.1.2
```



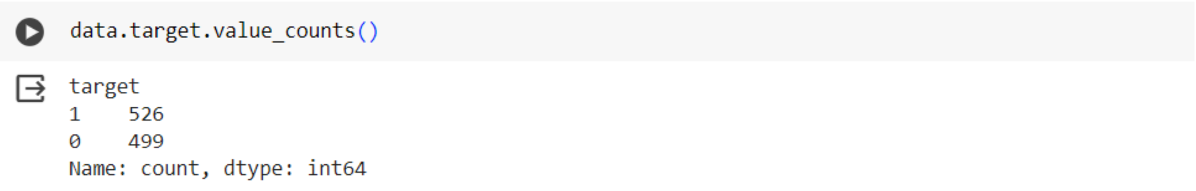
```
[ ] store_data=pd.read_csv("Market_Basket_Optimisation.csv",header=None)
```

2. PRINTING TOP 5 VALUES OF DATASET AND COUNTING TARGET VALUES



```
[ ] data.head()
```

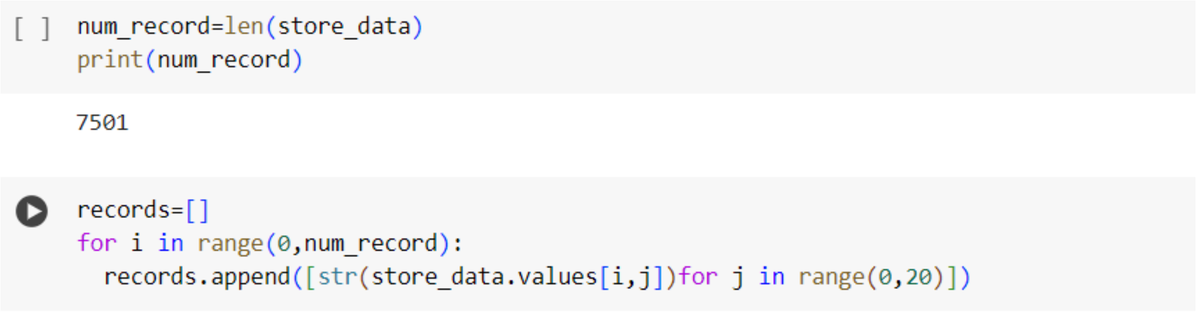
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0



```
data.target.value_counts()
```

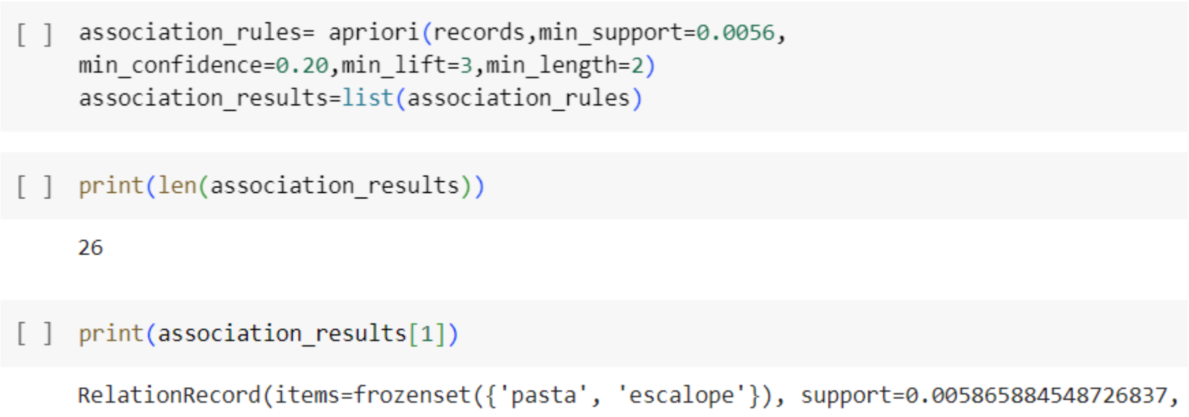
```
target
1    526
0    499
Name: count, dtype: int64
```

3. TAKING RECORDS



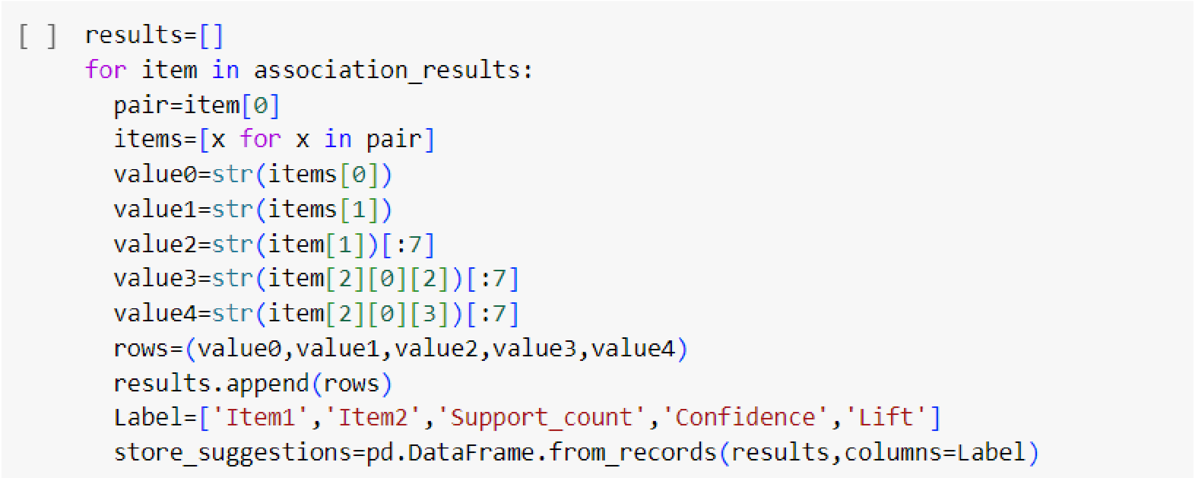
```
[ ] num_record=len(store_data)
    print(num_record)

    7501

    records=[]
    for i in range(0,num_record):
      records.append([str(store_data.values[i,j])for j in range(0,20)])
```

4. PERFORM APRIORI FUNCTIONS



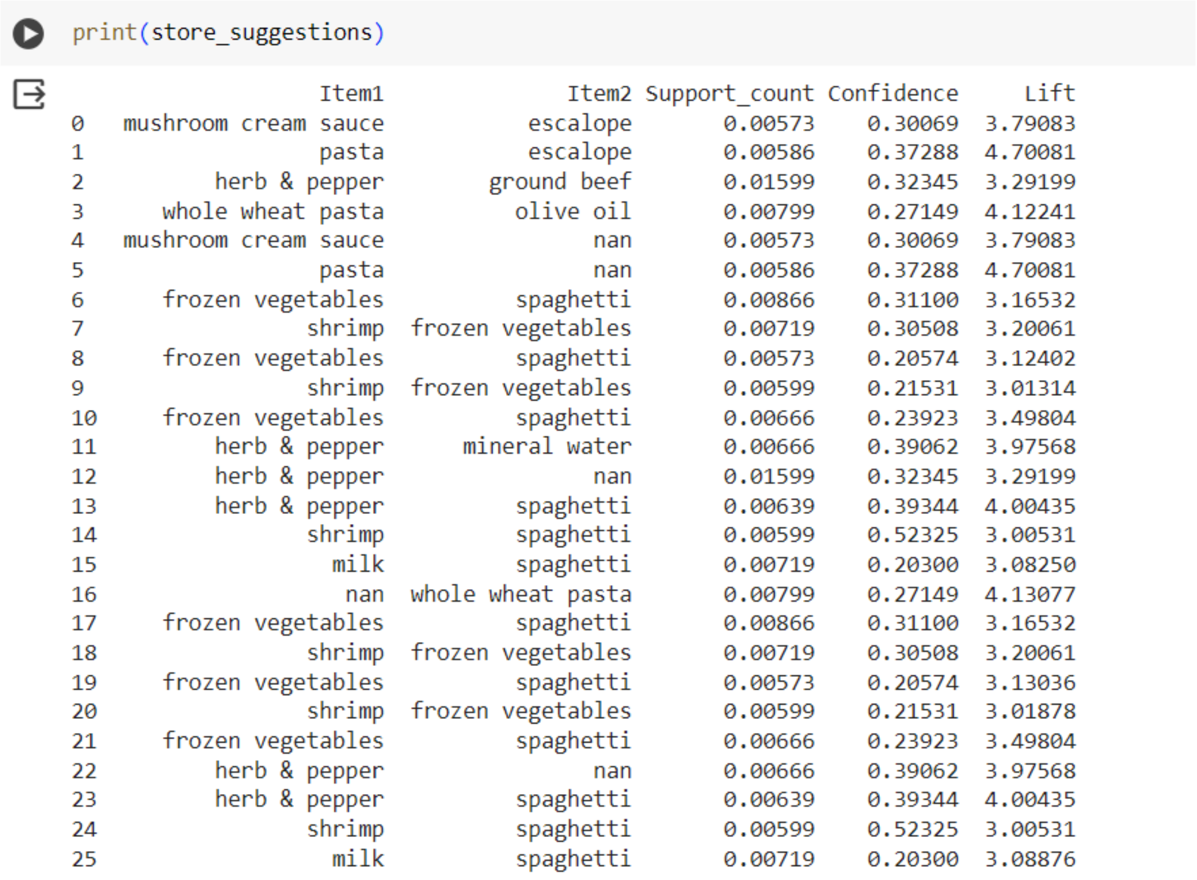
```
[ ] association_rules= apriori(records,min_support=0.0056,
    min_confidence=0.20,min_lift=3,min_length=2)
    association_results=list(association_rules)

[ ] print(len(association_results))

    26

[ ] print(association_results[1])

    RelationRecord(items=frozenset({'pasta', 'escalope'}), support=0.005865884548726837,
```

5. SETTING VALUES FOR RESULT



```
[ ] results=[]
    for item in association_results:
      pair=item[0]
      items=[x for x in pair]
      value0=str(items[0])
      value1=str(items[1])
      value2=str(item[1])[:7]
      value3=str(item[2][0][2])[:7]
      value4=str(item[2][0][3])[:7]
      rows=(value0,value1,value2,value3,value4)
      results.append(rows)
      Label=['Item1','Item2','Support_count','Confidence','Lift']
      store_suggestions=pd.DataFrame.from_records(results,columns=Label)
```

6. PRINTING ALL OUTPUT



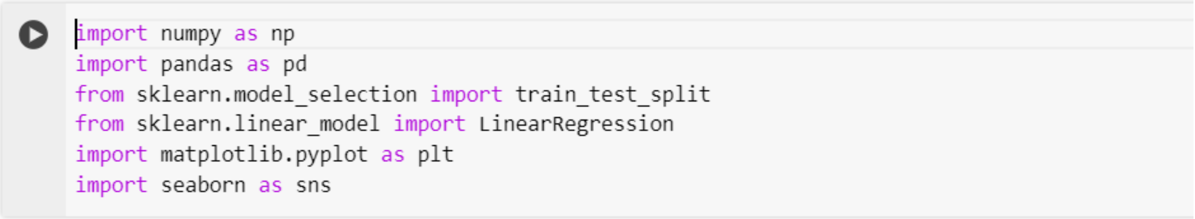
```
print(store_suggestions)
```

```
                   Item1              Item2 Support_count Confidence     Lift
0   mushroom cream sauce           escalope       0.00573    0.30069  3.79083
1                  pasta           escalope       0.00586    0.37288  4.70081
2          herb & pepper        ground beef       0.01599    0.32345  3.29199
3      whole wheat pasta          olive oil       0.00799    0.27149  4.12241
4   mushroom cream sauce                nan       0.00573    0.30069  3.79083
5                  pasta                nan       0.00586    0.37288  4.70081
6      frozen vegetables          spaghetti       0.00866    0.31100  3.16532
7                 shrimp  frozen vegetables       0.00719    0.30508  3.20061
8      frozen vegetables          spaghetti       0.00573    0.20574  3.12402
9                 shrimp  frozen vegetables       0.00599    0.21531  3.01314
10     frozen vegetables          spaghetti       0.00666    0.23923  3.49804
11         herb & pepper      mineral water       0.00666    0.39062  3.97568
12         herb & pepper                nan       0.01599    0.32345  3.29199
13         herb & pepper          spaghetti       0.00639    0.39344  4.00435
14                shrimp          spaghetti       0.00599    0.52325  3.00531
15                  milk          spaghetti       0.00719    0.20300  3.08250
16                   nan  whole wheat pasta       0.00799    0.27149  4.13077
17     frozen vegetables          spaghetti       0.00866    0.31100  3.16532
18                shrimp  frozen vegetables       0.00719    0.30508  3.20061
19     frozen vegetables          spaghetti       0.00573    0.20574  3.13036
20                shrimp  frozen vegetables       0.00599    0.21531  3.01878
21     frozen vegetables          spaghetti       0.00666    0.23923  3.49804
22         herb & pepper                nan       0.00666    0.39062  3.97568
23         herb & pepper          spaghetti       0.00639    0.39344  4.00435
24                shrimp          spaghetti       0.00599    0.52325  3.00531
25                  milk          spaghetti       0.00719    0.20300  3.08876
```

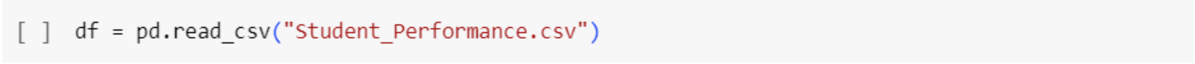
Program-11

Objective: To perform Multiple Linear Regression in Python

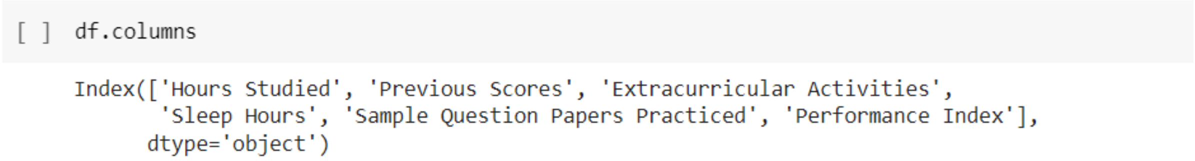
1. IMPORT REQUIRED LIBRARIES AND MARKET DATASET:



```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import seaborn as sns
```

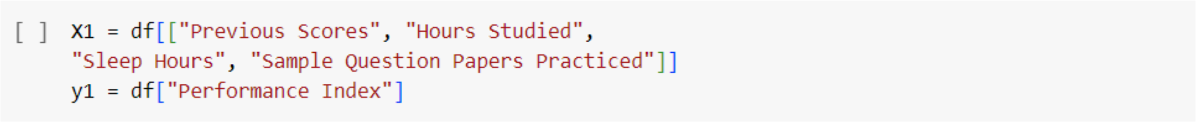


```
[ ]  df = pd.read_csv("Student_Performance.csv")
```

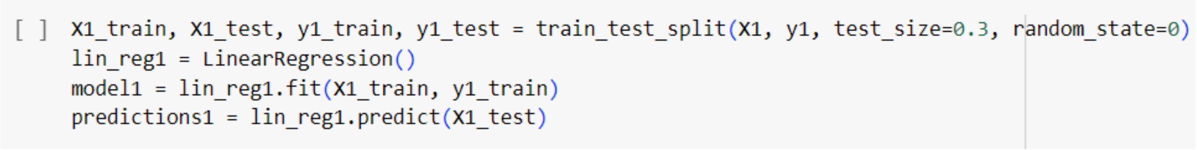


```
[ ]  df.columns

     Index(['Hours Studied', 'Previous Scores', 'Extracurricular Activities',
            'Sleep Hours', 'Sample Question Papers Practiced', 'Performance Index'],
           dtype='object')
```

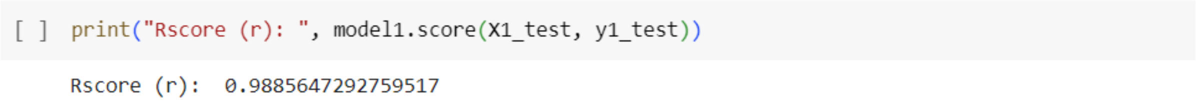
2. TRAINING DATA AND ACESSING COLUMNS OF DATASET



```
[ ]  X1 = df[["Previous Scores", "Hours Studied",
     "Sleep Hours", "Sample Question Papers Practiced"]]
     y1 = df["Performance Index"]
```

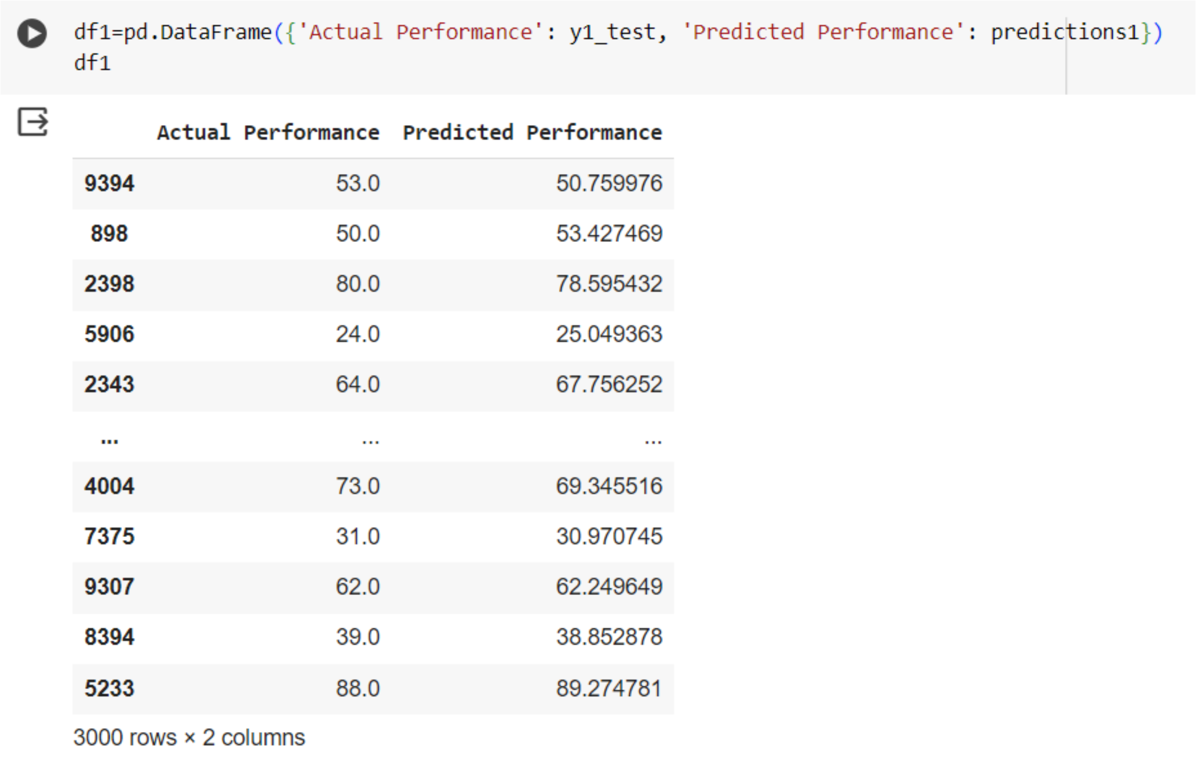


```
[ ]  X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.3, random_state=0)
     lin_reg1 = LinearRegression()
     model1 = lin_reg1.fit(X1_train, y1_train)
     predictions1 = lin_reg1.predict(X1_test)
```



```
[ ]  print("Rscore (r): ", model1.score(X1_test, y1_test))

     Rscore (r):  0.9885647292759517
```

3. PRINTING VALUES OF DATASET

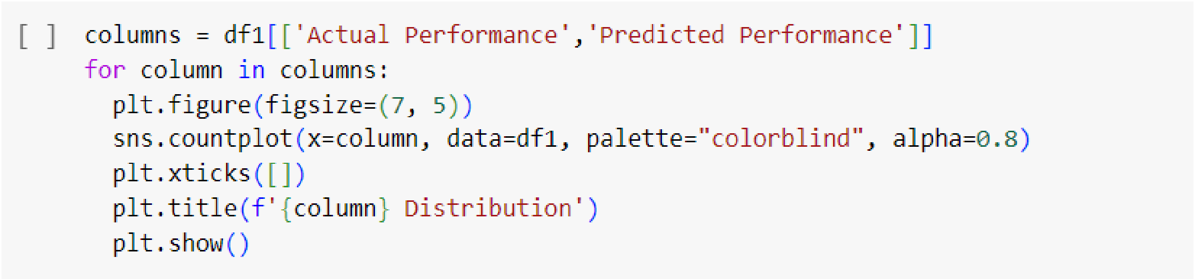


```python
df1=pd.DataFrame({'Actual Performance': y1_test, 'Predicted Performance': predictions1})
df1
```

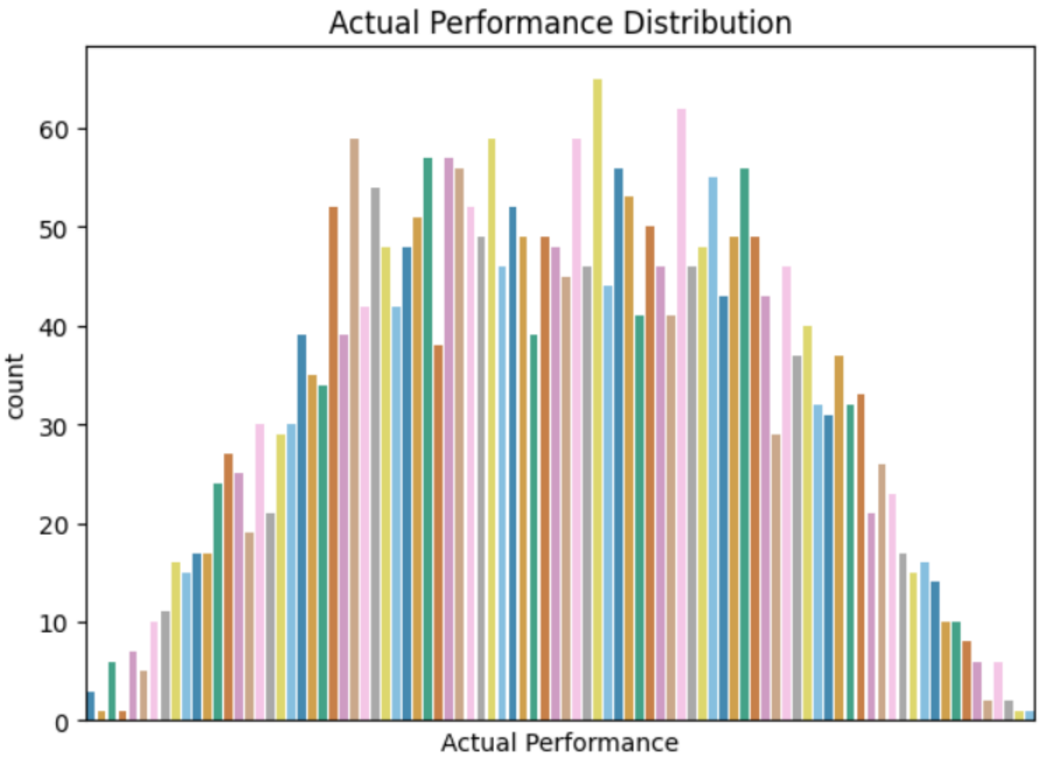
	Actual Performance	Predicted Performance
9394	53.0	50.759976
898	50.0	53.427469
2398	80.0	78.595432
5906	24.0	25.049363
2343	64.0	67.756252
...	...	...
4004	73.0	69.345516
7375	31.0	30.970745
9307	62.0	62.249649
8394	39.0	38.852878
5233	88.0	89.274781

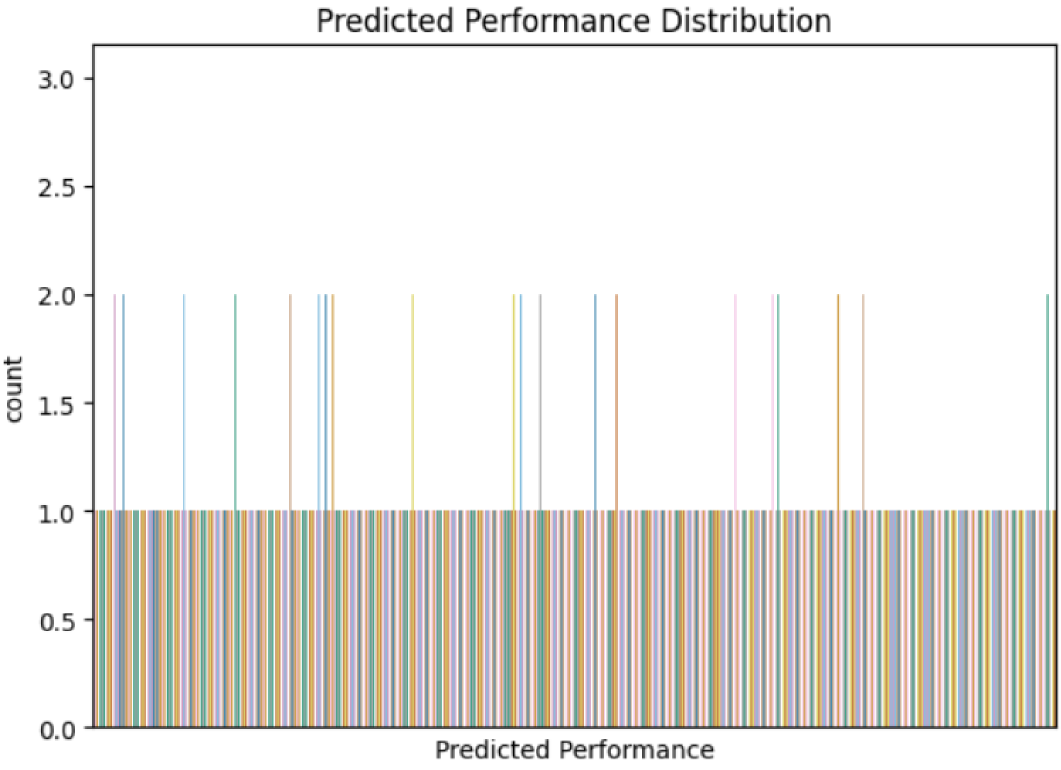
3000 rows × 2 columns

4. PLOTTING GRAPH FOR ACTUAL AND PREDICTED PERFORMANCE



```python
columns = df1[['Actual Performance','Predicted Performance']]
for column in columns:
  plt.figure(figsize=(7, 5))
  sns.countplot(x=column, data=df1, palette="colorblind", alpha=0.8)
  plt.xticks([])
  plt.title(f'{column} Distribution')
  plt.show()
```


Actual Performance Distribution
count
60
50
40
30
20
10
0
Actual Performance


Predicted Performance Distribution
count
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Predicted Performance