Applied Machine Learning

5th Sem Practical File(IoT-009)

Submitted to:-

DEI

Under the guidance:-

Dr. R.S.Pavithr & Miss. Sadhana Singh

Submitted by:

Dayal Nigam

(1803743)

DEPARTMENT OF PHYSICS & COMPUTER

SCIENCE FACULTY OF SCIENCE

DAYALBAGH EDUCATIONAL INSTITUTE

DAYALBAGH AGRA (UP)-282005

Table of Contents

Applied Machine Learning 1
Practical No: 1 3
    How does it works? 3
Practical No: 2 4
Code 5
Output 6
Practical No: 3 8
Code 9
Output 10
Practical No: 4 12
Code 13
Output 14
Practical No: 5 15
Code 16
Output 18
Practical No: 6 25
Code 26
Output 28
Practical No: 7 32
Code 33
Output 34
Practical No: 8 35
    What is Support Vector Machine? 35
Code 36
Output 37
Practical No: 9 39
Code 40
Output 41
Practical No: 10 54
Code 55
Output 56

Practical No: 1

Question :- Introduction to Machine Learning.

Theory Explanation :- A good start at a Machine Learning definition is that it is a core sub-area of Artificial Intelligence (AI). ML applications learn from experience (well data) like humans without direct programming. When exposed to new data, these applications learn, grow, change, and develop by themselves. In other words, with Machine Learning, computers find insightful information without being told where to look. Instead, they do this by leveraging algorithms that learn from data in an iterative process.

How does it works?
The Machine Learning process starts with inputting training data into the selected algorithm. Training data being known or unknown data to develop the final Machine Learning algorithm. The type of training data input does impact the algorithm, and that concept will be covered further momentarily.

Types Of Machine Learning:-

- Supervised Learning
    1. Polynomial Regression
    2. Linear Regression
    3. Random Forest
    4. Decision Tree
    5. Logistic Regression
    6. K-Nearest Neighbors
- Unsupervised Learning
    1. K-Means Clustering
- Reinforcement Learning
    1) The algorithm discovers data through a process of trial and error and then decides what action results in higher rewards. Three major components make up reinforcement learning: the agent, the environment, and the actions.

Practical No: 2

Question :- Implement Simple Linear Regression on real-state dataset for house price prediction use “X2 house age” as independent variable.

Theory Explanation :- The Dataset which was provided is

1. https://drive.google.com/file/d/1NdRZeIH6Do41wIk5RfXhAkOeH0ymKODy/view?usp=sharing

Dependent variable:- Y house price of unit area
Independent variable:- X2 house age
Formula:- y=mx+c

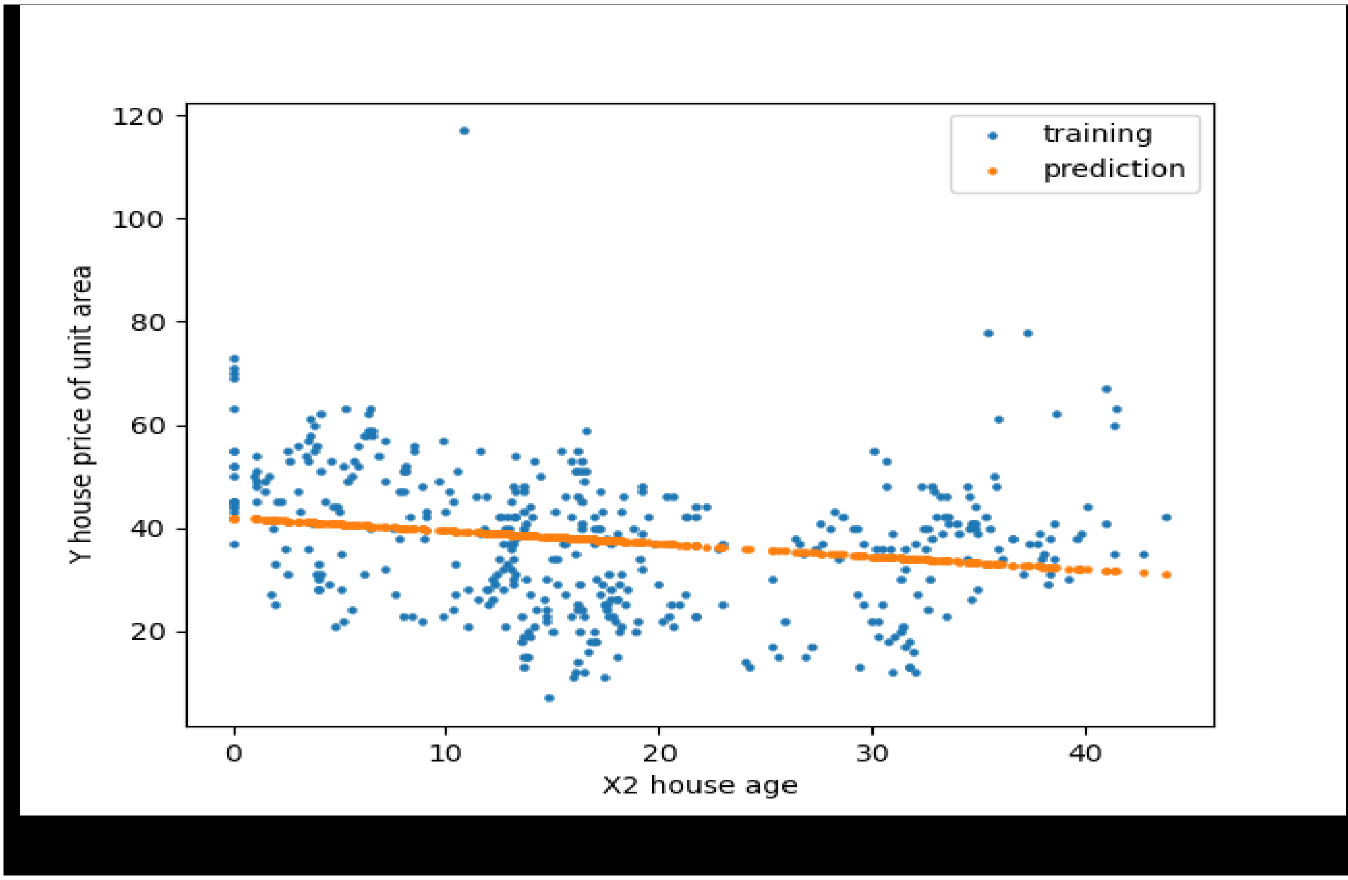
X2 house age	Y house price of unit area
32	37.9
19.5	42.2
13.3	47.3
13.3	54.8
5	43.1
7.1	32.1
34.5	40.3
20.3	46.7
31.7	18.8
17.9	22.1
34.8	41.4
6.3	58.1
13	39.3
20.4	23.8
13.2	34.3
35.7	50.5

Code

```python
import pandas as pd
import numpy as np
from sklearn import linear_model
from sklearn.metrics import mean_squared_error,r2_score
import matplotlib.pyplot as plt

data=pd.read_csv('Real_estate.csv')
print(data)

X=data.iloc[:,2:3]
Y=data.iloc[:,-1:]

data.isnull().sum(axis=0)
data.info()

from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X,Y)
y_pred=lr.predict(X)
print('Coefficients: \n', lr.coef_)
print('Intercept: \n',lr.intercept_)
print('Mean squared error: %.2f'
      % mean_squared_error(Y, y_pred))
print('Coefficient of determination: %.2f'
      % r2_score(Y, y_pred))
print(lr.score(Y,y_pred))
plt.scatter(X,Y,s=5, label='training')
plt.scatter(X,y_pred,s=5, label='prediction')
plt.xlabel('X2 house age')
plt.ylabel('Y house price of unit area')
plt.legend()
plt.show()
```

Output

```
      No  X1 transaction date  X2 house age  ...  X5 latitude  X6 longitude  Y
house price of unit area
0       1              2012.917          32.0  ...     24.98298     121.54024
37.9
1       2              2012.917          19.5  ...     24.98034     121.53951
42.2
2       3              2013.583          13.3  ...     24.98746     121.54391
47.3
3       4              2013.500          13.3  ...     24.98746     121.54391
54.8
4       5              2012.833           5.0  ...     24.97937     121.54245
43.1
..    ...                   ...           ...  ...          ...           ...
...
409   410              2013.000          13.7  ...     24.94155     121.50381
15.4
410   411              2012.667           5.6  ...     24.97433     121.54310
50.0
411   412              2013.250          18.8  ...     24.97923     121.53986
40.6
412   413              2013.000           8.1  ...     24.96674     121.54067
52.5
413   414              2013.500           6.5  ...     24.97433     121.54310
63.9

[414 rows x 8 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 414 entries, 0 to 413
Data columns (total 8 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   No                                      414 non-null    int64
 1   X1 transaction date                     414 non-null    float64
 2   X2 house age                            414 non-null    float64
 3   X3 distance to the nearest MRT station  414 non-null    float64
 4   X4 number of convenience stores         414 non-null    int64
 5   X5 latitude                             414 non-null    float64
 6   X6 longitude                            414 non-null    float64
 7   Y house price of unit area              414 non-null    float64
dtypes: float64(6), int64(2)
memory usage: 25.9 KB
Coefficients:
 [[-0.25148842]]
Intercept:
 [42.43469705]
Mean squared error: 176.50
Coefficient of determination: 0.04
-5.102052477660019
```


training
prediction
120
100
80
60
40
20
0
10
20
30
40
Y house price of unit area
X2 house age

Practical No: 3

Question :- Implement Multiple-Linear Regression on Real-State House prediction dataset. Consider all the attributes for prediction.

Theory Explanation :- The Dataset which was provided is

1. https://drive.google.com/file/d/1NdRZeIH6Do41wIk5RfXhAkOeH0ymKODy/view?usp=sharing

Dependent variable:- ‘X1 transaction date', 'X2 house age', 'X3 distance to the nearest MRT station', 'X4 number of convenience stores', 'X5 latitude', 'X6 longitude'.

Independent variable:- Y house price of unit area

Formula:- y=b0+b1x1+b2x2+b3x3+…+bnxn

Dependent Variable			Independent variable
X1 transaction date		X6 longitude	Y house price of unit area
2012.917		121.54024	37.9
2012.917		121.53951	42.2
2013.583		121.54391	47.3
2013.5		121.54391	54.8
2012.833	>>>>>>>>>	121.54245	43.1
2012.667		121.51254	32.1
2012.667		121.53642	40.3
2013.417		121.54228	46.7
2013.5		121.48458	18.8
2013.417		121.51486	22.1
2013.083		121.53372	41.4
2013.333		121.5431	58.1
2012.917		121.53737	39.3
2012.667		121.51046	23.8
2013.5		121.53406	34.3
2013.583		121.54619	50.5
2013.25		121.54458	70.1

Code

```
import pandas as pd
import numpy as np
from sklearn import linear_model
from sklearn.metrics import mean_squared_error,r2_score
from sklearn.model_selection import train_test_split
import seaborn as sns
data = pd.read_csv('Real_estate.csv')
data.head()
X = data.iloc[:, :-1]
Y = data.iloc[:, -1]
print(X)
print(Y)
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split (X, Y, test_size = 0.3,
random_state=1234)
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(x_train, y_train)
y_predict = lr.predict(x_test)
print(y_predict)
mlr_score = lr.score(x_test, y_test)
print(mlr_score*100,'%')

mlr_coefficient = lr.coef_
mlr_intercept = lr.intercept_
print(mlr_intercept)
print(mlr_coefficient)
from sklearn.metrics import mean_squared_error
import math
mlr_rmse = math.sqrt(mean_squared_error(y_test, y_predict))
```

Output

```
     No  X1 transaction date  X2 house age  ...  X4 number of convenience stores
X5 latitude  X6 longitude
0      1             2012.917          32.0  ...                              10
24.98298     121.54024
1      2             2012.917          19.5  ...                               9
24.98034     121.53951
2      3             2013.583          13.3  ...                               5
24.98746     121.54391
3      4             2013.500          13.3  ...                               5
24.98746     121.54391
4      5             2012.833           5.0  ...                               5
24.97937     121.54245
..   ...                  ...           ...  ...                             ...
...           ...
409  410             2013.000          13.7  ...                               0
24.94155     121.50381
410  411             2012.667           5.6  ...                               9
24.97433     121.54310
411  412             2013.250          18.8  ...                               7
24.97923     121.53986
412  413             2013.000           8.1  ...                               5
24.96674     121.54067
413  414             2013.500           6.5  ...                               9
24.97433     121.54310

[414 rows x 7 columns]
0      37.9
1      42.2
2      47.3
3      54.8
4      43.1
       ...
409    15.4
410    50.0
411    40.6
412    52.5
413    63.9
Name: Y house price of unit area, Length: 414, dtype: float64
[39.98175118 50.94233791 34.642237   41.33447936 41.44803301 28.44060884
 47.6832341  45.42095245 37.08876233 52.44672156 34.47574977 40.42450339
 27.41158154 30.88572973 47.36378717 43.64972588 47.02736332 27.30968244
 45.90134658 46.37718179 41.3455658  52.86563084 41.02113192 33.11149448
 32.40317878 47.73782586 46.39090603 36.44707727 33.69210502 34.78447202
 41.36907351 32.34202076 37.99731209 42.95439348 38.51049852 15.86339436
 52.67588639 47.07728764 29.16042655 15.6518418  52.75585985 37.02314169
 38.78229483 39.34077509 14.69339099 42.751746   32.18023613 38.51942156
 49.85388525 41.36461233 46.80463936 44.56120755 39.50754783 47.87379621
 49.59392722 15.19852192 27.87892964 41.89576636 43.95082579 47.9982445
 32.8739739  34.87923616 32.05944112 47.60883323 46.96642485 32.95998829
 36.29846562 45.15776999 39.98212821 31.79801589 26.45233823 36.6798582
 39.37322647 39.71217439  9.98216838  8.09133008 41.04136095 53.49868479
 37.01771291 38.3005362  38.51638022 34.86304278 41.33776162 38.36415755
 37.79871787 46.30677185 45.17662029 52.85634378 46.1567212  11.82952799
 45.26146502 43.81288591 25.2552145  41.37924075 35.25301921 13.04806047
 39.52403675 46.88879759 34.32550275 38.90137345 36.5518564  34.50536756
```

```
 42.80580394 46.16514329 45.16370184 36.13377847 31.04748153 31.49395191
 43.1039563  34.35185412 33.59124312 43.94209321 42.54563986 45.13206331
 30.30396119 49.10178987 31.38550927 47.19173681 32.42546457 46.86982554
 48.16346314 13.44700706 35.02939667 29.89143183 44.85162135]
50.258071710349924 %
-16668.388763953353
[-4.73279996e-03  3.92853167e+00 -2.32812566e-01 -4.27713278e-03
  1.02049509e+00  2.19335878e+02  2.73711244e+01]
```

Practical No: 4

Question:- Implement Polynomial regression OnlineNewsPopularity dataset taken from UCI repository.

Theory Explanation:- The Dataset which was provided is

1. https://drive.google.com/file/d/1ZStFWpHA6vwO0OlwgF_MC-M_ZVuLe_Rx/view?usp=sharing

To implement this on UCINewsPopularity we need to find out the features for which we consider the correlation matrix between shares and the set of columns also we plot this correlation matrix on the heat-map which basically gives us the idea and the data which shows high rate with respect to shares.

Using this heat- map we have found certain features which are features = ['kw_avg_avg','kw_max_min','kw_avg_min',kw_avg_avg',' weekday_is_saturday',' is_weekend',' LDA_03 ','abs_title_sentiment_polarity']

Also there is some NaN Values for which we have filled it using fillna().
After that we have applied LinearRegression model for predicting the values and polynomial features for feature selection.

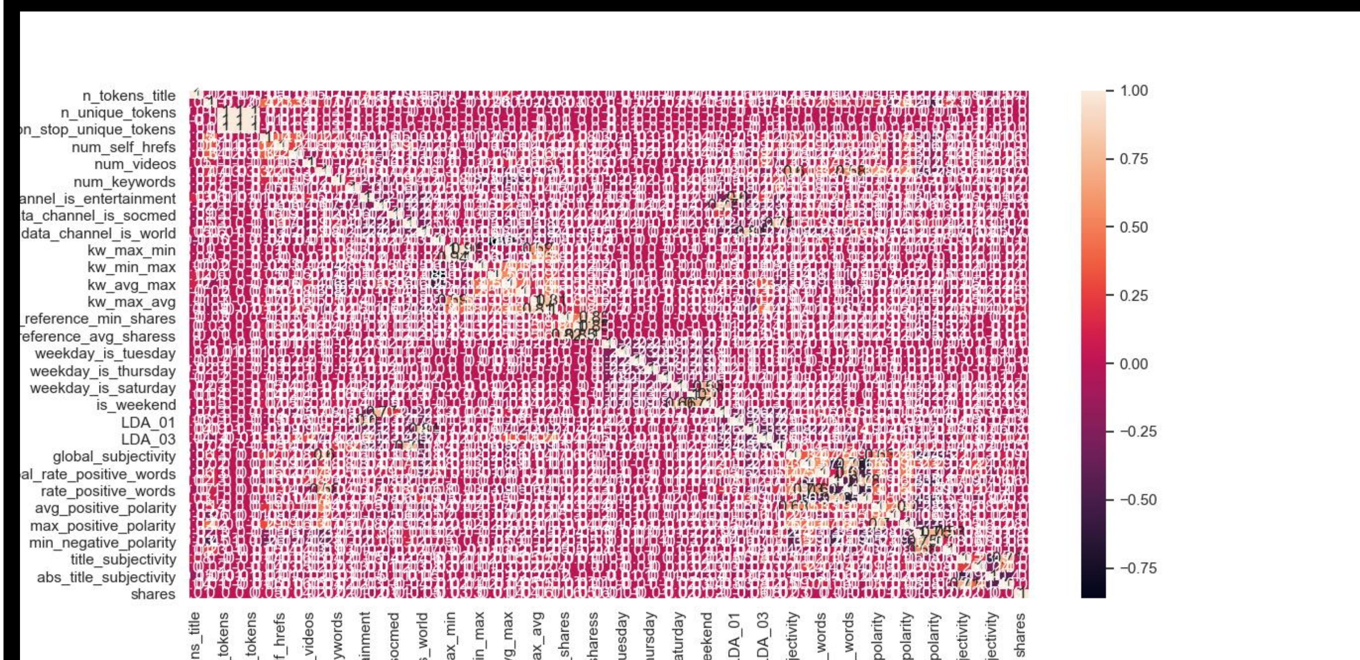
Formula = Y=bo + b1X + b2X^2 + … + $\theta_m X^m$ + **residual error**

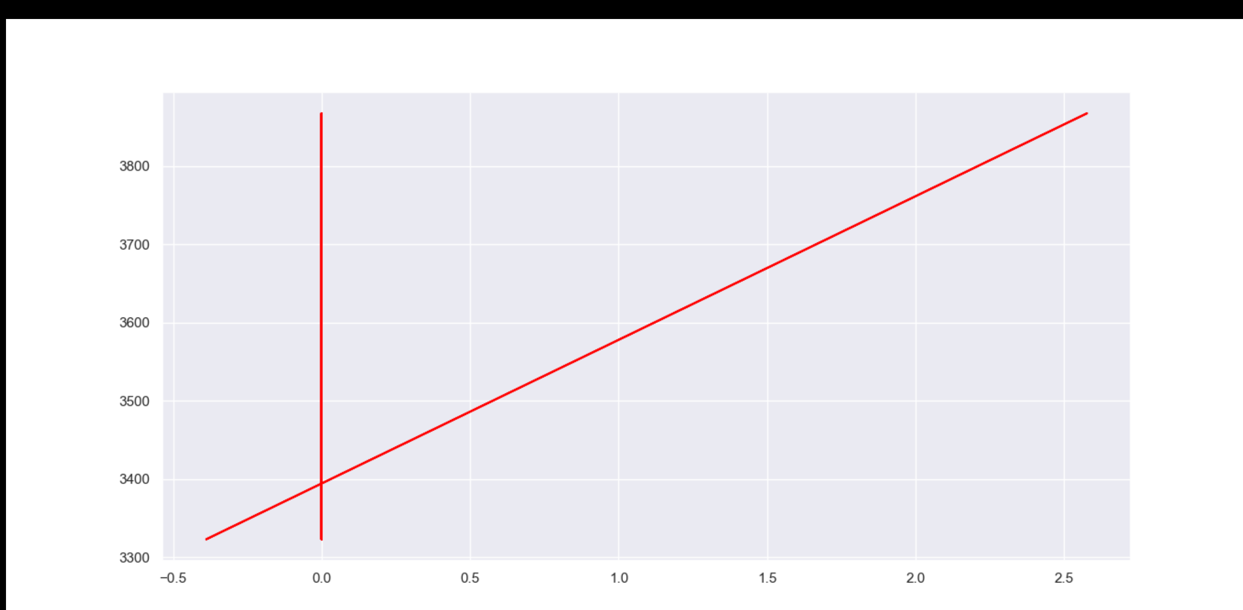
Code

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

data = pd.read_csv("OnlineNewsPopularity.csv")
data.head()

data.drop(labels=['url', ' timedelta'], axis = 1, inplace=True)
data.head()
X = data.iloc[:,0:58]
Y = data.iloc[:,-1:]
Y.head()

data1 = X
data1['shares'] = Y
data1.head()

sns.set(rc={'figure.figsize':(11.7,8.27)})
correlation_matrix = data1.corr().round(2)

sns.heatmap(data=correlation_matrix, annot=True)


df =X.reindex(columns =
['kw_avg_avg','kw_max_min','kw_avg_min','kw_avg_avg','weekday_is_saturday','
is_weekend',' LDA_03 ','abs_title_sentiment_polarity'])
df.fillna(0,inplace=True)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data_ = scaler.fit_transform(df.values)

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures

polynomial_features= PolynomialFeatures(degree=2)
x_poly = polynomial_features.fit_transform(data_)
model = LinearRegression()
model.fit(x_poly, Y)
y_poly_pred = model.predict(x_poly)
print(y_poly_pred)

Y.head()
rmse = np.sqrt(mean_squared_error(Y,y_poly_pred))
r2 = r2_score(Y,y_poly_pred)
print(rmse)
print(r2)

plt.plot(data_, y_poly_pred, color='red')
plt.show()
```

Output

```
y_poly_pred:
[[3323.12792969]
 [3323.12792969]
 [3323.12792969]
 ...
 [3323.12792969]
 [3323.12792969]
 [3323.12792969]]
root_mean_squared_error 11625.140254523036
r2_score 0.00028618907450805864
```





Practical No: 5

Question:- Implement Decision Tree regression on petrol consumption dataset and classification on Bank Deposit dataset taken from kaggle competitions.

Theory Explanation:- The Dataset which was provided is

1. https://drive.google.com/file/d/1OadZiYwgM96uE_jXI0kAJvvYO0mEadrJ/view?usp=sharing

2. https://drive.google.com/file/d/1nzMZ8a9CPrAq_r8FAtiyWQ8wi6MTpmaf/view?usp=sharing

It is a supervised learning approach used for both classification and regression tasks. A decision tree algorithm can handle both categorical and numeric data and is much efficient compared to other algorithms. Any missing value present in the data does not affect a decision tree which is why it is considered a flexible algorithm.
Interpretable and can be easily represented. Preprocessing of data such as normalization and scaling is not required which reduces the effort in building a model.

Two models are used one is :-

1. DecisionTreeRegressor()
2. DecisionTreeClassifier()

Code

```
# # Decision Tree regression on petrol consumption dataset

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor,DecisionTreeClassifier
from sklearn import metrics
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix
from sklearn import tree

data=pd.read_csv('petrol_consumption.csv')
print(data)

X = data.drop('Petrol_Consumption', axis=1)
Y = data['Petrol_Consumption']
print(X)

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.2
,random_state=42)
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)

dtc = DecisionTreeClassifier(random_state=1234)
dtc.fit(X_train, Y_train)
Y_predict = dtc.predict(X_test)

cm = confusion_matrix(Y_test, Y_predict)
score = dtc.score(X_test, Y_test)
print(score*100,'%')
print(cm)

fig = plt.figure(figsize=(50,50))
_ = tree.plot_tree(dtc, feature_names=X_test.columns, filled=True)
plt.savefig('tree1.png')

plt.plot(Y_test,Y_predict)
plt.show()

bank=pd.read_csv('Bank_Deposit_Data.csv')
print(bank)

print('any null values',bank[bank.isnull().any(axis=1)].count())

bank_data = bank.copy()
jobs = ['management','blue-
collar','technician','admin.','services','retired','self-employed','student',
'unemployed','entrepreneur','housemaid','unknown']

for j in jobs:
```

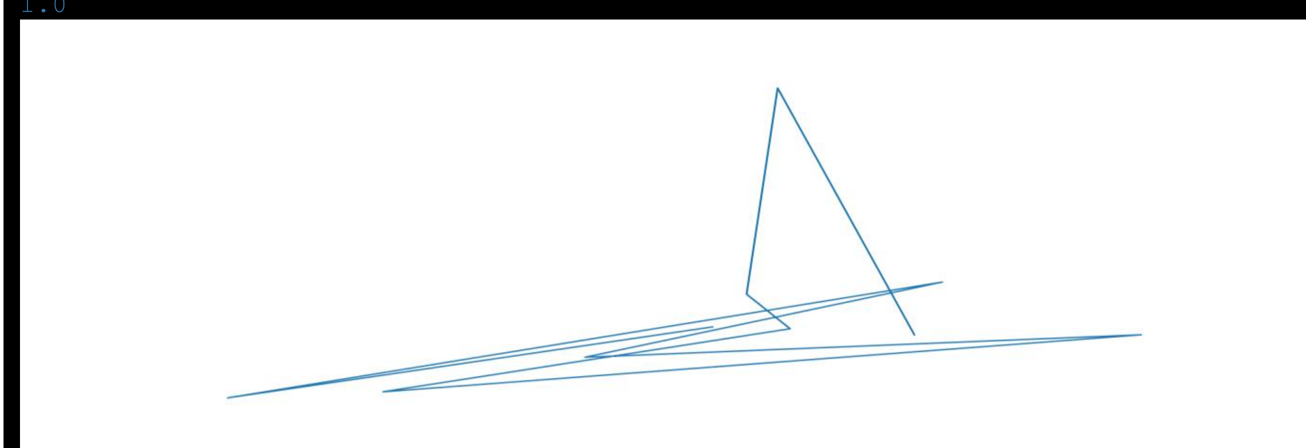
```
    print("{:15} : {:5}". format(j, len(bank_data[(bank_data.deposit == "yes") &
(bank_data.job ==j)])))

print('bank_data_job_value_counts',bank_data.job.value_counts())
print('bank_data_job_poutcome',bank_data.poutcome.value_counts())


labels = ['housing', 'default', 'loan','job', 'contact', 'marital','education',
'poutcome', 'month', 'day','deposit']
for label in labels:
    label_encoder = LabelEncoder()
    label_encoder.fit(bank_data[label])
    bank_data[label] = label_encoder.transform(bank_data[label])
print(bank_data)

X = bank_data.drop('deposit', axis=1)
y = bank_data['deposit']
print(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)
print('shape',X_train.shape)
print('shape',X_test.shape)
print('shape',y_train.shape)
print('shape',y_test.shape)


clf = DecisionTreeClassifier()
model = clf.fit(X_train, y_train)

y_pred_1 = clf.predict(X_test)
print('Actual against prediction',pd.DataFrame({'Actual':y_test,
'Predicted':y_pred_1}))


print("Accuracy:",metrics.accuracy_score(y_test, y_pred_1)*100,'%')

fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(clf, feature_names=X_test.columns, class_names=['0','1'],
filled=True)
plt.savefig('tree2.png')
```

Output

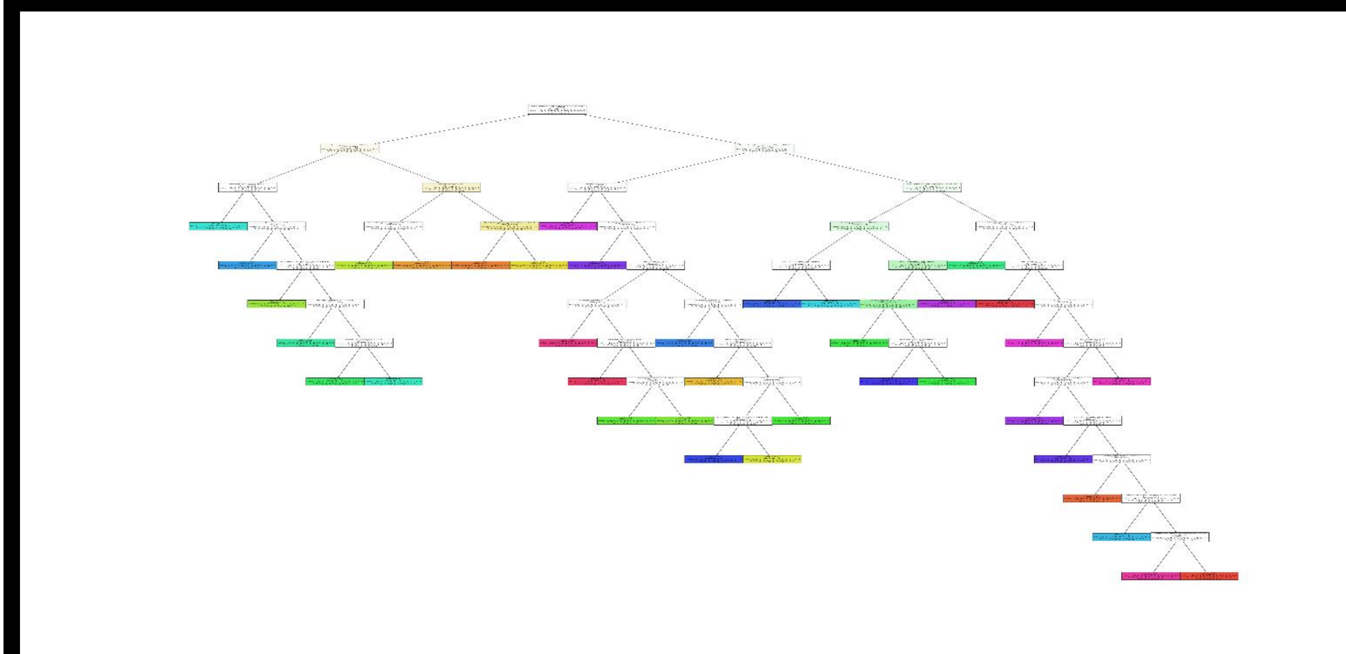
```
Petrol_tax  Average_income  Paved_Highways  Population_Driver_licence(%)
Petrol_Consumption
0          9.00            3571            1976                         0.525
541
1          9.00            4092            1250                         0.572
524
2          9.00            3865            1586                         0.580
561
3          7.50            4870            2351                         0.529
414
4          8.00            4399             431                         0.544
410
5         10.00            5342            1333                         0.571
457
6          8.00            5319           11868                         0.451
344
7          8.00            5126            2138                         0.553
467
8          8.00            4447            8577                         0.529
464
9          7.00            4512            8507                         0.552
498
10         8.00            4391            5939                         0.530
580
11         7.50            5126           14186                         0.525
471
12         7.00            4817            6930                         0.574
525
13         7.00            4207            6580                         0.545
508
14         7.00            4332            8159                         0.608
566
15         7.00            4318           10340                         0.586
635
16         7.00            4206            8508                         0.572
603
17         7.00            3718            4725                         0.540
714
18         7.00            4716            5915                         0.724
865
19         8.50            4341            6010                         0.677
640
20         7.00            4593            7834                         0.663
649
21         8.00            4983             602                         0.602
540
22         9.00            4897            2449                         0.511
464
23         9.00            4258            4686                         0.517
547
24         8.50            4574            2619                         0.551
460
```

```
25         9.00            3721            4746                           0.544
566
26         8.00            3448            5399                           0.548
577
27         7.50            3846            9061                           0.579
631
28         8.00            4188            5975                           0.563
574
29         9.00            3601            4650                           0.493
534
30         7.00            3640            6905                           0.518
571
31         7.00            3333            6594                           0.513
554
32         8.00            3063            6524                           0.578
577
33         7.50            3357            4121                           0.547
628
34         8.00            3528            3495                           0.487
487
35         6.58            3802            7834                           0.629
644
36         5.00            4045           17782                           0.566
640
37         7.00            3897            6385                           0.586
704
38         8.50            3635            3274                           0.663
648
39         7.00            4345            3905                           0.672
968
40         7.00            4449            4639                           0.626
587
41         7.00            3656            3985                           0.563
699
42         7.00            4300            3635                           0.603
632
43         7.00            3745            2611                           0.508
591
44         6.00            5215            2302                           0.672
782
45         9.00            4476            3942                           0.571
510
46         7.00            4296            4083                           0.623
610
47         7.00            5002            9794                           0.593
524
    Petrol_tax  Average_income  Paved_Highways  Population_Driver_licence(%)
0         9.00            3571            1976                           0.525
1         9.00            4092            1250                           0.572
2         9.00            3865            1586                           0.580
3         7.50            4870            2351                           0.529
4         8.00            4399             431                           0.544
5        10.00            5342            1333                           0.571
6         8.00            5319           11868                           0.451
7         8.00            5126            2138                           0.553
8         8.00            4447            8577                           0.529
9         7.00            4512            8507                           0.552
10        8.00            4391            5939                           0.530
```

```
11          7.50          5126          14186                  0.525
12          7.00          4817           6930                  0.574
13          7.00          4207           6580                  0.545
14          7.00          4332           8159                  0.608
15          7.00          4318          10340                  0.586
16          7.00          4206           8508                  0.572
17          7.00          3718           4725                  0.540
18          7.00          4716           5915                  0.724
19          8.50          4341           6010                  0.677
20          7.00          4593           7834                  0.663
21          8.00          4983            602                  0.602
22          9.00          4897           2449                  0.511
23          9.00          4258           4686                  0.517
24          8.50          4574           2619                  0.551
25          9.00          3721           4746                  0.544
26          8.00          3448           5399                  0.548
27          7.50          3846           9061                  0.579
28          8.00          4188           5975                  0.563
29          9.00          3601           4650                  0.493
30          7.00          3640           6905                  0.518
31          7.00          3333           6594                  0.513
32          8.00          3063           6524                  0.578
33          7.50          3357           4121                  0.547
34          8.00          3528           3495                  0.487
35          6.58          3802           7834                  0.629
36          5.00          4045          17782                  0.566
37          7.00          3897           6385                  0.586
38          8.50          3635           3274                  0.663
39          7.00          4345           3905                  0.672
40          7.00          4449           4639                  0.626
41          7.00          3656           3985                  0.563
42          7.00          4300           3635                  0.603
43          7.00          3745           2611                  0.508
44          6.00          5215           2302                  0.672
45          9.00          4476           3942                  0.571
46          7.00          4296           4083                  0.623
47          7.00          5002           9794                  0.593
(38, 4)
(38,)
(10, 4)
(10,)
0.0      %
1.0
```



```
[[0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

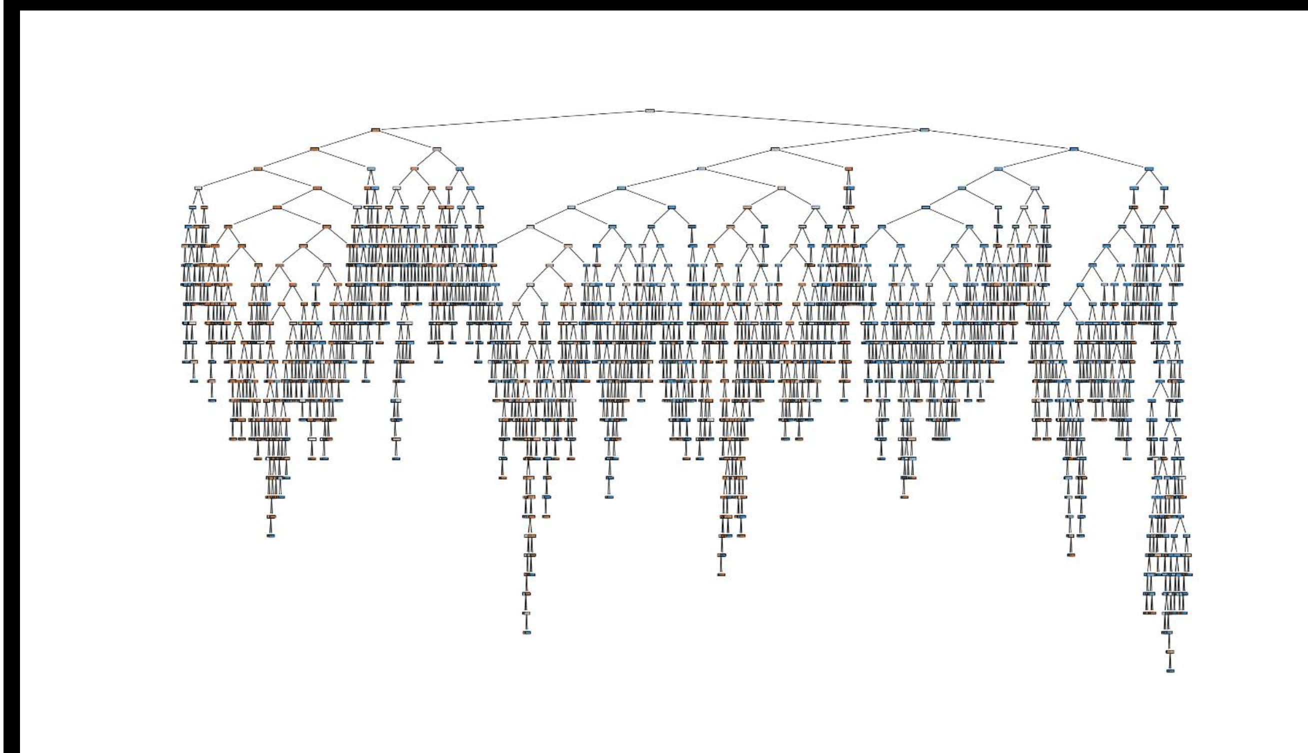


```
       age          job  marital  education default  balance  ... duration
campaign pdays  previous poutcome  deposit
0        59        admin.  married  secondary      no     2343  ...     1042
1     -1          0  unknown      yes
1        56        admin.  married  secondary      no       45  ...     1467
1     -1          0  unknown      yes
2        41   technician  married  secondary      no     1270  ...     1389
1     -1          0  unknown      yes
3        55      services  married  secondary      no     2476  ...      579
1     -1          0  unknown      yes
4        54        admin.  married   tertiary      no      184  ...      673
2     -1          0  unknown      yes
...     ...          ...       ...         ...      ...      ...  ...      ...
...   ...        ...       ...        ...
11157    33  blue-collar   single    primary      no        1  ...      257
1     -1          0  unknown       no
```

```
11158   39     services  married  secondary      no      733  ...       83
4    -1          0  unknown       no
11159   32   technician   single  secondary      no       29  ...      156
2    -1          0  unknown       no
11160   43   technician  married  secondary      no        0  ...        9
2   172          5  failure       no
11161   34   technician  married  secondary      no        0  ...      628
1    -1          0  unknown       no

[11162 rows x 17 columns]
any null values age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
deposit      0
dtype: int64

management      :  1301
blue-collar     :   708
technician      :   840
admin.          :   631
services        :   369
retired         :   516
self-employed   :   187
student         :   269
unemployed      :   202
entrepreneur    :   123
housemaid       :   109
unknown         :    34
bank_data_job_value_counts management       2566
blue-collar      1944
technician       1823
admin.           1334
services          923
retired           778
self-employed     405
student           360
unemployed        357
entrepreneur      328
housemaid         274
unknown            70
Name: job, dtype: int64
bank_data_job_poutcome unknown    8326
failure    1228
success    1071
other       537
```

```
Name: poutcome, dtype: int64
       age  job  marital  education  default  balance  ...  duration  campaign
pdays  previous  poutcome  deposit
0       59    0        1          1        0     2343  ...      1042         1
-1         0        3        1
1       56    0        1          1        0       45  ...      1467         1
-1         0        3        1
2       41    9        1          1        0     1270  ...      1389         1
-1         0        3        1
3       55    7        1          1        0     2476  ...       579         1
-1         0        3        1
4       54    0        1          2        0      184  ...       673         2
-1         0        3        1
...    ...  ...      ...        ...      ...      ...  ...       ...       ...
...       ...      ...      ...
11157   33    1        2          0        0        1  ...       257         1
-1         0        3        0
11158   39    7        1          1        0      733  ...        83         4
-1         0        3        0
11159   32    9        2          1        0       29  ...       156         2
-1         0        3        0
11160   43    9        1          1        0        0  ...         9         2
172        5        0        0
11161   34    9        1          1        0        0  ...       628         1
-1         0        3        0

[11162 rows x 17 columns]
       age  job  marital  education  default  balance  ...  month  duration
campaign  pdays  previous  poutcome
0       59    0        1          1        0     2343  ...      8      1042
1     -1          0        3
1       56    0        1          1        0       45  ...      8      1467
1     -1          0        3
2       41    9        1          1        0     1270  ...      8      1389
1     -1          0        3
3       55    7        1          1        0     2476  ...      8       579
1     -1          0        3
4       54    0        1          2        0      184  ...      8       673
2     -1          0        3
...    ...  ...      ...        ...      ...      ...  ...    ...       ...
...    ...        ...      ...
11157   33    1        2          0        0        1  ...      0       257
1     -1          0        3
11158   39    7        1          1        0      733  ...      6        83
4     -1          0        3
11159   32    9        2          1        0       29  ...      1       156
2     -1          0        3
11160   43    9        1          1        0        0  ...      8         9
2    172          5        0
11161   34    9        1          1        0        0  ...      5       628
1     -1          0        3

[11162 rows x 16 columns]
shape (8929, 16)
shape (2233, 16)
shape (8929,)
shape (2233,)
Actual against prediction       Actual  Predicted
```

```
9058       0        0
3279       1        1
6502       0        0
9327       0        1
9965       0        1
...      ...      ...
6003       0        0
5606       0        0
4808       1        1
3697       1        0
9952       0        1

[2233 rows x 2 columns]
Accuracy: 77.92207792207793 %
```



Practical No: 6

Question:- Implement Random Forest on petrol consumption dataset and classification on Bank Deposit dataset taken from kaggle competitions.

Theory Explanation:- The Dataset which was provided is

1. https://drive.google.com/file/d/1OadZiYwgM96uE_jXI0kAJvvYO0mEadrJ/view?usp=sharing

2. https://drive.google.com/file/d/1nzMZ8a9CPrAq_r8FAtiyWQ8wi6MTpmaf/view?usp=sharing

The Random Forest Classifier is a set of decision trees from randomly selected subset of training set.The principle is to aggregates the votes from different decision trees to decide the final class of the test object. Also known as ensemble tree-based learning algorithm.

RandomForestRegressor() || RandomForestClassifier()

Parameters controlling the size and depth are:

- n_estimators
- criterion
- max_depth
- min_samples_split
- min_samples_leaf

Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor,RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder

# # Random Forest Regressor on petrol consumption

data=pd.read_csv('petrol_consumption.csv')
print(data)

X = data.iloc[:, :-1]
Y = data.iloc[:, -1]

X_train, X_test, Y_train, Y_test = train_test_split(X,Y)
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)

rfc = RandomForestClassifier(n_estimators=100)

rfc.fit(X_train, Y_train)

Y_predict = rfc.predict(X_train)

cm2 = confusion_matrix(Y_train, Y_predict)
score2 = rfc.score(X_test,Y_test)
print(score2)
print(cm2)

# # Random Forest classification on Bank Deposit
bank=pd.read_csv('Bank_Deposit_Data.csv')
print(bank)

bank[bank.isnull().any(axis=1)].count()

bank_data = bank.copy()
jobs = ['management','blue-
collar','technician','admin.','services','retired','self-employed','student',
'unemployed','entrepreneur','housemaid','unknown']

for j in jobs:
    print("{:15} : {:5}". format(j, len(bank_data[(bank_data.deposit == "yes")
& (bank_data.job ==j)])))

print('bank_data_job_value_counts',bank_data.job.value_counts())
print('bank_data_job_poutcome',bank_data.poutcome.value_counts())

labels = ['housing', 'default', 'loan','job', 'contact', 'marital','education',
'poutcome', 'month', 'day','deposit']
```

```
for label in labels:
    label_encoder = LabelEncoder()
    label_encoder.fit(bank_data[label])
    bank_data[label] = label_encoder.transform(bank_data[label])
print(bank_data)


X = bank_data.drop('deposit', axis=1)
y = bank_data['deposit']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)

clf = RandomForestClassifier()
model = clf.fit(X_train, y_train)

y_pred_1 = clf.predict(X_test)
print('Actual against prediction',pd.DataFrame({'Actual':y_test,
'Predicted':y_pred_1}))


print("Accuracy:",metrics.accuracy_score(y_test, y_pred_1)*100,'%')
```

Output

```
   Petrol_tax  Average_income  Paved_Highways  Population_Driver_licence(%)
Petrol_Consumption
0        9.00            3571            1976                         0.525
541
1        9.00            4092            1250                         0.572
524
2        9.00            3865            1586                         0.580
561
3        7.50            4870            2351                         0.529
414
4        8.00            4399             431                         0.544
410
5       10.00            5342            1333                         0.571
457
6        8.00            5319           11868                         0.451
344
7        8.00            5126            2138                         0.553
467
8        8.00            4447            8577                         0.529
464
9        7.00            4512            8507                         0.552
498
10       8.00            4391            5939                         0.530
580
11       7.50            5126           14186                         0.525
471
12       7.00            4817            6930                         0.574
525
13       7.00            4207            6580                         0.545
508
14       7.00            4332            8159                         0.608
566
15       7.00            4318           10340                         0.586
635
16       7.00            4206            8508                         0.572
603
17       7.00            3718            4725                         0.540
714
18       7.00            4716            5915                         0.724
865
19       8.50            4341            6010                         0.677
640
20       7.00            4593            7834                         0.663
649
21       8.00            4983             602                         0.602
540
22       9.00            4897            2449                         0.511
464
23       9.00            4258            4686                         0.517
547
24       8.50            4574            2619                         0.551
460
```

```
25         9.00            3721            4746                       0.544
566
26         8.00            3448            5399                       0.548
577
27         7.50            3846            9061                       0.579
631
28         8.00            4188            5975                       0.563
574
29         9.00            3601            4650                       0.493
534
30         7.00            3640            6905                       0.518
571
31         7.00            3333            6594                       0.513
554
32         8.00            3063            6524                       0.578
577
33         7.50            3357            4121                       0.547
628
34         8.00            3528            3495                       0.487
487
35         6.58            3802            7834                       0.629
644
36         5.00            4045           17782                       0.566
640
37         7.00            3897            6385                       0.586
704
38         8.50            3635            3274                       0.663
648
39         7.00            4345            3905                       0.672
968
40         7.00            4449            4639                       0.626
587
41         7.00            3656            3985                       0.563
699
42         7.00            4300            3635                       0.603
632
43         7.00            3745            2611                       0.508
591
44         6.00            5215            2302                       0.672
782
45         9.00            4476            3942                       0.571
510
46         7.00            4296            4083                       0.623
610
47         7.00            5002            9794                       0.593
524
(36, 4)
(36,)
(12, 4)
(12,)
0.0
[[1 0 0 ... 0 0 0]
 [0 1 0 ... 0 0 0]
 [0 0 1 ... 0 0 0]
 ...
 [0 0 0 ... 1 0 0]
 [0 0 0 ... 0 1 0]
 [0 0 0 ... 0 0 1]]
```

```
       age          job  marital  education default  balance  ... duration
campaign pdays  previous poutcome  deposit
0       59       admin.  married  secondary      no     2343  ...     1042
1    -1         0  unknown      yes
1       56       admin.  married  secondary      no       45  ...     1467
1    -1         0  unknown      yes
2       41   technician  married  secondary      no     1270  ...     1389
1    -1         0  unknown      yes
3       55     services  married  secondary      no     2476  ...      579
1    -1         0  unknown      yes
4       54       admin.  married   tertiary      no      184  ...      673
2    -1         0  unknown      yes
...    ...          ...      ...        ...     ...      ...  ...      ...
...   ...       ...      ...      ...
11157   33  blue-collar   single    primary      no        1  ...      257
1    -1         0  unknown       no
11158   39     services  married  secondary      no      733  ...       83
4    -1         0  unknown       no
11159   32   technician   single  secondary      no       29  ...      156
2    -1         0  unknown       no
11160   43   technician  married  secondary      no        0  ...        9
2   172         5  failure       no
11161   34   technician  married  secondary      no        0  ...      628
1    -1         0  unknown       no

[11162 rows x 17 columns]
management       :  1301
blue-collar      :   708
technician       :   840
admin.           :   631
services         :   369
retired          :   516
self-employed    :   187
student          :   269
unemployed       :   202
entrepreneur     :   123
housemaid        :   109
unknown          :    34
bank_data_job_value_counts management       2566
blue-collar      1944
technician       1823
admin.           1334
services          923
retired           778
self-employed     405
student           360
unemployed        357
entrepreneur      328
housemaid         274
unknown            70
Name: job, dtype: int64
bank_data_job_poutcome unknown    8326
failure    1228
success    1071
other       537
Name: poutcome, dtype: int64
       age  job  marital  education  default  balance  ...  duration  campaign
pdays  previous  poutcome  deposit
```

```
0        59    0         1          1         0      2343  ...       1042          1
-1          0         3         1
1        56    0         1          1         0        45  ...       1467          1
-1          0         3         1
2        41    9         1          1         0      1270  ...       1389          1
-1          0         3         1
3        55    7         1          1         0      2476  ...        579          1
-1          0         3         1
4        54    0         1          2         0       184  ...        673          2
-1          0         3         1
...     ...  ...       ...        ...       ...       ...  ...        ...        ...
...        ...       ...       ...
11157    33    1         2          0         0         1  ...        257          1
-1          0         3         0
11158    39    7         1          1         0       733  ...         83          4
-1          0         3         0
11159    32    9         2          1         0        29  ...        156          2
-1          0         3         0
11160    43    9         1          1         0         0  ...          9          2
172          5         0         0
11161    34    9         1          1         0         0  ...        628          1
-1          0         3         0

[11162 rows x 17 columns]
Actual against prediction       Actual  Predicted
9058        0          0
3279        1          1
6502        0          0
9327        0          1
9965        0          1
...       ...        ...
6003        0          0
5606        0          0
4808        1          1
3697        1          0
9952        0          0

[2233 rows x 2 columns]
Accuracy: 84.10210479175997 %
```

Practical No: 7

Question:- Implement Logistic Regression on Heart Disease Prediction dataset to predict the 10 year risk of Coronary heart disease (Dataset taken from kaggle competition).

Theory Explanation:- The Dataset which was provided is

1. https://drive.google.com/file/d/1YLxN-USbFyM5xuyiKkXxj0LXOQGeGvcf/view?usp=sharing

Logistic regression is a classification algorithm. It is used to predict a binary outcome based on a set of independent variables. A **binary outcome** is one where there are only two possible scenarios—either the event happens (1) or it does not happen (0). **Independent variables** are those variables or factors which may influence the outcome (or dependent variable).

The independent variables can fall into any of the following categories:

- **Continuous**
- **Discrete, ordinal**
- **Discrete, nominal**

The three types of logistic regression are:

- **Binary logistic regression**
- **Multinomial logistic regression**
- **Ordinal logistic regression**

Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn import metrics

# # Logistic Regression on Heart Disease Prediction dataset
ds=pd.read_csv('framingham.csv')
ds.head()
print(ds)

X=ds.iloc[:,:-1].values.astype('int')
Y=ds.iloc[:,-1].values.astype('int')

from sklearn.preprocessing import StandardScaler
SS_Features = StandardScaler()
X_SSF=SS_Features.fit_transform(X)

from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X_SSF, Y, test_size=0.2,
random_state=0)
print(X_train)

from sklearn.linear_model import LogisticRegression
LOR = LogisticRegression()
LOR.fit(X_train,Y_train)
y_pred=LOR.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),Y_test.reshape(len(Y_test),1))
,1))

from sklearn.metrics import confusion_matrix
print('confusion_matrix:',confusion_matrix(Y_test,y_pred))

from sklearn.metrics import accuracy_score
print('Accuracy:',accuracy_score(Y_test,y_pred)*100,'%')
```

Output

```
      male  age  education  currentSmoker  cigsPerDay  BPMeds  ...  sysBP  diaBP
BMI  heartRate  glucose  TenYearCHD
0        1   39        4.0              0         0.0     0.0  ...  106.0   70.0
26.97        80.0     77.0           0
1        0   46        2.0              0         0.0     0.0  ...  121.0   81.0
28.73        95.0     76.0           0
2        1   48        1.0              1        20.0     0.0  ...  127.5   80.0
25.34        75.0     70.0           0
3        0   61        3.0              1        30.0     0.0  ...  150.0   95.0
28.58        65.0    103.0           1
4        0   46        3.0              1        23.0     0.0  ...  130.0   84.0
23.10        85.0     85.0           0
...    ...  ...        ...            ...         ...     ...  ...    ...    ...
...        ...      ...          ...
4233     1   50        1.0              1         1.0     0.0  ...  179.0   92.0
25.97        66.0     86.0           1
4234     1   51        3.0              1        43.0     0.0  ...  126.5   80.0
19.71        65.0     68.0           0
4235     0   48        2.0              1        20.0     NaN  ...  131.0   72.0
22.00        84.0     86.0           0
4236     0   44        1.0              1        15.0     0.0  ...  126.5   87.0
19.16        86.0      NaN           0
4237     0   52        2.0              0         0.0     0.0  ...  133.5   83.0
21.47        80.0    107.0           0

[4238 rows x 16 columns]
[[ 1.1531919   1.5651408   0.15939032 ...  0.06710768  0.01536279
   0.31745749]
 [ 1.1531919   0.28176554  0.15939032 ...  0.06710762  0.01536251
   0.31745742]
 [ 1.1531919  -1.35162116  0.15939032 ...  0.06710766  0.01536282
   0.31745742]
 ...
 [ 1.1531919  -1.23495068  0.15939032 ...  0.06710764  0.01536264
   0.31745743]
 [-0.86715836  0.86511793  0.15939032 ...  0.06710764  0.01536279
   0.31745743]
 [-0.86715836 -1.11828021  0.15939032 ...  0.06710766  0.01536279
   0.31745742]]
[[0 0]
 [0 0]
 [0 0]
 ...
 [0 0]
 [0 0]
 [0 0]]
confusion_matrix: [[707   3]
 [132   6]]
Accuracy: 84.08018867924528 %
```

Practical No: 8

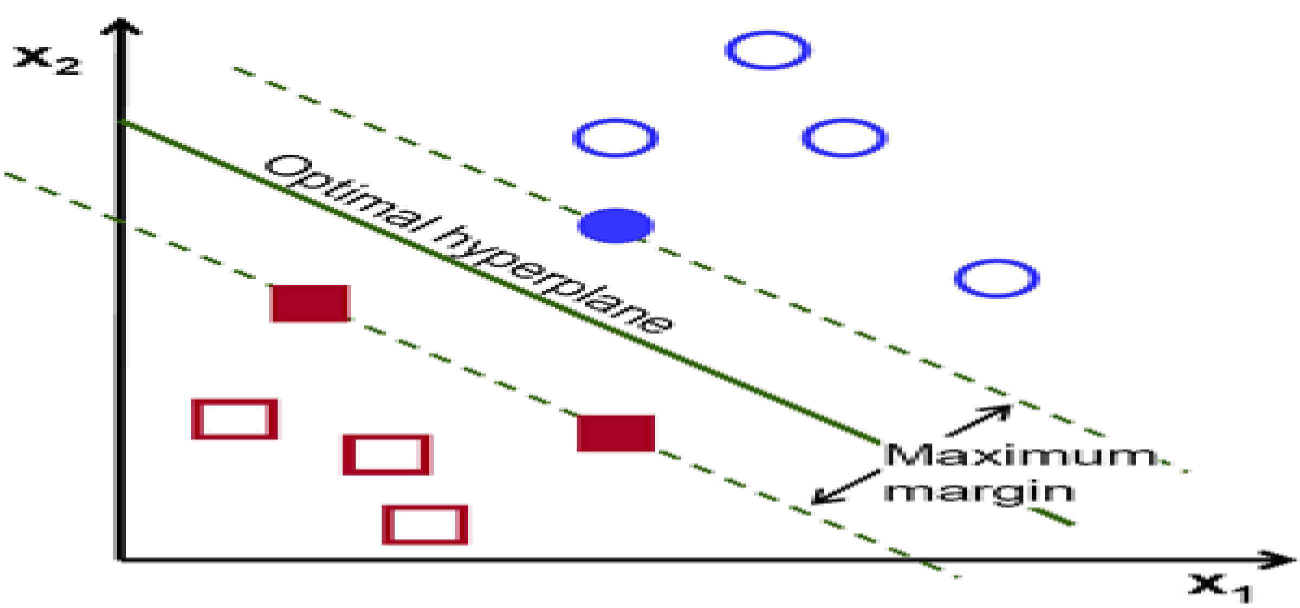
Question:- Implement Support Vector Machine on Credit Scoring Dataset taken from Kaggle Competition. Here “SeriousDlqin2yrs” is the dependent variable.

Theory Explanation:- The Dataset which was provided is

1. https://drive.google.com/file/d/1zZYNTHaTIr3xKzXvpPfwaYM1qPYYccRo/view?usp=sharing

What is Support Vector Machine?

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.



Types of Kernel in SVM:-

- **Gaussian Kernel**
- **Sigmoid Kernel**
- **Polynomial Kernel**
- **Linear Kernel**
- **Polynomial Kernel**

Code

```python
# # Support Vector Machine implementation on Credit Scoring Sample
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt


data = pd.read_csv("credit_scoring_sample.csv")
print(data)

print(data.corr())
print(data.isnull().sum())
print(data.dropna(inplace=True))
print(data.isnull().sum())

X = data.drop('SeriousDlqin2yrs', axis=1)
Y = data['SeriousDlqin2yrs']

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)

from sklearn.svm import SVC
from sklearn import metrics
svc=SVC()
svc.fit(X_train,y_train)

y_pred=svc.predict(X_test)

print('Accuracy Score:',metrics.accuracy_score(y_test,y_pred)*100,'%')
print(y_pred)
```

Output

```
       SeriousDlqin2yrs  age  ...  MonthlyIncome  NumberOfDependents
0                     0   64  ...         8158.0                 0.0
1                     0   58  ...            NaN                 0.0
2                     0   41  ...         6666.0                 0.0
3                     0   43  ...        10500.0                 2.0
4                     1   49  ...          400.0                 0.0
...                 ...  ...  ...            ...                 ...
45058                 1   31  ...         3000.0                 1.0
45059                 0   49  ...            0.0                 5.0
45060                 1   38  ...         3000.0                 2.0
45061                 0   47  ...        11720.0                 5.0
45062                 1   45  ...         9120.0                 2.0

[45063 rows x 8 columns]
                                      SeriousDlqin2yrs       age  ...
MonthlyIncome  NumberOfDependents
SeriousDlqin2yrs                              1.000000 -0.192937  ...       -
0.035469            0.075360
age                                          -0.192937  1.000000  ...
0.051417           -0.203924
NumberOfTime30-59DaysPastDueNotWorse          0.141638 -0.076529  ...       -
0.017516           -0.011138
DebtRatio                                    -0.012344  0.028774  ...       -
0.032343           -0.032297
NumberOfTimes90DaysLate                       0.131774 -0.077381  ...       -
0.020486           -0.016762
NumberOfTime60-89DaysPastDueNotWorse          0.114938 -0.072104  ...       -
0.018203           -0.017994
MonthlyIncome                                -0.035469  0.051417  ...
1.000000            0.055916
NumberOfDependents                            0.075360 -0.203924  ...
0.055916            1.000000

[8 rows x 8 columns]
SeriousDlqin2yrs                           0
age                                        0
NumberOfTime30-59DaysPastDueNotWorse       0
DebtRatio                                  0
NumberOfTimes90DaysLate                    0
NumberOfTime60-89DaysPastDueNotWorse       0
MonthlyIncome                           8643
NumberOfDependents                      1117
dtype: int64
None
SeriousDlqin2yrs                        0
age                                     0
NumberOfTime30-59DaysPastDueNotWorse    0
DebtRatio                               0
NumberOfTimes90DaysLate                 0
NumberOfTime60-89DaysPastDueNotWorse    0
MonthlyIncome                           0
NumberOfDependents                      0
dtype: int64
Accuracy Score:
0.8267435475013729
```

```
[1 0 0 ... 0 0 0]

C:\Users\RAZU\Desktop\New folder\python>S.py
       SeriousDlqin2yrs  age  ...  MonthlyIncome  NumberOfDependents
0                     0   64  ...         8158.0                 0.0
1                     0   58  ...            NaN                 0.0
2                     0   41  ...         6666.0                 0.0
3                     0   43  ...        10500.0                 2.0
4                     1   49  ...          400.0                 0.0
...                 ...  ...  ...            ...                 ...
45058                 1   31  ...         3000.0                 1.0
45059                 0   49  ...            0.0                 5.0
45060                 1   38  ...         3000.0                 2.0
45061                 0   47  ...        11720.0                 5.0
45062                 1   45  ...         9120.0                 2.0

[45063 rows x 8 columns]
                                      SeriousDlqin2yrs       age  ...
MonthlyIncome  NumberOfDependents
SeriousDlqin2yrs                              1.000000 -0.192937  ...       -
0.035469            0.075360
age                                          -0.192937  1.000000  ...
0.051417           -0.203924
NumberOfTime30-59DaysPastDueNotWorse          0.141638 -0.076529  ...       -
0.017516           -0.011138
DebtRatio                                    -0.012344  0.028774  ...       -
0.032343           -0.032297
NumberOfTimes90DaysLate                       0.131774 -0.077381  ...       -
0.020486           -0.016762
NumberOfTime60-89DaysPastDueNotWorse          0.114938 -0.072104  ...       -
0.018203           -0.017994
MonthlyIncome                                -0.035469  0.051417  ...
1.000000            0.055916
NumberOfDependents                            0.075360 -0.203924  ...
0.055916            1.000000

[8 rows x 8 columns]
SeriousDlqin2yrs                           0
age                                        0
NumberOfTime30-59DaysPastDueNotWorse       0
DebtRatio                                  0
NumberOfTimes90DaysLate                    0
NumberOfTime60-89DaysPastDueNotWorse       0
MonthlyIncome                           8643
NumberOfDependents                      1117
dtype: int64
None
SeriousDlqin2yrs                        0
age                                     0
NumberOfTime30-59DaysPastDueNotWorse    0
DebtRatio                               0
NumberOfTimes90DaysLate                 0
NumberOfTime60-89DaysPastDueNotWorse    0
MonthlyIncome                           0
NumberOfDependents                      0
dtype: int64
Accuracy Score: 82.6743547501373 %
[1 0 0 ... 0 0 0]
```

Practical No: 9

Question:- Implement K-Nearest Neighbour on Plant-Texture Dataset taken from Kaggle competition.

Theory Explanation:- The Dataset which was provided is

- https://drive.google.com/file/d/188pokuwa_LGASXsZENhvANfmQhQ4xw0e/view?usp=sharing

K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry. The following two properties would define KNN well −

- **Lazy learning algorithm** − KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.
- **Non-parametric learning algorithm** − KNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data.

Code

```
# # K-Nearest Neighbour on Plant-Texture

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

data = pd.read_csv("Plant_Texture.csv ")
print(data)
print(data.corr())
print(data.isnull().sum())

X=data.iloc[:, :-1]
Y =data.iloc[:,-1]

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
D,NN=knn.kneighbors(X_test,return_distance=True)
print("Distances",D)
print("Index",NN)

y_pred = knn.predict(X_test)
print(y_pred)

Graph = knn.kneighbors_graph(X_test,3,mode="connectivity")
Graph.toarray()
print(Graph)

plt.spy(Graph,precision=0.1, markersize=1)
plt.show()

from sklearn import metrics
print("Accuracy:",(metrics.accuracy_score(y_test, y_pred)*100),"%")
```

Output

```
Distances [[4.24316874 5.06503582 5.25445644]
 [2.33829968 2.44617944 3.42691743]
 [3.03217255 3.07034641 3.49865431]
 [2.91487161 2.96438355 3.26746158]
 [3.992196   4.57816698 4.61468286]
 [4.7267315  7.07095891 7.32692247]
 [3.01068319 4.26805036 4.39736052]
 [4.98060554 5.65255152 5.90353835]
 [3.26428722 3.80698454 4.07175936]
 [3.13955255 3.15609228 3.84926752]
 [3.30967898 3.68537532 4.23789959]
 [5.0776127  5.27276068 5.29232056]
 [2.90230742 3.37349423 3.86912544]
 [2.90566686 4.13182675 5.59843822]
 [2.47609881 3.63369903 3.8626991 ]
 [2.67837785 3.17023693 3.48962187]
 [3.30678658 3.36112726 3.61591171]
 [2.69262021 3.41584186 4.38011643]
 [4.03490572 4.21688651 4.41435709]
 [2.02927866 2.2673015  2.48407447]
 [3.32686545 3.83949169 4.01060653]
 [4.53834292 4.77788134 5.05852737]
 [4.22725317 4.52110611 5.63322778]
 [5.28860975 5.62681457 5.76797077]
 [3.18769875 3.41423096 3.91908669]
 [2.58459959 2.92763682 3.45822856]
 [2.19448657 4.20421619 4.21470631]
 [4.247792   5.09432391 5.76792787]
 [3.39818466 3.63021633 3.7856171 ]
 [3.09322008 3.40383131 3.961787  ]
 [4.01792415 4.77861765 4.92998992]
 [2.72975281 3.36324194 3.50898395]
 [5.96854058 7.32945037 7.69147178]
 [3.56108758 5.38599418 5.53168996]
 [3.26847232 4.44350212 4.69992346]
 [2.67025126 4.1562485  4.37696004]
 [3.9982078  4.28168143 4.79825827]
 [4.59917446 4.63546116 4.7013334 ]
 [0.75810417 1.33675641 1.66782473]
 [3.93377083 4.10211448 4.22054707]
 [5.04831827 5.48606291 7.53187044]
 [3.2976365  3.68299019 3.73324007]
 [2.54487778 2.6314135  2.67393102]
 [2.28275208 2.41139867 2.53933997]
 [2.56492534 3.30887763 3.9667599 ]
 [3.74549507 4.0970656  4.09855892]
 [2.99421746 3.6828013  3.93698799]
 [3.78766909 4.15781754 4.23805887]
 [2.41491719 2.46713462 2.55849966]
 [3.39170072 3.4935194  4.12668631]
 [2.61803623 2.68476777 2.81332932]
 [2.71640371 5.41818808 5.46082984]
 [2.91956482 3.37989617 4.50806099]
 [2.56518689 4.01336872 5.34259578]
 [4.3343313  4.65589735 4.71378073]
```

```
 [2.86568874 2.97629379 5.07174688]
 [2.35350154 3.07835563 3.08704083]
 [4.06148667 4.42412546 4.5181136 ]
 [3.02899492 3.30404962 3.91292394]
 [3.06788822 3.17624594 3.23931719]
 [1.78652563 2.13905934 3.3803458 ]
 [6.15378073 6.20533926 6.57778116]
 [3.3974265  3.95110454 4.00276332]
 [2.97905243 3.4331755  3.60315915]
 [3.68329489 4.16037353 4.70988711]
 [3.12675279 3.5659866  3.88715621]
 [3.02420377 3.20034413 3.52159842]
 [4.11126906 4.41783684 4.45633126]
 [3.59414916 3.99623289 4.16015902]
 [5.85323435 5.98664063 6.045317  ]
 [6.1775263  6.71404151 7.09937787]
 [3.34590793 3.41968025 3.42653909]
 [2.89906474 3.50053038 3.93454816]
 [2.38560679 2.97599265 3.07657762]
 [4.12081169 4.34100276 4.39444162]
 [3.53996821 4.01117101 4.59576618]
 [2.67744371 3.56729611 3.62004619]
 [3.28534079 3.44152114 3.92927422]
 [4.23808643 4.24229966 4.73108518]
 [4.24615779 5.12999057 5.14278774]
 [3.26149431 3.41799297 3.55109209]
 [4.0910185  4.83883638 4.84410421]
 [2.77001223 3.38904968 3.4223512 ]
 [3.71120715 3.90839067 4.26648524]
 [3.82277025 4.27824771 4.31216357]
 [3.17032753 4.14888238 5.0500271 ]
 [2.11280715 2.18630537 2.70119843]
 [2.74187938 3.28966402 3.64287264]
 [3.20364501 3.49619422 3.71756684]
 [5.59551716 6.13637912 6.14927513]
 [2.37918829 2.61467095 2.69770342]
 [2.36476811 2.56227471 2.68756008]
 [6.21857099 6.21967901 6.63579843]
 [3.12520629 3.20730551 4.21235984]
 [3.40487371 3.81209555 3.96407983]
 [3.64752178 3.78785954 4.21186147]
 [6.24763827 6.41142172 6.88871286]
 [3.26810312 3.7697974  3.81059984]
 [3.14691282 4.17367449 4.21154295]
 [5.06550962 5.09649652 6.12143316]
 [4.00722384 4.78208537 5.08421798]
 [3.9091767  3.95409756 4.07573628]
 [2.49134687 4.48175809 5.13549224]
 [3.03627097 4.2559922  4.27957673]
 [2.96510836 3.6448364  3.71282483]
 [4.23670454 5.12245347 5.44515883]
 [4.37419934 4.73144738 5.25463692]
 [3.50265792 4.44552447 4.89677892]
 [4.17919815 4.39614487 4.53159756]
 [3.85221978 3.95847051 4.65449392]
 [1.82396059 2.37299788 2.58206312]
 [3.27841713 3.67956619 3.82599949]
 [3.77157782 3.99402501 4.38172302]
```

```
[3.13380803 3.29727412 3.64216696]
[5.15688903 6.04425338 6.06666322]
[2.63512638 2.78453611 3.21454844]
[3.2069486  3.49781614 3.82233183]
[4.33133561 5.192732   5.23792549]
[5.58210272 7.64609762 9.13841879]
[6.03589726 7.95110358 7.97401761]
[3.05991005 3.52922332 3.65256004]
[3.45316396 3.51569798 3.59232276]
[3.88621257 4.45074336 4.55537373]
[2.44137251 2.59594969 2.66722408]
[3.83877393 3.88806366 4.00791318]
[3.51828825 3.9164769  3.92552917]
[4.19325805 4.71655679 5.12474419]
[4.05798559 4.35029711 4.36024796]
[3.49565912 3.82357626 3.93385453]
[3.73527492 3.8780365  4.55872252]
[4.2834719  4.38086942 4.42516974]
[2.8087324  2.86077075 2.97465848]
[4.65509928 4.76213156 4.85118764]
[5.69860807 6.46929275 6.52238396]
[4.32822367 5.5812683  5.76673607]
[2.23968851 2.73802267 3.6368149 ]
[3.2530314  4.17115765 4.21233641]
[2.85566557 3.8395847  5.05476418]
[3.06640238 3.7380387  3.74427976]
[5.42755344 5.49552619 5.74457157]
[3.7210172  3.87289387 4.14940032]
[4.04407194 4.49667195 4.78788302]
[4.70492446 6.02799458 6.10063895]
[2.60806048 3.00810259 3.08839449]
[3.25878617 4.33621265 4.53580376]
[2.48894227 3.88790393 4.07119636]
[4.52288362 4.56882303 4.66326415]
[3.25550884 3.45689763 3.51601844]
[4.5272772  4.99497592 5.96570501]
[2.841396   2.84453214 4.18579943]
[3.39317046 3.46048656 4.22335794]
[2.65618412 3.92945118 5.26732675]
[3.30176557 3.49986676 3.6229941 ]
[2.79688475 3.02920237 4.24436616]
[3.210927   4.0564877  4.08775831]
[2.90642036 3.83727135 4.06567129]
[2.5910389  2.79367197 2.83655161]
[6.28074616 6.51137891 6.5311614 ]
[4.46194187 4.47358746 4.47624332]
[4.8763657  5.30094362 5.36011926]
[3.35373846 4.235804   4.26961031]
[3.72161288 4.53918008 4.58753292]
[3.09234373 3.7562246  4.01567228]
[5.10300646 5.32893422 5.91474046]
[3.59521681 3.66001017 3.79915831]
[2.64286008 3.48899146 3.90655489]
[3.13556862 3.19301977 4.42317777]
[5.43957305 5.50911168 6.54673014]
[2.39305226 4.10988206 4.14195963]
[3.31697675 3.61920985 4.86996147]
[4.95741058 4.9602533  4.98164658]
```

```
 [4.2139609  5.38379756 6.38628022]
 [3.62545663 3.79793368 4.03132407]
 [5.34474093 5.78259518 5.98374442]
 [2.12048549 3.45500912 4.67215616]
 [2.22383421 3.22265664 3.59142197]
 [2.29154869 2.77117187 3.33059252]
 [4.16077294 4.36949686 4.61844407]
 [2.63858815 3.49678506 3.6790561 ]
 [3.31944977 4.06592081 4.27583085]
 [6.26526412 6.41345626 7.4638262 ]
 [3.12002417 3.57718912 3.67171688]
 [2.86275674 3.66517641 4.24245952]
 [1.80951402 2.20316823 2.61360394]
 [2.82576181 3.28060958 3.35983219]
 [3.37228374 3.54492324 3.82873459]
 [4.52827134 4.62293488 4.73622145]
 [3.8458886  4.76484864 5.36928271]
 [4.2876073  4.70870958 4.96384418]
 [4.08676833 4.75976079 4.89134677]
 [2.69231864 2.8995494  2.95961558]
 [3.87449655 4.1006165  4.12879644]
 [3.59916637 3.60558573 3.72751893]
 [6.24139081 6.61949939 7.48742622]
 [2.35451747 2.76845218 3.00123077]
 [3.85460947 3.91142039 3.95626633]
 [3.39452563 3.77673549 3.89691992]
 [4.51345096 4.78889257 4.81239779]
 [3.03391563 3.78811292 3.79954741]
 [3.39099018 4.61211641 4.96481595]
 [1.91474436 2.73572506 4.13212427]
 [2.59798583 2.86858643 2.97051799]
 [2.83368297 4.66002912 5.140336  ]
 [2.80266967 2.87618195 3.18342726]
 [3.5664361  3.79862221 3.8746207 ]
 [2.28099941 3.39467659 4.08330808]
 [4.22805236 4.43153517 4.56416994]
 [3.73612561 4.0080434  4.06318512]
 [3.86347126 4.35931578 4.36917113]
 [2.53795568 2.73902809 3.51155776]
 [2.70789796 3.53511082 3.67932358]
 [2.29084378 2.70440713 3.22063294]
 [2.31802103 3.46746808 4.08646661]
 [4.25012515 4.7754582  6.4056143 ]
 [2.95058587 3.90297589 3.90457738]
 [5.97177301 6.30489637 6.514525  ]
 [3.38590627 3.81766724 4.06763269]
 [3.7689926  4.39533683 4.44756866]
 [3.11585661 3.73416123 3.90053948]
 [2.15412504 2.53837431 3.35994447]
 [2.5963447  2.78806144 3.42366871]
 [2.8211904  3.82601495 4.8312813 ]
 [3.03401481 3.93194976 4.14683095]
 [3.72327965 3.9904541  4.006254  ]
 [3.53873894 4.88677109 4.91894246]
 [4.88848059 5.16856864 5.56177858]
 [3.64627244 4.40830191 4.60176585]
 [3.55879181 3.87287074 4.06496044]
 [4.82384111 5.28293752 6.35668988]
```

```
[3.13556819 3.74965376 3.84198012]
[2.78994835 3.42976617 3.61689404]
[6.29270775 6.30378773 7.26420195]
[6.05015575 6.22069316 6.33205425]
[4.00471714 4.66572067 5.00437371]
[2.67159071 3.15547883 3.26231654]
[3.32611894 3.55580612 3.6189532 ]
[3.45374681 4.06550793 4.34027683]
[3.05554679 3.89023155 4.31984924]
[3.23081867 3.39067903 4.10283657]
[3.55047902 3.81343161 4.08624214]
[3.10208771 3.45812186 3.46223967]
[3.13929231 3.17015222 3.2685844 ]
[2.97741666 3.16141322 3.23646525]
[5.30540252 6.03238408 6.08666071]
[3.15279904 3.20519741 3.78691501]
[3.69454477 4.2765228  4.45996486]
[2.43890533 3.88013198 4.07871683]
[2.36488512 2.77713732 2.87266885]
[3.35865939 4.22993994 4.32332472]
[4.23422683 4.73885394 4.99849716]
[2.62223592 2.66743028 2.87631339]
[3.47935417 3.60570027 3.61052772]
[2.94524564 3.03889762 3.61572747]
[3.7617074  4.09111251 4.22597092]
[4.88325876 5.27737286 5.39301032]
[3.45738839 3.59786215 3.6560966 ]
[2.82175561 3.68155972 3.85374706]
[6.66526773 7.12343012 7.21471649]
[2.52162559 3.078721   3.56002705]
[3.81749966 3.95634272 4.2952942 ]
[2.93106192 3.74041247 3.95490319]
[3.94788832 4.97127524 5.23378801]
[4.14013848 4.68709961 5.29501762]
[3.31835178 3.67703791 4.24704265]
[2.81335789 3.07756014 3.08181803]
[3.99005741 4.81260201 4.92559595]
[4.51855432 4.53209081 4.55144729]
[4.959618   5.04470087 5.12074327]
[3.45555629 3.96693124 4.13933917]
[3.85553705 4.14402767 4.50813866]
[3.2623699  3.93873765 4.44097655]
[1.80949309 1.89472554 2.07687702]
[3.77217998 3.90031692 3.94291609]
[3.79392717 4.03005689 4.5578971 ]
[4.25417724 4.27728472 4.61276581]
[2.86614487 3.00851533 3.51541038]
[2.02877023 2.54762033 2.57880387]
[3.31749847 4.37052847 4.42015975]
[3.16214806 3.37868594 3.87048845]
[3.79054444 4.63773162 5.74816377]
[4.95312321 5.07289573 5.50951204]
[5.88108077 6.43916492 6.75286933]
[4.5049954  4.58896948 4.64465124]
[4.29341744 4.63129719 4.90478424]
[5.02817951 5.05956261 5.09354474]
[1.64065036 2.29652158 2.46599663]
[7.94096271 9.05500566 9.76302789]
```

```
 [3.14044797 5.466481   5.65499728]
 [3.61864867 4.26321801 4.33209265]
 [4.2012136  4.3009503  4.3605516 ]
 [4.04466019 4.96687425 5.0859042 ]
 [3.55621498 3.71493404 4.11329223]
 [2.89608122 2.95093351 3.04444412]
 [7.24265905 8.81005006 9.23697081]
 [3.27123613 4.0451994  4.6407388 ]
 [4.40377824 4.59214731 4.6580155 ]
 [2.90411485 3.10018696 3.73391887]
 [2.70226597 3.45367187 3.88291596]
 [4.17129044 4.9500297  5.10049945]
 [3.38658819 3.60783242 3.86291351]
 [1.92792535 2.15759667 2.78833368]
 [4.54550946 4.72959288 5.24214973]
 [4.09374519 4.7278429  4.73275362]
 [2.10644972 4.58253416 4.79415156]
 [4.29301967 4.46886811 4.5009572 ]
 [3.51159994 4.0498695  5.60524628]
 [3.37175592 3.52796259 3.56320029]
 [3.4809593  3.80535622 4.01644691]
 [3.95173983 3.96461663 4.27860914]
 [2.95199    2.9693825  3.77494016]
 [3.96815274 4.21795106 4.22117304]
 [3.71394282 4.2162881  4.35335946]
 [2.75634939 3.43567616 3.92928709]
 [3.32140601 4.21377488 4.41361073]
 [3.68408993 4.02963597 4.41372609]
 [3.44294894 3.82244121 3.91985776]
 [3.21584028 3.70900432 3.97952759]
 [5.23008916 5.47434084 5.67067329]
 [3.86630827 3.89056517 3.89386788]
 [4.55743504 4.70613931 5.0511716 ]]
Index [[  12   70   30]
 [ 485  861  602]
 [ 285 1024  135]
 [ 292  542 1236]
 [ 134 1179  859]
 [ 934  771  492]
 [ 370 1032 1197]
 [ 844 1089  175]
 [ 218  584 1048]
 [1031  841    6]
 [   1   95  158]
 [ 708 1022 1009]
 [ 789 1074  484]
 [ 516   15  419]
 [  19  826  391]
 [1123  644  575]
 [ 967  724  930]
 [ 749  736  626]
 [1121  846  218]
 [ 497 1153 1262]
 [ 447  546  753]
 [ 502  967  785]
 [ 372  716   13]
 [1167    0  910]
 [ 954  390  422]
```

```
 [ 396  821  858]
 [1116  915  637]
 [ 723  823  711]
 [ 964  408 1252]
 [ 472 1252  592]
 [  22 1197  810]
 [1102  822  664]
 [ 887  836 1120]
 [1173  506 1215]
 [ 605 1219  708]
 [ 351   77 1247]
 [ 266  893  617]
 [1032 1197 1237]
 [ 693  487  868]
 [ 357  701 1200]
 [ 755  704  534]
 [ 907  112  820]
 [ 361  843 1110]
 [1206 1155  909]
 [1173  506  729]
 [  49  207  991]
 [ 146  630  232]
 [ 616  870  559]
 [ 374  188  248]
 [  98  893  918]
 [ 785  248  374]
 [ 803  118  586]
 [ 541  607  249]
 [ 203  550   70]
 [ 532  185 1238]
 [ 253  585  202]
 [1276  359 1068]
 [ 580  342 1090]
 [ 607  541  587]
 [ 395  348  898]
 [ 636  533  749]
 [ 705 1138  685]
 [  41  773 1152]
 [  23  488  858]
 [ 531   53 1043]
 [ 584   67 1140]
 [ 114  799 1015]
 [ 765  810 1032]
 [1067 1020  226]
 [ 816 1135  842]
 [1138 1202  480]
 [ 311  808  247]
 [ 534  257  190]
 [ 405   65 1273]
 [1204  899  650]
 [ 915 1116  637]
 [1125  165  577]
 [ 812  238    5]
 [ 142  760 1163]
 [ 730  688 1237]
 [ 594  554  875]
 [ 688  765 1237]
 [ 425 1136  995]
```

```
[ 921 1180 1105]
[ 311 1190  434]
[1128  681  899]
[  80  258  843]
[ 911  990  207]
[1226  170  609]
[  99  624  360]
[ 116  785  374]
[ 125  106 1139]
[1032 1197  370]
[ 894  928  208]
[ 895  466  441]
[ 655  252  899]
[ 151  760  530]
[1002  789  279]
[1261 1249  577]
[ 805  987  723]
[1235  720 1064]
[ 511  168 1099]
[ 813  829 1201]
[ 910  348 1232]
[1025  277  568]
[ 335  468  125]
[1043  343  950]
[1221  538 1087]
[   0  922  806]
[ 351 1247  176]
[1110  361  389]
[ 439  777  877]
[ 609  384  332]
[ 439  521 1122]
[ 355  211  798]
[1001  843  361]
[ 873  525  300]
[ 504  202  127]
[ 305  421  362]
[1193  310  805]
[ 276   38  307]
[ 583  410  916]
[  95    1  812]
[ 163  248  520]
[1004   96  653]
[ 387 1169  342]
[   0  864 1272]
[ 531 1043  477]
[ 447  753  546]
[ 674  577 1112]
[ 600  192  663]
[ 513  354  662]
[ 968  763 1278]
[1075  201  751]
[ 373  266  267]
[ 485  861  540]
[ 990  207  874]
[ 459  448  455]
[ 927 1109  687]
[ 453  563  730]
[1017  597 1219]
```

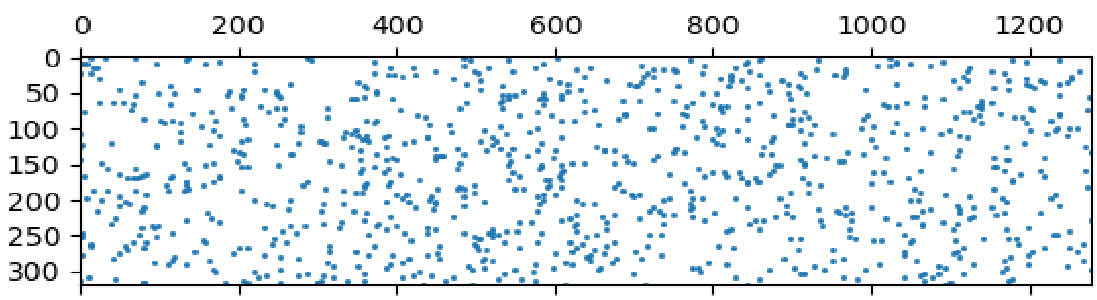
```
 [ 394 1154  944]
 [ 725  398  353]
 [ 387  447 1169]
 [1173    0  910]
 [ 315  199  691]
 [ 544  959  963]
 [ 479  676   26]
 [ 637  915  698]
 [ 621  132  168]
 [ 541  607  249]
 [ 878  884  330]
 [ 583  916  158]
 [1154  581  371]
 [1158  211  204]
 [  68 1103  355]
 [ 476  136   34]
 [ 780  390  579]
 [ 611  876 1061]
 [1215 1167 1272]
 [ 353  815  608]
 [ 383  545 1003]
 [ 833  799 1015]
 [  81  746  404]
 [  49 1189  870]
 [ 789  818  115]
 [  77 1219  605]
 [   4  555  175]
 [ 836  854  912]
 [  71  175   60]
 [1047  113   66]
 [ 593  428 1179]
 [ 607  541  587]
 [ 856  392  842]
 [1179  859  134]
 [1024  166  285]
 [ 843 1001  361]
 [ 921  763  679]
 [ 154  496  936]
 [ 134  593  359]
 [ 281  107  390]
 [ 676  479 1273]
 [ 590  358  608]
 [ 136 1055  476]
 [1155  451  549]
 [ 309  558  484]
 [ 333   15  770]
 [  33  686  131]
 [  70  110  203]
 [ 498  355  921]
 [ 898  348 1061]
 [ 600  697 1231]
 [ 584 1094  411]
 [ 400  705   82]
 [ 444   48  485]
 [ 699  577 1125]
 [ 827 1226  774]
 [1164  885  745]
 [ 785    8  408]
```

```
 [  25 1059  440]
 [ 626  736  264]
 [ 584  218 1048]
 [  56  414  509]
 [ 447  387 1169]
 [ 307  234   61]
 [ 770  634  333]
 [ 649  448  580]
 [1236  242 1109]
 [ 891  485   48]
 [ 843   80 1001]
 [1130  773 1152]
 [ 967  422  724]
 [ 263 1076 1112]
 [ 934  771  445]
 [ 955   20  719]
 [1065  731  577]
 [  76  344  303]
 [ 959  801  891]
 [ 975 1167 1214]
 [1110  361  159]
 [ 675  966 1157]
 [ 749  736  626]
 [ 392  899 1128]
 [ 880 1058  660]
 [1052  471  773]
 [1018   43  705]
 [ 165  619 1125]
 [1278  608  358]
 [ 974  864  852]
 [ 789  936  133]
 [ 329  666  570]
 [ 555 1089  175]
 [ 627  845 1065]
 [ 307   38  737]
 [1033  375 1108]
 [ 266  373   98]
 [ 624 1064  939]
 [ 399   90  283]
 [1199  437  232]
 [  92  327   76]
 [ 521  972  858]
 [1169  447  387]
 [ 664 1102  822]
 [ 539  254  816]
 [1250  552  625]
 [   1  410  583]
 [ 483  116  274]
 [ 521  514  439]
 [ 652  528 1229]
 [  75  720 1235]
 [ 734  363 1042]
 [ 521  638  972]
 [   1   95  496]
 [1173  910  506]
 [1142 1183  897]
 [ 502  244  979]
 [1002  416  154]
```

```
 [1013 1107 1051]
 [  67   83 1048]
 [ 619  674  577]
 [ 536  261  157]
 [1267 1186  634]
 [ 243  498   12]
 [  55  806  314]
 [ 499  453  666]
 [  13  716  372]
 [1117  824 1218]
 [  79  210  554]
 [ 201 1250  625]
 [ 512   58  906]
 [ 952  649  503]
 [ 347  620 1108]
 [1203  701  313]
 [1260  144 1040]
 [ 600  663  192]
 [1102  452   49]
 [ 734  363 1042]
 [ 645  394    3]
 [1199  437  232]
 [ 640  118 1175]
 [1111  252  642]
 [ 890  509  414]
 [ 708   41  963]
 [ 453  275  677]
 [1167  950  645]
 [ 116  520  163]
 [ 507 1057 1269]
 [ 673  407  795]
 [  92  311  614]
 [ 111  134  574]
 [ 392  816 1128]
 [1198  621 1010]
 [ 490  320  928]
 [1085 1247  351]
 [ 773  597  782]
 [ 656  902 1092]
 [ 387 1169  511]
 [ 870  992 1278]
 [ 940  494  548]
 [ 167  252  655]
 [ 573 1165  169]
 [ 427  783  218]
 [ 359  536 1068]
 [1116  698  915]
 [ 287 1084  910]
 [ 443  420 1063]
 [ 234  276  307]
 [ 408  964  472]
 [1077  828  319]
 [  11  224 1040]
 [  43  557  983]
 [1179  859  527]
 [ 359 1068  574]
 [ 338  460  303]
 [ 205    0 1173]
```

```
 [1100  251 1098]
 [ 210   79  298]
 [ 298   79  183]
 [ 433 1092  494]
 [ 613  844  714]]
[ 48  79  16  39  66  60  95  41  34   3  17  96  97 100   7  62  13  42
  26  90  46  13  98  26  13  43  29  13  11  11  95  85  91  26   9   9
  75  95  56  89  19  13  32  24  26  85  27  33  21  75  21  40  81  48
  44  64  51  87  81  50  42  63   6  35  25  34  65  95  88  44  63  10
  19  67  69  29  80  28  72  58  45  58  93  47  10  53  32  49  50  54
  21  37  95  31  35  69  82  97  84   4  99  99  62  50  98  37  25  14
  26   9  32   1   1  68   7  32  61  29 100   4  73  17  17  21  35  46
  26  25  46  80  18  78  22  23  75  79  49  87  39  58   9  27  92  46
  26  90  82  67  29  54  81  61  28  64  18  97  56  20   3  54  88  71
  65  36  49   3   9  41  91  41  76  51  81  47  66  16  32  47   5  51
   4  67  70  56  24   1  94  91  48  47  50  18  34  63  79  80  55  66
  11  75  42  34  14  46  73  94  87  39  79  32   6  13  89  60  66  63
  15  82  86  32   8  42  53  10   6  84  80  70  36   5  12  41  84  73
  30  75  99  96  31  15  43  46  85  23  23  17  24  68  47  99  38   5
  17  26  42  68   5  62  34  80  83  94  47  54  90  98  22  45  23  15
  77  30  89  45  18  85  38   3  31  40  69  14  96   8   3  21   4  27
  10  25  44  70  30   9   6  59  46  52  59  69  37  34  51  29  36   7
  73  11  81  58  84  66  51  33  26  78  72  72  59  41]
  (0, 12)	1.0
  (0, 70)	1.0
  (0, 30)	1.0
  (1, 485)	1.0
  (1, 861)	1.0
  (1, 602)	1.0
  (2, 285)	1.0
  (2, 1024)	1.0
  (2, 135)	1.0
  (3, 292)	1.0
  (3, 542)	1.0
  (3, 1236)	1.0
  (4, 134)	1.0
  (4, 1179)	1.0
  (4, 859)	1.0
  (5, 934)	1.0
  (5, 771)	1.0
  (5, 492)	1.0
  (6, 370)	1.0
  (6, 1032)	1.0
  (6, 1197)	1.0
  (7, 844)	1.0
  (7, 1089)	1.0
  (7, 175)	1.0
  (8, 218)	1.0
  :	:
  (311, 527)	1.0
  (312, 359)	1.0
  (312, 1068)	1.0
  (312, 574)	1.0
  (313, 338)	1.0
  (313, 460)	1.0
  (313, 303)	1.0
  (314, 205)	1.0
  (314, 0)	1.0
```

```
  (314, 1173)    1.0
  (315, 1100)    1.0
  (315, 251)     1.0
  (315, 1098)    1.0
  (316, 210)     1.0
  (316, 79)      1.0
  (316, 298)     1.0
  (317, 298)     1.0
  (317, 79)      1.0
  (317, 183)     1.0
  (318, 433)     1.0
  (318, 1092)    1.0
  (318, 494)     1.0
  (319, 613)     1.0
  (319, 844)     1.0
  (319, 714)     1.0
Accuracy: 81.25 %
```



Practical No: 10

Question:- Implement K-Means Clustering on Customer Segmentation Dataset taken from kaggle competition.

Theory Explanation:- The Dataset which was provided is

- https://drive.google.com/file/d/1pB7Z8QiFouMypSOQeOv8U4lqloxXjBlO/view?usp=sharing

Unsupervised machine learning
Given a dataset machine figures out how many groups are present in the dataset that consists of similar data-points.
For example:
Pattern detection.
Regions of images.
Useful when no such class or label information is available.

Types of Clustering:
Flat or Partition Clustering:
K-means
Fuzzy c-means
Hierarchical Clustering:
Agglomerative – Bottom Up
Divisive – Top Down

Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os

dataset = pd.read_csv('Customers-Segmentation-Kmeans.csv')
dataset.head(10)


dataset.shape

dataset.info()


dataset.isnull().sum()

X= dataset.iloc[:, [3,4]].values

from sklearn.cluster import KMeans
wcss=[]
for i in range(1,11):
    kmeans = KMeans(n_clusters= i, init='k-means++', random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1,11), wcss)
plt.title('The Elbow Method')
plt.xlabel('no of clusters')
plt.ylabel('wcss')
plt.show()

kmeansmodel = KMeans(n_clusters= 5, init='k-means++', random_state=0)
y_kmeans= kmeansmodel.fit_predict(X)

plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label
= 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue',
label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green',
label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan',
label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta',
label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s =
300, c = 'yellow', label = 'Centroids')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

Output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   CustomerID           200 non-null    int64
 1   Genre                200 non-null    object
 2   Age                  200 non-null    int64
 3   Annual_Income_(k$)   200 non-null    int64
 4   Spending_Score       200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.1+ KB
```