JNAN VIKAS MANDAL'S

Mohanlal Raichand Mehta College of Commerce
Diwali Maa College of Science
Amritlal Raichand Mehta College of Arts
Dr. R.T. Doshi College of Computer Science
NAAC Re-Accredited Grade 'A+' (CGPA : 3.31) (3rd Cycle)

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that **Victoria Martha Dsouza** bearing seat no. **2295165** has done the project work/journal work in the subject of **Machine Learning** of Semester III Practical Examination during the Academic Year 2023-24 under the guidance of **Shweta Gupta** being the partial requirement for the fulfilment of the curriculum of Degree in Master of Science in Information Technology under University of Mumbai.

Place: Airoli Date:

Sign of Subject in- Charge Sign of External Examiner

Sign of Coordinator

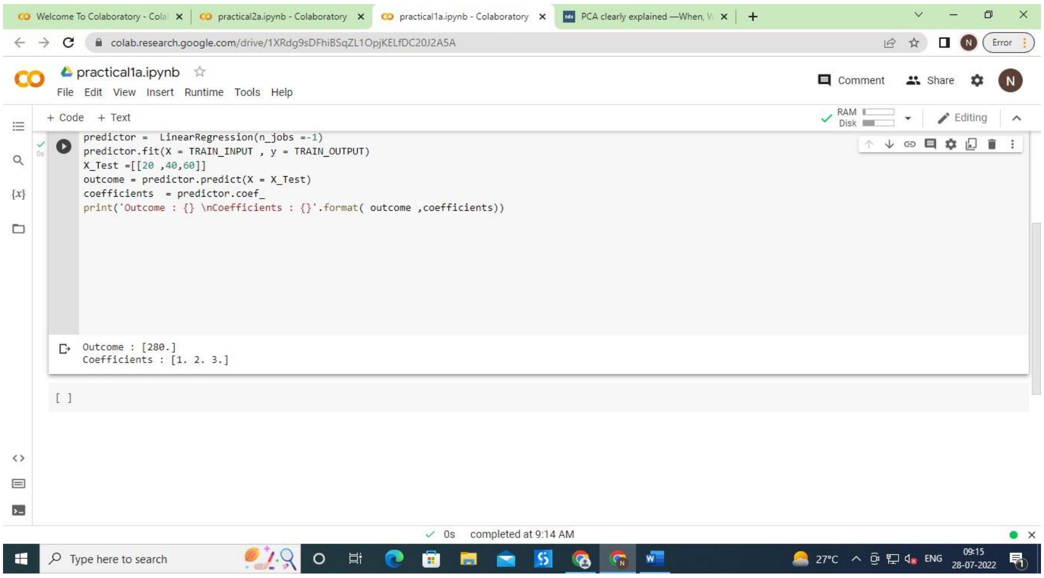
INDEX

7	**a. Implement the classification model using clustering for the following techniques with hierarchical clustering with Prediction, Test Score and Confusion Matrix**		
8	**a. Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.**		
	b. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.		
9	**a. Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.**		
	b. Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task		

Practical 1(A)

Aim: Design a simple machine learning model to train the training instances and test the same.

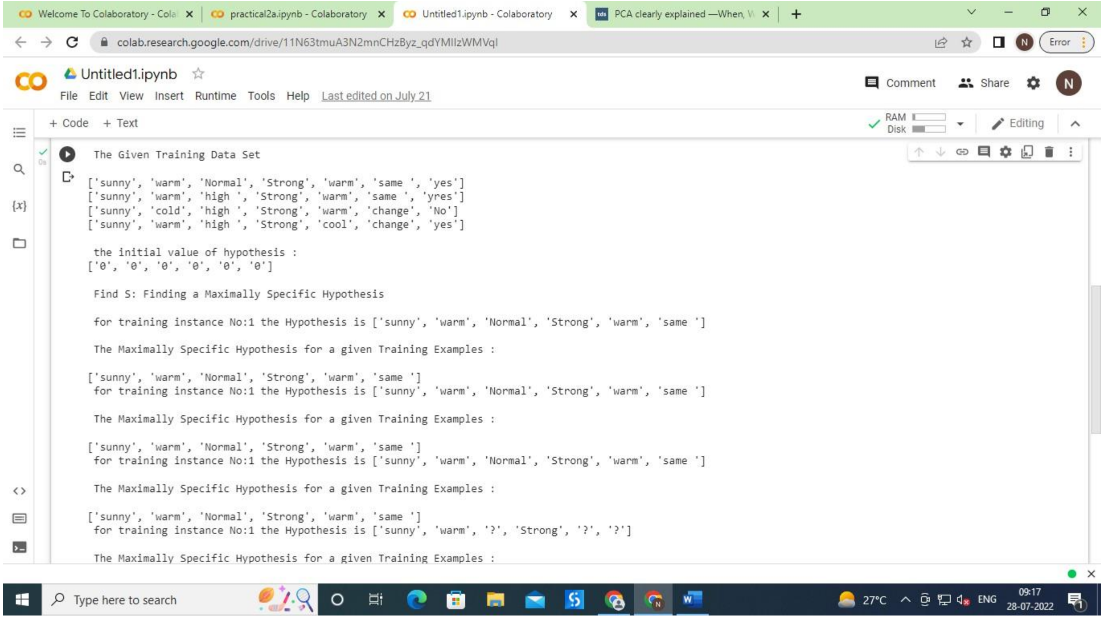
```
from random import randint
TRAIN_SET_LIMIT=1000
TRAIN_SET_COUNT=100
TRAIN_INPUT = list()
TRAIN_OUTPUT = list()
for i  in range (TRAIN_SET_COUNT) :
    a = randint(0, TRAIN_SET_LIMIT )
    b = randint(0, TRAIN_SET_LIMIT )
    c = randint(0, TRAIN_SET_LIMIT )
    op = a + (2 * b) + (3 * c)
    TRAIN_INPUT.append([a ,b, c])
    TRAIN_OUTPUT.append(op)
from sklearn.linear_model import LinearRegression
predictor =  LinearRegression(n_jobs =-1)
predictor.fit(X = TRAIN_INPUT , y = TRAIN_OUTPUT)
X_Test =[[20 ,40,60]]
outcome = predictor.predict(X = X_Test)
coefficients  = predictor.coef_
print('Outcome : {} \nCoefficients : {}'.format( outcome ,coefficients)
)
```

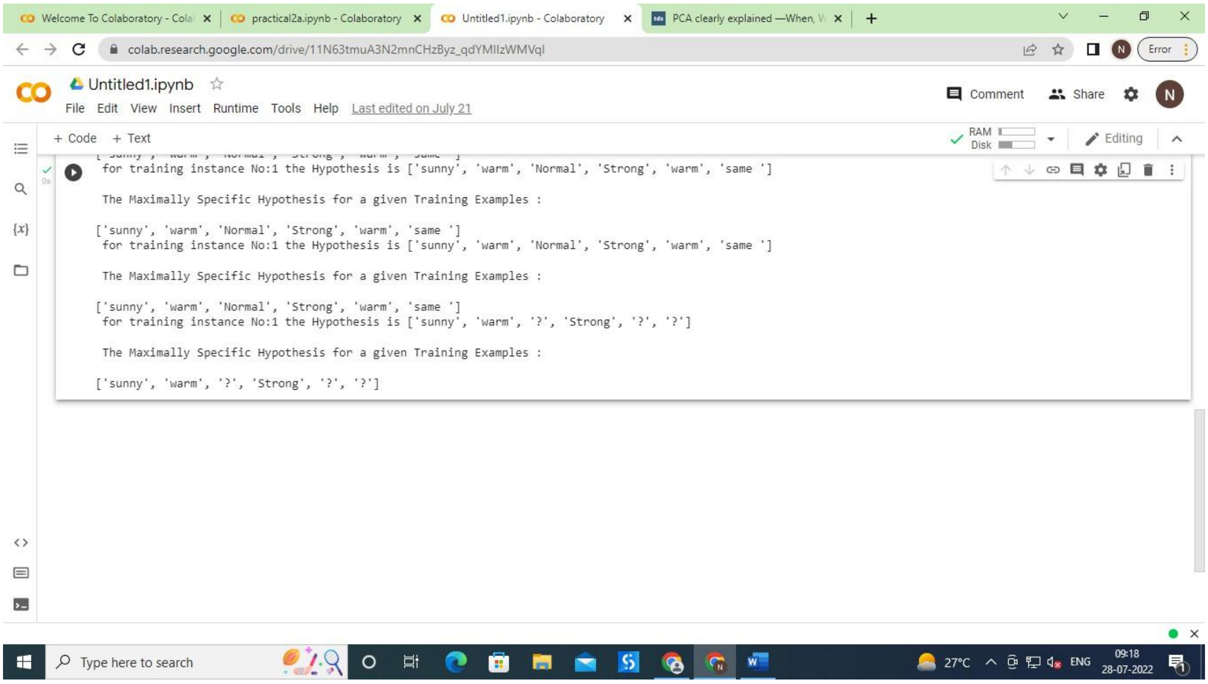


Practical 1(B)

Aim: Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file

```
import csv
num_attributes = 6
a =[]
print("\n The Given Training Data Set \n")
with open('enjoysports.csv' ,'r') as csvfile :
  reader = csv.reader(csvfile)
  for row in reader:
    a.append (row)
    print(row)
print("\n the initial value of hypothesis :")
hypothesis = ['0'] * num_attributes
print(hypothesis)
for j in range (0,num_attributes) :
  hypothesis[j] = a[0][j];
print("\n Find S: Finding a Maximally Specific Hypothesis\n")
for i  in range( 0, len(a)) :
  if a[i] [num_attributes] =='yes' :
      for j in range (0, num_attributes) :
        if a[i][j]!=  hypothesis[j] :
           hypothesis[j] ='?'
        else :  hypothesis[j] = a[i][j]
  print(" for training instance No:{0} the Hypothesis is".format(1) ,hy
pothesis)
  print("\n The Maximally Specific Hypothesis for a given Training Exam
ples :\n")
  print(hypothesis)
```

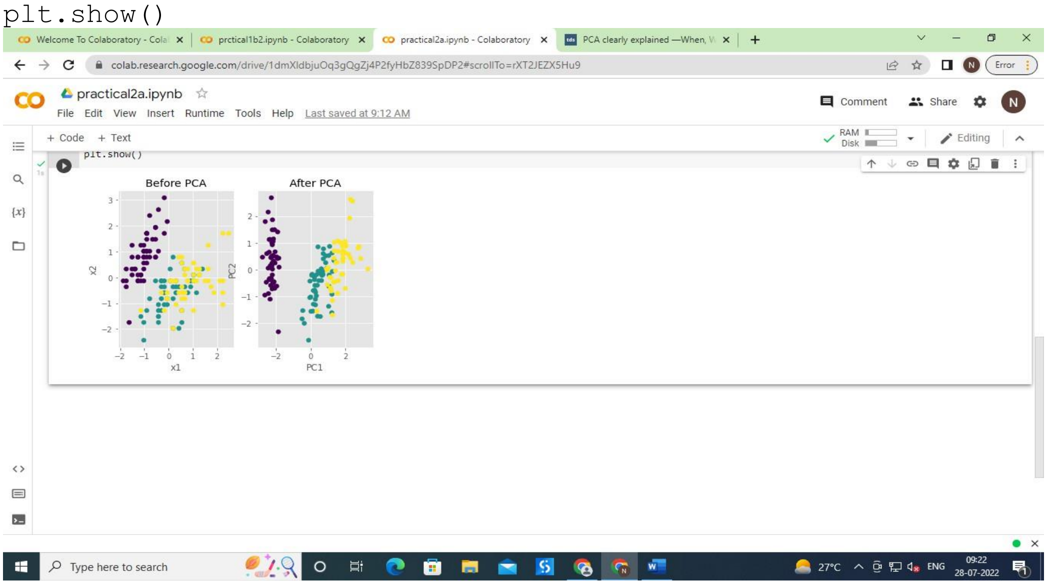

Untitled1.ipynb
The Given Training Data Set
['sunny', 'warm', 'Normal', 'Strong', 'warm', 'same ', 'yes']
['sunny', 'warm', 'high ', 'Strong', 'warm', 'same ', 'yres']
['sunny', 'cold', 'high ', 'Strong', 'warm', 'change', 'No']
['sunny', 'warm', 'high ', 'Strong', 'cool', 'change', 'yes']
the initial value of hypothesis :
['0', '0', '0', '0', '0', '0']
Find S: Finding a Maximally Specific Hypothesis
for training instance No:1 the Hypothesis is ['sunny', 'warm', 'Normal', 'Strong', 'warm', 'same ']
The Maximally Specific Hypothesis for a given Training Examples :
['sunny', 'warm', 'Normal', 'Strong', 'warm', 'same ']
for training instance No:1 the Hypothesis is ['sunny', 'warm', 'Normal', 'Strong', 'warm', 'same ']
The Maximally Specific Hypothesis for a given Training Examples :
['sunny', 'warm', 'Normal', 'Strong', 'warm', 'same ']
for training instance No:1 the Hypothesis is ['sunny', 'warm', 'Normal', 'Strong', 'warm', 'same ']
The Maximally Specific Hypothesis for a given Training Examples :
['sunny', 'warm', 'Normal', 'Strong', 'warm', 'same ']
for training instance No:1 the Hypothesis is ['sunny', 'warm', '?', 'Strong', '?', '?']
The Maximally Specific Hypothesis for a given Training Examples :


Untitled1.ipynb
for training instance No:1 the Hypothesis is ['sunny', 'warm', 'Normal', 'Strong', 'warm', 'same ']
The Maximally Specific Hypothesis for a given Training Examples :
['sunny', 'warm', 'Normal', 'Strong', 'warm', 'same ']
for training instance No:1 the Hypothesis is ['sunny', 'warm', 'Normal', 'Strong', 'warm', 'same ']
The Maximally Specific Hypothesis for a given Training Examples :
['sunny', 'warm', 'Normal', 'Strong', 'warm', 'same ']
for training instance No:1 the Hypothesis is ['sunny', 'warm', '?', 'Strong', '?', '?']
The Maximally Specific Hypothesis for a given Training Examples :
['sunny', 'warm', '?', 'Strong', '?', '?']

Practical 2(A)

Aim: Perform Data Loading, Feature selection (Principal Component analysis) and Feature Scoring and Ranking.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
import pandas as pd
from sklearn.preprocessing import StandardScaler
plt.style.use('ggplot')
# Load the data
iris = datasets.load_iris()
X = iris.data
y = iris.target
# Z-score the features
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)# The PCA model
pca = PCA(n_components=2) # estimate only 2 PCs
X_new = pca.fit_transform(X)
fig, axes = plt.subplots(1,2)
axes[0].scatter(X[:,0], X[:,1], c=y)
axes[0].set_xlabel('x1')
axes[0].set_ylabel('x2')
axes[0].set_title('Before PCA')
axes[1].scatter(X_new[:,0], X_new[:,1], c=y)
axes[1].set_xlabel('PC1')
axes[1].set_ylabel('PC2')
axes[1].set_title('After PCA')
plt.show()
```



Practical 2(B)

Aim: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

```
import csv
with open("sample_data/enjoysports.csv") as f:
    csv_file=csv.reader(f)
    data=list(csv_file)
    s=data[1][:-1]
    g=[['?' for i in range(len(s))] for j in range(len(s))]
 for i in data:
        if i[-1]=="Yes":
            for j in range(len(s)):
                if i[j]!=s[j]:
                    s[j]='?'
                    g[j][j]='?'

        elif i[-1]=="No":
            for j in range(len(s)):
                if i[j]!=s[j]:
                    g[j][j]=s[j]
                else:
                    g[j][j]="?"
        print("\nSteps of Candidate Elimination Algorithm",data.index(i
)+1)
        print(s)
        print(g)
    gh=[]
    for i in g:
      for j in i:
          if j!='?':
              gh.append(i)
              break
    print("\nFinal specific hypothesis:\n",s)

    print("\nFinal general hypothesis:\n",gh)
```

```
Steps of Candidate Elimination Algorithm 1
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 2
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 3
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 4
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 5
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

Practical 3(A)

Aim: Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

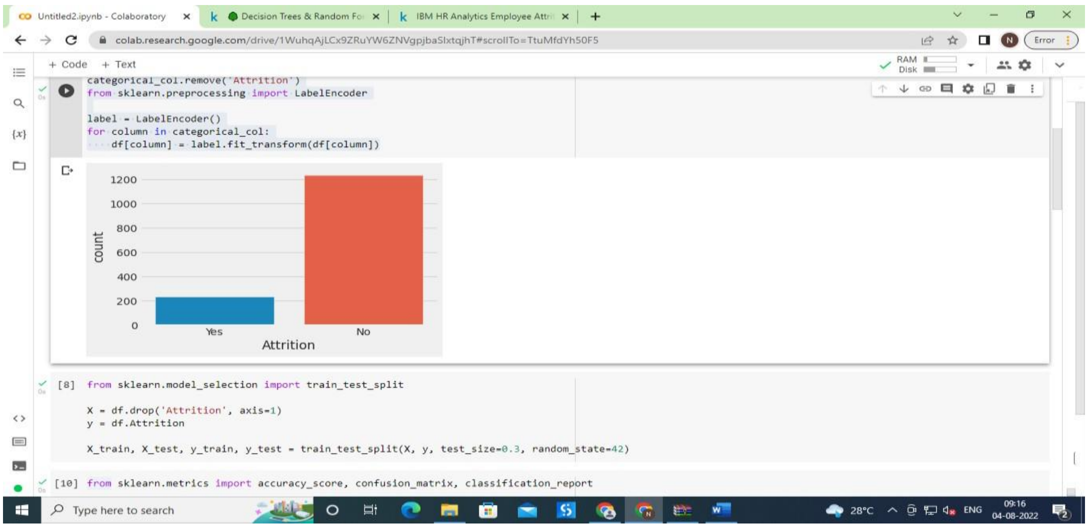
```
import numpy as np
import pandas as pd
from sklearn import datasets
wine= datasets.load_wine()
print(wine)
print("Feature:",wine.feature_names)
print("Labels:",wine.target_names)
X=pd.DataFrame(wine['data'])
print(X.head(0))
print(wine.data.shape)
y=print(wine.target)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train,y_test = train_test_split(wine.data,wine.targe
t,test_size=0.30,random_state=109)
from sklearn.naive_bayes import GaussianNB
gnb=GaussianNB()
gnb.fit(X_train ,y_train)
y_pred =gnb.predict(X_test)
print(y_pred)
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test,y_pred))
from sklearn.metrics import confusion_matrix
cm=np.array(confusion_matrix(y_test ,y_pred))
cm
```

```
[0 0 1 2 0 1 0 0 1 0 2 2 2 2 0 1 1 0 0 1 2 1 0 2 0 0 1 2 0 1 2 1 1 0 1 1 0
 2 2 0 2 1 0 0 0 2 2 0 1 1 2 0 0 2]
Accuracy: 0.9074074074074074
array([[20,  1,  0],
       [ 2, 15,  2],
       [ 0,  0, 14]])
```

Practical 3(B)

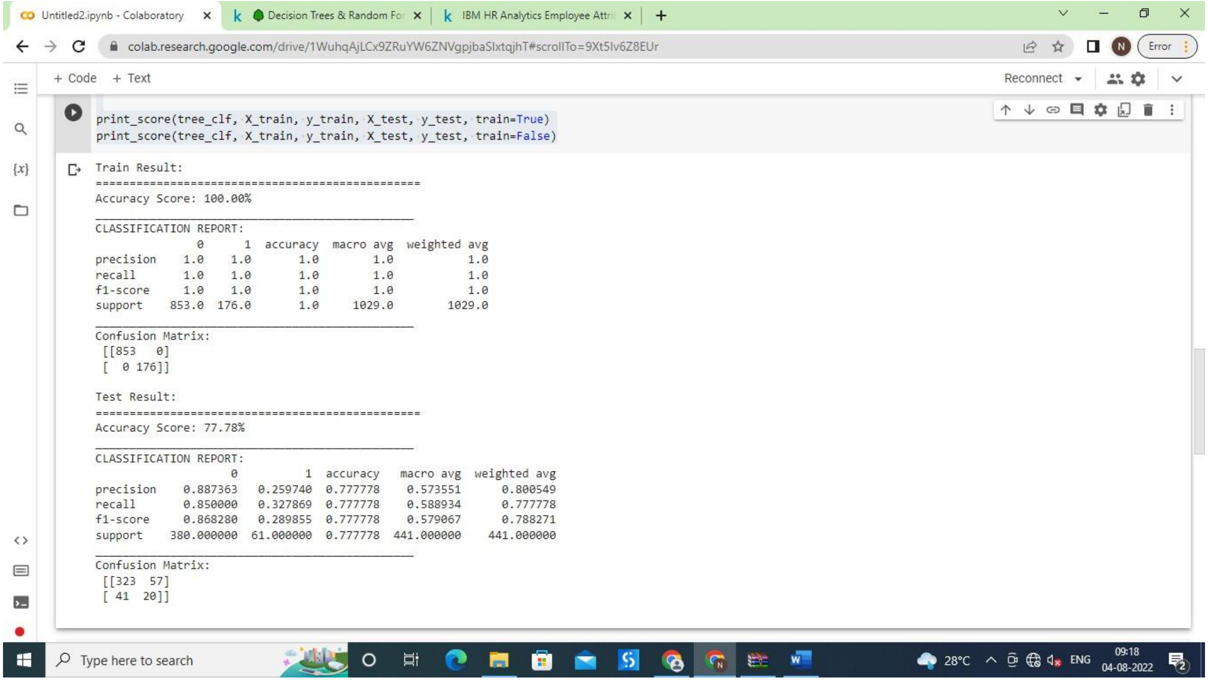
Aim:.Write a program to implement Decision Tree and Random forest with Prediction, Test Score and Confusion Matrix.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set_style("whitegrid")
plt.style.use("fivethirtyeight")
df = pd.read_csv("sample_data/WA_Fn-UseC_-HR-Employee-Attrition.csv")
df.head()
sns.countplot(x='Attrition', data=df)
df.drop(['EmployeeCount', 'EmployeeNumber', 'Over18', 'StandardHours'],
 axis="columns", inplace=True)
categorical_col = []
for column in df.columns:
    if df[column].dtype == object and len(df[column].unique()) <= 50:
        categorical_col.append(column)
df['Attrition'] = df.Attrition.astype("category").cat.codes
categorical_col.remove('Attrition')
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()
for column in categorical_col:
    df[column] = label.fit_transform(df[column])
```

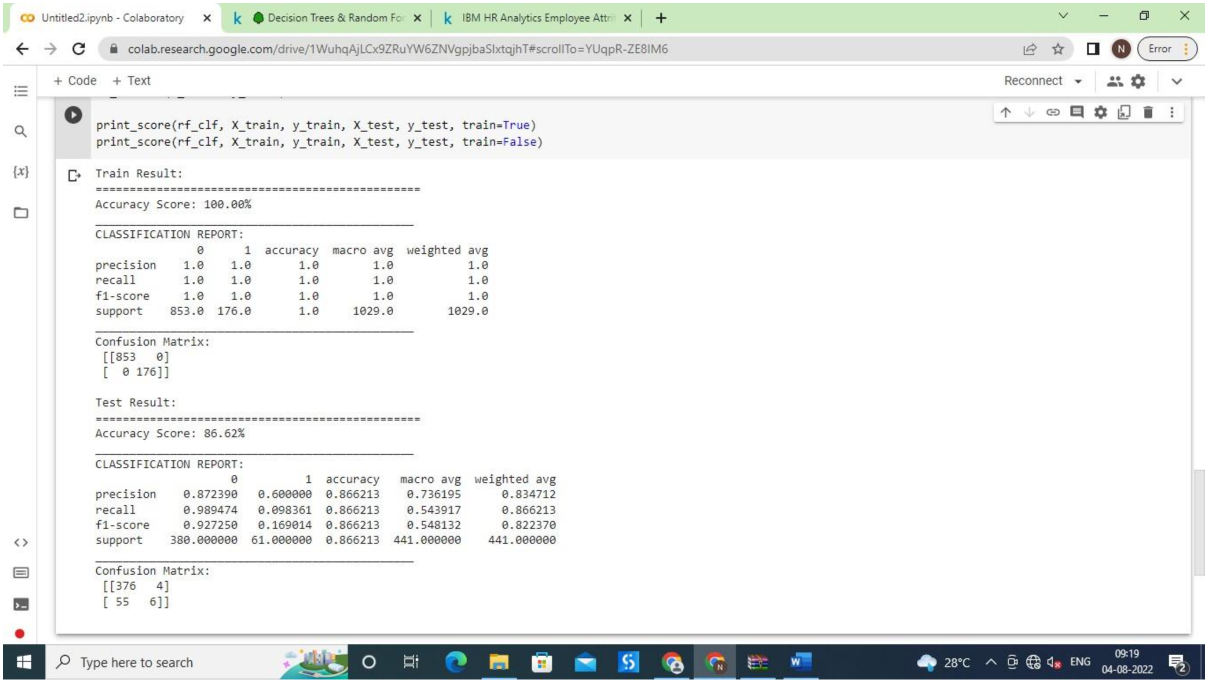



```
from sklearn.model_selection import train_test_split
X = df.drop('Attrition', axis=1)
y = df.Attrition
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3
, random_state=42)
from sklearn.metrics import accuracy_score,confusion_matrix,classificat
ion_report
def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        clf_report = pd.DataFrame(classification_report(y_train, pred,
output_dict=True))
        print("Train Result:\n========================================
=======")
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2
f}%")
        print("_______________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\
n")

    elif train==False:
        pred = clf.predict(X_test)
        clf_report = pd.DataFrame(classification_report(y_test, pred, o
utput_dict=True))
        print("Test Result:\n=========================================
======")
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f
}%")
        print("_______________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n
")

from sklearn.tree import DecisionTreeClassifier

tree_clf = DecisionTreeClassifier(random_state=42)
tree_clf.fit(X_train, y_train)

print_score(tree_clf, X_train, y_train, X_test, y_test, train=True)
print_score(tree_clf, X_train, y_train, X_test, y_test, train=False)
```




```python
from sklearn.ensemble import RandomForestClassifier
rf_clf = RandomForestClassifier(n_estimators=100)
rf_clf.fit(X_train, y_train)
print_score(rf_clf, X_train, y_train, X_test, y_test, train=True)
print_score(rf_clf, X_train, y_train, X_test, y_test, train=False)
```



Practical 4(A)

Aim: For a given set of training data examples stored in a .CSV file implement Least Square Regression algorithm.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
data=pd.read_csv("/content/sample_data/headbrain.csv")
print(data.shape)
```

```
(237, 4)
```

```
print(data.head())
```

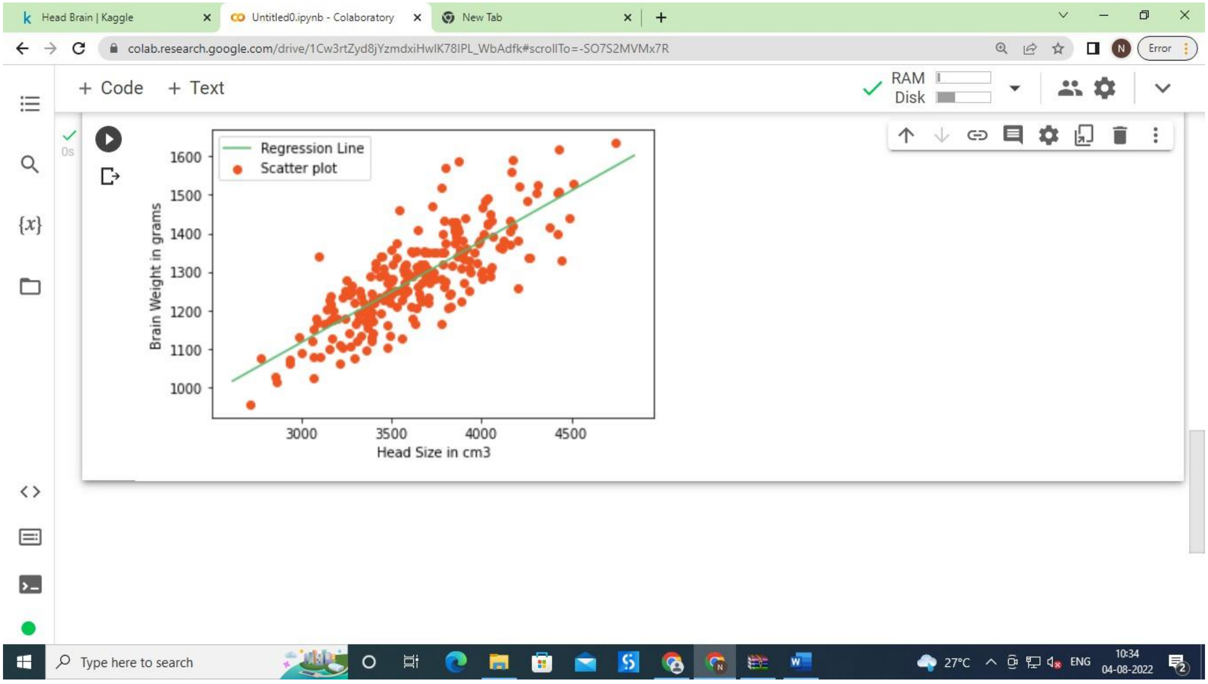
	Gender	Age Range	Head Size(cm^3)	Brain Weight(grams)
0	1	1	4512	1530
1	1	1	3738	1297
2	1	1	4261	1335
3	1	1	3777	1282
4	1	1	4177	1590

```
X=data['Head Size(cm^3)'].values
Y=data['Brain Weight(grams)'].values
mean_x=np.mean(X)
mean_y=np.mean(Y)
n=len(X)
numer = 0
denom = 0
for i in range(n):
 numer+= (X[i] - mean_x) * (Y[i]- mean_y)
 denom +=(X[i] - mean_x) ** 2
m = numer/denom
c= mean_y -( m * mean_x)
print("Coefficients")
print(m,c)
```

```
Coefficients
0.26342933948939945 325.57342104944223
```

```
max_x = np.max(X) + 100
min_x = np.min(X) - 100
x= np.linspace(min_x ,max_x , 1000)
y=c + m * x
plt.plot(x, y , color='#58b970', label='Regression Line')
plt.scatter(X,Y ,c ='#ef5423',label='Scatter plot')
plt.xlabel('Head Size in cm3')
```

```
plt.ylabel('Brain Weight in grams')
plt.legend()
plt.show()
```



Practical 4(B)

Aim: For a given set of training data examples stored in a .CSV file implement Logistic Regression algorithm

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('https://raw.githubusercontent.com/mk-gurucharan/Classification/master/DMVWrittenTests.csv')
X = dataset.iloc[:, [0, 1]].values
y = dataset.iloc[:, 2].values
dataset.head(5)
```

	DMV_Test_1	DMV_Test_2	Results
0	34.623660	78.024693	0
1	30.286711	43.894998	0
2	35.847409	72.902198	0
3	60.182599	86.308552	1
4	79.032736	75.344376	1

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0
.25, random_state = 0)
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
```

LogisticRegression()

```
y_pred = classifier.predict(X_test)
y_pred
```

```
array([1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0,
       1, 1, 0])
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(y_test, y_pred))
cm
```

```
Accuracy :  0.88
array([[11,  0],
       [ 3, 11]])
```

Practical 5(A)

Aim: Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

```
import numpy as np
import pandas as pd
eps = np.finfo(float).eps
from numpy import log2 as log
df_tennis = pd.read_csv('sample_data/play_tennis.csv')
print( df_tennis)
```

```
     day   outlook  temp humidity    wind play
0     D1     Sunny   Hot     High    Weak   No
1     D2     Sunny   Hot     High  Strong   No
2     D3  Overcast   Hot     High    Weak  Yes
3     D4      Rain  Mild     High    Weak  Yes
4     D5      Rain  Cool   Normal    Weak  Yes
5     D6      Rain  Cool   Normal  Strong   No
6     D7  Overcast  Cool   Normal  Strong  Yes
7     D8     Sunny  Mild     High    Weak   No
8     D9     Sunny  Cool   Normal    Weak  Yes
9    D10      Rain  Mild   Normal    Weak  Yes
10   D11     Sunny  Mild   Normal  Strong  Yes
11   D12  Overcast  Mild     High  Strong  Yes
12   D13  Overcast   Hot   Normal    Weak  Yes
13   D14      Rain  Mild     High  Strong   No
```

```
df = pd.DataFrame(df_tennis,columns=['outlook','temp','humidity','wind'
,'play'])
##1. claculate entropy o the whole dataset

entropy_node = 0  #Initialize Entropy
values = df.play.unique()  #Unique objects - 'Yes', 'No'
for value in values:
    fraction = df.play.value_counts()[value]/len(df.play)
    entropy_node += -fraction*np.log2(fraction)
def ent(df,attribute):
    target_variables = df.play.unique()
    variables = df[attribute].unique()
    entropy_attribute = 0
    for variable in variables:
        entropy_each_feature = 0
        for target_variable in target_variables:
            num = len(df[attribute][df[attribute]==variable][df.play ==
target_variable]) #numerator
            den = len(df[attribute][df[attribute]==variable])
```

```
 #denominator
            fraction = num/(den+eps)  #pi
            entropy_each_feature += -fraction*log(fraction+eps)
#This calculates entropy for one feature like 'Sweet'
        fraction2 = den/len(df)
        entropy_attribute += -fraction2*entropy_each_feature
  #Sums up all the entropy ETaste
    return(abs(entropy_attribute))
a_entropy = {k:ent(df,k) for k in df.keys()[:-1]}
a_entropy
```

```
{'outlook': 0.6935361388961914,
 'temp': 0.9110633930116756,
 'humidity': 0.7884504573082889,
 'wind': 0.892158928262361}
```

```
def ig(e_dataset,e_attr):
    return(e_dataset-e_attr)
#entropy_node = entropy of dataset
#a_entropy[k] = entropy of k(th) attr
IG = {k:ig(entropy_node,a_entropy[k]) for k in a_entropy}
IG
```

```
{'outlook': 0.2467498197744355,
 'temp': 0.029222565658955313,
 'humidity': 0.15183550136234203,
 'wind': 0.04812703040826993}
```

```
def find_entropy(df):
    Class = df.keys()[-
1]   #To make the code generic, changing target variable class name
    entropy = 0
    values = df[Class].unique()
    for value in values:
        fraction = df[Class].value_counts()[value]/len(df[Class])
        entropy += -fraction*np.log2(fraction)
    return entropy
def find_entropy_attribute(df,attribute):
  Class = df.keys()[-1]
  target_variables = df[Class].unique()  #This gives all 'Yes' and 'No'
  variables = df[attribute].unique()
  entropy2 = 0
  for variable in variables:
      entropy = 0
      for target_variable in target_variables:
          num = len(df[attribute][df[attribute]==variable][df[Class] ==
target_variable])
```

```
          den = len(df[attribute][df[attribute]==variable])
          fraction = num/(den+eps)
          entropy += -fraction*log(fraction+eps)
      fraction2 = den/len(df)
      entropy2 += -fraction2*entropy
  return abs(entropy2)
def find_winner(df):
    Entropy_att = []
    IG = []
    for key in df.keys()[:-1]:
#Entropy_att.append(find_entropy_attribute(df,key))
        IG.append(find_entropy(df)-find_entropy_attribute(df,key))
    return df.keys()[:-1][np.argmax(IG)]
 def get_subtable(df, node,value):
  return df[df[node] == value].reset_index(drop=True)
def buildTree(df,tree=None):
    Class = df.keys()[-1]
    #Get attribute with maximum information gain
    node = find_winner(df)
#Get distinct value of that attribute e.g Salary is node and Low,Med an
d High are values
    attValue = np.unique(df[node])
    #Create an empty dictionary to create tree
    if tree is None:
        tree={}
        tree[node] = {}
#We make loop to construct a tree by calling this function recursively.
    #In this we check if the subset is pure and stops if it is pure.
    for value in attValue:
        subtable = get_subtable(df,node,value)
        clValue,counts = np.unique(subtable['play'],return_counts=True)
        if len(counts)==1:#Checking purity of subset
            tree[node][value] = clValue[0]
        else:
            tree[node][value] = buildTree(subtable)
    return tree
t=buildTree(df)
import pprint
pprint.pprint(t)
```

```
{'outlook': {'Overcast': 'Yes',
             'Rain': {'wind': {'Strong': 'No', 'Weak': 'Yes'}},
             'Sunny': {'humidity': {'High': 'No', 'Normal': 'Yes'}}}}
```

Practical 5(B)

Aim: Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

```
import pandas as pd
import numpy as np
import operator
import matplotlib.pyplot as plt
data = pd.read_csv('sample_data/Iris.csv', header=None, names=['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class'])
print(data)
```

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

[150 rows x 5 columns]
```

```
indices = np.random.permutation(data.shape[0])
div = int(0.75 * len(indices))
development_id, test_id = indices[:div], indices[div:]
development_set, test_set = data.loc[development_id,:], data.loc[test_id,:]
print("Development Set:\n", development_set, "\n\nTest Set:\n", test_set)
```

```
Development Set:
     sepal_length  sepal_width  petal_length  petal_width            class
140           6.7          3.1           5.6          2.4   Iris-virginica
16            5.4          3.9           1.3          0.4      Iris-setosa
139           6.9          3.1           5.4          2.1   Iris-virginica
88            5.6          3.0           4.1          1.3  Iris-versicolor
20            5.4          3.4           1.7          0.2      Iris-setosa
..            ...          ...           ...          ...              ...
19            5.1          3.8           1.5          0.3      Iris-setosa
87            6.3          2.3           4.4          1.3  Iris-versicolor
138           6.0          3.0           4.8          1.8   Iris-virginica
60            5.0          2.0           3.5          1.0  Iris-versicolor
55            5.7          2.8           4.5          1.3  Iris-versicolor

[112 rows x 5 columns]
```

```
Test Set:
     sepal_length  sepal_width  petal_length  petal_width            class
23            5.1          3.3           1.7          0.5      Iris-setosa
77            6.7          3.0           5.0          1.7  Iris-versicolor
68            6.2          2.2           4.5          1.5  Iris-versicolor
11            4.8          3.4           1.6          0.2      Iris-setosa
141           6.9          3.1           5.1          2.3   Iris-virginica
58            6.6          2.9           4.6          1.3  Iris-versicolor
120           6.9          3.2           5.7          2.3   Iris-virginica
95            5.7          3.0           4.2          1.2  Iris-versicolor
24            4.8          3.4           1.9          0.2      Iris-setosa
30            4.8          3.1           1.6          0.2      Iris-setosa
66            5.6          3.0           4.5          1.5  Iris-versicolor
116           6.5          3.0           5.5          1.8   Iris-virginica
113           5.7          2.5           5.0          2.0   Iris-virginica
112           6.8          3.0           5.5          2.1   Iris-virginica
62            6.0          2.2           4.0          1.0  Iris-versicolor
83            6.0          2.7           5.1          1.6  Iris-versicolor
33            5.5          4.2           1.4          0.2      Iris-setosa
37            4.9          3.1           1.5          0.1      Iris-setosa
52            6.9          3.1           4.9          1.5  Iris-versicolor
131           7.9          3.8           6.4          2.0   Iris-virginica
128           6.4          2.8           5.6          2.1   Iris-virginica
50            7.0          3.2           4.7          1.4  Iris-versicolor
118           7.7          2.6           6.9          2.3   Iris-virginica
98            5.1          2.5           3.0          1.1  Iris-versicolor
92            5.8          2.6           4.0          1.2  Iris-versicolor
25            5.0          3.0           1.6          0.2      Iris-setosa
126           6.2          2.8           4.8          1.8   Iris-virginica
54            6.5          2.8           4.6          1.5  Iris-versicolor
136           6.3          3.4           5.6          2.4   Iris-virginica
91            6.1          3.0           4.6          1.4  Iris-versicolor
```

```
mean_development_set = development_set.mean()
mean_test_set = test_set.mean()
std_development_set = development_set.std()
std_test_set = test_set.std()
test_class = list(test_set.iloc[:,-1])
dev_class = list(development_set.iloc[:,-1])
def euclideanDistance(data_1, data_2, data_len):
    dist = 0
    for i in range(data_len):
        dist = dist + np.square(data_1[i] - data_2[i])
    return np.sqrt(dist)

def normalizedEuclideanDistance(data_1, data_2, data_len, data_mean, da
ta_std):
    n_dist = 0
    for i in range(data_len):
```

```
        n_dist = n_dist + (np.square(((data_1[i] - data_mean[i])/data_s
td[i]) - ((data_2[i] - data_mean[i])/data_std[i])))
    return np.sqrt(n_dist)

def cosineSimilarity(data_1, data_2):
    dot = np.dot(data_1, data_2[:-1])
    norm_data_1 = np.linalg.norm(data_1)
    norm_data_2 = np.linalg.norm(data_2[:-1])
    cos = dot / (norm_data_1 * norm_data_2)
    return (1-cos)
def knn(dataset, testInstance, k, dist_method, dataset_mean, dataset_st
d):
    distances = {}
    length = testInstance.shape[1]
    if dist_method == 'euclidean':
        for x in range(len(dataset)):
            dist_up = euclideanDistance(testInstance, dataset.iloc[x],
length)
            distances[x] = dist_up[0]
    elif dist_method == 'normalized_euclidean':
        for x in range(len(dataset)):
            dist_up = normalizedEuclideanDistance(testInstance, dataset
.iloc[x], length, dataset_mean, dataset_std)
            distances[x] = dist_up[0]
    elif  dist_method  ==  'cosine':
        for x in range(len(dataset)):
            dist_up = cosineSimilarity(testInstance, dataset.iloc[x])
            distances[x] = dist_up[0]
    # Sort values based on distance
    sort_distances = sorted(distances.items(), key=operator.itemgetter(
1))
    neighbors = []
    # Extracting nearest k neighbors
    for x in range(k):
        neighbors.append(sort_distances[x][0])
    # Initializing counts for 'class' labels counts as 0
    counts = {"Iris-setosa" : 0, "Iris-versicolor" : 0, "Iris-
virginica" : 0}
    # Computing the most frequent class
    for x in range(len(neighbors)):
        response = dataset.iloc[neighbors[x]][-1]
        if response in counts:
            counts[response] += 1
        else:
            counts[response] = 1
    # Sorting the class in reverse order to get the most frequest class
    sort_counts = sorted(counts.items(), key=operator.itemgetter(1), re
verse=True)
```

```
    return(sort_counts[0][0])
# Creating a list of list of all columns except 'class' by iterating th
rough the development set
row_list = []
for index, rows in development_set.iterrows():
    my_list =[rows.sepal_length, rows.sepal_width, rows.petal_length, r
ows.petal_width]
    row_list.append([my_list])
# k values for the number of neighbors that need to be considered
k_n = [1, 3, 5, 7]
# Distance metrics
distance_methods = ['euclidean', 'normalized_euclidean', 'cosine']
# Performing kNN on the development set by iterating all of the develop
ment set data points and for each k and each distance metric
obs_k = {}
for dist_method in distance_methods:
    development_set_obs_k = {}
    for k in k_n:
        development_set_obs = []
        for i in range(len(row_list)):
            development_set_obs.append(knn(development_set, pd.DataFram
e(row_list[i]), k, dist_method, mean_development_set, std_development_s
et))
        development_set_obs_k[k] = development_set_obs
    # Nested Dictionary containing the observed class for each k and ea
ch distance metric (obs_k of the form obs_k[dist_method][k])
    obs_k[dist_method] = development_set_obs_k
print(obs_k)
```

```
{'euclidean': {1: ['Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-virginica', 'Iris-virginica', 'Iris-versicolor', 'Iris-virginic
```

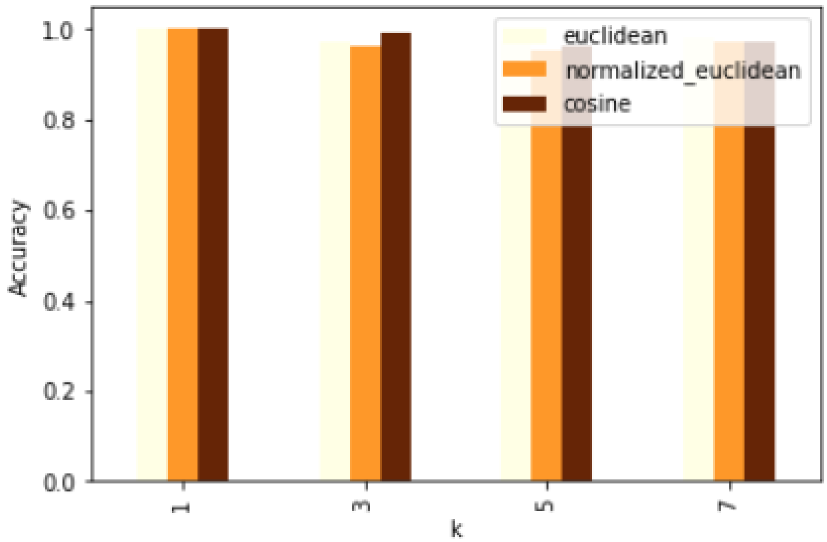
```
accuracy = {}
for key in obs_k.keys():
    accuracy[key] = {}
    for k_value in obs_k[key].keys():
        #print('k = ', key)
        count = 0
        for i,j in zip(dev_class, obs_k[key][k_value]):
            if i == j:
                count = count + 1
            else:
                pass
        accuracy[key][k_value] = count/(len(dev_class))

# Storing the accuracy for each k and each distance metric into a dataf
rame
df_res = pd.DataFrame({'k': k_n})
for key in accuracy.keys():
```

```
    value = list(accuracy[key].values())
    df_res[key] = value
print(df_res)
```

```
   k  euclidean  normalized_euclidean    cosine
0  1   1.000000              1.000000  1.000000
1  3   0.973214              0.964286  0.991071
2  5   0.973214              0.955357  0.964286
3  7   0.982143              0.973214  0.973214
```

```
# Plotting a Bar Chart for accuracy
draw = df_res.plot(x='k', y=['euclidean', 'normalized_euclidean', 'cosi
ne'], kind="bar", colormap='YlOrBr')
draw.set(ylabel='Accuracy')
```

```
[Text(0, 0.5, 'Accuracy')]
```



Practical 6(A)

Aim: Implement the different Distance methods (Euclidean) with Prediction, Test Score and Confusion Matrix.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sklearn
dataset = pd.read_csv('sample_data/Social_Network_Ads.csv')
X = dataset.iloc[:, [1, 2, 3]].values
y = dataset.iloc[:, -1].values
X
```

```
array([[1, 19, 19000],
       [1, 35, 20000],
       [0, 26, 43000],
       ...,
       [0, 50, 20000],
       [1, 36, 33000],
       [0, 49, 36000]], dtype=object)
```

```
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 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, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 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, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1])
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:,0] = le.fit_transform(X[:,0])
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size = 0.
20, random_state = 0)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.neighbors import KNeighborsClassifier
classifier=KNeighborsClassifier(n_neighbors=5,metric='minkowski',p = 2)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
y_pred
```

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1])
```

```
y_test
```

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1])
```

```
from sklearn.metrics import confusion_matrix,accuracy_score
cm = confusion_matrix(y_test, y_pred)
ac = accuracy_score(y_test,y_pred)
cm
```

```
array([[55,  3],
       [ 1, 21]])
```

```
ac
```

```
0.95
```

Practical 6(B)

Aim: Implement the classification model using clustering for the following techniques with K means clustering with Prediction, Test Score and Confusion Matrix.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # for data visualization
import seaborn as sns # for statistical data visualization
%matplotlib inline
data = '/content/sample_data/Live.csv'
df = pd.read_csv(data)
df.shape
```

```
(7050, 12)
```

```
df.head()
```

	status_id	status_type	status_published	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys
0	246675545449582_1649696485147474	video	4/22/2018 6:00	529	512	262	432	92	3	1	1	0
1	246675545449582_1649426988507757	photo	4/21/2018 22:45	150	0	0	150	0	0	0	0	0
2	246675545449582_1648730588577397	video	4/21/2018 6:17	227	236	57	204	21	1	1	0	0
3	246675545449582_1648576705259452	photo	4/21/2018 2:29	111	0	0	111	0	0	0	0	0
4	246675545449582_1645700502213739	photo	4/18/2018 3:22	213	0	0	204	9	0	0	0	0

```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   status_id         7050 non-null   object
 1   status_type       7050 non-null   object
 2   status_published  7050 non-null   object
 3   num_reactions     7050 non-null   int64
 4   num_comments      7050 non-null   int64
 5   num_shares        7050 non-null   int64
 6   num_likes         7050 non-null   int64
 7   num_loves         7050 non-null   int64
 8   num_wows          7050 non-null   int64
 9   num_hahas         7050 non-null   int64
 10  num_sads          7050 non-null   int64
 11  num_angrys        7050 non-null   int64
dtypes: int64(9), object(3)
```

```
df.isnull().sum()
```

```
status_id           0
status_type         0
status_published    0
num_reactions       0
num_comments        0
num_shares          0
num_likes           0
num_loves           0
num_wows            0
num_hahas           0
num_sads            0
num_angrys          0
dtype: int64
```

```
df.describe()
```

	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys
count	7050.000000	7050.000000	7050.000000	7050.000000	7050.000000	7050.000000	7050.000000	7050.000000	7050.000000
mean	230.117163	224.356028	40.022553	215.043121	12.728652	1.289362	0.696454	0.243688	0.113191
std	462.625309	889.636820	131.599965	449.472357	39.972930	8.719650	3.957183	1.597156	0.726812
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	17.000000	0.000000	0.000000	17.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	59.500000	4.000000	0.000000	58.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	219.000000	23.000000	4.000000	184.750000	3.000000	0.000000	0.000000	0.000000	0.000000
max	4710.000000	20990.000000	3424.000000	4710.000000	657.000000	278.000000	157.000000	51.000000	31.000000

```
df['status_id'].unique()
```

```
array(['246675545449582_1649696485147474',
       '246675545449582_1649426988507757',
       '246675545449582_1648730588577397', ...,
       '1050855161656896_1060126464063099',
       '1050855161656896_1058663487542730',
       '1050855161656896_1050858841656528'], dtype=object)
```

```
len(df['status_id'].unique())
df['status_published'].unique()
len(df['status_published'].unique())
df['status_type'].unique()
len(df['status_type'].unique())
df.drop(['status_id', 'status_published'], axis=1, inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   status_type    7050 non-null   object
 1   num_reactions  7050 non-null   int64
 2   num_comments   7050 non-null   int64
 3   num_shares     7050 non-null   int64
 4   num_likes      7050 non-null   int64
 5   num_loves      7050 non-null   int64
 6   num_wows       7050 non-null   int64
 7   num_hahas      7050 non-null   int64
 8   num_sads       7050 non-null   int64
 9   num_angrys     7050 non-null   int64
dtypes: int64(9), object(1)
memory usage: 550.9+ KB
```

```
df.head()
```

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys
0	video	529	512	262	432	92	3	1	1	0
1	photo	150	0	0	150	0	0	0	0	0
2	video	227	236	57	204	21	1	1	0	0
3	photo	111	0	0	111	0	0	0	0	0
4	photo	213	0	0	204	9	0	0	0	0

```
X = df
y = df['status_type']
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X['status_type'] = le.fit_transform(X['status_type'])
y = le.transform(y)
X.info()
X.head()
cols = X.columns
from sklearn.preprocessing import MinMaxScaler
ms = MinMaxScaler()
X = ms.fit_transform(X)
X = pd.DataFrame(X, columns=[cols])
X.head()
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans.fit(X)
kmeans.cluster centers
```

```
array([[3.28506857e-01, 3.90710874e-02, 7.54854864e-04, 7.53667113e-04,
        3.85438884e-02, 2.17448568e-03, 2.43721364e-03, 1.20039760e-03,
        2.75348016e-03, 1.45313276e-03],
       [9.54921576e-01, 6.46330441e-02, 2.67028654e-02, 2.93171709e-02,
        5.71231462e-02, 4.71007076e-02, 8.18581889e-03, 9.65207685e-03,
        8.04219428e-03, 7.19501847e-03]])
```

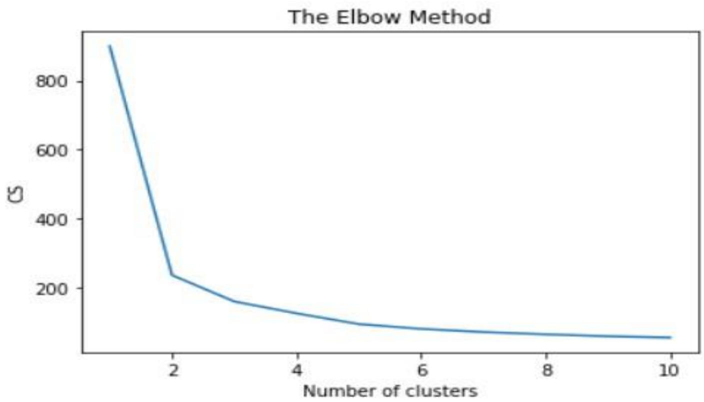
```
labels = kmeans.labels_
# check how many of the samples were correctly labeled
correct_labels = sum(y == labels)

print("Result: %d out of %d samples were correctly labeled." % (correct
_labels, y.size))
```

```
Result: 63 out of 7050 samples were correctly labeled.
```

```
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Accuracy score: 0.01
```

```
from sklearn.cluster import KMeans
cs = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-
means++', max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(X)
    cs.append(kmeans.inertia_)
plt.plot(range(1, 11), cs)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('CS')
plt.show()
```



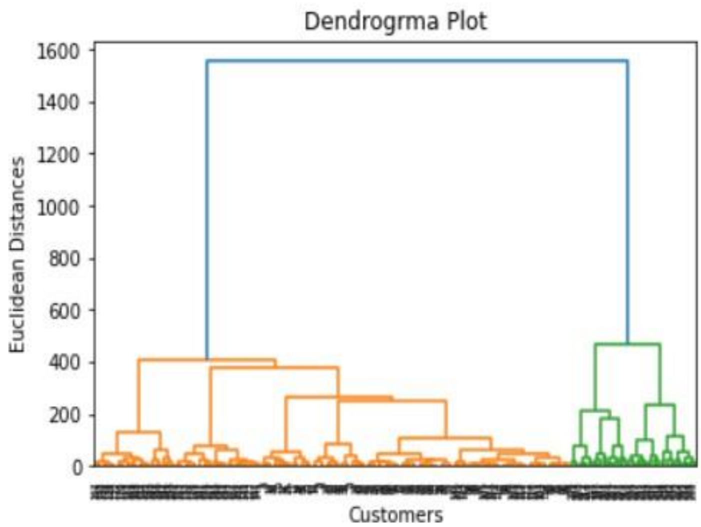

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4,random_state=0)
kmeans.fit(X)
labels = kmeans.labels_
# check how many of the samples were correctly labeled
correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." % (correct
_labels, y.size))
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Result: 4340 out of 7050 samples were correctly labeled.
Accuracy score: 0.62
```

Practical 7(A)

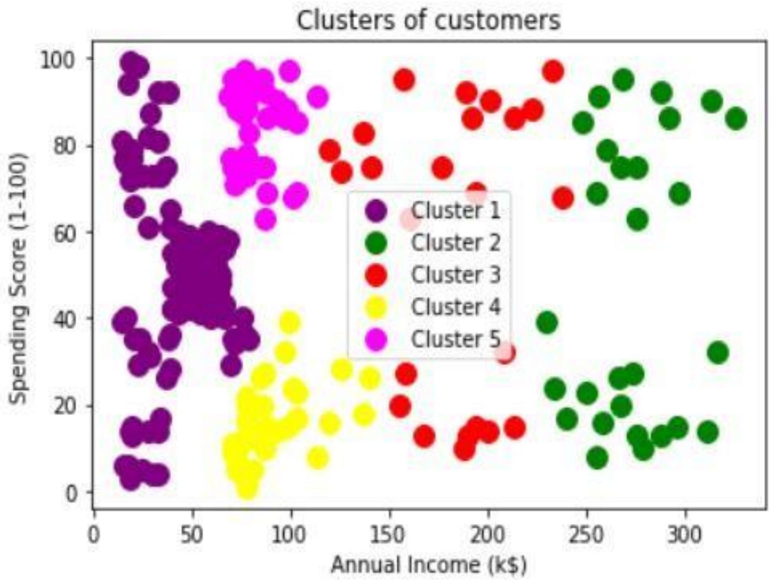
Aim : Implement the classification model using clustering for the following techniques with hierarchical clustering with Prediction, Test Score and Confusion Matrix

```
# Importing the libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('/content/sample_data/Mall_Customers.csv')
x = dataset.iloc[:, [3, 4]].values
#Finding the optimal number of clusters using the dendrogram
import scipy.cluster.hierarchy as shc
dendro = shc.dendrogram(shc.linkage(x, method="ward"))
mtp.title("Dendrogrma Plot")
mtp.ylabel("Euclidean Distances")
mtp.xlabel("Customers")
mtp.show()
```




```
#training the hierarchical model on dataset
from sklearn.cluster import AgglomerativeClustering
hc= AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage
='ward')
y_pred= hc.fit_predict(x)
#visulaizing the clusters
mtp.scatter(x[y_pred == 0, 0], x[y_pred == 0, 1], s = 100, c = 'purple'
, label = 'Cluster 1')
mtp.scatter(x[y_pred == 1, 0], x[y_pred == 1, 1], s = 100, c = 'green',
 label = 'Cluster 2')
mtp.scatter(x[y_pred== 2, 0], x[y_pred == 2, 1], s = 100, c = 'red', la
bel = 'Cluster 3')
```

```
mtp.scatter(x[y_pred == 3, 0], x[y_pred == 3, 1], s = 100, c = 'yellow'
, label = 'Cluster 4')
mtp.scatter(x[y_pred == 4, 0], x[y_pred == 4, 1], s = 100, c = 'magenta
', label = 'Cluster 5')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```



Practical 8(A)

Aim : Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.

```
import numpy as np
import pandas as pd
import csv
!pip install pgmpy
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
heartDisease = pd.read_csv('sample_data/heart.csv')
heartDisease = heartDisease.replace('?',np.nan)
print('\n Attributes and datatypes')
print(heartDisease.dtypes)
```

```
 Attributes and datatypes
age                int64
gender             int64
cp                 int64
trestbps           int64
chol               int64
fbs                int64
restecg            int64
thalach            int64
exang              int64
oldpeak          float64
slope              int64
ca                object
thal              object
heartdisease       int64
dtype: object
```

```
model= BayesianModel([('age','heartdisease'),('gender','heartdisease'),
('exang','heartdisease'),('cp','heartdisease'),('heartdisease','restecg
'),('heartdisease','chol')])
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
print('\n 1. Probability of HeartDisease given evidence= restecg')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'r
estecg':1})
print(q1)
```

```
 1. Probability of HeartDisease given evidence= restecg
Finding Elimination Order: : 100%  4/4 [00:00<00:00, 33.44it/s]
Eliminating: cp: 100%  4/4 [00:00<00:00, 54.19it/s]
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1012 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2392 |
+-----------------+---------------------+
| heartdisease(3) |              0.2015 |
+-----------------+---------------------+
| heartdisease(4) |              0.4581 |
+-----------------+---------------------+
```

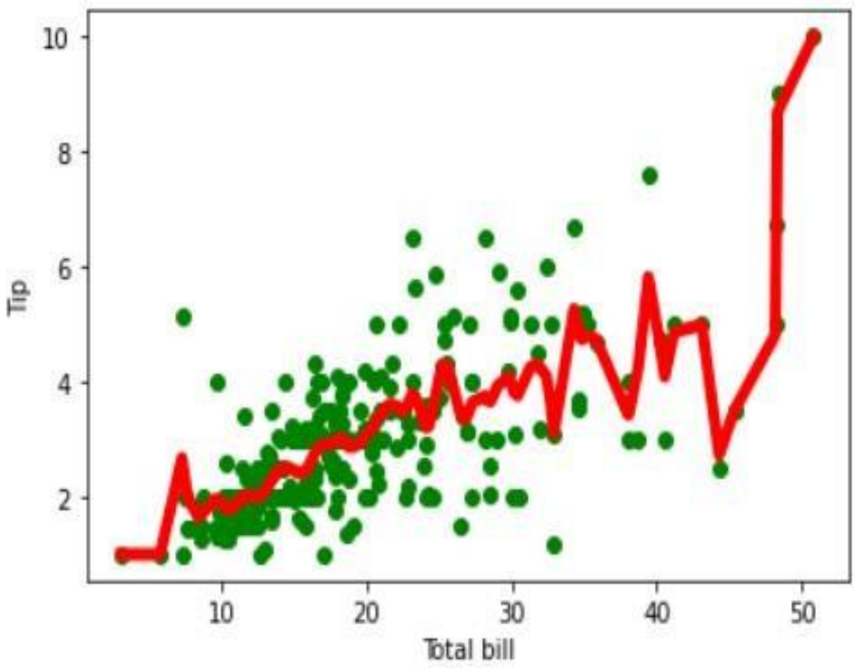
```
print('\n 2. Probability of HeartDisease given evidence= cp ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'c
p':2})
print(q2)
```

```
 2. Probability of HeartDisease given evidence= cp
Finding Elimination Order: : 0%  0/3 [00:00<?, ?it/s]
Eliminating: exang: 100%  3/3 [00:00<00:00, 34.91it/s]
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3610 |
+-----------------+---------------------+
| heartdisease(1) |              0.2159 |
+-----------------+---------------------+
| heartdisease(2) |              0.1373 |
+-----------------+---------------------+
| heartdisease(3) |              0.1537 |
+-----------------+---------------------+
| heartdisease(4) |              0.1321 |
+-----------------+---------------------+
```

Practical 8(B)

Aim : Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
def kernel(point, xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    return weights
def localWeight(point, xmat, ymat, k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W
def localWeightRegression(xmat, ymat, k):
    m,n = np.shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred
# load data points
data = pd.read_csv('sample_data/10-dataset.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)
#preparing and add 1 in bill
mbill = np.mat(bill)
mtip  =  np.mat(tip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T))
#set k here
ypred = localWeightRegression(X,mtip,0.5)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='green')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```


10
8
6
4
2
Tip
10
20
30
40
50
Total bill

Practical 9(A)

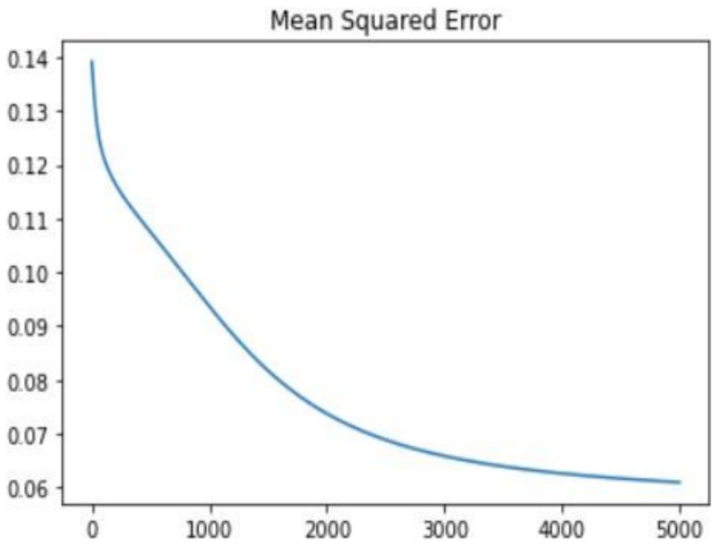
Aim : Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
# Load dataset
data = load_iris()
# Get features and target
X=data.data
y=data.target
# Get dummy variable
y = pd.get_dummies(y).values
y[:3]
```

```
array([[1, 0, 0],
       [1, 0, 0],
       [1, 0, 0]], dtype=uint8)
```

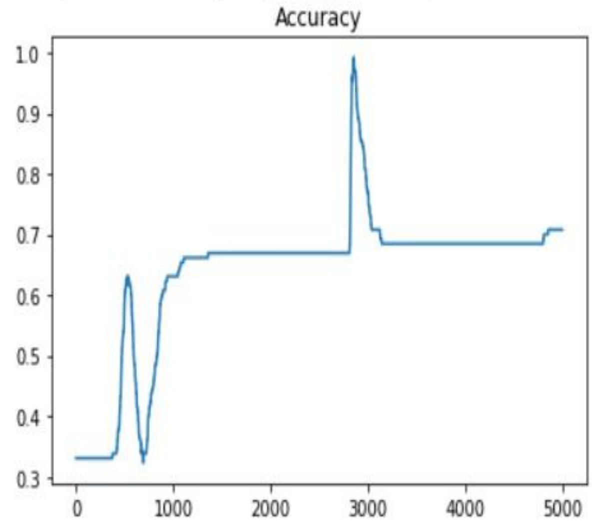
```
#Split data into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=20,
 random_state=4)
# Initialize variables
learning_rate = 0.1
iterations = 5000
N = y_train.size
# number of input features
input_size = 4
# number of hidden layers neurons
hidden_size = 2
# number of neurons at the output layer
output_size = 3
results = pd.DataFrame(columns=["mse", "accuracy"])
# Initialize weights
np.random.seed(10)
# initializing weight for the hidden layer
W1 = np.random.normal(scale=0.5, size=(input_size, hidden_size))
# initializing weight for the output layer
W2 = np.random.normal(scale=0.5, size=(hidden_size , output_size))
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def mean_squared_error(y_pred, y_true):
    return ((y_pred - y_true)**2).sum() / (2*y_pred.size)
```

```
def accuracy(y_pred, y_true):
    acc = y_pred.argmax(axis=1) == y_true.argmax(axis=1)
    return acc.mean()
for itr in range(iterations):
    # feedforward propagation
    # on hidden layer
    Z1 = np.dot(X_train, W1)
    A1 = sigmoid(Z1)
    # on output layer
    Z2 = np.dot(A1, W2)
    A2 = sigmoid(Z2)
    # Calculating error
    mse = mean_squared_error(A2, y_train)
    acc = accuracy(A2, y_train)
   results=results.append({"mse":mse,"accuracy":acc},ignore_index=True)
    # backpropagation
    E1 = A2 - y_train
    dW1 = E1 * A2 * (1 - A2)
    E2 = np.dot(dW1, W2.T)
    dW2 = E2 * A1 * (1 - A1)
    # weight updates
    W2_update = np.dot(A1.T, dW1) / N
    W1_update = np.dot(X_train.T, dW2) / N
    W2 = W2 - learning_rate * W2_update
    W1 = W1 - learning_rate * W1_update

  results.mse.plot(title="Mean Squared Error")
```




```
results.accuracy.plot(title="Accuracy")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f01548eb390>
```




```
# feedforward
Z1 = np.dot(X_test, W1)
A1 = sigmoid(Z1)
Z2 = np.dot(A1, W2)
A2 = sigmoid(Z2)
acc = accuracy(A2, y_test)
print("Accuracy: {}".format(acc))
```

```
Accuracy: 0.8
```

Practical 9(B)

Aim : Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
msg=pd.read_csv('sample_data/data.csv',names=['message','label'])
print('The dimensions of the dataset',msg.shape)
msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message
y=msg.labelnum
```

```
The dimensions of the dataset (18, 2)
```

```
xtrain,xtest,ytrain,ytest=train_test_split(X,y)
print ('\n the total number of Training Data :',ytrain.shape)
print ('\n the total number of Test Data :',ytest.shape)
```

```
the total number of Training Data : (13,)
the total number of Test Data : (5,)
```

```
cv = CountVectorizer()
xtrain_dtm = cv.fit_transform(xtrain)
xtest_dtm=cv.transform(xtest)
print('\n The words or Tokens in the text documents \n')
print(cv.get_feature_names())
df=pd.DataFrame(xtrain dtm.toarray(),columns=cv.get feature names())
```

```
The words or Tokens in the text documents

['about', 'am', 'and', 'bad', 'beers', 'best', 'boss', 'can', 'dance', 'deal', 'do', 'enemy', 'feel', 'fun', 'good', 'great', 'have', 'holiday', 'horrible', 'house', 'is', 'juice', 'li
```

```
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)
#printing accuracy, Confusion matrix, Precision and Recall
print('\n Accuracy of the classifier is',metrics.accuracy_score(ytest,p
redicted))
print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
print('\n The value of Precision',metrics.precision_score(ytest,predict
ed))
print('\n The value of Recall', metrics.recall_score(ytest,predicted))
```

```
Accuracy of the classifier is 0.6

Confusion matrix
[[2 0]
 [2 1]]

The value of Precision 1.0

The value of Recall 0.3333333333333333
```