INDEX

6B	**Implement the classification model using clustering for the following techniques with K means clustering with Prediction, Test Score and Confusion Matrix.**		
7A	**Implement the classification model using clustering for the following techniques with hierarchical clustering with Prediction, Test Score and Confusion Matrix.**		
7B	**Implement the Rule based method and test the same.**		
8A	**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.**		
8B	**Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.**		
9A	**Build an Artificial Neural Network by implementing the Back-propagation algorithm and test the same using appropriate data sets.**		
9B	**Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task.**		
10	**Perform Text pre-processing, Text clustering, classification with Prediction, Test Score and Confusion Matrix**		

Practical No. 1A

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

Code:

```
# Import Standard Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# Import ML Libaries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error
# Import Dataset from sklearn
from sklearn.datasets import load_iris

#Iris DataSet
iris = load_iris();
#Preparing Iris Data
iris_df = pd.DataFrame(data= iris.data, columns= iris.feature_names)
target_df = pd.DataFrame(data= iris.target, columns= ['species'])
def converter(specie):
  if specie == 0:
    return 'setosa'
  elif specie == 1:
    return 'versicolor'
  else:
    return 'virginica'
target_df['species'] = target_df['species'].apply(converter)
iris_df = pd.concat([iris_df, target_df], axis= 1)

#Data Overview
iris_df.describe()
```

```
# Import Standard Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# Import ML Libaries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error
# Import Dataset from sklearn
from sklearn.datasets import load_iris
```

```
#Iris DataSet
iris = load_iris();
#Preparing Iris Data
iris_df = pd.DataFrame(data= iris.data, columns= iris.feature_names)
target_df = pd.DataFrame(data= iris.target, columns= ['species'])
def converter(specie):
  if specie == 0:
    return 'setosa'
  elif specie == 1:
    return 'versicolor'
  else:
    return 'virginica'
target_df['species'] = target_df['species'].apply(converter)
iris_df = pd.concat([iris_df, target_df], axis= 1)
```

```
#Data Overview
iris_df.describe()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

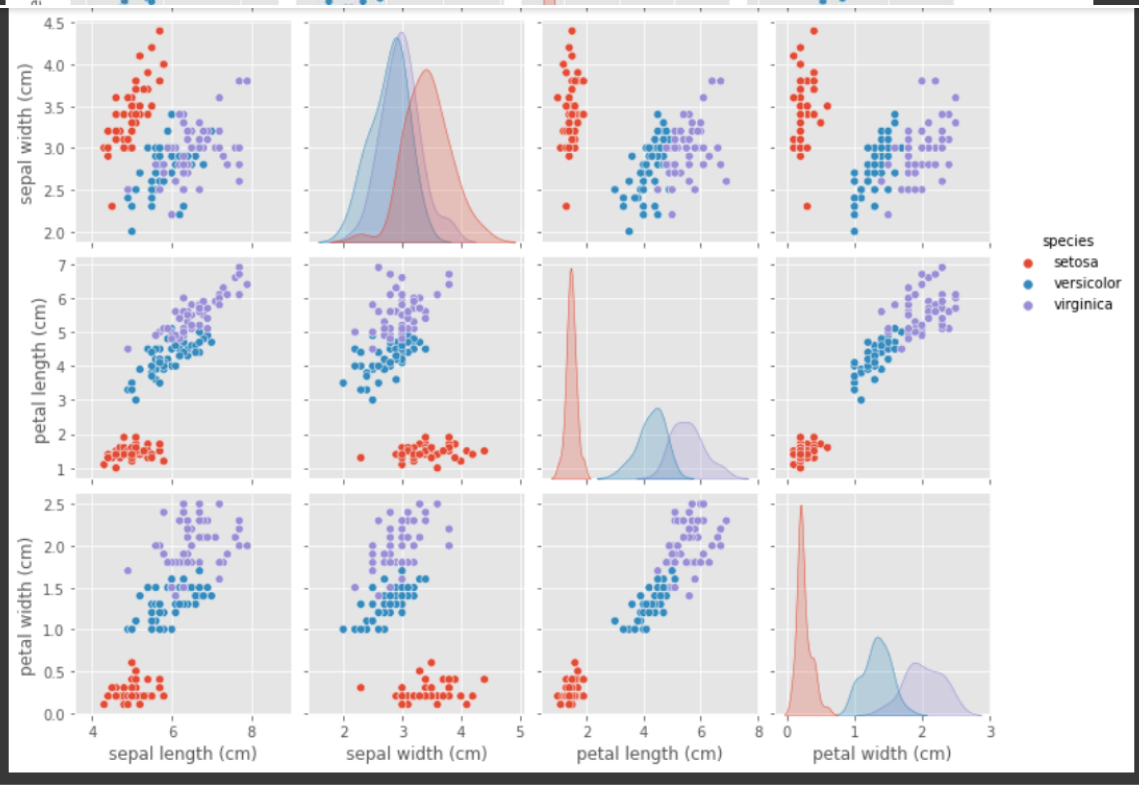
#Visualizing Iris Data

plt.style.use('ggplot')

sns.pairplot(iris_df, hue= 'species')

```
#Visualizing Iris Data
plt.style.use('ggplot')
sns.pairplot(iris_df, hue= 'species')
```

```
<seaborn.axisgrid.PairGrid at 0x7fd54b0dc1d0>
```




```
#Problem: Predict sepal length (cm)
iris_df.drop('species', axis= 1, inplace= True)
target_df = pd.DataFrame(columns= ['species'], data= iris.target)
iris_df = pd.concat([iris_df, target_df], axis= 1)
# Variables
X= iris_df.drop(labels= 'sepal length (cm)', axis= 1)
```

y= iris_df['sepal length (cm)']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.30, random_state= 101)

# Instantiating LinearRegression() Model
lr = LinearRegression()
lr.fit(X_train, y_train)
lr.predict(X_test)

```
#Problem: Predict sepal length (cm)
iris_df.drop('species', axis= 1, inplace= True)
target_df = pd.DataFrame(columns= ['species'], data= iris.target)
iris_df = pd.concat([iris_df, target_df], axis= 1)
# Variables
X= iris_df.drop(labels= 'sepal length (cm)', axis= 1)
y= iris_df['sepal length (cm)']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.30, random_state= 101)
```

```
[ ] # Instantiating LinearRegression() Model
    lr = LinearRegression()
    lr.fit(X_train, y_train)
    lr.predict(X_test)
```

```
array([5.45717101, 5.0736175 , 4.93673765, 6.99370677, 6.54266795,
       5.98776178, 5.68575522, 5.47347099, 5.88042239, 4.6829062 ,
       6.30253641, 5.52654207, 4.90660503, 7.34047628, 6.18132798,
       6.0940002 , 6.0029244 , 6.01620127, 4.7341514 , 6.69220901,
       5.49458357, 5.21235961, 6.03432712, 6.24474908, 6.0940002 ,
       5.54940766, 5.09716102, 5.87131658, 4.83850407, 4.09735861,
       6.65714163, 5.60251513, 6.64812159, 5.70985452, 6.47983902,
       6.185076  , 6.42205169, 5.96853495, 5.88715713, 6.83444425,
       5.10017123, 4.75884287, 4.96754821, 6.47873805, 6.20193176])
```

pred = lr.predict(X_test)
# Evaluating Model's Performance
print('Mean Absolute Error:', mean_absolute_error(y_test, pred))
print('Mean Squared Error:', mean_squared_error(y_test, pred))
print('Mean Root Squared Error:', np.sqrt(mean_squared_error(y_test, pred)))

```
pred = lr.predict(X_test)
# Evaluating Model's Performance
print('Mean Absolute Error:', mean_absolute_error(y_test, pred))
print('Mean Squared Error:', mean_squared_error(y_test, pred))
print('Mean Root Squared Error:', np.sqrt(mean_squared_error(y_test, pred)))
```

```
Mean Absolute Error: 0.26709945889131076
Mean Squared Error: 0.10744247204660265
Mean Root Squared Error: 0.32778418516853836
```

```
#Testing Model
iris_df.loc[6]
d = {'sepal length (cm)' : [4.6],
'sepal width (cm)' : [3.4],
'petal length (cm)' : [1.4],
'petal width (cm)' : [0.3],
'species' : 0}
test_df = pd.DataFrame(data= d)
test_df
X_test = test_df.drop('sepal length (cm)', axis= 1)
y_test = test_df['sepal length (cm)']
lr.predict(X_test)
pred = lr.predict(X_test)
print('Predicted Sepal Length (cm):', pred[0])
print('Actual Sepal Length (cm):', 4.6)
```

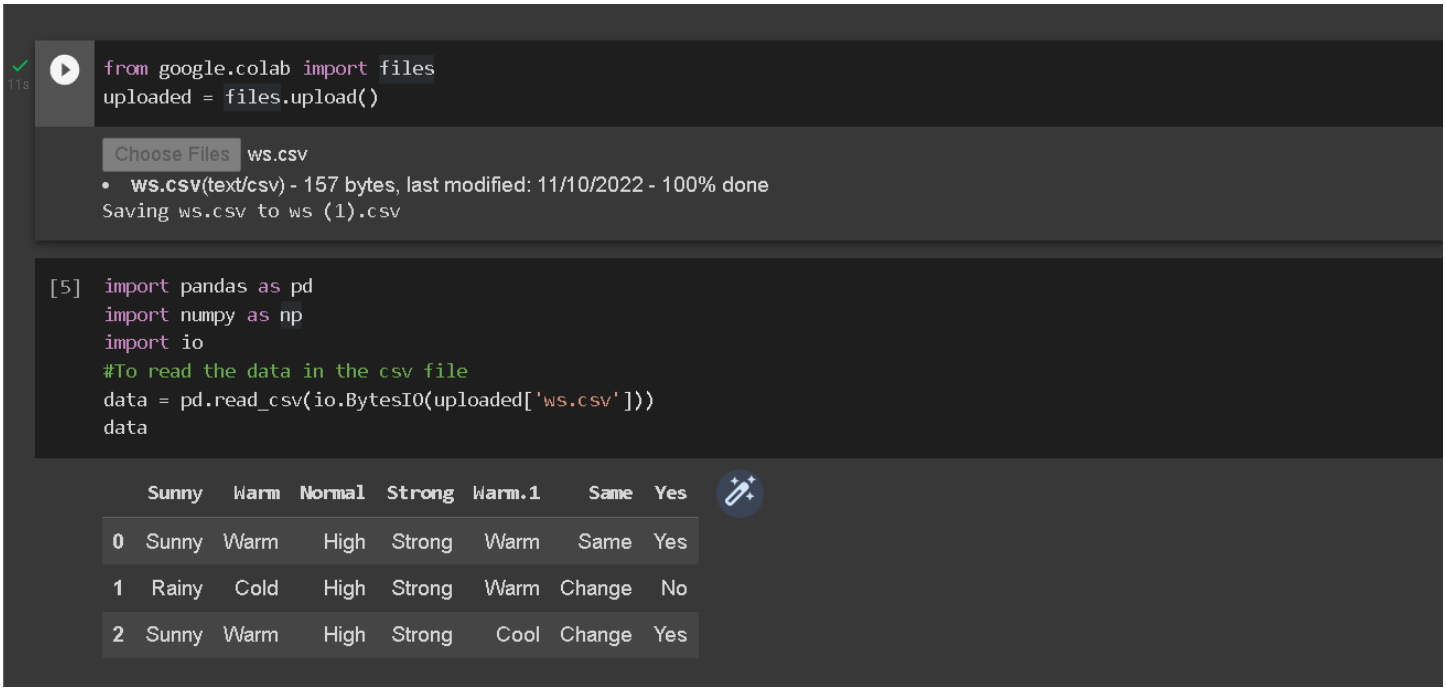
```
Predicted Sepal Length (cm): 4.884900282931658
Actual Sepal Length (cm): 4.6
```

Practical No. 1B

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

Code:

```
from google.colab import files
uploaded = files.upload()

import pandas as pd
import numpy as np
import io
#To read the data in the csv file
data = pd.read_csv(io.BytesIO(uploaded['ws.csv']))
data
```



```
from google.colab import files
uploaded = files.upload()
```

Choose Files ws.csv

- ws.csv(text/csv) - 157 bytes, last modified: 11/10/2022 - 100% done

Saving ws.csv to ws (1).csv

```
[5] import pandas as pd
    import numpy as np
    import io
    #To read the data in the csv file
    data = pd.read_csv(io.BytesIO(uploaded['ws.csv']))
    data
```

	Sunny	Warm	Normal	Strong	Warm.1	Same	Yes
0	Sunny	Warm	High	Strong	Warm	Same	Yes
1	Rainy	Cold	High	Strong	Warm	Change	No
2	Sunny	Warm	High	Strong	Cool	Change	Yes

```
#Making Array of all the atribute
d = np.array(data)[:,:-1]
print("The attributes are :",d)

#Segragating the target that has positive and negative examples
target = np.array(data)[:,-1]
print("The target is: ",target)
```

```
#Making Array of all the atribute
d = np.array(data)[:,:-1]
print("The attributes are :",d)
```

```
The attributes are : [['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
```

```
[7] #Segragating the target that has positive and negative examples
    target = np.array(data)[:,-1]
    print("The target is: ",target)
```

```
The target is:  ['Yes' 'No' 'Yes']
```

```
#Training function to implement find-S algorithm
def train(c,t):
  for i, val in enumerate(t):
    if val == "Yes":
      specific_hypothesis = c[i].copy()
      break

  for i, val in enumerate(c):
    if t[i] == "Yes":
      for x in range(len(specific_hypothesis)):
        if val[x] != specific_hypothesis[x]:
          specific_hypothesis[x] = '?'
        else:
          pass
  return specific_hypothesis

#obtaining the final hypothesis
print("The final hypothesis is:",train(d,target))
```

```
#Training function to implement find-S algorithm
def train(c,t):
  for i, val in enumerate(t):
    if val == "Yes":
      specific_hypothesis = c[i].copy()
      break

  for i, val in enumerate(c):
    if t[i] == "Yes":
      for x in range(len(specific_hypothesis)):
        if val[x] != specific_hypothesis[x]:
          specific_hypothesis[x] = '?'
        else:
          pass
  return specific_hypothesis

#obtaining the final hypothesis
print("The final hypothesis is:",train(d,target))
```

```
The final hypothesis is: ['Sunny' 'Warm' 'High' 'Strong' '?' '?']
```

Practical No. 2A

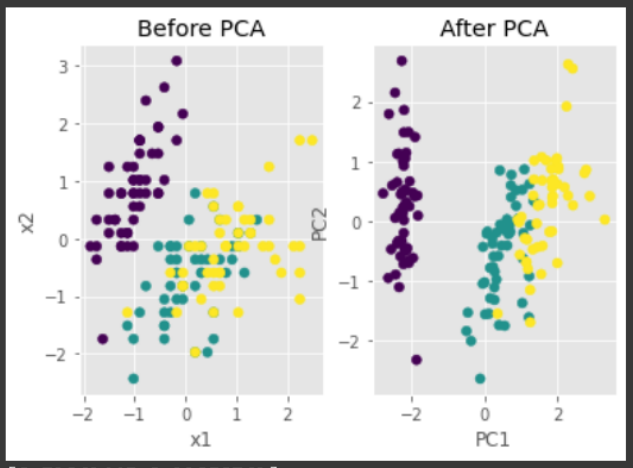
Aim: Perform Data Loading, Feature selection (Principal Component analysis) and Feature Scoring and Ranking.

Code:

```
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) #project the original data into the PCA space
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()
print(pca.explained_variance_ratio_)
```

```
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) #project the original data into the PCA space
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()
print(pca.explained_variance_ratio_)
```




```
[0.72962445 0.22850762]
```

Practical No. 2B

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.

Code:

```
#To uploat dataset in colab
from google.colab import files
uploaded = files.upload()

# Import Standard Libraries
import pandas as pd
import numpy as np
import io

data = pd.read_csv(io.BytesIO(uploaded['ENJOYSPORT.csv']))
data
```

```
#To uploat dataset in colab
from google.colab import files
uploaded = files.upload()
```

Choose Files ENJOYSPORT.csv

- **ENJOYSPORT.csv**(text/csv) - 197 bytes, last modified: 11/10/2022 - 100% done

Saving ENJOYSPORT.csv to ENJOYSPORT (2).csv

```
[3] # Import Standard Libraries
    import pandas as pd
    import numpy as np
    import io
```

```
[4] data = pd.read_csv(io.BytesIO(uploaded['ENJOYSPORT.csv']))
    data
```

	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
0	Sunny	Warm	Normal	Strong	Warm	Same	1
1	Sunny	Warm	High	Strong	Warm	Same	1
2	Rainy	Cold	High	Strong	Warm	Change	0
3	Sunny	Warm	High	Strong	Cool	Change	1

```
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)
```


```
def learn(concepts, target):
  specific_h = concepts[0].copy()
  print("\nInitialization of specific_h and genearal_h")
  print("\nSpecific Boundary: ", specific_h)
  general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
  print("\nGeneric Boundary: ",general_h)
  for i, h in enumerate(concepts):
    print("\nInstance", i+1 , "is ", h)
    if target[i] == "yes":
      print("Instance is Positive ")
      for x in range(len(specific_h)):
        if h[x]!= specific_h[x]:
          specific_h[x] ='?'
          general_h[x][x] ='?'
    if target[i] == "no":
      print("Instance is Negative ")
      for x in range(len(specific_h)):
        if h[x]!= specific_h[x]:
          general_h[x][x] = specific_h[x]
        else:
          general_h[x][x] = '?'
    print("Specific Bundary after ", i+1, "Instance is",specific_h)
    print("Generic Boundary after ", i+1, "Instance is",general_h)
    print("\n")
  indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
  for i in indices:
```

```
    general_h.remove(['?', '?', '?', '?', '?', '?'])
  return specific_h, general_h

s_final, g_final = learn(concepts, target)
print("Final Specific_h: ", s_final, sep="\n")
print("Final General_h: ", g_final, sep="\n")
```

```
def learn(concepts, target):
  specific_h = concepts[0].copy()
  print("\nInitialization of specific_h and genearal_h")
  print("\nSpecific Boundary: ", specific_h)
  general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
  print("\nGeneric Boundary: ",general_h)
  for i, h in enumerate(concepts):
    print("\nInstance", i+1 , "is ", h)
    if target[i] == "yes":
      print("Instance is Positive ")
      for x in range(len(specific_h)):
        if h[x]!= specific_h[x]:
          specific_h[x] ='?'
          general_h[x][x] ='?'
    if target[i] == "no":
      print("Instance is Negative ")
      for x in range(len(specific_h)):
        if h[x]!= specific_h[x]:
          general_h[x][x] = specific_h[x]
        else:
          general_h[x][x] = '?'
    print("Specific Bundary after ", i+1, "Instance is",specific_h)
    print("Generic Boundary after ", i+1, "Instance is",general_h)
    print("\n")
  indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
  for i in indices:
    general_h.remove(['?', '?', '?', '?', '?', '?'])
  return specific_h, general_h
```

```
[10] s_final, g_final = learn(concepts, target)
     print("Final Specific_h: ", s_final, sep="\n")
     print("Final General_h: ", g_final, sep="\n")

     Initialization of specific_h and genearal_h

     Specific Boundary:  ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

     Generic Boundary:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

     Instance 1 is  ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
     Specific Bundary after  1 Instance is ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
     Generic Boundary after  1 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



     Instance 2 is  ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
     Specific Bundary after  2 Instance is ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
     Generic Boundary after  2 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



     Instance 3 is  ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
     Specific Bundary after  3 Instance is ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
     Generic Boundary after  3 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



     Instance 4 is  ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']

     ✓ 9s  completed at 1:24 PM
```

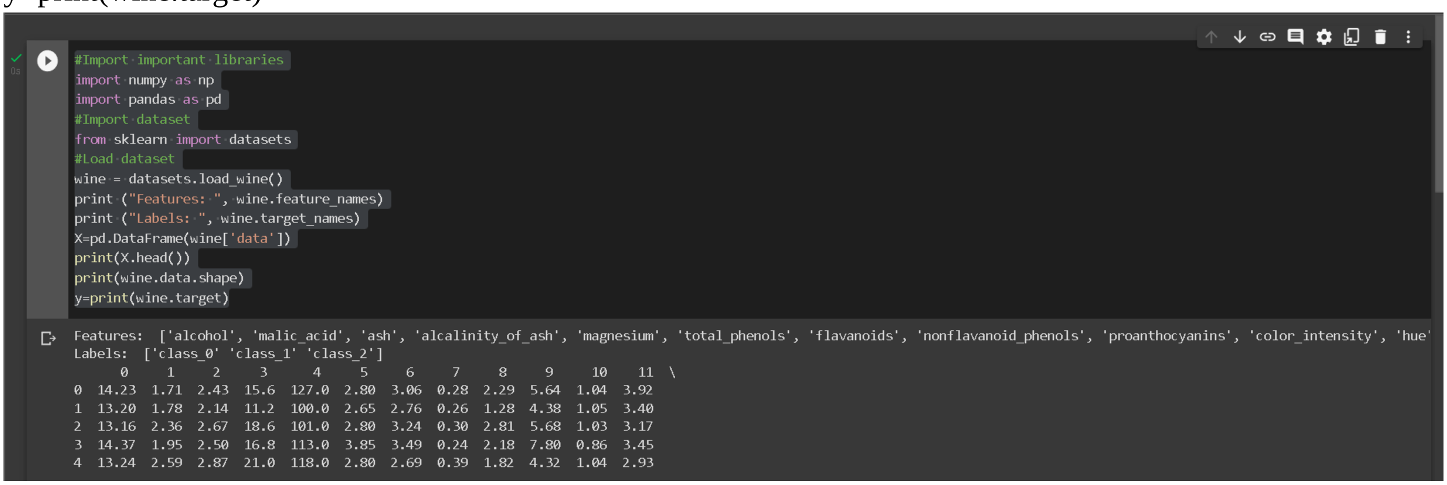
```
[10] Instance 4 is  ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']
     Specific Bundary after  4 Instance is ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
     Generic Boundary after  4 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?'


     Final Specific_h:
     ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
     Final General_h:
     []
```

Practical No. 3A

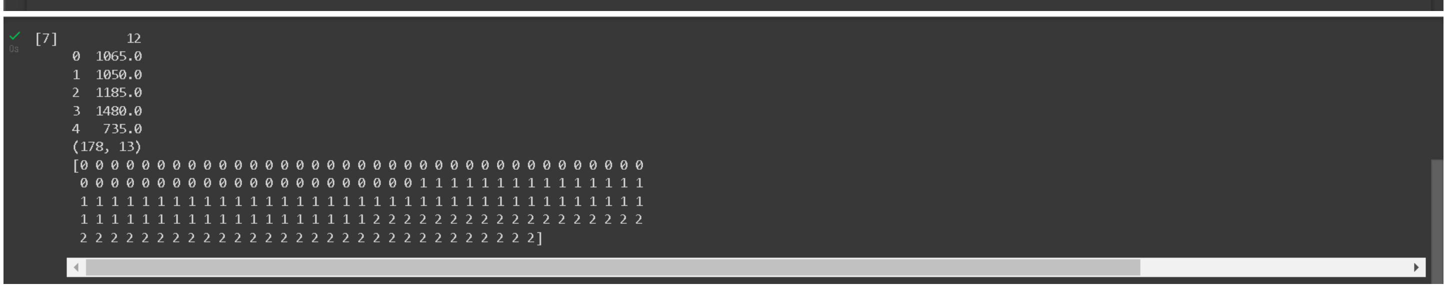
Aim: Write a program to implement the naive Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

Code:

```
#Import important libraries
import numpy as np
import pandas as pd
#Import dataset
from sklearn import datasets
#Load dataset
wine = datasets.load_wine()
print ("Features: ", wine.feature_names)
print ("Labels: ", wine.target_names)
X=pd.DataFrame(wine['data'])
print(X.head())
print(wine.data.shape)
y=print(wine.target)
```



```
#Import important libraries
import numpy as np
import pandas as pd
#Import dataset
from sklearn import datasets
#Load dataset
wine = datasets.load_wine()
print ("Features: ", wine.feature_names)
print ("Labels: ", wine.target_names)
X=pd.DataFrame(wine['data'])
print(X.head())
print(wine.data.shape)
y=print(wine.target)

Features:  ['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue'
Labels:  ['class_0' 'class_1' 'class_2']
       0     1     2     3      4     5     6     7     8     9    10    11  \
0  14.23  1.71  2.43  15.6  127.0  2.80  3.06  0.28  2.29  5.64  1.04  3.92
1  13.20  1.78  2.14  11.2  100.0  2.65  2.76  0.26  1.28  4.38  1.05  3.40
2  13.16  2.36  2.67  18.6  101.0  2.80  3.24  0.30  2.81  5.68  1.03  3.17
3  14.37  1.95  2.50  16.8  113.0  3.85  3.49  0.24  2.18  7.80  0.86  3.45
4  13.24  2.59  2.87  21.0  118.0  2.80  2.69  0.39  1.82  4.32  1.04  2.93
```



```
[7]        12
0  1065.0
1  1050.0
2  1185.0
3  1480.0
4   735.0
(178, 13)
[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 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

# Import train_test_split function
from sklearn.model_selection import train_test_split
# Split dataset into training set and test set

X_train, X_test, y_train, y_test = train_test_split(wine.data, wine.target, test_size=0.30,random_state=109)

#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB
#Create a Gaussian Classifier
gnb = GaussianNB()
#Train the model using the training sets
gnb.fit(X_train, y_train)
#Predict the response for test dataset
y_pred = gnb.predict(X_test)
print(y_pred)

```
# Import train_test_split function
from sklearn.model_selection import train_test_split
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(wine.data, wine.target, test_size=0.30,random_state=109)
```

```
[5] #Import Gaussian Naive Bayes model
    from sklearn.naive_bayes import GaussianNB
    #Create a Gaussian Classifier
    gnb = GaussianNB()
    #Train the model using the training sets
    gnb.fit(X_train, y_train)
    #Predict the response for test dataset
    y_pred = gnb.predict(X_test)
    print(y_pred)
```

```
[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]
```

#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
#confusion matrix
from sklearn.metrics import confusion_matrix
cm=np.array(confusion_matrix(y_test,y_pred))
cm

```
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
#confusion matrix
from sklearn.metrics import confusion_matrix
cm=np.array(confusion_matrix(y_test,y_pred))
cm
```

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

Practical No. 3B

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

Code:-

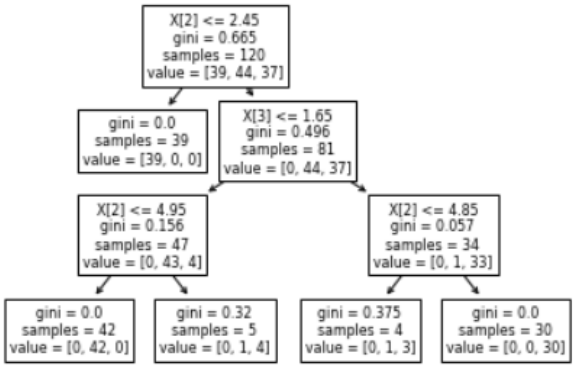
```
import pandas as pd                                          # File Handling
import numpy as np
from sklearn.model_selection import train_test_split         # Splitting Dataset into
from sklearn.tree import DecisionTreeClassifier              # For implementing Decision
from sklearn.metrics import accuracy_score                   # For calculating accuracy
from sklearn.metrics import classification_report            # For evaluating the model
from sklearn import tree

Dataset = pd.read_csv("/content/sample_data/iris.csv")
print(Dataset.head())


Dataset = Dataset.dropna()
Dataset.shape
Dataset["Species"].unique()                          # Unique values of Species
Dataset = Dataset.replace(to_replace ="Iris-setosa", value ="0")
Dataset = Dataset.replace(to_replace ="Iris-versicolor", value ="1")
Dataset = Dataset.replace(to_replace ="Iris-virginica", value ="2")
X = np.array(Dataset[['Sepal.Length', 'Sepal.Width', 'Petal.Length','Petal.Width']]) # Input
Y = np.array(Dataset["Species"])
```

	Unnamed: 0	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	1	5.1	3.5	1.4	0.2	setosa
1	2	4.9	3.0	1.4	0.2	setosa
2	3	4.7	3.2	1.3	0.2	setosa
3	4	4.6	3.1	1.5	0.2	setosa
4	5	5.0	3.6	1.4	0.2	setosa

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state = 100)
clf_gini = DecisionTreeClassifier(criterion = "gini", # Criterion
max_depth = 5,                                # Max Height of Tree
min_samples_leaf = 3,                         # Maximum Leaf samples
random_state = 100)
clf_gini.fit(X_train, Y_train) # Training the Model
y_pred_gini = clf_gini.predict(X_test)


print ("Accuracy : ", accuracy_score(Y_test,y_pred_gini)*100)        # Evaluating predictions with test labels
print ("Report : ", classification_report(Y_test, y_pred_gini))
tree.plot_tree(clf_gini)
```

OUTPUT:-

```
Accuracy :  96.66666666666667
Report :                precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        11
  versicolor       1.00      0.83      0.91         6
   virginica       0.93      1.00      0.96        13

    accuracy                           0.97        30
   macro avg       0.98      0.94      0.96        30
weighted avg       0.97      0.97      0.97        30

[Text(0.375, 0.875, 'X[2] <= 2.45\ngini = 0.665\nsamples = 120\nvalue = [39, 44, 37]'),
 Text(0.25, 0.625, 'gini = 0.0\nsamples = 39\nvalue = [39, 0, 0]'),
 Text(0.5, 0.625, 'X[3] <= 1.65\ngini = 0.496\nsamples = 81\nvalue = [0, 44, 37]'),
 Text(0.25, 0.375, 'X[2] <= 4.95\ngini = 0.156\nsamples = 47\nvalue = [0, 43, 4]'),
 Text(0.125, 0.125, 'gini = 0.0\nsamples = 42\nvalue = [0, 42, 0]'),
 Text(0.375, 0.125, 'gini = 0.32\nsamples = 5\nvalue = [0, 1, 4]'),
 Text(0.75, 0.375, 'X[2] <= 4.85\ngini = 0.057\nsamples = 34\nvalue = [0, 1, 33]'),
 Text(0.625, 0.125, 'gini = 0.375\nsamples = 4\nvalue = [0, 1, 3]'),
 Text(0.875, 0.125, 'gini = 0.0\nsamples = 30\nvalue = [0, 0, 30]')]
```

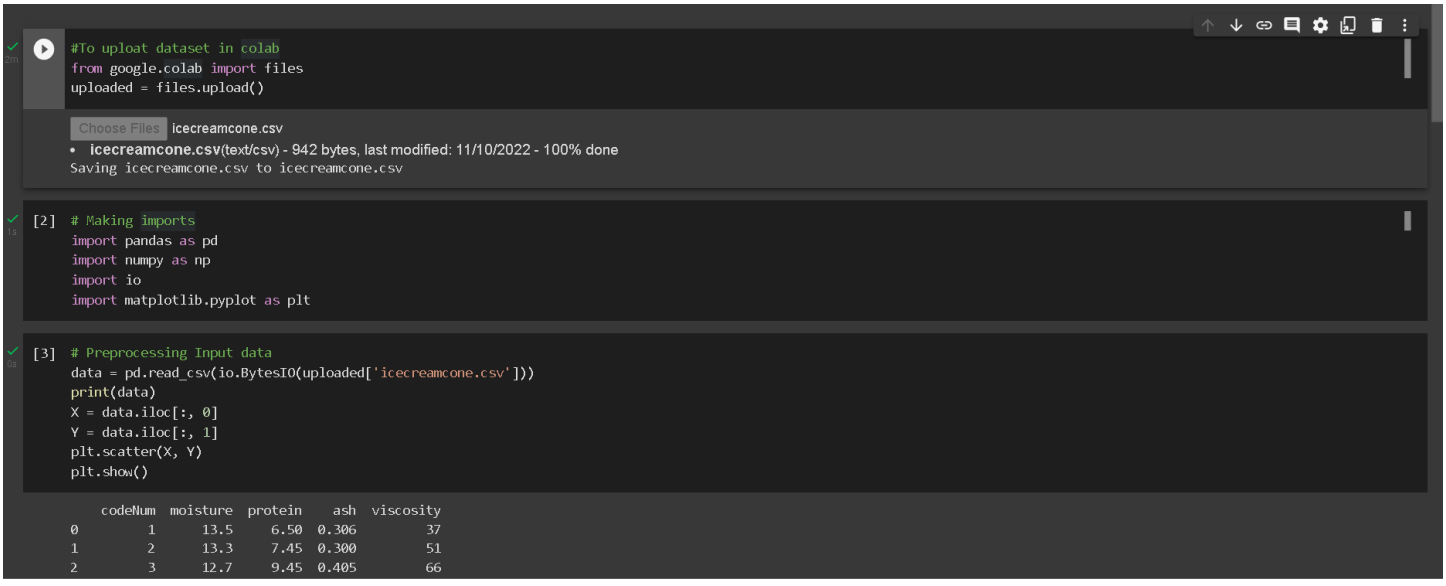


Practical No. 4A

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

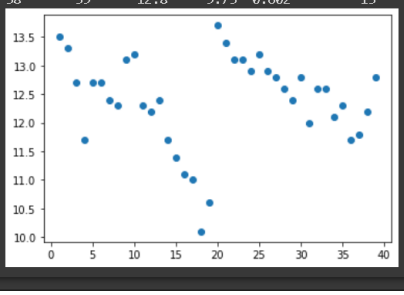
Code:

```
#To uploat dataset in colab
from google.colab import files
uploaded = files.upload()

# Making imports
import pandas as pd
import numpy as np
import io
import matplotlib.pyplot as plt

# Preprocessing Input data
data = pd.read_csv(io.BytesIO(uploaded['icecreamcone.csv']))
print(data)
X = data.iloc[:, 0]
Y = data.iloc[:, 1]
plt.scatter(X, Y)
plt.show()
```



3	4	11.7	12.05	0.940	15
4	5	12.7	6.25	0.277	30
5	6	12.7	7.20	0.287	52
6	7	12.4	7.20	0.290	43
7	8	12.3	8.30	0.300	65
8	9	13.1	8.35	0.298	73
9	10	13.2	9.00	0.310	93
10	11	12.3	9.00	0.305	89
11	12	12.2	10.60	0.356	119
12	13	12.4	8.15	0.440	38
13	14	11.7	9.35	0.460	64
14	15	11.4	10.55	0.538	107
15	16	11.1	10.90	0.740	23
16	17	11.0	11.50	0.840	30
17	18	10.1	10.80	1.120	15
18	19	10.6	11.40	1.080	19
19	20	13.7	7.30	0.270	45
20	21	13.4	8.90	0.362	73
21	22	13.1	7.30	0.302	40
22	23	13.1	8.05	0.324	57
23	24	12.9	9.30	0.514	52
24	25	13.2	8.85	0.328	66
25	26	12.9	9.00	0.304	70
26	27	12.8	9.25	0.336	84
27	28	12.6	9.75	0.332	96
28	29	12.4	9.85	0.362	97
29	30	12.8	10.85	0.460	116
30	31	12.0	9.20	0.616	12

31	32	12.6	13.10	1.208	29
32	33	12.6	10.40	0.498	93
33	34	12.1	12.20	0.670	77
34	35	12.3	12.50	1.416	32
35	36	11.7	13.90	1.410	40
36	37	11.8	12.15	1.264	30
37	38	12.2	13.50	1.300	38
38	39	12.8	9.75	0.602	15



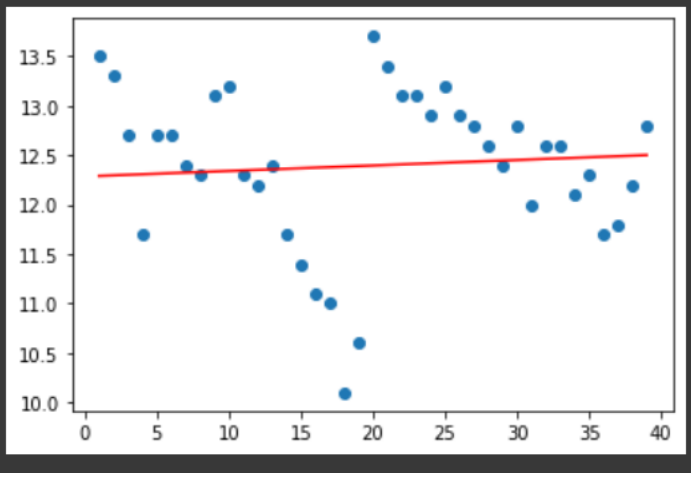

```
# Building the model
X_mean = np.mean(X)
Y_mean = np.mean(Y)
num = 0
den = 0
for i in range(len(X)):
  num += (X[i] - X_mean)*(Y[i] - Y_mean)
  den += (X[i] - X_mean)**2
m = num / den
c = Y_mean - m*X_mean

print (m, c)
```

```
# Building the model
X_mean = np.mean(X)
Y_mean = np.mean(Y)
num = 0
den = 0
for i in range(len(X)):
  num += (X[i] - X_mean)*(Y[i] - Y_mean)
  den += (X[i] - X_mean)**2
m = num / den
c = Y_mean - m*X_mean

print (m, c)
```

```
-0.005526315789473676 12.505398110661268
```

# Making predictions
Y_pred = m*X + c
plt.scatter(X, Y)
# actual
plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color='red')
# predicted
plt.show()

```
# Making predictions
Y_pred = m*X + c

plt.scatter(X, Y)
# actual
plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color='red')
# predicted
plt.show()
```

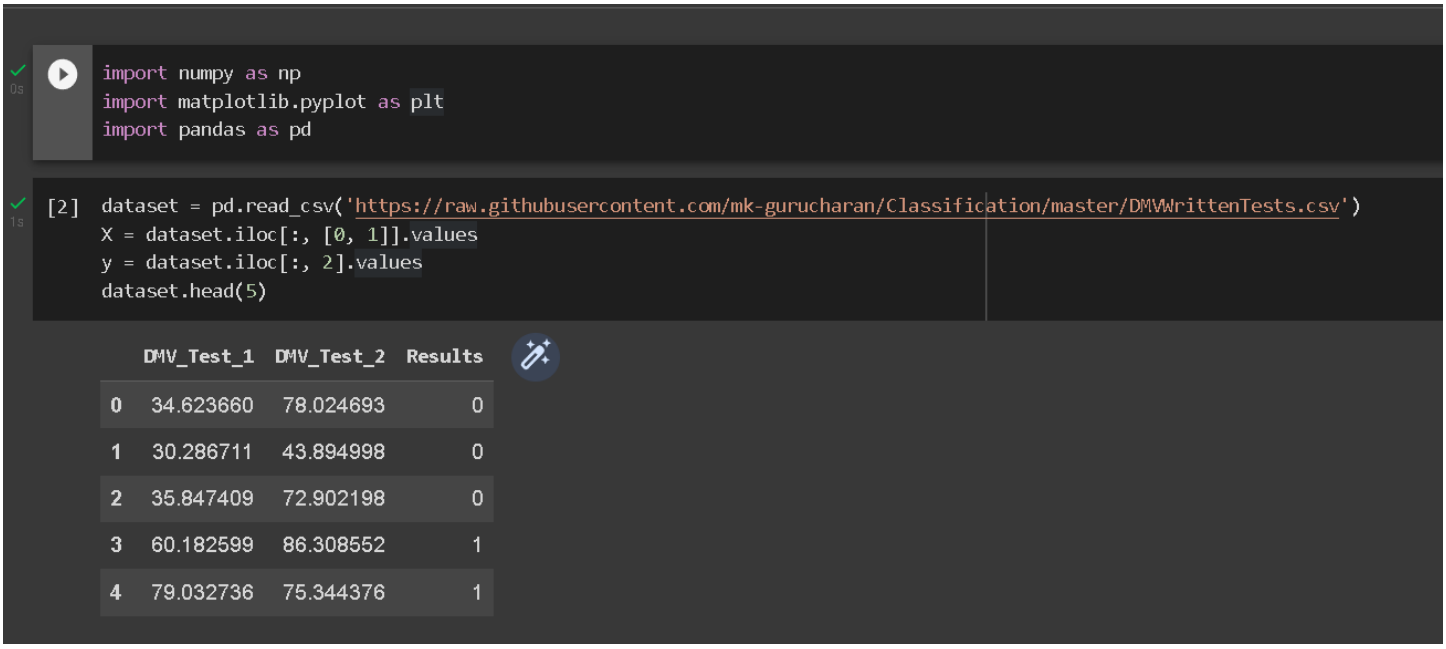


Practical No. 4B

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

Code:

```
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)
```



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

```
[2] 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)

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

```
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)
```

```
[4] from sklearn.linear_model import LogisticRegression
    classifier = LogisticRegression()
    classifier.fit(X_train, y_train)
```

```
LogisticRegression()
```

```
[5] y_pred = classifier.predict(X_test)
    y_pred
```

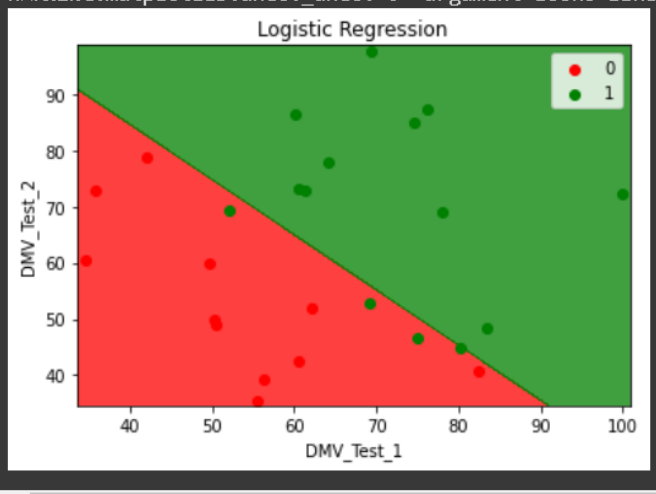
```
array([0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0,
       1, 1, 0])
```

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
              c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Logistic Regression')
plt.xlabel('DMV_Test_1')
plt.ylabel('DMV_Test_2')
plt.legend()
plt.show()
```

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
              c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Logistic Regression')
plt.xlabel('DMV_Test_1')
plt.ylabel('DMV_Test_2')
plt.legend()
plt.show()
```

```
WARNING:matplotlib.axes._axes:*c* argument looks like a single numeric RGB or RGBA sequence, which should be
```

```
WARNING:matplotlib.axes._axes:*c* argument looks like a single numeric RGB or RGBA sequence, which should be
WARNING:matplotlib.axes._axes:*c* argument looks like a single numeric RGB or RGBA sequence, which should be
```



Practical No. 5A

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.

Code:

```
from google.colab import files
uploaded = files.upload()
import io

import pandas as pd
import numpy as np

train_data_m = pd.read_csv(io.BytesIO(uploaded['PlayTennis.csv']))
train_data_m.head()
```

```
from google.colab import files
uploaded = files.upload()
import io
```

Choose Files PlayTennis.csv

- **PlayTennis.csv**(text/csv) - 419 bytes, last modified: 11/10/2022 - 100% done

Saving PlayTennis.csv to PlayTennis.csv

```
[2] import pandas as pd
    import numpy as np
```

```
[3] train_data_m = pd.read_csv(io.BytesIO(uploaded['PlayTennis.csv']))
    train_data_m.head()
```

	Outlook	Temperature	Humidity	Wind	Play Tennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes

```
def evaluate(tree, test_data_m, label):
  correct_preditct = 0
  wrong_preditct = 0
  for index, row in test_data_m.iterrows():
    result = predict(tree, test_data_m.iloc[index])
    if result == test_data_m[label].iloc[index]:
      correct_preditct += 1
    else:
      wrong_preditct += 1
  accuracy = correct_preditct / (correct_preditct + wrong_preditct)
  return accuracy

def calc_total_entropy(train_data, label, class_list):
  total_row = train_data.shape[0]
  total_entr = 0
  for c in class_list:
    total_class_count = train_data[train_data[label] == c].shape[0]
    total_class_entr = -(total_class_count/total_row)*np.log2(total_class_count/total_row)
    total_entr += total_class_entr
  return total_entr
```

```
def evaluate(tree, test_data_m, label):
  correct_preditct = 0
  wrong_preditct = 0
  for index, row in test_data_m.iterrows():
    result = predict(tree, test_data_m.iloc[index])
    if result == test_data_m[label].iloc[index]:
      correct_preditct += 1
    else:
      wrong_preditct += 1
  accuracy = correct_preditct / (correct_preditct + wrong_preditct)
  return accuracy
```

```
[5] def calc_total_entropy(train_data, label, class_list):
      total_row = train_data.shape[0]
      total_entr = 0
      for c in class_list:
        total_class_count = train_data[train_data[label] == c].shape[0]
        total_class_entr = -(total_class_count/total_row)*np.log2(total_class_count/total_row)
        total_entr += total_class_entr
      return total_entr
```

```
def calc_entropy(feature_value_data, label, class_list):
  class_count = feature_value_data.shape[0]
  entropy = 0
  for c in class_list:
    label_class_count = feature_value_data[feature_value_data[label] == c].shape[0]
    entropy_class = 0
  if label_class_count != 0:
    probability_class = label_class_count/class_count
    entropy_class = - probability_class * np.log2(probability_class)
    entropy += entropy_class
  return entropy

def calc_info_gain(feature_name, train_data, label, class_list):
  feature_value_list = train_data[feature_name].unique()
  total_row = train_data.shape[0]
  feature_info = 0.0
  for feature_value in feature_value_list:
    feature_value_data = train_data[train_data[feature_name] == feature_value]
    feature_value_count = feature_value_data.shape[0]
    feature_value_entropy = calc_entropy(feature_value_data, label, class_list)
    feature_value_probability = feature_value_count/total_row
    feature_info += feature_value_probability * feature_value_entropy
  return calc_total_entropy(train_data, label, class_list) - feature_info
```

```
def calc_entropy(feature_value_data, label, class_list):
  class_count = feature_value_data.shape[0]
  entropy = 0
  for c in class_list:
    label_class_count = feature_value_data[feature_value_data[label] == c].shape[0]
    entropy_class = 0
  if label_class_count != 0:
    probability_class = label_class_count/class_count
    entropy_class = - probability_class * np.log2(probability_class)
    entropy += entropy_class
  return entropy
```

```
[7]  def calc_info_gain(feature_name, train_data, label, class_list):
       feature_value_list = train_data[feature_name].unique()
       total_row = train_data.shape[0]
       feature_info = 0.0
       for feature_value in feature_value_list:
         feature_value_data = train_data[train_data[feature_name] == feature_value]
         feature_value_count = feature_value_data.shape[0]
         feature_value_entropy = calc_entropy(feature_value_data, label, class_list)
         feature_value_probability = feature_value_count/total_row
         feature_info += feature_value_probability * feature_value_entropy
       return calc_total_entropy(train_data, label, class_list) - feature_info
```

```
def find_most_informative_feature(train_data, label, class_list):
  feature_list = train_data.columns.drop(label)
  max_info_gain = -1
  max_info_feature = None
  for feature in feature_list:
    feature_info_gain = calc_info_gain(feature, train_data, label, class_list)
    if max_info_gain < feature_info_gain:
      max_info_gain = feature_info_gain
      max_info_feature = feature
  return max_info_feature

def generate_sub_tree(feature_name, train_data, label, class_list):
  feature_value_count_dict = train_data[feature_name].value_counts(sort=False)
  tree = {}
  for feature_value, count in feature_value_count_dict.iteritems():
    feature_value_data = train_data[train_data[feature_name] == feature_value]
    assigned_to_node = False
    for c in class_list:
      class_count = feature_value_data[feature_value_data[label] == c].shape[0]
      if class_count == count:
        tree[feature_value] = c
```

```
      train_data = train_data[train_data[feature_name] != feature_value]
      assigned_to_node = True
    if not assigned_to_node:
      tree[feature_value] = "?"
  return tree, train_data
```

```
def find_most_informative_feature(train_data, label, class_list):
  feature_list = train_data.columns.drop(label)
  max_info_gain = -1
  max_info_feature = None
  for feature in feature_list:
    feature_info_gain = calc_info_gain(feature, train_data, label, class_list)
    if max_info_gain < feature_info_gain:
      max_info_gain = feature_info_gain
      max_info_feature = feature
  return max_info_feature
```

```
[9] def generate_sub_tree(feature_name, train_data, label, class_list):
      feature_value_count_dict = train_data[feature_name].value_counts(sort=False)
      tree = {}
      for feature_value, count in feature_value_count_dict.iteritems():
        feature_value_data = train_data[train_data[feature_name] == feature_value]
        assigned_to_node = False
        for c in class_list:
          class_count = feature_value_data[feature_value_data[label] == c].shape[0]
          if class_count == count:
            tree[feature_value] = c
            train_data = train_data[train_data[feature_name] != feature_value]
            assigned_to_node = True
          if not assigned_to_node:
            tree[feature_value] = "?"
      return tree, train_data
```

```
def make_tree(root, prev_feature_value, train_data, label, class_list):
  if train_data.shape[0] != 0:
    max_info_feature = find_most_informative_feature(train_data, label, class_list)
    tree, train_data = generate_sub_tree(max_info_feature, train_data, label,class_list)
    next_root = None
    if prev_feature_value != None:
      root[prev_feature_value] = dict()
      root[prev_feature_value][max_info_feature] = tree
      next_root = root[prev_feature_value][max_info_feature]
    else:
      root[max_info_feature] = tree
      next_root = root[max_info_feature]
    for node, branch in list(next_root.items()):
      if branch == "?":
        feature_value_data = train_data[train_data[max_info_feature] == node]
        make_tree(next_root, node, feature_value_data, label, class_list)

def predict(tree, instance):
  if not isinstance(tree, dict):
```

```
    return tree
  else:
    root_node = next(iter(tree))
    feature_value = instance[root_node]
    if feature_value in tree[root_node]:
      return predict(tree[root_node][feature_value], instance)
    else:
      return None
```

```
def make_tree(root, prev_feature_value, train_data, label, class_list):
  if train_data.shape[0] != 0:
    max_info_feature = find_most_informative_feature(train_data, label, class_list)
    tree, train_data = generate_sub_tree(max_info_feature, train_data, label,class_list)
    next_root = None
    if prev_feature_value != None:
      root[prev_feature_value] = dict()
      root[prev_feature_value][max_info_feature] = tree
      next_root = root[prev_feature_value][max_info_feature]
    else:
      root[max_info_feature] = tree
      next_root = root[max_info_feature]
    for node, branch in list(next_root.items()):
      if branch == "?":
         feature_value_data = train_data[train_data[max_info_feature] == node]
         make_tree(next_root, node, feature_value_data, label, class_list)
```

```
[11] def predict(tree, instance):
       if not isinstance(tree, dict):
         return tree
       else:
         root_node = next(iter(tree))
         feature_value = instance[root_node]
         if feature_value in tree[root_node]:
           return predict(tree[root_node][feature_value], instance)
         else:
           return None
```

```
def id3(train_data_m, label):
  train_data = train_data_m.copy()
  tree = {}
  class_list = train_data[label].unique()
  make_tree(tree, None, train_data_m, label, class_list)
  return tree

tree = id3(train_data_m, 'Play Tennis')

test_data_m = pd.read_csv(io.BytesIO(uploaded['PlayTennis.csv']))
accuracy = evaluate(tree, test_data_m, 'Play Tennis')
accuracy
```

```
def id3(train_data_m, label):
  train_data = train_data_m.copy()
  tree = {}
  class_list = train_data[label].unique()
  make_tree(tree, None, train_data_m, label, class_list)
  return tree

tree = id3(train_data_m, 'Play Tennis')
```

```
[13] test_data_m = pd.read_csv(io.BytesIO(uploaded['PlayTennis.csv']))
     accuracy = evaluate(tree, test_data_m, 'Play Tennis')
     accuracy
```

```
1.0
```

Practical No. 5B

Aim: Write a program to implement K-Nearest Neighbour algorithm to classify the iris data set.

Code:

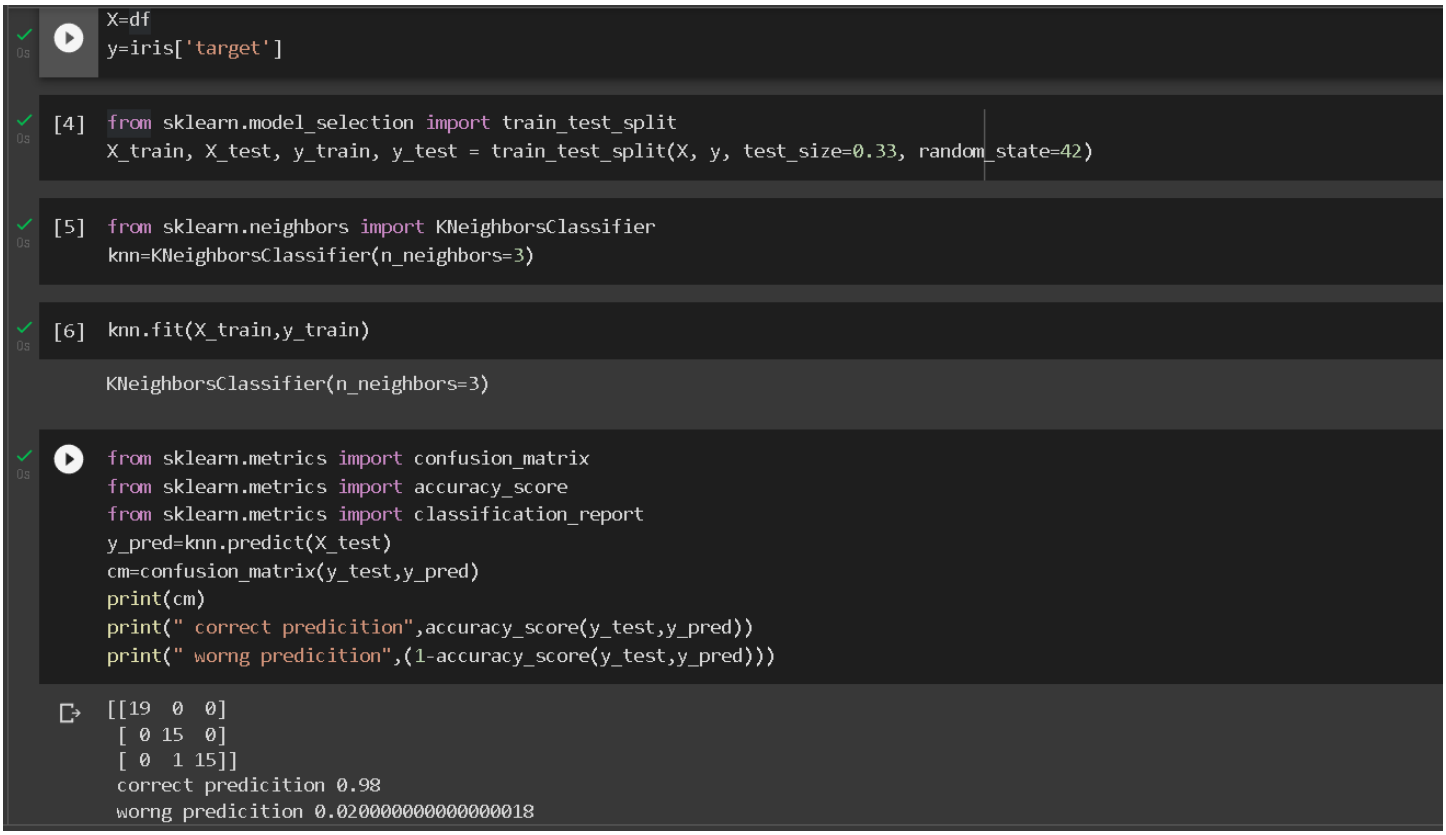
```
import sklearn
import pandas as pd
from sklearn.datasets import load_iris
iris=load_iris()
iris.keys()
df=pd.DataFrame(iris['data'])
print(df)
print(iris['target_names'])
iris['feature_names']
```

```
import sklearn
import pandas as pd
from sklearn.datasets import load_iris
iris=load_iris()
iris.keys()
df=pd.DataFrame(iris['data'])
print(df)
print(iris['target_names'])
iris['feature_names']
```

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

[150 rows x 4 columns]
['setosa' 'versicolor' 'virginica']
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

```
X=df
y=iris['target']

from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train,y_train)

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
y_pred=knn.predict(X_test)
cm=confusion_matrix(y_test,y_pred)
print(cm)
print(" correct predicition",accuracy_score(y_test,y_pred))
print(" worng predicition",(1-accuracy_score(y_test,y_pred)))
```



```
X=df
y=iris['target']

[4] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

[5] from sklearn.neighbors import KNeighborsClassifier
    knn=KNeighborsClassifier(n_neighbors=3)

[6] knn.fit(X_train,y_train)

    KNeighborsClassifier(n_neighbors=3)

    from sklearn.metrics import confusion_matrix
    from sklearn.metrics import accuracy_score
    from sklearn.metrics import classification_report
    y_pred=knn.predict(X_test)
    cm=confusion_matrix(y_test,y_pred)
    print(cm)
    print(" correct predicition",accuracy_score(y_test,y_pred))
    print(" worng predicition",(1-accuracy_score(y_test,y_pred)))

    [[19  0  0]
     [ 0 15  0]
     [ 0  1 15]]
     correct predicition 0.98
     worng predicition 0.020000000000000018
```

Practical No. 6A

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

Code:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets

iris=datasets.load_iris()

x = iris.data
y = iris.target
print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
print(x)
print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')
print(y)
```

```
sepal-length sepal-width petal-length petal-width
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
```

```
[3]
0s
       [7.2 3.  5.8 1.6]
       [7.4 2.8 6.1 1.9]
       [7.9 3.8 6.4 2. ]
       [6.4 2.8 5.6 2.2]
       [6.3 2.8 5.1 1.5]
       [6.1 2.6 5.6 1.4]
       [7.7 3.  6.1 2.3]
       [6.3 3.4 5.6 2.4]
       [6.4 3.1 5.5 1.8]
       [6.  3.  4.8 1.8]
       [6.9 3.1 5.4 2.1]
       [6.7 3.1 5.6 2.4]
       [6.9 3.1 5.1 2.3]
       [5.8 2.7 5.1 1.9]
       [6.8 3.2 5.9 2.3]
       [6.7 3.3 5.7 2.5]
       [6.7 3.  5.2 2.3]
       [6.3 2.5 5.  1.9]
       [6.5 3.  5.2 2. ]
       [6.2 3.4 5.4 2.3]
       [5.9 3.  5.1 1.8]]
      class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica
      [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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
       1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
       2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
       2 2]
```

```
x_train, x_test, y_train, y_test =train_test_split(x,y,test_size=0.3)
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)

y_pred=classifier.predict(x_test)
print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

```
x_train, x_test, y_train, y_test =train_test_split(x,y,test_size=0.3)
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)
```

```
KNeighborsClassifier()
```

```
[12] y_pred=classifier.predict(x_test)
```

```
[13] print('Confusion Matrix')
     print(confusion_matrix(y_test,y_pred))
     print('Accuracy Metrics')
     print(classification_report(y_test,y_pred))
```

```
Confusion Matrix
[[11  0  0]
 [ 0 16  1]
 [ 0  0 17]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       1.00      0.94      0.97        17
           2       0.94      1.00      0.97        17

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

Practical No. 6B

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

Code:

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np

iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']

y = pd.DataFrame(iris.target)
y.columns = ['Targets']
model = KMeans(n_clusters=3)
model.fit(X)
```

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np
```

```
[2] iris = datasets.load_iris()
    X = pd.DataFrame(iris.data)
    X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
```

```
[3] y = pd.DataFrame(iris.target)
    y.columns = ['Targets']
    model = KMeans(n_clusters=3)
    model.fit(X)
```

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

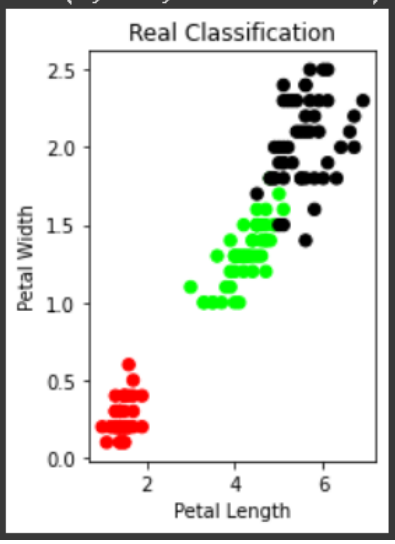
```
plt.figure(figsize=(14,7))
colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
```

```
plt.figure(figsize=(14,7))
colormap = np.array(['red', 'lime', 'black'])
```

```
<Figure size 1008x504 with 0 Axes>
```

```
[5] # Plot the Original Classifications
    plt.subplot(1, 2, 1)
    plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
    plt.title('Real Classification')
    plt.xlabel('Petal Length')
    plt.ylabel('Petal Width')
```

```
Text(0, 0.5, 'Petal Width')
```

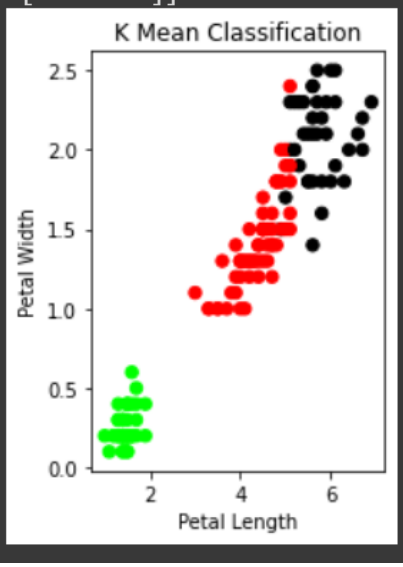



```
# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.labels_))
```

print('The Confusion matrixof K-Mean: ',sm.confusion_matrix(y, model.labels_))

```
# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean: ',sm.confusion_matrix(y, model.labels_))
```

```
The accuracy score of K-Mean:  0.24
The Confusion matrixof K-Mean:  [[ 0 50  0]
 [48  0  2]
 [14  0 36]]
```




```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)

#xs.sample(5)
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
y_gmm = gmm.predict(xs)

#y_cluster_gmm
plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_gmm], s=40)
plt.title('GMM Classification')
```

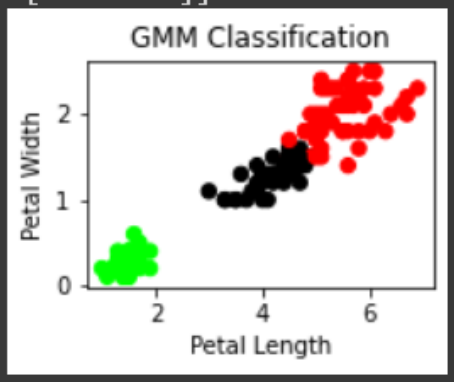
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The accuracy score of EM: ',sm.accuracy_score(y, y_gmm))
print('The Confusion matrix of EM: ',sm.confusion_matrix(y, y_gmm))

```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
```

```
[10] #xs.sample(5)
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
y_gmm = gmm.predict(xs)
```

```
#y_cluster_gmm
plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_gmm], s=40)
plt.title('GMM Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The accuracy score of EM: ',sm.accuracy_score(y, y_gmm))
print('The Confusion matrix of EM: ',sm.confusion_matrix(y, y_gmm))
```

```
The accuracy score of EM:  0.0
The Confusion matrix of EM:  [[ 0 50  0]
 [ 5  0 45]
 [50  0  0]]
```

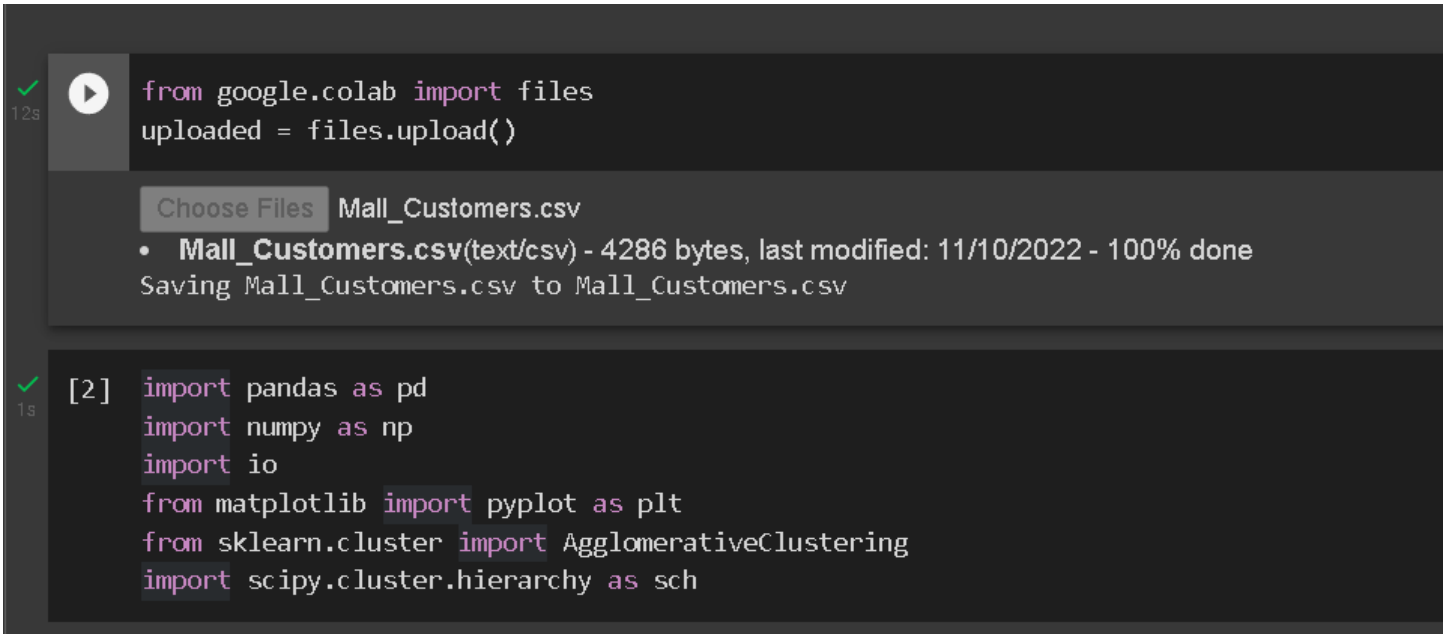


Practical No. 7A

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

Code:

```
from google.colab import files
uploaded = files.upload()

import pandas as pd
import numpy as np
import io
from matplotlib import pyplot as plt
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as sch
```



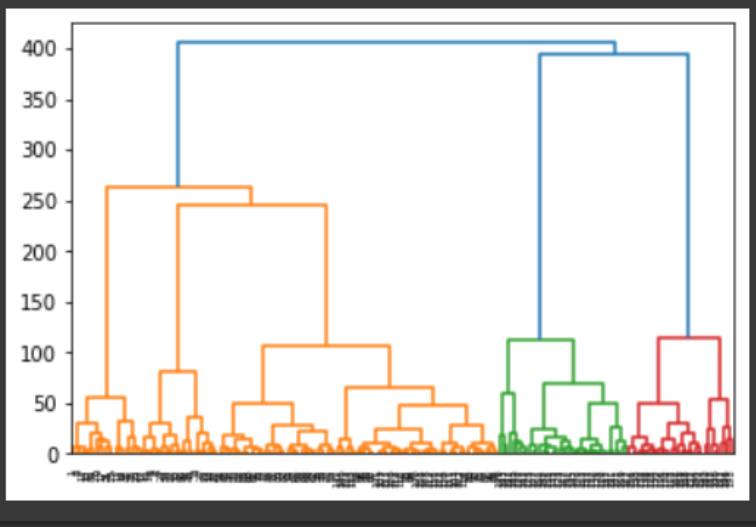

```
dataset = pd.read_csv(io.BytesIO(uploaded['Mall_Customers.csv']))
dataset

X = dataset.iloc[:, [3, 4]].values
dendrogram = sch.dendrogram(sch.linkage(X, method='ward'))
```

```
dataset = pd.read_csv(io.BytesIO(uploaded['Mall_Customers.csv']))
dataset
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
...	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

200 rows × 5 columns

```
X = dataset.iloc[:, [3, 4]].values
dendrogram = sch.dendrogram(sch.linkage(X, method='ward'))
```

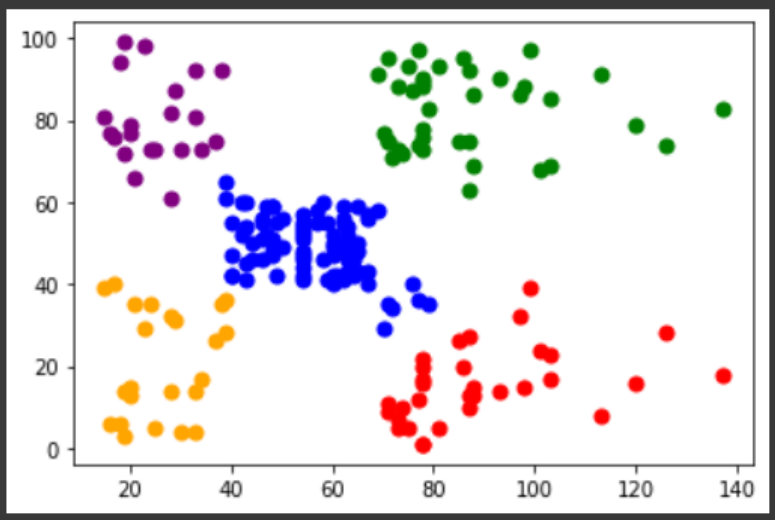



```
model = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
model.fit(X)
labels = model.labels_
```

```
plt.scatter(X[labels==0, 0], X[labels==0, 1], s=50, marker='o', color='red')
plt.scatter(X[labels==1, 0], X[labels==1, 1], s=50, marker='o', color='blue')
plt.scatter(X[labels==2, 0], X[labels==2, 1], s=50, marker='o', color='green')
plt.scatter(X[labels==3, 0], X[labels==3, 1], s=50, marker='o', color='purple')
plt.scatter(X[labels==4, 0], X[labels==4, 1], s=50, marker='o', color='orange')
plt.show()
```

```
model = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
model.fit(X)
labels = model.labels_
```

```
[6] plt.scatter(X[labels==0, 0], X[labels==0, 1], s=50, marker='o', color='red')
    plt.scatter(X[labels==1, 0], X[labels==1, 1], s=50, marker='o', color='blue')
    plt.scatter(X[labels==2, 0], X[labels==2, 1], s=50, marker='o', color='green')
    plt.scatter(X[labels==3, 0], X[labels==3, 1], s=50, marker='o', color='purple')
    plt.scatter(X[labels==4, 0], X[labels==4, 1], s=50, marker='o', color='orange')
    plt.show()
```



Practical No. 7B

Aim: Implement the Rule based method and test the same.

Code:-

```
import numpy as np
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
df = pd.read_csv('/content/sample_data/retail_dataset.csv')
df.head()
```

	0	1	2	3	4	5	6
0	Bread	Wine	Eggs	Meat	Cheese	Pencil	Diaper
1	Bread	Cheese	Meat	Diaper	Wine	Milk	Pencil
2	Cheese	Meat	Eggs	Milk	Wine	NaN	NaN
3	Cheese	Meat	Eggs	Milk	Wine	NaN	NaN
4	Meat	Pencil	Wine	NaN	NaN	NaN	NaN

```
items = (df['0'].unique())
items
```

```
array(['Bread', 'Cheese', 'Meat', 'Eggs', 'Wine', 'Bagel', 'Pencil',
       'Diaper', 'Milk'], dtype=object)
```

```
encoded_vals = []
for index, row in df.iterrows():
  labels = {}
```

```
uncommons = list(set(items) - set(row))
commons = list(set(items).intersection(row))
for uc in uncommons:
  labels[uc] = 0
for com in commons:
  labels[com] = 1
encoded_vals.append(labels)
ohe_df = pd.DataFrame(encoded_vals)
ohe_df
```

	Pencil	Cheese	Diaper	Milk	Eggs	Bread	Bagel	Meat	Wine
0	0	0	0	0	1	1	1	1	1

```
freq_items = apriori(ohe_df, min_support = 0.2, use_colnames = True)
freq_items.head()
```

	support	itemsets
0	1.0	(Eggs)
1	1.0	(Bread)
2	1.0	(Bagel)
3	1.0	(Meat)
4	1.0	(Wine)

```
association_rules(freq_items, metric = "confidence", min_threshold = 0.6)
```

OUTPUT:-

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Bread)	(Eggs)	1.0	1.0	1.0	1.0	1.0	0.0	inf
1	(Eggs)	(Bread)	1.0	1.0	1.0	1.0	1.0	0.0	inf
2	(Eggs)	(Bagel)	1.0	1.0	1.0	1.0	1.0	0.0	inf
3	(Bagel)	(Eggs)	1.0	1.0	1.0	1.0	1.0	0.0	inf
4	(Eggs)	(Meat)	1.0	1.0	1.0	1.0	1.0	0.0	inf
...	...	...	...	...	...	...	...	...	...
175	(Eggs)	(Bread, Bagel, Meat, Wine)	1.0	1.0	1.0	1.0	1.0	0.0	inf
176	(Bread)	(Eggs, Bagel, Meat, Wine)	1.0	1.0	1.0	1.0	1.0	0.0	inf
177	(Bagel)	(Bread, Eggs, Meat, Wine)	1.0	1.0	1.0	1.0	1.0	0.0	inf
178	(Meat)	(Bread, Eggs, Bagel, Wine)	1.0	1.0	1.0	1.0	1.0	0.0	inf
179	(Wine)	(Bread, Eggs, Bagel, Meat)	1.0	1.0	1.0	1.0	1.0	0.0	inf

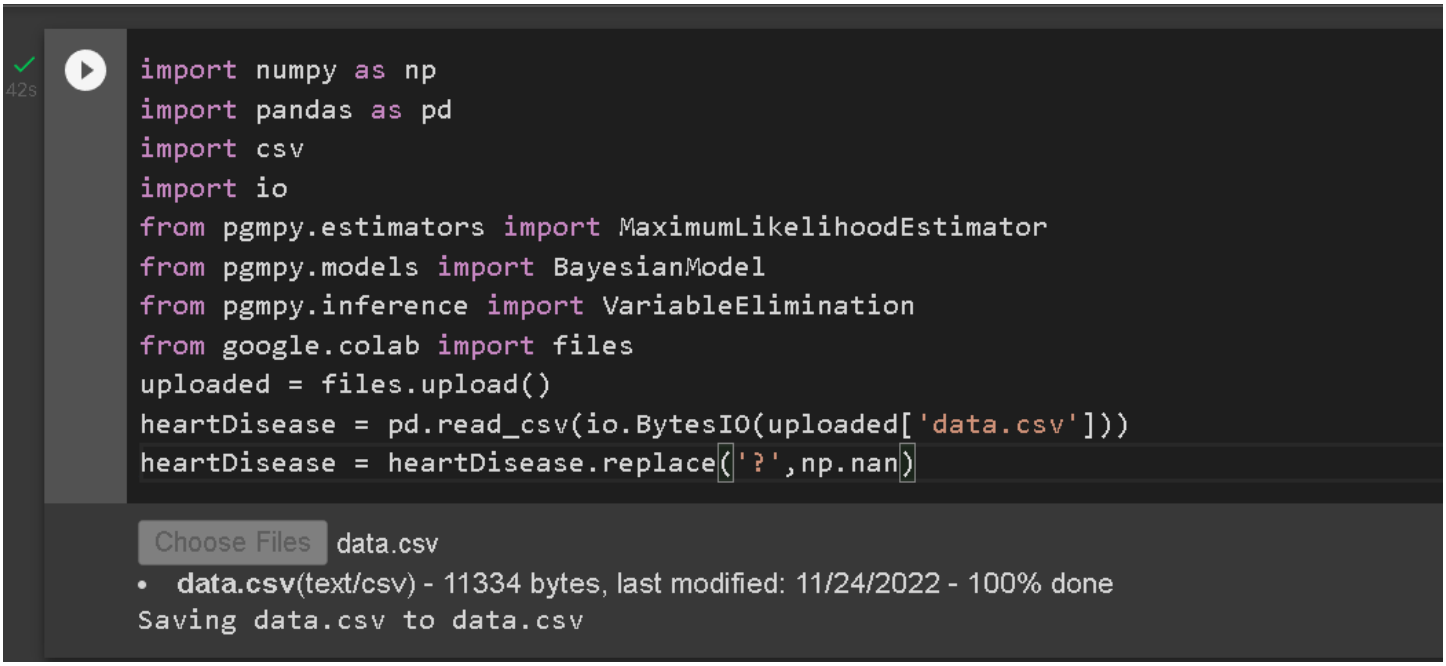
180 rows × 9 columns

Practical No. 8A

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.

Code:

```
import numpy as np
import pandas as pd
import csv
import io
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
from google.colab import files
uploaded = files.upload()
heartDisease = pd.read_csv(io.BytesIO(uploaded['data.csv']))
heartDisease = heartDisease.replace('?',np.nan)
```




```
print('Sample instances from the dataset are given below')
print(heartDisease.head())
print('\n Attributes and datatypes')
print(heartDisease.dtypes)
```

```
print('Sample instances from the dataset are given below')
print(heartDisease.head())
print('\n Attributes and datatypes')
print(heartDisease.dtypes)
```

```
Sample instances from the dataset are given below
   age  gender  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0   63       1   1       145   233    1        2      150      0      2.3
1   67       1   4       160   286    0        2      108      1      1.5
2   67       1   4       120   229    0        2      129      1      2.6
3   37       1   3       130   250    0        0      187      0      3.5
4   41       0   2       130   204    0        2      172      0      1.4

   slope ca thal  heartdisease
0      3  0    6             0
1      2  3    3             2
2      2  2    7             1
3      3  0    3             0
4      1  0    3             0
```

```
 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={'restecg':1})
print(q1)
```

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

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

```
 2. Probability of HeartDisease given evidence= cp
+----------------+--------------------+
| heartdisease   |   phi(heartdisease) |
+================+====================+
| heartdisease(0) |             0.3610 |
+----------------+--------------------+
| heartdisease(1) |             0.2159 |
+----------------+--------------------+
| heartdisease(2) |             0.1373 |
+----------------+--------------------+
| heartdisease(3) |             0.1537 |
+----------------+--------------------+
| heartdisease(4) |             0.1321 |
+----------------+--------------------+
```

Practical No. 8B

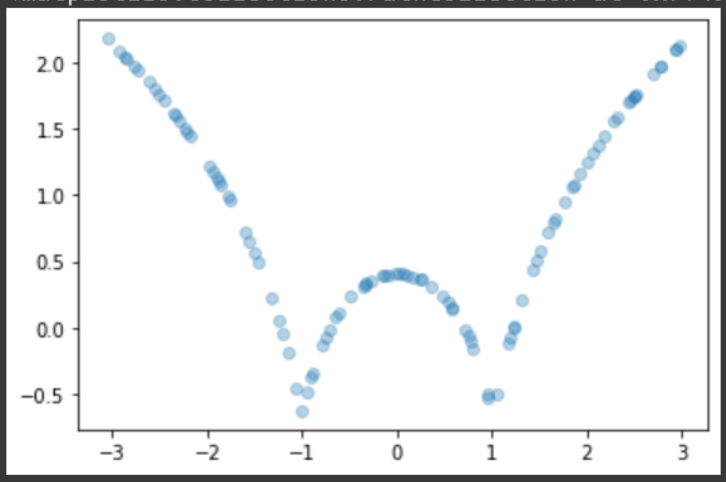
Aim: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
X=np.linspace(-3,3,100)
print(X)
X+=np.random.normal(scale=0.05,size=100)
Y=np.log(np.abs((X**2)-1)+0.5)
print(Y)
np.linspace(2.0, 3.0, num=5)
plt.scatter(X,Y,alpha=0.32)
```

```
[-3.         -2.93939394 -2.87878788 -2.81818182 -2.75757576 -2.6969697
 -2.63636364 -2.57575758 -2.51515152 -2.45454545 -2.39393939 -2.33333333
 -2.27272727 -2.21212121 -2.15151515 -2.09090909 -2.03030303 -1.96969697
 -1.90909091 -1.84848485 -1.78787879 -1.72727273 -1.66666667 -1.60606061
 -1.54545455 -1.48484848 -1.42424242 -1.36363636 -1.3030303  -1.24242424
 -1.18181818 -1.12121212 -1.06060606 -1.         -0.93939394 -0.87878788
 -0.81818182 -0.75757576 -0.6969697  -0.63636364 -0.57575758 -0.51515152
 -0.45454545 -0.39393939 -0.33333333 -0.27272727 -0.21212121 -0.15151515
 -0.09090909 -0.03030303  0.03030303  0.09090909  0.15151515  0.21212121
  0.27272727  0.33333333  0.39393939  0.45454545  0.51515152  0.57575758
  0.63636364  0.6969697   0.75757576  0.81818182  0.87878788  0.93939394
  1.          1.06060606  1.12121212  1.18181818  1.24242424  1.3030303
  1.36363636  1.42424242  1.48484848  1.54545455  1.60606061  1.66666667
  1.72727273  1.78787879  1.84848485  1.90909091  1.96969697  2.03030303
  2.09090909  2.15151515  2.21212121  2.27272727  2.33333333  2.39393939
```

```
 -0.07425239 -0.13433314 -0.01747788  0.07755679  0.1103033   0.23437295
  0.32559367  0.33035177  0.31484445  0.35076333  0.38760106  0.39825472
  0.39454591  0.40541399  0.40513774  0.40239446  0.39930045  0.38601555
  0.36791687  0.35866641  0.31473177  0.19896806  0.24433946  0.15037615
  0.1437064  -0.06506665 -0.02426953 -0.16017644 -0.10681846 -0.52558651
 -0.49955717 -0.50662344 -0.0763946  -0.12078643  0.00272492  0.20701164
  0.00378319  0.43996614  0.51178839  0.58000834  0.720806    0.8188045
  0.79641171  0.95167088  1.06412837  1.0815949   1.24768893  1.16307432
  1.37891132  1.32292051  1.44922554  1.55111629  1.59214349  1.71008831
  1.74880908  1.70416097  1.74825827  1.76242086  1.91028825  1.97436514
  1.97236775  2.09895023  2.09456023  2.12367124]
<matplotlib.collections.PathCollection at 0x7f49e1beb550>
```



Practical No. 9A

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

Code:

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0)
y = y/100

def sigmoid (x):
    return 1/(1 + np.exp(-x))

def derivatives_sigmoid(x):
    return x * (1 - x)

epoch=5000
lr=0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1

wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
```

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0)
y = y/100

def sigmoid (x):
    return 1/(1 + np.exp(-x))

def derivatives_sigmoid(x):
    return x * (1 - x)

epoch=5000
lr=0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1

wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
```

for i in range(epoch):

```
  #Forward Propogation
  hinp1=np.dot(X,wh)
  hinp=hinp1 + bh
  hlayer_act = sigmoid(hinp)
  outinp1=np.dot(hlayer_act,wout)
  outinp= outinp1+ bout
  output = sigmoid(outinp)

 #Backpropagation
  EO = y-output
  outgrad = derivatives_sigmoid(output)
  d_output = EO* outgrad
  EH = d_output.dot(wout.T)

  hiddengrad = derivatives_sigmoid(hlayer_act)
  d_hiddenlayer = EH * hiddengrad

wout += hlayer_act.T.dot(d_output) *lr
wh += X.T.dot(d_hiddenlayer) *lr
```

```
for i in range(epoch):

    #Forward Propogation
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+ bout
    output = sigmoid(outinp)

  #Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO* outgrad
    EH = d_output.dot(wout.T)

    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad

wout += hlayer_act.T.dot(d_output) *lr
wh += X.T.dot(d_hiddenlayer) *lr
```

print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)

```
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.90009484]
 [0.88689694]
 [0.90049882]]
```

Practical No. 9B

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

Code:

```
import pandas as pd
msg=pd.read_csv('/content/sample_data/naivetext.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
print(X)
print(y)
```

```
The dimensions of the dataset (18, 2)
0                         I love this sandwich
1                  This is an amazing place
2         I feel very good about these beers
3                         This is my best work
4                         What an awesome view
5               I do not like this restaurant
6                    I am tired of this stuff
7                      I can't deal with this
8                        He is my sworn enemy
9                         My boss is horrible
10                 This is an awesome place
11    I do not like the taste of this juice
12                               I love to dance
13        I am sick and tired of this place
14                        What a great holiday
15           That is a bad locality to stay
16           We will have good fun tomorrow
17         I went to my enemy's house today
Name: message, dtype: object
0     1
1     1
2     1
3     1
4     1
5     0
6     0
7     0
8     0
9     0
10    1
11    0
12    1
13    0
14    1
15    0
```

✓ 0s completed at 12:24 PM

#splitting the dataset into train and test data

```
from sklearn.model_selection import train_test_split
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,)
```

```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain)
xtest_dtm=count_vect.transform(xtest)
print('\n The words or Tokens in the text documents \n')
print(count_vect.get_feature_names())
df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names())
```

```
 The words or Tokens in the text documents

['about', 'am', 'amazing', 'an', 'and', 'awesome', 'bad', 'beers', 'boss', 'can', 'deal', 'do', 'enemy', 'feel', 'fun', 'good', 'have', 'he', 'horrib
/usr/local/lib/python3.8/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names is
  warnings.warn(msg, category=FutureWarning)
```

```
# Training Naive Bayes (NB) classifier on training data.
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)

#printing accuracy, Confusion matrix, Precision and Recall
from sklearn import metrics
print(" Accuracy of the classifer is", metrics.accuracy_score(ytest,predicted))

print('Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
print(" The value of Precision" ,metrics.precision_score(ytest,predicted))
print(" The value of Recall" ,metrics.recall_score(ytest,predicted))
```

OUTPUT

```
 Accuracy of the classifer is 0.4
Confusion matrix
[[1 0]
 [3 1]]
 The value of Precision 1.0
 The value of Recall 0.25
```

Practical No. 10

Aim: Perform Text pre-processing, Text clustering, classification with Prediction, Test Score and Confusion Matrix.

Code:-

```
import pandas as pd
data = pd.read_csv('spam.csv',encoding='latin-1')
data.head()
# drop unnecessary columns and rename cols
data.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], axis=1, inplace=True)
data.columns = ['label', 'text']
data.head()
# check missing values
data.isna().sum()
# check data shape
data.shape
# check target balance
data['label'].value_counts(normalize = True).plot.bar()
# text preprocessing
# download nltk
import nltk
nltk.download('all')
# create a list text

text = list(data['text'])
# preprocessing loop
import re
```

```
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
corpus = []
for i in range(len(text)):
r = re.sub('[^a-zA-Z]', ' ', text[i])
 r = r.lower()
 r = r.split()
 r = [word for word in r if word not in stopwords.words('english')]
 r = [lemmatizer.lemmatize(word) for word in r]
 r = ' '.join(r)
corpus.append(r)
#assign corpus to data['text']
data['text'] = corpus
data.head()
# Create Feature and Label sets
X = data['text']
y = data['label']
# train test split (66% train - 33% test)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=123)


print('Training Data :', X_train.shape)
print('Testing Data : ', X_test.shape)
# Train Bag of Words model
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
X_train_cv = cv.fit_transform(X_train)
X_train_cv.shape
#Training Logistic Regression model
```

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()

lr.fit(X_train_cv, y_train)

# transform X_test using CV

X_test_cv = cv.transform(X_test)

# generate predictions

predictions = lr.predict(X_test_cv)

predictions

# confusion matrix

import pandas as pd

from sklearn import metrics

df = pd.DataFrame(metrics.confusion_matrix(y_test,predictions), index=['ham','spam'], columns=['ham','spam'])

df
```

OUTPUT:-

In [1]:
```
import pandas as pd
```

In [5]:
```
data = pd.read_csv('spam.csv',encoding='latin-1')
data.head()
```

Out[5]:

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

In [6]:
```
# drop unnecessary columns and rename cols

data.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], axis=1, inplace=True)

data.columns = ['label', 'text']

data.head()
```

Out[6]:

	label	text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...

```
data.head()
```

Out[6]:

	label	text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

In [7]:
```
# check missing values

data.isna().sum()
```

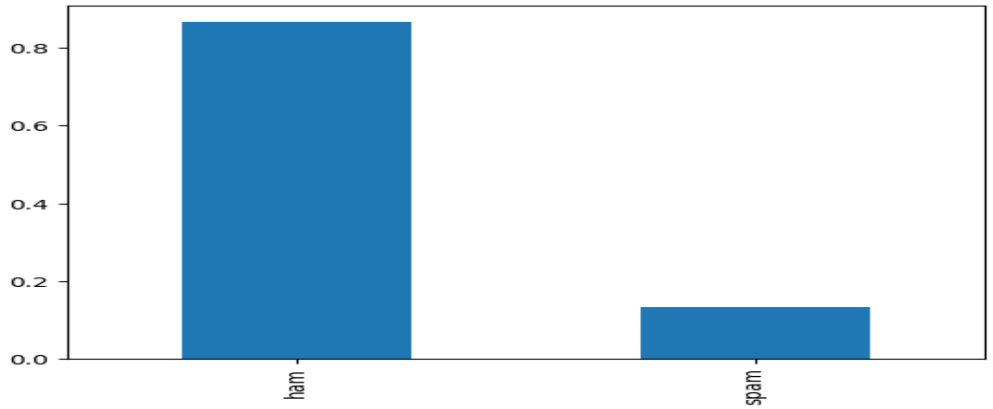
Out[7]:
```
label    0
text     0
dtype: int64
```

In [8]:
```
# check data shape

data.shape
```

Out[8]:
```
(5572, 2)
```

In [9]:
```
# check target balance
data['label'].value_counts(normalize = True).plot.bar()
```

Out[9]:
```
<AxesSubplot: >
```



```
cv = CountVectorizer()
X_train_cv = cv.fit_transform(X_train)
X_train_cv.shape
```

Out[12]: (3733, 5698)

In [14]:
```
#Training Logistic Regression model
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train_cv, y_train)
# transform X_test using CV
X_test_cv = cv.transform(X_test)
# generate predictions
predictions = lr.predict(X_test_cv)
predictions
```

Out[14]: array(['ham', 'spam', 'ham', ..., 'ham', 'ham', 'spam'], dtype=object)

In [15]:
```
# confusion matrix
import pandas as pd
from sklearn import metrics
df = pd.DataFrame(metrics.confusion_matrix(y_test,predictions), index=['ham','sp
df
```

Out[15]:

	ham	spam
ham	1600	2
spam	31	206