1. 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.

Program:

```
import pandas as pd

x=pd.read_csv(r"C:\Users\SRAVANI\Downloads\Income.csv")

print(x.head())

s=[]

for i in range(len(x)):

    y=list(x.iloc[i])

    if(y[-1]==1):

        if len(s)==0:

            s=y[:len(y)-1]

        else:

            for j in range(len(s)):

                if y[j]!=s[j]:

                    s[j]="?"

print(s)
```

OUTPUT:

```
   Age  Income  Assets  Height  Qualification  Status
0   Y    High      1       2         3             1
1   Y    Med       0       3         2             1
2   Y    High      1       1         2             1
3   Y    Med       1       1         2             0
4   M    Low       0       1         1             0

['?', '?', '?', '?', '?']
```

2. 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

Program:

```
import numpy as np

import pandas as pd

data = pd.DataFrame(data=pd.read_csv(r'C:\Users\Downloads\ws.csv'))
```

```
concepts = np.array(data.iloc[:,0:-1])
print(concepts)
target = np.array(data.iloc[:,-1])
print(target)
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and general_h")
    print(specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print(general_h)
    for i, h in enumerate(concepts):
        if target[i] == "Yes":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'
        if target[i] == "No":
            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("\nSteps of Candidate Elimination Algorithm",i+1)
        print(specific_h)
        print(general_h)
    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("\nFinal Specific_h:", s_final, sep="\n")
print("\nFinal General_h:", g_final, sep="\n")
```

OUTPUT:

[['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
['Yes' 'No' 'Yes']

Initialization of specific_h and general_h
['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

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

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

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

Final Specific_h:
['Sunny' 'Warm' 'High' 'Strong' '?' '?']

Final General_h:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

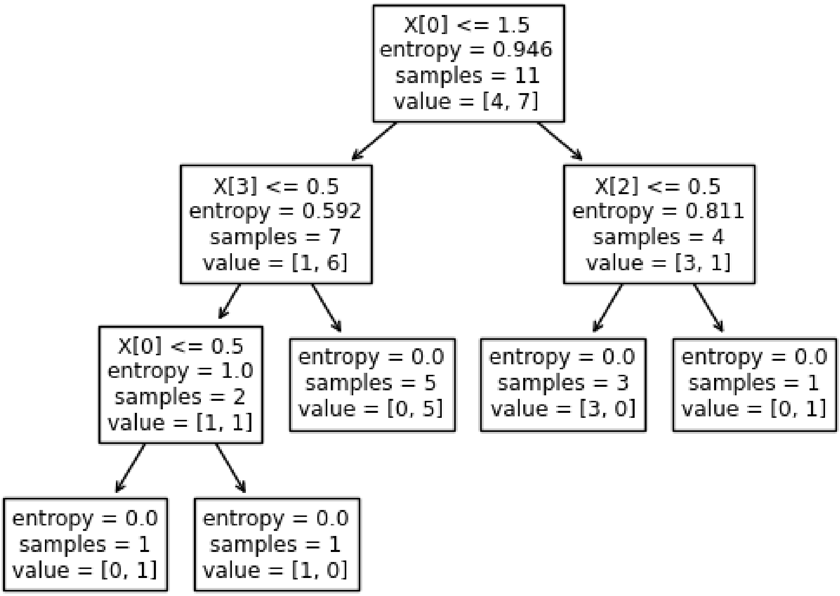
3. Aim: Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Program:

```
import pandas as pd

from sklearn import tree

from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier

df=pd.read_csv(r"C:\Users\ADI SESHU\Downloads\play_tennis.csv")

df = df.drop('day',axis=1)
```

```
from sklearn.preprocessing import LabelEncoder
Le = LabelEncoder()
df['outlook'] = Le.fit_transform(df['outlook'])
df['temp'] = Le.fit_transform(df['temp'])
df['humidity'] = Le.fit_transform(df['humidity'])
df['wind'] = Le.fit_transform(df['wind'])
df['play'] = Le.fit_transform(df['play'])
x = df.drop(['play'],axis=1)
y= df['play']
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(x,y,random_state=20,test_size=0.2)
# Fitting the model
from sklearn import tree
dt = tree.DecisionTreeClassifier(criterion = 'entropy')
dt = dt.fit(X_train, y_train)
dt
dt.score(X_train,y_train)
y_pred = dt.predict(X_test)
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))
new_df = pd.DataFrame({'outlook': 2,
                       'temp': 1,
                       'humidity':0,
                       'wind': 1},index=[0])
y_pred1 = dt.predict(new_df)
y_pred1
tree.plot_tree(dt)
```

Output:



4. Aim: Exercises to solve the real-world problems using the following machine learning methods: a) Linear Regression b) Logistic Regression c) Binary Classifier

Program:

```
#3.linear regression

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the datasets

datasets = pd.read_csv(r"C:\Users\ADI SESHU\Downloads\Salary_Data (1).csv")

X = datasets.iloc[:, :-1].values

Y = datasets.iloc[:, 1].values

# Splitting the dataset into the Training set and Test set

from sklearn.model_selection import train_test_split

X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size = 1/3, random_state = 0)
```

```
# Fitting Simple Linear Regression to the training set

from sklearn.linear_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X_Train, Y_Train)

# Predicting the Test set result

Y_Pred = regressor.predict(X_Test)

# Visualising the Training set results

plt.scatter(X_Train, Y_Train, color = 'red')

plt.plot(X_Train, regressor.predict(X_Train), color = 'blue')

plt.title('Salary vs Experience  (Training Set)')

plt.xlabel('Years of experience')

plt.ylabel('Salary')

plt.show()

# Visualising the Test set results

plt.scatter(X_Test, Y_Test, color = 'red')

plt.plot(X_Train, regressor.predict(X_Train), color = 'blue')

plt.title('Salary vs Experience  (Training Set)')

plt.xlabel('Years of experience')

plt.ylabel('Salary')

plt.show()
```

Output:


```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_Train = sc_X.fit_transform(X_Train)
X_Test = sc_X.transform(X_Test)
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_Train, Y_Train)
# Predicting the test set results
Y_Pred = classifier.predict(X_Test)
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_Test, Y_Pred)
plt.scatter(X_Train, Y_Train, color = 'red')
```

```
plt.plot(X_Train, classifier.predict(X_Train), color = 'blue')
plt.title('Salary vs Experience  (Training Set)')
plt.xlabel('Years of experience')
plt.ylabel('Salary')
plt.show()
# Visualising the Test set results
plt.scatter(X_Test, Y_Test, color = 'red')
plt.plot(X_Train, classifier.predict(X_Train), color = 'blue')
plt.title('Salary vs Experience  (Training Set)')
plt.xlabel('Years of experience')
plt.ylabel('Salary')
plt.show()
```

Output:

5. Aim: Develop a program for Bias, Variance, Remove duplicates, Cross Validation

Program:

```
from mlxtend.evaluate import bias_variance_decomp

from sklearn.model_selection import train_test_split

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

from sklearn.linear_model import LinearRegression, Lasso

import warnings

warnings.filterwarnings('ignore')

#We will load the Boston house dataset for our example

from sklearn.datasets import load_boston

from sklearn import metrics

# From the library load the necessary rows & columns

X, y = load_boston(return_X_y=True)

# Split the dataset into train and test

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)

# Model definition

model_lr = LinearRegression()

# Estimation of bias and variance using bias_variance_decomp

#Note here we are using loss as 'mse' and setting default bootstrap num_rounds to 200

mse, bias, var = bias_variance_decomp(model_lr, X_train, y_train, X_test, y_test, loss='mse', num_rounds=200,
random_seed=123)

y_pred=model_lr.predict(X_test)

# summarize results

print('MSE from bias_variance lib [avg expected loss]: %.3f' % mse)

print('Avg Bias: %.3f' % bias)

print('Avg Variance: %.3f' % var)

print('Mean Square error by Sckit-learn lib: %.3f' % metrics.mean_squared_error(y_test,y_pred))
```

output:

```
MSE from bias_variance lib [avg expected loss]: 22.128
Avg Bias: 20.522
Avg Variance: 1.606
Mean Square error by Sckit-learn lib: 22.069
```

6. Aim: write a program to implement categorical encoding, One-hot Encoding

Program:

```
print("original data")

print(df1)

from sklearn.preprocessing import LabelEncoder

encoder=LabelEncoder()

df1['team']=encoder.fit_transform(df1['team'])

print("after encoding")

print(df1)
```

output:

```
original data
  team  points
0    A      25
1    A      12
2    B      15
3    B      14
4    B      19
5    B      23
6    C      25
7    C      29
after encoding
   team  points
0     0      25
1     0      12
2     1      15
3     1      14
4     1      19
5     1      23
6     2      25
7     2      29
```

Program:

```
import pandas as pd


#create DataFrame

df = pd.DataFrame({'team': ['A', 'A', 'B', 'B', 'B', 'B', 'C', 'C'],

                   'points': [25, 12, 15, 14, 19, 23, 25, 29]})

df1=df

#view DataFrame
```

```
print("original data")

print(df)

from sklearn.preprocessing import OneHotEncoder

#creating instance of one-hot-encoder

encoder = OneHotEncoder(handle_unknown='ignore')

#perform one-hot encoding on 'team' column

encoder_df = pd.DataFrame(encoder.fit_transform(df[['team']]).toarray())

#merge one-hot encoded columns back with original DataFrame

final_df = df.join(encoder_df)

final_df.drop('team', axis=1, inplace=True)

#rename columns

final_df.columns = ['points', 'teamA', 'teamB', 'teamC']

#view final df

print("after one hot encoding")

print(final_df)
```

output:

```
original data
  team  points
0    A      25
1    A      12
2    B      15
3    B      14
4    B      19
5    B      23
6    C      25
7    C      29
after one hot encoding
   points  teamA  teamB  teamC
0      25    1.0    0.0    0.0
1      12    1.0    0.0    0.0
2      15    0.0    1.0    0.0
3      14    0.0    1.0    0.0
4      19    0.0    1.0    0.0
5      23    0.0    1.0    0.0
6      25    0.0    0.0    1.0
7      29    0.0    0.0    1.0
```

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

Program:

```
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) #maximum of X array longitudinally
y = y/100
#Sigmoid Function
def sigmoid (x):
    return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)

#Variable initialization
epoch=5 #Setting training iterations
lr=0.1 #Setting learning rate

inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
#weight and bias initialization

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))

#draws a random range of numbers uniformly of dim x*y
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)#how much hidden layer wts contributed to error
    d_hiddenlayer = EH * hiddengrad

    wout += hlayer_act.T.dot(d_output) *lr   # dotproduct of nextlayererror and currentlayerop
    wh += X.T.dot(d_hiddenlayer) *lr

    print ("-----------Epoch-", i+1, "Starts----------")
    print("Input: \n" + str(X))
    print("Actual Output: \n" + str(y))
    print("Predicted Output: \n" ,output)
    print ("-----------Epoch-", i+1, "Ends----------\n")

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

Training Examples:

Training Examples:

Example	Sleep	Study	Expected % in Exams
1	2	9	92
2	1	5	86

3	3	6	89

Normalize the input:

Example	Sleep	Study	Expected % in Exams
1	2/3 = 0.66666667	9/9 = 1	0.92
2	1/3 = 0.33333333	5/9 = 0.55555556	0.86
3	3/3 = 1	6/9 = 0.66666667	0.89

Output:

————Epoch- 1 Starts————-

Input:

[[0.66666667 1.]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.81951208]

[0.8007242]

[0.82485744]]

————Epoch- 1 Ends————-

————Epoch- 2 Starts————-

Input:

[[0.66666667 1.]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.82033938]

[0.80153634]

[0.82568134]]

————Epoch- 2 Ends————-

————Epoch- 3 Starts————-

Input:

[[0.66666667 1.]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.82115226]

[0.80233463]

[0.82649072]]

————Epoch- 3 Ends————-

————Epoch- 4 Starts————-

Input:

[[0.66666667 1.]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.82195108]

[0.80311943]

[0.82728598]]

————Epoch- 4 Ends———-

————Epoch- 5 Starts———-

Input:

[[0.66666667 1.]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.8227362]

[0.80389106]

[0.82806747]]

————Epoch- 5 Ends———-

Input:

[[0.66666667 1.]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.8227362]

[0.80389106]

[0.82806747]]

8. Aim: Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.

Program:

```
import pandas as pd

from sklearn.metrics import confusion_matrix , accuracy_score , classification_report

from sklearn.preprocessing import LabelEncoder

ds = pd.read_csv(r"D:\ML LAB 3-2\IRIS.csv")

encoder=LabelEncoder()

ds['species']=encoder.fit_transform(ds['species'])

x=ds.iloc[:,:-1].values

y=ds.iloc[:,-1].values

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=1)

from sklearn.neighbors import KNeighborsClassifier

clf=KNeighborsClassifier(n_neighbors=8)

clf.fit(x_train,y_train)

y_pred=clf.predict(x_test)

print("Accuracy: ", accuracy_score (y_test, y_pred))

print("Confusion Matrix :\n", confusion_matrix (y_test, y_pred))

print("Classification Report :\n", classification_report (y_test, y_pred))
```

Output:

```
Accuracy:  0.9777777777777777
Confusion Matrix :
 [[14  0  0]
 [ 0 17  1]
 [ 0  0 13]]
Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       1.00      0.94      0.97        18
           2       0.93      1.00      0.96        13

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

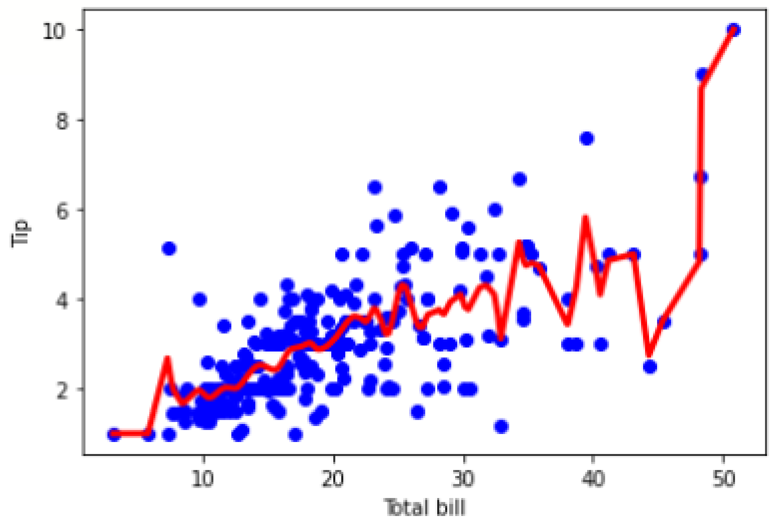
9. Aim: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Program:-

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
df = pd.read_csv('tips.csv')
features = np.array(df.total_bill)
labels = np.array(df.tip)

def kernel(data, point, xmat, k):
  m,n = np.shape(xmat)
  ws = np.mat(np.eye((m)))
  for j in range(m):
    diff = point - data[j]
    ws[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
  return ws

def local_weight(data, point, xmat, ymat, k):
  wei = kernel(data, point, xmat, k)
  return (data.T*(wei*data)).I*(data.T*(wei*ymat.T))

def local_weight_regression(xmat, ymat, k):
  m,n = np.shape(xmat)
  ypred = np.zeros(m)
  for i in range(m):
    ypred[i] = xmat[i]*local_weight(xmat, xmat[i],xmat,ymat,k)
  return ypred

m = features.shape[0]
mtip = np.mat(labels)
```

```
data = np.hstack((np.ones((m, 1)), np.mat(features).T))

ypred = local_weight_regression(data, mtip, 0.5)
indices = data[:,1].argsort(0)
xsort = data[indices][:,0]

fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(features, labels, color='blue')
ax.plot(xsort[:,1],ypred[indices], color = 'red', linewidth=3)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show()
```

Output:-



10. Aim: To Classify a set of documents using naive_bayes classifier

Program:

```
import numpy as np
import pandas as pd
data=pd.read_csv(r'D:\text_dataset.csv')
display(data.head())
print('The dimensions of the dataset',data.shape)
```

O/P :

	S.NO	Text Documents	Label
0	1	I love this sandwich	pos
1	2	This is an amazing place	pos
2	3	I feel very good about these beers	pos
3	4	This is my best work	pos
4	5	What an awesome view	pos

```
The dimensions of the dataset (18, 3)
```

data['Label']=data.Label.map({'pos':1,'neg':0})

display(data.head())

Output:

	S.NO	Text Documents	Label
0	1	I love this sandwich	1
1	2	This is an amazing place	1
2	3	I feel very good about these beers	1
3	4	This is my best work	1
4	5	What an awesome view	1

x=data['Text Documents']

y=data['Label']

from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)

print ('\n the total number of Training Data :',x_train.shape)

print ('\n the total number of Test Data :',x_test.shape)

Output:

the total number of Training Data : (14,)

the total number of Test Data : (4,)

from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer()

xtrain_dtm = cv.fit_transform(x_train)

xtest_dtm=cv.transform(x_test)

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())

Output:

```
The words or Tokens in the text documents

['about', 'am', 'an', 'and', 'awesome', 'bad', 'beers', 'best', 'boss', 'can', 'dance', 'deal', 'do', 'enemy', 'feel', 'fun',
'good', 'great', 'have', 'holiday', 'horrible', 'house', 'is', 'juice', 'like', 'locality', 'love', 'my', 'not', 'of', 'place',
'sick', 'stay', 'stuff', 'taste', 'that', 'the', 'these', 'this', 'tired', 'to', 'today', 'tomorrow', 'very', 'view', 'we', 'we
nt', 'what', 'will', 'with', 'work']
```

from sklearn.naive_bayes import MultinomialNB

clf = MultinomialNB().fit(xtrain_dtm,y_train)

predicted = clf.predict(xtest_dtm)

from sklearn import metrics

print('\n Accuracy of the classifier is',metrics.accuracy_score(y_test,predicted))

print('\n Confusion matrix')

print(metrics.confusion_matrix(y_test,predicted))

print('\n The value of Precision', metrics.precision_score(y_test,predicted))

print('\n The value of Recall', metrics.recall_score(y_test,predicted))

Output:

```
Accuracy of the classifier is 1.0

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

 The value of Precision 1.0

 The value of Recall 1.0
```

11. Aim: Apply EM algorithm to cluster a Heart Disease Data Set. Use the same data set for cluster ing using k-Means algorithm. Compare the results of these two algorithms and comment on the qualit y of clustering. You can add Java/Python ML library classes/API in the program.

Program:

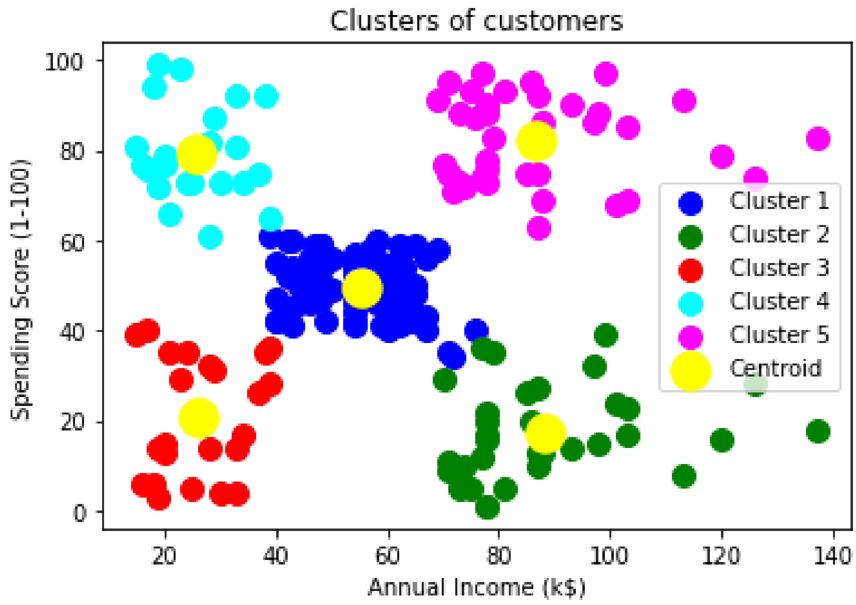
import numpy as nm

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

```
import pandas as pd

dataset = pd.read_csv(r"C:\Users\mmuni\Downloads\archive (1)\Mall_Customers.csv")

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

#training the K-means model on a dataset

kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)

y_predict= kmeans.fit_predict(x)

plt.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1') #for first cluster

plt.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2') #for second cluster

plt.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3') #for third cluster

plt.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4') #for fourth cluster

plt.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5') #for fifth cluster

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroid')

plt.title('Clusters of customers')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.legend()

plt.show()
```

output:



12. AIM:

Exploratory Data Analysis for Classification using Pandas or Matplotlib.

Program:

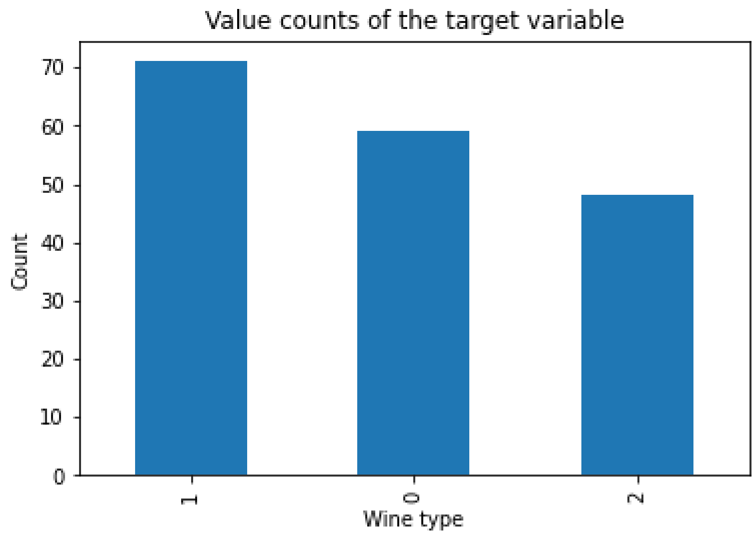
```
# data manipulation
import pandas as pd
import numpy as np
# data viz
import matplotlib.pyplot as plt
import seaborn as sns
# use sklearn to import a dataset
from sklearn.datasets import load_wine
wine = load_wine()
df = pd.DataFrame(data=wine.data, columns=wine.feature_names)
df["target"] = wine.target
#display(df.head(2))
#display(df.tail(2))
print("shape",df.shape)
print("***********")
print("information about dataset")
df.info()
```

```
print("***********")

print("Number of duplicate rows",df.duplicated().sum())

print("***********")

print("count of target variables \n",df.target.value_counts())

df.target.value_counts().plot(kind="bar")

plt.title("Value counts of the target variable")

plt.xlabel("Wine type")

plt.ylabel("Count")

plt.show()  # visualization

print("Skewness:", df['magnesium'].skew())

print("Kurtosis:",df['magnesium'].kurt())#statistical tools
```

OUTPUT:

```
shape (178, 14)
***********
information about dataset
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   alcohol                       178 non-null    float64
 1   malic_acid                    178 non-null    float64
 2   ash                           178 non-null    float64
 3   alcalinity_of_ash             178 non-null    float64
 4   magnesium                     178 non-null    float64
 5   total_phenols                 178 non-null    float64
 6   flavanoids                    178 non-null    float64
 7   nonflavanoid_phenols          178 non-null    float64
 8   proanthocyanins               178 non-null    float64
 9   color_intensity               178 non-null    float64
 10  hue                           178 non-null    float64
 11  od280/od315_of_diluted_wines  178 non-null    float64
 12  proline                       178 non-null    float64
 13  target                        178 non-null    int32
dtypes: float64(13), int32(1)
memory usage: 18.9 KB
***********
Number of duplicate rows 0
***********
count of target variables
1    71
0    59
2    48
```

Name: target, dtype: int64



13.Aim: Write a Python 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

Program:

```
import numpy as np

import pandas as pd

import csv

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.models import BayesianModel

from pgmpy.inference import VariableElimination


heartDisease = pd.read_csv('heart.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)
```

```
model=
BayesianModel([('age','heartdisease'),('sex','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)
```

Output:

```
Learning CPD using Maximum likelihood estimators

 Inferencing with Bayesian Network:

 1. Probability of HeartDisease given evidence= restecg

+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1012 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2392 |
+-----------------+---------------------+
| heartdisease(3) |              0.2015 |
+-----------------+---------------------+
| heartdisease(4) |              0.4581 |
+-----------------+---------------------+

 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 |
+-----------------+---------------------+
```

14. Aim: WRITE A PROGRAM TO IMPLEMENT SUPPORT VECTOR MACHINE

PROGRAM:

```
from sklearn import tree

from sklearn.model_selection import train_test_split

import pandas as pd

df=pd.read_csv(r"C:\Users\Downloads\heart (1).csv")

df.head()
```

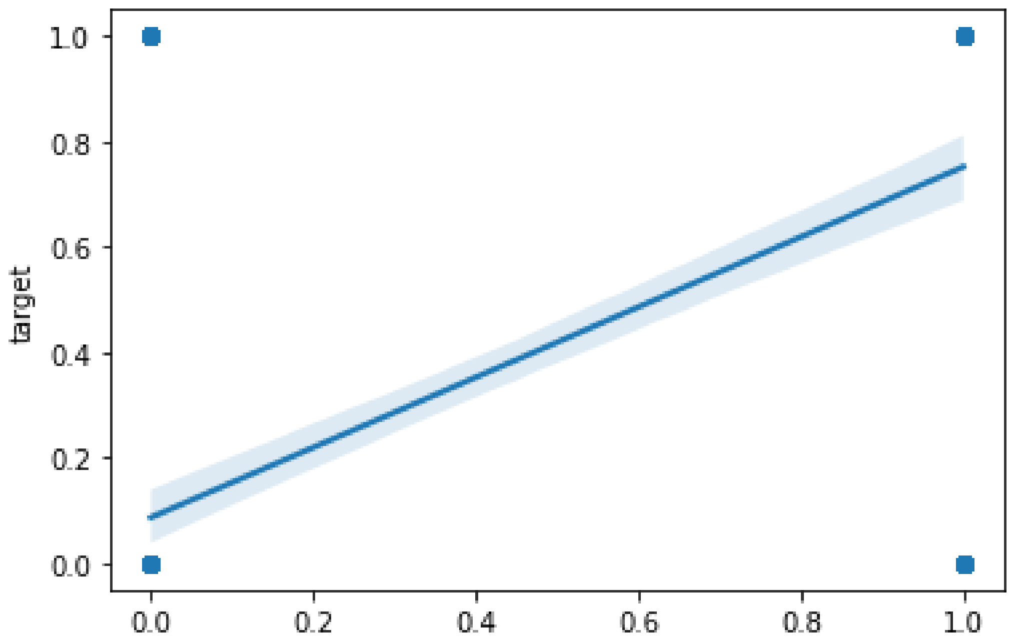
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

```
y=df['target']
x=df.drop("target",axis=1)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,random_state=1)
```

#USING KERNAL="LINEAR"

```
from sklearn.svm import SVC
clf=SVC(kernel="linear")
model=clf.fit(x_train,y_train)
y_pred= model.predict(x_test)
import seaborn as sns
import matplotlib.pyplot as plt
sns.regplot(y_pred,y_test)
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
result=confusion_matrix(y_test,y_pred)
print("Confusion Matrix:")
print(result)
print("Accuracy=",accuracy_score(y_test,y_pred))
print("Classification Report")
print(classification_report(y_test,y_pred))
```

OUTPUT:




```
Confusion Matrix:
[[116  45]
 [ 11 136]]
Accuracy= 0.8181818181818182
Classification Report
              precision    recall  f1-score   support

           0       0.91      0.72      0.81       161
           1       0.75      0.93      0.83       147

    accuracy                           0.82       308
   macro avg       0.83      0.82      0.82       308
weighted avg       0.84      0.82      0.82       308
```

#USING KERNAL="SIGMOID"

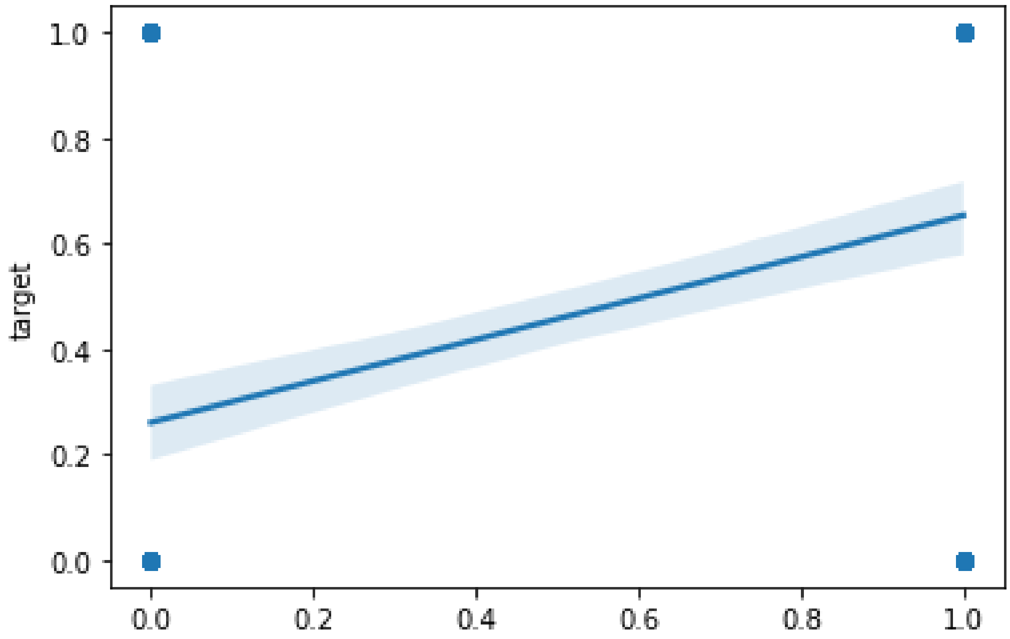
```
from sklearn.svm import SVC

clf=SVC(kernel="sigmoid")

model=clf.fit(x_train,y_train)

y_pred= model.predict(x_test)

import seaborn as sns

import matplotlib.pyplot as plt

sns.regplot(y_pred,y_test)

from sklearn.metrics import classification_report,confusion_matrix,accuracy_score

result=confusion_matrix(y_test,y_pred)

print("Confusion Matrix:")

print(result)
```

print("Accuracy=",accuracy_score(y_test,y_pred))

print("Classification Report")

print(classification_report(y_test,y_pred))

OUTPUT:




```
Confusion Matrix:
[[102  59]
 [ 36 111]]
Accuracy= 0.6915584415584416
Classification Report
              precision    recall  f1-score   support

           0       0.74      0.63      0.68       161
           1       0.65      0.76      0.70       147

    accuracy                           0.69       308
   macro avg       0.70      0.69      0.69       308
weighted avg       0.70      0.69      0.69       308
```

#USING KERNAL="POLY"

from sklearn.svm import SVC

clf=SVC(kernel="poly")

model=clf.fit(x_train,y_train)

y_pred= model.predict(x_test)

import seaborn as sns

import matplotlib.pyplot as plt

sns.regplot(y_pred,y_test)

from sklearn.metrics import classification_report,confusion_matrix,accuracy_score

result=confusion_matrix(y_test,y_pred)

print("Confusion Matrix:")

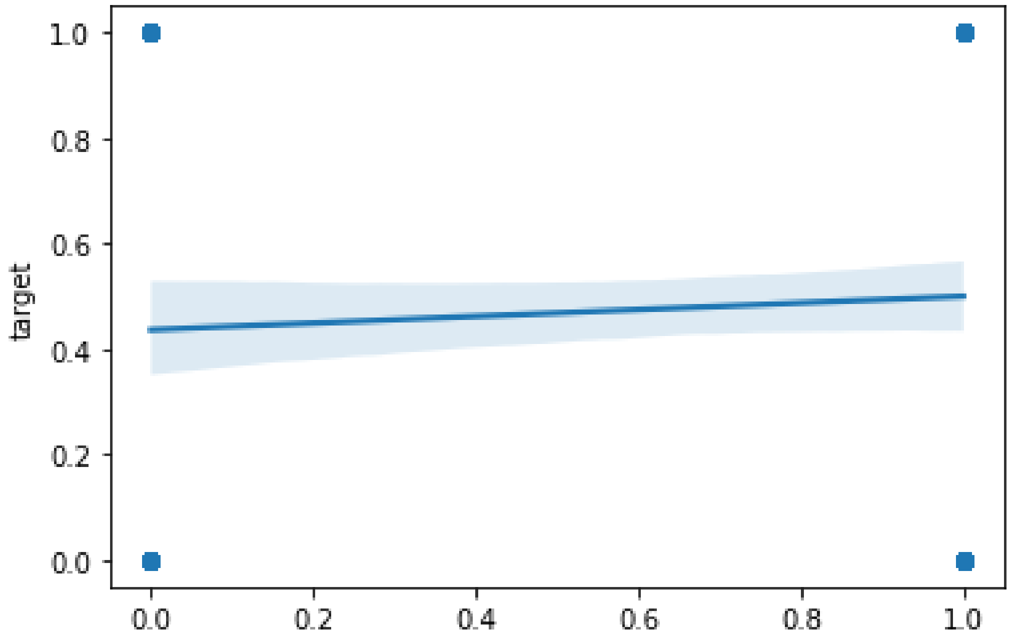
print(result)

print("Accuracy=",accuracy_score(y_test,y_pred))

print("Classification Report")

print(classification_report(y_test,y_pred))

OUTPUT:

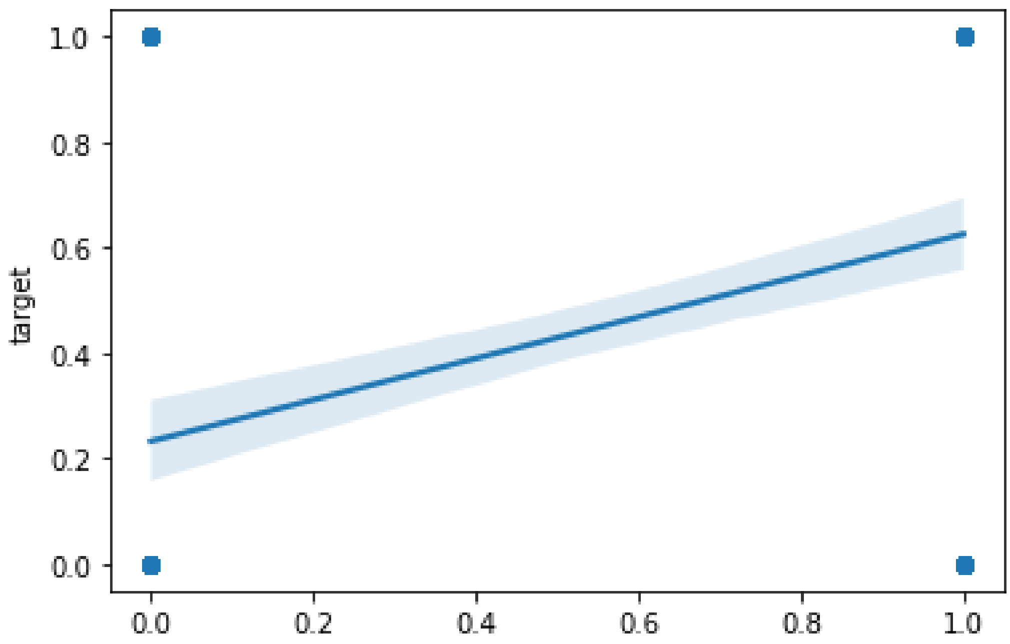



```
Confusion Matrix:
[[62 99]
 [48 99]]
Accuracy= 0.5227272727272727
Classification Report
              precision    recall  f1-score   support

           0       0.56      0.39      0.46       161
           1       0.50      0.67      0.57       147

    accuracy                           0.52       308
   macro avg       0.53      0.53      0.52       308
weighted avg       0.53      0.52      0.51       308
```

#USING KERNAL="RBF"

```
from sklearn.svm import SVC
clf=SVC(kernel="rbf")
model=clf.fit(x_train,y_train)
y_pred= model.predict(x_test)
import seaborn as sns
import matplotlib.pyplot as plt
sns.regplot(y_pred,y_test)
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
result=confusion_matrix(y_test,y_pred)
print("Confusion Matrix:")
print(result)
print("Accuracy=",accuracy_score(y_test,y_pred))
print("Classification Report")
print(classification_report(y_test,y_pred))
```

OUTPUT:




```
Confusion Matrix:
[[ 89  72]
 [ 27 120]]
Accuracy= 0.6785714285714286
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.77      0.55      0.64       161
           1       0.62      0.82      0.71       147

    accuracy                           0.68       308
   macro avg       0.70      0.68      0.68       308
weighted avg       0.70      0.68      0.67       308
```

15: WRITE A PROGRAM TO IMPLEMENT PRINCIPAL COMPONENT ANALYSIS

15. AIM: To implement principal component analysis

Program:

#BEFORE PCA

```
from sklearn import tree

from sklearn.model_selection import train_test_split

import pandas as pd

df=pd.read_csv(r"C:\Users\Downloads\heart (1).csv")

print(df.head())

y=df['target']

x=df.drop("target",axis=1)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,random_state=1)

model = tree.DecisionTreeClassifier()

model = model.fit(x_train, y_train)

y_pred = model.predict(x_test)W

from sklearn.metrics import classification_report,confusion_matrix,accuracy_score

result=confusion_matrix(y_test,y_pred)

print("Confusion Matrix:")

print(result)

print("Accuracy=",accuracy_score(y_test,y_pred))

print("Classification Report")

print(classification_report(y_test,y_pred))
```

#AFTER PCA

```
b=df['target']
a=df.drop("target",axis=1)
a_train, a_test, b_train, b_test = train_test_split(a, b, test_size=0.3,random_state=1)
from sklearn.decomposition import PCA
```

#USING 2 COMPONENTS

```
pca=PCA(n_components=2)
a_train=pca.fit_transform(a_train)
a_test=pca.transform(a_test)
model = tree.DecisionTreeClassifier()
model = model.fit(a_train, b_train)
y_pred = model.predict(a_test)
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
result=confusion_matrix(b_test,y_pred)
print("Confusion Matrix:")
print(result)
print("Accuracy=",accuracy_score(b_test,y_pred))
print("Classification Report")
print(classification_report(b_test,y_pred))
```

#USING 6 COMPONENTS

```
pca=PCA(n_components=6)
a_train=pca.fit_transform(a_train)
a_test=pca.transform(a_test)
model = tree.DecisionTreeClassifier()
model = model.fit(a_train, b_train)
y_pred = model.predict(a_test)
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
result=confusion_matrix(b_test,y_pred)
print("Confusion Matrix:")
print(result)
print("Accuracy=",accuracy_score(b_test,y_pred))
print("Classification Report")
```

print(classification_report(b_test,y_pred))

OUTPUT:

#BEFORE PCA

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

```
Confusion Matrix:
[[161   0]
 [  3 144]]
Accuracy= 0.9902597402597403
Classification Report
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       161
           1       1.00      0.98      0.99       147

    accuracy                           0.99       308
   macro avg       0.99      0.99      0.99       308
weighted avg       0.99      0.99      0.99       308
```

#AFTER PCA

#USING 2 COMPONENTS

```
Confusion Matrix:
[[161   0]
 [  6 141]]
Accuracy= 0.9805194805194806
Classification Report
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       161
           1       1.00      0.96      0.98       147

    accuracy                           0.98       308
   macro avg       0.98      0.98      0.98       308
weighted avg       0.98      0.98      0.98       308
```

#USING 6 COMPONENTS

```
Confusion Matrix:
[[157   4]
 [  3 144]]
Accuracy= 0.9772727272727273
Classification Report
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       161
           1       0.97      0.98      0.98       147

    accuracy                           0.98       308
   macro avg       0.98      0.98      0.98       308
weighted avg       0.98      0.98      0.98       308
```