1)Implement and demonstrate the FIND-S algorithm for finding the most specific

hypothesis based on a given set of training data samples.

PROGRAM :-

import csv

a = []

with open('enjoysport.csv', 'r') as csvfile:

for row in csv.reader(csvfile):

a.append(row)

print(a)

print("\n The total number of training instances are : ",len(a))

num\_attribute = len(a[0])-1

print("\n The initial hypothesis is : ")

hypothesis = ['0']\*num\_attribute

print(hypothesis)

for i in range(0, len(a)):

if a[i][num\_attribute] == 'yes':

for j in range(0, num\_attribute):

if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:

hypothesis[j] = a[i][j]

else:

hypothesis[j] = '?'

print("\n The hypothesis for the training instance {} is :\n" .format(i+1),hypothesis)

print("\n The Maximally specific hypothesis for the training instance is ")

print(hypothesis)

2) For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm in python 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('enjoysport.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("initialization 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] ='?'

print(specific\_h)

print(specific\_h)

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(" steps 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("Final Specific\_h:", s\_final, sep="\n")

print("Final General\_h:", g\_final, sep="\n")

3) 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

import numpy as np

import math

# Define a class for the decision tree node

class DecisionTreeNode:

def \_\_init\_\_(self, attribute=None, label=None, branches={}):

self.attribute = attribute # the attribute used to split the data

self.label = label # the label assigned to this node

self.branches = branches # the branches of the decision tree

# Define a function to calculate the entropy of a dataset

def entropy(data):

target = data['target']

n = len(target)

unique, counts = np.unique(target, return\_counts=True)

entropy = 0

for i in range(len(unique)):

p = counts[i] / n

entropy -= p \* math.log2(p)

return entropy

# Define a function to calculate the information gain of an attribute

def information\_gain(data, attribute):

n = len(data)

values = data[attribute].unique()

entropy\_s = entropy(data)

entropy\_attr = 0

for value in values:

subset = data[data[attribute] == value]

subset\_n = len(subset)

subset\_entropy = entropy(subset)

entropy\_attr += subset\_n / n \* subset\_entropy

return entropy\_s - entropy\_attr

# Define the ID3 algorithm

def id3(data, attributes):

target = data['target']

# If all the examples have the same target value, return a leaf node with that value

if len(target.unique()) == 1:

return DecisionTreeNode(label=target.iloc[0])

# If there are no attributes left to split on, return a leaf node with the most common target value

if len(attributes) == 0:

return DecisionTreeNode(label=target.value\_counts().idxmax())

# Otherwise, select the attribute with the highest information gain

gains = {attr: information\_gain(data, attr) for attr in attributes}

best\_attribute = max(gains, key=gains.get)

# Create a new decision tree node with the selected attribute

node = DecisionTreeNode(attribute=best\_attribute)

# Split the data based on the selected attribute and recursively build the tree

for value in data[best\_attribute].unique():

subset = data[data[best\_attribute] == value].drop(best\_attribute, axis=1)

if len(subset) == 0:

node.branches[value] = DecisionTreeNode(label=target.value\_counts().idxmax())

else:

new\_attributes = attributes.copy()

new\_attributes.remove(best\_attribute)

node.branches[value] = id3(subset, new\_attributes)

return node

# Load the dataset

data = pd.read\_csv('play\_tennis.csv')

# Split the dataset into attributes and target variable

attributes = data.columns[:-1].tolist()

# Build the decision tree using ID3 algorithm

root = id3(data, attributes)

# Define a function to classify a new sample using the decision tree

# Define a function to classify a new sample using the decision tree

def classify(sample, tree):

if tree.label is not None:

return tree.label

attribute = tree.attribute

value = sample[attribute]

if value not in tree.branches:

# If the value is not present in branches, return the label of the majority branch

majority\_branch = max(tree.branches, key=lambda k: len(tree.branches[k].branches))

return tree.branches[majority\_branch].label

subtree = tree.branches[value]

return classify(sample, subtree) # Recursively classify using the subtree

# Example usage

new\_sample = {'outlook': 'sunny', 'temperature': 'hot', 'humidity': 'high', 'windy': 'false'}

predicted\_label = classify(new\_sample, root)

print("Predicted label:", predicted\_label)

4) 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)

5) Write a program for Implementation of K-Nearest Neighbours (K-NN) in Python

PROGRAM :-

from math import sqrt

from statistics import mode

l=[[33.6,50,1],[26.6,30,0],[23.4,40,0],[43.1,67,0],[35.3,23,1],[35.9,67,1],[36.7,45,1],[25.7,46,0],[23.3,29,0],[31,56,1]]

n=[43.6,40]

k=3

m=[]

x=[]

for i in l:

a=0

for j in range(len(n)-1):

a+= (i[j]-n[j])\*(i[j]-n[j])

m.append(sqrt(a))

a=sorted(m)

for i in range(k):

x.append(m.index(a[i]))

y=[]

for i in x:

print(l[i])

y.append(l[i][-1])

print()

print("result -->",mode(y))

6) Write a program to implement Naive Bayes algorithm in python and to display the results using confusion matrix and

accuracy.

PROGRAM :-

# import required libraries

from sklearn.datasets import load\_iris

from sklearn.naive\_bayes import GaussianNB

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix, accuracy\_score

# load iris dataset

iris = load\_iris()

# split dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.3, random\_state=0)

# create Naive Bayes classifier

classifier = GaussianNB()

# train the classifier using the training data

classifier.fit(X\_train, y\_train)

# predict the target values for the testing data

y\_pred = classifier.predict(X\_test)

# display confusion matrix and accuracy score

cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(cm)

acc = accuracy\_score(y\_test, y\_pred)

print("Accuracy Score:", acc)

7) Write a program to implement Logistic Regression (LR) algorithm in python

PROGRAM :-

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

# Generate sample data

np.random.seed(0)

X = np.linspace(0, 10, 100).reshape(-1, 1)

y = 2 \* X + 1 + np.random.randn(100, 1)

# Create linear regression object

lr\_model = LinearRegression()

# Train the model using the training sets

lr\_model.fit(X, y)

# Print the coefficients

print('Coefficients: ', lr\_model.coef\_)

print('Intercept: ', lr\_model.intercept\_)

# Plot the data and the linear regression line

plt.scatter(X, y, color='blue')

plt.plot(X, lr\_model.predict(X), color='red', linewidth=3)

plt.title('Linear Regression')

plt.xlabel('X')

plt.ylabel('y')

plt.show()

8) Write a program to implement Linear Regression (LR) algorithm in python

PROGRAM :-

from sklearn.datasets import make\_classification

from matplotlib import pyplot as plt

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix

import pandas as pd

x, y = make\_classification(

n\_samples=100,

n\_features=1,

n\_classes=2,

n\_clusters\_per\_class=1,

flip\_y=0.03,

n\_informative=1,

n\_redundant=0,

n\_repeated=0

)

print(y)

plt.scatter(x, y, c=y, cmap='rainbow')

plt.title('Scatter Plot of Logistic Regression')

plt.show()

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, random\_state=1)

x\_train.shape

log\_reg = LogisticRegression()

log\_reg.fit(x\_train, y\_train)

y\_pred = log\_reg.predict(x\_test)

confusion\_matrix(y\_test, y\_pred)

9) Compare Linear and Polynomial Regression using Python

PROGRAM :-

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

from sklearn.pipeline import make\_pipeline

# Sample data

X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1) # Independent variable

y = np.array([2, 4, 5, 4, 6]) # Dependent variable

# Create a linear regression model

linear\_model = LinearRegression()

linear\_model.fit(X, y)

linear\_pred = linear\_model.predict(X)

# Create a polynomial regression model (degree 2)

degree = 2

poly\_model = make\_pipeline(PolynomialFeatures(degree), LinearRegression())

poly\_model.fit(X, y)

poly\_pred = poly\_model.predict(X)

# Plot the data, linear regression line, and polynomial regression curve

plt.scatter(X, y, label='Actual Data')

plt.plot(X, linear\_pred, color='blue', label='Linear Regression')

plt.plot(X, poly\_pred, color='red', label=f'Polynomial Regression (degree {degree})')

plt.xlabel('X')

plt.ylabel('y')

plt.legend()

plt.title('Linear vs Polynomial Regression')

plt.show()

10) Write a Python Program to Implement Expectation & Maximization Algorithm

PROGRAM :-

import numpy as np

from scipy.stats import multivariate\_normal

# Generate synthetic data

np.random.seed(0)

true\_mu1 = np.array([2, 2])

true\_cov1 = np.array([[1, 0.5], [0.5, 1]])

true\_mu2 = np.array([7, 7])

true\_cov2 = np.array([[1, -0.5], [-0.5, 1]])

true\_weights = [0.4, 0.6]

n\_samples = 300

n\_features = 2

X = np.concatenate([

np.random.multivariate\_normal(true\_mu1, true\_cov1, int(n\_samples \* true\_weights[0])),

np.random.multivariate\_normal(true\_mu2, true\_cov2, int(n\_samples \* true\_weights[1]))

], axis=0)

n\_clusters = 2

def e\_step(X, mus, sigmas, weights):

responsibilities = []

for i in range(n\_clusters):

pdf = multivariate\_normal.pdf(X, mean=mus[i], cov=sigmas[i])

responsibilities.append(weights[i] \* pdf)

responsibilities = np.array(responsibilities)

responsibilities /= np.sum(responsibilities, axis=0)

return responsibilities

def m\_step(X, responsibilities):

n\_samples, \_ = X.shape

mus = []

sigmas = []

weights = []

for i in range(n\_clusters):

r\_sum = np.sum(responsibilities[i])

weight = r\_sum / n\_samples

mu = np.sum(responsibilities[i][:, np.newaxis] \* X, axis=0) / r\_sum

sigma = (X - mu).T.dot((X - mu) \* responsibilities[i][:, np.newaxis]) / r\_sum

mus.append(mu)

sigmas.append(sigma)

weights.append(weight)

return np.array(mus), np.array(sigmas), np.array(weights)

# Initialize parameters randomly

initial\_weights = np.ones(n\_clusters) / n\_clusters

initial\_mus = np.random.rand(n\_clusters, n\_features) \* np.max(X, axis=0)

initial\_sigmas = np.array([np.eye(n\_features)] \* n\_clusters)

mus = initial\_mus

sigmas = initial\_sigmas

weights = initial\_weights

max\_iters = 100

tolerance = 1e-6

for i in range(max\_iters):

prev\_mus = mus.copy()

responsibilities = e\_step(X, mus, sigmas, weights)

mus, sigmas, weights = m\_step(X, responsibilities)

if np.allclose(prev\_mus, mus, atol=tolerance):

print(f"Converged after {i + 1} iterations.")

break

print("Estimated means:")

print(mus)

print("Estimated covariances:")

print(sigmas)

print("Estimated weights:")

print(weights)

11) Write a program for the task of Credit Score Classification.

PROGRAM :-

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import classification\_report, accuracy\_score

# Sample credit score data (features and labels)

# Replace this with your own dataset

X = np.array([[25, 50000], [30, 80000], [35, 75000], [22, 30000], [40, 100000], [28, 60000]])

y = np.array([0, 1, 1, 0, 1, 0]) # 0 represents bad credit, 1 represents good credit

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create a decision tree classifier

clf = DecisionTreeClassifier()

# Train the classifier on the training data

clf.fit(X\_train, y\_train)

# Make predictions on the testing data

y\_pred = clf.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

classification\_rep = classification\_report(y\_test, y\_pred, target\_names=['Bad Credit', 'Good Credit'])

# Print the results

print("Accuracy:", accuracy)

print("Classification Report:")

print(classification\_rep)

12) Implement Iris Flower Classification using KNN.

PROGRAM :-

import numpy as np

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report, accuracy\_score

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize the features (mean = 0, variance = 1)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Create a KNN classifier

k = 3 # Number of neighbors

clf = KNeighborsClassifier(n\_neighbors=k)

# Train the classifier on the training data

clf.fit(X\_train, y\_train)

# Make predictions on the testing data

y\_pred = clf.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

classification\_rep = classification\_report(y\_test, y\_pred, target\_names=iris.target\_names)

# Print the results

print("Accuracy:", accuracy)

print("Classification Report:")

print(classification\_rep)

13) Implement the Car Price Prediction Model using Python

PROGRAM :-

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean\_absolute\_error

# Importing the dataset

data = pd.read\_csv("CarPrice.csv")

# Data Exploration

data.head()

data.shape

data.isnull().sum() # Checking if the dataset has NULL Values

data.info()

data.describe()

data.CarName.unique()

# Drop the 'CarName' column

data = data.drop(columns=['CarName'])

# Convert categorical variables into numerical format using one-hot encoding

data = pd.get\_dummies(data, drop\_first=True)

# Prepare data for training

predict = "price"

X = data.drop([predict], axis=1)

y = data[predict]

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Training the Car Price Prediction Model using DecisionTreeRegressor

model = DecisionTreeRegressor(random\_state=42)

model.fit(X\_train, y\_train)

# Making predictions on the test set

predictions = model.predict(X\_test)

# Evaluating the model using Mean Absolute Error

mae = mean\_absolute\_error(y\_test, predictions)

print("Mean Absolute Error:", mae)

# Example usage: Predicting the price for a new car

new\_car\_features = pd.DataFrame({

'symboling': [4],

'wheelbase': [97.2],

'carlength': [172.0],

'carwidth': [65.4],

'carheight': [54.3],

'curbweight': [2330],

'enginesize': [109],

'boreratio': [3.19],

'stroke': [3.03],

'compressionratio': [9.0],

'horsepower': [85],

'peakrpm': [5800],

'citympg': [27],

'highwaympg': [32],

'fueltype\_gas': [1], # Example for fueltype: 'gas', you can set other columns based on your data

# Add other columns as required based on your data

})

# Align the new\_car\_features with the training data to ensure the same columns

new\_car\_features\_aligned = new\_car\_features.reindex(columns=X.columns, fill\_value=0)

# Making predictions on the new car data

predicted\_price = model.predict(new\_car\_features\_aligned)

print("Predicted Price for the new car:", predicted\_price[0])

14) Implement House price Prediction using appropriate machine learning algorithm

PROGRAM :-

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_absolute\_error

from sklearn.preprocessing import OneHotEncoder

# 1. Data Loading

data = pd.read\_csv("HousePricePrediction.csv")

# 2. Data Preprocessing

# One-hot encoding of categorical columns

categorical\_cols = ["MSSubClass", "MSZoning", "LotConfig", "BldgType", "Exterior1st"]

data = pd.get\_dummies(data, columns=categorical\_cols, drop\_first=True)

# Drop the ID column

data = data.drop(columns=["Id"])

data.dropna(inplace=True)

# Splitting data

X = data.drop("SalePrice", axis=1)

y = data["SalePrice"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# 3. Model Training

model = LinearRegression()

model.fit(X\_train, y\_train)

# 4. Evaluation

predictions = model.predict(X\_test)

mae = mean\_absolute\_error(y\_test, predictions)

print(f"Mean Absolute Error (MAE): ${mae:.2f}")

# After prediction

df\_output = pd.DataFrame({'Actual Prices': y\_test, 'Predicted Prices': predictions})

print(df\_output)

# If you wish to save this as a CSV

# df\_output.to\_csv('predicted\_prices.csv', index=False)

# 5. Prediction

# For a new dataset:

# new\_data\_transformed = ... (apply the same preprocessing as the training set)

# predicted\_prices = model.predict(new\_data\_transformed)

15) Implement Iris Flower Classification using Naive Bayes classifier

PROGRAM :- import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.datasets import load\_iris

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize the Naive Bayes classifier

model = GaussianNB()

# Train the model on the training data

model.fit(X\_train, y\_train)

# Make predictions on the test data

y\_pred = model.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.2f}")

# Display classification report

class\_names = iris.target\_names

report = classification\_report(y\_test, y\_pred, target\_names=class\_names)

print("Classification Report:\n", report)

16) Compare different types Classification Algorithms and evaluate their performance.

PROGRAM :-

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.datasets import load\_iris

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize classifiers

classifiers = {

'Decision Tree': DecisionTreeClassifier(random\_state=42),

'SVM': SVC(random\_state=42),

'Random Forest': RandomForestClassifier(random\_state=42)

}

# Train and evaluate classifiers

for name, clf in classifiers.items():

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred, target\_names=iris.target\_names)

print(f"Classifier: {name}")

print(f"Accuracy: {accuracy:.2f}")

print("Classification Report:\n", report)

print("="\*40)

17) Implement Mobile Price Prediction using appropriate machine learning algorithm

PROGRAM :-

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

#importing dataset

data = pd.read\_csv("mobile\_prices.csv")

print(data.head())

plt.figure(figsize=(12, 10))

sns.heatmap(data.corr(), annot=True, cmap="coolwarm", linecolor='white', linewidths=1)

#data preparation

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

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

x = StandardScaler().fit\_transform(x)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.20, random\_state=0)

# Logistic Regression algorithm provided by Scikit-learn:

from sklearn.linear\_model import LogisticRegression

lreg = LogisticRegression()

lreg.fit(x\_train, y\_train)

y\_pred = lreg.predict(x\_test)

#accuracy of the model:

accuracy = accuracy\_score(y\_test, y\_pred) \* 100

print("Accuracy of the Logistic Regression Model: ",accuracy)

#predictions made by the model:

print(y\_pred)

(unique, counts) = np.unique(y\_pred, return\_counts=True)

price\_range = np.asarray((unique, counts)).T

print(price\_range)

18) Implement Perceptron based IRIS classification

PROGRAM :-

from sklearn import datasets

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import Perceptron

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

iris = datasets.load\_iris()

X = iris.data[:, [2, 3]]

y = iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.3, random\_state=1, stratify=y)

sc = StandardScaler()

sc.fit(X\_train)

X\_train\_std = sc.transform(X\_train)

X\_test\_std = sc.transform(X\_test)

ppn = Perceptron(eta0=0.1, random\_state=1)

ppn.fit(X\_train\_std, y\_train)

y\_pred = ppn.predict(X\_test\_std)

print('Accuracy: %.3f' % accuracy\_score(y\_test, y\_pred))

print('Accuracy: %.3f' % ppn.score(X\_test\_std, y\_test))

19) Implementation of Naive Bayes classification for Bank Loan prediction

PROGRAM :-

import numpy as np

import pandas as pd

dataset = pd.read\_csv("breastcancer.csv")

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

y = dataset.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 = 0.25, random\_state = 0)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

classifier.fit(X\_train, y\_train)

GaussianNB(priors=None, var\_smoothing=1e-09)

from sklearn.metrics import confusion\_matrix, accuracy\_score

y\_pred = classifier.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

accuracy\_score(y\_test, y\_pred)

20) Implement Future Sales Prediction using a suitable machine learning algorithm

PROGRAM :-

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

import plotly.io as io

io.renderers.default='browser'

data = pd.read\_csv("futuresale prediction.csv")

print(data.head())

print(data.sample(5))

print(data.isnull().sum())

import plotly.express as px

import plotly.graph\_objects as go

figure = px.scatter(data\_frame = data, x="Sales",

y="TV", size="TV", trendline="ols")

figure.show()

figure = px.scatter(data\_frame = data, x="Sales",

y="Newspaper", size="Newspaper", trendline="ols")

figure.show()

figure = px.scatter(data\_frame = data, x="Sales",

y="Radio", size="Radio", trendline="ols")

figure.show()

correlation = data.corr()

print(correlation["Sales"].sort\_values(ascending=False))

x = np.array(data.drop(["Sales"], 1))

y = np.array(data["Sales"])

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.2, random\_state=42)

model = LinearRegression()

model.fit(xtrain, ytrain)

print(model.score(xtest, ytest))

features = [[TV, Radio, Newspaper]]

features = np.array([[230.1, 37.8, 69.2]])

print(model.predict(features))