1. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.

```
import csv
def find_s_algorithm(training_data):
    hypothesis = ['0'] * len(training_data[0][0])
    for example, label in training_data:
        if label == 'Y':
            for i, val in enumerate(example):
                if hypothesis[i] == '0':
                    hypothesis[i] = val
                elif hypothesis[i] != val:
                    hypothesis[i] = '?'
    return hypothesis
def read_csv_file(file_path):
    dataset = []
    with open(file_path, 'r') as file:
        reader = csv.reader(file)
        for row in reader:
            dataset.append((row[:-1], row[-1]))
    return dataset
file_path = 'C:/Users/student/Desktop/dataset/enjoysport (1).csv'
training_data = read_csv_file(file_path)
hypothesis = find_s_algorithm(training_data)
print("Final hypothesis:", 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

```
import csv
with open("C:/Users/student/Desktop/dataset/enjoysport (1).csv")as f:
    csv_file = csv.reader(f)
    data = list(csv_file)
    specific = data[1][:-1]
    general = [['?' for i in range(len(specific))] for j in range(len(specific))]

    for i in data:
        if i[-1] == "Yes":
            for j in range(len(specific)):
                if i[j] != specific[j]:
                    specific[j] = "?"
                    general[j][j] = "?"

        elif i[-1] == "No":
            for j in range(len(specific)):
                if i[j] != specific[j]:
                    general[j][j] = specific[j]
                else:
                    general[j][j] = "?"

        print("\nStep " + str(data.index(i)+1) + " of Candidate Elimination Algorithm")
        print(specific)
        print(general)

    gh = [] # gh = general Hypothesis
    for i in general:
        for j in i:
```

```
        if j != '?':
            gh.append(i)
            break
    print("\nFinal Specific hypothesis:\n", specific)
    print("\nFinal General hypothesis:\n", gh)
```

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.

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_text
iris=load_iris()
X=iris.data
Y=iris.target
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,random_state=42)
clf=DecisionTreeClassifier(criterion="entropy",random_state=42)
clf.fit(X_train,Y_train)
tree_rules=export_text(clf,feature_names=iris.feature_names)
print("Decision tree:/n",tree_rules)
Y_pred=clf.predict(X_test)
accuracy=np.mean(Y_pred==Y_test)
print("accuracy:",accuracy)
```

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

```
import numpy as np


# Define the sigmoid activation function and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

```
def sigmoid_derivative(x):
    return x * (1 - x)

# Define the Neural Network class
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        # Initialize weights and biases
        self.weights_input_hidden = np.random.uniform(-1, 1, (self.input_size, self.hidden_size))
        self.bias_hidden = np.zeros((1, self.hidden_size))

        self.weights_hidden_output = np.random.uniform(-1, 1, (self.hidden_size, self.output_size))
        self.bias_output = np.zeros((1, self.output_size))

    def forward(self, X):
        # Forward propagation
        self.hidden_input = np.dot(X, self.weights_input_hidden) + self.bias_hidden
        self.hidden_output = sigmoid(self.hidden_input)

        self.output_input = np.dot(self.hidden_output, self.weights_hidden_output) + self.bias_output
        self.predicted_output = sigmoid(self.output_input)

        return self.predicted_output

    def backward(self, X, y, learning_rate):
        # Backpropagation
        error = y - self.predicted_output
```

```
        d_output = error * sigmoid_derivative(self.predicted_output)
        d_hidden = d_output.dot(self.weights_hidden_output.T) *
sigmoid_derivative(self.hidden_output)

        self.weights_hidden_output += self.hidden_output.T.dot(d_output) * learning_rate
        self.bias_output += np.sum(d_output) * learning_rate

        self.weights_input_hidden += X.T.dot(d_hidden) * learning_rate
        self.bias_hidden += np.sum(d_hidden) * learning_rate

    def train(self, X, y, epochs, learning_rate):
        for epoch in range(epochs):
            predicted_output = self.forward(X)
            self.backward(X, y, learning_rate)
            if epoch % 1000 == 0:
                loss = np.mean(np.square(y - predicted_output))
                print(f"Epoch {epoch}, Loss: {loss:.4f}")

# Define XOR dataset
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])

# Create and train the neural network
input_size = 2
hidden_size = 4
output_size = 1
learning_rate = 0.1
epochs = 10000

nn = NeuralNetwork(input_size, hidden_size, output_size)
```

```
nn.train(X, y, epochs, learning_rate)


# Test the trained model
test_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
predictions = nn.forward(test_data)
print("Predictions:")
print(predictions)
```

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

```
#IMPORTING THE MODULES
import numpy as np
import pandas as pd


import matplotlib.pyplot as plt


from sklearn.datasets import make_blobs
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
#CREATING THE DATASETS
X, y = make_blobs(n_samples = 500, n_features = 2, centers = 4,cluster_std = 1.5, random_state = 4)
#VISUALIZE THE DATASETS
plt.style.use('seaborn')
plt.figure(figsize = (10,10))
plt.scatter(X[:,0], X[:,1], c=y, marker= '*',s=100,edgecolors='cyan')
plt.show()
#SPLITTING DATA INTO TRAINING SETS AND TESTING SETS
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)
#KNN CLASSIFIERS IMPLEMENTATION
knn5 = KNeighborsClassifier(n_neighbors = 5)
knn1 = KNeighborsClassifier(n_neighbors=1)
#PREDICITION FOR THE KNN CLASSIFIERS
```

```
knn5.fit(X_train, y_train)
knn1.fit(X_train, y_train)

y_pred_5 = knn5.predict(X_test)
y_pred_1 = knn1.predict(X_test)
#PREDICT ACCURACY FOR BOTH K VALUES
from sklearn.metrics import accuracy_score
print("Accuracy with k=5", accuracy_score(y_test, y_pred_5)*100)
print("Accuracy with k=1", accuracy_score(y_test, y_pred_1)*100)
#VISUALIZING THE PREDICITON
plt.figure(figsize = (15,5))
plt.subplot(1,2,1)
plt.scatter(X_test[:,0], X_test[:,1], c=y_pred_5, marker= '*', s=100,edgecolors='black')
plt.title("Predicted values with k=5", fontsize=20)

plt.subplot(1,2,2)
plt.scatter(X_test[:,0], X_test[:,1], c=y_pred_1, marker= '*', s=100,edgecolors='black')
plt.title("Predicted values with k=1", fontsize=20)
plt.show()
```

6. Write a program to implement Naïve Bayes algorithm in python and to display the results using confusion matrix and accuracy.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score

# Load the Iris dataset
iris = load_iris()
X, y = iris.data, 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)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create a Naïve Bayes classifier
naive_bayes_classifier = GaussianNB()

# Train the classifier
naive_bayes_classifier.fit(X_train, y_train)

# Make predictions on the test set
predictions = naive_bayes_classifier.predict(X_test)

# Calculate the accuracy
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy: {accuracy:.2f}")

# Display confusion matrix
conf_matrix = confusion_matrix(y_test, predictions)
print("Confusion Matrix:")
print(conf_matrix)
```

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

```
from sklearn.datasets import load_iris
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
```

```
# Load the Iris dataset
iris = load_iris()
X, y = iris.data, 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=0.9, random_state=90)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create a Logistic Regression classifier
logreg_classifier = LogisticRegression()

# Train the classifier
logreg_classifier.fit(X_train, y_train)

# Make predictions on the test set
predictions = logreg_classifier.predict(X_test)

# Calculate the accuracy
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy: {accuracy:.2f}")
```

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

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
```

```
# Generate some sample data
np.random.seed(0)
X = np.random.rand(100, 1) * 10
y = 2 * X + 1 + np.random.randn(100, 1) * 2

# 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 Linear Regression model
linear_regressor = LinearRegression()

# Train the model
linear_regressor.fit(X_train, y_train)

# Make predictions on the test set
predictions = linear_regressor.predict(X_test)

# Calculate metrics
mse = mean_squared_error(y_test, predictions)
r2 = r2_score(y_test, predictions)

print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")

# Plot the results
plt.scatter(X_test, y_test, color='blue')
plt.plot(X_test, predictions, color='red', linewidth=2)
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression')
```

```
plt.show()
```

9.Compare Linear and Polynomial Regression using Python(LINEAR)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
# Fit the model
model = LinearRegression()
model.fit(np.array(x).reshape((-1, 1)), y)
# Plot the data and the fitted line
plt.scatter(x, y)
plt.plot(x, model.predict(np.array(x).reshape((-1, 1))), color='red')
plt.show()
# Create a polynomial features object
poly_features = PolynomialFeatures(degree=2)
# Transform the data
x_poly = poly_features.fit_transform(np.array(x).reshape((-1, 1)))
# Fit the model
model = LinearRegression()
model.fit(x_poly, y)
# Plot the data and the fitted line
plt.scatter(x, y)
plt.plot(x, model.predict(x_poly), color='red')
plt.show()
```

9. Compare Linear and Polynomial Regression using Python(POLYNOMIAL)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
```

```
x = [1, 2, 3, 4, 5]

y = [2, 4, 6, 8, 10]

# Create a polynomial features object

poly_features = PolynomialFeatures(degree=2)

# Transform the data

x_poly = poly_features.fit_transform(np.array(x).reshape((-1, 1)))

# Fit the model

model = LinearRegression()

model.fit(x_poly, y)

# Plot the data and the fitted line

plt.scatter(x, y)

plt.plot(x, model.predict(x_poly), color='red')

plt.show()
```

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

```
import numpy as np

from scipy.stats import norm


# Define the data

data = np.array([1.2, 2.3, 0.7, 1.6, 1.1, 1.8, 0.9, 2.2])


# Initialize the parameters

mu1 = 0

mu2 = 1

sigma1 = 1

sigma2 = 1

p1 = 0.5

p2 = 0.5


# Run the EM algorithm

for i in range(10):
```

```
    # E-step
    likelihood1 = norm.pdf(data, mu1, sigma1)
    likelihood2 = norm.pdf(data, mu2, sigma2)
    weight1 = p1 * likelihood1 / (p1 * likelihood1 + p2 * likelihood2)
    weight2 = p2 * likelihood2 / (p1 * likelihood1 + p2 * likelihood2)

    # M-step
    mu1 = np.sum(weight1 * data) / np.sum(weight1)
    mu2 = np.sum(weight2 * data) / np.sum(weight2)
    sigma1 = np.sqrt(np.sum(weight1 * (data - mu1)**2) / np.sum(weight1))
    sigma2 = np.sqrt(np.sum(weight2 * (data - mu2)**2) / np.sum(weight2))
    p1 = np.mean(weight1)
    p2 = np.mean(weight2)

# Print the final estimates of the parameters
print("mu1:", mu1)
print("mu2:", mu2)
print("sigma1:", sigma1)
print("sigma2:", sigma2)
print("p1:", p1)
print("p2:", p2)
```

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

```
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
```

```
pio.templates.default = "plotly_white"

data = pd.read_csv("CREDITSCORE.csv")
print(data.head())

print(data.info())

x = data[["Annual_Income", "Monthly_Inhand_Salary",
        "Num_Bank_Accounts", "Num_Credit_Card",
        "Interest_Rate", "Num_of_Loan",
        "Delay_from_due_date", "Num_of_Delayed_Payment",
        "Credit_Mix", "Outstanding_Debt",
        "Credit_History_Age", "Monthly_Balance"]]
y = data[["Credit_Score"]]

# Convert "Credit_Mix" to numerical representation using .loc method
x.loc[:, "Credit_Mix"] = x["Credit_Mix"].map({"Bad": 0, "Standard": 1, "Good": 3})

# Handle missing values (if any)
# x.fillna(0, inplace=True)

# Scale numerical features
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)

xtrain, xtest, ytrain, ytest = train_test_split(x_scaled, y,
                                    test_size=0.33,
                                    random_state=42)

model = RandomForestClassifier()
model.fit(xtrain, ytrain.values.ravel())
```

```
print("Credit Score Prediction : ")
a = float(input("Annual Income: "))
b = float(input("Monthly Inhand Salary: "))
c = float(input("Number of Bank Accounts: "))
d = float(input("Number of Credit cards: "))
e = float(input("Interest rate: "))
f = float(input("Number of Loans: "))
g = float(input("Average number of days delayed by the person: "))
h = float(input("Number of delayed payments: "))
i = input("Credit Mix (Bad: 0, Standard: 1, Good: 3) : ")
j = float(input("Outstanding Debt: "))
k = float(input("Credit History Age: "))
l = float(input("Monthly Balance: "))

# Convert user input for Credit Mix to integer
credit_mix_map = {"Bad": 0, "Standard": 1, "Good": 3}
i = credit_mix_map.get(i)

features = np.array([[a, b, c, d, e, f, g, h, i, j, k, l]])
print("Predicted Credit Score = ", model.predict(features))
```

12. Implement Iris Flower Classification using KNN

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
iris = pd.read_csv("IRIS.csv")
print(iris.head())
print()
print(iris.describe())
print("Target Labels", iris["species"].unique())
```

```
import plotly.io as io
import plotly.express as px
fig = px.scatter(iris, x="sepal_width", y="sepal_length", color="species")
fig.show()
x = iris.drop("species", axis=1)
y = iris["species"]
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=0)

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(x_train, y_train)

x_new = np.array([[6, 2.9, 1, 0.2]])
prediction = knn.predict(x_new)
print("Prediction: {}".format(prediction))
```

13. Implement the Car Price Prediction Model using Python

```
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.tree import DecisionTreeRegressor

#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()

#Analysing correlations & using heatmap
print(data.corr())
plt.figure(figsize=(20, 15))
correlations = data.corr()
sns.heatmap(correlations, cmap="coolwarm", annot=True)
plt.show()

#Training a Car Price Prediction Model
predict = "price"
data = data[["symboling", "wheelbase", "carlength",
        "carwidth", "carheight", "curbweight",
        "enginesize", "boreratio", "stroke",
        "compressionratio", "horsepower", "peakrpm",
        "citympg", "highwaympg", "price"]]
x = np.array(data.drop([predict], 1))
y = np.array(data[predict])

from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.2)

from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor()
model.fit(xtrain, ytrain)
predictions = model.predict(xtest)

from sklearn.metrics import mean_absolute_error
```

```
model.score(xtest, predictions)
```

14. Implement House price Prediction using appropriate machine learning algorithm

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#Importing Dataset
dataset = pd.read_csv("HousePricePrediction.csv")
#Exploring dataset
print(dataset.head(5))
dataset.shape
obj = (dataset.dtypes == 'object')
object_cols = list(obj[obj].index)
print("Categorical variables:",len(object_cols))

int_ = (dataset.dtypes == 'int')
num_cols = list(int_[int_].index)
print("Integer variables:",len(num_cols))

fl = (dataset.dtypes == 'float')
fl_cols = list(fl[fl].index)
print("Float variables:",len(fl_cols))

plt.figure(figsize=(12, 6))
sns.heatmap(dataset.corr(),
        cmap = 'BrBG',
        fmt = '.2f',
        linewidths = 2,
        annot = True)
```

```
unique_values = []
for col in object_cols:
    unique_values.append(dataset[col].unique().size)
    plt.figure(figsize=(10,6))
    plt.title('No. Unique values of Categorical Features')
    plt.xticks(rotation=90)
sns.barplot(x=object_cols,y=unique_values)

plt.figure(figsize=(18, 36))
plt.title('Categorical Features: Distribution')
plt.xticks(rotation=90)
index = 1

for col in object_cols:
        y = dataset[col].value_counts()
        plt.subplot(11, 4, index)
        plt.xticks(rotation=90)
        sns.barplot(x=list(y.index), y=y)
        index += 1
dataset.drop(['Id'],axis=1,inplace=True)
dataset['SalePrice'] = dataset['SalePrice'].fillna(dataset['SalePrice'].mean())
new_dataset = dataset.dropna()
new_dataset.isnull().sum()

from sklearn.preprocessing import OneHotEncoder
s = (new_dataset.dtypes == 'object')
object_cols = list(s[s].index)
print("Categorical variables:")
print(object_cols)
print('No. of. categorical features: ',len(object_cols))
```

```
OH_encoder = OneHotEncoder(sparse=False)
OH_cols = pd.DataFrame(OH_encoder.fit_transform(new_dataset[object_cols]))
OH_cols.index = new_dataset.index
OH_cols.columns = OH_encoder.get_feature_names()
df_final = new_dataset.drop(object_cols, axis=1)
df_final = pd.concat([df_final, OH_cols], axis=1)
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split

X = df_final.drop(['SalePrice'], axis=1)
Y = df_final['SalePrice']

X_train, X_valid, Y_train, Y_valid = train_test_split(X, Y, train_size=0.8, test_size=0.2,
random_state=0)

from sklearn import svm
from sklearn.svm import SVC
from sklearn.metrics import mean_absolute_percentage_error

model_SVR = svm.SVR()
model_SVR.fit(X_train,Y_train)
Y_pred = model_SVR.predict(X_valid)

print(mean_absolute_percentage_error(Y_valid, Y_pred))

#LinearRegression
from sklearn.linear_model import LinearRegression

model_LR = LinearRegression()
model_LR.fit(X_train, Y_train)
Y_pred = model_LR.predict(X_valid)
```

```
print(mean_absolute_percentage_error(Y_valid, Y_pred))
```

15. Implement Iris Flower Classification using Naive Bayes classifier

```
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn import datasets
from sklearn.metrics import confusion_matrix
#Load the iris dataset
iris = datasets.load_iris()
#GaussianNB and MultinomialNB Models
gnb = GaussianNB()
mnb = MultinomialNB()
#Train both GaussianNB and MultinomialNB Models and print their confusion matrices
y_pred_gnb = gnb.fit(iris.data, iris.target).predict(iris.data)
cnf_matrix_gnb = confusion_matrix(iris.target, y_pred_gnb)
print("Confusion Matrix of GNB \n",cnf_matrix_gnb)

y_pred_mnb = mnb.fit(iris.data, iris.target).predict(iris.data)
cnf_matrix_mnb = confusion_matrix(iris.target, y_pred_mnb)
print("Confusion Matrix of MNB \n",cnf_matrix_mnb)
```

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

```
import numpy
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import BernoulliNB
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import PassiveAggressiveClassifier
from sklearn.metrics import classification_report

iris= pd.read_csv("C:/Users/Win10/Downloads/IRIS.csv")
print(iris.head())

x = iris.drop("species", axis=1)
y = iris["species"]
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,test_size=0,random_state=42)

#x = np.array(data[["Age", "EstimatedSalary"]])
#y = np.array(data[["Purchased"]])

#xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.10, random_state=42)
decisiontree = DecisionTreeClassifier()
logisticregression = LogisticRegression()
knearestclassifier = KNeighborsClassifier()
#svm_classifier = SVC()
bernoulli_naiveBayes = BernoulliNB()
passiveAggressive = PassiveAggressiveClassifier()

knearestclassifier.fit(x_train, y_train)
decisiontree.fit(x_train, y_train)
logisticregression.fit(x_train, y_train)
passiveAggressive.fit(x_train, y_train)

data1 = {"Classification Algorithms": ["KNN Classifier", "Decision Tree Classifier",
                                       "Logistic Regression", "Passive Aggressive Classifier"],
    "Score": [knearestclassifier.score(x,y), decisiontree.score(x, y),
```

```
            logisticregression.score(x, y), passiveAggressive.score(x,y) ]}
score = pd.DataFrame(data1)
score
```

17. Implement Mobile Price Prediction using appropriate machine learning algorithm

```
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

```
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. Implement Future Sales Prediction using a suitable machine learning algorithm

```
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

```
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))
```