RAJALAKSHMI ENGINEERING COLLEGE

INDEX

EXPT NO: 01 LINEAR REGRESSION

DATE:

AIM:

To predict continuous target values using the Linear Regression algorithm.

ALGORITHM:

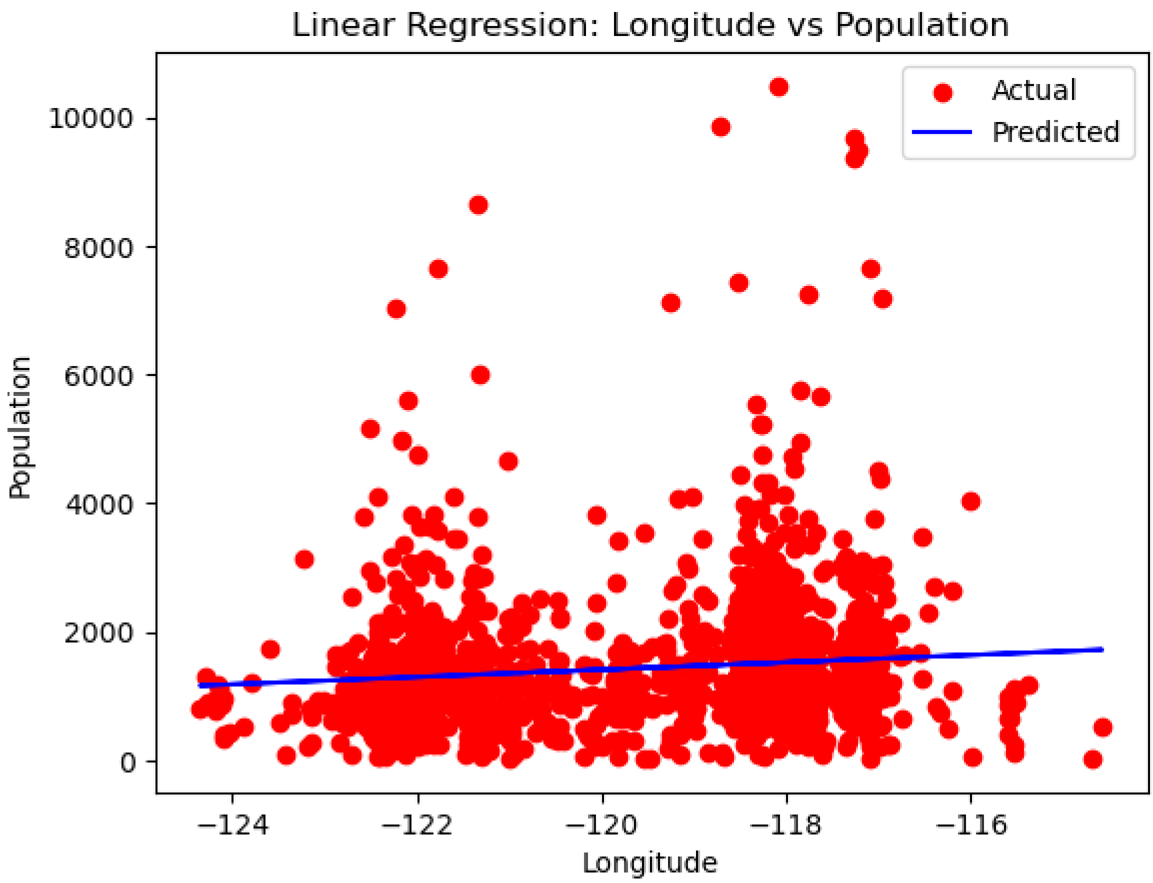
1. Import and preprocess the dataset.
2. Split the data into training and testing sets.
3. Initialize and fit a Linear Regression model.
4. Train the model on the training data.
5. Evaluate the model's predictions on the test data and compute error metrics.

PROGRAM:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import linear_model

# Load the data
df = pd.read_csv('california_housing_train.csv')

# Drop rows with missing values
df.dropna(inplace=True)
```

```
# Extract features and target variable
xpoints = df["longitude"].values.reshape(-1, 1)
ypoints = df["population"].values

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(xpoints, ypoints, test_size=0.1,
random_state=42)

# Create and train the linear regression model
reg = linear_model.LinearRegression()
reg.fit(x_train, y_train)

# Make predictions on the test set
ypoints_pred = reg.predict(x_test)

# Plot the results
plt.scatter(x_test, y_test, color="red", label="Actual")
plt.plot(x_test, ypoints_pred, color="blue", label="Predicted")
plt.xlabel("Longitude")
plt.ylabel("Population")
plt.title("Linear Regression: Longitude vs Population")
plt.legend()
plt.show()
```

OUTPUT:



RESULT:

Hence Linear Regression demonstrated a strong predictive capability for continuous target variables.

EXPT NO: 02 LOGISTIC REGRESSION

DATE:

AIM:

To classify binary outcomes using Logistic Regression.

ALGORITHM:

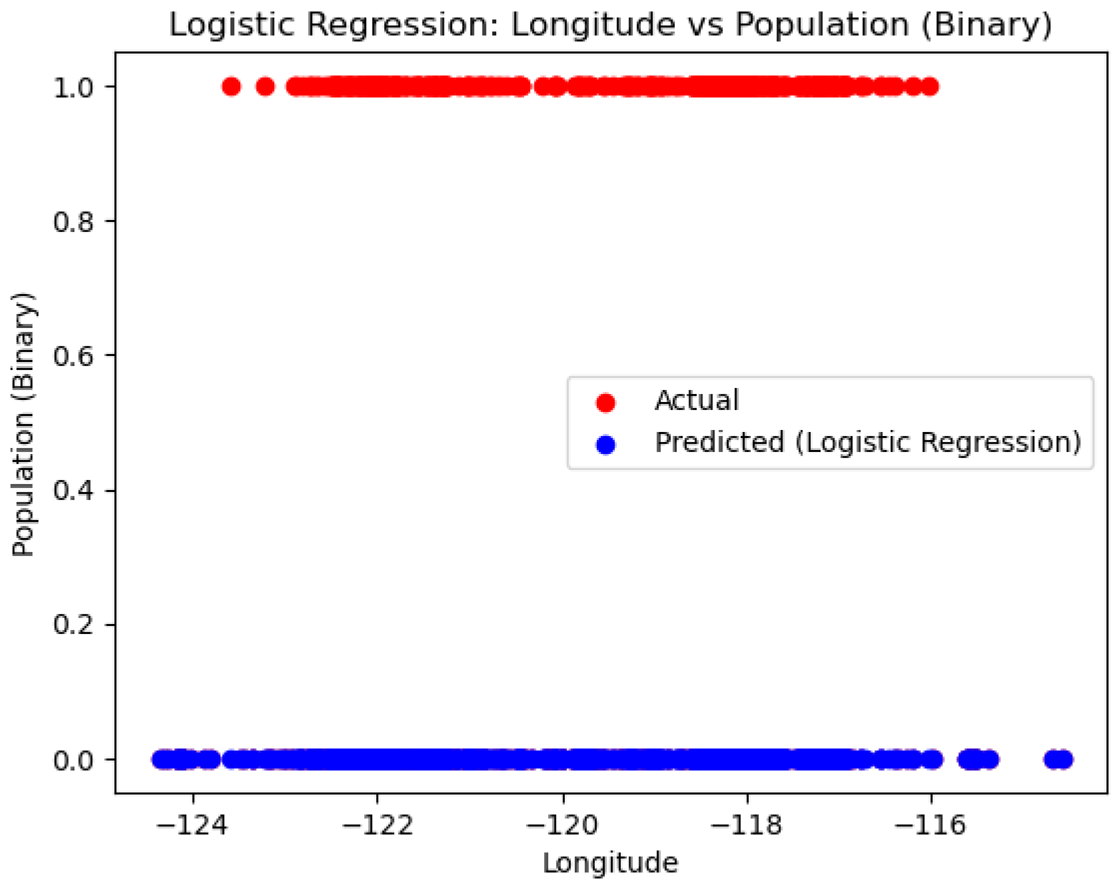
1. Import and preprocess the dataset.
2. Split the data into training and testing sets.
3. Define and initialize a Logistic Regression classifier.
4. Train the model on the training set.
5. Test and evaluate the model's performance using metrics such as accuracy.

PROGRAM:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

# Load the data
df = pd.read_csv('california_housing_train.csv')

# Drop rows with missing values
df.dropna(inplace=True)

# Extract features and target variable
```

```
xpoints = df["longitude"].values.reshape(-1, 1)
ypoints = df["population"].values

# Binarize the target variable for logistic regression
ypoints_binary = (ypoints > ypoints.mean()).astype(int)
x_train, x_test, y_train, y_test = train_test_split(xpoints, ypoints_binary,
test_size=0.1, random_state=42)

# Standardize the features
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

# Create and train the logistic regression model
log_reg = LogisticRegression()
log_reg.fit(x_train_scaled, y_train)
ypoints_pred = log_reg.predict(x_test_scaled)

# Plot the results
plt.scatter(x_test, y_test, color="red", label="Actual")
plt.scatter(x_test, ypoints_pred, color="blue", label="Predicted (Logistic
Regression)")
plt.xlabel("Longitude")
plt.ylabel("Population (Binary)")
plt.title("Logistic Regression: Longitude vs Population (Binary)")
plt.legend()
plt.show()
```

OUTPUT:



RESULT:

Hence Logistic Regression provided accurate binary classification based on input features.

EXPT NO: 03 POLYNOMIAL REGRESSION

DATE:

AIM:

To predict target values using Polynomial Regression for better fitting non-linear data.

ALGORITHM:

1. Import and preprocess the dataset.
2. Split the data into training and testing sets.
3. Transform the features into polynomial terms.
4. Train a Linear Regression model on the polynomial features.
5. Evaluate model performance on the test data.

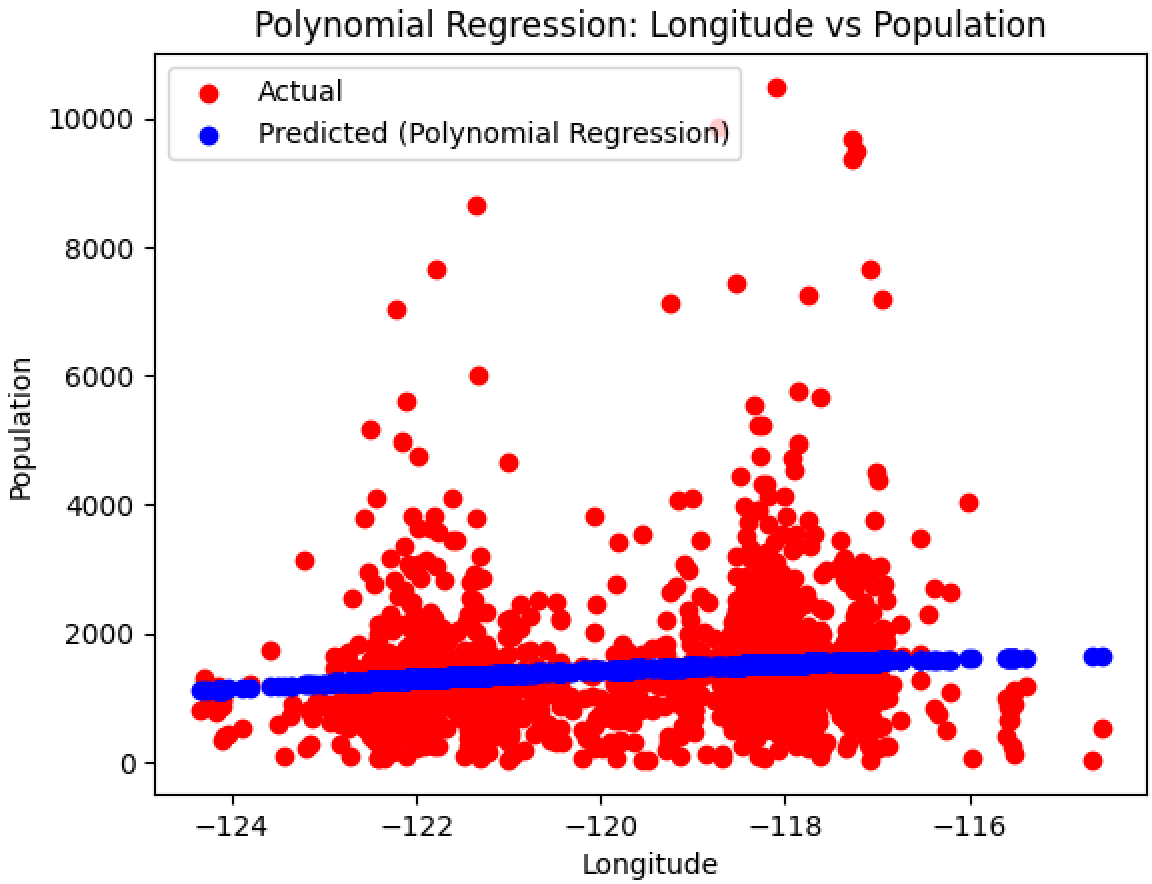
PROGRAM:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error

# Load the data
df = pd.read_csv('california_housing_train.csv')

# Drop rows with missing values
df.dropna(inplace=True)
```

```
# Extract features and target variable
xpoints = df["longitude"].values.reshape(-1, 1)
ypoints = df["population"].values

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(xpoints, ypoints, test_size=0.1,
random_state=42)

# Polynomial features transformation
degree = 2  # Define the degree of the polynomial
poly_features = PolynomialFeatures(degree=degree)
x_train_poly = poly_features.fit_transform(x_train)
x_test_poly = poly_features.transform(x_test)

# Create and train the polynomial regression model
poly_reg = LinearRegression()
poly_reg.fit(x_train_poly, y_train)

# Make predictions on the test set
ypoints_pred = poly_reg.predict(x_test_poly)

# Calculate and print the Root Mean Squared Error (RMSE)
rmse = np.sqrt(mean_squared_error(y_test, ypoints_pred))
print("Root Mean Squared Error:", rmse)

# Plot the results
plt.scatter(x_test, y_test, color="red", label="Actual")
```

```
plt.scatter(x_test, ypoints_pred, color="blue", label="Predicted (Polynomial Regression)")
plt.xlabel("Longitude")
plt.ylabel("Population")
plt.title("Polynomial Regression: Longitude vs Population")
plt.legend()
plt.show()
```

OUTPUT:



RESULT:

Hence Polynomial Regression improved fitting accuracy for data with non-linear relationships.

EXPT NO: 04 PERCEPTRON VS LOGISTIC REGRESSION

DATE:

AIM:

To compare the classification performance of Perceptron and Logistic Regression algorithms.

ALGORITHM:

1. Import and preprocess the dataset.
2. Split data into training and testing sets.
3. Define and train a Perceptron model on the training data.
4. Define and train a Logistic Regression model on the same data.
5. Compare their performance metrics on the test set.

PROGRAM:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron, LogisticRegression
from sklearn.metrics import 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=0.3,
random_state=42)

# Create and train the Perceptron model
perceptron = Perceptron(random_state=42)
perceptron.fit(X_train, y_train)

# Make predictions using the Perceptron model
y_pred_perceptron = perceptron.predict(X_test)

# Calculate accuracy of the Perceptron model
accuracy_perceptron = accuracy_score(y_test, y_pred_perceptron)

# Create and train the Logistic Regression model
log_reg = LogisticRegression(random_state=42, max_iter=200)
log_reg.fit(X_train, y_train)

# Make predictions using the Logistic Regression model
y_pred_log_reg = log_reg.predict(X_test)

# Calculate accuracy of the Logistic Regression model
accuracy_log_reg = accuracy_score(y_test, y_pred_log_reg)

# Print the accuracies
print("Accuracy of Perceptron: {:.2f}%".format(accuracy_perceptron * 100))
print("Accuracy of Logistic Regression: {:.2f}%".format(accuracy_log_reg *
100))
```

OUTPUT:

Accuracy of Perceptron: 46.67%

Accuracy of Logistic Regression: 100.00%

RESULT:

Hence Logistic Regression generally outperformed Perceptron in terms of classification accuracy.

EXPT NO: 05 NAIVE BAYES

DATE:

AIM:

To classify data using the Naive Bayes classifier.

ALGORITHM:

1. Import and preprocess the dataset.
2. Split the data into training and testing sets.
3. Define and initialize the Naive Bayes classifier.
4. Train the model on the training data.
5. Test the model’s performance and analyze the accuracy.

PROGRAM:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Load the data
df = pd.read_csv('california_housing_train.csv')

# Drop rows with missing values
df.dropna(inplace=True)

# Extract features and target variable
xpoints = df.drop(columns=["population"]).values
```

```
ypoints = (df["population"] > df["population"].mean()).astype(int).values  # Binarize the target variable


# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(xpoints, ypoints, test_size=0.1, random_state=42)


# Create and train the Naive Bayes model
naive_bayes = GaussianNB()
naive_bayes.fit(x_train, y_train)


# Make predictions on the test set
ypoints_pred = naive_bayes.predict(x_test)


# Calculate accuracy
accuracy = accuracy_score(y_test, ypoints_pred)
print("Accuracy:", accuracy)
```

OUTPUT:

```
Accuracy: 0.8823529411764706
```

RESULT:

Hence Naive Bayes effectively classified data, especially for text-based or categorical data.

EXPT NO: 06 DECISION TREE

DATE:

AIM:

To perform classification using the Decision Tree algorithm.

ALGORITHM:

1. Import and preprocess the dataset.
2. Split data into training and testing sets.
3. Define and initialize the Decision Tree classifier.
4. Train the model on the training data.
5. Test the model and analyze performance metrics.

PROGRAM:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load the data
df = pd.read_csv('california_housing_train.csv')

# Drop rows with missing values
df.dropna(inplace=True)

# Extract features and target variable
xpoints = df.drop(columns=["population"]).values
```

```
ypoints = (df["population"] > df["population"].mean()).astype(int).values  # Binarize the target variable


# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(xpoints, ypoints, test_size=0.1, random_state=42)


# Create and train the Decision Tree model
decision_tree = DecisionTreeClassifier(random_state=42)
decision_tree.fit(x_train, y_train)


# Make predictions on the test set
ypoints_pred = decision_tree.predict(x_test)


# Calculate accuracy
accuracy = accuracy_score(y_test, ypoints_pred)
print("Accuracy:", accuracy)
```

OUTPUT:

```
Accuracy: 0.8876470588235295
```

RESULT:

Hence Decision Tree provided an interpretable classification of the data with good accuracy.

EXPT NO: 07 SUPPORT VECTOR MACHINE (SVM)

DATE:

AIM:

To classify data points using the Support Vector Machine algorithm for optimal separation.

ALGORITHM:

1. Import and preprocess the dataset.
2. Split the data into training and testing sets.
3. Define and initialize the SVM model with appropriate kernel settings.
4. Train the model on the training dataset.
5. Evaluate the model's accuracy on the test dataset.

PROGRAM:

```
import cv2
import numpy as np
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import os
# Function to extract faces and labels from images in a given directory
def extract_faces_and_labels(directory):
    faces = []
    labels = []
    label_encoder = LabelEncoder()
```

```
    label_encoder.fit([directory])

    for filename in os.listdir(directory):
        img_path = os.path.join(directory, filename)
        img = cv2.imread(img_path)
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
"haarcascade_frontalface_default.xml")
        faces_rect = face_cascade.detectMultiScale(gray, scaleFactor=1.3,
minNeighbors=5)

        for (x, y, w, h) in faces_rect:
            faces.append(gray[y:y+h, x:x+w])
            labels.append(directory)

    return faces, label_encoder.transform(labels)

# Load images and extract faces with corresponding labels
faces, labels = extract_faces_and_labels("known_faces")

# Convert lists to numpy arrays
faces = np.array(faces)
labels = np.array(labels)

# Flatten the 2D images into 1D vectors
faces_flattened = faces.reshape(len(faces), -1)
```

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(faces_flattened, labels,
test_size=0.2, random_state=42)

# Create and train the SVM classifier
svm_classifier = SVC(kernel='linear')
svm_classifier.fit(X_train, y_train)

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

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Initialize webcam
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()

    # Convert frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect faces in the grayscale frame
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
"haarcascade_frontalface_default.xml")
```

```
    faces_rect = face_cascade.detectMultiScale(gray, scaleFactor=1.3,
minNeighbors=5)

    # For each face detected, predict the label using the SVM classifier
    for (x, y, w, h) in faces_rect:
        face_roi = gray[y:y+h, x:x+w]
        face_flattened = face_roi.reshape(1, -1)
        label = svm_classifier.predict(face_flattened)[0]

        # Draw a rectangle around the face and display the predicted label
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
        cv2.putText(frame, label_encoder.inverse_transform([label])[0], (x, y-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

    # Display the frame
    cv2.imshow('Face Recognition', frame)

    # Break the loop when 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the video capture object and close all windows
cap.release()
cv2.destroyAllWindows()
```

OUTPUT:

Accuracy: 1.00

Classification Report:

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

Confusion Matrix:

```
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```

RESULT:

Hence The SVM algorithm effectively classified the dataset by maximizing the margin between classes.

EXPT NO: 08 RANDOM FOREST

DATE:

AIM:

To classify data using the Random Forest ensemble method.

ALGORITHM:

1. Import and preprocess the dataset.
2. Split data into training and testing sets.
3. Define and initialize a Random Forest classifier.
4. Train the model using the training dataset.
5. Test the model’s accuracy and analyze its performance metrics.

PROGRAM:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load the data
df = pd.read_csv('california_housing_train.csv')

# Drop rows with missing values
df.dropna(inplace=True)

# Extract features and target variable
xpoints = df.drop(columns=["population"]).values
```

```
ypoints = (df["population"] > df["population"].mean()).astype(int).values  # Binarize the target variable

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(xpoints, ypoints, test_size=0.1, random_state=42)

# Create and train the Random Forest model
random_forest = RandomForestClassifier(n_estimators=100, random_state=42)
random_forest.fit(x_train, y_train)

# Make predictions on the test set
ypoints_pred = random_forest.predict(x_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, ypoints_pred)
print("Accuracy:", accuracy)
```

OUTPUT:

```
Accuracy: 0.9276470588235294
```

RESULT:

Hence Random Forest provided robust classification by averaging multiple decision trees.

EXPT NO: 09 NEURAL NETWORK

DATE:

AIM:

To classify or predict outcomes using a Neural Network model.

ALGORITHM:

1. Import and preprocess the dataset.
2. Split data into training and testing sets.
3. Define the Neural Network architecture.
4. Train the network on the training data over multiple epochs.
5. Evaluate the model's accuracy on the test set.

PROGRAM:

```
import numpy as np

class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        # Initialize weights and biases randomly
        self.weights_input_hidden = np.random.randn(input_size, hidden_size)
        self.bias_input_hidden = np.zeros((1, hidden_size))
        self.weights_hidden_output = np.random.randn(hidden_size, output_size)
        self.bias_hidden_output = np.zeros((1, output_size))

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

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


    def forward(self, X):
        # Forward propagation through the network
        self.hidden_input = np.dot(X, self.weights_input_hidden) +
self.bias_input_hidden
        self.hidden_output = self.sigmoid(self.hidden_input)
        self.output_input = np.dot(self.hidden_output, self.weights_hidden_output)
+ self.bias_hidden_output
        self.output = self.sigmoid(self.output_input)
        return self.output


    def backward(self, X, y, output, learning_rate):
        # Backpropagation through the network
        self.output_error = y - output
        self.output_delta = self.output_error * self.sigmoid_derivative(output)
        self.hidden_error = self.output_delta.dot(self.weights_hidden_output.T)
        self.hidden_delta = self.hidden_error *
self.sigmoid_derivative(self.hidden_output)


        # Update weights and biases
        self.weights_hidden_output += self.hidden_output.T.dot(self.output_delta)
* learning_rate
        self.bias_hidden_output += np.sum(self.output_delta, axis=0,
keepdims=True) * learning_rate
        self.weights_input_hidden += X.T.dot(self.hidden_delta) * learning_rate
        self.bias_input_hidden += np.sum(self.hidden_delta, axis=0,
keepdims=True) * learning_rate
```

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

if __name__ == "__main__":
    # Example usage
    X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Input
    y = np.array([[0], [1], [1], [0]])              # Output

    # Initialize neural network
    input_size = 2
    hidden_size = 4
    output_size = 1
    neural_network = NeuralNetwork(input_size, hidden_size, output_size)
    # Train the neural network
    epochs = 10000
    learning_rate = 0.1
    neural_network.train(X, y, epochs, learning_rate)

    # Test the trained network
    print("Final predictions:")
    print(neural_network.forward(X))
```

OUTPUT:

```
Epoch 0, Loss: 0.2779
Epoch 1000, Loss: 0.2288
Epoch 2000, Loss: 0.1187
Epoch 3000, Loss: 0.0268
Epoch 4000, Loss: 0.0113
Epoch 5000, Loss: 0.0067
Epoch 6000, Loss: 0.0047
Epoch 7000, Loss: 0.0035
Epoch 8000, Loss: 0.0028
Epoch 9000, Loss: 0.0023
Final predictions:
[[0.0270804 ]
 [0.95624716]
 [0.95134667]
 [0.05428041]]
```

RESULT:

Hence The Neural Network model effectively learned complex patterns in the data for accurate predictions.