MLA02-Machine Learning Lab Program  
  
1.Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples

**Code:**

import numpy as np

def find\_s(training\_data, target):

num\_features = len(training\_data[0])

hypothesis = [None] \* num\_features

for i, example in enumerate(training\_data):

if target[i] == "Yes":

if hypothesis[0] is None:

hypothesis = example.copy()

else:

for j in range(num\_features):

if hypothesis[j] != example[j]:

hypothesis[j] = "?"

return hypothesis

training\_data = [

["Sunny", "Warm", "Normal", "Strong", "Warm", "Same"],

["Sunny", "Warm", "High", "Strong", "Warm", "Same"],

["Rainy", "Cold", "High", "Strong", "Warm", "Change"],

["Sunny", "Warm", "High", "Strong", "Cool", "Change"]

]

target = ["Yes", "Yes", "No", "Yes"]

hypothesis = find\_s(training\_data, target)

print("Most Specific Hypothesis:", hypothesis)

**Output:**

****

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  
  
**Code:**

import pandas as pd

import csv

def create\_sample\_training\_data():

    with open('training\_data.csv', 'w', newline='') as file:

        writer = csv.writer(file)

        writer.writerow(['Attribute1', 'Attribute2', 'Attribute3', 'Target'])

        writer.writerow(['yes', 'yes', 'yes', 'yes'])

        writer.writerow(['yes', 'yes', 'no', 'yes'])

        writer.writerow(['no', 'yes', 'yes', 'no'])

        writer.writerow(['no', 'no', 'no', 'no'])

def load\_training\_data(csv\_file):

    try:

        data = pd.read\_csv(csv\_file)

        return data

    except Exception as e:

        print(f"Error loading training data: {e}")

        return None

def candidate\_elimination(data):

    try:

        general\_hypothesis = ['?' for \_ in range(len(data.columns) - 1)]

        specific\_hypothesis = ['?' for \_ in range(len(data.columns) - 1)]

        for index, row in data.iterrows():

            if row['Target'] == 'yes':

                for i in range(len(general\_hypothesis)):

                    if general\_hypothesis[i] != '?' and general\_hypothesis[i] != row[data.columns[i]]:

                        general\_hypothesis[i] = '?'

                for i in range(len(specific\_hypothesis)):

                    if specific\_hypothesis[i] == '?' or specific\_hypothesis[i] == row[data.columns[i]]:

                        specific\_hypothesis[i] = row[data.columns[i]]

            else:

                for i in range(len(general\_hypothesis)):

                    if general\_hypothesis[i] != '?' and general\_hypothesis[i] == row[data.columns[i]]:

                        general\_hypothesis[i] = '?'

                for i in range(len(specific\_hypothesis)):

                    if specific\_hypothesis[i] != '?' and specific\_hypothesis[i] != row[data.columns[i]]:

                        specific\_hypothesis[i] = '?'

        return general\_hypothesis, specific\_hypothesis

    except Exception as e:

        print(f"Error running Candidate-Elimination algorithm: {e}")

        return None, None

def main():

    try:

        pd.read\_csv('training\_data.csv')

    except FileNotFoundError:

        create\_sample\_training\_data()

    data = load\_training\_data('training\_data.csv')

    if data is not None:

        general\_hypothesis, specific\_hypothesis = candidate\_elimination(data)

        if general\_hypothesis is not None and specific\_hypothesis is not None:

            print("General Hypothesis:", general\_hypothesis)

            print("Specific Hypothesis:", specific\_hypothesis)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output:**



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.

**Code:**

import pandas as pd

from collections import Counter

def generate\_data():

data = {

'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast', 'Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast', 'Overcast'],

'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot'],

'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal', 'Normal', 'Normal', 'High'],

'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Weak', 'Strong'],

'Play': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes']

}

return pd.DataFrame(data)

def calculate\_entropy(data):

target\_values = data.iloc[:, -1]

entropy = 0

for value in set(target\_values):

probability = len(target\_values[target\_values == value]) / len(target\_values)

entropy -= probability \* (probability \* 1)

return entropy

def calculate\_information\_gain(data, feature):

feature\_values = data[feature]

information\_gain = calculate\_entropy(data)

for value in set(feature\_values):

subset = data[feature\_values == value]

probability = len(subset) / len(data)

information\_gain -= probability \* calculate\_entropy(subset)

return information\_gain

def build\_decision\_tree(data, features):

if len(set(data.iloc[:, -1])) == 1:

return data.iloc[0, -1]

elif len(features) == 0:

return Counter(data.iloc[:, -1]).most\_common(1)[0][0]

else:

best\_feature = max(features, key=lambda feature: calculate\_information\_gain(data, feature))

tree = {best\_feature: {}}

for value in set(data[best\_feature]):

subset = data[data[best\_feature] == value]

subset\_features = [feature for feature in features if feature != best\_feature]

tree[best\_feature][value] = build\_decision\_tree(subset, subset\_features)

return tree

def classify\_sample(tree, sample):

if isinstance(tree, dict):

feature = list(tree.keys())[0]

value = sample[feature]

if value in tree[feature]:

return classify\_sample(tree[feature][value], sample)

else:

return None

else:

return tree

def main():

data = generate\_data()

print("Data Set:")

print(data)

features = list(data.columns)[:-1]

tree = build\_decision\_tree(data, features)

print("\nDecision Tree:")

print(tree)

sample = {'Outlook': 'Sunny', 'Temperature': 'Hot', 'Humidity': 'High', 'Wind': 'Weak'}

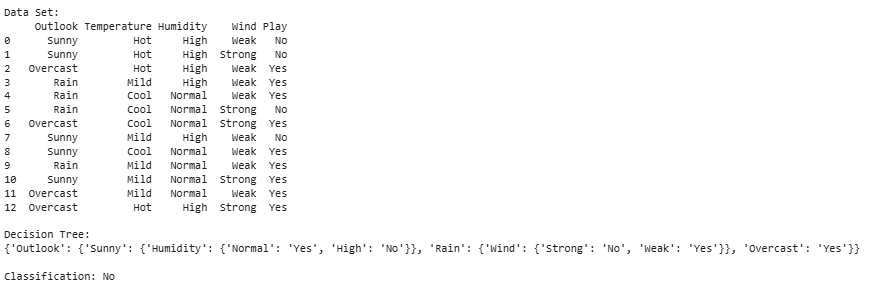
classification = classify\_sample(tree, sample)

print("\nClassification:", classification)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Output:**



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

**Code:**

import numpy as np

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

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.datasets import load\_iris

def sigmoid(x):

x = np.clip(x, -500, 500)

return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(x):

return x \* (1 - x)

class NeuralNetwork:

def \_\_init\_\_(self, input\_size, hidden\_size, output\_size, learning\_rate=0.01):

self.input\_size = input\_size

self.hidden\_size = hidden\_size

self.output\_size = output\_size

self.learning\_rate = learning\_rate

np.random.seed(42)

self.W1 = np.random.uniform(-1, 1, (self.input\_size, self.hidden\_size))

self.b1 = np.zeros((1, self.hidden\_size))

self.W2 = np.random.uniform(-1, 1, (self.hidden\_size, self.output\_size))

self.b2 = np.zeros((1, self.output\_size))

def forward(self, X):

self.z1 = np.dot(X, self.W1) + self.b1

self.a1 = sigmoid(self.z1)

self.z2 = np.dot(self.a1, self.W2) + self.b2

self.a2 = sigmoid(self.z2)

return self.a2

def backward(self, X, y, output):

m = X.shape[0]

output\_error = y - output

output\_delta = output\_error \* sigmoid\_derivative(output)

hidden\_error = output\_delta.dot(self.W2.T)

hidden\_delta = hidden\_error \* sigmoid\_derivative(self.a1)

self.W2 += self.a1.T.dot(output\_delta) \* self.learning\_rate / m

self.b2 += np.sum(output\_delta, axis=0, keepdims=True) \* self.learning\_rate / m

self.W1 += X.T.dot(hidden\_delta) \* self.learning\_rate / m

self.b1 += np.sum(hidden\_delta, axis=0, keepdims=True) \* self.learning\_rate / m

def train(self, X, y, epochs=10000):

for epoch in range(epochs):

output = self.forward(X)

self.backward(X, y, output)

if epoch % 1000 == 0:

loss = np.mean(np.square(y - output))

print(f'Epoch {epoch}, Loss: {loss:.4f}')

def predict(self, X):

output = self.forward(X)

return np.argmax(output, axis=1)

data = load\_iris()

X = data.data

y = data.target.reshape(-1, 1)

encoder = OneHotEncoder(sparse\_output=False)

y\_encoded = encoder.fit\_transform(y)

scaler = StandardScaler()

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

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

nn = NeuralNetwork(input\_size=4, hidden\_size=5, output\_size=3, learning\_rate=0.1)

nn.train(X\_train, y\_train, epochs=10000)

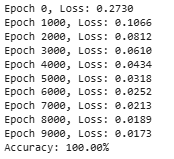
y\_pred = nn.predict(X\_test)

y\_test\_labels = np.argmax(y\_test, axis=1)

accuracy = np.mean(y\_pred == y\_test\_labels)

print(f'Accuracy: {accuracy \* 100:.2f}%')

**Output:**

****

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

**Code:**

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

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn import metrics

import pandas as pd

import numpy as np

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']

dataset = pd.read\_csv(url, names=names)

X = dataset[['sepal-length', 'sepal-width', 'petal-length', 'petal-width']]

y = dataset['class']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1)

sc = StandardScaler()

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

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

knn = KNeighborsClassifier(n\_neighbors=5)

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

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

print("Accuracy:", metrics.accuracy\_score(y\_test, y\_pred))

new\_sample = [[5.1, 3.5, 1.4, 0.2]]

new\_sample = sc.transform(new\_sample)

prediction = knn.predict(new\_sample)

print("Prediction:", prediction)

**Output:**



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

**Code:**

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

from sklearn.naive\_bayes import GaussianNB

from sklearn import metrics

import pandas as pd

import numpy as np

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']

dataset = pd.read\_csv(url, names=names)

dataset['class'] = dataset['class'].map({'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2})

X = dataset[['sepal-length', 'sepal-width', 'petal-length', 'petal-width']]

y = dataset['class']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1)

gnb = GaussianNB()

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

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

print("Accuracy:", metrics.accuracy\_score(y\_test, y\_pred))

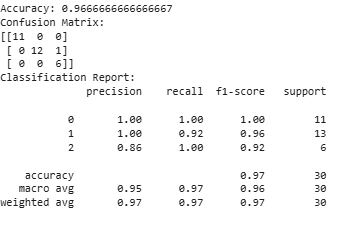
print("Confusion Matrix:")

print(metrics.confusion\_matrix(y\_test, y\_pred))

print("Classification Report:")

print(metrics.classification\_report(y\_test, y\_pred))

**Output:**



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

**Code:**

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, confusion\_matrix, classification\_report

from sklearn.datasets import load\_breast\_cancer

data = load\_breast\_cancer()

X = data.data

y = data.target

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

scaler = StandardScaler()

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

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

model = LogisticRegression()

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

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

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

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

class\_report = classification\_report(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.4f}")

print("Confusion Matrix:\n", conf\_matrix)

print("Classification Report:\n", class\_report)

plt.figure(figsize=(5,4))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=data.target\_names, yticklabels=data.target\_names)

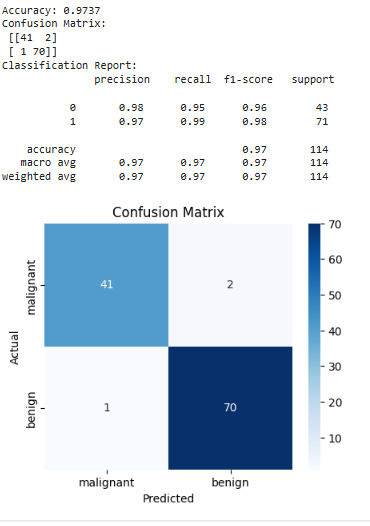
plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

**Output:**

****

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

**Code:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

np.random.seed(42)

X = 2 \* np.random.rand(100, 1)

y = 4 + 3 \* X + np.random.randn(100, 1)

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

model = LinearRegression()

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

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

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f"Mean Squared Error: {mse:.4f}")

print(f"R² Score: {r2:.4f}")

print(f"Intercept: {model.intercept\_[0]:.4f}")

print(f"Coefficient: {model.coef\_[0][0]:.4f}")

plt.scatter(X\_test, y\_test, color="blue", label="Actual Data")

plt.plot(X\_test, y\_pred, color="red", linewidth=2, label="Regression Line")

plt.xlabel("Independent Variable (X)")

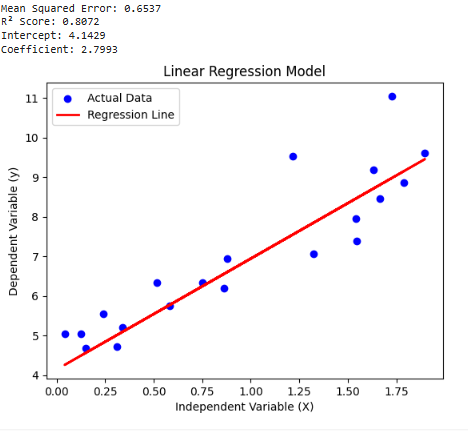
plt.ylabel("Dependent Variable (y)")

plt.title("Linear Regression Model")

plt.legend()

plt.show()

**Output:**



9. Compare Linear and Polynomial Regression using Python

**Code:**

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

np.random.seed(42)

X = 6 \* np.random.rand(100, 1) - 3

y = 0.5 \* X\*\*2 + X + 2 + np.random.randn(100, 1)

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

lin\_reg = LinearRegression()

lin\_reg.fit(X\_train, y\_train)

y\_pred\_linear = lin\_reg.predict(X\_test)

poly\_features = PolynomialFeatures(degree=2)

X\_train\_poly = poly\_features.fit\_transform(X\_train)

X\_test\_poly = poly\_features.transform(X\_test)

poly\_reg = LinearRegression()

poly\_reg.fit(X\_train\_poly, y\_train)

y\_pred\_poly = poly\_reg.predict(X\_test\_poly)

mse\_linear = mean\_squared\_error(y\_test, y\_pred\_linear)

r2\_linear = r2\_score(y\_test, y\_pred\_linear)

mse\_poly = mean\_squared\_error(y\_test, y\_pred\_poly)

r2\_poly = r2\_score(y\_test, y\_pred\_poly)

print(f"Linear Regression MSE: {mse\_linear:.4f}, R² Score: {r2\_linear:.4f}")

print(f"Polynomial Regression MSE: {mse\_poly:.4f}, R² Score: {r2\_poly:.4f}")

X\_range = np.linspace(-3, 3, 100).reshape(-1, 1)

y\_linear\_pred = lin\_reg.predict(X\_range)

y\_poly\_pred = poly\_reg.predict(poly\_features.transform(X\_range))

plt.scatter(X, y, color="blue", label="Actual Data", alpha=0.6)

plt.plot(X\_range, y\_linear\_pred, color="red", label="Linear Regression", linewidth=2)

plt.plot(X\_range, y\_poly\_pred, color="green", label="Polynomial Regression", linewidth=2)

plt.xlabel("Feature X")

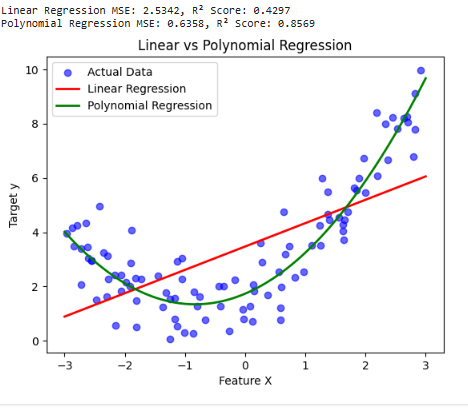
plt.ylabel("Target y")

plt.title("Linear vs Polynomial Regression")

plt.legend()

plt.show()

**Output:**



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

**Code:**

import numpy as np

import matplotlib.pyplot as plt

from scipy.stats import multivariate\_normal

from sklearn.mixture import GaussianMixture

np.random.seed(42)

X1 = np.random.normal(2, 1, (150, 1))

X2 = np.random.normal(-2, 1, (150, 1))

X = np.vstack((X1, X2))

k = 2

means = np.random.choice(X.flatten(), k).reshape(k, 1)

covariances = [np.array([[np.var(X)]]) + 1e-6 for \_ in range(k)]

weights = np.full(k, 1/k)

responsibilities = np.zeros((X.shape[0], k))

for \_ in range(100):

    responsibilities.fill(0)

    for i in range(k):

        responsibilities[:, i] = weights[i] \* multivariate\_normal.pdf(X[:,0], mean=means[i][0], cov=covariances[i])

    responsibilities /= np.clip(responsibilities.sum(axis=1)[:, None], a\_min=1e-10, a\_max=None)

    Nk = responsibilities.sum(axis=0)

    weights[:] = Nk / X.shape[0]

    means[:] = (responsibilities.T @ X) / Nk[:, None]

    for i in range(k):

        centered\_X\_i\_mean\_diffs\_squared\_summed\_over\_Ni\_divided\_by\_Ni\_plus\_epsilon \

            =(responsibilities[:, i][: ,None]\*(X - means[i])\*\*2).sum(axis=0)/Nk[i]+ 5e-7

        covariances[i][:,:] += centered\_X\_i\_mean\_diffs\_squared\_summed\_over\_Ni\_divided\_by\_Ni\_plus\_epsilon

gmm\_model=GaussianMixture(n\_components=k,covariance\_type='full',random\_state=np.random.randint(low=42))

gmm\_model.fit(X)

gmm\_preds=gmm\_model.predict(X)

plt.scatter(X,gmm\_preds,cmap='coolwarm',c=gmm\_preds,label="Clusters")

plt.scatter(means,np.zeros\_like(means),marker='x',color='black',s=100,label="Final Means")

plt.title("Expectation-Maximization(GMM)")

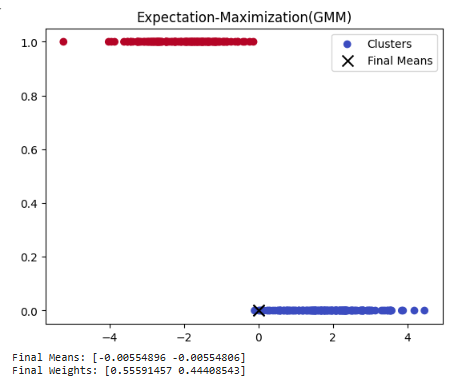
plt.legend()

plt.show()

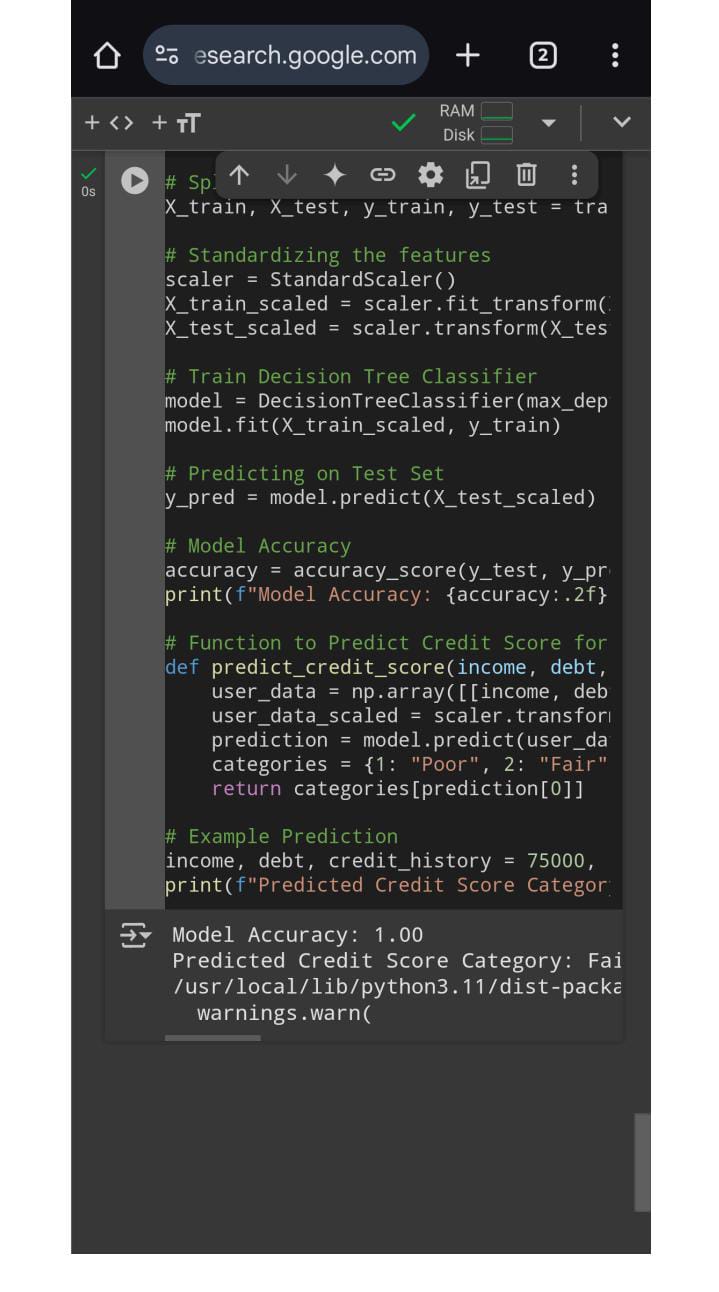
print("Final Means:", means.flatten())

print("Final Weights:", weights)

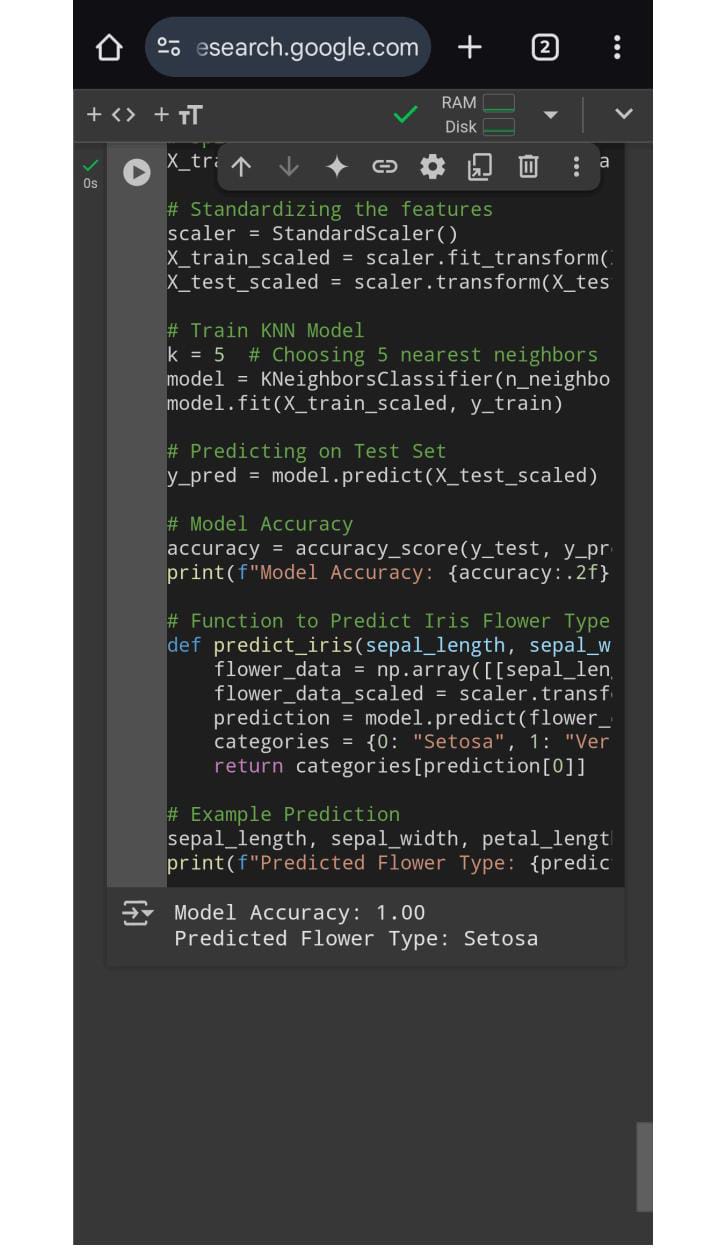
**Output:**

****

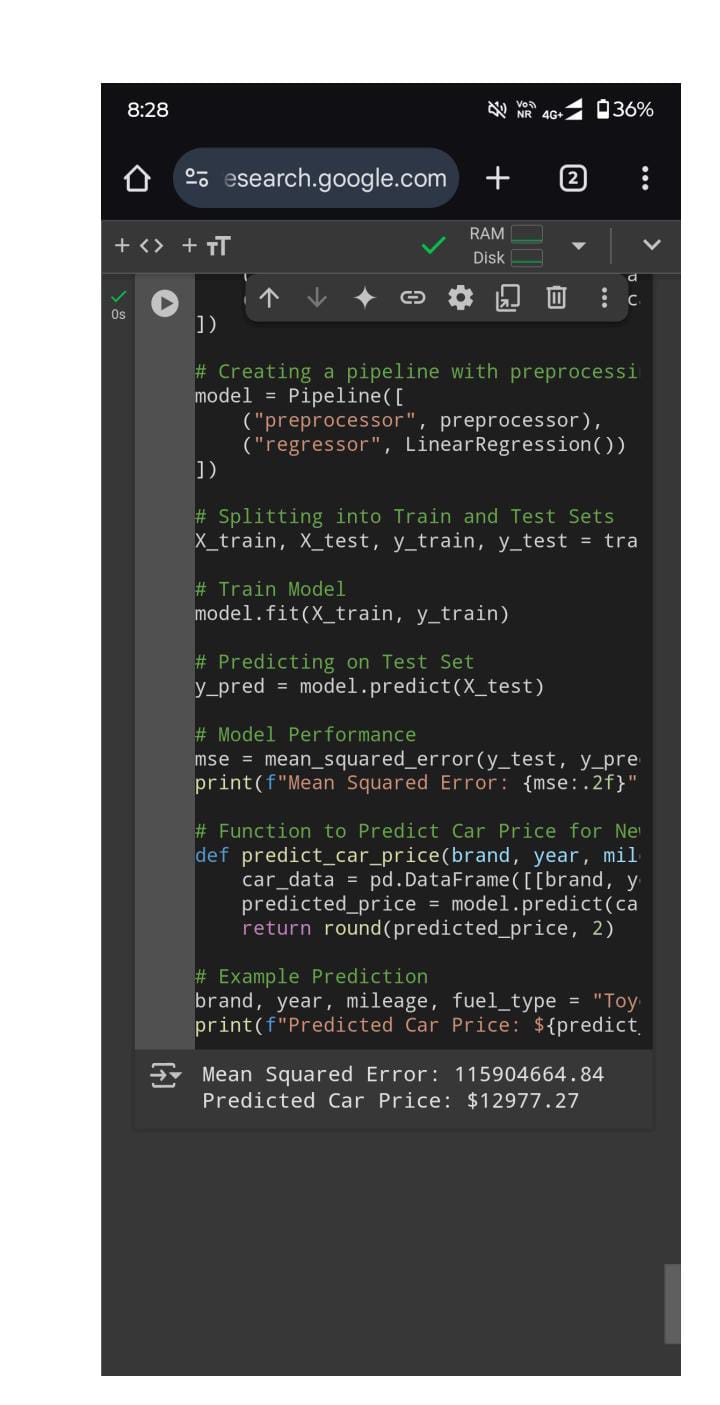
11 Write a program for the task of Credit Score Classification



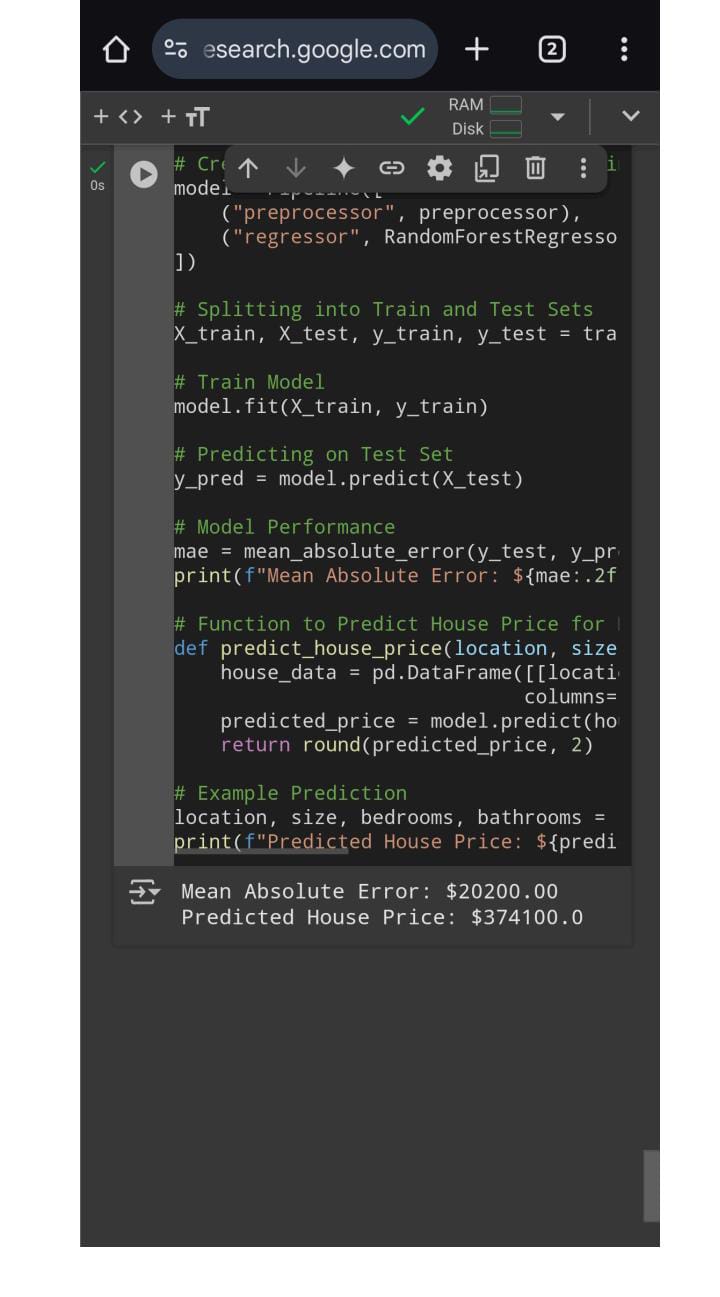
12 Implement Iris Flower Classification using KNN



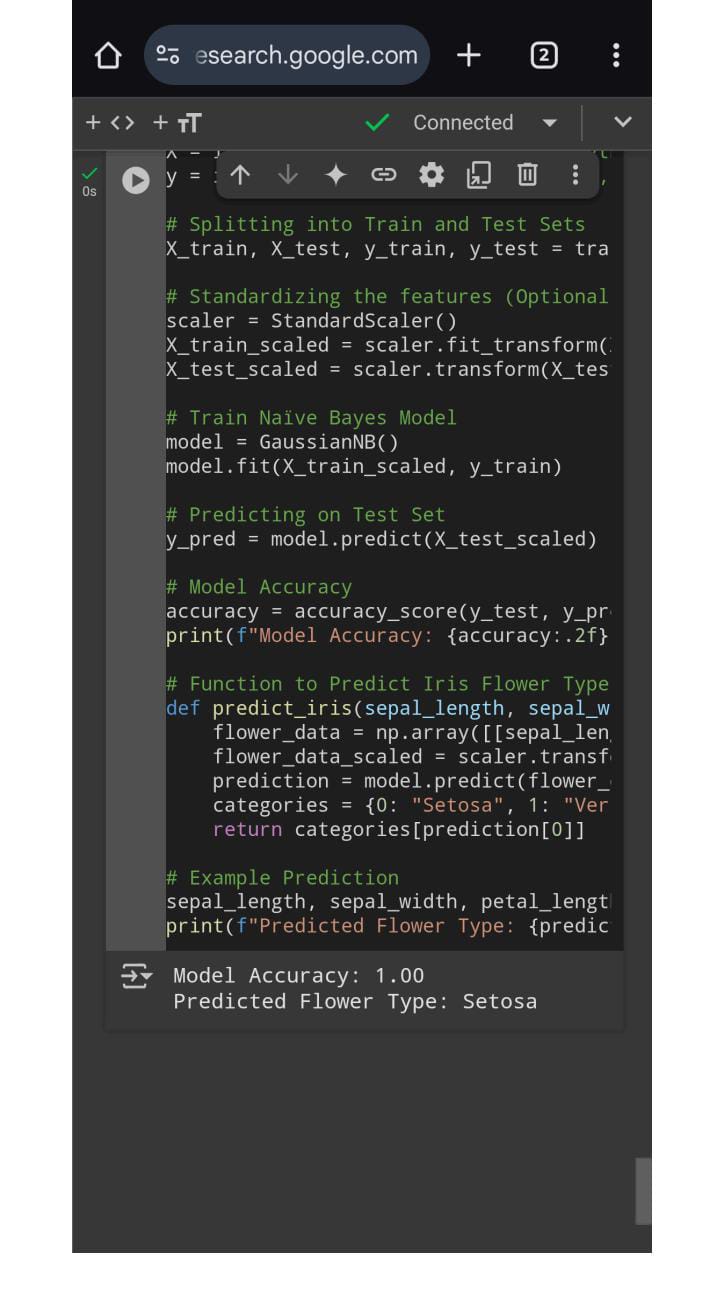
13 Implement the Car Price Prediction Model using Python



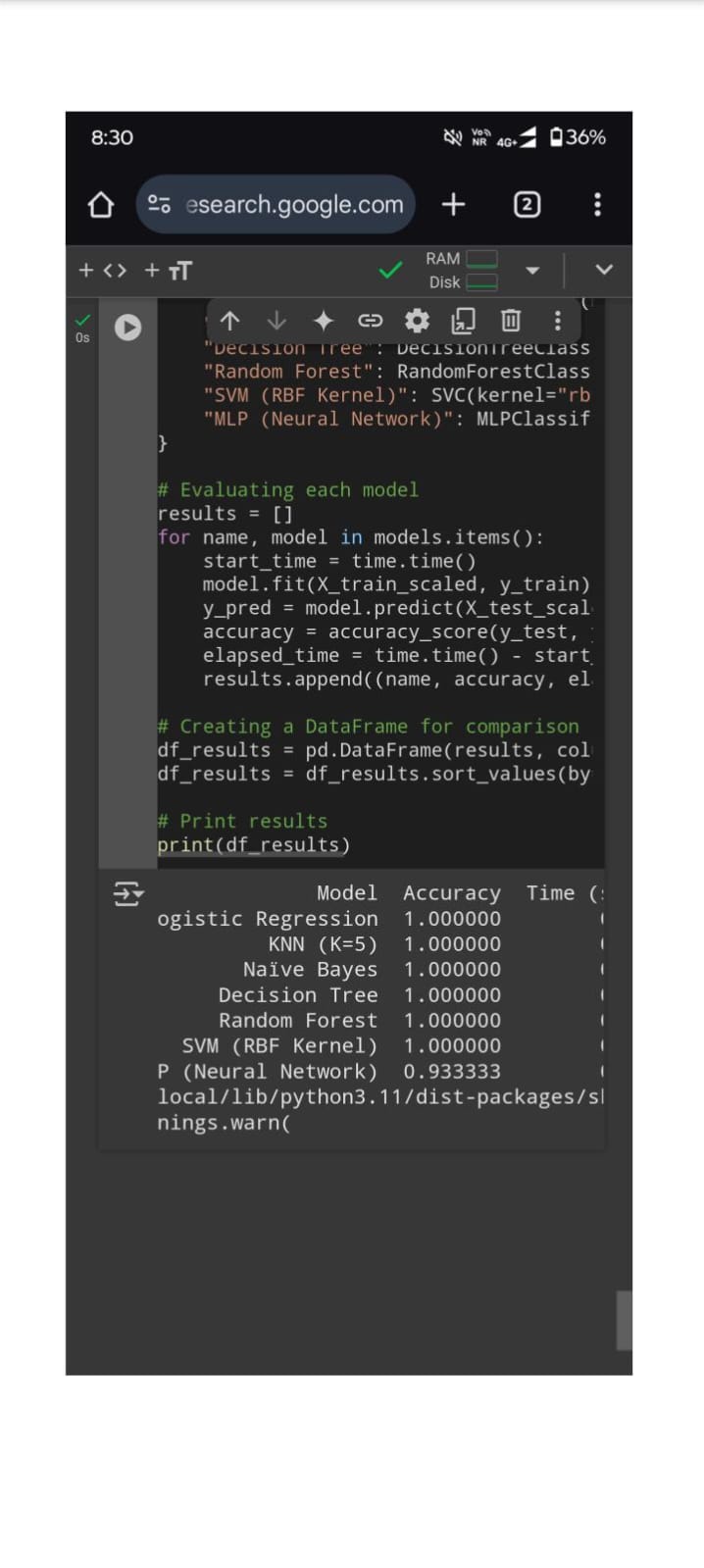
14 Implement House price Prediction using appropriate machine learning algorithm



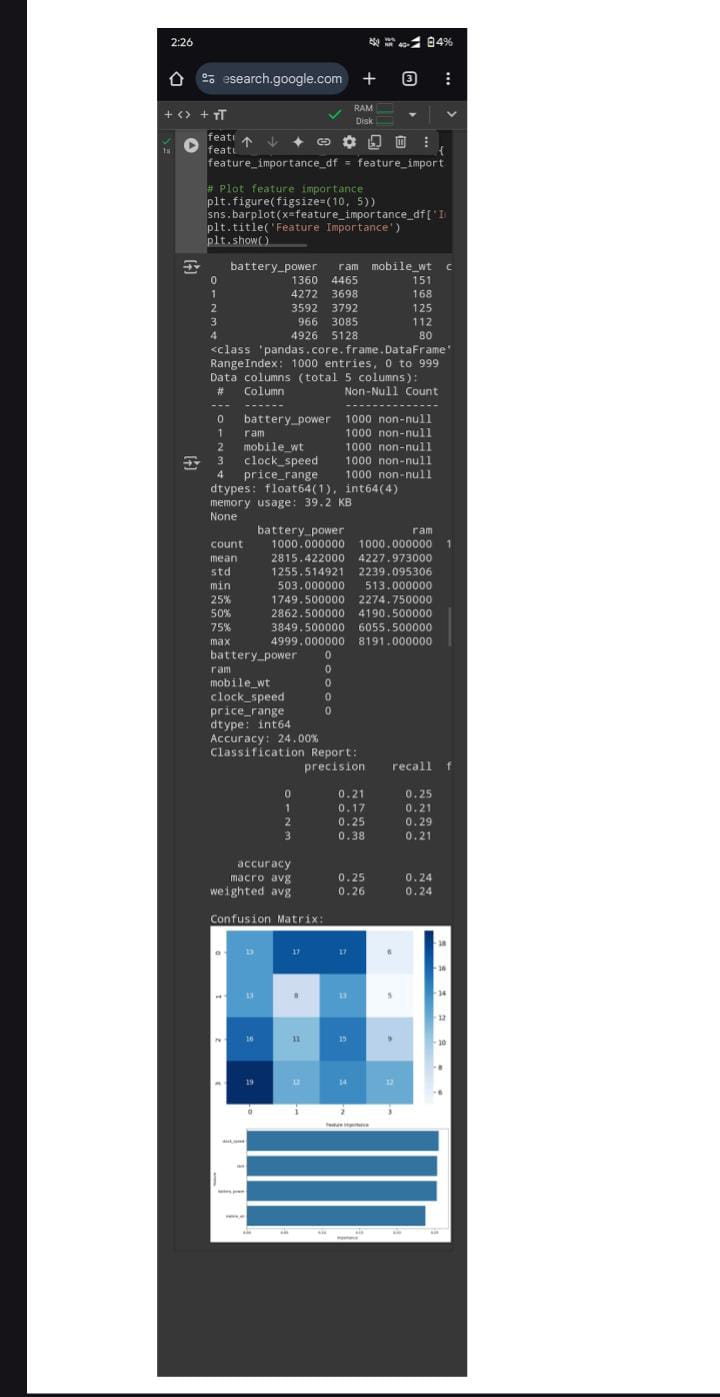
15 Implement Iris Flower Classification using Naive Bayes classifier



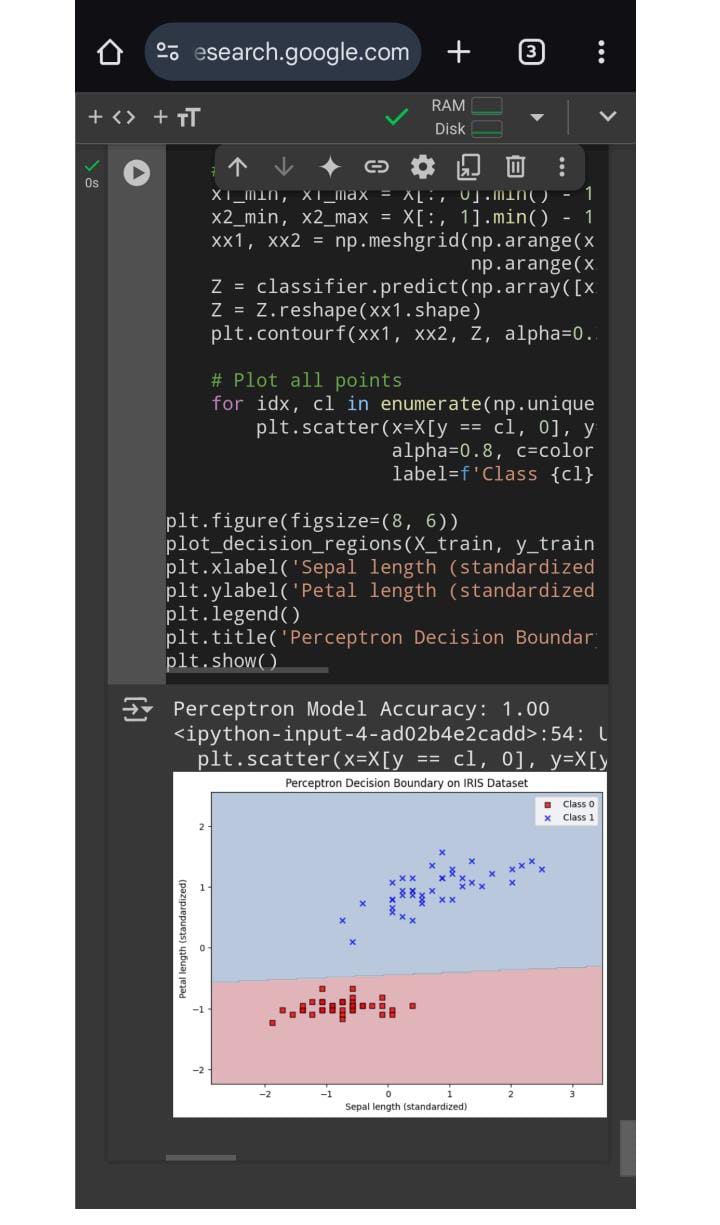
16 Compare different types Classification Algorithms and evaluate their performance.



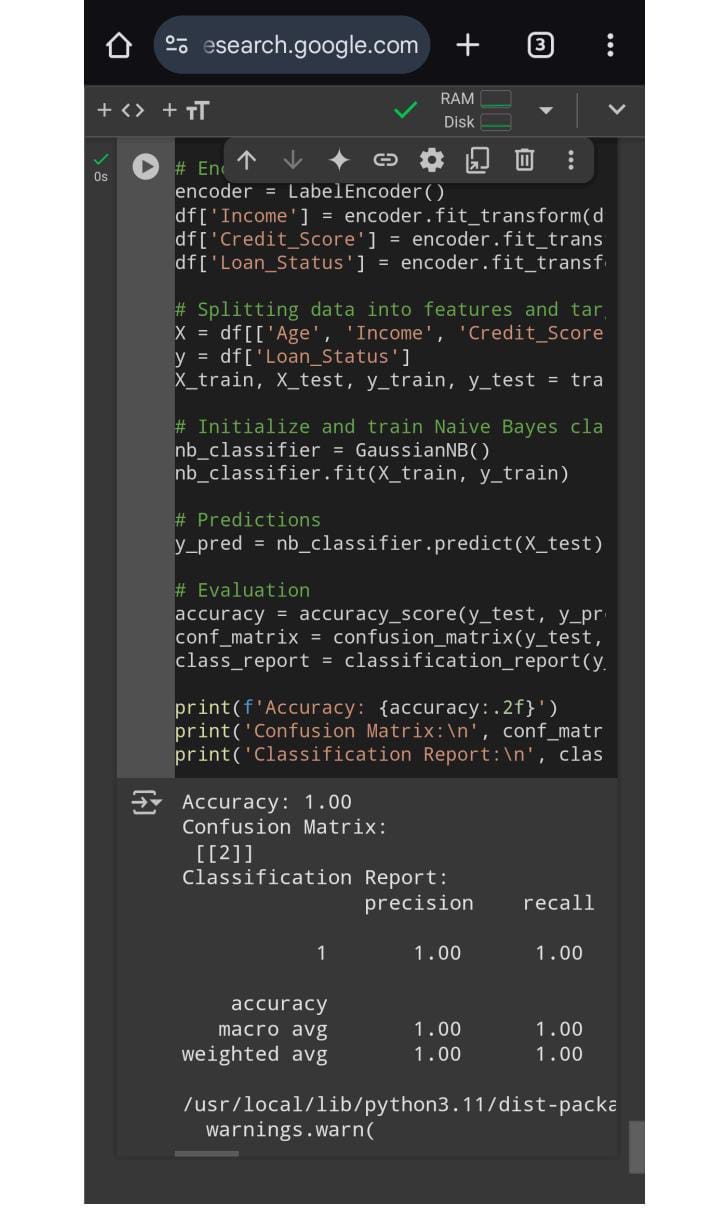
17 Implement Mobile Price Prediction using appropriate machine learning algorithm



18 Implement Perceptron based IRIS classification



19 Implementation of Naive Bayes classification for Bank Loan prediction



20 Implement Future Sales Prediction using a suitable machine learning algorithm

