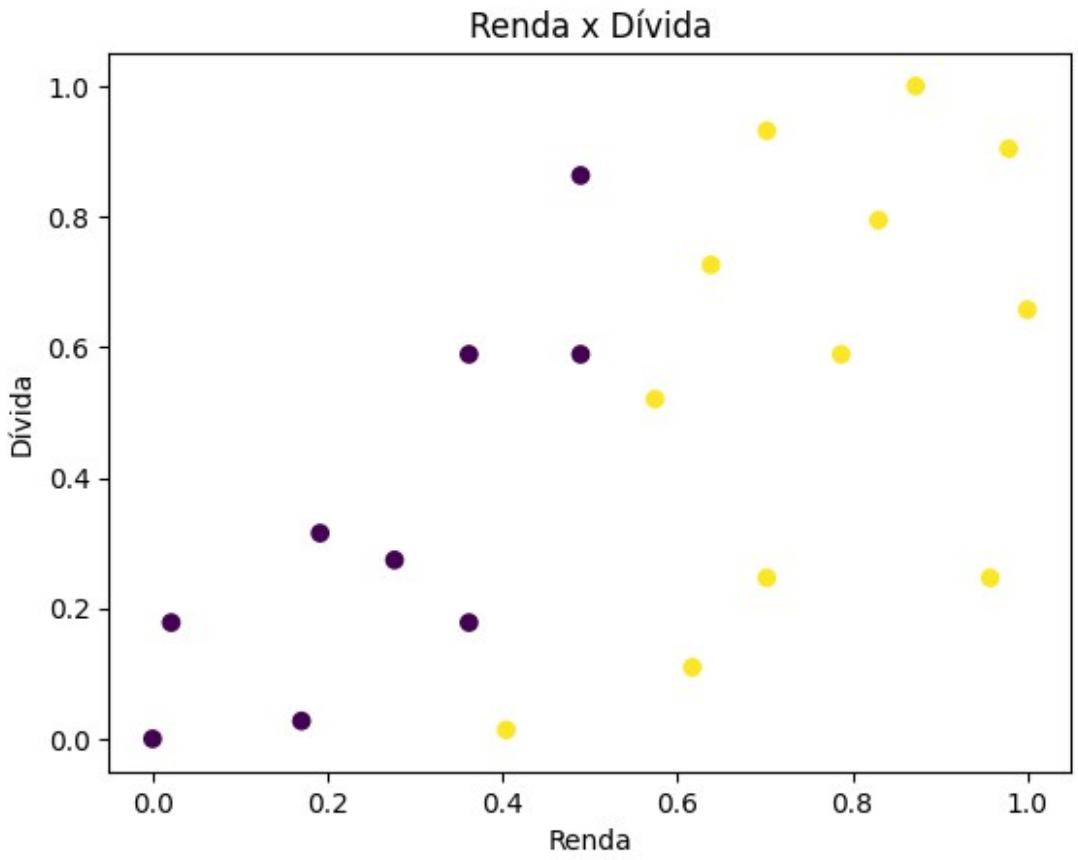
```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from sklearn import metrics
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from google.colab import drive

class DatasetLoader:
    @staticmethod
    def load_dataset(file_path):
        return pd.read_csv(file_path)

class Preprocessor:
    @staticmethod
    def normalize_data(X):
        scaler = preprocessing.MinMaxScaler()
        return scaler.fit_transform(X)

class ModelTrainer:
    @staticmethod
    def train_model(X_train, y_train):
        perceptron = Perceptron(max_iter=100, alpha=0.01,
n_iter_no_change=10)
        perceptron.fit(X_train, y_train)
        return perceptron

class ModelEvaluator:
    @staticmethod
    def evaluate_model(model, X_test, y_test):
        predictions_test = model.predict(X_test)
        test_score = accuracy_score(predictions_test, y_test)
        return test_score, predictions_test

class IndividualTester:
    @staticmethod
    def test_individual(model, maxXRenda, maxXDivida, renda, divida):
        renda_norm = renda / maxXRenda
        divida_norm = divida / maxXDivida
        sample = np.array([renda_norm, divida_norm])
        return model.predict([sample])

class ResultDisplayer:
    @staticmethod
    def display_accuracy(train_score, test_score):
        print("Acurácia com dados de treinamento: ", train_score)
        print("Acurácia com dados de teste: ", test_score)
```

```
    @staticmethod
    def display_classification_report(predictions_test, y_test):
        print(classification_report(predictions_test, y_test))

    @staticmethod
    def display_confusion_matrix(y_test, predictions_test):
        conf_matrix = confusion_matrix(y_test, predictions_test)
        cm_display =
metrics.ConfusionMatrixDisplay(confusion_matrix=conf_matrix,
display_labels=['mau', 'bom'])
        cm_display.plot()
        plt.show()

    @staticmethod
    def display_individual_accuracy(score_A, score_B):
        print("Acurácia com dados de A: ", score_A)
        print("Acurácia com dados de B: ", score_B)


# Carregando o dataset
df = DatasetLoader.load_dataset("bancario.csv")
print(df.head())
```

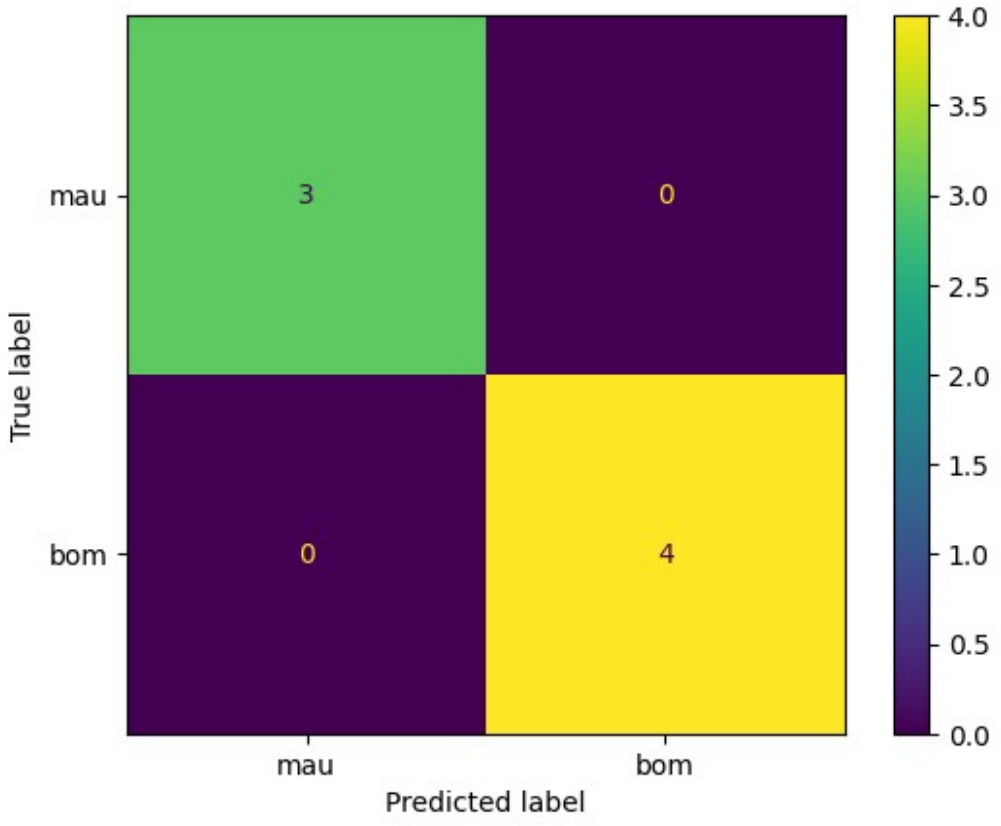
```
   Conta  Renda  Dívida Classe
0    101   2800     550    bom
1    102   1300     500    mau
2    103   1400      80    bom
3    104    500     200    mau
4    105   1100     270    mau
```

```
# Separando os dados
y = np.where(df['Classe'] == 'mau', -1, 1)
X = df[['Renda', 'Dívida']].values

# Normalizando os dados
X = Preprocessor.normalize_data(X)

# Plotando o gráfico
plt.scatter(X[:,0], X[:,1], c=y)
plt.title("Renda x Dívida" )
plt.xlabel('Renda')
plt.ylabel('Dívida')
plt.show()
```




```
# Separando os dados para treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.30, random_state=12)
# Treinando o modelo
model = ModelTrainer.train_model(X_train, y_train)

# Avaliando o modelo
test_score, predictions_test = ModelEvaluator.evaluate_model(model,
X_test, y_test)
train_score = accuracy_score(model.predict(X_train), y_train)
ResultDisplayer.display_accuracy(train_score, test_score)
ResultDisplayer.display_classification_report(predictions_test,
y_test)
ResultDisplayer.display_confusion_matrix(y_test, predictions_test)

Acurácia com dados de treinamento:  1.0
Acurácia com dados de teste:  1.0
              precision    recall  f1-score   support

          -1       1.00      1.00      1.00         3
           1       1.00      1.00      1.00         4

    accuracy                           1.00         7
```

```
   macro avg       1.00      1.00      1.00         7
weighted avg       1.00      1.00      1.00         7
```




```
# Testando amostras individuais
rendaA = 9000
dividaA = 100
rendaB = 5000
dividaB = 600
maxXRenda = X[:,0].max()
maxXDivida = X[:,1].max()

prediction_A = IndividualTester.test_individual(model, maxXRenda,
maxXDivida, rendaA, dividaA)
prediction_B = IndividualTester.test_individual(model, maxXRenda,
maxXDivida, rendaB, dividaB)
score_A = accuracy_score(prediction_A, [1])
score_B = accuracy_score(prediction_B, [-1])
ResultDisplayer.display_individual_accuracy(score_A, score_B)
```

```
Acurácia com dados de A:  1.0
Acurácia com dados de B:  0.0
```