**NFT ASSIGNMENT-1 CODE**

import pandas as pd

import numpy as np

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

from sklearn.preprocessing import LabelEncoder, StandardScaler

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

# Load the dataset

df = pd.read\_csv("Loan Prediction Dataset.csv")

# Data preprocessing

# Handle missing values

df['LoanAmount'] = df['LoanAmount'].fillna(df['LoanAmount'].mean())

df['Loan\_Amount\_Term'] = df['Loan\_Amount\_Term'].fillna(df['Loan\_Amount\_Term'].mean())

df['Credit\_History'] = df['Credit\_History'].fillna(df['Credit\_History'].mean())

df['Gender'] = df['Gender'].fillna(df['Gender'].mode()[0])

df['Married'] = df['Married'].fillna(df['Married'].mode()[0])

df['Dependents'] = df['Dependents'].fillna(df['Dependents'].mode()[0])

df['Self\_Employed'] = df['Self\_Employed'].fillna(df['Self\_Employed'].mode()[0])

# Log-transform numerical features

df['ApplicantIncome'] = np.log(df['ApplicantIncome'])

df['CoapplicantIncome'] = np.log(df['CoapplicantIncome'] + 1)  # Adding 1 to avoid log(0)

df['LoanAmount'] = np.log(df['LoanAmount'])

df['Loan\_Amount\_Term'] = np.log(df['Loan\_Amount\_Term'] + 1)  # Adding 1 to avoid log(0)

# Encode categorical variables

encoder = LabelEncoder()

categorical\_columns = ['Gender', 'Married', 'Dependents', 'Education', 'Self\_Employed', 'Property\_Area', 'Loan\_Status']

for col in categorical\_columns:

    df[col] = encoder.fit\_transform(df[col])

# Data Split

X = df.drop(columns=['Loan\_Status', 'Loan\_ID'], axis=1)

y = df['Loan\_Status']

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)

# Define the Multi-Layer Perceptron (MLP) architecture

class MLP:

    def \_\_init\_\_(self, input\_size):

        self.input\_size = input\_size

        self.hidden\_size = 128

        self.output\_size = 1

        self.weights\_input\_hidden = np.random.randn(self.input\_size, self.hidden\_size)

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

        self.weights\_hidden\_output = np.random.randn(self.hidden\_size, self.output\_size)

        self.bias\_output = np.zeros((1, self.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):

        self.hidden\_input = np.dot(X, self.weights\_input\_hidden) + self.bias\_hidden

        self.hidden\_output = self.sigmoid(self.hidden\_input)

        self.output = self.sigmoid(np.dot(self.hidden\_output, self.weights\_hidden\_output) + self.bias\_output)

        return self.output

    def backward(self, X, y, output, learning\_rate):

        self.output\_error = y - output

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

        self.hidden\_layer\_error = self.output\_delta.dot(self.weights\_hidden\_output.T)

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

        self.weights\_hidden\_output += self.hidden\_output.T.dot(self.output\_delta) \* learning\_rate

        self.weights\_input\_hidden += X.T.dot(self.hidden\_layer\_delta) \* learning\_rate

        self.bias\_output += np.sum(self.output\_delta, axis=0, keepdims=True) \* learning\_rate

        self.bias\_hidden += np.sum(self.hidden\_layer\_delta, axis=0) \* learning\_rate

    def train(self, X, y, epochs, learning\_rate):

        for \_ in range(epochs):

            output = self.forward(X)

            self.backward(X, y.values.reshape(-1, 1), output, learning\_rate)

# Initialize and train the MLP

input\_size = X\_train.shape[1]

mlp = MLP(input\_size)

mlp.train(X\_train, y\_train, epochs=50, learning\_rate=0.01)

# Predict on test data

y\_pred = (mlp.forward(X\_test) > 0.5).astype(int)

# Calculate and print accuracy

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

print("Accuracy:", accuracy)

# Print classification report and confusion matrix

print(classification\_report(y\_test, y\_pred, zero\_division=0))

print("Confusion Matrix:")

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

Accuracy: 0.7642276422764228

precision recall f1-score support

0 0.79 0.44 0.57 43

1 0.76 0.94 0.84 80

accuracy 0.76 123

macro avg 0.77 0.69 0.70 123

weighted avg 0.77 0.76 0.74 123

Confusion Matrix:

[[19 24]

[ 5 75]]

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import confusion\_matrix, roc\_curve, roc\_auc\_score, auc

# Confusion Matrix

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

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

plt.show()

# ROC Curve and AUC

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred)

roc\_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

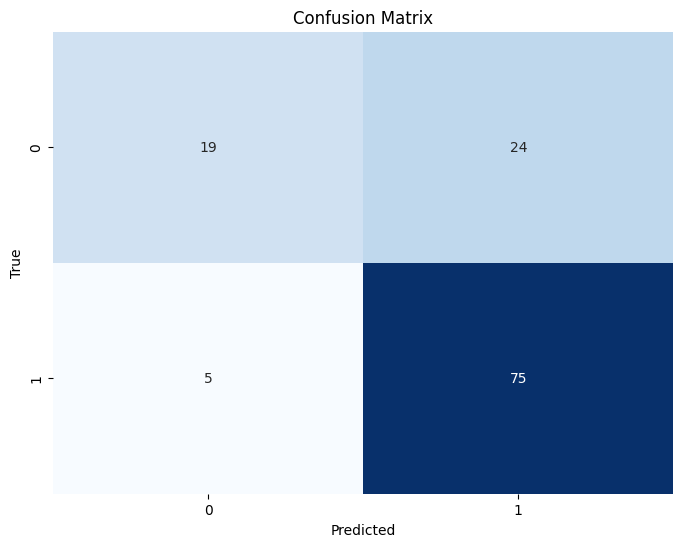
plt.xlabel('False Positive Rate')

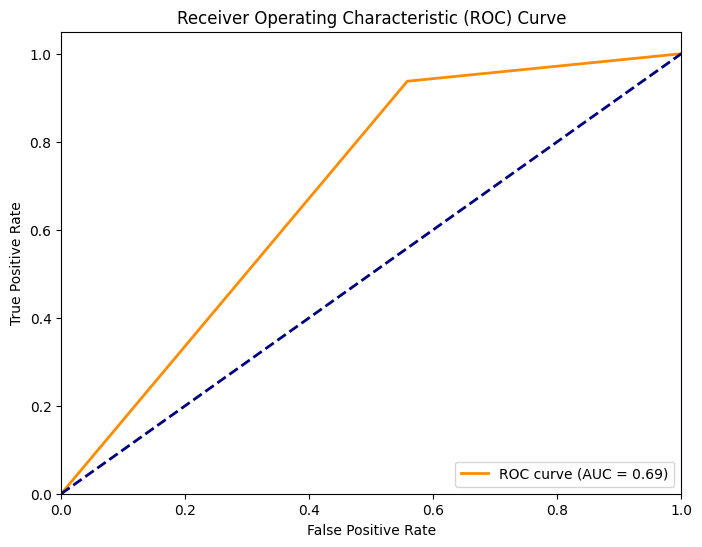
plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.legend(loc='lower right')

plt.show()





**DONE BY:**

**BT21CSE100-Katnapalli Rishitha**

**BT21CSE106-Anandapu Srihitha**