Ml model for credit risk assessment and predict loan approval

import pandas as pd

import numpy as np

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

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report

import seaborn as sns

import matplotlib.pyplot as plt

# --- Step 1: Load and check datasets ---

data1 = pd.read\_csv('D:/python wadaaaa/credit\_risk\_dataset.csv') # Dataset 1

data2 = pd.read\_csv('D:/python wadaaaa/credit\_risk\_data\_set.csv') # Dataset 2

# --- Step 2: Preprocessing ---

# Preprocess Dataset 1: Encode categorical features, fill missing values

label\_encoders = {}

categorical\_cols\_1 = ['person\_home\_ownership', 'loan\_intent', 'loan\_grade', 'cb\_person\_default\_on\_file']

for col in categorical\_cols\_1:

label\_encoders[col] = LabelEncoder()

data1[col] = label\_encoders[col].fit\_transform(data1[col])

# Avoid chained assignment warnings

data1['person\_emp\_length'] = data1['person\_emp\_length'].fillna(data1['person\_emp\_length'].median())

data1['loan\_int\_rate'] = data1['loan\_int\_rate'].fillna(data1['loan\_int\_rate'].median())

# Preprocess Dataset 2: Handle missing values, encode categorical variables if necessary

numeric\_cols\_2 = data2.select\_dtypes(include=[np.number]).columns

data2[numeric\_cols\_2] = data2[numeric\_cols\_2].fillna(data2[numeric\_cols\_2].mean())

categorical\_cols\_2 = data2.select\_dtypes(include=[object]).columns

data2[categorical\_cols\_2] = data2[categorical\_cols\_2].fillna('Unknown')

# Encode categorical columns in Dataset 2 (if any)

for col in categorical\_cols\_2:

label\_encoders[col] = LabelEncoder()

data2[col] = label\_encoders[col].fit\_transform(data2[col])

# Feature engineering: Example debt-to-income ratio for Dataset 1

data1['debt\_to\_income\_ratio'] = data1['loan\_amnt'] / data1['person\_income']

data1['loan\_amnt'] = data1['loan\_amnt'].clip(lower=0)

# --- Step 3: Define features and target variables ---

X1 = data1.drop(columns=['loan\_status'])

y1 = data1['loan\_status']

X2 = data2.drop(columns=['Risk'])

y2 = data2['Risk'].apply(lambda x: 1 if x == 'bad' else 0)

# --- Step 4: Split the dataset for training and testing ---

X1\_train, X1\_test, y1\_train, y1\_test = train\_test\_split(X1, y1, test\_size=0.3, random\_state=42)

# --- Step 5: Feature scaling ---

scaler = StandardScaler()

X1\_train = scaler.fit\_transform(X1\_train)

X1\_test = scaler.transform(X1\_test)

# --- Step 6: Train the RandomForest model for credit risk assessment ---

rf\_model\_1 = RandomForestClassifier(random\_state=42)

rf\_model\_1.fit(X1\_train, y1\_train)

# Predictions and evaluation for Dataset 1

y1\_pred = rf\_model\_1.predict(X1\_test)

print("Random Forest Classifier Report for Dataset 1:")

print(classification\_report(y1\_test, y1\_pred))

# --- Step 7: Credit Risk Assessment (Generating Risk Scores) ---

y1\_prob = rf\_model\_1.predict\_proba(X1\_test)[:, 1] # Probability of bad credit

# Create a DataFrame to store the results

assessment\_df = pd.DataFrame(X1\_test, columns=X1.columns)

assessment\_df['Risk\_Score'] = y1\_prob

# Categorize risk into 'Low', 'Medium', and 'High' based on thresholds

assessment\_df['Risk\_Category'] = pd.cut(assessment\_df['Risk\_Score'], bins=[0, 0.3, 0.7, 1], labels=['Low Risk', 'Medium Risk', 'High Risk'])

# Display the risk assessment report

print("\nCredit Risk Assessment Report:")

print(assessment\_df[['Risk\_Score', 'Risk\_Category']].head(10))

# Save the assessment results to a CSV file

assessment\_df.to\_csv('credit\_risk\_assessment\_report.csv', index=False)

print("\nRisk assessment report saved as 'credit\_risk\_assessment\_report.csv'.")

# --- Step 8: Train Loan Prediction Model ---

# Assuming that the loan approval depends on the risk score and other financial factors

# Create a new target variable 'Loan Approval' (1 for approved, 0 for not approved)

assessment\_df['Loan\_Approval'] = assessment\_df['Risk\_Score'].apply(lambda x: 1 if x < 0.5 else 0)

# Define features and target for loan prediction

loan\_X = assessment\_df.drop(columns=['Loan\_Approval', 'Risk\_Category'])

loan\_y = assessment\_df['Loan\_Approval']

# Split the data for loan prediction

loan\_X\_train, loan\_X\_test, loan\_y\_train, loan\_y\_test = train\_test\_split(loan\_X, loan\_y, test\_size=0.3, random\_state=42)

# Train a RandomForest model for loan approval prediction

loan\_model = RandomForestClassifier(random\_state=42)

loan\_model.fit(loan\_X\_train, loan\_y\_train)

# Predict and evaluate loan approval

loan\_y\_pred = loan\_model.predict(loan\_X\_test)

print("\nRandom Forest Classifier Report for Loan Prediction:")

print(classification\_report(loan\_y\_test, loan\_y\_pred))

# --- Step 9: Visualizations ---

# 1. Bar Chart: Distribution of Loan Approval

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

sns.countplot(x=loan\_y\_test, palette='coolwarm')

plt.title('Loan Approval Prediction Distribution')

plt.xlabel('Loan Approval (1=Approved, 0=Not Approved)')

plt.ylabel('Count')

plt.show()

# 2. Bar Chart: Distribution of Risk Categories

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

sns.countplot(x='Risk\_Category', data=assessment\_df, palette='viridis')

plt.title('Distribution of Risk Categories')

plt.xlabel('Risk Category')

plt.ylabel('Number of Individuals')

plt.show()

# 3. Histogram: Distribution of Risk Scores

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

sns.histplot(assessment\_df['Risk\_Score'], bins=20, kde=True, color='blue')

plt.title('Distribution of Risk Scores')

plt.xlabel('Risk Score')

plt.ylabel('Frequency')

plt.show()

# 4. Pie Chart: Proportion of Loan Approval

plt.figure(figsize=(7, 7))

loan\_approval\_counts = assessment\_df['Loan\_Approval'].value\_counts()

plt.pie(loan\_approval\_counts, labels=['Not Approved', 'Approved'], autopct='%1.1f%%', startangle=90, colors=['orange', 'green'])

plt.title('Proportion of Loan Approval')

plt.show()

# 5. Heatmap: Correlations Between Features

numeric\_columns = assessment\_df.select\_dtypes(include=[np.number]).columns

plt.figure(figsize=(12, 8))

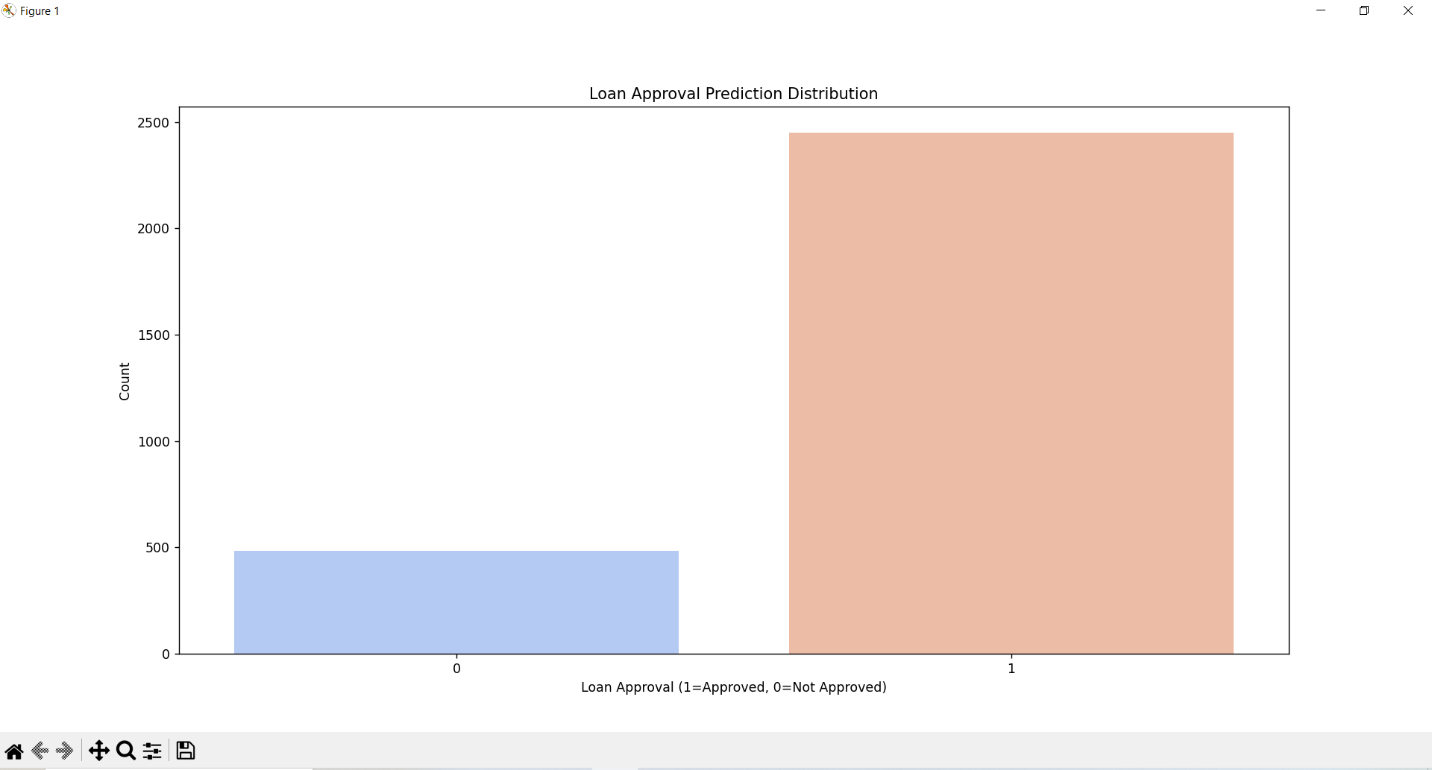
corr\_matrix = assessment\_df[numeric\_columns].corr() # Compute correlation matrix

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm', linewidths=0.5)

plt.title('Heatmap of Feature Correlations')

plt.show()

screen shorts



A graph with different colored squares

Description automatically generated with medium confidence

A graph of a function

Description automatically generated

A screenshot of a graph

Description automatically generated

A screenshot of a computer

Description automatically generated