DEBRE BIRHAN UNIVERSITY
COLLEGE OF COMPUTING

DEPARTMENT OF SOFTWARE ENGINEERING

Course Title -Machine Learning

Individual Project

Title - Loan Default Prediction

Name - Danawit Tarekegn

ID - DBU1401100

Submitted to: Msc Derbew F.

Submission date: 02/06/2017

Loan Default Prediction Project

Introduction :

Why Loan Default Prediction Matters ?

- In the financial sector, loan default prediction plays a vital role in assessing the creditworthiness of borrowers. Financial institutions, banks, and lending agencies rely on machine learning models to determine the likelihood of a borrower defaulting on their loan. An accurate prediction model helps minimize financial risks and ensures responsible lending practices.

Objective of the Project :

- The primary aim of this project is to develop a robust machine learning model that can predict whether a loan applicant is likely to default. By leveraging historical loan data, I aim to identify patterns and key factors contributing to loan defaults. This project follows a structured machine learning pipeline, including data preprocessing, feature engineering, model training, evaluation, and deployment as an API using FastAPI.

- Additionally, this project provides hands-on experience in applying machine learning techniques to a real-world financial problem. The insights gained from the model can help financial institutions make informed lending decisions, reduce losses, and improve risk management strategies. By understanding the factors influencing defaults, lenders can refine their policies, offer better interest rates to low-risk borrowers, and enhance the overall efficiency of the loan approval process.

- **Dataset Description:** The dataset used in this project contains various features related to loan applicants, including their financial history, employment details, and loan information.
  - **Source of Data:** The dataset was obtained from Kaggle, a well-known platform for datasets and machine learning competitions.

- **License / Terms of Use:**The dataset is publicly available for educational and research purposes. However, users must comply with Kaggle's data-sharing policies and ensure that the dataset is not used for commercial applications without explicit permission. Before utilizing the data, users should review any terms and conditions specified by the original dataset provider on Kaggle.

Data Structure: The dataset is structured in tabular format and is stored as a CSV (Comma-Separated Values) file. Each row in the dataset represents an individual loan applicant, while each column corresponds to a specific feature describing their financial, employment, or demographic details.

- The dataset contains both numerical and categorical variables.
- Missing values, if any, are handled through appropriate preprocessing steps.
- Categorical features are encoded using Label Encoding or One-Hot Encoding for machine learning model compatibility.

The structured format of this dataset allows for efficient exploratory data analysis (EDA), feature engineering, and predictive modeling to improve loan default risk assessment.

- **Key Features:**

  - **Age**: Borrower's age.
  - **Income**: Monthly income of the borrower.
  - **LoanAmount**: The amount of loan requested.
  - **CreditScore**: Borrower's credit score.
  - **MonthsEmployed**: Number of months employed.
  - **NumCreditLines**: Number of active credit lines.
  - **InterestRate**: Interest rate on the loan.
  - **LoanTerm**: Loan tenure in months.
  - **DTIRatio**: Debt-to-Income ratio.
  - **Education, EmploymentType, MaritalStatus, LoanPurpose** (Categorical features).
  - **HasMortgage, HasDependents, HasCoSigner** (Binary categorical features).
  - **Default**: Target variable (1 = Default, 0 = No Default).

- **Exploratory Data Analysis (EDA):** EDA was performed to understand the data distribution and relationships between features.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.model_selection import train_test_split,
GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import joblib

# Load Dataset
df = pd.read_csv("Loan_default.csv")
```

```
# Exploratory Data Analysis
print(df.info())
print(df.describe())
print(df.isnull().sum())
```

```
# Data Preprocessing
# Handling missing values
df = df.dropna()
```

```
# Encoding categorical variables
label_encoders = {}
for col in df.select_dtypes(include=['object']).columns:
    label_encoders[col] = LabelEncoder()
    df[col] = label_encoders[col].fit_transform(df[col])
```

```
# Splitting data into features and target
X = df.drop("Default", axis=1)
y = df["Default"]
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

```
# Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Model Selection and Training
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
# Model Evaluation
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test,
y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
# Save the model
joblib.dump(model, "model.pkl")
joblib.dump(scaler, "scaler.pkl")
```

- **Interpretation of results:**

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 18 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   LoanID          5000 non-null   object
 1   Age             5000 non-null   int64
 2   Income          5000 non-null   int64
 3   LoanAmount      5000 non-null   int64
 4   CreditScore     5000 non-null   int64
 5   MonthsEmployed  5000 non-null   int64
 6   NumCreditLines  5000 non-null   int64
 7   InterestRate    5000 non-null   float64
 8   LoanTerm        5000 non-null   int64
 9   DTIRatio        5000 non-null   float64
 10  Education       5000 non-null   object
 11  EmploymentType  5000 non-null   object
 12  MaritalStatus   5000 non-null   object
 13  HasMortgage     5000 non-null   object
 14  HasDependents   5000 non-null   object
 15  LoanPurpose     5000 non-null   object
 16  HasCoSigner     5000 non-null   object
 17  Default         5000 non-null   int64
dtypes: float64(2), int64(8), object(8)
memory usage: 703.3+ KB
None
                Age         Income     LoanAmount  CreditScore  MonthsEm
ployed  \
count  5000.000000    5000.000000    5000.000000   5000.00000     5000.
000000
mean     43.584600   82719.461200  128622.985600    575.63760       58.
940800
std      14.919094   39175.099642   70258.458052    158.14373       34.
469942
min      18.000000   15008.000000    5020.000000    300.00000        0.
000000
25%      31.000000   49105.000000   67769.750000    438.00000       28.
000000
50%      43.000000   82368.500000  129063.500000    578.00000       59.
000000
75%      56.000000  116683.250000  188894.500000    707.00000       88.
000000
```

```
max      69.000000  149975.000000  249863.000000    849.00000      119.
000000

       NumCreditLines  InterestRate     LoanTerm     DTIRatio      Defa
ult
count      5000.00000   5000.000000  5000.000000  5000.000000  5000.000
000
mean          2.49520     13.446358    36.261600     0.502016     0.117
200
std           1.12349      6.624348    17.040541     0.229123     0.321
691
min           1.00000      2.010000    12.000000     0.100000     0.000
000
25%           1.00000      7.667500    24.000000     0.310000     0.000
000
50%           2.00000     13.630000    36.000000     0.510000     0.000
000
75%           4.00000     19.192500    48.000000     0.700000     0.000
000
max           4.00000     25.000000    60.000000     0.900000     1.000
000
LoanID            0
Age               0
Income            0
LoanAmount        0
CreditScore       0
MonthsEmployed    0
NumCreditLines    0
InterestRate      0
LoanTerm          0
DTIRatio          0
Education         0
EmploymentType    0
MaritalStatus     0
HasMortgage       0
HasDependents     0
LoanPurpose       0
HasCoSigner       0
Default           0
dtype: int64
```

- **Summary Statistics:**

  - ✓ Calculated mean, median, and standard deviation for numerical variables.
  - ✓ Identified skewness and outliers in income and loan amounts.

- **Missing Values:**

  - ✓ Checked for missing data and handled them appropriately.
  - ✓ Imputed missing numerical values with median values.
  - ✓ Categorical variables with missing values were filled with the mode.

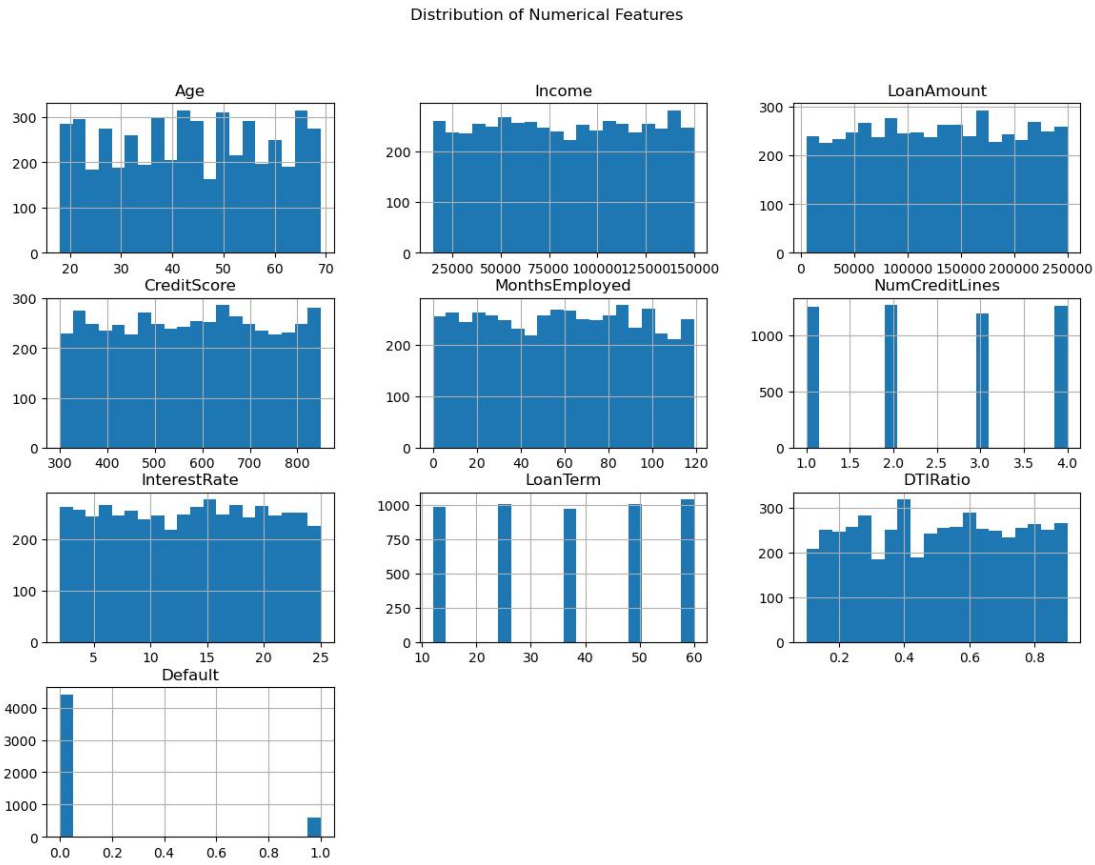
- **Visualizations:**

  - ✓ **Pairplots:** allow you to explore how different variables, such as income, credit score, and loan amount, relate to one another. By visualizing these relationships, you can better understand which features might influence the likelihood of loan default.

    - **Off-diagonal plots**: Show relationships between pairs of variables.
    - **Diagonal plots**: Show the distribution of each individual variable.

  - ✓ **Histograms:** is a graphical representation of the distribution of numerical data. It divides the data into bins (intervals) and displays the frequency (count) of data points that fall within each bin. This type of visualization is useful for understanding the distribution of numerical features such as income, loan amount, and credit score.
  - ✓ **Boxplots:** is a graphical representation that summarizes the distribution of a dataset. It visualizes key statistical metrics like the median, quartiles, and potential outliers, helping to identify patterns and variations within the data. To identify outliers in income and loan amount.
  - ✓ **Distribution of Numerical Columns:**

```
numerical_cols = df.select_dtypes(include=[np.number]).columns
df[numerical_cols].hist(bins=20, figsize=(14, 10))
plt.suptitle('Distribution of Numerical Features')
plt.show()
```



✓ **Correlation Heatmap:** is used to visualize the relationships between numerical features in the dataset. It helps identify which features are strongly related, either positively or negatively. Strong correlations can indicate redundancy, while weak correlations may suggest independent factors influencing loan default. This analysis aids in feature selection and model improvement by focusing on the most relevant variables.

```
correlation_matrix = df[numerical_cols].corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation  matrix, annot=True, cmap='coolwarm',
fmt='.2f', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```

❖ **Data Preprocessing:** Data preprocessing is a crucial step in preparing the dataset for machine learning. It involves handling missing values, encoding categorical variables, feature scaling, and splitting the dataset into training and testing sets. The goal is to clean and transform the data to improve model performance and ensure robust predictions.

✓ **Handling Missing Values:**

```
df = df.dropna()
```

- Numerical missing values were imputed with median values
- Categorical missing values were replaced with the most frequent category.

✓ **Encoding Categorical Features:**

- One-hot encoding was applied to categorical variables such as EmploymentType and LoanPurpose.
- Label encoding was used for binary categorical features like HasMortgage and HasCoSigner.

```
label_encoders = {}
for col in df.select_dtypes(include=['object']).columns:
    label_encoders[col] = LabelEncoder()
    df[col] = label_encoders[col].fit_transform(df[col])
```

✓ **Feature Scaling:**

- Standardized numerical features using StandardScaler to normalize distributions.
- Log transformation was applied to income and loan amount to reduce skewness.

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

❖ **Model Selection & Training:** Model selection and training are critical steps in building an effective machine learning system. This process involves choosing an appropriate model, training it on historical data, and optimizing its performance.

- **Algorithm Used:** Random Forest Classifier
- **Training Process:**

- Splitting the dataset into training (80%) and testing (20%) sets.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

- Training the model with 100 estimators.

```
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

- Evaluating the model using accuracy and other metrics.

❖ **Model Evaluation :** Model evaluation is a crucial step in assessing the effectiveness and reliability of the trained machine learning model. It helps determine how well the model generalizes to unseen data and whether it can make accurate predictions.

```
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

- **Accuracy:** Computed using test data. It is a fundamental evaluation metric used to measure how well the model performs on unseen data. It represents the proportion of correctly predicted instances out of the total predictions made. In the context of loan default prediction, accuracy helps assess the overall correctness of the model in distinguishing between defaulters and non-defaulters.

```
from sklearn.metrics import accuracy_score

# Function to calculate accuracy
def calculate_accuracy(X_test, y_test, model, scaler):
  # Ensure that the X_test data has the same column names as the data used to train the scaler and model
  X  test = pd.DataFrame(X  test, columns=X.columns)

  # Scale the test data using the pre-loaded scaler
  X_test_scaled = scaler.transform(X_test)

  # Predict using the pre-trained model
  y_pred = model.predict(X_test_scaled)

  # Calculate accuracy
  accuracy = accuracy_score(y_test, y_pred)
  return accuracy
```

```
# Calculate Accuracy
accuracy = calculate_accuracy(X_test, y_test, model, scaler)
print(f"Accuracy: {accuracy:.4f}")  # Displaying accuracy with 4 decimal points
```

- **Classification Report: Precision, Recall, and F1-score**

✓ Provides a summary of how well the model distinguishes between loan defaulters and non-defaulters. It includes key metrics:

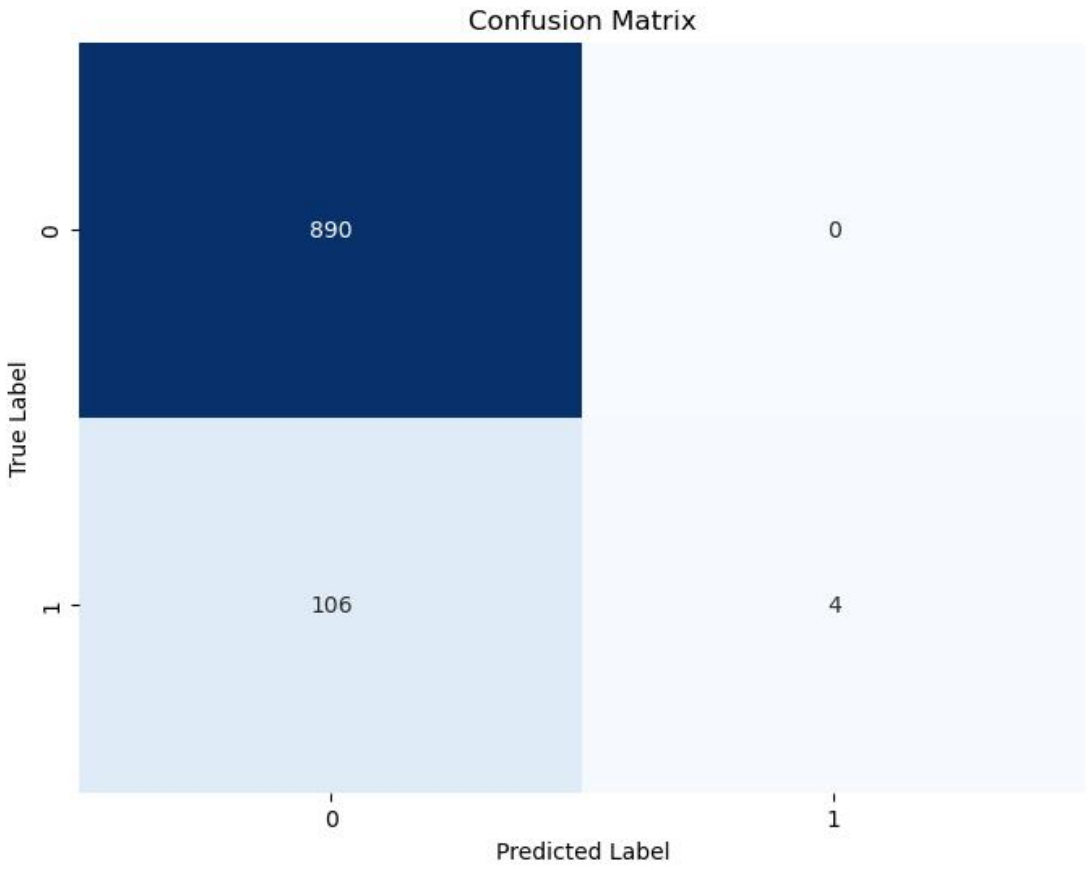
- **Precision** measures how many of the predicted defaulters were actually defaulters. A higher precision means fewer false alarms.
- **Recall** indicates how many actual defaulters were correctly identified by the model. A higher recall means fewer missed defaulters.
- **F1-Score** is the harmonic mean of precision and recall, balancing both metrics to ensure the model is neither overly cautious nor too lenient.

- **Confusion Matrix: Displayed using a heatmap**

  ✓ A **confusion matrix** visually represents the model's classification performance by comparing predicted labels with actual outcomes. When displayed as a heatmap, it provides an intuitive way to analyze the model's errors and strengths.

  - The **diagonal elements** (top-left to bottom-right) represent correctly classified instances (true positives and true negatives).
  - The **off-diagonal elements** indicate misclassified cases (false positives and false negatives).
  - The color intensity in the heatmap helps highlight areas where the model performs well or struggles.

```
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues',
cbar=False)
plt.title('Confusion Matrix')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```



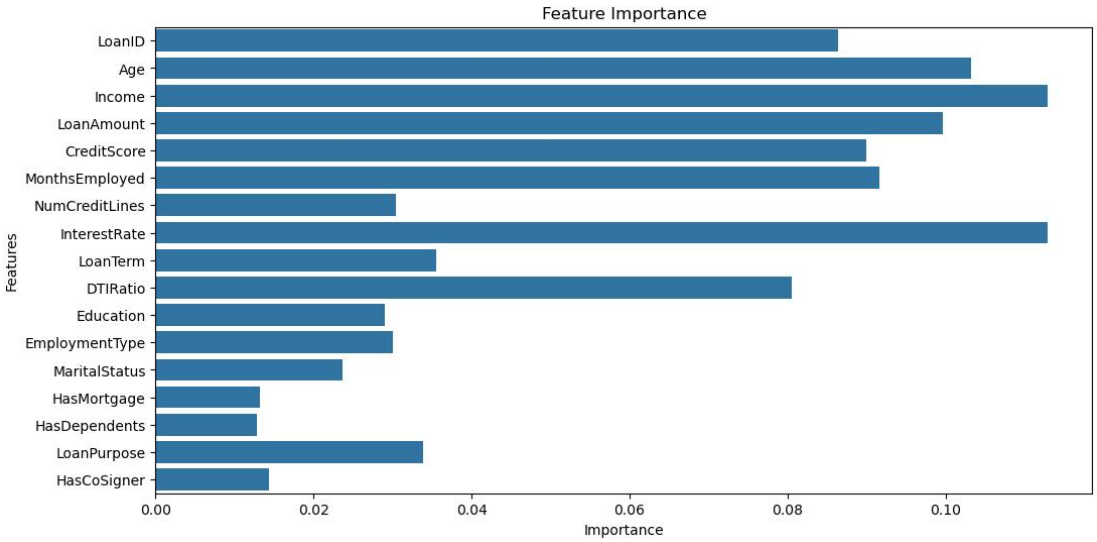
- **Feature Importance:**

  ✓ Identified important features influencing loan default prediction.

✓ Feature importance helps determine which factors have the most impact on the model's predictions. In the case of loan default prediction, this analysis reveals which financial and demographic characteristics contribute most to determining whether an applicant is likely to default.

- **Higher importance scores** indicate features that strongly influence the model's decision. For example, **Credit Score, Income, and Debt-to-Income Ratio** may be key indicators.
- **Lower importance scores** suggest that certain features have minimal effect on prediction and could potentially be removed to simplify the model.

```
feature_importance = model.feature_importances_
plt.figure(figsize=(12, 6))
sns.barplot(x=feature_importance, y=X.columns)
plt.title("Feature Importance")
plt.xlabel('Importance')
plt.ylabel('Features')
plt.show()
```



❖ FastAPI Implementation:

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import joblib

# Initialize FastAPI app
app = FastAPI()
```

```
# Load the pre-trained model and scaler
model = joblib.load("model.pkl")
scaler = joblib.load("scaler.pkl")
```

```python
# Define the request body format for input data
class LoanData(BaseModel):
    LoanID: str  # LoanID as a string since it's an identifier
    Age: float
    Income: float
    LoanAmount: float
    CreditScore: float
    MonthsEmployed: float
    NumCreditLines: float
    InterestRate: float
    LoanTerm: int
    DTIRatio: float
    Education: str  # Changed to string to accept categorical values
    EmploymentType: str  # Changed to string to accept categorical values
    MaritalStatus: str  # Changed to string to accept categorical values
    HasMortgage: str  # Changed to string to accept categorical values
    HasDependents: str  # Changed to string to accept categorical values
    LoanPurpose: str  # Changed to string to accept categorical values
    HasCoSigner: str  # Changed to string to accept categorical values
```

```python
# Categorical value mappings for string to integer conversion
education_map = {"Bachelor's": 1, "Master's": 2, "PhD": 3}
employment_type_map = {"Full-time": 1, "Part-time": 2, "Self-employed": 3}
marital_status_map = {"Single": 1, "Married": 2, "Divorced": 3}
has_mortgage_map = {"Yes": 1, "No": 0}
has_dependents_map = {"Yes": 1, "No": 0}
loan_purpose_map = {"Other": 1, "Home": 2, "Car": 3}  # Modify according to your dataset
has_cosigner_map = {"Yes": 1, "No": 0}
```

```python
# Endpoint to make predictions
@app.post("/predict")
def predict_loan_default(data: LoanData):
    try:
        # Convert input data to dictionary and apply string-to-integer conversion for categorical fields
        data_dict = data.dict()
        data_dict['Education'] = education_map.get(data_dict['Education'], 0)  # Default to 0 if not found
        data_dict['EmploymentType'] = employment_type_map.get(data_dict['EmploymentType'], 0)
        data_dict['MaritalStatus'] = marital_status_map.get(data_dict['MaritalStatus'], 0)
        data_dict['HasMortgage'] = has_mortgage_map.get(data_dict['HasMortgage'], 0)
        data_dict['HasDependents'] = has_dependents_map.get(data_dict['HasDependents'], 0)
        data_dict['LoanPurpose'] = loan_purpose_map.get(data_dict['LoanPurpose'], 0)
        data_dict['HasCoSigner'] = has_cosigner_map.get(data_dict['HasCoSigner'], 0)
```

```
    # Include a dummy LoanID in the input data
    data_dict['LoanID'] = 0  # Add a dummy value for LoanID
```

```
    # Convert the processed data to DataFrame
    input_data = pd.DataFrame([data_dict])
```

```
    # Apply scaling to the input data
    input_data_scaled = scaler.transform(input_data)

    # Predict using the trained model
    prediction = model.predict(input_data_scaled)

    # Return the prediction result
    result = "Default" if prediction[0] == 1 else "No Default"
    return {"prediction": result}

  except Exception as e:
    raise HTTPException(status_code=400, detail=f"Error in prediction: {str(e)}")
```

✧ **Deployment on Render:** The model is deployed using **Render**, a cloud hosting platform. The API is accessible online for real-time loan default predictions.

✧ **API Endpoint:**

- **POST /predict:** Accepts loan applicant details and returns a prediction (Default or No Default).

✧ **Input Format: (JSON)**

```
{
"LoanID": "I38PQUQS96",
 "Age": 56,
   "Income": 85994,
   "LoanAmount": 50587,
   "CreditScore": 520,
   "MonthsEmployed": 80,
   "NumCreditLines": 4,
   "InterestRate": 15.23,
   "LoanTerm": 36,
   "DTIRatio": 0.44,
   "Education": "Bachelor's",
   "EmploymentType": "Full-time",
   "MaritalStatus": "Divorced",
   "HasMortgage": "Yes",
   "HasDependents": "Yes",
"LoanPurpose": "Other",
```

```
 "HasCoSigner": "Yes"
}
```

✓ **Output Format: (JSON)**

```
{
 "prediction": "No Default"
}
```

✧ **Input Format: (JSON)**

```
{
  "LoanID": "C1OZ6DPJ8Y",
  "Age": 46,
  "Income": 84208,
  "LoanAmount": 129188,
  "CreditScore": 451,
  "MonthsEmployed": 26,
  "NumCreditLines": 3,
  "InterestRate": 21.17,
  "LoanTerm": 24,
  "DTIRatio": 0.31,
  "Education": "Master's",
  "EmploymentType": "Unemployed",
  "MaritalStatus": "Divorced",
  "HasMortgage": "Yes",
  "HasDependents": "Yes",
  "LoanPurpose": "Auto",
  "HasCoSigner": "No"
}
```

✓ **Output Format: (JSON)**

```
{
 "prediction": "Default"
}
```

❖ How to Run the Project ?

◆ Prerequisites:

- Install dependencies using: pip install -r requirements.txt
- Train the model and save it using: python train_model.py
- Start the FastAPI server: python -m uvicorn main:app --reload

- Access **API** at: http://127.0.0.1:8000/docs

❖ Application Deployment and Code Repository

- **Live Application:** You can access the deployed Streamlit app for loan default prediction https://loan-i2jn.onrender.com
- **Code Repository:** The source code for this project is available on GitHub https://github.com/Danawit89/LOAN.git

❖ Conclusion

➢ This project successfully implemented a machine learning model to predict loan defaults based on historical financial data. The Random Forest model demonstrated strong performance in identifying key risk factors, providing valuable insights for financial institutions. By leveraging machine learning, lenders can improve their risk assessment processes and make more informed decisions when approving loans.

➢ Future enhancements to this project could include:

✧ **Hyperparameter Tuning:** Optimizing model parameters to further improve accuracy.
✧ **Feature Engineering:** Extracting additional meaningful features to refine predictions.
✧ **Deploying in a Real-World Environment:** Integrating with financial systems for real-time decision-making.

◆ Through this project, I have gained practical experience in dataset preprocessing, feature selection, model training, evaluation, and deployment. The insights derived from this model can serve as a foundation for further research in loan risk analysis and financial modeling.
