Loan Default Prediction

Sara Vaez

Introduction

Can we accurately predict a borrower's likelihood of defaulting on a loan based on their financial history and related features such as income, loan amount, credit score, and other demographic or behavioral attributes?

The primary goal of this project is to develop a machine learning model that predicts whether a borrower will default on a loan. This involves gathering and cleaning a real-world dataset, performing exploratory data analysis (EDA) to understand key patterns, engineering relevant features, and applying Logistic Regression and Random Forest machine learning models to build an accurate classifier.

Methodology

In this project, I Develop a streamlined workflow (including data cleaning, feature engineering, and model evaluation) that could potentially be integrated into a real-world credit risk assessment pipeline.

This project will not only validate the hypothesis that key borrower attributes can forecast loan default, but it will also provide a practical, end-to-end demonstration of how data science and machine learning can drive informed decision-making in the financial services industry.

Data Description

This dataset is from Kaggle Loan Default Dataset, which contains 255,347 rows and 18 columns in total.

Features

	Column_name	Column_type	Data_type	Description
0	LoanID	Identifier	string	A unique identifier for each loan.
1	Age	Feature	integer	The age of the borrower.
2	Income	Feature	integer	The annual income of the borrower.
3	LoanAmount	Feature	integer	The amount of money being borrowed.
4	CreditScore	Feature	integer	The credit score of the borrower, indicating their creditworthiness.
5	MonthsEmployed	Feature	integer	The number of months the borrower has been employed.
6	NumCreditLines	Feature	integer	The number of credit lines the borrower has open.
7	InterestRate	Feature	float	The interest rate for the loan.
8	LoanTerm	Feature	integer	The term length of the loan in months.
9	DTIRatio	Feature	float	The Debt-to-Income ratio, indicating the borrower's debt compared to their income.
10	Education	Feature	string	The highest level of education attained by the borrower (PhD, Master's, Bachelor's, High School).
11	EmploymentType	Feature	string	The type of employment status of the borrower (Full-time, Part-time, Self-employed, Unemployed).
12	MaritalStatus	Feature	string	The marital status of the borrower (Single, Married, Divorced).
13	HasMortgage	Feature	string	Whether the borrower has a mortgage (Yes or No).
14	HasDependents	Feature	string	Whether the borrower has dependents (Yes or No).
15	LoanPurpose	Feature	string	The purpose of the loan (Home, Auto, Education, Business, Other).
16	HasCoSigner	Feature	string	Whether the loan has a co-signer (Yes or No).
17	Default	Target	integer	The binary target variable indicating whether the loan defaulted (1) or not (0).

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [12]:
```
df = pd.read_csv('Loan_default.csv')
df.head()
```

Out[12]:

	LoanID	Age	Income	LoanAmount	CreditScore	MonthsEmployed	NumCreditLines	InterestRate	LoanTerm	DTIRatio	Education	EmploymentType	M
0	I38PQUQS96	56	85994	50587	520	80	4	15.23	36	0.44	Bachelor's	Full-time	
1	HPSK72WA7R	69	50432	124440	458	15	1	4.81	60	0.68	Master's	Full-time	
2	C1OZ6DPJ8Y	46	84208	129188	451	26	3	21.17	24	0.31	Master's	Unemployed	
3	V2KKSFM3UN	32	31713	44799	743	0	3	7.07	24	0.23	High School	Full-time	
4	EY08JDHTZP	60	20437	9139	633	8	4	6.51	48	0.73	Bachelor's	Unemployed	

Data Understanding and Data Cleaning

I check for missing data and drop rows or fill resulted missing values. In order to prepare dataset for using in machine learning workflows, I explore, clean and analyze data.

In [13]:
```
# 1) check for missing data

print(df.isnull().sum())

# drop rows or fill missing values

df.fillna(df.mean(), inplace=True)
```

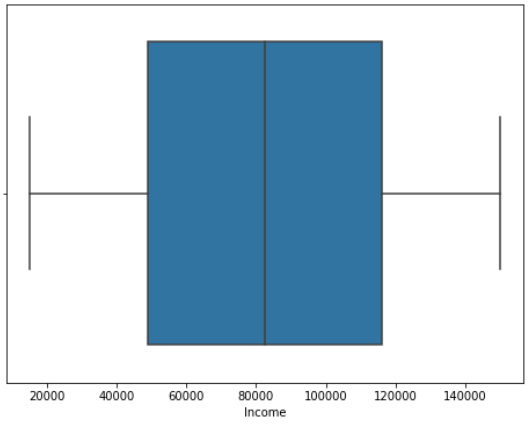
```
LoanID           0
Age              0
Income           0
LoanAmount       0
CreditScore      0
MonthsEmployed   0
NumCreditLines   0
InterestRate     0
LoanTerm         0
DTIRatio         0
Education        0
EmploymentType   0
MaritalStatus    0
HasMortgage      0
HasDependents    0
LoanPurpose      0
HasCoSigner      0
Default          0
dtype: int64
```

Fortunately, the dataset is complete and doesn't have any missing value, otherwise we requested to replace the missing cells with the average value of that column.

Next, I identify duplicate rows and recognize if the dataset contains outliers that may skew the analysis.

In [14]:
```
# 2) Identify duplicate rows

df.drop_duplicates(inplace=True)
```

In [15]:
```
# 3) Handle outliers using Boxplots

# a boxplot for income distribution

plt.figure(figsize=(8,6))
sns.boxplot(x=df['Income'])
plt.show()
```



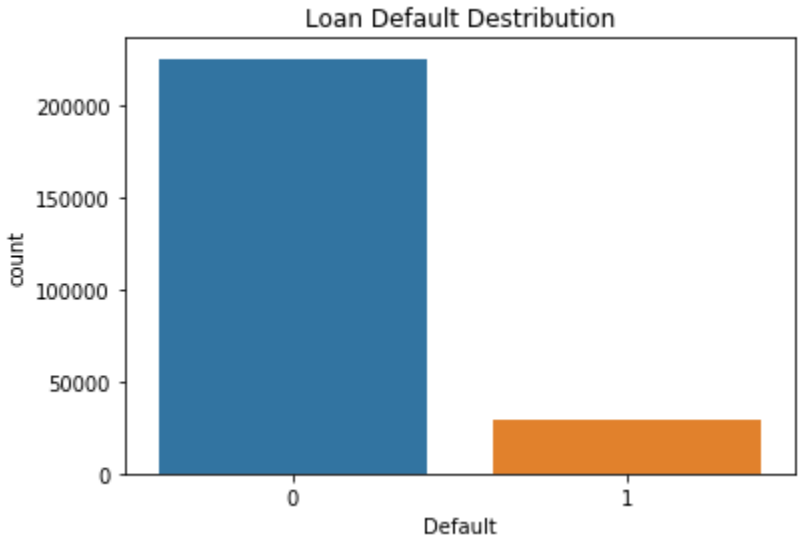
The goal of this step is to visually inspect the distribution of the "Income" column and identify any outliers. A boxplot summarizes the data by showing the median, quartiles, and potential outliers. This visual tool helps quickly identify whether the "Income" column has extreme values that might impact my analysis or modeling. I consider "Income" data because Income is often a key variable in financial datasets. It typically has a wide range of values and may contain extreme values (outliers) that could skew the analysis. In this dataset there is no outliers in the Income column.

Exploratory Data Analysis (EDA)

In this section, I try to understand the data distribution and its statistic information. Additionally, I compare the number of default borrowers with non-default borrowers in the dataset.

In [16]:
```
# 1) Understanding the data distribution: Summary statistics

print(df.describe())

# count of default vs. non-default borrowers

sns.countplot(x = 'Default', data=df)
plt.title('Loan Default Destribution')
plt.show()
```

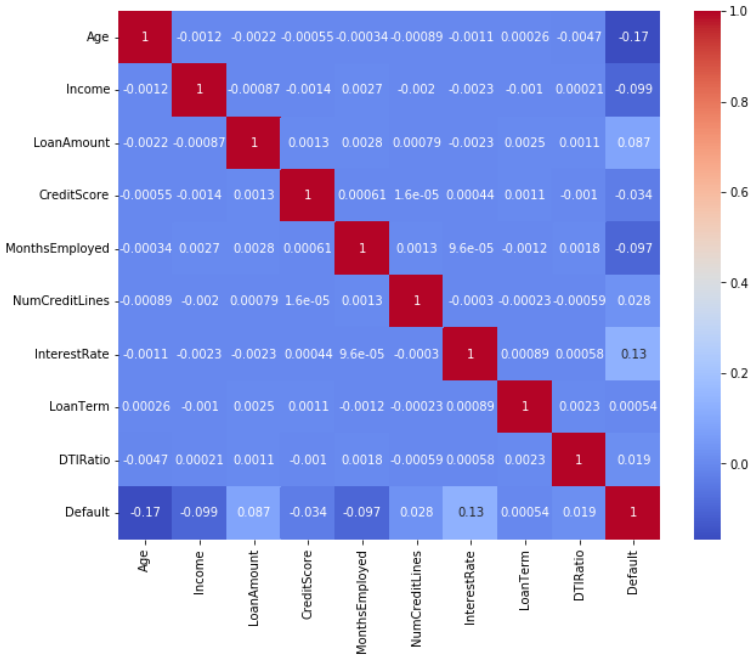
```
                Age         Income     LoanAmount    CreditScore  \
count  255347.000000  255347.000000  255347.000000  255347.000000
mean       43.498306   82499.304597  127578.865512     574.264346
std        14.990258   38963.013729   70840.706142     158.903867
min        18.000000   15000.000000    5000.000000     300.000000
25%        31.000000   48825.500000   66156.000000     437.000000
50%        43.000000   82466.000000  127556.000000     574.000000
75%        56.000000  116219.000000  188985.000000     712.000000
max        69.000000  149999.000000  249999.000000     849.000000

       MonthsEmployed  NumCreditLines   InterestRate       LoanTerm  \
count   255347.000000   255347.000000  255347.000000  255347.000000
mean        59.541976        2.501036      13.492773      36.025894
std         34.643376        1.117018       6.636443      16.969330
min          0.000000        1.000000       2.000000      12.000000
25%         30.000000        2.000000       7.770000      24.000000
50%         60.000000        2.000000      13.460000      36.000000
75%         90.000000        3.000000      19.250000      48.000000
max        119.000000        4.000000      25.000000      60.000000

            DTIRatio        Default
count  255347.000000  255347.000000
mean        0.500212       0.116128
std         0.230917       0.320379
min         0.100000       0.000000
25%         0.300000       0.000000
50%         0.500000       0.000000
75%         0.700000       0.000000
max         0.900000       1.000000
```



Correlation analysis

The following graph illustrates the potential pairwise correlation coefficients for all numerical columns in the DataFrame. The correlation values are in the range of -1 to 1.

```
In [7]: # 2) corrolation analysis using heatmap

        plt.figure(figsize=(10,8))
        sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
        plt.show()
```



Based on the heatmap above, we observe the following:

- **Age and Default:** There is a weak negative correlation between Age and Default, suggesting that older individuals are slightly less likely to default.
- **Income and Default:** Higher income is associated with a marginally reduced probability of default.
- **Loan Amount and Default:** Larger loan amounts slightly increase the likelihood of default.
- **Interest Rate and Default:** Higher interest rates tend to slightly increase default risk, with the highest observed correlation between Interest Rate and Default (0.13).

Overall, most correlations are very weak, indicating that no single variable strongly predicts default. However, factors like income, age, and interest rate do have small influences.

Given these findings, it would be beneficial to employ nonlinear modeling techniques to capture any hidden relationships in the data. For example, you could:

- **Train a machine learning model** such as a Random Forest or Logistic Regression.
- **Extract feature importances** to identify which factors most significantly influence default.

Feature Engineering

I create new features and measurements from information in the dataset for better analysis.

In [8]:
```
# 3) Feature Engineering => Creating New Features

#creating LTI = Loan-to-Income ratio

df['LTI'] = df['LoanAmount'] / df['Income']
```

Data Processing and Feature Selection

Many machine learning models require numeric inputs to be able to create an efficient model. Thus, I convert categorical values to numerical using One-hot encoding and Target Encoding.

In the following One-hot encoding we create separate columns. It has converted the categorical columns of "EmploymentType" and "LoanPurpose" into several new binary (0/1) columns. When we have too many unique values, this will cause memory issues.

In [17]:
```
# 1) Convert categorical variables to Numeric. Because many ML models require numeric inputs

df = pd.get_dummies(df, columns=['EmploymentType', 'LoanPurpose'], drop_first=True)
```

In this dataset, we have many categorical columns (object type), so I use Target Encoding, instead. Instead of creating multiple binary columns, target encoding replaces each category with a value computed from the target variable. For example, if a category "Employed" in 'EmploymentType' is associated with a 5% default rate, the category is replaced with 0.05. This code automatically identifies all object-type columns in X_train and applies the target encoder, then applies the same transformation to X_test. One-hot encoding can result in many new columns if a categorical variable has many unique values. Target encoding keeps the number of features the same as the original columns.

In [21]:
```
# Instead of one-hot encoding, replace categories with their mean target values. This is efficient and avoids memory issues.

from category_encoders import TargetEncoder

# Identify all categorical columns (object type)
categorical_cols = X_train.select_dtypes(include=['object']).columns

# Initialize Target Encoder
encoder = TargetEncoder()

# Apply Target Encoding
X_train[categorical_cols] = encoder.fit_transform(X_train[categorical_cols], y_train)
X_test[categorical_cols] = encoder.transform(X_test[categorical_cols])
```

Train-Test Split & Model Selection

Split the dataset into training and testing sets

```
In [20]: from sklearn.model_selection import train_test_split

         X = df.drop(columns =['Default']) # Features
         y = df['Default'] # Target variable

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

This part removes the 'Default' column from the dataset because it is the value I aim to predict. The remaining columns (like Income, Age, etc.) are the features that I will use to predict default.

Splitting the dataset into features and target helps the model understand what inputs (features) should be used to predict the output (target). In this code, 20% of the data will be kept aside for testing, and the remaining 80% will be used for training.

Model Training & Evaluation

- **Train a Logistic Regression Model**

training and evaluating a Logistic Regression model using scikit-learn.

It helps in binary classification problems, such as predicting whether a loan will be repaid or not (1 = Yes, 0 = No).

```
In [30]: from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import accuracy_score, classification_report

         # train model
         model = LogisticRegression()
         model.fit(X_train, y_train)

         # Predictions
         y_pred = model.predict(X_test)

         # Model Evaluation => Compares the model's predictions (y_pred) with the actual values (y_test).
         # e.g. Accuracy: 0.87 => Means 87% of predictions were correct

         print('Accuracy:', accuracy_score(y_test, y_pred))
         print('Classification Report:\n', classification_report(y_test, y_pred))
```

```
Accuracy: 0.8851576267867632
Classification Report:
              precision    recall  f1-score   support

           0       0.89      1.00      0.94     45170
           1       0.63      0.01      0.03      5900

    accuracy                           0.89     51070
   macro avg       0.76      0.51      0.48     51070
weighted avg       0.86      0.89      0.83     51070
```

Based on the classification report, for the class 0 (Non-default), the precision amount is 0.89, which means of all instances predicted as non-default, 89% were truly non-default. Recall is 1.00 meaning from all actual non-default cases, 100% were correctly identified by the model. The F1-score, which combines precision and recall into a single metric is 0.94 and indicates very good performance for class 0.

However, for the class 1 (default) Precision is 0.63, which is of all instances predicted as default, 63% were actually defaults. Also, the Recall score is 0.01. Out of all actual default cases, the model only identified

1% correctly. Finally, F1-Score is 0.03. This low score reflects that the model is performing very poorly on identifying defaults.

The model performs very well in predicting class 0 (non-default), which is the majority class.

However, it struggles significantly with class 1 (default). The very low recall (0.01) for defaults means that the model misses 99% of actual defaults. This is a critical issue if predicting defaults is your main goal.

- **Improve Performance with Random Forest Model**

The model learns from the training data by creating multiple decision trees based on different random parts of the data.

In [31]:
```
from sklearn.ensemble import RandomForestClassifier

# create the model with 100 decision trees. The idea is to build many small models (trees) and combine their results.
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train (fit) the model on the training data
rf_model.fit(X_train, y_train)

# Make prediction on the test set
y_pred_rf = rf_model.predict(X_test)

# Evaluate how accurate the model is
print('Random Forest Accuracy:', accuracy_score(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 0.8844722929312708
```

Conclusion

In this project, I developed a predictive model for loan defaults using a real-world dataset, employing a comprehensive data science workflow. The process began with data cleaning—addressing missing values and identifying outliers through boxplots—to ensure data quality. I conducted exploratory data analysis (EDA), including generating a heatmap to visualize the correlations among key features, which helped reveal the weak linear relationships between variables and default.

I performed feature engineering and applied encoding techniques to transform categorical variables into numerical ones, enabling the effective use of machine learning algorithms. After splitting the data into training and testing sets, I standardized key features to enhance model performance.

To complete machine learning modeling, I trained both a Logistic Regression model and a Random Forest model. The Logistic Regression model provided insights into the linear relationships between features and default, while the Random Forest model captured nonlinear interactions and allowed me to extract feature importances—highlighting which factors had the most influence on predicting defaults.

Overall, this project showcased a robust approach to predictive modeling through meticulous data preparation, exploratory analysis, and the application of both linear (Logistic Regression) and nonlinear (Random Forest) models. Although no single variable strongly predicts default, features like age, income, and interest rate exhibited modest influences. The dual-model strategy not only improved predictive performance but also demonstrated my proficiency in selecting and implementing diverse machine learning techniques to address complex financial challenges.