Credit Score Prediction

Credit score prediction is a critical tool for financial institutions to evaluate customers' creditworthiness and make informed lending decisions.

By analyzing customer attributes such as income, payment behavior, and credit utilization, a machine learning model can classify individuals into appropriate credit score ranges.

This project focuses on developing a robust classification model to predict credit scores, enabling efficient risk assessment and personalized financial solutions.

Dataset Overview

the **test data** does **not** include the **target** column **Credit_score**

so we will **ignore** it and we will work only with training set as train and test.

The train dataset contains **100,000 entries** and **28 columns**. It includes a mix of numerical and categorical variables :

1. **Numerical Columns**:

   Include features like salary, accounts, credit inquiries, and ratios.

   Columns: Monthly_Inhand_Salary, Num_Bank_Accounts, Num_Credit_Card, Interest_Rate, Delay_from_due_date, Num_Credit_Inquiries, Credit_Utilization_Ratio, Total_EMI_per_month.

2. **Categorical Columns**:
   - Include features like IDs, occupation, payment behavior, and credit score.
   - Columns: ID, Customer_ID, Name, Age, SSN, Occupation, Type_of_Loan, Credit_Mix, Payment_Behaviour, etc.

Data Quality Issues Identified:

1. **Missing Values**:
    - Columns like Name, Monthly_Inhand_Salary, Type_of_Loan, Num_of_Delayed_Payment, Credit_History_Age, and Amount_invested_monthly contain missing values.

2. **Irregular Values**:
    - Credit_Mix: Contains an underscore (_) as an invalid category.
    - Payment_Behaviour: Includes invalid entry !@9#%8.
    - Payment_of_Min_Amount: Contains the category NM (possibly ambiguous or a typo).
    - Occupation: Contains placeholder value _______.

3. **Data Types**:
    - Some columns with numerical information (e.g., Age, Annual_Income, Num_of_Loan, Outstanding_Debt) are stored as strings, requiring conversion.

Data Cleaning Process

To ensure the dataset is clean and suitable for analysis, we applied the following data cleaning steps:

1.Cleaning Numerical Columns

For columns that should hold numerical values but are stored as strings, we performed the following operations:

- **Remove Invalid Characters**: We removed any non-numeric characters, such as underscores (_), commas, or special symbols.
- **Convert Data Types**: After cleaning, we converted these columns to appropriate numerical types (float or int).

2.Cleaning Categorical Columns

For columns with categorical values, we addressed irregularities such as placeholder values, invalid entries, and typos.

- **Standardize Categories**: Replace placeholders or ambiguous entries with appropriate labels (e.g., "Other" for unknown categories).
- **Remove Noise**: Remove or map invalid entries like special symbols or empty strings.

3.Handling Missing Values

We handled missing data using appropriate imputation strategies:

- **Numerical Columns**: Imputed missing values with the median or mean of the column.
- **Categorical Columns**: Filled missing values with the mode or replaced them with a new category like Unknown.

4.Ensuring Consistent Data Types

After cleaning, we ensured each column had the correct data type:

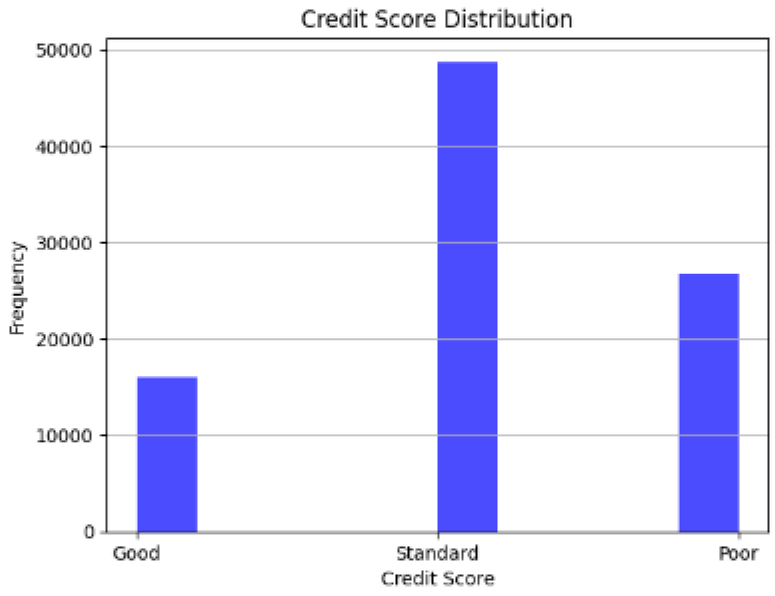
- Numerical columns were converted to float or int.
- Categorical columns were encoded as category.

Dealing with Imbalanced Data Set :

Addressing Imbalanced Credit Score Distribution

The dataset exhibits an imbalanced distribution of credit score values, which could lead to biased predictions by the machine learning model. The class counts are as follows:

1. **Good**: 17,828 instances
2. **Poor**: 28,998 instances
3. **Standard**: 53,174 instances



This imbalance can cause the model to overfit to the majority class (Standard) and underperform in predicting the minority classes (Good and Poor). To address this issue, we will employ techniques to balance the dataset, ensuring the model learns effectively from all classes.

Strategies for Handling Imbalanced Data

Oversampling Minority Classes

- Increase the number of instances in the Good and Poor classes by duplicating existing samples or generating synthetic data.
- **SMOTE (Synthetic Minority Oversampling Technique)**: Generate synthetic samples for minority classes based on their nearest neighbors.

Handling Outliers

Outliers are data points that significantly differ from the majority of the data and can distort statistical analyses or model predictions.

To improve model performance and avoid bias, it is important to detect and handle these outliers effectively.

We calculated it by calculates the mean and standard deviation. It then identifies outliers in each column based on a range defined by the mean plus or minus 4 times the standard deviation and removes these outliers from the dataset.

Feature Selection

In the feature selection phase, we identify and remove irrelevant or redundant columns that do not contribute significantly to the predictive power of the machine learning model. This step is crucial for improving model efficiency, reducing overfitting, and enhancing interpretability.

Explanation of Columns Removed:

1. **Name**: The Name column is a unique identifier for each customer, and it does not provide any meaningful information for predicting the credit score. It is highly unlikely to correlate with the target variable and may lead to overfitting.
2. **ID and Customer_ID**: Both of these columns contain unique identifiers for each individual in the dataset. These are essentially categorical variables that are not relevant for model training as they do not contribute to the prediction of credit scores.

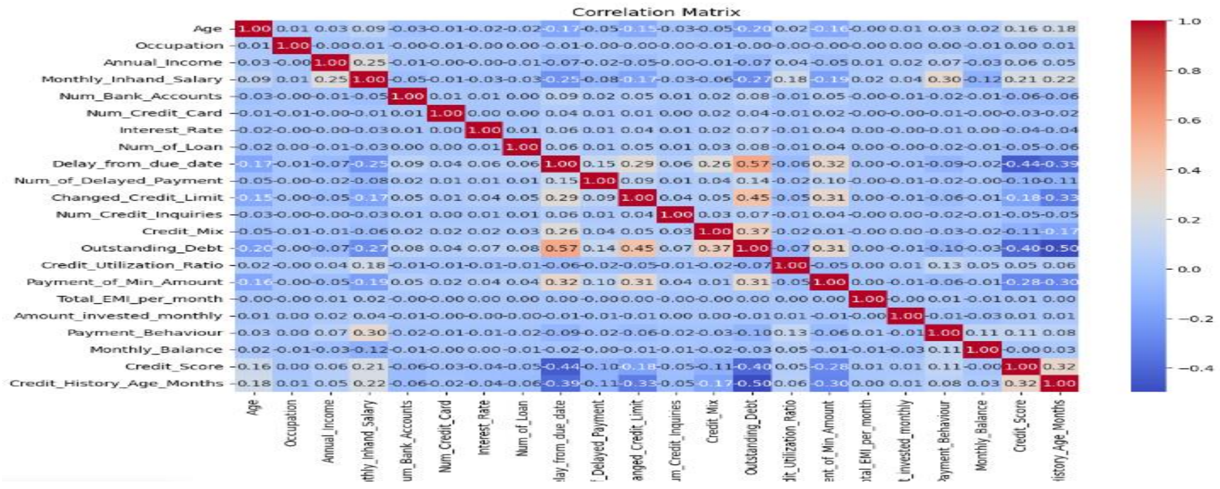
3. **SSN :**The SSN is a personal identifier, and while it may uniquely identify an individual, it does not provide any useful information for predicting credit score. Additionally, using such sensitive data may raise privacy concerns.
4. **Month**: The Month column represents a time-related variable, which may not directly influence the credit score. Since the credit score is typically a more static measure of financial behavior, the specific month might not offer much predictive power in this context.

5. **Type_of_Loan**: While this could be a useful feature in some cases, in this dataset, the Type_of_Loan column had significant missing values, and it was deemed less relevant for the specific objective of predicting credit scores. It may not add meaningful value in comparison to other available features.

Data Visualization

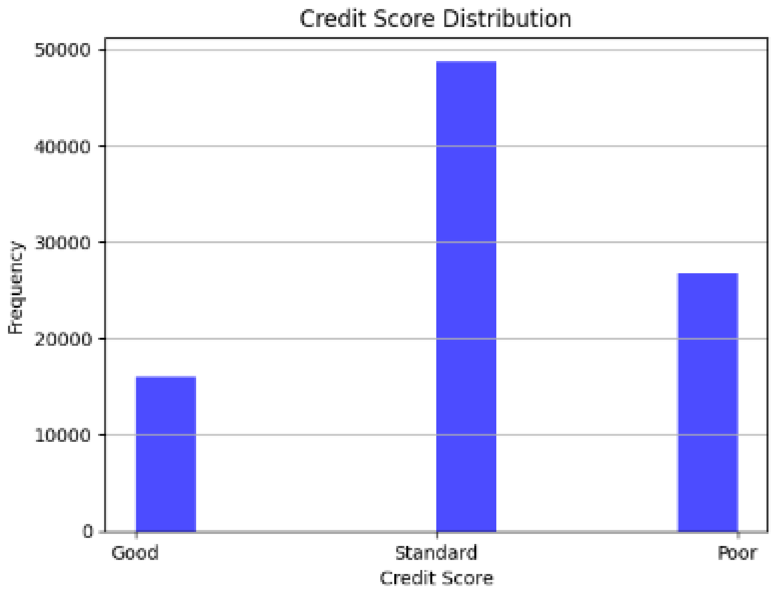
1.Correlation Heatmap for Numerical Columns

The correlation heatmap reveals relationships between various numerical features in the dataset. Notably, **"Annual_Income"** and **"Monthly_Inhand_Salary"** exhibit a strong positive correlation, as indicated by the red squares in the heatmap. This makes sense, as higher annual income typically corresponds to a higher monthly salary. Additionally, **"Age"** and **"Credit_Utilization_Ratio"** show a slight negative correlation, suggesting that as age increases, credit utilization tends to decrease, albeit with a weak relationship.



2.Credit Score Distribution

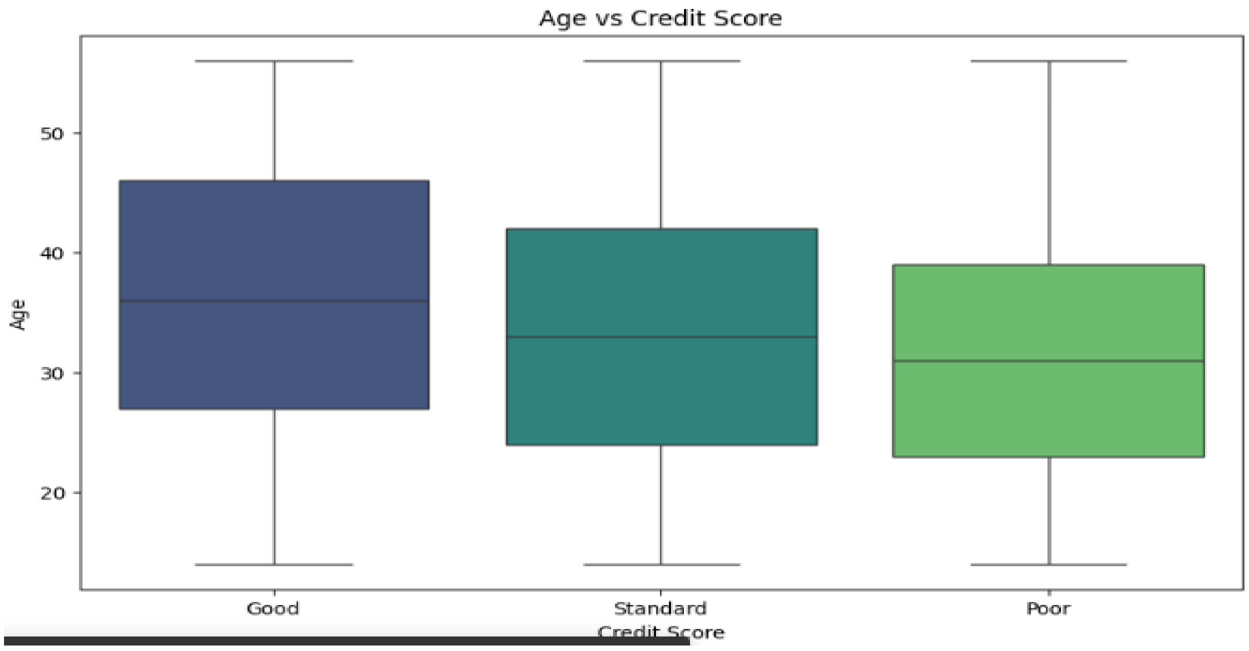
The distribution of the target variable, **Credit_Score**, is highly imbalanced across the dataset. The majority of customers have a **Standard** credit score, followed by **Poor** and **Good** credit scores. This imbalance poses a challenge for the model, as it may lead to biased predictions that favor the majority class.



3.Relation Between Age and Credit Score

The relationship between **Age** and **Credit_Score** reveals some interesting patterns:

- **Good Credit Score**: Individuals with a good credit score show a wide age distribution, ranging from the early 20s to over 50 years old. This suggests that people of various age groups can maintain a high credit score, indicating a broader demographic of financially responsible individuals.
- **Standard Credit Score**: The age distribution for individuals with a standard credit score is slightly narrower, with most individuals concentrated between 30 and 50 years of age.



-

4.Relation Between Monthly Balance and Credit Score

When analyzing the **Monthly_Balance** in relation to credit score, we observe that all three credit score categories (**Good**, **Standard**, and **Poor**) have nearly identical distributions for **Monthly_Balance**. There is minimal variability across these categories, as indicated by the extremely compressed y-axis scale. This suggests that **Monthly_Balance** has limited discriminative power in predicting the credit score of individuals.

5.Outstanding Debt and Credit Score

The distribution of **Outstanding_Debt** across credit score categories shows some distinct patterns:

- **Poor Credit Score**: The **Outstanding_Debt** distribution is wider for individuals with poor credit scores, indicating that those with poor scores tend to have a greater variety of outstanding debt amounts.
- **Standard Credit Score**: The distribution for individuals with standard credit scores is slightly narrower compared to the "Good" category, with more noticeable outliers.

- **Good Credit Score**: For individuals with a good credit score, the Outstanding_Debt distribution is more concentrated, with fewer outliers. This suggests that people with good credit scores generally have more manageable levels of outstanding debt.

Data Transformation and Encoding

To prepare the dataset for modeling, we will use **Ordinal Encoding** for the categorical features **"Occupation," "Payment_of_Min_Amount,"** and **"Payment_Behaviour."** This approach is chosen over label encoding or one-hot encoding for several reasons:

1. **Preserving Order**: Ordinal encoding is ideal for categorical features that have a natural order or ranking. This encoding method ensures that any inherent ordinal relationships in the data are preserved, which can potentially improve the performance of the machine learning model.
2. **Streamlined Data Handling**: Ordinal encoding reduces dimensionality compared to one-hot encoding, which creates multiple binary columns for each category. This is particularly beneficial when deploying the model, as it avoids the issue of managing a large number of input features and simplifies processing of user-entered data.

By encoding these categorical features as ordered values, we can maintain consistency and prevent unnecessary complexity in the dataset.

Train-Test Split

the **test data** does **not** include the **target** column **Credit_score**

so we will ignore it and we will work only with training set as train and test

To evaluate the model effectively, the dataset is split into a training set and a test set. This separation ensures that the model can be trained on one portion of the data and validated on another.

Scale Data

Before training the model, we ensure that the data is appropriately scaled. The **transform()** method is applied on **X_test** to ensure that the scaling parameters learned from **X_train** are consistently applied. This step prevents data leakage, ensuring that the test set is transformed in the same way as the training set.

Hyperparameter Tuning with Grid Search

To optimize the performance of the **RandomForestClassifier**, we perform a **Grid Search with Cross-Validation**. This method allows us to exhaustively search through a predefined set of hyperparameters and find the best combination for our model.

In the grid search, we include both **default** and **custom parameter values** to explore a wide range of options. The key parameters that are typically tuned in a **RandomForestClassifier** include:

- **n_estimators** (number of trees in the forest)
- **max_depth** (maximum depth of the tree)
- **min_samples_split** (minimum number of samples required to split an internal node)
- **min_samples_leaf** (minimum number of samples required to be at a leaf node)
- **max_features** (number of features to consider when looking for the best split)

By systematically testing different combinations of these hyperparameters, we aim to find the optimal configuration that maximizes model accuracy while preventing overfitting. This approach ensures the model generalizes well to unseen data, providing a robust and reliable prediction of credit scores.

```
Best Parameters: {'max_depth': None, 'max_features': 'sqrt',
'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}
```

This **Grid Search** process, combined with **Cross-Validation**, allows us to evaluate the model on different subsets of the training data, ensuring a more reliable and stable hyperparameter selection.

Results

RandomForestClassifier

```
Parameters: {'max_depth': None, 'max_features': 'sqrt',
'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}
```

test accuracy: 86%

Precision, Recall, and F1-score:

1. Class 0 (Good): Precision = 0.87, Recall = 0.89, F1 = 0.88
2. Class 1 (Standard): Precision = 0.86, Recall = 0.76, F1 = 0.81
3. Class 2 (Poor): Precision = 0.86, Recall = 0.95, F1 = 0.90

Confusion Matrix Highlights:

1. Class 0 has low misclassification compared to other classes.
2. Class 1 has significant misclassification into both Class 0 and Class 2, indicating it is harder for the model to distinguish.
3. Class 2 has minimal misclassification, showing high recall and good precision.

Strengths:

1. The Random Forest model performs well for Classes 0 and 2.
2. It achieves high precision and recall for Class 2 (Poor), which is often critical in credit scoring contexts.

Weaknesses:

1. Class 1 (Standard) is the most problematic, as its recall (0.75) is significantly lower compared to other classes.
2. Precision for Class 1 is slightly weaker, suggesting the model confuses Standard customers with Good or Poor.

XGBoost Classifier

```
xgb_classifier = XGBClassifier(
    objective='multi:softmax',  # Multi-class classification
    num_class=len(y_train.unique()),  # Number of unique classes
    random_state=42,
    eval_metric='mlogloss')
```

Accuracy: 81%

Precision, Recall, and F1-score:

1. Class 0 (Good): Precision = 0.84, Recall = 0.81, F1 = 0.82
2. Class 1 (Standard): Precision = 0.78, Recall = 0.72, F1 = 0.75
3. Class 2 (Poor): Precision = 0.91, Recall = 0.91, F1 = 0.86

Confusion Matrix Highlights:

1. Class 0 has moderate misclassification into Class 1.
2. Class 1 shows relatively high confusion, with significant misclassification into Classes 0 and 2.
3. Class 2 has strong precision and recall, indicating effective differentiation of "Poor" credit scores.

Strengths:

1. XGBoost handles Class 2 (Poor) particularly well with excellent precision and recall, similar to Random Forest.
2. Class 0 (Good) also has reasonable performance.

Weaknesses:

1. XGBoost struggles more than Random Forest with Class 1 (Standard), evidenced by lower precision, recall, and F1.
2. Overall accuracy is 5% lower than Random Forest.

Deep Learning Model

Model description{ model is a deep neural network with multiple fully connected layers, using **ReLU** activation functions, **BatchNormalization**, and **Dropout** for regularization. It outputs a 3-class classification using the **Softmax** activation. The model is compiled with the **Adam** optimizer and **sparse categorical cross-entropy** loss, aiming to predict customer credit scores with **accuracy** as the evaluation metric.}

Accuracy: 79%

Precision, Recall, and F1-score:

1. Class 0 (Good): Precision = 0.80, Recall = 0.84, F1 = 0.82
2. Class 1 (Standard): Precision = 0.78, Recall = 0.64, F1 = 0.70
3. Class 2 (Poor): Precision = 0.80, Recall = 0.91, F1 = 0.85

Confusion Matrix Highlights:

1.Class 0 (Good): Moderate misclassification into Class 1. The model performs fairly well in predicting "Good" credit scores, with a strong precision of 0.80 and recall of 0.84. 2.Class 1 (Standard): The model has challenges with Class 1, as it misclassifies a considerable number of instances into both Class 0 and Class 2. This is evident in the lower recall (0.64) and F1-score (0.70). 3.Class 2 (Poor): Class 2 ("Poor" credit scores) is predicted with high accuracy, with strong precision (0.80) and recall (0.91), indicating the model can distinguish "Poor" credit scores effectively.

Strengths:

1.Class 2 (Poor): The model is very effective in identifying "Poor" credit scores, with a precision of 0.80 and a very high recall of 0.91, which is a key strength.

2. Class 0 (Good): The performance of Class 0 is also good, with solid precision and recall scores, showing the model's ability to identify "Good" credit scores.

Weaknesses:

1.Class 1 (Standard): The model's performance on Class 1 (Standard) is weaker. It has lower precision and recall, which indicates that the model struggles with this class, misclassifying it into both Class 0 and Class 2.

Flask Web Application for Credit Score Prediction

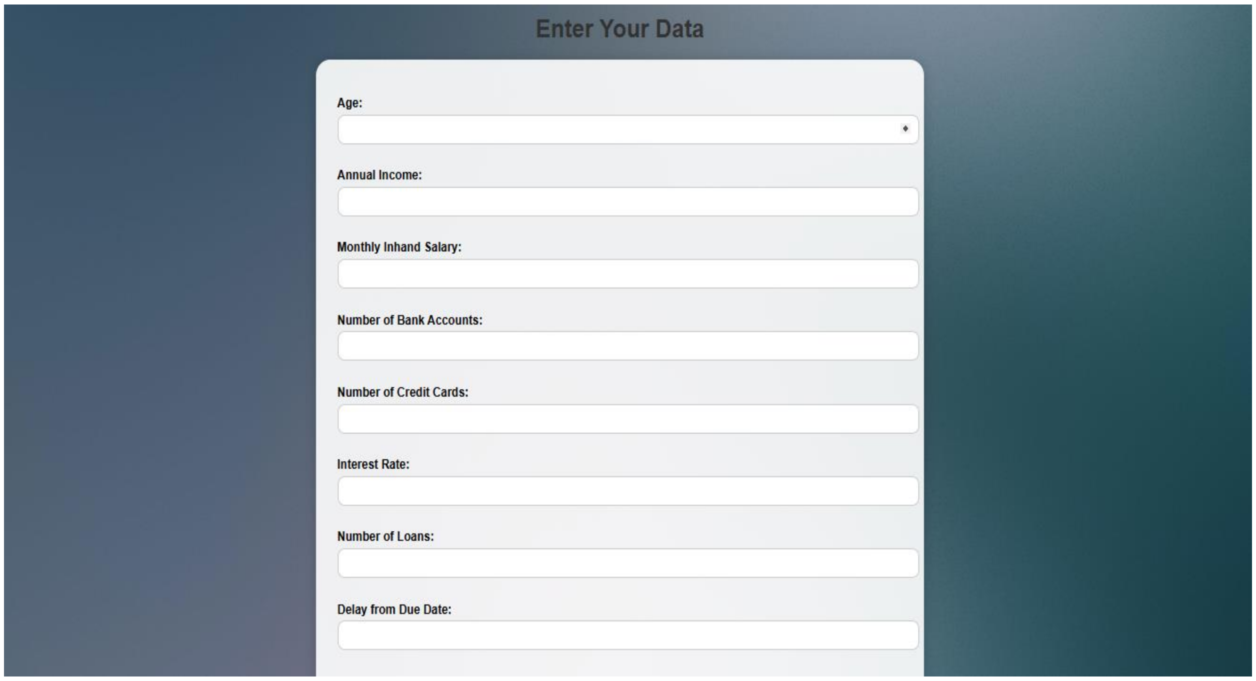
This code implements a **Flask web application** that predicts a customer's credit score based on various financial attributes using a pre-trained **XGBoost model or Random Forest model**. The app contains two routes:

1. The / route serves a form (form.html) where users input their financial data.
2. The /predict route handles the form submission, processes the inputs, and uses the model to predict the credit score.

The input features are mapped from categorical values (like occupation and payment behavior) to numerical values using predefined dictionaries. The model then makes a prediction, and the result is displayed on a new page (result.html). The application uses **pickle** to load the trained model and applies the prediction logic based on the inputs provided. The app runs locally with Flask's built-in server for testing purposes.

The Form

The form.html page contains a user-friendly form where users can input their financial details such as age, income, bank accounts, and other relevant attributes. The form uses **POST** method to send the data to the /predict route when submitted. Each input field corresponds to a feature in the model, allowing users to enter their information for credit score prediction.



Enter Your Data

Age:

Annual Income:

Monthly Inhand Salary:

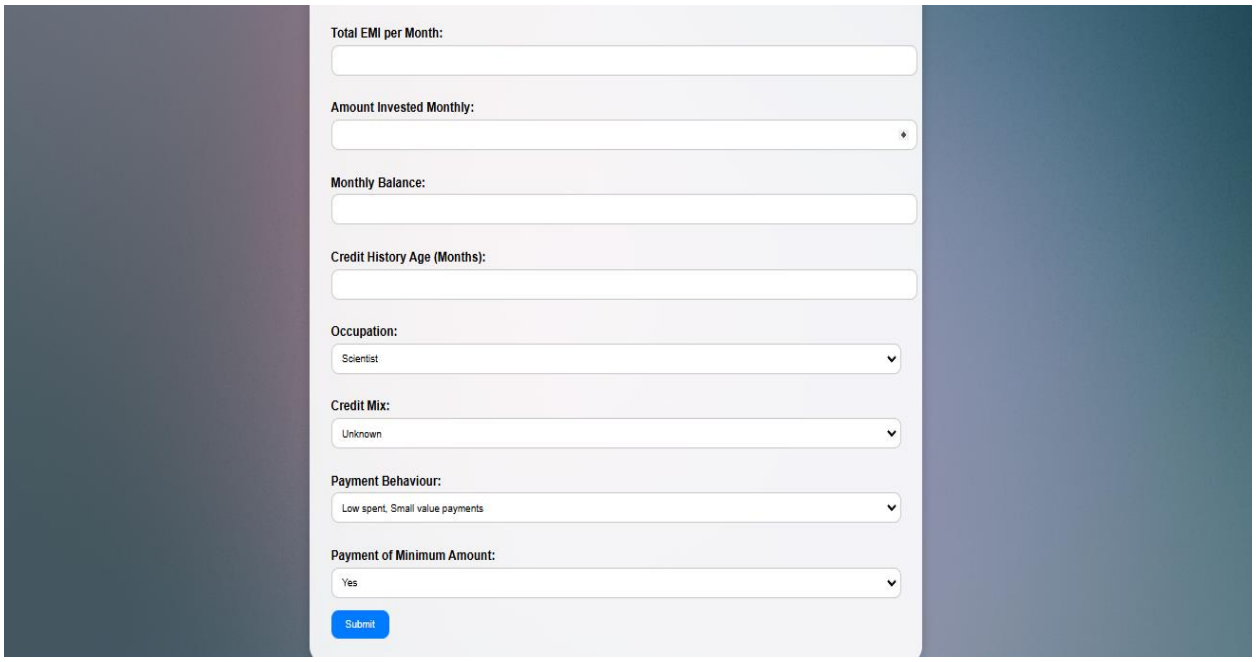
Number of Bank Accounts:

Number of Credit Cards:

Interest Rate:

Number of Loans:

Delay from Due Date:



Total EMI per Month:

Amount Invested Monthly:

Monthly Balance:

Credit History Age (Months):

Occupation: Scientist

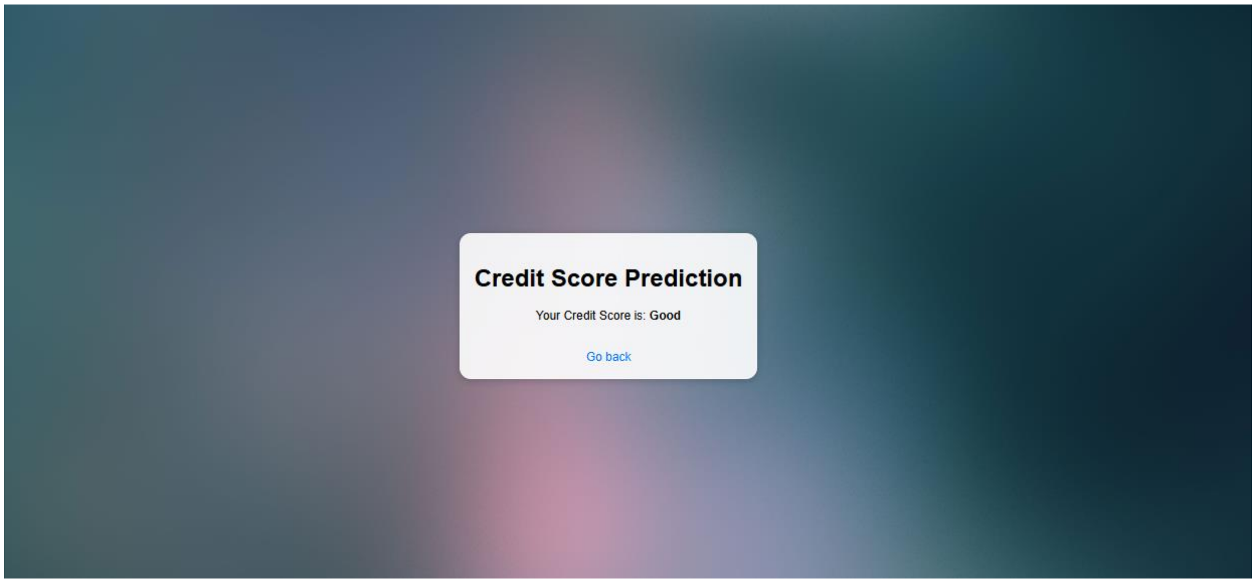
Credit Mix: Unknown

Payment Behaviour: Low spent, Small value payments

Payment of Minimum Amount: Yes

result.html

The `result.html` page displays the predicted credit score based on the user's input, showing whether the individual is classified as **Good**, **Standard**, or **Poor**. It receives the prediction result from the `/predict` route and presents it in a simple format. This page provides immediate feedback to users on their creditworthiness based on the model's prediction.


Credit Score Prediction

Your Credit Score is: **Good**