**Project Title**

**Machine Learning Pipeline for Mortgage Backed Securities Prepayment Risk**

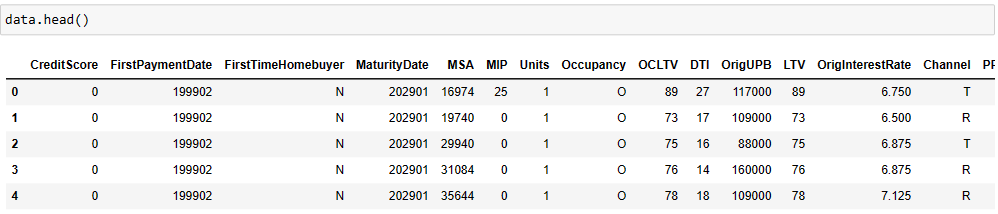
**Description**

This project aims to predict loan pre-payment likelihood based on various factors such as Equated Monthly Installment (EMI), total payment, interest amount, monthly income, current principal, and whether the person has ever been delinquent in their loan payments. The goal is to provide insights into the likelihood of pre-payment and assist in making informed decisions related to loan repayment strategies.

**Dataset**

The data dictionary provides an overview of the dataset used in this analysis. It describes the 28 columns present in the dataset, including details such as column names, data types, and a brief explanation of each column's meaning. Understanding the data dictionary is crucial for interpreting and analysing the dataset accurately.

The dataset used for this analysis contains information about borrowers and their loan details, including EMI, total payment, interest amount, monthly income, current principal, and whether they have been delinquent in their loan payments. The dataset is in a tabular format and is provided as a CSV file called LoanExport.csv.



**Pre**-**processing** **Steps**

The following steps were taken to ensure the data's integrity and quality:

* Removal of duplicated rows: Any duplicated rows were removed to eliminate any redundant data that could potentially skew the analysis.
* Handling missing values: Missing values were handled by removing rows that contained missing values in the SellerName column.
* Unique value analysis: The unique values present in each column were analyzed to identify any unusual or unexpected values.
* Handling missing and erroneous values: Any rows that contained the value X in the NumBorrowers, PropertyType, MSA, or PPM columns were removed, as these values were likely placeholders for missing or invalid data.

**Feature** **Engineering**

The following steps were taken to enhance the analysis by creating additional columns:

* CreditScoreRange: A new column, CreditScoreRange, was created by categorizing the CreditScore column into four bands based on credit score ranges.
* RepaymentRange: Another new column, RepaymentRange, was created by dividing the MonthsInRepayment column into five bands to represent different repayment periods.
* Label encoding: Categorical columns were encoded using label encoding or ordinal encoding to convert categorical values into numerical representations that can be processed by machine learning models.
* **Encoding the Data:** To prepare the categorical columns for analysis, we performed label encoding and ordinal encoding.
* from sklearn.preprocessing import LabelEncoder
* columns\_to\_encode =[
* 'FirstTimeHomebuyer',
* 'PPM'
* ]
* #Initialise the LabelEncoder
* label\_encoder = LabelEncoder()
* # Perform label encoding on the specified columns
* for column in columns\_to\_encode:
* data[column] = label\_encoder.fit\_transform(data[column])

Label encoding was applied to columns such as **'FirstTimeHomebuyer'** and **'PPM'** using the category approach. The LabelEncoder from the scikit-learn library was used for this purpose.

columns\_to\_encode = [

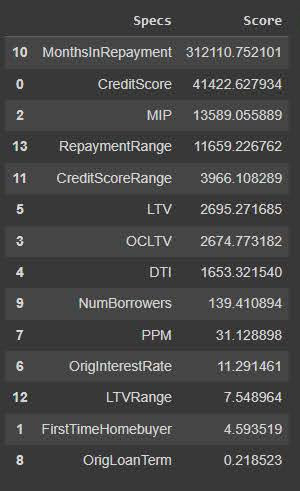
ordinal\_encoder = OrdinalEncoder(categories=category\_order)

# Encode the specified columns

data[columns\_to\_encode] = ordinal\_encoder.fit\_transform(data[columns\_to\_encode])

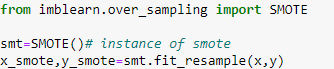
Ordinal encoding was used for columns with ordinal data, specifically 'OrigInterestRateRange' and 'RepaymentRange'. The encoding was performed using the OrdinalEncoder from scikit-learn, with specified category orders for each column.

* **Feature Selection:** Feature selection is crucial to identify the most relevant features for the model. In this project, we employed the SelectKBest algorithm with the chi-square (chi2) scoring function. The algorithm was used to rank the features based on their correlation with the target variable 'EverDelinquent'. The top 14 features were selected based on their scores, and a dataframe, featureScores, was created to provide insights into the selected features and their corresponding scores.
* bestfeatures = SelectKBest(score\_func=chi2, k=14)
* fit = bestfeatures.fit(x, y)
* dfscores = pd.DataFrame(fit.scores\_)
* dfcolumns = pd.DataFrame(x.columns)
* featureScores = pd.concat([dfcolumns, dfscores], axis=1)
* featureScores.columns = ['Specs', 'Score']
* featureScores.sort\_values(by='Score', ascending=False)



**SMOTE**

To address the class imbalance problem, the Synthetic Minority Over-sampling Technique (SMOTE) is employed. SMOTE is a widely used technique in machine learning that generates synthetic samples for the minority class to balance the class distribution. By creating synthetic samples, SMOTE helps improve model performance and prevent biased predictions.

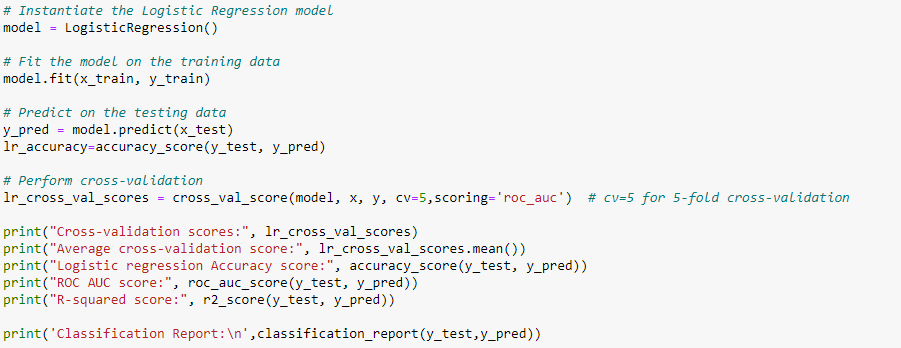


It creates synthetic samples by selecting nearest neighbors from the minority class and generating new instances along the line connecting them. This helps balance the class distribution and improve model performance. SMOTE was applied to predict Mortgage Backed Securities prepayment risk, ensuring more accurate predictions.

**Modelling**

The model for predicting Mortgage Backed Securities prepayment risk utilizes various machine learning algorithms, including Logistic Regression, Random Forest Classifier, XGBoost Classifier, and K-Nearest Neighbors (KNN). These algorithms classify whether a loan is likely to become delinquent (EverDelinquent = 1) or not (EverDelinquent = 0).

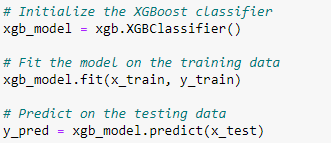
Logistic Regression is a binary classification algorithm that estimates coefficients for each feature to determine their impact on the target variable. It predicts the probability of delinquency based on these coefficients and classifies the loan accordingly.



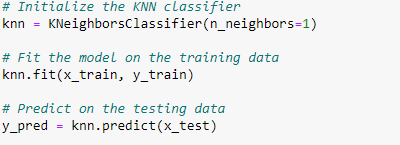
Random Forest Classifier is an ensemble learning method that combines multiple decision trees to make predictions. It creates collections of decision trees by training on random subsets of data and features, improving accuracy and reducing overfitting.



XGBoost Classifier is an ensemble technique based on gradient boosting, which sequentially adds decision trees to the ensemble. It handles missing values, captures feature interactions, and applies regularization to prevent overfitting, making it effective for classification tasks.



K-Nearest Neighbors (KNN) is a simple yet powerful classification algorithm that assigns a data point to a class based on its k-nearest neighbors in the feature space. KNN does not assume any underlying data distribution and is non-parametric.Model Evaluation

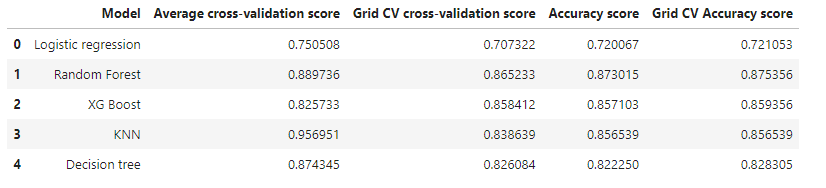


The models are evaluated using various performance metrics, including accuracy, cross-validation scores, and ROC AUC. The performance evaluation provides insights into the accuracy and generalization capabilities of each model.

Based on the performance metrics, the XGBoost Classifier is identified as the top-performing model for predicting MBS prepayment risk due to its high accuracy, excellent generalization, and advanced ensemble technique.

**Model’s Performance**

The performance metrics of different models as follows:

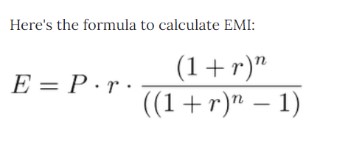


All models perform well in predicting Ever Delinquent status of Mortgage Backed Securities. Random forest model show higher cross-validation and accuracy scores, indicating superior performance compared to Logistic Regression and other models . However, the absence of ROC AUC scores makes it challenging to assess the models' discrimination ability fully. Considering multiple performance metrics is crucial for a comprehensive evaluation of classification models.

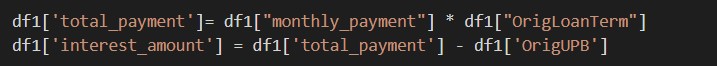
**Data** **Preparation** **for** **pipelining**

To build the pre-payment predictor, the following steps were performed on the dataset:

Calculate EMI Equated Monthly Installment - EMI for short - is the amount payable every month to the bank or any other financial institution until the loan amount is fully paid off. It consists of the interest on loan as well as part of the principal amount to be repaid.

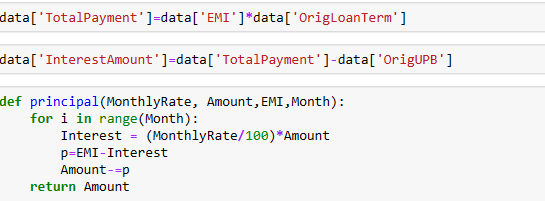


Calculation of Total Payment and Interest Amount: The total payment was calculated by multiplying the EMI with the loan tenure. The interest amount was derived by subtracting the principal amount from the total payment.



Calculation of Monthly Income: The monthly income of borrowers was estimated by dividing the DTI (debt-to-income ratio) with the monthly debt (EMI). This provides an approximation of the borrowers' monthly income based on their loan obligations.

Calculation of Current Principal: The remaining principal amount was calculated based on the number of months in repayment, the monthly interest rate, the original principal amount, and the EMI. This reflects the outstanding loan amount after deducting the principal paid during the repayment period.

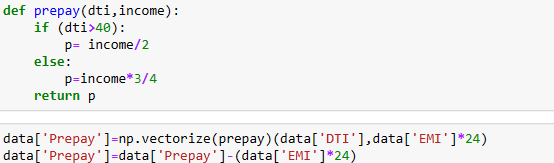


**Pre**-**payment** **Calculation**

To simulate pre-payment, the following assumptions were made:

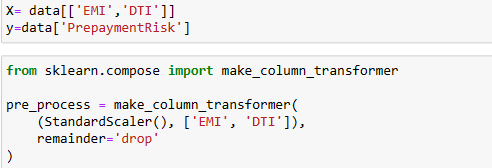
Savings Calculation: Borrowers with a DTI below 40% are assumed to save 50% of their income, inclusive of the EMI, for a period of two years. Borrowers with a DTI above 40% are assumed to save 75% of their income.

Pre-payment Amount: The pre-payment amount is calculated by subtracting the total savings from the outstanding loan amount after two years. This represents the amount borrowers are expected to pre-pay at the end of the designated period.



**Data splitting and pipeline model implementation**

We did preprocessing part for the pipelinemodel so that it only allows the required columns to be used in the pipeline, that is only they are allowed to passthrough the pipeline using the column transform method.

****



After pre-processing the data and separating the features and targets, we divided our data into training and testing sets using the train\_test\_split function from the scikit-learn library. The input features are denoted as "X" and the corresponding target variables or labels are denoted as "y". We allocated 20% of the data for testing and used the remaining 80% for training.



To evaluate the performance of the Regression model, we calculated several evaluation metrics. The Mean Squared Error (MSE) measures the average squared difference between the actual target values (y\_test) and the predicted values (y\_pred). The Mean Absolute Error (MAE) measures the average absolute difference between the actual target values and the predicted values. The R-squared (R2) score indicates the proportion of the variance in the target variable that is explained by the linear regression model.

Finally, we printed the coefficients, intercept, and evaluation metrics of the Linear Regression model. The coefficients represent the magnitude and direction of the impact of each input feature on the target variable, while the intercept represents the model's predicted target value when all input features are zero. This information provides insights into the model's performance and the relationship between the features and the target variable.

**Linear Regression**

To implement Linear Regression, we followed these steps:

First, we created a Linear Regression model by importing the LinearRegression class from the scikit-learn library. This model represents a linear regression model. Next, we fitted the model using the training data (X\_train and y\_train) by calling the fit() method. This step allowed the model to learn the coefficients and intercept, which define the linear relationship between the input features and the target variable.

After training the model, we made predictions on the test data (X\_test) by calling the predict() method. The predicted values were assigned to the variable "y\_pred".



**Ridge Regression**

To implement Ridge Regression, we followed similar steps as in Linear Regression:



Firstly, we created a Ridge Regression model by importing the Ridge class from the scikit-learn library. We also specified a specific alpha value, which controls the strength of L2 regularization. This regularization helps balance between fitting the training data and preventing overfitting.

Next, we fitted the Ridge regression model using the training data (X\_train and y\_train) by calling the fit() method. This step optimized the coefficients of the model by minimizing the sum of squared residuals between the predicted values and the actual target values, taking into account the additional penalty term introduced by the regularization. Once the model was trained, we made predictions on the test data (X\_test) by calling the predict() method. The predicted values were stored in the variable "y\_pred".

**Random Forest Classifier**

We also implemented a Random Forest Classifier using the following approach:



Firstly, we imported the necessary modules from scikit-learn and imbalanced-learn to enable Random Forest classification. Then, we split the data into training and testing sets using the train\_test\_split function. This step allowed us to have separate datasets to train and evaluate our Random Forest Classifier.

Next, we created a pipeline using ImbPipeline from the imbalanced-learn library. The pipeline consisted of a pre-processing step, which could include transformations such as scaling or encoding, and a Random Forest Classifier as the model. The pipeline helps to streamline the workflow and ensures consistent application of pre-processing steps to the data.

After creating the pipeline, we fitted it with the training data. This step involved training the Random Forest Classifier model on the training set using the fit() method. The pipeline automatically applied the pre-processing steps before fitting the model.

**Results**

The accuracy we obtained for each model was as follows:

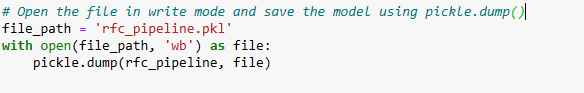
- Linear Regression: 74%

- Ridge Regression: 74% (negligible difference from Linear Regression)

- Random Forest Classifier: 87%

**Creating Pickles for the pipeline models**

We need Pipeline models for Randomforest model of Everdelinquent part and the linear regression pipeline for prepayment risk predictor. So we use pickle library and picle it and store them to use for the deployment



We can load the file whenever needed



**Deployment of the model**

This code implements a FastAPI web application for predicting loan eligibility based on various input parameters. The application uses a machine learning model trained on loan data. Here is an overview of the code:

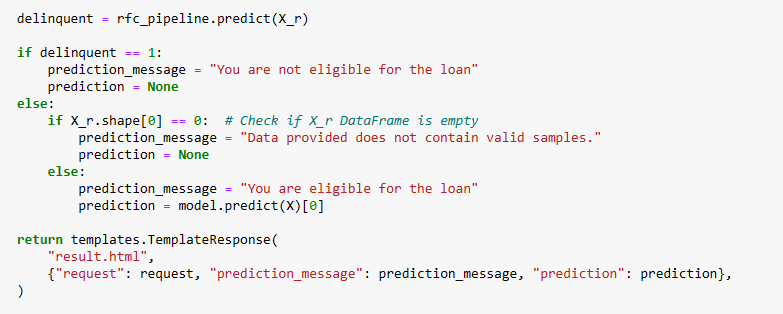
1. Importing Libraries: The necessary libraries and modules are imported, including uvicorn for running the FastAPI application, pandas for data manipulation, numpy for array operations, and FastAPI and Jinja2Templates for building the web interface.

2. Loading the Model: The pre-trained machine learning model is loaded using the pickle module.

3. Defining Endpoints: Two endpoints are defined. The first endpoint ("/") renders an HTML form using a Jinja2 template to collect user input. The second endpoint ("/predict") handles the form submission and predicts loan eligibility based on the provided input.

4. Handling Form Submission: The "/predict" endpoint retrieves the form data submitted by the user and converts it into a pandas DataFrame. Some data preprocessing steps are performed, such as label encoding categorical variables and creating derived features.

5. Predicting Loan Eligibility: The loaded machine learning model is used to predict loan eligibility based on the preprocessed input data. If the loan is predicted as delinquent, a message indicating ineligibility is displayed. Otherwise, if the input data is valid and the model predicts eligibility, an appropriate message is displayed along with the predicted loan outcome.

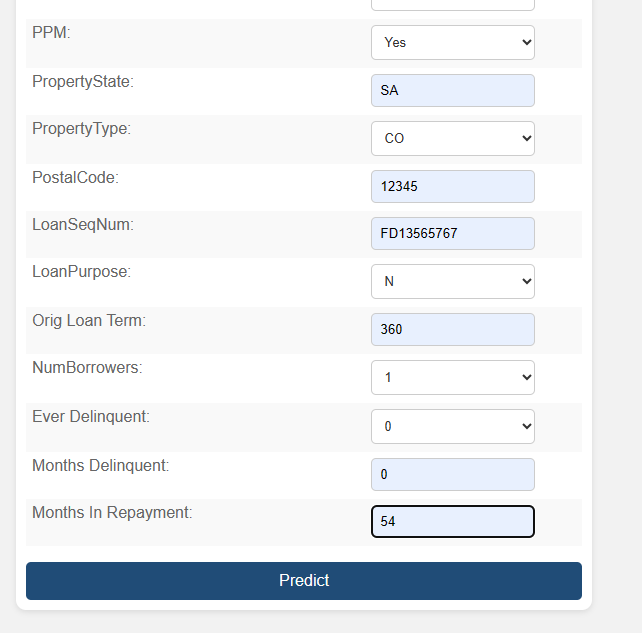


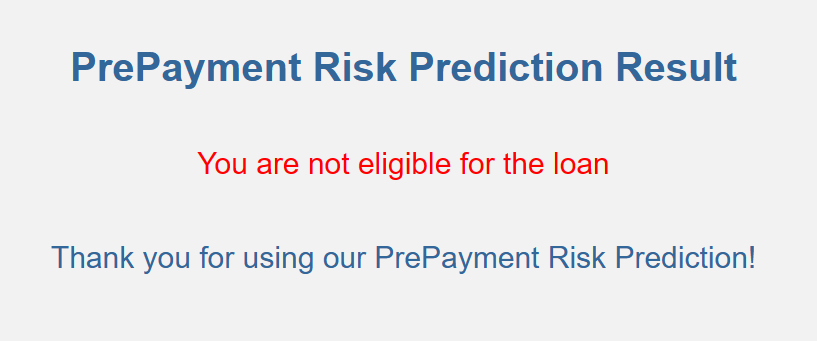
6. Running the Application: The FastAPI application is run using the uvicorn server on port 8000.

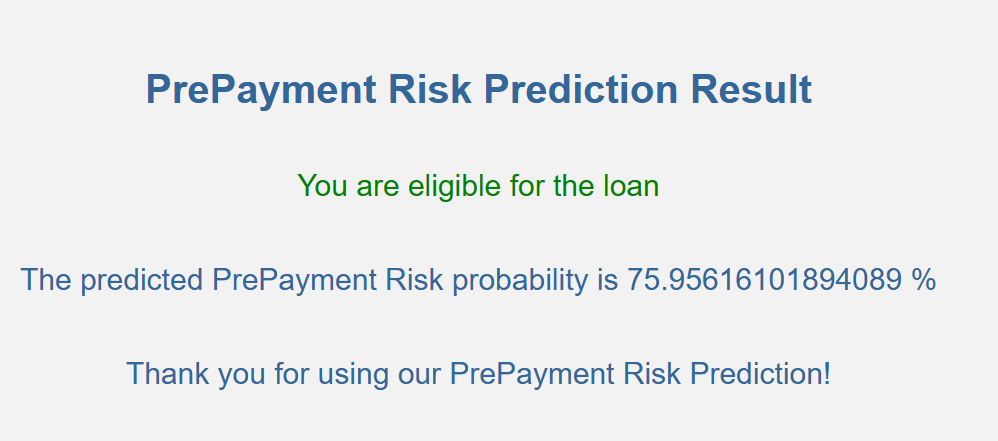
Additionally, the code includes the use of ngrok to create a tunnel and expose the application to the internet for testing purposes.

It's worth noting that some parts of the code, such as the file paths for loading the model and templates, should be adjusted based on the actual directory structure of the project.

Overall, this code provides a functional implementation of a FastAPI web application for loan eligibility prediction, allowing users to input loan details and obtain predictions in real-time.







**Conclusion**

The loan pre-payment predictor offers valuable insights into borrowers' pre-payment behavior, allowing financial institutions and lenders to optimize loan repayment strategies. By considering factors such as EMI, loan tenure, DTI, and delinquency history, lenders can identify borrowers who are likely to pre-pay their loans and tailor their services accordingly.