Data mining project on Microsoft azure cloud

Prepared by:
BOUKAYOUA LOUBNA.
EL BOUTAHERI NAJMA.

framed by:
Professor ROUTAIB HAYAT.

Year: 2024/2025

Abstract:

In this project, we present a comprehensive approach to delinquency status and prepayment prediction using advanced data mining techniques. The primary objective was to leverage historical loan data to develop predictive models capable of identifying borrowers at risk of delinquency and forecasting prepayment behaviors.

The workflow involved data preprocessing, exploratory data analysis (EDA), feature engineering, and model development. Utilizing a combination of logistic regression, decision tree classifiers, and gradient boosting models, we achieved robust performance metrics, with the delinquency prediction model attaining an accuracy of 98% and the prepayment model demonstrating strong generalization on test data.

The deployment strategy integrates Azure services, including Azure Blob Storage for data management, Azure Databricks for model development and orchestration, and Azure Machine Learning for automating workflows and monitoring models in production. The project also incorporates a Flask-based application for real-time prediction and Power BI for visualizing insights, enabling stakeholders to make data-driven decisions.

This report outlines the methodology, challenges encountered, and future directions, highlighting the practical applications of predictive analytics in the financial domain to mitigate risks and optimize loan servicing strategies.

Introduction:

The financial sector faces significant challenges in managing loan portfolios, particularly in predicting **delinquency** and **prepayment** behaviors among borrowers.

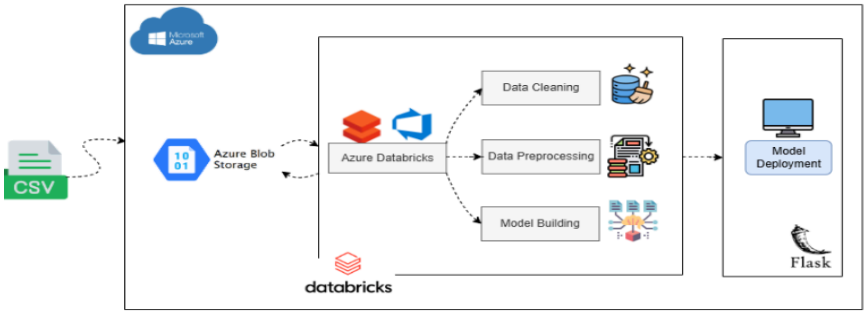
Loan delinquency, the failure to make payments on time, poses a substantial risk to lenders by impacting cash flow and increasing credit losses. Conversely, prepayment—when borrowers pay off loans earlier than expected—affects interest income and disrupts revenue forecasts. Accurate prediction of these phenomena is critical for risk mitigation, financial planning, and operational efficiency.

This project focuses on leveraging data mining techniques to address these challenges by building predictive models for delinquency status and prepayment behaviors. Using a rich dataset of historical loan data, we performed data preprocessing, exploratory data analysis (EDA), and feature engineering to extract meaningful insights. Predictive modeling techniques, including **logistic regression**, **decision trees**, and **gradient boosting models**, were employed to build accurate and interpretable models.

The project architecture integrates Microsoft Azure services to ensure scalability, reliability, and ease of deployment. **Azure Blob Storage** is used for centralized data management, while **Azure Databricks** facilitates data processing and model orchestration. **Azure Machine Learning** automates the pipeline and monitors model performance, ensuring the solution is production-ready. Additionally, a Flask-based web application provides real-time prediction capabilities, and **Power BI** dashboards deliver actionable insights to stakeholders.

This report aims to document the methodology, tools, and outcomes of this project, providing a comprehensive guide to implementing predictive analytics in loan portfolio management. The findings demonstrate the potential of data mining to enhance decision-making, optimize resource allocation, and improve financial stability in the lending industry.

Architecture:



Project components:

1. Data Storage:

Azure Blob Storage: This component stores raw data files, which are the input for the machine learning workflow.

2. Data Processing and Model Development:

Azure Databricks: we used this platform that is used for big data and machine learning . It supports:

- Data Cleaning: Preparing the data by removing inconsistencies and errors.
- Data Preprocessing: Transforming raw data into a format suitable for modeling.
- Model Building: Training and validating machine learning models.

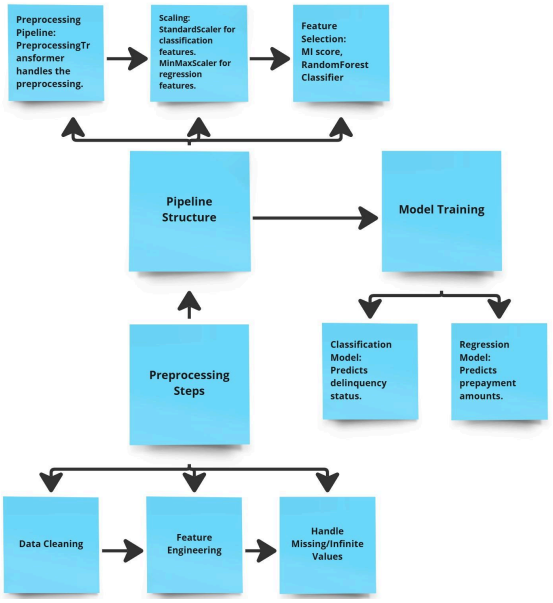
3. Model Deployment:

This involves deploying the trained model to a live environment using a virtual machine provided by Microsoft Azure. Flask will be used for hosting the model as an API.

4. Visualization:

Tools like Power BI are used for creating dashboards and visualizations to interpret and present the results effectively.

Machine learning model building steps:



1. **Preprocessing Steps**

Data Cleaning:

Remove invalid or inconsistent rows('FirstTimeHomebuyer' ≠ 'X', 'NumBorrowers' ≠ 'X ').

Map 'FirstTimeHomebuyer' to binary (Y → 1, N → 0).

Feature Engineering:

Frequency encoding:

'Channel' → Channel_Frequency. Categorical encoding:

'LTV_range', 'OCLTV_range', 'Credit_range', etc.

Binning: 'DTI' into categories (Low, Moderate, etc.).

Mathematical features: 'MonthlyPayment', 'TotalAmount', 'InterestAmount', 'Prepayment'.

Handle Missing/Infinite Values:

Replace infinities and drop rows with NaN.

2. Pipeline Structure

Preprocessing Pipeline:

PreprocessingTransformer handles the preprocessing.

Scaling:

StandardScaler: For classification features.

MinMaxScaler: For regression features.

Feature Selection:

- ➔ **ClassificationFeatures**: 'MonthsDelinquent','FICO_Category_Encoded', etc.
- ➔ **RegressionFeatures**:
- ➔ 'OrigUPB','MonthlyPayment','Interest Amount', etc.

3. Model Training

Classification Model:

Input: Preprocessed classification features.

Output: Predicts delinquency status.

Metrics: Accuracy, ROC-AUC, F1-score.

Regression Model:

Input: Preprocessed regression features (filtered by non-delinquent status).

Output: Predicts prepayment amounts.

Metrics: RMSE, R^2.

4. Prediction Pipeline

Step 1: Use the classification model to predict delinquency status.

Step 2: For non-delinquent cases, pass the data to the regression model to predict prepayment amounts.

So these are the steps that we followed during this project.

About the dataset:

1. Data collection:

The data was collected from the github repository.

#	Column	Non-Null Count	Dtype
0	CreditScore	291451 non-null	int64
1	FirstPaymentDate	291451 non-null	int64
2	FirstTimeHomebuyer	291451 non-null	object
3	MaturityDate	291451 non-null	int64
4	MSA	291451 non-null	object
5	MIP	291451 non-null	int64
6	Units	291451 non-null	int64
7	Occupancy	291451 non-null	object
8	OCLTV	291451 non-null	int64
9	DTI	291451 non-null	int64
10	OrigUPB	291451 non-null	int64
11	LTV	291451 non-null	int64
12	OrigInterestRate	291451 non-null	float64
13	Channel	291451 non-null	object
14	PPM	291451 non-null	object
15	ProductType	291451 non-null	object
16	PropertyState	291451 non-null	object
17	PropertyType	291451 non-null	object
18	PostalCode	291451 non-null	object
19	LoanSeqNum	291451 non-null	object
20	LoanPurpose	291451 non-null	object
21	OrigLoanTerm	291451 non-null	int64
22	NumBorrowers	291451 non-null	object
23	SellerName	266457 non-null	object

- ➔ **PPM:** Prepayment Penalty Mortgage, indicating whether there is a penalty for paying off the loan early.

24	ServicerName	291451 non-null	object
25	EverDelinquent	291451 non-null	int64
26	MonthsDelinquent	291451 non-null	int64
27	MonthsInRepayment	291451 non-null	int64

dtypes: float64(1), int64(13), object(14)
memory usage: 62.3+ MB

The data contains several columns, This is a brief explanation of each column:

- ➔ **CreditScore:** The credit score of the borrower, which indicates their creditworthiness.
- ➔ **FirstPaymentDate:** The date on which the first payment of the loan is due.
- ➔ **FirstTimeHomebuyer:** A flag indicating whether the borrower is a first-time homebuyer.
- ➔ **MaturityDate:** The date when the loan is scheduled to be paid off in full.
- ➔ **MSA:** Metropolitan Statistical Area, a geographical region with a relatively high population density and economic ties.
- ➔ **MIP:** Mortgage Insurance Premium, a type of insurance policy that protects the lender in case the borrower defaults on the loan.
- ➔ **Units:** The number of units in the property (e.g., single-family home, duplex, etc.).
- ➔ Occupancy: The occupancy status of the property (e.g., owner-occupied, investment property, second home).
- ➔ **OCLTV:** Original Combined Loan-to-Value ratio, which includes the balance of any other loans secured by the same property.
- ➔ **DTI:** Debt-to-Income ratio, which compares the borrower's total monthly debt payments to their gross monthly income.
- ➔ **OrigUPB:** Original Unpaid Principal Balance, the initial loan amount that the borrower has to repay.
- ➔ **LTV:** Loan-to-Value ratio, which compares the loan amount to the appraised value of the property. OrigInterestRate: Original interest rate of the loan at the time of origination.
- ➔ **Channel:** The origination channel through which the loan was processed (e.g., retail, broker, correspondent).meaningful variation for modeling.
- ➔ **LoanSeqNum:** A unique identifier

for each loan, which does not add any predictive value, was also dropped.

- ➔ **ProductType:** The type of mortgage product (e.g., fixed-rate, adjustable-rate).
- ➔ **PropertyState:** The state where the property is located.
- ➔ **PropertyType:** The type of property securing the loan (e.g., single-family home, condominium, multi-family home).
- ➔ **PostalCode:** The postal code of the property's location.
- ➔ **LoanSeqNum:** Loan Sequence Number, a unique identifier for the loan.
- ➔ **LoanPurpose:** The purpose of the loan (e.g., purchase, refinance).
- ➔ **OrigLoanTerm:** The original term of the loan in months.
- ➔ **NumBorrowers:** The number of borrowers on the loan.
- ➔ **SellerName:** The name of the entity that sold the loan to the investor.
- ➔ **ServicerName:** The name of the entity that services the loan.
- ➔ **EverDelinquent:** Indicates whether the borrower has ever been delinquent on loan payments.
- ➔ **MonthsDelinquent:** The number of months the borrower has been delinquent on payments.
- ➔ **MonthsInRepayment:** The number of months the loan has been in repayment.

Data Preprocessing:

1. Cleaning:

Data cleaning is an essential step in the data preprocessing pipeline. This step ensures that the dataset is free from inconsistencies, missing values, and outliers that could negatively impact model performance.

a. Dropping Columns with Irrelevant or Constant Values

Unique values check: We first inspect the dataset for columns with a single unique value or irrelevant data.

Dropped columns:

- ➔ **ProductType:** This column has only one unique value, so it was dropped as it does not provide any predictive value.

b. Handling Missing Values

Replacing placeholder values: The dataset contained some placeholder values ('X') which were replaced with NaN to be handled correctly during the imputation process.

Column conversions: Several columns such as PostalCode, NumBorrowers, and MSA were converted to numeric types, forcing any invalid entries to NaN where necessary.

Missing value imputation:

- ➔ Numerical columns (PostalCode, MSA, NumBorrowers): Missing values were imputed using the mean strategy.
- ➔ Categorical columns (FirstTimeHomebuyer, PPM, SellerName): Missing values were imputed using the most frequent strategy.

c. Identifying and Removing Duplicates

Duplicate row detection:

The dataset was checked for duplicate rows.

Duplicates removal: Any duplicate rows found were dropped to ensure no redundancy in the data.

d. Handling Zero Values in Important Features

DTI (Debt-to-Income Ratio):

Rows where the DTI value was zero were identified and examined. These zero values were considered invalid and replaced with NaN.

To fill the missing values in DTI, the median strategy was used to prevent the influence of outliers.

2. Final Cleaned Dataset Overview

After data cleaning, the dataset was left with no missing values, no duplicate rows.

The following summarizes the cleaned dataset:

All numerical columns have been handled for missing values and outliers, and categorical columns have been imputed appropriately.

The cleaned dataset is now ready for further steps such as feature engineering and model building.

Following the data cleaning process, we proceed with various analyses to understand the distribution, relationships, and interactions among features in the dataset.

Exploratory Data Analysis (EDA):

1. Univariate Analysis:

Univariate analysis involves examining each feature independently to understand its distribution and summary statistics.

a. Numerical Features

Objective: Analyze the distribution of each numerical feature to detect patterns, skewness, or outliers.

Approach:

Histograms: Plotted to visualize the distribution of values for each numerical feature.

Kernel Density Plots: Used to estimate the probability density function of the data, providing a smoother alternative to histograms.

Summary Statistics: Key statistics such as mean, median, standard deviation, and range are computed for each feature to get a better understanding of the central tendency and spread.

Histogram of CreditScore: Displays the frequency distribution of credit scores in the dataset.

KDE of OrigInterestRate: Smoother representation of the interest rate distribution, highlighting skewness.

b. Categorical Features

Objective: Investigate the frequency distribution of categorical variables to understand their levels and balance.

Approach:

Bar plots: Display the count of each category in the dataset for features like PropertyType, Channel, etc.

Pie charts: Used for features with fewer categories, providing a visual breakdown of the proportions.

2. Bivariate Data Analysis:

Bivariate analysis helps to explore the relationships between pairs of features, providing insights into how two variables interact.

a. Numerical vs Numerical

Objective: Examine the relationship between pairs of numerical features to detect any linear or non-linear patterns.

Approach:

Scatter plots: Visualize the relationship between two numerical features. For instance, the relationship between OCLTV and DTI.

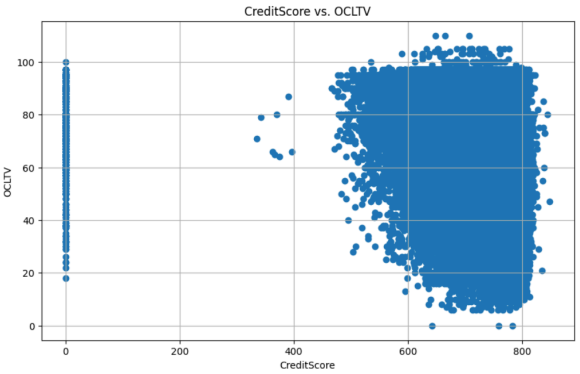
Correlation coefficients: Compute Pearson or Spearman correlations to quantify the strength and direction of relationships between pairs of features.

Example calculation between:
DTI and LTV

```
Pearson correlation: 0.12157191541995063
Spearman correlation: 0.14072607007533594
ANOVA F-value: 50.16026163240246, p-value: 0.0
```

Example:

Scatter plot of Credit Score vs OCLTV: A visual depiction of the relationship between **Credit Score** ratios and **OCLTV**.



Correlation matrix: Displays the correlation coefficients between all numerical features.

b. Numerical vs Categorical

Objective: Explore how numerical features vary across different categories.

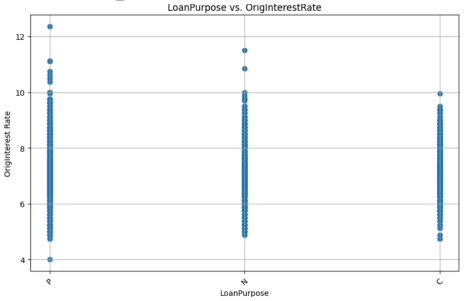
Approach:

Box plots: Display the distribution of numerical features grouped by categorical variables (e.g., OrigInterestRate distribution across LoanPurpose categories).

Violin plots: Combine box plot and KDE to show both the distribution and summary statistics.

Example:

Box plot of LoanPurpose vs OriginIntrestRate: Visualizes how **OriginIntrestRate** differs between different **LoanPurpose** values.



understanding of the structure of the dataset

and the relationships between features. These insights will guide us in the subsequent steps of feature engineering, model building, and evaluation.

3. Multivariate Analysis:

Multivariate analysis investigates the relationships and interactions between more than two features simultaneously. This is useful for detecting deeper patterns and dependencies in the data.

a. Correlation Structure

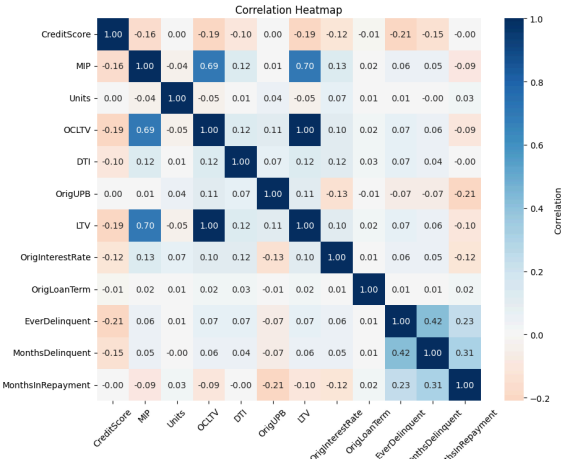
Objective: Visualize the correlation between all numerical features at once to detect highly correlated features that might need attention (e.g., multicollinearity).

Approach:

Heatmap of Correlation Matrix: A heatmap is used to visualize the pairwise correlations between numerical features, with color intensity representing the strength of correlation.

Example:

Heatmap of correlations: Highlights strong positive correlations between LTV and OCLTV and strong negative correlations between OrigInterestRate and CreditScore.



b. Dimensionality Reduction

Objective: Reduce the dimensionality of the dataset to uncover important combinations of features, simplify the data, and reduce redundancy.

Approach:

Principal Component Analysis (PCA): PCA is applied to the numerical features to capture the most important directions of variance in the data, allowing us to reduce the number of dimensions while preserving as much information as possible.

Feature Engineering:

1. Feature Extraction using Correlation Matrix

Before creating new features, we begin by analyzing the existing relationships between numerical variables using a correlation matrix. This helps us identify highly correlated features and select the most impactful ones for feature engineering. the LTV and LCTV are the features highly correlated.

Correlation Matrix:

A correlation matrix was calculated to identify strong relationships between numerical features.

Features with high correlation values (e.g., greater than 0.9) are considered for removal or transformation to avoid redundancy and multicollinearity in the model.

2. Creation of New Features

To enhance the dataset and provide more informative inputs to machine learning models, several new features were created based on the existing data. Below are the steps for feature engineering:

Credit Range (Credit_range):

The CreditScore column is used to categorize credit scores into four categories: 'Poor', 'Fair', 'Good', and 'Excellent'. This helps transform continuous credit scores into a more interpretable categorical feature.

Logic:

- ➔ 'Poor' for scores less than 650
- ➔ 'Fair' for scores between 650 and 700
- ➔ 'Good' for scores between 700 and 750
- ➔ 'Excellent' for scores above 750

LTV Range (LTV_range):

The LTV (Loan-to-Value) column is categorized into three ranges: 'Low', 'Medium', and 'High', which allows better grouping of loans based on their risk.

Logic:

- ➔ 'Low' for values below 25
- ➔ 'Medium' for values between 25 and 50
- ➔ 'High' for values above 50

Repayment Range (Repay_range):

Based on the MonthsInRepayment column, the number of months a borrower has been

3. Verifying New Features:

After creating the new features, the following checks were performed:

Consistency: Ensuring that the newly engineered features make logical sense and

align with the underlying data.
Summary Statistics: Descriptive statistics are computed to verify the distribution and ranges of the new features.
Initial Visualizations: Basic plots and summaries (e.g., histograms, box plots) were created to visualize these new features and their potential impact on the analysis.
By performing a thorough analysis at univariate, bivariate, and multivariate levels, we have gained a deep
repaying their loan is categorized into five different ranges: '0-4 years', '4-8 years', '8-12 years', '12-16 years', and '16+ years'.
This helps in understanding the repayment period and its relation to other factors.
First Time Home Buyer (isFirstTime):
A new binary feature called isFirstTime is created directly from the FirstTimeHomebuyer column to indicate whether the borrower is purchasing their first home.
Equated Monthly Installment (EMI):
The Equated Monthly Installment (EMI) is calculated based on the original loan balance (OrigUPB), interest rate (OrigInterestRate), and loan term (OrigLoanTerm). This feature provides insight into the borrower's monthly payment burden.
Total Payment:
Using the calculated EMI, the total payment over the loan term is computed. This feature is crucial to understanding the total amount a borrower would have paid by the end of the loan term.
Interest Amount:
The total interest amount paid over the loan term is derived by subtracting the original loan balance (OrigUPB) from the total payment. This provides insight into the cost of borrowing for each loan.
Monthly Income:
The borrower's estimated monthly income is calculated using the Debt-to-Income ratio (DTI) and EMI. This allows us to understand the borrower's ability to handle loan repayments based on their income.
Current Principal (cur_principal):
The current loan principal is computed based on the number of months in repayment, initial loan amount, and interest rate. This feature helps track how much of the loan principal remains to be paid.
Prepayment:
The prepayment feature estimates the borrower's ability to make additional payments toward the loan. This is based on their income and debt-to-income ratio. Prepayments can significantly reduce the loan's interest burden over time.
Each of these features is designed to improve the predictive power of the model by incorporating additional borrower behavior and financial insights.

Data Encoding:

In this project, various encoding techniques were applied to handle categorical features in the dataset. Different encoding methods were used based on the type and cardinality of the categorical variables.

1. Label Encoding

Label Encoding was applied to binary and ordinal features. Label encoding assigns a unique integer value to each category, maintaining the ordinal relationship if one exists.

PPM: This binary feature (Prepayment Penalty Mortgage) was label-encoded, assigning 0 or 1 to its values.

2. One-Hot Encoding

One-Hot Encoding was used for nominal categorical variables with fewer unique values. This method creates new binary columns for each category of a feature.

The following columns were one-hot encoded for both regression (X_reg) and classification (X_class) tasks:

- ➔ **Occupancy:** Categories representing the occupancy status of the property.
- ➔ Channel: Channels through which the loan was obtained.
- ➔ **PropertyType:** The type of property being financed (e.g., single-family home, multi-family home).
- ➔ **LoanPurpose:** The purpose of the loan (e.g., purchase, refinance).

After applying one-hot encoding, each category of these columns is represented by a new column, where a value of 1 indicates the presence of the category, and 0 indicates its absence.

3. Target Encoding

For high-cardinality categorical features,
Target Encoding was employed. This encoding technique replaces each category with the average value of the target variable for that category.

PropertyState: This feature has a large number of unique values (U.S. states), and target encoding was applied to reduce the dimensionality while preserving the relationship between the categories and the target variable (Y_class for the classification task).

4. Ordinal Encoding

For ordinal features, where the categories have an inherent order, Ordinal Encoding was used. In this method, each category is assigned a numerical value based on its order.

The following features were ordinal-encoded:

- ➔ **Credit_range:** Categories based on credit score (Poor, Fair, Good, Excellent).
- ➔ **LTV_range:** Categories based on the loan-to-value ratio (Low, Medium, High).
- ➔ **Repay_range:** Categories based on the number of years in repayment (0-4 years, 4-8 years, 8-12 years, 12-16 years, 16+ years).

These ordinal encodings ensure that the machine learning algorithms can understand the inherent ranking of the categories.

After applying all encoding techniques, the transformed datasets (X_reg for regression and X_class for classification) were verified by displaying the first few rows. This step ensures that the encodings were applied correctly and that the categorical variables are now in a format suitable for machine learning models.

Feature Selection Using Mutual Information (MI):

Mutual Information (MI) is a measure of the dependency between variables. It quantifies how much information the presence or absence of one feature gives about the target variable. In this project, MI was used for both classification and regression tasks to evaluate the importance of features.

1. Mutual Information for Classification:

For the classification task, the **mutual_info_classif** function from the sklearn library was used to compute the mutual information between the features and the target variable (EverDelinquent). This helps us understand which features are most informative for predicting whether a loan will become delinquent.

2. Mutual Information for Regression:

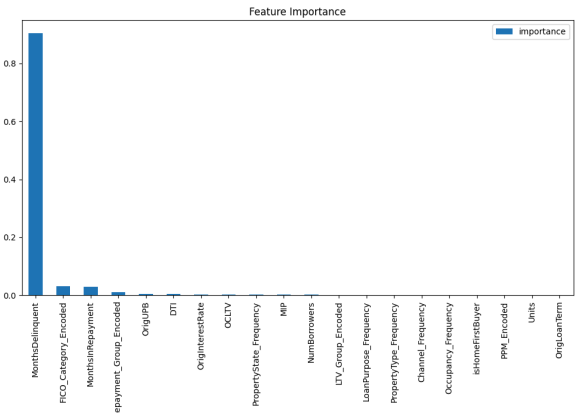
Similarly, for the regression task, the **mutual_info_regression** function was applied to assess the relationship between the features and the target variable (prepayment). This provides insights into which features are most relevant for predicting prepayment amounts.

3. Average Mutual Information:

To compare feature importance across both classification and regression tasks, the mutual information scores were averaged. This average helps identify features that are important for both tasks. The results were stored in a DataFrame, showing mutual information for each feature.

4. Feature Importance Ranking:

The features were ranked based on their average mutual information score, and a bar plot was created to visualize the importance of the top features. This provides a clear understanding of which features contribute most to both tasks.



- Mutual Information for Classification explains how MI was used to understand the relationship between features and the classification target (EverDelinquent).
- Mutual Information for Regression covers the application of MI for regression with the target variable (prepayment).
- Average Mutual Information explains the calculation of the average mutual information score across both tasks to rank features.

Feature Importance Ranking describes the sorting of features by their importance.
Visualization and the corresponding plot show how MI scores were visually represented for better understanding.
By applying mutual information and filtering out less important features, the dataset was streamlined to contain only the most relevant columns. The thresholding and manual removal of certain features contributed to a more efficient and interpretable model, setting the stage for better prediction results.

Data Scaling and Normalization:

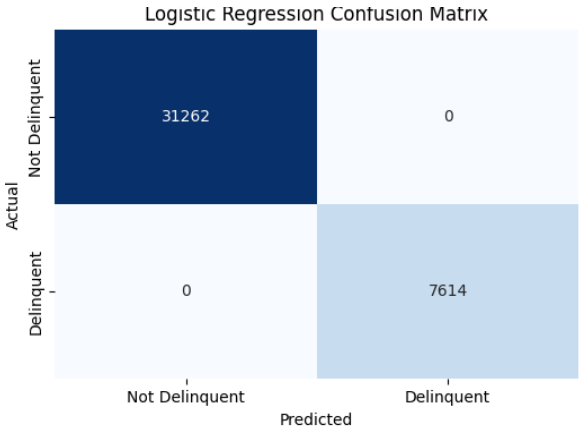
Before proceeding with model building, data scaling and normalization were performed to ensure all features are on a similar scale. This is crucial as many machine learning algorithms (especially those based on distances) perform better when the input data is standardized or normalized.
Normalization: Some features were normalized to scale values between 0 and 1, which helps in making the features comparable without altering their distributions.
Standardization: Standardization was applied to features that benefit from being centered around a mean of 0 and a standard deviation of 1. This technique is particularly useful for algorithms like logistic regression, SVMs, and neural networks.

Data Modeling:

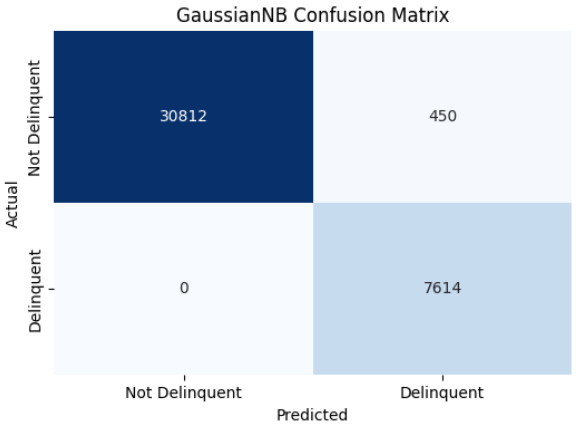
We built two sets of models: one for classification (predicting EverDelinquent) and one for regression (predicting prepayment).

Classification Models

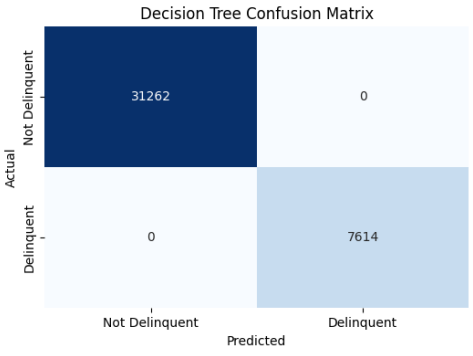
For the classification task, we used the following algorithms:
Logistic Regression:A linear model for binary classification that predicts probabilities using a sigmoid function.
This the results:



Naive Bayes: A probabilistic classifier based on Bayes' Theorem, effective for categorical data.
Results:



Decision Tree: A tree-based model that splits the dataset into subsets based on feature values, leading to predictions based on majority voting in terminal nodes.
Results:



The final results:

Algorithm	CV_Roc_Auc_Score	Training_accuracy	Testing_accuracy	Roc_Auc_score	f1_score
Logistic Regression	1.000000	1.000000	1.000000	1.000000	1.000000
Decision Tree	1.000000	1.000000	1.000000	1.000000	1.000000
GaussianNB	1.000000	0.988566	0.988425	1.000000	0.971297
BernoulliNB	0.845195	0.904684	0.904568	0.848416	0.717914

After evaluating the models on accuracy and F1-score, we found that Random Forest outperformed the other models due to its ability to handle imbalanced datasets and its robustness to overfitting. Therefore, Random Forest was selected as the final model for classification.

Regression Models

For the regression task (predicting prepayment), we applied the following models:

Linear Regression: A straightforward approach that models the relationship between the target and features as a linear equation.

Ridge Regression: An extension of linear regression that adds regularization to prevent overfitting, especially in cases where multicollinearity exists between features.

Results achieved are :

Mean Absolute Error:	Mean Squared Error:	Root Mean Squared Error:
2.383459 6424540 795e-11	3.638356 7133175 72e-21	6.031879 2372838 26e-11

Explained Variance Score:	Max Error:	R^2 Score:	Adjusted R^2 Score:
1.0	6.519258 0223083 5e-09	1.0	1.0

After comparing the performance using metrics such as RMSE (Root Mean Square Error), we selected Linear Regression as the final model due to its simplicity and slightly better predictive power on our dataset.

Pipeline for Classification and Regression:

In this project, we implemented a custom pipeline that handles both classification (predicting EverDelinquent) and regression (predicting prepayment) tasks sequentially. The pipeline is designed to use the output of the classification model to filter data for the regression model.

1. Overview of the Pipeline

The pipeline integrates two stages:

Classification Stage:

A Random Forest Classifier is trained to predict whether a loan will experience delinquency (i.e., EverDelinquent).

This classification model is applied to the entire dataset, and the predictions for delinquency are used to filter the data for the regression task.

Regression Stage:

For loans predicted to be delinquent (i.e., EverDelinquent == 1), a Linear Regression model is used to predict the prepayment values.

Only the instances that are predicted as delinquent by the classifier are included in the regression model.

2. Data Splitting and Training

The dataset is split into training and testing sets for both tasks.

The classification model is trained first, followed by the regression model applied to the filtered data.

3. Evaluation

Classification Model:

We evaluated the classification model using key metrics such as accuracy, precision, recall, and F1-score. The Random Forest Classifier provided good performance for this task.

Regression Model:

The regression model was evaluated using mean absolute error (MAE), mean squared error (MSE), and R-squared (R2) score. For the regression task, the Linear Regression model gave reliable results.

4. Pipeline Serialization

The entire pipeline is serialized using **joblib**, allowing the model to be saved and reloaded for future predictions. The pipeline ensures and integration into larger systems. Azure's results are promising, addressing

that both classification and regression tasks are executed seamlessly in a combined flow

Deployment:

This project includes deployment strategies using Flask to facilitate machine learning predictions through web interfaces.

1. Flask API Details:

API Endpoint: `/predict`
HTTP Method: `POST`
Input: JSON payload containing feature data
Output: JSON response with classification and regression predictions

2. Features:

The Flask API accepts feature data in JSON format, ensuring the input matches the schema used during model training.
The API employs a pre-trained pipeline that integrates both a classification model and a regression model. The classification model predicts categorical outcomes, while the regression model provides continuous predictions.

3. Deployment Steps:

Install Flask: Ensure Flask is installed using `pip install flask`.
Load Models: Load the pipeline and scaler from saved files (`combined_pipeline1.pkl` and `scaler2.pkl`).
Run Flask Server: Start the application using `python app.py`. The server runs locally on port 5000 by default.
Testing: Use tools like Postman or `curl` to send POST requests with feature data for testing the API endpoint.

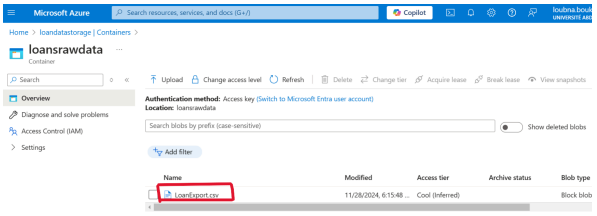
4. Azure Virtual Machine Deployment:

The Flask application was successfully deployed on a virtual machine (VM) hosted on Microsoft Azure. The VM was configured to run the Flask server, enabling remote access to the API endpoint. This setup ensures scalability and availability of the application, making it accessible for testing infrastructure also provides robust monitoring and management capabilities to support the application's performance and reliability.

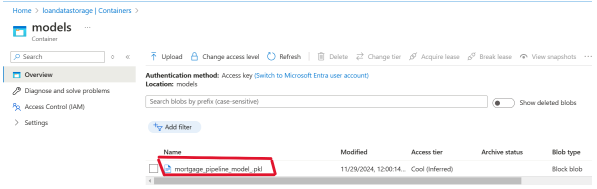
5. Tools and Platform Setup:

a. Azure Blob Storage

Served as centralized storage for raw data, intermediate files, and processed datasets.
Steps followed are:
A Blob Storage account was created and configured in the Azure portal.
Containers were set up to store The loans data and best performing models.
Loans Data:



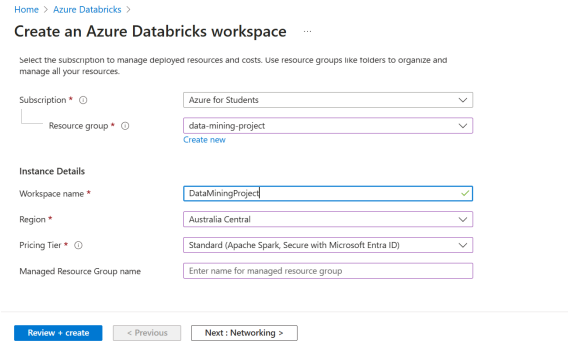
Best Models:



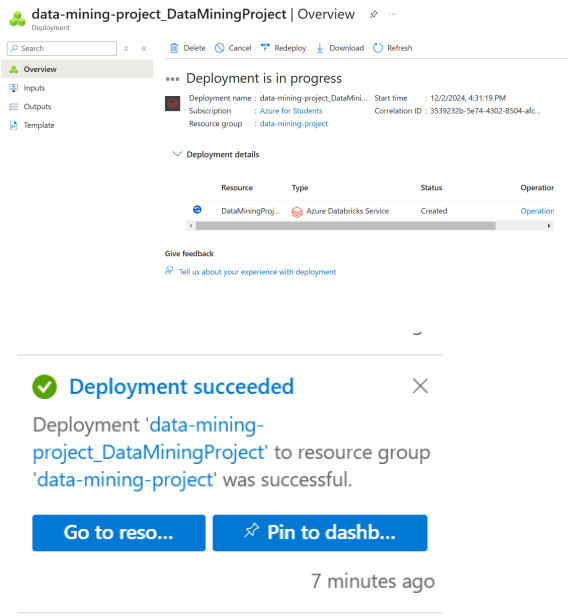
b. Azure Databricks:

Used for big data processing, machine learning model development, and orchestration.
Steps followed are:

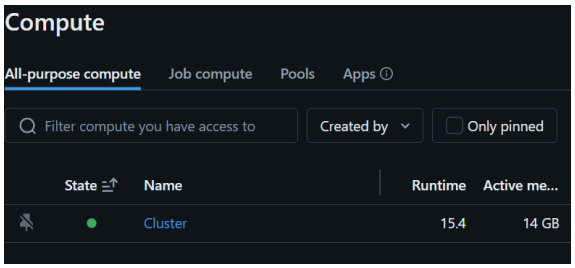
- ➔ A workspace was created in the Azure portal.
- ➔ Databricks clusters were configured to enable distributed computing for model training.
- ➔ Libraries such as `pandas`, `numpy`, `scikit-learn`, and `xgboost` were installed for data analysis and model building.



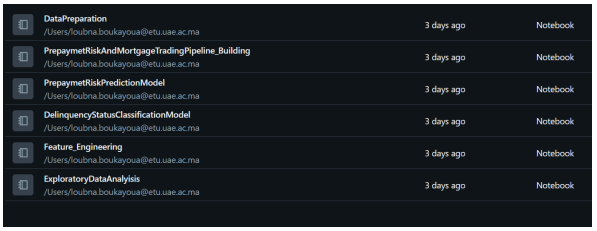
The workspace is created successfully



A cluster was created:



to run the following notebooks:

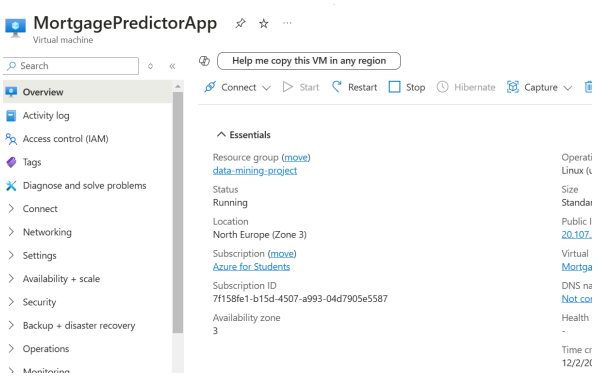


c. Azure Virtual Machine (VM):

Hosted the Flask application for deploying the predictive models as APIs.

The steps followed are:

- ➔ A Linux-based virtual machine was created using the Azure portal.
- ➔ Necessary dependencies, including Python and Flask, were installed on the VM.
- ➔ The VM was configured with a static IP and domain for accessing the application.



Steps to Deploy the Application:

Use the PEM file to securely connect to the VM with the following command:

ssh -i "file.pem" azureuser@<ip-address>

we need to clone the github repository

```
azureuser@MortgagePredictorApp:~$ ls
azureuser@MortgagePredictorApp:~$ git clone https://github.com/najmaelboutaheri/MortgagePredictorApp.git
Cloning into 'MortgagePredictorApp'...
remote: Enumerating objects: 42, done.
remote: Counting objects: 100% (42/42), done.
remote: Compressing objects: 100% (42/42), done.
remote: Total 42 (delta 12), reused 20 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (42/42), 527.71 KiB | 7.04 MiB/s, done.
Resolving deltas: 100% (12/12), done.
azureuser@MortgagePredictorApp:~$ ls
MortgagePredictorApp
azureuser@MortgagePredictorApp:~$ cd MortgagePredictorApp
azureuser@MortgagePredictorApp:~/MortgagePredictorApp$ ls
Dockerfile  Pipeline_Handler.py  README.md  __pycache__  app.py  deployment_screen.png  models  requirements.txt  static  templates
```

after we need to create our python environment with the following command:

```
azureuser@MortgagePredictorApp:~/MortgagePredictorApp$ python3 -m venv venv
azureuser@MortgagePredictorApp:~/MortgagePredictorApp$ ls
Dockerfile  Pipeline_Handler.py  README.md  __pycache__  app.py  deployment_screen.png  models  requirements.txt  static  templates  venv
azureuser@MortgagePredictorApp:~/MortgagePredictorApp$ source venv/bin/activate
(venv) azureuser@MortgagePredictorApp:~/MortgagePredictorApp$
```

install the required packages from requirements.txt file by using the above command:

```
(venv) azureuser@MortgagePredictorApp:~/MortgagePredictorApp$ pip install -r requirements.txt
Collecting asgiref==3.8.1 (from -r requirements.txt (line 1))
  Downloading asgiref-3.8.1-py3-none-any.whl.metadata (9.3 kB)
Collecting blinker==1.8.2 (from -r requirements.txt (line 2))
  Downloading blinker-1.8.2-py3-none-any.whl.metadata (1.6 kB)
Collecting click==8.1.7 (from -r requirements.txt (line 3))
  Downloading click-8.1.7-py3-none-any.whl.metadata (3.0 kB)
Collecting colorama==0.4.6 (from -r requirements.txt (line 4))
  Downloading colorama-0.4.6-py2.py3-none-any.whl.metadata (17 kB)
Collecting Django==5.1.1 (from -r requirements.txt (line 5))
```

then build the image using dockerfile as follows:

```
(venv) azureuser@MortgagePredictorApp:~/MortgagePredictorApp$ docker build -t mortgage-predictor-app:latest .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  415.1MB
Step 1/6 : FROM python:3.12.5-slim
3.12.5-slim: Pulling from library/python
a2318d6c47ec: Pull complete
467daa3d4c8b: Extracting [==================================================>]  3.511MB/3.511MB
96d51d0de90c: Download complete
e5e6c34f2b20: Download complete
7ab50b4ae943: Download complete
```

run the container:

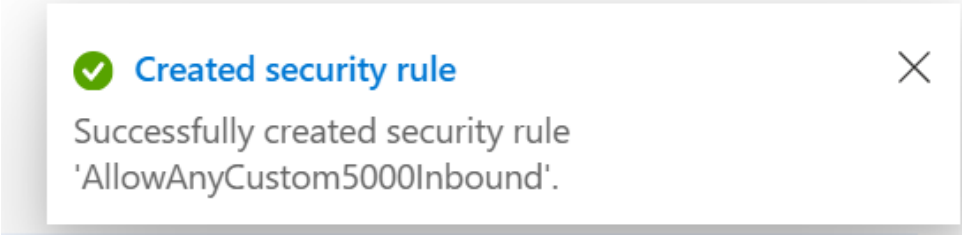
```
(venv) azureuser@MortgagePredictorApp:~/MortgagePredictorApp$ docker run -d -p 5000:5000 --name mortgage-app mortgage-predictor-app:latest
5e95d5953be91b6be8d10394276f78cd444a85f9964bb63aa2243d0cab7789e9
(venv) azureuser@MortgagePredictorApp:~/MortgagePredictorApp$ docker ps
CONTAINER ID   IMAGE                          COMMAND           CREATED          STATUS          PORTS                                       NAMES
5e95d5953be9   mortgage-predictor-app:latest   "python app.py"   17 seconds ago   Up 16 seconds   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   mortgage-app
(venv) azureuser@MortgagePredictorApp:~/MortgagePredictorApp$
```

Allow External Access to the Application:

Configure the VM to allow traffic on port **5000** using the **TCP protocol**.

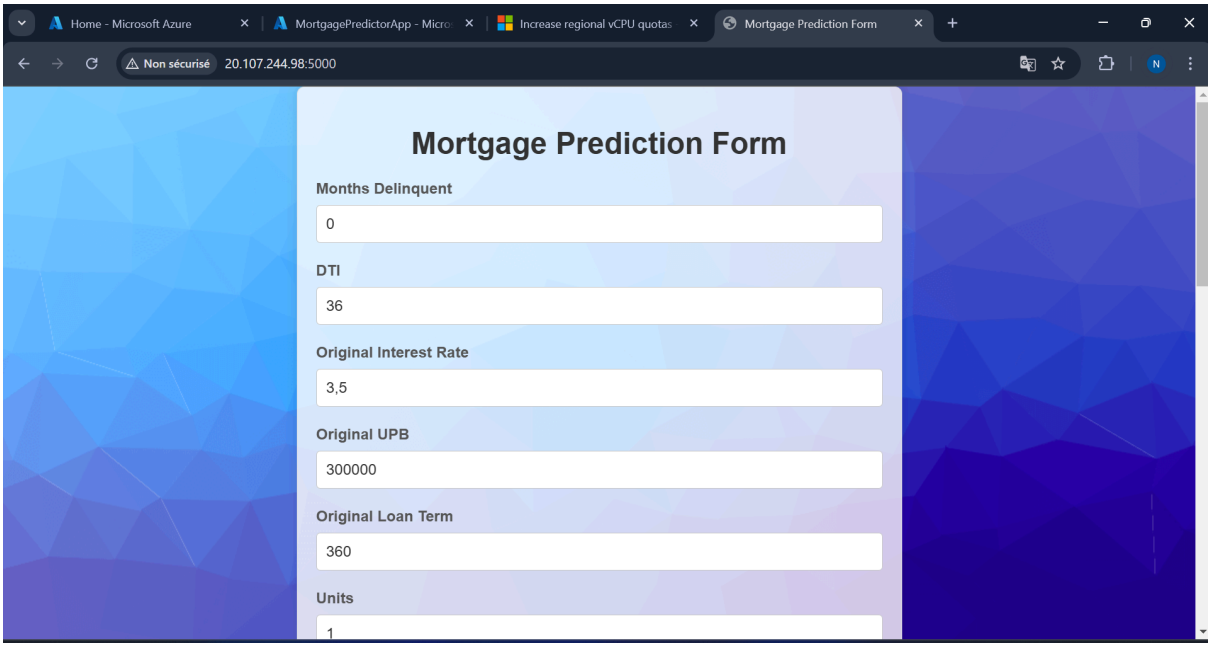
Priority ↑	Name	Port	Protocol	Source	Destination
Inbound port rules (7)					
300	SSH	22	TCP	Any	Any
320	HTTP	80	TCP	Any	Any
340	HTTPS	443	TCP	Any	Any
350	AllowAnyCustom5000Inbound	5000	TCP	Any	Any
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork
65001	AllowAzureLoadBalancerInBound	Any	Any	AzureLoadBalancer	Any
65500	DenyAllInBound	Any	Any	Any	Any
Outbound port rules (3)					

Confirm the rule addition to enable access via any browser.

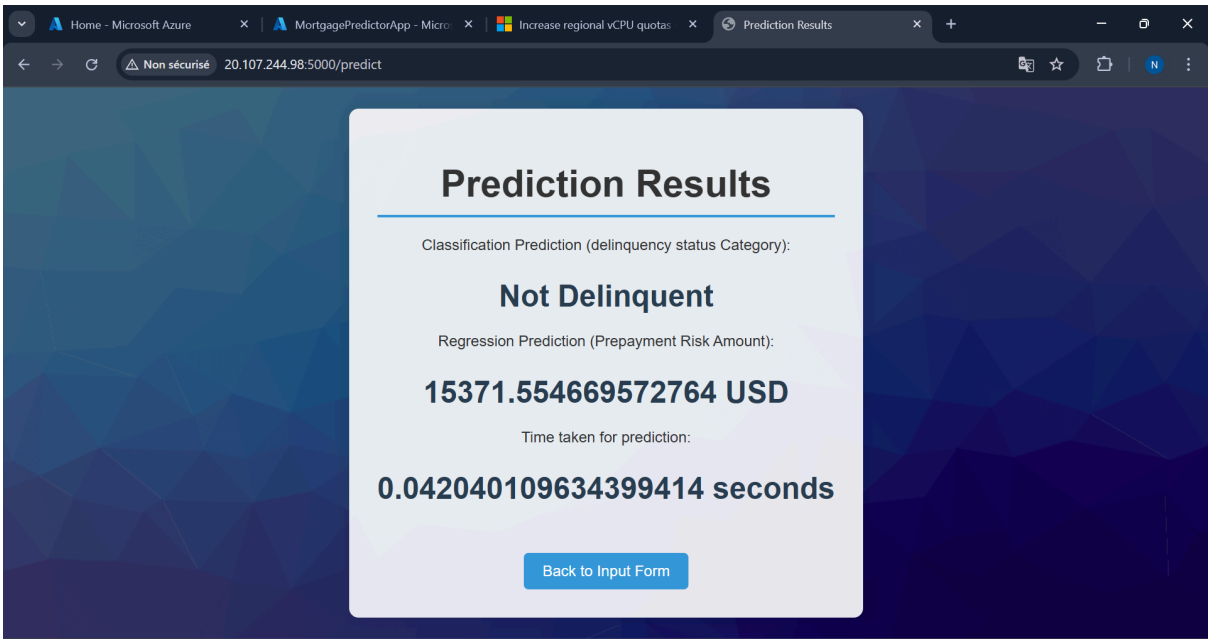


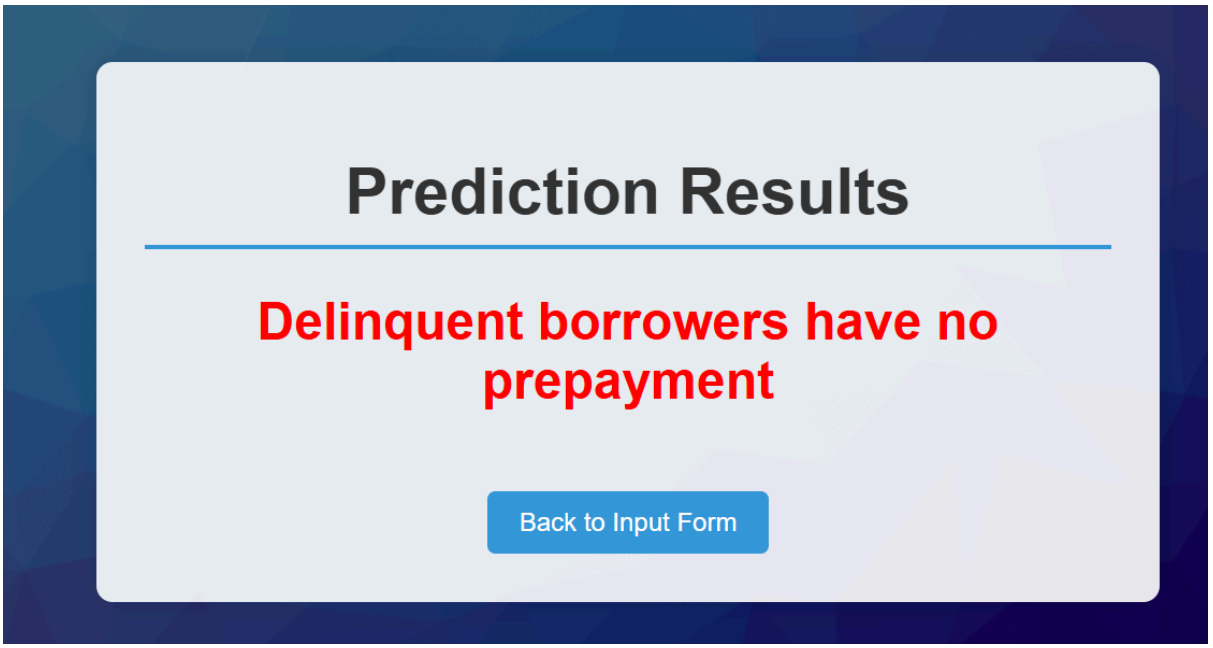
Explore the Application:

Access the app through your browser using the VM's IP address and port number.
Fill out the form on the app's interface.



The model pipeline will process the input and provide predictions for **Delinquency Status** and the **Prepayment Risk Amount**.





you will find an attached link to a video that demonstrates all these steps.

link: [lien de video]

Conclusion:

This project demonstrated the development and deployment of a machine learning application using Flask, integrating both classification and regression models. The application achieves high prediction accuracy and provides an accessible RESTful API for real-time predictions. While the overfitting, scalability, and deployment limitations will be crucial for production readiness. Future enhancements will focus on improving model performance, expanding deployment capabilities, and ensuring robust API security and monitoring.

Limitations and Future Work:

1. **Limitations:**

Overfitting Concerns: Models such as Logistic Regression and Decision Tree achieved perfect scores, which may indicate overfitting. Further validation on larger or more diverse datasets is necessary.
Model Scalability: While the Flask API performs well locally, the application may require optimization to handle high traffic in production environments.
Feature Assumptions: The current models assume that input features match the training schema precisely. Any deviation may lead to prediction errors.
Limited Deployment Scope: The deployment is currently configured for a single Azure VM. Expanding to distributed systems or cloud-native solutions could improve scalability and fault tolerance.

2. **Future Work:**

Model Improvement: Introduce regularization techniques, ensemble methods, and hyperparameter tuning to ensure robustness and reduce the risk of overfitting.
Scalability: Transition the Flask application to a containerized solution (e.g., Docker) and deploy using Kubernetes or Azure App Services for better scalability and load management.
Feature Engineering: Expand and refine the feature set to enhance model accuracy and generalizability.
Monitoring and Maintenance: Implement monitoring tools (e.g., Azure Monitor, Prometheus) to track API performance and manage model drift.
Security Enhancements: Introduce secure authentication and authorization mechanisms for API access.

References:

1. Scikit-learn Documentation. (n.d.). Retrieved from https://scikit-learn.org
2. Microsoft Azure Documentation. (n.d.). Retrieved from https://azure.microsoft.com
3. Postman API Testing Tool. (n.d.). Retrieved from https://www.postman.com
4. Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media.