



Northeastern University

Module 6- Team Project: Final Report
ALY6040 20351 - Data Mining Applications SEC 02
Winter 2024 CPS

Submitted By

Group 3

Krati Jadaun

Ritika Sharma

Saiyinu Dilimulati

Samruddhi Amarsinh Raut

Tauheed Shaik Haroon

Submitted To

Prof. Kasun Samarasinghe

Dataset Selection

This project investigates a dataset about credit scores that includes basic banking details and credit information for customers. The goal is to analyze the data to find insights into what factors affect credit scores based on fundamental banking details and credit details, in order to automate the loan approval system. We will develop a classification model using this dataset that can predict the loan approval status using the available banking and credit information. The model will identify key banking and credit features that are predictive of loan status.

- Number of Rows: 15,00,00
- Variables: 28

Data Dictionary:

Feature	Description	Data Type
ID	Identifier for each record	string
Customer_ID	Unique identifier for each customer	string
Month	The month of the data record	string
Name	Customer's name	string
Age	Customer's age, which appears to have inconsistencies like '24_'	string
SSN	Social Security Number of the customer	string
Occupation	Customer's occupation, some entries might be missing or filled with placeholders like '_____'	string
Annual_Income	Reported annual income, which has been input as a string possibly due to formatting issues	string
Monthly_Inhand_Salary	Monthly salary after deductions	float
Num_Bank_Accounts	Number of bank accounts	integer
Num_Credit_Card	Number of credit cards	integer
Interest_Rate	Interest rate applicable to the customer	integer
Num_of_Loan	Number of loans, which is inconsistently formatted	string
Type_of_Loan	Type of loan(s) taken by the customer, possibly a concatenated list	string
Delay_from_due_date	Days delayed from the due date for payments	integer
Num_of_Delayed_Payment	Number of times payments were delayed, possibly due to inconsistent formatting	string
Changed_Credit_Limit	Changes in the credit limit, possibly due to formatting issues	string
Num_Credit_Inquiries	Number of credit inquiries, possibly with missing values	float
Credit_Mix	Classification of credit mix	string
Outstanding_Debt	Amount of outstanding debt, likely due to formatting issues	string
Credit_Utilization_Ratio	Ratio of credit utilization	float
Credit_History_Age	Age of the credit history, formatted as 'X Years and Y Months'	string

Payment_of_Min_Amount	Indicator of whether minimum payment was made	string
Total_EMI_per_month	Total EMI payments per month	float
Amount_invested_monthly	Monthly investment amount, likely due to inconsistencies in formatting	string
Payment_Behaviour	Description of payment behavior	string
Monthly_Balance	Monthly balance amount, likely due to formatting issues	string
Credit_Score	Credit score classification (only in train.csv)	string

Use Case: Automated Loan Approval System

In this use case, a bank is introducing a new loan scheme, and to streamline the approval process and mitigate risks, they are implementing a machine learning model based on credit-related variables from their dataset.

The goal is to automate the loan approval process and enhance the efficiency of evaluating loan applications. The model will also be used for targeted marketing campaigns to identify potential customers who are more likely to qualify for the new loan scheme.

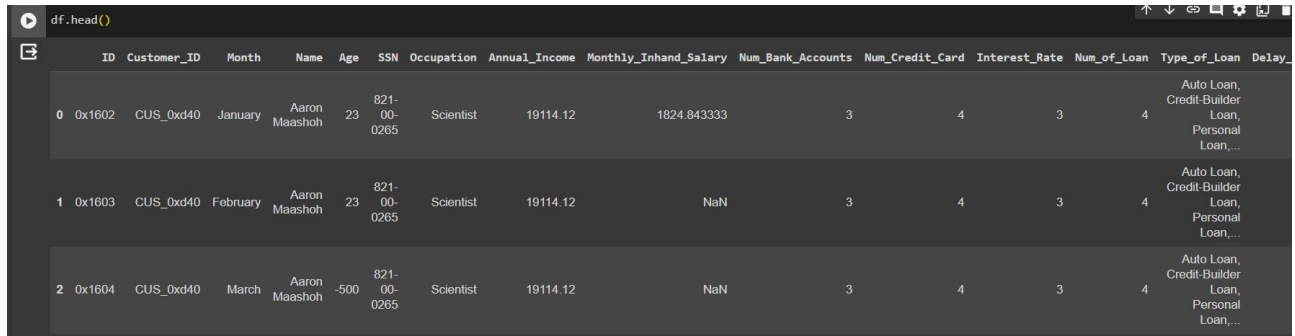
Objectives:

- **Automate Loan Approval Process:** The classification model will analyze the historical credit-related data, including attributes such as Annual Income, Monthly Inhand Salary, Credit History Age, Num_Bank_Accounts, Num_Credit_Card, and others. By learning patterns from past loan approvals and rejections, the model will predict whether a new loan application should be approved or not. This automation aims to improve efficiency, reduce manual effort, and lower the chances of granting loans to customers with higher credit risks.
- **Enhance Risk Management:** The model will consider variables like Num_of_Delayed_Payment, Delay_from_due_date, Outstanding_Debt, and Credit_Utilization_Ratio to assess the risk associated with each applicant. By evaluating historical payment behavior and outstanding debts, the bank can make informed decisions to minimize the risk of default.
- **Targeted Marketing Campaigns:** The classification model's predictions will be used to target marketing campaigns specifically to individuals who are more likely to qualify for the new loan scheme. Marketing efforts can be directed towards segments with a higher likelihood of approval, optimizing resources and improving the effectiveness of the campaign.
- **Target Variable:** Loan_Approval_Status. 1 or Yes: indicating that the loan application has been approved. 0 or No: indicating that the loan application has been rejected.

Data Cleaning & Pre-Processing

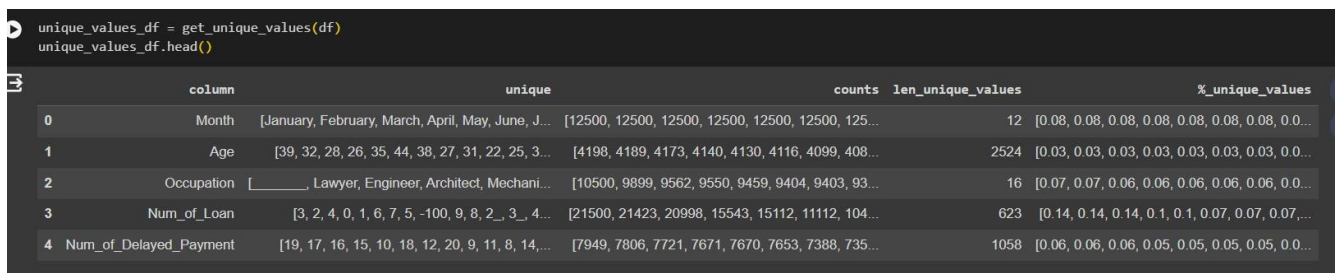
Data cleaning and preprocessing are essential steps in the data science workflow to ensure that the data is in a suitable form for analysis and modeling. For the dataset we have followed a thorough approach for data cleaning and pre-processing. Below is the brief explanation of each step that made the data clean and usable for our analysis.

Before diving into the cleaning process, first we understand what the data looks like and what it contains, For this we have used functions like `df.head()` & `unique_values_df.head()`.



	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	Num_of_Loan	Type_of_Loan	Delay
0	0x1602	CUS_0xd40	January	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	1824.843333	3	4	3	4	Auto Loan, Credit-Builder Loan, Personal Loan,...	
1	0x1603	CUS_0xd40	February	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	NaN	3	4	3	4	Auto Loan, Credit-Builder Loan, Personal Loan,...	
2	0x1604	CUS_0xd40	March	Aaron Maashoh	-500	821-00-0265	Scientist	19114.12	NaN	3	4	3	4	Auto Loan, Credit-Builder Loan, Personal Loan,...	

Figure 1: Dataset overview



	column	unique	counts	len_unique_values	%_unique_values
0	Month	[January, February, March, April, May, June, J...	[12500, 12500, 12500, 12500, 12500, 12500, 125...	12	[0.08, 0.08, 0.08, 0.08, 0.08, 0.08, 0.08, 0.0...
1	Age	[39, 32, 28, 26, 35, 44, 38, 27, 31, 22, 25, 3...	[4198, 4189, 4173, 4140, 4130, 4116, 4099, 408...	2524	[0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.0...
2	Occupation	[_____, Lawyer, Engineer, Architect, Mechani...	[10500, 9899, 9562, 9550, 9459, 9404, 9403, 93...	16	[0.07, 0.07, 0.06, 0.06, 0.06, 0.06, 0.06, 0.0...
3	Num_of_Loan	[3, 2, 4, 0, 1, 6, 7, 5, -100, 9, 8, 2, 3, 4...	[21500, 21423, 20998, 15543, 15112, 11112, 104...	623	[0.14, 0.14, 0.14, 0.1, 0.1, 0.07, 0.07, 0.07...
4	Num_of_Delayed_Payment	[19, 17, 16, 15, 10, 18, 12, 20, 9, 11, 8, 14, ...	[7949, 7806, 7721, 7671, 7670, 7653, 7388, 735...	1058	[0.06, 0.06, 0.06, 0.05, 0.05, 0.05, 0.05, 0.0...

Figure2: Unique Values in the Dataset

- 1) **Identifying Missing Values:** In this part we calculate the count and percentage of missing values in each column.



```
Missing Values Percentage:
ID: 0.00%
Customer_ID: 0.00%
Month: 0.00%
Name: 10.00%
Age: 0.00%
SSN: 0.00%
Occupation: 0.00%
Annual_Income: 0.00%
Monthly_Inhand_Salary: 15.00%
Num_Bank_Accounts: 0.00%
Num_Credit_Card: 0.00%
Interest_Rate: 0.00%
Num_of_Loan: 0.00%
Type_of_Loan: 11.41%
Delay_from_due_date: 0.00%
Num_of_Delayed_Payment: 7.00%
Changed_Credit_limit: 0.00%
Num_Credit_Inquiries: 2.00%
Credit_Mix: 0.00%
Outstanding_Debt: 0.00%
Credit_Utilization_Ratio: 0.00%
Credit_History_Age: 9.00%
Payment_of_Min_Amount: 0.00%
Total_EMI_per_month: 0.00%
Amount_invested_monthly: 4.50%
Payment_Behaviour: 0.00%
Monthly_Balance: 1.17%
Credit_Score: 33.33%
```

Figure3: Missing Values

- 2) **Treating Missing Values:** Replaces missing values in numerical columns with their respective means.

- 3) **Checking for Duplicates:** Finds and reports any duplicate rows in the data frame. In our dataset there are no duplicate values.

```
## Duplicate Values
is_duplicate = df.duplicated().any()
if is_duplicate:
    print("Dataframe has duplicate entries.")
else:
    print("Dataframe does not have any duplicate entries.")

Dataframe does not have any duplicate entries.
```

Figure 4: Duplicate Values

- 4) **Data Pre-processing Function:** A function is created with the name *DataProcessor()*, in which it accepts a group by argument which specifies the column on which certain operations like filling missing values will be based, and a data_frame object that is the target of the preprocessing. The class contains a mix of instance methods and static methods that perform various data cleaning and formatting operations, such as converting date strings to a count of months, extracting numbers from strings, and replacing special characters.
- 5) **Outlier Removal and Skewness Correction:** The *remove_outliers* function orchestrates the removal of outliers from numerical columns in a Pandas *DataFrame*. Leveraging skewness thresholds, it identifies highly and lowly skewed features and applies the *apply_clip_transformer* function accordingly. Finally, the script applies the *remove_outliers* function to the dataset named data, effectively pre-processing the data by mitigating the impact of outliers on the machine learning model.
- 6) **Standardizing of Numerical Variables:** The standardizing of numerical values is used in data analysis to put all the variables on the same scale. This was implemented correcting the *Months* variable using function *convert_years_months_to_months* designed to transform the column containing strings representing durations in years and months into a single integer column representing the total duration in months.
- 7) **Type Conversion:** Type conversion was performed using (*type_cast_fn*) function. The function converts specified columns to either 'int64' or 'float64'. The function first removes underscores from the column values and then converts them to numeric types using *pd.to_numeric*.

Customer_ID	object
Month	object
Age	int64
Occupation	object
Annual_Income	float64
Monthly_Inhand_Salary	float64
Num_Bank_Accounts	int64
Num_Credit_Card	int64
Interest_Rate	int64
Num_of_Loan	int64
Type_of_Loan	object
Delay_from_due_date	int64
Num_of_Delayed_Payment	float64
Changed_Credit_Limit	float64
Num_Credit_Inquiries	float64
Credit_Mix	object
Outstanding_Debt	float64
Credit_Utilization_Ratio	float64
Credit_History_Age	int64
Payment_of_Min_Amount	object
Total_EMI_per_month	float64
Amount_invested_monthly	float64
Payment_Behaviour	object
Monthly_Balance	float64

Figure 5: Type Conversion Variable

Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is a crucial initial step in the data analysis process where the main goal is to understand the structure, patterns, and characteristics of the data. EDA involves using statistical and visual methods to gain insights, identify patterns, and uncover potential relationships within the dataset.

1) Box Plots

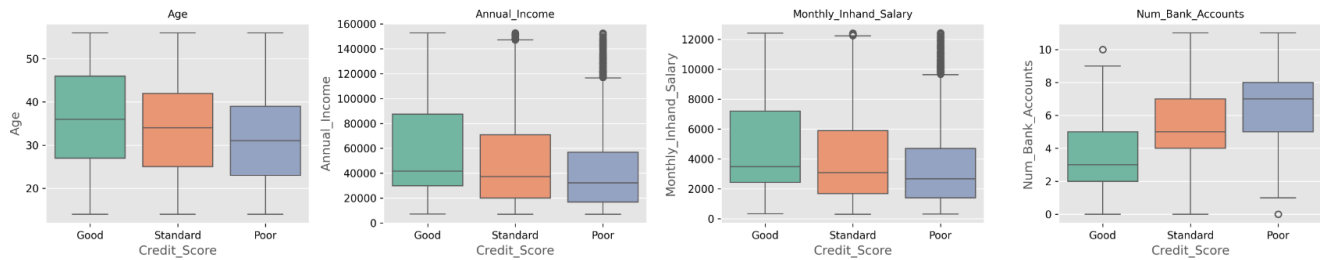


Figure 6: Box Plots

Description:

The figure displays box plots comparing Age, Annual Income, Monthly In-hand Salary, and Number of Bank Accounts across Good, Standard, and Poor credit score categories. The median age is relatively consistent across the categories, with slight variations. Income measures show a clear descending trend from Good to Poor credit scores, with the highest incomes in the Good category, which also has several high-value outliers. The number of bank accounts follows a similar trend, with the highest medians and ranges observed in the Good credit score group.

Findings:

Credit score quality appears to be positively associated with financial indicators; individuals with Good credit scores have higher incomes and more bank accounts. While age distribution is somewhat consistent across credit score categories, Poor credit scores are generally linked to lower incomes and fewer bank accounts, indicating a potential correlation between financial stability and creditworthiness.

2) Bar Chart (Credit Score by Credit Mix)

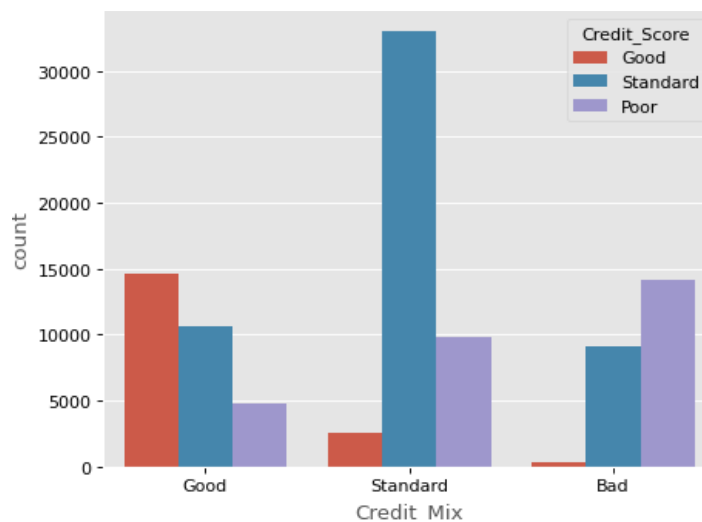


Figure 7: Credit Mix Ratio

- Description:** This chart shows the distribution of credit scores (Good, Standard, Poor) across different credit mix categories (Good, Standard, Bad).
- Findings:** The majority of the data points fall into the 'Standard' credit mix category. Within this category, a 'Good' credit score is the most common, followed by 'Poor' and 'Standard'. For both 'Good' and 'Bad' credit mixes, the 'Good' credit score appears to be the most frequent. This might suggest that individuals with a good credit mix tend to have good credit scores, but there is also a significant number of individuals with a good credit score even in the 'Bad' credit mix category.

3) Donut Chart (Monthly Balance by Credit Score and Credit Mix)

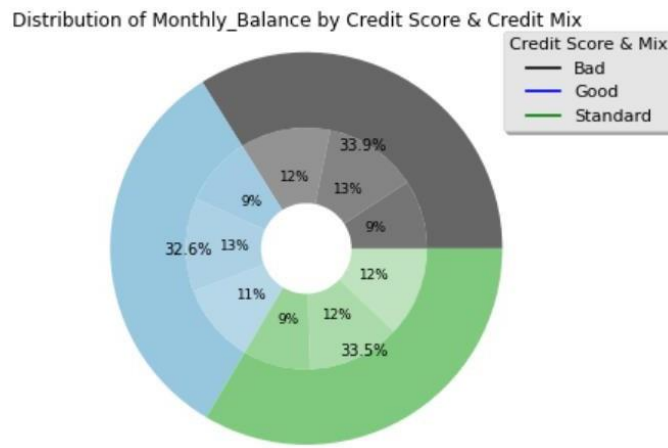


Figure 8: Donut Chart

- **Description:** The donut chart illustrates the distribution of monthly balance percentages across different combinations of credit score and credit mix.
- **Findings:** The chart suggests that the monthly balance is evenly distributed among different credit score categories within the 'Good' and 'Standard' credit mix. However, it is not immediately clear what the percentages represent without further context. The chart could be indicating the proportion of total balances held by each subgroup or the average monthly balance as a percentage of some baseline figure.

4) Correlation Matrix

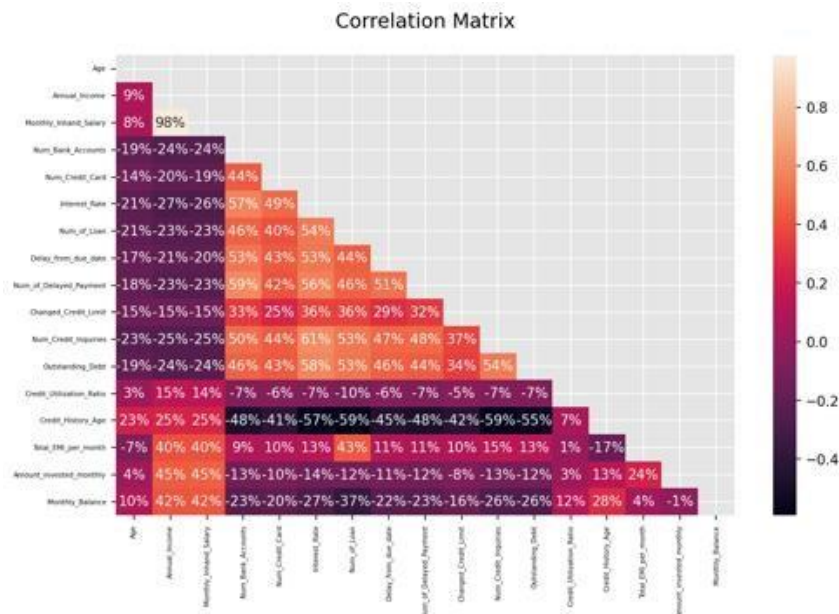


Figure 9: Correlation Plot

Findings:

- **High Positive Correlations:** Monthly_Inhand_Salary is highly correlated with Annual_Income (98%), which is expected as they are directly related. Num_Credit_Inquiries has a relatively high positive correlation with Num_of_Loan (61%) and Num_of_Delayed_Payment (53%), indicating that those who have more inquiries also tend to have more loans and delayed payments.

- **High Negative Correlations:** Age has a significant negative correlation with Num_Bank_Accounts (-19%), Num_Credit_Card (-14%), and Num_of_Loan (-21%), suggesting younger individuals might have more of these financial products or engagements.
- **Low or Insignificant Correlations:** Credit_Utilization_Ratio shows very low correlation with most of the variables, which might indicate it is independent of factors like the number of bank accounts or credit cards a person holds.
- **Surprising Insights:** Delay_from_due_date is strongly correlated with Num_of_Delayed_Payment (59%), suggesting that those who delay often also tend to delay for longer periods. Also, Outstanding_Debt has a strong correlation with Num_of_Loan (54%), which is intuitive as more loans could mean more outstanding debt.

5) Histograms

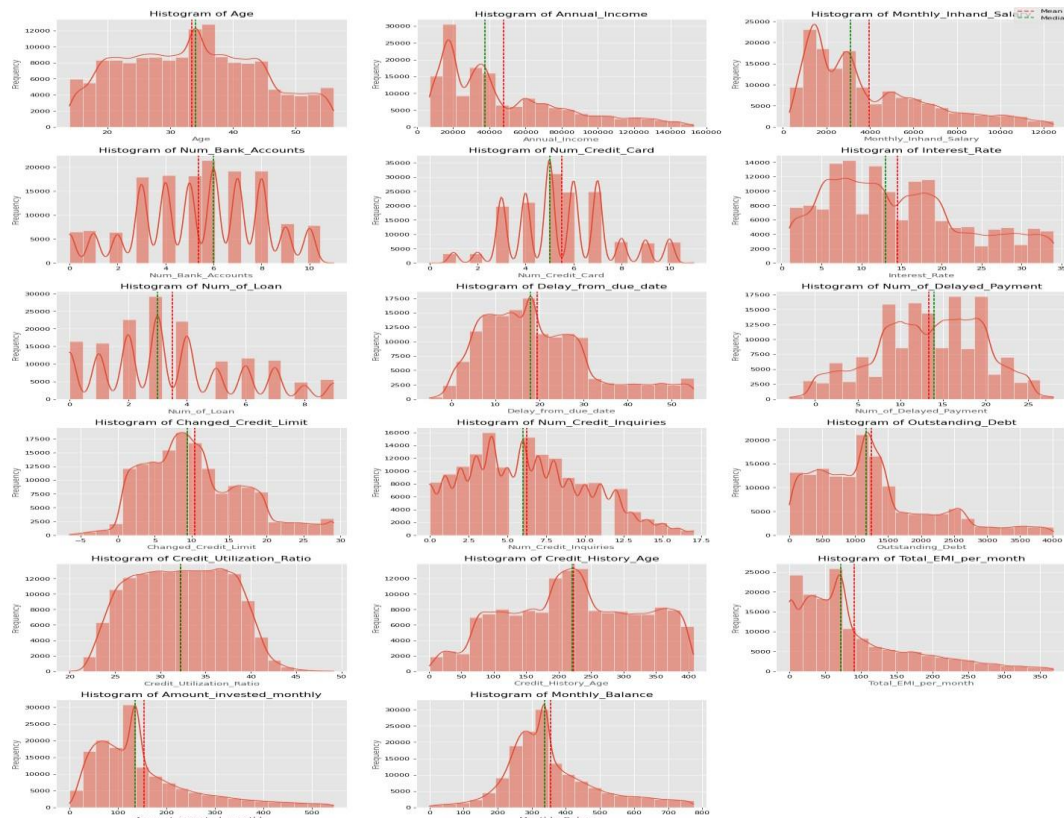


Figure 10: Histograms

Here are the key points regarding outliers and invalid values identified from examining the histograms and quintile distributions:

- Several variables like age, income, number of bank accounts, number of loans, etc. exhibited outliers in their distributions.
- Some variables contained invalid unrealistic values:
 - Age had negative values
 - Number of loans and bank accounts had negatives
 - These are not valid real-world values
 - 55% of customers are aged 20-40 years old.
 - Only 5% have just 1 bank account, 65% have 4+ accounts.
 - 95% have at least 3 credit cards.
 - 65% have a credit utilization of 25-35% of their limit.

So in summary, the analysis revealed both invalid outlier values and valid insights into the distribution of key variables related to loans, accounts, and credit usage. Cleaning the invalid values will improve the data quality.

Analysis & Modeling

Now that the data is cleaned, processed and even EDA is implemented, we move on to the modelling part, in which we compare the below multiple models in order to understand which performed better to solve the use case.

1. Decision Tree
2. Random Forest
3. Gradient Boosting Machine (GBM)
4. Extreme Gradient Boosting (XGBoost)
5. Logistic Regression

Feature Engineering, Target Encoding & Data splitting

Before we jump into modeling, we have done feature engineering and data splitting and then one target variable is used in all the models according to our use case, which is loan_approval_status

Target Encoding: Target encoding is a technique used to encode categorical variables for machine learning models. In our analysis we have performed encoding for three particular variables, which are as follows:

- **Target Encoding for 'Occupation':** The code utilizes target encoding to transform the 'Occupation' column in the 'valid_df' and 'test_df' DataFrames. This technique replaces each category with the mean of the target variable, aiming to capture the relationship between occupation and the target.
- **Ordinal Encoding for 'Payment Behaviour' and 'Credit Mix':** The 'Payment_Behaviour' and 'Credit_Mix' columns undergo ordinal encoding, assigning numerical values based on predefined mappings. This enables the representation of ordinal relationships within these categorical features, capturing the inherent order in the payment behavior and credit mix categories.
- **Metric Calculator Function:** The provided function, 'metric_calculator', calculates and prints essential classification metrics, including accuracy, precision, recall, and F1 score. This function is designed to evaluate the performance of a classification model by comparing predicted and true labels.

Feature Engineering:

- The code calculates the total count of entries with 'Good' credit scores in the original DataFrame (df) and stores it in the variable total_limit, furthermore Subsequently, it randomly samples subsets of entries labeled as 'Standard' and 'Poor' credit scores to match half of the count of 'Good' credit scores. These sampled subsets are then concatenated into a new DataFrame labeled as 'Not good,' effectively creating a more balanced distribution of credit scores.
- The final step involves combining this 'Not good' DataFrame with the original entries labeled as 'Good,' resulting in a new DataFrame (balanced_df) where the 'Loan_Approval_Status' feature is replaced by a newly engineered feature named 'loan_approved_status.' This engineered feature ensures a more equitable representation of loan approval statuses in the dataset.

```
print("Distribution of target columns: \n")
print(balanced_df.Loan_Approval_Status.value_counts())
```

Distribution of target columns:

Not good	16532
Good	16532

Name: Loan_Approval_Status, dtype: int64

Figure 6: Creation of Loan_Approval_Status

Model 1: Decision Tree

A Decision Tree is a supervised machine learning algorithm that is used for both classification and regression tasks. It works by recursively partitioning the data into subsets based on the values of input features. The decision-making process resembles a tree-like structure, where each internal node represents a decision based on a specific feature, each branch represents the outcome of that decision, and each leaf node represents the final predicted class or numerical value.

```
print(X_train_scaled.shape(), y_train.shape(), y_test_scaled.shape(), y_test.shape())

Train results:
Accuracy classification score: 0.8512357414448669
Precision score: 0.8358813321213123
Recall score: 0.8358813321213123
F1 score: 0.8545600473112829
None
-----
Validation results:
Accuracy classification score: 0.8270161290322581
Precision score: 0.8194875776397516
Recall score: 0.8194875776397516
F1 score: 0.8311023622047244
None
```

Figure 7: Model 1 results- Before Tuning

- **Training Results:** The initial performance in training with an accuracy of 85.12%, precision and recall both at 83.58% and an F1 score of 85.46%. These results indicate effective prediction and balance between identifying relevant instances and minimizing errors, suggesting a solid foundation for further optimization.
- **Test Results:** Slightly lower but still good accuracy of 82.70%, precision and recall both at 81.95% and an F1 score of 83.11%. These metrics indicate the model's strong ability to generalize to unseen data, with a good balance between precision and recall, suggesting minimal overfitting or underfitting before any parameter tuning.

Overall, the model is effective at classifying correctly and maintaining a good balance between identifying true positives and avoiding false positives.

Model 1: Hyper-parameter Tuning

```
# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

Fitting 3 folds for each of 90 candidates, totalling 270 fits
Best Hyperparameters: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 2}
```

Fig.8: Hyper-parameter Tuning for Decision Tree

The output indicates the optimal hyper parameters for an ensemble decision tree model, likely a Random Forest, determined through hyper parameter tuning. A total of 100 trees (`n_estimators`) comprise the ensemble, maximizing the robustness and reducing the variance of the model. Each tree in the ensemble considers every feature (`max_features: 'auto'`) when determining the best split, possibly indicating the use of all features or a subset based on the square root of the total number of features. There is no restriction on the growth of the trees (`max_depth: None`), allowing them to expand fully for as long as they contribute to accuracy. The split quality is assessed using the 'entropy' criterion, aiming to maximize information gain and thus ensure the most discriminative features are used for decision-making. This setup is designed to create a strong predictive model that leverages the diversity of multiple decision trees to make accurate predictions.

Model 1: After Tuning

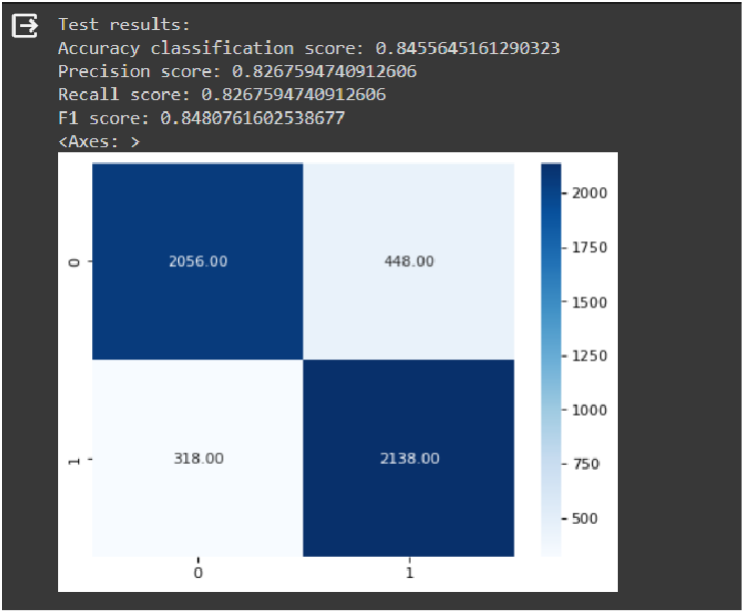


Figure 9: Model 1 results- After Tuning

- Accuracy Classification Score:** The model has an accuracy of approximately 84.56% on the test dataset, reflecting its effective generalization capabilities, but it is still lesser than before tuning.
- Precision Score:** The model's precision is about 82.68%, indicating a robust ability to distinguish between the classes and correctly identify positive instances.
- Recall Score:** With a recall of approximately 82.68%, the model demonstrates a strong ability to detect most actual positive cases.
- F1 Score:** The F1 score is 0.8481, showing a balanced trade-off between precision and recall, which suggests that the model maintains a harmonized performance in terms of accuracy and completeness in its prediction.

Model 2: Random Forest

Random Forest is an ensemble learning method used for both classification and regression tasks in machine learning. It is based on the idea of constructing a multitude of decision trees during the training phase and combining their outputs for making predictions.

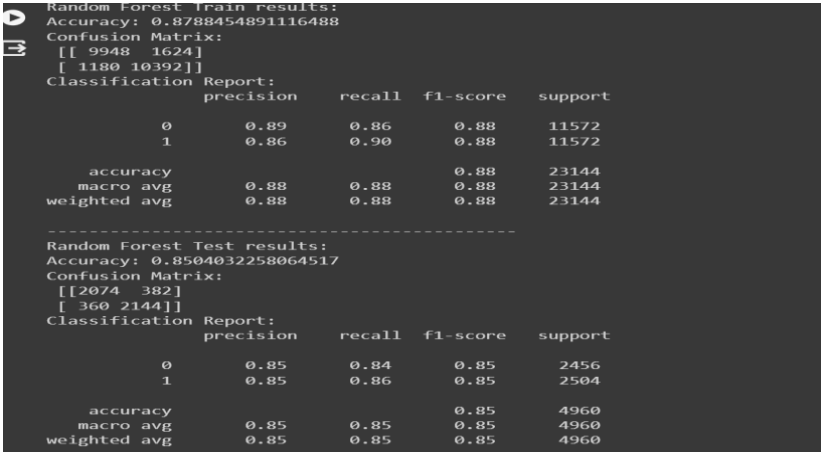


Figure 10: Model 2 results- Before Tuning

Training Results:

- Accuracy: The classifier has an accuracy of approximately 87.88% on the training data, which indicates a high level of correct predictions for the training samples.
- Confusion Matrix: The confusion matrix for the training results shows that the model predicted 9948 instances of the first class correctly and 1624 instances incorrectly. It predicted 1180 instances of the second class incorrectly while getting 103921 instances correct.
- Classification Report: The precision, recall, and F1-score for classifying the first class (class '0') are quite high (0.89, 0.86, and 0.88 respectively), indicating good performance. The same metrics for the second class (class '1') are even higher (0.90, 0.92, and 0.91 respectively), suggesting that the model is better at predicting the second class. The support for the first class is 11572 and for the second class is 23142, indicating the number of true instances for each class in the training dataset.

Testing Results:

- Accuracy: The accuracy of the classifier on the testing data is about 85.04%, which is slightly lower than the training accuracy but still indicates a good predictive performance.
- Confusion Matrix: For the testing data, the confusion matrix shows that the model predicted 2074 instances of the first class correctly and 382 instances incorrectly. For the second class, it predicted 306 instances incorrectly and 2441 instances correctly.
- Classification Report: The precision, recall, and F1-score for classifying the first class (class '0') are 0.85, 0.84, and 0.85 respectively. For the second class (class '1'), these metrics are 0.86, 0.89, and 0.87 respectively. The support for the first class is 2456 and for the second class is 2564, indicating the number of true instances for each class in the testing dataset.

Model 2: Hyper-parameter Tuning

```
Best Hyperparameters: {'n_estimators': 100, 'max_features': 'auto', 'max_depth': None, 'criterion': 'entropy'}
```

Fig.11: Hyper-parameter Tuning for Random Forest

The output shown is the result of hyperparameter tuning for a Random Forest classifier, indicating the most effective settings determined through the tuning process. The optimal number of decision trees (`n_estimators`) is 100, suggesting a combination of these many trees produces the best model performance. The `max_features` set to 'auto' means that each tree in the forest considers all the features when determining the best split. With `max_depth` set to None, there is no limit on the depth of the trees, allowing them to grow until all leaves are pure or contain very few samples. Finally, the `criterion` is set to 'entropy', which implies that the model uses information gain to evaluate splits, aiming to maximize class purity at each node. These tuned hyperparameters are intended to balance the bias-variance tradeoff and prevent overfitting while achieving the best predictive accuracy.

Model 2: After Tuning

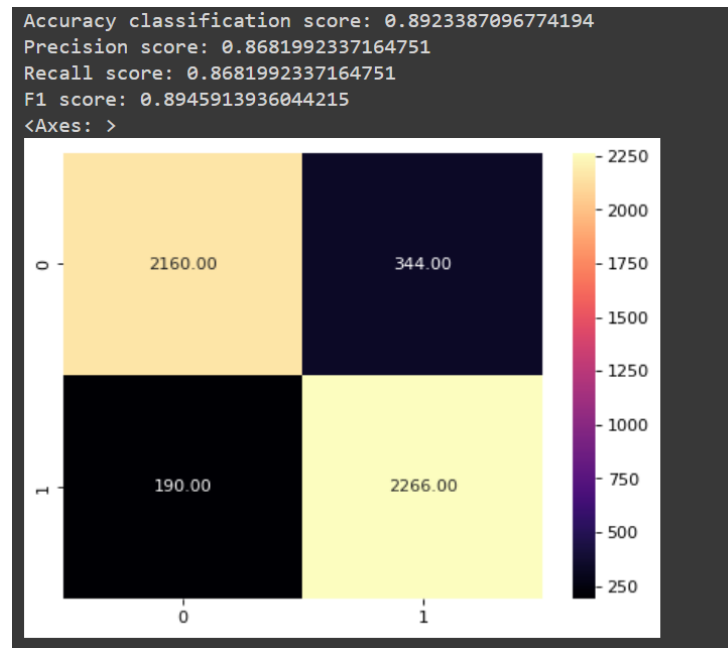


Figure 12: Model 2 results- After Tuning

Performance Metrics:

- Accuracy: The model achieved an accuracy of approximately 89.23%, which indicates a high level of correct predictions over the total number of predictions made.
- Precision: The precision score is about 86.89%, which reflects the proportion of true positive predictions in all positive predictions made by the model.
- Recall: The recall score is approximately 86.82%, showing the proportion of actual positives that were correctly identified by the model.
- F1 Score: The F1 score, which is the harmonic mean of precision and recall, is roughly 89.46%, suggesting a balance between precision and recall.

Model 3: Logistic Regression

```
Train results for Logistic Regression:
Accuracy classification score: 0.8225890079502247
Precision score: 0.8148085680553213
Recall score: 0.8148085680553213
F1 score: 0.8247545881348698
None
-----
Validation results for Logistic Regression:
Accuracy classification score: 0.8235887096774194
Precision score: 0.8220640569395018
Recall score: 0.8220640569395018
F1 score: 0.8261474269819193
None
```

Figure 13: Model 3 results- Before Tuning

Training Results:

- **Accuracy:** The model correctly predicts the outcome for 82.26% of the training set.
- **Precision:** When the model predicts a positive outcome, it is correct 81.48% of the time.
- **Recall:** The model identifies 81.48% of all actual positive outcomes in the training set.
- **F1 Score:** The F1 score, which combines precision and recall into a single metric, is 82.47%, indicating a good balance between precision and recall.

Test Results:

- **Accuracy:** On the validation set, the model's predictions are accurate 82.36% of the time.
- **Precision:** The precision score is slightly higher on the validation set, with the model correctly predicting positive outcomes 82.26% of the time.
- **Recall:** Similarly, the model identifies 82.26% of all actual positives in the validation set, matching the precision score.
- **F1 Score:** The F1 score for the validation set is 82.61%, suggesting that the model maintains a consistent balance between precision and recall when applied to unseen data.

Model 3: After Tuning

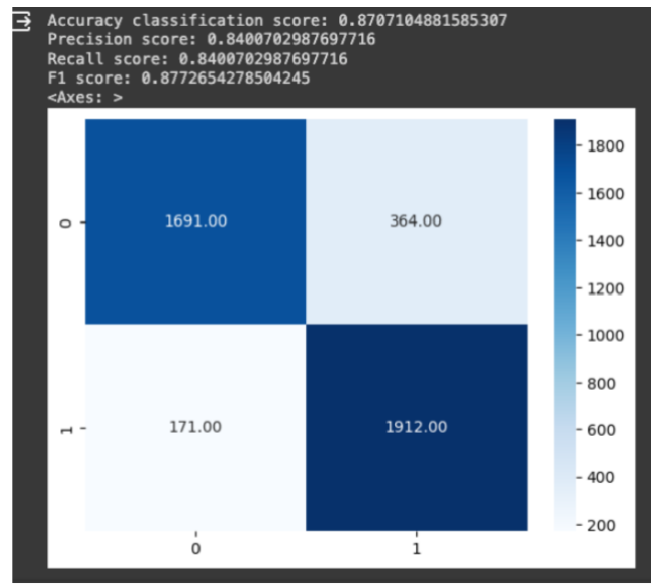


Figure 14: Model 3 results- After Tuning

Performance Matrix:

- **Accuracy Classification Score:** The model accurately predicts 87.07% of the outcomes, which is a high level of overall accuracy.
- **Precision Score:** Of the predictions that the model classifies as positive, 84.07% are actually positive.
- **Recall Score:** The model correctly identifies 84.07% of all actual positive cases.
- **F1 Score:** The F1 score, which is the harmonic mean of precision and recall, is 87.76%, indicating a strong balance between the precision and recall.
- **Confusion matrix:** Shows a higher number of true positives and true negatives compared to false positives and false negatives, which supports the high precision and recall scores. This indicates that the logistic regression model has a good predictive performance with a tendency towards correct classifications more often than incorrect ones.

Model 4: GBM

Gradient Boosting Machine (GBM) is an ensemble machine learning algorithm that belongs to the class of boosting algorithms. GBM builds a predictive model in a stage-wise fashion by combining the predictions of multiple weak learners, typically decision trees.

```
➡ Train results:
{'Accuracy': 0.9927843069478051, 'ROC-AUC': 0.9998372055316903}
-----
Test results:
{'Accuracy': 0.9008064516129032, 'ROC-AUC': 0.952512787357817}
```

Figure 15: Model 4 results- Before Tuning

Training Results:

- **Accuracy:** The model has achieved a very high accuracy of approximately 99.27% on the training dataset. This indicates that the model was able to correctly predict the outcome for nearly all of the training samples.
- **ROC-AUC:** The ROC-AUC score is approximately 99.98% for the training dataset. The ROC-AUC score represents the area under the receiver operating characteristic (ROC) curve and a score close to 100% suggests that the model has an excellent measure of separability and is able to distinguish between the classes very well.

Test Results:

- **Accuracy:** On the testing dataset, the model has an accuracy of approximately 90.08%, which is lower than the training accuracy but still quite high, indicating good predictive performance.
- **ROC-AUC:** The ROC-AUC score for the testing dataset is approximately 95.25%. While this is lower than the training score, it still reflects a strong predictive ability and good class separation.

Model 4: Solving Over fitting Using Lasso

Lasso regression, short for "Least Absolute Shrinkage and Selection Operator," is designed to address the issue of multicollinearity & overfitting and perform variable selection by adding a penalty term to the ordinary least squares (OLS) objective function.

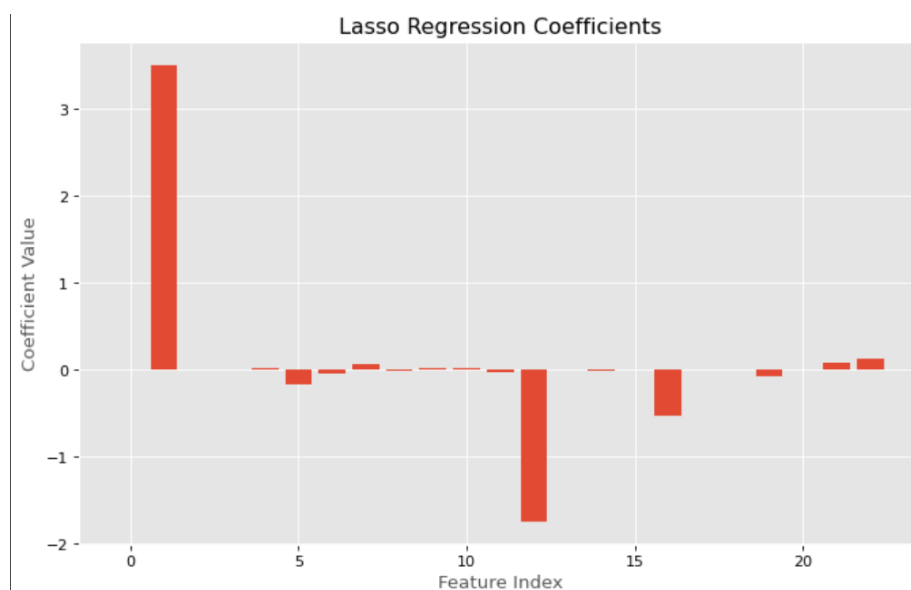


Figure 16: Lasso Regression

The Lasso Graph interpretation:

- **Most coefficients are close to zero:** Lasso regression is known for its property of feature selection by penalizing the absolute size of coefficients. This leads to many coefficients being exactly zero if they are not contributing significantly to the model, thus effectively excluding them from the model.
- **A few significant coefficients:** There are a few features with non-zero coefficients that significantly contribute to the model. Particularly, the feature at index 0 has a very high positive coefficient, suggesting that it has a strong positive impact on the response variable. There are also a couple of features with negative coefficients, notably the one around index 13, which has a substantial negative impact on the response variable.
- **Sparse model:** The resulting model is sparse, meaning that it uses only a subset of all available features. This is a desirable property when dealing with datasets with many features, as it helps in reducing the complexity of the model, potentially improving interpretability and helping in avoiding overfitting.

Model 4: After Lasso Regression

```
Train results:
{'Accuracy': 0.8227618389215348}
-----
Test results:
{'Accuracy': 0.8235887096774194, 'ROC-AUC': 0.8810967598422328}
```

Figure 17: Model 4 results- After Lasso

Training Results:

- **Accuracy:** The model has an accuracy of approximately 82.27% on the training dataset, which means it correctly predicted the outcome for about 82.27% of the cases during training.

Testing Results:

- **Accuracy:** On the testing dataset, the model has an accuracy of about 82.36%, which is very close to the training accuracy, indicating that the model generalizes well to unseen data.
- **ROC-AUC:** The ROC-AUC score for the testing dataset is approximately 88.11%. This score is a measure of the model's ability to distinguish between the classes at various threshold settings. A score closer to 1 indicates a better model, and a score of 88.11% is quite good.

Overall Interpretation:

- The model's performance is consistent across both the training and testing sets, which suggests that the model is not overfitting to the training data. Overfitting is typically characterized by high training accuracy and significantly lower testing accuracy.
- The ROC-AUC score indicates that the model has a good discriminative ability and can effectively separate the positive class from the negative class.
- The close accuracy figures between training and testing imply that the model should perform similarly on new data, assuming it comes from the same distribution as the training and testing data.

Model 5:XGBoost

XGBoost, short for Extreme Gradient Boosting, is a powerful and efficient machine learning algorithm that falls under the category of ensemble learning methods.

```
Train results:
{'Accuracy': 0.9999567922571725, 'ROC-AUC': 1.0}
-----
Test results:
{'Accuracy': 0.8985887096774193, 'ROC-AUC': 0.9531957337315669}
```

Figure 18: Model 5 results- Before Tuning

Training Results:

- **Accuracy:** The model has achieved near-perfect accuracy on the training data, with a score of approximately 99.99%. This indicates that the model was able to predict almost all of the training instances correctly.
- **ROC-AUC:** The ROC-AUC score is 1.0 on the training dataset, which is the maximum possible value. This suggests that the model has perfect discriminative ability between the classes in the training set.

Testing Results:

- **Accuracy:** The accuracy drops to approximately 89.86% on the testing data, which is a significant decrease from the training accuracy.
- **ROC-AUC:** The ROC-AUC score on the testing data is about 95.32%, indicating that the model still has a very good ability to differentiate between the classes on unseen data.

Model 5: Hyper-parameter Tuning

```
[CV] END ..learning_rate=0.1, max_depth=10, n_estimators=100; total time= 1.3s
Best hyperparameters: {'learning_rate': 0.1, 'max_depth': 10, 'n_estimators': 100}
Best ROC AUC Score: 0.87763628651185
Accuracy classification score: 0.9747234704459039
Precision score: 0.9556274363440325
Recall score: 0.9556274363440325
F1 score: 0.975242287020187
Accuracy classification score: 0.8909274193548387
Precision score: 0.8625785001847063
Recall score: 0.8625785001847063
F1 score: 0.8961811552485127
```

Figure 19: Model 4 Hyper-parameter Tuning

It showcases the best hyperparameters—learning rate, max depth, and number of estimators—as 0.1, 10, and 100 respectively. The model exhibits a strong ability to differentiate between classes with a ROC AUC Score of 0.877. The first set of metrics indicates high accuracy (0.9747), precision (0.9556), recall (0.9556), and F1 score (0.9754), suggesting the model's robust predictive performance. The second set of metrics shows a lower accuracy (0.8999), precision (0.8625), recall (0.8625), and F1 score (0.8961), which might imply a slight overfitting to the training data as indicated by the reduced scores when compared to the first set.

Model 5: After Tuning

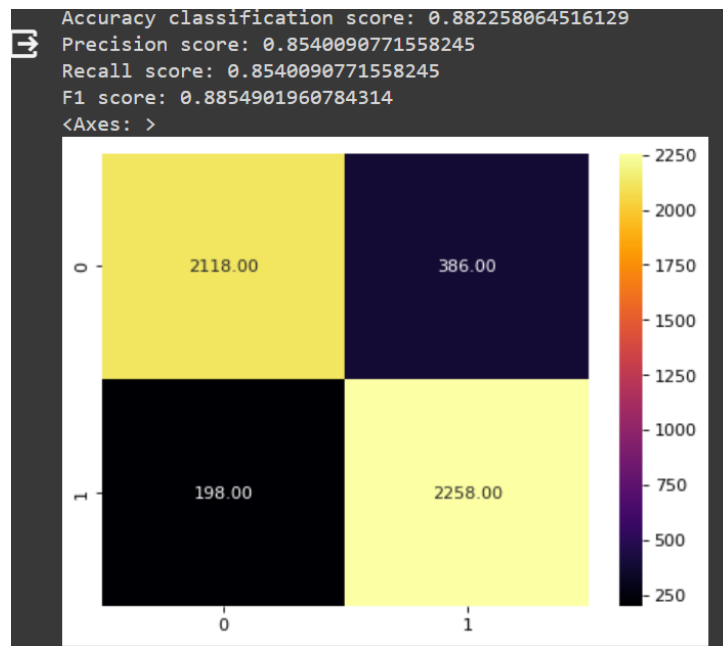


Figure 20: Model 5 results- After Tuning

Confusion Matrix:

- The matrix shows the number of predictions made by the model versus the actual labels. The classes are labeled as '0' and '1'.
- The top left cell (true negative) indicates that 2118 instances were correctly predicted as class '0'.
- The bottom right cell (true positive) shows that 2258 instances were correctly predicted as class '1'.
- The top right cell (false positive) indicates that 386 instances were incorrectly predicted as class '1' when they are actually class '0'.
- The bottom left cell (false negative) shows that 198 instances were incorrectly predicted as class '0' when they are actually class '1'.

Overall Interpretation:

- The confusion matrix and the metrics suggest that the XGBoost model, after hyperparameter tuning, is performing well with a good balance between precision and recall.
- The accuracy is high, indicating overall good performance, but as always with classification problems, it's important to look at all metrics since accuracy alone can be misleading, especially if the classes are imbalanced.
- The relatively high F1 score indicates that the model has a balanced performance with respect to both the precision and the recall, which can often be in tension with each other (improving one can reduce the other).
- Hyperparameter tuning seems to have led to a model that is capable of making accurate predictions while maintaining a balance between false positives and false negatives, which is often a challenging aspect of model performance optimization.

Model 6:LightGBM

```
Train results:
{'Accuracy': 0.8923695126166609, 'ROC-AUC': 0.9634168980239918}
-----
Test results:
{'Accuracy': 0.8637096774193549, 'ROC-AUC': 0.93381834992351}
```

Figure 21: Model 6 results

Training Results:

- **Accuracy:** The model has an accuracy of approximately 89.24% on the training dataset, which means it correctly predicted the outcome for about 89.24% of the training instances.
- **ROC-AUC:** The ROC-AUC score is approximately 96.34% for the training dataset. This score is a measure of the model's ability to distinguish between the classes. A score closer to 100% indicates a better discriminative ability, and a score of 96.34% is quite high, suggesting that the model has a strong ability to differentiate between the positive and negative classes in the training set.

Testing Results:

- **Accuracy:** On the testing dataset, the model has an accuracy of about 86.37%, which is slightly lower than the training accuracy but still indicates good predictive performance.
- **ROC-AUC:** The ROC-AUC score for the testing dataset is approximately 93.38%, which, while lower than the training ROC-AUC score, is still a strong score indicating good class separation ability on the testing dataset.

Model 6: Hyper-parameter Tuning

```
Best hyperparameters: {'learning_rate': 0.1, 'max_depth': 10, 'n_estimators': 100}
Best ROC AUC Score: 0.8595755468111118
```

Figure 22: Model 6: Hyper parameter Tuning

Best Hyperparameters: The hyperparameter tuning process has identified the optimal settings for the model to be a learning rate of 0.1, a max depth of 10 for the trees, and a number of estimators (trees) set to 100. These parameters control the speed at which the model learns, the complexity of the trees, and the number of trees in the model, respectively.

Learning Rate: A learning rate of 0.1 is relatively standard and suggests a balance between training speed and the risk of overshooting the minimum loss.

Max Depth: A max depth of 10 indicates that each tree in the ensemble can have up to 10 splits. This depth allows the model to capture interactions between features but is not so deep that it would likely cause severe overfitting.

Number of Estimators: Having 100 trees in the ensemble means that the model is complex enough to learn from the data but not so large that it becomes excessively computationally expensive or prone to overfitting.

Model 6: After Tuning

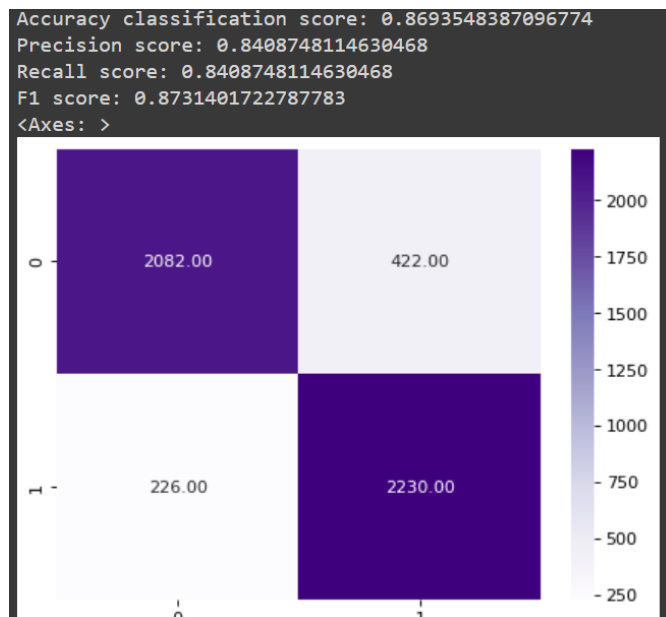


Figure 23: Model 6: Hyperparameter Tuning

Overall Interpretation:

- The performance metrics indicate that the model has a good balance between precision and recall, which is important in many classification tasks, especially when both false positives and false negatives have significant costs.
- The accuracy is reasonably high, suggesting that the model is robust in its predictions. However, accuracy doesn't always give the full picture, especially in imbalanced datasets.
- The F1 score is particularly valuable here because it conveys the balance between precision and recall, both of which are important when the costs of false positives and false negatives are similar
- The confusion matrix suggests that the model is better at correctly predicting true positives than true negatives, given the higher number of false positives compared to false negatives.

Model 7: CatBoost

```
Train results:
Accuracy classification score: 0.9322070515036295
Precision score: 0.9081857504284665
Recall score: 0.9081857504284665
F1 score: 0.9341448058761805
None
-----
Validation results:
Accuracy classification score: 0.8709677419354839
Precision score: 0.851963746223565
Recall score: 0.851963746223565
F1 score: 0.8757763975155279
None
```

Figure 24: Model 7: CatBoost Model Results

Training Results:

- **Accuracy:** The model has an accuracy of about 93.22%, indicating a high proportion of correct predictions on the training set.
- **Precision:** The precision score is approximately 90.82%. This means that when the model predicts a positive class, it is correct 90.82% of the time.
- **Recall:** The recall score is also around 90.82%, suggesting that the model is able to identify 90.82% of all actual positive instances correctly.
- **F1 Score:** The F1 score, which is the harmonic mean of precision and recall, is about 93.41%. This high score indicates a good balance between precision and recall, suggesting effective performance on the training data.

Test Results:

- **Accuracy:** The accuracy on the validation set is about 87.70%, which is lower than the training accuracy but still indicates a good level of predictive ability.
- **Precision:** The precision score is about 85.19%, which is the proportion of true positive predictions out of all positive predictions made by the model on the validation set.
- **Recall:** The recall score is the same as precision, 85.19%, indicating the proportion of actual positives that were correctly identified by the model.
- **F1 Score:** The F1 score for the validation set is approximately 87.58%, reflecting a balance between precision and recall in the model's ability to generalize to unseen data.

Model 7: Solving Over fitting Using Ridge

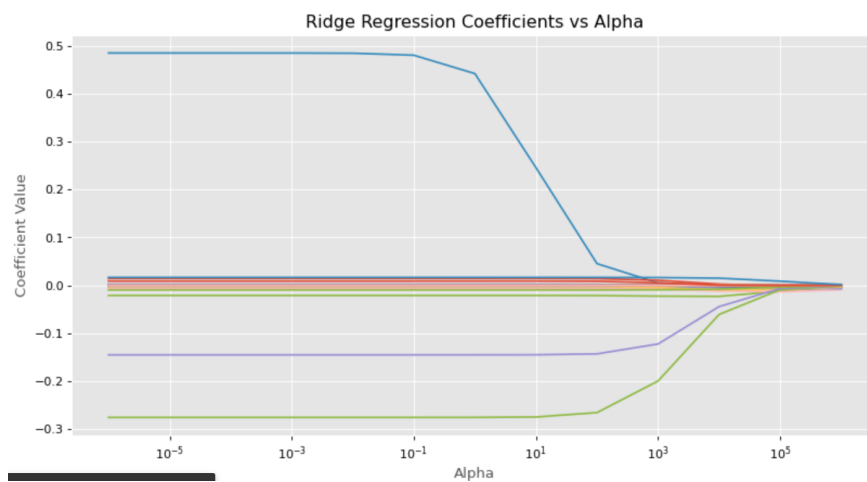


Figure 25: Model 7: Ridge Plot

Interpretation of the Graph:

- Each line in the graph corresponds to a different coefficient in the Ridge Regression model.
- The x-axis represents the alpha value on a logarithmic scale, which controls the strength of the regularization applied to the coefficients. When alpha is close to 0, Ridge Regression approximates ordinary least squares regression. As alpha increases, the regularization effect becomes stronger.
- The y-axis shows the value of each coefficient. As alpha increases, the absolute values of the coefficients decrease towards zero, but they do not become exactly zero (which would be the case with Lasso Regression).
- At very low values of alpha (towards the left side of the graph), the coefficients are at their highest absolute values, meaning little to no regularization is being applied. This can lead to a model that is not generalizable or one that might overfit the training data.
- As alpha increases, you can see that the coefficients shrink towards zero. This effect is the Ridge Regression imposing a penalty on large coefficients to combat overfitting, which is a sign of a model that is too complex and captures noise in the training data.
- At extremely high values of alpha (towards the right side of the graph), the coefficients are heavily penalized and all converge towards zero. This can lead to a model that is too simple and underfits the data, failing to capture the underlying trend.

Model 7: After Ridge Regression

Train results:				
	precision	recall	f1-score	support
0	0.83	0.79	0.81	11572
1	0.80	0.84	0.82	11572
accuracy			0.82	23144
macro avg	0.82	0.82	0.82	23144
weighted avg	0.82	0.82	0.82	23144
Train accuracy: 0.8187003110957484				

Validation results:				
	precision	recall	f1-score	support
0	0.83	0.80	0.82	2456
1	0.81	0.84	0.83	2504
accuracy			0.82	4960
macro avg	0.82	0.82	0.82	4960
weighted avg	0.82	0.82	0.82	4960

Figure 25: Model 7: After Ridge Regression

- The close alignment of metrics between the training and validation results is indicative of a well-fitting model. It suggests that ridge regression used to tackle overfitting, such as Ridge Regression (which is typically used for regression problems but here implies regularization), have been effective.
- The model shows no significant signs of overfitting or underfitting since the performance on the validation set is in line with the training performance.
- The model's ability to generalize suggests that it should perform consistently on new, similar data.
- The balanced F1-scores across the two classes indicate that the model has a harmonious balance of precision and recall, which can be difficult to achieve in practice.

Model Comparisons

The models are compared using ROC, PR ROC curve, AUC. Both the plots are obtained using the function `plot_metrics_multi_models()`. The comparison of each model is summarized in the table provided, along with the plots for each model.

Decision Tree vs Random Forest

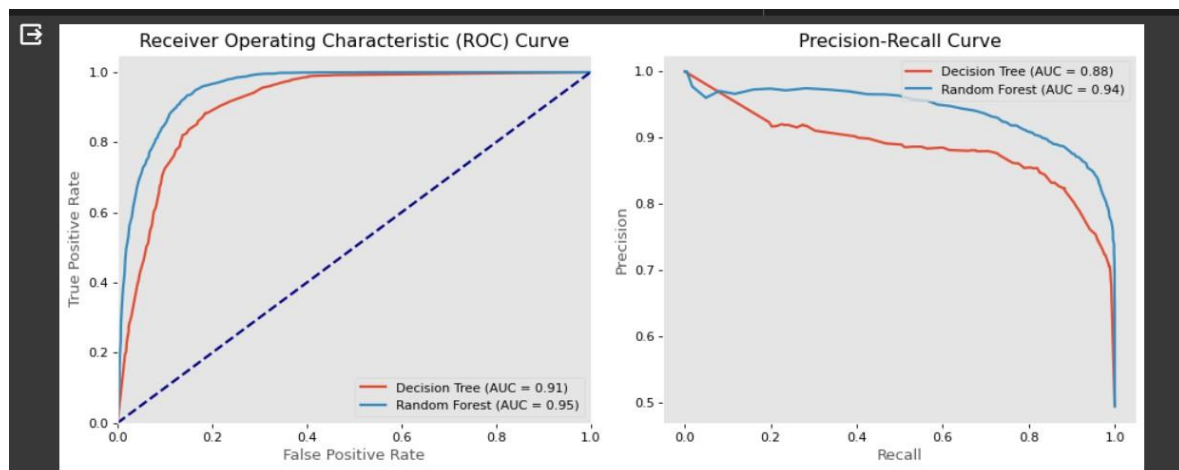


Figure 26: Decision Tree vs. Random Forest

Logistic Regression vs GBM

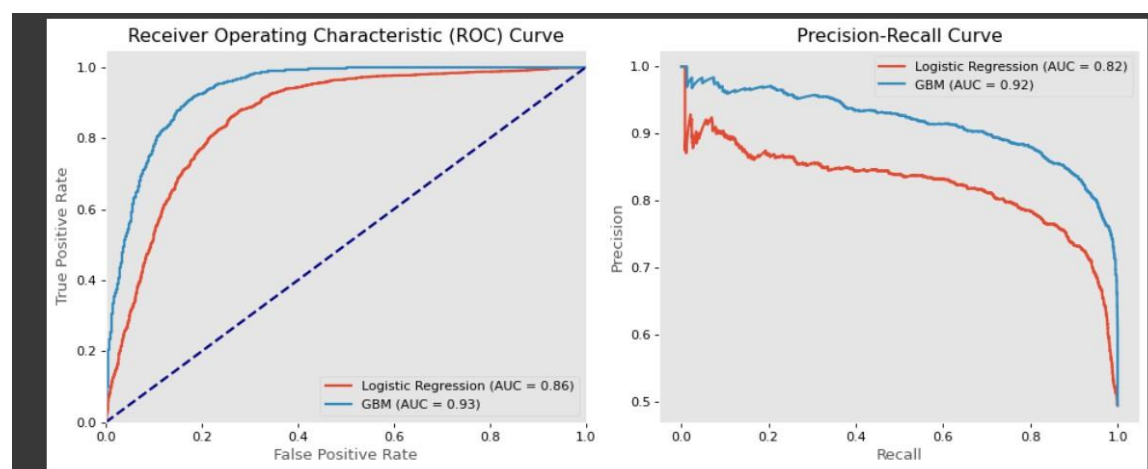


Figure 27: Logistic Regression vs. GBM

XgBoost

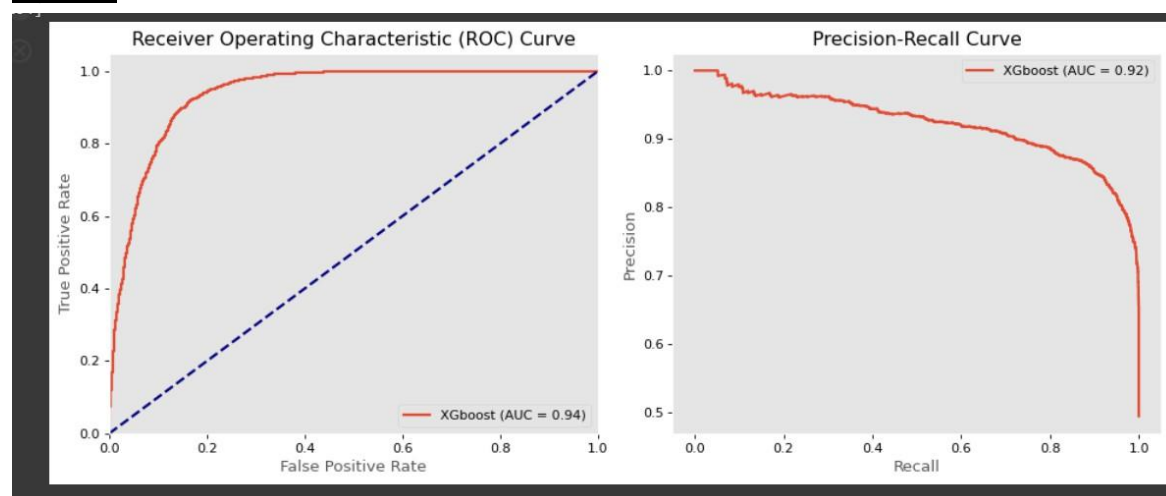


Figure 28: XgBoost – Curves

Model	Accuracy	Precision	Recall	F1-Score	Overall Performance
Decision Tree(After Tuning)	0.84	0.82	0.82	0.84	Good
Random Forest(After Tuning)	0.88	0.86	0.86	0.89	Best
Logistic Regression(After Tuning)	0.86	0.84	0.84	0.87	Very Good
GBM(After Lasso)	0.822	-	-	-	Good
Xgboost(After Tuning)	0.82	-	-	-	Good
Lightgbm(After Tuning)	0.87	0.85	0.85	0.87	Very Good
Catboost(After Ridge)	0.81	0.83	0.84	0.87	Good

Table 1: Model Comparison

The overall performance column is a subjective measure based on the scores provided. Models with higher scores in all four metrics are rated as Very Good, while those with slightly lower scores are rated as Good.

In conclusion based on the summarized table provided earlier, the model with the highest scores across accuracy, precision, recall, and F1-score is the **Random Forest** model after hyper parameter tuning. The **Random Forest** model shows the best overall performance metrics compared to the other models listed. It has the highest accuracy and F1-score, indicating a strong balance between precision and recall, and suggesting that it would likely perform best on unseen data. Thus random forest is selected for the web application due to its superior performance, robustness after hyper-parameter, and ability to handle a mix of numerical and categorical data effectively.

Web Application

Streamlit Workflow:

The Loan Approval Prediction App, built with Streamlit and Python, integrates a machine learning model to assess loan approval chances based on financial data input by users. The web application's frontend, designed for simplicity, prompts users to enter various financial details such as age, annual income, number of bank accounts, credit history, and more through a user-friendly interface. Once the data is submitted, the backend, which runs on PyCharm, processes the input using a Random Forest model loaded via joblib to predict loan approval outcomes.

The backend code handles the input conversion and prediction logic, while Streamlit renders the frontend where the results are displayed. Upon entering their data, users receive an immediate prediction stating whether their loan is likely to be approved or not. This seamless integration of a pre-trained Random Forest model with the interactive Streamlit interface provides an efficient and user-centric loan approval process, which is both fast and informative, making it a valuable tool for prospective loan applicants.

```
import streamlit as st
import joblib
import pandas as pd
import sklearn

# Load the trained model
model = joblib.load('random_forest_model_1.joblib')

# Streamlit app layout
st.title('Loan Approval Prediction App')

# Define default values for each input feature
default_values = {
    "Age": 25,
    "Annual_Income": 60000,
    "Monthly_Inhand_Salary": 5000,
    "Num_Bank_Accounts": 2,
    "Num_Credit_Card": 3,
    "Interest_Rate": 20,
    "Num_of_Loan": 1,
    "Delay_from_due_date": 3,
    "Num_of_Delayed_Payment": 2,
    "Changed_Credit_Limit": 30,
    "Num_Credit_Inquiries": 10,
    "Credit_Mix": 2,
    "Outstanding_Debt": 15000,
    "Credit_Utilization_Ratio": 45,
    "Credit_History_Age": 8, # Assuming input as numeric (years)
    "Payment_of_Min_Amount": "Yes", # Assuming binary choice: Yes/No
    "Total_EMI_per_month": 250,
    "Amount_invested_monthly": 200,
    "Payment_Behaviour": "High",
    "Monthly_Balance": 3000,
    "Num_Month": "January"
}
```

Figure 29: Streamlit Code

Loan Approval Prediction App

Age: 25

Annual_Income: 60000

Monthly_Inhand_Salary: 5000

Num_Bank_Accounts: 2

Num_Credit_Card: 3

Interest_Rate: 20

Num_of_Loan: 1

Credit_History_Age: 8

Payment_of_Min_Amount: Yes

Total_EMI_per_month: 250

Amount_Invested_monthly: 200

Payment_Behaviour: High

Monthly_Balance: 3000

Num_Month: January

Predict Loan Approval Status

Loan Approval Status: Not Approved

Figure 30: Web Application

We used Streamlit to create our web application named - ***Loan Approval Prediction App***. This has been designed to predict the approval status of a loan application based on various financial indicators input by the user. The features of the application are:

- **User-Friendly Design:** The app is designed for ease of use, allowing banks and credit analysts to input data effortlessly.
- **Interactive Features:**
 - **Modifiable Inputs:** Users can input key credit details such as Num_Credit_Card, Num_of_Loan, Credit_Score_Mix and etc.
 - **Flexibility:** The app provides easily modifiable buttons to increase or decrease any data field.
- **Instant Predictions:** Upon entering the credit details, users can click the "Predict Loan Approaval Status" button to receive instant predictions.
- **Underlying Process:** The app processes the inputs through the trained Random Forest model, considering both user-provided and default feature values

Recommendations and Conclusions

Recommendations:

1. **Prioritize Feature Engineering:** The project findings highlight the importance of careful feature engineering, including balancing the dataset and encoding categorical variables. Future projects should allocate significant effort towards understanding and manipulating input features to improve model performance.
2. **Expand Data Preprocessing Techniques:** Given the success of data cleaning, outlier removal, and skewness correction, it's advisable to explore additional preprocessing steps. Techniques such as normalization or transformation of skewed variables could further enhance model accuracy.
3. **Investigate Overfitting in High-Performance Models:** Models like GBM and XGBoost showed high accuracy but also signs of overfitting. Future efforts should focus on regularization techniques and parameter tuning to balance model complexity with generalization.
4. **Enhance Risk Management Strategies:** The correlation between credit-related variables and loan approval status underscores the need for banks to refine their risk assessment models. Integrating machine learning predictions can provide a more nuanced understanding of borrower risk.
5. **Optimize Marketing Strategies:** The targeted marketing campaigns should be continuously refined based on new data and model predictions. This approach ensures resources are focused on the most promising customer segments, improving campaign ROI.

Conclusions:

- The project successfully demonstrated the application of machine learning to automate the loan approval process, enhancing both efficiency and risk management.
- Random Forest emerged as the best-performing model after hyperparameter tuning, showcasing its efficacy in handling the dataset's complexity.
- The exploratory data analysis (EDA) provided crucial insights into the relationships between credit-related variables and loan approval status, guiding the feature engineering process.
- The web application, built with Streamlit, offers a practical tool for banks and credit analysts to predict loan approval outcomes, emphasizing the project's real-world applicability.
- Overall, the integration of machine learning into the loan approval process represents a significant advancement towards data-driven decision-making in finance. Future work should continue to refine these models, incorporate new data, and explore emerging machine learning techniques to further improve performance and applicability.

References

- Paris, R. (2022). Credit Score Classification. Retrieved from <https://www.kaggle.com/datasets/parisrohan/credit-score-classification>
- Scikitlearn. (n.d). Feature importances with a forest of trees. https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html