Data 245 - Machine Learning

Fall, 2022

Project Report

Credit Risk Analysis

Dhruv Jain, Edward Montoya, Nghi Nguyen, Tam Huynh

Contents

1. Abstract

Over the last few decades, financial management has been a huge issue for individuals globally, especially when the global financial crisis occurred in 2008. Credit risk assessment is crucial in helping financial institutions define their policies and standards. Through appropriate ruling, only individuals with low credit risk are lent big loans, which minimizes the possibility of financial breakdown in the future. By applying machine learning, this project aims to classify people's credit card status from input information on their personal information, including salary, spending habits, and loans. This paper examines four different machine learning algorithms: Logistic Regression, K-nearest neighbors, eXtreme Gradient Boosting, and Random Forest. This research bridges previous papers by applying an Ensemble Voting Classifier technique to the performance of the mentioned models to build a robust credit score classification classifier. The oversampling method Synthetic Minority Oversampling Technique (SMOTE) and the feature selection technique Recursive Feature Elimination, Cross-Validated (RFECV) are used to improve the efficiency of the predictions. A comparative analysis is performed between the naive models, models with the application of SMOTE, and models with the employment of SMOTE and RFECV. Random Forest model with sampling and feature selection techniques outperforms the other with 81.25% accuracy after performing hyper-parameter tuning with Random Search.

2. Introduction

Credit is the ability of an individual to borrow money or goods or services and pay back at a later time, and credit cards are one of the most popular forms of credit. Undeniably, credit plays an important role in pushing up the economy in the U.S. as people tend to spend more when they use credit cards than when they use cash. However, if financial organizations don't take credit risk assessment seriously, the economy can bubble and explode. The target of the project is to find the best machine learning method for credit risk modeling using a basic bank details dataset that contains a vast array of credit-related information. The model classifies credit scores into three categories: 0 stands for poor, 1 stands for standard, and 2 stands for good. This will help financial institutions determine qualified candidates for any specific loans.

The objectives of the project are shown as follows:

- Conduct data preparation (e.g., data cleaning, data preprocessing) to clean and transform the dataset.
- Conduct data exploration to understand the dataset and observe any patterns or possible relationships among the variables.
- Implement a variety of machine learning algorithms, including Logistic Regression, K-Nearest Neighbor, Random Forest, XGBoosting, and Ensemble Voting Classifier.
- Implement all algorithms again with the synthetic minority oversampling technique (SMOTE) function. The SMOTE function will re-distribute the training and testing dataset with balanced portions of target categories.
- Perform feature selection technique Recursive Feature Elimination, Cross-Validated (RFECV) and implement all algorithms one last time
- Evaluate and compare different models which are:
    - pure (no SMOTE function and feature selection)

        - applied the SMOTE function
        - applied the SMOTE function and the RFECV feature selection technique.
    - Deploy the models

3. Problem Statement

3.1 Project Background

In 2008, a financial crisis was triggered in the U.S. that was also known as the Great Recession. The cause of the crisis is believed to be the collapse of the subprime mortgage market, meaning individuals with below-average credit scores took out mortgages and did not have the ability to pay the loans afterward. The situation occurred due to poor risk assessment as financial organizations and institutions kept lending money to high-risk-credit individuals.

The dataset for this project consists of 100,000 records of credit-related information. With this dataset, we are able to find out the best machine learning algorithm for the credit risk modeling problem. Ensemble Voting Classifier and SMOTE function are some highlights of the project as no research papers we have read use the Ensemble Voting Classifier or combine the Ensemble Voting Classifier and SMOTE.

3.2 Literature Survey

Gahlaut et al. (2017) propose data mining models including a Decision Tree, Random Forest, Neural Network, Support Vector Machine, Adaptive Boosting Model, and Linear Regression to determine customers' ability to pay back their credit loans by classifying them as "Good credit" or "Bad credit." Among the models, Random Forest returns the best result with high accuracy even when its running time is quite high. The authors use a dataset from the UCI machine learning data repository, which contains a variety of features, such as demographic characteristics (e.g., race, age, occupation), credit, rate, etc.

In the research by Machado and Karray (2022), hybrid algorithms are compared with the six individual algorithms, Adaboost, Gradient Boosting, Decision Tree, Random Forest, Support Vector Machine, and Artificial Neural Network in predicting commercial customers' credit scores of a large bank in the U.S. Hybrid algorithms combine supervised and unsupervised machine learning models; for example, first, it uses k-Means to cluster the data then applies Adaboost to obtain the prediction. The result shows that hybrid models outperform individual models in terms of mean absolute error, explained variance, and mean squared error.

Laborda and Ryoo (2021) work on four different machine learning algorithms - Random Forest, K-nearest neighbor, Logistic Regression, and Support Vector Machine to find out the best model for the credit score classification problem. The authors offer three different feature selection methods - one filter method that uses the Chi-squared test and correlation analysis and two wrapper methods that use forward and backward stepwise selection. Their research paper shows that forward stepwise selection outperforms the other two methods, especially when it collaborates with the Random Forest model.

Moscato et al.(2021) focus on P2P credit risk modelings. P2P is a financial platform that allows people to lend their money to others without going through a bank, and the models try to predict if the borrowers have the ability to pay back the P2P loan. A benchmarking study is proposed and tested by combining various machine learning algorithms and sampling methodologies and performance comparison XAI tools. Algorithms applied include Logistic Regression, Random Forest, and Multi-layer perceptron, and sampling approaches consist of Random Under Sampling (RUS), IHT, Random Over Sampling (ROS), SMOTE, ADASYN, SMOTE-TOKEN, and SMOTE-EN. In conclusion, Logistics Regression is the best approach for over-sampling technology ROS, Logistic Regression is the best algorithm for hybrid sampling methods SMOTE-TOKEN, and Random Forest is the best for under-sampling methodology.

In the work of Singh (2017), twenty-five major classification techniques of individuals and ensembles are examined to determine the best approaches for the credit score modeling problem. Some of the algorithms are Multilayer perceptron, Logistic, Random Forest, Bagging, Artificial neural networks, and so forth. For individuals, Multilayer perceptron has the best performance in terms of AUC, and Random Forest is the runner-up with ROC 0.178. For ensembles, Random Forest and Bagging form the best combination with the highest accuracy score among all ensembles.

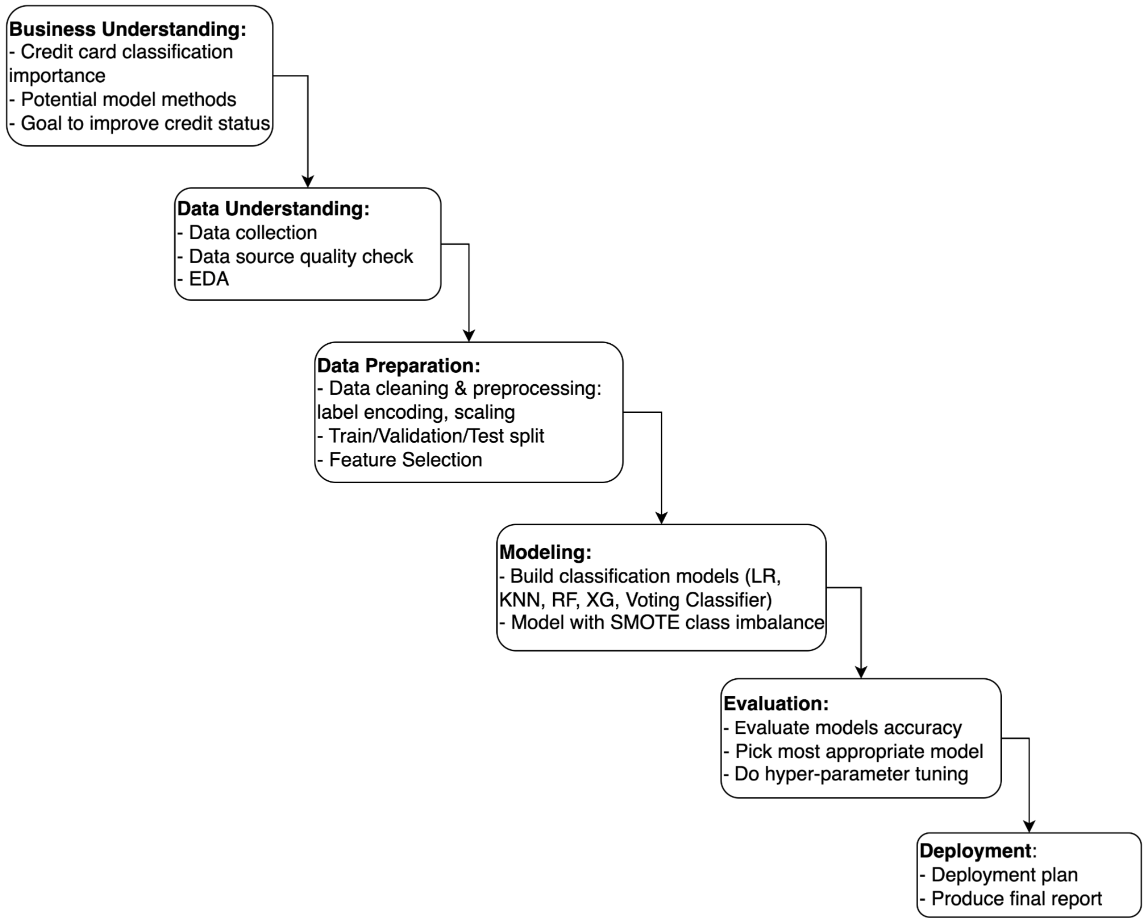
Trivedi (2020) implements five machine learning classifiers (Random Forest, Support Vector Machine, Decision Tree, Bayesian, and Naïve Bayes) and three feature selection methods (Gain-Ratio, Information Gain, and Chi-Square) to identify the best approach for the credit score classification model. In general, Random Forest is the best algorithm with 93% accuracy, and Chi-Square appears to be the most informative feature selection for all machine learning models. A disadvantage of Random Forest is the slightly high running time.

4. Project Management - CRISP DM

The project management follows the CRoss Industry Standard Process for Data Mining, also known as CRISP-DM. Figure 1 shows six phases of the process method.

Figure 1

CRISP-DM Process WorkFlow


Business Understanding:
- Credit card classification importance
- Potential model methods
- Goal to improve credit status
Data Understanding:
- Data collection
- Data source quality check
- EDA
Data Preparation:
- Data cleaning & preprocessing: label encoding, scaling
- Train/Validation/Test split
- Feature Selection
Modeling:
- Build classification models (LR, KNN, RF, XG, Voting Classifier)
- Model with SMOTE class imbalance
Evaluation:
- Evaluate models accuracy
- Pick most appropriate model
- Do hyper-parameter tuning
Deployment:
- Deployment plan
- Produce final report

The first phase is Business Understanding. It is vital to have a credit card classification system to classify a credit card status. Classifying credit cards based on a customer's income and spending habits, loans, and other conditions can help prevent economic issues. After understanding the business goal, literature surveys are being done to understand the potential model approaches that can be used to solve this problem.

Data Understanding first needs to collect the data based on the scope of credit risk analysis. Data is first collected through the Kaggle dataset, then checked on source and quality.

After that, appropriate exploratory data analysis was done to better understand the data, potential outliers, categorical and numerical variables, and the target features.

Next is Data Preparation. After understanding data types, values, and distribution, appropriate data cleaning and preprocessing have been considered for processing the data. Numerical normalization and categorical encoding are also applied for the possibility of model prediction and better performance. Furthermore, the preprocessed data is also split into training, testing, and validation sets. Finally, feature selection has been applied to get the best feature for the model.

The fourth step, which is Model development, is where data has been trained and tested on four different machine learning models. Ensemble voting classification, which is our unique technique, has also been applied to try to improve the performance of the project.

After the modeling phase comes the Evaluation. Based on different metrics, such as accuracy, precision, recall, and F1-score, each model is evaluated for accuracy and complexity performance to see which meets the business objectives the most. Hyper-parameter tuning is also being done by using Random Grid Search.

The last phase is deployment. After reviewing the whole project, the code and implementation are wrapped up and uploaded to GitHub for public access.

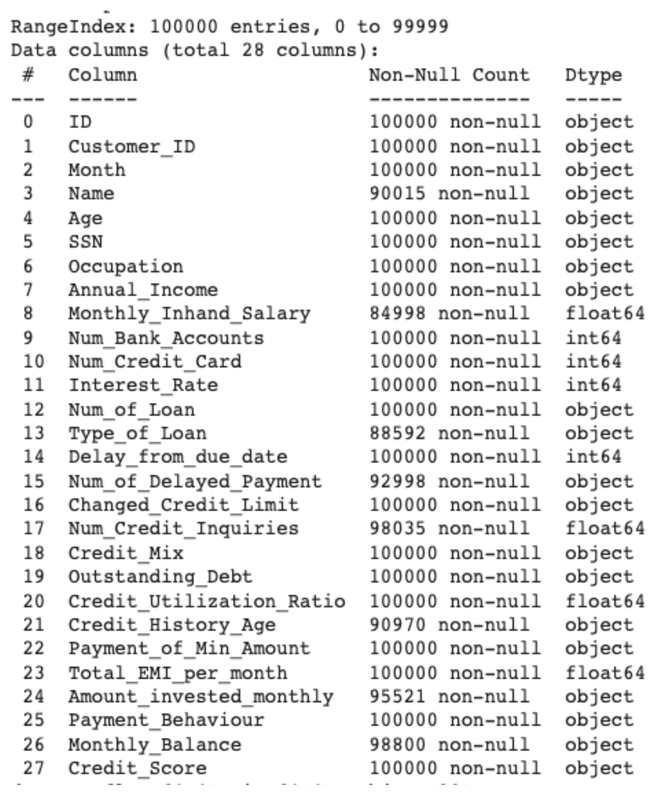
5. Exploratory Data Analysis

5.1 Data Exploration

The data, Credit score classification, was originally collected by Paris (2022) on Kaggle. Looking at the data information, the dataset consists of 100,000 rows and 28 columns. There are 20 categorical features and 8 numerical features. The target feature is ‘Credit_Score,’ which has an object type. There are eight columns with null values. These columns will be taken into consideration in the next steps. Figure 2 shows the overall description of the dataset.

Figure 2

Credit Score Data Types and Null Values Information

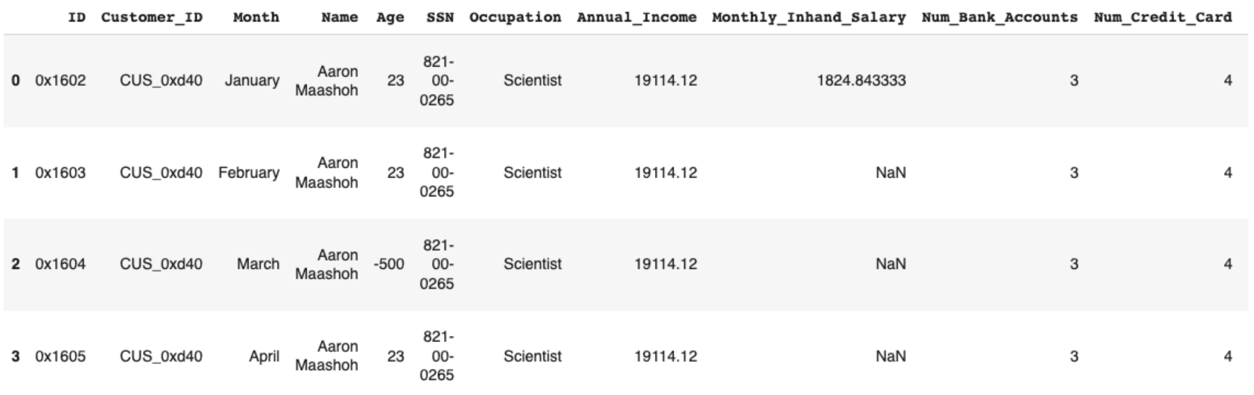


```
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 28 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   ID                        100000 non-null  object
 1   Customer_ID               100000 non-null  object
 2   Month                     100000 non-null  object
 3   Name                      90015 non-null   object
 4   Age                       100000 non-null  object
 5   SSN                       100000 non-null  object
 6   Occupation                100000 non-null  object
 7   Annual_Income             100000 non-null  object
 8   Monthly_Inhand_Salary     84998 non-null   float64
 9   Num_Bank_Accounts         100000 non-null  int64
 10  Num_Credit_Card           100000 non-null  int64
 11  Interest_Rate             100000 non-null  int64
 12  Num_of_Loan               100000 non-null  object
 13  Type_of_Loan              88592 non-null   object
 14  Delay_from_due_date       100000 non-null  int64
 15  Num_of_Delayed_Payment    92998 non-null   object
 16  Changed_Credit_Limit      100000 non-null  object
 17  Num_Credit_Inquiries      98035 non-null   float64
 18  Credit_Mix                100000 non-null  object
 19  Outstanding_Debt          100000 non-null  object
 20  Credit_Utilization_Ratio  100000 non-null  float64
 21  Credit_History_Age        90970 non-null   object
 22  Payment_of_Min_Amount     100000 non-null  object
 23  Total_EMI_per_month       100000 non-null  float64
 24  Amount_invested_monthly   95521 non-null   object
 25  Payment_Behaviour         100000 non-null  object
 26  Monthly_Balance           98800 non-null   object
 27  Credit_Score              100000 non-null  object
```

Figure 3 shows a part of the data for sample inspection.

Figure 3

Credit Score Data Sample



	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card
0	0x1602	CUS_0xd40	January	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	1824.843333	3	4
1	0x1603	CUS_0xd40	February	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	NaN	3	4
2	0x1604	CUS_0xd40	March	Aaron Maashoh	-500	821-00-0265	Scientist	19114.12	NaN	3	4
3	0x1605	CUS_0xd40	April	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	NaN	3	4

There are three values for the target Credit_Score feature, which are ‘Poor’, ‘Standard,’ and ‘ Good.’ The distribution of the target is shown as a donut pie chart in figure 4 below. Clearly, there is an imbalance among the three types, with ‘Poor’ credit status accounting for 28998 rows, ‘Standard’ credit accounts for 53174 rows, and the rest is ‘Good’ credit with 17828 records. Being aware of this target imbalance, there will be an idea to apply target resampling techniques to solve this problem.

Figure 4

Credit Score Target Feature Distribution

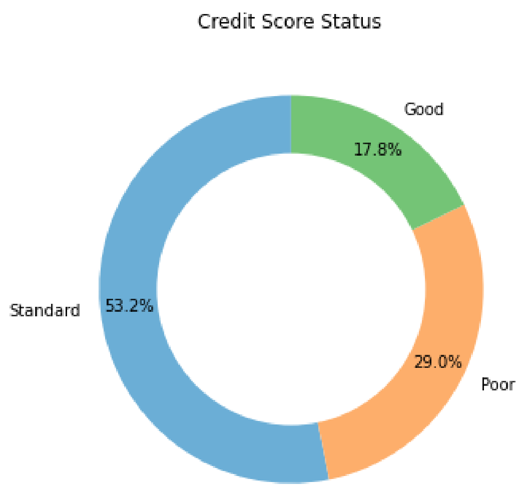


Figure 5 shows how outliers can be detected through a boxplot for each class of the target feature and potentially the correlation between the target and the ‘Monthly_Inhand_Salary’ feature. From the plot, ‘Poor’ targets more outliers in the high salary range, then comes ‘Standard’ status. ‘Good’ credit status people do not have any outliers in a high salary. In this case, we can see the possible positive correlation between a high monthly salary and good credit status.

Figure 5

Monthly Inhand Salary Boxplot by Credit Score Status

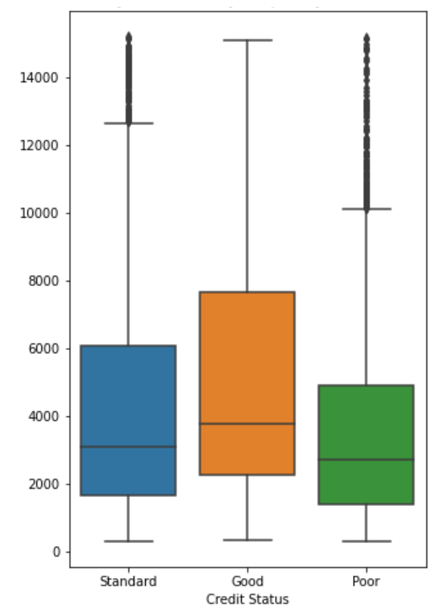


Figure 6 shows the number of credit cards each customer has for each credit card status. There are a lot of outliers going on in this particular feature, and the preprocessing step will take care of it. However, by looking at the distribution, ‘Good’ status credit cards tend to have fewer outliers than the other two classes.

Figure 6

Number of Credit Cards Boxplot by Credit Score Status

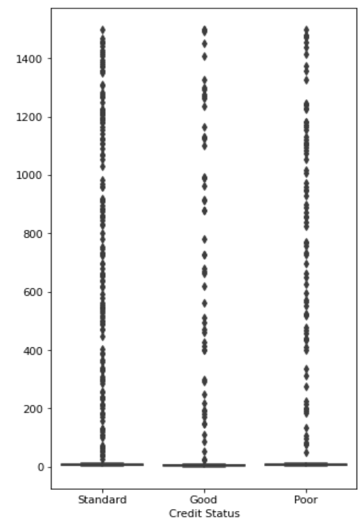


Figure 7 shows the distribution of the number of days delayed after the due date by each of the credit card class statuses. It is taken from the figure that ‘Good’ status people are less likely to delay their payment after 30 days. The better the credit status is the less number of days the payment is delayed.

Figure 7

Number of Days Delayed From Due Date by Credit Status

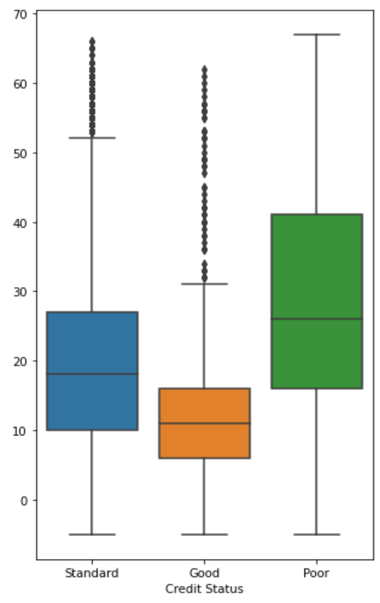
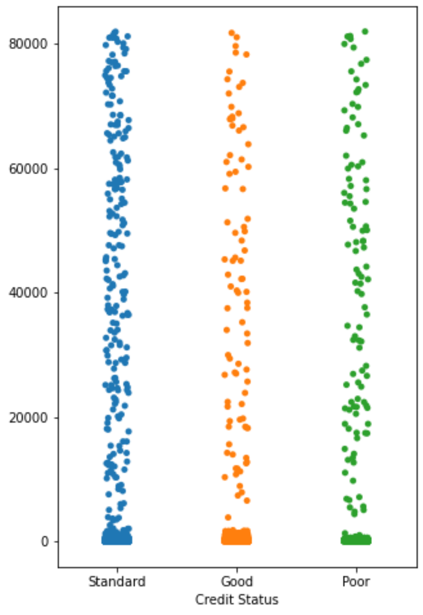


Figure 8 shows the distribution of equated monthly installments by credit status in a strip plot. As inferred from the figure, ‘good’ credit card status has the least outliers, and ‘poor’ has fewer outliers than ‘standard’. It can be inferred that ‘good’ credit might have the least loan to pay every month, and ‘standard’ credit has the most.

Figure 8

Equated Monthly Installment by Credit Status

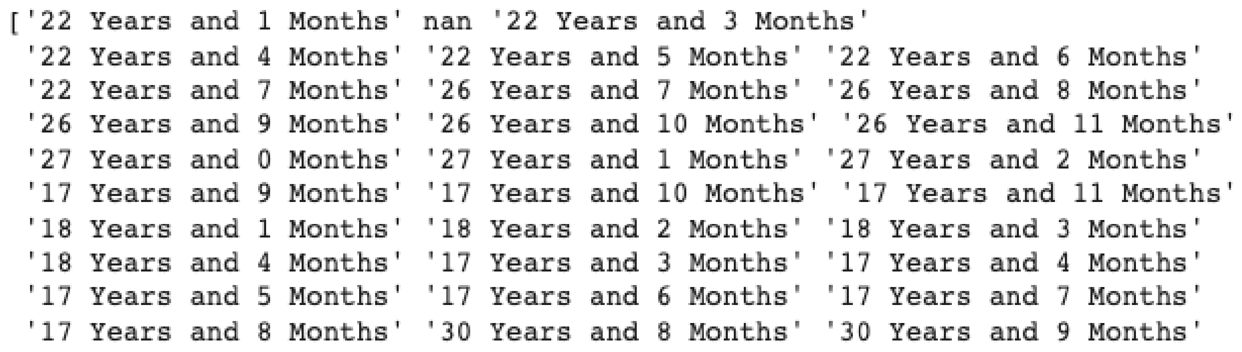


When comparing Figure 5, 6, 7, and 8, we can see that their range value differs from one another, with ‘Monthly_Inhand_Salary’ having the highest value of around 14,000 while ‘Delay_from_due_date’ has the highest value of around 1,400, ‘Num_Credit_Card’ max value is around 70, and ‘Total_EMI_per_month’ highest value is around 80,000. Inspecting three numerical features raises the idea that there is a need for normalizing the numerical variables so that different features can be weighted similarly without changing the differences in values within each feature.

After understanding some numerical features, we also try to understand the categorical features. Figure 9 shows some sample values for credit card history age. Notice that it is in years and months, which brings the idea to convert that into months values only, eventually converting to a numerical variable and normalizing it.

Figure 9

Credit_History_Age Sample Values

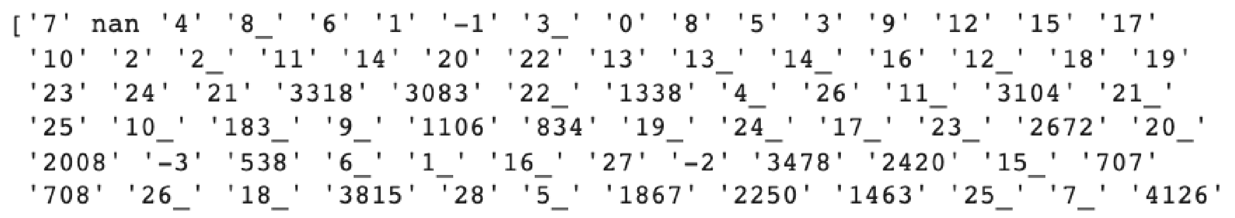


```
['22 Years and 1 Months' nan '22 Years and 3 Months'
 '22 Years and 4 Months' '22 Years and 5 Months' '22 Years and 6 Months'
 '22 Years and 7 Months' '26 Years and 7 Months' '26 Years and 8 Months'
 '26 Years and 9 Months' '26 Years and 10 Months' '26 Years and 11 Months'
 '27 Years and 0 Months' '27 Years and 1 Months' '27 Years and 2 Months'
 '17 Years and 9 Months' '17 Years and 10 Months' '17 Years and 11 Months'
 '18 Years and 1 Months' '18 Years and 2 Months' '18 Years and 3 Months'
 '18 Years and 4 Months' '17 Years and 3 Months' '17 Years and 4 Months'
 '17 Years and 5 Months' '17 Years and 6 Months' '17 Years and 7 Months'
 '17 Years and 8 Months' '30 Years and 8 Months' '30 Years and 9 Months'
```

Similarly, when understanding the number of delayed payments for each customer, we see that the data integrity wasn’t being ensured, as shown in Figure 10. There are ‘_’ characters for many values; therefore, we will consider cleaning them, converting them into numerical features, and normalizing them.

Figure 10

Num_of_Delayed_Payment Sample Values

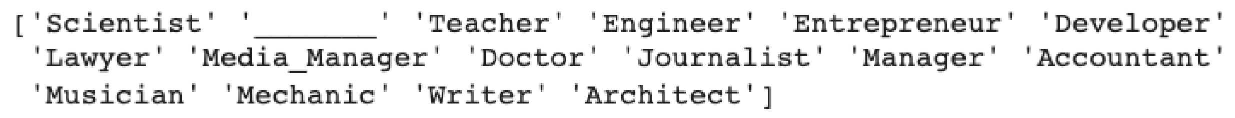


```
['7' nan '4' '8_' '6' '1' '-1' '3_' '0' '8' '5' '3' '9' '12' '15' '17'
 '10' '2' '2_' '11' '14' '20' '22' '13' '13_' '14_' '16' '12_' '18' '19'
 '23' '24' '21' '3318' '3083' '22_' '1338' '4_' '26' '11_' '3104' '21_'
 '25' '10_' '183_' '9_' '1106' '834' '19_' '24_' '17_' '23_' '2672' '20_'
 '2008' '-3' '538' '6_' '1_' '16_' '27' '-2' '3478' '2420' '15_' '707'
 '708' '26_' '18_' '3815' '28' '5_' '1867' '2250' '1463' '25_' '7_' '4126'
```

Figure 11 shows the unique values in the 'Occupation' feature. Notice that there are odd values that need to be cleaned, like the above variable; however, after cleaning, Occupation should still be kept as a categorical feature. In this case, an appropriate label encoding will be applied in the pre-processing step so that Occupation will be converted into machine-readable values for better performance results.

Figure 11

Occupation Feature Values



```
['Scientist' '_______' 'Teacher' 'Engineer' 'Entrepreneur' 'Developer'
 'Lawyer' 'Media_Manager' 'Doctor' 'Journalist' 'Manager' 'Accountant'
 'Musician' 'Mechanic' 'Writer' 'Architect']
```

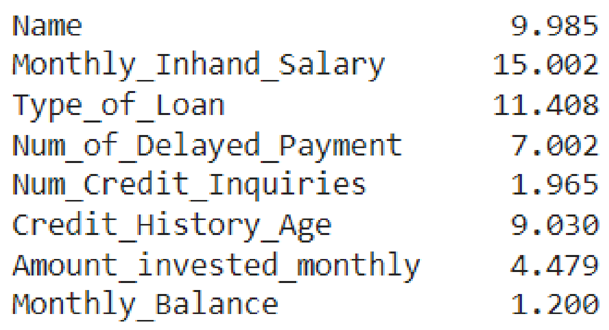
5.2 Data Cleaning

5.2.1 Handling Missing Values

The first step while cleaning data is to look for missing values. This dataset had quite a few missing values that had to be dealt with. Figure 12 shows the percentage of missing values in a particular feature.

Figure 12

Percentage Of Missing Values



```
Name                        9.985
Monthly_Inhand_Salary      15.002
Type_of_Loan               11.408
Num_of_Delayed_Payment      7.002
Num_Credit_Inquiries        1.965
Credit_History_Age          9.030
Amount_invested_monthly     4.479
Monthly_Balance             1.200
```

These missing values were handled in the following ways:

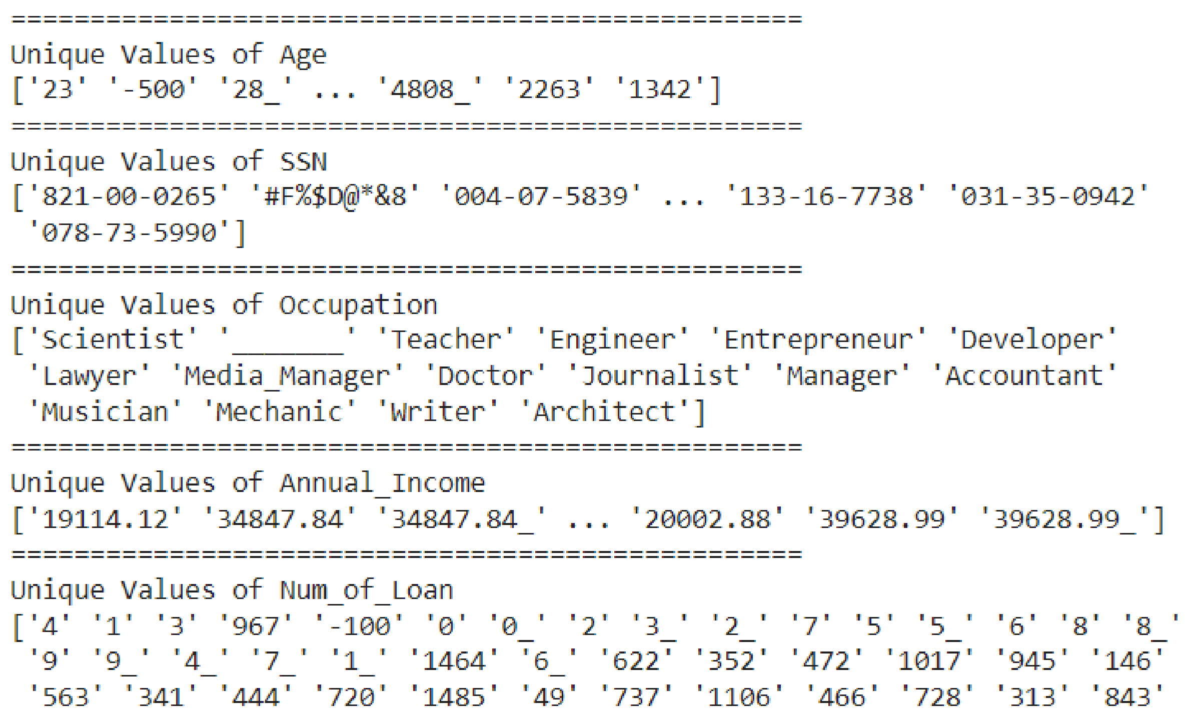
- Name - mode imputation based on Customer_ID
- Monthly_Inhand_Salary - mode imputation based on Customer_ID
- Num_of_Delayed_Payments - mode imputation based on Customer_ID
- Num_Credit_Inquiries - mode imputation based on Customer_ID
- Credit_History_Age- mode imputation based on Customer_ID
- Amount_invested_monthly- mode imputation based on Customer_ID
- Monthly_Balance- mode imputation based on Customer_ID

5.2.2 Handling Messy Data

The dataset also had data that was not needed in that particular field. These kinds of data create a noise in the model development phase and cannot be handled by the models very well. It is important to find such data and clean it so that it can be used by the machine learning model. We found these noisy data by going through the unique values of those fields. We found that many numerical features had '_' in the end the necessary data. We cleaned them by removing those discrepancies. SSN feature had values such as '#F%$D@*&8'. Such values were replaced with the mode of that particular customer ID. A sample of these messy data can be seen in Figure 13.

Figure 13

Sample Of Messy Data



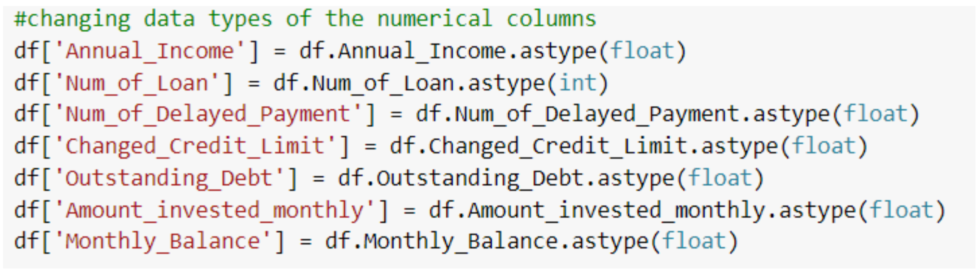
```
==================================================
Unique Values of Age
['23' '-500' '28_' ... '4808_' '2263' '1342']
==================================================
Unique Values of SSN
['821-00-0265' '#F%$D@*&8' '004-07-5839' ... '133-16-7738' '031-35-0942'
 '078-73-5990']
==================================================
Unique Values of Occupation
['Scientist' '_______' 'Teacher' 'Engineer' 'Entrepreneur' 'Developer'
 'Lawyer' 'Media_Manager' 'Doctor' 'Journalist' 'Manager' 'Accountant'
 'Musician' 'Mechanic' 'Writer' 'Architect']
==================================================
Unique Values of Annual_Income
['19114.12' '34847.84' '34847.84_' ... '20002.88' '39628.99' '39628.99_']
==================================================
Unique Values of Num_of_Loan
['4' '1' '3' '967' '-100' '0' '0_' '2' '3_' '2_' '7' '5' '5_' '6' '8' '8_'
 '9' '9_' '4_' '7_' '1_' '1464' '6_' '622' '352' '472' '1017' '945' '146'
 '563' '341' '444' '720' '1485' '49' '737' '1106' '466' '728' '313' '843'
```

5.2.3 Changing Data Types

Since there was noisy data in the numerical features, the datatypes of those features were not in the correct format. Once the noisy data was treated, the data types were changed to the correct type so that the machine understands the data correctly. Figure 14 shows the code that was used to correct these data types.

Figure 14

Code Implementation To Change Data Types



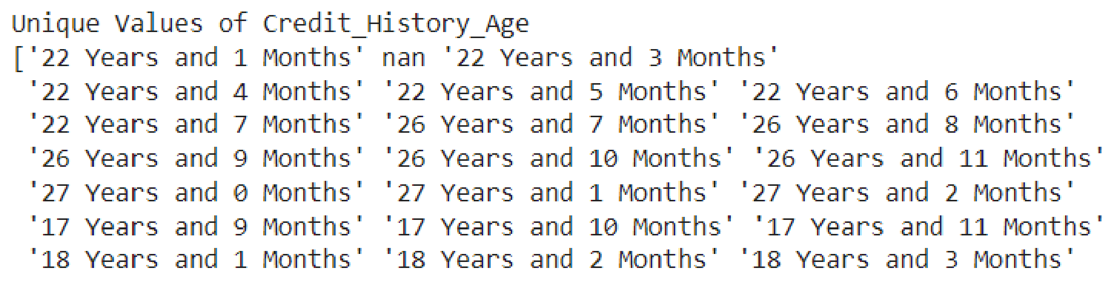
```python
#changing data types of the numerical columns
df['Annual_Income'] = df.Annual_Income.astype(float)
df['Num_of_Loan'] = df.Num_of_Loan.astype(int)
df['Num_of_Delayed_Payment'] = df.Num_of_Delayed_Payment.astype(float)
df['Changed_Credit_Limit'] = df.Changed_Credit_Limit.astype(float)
df['Outstanding_Debt'] = df.Outstanding_Debt.astype(float)
df['Amount_invested_monthly'] = df.Amount_invested_monthly.astype(float)
df['Monthly_Balance'] = df.Monthly_Balance.astype(float)
```

5.2.4 Feature Engineering

The Credit_History_Age feature had the values in string which made it difficult to gain insights from it. The sample values can be seen in Figure 15. A function was written to convert these into the number of months. The split function was used to divide the values of the string. The number of years and the number of months were assigned to temporary variables year and month. Then, the year was multiplied by 12 and added to the month. This gave us the total number of months and this was returned back to the field. Figure 16 shows how the field looks after this transformation.

Figure 15

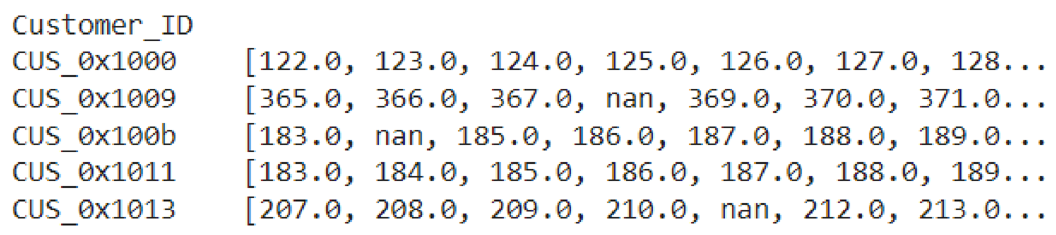
Credit_History_Age Before Transformation



```
Unique Values of Credit_History_Age
['22 Years and 1 Months' nan '22 Years and 3 Months'
 '22 Years and 4 Months' '22 Years and 5 Months' '22 Years and 6 Months'
 '22 Years and 7 Months' '26 Years and 7 Months' '26 Years and 8 Months'
 '26 Years and 9 Months' '26 Years and 10 Months' '26 Years and 11 Months'
 '27 Years and 0 Months' '27 Years and 1 Months' '27 Years and 2 Months'
 '17 Years and 9 Months' '17 Years and 10 Months' '17 Years and 11 Months'
 '18 Years and 1 Months' '18 Years and 2 Months' '18 Years and 3 Months'
```

Figure 16

Credit_History_Age After Transformation



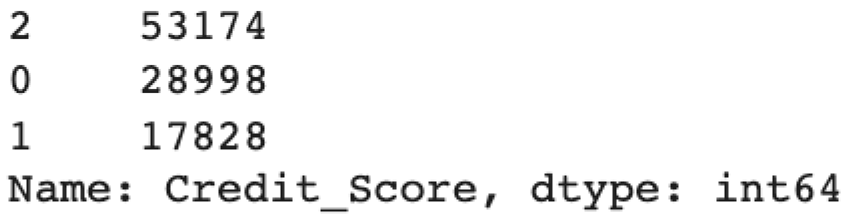
```
Customer_ID
CUS_0x1000    [122.0, 123.0, 124.0, 125.0, 126.0, 127.0, 128...
CUS_0x1009    [365.0, 366.0, 367.0, nan, 369.0, 370.0, 371.0...
CUS_0x100b    [183.0, nan, 185.0, 186.0, 187.0, 188.0, 189.0...
CUS_0x1011    [183.0, 184.0, 185.0, 186.0, 187.0, 188.0, 189...
CUS_0x1013    [207.0, 208.0, 209.0, 210.0, nan, 212.0, 213.0...
```

5.3 Data Preprocessing

After exploring the data, there is a need to preprocess some numerical and categorical features so that the model can perform better. We first need to map our target feature, ‘Credit_Score’, into numerical variables. ‘Poor’ credit card is mapped as 0, ‘Good’ as 1, and ‘Standard’ as 2. Figure 17 shows the target feature after encoding.

Figure 17

Credit_Score Target After Encoding

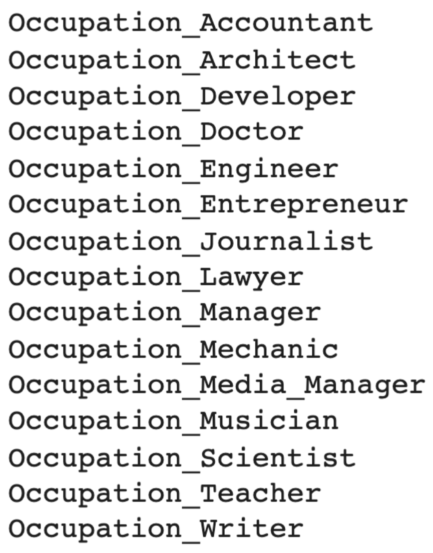


```
2    53174
0    28998
1    17828
Name: Credit_Score, dtype: int64
```

Next, ‘Occupation’ is a nominal categorical variable, meaning there is no actual order for the values to follow under. Therefore, dummy encoding has been applied for this pre-processing step. After being encoded, each unique value of the Occupation feature will be transformed into a new column with binary values. For example, if a customer is a Doctor, the column Occupation_Doctor will have a value of 1, and the rest will have values of 0. Figure 18 shows a new column after Occupation encoding with 15 more columns being added to the data frame.

Figure 18

Post Dummy Encoding for Occupation - New Columns

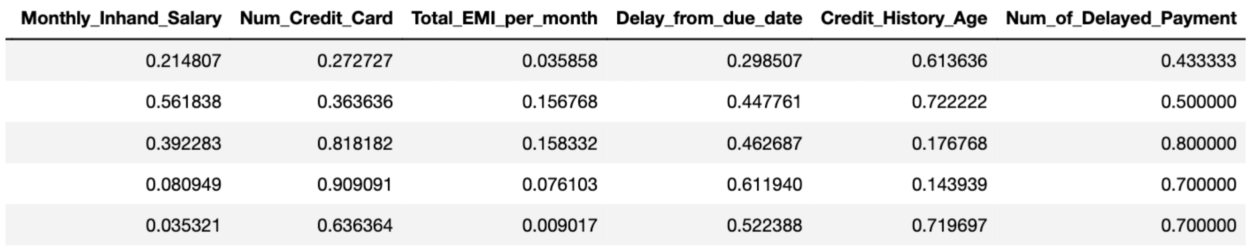


```
Occupation_Accountant
Occupation_Architect
Occupation_Developer
Occupation_Doctor
Occupation_Engineer
Occupation_Entrepreneur
Occupation_Journalist
Occupation_Lawyer
Occupation_Manager
Occupation_Mechanic
Occupation_Media_Manager
Occupation_Musician
Occupation_Scientist
Occupation_Teacher
Occupation_Writer
```

Finally, for the numerical variables, MinMax Scaling will be applied to normalize different ranges among the features. Basically, MinMax Scaling will scale each of the declared variables into the range [0,1], with 0 being the minimum value of that feature and 1 being the maximum. Figure 19 shows part of the dataframe after applying normalization.

Figure 19

Post Normalization For Numerical Variables



Monthly_Inhand_Salary	Num_Credit_Card	Total_EMI_per_month	Delay_from_due_date	Credit_History_Age	Num_of_Delayed_Payment
0.214807	0.272727	0.035858	0.298507	0.613636	0.433333
0.561838	0.363636	0.156768	0.447761	0.722222	0.500000
0.392283	0.818182	0.158332	0.462687	0.176768	0.800000
0.080949	0.909091	0.076103	0.611940	0.143939	0.700000
0.035321	0.636364	0.009017	0.522388	0.719697	0.700000

6. Model Selection

There are a high number of machine learning algorithms available to solve all sorts of problems. For this project we choose Logistic Regression (LR), K Nearest Neighbors (KNN), Random Forest (RF), XGBoost and Ensemble Voting Classifier. These algorithms were chosen based on the literature survey that was conducted.

Logistic Regression

Logistic Regression is classification based supervised machine learning algorithm. Since it is an easy to implement and simple to use algorithm, it was used as the baseline model for our project. A baseline model is used to set a benchmark for the project to get an idea how well the preprocessing and transformation steps were done for the model development. It generally works well on binary classification problems but can be extended into multiclass classification as well. It uses the concept of sigmoid function to classify instances. A certain threshold is set, based on which the instances are classified into their classes.

K Nearest Neighbors

Another simple yet efficient classification algorithm is KNN. The way a KNN works is that a training subset of pattern vectors from all the classes are provided together with a collection of sample prototypes for that class. The class label is chosen using a majority rule after finding the k closest neighbors of an unknown vector among all the prototype vectors. The value of k should be odd in order to prevent ties on class overlap regions (Laaksonen & Oja, 1996).

Random Forest

Random Forest is an ensemble based machine learning algorithm that solves both classification and regression problems. The way a RF works is that multiple decision trees are created and are trained parallel to each other. Each tree gives out its own prediction and the final prediction is the majority output of all those trees. Since the prediction is based on majority, the performance is much better than an individual tree. This makes this model highly used in the world of machine learning.

XGBoost

XGBoost is another ensemble machine learning algorithm which is based on gradient boosted trees. In recent years it has become highly popular. The major advantages of this algorithm are that it is a parallelized algorithm and also is optimized well. The way an XGBoost works is that it runs multiple trees one after the other. These trees are weak and hence called base learners. Collectively these trees make up a strong learner which gives out predictions. XGBoost works very well on sparse and large datasets.

Ensemble Voting Classifier

Ensemble Voting Classifier is nothing but a combination of two or more models individually in an ensemble. With an increase in the volumes of data, the traditional algorithms cannot handle such volumes. Hence, voting classifiers come into the picture. It helps improve model performance by using the voting mechanism. The way it works is that different predictions will be obtained by each classifier. Using a voting mechanism where the majority will be from the winning class will yield the best prediction. The benefit of using ensembles of various classifiers is that no two of them will make the same error (Leon et al., 2017). There are a number of voting techniques such as soft vote and hard vote which can be used according to the use case.

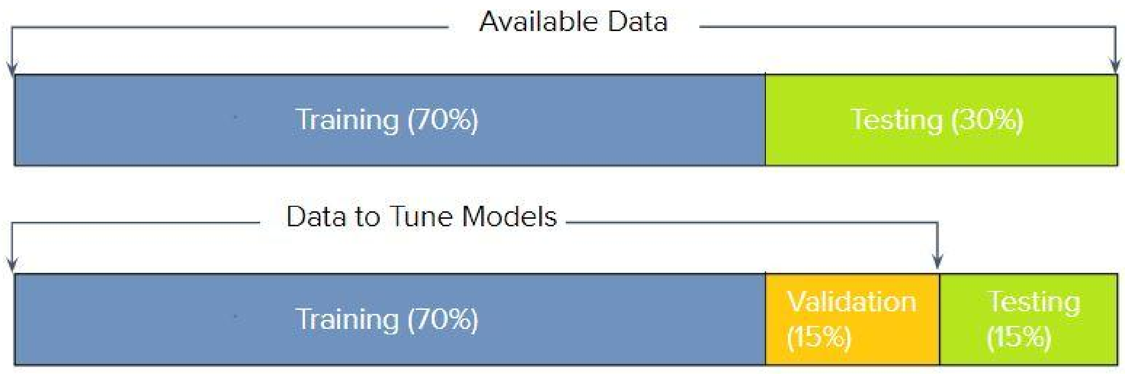
7. Model Development

7.1 Model Preparation

The model development process begins by reading the cleaned preprocessed dataset into a pandas dataframe. Since the dataset is large at 100,000 rows, we decided to perform a train/validate/test split using Sklearn's model selection library. This allowed the 100,000 rows to be split into 70,000 rows for training, 15,000 for validation, and 15,000 for testing. We considered doing K Fold cross-validation, but we felt that our dataset was large enough that we would be okay with using just the train test split. Typically Kfold is used for projects that do not have as much training data and need to use as much training data as possible. We believed that 70,000 rows were more than adequate for training our models. The figure below shows our data split.

Figure 20

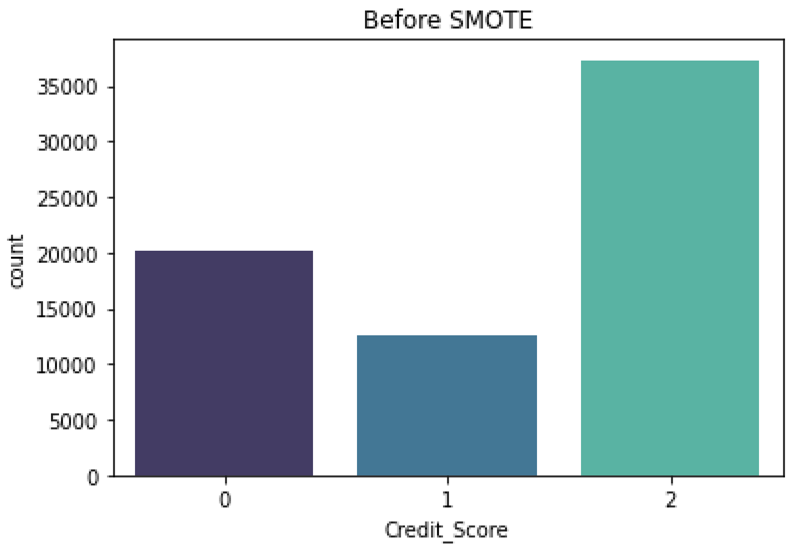
Train, Validate, and Test Diagram



After completing the split, the training data was oversampled, which can be seen in figure 22. Based on the literature review, we knew our project would need to balance the target classes. If not, we risk oversampling the majority class and inducing bias into our models (Zhu & Lin, 2017). For this reason, we applied the Synthetic Minority Oversampling Technique (SMOTE). Implementing SMOTE allowed us to balance the target classes evenly. The figure below shows our target classes before applying SMOTE.

Figure 21

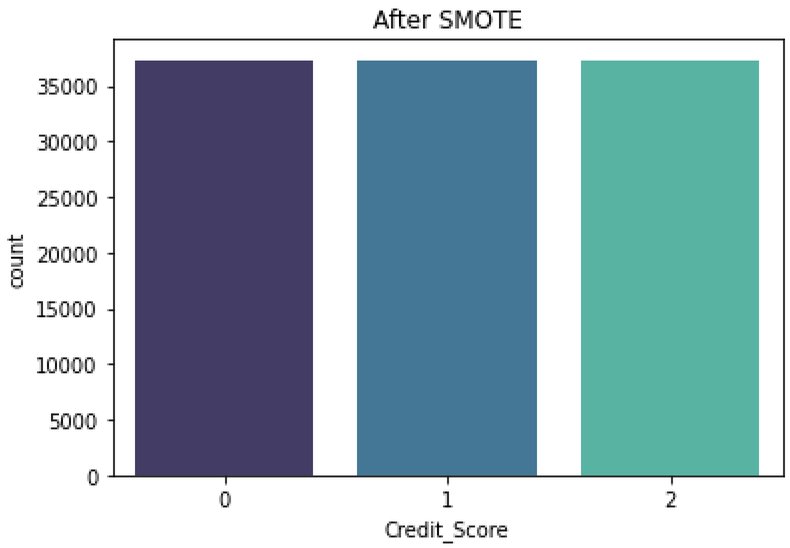
Target Classes Before SMOTE



The figure above clearly shows the imbalance of target classes. The distribution breaks down as 0-20,193 instances, 1-12,506 instances, and 2-37,301 instances. The application of SMOTE oversampled all of the non-majority classes to the same amount as the majority class. Meaning that all of the classes are synthetically oversampled to 37,301 instances each. The figure below shows the target class distribution after SMOTE is applied.

Figure 22

Target Class After SMOTE



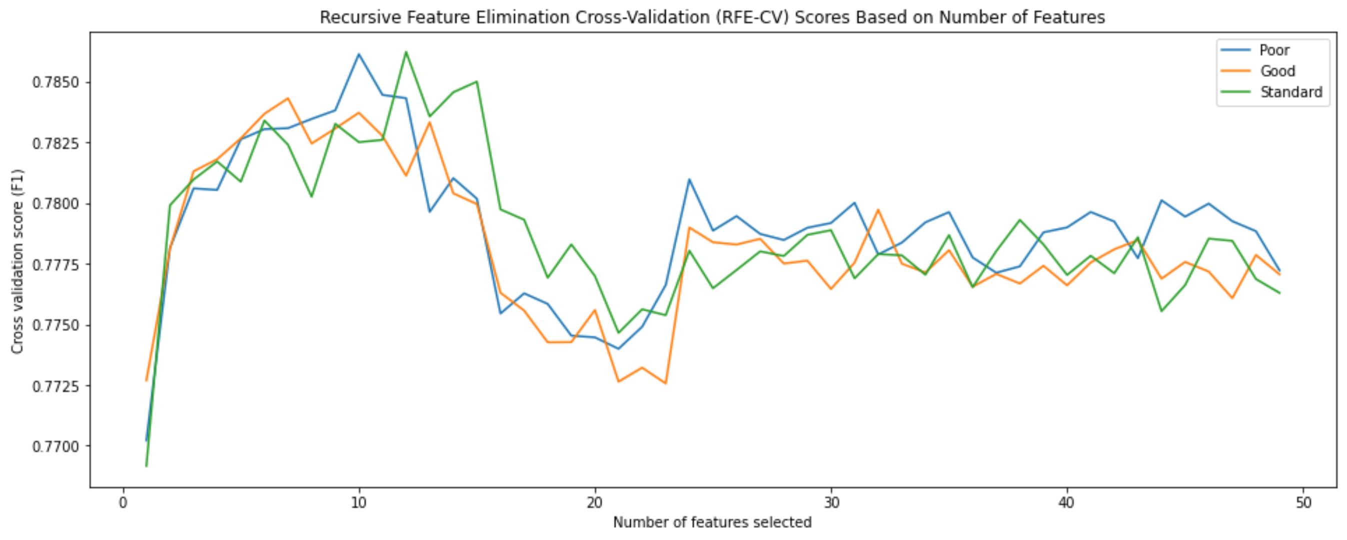
By applying SMOTE, all of the target classes are now synthetically the same amount. This means that the training dataset has grown from 70,000 rows to 111,903 rows. This should equate to better performance for the classifier on every target class.

7.2 Feature Selection

When we first approached this project, we should have considered implementing a form of feature selection. Upon completing the initial process of validating the models, we identified that some of our models were overfitting. The extent of overfitting will be discussed in detail in section 8 of this paper. To remedy this problem, we decided to implement Recursive Feature Elimination Cross-Validation (RFECV). RFECV functions by removing features with low importance and re-fitting the model until it finds the least amount of features with the best performance (Kuhn & Johnson, 2018, p. 494). For our project, RFECV started with 50 features and determined that 10 features were the optimal amount for feature selection purposes. The figure below shows the plot for RFECV and how 10 features was the best performance based on the metric F1 with the minimal amount of features.

Figure 23

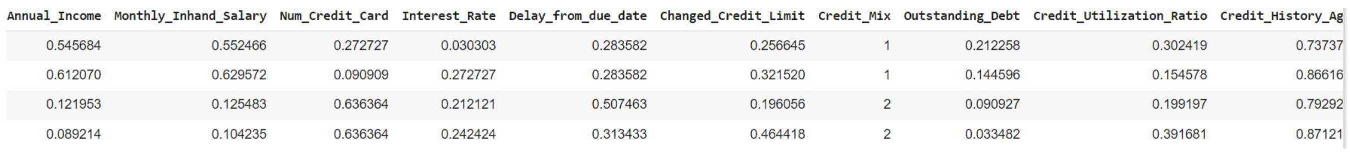
RFECV Plot



The result of the selected features with the highest importance can be seen in figure ().

Figure 24

View of Features Selected From RFECV



Annual_Income	Monthly_Inhand_Salary	Num_Credit_Card	Interest_Rate	Delay_from_due_date	Changed_Credit_Limit	Credit_Mix	Outstanding_Debt	Credit_Utilization_Ratio	Credit_History_Ag
0.545684	0.552466	0.272727	0.030303	0.283582	0.256645	1	0.212258	0.302419	0.73737
0.612070	0.629572	0.090909	0.272727	0.283582	0.321520	1	0.144596	0.154578	0.86616
0.121953	0.125483	0.636364	0.212121	0.507463	0.196056	2	0.090927	0.199197	0.79292
0.089214	0.104235	0.636364	0.242424	0.313433	0.464418	2	0.033482	0.391681	0.87121

8. Model Evaluation

8.1 Metrics (*F1-macro, Accuracy, AUC, ROC curve)*

The metrics used to evaluate the models were accuracy, AUC score, ROC curve, and F1-macro. The primary metric was F1-macro. This is due to the fact that our project was an imbalanced multi-class classification problem, and this meant that using a metric like accuracy as the primary means of evaluating performance would be flawed. Accuracy would not be able to correctly identify if a single target class was being under-represented and the model was suffering from bias. Instead, the metric F1-macro calculated the average F1 score of all individual target classes (Géron, 2019, p. 160). This means that if a single target class is underperforming, then the F1-macro score will be sensitive enough to represent that issue by its scoring adequately.

Accuracy was mainly used in the initial phase of model building. When evaluating the performance of models on training data versus the validation data, accuracy allowed for the ability to quickly identify if the model was displaying symptoms of possibly overfitting.

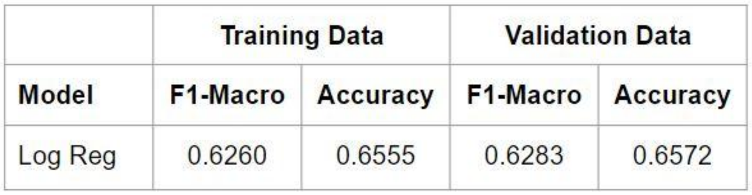
Finally, the ROC curve was implemented as a means to compare how the model was performing visually.

8.2 Logistic Regression

The first model implemented was Logistic Regression. Due to the simplicity of this model, it acted as our pseudo-baseline model. A model which we could compare against for a general idea of what a simplistic model would perform like. Table 1 below shows the initial performance of Logistic Regression on the training and validation data. The metric accuracy demonstrated that the model was not having issues with overfitting because there was not much difference between the two.

Table 1

Logistic Regression Initial Model

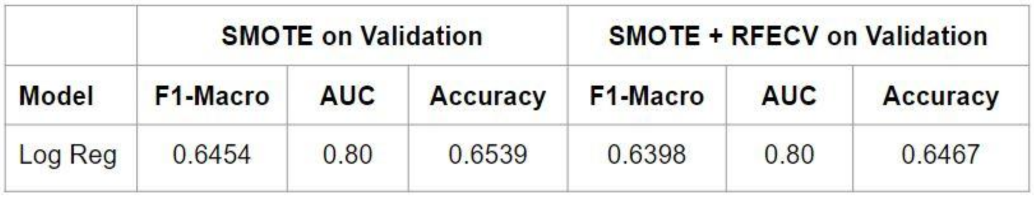


	Training Data		Validation Data	
Model	**F1-Macro**	**Accuracy**	**F1-Macro**	**Accuracy**
Log Reg	0.6260	0.6555	0.6283	0.6572

After the initial validation, the model had SMOTE, and SMOTE plus RFECV applied. The model benefited from applying SMOTE as the F1-macro score went up, which is our primary metric to focus upon. The addition of RFECV caused the model to perform worse, which makes sense because it was not having issues with overfitting. The best-performing Logistic Regression model was just the implementation of SMOTE. Table 2 below shows the performance of Logistic Regression.

Table 2

Logistic Regression with SMOTE and SMOTE + RFECV

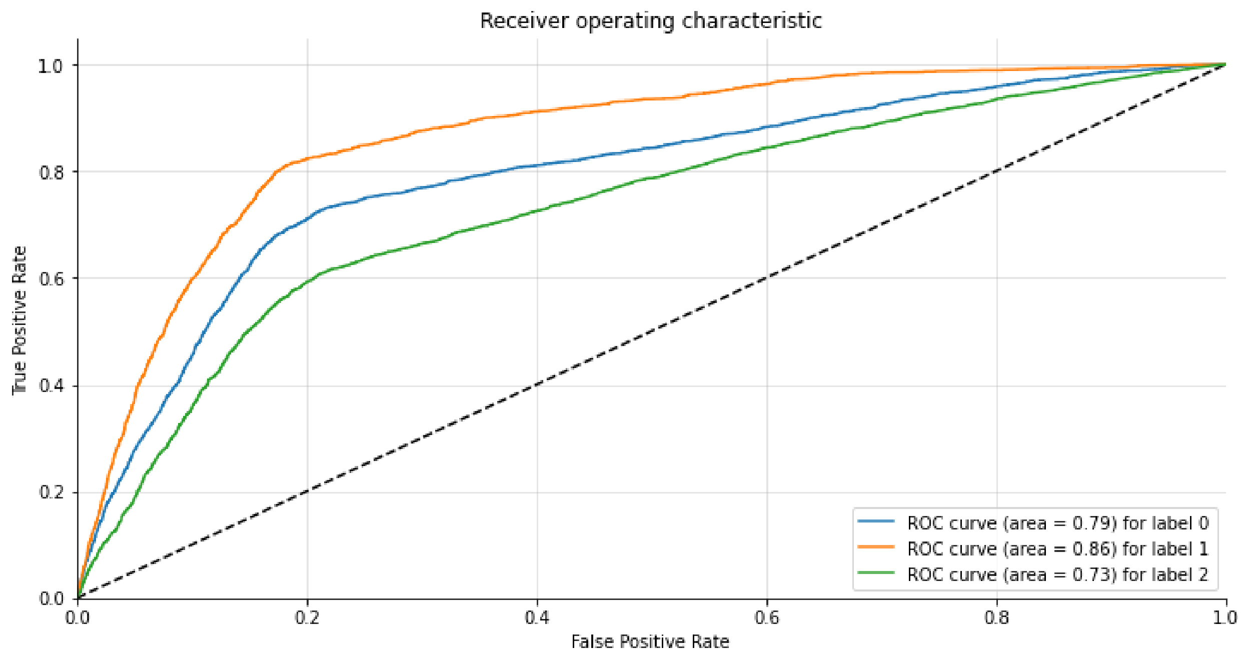


	SMOTE on Validation			SMOTE + RFECV on Validation		
Model	**F1-Macro**	**AUC**	**Accuracy**	**F1-Macro**	**AUC**	**Accuracy**
Log Reg	0.6454	0.80	0.6539	0.6398	0.80	0.6467

The plot below depicts the ROC curve for best performing Logistic Regression, which was with SMOTE.

Figure 25

Logistic Regression with SMOTE, ROC curve



8.3 KNN

During the initial validation phase for KNN, we identified that the model was having issues with overfitting. The table below shows that the accuracy of the training data was significantly higher than the accuracy of the validation data. This meant that the implementation of RFECV would potentially help curtail this issue.

Table 3

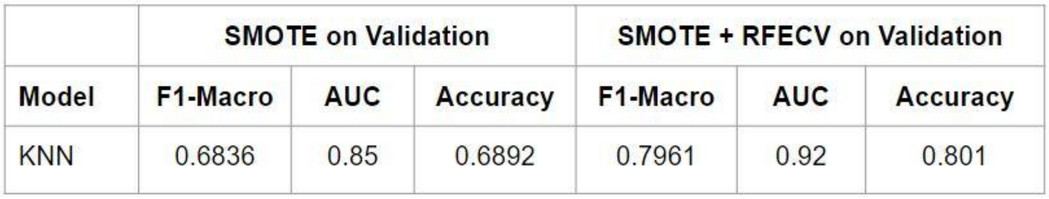
KNN Initial Model

	Training Data		**Validation Data**	
Model	**F1-Macro**	**Accuracy**	**F1-Macro**	**Accuracy**
KNN	0.7690	0.7864	0.6599	0.6859

KNN is known to suffer from dimensionality issues, which is the probable cause of its overfitting (Kelleher et al., 2015, p. 281). Table 4 below shows how the implementation of SMOTE helped slightly improve the performance of KNN, and the addition of RFECV feature selection dramatically improved its performance. RFECV mitigated the overfitting issue by training the model with just 10 features and allowed for the model to perform optimally.

Table 4

KNN with SMOTE and SMOTE + RFECV

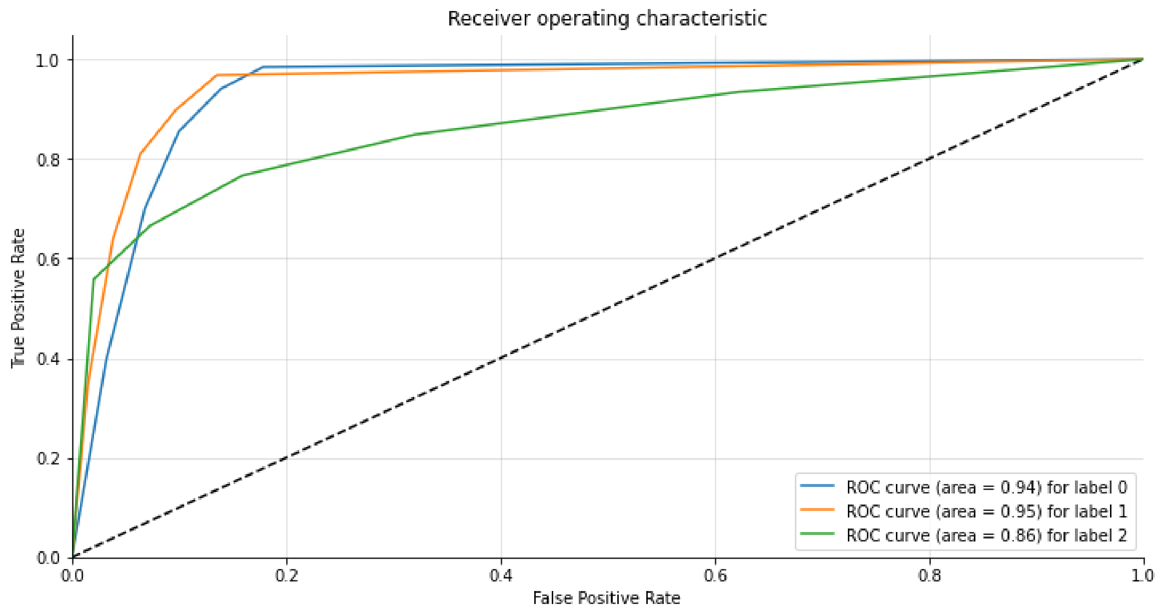


	SMOTE on Validation			SMOTE + RFECV on Validation		
Model	**F1-Macro**	**AUC**	**Accuracy**	**F1-Macro**	**AUC**	**Accuracy**
KNN	0.6836	0.85	0.6892	0.7961	0.92	0.801

The plot below depicts the ROC curve for the best performing KNN, which was with SMOTE + RFECV.

Figure 26

KNN with SMOTE + RFECV, ROC curve

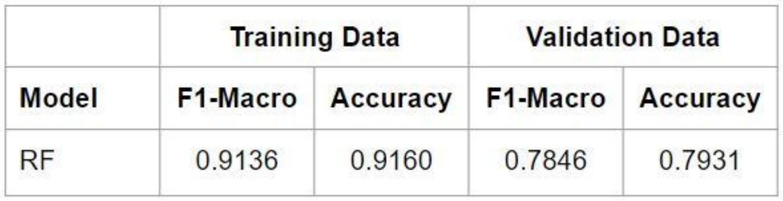


8.4 Random Forest

Random Forest is a machine learning technique that is known to not typically overfit due to its “extra randomness when growing trees” (Géron, 2019, p. 261). During our initial validation phase, we identified that our model was overfitting. Table 5 below shows the performance differences between training and validation data. Similar to standard Decision Trees, it is possible that the Random Forest model was growing without restrictions which means the trees grew too deep and overfit the training data (Géron, 2019, p. 246).

Table 5

Random Forest Initial Model



	Training Data		Validation Data	
Model	**F1-Macro**	**Accuracy**	**F1-Macro**	**Accuracy**
RF	0.9136	0.9160	0.7846	0.7931

Applying SMOTE and SMOTE + RFECV improved the model performance slightly. Out-of-the-box Random Forest is a very capable technique, and it is not surprising that the addition of SMOTE and RFECV offered minimal improvements. Perhaps, it is due to how Random Forest works. Random Forest operates by randomly sampling data, and features in an aggregated form, which is like a pseudo-feature selection (Géron, 2019, p. 262). Plus, the model does this sampling with replacement, which means that it can potentially train on more instances of an under-represented target class, which is similar to the benefits of SMOTE (Vanderplas & VanderPlas, 2016, p.444). Table 6 below shows the performance of Random Forest with SMOTE and SMOTE + RFECV.

Table 6

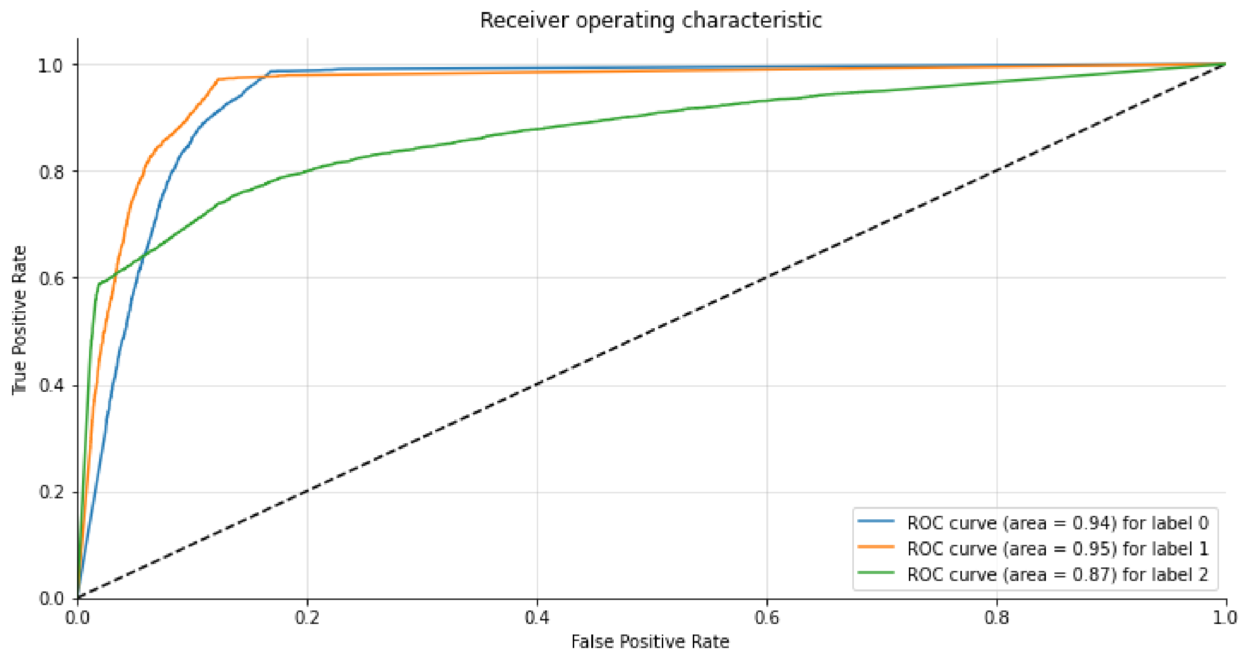
Random Forest with SMOTE and SMOTE + RFECV

	SMOTE on Validation			SMOTE + RFECV on Validation		
Model	**F1-Macro**	**AUC**	**Accuracy**	**F1-Macro**	**AUC**	**Accuracy**
RF	0.7856	0.92	0.7922	0.8010	0.92	0.8045

The best-performing model ended up being Random Forest with SMOTE + RFECV, but the performance gain over the initial model was minimal. The plot below depicts the ROC curve for Random Forest with SMOTE + RFECV.

Figure 27

Random Forest with SMOTE + RFECV, ROC curve



8.5 XGBoost

The initial validation phase for XGBoost demonstrated that the model was not exhibiting overfitting issues. Table 7 below shows the model performance on training and validation data.

Table 7

XGBoost Initial Model

	Training Data		Validation Data	
Model	**F1-Macro**	**Accuracy**	**F1-Macro**	**Accuracy**
XGB	0.6998	0.7191	0.6746	0.6742

The implementation of SMOTE produced some measurable benefits for XGBoost, but RFECV feature selection reduced the model performance. As stated above, this is because the model was not having issues with overfitting. Table 8 below shows the performance of XGBoost with SMOTE and SMOTE + RFECV.

Table 8

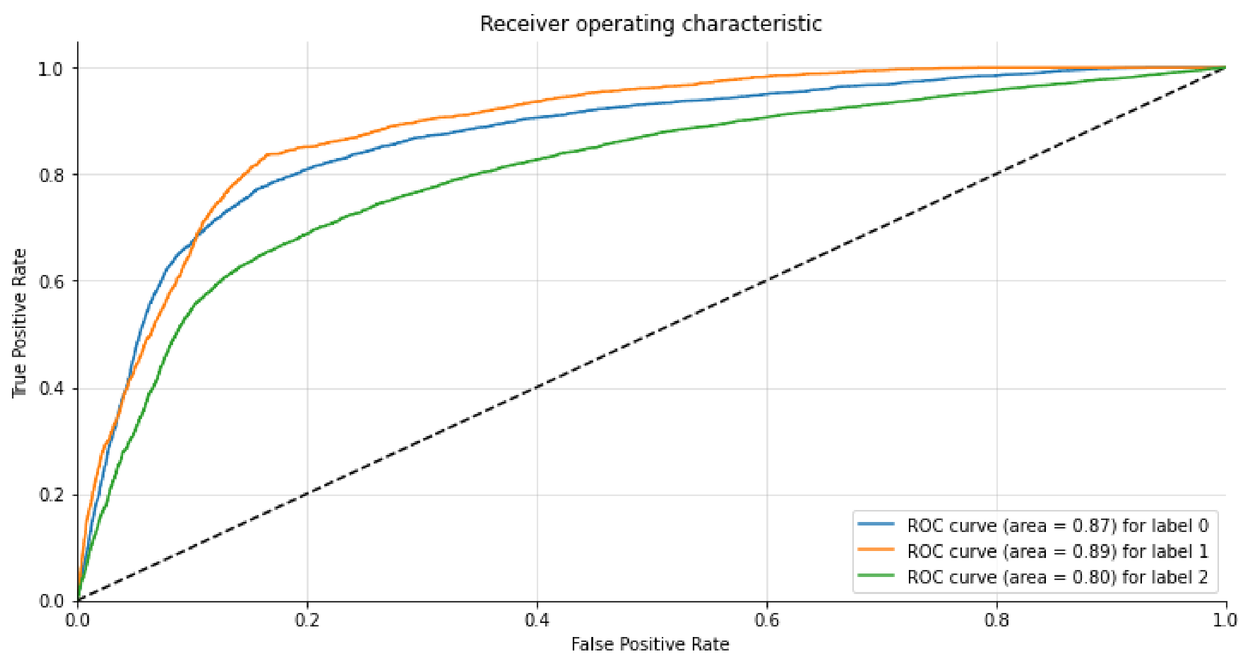
XGBoost with SMOTE and SMOTE + RFECV

	SMOTE on Validation			SMOTE + RFECV on Validation		
Model	**F1-Macro**	**AUC**	**Accuracy**	**F1-Macro**	**AUC**	**Accuracy**
XGB	0.6830	0.85	0.6888	0.6700	0.85	0.6764

The best-performing model for XGBoost was XGBoost with SMOTE. The ROC curve plot for this model can be seen in figure 28 below.

Figure 28

XGBoost with SMOTE, ROC curve



8.6 Voting Classifier

Finally, we wanted to implement a novel classifier that had yet to be done before for this particular multi-class classification problem. The ensemble Voting classifier we constructed consisted of our KNN, Random Forest, and XGboost models. This Voting classifier functioned by aggregating the predictions of all three models to make a classification (Géron, 2019, p. 254). For our model implementation, we decided to use soft voting, which yielded higher performance than hard voting. During the initial validation phase, the Voting classifier struggled with overfitting, which is expected because two of the three models that made up the Voting classifier were dealing with overfitting issues (KNN, Random Forest). This performance can be seen in the table below.

Table 9

Voting Classifier Initial Model

	Training Data		Validation Data	
Model	**F1-Macro**	**Accuracy**	**F1-Macro**	**Accuracy**
Voting	0.8720	0.8786	0.7807	0.7925

Implementing SMOTE and SMOTE + RFECV improved the overall performance of the Voting classifier. The SMOTE + RFECV model was the best-performing classifier out of all the models created in this report.

Table 10

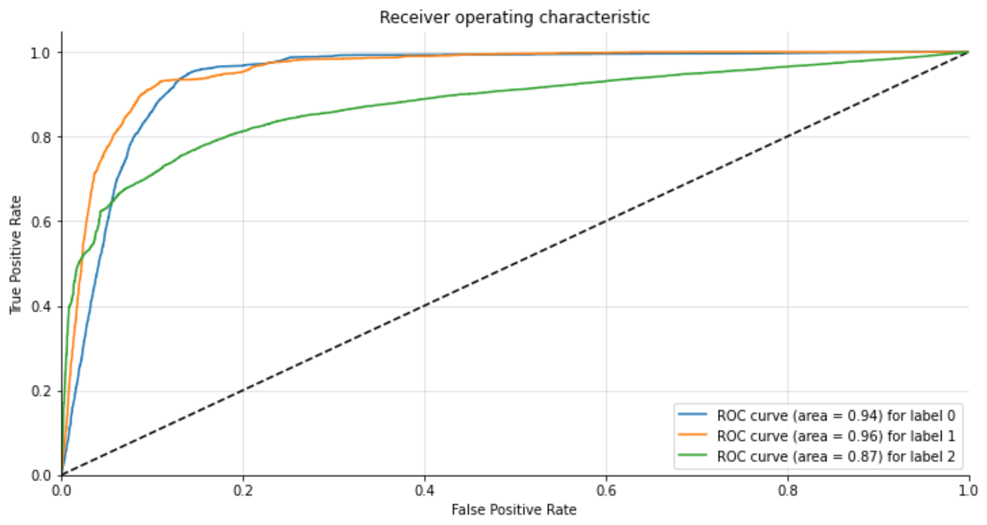
Voting Classifier with SMOTE and SMOTE + RFECV

	SMOTE on Validation			SMOTE + RFECV on Validation		
Model	**F1-Macro**	**AUC**	**Accuracy**	**F1-Macro**	**AUC**	**Accuracy**
Voting	0.7820	0.91	0.7886	0.8038	0.92	0.8081

The plot below visualizes the ROC curve for the Voting classifier

Figure 29

Voting with SMOTE + RFECV, ROC curve

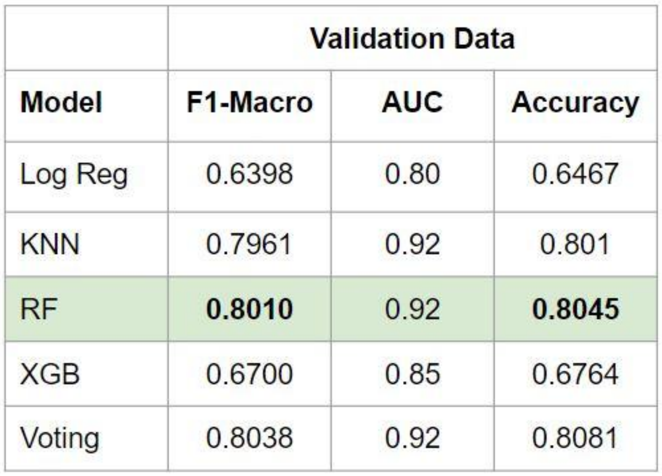


8.7 Comparing All Models

All of the best-performing models from each machine learning technique were compared amongst one another using the validation data. The ensemble Voting classifier was the best overall model. For deployment, we decided to use the next best-performing model, Random Forest. We were concerned that the Voting classifier would be too computationally expensive for any real-world deployment, and that a single classifier would be more efficient. Plus, the performance difference between the Voting classifier and Random Forest were minimal. Table 11 below shows the performance metrics for all of the models evaluated on validation data.

Table 11

Comparing All of the Models



	Validation Data		
Model	**F1-Macro**	**AUC**	**Accuracy**
Log Reg	0.6398	0.80	0.6467
KNN	0.7961	0.92	0.801
RF	**0.8010**	0.92	**0.8045**
XGB	0.6700	0.85	0.6764
Voting	0.8038	0.92	0.8081

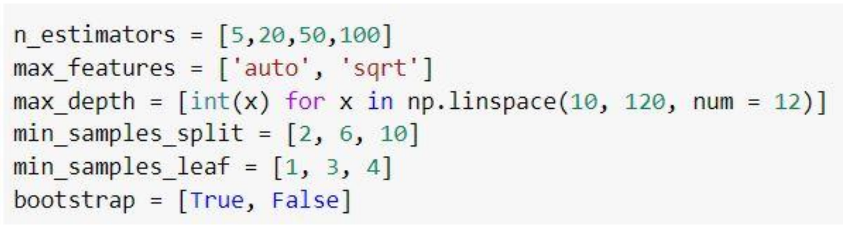
8.8 Best Model Tuned

Random Forest SMOTE + RFECV was our best-performing single model on validation data that we wanted to hyper-parameter tune. We believed that with the tuning, we could gain some performance improvements. For this, we decided to implement a Randomized Search. This approach is computationally friendly compared to a traditional Grid Search (Géron, 2019, p. 420).

We set up the search parameters as shown in figure 30 below.

Figure 30

Randomized Search Parameters

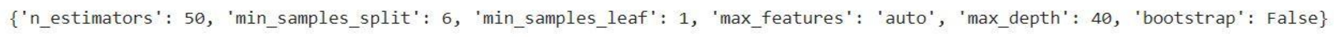


```
n_estimators = [5,20,50,100]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10, 120, num = 12)]
min_samples_split = [2, 6, 10]
min_samples_leaf = [1, 3, 4]
bootstrap = [True, False]
```

The result from the Randomized Search provided the following hyper-parameters as being the best parameters.

Figure 31

Randomized Search Best Parameters



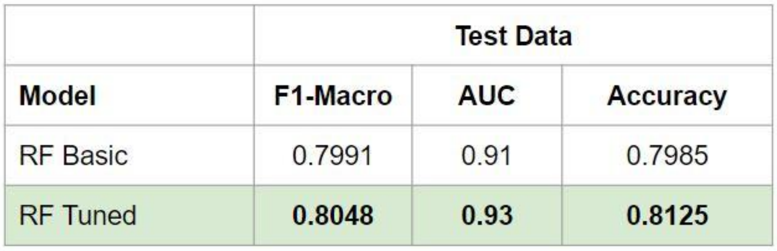
```
{'n_estimators': 50, 'min_samples_split': 6, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth': 40, 'bootstrap': False}
```

The Random Forest model with SMOTE + RFECV was re-trained with new parameters and tested on the validation data, demonstrating performance gains over the non-tuned model.

Finally, the non-tuned and tuned Random Forest models were evaluated on Test data. The table below shows their performance.

Table 12

Un-tuned Random Forest Compared to Tuned Random Forest on Test Data



	Test Data		
Model	**F1-Macro**	**AUC**	**Accuracy**
RF Basic	0.7991	0.91	0.7985
RF Tuned	**0.8048**	**0.93**	**0.8125**

The tuned Random Forest performed superior with a performance increase on all metrics. We felt that this tuned model was ready for deployment.

9. Deployment

All the work done and the code implementation has been uploaded to GitHub. GitHub is a version control software that also allows for collaboration between users. Code implementation can be found at the link below for reference, which provides a detailed understanding of our project. The repository consists of separate python files, one for data preprocessing and the other for model development.

Github URL: https://github.com/slushi7/Credit_Risk_Analysis

10. Conclusion

In conclusion, this project provides the best prediction of credit card status based on collected data features. After conducting essential cleaning and preprocessing steps, the data is utilized for modeling. SMOTE is implemented to mitigate target class imbalance, and RFECV feature selection is implemented to assist with overfitting issues. Random Forest and ensemble Voting Classifier yield the best predictions, with F1-macro scores of 80.10% and 80.38% on validation data, respectively. Considering the trade-off between computational expense and performance for potential future deployment, the project group decided to select the most appropriate modeling algorithm, Random Forest, for fine-tuning. This led to Random Forest being tuned with a randomized search, resulting in the best overall model that was capable of successfully classifying with an F1-macro score of 80.48% on test data.

References

F. Leon, S. A. Floria and C. Bădică, "Evaluating the effect of voting methods on ensemble-based classification," 2017 *IEEE International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*, 2017, pp. 1-6, doi: 10.1109/INISTA.2017.8001122.

Gahlaut, A., Tushar, & Singh, P. K. (2017). Prediction analysis of risky credit using data mining classification models. *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. https://doi.org/10.1109/icccnt.2017.8203982

Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems (2nd ed.). O'Reilly Media.

Kelleher, J. D., Namee, B. M., & D'Arcy, A. (2015). Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies. Amsterdam University Press.

Kuhn, M., & Johnson, K. (2018). Applied Predictive Modeling. Springer Publishing.

J. Laaksonen and E. Oja, "Classification with learning k-nearest neighbors," Proceedings of International Conference on Neural Networks (ICNN'96), 1996, pp. 1480-1483 vol.3, doi: 10.1109/ICNN.1996.549118.

Laborda, J., & Ryoo, S. (2021). Feature selection in a credit scoring model. *Mathematics*, *9*(7), 746. https://doi.org/10.3390/math9070746

Machado, M. R., & Karray, S. (2022). Assessing credit risk of commercial customers using hybrid machine learning algorithms. Expert Systems with Applications, 200, 116889. https://doi.org/10.1016/j.eswa.2022.116889

Moscato, V., Picariello, A., & Sperlí, G. (2021). A benchmark of machine learning approaches for Credit Score Prediction. *Expert Systems with Applications*, *165*, 113986. https://doi.org/10.1016/j.eswa.2020.113986

Paris, R. (2022). Credit score classification [Data set]. *Kaggle*. https://www.kaggle.com/datasets/parisrohan/credit-score-classification

Singh, P. (2017). Comparative study of individual and ensemble methods of classification for credit scoring. *2017 International Conference on Inventive Computing and Informatics (ICICI)*. https://doi.org/10.1109/icici.2017.8365282

Trivedi, S. K. (2020). A study on credit scoring modeling with different feature selection and machine learning approaches. *Technology in Society*, *63*, 101413. https://doi.org/10.1016/j.techsoc.2020.101413

Vanderplas, J., & VanderPlas, J. (2016). Python Data Science Handbook: Essential Tools for Working with Data. Van Duuren Media.

What is XGBoost? (n.d.). NVIDIA Data Science Glossary. https://www.nvidia.com/en-us/glossary/data-science/xgboost/

Zhu, & Lin. (2017, December). Synthetic minority oversampling technique for multiclass imbalance problems. ScienceDirect. https://www.sciencedirect.com/science/article/pii/S0031320317302947?via=ihub