

LendingClub Investment Analysis:

Borrower’s Default Prediction

Ki Min Lee, Thinh Mai, Yovinda Pricila, Sylvester Setio

# 

# **Index**

Executive Summary …………..……………………………………………………………………2

Section 1 - Introduction……………………………………………………………………….......2

1.1 Problem Statement …………………………………………………………………………...3

1.2 Data source …………………………………………………………………………………...3

1.3 Data description ……………………………………………………………………………....3

Section 2 - Literature Review ……………………………………………………………………..4

Section 3 - Data Preprocessing …………………………………………………………………...4

* Join data …………………………………………………………………………………..4
* Filtering, variable selections, and null handling ………………………………………….5
* Correlation analysis ……………………………………………………………………....5
* Outlier analysis …………………………………………………………………………...6
* Converting categorical data ……………………………………………………………....6
* Principal Component Analysis …………………………………………………………....6

Section 4 - Statistical modelling …………………………………………………………………..7

* Naive Bayes ……………………………………………………………………………….8
* Decision Tree …………………………………………………………………………...8
* Random Forest …………………………………………………………………………..8
* Neural Network ………………………………………………………………………8
* Resampling ……………………………………………………………………………….8
* Interpretation and Model results ………………………………………………………...9

Section 5 - Model evaluations …………………………………………………………………...12

Section 6 - Conclusion……………………………………………………………………………12

References ……………………………………………………………………………………….13

Figures and Tables ……………………………………………………………………………….14

Appendix ………………………………………………………………………………………...16

# 

# 

# **Executive Summary**

Social lending provides opportunities for micro-loan investors to earn an interest rate higher than savings accounts or traditional finance intermediaries. However, default on social lending loans will result in investors losing their entire principal. Considering these risks on the social lending landscape, our paper will use machine learning algorithms to predict borrower’s default, in hopes to minimize peer-to-peer lending risks. The data used was retrieved from the publicly available LendingClub dataset uploaded through Kaggle. The data provides borrower’s loan listings from April 2008 to September 2018.

Due to the imbalance classes within the LendingClub loan data, we perform multiple resampling methods - Random Under Sampling (RUS) and Synthetic Minority Oversampling Technique (SMOTE) - to rectify this explored problem. We perform four classifiers - Naive Bayes, Decision Tree, Random Forest, and Neural Network - and compare the predictive power by using the entire dataset and resampled dataset. The RUS technique significantly improved results in comparison with the classifiers. We found that resampling methods are able to boost models’ sensitivity, F1 score, and Area Under Curve (AUC) rate by 55%, 11%, and 25%. respectively. The proposed RUS yields its best result when used with the Neural Network, producing 66% accuracy, 67% recall, and 66% AUC. These results would help investors engaged in social peer-to-peer lending to make calculated, informed decisions when selecting potential borrowers and eliminating higher risks that exist in this platform.

# **Section 1: Introduction**

Technological and financial innovation allows peer-to-peer (P2P) lending to connect lenders to borrowers without the interference of institutional finance intermediaries, such as banks or credit bureau. Social lending has gained popularity and strong momentum, with platforms circulating billion dollar loans. The popularity of these platforms is growing as recently indicated by LendingClub (LC) which has reached 59.6 billion USD in circulating loans by June 2020, and thus becoming the world’s largest social lending platform (*LendingClub.com, 2020*). This blossoming of social lending attracts both borrowers and investors due to the lower transaction costs than the traditional financial institutions, allowing borrowers paying lower interest and lenders receiving higher return than they would attain from banks.

Evaluating the creditworthiness of loan applicants is a challenge in an investor's micro-financing decision, especially in cases where loans are unsecured. Aside from the convenience that social lending provides, the system also introduced lenders to new risks. For instance, lenders may not be repaid in full, borrower’s privacy intrusion, and other issues in the transacting parties (*Verstein, 2011*). While users rely on LC to facilitate the lending-borrowing connection, no legal financial institution will interpose their credit risks and guarantees. Traditionally, LC accepts fewer than 10% of all applicants to increase their stringent lending criteria (*LendingClub.com, 2020*). Despite these regulations imposed by LC, investors must be able to identify the borrowers who will pay back their loan in full, within its due time, in order to achieve a good investment. In all, profitability of P2P investors is a pivotal component in continued interest of the social lending market.

**Problem Statement**

Therefore, the purpose of our study is to identify borrower listings that will likely to pay in full in due time within the context of P2P lending. The analysis would focus to identify the borrower’s characteristics and financial features that would help investors to assess their credit-worthiness. By examining the loan status, we can classify a specific loan as ‘charged-off’ or ‘fully paid’. Charged-off loans are loans that have been declared by a creditor that an amount of the debt is unlikely to be collected. Fully paid loans are loans that have been fully repaid both in principal and its interests. Should the model work, the model will examine the factors that encourage or discourage loan extension and these factors will be put into higher considerations.

This study proposes and compares different predictive Machine Learning (ML) methods including Naive Bayes (NB), Random Forest (RF), Neural Network (NN), and Decision Tree (DT). The remainder of the study is organized as follows. Section 2 provides literature review on P2P lending. Section 3 provides the data preprocessing techniques to improve data quality. Section 4 presents the classifiers, Section 5 presents the model evaluation. Section 6 presents summary, conclusion, and future research recommendations.

**Data Source and Data Description**The Lending Club data contains the complete loan data from April 2008 to September 2018. The data has 2,260,668 observations and 145 attributes, such as the credit scores, number of finance inquiries, location (including zip codes and states), and collections among others. Although ID and memberID are the two primary keys, the data source has removed these two columns to protect their customers’ privacy. When building the data dictionary, we observe that our data consists of nominal, ordinal, numerical, and date data types. The data is posted in Kaggle.com, as exported by the LendingClub site.   
  
We have also done a literature review surrounding the business and the data set. A study conducted by Li et al. suggest subsequent studies to extract, process, and input macroeconomic features into their model. These will be the factors surrounding the loans [5]. Thus, we will be undertaking this suggestion; we agree that macroeconomic factors can often determine each loan’s success factor and, consequently, its rate of return. Therefore, we will be joining the Kaggle data set above with unemployment rates and gross domestic product data per state as provided by the Bureau of Labor Statistics.

In our data exploration, we have observed several interesting patterns by doing univariate data exploration. The loan amounts range from $0 to $40,000, as previously mentioned by the data description. We also see that the interest rate seems to be positively skewed, with most of the loans having interest rates between 5% to 20% and the skew can range till mid-30%. We also see that most of the loans have B and C grades, followed by A and lower grades. We note that most of our samples seem to be using the 36-months loan rather than its alternative, 60-months loan. Moreover, most of our samples are being done and/or almost being done with their loans, and a relatively small amount of our samples not being able to make full payments. We see that most of our samples are borrowing for the purpose of debt consolidation and repayment of credit cards. We are also seeing that our samples’ age of credits is positively skewed and averaging at around 15 years.

The important point seems to be the distribution of our samples’ loan status. Our initial dataset is showing an imbalance ratio between default and full-paid history. We will be careful at handling this data, possibly incorporating Costello et al.’s suggestion of using SMOTE, in order to synthetically equate the sample coming from a relatively smaller sample. [4]

**Section 2: Literature Review**

P2P lending is a relatively-new platform in the online financial marketplace. P2P lending does not fully conform to the theory; it only has difficulty with information asymmetry as it is not subject to as many regulations as traditional financial intermediaries. That said, class imbalance problems arise when there are a far greater or fewer number of objects in one class than another. Xia et al. state that effectively predicting credit risk from an imbalanced dataset is difficult because imbalanced data affects potentially discriminate between good borrowers and potential defaulters.

Resampling is one of the most prominent strategies for solving this issue. Study conducted by Namvar et al. compares the results of ML algorithms after performing three different resampling methods, including Under-Sampling (US), Over-Sampling (OS), and Hybrids [9]. Under-sampling reduces the number of instances in the majority class, whereas, over-sampling increases the number of instances in the minority class. Their research shows that Random Forest (RF) and RUS emerged as the best method for predicting a borrower’s status in P2P platforms.

In other hand, Costello et al. applied over-sampling technique, Synthetic Minority Over-sampling Technique (SMOTE), and adjusting cost-sensitivity, that is assigning different penalties to misclassifications. In this study, the authors use a k-Nearest Neighbor (kNN) algorithm to produce instances from the minority class. The results show that, using Area Under Curve (AUC) and accuracy as performance measures, implementation of SMOTE has shown considerably greater results compared to the baseline models. [4]

Few other studies have explored credit risk prediction in P2P lending with different ML models. Malekipirbazaari et al. used a variety of ML methods to classify the ‘good’ and ‘bad’ borrowers, and found that LC grades provide a better predictive power when plugged into their models than the borrower’s FICO score [2]. Another research conducted by Zhou and Wang further explored the loan default prediction by using weighted majority votes in tree aggregation to handle the imbalance dataset [6]. Aside from the SMOTE sampling method, parallelism random forest was implemented in an attempt to improve the algorithms’ efficiency. Zhou and Wang’s research shows that the result of this research demonstrates that the proposed random forest algorithm by weight allocation to trees and a weight majority-based prediction decision outperforms the original algorithm of random forest without weight consideration on the imbalanced dataset. [6]

Reading through the future research recommendation, Li and Han suggest that it would be helpful to involve new features such as the stock market or housing trend, so as to include more macroeconomic factors into the models to avoid the bias problems. With these relevant works, our research would focus on handling the imbalance dataset to achieve a higher sensitivity score, by also merging macroeconomic factors to our consideration. [5]

**Section 3: Data Pre-Processing**

**Joining Dataset**

For data pre-processing steps, concerning the main data frame and asides from formatting issues, we joined growth of unemployment rates from previous month to the issue date month and similar growth for the previous month from Bureau of Labor Statistics data set, joined growth of real GDP from previous quarter to the issue date quarter and similar growth for the previous quarter from Bureau of Economic Analysis data set, and joined abbreviated states names for easier subsequent joins from World Population Review.

For the data sets from Bureau of Labor Statistics (BLS) and Bureau of Economic Analysis (BEA), we joined these data based on the issue dates and the locations. By doing this, we will know what the macroeconomic factors influencing the loans during that specific period are in those specific states.

**Filtering, Variables Selection, and Treatment for Null Values**  
As for observation selection, we want to train our model to identify loans that will be fully paid and charged off. Thus, firstly, we select only loans that are fully paid and charged off. This cuts our original observation of 2,260,668 to 1,303,607.

Next, we examine missing values to remove incomplete features. Out of 145 features in our original dataset, we found that only 59 of those features are publicly available on the borrower's listing profile. The remaining variables are the private borrower’s information that Lending Club may obtain by performing hard pull on borrower’s credit. For this reason, we have decided to omit those features on our data, because our model’s feature needs to be obtainable by the potential investor. By doing so, we reduced the total number of features from 145 to 51.

However, we have found that most of them are overlapping information, such as borrowers’ zip code and states where they apply. Although different, the purpose is to identify their location and the surrounding location, thus we drop zip code down. Similarly, we dropped down columns such as subgrade, employment titles, and title purposes since we either have each of the information in the main data frame or because the information is too inconsistent. We also transformed the earliest credit line date to identify age of credit.

Null values may prevent our machine learning algorithms from running. Rather than replacing with mean or median, due to the small number of null values, we have decided to remove them all. The resultant shape of the data set from 1303607 is reduced to 1160023 from this process.

**Correlation Analysis**

Using our correlation analysis, we found the following results. As we can see in Figure 1, our correlation analysis yields overlapping variables. Other than higher loan amounts leading to higher installment, we believe that the rest indicates the same information. For example, more open accounts will increase the amount of total accounts. As for the numerical variables, we use Pearson’s correlation coefficient. It is quite different for categorical variables since we are using chi-squared analysis. We conduct chi-squared analysis for loan status, term, subgrade, home ownership status, verification status, purpose, address state, and initial listing status against every other factor. Though all of them resulted in extremely low P-values and indicated statistical significance, the Chi-squared value per se can be high. Thus, we selected one of the variables that are highly correlated among each other to filter out redundant variables and to avoid collinearity. In result, we obtained 36 columns that were not highly correlated to one another but are correlated to the loan\_status column (refer to Figure 2 and Figure 3).

We have also cut down our observations by choosing the issue date between 2015 and 2018 to keep the machine learning algorithm relevant and current. We find 2010 economic circumstances differ substantially from the 2015-2018 economic circumstances. Furthermore, the majority of loans have already had its maturity. In addition, this also helps to reduce the training time significantly due to the big size of our original data. By cutting down our observation, we are left with 797,003 observations.

**Outlier Analysis**

To process the outliers, we have decided to pick the numerical values from the chosen 36 variables and tested both z-score and Interquartile range to handle outliers. By removing observations that lie outside of Z-score ± 3, our observations were reduced to 668,135. However, we found that many features including loan\_status,, emp\_length and int\_rate were not normally distributed. In our InterQuartile Range (IQR) analysis, when we removed outliers that were outside of [Q1 - 1.5\*IQR] and [Q3 + 1.5\*IQR], we lost 70.2% of the data, which caused too much information loss. Since we are unable to use Z-outlier analysis as the important variables are not following normal distribution and IQR outlier analysis as we will be left with small observations,, we have decided not to remove outliers.

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

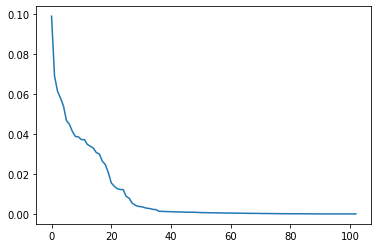
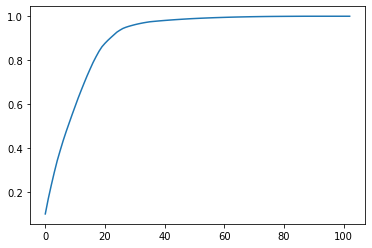
*Figure 2: Correlation Matrix of Post-filtering Figure 3: Correlation Analysis Variables with Rho Higher Than 0.5*

**Converting Categorical Data**

Conversion from ordinal to numeric was only performed on grade columns because the remaining categorical columns are non-ordered, such states, home ownership, and so on. For that reason, we created dummy variables of the nine columns, which transformed the dataset shape to 797,003 observations and 108 columns. Creating dummy variables helps to eliminate redundancy compared with other methods such as one-hot encoding and ordinal encoding. In addition to being less redundant, a dummy variable representation is required for some specific models.

**Principal Component Analysis (PCA)**

Lastly, we performed Principal Component Analysis (PCA) on our processed dataset. The objective of PCA is to choose the best fitting-line that will minimize the average squared distance from a point to the line, given certain points. With this, we scaled the numeric variables and selected all the 106 features as X and target variable loan status as Y. The result of the PCA is as follows and, to measure which PCA works, we build multiple decision trees. More specifically, we will be testing PCA 25 variables since we are trying to retain information while reducing dimensionality. We selected 25 components by looking at the *elbow* of the scree plot where the curve is flattened.

*Figure 4: Principal Component Analysis Scree Plot. Figure 5: Principal Component Analysis Cumulative Sum*

**Section 4: Statistical Modeling**

**Methodology**

In the following sections, we will share and discuss our experiments using Naive Bayes, Decision Tree, Random Forest and Neural Networks for classification problems. In our research, we find it inappropriate to use only accuracy score as our main performance measure due to the imbalance classification problem where the majority of records represents “Fully Paid” loans and overwhelm the number of “charged off” loans, indicating a 99% accuracy might be meaningless in this case. As a consequence, we will also ensure to include the precision and recall metrics. Finally, we split the data using a random split (0.7, 0.3) into training and test sets respectively.

Classification Report Elements:

Precision =

Recall =

F1-score = 2 x

Support = the number of true instances for each label

Weighted-avg metric = metric weighted by support

**Naive Bayes**

A Naive Bayes classifier is a probabilistic machine learning model that is used for classification tasks. The crux of the classifier is based on the Bayes theorem.

P(A|B) =

Naive Bayes can be extended to real-valued attributes, most commonly by assuming a Gaussian distribution. Out of all functions, we believe Gaussian (or Normal distribution) is the easiest to work with because the model can then be fit by simply finding the mean and standard deviation of the points within each label. We also choose to run Naive Bayes as our first model because it is very low computation cost and fast for both training and prediction. In addition, it works efficiently with a large dataset. The model gives an accuracy score of 0.55 which is fairly bad considering the fact that we can randomly guess if the loan will be charged off or paid fully with 50% chance. The weight precision is better at 0.76, while we still face a low weighted recall of 0.55.

**Decision Tree**

For our second model, we use Decision Tree with its default parameters. This includes the “gini” as the function to measure the quality of a split, “best” support strategies to choose the best split at each node and max\_depth as None, indicating nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples (default = 2). It is fast to train the decision tree model and also easy to interpret thanks to its straightforward visualizations. Even though we have a much better accuracy score of 0.70 and precision score of 0.70, our concern is that the recall rate for “charged-off” loan (1) is very low, at only 0.30. Furthermore, decision trees are well known prone to overfitting, especially when a tree is particularly deep. We consider this is a good model, but ultimately will not be a strong predictive model.

**Random Forest**

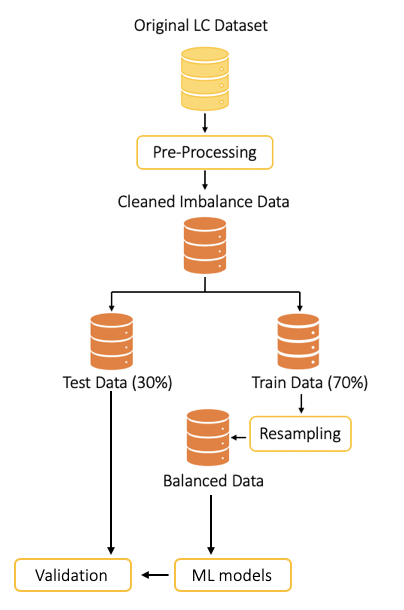
For our second model, we use the Random Forest Classifier. In general, Random Forest builds multiple trees and combines them together to get a more accurate result. We start off with the default feature of 100 trees in the forest along with *gini* as the function to measure the quality of a split. We notice there is a significant increase in the accuracy score changing from 0.55 (Naive Bayes) to 0.79. The weighted recall also increases to 0.79, while there is a slight decrease in weighted precision with a value of 0.75. Compared with the Naive Bayes model, this Random Forest achieves a better weighted recall and a much better accuracy score. This is one of many advantages of using Random Forest, in which every decision tree forecasts a response (charged off or fully paid) for an occurrence and the endmost response is decided through voting.

**Neural Network**

Next, we construct a Neural Networks model to see if we can achieve a better result. By definition, a neural network is trained by adjusting neuron input weights based on the network’s performance on example inputs. If the network classifies a loan status correctly on whether it is charged off or fully paid, then weights contributing to the correct answer are increased, while other weights are decreased. We set up a neural network model using MLP Classifier with 1 hidden layer and 20 nodes of shape (20,), max\_iter = 1000 and the Sigmoid - Logistic activation for all neurons. Comparing the results we get back based on the confusion matrix and classification report with the other models we use (Naive Bayes and Random Forest), the model gives us the best result with a score of 0.8 for both accuracy and weighted recall, while the weighted precision is at 0.76. The issue we have for using Neural Networks is that the recall score for “charged-off” loans is very low, at 0.12.

**Resampling**

We performed resampling by importing the Imbalanced-Learn package and performed RandomUnderSampling and SMOTE functions to our cleaned imbalance dataset. As suggested by prior studies *(Namvar et al, 2018*), resampling technique is deemed to be the proper solution when dealing with imbalance independent classes. Therefore, we will be comparing the classifiers result on the following section.



Technically, we split the data into 30% testing data and 70% training data. Our cleaned imbalance data have 629,394 full-paid and 167,609 charged-off data. This reflects 3.76:1 ratio between fully paid and charged-off data.

Out of multiple resampling methods, we would only perform two strategies as recommended by prior studies. Namvar et al. suggests Random Under Sampling (RUS) to yield the best predictive performance, while Costello et al. recommends Oversampling - Synthetic Minority Oversampling Technique (SMOTE) to be the best viable strategy. [9]

Using RUS would reduce our training data to 335,218 observations; it equally divides the data to 167,609 charged-off and 167,609 fully paid. The function randomly selected 167,709 observations out of its initial 629,394 fully paid data. On the other side, SMOTE boosted the charged-off data to 440,576 observations, which is the equal observations count for the training fully-paid data.

*Figure 6: Resampling process to rectify imbalance classes*

# **Interpretation**

**Performance Metrics**

Based on Namvar et al., accuracy would not be the best metric to evaluate imbalance class because accuracy tends to emphasize primarily on the majority, but not the minority class [9]. In terms of credit risk, accuracy can be a misleading criterion that causes erroneous results. Lenders are far worse off having to lend money to default borrowers that are predicted to be able to pay in full (false positives) than its false negative counterpart. Haixiang et al. confirms that Receiver Operating Characteristics (ROC), Area Under Curve (AUC), F1 measure (FM) are preferred as the likelihood that these measures will be affected by imbalance dataset is low. [10] Higher AUC describes a model's better ability to distinguish default and fully-paid borrowers, hence we will compare the improved AUC before and after data resampling.

**Classifier Results**

This section will discuss the performance scores of all four classifiers and datasets. Please refer to Table 3.1 for full classifier results. We found that using the entire dataset to train the model will result in the highest accuracy rate for NN amongst all other classifiers and datasets. However, our imbalance dataset results in the lowest recall rate at mere 9%. However considering our imbalance classes, we conclude that NB yields the highest AUC, F1 score, and RMSE. Despite its low accuracy, NB would be our most favorable model if trained with the whole dataset.

PCA with 25 components seemed to be counterproductive because NB yields to lower AUC, F1-score, and RMSE. Hence, we would not recommend using PCA to predict credit risks.

By using majority strategy in RUS, we found that our model has been substantially improved. Performing RF and NN would yield the highest AUC, RMSE, and F1 score in comparison to NB-Whole dataset. We found that our analysis aligns with Namvar et al., because the under-sampling method boosted our RF’s sensitivity rate by 58% and NN’s sensitivity rate by 55%. [9] Overall, we found that NN-RUS yields the best predictive power amongst all classifiers and dataset that we trained. Refer to Graph 1.7 and Graph 1.8 on the Graph and Tables section to analyze the changes on ROC and AUC.

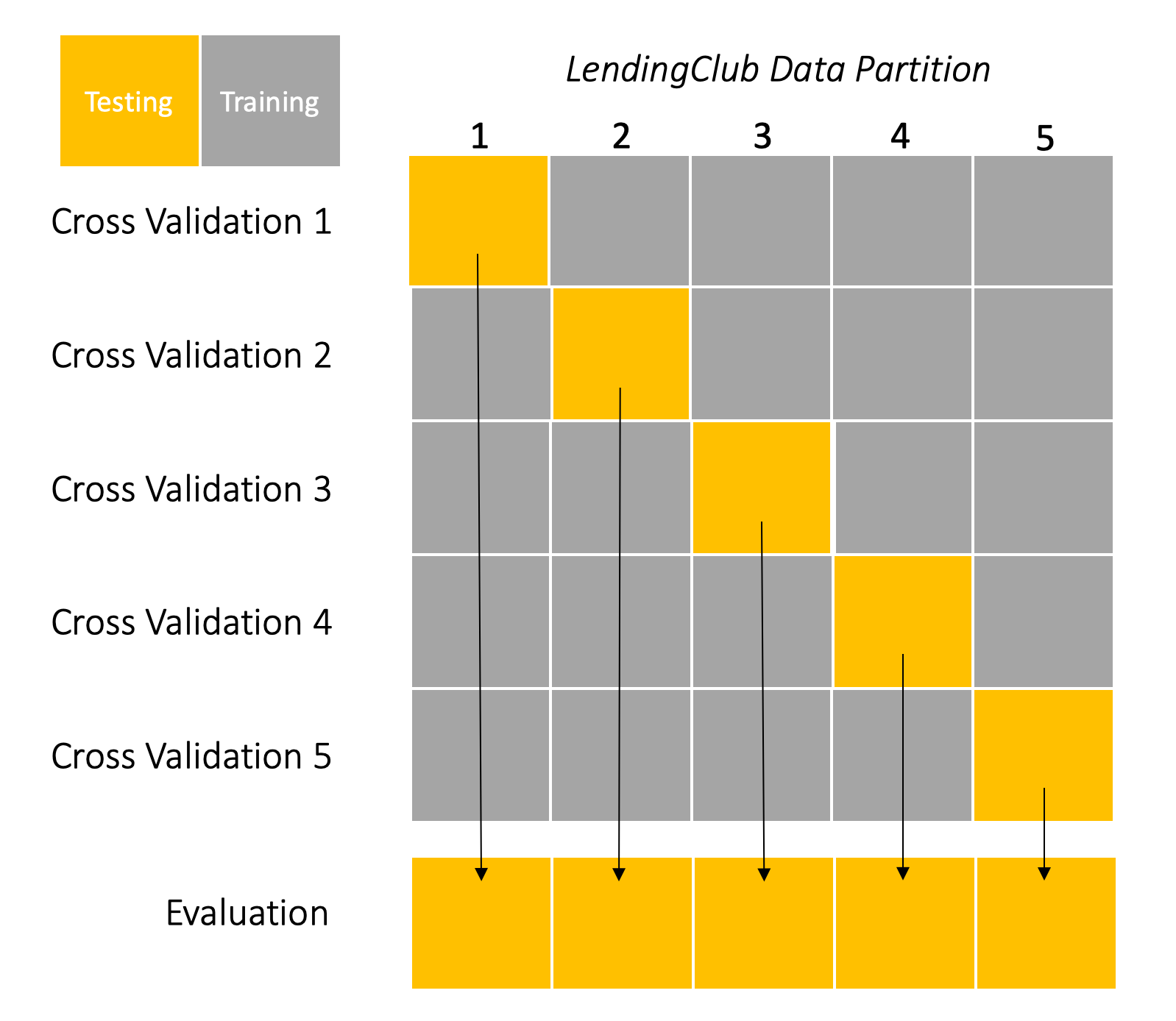
Our last analysis trained our data by using SMOTE as suggested by Costello et al. SMOTE technique does not solve RF and NN’s low sensitivity and precision rate. However, SMOTE boosted NB’s performance by increasing its sensitivity to 96% and RMSE to 89%, which is highest amongst all trained classifiers. [4]

In all, we found that RUS method to improve our model’s best performance by increasing its AUC, sensitivity, F1 score, and RMSE, but in return, reduce model’s Accuracy and Precision. In terms of credit risk prediction, we found that resampling methods would be beneficial because lowering False Positives errors should be prioritized to prevent erroneous effects on lender’s risk.

*Figure 7: Consolidated performance scores table by using 4 classifiers*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Classifier** | **PERFORMANCE SCORES** | | | | | |
| **Accuracy** | **AUC** | **Sensitivity** | **Precision** | **F1 score** | **RMSE** |
| Entire Processed Dataset | Naive Bayes | 0.55 | 0.63 | 0.77 | 0.29 | 0.42 | 0.67 |
| Decision Tree | 0.70 | 0.55 | 0.31 | 0.29 | 0.30 | 0.55 |
| Random Forest | 0.79 | 0.54 | 0.09 | 0.57 | 0.15 | 0.45 |
| Neural Network | 0.80 | 0.55 | 0.12 | 0.57 | 0.20 | 0.45 |
| PCA dataset (n = 25) | Naive Bayes | 0.77 | 0.58 | 0.25 | 0.41 | 0.31 | 0.45 |
| Decision Tree | 0.69 | 0.55 | 0.30 | 0.28 | 0.29 | 0.56 |
| Random Forest | 0.79 | 0.53 | 0.08 | 0.54 | 0.13 | 0.46 |
| Neural Network | 0.79 | 0.54 | 0.12 | 0.54 | 0.19 | 0.45 |
| Random Under Sampling (RUS) | Naive Bayes | 0.57 | 0.64 | 0.75 | 0.30 | 0.42 | 0.65 |
| Decision Tree | 0.58 | 0.58 | 0.57 | 0.27 | 0.36 | 0.65 |
| Random Forest | 0.65 | 0.66 | 0.67 | 0.34 | 0.45 | 0.59 |
| Neural Network | 0.66 | 0.66 | 0.67 | 0.34 | 0.45 | 0.58 |
| Synthetic Minority Oversampling Technique (SMOTE) | Naive Bayes | 0.26 | 0.52 | 0.96 | 0.22 | 0.35 | 0.89 |
| Decision Tree | 0.69 | 0.56 | 0.33 | 0.29 | 0.31 | 0.56 |
| Random Forest | 0.79 | 0.55 | 0.12 | 0.50 | 0.19 | 0.46 |
| Neural Network | 0.75 | 0.61 | 0.38 | 0.39 | 0.39 | 0.50 |
| K-fold Cross Validation (K = 5) | Naive Bayes | 0.60 | 0.68 | 0.64 | 0.31 | 0.39 | 0.63 |
| Decision Tree | 0.69 | 0.55 | 0.30 | 0.28 | 0.29 | 0.56 |
| Random Forest | 0.79 | 0.70 | 0.07 | 0.54 | 0.13 | 0.46 |
| Neural Network | 0.79 | 0.71 | 0.10 | 0.55 | 0.17 | 0.45 |

# **Section 5: Model Evaluations**

To allow all data points to be trained and tested against one another, we used the k-fold cross validation method to test the effectiveness of our chosen models. In terms of accuracy, we can see that the Naive Bayesian model is improved by 15 percent while the other models are relatively similar. Furthermore, we can also see that in terms of AUC, the Random Forest and Neural Network models gain 16 percent while the other models are relatively similar. Using sensitivity as a measure, the Naive Bayes model is reduced by 13 percent while the other models are relatively similar. Using other measurements of precision, f-1 scores, and RMSE, the percentages do not differ in all models.

*Figure 8: K-fold validation method on LC Data*

Additionally, by using k-fold cross validation, we have found that Naive Bayesian gives unreliable results as its performance measures fluctuate as tested with different cross validations. For example, its accuracy fluctuates from 77.20% to 60.02% to 63.19% to 47.61% to 51.40%. We, fortunately, would not have any inconsistency measurement problems with other models. This indicates that Naive Bayesian data might be inadequate for this data set or this method as it is highly reliant on historical events that have happened. Using this method, we can enhance the predictability of the model as we are switching training sets from one another and testing it against the remaining sets.

# **Section 6: Conclusion**

This paper has demonstrated the use of various data mining techniques to train a prediction model that could classify charged off and fully paid loans with up to 66% in accuracy, 67% and 34% in recall and precision, respectively, by using neural networks. With under sampling, Random Forest has also similar predictive performance at 65% accuracy, 67% recall, and 34% Precision. We found resampling methods to be highly effective in combating imbalance classes, and propose RUS to be the best approach in social credit risk assessment. This was achieved using information available for investors on the loan applications; included, but not limited to: interest rate, debt-to-income ratio, loan amount, borrowers’ credit history, and more. Even so, the accuracy was not that good compared with a blind guessing of whether a loan is going to be fully paid or charged off. This raises the question “What is the point of building a machine learning model to predict the status of a loan, instead of using the 50/50 chance of randomly guessing?”.

By conducting a completed data analysis and trying out different models, we could obviously provide great interpretations for investors on which features are important in defining the loan status and why a borrower might not pay fully on his or her loan. Our paper also shows the importance of using multiple sampling techniques for extremely imbalanced data to avoid bias towards the majority class, in this case the fully paid loans. Accordingly, our experiments of applying machine learning methods to four different models (Naive Bayes, Decision Tree, Random Forest and Neural Networks) show that Neural Networks and random under-sampling are the best combination of classifier and resampling strategy to predict a loan’s status, based on the accuracy, recall and precision scores.

Finally, our results point out that interest rate and debt-to-income rate are the two most important variables for predicting charge-off, which are incredibly helpful for investors in their decision making process. In particular, this model, while far from perfect, can assist investors to predict whether a loan will be fully paid or not, using only data available before the loan is fully funded as well as evaluate borrowers’ creditworthiness from many perspectives.

**Future work recommendations**

We worked with a large imbalance dataset, asymmetrical information, and redundant data. Data redundancy adds more weight to the information repeated in multiple features while not adding more worthwhile information. As a result, data redundancy suffers data balance and increases dimensionality. On the other hand, although we added GDP and unemployment features, there still are factors that contribute to defaulted loans, which are not included in our dataset. We can include more relevant and unique features having high correlation to loan default rate in historical data.

The data imbalance caused low sensitivity and precision in our model using the pre-processed dataset. Although RUS significantly improved the sensitivity and specificity, our model still has a low precision rate. For further improvement of the model, we can apply algorithmic ensemble techniques with modified random forests using weighted majority votes based on out-of-bag errors in tree aggregation as *Lifeng et al*. [6] did.

Finally, we noticed that our outlier analysis using z-score and IQR did not successfully handle outliers. As *Nonso et al*. [9] proposed, we can apply a two-step method using IQR and SMOTE to deal with both data imbalance and outlier problems in the data.

**References**

1. Hajek, P., & Michalak, K. (2013). Feature selection in corporate credit rating prediction. *Knowledge-Based Systems,* *51*, 72-84. doi:10.1016/j.knosys.2013.07.008. Retrieved July 2, 2020.
2. Malekipirbazari, M., & Aksakalli, V. (2015). Risk assessment in social lending via random forests. *Expert Systems with Applications,* *42*(10), 4621-4631. doi:10.1016/j.eswa.2015.02.001. Retrieved July 2, 2020.
3. Serrano-Cinca, C., Gutiérrez-Nieto, B., & López-Palacios, L. (2015). **Determinants of Default in P2P Lending**. PLoS ONE, 10(10), 1–22. <https://doi-org.proxy.seattleu.edu/10.1371/journal.pone.0139427>
4. Costello, F. J., & Kun Chang Lee. (2019). **Exploring the Performance of Synthetic Minority Over-sampling Technique (SMOTE) to Predict Good Borrowers in P2P Lending**. Journal of Digital Convergence, 17(9), 71–78. <https://doi-org.proxy.seattleu.edu/10.14400/JDC.2019.17.9.071>
5. P. Li and G. Han, ***LendingClub Loan Default and Profitability Prediction*,** [online] Available: <http://cs229.stanford.edu/proj2018/report/69.pdf>.
6. Lifeng Zhou, Hong Wang, Loan Default Prediction on Large Imbalanced Data Using Random Frest, TELKOMNKA Indonesian Journal of Electrical Engineering Vol. 10, No.6, October 2012, pp.1519-1525. <https://www.researchgate.net/publication/267864165_Loan_Default_Prediction_on_Large_Imbalanced_Data_Using_Random_Forests>
7. Tsai, Kevin, et al. “Peer Lending Risk Predictor.” Researchgate, 2014, [www.researchgate.net/profile/Sudhanshu\_Singh8/publication/269699712\_Peer\_Lending\_Risk\_Predictor/links/549321420cf286fe3125b7d3/Peer-Lending-Risk-Predictor.pdf.](http://www.researchgate.net/profile/Sudhanshu_Singh8/publication/269699712_Peer_Lending_Risk_Predictor/links/549321420cf286fe3125b7d3/Peer-Lending-Risk-Predictor.pdf.)
8. Nnamoko N, Korkontzelos I. Efficient treatment of outliers and class imbalance for diabetes prediction. Artificial Intelligence in Medicine. 2020 Apr;104:101815. DOI: 10.1016/j.artmed.2020.101815,

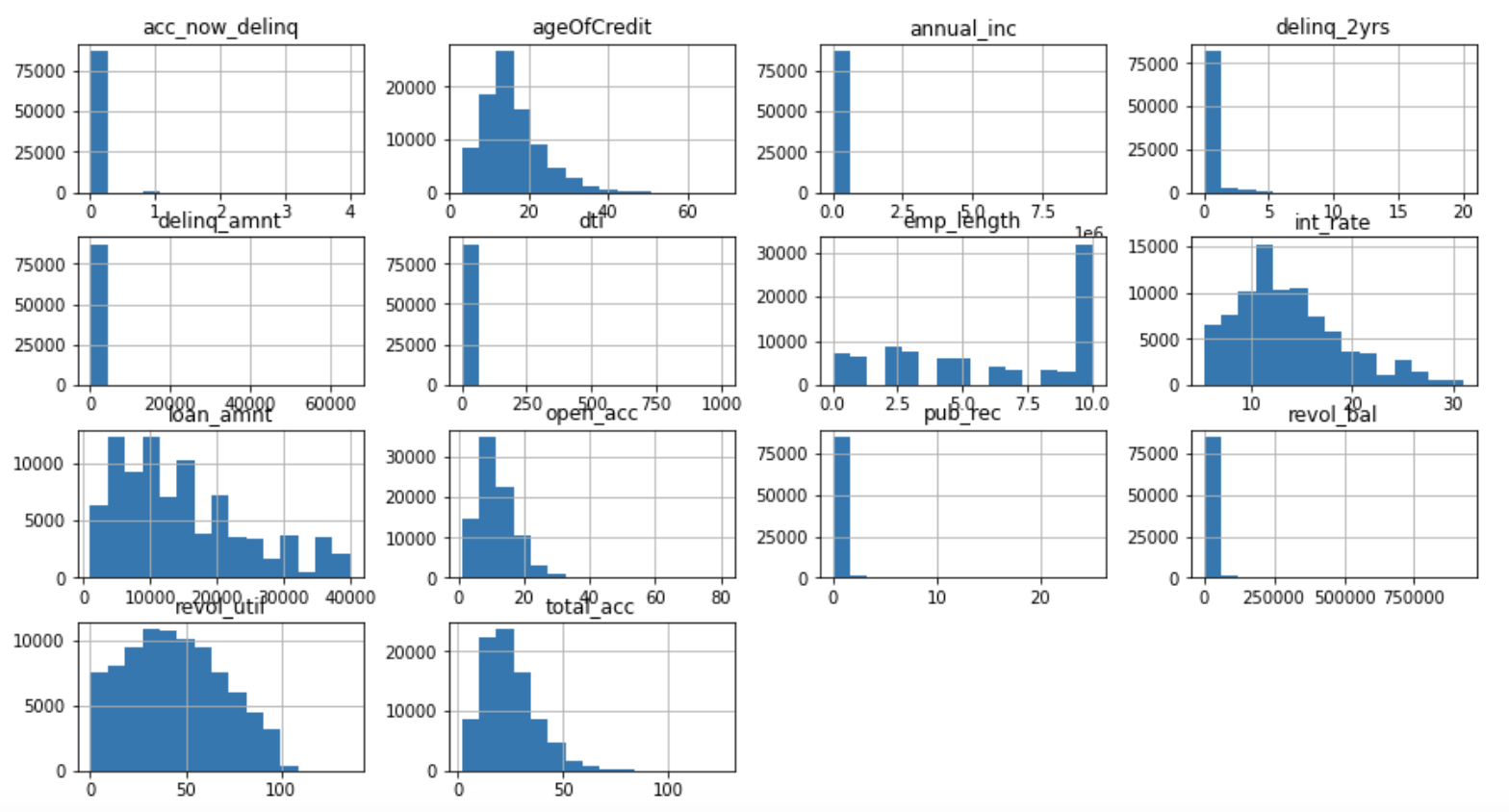
<https://www.sciencedirect.com/science/article/pii/S093336571830681X>

1. Anahita Namvar, et al. “Credit risk prediction in an imbalanced social lending environment.” Int. J. Comput. Intell. Cyst. 2018,

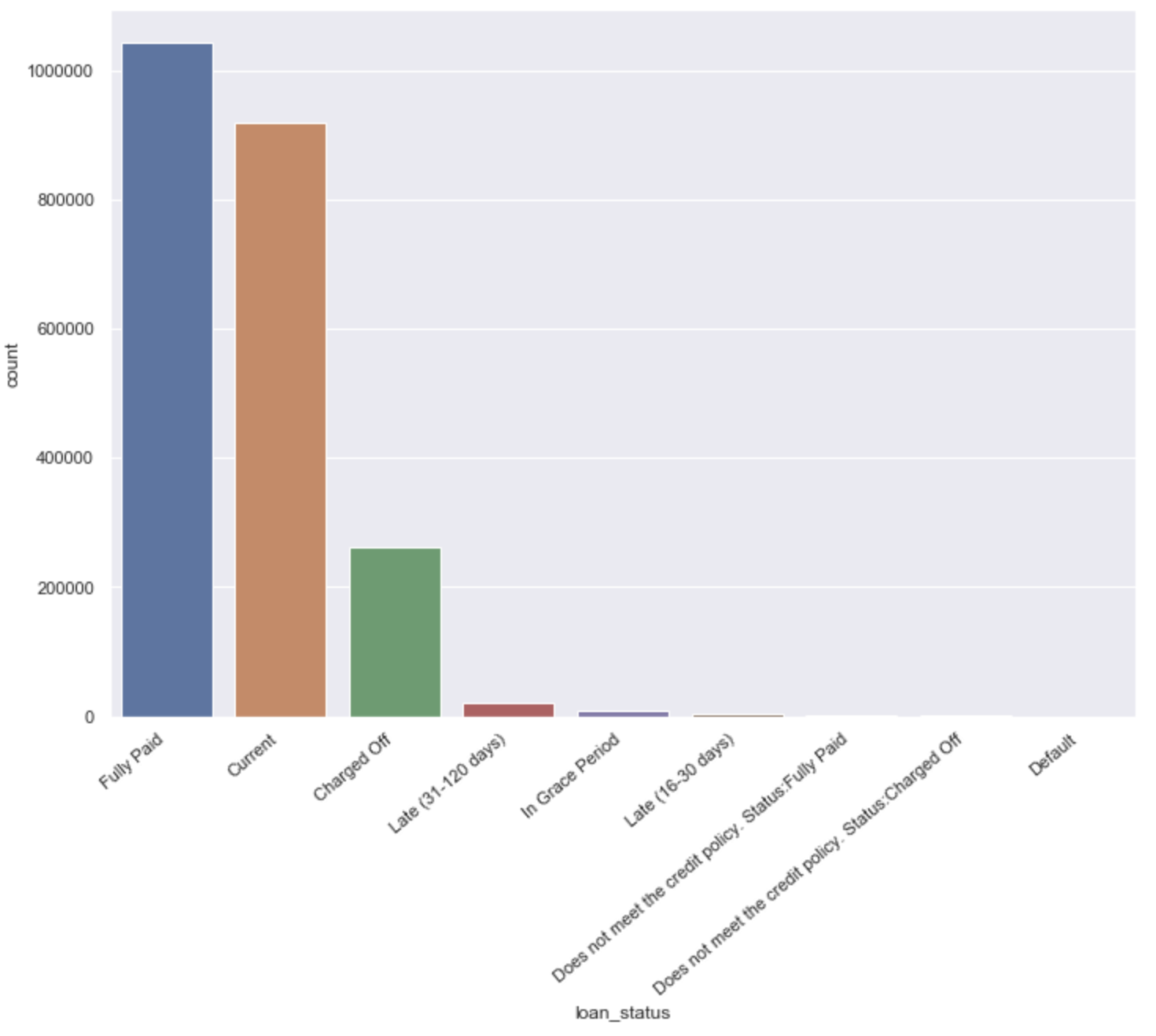
<https://arxiv.org/ftp/arxiv/papers/1805/1805.00801.pdf>

1. Haixiang, G., Yijing, L., Shang, J., Mingyun, G., Yuanyue, H., and Bing, G.: ‘Learning from class- imbalanced data: review of methods and applications’, Expert Systems with Applications, 2017, 73, pp. 220- 239

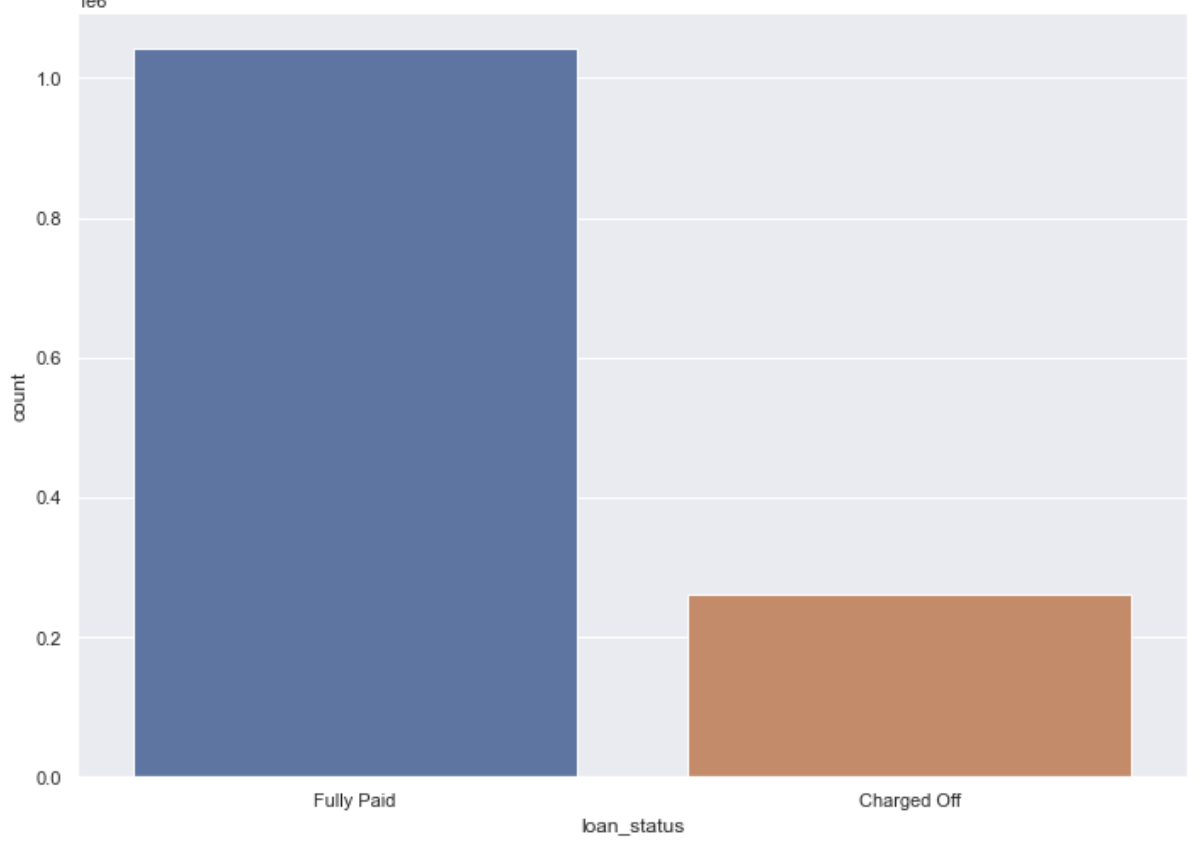
**Graphs and Tables**   
Graph 1.1 - Skewed distribution of LC numeric data



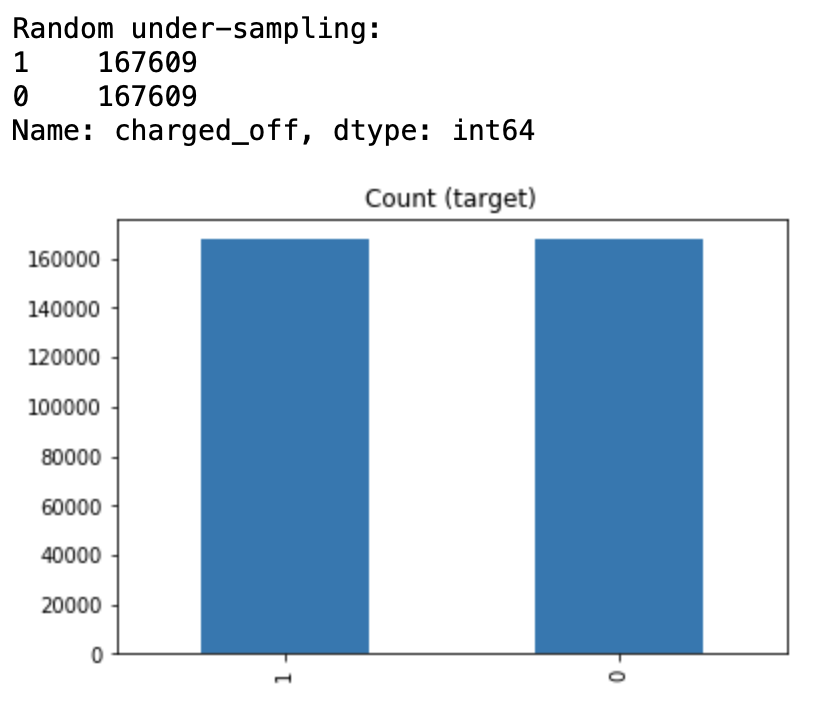
Graph 1.2 - Bar graph of Original Dataset - Loan Status frequency (before preprocessing)



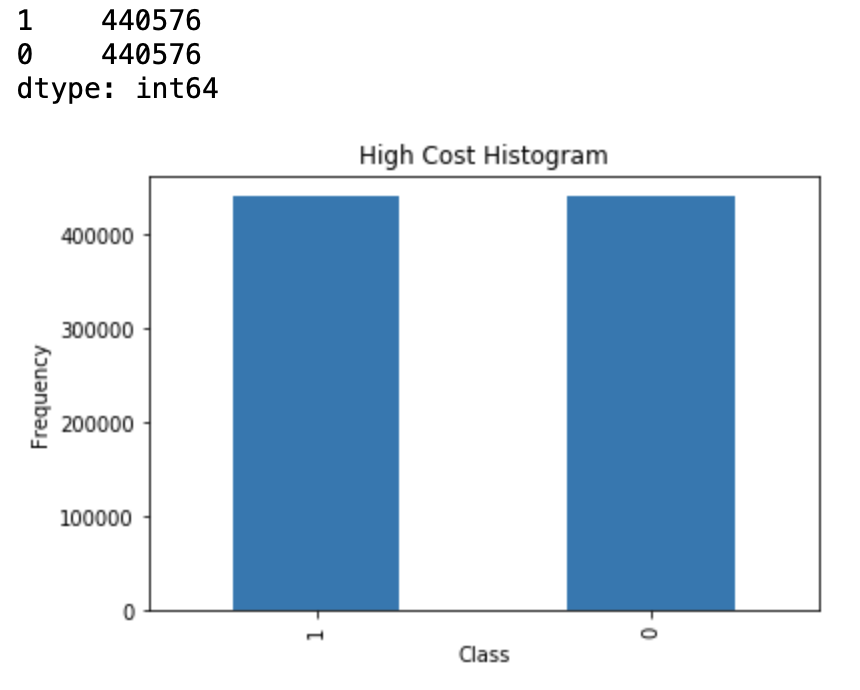
Graph 1.3 - Bar graph of pre-processed dataset - Loan Status Frequency (After preprocessing)



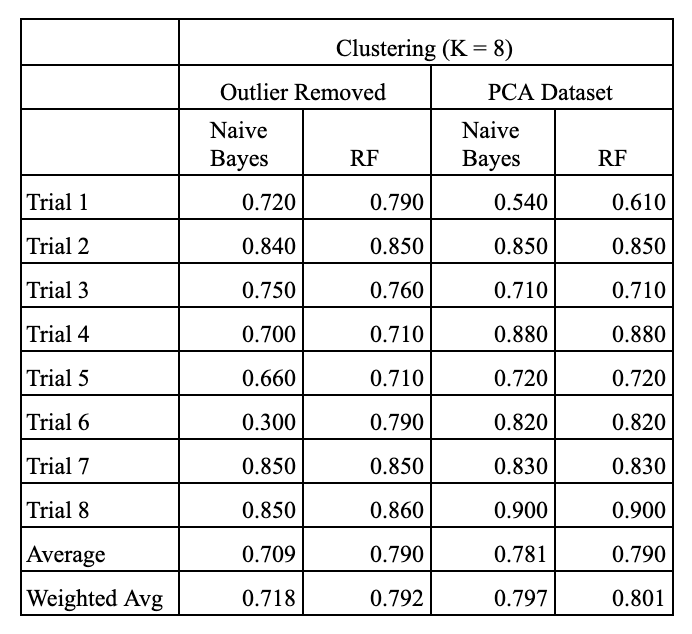
Graph 1.4 - Count of Loan\_Status after applying RUS - Majority technique



Graph 1.5 - Count of Loan\_status after applying SMOTE - Minority technique

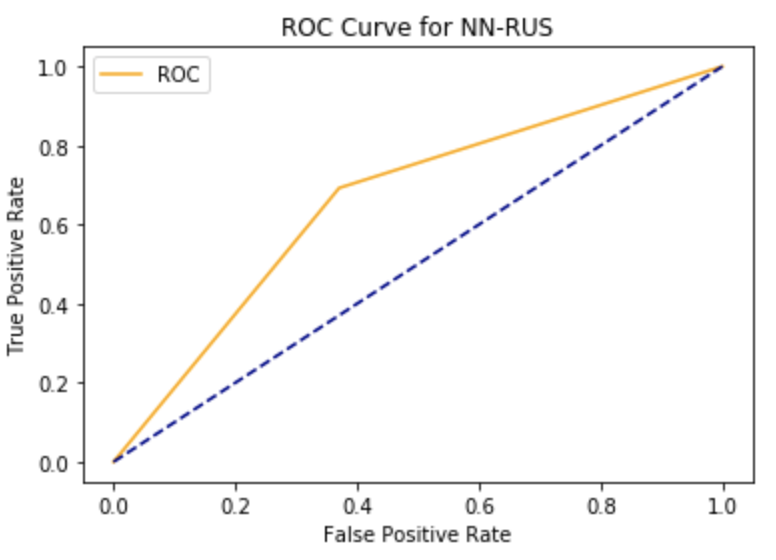
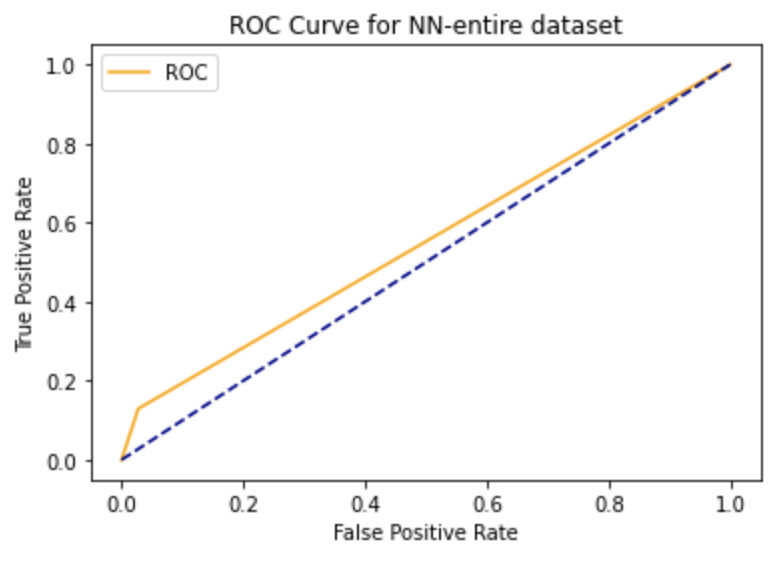


Graph 1.6 - Clustering attempt; selecting 8 clusters and ML result on two best performing classifiers. Decided to discard Clustering because it did not improve our predictive power.

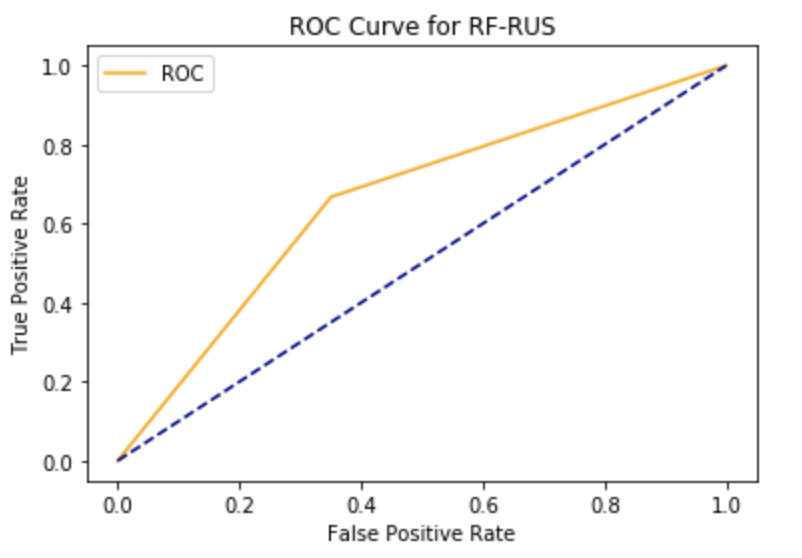
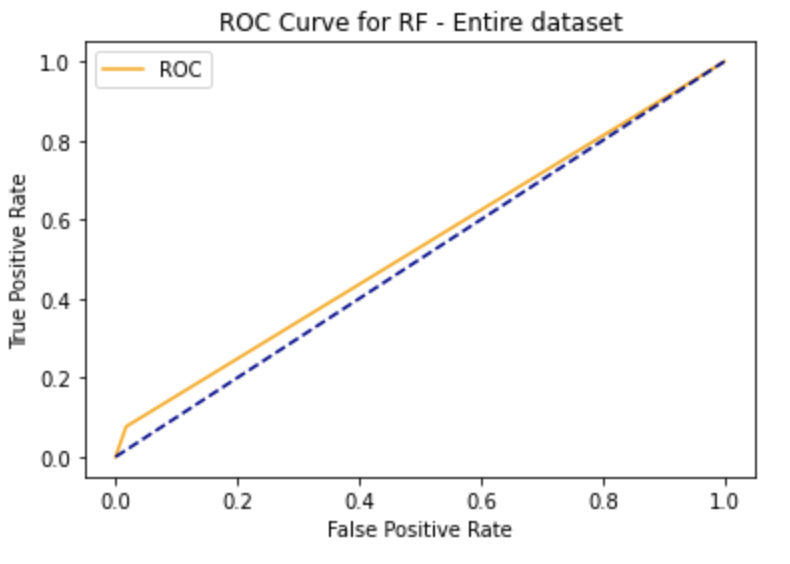


# A close up of text on a white background Description automatically generated

Graph 1.7 - ROC for NN entire dataset vs. NN RUS



Graph 1.8 - ROC for RF entire dataset vs. RF RUS



# **Appendix A: Data Dictionary**

1. **Original Data Frame Dictionary**

|  |  |
| --- | --- |
| **Field** | **Description** |
| **Index** | Position of the Column in a Microsoft Excel Worksheet. |
| **Column Name** | Name of the loan data element |
| **Description** | Definition of the loan data element. |
| **Data Type** | The type of data found in each column:  • Numeric – contains only numbers  • Ordinal – contains categorical data • Nominal – contains only texts  • Date – represents a specific date (Y = Year, M = Month) |
| **Value Description** | Allowable values for the specific data field and the calculations used (if applicable). |
| **Null Ratio** | The ratio between NULL observation over the total number of observations |

1. **Original DF**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Index** | **Column Name** | **Formal Name and Definition** | **Data Type** | **Length** | **Value description** | **Null ratio** |
| 1 | id | A unique LC assigned ID for the loan listing. | Nominal (Text) |  |  | 1.000 |
| 2 | member\_id | A unique LC assigned Id for the borrower member. | Nominal (Text) |  |  | 1.000 |
| 3 | loan\_amnt | The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value. | Numeric |  |  | 0.000 |
| 4 | funded\_amnt | funded\_amnt: The total amount committed to that loan at that point in time. | Numeric |  |  | 0.000 |
| 5 | funded\_amnt\_inv | The total amount committed by investors for that loan at that point in time. | Numeric |  |  | 0.000 |
| 6 | term | The number of payments on the loan. Values are in months and can be either 36 or 60. | Ordinal |  | 36 months , 60 months | 0.000 |
| 7 | int\_rate | Interest Rate on the loan | Numeric (float) |  |  | 0.000 |
| 8 | installment | The monthly payment owed by the borrower if the loan originates. | Numeric (float) |  |  | 0.000 |
| 9 | grade | LC assigned loan grade | Ordinal |  | A,B,C,D,E,F,G | 0.000 |
| 10 | sub\_grade | LC assigned loan subgrade | Ordinal |  | A1..A5, B1..B5, C1..C5, D1..D5, E1..E5, G1..G5 | 0.000 |
| 11 | emp\_title | The job title supplied by the Borrower when applying for the loan.\* | Nominal (Text) |  |  | 0.074 |
| 12 | emp\_length | Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years. | Ordinal |  | 10+ years', '6 years', '4 years', '< 1 year', '2 years', '9 years', nan, '5 years', '3 years', '7 years', '1 year', '8 years' | 0.065 |
| 13 | home\_ownership | home\_ownership: The home ownership status provided by the borrower during registration or obtained from the credit report. Our values are: RENT, OWN, MORTGAGE, OTHER | Nominal (Text) |  | RENT', 'MORTGAGE', 'OWN', 'ANY', 'NONE', 'OTHER' | 0.000 |
| 14 | annual\_inc | The self-reported annual income provided by the borrower during registration. | Numeric |  |  | 0.000 |
| 15 | verification\_status | Indicates if income was verified by LC, not verified, or if the income source was verified | Ordinal |  | Not Verified', 'Source Verified', 'Verified' | 0.000 |
| 16 | issue\_d | The month which the loan was funded | Date (Month) |  |  | 0.000 |
| 17 | loan\_status | Current status of the loan | Ordinal |  | Current', 'Fully Paid', 'Late (31-120 days)', 'In Grace Period', 'Charged Off', 'Late (16-30 days)', 'Default', 'Does not meet the credit policy. Status:Fully Paid', 'Does not meet the credit policy. Status:Charged Off' | 0.000 |
| 18 | pymnt\_plan | Indicates if a payment plan has been put in place for the loan | Ordinal |  | n', 'y' | 0.000 |
| 19 | url | URL for the LC page with listing data. | Nominal (Text) |  |  | 1.000 |
| 20 | desc | Loan description provided by the borrower | Nominal (Text) |  |  | 0.944 |
| 21 | purpose | A category provided by the borrower for the loan request. | Ordinal |  | debt\_consolidation', 'credit\_card', 'house', 'car', 'other', 'vacation', 'home\_improvement', 'small\_business', 'major\_purchase', 'medical', 'renewable\_energy', 'moving', 'wedding', 'educational' | 0.000 |
| 22 | title | The loan title provided by the borrower | Nominal |  |  | 0.001 |
| 23 | zip\_code | The first 3 numbers of the zip code provided by the borrower in the loan application. | Numeric/Categorical | 3 |  | 0.000 |
| 24 | addr\_state | The state provided by the borrower in the loan application | Nominal | 2 | NY', 'LA', 'MI', 'WA', 'MD', 'IN', 'IL', 'FL', 'CT', 'GA', 'UT', 'NC', 'KY', 'OH', 'AR', 'OK', 'CA', 'WV', 'NJ', 'SC', 'TX', 'PA', 'KS', 'AL', 'VA', 'MO', 'AZ', 'NM', 'CO', 'RI', 'WI', 'TN', 'NV', 'MA', 'NE', 'MN', 'NH', 'OR', 'VT', 'DC', 'MS', 'ID', 'DE', 'ND', 'HI', 'ME', 'AK', 'WY', 'MT', 'SD', 'IA' | 0.000 |
| 25 | dti | A ratio calculated using the borrower’s total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower’s self-reported monthly income. | Numeric (float) |  |  | 0.001 |
| 26 | delinq\_2yrs | delinq\_2yrs: The number of 30+ days past-due incidences of delinquency in the borrower's credit file for the past 2 years | Numeric |  | 0., 1., 2., 7., 4., 3., 6., 5., 8., 16., 14., 10., 11., 9., 17., 12., 21., 15., 13., 19., 23., 24., 18., 58., 35., 20., 22., 29., 30., 26., 27., 39., 28., 25., 32., 42., nan, 36. | 0.000 |
| 27 | earliest\_cr\_line | The month the borrower's earliest reported credit line was opened | Date (Month) |  |  | 0.000 |
| 28 | inq\_last\_6m | Number of credit inquiries in past 6 months | Ordinal |  |  | 0.512 |
| 29 | mths\_since\_last\_delinq | The number of months since the borrower's last delinquency. | Numeric |  |  | 0.841 |
| 30 | mths\_since\_last\_record | The number of months since the last public record. | Numeric |  |  | 0.000 |
| 31 | open\_acc | The number of open credit lines in the borrower's credit file. | Numeric |  |  | 0.000 |
| 32 | pub\_rec | Number of derogatory public records | Numeric |  |  | 0.000 |
| 33 | revol\_bal | Total credit revolving balance | Numeric (float) |  |  | 0.001 |
| 34 | revol\_util | Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit. | Numeric (float) |  |  | 0.000 |
| 35 | total\_acc | The total number of credit lines currently in the borrower's credit file | Numeric |  |  | 0.000 |
| 36 | initial\_list\_status | The initial listing status of the loan. Possible values are – W (Whole loans), F(Fractional) | Nominal | 1 | W = Whole, F = Fractional | 0.000 |
| 37 | out\_prncp | Remaining outstanding principal for total amount funded | Numeric |  |  | 0.000 |
| 38 | out\_prncp\_inv | Remaining outstanding principal for portion of total amount funded by investors | Numeric |  |  | 0.000 |
| 39 | total\_pymnt | Payments received to date for total amount funded | Numeric |  |  | 0.000 |
| 40 | total\_pymnt\_inv | Payments received to date for portion of total amount funded by investors | Numeric |  |  | 0.000 |
| 41 | total\_rec\_prncp | Principal received to date | Numeric |  |  | 0.000 |
| 42 | total\_rec\_int | Interest received to date | Numeric |  |  | 0.000 |
| 43 | total\_rec\_late\_fee | Late fees received to date | Numeric |  |  | 0.000 |
| 44 | recoveries | post charge off gross recovery | Numeric |  |  | 0.000 |
| 45 | collection\_recovery\_fee | post charge off collection fee | Numeric |  |  | 0.000 |
| 46 | last\_pymnt\_d | Last month payment was received | Date (Month) |  |  | 0.001 |
| 47 | last\_pymnt\_amnt | Last total payment amount received | Date (Month) |  |  | 0.000 |
| 48 | next\_pymnt\_d | Next scheduled payment date | Numeric |  |  | 0.577 |
| 49 | last\_credit\_pull\_d | The most recent month LC pulled credit for this loan | Date(Month) |  |  | 0.000 |
| 50 | collections\_12\_mths\_ex\_med | Number of collections in 12 months excluding medical collections | Numeric |  |  | 0.000 |
| 51 | mths\_since\_last\_major\_derog | Months since most recent 90-day or worse rating | Numeric |  |  | 0.743 |
| 52 | policy\_code | publicly available policy\_code=1 new products not publicly available policy\_code=2 | Nominal | 1 | 1' or '2' | 0.000 |
| 53 | application\_type | Indicates whether the loan is an individual application or a joint application with two co-borrowers | Nominal | 9 | Individual' or 'Joint App' | 0.000 |
| 54 | annual\_inc\_joint | The combined self-reported annual income provided by the co-borrowers during registration | Numeric |  |  | 0.947 |
| 55 | dti\_joint | A ratio calculated using the co-borrowers' total monthly payments on the total debt obligations, excluding mortgages and the requested LC loan, divided by the co-borrowers' combined self-reported monthly income | Numeric |  |  | 0.947 |
| 56 | verification\_status\_joint | Indicates if the co-borrowers' joint income was verified by LC, not verified, or if the income source was verified | Nominal | 12 | Verified' or 'Not Verified' or 'Source Verified' | 0.949 |
| 57 | acc\_now\_delinq | The number of accounts on which the borrower is now delinquent. | Numeric |  |  | 0.000 |
| 58 | tot\_coll\_amt | Total collection amounts ever owed | Numeric |  |  | 0.031 |
| 59 | tot\_cur\_bal | Total current balance of all accounts | Numeric |  |  | 0.031 |
| 60 | open\_acc\_6m | Number of open trades in last 6 months | Numeric |  |  | 0.383 |
| 61 | open\_act\_il | Number of currently active installment trades | Numeric |  |  | 0.383 |
| 62 | open\_il\_12m | Number of installment accounts opened in past 12 months | Numeric |  |  | 0.383 |
| 63 | open\_il\_24m | Number of installment accounts opened in past 24 months | Numeric |  |  | 0.383 |
| 64 | mths\_since\_rcnt\_il | Months since most recent installment accounts opened | Numeric |  |  | 0.403 |
| 65 | total\_bal\_il | Total current balance of all installment accounts | Numeric |  |  | 0.383 |
| 66 | il\_util | Ratio of total current balance to high credit/credit limit on all install acct | Numeric |  |  | 0.473 |
| 67 | open\_rv\_12m | Number of revolving trades opened in past 12 months | Numeric |  |  | 0.383 |
| 68 | open\_rv\_24m | Number of revolving trades opened in past 24 months | Numeric |  |  | 0.383 |
| 69 | max\_bal\_bc | Maximum current balance owed on all revolving accounts | Numeric |  |  | 0.383 |
| 70 | all\_util | Balance to credit limit on all trades | Numeric |  |  | 0.383 |
| 71 | total\_rev\_hi\_lim | Total revolving high credit/credit limit | Numeric |  |  | 0.031 |
| 72 | inq\_fi | Number of personal finance inquiries | Numeric |  |  | 0.383 |
| 73 | total\_cu\_tl | The total number of finances trade | Numeric | 3 |  | 0.383 |
| 74 | inq\_last\_12m | The total number of inqueries for the last 12 months | Numeric | 2 |  | 0.383 |
| 75 | acc\_open\_past\_24mths | The total number of account opened the past 24 months | Numeric | 2 |  | 0.022 |
| 76 | avg\_cur\_bal | The average currenct balance of all accounts | Numeric | 5 |  | 0.033 |
| 77 | bc\_open\_to\_buy | The total open to buy on revolving bankcards | Numeric | 5 |  | 0.033 |
| 78 | bc\_util | Ration of total current balance to high credit/credit limit for all bankcard accounts. | Numeric (float) | 3 |  | 0.034 |
| 79 | chargeoff\_within\_12\_mths | Number of charge-offs within 12 months | Numeric | 1 |  | 0.000 |
| 80 | delinq\_amnt | The past-due amount owed for the accounts on which the borrower is now delinquent. | Numeric | 4 |  | 0.000 |
| 81 | mo\_sin\_old\_il\_acct | Months since oldest bank installment account opened | Numeric | 3 |  | 0.062 |
| 82 | mo\_sin\_old\_rev\_tl\_op | Months since oldest revolving account opened | Numeric | 3 |  | 0.031 |
| 83 | mo\_sin\_rcnt\_rev\_tl\_op | Months since most recent revolving account opened | Numeric | 3 |  | 0.031 |
| 84 | mo\_sin\_rcnt\_tl | Months since most recent account opened | Numeric | 3 |  | 0.031 |
| 85 | mort\_acc | Number of mortgage accounts. | Numeric | 2 |  | 0.022 |
| 86 | mths\_since\_recent\_bc | Months since most recent bankcard account opened. | Numeric | 2 |  | 0.032 |
| 87 | mths\_since\_recent\_bc\_dlq | Months since most recent bankcard delinquency | Numeric | 2 |  | 0.770 |
| 88 | mths\_since\_recent\_inq | Months since most recent inquiry. | Numeric | 2 |  | 0.131 |
| 89 | mths\_since\_recent\_revol\_delinq | Months since most recent revolving delinquency. | Numeric | 2 |  | 0.673 |
| 90 | num\_accts\_ever\_120\_pd | Number of accounts ever 120 or more days past due | Numeric | 2 |  | 0.031 |
| 91 | num\_actv\_bc\_tl | Number of currently active bankcard accounts | Numeric | 2 |  | 0.031 |
| 92 | num\_actv\_rev\_tl | Number of currently active revolving trades | Numeric | 2 |  | 0.031 |
| 93 | num\_bc\_sats | Number of satisfactory bankcard accounts | Numeric | 2 |  | 0.026 |
| 94 | num\_bc\_tl | Number of bankcard accounts | Numeric | 2 |  | 0.031 |
| 95 | num\_il\_tl | Number of installment accounts | Numeric | 2 |  | 0.031 |
| 96 | num\_op\_rev\_tl | Number of open revolving accounts | Numeric | 2 |  | 0.031 |
| 97 | num\_rev\_accts | Number of revolving accounts | Numeric | 2 |  | 0.031 |
| 98 | num\_rev\_tl\_bal\_gt\_0 | Number of revolving trades with balance >0 | Numeric | 2 |  | 0.031 |
| 99 | num\_sats | Number of satisfactory accounts | Numeric | 2 |  | 0.026 |
| 100 | num\_tl\_120dpd\_2m | Number of accounts currently 120 days past due (updated in past 2 months) | Numeric | 2 |  | 0.068 |
| 101 | num\_tl\_30dpd | Number of accounts currently 30 days past due (updated in past 2 months) | Numeric | 2 |  | 0.031 |
| 102 | num\_tl\_90g\_dpd\_24m | Number of accounts 90 or more days past due in last 24 months | Numeric | 2 |  | 0.031 |
| 103 | num\_tl\_op\_past\_12m | Number of accounts opened in past 12 months | Numeric | 2 |  | 0.031 |
| 104 | pct\_tl\_nvr\_dlq | Percent of trades never delinquent | Numeric (float) | 3 |  | 0.031 |
| 105 | percent\_bc\_gt\_75 | Percentage of all bankcard accounts > 75% of limit. | Numeric (float) | 3 |  | 0.033 |
| 106 | pub\_rec\_bankruptcies | Number of public record bankruptcies | Numeric | 1 | 1 = Yes; 0 = No | 0.010 |
| 107 | tax\_liens | Number of tax liens | Numeric | 1 |  | 0.000 |
| 108 | tot\_hi\_cred\_lim | Total high credit/credit limit | Numeric |  |  | 0.031 |
| 109 | total\_bal\_ex\_mort | Total credit balance excluding mortgage | Numeric |  |  | 0.022 |
| 110 | total\_bc\_limit | Total bankcard high credit/credit limit | Numeric |  |  | 0.022 |
| 111 | total\_il\_high\_credit\_limit | Total installment high credit/credit limit | Numeric |  |  | 0.031 |
| 112 | revol\_bal\_joint | Sum of revolving credit balance of the co-borrowers, net of duplicate balances | Numeric |  |  | 0.952 |
| 113 | sec\_app\_earliest\_cr\_line | Earliest credit line at time of application for the secondary applicant | Date |  |  | 0.952 |
| 114 | sec\_app\_inq\_last\_6mths | Credit inquiries in the last 6 months at time of application for the secondary applicant | Numeric |  |  | 0.952 |
| 115 | sec\_app\_mort\_acc | Number of mortgage accounts at time of application for the secondary applicant | Numeric |  |  | 0.952 |
| 116 | sec\_app\_open\_acc | Number of open trades at time of application for the secondary applicant | Numeric |  |  | 0.952 |
| 117 | sec\_app\_revol\_util | Ratio of total current balance to high credit/credit limit for all revolving accounts | Numeric |  |  | 0.953 |
| 118 | sec\_app\_open\_act\_il | Number of currently active installment trades at time of application for the secondary applicant | Numeric |  |  | 0.952 |
| 119 | sec\_app\_num\_rev\_accts | Number of revolving accounts at time of application for the secondary applicant | Numeric |  |  | 0.952 |
| 120 | sec\_app\_chargeoff\_within\_12\_mths | Number of charge-offs within last 12 months at time of application for the secondary applicant | Numeric |  |  | 0.952 |
| 121 | sec\_app\_collections\_12\_mths\_ex\_med | Number of collections within last 12 months excluding medical collections at time of application for the secondary applicant | Numeric |  |  | 0.952 |
| 122 | sec\_app\_mths\_since\_last\_major\_derog | Months since most recent 90-day or worse rating at time of application for the secondary applicant | Numeric |  |  | 0.984 |
| 123 | hardship\_flag | Flags whether or not the borrower is on a hardship plan | Nominal |  | Y = Yes, N = No | 0.000 |
| 124 | hardship\_type | Describes the hardship plan offering | Ordinal |  |  | 0.995 |
| 125 | hardship\_reason | Describes the reason the hardship plan was offered | Ordinal |  |  | 0.995 |
| 126 | hardship\_status | Describes if the hardship plan is active, pending, canceled, completed, or broken | Ordinal |  | active, pending, canceled, completed, or broken | 0.995 |
| 127 | deferral\_term | Describes if the hardship plan is active, pending, canceled, completed, or broken | Numeric |  |  | 0.995 |
| 128 | hardship\_amount | The interest payment that the borrower has committed to make each month while they are on a hardship plan | Numeric |  |  | 0.995 |
| 129 | hardship\_start\_date | The start date of the hardship plan period | Date |  |  | 0.995 |
| 130 | hardship\_end\_date | The end date of the hardship plan period | Date |  |  | 0.995 |
| 131 | payment\_plan\_start\_date | The day the first hardship plan payment is due. For example, if a borrower has a hardship plan period of 3 months, the start date is the start of the three-month period in which the borrower is allowed to make interest-only payments. | Date |  |  | 0.995 |
| 132 | hardship\_length | The number of months the borrower will make smaller payments than normally obligated due to a hardship plan | Numeric |  |  | 0.995 |
| 133 | hardship\_dpd | Account days past due as of the hardship plan start date | Numeric |  |  | 0.995 |
| 134 | hardship\_loan\_status | Loan Status as of the hardship plan start date | Ordinal |  |  | 0.995 |
| 135 | orig\_projected\_additional\_accrued\_interest | The original projected additional interest amount that will accrue for the given hardship payment plan as of the Hardship Start Date. This field will be null if the borrower has broken their hardship payment plan. | Numeric |  |  | 0.996 |
| 136 | hardship\_payoff\_balance\_amount | The payoff balance amount as of the hardship plan start date | Numeric |  |  | 0.995 |
| 137 | hardship\_last\_payment\_amount | The last payment amount as of the hardship plan start date | Numeric |  |  | 0.995 |
| 138 | disbursement\_method | The method by which the borrower receives their loan. Possible values are: CASH, DIRECT\_PAY | Ordinal |  | CASH, DIRECT\_PAY | 0.000 |
| 139 | debt\_settlement\_flag | Flags whether or not the borrower, who has charged-off, is working with a debt-settlement company. | Nominal |  | Y = Yes, N = No | 0.000 |
| 140 | debt\_settlement\_flag\_date | The most recent date that the Debt\_Settlement\_Flag has been set | Date |  |  | 0.985 |
| 141 | settlement\_status | The status of the borrower’s settlement plan. Possible values are: COMPLETE, ACTIVE, BROKEN, CANCELLED, DENIED, DRAFT | Ordinal |  | COMPLETE, ACTIVE, BROKEN, CANCELLED, DENIED, DRAFT | 0.985 |
| 142 | settlement\_date | The date that the borrower agrees to the settlement plan | Date |  |  | 0.985 |
| 143 | settlement\_amount | The loan amount that the borrower has agreed to settle for | Numeric |  |  | 0.985 |
| 144 | settlement\_percentage | The settlement amount as a percentage of the payoff balance amount on the loan | Numeric |  |  | 0.985 |
| 145 | settlement\_term | The number of months that the borrower will be on the settlement plan | Numeric |  |  | 0.985 |

1. **DF2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Index** | **Column Name** | **Formal Name and Definition** | **Data Type** | **Value description** |
| 1 | Diff | The difference of quarterly GDP from this quarter to last quarter | Numerical |  |
| 2 | Month | The month of the issue quarter | Date (Month) |  |
| 4 | UR | Unemployment rate of the month of the issue | Numerical |  |
| 5 | URDiff | The difference of this month unemployment rate to last month unemployment rate | Numerical |  |
| 6 | URprevMonthDiff | The difference of last month unemployment rate to unemployment rate 2 months ago | Numerical |  |
| 7 | acc\_now\_delinq | The number of accounts on which the borrower is now delinquent. | Numerical |  |
| 8 | addr\_state | The state provided by the borrower in the loan application | Nominal | NY', 'LA', 'MI', 'WA', 'MD', 'IN', 'IL', 'FL', 'CT', 'GA', 'UT', 'NC', 'KY', 'OH', 'AR', 'OK', 'CA', 'WV', 'NJ', 'SC', 'TX', 'PA', 'KS', 'AL', 'VA', 'MO', 'AZ', 'NM', 'CO', 'RI', 'WI', 'TN', 'NV', 'MA', 'NE', 'MN', 'NH', 'OR', 'VT', 'DC', 'MS', 'ID', 'DE', 'ND', 'HI', 'ME', 'AK', 'WY', 'MT', 'SD', 'IA' |
| 9 | ageOfCredit | The age of credit in years between the issue date and the applicant's date of first credit | Numerical |  |
| 10 | annual\_inc | The self-reported annual income provided by the borrower during registration. | Numerical |  |
| 11 | application\_type | Indicates whether the loan is an individual application or a joint application with two co-borrowers | Nominal | Individual' or 'Joint App' |
| 12 | collections\_12\_mths\_ex\_med | Number of collections in 12 months excluding medical collections | Numerical |  |
| 13 | delinq\_2yrs | delinq\_2yrs: The number of 30+ days past-due incidences of delinquency in the borrower's credit file for the past 2 years | Numerical |  |
| 14 | delinq\_amnt | The past-due amount owed for the accounts on which the borrower is now delinquent. | Numerical |  |
| 15 | dti | A ratio calculated using the borrower’s total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower’s self-reported monthly income. | Numerical |  |
| 16 | emp\_length | Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years. | Ordinal | 10+ years', '6 years', '4 years', '< 1 year', '2 years', '9 years', nan, '5 years', '3 years', '7 years', '1 year', '8 years' |
| 17 | grade | LC assigned loan grade | Ordinal | A,B,C,D,E,F,G |
| 18 | home\_ownership | home\_ownership: The home ownership status provided by the borrower during registration or obtained from the credit report. Our values are: RENT, OWN, MORTGAGE, OTHER | Nominal | RENT', 'MORTGAGE', 'OWN', 'ANY', 'NONE', 'OTHER' |
| 19 | initial\_list\_status | The initial listing status of the loan. Possible values are – W (Whole loans), F(Fractional) | Nominal | W = Whole, F = Fractional |
| 20 | inq\_last\_6mths | Number of credit inquiries in past 6 months | Numerical |  |
| 21 | int\_rate | Interest Rate on the loan | Numerical |  |
| 22 | issue\_d | The month which the loan was funded | Date (Month) |  |
| 23 | loan\_amnt | The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value. | Numeric |  |
| 24 | loan\_status | Current status of the loan | Ordinal | Current', 'Fully Paid', 'Late (31-120 days)', 'In Grace Period', 'Charged Off', 'Late (16-30 days)', 'Default', 'Does not meet the credit policy. Status:Fully Paid', 'Does not meet the credit policy. Status:Charged Off' |
| 25 | log\_annual\_inc | Log base 10 of annual income | Numerical |  |
| 26 | log\_revol\_bal | Log base 10 of revolving balance | Numerical |  |
| 27 | open\_acc | The number of open credit lines in the borrower's credit file. | Numerical |  |
| 28 | prevQuarterDiff | The difference between previous quarter GDP to GDP two quarters ago | Numerical |  |
| 29 | pub\_rec | Number of derogatory public records | Numerical |  |
| 30 | purpose | A category provided by the borrower for the loan request. | Ordinal | debt\_consolidation', 'credit\_card', 'house', 'car', 'other', 'vacation', 'home\_improvement', 'small\_business', 'major\_purchase', 'medical', 'renewable\_energy', 'moving', 'wedding', 'educational' |
| 31 | revol\_bal | Total credit revolving balance | Numerical |  |
| 32 | revol\_util | Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit. | Numerical |  |
| 33 | term | The number of payments on the loan. Values are in months and can be either 36 or 60. | Nominal | 36 months , 60 months |
| 34 | tot\_coll\_amt | Total collection amounts ever owed | Numerical |  |
| 35 | total\_acc | The total number of credit lines currently in the borrower's credit file | Numerical |  |
| 36 | verification\_status | Indicates if income was verified by LC, not verified, or if the income source was verified | Ordinal | Not Verified', 'Source Verified', 'Verified' |

1. **BLS Unemployment Rate**

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| FIPS Code | Codes for the identification of the States, the District of Columbia, Puerto Rico, and the Insular Areas of the United States.  https://catalog.data.gov/dataset/fips-state-codes |
| state and area | Name for the identification of the area |
| period | Year and Month |
| Civilian non-institutional population | Included are persons 16 years of age and older residing in the 50 states and the District of Columbia who do not live in institutions (for example, correctional facilities, long-term care hospitals, and nursing homes) and who are not on active duty in the Armed Forces. |
| total civilian labor force | The labor force includes all persons classified as employed or unemployed in accordance with the definitions contained in this glossary. |
| percentage civilian labor force to civilian non-institutional population | Percentage of total civilian labor force to the civilian non-institutional population |
| employed civilian labor force | Persons 16 years and over in the civilian noninstitutional population who, during the reference week, (a) did any work at all (at least 1 hour) as paid employees; worked in their own business, profession, or on their own farm, or worked 15 hours or more as unpaid workers in an enterprise operated by a member of the family; and (b) all those who were not working but who had jobs or businesses from which they were temporarily absent because of vacation, illness, bad weather, childcare problems, maternity or paternity leave, labor-management dispute, job training, or other family or personal reasons, whether or not they were paid for the time off or were seeking other jobs. Each employed person is counted only once, even if he or she holds more than one job. Excluded are persons whose only activity consisted of work around their own house (painting, repairing, or own home housework) or volunteer work for religious, charitable, and other organizations. |
| percentage employed civilian labor force to total civilian labor force population | Percentage of employed civilian labor force to total civilian labor force population |
| unemployed civilian labor force | The number of persons aged 16 years and older who had no employment during the reference week, were available for work, except for temporary illness, and had made specific efforts to find employment sometime during the 4-week period ending with the reference week. Persons who were waiting to be recalled to a job from which they had been laid off need not have been looking for work to be classified as unemployed. |
| percentage unemployed civilian labor force to total civilian labor force population | Percentage of unemployed civilian labor force to total civilian labor force |

1. **BEA Quarterly GDP per State**

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| **geofips** | Codes for the identification of the States, the District of Columbia, Puerto Rico, and the Insular Areas of the United States.  https://catalog.data.gov/dataset/fips-state-codes |
| **geoname** | Name for the identification of the area |
| **region** | Region number for the area |
| **TableName** | Table category for the group of areas |
| **LineCODE** | Line code for the description |
| **IndustryClassification** | Empty value |
| **Description** | **Real GDP (millions of chained 2012 dollars): Real GDP by state is an inflation-adjusted measure of each state's gross product that is based on national prices for the goods and services produced within the state. The real estimates of gross domestic product (GDP) by state are measured in chained (2012) dollars (SIC-based statistics are in chained (1997) dollars)  Chain-type quantity indexes for real GDP: A quantity index is an index number that measures the change in the level of a quantity from a base year, apart from any changes in relative prices. The value of the quantity index is 100 for the base year.BEA uses chain-type annual-weighted indexes, also known as Fisher indexes, as its measure of real output and prices. These measures allow for the effects of changes in relative prices and in the composition of output over time, thereby eliminating a major source of bias inherent in fixed-weight indexes.  Current-dollar GDP (millions of current dollars): Gross domestic product (GDP) by state the measure of the market value of all final goods and services produced within a state in a particular period of time. In concept, an industry's GDP by state, referred to as its "value added", is equivalent to its gross output (sales or receipts and other operating income, commodity taxes, and inventory change) minus its intermediate inputs (consumption of goods and services purchased from other U.S. industries or imported). GDP by state is the state counterpart of the Nation's GDP, the Bureau's featured and most comprehensive measure of U.S. economic activity. GDP by state differs from national GDP for the following reasons: GDP by state excludes and national GDP includes the compensation of federal civilian and military personnel stationed abroad and government consumption of fixed capital for military structures located abroad and for military equipment, except office equipment; and GDP by state and GDP have different revision schedules.** |
| **Unit** | Units of measurement |
| **years and quarter** | Format is “XXXX:QY” where XXXX is the year and Y is the period of the quarter. |

# **Appendix B: Python Codes**

1. **Opening File and Joining Other Data Sets**
   1. **Library List:**

|  |
| --- |
| **import numpy as np import pandas as pd** |

* 1. **Opening Data:**

|  |
| --- |
| **df = pd.read\_csv("loan.csv", low\_memory = False) df.shape  Result:  (2260668, 145)** |

* 1. **Changing Several Columns’ Data Type and Adding Needed Columns:**

|  |
| --- |
| **# Change certain columns to date time  ## Getting the year out of the month-year format in the column df['earliest\_cr\_line\_year'] = df['earliest\_cr\_line'].str.strip().str[-4:].fillna(0).astype('int')  ## Changing earliest\_cr\_line into pd\_datetime format df['earliest\_cr\_line'] = pd.to\_datetime(df['earliest\_cr\_line'])  ## Changing issue\_d into pd\_datetime format df['issue\_d'] = pd.to\_datetime(df['issue\_d'])  # Create new column in loan data of previous quarter df['pqissue\_d'] = df['issue\_d'] - pd.tseries.offsets.DateOffset(months = 3)  # Create new column in loan data of current quarter of issue date df['issue\_q'] = pd.PeriodIndex(pd.to\_datetime(df['issue\_d']), freq = 'Q')  # Create new column in loan data for quarter of issue\_date minus 1 quarter - previous quarter issue quarter df['pqissue\_q'] = pd.PeriodIndex(pd.to\_datetime(df['pqissue\_d']), freq = 'Q')  # Change earliest\_credit\_line and issue\_date to correct datetime format to calculate age of Credit df['earliest\_cr\_line'] = pd.to\_datetime(df['earliest\_cr\_line']) df['issue\_d'] = pd.to\_datetime(df['issue\_d'])  # Change last payment date to datetime format df['last\_pymnt\_d'] = pd.to\_datetime(df['last\_pymnt\_d'])** |

|  |
| --- |
| **# Calculate age of credit in year df['ageOfCredit'] =  ((df['issue\_d']-df['earliest\_cr\_line']))/np.timedelta64(1,'Y')** |

|  |
| --- |
| **# Transforming employment length longer than 10 years to 10 years and less than 1 years to 0 years**  **df['emp\_length'].replace(to\_replace='10+ years', value='10 years', inplace=True) df['emp\_length'].replace('< 1 year', '0 years', inplace=True)   def emp\_length\_to\_int(s):  if pd.isnull(s):  return s  else:  return np.int8(s.split()[0])  df['emp\_length'] = df['emp\_length'].apply(emp\_length\_to\_int)** |

|  |
| --- |
| **# Log base 10 the two numeric variables for easier visualizations**  **df['log\_annual\_inc'] = df['annual\_inc'].apply(lambda x: np.log10(x+1)) df['log\_revol\_bal'] = df['revol\_bal'].apply(lambda x: np.log10(x+1))** |

* 1. **Opening Other Data Set for Easier Join of State Names**

|  |
| --- |
| **# Open csv state abbrv to state name sn = pd.read\_csv("https://worldpopulationreview.com/static/states/abbr-name.csv", header = None)  # Rename column  sn = sn.rename(columns = {0: "addr\_state" , 1:"length\_as"})  # Merge df = df.merge(sn, how = 'left', on = 'addr\_state')** |

|  |
| --- |
| **# Changing the o in the new joined data set to capital O for joining with other data sets**  **df.loc[df['length\_as'] == 'District Of Columbia', 'length\_as'] = 'District of Columbia'** |

* 1. **Opening Other Data Set for Joining GDP Information**

|  |
| --- |
| **# gdp open gdp = pd.read\_csv("SQGDP1\_\_ALL\_AREAS\_2005\_2020.csv")  # get only real GDP description gdp = gdp.loc[gdp['Description'] == 'Real GDP (millions of chained 2012 dollars)']  # Drop unnecessary columns gdp = gdp.drop(columns = ['GeoFIPS', 'Region', 'TableName', 'LineCode', 'IndustryClassification', 'Description', 'Unit'])  # Readjust into time-series table gdp = gdp.melt(id\_vars = ['GeoName'],  var\_name = "YEAR:Q",  value\_name = "RealGDP")  # Change time period into datetime pandas format gdp['YEAR:Q'] = gdp['YEAR:Q'].str.replace(r'(\d+):(Q\d)', r'\1-\2') gdp['startQuarter'] = pd.to\_datetime(gdp['YEAR:Q']) gdp['endQuarter'] = pd.to\_datetime(gdp['startQuarter'] + pd.tseries.offsets.QuarterEnd(0))  # Sort values gdp = gdp.sort\_values(by = ['GeoName', 'YEAR:Q'])  # Get first difference percentage change gdp['Diff'] = gdp.groupby(['GeoName'])['RealGDP'].pct\_change().fillna(0)  # Get previous quarter in 2 new columns gdp['prevQuarterStartDate'] = gdp['startQuarter'] - pd.tseries.offsets.DateOffset(months = 3) gdp['prevQuarterEndDate'] = gdp['endQuarter'] - pd.tseries.offsets.DateOffset(months = 3) gdp['prevQuarterDiff'] = gdp.groupby(['GeoName'])['Diff'].shift(1) gdp['issue\_q'] = pd.PeriodIndex(pd.to\_datetime(gdp['startQuarter']), freq = 'Q') gdp['pqissue\_q'] = pd.PeriodIndex(pd.to\_datetime(gdp['prevQuarterStartDate']), freq = 'Q')  # Check for final and ready to join gdp = gdp.rename(columns = {'GeoName':"length\_as"}) gdp = gdp[['length\_as', 'issue\_q', 'Diff', 'pqissue\_q', 'prevQuarterDiff']] gdp** |

* 1. **Joining with GDP Information**

|  |
| --- |
| **df = pd.merge(df, gdp, on = ['length\_as', 'issue\_q'], how = 'left') df = df.drop(columns = 'pqissue\_q\_x') df = df.rename(columns = {'pqissue\_q\_y':'pqissue\_q'}) df** |

* 1. **Opening Other Data Set for Joining Unemployment Rate Information**

|  |
| --- |
| **bls = pd.read\_excel('ststdsadata.xlsx')  # Renaming the state to it's abbreviation  bls.loc[bls['Unnamed: 1'] == "Alabama", ['Unnamed: 1']] = 'AL' bls.loc[bls['Unnamed: 1'] == "Alaska", ['Unnamed: 1']] = 'AK' bls.loc[bls['Unnamed: 1'] == "Arizona", ['Unnamed: 1']] = 'AZ' bls.loc[bls['Unnamed: 1'] == "Arkansas", ['Unnamed: 1']] = 'AR' bls.loc[bls['Unnamed: 1'] == "California", ['Unnamed: 1']] = 'CA' bls.loc[bls['Unnamed: 1'] == "Colorado", ['Unnamed: 1']] = 'CO' bls.loc[bls['Unnamed: 1'] == "Connecticut", ['Unnamed: 1']] = 'CT' bls.loc[bls['Unnamed: 1'] == "Delaware", ['Unnamed: 1']] = 'DE' bls.loc[bls['Unnamed: 1'] == "District of Columbia", ['Unnamed: 1']] = 'DC' bls.loc[bls['Unnamed: 1'] == "Florida", ['Unnamed: 1']] = 'FL' bls.loc[bls['Unnamed: 1'] == "Georgia", ['Unnamed: 1']] = 'GA' bls.loc[bls['Unnamed: 1'] == "Hawaii", ['Unnamed: 1']] = 'HI' bls.loc[bls['Unnamed: 1'] == "Idaho", ['Unnamed: 1']] = 'ID' bls.loc[bls['Unnamed: 1'] == "Illinois", ['Unnamed: 1']] = 'IL' bls.loc[bls['Unnamed: 1'] == "Indiana", ['Unnamed: 1']] = 'IN' bls.loc[bls['Unnamed: 1'] == "Iowa", ['Unnamed: 1']] = 'IA' bls.loc[bls['Unnamed: 1'] == "Kansas", ['Unnamed: 1']] = 'KS' bls.loc[bls['Unnamed: 1'] == "Kentucky", ['Unnamed: 1']] = 'KY' bls.loc[bls['Unnamed: 1'] == "Louisiana", ['Unnamed: 1']] = 'LA' bls.loc[bls['Unnamed: 1'] == "Maine", ['Unnamed: 1']] = 'ME' bls.loc[bls['Unnamed: 1'] == "Maryland", ['Unnamed: 1']] = 'MD' bls.loc[bls['Unnamed: 1'] == "Massachusetts", ['Unnamed: 1']] = 'MA' bls.loc[bls['Unnamed: 1'] == "Michigan", ['Unnamed: 1']] = 'MI' bls.loc[bls['Unnamed: 1'] == "Minnesota", ['Unnamed: 1']] = 'MN' bls.loc[bls['Unnamed: 1'] == "Mississippi", ['Unnamed: 1']] = 'MS' bls.loc[bls['Unnamed: 1'] == "Missouri", ['Unnamed: 1']] = 'MO' bls.loc[bls['Unnamed: 1'] == "Montana", ['Unnamed: 1']] = 'MT' bls.loc[bls['Unnamed: 1'] == "Nebraska", ['Unnamed: 1']] = 'NE' bls.loc[bls['Unnamed: 1'] == "Nevada", ['Unnamed: 1']] = 'NV' bls.loc[bls['Unnamed: 1'] == "New Hampshire", ['Unnamed: 1']] = 'NH' bls.loc[bls['Unnamed: 1'] == "New Jersey", ['Unnamed: 1']] = 'NJ' bls.loc[bls['Unnamed: 1'] == "New Mexico", ['Unnamed: 1']] = 'NM' bls.loc[bls['Unnamed: 1'] == "New York", ['Unnamed: 1']] = 'NY' bls.loc[bls['Unnamed: 1'] == "North Carolina", ['Unnamed: 1']] = 'NC' bls.loc[bls['Unnamed: 1'] == "North Dakota", ['Unnamed: 1']] = 'ND' bls.loc[bls['Unnamed: 1'] == "Ohio", ['Unnamed: 1']] = 'OH' bls.loc[bls['Unnamed: 1'] == "Oklahoma", ['Unnamed: 1']] = 'OK' bls.loc[bls['Unnamed: 1'] == "Oregon", ['Unnamed: 1']] = 'OR' bls.loc[bls['Unnamed: 1'] == "Pennsylvania", ['Unnamed: 1']] = 'PA' bls.loc[bls['Unnamed: 1'] == "Rhode Island", ['Unnamed: 1']] = 'RI' bls.loc[bls['Unnamed: 1'] == "South Carolina", ['Unnamed: 1']] = 'SC' bls.loc[bls['Unnamed: 1'] == "South Dakota", ['Unnamed: 1']] = 'SD' bls.loc[bls['Unnamed: 1'] == "Tennessee", ['Unnamed: 1']] = 'TN' bls.loc[bls['Unnamed: 1'] == "Texas", ['Unnamed: 1']] = 'TX' bls.loc[bls['Unnamed: 1'] == "Utah", ['Unnamed: 1']] = 'UT' bls.loc[bls['Unnamed: 1'] == "Vermont", ['Unnamed: 1']] = 'VT' bls.loc[bls['Unnamed: 1'] == "Virginia", ['Unnamed: 1']] = 'VA' bls.loc[bls['Unnamed: 1'] == "Washington", ['Unnamed: 1']] = 'WA' bls.loc[bls['Unnamed: 1'] == "West Virginia", ['Unnamed: 1']] = 'WV' bls.loc[bls['Unnamed: 1'] == "Wisconsin", ['Unnamed: 1']] = 'WI' bls.loc[bls['Unnamed: 1'] == "Wyoming", ['Unnamed: 1']] = 'WY'  # Creating Month column  df['Month'] = pd.DatetimeIndex(df['issue\_d']).month  # Converting to leading 0, because that's the format used in BLS dataset  # for instance, January is 01 instead of 1 df["Month"] = df.Month.map("{:02}".format)  # Creating Year column in df  df['Year'] = pd.DatetimeIndex(df['issue\_d']).year  # renaming columns in bls dataset  bls = bls.rename(columns = {"Unnamed: 1": "addr\_state"}) bls = bls.rename(columns = {"Unnamed: 2": "Year"}) bls = bls.rename(columns = {"Unnamed: 3": "Month"}) bls = bls.rename(columns = {"Unnamed: 10": "UR"}) #unemployment rate  bls = pd.DataFrame(bls, columns=['addr\_state', 'Year', 'Month', 'UR']) bls = bls.dropna() #CLEANED bls  bls.head()** |

|  |
| --- |
| **# Adding columns for better joining**  **bls['YearMonth'] = bls['Year'].astype(str)+'-'+bls['Month'].astype(str) bls['YearQuarter'] = pd.PeriodIndex(pd.to\_datetime(bls['YearMonth']), freq = 'Q')**  **# Sorting values to calculate difference between one period to another bls = bls.sort\_values(by = ['addr\_state', 'YearMonth']) bls['URDiff'] = bls.groupby(['addr\_state'])['UR'].pct\_change().fillna(0)**  **# Sorting values to calculate difference between previous period and two periods ago bls['URprevMonthDiff'] = bls.groupby(['addr\_state'])['URDiff'].shift(1) bls** |

* 1. **Joining with Unemployment Rate Information**

|  |
| --- |
| **bls['Year']=bls['Year'].astype(int) #has to convert to allow merging; datatypes have to be the same.  df = pd.merge(df, bls, on = ['Month', 'Year', 'addr\_state'], how = 'left')** |

* 1. **Transfer to CSV**

|  |
| --- |
| **df.to\_csv('df.csv')** |

1. **Data Pre-processing**
   1. **Library List:**

|  |
| --- |
| **import numpy as np import pandas as pd** |

* 1. **Opening Folder:**

|  |
| --- |
| **# Opening the data frame after joining (resultant of previous section) df = pd.read\_csv("df.csv", low\_memory = False)  # Removing that first column: ("Unnamed: 0") df = df.drop(columns = "Unnamed: 0")** |

* 1. **Dealing with Target Values:**

|  |
| --- |
| **# We want to train our model to evaluate charged-off (failing loans) or fully paid (successful loans) # Thus, we want to separate all other loan status categories from these two categories  print(df['loan\_status'].describe(), "\n") print("Before removing other categories") print(df['loan\_status'].value\_counts(dropna = False)) df = df.loc[df['loan\_status'].isin(['Fully Paid', 'Charged Off'])].copy() print("After removing other categories", "\n") print(df['loan\_status'].value\_counts(dropna = False))** |

|  |
| --- |
| **Results:**  **count 2260668 unique 9 top Fully Paid freq 1041952 Name: loan\_status, dtype: object   Before removing other categories Fully Paid 1041952 Current 919695 Charged Off 261655 Late (31-120 days) 21897 In Grace Period 8952 Late (16-30 days) 3737 Does not meet the credit policy. Status:Fully Paid 1988 Does not meet the credit policy. Status:Charged Off 761 Default 31 Name: loan\_status, dtype: int64 After removing other categories   Fully Paid 1041952 Charged Off 261655 Name: loan\_status, dtype: int64** |

* 1. **Filtering Data Sets:**

|  |
| --- |
| **# What is available to investor keep\_list1 = ['loan\_amnt', 'issue\_d', 'loan\_status', 'funded\_amnt',**  **'funded\_amnt\_inv', 'verification\_status', 'installment', 'grade',**  **'sub\_grade', 'home\_ownership', 'emp\_length', 'emp\_title',**  **'addr\_state', 'zip\_code', 'annual\_inc', 'dti',   'ageOfCredit', 'earliest\_cr\_line', 'earliest\_cr\_line\_year',**  **'open\_acc', 'total\_acc', 'revol\_bal', 'revol\_util',**  **'inq\_last\_6mths', 'acc\_now\_delinq', 'delinq\_amnt', 'delinq\_2yrs',**  **'pub\_rec', 'collections\_12\_mths\_ex\_med', 'int\_rate',**  **'tot\_coll\_amt', 'purpose', 'term', 'initial\_list\_status',**  **'application\_type', 'length\_as', 'pqissue\_d', 'issue\_q',**  **'log\_annual\_inc', 'log\_revol\_bal', 'Diff', 'pqissue\_q',**  **'prevQuarterDiff', 'Month', 'Year', 'UR', 'YearMonth',**  **'YearQuarter', 'URDiff', 'URprevMonthDiff']** |

|  |
| --- |
| **# Adjusting after correlation analysis and Cramer's V keep\_list3 = ['Month', 'loan\_amnt', 'issue\_d', 'loan\_status',**  **'verification\_status', 'grade', 'home\_ownership', 'emp\_length',   'addr\_state', 'annual\_inc', 'log\_annual\_inc', 'dti',**  **'ageOfCredit', 'open\_acc', 'total\_acc', 'revol\_bal',   'revol\_util', 'acc\_now\_delinq', 'delinq\_amnt', 'delinq\_2yrs',   'pub\_rec', 'collections\_12\_mths\_ex\_med',  'int\_rate', 'inq\_last\_6mths', 'tot\_coll\_amt', 'purpose', 'term',**  **'initial\_list\_status', 'application\_type',  'log\_revol\_bal', 'Diff', 'prevQuarterDiff', 'UR', 'URDiff',**  **'URprevMonthDiff']** |

|  |
| --- |
| **# df1 will contain whatever in keep\_list1 - before correlation analysis**  **# df2 will contain whatever in keep\_list3 - after correlation analysis # We discard keep\_list2 BECAUSE it's not available to potential investor in the GUI df1 = df[keep\_list1] df2 = df[keep\_list3]** |

|  |
| --- |
| **# Remove NA df1 = df1.dropna() df2 = df2.dropna()  print("DF1 shape is: ", df1.shape) print("DF2 shape is: ", df2.shape)**  **Results:**  **DF1 shape is: (1154570, 51)**  **DF2 shape is: (1160023, 36)** |

* 1. **Creating Dummy Variables:**

|  |
| --- |
| **df2\_with\_dummies = df2.copy() # Creating a category for charged off or not df2\_with\_dummies['charged\_off'] = (df2\_with\_dummies['loan\_status'] == 'Charged Off').apply(np.uint8)  # Change grade into different kinds of dummies df2\_with\_dummies['grade'] = pd.factorize(df2\_with\_dummies['grade'])[0] + 1  # Get dummies for certain variables only df2\_with\_dummies = pd.get\_dummies(df2\_with\_dummies,columns=['verification\_status', 'home\_ownership', 'addr\_state', 'purpose', 'term', 'initial\_list\_status',  'application\_type'],   drop\_first = True,  dummy\_na = True)  # Getting observations between 2015 and 2018 df2\_with\_dummies = df2\_with\_dummies[(df2\_with\_dummies['issue\_d'] >= '2015-01-01') & (df2\_with\_dummies['issue\_d'] <= '2018-12-31')] # Dropping issue date because it's unnecessary df2\_with\_dummies.drop(columns = ['issue\_d', 'loan\_status'], axis = 1, inplace = True)  # Check results df2\_with\_dummies.shape**  **Results: (797003, 108)**  **# Create new CSV df2\_with\_dummies.to\_csv('df2\_with\_dummies.csv')** |

1. **Principal Component Analysis**
   1. **Library List:**

|  |
| --- |
| **import pandas as pd, numpy as np from sklearn.preprocessing import StandardScaler from sklearn.decomposition import PCA import matplotlib.pyplot as plt from sklearn import metrics, datasets** |

* 1. **Opening Folder:**

|  |
| --- |
| **# Open data df2\_with\_dummies = pd.read\_csv("df2\_with\_dummies.csv", low\_memory = False)  # Drop unnecessary columns # annual\_inc and revol\_bal are replaced with its log version # month isn't used df2\_with\_dummies = df2\_with\_dummies.drop(columns = ["Unnamed: 0", 'annual\_inc', 'revol\_bal', 'Month'])** |

* 1. **Selecting Numerical Variables to Scale and Run PCA on the Data Set**

|  |
| --- |
| **# Separating out the features Scaled = ['loan\_amnt', 'emp\_length', 'log\_annual\_inc', 'dti', 'ageOfCredit',  'open\_acc', 'total\_acc', 'revol\_util', 'inq\_last\_6mths',**  **'acc\_now\_delinq', 'delinq\_amnt', 'delinq\_2yrs', 'pub\_rec',  'collections\_12\_mths\_ex\_med', 'int\_rate', 'tot\_coll\_amt',  'log\_revol\_bal', 'Diff', 'prevQuarterDiff', 'UR', 'URDiff',**  **'URprevMonthDiff']  df2\_with\_dummies[Scaled] = StandardScaler().fit\_transform(df2\_with\_dummies[Scaled]) df2\_with\_dummies  # Create a copy for PCA df2\_with\_dummies\_PCA = df2\_with\_dummies.copy()  # Isolate target variable y = df2\_with\_dummies\_PCA['charged\_off'] x = df2\_with\_dummies\_PCA.drop(columns = ['charged\_off'])  # Apply PCA pca = PCA() x = pca.fit\_transform(x)  # Check the x aftermath - a bunch of eigenvector x**  **Results:**  **array([[ 2.38607385e-01, 1.91349437e+00, -4.91986783e-01, ...,**  **7.13808389e-17, 5.24093967e-17, 6.48990869e-17],**  **[ 1.97709847e+00, 1.47595650e+00, -1.39019285e+00, ...,**  **8.22984850e-17, 4.40812323e-17, -8.48432245e-17],**  **[ 6.24092665e-01, -1.17856352e-01, 1.29817408e-03, ...,**  **1.20739674e-16, 1.75298170e-17, -6.95053260e-18],**  **...,**  **[ 4.51512474e-01, -3.97287780e+00, -5.26907966e-01, ...,**  **1.74157859e-20, -6.38728923e-19, -8.47875286e-19],**  **[ 2.48982092e+00, 1.00538332e+00, 3.23472853e+00, ...,**  **5.23049956e-19, 7.27145091e-19, -3.77664094e-18],**  **[-1.11076526e+00, -2.13479984e+00, 1.11931589e+00, ...,**  **-9.53976355e-19, 7.96925505e-21, -1.18863174e-19]])  # Print to determine elbow ev = pca.explained\_variance\_ratio\_ print(ev) plt.plot(ev)**  **Results:**  **[9.90339454e-02 6.91485891e-02 6.14576148e-02 5.80398063e-02**  **5.40597334e-02 4.67660741e-02 4.48598027e-02 4.12486354e-02**  **3.87963332e-02 3.85932536e-02 3.72496217e-02 3.72191871e-02**  **3.47991364e-02 3.39056468e-02 3.29603783e-02 3.07326477e-02**  **3.00372934e-02 2.64802790e-02 2.47814927e-02 2.08810833e-02**  **1.56059418e-02 1.38785549e-02 1.26357879e-02 1.22370961e-02**  **1.21818087e-02 8.82640173e-03 7.87367525e-03 5.45407813e-03**  **4.35916593e-03 3.81355260e-03 3.62460172e-03 3.21670781e-03**  **2.85399158e-03 2.65433411e-03 2.27323853e-03 2.14739637e-03**  **1.36131097e-03 1.25721568e-03 1.22219175e-03 1.15809397e-03**  **1.09531434e-03 1.05358599e-03 1.01323165e-03 9.87128348e-04**  **9.34988387e-04 8.76701002e-04 8.62801158e-04 8.45439818e-04**  **8.13444950e-04 7.61105530e-04 6.73476767e-04 6.45966385e-04**  **6.06980810e-04 5.84143987e-04 5.71175605e-04 5.42564454e-04**  **5.28803450e-04 4.90472443e-04 4.43567816e-04 4.38800318e-04**  **4.30083919e-04 4.01569351e-04 3.94362746e-04 3.56186976e-04**  **3.33836269e-04 3.11001897e-04 2.98708999e-04 2.74416863e-04**  **2.69438494e-04 2.65821103e-04 2.23903503e-04 2.16949033e-04**  **2.03360989e-04 1.82820125e-04 1.69727627e-04 1.62235418e-04**  **1.39120244e-04 1.10470721e-04 1.04642391e-04 1.00437652e-04**  **9.01869000e-05 8.11648660e-05 7.79638858e-05 7.62772300e-05**  **7.26852621e-05 6.52045515e-05 6.39344761e-05 4.09292105e-05**  **2.12362318e-05 3.87855433e-06 1.63174898e-06 2.80292658e-07**  **9.27253037e-08 4.67252386e-08 1.95310019e-32 5.11300709e-34**  **5.11300709e-34 5.11300709e-34 5.11300709e-34 5.11300709e-34**  **5.11300709e-34 5.11300709e-34 5.11300709e-34]**  **[<matplotlib.lines.Line2D at 0x19736598d08>]**    **# Print to cumulative sum plt.plot(ev.cumsum())**  **Results:** |

|  |
| --- |
| **# Isolate target variable y = df2\_with\_dummies\_PCA['charged\_off'] x = df2\_with\_dummies\_PCA.drop(columns = ['charged\_off'])  # Apply PCA 25 pca = PCA(n\_components=25) x = pca.fit\_transform(x)  # Print to determine elbow ev = pca.explained\_variance\_ratio\_ print(ev) plt.plot(ev)**  **# Getting cumulative sum**  **plt.plot(ev.cumsum())**  **Results:**  **[0.09903395 0.06914859 0.06145761 0.05803981 0.05405973 0.04676607**  **0.0448598 0.04124864 0.03879633 0.03859325 0.03724962 0.03721919**  **0.03479914 0.03390565 0.03296038 0.03073265 0.03003729 0.02648028**  **0.02478149 0.02088108 0.01560594 0.01387855 0.01263579 0.0122371**  **0.01218181]**  **[<matplotlib.lines.Line2D at 0x1973760bbc8>]**    **Results:**  **[<matplotlib.lines.Line2D at 0x1973766bb88>]** |

|  |
| --- |
| **pd.DataFrame(x).to\_csv("x\_with\_dummies\_PCA25.csv") pd.DataFrame(y).to\_csv("y\_with\_dummies\_PCA25.csv")** |

1. **Machine Learning Algorithms**
   1. **Library List:**

|  |
| --- |
| **import pandas as pd import numpy as np from sklearn.model\_selection import train\_test\_split from sklearn.naive\_bayes import GaussianNB**  **from sklearn.tree import DecisionTreeClassifier**  **from sklearn.ensemble import RandomForestClassifier**  **from sklearn.preprocessing import scale**  **from sklearn.neural\_network import MLPClassifier from sklearn import metrics, datasets,tree import matplotlib.pyplot as plt from sklearn.metrics import accuracy\_score, recall\_score, precision\_score, f1\_score from sklearn.metrics import roc\_auc\_score**  **from sklearn.metrics import mean\_squared\_error**  **import math** |

* 1. **Opening Files and Separating Target and Features**

|  |
| --- |
| **df = pd.read\_csv("df2\_with\_dummies.csv") df = df.drop(columns = ['Unnamed: 0','log\_annual\_inc', 'revol\_bal', 'UR'])** |

|  |
| --- |
| **# defining target and feature variable X = df.drop(columns = 'charged\_off') X = X.replace([np.inf, -np.inf], np.nan).dropna(axis=1) y = df['charged\_off']  # splitting dataset  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size =.3,random\_state=1234, stratify=y)** |

* 1. **Naive Bayes**

|  |
| --- |
| **# Create a Bayesian Classifier instance for classification gnb = GaussianNB()   # Build a Bayesian Classification Model and predict the type using the test data. gnb.fit(X\_train, y\_train) y\_pred\_nb = gnb.predict(X\_test) cm = metrics.confusion\_matrix(y\_test,y\_pred\_nb) print(metrics.classification\_report(y\_test,y\_pred\_nb))  print('PERFORMANCE SCORE: NB ENTIRE DATASET') print ('Accuracy:', accuracy\_score(y\_test, y\_pred\_nb)) print ('F1 score:', f1\_score(y\_test, y\_pred\_nb)) print ('Recall:', recall\_score(y\_test, y\_pred\_nb)) print ('Precision:', precision\_score(y\_test, y\_pred\_nb)) print ('AUC:', roc\_auc\_score(y\_test, y\_pred\_nb))**  **mse\_nb = mean\_squared\_error(y\_test, y\_pred\_nb)**  **rmse\_nb = math.sqrt(mse\_nb)**  **print(rmse\_nb)**  **Results:**  **precision recall f1-score support**  **0 0.89 0.49 0.63 188818**  **1 0.29 0.77 0.42 50283**  **accuracy 0.55 239101**  **macro avg 0.59 0.63 0.52 239101**  **weighted avg 0.76 0.55 0.58 239101**  **PERFORMANCE SCORE: NB ENTIRE DATASET**  **Accuracy: 0.5467898503143023**  **F1 score: 0.4156092089155418**  **Recall: 0.7663226140047332**  **Precision: 0.28512127624938954**  **AUC: 0.6273249990232544**  **0.6732088455194998** |

* 1. **Decision Trees**

|  |
| --- |
| **from sklearn.tree import DecisionTreeClassifier # Create a model (object) for classification dtm = DecisionTreeClassifier()   # Build a decision tree dtm.fit(X\_train, y\_train) y\_pred\_dt = dtm.predict(X\_test)   # Calculate accuracy accuracy = dtm.score(X\_test, y\_test)**  **print (metrics.classification\_report(y\_test,y\_pred\_dt)) print ('Accuracy:', accuracy\_score(y\_test, y\_pred\_dt)) print ('F1 score:', f1\_score(y\_test, y\_pred\_dt)) print ('Recall:', recall\_score(y\_test, y\_pred\_dt)) print ('Precision:', precision\_score(y\_test, y\_pred\_dt)) print ('AUC:', roc\_auc\_score(y\_test, y\_pred\_dt)) mse\_dt = mean\_squared\_error(y\_test, y\_pred\_dt) rmse\_dt = math.sqrt(mse\_dt) print(rmse\_dt)**  **Results:**  **precision recall f1-score support**  **0 0.81 0.80 0.81 188818**  **1 0.29 0.30 0.30 50283**  **accuracy 0.70 239101**  **macro avg 0.55 0.55 0.55 239101**  **weighted avg 0.70 0.70 0.70 239101**  **Accuracy: 0.6953379534171752**  **F1 score: 0.2955642158806293**  **Recall: 0.3039198138535887**  **Precision: 0.287655761773896**  **AUC: 0.5517470034959774**  **0.5519619974081774** |

* 1. **Random Forest**

|  |
| --- |
| **# Create a model (object) for classification rfcm = RandomForestClassifier()  # Build a random forest classification model rfcm.fit(X\_train, y\_train) y\_pred\_rf = rfcm.predict(X\_test)  # Calculate accuracy #accuracy = rfcm.score(X\_test, y\_test) #print('Accuracy: {0:.2f}'.format(accuracy))  # Build a confusion matrix and show the Classification Report cm1 = metrics.confusion\_matrix(y\_test,y\_pred\_rf) print('\nConfusion Matrix','\n',cm1) print('\nClassification Report','\n',metrics.classification\_report(y\_test,y\_pred\_rf))  print ('Accuracy:', accuracy\_score(y\_test, y\_pred\_rf)) print ('F1 score:', f1\_score(y\_test, y\_pred\_rf)) print ('Recall:', recall\_score(y\_test, y\_pred\_rf)) print ('Precision:', precision\_score(y\_test, y\_pred\_rf)) print ('AUC:', roc\_auc\_score(y\_test, y\_pred\_rf)) mse\_rf = mean\_squared\_error(y\_test, y\_pred\_rf) rmse\_rf = math.sqrt(mse\_rf) print(rmse\_rf)**  **Results:**  **Confusion Matrix**  **[[185400 3418]**  **[ 45857 4426]]**  **Classification Report**  **precision recall f1-score support**  **0 0.80 0.98 0.88 188818**  **1 0.56 0.09 0.15 50283**  **accuracy 0.79 239101**  **macro avg 0.68 0.53 0.52 239101**  **weighted avg 0.75 0.79 0.73 239101**  **Accuracy: 0.7939155419676204**  **F1 score: 0.15228723312746228**  **Recall: 0.08802179663106816**  **Precision: 0.5642529321774605**  **AUC: 0.5349598544531905**  **0.4539652608211113** |

* 1. **Neural Network**

|  |
| --- |
| **# Normalize the data Xn = scale(X)   # Set the 'stratify' option 'y' to sample  Xn\_train, Xn\_test = train\_test\_split(Xn, test\_size =.3,random\_state=1234, stratify=y) y\_train, y\_test = train\_test\_split(y, test\_size=.3, random\_state=1234, stratify=y) nnm = MLPClassifier(hidden\_layer\_sizes=(20,), max\_iter=1000,activation='logistic')  # Make predictions nnm.fit(Xn\_train, y\_train) y\_pred\_NN = nnm.predict(Xn\_test)  print('\nClassification Report','\n',metrics.classification\_report(y\_test,y\_pred\_NN)) print ('Accuracy:', accuracy\_score(y\_test, y\_pred\_NN)) print ('F1 score:', f1\_score(y\_test, y\_pred\_NN)) print ('Recall:', recall\_score(y\_test, y\_pred\_NN)) print ('Precision:', precision\_score(y\_test, y\_pred\_NN)) print ('AUC:', roc\_auc\_score(y\_test, y\_pred\_NN)) mse\_nn = mean\_squared\_error(y\_test, y\_pred\_NN) rmse\_nn = math.sqrt(mse\_nn) print(rmse\_nn)**  **Results:**  **Classification Report**  **precision recall f1-score support**  **0 0.81 0.98 0.88 188818**  **1 0.57 0.12 0.19 50283**  **accuracy 0.80 239101**  **macro avg 0.69 0.55 0.54 239101**  **weighted avg 0.76 0.80 0.74 239101**  **Accuracy: 0.7956846688219623**  **F1 score: 0.19148654463605969**  **Recall: 0.11504882365809518**  **Precision: 0.570569089653812**  **AUC: 0.5459947907124167**  **0.4520125343151866** |

* 1. **Naive Bayesian Using PCA25 Data Set**

|  |
| --- |
| **# defining target and feature variable X = pd.read\_csv("x\_with\_dummies\_PCA25.csv") X = X.drop(columns = 'Unnamed: 0') y = pd.read\_csv("y\_with\_dummies\_PCA25.csv") y = y.drop(columns = 'Unnamed: 0')  # splitting dataset  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size =.3,random\_state=1234, stratify=y)  # Create a Bayesian Classifier instance for classification gnb\_PCA = GaussianNB()   # Build a Bayesian Classification Model and predict the type using the test data. gnb\_PCA.fit(X\_train, y\_train) y\_pred\_nb\_pca = gnb\_PCA.predict(X\_test)  cm\_nb\_pca = metrics.confusion\_matrix(y\_test,y\_pred\_nb\_pca) print(metrics.classification\_report(y\_test,y\_pred\_nb\_pca))  print('PERFORMANCE SCORE: NB ENTIRE DATASET') print ('Accuracy:', accuracy\_score(y\_test, y\_pred\_nb\_pca)) print ('F1 score:', f1\_score(y\_test, y\_pred\_nb\_pca)) print ('Recall:', recall\_score(y\_test, y\_pred\_nb\_pca)) print ('Precision:', precision\_score(y\_test, y\_pred\_nb\_pca)) print ('AUC:', roc\_auc\_score(y\_test, y\_pred\_nb\_pca)) mse\_nb\_pca = mean\_squared\_error(y\_test, y\_pred\_nb\_pca) rmse\_nb\_pca = math.sqrt(mse\_nb\_pca) print('RMSE:', rmse\_nn)**  **Results:**  **precision recall f1-score support**  **0 0.82 0.90 0.86 188818**  **1 0.41 0.25 0.31 50283**  **accuracy 0.77 239101**  **macro avg 0.61 0.58 0.59 239101**  **weighted avg 0.73 0.77 0.74 239101**  **PERFORMANCE SCORE: NB ENTIRE DATASET**  **Accuracy: 0.766696918875287**  **F1 score: 0.3113804979816559**  **Recall: 0.2508203567806217**  **Precision: 0.4104934253352428**  **AUC: 0.5774486492988048**  **RMSE: 0.4520125343151866** |

* 1. **Decision Tree Using PCA25 Data Set**

|  |
| --- |
| **dtm\_pca = DecisionTreeClassifier()   # Build a decision tree dtm\_pca.fit(X\_train, y\_train) y\_pred\_dt\_pca = dtm\_pca.predict(X\_test)  cm\_dt\_pca = metrics.confusion\_matrix(y\_test,y\_pred\_dt\_pca) print(metrics.classification\_report(y\_test,y\_pred\_dt\_pca))  print ('Accuracy:', accuracy\_score(y\_test, y\_pred\_dt\_pca)) print ('F1 score:', f1\_score(y\_test, y\_pred\_dt\_pca)) print ('Recall:', recall\_score(y\_test, y\_pred\_dt\_pca)) print ('Precision:', precision\_score(y\_test, y\_pred\_dt\_pca)) print ('AUC:', roc\_auc\_score(y\_test, y\_pred\_dt\_pca)) mse\_dt\_pca = mean\_squared\_error(y\_test, y\_pred\_dt\_pca) rmse\_dt\_pca = math.sqrt(mse\_dt\_pca) print(rmse\_dt\_pca)**  **Results:**  **precision recall f1-score support**  **0 0.81 0.79 0.80 188818**  **1 0.28 0.30 0.29 50283**  **accuracy 0.69 239101**  **macro avg 0.54 0.55 0.55 239101**  **weighted avg 0.70 0.69 0.69 239101**  **Accuracy: 0.6891020949305942**  **F1 score: 0.29011803353833227**  **Recall: 0.30209016963983853**  **Precision: 0.2790586592691933**  **AUC: 0.5471275557707821**  **0.5575821957966429** |

* 1. **Random Forest Using PCA25 Data Set**

|  |
| --- |
| **from sklearn.ensemble import RandomForestClassifier # Create a model (object) for classification rfcm\_pca = RandomForestClassifier()  # Build a random forest classification model rfcm\_pca.fit(X\_train, y\_train) y\_pred\_rf\_pca = rfcm\_pca.predict(X\_test)  print(metrics.classification\_report(y\_test,y\_pred\_rf\_pca)) print ('Accuracy:', accuracy\_score(y\_test, y\_pred\_rf\_pca)) print ('F1 score:', f1\_score(y\_test, y\_pred\_rf\_pca)) print ('Recall:', recall\_score(y\_test, y\_pred\_rf\_pca)) print ('Precision:', precision\_score(y\_test, y\_pred\_rf\_pca)) print ('AUC:', roc\_auc\_score(y\_test, y\_pred\_rf\_pca)) mse\_rf\_pca = mean\_squared\_error(y\_test, y\_pred\_rf\_pca) rmse\_rf\_pca = math.sqrt(mse\_rf\_pca) print(rmse\_rf\_pca)**  **Results:  precision recall f1-score support   0 0.80 0.98 0.88 188818  1 0.54 0.07 0.13 50283   accuracy 0.79 239101  macro avg 0.67 0.53 0.51 239101 weighted avg 0.74 0.79 0.72 239101  Accuracy: 0.7919289337978511 F1 score: 0.131186476197129 Recall: 0.07469721377006146 Precision: 0.5381859865310217 AUC: 0.5288139332839968 0.45614807486401704** |

* 1. **Neural Network Using PCA25 Data Set**

|  |
| --- |
| **# Normalize the data Xn = scale(X)   # Set the 'stratify' option 'y' to sample  Xn\_train, Xn\_test = train\_test\_split(Xn, test\_size =.3,random\_state=1234, stratify=y) y\_train, y\_test = train\_test\_split(y, test\_size=.3, random\_state=1234, stratify=y)  nnm\_PCA = MLPClassifier(hidden\_layer\_sizes=(20,), max\_iter=1000,activation='logistic')  # Make predictions nnm\_PCA.fit(Xn\_train, y\_train) y\_pred\_NN\_PCA = nnm\_PCA.predict(Xn\_test)  print('PERFORMANCE SCORE NN- ENTIRE DATASET') print(metrics.classification\_report(y\_test,y\_pred\_rf\_pca), "\n") print ('Accuracy:', accuracy\_score(y\_test, y\_pred\_NN\_PCA)) print ('F1 score:', f1\_score(y\_test, y\_pred\_NN\_PCA)) print ('Recall:', recall\_score(y\_test, y\_pred\_NN\_PCA)) print ('Precision:', precision\_score(y\_test, y\_pred\_NN\_PCA)) print ('AUC:', roc\_auc\_score(y\_test, y\_pred\_NN\_PCA)) mse\_nn\_pca = mean\_squared\_error(y\_test, y\_pred\_NN\_PCA) rmse\_nn\_pca = math.sqrt(mse\_nn\_pca) print(rmse\_nn\_pca)**  **Results:**  **PERFORMANCE SCORE NN- ENTIRE DATASET**  **precision recall f1-score support**  **0 0.80 0.98 0.88 188818**  **1 0.54 0.07 0.13 50283**  **accuracy 0.79 239101**  **macro avg 0.67 0.53 0.51 239101**  **weighted avg 0.74 0.79 0.72 239101**    **Accuracy: 0.7941037469521248**  **F1 score: 0.17310534802472452**  **Recall: 0.10247996340711572**  **Precision: 0.5569004647141468**  **AUC: 0.5403829659529409**  **0.4537579233995536** |

1. **Machine Learning Algorithms Using Oversampling and Undersampling**
   1. **Library List and Opening File**

|  |
| --- |
| **import pandas as pd, numpy as np from sklearn.decomposition import PCA from sklearn.preprocessing import scale from sklearn.model\_selection import train\_test\_split from sklearn.tree import DecisionTreeClassifier from sklearn.ensemble import RandomForestClassifier**  **from imblearn.under\_sampling import RandomUnderSampler**  **from sklearn.neural\_network import MLPClassifier from sklearn import metrics from sklearn.datasets import make\_classification from matplotlib import pyplot as plt import imblearn from sklearn.metrics import recall\_score, precision\_score, f1\_score from sklearn.metrics import roc\_auc\_score**  **from imblearn.under\_sampling import RandomUnderSampler** |

|  |
| --- |
| **df = pd.read\_csv("df2\_with\_dummies.csv") df = df.drop(columns = ['Unnamed: 0','log\_annual\_inc', 'revol\_bal', 'UR'])  # defining target and feature variable X = df.drop(columns = 'charged\_off') X = X.replace([np.inf, -np.inf], np.nan).dropna(axis=1) y = df['charged\_off']  # Separate train and test x\_train, x\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size =.3,random\_state=1234, stratify=y) print("The shape of train data set using df2\_with\_dummies is: x = {} and y = {}".format(x\_train.shape, y\_train.shape)) print("The shape of test data set using df2\_with\_dummies is: x = {} and y = {}". format(x\_test.shape, y\_test.shape))**  **Results:**  **The shape of train data set using df2\_with\_dummies is: x = (557902, 103) and y = (557902,)**  **The shape of test data set using df2\_with\_dummies is: x = (239101, 103) and y = (239101,)** |

|  |
| --- |
| **# define undersample strategy rus = RandomUnderSampler(sampling\_strategy='majority') x\_train\_rus, y\_train\_rus = rus.fit\_sample(x\_train, y\_train.ravel())  #define smote strategy smote = SMOTE(sampling\_strategy='minority') x\_train\_smote, y\_train\_smote = smote.fit\_sample(x\_train, y\_train.ravel())  print("Before RUS, counts of label '1': {}".format(sum(y\_train==1))) print("Before RUS, counts of label '0': {} \n".format(sum(y\_train==0)))  print("The shape of train data set using df2\_with\_dummies is: x = {} and y = {}".format(x\_train\_res.shape, y\_train\_res.shape)) print("The shape of test data set using df2\_with\_dummies is: x = {} and y = {} \n". format(x\_test.shape, y\_test.shape))  print("After RUS, counts of label '1': {}".format(sum(y\_train\_res==1))) print("After RUS, counts of label '0': {}".format(sum(y\_train\_res==0)))**  **Results:**  **Before RUS, counts of label '1': 117326**  **Before RUS, counts of label '0': 440576**  **The shape of train data set using df2\_with\_dummies is: x = (234652, 103) and y = (234652,)**  **The shape of test data set using df2\_with\_dummies is: x = (239101, 103) and y = (239101,)**  **After RUS, counts of label '1': 117326**  **After RUS, counts of label '0': 117326** |

* 1. **Naive Bayesian Classifier Using df2\_with\_dummies Data Set**

|  |
| --- |
| **# Create a model (object) for classification gnb\_rus = GaussianNB() gnb\_smote = GaussianNB()  # Build a random forest classification model gnb\_rus.fit(x\_train\_rus, y\_train\_rus) gnb\_smote.fit(x\_train\_smote, y\_train\_smote)  y\_pred\_gnb\_rus = gnb\_rus.predict(x\_test) y\_pred\_gnb\_smote = gnb\_smote.predict(x\_test)  # Build a confusion matrix and show the Classification Report cm\_nb\_rus = metrics.confusion\_matrix(y\_test,y\_pred\_gnb\_rus) print('\nConfusion Matrix Naive Bayes - RUS','\n',cm\_nb\_rus) print('\nClassification Report','\n',metrics.classification\_report(y\_test,y\_pred\_gnb\_rus)) print("---------------------------------------------------------------------------------")   cm\_nb\_smote = metrics.confusion\_matrix(y\_test,y\_pred\_gnb\_smote) print('\nConfusion Matrix Naive Bayes - SMOTE','\n',cm\_nb\_smote) print('\nClassification Report Naive Bayes - SMOTE','\n',metrics.classification\_report(y\_test,y\_pred\_gnb\_smote)) print("---------------------------------------------------------------------------------")**  **Results:**  **Confusion Matrix Naive Bayes - RUS**  **[[99714 89104]**  **[12782 37501]]**  **Classification Report**  **precision recall f1-score support**  **0 0.89 0.53 0.66 188818**  **1 0.30 0.75 0.42 50283**  **accuracy 0.57 239101**  **macro avg 0.59 0.64 0.54 239101**  **weighted avg 0.76 0.57 0.61 239101**  **---------------------------------------------------------------------------------**  **Confusion Matrix Naive Bayes - SMOTE**  **[[ 14925 173893]**  **[ 2174 48109]]**  **Classification Report Naive Bayes - SMOTE**  **precision recall f1-score support**  **0 0.87 0.08 0.14 188818**  **1 0.22 0.96 0.35 50283**  **accuracy 0.26 239101**  **macro avg 0.54 0.52 0.25 239101**  **weighted avg 0.73 0.26 0.19 239101**  **---------------------------------------------------------------------------------** |

|  |
| --- |
| **print("Performance score for NB - RUS ") print ('Accuracy:', accuracy\_score(y\_test, y\_pred\_gnb\_rus)) print ('F1 score:', f1\_score(y\_test, y\_pred\_gnb\_rus)) print ('Recall:', recall\_score(y\_test, y\_pred\_gnb\_rus)) print ('Precision:', precision\_score(y\_test, y\_pred\_gnb\_rus)) print ('AUC:', roc\_auc\_score(y\_test, y\_pred\_gnb\_rus))  print("---------------------------------------------------------------------------------------")  print("Performance score for NB - SMOTE ") print ('Accuracy:', accuracy\_score(y\_test, y\_pred\_gnb\_smote)) print ('F1 score:', f1\_score(y\_test, y\_pred\_gnb\_smote)) print ('Recall:', recall\_score(y\_test, y\_pred\_gnb\_smote)) print ('Precision:', precision\_score(y\_test, y\_pred\_gnb\_smote)) print ('AUC:', roc\_auc\_score(y\_test, y\_pred\_gnb\_smote))**  **Results:**  **Performance score for NB - RUS**  **Accuracy: 0.573878821083977**  **F1 score: 0.4240084121025734**  **Recall: 0.7457987789113617**  **Precision: 0.2962047312507405**  **AUC: 0.6369473086159304**  **---------------------------------------------------------------------------------------**  **Performance score for NB - SMOTE**  **Accuracy: 0.26362917762786436**  **F1 score: 0.35337238555190337**  **Recall: 0.9567647117315992**  **Precision: 0.21670525490761344**  **AUC: 0.5179045412506675** |

* 1. **Decision Trees Classifier Using df2\_with\_dummies Data Set**

|  |
| --- |
| **# Create instance of Decision Tree Classifier dtm\_rus = DecisionTreeClassifier() dtm\_smote = DecisionTreeClassifier()  # Fit the instance with training data dtm\_rus.fit(x\_train\_rus, y\_train\_rus) dtm\_smote.fit(x\_train\_smote, y\_train\_smote)  # Predict using the fitted model y\_pred\_dt\_rus = dtm\_rus.predict(x\_test) y\_pred\_dt\_smote = dtm\_smote.predict(x\_test)  # Build a confusion matrix and show the Classification Report cm\_dt\_rus = metrics.confusion\_matrix(y\_test,y\_pred\_dt\_rus) print('\nConfusion Matrix DT RUS','\n',cm\_dt\_rus) print('\nClassification Report DECISION TREE RUS','\n',metrics.classification\_report(y\_test,y\_pred\_dt\_rus)) print('--------------------------------------------------------------------------------')   cm\_dt\_smote = metrics.confusion\_matrix(y\_test,y\_pred\_dt\_smote) print('\nConfusion Matrix DT SMOTE','\n',cm\_dt\_smote) print('\nClassification Report DECISION TREE SMOTE','\n',metrics.classification\_report(y\_test,y\_pred\_dt\_smote)) print('--------------------------------------------------------------------------------')  Results:  Confusion Matrix DT RUS   [[109130 79688]  [ 21532 28751]]  Classification Report DECISION TREE RUS   precision recall f1-score support   0 0.84 0.58 0.68 188818  1 0.27 0.57 0.36 50283   accuracy 0.58 239101  macro avg 0.55 0.57 0.52 239101 weighted avg 0.72 0.58 0.62 239101  -------------------------------------------------------------------------------  Confusion Matrix DT SMOTE   [[148983 39835]  [ 33925 16358]]  Classification Report DECISION TREE SMOTE   precision recall f1-score support   0 0.81 0.79 0.80 188818  1 0.29 0.33 0.31 50283   accuracy 0.69 239101  macro avg 0.55 0.56 0.55 239101 weighted avg 0.70 0.69 0.70 239101  -------------------------------------------------------------------------------** |

* 1. **Random Forest Classifier Using df2\_with\_dummies Data Set**

|  |
| --- |
| **# Create a model (object) for classification rfcm\_rus = RandomForestClassifier() rfcm\_smote = RandomForestClassifier()  # Build a random forest classification model rfcm\_rus.fit(x\_train\_rus, y\_train\_rus) rfcm\_smote.fit(x\_train\_smote, y\_train\_smote)  y\_pred\_rf\_rus = rfcm\_rus.predict(x\_test) y\_pred\_rf\_smote = rfcm\_smote.predict(x\_test)  # Build a confusion matrix and show the Classification Report cm\_rf\_rus = metrics.confusion\_matrix(y\_test,y\_pred\_rf\_rus) print('\nConfusion Matrix RF RUS','\n',cm\_rf\_rus) print('\nClassification Report RF RUS','\n',metrics.classification\_report(y\_test,y\_pred\_rf\_rus)) print("---------------------------------------------------------------------------------------")   cm\_rf\_smote = metrics.confusion\_matrix(y\_test, y\_pred\_rf\_smote) print('\nConfusion Matrix RF SMOTE','\n',cm\_rf\_smote) print('\nClassification Report RF SMOTE','\n',metrics.classification\_report(y\_test,y\_pred\_rf\_smote))**  **Results:**  **Confusion Matrix RF RUS**  **[[122102 66716]**  **[ 16616 33667]]**  **Classification Report RF RUS**  **precision recall f1-score support**  **0 0.88 0.65 0.75 188818**  **1 0.34 0.67 0.45 50283**  **accuracy 0.65 239101**  **macro avg 0.61 0.66 0.60 239101**  **weighted avg 0.77 0.65 0.68 239101**  **---------------------------------------------------------------------------------------**  **Confusion Matrix RF SMOTE**  **[[182673 6145]**  **[ 44190 6093]]**  **Classification Report RF SMOTE**  **precision recall f1-score support**  **0 0.81 0.97 0.88 188818**  **1 0.50 0.12 0.19 50283**  **accuracy 0.79 239101**  **macro avg 0.65 0.54 0.54 239101**  **weighted avg 0.74 0.79 0.74 239101** |

|  |
| --- |
| **print("Performance score for RF - RUS ") print ('Accuracy:', accuracy\_score(y\_test, y\_pred\_rf\_rus)) print ('F1 score:', f1\_score(y\_test, y\_pred\_rf\_rus)) print ('Recall:', recall\_score(y\_test, y\_pred\_rf\_rus)) print ('Precision:', precision\_score(y\_test, y\_pred\_rf\_rus)) print ('AUC:', roc\_auc\_score(y\_test, y\_pred\_rf\_rus))  print("---------------------------------------------------------------------------------------")  print("Performance score for RF - SMOTE ") print ('Accuracy:', accuracy\_score(y\_test, y\_pred\_rf\_smote)) print ('F1 score:', f1\_score(y\_test, y\_pred\_rf\_smote)) print ('Recall:', recall\_score(y\_test, y\_pred\_rf\_smote)) print ('Precision:', precision\_score(y\_test, y\_pred\_rf\_smote)) print ('AUC:', roc\_auc\_score(y\_test, y\_pred\_rf\_smote))**  **Results:**  **Performance score for RF - RUS**  **Accuracy: 0.6514778273616589**  **F1 score: 0.4469090571197218**  **Recall: 0.6695503450470338**  **Precision: 0.3353854736359742**  **AUC: 0.6581076937873795**  **---------------------------------------------------------------------------------------**  **Performance score for RF - SMOTE**  **Accuracy: 0.7894822689992932**  **F1 score: 0.19491051006861693**  **Recall: 0.12117415428673707**  **Precision: 0.4978754698480144**  **AUC: 0.5443147937805536** |

* 1. **Neural Network Classifier Using df2\_with\_dummies Data Set**

|  |
| --- |
| **# Normalize the data Xn = scale(X) # Set the 'stratify' option 'y' to sample  Xn\_train, Xn\_test = train\_test\_split(Xn, test\_size =.3,random\_state=1234, stratify=y) y\_train, y\_test = train\_test\_split(y, test\_size=.3, random\_state=1234, stratify=y)  # define undersample strategy rus = RandomUnderSampler(sampling\_strategy='majority') smote = SMOTE(sampling\_strategy = 'minority') xn\_train\_rus, y\_train\_rus = rus.fit\_sample(Xn\_train, y\_train.ravel()) xn\_train\_smote, y\_train\_smote = smote.fit\_sample(Xn\_train, y\_train.ravel())  nnm\_rus = MLPClassifier(hidden\_layer\_sizes=(20,), max\_iter=1000,activation='logistic') nnm\_smote = MLPClassifier(hidden\_layer\_sizes=(20,), max\_iter=1000,activation='logistic')  # Make predictions nnm\_rus.fit(xn\_train\_rus, y\_train\_rus) nnm\_smote.fit(xn\_train\_smote, y\_train\_smote)  y\_pred\_nn\_rus = nnm\_rus.predict(Xn\_test) y\_pred\_nn\_smote = nnm\_smote.predict(Xn\_test)  print('\n \*\* Performance Scores \*\*') # Build a confusion matrix and show the Classification Report cm\_nn\_rus = metrics.confusion\_matrix(y\_test,y\_pred\_nn\_rus) print('\nConfusion Matrix','\n',cm\_nn\_rus) print('\nClassification Report Neural Network - RUS','\n',metrics.classification\_report(y\_test,y\_pred\_nn\_rus))  print("---------------------------------------------------------------------------------")  cm\_nn\_smote = metrics.confusion\_matrix(y\_test,y\_pred\_nn\_smote) print('\nConfusion Matrix','\n',cm\_nn\_smote) print('\nClassification Report Neural Network - SMOTE','\n',metrics.classification\_report(y\_test,y\_pred\_nn\_smote))**  **Results:**  **\*\* Performance Scores \*\***  **Confusion Matrix**  **[[124415 64403]**  **[ 16823 33460]]**  **Classification Report Neural Network - RUS**  **precision recall f1-score support**  **0 0.88 0.66 0.75 188818**  **1 0.34 0.67 0.45 50283**  **accuracy 0.66 239101**  **macro avg 0.61 0.66 0.60 239101**  **weighted avg 0.77 0.66 0.69 239101**  **---------------------------------------------------------------------------------**  **Confusion Matrix**  **[[159306 29512]**  **[ 31250 19033]]**  **Classification Report Neural Network - SMOTE**  **precision recall f1-score support**  **0 0.84 0.84 0.84 188818**  **1 0.39 0.38 0.39 50283**  **accuracy 0.75 239101**  **macro avg 0.61 0.61 0.61 239101**  **weighted avg 0.74 0.75 0.74 239101** |

|  |
| --- |
| **print ('Accuracy:', accuracy\_score(y\_test, y\_pred\_nn\_rus)) print ('F1 score:', f1\_score(y\_test, y\_pred\_nn\_rus)) print ('Recall:', recall\_score(y\_test, y\_pred\_nn\_rus)) print ('Precision:', precision\_score(y\_test, y\_pred\_nn\_rus)) print ('AUC:', roc\_auc\_score(y\_test, y\_pred\_nn\_rus)) print("--------------------------------------------") print ('Accuracy:', accuracy\_score(y\_test, y\_pred\_nn\_smote)) print ('F1 score:', f1\_score(y\_test, y\_pred\_nn\_smote)) print ('Recall:', recall\_score(y\_test, y\_pred\_nn\_smote)) print ('Precision:', precision\_score(y\_test, y\_pred\_nn\_smote)) print ('AUC:', roc\_auc\_score(y\_test, y\_pred\_nn\_smote))**  **Results:**  **Accuracy: 0.6602858206364675**  **F1 score: 0.4517165498899734**  **Recall: 0.6654336455660959**  **Precision: 0.3419065428200648**  **AUC: 0.6621742897618317**  **--------------------------------------------**  **Accuracy: 0.7458730829231162**  **F1 score: 0.38517424211761847**  **Recall: 0.3785175904381202**  **Precision: 0.3920692141312185**  **AUC: 0.6111094662355946** |

|  |
| --- |
| **from sklearn.metrics import mean\_squared\_error import math  mse = mean\_squared\_error(y\_test, y\_pred\_nn\_rus) rmse = math.sqrt(mse) print(rmse)**  **Results:**  **0.5828500487805869** |

1. **Machine Learning Algorithms Using K-Fold Cross Validation:**

|  |
| --- |
| **# Opening the data frame after joining df2\_with\_dummies = pd.read\_csv("df2\_with\_dummies.csv", low\_memory = False) # Removing that first column: ("Unnamed: 0") df2\_with\_dummies = df2\_with\_dummies.drop(columns = "Unnamed: 0") # Separate X from Y y = df2\_with\_dummies['charged\_off'] x = df2\_with\_dummies.drop(columns = ['charged\_off']) x = x.replace([np.inf, -np.inf], np.nan).dropna(axis=1)** |

* 1. **Naive Bayesian**

|  |
| --- |
| **# Create instance of Gaussian Naive Bayesian Classifier gnb = GaussianNB()  # Gather all the scores from the cross-validation scores = cross\_validate(gnb, x, y, cv = 5, scoring = ['accuracy', 'f1', 'recall', 'precision', 'roc\_auc', 'neg\_root\_mean\_squared\_error'])  print("Average accuracy is: ", scores['test\_accuracy'].mean(), "with values as following: ", scores['test\_accuracy']) print("Average F1 is: ", scores['test\_f1'].mean(), "with values as following: ", scores['test\_f1']) print("Average recall/sensitivity is: ", scores['test\_recall'].mean(), "with values as following: ", scores['test\_recall']) print("Average precision is: ", scores['test\_precision'].mean(), "with values as following: ", scores['test\_precision']) print("Average ROC is: ", scores['test\_roc\_auc'].mean(), "with values as following: ", scores['test\_roc\_auc']) print("Average RMSE is: ", -scores['test\_neg\_root\_mean\_squared\_error'].mean(), "with values as following: ", -scores['test\_neg\_root\_mean\_squared\_error'])**  **Results:**  **Average accuracy is: 0.5988582131472013 with values as following: [0.77199014 0.60022835 0.63190319 0.47614178 0.5140276 ]**  **Average F1 is: 0.38589208613870934 with values as following: [0.2588552 0.42426049 0.43397228 0.40141648 0.41095599]**  **Average recall/sensitivity is: 0.6404202330854024 with values as following: [0.18933834 0.7004057 0.67099218 0.83526744 0.80609749]**  **Average precision is: 0.31479645107024 with values as following: [0.40903525 0.30428979 0.32069177 0.26419136 0.27577409]**  **Average ROC is: 0.6843010638028596 with values as following: [0.67085576 0.6870313 0.69490369 0.688382 0.68033256]**  **Average RMSE is: 0.6274772584072753 with values as following: [0.47750378 0.63227498 0.60670983 0.7237805 0.6971172 ]** |

* 1. **Decision Tree**

|  |
| --- |
| **# Separate X from Y y = df2\_with\_dummies['charged\_off'] x = df2\_with\_dummies.drop(columns = ['charged\_off']) x = x.replace([np.inf, -np.inf], np.nan).dropna(axis=1)  # Create instance of Decision Tree Classifier dtm = DecisionTreeClassifier()  # Get all the scores scores = cross\_validate(dtm, x, y, cv = 5, scoring = ['accuracy', 'f1', 'recall', 'precision', 'roc\_auc', 'neg\_root\_mean\_squared\_error'])  print("Average accuracy is: ", scores['test\_accuracy'].mean()) print("Average F1 is: ", scores['test\_f1'].mean()) print("Average recall/sensitivity is: ", scores['test\_recall'].mean()) print("Average precision is: ", scores['test\_precision'].mean()) print("Average ROC is: ", scores['test\_roc\_auc'].mean()) print("Average RMSE is: ", -scores['test\_neg\_root\_mean\_squared\_error'].mean())**  **Results:**  **Average accuracy is: 0.6895519823559652**  **Average F1 is: 0.28879069008612657**  **Average recall is: 0.2998407676077757**  **Average precision is: 0.2787744890141376**  **Average ROC is: 0.5465868552723288**  **Average RMSE is: 0.557156854924912** |

* 1. **Random Forest**

|  |
| --- |
| **# Separate X from Y y = df2\_with\_dummies['charged\_off'] x = df2\_with\_dummies.drop(columns = ['charged\_off']) x = x.replace([np.inf, -np.inf], np.nan).dropna(axis=1)  rfcm = RandomForestClassifier()  scores = cross\_validate(rfcm, x, y, cv = 5, scoring = ['accuracy', 'f1', 'recall', 'precision', 'roc\_auc', 'neg\_root\_mean\_squared\_error'])  print("Average accuracy is: ", scores['test\_accuracy'].mean(), "with values as following: ", scores['test\_accuracy']) print("Average F1 is: ", scores['test\_f1'].mean(), "with values as following: ", scores['test\_f1']) print("Average recall/sensitivity is: ", scores['test\_recall'].mean(), "with values as following: ", scores['test\_recall']) print("Average precision is: ", scores['test\_precision'].mean(), "with values as following: ", scores['test\_precision']) print("Average ROC is: ", scores['test\_roc\_auc'].mean(), "with values as following: ", scores['test\_roc\_auc']) print("Average RMSE is: ", -scores['test\_neg\_root\_mean\_squared\_error'].mean(), "with values as following: ", -scores['test\_neg\_root\_mean\_squared\_error'])**  **Results:**  **Average accuracy is: 0.7915252534077212 with values as following: [0.79104272 0.78832002 0.79367131 0.79453576 0.79005646]**  **Average F1 is: 0.12893864921667322 with values as following: [0.14008365 0.14229792 0.10092671 0.15181416 0.10957082]**  **Average recall/sensitivity is: 0.07367154368907347 with values as following: [0.08093193 0.0834974 0.05506831 0.08743773 0.06142235]**  **Average precision is: 0.5375429330087163 with values as following: [0.52052955 0.48109316 0.60346518 0.5756088 0.50701798]**  **Average ROC is: 0.7014482201027674 with values as following: [0.69108974 0.69289025 0.71068061 0.71356258 0.69901793]**  **Average RMSE is: 0.45658343184535355 with values as following: [0.45711846 0.46008692 0.45423418 0.45328164 0.45819596]** |

* 1. **Neural Network:**

|  |
| --- |
| **y = df2\_with\_dummies['charged\_off'] x = df2\_with\_dummies.drop(columns = ['charged\_off']) x = x.replace([np.inf, -np.inf], np.nan).dropna(axis=1) x = StandardScaler().fit\_transform(x)  nnm = MLPClassifier(hidden\_layer\_sizes=(20,), max\_iter=2000,activation='logistic')  scores = cross\_validate(nnm, x, y, cv = 5, scoring = ['accuracy', 'f1', 'recall', 'precision', 'roc\_auc', 'neg\_root\_mean\_squared\_error'])  print("Average accuracy is: ", scores['test\_accuracy'].mean(), "with values as following: ", scores['test\_accuracy']) print("Average F1 is: ", scores['test\_f1'].mean(), "with values as following: ", scores['test\_f1']) print("Average recall/sensitivity is: ", scores['test\_recall'].mean(), "with values as following: ", scores['test\_recall']) print("Average precision is: ", scores['test\_precision'].mean(), "with values as following: ", scores['test\_precision']) print("Average ROC is: ", scores['test\_roc\_auc'].mean(), "with values as following: ", scores['test\_roc\_auc']) print("Average RMSE is: ", -scores['test\_neg\_root\_mean\_squared\_error'].mean(), "with values as following: ", -scores['test\_neg\_root\_mean\_squared\_error'])**  **Results:**  **Average accuracy is: 0.792623114607772 with values as following: [0.79255463 0.79105526 0.79404772 0.79508783 0.79037014]**  **Average F1 is: 0.17027252420063138 with values as following: [0.20060438 0.21015936 0.13221961 0.14398407 0.16439521]**  **Average recall/sensitivity is: 0.10211253474511794 with values as following: [0.12376946 0.13218185 0.07460772 0.08194863 0.09805501]**  **Average precision is: 0.5445441221915995 with values as following: [0.52900676 0.51249133 0.58041309 0.59253667 0.50827277]**  **Average ROC is: 0.7122400752810581 with values as following: [0.7096898 0.71454287 0.71018736 0.71887435 0.707906 ]**  **Average RMSE is: 0.4553823801328357 with values as following: [0.45546171 0.45710473 0.45381966 0.45267226 0.45785354]** |

1. **Area Under Curve Plotting**
   1. **Library List and Opening the File**

|  |
| --- |
| **import pandas as pd, numpy as np from sklearn.decomposition import PCA from sklearn.preprocessing import scale from sklearn.model\_selection import train\_test\_split from sklearn.tree import DecisionTreeClassifier from sklearn.neural\_network import MLPClassifier from sklearn.ensemble import RandomForestClassifier # from sklearn.ensemble import RandomForestClassifier from sklearn import metrics from sklearn.datasets import make\_classification from matplotlib import pyplot as plt import imblearn from sklearn.metrics import recall\_score, precision\_score, f1\_score from sklearn.metrics import roc\_auc\_score  from sklearn.datasets import make\_classification from sklearn.tree import DecisionTreeClassifier from sklearn.linear\_model import LogisticRegression from sklearn.metrics import roc\_curve, roc\_auc\_score from sklearn.model\_selection import train\_test\_split import matplotlib.pyplot as plt  import pandas as pd import numpy as np from sklearn.model\_selection import train\_test\_split from sklearn import metrics import statsmodels.api as sm from sklearn.metrics import \* import statsmodels.formula.api as smf import random**  **from imblearn.under\_sampling import RandomUnderSampler** |

|  |
| --- |
| **df = pd.read\_csv("df2\_with\_dummies.csv") df = df.drop(columns = ['Unnamed: 0','log\_annual\_inc', 'revol\_bal', 'UR'])  # defining target and feature variable X = df.drop(columns = 'charged\_off') X = X.replace([np.inf, -np.inf], np.nan).dropna(axis=1) y = df['charged\_off']  # Separate train and test x\_train, x\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size =.3,random\_state=1234, stratify=y) print("The shape of train data set using df2\_with\_dummies is: x = {} and y = {}".format(x\_train.shape, y\_train.shape)) print("The shape of test data set using df2\_with\_dummies is: x = {} and y = {}". format(x\_test.shape, y\_test.shape))**  **Results:**  **The shape of train data set using df2\_with\_dummies is: x = (557902, 103) and y = (557902,)**  **The shape of test data set using df2\_with\_dummies is: x = (239101, 103) and y = (239101,)** |

|  |
| --- |
| **# define undersample strategy rus = RandomUnderSampler(sampling\_strategy='majority') x\_train\_rus, y\_train\_rus = rus.fit\_sample(x\_train, y\_train.ravel())  # Normalize the data Xn = scale(X)** |

1. **Neural Network**

|  |
| --- |
| **# Set the 'stratify' option 'y' to sample  Xn\_train, Xn\_test = train\_test\_split(Xn, test\_size =.3,random\_state=1234, stratify=y) y\_train, y\_test = train\_test\_split(y, test\_size=.3, random\_state=1234, stratify=y)  # define undersample strategy rus = RandomUnderSampler(sampling\_strategy='majority') # smote = SMOTE(sampling\_strategy = 'minority') xn\_train\_rus, y\_train\_rus = rus.fit\_sample(Xn\_train, y\_train.ravel()) # xn\_train\_smote, y\_train\_smote = smote.fit\_sample(Xn\_train, y\_train.ravel())  nnm\_rus = MLPClassifier(hidden\_layer\_sizes=(20,), max\_iter=2000,activation='logistic') nnm\_rus.fit(xn\_train\_rus, y\_train\_rus)  y\_pred\_nn\_rus = nnm\_rus.predict(Xn\_test)** |

|  |
| --- |
| **false\_positive\_rate1, true\_positive\_rate1, threshold1 = roc\_curve(y\_test, y\_pred\_nn\_rus)  plt.subplots(1, figsize=(10,10)) plt.title('ROC Curve for NN - RUS', fontsize = 30) plt.plot(false\_positive\_rate1, true\_positive\_rate1) plt.plot([0, 1], ls="--") plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7") plt.ylabel('True Positive Rate') plt.xlabel('False Positive Rate') plt.ylabel('True Positive Rate', fontsize = 20) plt.xlabel('False Positive Rate', fontsize = 20) plt.show()**  **Results:** |

|  |
| --- |
| **X = df.drop(columns = 'charged\_off') X = X.replace([np.inf, -np.inf], np.nan).dropna(axis=1) y = df['charged\_off']  # splitting dataset  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size =.3,random\_state=1234, stratify=y)   # Normalize the data Xn = scale(X)   # Set the 'stratify' option 'y' to sample  Xn\_train, Xn\_test = train\_test\_split(Xn, test\_size =.3,random\_state=1234, stratify=y) y\_train, y\_test = train\_test\_split(y, test\_size=.3, random\_state=1234, stratify=y)   nnm = MLPClassifier(hidden\_layer\_sizes=(20,), max\_iter=1000,activation='logistic')  # Make predictions nnm.fit(Xn\_train, y\_train) y\_pred\_NN = nnm.predict(Xn\_test)  false\_positive\_rate1, true\_positive\_rate1, threshold1 = roc\_curve(y\_test, y\_pred\_NN)  plt.subplots(1, figsize=(10,10)) plt.title('ROC Curve for NN - Before RUS', fontsize = 30) plt.plot(false\_positive\_rate1, true\_positive\_rate1) plt.plot([0, 1], ls="--") plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7") plt.ylabel('True Positive Rate', fontsize = 20) plt.xlabel('False Positive Rate', fontsize = 20)  plt.show()**  **Results:** |

1. **Random Forest**

|  |
| --- |
| **# Create a model (object) for classification rfcm\_pca = RandomForestClassifier()  # Build a random forest classification model rfcm\_pca.fit(X\_train, y\_train) y\_pred\_rf\_pca = rfcm\_pca.predict(X\_test) fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_rf\_pca)  def plot\_roc\_curve(fpr, tpr):  plt.plot(fpr, tpr, color='orange', label='ROC')  plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')  plt.xlabel('False Positive Rate')  plt.ylabel('True Positive Rate')  plt.title('Receiver Operating Characteristic (ROC) Curve')  plt.legend()  plt.show()   plot\_roc\_curve(fpr, tpr)**  **Results: [Before RUS]** |

|  |
| --- |
| **# Create a model (object) for classification rfcm\_pca = RandomForestClassifier()  # Build a random forest classification model rfcm\_pca.fit(x\_train\_rus, y\_train\_rus) y\_pred\_rf\_pca = rfcm\_pca.predict(X\_test)  fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_rf\_pca)  def plot\_roc\_curve(fpr, tpr):  plt.plot(fpr, tpr, color='orange', label='ROC')  plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')  plt.xlabel('False Positive Rate')  plt.ylabel('True Positive Rate')  plt.title('Receiver Operating Characteristic (ROC) Curve')  plt.legend()  plt.show()   plot\_roc\_curve(fpr, tpr)**  **Results: [After RUS]** |

1. **Getting Important Features**

|  |
| --- |
| **import pandas as pd import numpy as np from sklearn.ensemble import ExtraTreesClassifier import matplotlib.pyplot as plt model = ExtraTreesClassifier() model.fit(X, y) print(model.feature\_importances\_) feat\_importances = pd.Series(model.feature\_importances\_, index = X.columns) feat\_importances.nlargest(10).plot(kind = 'barh') plt.show**  **Results:**  **[3.66552394e-02 4.42733855e-02 3.33407499e-02 3.61128577e-02**  **4.28848938e-02 4.61259260e-02 4.41893859e-02 4.20801129e-02**  **4.24434733e-02 4.47916074e-02 1.40047235e-03 1.34396737e-03**  **2.27737101e-02 2.05052217e-02 4.69377093e-03 6.96171054e-02**  **2.49303076e-02 2.21973915e-02 4.30687578e-02 3.51850614e-02**  **3.55248067e-02 3.20134880e-02 3.21831627e-02 8.76753978e-03**  **7.41317087e-03 0.00000000e+00 5.09140119e-03 6.15187792e-08**  **0.00000000e+00 2.65487753e-03 4.97853162e-03 0.00000000e+00**  **2.58126530e-03 1.92176614e-03 4.22989499e-03 7.74750011e-03**  **2.68551412e-03 2.68957831e-03 6.24763031e-04 9.39576664e-04**  **6.36480695e-03 4.38125084e-03 1.44549114e-03 0.00000000e+00**  **6.30383080e-04 4.90003820e-03 3.34280081e-03 1.87304773e-03**  **2.42199101e-03 2.43383253e-03 4.02090792e-03 4.31861551e-03**  **5.77142601e-04 4.46701038e-03 3.63366370e-03 3.12215752e-03**  **1.80788748e-03 8.75501190e-04 4.73681053e-03 6.06585550e-04**  **1.44303869e-03 1.11608667e-03 5.03055183e-03 1.63854107e-03**  **2.95112348e-03 6.08258005e-03 4.75786495e-03 2.10228572e-03**  **1.63982717e-03 4.83262817e-03 1.31847590e-03 2.09660881e-03**  **7.60728839e-04 3.29915843e-03 7.64059511e-03 1.73172611e-03**  **4.47894142e-03 5.74154785e-04 2.54276563e-03 2.71522541e-03**  **8.28758656e-04 6.15446949e-04 0.00000000e+00 5.62387959e-03**  **8.01361794e-03 3.49663756e-08 4.21721083e-03 1.26713300e-03**  **2.96383512e-03 2.08947155e-03 1.40102409e-03 4.69591948e-03**  **2.87777361e-04 1.76225128e-03 1.38681652e-03 1.32853387e-06**  **0.00000000e+00 2.40933443e-02 0.00000000e+00 1.13475978e-02**  **0.00000000e+00 4.03142273e-03 0.00000000e+00]**  **<function matplotlib.pyplot.show(\*args, \*\*kw)>** |

1. **Attempting Clustering using K-Means Clustering**

|  |
| --- |
| **Create the k number of clusters after finding k using the elbow method.  df\_n = minmax\_scale(X) df\_n   ssd = [] # Initialize the list for inertia values - sum of squared distances for i in range(2,50):  km = KMeans(n\_clusters=i, random\_state=1234)  km.fit(df\_n)  ssd.append(km.inertia\_)   # Check the inertia values. for i in range(len(ssd)):  print('{0}: {1:.2f}'.format(i+2, ssd[i]))    # Draw the plot to find the elbow   plt.plot(range(2,50), ssd) plt.grid(True) plt.xlabel('Number of Clusters') plt.ylabel('Sum of Squared Distances') plt.show()  km = KMeans(n\_clusters=8, random\_state=1234)   # Create clusters km.fit(df\_n) km.inertia\_   # Add the cluster number to the original data. df2\_with\_dummies['ClusterNo'] = km.labels\_ df2\_with\_dummies.head()  # Divide the original data into the clusters.   Cluster0 = df2\_with\_dummies.loc[df\_with\_dummies.ClusterNo == 0] Cluster0.describe() Cluster0.info()   Cluster1 = df2\_with\_dummies.loc[df2\_with\_dummies.ClusterNo == 1] Cluster2 = df2\_with\_dummies.loc[df2\_with\_dummies.ClusterNo == 2] Cluster3 = df2\_with\_dummies.loc[df2\_with\_dummies.ClusterNo == 3] Cluster4 = df2\_with\_dummies.loc[df2\_with\_dummies.ClusterNo == 4] Cluster5 = df2\_with\_dummies.loc[df2\_with\_dummies.ClusterNo == 5] Cluster6 = df2\_with\_dummies.loc[df2\_with\_dummies.ClusterNo == 6] Cluster7 = df2\_with\_dummies.loc[df2\_with\_dummies.ClusterNo == 7]  # Now, you can apply ml algorithms to each cluster.   # Description of profiles of each cluster df2\_with\_dummies.groupby(['ClusterNo']).count() df2\_with\_dummies.groupby(['ClusterNo']).mean() df2\_with\_dummies.groupby(['ClusterNo']).median() df2\_with\_dummies.groupby(['ClusterNo']).max() df2\_with\_dummies.groupby(['ClusterNo']).min()   # Set up X and y for each cluster X0 = Cluster0.drop(columns=['charged\_off','ClusterNo'], axis = 1) X0 = X.replace([np.inf, -np.inf], np.nan).dropna(axis=1) y0 = Cluster0.charged\_off X1 = Cluster1.drop(columns=['charged\_off','ClusterNo'], axis = 1) X1 = X1.replace([np.inf, -np.inf], np.nan).dropna(axis=1) y1 = Cluster1.charged\_off X2 = Cluster2.drop(columns=['charged\_off','ClusterNo'], axis = 1) X2 = X2.replace([np.inf, -np.inf], np.nan).dropna(axis=1) y2 = Cluster2.charged\_off X3 = Cluster3.drop(columns=['charged\_off','ClusterNo'], axis = 1) X3 = X3.replace([np.inf, -np.inf], np.nan).dropna(axis=1) y3 = Cluster3.charged\_off X4 = Cluster4.drop(columns=['charged\_off','ClusterNo'], axis = 1) X4 = X4.replace([np.inf, -np.inf], np.nan).dropna(axis=1) y4 = Cluster4.charged\_off X5 = Cluster5.drop(columns=['charged\_off','ClusterNo'], axis = 1) X5 = X5.replace([np.inf, -np.inf], np.nan).dropna(axis=1) y5 = Cluster5.charged\_off X6 = Cluster6.drop(columns=['charged\_off','ClusterNo'], axis = 1) X6 = X6.replace([np.inf, -np.inf], np.nan).dropna(axis=1) y6 = Cluster6.charged\_off X7 = Cluster7.drop(columns=['charged\_off','ClusterNo'], axis = 1) X7 = X7.replace([np.inf, -np.inf], np.nan).dropna(axis=1) y7 = Cluster7.charged\_off   #Split to train and test set X\_train0, X\_test0, y\_train0, y\_test0 = train\_test\_split(X0,y0,test\_size =.3,random\_state=1234, stratify=y0) X\_train1, X\_test1, y\_train1, y\_test1 = train\_test\_split(X1,y1,test\_size =.3,random\_state = 1234, stratify=y1) X\_train2, X\_test2, y\_train2, y\_test2 = train\_test\_split(X2,y2,test\_size =.3,random\_state=1234, stratify=y2) X\_train3, X\_test3, y\_train3, y\_test3 = train\_test\_split(X3,y3,test\_size =.3,random\_state=1234, stratify=y3) X\_train4, X\_test4, y\_train4, y\_test4 = train\_test\_split(X4,y4,test\_size =.3,random\_state=1234, stratify=y4) X\_train5, X\_test5, y\_train5, y\_test5 = train\_test\_split(X5,y5,test\_size =.3,random\_state=1234, stratify=y5) X\_train6, X\_test6, y\_train6, y\_test6 = train\_test\_split(X6,y6,test\_size =.3,random\_state=1234, stratify=y6) X\_train7, X\_test7, y\_train7, y\_test7 = train\_test\_split(X7,y7,test\_size =.3,random\_state=1234, stratify=y7)   # Create a model (object) for classification rfcm0 = RandomForestClassifier() rfcm1 = RandomForestClassifier() rfcm2 = RandomForestClassifier() rfcm3 = RandomForestClassifier() rfcm4 = RandomForestClassifier() rfcm5 = RandomForestClassifier() rfcm6 = RandomForestClassifier() rfcm7 = RandomForestClassifier()   # Build random forest classification models rfcm0.fit(X\_train0, y\_train0) y\_pred0 = rfcm0.predict(X\_test0) rfcm1.fit(X\_train1, y\_train1) y\_pred1 = rfcm1.predict(X\_test1) rfcm2.fit(X\_train2, y\_train2) y\_pred2 = rfcm2.predict(X\_test2) rfcm3.fit(X\_train3, y\_train3) y\_pred3 = rfcm3.predict(X\_test3) rfcm4.fit(X\_train4, y\_train4) y\_pred4 = rfcm4.predict(X\_test4) rfcm5.fit(X\_train5, y\_train5) y\_pred5 = rfcm5.predict(X\_test5) rfcm6.fit(X\_train6, y\_train6) y\_pred6 = rfcm6.predict(X\_test6) rfcm7.fit(X\_train7, y\_train7) y\_pred7 = rfcm7.predict(X\_test7)   # Print the performance  print('\n \*\* Performance Scores \*\*')    # Calculate accuracy accuracy0 = rfcm0.score(X\_test0, y\_test0) print('Accuracy0: {0:.2f}'.format(accuracy0))   # Build a confusion matrix and show the Classification Report cm0 = metrics.confusion\_matrix(y\_test0,y\_pred0) print('\nConfusion Matrix','\n',cm0) print('\nClassification Report','\n',metrics.classification\_report(y\_test0,y\_pred0))   # Print the performance  print('\n \*\* Performance Scores \*\*')   # Calculate accuracy accuracy1 = rfcm1.score(X\_test1, y\_test1) print('Accuracy1: {0:.2f}'.format(accuracy1))   # Build a confusion matrix and show the Classification Report cm1 = metrics.confusion\_matrix(y\_test1,y\_pred1) print('\nConfusion Matrix','\n',cm1) print('\nClassification Report','\n',metrics.classification\_report(y\_test1,y\_pred1))   # Print the performance  print('\n \*\* Performance Scores \*\*')   # Calculate accuracy accuracy2 = rfcm2.score(X\_test2, y\_test2) print('Accuracy2: {0:.2f}'.format(accuracy2))   # Build a confusion matrix and show the Classification Report cm2 = metrics.confusion\_matrix(y\_test2,y\_pred2) print('\nConfusion Matrix','\n',cm2) print('\nClassification Report','\n',metrics.classification\_report(y\_test2,y\_pred2))   # Print the performance  print('\n \*\* Performance Scores \*\*')   # Calculate accuracy accuracy3 = rfcm3.score(X\_test3, y\_test3) print('Accuracy3: {0:.2f}'.format(accuracy3))   # Build a confusion matrix and show the Classification Report cm3 = metrics.confusion\_matrix(y\_test3,y\_pred3) print('\nConfusion Matrix','\n',cm3) print('\nClassification Report','\n',metrics.classification\_report(y\_test3,y\_pred3))   # Print the performance  print('\n \*\* Performance Scores \*\*')   # Calculate accuracy accuracy4 = rfcm4.score(X\_test4, y\_test4) print('Accuracy4: {0:.2f}'.format(accuracy4))   # Build a confusion matrix and show the Classification Report cm4 = metrics.confusion\_matrix(y\_test4,y\_pred4) print('\nConfusion Matrix','\n',cm4) print('\nClassification Report','\n',metrics.classification\_report(y\_test4,y\_pred4))   # Print the performance  print('\n \*\* Performance Scores \*\*')   # Calculate accuracy accuracy5 = rfcm5.score(X\_test5, y\_test5) print('Accuracy5: {0:.2f}'.format(accuracy5))   # Build a confusion matrix and show the Classification Report cm5 = metrics.confusion\_matrix(y\_test5,y\_pred5) print('\nConfusion Matrix','\n',cm) print('\nClassification Report','\n',metrics.classification\_report(y\_test5,y\_pred5))   # Print the performance  print('\n \*\* Performance Scores \*\*')   # Calculate accuracy accuracy6 = rfcm6.score(X\_test6, y\_test6) print('Accuracy6: {0:.2f}'.format(accuracy6))   # Build a confusion matrix and show the Classification Report cm6 = metrics.confusion\_matrix(y\_test6,y\_pred6) print('\nConfusion Matrix','\n',cm6) print('\nClassification Report','\n',metrics.classification\_report(y\_test6,y\_pred6))   # Print the performance  print('\n \*\* Performance Scores \*\*')   # Calculate accuracy accuracy7 = rfcm7.score(X\_test7, y\_test7) print('Accuracy7: {0:.2f}'.format(accuracy7))   # Build a confusion matrix and show the Classification Report cm7 = metrics.confusion\_matrix(y\_test7,y\_pred7) print('\nConfusion Matrix','\n',cm7) print('\nClassification Report','\n',metrics.classification\_report(y\_test7,y\_pred7))   # Create a Bayesian Classifier instance for classification gnb0 = GaussianNB() gnb1 = GaussianNB() gnb2 = GaussianNB() gnb3 = GaussianNB() gnb4 = GaussianNB() gnb5 = GaussianNB() gnb6 = GaussianNB() gnb7 = GaussianNB()   gnb0.fit(X\_train0, y\_train0) y\_pred0 = gnb0.predict(X\_test0) gnb1.fit(X\_train1, y\_train1) y\_pred1 = gnb1.predict(X\_test1) gnb2.fit(X\_train2, y\_train2) y\_pred2 = gnb2.predict(X\_test2) gnb3.fit(X\_train3, y\_train3) y\_pred3 = gnb3.predict(X\_test3) gnb4.fit(X\_train4, y\_train4) y\_pred4 = gnb4.predict(X\_test4) gnb5.fit(X\_train5, y\_train5) y\_pred5 = gnb5.predict(X\_test5) gnb6.fit(X\_train6, y\_train6) y\_pred6 = gnb6.predict(X\_test6) gnb7.fit(X\_train7, y\_train7) y\_pred7 = gnb7.predict(X\_test7)   # Calculate the accuracy accuracy0 = gnb0.score(X\_test0, y\_test0) print('Accuracy0: {0:.2f}'.format(accuracy0)) accuracy1 = gnb1.score(X\_test1, y\_test1) print('Accuracy1: {0:.2f}'.format(accuracy1)) accuracy2 = gnb2.score(X\_test2, y\_test2) print('Accuracy2: {0:.2f}'.format(accuracy2)) accuracy3 = gnb3.score(X\_test3, y\_test3) print('Accuracy3: {0:.2f}'.format(accuracy3)) accuracy4 = gnb4.score(X\_test4, y\_test4) print('Accuracy4: {0:.2f}'.format(accuracy4)) accuracy5 = gnb5.score(X\_test5, y\_test5) print('Accuracy5: {0:.2f}'.format(accuracy5)) accuracy6 = gnb6.score(X\_test6, y\_test6) print('Accuracy6: {0:.2f}'.format(accuracy6)) accuracy7 = gnb7.score(X\_test7, y\_test7) print('Accuracy7: {0:.2f}'.format(accuracy7))   # Build a confusion matrix cm0 = metrics.confusion\_matrix(y\_test0,y\_pred0) print(metrics.classification\_report(y\_test0,y\_pred0)) cm1 = metrics.confusion\_matrix(y\_test1,y\_pred1) print(metrics.classification\_report(y\_test1,y\_pred1)) cm2 = metrics.confusion\_matrix(y\_test2,y\_pred2) print(metrics.classification\_report(y\_test2,y\_pred2)) cm3 = metrics.confusion\_matrix(y\_test3,y\_pred3) print(metrics.classification\_report(y\_test3,y\_pred3)) cm4 = metrics.confusion\_matrix(y\_test4,y\_pred4) print(metrics.classification\_report(y\_test4,y\_pred4)) cm5 = metrics.confusion\_matrix(y\_test5,y\_pred5) print(metrics.classification\_report(y\_test5,y\_pred5)) cm6 = metrics.confusion\_matrix(y\_test6,y\_pred6) print(metrics.classification\_report(y\_test6,y\_pred6)) cm7 = metrics.confusion\_matrix(y\_test7,y\_pred7) print(metrics.classification\_report(y\_test7,y\_pred7))** |