Fixed Income Final Project

Leading Club Loan Risk Analysis

Chang Yu Shuke Guan Yibin Zhang Yixiao Wang

December 2019

Intentionally left black

Disclaimer

This disclaimer governs the use of this report. By using this report, you accept this disclaimer in full.

You must not rely on the information in the report as an alternative to financial advice from an appropriately qualified professional. If you have any specific questions about any financial matter you should consult an appropriately qualified professional.

Without prejudice to the generality of the foregoing paragraph, we do not represent, warrant, undertake or guarantee that the use of guidance in the report will lead to any particular outcome or result.

We will not be liable to you in respect of any business losses, including without limitation loss of or damage to profits, income, revenue, use, production, anticipated savings, business, contracts, commercial opportunities or goodwill.

Intentionally left black

Acknowledgement

We would like to express our special thanks of gratitude to our instructor Tony Zhang who provides us with valuable guidance in the class and precious criticism on presentation. His keen and vigorous academic observation enlightens me not only in this thesis but also in my future study. Tony gives us the golden opportunity to do this wonderful project on the topic of credit risk prediction model based on real world data, which also helps us in doing a lot of research and we came to know about so many new things we are really thankful to them.

Our sincere appreciation also goes to the students from MQF program, who participated this study with great cooperation.

We appreciate Rutgers University gives us the necessary technique support and academic database.

Intentionally left black

Contents

Contents 7
Chapter I 2
1.1 Introduction 2
1.1.1 Background 2
1.1.2 Report Structures 3
1.2 Business Understanding 4
1.2.1 Business Objectives 4
1.2.2 Assess Situation 4
1.2.3 Data Mining Goals 5
1.2.4 Project Plan 5
Chapter II 6
2.1 Descriptive Statistics 6
2.1.1 Data Dimensions 6
2.1.2 Loan Status 7
2.1.3 Interest rate 7
2.1.4 Loan 9
2.1.5 Income and Purpose 11
2.1.6 Region 13
2.2 Data Preparation 15
2.2.1 General Steps 15
2.2.2 Attributes Analysis 15
2.2.3 Deal with Logical Contradiction and Information Leakage Problem 17
2.2.4 Outcome 17
Chapter III 18
3.1 Random Forest 18
3.1.1 Decision tree learning 18
3.1.2 Bagging (Bootstrap aggregating) 19
3.1.3 From bagging to random forests 20
3.1.4 Basic Evaluation 20

3.2 Data Resampling 21
3.2.1 Random Over Sampling 21
3.2.2 Random Under Sampling 23
3.3 Tuning parameters 25
3.3.1 n_estimators 25
3.3.2 max_depth 25
3.3.3 min_sample_split 25
3.3.4 min_sample_leaf 25
3.3.5 max_features 25
3.4 Model Evaluation 26
Chapter IV 28
4.1 Conclusions 28
4.1.1 Model evaluation 28
4.1.2 Future Improvements 28
Appendix 28

Chapter I

Project Overview

1.1 Introduction

With the development of computer technologies and algorithms, it is possible to find the trend blew the face of the world. The ability to collect, storage, and distribution data also enable companies, governments, and even individuals to utilize different methods and develop different approaches to generalize and summaries the patterns from nature world to human society. This project focuses on investigating the risks on the loan applications by implementing data mining and machine learning techniques.

1.1.1 Background

The development of computer and internet does not only enhance the development of science and techniques but also dramatically changed other industries—biology, psychology, architecture, and finance.

The ability to communicate frequently and shortly made the finance changing. The traditional products, which catch clients off-line and were limited in a local area, becomes a globally oriented products.

Loans, was designed and distributed locally, can be applied by those from all over the country through internet. With the tremendous increasing of target clients, the financial companies are rewarded by the quick growing but also facing increasing risk. In such environment, it is especially important to have the ability to detect and prevent risks.

By deploying technologies such as big data and machine learning, it is possible for these financial companies to acquire the ability to discover risks and prevent risks through historical data analysis and feature learning methods. This project aims to predict the probability of loan defaults by analyzing historical loan data and training models to help these financial companies in the probability of these risk events in the operation process.

This is the introduction for the whole project. In this project, I don't do much job, and I hope all of you can understand my role and don't do anything out of your boundary.

1.1.2 Report Structures

This report mainly describes all the steps and techniques applied in this project.

The first part introduces the background, main goals and analysis of the project. The second part analyzes the data used in the project in detail, and all the attributes used in the project for machine learning. The third part shows the process of model establishment and the results of model training. And improved results from previous results. The different performances of different models on this project are also discussed. The last section discusses the shortcomings of this project and areas for improvement.

1.2 Business Understanding

This part we discuss the understanding of this project. Objectives, successes threshold are presented.

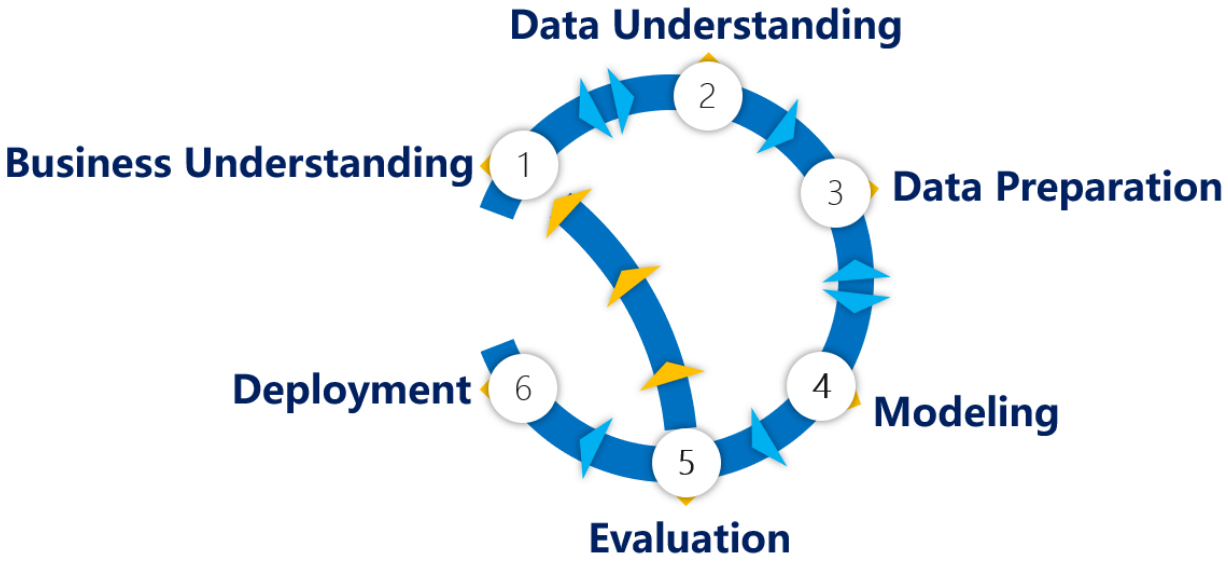


Figure 1The general steps of data mining projects

1.2.1 Business Objectives

According to analysis results of the loan applications and history credit of clients, the prediction model should presents the possibility of the default of the loan. This model is designed to help company decide should ratify the loan applications or not.

1.2.2 Assess Situation

Data Availability

Historical loans data with various attributes including credit score, loan amount, house own situation and so on from historical operations.

Schedule of Completion

The model should be presented by the end of Dec 10, 2019.

Costs and Benefits

Roughly cost 72 to 120 hours.

This project can help all teammates on machine learning and teamwork skills and also help all teammates get good grade on fixed income course.

1.2.3 Data Mining Goals

Perform Anomaly Detection

Try to identify the rare items, events or observations with raise suspicions by differing significantly from the majority of the data.

In this case, we want to detect the events of charged off or default.

Success Criteria

We expect the overall accuracy of prediction model excesses 60%.

1.2.4 Project Plan

Follow the general steps of data mining project, preforming data understating, data preparation, modeling, evaluation and deployment stages.

Restart from business understanding stage if necessary.

Chapter II

The Data

2.1 Descriptive Statistics

In order to build the model, we need to fully understand the data. We need to use data mining methods to discover the deeper relationships between different features of the data. We try to do the descriptive statistics on the dataset.

2.1.1 Data Dimensions

Firstly, check at the dimensions of the dataset. The dataset contains 35808 entries, which was from the history of the LendingClub operations, these are all real-world data. Each entry has 138 attributes, each detailed depicted the applicant's identifications. The datatype of the dataset mainly contains two types of data: the numerical and categorical ones. We should treat these two kinds of data differently, since they may represent different features of the loan.

2.1.2 Loan Status

Our mainly target of the model is to predict the result of the loan, so we would like to see how the data distribution is like, so we draw the pie chart and the loan status frequency by year.

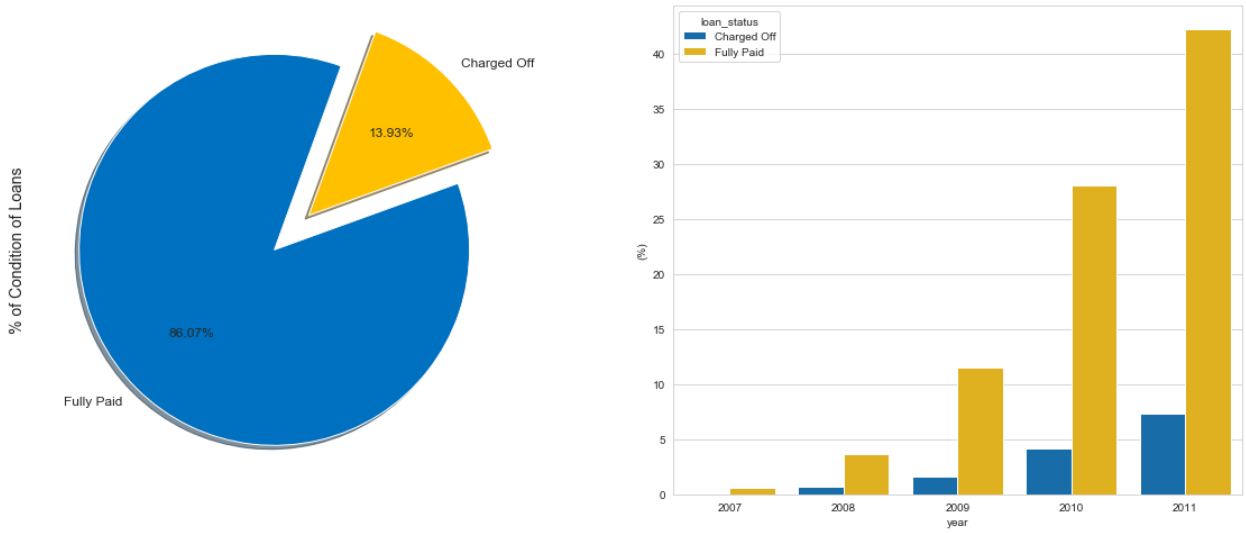


Figure 2Information on Loan Status

From figure I we can see that it is a biased dataset, the status is not balanced. The loans that are charged of takes place of 13.93%, the loans that are fully paid takes place of 96.07%. The great difference between these two numbers indicated that we must be aware of the unbalanced feature. We need to do further analysis to see how we can better deal with this feature.

2.1.3 Interest rate

Interest rate is a very important feature in the dataset. Interest rate can reflect the credit risk of the loan, a higher interest rate implies that the loan may have a greater possibility of default. Also, a higher interest rate may cause the borrower to choose to default. So we must have a deep look at the interest rate.

Firstly we look at the distribution of the interest rate, we would like to see how is the interest rate distributed. Figure II is the interest rate distribution graph.

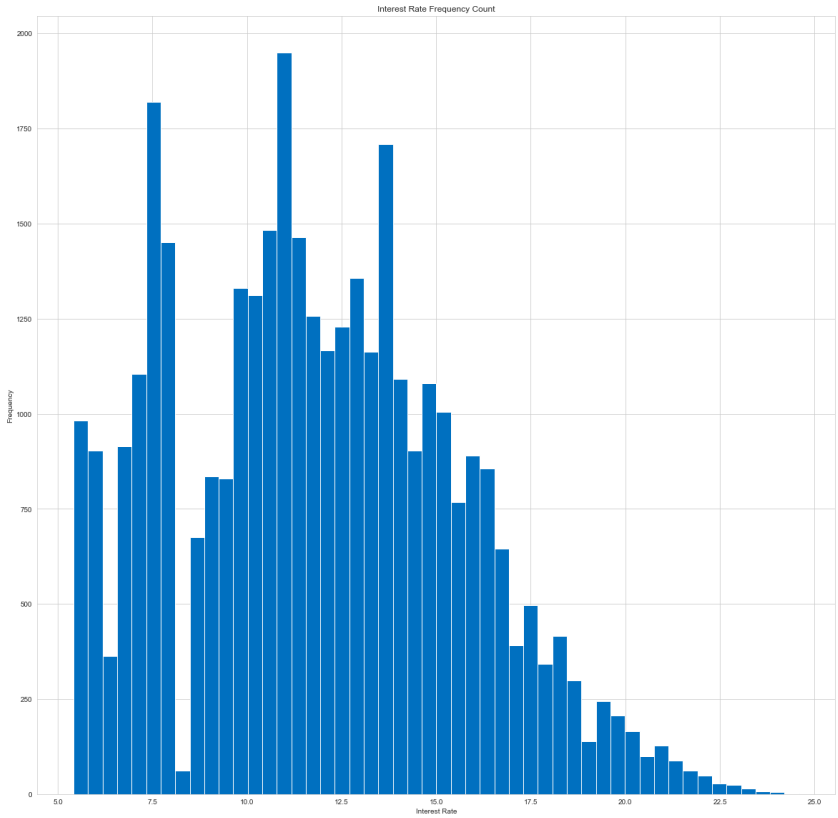


Figure 3Interest Distribution

We can see from the figure that the interest rate mainly is a skewed distribution, it doesn't obey normal distribution and it is skewed to the right. Further, it is possible for us to divide the interest rate into two groups. The low group and the high group. The criterion value is set to be 13.23%.

Then we would be possible to see the impact of two groups.

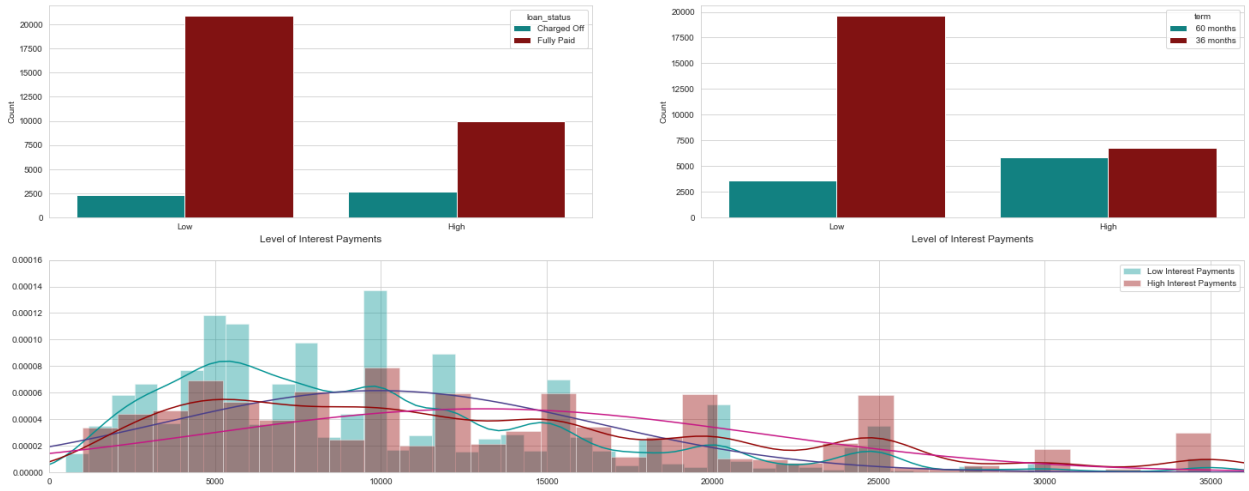


Figure 4Interest analysis

Figure III gives us the impact of interest rate on many different factors. Sub figure 1 gives us the impact of interest rate on the status condition of the loan, we can see that the high group has a great number of fully paid while it also has a great number of charged off. Based on the frequency count, we should say that the low interest rate group has lower ratio of charged off.

Sub figure 2 gives us the impact of maturity date on interest rates, which we can see that the lower interest rate group has a bigger ratio of short-term loan than the high interest rate group, which we should see the correlation of these features later.

Figure 5Average Purpose Interest Rate by Income Category

We would like to compare the income category with the purpose category, so by figure IV we would be able to see the impact of income on interest rate, also we can compare the interest rate within different types of purpose category. From the scatter we can find that the relationship is not really linear, the high-income group always have high interest rate, which we should dig deeper in future analysis.

2.1.4 Loan

We would also be interested in several features about the loan. Firstly, we would like to check the grade and the sub-grade of these loans. These are very useful factors in the analysis of credit risk.

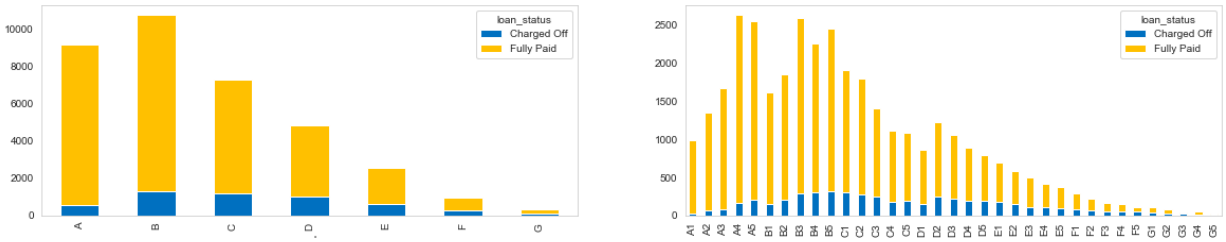


Figure 6 Types of Loans by Grade and Sub-grade

Figure V gives us the count of the grades and subgrades of the total loan. We can find in the figure that all grades of loan may default. Even grade A loan also have the possibility of default. Also, not all the grade G loan defaulted, but the ratio of default loans of A is much smaller than G, which is very easy to understand that the less the grade, the more likely that the loan would default. The sub figure gives us the information about the sub-grade, the several rule in the grade can also be applied in the sub grades.

Figure VI gives us the average interest rate and their loan status. From the figure we can see that the interest rate goes up year by year from 2007 to 2011. Also, we can find that the average charged off interest rate is much higher than the fully paid group, which is the same rule as we've discussed in the interest section.

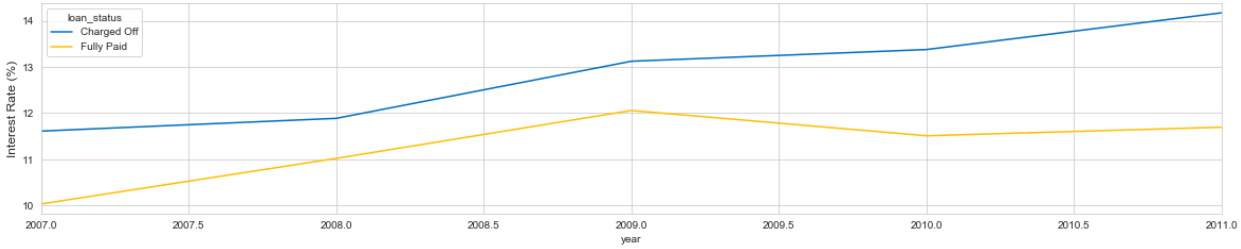


Figure 7 Average Interest Rate by Loan Condition

The last we want to check is the relation ship between the purpose of the loan and the loan status. Figure VII gives us the information we want. We can check from it that the purpose of small business has the most default rate, the purpose of wedding has the least default rate.

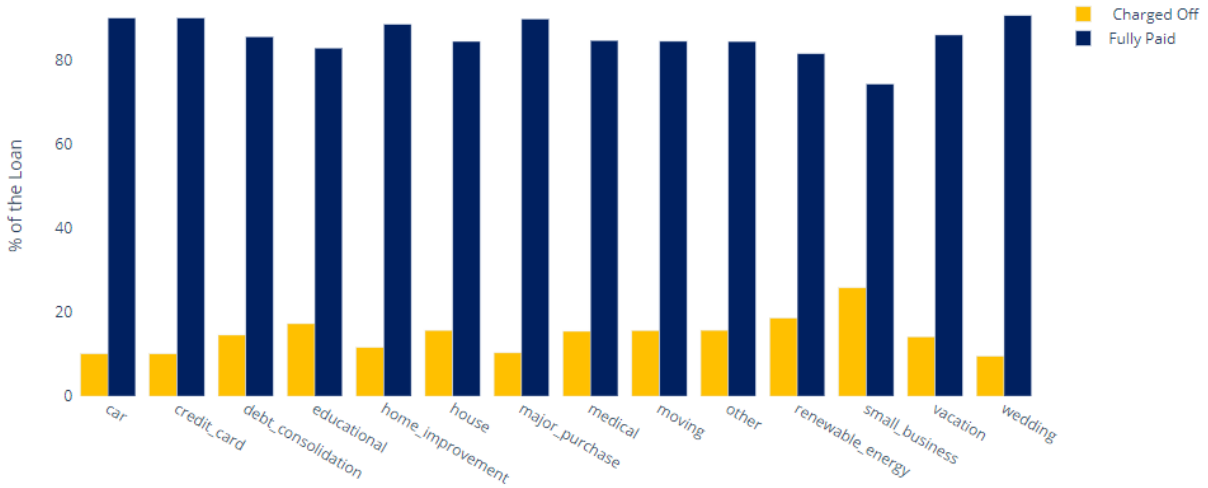


Figure 8 Condition of Loan by Purpose

2.1.5 Income and Purpose

We also want to check the category of income and purpose. We would like to see the count of the different category. The figure VIII gives us the frequency count, average interest rate and the bad/good ratio of all the loans.

From the figure we can have lots of useful data, we can find that high income loan with purpose of credit card and educational has a high default rate. Medium income loan with purpose of major purpose has no default cases. Low income loan with purpose of home improvement has more default cases than fully paid cases.

In deeper analysis, we may need to divide into several parts using the category and treat them differently, because different category have many different features.

	income_category	purpose	interest_rate	total_loan_amount	good_loans_count	bad_loans_count	total_loans_issued	bad/good ratio (%)
0	High	car	9.68727	11427.3	11	5	16	31.25
6	High	major_purchase	12.8153	14507	30	73	103	70.87
1	High	credit_card	12.6445	17605.4	51	512	563	90.94
2	High	debt_consolidation	14.2419	20588.2	173	2	175	1.14
4	High	home_improvement	12.5839	20042	100	1	101	0.99
5	High	house	12.66	25222.2	8	9	17	52.94
3	High	educational	11.42	11000	1	18	19	94.74
7	High	medical	12.645	17483.9	14	1	15	6.67
8	High	moving	14.8513	16512.5	6	2	8	25
10	High	renewable_energy	17.7333	18333.3	2	9	11	81.82
11	High	small_business	13.0884	20961.2	33	5	38	13.16
12	High	wedding	13.2062	16396.2	11	2	13	15.38
9	High	other	14.284	14724.1	44	1	45	2.22
16	Low	educational	11.65	6219.48	240	382	622	61.41
19	Low	major_purchase	10.8808	7481.71	1625	1	1626	0.06
13	Low	car	10.5383	6446.21	1161	49	1210	4.05
14	Low	credit_card	11.3114	10539.3	3224	198	3422	5.79
15	Low	debt_consolidation	12.2598	11639.9	12247	137	12384	1.11
17	Low	home_improvement	11.2285	9827.14	1790	2158	3948	54.66
18	Low	house	11.9736	11385.2	249	50	299	16.72
21	Low	moving	11.5697	5710.43	390	44	434	10.14
23	Low	renewable_energy	11.3875	7337.99	61	80	141	56.74
24	Low	small_business	12.9009	12293.3	1013	92	1105	8.33
25	Low	vacation	10.724	5068.33	269	16	285	5.61
26	Low	wedding	11.735	9158.5	692	380	1072	35.45
22	Low	other	11.6837	7367.19	2741	46	2787	1.65
20	Low	medical	11.4969	7756.1	461	273	734	37.19
33	Medium	major_purchase	10.3167	12404.8	218	0	218	0
27	Medium	car	10.1463	8833.17	140	9	149	6.04
28	Medium	credit_card	12.209	16278	630	9	639	1.41
29	Medium	debt_consolidation	13.1585	18215.5	1737	3	1740	0.17
31	Medium	home_improvement	11.5619	15063	575	10	585	1.71
32	Medium	house	12.7828	17503.5	34	50	84	59.52
36	Medium	other	12.0649	12513.9	336	15	351	4.27
34	Medium	medical	11.4913	11085	68	40	108	37.04
35	Medium	moving	10.9968	12378.8	63	5	68	7.35
37	Medium	renewable_energy	11.5694	13891.2	16	7	23	30.43
38	Medium	small_business	13.5485	18211.4	205	3	208	1.44
39	Medium	vacation	10.9803	7617.19	29	59	88	67.05
30	Medium	educational	11.6718	11750	28	231	259	89.19
40	Medium	wedding	12.4879	13379.3	95	0	95	0

Figure 9 Group by Different Features

2.1.6 Region

In this part we would like to analyze the regional impact on loans. Figure IX gives us the all the loans that are issued in different regions. We can see that with years gone by, the loans issued in all areas goes up greatly, in southwest has the least loan issued, north east area has the most loan issued.

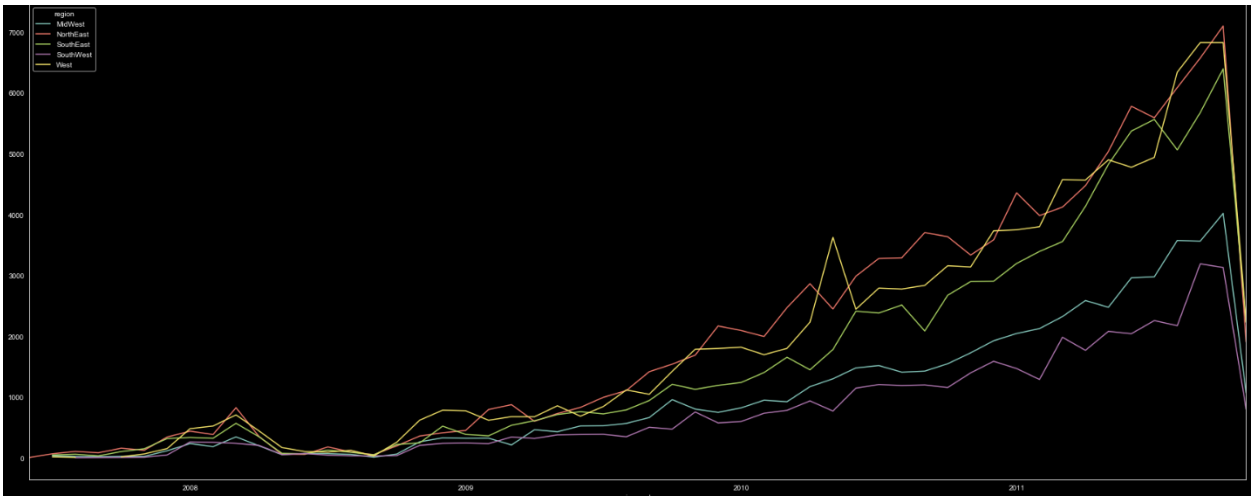


Figure 10 Loan Issued by Region

Figure X gives us the loan counts that are issued by state, we can see from the graph that the California state has the most loan issued.

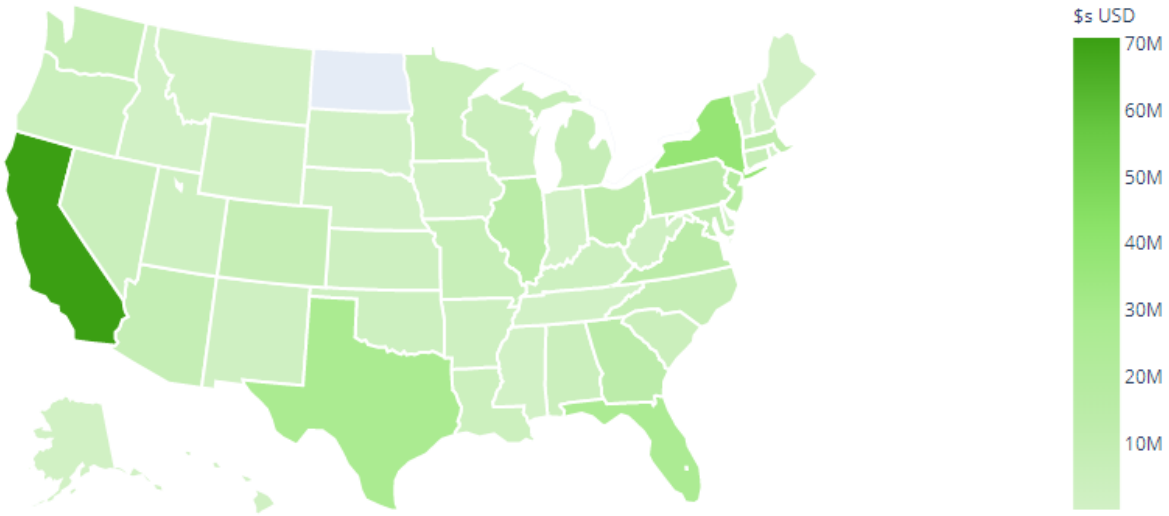


Figure 11 Loans Issued by States

Figure XI gives us the counts of the loans that have been charged off. We can see from the graph that the Nebraska State has the most numbers of charged off loans.

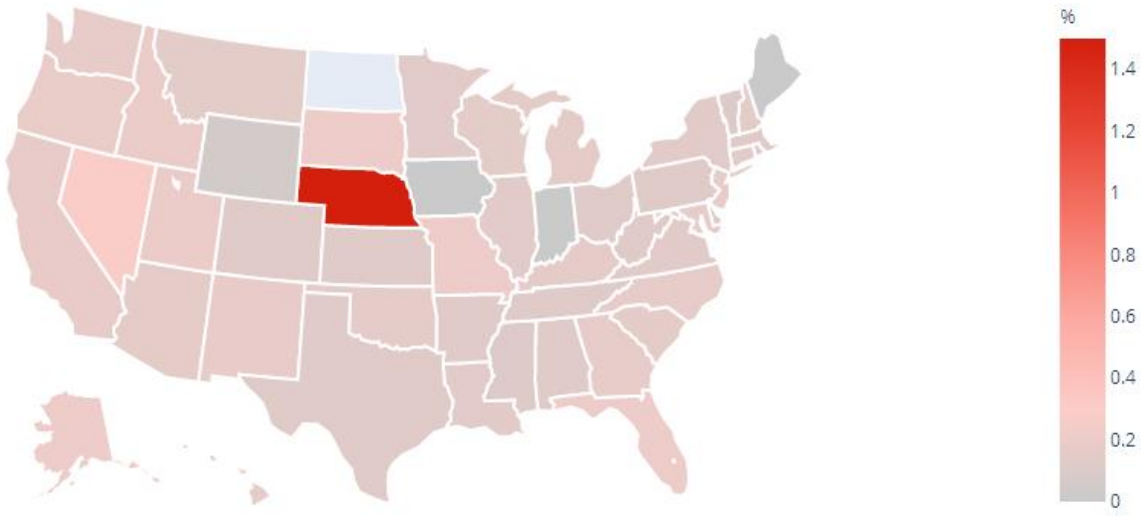


Figure 12 Charged Off Rates by States

2.2 Data Preparation

Data preparation, also can be called as data preprocessing or data cleaning which means to deeply work out data we have to make it accessible and effective. According to the results of data understanding, we can choose meaningful variables for mode training and testing.

2.2.1 General Steps

There are four steps we went through in order to sort and filter our data:

2.2.1.1 Deal with Null Values

According to the data set of this project, there are 35,808 observations and 138 variables collected. Plenty of them are null values which means the data for some attributes are incomplete. Thus, we have to get rid of these data to proceed our model.

2.2.1.2 Deal with Useless Attributes

Also, after we cleaning some values, we have to figure out if there are some attributes are irrelevant to our target. In this step, we will go through general and statistical ways to consider and test if there are some attributes should be removed.

2.2.1.3 Normalization

At this step, we will adjust some data to make their mathematical characteristics better to plug into our model. In this case, mostly we use the formula below to normalize our

attributes.

$$x_{normalization} = \frac{x - Min}{Max - Min}$$

Also, we transformed some string attributes to several vectors to complete our digitalization.

2.2.1.4 One Hot Encoding

One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction. we use one hot encoder to perform “binarization” of the category and include it as a feature to train the model.

2.2.2 Attributes Analysis

This part is focusing on this specific project to show how we deal with this dataset and finalize our attributes.

2.2.2.1 Deal with Data Deficient

Here, the data deficient I specified means some attributes have totally blank values or some attributes have the identical values for each observation. For example, ‘application_type" for

each observation is ‘individual’ which shows zero difference; ‘mths_since_recent_revol_delinq’ is totally blank. Therefore, from this step, we can delete 94 useless attributes.

2.2.2.2 Deal with Same Implication

Same implication means there are some attributes possessing the identical meanings. We go through this step before statistical analysis with correlation matrix because even if the perfect statistical properties cannot veil the homogeneity in explaining probability of default. For example, ‘sub_grade’ ‘revol_bal’ are both reflecting the credit level of our borrower. General speaking, their implications are the same. Thus, for the diverse analysis, we decide to remove ‘revol_bal’ to make our model more convincible.

2.2.2.3 Deal with High Correlation

Then, we would like to check out the correlation matrix to get rid of multicollinearity.

The partial heatmap shows below:

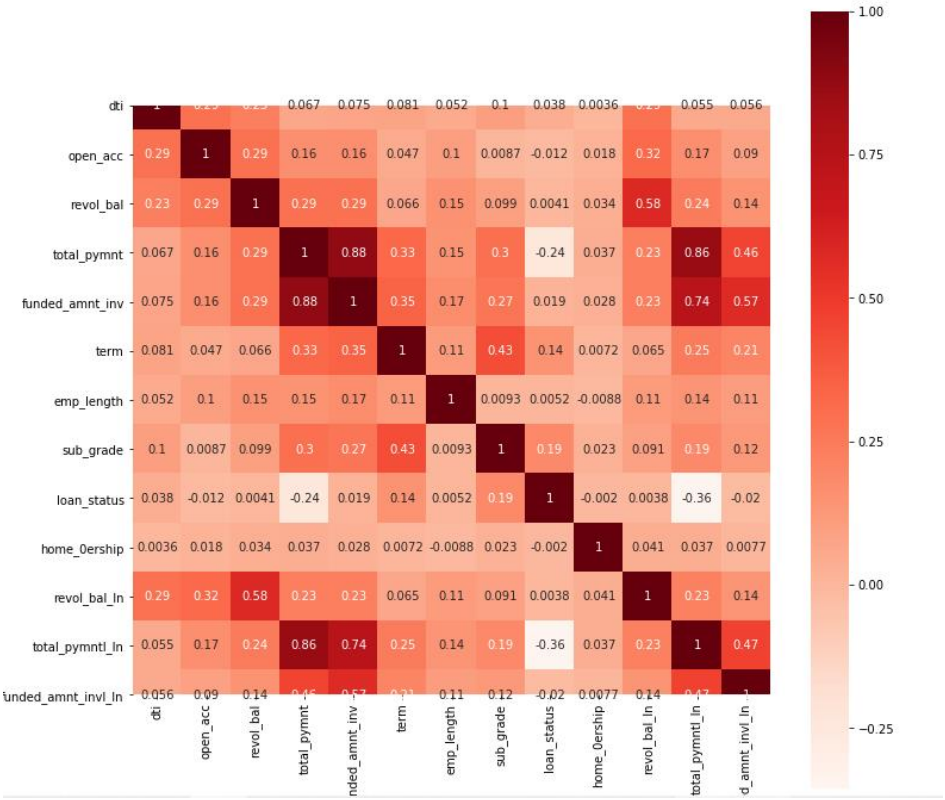


Figure 13 Correlation Matrix for Numeric Attributes

Obviously, we find out ‘funded_amnt_inv’ and ‘dpi’ have the high correlation. Thus, after several testing in model, we removed ‘funded_amnt_inv’ for reducing the multicollinearity.

2.2.3 Deal with Logical Contradiction and Information Leakage Problem

Here is our last step, but the most vital one since this could be very lethal if we take some attributes into our model.

Apparently, we use ‘loan_status’ as the explained variable in our model. Therefore, we cannot take the attributes which are caused by ‘loan_status’ into account. For example, ‘recoveries’ means “post charge off gross recovery” which is the consequence of charged-off. So, it cannot be used to predict if one borrower will default.

Another lethal attribute is ‘total_rec_prncp’. Total recovered principal minus loan_amount is the outstanding principal, which indicates charged off or default. This means it is identical to the loan status. This is an information Leakage problem. It may be the reason why some models get 99% accuracy. Using such attributes will totally destroy the predication ability of the model.

2.2.4 Outcome

From all steps above, we strictly pick up 12 attributes applied in our model:

Table 1 Attibutes Selected

loan_status
loan_amnt
sub_grade
term
home_ownership
annual_inc
purpose
revol_util
verification_status
dti
open_acc
inq_last_6mths

Among this, the ‘loan_status’ is our target(y) and others are used to predict the probability of ‘loan_status’.

Chapter III

The Model

3.1 Random Forest

Random forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set. For this topic, as half of the attributes are categorical variables and we've transformed them into vector variables, it would be better use random forest to process training.

3.1.1 Decision tree learning

A decision tree is a tree in which each internal (non-leaf) node is labeled with an input feature. The arcs coming from a node labeled with an input feature are labeled with each of the possible values of the target or output feature or the arc leads to a subordinate decision node on a different input feature. Each leaf of the tree is labeled with a class or a probability distribution over the classes, signifying that the data set has been classified by the tree into either a specific class, or

into a particular probability distribution (which, if the decision tree is well-constructed, is skewed towards certain subsets of classes).

A tree is built by splitting the source set, constituting the root node of the tree, into subsets – which constitute the successor children. The splitting is based on a set of splitting rules based on classification features. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node has all the same values of the target variable, or when splitting no longer adds value to the predictions. This process of top-down induction of decision is an example of a greedy algorithm, and it is by far the most common strategy for learning decision trees from data.

In particular, trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, i.e. have low bias, but very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model.

3.1.2 Bagging (Bootstrap aggregating)

Bootstrap aggregating, also called bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting.

Main description of the technique: Given a standard training set ***D*** of size ***N***, bagging generates ***m*** new training sets ***d***, each of size ***n***, by sampling from ***D*** uniformly and with replacement. By sampling with replacement, some observations may be repeated in each ***d***. Then ***m*** models are fitted using the above ***m*** bootstrap samples and combined by averaging the voting for classification.

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

The number of samples/trees is a free parameter. Typically, a few hundred to several thousand trees are used, depending on the size and nature of the training set. An optimal number of trees can be found by using cross-validation, or by observing the out-of-bag error. The training and test error tend to level off after some number of trees have been fit.

3.1.3 From bagging to random forests

The above procedure describes the original bagging algorithm for trees. Random forests differ in only one way from this general scheme: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called "feature bagging". The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the B trees, causing them to become correlated.

Typically, for a classification problem with p features, √p (rounded down) features are used in each split. For regression problems the inventors recommend p/3 (rounded down) with a minimum node size of 5 as the default. In practice the best values for these parameters will depend on the problem, and they should be treated as tuning parameters.

3.1.4 Basic Evaluation

Firstly, we choose two main scores to judge our model: Accuracy score and AUC-ROC score. For default detect task, we should focus on the default detect ability. Thus, we should not only judge model by accuracy score but mainly based on the AUC-ROC Curve (The Area Under the Receiver Operating Characteristic Curve). The AUC-ROC Curve score gives us a good idea of how well the model performances which gives us the true prediction ability on each tag data. Then we use python package named “scikit-learn” to implement random forest algorithm on our original processed data. The accuracy score gave us 82.7% which is not bad at all but the AUC-ROC score showed only 50.59% which indicates that our model can’t actually judge default. We compute confusion matrix (figure I) which indicates our prediction ability on each category.

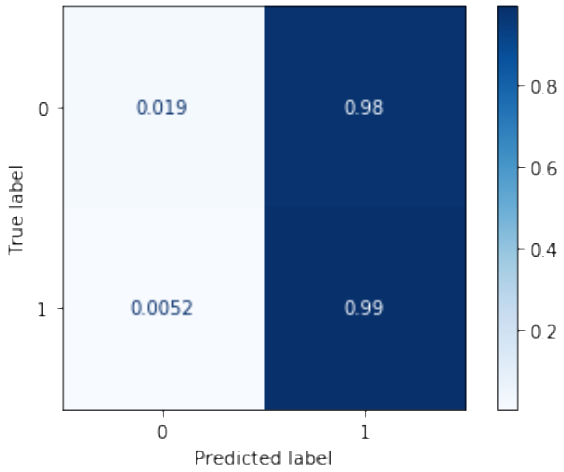


Figure 14 Basic Evaluation

It's obvious that this model predicts almost all the input to non-default target. As we've discussed before, our training data set is imbalanced on two categories. We should implement some technologies to transform our training data into balanced status.

3.2 Data Resampling

There are two basic ways to implement resample process: **Over Sampling Process** and **Under Sampling Process**. We can't decide which one is better on this data set, so we implement both of them to find the better score one to implement parameters optimization process. And for each process we implemented three methods to resample original dataset and find out the best one to present each process. There several methods for each process and we use the python package "imbalanced-learn" to implement those methods.

For Over Sampling Process, we use Random Over Sampling, SMOTE, and ADASYN.

3.2.1 Random Over Sampling

The most naïve strategy is to generate new samples by randomly sampling with replacement the current available samples. The **Random Over Sampling** offers such scheme. As the result (figure II) shows, the majority class does not take over the other classed during the training process. Consequently, all classes are represented by the decision function. The AUC-AOC score is higher than we predicted on original data set.

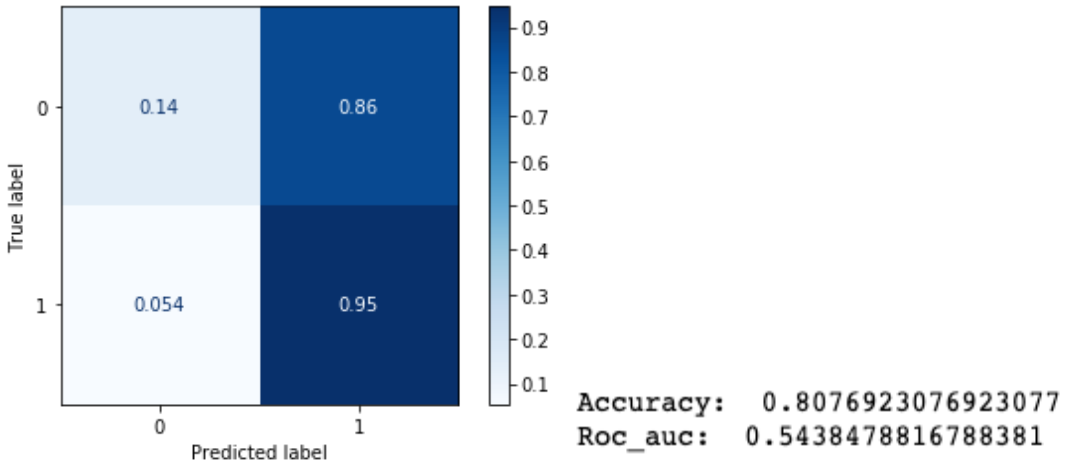


Figure 15 Original data random forest model

3.2.1.1 The Synthetic Minority Oversampling Technique (SMOTE)

This is a statistical technique for increasing the number of cases in our dataset in a balanced way. The module works by generating new instances from existing minority input cases. This implementation of SMOTE does not change the number of majority cases.

The new instances are not just copying of existing minority cases; instead, the algorithm takes samples of the feature space for each target class and its nearest neighbors, and generates new examples that combine features of the target case with features of its neighbors. This approach increases the features available to each class and makes the samples more general. Algorithm as following:

1. For each sample x in the minority class, calculate its distance to all samples in the sub sample of the minority class using the Euclidean distance as a standard, and obtain its k nearest neighbor.
2. A sampling ratio is set according to the sample imbalance ratio to determine the sampling magnification N. For each minority class subset x, several samples are randomly selected from its k nearest neighbors, assuming that the selected neighbor is x_n.
3. For each randomly selected neighbor x_n, a new sample is constructed with the original sample according to the following formula:

$$x_{new} = x + rand(0,1) \times |x - x_n|$$

As the result shows, it's also better than the original data set. However, this algorithm has some disadvantages. One is that it increases the possibility of data overlapping, and the other one is that it could generate some useless information data samples.

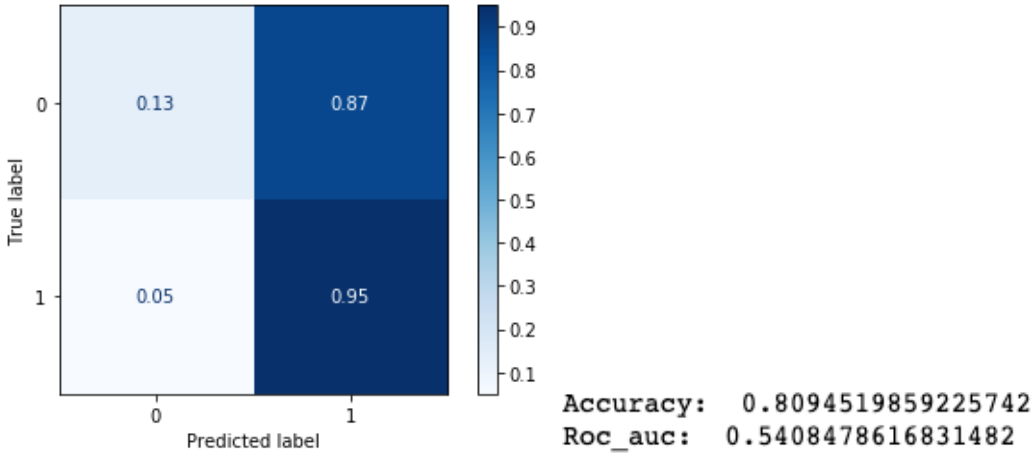


Figure 16 SMOTE

3.2.1.2 The Adaptive Synthetic (ADASYN)

This algorithm is an improvement version of SMOTE. It automatically decides each minority class sample need to generate how many synthetic samples, not generates the same number of synthetic samples for each minority class sample as SMOTE does. The process as following:

1. Calculate the total number of synthetic.
2. Calculate k nearest neighbor and generate distribution for each minority class sample.
3. Calculate synthetic samples for each minority class sample and use SMOTE algorithm to generate new sample.

As the result shows, it's also better than the original data set.

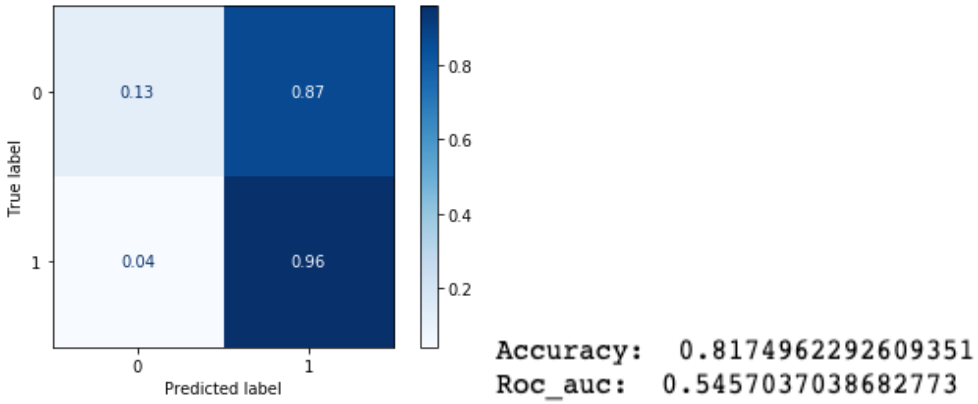


Figure 17ADASYN

We can find out each over sampling process has almost the same result for accuracy score and AUC-ROC score, we choose Random Over Sampling as the representation of Over Sampling Process because of the highest AUC-ROC score.

For Under Sampling Process, we use Random Under Sampling, NearMiss-1, and Edited Nearest Neighbors.

3.2.2 Random Under Sampling

This method is almost as the same as Random Over Sampling but adverse. It allows to bootstrap the data and the resampling with multiple classes is performed by considering independently each targeted class. As the result (figure V) shows, it improves obviously on ROC-AUC score and have a better understanding on detecting default task although it has an unreliable prediction accuracy.

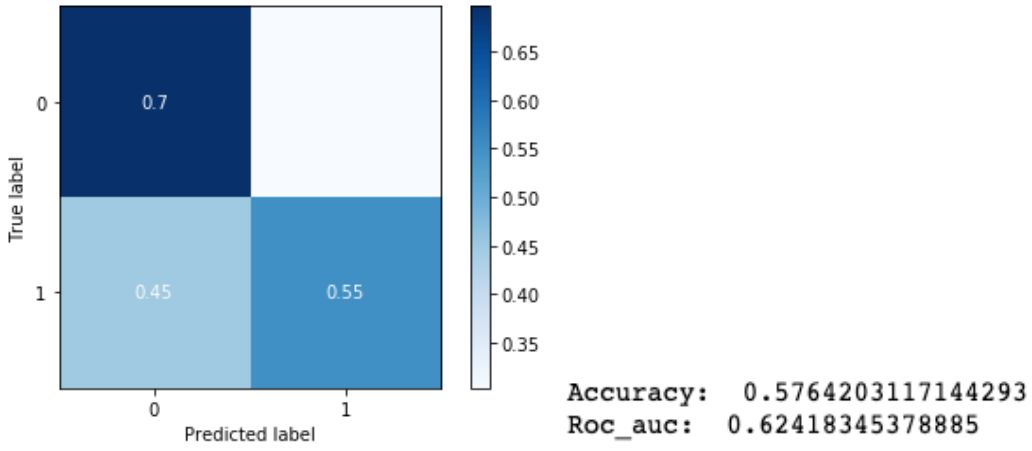


Figure 18 Random Under Sampling

3.2.2.1 NearMiss-1

Let positive samples be the samples belonging to the targeted class to the under-sampled. Negative sample refers to the samples from the minority class. NearMiss-1 selects the positive

sample for which the average distance to the N closest samples of the negative class is the smallest. As the result shows, this method has an even higher precision on default detection but has a worse result of accuracy.

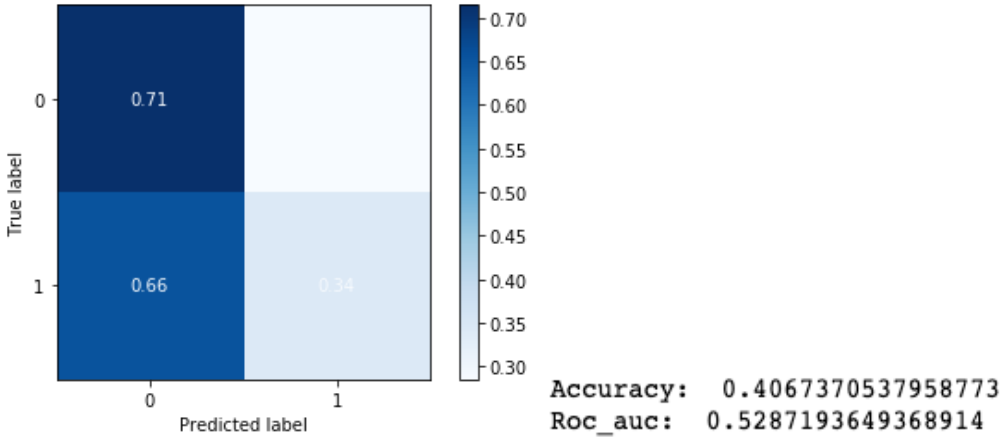


Figure 19 NearMiss-1

3.2.2.2 Edited Nearest Neighbors

Edited data set using nearest neighbors (Edited Nearest Neighbors) applies a nearest-neighbors algorithm and "edit" the dataset by removing samples which do not agree "enough" with their neighborhood. For each sample in the class to be under-sampled, the nearest-neighbors are computed and if the selection criterion is not fulfilled, the sample is removed.

As the following result shows, it looks like an over sampling process result. It should have a better interpretation on the default task, actually it not.

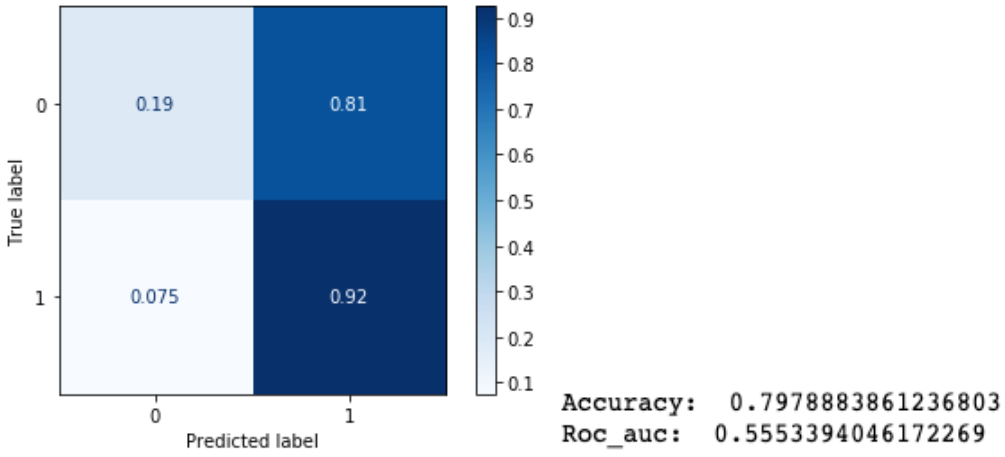


Figure 20 Edited Nearest Neighbors

We can find out each under sampling process has different results for accuracy score and AUC-ROC score. Random Under Sampling method dominates both accuracy score and AUC-ROC score on NearMiss-1 method. And as we focus on the ability to detect default ability, the Random Under Sampling method's probability of prediction correctly on default target is higher than NearMiss-1 method's probability. We choose Random Under Sampling method as the representation of Under Sampling Process.

3.3 Tuning parameters

All tests above is based on the default settings of random forest algorithm on scikit-learn package. In order to improve our model, we should implement some tuning parameters technologies on our model. There five main parameters in random forest algorithm we should focus on.

3.3.1 n_estimators

The number of trees in the forest. In general, n_estimators is too small, which is easy to underfit. If n_estimators is too large, the amount of calculation will be too large, and after n_estimators reaches a certain number, increasing the model precision obtained by n_estimators will be small, so generally choose a moderate value.

3.3.2 max_depth

The maximum depth of the tree. If not be set, decision tree will be built without limitation of the sub tree's depth. As for this topic, we should limit the maximum depth of the tree.

3.3.3 min_sample_split

The minimum number of samples required to split an internal node. If the number of samples is less than this value, it will not continue to split. We consider this parameter because we have up to more than 60,000 samples in our over sampling process result.

3.3.4 min_sample_leaf

The minimum number of samples required to be at a leaf node. This parameter can be the function as pruning. Setting this parameter can speed up our training process.

3.3.5 max_features

The minimum number of samples required to be at a leaf node. The most important one in all parameters. In general, set it to $\sqrt{N}$. If the number of features is big enough, we should use other value to control the time for decision tree generating.

In order to find the best parameters for each resampling process, we use grid search method to do it. The basic idea for this method is that we change one or two parameters and let others to be

fixed, and repeat this process as all parameters go through it. We also need a target score to judge which parameter is better based on the training data set. In this topic, as we've discussed before, we use AUC-ROC score as the target score. The tuning parameters result as following table.

	Random Over Sampling	Random Under Sampling
n_estimators	70	70
max_depth	13	13
min_samples_leaf	10	150
min_samples_split	10	10
max_features	9	7
Training AUC-ROC	0.7749	0.6827

3.4 Model Evaluation

In previous section, we find out our best parameters for each process. In this section, we implement those parameters into our random forest model and test the test data set.

For Random Over Sampling process, we get the **Accuracy Score**: **63.50%**, **AUC-ROC Score**: **63.98%**. For Random Under Sampling process, we get the **Accuracy Score**: **62.09%**, **AUC-ROC Score**: **64.40%**. The confusion matrix for each one as following:

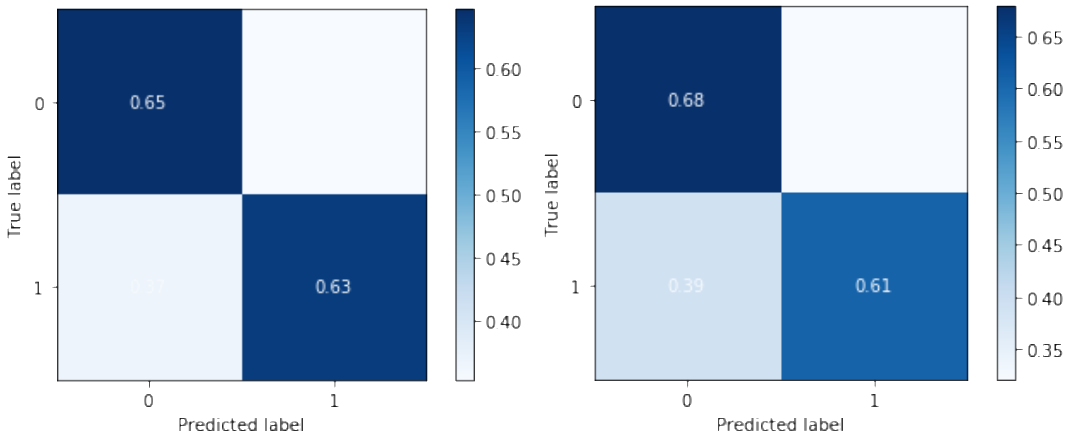


Figure 21 Random Over Sampling vs Random Under Sampling

For this condition, random under sampling process can beat random over sampling process. However, as we use random sampling method, the result for each process will be a little different. We perform resampling trainings for each of the two processes, and ensemble the average result of each model to obtain the final model.

We implement final model and get the **AUC-ROC Score: 65%,** the AUC-ROC Curve as following:

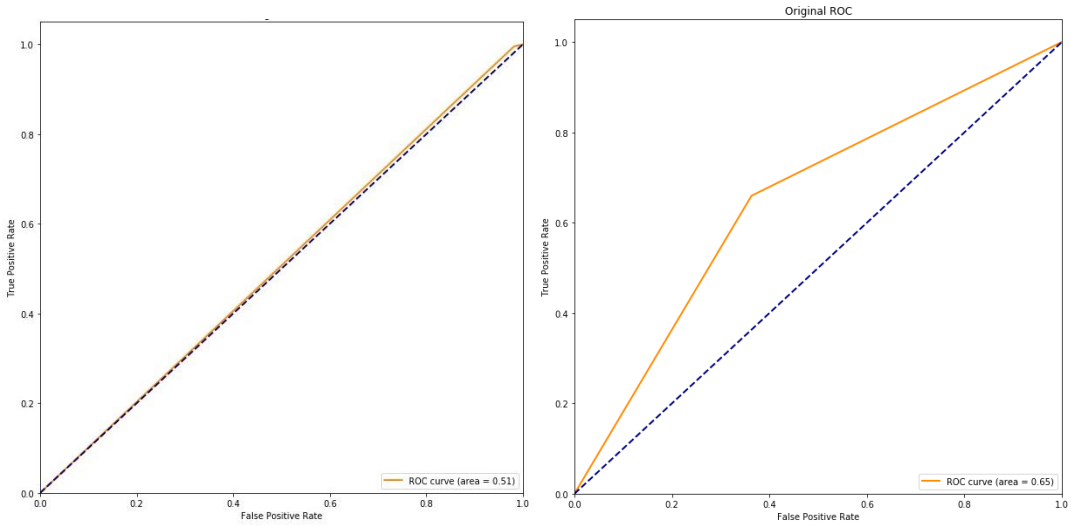


Figure 22 Original AUC-ROC Curve vs Final AUC-ROC Curve

We can say that our model have a basic prediction ability on the default detection task.

Chapter IV

Summary

4.1 Conclusions

4.1.1 Model evaluation

After these steps, we got the model and our prediction result. Since the use of our model is to predict the personal loan credit risk, we need to trade off between total accuracy and the roc value. For financial institutions, it would be more harmful to mismatch a default loan to fully paid rather than inverse. We need to have a higher roc-value in the trade off over the total accuracy, but it would be more useful since we can catch more than 60% of the default cases.

Another important part in the model is that since we are dealing with an unbalanced dataset, we need to resample the dataset to make the classification procedure more efficient. Different kinds of resampling methods are tried, including over sample and under sample. For out purpose, it would be better if we resample the fully paid data.

4.1.2 Future Improvements

The model can still be improved in several sections. More methods can be tried in the resampling procedure to try to get a better representation of the original data. We need to select the data that contains the more general cases.

We can also get a better trade off between the total accuracy and the roc-value. It would be great if we can improve these two things at the same time, that would be a perfect model that we can predict most of the data, both charge off and fully paid.

Appendix

Appendix I Data Preprocessing

```
# Import our libraries we are going to use for our data analysis.
import tensorflow as tf
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

# Plotly visualizations
from plotly import tools
import chart_studio.plotly as py
import plotly.figure_factory as ff
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected=True)
# plotly.tools.set_credentials_file(username='AlexanderBach', api_key='o4fx6i1MtEIJQxfWYvU1')


# For oversampling Library (Dealing with Imbalanced Datasets)
from imblearn.over_sampling import SMOTE
from collections import Counter

# Other Libraries
import time

%matplotlib inline
```

```
pd.set_option("display.max_rows", 100)
pd.set_option("display.max_columns", 100)
```

```
df = pd.read_excel('LendingClubData_training.xlsx')
```

```
# Copy of the dataframe
original_df = df.copy()

print("Data imported!")
print(df.shape)
```

```
Data imported!
(35808, 145)
```

```
df = original_df.copy()
```

```
# Replace the name of some columns
df = df.rename(columns={"loan_amnt": "loan_amount", "funded_amnt": "funded_amount", "funded_amnt_inv":
"investor_funds",
                        "int_rate": "interest_rate", "annual_inc": "annual_income"})

# Drop irrelevant columns
df.drop(['id', 'member_id', 'emp_title', 'url', 'desc', 'zip_code', 'title'], axis=1, inplace=True)
```

```
df['interest_rate'] = df['interest_rate'].multiply(100)
```

Data Understanding & Preprocessing

```
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	loan_amount	funded_amount	investor_funds	term	interest_rate	installment	grade	sub_grade	emp_length	home_ownership	ar
0	35000	35000	34975.0	60 months	11.71	773.44	B	B3	10+ years	MORTGAGE	11
1	9500	9500	9500.0	36 months	14.65	327.70	C	C3	10+ years	RENT	54
2	3800	3800	3800.0	36 months	7.51	118.23	A	A3	< 1 year	MORTGAGE	47
3	12400	12400	12400.0	60 months	22.06	342.90	F	F4	9 years	OWN	65
4	4000	4000	4000.0	60 months	17.27	100.00	D	D3	4 years	RENT	45

5 rows × 138 columns

```
df.shape
```

```
(35808, 138)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35808 entries, 0 to 35807
Columns: 138 entries, loan_amount to settlement_term
dtypes: datetime64[ns](6), float64(103), int64(13), object(16)
memory usage: 37.7+ MB
```

```
print(df['loan_status'].value_counts())
```

```
Fully Paid     30821
Charged Off     4987
Name: loan_status, dtype: int64
```

```
df.isnull().sum().sort_values(ascending=True)
```

```
loan_amount                         0
revol_bal                           0
total_acc                           0
out_prncp                           0
out_prncp_inv                       0
                                  ...
mths_since_recent_revol_delinq    35808
mths_since_recent_inq             35808
mths_since_recent_bc_dlq          35808
total_bal_il                      35808
```

```
avg_cur_bal                     35808
Length: 138, dtype: int64
```

```
df = df.dropna(axis=1,how='all')
```

```
df.isnull().sum().sort_values(ascending=True)
```

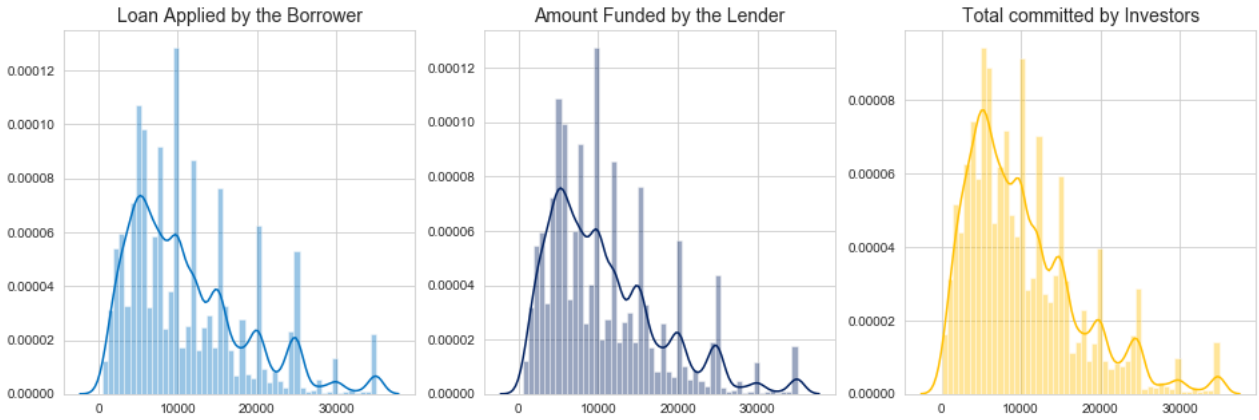
```
loan_amount                         0
revol_bal                           0
total_acc                           0
out_prncp                           0
out_prncp_inv                       0
total_pymnt                         0
total_pymnt_inv                     0
total_rec_prncp                     0
total_rec_int                       0
total_rec_late_fee                  0
recoveries                          0
collection_recovery_fee             0
last_pymnt_amnt                     0
policy_code                         0
application_type                    0
acc_now_delinq                      0
delinq_amnt                         0
hardship_flag                       0
disbursement_method                 0
debt_settlement_flag                0
pub_rec                             0
open_acc                            0
initial_list_status                 0
annual_income                       0
interest_rate                       0
installment                         0
grade                               0
sub_grade                           0
investor_funds                      0
home_ownership                      0
verification_status                 0
issue_d                             0
loan_status                         0
term                                0
purpose                             0
addr_state                          0
dti                                 0
funded_amount                       0
delinq_2yrs                         0
earliest_cr_line                    0
inq_last_6mths                      0
pymnt_plan                          0
last_credit_pull_d                  2
tax_liens                          39
revol_util                         49
collections_12_mths_ex_med         56
chargeoff_within_12_mths           56
last_pymnt_d                       68
pub_rec_bankruptcies              697
emp_length                        953
mths_since_last_delinq          22933
mths_since_last_record          33140
settlement_percentage           35682
debt_settlement_flag_date       35682
settlement_status               35682
settlement_date                 35682
settlement_amount               35682
settlement_term                 35682
dtype: int64
```

```
df.drop(['settlement_percentage', 'debt_settlement_flag_date', 'settlement_status',
         'settlement_date', 'settlement_amount', 'settlement_term'], axis=1, inplace=True)
```

```
for col in df.columns:
    results = df[col].value_counts()
    if results.size == 1:
        df.drop([col], axis=1, inplace=True)
```

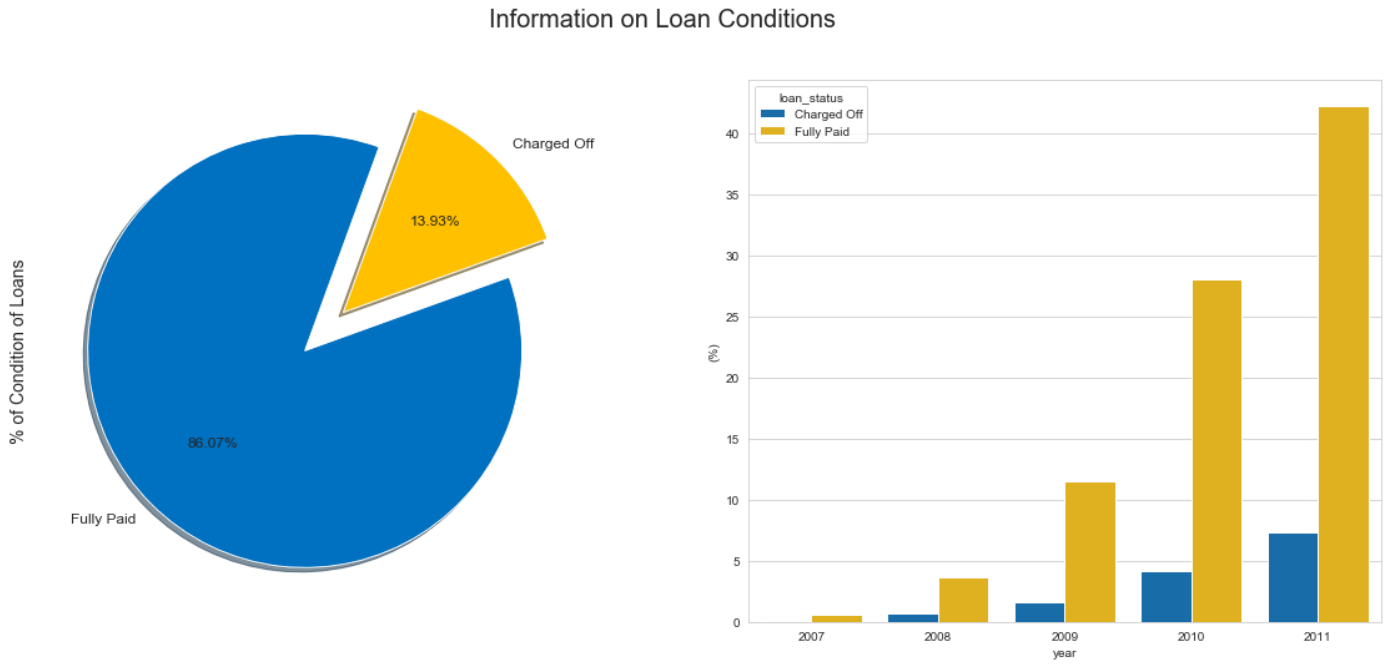
```
fig, ax = plt.subplots(1, 3, figsize=(16,5))

loan_amount = df["loan_amount"].values
funded_amount = df["funded_amount"].values
investor_funds = df["investor_funds"].values

sns.distplot(loan_amount, ax=ax[0], color="#0070C0")
ax[0].set_title("Loan Applied by the Borrower", fontsize=14)
sns.distplot(funded_amount, ax=ax[1], color="#002060")
ax[1].set_title("Amount Funded by the Lender", fontsize=14)
sns.distplot(investor_funds, ax=ax[2], color="#FFC000")
ax[2].set_title("Total committed by Investors", fontsize=14)
```

```
Text(0.5, 1.0, 'Total committed by Investors')
```




```
dt_series = pd.to_datetime(df['issue_d'])
df['year'] = dt_series.dt.year
```

```
f, ax = plt.subplots(1,2, figsize=(20,8))

colors = ["#0070C0", "#FFC000"]
labels ="Fully Paid", "Charged Off"

plt.suptitle('Information on Loan Conditions', fontsize=20)

df["loan_status"].value_counts().plot.pie(explode=[0,0.25], autopct='%1.2f%%', ax=ax[0], shadow=True, colors=colors,
                                          labels=labels, fontsize=12, startangle=70)


# ax[0].set_title('State of Loan', fontsize=16)
ax[0].set_ylabel('% of Condition of Loans', fontsize=14)

# sns.countplot('loan_condition', data=df, ax=ax[1], palette=colors)
# ax[1].set_title('Condition of Loans', fontsize=20)
# ax[1].set_xticklabels(['Good', 'Bad'], rotation='horizontal')
palette = ["#0070C0", "#FFC000"]

sns.barplot(x="year", y="loan_amount", hue="loan_status", data=df, palette=palette, estimator=lambda x: len(x) /
len(df) * 100)
ax[1].set(ylabel="(%)")
```

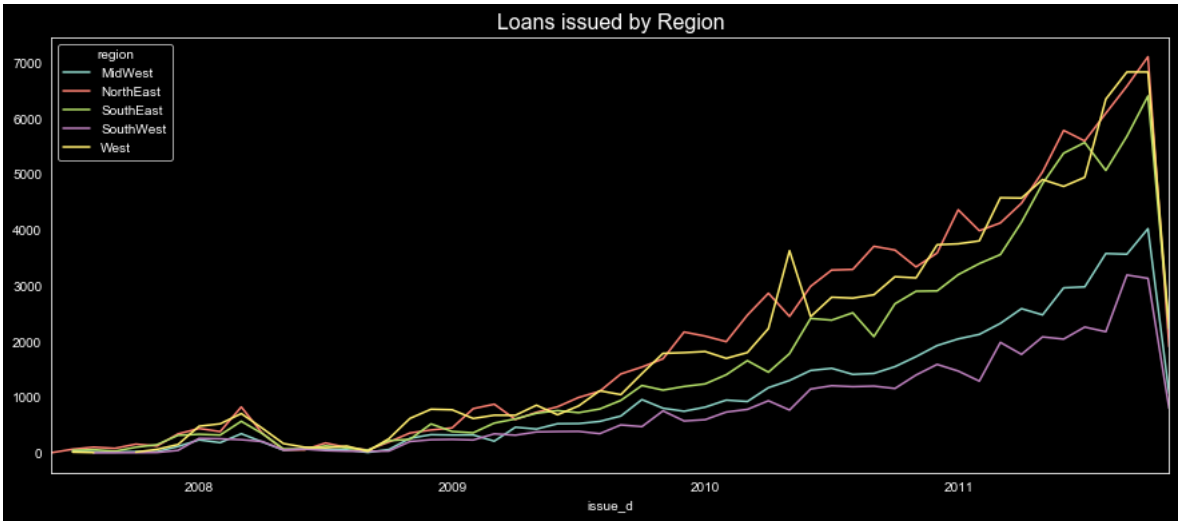



```
df['addr_state'].unique()

# Make a list with each of the regions by state.

west = ['CA', 'OR', 'UT','WA', 'CO', 'NV', 'AK', 'MT', 'HI', 'WY', 'ID']
south_west = ['AZ', 'TX', 'NM', 'OK']
south_east = ['GA', 'NC', 'VA', 'FL', 'KY', 'SC', 'LA', 'AL', 'WV', 'DC', 'AR', 'DE', 'MS', 'TN' ]
mid_west = ['IL', 'MO', 'MN', 'OH', 'WI', 'KS', 'MI', 'SD', 'IA', 'NE', 'IN', 'ND']
north_east = ['CT', 'NY', 'PA', 'NJ', 'RI','MA', 'MD', 'VT', 'NH', 'ME']



df['region'] = np.nan

def finding_regions(state):
    if state in west:
        return 'West'
    elif state in south_west:
        return 'SouthWest'
    elif state in south_east:
        return 'SouthEast'
    elif state in mid_west:
        return 'MidWest'
    elif state in north_east:
        return 'NorthEast'



df['region'] = df['addr_state'].apply(finding_regions)
```

```
# This code will take the current date and transform it into a year-month format
df['complete_date'] = pd.to_datetime(df['issue_d'])

group_dates = df.groupby(['complete_date', 'region'], as_index=False).sum()

group_dates['issue_d'] = [month.to_period('M') for
                          month in group_dates['complete_date']]

group_dates = group_dates.groupby(['issue_d', 'region'], as_index=False).sum()
group_dates = group_dates.groupby(['issue_d', 'region'], as_index=False).sum()
group_dates['loan_amount'] = group_dates['loan_amount']/1000


df_dates = pd.DataFrame(data=group_dates[['issue_d','region','loan_amount']])
```

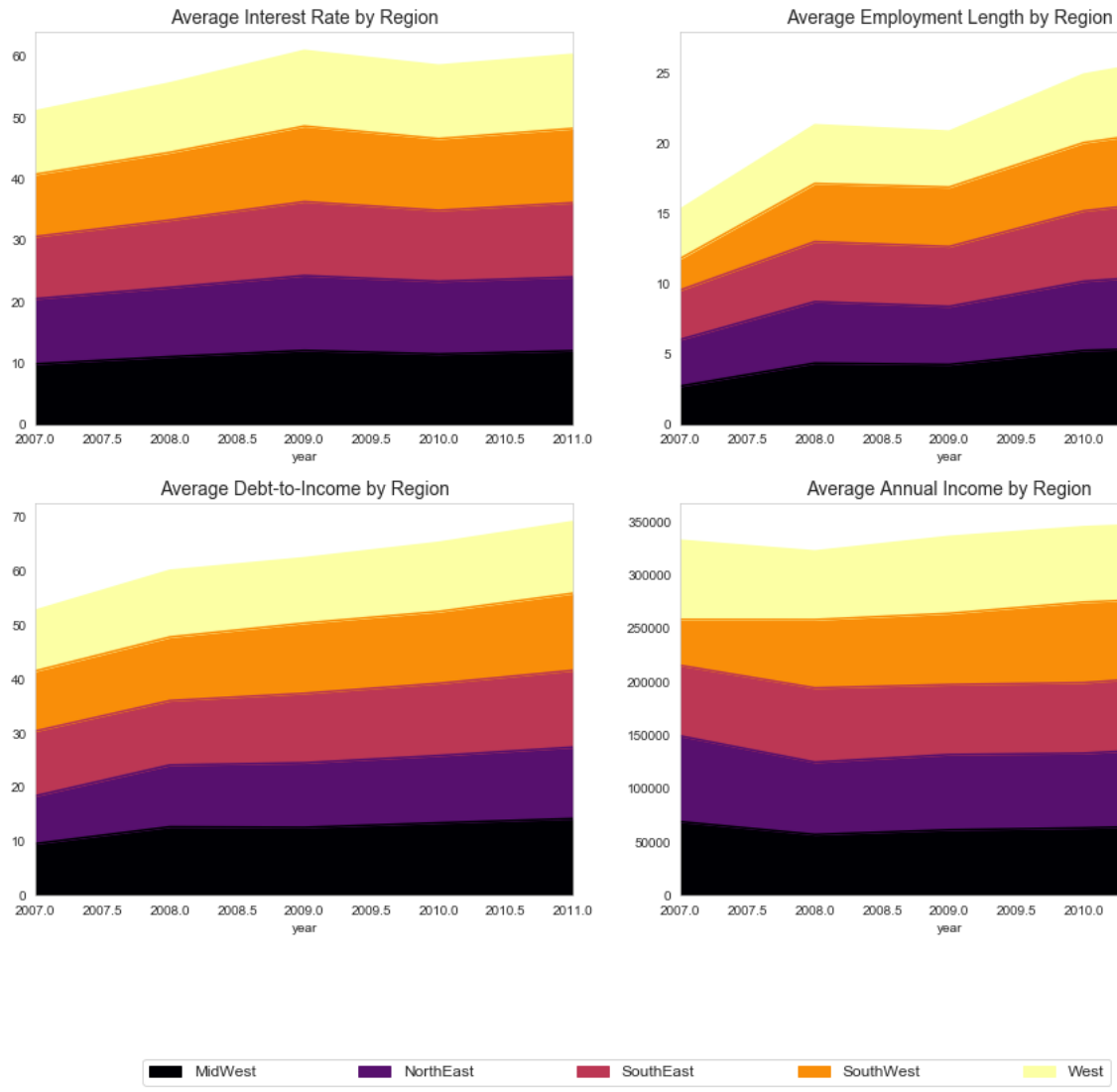
```
plt.style.use('dark_background')
cmap = plt.cm.Set3
```

```
by_issued_amount = df_dates.groupby(['issue_d', 'region']).loan_amount.sum()
by_issued_amount.unstack().plot(stacked=False, colormap=cmap, grid=False, legend=True, figsize=(15,6))

plt.title('Loans issued by Region', fontsize=16)
```




```
employment_length = ['10+ years', '< 1 year', '1 year', '3 years', '8 years', '9 years',
                     '4 years', '5 years', '6 years', '2 years', '7 years', 'n/a']

# Create a new column and convert emp_length to integers.

lst = [df]
df['emp_length_int'] = np.nan

for col in lst:
    col.loc[col['emp_length'] == '10+ years', "emp_length_int"] = 10
    col.loc[col['emp_length'] == '9 years', "emp_length_int"] = 9
    col.loc[col['emp_length'] == '8 years', "emp_length_int"] = 8
    col.loc[col['emp_length'] == '7 years', "emp_length_int"] = 7
    col.loc[col['emp_length'] == '6 years', "emp_length_int"] = 6
    col.loc[col['emp_length'] == '5 years', "emp_length_int"] = 5
    col.loc[col['emp_length'] == '4 years', "emp_length_int"] = 4
    col.loc[col['emp_length'] == '3 years', "emp_length_int"] = 3
    col.loc[col['emp_length'] == '2 years', "emp_length_int"] = 2
    col.loc[col['emp_length'] == '1 year', "emp_length_int"] = 1
    col.loc[col['emp_length'] == '< 1 year', "emp_length_int"] = 0.5
    col.loc[col['emp_length'] == 'n/a', "emp_length_int"] = 0
```

```
sns.set_style('whitegrid')

f, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2)
cmap = plt.cm.inferno

by_interest_rate = df.groupby(['year', 'region']).interest_rate.mean()
by_interest_rate.unstack().plot(kind='area', stacked=True, colormap=cmap, grid=False, legend=False, ax=ax1, figsize=
(16,12))
ax1.set_title('Average Interest Rate by Region', fontsize=14)


by_employment_length = df.groupby(['year', 'region']).emp_length_int.mean()
by_employment_length.unstack().plot(kind='area', stacked=True, colormap=cmap, grid=False, legend=False, ax=ax2,
figsize=(16,12))
ax2.set_title('Average Employment Length by Region', fontsize=14)
# plt.xlabel('Year of Issuance', fontsize=14)

by_dti = df.groupby(['year', 'region']).dti.mean()
by_dti.unstack().plot(kind='area', stacked=True, colormap=cmap, grid=False, legend=False, ax=ax3, figsize=(16,12))
ax3.set_title('Average Debt-to-Income by Region', fontsize=14)

by_income = df.groupby(['year', 'region']).annual_income.mean()
by_income.unstack().plot(kind='area', stacked=True, colormap=cmap, grid=False, ax=ax4, figsize=(16,12))
ax4.set_title('Average Annual Income by Region', fontsize=14)
ax4.legend(bbox_to_anchor=(-1.0, -0.5, 1.8, 0.1), loc=10,prop={'size':12},
           ncol=5, mode="expand", borderaxespad=0.)
```



The Business Perspective

Understanding the Operative Side of Business

Now we will have a closer look at the **operative side** of business by state. This will give us a clearer idea in which state we have a higher operating activity. This will allow us to ask further questions such as Why do we have a higher level of operating activity in this state? Could it be because of economic factors? or the risk level is low and returns are fairly decent? Let's explore!

What we need to know:

- We will focus on **three key metrics**: Loans issued by state (Total Sum), Average interest rates charged to customers and average annual income of all customers by state.
- The purpose of this analysis is to see states that give high returns at a descent risk.

Summary:

- **California, Texas, New York and Florida** are the states in which the highest amount of loans were issued.
- Interesting enough, all four states have a approximate **interest rate of 13%** which is at the same level of the average interest rate for all states (13.24%)
- California, Texas and New York are **all above the average annual income** (with the exclusion of Florida), this might give possible indication why most loans are issued in these states.

```
# Plotting by states

# Grouping by our metrics
# First Plotly Graph (We evaluate the operative side of the business)
by_loan_amount = df.groupby(['region','addr_state'], as_index=False).loan_amount.sum()
by_interest_rate = df.groupby(['region', 'addr_state'], as_index=False).interest_rate.mean()
by_income = df.groupby(['region', 'addr_state'], as_index=False).annual_income.mean()



# Take the values to a list for visualization purposes.
states = by_loan_amount['addr_state'].values.tolist()
```

```
average_loan_amounts = by_loan_amount['loan_amount'].values.tolist()
average_interest_rates = by_interest_rate['interest_rate'].values.tolist()
average_annual_income = by_income['annual_income'].values.tolist()


from collections import OrderedDict

# Figure Number 1 (Perspective for the Business Operations)
metrics_data = OrderedDict([('state_codes', states),
                            ('issued_loans', average_loan_amounts),
                            ('interest_rate', average_interest_rates),
                            ('annual_income', average_annual_income)])


metrics_df = pd.DataFrame.from_dict(metrics_data)
metrics_df = metrics_df.round(decimals=2)
metrics_df.head()



# Think of a way to add default rate
# Consider adding a few more metrics for the future
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	state_codes	issued_loans	interest_rate	annual_income
0	IA	56450	8.97	50599.20
1	IL	15555675	11.91	70547.21
2	IN	86225	10.80	35416.11
3	KS	2627800	11.71	63280.35
4	MI	7519150	11.98	65441.87

```
# Now it comes the part where we plot out plotly United States map
import chart_studio.plotly as py
import plotly.graph_objs as go


for col in metrics_df.columns:
    metrics_df[col] = metrics_df[col].astype(str)

scl = [[0.0, 'rgb(210, 241, 198)'],[0.2, 'rgb(188, 236, 169)'],[0.4, 'rgb(171, 235, 145)'],\
            [0.6, 'rgb(140, 227, 105)'],[0.8, 'rgb(105, 201, 67)'],[1.0, 'rgb(59, 159, 19)']]

metrics_df['text'] = metrics_df['state_codes'] + '<br>' +\
'Average loan interest rate: ' + metrics_df['interest_rate'] + '<br>'+\
'Average annual income: ' + metrics_df['annual_income']


data = [ dict(
        type='choropleth',
        colorscale = scl,
        autocolorscale = False,
        locations = metrics_df['state_codes'],
        z = metrics_df['issued_loans'],
        locationmode = 'USA-states',
        text = metrics_df['text'],
        marker = dict(
            line = dict (
                color = 'rgb(255,255,255)',
                width = 2
            ) ),
        colorbar = dict(
            title = "$s USD")
        ) ]
```

```
layout = dict(
    title = 'Lending Clubs Issued Loans <br> (A Perspective for the Business Operations)',
    geo = dict(
        scope = 'usa',
        projection=dict(type='albers usa'),
        showlakes = True,
        lakecolor = 'rgb(255, 255, 255)')
)

fig = dict(data=data, layout=layout)
iplot(fig, filename='d3-cloropleth-map')
```

Analysis by Income Category:

In this section we will create different **income categories** in order to detect important patters and go more into depth in our analysis.

What we need to know:

- **Low income category:** Borrowers that have an annual income lower or equal to 100,000 usd.
- **Medium income category:** Borrowers that have an annual income higher than 100,000 usd but lower or equal to 200,000 usd.
- **High income category:** Borrowers that have an annual income higher tha 200,000 usd.

Summary:

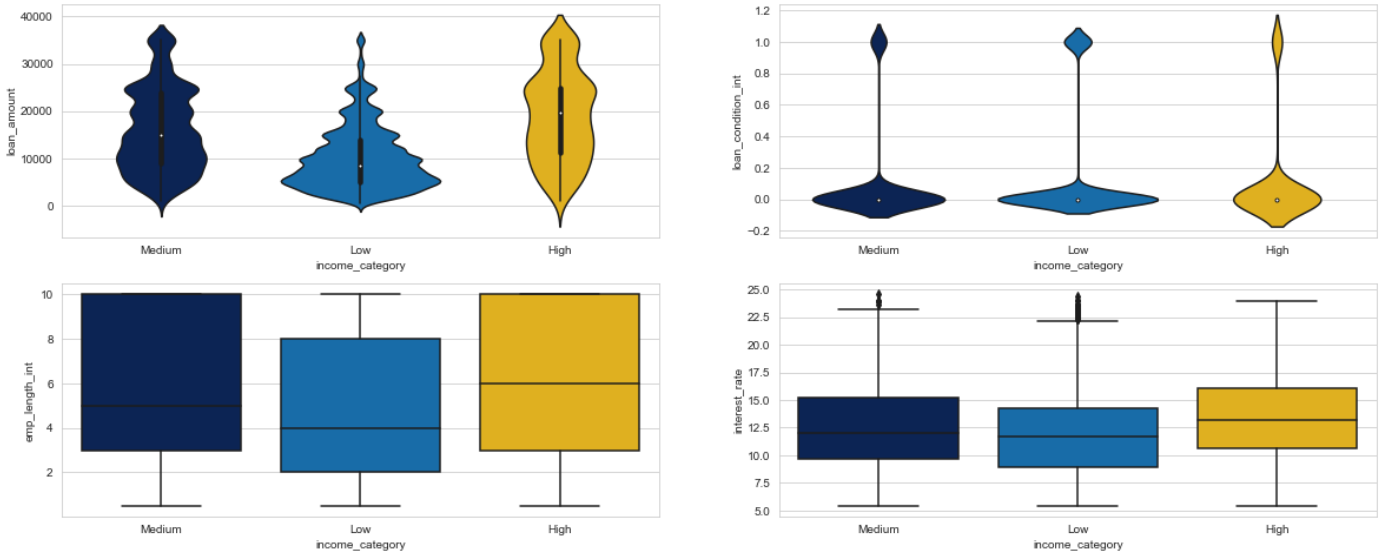
- Borrowers that made part of the **high income category** took higher loan amounts than people from **low** and **medium income categories.** Of course, people with higher annual incomes are more likely to pay loans with a higher amount. (First row to the left of the subplots)
- Loans that were borrowed by the **Low income category** had a slightly higher change of becoming a bad loan. (First row to the right of the subplots)
- Borrowers with **High** and **Medium** annual incomes had a longer employment length than people with lower incomes.(Second row to the left of the subplots)
- Borrowers with a lower income had on average **higher interest rates** while people with a higher annual income had **lower interest rates** on their loans. (Second row to the right of the subplots)

```
# Let's create categories for annual_income since most of the bad loans are located below 100k

df['income_category'] = np.nan
lst = [df]

for col in lst:
    col.loc[col['annual_income'] <= 100000, 'income_category'] = 'Low'
    col.loc[(col['annual_income'] > 100000) & (col['annual_income'] <= 200000), 'income_category'] = 'Medium'
    col.loc[col['annual_income'] > 200000, 'income_category'] = 'High'
```

```
# Let's transform the column loan_condition into integrers.

lst = [df]
df['loan_condition_int'] = np.nan

for col in lst:
    col.loc[df['loan_status'] == 'Fully Paid', 'loan_condition_int'] = 0 # Negative (Bad Loan)
    col.loc[df['loan_status'] == 'Charged Off', 'loan_condition_int'] = 1 # Positive (Good Loan)

# Convert from float to int the column (This is our label)
df['loan_condition_int'] = df['loan_condition_int'].astype(int)
```

```
fig, ((ax1, ax2), (ax3, ax4))= plt.subplots(nrows=2, ncols=2, figsize=(20,8))

# Change the Palette types tomorrow!
wang=['#002060','#0070C0','#FFC000']
sns.violinplot(x="income_category", y="loan_amount", data=df, palette=wang, ax=ax1 )
sns.violinplot(x="income_category", y="loan_condition_int", data=df, palette=wang, ax=ax2)
sns.boxplot(x="income_category", y="emp_length_int", data=df, palette=wang, ax=ax3)
sns.boxplot(x="income_category", y="interest_rate", data=df, palette=wang, ax=ax4)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2792cfa6b88>
```



Assesing Risks

Understanding the Risky side of Business

Although the **operative side of business** is important, we have to also analyze the level of risk in each state. Credit scores are important metrics to analyze the level of risk of an individual customer. However, there are also other important metrics to somehow estimate the level of risk of other states.

What we need to know:

- **Debt-to-income** is an important metric since it says approximately the level of debt of each individual consumer with respect to its total income.
- The **average length of employment** tells us a better story about the labor market in each state which is helpful to assess the levelof risk.

Summary:

- **IOWA** has the highest level of default ratio neverthless, the amount of loans issued in that state is **too low**. (Number of Bad loans is equal to 3)
- California and Texas seem to have the lowest risk and the highest possible return for investors. However, I will look more deeply into these states and create other metrics analyze the level of risk for each state.

Note: I will be updating these section sooner or later (Stay in touch!)

```
by_condition = df.groupby('addr_state')['loan_status'].value_counts()/ df.groupby('addr_state')
['loan_status'].count()
by_emp_length = df.groupby(['region', 'addr_state'],
as_index=False).emp_length_int.mean().sort_values(by="addr_state")

loan_condition_bystate = pd.crosstab(df['addr_state'], df['loan_status'] )

cross_condition = pd.crosstab(df["addr_state"], df["loan_status"])
# Percentage of condition of loan
percentage_loan_contributor = pd.crosstab(df['addr_state'], df['loan_status']).apply(lambda x: x/x.sum() * 100)
condition_ratio = cross_condition["Charged Off"]/cross_condition["Fully Paid"]
by_dti = df.groupby(['region', 'addr_state'], as_index=False).dti.mean()
state_codes = sorted(states)


# Take to a list
default_ratio = condition_ratio.values.tolist()
average_dti = by_dti['dti'].values.tolist()
average_emp_length = by_emp_length["emp_length_int"].values.tolist()
number_of_badloans = loan_condition_bystate['Charged Off'].values.tolist()
percentage_ofall_badloans = percentage_loan_contributor['Charged Off'].values.tolist()


# Figure Number 2
risk_data = OrderedDict([('state_codes', state_codes),
                         ('default_ratio', default_ratio),
                         ('badloans_amount', number_of_badloans),
                         ('percentage_of_badloans', percentage_ofall_badloans),
                         ('average_dti', average_dti),
                         ('average_emp_length', average_emp_length)])


# Figure 2 Dataframe
```

```
risk_df = pd.DataFrame.from_dict(risk_data)
risk_df = risk_df.round(decimals=3)
risk_df.head()
```

	state_codes	default_ratio	badloans_amount	percentage_of_badloans	average_dti	average_emp_length
0	AK	0.208	11	0.221	12.910	5.444
1	AL	0.129	47	0.942	13.087	5.322
2	AR	0.133	26	0.521	15.971	5.851
3	AZ	0.165	113	2.266	13.980	4.783
4	CA	0.184	990	19.852	13.435	4.879

```
# Now it comes the part where we plot out plotly United States map
import chart_studio.plotly as py
import plotly.graph_objs as go


for col in risk_df.columns:
    risk_df[col] = risk_df[col].astype(str)

scl = [[0.0, 'rgb(202, 202, 202)'],[0.2, 'rgb(253, 205, 200)'],[0.4, 'rgb(252, 169, 161)'],\
            [0.6, 'rgb(247, 121, 108  )'],[0.8, 'rgb(232, 70, 54)'],[1.0, 'rgb(212, 31, 13)']]

risk_df['text'] = risk_df['state_codes'] + '<br>' +\
'Number of Bad Loans: ' + risk_df['badloans_amount'] + '<br>' + \
'Percentage of all Bad Loans: ' + risk_df['percentage_of_badloans'] + '%' +  '<br>' + \
'Average Debt-to-Income Ratio: ' + risk_df['average_dti'] + '<br>'+\
'Average Length of Employment: ' + risk_df['average_emp_length']


data = [ dict(
        type='choropleth',
        colorscale = scl,
        autocolorscale = False,
        locations = risk_df['state_codes'],
        z = risk_df['default_ratio'],
        locationmode = 'USA-states',
        text = risk_df['text'],
        marker = dict(
            line = dict (
                color = 'rgb(255,255,255)',
                width = 2
            ) ),
        colorbar = dict(
            title = "%")
        ) ]


layout = dict(
    title = 'Lending Clubs Default Rates <br> (Analyzing Risks)',
    geo = dict(
        scope = 'usa',
        projection=dict(type='albers usa'),
        showlakes = True,
        lakecolor = 'rgb(255, 255, 255)')
)

fig = dict(data=data, layout=layout)
iplot(fig, filename='d3-cloropleth-map')
```

The Importance of Credit Scores:

Credit scores are important metrics for assesing the overall level of risk. In this section we will analyze the level of risk as a whole and how many loans were bad loans by the type of grade received in the credit score of the customer.

What we need to know:

- The lower the grade of the credit score, the higher the risk for investors.
- There are different factors that influence on the level of risk of the loan.

Summary:

- The scores that has a lower grade received a larger amounts of loans (which might had contributed to a higher level of risk).

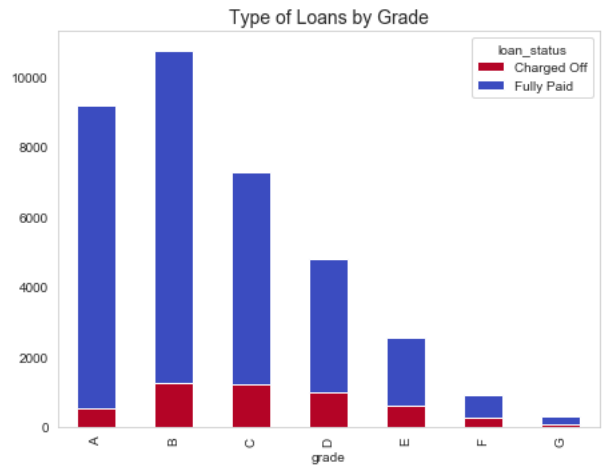
- Logically, the **lower the grade the higher the interest** the customer had to pay back to investors.
- Interstingly, customers with a **grade** of "C" were more likely to default on the loan

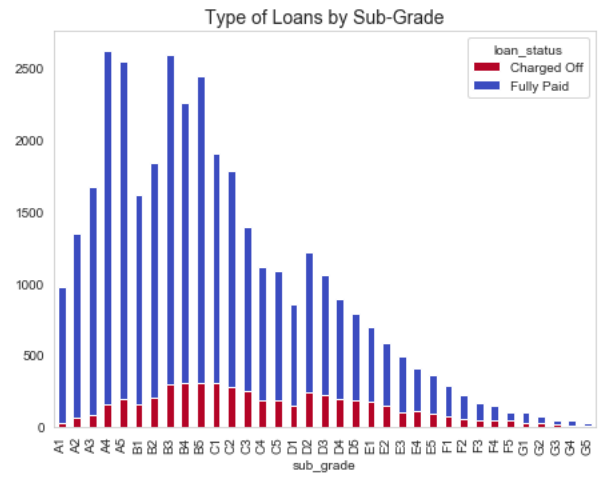
```
# Let's visualize how many loans were issued by creditscore
f, ((ax1, ax2)) = plt.subplots(1, 2)
cmap = plt.cm.coolwarm

by_credit_score = df.groupby(['year', 'grade']).loan_amount.mean()
by_credit_score.unstack().plot(legend=False, ax=ax1, figsize=(14, 4), colormap=cmap)
ax1.set_title('Loans issued by Credit Score', fontsize=14)


by_inc = df.groupby(['year', 'grade']).interest_rate.mean()
by_inc.unstack().plot(ax=ax2, figsize=(14, 4), colormap=cmap)
ax2.set_title('Interest Rates by Credit Score', fontsize=14)

ax2.legend(bbox_to_anchor=(-1.0, -0.3, 1.7, 0.1), loc=5, prop={'size':12},
           ncol=7, mode="expand", borderaxespad=0.)
```

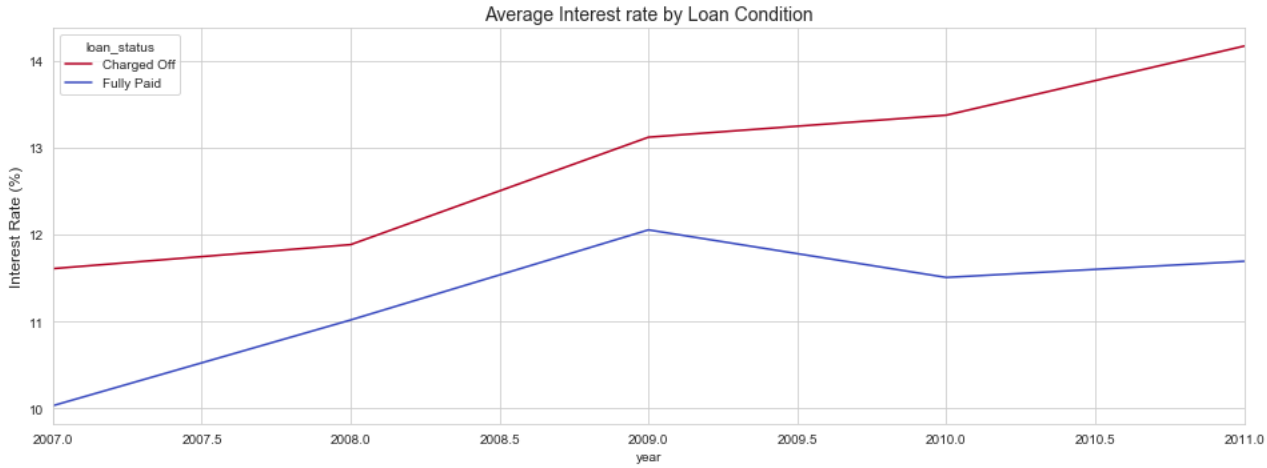
```
<matplotlib.legend.Legend at 0x2791c790b88>
```


```
fig = plt.figure(figsize=(16,12))

ax1 = fig.add_subplot(221)
ax2 = fig.add_subplot(222)
ax3 = fig.add_subplot(212)

cmap = plt.cm.coolwarm_r

loans_by_region = df.groupby(['grade', 'loan_status']).size()
loans_by_region.unstack().plot(kind='bar', stacked=True, colormap=cmap, ax=ax1, grid=False)
ax1.set_title('Type of Loans by Grade', fontsize=14)


loans_by_grade = df.groupby(['sub_grade', 'loan_status']).size()
loans_by_grade.unstack().plot(kind='bar', stacked=True, colormap=cmap, ax=ax2, grid=False)
ax2.set_title('Type of Loans by Sub-Grade', fontsize=14)

by_interest = df.groupby(['year', 'loan_status']).interest_rate.mean()
by_interest.unstack().plot(ax=ax3, colormap=cmap)
ax3.set_title('Average Interest rate by Loan Condition', fontsize=14)
ax3.set_ylabel('Interest Rate (%)', fontsize=12)
```

```
Text(0, 0.5, 'Interest Rate (%)')
```







Defaulted Loans and Level of Risk:

From all the bad loans the one we are most interested about are the loans that are defaulted. Therefore, in this section we will implement an in-depth analysis of these types of Loans and see if we can gain any insight as to which features have a high correlation with the loan being defaulted.

Main Aim:

- Determine patters that will allow us to understand somehow factors that contribute to a loan being **defaulted**

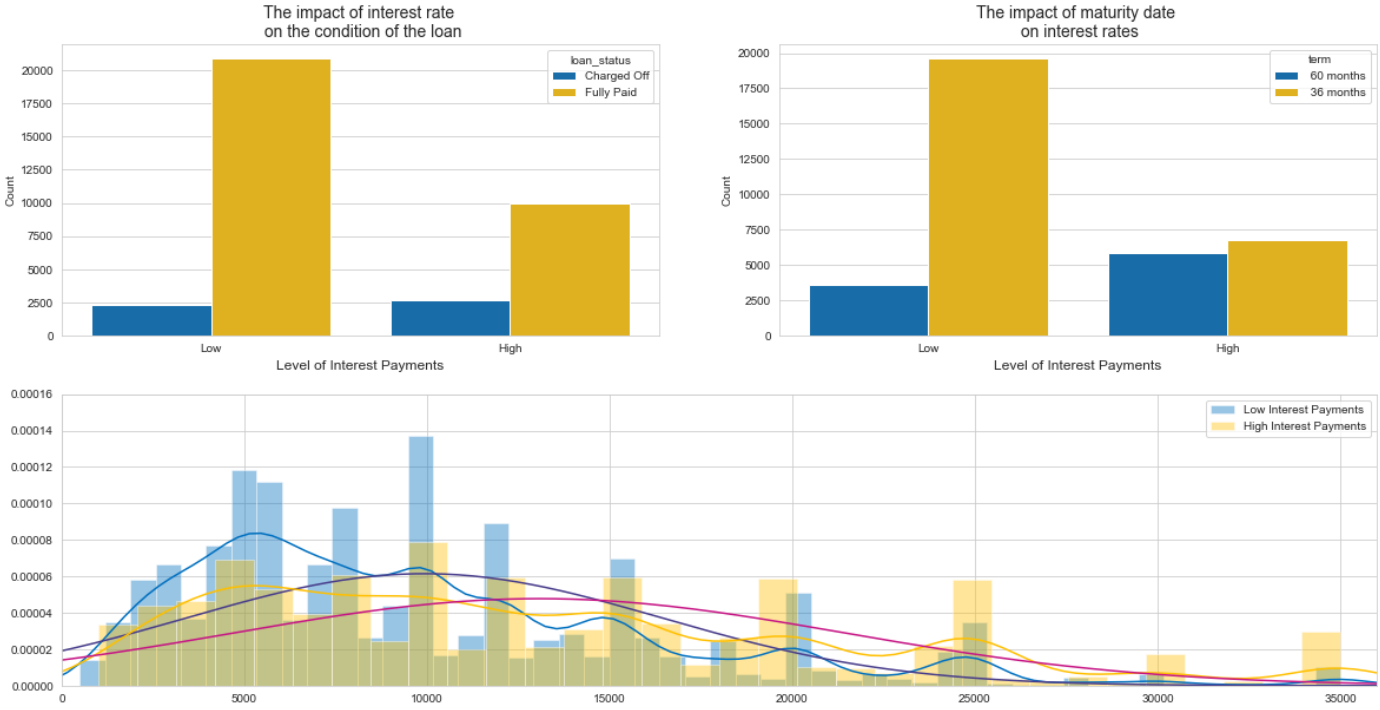
Summary:

- In the last year recorded, the **Midwest** and **SouthEast** regions had the most defaults.
- Loans that have a **high interest rate**(above 13.23%) are more likely to become a **bad loan** .
- Loans that have a longer **maturity date (60 months)** are more likely to be a bad loan.

```
df['interest_rate'].describe()
# Average interest is 13.26% Anything above this will be considered of high risk let's see if this is true.
df['interest_payments'] = np.nan
lst = [df]

for col in lst:
    col.loc[col['interest_rate'] <= 13.23, 'interest_payments'] = 'Low'
    col.loc[col['interest_rate'] > 13.23, 'interest_payments'] = 'High'
```

```
from scipy.stats import norm

plt.figure(figsize=(20,10))

palette = ["#0070C0", "#FFC000"]
plt.subplot(221)
ax = sns.countplot(x='interest_payments', data=df,
                   palette=palette, hue='loan_status')

ax.set_title('The impact of interest rate \n on the condition of the loan', fontsize=14)
ax.set_xlabel('Level of Interest Payments', fontsize=12)
ax.set_ylabel('Count')

plt.subplot(222)
```

```
ax1 = sns.countplot(x='interest_payments', data=df,
                    palette=palette, hue='term')

ax1.set_title('The impact of maturity date \n on interest rates', fontsize=14)
ax1.set_xlabel('Level of Interest Payments', fontsize=12)
ax1.set_ylabel('Count')


plt.subplot(212)
low = df['loan_amount'].loc[df['interest_payments'] == 'Low'].values
high = df['loan_amount'].loc[df['interest_payments'] == 'High'].values


ax2= sns.distplot(low, color='#0070C0', label='Low Interest Payments', fit=norm, fit_kws={"color":"#483d8b"}) # Dark
Blue Norm Color
ax3= sns.distplot(high,color='#FFC000', label='High Interest Payments',fit=norm, fit_kws={"color":"#c71585"}) #  Red
Norm Color
plt.axis([0, 36000, 0, 0.00016])
plt.legend()


plt.show()
```



Condition of Loans and Purpose:

In this section we will go into depth regarding the **reasons for clients to apply for a loan.** Our main aim is to see if there are purposes that contribute to a **"higher"** risk whether the loan will be repaid or not.

Summary:

- **Bad Loans Count:** People that apply for educational and small business purposed tend to have a higher risk of being a bad loan. (% wise)
- **Most frequent Purpose:** The reason that clients applied the most for a loan was to consolidate debt.
- **Less frequent purpose:** Clients applied less for educational purposes for all three income categories.
- **Interest Rates:** In all reasons for application except (medical, small business and credi card), the low income category has a higher interest rate. Something that could possibly explain this is the amount of capital that is needed from other income categories that might explain why the low income categories interest rate for these puposes are lower.
- **Bad/Good Ratio:** Except for educational purposes (we see a spike in high income this is due to the reasons that only two loans were issued and one was a bad loan which caused this ratio to spike to 50%.), but we can see that in all other purposed the bad good ratio is lower the higher your income category.

```
df['purpose'].value_counts()

# Education, renewable energy, wedding are the purposed that contains highest bad loans percent wise.

purpose_condition = round(pd.crosstab(df['loan_status'], df['purpose']).apply(lambda x: x/x.sum() * 100), 2)

purpose_bad_loans = purpose_condition.values[0].tolist()
purpose_good_loans = purpose_condition.values[1].tolist()
purpose = purpose_condition.columns
```

```
bad_plot = go.Bar(
    x=purpose,
    y=purpose_bad_loans,
    name = ' Charged Off',
    text='%',
    marker=dict(
        color='#0070C0',
        line = dict(
            color='#0070C0',
            width=2
        )
    )
)

good_plot = go.Bar(
    x=purpose,
    y=purpose_good_loans,
    name='Fully Paid',
    text='%',
    marker=dict(
        color='#FFC000',
        line = dict(
            color='#FFC000',
            width=2
        )
    )
)


data = [bad_plot, good_plot]

layout = go.Layout(
    title='Condition of Loan by Purpose',
    xaxis=dict(
        title=''
    ),
    yaxis=dict(
        title='% of the Loan',
    ),
    # paper_bgcolor='#FFF8DC',
    # plot_bgcolor='#FFF8DC',
    showlegend=True
)

fig = dict(data=data, layout=layout)
iplot(fig, filename='condition_purposes')
```

```
group_income_purpose = df.groupby(['income_category', 'purpose'], as_index=False).interest_rate.mean()
group_dti_purpose = df.groupby(['income_category', 'purpose'], as_index=False).loan_amount.mean()
loan_a = group_dti_purpose['loan_amount'].values



# High Car 10.32 15669
new_groupby = group_income_purpose.assign(total_loan_amount=loan_a)
sort_group_income_purpose = new_groupby.sort_values(by="income_category", ascending=True)
```

```
loan_count = df.groupby(['income_category', 'purpose'])['loan_status'].apply(lambda x: x.value_counts())
d={"loan_c": loan_count}
loan_c_df = pd.DataFrame(data=d).reset_index()
loan_c_df = loan_c_df.rename(columns={"level_2": "loan_status"})


# Good loans & Bad Loans
good_loans = loan_c_df.loc[loan_c_df['loan_status'] == "Fully Paid"].sort_values(by="income_category",
ascending=True)
bad_loans = loan_c_df.loc[loan_c_df['loan_status'] == "Charged Off"].sort_values(by="income_category",
ascending=True)

bad_loans.loc[1]={'income_category':'High','purpose':'car','loan_status':'Charged Off','loan_c':0}
bad_loans.loc[3]={'income_category':'High','purpose':'medical','loan_status':'Charged Off','loan_c':0}

sort_group_income_purpose['good_loans_count'] = good_loans['loan_c'].values
sort_group_income_purpose['bad_loans_count'] = bad_loans['loan_c'].values
```

```
sort_group_income_purpose['total_loans_issued'] = (good_loans['loan_c'].values + bad_loans['loan_c'].values)
sort_group_income_purpose['bad/good ratio (%)'] = np.around(bad_loans['loan_c'].values / (bad_loans['loan_c'].values
+ good_loans['loan_c'].values), 4) * 100
final_df = sort_group_income_purpose.sort_values(by='income_category', ascending=True)
final_df.style.background_gradient('coolwarm')
```

```
final_df = final_df.sort_values(by="purpose", ascending=False)
```

```
# Work on a plot to explain better the correlations between the different columns in final_df dataframe.
# We will do a Subplot in Plotly with

import chart_studio.plotly as py
import plotly.graph_objs as go

# Labels
purpose_labels = df['purpose'].unique()

# Average Interest Rate Dot Plots # 1st Subplot
high_income = final_df['interest_rate'].loc[final_df['income_category'] == 'High'].values.tolist()
medium_income = final_df['interest_rate'].loc[final_df['income_category'] == 'Medium'].values.tolist()
low_income = final_df['interest_rate'].loc[final_df['income_category'] == 'Low'].values.tolist()

high_lst = ['%.2f' % val for val in high_income]
med_lst = ['%.2f' % val for val in medium_income]
low_lst = ['%.2f' % val for val in low_income]



trace1 = {"x": high_lst,
          "y": purpose_labels,
          "marker": {"color": "#0040FF", "size": 16},
          "mode": "markers",
          "name": "High Income",
          "type": "scatter"
}

trace2 = {"x": med_lst,
          "y": purpose_labels,
          "marker": {"color": "#FE9A2E", "size": 16},
          "mode": "markers",
          "name": "Medium Income",
          "type": "scatter",
}

trace3 = {"x": low_lst,
          "y": purpose_labels,
          "marker": {"color": "#FE2E2E", "size": 16},
          "mode": "markers",
          "name": "Low Income",
          "type": "scatter",
}




data = [trace1, trace2, trace3]
layout = {"title": "Average Purpose Interest Rate <br> <i> by Income Category </i> ",
          "xaxis": {"title": "Average Interest Rate", },
          "yaxis": {"title": ""}}

fig = go.Figure(data=data, layout=layout)
iplot(fig)
```

Statistic Analysis

```
numeric_df = df.select_dtypes(exclude=["object"])
categorical_df = df.select_dtypes(["object"])
```

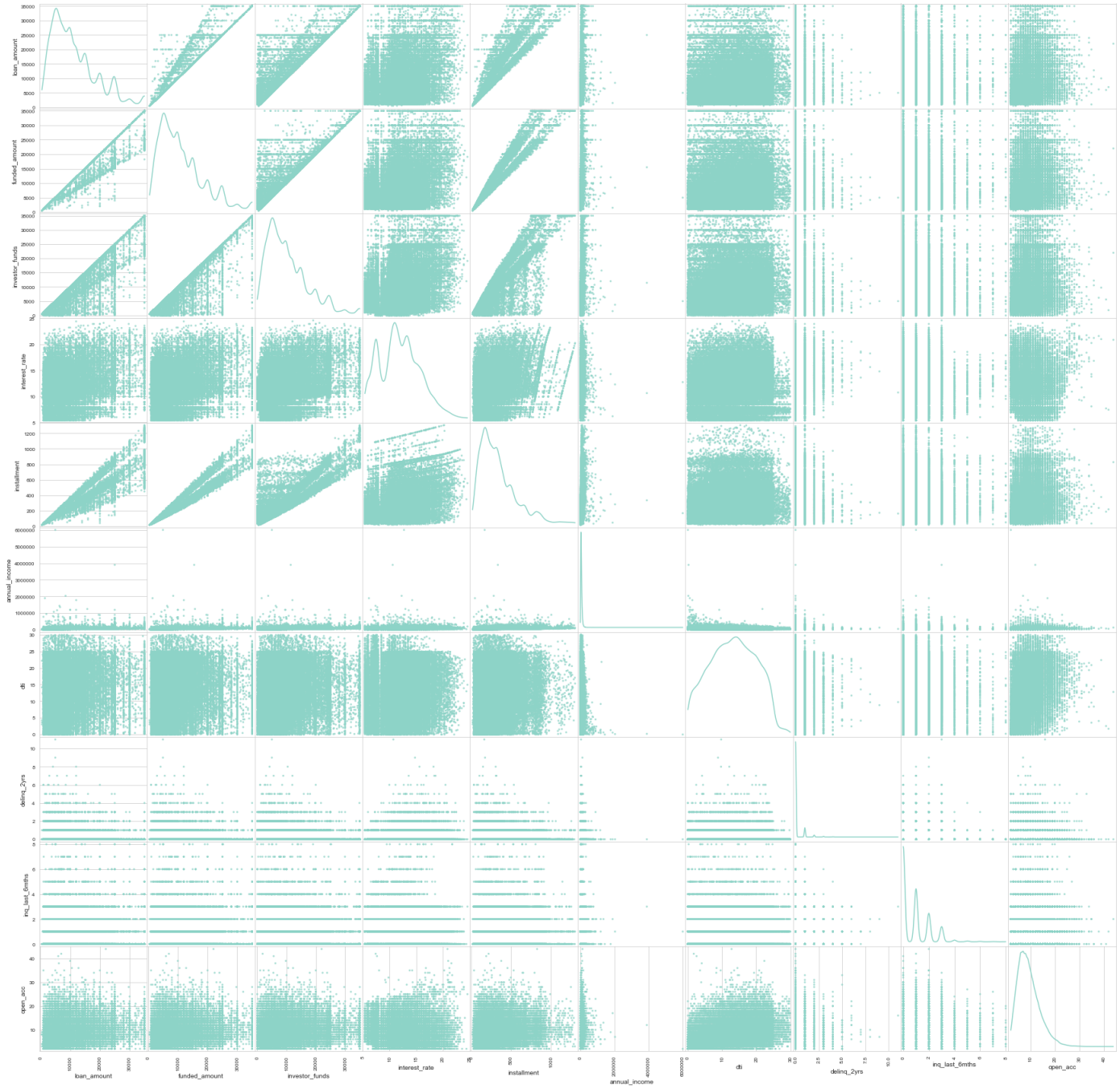
```
numeric_df.describe()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	loan_amount	funded_amount	investor_funds	interest_rate	installment	annual_income	dti	delinq_2yrs	inq_last_6r
count	35808.000000	35808.000000	35808.000000	35808.000000	35808.000000	3.580800e+04	35808.000000	35808.000000	35808.0000
mean	11034.324732	10749.145442	10143.715379	11.924765	319.161034	6.927846e+04	13.194689	0.150721	0.870643
std	7357.608391	7070.498203	6990.952813	3.663124	206.866440	6.580233e+04	6.698770	0.499995	1.077923
min	500.000000	500.000000	0.000000	5.420000	15.690000	4.000000e+03	0.000000	0.000000	0.000000
25%	5118.750000	5000.000000	4996.157009	9.070000	164.010000	4.050000e+04	8.020000	0.000000	0.000000
50%	9600.000000	9500.000000	8500.000000	11.710000	274.450000	5.944900e+04	13.290000	0.000000	1.000000
75%	15000.000000	15000.000000	14000.000000	14.350000	421.220000	8.300000e+04	18.490000	0.000000	1.000000
max	35000.000000	35000.000000	35000.000000	24.590000	1305.190000	6.000000e+06	29.990000	11.000000	8.000000

```
def plotScatterMatrix(df, plotSize, textSize):
    df = df.select_dtypes(include =[np.number]) # keep only numerical columns
    # Remove rows and columns that would lead to df being singular
    df = df.dropna('columns')
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are more than 1 unique values
    columnNames = list(df)
    if len(columnNames) > 10: # reduce the number of columns for matrix inversion of kernel density plots
        columnNames = columnNames[:10]
    df = df[columnNames]
    ax = pd.plotting.scatter_matrix(df, alpha=0.75, figsize=[plotSize, plotSize], diagonal='kde')
    plt.suptitle('Scatter and Density Plot')
    plt.show()
```

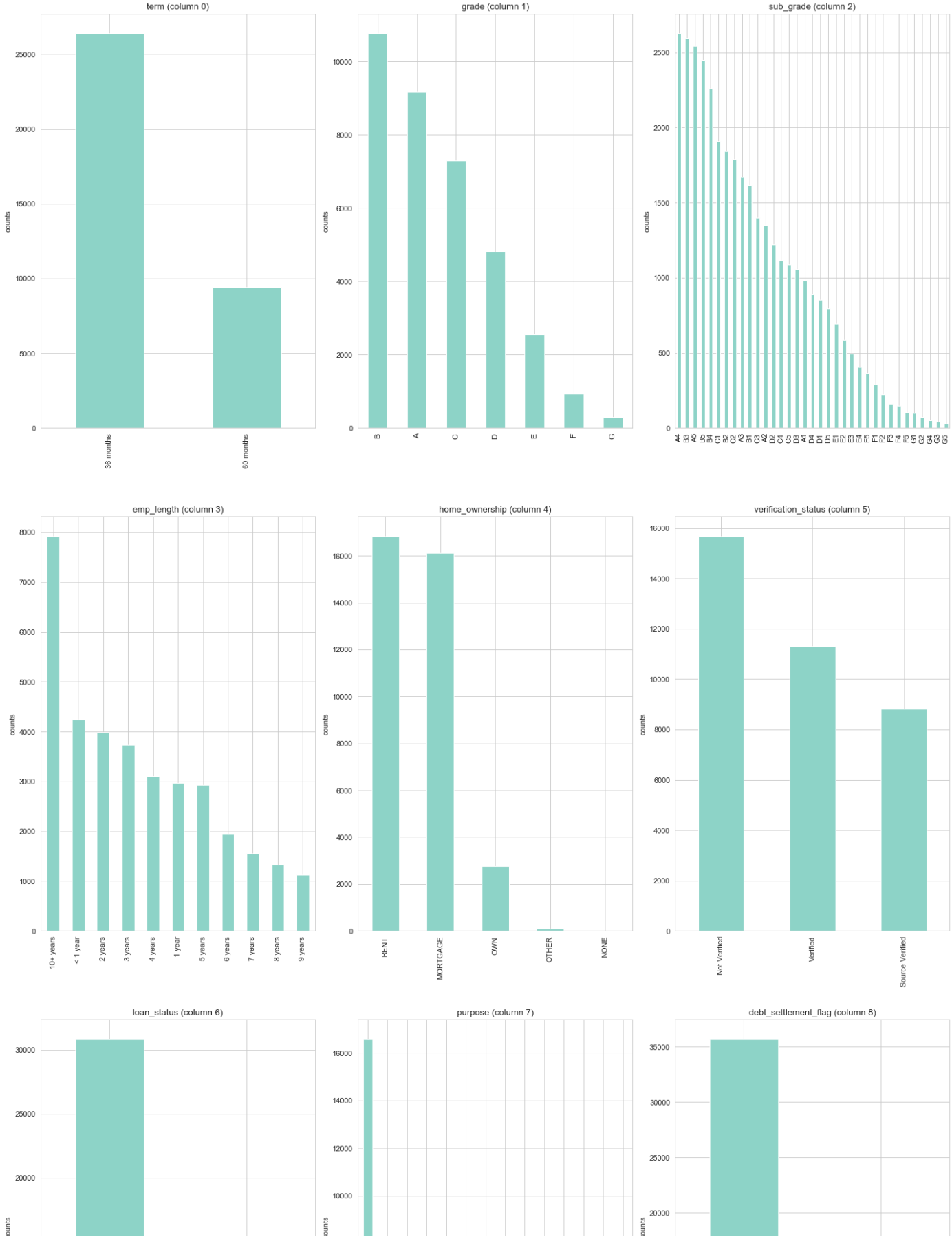
```
plotScatterMatrix(numeric_df, 30, 10)
```

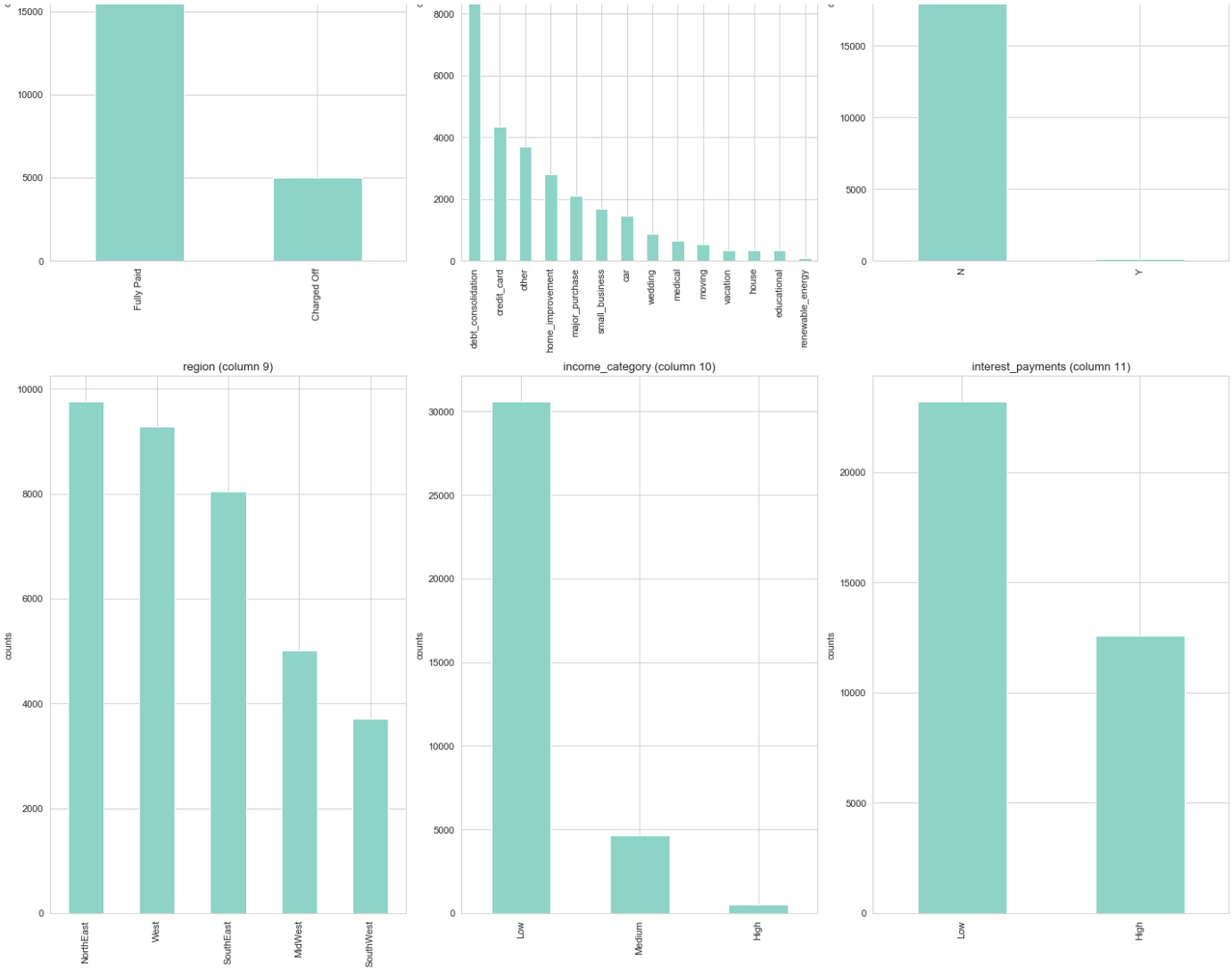



```
def plotPerColumnDistribution(df, nGraphShown, nGraphPerRow):
    nunique = df.nunique()
    df = df[[col for col in df if nunique[col] > 1 and nunique[col] < 50]] # For displaying purposes, pick columns
that have between 1 and 50 unique values
    nRow, nCol = df.shape
    columnNames = list(df)
    nGraphRow = (nCol + nGraphPerRow - 1) / nGraphPerRow
    plt.figure(num = None, figsize = (6 * nGraphPerRow, 8 * nGraphRow), dpi = 80, facecolor = 'w', edgecolor = 'k')
    for i in range(min(nCol, nGraphShown)):
        plt.subplot(nGraphRow, nGraphPerRow, i + 1)
        columnDf = df.iloc[:, i]
        if (not np.issubdtype(type(columnDf.iloc[0]), np.number)):
            valueCounts = columnDf.value_counts()
            valueCounts.plot.bar()
        else:
            columnDf.hist()
        plt.ylabel('counts')
        plt.xticks(rotation = 90)
```

```
            plt.title(f'{columnNames[i]} (column {i})')
        plt.tight_layout(pad = 1.0, w_pad = 1.0, h_pad = 1.0)
        plt.show()
```

```
plotPerColumnDistribution(categorical_df, 12, 3)
```






```
def plot_correlation_map( df ):
    corr = df.corr()
    _ , ax = plt.subplots( figsize =( 30 , 30 ) )
    cmap = sns.diverging_palette(255 , 20 , as_cmap = True )
    _ = sns.heatmap(
        corr,
        cmap = cmap,
        square=True,
        cbar_kws={ 'shrink' : 0.9 },
        ax=ax,
        annot = True,
        annot_kws = { 'fontsize' : 12 }
    )
```

```
for col in numeric_df.columns:
    print("'",col,"' ,")
```

```
' loan_amount ' ,
' funded_amount ' ,
' investor_funds ' ,
' interest_rate ' ,
' installment ' ,
' annual_income ' ,
' issue_d ' ,
' dti ' ,
' delinq_2yrs ' ,
' earliest_cr_line ' ,
' inq_last_6mths ' ,
' mths_since_last_delinq ' ,
' mths_since_last_record ' ,
```

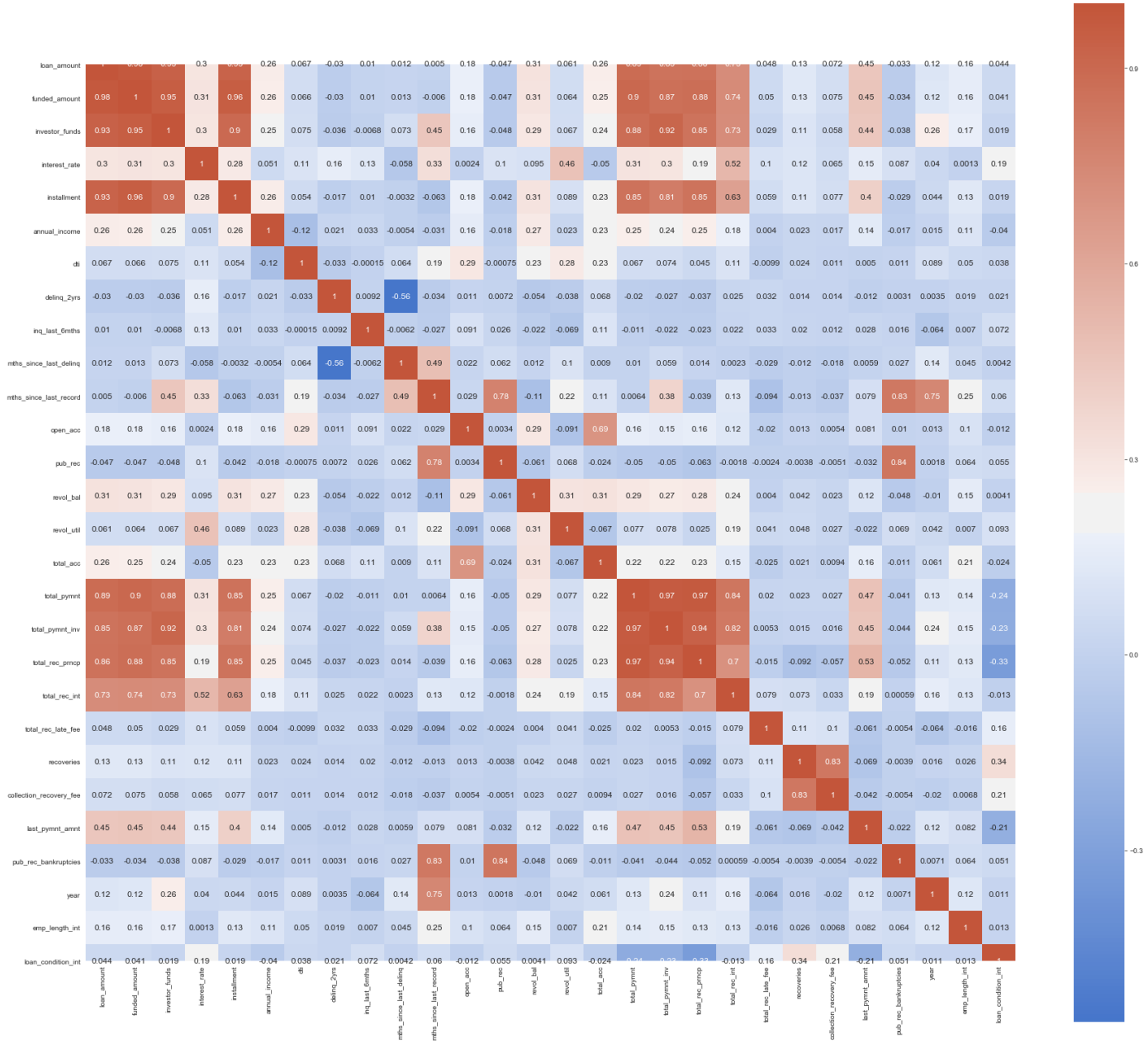
```
' open_acc ' ,
' pub_rec ' ,
' revol_bal ' ,
' revol_util ' ,
' total_acc ' ,
' total_pymnt ' ,
' total_pymnt_inv ' ,
' total_rec_prncp ' ,
' total_rec_int ' ,
' total_rec_late_fee ' ,
' recoveries ' ,
' collection_recovery_fee ' ,
' last_pymnt_d ' ,
' last_pymnt_amnt ' ,
' last_credit_pull_d ' ,
' pub_rec_bankruptcies ' ,
' year ' ,
' complete_date ' ,
' emp_length_int ' ,
' loan_condition_int ' ,
```

```
statistical = numeric_df[numeric_df.columns]
plot_correlation_map(statistical)
statistical.corr()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	loan_amount	funded_amount	investor_funds	interest_rate	installment	annual_income	dti	delinq_2yrs	inq_
loan_amount	1.000000	0.980434	0.933444	0.301424	0.928499	0.264428	0.066529	-0.030041	0.01
funded_amount	0.980434	1.000000	0.952598	0.305973	0.955960	0.260369	0.066357	-0.030085	0.00
investor_funds	0.933444	0.952598	1.000000	0.297791	0.897896	0.247671	0.074630	-0.036306	-0.0
interest_rate	0.301424	0.305973	0.297791	1.000000	0.278219	0.051345	0.107563	0.163789	0.13
installment	0.928499	0.955960	0.897896	0.278219	1.000000	0.264100	0.054384	-0.016771	0.01
annual_income	0.264428	0.260369	0.247671	0.051345	0.264100	1.000000	-0.117232	0.020701	0.03
dti	0.066529	0.066357	0.074630	0.107563	0.054384	-0.117232	1.000000	-0.032597	-0.0
delinq_2yrs	-0.030041	-0.030085	-0.036306	0.163789	-0.016771	0.020701	-0.032597	1.000000	0.00
inq_last_6mths	0.010325	0.009963	-0.006825	0.130162	0.010232	0.033141	-0.000146	0.009231	1.00
mths_since_last_delinq	0.012119	0.012961	0.072970	-0.057688	-0.003177	-0.005356	0.063537	-0.563739	-0.0
mths_since_last_record	0.004995	-0.006004	0.447066	0.332577	-0.062938	-0.030525	0.192088	-0.033539	-0.02
open_acc	0.180182	0.178711	0.164880	0.002396	0.175384	0.155530	0.289843	0.011029	0.09
pub_rec	-0.046970	-0.047418	-0.047756	0.104168	-0.041566	-0.018467	-0.000752	0.007154	0.02
revol_bal	0.314839	0.306916	0.285549	0.095377	0.309338	0.274227	0.228044	-0.054042	-0.02
revol_util	0.060739	0.064032	0.066650	0.463987	0.089422	0.022838	0.277195	-0.038380	-0.06
total_acc	0.257395	0.251909	0.244140	-0.049723	0.231604	0.229987	0.232628	0.067968	0.11
total_pymnt	0.886462	0.903971	0.879725	0.307296	0.854306	0.250525	0.067157	-0.020442	-0.01
total_pymnt_inv	0.849459	0.866635	0.915186	0.300792	0.809016	0.239487	0.073765	-0.026751	-0.02
total_rec_prncp	0.857470	0.876801	0.850208	0.192918	0.852170	0.252621	0.045056	-0.036574	-0.02
total_rec_int	0.729093	0.737302	0.729550	0.523498	0.633365	0.180982	0.106948	0.025217	0.02
total_rec_late_fee	0.047753	0.050266	0.029319	0.100385	0.059475	0.004042	-0.009856	0.031751	0.03
recoveries	0.129316	0.130059	0.113805	0.118069	0.114798	0.023204	0.023885	0.014483	0.02

	loan_amount	funded_amount	investor_funds	interest_rate	installment	annual_income	dti	delinq_2yrs	inq_
collection_recovery_fee	0.072374	0.074726	0.058223	0.065157	0.076911	0.016802	0.010708	0.013716	0.01
last_pymnt_amnt	0.449141	0.453935	0.441733	0.151774	0.400833	0.138764	0.005047	-0.012314	0.02
pub_rec_bankruptcies	-0.033069	-0.033916	-0.038072	0.087316	-0.029298	-0.017123	0.010690	0.003123	0.01
year	0.115089	0.124641	0.260005	0.040477	0.043557	0.015221	0.088554	0.003492	-0.06
emp_length_int	0.157728	0.157484	0.169602	0.001328	0.127271	0.113938	0.049790	0.019371	0.00
loan_condition_int	0.044322	0.041278	0.019406	0.191863	0.019090	-0.040003	0.038357	0.020868	0.07



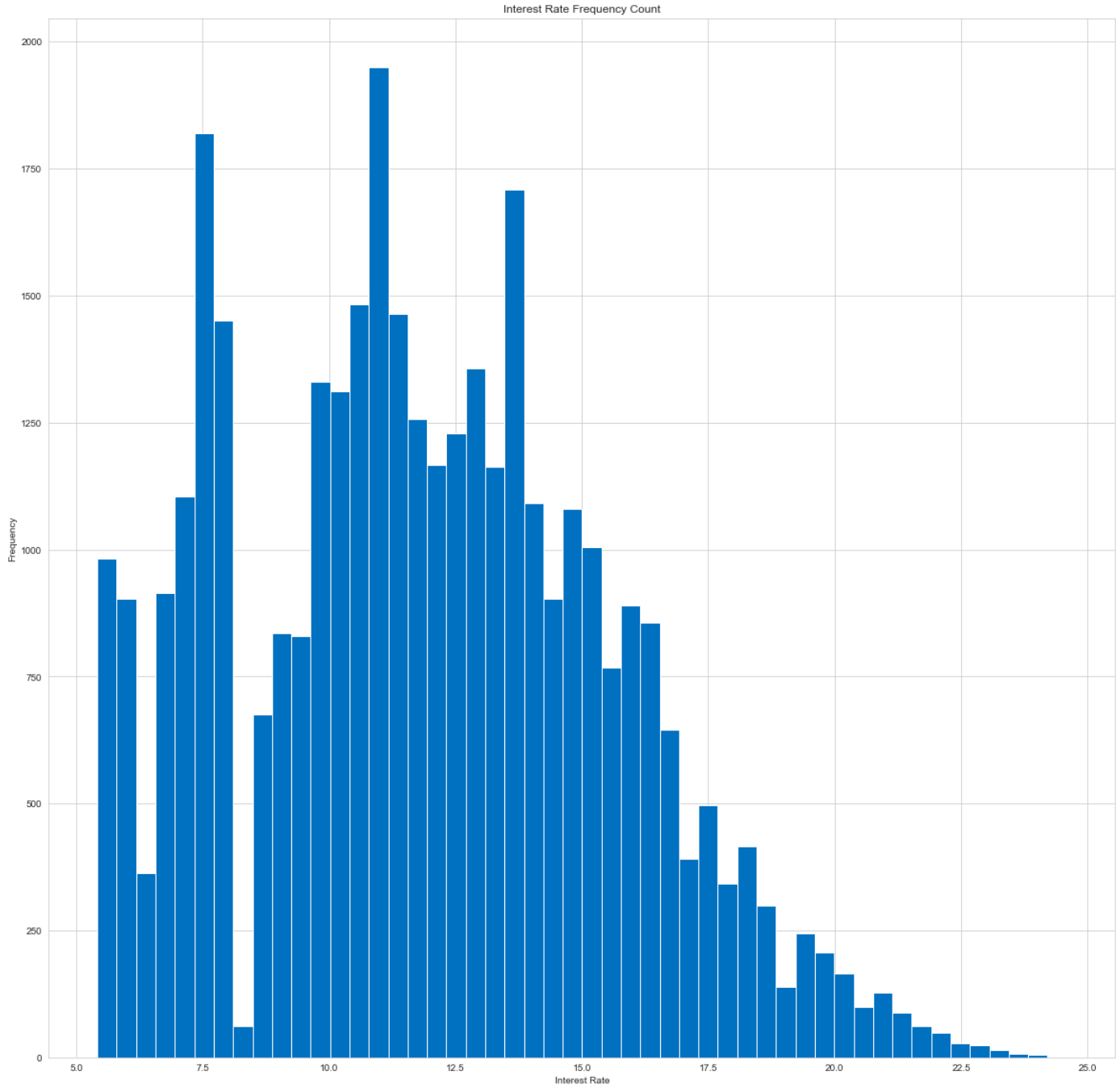

```
df['interest_payments']
```

```
0         Low
1        High
2         Low
3        High
4        High
        ...
35803     Low
35804     Low
35805     Low
35806     Low
```

```
35807    High
Name: interest_payments, Length: 35808, dtype: object
```

```
palette = ["#0070C0", "#FFC000"]
df['interest_rate'].hist(bins=50,color='#0070C0',figsize=(20,20))
plt.xlabel('Interest Rate')
plt.ylabel('Frequency')
plt.title('Interest Rate Frequency Count')
```

```
Text(0.5, 1.0, 'Interest Rate Frequency Count')
```



Appendix I - Model Code

1. RandomForest

```
# import package to process data and machine learning
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn import metrics
from sklearn.metrics import plot_confusion_matrix
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
```

```
# read data from file
df = pd.read_excel('LendingClubData_training.xlsx')
df_test_original = pd.read_excel('LendingClubData_testing.xlsx')
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	...	hardship_payoff_ba
0	NaN	NaN	35000	35000	34975.0	60 months	0.1171	773.44	B	B3	...	NaN
1	NaN	NaN	9500	9500	9500.0	36 months	0.1465	327.70	C	C3	...	NaN
2	NaN	NaN	3800	3800	3800.0	36 months	0.0751	118.23	A	A3	...	NaN
3	NaN	NaN	12400	12400	12400.0	60 months	0.2206	342.90	F	F4	...	NaN
4	NaN	NaN	4000	4000	4000.0	60 months	0.1727	100.00	D	D3	...	NaN

5 rows × 145 columns

```
# according to data analysis to determine useful features
useful_columns = ['loan_amnt', 'sub_grade', 'term', 'home_ownership', 'annual_inc',
                  'verification_status', 'dti', 'open_acc', 'inq_last_6mths', 'purpose',
                  'revol_util', 'loan_status']
parameters = ['loan_amnt', 'sub_grade', 'term', 'home_ownership','annual_inc', 'purpose', 'revol_util',
                  'verification_status', 'dti', 'open_acc', 'inq_last_6mths']
```

```
# drop missing data, low proportion
df_train = df[useful_columns]
df_train.dropna(inplace=True)
```

```
# drop features which are not in test data set
df_train = df_train[~df_train['purpose'].str.contains('educational')]
df_train = df_train[~df_train['home_ownership'].str.contains('NONE')]
df_train = df_train[~df_train['home_ownership'].str.contains('OTHER')]
```

```
df_train.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	loan_amnt	sub_grade	term	home_ownership	annual_inc	verification_status	dti	open_acc	inq_last_6mths	purpose
0	35000	B3	60 months	MORTGAGE	110000.0	Verified	1.06	10	0	small_business
1	9500	C3	36 months	RENT	54000.0	Verified	17.69	6	1	other
2	3800	A3	36 months	MORTGAGE	47000.0	Source Verified	22.52	10	3	car
3	12400	F4	60 months	OWN	65004.0	Source Verified	6.26	11	3	debt_consolidation
4	4000	D3	60 months	RENT	45000.0	Source Verified	7.37	10	0	other

```
# change targe value into (0, 1)
dic = {'Charged Off':0, 'Fully Paid': 1}
df_train['loan_status'].replace(dic, inplace=True)
```

```
# one-hot encoding transform categorical parameters into vectors, normalize numerical parameters
X_original_train = pd.get_dummies(df_train[parameters])

loan_amnt_max = X_original_train['loan_amnt'].max()
loan_amnt_min = X_original_train['loan_amnt'].min()

annul_inc_max = X_original_train['annual_inc'].max()
annul_inc_min = X_original_train['annual_inc'].min()

X_original_train['loan_amnt'] = (X_original_train['loan_amnt'] - loan_amnt_min)/(loan_amnt_max - loan_amnt_min)
X_original_train['annual_inc'] = (X_original_train['annual_inc'] - annul_inc_min)/(annul_inc_max - annul_inc_min)
Y_original_train = df_train['loan_status']
```

```
X_original_train.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	loan_amnt	annual_inc	revol_util	dti	open_acc	inq_last_6mths	sub_grade_A1	sub_grade_A2	sub_grade_A3	sub_grade_A4	...
0	1.000000	0.017665	0.064	1.06	10	0	0	0	0	0	...
1	0.260870	0.008326	0.853	17.69	6	1	0	0	0	0	...
2	0.095652	0.007158	0.393	22.52	10	3	0	0	1	0	...
3	0.344928	0.010161	0.775	6.26	11	3	0	0	0	0	...
4	0.101449	0.006825	0.825	7.37	10	0	0	0	0	0	...

5 rows × 62 columns

```
# same data process to test data set,
# using pad method to fill null value of only one missing value in revol_util feature
df_test = df_test_original[useful_columns]
df_test = df_test[useful_columns].fillna(method='pad')
y_test = df_test['loan_status'].replace(dic)
X_test = pd.get_dummies(df_test[parameters])
X_test['loan_amnt'] = (X_test['loan_amnt'] - loan_amnt_min)/(loan_amnt_max - loan_amnt_min)
X_test['annual_inc'] = (X_test['annual_inc'] - annul_inc_min)/(annul_inc_max - annul_inc_min)
```

```
# import Random Forest classifier
# import three resample process methods
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import RandomOverSampler, SMOTE, ADASYN
from imblearn.under_sampling import RandomUnderSampler, NearMiss, EditedNearestNeighbours
ros = RandomOverSampler(random_state=0)
sms = SMOTE()
ads = ADASYN()
rus = RandomUnderSampler()
nms = NearMiss(version=1)
enns = EditedNearestNeighbours()
```

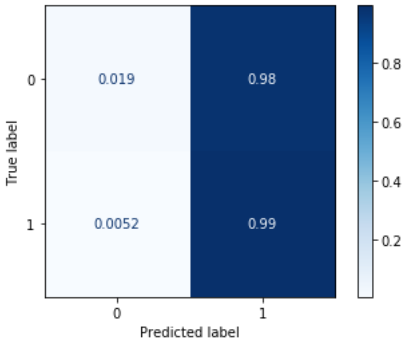
```
# default training and acurracy result
rf0 = RandomForestClassifier(random_state=10)
rf0.fit(X_original_train, Y_original_train)
rf0.score(X_test, y_test)
```

```
0.8273001508295625
```

```
# default training and AUC-ROC result
y_pred = rf0.predict(X_test)
metrics.roc_auc_score(y_test, y_pred)
```

```
0.5069371713208486
```

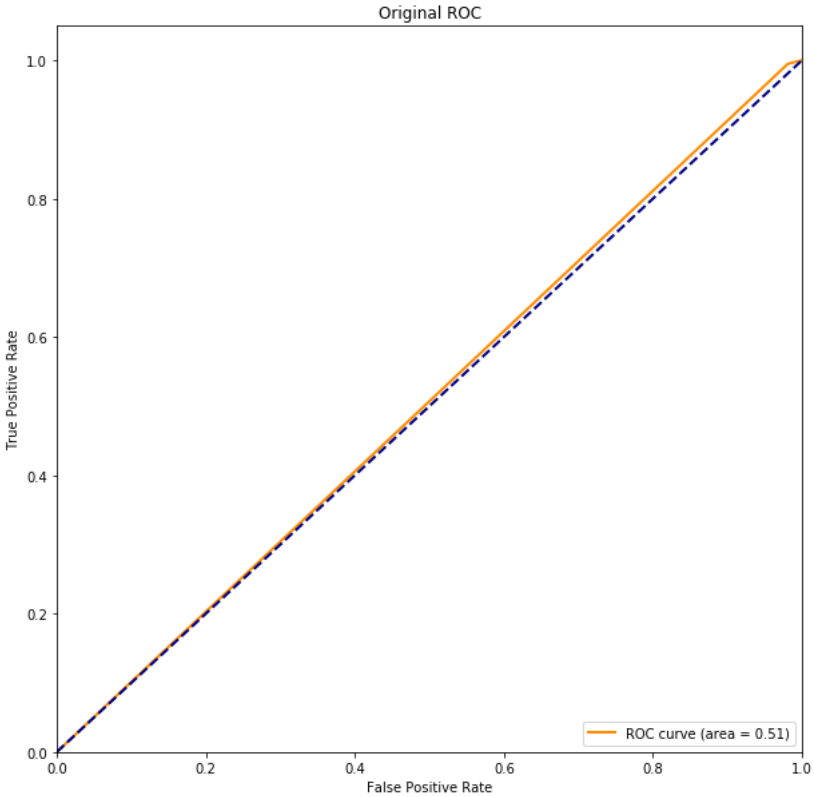
```
# confusion matrix display
disp = plot_confusion_matrix(rf0, X_test, y_test, cmap=plt.cm.Blues, normalize='true')
```




```
# AUC-ROC curve display
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)
plt.figure()
lw = 2
plt.figure(figsize=(10,10))
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Original ROC')
plt.legend(loc="lower right")
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```

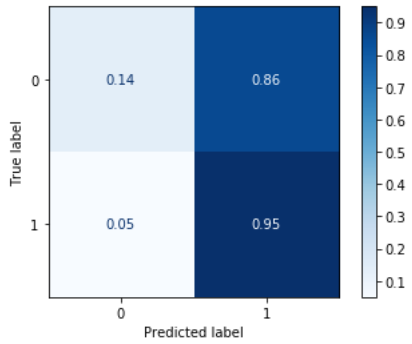


2. Over Resample and Under Resample

```
# prototype function for each resample method
def resample_result(model, x, y, x_test, y_test):
    X_train, y_train = model.fit_resample(x, y)
    Xtrain, Xtest, ytrain, ytest = train_test_split(X_train, y_train)
    rf = RandomForestClassifier(random_state=10)
    rf.fit(Xtrain, ytrain)
    print("Accuracy: ", rf.score(x_test, y_test))
    y_pred = rf.predict(X_test)
    print("Roc_auc: ", metrics.roc_auc_score(y_test, y_pred))
    disp = plot_confusion_matrix(rf, x_test, y_test, cmap=plt.cm.Blues, normalize='true')
```

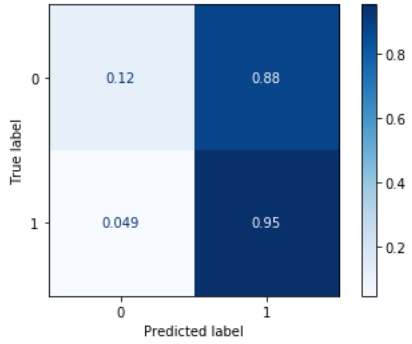
```
# Random Over Sampling
resample_result(ros, X_original_train, Y_original_train, X_test, y_test)
```

```
Accuracy:  0.8109602815485168
Roc_auc:  0.5452402482131629
```



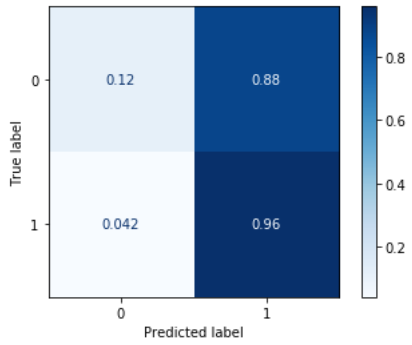

```
# SMOTE
resample_result(sms, X_original_train, Y_original_train, X_test, y_test)
```

```
Accuracy:  0.8086978381096028
Roc_auc:  0.5363303910046057
```



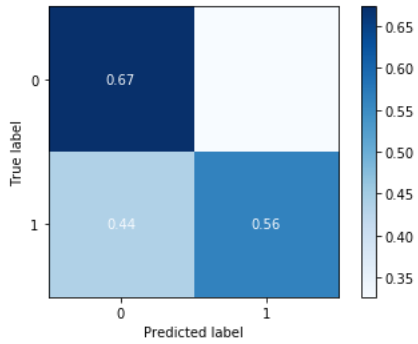

```
# ADASYN
resample_result(ads, X_original_train, Y_original_train, X_test, y_test)
```

```
Accuracy:  0.8137254901960784
Roc_auc:  0.5382046536635436
```



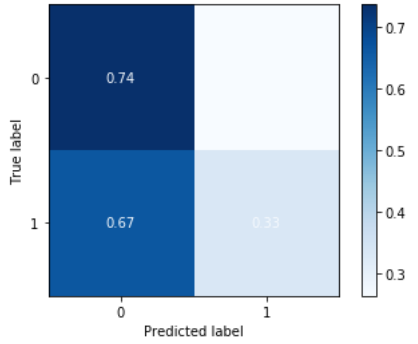

```
# Random Under Sampling
resample_result(rus, X_original_train, Y_original_train, X_test, y_test)
```

```
Accuracy:  0.5824534942182001
Roc_auc:  0.6185402257735555
```



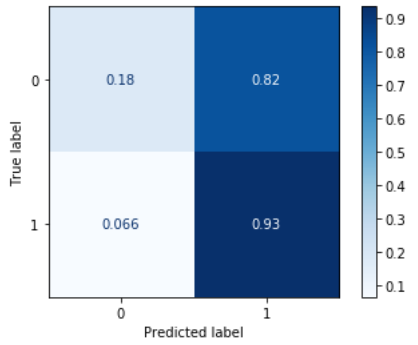

```
# NearMiss-1
resample_result(nms, X_original_train, Y_original_train, X_test, y_test)
```

```
Accuracy:  0.4034690799396682
Roc_auc:  0.5354514693499401
```




```
# Edited Nearest Neighbors
resample_result(enns, X_original_train, Y_original_train, X_test, y_test)
```

```
Accuracy:  0.8051784816490699
Roc_auc:  0.5585793728907325
```



3.1 Random Over Sample

```
# using Grid Search Method to tuning parameters
# random selection part of training data to tuning
from sklearn.model_selection import GridSearchCV
X_train, y_train = ros.fit_resample(X_original_train, Y_original_train)
Xtrain, Xtest, ytrain, ytest = train_test_split(X_train, y_train)
```

```
# tuning n_estimators
param_test1 = {'n_estimators': range(10, 71, 10)}
gsearch1 = GridSearchCV(estimator=RandomForestClassifier(min_samples_split=100,
```

```
                                                    min_samples_leaf=20,
                                                    max_depth=8,
                                                    max_features='sqrt',
                                                    random_state=10),
                        param_grid=param_test1, scoring='roc_auc', cv=5, n_jobs=4)
gsearch1.fit(Xtrain, ytrain)
gsearch1.best_params_, gsearch1.best_score_
```

```
({'n_estimators': 70}, 0.7049828745164634)
```

```
# tuning max_depth
param_test2 = {'max_depth':range(3, 14, 2), 'min_samples_split':range(50, 201, 20)}
gsearch2 = GridSearchCV(estimator=RandomForestClassifier(n_estimators=70,
                                                    min_samples_leaf=20,
                                                    max_features='sqrt',
                                                    random_state=10,
                                                    ),
                        param_grid=param_test2, scoring='roc_auc', cv=5, n_jobs=4)
gsearch2.fit(Xtrain, ytrain)
gsearch2.best_params_, gsearch1.best_score_
```

```
({'max_depth': 13, 'min_samples_split': 50}, 0.7071740643350222)
```

```
# tuning min_samples_leaf and min_samples_split
param_test3 = {'min_samples_split':range(10,90,20), 'min_samples_leaf':range(10,60,10)}
gsearch3 = GridSearchCV(estimator = RandomForestClassifier(n_estimators= 70, max_depth=13,
                                max_features='sqrt', random_state=10),
                         param_grid = param_test3, scoring='roc_auc', cv=5, n_jobs=4)
gsearch3.fit(Xtrain, ytrain)
gsearch3.best_params_, gsearch3.best_score_
```

```
({'min_samples_leaf': 10, 'min_samples_split': 10}, 0.762481897416864)
```

```
# tuning max_features
param_test4 = {'max_features':range(3,11,2)}
gsearch4 = GridSearchCV(estimator = RandomForestClassifier(n_estimators= 70, max_depth=13, min_samples_split=10,
                                min_samples_leaf=10, random_state=10),
   param_grid = param_test4, scoring='roc_auc', cv=5, n_jobs=4)
gsearch4.fit(Xtrain, ytrain)
gsearch4.best_params_, gsearch4.best_score_
```

```
({'max_features': 9}, 0.7752240457345134)
```

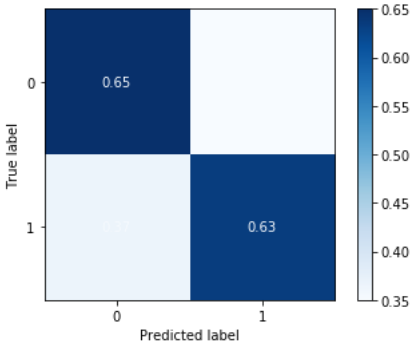
```
# fit data and display result
rf2 = RandomForestClassifier(n_estimators= 70, max_depth=13, min_samples_split=10,
                                min_samples_leaf=10, max_features=9, random_state=10)
rf2.fit(Xtrain, ytrain)
print("Accuracy:", rf2.score(X_test, y_test))
```

```
Accuracy: 0.6370035193564605
```

```
y_pred = rf2.predict(X_test)
print("AUC-ROC: ", metrics.roc_auc_score(y_test, y_pred))
```

```
AUC-ROC:  0.6421837959373202
```

```
# display confusion matrix
disp = plot_confusion_matrix(rf2, X_test, y_test, cmap=plt.cm.Blues, normalize='true')
```



3.2 Random Under Sampling (same process above)

```
X_train, y_train = rus.fit_resample(X_original_train, Y_original_train)
Xtrain, Xtest, ytrain, ytest = train_test_split(X_train, y_train)
```

```
param_test1 = {'n_estimators': range(10, 71, 10)}
gsearch1 = GridSearchCV(estimator=RandomForestClassifier(min_samples_split=100,
                                                         min_samples_leaf=20,
                                                         max_depth=8,
                                                         max_features='sqrt',
                                                         random_state=10),
                        param_grid=param_test1, scoring='roc_auc', cv=5, n_jobs=4)
gsearch1.fit(Xtrain, ytrain)
gsearch1.best_params_, gsearch1.best_score_
```

```
({'n_estimators': 70}, 0.6788839497144867)
```

```
param_test2 = {'max_depth':range(3, 14, 2), 'min_samples_split':range(50, 201, 20)}
gsearch2 = GridSearchCV(estimator=RandomForestClassifier(n_estimators=70,
                                                         min_samples_leaf=20,
                                                         max_features='sqrt',
                                                         random_state=10,
                                                         ),
                        param_grid=param_test2, scoring='roc_auc', cv=5, n_jobs=4)
gsearch2.fit(Xtrain, ytrain)
gsearch2.best_params_, gsearch1.best_score_
```

```
({'max_depth': 13, 'min_samples_split': 150}, 0.6788839497144867)
```

```
param_test3 = {'min_samples_split':range(110,190,20), 'min_samples_leaf':range(10,60,10)}
gsearch3 = GridSearchCV(estimator = RandomForestClassifier(n_estimators= 70, max_depth=13,
                                 max_features='sqrt', random_state=10),
                         param_grid = param_test3, scoring='roc_auc', cv=5, n_jobs=4)
gsearch3.fit(Xtrain, ytrain)
gsearch3.best_params_, gsearch3.best_score_
```

```
({'min_samples_leaf': 10, 'min_samples_split': 150}, 0.6826609159702903)
```

```
param_test4 = {'max_features':range(3,11,2)}
gsearch4 = GridSearchCV(estimator = RandomForestClassifier(n_estimators= 70, max_depth=13, min_samples_split=150,
                                 min_samples_leaf=10, random_state=10),
   param_grid = param_test4, scoring='roc_auc', cv=5, n_jobs=4)
```

```
gsearch4.fit(Xtrain, ytrain)
gsearch4.best_params_, gsearch4.best_score_
```

```
({'max_features': 7}, 0.6826609159702903)
```

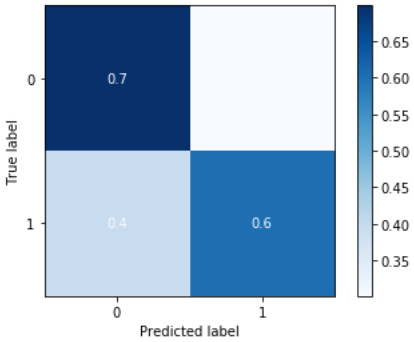
```
rf3 = RandomForestClassifier(n_estimators= 70, max_depth=13, min_samples_split=150,
                                 min_samples_leaf=10, max_features=7, random_state=10)
rf3.fit(Xtrain, ytrain)
rf3.score(X_test, y_test)
```

```
0.6191553544494721
```

```
y_pred = rf3.predict(X_test)
metrics.roc_auc_score(y_test, y_pred)
```

```
0.6505604347507316
```

```
disp = plot_confusion_matrix(rf3, X_test, y_test, cmap=plt.cm.Blues, normalize='true')
```



4. Ensemble Model

```
# ensemble models into list
ensemble_model = []
for i in range(11):
    X_train, y_train = rus.fit_resample(X_original_train, Y_original_train)
    Xtrain, Xtest, ytrain, ytest = train_test_split(X_train, y_train)
    under_sample_model = RandomForestClassifier(n_estimators= 70,
                                                max_depth=13,
                                                min_samples_split=150,
                                                min_samples_leaf=10,
                                                max_features=7,
                                                random_state=10)
    under_sample_model.fit(Xtrain, ytrain)
    ensemble_model.append(under_sample_model)

    X_train, y_train = ros.fit_resample(X_original_train, Y_original_train)
    Xtrain, Xtest, ytrain, ytest = train_test_split(X_train, y_train)
    over_sample_model = RandomForestClassifier(n_estimators=70,
                                               max_depth=13,
                                               min_samples_split=10,
                                               min_samples_leaf=10,
                                               max_features=9,
                                               random_state=10)
    over_sample_model.fit(Xtrain, ytrain)
    ensemble_model.append(over_sample_model)
```

```
# using ensemble model to predict
results = []
for model in ensemble_model:
    y_pred = model.predict(X_test)
    results.append(y_pred)

y_ensemble_result = []
for i in range(len(results[0])):
    y_single_pred = 0
    for j in range(len(results)):
        y_single_pred = y_single_pred + results[j][i]
    y_ensemble_result.append(y_single_pred > 6)
print("Acurracy: ", y_ensemble_result.count(1)/len(y_ensemble_result))
```

```
Acurracy:  0.6085972850678733
```

```
fpr, tpr, thresholds = roc_curve(y_test, y_ensemble_result)
roc_auc = auc(fpr, tpr)
plt.figure()
lw = 2
plt.figure(figsize=(10,10))
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc) ###假正率为横坐标，真正率为纵坐标做曲线
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Original ROC')
plt.legend(loc="lower right")
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```