Lending Club Data Analysis and DEFAULT LoaN/RATING Prediction

Project report

*submitted by*

**KAUSHIK PRASAD DEY**

# Index

[**Index 2**](#_Toc512968977)

[**Overview 2**](#_Toc512968978)

[**1. Project Background and Description 2**](#_Toc512968979)

[**2. Project Scope 3**](#_Toc512968980)

[**3. Explanatory Data Analysis 4**](#_Toc512968981)

[**4. Data Cleaning, Encoding & Missing Data Imputation 17**](#_Toc512968982)

[**5. Learning Algorithm Implementation 18**](#_Toc512968983)

[**6. Useful-ness of the model 23**](#_Toc512968984)

[**7. Summary 23**](#_Toc512968985)

[**8. Future Work 24**](#_Toc512968986)

[**9. References 24**](#_Toc512968987)

[**10. Acknowledgement 24**](#_Toc512968988)

# 

# Overview

## Project Background and Description

This is a Course project for CISC-5950 Big Data Programming, Fordham University. Under the scope of the course work, we are required to solve an analysis/learning problem using the Big-Data frameworks and techniques taught in the course.

We used “*Lending Club historical dataset*” for our analysis and modeling. This is an open source dataset from lending club, which contains complete loan data for all the loan issued through 2007-2015. The data is available to download on the following site- <https://www.kaggle.com/wendykan/lending-club-loan-data>

The peer-to-peer lending industry has grown significantly since its inception in 2007. With billions in annual loans, there are significant opportunities to capitalize on this alternative investment instrument. We have developed a sophisticated learning model to bolster investment strategy that utilizes Lending Club Corporation’s massive historical datasets to understand which features best predict someone’s probability of defaulting loan. To model our default loan prediction, we used many machine learning algorithms/techniques and implemented them using big-data distributed computing frameworks we learnt during our course work.

**Summary of results**

For this project, we performed explanatory data analysis using apache spark framework to gather business insights from the historical loan data. We build four machine learning classifiers to identify the borrowers who are more likely to default on their loan. We develop Logistic Regression, Naïve Bayes, Random forest and Gradient Boosting Classifier. We implemented spark’s ML pipeline with parameter grid to find the best hyper parameters for each classifier. Cross validation, confusion matrix and ROC are used to evaluate each classifier’s performance. Among these classifiers, Gradient Boosting and Random Forest are predicting the nearly perfect scores.

**Technology Specs:** findSpark 1.2.0, pySpark 2.3.0, Spark 2.3.0, Spark MLlib, Anaconda, Jupyter Notebook, python and R-programming, Seaborn, pyplot libraries.

## Project Scope

Our project scope is to run the exploratory data analysis using apache spark framework to find the business insights from our loan data, and to build a learning model using data mining techniques / machine leaning algorithms that will use the historic loan data to learn and helps to identify loans/borrowers which are likely to default.

As per the recent studies, 3-4% of the total loans defaults every year. This is a huge risk for the investors who is funding the loans. Investors require more comprehensive assessment of these borrowers than what is presented by Lending Club to make a smart business decision. Data mining techniques and Machine Learning model/analysis could help predicting the loan default likelihood which may allow investors to avoid loan defaults thus limiting the risk of their investments.

## Explanatory Data Analysis

The data used for this project is the structured data with few missing/null values. This data consists of 80+ features of three distinct types: continuous, categorical, ordinal. We gathered business domain knowledge about the data to deal with data cleaning and missing data imputation. During our preliminary exploration of data, we noticed some of the features with more than 50% of the missing data, we planned to drop these features while data processing for learning model.

For data analysis phase, we converted some of the feature type to relevant primitive types and moved on. We handled data cleaning and imputation part while preparing data for learning model. For our preliminary data cleanup, we used spark data frame transformation API’s to convert the feature data types.

As part of our exploratory data analysis we tried to find the interesting fact and findings about the loans from the historical loan data. This analysis helped us to develop our understanding about data and its distribution patterns. In addition, this assisted us to select the most effecting features and develop business rules for missing data imputation.

***3.1) Analyzing Loan amount and Interest rates:***

By using spark’s transformation and action API’s, we aggregated our data by loan amount and loan interest rate and plotted the distribution plot and box plot.

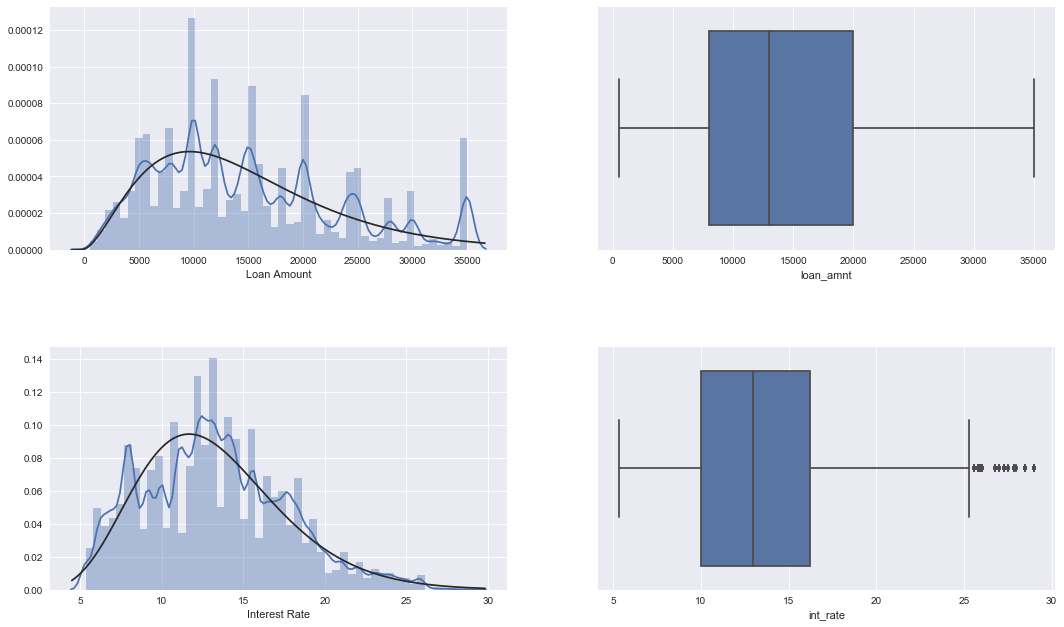


Figure 1: Distribution of Loan and Interest

From above Figure 1, we conclude that both Interest rate & loan amount is normally distributed, slightly right skewed. i.e. most of the customers are seeking loan amount ranging 5-15K and their interest rate is ranging from 10-15%. Our loan amount shows no outliers point, since it fits perfectly within the five-point summary. However, Interest rates shows few outliers falling outside the 1.5IQR range. Those customers may have **poor credit ratings**, therefore **high interest rate.**

We analyzed the distribution for each of our features included in our final model. From our analysis, we determine the appropriate distribution function to use in estimating Maximum Likelihood Estimator. The more detailed description is added in the later section of the report.

***3.2) Analyzing Loans Interest rates over time:***

To analyze how are loan book is changing over time in terms of loan amount and interest rate, we first created a new column for quarterly date range. We used spark’s *withcolumn* and *to\_date* API’s to extract this information from the loan issue date column. After which we aggregated the data over quarterly date range and plotted in the point plot. To get the total loan amounts over the time we grouped by on the date variable and aggregate all the loans amount to get the total loans. We calculated customers loan requirements over the time (Median loan amount and median interest) by using *percentile\_approx* function.

*This function is faster way to compute the quantile range in very large set of data where data stored in multiple partition. The underline algorithm usages approximation technique to find the quantile range instead of running the computation in all the partitions.*

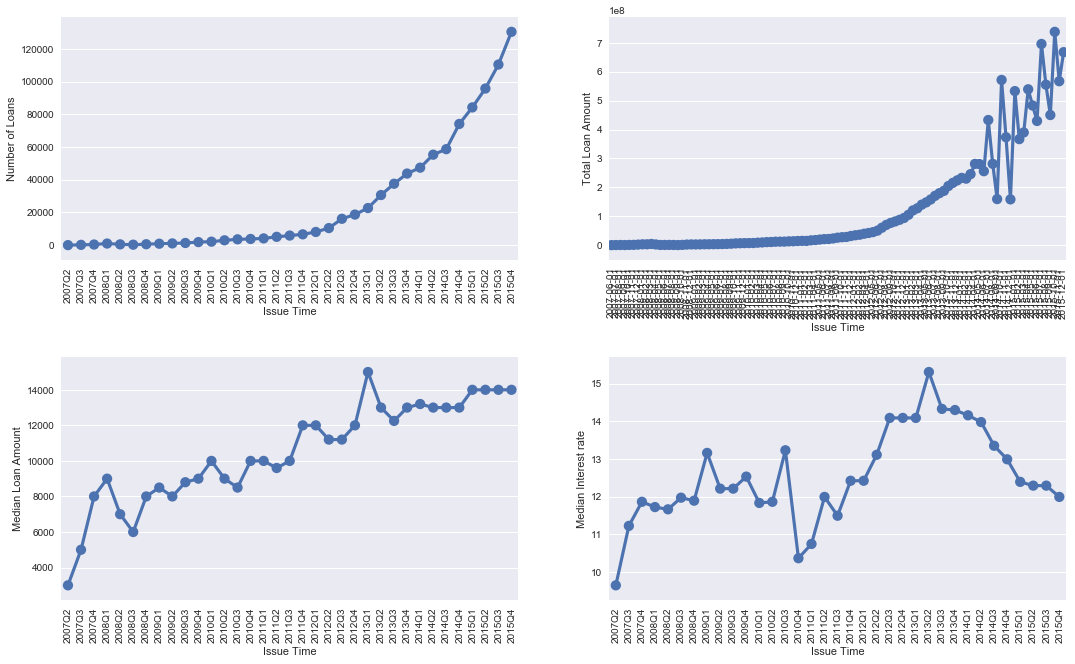


Figure 2:Analyzing Loans Interest rates over time

The first two plots show how number of loans and total loan amount is growing over time, this gives an indication of growing borrowers at lending club platform. The next two plots are between median loan amount & interest over time. This indicates that the borrower’s loan requirements are increasing over time so as the median interest rates. However, there is a sharp decline in the interest rate during 2010, which might be due to 2009 fiscal crisis.

***3.3) Analyzing Loans over loan status:***

We analyze the loans according to their current status. We used complex spark data frame operations to know the distribution of total accounts for each loan status and distribution of loan amount with the probability density for each loan value over the loan status. We used the density distribution to analyze any anomalies/outliers specific to the status of loans.

To show the distribution of loan amount, we have used the violin plot (figure 5). A *Violin Plot* is used to visualize the distribution of the data and its probability density. This chart is a combination of a Box Plot and a Density Plot. The density plot is rotated and placed on both sides of the box plot, to show the distribution shape of the data.

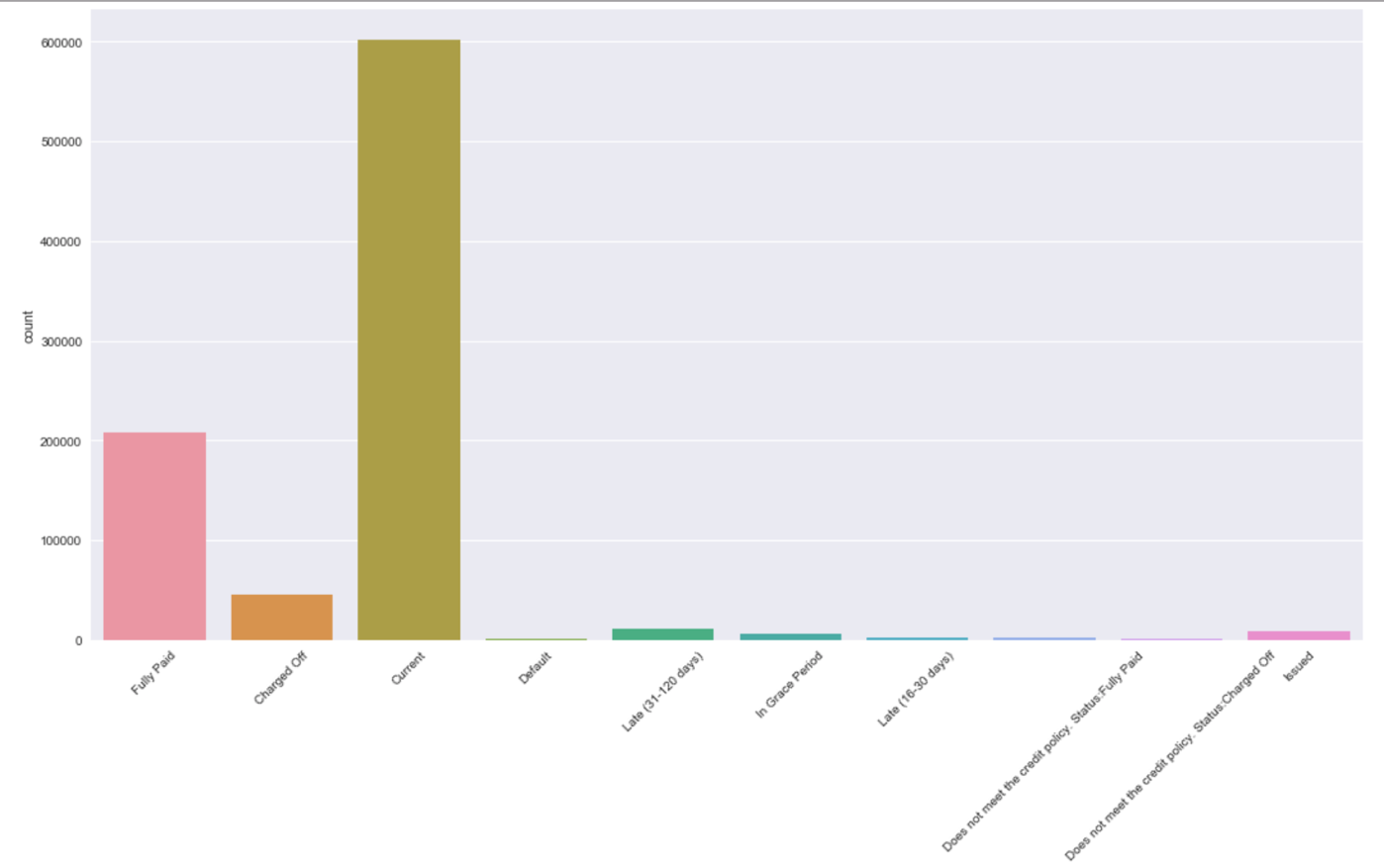


Figure 3: Bar plot for Analyzing Loans over loan status

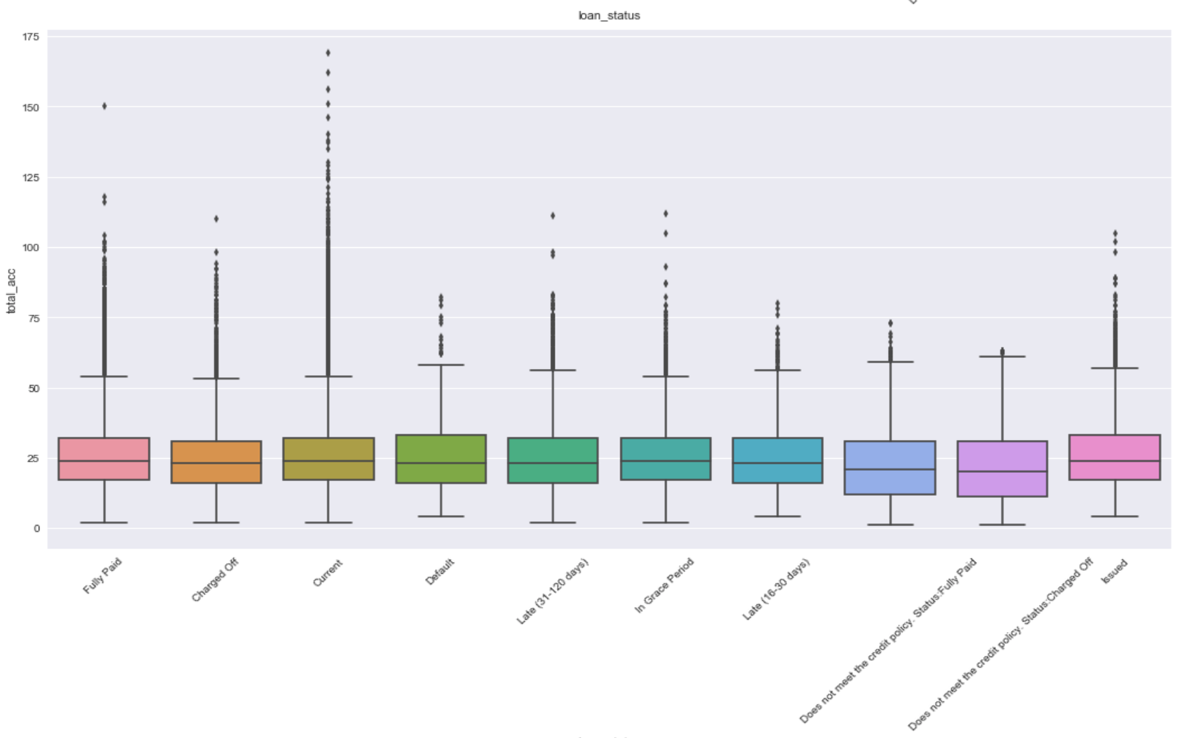


Figure : Box plot for Analyzing Loans over loan status

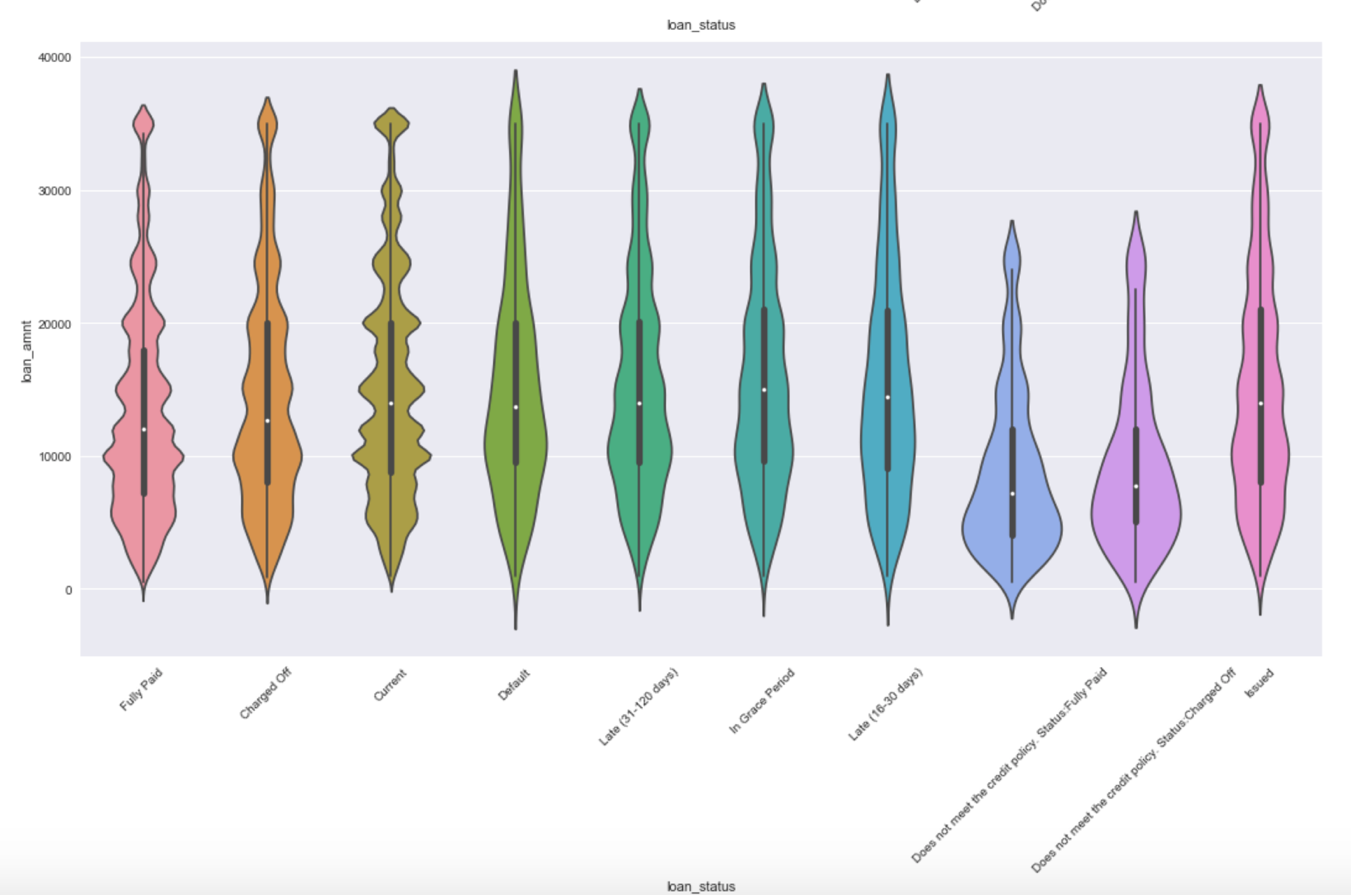


Figure : Violin Plot for Analyzing Loans over loan status

By running a group by query, we found out the number of loans for each loan status. The results are listed in following table.

|  |  |
| --- | --- |
| Loan status | count |
| Current | 601776 |
| Fully Paid | 207533 |
| Charged Off | 45215 |
| Late (31-120 days) | 11591 |
| Issued | 8460 |
| In Grace Period | 6253 |
| Late (16-30 days) | 2357 |
| Does not meet the... | 1969 |
| Default | 1219 |
| Does not meet the... | 751 |

We also analyzed about the verification status of the loan which were defaulted. We achieve this result by running a group by aggregations on *verification\_status* and default *loan status*.

|  |  |
| --- | --- |
| verification\_status | Count |
| Verified | 479 |
| Source Verified | 462 |
| Not Verified | 278 |

***3.4) Analysis of Grades and Subgrades: Analyzing loan amount and interest rate quantile summary for each grade, factored over sub grade.***

Based on borrower’s credit score, credit history, desired loan amount and the borrower’s debt-to-income ratio, Lending Club determines whether the borrower is credit worthy. After that they assign a credit grade that determines payable interest rate and fees to their approved loans. These grades are assigned within an alphabetical range from A to G. Each of these letter grades has five finer-grain sub-grades, numbered 1 to 5, with 1 being the highest category with-in the grade. Loan interest rates is inverse proportionate to the credit grade. ‘A’ being the highest grade, therefore low interest rate and vice-a-versa.

In figure 6, we can see a linear relationship between loan amount and customer credit ratings, notice here that requested loan amount is slightly higher for the low rating customers. This indicates that higher rating borrowers are financially stronger and require lesser loan amount in general.

In figure 7, we can see that the median interest rates increase for low credit rating customers.

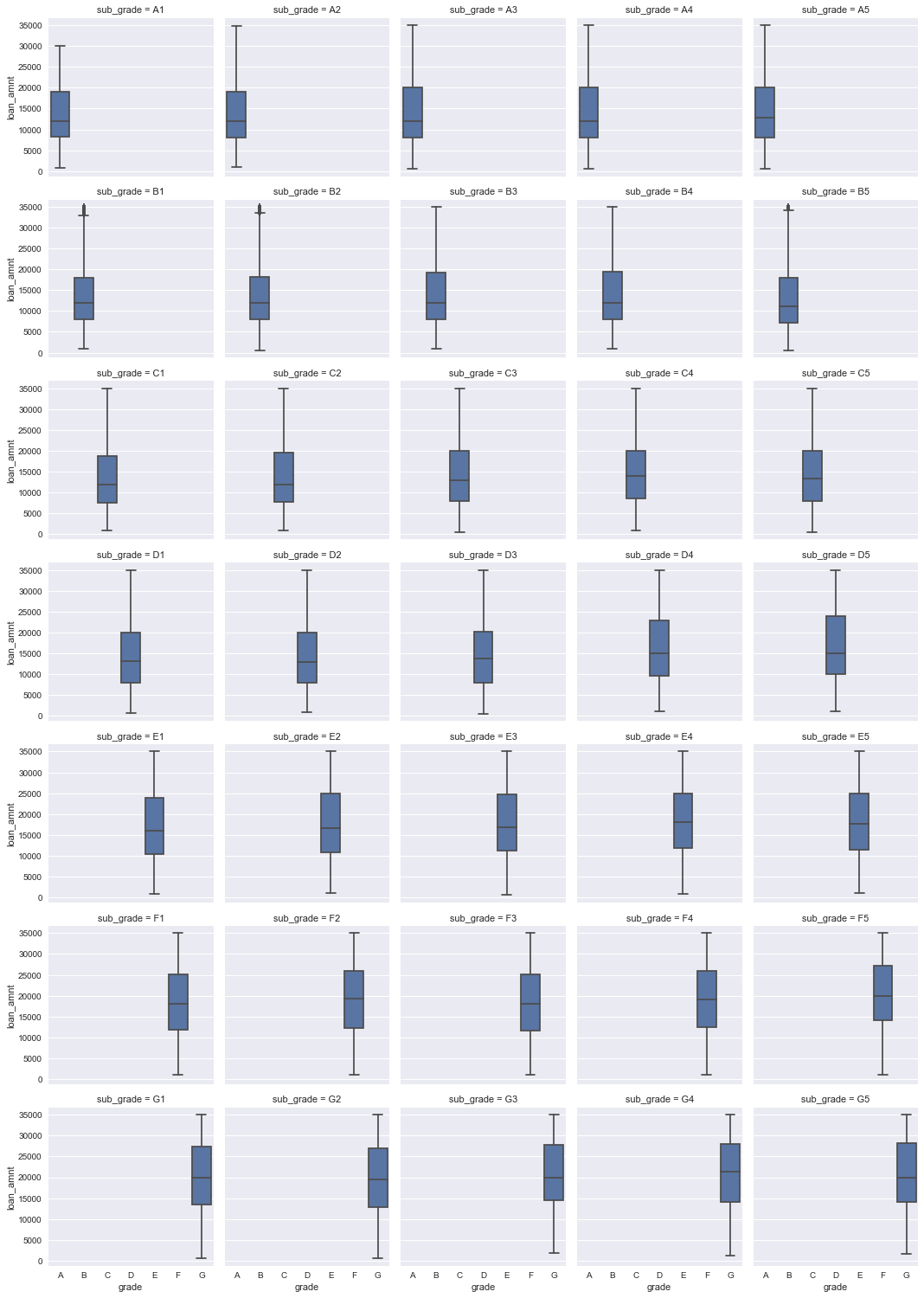


Figure : Analyzing loan amount distribution for each grade, factored over sub grade.

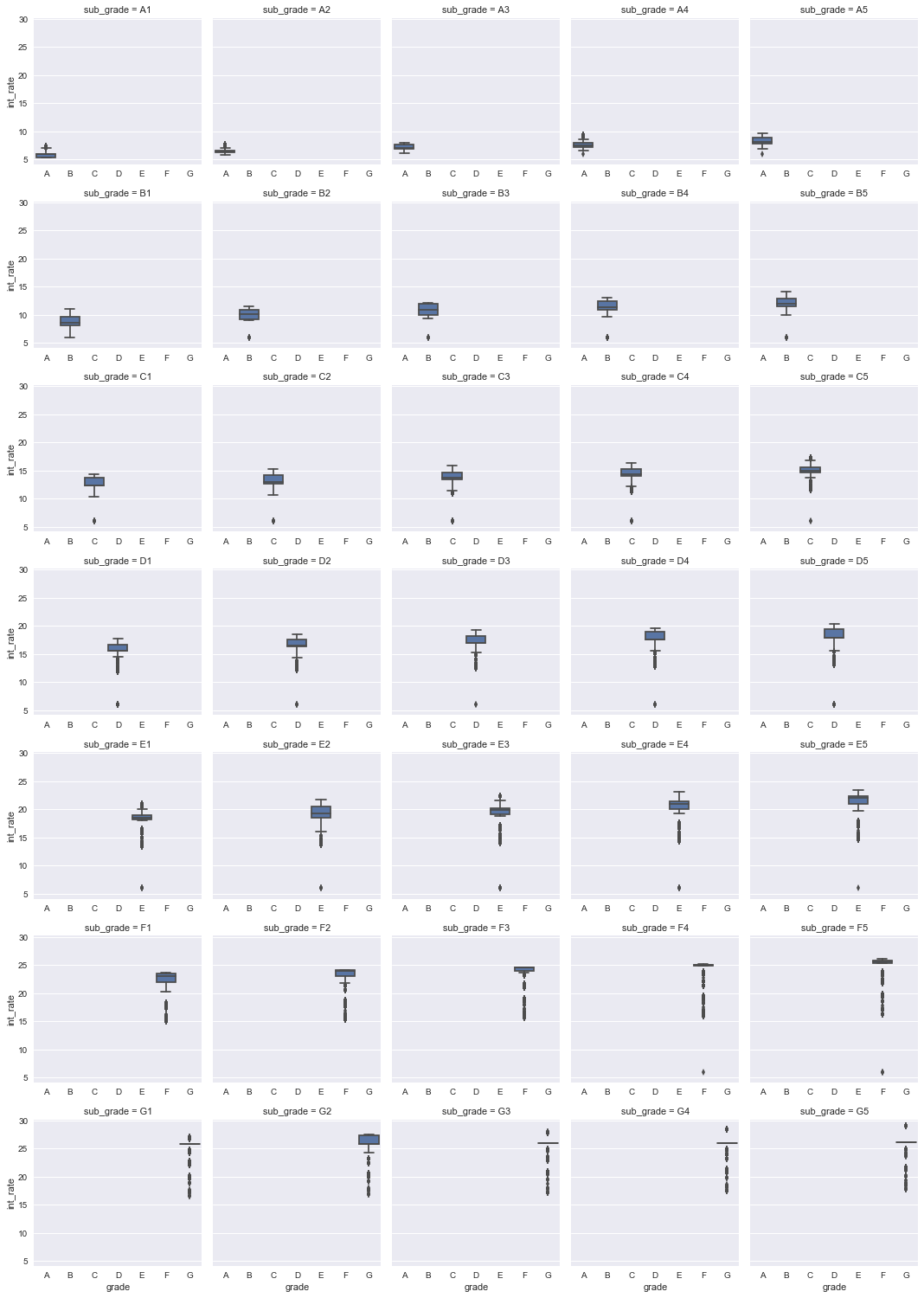


Figure : Analyzing interest rate distribution for each grade, factored over sub grade.

***3.5) US states map with the total loan amount***

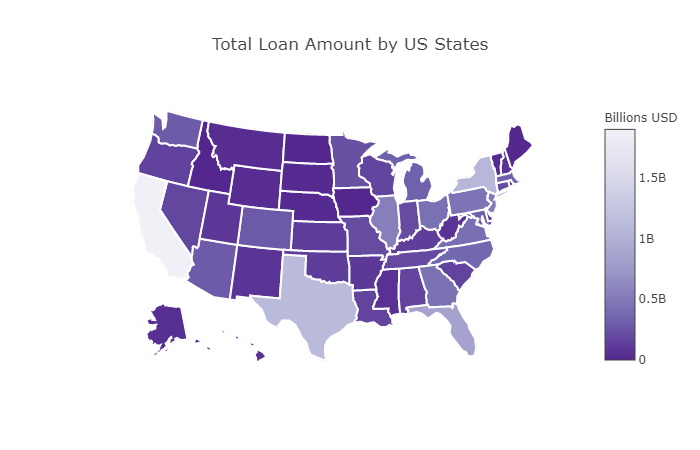


Figure :US states map with the total loan amount

***3.6) US states map with the median interest rates***

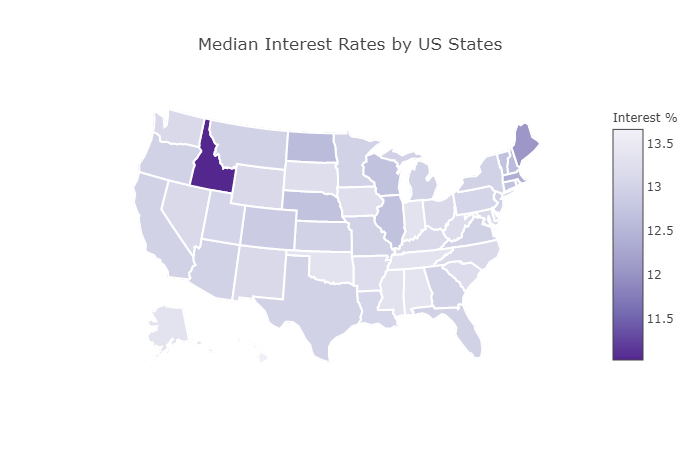


Figure :US states map with the median interest rates

*Note: - To further analyze the state wise effect on loan default, we tried to merge the census data for median house-hold income and inflation rate for each state. But the data we collected was not cleaned and had lot of missing states.*

***3.7) Total loan amount by income range and loan status***

We created bins for continuous variables to define the income range to enhance interpretation of results. To perform the analysis of income range, we created 15 bins for income range. To create these bins, we used spark’s ***Bucketizer*** API to create the bins. Bucketizer transforms a column of continuous features to a column of feature buckets, where the buckets are specified by users. However, this method was adding an extra computation time in our processing, so we create these bins by writing our own customize code.

We created the income range bins in a way so that our data can fit in to the original distribution (Gaussian) of continuous feature. Using the new range column, we calculated:

* Total number of loans grouped by income range and loan status
* Total loan amount by income range and loan status

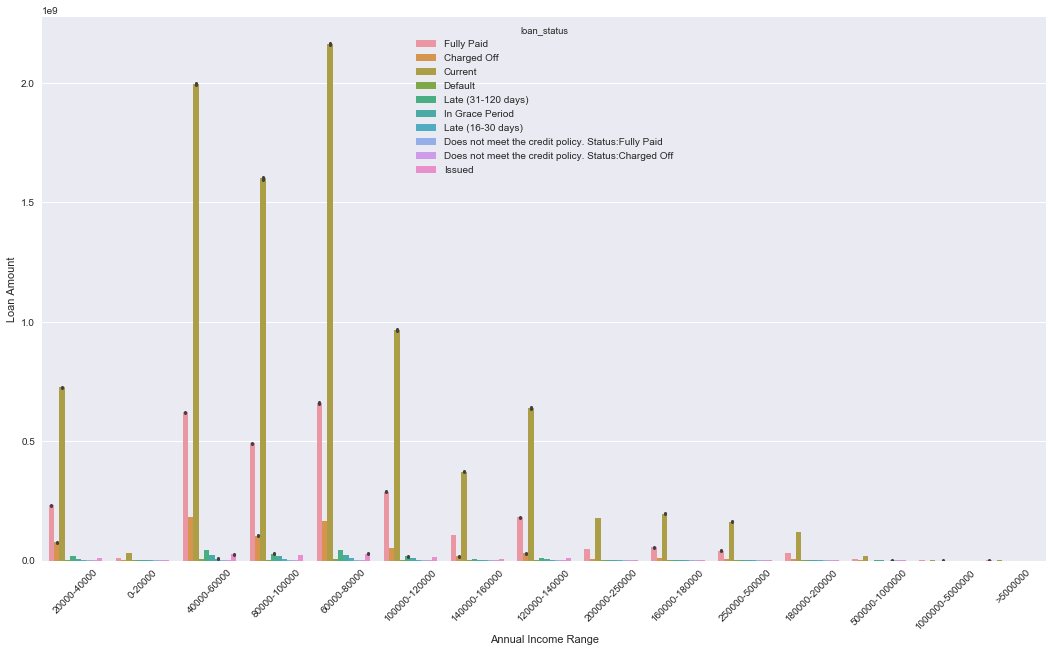


Figure :Total loan amount by income range and loan status

***3.8) Analyzing Loan Amount and interest rate over customers employment length (With the loan term).***

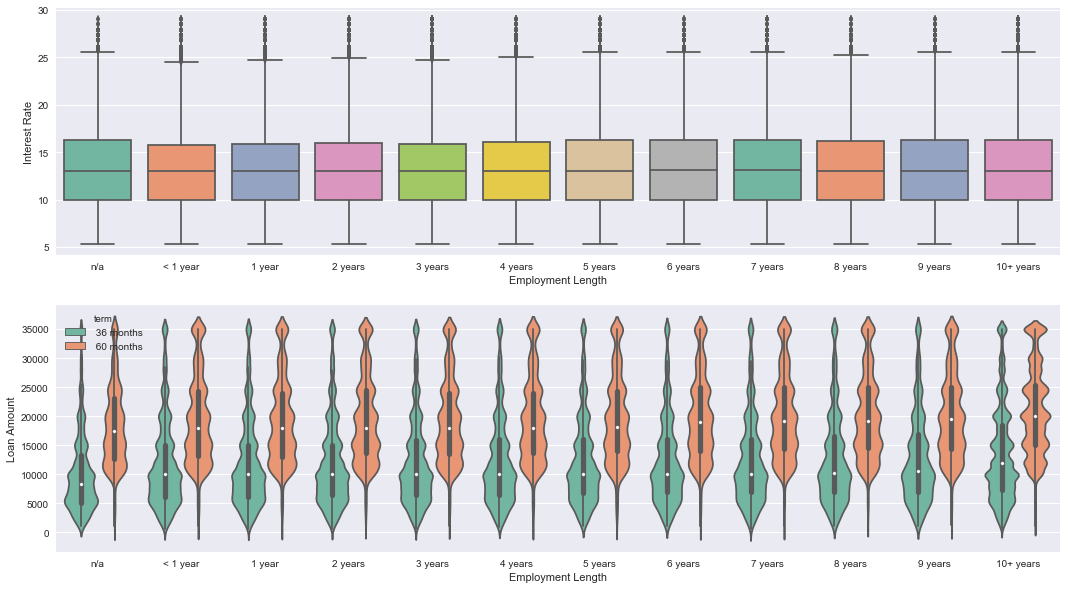
****

Figure : Analyzing Loan Amount and interest rate over customers employment length

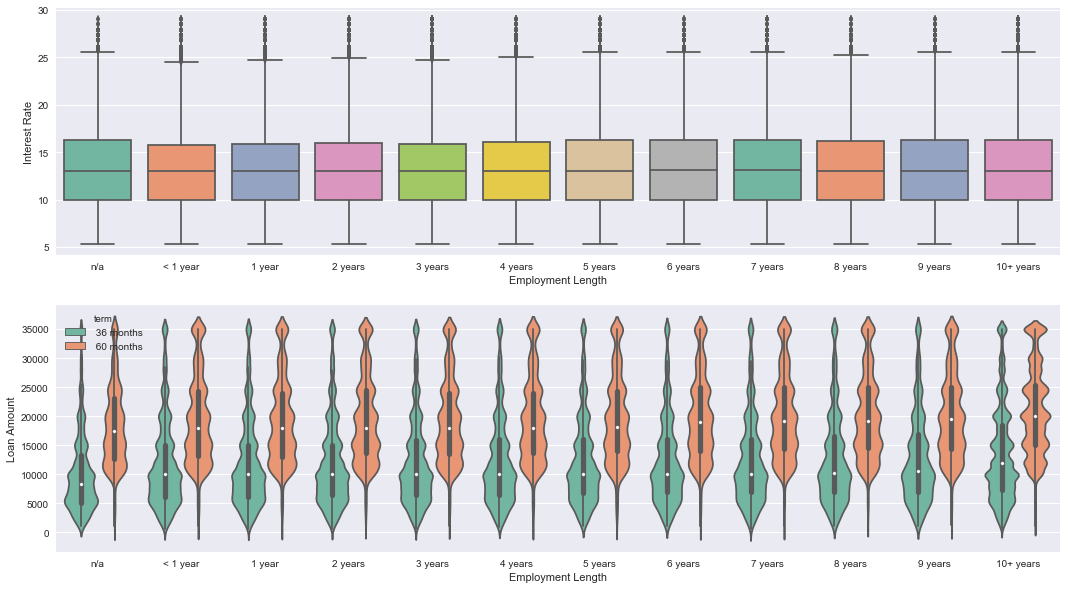
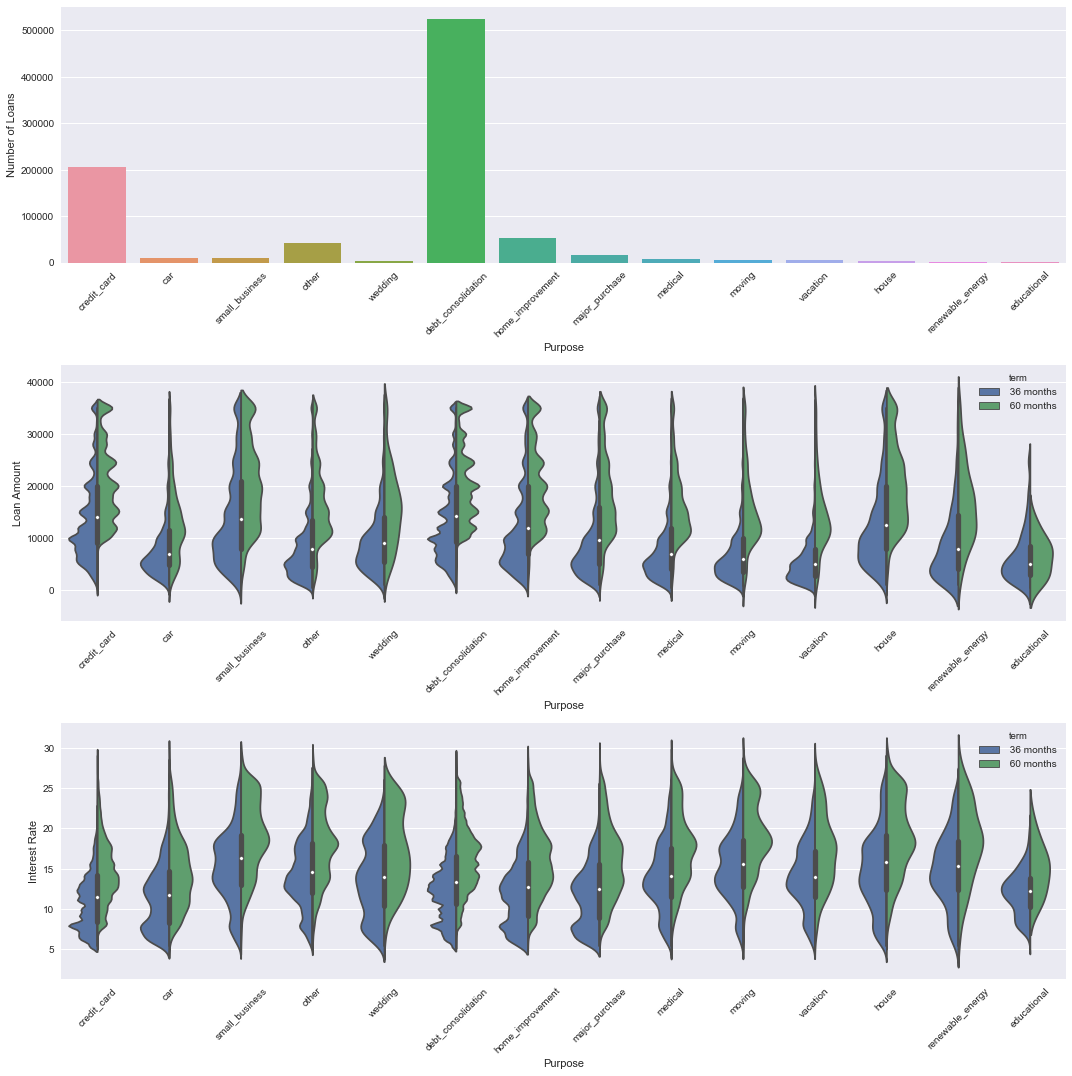


Figure : Analyzing Loan Amount and interest rate over customers employment length

***3.9) Analyzing loans by its purpose:***

In figure 13 below, the first plot shows number of loans by its purposes. The second plot shows the Loan amount with its distribution pattern by purpose and the third plot shows interest rate with its distribution pattern by purpose.



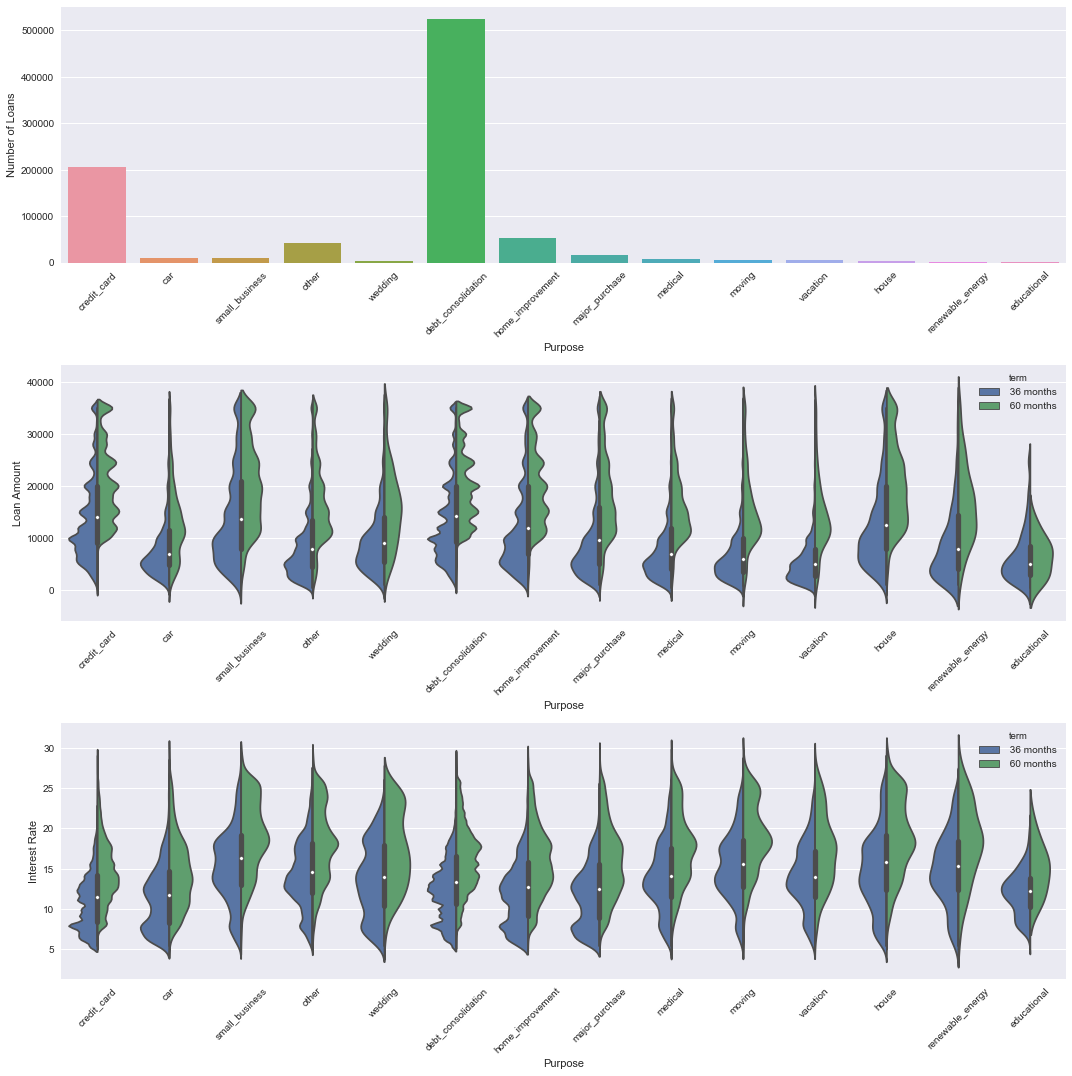


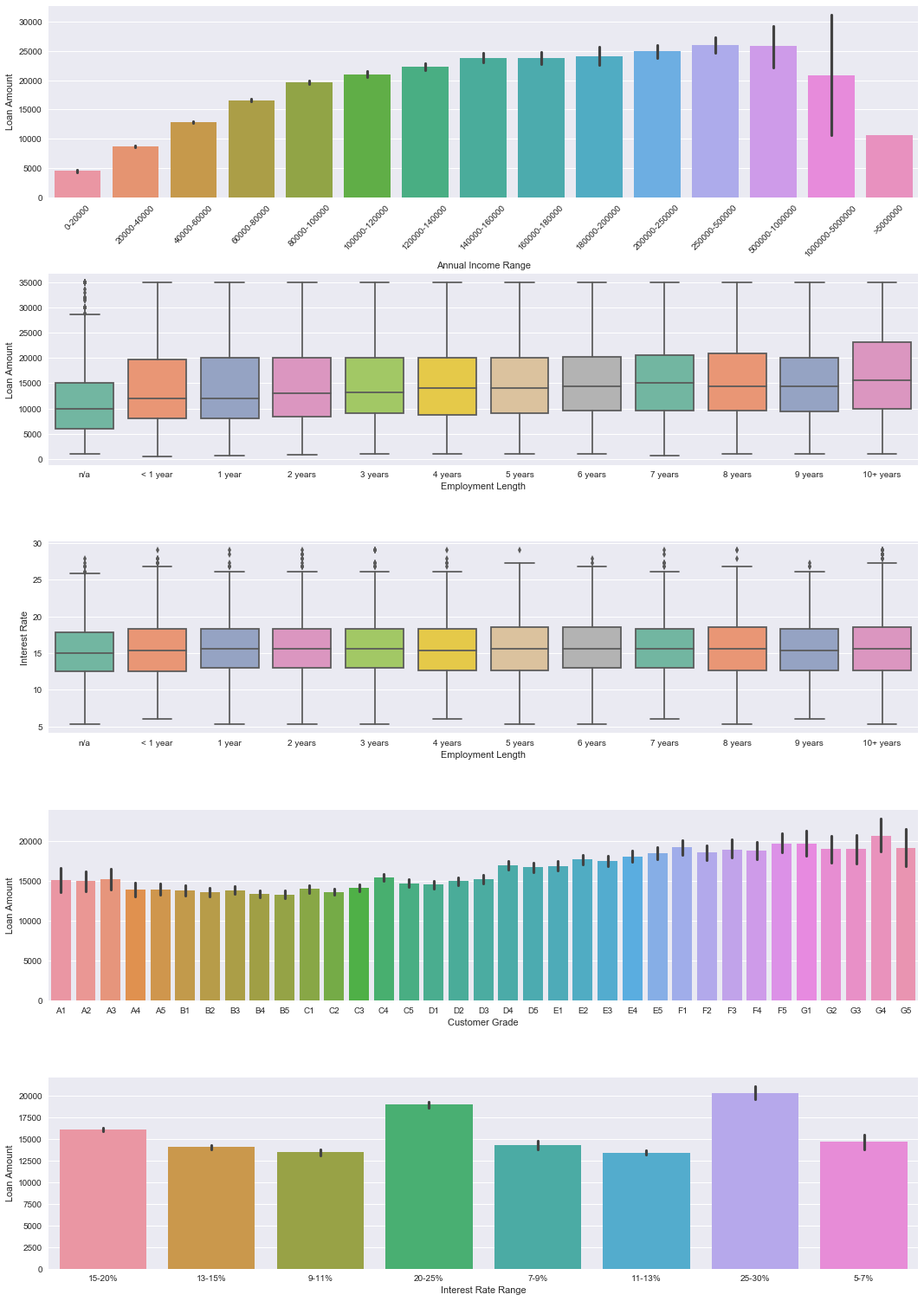
Figure : Analyzing loans by its purpose

***3.10) Analyzing Default loans***

We created the bins for interest rates as well in the same way as mentioned as above.

Loan status which are in following status are considered as defaulted –

* Default, Late (31-120 days),
* In Grace Period, Late (16-30 days),
* Does not meet the credit policy.
* Status: Charged Off



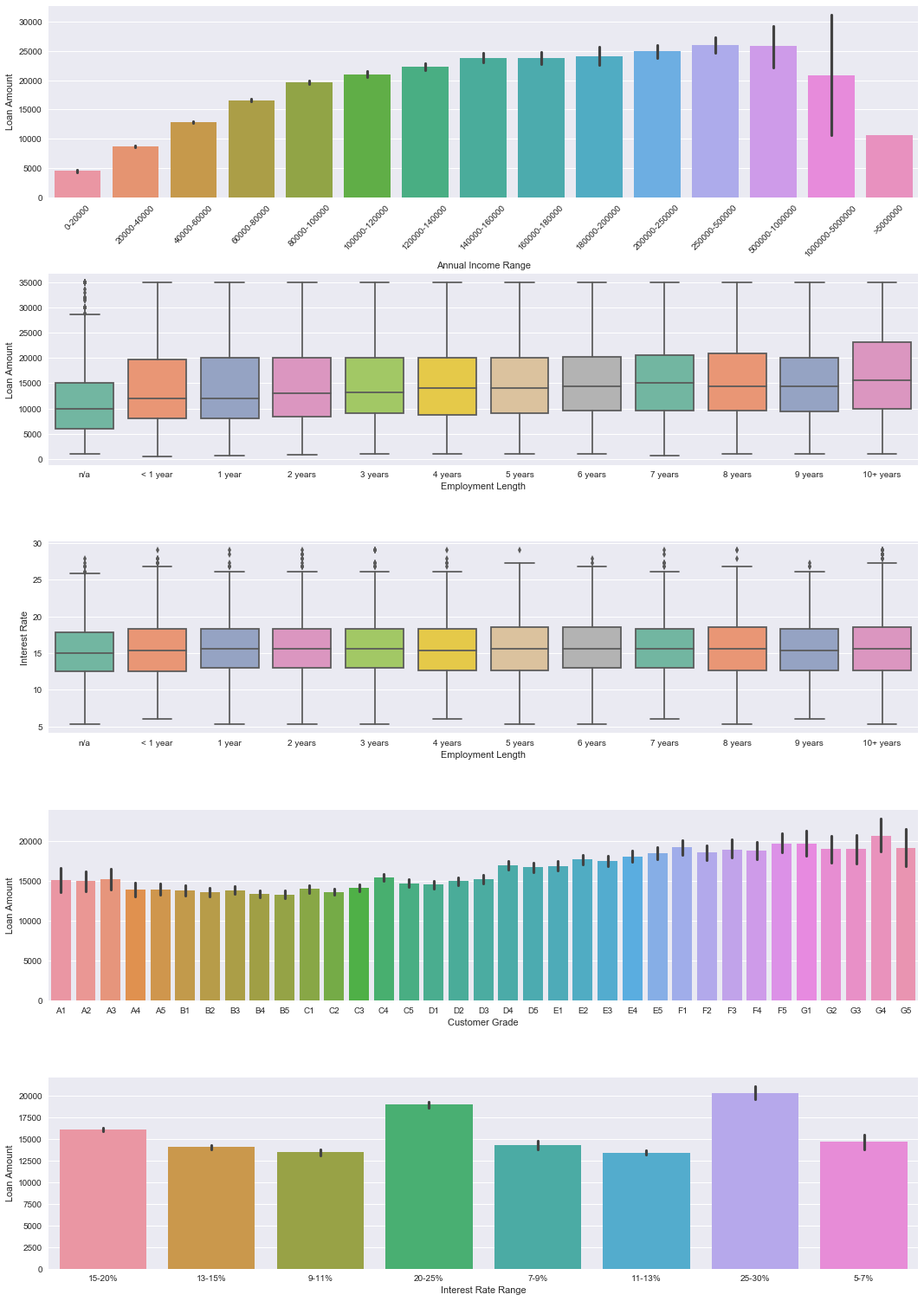


Figure :Analyzing Default loans

In above figure 11, the first plot shows the loan amount for each income range that we have created. We can see that the requirement for loan amount is increasing for income range from 120K to 500K, after which it decreases. The second plot shows loan amount according to the employment length. We observed that people who have larger employer length usually borrow large amount of loan, which is mostly for mortgage purpose. The third plot shows the interest rate for each employment length. The fourth plot shows the loan amount according to subgrade. And the final 5th plot show the loan amount for each interest rate range.

## Data Cleaning, Encoding & Missing Data Imputation

Features present within the dataset provided an ample amount of information which we could use to identify relationships and gauge their effect upon the success or failure of a borrower fulfilling the terms of their loan agreement. We required only the variables that had a direct or indirect response to a borrower’s potential to default. To achieve this, we have prepared the data by choosing select variables that would best fit these criteria.

**4.1) Data Cleaning:**

As a first step, we removed all unique id fields which is to represent a loan request, since it does not contribute in data analysis or model building. Next, we removed all the features which has more than 50% missing data. After which removed few categorical features which had only one category in the data.

*There were few features in our dataset which were specific to the defaulted loans, we removed all the features since they possibly can create leakage in our model.*

**4.2) Encoding:**

After data cleaning, we encode all the categorical columns using in-built spark API’s. Our data consist of these types of categorical features: ordinal, nominal and binary. We use *StringIndexer API* to convert the ordinal features such as borrowers experience level and *VectorAssembler API* to convert the rest nominal features. Following which, we binary encoded the variables with binary values.

**4.3) Missing data imputation:**

While doing exploratory data analysis, we developed some business rules which helped us to handle the missing data imputation such as; 90% of the missing data in "*tot\_cur\_bal*", "*tot\_coll\_amt*" column can be filled with 0 cause their loan status is either "Fully Paid" or "Charged Off". Median value can be imputed for missing values in *total\_rev\_hi\_lim*, since 86% of the data has “0”.

After implying all imputation rules, we removed the observations from data set which still has missing data (which is approximately 0.8% of total records).

**4.4) Label creation:**

Our data doesn’t have the label column by itself but there is combination of the features which represent the default loan. For our classification model we needed to add a class label variable to our data set. We added a new column “*classlabel*” and populate the values based in the combinations of features such as *loan\_status, tot\_curr\_bal, delig\_month* etc. The class count for each class is shown in following table.

|  |  |
| --- | --- |
| Class Label | Count |
| 1(Defaulted) | 60153 |
| 0(Not Defaulted) | 819919 |

Additionally, we classified any loan that defaulted, were charged off, or were late on payments was classified as negative examples (defaulted), while we classified any loan that was fully paid or current was classified as positive examples (non-defaulted).

After labelling our data, we dropped the features which were used for labelling at the first place.

## Learning Algorithm Implementation

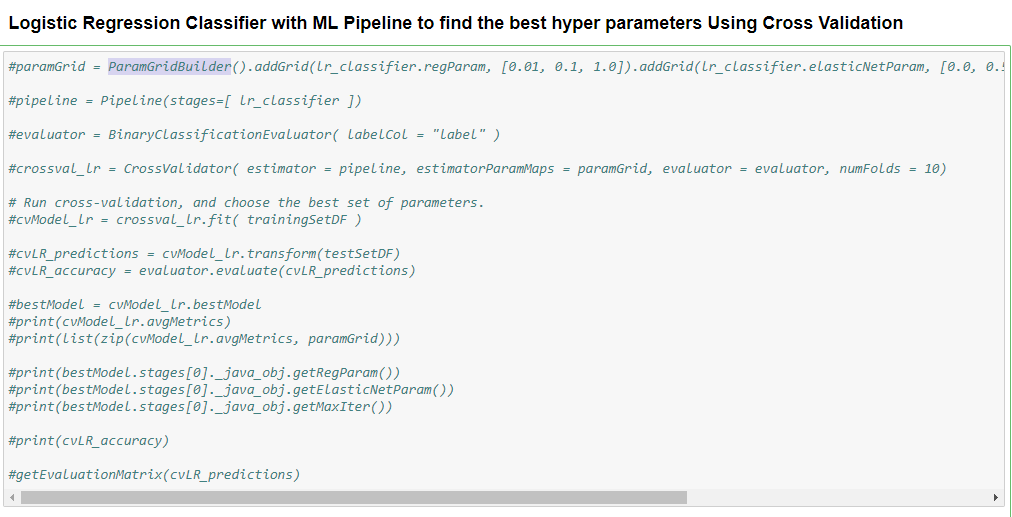
For this project, we built and tested five different classifier learning models. Each of our learning models was comprised by a different combination of the hyper parameters relevant to the individual learning models we learnt over the span of our degree courses. Each of these classifiers are implemented in Apache Spark’s machine learning framework (MLlib) which uses the computational power from spark-core, we used pyspark’s libraries for these implementations. pyspark is an API developed in python for spark programming and writing spark applications in Python style, but underlying execution model is the distributed in-memory computing framework provided by spark-core.

Furthermore, to tune/optimize our learning model’s hyper parameters, we implemented Spark’s machine learning pipeline using parameter grid search to optimize our learning hyper parameters. A Spark ML Pipeline is specified as a sequence of stages, and each stage is either a transformer (Feature Vectorization, Selection, Feature Encoding etc.) or an Estimator (Learning classifier, Model Evaluator). These stages are run in order, and the input spark DataFrame is transformed as it passes through each stage. This pipeline internally creates the DAG (directed acyclic graph) of stages consist of transformers and estimators. To ensure faster processing speed, ML Pipeline employs spark’s parallel computation power to run each stage on individual partition of the data and pass-through processed data to next stage instead of computing a stage on the whole input data. In case of a node failure, Spark usages the DAG (Logical execution plans – Lineage) to recover the data and re-commute the stages.

To choose best hyper parameter, we implemented cross validation with spark ML pipeline and evaluated each model using the binary model evaluator. Binary model evaluator is the built-in implementation in spark ml package, which helps to validate the cross-validation results.

*CrossValidator* class provides a function “*bestmodel*” which returned the “*pipelinemodel*” with the best selected grid parameters (Hyper parameters). This pipeline model can then be cast to the original learning classifier, which was added in the pipeline as estimator. Please note, that this functionality has a different implementation in pyspark (than Scala/java), where one must call the stage array on the pipeline model object based on the index of our estimator in stage array. Once, we get our estimator object for best model, we can call underline “\_java\_obj” to access the best values for our individual hyper parameters, which were added to the parameter grid. (This implementation is specific to pyspark, other languages such as Java/Scala has separate way to do this).

Following is the snap shot of the code for better explanation –



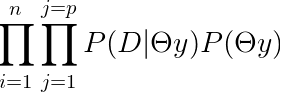
***Note:*** *When we use ML pipeline with cross validation to select the best hyper parameter, spark internally runs multiple different copies of model with combinations of given hyper parameters. For example, if we are adding 3 parameters in the parameter grid; each with 3 distinct values, spark will compare 27(3^3) different model. Furthermore, we are running 10 fold cross validation to evaluate the best model which increases our computations 10 fold. Therefore, we ran the pipeline only once to find the best params and using them in the individual classifier. We have commented this code in the final code submission to save the computational time. However, you can uncomment and run if you want to test this code. This usually takes 30-45 mins to run with 2 executor core assigned with 4GB memory.*

**5.1) Naïve Bayes: -**To start with our model building, we wanted to include a generative machine learning model (probabilistic classifier) in our trails. Naïve Bayes is a simple probabilistic classifier based on Bayes theorem with strong independence assumptions between the features. The intention behind using this model was to handle the class imbalance issue by adding the default prior belief (beta distribution function) to calculate the max posterior probability.

We used pyspark.ml.classifier package to implement this algorithm with only one optimized hyper parameter for Laplace smoothing. Following are the hyper parameters we tested and added in the param grid search.

*smoothing*– [1.0, 2.0, 3.0]

***Note:*** *- As we notice from the results, this algorithm is not performing well as compared to others. We tried to drill down the root cause for the issue, but the spark provides very abstract level API for the algorithm implementation and doesn’t provide much control on its implementation. As a side work, we try implementing this same model in R and Python, where we first calculated the Maximum Likelihood Estimator based on the distribution of each feature and used combinations of Alpha & Beta values (For beta distribution) for prior. We then calculated the max posterior probability to each observation and achieved 92.6% precision. Below is the likelihood function which we implemented.*

******

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | Actual | | | ROC Curve |
| Predicted |  | Positive | | Negative |  |
| Positive | TP = 122845 | | FP = 123041 |
| Negative | FN = 7406 | | TN = 10733 |
|  |  |  | |  |
| Accuracy | | | 50% | |
| Precision | | | 49.96 % | |
| Recall | | | 94.31 % | |
| F1 Score | | | 65.31 % | |
| Sensitivity | | | 94.31 | |
| Specificity | | | 8.023 | |
| ROC Score | | | 0.551 | |

Table 1:Result Summary table for Naive Bayes

**5.2) Logistic Regression: -** Logistic regression is a popular method to predict a categorical/binary response. It is a specialized case of “Generalized Linear Models” that predicts the probability of the outcomes. The main intention behind using this learning algorithm is to handle the class imbalance. Logistic regression is one of the implementation of linear models, for which spark provides ability to handle the class imbalance by adding a class weights. Apparently, spark doesn’t provide any built-in implementation to handle class imbalancing for other nonlinear machine learning models.

We used pyspark.ml.classifierpackage to implement the model with optimized hyper parameter. We also implemented “elasticnet” regularization since it solves the limitations of both L1 and L2 regularization. Following are the hyper parameters we tested and added in the param grid search.

*regParam* – [0.01, 0.1, 1.0] ,*elasticNetParam* - [0.0, 0.5, 1.0], *maxIter* - [1, 5, 10]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Actual | | ROC Curve |
| Predicted |  | Positive | Negative |  |
| Positive | TP = 236174 | FP = 9712 |
| Negative | FN = 2375 | TN = 15764 |
|  |  |  |  |
| Accuracy | | 95 % | |
| Precision | | 96.05 % | |
| Recall | | 99.00 % | |
| F1 Score | | 97.50 % | |
| Sensitivity | | 99.00 | |
| Specificity | | 61.87 | |
| ROC score | | 0.978 % | |

Table 2:Result Summary table for Logistic Regression

**5.3) Random Forest:** The next algorithm we tested was Random Forest (the ensembles of decision trees) to further deal with the data imbalance issue. Random forest is the aggregation of multiple decision trees which uses entropy/Gini to find the impurities, which makes it less sensitive to the class imbalance. The Spark implementation supports random forest for binary as well as multiclass classifications.

We used pyspark.ml.classifierpackage to implement this algorithm with optimized hyper parameters. We also tested our model with different bin sizes and tree depths. Following are the hyper parameters we tested and added in the param grid search.

*maxBins*– [25, 28, 31, 34],*maxDepth*- [4, 6, 8, 10], *impurity*- ["entropy", "gini"]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Actual | | ROC Curve |
| Predicted |  | Positive | Negative |  |
| Positive | TP = 245886 | FP = 0 |
| Negative | FN = 44 | TN = 18095 |
|  |  |  |  |
| Accuracy | | 99% | |
| Precision | | 100.0 % | |
| Recall | | 99.98 % | |
| F1 Score | | 99.99 % | |
| Sensitivity | | 99.98 % | |
| Specificity | | 100.0 | |
| ROC Score | | 1.0 | |

Table 3:Result Summary table for Logistic Random Forest

**5.4) Gradient Boosted Trees: -** In contrary to random forest, which tried to minimize the error by reducing the variance, we tested the opposite way by reducing the bias. Boosting reduces error mainly by reducing bias and to some extent variance, by aggregating the output from many models. GBM is a boosting method, which builds on [weak classifiers](https://stats.stackexchange.com/questions/82049/what-is-meant-by-weak-learner). The idea is to add a classifier at a time, so that the next classifier is trained to improve the already trained ensemble in sequential order. On the other hand, for RF each iteration the classifier is trained independently from the rest in parallel.

We used pyspark.ml.classifier package to implement this algorithm with optimized hyper parameter. We also tested our model with different step sizes for gradient descent and tree debts. Following are the hyper parameters we tested and added in the param grid search.

*stepSize*– [0.01, 0.1, 1.0],*maxIter*- [15, 20, 25], *maxDepth*- [5, 10, 15]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | Actual | | | ROC Curve |
| Predicted |  | Positive | | Negative |  |
| Positive | TP = 245886 | | FP = 0 |
| Negative | FN = 0 | | TN = 18139 |
| Accuracy | | | 100% | |
| Precision | | | 100.0 % | |
| Recall | | | 100.0 % | |
| F1 Score | | | 100.0 % | |
| Sensitivity | | | 100.0 | |
| Specificity | | | 100.0 | |
| ROC score | | | 1.0 | |

Table 4:Result Summary table for Gradient Boosted Trees

**5.5) Support Vector Machine (SVM):** Finally, we attempted Support Vector Machines to further expand our learning about spark machine learning framework. This is another type of discriminative classifier which construct hyper plane in a high or infinite dimensional space which is used to define the prediction boundaries for our classification model. This implementation of linear model in Spark MLLib framework provides ability to handle the class imbalance by adding weights to our prediction class.

We used pyspark.ml.classifier package to implement this algorithm with optimized hyper parameter for penalty term for L-2 regularization (C) and gamma values. Following are the hyper parameters we tested and added in the param grid search.

*regParam*– [0.01, 0.1, 1.0], *maxIter*– [50, 75, 100], *threshold*- [0.01, 0.1, 1.0]

## Useful-ness of the model

As part of the Lending Club’s revenue generation model, it charges an origination fee to the borrowers and service fee to the investors. Our model can be useful to provide comprehensive analysis of the historical data as well as a smart prediction about the investor’s money to lower their risk. By developing a nearly perfect prediction model, we would hope to reduce the number of delinquencies in the investment and helps genuine borrowers to maintain their credit ratings. This would help lending club to engage more investors and borrowers in their platform, hence increasing the revenue growth.

Furthermore, we would like to highlight few recommendations noticed from our exploratory data analysis:

1. The higher the loan amount, the higher the likelihood of default. Investors should invest in loans that are approximation $9000 or less.
2. Loans with term of 36 months tended to be defaulted a lot more than loans with term of 60 months. Investor should invest in long terms.
3. Certain sub-grades were almost likely to default compared to other sub-grades. Selecting loans of subgrade B5 and higher will result in a 90% chance of repayment.

## Summary

If lending club must provide the more precise information to the investors to reduce their investment risk, they need to provide the default likeliness of borrower more accurately. Should this model be used in real life, our goal is to retain more investors for Lending Club. Therefore, we focused on precision and specificity of our model.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy (%) | Precision | Recall | F1 Score |
| Naïve Bayes | 50 | 49.96 | 94.31 | 65.32 |
| Logistic Regression | 95 | 96.05 | 99.0 | 97.50 |
| Random Forest | 99 | 100.0 | 99.9 | 99.99 |
| Gradient Boosting Classifier | 100 | 100.0 | 100.0 | 100.00 |

## Future Work

Under the scope of this project, we currently tried to handle class imbalance issue by providing class weights, because as of this moment, apache spark’s existing MLlib framework only allows this technique with only few algorithms. In the future, we would like to try out innovative approaches to handle class imbalance using SMOTE and balanced bagging techniques. Furthermore, spark’s machine learning framework has less algorithm coverage as compared to other frameworks as such as Scikit-learn, Tensor-Flow, XGBoost etc. We would also like to integrate these frameworks with spark to utilize its distributed computational power along with vast coverage of machine learning algorithms.

Last but not the least, we would hope to modify the goal of our modelling and try to predict the customer’s credit rating (grade/sub grade) based on their credit loan history. This would extend our learnings to solve a multi-class problem.

## References

<https://www.lendingclub.com/>

<https://www.kaggle.com/wendykan>

<https://data.world/lpetrocelli/lendingclub-loan-data-2017-q-1>

<https://www.wikipedia.org/>

<https://spark.apache.org/docs/2.2.0/ml-guide.html>

<https://spark.apache.org/docs/2.3.0/api/python/pyspark.html>

<https://stackoverflow.com>