Show Code

# Machine Learning Engineer Nanodegree

## Capstone Project

Rahul Sharma

January 23rd, 2018

## I. Definition

*(approx. 1-2 pages)*

In an attempt to explore a problem most closely resembling the work we do at Capital One, I proposed to work on a problem related to credit approval using the data from Lending Club.

This is a well known challenge1 (http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0139427) pertaining to the marketplace which matches borrowers seeking loans to investors offering funds.

As a typical credit risk problem, I am primarily interested in identifying characteristics that can help identify factors leading to loan defaults (not fully repaying the loan). I have attempted to predict such behavior using various attributes describing credit worthiness of a borrower. These characteristics include (but are not limited to) income, geography, employment, purpose of loan. I avoided using any prior lending history of the borrower which is not available at the time of application or which can change over time.

Based on the eligibility of a borrower, the loans are either approved or declined. If approved, the loans may have variable interest rates based on the risk profiling. I find the challenge of predicting human behavior quite interesting and because this deals with the type of data we usually handle at Capital One, I thought this may be a fruitful exercise to undertake for the capstone project.

Details of how Lending Club works can be found here (https://www.lendingclub.com/public/how-peer-lending-works.action).

## Project Overview

In this section, look to provide a high-level overview of the project in layman's terms. Questions to ask yourself when writing this section:

- *Has an overview of the project been provided, such as the problem domain, project origin, and related datasets or input data?*
- *Has enough background information been given so that an uninformed reader would understand the problem domain and following problem statement?*

The primary motivation for this exercise is to identify how much money could have been saved if we have a model that can identify the loan defaults appropriately based on data available to us at the time of lending.

It is a supervised classifiation problem where we are trying detect if the borrower defaults on repaying the loan given the historical data provided to us and the outcome of that decision.

We will make use of close to 200,000 data examples which will use to learn what characteristics contribute towards someone not being able to keep up on loan payments and we use out of time data to test how well our model holds ground when used on a dataset it hasn't previously seen.

For the purpose of this exercise, we will focus on two questions:

1. How well our model predicts loan defaults? To predict the accuracy, we will take into consideration Area under the ROC along with F1 ratio. We use F1 ratio because based on personal experience, I know that such datasets are highly imbalanced where the default rates are usually less than *20%*.
2. If our model were to be used, how much money would Lending Club be able to lose/save by rejecting the defaulted loans and save/lose accepting the loans the loans that our model predicts to be paid in full?

We will build a model to first predict whether the borrower will fully repay the loan and based on that prediction, we will calculate the dollar amount of money saved by accepting/rejecting these loan requests with the model. Similarly. I plan to explore if we can combine this with the profits lost in rejecting good loans. We will use Lending Club's Open Data to obtain the probability of a loan request defaulting or being charged off.

### Datasets

The datasets required to conduct the analysis are available at Lending Club (https://www.lendingclub.com/info/download-data.action) where the data has been broken down by both Accepted and Rejected as well as origination year/quarter of the loan originations.

- **Time Period** : The loan level data is available starting from 2007 until 2016 broken by years and quarters.
- **Attributes** : As a quick overview, it appears that there are close to 145 attributes available including credit history for the borrower, funding amount, origination dates and interest rate.
- **Number of Borrowers** : We will make use of the loan files which consists of 200,000 borrowers
    - We will take these dataset into consideration by breaking them into training set, validation set for hyper-parameter tuning and both a test set as well as out of time test set to see the effectiveness of our model over long term. The distribution of train, validation and test set will be kept at 85%, 15% and 15% respectively.

    - To ensure that our models are robust over time, we will also make use of out of time sample, this is the data that will not be used to train the model.
    - We will exclude any attributes that are not available at the time of loan approval. These include a bunch of variables which are provided from credit bureau(s).
    - When I last reviewed this data (2013), I recall having FICO scores which have since been removed from the files (or may be available only to the lenders).
- Considerations:
    - Usually, in such type of data, there are only a small percentage of borrowers who don't fully repay their loans. These can be between 10%-20% and such type of problem is generally known as class imbalance.
    - There are several ways to handle class imbalance, specifically oversampling of the minority class, undersampling of majority class or providing different weights to defaulters vs non-defaulters but these approaches can affect bias. Hence, we will first try not to perform such operations and if model performance needs to be improved, one of these techniques will be applied to improve model performance.
- **Software** : I will be making use of open source language Python, specifically sklearn library which has some of the algorithms implemented for easier use.

- References:

  HBS Summary of Lending Club (http://www.hbs.edu/openforum/openforum.hbs.org/goto/challenge/understand-digital-transformation-of-business/lending-club-using-data-to-redefine-the-credit-score-and-improve-access-to-consumer-credit.html)

## Problem Statement

In this section, you will want to clearly define the problem that you are trying to solve, including the strategy (outline of tasks) you will use to achieve the desired solution. You should also thoroughly discuss what the intended solution will be for this problem. Questions to ask yourself when writing this section:

- *Is the problem statement clearly defined? Will the reader understand what you are expecting to solve?*
- *Have you thoroughly discussed how you will attempt to solve the problem?*
- *Is an anticipated solution clearly defined? Will the reader understand what results you are looking for?*

The applicable solution to the problem will be a classification model which most accurately identify the loans which have the highest potential to default.

As a typical *Supervised Learning* problem, we will try to explore the data to build a model which uses the credit history, data on loans for users to predict whether the loan will default or not. We will make use of historical data for approved loans from 2007 to 2014 along with their **actual** outcome which serves as our labeled target that we intent to learn. Based on this historical data, our model shall be able to distinguish between a borrower whose characteristics match with those of a borrower who fully repays the loan vs another who doesn't.

Once we have a model which predicts such information, we will use the data for loans offered in the first quarter of 2016 to assess how well our algorithm was able to identify defaulter.

Next, I will also make use of these predictions to cross-check against the existing actual data to assess the total dollars Lending Club could have potentially saved (or lost) if our model was utilized.

It is to measure our model's effectiveness that we will conduct out of time testing which helps to assess if our model holds it's strength across time periods.

## Metrics

In this section, you will need to clearly define the metrics or calculations you will use to measure performance of a model or result in your project. These calculations and metrics should be justified based on the characteristics of the problem and problem domain. Questions to ask yourself when writing this section:

- *Are the metrics you've chosen to measure the performance of your models clearly discussed and defined?*
- *Have you provided reasonable justification for the metrics chosen based on the problem and solution?*

While the most popular measure of a model is **Accuracy**, it may not be the best measure for the type of problem which falls under the umbrella of class imbalance. Paying a close attention to the below formula leads us to observe that accuracy is a simple measure depicting the proportion of how many instances we were able to accurately identify. Given a class imbalance problem, where the event of interest only occurs at 20% of the time, it is easy for the algorithm to classify every instance as that belonging to the majority class and hence achieve 80% accuracy.

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n}\sum_{i=0}^{n-1} 1(\hat{y}_i = y_i)$$

$$\text{accuracy} = \frac{TruePositive(TP) + TrueNegative(TN)}{FalsePositive(FP) + FalseNegative(FN) + TruePositive(TP) + TrueNegative(TN)}$$

A better measure of the model's effectiveness would be to observe the correct classifications for both classes. In our case, this means that we should look at how many defaulters were accurately identified as well. A binary confusion matrix is an ideal way to gain such clarity. It has four boxes and diagonal boxes provide us the number of cases identified accurately for each class.

| | |
|---|---|
| True Positive (TP) | False Negative (FN) |
| False Positive (FP) | True Negative (TN) |

Another measure which we will use for our needs is called F1-ratio. The applicable solution to the problem will be a classification model which most accurately identify the loans which have the highest potential to default. To measure the effectiveness of the model, we will take into consider F1 score:

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

where:

$$precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

$$recall = \frac{TruePositive}{TruePositive + TrueNegative}$$

As previously discussed, credit risk problems tend to have imbalanced targets which requires us to look at how well we identify the imbalanced classes. A typical meausre like *Accuracy* can be misleading in such scenario. Hence, I decided to use F1 ratio for the purpose of this exercise. The main goal is to identify as many defaulted

loans accuratly as possible while not being overly restrictive to classify good loans as the ones which will default. Latter is important as it can lead to rejecting loans which can contribute towards the profit of the company. Our solution should minimize risk and not avoid risk by being overly strict around the decision criteria. To convey this point, it's important to keep in mind that easiest way to not have any risk in our portfolio is by not offering any loans. However, our intention is to offer loans while assessing the risk associated with the loans apporpriately.

# II. Analysis

*(approx. 2-4 pages)*

## Data Exploration

In this section, you will be expected to analyze the data you are using for the problem. This data can either be in the form of a dataset (or datasets), input data (or input files), or even an environment. The type of data should be thoroughly described and, if possible, have basic statistics and information presented (such as discussion of input features or defining characteristics about the input or environment). Any abnormalities or interesting qualities about the data that may need to be addressed have been identified (such as features that need to be transformed or the possibility of outliers). Questions to ask yourself when writing this section:

- *If a dataset is present for this problem, have you thoroughly discussed certain features about the dataset? Has a data sample been provided to the reader?*
- *If a dataset is present for this problem, are statistics about the dataset calculated and reported? Have any relevant results from this calculation been discussed?*
- *If a dataset is **not** present for this problem, has discussion been made about the input space or input data for your problem?*
- *Are there any abnormalities or characteristics about the input space or dataset that need to be addressed? (categorical variables, missing values, outliers, etc.)*

## Dependencies: Import Libraries to be used for analysis

```
================================================================================
```

```
2.7.14 |Anaconda, Inc.| (default, Dec  7 2017, 11:07:58)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
Pandas:  0.22.0
Numpy:  1.13.1
Seaborn:  0.8.1
Ipython Version: 5.4.1
sklearn:  0.19.0
Encoding: ascii
```

**Download Data Dictionary and Source Data for the challenge**

The data we plan to leverage for this exercise is available in the form of zip files and require us to download separate data files. As this is a repetitive task, I plan to leverage a function I have written to download and unzip the data files.

As there can be need for proxy working on a corporate network, one can leverage below statements to set the proxy to ensure that our code is able to download the necessary files.

In order to ensure that we are not running into any issues at the time of downloading, we first create a data directory (if it doesn't already exist'). Next, we unzip the file and provide the user with the location of unzipped file. This ensures that we can leverage this as an input to the next steps.

Prior to downloading any data, we first download the **dictionary** or the **metadata** file which contains the definition of the fields which we will be leveraging for our exercise.

```
Downloading LCDataDictionary.xlsx from https://resources.lendingclub.co
m/LCDataDictionary.xlsx


File download complete in 0.03s

Downloaded is available in the following directory:
/Users/xtl476/Desktop/OneDrive - Capital One Financial Corporation/CRM/
Trainings/Udacity/capstone_project/final_submission_v0/data
```

Next, we download all the necessary data files. Here we download files from 2007 to 2014. The data for 2015 seems to be corrupted, hence we will leverage first quarter of 2016 loans to test our model. This is actually better as we will be able to test our model not only on the data it hasn't seen but also from a completely different time period. This implies that we will be learning from data in one economic period while testing it in another leading to a better measurement of our model's effectiveness.

```
Downloading LoanStats3a.csv.zip from https://resources.lendingclub.com/
LoanStats3a.csv.zip


File download complete in 0.29s

Downloaded file is a zip file, unzipping...

File LoanStats3a.csv.zip has been unzipped

Downloaded is available in the following directory:
/Users/xtl476/Desktop/OneDrive - Capital One Financial Corporation/CRM/
Trainings/Udacity/capstone_project/final_submission_v0/data

Downloading LoanStats3b.csv.zip from https://resources.lendingclub.com/
LoanStats3b.csv.zip


File download complete in 1.00s

Downloaded file is a zip file, unzipping...

File LoanStats3b.csv.zip has been unzipped

Downloaded is available in the following directory:
/Users/xtl476/Desktop/OneDrive - Capital One Financial Corporation/CRM/
Trainings/Udacity/capstone_project/final_submission_v0/data

Downloading LoanStats3c.csv.zip from https://resources.lendingclub.com/
LoanStats3c.csv.zip


File download complete in 1.09s

Downloaded file is a zip file, unzipping...

File LoanStats3c.csv.zip has been unzipped

Downloaded is available in the following directory:
/Users/xtl476/Desktop/OneDrive - Capital One Financial Corporation/CRM/
Trainings/Udacity/capstone_project/final_submission_v0/data

Downloading LoanStats_2016Q1.csv.zip from https://resources.lendingclu
b.com/LoanStats_2016Q1.csv.zip


File download complete in 0.69s

Downloaded file is a zip file, unzipping...

File LoanStats_2016Q1.csv.zip has been unzipped

Downloaded is available in the following directory:
/Users/xtl476/Desktop/OneDrive - Capital One Financial Corporation/CRM/
Trainings/Udacity/capstone_project/final_submission_v0/data
```

After download the data, I observed that each file had a summary of loans towards the end which meant that we need to perform some clean-up of deleting empty rows (and rows followed by empty rows). In the LoanStats3a.csv, the empty rows were also followed by loans which did not meet the credit policy. As we wanted to ensure that we are only using the loands which meet the credit policy, I chose to delete such rows as well.

**Metadata Review:**

It is important to first understand the source and definitions of each attribute available to us, hence, I will conduct a thorough review on every field available to us. It is quite likely that the data available to us is not all from the same time period, meaning that application stage data is mixed with the recent transactions, credit activity of a borrower.

- We will only keep the fields which are available to us at the time of applications.
- Reviewing the data dictionary will help us make that distinction clear.
- As we want to avoid data leakage by passing any fields to our future models which are not available at the time of application, we will carefully review the metadata first.

```
The loan_data_dict has 151 rows and 2 features
```

Out[13]:

| | Attribute | Definition |
|---|---|---|
| **146** | settlement_status | The status of the borrower’s settlement plan. ... |
| **147** | settlement_date | The date that the borrower agrees to the settl... |
| **148** | settlement_amount | The loan amount that the borrower has agreed t... |
| **149** | settlement_percentage | The settlement amount as a percentage of the p... |
| **150** | settlement_term | The number of months that the borrower will be... |

Out[14]:

| | Attribute | Definition |
|---|---|---|
| **0** | acc_now_delinq | The number of accounts on which the borrower i... |
| **1** | acc_open_past_24mths | Number of trades opened in past 24 months. |
| **2** | addr_state | The state provided by the borrower in the loan... |
| **3** | all_util | Balance to credit limit on all trades |
| **4** | annual_inc | The self-reported annual income provided by th... |
| **5** | annual_inc_joint | The combined self-reported annual income provi... |

| | Attribute | Definition |
|---|---|---|
| **6** | application_type | Indicates whether the loan is an individual ap... |
| **7** | avg_cur_bal | Average current balance of all accounts |
| **8** | bc_open_to_buy | Total open to buy on revolving bankcards. |
| **9** | bc_util | Ratio of total current balance to high credit/... |
| **10** | chargeoff_within_12_mths | Number of charge-offs within 12 months |
| **11** | collection_recovery_fee | post charge off collection fee |
| **12** | collections_12_mths_ex_med | Number of collections in 12 months excluding m... |
| **13** | delinq_2yrs | The number of 30+ days past-due incidences of ... |
| **14** | delinq_amnt | The past-due amount owed for the accounts on w... |
| **15** | desc | Loan description provided by the borrower |
| **16** | dti | A ratio calculated using the borrower’s total ... |
| **17** | dti_joint | A ratio calculated using the co-borrowers' tot... |
| **18** | earliest_cr_line | The month the borrower's earliest reported cre... |
| **19** | emp_length | Employment length in years. Possible values ar... |
| **20** | emp_title | The job title supplied by the Borrower when ap... |
| **21** | fico_range_high | The upper boundary range the borrower’s FICO a... |
| **22** | fico_range_low | The lower boundary range the borrower’s FICO a... |
| **23** | funded_amnt | The total amount committed to that loan at tha... |
| **24** | funded_amnt_inv | The total amount committed by investors for th... |
| **25** | grade | LC assigned loan grade |
| **26** | home_ownership | The home ownership status provided by the borr... |
| **27** | id | A unique LC assigned ID for the loan listing. |

| | Attribute | Definition |
|---|---|---|
| **28** | il_util | Ratio of total current balance to high credit/... |
| **29** | initial_list_status | The initial listing status of the loan. Possib... |
| **30** | inq_fi | Number of personal finance inquiries |
| **31** | inq_last_12m | Number of credit inquiries in past 12 months |
| **32** | inq_last_6mths | The number of inquiries in past 6 months (excl... |
| **33** | installment | The monthly payment owed by the borrower if th... |
| **34** | int_rate | Interest Rate on the loan |
| **35** | issue_d | The month which the loan was funded |
| **36** | last_credit_pull_d | The most recent month LC pulled credit for thi... |
| **37** | last_fico_range_high | The upper boundary range the borrower’s last F... |
| **38** | last_fico_range_low | The lower boundary range the borrower’s last F... |
| **39** | last_pymnt_amnt | Last total payment amount received |
| **40** | last_pymnt_d | Last month payment was received |
| **41** | loan_amnt | The listed amount of the loan applied for by t... |
| **42** | loan_status | Current status of the loan |
| **43** | max_bal_bc | Maximum current balance owed on all revolving ... |
| **44** | member_id | A unique LC assigned Id for the borrower member. |
| **45** | mo_sin_old_il_acct | Months since oldest bank installment account o... |
| **46** | mo_sin_old_rev_tl_op | Months since oldest revolving account opened |
| **47** | mo_sin_rcnt_rev_tl_op | Months since most recent revolving account opened |
| **48** | mo_sin_rcnt_tl | Months since most recent account opened |
| **49** | mort_acc | Number of mortgage accounts. |
| **50** | mths_since_last_delinq | The number of months since the borrower's last... |

| | Attribute | Definition |
|---|---|---|
| **51** | mths_since_last_major_derog | Months since most recent 90-day or worse rating |
| **52** | mths_since_last_record | The number of months since the last public rec... |
| **53** | mths_since_rcnt_il | Months since most recent installment accounts ... |
| **54** | mths_since_recent_bc | Months since most recent bankcard account opened. |
| **55** | mths_since_recent_bc_dlq | Months since most recent bankcard delinquency |
| **56** | mths_since_recent_inq | Months since most recent inquiry. |
| **57** | mths_since_recent_revol_delinq | Months since most recent revolving delinquency. |
| **58** | next_pymnt_d | Next scheduled payment date |
| **59** | num_accts_ever_120_pd | Number of accounts ever 120 or more days past due |
| **60** | num_actv_bc_tl | Number of currently active bankcard accounts |
| **61** | num_actv_rev_tl | Number of currently active revolving trades |
| **62** | num_bc_sats | Number of satisfactory bankcard accounts |
| **63** | num_bc_tl | Number of bankcard accounts |
| **64** | num_il_tl | Number of installment accounts |
| **65** | num_op_rev_tl | Number of open revolving accounts |
| **66** | num_rev_accts | Number of revolving accounts |
| **67** | num_rev_tl_bal_gt_0 | Number of revolving trades with balance >0 |
| **68** | num_sats | Number of satisfactory accounts |
| **69** | num_tl_120dpd_2m | Number of accounts currently 120 days past due... |
| **70** | num_tl_30dpd | Number of accounts currently 30 days past due ... |
| **71** | num_tl_90g_dpd_24m | Number of accounts 90 or more days past due in... |
| **72** | num_tl_op_past_12m | Number of accounts opened in past 12 months |
| **73** | open_acc | The number of open credit lines in the borrowe... |

| | Attribute | Definition |
|---|---|---|
| **74** | open_acc_6m | Number of open trades in last 6 months |
| **...** | ... | ... |
| **76** | open_il_24m | Number of installment accounts opened in past ... |
| **77** | open_act_il | Number of currently active installment trades |
| **78** | open_rv_12m | Number of revolving trades opened in past 12 m... |
| **79** | open_rv_24m | Number of revolving trades opened in past 24 m... |
| **80** | out_prncp | Remaining outstanding principal for total amou... |
| **81** | out_prncp_inv | Remaining outstanding principal for portion of... |
| **82** | pct_tl_nvr_dlq | Percent of trades never delinquent |
| **83** | percent_bc_gt_75 | Percentage of all bankcard accounts > 75% of l... |
| **84** | policy_code | publicly available policy_code=1\nnew products... |
| **85** | pub_rec | Number of derogatory public records |
| **86** | pub_rec_bankruptcies | Number of public record bankruptcies |
| **87** | purpose | A category provided by the borrower for the lo... |
| **88** | pymnt_plan | Indicates if a payment plan has been put in pl... |
| **89** | recoveries | post charge off gross recovery |
| **90** | revol_bal | Total credit revolving balance |
| **91** | revol_util | Revolving line utilization rate, or the amount... |
| **92** | sub_grade | LC assigned loan subgrade |
| **93** | tax_liens | Number of tax liens |
| **94** | term | The number of payments on the loan. Values are... |
| **95** | title | The loan title provided by the borrower |
| **96** | tot_coll_amt | Total collection amounts ever owed |
| **97** | tot_cur_bal | Total current balance of all accounts |

| | Attribute | Definition |
|---|---|---|
| **98** | tot_hi_cred_lim | Total high credit/credit limit |
| **99** | total_acc | The total number of credit lines currently in ... |
| **100** | total_bal_ex_mort | Total credit balance excluding mortgage |
| **101** | total_bal_il | Total current balance of all installment accounts |
| **102** | total_bc_limit | Total bankcard high credit/credit limit |
| **103** | total_cu_tl | Number of finance trades |
| **104** | total_il_high_credit_limit | Total installment high credit/credit limit |
| **105** | total_pymnt | Payments received to date for total amount funded |
| **106** | total_pymnt_inv | Payments received to date for portion of total... |
| **107** | total_rec_int | Interest received to date |
| **108** | total_rec_late_fee | Late fees received to date |
| **109** | total_rec_prncp | Principal received to date |
| **110** | total_rev_hi_lim | Total revolving high credit/credit limit |
| **111** | url | URL for the LC page with listing data. |
| **112** | verification_status | Indicates if income was verified by LC, not ve... |
| **113** | verified_status_joint | Indicates if the co-borrowers' joint income wa... |
| **114** | zip_code | The first 3 numbers of the zip code provided b... |
| **115** | revol_bal_joint | Sum of revolving credit balance of the co-bor... |
| **116** | sec_app_fico_range_low | FICO range (high) for the secondary applicant |
| **117** | sec_app_fico_range_high | FICO range (low) for the secondary applicant |
| **118** | sec_app_earliest_cr_line | Earliest credit line at time of application f... |
| **119** | sec_app_inq_last_6mths | Credit inquiries in the last 6 months at time... |
| **120** | sec_app_mort_acc | Number of mortgage accounts at time of applic... |

| | Attribute | Definition |
|---|---|---|
| **121** | sec_app_open_acc | Number of open trades at time of application ... |
| **122** | sec_app_revol_util | Ratio of total current balance to high credit... |
| **123** | sec_app_open_act_il | Number of currently active installment trades... |
| **124** | sec_app_num_rev_accts | Number of revolving accounts at time of appli... |
| **125** | sec_app_chargeoff_within_12_mths | Number of charge-offs within last 12 months a... |
| **126** | sec_app_collections_12_mths_ex_med | Number of collections within last 12 months e... |
| **127** | sec_app_mths_since_last_major_derog | Months since most recent 90-day or worse rati... |
| **128** | hardship_flag | Flags whether or not the borrower is on a hard... |
| **129** | hardship_type | Describes the hardship plan offering |
| **130** | hardship_reason | Describes the reason the hardship plan was off... |
| **131** | hardship_status | Describes if the hardship plan is active, pend... |
| **132** | deferral_term | Amount of months that the borrower is expected... |
| **133** | hardship_amount | The interest payment that the borrower has com... |
| **134** | hardship_start_date | The start date of the hardship plan period |
| **135** | hardship_end_date | The end date of the hardship plan period |
| **136** | payment_plan_start_date | The day the first hardship plan payment is due... |
| **137** | hardship_length | The number of months the borrower will make sm... |
| **138** | hardship_dpd | Account days past due as of the hardship plan ... |
| **139** | hardship_loan_status | Loan Status as of the hardship plan start date |
| **140** | orig_projected_additional_accrued_interest | The original projected additional interest amo... |

| | Attribute | Definition |
|---|---|---|
| **141** | hardship_payoff_balance_amount | The payoff balance amount as of the hardship p... |
| **142** | hardship_last_payment_amount | The last payment amount as of the hardship pla... |
| **143** | disbursement_method | The method by which the borrower receives thei... |
| **144** | debt_settlement_flag | Flags whether or not the borrower, who has cha... |
| **145** | debt_settlement_flag_date | The most recent date that the Debt_Settlement_... |
| **146** | settlement_status | The status of the borrower’s settlement plan. ... |
| **147** | settlement_date | The date that the borrower agrees to the settl... |
| **148** | settlement_amount | The loan amount that the borrower has agreed t... |
| **149** | settlement_percentage | The settlement amount as a percentage of the p... |
| **150** | settlement_term | The number of months that the borrower will be... |

151 rows × 2 columns

**Features to be kept:**

After a careful review of the fields available to us in the metadata, I shortlistd the number of features to only those that shall be available to us at the time of the application. I kept couple of features which are mostly from future activity of a borrower, but it's only for performing analysis, and it's not to be used for learning from the data.

Features kept to be used can be broadly divided into the following categories:

1. Loan features: e.g. Amount of loan, length of loan term etc.
2. Borrower characteristics: e.g. employment history, employment title, home ownership etc.
3. Quality of borrower: Grade
4. Geographical features: Zip code and State

```
Number of features retained: 19
Following columns will be kept from the original dataset:
```

Out[15]:

| | Attribute | Definition |
|---|---|---|
| **2** | addr_state | The state provided by the borrower in the loan... |
| **4** | annual_inc | The self-reported annual income provided by th... |
| **18** | earliest_cr_line | The month the borrower's earliest reported cre... |
| **19** | emp_length | Employment length in years. Possible values ar... |
| **20** | emp_title | The job title supplied by the Borrower when ap... |
| **23** | funded_amnt | The total amount committed to that loan at tha... |
| **25** | grade | LC assigned loan grade |
| **26** | home_ownership | The home ownership status provided by the borr... |
| **33** | installment | The monthly payment owed by the borrower if th... |
| **34** | int_rate | Interest Rate on the loan |
| **35** | issue_d | The month which the loan was funded |
| **36** | last_credit_pull_d | The most recent month LC pulled credit for thi... |
| **41** | loan_amnt | The listed amount of the loan applied for by t... |
| **42** | loan_status | Current status of the loan |
| **87** | purpose | A category provided by the borrower for the lo... |
| **94** | term | The number of payments on the loan. Values are... |
| **105** | total_pymnt | Payments received to date for total amount funded |
| **112** | verification_status | Indicates if income was verified by LC, not ve... |
| **114** | zip_code | The first 3 numbers of the zip code provided b... |

**Data Ingestion:**

We now make use of the function we had earlier defined to read the data in a *Pandas* dataframe.

```
['   42544 data/LoanStats3a.csv'] rows were reduced to ['   39788 data/
LoanStats2011.csv'] after clean-up
['  188187 data/LoanStats3b.csv'] rows were reduced to ['  188183 data/
LoanStats2013.csv'] after clean-up
['  235635 data/LoanStats3c.csv'] rows were reduced to ['  235631 data/
LoanStats2014.csv'] after clean-up
['  133893 data/LoanStats_2016Q1.csv'] rows were reduced to ['  133889
data/LoanStats2016.csv'] after clean-up
```

```
Dataset **lc_2011** has the following description:
There are 39786 rows and 19 fields

Dataset **lc_2013** has the following description:
There are 188181 rows and 19 fields

Dataset **lc_2014** has the following description:
There are 235629 rows and 19 fields

Dataset **lc_2016Q1** has the following description:
There are 133887 rows and 19 fields
```

As we are trying to merge the files from different time period, and even though we can see that each file has the 19 features that we wanted to read, we shall once again make sure that there aren't any features that are available in one file but not the other.

At the same time, we will also combine all the data from various files in one dataframe for ease of use.

```
set([])
set([])
set([])
set([])
set([])
set([])
(39786, 19)
(188181, 19)
(235629, 19)
(133887, 19)

Row count in Combined File == Sum of row count in all files: True
```

In the next step, I will write the data to disk in order to avoid re-downloading the files in case we run into Jupyter kernel issues.

```
Dataset **all_loans_data** has the following description:
There are 597483 rows and 19 fields
```

**Data Types:**

Prior to starting any analysis, it's vital to check the data types and ensure that they are coded as expected. An an example, we will like our numbers to be coded as Integers or Floats and not strings. Let's review the data types:

```
issue_d                 object
funded_amnt              int64
loan_amnt                int64
term                    object
installment            float64
total_pymnt            float64
loan_status             object
grade                   object
purpose                 object
earliest_cr_line        object
annual_inc             float64
verification_status     object
home_ownership          object
emp_length              object
emp_title               object
zip_code                object
addr_state              object
last_credit_pull_d      object
int_rate                object
dtype: object
```

It appears that a lot of columns one would expect to be numbers are coded as *object*. It could be due to the data was saved in the csv files from which we read the data or could be due to other reasons (unexpected characters).

**Sample Data:**

At the same time, it's always a good idea to review the sample data to ensure there are no unfo

Out[34]:

| | 597478 | 597479 | 597480 | 597481 | 59 |
|---|---|---|---|---|---|
| **issue_d** | Jan-2016 | Jan-2016 | Jan-2016 | Jan-2016 | Jan-2016 |
| **funded_amnt** | 6000 | 6000 | 14400 | 34050 | 5000 |
| **loan_amnt** | 6000 | 6000 | 14400 | 34050 | 5000 |
| **term** | 36 months | 36 months | 60 months | 36 months | 36 months |
| **installment** | 187.72 | 191.28 | 328.98 | 1187.21 | 164.22 |
| **total_pymnt** | 4314.93 | 4417.05 | 7227.02 | 27480.7 | 3773.94 |
| **loan_status** | Current | Current | Late (16-30 days) | Current | Current |
| **grade** | A | B | C | D | B |
| **purpose** | credit_card | debt_consolidation | credit_card | credit_card | home_improve |
| **earliest_cr_line** | Feb-2006 | Sep-1997 | Oct-1976 | Nov-2005 | Jul-2005 |
| **annual_inc** | 38000 | 32640 | 47000 | 87800 | 65000 |
| **verification_status** | Source Verified | Not Verified | Verified | Source Verified | Source Verifiec |
| **home_ownership** | OWN | RENT | RENT | MORTGAGE | MORTGAGE |
| **emp_length** | < 1 year | NaN | 10+ years | 10+ years | 7 years |
| **emp_title** | Warehouse Clerk | NaN | Meatcutter | Supervisor | carpenter |
| **zip_code** | 432xx | 600xx | 531xx | 212xx | 201xx |
| **addr_state** | OH | IL | WI | MD | VA |
| **last_credit_pull_d** | Dec-2017 | Dec-2017 | Dec-2017 | Dec-2017 | Dec-2017 |
| **int_rate** | 7.89% | 9.17% | 13.18% | 15.41% | 11.22% |

**Data Cleaning:**

It appears that our data needs substantial cleaning. We will use a mix of in-built functions and user-defined functions to clean the fields.

We will be performing the following steps:

- Check for nulls and if nulls are present in a significant volume, I prefer to create a flag which detects if the field was missing. At the same time, we will impute the missing values.
- Remove whitespace from verification status
- Convert "loan_status" to 0/1 (Booleans)
- Cleaning interest rate by removing % sign from the data
- Removing xxx from zip_code feature
- Remove text from emp_length feature and convert it to numbers for ease of calculation
- Convert date columns in the appropriate datetime object (issue_d, earliest_cr_line, last_credit_pull_d)

We start by checking the number of nulls/N/As in each of the fields that we plan to leverage to build our model

```
Out[135]: issue_d                   0
          funded_amnt               0
          loan_amnt                 0
          term                      0
          installment               0
          total_pymnt               0
          loan_status               0
          grade                     0
          purpose                   0
          earliest_cr_line          0
          annual_inc                0
          verification_status       0
          home_ownership            0
          emp_length            29928
          emp_title             36421
          zip_code                  0
          addr_state                0
          last_credit_pull_d       41
          int_rate                  0
          dtype: int64
```

Let's create missing flag before imputation and check if it matches the above numbers for correctness of implementation. As the proportion of missing is really small (<5%), in place of any complex criteria of imputation, (e.g. replacing with the most common categorical value), I will simply replace the value with "Missing". In a dataset with a low count of rows, this may result in what's called *Multicollinearity* but with a large dataset like ours and the low incidence rate of missing, I highly doubt if this shall cause any issues.

```
Out[136]: 0    567555
          1     29928
          Name: emp_length_mi_flag, dtype: int64
```

```
Out[137]: 0    561062
          1     36421
          Name: emp_title_mi_flag, dtype: int64



          Verified           203824
          Source Verified    200146
          Not Verified       193513
          Name: verification_status, dtype: int64

          Verified           203824
          Source_Verified    200146
          Not_Verified       193513
          Name: verification_status_mod, dtype: int64
```

Let's review the number of unique values for *emp_title* field, and if there are fairly large number of values, we can keep top 10 values and combine the remaining ones under the category "Other". This will reduce the information we have at hand, but it will make it easier to include such data in the model. Usually, there are many values with low instances, hence, combining them will not take away much from our model's ability to learn, rather it shall help our model generalize better.

```
Number of unique values for Emp Title:
234920

Missing             36421
Teacher              7945
Manager              6744
Registered Nurse     3356
RN                   3213
Owner                3210
Supervisor           2933
Sales                2533
Project Manager      2428
Driver               2258
Name: emp_title, dtype: int64

Other               526442
Missing              36421
Teacher               7945
Manager               6744
Registered Nurse      3356
RN                    3213
Owner                 3210
Supervisor            2933
Sales                 2533
Project Manager       2428
Driver                2258
Name: emp_title_mod, dtype: int64
```

Out[141]:

```
Other               526442
Missing              36421
Teacher               7945
Manager               6744
Registered_Nurse      3356
RN                    3213
Owner                 3210
Supervisor            2933
Sales                 2533
Project_Manager       2428
Driver                2258
Name: emp_title_mod, dtype: int64
```

All the cleaning leaves us with 41 null values in the column *last_credit_pull_d* which shall be easy to drop as we have over 500,000 rows of data.

```
Dataset **all_loans_data** has the following description:
There are 597483 rows and 23 fields

Dataset **all_loans_data** has the following description:
There are 597442 rows and 23 fields

issue_d                   0
funded_amnt               0
loan_amnt                 0
term                      0
installment               0
total_pymnt               0
loan_status               0
grade                     0
purpose                   0
earliest_cr_line          0
annual_inc                0
verification_status       0
home_ownership            0
emp_length                0
emp_title                 0
zip_code                  0
addr_state                0
last_credit_pull_d        0
int_rate                  0
emp_length_mi_flag        0
emp_title_mi_flag         0
verification_status_mod   0
emp_title_mod             0
dtype: int64
```

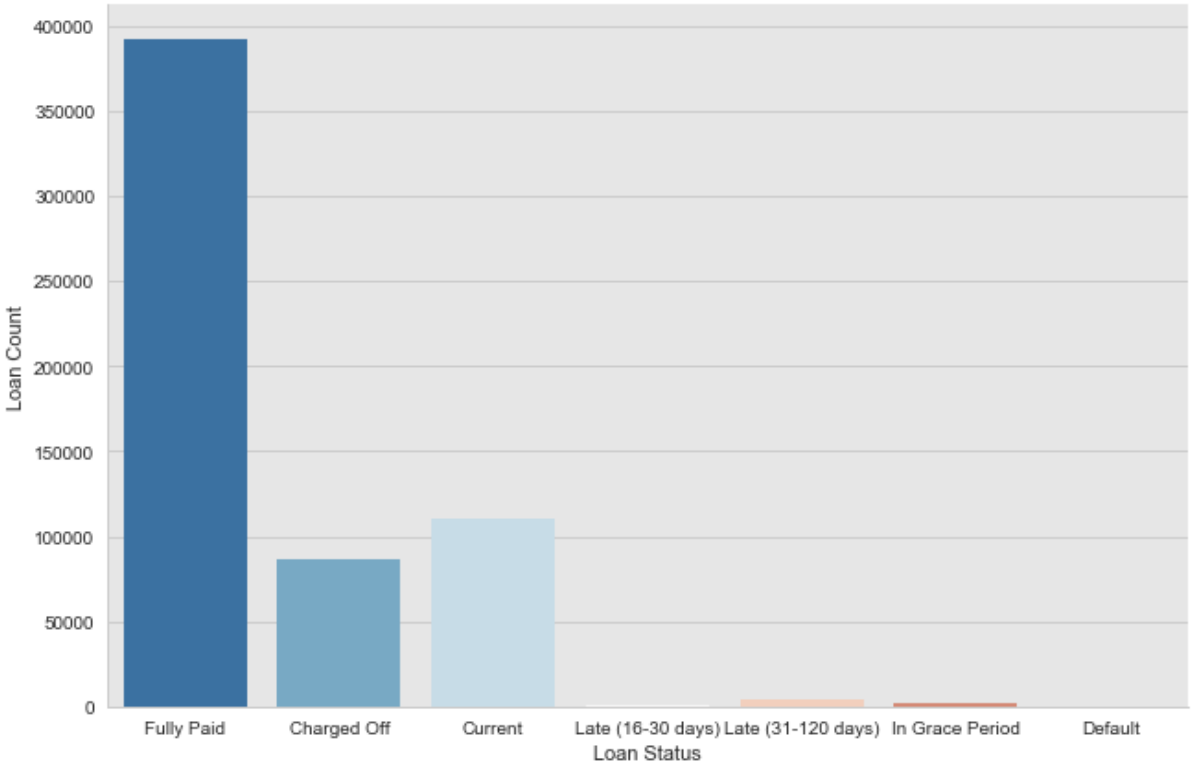
We will change the data type of *term* as it only consists of two values (36 and 60)

```
36    353079
60    103556
Name: term, dtype: int64
```

**Target: Loan Status**

It is time to take a closer look at the feature that we care the most about, which is, the **target** feature which we hope to predict accurately. In our data, it is denoted by the field name **loan_status**.

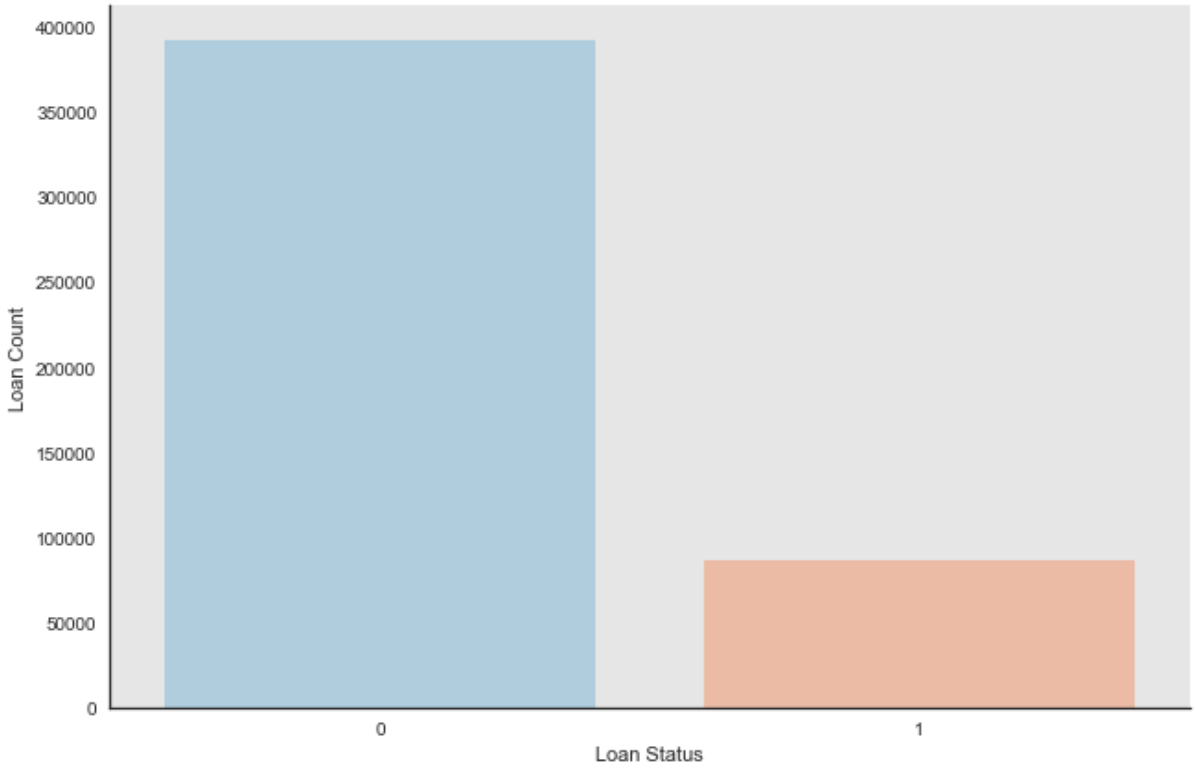
We will first review the distribution of the field and then assess if we need to handle it in any mannner. We shall be converting it to a binary feature with possible values of 0 to denote that someone **Paid_in_Full** and 1 to denote **Default**.




Out[144]:

| | Proportion | Description |
|---|---|---|
| **Fully Paid** | 65.729895 | Loan has been fully paid off. |
| **Current** | 18.530167 | Loan is up to date on current payments. |
| **Charged Off** | 14.555890 | Loan has been charged-off. |
| **Late (31-120 days)** | 0.727100 | Loan hasn't been paid in 31 to 120 days |
| **In Grace Period** | 0.298774 | The loan is in the grace period of 15 days pas... |
| **Late (16-30 days)** | 0.155162 | Loan hasn't been paid in 16 to 30 days |
| **Default** | 0.003013 | Loan is defaulted on and no payment has been m... |

Upon a closer look it appears that it consist of values which are other than *Fully Paid* and *Charged Off (Default)*. For our purposes, we need to be sure of an account's terminal status, hence it is advisable for us to only consider those loans which have either fully paid or have charged-off. I have written a small function in the *utilities.py* script which will filter the dataframe and convert the data type for the column.




```
007xx      4
008xx      1
010xx    792
011xx    220
012xx    179
Name: zip_code, dtype: int64
```

```
007      4
008      1
010    792
011    220
012    179
Name: zip_code, dtype: int64
```

```
 36 months    373076
 60 months    106585
Name: term, dtype: int64
```

```
36    373076
60    106585
Name: term, dtype: int64
```

To clean up the employment length, we will remove the text and convert <1 year to 0.

```
0             23026
1 year        30871
10+ years    152745
2 years       42855
3 years       38176
4 years       28954
5 years       31708
6 years       26324
7 years       25878
8 years       23020
9 years       18477
< 1 year      37627
Name: emp_length, dtype: int64
```

```
0      60653
1      30871
2      42855
3      38176
4      28954
5      31708
6      26324
7      25878
8      23020
9      18477
10    152745
Name: emp_length, dtype: int64
```

Next we will clean all the columns with date type of data by converting them to pandas datetime. This will make it easier for us to analyze the data by dates.

| | 597475 | 597476 |
|---|---|---|
| **issue_d** | Jan-2016 | Jan-2016 |
| **last_credit_pull_d** | Dec-2017 | Oct-2017 |
| **earliest_cr_line** | Sep-1999 | Jul-2002 |

| | 597475 | 597476 |
|---|---|---|
| **last_credit_pull_dt** | 2017-12-01 | 2017-10-01 |
| **issue_dt** | 2016-01-01 | 2016-01-01 |
| **earliest_cr_line_dt** | 1999-09-01 | 2002-07-01 |

```
2014    208119
2013    126872
2012     53287
2016     51599
2011     21721
2010     11535
2009      4716
2008      1562
2007       250
Name: issue_dt, dtype: int64
```

At the same time, we will drop any columns which are of no longer any use to us.

Let's review our data types after data cleaning to ensure that they are of appropriate data types.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 479661 entries, 0 to 597476
Data columns (total 20 columns):
funded_amnt                479661 non-null int64
loan_amnt                  479661 non-null int64
term                       479661 non-null int64
installment                479661 non-null float64
total_pymnt                479661 non-null float64
loan_status                479661 non-null int64
grade                      479661 non-null object
purpose                    479661 non-null object
annual_inc                 479661 non-null float64
home_ownership             479661 non-null object
emp_length                 479661 non-null int64
addr_state                 479661 non-null object
int_rate                   479661 non-null float64
emp_length_mi_flag         479661 non-null int64
emp_title_mi_flag          479661 non-null int64
verification_status_mod    479661 non-null object
emp_title_mod              479661 non-null object
last_credit_pull_dt        479661 non-null datetime64[ns]
issue_dt                   479661 non-null datetime64[ns]
earliest_cr_line_dt        479661 non-null datetime64[ns]
dtypes: datetime64[ns](3), float64(4), int64(7), object(6)
memory usage: 76.8+ MB
None
```

Lastly, let's review any null values which may exist in our dataset.

```
funded_amnt                0
loan_amnt                  0
term                       0
installment                0
total_pymnt                0
loan_status                0
grade                      0
purpose                    0
annual_inc                 0
home_ownership             0
emp_length                 0
addr_state                 0
int_rate                   0
emp_length_mi_flag         0
emp_title_mi_flag          0
verification_status_mod    0
emp_title_mod              0
last_credit_pull_dt        0
issue_dt                   0
earliest_cr_line_dt        0
dtype: int64
```

## Sanity Checks

After cleaning our data, it's best to conduct sanity checks to ensure that the distribution of data makes sense at high level. As we are not aware of the Lending Club's business model, it's difficult to catch data nuances, but a quick review can always help can any issues that may be quite obvious.

```
************************************************
Summary of Training Data
************************************************

Number of Columns with atleast one null value:
0

Number of Rows with atleast one column as null:
0

Shape of Original Data:  (479661, 20)
Shape of De-Dup Data:  (479661, 20)

Number of numerical columns with more than 10% nulls:
False    11
Name: count, dtype: int64

*********************************
      loan_status
            count      mean       std  min  max
grade
A         82614.0  0.058465  0.234621  0.0  1.0
B        145782.0  0.119233  0.324063  0.0  1.0
C        126479.0  0.201765  0.401319  0.0  1.0
D         73965.0  0.271034  0.444496  0.0  1.0
E         34539.0  0.355714  0.478736  0.0  1.0
F         13097.0  0.411468  0.492119  0.0  1.0
G          3185.0  0.474097  0.499407  0.0  1.0

*********************************
                  loan_status
                        count      mean       std  min  max
purpose
car                    5680.0  0.124824  0.330548  0.0  1.0
credit_card          106634.0  0.153244  0.360224  0.0  1.0
debt_consolidation   280676.0  0.192190  0.394022  0.0  1.0
educational             325.0  0.172308  0.378230  0.0  1.0
home_improvement      27805.0  0.160403  0.366986  0.0  1.0
house                  2415.0  0.197930  0.398521  0.0  1.0
major_purchase        10401.0  0.152197  0.359229  0.0  1.0
medical                4961.0  0.196130  0.397108  0.0  1.0
moving                 3208.0  0.215711  0.411379  0.0  1.0
other                 25080.0  0.196252  0.397169  0.0  1.0
renewable_energy        375.0  0.205333  0.404485  0.0  1.0
small_business         7083.0  0.275589  0.446842  0.0  1.0
vacation               2749.0  0.181884  0.385819  0.0  1.0
wedding                2269.0  0.122080  0.327451  0.0  1.0
```

```
**********
              loan_status
                   count      mean       std  min  max
home_ownership
ANY                  1.0  0.000000       NaN  0.0  0.0
MORTGAGE        239541.0  0.161872  0.368334  0.0  1.0
NONE               45.0  0.155556  0.366529  0.0  1.0
OTHER             144.0  0.187500  0.391675  0.0  1.0
OWN             44726.0  0.187810  0.390565  0.0  1.0
RENT           195204.0  0.203654  0.402715  0.0  1.0

**********
                        loan_status
                             count      mean       std  min  max
verification_status_mod
Not_Verified              159260.0  0.136148  0.342947  0.0  1.0
Source_Verified           152768.0  0.199335  0.399502  0.0  1.0
Verified                  167633.0  0.207763  0.405708  0.0  1.0

**********
                loan_status
                     count      mean       std  min  max
emp_title_mod
Driver                1566.0  0.271392  0.444820  0.0  1.0
Manager               4865.0  0.213361  0.409722  0.0  1.0
Missing              29260.0  0.230519  0.421173  0.0  1.0
Other               426097.0  0.176866  0.381556  0.0  1.0
Owner                 1836.0  0.238017  0.425986  0.0  1.0
Project_Manager       1764.0  0.153628  0.360694  0.0  1.0
RN                    2309.0  0.180165  0.384408  0.0  1.0
Registered_Nurse      2433.0  0.182080  0.385990  0.0  1.0
Sales                 1857.0  0.211093  0.408194  0.0  1.0
Supervisor            2115.0  0.223641  0.416782  0.0  1.0
Teacher               5559.0  0.172873  0.378171  0.0  1.0
```

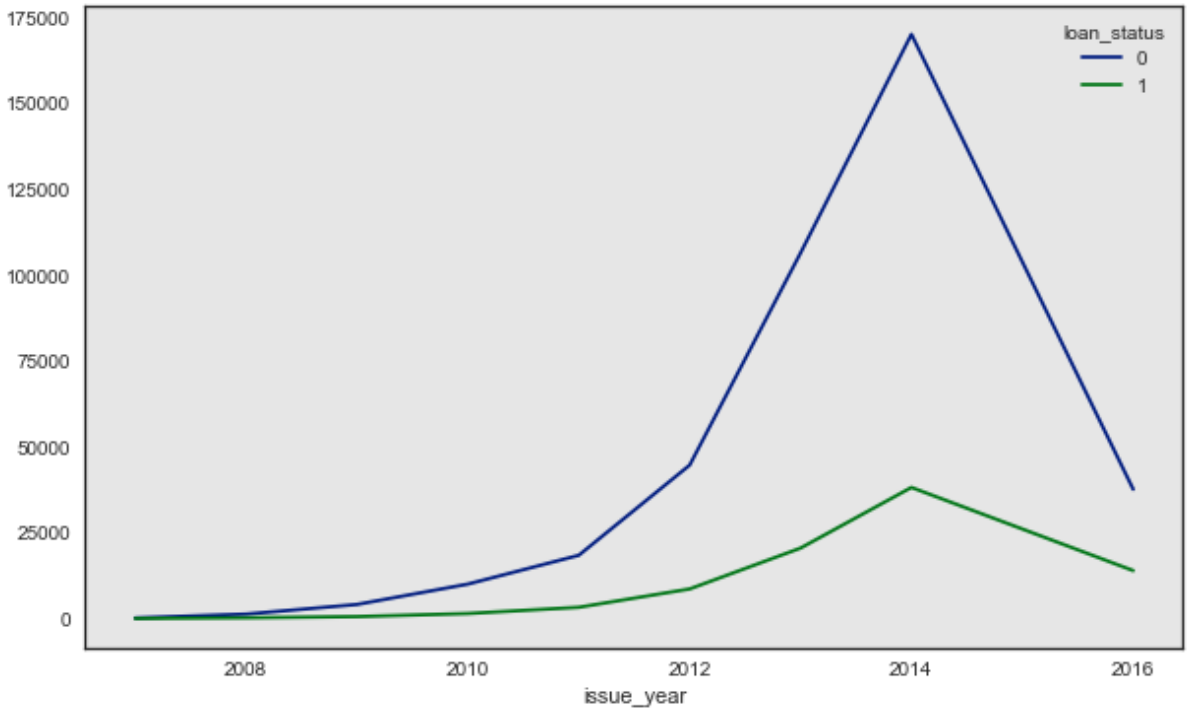
```
**********
           loan_status
                 count      mean       std  min  max
addr_state
AK              1312.0  0.170732  0.376418  0.0  1.0
AL              5954.0  0.218005  0.412925  0.0  1.0
AR              3569.0  0.209022  0.406667  0.0  1.0
AZ             11299.0  0.178865  0.383257  0.0  1.0
CA             74967.0  0.174583  0.379613  0.0  1.0
CO             10299.0  0.144577  0.351691  0.0  1.0
CT              7160.0  0.156983  0.363810  0.0  1.0
DC              1466.0  0.103001  0.304064  0.0  1.0
DE              1271.0  0.189614  0.392150  0.0  1.0
FL             33036.0  0.196967  0.397713  0.0  1.0
GA             15229.0  0.168363  0.374201  0.0  1.0
HI              2577.0  0.190920  0.393102  0.0  1.0
IA                 7.0  0.142857  0.377964  0.0  1.0
ID               140.0  0.242857  0.430349  0.0  1.0
IL             18657.0  0.164389  0.370638  0.0  1.0
IN              6642.0  0.203403  0.402560  0.0  1.0
KS              4185.0  0.146953  0.354102  0.0  1.0
KY              4467.0  0.191851  0.393801  0.0  1.0
LA              5643.0  0.199008  0.399289  0.0  1.0
MA             11337.0  0.172709  0.378012  0.0  1.0
MD             11047.0  0.190278  0.392538  0.0  1.0
ME               174.0  0.160920  0.368517  0.0  1.0
MI             11863.0  0.186884  0.389834  0.0  1.0
MN              8348.0  0.181720  0.385637  0.0  1.0
MO              7650.0  0.194379  0.395748  0.0  1.0
MS              1345.0  0.251301  0.433923  0.0  1.0
MT              1415.0  0.156890  0.363826  0.0  1.0
NC             13137.0  0.192586  0.394345  0.0  1.0
ND               136.0  0.250000  0.434613  0.0  1.0
NE               248.0  0.278226  0.449031  0.0  1.0
NH              2250.0  0.129333  0.335643  0.0  1.0
NJ             18116.0  0.190329  0.392571  0.0  1.0
NM              2698.0  0.203484  0.402664  0.0  1.0
NV              6965.0  0.209620  0.407066  0.0  1.0
NY             40871.0  0.195371  0.396491  0.0  1.0
OH             15416.0  0.190192  0.392465  0.0  1.0
OK              4250.0  0.211294  0.408275  0.0  1.0
OR              6202.0  0.148017  0.355145  0.0  1.0
PA             16393.0  0.191057  0.393146  0.0  1.0
RI              2051.0  0.172599  0.377992  0.0  1.0
SC              5655.0  0.155968  0.362857  0.0  1.0
SD              1026.0  0.188109  0.390990  0.0  1.0
TN              6179.0  0.214274  0.410351  0.0  1.0
TX             37828.0  0.170958  0.376477  0.0  1.0
UT              3647.0  0.168906  0.374720  0.0  1.0
VA             14334.0  0.186898  0.389843  0.0  1.0
VT               907.0  0.148842  0.356129  0.0  1.0
WA             10814.0  0.155354  0.362259  0.0  1.0
WI              6062.0  0.165292  0.371475  0.0  1.0
WV              2286.0  0.150919  0.358048  0.0  1.0
WY              1131.0  0.157383  0.364322  0.0  1.0
```

## Exploratory Visualization

In this section, you will need to provide some form of visualization that summarizes or extracts a relevant characteristic or feature about the data. The visualization should adequately support the data being used. Discuss why this visualization was chosen and how it is relevant. Questions to ask yourself when writing this section:

- *Have you visualized a relevant characteristic or feature about the dataset or input data?*
- *Is the visualization thoroughly analyzed and discussed?*
- *If a plot is provided, are the axes, title, and datum clearly defined?*

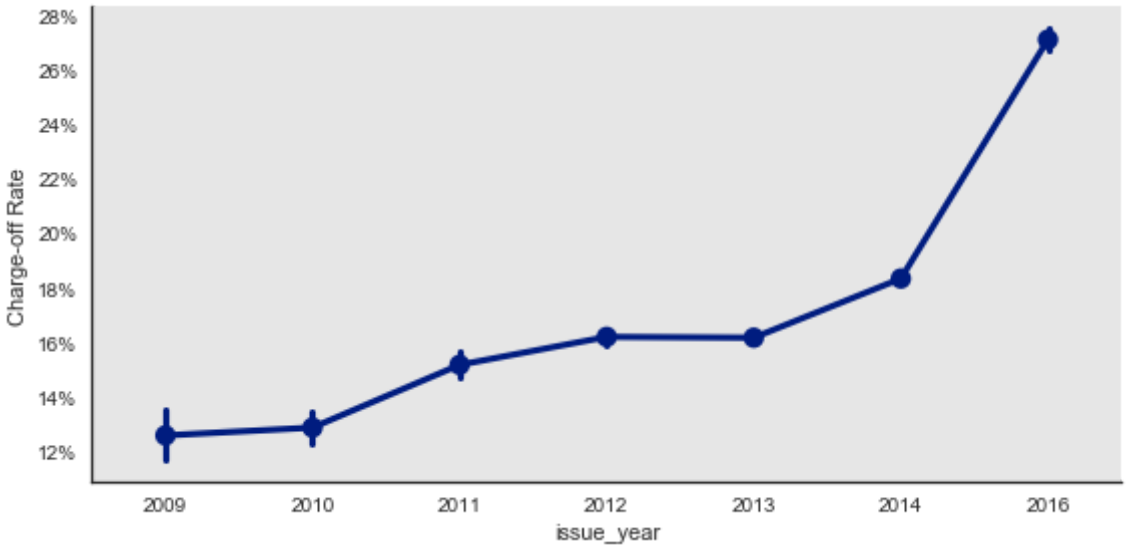
```
Out[157]: <matplotlib.axes._subplots.AxesSubplot at 0x1237fb2d0>
```




**Charge-off Rate by Year**:

The above chart helps us under the distribution of good vs bad loans but just to get a cleared idea of the trend, we should take a close look at the rate of bad loans over the years. It is important to note that we are only looking at one quarter of originations for 2016, so may not need to read too much into the trend as it doesn't include the rest of the year.

```
Out[158]: <seaborn.axisgrid.FacetGrid at 0x136b79c10>
```

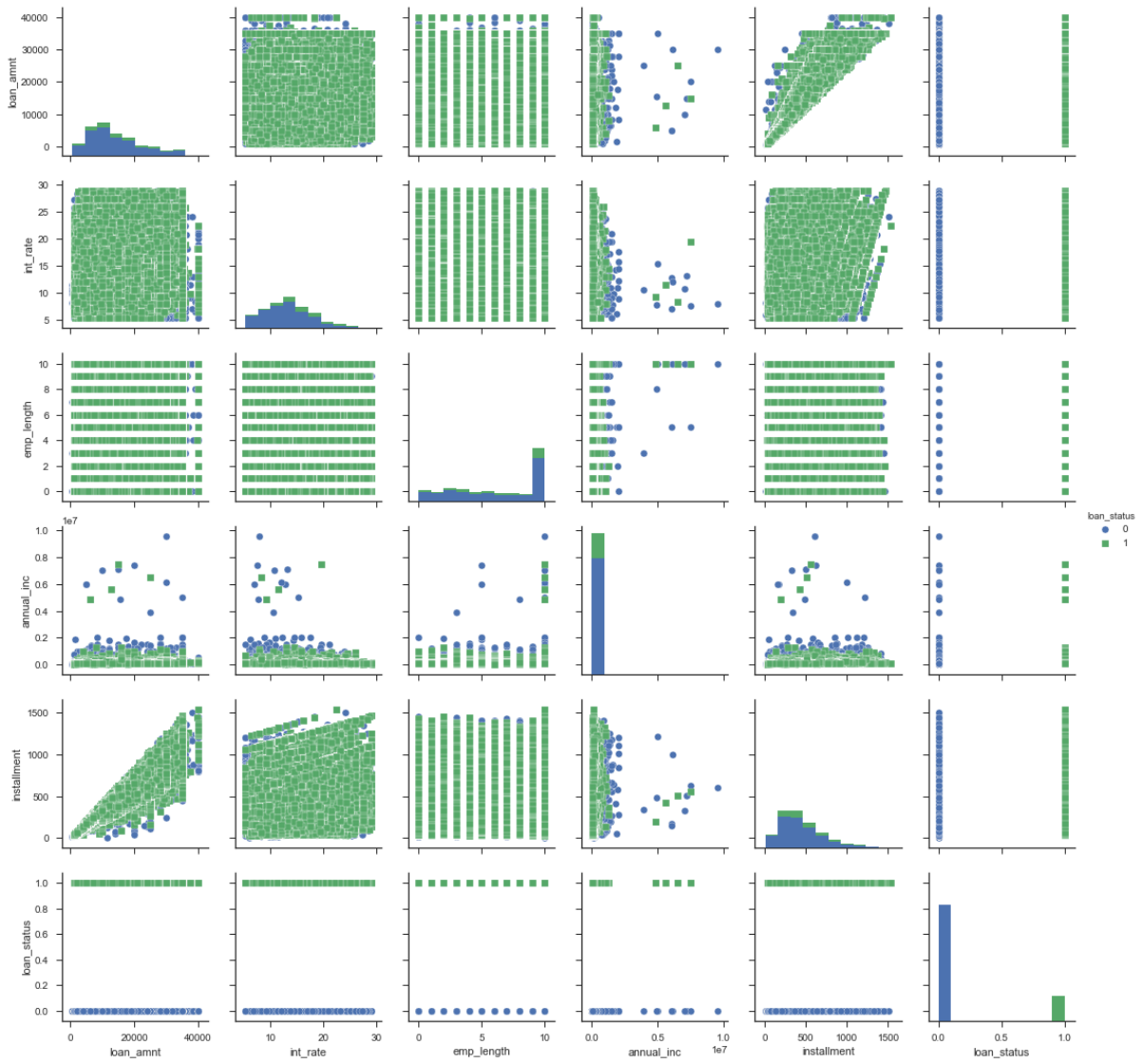



Reviewing the above chart shows a clear trend of rise in borrowers who are failing to repay their loan fully, making a need of the type of model we are proposing even more critical

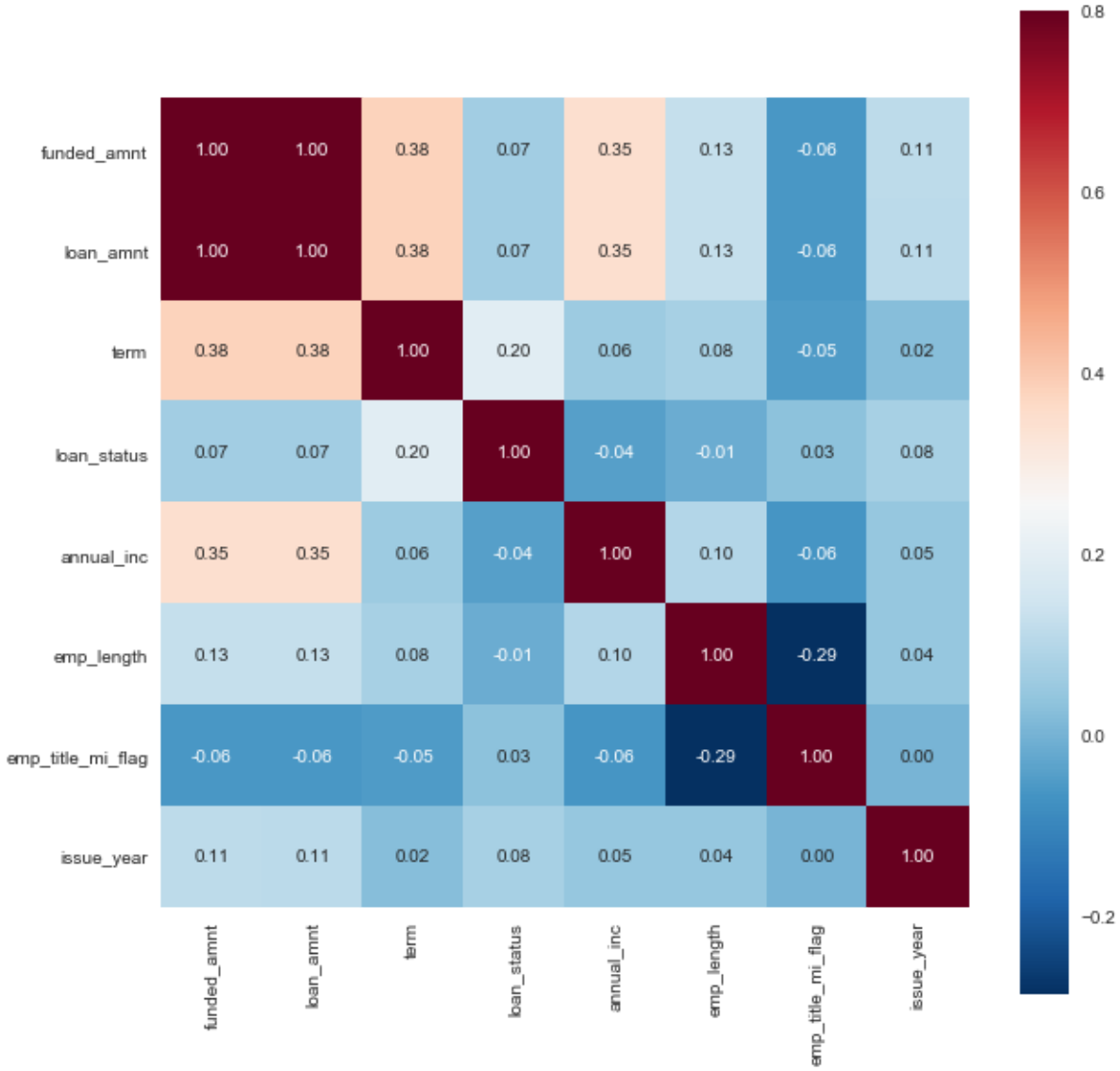
**Numerical Variables**:

In the prior set of sanity checks, we took a look at the distribution by categorical variables. Here, we will start by reviewing the numerical variables. Specifically, we will review the correlations between numerical variables. The primary motivation to check correlations is to assess if we have features which move together (either in same or opposite direction). If such variables, exist, we need to be careful incorporating them in our analysis as they could lead to Multicollinearity giving rise to instability in the coefficients(parameter estimates) obtained from fitting a linear model.

```
Out[45]: <seaborn.axisgrid.PairGrid at 0x121ff20d0>
```



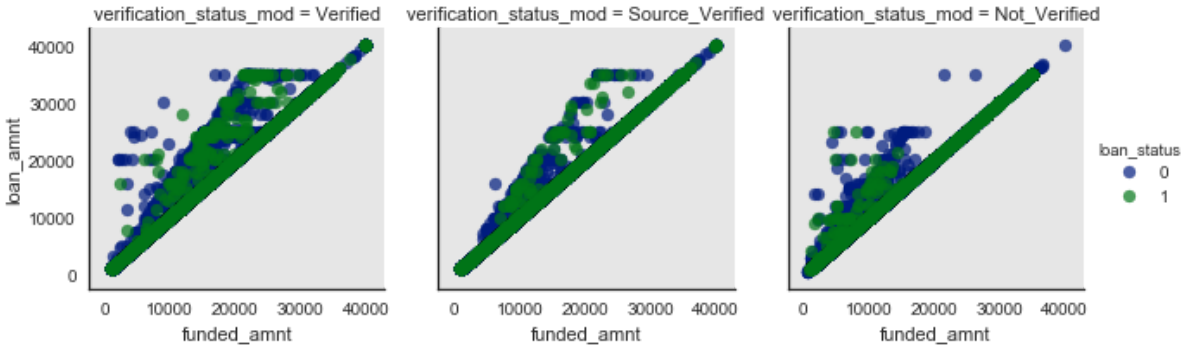
```
Correlation among variables:
```



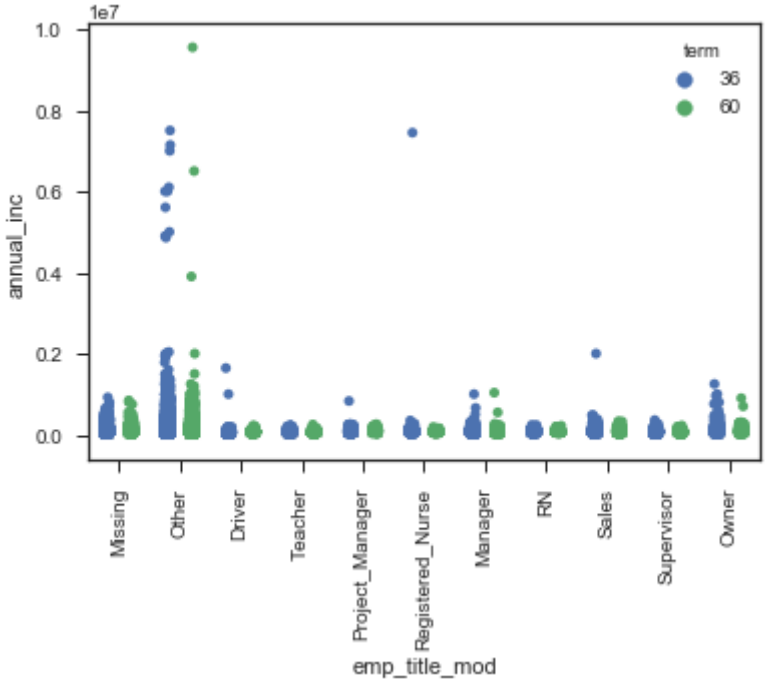

*Funded and Loan Amounts* seem to be perfectly correlated, which implies that we can keep either one in our model. I will keep loan_amnt in our future analysis and the ratio of funded to loan amount as there could be some loans which are not fully funded (1.00 could simply be a rounding term displayed on the above correlogram.

We will next check if there is a differnece in such correlation if we were to look at the correlation by **loan_status** and other variables. This will help us de-averge any relationship between our independent variables and dependent variables.

Taking a closer look by **verification_status**, there does seem to be some difference but it's broadly coming from difference in the distribution of loan amounts in those categories.



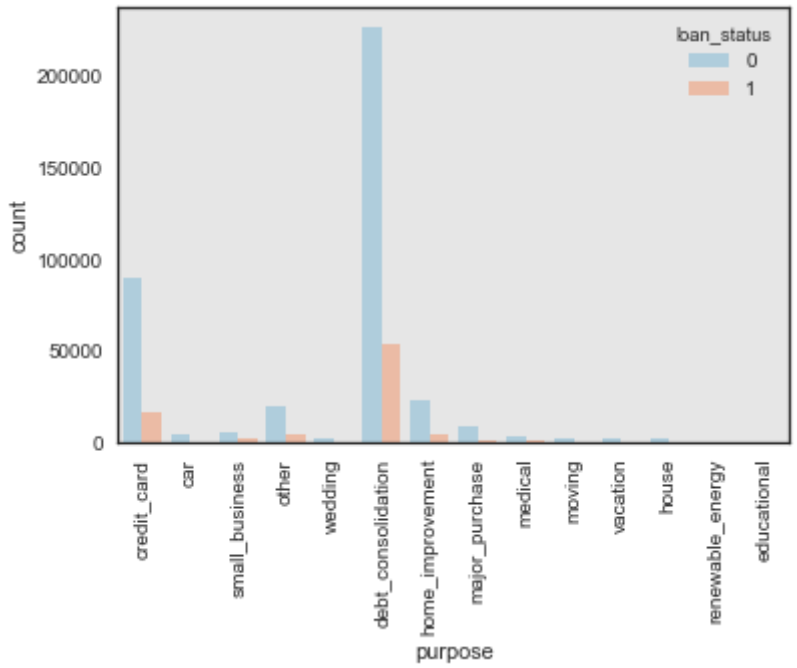

```
Out[64]: <matplotlib.axes._subplots.AxesSubplot at 0x12328e2d0>
```

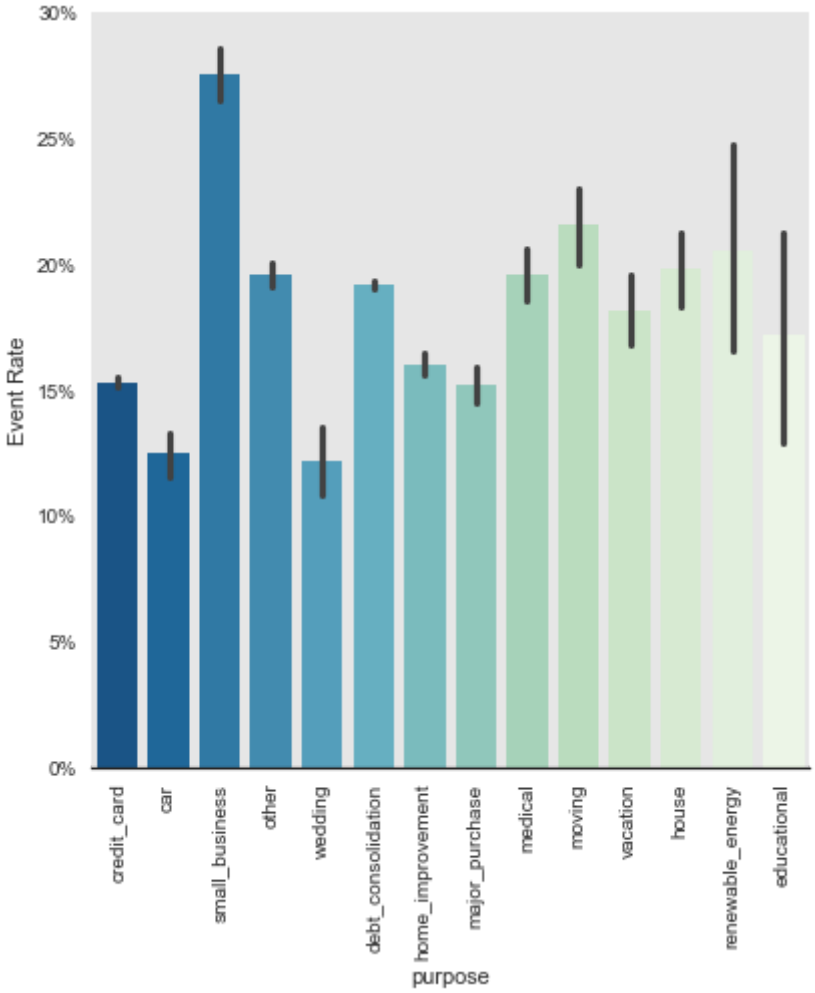



Next, we will take a closer look at the categorical features available to us and here we will review both the distribution by each of the features along with the event rate (charge-off rate in this case).

**Distribution of Loans by Purpose and Home ownership**:

```
Out[160]: debt_consolidation    280676
          credit_card           106634
          home_improvement       27805
          major_purchase         10401
          small_business          7083
          car                     5680
          medical                 4961
          moving                  3208
          vacation                2749
          house                   2415
          wedding                 2269
          renewable_energy         375
          educational              325
          Name: purpose, dtype: int64
```

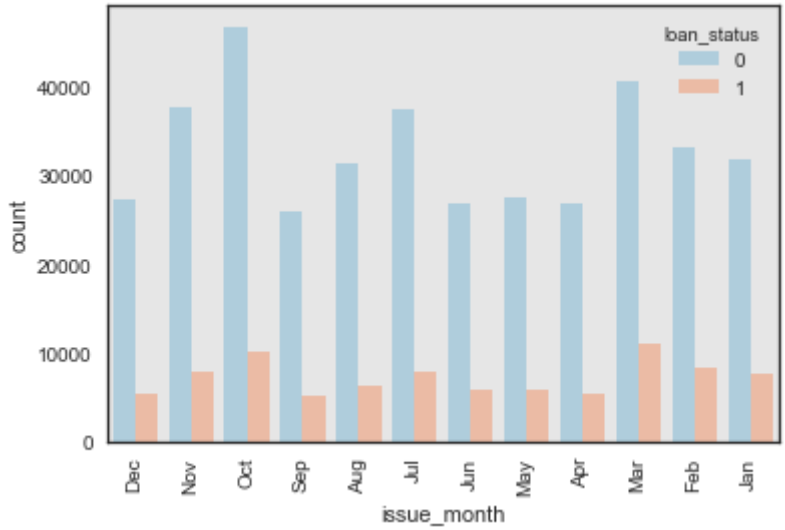


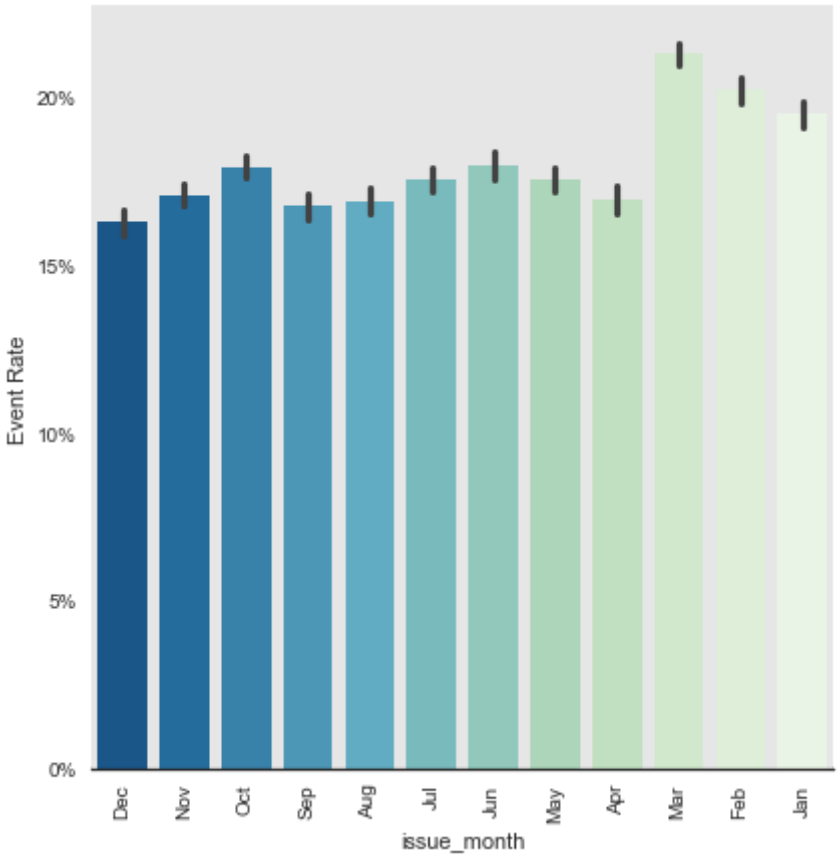




Looking at the chart above (bottom), one can easily observe that loans borrowed for the purpose _small_business_are way riskier than any other category. However, looking at the chart on the top, we can see it's only a small proportion of total accounts.

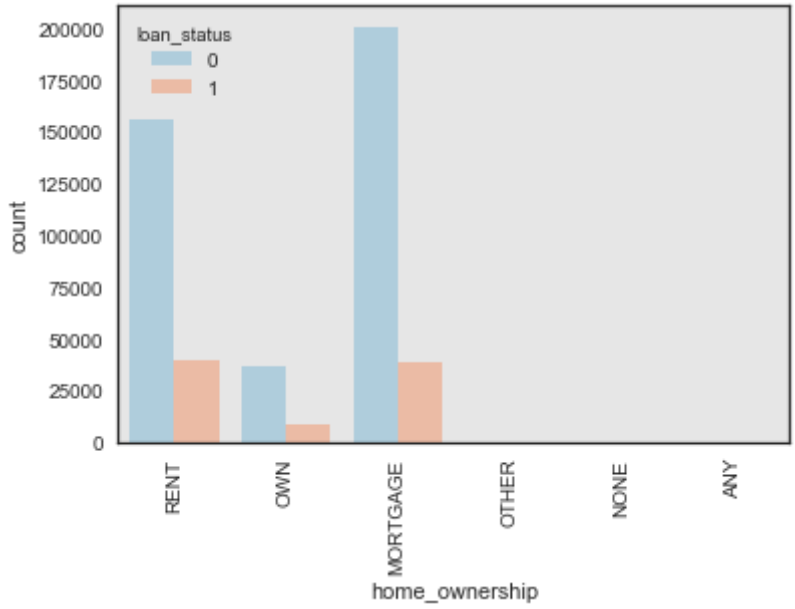
There could be factors like *seasonality* that can play a major role in financial data. For example, some borrowers could be credit hungry right before the holidays for shopping needs. Let's try to look at the data by **issue_month**.

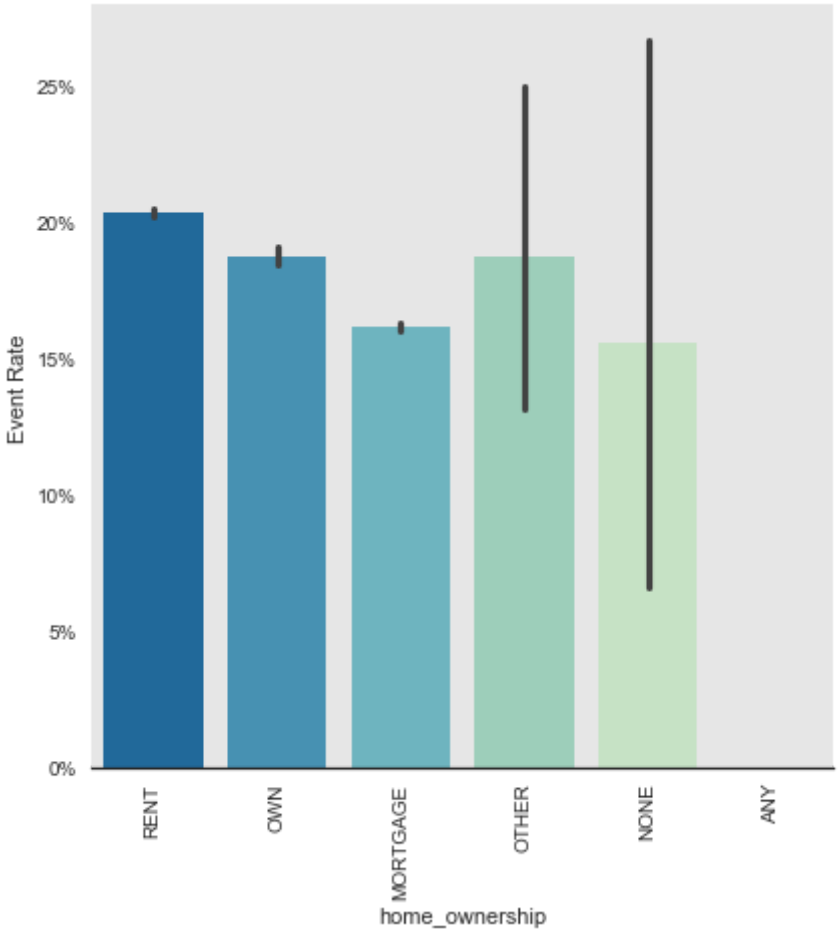
Looking at the below charts, it appears that we issue more loans during Oct, but it's actually loans issued in March, Feb and Jan which tend to be riskier.




Lastly, we will review the loans by **Home Ownership** information.



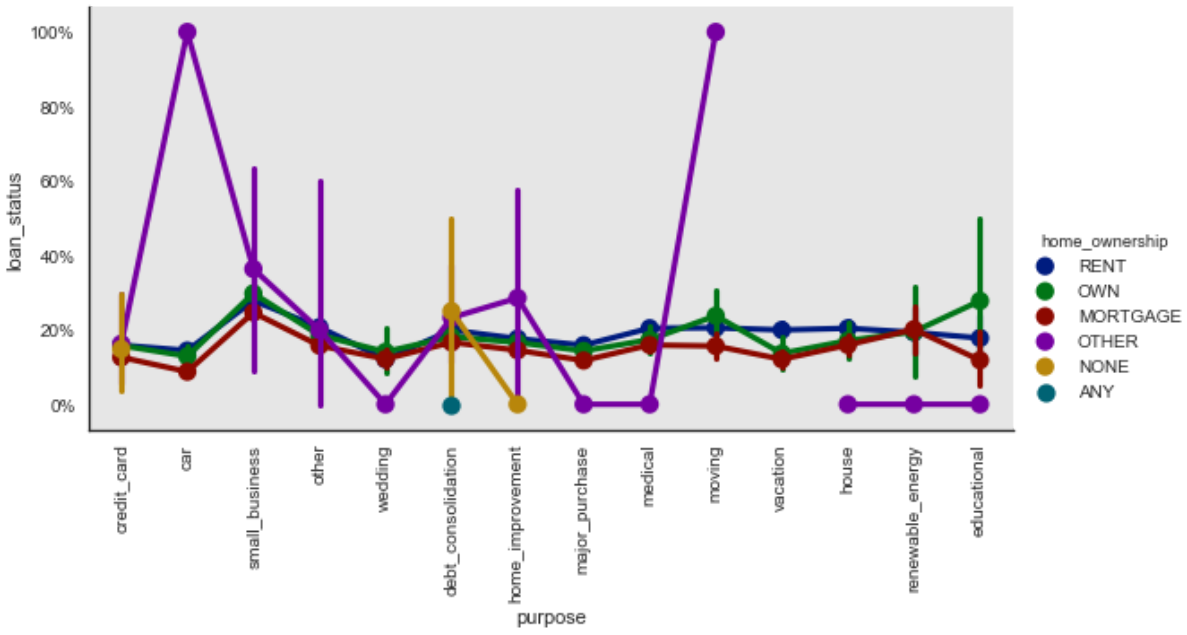




The above chart appear to convey that while most of our borrowers are home-owners, it is actually renters who tend to be riskier. All this information can be later verified when we develop our model.

**Interaction Effects**:

There could be effects which help us understand the relationship between multiple independent variables and our target. Such effect is called an *Interaction effect* and can yield interesting insights. To observe this, we will look at the risk rate by home ownership and purpose of the loan.

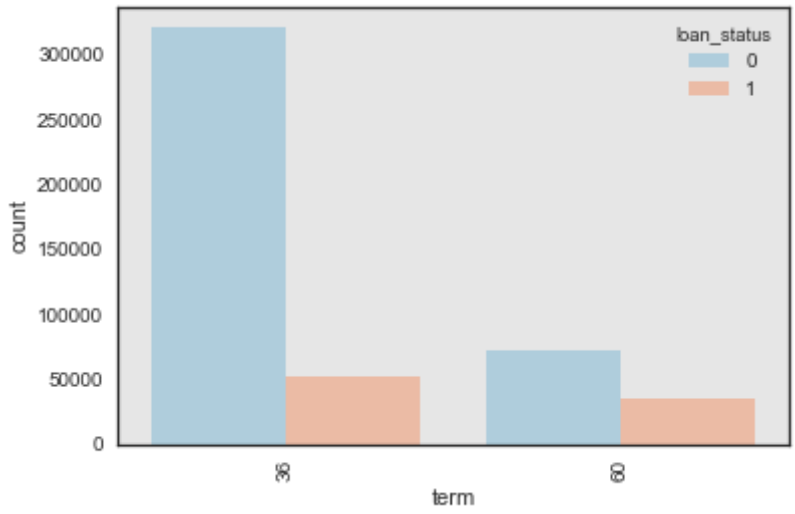
```
Out[164]: <seaborn.axisgrid.FacetGrid at 0x1200f13d0>
```

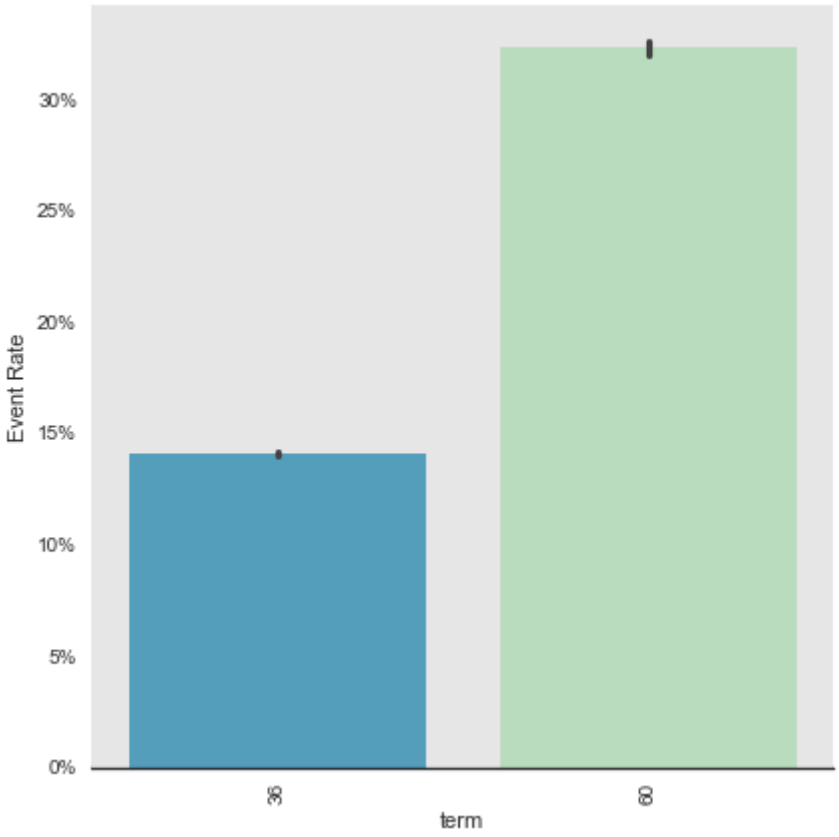



As the above lines cross each other at several points, it is leading me to think that there could be some two-way interactions that we may need to keep in mind when building a linear model which doesn't pick up such interactions like other Tree-based methods.

If the above ranking holds true, we should also observe separtion between distribution by interest rates for good and defaulted loans. We shall verify the same:

**Term**

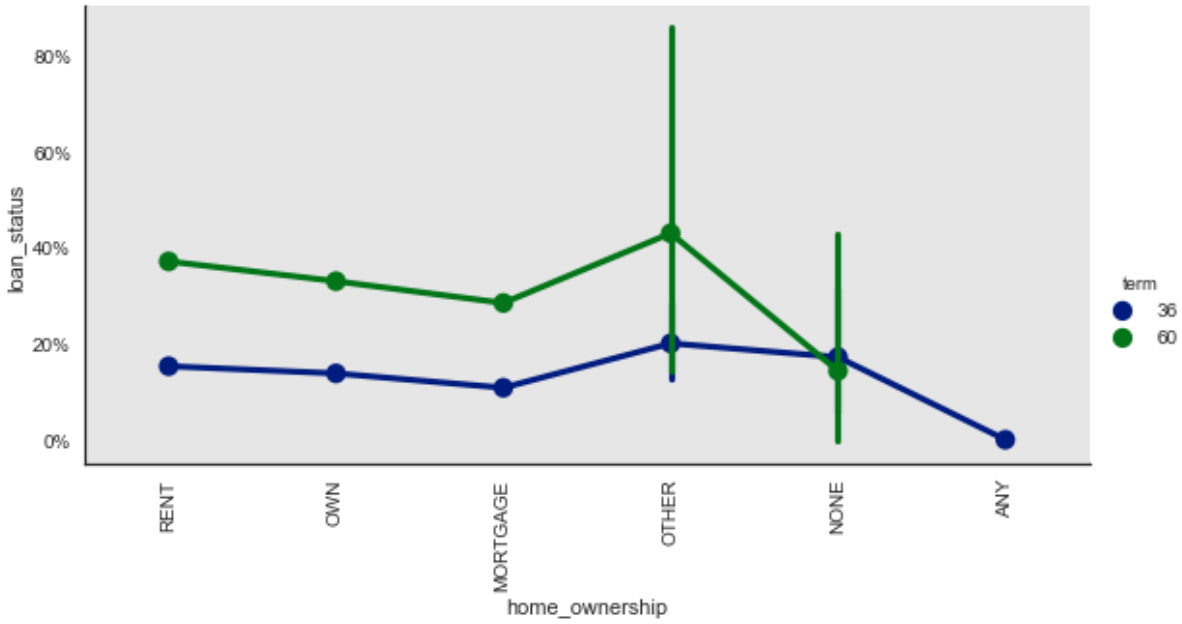


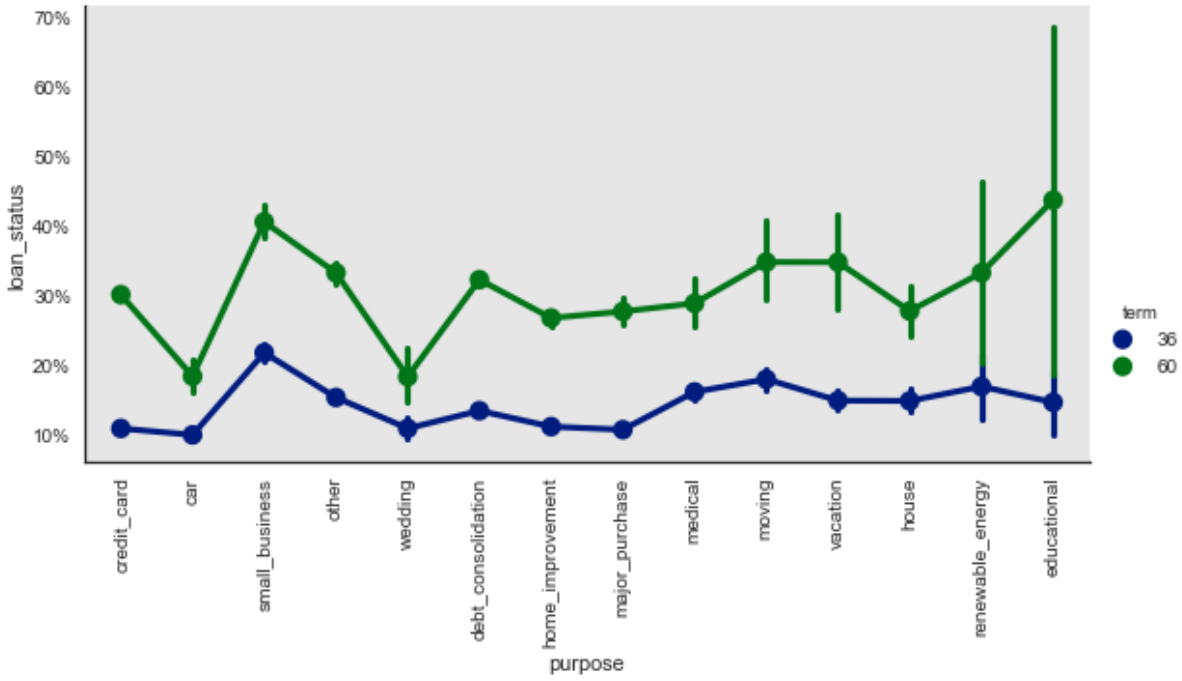

It's striking to observe how riskier longer term loans are , they carry more than twice the risk but we only book less than 1/6th of loans we book for 36 months.

Whenever I observe such diverging trends, I like to de-average them by various cateogries to see if this is coming from a particular section of loans or it's prevalent in all categories.

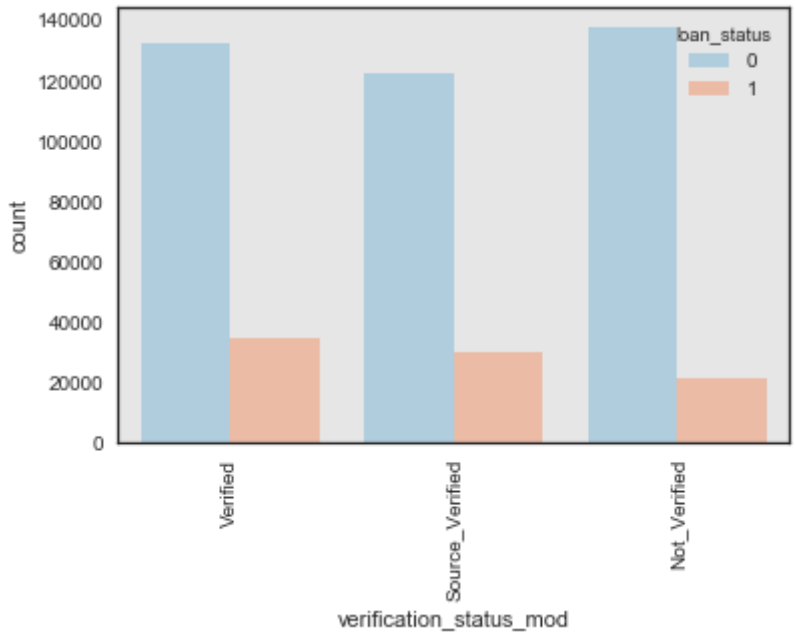
Out[166]: <seaborn.axisgrid.FacetGrid at 0x128175150>

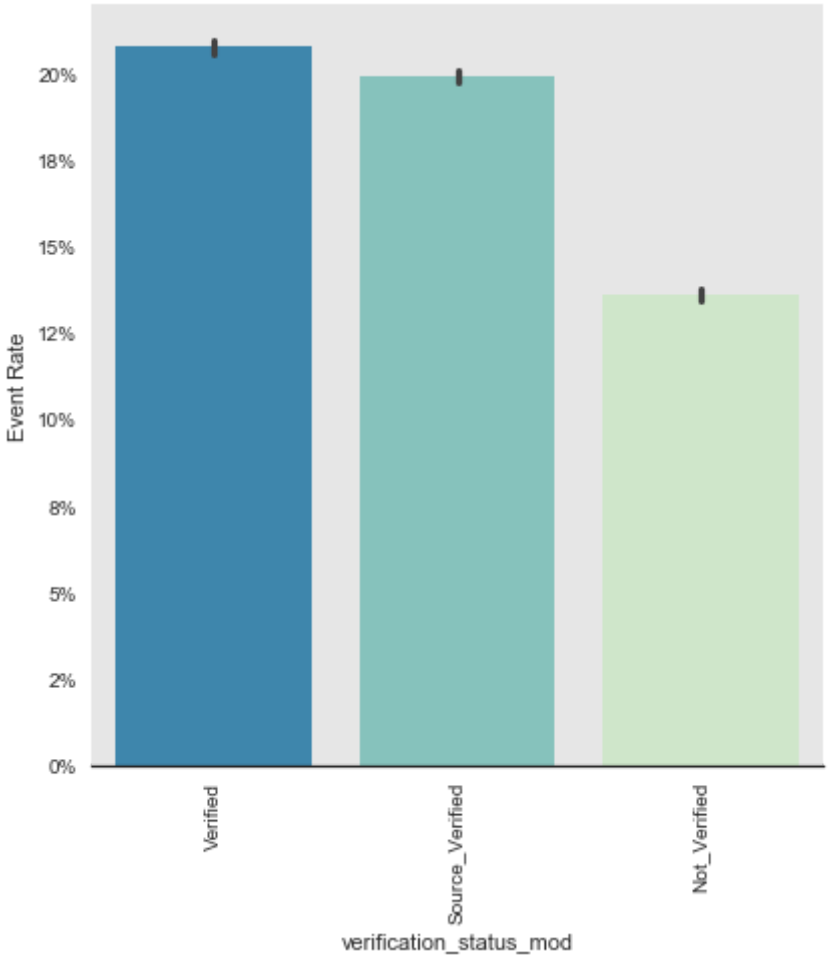


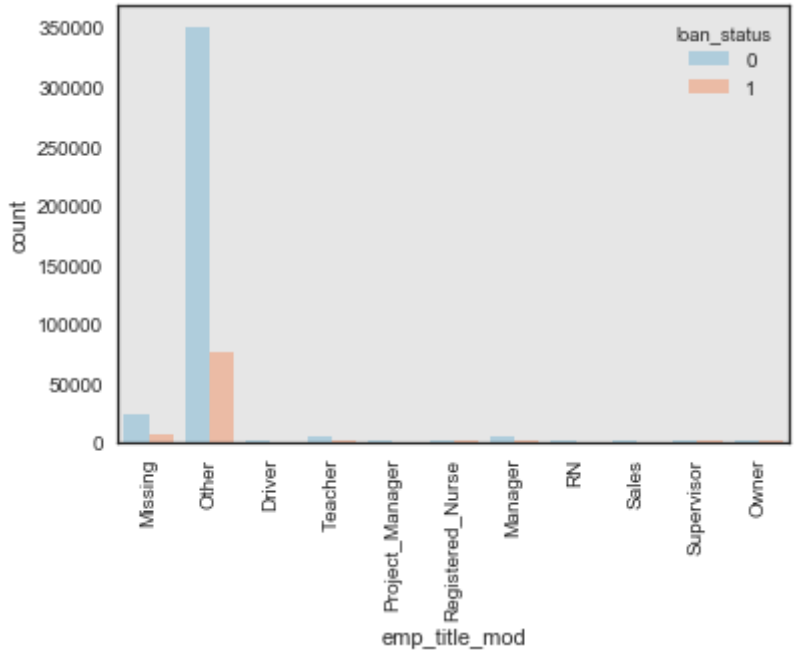

Out[167]: <seaborn.axisgrid.FacetGrid at 0x128175ad0>

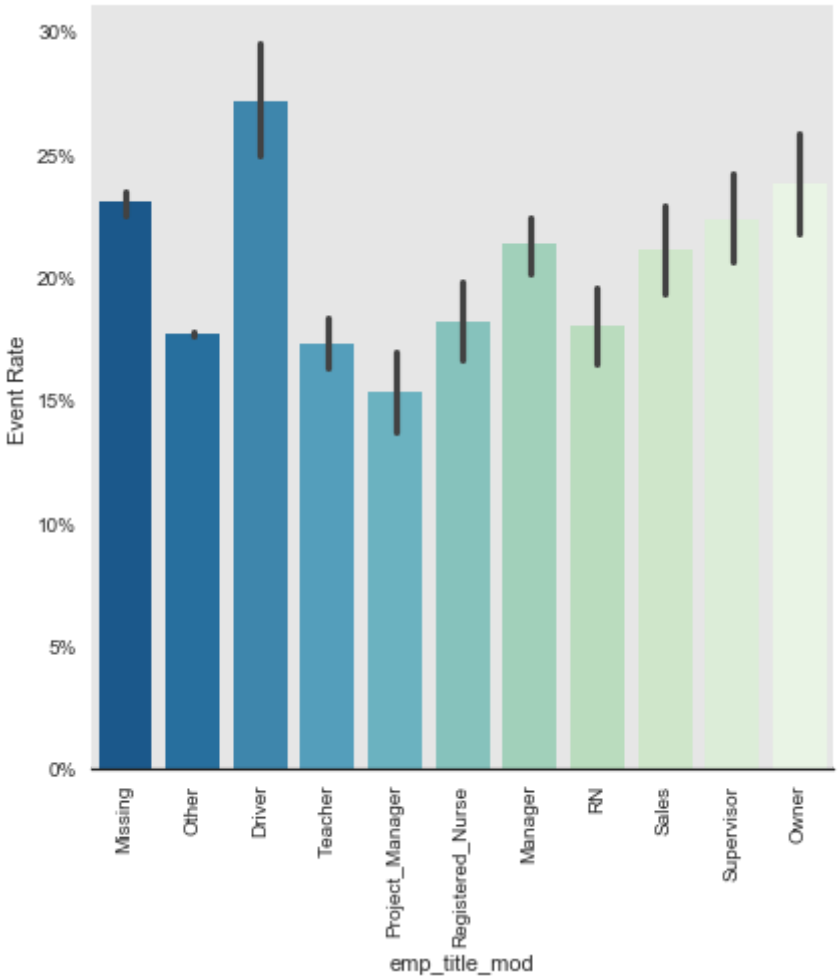


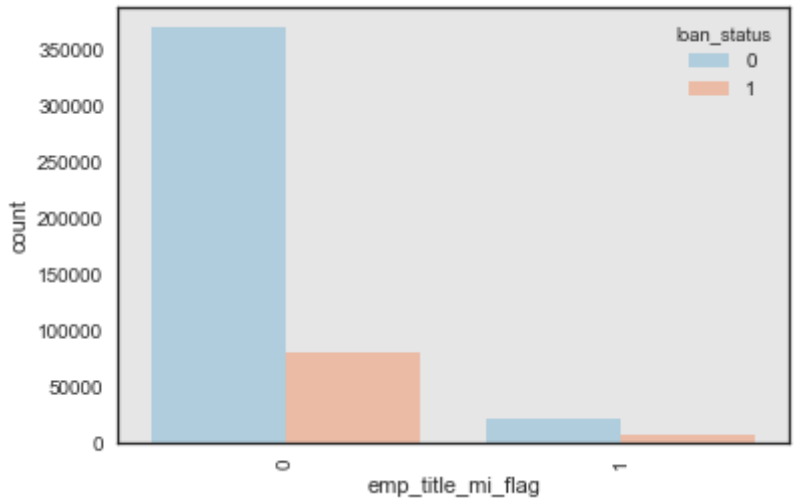

**Categorical Variables**:

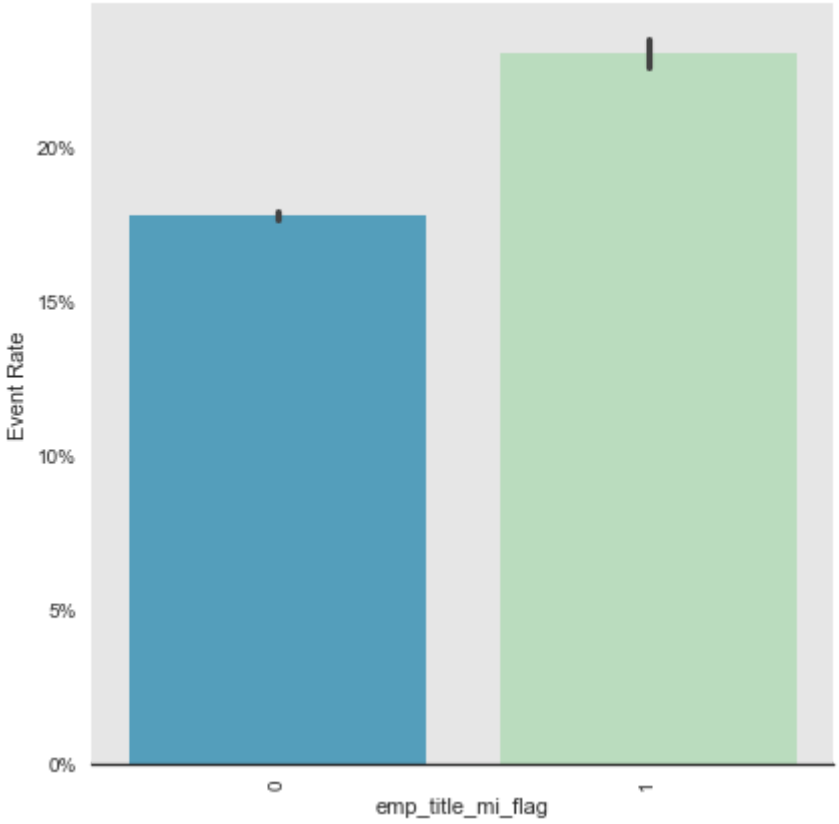

loan_status
0
1
count
140000
120000
100000
80000
60000
40000
20000
0
Verified
Source_Verified
Not_Verified
verification_status_mod


Event Rate
20%
18%
15%
12%
10%
8%
5%
2%
0%
Verified
Source_Verified
Not_Verified
verification_status_mod

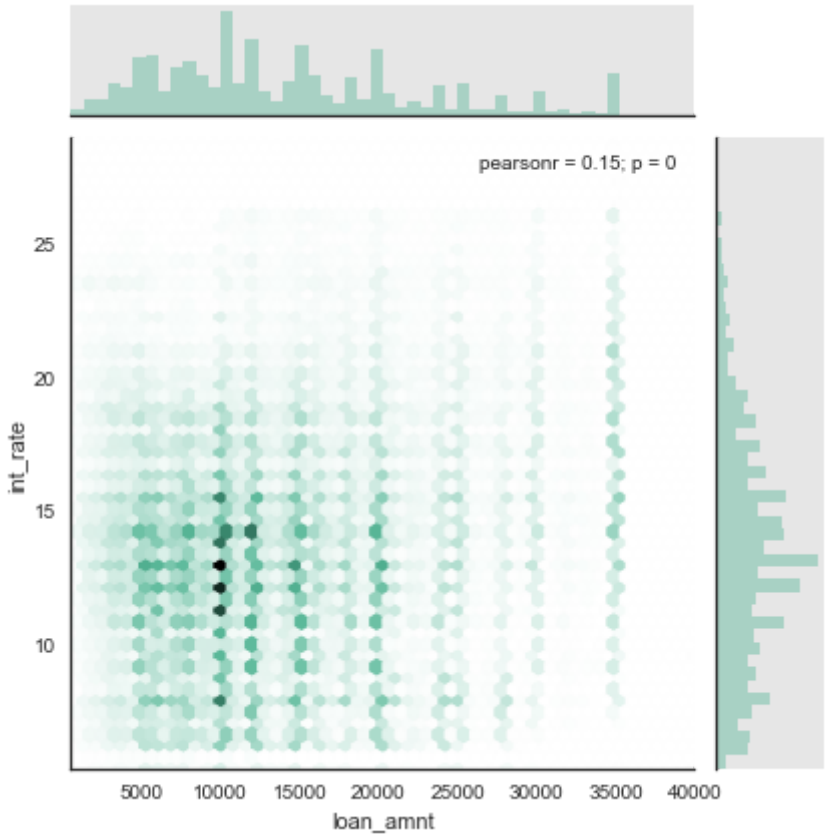

loan_status
0
1
count
350000
300000
250000
200000
150000
100000
50000
0
Missing
Other
Driver
Teacher
Project_Manager
Registered_Nurse
Manager
RN
Sales
Supervisor
Owner
emp_title_mod


Event Rate
30%
25%
20%
15%
10%
5%
0%
Missing
Other
Driver
Teacher
Project_Manager
Registered_Nurse
Manager
RN
Sales
Supervisor
Owner
emp_title_mod



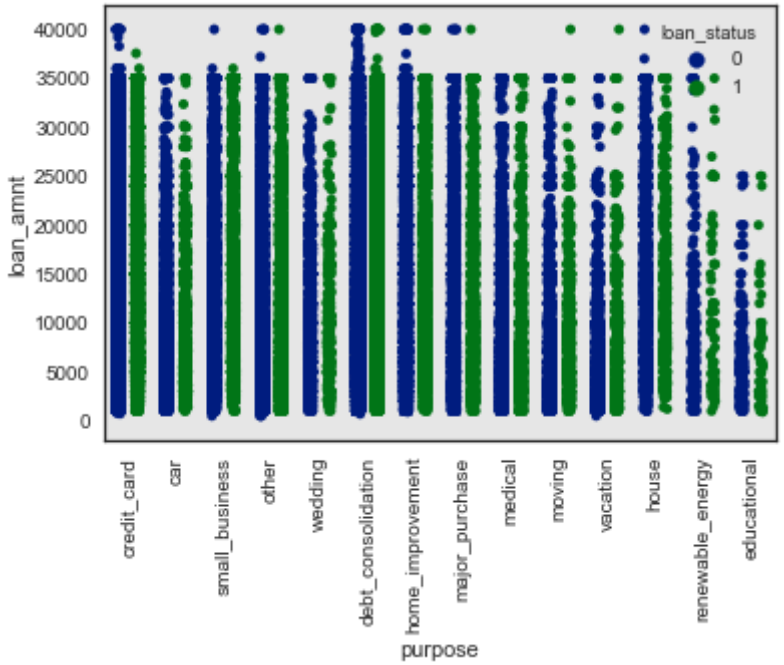




**Multi-variate Distributions to get a better sense of data**:

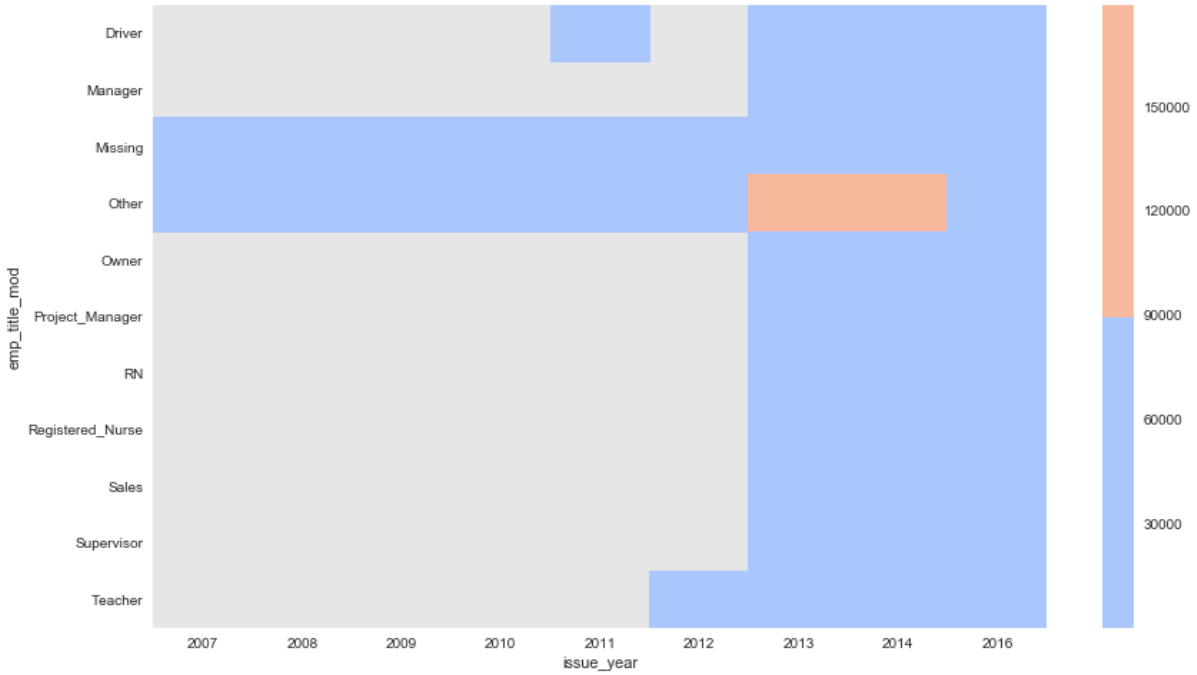


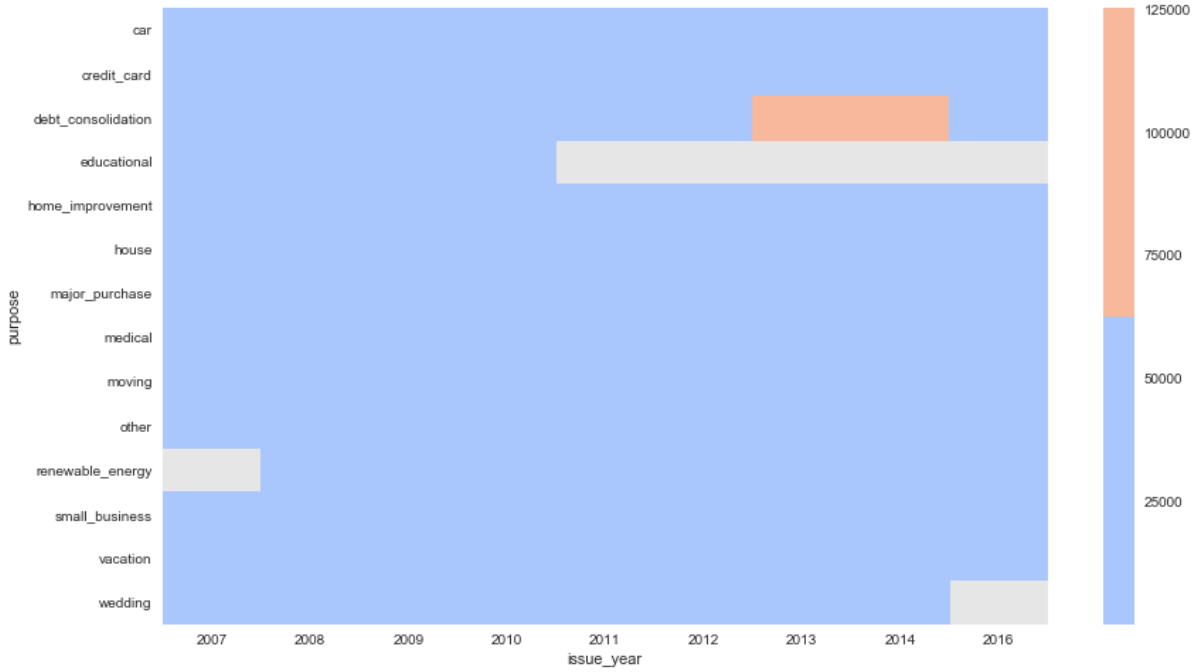

```
/Users/xtl476/anaconda/envs/capstone/lib/python2.7/site-packages/seabor
n/categorical.py:2586: UserWarning: The `split` parameter has been rena
med to `dodge`.
  warnings.warn(msg, UserWarning)
```

Out[172]: <matplotlib.axes._subplots.AxesSubplot at 0x1237ead10>



We can also review some of the variables by year to visualize if there were any significant strategy shifts in booking one kind of loans over the other over time. Below we can see that in the recent years, significant number of short-term loans have been booked compared to in the past



## Algorithms and Techniques

In this section, you will need to discuss the algorithms and techniques you intend to use for solving the problem. You should justify the use of each one based on the characteristics of the problem and the problem domain. Questions to ask yourself when writing this section:

- *Are the algorithms you will use, including any default variables/parameters in the project clearly defined?*
- *Are the techniques to be used thoroughly discussed and justified?*
- *Is it made clear how the input data or datasets will be handled by the algorithms and techniques chosen?*

## Algorithms

* __Extension of the solution that can improve the solution__:*

__Logistic Regression__:

This is a supervised technique which is able to classify our targe using a linear boundary. It is popular approach for classification problems as it can generate insights by not only showing features that affect the performance but also by easy to interpret impact of these features by providing coeffecients. It takes into consideration the observerd data and identifiess the relationships between input feature and log-odds of event we are interested in predicting. It provides weights to each input feature, combines them together (and adds a bias) and using a sigmoid transformation, returns a probability (log-odds) associated with the event of interest. This provides us a nice way to understand the impact of changing any input feature on the desired outcome.*

_Benefits_:

- Simplistic Approach
- Fast to Train
- Probabilistic approach
- Returns p-values provding statistical significance of features

_Cons_:

- Cannot identify non-linear boundaries
- Requires variable treatment like interaction variables

__Random Forest__

It is a popular ensemble technique which is built upon the idea of collective intelligence of many decision trees built of subset of data and features. It is a simple idea of taking many decision trees and having them vote almost like having a committee of experts vote by looking at different aspects of a problem at any given time. A single Decision Tree has low bias and high variance, i.e., it may be difficult to take an individual decision tree and generalize it to new datasets. We overcome the high variance and inability of Decision Trees to predict on newer data by building not one but many trees. However, we don't build many trees on the same data, rather, we take sample of rows and features in each step of tree building and this helps the algorithm to generalize. Only those features that matter will repeatedly show up as important splitting criteria on many different trees.*

_Benefits_:

- Identifies non-linear boundaries
- Faster to train when compared to other ensemble techniquest
- Can be parallelized

_Cons_:

- Black-Box, lacks interpretability
- Easy to overfit
- Requires careful hyperparameter tuning (number of trees, tree depth etc.)

__Gradient Boosting__

This is another ensemble technique which has shown promise in recent years. It is built upon the idea of taking a weak learner and then try to improve upon the predictions by assigning higher weight to instances which were previously incorrectly classified. Here, we are sequentially trying to improve the results and instead of voting, we weigh the weak learners to come up with the final prediction. This can be slow at times as the weak learners (often trees) are grown sequentially and can't be parallelized like Random Forest. The reason the approach is called Gradient Boosting is because the algorithm uses Gradient Descent to optimize a given loss function when it sequentially builds trees.*

__Benefits__:

- Identifies non-linear boundaries
- Highly accurate and can have best performance among ensemble techniques

__Cons__:

- Slow to train
- Sensitive to outliers
- Black-Box, lacks interpretability
- As trees are built sequentially on residuals, it is not possible to parallelize (althought xgboost has recently tried to parallelize this)
- Requires careful hyperparameter tuning, else can lead to overfitting (learning rate, number of estimators etc.)

## Benchmark

In this section, you will need to provide a clearly defined benchmark result or threshold for comparing across performances obtained by your solution. The reasoning behind the benchmark (in the case where it is not an established result) should be discussed. Questions to ask yourself when writing this section:

- *Has some result or value been provided that acts as a benchmark for measuring performance?*
- *Is it clear how this result or value was obtained (whether by data or by hypothesis)?*

Our benchmark is not only going to be a model output but also a dollar threshold. As we plan to leverage 2016 (Q1) loan originations as our validation datasets, we will check how much money Lending Club made/lost on those bookings.

The reason this serves as a good benchmark is because it already takes into conisderataion the credit policy and the funding criteria applied by Lending Club (i.e. individual investors).

Our benchmark model will be a naive classifier which uses the most predictive information available to us, i.e. Loan Grades and interest rates. In the scenario where we don't really know what Lending Club's model looks like, I thought it may be a good idea to use an output of their complex model to assess if it helps classify risk.

At the same time, we have the privilege of comparing it to the actual outcome of someone either paying the loan back or not.

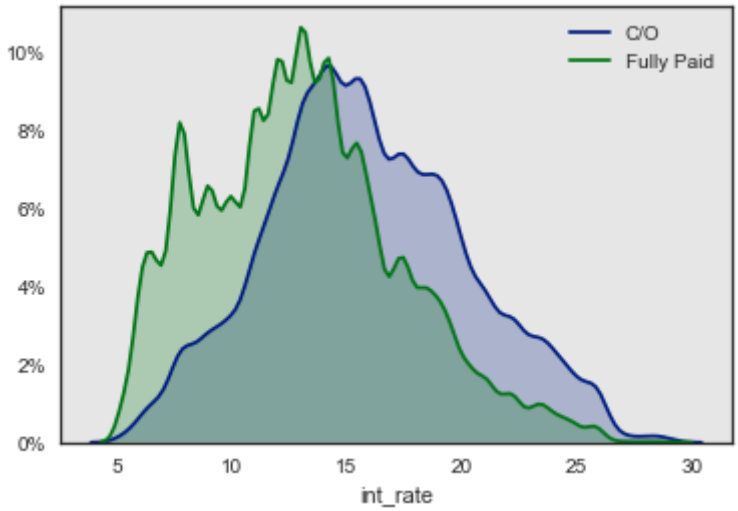
**Loan Grades**: As per Lending Club (https://www.lendingclub.com/public/rates-and-fees.action): "Based on each loan application and credit report, every loan is assigned a grade ranging from A1 to E5 with a corresponding interest rate. Each loan grade and its corresponding interest rate is displayed below."

Let's verify this holds true.

**Interest Rate**:

Another important variable that can help assess whether an account is risky is interest rate. Often the interest rates are closely associated with the risk profile of a borrower, i.e. higher risk customers tend to get loans approved at higher interest rates due to perceived notion of credit hungriness.

The below chart helps confirm that theory as most of the defaulted loans had higher interest rates than those who paid the loans fully.

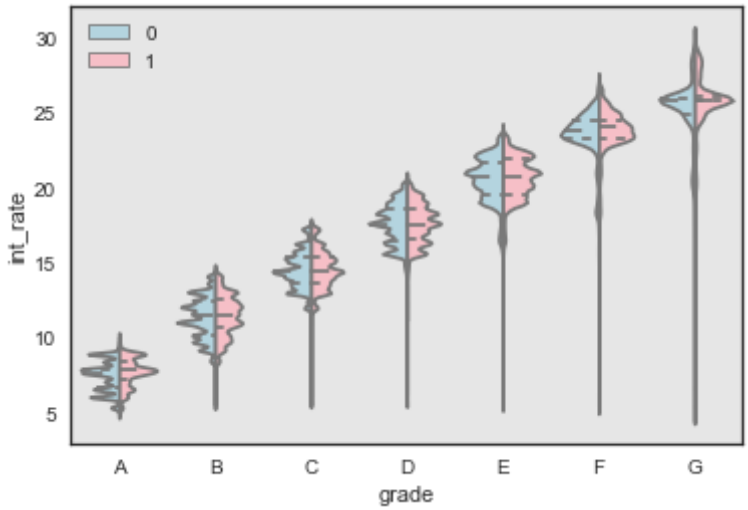



At the same time, we should also confirm the risk sloping by interst rates. The below chart confirms our assumption thatt the riskier grades had higher interest rates.

***Distribution by Loan Grade and Interest Rates***

If the ranking holds true, we should also observe separtion between distribution by interest rates for good and defaulted loans. Here we have verified the same.

```
Out[178]: (array([0, 1, 2, 3, 4, 5, 6]), <a list of 7 Text xticklabel objects>)
```

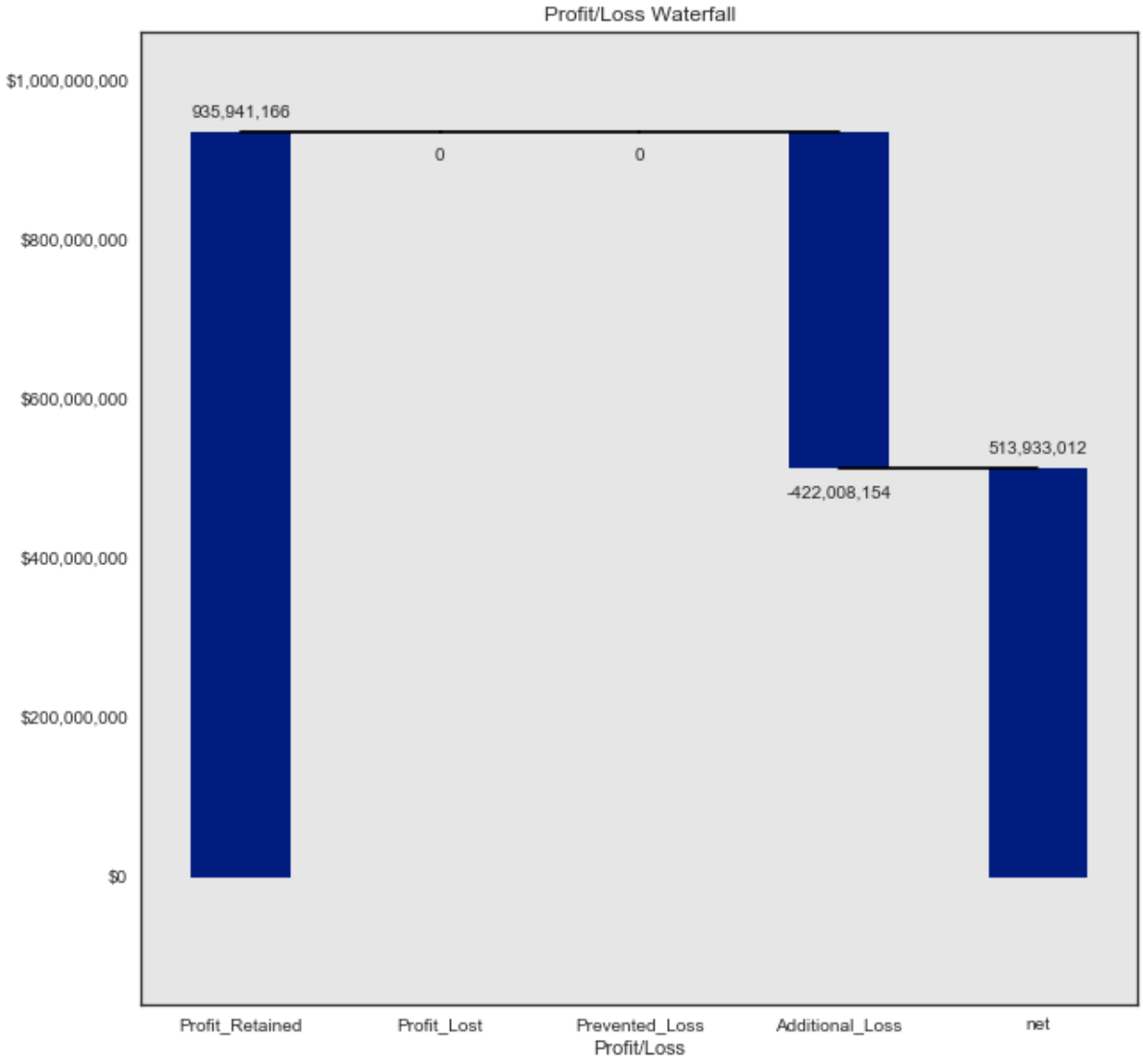



To obtain a rough sense of data (and to ensure that we are making money), let's look at the profit in dollars. Below, we will see how much money Lending Club made from loan originations between 2007 and 2014.

**Loan Originations from 2007-2014**

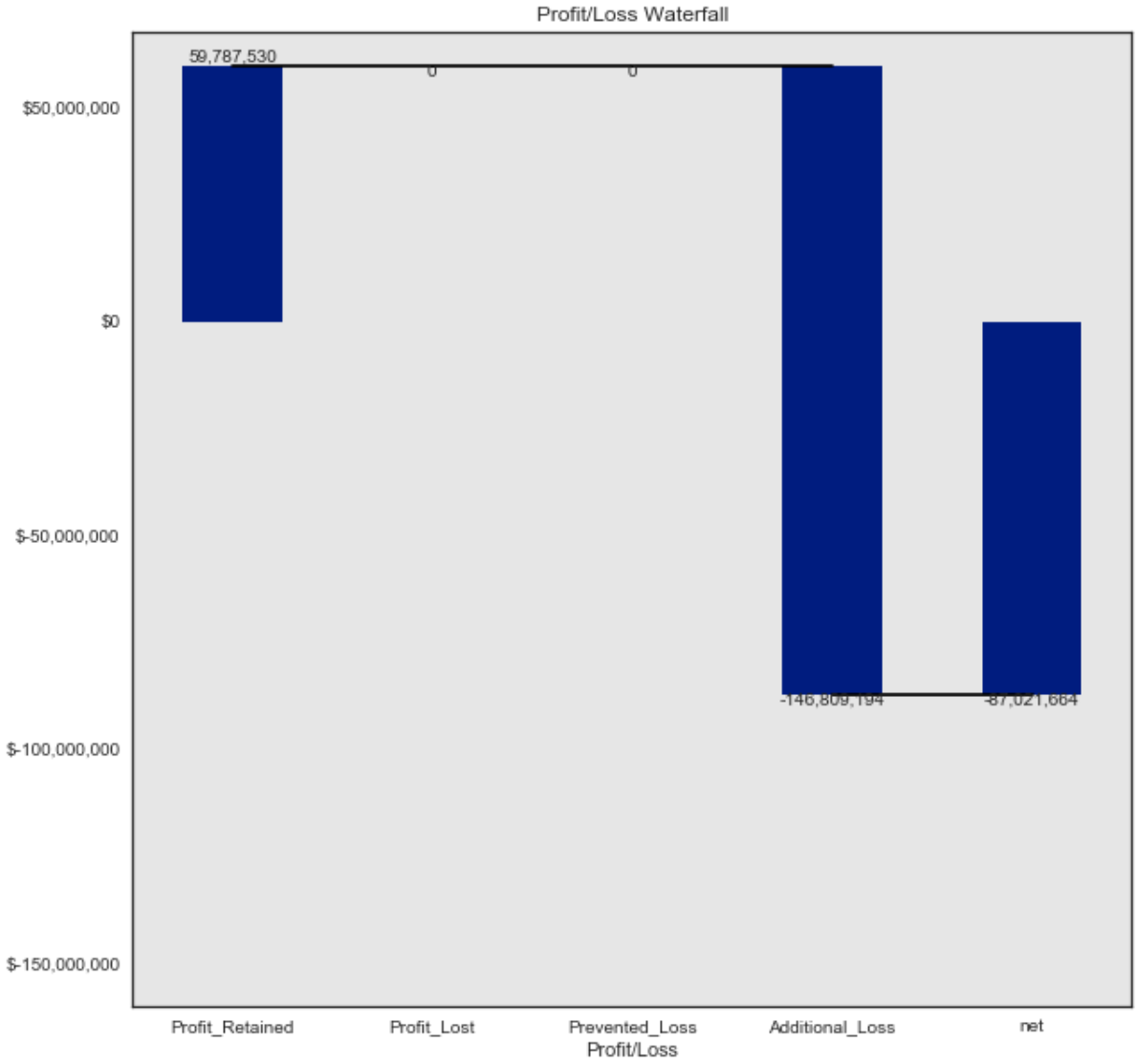
| | loan_amnt | total_pymnt |
|---|---|---|
| **0** | 4844927950 | 5.780869e+09 |
| **1** | 1100015550 | 6.780074e+08 |

```
Profit made in 2007-2014: $935,941,166.25
Losses made in 2007-2014:  $-422,008,154.04
```




### Loan Originations in 2016(Q1)

```
Profit made in Q1 2016: $59,787,529.73
Losses made in Q1 2016:  $-146,809,194.18
```




This doesn't seem too profitable (this could be because of many factors), so let's see if our model can do better. Ideally, our model should be able to approve the loans which are profitable but as long as we are not losing $80MM overall on money invested during Q1-2016, we should consider a worthwhile investment.

# III. Methodology

*(approx. 3-5 pages)*

## Data Preprocessing

In this section, all of your preprocessing steps will need to be clearly documented, if any were necessary. From the previous section, any of the abnormalities or characteristics that you identified about the dataset will be addressed and corrected here. Questions to ask yourself when writing this section:

- *If the algorithms chosen require preprocessing steps like feature selection or feature transformations, have they been properly documented?*
- *Based on the **Data Exploration** section, if there were abnormalities or characteristics that needed to be addressed, have they been properly corrected?*
- *If no preprocessing is needed, has it been made clear why?*

#### Data Distribution:

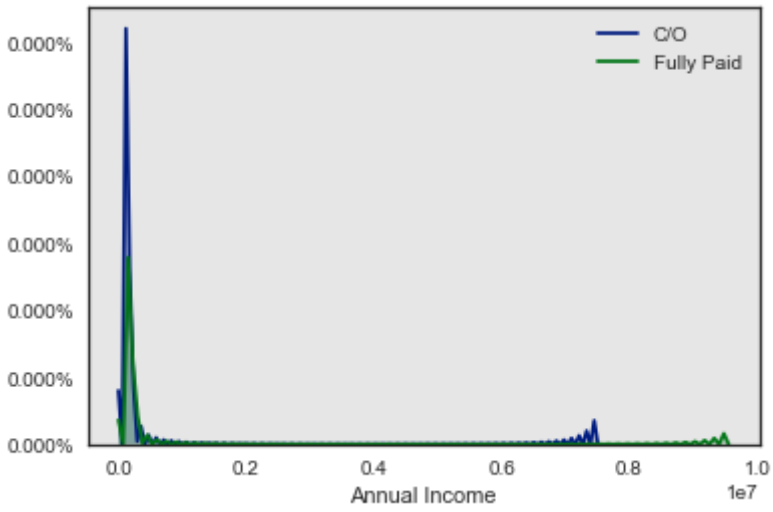
As part of data processing, our first aim will be to better understand the data distribution and then make necessary adjustments (if needed). We will start by observing the data distributions for numerical features.

Out[181]:

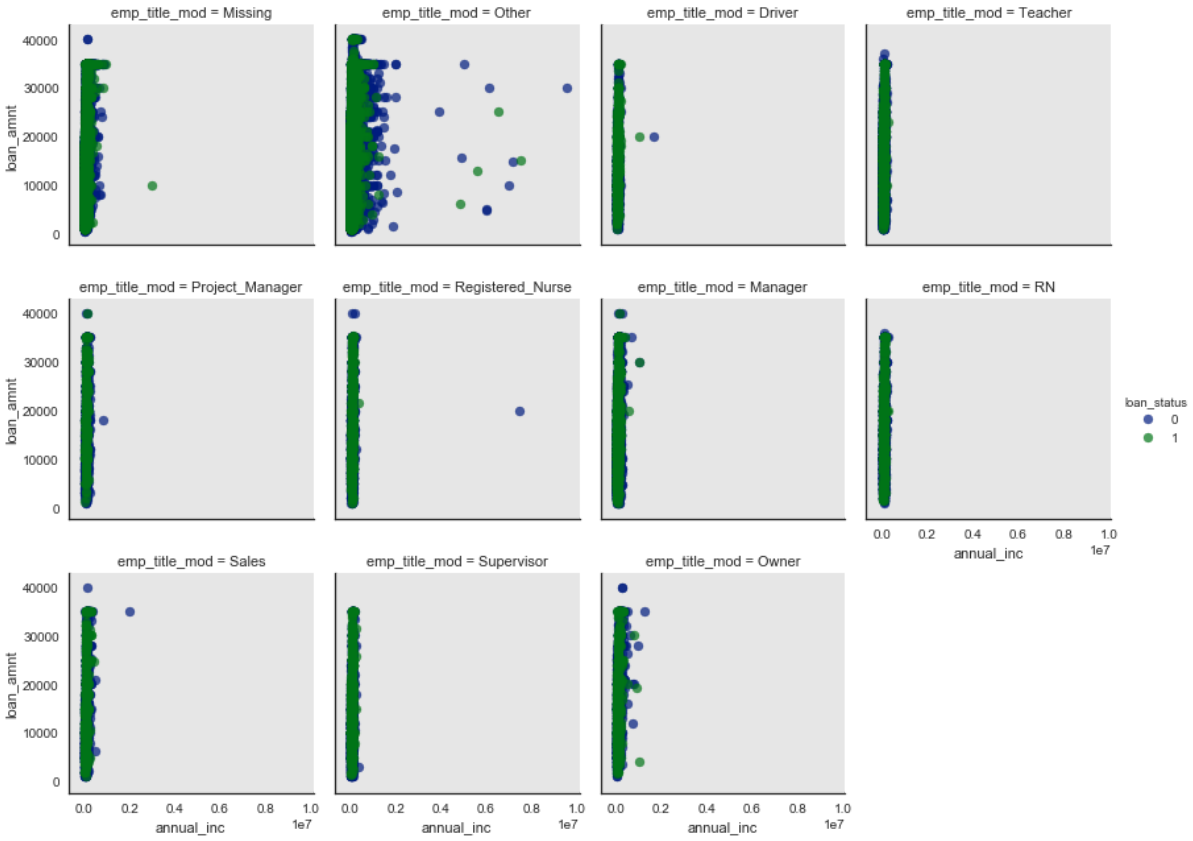
| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **funded_amnt** | 479661 | 14024 | 8270 | 500 | 7925 | 12000 | 19750 | 40000 |
| **loan_amnt** | 479661 | 14047 | 8282 | 500 | 7975 | 12000 | 19800 | 40000 |
| **term** | 479661 | 41 | 9 | 36 | 36 | 36 | 36 | 60 |
| **installment** | 479661 | 432 | 249 | 4 | 250 | 375 | 568 | 1536 |
| **total_pymnt** | 479661 | 14937 | 10083 | 0 | 7265 | 12390 | 20454 | 61557 |
| **loan_status** | 479661 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **annual_inc** | 479661 | 73528 | 59805 | 0 | 45000 | 63000 | 89000 | 9550000 |
| **emp_length** | 479661 | 5 | 3 | 0 | 2 | 6 | 10 | 10 |
| **int_rate** | 479661 | 13 | 4 | 5 | 10 | 13 | 16 | 28 |
| **emp_length_mi_flag** | 479661 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **emp_title_mi_flag** | 479661 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **issue_year** | 479661 | 2013 | 1 | 2007 | 2013 | 2014 | 2014 | 2016 |

#### Handle Extreme Values:

Annual Income feature appears to be a feature which is quite skewed. This is common for features of *wealth* type as small number of people can hold large amount of wealth.




We can observe this distribution by Employee Title as we possess that information:




Interestingly, there are instances where higher reported annual income has a quite a few instances of defaults. Mostly, the titles don't really have the disperion we observed in this particular category.

```
Out[184]: 0.05     27996.000
          0.25     45000.000
          0.50     63000.000
          0.75     89000.000
          0.95    150000.000
          0.99    248108.708
          Name: annual_inc, dtype: float64
```

There are many ways to handle such skewness in the data. A popular approach is to take the log of such fields. This causes issues if the raw data is 0. Hence, I will be using inverse sine hyperbolic transformation which has been accepted a reasonable transformation for such type of data.

The inverse hyperbolic sine transformation is defined as:

$$\texttt{f}(x) = (x_i + (x_i^2 + 1)^{\frac{1}{2}})$$

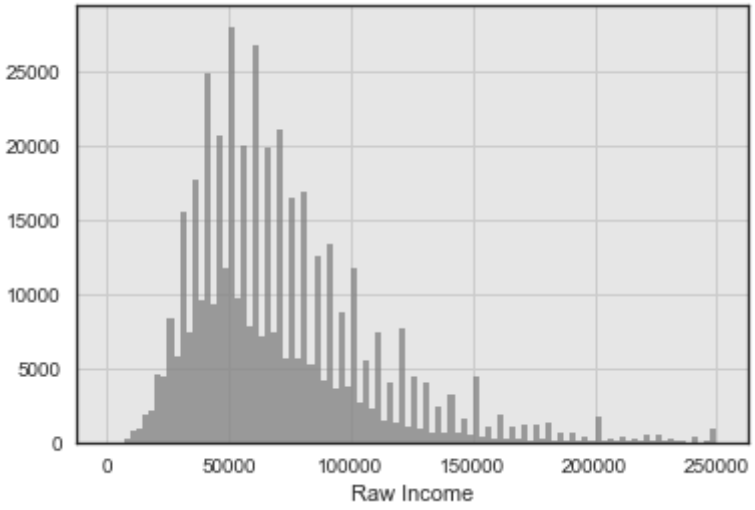
> Except for very small values of y, the inverse sine is approximately equal to log(2yi) or log(2)+log(yi), and so it can be interpreted in exactly the same way as a standard logarithmic dependent variable. For example, if the coefficient on "urban" is 0.1, that tells us that urbanites have approximately 10 percent higher wealth than non- urban people.) 1 (http://worthwhile.typepad.com/worthwhile_canadian_initi/2011/07/a-rant-on-inverse-hyperbolic-sine-transformations.html)

Reference:

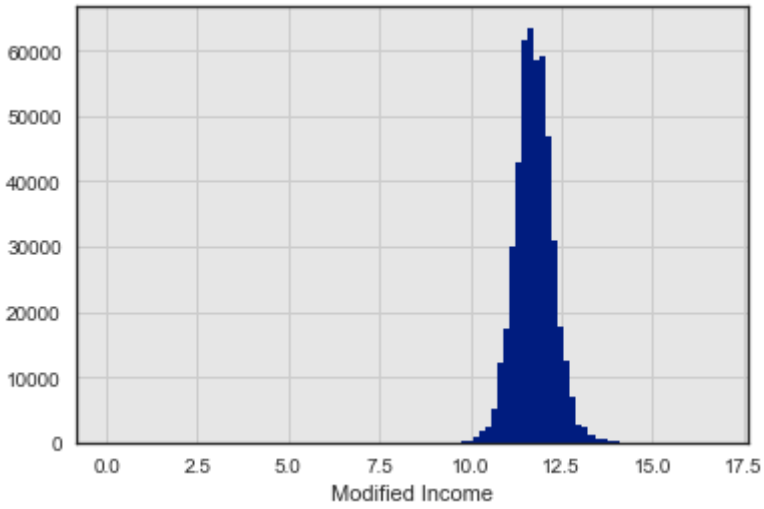
Data Transformations (https://robjhyndman.com/hyndsight/transformations/)

John B. Burbidge, Lonnie Magee and A. Leslie Robb, 1988. "Alternative Transformations to Handle Extreme Values of the Dependent Variable" Journal of the American Statistical Association Vol. 83, No. 401, pp. 123-127

**Post Transformation Analysis**:

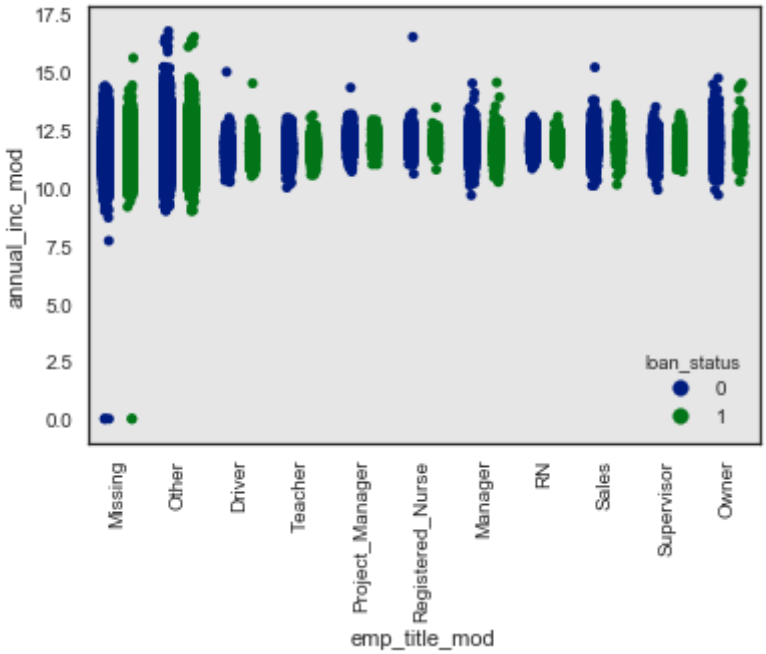



```
Out[185]: Text(0.5,0,u'Modified Income')
```




We will review the transformed field along with the loan status and employee title to ensure that it has preserved the overall distribution of the data.

```
Out[186]: <matplotlib.axes._subplots.AxesSubplot at 0x12728b950>
```




## Feature Engineering

In this section, we will try to derive features which may be difficult to use on their own. For example, earliest credit history date itself may not be a usable feature but if we can calculate the length of credit history from it, it can then be utilized in the future model fitting processes.

One key point I want to bring to attention is that ideally, we will like to perform a *fit* on training set of the data in this phase and using that on test as a *tranform* application at a later step. However, as I am not conducting any feature selection or deriving features based on any model applied on training data, I consider this step as more of data pre-processing. Hence, I am applying these treatments on the complete dataset.

As we will see later, I will split the data prior to fitting any model or conducting feature selection steps.

```
1944     1
1946     1
1948     1
1949     1
1950     3
1951     1
1953     4
1954     4
1955    10
1956     6
Name: earliest_cr_line_dt, dtype: int64
```

```
70    1
66    1
65    2
64    1
63    2
62    1
61    5
60    5
59    6
58    7
Name: credit_length_yrs, dtype: int64
```

As previously discussed, we will also derive a feature which captures what percentage of loan amount was funded for each loan. This will help us capture effect of these two variables in the model.

```
1.000000    477741
0.999091         1
0.999000         2
0.998276         1
0.997976         1
0.997917         1
0.997872         1
0.996875         2
0.995000         2
0.994737         1
Name: funded_amnt_pct, dtype: int64
```

```
479661
```

**Encode categorical data**

In order to use some of the algorithms (non-tree based sklearn implementations of linear models), we need to convert categorical variables into acceptable data types. One such trick is to convert such a variables to multiple binary variables corresponding to each value of that categorical column.

One downside of such a technique is that it can explode the number of features if the categorical variables have a large set of diverse possible entries. As we will see below, our original set of features increase from 26 to 124.

```
Following Variables will be dummified:
home_ownership
purpose
emp_title_mod
grade
issue_month
addr_state
verification_status_mod
Original Training Data Shape:  (479661, 25)
Dummified_Imputed Training Data Shape:  (479661, 122)
```

**Normalize numerical features:**

As part of our solution, we will be fitting linear models which return coefficients corresponding to every feature in our model. The scale of our features influence the size of these coefficients, i.e., a feature measured in Millions may get a small coefficient and can cause interpretation issues. To avoid this scenario, I will normalize some of the numerical features, particularly loan_amnt, and installment which are in thousands and hunderds of dollars respectively. We will make use of sklearn's preprocessing library (http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html) to conduct this tranformation.In general, applying normalization will rescale our metrics into the range of 0 to 1.

## Implementation

In this section, the process for which metrics, algorithms, and techniques that you implemented for the given data will need to be clearly documented. It should be abundantly clear how the implementation was carried out, and discussion should be made regarding any complications that occurred during this process. Questions to ask yourself when writing this section:

- *Is it made clear how the algorithms and techniques were implemented with the given datasets or input data?*
- *Were there any complications with the original metrics or techniques that required changing prior to acquiring a solution?*
- *Was there any part of the coding process (e.g., writing complicated functions) that should be documented?*

**Train, Test and Out of Time Validation split:**

We will make 2 copies of the data first:

- *LoanDataOutofTime*: Out of time sample - This will consist of loans originated in 2016(Q1) and will serve as our final validation sample.
- *LoanDataModel* : This will serve as our main copy of the data which we will use to train and test our model.

```
Dataset **LoanDataOutofTime** has the following description:
There are 51599 rows and 124 fields

Dataset **LoanDataModel** has the following description:
There are 428062 rows and 124 fields
```

**Sanity Check to ensure that LoanDataModel only consists of loans originated prior to 2016:**

```
Out[192]: 2014    208119
          2013    126872
          2012     53287
          2011     21721
          2010     11535
          2009      4716
          2008      1562
          2007       250
          Name: issue_year, dtype: int64
```

**Sanity Check to ensure out of time data consists of loans originated in 2016 only:**

```
Out[193]: 2016    51599
          Name: issue_year, dtype: int64
```

## Develop a Benchmark Model

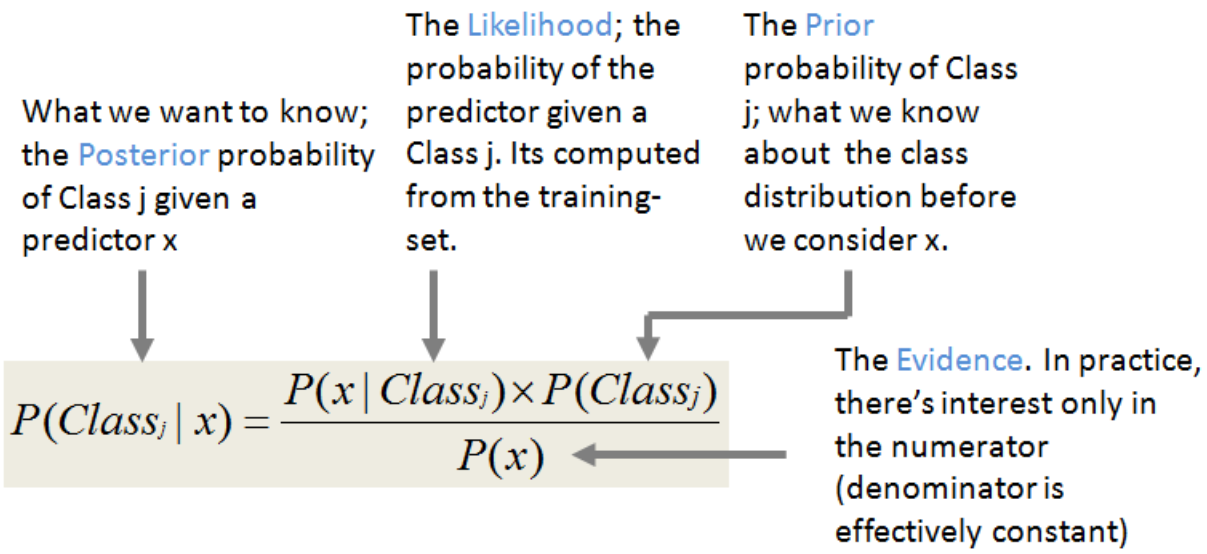
We will consider a Naive Bayes classifier to develop our Benchmark model using Loan Grade and Interest Rate (as previously discussed). It is a well know and successful approach used for classification problems.

References:

- Spam Filtering (https://en.wikipedia.org/wiki/Naive_Bayes_spam_filtering)
- Hybrid Recommender System (http://eprints.ecs.soton.ac.uk/18483)

This classifier is based on Bayes' Theorem which provides an "naive" way to calculate the probability of a hypothesis given our prior data. The reason it is called Naive is because it makes an assumption about the independence of each input variable. I am using it to build the benchmark model as it is a rather simplistic approach and only uses probabilities derived from the observed data to make predictions on new data.

Below image is a great reference to the apporoach which I found to be extremely useful. Credit goes to ShatterLine blog for the image. Let me simply provide some context on the below image by adding the following: P(B|A) is the (Posterior) probability of observing the class label (B) given that we have seen the data (A), accoring to Bayes' Theorem is same as the probability of observing the data (A) given that we have class (B), times the Prior probability of being in class (B), divided by the probability of observing (data) A.




Applying the independence assumption

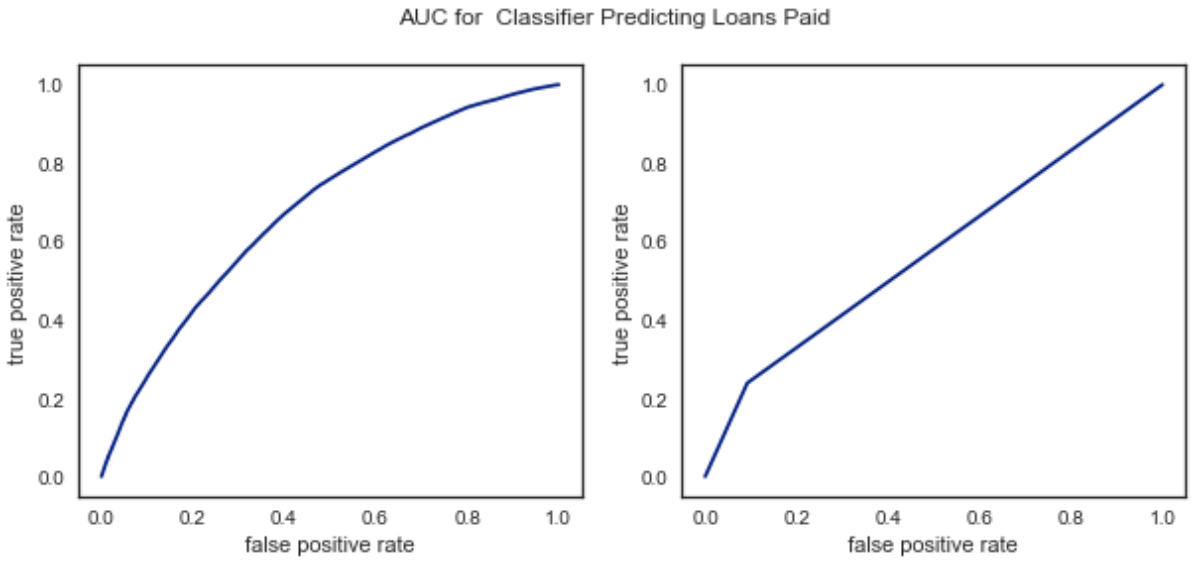
$$P(x \mid Class_j) = P(x_1 \mid Class_j) \times P(x_2 \mid Class_j) \times \ldots \times P(x_k \mid Class_j)$$

Substituting the independence assumption, we derive the Posterior probability of Class j given a new instance x' as...

$$P(Class_j \mid x') = P(x'_1 \mid Class_j) \times P(x'_2 \mid Class_j) \times \ldots \times P(x'_k \mid Class_j) \times P(Class_j)$$

We will use sklearn's implementaiton of the algorigthm to calculate this probability and produce some metrics(previously discussed) to assess the performance of the algorithm. I would later move these functions to the utlities file to make use of them and avoid re-writing the code.

```
Naive Predictor: [Accuracy score: 0.7938, F-score: 0.3180]
Accuracy:0.793

Classification report
             precision    recall  f1-score   support

          0       0.85      0.91      0.88     70959
          1       0.35      0.24      0.28     14654

avg / total       0.77      0.79      0.78     85613


Confusion matrix
[[64359  6600]
 [11160  3494]]

False-positive rate: [ 0.          0.09224401  1.        ]
True-positive rate:  [ 0.          0.23857851  1.        ]
Thresholds:          [2 1 0]
Figure(720x288)
```




In the above charts we can see that the Naive predictor performs fairly well using only two variables. Our measures of accuracy and f1-score are in an acceptable range and we are able to easily identify 30% of the defaulted loans. The earlier discussion around imbalanced target is important to be brought up as having an accuracy of 80% doesn't mean much in this particular problem.

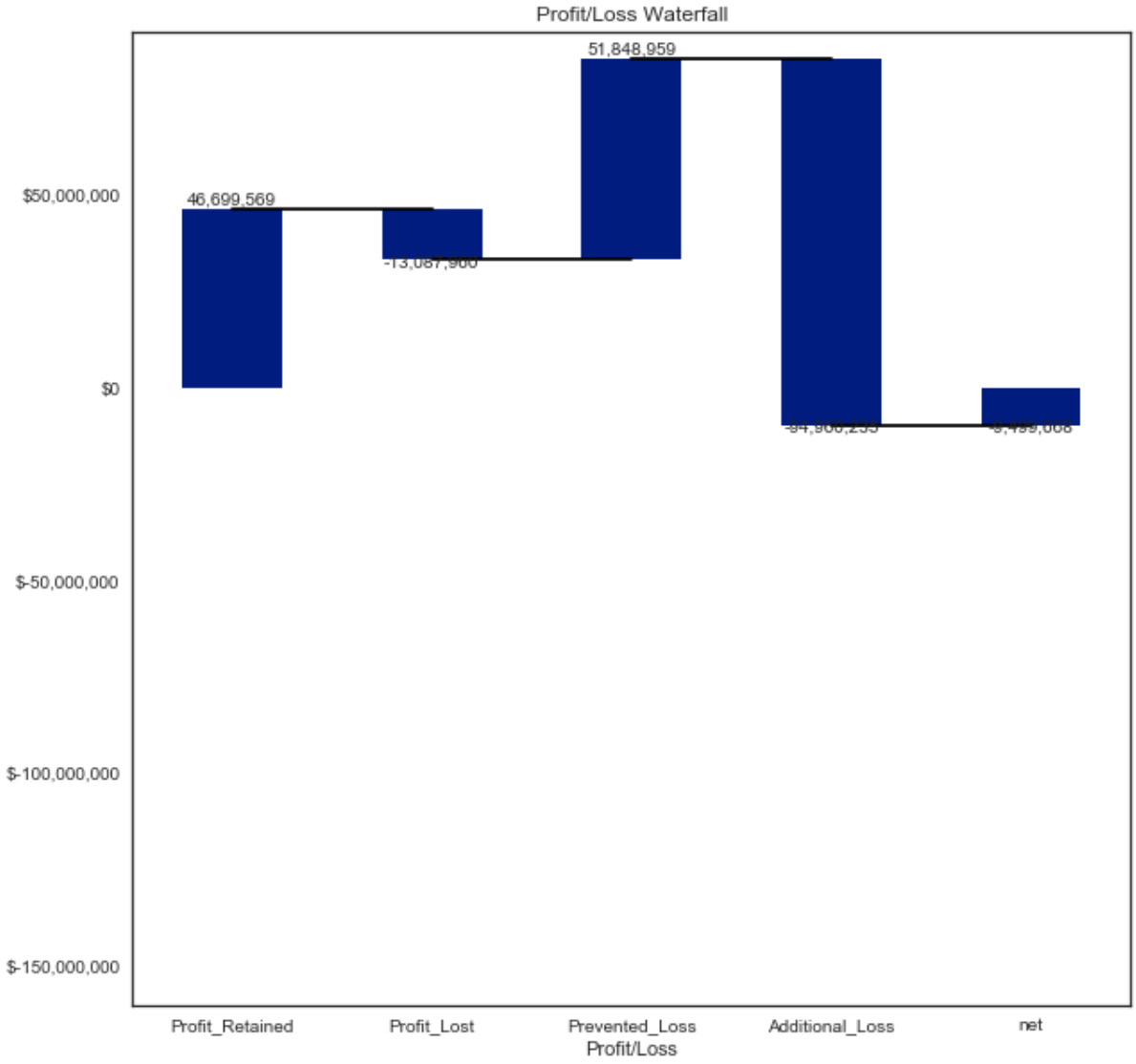
I find it easier to convey the performance measure of our model by displaying how much money we save/lose by implementing the model. We will use out of time dataset for this.

As we can observe below, by implementing the model, we will save a huge amount of money over the actual loss we incurred. If you recall from the previous discussion, Lending Club lost around 80MM on these loans. This simplistic model has cut that loss to only $9MM.

It achieves this by not approving loans which can go bad (saving $51MM). At the same time, due to classification error of our model, we also lose $13MM by not approving loans which would have fully paid back.

This is the balance we will need to maintain and this is where it is important to keep the final usage of the model in mind.

```
Accuracy:0.729

Classification report
             precision    recall  f1-score   support

          0       0.77      0.90      0.83     37623
          1       0.50      0.28      0.36     13976

avg / total       0.70      0.73      0.70     51599


Confusion matrix
[[33714  3909]
 [10075  3901]]


utilities.py:411: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy
  df["pred"] = pred
utilities.py:413: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy
  df["net_returns"] = df["total_pymnt"] - df["loan_amnt"]

Profit retained by approving good lenders : $46,699,569
Profit lost due to declining good lenders Profit lost : $13,087,960
Loss  prevented by not lending to defaulters: $-51,848,959
Additional loss due to approving defaulters: $-94,960,235
Total profit by implementing model : $ $-9,499,668
```




## Results on Out of Time data:

| Metric | Benchmark Predictor |
|---|---|
| Accuracy Score | 0.73 |
| F-score | 0.36 |
| Precision | 0.50 |
| Recall | 0.28 |
| Profit/Loss | -$9MM |

**Initial Model Evaluation:**

As a preliminary step, I will build upon what we learnt in the *Finding Donors* exercise. I will need to implement the following:

- Import the three supervised learning models :
  - Logistic Regression
  - Random Forest
  - Gradient Boosting
- Initialize these three models and store them in 'clf_A', 'clf_B', and 'clf_C'.
- Use a 'random_state' for each model you use, if provided.

Note: Here we use the default settings for each model — we will tune these models in a later section.

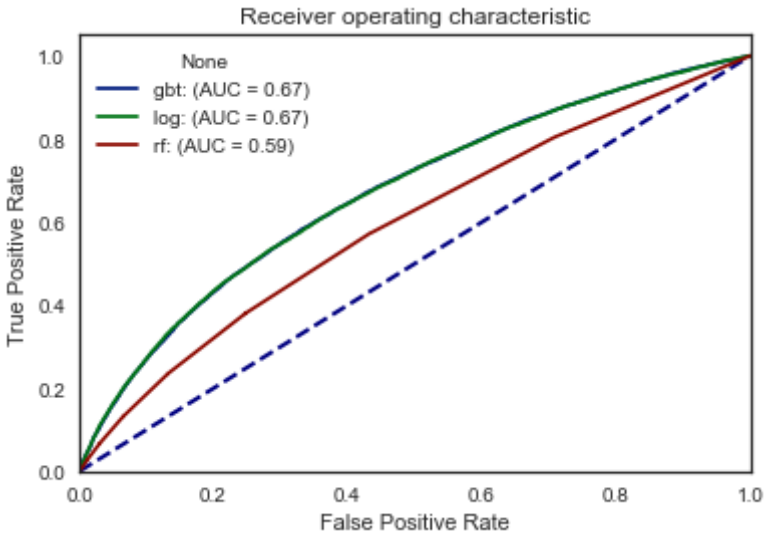
- Calculate the number of records equal to 1%, 10%, and 100% of the training data.
- Store those values in 'samples_1', 'samples_10', and 'samples_100' respectively.

**Test Train Split:**

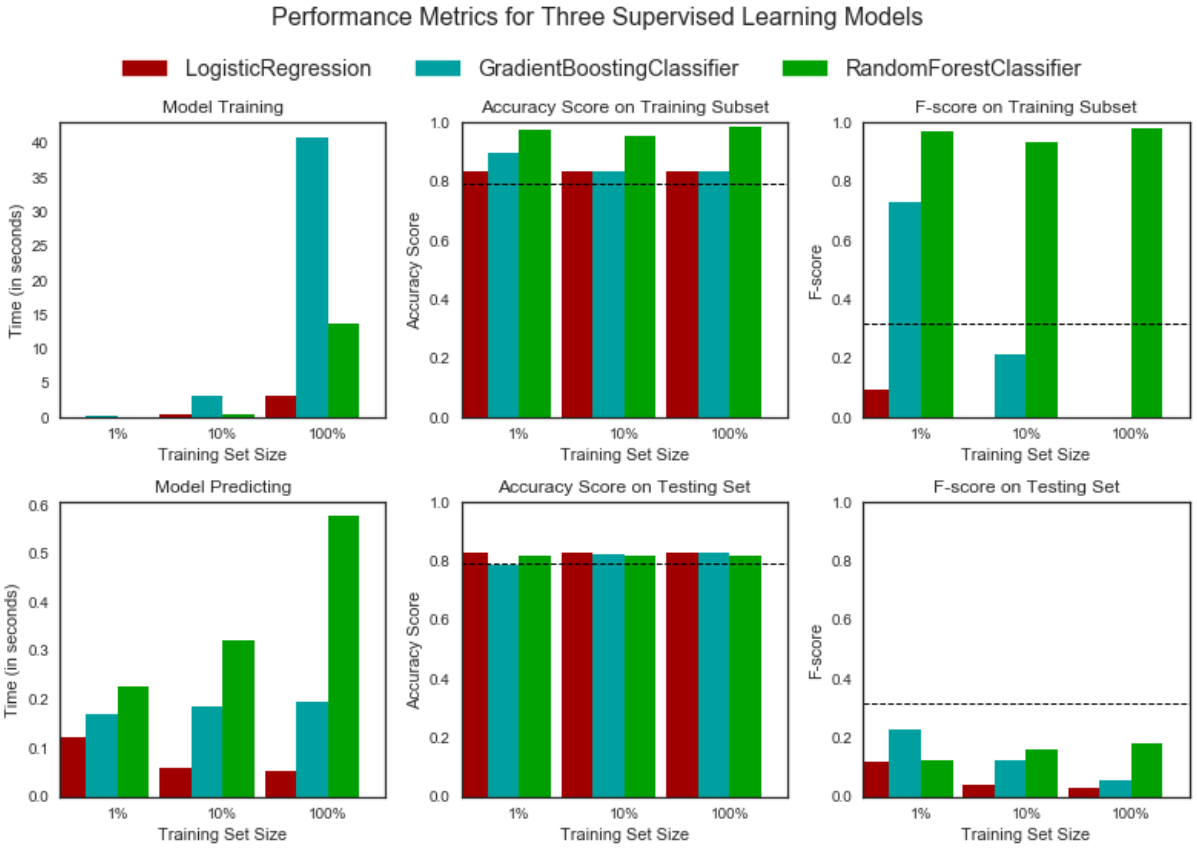
We first split our data in test vs train. At the same time, we want to discard a lot of features which shouldn't be part of our models.

```
Dataset **X** has the following description:
There are 428062 rows and 106 fields

Dataset **X_train** has the following description:
There are 342449 rows and 106 fields

Dataset **X_test** has the following description:
There are 85613 rows and 106 fields

Training/Test Ratio: 3.99996495859


LogisticRegression trained on 3424 samples.

/Users/xtl476/anaconda/envs/capstone/lib/python2.7/site-packages/sklear
n/metrics/classification.py:1135: UndefinedMetricWarning: F-score is il
l-defined and being set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)

LogisticRegression trained on 34244 samples.
LogisticRegression trained on 342449 samples.
GradientBoostingClassifier trained on 3424 samples.
GradientBoostingClassifier trained on 34244 samples.
GradientBoostingClassifier trained on 342449 samples.
RandomForestClassifier trained on 3424 samples.
RandomForestClassifier trained on 34244 samples.
RandomForestClassifier trained on 342449 samples.
```

```
No handlers could be found for logger "matplotlib.legend"
```




```
Out[199]: <matplotlib.axes._subplots.AxesSubplot at 0x156fe0a10>
```

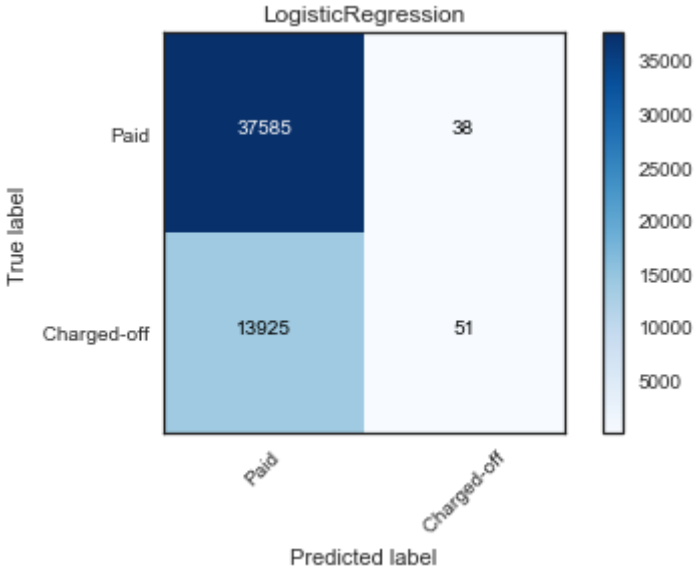


As it can be seen in the above charts:

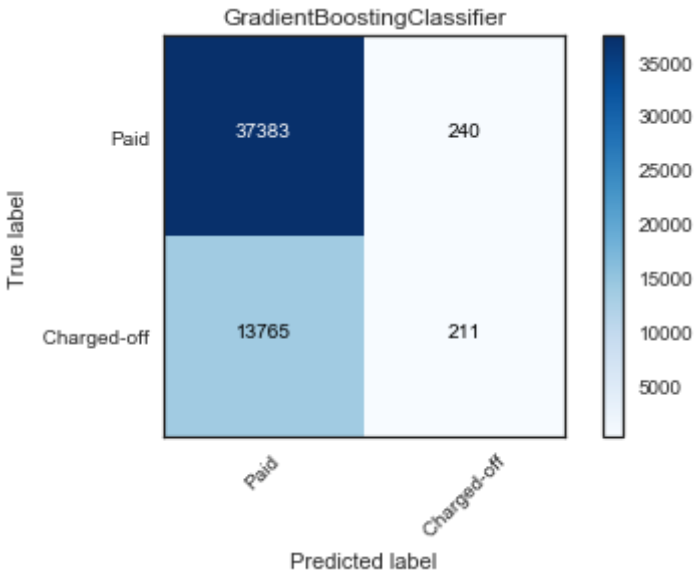
1. Logistic Regression wins when it comes to time to train the model, it is extremely fast compared to the other two methods tested above. However, it performs quite poorly overall on accuracy and F-score. Developing bad model fast shouldn't be our aim.
2. It is interesting to note that Random Forest is quite consistent across data sizes for both accuracy and F-score. Gradient Boosting didn't perform too well and it could be because of the paramters we passed when intializing the model. Due to computing issues, I tried to test it with small number of instances in this phase. We will later see how to tune these models.

To get a better understanding our models, let's observe if it was able to identify the bad loans on our Out of time dataset. We will use the confusion matrix here to easily interpret the results.

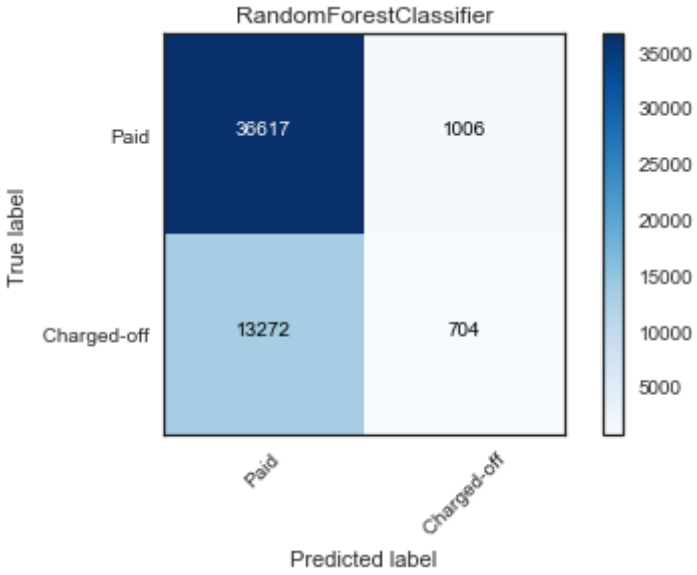
```
Model Summary for LogisticRegression
```




```
Model Summary for GradientBoostingClassifier
```



```
Model Summary for RandomForestClassifier
```

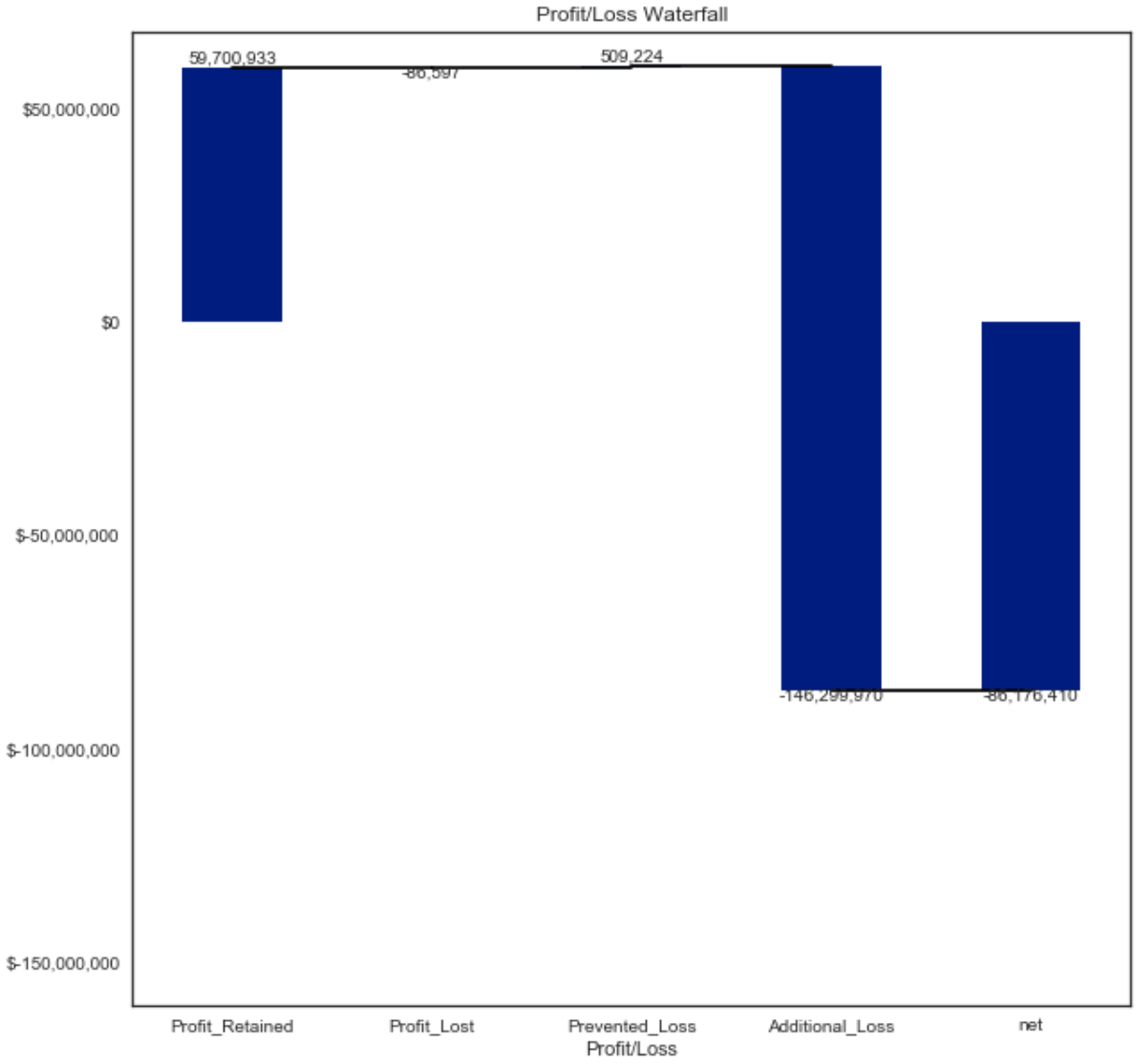


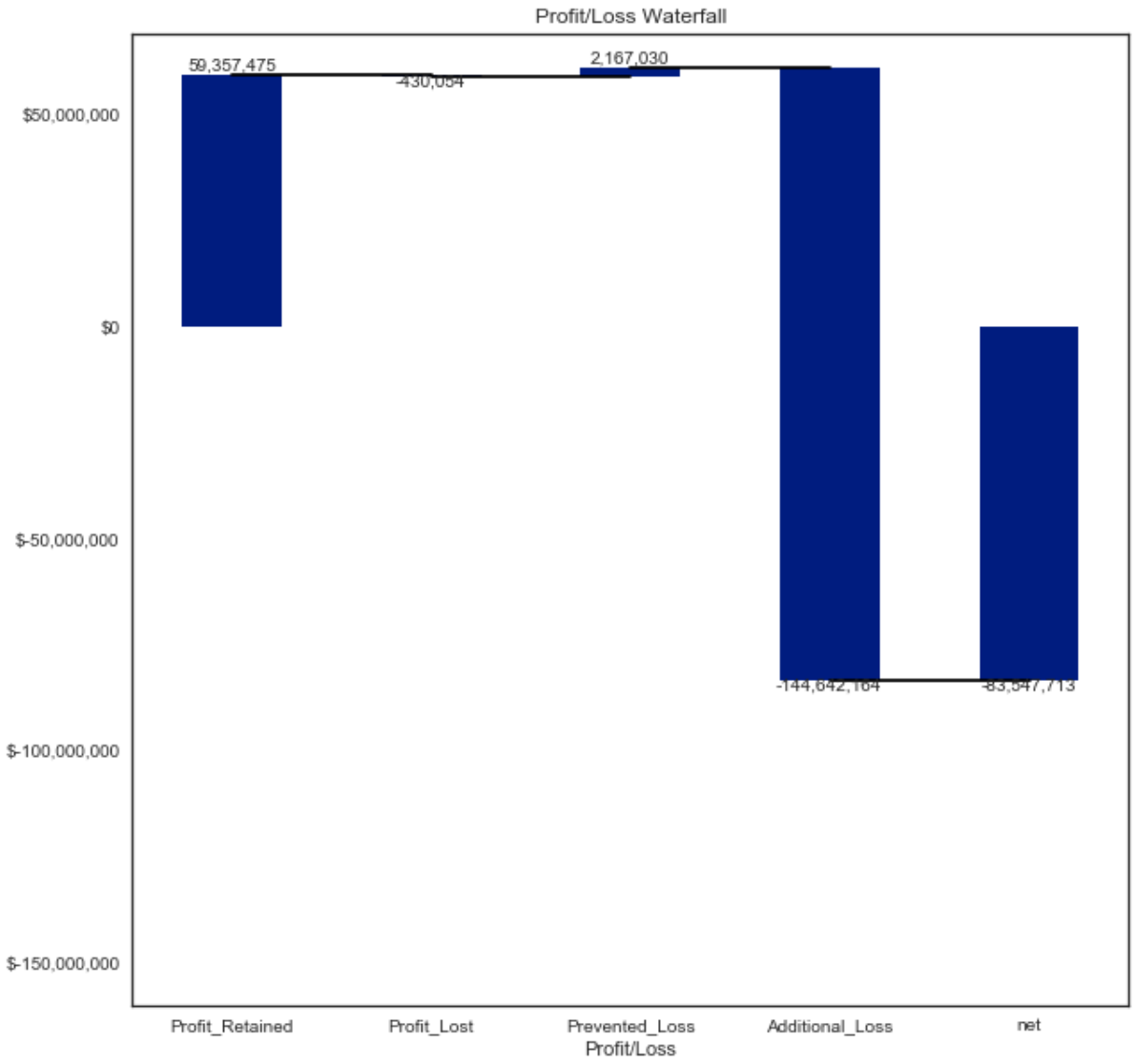

Given the above findings, we can easily conclude that our model performed quite poorly and only Random forest was able to identify atleast 500 bad loans among 48,000 loans. Needless to say, these models will not save us much money but let's double check the same.

An interesting observation in the below results is how each model focuses on completely different areas. I will exclude Logistic Regression from our discussion for now as it barely identified any loans.

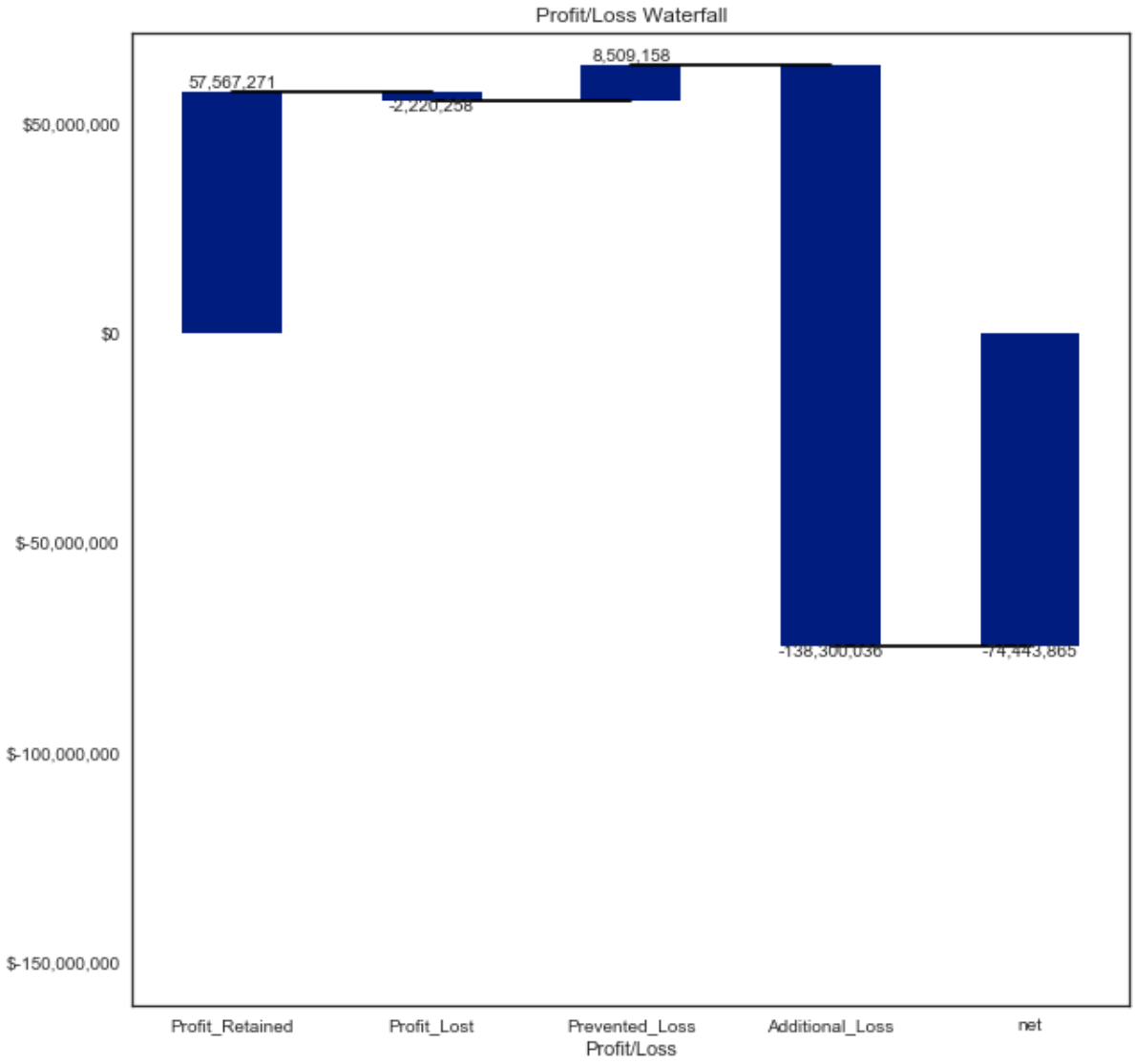
While Random Forest was able to identify bad loans and save us some money, it also incorrectly identified some good loans as bad, the volume of the same was quite low in Gradient Boosting.

```
***Model Profit from for LogisticRegression***

Profit retained by approving good lenders : $59,700,933
Profit lost due to declining good lenders Profit lost : $86,597
Loss  prevented by not lending to defaulters: $-509,224
Additional loss due to approving defaulters: $-146,299,970
Total profit by implementing model : $ $-86,176,410

***Model Profit from for GradientBoostingClassifier***

Profit retained by approving good lenders : $59,357,475
Profit lost due to declining good lenders Profit lost : $430,054
Loss  prevented by not lending to defaulters: $-2,167,030
Additional loss due to approving defaulters: $-144,642,164
Total profit by implementing model : $ $-83,547,713
```


Profit/Loss Waterfall
59,700,933
-86,597
509,224
-146,299,970
-86,176,410
$50,000,000
$0
$-50,000,000
$-100,000,000
$-150,000,000
Profit_Retained
Profit_Lost
Prevented_Loss
Additional_Loss
net
Profit/Loss


Profit/Loss Waterfall
59,357,475
-430,054
2,167,030
-144,642,164
-83,547,713
$50,000,000
$0
$-50,000,000
$-100,000,000
$-150,000,000
Profit_Retained
Profit_Lost
Prevented_Loss
Additional_Loss
net
Profit/Loss

```
***Model Profit from for RandomForestClassifier***

Profit retained by approving good lenders : $57,567,271
Profit lost due to declining good lenders Profit lost : $2,220,258
Loss  prevented by not lending to defaulters: $-8,509,158
Additional loss due to approving defaulters: $-138,300,036
Total profit by implementing model : $ $-74,443,865
```




## Refinement

In this section, you will need to discuss the process of improvement you made upon the algorithms and techniques you used in your implementation. For example, adjusting parameters for certain models to acquire improved solutions would fall under the refinement category. Your initial and final solutions should be reported, as well as any significant intermediate results as necessary. Questions to ask yourself when writing this section:

- *Has an initial solution been found and clearly reported?*
- *Is the process of improvement clearly documented, such as what techniques were used?*
- *Are intermediate and final solutions clearly reported as the process is improved?*

In order to refine our models, we will use the following approaches to improve our models:

- Addressing class imabalance issue - To address this issue, we will make use of sklearn's class_weight parameter (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) for Logistic regression and Random Forest algorithms. When the “balanced” mode is used, it automatically adjusts “weights of our samples accordingly. It does that by setting the weights to inversely proportional to class frequencies” in the data using “n_samples/(n_classes*np.bincount(y))”

Example:

```
from sklearn.utils.class_weight import compute_sample_weight
    y = [1,1,1,1,0,0,1]
    print(compute_sample_weight(class_weight='balanced', y=y))
    # The above is same as sample_weight = Y_train.shape[0]/(2*np.bincount(Y
_train))
    #sample_weight[Y_train.values]
    [ 0.7   0.7   0.7   0.7   1.75  1.75  0.7 ]
```

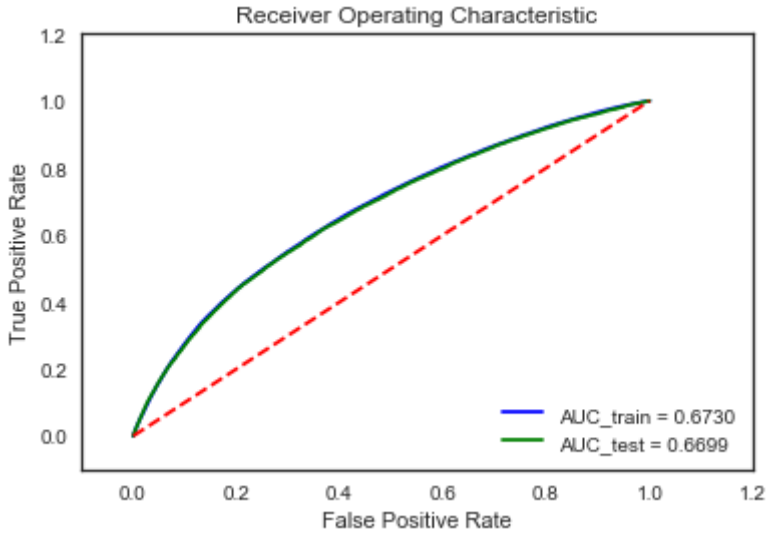
- Hyper-parameter tuning for ensemble approaches: While powerful, ensemble methods require careful hyperparameter tuning. For example, as our base learners are trees, we need to identify how many trees we want to build and at how much depth. All such decisions can have a huge impact on the final outcome.
- Identifying Interaction terms: As linear models can't identify interaction among variables, we will try to identify all such interactions using sklearn's polynomialFeatures library.
- Conducting feature selection for Random Forest and Logistic Regression
  - We will make use of Borutapy (https://pypi.python.org/pypi/Boruta/0.1.5) package for this purpose. - Using feature importances from Random Forest implementation has been identified problematic as it may not lead to best set of features. Adding random feature to the dataset helps identify whether our features are better than random noise. Instead of uniform random variables, we will instead randomize our existing features and use them to rank feature importances. It trains a random forest classifier on the extended data set and applies a feature importance measure (the default is Mean Decrease Accuracy) to evaluate the importance of each feature. Only features which are deemed to have higher importance than the best of it's shadow features are kept in the final implementation.shadow features
  - We will also conduct a stepwise feature (https://en.wikipedia.org/wiki/Stepwise_regression) selection method.
  - Lastly, we will use SelectKbest implementation of sklearn.

References:

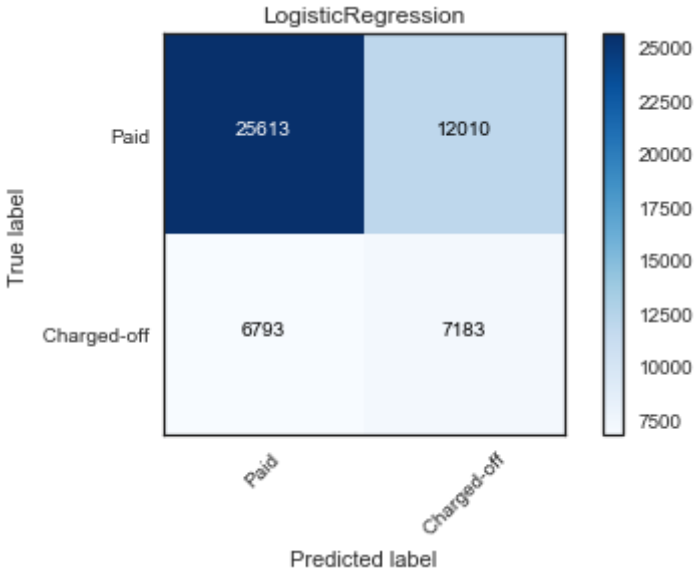
Boruta (https://www.jstatsoft.org/article/view/v036i11/v36i11.pdf)

## Model 1: Logistic Regression- Balanced class_weight

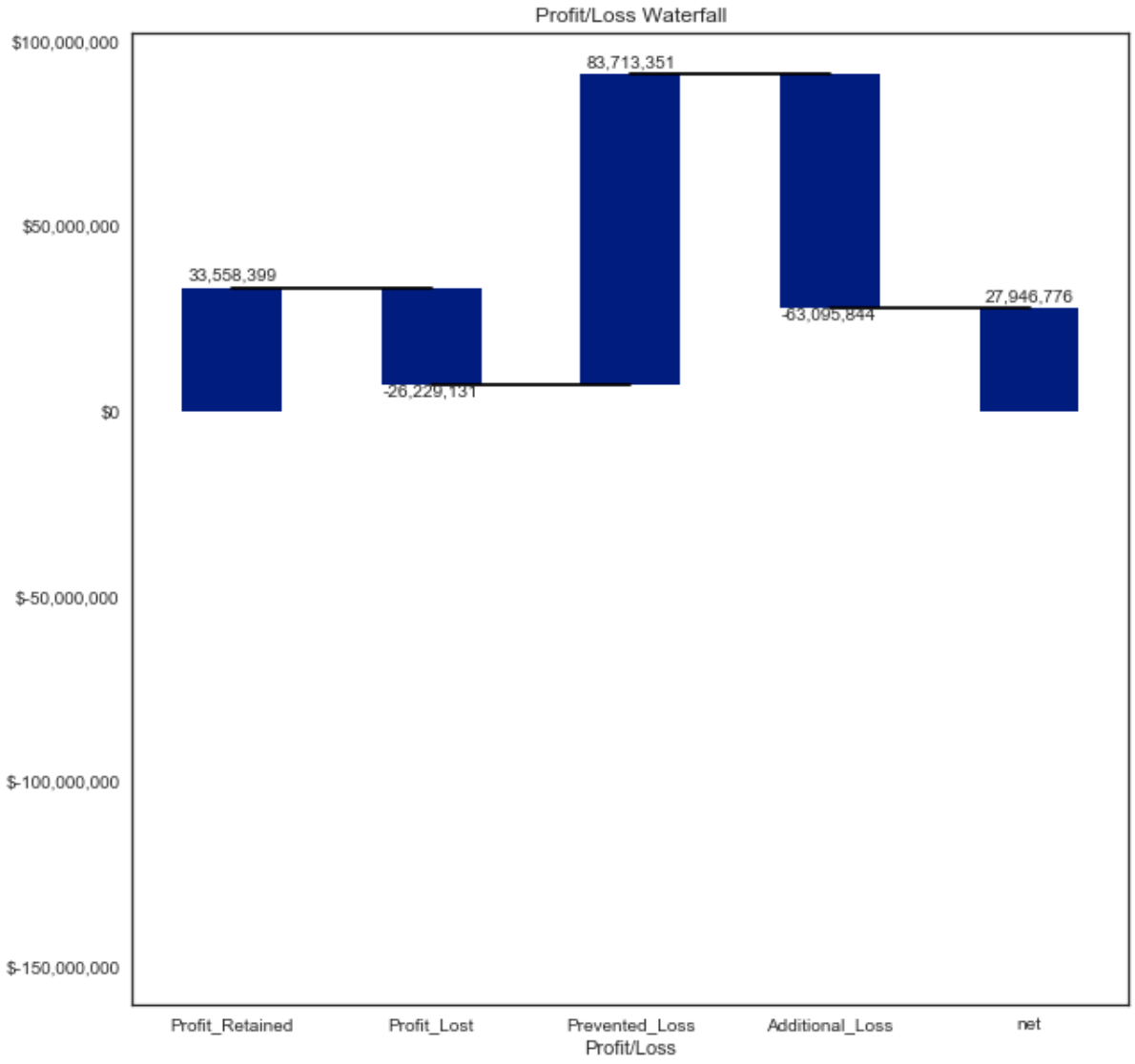
```
Out[204]: LogisticRegression(C=1.0, class_weight='balanced', dual=False,
                  fit_intercept=True, intercept_scaling=1, max_iter=100,
                  multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
                  solver='liblinear', tol=0.0001, verbose=0, warm_start=False)

          Train Score:  0.672485537993
          Test Score:  0.672970226484
          Classes Predicted:  [0 1]
          Logistic Regression Model Score:  0.672485537993

           Confusion Matrix:
          [[49586 21401]
           [ 6597  8029]]

           Classification Report:
                       precision    recall  f1-score   support

                    0       0.88      0.70      0.78     70987
                    1       0.27      0.55      0.36     14626

          avg / total       0.78      0.67      0.71     85613
```




```
          Accuracy:0.636

          Classification report
                       precision    recall  f1-score   support

                    0       0.79      0.68      0.73     37623
                    1       0.37      0.51      0.43     13976

          avg / total       0.68      0.64      0.65     51599


          Confusion matrix
          [[25613 12010]
           [ 6793  7183]]
```


LogisticRegression
Paid
Charged-off
True label
25613
12010
6793
7183
25000
22500
20000
17500
15000
12500
10000
7500
Paid
Charged-off
Predicted label

```
Profit retained by approving good lenders : $33,558,399
Profit lost due to declining good lenders Profit lost : $26,229,131
Loss  prevented by not lending to defaulters: $-83,713,351
Additional loss due to approving defaulters: $-63,095,844
Total profit by implementing model : $ $27,946,776
```




Simply addressing the class imbalance issue has made a huge difference in the results of logistic regression. Let's make some quick observations about the changes due to this refinement:

1. While the accuracy takes a hit, we still make considerable profit.
2. Key benefit comes from increase in Recall, which highlights the fact that our model has increased capacity to identify the type of loans which tend to default.
3. This increase in Recall comes at the price of decrease in Precision, not every loan that the model is identifying as a bad loan is actually defaulting.
4. This incorrect classfication of good loans as bad costs us $26MM but due to higher precision, we are able to save $84MM as well.

## Results on Out of Time data:

| Metric | Benchmark Predictor | Logistic Regression (Balanced) |
|---|---|---|
| Accuracy Score | 0.73 | .64 |
| F-score | 0.36 | 0.43 |
| Precision | 0.49 | 0.37 |
| Recall | 0.28 | 0.51 |
| Profit/Loss | -$7MM | +$27MM |

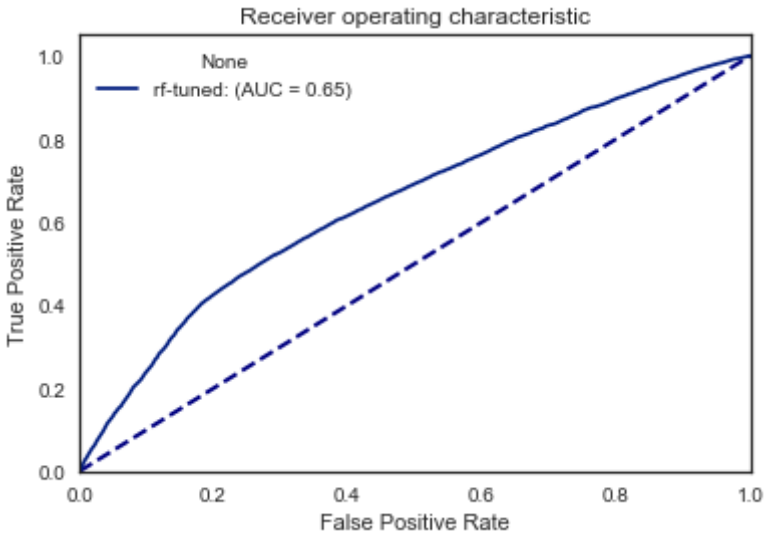
**Model 2: Tuned-Random Forest:**

In this step, we will be improving the performance of Random Forst by tuning the hyper-parameters for the algorithm. For this will be making use of the sample code (http://scikit-learn.org/0.15/auto_examples/randomized_search.html) available on sklearn's page.

```
Model with rank: 1
Mean validation score: 0.6803)
Parameters: {'max_leaf_nodes': 20, 'min_samples_leaf': 20, 'n_estimator
s': 20, 'min_weight_fraction_leaf': 0.1, 'min_samples_split': 20, 'max_
depth': 5}

Model with rank: 2
Mean validation score: 0.6720)
Parameters: {'max_leaf_nodes': 40, 'min_samples_leaf': 10, 'n_estimator
s': 20, 'min_weight_fraction_leaf': 0.1, 'min_samples_split': 15, 'max_
depth': 3}

Model with rank: 3
Mean validation score: 0.6717)
Parameters: {'max_leaf_nodes': 20, 'min_samples_leaf': 10, 'n_estimator
s': 50, 'min_weight_fraction_leaf': 0.1, 'min_samples_split': 15, 'max_
depth': 3}

Model with rank: 4
Mean validation score: 0.6703)
Parameters: {'max_leaf_nodes': 40, 'min_samples_leaf': 10, 'n_estimator
s': 100, 'min_weight_fraction_leaf': 0.1, 'min_samples_split': 20, 'max
_depth': 3}
```

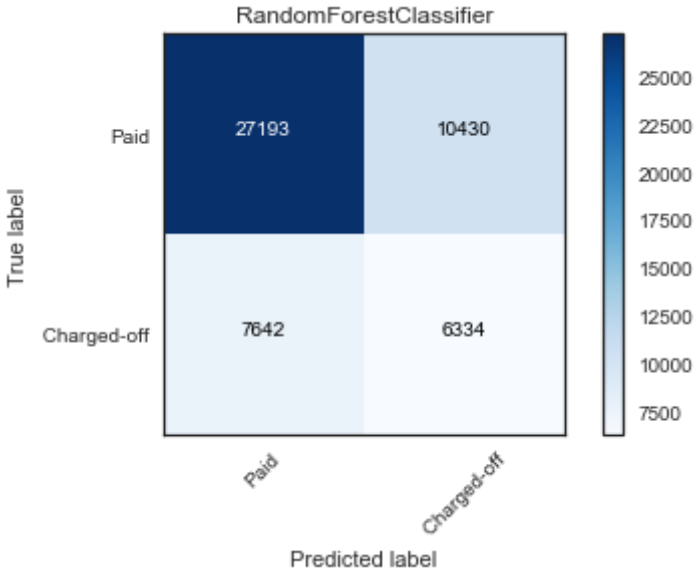
Next, we fit our model using the parameters obtained from the above tuning step.

```
Out[212]: RandomForestClassifier(bootstrap=True, class_weight='balanced',
                       criterion='gini', max_depth=5, max_features='auto',
                       max_leaf_nodes=40, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=20,
                       min_samples_split=20, min_weight_fraction_leaf=0.1,
                       n_estimators=20, n_jobs=4, oob_score=False, random_state=No
          ne,
                       verbose=0, warm_start=False)
```

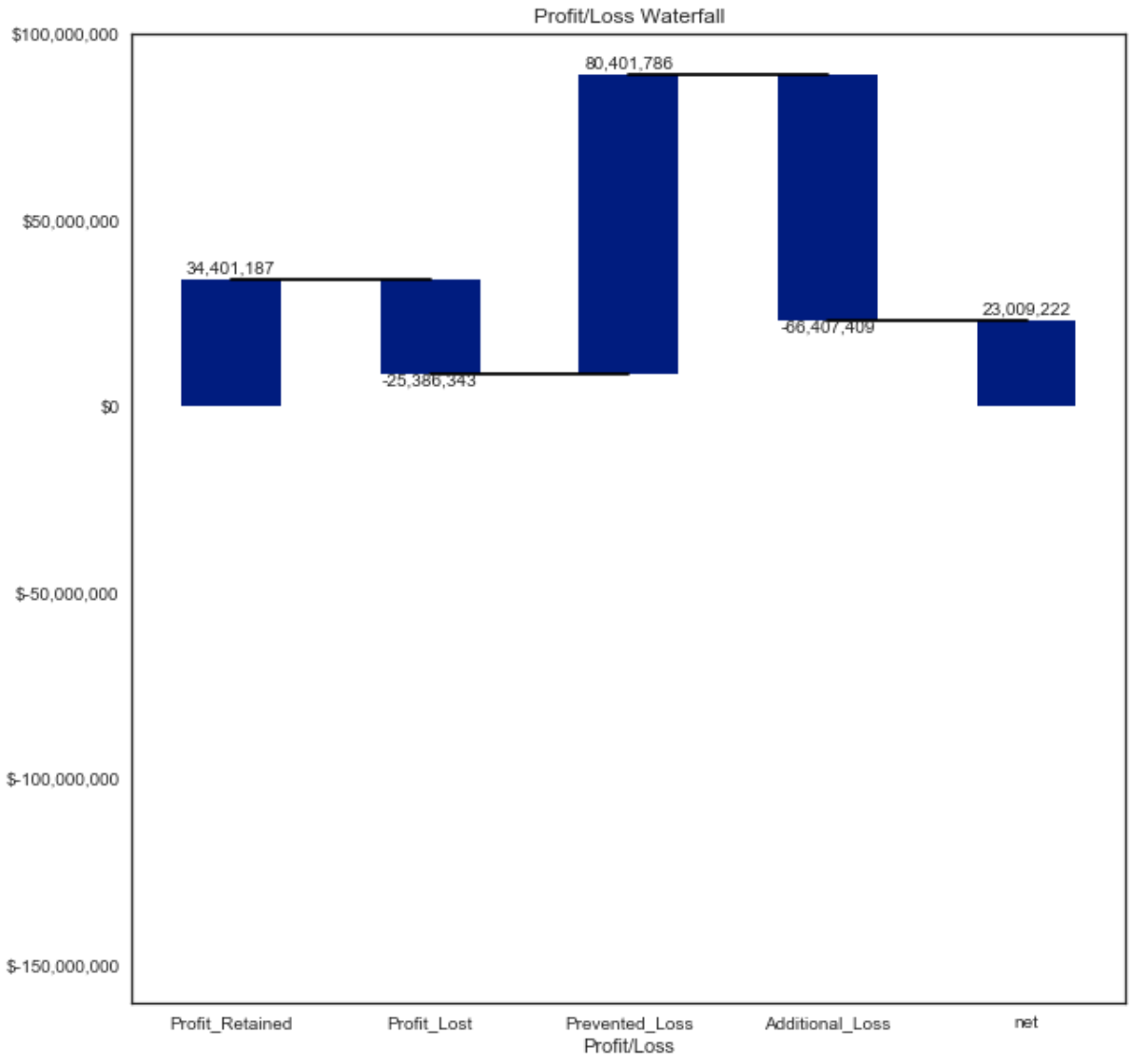



```
Out[213]: <matplotlib.axes._subplots.AxesSubplot at 0x122ada2d0>
```

**Check the tuned model on the out of time data:**



```
Accuracy:0.650

Classification report
             precision    recall  f1-score   support

          0       0.78      0.72      0.75     37623
          1       0.38      0.45      0.41     13976

avg / total       0.67      0.65      0.66     51599


Confusion matrix
[[27191 10432]
 [ 7643  6333]]


Profit retained by approving good lenders : $34,401,187
Profit lost due to declining good lenders Profit lost : $25,386,343
Loss  prevented by not lending to defaulters: $-80,401,786
Additional loss due to approving defaulters: $-66,407,409
Total profit by implementing model : $ $23,009,222
```



Compared to our previous Logistic Regression Model, Random forest does a better job of having higher precision, which means that it is better at identifying whether a loan is good or bad. However, where it lags behind Logistic Regression is when it fails to recall as many bad loans as Logistic Regression. It still ends up saving us as a lot of money though.

## Results on Out of Time data:

| Metric | Benchmark Predictor | Logistic Regression (Balanced) | Random Forest(Tuned) |
|---|---|---|---|
| Accuracy Score | 0.73 | .64 | .65 |
| F-score | 0.36 | 0.43 | 0.41 |
| Precision | 0.49 | 0.37 | 0.38 |
| Recall | 0.28 | 0.51 | 0.45 |
| Profit/Loss | -$7MM | **+$27MM** | +$23MM |

## Feature Selection:

As previously discussed, in this phase of refinement, we will be training our model on a subset of features and we will be making use of various techniques to conduct feature selection. We begin with using BorutaPy package we earlier discussed. Due to limitations around computing power, I am only requesting 15 iteations of this package using our previously tuned model. Ideally, I should be re-tuning our model after feature selection but both feature selection and hyper-parameter have turned out to be computationally intensive along with time-consuming. Hence, I am being very specific around the tuning parameters.

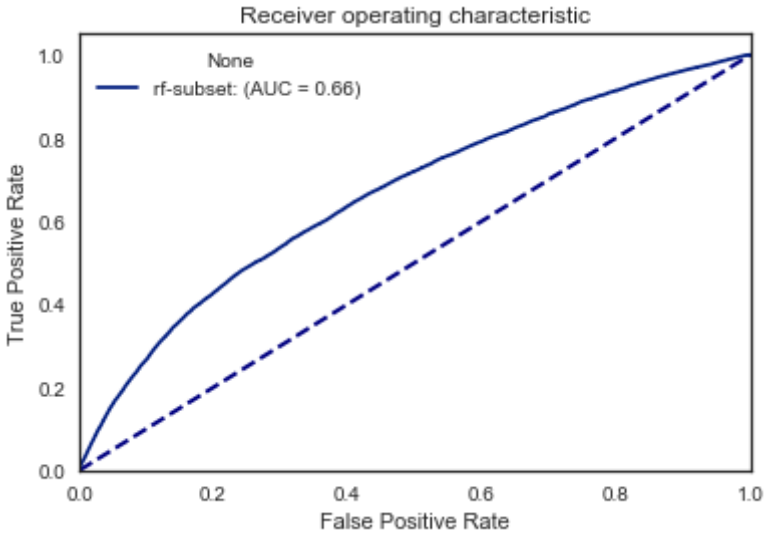
**Using BorutaPy**

```
Iteration:      1 / 15
Confirmed:      0
Tentative:      106
Rejected:       0
Iteration:      2 / 15
Confirmed:      0
Tentative:      106
Rejected:       0
Iteration:      3 / 15
Confirmed:      0
Tentative:      106
Rejected:       0
Iteration:      4 / 15
Confirmed:      0
Tentative:      106
Rejected:       0
Iteration:      5 / 15
```

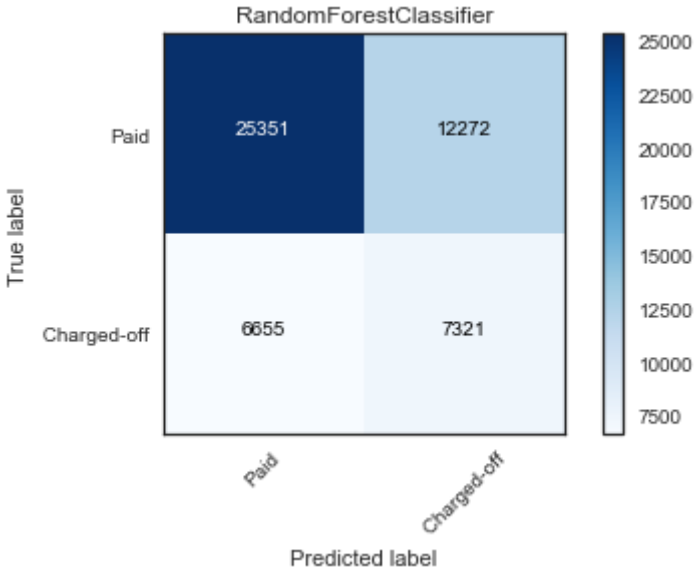
```
Confirmed:      0
Tentative:      106
Rejected:       0
Iteration:      6 / 15
Confirmed:      0
Tentative:      106
Rejected:       0
Iteration:      7 / 15
Confirmed:      0
Tentative:      106
Rejected:       0
Iteration:      8 / 15
Confirmed:      20
Tentative:      9
Rejected:       77
```

```
/Users/xtl476/anaconda/envs/capstone/lib/python2.7/site-packages/borut
a/boruta_py.py:418: RuntimeWarning: invalid value encountered in greate
r
  hits = np.where(cur_imp[0] > imp_sha_max)[0]
```

```
Iteration:      9 / 15
Confirmed:      20
Tentative:      9
Rejected:       77
Iteration:      10 / 15
Confirmed:      20
Tentative:      9
Rejected:       77
Iteration:      11 / 15
Confirmed:      20
Tentative:      9
Rejected:       77
Iteration:      12 / 15
Confirmed:      20
Tentative:      6
Rejected:       80
Iteration:      13 / 15
Confirmed:      20
Tentative:      6
Rejected:       80
Iteration:      14 / 15
Confirmed:      20
Tentative:      6
Rejected:       80


BorutaPy finished running.

Iteration:      15 / 15
Confirmed:      20
Tentative:      4
Rejected:       80
Train Score:  0.667357767142
```



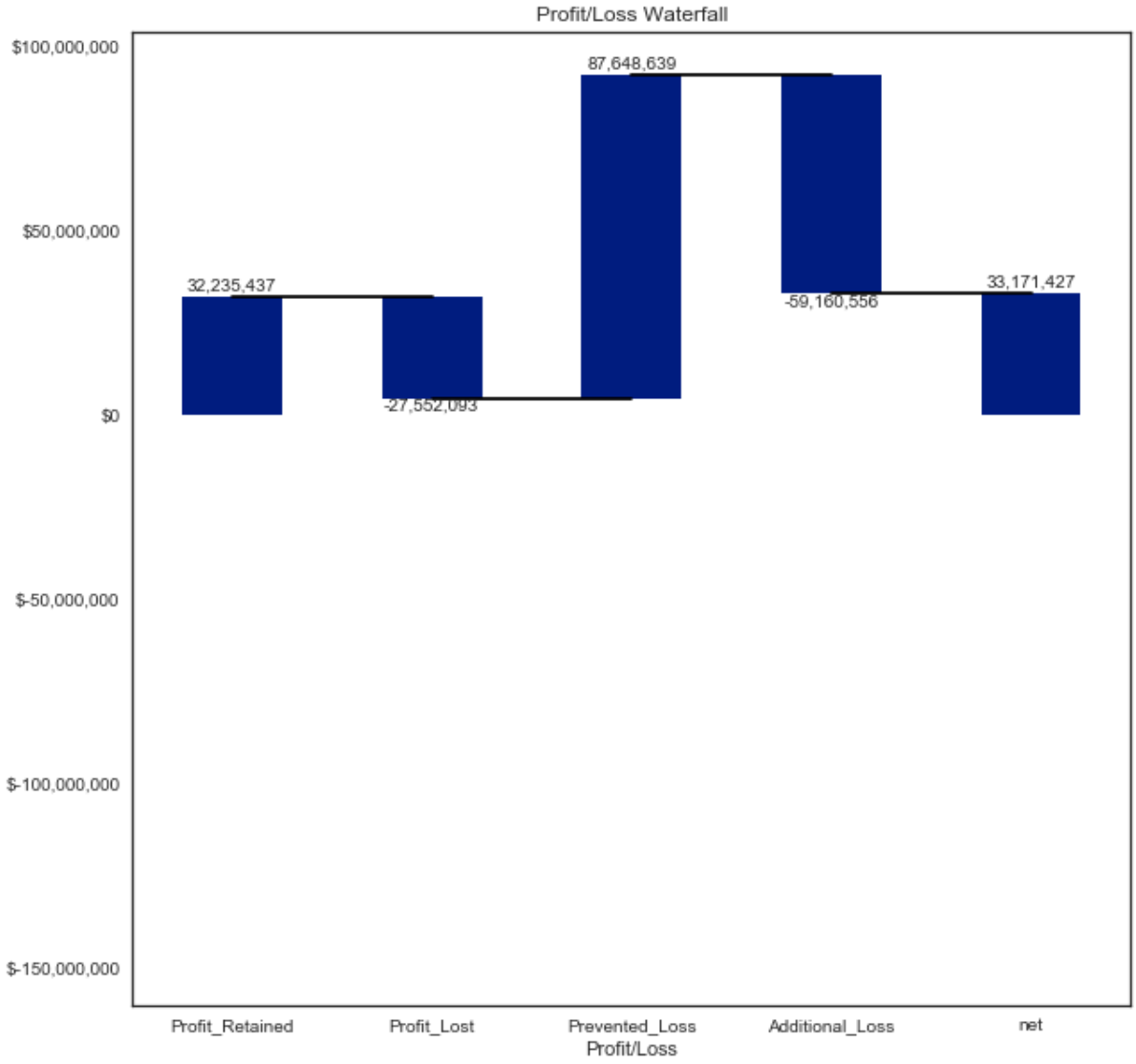

Out[222]: <matplotlib.axes._subplots.AxesSubplot at 0x11af02c10>



```
Accuracy:0.626

Classification report
             precision    recall  f1-score   support

          0       0.80      0.66      0.72     37623
          1       0.37      0.54      0.44     13976

avg / total       0.68      0.63      0.64     51599


Confusion matrix
[[24704 12919]
 [ 6369  7607]]

Profit retained by approving good lenders : $32,235,437
Profit lost due to declining good lenders Profit lost : $27,552,093
Loss  prevented by not lending to defaulters: $-87,648,639
Additional loss due to approving defaulters: $-59,160,556
```

```
Total profit by implementing model : $ $33,171,427

 Top 20 features:
```

| | features | rank |
|---|---|---|
| 0 | credit_length_yrs | 1 |
| 1 | emp_length | 1 |
| 2 | emp_length_mi_flag | 1 |
| 3 | emp_title_mi_flag | 1 |
| 4 | emp_title_mod_Missing | 1 |
| 5 | emp_title_mod_Other | 1 |
| 6 | home_ownership_MORTGAGE | 1 |
| 7 | verification_status_mod_Source_Verified | 1 |
| 8 | home_ownership_RENT | 1 |
| 9 | purpose_credit_card | 1 |
| 10 | purpose_debt_consolidation | 1 |
| 11 | purpose_major_purchase | 1 |
| 12 | purpose_small_business | 1 |
| 13 | term | 1 |
| 14 | verification_status_mod_Not_Verified | 1 |
| 15 | purpose_car | 1 |
| 16 | annual_inc_mod | 1 |
| 17 | verification_status_mod_Verified | 1 |
| 18 | addr_state_CO | 1 |
| 19 | addr_state_CA | 1 |




With barely 20 variables, our tuned model has outperformed prior algorithms by making us a profit of $33MM. Let's observe what may be behind this improvement:

1. With less number of features, it is possible that our algorithm is not overfitting. This occurs when our model performs well on the training data but doesn't fit well on the data it hasn't seen before.
2. However, practically assessing, most of the profits are coming from increased identification of bad loans, thereby saving us close to $87MM in prevented loss.

| Metric | Benchmark Predictor | Logistic Regression (Balanced) | Random Forest(Tuned) | Random Forest (Subset) |
|---|---|---|---|---|
| Accuracy Score | 0.73 | .64 | .65 | 0.626 |
| F-score | 0.36 | 0.43 | 0.41 | 0.44 |
| Precision | 0.49 | 0.37 | 0.38 | 0.37 |
| Recall | 0.28 | 0.51 | 0.45 | 0.54 |
| Profit/Loss | -$7MM | +$27MM | +$23MM | **$33MM** |

**Using SelectKbest:**

As a rudimentary method of feature selection, we can make use of sklearn's SelectKBest () implementation. It takes as a parameter a score function, which must be applicable to a pair (X, y).SelectKBest then simply retains the first k features of X with the highest scores.

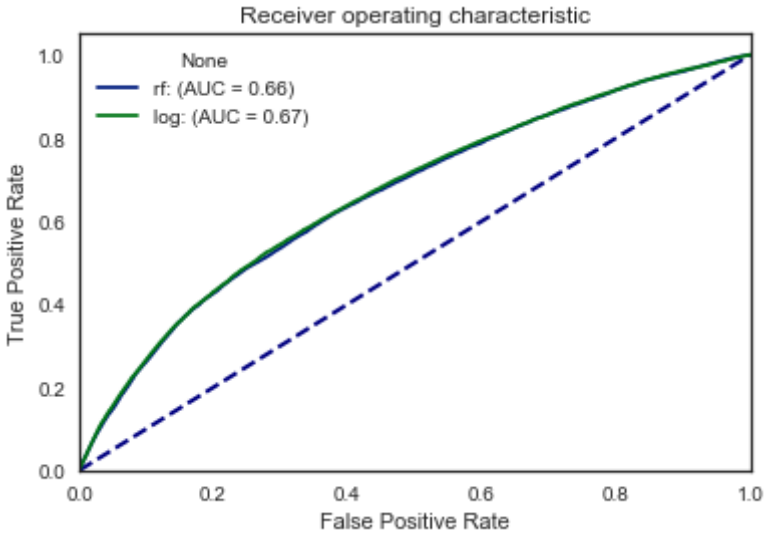
So, for example, if we pass chi2 as a score function, SelectKBest will compute the chi2 statistic between each feature of X and y. A small value will mean the feature is independent of y whereas a large value implies the feature may be related to y and may convey important information.

For our simple check, we will throw k=15 to obtain top 15 features to be used in each model type.

As SelectKBest is relatively, fast, in this instance, we will try to pass these features to all the models to assess if we seen an improvemnt.

```
                                  Feat_names      F_Scores
11                                      term  14747.549157
12     verification_status_mod_Not_Verified   2222.948036
0                           annual_inc_mod   1543.403275
14         verification_status_mod_Verified    767.943015
6                   home_ownership_MORTGAGE    611.697935
7                       home_ownership_RENT    578.547535
8                       purpose_credit_card    551.523857
9                purpose_debt_consolidation    432.753967
13  verification_status_mod_Source_Verified    368.152250
10                   purpose_small_business    307.749055
2                        emp_length_mi_flag    297.829602
3                         emp_title_mi_flag    269.407282
4                     emp_title_mod_Missing    269.407282
1                        credit_length_yrs    269.083634
5                       emp_title_mod_Other    234.304384

/Users/xtl476/anaconda/envs/capstone/lib/python2.7/site-packages/sklear
n/feature_selection/univariate_selection.py:113: UserWarning: Features
[28 74 87] are constant.
  UserWarning)
/Users/xtl476/anaconda/envs/capstone/lib/python2.7/site-packages/sklear
n/feature_selection/univariate_selection.py:114: RuntimeWarning: invali
d value encountered in divide
  f = msb / msw
```

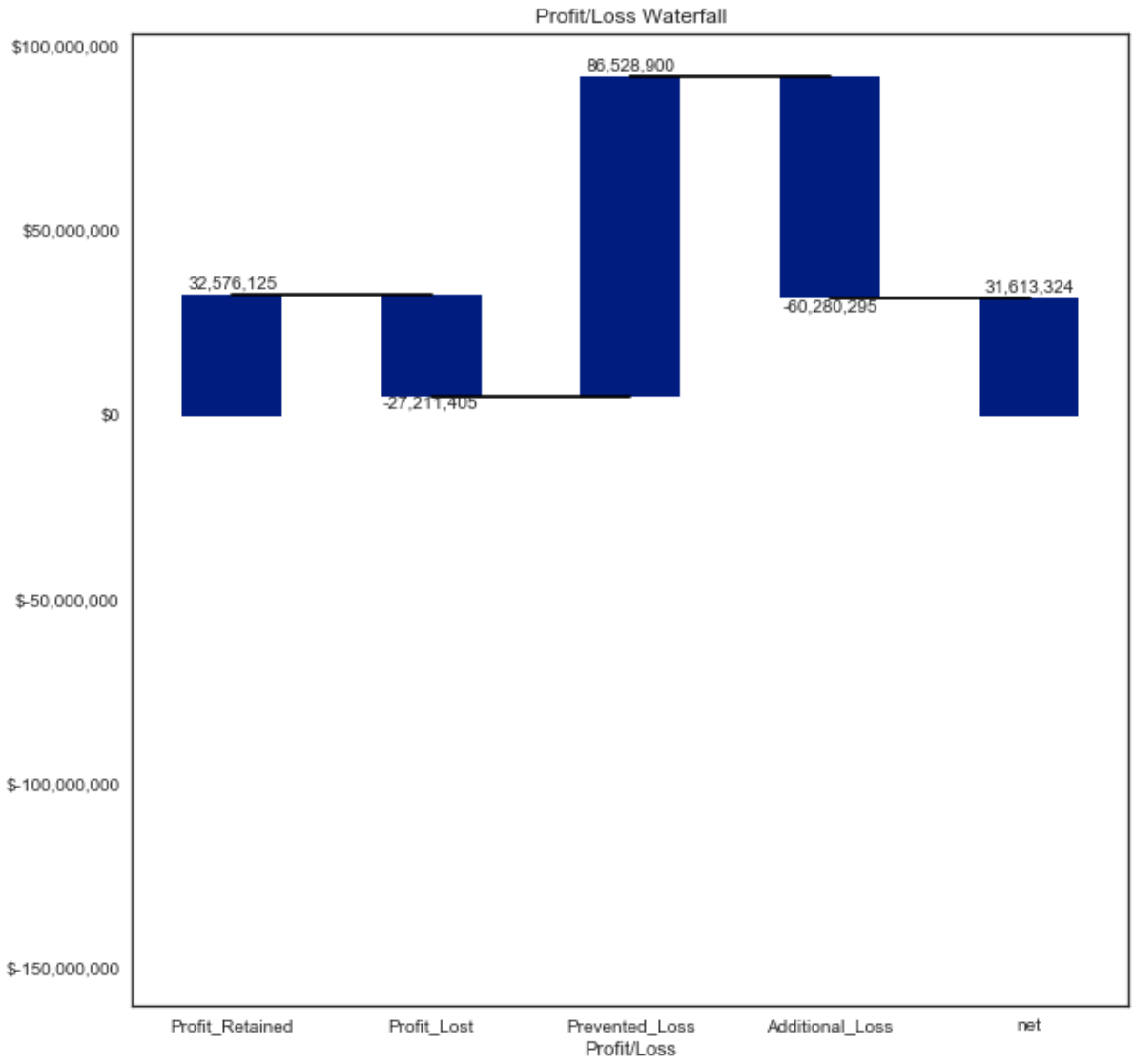
Out[234]:
```
LogisticRegression(C=1.0, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

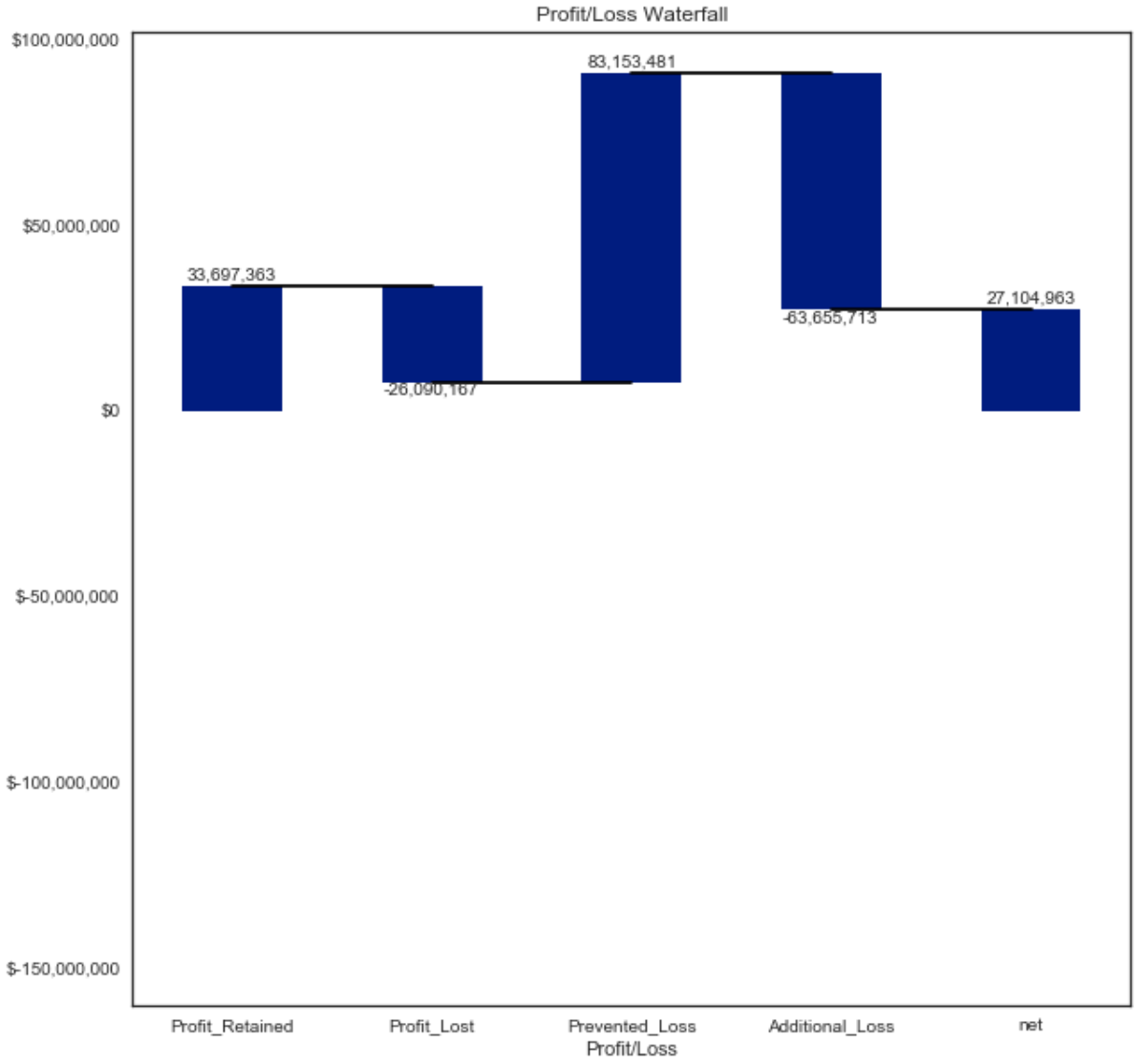



Out[236]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x128097190>
```

```
***Model Performance from for RandomForestClassifier***

Accuracy:0.631

Classification report
             precision    recall  f1-score   support

          0       0.79      0.67      0.73     37623
          1       0.37      0.53      0.44     13976

avg / total       0.68      0.63      0.65     51599


Confusion matrix
[[25148 12475]
 [ 6572  7404]]


***Model Performance from for LogisticRegression***

Accuracy:0.637

Classification report
             precision    recall  f1-score   support

          0       0.79      0.69      0.73     37623
          1       0.37      0.51      0.43     13976

avg / total       0.68      0.64      0.65     51599


Confusion matrix
[[25822 11801]
 [ 6906  7070]]
```

```
***Model Profit from for RandomForestClassifier***

Profit retained by approving good lenders : $32,576,125
Profit lost due to declining good lenders Profit lost : $27,211,405
Loss  prevented by not lending to defaulters: $-86,528,900
Additional loss due to approving defaulters: $-60,280,295
Total profit by implementing model : $ $31,613,324

***Model Profit from for LogisticRegression***

Profit retained by approving good lenders : $33,697,363
Profit lost due to declining good lenders Profit lost : $26,090,167
Loss  prevented by not lending to defaulters: $-83,153,481
Additional loss due to approving defaulters: $-63,655,713
Total profit by implementing model : $ $27,104,963
```


Profit/Loss Waterfall
$100,000,000
$50,000,000
$0
$-50,000,000
$-100,000,000
$-150,000,000
32,576,125
-27,211,405
86,528,900
-60,280,295
31,613,324
Profit_Retained
Profit_Lost
Prevented_Loss
Additional_Loss
net
Profit/Loss




Nothing noteworthy using these features other than we have a better performance from Logistic regression and now it is saving us **$27MM** as earlier but with less complexity, i.e. only 20 features. This leads me to believe that there is potential to improve the linear model and may be we should invest some time to explore the same.

In the next few steps, I will completely focus on the improvement in the Logistic Regression model using the following technique:

- Identifying interaction effects

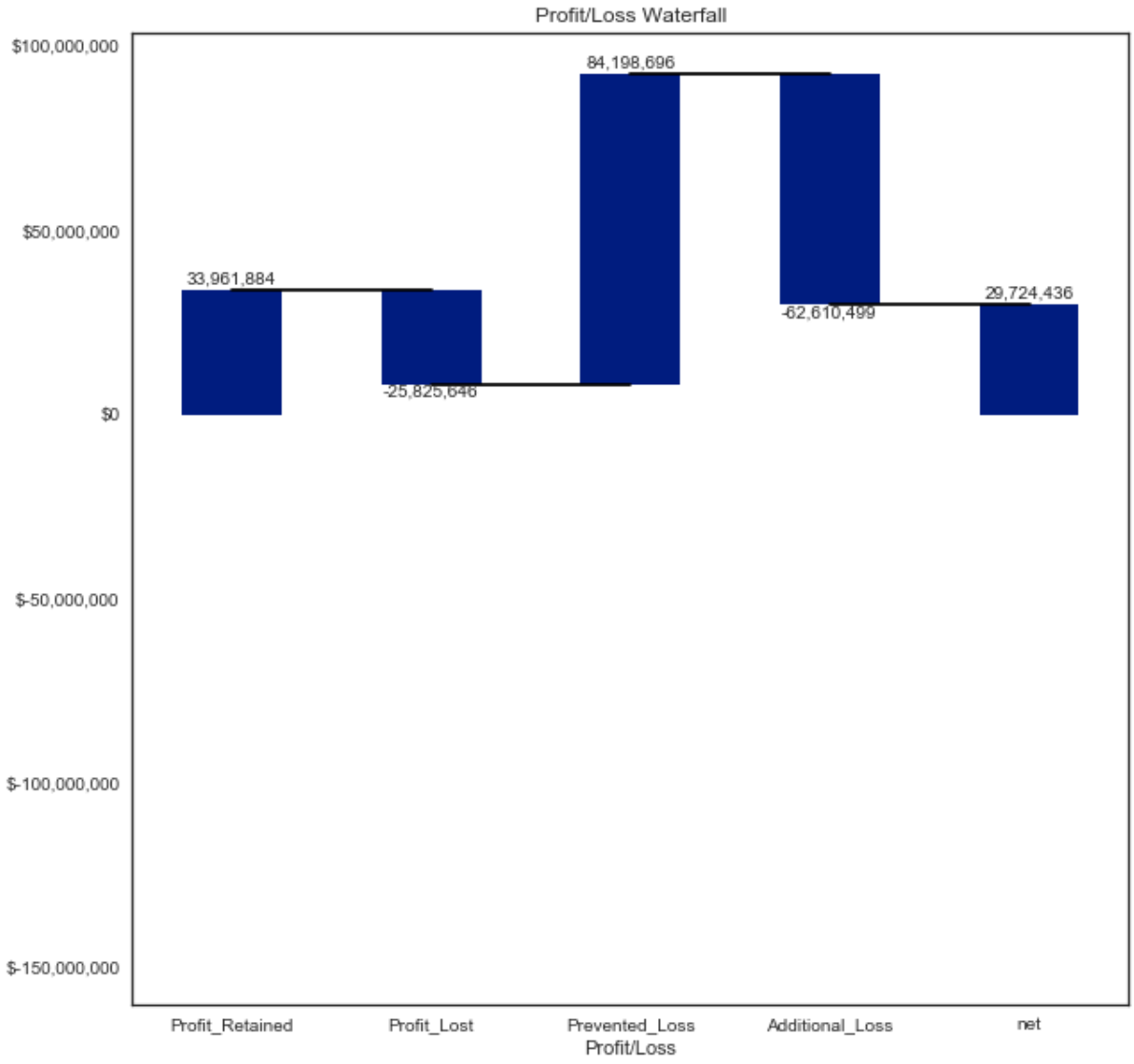
**Identifying interaction effects and using it in a linear model:**

This step is being conducted purely to improve the performance of the linear model (Logistic Regression). Unlike tree based methods, linear models can't identify the feature interactions. Hence, we will throw all two-way interactions among our feature set in the model and see if that improves the performance of our model.

```
Train Score:  0.667845430998
Test Score:  0.663976265287

Classes Predicted:  [0 1]
Logistic Regression Model Score:  0.667845430998

 Confusion Matrix:
[[48644 22343]
 [ 6425  8201]]

 Classification Report:
             precision    recall  f1-score   support

          0       0.88      0.69      0.77     70987
          1       0.27      0.56      0.36     14626

avg / total       0.78      0.66      0.70     85613


Accuracy:0.635

Classification report
             precision    recall  f1-score   support

          0       0.79      0.68      0.73     37623
          1       0.37      0.52      0.44     13976

avg / total       0.68      0.63      0.65     51599


Confusion matrix
[[25453 12170]
 [ 6680  7296]]
```

Out[245]:

| pred | 0 | 1 |
|---|---|---|
| **loan_status** | | |
| **0** | 25453 | 12170 |
| **1** | 6680 | 7296 |




While we are observing a net profit of additional $2MM, it comes at the expense of a more complex model and extended training time. Such factors may influence the decision of management to deploy the models in production and should be taken into consideration.

**Model 3: Gradient Boosting Machines**

Boosting is a popular technique to achieve higher performance but it also comes at the expense of extended times to tune and carefully pruning the trees to avoid overfitting. xgboost package has been identified as a breakthrough but due to issues with it's installation on my Mac machine, I will go with GradientBoostingClassifier in sklearn library.

Prior to using the model, we will need to invest in tuning the hyperparameters. At the same time, we need to address the class imbalance issue. We will address it using the compute_sample_weight method to pass weights to the fit method of this estimator.

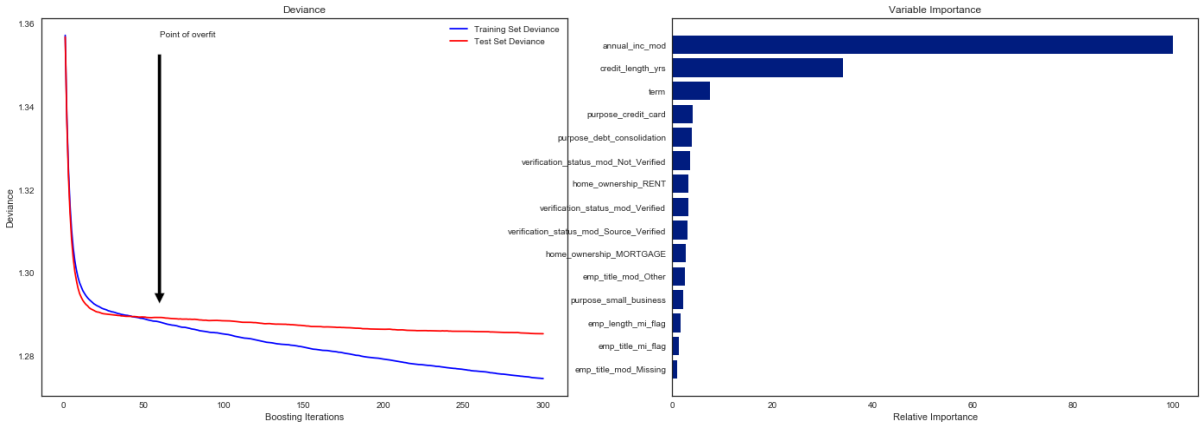
Also, to limit the computation power needed, I will only run this on features returned by SelectKBest method.

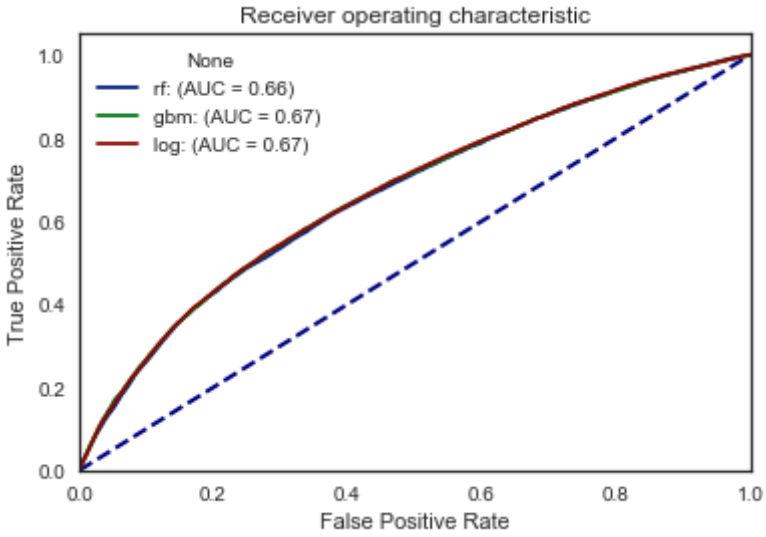
This will allow us to compare the three methods side-by-side.

```
Unoptimized model
------
Accuracy score on testing data: 0.8293
F-score on testing data: 0.0078

Optimized Model
------
Final accuracy score on the testing data: 0.6740
Final F-score on the testing data: 0.3024
```

Let me also showcase the need for tuning and pruning the number of estimators. Here, I will use our model and try to plot the loss that the model tries to minimize to estimate the model (in this case that loss will be deviance) by number of estimators to identify the point of overfitting. At this point, we shall be able to see the improvement on the training data while the test scores tend to get worse.

```
GBT_log_loss: 11.603
```

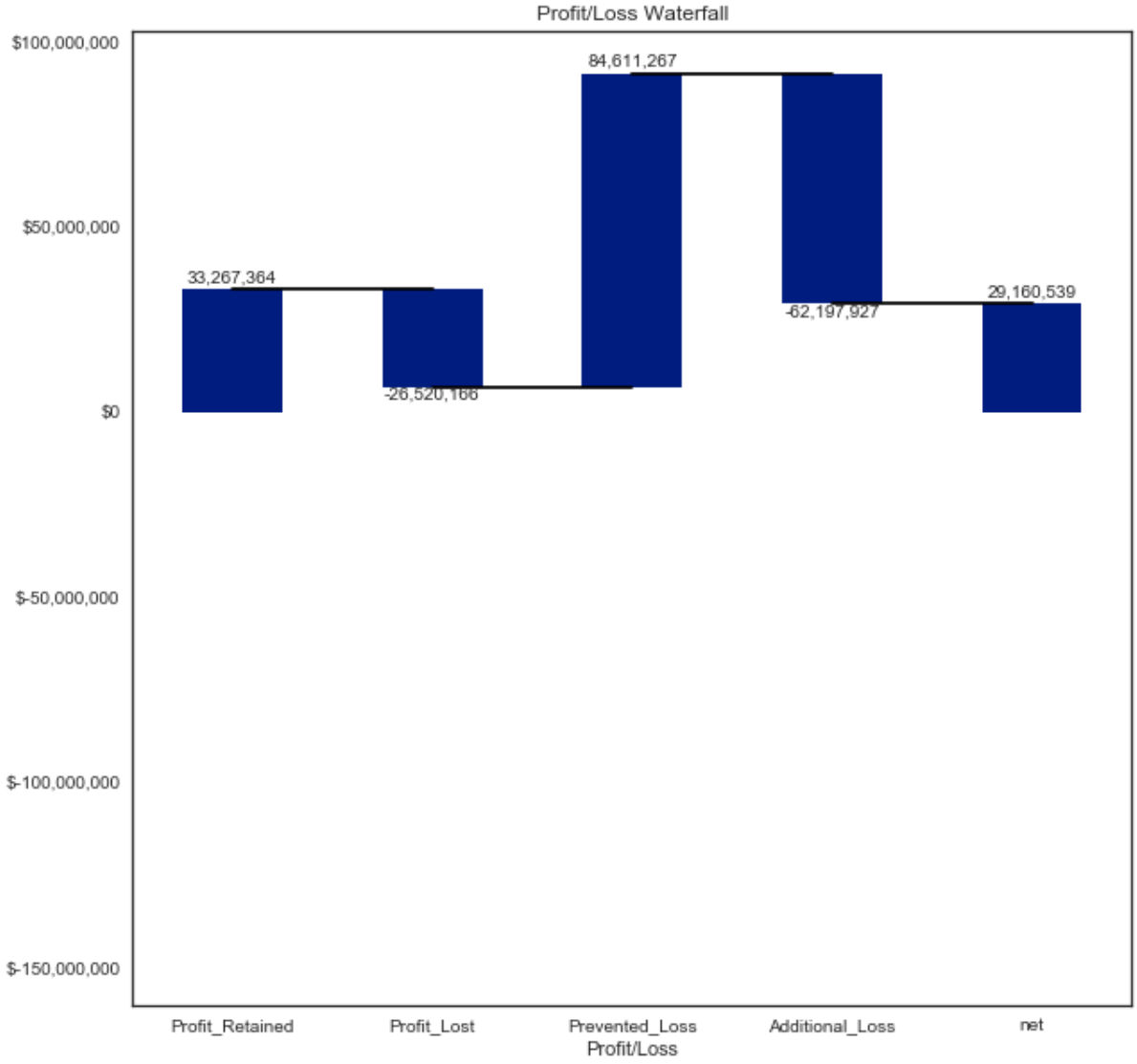





Out[272]: <matplotlib.axes._subplots.AxesSubplot at 0x11c1888d0>

```
***Model Performance from for GradientBoostingClassifier***

Accuracy:0.630

Classification report
             precision    recall  f1-score   support

          0       0.79      0.67      0.72     37623
          1       0.37      0.52      0.43     13976

avg / total       0.68      0.63      0.65     51599


Confusion matrix
[[25187 12436]
 [ 6677  7299]]
```

```
***Model Profit from for GradientBoostingClassifier***

Profit retained by approving good lenders : $33,267,364
Profit lost due to declining good lenders Profit lost : $26,520,166
Loss  prevented by not lending to defaulters: $-84,611,267
Additional loss due to approving defaulters: $-62,197,927
Total profit by implementing model : $ $29,160,539
```




# IV. Results

*(approx. 2-3 pages)*

## Results on Out of Time data:

| Metric | Benchmark Predictor | Logistic Regression (Balanced) | Random Forest(Tuned) | Random Forest (Subset) | GBM |
|---|---|---|---|---|---|
| Accuracy Score | 0.73 | .64 | .65 | 0.626 | 0.63 |
| F-score | 0.36 | 0.43 | 0.41 | 0.44 | 0.43 |
| Precision | 0.49 | 0.37 | 0.38 | 0.37 | 0.37 |
| Recall | 0.28 | 0.51 | 0.45 | 0.54 | 0.52 |
| Profit/Loss | -$7MM | +$27MM | +$23MM | **$33MM** | $29MM |

As discussed in the prior section, the results seem in favor of Random Forest due to following reasons:

1. Faster to tune comapred to GBM
2. Captures non-linerity in the data as it's a tree based method
3. Being an ensemble approach,it's fairly high performing when it comes to classifying our outcome.

## Model Evaluation and Validation

In this section, the final model and any supporting qualities should be evaluated in detail. It should be clear how the final model was derived and why this model was chosen. In addition, some type of analysis should be used to validate the robustness of this model and its solution, such as manipulating the input data or environment to see how the model's solution is affected (this is called sensitivity analysis). Questions to ask yourself when writing this section:

- *Is the final model reasonable and aligning with solution expectations? Are the final parameters of the model appropriate?*
- *Has the final model been tested with various inputs to evaluate whether the model generalizes well to unseen data?*
- *Is the model robust enough for the problem? Do small perturbations (changes) in training data or the input space greatly affect the results?*
- *Can results found from the model be trusted?*

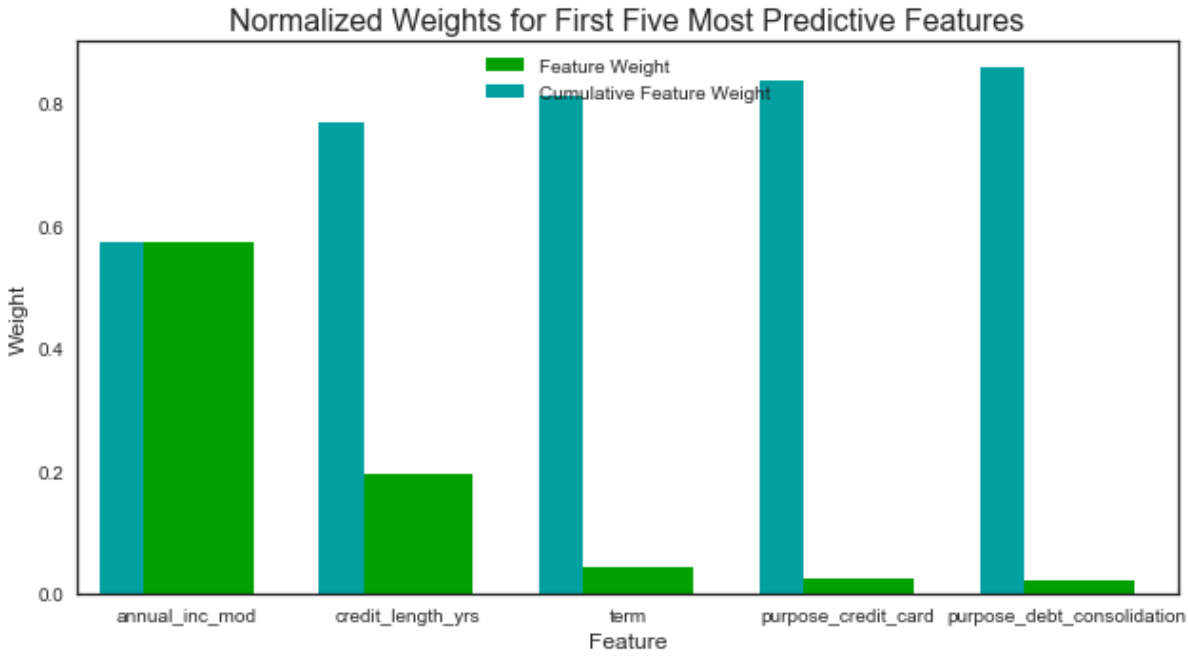
While we have conducted thorough validation of our model(s) at each step by not only considering test dataset but also a out of time sample, we can go further to ascertain whether our model makes sense by digging deeper into the feature set our model used to make judegement.

Also, recall that in exploratory data phase, we observered certain characteristics which were associated with the riskier behavior of a borrower.

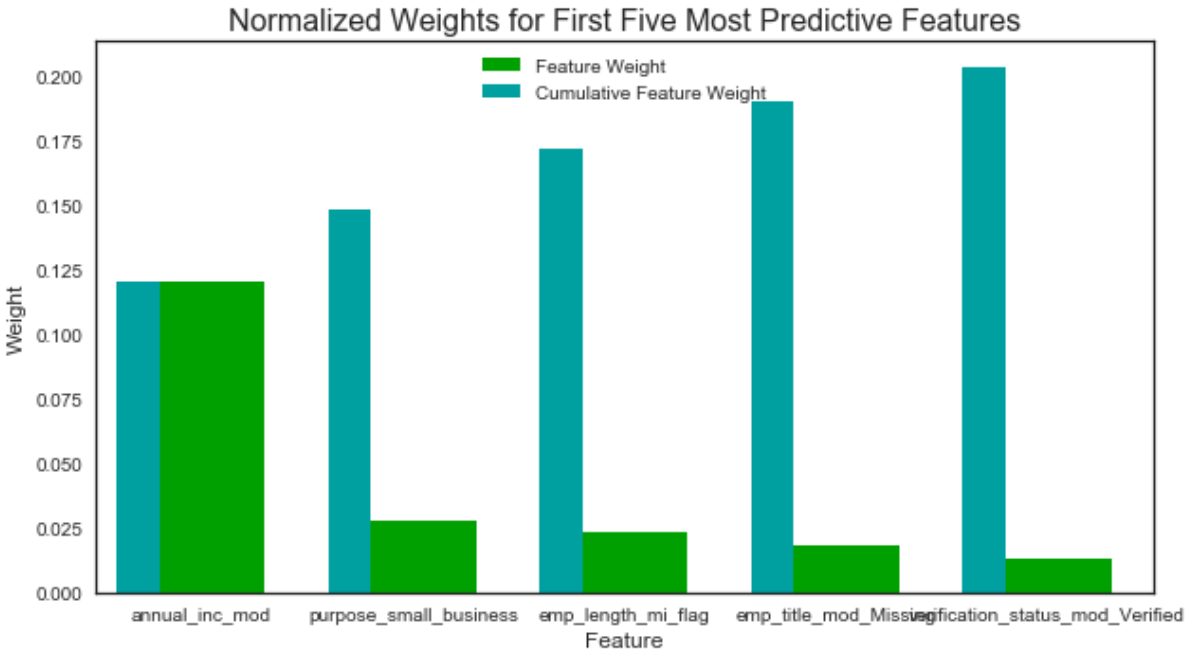
Here can we can build upon that idea and assess if intuitively our model makes sense. In order to conduct this excerise, I will use variable importances from the Random Forest (subset) and GBM that we have trained earlier and explore if the same features show up as important that we earlier observed.

However, in order to better understand the relationship between the risk and features, I will make use of a stepwise selection to select features using a function I found here (https://datascience.stackexchange.com/questions/937/does-scikit-learn-have-forward-selection-stepwise-regression-algorithm). This will help us confirm the nature and effect of the relationship.

**Feature importances from the GBM model:**




**Feature importances from the Random Forest model:**




It is rather comforting to know that the top feature (annual income) is common between the two methodologies. Next, let's retrieve feature coefficients from a stepwise selected feature set after fittiing a linear model (Logistic Regression)

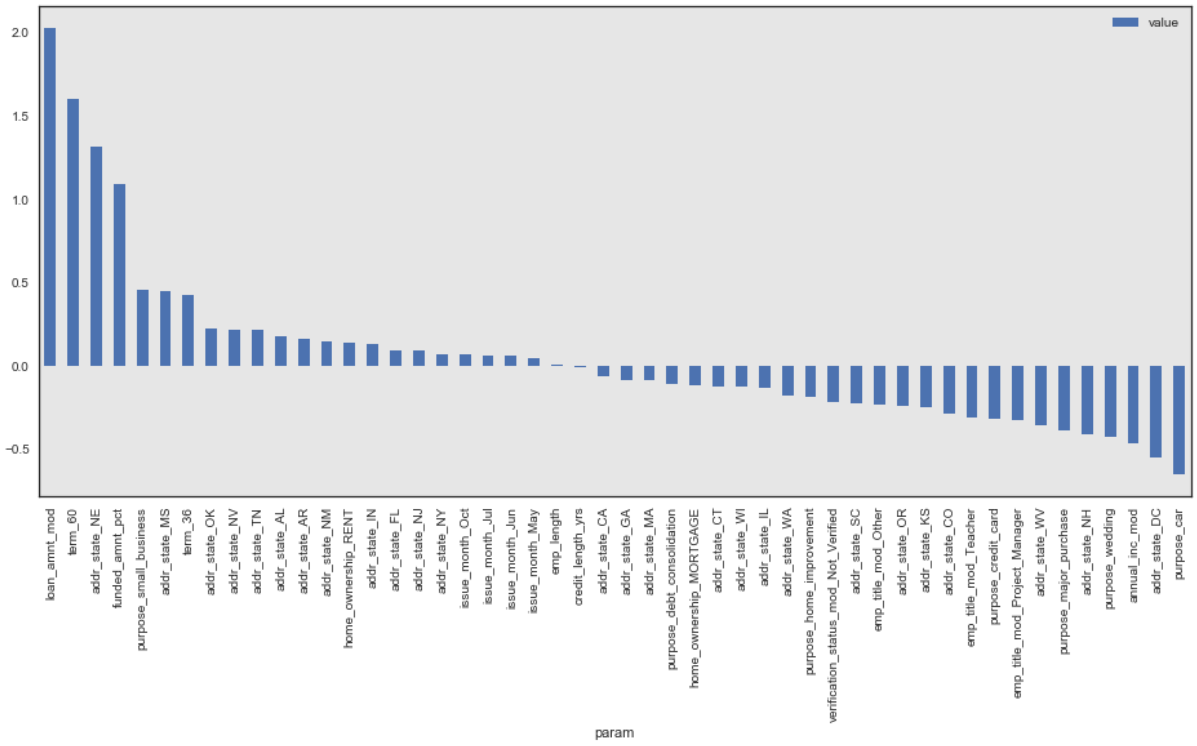
```
/Users/xtl476/anaconda/envs/capstone/lib/python2.7/site-packages/statsm
odels/compat/pandas.py:56: FutureWarning: The pandas.core.datetools mod
ule is deprecated and will be removed in a future version. Please use t
he pandas.tseries module instead.
  from pandas.core import datetools
/Users/xtl476/anaconda/envs/capstone/lib/python2.7/site-packages/statsm
odels/base/model.py:1036: RuntimeWarning: invalid value encountered in
divide
  return self.params / self.bse
/Users/xtl476/anaconda/envs/capstone/lib/python2.7/site-packages/scipy/
stats/_distn_infrastructure.py:879: RuntimeWarning: invalid value encou
ntered in greater
  return (self.a < x) & (x < self.b)
/Users/xtl476/anaconda/envs/capstone/lib/python2.7/site-packages/scipy/
stats/_distn_infrastructure.py:879: RuntimeWarning: invalid value encou
ntered in less
  return (self.a < x) & (x < self.b)
/Users/xtl476/anaconda/envs/capstone/lib/python2.7/site-packages/scipy/
stats/_distn_infrastructure.py:1818: RuntimeWarning: invalid value enco
untered in less_equal
  cond2 = cond0 & (x <= self.a)
/Users/xtl476/anaconda/envs/capstone/lib/python2.7/site-packages/ipyker
nel_launcher.py:34: FutureWarning: 'argmin' is deprecated. Use 'idxmin'
instead. The behavior of 'argmin' will be corrected to return the posit
ional minimum in the future. Use 'series.values.argmin' to get the posi
tion of the minimum now.
```

```
Add  verification_status_mod_Not_Verified with p-value 0.0
Add  annual_inc_mod                  with p-value 0.0
Add  term_60                         with p-value 0.0
Add  installment_mod                 with p-value 0.0
Add  loan_amnt_mod                   with p-value 0.0
Add  term_36                         with p-value 0.0
Add  home_ownership_RENT             with p-value 6.8048e-135
Add  purpose_small_business          with p-value 1.1872e-77
Add  purpose_credit_card             with p-value 4.55571e-48
Add  purpose_car                     with p-value 2.31936e-27
Add  emp_title_mod_Other             with p-value 3.33327e-17
Add  addr_state_CO                   with p-value 8.84511e-17
Add  credit_length_yrs               with p-value 8.57513e-16
Add  purpose_major_purchase          with p-value 1.56804e-13
Add  home_ownership_MORTGAGE         with p-value 5.37239e-11
Add  funded_amnt_pct                 with p-value 3.17342e-10
Add  addr_state_TN                   with p-value 1.64735e-08
Add  addr_state_FL                   with p-value 7.4117e-09
Add  addr_state_NY                   with p-value 3.90638e-10
Add  addr_state_NV                   with p-value 5.04749e-09
Add  emp_title_mod_Teacher           with p-value 2.89966e-08
Add  purpose_wedding                 with p-value 6.862e-08
Add  addr_state_MS                   with p-value 7.55639e-08
Add  addr_state_NJ                   with p-value 7.23573e-08
Add  addr_state_AL                   with p-value 6.83563e-08
Add  addr_state_OK                   with p-value 2.07281e-07
Add  addr_state_DC                   with p-value 1.91193e-06
Add  addr_state_IN                   with p-value 3.00194e-06
Add  addr_state_NH                   with p-value 6.40374e-06
Add  emp_length                      with p-value 2.23622e-05
Add  addr_state_OR                   with p-value 2.28342e-05
Add  purpose_other                   with p-value 3.59524e-05
Add  addr_state_WV                   with p-value 3.55999e-05
Add  addr_state_WA                   with p-value 4.1849e-05
Add  addr_state_KS                   with p-value 0.000107607
Add  purpose_home_improvement        with p-value 0.000123479
Add  purpose_debt_consolidation      with p-value 4.52955e-05
```

```
/Users/xtl476/anaconda/envs/capstone/lib/python2.7/site-packages/ipyker
nel_launcher.py:47: FutureWarning: 'argmax' is deprecated. Use 'idxmax'
instead. The behavior of 'argmax' will be corrected to return the posit
ional maximum in the future. Use 'series.values.argmax' to get the posi
tion of the maximum now.
```

```
Drop purpose_other                  with p-value 0.427973
Add  addr_state_SC                  with p-value 0.00016113
Add  addr_state_AR                  with p-value 0.000522504
Add  emp_title_mod_Project_Manager  with p-value 0.000567101
Add  issue_month_Oct                with p-value 0.000543383
Add  issue_month_Jul                with p-value 0.000338636
Add  addr_state_IL                  with p-value 0.000816918
Add  addr_state_CA                  with p-value 0.000303289
Add  issue_month_Jun                with p-value 0.00266596
Add  addr_state_NE                  with p-value 0.00466894
Add  addr_state_NM                  with p-value 0.00548277
Add  addr_state_WI                  with p-value 0.00682968
Add  addr_state_CT                  with p-value 0.00837117
Add  addr_state_GA                  with p-value 0.00825648
Add  addr_state_MA                  with p-value 0.0077028
Add  issue_month_May                with p-value 0.00831955
resulting features:
['verification_status_mod_Not_Verified', 'annual_inc_mod', 'term_60',
'installment_mod', 'loan_amnt_mod', 'term_36', 'home_ownership_RENT',
'purpose_small_business', 'purpose_credit_card', 'purpose_car', 'emp_ti
tle_mod_Other', 'addr_state_CO', 'credit_length_yrs', 'purpose_major_pu
rchase', 'home_ownership_MORTGAGE', 'funded_amnt_pct', 'addr_state_TN',
'addr_state_FL', 'addr_state_NY', 'addr_state_NV', 'emp_title_mod_Teach
er', 'purpose_wedding', 'addr_state_MS', 'addr_state_NJ', 'addr_state_A
L', 'addr_state_OK', 'addr_state_DC', 'addr_state_IN', 'addr_state_NH',
'emp_length', 'addr_state_OR', 'addr_state_WV', 'addr_state_WA', 'addr_
state_KS', 'purpose_home_improvement', 'purpose_debt_consolidation', 'a
ddr_state_SC', 'addr_state_AR', 'emp_title_mod_Project_Manager', u'issu
e_month_Oct', u'issue_month_Jul', 'addr_state_IL', 'addr_state_CA', u'i
ssue_month_Jun', 'addr_state_NE', 'addr_state_NM', 'addr_state_WI', 'ad
dr_state_CT', 'addr_state_GA', 'addr_state_MA', u'issue_month_May']
```

Out[246]:
```
LogisticRegression(C=1.0, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)

Train Score:  0.674887068934
Test Score:  0.675505282358
```



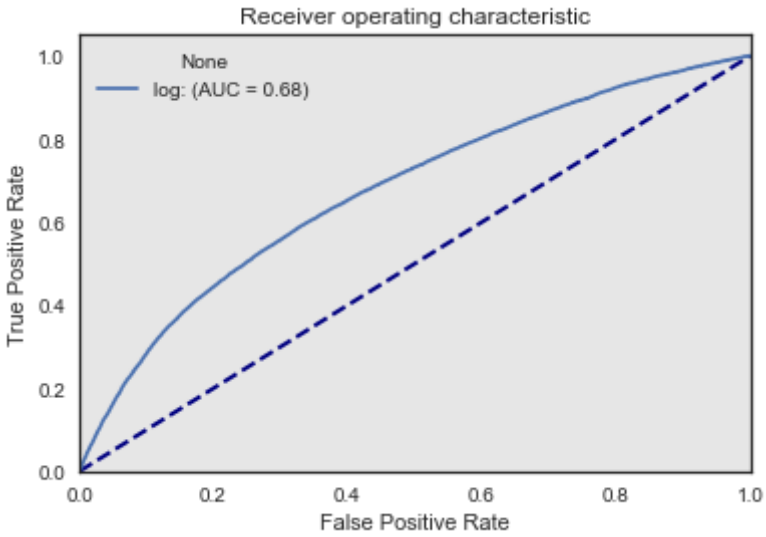

**Comparing feature importances with feature coefficeints obtained from Logistic Regression:**

We again see a lot of common features across varios methodologies but here we are able to also assess the nature of the relationship:

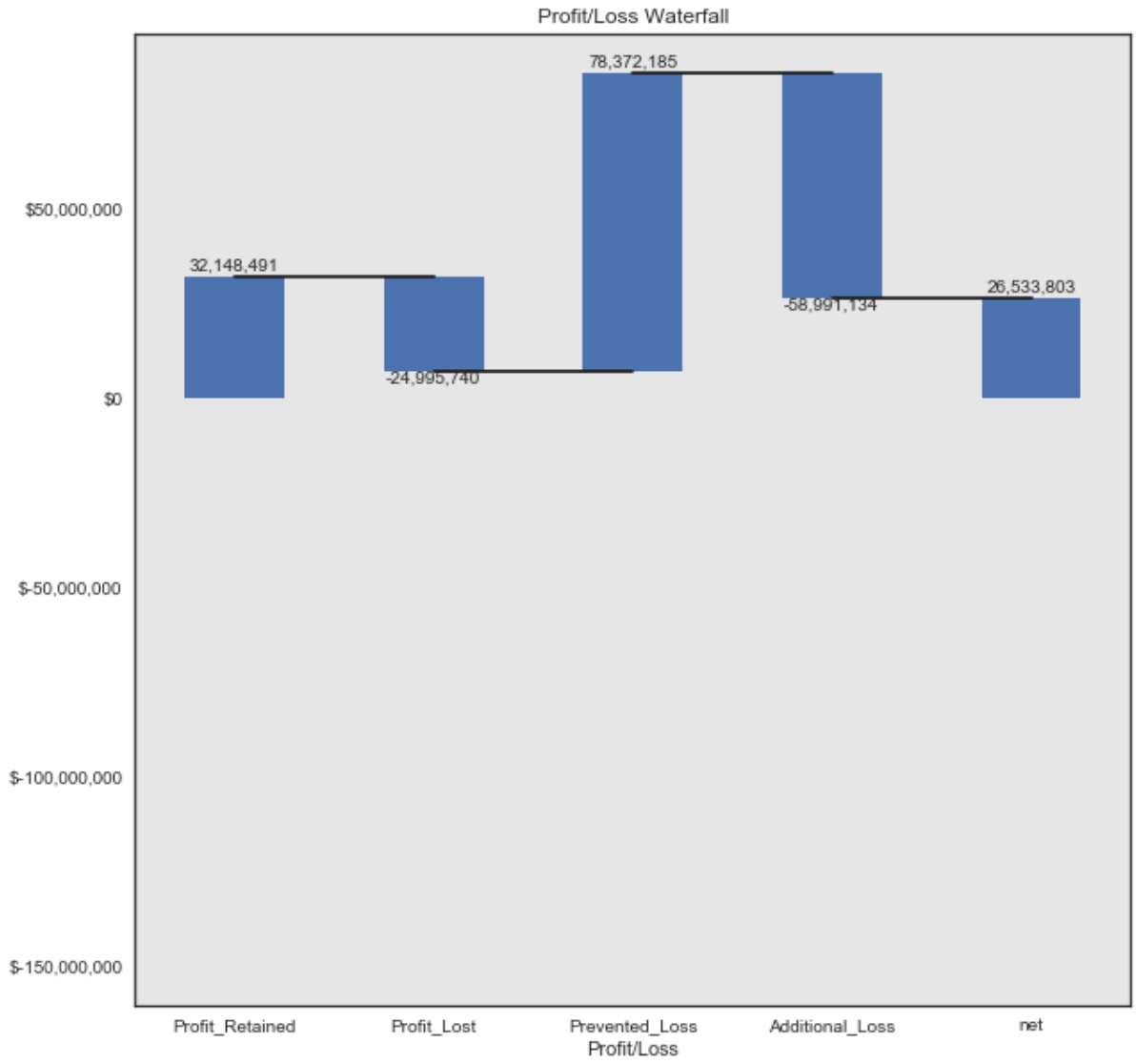
1. We observed that higher term (60 months) was several folds risk and the positive coefficient here confirms the same.
2. Small Business as a purpose has shown on two of the above methodologies which is another observation we made during exploratory phase.
3. Higher income is associated with low risk and this would make intuitive sense.
4. Renters tend to be higher risk than Mortgage holders is another concept we can verify using the above chart.

Given our validation combined with relationship of our features with the target, we can assess that our model validates well along with making intuitive sense.

```
WARNING:matplotlib.legend:No handles with labels found to put in legen
d.
```




```
Accuracy:0.639

Classification report
             precision    recall  f1-score   support

          0       0.79      0.69      0.74     35457
          1       0.37      0.50      0.43     12743

avg / total       0.68      0.64      0.65     48200


Confusion matrix
[[24369 11088]
 [ 6308  6435]]

Profit retained by approving good lenders : $32,148,491
Profit lost due to declining good lenders Profit lost : $24,995,740
Loss  prevented by not lending to defaulters: $-78,372,185
Additional loss due to approving defaulters: $-58,991,134
Total profit by implementing model : $ $26,533,803
```




```
Classes Predicted:  [0 1]
Logistic Regression Model Score:  0.674887068934

 Confusion Matrix:
[[47481 20400]
 [ 6107  7699]]

 Classification Report:
             precision    recall  f1-score   support

          0       0.89      0.70      0.78     67881
          1       0.27      0.56      0.37     13806

avg / total       0.78      0.68      0.71     81687
```