Prediction of Bad Loans for Peer-to-Peer Lending

1. Definition

1.1. Domain Background

Peer-to-peer lending (P2P lending) is the practice of lending money to individuals or businesses through the any online platform which matches lenders with borrowers (https://en.wikipedia.org/wiki/Peer-to-peer_lending). P2P lending has been around for quite a while. While their volume is substantially low compared to standard bank-based loans, P2P lending has been growing at a very fast pace. From 2012 to 2016 it rose 15 times to $77 billion in 2016. The expectation in US is that this market will grow to reach $150 billion by 2026 (https://www.debt.org/credit/solutions/peer-lending/). Some of the major players in this field are Lending Club (https://www.lendingclub.com/), Prosper (https://www.prosper.com/) and SoFi (https://www.sofi.com/).

These loans require substantially less paper work, and the interest rates are lower. The lenders also benefit as the loan yields higher returns than traditional bank or bonds. It's a win-win situation for both the borrower and the lender.

I personally got involved in the Loans and lending when I was searching for our first house last year. And more so in the last month, when I was applying for a car loan. For my car loan, I applied to a P2P lending company also. I got the same rate as big national banks and hence, I didn't go through the P2P lending. But my curiosity didn't die as to how do financial institutions judge your ability to repay a loan. This project is partly to cater my curiosity and partly for the fulfillment of the Udacity Nanodegree.

1.2. Problem Statement

When it comes to money lending business there is no such thing as risk-free lending. In the US, P2P lending is treated as investment and hence repayment in case of a default is not guaranteed by the Federal Govt. Despite the credit checks and other kinds of checks, the default rate for such loans has been high. In 2014, Lending Club's default rate was 8.7% and Prosper's was 3.6% (https://investorjunkie.com/9328/lending-club-vs-prosper/). In the first quarter of 2017, in a survey, 17% of US consumers said they were likely to default on a loan payment in the next year (http://www.financialexpress.com/economy/bad-loans-problem-in-us-trumps-america-is-facing-a-13-trillion-consumer-debt-hangover-a-new-record-high/704363/). Hence, the problem of loan defaulting is still very high, and it puts the individual lenders at risk of losing their money.

In terms of machine learning jargon this is a classification problem. The loans are to be classified as good loans and bad loans, where bad loan means it will be defaulted. In addition, I intend to do regression analysis to predict the interest rate of good loans.

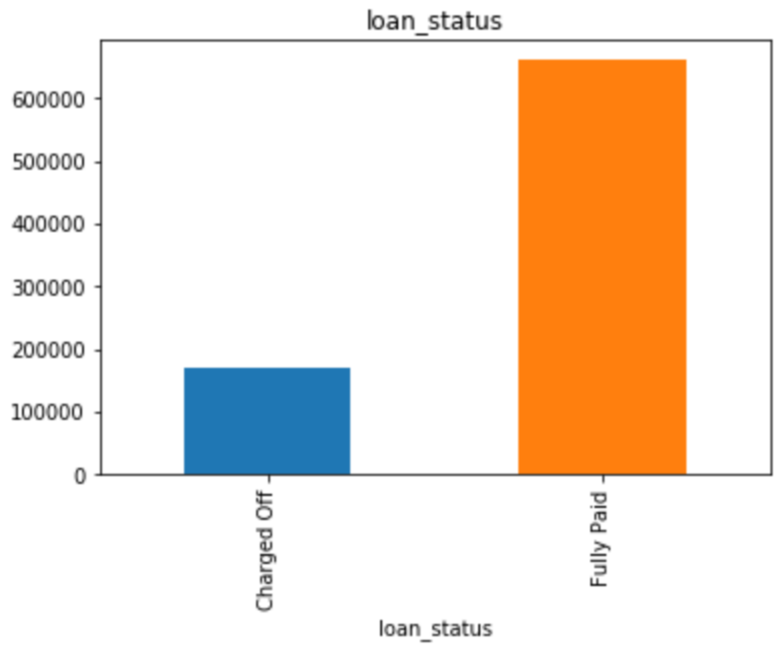
1.3. Datasets and Inputs

Lending club has shared their loan data from the year 2012 to 2017 at this webpage: https://www.lendingclub.com/info/download-data.action. For each loan, it gives the financial, employment, credit, demographic, bankruptcy and other decision-making information in a CSV format. https://resources.lendingclub.com/LCDataDictionary.xlsx gives the list of all the data attributes used. It then gives the interest rate which the loan was granted. And it also mentions if the loan was defaulted. In addition to the complete list of loans granted it has another set of CSV files which lists all the loans denied.

This is data is almost like gold for any P2P lending business. This data can be used for predicting loan interest rate by running machine learning training on the existing data. Similarly, it can be used to predict if a loan will result in default or not.

It must be noted that the data is not fully curated and clean. There are a lot of missing entries in the data. Appropriate methods would have to be used to fill those entries. The data is either numbers or text based on what kind of attribute is being described.

For predicting the loan as good or bad, the target variable to be trained for is “loan_status”. It can take the following values: “Issued”, “Current”, “Fully Paid” and “Charged Off”. “Charged Off” means the loan was defaulted. The distribution of loan_status is show below. As expected, the number of loans defaulted is a relatively low number. The data is not balanced and only 20.5% loans end up in default.



1.4. Evaluation Metrics

The original data is divided into three parts – Train, Validate and Test data. The model is trained on the training set. Then we run the trained weights on the validation set. The train+validation is run multiple times until the validation accuracy over the validation set is remaining a constant. That is when it means that the model has learnt the best weights and there is no overfitting. At this point those weights are tried on the test dataset. Since, the model has never seen test dataset, the test data works like new real-world data for the algorithm.

In this case, I plan to use 60% of the data for training, 20% for validation and the rest of the 20% for training. Not that, I will use cross-validation method where the train+validation set (a total of 80%) is used for both training and validation. There is no separate 20% of data separated for validation. Instead four different chunks of 20% of data are used for validation so that each data in the 80% is used for both training and validation.

1.4.1. Classifier evaluation

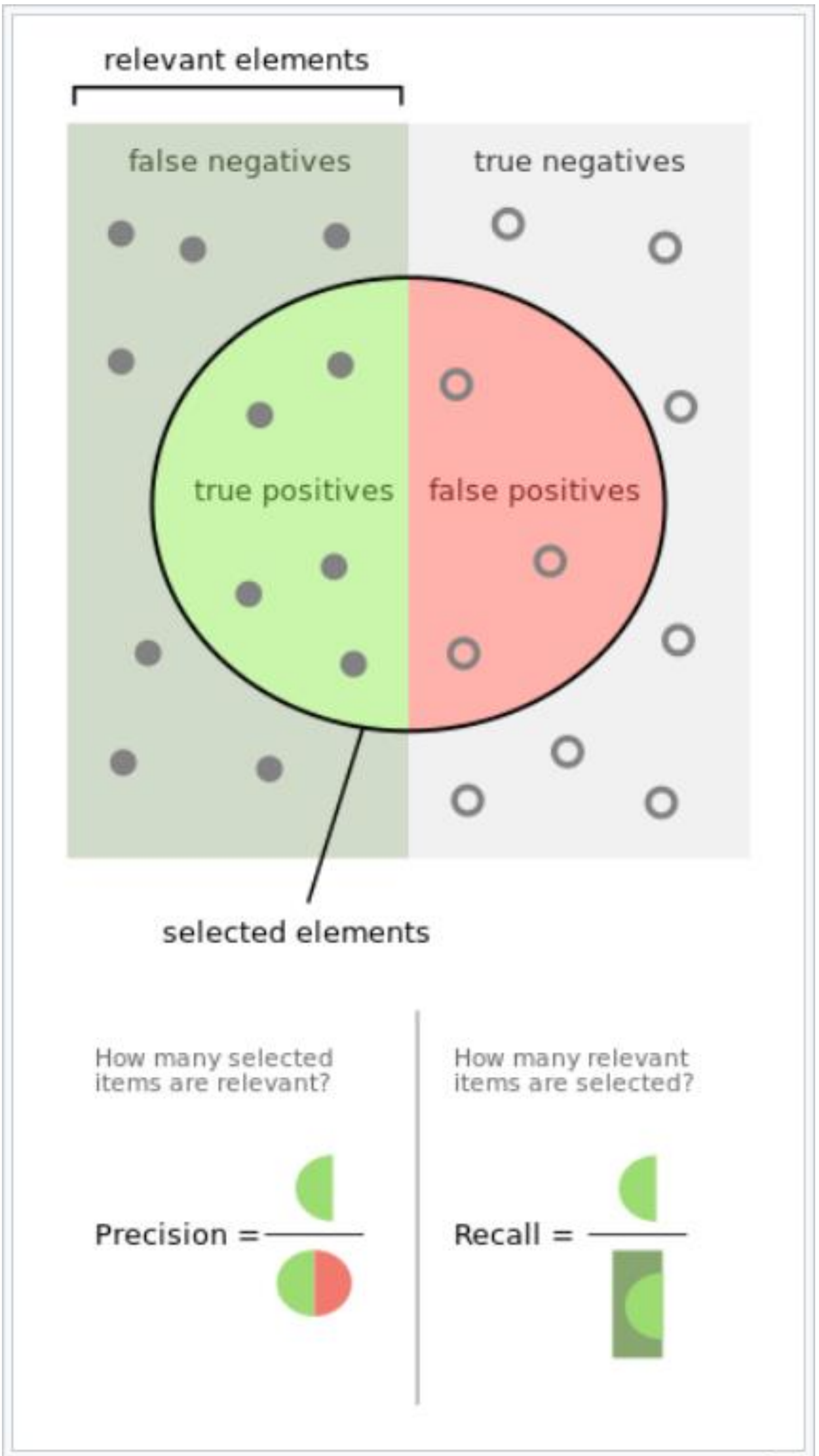
The metrics I will use for measuring the goodness of my model are the accuracy score and F1-scores. F1-score is a measure for the test's accuracy. It considers the precision and the recall.

Precision is the measure of how many of the selected items are relevant. In this case, if my predictor predicts 1000 bad loans, precision of is the measure of how many are bad loans in that 1000.

And Recall is the measure of how many relevant items are selected. In this case, if there were total 5000 loans, and there were 1200 bad loans, recall is the measure of how many of the 1200 did the predictor predict correctly.

The following snippet from Wikipedia page (https://en.wikipedia.org/wiki/Precision_and_recall) shows it pictorially.

Again, borrowing from Wikipedia (https://en.wikipedia.org/wiki/F1_score), in its simplest form, the F1-score is the harmonic mean of the Precision and Recall.



Precision and recall

Again, borrowing from Wikipedia (https://en.wikipedia.org/wiki/F1_score), in its simplest form, the F1-score is the harmonic mean of the Precision and Recall.

$$F_1 = \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

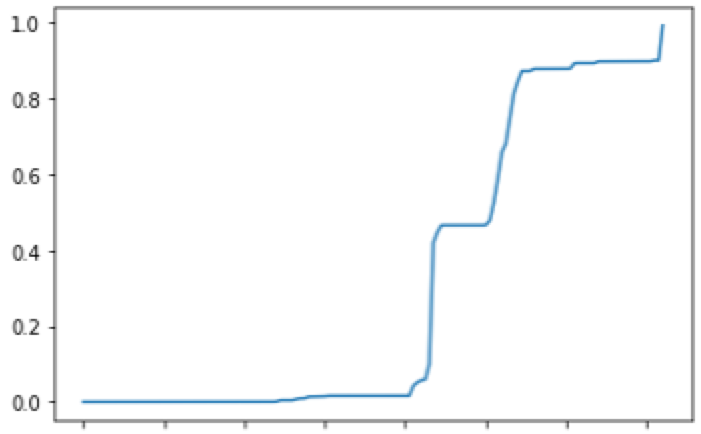
Referring to the above Pie-chart, accuracy is the fraction of true-positives out of the whole dataset.

It is desirable to have a high accuracy score and a high F1-score.

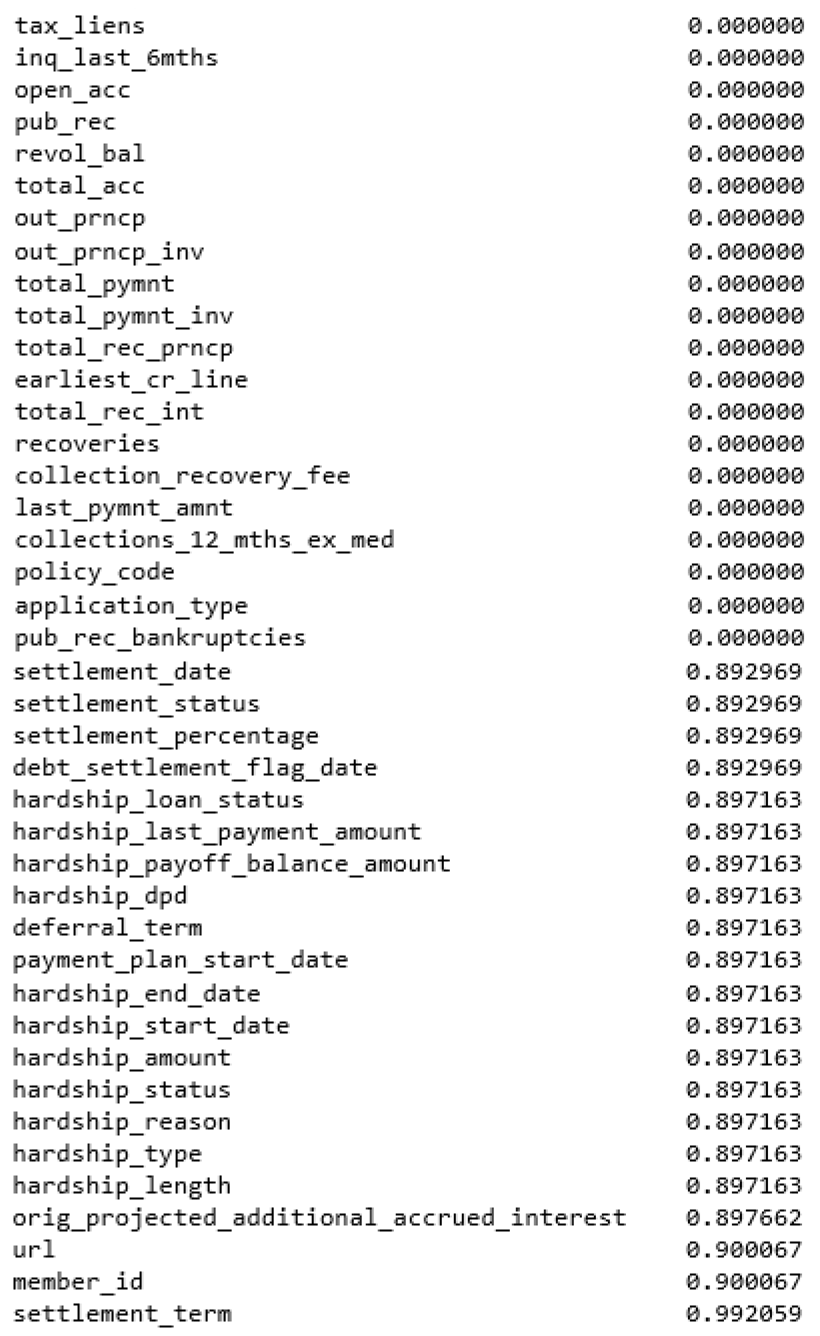
2. Data Cleanup

2.1. Remove null

The loan data is not at all clean. There is a good percentage of data, which is null (empty). The following graph plots the fraction of entries that are null in each feature. The x-axis represents the feature names sorted in order of number of nulls. One can notice that close to 50% of features have significant number of nulls.



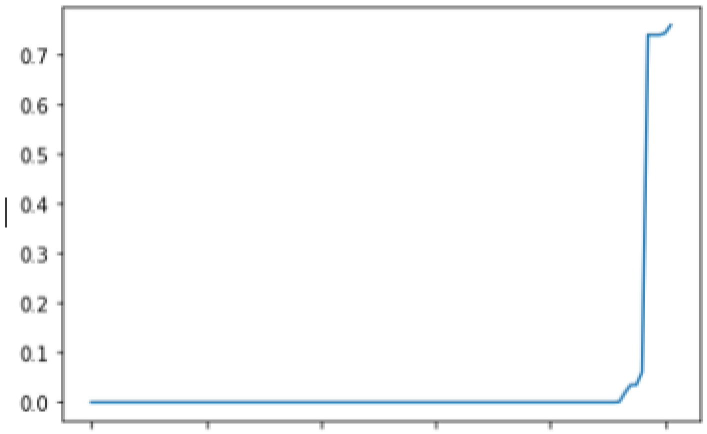
The following shows some example features and the fraction of entries that are null for each.



```
tax_liens                                      0.000000
inq_last_6mths                                 0.000000
open_acc                                       0.000000
pub_rec                                        0.000000
revol_bal                                      0.000000
total_acc                                      0.000000
out_prncp                                      0.000000
out_prncp_inv                                  0.000000
total_pymnt                                    0.000000
total_pymnt_inv                                0.000000
total_rec_prncp                                0.000000
earliest_cr_line                               0.000000
total_rec_int                                  0.000000
recoveries                                     0.000000
collection_recovery_fee                        0.000000
last_pymnt_amnt                                0.000000
collections_12_mths_ex_med                     0.000000
policy_code                                    0.000000
application_type                               0.000000
pub_rec_bankruptcies                           0.000000
settlement_date                                0.892969
settlement_status                              0.892969
settlement_percentage                          0.892969
debt_settlement_flag_date                      0.892969
hardship_loan_status                           0.897163
hardship_last_payment_amount                   0.897163
hardship_payoff_balance_amount                 0.897163
hardship_dpd                                   0.897163
deferral_term                                  0.897163
payment_plan_start_date                        0.897163
hardship_end_date                              0.897163
hardship_start_date                            0.897163
hardship_amount                                0.897163
hardship_status                                0.897163
hardship_reason                                0.897163
hardship_type                                  0.897163
hardship_length                                0.897163
orig_projected_additional_accrued_interest     0.897662
url                                            0.900067
member_id                                      0.900067
settlement_term                                0.992059
```

In the above example, “settlement_term” has 99% or data as null and practically that feature is useless. Luckily for us, “settlement_term” is something that is used only if the loan defaults. And that means this information is not available during the loan process.

After removing the features described in sections 2.2, 2.3, 2.4 and 2.7, the plot of nulls looks like the following.



This is good news as most features have all non-null values. The following are the only null values left.

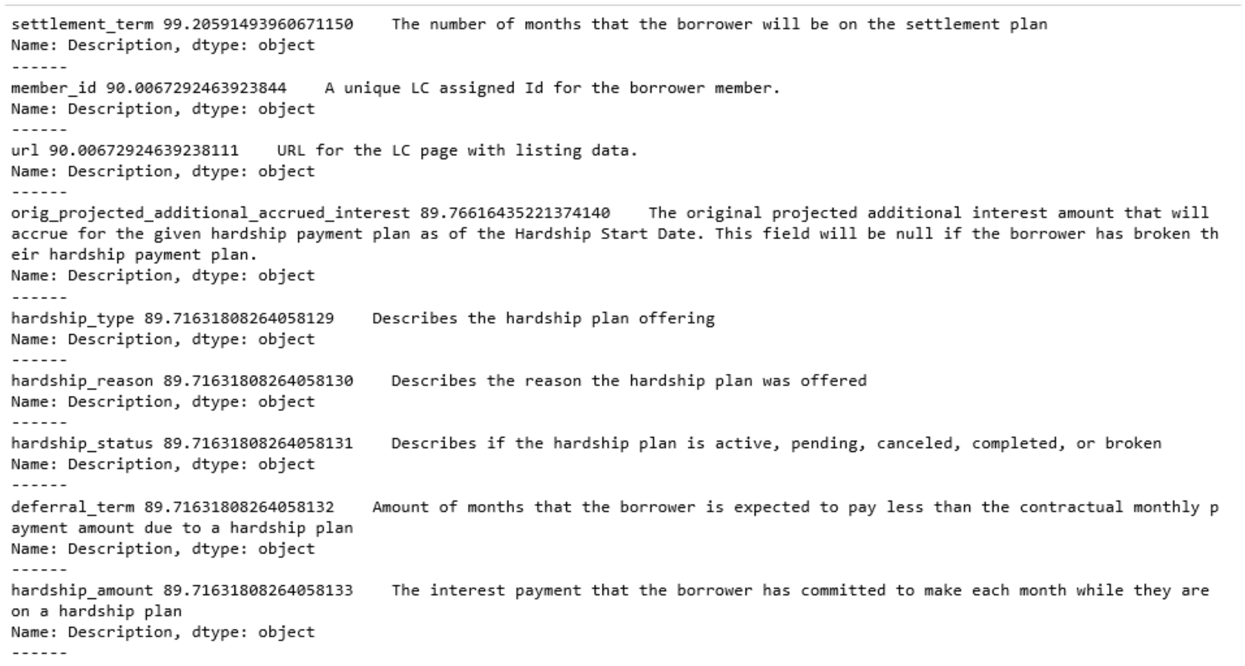
```
il_util 75.87354992920147
28    Ratio of total current balance to high credit/credit limit on all install acct
Name: Description, dtype: object
------
mths_since_rcnt_il 74.27207420205511
53    Months since most recent installment accounts opened
Name: Description, dtype: object
------
total_bal_il 73.93159173281025
101    Total current balance of all installment accounts
Name: Description, dtype: object
------
open_rv_12m 73.93159173281025
78    Number of revolving trades opened in past 12 months
Name: Description, dtype: object
------
open_rv_24m 73.93159173281025
79    Number of revolving trades opened in past 24 months
Name: Description, dtype: object
------
mo_sin_old_il_acct 6.0602242169942455
45    Months since oldest bank installment account opened
Name: Description, dtype: object
------
mo_sin_old_rev_tl_op 3.4864053832863506
46    Months since oldest revolving account opened
Name: Description, dtype: object
------
mo_sin_rcnt_rev_tl_op 3.4864053832863506
47    Months since most recent revolving account opened
Name: Description, dtype: object
------
bc_util 1.9067537899946738
9    Ratio of total current balance to high credit/credit limit for all bankcard accounts.
Name: Description, dtype: object
------
revol_util 0.05014354564231804
91    Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credi
t.
Name: Description, dtype: object
------
```

As an example, if we look at mths_since_rcnt_il, it shows 74.2% values are null.

I am going to convert all these remaining nulls to 0.

2.2. Remove features related to defaulted payment plans

Since the loan data uploaded by Lending Club is a comprehensive data, there is lot of entries regarding the loan default and a payment plan. Such data are not available during the loan processing. Hence, those should be removed. Examples of such data are the following:

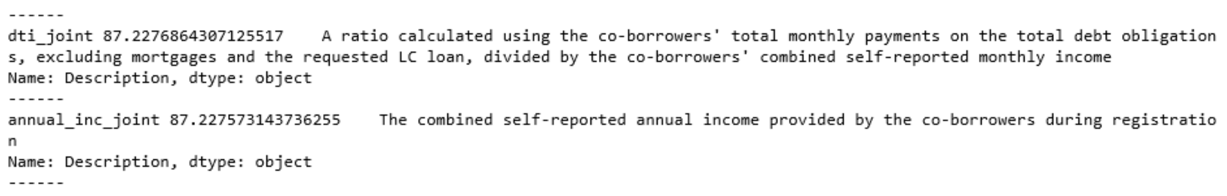


```
settlement_term 99.20591493960671150    The number of months that the borrower will be on the settlement plan
Name: Description, dtype: object
------
member_id 90.0067292463923844    A unique LC assigned Id for the borrower member.
Name: Description, dtype: object
------
url 90.00672924639238111    URL for the LC page with listing data.
Name: Description, dtype: object
------
orig_projected_additional_accrued_interest 89.76616435221374140    The original projected additional interest amount that will
accrue for the given hardship payment plan as of the Hardship Start Date. This field will be null if the borrower has broken th
eir hardship payment plan.
Name: Description, dtype: object
------
hardship_type 89.71631808264058129    Describes the hardship plan offering
Name: Description, dtype: object
------
hardship_reason 89.71631808264058130    Describes the reason the hardship plan was offered
Name: Description, dtype: object
------
hardship_status 89.71631808264058131    Describes if the hardship plan is active, pending, canceled, completed, or broken
Name: Description, dtype: object
------
deferral_term 89.71631808264058132    Amount of months that the borrower is expected to pay less than the contractual monthly p
ayment amount due to a hardship plan
Name: Description, dtype: object
------
hardship_amount 89.71631808264058133    The interest payment that the borrower has committed to make each month while they are
on a hardship plan
Name: Description, dtype: object
------
```

The number next to the feature (e.g., 89.71) is the percentage of entries that are null (already described in previous section). All the payment plan related features are removed.

2.3. Remove features related to joint account

Another set of features where most of the entries are null is joint account related features. Hence, such features are also removed. Following are some examples:



```
------
dti_joint 87.2276864307125517    A ratio calculated using the co-borrowers' total monthly payments on the total debt obligation
s, excluding mortgages and the requested LC loan, divided by the co-borrowers' combined self-reported monthly income
Name: Description, dtype: object
------
annual_inc_joint 87.227573143736255    The combined self-reported annual income provided by the co-borrowers during registratio
n
Name: Description, dtype: object
------
```

2.4. Remove features related to payments already made

Again, we need to remove features which represents the payments already made in an ongoing loan. Examples include last_pymnt_amnt, total_pymnt, etc.

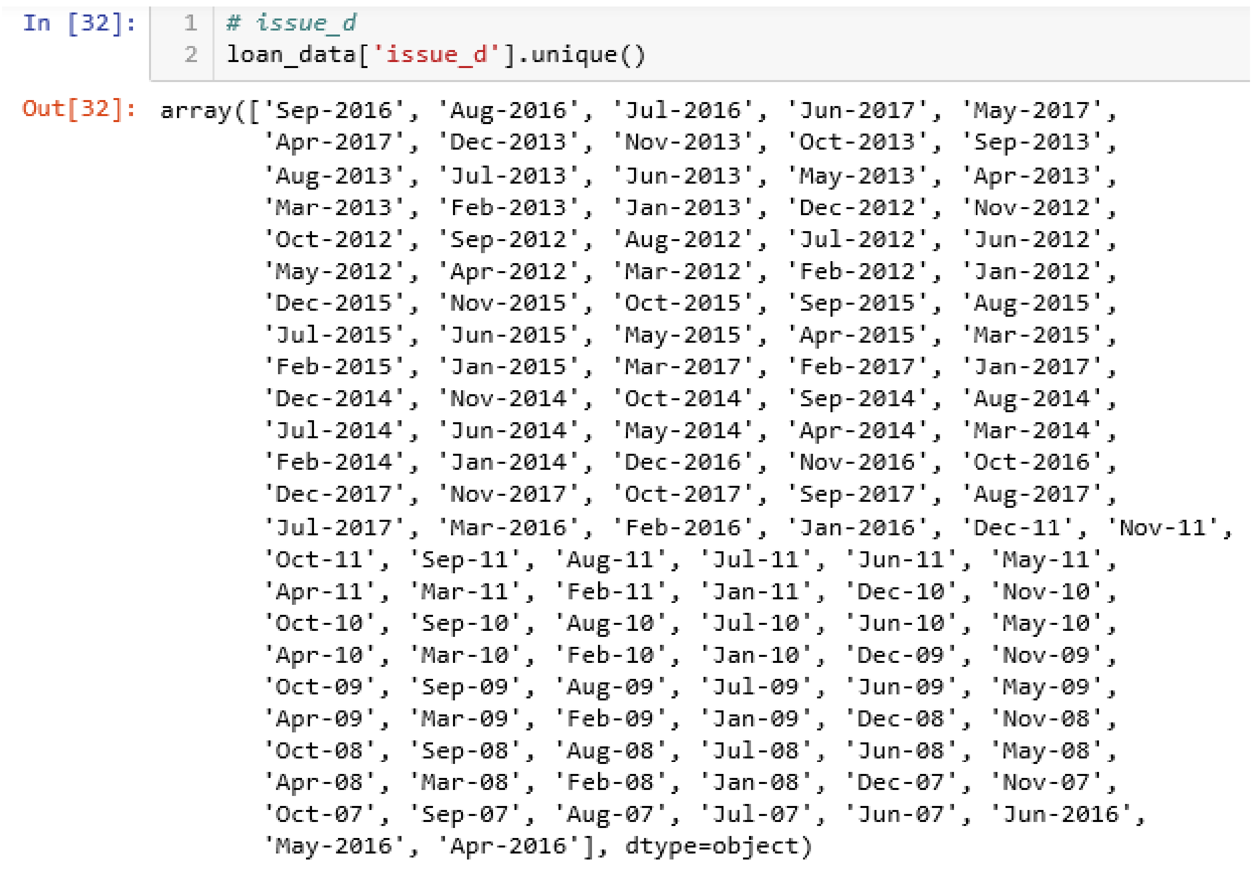
2.5. Clean numeric entries

There are multiple numeric entries, which are not loaded as numeric while reading from CSV file. Examples include the following.

1. When entries are loaded as objects because of one or more numeric entries or null data.
2. Term related entries can show up as “12 month”. Those are made simply “12”.
3. Interest entries look like “4.3%”. Those are cleaned to “4.3”.

2.6. Cleaning date entries

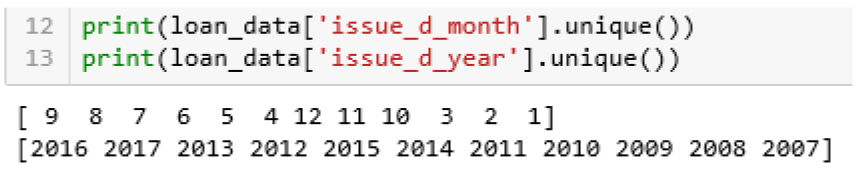
Date entries are written as “May-16”, or “May-2016”. I divided date entries to two separate features. One holds the date and the other holds the year. The issue-date feature looked like this before the change:



```
In [32]:  # issue_d
          loan_data['issue_d'].unique()

Out[32]: array(['Sep-2016', 'Aug-2016', 'Jul-2016', 'Jun-2017', 'May-2017',
                'Apr-2017', 'Dec-2013', 'Nov-2013', 'Oct-2013', 'Sep-2013',
                'Aug-2013', 'Jul-2013', 'Jun-2013', 'May-2013', 'Apr-2013',
                'Mar-2013', 'Feb-2013', 'Jan-2013', 'Dec-2012', 'Nov-2012',
                'Oct-2012', 'Sep-2012', 'Aug-2012', 'Jul-2012', 'Jun-2012',
                'May-2012', 'Apr-2012', 'Mar-2012', 'Feb-2012', 'Jan-2012',
                'Dec-2015', 'Nov-2015', 'Oct-2015', 'Sep-2015', 'Aug-2015',
                'Jul-2015', 'Jun-2015', 'May-2015', 'Apr-2015', 'Mar-2015',
                'Feb-2015', 'Jan-2015', 'Mar-2017', 'Feb-2017', 'Jan-2017',
                'Dec-2014', 'Nov-2014', 'Oct-2014', 'Sep-2014', 'Aug-2014',
                'Jul-2014', 'Jun-2014', 'May-2014', 'Apr-2014', 'Mar-2014',
                'Feb-2014', 'Jan-2014', 'Dec-2016', 'Nov-2016', 'Oct-2016',
                'Dec-2017', 'Nov-2017', 'Oct-2017', 'Sep-2017', 'Aug-2017',
                'Jul-2017', 'Mar-2016', 'Feb-2016', 'Jan-2016', 'Dec-11', 'Nov-11',
                'Oct-11', 'Sep-11', 'Aug-11', 'Jul-11', 'Jun-11', 'May-11',
                'Apr-11', 'Mar-11', 'Feb-11', 'Jan-11', 'Dec-10', 'Nov-10',
                'Oct-10', 'Sep-10', 'Aug-10', 'Jul-10', 'Jun-10', 'May-10',
                'Apr-10', 'Mar-10', 'Feb-10', 'Jan-10', 'Dec-09', 'Nov-09',
                'Oct-09', 'Sep-09', 'Aug-09', 'Jul-09', 'Jun-09', 'May-09',
                'Apr-09', 'Mar-09', 'Feb-09', 'Jan-09', 'Dec-08', 'Nov-08',
                'Oct-08', 'Sep-08', 'Aug-08', 'Jul-08', 'Jun-08', 'May-08',
                'Apr-08', 'Mar-08', 'Feb-08', 'Jan-08', 'Dec-07', 'Nov-07',
                'Oct-07', 'Sep-07', 'Aug-07', 'Jul-07', 'Jun-07', 'Jun-2016',
                'May-2016', 'Apr-2016'], dtype=object)
```

After the change, there are two features: issue_d_month and issue_d_year. And their unique values look like this:



```
print(loan_data['issue_d_month'].unique())
print(loan_data['issue_d_year'].unique())

[ 9  8  7  6  5  4 12 11 10  3  2  1]
[2016 2017 2013 2012 2015 2014 2011 2010 2009 2008 2007]
```

After that, I removed the original “issue_d” feature.

2.7. Remove text-only features

Some features like “description”, “title”, “url”, etc. have only text in them. Hence, they are removed.

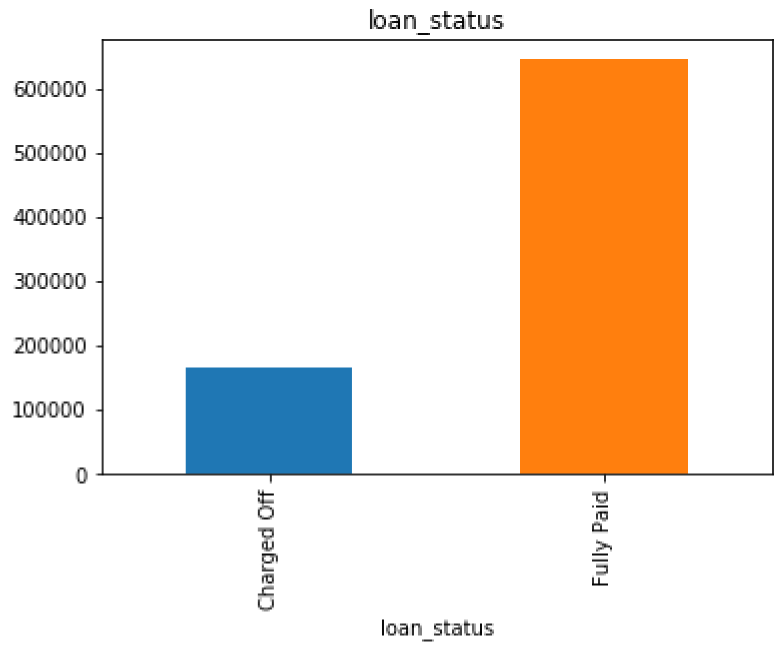
2.8. Special treatment for mths_since_last_major_derog == NULL

While, at most of the places, I either removed the category with a lot of null entries, or I assigned them to 0, the feature “mths_since_last_major_derog” needed special attention. It is the number of months since last negative remarks in the credit history. And a null entry means, there was no major negative entry. Hence, I assign it a value of 1200 which stands for 100 years. That means, that loan applicant’s last major derogatory remark was 100 years back.

3. Data Analysis

3.1. Loan-Status (the label)

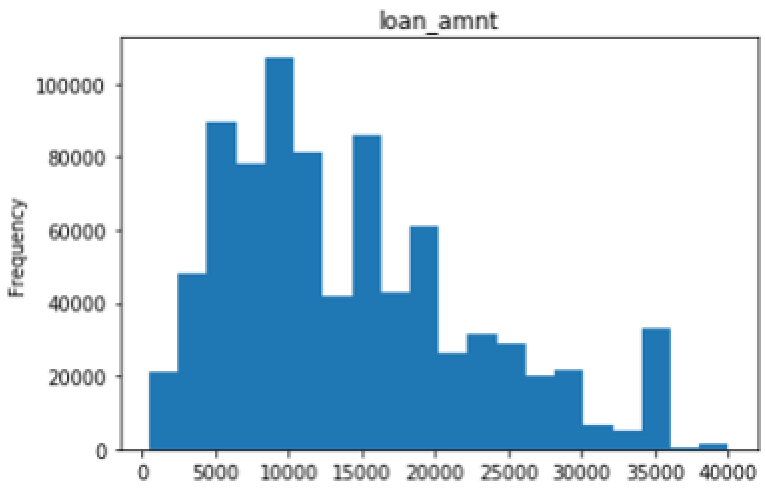
The following plot shows the number of good loan vs bad loan. As you see, the label data is skewed. We need to keep an eye on the F1-score of the algorithm. And if needed we will have to try an imbalanced learning.



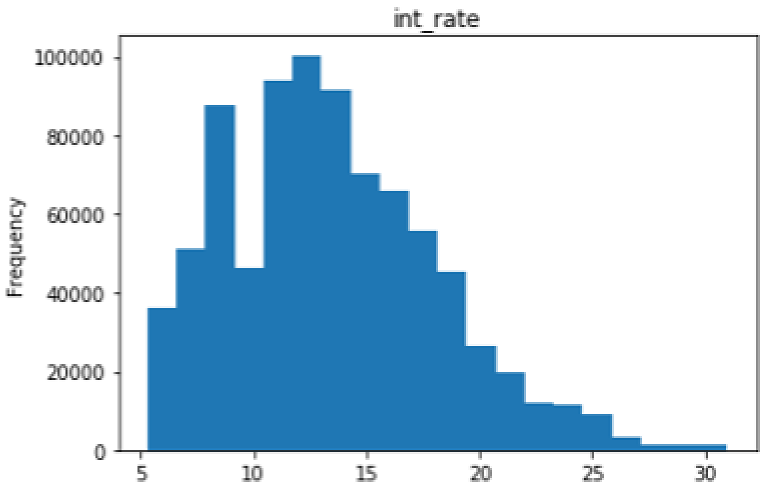
3.2. Histogram

The histograms of all the features are plotted in the file data-cleanup.ipynb.

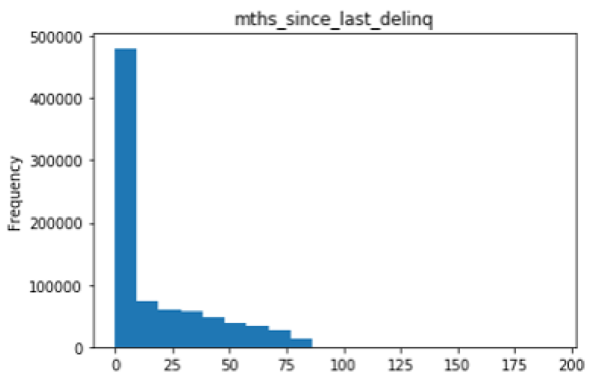
The loan amount histogram is the following. It shows that many loans were around 10k dollars. And the maximum amount was around 40k dollars.

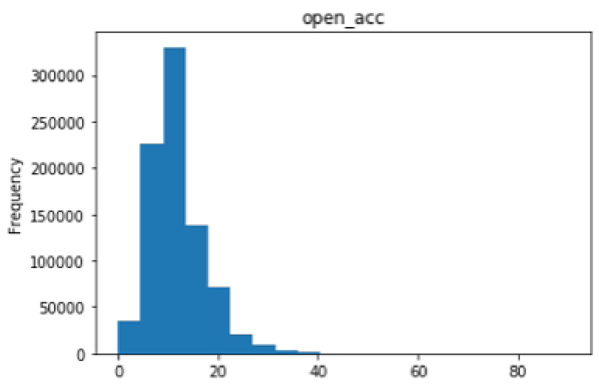


The following shows interest rate histogram. While the peak of the histogram is at around 12%, it is surprising to see 30% interest rates. Important to note is the point that I don't use interest rate in my training and prediction. It is because, interest rate is also a function of all the input features and it can't be used as a primary input feature for judging loan status.

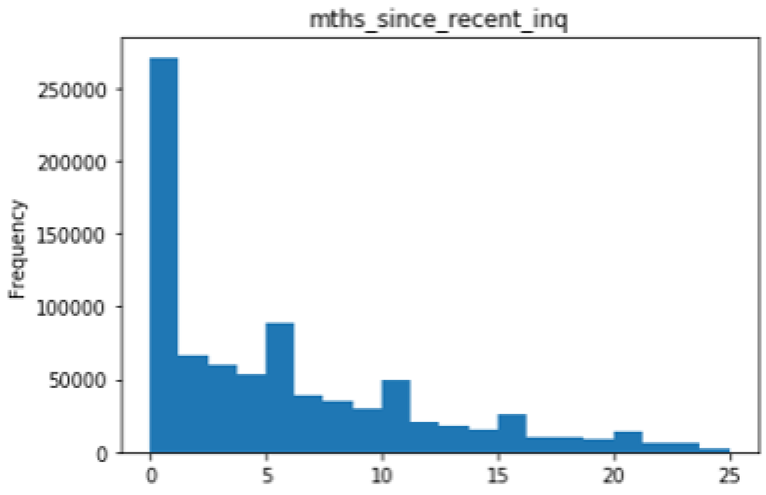


The following shows the number of months since last delinquency and the number of open account. The don't show anything out of ordinary.

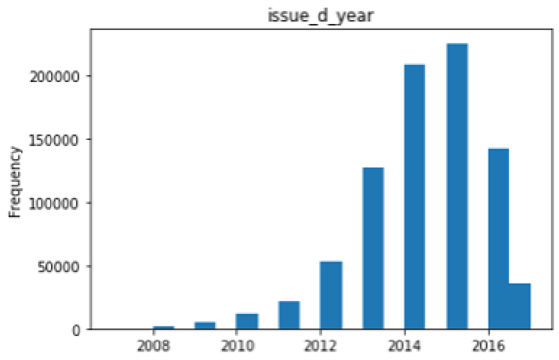


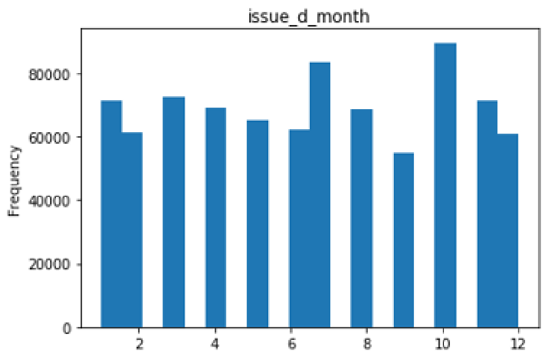


The following shows the months since recent inquiry. The number of 0s is high, which means the applicant has applied for loan (or finance related help) recently.

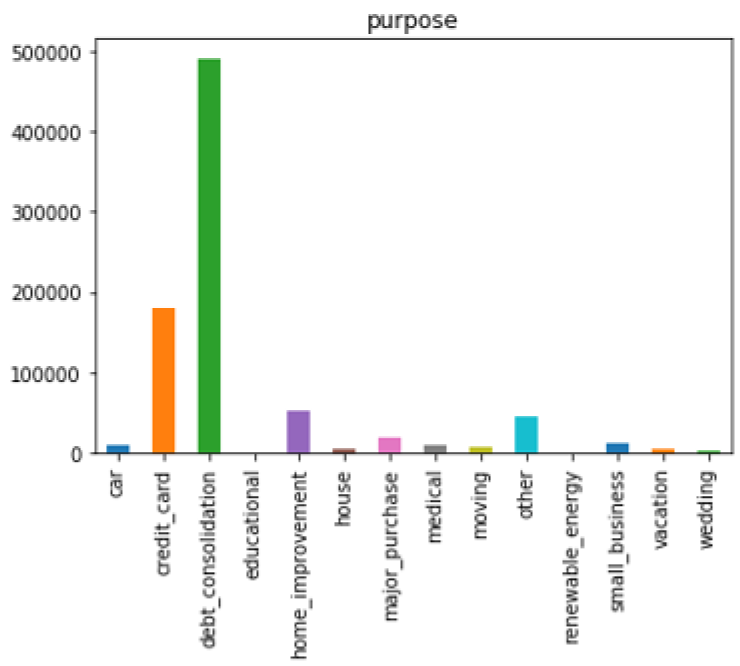


The following shows the yearly distribution of loans and one can see there has been a surge in loans in 2015. Similarly, if you look at the months, July and October has maximum number of loans and September has the least. July could be summer vacation plans and October could be thanksgiving plans.





In terms of purpose, debt-consolidation is the biggest reason. This contradicts my above inference that the biggest loan reasons could be vacationing.



The file data-cleanup.ipynb has more histograms.

3.3. Correlation

After running correlation function on the dataset, I didn't find anything non-obvious. Finance institutions collect a lot of redundant information. And hence a bunch of features show high correlations. The following is the feature pairs, where the correlation was more than 0.5.

It is obvious that "total high credit limit" and "total current balance" would be correlated. Similarly, "number of revolving accounts" and "number of bankcard accounts" would be similar. In the following, most of the pairs are like this.

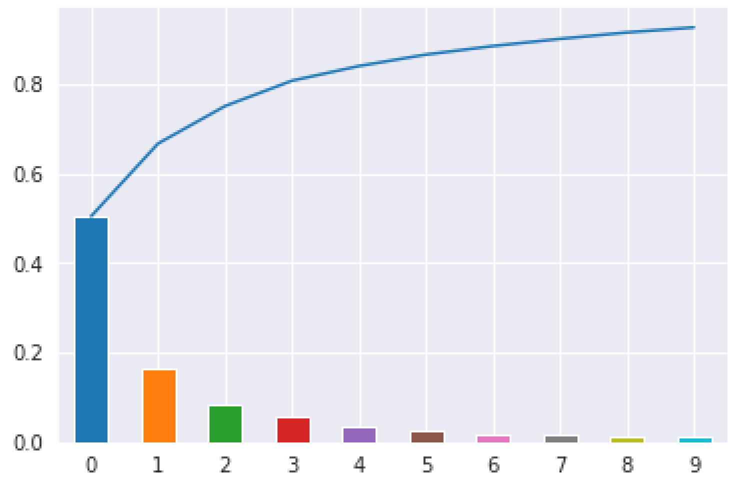
There are others which don't make sense. Like "issue year" and "percentage of trade never delinquent". I believe they are just coincidence. Or, may be, as years pass the collective financial behavior change. And hence, some of these features are correlated to the issue-year.

```
0.5 ('num_rev_accts', 'mo_sin_old_rev_tl_op')
0.51 ('last_credit_pull_d_year', 'last_credit_pull_d_month')
0.53 ('total_bal_ex_mort', 'tot_hi_cred_lim')
0.54 ('total_bal_ex_mort', 'num_il_tl')
0.55 ('num_sats', 'acc_open_past_24mths')
0.56 ('issue_d_year', 'il_util')
0.57 ('num_sats', 'num_bc_tl')
0.58 ('num_rev_accts', 'open_acc')
0.59 ('issue_d_year', 'pct_tl_nvr_dlq')
0.6 ('issue_d_year', 'all_util')
0.61 ('num_sats', 'total_acc')
0.62 ('inq_last_12m', 'open_rv_24m')
0.63 ('total_il_high_credit_limit', 'num_il_tl')
0.64 ('num_tl_90g_dpd_24m', 'delinq_2yrs')
0.65 ('all_util', 'max_bal_bc')
0.66 ('pub_rec_bankruptcies', 'pub_rec')
0.67 ('num_sats', 'num_bc_sats')
0.68 ('num_rev_accts', 'total_acc')
0.69 ('tax_liens', 'pub_rec')
0.7 ('total_bc_limit', 'total_rev_hi_lim')
0.71 ('num_sats', 'num_rev_tl_bal_gt_0')
0.72 ('bc_util', 'revol_util')
0.73 ('num_bc_tl', 'num_bc_sats')
0.74 ('earliest_cr_line_year', 'mo_sin_old_rev_tl_op')
0.75 ('num_op_rev_tl', 'open_acc')
0.76 ('num_tl_op_past_12m', 'acc_open_past_24mths')
0.77 ('open_rv_12m', 'open_acc_6m')
0.78 ('total_rev_hi_lim', 'revol_bal')
0.79 ('pub_rec_bankruptcies', 'mths_since_last_record')
0.8 ('num_tl_30dpd', 'acc_now_delinq')
0.81 ('tot_hi_cred_lim', 'avg_cur_bal')
0.83 ('num_rev_tl_bal_gt_0', 'num_actv_bc_tl')
0.84 ('total_bc_limit', 'bc_open_to_buy')
0.85 ('percent_bc_gt_75', 'bc_util')
0.86 ('total_il_high_credit_limit', 'total_bal_ex_mort')
0.87 ('num_rev_accts', 'num_bc_tl')
0.9 ('num_sats', 'open_acc')
0.95 ('installment', 'funded_amnt_inv')
0.98 ('tot_hi_cred_lim', 'tot_cur_bal')
0.99 ('num_rev_tl_bal_gt_0', 'num_actv_rev_tl')
1.0 ('funded_amnt_inv', 'funded_amnt')
```

The file predict-bad-loans.ipynb has a correlation heat map. The plot is really big and dense as it is an 81x81 size matrix.

3.4. PCA

I run PCA decomposition on the features. The explained variance and cumulative explained variances is shown below.



The cumulative explained variance for the first 10 components is: `[0.50465254 0.66727775 0.75096884 0.80764944 0.84039564 0.8660307 0.88508892 0.90108556 0.91546365 0.92665589]`

Thus, with 10 components, I get 92.7% of the variance of the original data captured. This information is very useful when working with a medium compute resource. Since, I used Azure VMs with GPUs, I could use the full feature set of around 200 features.

4. Data Preprocessing

4.1. Convert categorical data to numeric

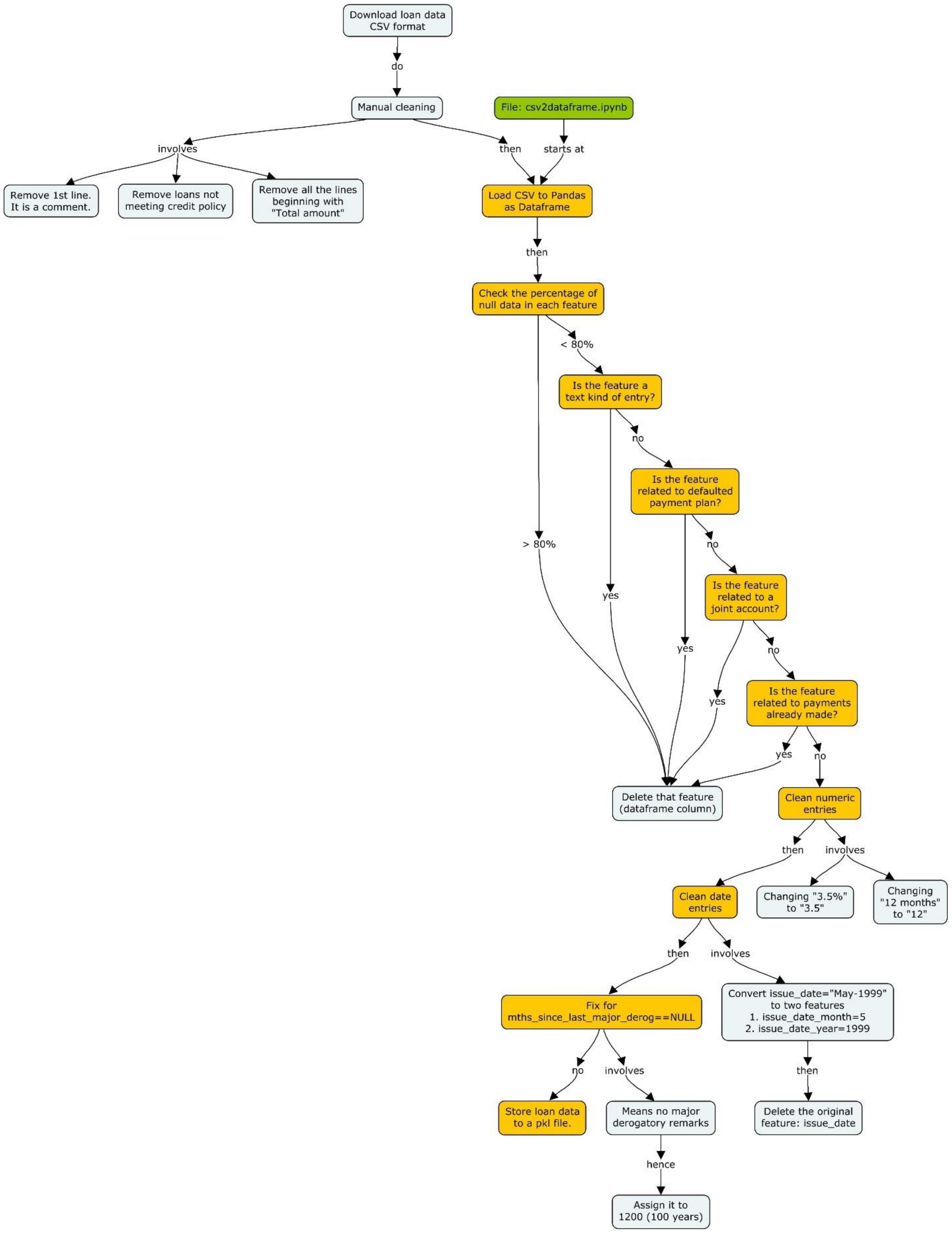
After all the cleaning 8 features were categorical. The following code was used to convert them to numeric. I used "get_dummies" from pandas.

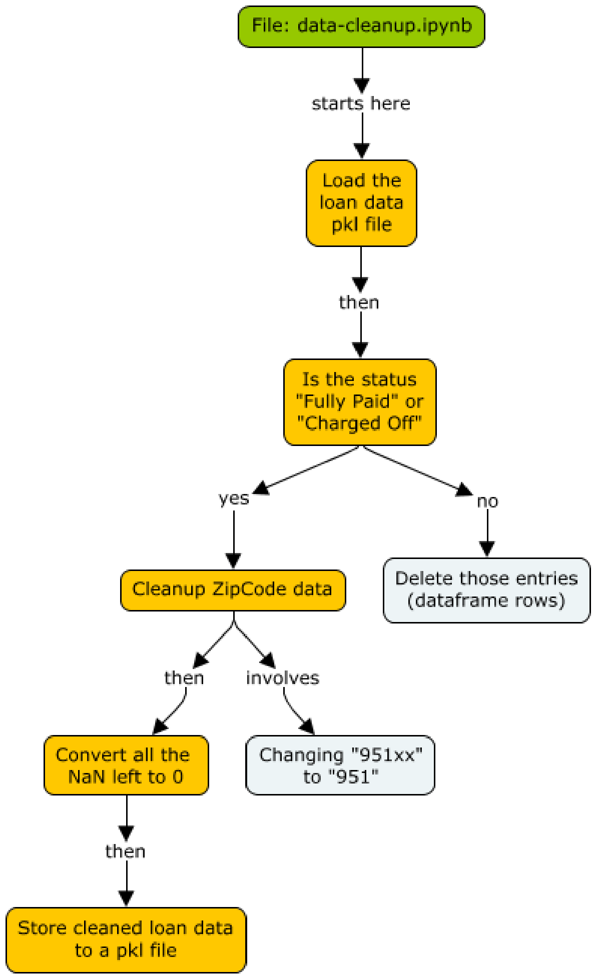
```
# Categorical features
cat_features = features.select_dtypes(include=['object']).columns
print(cat_features)

for y in cat_features:
    features = features.join(pd.get_dummies(features[y], prefix=y))
    features.drop(y, axis=1, inplace=True)
```

```
Index(['grade', 'sub_grade', 'emp_length', 'home_ownership',
       'verification_status', 'purpose', 'addr_state', 'initial_list_status'],
      dtype='object')
```

The cleaning and processing mentioned in Sections 2 and 4 are shown in the following two mind maps.

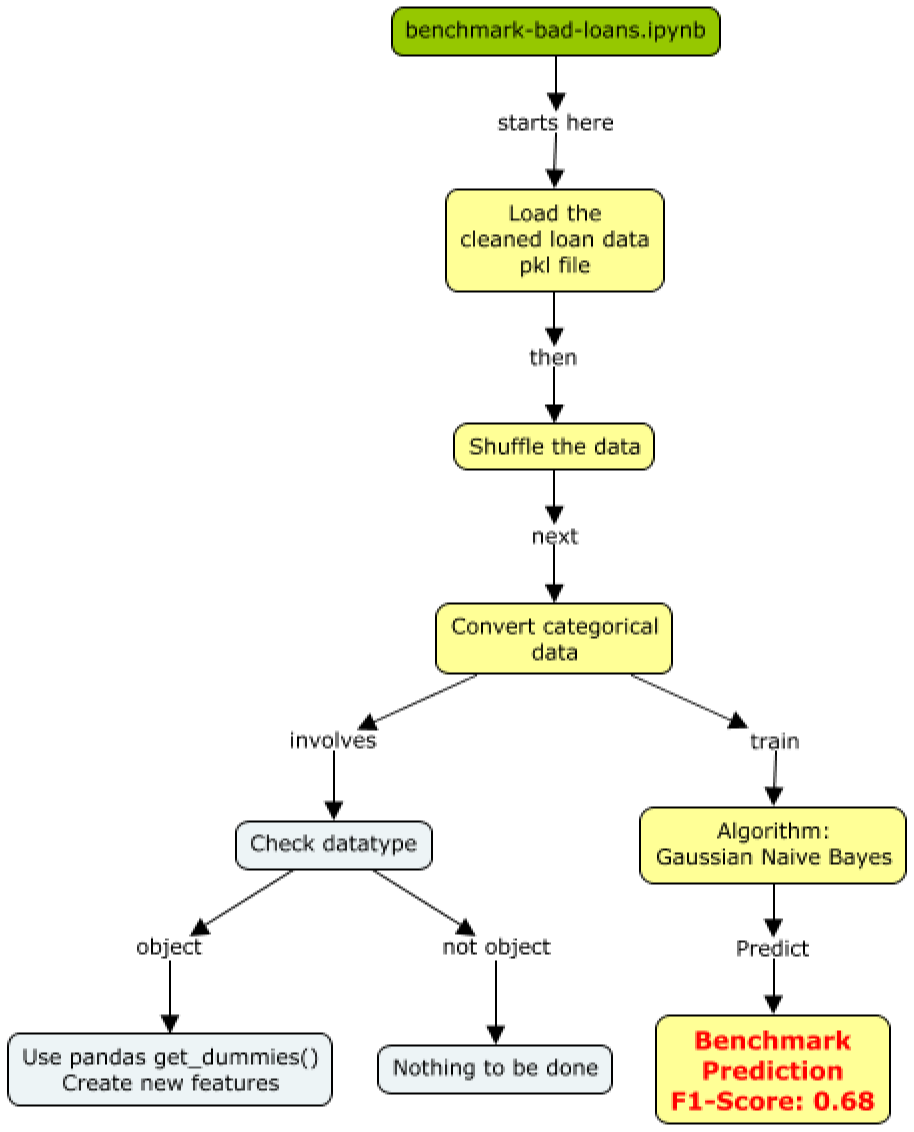

Download loan data CSV format
do
Manual cleaning
involves
Remove 1st line. It is a comment.
Remove loans not meeting credit policy
Remove all the lines beginning with "Total amount"
then
File: csv2dataframe.ipynb
starts at
Load CSV to Pandas as Dataframe
then
Check the percentage of null data in each feature
< 80%
> 80%
Is the feature a text kind of entry?
no
yes
Is the feature related to defaulted payment plan?
no
yes
Is the feature related to a joint account?
no
yes
Is the feature related to payments already made?
yes
no
Delete that feature (dataframe column)
Clean numeric entries
then
involves
Changing "3.5%" to "3.5"
Changing "12 months" to "12"
Clean date entries
then
involves
Convert issue_date="May-1999" to two features 1. issue_date_month=5 2. issue_date_year=1999
then
Delete the original feature: issue_date
Fix for mths_since_last_major_derog==NULL
no
involves
Store loan data to a pkl file.
Means no major derogatory remarks
hence
Assign it to 1200 (100 years)


File: data-cleanup.ipynb
starts here
Load the loan data pkl file
then
Is the status "Fully Paid" or "Charged Off"
yes
no
Cleanup ZipCode data
Delete those entries (dataframe rows)
then
involves
Convert all the NaN left to 0
Changing "951xx" to "951"
then
Store cleaned loan data to a pkl file

5. Benchmark

For benchmarking purpose, I used Naïve Bayes classifier for finding bad loans and Logistic Regression for predicting the interest rates.

The bad-loan benchmarking is done in benchmark-bad-loans.ipynb. The benchmarking process is shown in the following mind map.



benchmark-bad-loans.ipynb
starts here
Load the cleaned loan data pkl file
then
Shuffle the data
next
Convert categorical data
involves
train
Check datatype
Algorithm: Gaussian Naive Bayes
object
not object
Predict
Use pandas get_dummies() Create new features
Nothing to be done
Benchmark Prediction F1-Score: 0.68

Here are the results:

```
    GaussianNB trained on 648460 samples.

 {'train_time': 2.4216957092285156,
  'pred_time': 2.8948965072631836,
  'acc_train': 0.45397248866545353,
  'acc_test': 0.453650472501172,
  'f_train': 0.6838292667878828,
  'f_test': 0.6839424103307327}
```

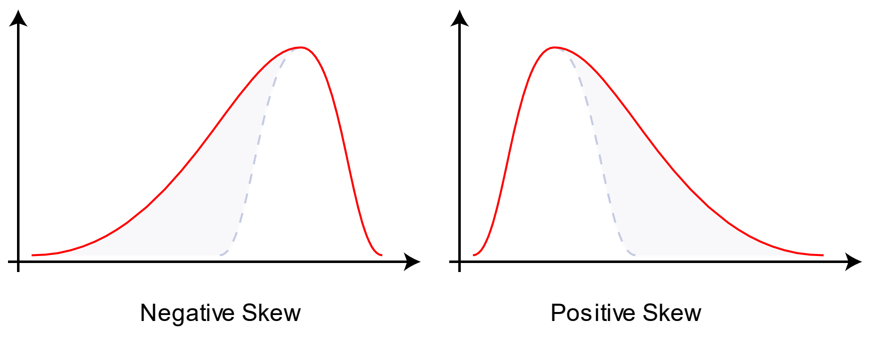
The F1-Score on the test data is 0.68.

6. Solution

6.1. Checking Skewness

In probability theory and statistics, skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean.

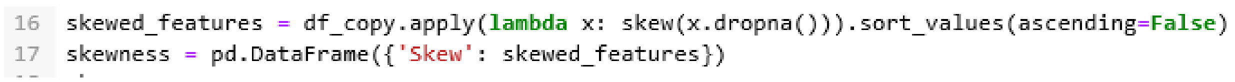
The figures below show a Negative and Positive skew.



Most of the Algorithms for machine learning assume that the data follow a normal distribution. Skewed data affects the regression intercept and the coefficients associated with the model.

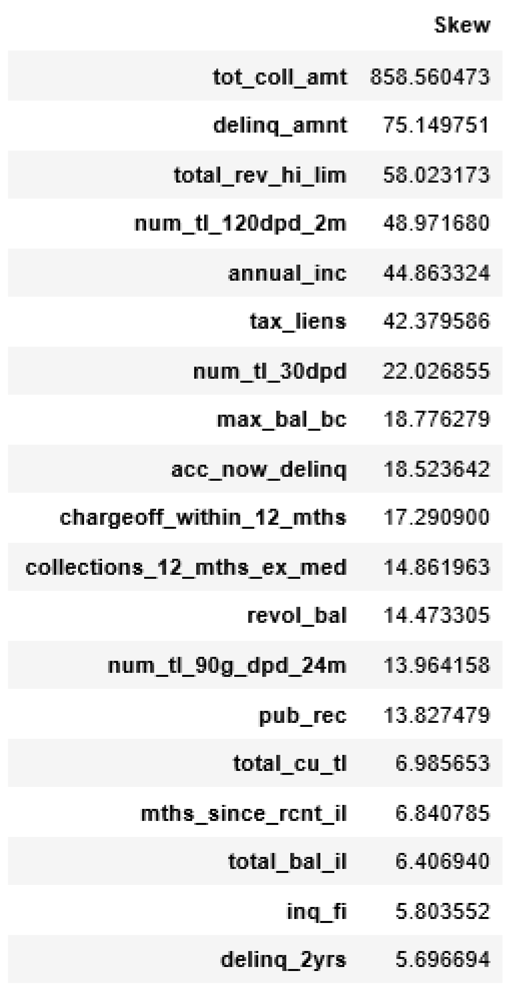
A quick look at the skewness of the data shows that the data is heavily skewed. The following is a short snapshot of the skew data. And data outside the rage of (-1, 1) is considered highly skewed.

I used pandas skew function to check the skewness.



```
skewed_features = df_copy.apply(lambda x: skew(x.dropna())).sort_values(ascending=False)
skewness = pd.DataFrame({'Skew': skewed_features})
```

The following sections talks about Box Cox transform which tries to convert this skewed data to a normal curve.



	Skew
tot_coll_amt	858.560473
delinq_amnt	75.149751
total_rev_hi_lim	58.023173
num_tl_120dpd_2m	48.971680
annual_inc	44.863324
tax_liens	42.379586
num_tl_30dpd	22.026855
max_bal_bc	18.776279
acc_now_delinq	18.523642
chargeoff_within_12_mths	17.290900
collections_12_mths_ex_med	14.861963
revol_bal	14.473305
num_tl_90g_dpd_24m	13.964158
pub_rec	13.827479
total_cu_tl	6.985653
mths_since_rcnt_il	6.840785
total_bal_il	6.406940
inq_fi	5.803552
delinq_2yrs	5.696694

6.2. Box Cox transform

A Box Cox transform is a way to convert non-normal variables to fit to normal curve. For a data ‘y’, it does the following function:

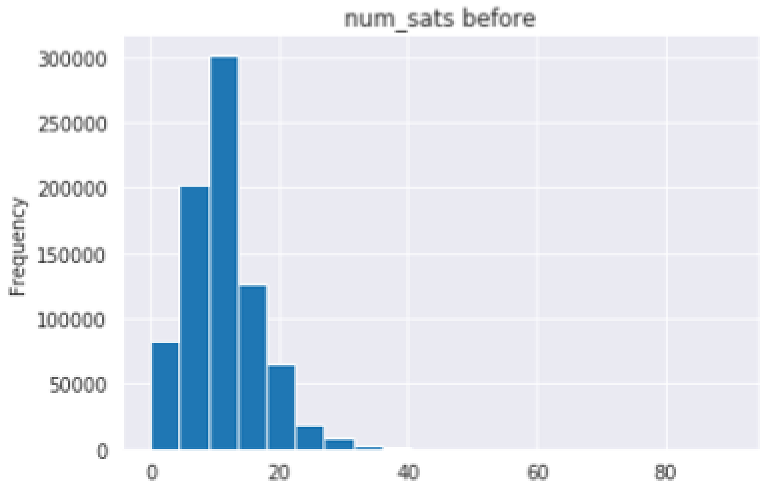
$$y(\lambda) = \begin{cases} \frac{y^\lambda - 1}{\lambda}, & \text{if } \lambda \neq 0; \\ \log y, & \text{if } \lambda = 0. \end{cases}$$

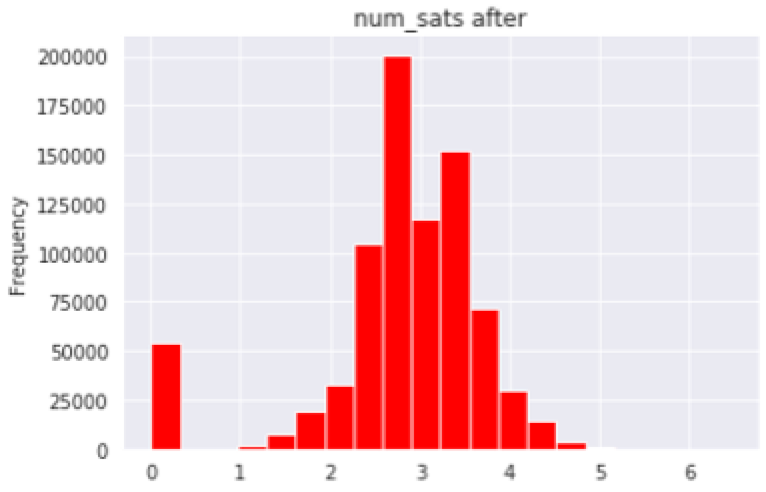
The value for lambda is varied from -5 to +5. Then the best value of lambda, which gives the best approximation of the normal curve is considered. The whole process is achieved in Python scipy library by the following call.

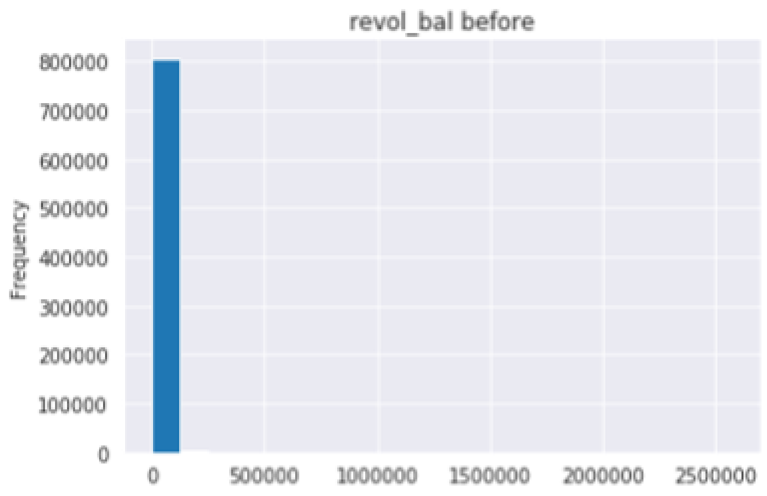
```
df[feat] = boxcox1p(df[feat], lam)
```

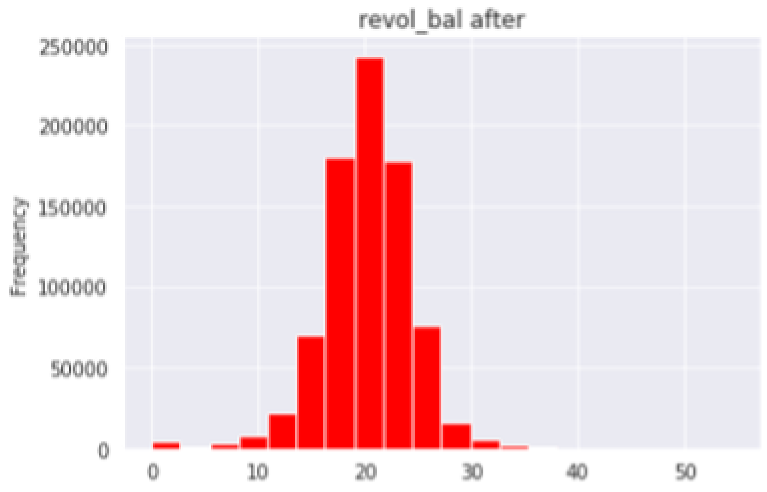
I used a value of 0.15 for lambda. More about boxcox1p function is available here.

The following plots show two examples of how boxcox function improved the normality of the curve. The blue graph is the original data and the red one is after the transform.







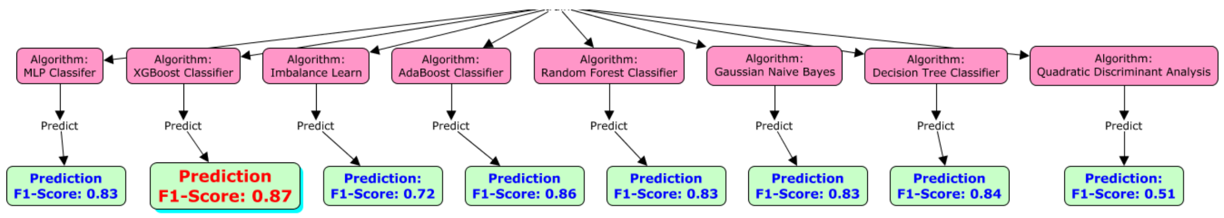


It must be noted that not all features in the loan-data-set showed this kind of property with boxcox transform. Some of the features were still very skewed.

6.3. Using different algorithms

For benchmarking purpose, only Gaussian Naïve Bayes classifier was used. But to get better prediction performance, I looked for other algorithms also. The following shows the performance of all the algorithm after applying boxcox transform. The algorithms I used are:

1. MLP classifier
2. XGBoost classifier
3. Imbalance learn classifier
4. AdaBoost classifier
5. Random Forest classifier
6. Gaussian Naïve Bayes classifier
7. Decision Tree classifier
8. Quadratic Discriminant Analysis classifier


Algorithm: MLP Classifer
Algorithm: XGBoost Classifier
Algorithm: Imbalance Learn
Algorithm: AdaBoost Classifier
Algorithm: Random Forest Classifier
Algorithm: Gaussian Naive Bayes
Algorithm: Decision Tree Classifier
Algorithm: Quadratic Discriminant Analysis
Predict
Prediction F1-Score: 0.83
Prediction F1-Score: 0.87
Prediction: F1-Score: 0.72
Prediction F1-Score: 0.86
Prediction F1-Score: 0.83
Prediction F1-Score: 0.83
Prediction F1-Score: 0.84
Prediction: F1-Score: 0.51

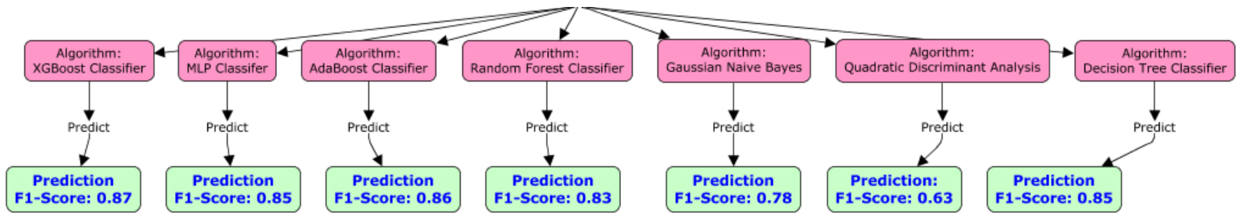
6.4. Scale / Normalize data

Several algorithms converge faster when the data is normalized. Also, in the case of loan data a lot of features have different units and ranges compared to others. For example, the loan amount requested is a value which is in high thousands. But, the number bankruptcies in the past is generally less than 10. Hence, it is a good idea to apply a scaling transform.

I used the StandardScaler function from sklearn. The code looks like the following:

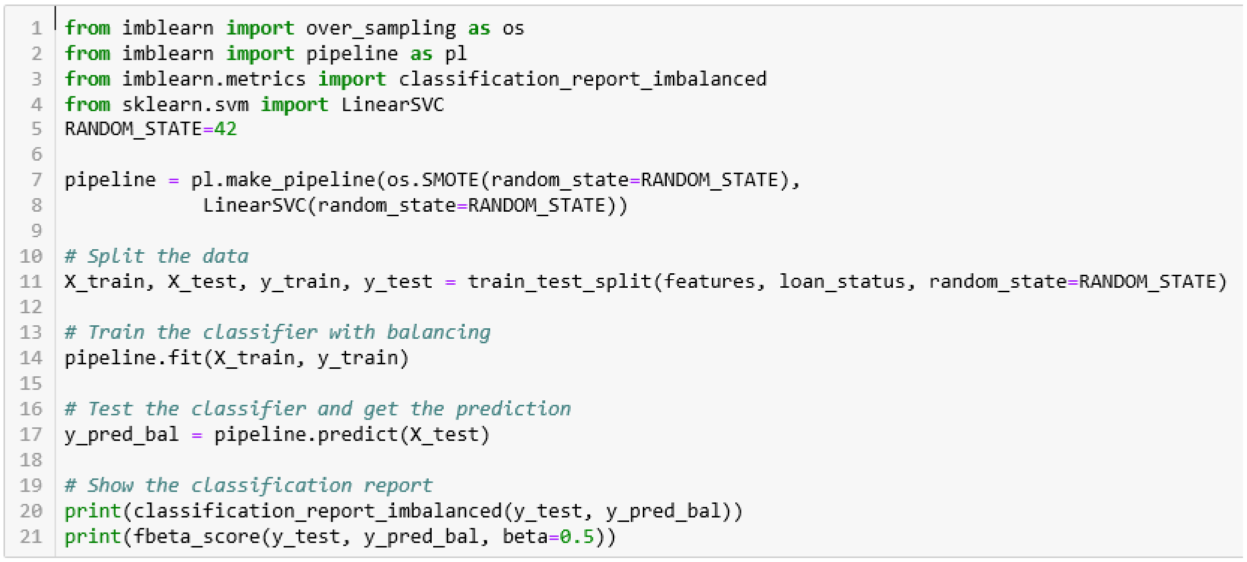
```
features =  StandardScaler().fit_transform(features)
```

The prediction results after scaling is shown below. Only Quadratic Discriminant Analysis (QDA) algorithm benefitted from scaling. And Gaussian NB has affected negatively because of scaling. None of the other algorithms show any changes.


Algorithm: XGBoost Classifier
Algorithm: MLP Classifer
Algorithm: AdaBoost Classifier
Algorithm: Random Forest Classifier
Algorithm: Gaussian Naive Bayes
Algorithm: Quadratic Discriminant Analysis
Algorithm: Decision Tree Classifier
Predict
Prediction F1-Score: 0.87
Prediction F1-Score: 0.85
Prediction F1-Score: 0.86
Prediction F1-Score: 0.83
Prediction F1-Score: 0.78
Prediction: F1-Score: 0.63
Prediction F1-Score: 0.85

6.5. Imbalanced learn

As mentioned in section 3.1, the label is not balanced and hence I used imbalanced learn. Imbalanced learn can be downloaded from here. The following code is used to run the imbalance learn model.



```python
from imblearn import over_sampling as os
from imblearn import pipeline as pl
from imblearn.metrics import classification_report_imbalanced
from sklearn.svm import LinearSVC
RANDOM_STATE=42

pipeline = pl.make_pipeline(os.SMOTE(random_state=RANDOM_STATE),
            LinearSVC(random_state=RANDOM_STATE))

# Split the data
X_train, X_test, y_train, y_test = train_test_split(features, loan_status, random_state=RANDOM_STATE)

# Train the classifier with balancing
pipeline.fit(X_train, y_train)

# Test the classifier and get the prediction
y_pred_bal = pipeline.predict(X_test)

# Show the classification report
print(classification_report_imbalanced(y_test, y_pred_bal))
print(fbeta_score(y_test, y_pred_bal, beta=0.5))
```

The F1-Score for imbalance learn is 0.72 (as shown in section 6.3), which is not the best.

7. Final verdict

Based on the F1-score numbers seen in sections 6.3 and 6.4, I select XGBoost Classifier algorithm. And I intend to go with the original dataset and not the scaled version. The following table summarizes my final verdict.

	Benchmark	**My Solution**
Algorithm	Gaussian Naïve Bayes	XGBoost Classifier
F1-Score	0.68	0.87

8. Reflections and Learnings

8.1. Data cleanup

I found that a lot of time is spent in data cleanup. The data available in real world is not clean and there are lot of missing data.

8.2. Skew

A big surprise to me was how bad the skewed data can make the predictions. In all the learning I did in the class the dataset available was always balanced and I never appreciated the need to un-skew the data. Boxcox is a good technique and I found there are other transformations (Kruskal) also which are used in the industry. In the end, my solution was simply fixing the skewness and done.

8.3. F1-Score vs Accuracy

Being a layperson, I always thought accuracy is a good measure of and prediction technique. Here, with 20% of the loans being defaulted if my prediction algorithm predicts all the loans as good loans, then I am already getting 80% accurate. In every sense, it looks like a good algorithm. Or consider the algorithm to catch fraudulent transaction, which could be less then 0.1% of the total. So, if I blindly declare all the transactions as good ones, will get more than 99.9% accuracy. That is where F1-score is so important. In both the examples, I mentioned just now, the F1-score will be 0, if I blindly predict all the transaction to the good (or good loan).

8.4. GridSearchCV

GridSearchCV was a letdown. I was thinking of running GridSearchCV on my final selected algorithm and fine tune the parameters. But, with around 10 parameters and multiple values for each parameter, the time taken was too much. Hence, I abandoned the effort. One method I found online is to divide the 10 parameters into smaller sets of say 3 parameters each. And then run GridSearchCV. I guess, I will explore those methods later in my future endeavors.

8.5. PCA

PCA shows some surprising results. Even though, one component could account only 50% variance, I found that with just one component I could get the F1-score very close to the full dataset F1-score. I exactly don't know the meaning of that.

8.6. Similarity between algorithms

I found that the F1-score for lot of algorithms were very similar. The exception was QDA and Gaussian NB. I guess, my data was such that all the algorithms could behave the same.