FM 9528 Banking Analytics

Coursework 2

Student ID: 251121253

Word Count: 2201

Part a) Data Cleaning

Loan Data Cleaned as Following :

Missing Values:

Variable Value Missing Condition	Treatment
• >80% missing values	• Dropped
• 50% ~ 80% missing values	• If identified as potential important features for credit scoring and LGD: transformed to indictor value (0 & 1) to represent value exist or not. • If not been identified as potential useful feature for evaluate PD & LGD, dropped.
• 12% ~ 50% missing values	• If identified as potential important features for credit scoring and LGD: filled with values exceeding the range of feature values to represent NaN values. (e.g. -1) otherwise dropped
• A few important variable related PD or LGD with missing values from 2% ~ 6%:	• rows of missing values were removed
• <2% missing values	• replaced either median (for continuous value) or mode (for categorical value).

• Any rows with greater than 50% cells has missing values are removed.

(See Appendix B, Part I, Section "Dealing with Large Missing Values" for details)

Correlated Variables:

• "Spearman" Correlation Matrix were used to capture linear, non-linear correlations among continuous and ordinal variables.

• All correlated variables which are at least 75% correlated were dropped until one or two closely correlated to 'loan_status' remain.

(See Appendix B, Part I, Section "Correlation Analysis & Filtering")

Outliers:

• Box Plots were used on each survived variables.

• All extreme points disjointed from with the continuous dots were eliminated, except those extreme dots which used to represents Null values (meaningful empty cells).

Objective Variable	Decision & Assumptions
“loan_status” Objective Variable for Application Score Card	Original Values contained: `“Fully Paid”, “Current”, “Charged Off”, “Late (31-120 days)”, “In Grace Period”, “Late (16-30 days)”, “Does not meet the credit policy. Status:Fully Paid”, “Does not meet the credit policy. Status:Charged Off”,”Default”.` Objective Transformation Assumptions: • No indeterminate cases are used. • All loan status are transformed to binary results: ❖ “1” : “Charged Off” / Bad / Default ❖ “0” : “Fully Paid” / Good / not default / • For "Current", "In Grace Period”, "Late (16-30 days)”, I still consider them as “good” (0) cases since these cases doesn’t belongs to any definition of default in this course yet. • "Late (31-120 days)" and "Default", I consider them as "Bad" (1) case since the probability of default is really high in these status. • Since “Does not meet the credit policy. Status: Fully Paid” and “Does not meet the credit policy. Status: Charged Off” will not be entered into the application pool, they should be discarded from the sample. In addition to that, they leaks Loan Club’s credit scoring policy.
LGD (Constructed by several original variables)	Construction formula: $$LGD = 1 - \frac{Recoveries + tot\_rec\_late\_fee - collection\_recovery\_fee}{Funded\_amnt - total\_rec\_prncp}$$ Objective Construction Assumption: • Discount rate is ignored here because of the complexity of the dates • All settlement amounts are included in the recoveries • All the late fee charge are considered as a part of recovery • Although “collection_recovery_fee” is paid by investor instead of Lending Club (LC) , I still consider it as a cost in workout LGD because LC has to pay it first, and it may face the risk which investor refuse to pay the collection fee because recovery is too small. I treated it in a conservative way. • All the features used to constructed LGD were dropped to to avoid multicollinearity • All LGD values are caped at 1, floored at 0

Survived Variables for Credit Scoring after Data Cleaning	
Potential Useful Variables	Reason to be Selected (Potentially)
last_credit_pull_d, all_util, open_rv_24m, open_il_24m, inq_fi, max_bal_bc open_act_il, total_cu_tl term, num_tl_op_past_12m, total_bc_limit, mths_since_recent_inq percent_bc_gt_75, inq_last_6mths dti, total_rev_hi_lim mo_sin_rcnt_rev_tl_op, tot_hi_cred_lim, loan_amnt tot_cur_bal, annual_inc purpose, application_type, home_ownership mort_acc, mo_sin_old_rev_tl_op num_op_rev_tl, earliest_cr_line total_acc, mths_since_last_record mths_since_last_major_derog delinq_2yrs, num_actv_bc_tl num_accts_ever_120_pd, zip_code total_il_high_credit_limit mths_since_recent_revol_delinq pub_rec_bankruptcies emp_length tax_liens tot_coll_amt acc_now_delinq delinq_amnt collections_12_mths_ex_med num_tl_120dpd_2m num_tl_90g_dpd_24m chargeoff_within_12_mth	• These variables mainly include bureau scores, background information of the applicants and historical credit behavior / record. • These variables don't contain any information specific to the loan or after the loan was funded since application credit score were evaluated before any loan request was approved. • Most of them has a spearman correlation less than 70% • Do not reveal applicants' identities

Survived Variables for LGD after Data Cleaning	
Potential Useful Variables	Reason to be Selected (Potentially)
All the variables listed in above table and plus following: term, tot_rec_int, installment, loan_amnt, total_pymnt, debt_settlement_flag, int_rate	• These variables includes all the available LGD driver in this data set: credit worthiness, salary, time at job, type of house, partial information related to EAD, and geographic information.

Discarded Variables	
Variables	Reasons not be Selected
All the variables did not show in previous two tables	• Did not survived through data cleaning • Too many categories • (eg. emp_title) • Irrelevant random number / text • (e.g. “url”, “desc”, “member_id”) • Duplicated geographic Information • (e.g. zip_code and states, zip_code survived) • All most the same value for all rows • (e.g. 95% are no for pymt_plan)

Part b) Score Card Construction

(Details in Appendix B, Part II)

Discussion on choices of variables:

Variables are selected based on following criteria:

- Variables are evaluated based on the Weight of Evidence (WoE) and the Information Value (IV)
- Variables’ WoE were divided into bins where an explainable trend of probability of default can be observed
- All the variable whose WoE has an IV greater than 0.1 were selected:

Variables Selected based on Criteria above	Information Value of WoE	Transformation Made
last_fico_range_low	4.071438	None
last_credit_pull_d	0.406819	Transformed from specific dates to years of the date (numerical)
all_util **	0.156401	NaN values were represent as -1 (since -1 is out of range of those variables)
open_rv_24m	0.151765	
open_il_24m	0.146334	
inq_fi	0.130523	
max_bal_bc	0.127302	
open_act_il	0.124997	
total_cu_tl	0.120770	

** “all_util” was further eliminated due to its high correlation with “open_act_il”, “inq_fi” (above 80%)

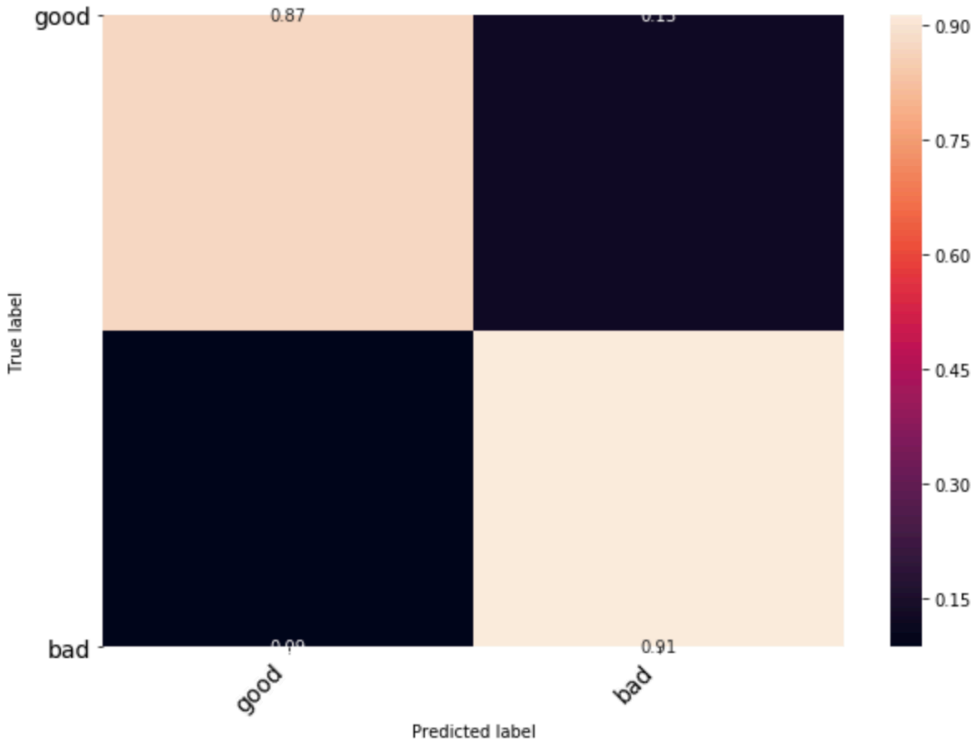
Final Variables Selected	Reason to be Selected
open_act_il	• These are the records of financial accounts of the applicants which reflect applicants’ historical financial behavior. • The higher number of these financial accounts / records, the higher probability of default: more cashflow of the applicants will be drained by these debt accounts and inquires so that applicant are less likely have money left to pay the LC loans. • The WoE also reflects that if these variables are empty, it will have even higher chance of default than those ones which has a record. This is because empty records reflect higher uncertainty of the applicant’s credit history, which could transforms to higher risk of default
open_il_24m	
open_rv_24m	
total_cu_tl	
inq_fi	
last_fico_range_low	• WoE show as PD decreases as FICO score increases. **IV is 4.07 which is suspiciously high, but I included it for following reason:** • Dominance of prediction power. • **(See Performance of the Logistic Regression the next section)** • FICO is one of the largest credit score company in the US, it must have more solid credit information than LC does. (applicants and sales can enter fake information) Therefore, FICO score is much more trustworthy reference, no reason to exclude it.
max_bal_bc	• The higher current balance owed on revolve accounts, the higher liability the applicant has, and therefore higher PD • those ones who don’t have a record has a lower PD since they don’t have a current balance on revolve accounts yet, which leads to lower PD
last_credit_pull_d	• WoE shows that credit record last pulled by LC during the year 2017 & 2018 has the highest probability of default, this may caused by economic downturn in those two years so that applicants face some liquidity problem.

Scorecard Production

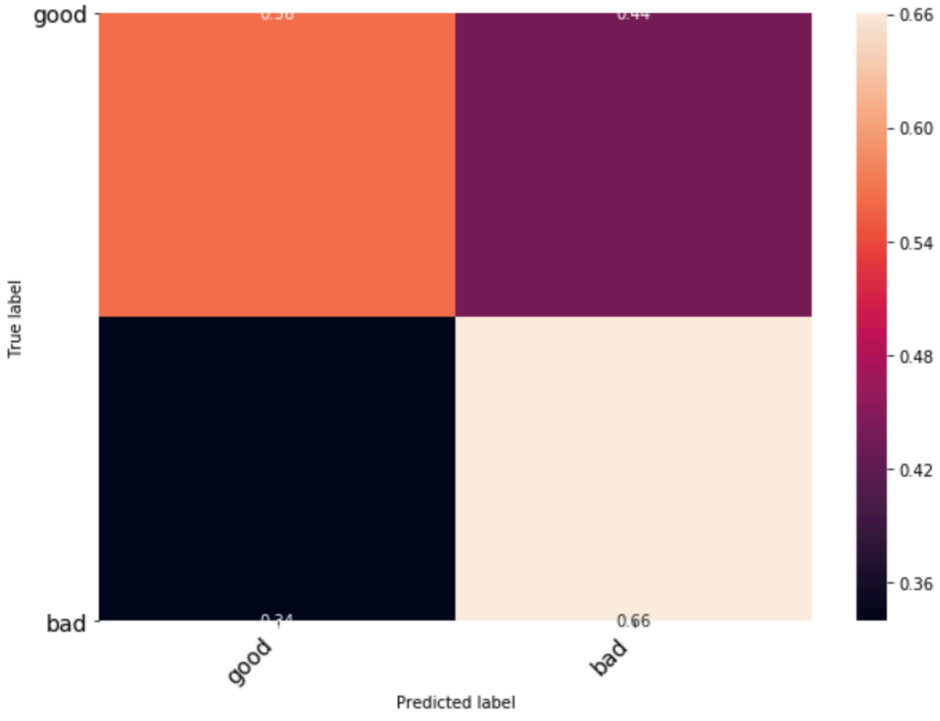
(Please see Appendix A: Application Score Card for Loan Club)

Performance of the Logistic Regression on Selected Variables

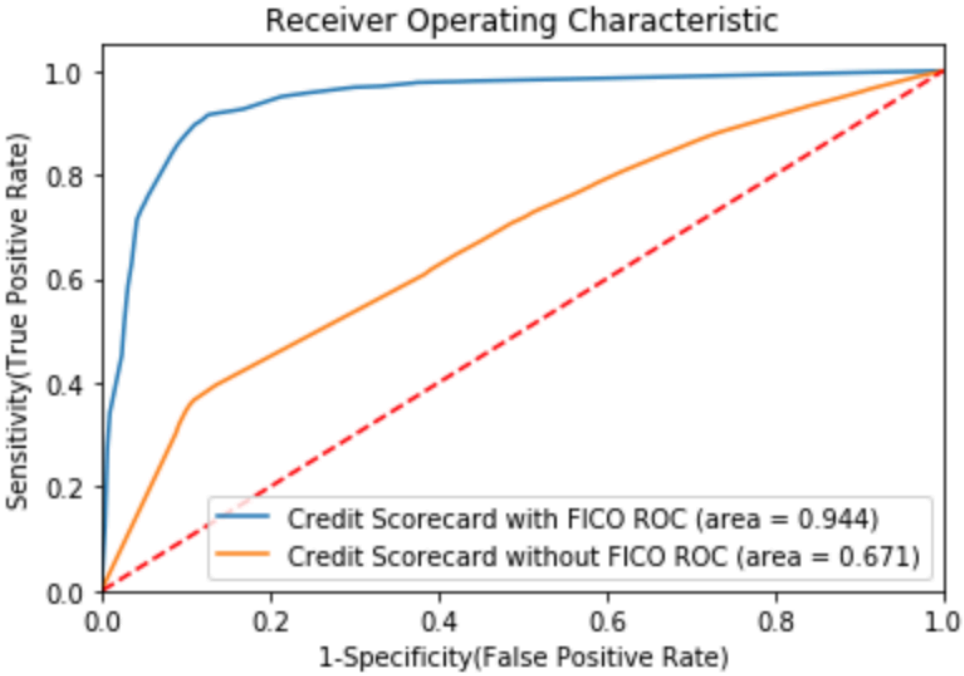
Heat Map of the Scorecard (With FICO)



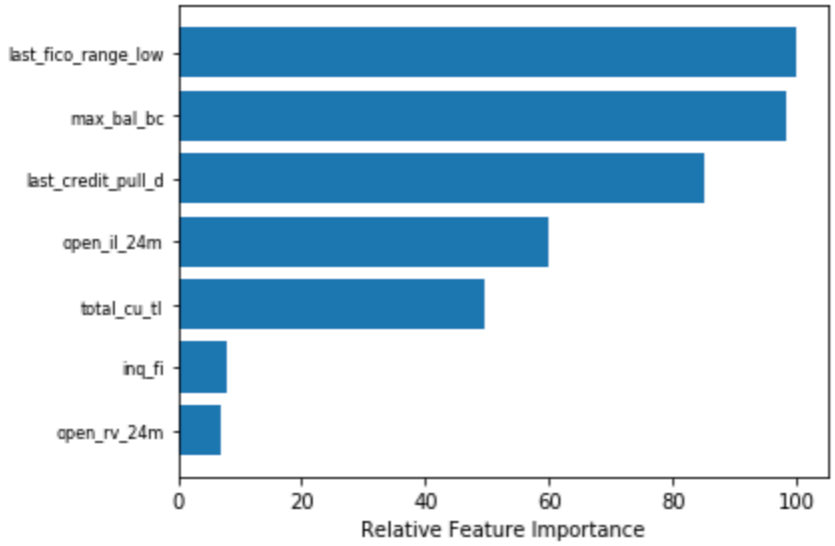
Heat Map of the Scorecard (Without FICO)



AUC Comparison between Models with and without FICO Score



Relative Feature Importance In model (In terms of value of coefficient)



From the above plots, we can see the variable “last_fico_range_low” played an essential role in our model for probability of default here. With it included in our model, the accuracy of predicting “non-defaulted” and “defaulted” loan correctly reached 87% and 91% respectively. These rates will drop to 56% and 66% if we exclude it. We can also see the same results that AUC reduced from 0.944 to 0.671 if we drop “last_fico_range_low” on the ROC curve. Therefore, “last_fico_range_low” is the most important variable in our model because of its incredible prediction power. From the business perspective, although we should be careful with adding other model (FICO score) to our model, such powerful predictor should be included to maximize LC’s business profits.

The “inq_fi” and “open_rv_24m” has little influence on the model given the FICO score included in the model, while rest of the variables has moderate prediction power. In conclusion, it is safe to say we can drop “inq_fi” and “open_rv_24m” if FICO score is included in the application score card. So the optimal model would include 5 features:

- last_fico_range_low
- max_bal_bc
- last_credit_pull_d
- open_il_24m
- total_cu_tl

And this model would provide approximately 90% of accuracy on predicting “default” (89%) and “non-default” (91%) loans.

Part c) LGD Prediction Model

(Details in Appendix B, Part III)

Optimal parameters found using RandomCV and GridSearchCV:

Random Forest:

Bootstrap	Max_depth	max_features	min_sample_leaf	min_sample_split	n_estimator
FALSE	500	sqrt	8	2	5000

XGBoosting

learning _rate	Max_depth	max_features	min_sample_leaf	min_sample_split	n_estimator
0.01	3	sqrt	8	4	2000

The following feature are selected as the representative for the potential driver of LGD in this data set:

- The credit worthiness variables:
  application score card variables selected in Part b)

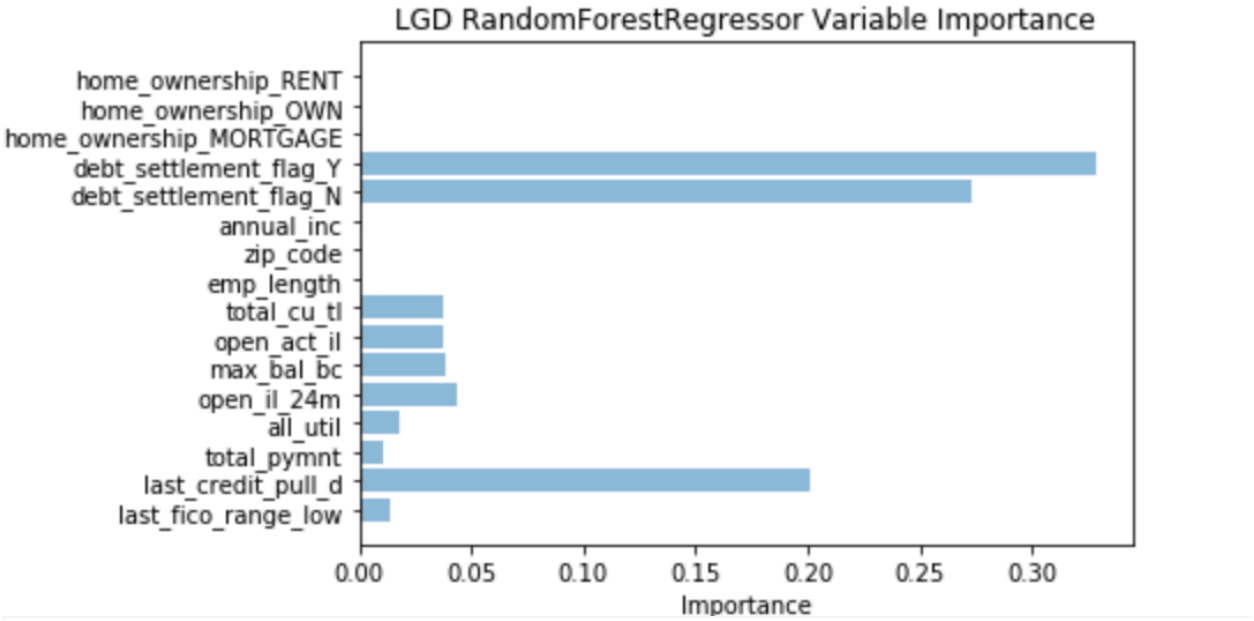
- Salary:
  'annual_inc'

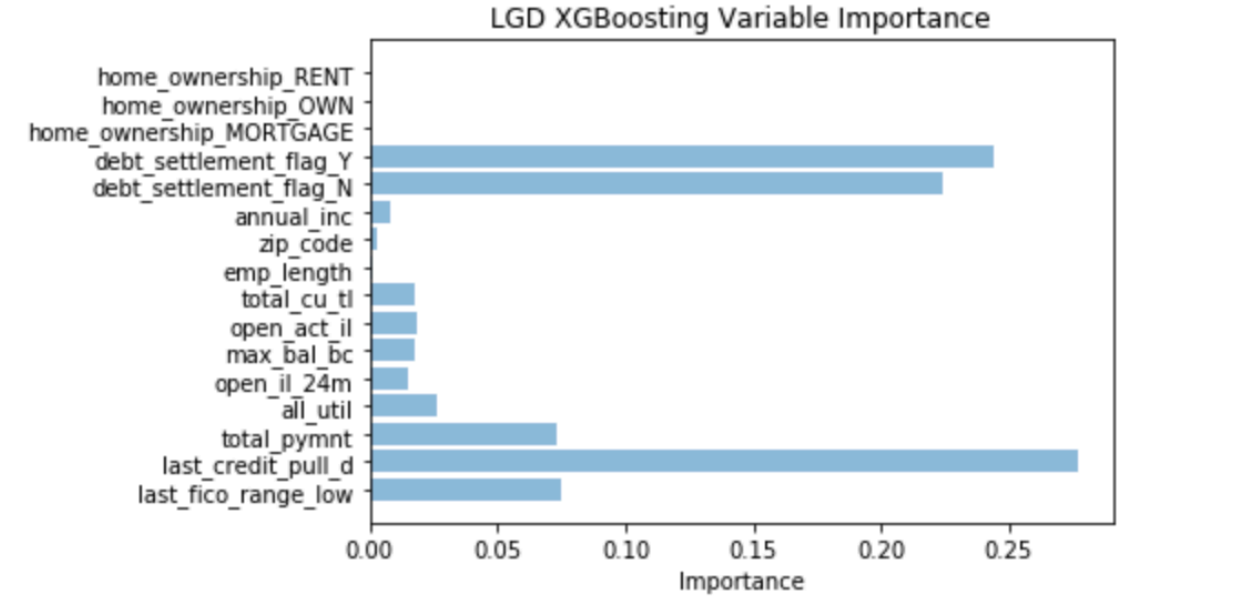
- Time at job:
  ‘emp_length'

- Partial Information related to EAD & recovery :
  ‘total_pymnt' & 'debt_settlement_flag'

- Geographic information:
  ‘zip_code’ (transformed to first letter)

Feature Importance Comparison:





By comparing the two Variable Importance plots, we can see that the both model ranked "debt_settlement_flag" and "last_credit_pull_d" as the top two most useful predictor for LGD. This is most likely because that "debt_settlement_flag" is closely related to recoveries, if applicant debt_settlement agrees to do debt settlements, the recovery rate will most likely be higher than those who don't. The year of "last_credit_pull_d" is important maybe because applicants who applied loan in some years (most likely 2017 ~ 2018, reason explained in part b) seems have difficulties to repay the loan. On the other hand, XGBoosting thinks FICO score and total payment is more relevant to LGD than the rest of predictors, while the Random Forest thinks the opposite. I can't judge their decisions on these features without evaluating their performance:

Model Performance:

	MSE	R^2	Adjusted R^2
Random Forest Regressor	0.8807%	29.2983%	29.2790%
XGBoosting Regressor	0.7509%	39.7196%	39.7032%

From the above accuracy metrics, it looks like XGBoosting outperformed the Random Forest. XGBoosting's model has less mean squared error, and the model explains 10% more variation of the LGD than the Random Forest Model. One reason to explain this is that the training set of the defaulted cases is relatively small (about 100,000), and XGBoosting model outperforms Random Forest on small training sets. Random Forest needs relatively large training set (maybe 1,000,000) in order to get more accurate prediction.

Back to the judgement on the feature importance left from last section, it seems XGBoosting's evaluation on those features are more accurate. (FICO & total payment matters more)

In general, both models are not quite ideal in terms of performance of prediction since the adjusted R^2 are less than 50%. This might be caused by my choices of variables, transformation of those variables, and hyper parameters entered into the models. This requires some further investigations on those choices.

Apply the Models on non-Defaulted Cases

	Random Forest Regressor	XG Boosting
Average LGD estimated on non-defaulted loans	92.9614%	90.1233%

Although my models are not quite ideal in terms of adjusted R^2, the average LGD estimate on the non-defaulted cases seems reasonable. Recall that LGD for all unsecured loans in Coursework 1 were considered 100% when we used F-IRB Approach, so these unsecured LC loans should also be expected to have LGDs close to 100%. In that sense, the estimations of LGD on non-defaulted loans by both models seems in the correct track. Because XGBoosting model is more accurate than the Random Forest in this case, XGBoosting's estimation should be preferred, and this should reduce the expected loss on those non-defaulted loans for Loan Club.

Words: 2201

Appendix A

Application Score Card for Loan Club (Total 500 Points)		
Variables & Range		**Points**
Base Points		166
inq_fi	Empty Record	-2
	0 ~ 2	0
	more than 2	2
last_credit_pull_d	During year 2017 ~ 2018	-76
	Before year 2017	-71
	After year 2018	21
last_fico_range_low	< 580	-206
	580 ~ 640	-92
	640 ~ 680	67
	> 680	214
max_bal_bc	> 7000	-35
	< 7000	-16
	Empty Record	29
open_act_il	0 ~ 2	2
	> 2	1
	Empty Record	-2
open_il_24m	0 ~ 2	19
	2 ~ 3	8
	> 3	-4
	Empty Record	-17

Application Score Card for Loan Club (Total 500 Points)		
open_rv_24m	0 ~ 1	3
	1 ~ 3	2
	> 3	0
	Empty Record	-2
total_cu_tl	0 ~ 2	10
	2 ~ 3	8
	3 ~ 7	7
	Empty Record	-13

Appendix B

Part I & II https://colab.research.google.com/drive/1TzzZciYVds_JyGNTstfE82Pe7he0E3P0
Part III https://colab.research.google.com/drive/1WwVhkoyVTgOyf_GARuucu1otgpZofuW0

Part I Understanding Data & Data Processing

Download the Data

```
!gdown https://drive.google.com/uc?id=1HAItdPa8TP-09OLSQ-Z5z3T_UbMp5mrt
!unzip '/content/Coursework 2 - Lending Club Data.zip'
```

```
Downloading...
From: https://drive.google.com/uc?id=1HAItdPa8TP-09OLSQ-Z5z3T_UbMp5mrt
To: /content/Coursework 2 - Lending Club Data.zip
369MB [00:03, 102MB/s]
Archive:  /content/Coursework 2 - Lending Club Data.zip
  inflating: LCFinal.csv
  inflating: Variable Dictionary.docx
```

```
## checking if file is correct
!head LCFinal.csv
```

```
,id,member_id,loan_amnt,funded_amnt,funded_amnt_inv,term,int_rate,installment,emp_title,em
0,68407277,,3600.0,3600.0,3600.0, 36 months,13.99,123.03,leadman,10+ years,MORTGAGE,55000.
1,68355089,,24700.0,24700.0,24700.0, 36 months,11.99,820.28,Engineer,10+ years,MORTGAGE,65
2,68341763,,20000.0,20000.0,20000.0, 60 months,10.78,432.66,truck driver,10+ years,MORTGAG
3,66310712,,35000.0,35000.0,35000.0, 60 months,14.85,829.9,Information Systems Officer,10+
4,68476807,,10400.0,10400.0,10400.0, 60 months,22.45,289.91,Contract Specialist,3 years,MO
5,68426831,,11950.0,11950.0,11950.0, 36 months,13.44,405.18,Veterinary Tecnician,4 years,R
6,68476668,,20000.0,20000.0,20000.0, 36 months,9.17,637.58,Vice President of Recruiting Op
7,67275481,,20000.0,20000.0,20000.0, 36 months,8.49,631.26,road driver,10+ years,MORTGAGE,
8,68466926,,10000.0,10000.0,10000.0, 36 months,6.49,306.45,SERVICE MANAGER,6 years,RENT,85
```

Initializing all the packages for doing data processing

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
Loan_data = pd.read_csv('LCFinal.csv')
```

```
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2718: DtypeWarning
  interactivity=interactivity, compiler=compiler, result=result)
```

```
#Checking the data size
Loan_data.shape
```

```
(2260701, 148)
```

Clean from from first impression

```
#We delete useless rows where loan amount is Null

#Find the subset of the Loan where loan amount is Null
nan_rows = Loan_data[Loan_data['loan_amnt'].isnull()]
#Drop those rows
Loan_data = Loan_data.drop(nan_rows.index)


#Drop the irrelevent variables for credit scoring and LGD Modelling
#from our first impression


#The drop list are irrelevent text columns so drop them from the beginning
drop_list_round1 = ['id','member_id', 'url', 'desc', 'emp_title', 'issue_d',\
                    'last_pymnt_d', 'next_pymnt_d', 'initial_list_status',
                    'verification_status_joint', 'disbursement_method']
Loan_data = Loan_data.drop(drop_list_round1, axis = 1)
Loan_data = Loan_data.drop(Loan_data.columns[0], axis = 1)


#Modify Zip Code
Loan_data.loc[Loan_data['zip_code'] != np.nan, 'zip_code'] = \
Loan_data.loc[Loan_data['zip_code'] != np.nan, 'zip_code'].str[0]

len(list(Loan_data['zip_code'].unique()))
```

```
11
```

```
len(list(Loan_data['addr_state'].unique()))
```

```
51
```

Because the State values are far less than zip_code, so we can drop the zip code while they both represents addr

```
Loan_data = Loan_data.drop('addr_state', axis = 1)


Loan_data.shape
```

```
(2260668, 135)
```

Constructing our objective variables

Scorecard Objective

```
#For Credit Scoring our objective variable is loan_status
#For LGD Modelling, we only cosider defaulted cases which means that loan_status
#is "charged_off"

#First we check how many unique value are in the objective variable:
Loan_data['loan_status'].value_counts()
```

```
Fully Paid                                             1076751
Current                                                 878317
Charged Off                                             268559
Late (31-120 days)                                       21467
In Grace Period                                           8436
Late (16-30 days)                                         4349
Does not meet the credit policy. Status:Fully Paid        1988
Does not meet the credit policy. Status:Charged Off        761
Default                                                     40
Name: loan_status, dtype: int64
```

Because we only want binary results "Good" & "Bad" for Credit Scoring, the main groups matches "Good" and "Ba are [Fully paid] and [Charged Off].

We also have the cases where is status is "Current", "Late","In Grace Period","default".

For "Current", "In Grace Period", "Late (16-30 days)" We can still consider it as "Good" cases because the PD for those cases are really low, and they do not belong to any defination of default we have learned in this course. As "Late (31-120 days)" & "Default", I condisder it as "Bad" Cases because the PD is pretty high in these cases

As for the rest of the groups "Does not meet the credit policy. Status:Fully Paid", "Does not meet the credit policy. Status:Charged Off", since they will not be allowed to enter application anymore, we can just discard them. Besid it kind of revealed Loan Club's credit scoring system, which is not desired.

```
##Now we map all those values into "0" (good) and "1" (bad)

Loan_data.loc[Loan_data['loan_status'] == 'Fully Paid', 'loan_status'] = 0
Loan_data.loc[Loan_data['loan_status'] == 'Current', 'loan_status'] = 0
Loan_data.loc[Loan_data['loan_status'] == 'In Grace Period', 'loan_status'] = 0
Loan_data.loc[Loan_data['loan_status'] == 'Late (16-30 days)','loan_status'] = 0


Loan_data.loc[Loan_data['loan_status'] == 'Late (31-120 days)','loan_status']=1
Loan_data.loc[Loan_data['loan_status'] == 'Charged Off','loan_status']=1
Loan_data.loc[Loan_data['loan_status'] == 'Default','loan_status']=1



#And Discard those ones which are out of policy
drop_out_policy1 = Loan_data.loc[Loan_data['loan_status'] ==
                'Does not meet the credit policy. Status:Fully Paid'].index

drop_out_policy2 = Loan_data.loc[Loan_data['loan_status'] ==
                'Does not meet the credit policy. Status:Charged Off'].index

Loan_data = Loan_data.drop(drop_out_policy1, axis = 0)
Loan_data = Loan_data.drop(drop_out_policy2, axis = 0)
```

```
Loan_data['loan_status'].value_counts()
```

```
0    1967853
1     290066
Name: loan_status, dtype: int64
```

```
Lenth_default = 290066
```

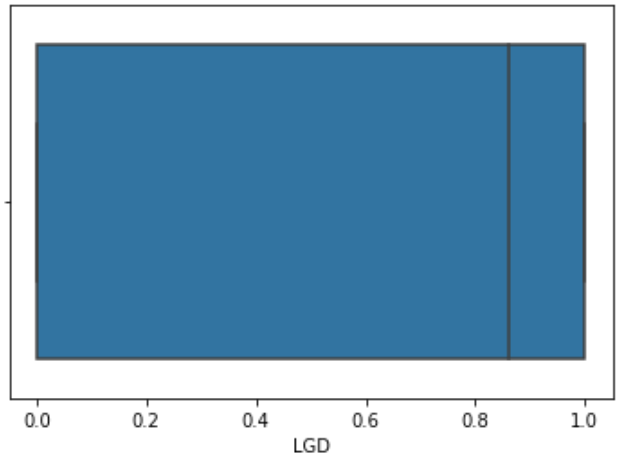
LGD Variable

```
Loan_data['LGD'] = 1- ((Loan_data['recoveries'] + Loan_data['total_rec_late_fee'] - \
                        Loan_data['collection_recovery_fee']) \
                       / (Loan_data['funded_amnt']  - Loan_data['total_rec_prncp']))
```

```
Loan_data['LGD'].fillna(0, inplace = True)
## Add lower and upper cap 0 and 1 respectively
Loan_data.loc[Loan_data['LGD'] < 0, 'LGD'] = 0
Loan_data.loc[Loan_data['LGD'] > 1, 'LGD'] = 1
```

```
## Check transformation is coorect
sns.boxplot(Loan_data['LGD'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5f0d5a1898>
```




```
## Since our LGD is constructed by the following variables
## we have to drop them to avoid collieaity

Loan_data = Loan_data.drop(['recoveries', 'total_rec_late_fee', \
                            'collection_recovery_fee', 'funded_amnt',
                            'total_rec_prncp'], axis = 1)
```

Dealing with Missing Data

```
nullset = Loan_data.isnull()
null_counts = Loan_data.isnull().sum() / len(Loan_data)
print("percentage of null values in each column:\n{}".format(null_counts.head(143)))
```

```
percentage of null values in each column:
loan_amnt                 0.000000
funded_amnt_inv           0.000000
term                      0.000000
int_rate                  0.000000
installment               0.000000
                             ...
settlement_date           0.984837
settlement_amount         0.984837
settlement_percentage     0.984837
settlement_term           0.984837
LGD                       0.000000
Length: 131, dtype: float64
```

```
Missing_columns = list(null_counts.loc[null_counts.head(143) > 0].index)
len(Missing_columns)

#So we have 99 columns that have missing values
```

```
99
```

Dealing those variables that have too many missing values with careful investigation

Our principle is: "Delete - Keep - Replace"

```
Missing_column_Big_o = list(null_counts.loc[null_counts.head(137) > 0.9].index)
Missing_column_Big = Missing_column_Big_o
Missing_column_Big
```

```
['annual_inc_joint',
 'dti_joint',
 'revol_bal_joint',
 'sec_app_fico_range_low',
 'sec_app_fico_range_high',
 'sec_app_earliest_cr_line',
 'sec_app_inq_last_6mths',
 'sec_app_mort_acc',
 'sec_app_open_acc',
 'sec_app_revol_util',
 'sec_app_open_act_il',
 'sec_app_num_rev_accts',
 'sec_app_chargeoff_within_12_mths',
 'sec_app_collections_12_mths_ex_med',
 'sec_app_mths_since_last_major_derog',
 'hardship_type',
 'hardship_reason',
 'hardship_status',
 'deferral_term',
 'hardship_amount',
 'hardship_start_date',
 'hardship_end_date',
 'payment_plan_start_date',
 'hardship_length',
 'hardship_dpd',
 'hardship_loan_status',
 'orig_projected_additional_accrued_interest',
 'hardship_payoff_balance_amount',
 'hardship_last_payment_amount',
 'debt_settlement_flag_date',
 'settlement_status',
 'settlement_date',
 'settlement_amount',
 'settlement_percentage',
 'settlement_term']
```

Notice that most of those massive missing columns are clustering with key words 'settlement', 'hardship' and 'sec_app'.

Dealing with sec_app group

"Sec_app" have too many missing values, Therefore it is useless to be analyzed

Because 'sec_app' variables would have values if and only if the applicant has a secondary applicant. From the da we can see 98% of applicant doesn't have a sec_applicant, therefore it is useless to analyze those values even if some of them are closely related to PD and LGD.

Conclusion: drop them all

```
drop_list_round2 = ['sec_app_fico_range_low',
 'sec_app_fico_range_high',
 'sec_app_earliest_cr_line',
 'sec_app_inq_last_6mths',
 'sec_app_mort_acc',
 'sec_app_open_acc',
 'sec_app_revol_util',
 'sec_app_open_act_il'
```

```
 'sec_app_open_act_il',
 'sec_app_num_rev_accts',
 'sec_app_mths_since_last_major_derog',
 'sec_app_collections_12_mths_ex_med',
 'sec_app_chargeoff_within_12_mths']
Loan_data = Loan_data.drop(drop_list_round2, axis = 1)


Loan_data.shape
```

```
(2257919, 119)
```

Dealing with hardship

Similar to "Sec_app", "hardship" variables are clustering. Which means we only need to keep the most useful one indicate the exsitence of hardship and the information that variable contains. NaN means the hardship does not exist.

And this information is only probably useful in LGD

In this case, I choose 'hardship_amnt' as the representive variable for 'hard_ship" group and drop the rest

```
Loan_data['hardship_amount'].value_counts()
```

```
69.90     5
48.56     5
53.05     5
132.33    5
94.59     5
          ..
494.35    1
62.04     1
62.21     1
21.22     1
243.11    1
Name: hardship_amount, Length: 9162, dtype: int64
```

```
Loan_data[Loan_data['hardship_amount'].isin([0])]

##Since there is no value "0" in 'hardship_amount', we can use 0 to fill NaNs
##to indicate those rows with no hardship

Loan_data['hardship_amount'].fillna(0, inplace = True)

## Now we can drop the rest of the "hard_ship" cluster

drop_list_round3 = ['hardship_type','hardship_reason','hardship_status',
 'deferral_term','hardship_start_date','hardship_end_date',
 'payment_plan_start_date','hardship_length','hardship_dpd',
 'hardship_loan_status','orig_projected_additional_accrued_interest',
 'hardship_payoff_balance_amount','hardship_last_payment_amount',]

Loan_data = Loan_data.drop(drop_list_round3, axis = 1)


Missing_column_Big = list(set(Missing_column_Big)\
                          - set(['hardship_amount']))
```

Deal with Settlement

Since settlement clusters are included in the recoverie and settlements obervations are missing more than 90% o values. Although they are related to LGD, but they only accounts 10% of the defaulted cases. I decide to drop ther

```
Loan_data['settlement_amount'].notnull().sum() / Lenth_default

## They only accounts 10% of defaulted cases
```

```
0.11802831079823213
```

```
Settlement = ['settlement_status',
 'settlement_date',
 'settlement_amount',
 'settlement_percentage',
 'settlement_term',
 'debt_settlement_flag_date']

Loan_data = Loan_data.drop(Settlement, axis = 1)

Missing_column_Big = list(set(Missing_column_Big) - set(Settlement))


removed = drop_list_round2 + drop_list_round3
Missing_column_Big = list(set(Missing_column_Big) - set(removed))
Missing_column_Big
```

```
['annual_inc_joint', 'revol_bal_joint', 'dti_joint']
```

Dealing With Joint Application Type

Related Variables: 'revol_bal_joint', 'annual_inc_joint', 'dti_joint', 'verification_status_joint'

Notice that all the variables are related to joint account, and I found that if these variables are NaN, then it means individual application. Since these variables have less than 10% of data, I decide to drop them all

```
drop_list_round4 = ['revol_bal_joint', 'annual_inc_joint', 'dti_joint']
Loan_data = Loan_data.drop(drop_list_round4, axis = 1)


Missing_column_Big = list(set(Missing_column_Big) - set(drop_list_round4))
Missing_column_Big
```

```
[]
```

Now we have finished clean data that has greater than 90% of the data, Lets check medium amount of missing values:

```
null_counts2 = Loan_data.isnull().sum() / len(Loan_data)
Missing_column_Medium = list(null_counts2.loc[null_counts2.head(137) > 0.5].index)
Missing_column_Medium
```

```
['mths_since_last_delinq',
 'mths_since_last_record',
 'mths_since_last_major_derog',
 'mths_since_recent_bc_dlq',
 'mths_since_recent_revol_delinq']
```

Dealing With important indictor of PD & LGD

['mths_since_recent_revol_delinq','mths_since_last_major_derog','mths_since_recent_bc_dlq','mths_since_last_del
'mths_since_last_record']

They are important indicator of PD and LGD, NaN means no record or never had bad records recently. so we try to
NaN Values with something out of range, I used -1 to represent this information, and treat them as NaN

```
Loan_data.loc[Loan_data['mths_since_recent_revol_delinq'].isnull() == True, 'mths_since_recent_
```

```
Loan_data.loc[Loan_data['mths_since_last_major_derog'].isnull() == True, 'mths_since_last_major
```

```
Loan_data.loc[Loan_data['mths_since_recent_bc_dlq'].isnull() == True, 'mths_since_recent_bc_dlq
```

```
Loan_data.loc[Loan_data['mths_since_last_delinq'].isnull() == True, 'mths_since_last_delinq'] =
```

```
Loan_data.loc[Loan_data['mths_since_last_record'].isnull() == True,  'mths_since_last_record']
```

```
bad_record =  ['mths_since_recent_revol_delinq','mths_since_last_major_derog','mths_since_recen
Missing_column_Medium = list(set(Missing_column_Big) - set(bad_record))
Missing_column_Medium
```

```
[]
```

Dealling with small portion of the data are missing

```
Missing_column_Small = list(null_counts2.loc[null_counts2.head(137) <= 0.5].index)
Missing_column_Small

for i in Missing_column_Small:
  print (str(i) + "       " + str(Loan_data[i].isnull().sum() / len(Loan_data)))
```

```
loan_amnt       0.0
funded_amnt_inv       0.0
term       0.0
int_rate       0.0
installment       0.0
emp_length       0.06504794901854317
home_ownership       0.0
annual_inc       0.0
loan_status       0.0
pymnt_plan       0.0
purpose       0.0
title       0.010329422800375035
zip_code       4.428856836759866e-07
dti       0.0007577774047696131
delinq_2yrs       0.0
earliest_cr_line       0.0
fico_range_low       0.0
fico_range_high       0.0
inq_last_6mths       4.428856836759866e-07
open_acc       0.0
pub_rec       0.0
revol_bal       0.0
revol_util       0.0007803645746370884
total_acc       0.0
out_prncp       0.0
out_prncp_inv       0.0
total_pymnt       0.0
total_pymnt_inv       0.0
total_rec_int       0.0
last_pymnt_amnt       0.0
last_credit_pull_d       3.100199785731906e-05
last_fico_range_high       0.0
last_fico_range_low       0.0
collections_12_mths_ex_med       2.480159828585525e-05
application_type       0.0
acc_now_delinq       0.0
tot_coll_amt       0.029906741561588346
tot_cur_bal       0.029906741561588346
open_acc_6m       0.382379084457857
open_act_il       0.38237864157217333
open_il_12m       0.38237864157217333
open_il_24m       0.38237864157217333
mths_since_rcnt_il       0.40177482008876314
total_bal_il       0.38237864157217333
il_util       0.472160870252653
open_rv_12m       0.38237864157217333
open_rv_24m       0.38237864157217333
max_bal_bc       0.38237864157217333
all_util       0.38247563353689834
total_rev_hi_lim       0.029906741561588346
inq_fi       0.38237864157217333
total_cu_tl       0.382379084457857
inq_last_12m       0.382379084457857
acc_open_past_24mths       0.020940078009884323
avg_cur_bal       0.029937743559445666
bc_open_to_buy       0.03197014596183477
bc_util       0.03247326409849069
chargeoff_within_12_mths       2.480159828585525e-05
delinq_amnt       0.0
mo_sin_old_il_acct       0.06037506217007785
mo_sin_old_rev_tl_op       0.029907184447272025
mo_sin_rcnt_rev_tl_op       0.029907184447272025
mo_sin_rcnt_tl       0.029906741561588346
mort_acc       0.020940078009884323
```

```
mths_since_recent_bc        0.03129563106559624
mths_since_recent_inq        0.1296264392123898
num_accts_ever_120_pd        0.029906741561588346
num_actv_bc_tl       0.029906741561588346
num_actv_rev_tl       0.029906741561588346
num_bc_sats       0.024731179462150768
num_bc_tl       0.029906741561588346
num_il_tl       0.029906741561588346
num_op_rev_tl       0.029906741561588346
num_rev_accts       0.029907184447272025
num_rev_tl_bal_gt_0       0.029906741561588346
num_sats       0.024731179462150768
num_tl_120dpd_2m       0.06683499275217579
num_tl_30dpd       0.029906741561588346
num_tl_90g_dpd_24m       0.029906741561588346
num_tl_op_past_12m       0.029906741561588346
pct_tl_nvr_dlq       0.029975388842558126
percent_bc_gt_75       0.032166787205386904
pub_rec_bankruptcies       0.00030869132152216267
tax_liens       1.7272541663363478e-05
tot_hi_cred_lim       0.029906741561588346
total_bal_ex_mort       0.020940078009884323
total_bc_limit       0.020940078009884323
total_il_high_credit_limit       0.029906741561588346
hardship_flag       0.0
hardship_amount       0.0
debt_settlement_flag       0.0
LGD       0.0
```

```
Loan_data.shape
```

```
(2257919, 97)
```

Replace all missing cells is the missing portion is smaller than 3%

```
Missing_column_Small = list(null_counts2.loc[(null_counts2.head(115) < 0.5) &
                                             (null_counts2.head(115) > 0)].index)
Missing_column_small_o = list(null_counts2.loc[(null_counts2.head(115) < 0.03) &
                                               (null_counts2.head(115) > 0)].index)
Missing_column_small_o
```

```
     ['title',
      'zip_code',
      'dti',
      'inq_last_6mths',
      'revol_util',
      'last_credit_pull_d',
      'collections_12_mths_ex_med',
      'tot_coll_amt',
      'tot_cur_bal',
      'total_rev_hi_lim',
      'acc_open_past_24mths',
      'avg_cur_bal',
      'chargeoff_within_12_mths',
      'mo_sin_old_rev_tl_op',
      'mo_sin_rcnt_rev_tl_op',
      'mo_sin_rcnt_tl',
      'mort_acc',
      'num_accts_ever_120_pd',
      'num_actv_bc_tl',
      'num_actv_rev_tl',
      'num_bc_sats',
      'num_bc_tl',
      'num_il_tl',
      'num_op_rev_tl',
      'num_rev_accts',
      'num_rev_tl_bal_gt_0',
      'num_sats',
      'num_tl_30dpd',
      'num_tl_90g_dpd_24m',
      'num_tl_op_past_12m',
      'pct_tl_nvr_dlq',
      'pub_rec_bankruptcies',
      'tax_liens',
      'tot_hi_cred_lim',
      'total_bal_ex_mort',
      'total_bc_limit',
      'total_il_high_credit_limit']


for i in Missing_column_small_o:
  if Loan_data[i].dtype == 'float64':
    median = Loan_data[i].median()
    Loan_data[i].fillna(median, inplace=True)
  elif Loan_data[i].dtype == 'O':
    mode = Loan_data[i].mode()[0]
    Loan_data[i].fillna(mode, inplace=True)


## Update the miss_value list
Missing_column_Small = list(set(Missing_column_Small) - set(Missing_column_small_o))

for i in Missing_column_Small:
  print (str(i) + "       " + str(Loan_data[i].isnull().sum() / len(Loan_data)))
```

```
emp_length       0.06504794901854317
max_bal_bc       0.38237864157217333
all_util        0.38247563353689834
inq_fi        0.38237864157217333
bc_open_to_buy       0.03197014596183477
total_bal_il       0.38237864157217333
bc_util        0.03247326409849069
open_rv_12m       0.38237864157217333
open_rv_24m       0.38237864157217333
open_act_il       0.38237864157217333
total_cu_tl       0.382379084457857
open_il_12m       0.38237864157217333
open_acc_6m       0.382379084457857
mo_sin_old_il_acct       0.06037506217007785
percent_bc_gt_75       0.032166787205386904
mths_since_recent_bc       0.03129563106559624
open_il_24m       0.38237864157217333
il_util       0.472160870252653
mths_since_rcnt_il       0.40177482008876314
inq_last_12m       0.382379084457857
num_tl_120dpd_2m       0.06683499275217579
mths_since_recent_inq       0.1296264392123898
```

These columns seems pretty important, So we will test correlation among them and hopefully we can eliminate down to a narrower list

Correlation Analysis & Filtering

Our basic principle is to drop variables that has a correlation which has greater than 70%

```
Loan_data.shape
```

```
(2257919, 97)
```

Now we split the variables into three parts to further reduce variables

```
##Define a function to sort correlation
##Get diagonal and lower triangular pairs of correlation matrix
def get_redundant_pairs(df):
    pairs_to_drop = set()
    cols = df.columns
    for i in range(0, df.shape[1]):
        for j in range(0, i+1):
            pairs_to_drop.add((cols[i], cols[j]))
    return pairs_to_drop

## Get

def get_top_abs_correlations(df, n=100):
    au_corr = df.corr().abs().unstack()
    labels_to_drop = get_redundant_pairs(df)
    au_corr = au_corr.drop(labels=labels_to_drop).sort_values(ascending=False)
    return au_corr[0:n]
```

```
min_obs = int(0.5 * len(Loan_data))
min_obs
```

```
1128959
```

```
%%script false
df_part1 = Loan_data.iloc[:,0:35]
corelation_part1 = df_part1.corr(method = 'spearman', min_periods = min_obs)
```

```
%%script false
get_top_abs_correlations(corelation_part1, 35)
```

```
fico_range_low          fico_range_high         1.000000
last_fico_range_high    last_fico_range_low     1.000000
out_prncp               out_prncp_inv           1.000000
total_pymnt             total_pymnt_inv         0.999999
loan_amnt               funded_amnt_inv         0.999996
mths_since_last_record  pub_rec                 0.999788
loan_amnt               installment             0.996860
funded_amnt_inv         installment             0.996794
open_acc                total_acc               0.921634
total_pymnt             last_pymnt_amnt         0.882018
total_pymnt_inv         last_pymnt_amnt         0.881919
installment             total_rec_int           0.875587
loan_amnt               total_rec_int           0.859182
funded_amnt_inv         total_rec_int           0.858857
total_pymnt_inv         total_rec_int           0.847995
total_pymnt             total_rec_int           0.847633
installment             total_pymnt_inv         0.827813
                        total_pymnt             0.827387
loan_amnt               total_pymnt_inv         0.806098
                        total_pymnt             0.805657
funded_amnt_inv         total_pymnt_inv         0.804972
                        total_pymnt             0.804527
int_rate                fico_range_low          0.787472
                        fico_range_high         0.787472
                        last_fico_range_low     0.785806
                        last_fico_range_high    0.785805
fico_range_high         last_fico_range_low     0.739816
fico_range_low          last_fico_range_low     0.739816
fico_range_high         last_fico_range_high    0.739815
fico_range_low          last_fico_range_high    0.739815
installment             last_pymnt_amnt         0.693473
loan_amnt               annual_inc              0.688636
funded_amnt_inv         annual_inc              0.688375
installment             annual_inc              0.680000
fico_range_high         revol_util              0.668949
dtype: float64
```

```
##From our first round correlation filtering, we drop the following variables
new_drop_list1 = ['last_fico_range_high', 'fico_range_high', 'out_prncp_inv',
                  'funded_amnt_inv', 'open_acc', 'total_pymnt_inv',
                  'pub_rec', 'mths_since_last_delinq', 'last_pymnt_amnt',
                  'fico_range_low']
```

```
df_part2 = Loan_data.iloc[:,35:71]
corelation_part2 = df_part2.corr(method = 'spearman', min_periods = min_obs)
```

```
corelation_part2 = df_part2.corr(method = 'spearman', min_periods = min_obs)
```

```
get_top_abs_correlations(corelation_part2, 60)
```

```
open_acc_6m                 mo_sin_rcnt_tl                  0.994186
open_il_12m                 mths_since_rcnt_il              0.992825
open_rv_12m                 mo_sin_rcnt_rev_tl_op           0.987563
                            open_rv_24m                     0.971694
mths_since_recent_bc_dlq    mths_since_recent_revol_delinq  0.970329
open_il_24m                 mths_since_rcnt_il              0.969903
open_rv_12m                 mths_since_recent_bc            0.966719
acc_now_delinq              delinq_amnt                     0.962148
mo_sin_rcnt_rev_tl_op       mths_since_recent_bc            0.961657
open_il_12m                 open_il_24m                     0.957778
tot_cur_bal                 avg_cur_bal                     0.956411
open_act_il                 total_bal_il                    0.950963
open_rv_24m                 mths_since_recent_bc            0.949524
                            mo_sin_rcnt_rev_tl_op           0.947482
inq_last_12m                mths_since_recent_inq           0.929091
mo_sin_rcnt_rev_tl_op       mo_sin_rcnt_tl                  0.919409
open_acc_6m                 mo_sin_rcnt_rev_tl_op           0.915444
acc_open_past_24mths        mo_sin_rcnt_tl                  0.912757
open_acc_6m                 acc_open_past_24mths            0.905850
open_rv_24m                 acc_open_past_24mths            0.895823
open_rv_12m                 mo_sin_rcnt_tl                  0.884460
open_acc_6m                 open_rv_12m                     0.882960
open_rv_12m                 acc_open_past_24mths            0.865700
acc_open_past_24mths        mo_sin_rcnt_rev_tl_op           0.857862
open_acc_6m                 mths_since_recent_bc            0.857690
mo_sin_rcnt_tl              mths_since_recent_bc            0.855804
tot_cur_bal                 mort_acc                        0.851924
total_rev_hi_lim            bc_open_to_buy                  0.848409
acc_open_past_24mths        mths_since_recent_bc            0.846199
avg_cur_bal                 mort_acc                        0.846062
bc_open_to_buy              bc_util                         0.840009
open_rv_24m                 mo_sin_rcnt_tl                  0.837288
open_acc_6m                 open_rv_24m                     0.835597
open_il_24m                 total_bal_il                    0.829476
open_act_il                 open_il_24m                     0.807513
inq_fi                      inq_last_12m                    0.798154
inq_last_12m                acc_open_past_24mths            0.793465
                            mo_sin_rcnt_tl                  0.786707
mths_since_rcnt_il          total_bal_il                    0.783872
open_acc_6m                 inq_last_12m                    0.777814
mths_since_rcnt_il          il_util                         0.766011
open_il_12m                 total_bal_il                    0.764385
                            il_util                         0.753689
open_il_24m                 inq_fi                          0.749850
all_util                    bc_open_to_buy                  0.748916
mths_since_last_major_derog mths_since_recent_revol_delinq  0.743661
open_act_il                 mths_since_rcnt_il              0.743168
all_util                    bc_util                         0.740071
mths_since_last_major_derog mths_since_recent_bc_dlq        0.728409
open_act_il                 open_il_12m                     0.727860
open_il_24m                 il_util                         0.727566
mths_since_rcnt_il          inq_fi                          0.726104
open_il_12m                 inq_fi                          0.718411
                            inq_last_12m                    0.707806
open_il_24m                 acc_open_past_24mths            0.707209
mths_since_rcnt_il          acc_open_past_24mths            0.705046
mo_sin_rcnt_tl              mths_since_recent_inq           0.704574
open_acc_6m                 mths_since_recent_inq           0.702635
mths_since_rcnt_il          inq_last_12m                    0.701969
open_il_12m                 acc_open_past_24mths            0.692668
dtype: float64
```

```
new_drop_list2 = ['open_acc_6m','open_il_12m', 'mths_since_rcnt_il',
                  'mths_since_recent_bc_dlq', 'open_rv_12m',
                  'mths_since_recent_bc','avg_cur_bal','total_bal_il',
                  'inq_last_12m','bc_open_to_buy','inq_last_12m','bc_util']



df_part3 = Loan_data.iloc[:,71:106]
corelation_part3 = df_part3.corr(method = 'spearman', min_periods = min_obs)



get_top_abs_correlations(corelation_part3, 60)
```

```
num_actv_rev_tl        num_rev_tl_bal_gt_0           0.999921
num_actv_bc_tl         num_bc_sats                   0.962183
num_bc_tl              num_rev_accts                 0.958566
num_actv_bc_tl         num_actv_rev_tl               0.947868
                       num_rev_tl_bal_gt_0           0.946914
total_bal_ex_mort      total_il_high_credit_limit    0.944701
num_op_rev_tl          num_rev_tl_bal_gt_0           0.942961
num_actv_rev_tl        num_op_rev_tl                 0.942959
num_op_rev_tl          num_rev_accts                 0.940419
num_bc_sats            num_op_rev_tl                 0.936511
                       num_bc_tl                     0.928353
num_op_rev_tl          num_sats                      0.913517
num_bc_tl              num_op_rev_tl                 0.905561
num_accts_ever_120_pd  pct_tl_nvr_dlq                0.904416
num_actv_bc_tl         num_op_rev_tl                 0.903647
num_il_tl              total_il_high_credit_limit    0.901092
num_actv_rev_tl        num_bc_sats                   0.891790
num_bc_sats            num_rev_tl_bal_gt_0           0.890598
num_rev_accts          num_sats                      0.876041
num_bc_sats            num_rev_accts                 0.863959
                       total_bc_limit                0.863128
num_actv_bc_tl         num_bc_tl                     0.851288
tot_hi_cred_lim        total_bal_ex_mort             0.838409
num_rev_tl_bal_gt_0    num_sats                      0.836313
num_actv_rev_tl        num_sats                      0.835118
                       num_rev_accts                 0.833206
num_rev_accts          num_rev_tl_bal_gt_0           0.833082
num_bc_sats            num_sats                      0.832451
num_bc_tl              num_sats                      0.818572
num_il_tl              total_bal_ex_mort             0.816096
num_actv_rev_tl        num_bc_tl                     0.806848
num_bc_tl              num_rev_tl_bal_gt_0           0.805956
                       total_bc_limit                0.793936
num_actv_bc_tl         num_rev_accts                 0.791600
                       num_sats                      0.783480
                       total_bc_limit                0.782643
tot_hi_cred_lim        total_il_high_credit_limit    0.736338
num_op_rev_tl          total_bc_limit                0.730624
num_sats               total_bc_limit                0.710774
num_rev_accts          total_bc_limit                0.686056
num_rev_tl_bal_gt_0    total_bc_limit                0.642931
num_actv_rev_tl        total_bc_limit                0.642510
num_il_tl              tot_hi_cred_lim               0.619662
num_accts_ever_120_pd  num_tl_90g_dpd_24m            0.619289
num_tl_90g_dpd_24m     pct_tl_nvr_dlq                0.592131
num_sats               tot_hi_cred_lim               0.546430
                       num_tl_op_past_12m            0.533584
tot_hi_cred_lim        total_bc_limit                0.517900
num_sats               total_bal_ex_mort             0.513279
num_rev_accts          num_tl_op_past_12m            0.472998
num_op_rev_tl          num_tl_op_past_12m            0.469542
num_accts_ever_120_pd  total_bc_limit                0.404237
num_bc_tl              num_tl_op_past_12m            0.389042
pub_rec_bankruptcies   tot_hi_cred_lim               0.385332
percent_bc_gt_75       total_bc_limit                0.373257
num_actv_rev_tl        num_tl_op_past_12m            0.373220
num_rev_tl_bal_gt_0    num_tl_op_past_12m            0.371063
pub_rec_bankruptcies   total_bc_limit                0.370031
pct_tl_nvr_dlq         total_bc_limit                0.366275
num_tl_op_past_12m     percent_bc_gt_75              0.361552
dtype: float64
```

```
new_drop_list3 = ['num_actv_rev_tl','num_bc_sats','num_bc_tl',
                  'num_rev_accts','num_sats','pct_tl_nvr_dlq',
                  'num_il_tl']


Total_drop_list1 = new_drop_list1 + new_drop_list2 + new_drop_list3

Loan_data = Loan_data.drop(Total_drop_list1, axis = 1)

Loan_data.shape
```

```
(2257919, 69)
```

```
Loan_data.head(1)
```

	loan_amnt	term	int_rate	installment	emp_length	home_ownership	annual_inc	loan_st
0	3600.0	36 months	13.99	123.03	10+ years	MORTGAGE	55000.0	

```
df_part4 = Loan_data.iloc[:,0:41]
corelation_part4 = df_part4.corr(method = 'spearman', min_periods = min_obs)
get_top_abs_correlations(corelation_part4, 50)
```

```
loan_amnt            installment             0.997649
revol_bal            max_bal_bc              0.965967
open_rv_24m          acc_open_past_24mths    0.886899
total_pymnt          total_rec_int           0.885693
installment          total_rec_int           0.876348
                     total_pymnt             0.871124
loan_amnt            total_rec_int           0.866227
                     total_pymnt             0.859289
revol_bal            total_rev_hi_lim        0.807593
annual_inc           tot_cur_bal             0.761272
max_bal_bc           total_rev_hi_lim        0.753032
revol_util           all_util                0.751703
int_rate             last_fico_range_low     0.747663
open_act_il          open_il_24m             0.744679
loan_amnt            revol_bal               0.694297
open_il_24m          acc_open_past_24mths    0.693971
loan_amnt            annual_inc              0.689122
installment          revol_bal               0.683262
annual_inc           total_rev_hi_lim        0.680539
open_il_24m          inq_fi                  0.678585
installment          annual_inc              0.672634
loan_amnt            max_bal_bc              0.667352
installment          max_bal_bc              0.653228
annual_inc           revol_bal               0.646165
                     max_bal_bc              0.628511
loan_amnt            total_rev_hi_lim        0.619684
total_acc            tot_cur_bal             0.616509
revol_bal            tot_cur_bal             0.613516
tot_cur_bal          total_rev_hi_lim        0.607626
installment          total_rev_hi_lim        0.593263
tot_cur_bal          max_bal_bc              0.576545
dti                  open_act_il             0.576404
revol_bal            total_pymnt             0.575731
il_util              all_util                0.574024
open_il_24m          il_util                 0.572249
annual_inc           total_pymnt             0.565907
total_acc            total_rev_hi_lim        0.564443
inq_fi               acc_open_past_24mths    0.558650
int_rate             total_rev_hi_lim        0.555338
total_acc            open_act_il             0.535418
revol_bal            total_rec_int           0.527452
last_fico_range_low  total_rev_hi_lim        0.525046
total_pymnt          max_bal_bc              0.517841
inq_last_6mths       open_rv_24m             0.517365
revol_util           open_rv_24m             0.516838
                     acc_open_past_24mths    0.513354
revol_bal            tot_coll_amt            0.510404
revol_util           max_bal_bc              0.508116
tot_coll_amt         max_bal_bc              0.506234
int_rate             all_util                0.503344
dtype: float64
```

```
new_drop_list4 = ['revol_bal', 'acc_open_past_24mths', 'installment',
                  'int_rate', 'revol_util']
```

```
df_part5 = Loan_data.iloc[:,41:81]
corelation_part5= df_part5.corr(method = 'spearman', min_periods = min_obs)
get_top_abs_correlations(corelation_part5, 50)
```

```
mo_sin_rcnt_tl                  num_tl_op_past_12m              0.962573
mo_sin_rcnt_rev_tl_op           mo_sin_rcnt_tl                  0.956591
num_actv_bc_tl                  num_rev_tl_bal_gt_0             0.954759
num_op_rev_tl                   num_rev_tl_bal_gt_0             0.953200
total_bal_ex_mort               total_il_high_credit_limit      0.952145
mo_sin_rcnt_rev_tl_op           num_tl_op_past_12m              0.934788
delinq_amnt                     num_tl_30dpd                    0.926013
num_actv_bc_tl                  num_op_rev_tl                   0.903193
mort_acc                        tot_hi_cred_lim                 0.828736
tot_hi_cred_lim                 total_bal_ex_mort               0.804227
num_actv_bc_tl                  total_bc_limit                  0.750487
tot_hi_cred_lim                 total_il_high_credit_limit      0.714925
num_op_rev_tl                   total_bc_limit                  0.695888
mths_since_recent_revol_delinq  num_accts_ever_120_pd           0.688989
mo_sin_rcnt_rev_tl_op           num_op_rev_tl                   0.659437
mo_sin_rcnt_tl                  mths_since_recent_inq           0.649613
mths_since_recent_inq           num_tl_op_past_12m              0.633080
num_rev_tl_bal_gt_0             total_bc_limit                  0.627111
tot_hi_cred_lim                 total_bc_limit                  0.590627
num_op_rev_tl                   num_tl_op_past_12m              0.589097
mo_sin_rcnt_rev_tl_op           num_rev_tl_bal_gt_0             0.579284
                                mths_since_recent_inq           0.572300
mo_sin_rcnt_tl                  num_op_rev_tl                   0.568430
num_accts_ever_120_pd           num_tl_90g_dpd_24m              0.558333
mo_sin_old_rev_tl_op            mort_acc                        0.551505
total_bal_ex_mort               total_bc_limit                  0.501123
num_rev_tl_bal_gt_0             num_tl_op_past_12m              0.491556
mo_sin_rcnt_tl                  num_rev_tl_bal_gt_0             0.475904
chargeoff_within_12_mths        num_tl_90g_dpd_24m              0.475803
mo_sin_rcnt_rev_tl_op           num_actv_bc_tl                  0.466745
mo_sin_old_il_acct              mo_sin_old_rev_tl_op            0.458942
                                tot_hi_cred_lim                 0.430252
mort_acc                        total_bal_ex_mort               0.429524
num_accts_ever_120_pd           total_bc_limit                  0.425955
mo_sin_old_il_acct              mort_acc                        0.424917
mo_sin_old_rev_tl_op            tot_hi_cred_lim                 0.422073
                                total_bc_limit                  0.415025
mo_sin_old_il_acct              total_bal_ex_mort               0.397311
                                total_il_high_credit_limit      0.389063
num_actv_bc_tl                  num_tl_op_past_12m              0.379033
mths_since_recent_inq           num_op_rev_tl                   0.364894
mo_sin_rcnt_tl                  num_actv_bc_tl                  0.364364
mort_acc                        total_il_high_credit_limit      0.356096
                                total_bc_limit                  0.350408
num_tl_90g_dpd_24m              total_bc_limit                  0.330309
num_tl_op_past_12m              total_bal_ex_mort               0.324805
                                percent_bc_gt_75                0.324535
num_op_rev_tl                   tot_hi_cred_lim                 0.323656
mo_sin_rcnt_tl                  total_bal_ex_mort               0.323599
                                tot_hi_cred_lim                 0.313435
dtype: float64
```

```
new_drop_list5 = ['mo_sin_rcnt_tl', 'total_bal_ex_mort', 'num_tl_30dpd',
                  'num_rev_tl_bal_gt_0']
```

```
Total_drop_list2 = new_drop_list4 + new_drop_list5
Loan_data = Loan_data.drop(Total_drop_list2, axis = 1)
```

```
Loan_data.shape
```

```
(2257919, 60)
```

```
null_counts3 = Loan_data.isnull().sum() / len(Loan_data)

null_counts3[0:-1]
```

```
loan_amnt                       0.000000
term                            0.000000
emp_length                      0.065048
home_ownership                  0.000000
annual_inc                      0.000000
loan_status                     0.000000
pymnt_plan                      0.000000
purpose                         0.000000
title                           0.000000
zip_code                        0.000000
dti                             0.000000
delinq_2yrs                     0.000000
earliest_cr_line                0.000000
inq_last_6mths                  0.000000
mths_since_last_record          0.000000
total_acc                       0.000000
out_prncp                       0.000000
total_pymnt                     0.000000
total_rec_int                   0.000000
last_credit_pull_d              0.000000
last_fico_range_low             0.000000
collections_12_mths_ex_med      0.000000
mths_since_last_major_derog     0.000000
application_type                0.000000
acc_now_delinq                  0.000000
tot_coll_amt                    0.000000
tot_cur_bal                     0.000000
open_act_il                     0.382379
open_il_24m                     0.382379
il_util                         0.472161
open_rv_24m                     0.382379
max_bal_bc                      0.382379
all_util                        0.382476
total_rev_hi_lim                0.000000
inq_fi                          0.382379
total_cu_tl                     0.382379
chargeoff_within_12_mths        0.000000
delinq_amnt                     0.000000
mo_sin_old_il_acct              0.060375
mo_sin_old_rev_tl_op            0.000000
mo_sin_rcnt_rev_tl_op           0.000000
mort_acc                        0.000000
mths_since_recent_inq           0.129626
mths_since_recent_revol_delinq  0.000000
num_accts_ever_120_pd           0.000000
num_actv_bc_tl                  0.000000
num_op_rev_tl                   0.000000
num_tl_120dpd_2m                0.066835
num_tl_90g_dpd_24m              0.000000
num_tl_op_past_12m              0.000000
percent_bc_gt_75                0.032167
pub_rec_bankruptcies            0.000000
tax_liens                       0.000000
tot_hi_cred_lim                 0.000000
total_bc_limit                  0.000000
total_il_high_credit_limit      0.000000
hardship_flag                   0.000000
hardship_amount                 0.000000
debt_settlement_flag            0.000000
dtype: float64
```

Now we have eliminated the highly correlated variables and have the narrower list of missing value variables

For those important credit record, we use "-1" to fill out the missing cells and represent as "information unavaiabl

Some of rows were removed due to relative small portion of missing value 3% to 6%

Others missing 40% missing value varaibles 'il_util','mo_sin_old_il_acct', since we already have some correlated variables, I decided to drop them

```
Loan_data = Loan_data.dropna(axis=0, subset=['emp_length'])
Loan_data['open_il_24m'].fillna(-1, inplace = True)
Loan_data['open_act_il'].fillna(-1, inplace = True)
Loan_data['mths_since_recent_inq'].fillna(-1, inplace = True)
Loan_data['open_rv_24m'].fillna(-1, inplace = True)
Loan_data['max_bal_bc'].fillna(-1, inplace = True)
Loan_data['all_util'].fillna(-1, inplace = True)
Loan_data['inq_fi'].fillna(-1, inplace = True)
Loan_data['total_cu_tl'].fillna(-1, inplace = True)
Loan_data = Loan_data.dropna(axis=0, subset=['num_tl_120dpd_2m'])
Loan_data = Loan_data.dropna(axis=0, subset=['percent_bc_gt_75'])
Loan_data = Loan_data.drop(['il_util','mo_sin_old_il_acct'], axis = 1)
```

Investigating catigoriacal variables

```
object_columns_df = Loan_data.select_dtypes(include=['object'])
print(object_columns_df.iloc[0])
```

```
term                             36 months
home_ownership                    MORTGAGE
loan_status                              0
pymnt_plan                               n
purpose                 debt_consolidation
title                   Debt consolidation
zip_code                                 1
earliest_cr_line                  Aug-2003
last_credit_pull_d                Mar-2019
application_type                Individual
hardship_flag                            N
debt_settlement_flag                     N
Name: 0, dtype: object
```

```
##Transform the employment_length to numbers

Loan_data.loc[Loan_data['emp_length'] == '10+ years', 'emp_length'] = 10
Loan_data.loc[Loan_data['emp_length'] == '< 1 year', 'emp_length'] = 0
Loan_data.loc[Loan_data['emp_length'] == '2 years', 'emp_length'] = 2
Loan_data.loc[Loan_data['emp_length'] == '3 years', 'emp_length'] = 3
Loan_data.loc[Loan_data['emp_length'] == '4 years', 'emp_length'] = 4
Loan_data.loc[Loan_data['emp_length'] == '5 years', 'emp_length'] = 5
Loan_data.loc[Loan_data['emp_length'] == '6 years', 'emp_length'] = 6
Loan_data.loc[Loan_data['emp_length'] == '7 years', 'emp_length'] = 7
Loan_data.loc[Loan_data['emp_length'] == '8 years', 'emp_length'] = 8
Loan_data.loc[Loan_data['emp_length'] == '9 years', 'emp_length'] = 9
Loan_data.loc[Loan_data['emp_length'] == '1 year', 'emp_length'] = 1
```

```
object_columns_df.columns.values
```

```
array(['term', 'home_ownership', 'loan_status', 'pymnt_plan', 'purpose',
       'title', 'zip_code', 'earliest_cr_line', 'last_credit_pull_d',
       'application_type', 'hardship_flag', 'debt_settlement_flag'],
      dtype=object)
```

```
cols = object_columns_df.columns.values
for name in cols:
  print(name,':')
  print(object_columns_df[name].value_counts(),'\n')
```

```
term :
 36 months    1370853
 60 months     574592
Name: term, dtype: int64

home_ownership :
MORTGAGE    961235
RENT        775754
OWN         207547
ANY            826
NONE            42
OTHER           41
Name: home_ownership, dtype: int64

loan_status :
0    1704776
1     240669
Name: loan_status, dtype: int64

pymnt_plan :
n    1944918
y        527
Name: pymnt_plan, dtype: int64

purpose :
debt_consolidation    1103796
credit_card            452768
home_improvement       127229
other                  116624
major_purchase          42661
medical                 22707
small_business          20072
car                     19633
vacation                13000
moving                  12903
house                   12033
renewable_energy         1163
wedding                   854
educational                 2
Name: purpose, dtype: int64

title :
Debt consolidation                      1037450
Credit card refinancing                  418564
Home improvement                         119651
Other                                    111332
Major purchase                            40293
                                          ...
Yay!                                          1
balance transfert                             1
Consolidate & Refinance                       1
New Pool and Spa                              1
Business Loan- Building Improvements          1
Name: title, Length: 37132, dtype: int64

zip_code :
9    347762
3    265302
1    232177
7    217649
2    189944
0    176327
4    164584
8    146428
```

```
6    132362
5     72910
Name: zip_code, dtype: int64

earliest_cr_line :
Sep-2004    13971
Sep-2003    13747
Sep-2005    13413
Aug-2003    13178
Aug-2004    13039
            ...
Jul-1955        1
Nov-1957        1
May-1955        1
May-1953        1
Nov-1953        1
Name: earliest_cr_line, Length: 736, dtype: int64

last_credit_pull_d :
Mar-2019    1204767
Feb-2019      66986
Jan-2019      54684
Jul-2018      46681
Oct-2018      40668
             ...
Jan-2013         14
Oct-2012         13
Sep-2012          2
Apr-2019          2
Aug-2012          1
Name: last_credit_pull_d, Length: 81, dtype: int64

application_type :
Individual    1843614
Joint App      101831
Name: application_type, dtype: int64

hardship_flag :
N    1944731
Y        714
Name: hardship_flag, dtype: int64

debt_settlement_flag :
N    1916396
Y      29049
Name: debt_settlement_flag, dtype: int64
```

```
Loan_data.shape[0]*0.01
```

```
19454.45
```

From the above obersavation, we can see that following categorcial variables can be further modified:

1. Purpose & title contains similar values, but Purpose's categories are more clear so we can drop title
2. "pymnt_plan : n 2257299 y 620." Since there are only 620 cases have payment plan which has less than 1% obervation. We can drop this variable because almost all obs has n in pymnt_plan.

3. For the Homeownership, Aggregate the "ANY" "OTHER" "NONE" cases together constitutes less than 1% of t data obs. So I decide to drop them.
4. As for dates category such as "earliest_cr_line","last_credit_pull_d". We can modify them as year instead of months

```
drop_list_round5 = ['title', 'pymnt_plan']
Loan_data = Loan_data.drop(drop_list_round5, axis = 1)

Loan_data = Loan_data.drop(Loan_data[Loan_data['home_ownership'] == 'NONE'].index, axis = 0)
Loan_data = Loan_data.drop(Loan_data[Loan_data['home_ownership'] == 'OTHER'].index, axis = 0)
Loan_data = Loan_data.drop(Loan_data[Loan_data['home_ownership'] == 'ANY'].index, axis = 0)

Loan_data.loc[Loan_data['earliest_cr_line'] != np.nan, 'earliest_cr_line'] =  \
Loan_data.loc[Loan_data['earliest_cr_line'] != np.nan, 'earliest_cr_line'].str[4:]



Loan_data.loc[Loan_data['last_credit_pull_d'] != np.nan, 'last_credit_pull_d'] =  \
Loan_data.loc[Loan_data['last_credit_pull_d'] != np.nan, 'last_credit_pull_d'].str[4:]



Loan_data['earliest_cr_line'] = Loan_data['earliest_cr_line'].astype('float')
Loan_data['last_credit_pull_d'] = Loan_data['last_credit_pull_d'].astype('float')
```

Delete all rows with too many Null values

Our threshold is half of the coulmns left

```
Loan_data = Loan_data.dropna(thresh= Loan_data.shape[1] * 0.5, axis = 0)

Loan_data.head(1)
```

	loan_amnt	term	emp_length	home_ownership	annual_inc	loan_status	purpose	z
0	3600.0	36 months	10	MORTGAGE	55000.0	0	debt_consolidation	

Dealing with Outliers

```
Loan_data.columns
```

```
      Index(['loan_amnt', 'term', 'emp_length', 'home_ownership', 'annual_inc',
             'loan_status', 'purpose', 'zip_code', 'dti', 'delinq_2yrs',
             'earliest_cr_line', 'inq_last_6mths', 'mths_since_last_record',
             'total_acc', 'out_prncp', 'total_pymnt', 'total_rec_int',
             'last_credit_pull_d', 'last_fico_range_low',
             'collections_12_mths_ex_med', 'mths_since_last_major_derog',
             'application_type', 'acc_now_delinq', 'tot_coll_amt', 'tot_cur_bal',
             'open_act_il', 'open_il_24m', 'open_rv_24m', 'max_bal_bc', 'all_util',
             'total_rev_hi_lim', 'inq_fi', 'total_cu_tl', 'chargeoff_within_12_mths',
             'delinq_amnt', 'mo_sin_old_rev_tl_op', 'mo_sin_rcnt_rev_tl_op',
             'mort_acc', 'mths_since_recent_inq', 'mths_since_recent_revol_delinq',
             'num_accts_ever_120_pd', 'num_actv_bc_tl', 'num_op_rev_tl',
             'num_tl_120dpd_2m', 'num_tl_90g_dpd_24m', 'num_tl_op_past_12m',
             'percent_bc_gt_75', 'pub_rec_bankruptcies', 'tax_liens',
             'tot_hi_cred_lim', 'total_bc_limit', 'total_il_high_credit_limit',
             'hardship_flag', 'hardship_amount', 'debt_settlement_flag', 'LGD'],
            dtype='object')
```

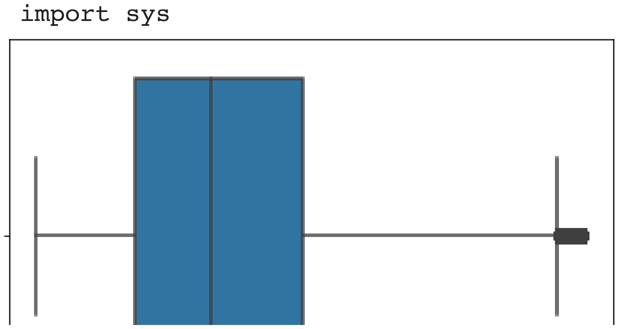
```
## Final check with missing columns
nullset = Loan_data.isnull()
null_counts = Loan_data.isnull().sum() / len(Loan_data)
print("percentage of null values in each column:\n{}".format(null_counts.head(60)))
```

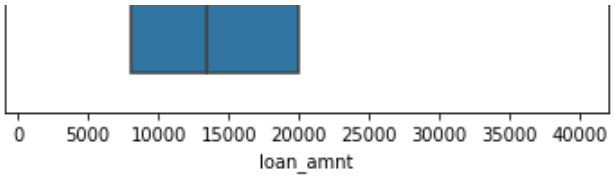
```
percentage of null values in each column:
loan_amnt                          0.0
term                               0.0
emp_length                         0.0
home_ownership                     0.0
annual_inc                         0.0
loan_status                        0.0
purpose                            0.0
zip_code                           0.0
dti                                0.0
delinq_2yrs                        0.0
earliest_cr_line                   0.0
inq_last_6mths                     0.0
mths_since_last_record             0.0
total_acc                          0.0
out_prncp                          0.0
total_pymnt                        0.0
total_rec_int                      0.0
last_credit_pull_d                 0.0
last_fico_range_low                0.0
collections_12_mths_ex_med         0.0
mths_since_last_major_derog        0.0
application_type                   0.0
acc_now_delinq                     0.0
tot_coll_amt                       0.0
tot_cur_bal                        0.0
open_act_il                        0.0
open_il_24m                        0.0
open_rv_24m                        0.0
max_bal_bc                         0.0
all_util                           0.0
total_rev_hi_lim                   0.0
inq_fi                             0.0
total_cu_tl                        0.0
chargeoff_within_12_mths           0.0
delinq_amnt                        0.0
mo_sin_old_rev_tl_op               0.0
mo_sin_rcnt_rev_tl_op              0.0
mort_acc                           0.0
mths_since_recent_inq              0.0
mths_since_recent_revol_delinq     0.0
num_accts_ever_120_pd              0.0
num_actv_bc_tl                     0.0
num_op_rev_tl                      0.0
num_tl_120dpd_2m                   0.0
num_tl_90g_dpd_24m                 0.0
num_tl_op_past_12m                 0.0
percent_bc_gt_75                   0.0
pub_rec_bankruptcies               0.0
tax_liens                          0.0
tot_hi_cred_lim                    0.0
total_bc_limit                     0.0
total_il_high_credit_limit         0.0
hardship_flag                      0.0
hardship_amount                    0.0
debt_settlement_flag               0.0
LGD                                0.0
dtype: float64
```

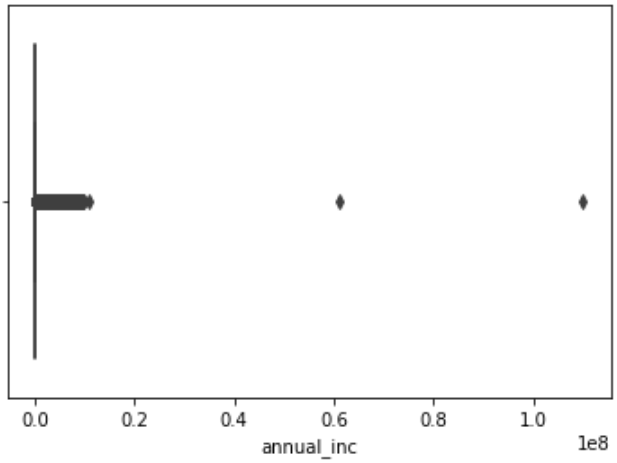
plot a boxplot for each survived features, and delete any outlier that is disjoint with continious dots plot

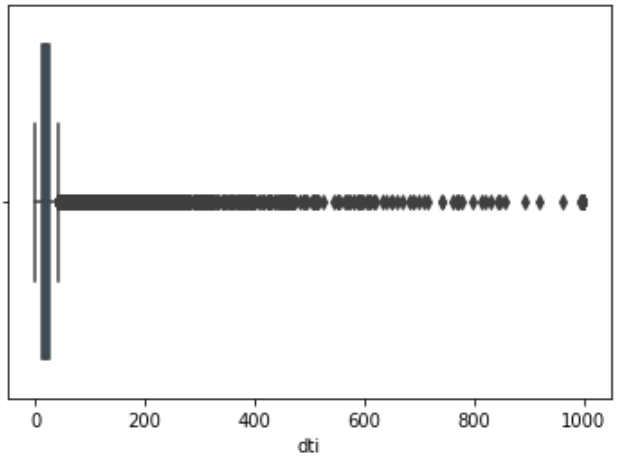
```
import matplotlib
import matplotlib.pyplot as plt
import seaborn
numerical_columns_df = Loan_data.select_dtypes(include=['float64'])
for column in numerical_columns_df.columns:
  if Loan_data[column].isnull().sum() == 0:
    plt.figure()
    sns.boxplot(Loan_data[column])
  else:
    pass
```

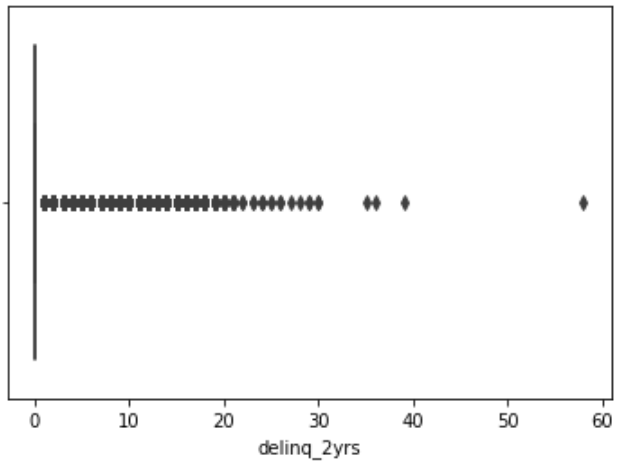
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: More than
  import sys
```

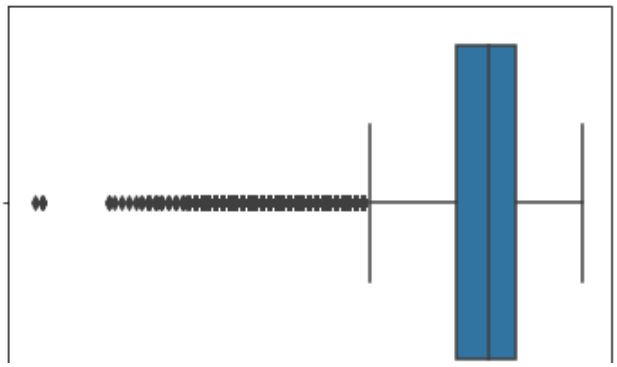


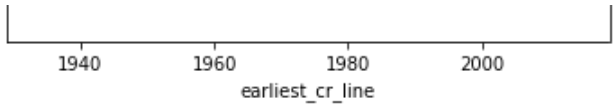

0
5000
10000
15000
20000
25000
30000
35000
40000
loan_amnt

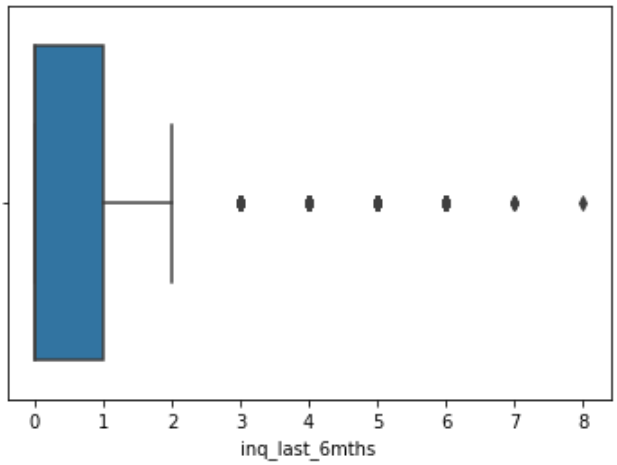

0.0
0.2
0.4
0.6
0.8
1.0
annual_inc
1e8

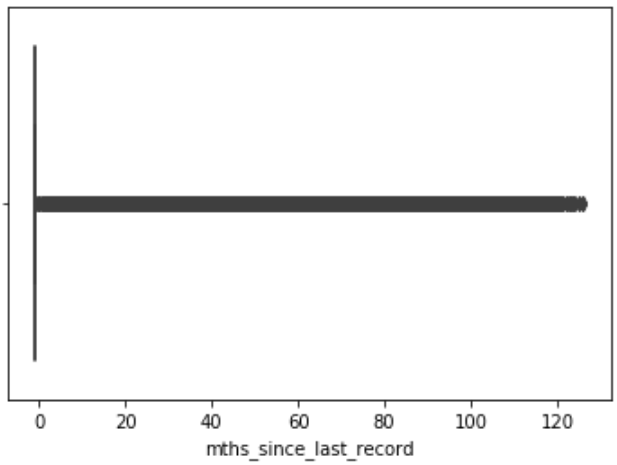

0
200
400
600
800
1000
dti

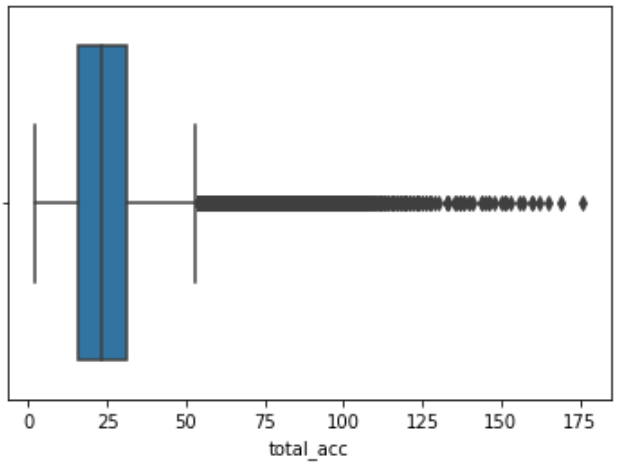

0
10
20
30
40
50
60
delinq_2yrs

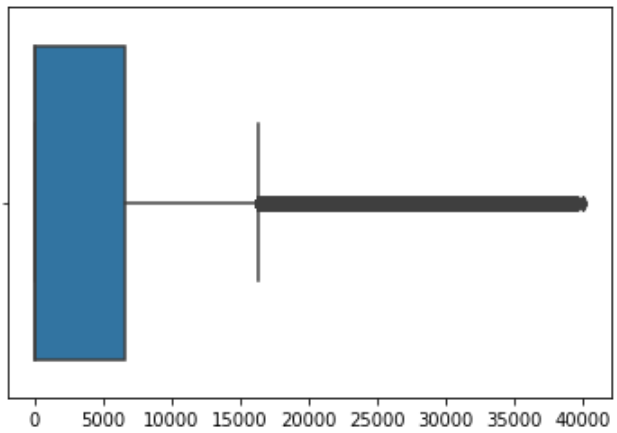


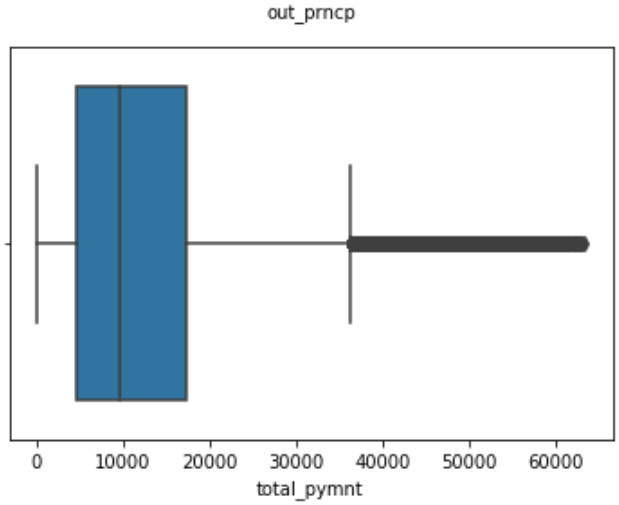

1940
1960
1980
2000
earliest_cr_line

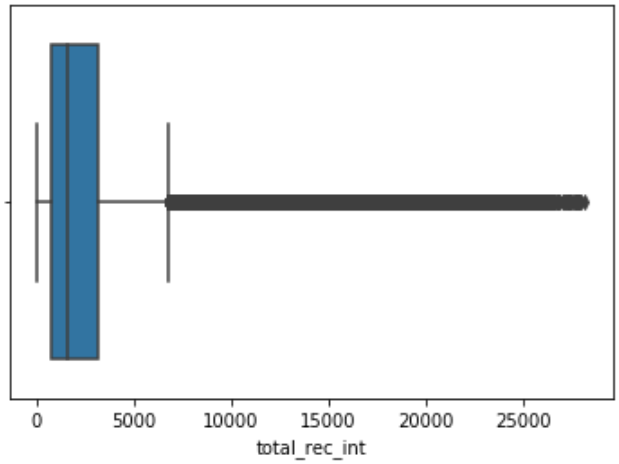

0
1
2
3
4
5
6
7
8
inq_last_6mths

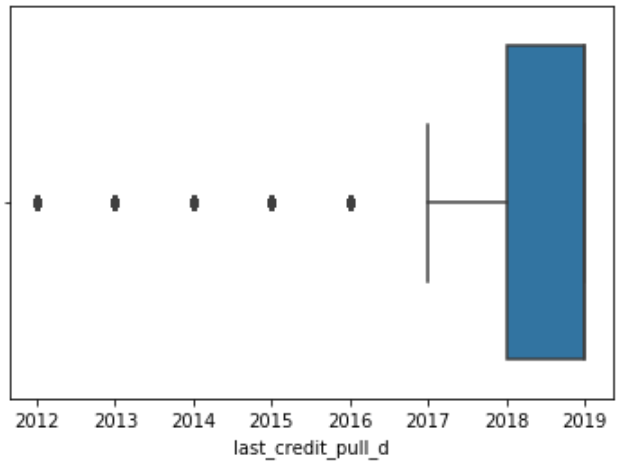

0
20
40
60
80
100
120
mths_since_last_record

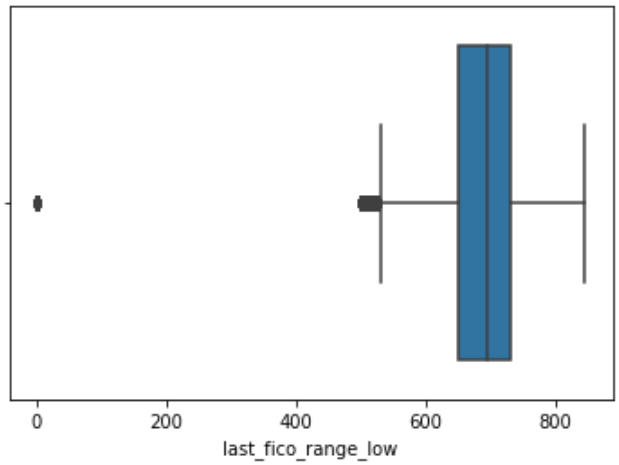

0
25
50
75
100
125
150
175
total_acc

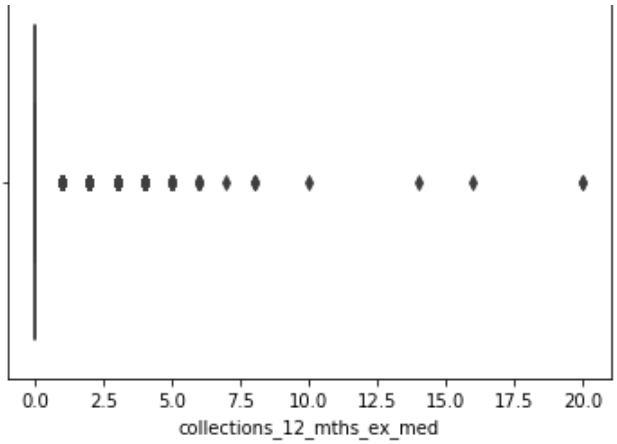

0
5000
10000
15000
20000
25000
30000
35000
40000

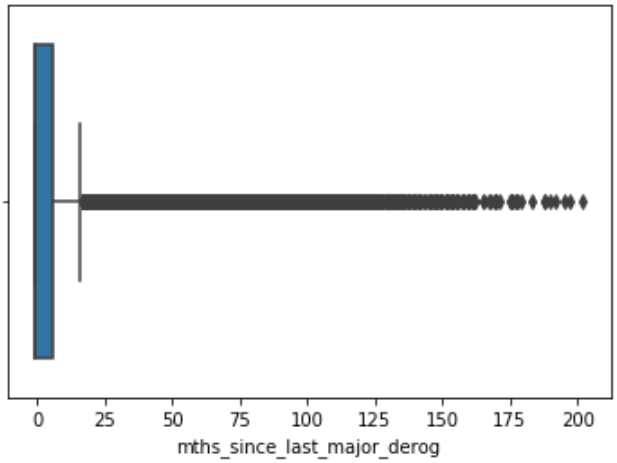

out_prncp
0
10000
20000
30000
40000
50000
60000
total_pymnt

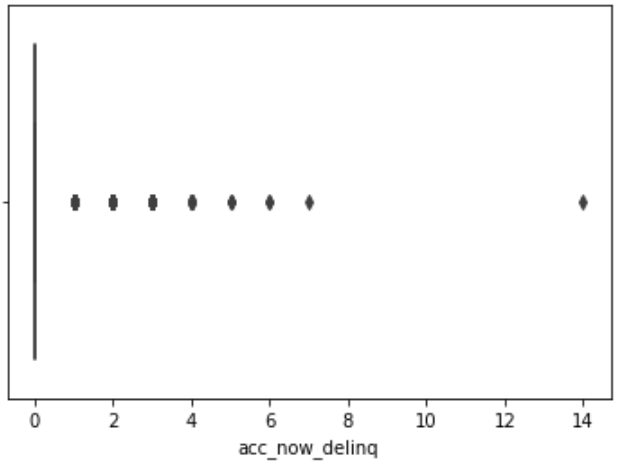

0
5000
10000
15000
20000
25000
total_rec_int

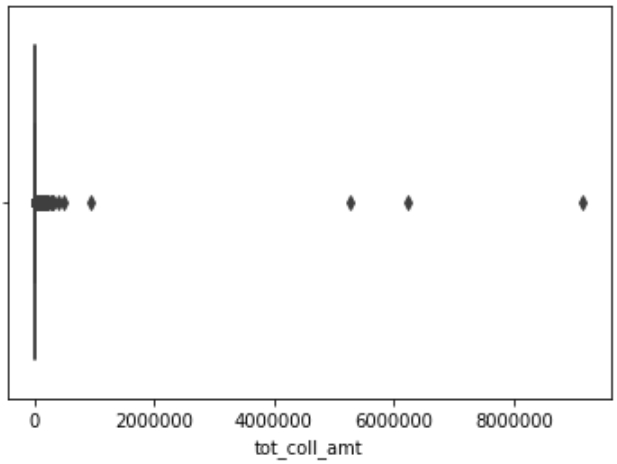

2012
2013
2014
2015
2016
2017
2018
2019
last_credit_pull_d

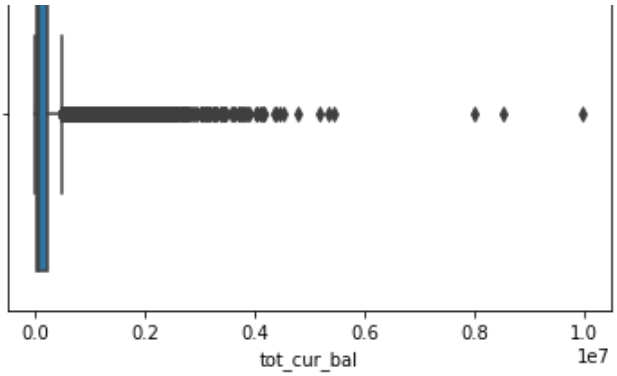

0
200
400
600
800
last_fico_range_low

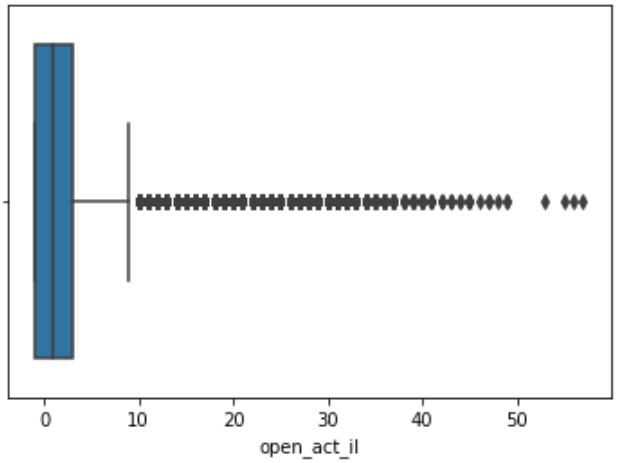

0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
collections_12_mths_ex_med

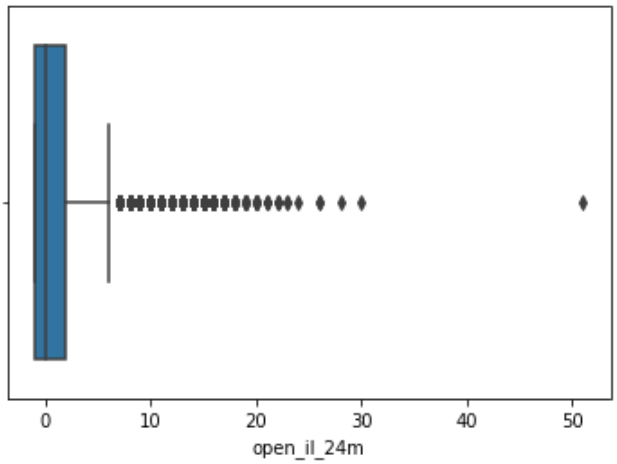

0
25
50
75
100
125
150
175
200
mths_since_last_major_derog

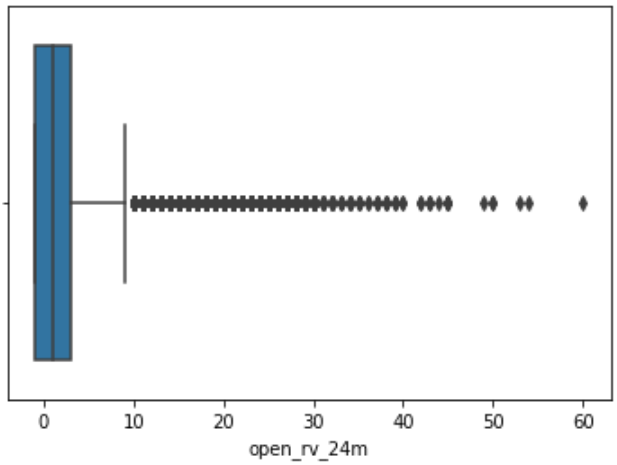

0
2
4
6
8
10
12
14
acc_now_delinq

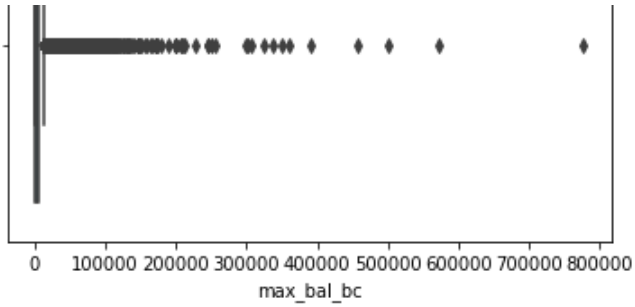

0
2000000
4000000
6000000
8000000
tot_coll_amt

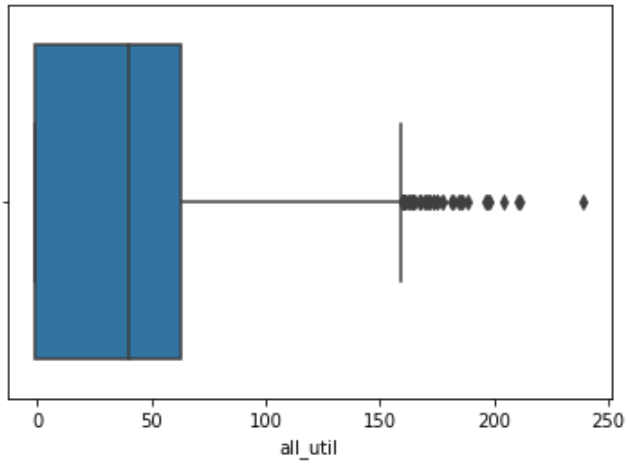

0.0
0.2
0.4
0.6
0.8
1.0
1e7
tot_cur_bal

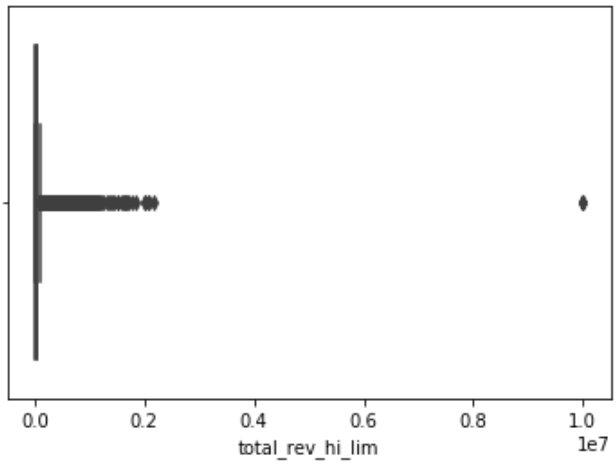

0
10
20
30
40
50
open_act_il

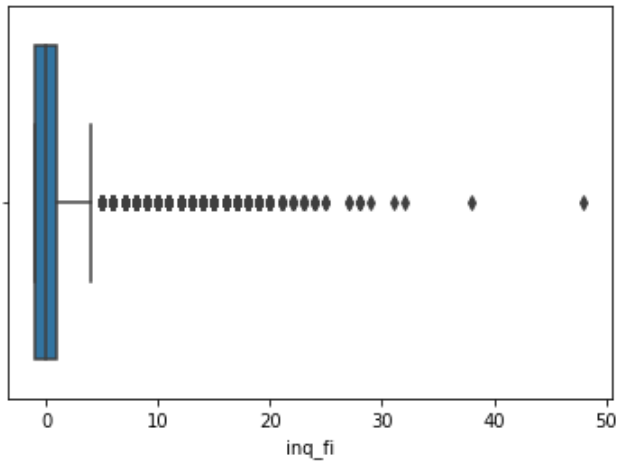

0
10
20
30
40
50
open_il_24m

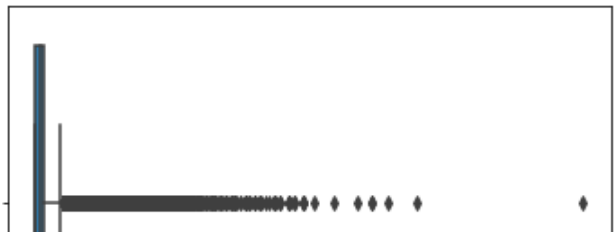

0
10
20
30
40
50
60
open_rv_24m

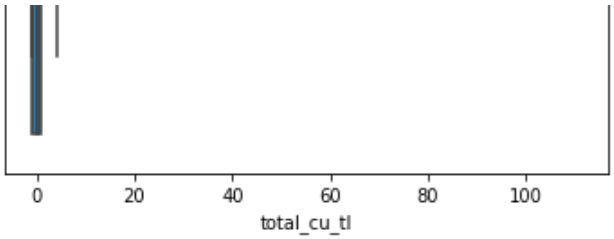

0
100000
200000
300000
400000
500000
600000
700000
800000
max_bal_bc

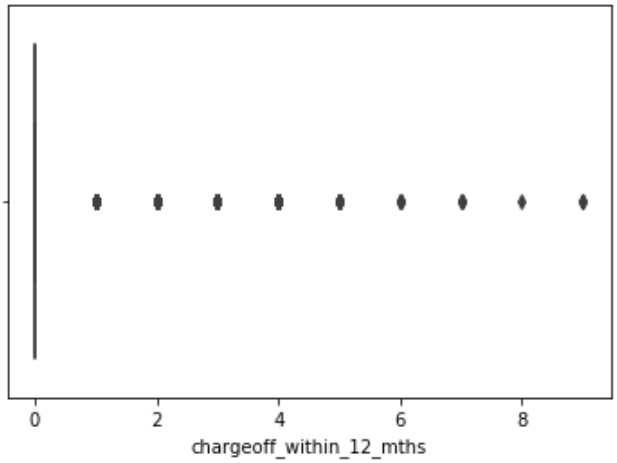

0
50
100
150
200
250
all_util

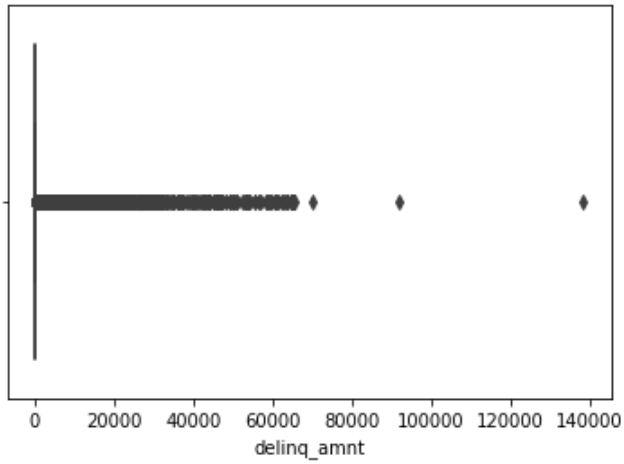

0.0
0.2
0.4
0.6
0.8
1.0
1e7
total_rev_hi_lim

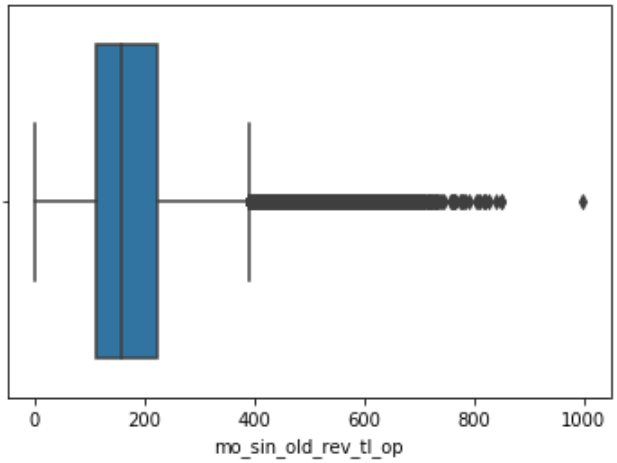

0
10
20
30
40
50
inq_fi

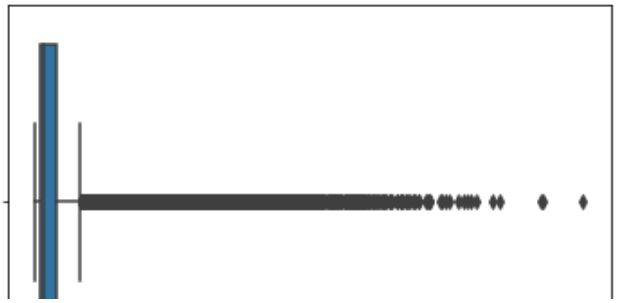


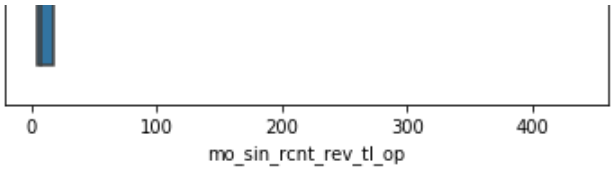

0
20
40
60
80
100
total_cu_tl

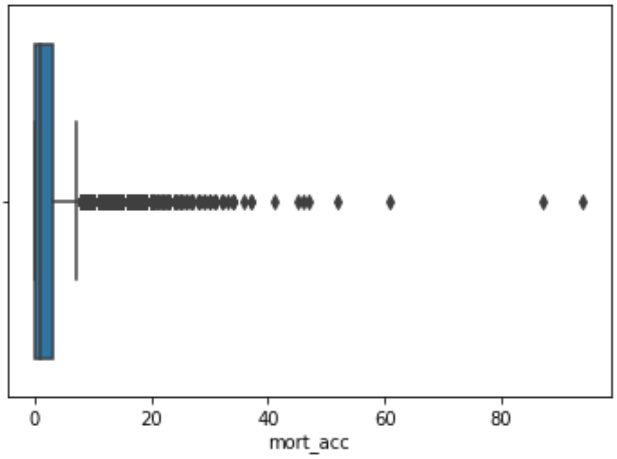

0
2
4
6
8
chargeoff_within_12_mths

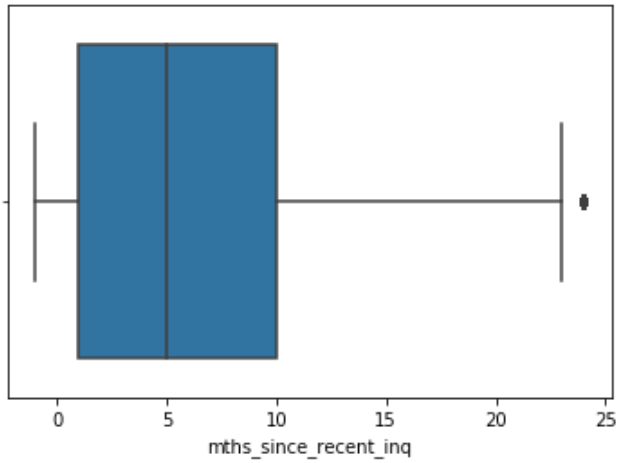

0
20000
40000
60000
80000
100000
120000
140000
delinq_amnt

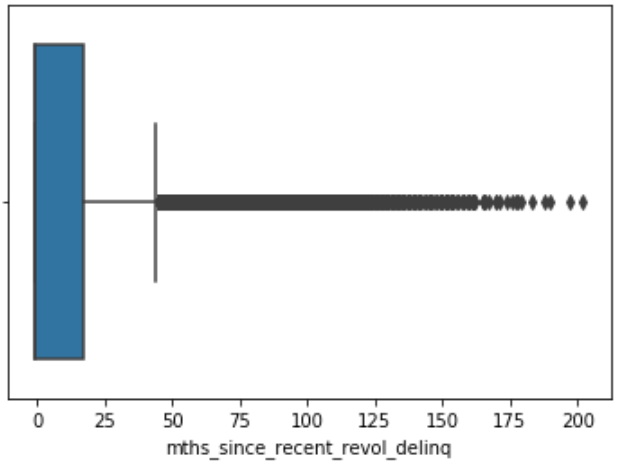

0
200
400
600
800
1000
mo_sin_old_rev_tl_op

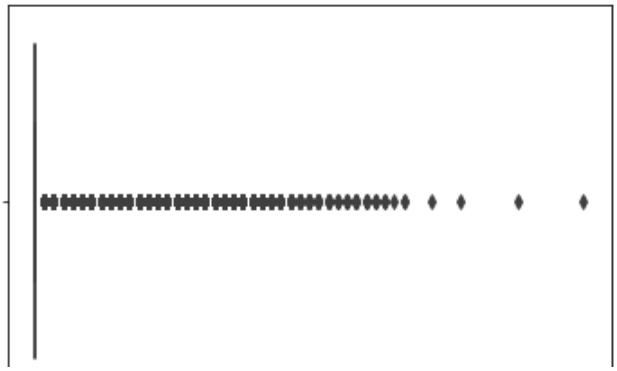


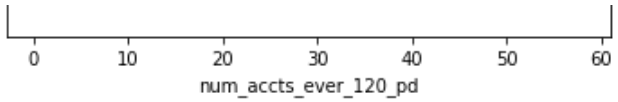

0
100
200
300
400
mo_sin_rcnt_rev_tl_op

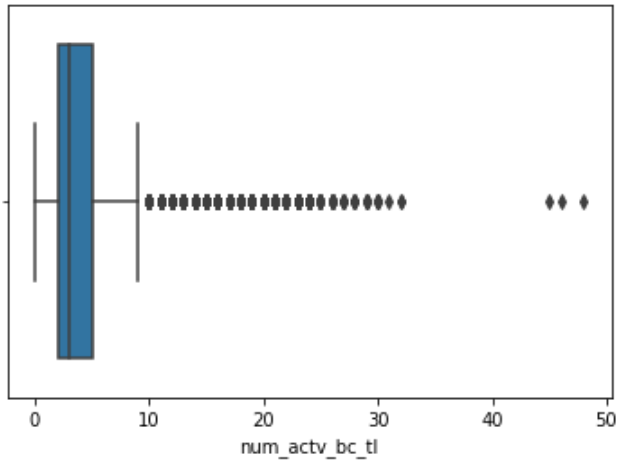

0
20
40
60
80
mort_acc

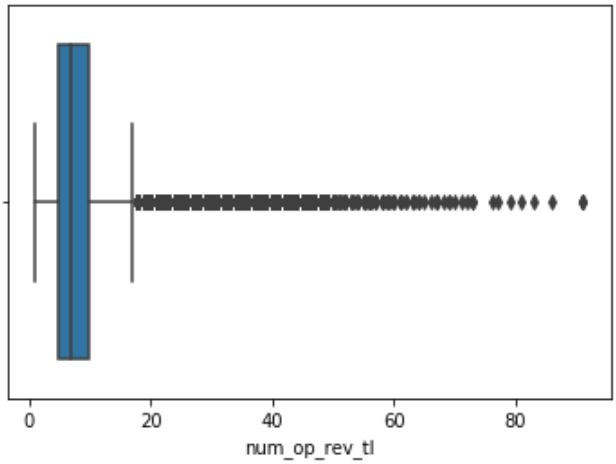

0
5
10
15
20
25
mths_since_recent_inq

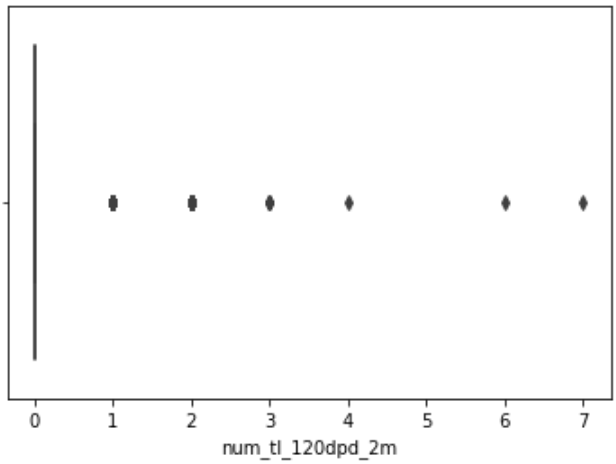

0
25
50
75
100
125
150
175
200
mths_since_recent_revol_delinq

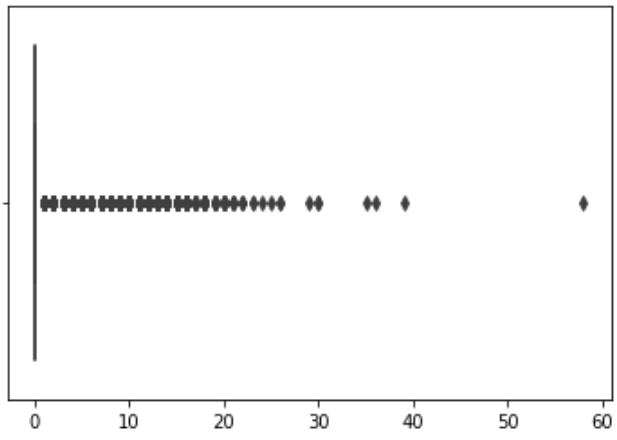


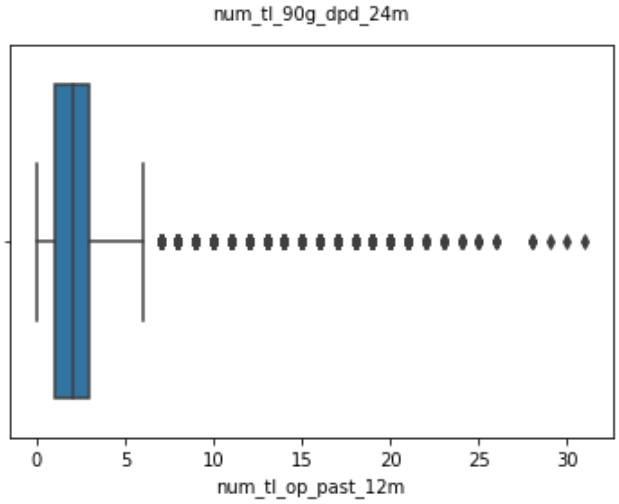

0
10
20
30
40
50
60
num_accts_ever_120_pd

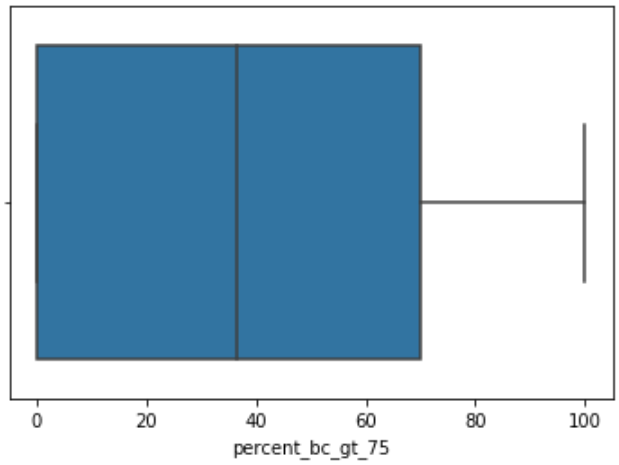

0
10
20
30
40
50
num_actv_bc_tl

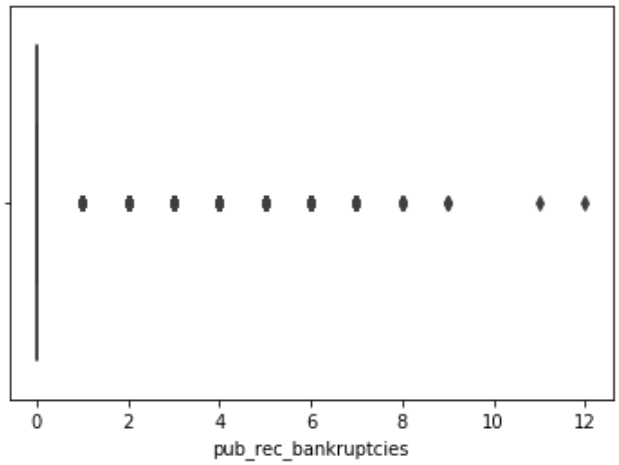

0
20
40
60
80
num_op_rev_tl

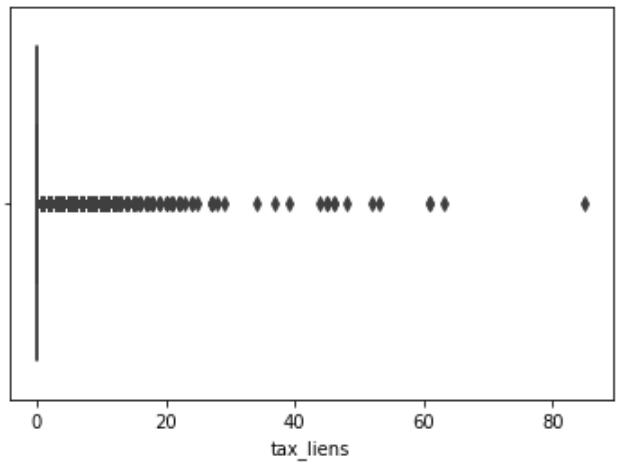

0
1
2
3
4
5
6
7
num_tl_120dpd_2m

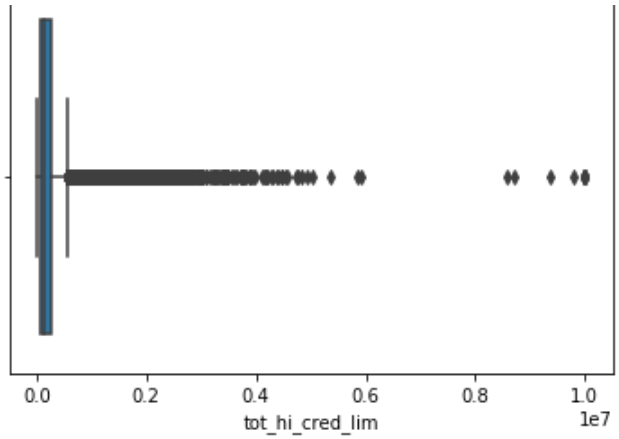

0
10
20
30
40
50
60

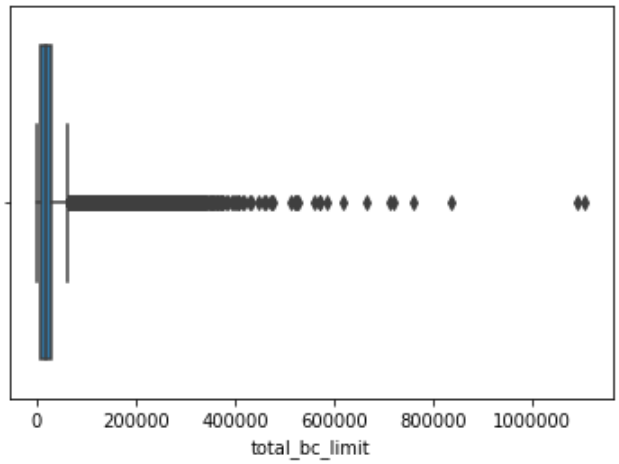

num_tl_90g_dpd_24m
0
5
10
15
20
25
30
num_tl_op_past_12m

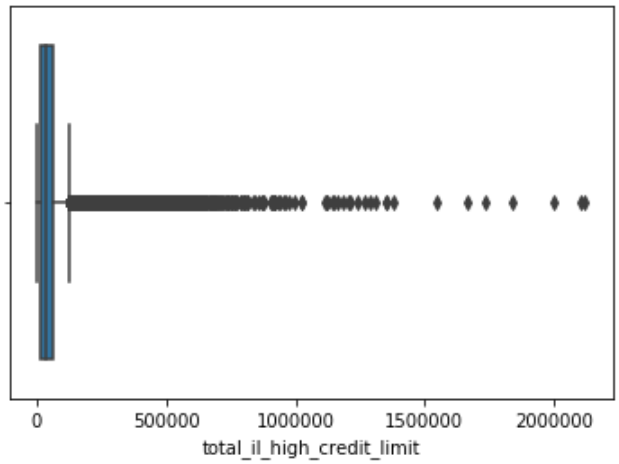

0
20
40
60
80
100
percent_bc_gt_75

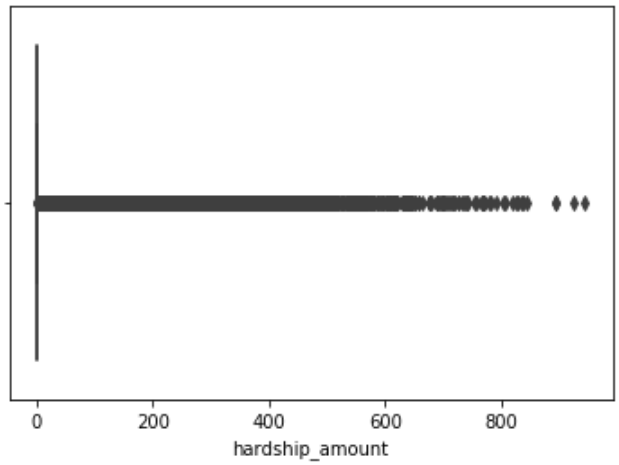

0
2
4
6
8
10
12
pub_rec_bankruptcies

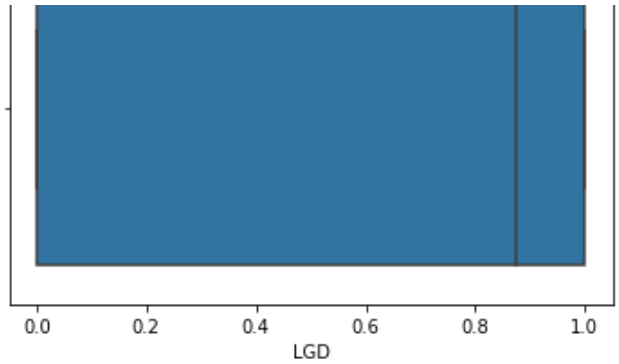

0
20
40
60
80
tax_liens


0.0
0.2
0.4
0.6
0.8
1.0
1e7
tot_hi_cred_lim


0
200000
400000
600000
800000
1000000
total_bc_limit


0
500000
1000000
1500000
2000000
total_il_high_credit_limit


0
200
400
600
800
hardship_amount



From the Above distribution graphs, we can see we need to deal with outliers in

```
## Find number of outliers
print(Loan_data.loc[(Loan_data['annual_inc'] > 0.5 * 10 ** 8)].count()[0])
print(Loan_data.loc[(Loan_data['annual_inc'] < 0)].count()[0])
## Delete them
Loan_data = Loan_data.drop(Loan_data[Loan_data['annual_inc'] > 0.5 * 10 ** 8].index, axis = 0)
```

```
2
0
```

```
## Find number of outliers
print(Loan_data.loc[Loan_data['dti'] > 700].count()[0])
print(Loan_data.loc[Loan_data['dti'] < 0].count()[0])

Loan_data = Loan_data.drop(Loan_data[Loan_data['dti'] < 0].index, axis = 0)
Loan_data = Loan_data.drop(Loan_data[Loan_data['dti'] > 700].index, axis = 0)
```

```
74
2
```

```
print(Loan_data.loc[Loan_data['delinq_2yrs'] > 30].count()[0])
Loan_data = Loan_data.drop(Loan_data[Loan_data['delinq_2yrs'] > 30].index, axis = 0)
```

```
4
```

```
print(Loan_data.loc[Loan_data['total_acc'] > 160].count()[0])
Loan_data = Loan_data.drop(Loan_data[Loan_data['total_acc'] > 30].index, axis = 0)
```

```
4
```

```
print(Loan_data.loc[Loan_data['inq_last_6mths'] > 7].count()[0])
#Loan_data = Loan_data.drop(Loan_data[Loan_data['delinq_2yrs'] > 8].index, axis = 0)
```

```
1
```

```
print(Loan_data.loc[Loan_data['total_rev_hi_lim'] > 0.8*10**7].count()[0])
Loan_data = Loan_data.drop(Loan_data[Loan_data['total_rev_hi_lim'] > 0.8*10**7\
                                     ].index, axis = 0)
```

```
print(Loan_data.loc[Loan_data['last_fico_range_low'] < 200].count()[0] )
## We drop the extreme cases where fico is 0, that is the group where
## people haven't created any credit record with FICO, so that
## would be extrme cases, those doesn't belongs to majority of the group

Loan_data = Loan_data.drop(Loan_data[Loan_data['last_fico_range_low'] < 200].index, axis = 0)
```

```
26842
```

```
print(Loan_data.loc[Loan_data['acc_now_delinq'] > 13].count()[0] )
Loan_data = Loan_data.drop(Loan_data[Loan_data['acc_now_delinq'] > 13].index, axis = 0)
```

```
0
```

```
print(Loan_data.loc[Loan_data['tot_coll_amt'] > 4000000].count()[0] )
Loan_data = Loan_data.drop(Loan_data[Loan_data['tot_coll_amt'] > 4000000].index, axis = 0)
```

```
1
```

```
print(Loan_data.loc[Loan_data['total_bc_limit'] > 1000000].count()[0] )
Loan_data = Loan_data.drop(Loan_data[Loan_data['total_bc_limit'] > 1000000].index, axis = 0)
```

```
1
```

```
print(Loan_data.loc[Loan_data['tax_liens'] > 40].count()[0] )
Loan_data = Loan_data.drop(Loan_data[Loan_data['tax_liens'] > 40].index, axis = 0)
```

```
9
```

```
print(Loan_data.loc[Loan_data['mort_acc'] > 50].count()[0] )
Loan_data = Loan_data.drop(Loan_data[Loan_data['mort_acc'] > 50].index, axis = 0)
```

```
0
```

```
print(Loan_data.loc[Loan_data['num_tl_90g_dpd_24m'] > 28].count()[0] )
```

```
0
```

```
print(Loan_data.loc[Loan_data['tot_hi_cred_lim'] > 0.6 * 10 ** 7].count()[0] )
Loan_data = Loan_data.drop(Loan_data[Loan_data['tot_hi_cred_lim'] > 0.6 \
                                     * 10 ** 7].index, axis = 0)
```

```
5
```

```
print(Loan_data.loc[Loan_data['total_bc_limit'] > 500000].count()[0] )
Loan_data = Loan_data.drop(Loan_data[Loan_data['total_bc_limit'] > 500000\
                                     ].index, axis = 0)
```

```
3
```

```
print(Loan_data.loc[Loan_data['total_il_high_credit_limit'] > 1000000].count()[0] )
Loan_data = Loan_data.drop(Loan_data[Loan_data['total_il_high_credit_limit'] > \
                                     1000000].index, axis = 0)
```

```
7
```

```
Loan_data.shape
```

```
(1405627, 56)
```

Now we finished clean, and we can create a copy of data to feed in LGD model

```
LGD_Variables = ['hardship_amount','loan_amnt','term','emp_length','home_ownership',
 'annual_inc','zip_code','delinq_2yrs','earliest_cr_line','mths_since_last_record',
 'out_prncp','total_rec_int','total_pymnt','last_fico_range_low',
 'collections_12_mths_ex_med','mths_since_last_major_derog',
 'application_type','acc_now_delinq','tot_coll_amt',
 'chargeoff_within_12_mths','delinq_amnt','mort_acc',
 'mths_since_recent_revol_delinq','num_accts_ever_120_pd',
 'num_tl_120dpd_2m','num_tl_90g_dpd_24m','percent_bc_gt_75',
 'pub_rec_bankruptcies', 'tax_liens',
 'total_bc_limit','sec_app_chargeoff_within_12_mths',
 'debt_settlement_flag','LGD', 'loan_status']
```

```
LGD_data = Loan_data.loc[Loan_data['loan_status'] != 0]
```

```
Loan_data.to_csv("credit_data.csv", index=False)
LGD_data.to_csv('LGD_data.csv',index = False)
!ls
```

```
'Coursework 2 - Lending Club Data.zip'   sample_data
 credit_data.csv                         test_woe.csv
 LCFinal.csv                             train_woe.csv
 LGD_data.csv                           'Variable Dictionary.docx'
```

```
from google.colab import files
```

```
%%script false
files.download('credit_data.csv')
files.download('LGD_data.csv')
```

```
---------------------------------------------------------------------------
MessageError                                Traceback (most recent call last)
<ipython-input-254-86fa8f952084> in <module>()
----> 1 files.download('credit_data.csv')
      2 files.download('LGD_data.csv')

                                 2 frames
/usr/local/lib/python3.6/dist-packages/google/colab/_message.py in read_reply_from_input(m
    104         reply.get('colab_msg_id') == message_id):
    105       if 'error' in reply:
--> 106         raise MessageError(reply['error'])
    107       return reply.get('data', None)
    108

MessageError: TypeError: Failed to fetch
```

SEARCH STACK OVERFLOW

Part II Construct ScoreCard

Convert Variables to WOE Values

```
!pip install scorecardpy
import scorecardpy as sc
import numpy as np
```

```
Requirement already satisfied: scorecardpy in /usr/local/lib/python3.6/dist-packages (0.1.
Requirement already satisfied: pandas>=0.25.0 in /usr/local/lib/python3.6/dist-packages (f
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from score
Requirement already satisfied: scikit-learn>=0.19.1 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages (fro
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (fro
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.6/dist-packages (fr
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (fro
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/dist-packages (from py
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (from
```

```
Loan_data.head(1)
```

```
# Split in train and test BEFORE we apply WoE
train, test = sc.split df(Loan data.iloc[:,:-1],
```

```
                            y = 'loan_status',
                            ratio = 0.66, seed = 251121253).values()
```

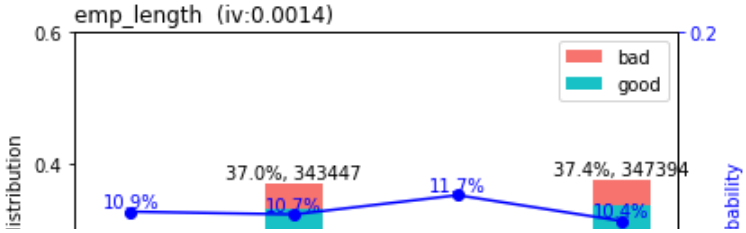
```
train.describe()
```

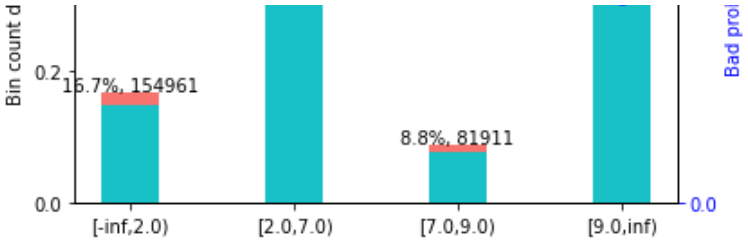
```
bins = sc.woebin(train, y = 'loan_status',
                 min_perc_fine_bin=0.05,
                 min_perc_coarse_bin=0.05,
                 stop_limit=0.1,
                 max_num_bin=8,
                 method='tree')
```

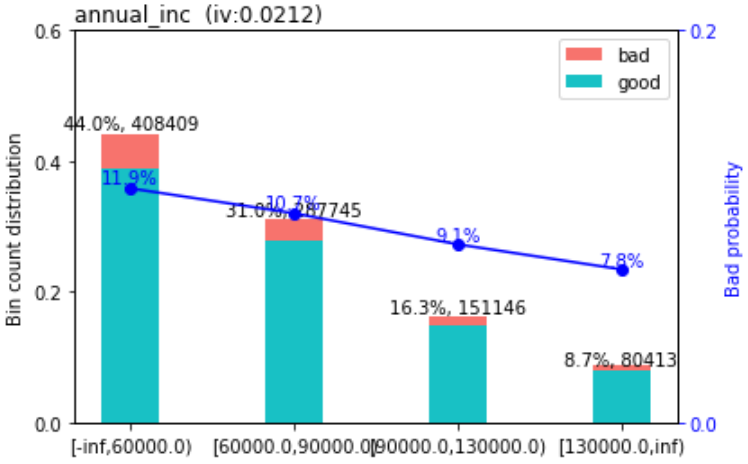
```
[INFO] creating woe binning ...
Binning on 927713 rows and 55 columns in 00:02:58
```

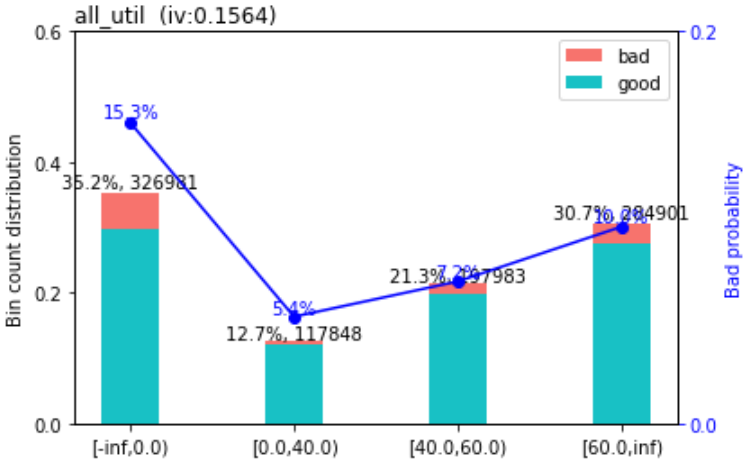
```
sc.woebin_plot(bins)
```

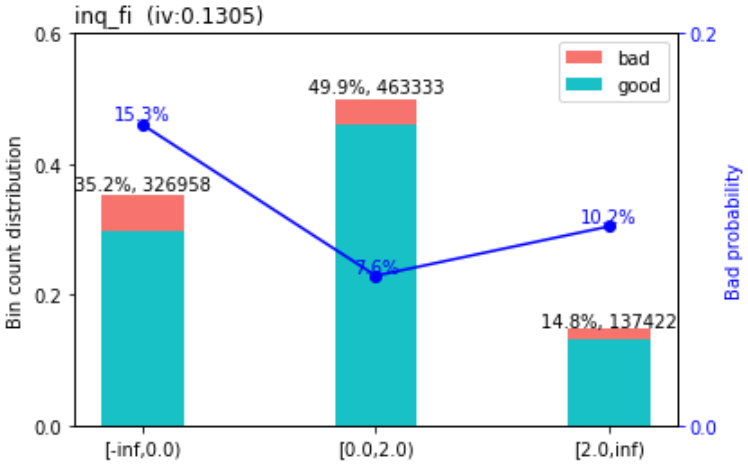
```
/usr/local/lib/python3.6/dist-packages/scorecardpy/woebin.py:1197: RuntimeWarning: More th
  fig, ax1 = plt.subplots()
{'acc_now_delinq': <Figure size 432x288 with 2 Axes>,
 'all_util': <Figure size 432x288 with 2 Axes>,
 'annual_inc': <Figure size 432x288 with 2 Axes>,
 'application_type': <Figure size 432x288 with 2 Axes>,
 'chargeoff_within_12_mths': <Figure size 432x288 with 2 Axes>,
 'collections_12_mths_ex_med': <Figure size 432x288 with 2 Axes>,
 'debt_settlement_flag': <Figure size 432x288 with 2 Axes>,
 'delinq_2yrs': <Figure size 432x288 with 2 Axes>,
 'delinq_amnt': <Figure size 432x288 with 2 Axes>,
 'dti': <Figure size 432x288 with 2 Axes>,
 'earliest_cr_line': <Figure size 432x288 with 2 Axes>,
 'emp_length': <Figure size 432x288 with 2 Axes>,
 'hardship_amount': <Figure size 432x288 with 2 Axes>,
 'hardship_flag': <Figure size 432x288 with 2 Axes>,
 'home_ownership': <Figure size 432x288 with 2 Axes>,
 'inq_fi': <Figure size 432x288 with 2 Axes>,
 'inq_last_6mths': <Figure size 432x288 with 2 Axes>,
 'last_credit_pull_d': <Figure size 432x288 with 2 Axes>,
 'last_fico_range_low': <Figure size 432x288 with 2 Axes>,
 'loan_amnt': <Figure size 432x288 with 2 Axes>,
 'max_bal_bc': <Figure size 432x288 with 2 Axes>,
 'mo_sin_old_rev_tl_op': <Figure size 432x288 with 2 Axes>,
 'mo_sin_rcnt_rev_tl_op': <Figure size 432x288 with 2 Axes>,
 'mort_acc': <Figure size 432x288 with 2 Axes>,
 'mths_since_last_major_derog': <Figure size 432x288 with 2 Axes>,
 'mths_since_last_record': <Figure size 432x288 with 2 Axes>,
 'mths_since_recent_inq': <Figure size 432x288 with 2 Axes>,
 'mths_since_recent_revol_delinq': <Figure size 432x288 with 2 Axes>,
 'num_accts_ever_120_pd': <Figure size 432x288 with 2 Axes>,
 'num_actv_bc_tl': <Figure size 432x288 with 2 Axes>,
 'num_op_rev_tl': <Figure size 432x288 with 2 Axes>,
 'num_tl_120dpd_2m': <Figure size 432x288 with 2 Axes>,
 'num_tl_90g_dpd_24m': <Figure size 432x288 with 2 Axes>,
 'num_tl_op_past_12m': <Figure size 432x288 with 2 Axes>,
 'open_act_il': <Figure size 432x288 with 2 Axes>,
 'open_il_24m': <Figure size 432x288 with 2 Axes>,
 'open_rv_24m': <Figure size 432x288 with 2 Axes>,
 'out_prncp': <Figure size 432x288 with 2 Axes>,
 'percent_bc_gt_75': <Figure size 432x288 with 2 Axes>,
 'pub_rec_bankruptcies': <Figure size 432x288 with 2 Axes>,
 'purpose': <Figure size 432x288 with 2 Axes>,
 'tax_liens': <Figure size 432x288 with 2 Axes>,
 'term': <Figure size 432x288 with 2 Axes>,
 'tot_coll_amt': <Figure size 432x288 with 2 Axes>,
 'tot_cur_bal': <Figure size 432x288 with 2 Axes>,
 'tot_hi_cred_lim': <Figure size 432x288 with 2 Axes>,
 'total_acc': <Figure size 432x288 with 2 Axes>,
 'total_bc_limit': <Figure size 432x288 with 2 Axes>,
 'total_cu_tl': <Figure size 432x288 with 2 Axes>,
 'total_il_high_credit_limit': <Figure size 432x288 with 2 Axes>,
 'total_pymnt': <Figure size 432x288 with 2 Axes>,
 'total_rec_int': <Figure size 432x288 with 2 Axes>,
 'total_rev_hi_lim': <Figure size 432x288 with 2 Axes>,
 'zip_code': <Figure size 432x288 with 2 Axes>}
```

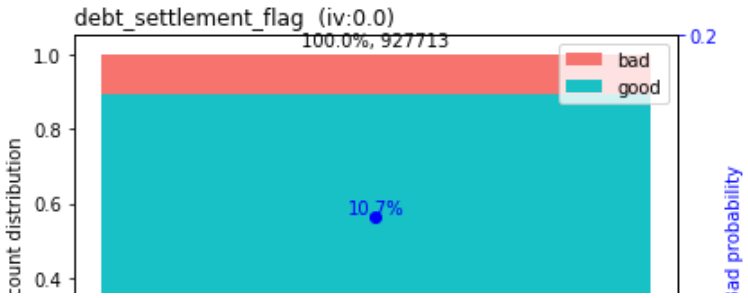


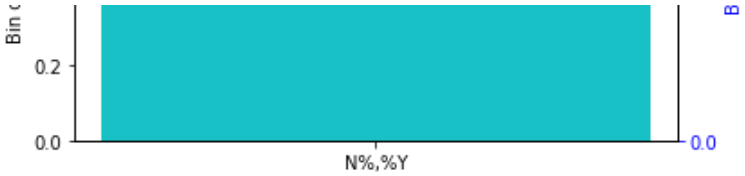

Bin count d
Bad pro
16.7%, 154961
8.8%, 81911
[-inf,2.0)
[2.0,7.0)
[7.0,9.0)
[9.0,inf)

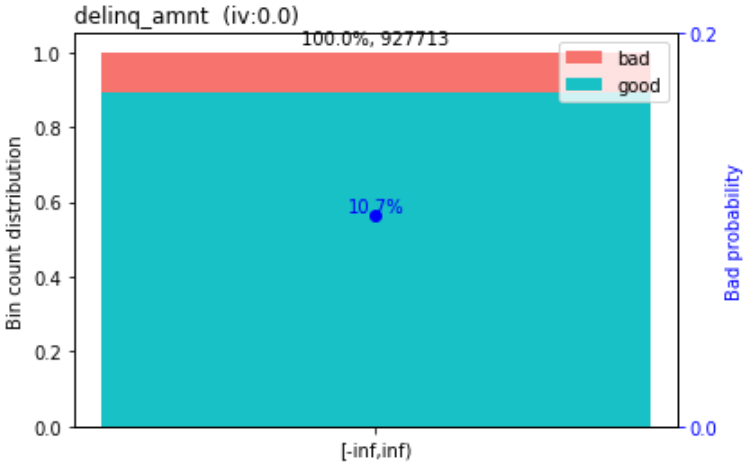

annual_inc (iv:0.0212)
bad
good
Bin count distribution
Bad probability
44.0%, 408409
11.9%
31.0%, 287745
10.7%
9.1%
7.8%
16.3%, 151146
8.7%, 80413
[-inf,60000.0)
[60000.0,90000.0)
[90000.0,130000.0)
[130000.0,inf)

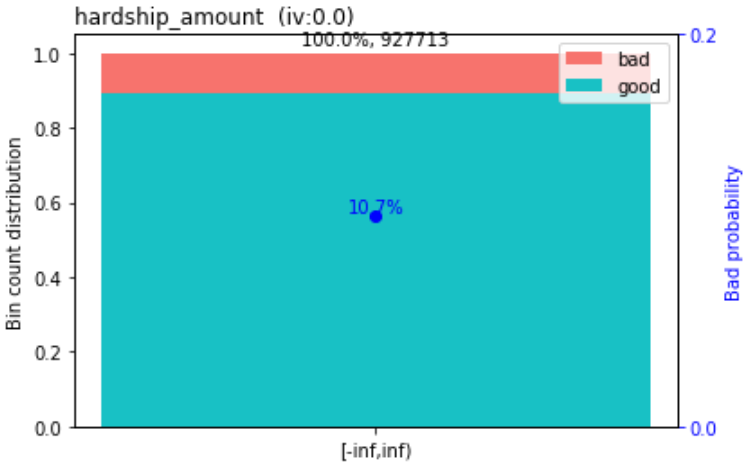

all_util (iv:0.1564)
bad
good
Bin count distribution
Bad probability
15.3%
35.2%, 326981
5.4%
12.7%, 117848
21.3%, 197983
30.7%, 284901
[-inf,0.0)
[0.0,40.0)
[40.0,60.0)
[60.0,inf)

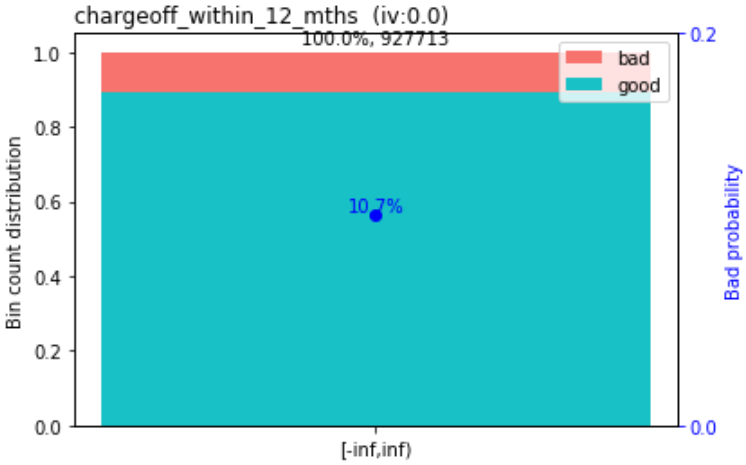

inq_fi (iv:0.1305)
bad
good
Bin count distribution
Bad probability
49.9%, 463333
15.3%
35.2%, 326958
7.6%
10.2%
14.8%, 137422
[-inf,0.0)
[0.0,2.0)
[2.0,inf)

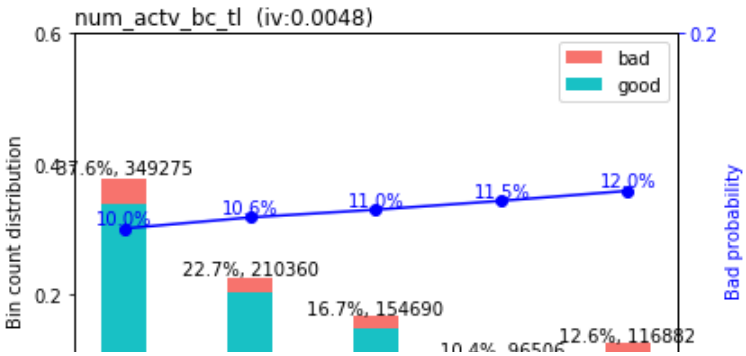

debt_settlement_flag (iv:0.0)
100.0%, 927713
bad
good
10.7%


N%,%Y

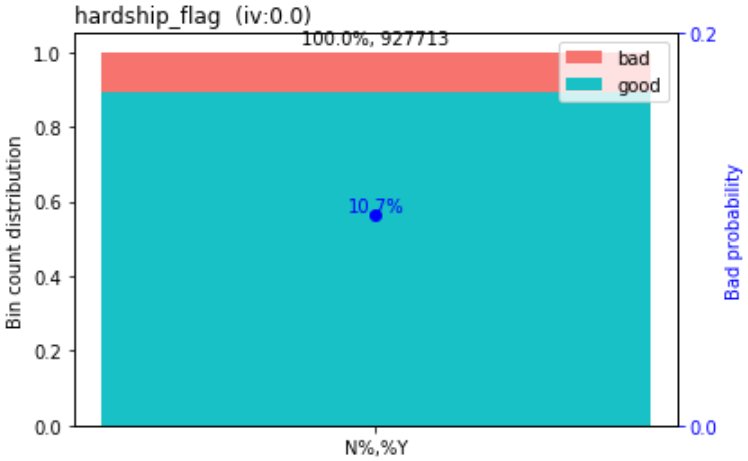

delinq_amnt (iv:0.0)
100.0%, 927713
bad
good
10.7%
Bin count distribution
Bad probability
[-inf,inf)

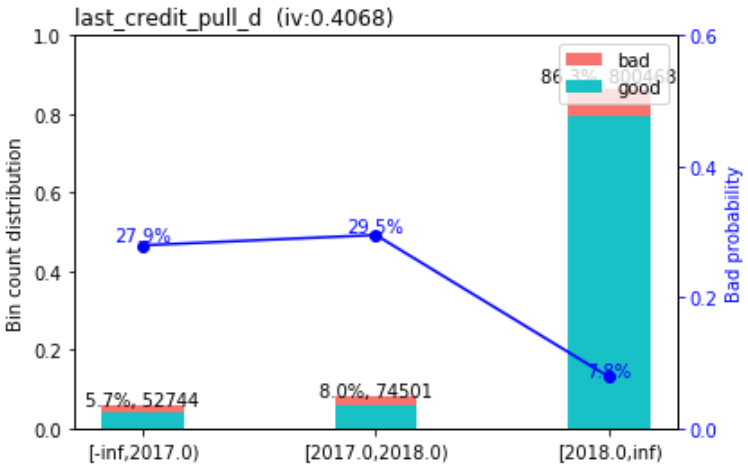

hardship_amount (iv:0.0)
100.0%, 927713
bad
good
10.7%
Bin count distribution
Bad probability
[-inf,inf)

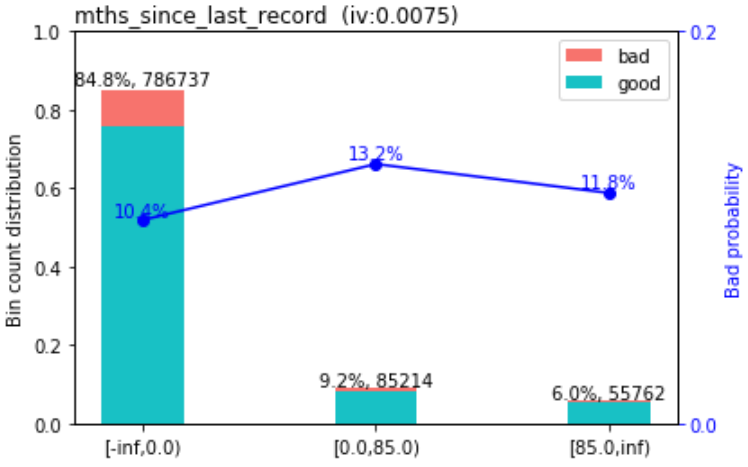

chargeoff_within_12_mths (iv:0.0)
100.0%, 927713
bad
good
10.7%
Bin count distribution
Bad probability
[-inf,inf)

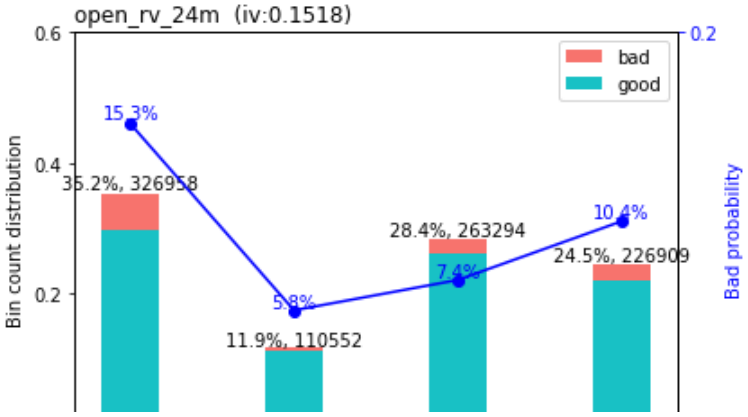

num_actv_bc_tl (iv:0.0048)
bad
good
37.6%, 349275
10.0%
10.6%
11.0%
11.5%
12.0%
22.7%, 210360
16.7%, 154690
12.6%, 116882
Bin count distribution
Bad probability

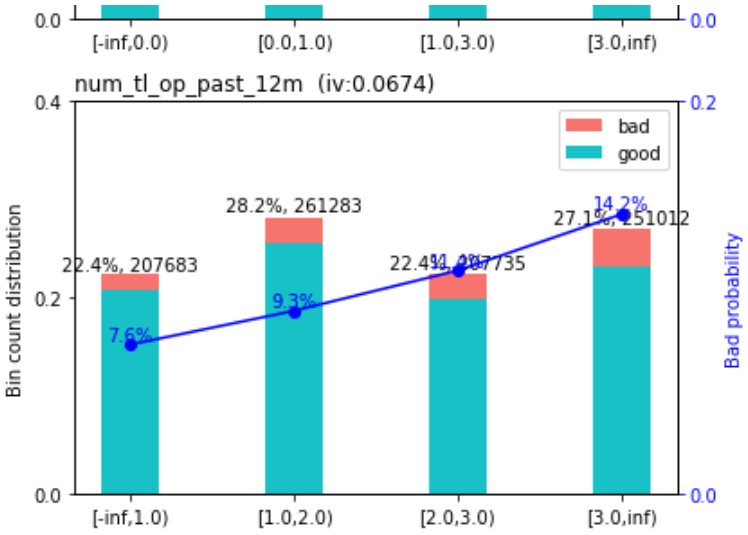

0.0
[-inf,3.0)
[3.0,4.0)
[4.0,5.0)
[5.0,6.0)
[6.0,inf)
0.0


hardship_flag (iv:0.0)
100.0%, 927713
bad
good
10.7%
Bin count distribution
Bad probability
N%,%Y

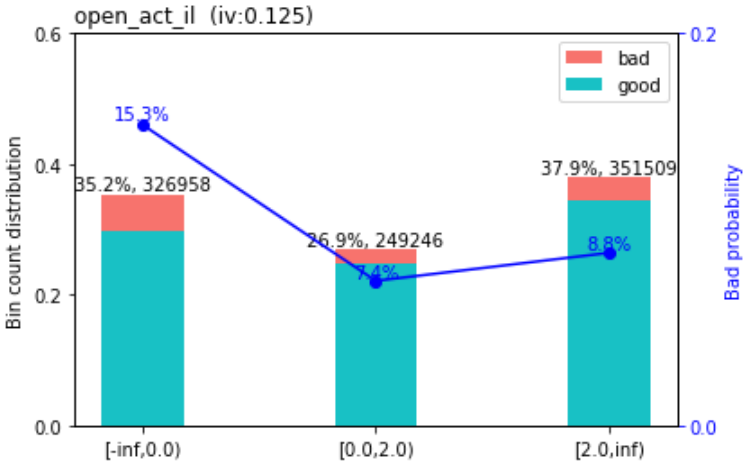

last_credit_pull_d (iv:0.4068)
bad
good
27.9%
29.5%
5.7%, 52744
8.0%, 74501
[-inf,2017.0)
[2017.0,2018.0)
[2018.0,inf)
Bin count distribution
Bad probability

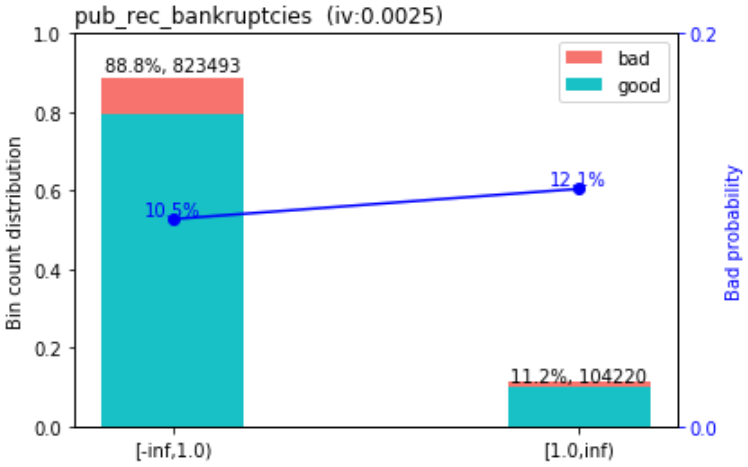

mths_since_last_record (iv:0.0075)
bad
good
84.8%, 786737
13.2%
11.8%
10.4%
9.2%, 85214
6.0%, 55762
[-inf,0.0)
[0.0,85.0)
[85.0,inf)
Bin count distribution
Bad probability

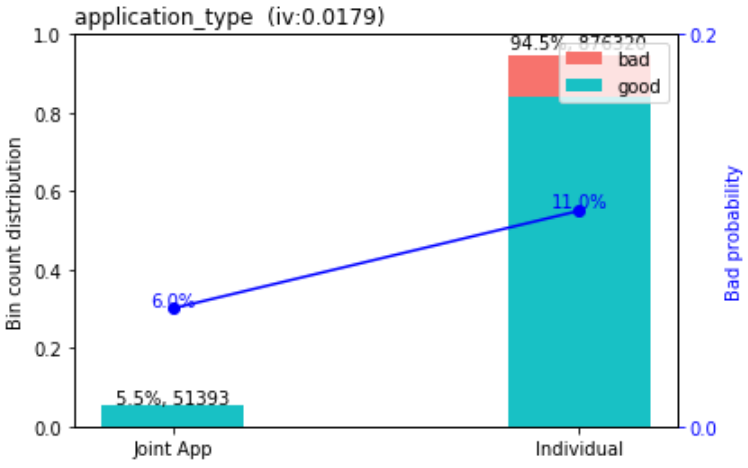

open_rv_24m (iv:0.1518)
bad
good
15.3%
35.2%, 326958
28.4%, 263294
10.4%
24.5%, 226909
7.4%
5.8%
11.9%, 110552
Bin count distribution
Bad probability

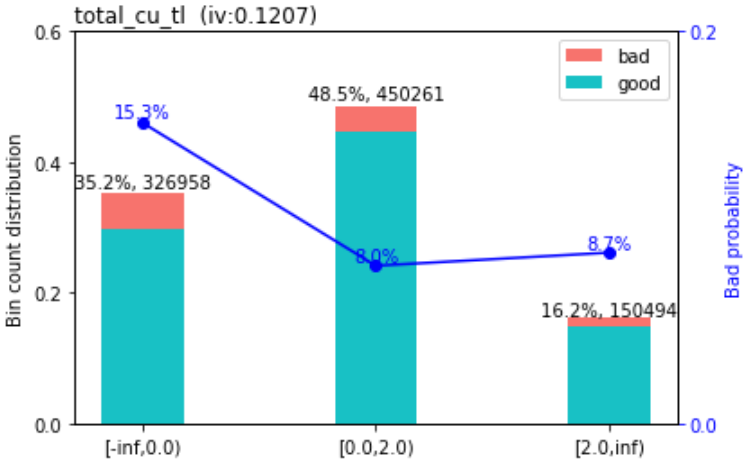

[-inf,0.0)
[0.0,1.0)
[1.0,3.0)
[3.0,inf)

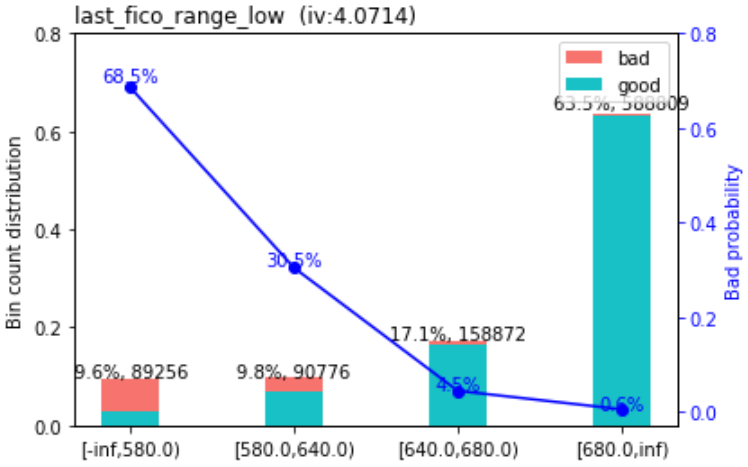
num_tl_op_past_12m (iv:0.0674)
bad
good
Bin count distribution
Bad probability
22.4%, 207683
28.2%, 261283
7.6%
9.3%
14.2%
[-inf,1.0)
[1.0,2.0)
[2.0,3.0)
[3.0,inf)

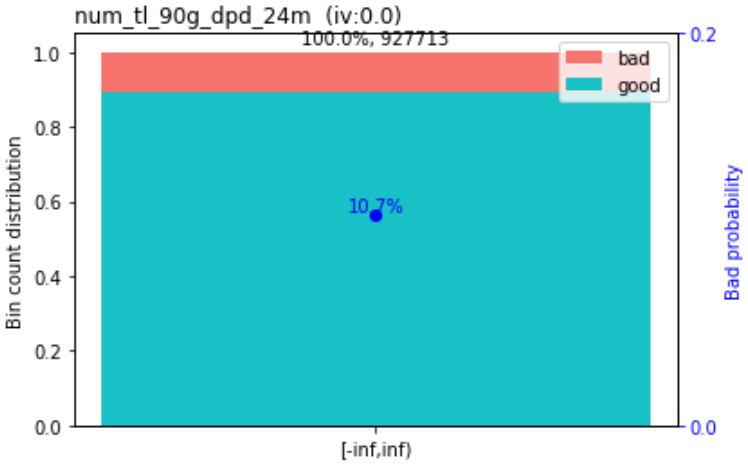

open_act_il (iv:0.125)
bad
good
Bin count distribution
Bad probability
15.3%
35.2%, 326958
26.9%, 249246
37.9%, 351509
8.8%
[-inf,0.0)
[0.0,2.0)
[2.0,inf)

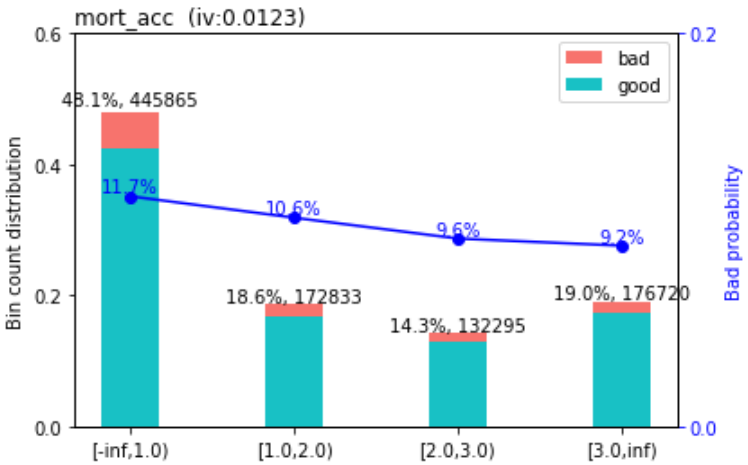

pub_rec_bankruptcies (iv:0.0025)
bad
good
Bin count distribution
Bad probability
88.8%, 823493
10.5%
12.1%
11.2%, 104220
[-inf,1.0)
[1.0,inf)

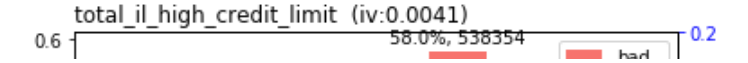

application_type (iv:0.0179)
bad
good
Bin count distribution
Bad probability
94.5%
11.0%
6.0%
5.5%, 51393
Joint App
Individual

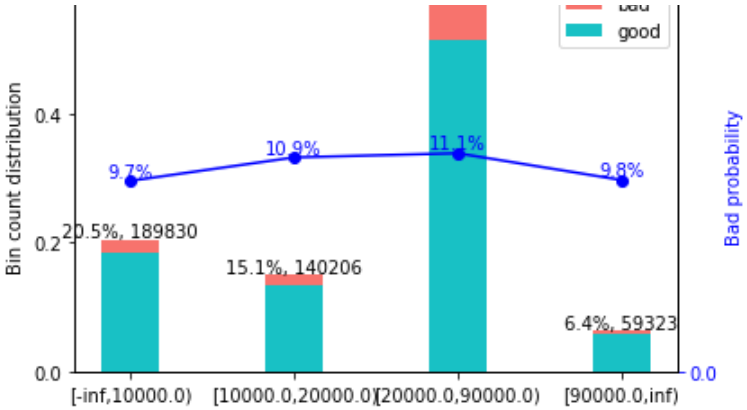

total_cu_tl (iv:0.1207)
bad
good
Bin count distribution
Bad probability
15.3%
48.5%, 450261
35.2%, 326958
8.0%
8.7%
16.2%, 150494
[-inf,0.0)
[0.0,2.0)
[2.0,inf)


last_fico_range_low (iv:4.0714)
bad
good
Bin count distribution
Bad probability
68.5%
30.5%
17.1%, 158872
9.6%, 89256
9.8%, 90776
4.5%
0.6%
[-inf,580.0)
[580.0,640.0)
[640.0,680.0)
[680.0,inf)

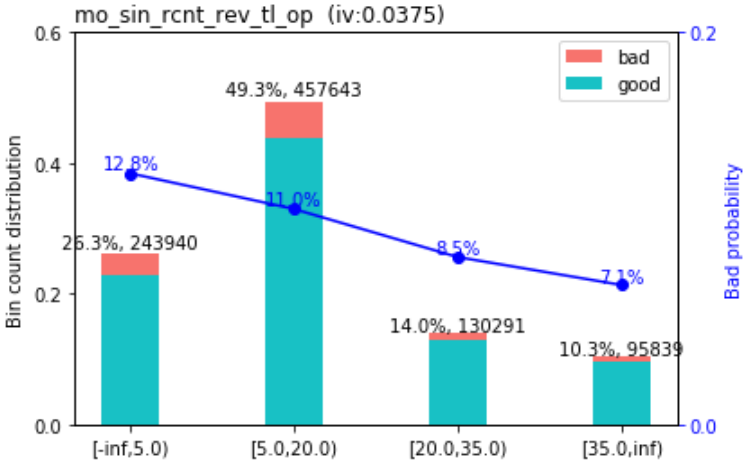

num_tl_90g_dpd_24m (iv:0.0)
100.0%, 927713
bad
good
Bin count distribution
Bad probability
10.7%
[-inf,inf)


mort_acc (iv:0.0123)
bad
good
Bin count distribution
Bad probability
48.1%, 445865
11.7%
10.6%
9.6%
9.2%
18.6%, 172833
14.3%, 132295
19.0%, 176720
[-inf,1.0)
[1.0,2.0)
[2.0,3.0)
[3.0,inf)

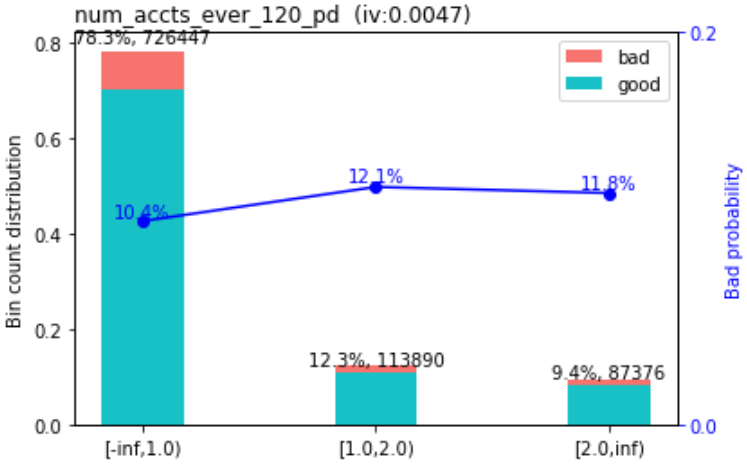

total_il_high_credit_limit (iv:0.0041)
58.0%, 538354



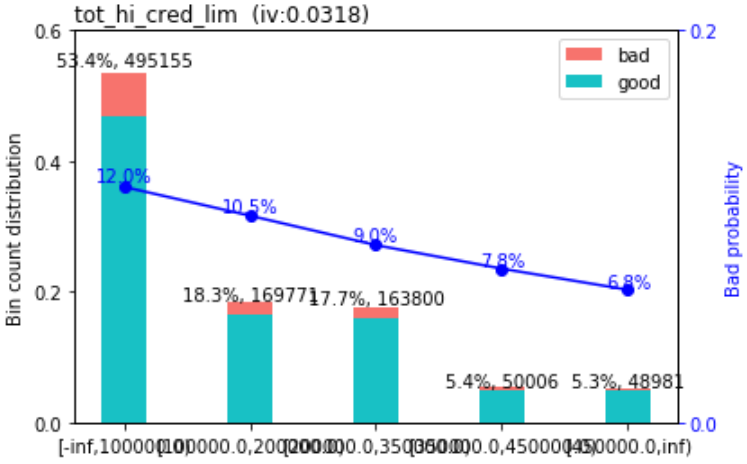
mo_sin_rcnt_rev_tl_op (iv:0.0375)



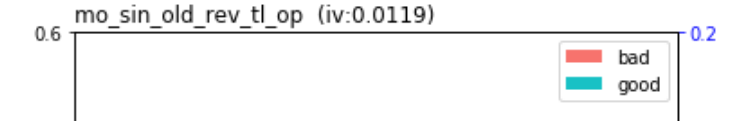
num_accts_ever_120_pd (iv:0.0047)

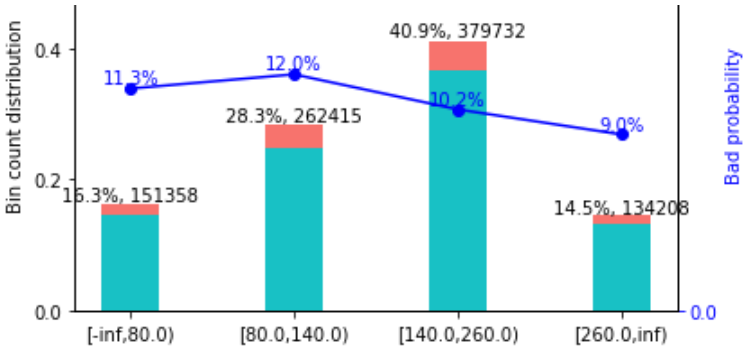


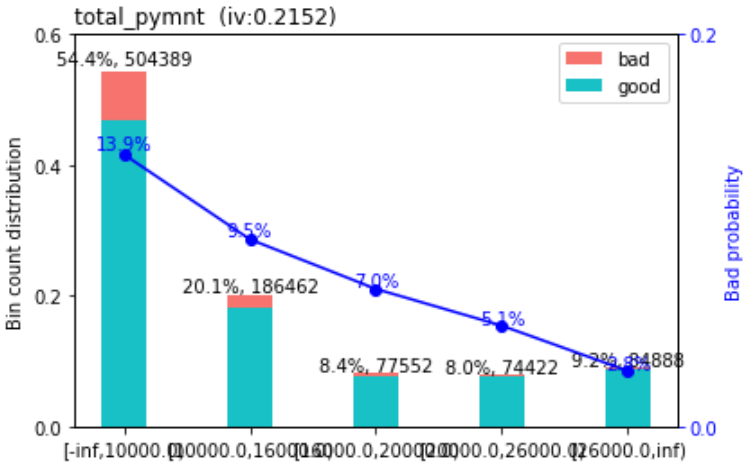
tot_hi_cred_lim (iv:0.0318)

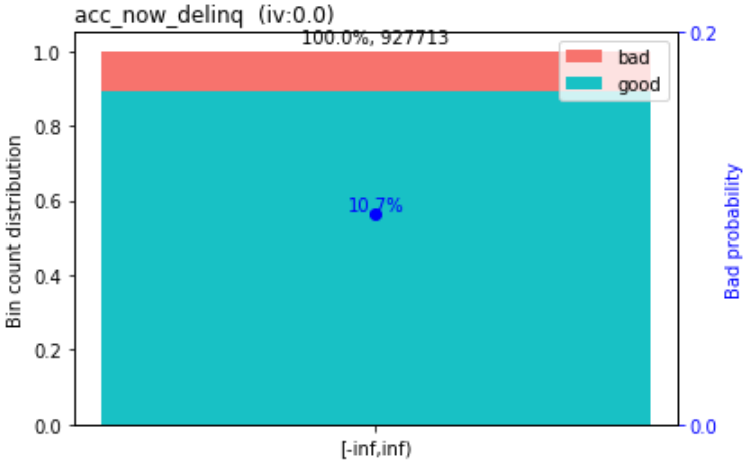


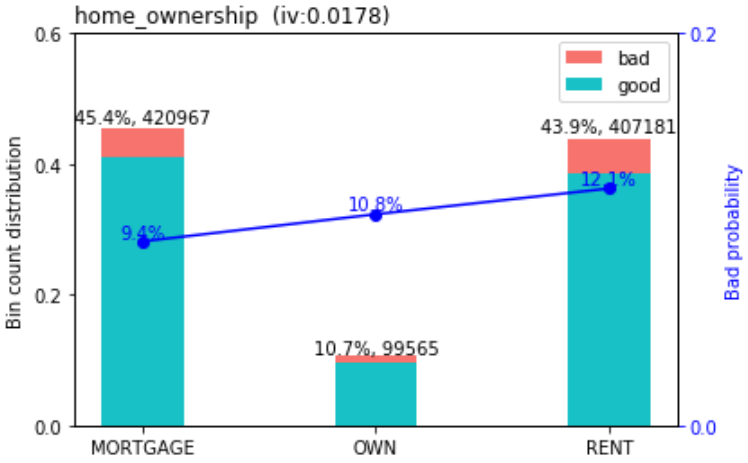
mo_sin_old_rev_tl_op (iv:0.0119)

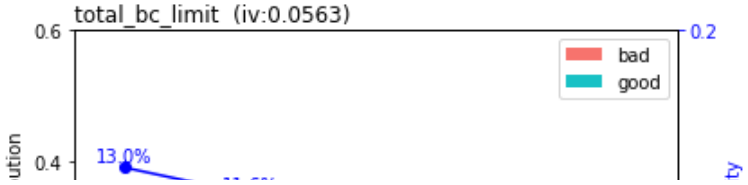


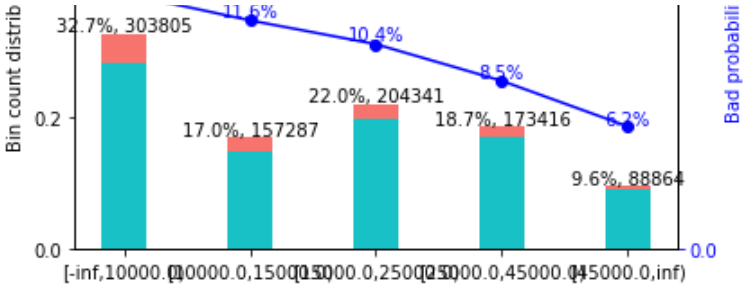

Bin count distribution
Bad probability
40.9%, 379732
11.3%
12.0%
10.2%
9.0%
28.3%, 262415
16.3%, 151358
14.5%, 134208
[-inf,80.0)
[80.0,140.0)
[140.0,260.0)
[260.0,inf)

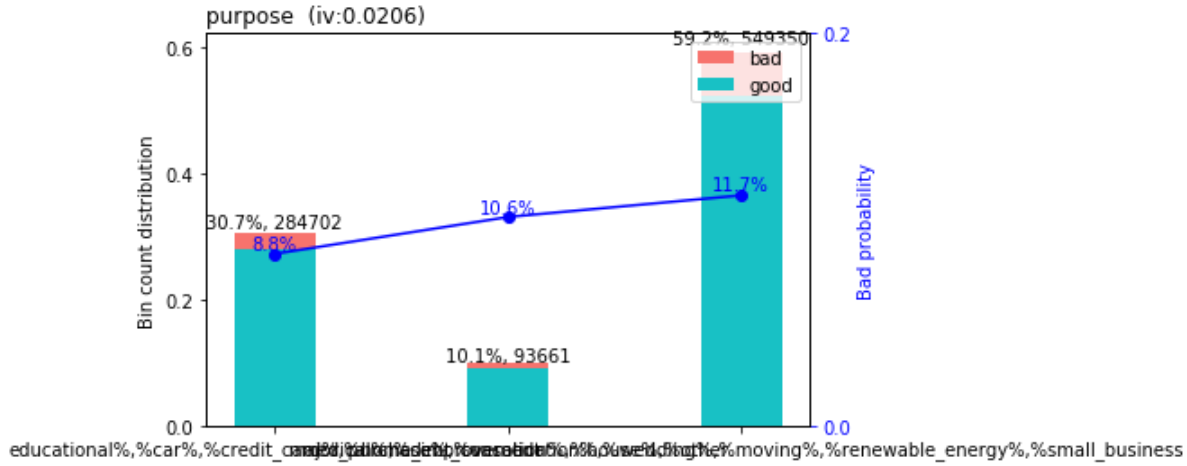

total_pymnt (iv:0.2152)
bad
good
Bin count distribution
Bad probability
54.4%, 504389
13.9%
9.5%
20.1%, 186462
7.0%
5.1%
8.4%, 77552
8.0%, 74422
[-inf,10000.0)
[26000.0,inf)

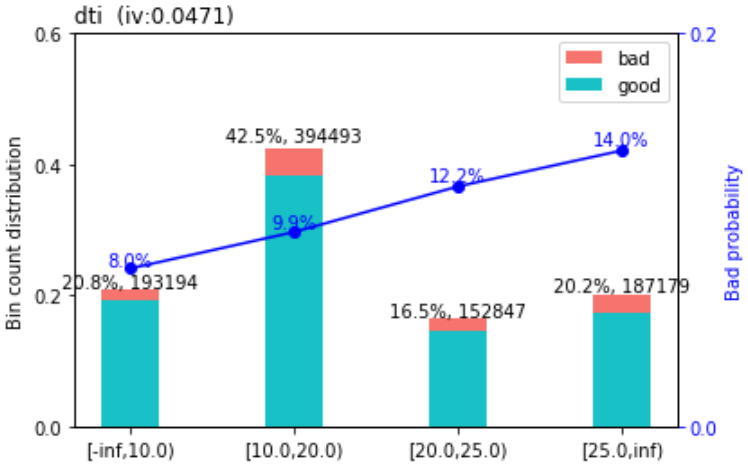

acc_now_delinq (iv:0.0)
bad
good
Bin count distribution
Bad probability
100.0%, 927713
10.7%
[-inf,inf)

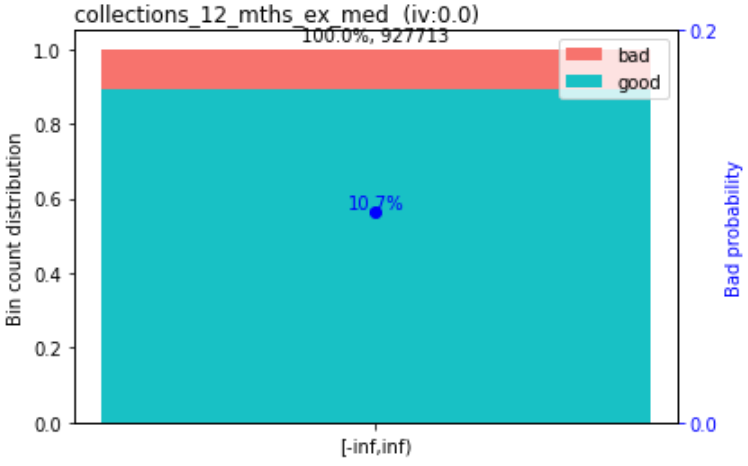

home_ownership (iv:0.0178)
bad
good
Bin count distribution
Bad probability
45.4%, 420967
43.9%, 407181
12.1%
10.8%
9.4%
10.7%, 99565
MORTGAGE
OWN
RENT

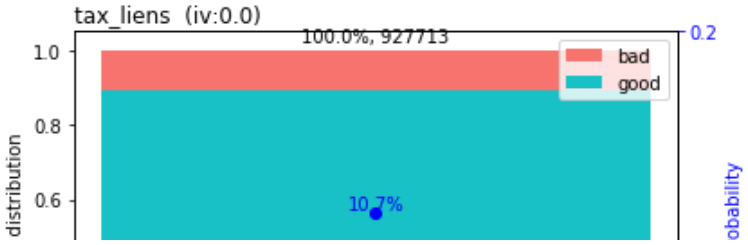

total_bc_limit (iv:0.0563)
bad
good
13.0%

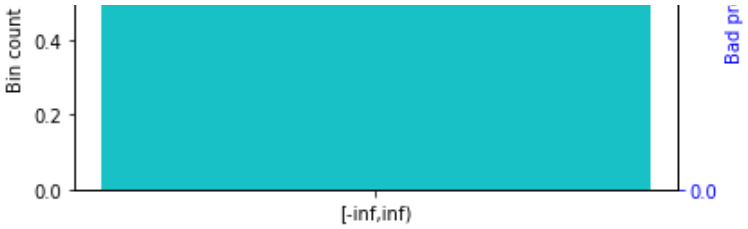

Bin count distrib
Bad probabili
32.7%, 303805
11.6%
10.4%
8.5%
6.2%
22.0%, 204341
17.0%, 157287
18.7%, 173416
9.6%, 88864
0.0
0.2
0.0
[-inf,10000.0)
[10000.0,15000.0)
[15000.0,25000.0)
[25000.0,45000.0)
[45000.0,inf)

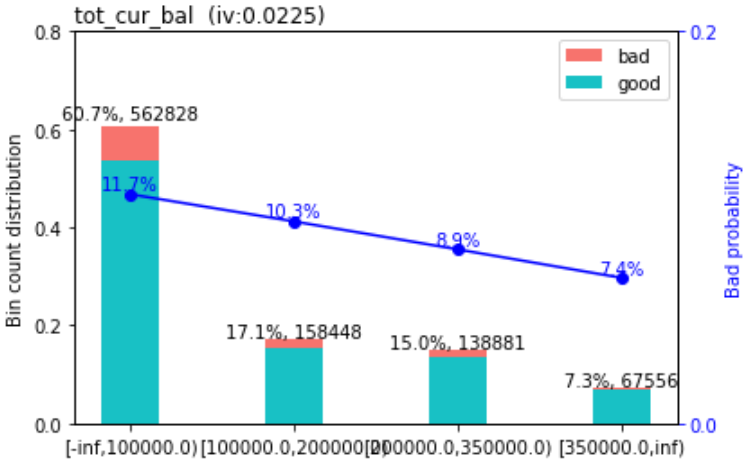

purpose (iv:0.0206)
Bin count distribution
Bad probability
bad
good
59.2%, 549350
30.7%, 284702
8.8%
10.6%
11.7%
10.1%, 93661
0.0
0.2
0.4
0.6
educational%,%car%,%credit_
moving%,%renewable_energy%,%small_business

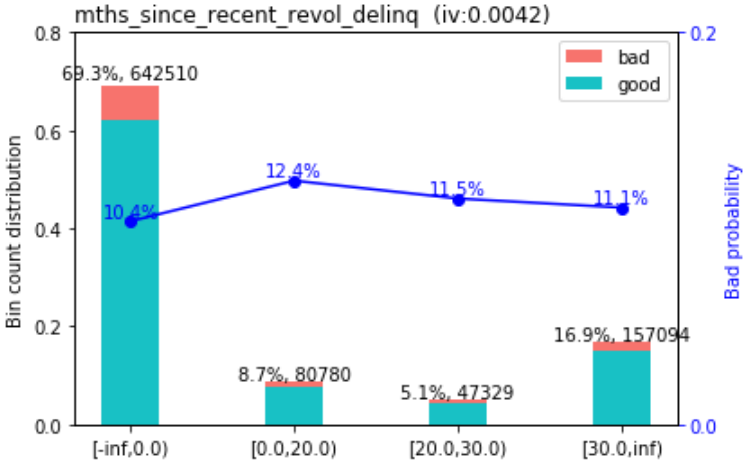

dti (iv:0.0471)
Bin count distribution
Bad probability
bad
good
42.5%, 394493
14.0%
12.2%
9.9%
8.0%
20.8%, 193194
16.5%, 152847
20.2%, 187179
0.0
0.2
0.4
0.6
[-inf,10.0)
[10.0,20.0)
[20.0,25.0)
[25.0,inf)

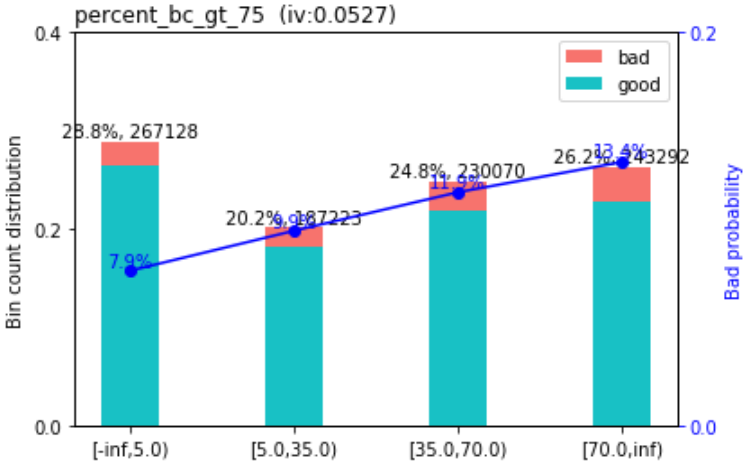

collections_12_mths_ex_med (iv:0.0)
Bin count distribution
Bad probability
bad
good
100.0%, 927713
10.7%
0.0
0.2
0.4
0.6
0.8
1.0
[-inf,inf)

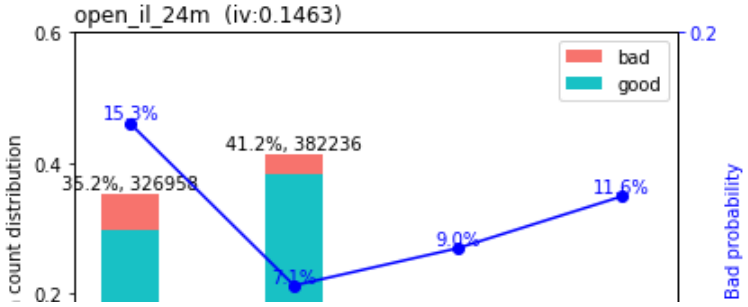

tax_liens (iv:0.0)
bad
good
100.0%, 927713
10.7%
0.6
0.8
1.0

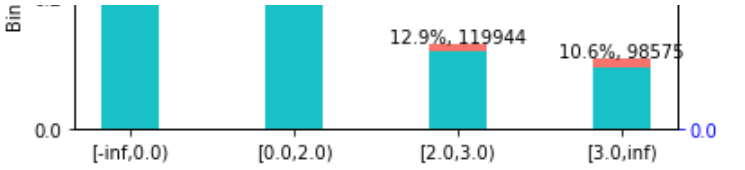

Bin count
Bad pr
0.0
0.2
0.4
[-inf,inf)

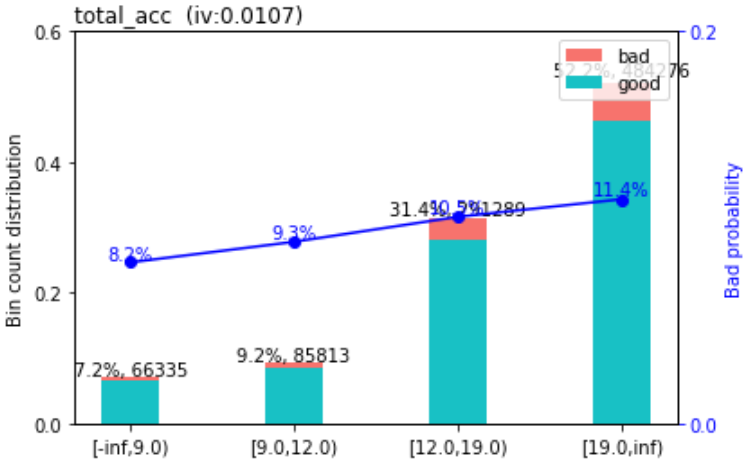

tot_cur_bal (iv:0.0225)
bad
good
Bin count distribution
Bad probability
60.7%, 562828
11.7%
10.3%
8.9%
7.4%
17.1%, 158448
15.0%, 138881
7.3%, 67556
[-inf,100000.0)
[100000.0,200000.0)
[200000.0,350000.0)
[350000.0,inf)

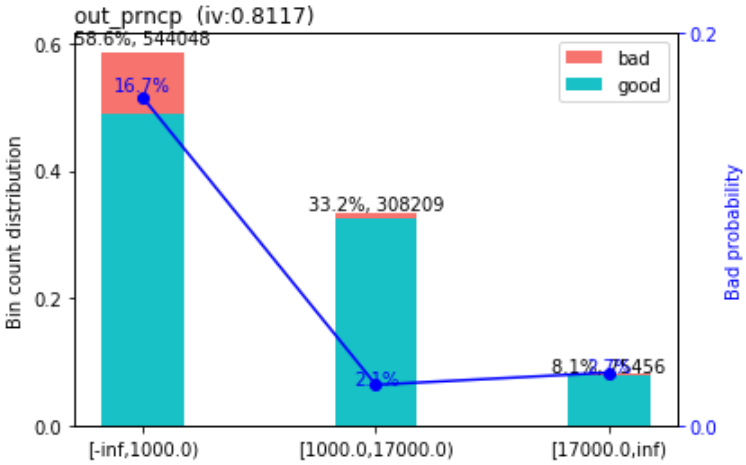

mths_since_recent_revol_delinq (iv:0.0042)
bad
good
Bin count distribution
Bad probability
69.3%, 642510
10.4%
12.4%
11.5%
11.1%
8.7%, 80780
5.1%, 47329
16.9%, 157094
[-inf,0.0)
[0.0,20.0)
[20.0,30.0)
[30.0,inf)

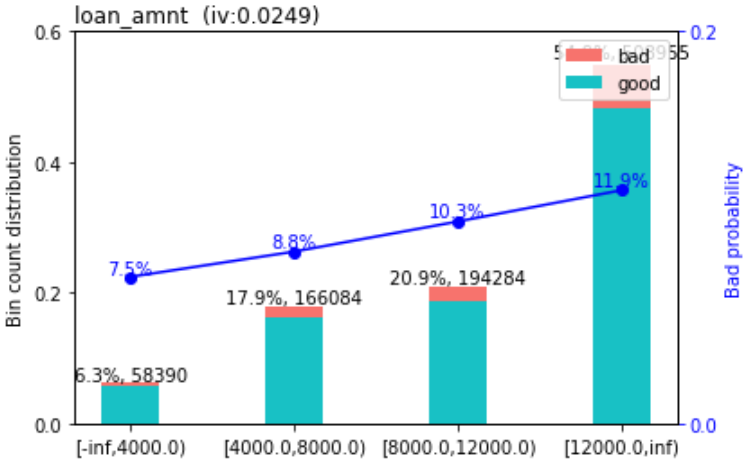

percent_bc_gt_75 (iv:0.0527)
bad
good
Bin count distribution
Bad probability
28.8%, 267128
7.9%
20.2%, 187223
24.8%, 230070
26.2%, 243292
[-inf,5.0)
[5.0,35.0)
[35.0,70.0)
[70.0,inf)

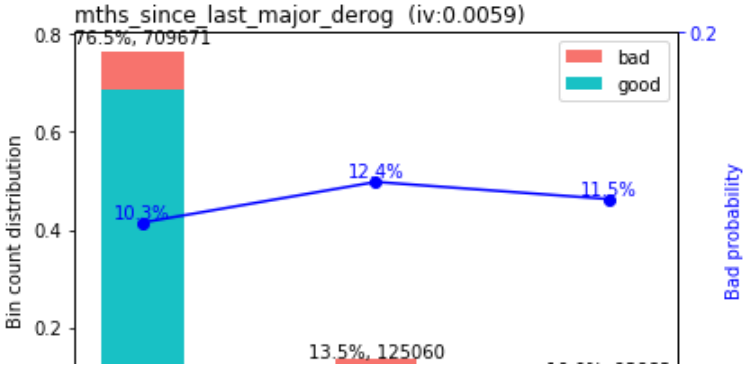

open_il_24m (iv:0.1463)
bad
good
Bad probability
15.3%
41.2%, 382236
35.2%, 326958
11.6%
9.0%


Bin
12.9%, 119944
10.6%, 98575
0.0
[-inf,0.0)
[0.0,2.0)
[2.0,3.0)
[3.0,inf)

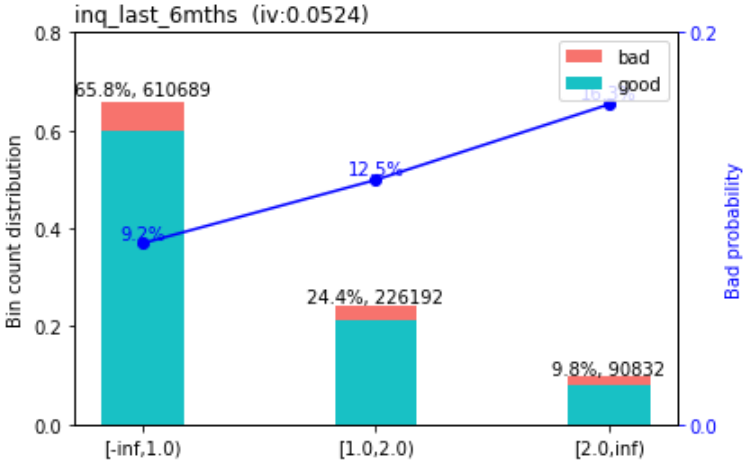

total_acc (iv:0.0107)
bad
good
Bin count distribution
Bad probability
8.2%
9.3%
11.4%
7.2%, 66335
9.2%, 85813
[-inf,9.0)
[9.0,12.0)
[12.0,19.0)
[19.0,inf)

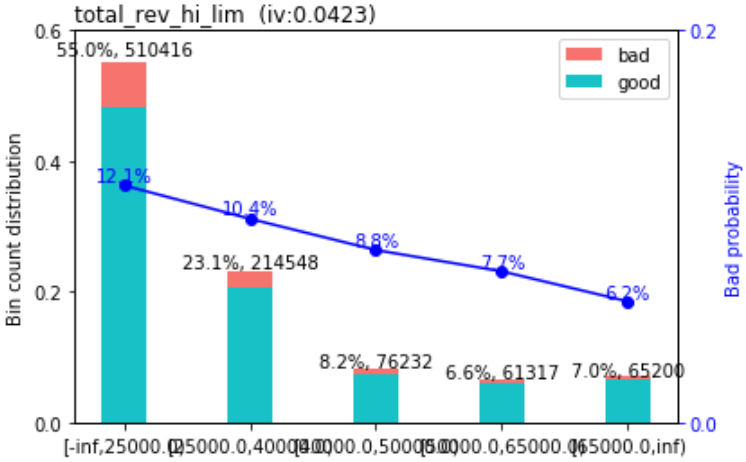

out_prncp (iv:0.8117)
58.6%, 544048
16.7%
33.2%, 308209
bad
good
Bin count distribution
Bad probability
[-inf,1000.0)
[1000.0,17000.0)
[17000.0,inf)

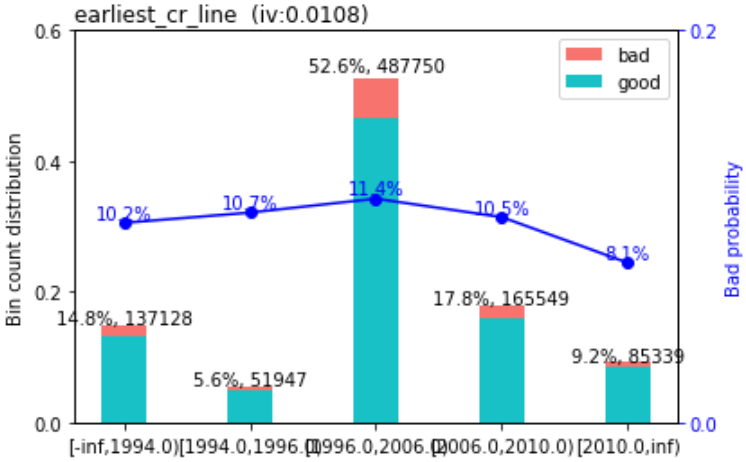

loan_amnt (iv:0.0249)
bad
good
Bin count distribution
Bad probability
7.5%
8.8%
10.3%
11.9%
6.3%, 58390
17.9%, 166084
20.9%, 194284
[-inf,4000.0)
[4000.0,8000.0)
[8000.0,12000.0)
[12000.0,inf)

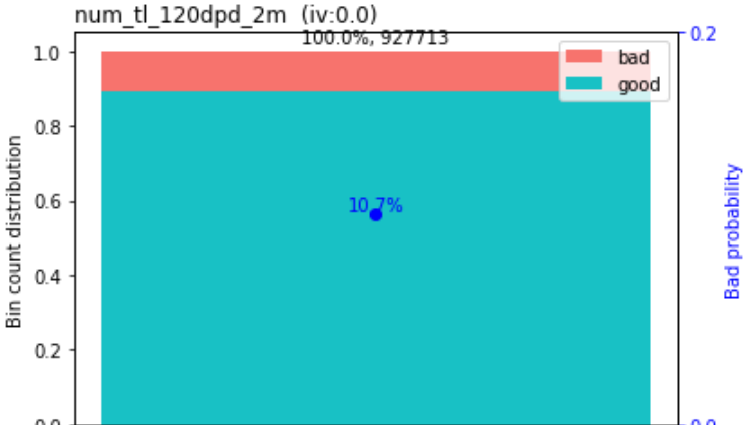

mths_since_last_major_derog (iv:0.0059)
76.5%, 709671
bad
good
Bin count distribution
Bad probability
10.3%
12.4%
11.5%
13.5%, 125060

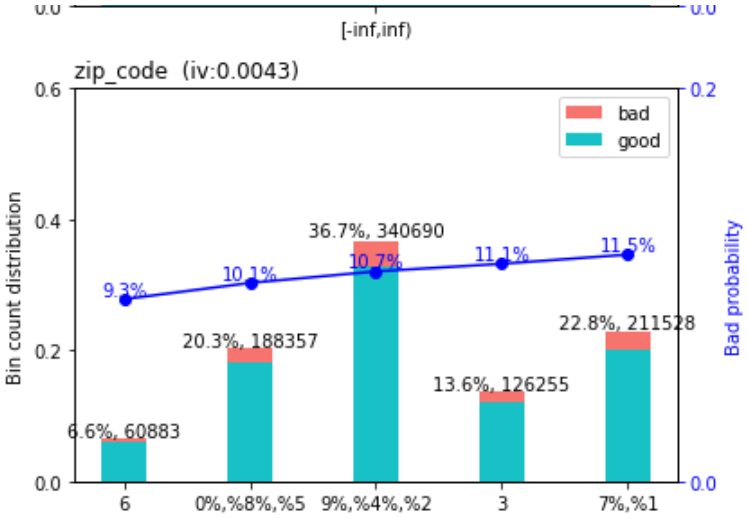

10.0%, 92982
0.0
[-inf,0.0)
[0.0,50.0)
[50.0,inf)
0.0


inq_last_6mths (iv:0.0524)
bad
good
65.8%, 610689
12.5%
9.2%
24.4%, 226192
9.8%, 90832
Bin count distribution
Bad probability
[-inf,1.0)
[1.0,2.0)
[2.0,inf)

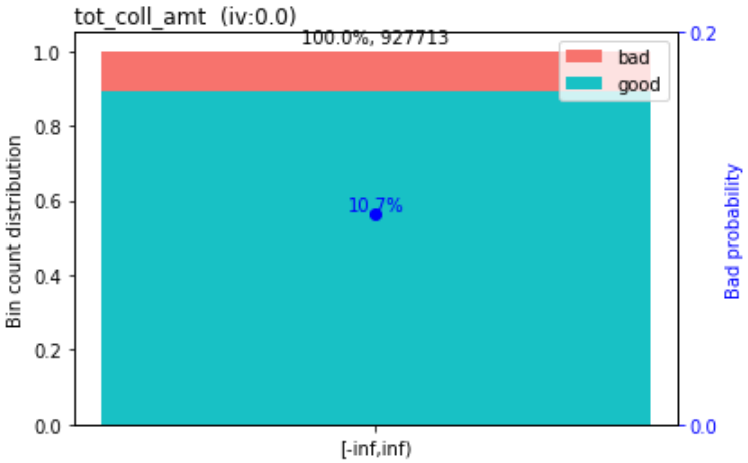

total_rev_hi_lim (iv:0.0423)
bad
good
55.0%, 510416
12.1%
10.4%
8.8%
7.7%
6.2%
23.1%, 214548
8.2%, 76232
6.6%, 61317
7.0%, 65200
Bin count distribution
Bad probability
[-inf,25000.0)
[65000.0,inf)

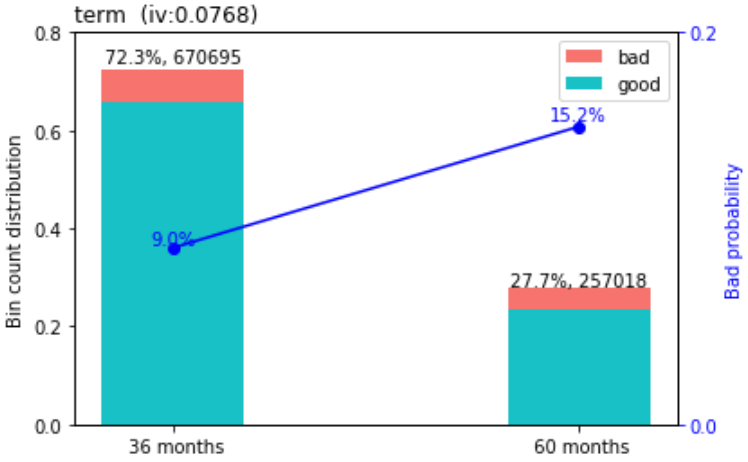

earliest_cr_line (iv:0.0108)
bad
good
52.6%, 487750
10.2%
10.7%
11.4%
10.5%
8.1%
14.8%, 137128
5.6%, 51947
17.8%, 165549
9.2%, 85339
Bin count distribution
Bad probability
[-inf,1994.0)
[2010.0,inf)

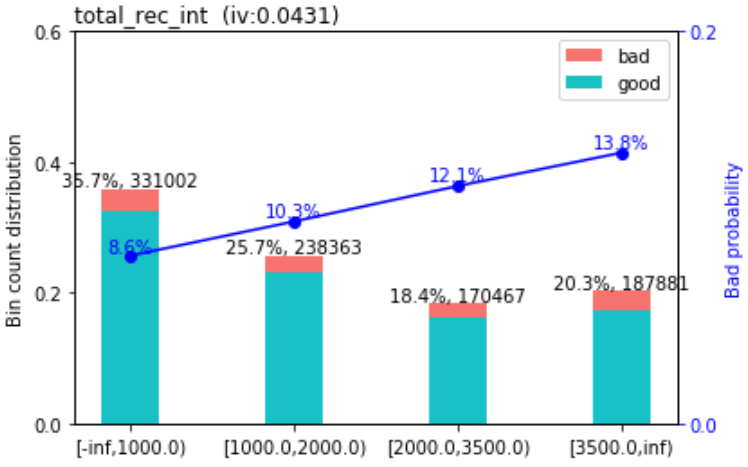

num_tl_120dpd_2m (iv:0.0)
100.0%, 927713
bad
good
10.7%
Bin count distribution
Bad probability

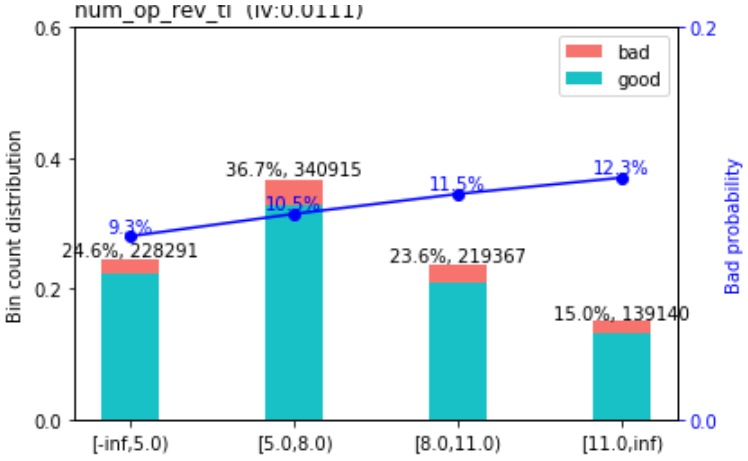

[-inf,inf)

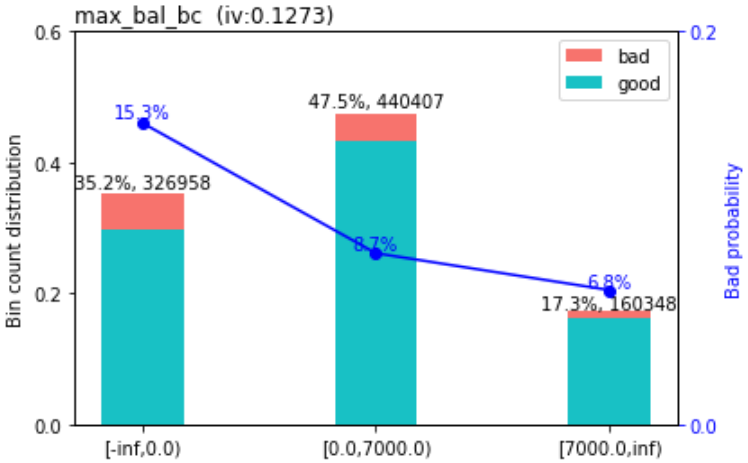
zip_code (iv:0.0043)
bad
good
Bin count distribution
Bad probability
9.3%
10.1%
10.7%
11.1%
11.5%
6.6%, 60883
20.3%, 188357
36.7%, 340690
13.6%, 126255
22.8%, 211528
6
0%,%8%,%5
9%,%4%,%2
3
7%,%1

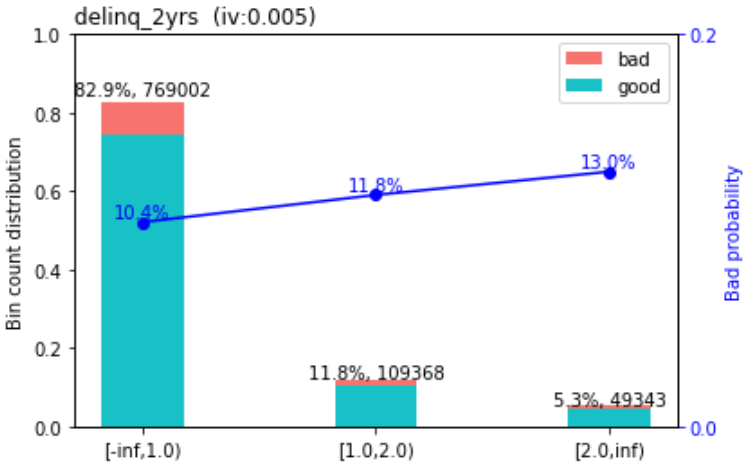

tot_coll_amt (iv:0.0)
bad
good
Bin count distribution
Bad probability
100.0%, 927713
10.7%
[-inf,inf)

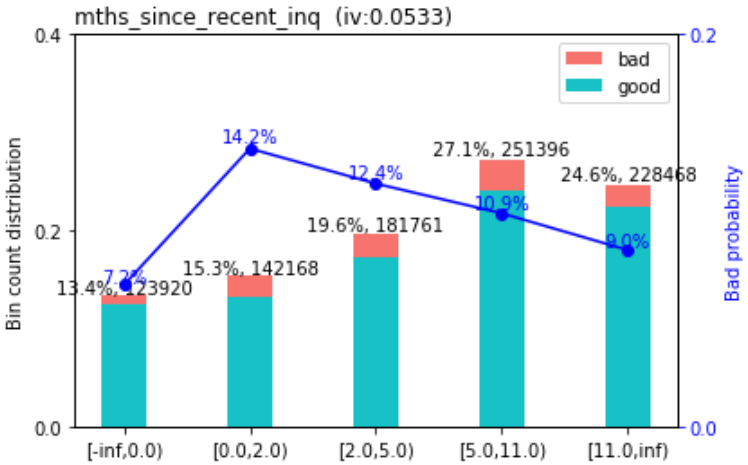

term (iv:0.0768)
bad
good
Bin count distribution
Bad probability
72.3%, 670695
9.0%
15.2%
27.7%, 257018
36 months
60 months


total_rec_int (iv:0.0431)
bad
good
Bin count distribution
Bad probability
35.7%, 331002
8.6%
10.3%
25.7%, 238363
12.1%
18.4%, 170467
13.8%
20.3%, 187881
[-inf,1000.0)
[1000.0,2000.0)
[2000.0,3500.0)
[3500.0,inf)


num_op_rev_tl (iv:0.0111)
bad
good
Bin count distribution
Bad probability
36.7%, 340915
11.5%
12.3%
9.3%
24.6%, 228291
23.6%, 219367
15.0%, 139140
[-inf,5.0)
[5.0,8.0)
[8.0,11.0)
[11.0,inf)

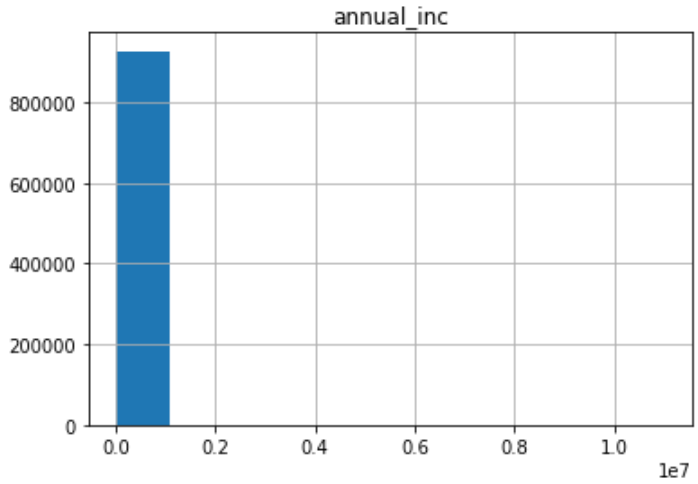

max_bal_bc (iv:0.1273)
bad
good
Bin count distribution
Bad probability
15.3%
47.5%, 440407
35.2%, 326958
8.7%
6.8%
17.3%, 160348
[-inf,0.0)
[0.0,7000.0)
[7000.0,inf)


delinq_2yrs (iv:0.005)
bad
good
Bin count distribution
Bad probability
82.9%, 769002
10.4%
11.8%
13.0%
11.8%, 109368
5.3%, 49343
[-inf,1.0)
[1.0,2.0)
[2.0,inf)

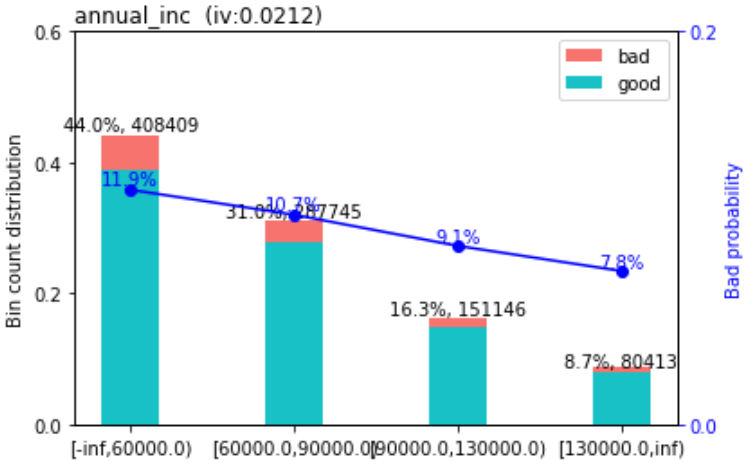

mths_since_recent_inq (iv:0.0533)
bad
good
Bin count distribution
Bad probability
14.2%
27.1%, 251396
12.4%
24.6%, 228468
10.9%
19.6%, 181761
9.0%
15.3%, 142168
7.2%
13.4%, 123920
[-inf,0.0)
[0.0,2.0)
[2.0,5.0)
[5.0,11.0)
[11.0,inf)

```
breaks_adj = sc.woebin_adj(train, "loan_status", bins)
```

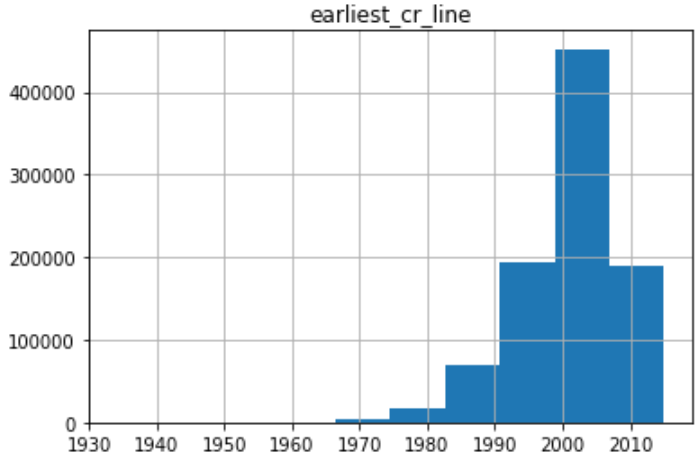
```
-------- 1/28 annual_inc --------
>>> dt[annual_inc].describe():
count    9.277130e+05
mean     7.432955e+04
std      6.959218e+04
min      0.000000e+00
25%      4.500000e+04
50%      6.300000e+04
75%      8.950000e+04
max      1.099920e+07
Name: annual_inc, dtype: float64
```



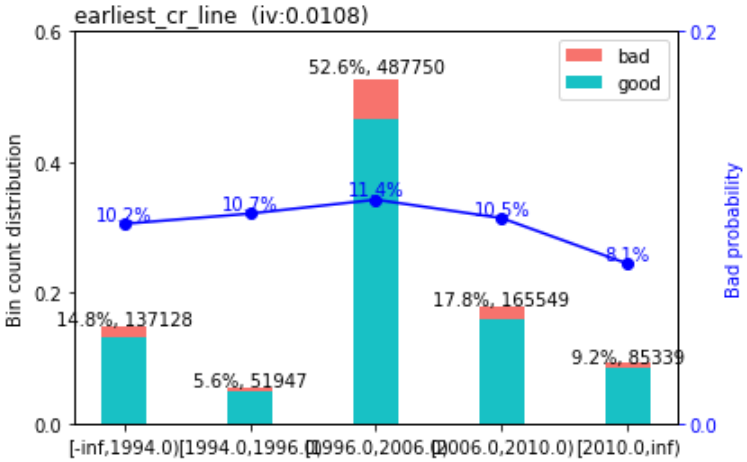

```
>>> Current breaks:
60000.0,90000.0,130000.0
```



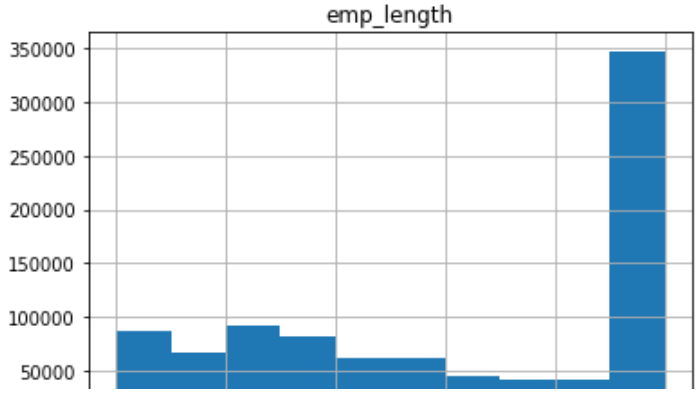

```
>>> Adjust breaks for (1/28) annual_inc?
1: next
2: yes
3: back
Selection: 1
-------- 2/28 earliest_cr_line --------
>>> dt[earliest_cr_line].describe():
count    927713.000000
mean       2000.822932
std           7.429960
min        1934.000000
25%        1997.000000
50%        2002.000000
75%        2006.000000
max        2015.000000
Name: earliest_cr_line, dtype: float64
```



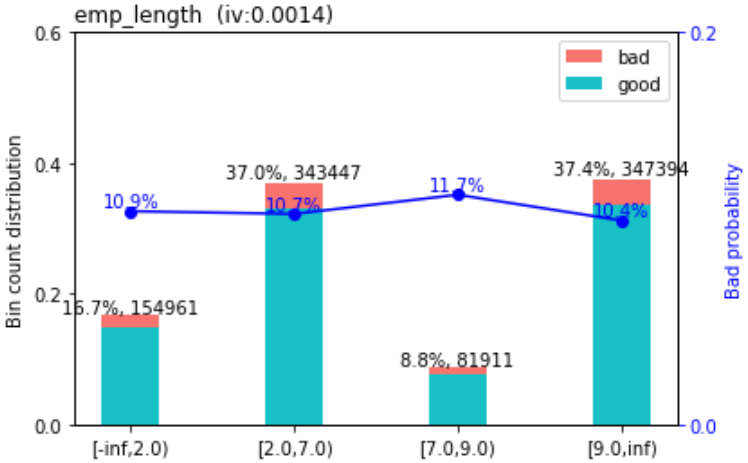

```
>>> Current breaks:
1994.0,1996.0,2006.0,2010.0
```



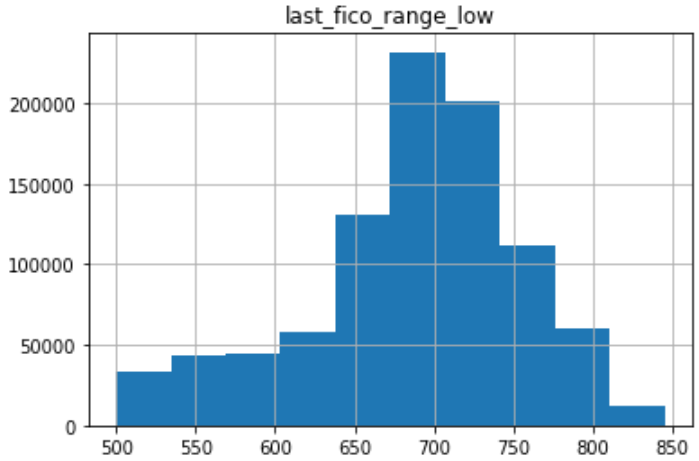

```
>>> Adjust breaks for (2/28) earliest_cr_line?
1: next
2: yes
3: back
Selection: 1
-------- 3/28 emp_length --------
>>> dt[emp_length].describe():
count    927713.000000
mean          5.800675
std           3.720349
min           0.000000
25%           2.000000
50%           6.000000
75%          10.000000
max          10.000000
Name: emp_length, dtype: float64
```



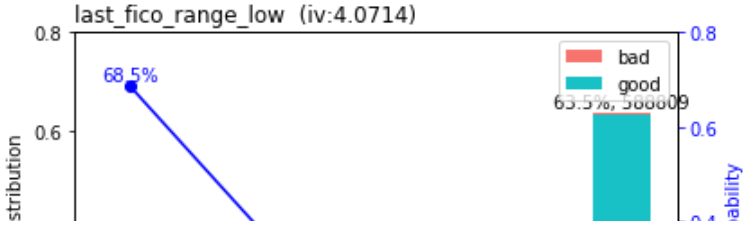

```
>>> Current breaks:
2.0,7.0,9.0
```

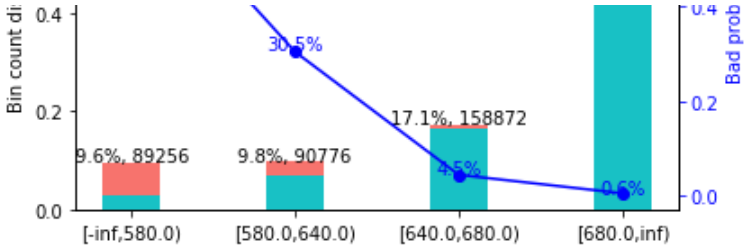



```
>>> Adjust breaks for (3/28) emp_length?
1: next
2: yes
3: back
Selection: 1
-------- 4/28 last_fico_range_low --------
>>> dt[last_fico_range_low].describe():
count    927713.000000
mean        687.347693
std          69.185637
min         500.000000
25%         655.000000
50%         695.000000
75%         730.000000
max         845.000000
Name: last_fico_range_low, dtype: float64
```

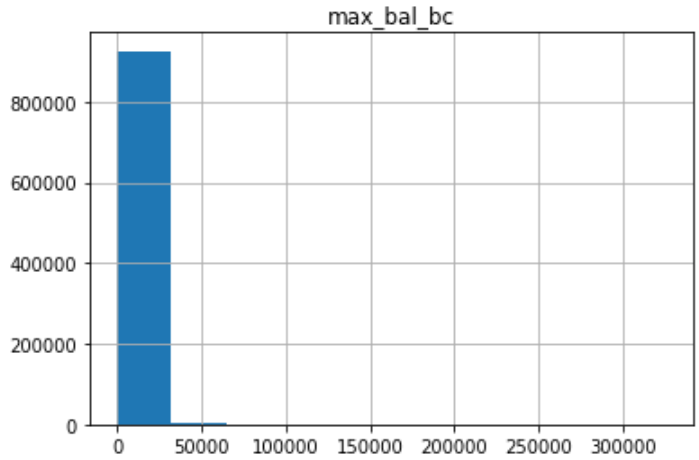



```
>>> Current breaks:
580.0,640.0,680.0
```

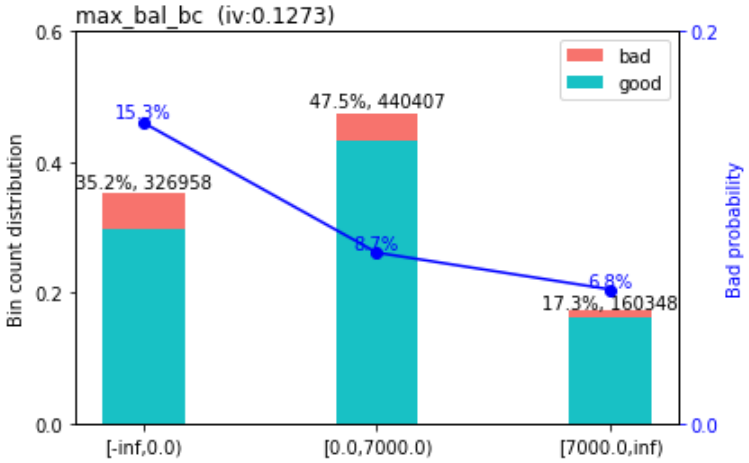





```
>>> Adjust breaks for (4/28) last_fico_range_low?
1: next
2: yes
3: back
Selection: 1
-------- 5/28 max_bal_bc --------
>>> dt[max_bal_bc].describe():
count    927713.000000
mean       3631.251179
std        4996.470003
min          -1.000000
25%          -1.000000
50%        2052.000000
75%        5351.000000
max      325000.000000
Name: max_bal_bc, dtype: float64
```

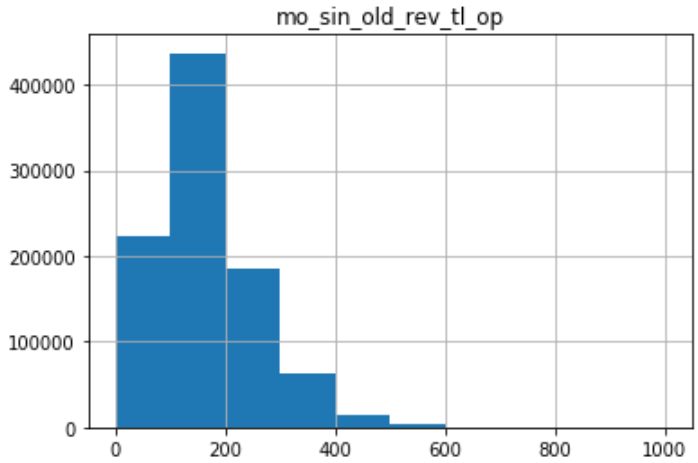



```
>>> Current breaks:
0.0,7000.0
```

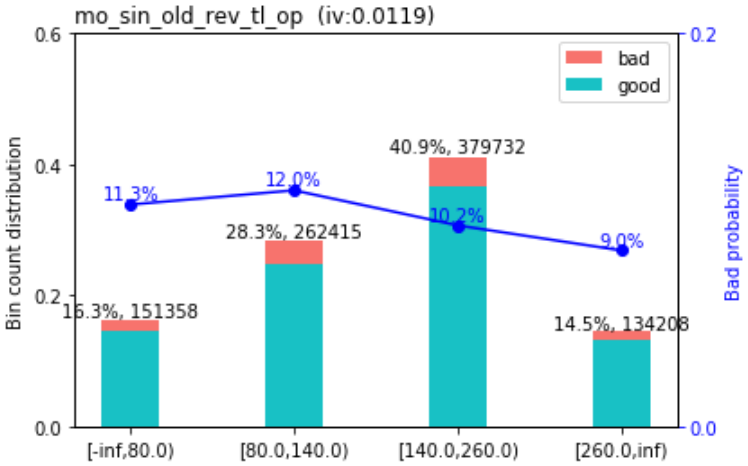



```
>>> Adjust breaks for (5/28) max_bal_bc?
1: next
2: yes
3: back
Selection: 1
```

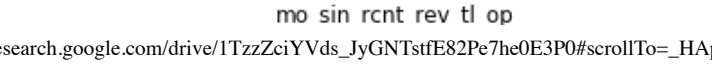
```
-------- 6/28 mo_sin_old_rev_tl_op --------
>>> dt[mo_sin_old_rev_tl_op].describe():
count    927713.000000
mean        165.716129
std          90.925495
min           1.000000
25%         103.000000
50%         149.000000
75%         213.000000
max         999.000000
Name: mo_sin_old_rev_tl_op, dtype: float64
```

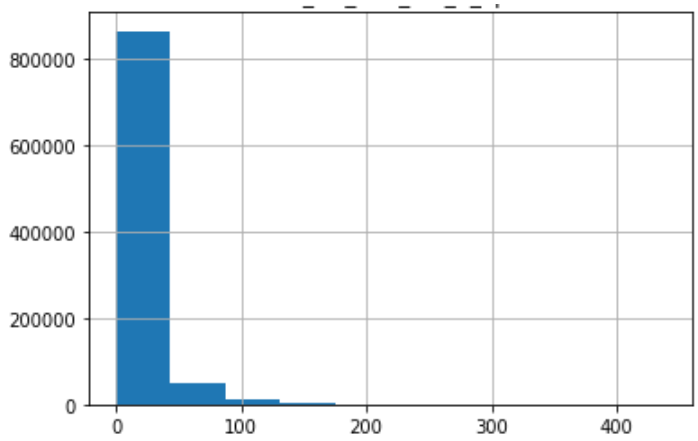



```
>>> Current breaks:
80.0,140.0,260.0
```

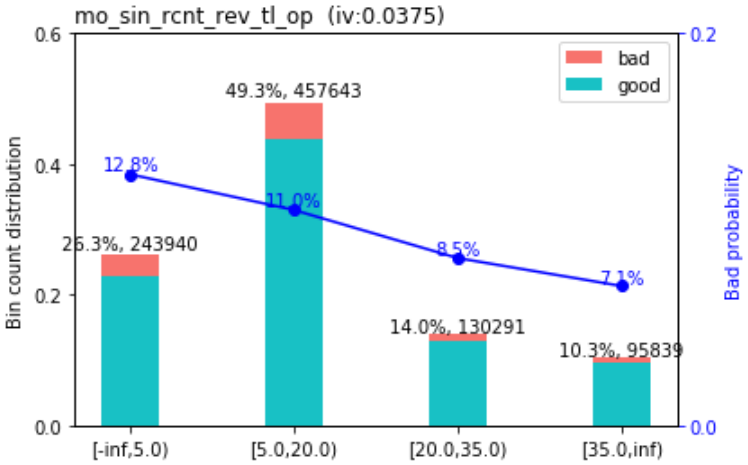



```
>>> Adjust breaks for (6/28) mo_sin_old_rev_tl_op?
1: next
2: yes
3: back
Selection: 1
-------- 7/28 mo_sin_rcnt_rev_tl_op --------
>>> dt[mo_sin_rcnt_rev_tl_op].describe():
count    927713.000000
mean         15.439982
std          18.667166
min           0.000000
25%           4.000000
50%           9.000000
75%          19.000000
max         438.000000
Name: mo_sin_rcnt_rev_tl_op, dtype: float64
```

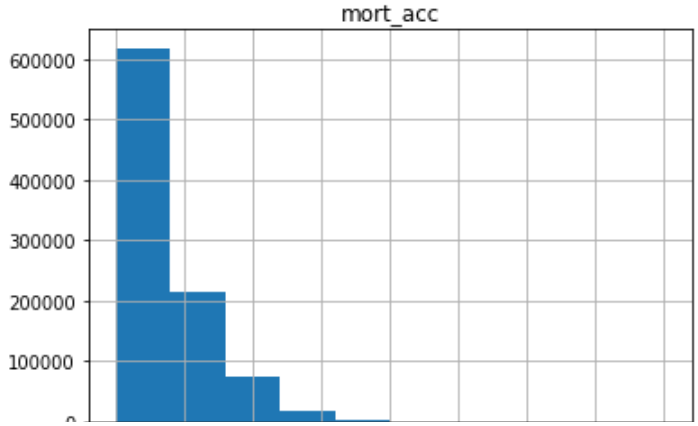


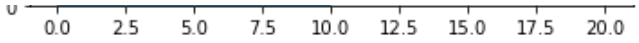



```
>>> Current breaks:
5.0,20.0,35.0
```

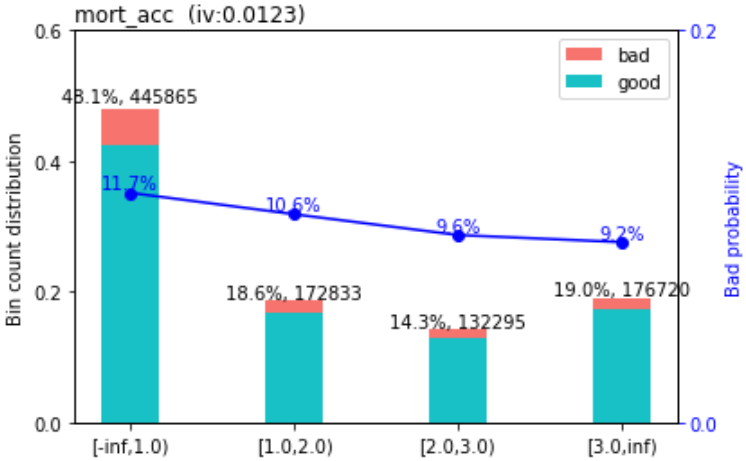



```
>>> Adjust breaks for (7/28) mo_sin_rcnt_rev_tl_op?
1: next
2: yes
3: back
Selection: 1
-------- 8/28 mort_acc --------
>>> dt[mort_acc].describe():
count    927713.000000
mean          1.229459
std           1.603452
min           0.000000
25%           0.000000
50%           1.000000
75%           2.000000
max          20.000000
Name: mort_acc, dtype: float64
```

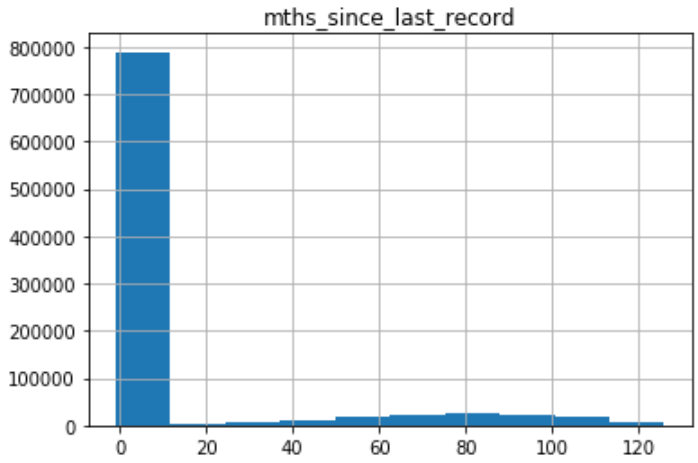




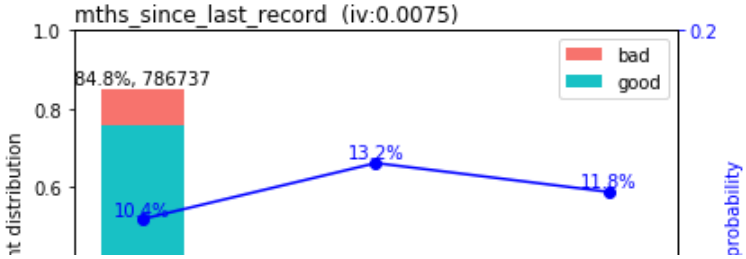

```
>>> Current breaks:
1.0,2.0,3.0
```

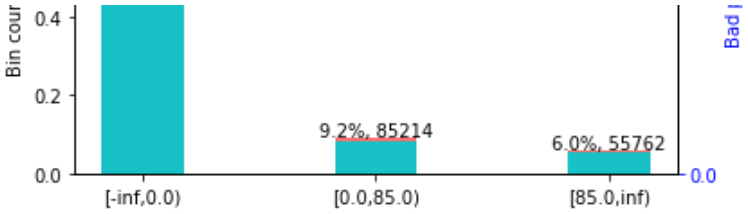



```
>>> Adjust breaks for (8/28) mort_acc?
1: next
2: yes
3: back
Selection: 1
-------- 9/28 mths_since_last_record --------
>>> dt[mths_since_last_record].describe():
count    927713.000000
mean         10.526609
std          29.101855
min          -1.000000
25%          -1.000000
50%          -1.000000
75%          -1.000000
max         126.000000
Name: mths_since_last_record, dtype: float64
```

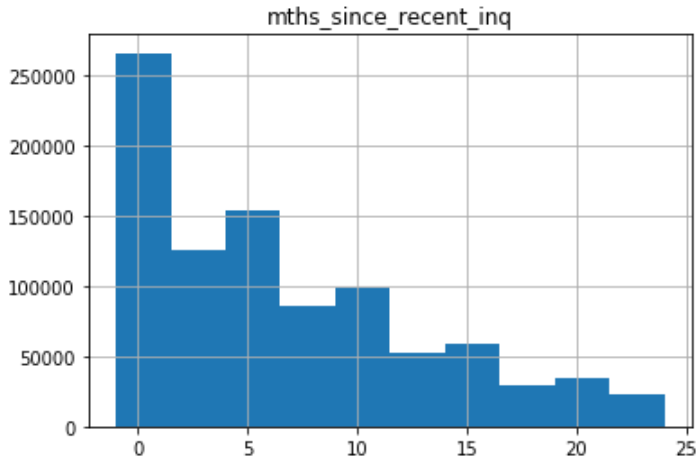



```
>>> Current breaks:
0.0,85.0
```

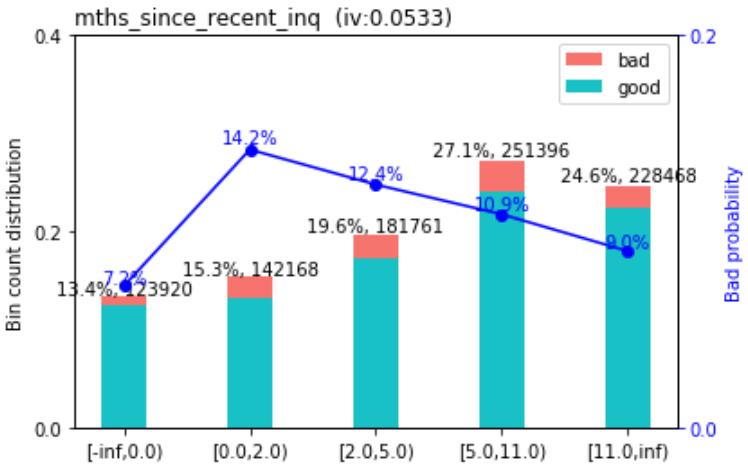





```
>>> Adjust breaks for (9/28) mths_since_last_record?
1: next
2: yes
3: back
Selection: 1
-------- 10/28 mths_since_recent_inq --------
>>> dt[mths_since_recent_inq].describe():
count    927713.000000
mean          6.367142
std           6.395820
min          -1.000000
25%           1.000000
50%           5.000000
75%          10.000000
max          24.000000
Name: mths_since_recent_inq, dtype: float64
```

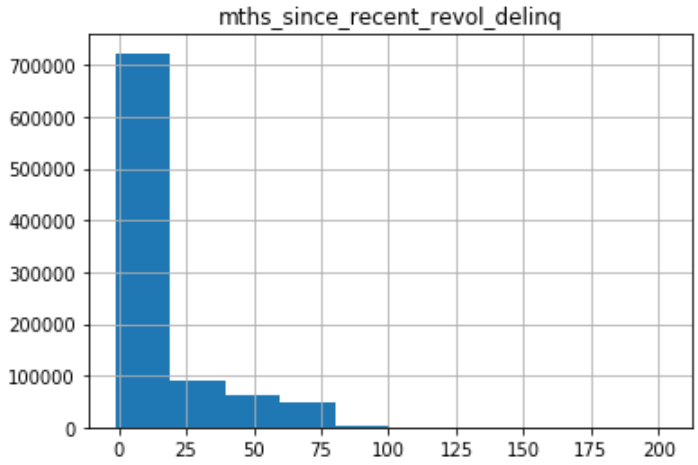



```
>>> Current breaks:
0.0,2.0,5.0,11.0
```

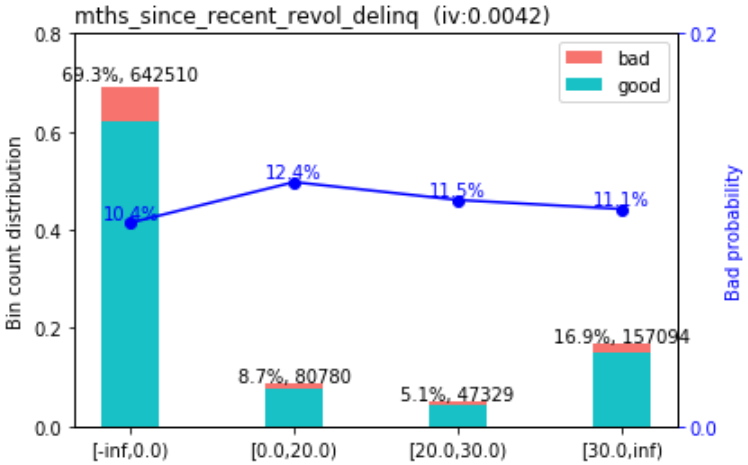



```
>>> Adjust breaks for (10/28) mths_since_recent_inq?
1: next
2: yes
3: back
Selection: 1
-------- 11/28 mths_since_recent_revol_deling --------
```

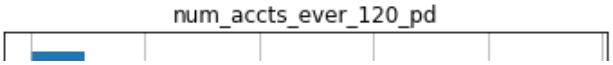
```
>>> dt[mths_since_recent_revol_delinq].describe():
count    927713.000000
mean         10.318800
std          20.988355
min          -1.000000
25%          -1.000000
50%          -1.000000
75%          14.000000
max         202.000000
Name: mths_since_recent_revol_delinq, dtype: float64
```

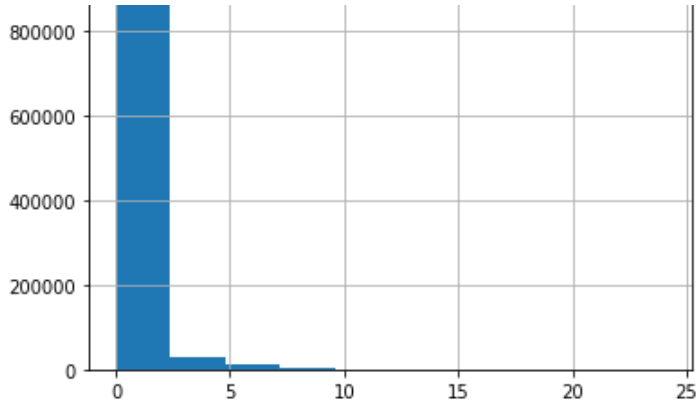



```
>>> Current breaks:
0.0,20.0,30.0
```

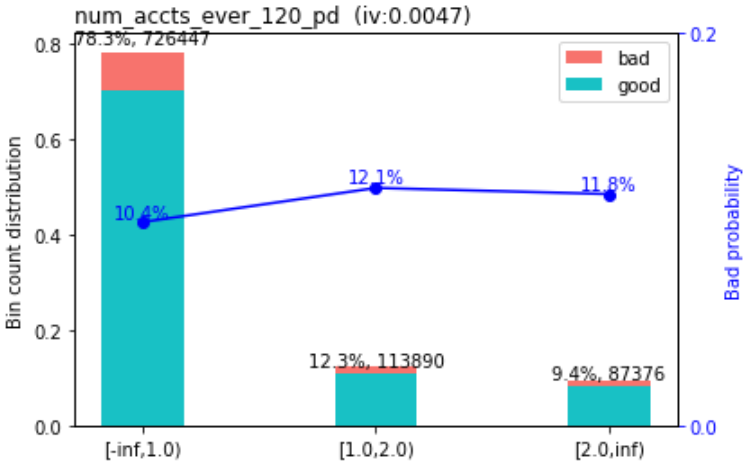



```
>>> Adjust breaks for (11/28) mths_since_recent_revol_delinq?
1: next
2: yes
3: back
Selection: 1
-------- 12/28 num_accts_ever_120_pd --------
>>> dt[num_accts_ever_120_pd].describe():
count    927713.000000
mean          0.422499
std           1.097066
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max          24.000000
Name: num_accts_ever_120_pd, dtype: float64
```

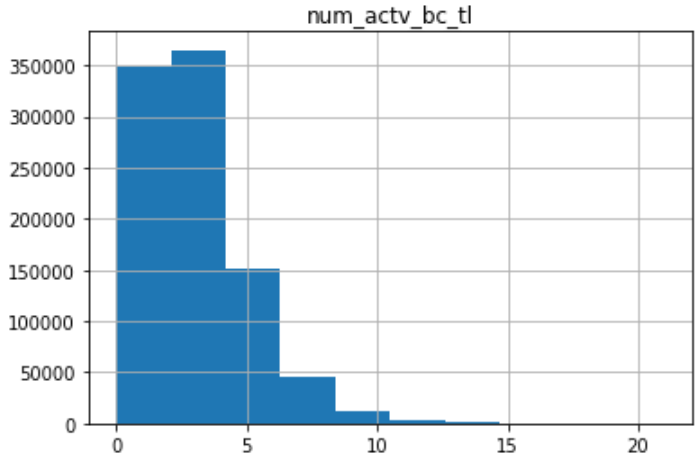




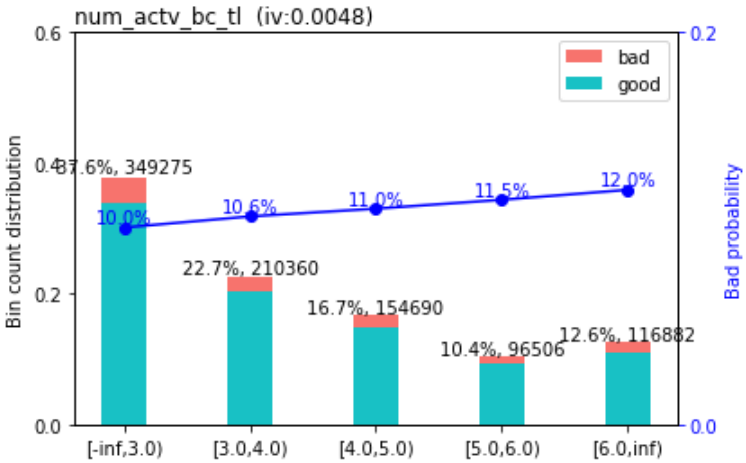

```
>>> Current breaks:
1.0,2.0
```



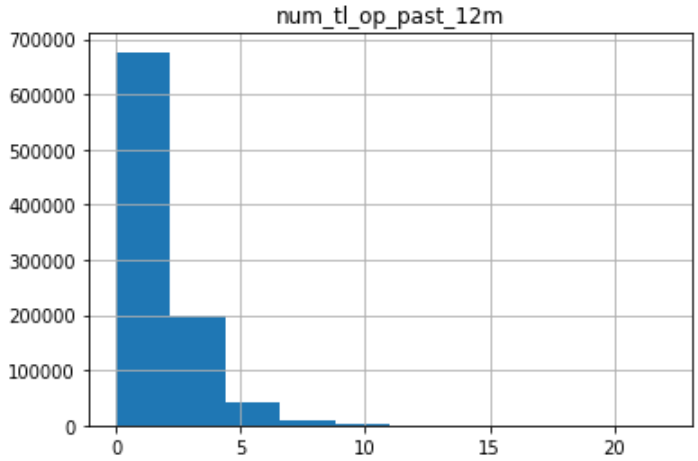

```
>>> Adjust breaks for (12/28) num_accts_ever_120_pd?
1: next
2: yes
3: back
Selection: 1
-------- 13/28 num_actv_bc_tl --------
>>> dt[num_actv_bc_tl].describe():
count    927713.000000
mean          3.350232
std           1.920374
min           0.000000
25%           2.000000
50%           3.000000
75%           4.000000
max          21.000000
Name: num_actv_bc_tl, dtype: float64
```



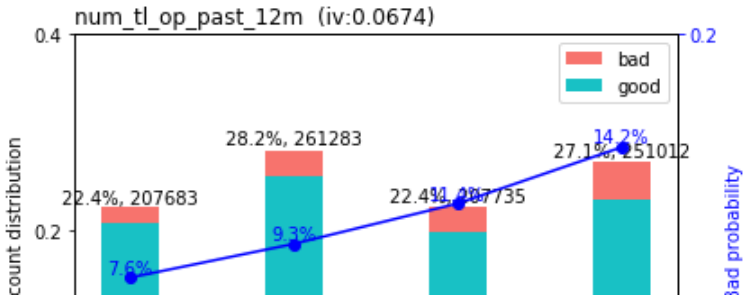
```
>>> Current breaks:
3.0,4.0,5.0,6.0
```

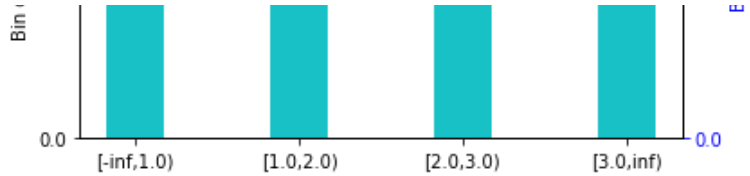



```
>>> Adjust breaks for (13/28) num_actv_bc_tl?
1: next
2: yes
3: back
Selection: 1
-------- 14/28 num_tl_op_past_12m --------
>>> dt[num_tl_op_past_12m].describe():
count    927713.000000
mean          1.778645
std           1.587047
min           0.000000
25%           1.000000
50%           1.000000
75%           3.000000
max          22.000000
Name: num_tl_op_past_12m, dtype: float64
```

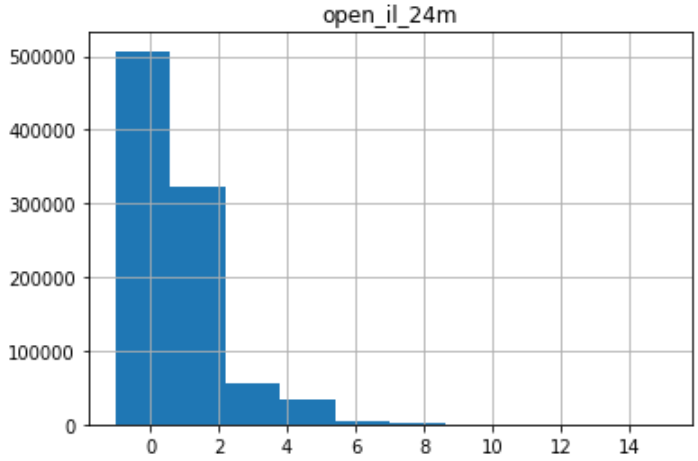



```
>>> Current breaks:
1.0,2.0,3.0
```

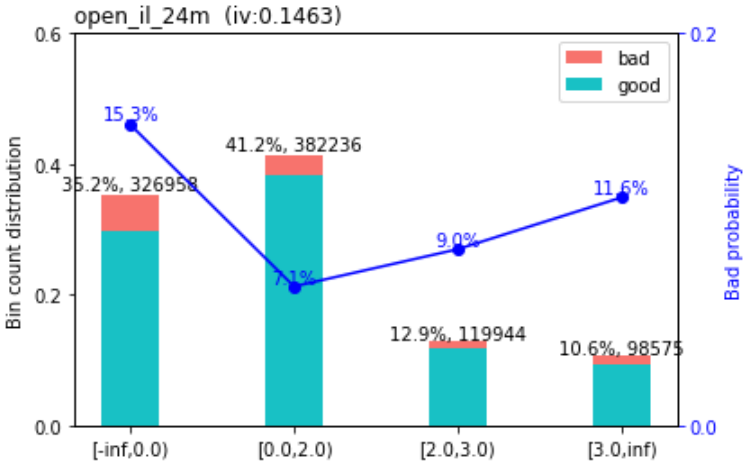





```
>>> Adjust breaks for (14/28) num_tl_op_past_12m?
1: next
2: yes
3: back
Selection: 1
-------- 15/28 open_il_24m --------
>>> dt[open_il_24m].describe():
count    927713.000000
mean          0.522417
std           1.563088
min          -1.000000
25%          -1.000000
50%           0.000000
75%           1.000000
max          15.000000
Name: open_il_24m, dtype: float64
```

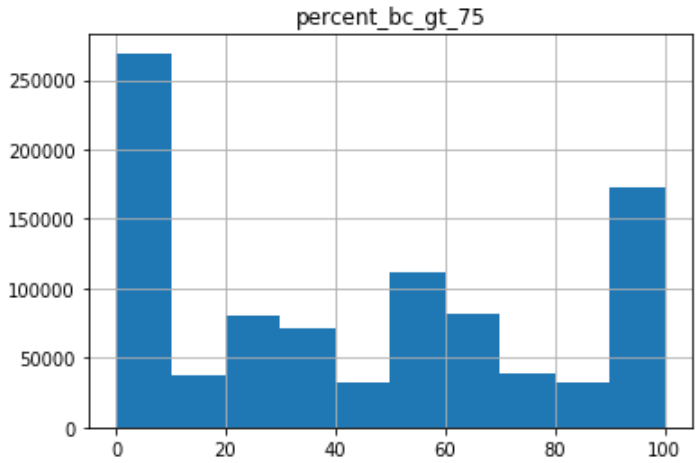



```
>>> Current breaks:
0.0,2.0,3.0
```

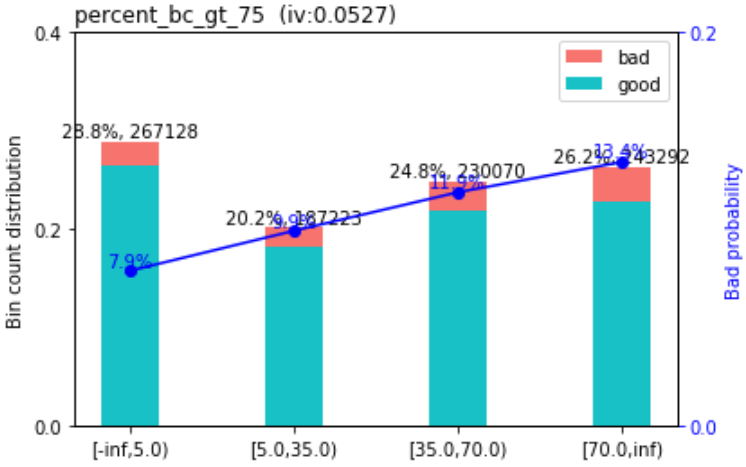



```
>>> Adjust breaks for (15/28) open_il_24m?
1: next
2: yes
3: back
Selection: 1
-------- 16/28 percent_bc_gt_75 --------
>>> dt[percent_bc_gt_75].describe():
```

```
count    927713.000000
mean         42.996655
std          36.835549
min           0.000000
25%           0.000000
50%          40.000000
75%          75.000000
max         100.000000
Name: percent_bc_gt_75, dtype: float64
```

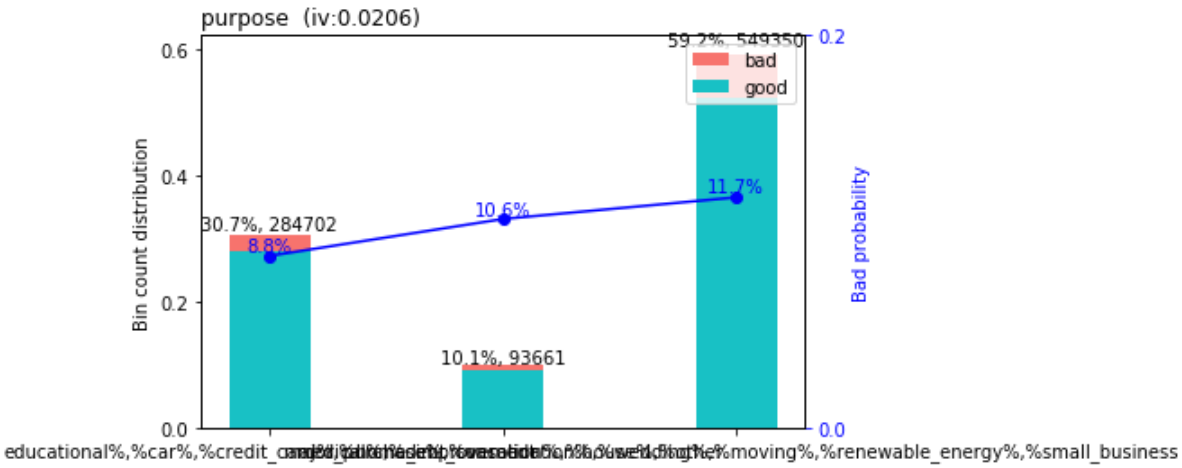



```
>>> Current breaks:
5.0,35.0,70.0
```

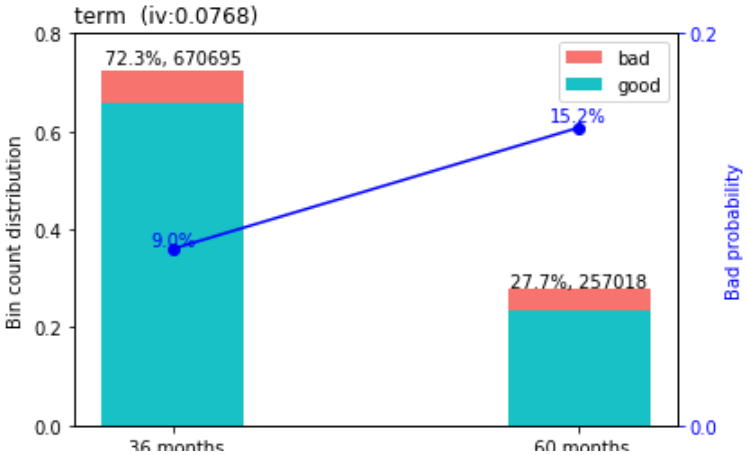



```
>>> Adjust breaks for (16/28) percent_bc_gt_75?
1: next
2: yes
3: back
Selection: 1
-------- 17/28 purpose --------
>>> dt[purpose].describe():
count                 927713
unique                    14
top       debt_consolidation
freq                  520472
Name: purpose, dtype: object

>>> dt[purpose].value_counts():
debt_consolidation    520472
credit_card           216164
other                  59490
home_improvement       58234
major_purchase         21763
medical                11140
```

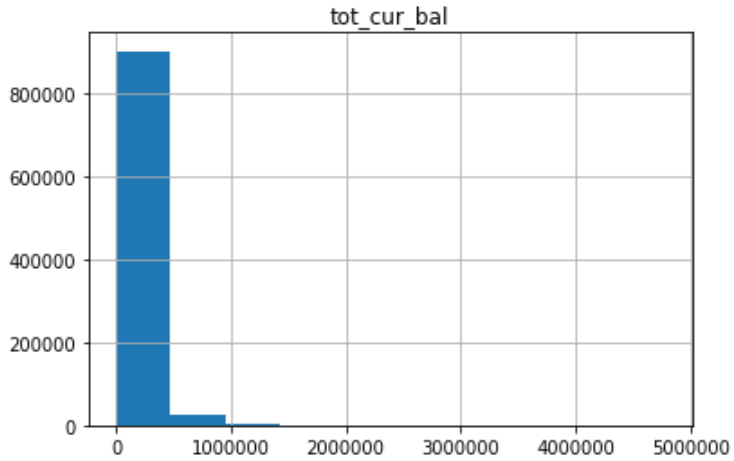
```
car                   10303
small_business        10095
moving                 6596
vacation               6420
house                  5988
renewable_energy        583
wedding                 464
educational               1
Name: purpose, dtype: int64

>>> Current breaks:
'educational%,%car%,%credit_card%,%home_improvement','major_purchase%,%vacation%,%house%,%
```



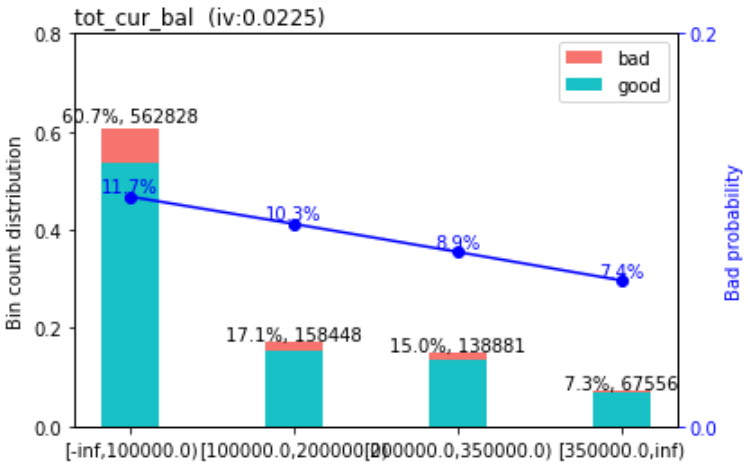

```
>>> Adjust breaks for (17/28) purpose?
1: next
2: yes
3: back
Selection: 1
-------- 18/28 term --------
>>> dt[term].describe():
count          927713
unique              2
top         36 months
freq           670695
Name: term, dtype: object

>>> dt[term].value_counts():
 36 months    670695
 60 months    257018
Name: term, dtype: int64

>>> Current breaks:
' 36 months',' 60 months'
```



```
>>> Adjust breaks for (18/28) term?
1: next
2: yes
3: back
Selection: 1
-------- 19/28 tot_cur_bal --------
>>> dt[tot_cur_bal].describe():
count    9.277130e+05
mean     1.221721e+05
std      1.440997e+05
min      0.000000e+00
25%      2.527600e+04
50%      5.813200e+04
75%      1.828060e+05
max      4.772549e+06
Name: tot_cur_bal, dtype: float64
```

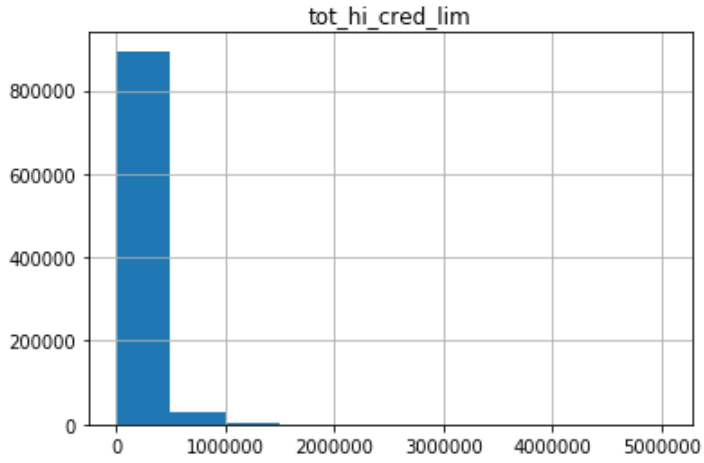



```
>>> Current breaks:
100000.0,200000.0,350000.0
```

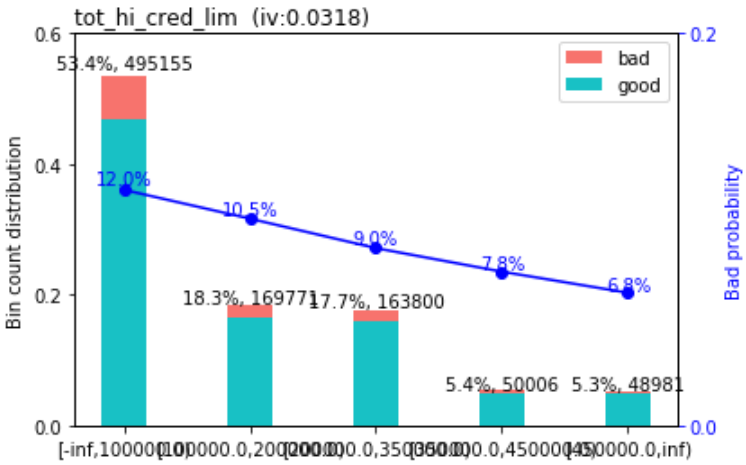



```
>>> Adjust breaks for (19/28) tot_cur_bal?
1: next
2: yes
3: back
Selection: 1
-------- 20/28 tot_hi_cred_lim --------
>>> dt[tot_hi_cred_lim].describe():
count    9.277130e+05
mean     1.533256e+05
std      1.598689e+05
min      1.000000e+02
25%      4.468100e+04
```

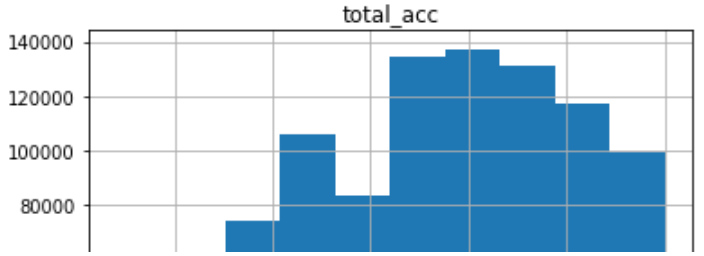
```
50%       8.880100e+04
75%       2.210810e+05
max       5.022617e+06
Name: tot_hi_cred_lim, dtype: float64
```

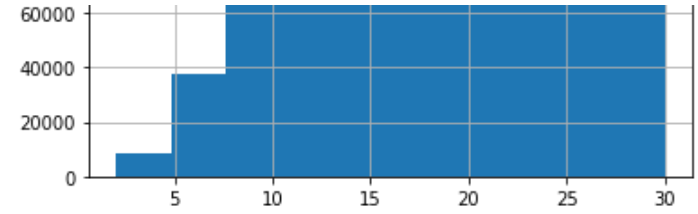



```
>>> Current breaks:
100000.0,200000.0,350000.0,450000.0
```

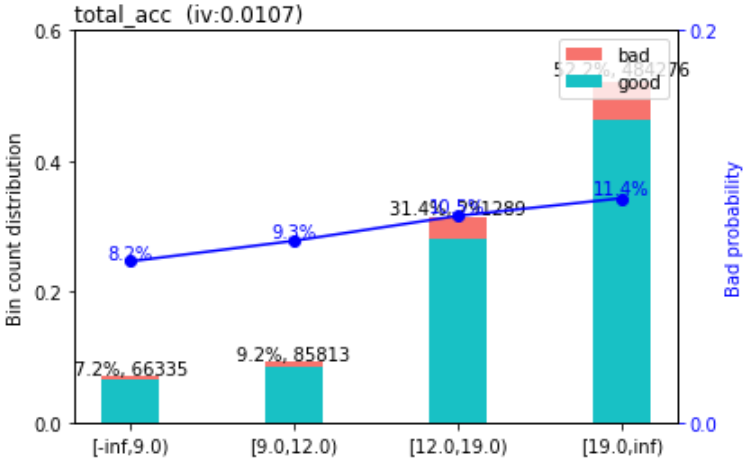



```
>>> Adjust breaks for (20/28) tot_hi_cred_lim?
1: next
2: yes
3: back
Selection: 1
-------- 21/28 total_acc --------
>>> dt[total_acc].describe():
count    927713.000000
mean         18.725004
std           6.638534
min           2.000000
25%          14.000000
50%          19.000000
75%          24.000000
max          30.000000
Name: total_acc, dtype: float64
```

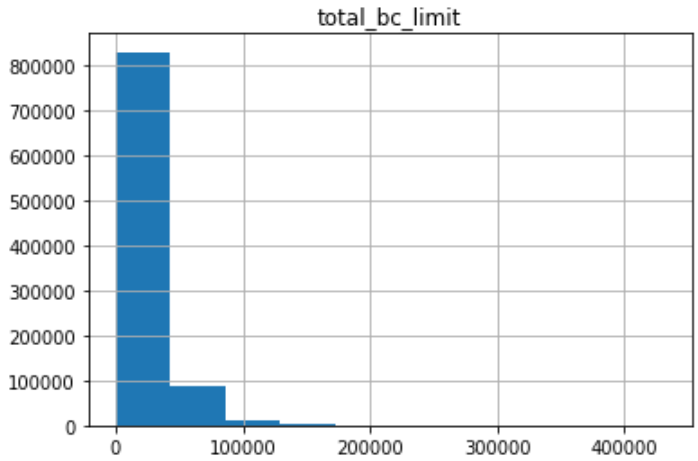




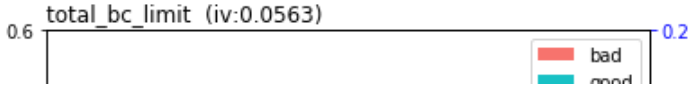

```
>>> Current breaks:
9.0,12.0,19.0
```

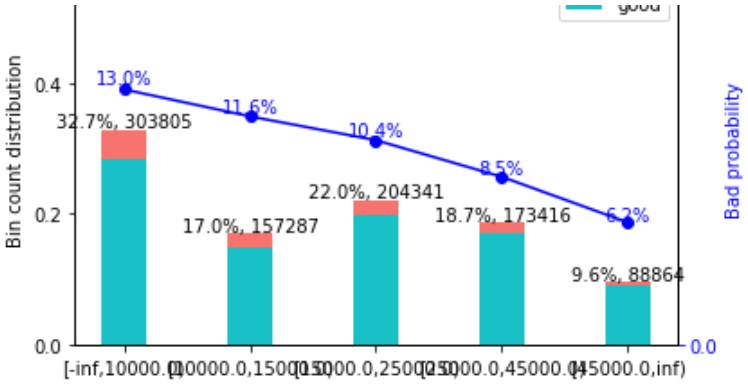



```
>>> Adjust breaks for (21/28) total_acc?
1: next
2: yes
3: back
Selection: 1
-------- 22/28 total_bc_limit --------
>>> dt[total_bc_limit].describe():
count    927713.000000
mean      20623.255690
std       19015.661616
min         100.000000
25%        7900.000000
50%       15000.000000
75%       27100.000000
max      431100.000000
Name: total_bc_limit, dtype: float64
```

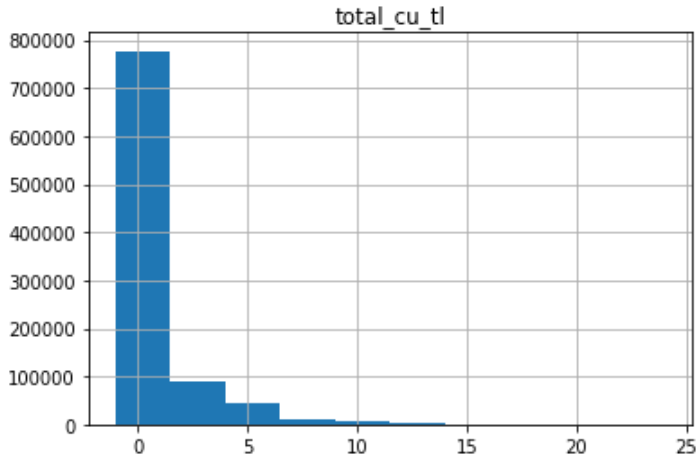



```
>>> Current breaks:
10000.0,15000.0,25000.0,45000.0
```

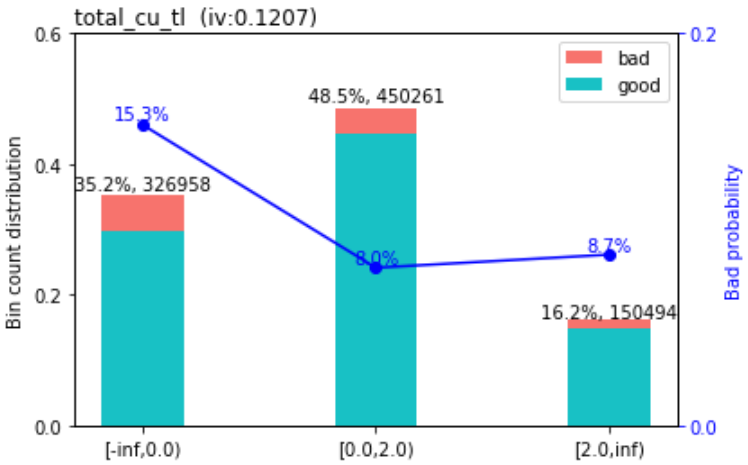





```
>>> Adjust breaks for (22/28) total_bc_limit?
1: next
2: yes
3: back
Selection: 1
-------- 23/28 total_cu_tl --------
>>> dt[total_cu_tl].describe():
count    927713.000000
mean          0.383784
std           1.900223
min          -1.000000
25%          -1.000000
50%           0.000000
75%           1.000000
max          24.000000
Name: total_cu_tl, dtype: float64
```

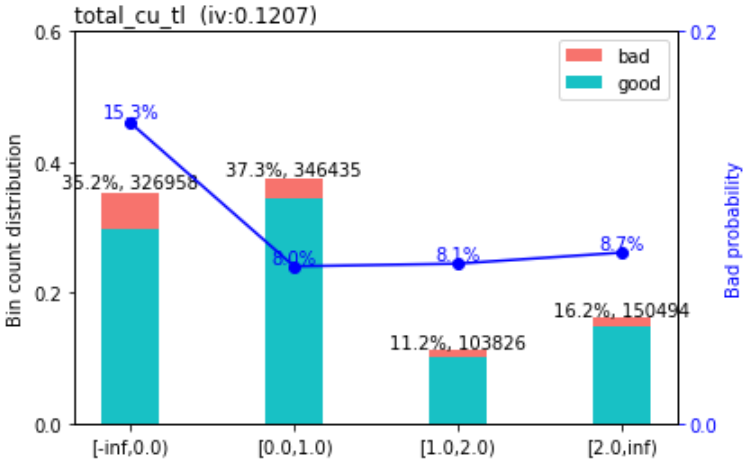



```
>>> Current breaks:
0.0,2.0
```

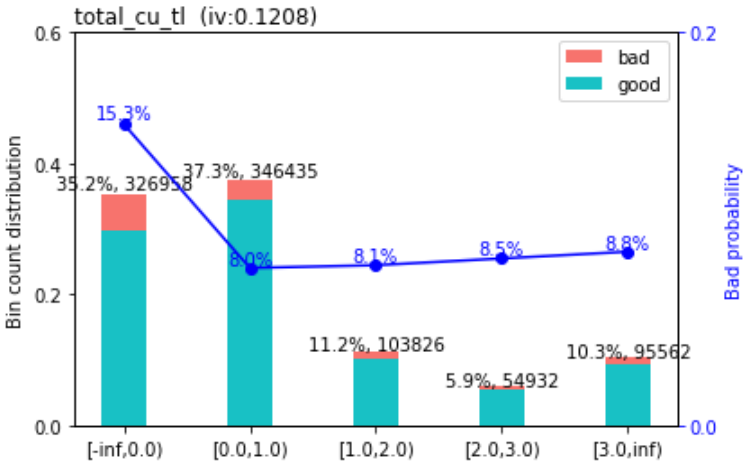



```
>>> Adjust breaks for (23/28) total_cu_tl?
```

```
>>> Adjust breaks for (23/28) total_cu_tl?
1: next
2: yes
3: back
Selection: 2
>>> Enter modified breaks: 0.0,1.0,2.0
[INFO] creating woe binning ...
>>> Current breaks:
0.0, 1.0, 2.0
```

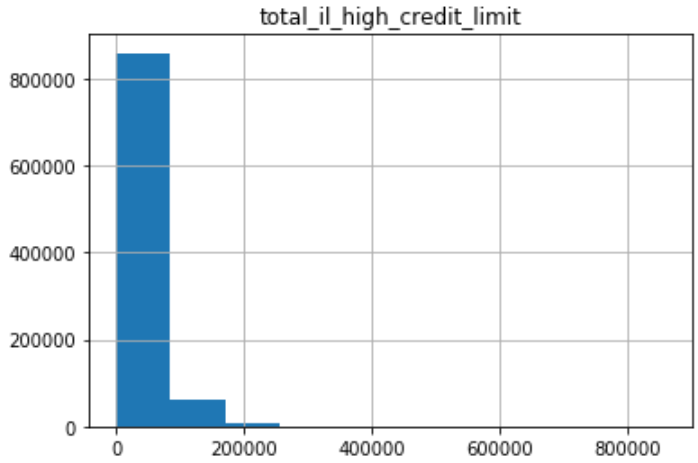



```
>>> Adjust breaks for (23/28) total_cu_tl?
1: next
2: yes
3: back
Selection: 2
>>> Enter modified breaks: 0.0, 1.0, 2.0, 3.0
[INFO] creating woe binning ...
>>> Current breaks:
0.0, 1.0, 3.0, 2.0
```

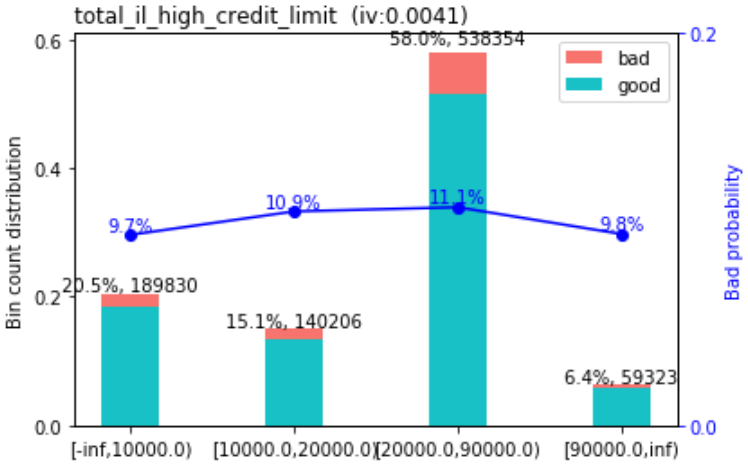



```
>>> Adjust breaks for (23/28) total_cu_tl?
1: next
2: yes
3: back
Selection: 1
-------- 24/28 total_il_high_credit_limit --------
>>> dt[total_il_high_credit_limit].describe():
count    927713.000000
mean      36011.916856
std       34501.287665
min           0.000000
25%       13068.000000
50%       28703.000000
75%       49499.000000
```

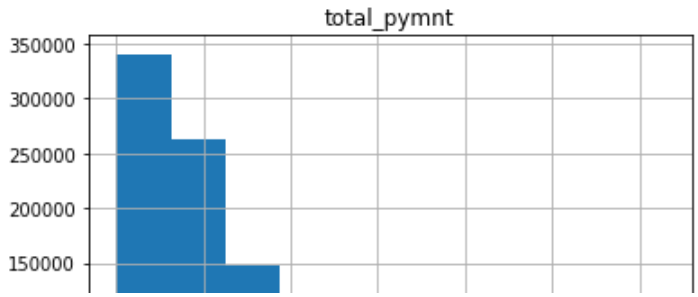
```
max       856554.000000
Name: total_il_high_credit_limit, dtype: float64
```

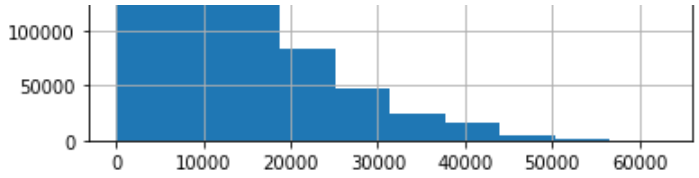



```
>>> Current breaks:
10000.0,20000.0,90000.0
```

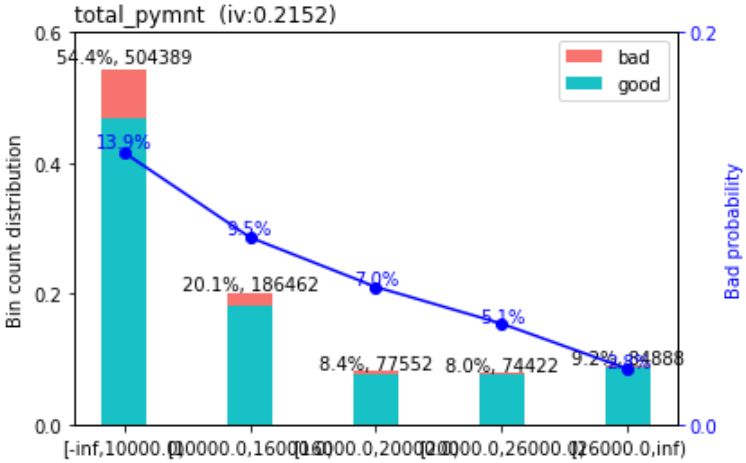



```
>>> Adjust breaks for (24/28) total_il_high_credit_limit?
1: next
2: yes
3: back
Selection: 1
-------- 25/28 total_pymnt --------
>>> dt[total_pymnt].describe():
count    927713.000000
mean      11563.375957
std        9515.251949
min           0.000000
25%        4356.100000
50%        8943.120000
75%       16214.480000
max       62837.639685
Name: total_pymnt, dtype: float64
```

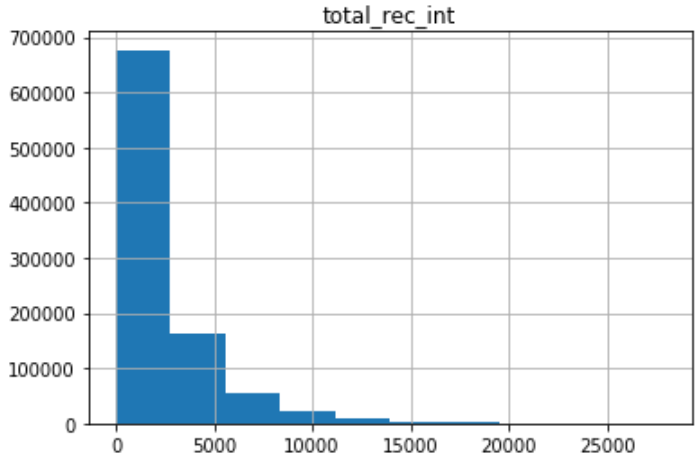




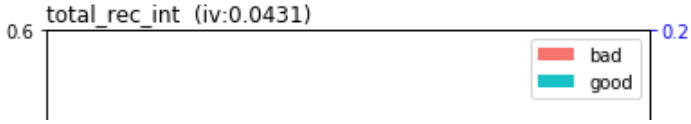

```
>>> Current breaks:
10000.0,16000.0,20000.0,26000.0
```

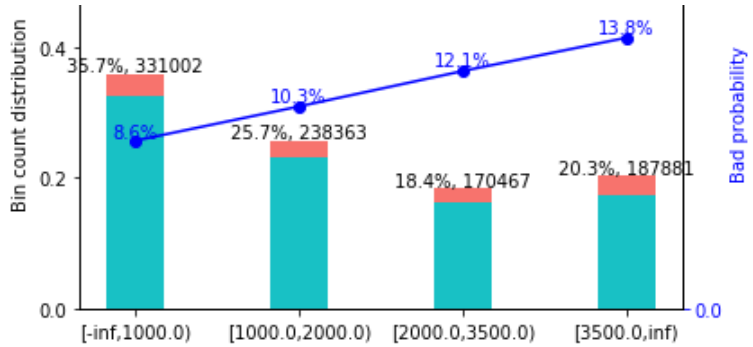



```
>>> Adjust breaks for (25/28) total_pymnt?
1: next
2: yes
3: back
Selection: 1
-------- 26/28 total_rec_int --------
>>> dt[total_rec_int].describe():
count    927713.000000
mean       2342.903322
std        2580.948897
min           0.000000
25%         712.000000
50%        1478.550000
75%        2979.310000
max       27922.660000
Name: total_rec_int, dtype: float64
```

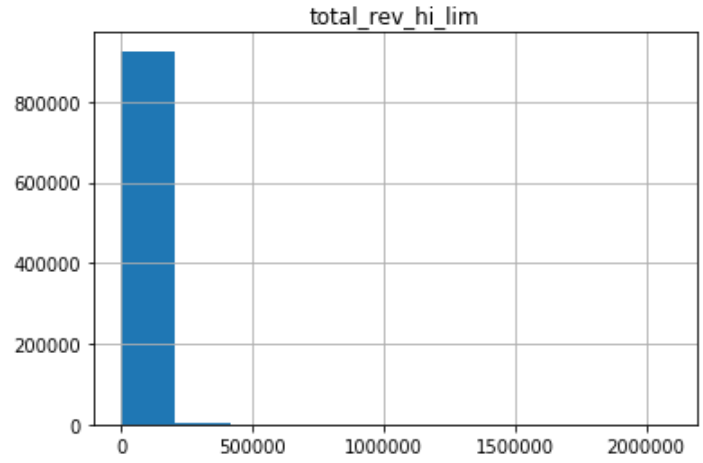



```
>>> Current breaks:
1000.0,2000.0,3500.0
```

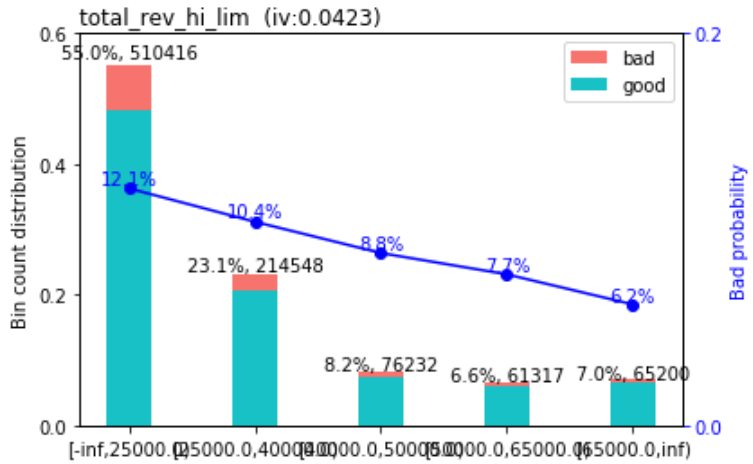





```
>>> Adjust breaks for (26/28) total_rec_int?
1: next
2: yes
3: back
Selection: 1
-------- 27/28 total_rev_hi_lim --------
>>> dt[total_rev_hi_lim].describe():
count    9.277130e+05
mean     2.950474e+04
std      2.789601e+04
min      1.000000e+02
25%      1.350000e+04
50%      2.270000e+04
75%      3.710000e+04
max      2.087500e+06
Name: total_rev_hi_lim, dtype: float64
```

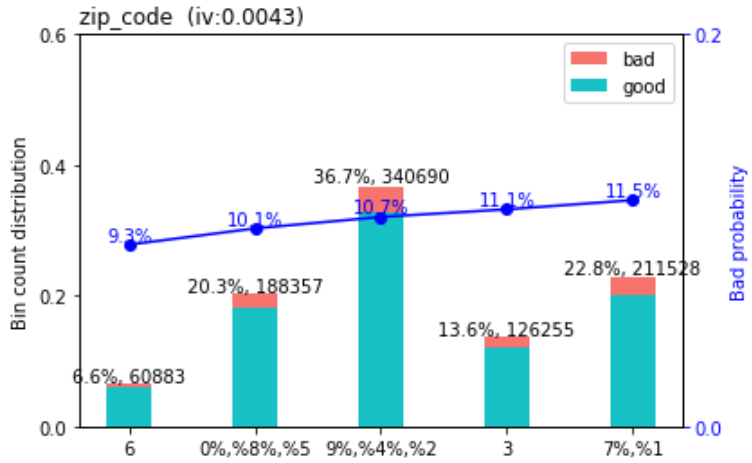



```
>>> Current breaks:
25000.0,40000.0,50000.0,65000.0
```




```
>>> Adjust breaks for (27/28) total_rev_hi_lim?
1: next
```

```
2: yes
3: back
Selection: 1
-------- 28/28 zip_code --------
>>> dt[zip_code].describe():
count     927713
unique        10
top            9
freq      176503
Name: zip_code, dtype: object

>>> dt[zip_code].value_counts():
9    176503
3    126255
1    112827
7     98701
2     88237
0     84371
4     75950
8     70368
6     60883
5     33618
Name: zip_code, dtype: int64

>>> Current breaks:
'6','0%,%8%,%5','9%,%4%,%2','3','7%,%1'
```




```
>>> Adjust breaks for (28/28) zip_code?
1: next
2: yes
3: back
Selection: 1
```

```
bins_adj = sc.woebin(train, y="loan_status", breaks_list=breaks_adj) # Apply new cuts
train_woe = sc.woebin_ply(train, bins_adj) # Calculate WoE dataset (train)
test_woe = sc.woebin_ply(test, bins_adj) # Calculate WoE dataset (test)
```

```
[INFO] creating woe binning ...
Binning on 927713 rows and 55 columns in 00:02:54
[INFO] converting into woe values ...
Woe transformating on 927713 rows and 54 columns in 00:02:08
[INFO] converting into woe values ...
Woe transformating on 477914 rows and 54 columns in 00:01:11
```

```
train_woe.head()
```

	loan_status	emp_length_woe	annual_inc_woe	all_util_woe	inq_fi_woe	debt_settlement_f
0	0	-0.033730	0.121353	-0.735615	-0.060631	
2	0	-0.033730	-0.005652	-0.074370	-0.060631	
5	0	0.001594	0.121353	-0.074370	-0.373631	
7	0	-0.033730	-0.005652	-0.430728	-0.373631	
8	0	0.001594	-0.005652	-0.430728	-0.060631	

```
sc.iv(train_woe, 'loan_status')
```

	variable	info_value
16	last_fico_range_low_woe	4.071438
6	out_prncp_woe	0.811687
14	last_credit_pull_d_woe	0.406819
19	total_pymnt_woe	0.215189
27	all_util_woe	0.156401
42	open_rv_24m_woe	0.151765
38	open_il_24m_woe	0.146334
11	inq_fi_woe	0.130523
22	max_bal_bc_woe	0.127302
18	open_act_il_woe	0.124997
32	total_cu_tl_woe	0.120770
40	term_woe	0.076819
26	num_tl_op_past_12m_woe	0.067446
48	total_bc_limit_woe	0.056288
53	mths_since_recent_inq_woe	0.053287
36	percent_bc_gt_75_woe	0.052656
29	inq_last_6mths_woe	0.052372
20	dti_woe	0.047098
30	total_rec_int_woe	0.043100
25	total_rev_hi_lim_woe	0.042325
0	mo_sin_rcnt_rev_tl_op_woe	0.037520
23	tot_hi_cred_lim_woe	0.031798
34	loan_amnt_woe	0.024936
45	tot_cur_bal_woe	0.022470
7	annual_inc_woe	0.021164
49	purpose_woe	0.020586
50	application_type_woe	0.017883
10	home_ownership_woe	0.017815
2	mort_acc_woe	0.012331
9	mo_sin_old_rev_tl_op_woe	0.011917
12	num_op_rev_tl_woe	0.011130
1	earliest_cr_line_woe	0.010827
28	total_acc_woe	0.010722
51	mths_since_last_record_woe	0.007541

13	mths_since_last_major_derog_woe	0.005854
3	delinq_2yrs_woe	0.005035
15	num_actv_bc_tl_woe	0.004756
37	num_accts_ever_120_pd_woe	0.004669
46	zip_code_woe	0.004301
43	mths_since_recent_revol_delinq_woe	0.004153
4	total_il_high_credit_limit_woe	0.004102
8	pub_rec_bankruptcies_woe	0.002469
33	emp_length_woe	0.001381
47	num_tl_90g_dpd_24m_woe	0.000000
52	tot_coll_amt_woe	0.000000
21	acc_now_delinq_woe	0.000000
31	debt_settlement_flag_woe	0.000000
35	num_tl_120dpd_2m_woe	0.000000
24	hardship_flag_woe	0.000000
44	collections_12_mths_ex_med_woe	0.000000
17	tax_liens_woe	0.000000
41	chargeoff_within_12_mths_woe	0.000000
39	delinq_amnt_woe	0.000000
5	hardship_amount_woe	0.000000

```
IV = sc.iv(train_woe, 'loan_status')
```

```
list(IV.iloc[0:11,0].str[:-4])
```

```
feature = list(IV.iloc[0:14,0].str[:-4])
```

Variable Selection

Although "out_prncp" and "total_pymnt" has relative high WOE value, but they are the information we would never have when some applicants come and apply for loans.

So My decison is drop "out_prncp" and "total_pymnt".

At the same time, the variable "last_fico_range_low" has exordinary value for WOE exceed 1.0. This is because fic one of the largest credit bureal in the United States, it has much more large sample than we do, and the fico scor already a result of modelling result. I am going to include it in my model just in case we need its strong predictive power, and will compare the prediction power between has FICO and without FICO

```
train_woe.columns
```

```
start = 0
for i in train_woe.columns:
  print(i + '   ' + str(start))
  start = start + 1
```

```
loan_status   0
emp_length_woe   1
annual_inc_woe   2
all_util_woe   3
inq_fi_woe   4
debt_settlement_flag_woe   5
delinq_amnt_woe   6
hardship_amount_woe   7
chargeoff_within_12_mths_woe   8
num_actv_bc_tl_woe   9
hardship_flag_woe   10
last_credit_pull_d_woe   11
mths_since_last_record_woe   12
open_rv_24m_woe   13
num_tl_op_past_12m_woe   14
open_act_il_woe   15
pub_rec_bankruptcies_woe   16
application_type_woe   17
total_cu_tl_woe   18
last_fico_range_low_woe   19
num_tl_90g_dpd_24m_woe   20
mort_acc_woe   21
total_il_high_credit_limit_woe   22
mo_sin_rcnt_rev_tl_op_woe   23
num_accts_ever_120_pd_woe   24
tot_hi_cred_lim_woe   25
mo_sin_old_rev_tl_op_woe   26
total_pymnt_woe   27
acc_now_delinq_woe   28
home_ownership_woe   29
total_bc_limit_woe   30
purpose_woe   31
dti_woe   32
collections_12_mths_ex_med_woe   33
tax_liens_woe   34
tot_cur_bal_woe   35
mths_since_recent_revol_delinq_woe   36
percent_bc_gt_75_woe   37
open_il_24m_woe   38
total_acc_woe   39
out_prncp_woe   40
loan_amnt_woe   41
mths_since_last_major_derog_woe   42
inq_last_6mths_woe   43
total_rev_hi_lim_woe   44
earliest_cr_line_woe   45
num_tl_120dpd_2m_woe   46
zip_code_woe   47
tot_coll_amt_woe   48
term_woe   49
total_rec_int_woe   50
num_op_rev_tl_woe   51
max_bal_bc_woe   52
delinq_2yrs_woe   53
mths_since_recent_inq_woe   54
```

Before we select those features, we check those selected features correlation again.

```
df_part6 = Loan_data.loc[:, feature]
corelation_part6 = df_part6.corr(method = 'pearson', min_periods = 1)
```

```
get_top_abs_correlations(corelation_part6, 60)
```

From the above result, we can see we can futher eliminate "all_util"

```
accepted_feature = ([0, 19, 11, 13, 38, 4, 52, 18, 15])
```

```
# Now we create range of accepted variables
train_woe = train_woe.iloc[:, accepted_feature]
test_woe = test_woe.iloc[:, accepted_feature]
train_woe.head()
```

	loan_status	last_fico_range_low_woe	last_credit_pull_d_woe	open_rv_24m_woe	open_il_2
0	0	2.897998	-0.345257	-0.038696	-0
2	0	-3.008662	-0.345257	-0.412297	0
5	0	-3.008662	1.249359	-0.664759	-0
7	0	-0.936299	-0.345257	-0.664759	0
8	0	-3.008662	-0.345257	-0.038696	-0

```
#%%script false
train_woe.to_csv("train_woe.csv", index = False)
test_woe.to_csv("test_woe.csv", index = False)
!ls
```

```
#%%script false
files.download("train_woe.csv")
```

```
#%%script false
files.download("test_woe.csv")
```

```
train_woe['loan_status'].value_counts()
```

```
train_woe['loan_status'] = train_woe['loan_status'].astype('int')
test_woe['loan_status'] = test_woe['loan_status'].astype('int')
```

Perform the Logistic Regression

```
from sklearn.linear_model import LogisticRegression

LC_logreg = LogisticRegression(penalty='l1', # Type of penalization l1 = lasso, l2 = ridge
                               tol=0.0001, # Tolerance for parameters
                               C=100,
                               fit_intercept=True,
                               class_weight='balanced',
                               random state=251121253,
```

```
                                    max_iter=100,
                                    verbose=1,
                                    solver = 'saga',
                                    warm_start=False
                                   )
```

```
feature_train = train_woe.iloc[:,1:]
objective_train = train_woe['loan_status']
feature_test = test_woe.iloc[:,1:]
objective_test = test_woe['loan_status']
```

```
LC_logreg.fit(X = feature_train, y = objective_train)
```

```
coef_df = pd.concat([pd.DataFrame({'column': train_woe.columns[1:]}),
                     pd.DataFrame(np.transpose(LC_logreg.coef_))],
                     axis = 1
                    )
```

```
coef_df
```

```
LC_logreg.intercept_
```

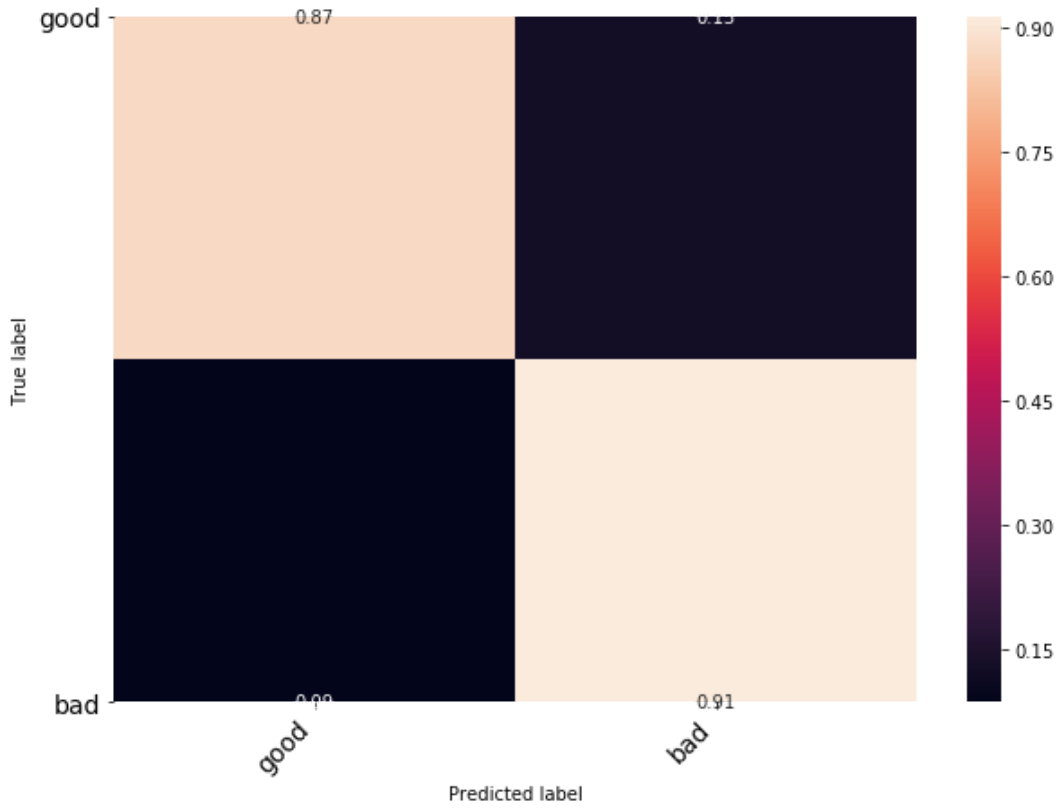
```
pred_class_test = LC_logreg.predict(feature_test)
probs_test = LC_logreg.predict_proba(feature_test)
print(probs_test[0:5], pred_class_test[0:5])
```

Evaluate by Confusion Matrix

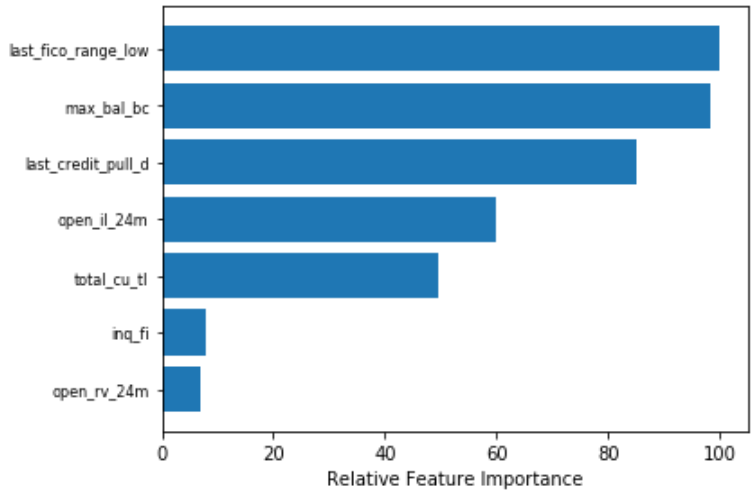
```
from sklearn.metrics import confusion_matrix


confusion_matrix_Loan = \
confusion_matrix(y_true = objective_test, y_pred = pred_class_test)


confusion_matrix_Loan
```

```
array([[373283,  53398],
       [  4408,  46825]])
```

```
confusion_matrix_Loan = confusion_matrix_Loan.astype('float') \
           / confusion_matrix_Loan.sum(axis=1)[:, np.newaxis]


Loan_cm = pd.DataFrame(
        confusion_matrix_Loan, index=['good', 'bad'], columns=['good', 'bad'],
)


# Parameters of the image
figsize = (10,7)
fontsize=14

# Create image
fig = plt.figure(figsize=figsize)
heatmap = sns.heatmap(Loan_cm, annot=True, fmt='.2f')

# Make it nicer
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0,
                             ha='right', fontsize=fontsize)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45,
                             ha='right', fontsize=fontsize)

# Add labels
plt.ylabel('True label')
plt.xlabel('Predicted label')

# Plot!
plt.show()
```




```
LC_sc = sc.scorecard(bins_adj, LC_logreg,
             train_woe.columns[1:], # The column names in the trained LR
             points0=500, # Base points
             odds0=0.01, # Base odds
             pdo=50) # PDO


feature_importance = abs( LC_logreg.coef_[0])
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5

featfig = plt.figure()
featax = featfig.add_subplot(1, 1, 1)
featax.barh(pos, feature_importance[sorted_idx], align='center')
featax.set_yticks(pos)
featax.set_yticklabels(np.array(feature_test.columns.str[:-4])[sorted_idx], fontsize=8)
featax.set_xlabel('Relative Feature Importance')

plt.tight_layout()
plt.show()
```



Scorecard Construction

LC_sc

{'basepoints':   variable bin points
0 basepoints NaN   166.0, 'inq_fi':   variable   bin points
12   inq_fi [-inf,0.0)   3.0
13   inq_fi   [0.0,2.0)   -2.0
14   inq_fi   [2.0,inf)   -0.0, 'last_credit_pull_d':   variable
25 last_credit_pull_d   [-inf,2017.0)   -71.0
26 last_credit_pull_d [2017.0,2018.0)   -76.0
27 last_credit_pull_d   [2018.0,inf)   21.0, 'last_fico_range_low':   va
51 last_fico_range_low   [-inf,580.0) -206.0
52 last_fico_range_low [580.0,640.0)   -92.0
53 last_fico_range_low [640.0,680.0)   67.0
54 last_fico_range_low   [680.0,inf)   214.0, 'max_bal_bc':   variable   b
163 max_bal_bc   [-inf,0.0)   29.0
164 max_bal_bc [0.0,7000.0)   -16.0
165 max_bal_bc [7000.0,inf)   -35.0, 'open_act_il':   variable   bin points
39 open_act_il [-inf,0.0)   -2.0
40 open_act_il   [0.0,2.0)   2.0
41 open_act_il   [2.0,inf)   1.0, 'open_il_24m':   variable   bin points
115 open_il_24m [-inf,0.0)   -17.0
116 open_il_24m   [0.0,2.0)   19.0
117 open_il_24m   [2.0,3.0)   8.0
118 open_il_24m   [3.0,inf)   -4.0, 'open_rv_24m':   variable   bin points
31 open_rv_24m [-inf,0.0)   -2.0
32 open_rv_24m   [0.0,1.0)   3.0
33 open_rv_24m   [1.0,3.0)   2.0
34 open_rv_24m   [3.0,inf)   0.0, 'total_cu_tl':   variable   bin points
46 total_cu_tl [-inf,0.0)   -13.0
47 total_cu_tl   [0.0,1.0)   10.0
48 total_cu_tl   [1.0,2.0)   10.0
49 total_cu_tl   [2.0,3.0)   8.0
50 total_cu_tl   [3.0,inf)   7.0}

visualize score card

{'basepoints':   variable   bin   points

```
        0          basepoints   NaN    166.0,
 'inq_fi':    variable         bin       points
         17   inq_fi       [-inf,0.0)     2.0
         18   inq_fi        [0.0,2.0)     -2.0
         19    inq_fi       [2.0,inf)    -0.0,
'last_credit_pull_d':                variable            bin        points
                          114  last_credit_pull_d  [-inf,2018.0)   -74.0
                          115  last_credit_pull_d   [2018.0,inf)    21.0,

'last_fico_range_low':               variable            bin      points
                         33  last_fico_range_low   [-inf,580.0)  -206.0
                         34  last_fico_range_low  [580.0,640.0)   -92.0
                         35  last_fico_range_low  [640.0,680.0)    67.0
                         36  last_fico_range_low    [680.0,inf)   214.0,

'max_bal_bc':               variable           bin          points
                        146  max_bal_bc    [-inf,0.0)    29.0
                        147  max_bal_bc  [0.0,7000.0)   -16.0
                        148  max_bal_bc  [7000.0,inf)   -35.0,
'open_act_il':             variable         bin              points
                         69  open_act_il  [-inf,0.0)    -3.0
                         70  open_act_il   [0.0,2.0)     3.0
                         71  open_act_il   [2.0,inf)     1.0,
'open_il_24m':              variable         bin       points
                         60  open_il_24m  [-inf,0.0)   -17.0
                         61  open_il_24m   [0.0,2.0)    19.0
                         62  open_il_24m   [2.0,3.0)     8.0
                         63  open_il_24m   [3.0,inf)    -4.0,

'open_rv_24m':               variable         bin        points
                        4    open_rv_24m    [-inf,0.0)      -2.0
                        5   open_rv_24m      [0.0,1.0)      3.0
                        6   open_rv_24m       [1.0,3.0)     2.0
                        7   open_rv_24m       [3.0,inf)      0.0,

'total_cu_tl':                 variable        bin        points
                    20  total_cu_tl        [-inf,0.0)       -12.0
                    21  total_cu_tl         [0.0,2.0)        9.0
                    22  total_cu_tl          [2.0,3.0)       8.0
                    23  total_cu_tl         [3.0,inf)        6.0}
```

```
train_score = sc.scorecard_ply(train, LC_sc,
                               print_step=0)
test_score = sc.scorecard_ply(test, LC_sc,
                              print_step=0)
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/indexing.py:376: 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-docs/stable/user_gui
  self.obj[key] = _infer_fill_value(value)
/usr/local/lib/python3.6/dist-packages/pandas/core/indexing.py:576: 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-docs/stable/user_gui
  self.obj[item_labels[indexer[info_axis]]] = value
```

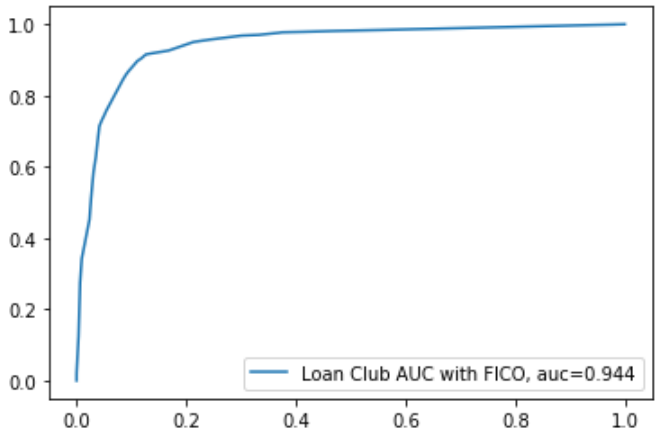
```
train_score.describe()
```

	score
count	927713.000000
mean	293.433869
std	153.104696
min	-149.000000
25%	251.000000
50%	392.000000
75%	399.000000
max	419.000000

```
from sklearn.metrics import roc_auc_score,roc_curve
```

Evaluate By ROC Curve

```
# Calculate the ROC curve points
fpr, tpr, thresholds = roc_curve(test['loan_status'].astype('float'), probs_test[:,1])

# Save the AUC in a variable to display it. Round it first
auc = np.round(roc_auc_score(y_true = test['loan_status'].astype('float'),
                             y_score = probs_test[:,1]),
               decimals = 3)

# Create and show the plot
plt.plot(fpr,tpr,label="Loan Club AUC with FICO, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



Comparison with no FICO score added

Because it is the most important feature of FICO is also a credit score, we will see what would happen if no FICO added

```
feature_train_nofico = train_woe.iloc[:,1:]
feature_train_nofico = feature_train_nofico.drop(['last_fico_range_low_woe'], axis = 1)
objective_train_nofico = train_woe['loan_status']
feature_test_nofico = test_woe.iloc[:,1:]
feature_test_nofico = feature_test_nofico.drop(['last_fico_range_low_woe'], axis = 1)
objective_test_nofico = test_woe['loan_status']
```

```
LC_logreg_nofico = LogisticRegression(penalty='l1', # Type of penalization l1 = lasso, l2 = rid
                                      tol=0.0001, # Tolerance for parameters
                                      C=100, # Penalty constant, see below
                                      fit_intercept=True, # Use constant?
                                      class_weight='balanced', # Weights, see below
                                      random_state=251121253, # Random seed
                                      max_iter=100, # Maximum iterations
                                      verbose=1, # Show process. 1 is yes.
                                      solver = 'saga',
                                      warm_start=False # Train anew or start from previous weigh
                                     )
```

```
LC_logreg_nofico.fit(X = feature_train_nofico, y = objective_train_nofico)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
max_iter reached after 37 seconds
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/sag.py:337: ConvergenceWarning
  "the coef_ did not converge", ConvergenceWarning)
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   37.5s finished
LogisticRegression(C=100, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l1',
                   random_state=251121253, solver='saga', tol=0.0001, verbose=1,
                   warm_start=False)
```

```
LC_logreg_nofico.intercept
```

```
LC_logreg_nofico.intercept_
```

```
array([0.24818346])
```

```
pred_class_test2 = LC_logreg_nofico.predict(feature_test_nofico)
probs_test2 = LC_logreg_nofico.predict_proba(feature_test_nofico)
print(probs_test[0:5], pred_class_test[0:5])
```

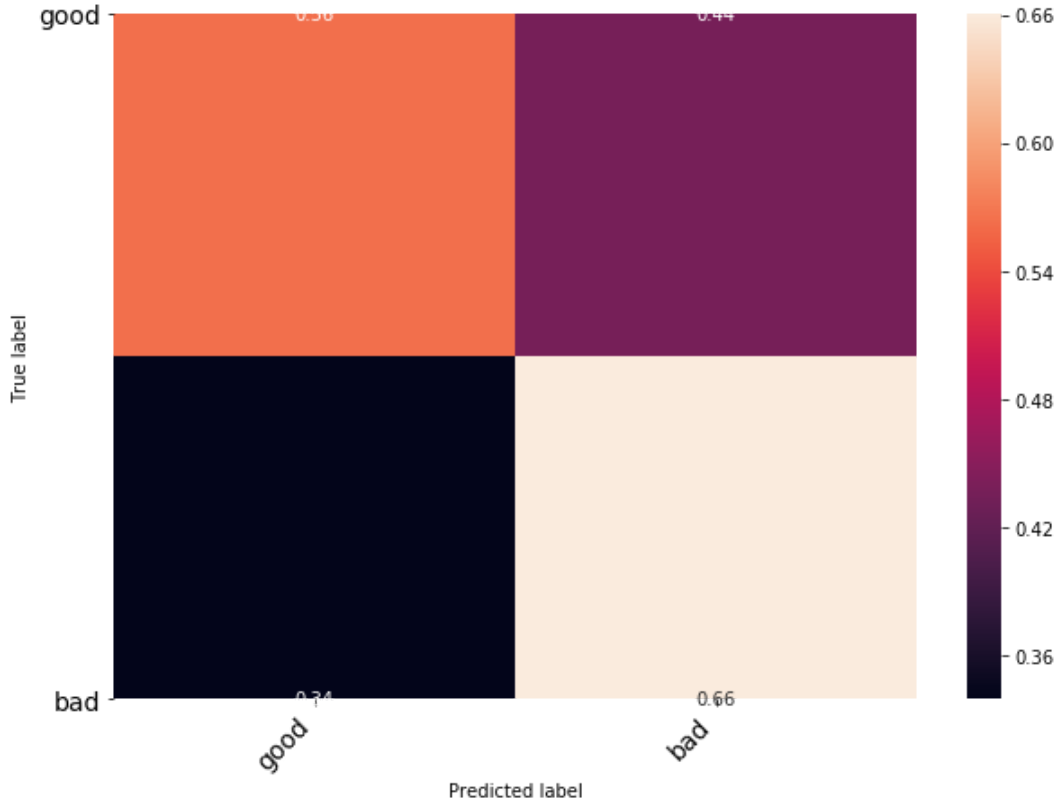
```
[[0.80098414 0.19901586]
 [0.72203401 0.27796599]
 [0.95245494 0.04754506]
 [0.07752698 0.92247302]
 [0.95951199 0.04048801]] [0 0 0 1 0]
```

```
confusion_matrix_Loan_nofico = \
confusion_matrix(y_true = objective_test_nofico, y_pred = pred_class_test2)
```

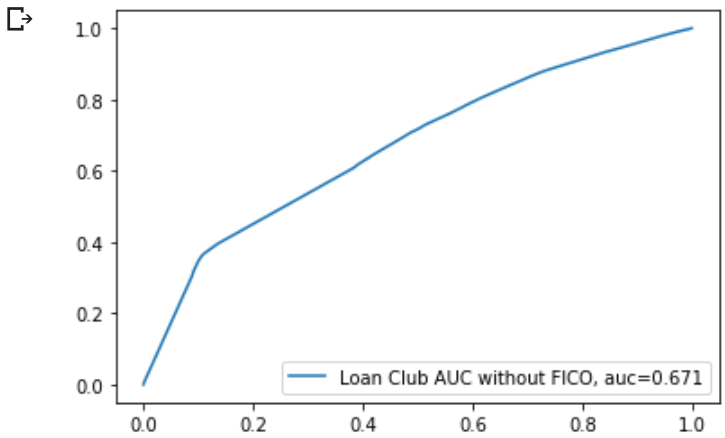
```
confusion_matrix_Loan_nofico = confusion_matrix_Loan_nofico.astype('float') \
            / confusion_matrix_Loan_nofico.sum(axis=1)[:, np.newaxis]
```

```
Loan_cm_nofico = pd.DataFrame(
        confusion_matrix_Loan_nofico, index=['good', 'bad'], columns=['good', 'bad'],
)
```

```
# Parameters of the image
figsize = (10,7)
fontsize=14

# Create image
fig = plt.figure(figsize=figsize)
heatmap2 = sns.heatmap(Loan_cm_nofico, annot=True, fmt='.2f')

# Make it nicer
heatmap2.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0,
                              ha='right', fontsize=fontsize)
heatmap2.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45,
                              ha='right', fontsize=fontsize)

# Add labels
plt.ylabel('True label')
plt.xlabel('Predicted label')

# Plot!
plt.show()
```

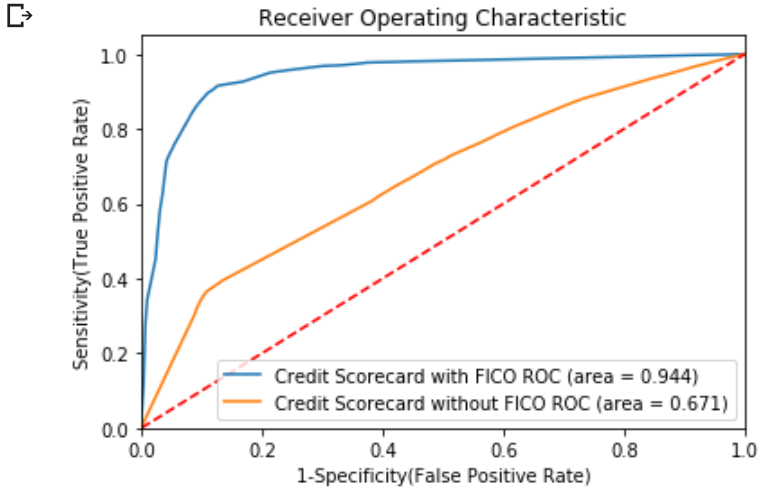



```
# Calculate the ROC curve points
fpr, tpr, thresholds = roc_curve(test['loan_status'].astype('float'), probs_test2[:,1])

# Save the AUC in a variable to display it. Round it first
auc = np.round(roc_auc_score(y_true = test['loan_status'].astype('float'),
                             y_score = probs_test2[:,1]),
               decimals = 3)

# Create and show the plot
plt.plot(fpr,tpr,label="Loan Club AUC without FICO, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



Final Comparison betweem scorecard with FICO included and excluded

```
# Set models and probabilities. This structure is called a dictionary.
models = [
{
    'label': 'Credit Scorecard with FICO',
    'probs': probs_test[:,1]
},
{
    'label': 'Credit Scorecard without FICO',
    'probs': probs_test2[:,1]
},
]

# Loop that creates the plot. I will pass each ROC curve one by one.
for m in models:
  auc = roc_auc_score(y_true = test['loan_status'].astype('float'),
                              y_score = m['probs'])
  fpr, tpr, thresholds = roc_curve(test['loan_status'].astype('float'),
                                           m['probs'])
  plt.plot(fpr, tpr, label='%s ROC (area = %0.3f)' % (m['label'], auc))



# Settings
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('1-Specificity(False Positive Rate)')
plt.ylabel('Sensitivity(True Positive Rate)')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")

# Plot!
plt.show()
```



Part III Constructing LGD Using Random Forest Tree & XGBoosting

```
import pandas as pd
```

```
!gdown https://drive.google.com/uc?id=1zdZXOrj7_2li4LNZTbgMW7d12k0YqUao
```

```
Downloading...
From: https://drive.google.com/uc?id=1zdZXOrj7_2li4LNZTbgMW7d12k0YqUao
To: /content/LGD_data2.csv
47.0MB [00:00, 219MB/s]
```

```
LGD_data = pd.read_csv('LGD_data2.csv')
```

```
LGD_data.head()
```

	loan_amnt	term	emp_length	home_ownership	annual_inc	loan_status	purpose	z
0	18000.0	60 months	7	RENT	150000.0	1	debt_consolidation	
1	16000.0	36 months	10	MORTGAGE	65000.0	1	small_business	
2	10000.0	60 months	2	MORTGAGE	55000.0	1	credit_card	
3	35000.0	60 months	7	MORTGAGE	106000.0	1	debt_consolidation	
4	14000.0	60 months	2	MORTGAGE	60000.0	1	debt_consolidation	

Since we already know LGD only works with "defaulted" loans, so we can drop 'loan_status' indicator

```
LGD_data = LGD_data.drop(['loan_status'], axis = 1)
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
LGD_data.shape
```

```
(150684, 55)
```

Since prime driver of LGD in the this Loan Club is credit worthiness, salary, time at job, EAD, and geographic information.

The credit worthiness variable: we will use credit score card's variable.

salary: annual income

time at job: employ length

Partial Information fo EAD: total payment and debt settlement flag

geographic information: first letter of zip code

```
LGD_Variables = ['last_fico_range_low',
 'last_credit_pull_d',
 'total_pymnt',
 'all_util',
 'open_il_24m',
 'max_bal_bc',
 'open_act_il',
 'total_cu_tl',
 'debt_settlement_flag',
 'LGD',
 'emp_length',
 'home_ownership',
 'zip_code',
 'annual_inc']


LGD_data = LGD_data[LGD_Variables]


object_columns_df = LGD_data.select_dtypes(include=['object'])
print(object_columns_df.iloc[0])
```

```
debt_settlement_flag       N
home_ownership          RENT
Name: 0, dtype: object
```

```
LGD_data = pd.get_dummies(LGD_data)


LGD_data.isnull().sum()
```

```
last_fico_range_low        0
last_credit_pull_d         0
total_pymnt                0
all_util                   0
open_il_24m                0
max_bal_bc                 0
open_act_il                0
total_cu_tl                0
LGD                        0
emp_length                 0
zip_code                   0
annual_inc                 0
debt_settlement_flag_N     0
debt_settlement_flag_Y     0
home_ownership_MORTGAGE    0
home_ownership_OWN         0
home_ownership_RENT        0
dtype: int64
```

Now we calibrate optimal parameter for random forest

```
LGD_data.shape
```

```
(150684, 17)
```

Build Random Forest Model for LGD

Cross Validation to choose the optimal paramters for Random Forest

```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV
!pip install scorecardpy
import scorecardpy as sc
```

```
Requirement already satisfied: scorecardpy in /usr/local/lib/python3.6/dist-packages (0.1.
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: scikit-learn>=0.19.1 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from score
Requirement already satisfied: pandas>=0.25.0 in /usr/local/lib/python3.6/dist-packages (f
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (fro
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (fro
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.6/dist-packages (fr
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages (fro
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from cycler>
```

```
LGDtrain, LGDtest = sc.split_df(LGD_data, ratio = 0.66, seed = 251121253).values()
```

```
n_estimators = [int(x) for x in np.linspace(start = 500, stop = 10000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 500, num = 10)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4, 8, 16]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap }
```

```
rf = RandomForestRegressor()


LGD_random = RandomizedSearchCV(estimator = rf,
                                param_distributions = random_grid,
                                n_iter = 10,
                                cv = 10, verbose=1, random_state=251121253
                                , n_jobs = -1)


sample = LGDtrain.sample(frac = 0.01)


LGD_random.fit((sample.iloc[:,sample.columns != 'LGD']), sample['LGD'])
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
/usr/local/lib/python3.6/dist-packages/joblib/externals/loky/process_executor.py:706: User
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  9.9min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 15.7min finished
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_search.py:814: Deprecation
  DeprecationWarning)
RandomizedSearchCV(cv=10, error_score='raise-deprecating',
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   max_leaf_nodes=None,
                                                   min_impurity_decrease=0.0,
                                                   min_impurity_split=None,
                                                   min_samples_leaf=1,
                                                   min_samples_split=2,
                                                   min_weight_fraction_leaf=0.0,
                                                   n_estimators='warn',
                                                   n_jobs=None, oob_score=False,
                                                   random_st...
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 64, 118, 173, 227,
                                                      282, 336, 391, 445, 500,
                                                      None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4, 8, 16],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [500, 1555, 2611, 3666,
                                                         4722, 5777, 6833, 7888,
                                                         8944, 10000]},
                   pre_dispatch='2*n_jobs', random_state=251121253, refit=True,
                   return_train_score=False, scoring=None, verbose=1)
```

```
LGD_random.best_params_
```

```
{'bootstrap': False,
 'max_depth': 500,
 'max_features': 'sqrt',
 'min_samples_leaf': 8,
 'min_samples_split': 2,
 'n_estimators': 500}
```

```
from sklearn.model_selection import GridSearchCV
```

```
# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [False],
    'max_depth': [400, 500, 600],
    'max_features': ['auto','sqrt'],
    'min_samples_leaf': [4, 8, 16],
    'min_samples_split': [2, 4, 8],
    'n_estimators': [500, 1000, 2000]
}
# Create a based model
rf = RandomForestRegressor()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                           cv = 10, n_jobs = -1, verbose = 1)


grid_search.fit((sample.iloc[:,sample.columns != 'LGD']), sample['LGD'])
```

```
Fitting 10 folds for each of 162 candidates, totalling 1620 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  4.4min
[Parallel(n_jobs=-1)]: Done 196 tasks      | elapsed: 18.0min
[Parallel(n_jobs=-1)]: Done 446 tasks      | elapsed: 27.6min
[Parallel(n_jobs=-1)]: Done 796 tasks      | elapsed: 51.5min
[Parallel(n_jobs=-1)]: Done 1246 tasks      | elapsed: 74.2min
[Parallel(n_jobs=-1)]: Done 1620 out of 1620 | elapsed: 88.2min finished
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_search.py:814: Deprecation
  DeprecationWarning)
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=RandomForestRegressor(bootstrap=True, criterion='mse',
                                             max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             n_estimators='warn', n_jobs=None,
                                             oob_score=False, random_state=None,
                                             verbose=0, warm_start=False),
             iid='warn', n_jobs=-1,
             param_grid={'bootstrap': [False], 'max_depth': [400, 500, 600],
                         'max_features': ['auto', 'sqrt'],
                         'min_samples_leaf': [4, 8, 16],
                         'min_samples_split': [2, 4, 8],
                         'n_estimators': [500, 1000, 2000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=1)
```

```
grid_search.best_params_
```

```
{'bootstrap': False,
 'max_depth': 600,
 'max_features': 'sqrt',
 'min_samples_leaf': 8,
 'min_samples_split': 4,
 'n_estimators': 500}
```

From the randomCV and Grid-search: we have the following conclusion about the optimal hyperparameters:

{'bootstrap': False,
'max_depth': 600, (may be the more the better)
'max_features': 'sqrt',
'min_samples_leaf': 8,
'min_samples_split': 4,
'n_estimators': 500 (may be the more the better}

Build LGD Model using RandomForest Regressor

We apply the optimal parameters, since sample is 1% of the training data, we multiply the number of estimator by 10.

```
LGD_rf = RandomForestRegressor(n_estimators=5000, # Number of trees to train
                       criterion='mse', # How to train the trees. Also supports entropy.
                       max_depth= 600, # Max depth of the trees. Not necessary to change.
                       min_samples_split= 4, # Minimum samples to create a split.
                       min_samples_leaf= 8, # Minimum samples in a leaf. Accepts fractions for
                       min_weight_fraction_leaf= 0.0, # Same as above, but uses the class weigh
                       max_features='sqrt', # Maximum number of features per split (not tree!)
                       max_leaf_nodes=None, # Maximum number of nodes.
                       min_impurity_decrease=0.0001, # Minimum impurity decrease. This is 10^-3
                       bootstrap=False, # If sample with repetition. For large samples (>100.00
                       oob_score=False,  # If report accuracy with non-selected cases.
                       n_jobs=2, # Parallel processing. Set to -1 for all cores. Watch your RAM
                       random_state=251121253, # Seed
                       verbose=1, # If to give info during training. Set to 0 for silent traini
                       warm_start=False, # If train over previously trained tree.
                                   )

LGD_rf.fit(LGDtrain.iloc[:,LGDtrain.columns != 'LGD'], LGDtrain['LGD'])
```

```
[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.8s
[Parallel(n_jobs=2)]: Done 196 tasks      | elapsed:    3.4s
[Parallel(n_jobs=2)]: Done 446 tasks      | elapsed:    7.7s
[Parallel(n_jobs=2)]: Done 796 tasks      | elapsed:   13.4s
[Parallel(n_jobs=2)]: Done 1246 tasks      | elapsed:   20.8s
[Parallel(n_jobs=2)]: Done 1796 tasks      | elapsed:   30.0s
[Parallel(n_jobs=2)]: Done 2446 tasks      | elapsed:   41.0s
[Parallel(n_jobs=2)]: Done 3196 tasks      | elapsed:   53.5s
[Parallel(n_jobs=2)]: Done 4046 tasks      | elapsed:  1.1min
[Parallel(n_jobs=2)]: Done 4996 tasks      | elapsed:  1.4min
[Parallel(n_jobs=2)]: Done 5000 out of 5000 | elapsed:  1.4min finished
RandomForestRegressor(bootstrap=False, criterion='mse', max_depth=600,
                      max_features='sqrt', max_leaf_nodes=None,
                      min_impurity_decrease=0.0001, min_impurity_split=None,
                      min_samples_leaf=8, min_samples_split=4,
                      min_weight_fraction_leaf=0.0, n_estimators=5000, n_jobs=2,
                      oob_score=False, random_state=251121253, verbose=1,
                      warm_start=False)
```

```
print(LGD_rf.feature_importances_)
```

```
[0.01359394 0.20079044 0.01009458 0.01796233 0.04328685 0.03837624
 0.03768913 0.03727804 0.         0.         0.         0.27278557
 0.32814288 0.         0.         0.        ]
```

```
R2 = LGD_rf.score(LGDtest.iloc[:,LGDtest.columns != 'LGD'], LGDtest['LGD'])
```

```
[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.0s
[Parallel(n_jobs=2)]: Done 196 tasks      | elapsed:    0.1s
[Parallel(n_jobs=2)]: Done 446 tasks      | elapsed:    0.2s
[Parallel(n_jobs=2)]: Done 796 tasks      | elapsed:    0.4s
[Parallel(n_jobs=2)]: Done 1246 tasks      | elapsed:    0.6s
[Parallel(n_jobs=2)]: Done 1796 tasks      | elapsed:    0.9s
[Parallel(n_jobs=2)]: Done 2446 tasks      | elapsed:    1.1s
[Parallel(n_jobs=2)]: Done 3196 tasks      | elapsed:    1.5s
[Parallel(n_jobs=2)]: Done 4046 tasks      | elapsed:    1.9s
[Parallel(n_jobs=2)]: Done 4996 tasks      | elapsed:    2.3s
[Parallel(n_jobs=2)]: Done 5000 out of 5000 | elapsed:    2.3s finished
```

```
n = LGDtest.shape[0]
p = len(LGD_Variables)
Adj_r2 = 1-(1-R2)*(n-1)/(n-p-1)
print(R2)
print(Adj_r2)
```

```
0.2929827747483723
0.29278951766778494
```

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(LGD_rf.predict(LGDtest.iloc[:,LGDtest.columns != 'LGD']),
                         LGDtest['LGD'])
```

```
[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.0s
[Parallel(n_jobs=2)]: Done 196 tasks      | elapsed:    0.1s
[Parallel(n_jobs=2)]: Done 446 tasks      | elapsed:    0.2s
[Parallel(n_jobs=2)]: Done 796 tasks      | elapsed:    0.4s
[Parallel(n_jobs=2)]: Done 1246 tasks      | elapsed:    0.6s
[Parallel(n_jobs=2)]: Done 1796 tasks      | elapsed:    0.9s
[Parallel(n_jobs=2)]: Done 2446 tasks      | elapsed:    1.1s
[Parallel(n_jobs=2)]: Done 3196 tasks      | elapsed:    1.5s
[Parallel(n_jobs=2)]: Done 4046 tasks      | elapsed:    1.9s
[Parallel(n_jobs=2)]: Done 4996 tasks      | elapsed:    2.4s
[Parallel(n_jobs=2)]: Done 5000 out of 5000 | elapsed:    2.4s finished
```

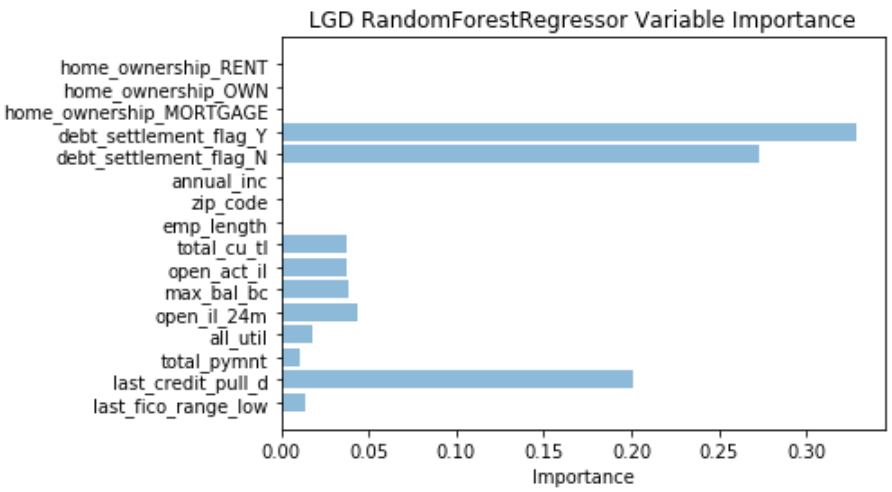
```
mse
```

```
0.008806893251565192
```

Importance of parameters for Random Forest

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
plt.barh(LGDtest.iloc[:,LGDtest.columns != 'LGD'].columns, \
         LGD_rf.feature_importances_,
        align='center', alpha=0.5)
plt.yticks(LGDtest.iloc[:,LGDtest.columns != 'LGD'].columns)
plt.xlabel('Importance')
plt.title('LGD RandomForestRegressor Variable Importance')
plt.show()
```



Build XGBoosting Model for LGD

Find the optimal parameter for the XG Boosting

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV

# Create the parameter grid based on the results of random search
param_grid2 = {
    'learning_rate': [0.1, 0.01],
    'n_estimators': [2000, 4000, 8000, 16000],
    'min_samples_leaf': [1, 4, 8, 16],
    'min_samples_split': [2, 4, 8],
    'max_depth': [2,3],
    'max_features': ['auto', 'sqrt'],
}
# Create a based model
xg = GradientBoostingRegressor()
# Instantiate the grid search model
grid_search2 = GridSearchCV(estimator = xg, param_grid = param_grid2,
                           cv = 3, n_jobs = -1, verbose = 1)



grid_search2.fit((sample.iloc[:,sample.columns != 'LGD']), sample['LGD'])
```

```
Fitting 3 folds for each of 384 candidates, totalling 1152 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:  5.0min
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed: 10.6min
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed: 18.1min
[Parallel(n_jobs=-1)]: Done 1152 out of 1152 | elapsed: 27.9min finished
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_search.py:814: Deprecation
  DeprecationWarning)
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=GradientBoostingRegressor(alpha=0.9,
                                                 criterion='friedman_mse',
                                                 init=None, learning_rate=0.1,
                                                 loss='ls', max_depth=3,
                                                 max_features=None,
                                                 max_leaf_nodes=None,
                                                 min_impurity_decrease=0.0,
                                                 min_impurity_split=None,
                                                 min_samples_leaf=1,
                                                 min_samples_split=2,
                                                 min_weight_fraction_leaf=0.0,
                                                 n_estimators=100,
                                                 n_iter...
                                                 subsample=1.0, tol=0.0001,
                                                 validation_fraction=0.1,
                                                 verbose=0, warm_start=False),
             iid='warn', n_jobs=-1,
             param_grid={'learning_rate': [0.1, 0.01], 'max_depth': [2, 3],
                         'max_features': ['auto', 'sqrt'],
                         'min_samples_leaf': [1, 4, 8, 16],
                         'min_samples_split': [2, 4, 8],
                         'n_estimators': [2000, 4000, 8000, 16000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=1)
```

```
grid_search2.best_params_
```

```
{'learning_rate': 0.01,
 'max_depth': 3,
 'max_features': 'sqrt',
 'min_samples_leaf': 8,
 'min_samples_split': 4,
 'n_estimators': 2000}
```

Build LGD Model using XGBoosting Regressor

```
from sklearn.ensemble import GradientBoostingRegressor
LGD_xg = GradientBoostingRegressor(loss='ls', #least square
                           learning_rate=0.01, # How much to shrink error in each subsequent tr
                           n_estimators=2000, # Use optimal one found above
                           subsample=0.632, # Subsampling to use. 63.2% of data is standard for
                           criterion='friedman_mse', # Error to use for each split.
                           min_samples_split=4, # Minimum samples for a split.
                           min_samples_leaf=8, # Minimum samples in a leaf.
                           min_weight_fraction_leaf=0.0, # Minimum fraction of samples in a lea
                           max_depth=3, # Maximum depth. Keep it small!
                           min_impurity_decrease=0.01, # Minimum impurity decrease. Might want
```

```
                           init=None, # How to make first prediction (it needs one). Can give m
                           random_state=251121253, # Seed
                           max_features='sqrt',
                           verbose=1,
                           max_leaf_nodes=None,
                           warm_start=False,
                           presort='auto', # Whether to presort the data to speed up training.
                           validation_fraction=0.3,
                           n_iter_no_change=None, # Iters to stop training if no change occurs
                           tol=0.0001 # Tolerance. Means maximum change of 10^-4
                          )


LGD_xg.fit(LGDtrain.iloc[:,LGDtrain.columns != 'LGD'],
              LGDtrain['LGD'])
```

```
        Iter       Train Loss      OOB Improve   Remaining Time
           1           0.0126           0.0000            1.15m
           2           0.0127           0.0000            1.04m
           3           0.0126           0.0001            1.02m
           4           0.0127           0.0001            1.01m
           5           0.0125           0.0001            1.02m
           6           0.0124           0.0000            1.00m
           7           0.0123           0.0001            1.01m
           8           0.0122           0.0001            1.01m
           9           0.0121           0.0001            1.00m
          10           0.0121           0.0000           59.67s
          20           0.0117           0.0000           58.26s
          30           0.0112           0.0000           58.28s
          40           0.0108           0.0000           57.54s
          50           0.0107           0.0000           56.94s
          60           0.0102           0.0000           56.45s
          70           0.0101           0.0000           56.04s
          80           0.0097           0.0000           55.79s
          90           0.0096           0.0000           55.39s
         100           0.0093           0.0000           55.21s
         200           0.0083           0.0000           52.19s
         300           0.0082           0.0000           49.21s
         400           0.0079           0.0000           45.99s
         500           0.0077           0.0000           42.89s
         600           0.0076           0.0000           39.90s
         700           0.0076           0.0000           36.96s
         800           0.0077           0.0000           34.00s
         900           0.0077           0.0000           31.15s
        1000           0.0074          -0.0000           28.33s
        2000           0.0071          -0.0000            0.00s
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                          learning_rate=0.01, loss='ls', max_depth=3,
                          max_features='sqrt', max_leaf_nodes=None,
                          min_impurity_decrease=0.01, min_impurity_split=None,
                          min_samples_leaf=8, min_samples_split=4,
                          min_weight_fraction_leaf=0.0, n_estimators=2000,
                          n_iter_no_change=None, presort='auto',
                          random_state=251121253, subsample=0.632, tol=0.0001,
                          validation_fraction=0.3, verbose=1, warm_start=False)
```

```
print(LGD_xg.feature_importances_)
```

```
[7.45137162e-02 2.76683342e-01 7.27902683e-02 2.60808052e-02
```

```
R2_xg = LGD_xg.score(LGDtest.iloc[:,LGDtest.columns != 'LGD'], LGDtest['LGD'])
```

```
n = LGDtest.shape[0]
p = len(LGD_Variables)
Adj_r2_xg = 1-(1-R2_xg)*(n-1)/(n-p-1)
print(R2_xg)
print(Adj_r2_xg)
```

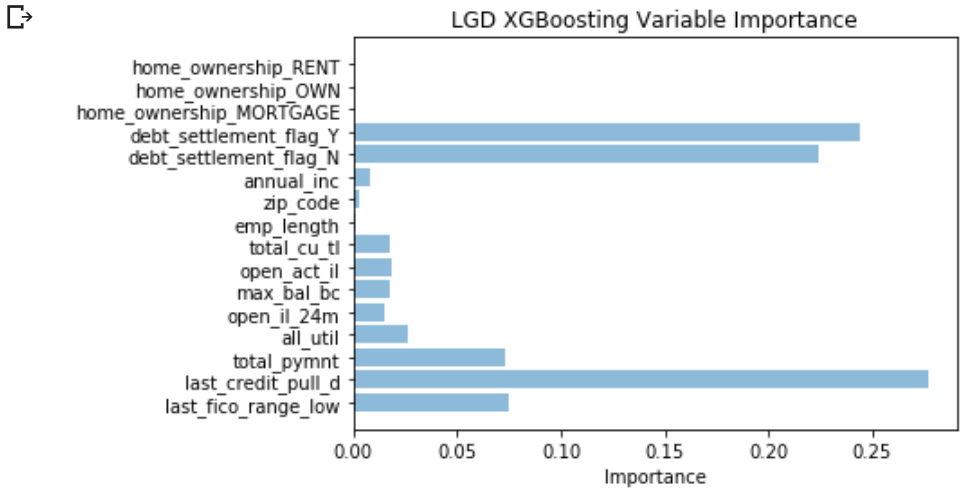
```
0.3971963273327419
0.3970315561308726
```

```
mse = mean_squared_error(LGD_xg.predict\
                         (LGDtest.iloc[:,LGDtest.columns != 'LGD']), \
                         LGDtest['LGD'])
mse
```

```
0.0075087669822225
```

Importance of parameter for XG Boosting

```
plt.barh(LGDtest.iloc[:,LGDtest.columns != 'LGD'].columns, \
         LGD_xg.feature_importances_,
        align='center', alpha=0.5)
plt.yticks(LGDtest.iloc[:,LGDtest.columns != 'LGD'].columns)
plt.xlabel('Importance')
plt.title('LGD XGBoosting Variable Importance')
plt.show()
```



Apply the regressor on the non-defaulted Loans

```
## Download the pre saved, cleaned data
!gdown https://drive.google.com/uc?id=142LOcmTXygAqkDhA7yuQwaiGIL_UUT1-
```

```
Downloading...
From: https://drive.google.com/uc?id=142LOcmTXygAqkDhA7yuQwaiGIL_UUT1-
To: /content/credit_data2.csv
129MB [00:00, 134MB/s]
```

```
Loan_data = pd.read_csv('credit_data2.csv')
```

```
## We just need the non-defaulted ones
Undefaulted = Loan_data.loc[Loan_data['loan_status'] == 0]
```

```
Undefaulted.shape
```

```
(367591, 56)
```

```
## Leave only selected variables
Undefaulted = Undefaulted[LGD_Variables]
```

```
Undefaulted = pd.get_dummies(Undefaulted)
```

```
np.any(np.isnan(Undefaulted))
```

```
True
```

```
Undefaulted.isnull().sum()
```

```
last_fico_range_low        0
last_credit_pull_d         0
total_pymnt                0
all_util                   0
open_il_24m                0
max_bal_bc                 0
open_act_il                0
total_cu_tl                0
LGD                        1
emp_length                 0
zip_code                   0
annual_inc                 0
debt_settlement_flag_N     0
debt_settlement_flag_Y     0
home_ownership_MORTGAGE    0
home_ownership_OWN         0
home_ownership_RENT        0
dtype: int64
```

```
## It seems there is a cell lost during the transmission
## It's OK to drop only one row
Undefaulted = Undefaulted.dropna(axis=0, subset=['LGD'])
```

```
Undefaulted.shape
```

```
(367590, 17)
```

Estimate the LGD for the non-defaulted class using random forest

```
LGDRF_estimate = LGD_rf.predict(Undefaulted.iloc[:, Undefaulted.columns != 'LGD'])
```

```
[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.1s
[Parallel(n_jobs=2)]: Done 196 tasks      | elapsed:    0.4s
[Parallel(n_jobs=2)]: Done 446 tasks      | elapsed:    0.8s
[Parallel(n_jobs=2)]: Done 796 tasks      | elapsed:    1.5s
[Parallel(n_jobs=2)]: Done 1246 tasks      | elapsed:    2.3s
[Parallel(n_jobs=2)]: Done 1796 tasks      | elapsed:    3.3s
[Parallel(n_jobs=2)]: Done 2446 tasks      | elapsed:    4.5s
[Parallel(n_jobs=2)]: Done 3196 tasks      | elapsed:    5.9s
[Parallel(n_jobs=2)]: Done 4046 tasks      | elapsed:    7.4s
[Parallel(n_jobs=2)]: Done 4996 tasks      | elapsed:    9.1s
[Parallel(n_jobs=2)]: Done 5000 out of 5000 | elapsed:    9.1s finished
```

```
Average_LGD_rf = LGDRF_estimate.sum() / Undefaulted.shape[0]
print(Average_LGD_rf)
```

```
0.9296144607002328
```

Estimate the LGD for the non-defaulted class using XGBoosting

```
LGDxg_estimate = LGD_xg.predict(Undefaulted.iloc[:, Undefaulted.columns != 'LGD'])
```

```
Average_LGD_xg = LGDxg_estimate.sum() / Undefaulted.shape[0]
print(Average_LGD_xg)
```

```
0.9012328189895862
```