RUTGERS BUSINESS SCHOOL
OOP - PYTHON HOMEWORK 4)

CREDIT RISK MANAGEMENT

Name: Yifu He ID: 190003956
Name: Junjian Yao ID: 197005736
Name: Haoyu Zhang ID: 198002926
Date: 20 April 2020

Contents

1 Abstract

Many rural banks have sustained heavy losses because of poor credit activities. Therefore, the project sought to establish how to distinguish the potential applicants who will default. Even though we have a lot of attributes which will influence whether lender will default. We use data mining to process the data. As for the methods, we delete the variable which has more than 5% missing, the future value and the replicated values. Then we run decision tree model, random forest model, SVM model, KNN model, LDA, QDA and logistic regression model. The accuracy is not the only judging criteria but also the precision which measures the type II error. So we choose distribution tree model. Also, further consideration are following the conclusion.
Keyword: **default risk, Classification, Decision Tree, Random Forest**

2 Description

Lending club:
https://www.lendingclub.com
Lending Club, Corp LC is the first and largest online Peer-to-Peer ("P2P") platform to facilitate lending and borrowing of unsecured loans ranging from $1,000 to $35,000. Aiming at providing lower cost transaction fees than other financial intermediaries, LendingClub hit the highest IPO in the tech sector in 2014.
We chose online peer-to-peer lending as our data for several reasons. First, this is an application area that most people can easily relate to. Second, beyond the use of predictive models, we can also develop prescriptive tools (in this context, investment strategies) so we can directly observe the potential business impact of our models. Third, the largest two online U.S. platforms make their data publicly available, allowing readers to easily reproduce and extend our results.
Objective:
To build a machine learning model for credit risk management, we collect the data from the lending club which peer-to-peer online lending platform. The dataset covers an extensive amount of information on the borrower's side that was originally available to lenders when they made investment choices. By further segmenting the loan dataset into finished cases and current outstanding loans, this project breaks down the composition of the default cases and examines the correlation among indicators. In the end, the goal is to provide investors and borrowers, as well as Lending Club, additional insights regarding investment opportunities and contingent loan collection advice.
How lending club works:
1. Customers interested in a loan complete a simple application at LendingClub.com
2. LC leverage online data and technology to quickly assess risk, determine a credit rating and assign appropriate interest rates.
3. Qualified applicants receive offers in just minutes and can evaluate loan op-

tions with no impact to their credit score.
4. Investors ranging from individuals to institutions select loans in which to invest and can earn monthly returns.
Data Source:
All data regarding this project can be accessed https://www.lendingclub.com. As for the analysis or data scrubbing, you can check the data dictionary. Then you can have a quick check for what variables you want.

3 Python Module I used

Module for data processing: pandas, numpy, ipywidgets, IPython.display, pdfkit
Module for Machine Learning model: sklearn
Module for visualization: seaborn, matplotlib
How to downloan: **pip install** packagename

4 reference

https://zhuanlan.zhihu.com/p/34782497
https://zhuanlan.zhihu.com/p/38687978
https://www.cnblogs.com/pinard/p/6056319.html
https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python
https://www.datacamp.com/community/tutorials/random-forests-classifier-python
https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn
https://zhuanlan.zhihu.com/p/35182003
https://bacterous.github.io/2018/09/13/LightGBM

https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/

http://ywtail.github.io/2017/06/08/sklearn

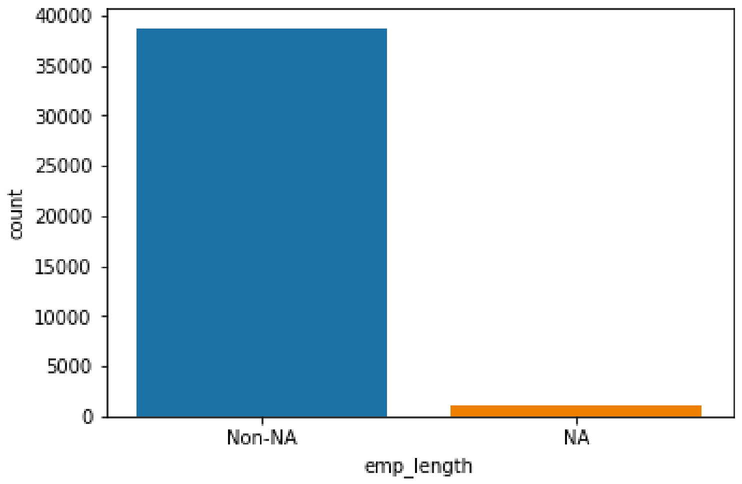
5 Data Pre-processing

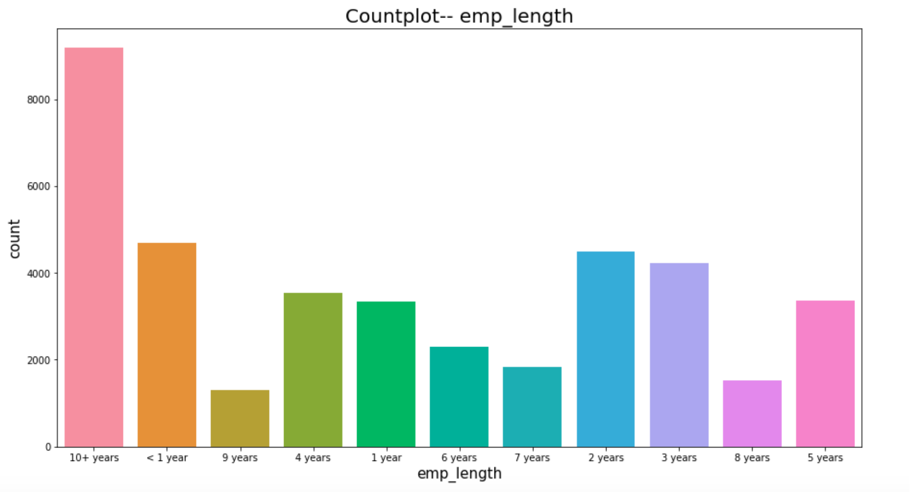
5.1 Missing Value

We calculated all missing value from the data set. We cannot just delete the instances which contain missing values, because of bias problem. Set the criterion:
1. If the missing part of each attribute is **greater than 95%**, we will delete this attribute.
2. If the missing part of each attribute is **less than 95%**, we will fill the missing

part (we use **forward fill** from pandas).

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2b4a2048>
```





After filtering, we show the attributes left:

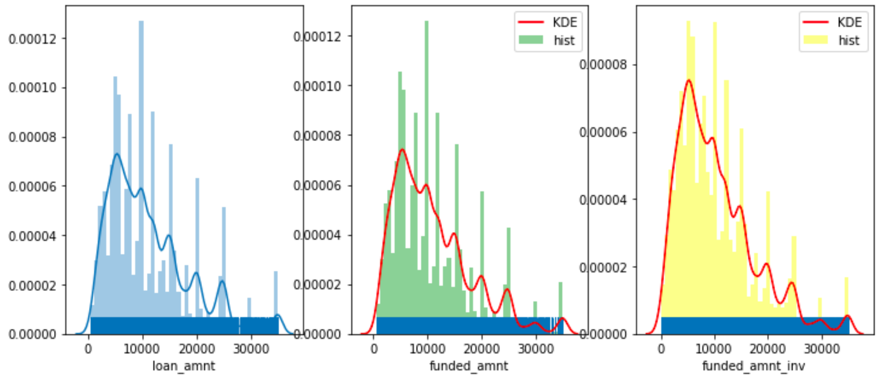
```
After delete the attributes with missing values: cleaned1.csv
Shape of DataFrame: (39786, 52)
loan_amnt       funded_amnt     funded_amnt_inv term    int_rate        installment     grade   sub_grade
emp_length      home_ownership  annual_inc      verification_status     issue_d loan_status     pymnt_plan
purpose title   zip_code        addr_state      dti     delinq_2yrs     earliest_cr_line        inq_last_6mths
open_acc        pub_rec revol_bal       revol_util      total_acc       initial_list_status     out_prncp
out_prncp_inv   total_pymnt     total_pymnt_inv total_rec_prncp total_rec_int   total_rec_late_fee      recoveries
collection_recovery_fee last_pymnt_d    last_pymnt_amnt last_credit_pull_d      collections_12_mths_ex_med
policy_code     application_type        acc_now_delinq  chargeoff_within_12_mths        delinq_amnt
pub_rec_bankruptcies    tax_liens       hardship_flag   disbursement_method     debt_settlement_flag
```

5.2 Feature Subset Selection

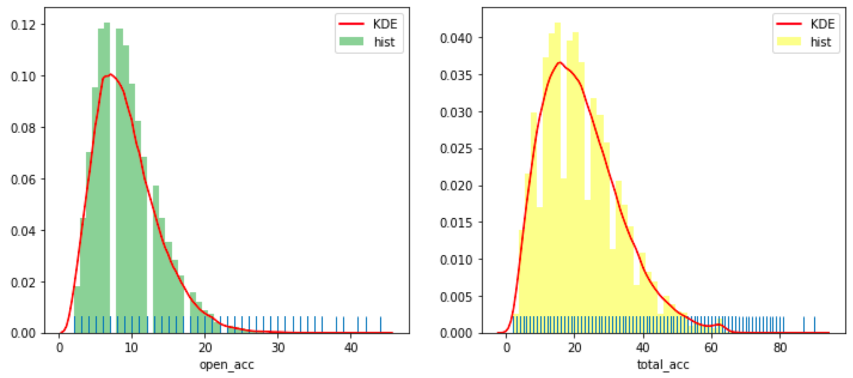
Selection criterion:
1. Duplicated Value If one of the class in the attribute takes up more than **95%**, we delete this attribute.
2. Colinearity Some of the attributes may have colinearity problem .For example, we plot the distribution of loan_amnt, funded_amnt, funded_amnt_inv, we can see they have similar distribution. After we did the VIF test, we can confirm they have highly correlations. Thus, we just keep one and delete the rest of them.
VIF test:

	VIF Factor	features
0	3.3	Intercept
1	27.3	loan_amnt
2	39.1	funded_amnt
3	12.3	funded_amnt_inv

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a36b10550>
```



```
<matplotlib.axes._subplots.AxesSubplot at 0x1a3a099518>
```



3. Future Information Some of the features from the original data set cannot be obtained when the client apply for the loan, such as the recovery rate or the last payment amount. Since we cannot use the future information to infer future, we delete those attribute.
4. Similar Information Some of the features may have similar information, such as zip code and state, which both reflect information about location. We delete zip code and just keep state.
5. Date We delete the attribute in the form of date.

5.3 Feature Creation

When we first counted the missing value, we found that those missing values have larger portion in the default class "Charged Off" than in the non-default class "Fully Paid". We assume that those missing value can reflect some property of potentially default client. For example, they may not be able to offer some important information to prove they are qualified enough to pay off their debt in the future. Thus, we create one feature,**missing_amount**, to represent this property.

After we finished feature selection, those are the features we selected:

```
new3.columns
```

```
Index(['loan_amnt', 'term', 'emp_length', 'home_ownership', 'annual_inc',
       'verification_status', 'loan_status', 'purpose', 'addr_state', 'dti',
       'delinq_2yrs', 'open_acc', 'missing_amnt'],
      dtype='object')
```

We stuck the statistics and distribution chart of all the left features in **Appendix A**.

5.4 Scaling

1.Categorical Variable: Convert categorical variables into binary dummy variables. We use OneHotEncoder to perform this task.

2.Numerical Variable: We use minmax normalization as following.

$$z = \frac{x \text{ - } \min(x)}{\max(x) \text{ - } \min(x)}$$

Some alternative methods are: Standard Score.

$$X_{changed} = \frac{X - \mu}{\sigma}$$

5.5 Separation of Training Set and Test Set

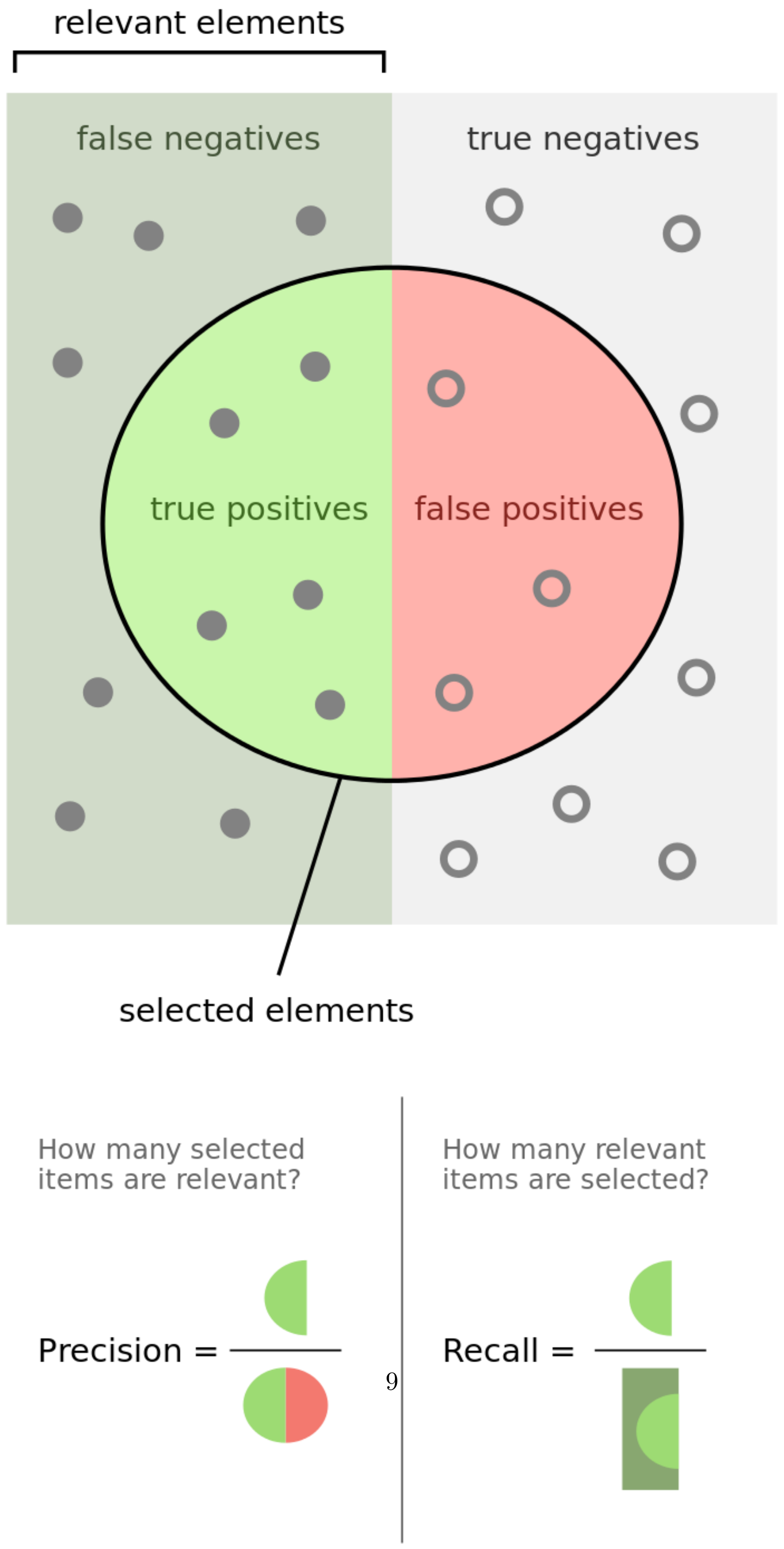
The ratio of **Training Set : Test Set = 9:1**.
Purpose of splitting data into validation set and training set is to avoid overfitting which is paying attention to minor details/noise which are not necessary and only optimizes the training dataset accuracy. We need a model that performs well on dataset that it has never seen (test data), this is called generalization.

6 Model Performance and Selection

Generally, for all the classification model, we have accuracy, precision, recall and F-measure to evaluate the performance of models. For this scenario, we need to pay attention to the precision, which is the ratio of true positive to all the instance which are categorized into positive. Because from the perspective of banks, it's more risky to approve someone who has high potential to default than to reject someone who has good credit situation.

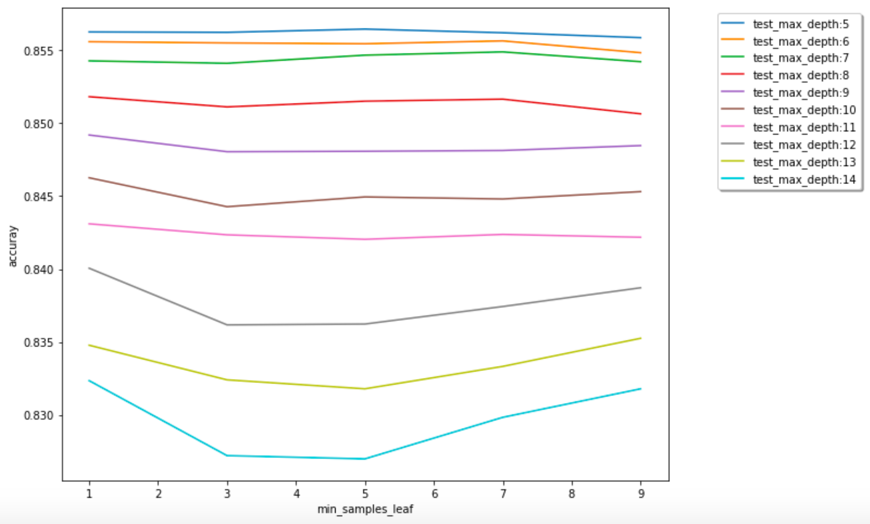
6.1 Decision Tree

All the code are in the **Appendix B**, we just articulate the logic here.



We used GridSearchCV method to find optimal parameter in the Decision Tree model.
Here is the result:

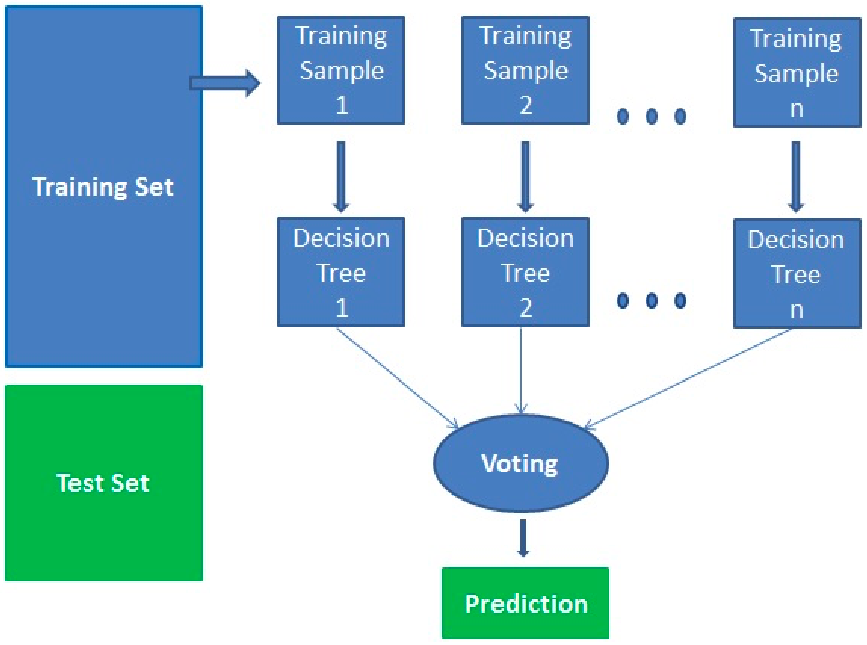
```
Best: 0.856453 using {'max_depth': 5, 'min_samples_leaf': 5}
The AUC of GridSearchCV Desicion Tree is 0.4998540572095738
```



We found that the algorithm can only give us the criterion of accuracy if the selection criterion is based on accuracy, the model works actually very badly. Because there are greater portion of ”Fully Paid” than ”Charged Off” in the dataset. The most accurate model might be the simplest one which just put every instance into ”Fully Paid”. This is another reason why we need to choose precision to select models.

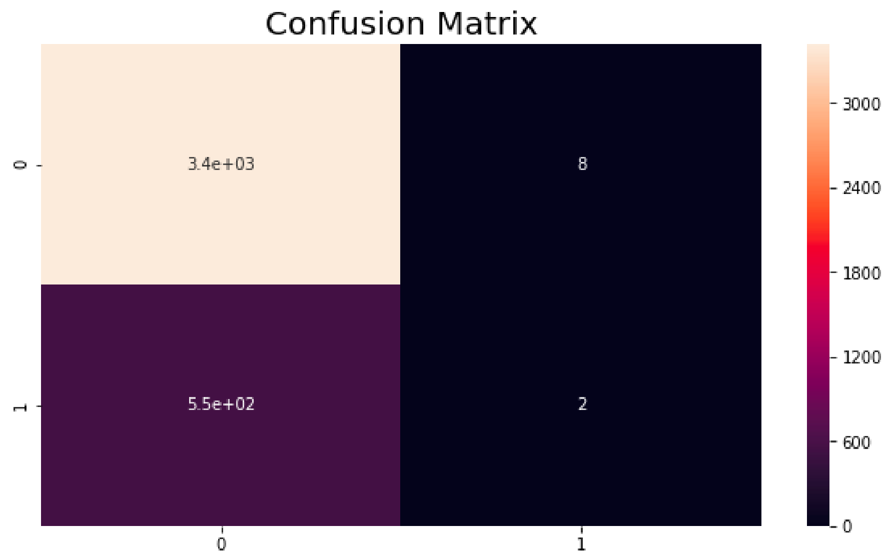
6.2 Random Forests

For Random Forests model, the algorithm use some method, such as bootstrap to generate some subset from the original dataset, then run them in subtrees and finally aggregate the result to perform the classification. Here is the workflow:

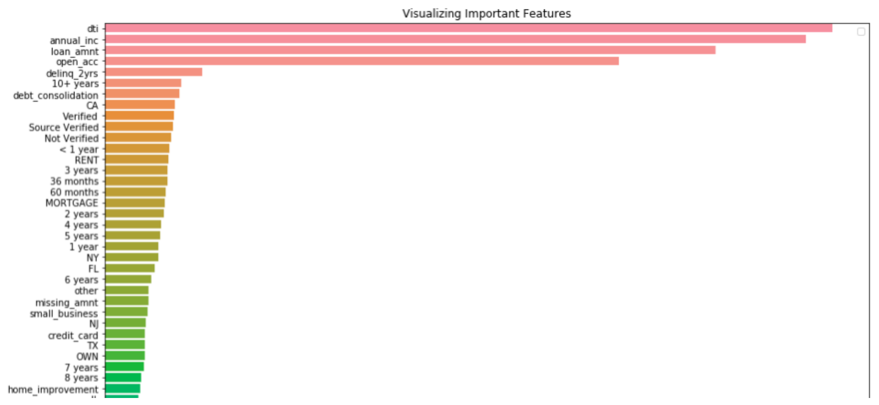


After we run the Random Forests model, we can find that the model accuracy is significant higher than Decision Tree. However, when we print its confusion matrix, we can find it is because of the same problem we mentioned before. The algorithm categorized almost every instance into "Fully Paid".

```
Accuracy: 0.8595124403116361
[[3418    8]
 [ 551    2]]
```

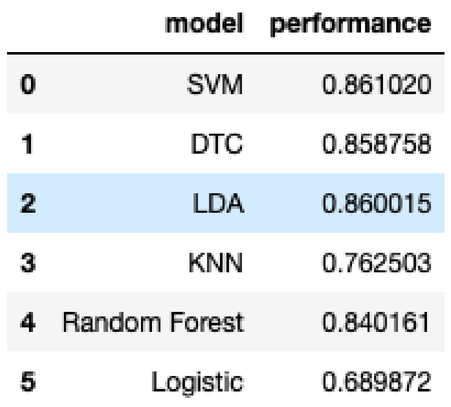


However, we can still generate the weight of each features in the algorithm and find the most important features.
We plot them here, cause the graph is too large, we just keep the top of it. From the graph, we can actually find that some important features are actually accord with our anticipation, such as ”dti”, ”annual_inc”, ”loan_amnt”,”open_acc” and ”delinq_2yrs”.

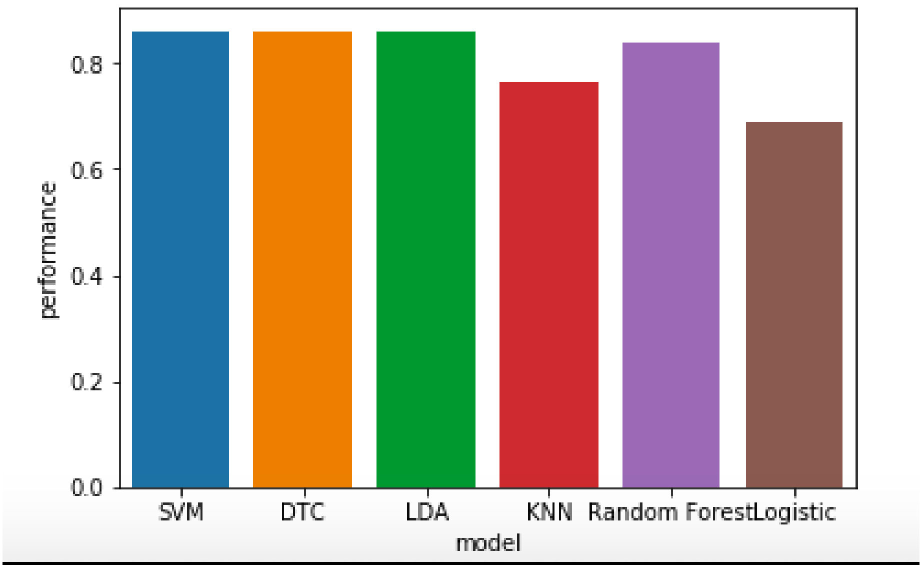


6.3 SVM,Logistic Regression, LDA&QDA, KNN

For those 5 algorithm, the criterion precision is quite close. We visualized their precision:



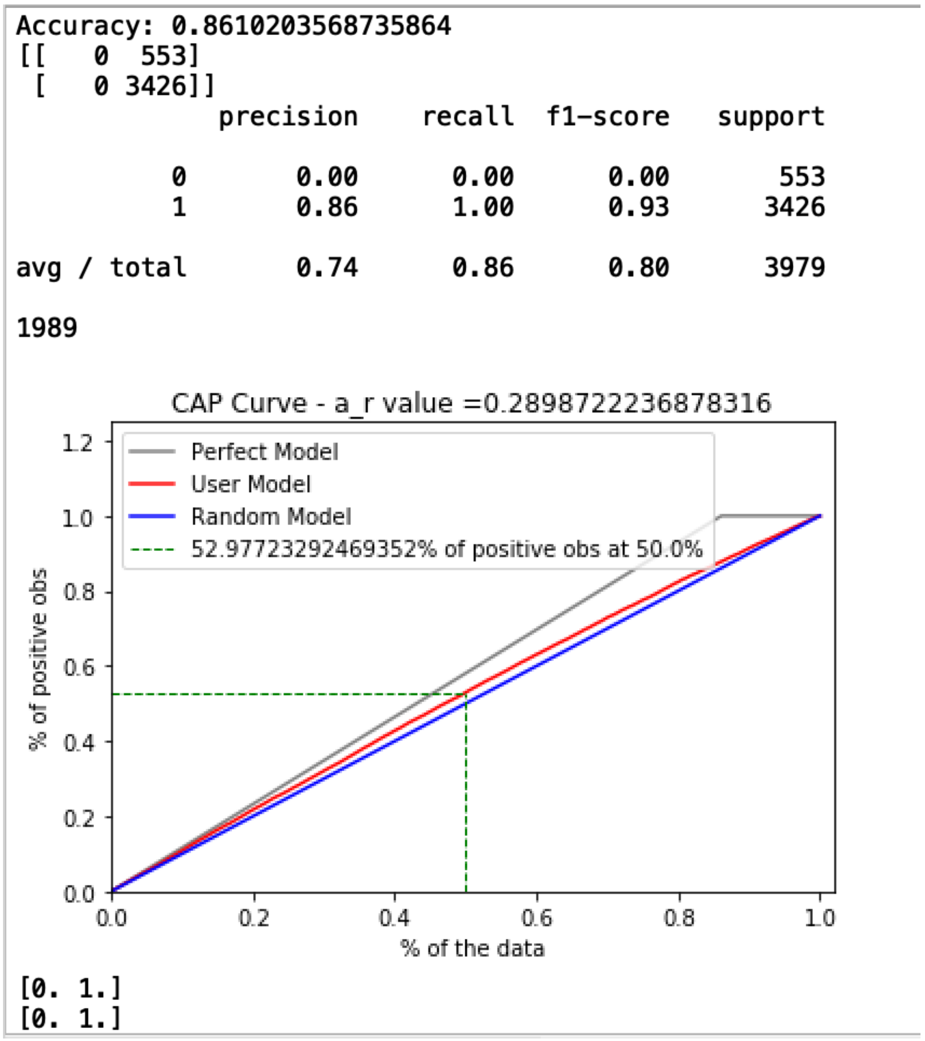
	model	performance
0	SVM	0.861020
1	DTC	0.858758
2	LDA	0.860015
3	KNN	0.762503
4	Random Forest	0.840161
5	Logistic	0.689872



According to Occam's Razor principal, when the performance of model are similar, we should choose the simplest one which can be more interpretable.
For Support Vector Machine, LDA & QDA models: When the dimention is greater than 3, it is hard to visualize the classification boundary of two different classes. It's sophisticated and time-consuming to use the model and also hard to interpret to clients.
For KNN and logistic regression model: The precision is lower than the former 4 models, so we don't use them.
Therefore, we use Decision Tree model, because it has a clear workflow for users to determine which features have great influence on the result, which will also give the information at which stage the clients are classified.

7 Result Analysis and Evaluation

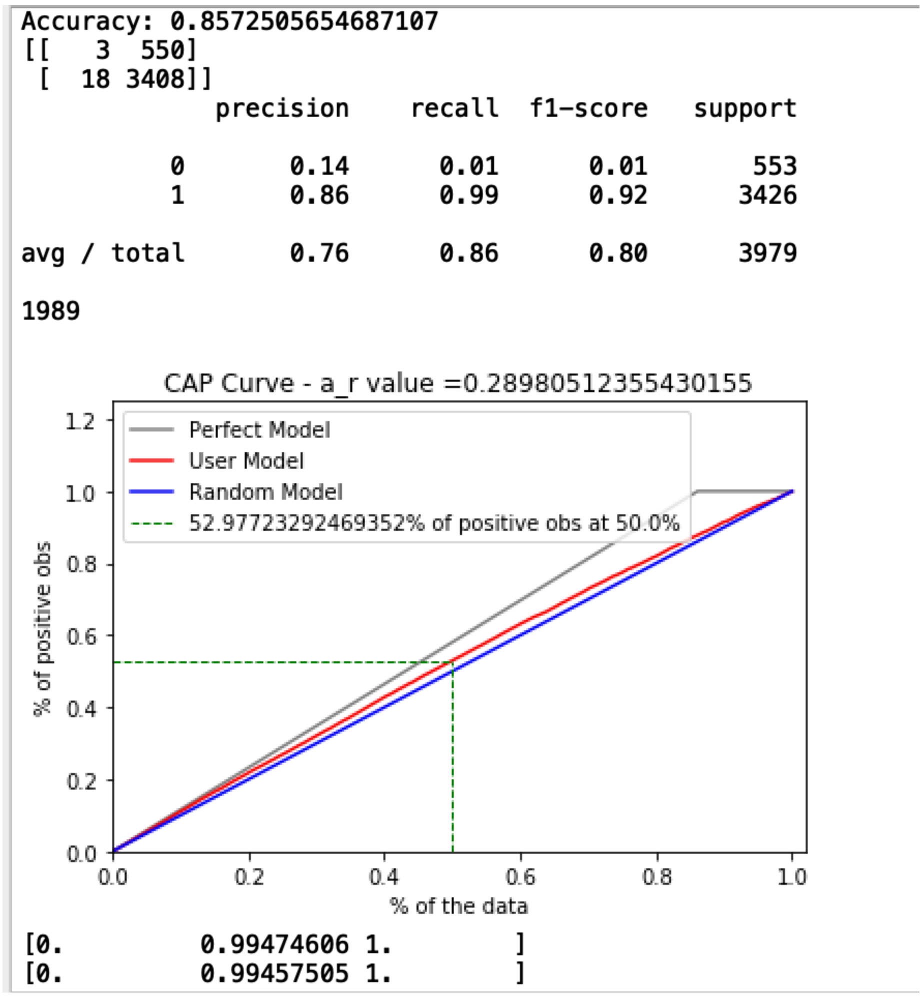
In the previous chapter, we selected Decision Tree model because of its interpretability and precision. After we changed it's parameter, we printed the confusion matrix and CAP curve of different parameter(max_depth). Here are the different plots.



```
Accuracy: 0.8610203568735864
[[   0  553]
 [   0 3426]]
             precision    recall  f1-score   support

          0       0.00      0.00      0.00       553
          1       0.86      1.00      0.93      3426

avg / total       0.74      0.86      0.80      3979

1989
```

```
[0. 1.]
[0. 1.]
```

Figure 1: Result for max_depth=3

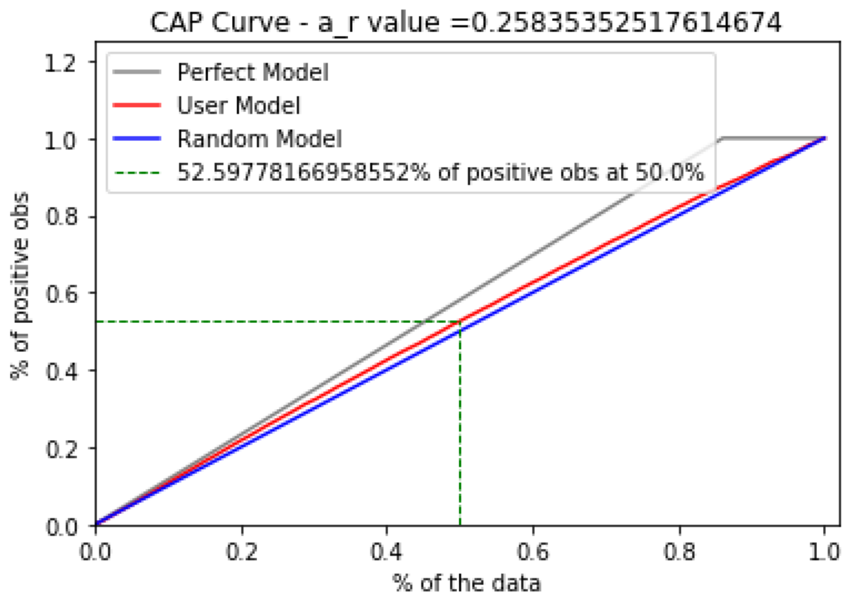


```
Accuracy: 0.8572505654687107
[[   3  550]
 [  18 3408]]
             precision    recall  f1-score   support

          0       0.14      0.01      0.01       553
          1       0.86      0.99      0.92      3426

avg / total       0.76      0.86      0.80      3979

1989
```

```
[0.         0.99474606 1.        ]
[0.         0.99457505 1.        ]
```

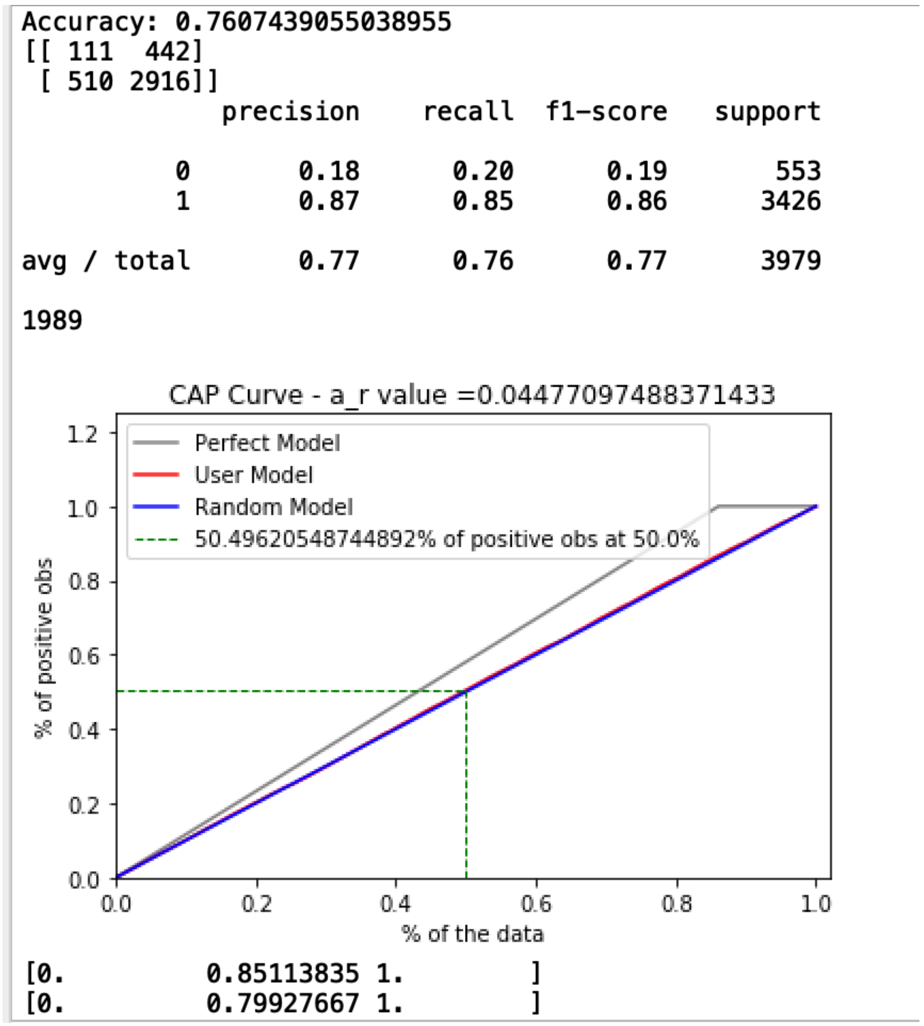
Figure 2: Result for max_depth=7

```
Accuracy: 0.8529781352098518
[[  23  530]
 [  55 3371]]
             precision    recall  f1-score   support

          0       0.29      0.04      0.07       553
          1       0.86      0.98      0.92      3426

avg / total       0.79      0.85      0.80      3979

1989
```



```
[0.         0.98394629 1.        ]
[0.         0.95840868 1.        ]
```

Figure 3: Result for max_depth=10



```
Accuracy: 0.7607439055038955
[[ 111  442]
 [ 510 2916]]
             precision    recall  f1-score   support

          0       0.18      0.20      0.19       553
          1       0.87      0.85      0.86      3426

avg / total       0.77      0.76      0.77      3979

1989
```

```
[0.         0.85113835 1.        ]
[0.         0.79927667 1.        ]
```

Figure 4: Result for no max_depth

As we could see, different parameter(max_depth) leads to totally different result. For max_depth=3 we may conclude that all the customers are classified into non-default, and that's why we obtain around 86% result. For the parameter=3, 7, 10 and no parameter, we have slightly lower accuracy, however, more customers are classified into default. This implies that our model really works. For the result for no parameter, we could conclude from the confusion matrix, cap curve and (TPR, FPR) value that it is more efficient than random guess, which is a satisfactory result.

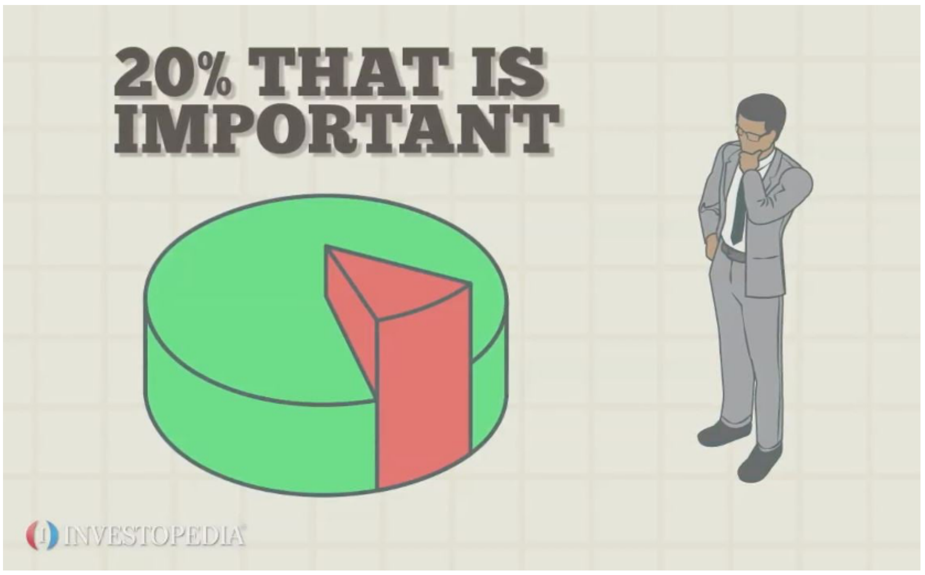
8 Further Rumination

8.1 Overfitting Problem

When we tested different combination of features, we found that there weren't significant enhancement for our result. We think the reasons may be that, when we use One-hot-encoder method to convert variables, the dimension become too high. It will cause overfitting problem because of high complexity in the model.
Solution Proposal:
Using dimension reduction technique such as principal component analysis or factor analysis to reduce dimension and then keep the important property from dataset.
Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components. In our model some of the factors obviously have some linear relationship, we could resolve the problem by PCA to generate fewer features that are more significant.
Factor Analysis searches for joint variations in response to unobserved latent variables. The observed variables are modelled as linear combinations of the potential factors, plus "error" terms. Factor analysis aims to find independent latent variables. We could carry out some linear combination for the factors we already obtained using factor analysis to generate fewer features that are more significant.

8.2 Unbalance of Data

The second is unbalance data. The problem we had is when we train our data using different models, the accuracy we obtain is usually around 0.86. We found that the portion of non-default data in the whole dataset is around 86%, and these models have classified all the data into non-default. We think this due to the unbalance of data. For example, if there are data for 99 cats and 1 dog, we could easily recognize the data for cats, however for the dog there are too few data for us to catch the character of the dog.
It is similar with the Pareto's principle which is 20% people grasp 80% wealth.



Solution Proposal:
If we want to solve this problem, we have several ways. The first is oversample and undersample. The oversample is to duplicate some small classes and then we got a balanced data. The undersample is to delete some large classes and then we got a balanced data.

The biggest advantage of random sampling is simplicity, but the disadvantages are also obvious. After the upsampling data set, there will be some samples repeatedly, and the trained model will have a certain overfitting. The disadvantage of downsampling is obvious, that is, the final training set loses data, and the model only learns a part of the overall pattern.

Upsampling will make multiple copies of the niche sample, and a point will repeatedly appear in high-dimensional space, which will cause a problem that if you are lucky, you can divide many points, otherwise you will divide many points. To solve this problem, a slight random perturbation can be added every time a new data point is generated, and experience has shown that this approach is very effective.

Because down-sampling will lose information, how to reduce the loss of information? The first method is called EasyEnsemble. It uses model fusion (Ensemble): multiple downsampling (putting back the sampling so that the training set generated is independent of each other) to generate multiple different training sets, and then train multiple different classifiers. The final result is obtained by combining the results of multiple classifiers. The second method is called BalanceCascade, which uses the idea of incremental training (Boosting): first generate a training set by one downsampling, train a classifier, do not put back those correctly classified popular samples, and then apply this smaller public sample Downsampling generates a training set, trains a second classifier, and so on, and finally combines the results of all classifiers to get the final result. The third method is to use KNN to try to select the most representative popular samples, called NearMiss, which is very computationally intensive.

8.3 Information in Missing Value

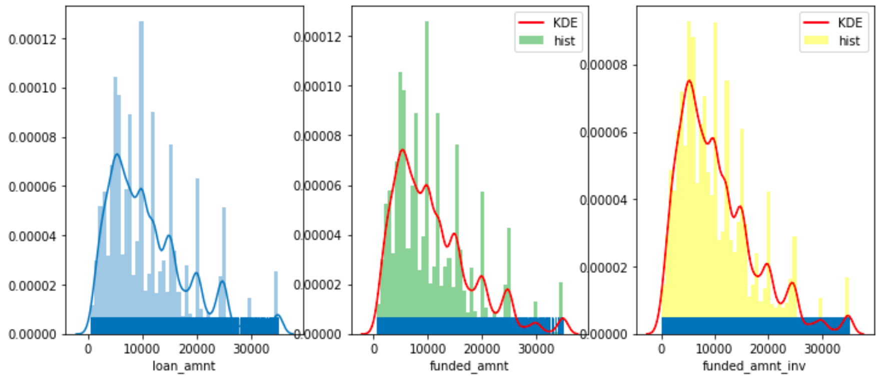
In the data preprocessing part, we happened to delete some instances with missing value directly without considering bias problem. The ratio of ”Fully Paid” : ”Charged Off” changed from 0.86 to 0.93. It came to us that those who have missing value may have higher default probability. We can think like this: those who may default in the future may intentionally ignore some information while filling out the application form in order to avoid giving valid information resulting in application disapproval. It means that the amount of missing value itself is a property of each client. Inspired by this, we create a new attribute ”missing_amnt” in the dataset.

9 Appendix

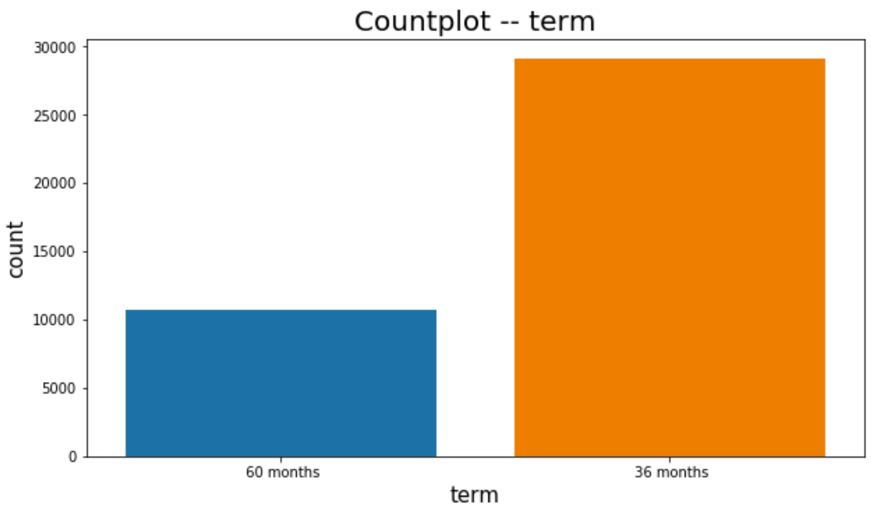
9.1 Appendix A: Distribution and Statistics of Features

1.loan_amnt

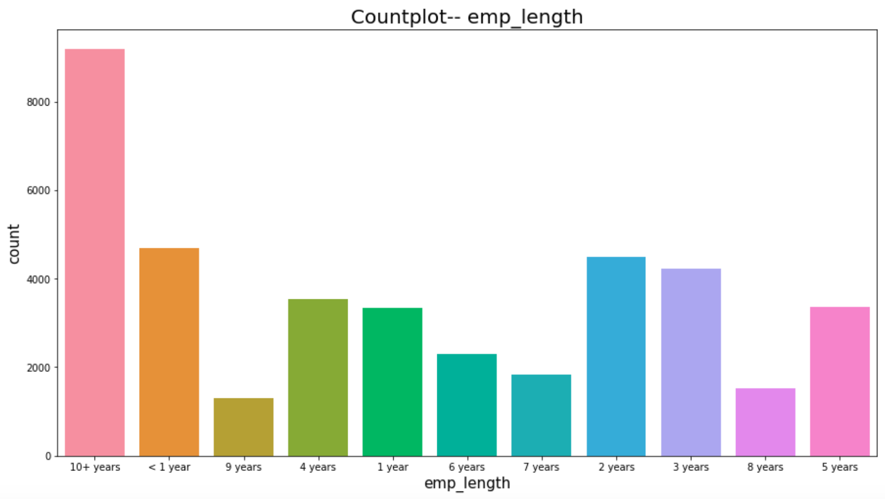
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a36b10550>
```



2.term

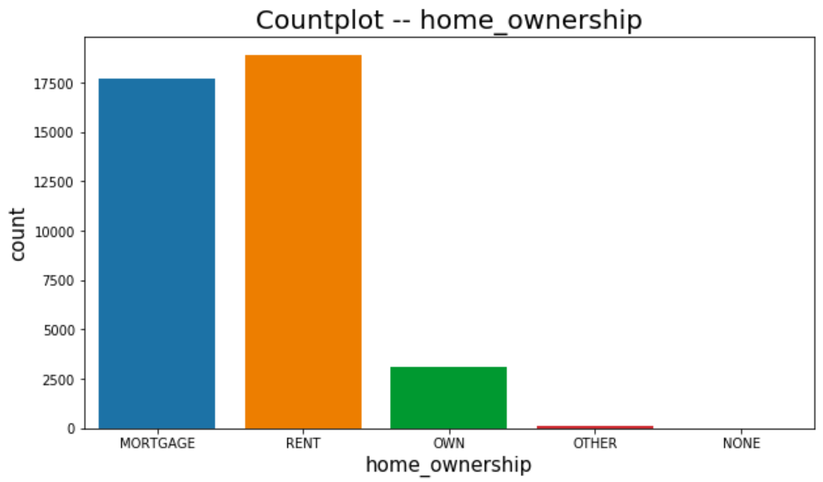


3.emp_length



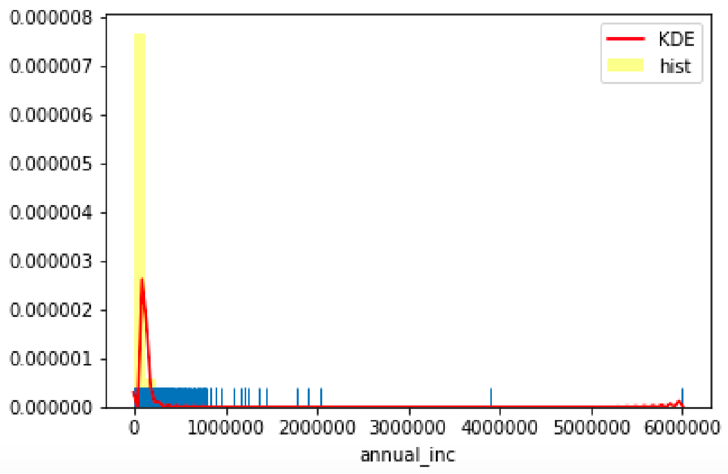
4.home_ownership

```
count      39786
unique         5
top         RENT
freq       18918
Name: home_ownership, dtype: object
```



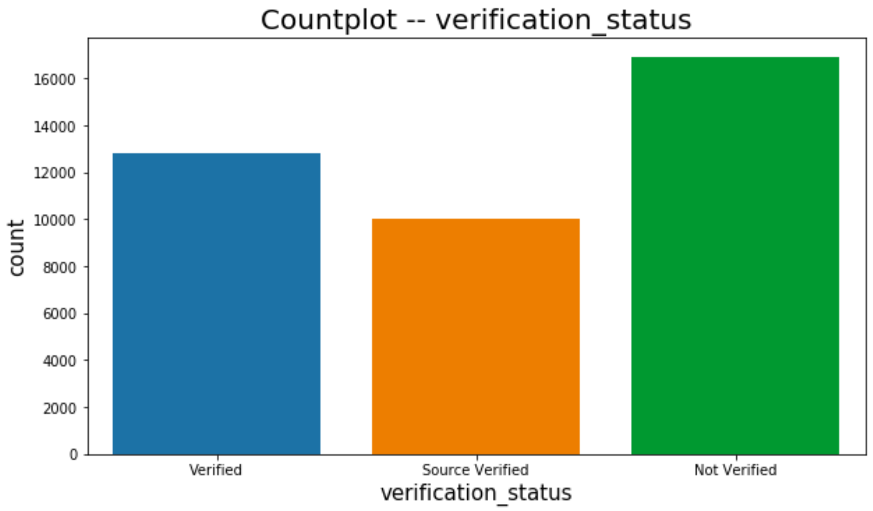
5.annual_in

```
count    3.978600e+04
mean     6.897907e+04
std      6.376263e+04
min      4.000000e+03
25%      4.050000e+04
50%      5.900000e+04
75%      8.234250e+04
max      6.000000e+06
Name: annual_inc, dtype: float64

<matplotlib.axes._subplots.AxesSubplot at 0x1a2ec87390>
```



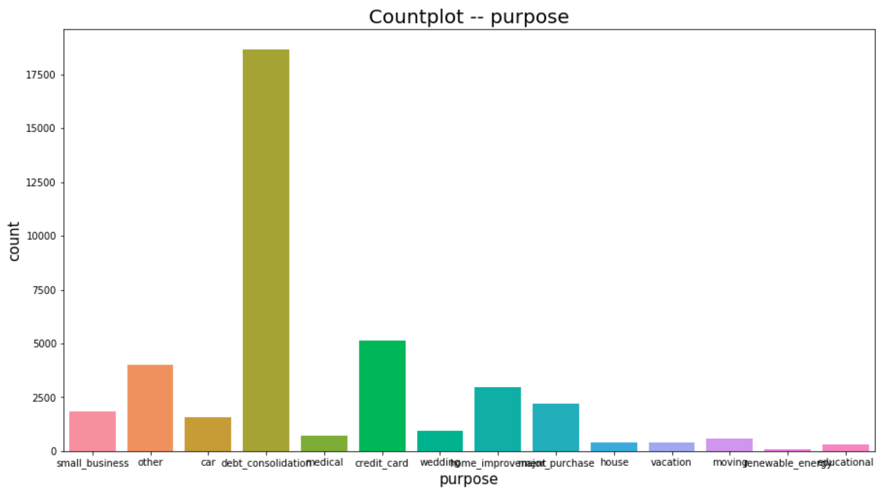
6.verification_status

```
count             39786
unique                3
top        Not Verified
freq              16926
Name: verification_status, dtype: object
```

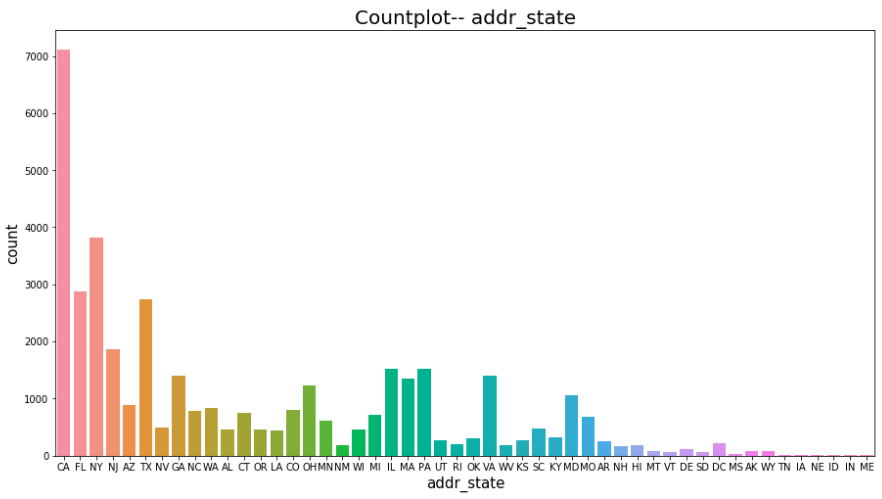


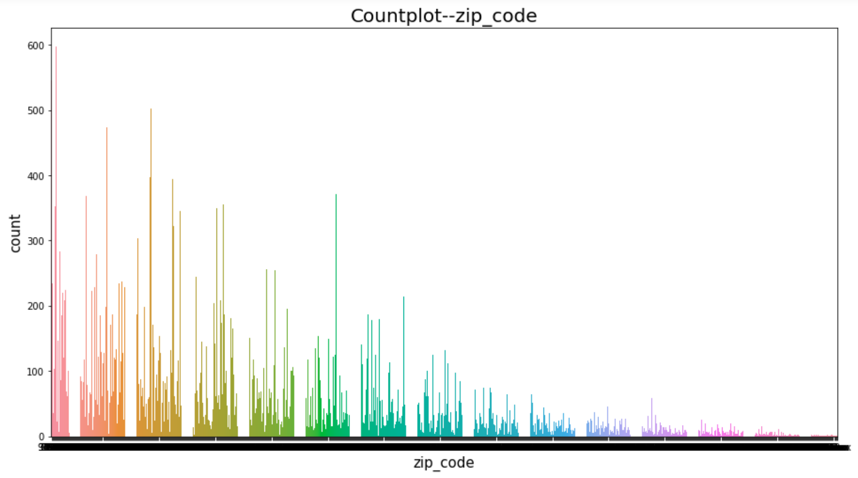
7.purpose

```
count                   39786
unique                     14
top        debt_consolidation
freq                    18676
Name: purpose, dtype: object
```



8.addr_state



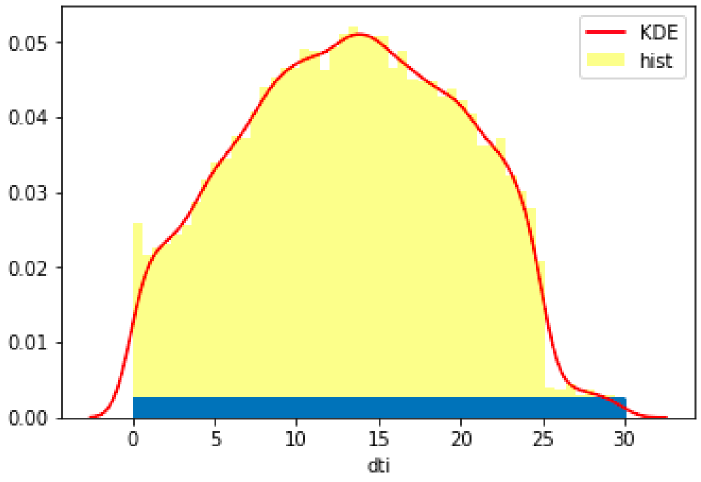


```
count      39786
unique       823
top        100xx
freq         597
Name: zip_code, dtype: object
```

9.dti

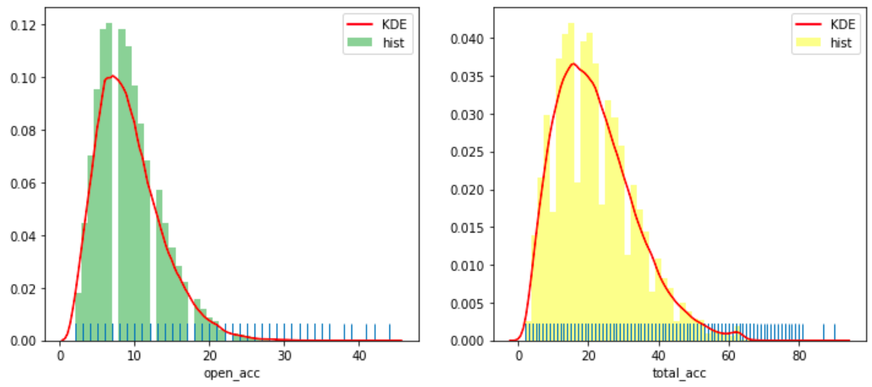
```
count    39786.000000
mean        13.317794
std          6.678300
min          0.000000
25%          8.180000
50%         13.410000
75%         18.600000
max         29.990000
Name: dti, dtype: float64
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a31099320>
```



10.open_acc

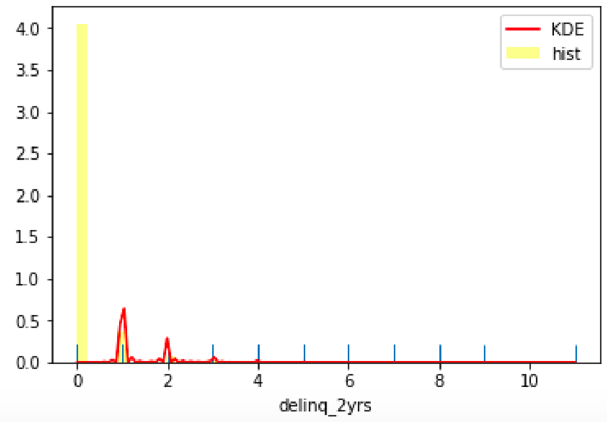
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a3a099518>
```



11.delinq_2yrs

```
count    39786.000000
mean         0.146534
std          0.491826
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max         11.000000
Name: delinq_2yrs, dtype: float64
```

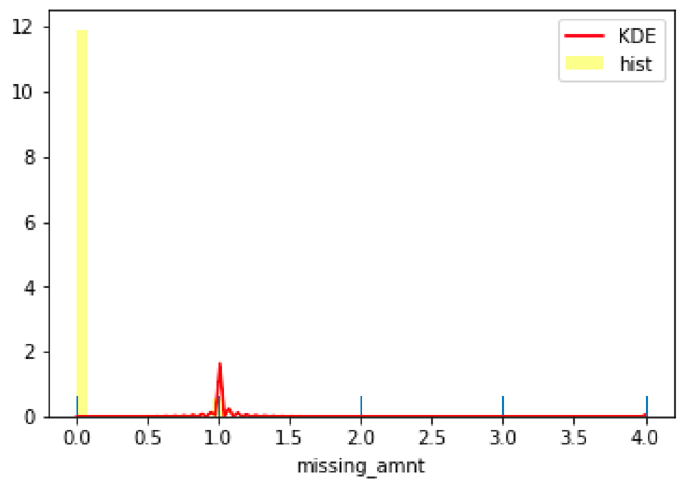
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a32444f28>
```



12.missing_amnt

```
count    39786.000000
mean         0.051802
std          0.252883
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          4.000000
Name: missing_amnt, dtype: float64
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a314d3400>
```



9.2 Appendix B: Python Code

```
import pandas as pd
import numpy as np
from ipywidgets import interact, IntSlider
from IPython.display import display
import pdfkit as pdf
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

#### freeze window to checkup the data ############################
def freeze_header(df, num_rows=1000, num_columns=10, step_rows=1000
                                                   ,
                  step_columns=1):
    """
    Freeze the headers (column and index names) of a Pandas
                                       DataFrame. A widget
    enables to slide through the rows and columns.
    Parameters
    ----------
    df : Pandas DataFrame
        DataFrame to display
    num_rows : int, optional
        Number of rows to display
    num_columns : int, optional
        Number of columns to display
    step_rows : int, optional
        Step in the rows
    step_columns : int, optional
        Step in the columns

    Returns
    -------
    Displays the DataFrame with the widget
    """
    @interact(last_row=IntSlider(min=min(num_rows, df.shape[0]),
                                 max=df.shape[0],
                                 step=step_rows,
                                 description='rows',
                                 readout=False,
                                 disabled=False,
                                 continuous_update=True,
                                 orientation='horizontal',
                                 slider_color='purple'),
              last_column=IntSlider(min=min(num_columns, df.shape[1
                                                               ]),
                                    max=df.shape[1],
                                    step=step_columns,
                                    description='columns',
                                    readout=False,
                                    disabled=False,
                                    continuous_update=True,
                                    orientation='horizontal',
                                    slider_color='purple'))
    def _freeze_header(last_row, last_column):
        display(df.iloc[max(0, last_row-num_rows):last_row,
                        max(0, last_column-num_columns):last_column
                                                                  ])

### 1.Original Data import ###################################
path = "/Users/yifuhe/Learning/RBS/fixed_income/finalproject/"
#T1 = pd.read_excel("/Users/yifuhe/Learning/RBS/fixed_income/
                                  finalproject/
                                  LendingClubData_training.xlsx")
#T2 = pd.read_excel("/Users/yifuhe/Learning/RBS/fixed_income/
                                  finalproject/
                                  LendingClubData_testing.xlsx")
#data = pd.concat([T1,T2])
#data = data.reset_index().drop(columns=["index"])
#data.to_csv(path+"LendingClubData_original.csv",index = False)
data = pd.read_csv(path + "LendingClubData_original.csv")
freeze_header(data)

### 2.data preprocessing ###################################
```

```
# deal with missing value
# data preprocessing
print("Original data: LendingClubData_original.csv")
print(f"Shape of DataFrame: {data.shape}")
#print(f"Check the missing value:\n {data.isnull().sum()}")
freeze_header(data)
#data.columns[data.isnull().sum()>data.shape[0]*0.05]
# data preprocessing
print("Original data: LendingClubData_original.csv")
print(f"Shape of DataFrame: {data.shape}")
#print(f"Check the missing value:\n {data.isnull().sum()}")
freeze_header(data)
#data.columns[data.isnull().sum()>data.shape[0]*0.05]

# feature creation: missing_amnt
new1.loc[:,"missing_amnt"] = new1.isnull().sum(axis=1)


# save the data set
#new1.to_csv(path+"cleaned1.csv",index= False)
print("After delete the attributes with missing values: cleaned.csv
                                  ")
print(f"Shape of DataFrame: {new1.shape}")
for i in new1.columns:
    print(i,end = "\t")

print(new1["missing_amnt"].describe())
sns.distplot(new1["missing_amnt"],rug=True,
                     hist_kws={'color':'yellow','label':'hist'},
                     kde_kws={'color':'red','label':'KDE'})

# deal with replicated values
# delete the attribute with replicated values ()
temp = new1.describe(include=["object"])
temp.columns[temp.loc["freq"] > n_instances * 0.95]
new2 = new1.drop(temp.columns[temp.loc["freq"] > n_instances * 0.95
                                  ], axis=1)

temp2 = new1.describe()
new2 = new2.drop(temp2.columns[temp2.loc["std"] == 0],axis=1)

#print(new2.isnull().sum())

new2.to_csv(path+"cleaned.csv",index= False)
print("After delete the duplicated: cleaned.csv")
print(f"Shape of DataFrame: {new2.shape}")
freeze_header(new2)
# count the frequency
#pd.value_counts(new2.iloc[:,3])
#plt.hist(new2.iloc[:,3])
new2.columns

temp = new1.describe(include=["object"])
temp2 = new1.describe()
new2.loc[:,new2.columns[new2.dtypes==object]].head()

## check each attribute to determine whether keep it
## colinearity: ["loan_amnt", "funded_amnt", "funded_amnt_inv"]
                                    these 3 attributes are similar,
                                    just choose 1
# check each indivivual attribute
new2 = pd.read_csv(path+"cleaned.csv")
new3 = new2

label = new3.loc[:,["loan_status"]]
print(f"The number of instance with no label: {label.isnull().sum()
                                    }")
new3.columns[new3.isnull().sum()!=0]

# visualization
fig,axes=plt.subplots(1,3,figsize=(12,5))
sns.distplot(new3["loan_amnt"],rug=True,ax=axes[0])
sns.distplot(new3["funded_amnt"],rug=True,
                     hist_kws={'color':'green','label':'hist'},
                     kde_kws={'color':'red','label':'KDE'},
                     ax=axes[1])
sns.distplot(new3["funded_amnt_inv"],rug=True,
                     hist_kws={'color':'yellow','label':'hist'},
                     kde_kws={'color':'red','label':'KDE'},
                     ax=axes[2])

# VIF test

#Imports
import pandas as pd
import numpy as np
from patsy import dmatrices
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import
                                    variance_inflation_factor

df = new2[['annual_inc','loan_amnt', 'funded_amnt','funded_amnt_inv
                                    ']]
#%%capture
#gather features
features = "+".join(df.columns[1:])

# get y and X dataframes based on this regression:
y, X = dmatrices('annual_inc ~' + features, df, return_type='
                                    dataframe')

# For each X, calculate VIF and save in dataframe
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(X.values, i) for i
                                    in range(X.shape[1])]
vif["features"] = X.columns

vif.round(1)

## drop
# ["loan_amnt", "funded_amnt", "funded_amnt_inv"] these 3
                                    attributes are similar, just
                                    choose 1.
new3 = new3.drop(["funded_amnt","funded_amnt_inv"],axis=1)
```

```
# term
fig = plt.figure(figsize=(10,5.5))
ax = fig.add_subplot(111)
sns.countplot(new3["term"])
ax.set_title("Countplot -- term",size=20)
ax.set_ylabel("count",size = 15)
ax.set_xlabel("term",size = 15)
plt.show()
plt.close(fig)

### avoid including future information
'''
1.Interes rate is calculated based on the default probability and
                                  credit rank. Thus, we cannot get
                                  this information before we
                                  evaluate the credit risk. Thus,
                                  delete these two features.
2.Installment: The monthly payment owed by the borrower if the loan
                                   originates. This feature is
                                  determined by interest_rate and
                                  loan_amount. Thus ignore these
                                  attribute.
3.Grade and sub_grade are also calculated after the default
                                  probability. Those information
                                  are also future information.
                                  Delete these two features.
4.revol_bal,Total credit revolving balance!
5.revol_util, Revolving line utilization rate, or the amount of
                                  credit the borrower is using
                                  relative to all available
                                  revolving credit.
6.total_pymnt,Payments received to date for total amount funded!
7.total_pymnt_inv,Payments received to date for portion of total
                                  amount funded by investors!
8.total_rec_prncp
9.total_rec_int
10.total_rec_late_fee
11.recoveries
12.collection_recovery_fee
13.last_pymnt_amnt
14.pub_rec_bankruptcies
15.'inq_last_6mths'
'''
#new3["int_rate"] = (new3["int_rate"]-new3["int_rate"].min())/(new3
                                  ["int_rate"].max()-new3["int_rate
                                  "].min())
#print(new3["int_rate"].describe())
#print(new3["installment"].describe())
#print(new3["grade"].describe())
#print(new3["sub_grade"].describe())
new3 = new3.drop(["int_rate", "installment","grade","sub_grade","
                                  revol_util","revol_bal",
                 "total_pymnt","total_pymnt_inv","total_rec_prncp",
                                                  "total_rec_int",
                                                  "
                                                  total_rec_late_fee
```

```
                                                    ",
                 "collection_recovery_fee","recoveries","
                                                    last_pymnt_amnt"
                                                    ,"
                                                    pub_rec_bankruptcies
                                                    "],axis=1)
new3 = new3.drop(["inq_last_6mths","pub_rec"],axis=1)


### emp_length ( fill the NA value with the method of "forward fill
                                  ")
# Employment length in years. Possible values are between 0 and 10
                                  where 0 means less than one year
                                  and 10 means ten or more years.
The missing value is less than 5% of the sample, so we propagate
                                  last valid observation forward to
                                   next

temp = new3["emp_length"].copy()
temp[temp.isnull()] ="NA"
temp[temp!="NA"] ="Non-NA"
sns.countplot(temp)
new3["emp_length"]=new3["emp_length"].fillna(method ='ffill')

# visualization
fig = plt.figure(figsize=(15,8))
ax = fig.add_subplot(111)

sns.countplot(new3["emp_length"])

ax.set_title("Countplot-- emp_length",size=20)
ax.set_ylabel("count",size = 15)
ax.set_xlabel("emp_length",size = 15)

plt.show()
plt.close(fig)

# home_ownership
# The home ownership status provided by the borrower during
                                  registration or obtained from the
                                   credit report. Our values are:
                                  RENT, OWN, MORTGAGE, OTHER

print(new3["home_ownership"].describe())
fig = plt.figure(figsize=(10,5.5))
ax = fig.add_subplot(111)
sns.countplot(new3["home_ownership"])
ax.set_title("Countplot -- home_ownership",size=20)
ax.set_ylabel("count",size = 15)
ax.set_xlabel("home_ownership",size = 15)
plt.show()
plt.close(fig)

# annual_inc
print(new3["annual_inc"].describe())
sns.distplot(new3["annual_inc"],rug=True,
                     hist_kws={'color':'yellow','label':'hist'},
                    kde_kws={'color':'red','label':'KDE'})

## verification_status
## Indicates if income was verified by LC, not verified, or if the
                                  income source was verified

print(new3["verification_status"].describe())
fig = plt.figure(figsize=(10,5.5))
ax = fig.add_subplot(111)
sns.countplot(new3["verification_status"])
ax.set_title("Countplot -- verification_status",size=20)
ax.set_ylabel("count",size = 15)
ax.set_xlabel("verification_status",size = 15)
plt.show()
plt.close(fig)

# delete all the date attribute: ["issue_d","earliest_cr_line","
                                  last_pymnt_d","last_credit_pull_d
                                  "]
# Date are not useful in the classification
new3 = new3.drop(["issue_d","earliest_cr_line","last_pymnt_d","
                                  last_credit_pull_d"],axis=1)

# purpose
# A category provided by the borrower for the loan request.
print(new3["purpose"].describe())
fig = plt.figure(figsize=(15,8))
ax = fig.add_subplot(111)
sns.countplot(new3["purpose"])
ax.set_title("Countplot -- purpose",size=20)
ax.set_ylabel("count",size = 15)
ax.set_xlabel("purpose",size = 15)
plt.show()
plt.close(fig)

# title
"""
The loan title provided by the borrower.There is no standardized
                                  label. Duplicated labels exist,
                                  like:
"Debt Consolidation","Debt Consolidation Loan","Debt consolidation
                                  ","Consolidation", "consolidation
                                  "
"Personal", "personal", "personal loan","loan"
"""

# count the frequency of "title"
new3.loc[:,["title"]].apply(pd.value_counts)
new3 = new3.drop(["title",],axis=1)

# > 90% similarity
#new3.loc[:,["pub_rec"]].apply(pd.value_counts)
#new3 = new3.drop(["pub_rec",],axis=1)

new3.shape
```

```
# zip_code & addr_state: those two attributes contain similar
                                  infomation, so just keep one

# visualization
temp = new3["zip_code"]

fig = plt.figure(figsize=(15,8))
ax = fig.add_subplot(111)
sns.countplot(temp)

ax.set_title("Countplot--zip_code",size=20)
ax.set_ylabel("count",size = 15)
ax.set_xlabel("zip_code",size = 15)

plt.show()
plt.close(fig)
print(new3["zip_code"].describe())

# visualization
temp = new3["addr_state"]

fig = plt.figure(figsize=(15,8))
ax = fig.add_subplot(111)
sns.countplot(temp)

ax.set_title("Countplot-- addr_state",size=20)
ax.set_ylabel("count",size = 15)
ax.set_xlabel("addr_state",size = 15)

plt.show()
plt.close(fig)

# drop zip_code
new3 = new3.drop(["zip_code",],axis=1)

### dti
# A ratio calculated using the borrowers total monthly debt
                                  payments on the total debt
                                  obligations, excluding mortgage
                                  and the requested LC loan,
                                  divided by the borrowers self-
                                  reported monthly income.

print(new3["dti"].describe())
sns.distplot(new3["dti"],rug=True,
                     hist_kws={'color':'yellow','label':'hist'},
                     kde_kws={'color':'red','label':'KDE'})

## total_acc & open_acc (only keep one)
# The total number of credit lines currently in the borrower's
                                  credit file

new3["total_acc"].describe()
new3["open_acc"].describe()

# visualization
fig,axes=plt.subplots(1,2,figsize=(12,5))
```

```
sns.distplot(new3["open_acc"],rug=True,
                     hist_kws={'color':'green','label':'hist'},
                     kde_kws={'color':'red','label':'KDE'},
                     ax=axes[0])
sns.distplot(new3["total_acc"],rug=True,
                     hist_kws={'color':'yellow','label':'hist'},
                     kde_kws={'color':'red','label':'KDE'},
                     ax=axes[1])

new3 = new3.drop(["total_acc"],axis=1)

# 'delinq_2yrs'

print(new3['delinq_2yrs'].describe())
sns.distplot(new3["delinq_2yrs"],rug=True,
                     hist_kws={'color':'yellow','label':'hist'},
                     kde_kws={'color':'red','label':'KDE'})

# missing_amnt

print(new3['missing_amnt'].describe())
sns.distplot(new3["missing_amnt"],rug=True,
                     hist_kws={'color':'yellow','label':'hist'},
                     kde_kws={'color':'red','label':'KDE'})

print(new3.shape)
freeze_header(new3)
new3.to_csv(path+"data.csv",index=False)

new3.columns

'''
da = pd.read_csv(path+"data.csv")
da = da.iloc[:,[1,5,7,10]]
da["loan_amnt"] = (da["loan_amnt"]-da['loan_amnt'].min())/(da['
                                  loan_amnt'].max()-da["loan_amnt
                                  "].min())
da["annual_inc"] = (da["annual_inc"]-da['annual_inc'].min())/(da['
                                  annual_inc'].max()-da["annual_inc
                                  "].min())
da["dti"] = (da["dti"]-da['dti'].min())/(da['dti'].max()-da["dti"].
                                  min())
sns.pairplot(da, hue="loan_status",markers=["o", "s"])
da.head()
'''

### Scaling ####################################
# Normalization of numerical attributes
new3=pd.read_csv(path+"data.csv")
# normalization
new3["loan_amnt"] = (new3["loan_amnt"] - new3["loan_amnt"].min())/(
                                  new3["loan_amnt"].max()-new3["
                                  loan_amnt"].min())
new3["annual_inc"] = (new3["annual_inc"] - new3["annual_inc"].min()
                                  )/(new3["annual_inc"].max()-new3[
                                  "annual_inc"].min())
```

```
new3["dti"] = (new3["dti"] - new3["dti"].min())/(new3["dti"].max()-
                                  new3["dti"].min())
new3["open_acc"] = (new3["open_acc"] - new3["open_acc"].min())/(
                                  new3["open_acc"].max()-new3["
                                  open_acc"].min())
new3["delinq_2yrs"] = (new3["delinq_2yrs"] - new3["delinq_2yrs"].
                                  min())/(new3["delinq_2yrs"].max()
                                  -new3["delinq_2yrs"].min())
new3["missing_amnt"] = (new3["missing_amnt"] - new3["missing_amnt"]
                                  .min())/(new3["missing_amnt"].max
                                  ()-new3["missing_amnt"].min())

freeze_header(new3)
new3.shape

# convert categorical variable into dummy variable
# get_dummies
d1=pd.get_dummies(new3["term"])
d2=pd.get_dummies(new3["emp_length"])
d3=pd.get_dummies(new3["home_ownership"])
d4=pd.get_dummies(new3["verification_status"])
d5=pd.get_dummies(new3["purpose"])
d6=pd.get_dummies(new3["addr_state"])
new3.drop(['term', 'emp_length', 'home_ownership','
                                  verification_status','purpose', '
                                  addr_state'],axis=1,inplace =
                                  True)
new3 = pd.concat([new3,d1,d2,d3,d4,d5,d6],axis=1)
new3.to_csv(path+"final_clean_set.csv",index = False)

# alternative algo: labelencoder

'''
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()
for col in data.columns:
    data[col] = labelencoder.fit_transform(data[col])

data.head()

y = data['class']    #
X = data.drop('class', axis = 1)
'''
# alternative algo: one hot encoder
                                          tree         model
# https://www.cnblogs.com/king-lps/p/7846414.html

new4= pd.read_csv(path+"final_clean_set.csv")
freeze_header(new4)


### 3.Split sample set into training set and test set -- 9:1
                                  #######

from sklearn.model_selection import train_test_split
Y = new4.loc[:,"loan_status"]
X = new4.drop(["loan_status"],axis=1)
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size
                                 = 0.1, random_state = 190003956)

L=["train"]*X_train.shape[0]+["test"]*X_test.shape[0]
sns.countplot(L)


### 4. Model selection and performance
# Decision Tree

from sklearn.tree import DecisionTreeClassifier
# Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# gridesearchCV
from  sklearn.model_selection import GridSearchCV

from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score

'''
https://blog.csdn.net/evolution23/article/details/85235397
criterion: "gini" --- CART, "entropy" --- ID3, C4.5
splitter: "best" --- small dataset, "random" --- large dataset
max_depth(            )
max_leaf_nodes(            )
max_features(              )
min_samples_leaf(                              )
min_sample_split(                              )
min_weight_fraction_leaf(

                                 )
min_impurity_split(                  )
'''

# Create Decision Tree classifer object
Tree = DecisionTreeClassifier()

Max_depth = range(5,15,1)
Min_sample_leaf = range(1,10,2)
tuned_paramters = dict(max_depth = Max_depth, min_samples_leaf =
                                 Min_sample_leaf)

# Train Decision Tree Classifer
DD = GridSearchCV(Tree, tuned_paramters, cv = 10)
clf = DD.fit(X_train,y_train)

print("Best: %f using %s" % (DD.best_score_, DD.best_params_))

y_prob = DD.predict_proba(X_test)[:,1] # This will give you
                                 positive class prediction
                                 probabilities
y_pred = np.where(y_prob > 0.5, 1, 0) # This will threshold the
                                 probabilities to give class
                                 predictions.
#DD.score(X_test, y_pred)
```

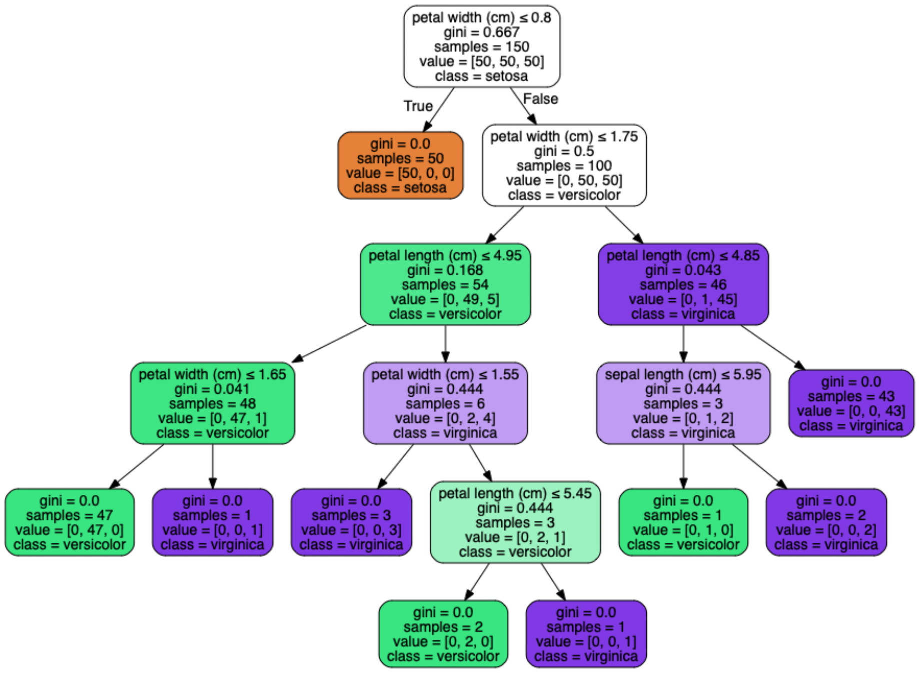
```
print('The AUC of GridSearchCV Desicion Tree is', roc_auc_score(
                                  y_test,y_pred))

#DD.grid_scores_

test_means = DD.cv_results_[ 'mean_test_score' ]
#test_stds = DD.cv_results_[ 'std_test_score' ]
#pd.DataFrame(DD.cv_results_).to_csv('DD_min_samples_leaf_maxdepth.
                                  csv')

# plot results
test_scores = np.array(test_means).reshape(len(Max_depth), len(
                                  Min_sample_leaf))

fig = plt.figure(figsize=(10,8))
for i, value in enumerate(Max_depth):
    plt.plot(Min_sample_leaf, test_scores[i], label= '
                                      test_max_depth:'   + str(
                                      value))

plt.legend(loc='upper center', bbox_to_anchor=(1.2, 1), shadow=True
                                  , ncol=1)
plt.xlabel( 'min_samples_leaf' )
plt.ylabel( 'accuray' )
plt.show()

# Create Decision Tree classifer object
Tree_res = DecisionTreeClassifier(criterion="entropy")

# Train Decision Tree Classifer
Tree_res = Tree_res.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = Tree_res.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

from sklearn.metrics import confusion_matrix
conf_matrix=confusion_matrix(y_test, y_pred, labels=["Fully Paid","
                                  Charged Off"])

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print(conf_matrix)
fig = plt.figure(figsize=(10,5.5))
ax = fig.add_subplot(111)

sns.heatmap(conf_matrix, annot=True)
ax.set_title("Confusion Matrix",size=20)


plt.show()
plt.close(fig)

from sklearn.tree import DecisionTreeClassifier

model_tree = DecisionTreeClassifier()
```

```
model_tree.fit(X_train, y_train)
y_prob = model_tree.predict_proba(X_test)[:,1] # This will give you
                                   positive class prediction
                                  probabilities
y_pred = np.where(y_prob > 0.5, 1, 0) # This will threshold the
                                  probabilities to give class
                                  predictions.
model_tree.score(X_test, y_pred)
print('The AUC of default Desicion Tree is',roc_auc_score(y_test,
                                  y_pred))

#
df = pd.DataFrame({"columns":list(columns), "importance":list(
                                  model_tree.feature_importances_.T
                                  )})
df.sort_values(by=['importance'],ascending=False)

plt.bar(range(len(model_tree.feature_importances_)), model_tree.
                                  feature_importances_)

y_pred.head()

# visualizaiton of decision tree (            )
"""
from sklearn import tree
#After  t r a i n i n g use  export_graphviz to export the Tree into
                                  Graphviz format
with open("iris.dot", 'w') as f:
    f = tree.export_graphviz(clf, out_file=f)

import pydotplus
dot_data=tree.export_graphviz(clf,out_file=None)
graph=pydotplus.graph_from_dot_data(dot_data)
graph.write_pdf('iris_2.pdf')


Feature = new4.columns
target = ["Charged Off","Fully Paid"]
from IPython.display import Image
dot_data = tree.export_graphviz(clf)
                                #out_file=None,
                         #feature_names=new4.columns,
                         #class_names=target,
                         #filled=True, rounded=True,
                         #special_characters = True)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
"""


# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
```

```
y_pred = clf.predict(X_test)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus

#dot_data = StringIO()
#export_graphviz(Tree, out_file=dot_data,  filled=True, rounded=
                                  True, special_characters=True,
                                  feature_names = list(X_train.
                                  columns),class_names=['Charged
                                  Off','Fully Paid'])
#graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
#graph.write_png(path+'diabetes.png')
#Image(graph.create_png())

dot_data = StringIO()
export_graphviz(Tree1, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())

### using iris dataset to plot the decision tree
from sklearn import tree
from sklearn.datasets import load_iris

#                      sklearnIris

iris=load_iris()
clf=tree.DecisionTreeClassifier()
clf=clf.fit(iris.data,iris.target)
from sklearn import tree
#After  training use  export_graphviz to export the Tree into
                                  Graphviz format
with open("iris.dot", 'w') as f:
    f = tree.export_graphviz(clf, out_file=f)

import pydotplus
dot_data=tree.export_graphviz(clf,out_file=None)
graph=pydotplus.graph_from_dot_data(dot_data)
graph.write_pdf('iris_2.pdf')

import pydotplus
dot_data=tree.export_graphviz(clf,out_file=None)
graph=pydotplus.graph_from_dot_data(dot_data)
graph.write_pdf('iris_2.pdf')
from IPython.display import Image
dot_data = tree.export_graphviz(clf, out_file=None,
                         feature_names=iris.feature_names,
                         class_names=iris.target_names,
                         filled=True, rounded=True,
                         special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data)
```

```
Image(graph.create_png())

dot_data = tree.export_graphviz(clf, out_file = None,filled=True)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

Sample of Decision Tree



```
#Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=300)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)

#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
name = X.columns
RandomForestClassifier(bootstrap=True, class_weight=None, criterion
                                   ='gini',
            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=100, n_jobs=
                                                  1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)


import pandas as pd
feature_imp = pd.Series(clf.feature_importances_,index=name).
                                  sort_values(ascending=False)
feature_imp
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
# Creating a bar plot
fig = plt.figure(figsize=(15,20))
sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to your graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.legend()
plt.show()

from sklearn.metrics import confusion_matrix
conf_matrix=confusion_matrix(y_test, y_pred, labels=["Fully Paid","
                                  Charged Off"])

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print(conf_matrix)
fig = plt.figure(figsize=(10,5.5))
ax = fig.add_subplot(111)

sns.heatmap(conf_matrix, annot=True)
ax.set_title("Confusion Matrix",size=20)


plt.show()
plt.close(fig)

from sklearn.ensemble import RandomForestClassifier as rfc
from sklearn.metrics import accuracy_score
for n_estimator in range(1, 300, 15):
    rf = rfc(n_estimators=n_estimator)
    rf.fit(train_x, train_y)
    predicted_rf = rf.predict(test_x)
    acc_rf = (accuracy_score(test_y, predicted_rf))
    print ("Accuracy: with estimator =\t ", n_estimator, "\t",
                                      acc_rf)


# KNN
from sklearn.neighbors import KNeighborsClassifier as knc
from sklearn.metrics import accuracy_score
for k in range(1,10):
    knn= knc(n_neighbors=k, weights="uniform")
    knn.fit(train_x, train_y)
    predicted_knn = knn.predict(test_x)
    acc_knn = (accuracy_score(test_y, predicted_knn))
    print ("Accuracy: with k =\t ", k, "\t", acc_knn)

from sklearn.neighbors import KNeighborsClassifier as knc
from sklearn.metrics import accuracy_score
for k in range(1,10):
    for p in range(1,5):
        knn=knc(n_neighbors=k, weights="distance", p=p)
        knn.fit(train_x, train_y)
        predicted_knn = knn.predict(test_x)
        acc_knn = (accuracy_score(test_y, predicted_knn))
        print ("Accuracy: with k = ", k, "  and p value in
                                          Minkowski distance = ", p
                                          , "\t", acc_knn)

## SVM

from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

svm_linear = SVC(C=1.0, kernel="linear", gamma="auto" )
svm_poly = SVC(C=1.0, kernel="poly", degree=3, gamma="auto")
svm_rbf = SVC(C=1.0, kernel="rbf", gamma=0.5)

svms = [svm_linear, svm_poly, svm_rbf]

for svm, i in zip(svms, range(len(svms))):
    svm.fit(train_x, train_y)
    predicted_svm = svm.predict(test_x)
    acc_svm = (accuracy_score(test_y, predicted_svm))
    print ("Accuracy: \t", acc_svm)

# LDA
from sklearn.discriminant_analysis import
                                    LinearDiscriminantAnalysis as lda
from sklearn.metrics import accuracy_score
ld = lda()
ld.fit(train_x, train_y)
predicted_ld = ld.predict(test_x)
acc_ld = (accuracy_score(test_y, predicted_ld))
acc_ld



#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  9 20:39:33 2019

@author: peng_kaiwen
"""

#importing libraries
import numpy as np
import pandas as pd
from matplotlib import cm

#loading the dataset
```

```
dataset = pd.read_csv('dataset-4.csv')
#X = dataset.iloc[:,0:6].values
X = dataset.iloc[:,0:3].values
y = dataset.iloc[:,len(dataset.iloc[0])-1].values

#train/test
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
                                  =0.25, random_state=1)

#fitting the classifier to the training set
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 100, criterion='
                                  entropy', random_state=0)
classifier.fit(X_train, y_train)


import matplotlib.pyplot as plt
from scipy import integrate
def capcurve(y_values, y_preds_proba):
    num_pos_obs = np.sum(y_values)
    num_count = len(y_values)
    rate_pos_obs = float(num_pos_obs) / float(num_count)
    ideal = pd.DataFrame({'x':[0,rate_pos_obs,1],'y':[0,1,1]})
    xx = np.arange(num_count) / float(num_count - 1)

    y_cap = np.c_[y_values,y_preds_proba]
    y_cap_df_s = pd.DataFrame(data=y_cap)
    y_cap_df_s = y_cap_df_s.sort_values([1], ascending=False).
                                      reset_index('index', drop=
                                      True)

    print(y_cap_df_s.head(20))

    yy = np.cumsum(y_cap_df_s[0]) / float(num_pos_obs)
    yy = np.append([0], yy[0:num_count-1]) #add the first curve
                                      point (0,0) : for xx=0 we
                                      have yy=0

    percent = 0.5
    row_index = np.trunc(num_count * percent)

    val_y1 = yy[row_index]
    val_y2 = yy[row_index+1]
    if val_y1 == val_y2:
        val = val_y1*1.0
    else:
        val_x1 = xx[row_index]
        val_x2 = xx[row_index+1]
        val = val_y1 + ((val_x2 - percent)/(val_x2 - val_x1))*(
                                          val_y2 - val_y1)

    sigma_ideal = 1 * xx[num_pos_obs - 1 ] / 2 + (xx[num_count - 1]
                                       - xx[num_pos_obs]) * 1
    sigma_model = integrate.simps(yy,xx)
    sigma_random = integrate.simps(xx,xx)
    ar_value = (sigma_model - sigma_random) / (sigma_ideal -
                                        sigma_random)
    #ar_label = 'ar value = %s' % ar_value

    fig, ax = plt.subplots(nrows = 1, ncols = 1)
    ax.plot(ideal['x'],ideal['y'], color='grey', label='Perfect
                                        Model')
    ax.plot(xx,yy, color='red', label='User Model')
    #ax.scatter(xx,yy, color='red')
    ax.plot(xx,xx, color='blue', label='Random Model')
    ax.plot([percent, percent], [0.0, val], color='green',
                                        linestyle='--', linewidth=1)
    ax.plot([0, percent], [val, val], color='green', linestyle='--'
                                        , linewidth=1, label=str(val*
                                        100)+'% of positive obs at '+
                                        str(percent*100)+'%')

    plt.xlim(0, 1.02)
    plt.ylim(0, 1.25)
    plt.title("CAP Curve - a_r value ="+str(ar_value))
    plt.xlabel('% of the data')
    plt.ylabel('% of positive obs')
    plt.legend()
    plt.show()


y_pred_proba = classifier.predict_proba(X=X_test)
capcurve(y_values=y_test, y_preds_proba=y_pred_proba[:,1])
```