Final Task Data Science VIX Rakamin with ID/X

Muhammad Hazim M.
https://www.linkedin.com/in/hazim-m/
Full code on:
https://github.com/hazim17/Final_project_VIX_rakamin_idx

Credit / Loan Risk Prediction

Table of content:

Context

A credit score can help the bank to determine a consumer's creditworthiness. The higher the score, the better a borrower looks to potential lenders. A credit score is based on credit history.

Lenders use credit scores to evaluate the probability that an individual will repay loans in a timely manner.

People with credit scores lower, for example, are generally considered to be subprime borrowers. Lending institutions often charge interest on subprime mortgages at a rate higher than a conventional mortgage in order to compensate themselves for carrying more risk.

Objective:

For this project, we won't calculate the credit score, rather we want to classify the customer that is 'good' or 'bad' consumer creditworthiness.

So, using the dataset given dataset, IDX partners want us to create a model to predict loan credit risk (To classify Is their loaner/customer will be a good or bad loaner).

Then, we need to create a visual presentation to present the result to the client.

Data Understanding: Look at the dataset

```
[ ]  # Info of the data
     df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 466285 entries, 0 to 466284
Data columns (total 75 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   Unnamed: 0                  466285 non-null  int64
 1   id                          466285 non-null  int64
 2   member_id                   466285 non-null  int64
 3   loan_amnt                   466285 non-null  int64
 4   funded_amnt                 466285 non-null  int64
 5   funded_amnt_inv             466285 non-null  float64
 6   term                        466285 non-null  object
 7   int_rate                    466285 non-null  float64
 8   installment                 466285 non-null  float64
 9   grade                       466285 non-null  object
 10  sub_grade                   466285 non-null  object
 11  emp_title                   438697 non-null  object
 12  emp_length                  445277 non-null  object
 13  home_ownership              466285 non-null  object
 14  annual_inc                  466281 non-null  float64
 15  verification_status         466285 non-null  object
 16  issue_d                     466285 non-null  object
 17  loan_status                 466285 non-null  object
 18  pymnt_plan                  466285 non-null  object
 19  url                         466285 non-null  object
 20  desc                        125983 non-null  object
 21  purpose                     466285 non-null  object
 22  title                       466265 non-null  object
 23  zip_code                    466285 non-null  object
 24  addr_state                  466285 non-null  object
 25  dti                         466285 non-null  float64
 26  delinq_2yrs                 466256 non-null  float64
 27  earliest_cr_line            466256 non-null  object
 28  inq_last_6mths              466256 non-null  float64
 29  mths_since_last_delinq      215934 non-null  float64
 30  mths_since_last_record      62638 non-null   float64
 31  open_acc                    466256 non-null  float64
 32  pub_rec                     466256 non-null  float64
 33  revol_bal                   466285 non-null  int64
 34  revol_util                  465945 non-null  float64
 35  total_acc                   466256 non-null  float64
 36  initial_list_status         466285 non-null  object
 37  out_prncp                   466285 non-null  float64
 38  out_prncp_inv               466285 non-null  float64
 39  total_pymnt                 466285 non-null  float64
 40  total_pymnt_inv             466285 non-null  float64
 41  total_rec_prncp             466285 non-null  float64
 42  total_rec_int               466285 non-null  float64
 43  total_rec_late_fee          466285 non-null  float64
 44  recoveries                  466285 non-null  float64
 45  collection_recovery_fee     466285 non-null  float64
 46  last_pymnt_d                465909 non-null  object
 47  last_pymnt_amnt             466285 non-null  float64
 48  next_pymnt_d                239071 non-null  object
 49  last_credit_pull_d          466243 non-null  object
 50  collections_12_mths_ex_med  466140 non-null  float64
 51  mths_since_last_major_derog 98974 non-null   float64
 52  policy_code                 466285 non-null  int64
 53  application_type            466285 non-null  object
 54  annual_inc_joint            0 non-null       float64
 55  dti_joint                   0 non-null       float64
 56  verification_status_joint   0 non-null       float64
 57  acc_now_delinq              466256 non-null  float64
 58  tot_coll_amt                396009 non-null  float64
 59  tot_cur_bal                 396009 non-null  float64
 60  open_acc_6m                 0 non-null       float64
 61  open_il_6m                  0 non-null       float64
 62  open_il_12m                 0 non-null       float64
 63  open_il_24m                 0 non-null       float64
 64  mths_since_rcnt_il          0 non-null       float64
 65  total_bal_il                0 non-null       float64
 66  il_util                     0 non-null       float64
 67  open_rv_12m                 0 non-null       float64
 68  open_rv_24m                 0 non-null       float64
 69  max_bal_bc                  0 non-null       float64
 70  all_util                    0 non-null       float64
 71  total_rev_hi_lim            396009 non-null  float64
 72  inq_fi                      0 non-null       float64
 73  total_cu_tl                 0 non-null       float64
 74  inq_last_12m                0 non-null       float64
dtypes: float64(46), int64(7), object(22)
memory usage: 266.8+ MB
```

There are 75 columns with 466285 (around 400k) rows.

Some of the rows has null values.

Data Understanding: Feature and Target

Feature and Target Variables:

- Target Variable: 'loan_status'
- Feature Variable: All variables except the target variable

Explanation of Target Variable:

- **'Fully Paid':** fully paid means the payment in cash or cash equivalents in full of all obligations (other than indemnity obligations that survive payment in full) under the Credit Facility Loan Documents.
- **'Charged Off':** A charge-off does not mean you no longer have to repay a debt. A creditor can still come after full payment. A charge-off occurs when an account is seriously delinquent — for credit cards, that's after 180 days of not making the minimum payment.
- **'Current':** Current Loan means, as of any date, a Loan as to which all payments of principal and interest then due have been paid, except for the most recent payment then due of principal and interest.
- **'Default':** Defaulting on a loan is the failure of a borrower to pay the principal or interest on a security or loan.
- **'Late (31-120 days)'**

Data Understanding: Feature and Target

- **'In Grace Period':** A grace period is a period of time creditors give borrowers to make their payments before incurring a late charge or risk defaulting on the loan.
- **'Late (16-30 days)'**
- **'Does not meet the credit policy. Status:Fully Paid'**; While the loan was paid off, the loan application today would no longer meet the credit policy and wouldn't be approved on to the marketplace.
- **'Does not meet the credit policy. Status:Charged Off'**; While the loan was charged off, the loan application today would no longer meet the credit policy and wouldn't be approved on to the marketplace.

Conclusion:

→ To make the classification easier

- **0: good loaner / low risk**
  - 'Fully Paid', 'Current'
- **1: Bad loaner / High risk**
  - 'Charged Off', 'Default', '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'

Data Understanding: Note For Data Cleaning

- **There are some column that don't have any data, the tables are:**
  - 'inq_last_12m', 'total_cu_tl', 'inq_fi', 'open_acc_6m', 'open_il_6m', 'open_il_12m', 'open_il_24m', 'mths_since_rcnt_il', 'total_bal_il', 'il_util', 'open_rv_12m', 'open_rv_24m', 'max_bal_bc', 'all_util', 'annual_inc_joint', 'dti_joint','verification_status_joint'
  - We should consider deleting all columns listed above because it completely null

- **At a glance, there is some column that is not important:**
  - 'Unnamed: 0': because it's like id in dataframe
  - 'zip_code', 'desc', 'url', 'title': Hard to interpret, too much unique value (as an object), and it's not suitable for modeling
  - 'application_type': It's only has one value, it's not gonna have 0 correlation with another variable
  - We should consider deleting all columns listed above because it's not important

Data Understanding: Note For Data Cleaning

- **Handle the columns that should have date as their data type:**
  - 'issue_d', 'next_pymnt_d', 'last_credit_pull_d', 'last_pymnt_d', 'earliest_cr_line'
  - All of those columns are not really helping in classifying good or bad loaners, consider dropping it.
  - Because of data leakage, we wouldn't know when our model will use to predict the loan risk.
  - In simple terms, Data Leakage occurs when the data used in the training process contains information about what the model is trying to predict. It appears like “cheating” but since we are not aware of it so, it is better to call it “leakage” instead of cheating.
- **We need to look at the outliers, treating outliers can reduce variance and increase the accuracy of the model**
- **We need to look at the data consistency, just in case some value is not consistence**
- **We need to look if there is any duplicate value.**

Data Cleaning:

Delete Columns with all Null values and not important columns

```
no_use_columns = [
    'inq_last_12m', 'total_cu_tl', 'inq_fi', 'open_acc_6m',
    'open_il_6m', 'open_il_12m',  'open_il_24m',  'mths_since_rcnt_il',
    'total_bal_il', 'il_util',  'open_rv_12m',  'open_rv_24m',  'max_bal_bc',
    'all_util', 'annual_inc_joint', 'dti_joint','verification_status_joint',
    'Unnamed: 0', 'zip_code', 'desc', 'url', 'title', 'application_type',
    'issue_d', 'next_pymnt_d',  'last_credit_pull_d',
    'last_pymnt_d', 'earliest_cr_line'
]

df.drop(no_use_columns, inplace=True, axis=1)
```

Data Cleaning:
Look at the unique value in each column

```
[ ]  # Function to look at each unique values in column
     def unique_value(x, y):
       column_res = []
       for i in range(x, y+1):
         if len(df.iloc[:, i].unique()) <= 30:
           print(df.columns[i])
           print(df.iloc[:, i].unique())
           print('----------------------')
         else:
           column_res.append(df.columns[i])
       print('Columns with too much unique value:', column_res)
```

```
[ ]  # column 0 - 13
     unique_value(0, 13)
```

```
term
[' 36 months' ' 60 months']
----------------------
grade
['B' 'C' 'A' 'E' 'F' 'D' 'G']
----------------------
emp_length
['10+ years' '< 1 year' '1 year' '3 years' '8 years' '9 years' '4 years'
 '5 years' '6 years' '2 years' '7 years' nan]
----------------------
home_ownership
['RENT' 'OWN' 'MORTGAGE' 'OTHER' 'NONE' 'ANY']
----------------------
Columns with too much unique value: ['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'int_rate', 'installment', 'sub_grade', 'emp_title', 'annual_inc']
```

Data Cleaning:

Look at the unique value in each column

```
[ ] df['sub_grade'].unique()
    # The meaning of loan subgrade
    # https://www.lendingclub.com/foliofn/rateDetail.action
```

```
array(['B2', 'C4', 'C5', 'C1', 'B5', 'A4', 'E1', 'F2', 'C3', 'B1', 'D1',
       'A1', 'B3', 'B4', 'C2', 'D2', 'A3', 'A5', 'D5', 'A2', 'E4', 'D3',
       'D4', 'F3', 'E3', 'F4', 'F1', 'E5', 'G4', 'E2', 'G3', 'G2', 'G1',
       'F5', 'G5'], dtype=object)
```

```
[ ] print(df['emp_title'].unique())
    print(df['emp_title'].nunique())
```

```
[nan 'Ryder' 'AIR RESOURCES BOARD' ... 'MecÃ¡nica'
 'Chief of Interpretation (Park Ranger)' 'Server Engineer Lead']
```

```
[ ] # column 14 - 26
    unique_value(14, 26)
```

```
verification_status
['Verified' 'Source Verified' 'Not Verified']
----------------------
loan_status
['Fully Paid' 'Charged Off' 'Current' 'Default' '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']
----------------------
pymnt_plan
['n' 'y']
----------------------
purpose
['credit_card' 'car' 'small_business' 'other' 'wedding'
 'debt_consolidation' 'home_improvement' 'major_purchase' 'medical'
 'moving' 'vacation' 'house' 'renewable_energy' 'educational']
```

```
----------------------
delinq_2yrs
[ 0.  2.  3.  1.  4.  6.  5.  8.  7.  9. 11. nan 13. 15. 10. 12. 17. 18.
 29. 24. 14. 21. 22. 19. 16.]
----------------------
inq_last_6mths
[ 1.  5.  2.  0.  3.  4.  6.  7.  8.  9. 10. 11. 12. 15. 14. 33. 17. 32.
 24. 13. 18. 16. 31. 28. 25. 27. 20. 19. nan]
----------------------
pub_rec
[ 0.  1.  2.  3.  4.  5. nan  6.  9.  8.  7. 11. 49. 10. 54. 12. 18. 19.
 16. 15. 14. 40. 63. 13. 21. 34. 17.]
----------------------
Columns with too much unique value: ['addr_state', 'dti', 'mths_since_last_delinq', 'mths_since_last_record', 'open_acc', 'revol_bal']
```

```
[ ] df['mths_since_last_delinq'].unique()
```

```
array([ nan,  35.,  38.,  61.,   8.,  20.,  18.,  68.,  45.,  48.,  41.,
        40.,  74.,  25.,  53.,  39.,  10.,  26.,  56.,  77.,  28.,  52.,
        24.,  16.,  60.,  54.,  23.,   9.,  11.,  13.,  65.,  19.,  80.,
        22.,  59.,  79.,  44.,  64.,  57.,  14.,  63.,  49.,  15.,  73.,
        70.,  29.,  51.,   5.,  75.,  55.,   2.,  30.,  47.,  33.,  69.,
         4.,  43.,  21.,  27.,  46.,  81.,  78.,  82.,  31.,  76.,  62.,
        72.,  42.,  50.,   3.,  12.,  67.,  36.,  34.,  58.,  17.,  71.,
        66.,  32.,   6.,  37.,   7.,   1.,  83.,  86., 115.,  96., 103.,
       120., 106.,  89., 107.,  85.,  97.,  95.,   0., 110.,  84., 135.,
        88.,  87., 122.,  91., 146., 134., 114.,  99.,  93., 127., 101.,
        94., 102., 129., 113., 139., 131., 156., 143., 109., 119., 149.,
       118., 130., 148., 126.,  90., 141., 116., 100., 152.,  98.,  92.,
       108., 133., 104., 111., 105., 170., 124., 136., 180., 188., 140.,
       151., 159., 121.])
```

```
[ ] df['open_acc'].unique()
```

```
array([ 3.,  2., 10., 15.,  9.,  7.,  4., 11., 14., 12., 20.,  8.,  6.,
       17.,  5., 13., 16., 30., 21., 18., 19., 27., 23., 34., 25., 22.,
       24., 26., 32., 28., 29., 33., 31., 39., 35., 36., 38., 44., 41.,
       42.,  1., 46., 37., 47., nan, 40., 45., 49., 53., 51., 43.,  0.,
       62., 48., 50., 52., 54., 76., 58., 55., 84., 75., 61.])
```

Data Cleaning:
Look at the unique value in each column

```
[ ]  # column 27 - 39
     unique_value(27, 39)

     initial_list_status
     ['f' 'w']
     ----------------------
     Columns with too much unique value: ['revol_util', 'total_acc', 'out_prncp', 'out_prncp_inv', 'total_pymnt', 'total_pymnt_inv', 'total_rec_prncp', 'total_rec_int', 'total_rec_late_fee', 'recoveries', 'collection_recovery_fee', 'l
```

```
[ ]  # column 40 - 46
     unique_value(40, 46)

     collections_12_mths_ex_med
     [ 0. nan  1.  2.  4.  3.  6. 16. 20.  5.]
     ----------------------
     policy_code
     [1]
     ----------------------
     acc_now_delinq
     [ 0.  1. nan  2.  3.  5.  4.]
     ----------------------
     Columns with too much unique value: ['mths_since_last_major_derog', 'tot_coll_amt', 'tot_cur_bal', 'total_rev_hi_lim']
```

Data Cleaning:
Note for deleting unimportant columns

After we look at unique in each columns, we can conclude that:

- We could delete grade, because we already have sub grade column, it'll be redundant
- Delete member_id and id, it doesn't related to anything for our modelling
- Delete emp_title, too much unique values, curse dimentionality

```
[ ]  no_use_columns2 = [
         'grade', 'id', 'member_id', 'emp_title'
     ]

     df.drop(no_use_columns2, inplace=True, axis=1)
```

Data Cleaning:
Missing Values

```
[ ] # look at the missing values
    df.isna().sum()
```

```
loan_amnt                       0
funded_amnt                     0
funded_amnt_inv                 0
term                            0
int_rate                        0
installment                     0
sub_grade                       0
emp_length                  21008
home_ownership                  0
annual_inc                      4
verification_status             0
loan_status                     0
pymnt_plan                      0
purpose                         0
addr_state                      0
dti                             0
delinq_2yrs                    29
inq_last_6mths                 29
mths_since_last_delinq     250351
mths_since_last_record     403647
open_acc                       29
pub_rec                        29
revol_bal                       0
revol_util                    340
total_acc                      29
initial_list_status             0
out_prncp                       0
out_prncp_inv                   0
total_pymnt                     0
total_pymnt_inv                 0
total_rec_prncp                 0
total_rec_int                   0
total_rec_late_fee              0
recoveries                      0
collection_recovery_fee         0
last_pymnt_amnt                 0
collections_12_mths_ex_med    145
mths_since_last_major_derog    367311
policy_code                     0
acc_now_delinq                 29
tot_coll_amt                70276
tot_cur_bal                 70276
total_rev_hi_lim            70276
dtype: int64
```

Note for cleaning the missing values:

- drop the column: 'mths_since_last_record', 'mths_since_last_delinq' and 'mths_since_last_major_derog. Too much missing value.
- Since we have so many rows, even if we drop some rows, the remaining rows still will be a lot useful. Drop null values in these columns: 'emp_length', 'annual_inc', 'delinq_2yrs', 'inq_last_6mths', 'open_acc', 'pub_rec', 'revol_util', 'total_acc', 'collections_12_mths_ex_med', 'acc_now_delinq', 'tot_coll_amt', 'tot_cur_bal', and 'total_rev_hi_lim'.

Data Cleaning:
Drop Missing Values

```
[ ]  # copy the dataset just in case we need to use the original
     df_copy = df.copy()
```

Drop column that has too much missing values

```
[ ]  df_copy.drop('mths_since_last_record',axis=1,inplace=True)
     df_copy.drop('mths_since_last_delinq',axis=1,inplace=True)
     df_copy.drop('mths_since_last_major_derog',axis=1,inplace=True)
```

Drop the row that has missing values

```
[ ]  df_clean = df_copy.dropna(axis=0)
```

Data Cleaning:
Check Duplicate Data

```
[ ]  duplicate = df_clean[df_clean.duplicated()]

     print("Duplicate Rows :")

     Duplicate Rows :
```

We don't have any duplicate value

Data Cleaning:
Check Data Type

```
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 376850 entries, 42535 to 466284
Data columns (total 40 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   loan_amnt                   376850 non-null  int64
 1   funded_amnt                 376850 non-null  int64
 2   funded_amnt_inv             376850 non-null  float64
 3   term                        376850 non-null  object
 4   int_rate                    376850 non-null  float64
 5   installment                 376850 non-null  float64
 6   sub_grade                   376850 non-null  object
 7   emp_length                  376850 non-null  object
 8   home_ownership              376850 non-null  object
 9   annual_inc                  376850 non-null  float64
 10  verification_status         376850 non-null  object
 11  loan_status                 376850 non-null  object
 12  pymnt_plan                  376850 non-null  object
 13  purpose                     376850 non-null  object
 14  addr_state                  376850 non-null  object
 15  dti                         376850 non-null  float64
 16  delinq_2yrs                 376850 non-null  float64
 17  inq_last_6mths              376850 non-null  float64
 18  open_acc                    376850 non-null  float64
 19  pub_rec                     376850 non-null  float64
 20  revol_bal                   376850 non-null  int64
 21  revol_util                  376850 non-null  float64
 22  total_acc                   376850 non-null  float64
 23  initial_list_status         376850 non-null  object
 24  out_prncp                   376850 non-null  float64
 25  out_prncp_inv               376850 non-null  float64
 26  total_pymnt                 376850 non-null  float64
 27  total_pymnt_inv             376850 non-null  float64
 28  total_rec_prncp             376850 non-null  float64
 29  total_rec_int               376850 non-null  float64
 30  total_rec_late_fee          376850 non-null  float64
 31  recoveries                  376850 non-null  float64
 32  collection_recovery_fee     376850 non-null  float64
 33  last_pymnt_amnt             376850 non-null  float64
 34  collections_12_mths_ex_med  376850 non-null  float64
 35  policy_code                 376850 non-null  int64
 36  acc_now_delinq              376850 non-null  float64
 37  tot_coll_amt                376850 non-null  float64
 38  tot_cur_bal                 376850 non-null  float64
 39  total_rev_hi_lim            376850 non-null  float64
dtypes: float64(26), int64(4), object(10)
memory usage: 117.9+ MB
```

Note for Data type:

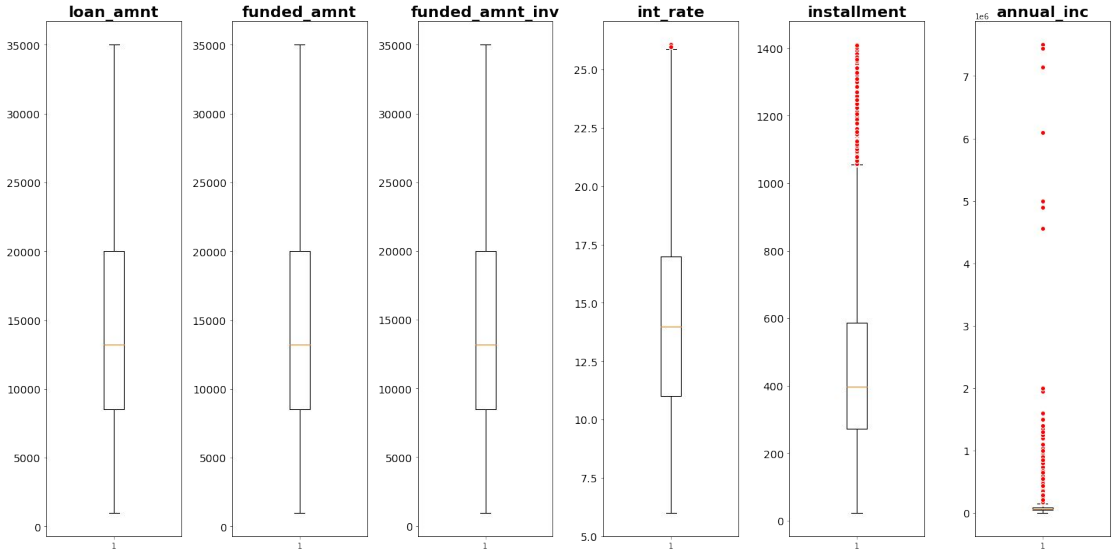
- The data type of each column seem right
- We could convert object type data to numeric using labelEncoder later

Data Cleaning:
Detecting Outliers

```
# we will create  boxplot for numerical column
# but we need to divide the column, so the graph still look clear
numcols1 = numcols[0:6]
numcols2 = numcols[6:12]
numcols3 = numcols[12:18]
numcols4 = numcols[18:24]
numcols5 = numcols[24:31]
print(numcols5)
print(len(numcols4))
```

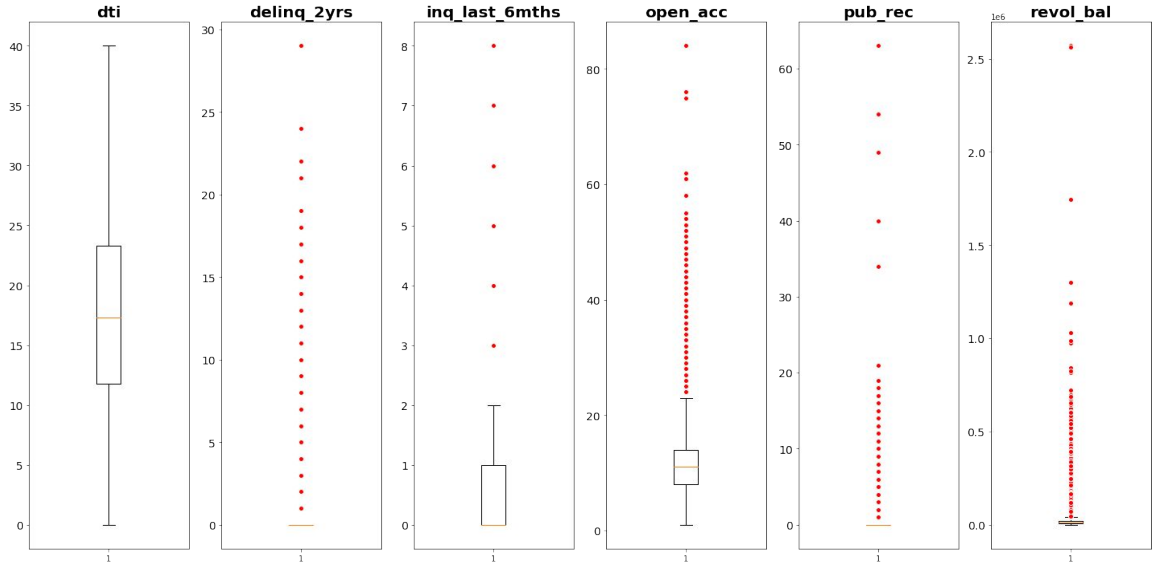
```
Index(['collections_12_mths_ex_med', 'policy_code', 'acc_now_delinq',
       'tot_coll_amt', 'tot_cur_bal', 'total_rev_hi_lim'],
      dtype='object')
6
```

```
[ ]  #Creating subplot of each column with its own scale
     try:
       red_circle = dict(markerfacecolor='red', marker='o', markeredgecolor='white')

       fig, axs = plt.subplots(1, len(numcols1), figsize=(20,10))

       for i, ax in enumerate(axs.flat):
           ax.boxplot(df[numcols1].iloc[:,i], flierprops=red_circle)
           ax.set_title(df[numcols1].columns[i], fontsize=20, fontweight='bold')
           ax.tick_params(axis='y', labelsize=14)

       plt.tight_layout()
     except:
       print("Error")
```



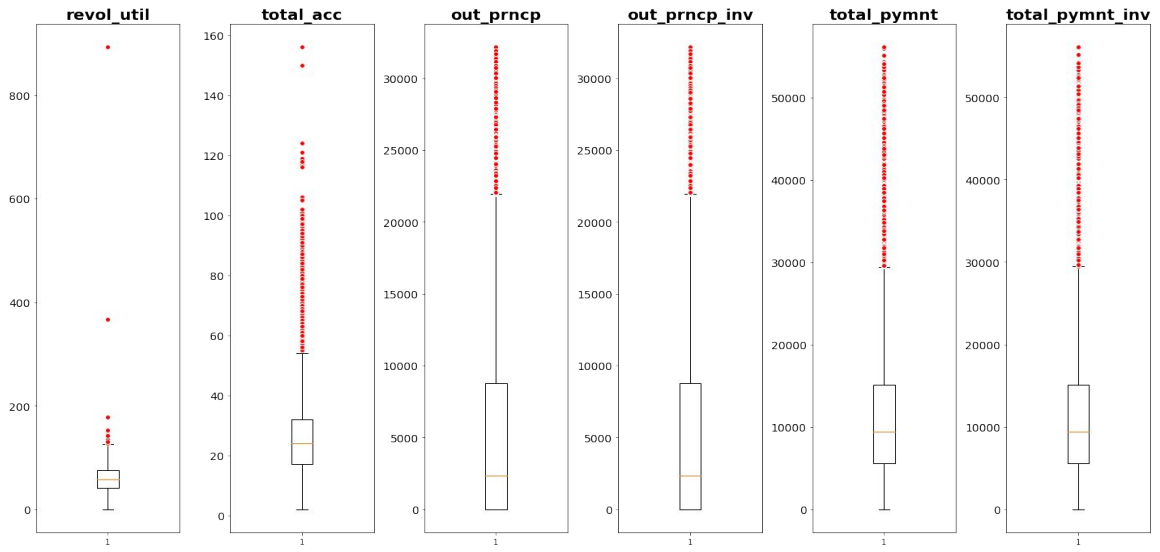
Data Cleaning:
Detecting Outliers

```
[ ]  #Creating subplot of each column with its own scale
     try:
       red_circle = dict(markerfacecolor='red', marker='o', markeredgecolor='white')

       fig, axs = plt.subplots(1, len(numcols2), figsize=(20,10))

       for i, ax in enumerate(axs.flat):
           ax.boxplot(df[numcols2].iloc[:,i], flierprops=red_circle)
           ax.set_title(df[numcols2].columns[i], fontsize=20, fontweight='bold')
           ax.tick_params(axis='y', labelsize=14)

       plt.tight_layout()
     except:
       print("Error")
```



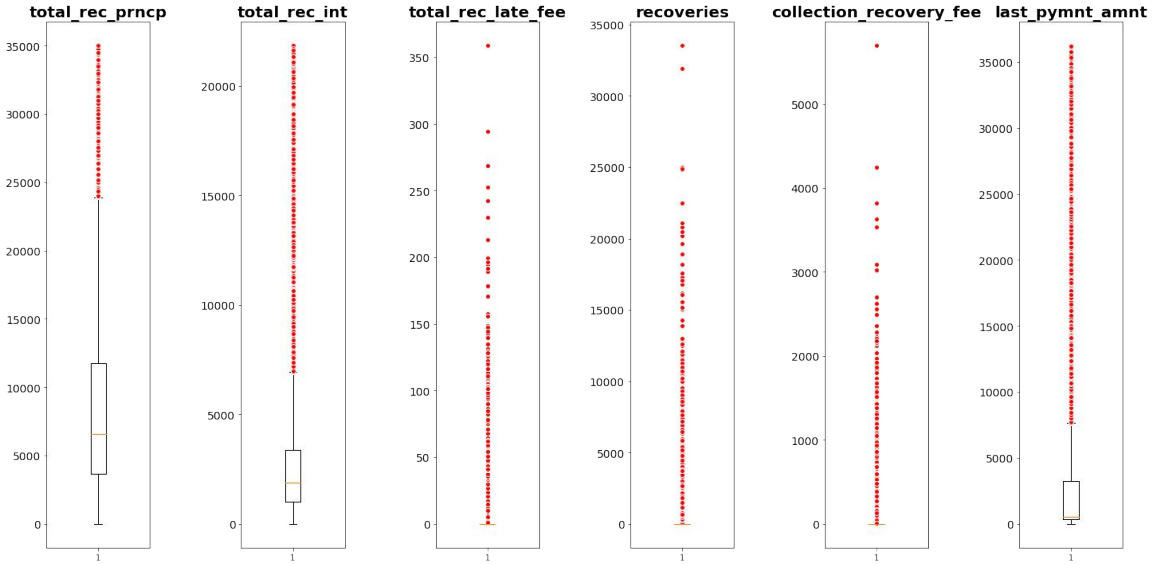
Data Cleaning:
Detecting Outliers

```
#Creating subplot of each column with its own scale
try:
  red_circle = dict(markerfacecolor='red', marker='o', markeredgecolor='white')

  fig, axs = plt.subplots(1, len(numcols3), figsize=(20,10))

  for i, ax in enumerate(axs.flat):
      ax.boxplot(df[numcols3].iloc[:,i], flierprops=red_circle)
      ax.set_title(df[numcols3].columns[i], fontsize=20, fontweight='bold')
      ax.tick_params(axis='y', labelsize=14)

  plt.tight_layout()
except:
  print("Error")
```



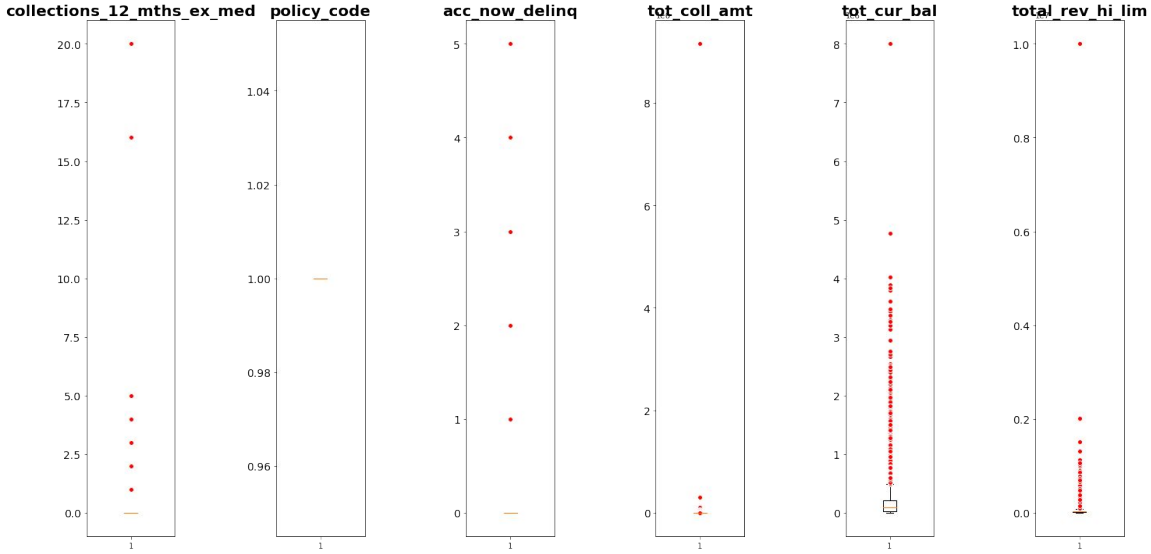
Data Cleaning:
Detecting Outliers

```
]  #Creating subplot of each column with its own scale
   try:
     red_circle = dict(markerfacecolor='red', marker='o', markeredgecolor='white')

     fig, axs = plt.subplots(1, len(numcols4), figsize=(20,10))

     for i, ax in enumerate(axs.flat):
         ax.boxplot(df[numcols4].iloc[:,i], flierprops=red_circle)
         ax.set_title(df[numcols4].columns[i], fontsize=20, fontweight='bold')
         ax.tick_params(axis='y', labelsize=14)

     plt.tight_layout()
   except:
     print("Error")
```



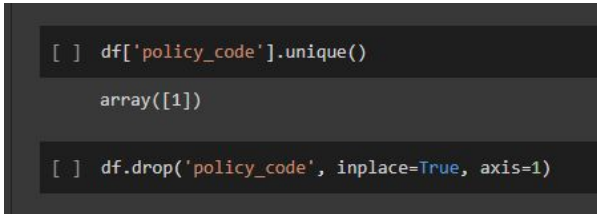
Data Cleaning:
Detecting Outliers

```
] #Creating subplot of each column with its own scale
  try:
    red_circle = dict(markerfacecolor='red', marker='o', markeredgecolor='white')

    fig, axs = plt.subplots(1, len(numcols5), figsize=(20,10))

    for i, ax in enumerate(axs.flat):
        ax.boxplot(df[numcols5].iloc[:,i], flierprops=red_circle)
        ax.set_title(df[numcols5].columns[i], fontsize=20, fontweight='bold')
        ax.tick_params(axis='y', labelsize=14)

    plt.tight_layout()
  except:
    print("Error")
```



Data Cleaning:
Note for handling Outlier

- Remove the policy_code column, it only has one value
- Some of the columns have outliers, but if we think again, it doesn't look like the wrong data.
  - Installment is The monthly payment owed by the borrower if the loan originates, some people could have higher installment
  - annual_inc also looks normal, some people could have a higher income than others
  - delinq_2_years is The number of 30+ days past-due incidences of delinquency in the borrower's credit file for the past 2 years. Some people have it, but most people don't have it. Since we want to classify which one is a good and bad loaner, keeping the outlier in this column wouldn't be a bad idea.
  - and that includes other column that has outliers



Data Preprocessing:
Change label in Status (For classification)

- **Handle the columns that should have date as their data type:**
  - 'issue_d', 'next_pymnt_d', 'last_credit_pull_d', 'last_pymnt_d', 'earliest_cr_line'
  - All of those columns are not really helping in classifying good or bad loaners, consider dropping it.
  - Because of data leakage, we wouldn't know when our model will use to predict the loan risk.
  - In simple terms, Data Leakage occurs when the data used in the training process contains information about what the model is trying to predict. It appears like “cheating” but since we are not aware of it so, it is better to call it “leakage” instead of cheating.
- **We need to look at the outliers, treating outliers can reduce variance and increase the accuracy of the model**
- **We need to look at the data consistency, just in case some value is not consistence**
- **We need to look if there is any duplicate value.**

Data Preprocessing:
Change label in Status (For classification)

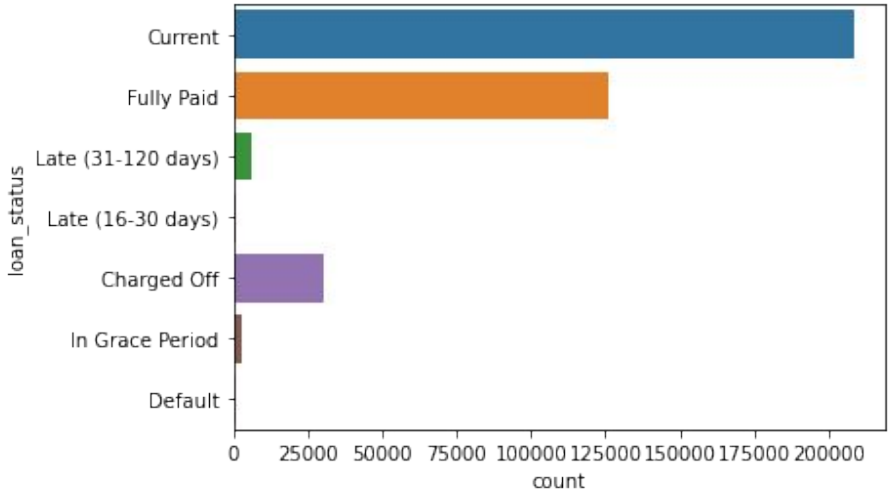
- **Handle the columns that should have date as their data type:**
  - 'issue_d', 'next_pymnt_d', 'last_credit_pull_d', 'last_pymnt_d', 'earliest_cr_line'
  - All of those columns are not really helping in classifying good or bad loaners, consider dropping it.
  - Because of data leakage, we wouldn't know when our model will use to predict the loan risk.
  - In simple terms, Data Leakage occurs when the data used in the training process contains information about what the model is trying to predict. It appears like “cheating” but since we are not aware of it so, it is better to call it “leakage” instead of cheating.
- **We need to look at the outliers, treating outliers can reduce variance and increase the accuracy of the model**
- **We need to look at the data consistency, just in case some value is not consistence**
- **We need to look if there is any duplicate value.**

Data Preprocessing:
Change label in Status (For classification)

Look at the distribution of loan_status

```
# See the loan_status Distribution
sns.countplot(y='loan_status', data=df)
plt.show()
```

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

Current                208951
Fully Paid             126115
Charged Off             30540
Late (31-120 days)       6376
In Grace Period          2978
Late (16-30 days)        1123
Default                   767
Name: loan_status, dtype: int64
```



Data Preprocessing:
Change label in Status (For classification)

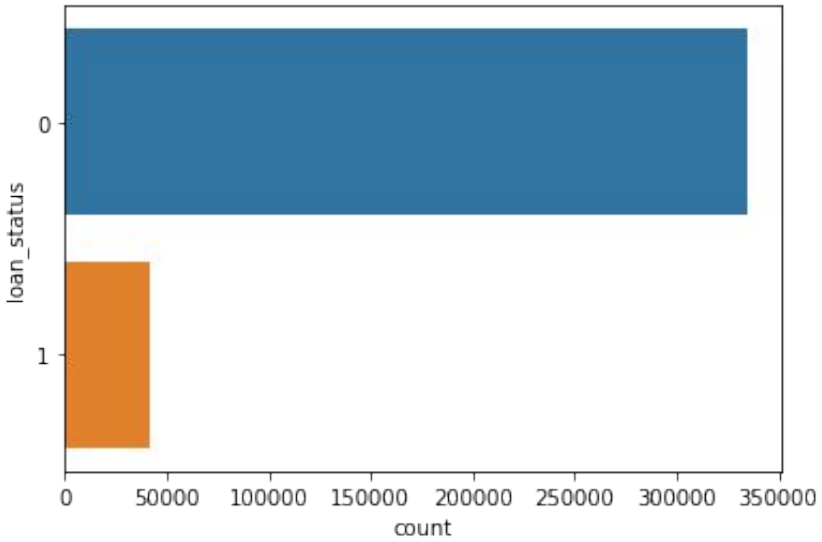
Change the value

```
[ ]  # Change the value
     df['loan_status'].replace({'Fully Paid': 0, 'Current' : 0}, inplace=True)
     df['loan_status'].replace({'Charged Off': 1,
                                'Late (31-120 days)' : 1,
                                'In Grace Period' : 1,
                                'Late (16-30 days)' : 1,
                                'Default' : 1}, inplace=True)
     df['loan_status'] = df['loan_status'].astype('int')
```

```
[ ]  # Look at the distribution of Status now
     df['loan_status'].value_counts(normalize=True)
```

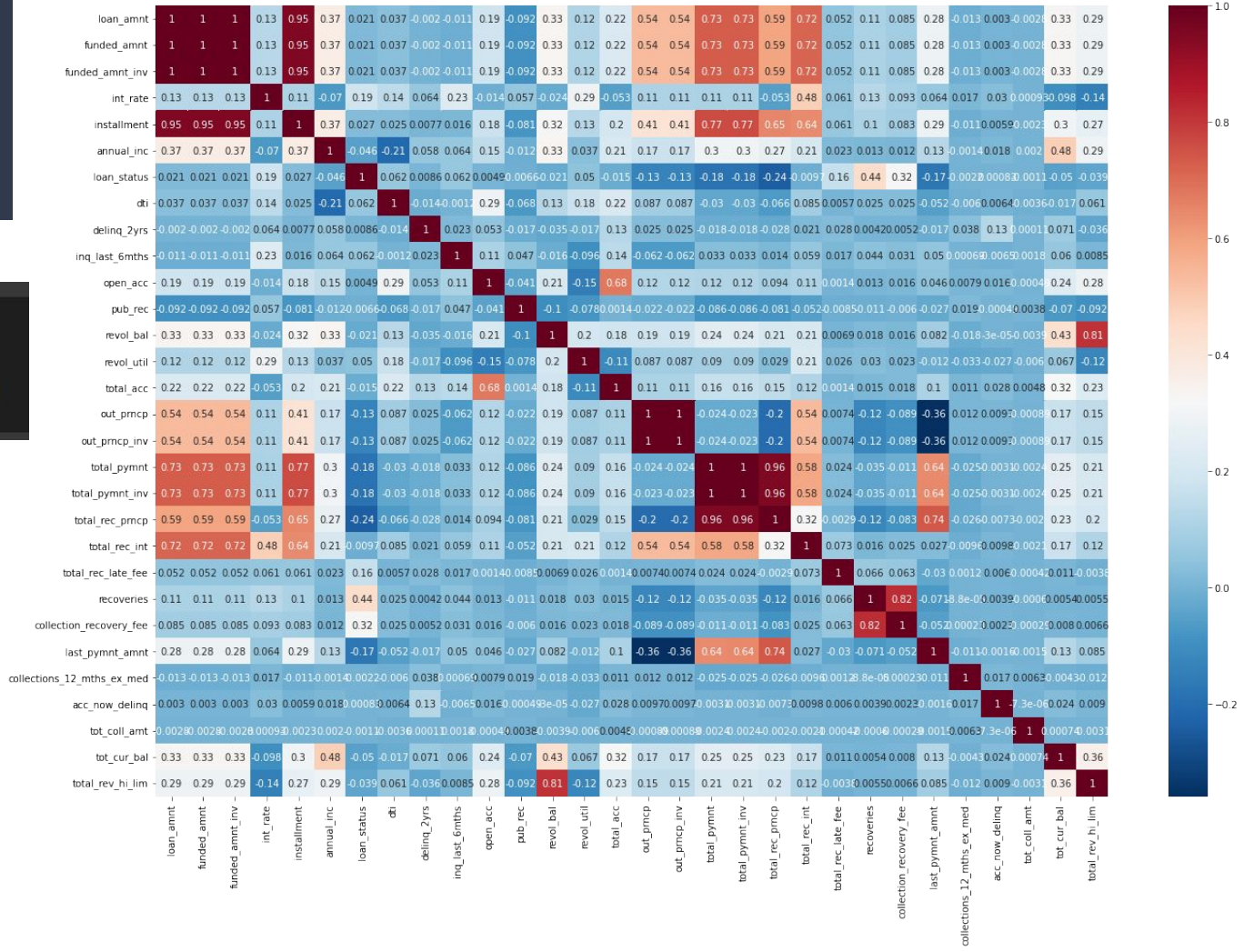
```
0    0.889123
1    0.110877
Name: loan_status, dtype: float64
```

```
[ ]  sns.countplot(y='loan_status', data=df)
     plt.show()
```



Data Preprocessing:
EDA (Multivariate)

```
[ ]  # Heatmap
     plt.figure(figsize=(22,15))
     corrs = df.corr()
     sns.heatmap(corrs, cmap='RdBu_r', annot=True)
     plt.show()
```



Data Preprocessing:

Note after the heatmap result

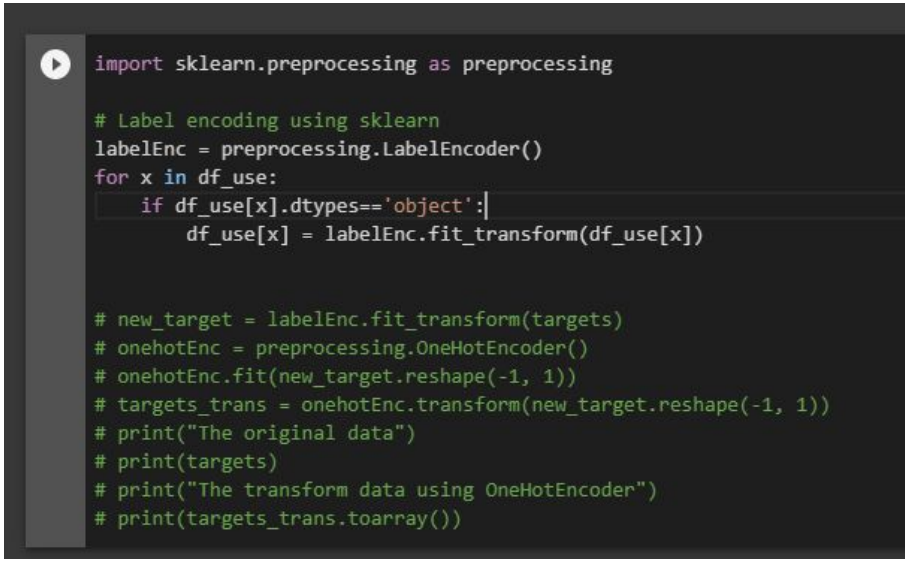
Some columns are highly correlated.

- loan_amnt, funded_amnt, funded_amnt_inv and Installment have high correlation. **We choose loan_amnt.** Because it's the value that the loan applied for by the borrower
- revol_bal and total_rev_hi_lim has high correlation. **We choose total_rev_hi_lim.**
- out_prncp and out_prncp_inv has high correlation. **We choose. We choose out_prncp**
- total_pymnt, total_pymnt_inv, total_rec_prncp has high correlation. **We choose total_pymnt.**
- collection_recoveries_fee and recoveries has high correlation. **We choose collection_recoveries_fee.**

Delete Row that is not chosen:

```
[ ]  df_use = df.copy()
     df_use.drop(columns=['funded_amnt', 'funded_amnt_inv',
                          'revol_bal', 'out_prncp_inv', 'total_pymnt_inv',
                          'total_rec_prncp', 'recoveries'
                          ], axis=1, inplace=True)
```

Data Preprocessing:
Encoding Categorical Feature



```
import sklearn.preprocessing as preprocessing

# Label encoding using sklearn
labelEnc = preprocessing.LabelEncoder()
for x in df_use:
    if df_use[x].dtypes=='object':
        df_use[x] = labelEnc.fit_transform(df_use[x])


# new_target = labelEnc.fit_transform(targets)
# onehotEnc = preprocessing.OneHotEncoder()
# onehotEnc.fit(new_target.reshape(-1, 1))
# targets_trans = onehotEnc.transform(new_target.reshape(-1, 1))
# print("The original data")
# print(targets)
# print("The transform data using OneHotEncoder")
# print(targets_trans.toarray())
```

Data Preprocessing:
Split Train Test

```
[ ]  # split to variable feature and target/labels (x and y)
     x = df_use.loc[:, df_use.columns != 'loan_status'] # X value contains all the variables except labels
     y = df_use['loan_status'] # these are the labels
```

```
[ ]  y.head()

     0    0
     1    0
     2    0
     3    0
     4    0
     Name: loan_status, dtype: int64
```

```
[ ]  # Split to Train and test
     from sklearn.model_selection import train_test_split
     x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)
```

Data Preprocessing:
Feature Scaling

```
# Scaling using Standard Scaller
from sklearn import preprocessing
ss = preprocessing.StandardScaler()
x_train_scaled = pd.DataFrame(ss.fit_transform(x_train), columns=x_train.columns)
x_test_scaled = pd.DataFrame(ss.transform(x_test), columns=x_test.columns)


# # Scalling using MinMaxScaler
# from sklearn.preprocessing import MinMaxScaler
# mms = MinMaxScaler()
# X_scaled = pd.DataFrame(mms.fit_transform(X_train), columns=X_train.columns)
# X_test_scaled = pd.DataFrame(mms.transform(X_test), columns=X_test.columns)
# # we have now fit and transform the data into a scaler for accurate reading and results.
```

Data Preprocessing:
Handling Imbalanced Data

```
[ ] # Oversampling
    from imblearn.over_sampling import SMOTE
    oversample = SMOTE()
    x_train_balanced, y_train_balanced = oversample.fit_resample(x_train_scaled, y_train)
    x_test_balanced, y_test_balanced = oversample.fit_resample(x_test_scaled, y_test)
```

```
[ ] y_train.value_counts()
```

```
0    234550
1     29245
Name: loan_status, dtype: int64
```

```
[ ] y_train_balanced.value_counts()
```

```
1    234550
0    234550
Name: loan_status, dtype: int64
```

```
[ ] y_test.value_counts()
```

```
0    100516
1     12539
Name: loan_status, dtype: int64
```

```
[ ] y_test_balanced.value_counts()
```

```
0    100516
1    100516
Name: loan_status, dtype: int64
```

Modeling

```
[ ]  # import libraries
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.metrics import roc_curve,auc

     # from sklearn.linear_model import LogisticRegression
     # from sklearn.neighbors import KNeighborsClassifier
     # from sklearn.tree import DecisionTreeClassifier
     # from sklearn.svm import SVC
     # from xgboost import XGBClassifier
```

```
[ ]  # Training Phase
     rf=RandomForestClassifier()
     model_rf = rf.fit(x_train_balanced,y_train_balanced)
     train_score = rf.score(x_train_balanced, y_train_balanced)
     test_score = rf.score(x_test_balanced, y_test_balanced)

     # Predict
     pred_rf=rf.predict(x_test_balanced)
     probs_rf=rf.predict_proba(x_test_balanced)[:,1]
```

Evaluation Metrics

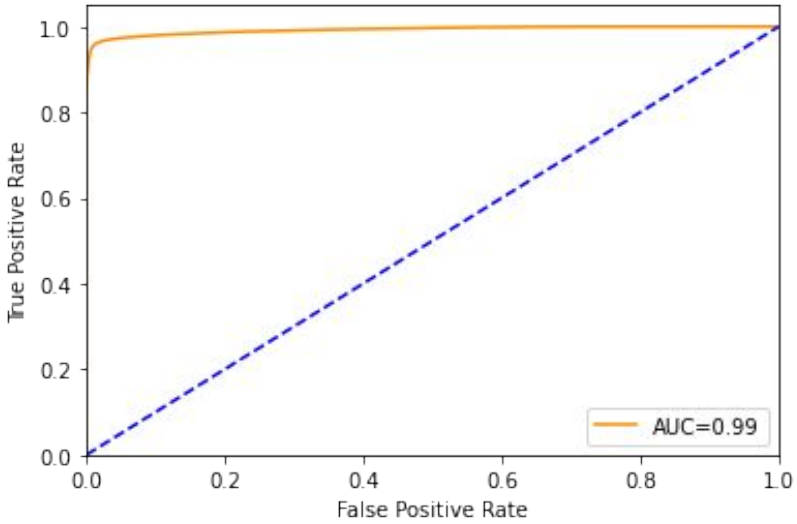
```
[ ] from sklearn.metrics import classification_report
    print(classification_report(y_test_balanced, pred_rf))
```

	precision	recall	f1-score	support
0	0.93	1.00	0.96	100671
1	1.00	0.92	0.96	100671
accuracy			0.96	201342
macro avg	0.96	0.96	0.96	201342
weighted avg	0.96	0.96	0.96	201342

Evaluation Metrics

```
[ ]  # Create a function to plot ROC Curves
     def plot_roc(y_test,probs):
         fpr,tpr,threshold=roc_curve(y_test_balanced,probs)
         roc_auc=auc(fpr,tpr)
         print('ROC AUC=%0.2f'%roc_auc)
         plt.plot(fpr,tpr,label='AUC=%0.2f'%roc_auc,color='darkorange')
         plt.legend(loc='lower right')
         plt.plot([0,1],[0,1],'b--')
         plt.xlim([0,1])
         plt.ylim([0,1.05])
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.show()
```

```
#random forest
plot_roc(y_test,probs_rf)
```



Result of Evaluation Metric

- **The recall** means "how many of this class you find over the whole number of element of this class"
- **The precision** will be "how many are correctly classified among that class"
- **The f1-score** is the harmonic mean between precision & recall
- The support is the number of occurence of the given class in your dataset (so we have 100671 of class 0 and 100671 of class 1, which is a really well balanced dataset.
- We have imbalanced dataset (before oversampling), precision and recall is highly used for imbalanced dataset. Because in an highly imbalanced dataset, a 99% accuracy can be meaningless.
- **AUC - ROC** curve is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes.
- Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1. By analogy, the Higher the AUC, the better the model is at distinguishing between good loaner and bad loaner.

Result of Evaluation Metric

- **Precision**: Based on the classification report, out of all the bad loaner that the model predicted, 100% actually did. Either they do 'Charged Off', 'Default', 'Late (31-120 days)', 'In Grace Period', 'Late (16-30 days)', 'Does not meet the credit policy. Status:Fully Paid', or 'Does not meet the credit policy. Status:Charged Off'.
- **Recall**: Based on the classification report, out of all the bad loaners that actually do something bad (late, charged off, etc), the model predicted this outcome correctly for 92%.
- **F1 Score**: Since this value is very close to 1, it tells us that the model does a good job of predicting whether or not the loaner will be a bad loaner or not.
- **AUC:** Has value 0.99, it means the model can differentiate 0 and 1 in classification.

Save the model

```
[ ]  # save the model
     from pickle import dump
     from pickle import load
     filename = '/content/drive/MyDrive/0.Data_analyst_important_thing/1.VIX_rakamin_DS_IDX/Tugas2_VIX_Rakamin_DS_IDX/file/finalized_model.sav'
     dump(model_rf, open(filename, 'wb'))
```

References:

Dataset:

- https://www.kaggle.com/datasets/wordsforthewise/lending-club?datasetId=902&sortBy=voteCount
- https://www.kaggle.com/code/faressayah/lending-club-loan-defaulters-prediction

Data Understanding

- https://www.lawinsider.com/dictionary/fully-paid
- https://www.nerdwallet.com/article/finance/credit-card-debt-charged-off
- https://lawinsider.com/dictionary/current-loan#:~:text=Current%20Loan%20means%20any%20loan,date%20according%20to%20a%20contract
- https://www.bankrate.com/glossary/l/loan-default/#:~:text=Defaulting%20on%20a%20loan%20is,he%20defaults%20on%20the%20loan.
- https://www.uxax.org/post/data-cleaning-and-preparation-for-machine-learning#:~:text=2Does%20not%20meet%20the%20credit,approved%20on%20to%20the%20marketplace.
- https://www.uxax.org/post/data-cleaning-and-preparation-for-machine-learning#:~:text=2Does%20not%20meet%20the%20credit,approved%20on%20to%20the%20marketplace.
- https://www.analyticsvidhya.com/blog/2021/07/data-leakage-and-its-effect-on-the-performance-of-an-ml-model/

Data Cleaning:

- https://towardsdatascience.com/creating-boxplots-of-well-log-data-using-matplotlib-in-python-34c3816e73f4
- Unique value: https://www.geeksforgeeks.org/pandas-find-unique-values-from-multiple-columns/#:~:text=Pandas%20series%20aka%20columns%20has,unique%20of%20the%20resultant%20column.
- Percentage missing values: https://www.thiscodeworks.com/python-find-out-the-percentage-of-missing-values-in-each-column-in-the-given-dataset-stack-overflow-python/607d4c3f6013b5001411542c
- Handling missing value: https://www.analyticsvidhya.com/blog/2021/10/handling-missing-value/
- Check duplicate Value: https://www.geeksforgeeks.org/find-duplicate-rows-in-a-dataframe-based-on-all-or-selected-columns

References:

- replace value: https://predictivehacks.com/?all-tips=replace-values-based-on-index-in-pandas-dataframes#:~:text=You%20can%20easily%20replace%20a,its%20column%20and%20its%20index.&text=Having%20the%20dataframe%20above%2C%20we,the%20second%20is%20its%20column.
- Outlier: https://www.analyticsvidhya.com/blog/2021/05/why-you-shouldnt-just-delete-outliers/
- Outlier: https://stats.stackexchange.com/questions/200534/is-it-ok-to-remove-outliers-from-data

Data preprocessing:

- https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/
- https://vitalflux.com/minmaxscaler-standardscaler-python-examples/#:~:text=The%20MinMaxscaler%20is%20a%20type,range%20from%20min%20to%20max.
- https://ai.plainenglish.io/exploratory-data-analysis-eda-and-data-preprocessing-a-beginners-guide-b1be46338006
- https://medium.com/analytics-vidhya/data-preprocessing-and-exploratory-data-analysis-for-machine-learning-75b8a6468b72#:~:text=Exploratory%20data%20analysis%20is%20often,work%20with%20statistics%20and%20data.&text=Preprocessing%3A,them%20and%20some%20statistical%20decision.
- https://medium.com/@ugursavci/complete-exploratory-data-analysis-using-python-9f685d67d1e4
- https://www.analyticsvidhya.com/blog/2021/04/exploratory-analysis-using-univariate-bivariate-and-multivariate-analysis-techniques/#:~:text=The%20objective%20of%20univariate%20analysis,of%20variables%2D%20Categorical%20and%20Numerical.
- https://towardsdatascience.com/13-key-code-blocks-for-eda-classification-task-94890622be57

Modeling

- https://machinelearningmastery.com/framework-for-imbalanced-classification-projects/#:~:text=A%20common%20way%20to%20deal,like%20Random%20Forest%20or%20SMOTE.
- https://towardsdatascience.com/machine-learning-with-python-classification-complete-tutorial-d2c99dc524ec
- https://towardsdatascience.com/detecting-credit-card-fraud-using-machine-learning-a3d83423d3b8
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

References:

- https://github.com/vappiah/Machine-Learning-Tutorials/blob/main/notebooks/ROC%20Curves-Binary%20Classification.ipynb
- https://towardsdatascience.com/top-5-metrics-for-evaluating-classification-model-83ede24c7584
- https://medium.com/datasciencestory/performance-metrics-for-evaluating-a-model-on-an-imbalanced-data-set-1feeab6c36fe
- https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/
- https://datascience.stackexchange.com/questions/64441/how-to-interpret-classification-report-of-scikit-learn
- https://www.statology.org/sklearn-classification-report/
- https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5#:~:text=Higher%20the%20AUC%2C%20the%20better.is%20on%20the%20x%2Daxis.
- https://machinelearningmastery.com/save-load-machine-learning-models-python-scikit-learn/