LoanTap_LogisticRegression

February 3, 2024

[1]:
```python
import numpy as np
import pandas as pd
```

[2]:
```python
df = pd.read_csv('./logistic_regression.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  object
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  object
 5   sub_grade             396030 non-null  object
 6   emp_title             373103 non-null  object
 7   emp_length            377729 non-null  object
 8   home_ownership        396030 non-null  object
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  object
 11  issue_d               396030 non-null  object
 12  loan_status           396030 non-null  object
 13  purpose               396030 non-null  object
 14  title                 394274 non-null  object
 15  dti                   396030 non-null  float64
 16  earliest_cr_line      396030 non-null  object
 17  open_acc              396030 non-null  float64
 18  pub_rec               396030 non-null  float64
 19  revol_bal             396030 non-null  float64
 20  revol_util            395754 non-null  float64
 21  total_acc             396030 non-null  float64
 22  initial_list_status   396030 non-null  object
 23  application_type      396030 non-null  object
 24  mort_acc              358235 non-null  float64
 25  pub_rec_bankruptcies  395495 non-null  float64
 26  address               396030 non-null  object
```

```
dtypes: float64(12), object(15)
memory usage: 81.6+ MB
```

[3]:
```
df.dtypes.value_counts()
```

[3]:
```
object     15
float64    12
Name: count, dtype: int64
```

[4]:
```
df.describe(include=object)
```

[4]:
```
              term   grade sub_grade emp_title emp_length home_ownership  \
count       396030  396030    396030    373103     377729         396030
unique           2       7        35    173105         11              6
top      36 months       B        B3   Teacher  10+ years       MORTGAGE
freq        302005  116018     26655      4389     126041         198348

       verification_status   issue_d loan_status            purpose  \
count               396030    396030      396030             396030
unique                   3       115           2                 14
top               Verified  Oct-2014  Fully Paid  debt_consolidation
freq                139563     14846      318357             234507

                     title earliest_cr_line initial_list_status  \
count               394274           396030              396030
unique               48816              684                   2
top     Debt consolidation         Oct-2000                   f
freq                152472             3017              238066

       application_type                      address
count            396030                       396030
unique                3                       393700
top          INDIVIDUAL  USCGC Smith\r\nFPO AE 70466
freq             395319                            8
```

[5]:
```
df.describe(include=float)
```

[5]:
```
           loan_amnt       int_rate    installment    annual_inc  \
count  396030.000000  396030.000000  396030.000000  3.960300e+05
mean    14113.888089      13.639400     431.849698  7.420318e+04
std      8357.441341       4.472157     250.727790  6.163762e+04
min       500.000000       5.320000      16.080000  0.000000e+00
25%      8000.000000      10.490000     250.330000  4.500000e+04
50%     12000.000000      13.330000     375.430000  6.400000e+04
75%     20000.000000      16.490000     567.300000  9.000000e+04
max     40000.000000      30.990000    1533.810000  8.706582e+06

                 dti       open_acc        pub_rec     revol_bal  \
```

```
count  396030.000000  396030.000000  396030.000000  3.960300e+05
mean       17.379514      11.311153       0.178191  1.584454e+04
std        18.019092       5.137649       0.530671  2.059184e+04
min         0.000000       0.000000       0.000000  0.000000e+00
25%        11.280000       8.000000       0.000000  6.025000e+03
50%        16.910000      10.000000       0.000000  1.118100e+04
75%        22.980000      14.000000       0.000000  1.962000e+04
max      9999.000000      90.000000      86.000000  1.743266e+06

           revol_util      total_acc       mort_acc  pub_rec_bankruptcies
count  395754.000000  396030.000000  358235.000000         395495.000000
mean       53.791749      25.414744       1.813991              0.121648
std        24.452193      11.886991       2.147930              0.356174
min         0.000000       2.000000       0.000000              0.000000
25%        35.800000      17.000000       0.000000              0.000000
50%        54.800000      24.000000       1.000000              0.000000
75%        72.900000      32.000000       3.000000              0.000000
max       892.300000     151.000000      34.000000              8.000000
```

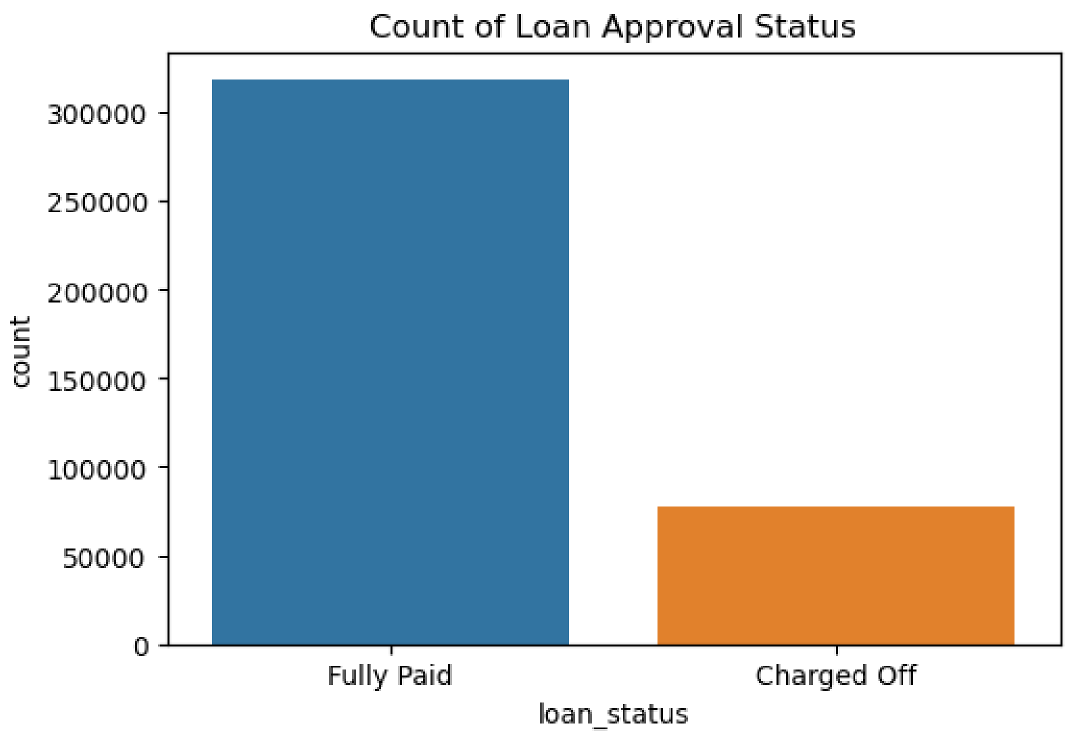
[6]:
```
df.head()
```

[6]:
```
   loan_amnt        term  int_rate  installment grade sub_grade  \
0    10000.0   36 months     11.44       329.48     B        B4
1     8000.0   36 months     11.99       265.68     B        B5
2    15600.0   36 months     10.49       506.97     B        B3
3     7200.0   36 months      6.49       220.65     A        A2
4    24375.0   60 months     17.27       609.33     C        C5

                 emp_title emp_length home_ownership  annual_inc  …  \
0                Marketing  10+ years           RENT    117000.0  …
1          Credit analyst     4 years       MORTGAGE     65000.0  …
2             Statistician   < 1 year           RENT     43057.0  …
3           Client Advocate    6 years           RENT     54000.0  …
4  Destiny Management Inc.    9 years       MORTGAGE     55000.0  …

  open_acc pub_rec revol_bal revol_util total_acc  initial_list_status  \
0     16.0     0.0   36369.0       41.8      25.0                    w
1     17.0     0.0   20131.0       53.3      27.0                    f
2     13.0     0.0   11987.0       92.2      26.0                    f
3      6.0     0.0    5472.0       21.5      13.0                    f
4     13.0     0.0   24584.0       69.8      43.0                    f

  application_type  mort_acc  pub_rec_bankruptcies  \
0       INDIVIDUAL       0.0                   0.0
1       INDIVIDUAL       3.0                   0.0
2       INDIVIDUAL       0.0                   0.0
3       INDIVIDUAL       0.0                   0.0
```

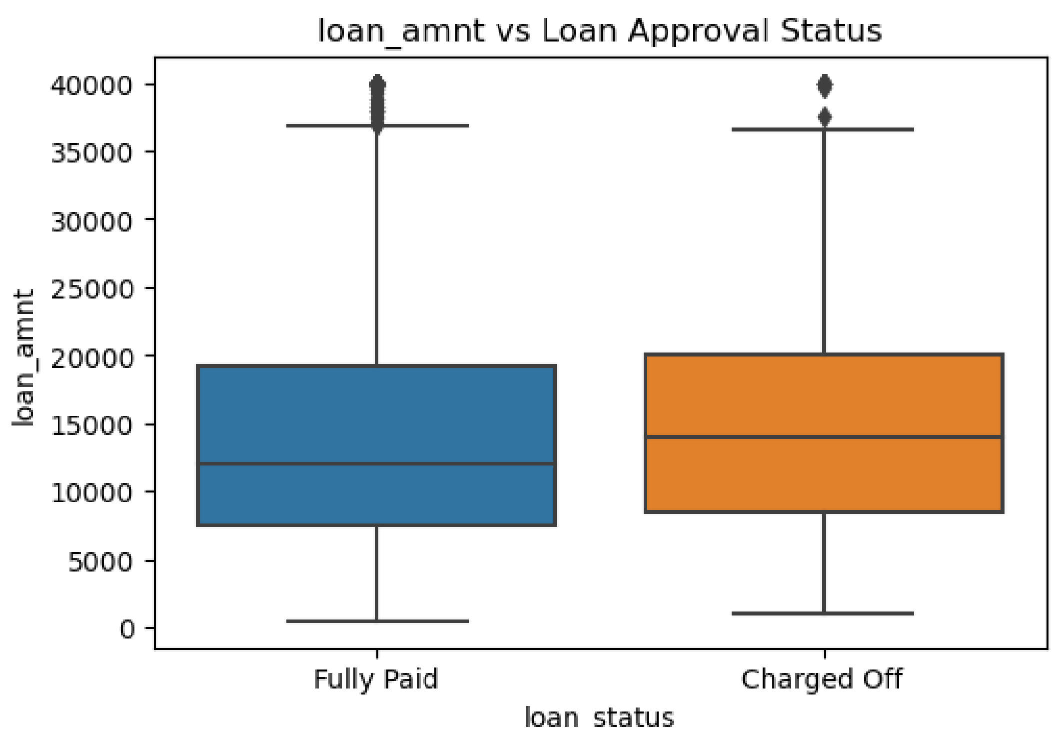
```
4       INDIVIDUAL       1.0                   0.0

                                             address
0     0174 Michelle Gateway\r\nMendozaberg, OK 22690
1  1076 Carney Fort Apt. 347\r\nLoganmouth, SD 05113
2  87025 Mark Dale Apt. 269\r\nNew Sabrina, WV 05113
3            823 Reid Ford\r\nDelacruzside, MA 00813
4             679 Luna Roads\r\nGreggshire, VA 11650

[5 rows x 27 columns]
```

0.1 Data Visualisation

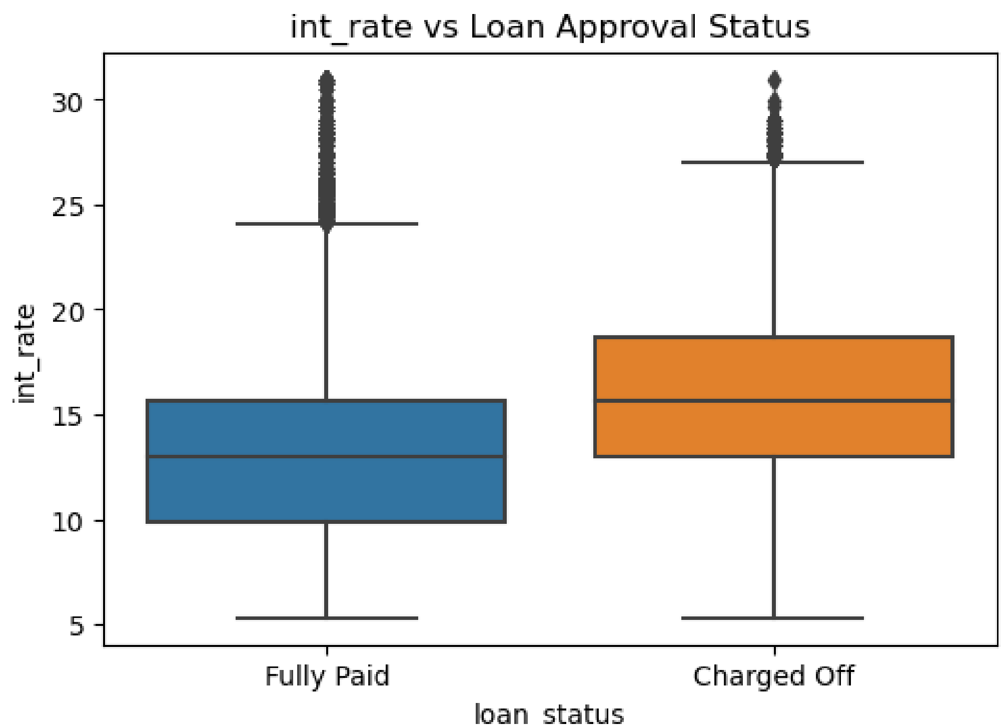
[7]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'df' is your DataFrame

# Count plot for Loan_Status
plt.figure(figsize=(6, 4))
sns.countplot(x='loan_status', data=df)
plt.title('Count of Loan Approval Status')
plt.show()

numerical_columns = df.select_dtypes(include=['float64', 'int64'])
# Box plot for Loan_Amount and Loan_Status
for variable in numerical_columns:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x='loan_status', y=variable, data=df)
    plt.title( variable + ' vs Loan Approval Status')
    plt.show()
```

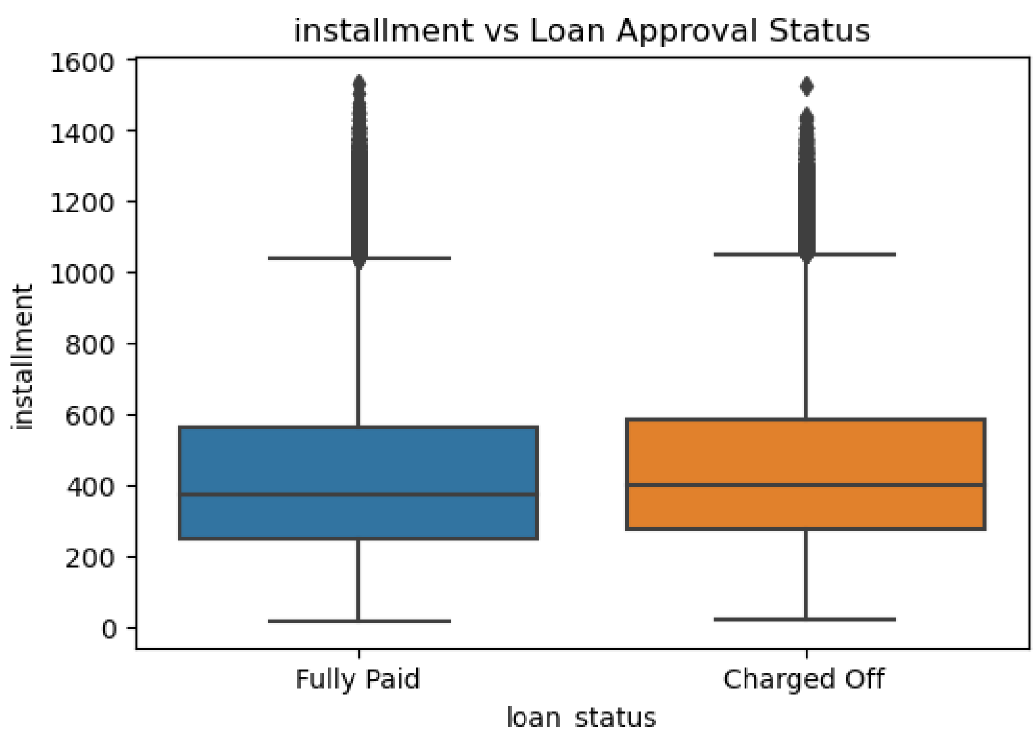


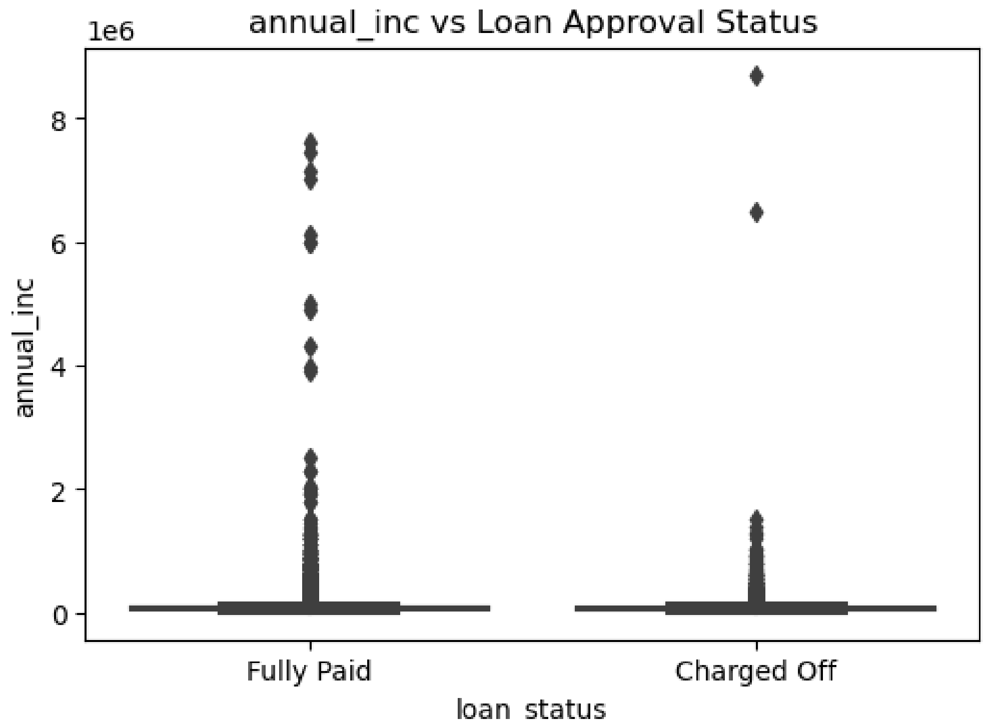
Count of Loan Approval Status
300000
250000
200000
150000
100000
50000
0
count
Fully Paid
Charged Off
loan_status

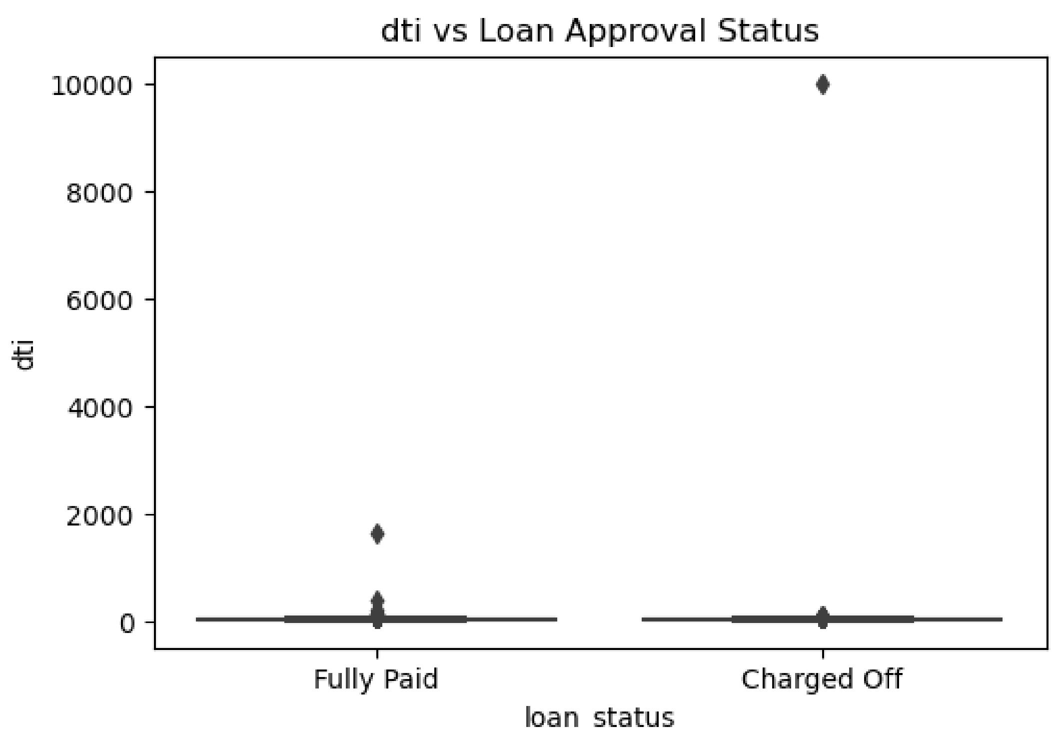


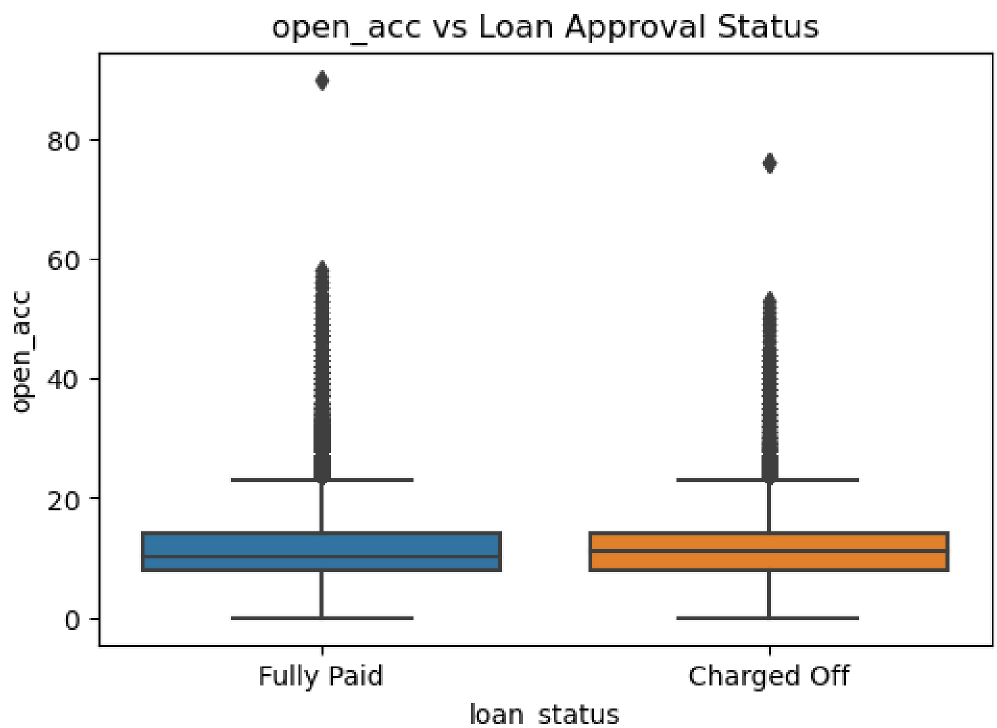
loan_amnt vs Loan Approval Status
40000
35000
30000
25000
20000
15000
10000
5000
0
loan_amnt
Fully Paid
Charged Off
loan_status

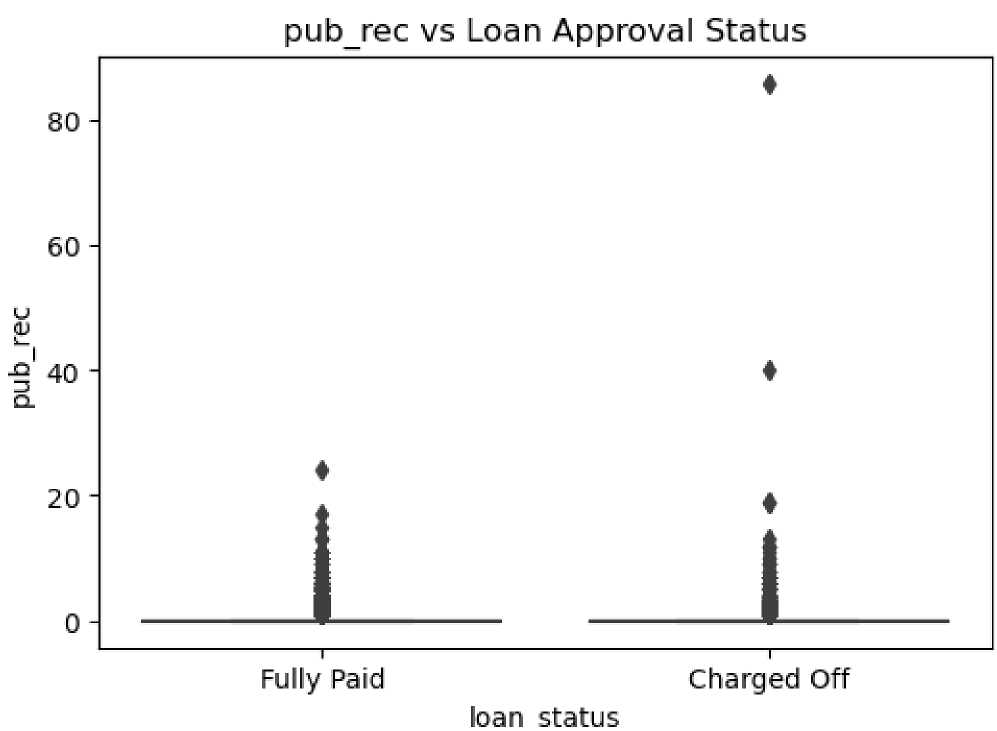

int_rate vs Loan Approval Status
30
25
20
15
10
5
int_rate
Fully Paid
Charged Off
loan_status

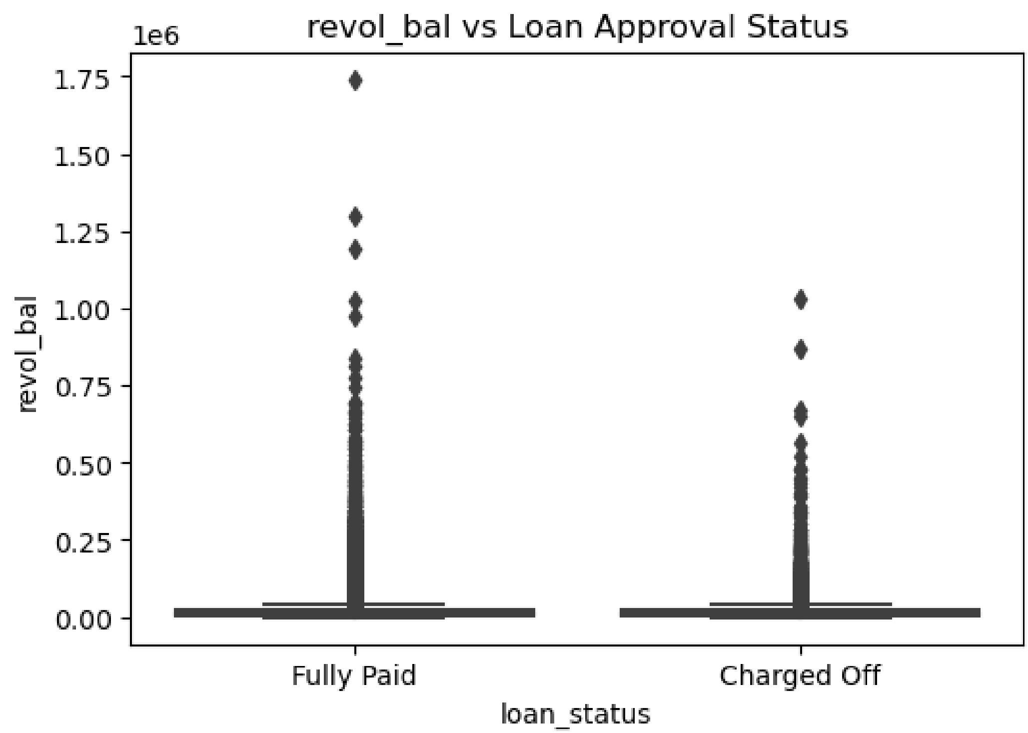

installment vs Loan Approval Status
1600
1400
1200
1000
800
600
400
200
0
installment
Fully Paid
Charged Off
loan_status

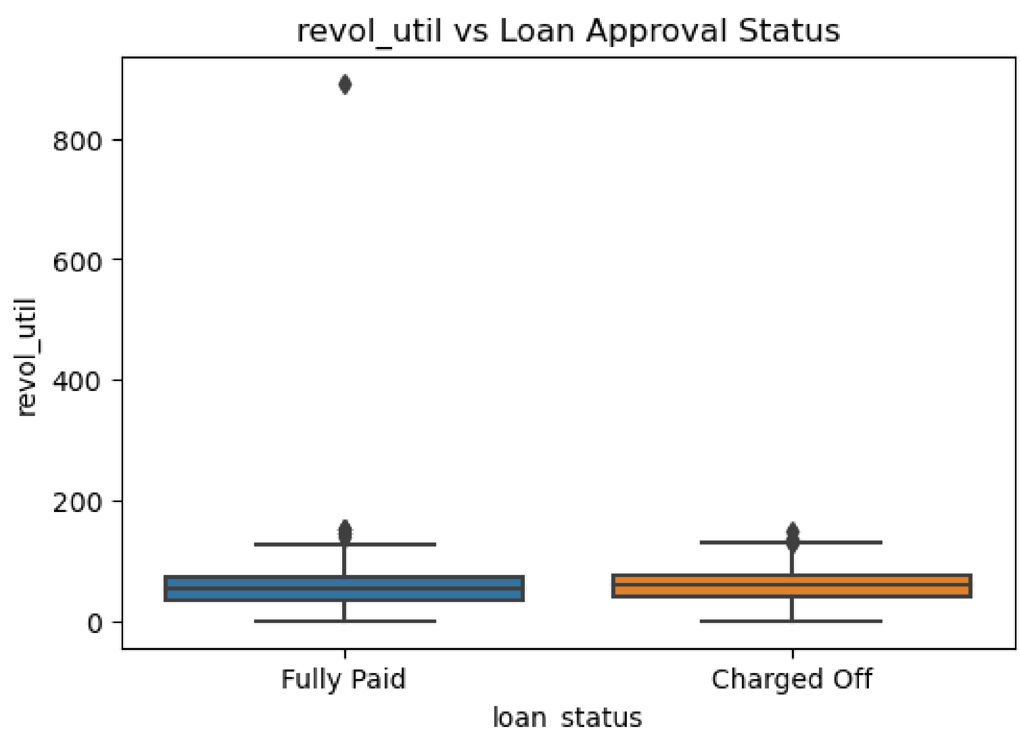

annual_inc vs Loan Approval Status
1e6
8
6
4
2
0
annual_inc
Fully Paid
Charged Off
loan_status

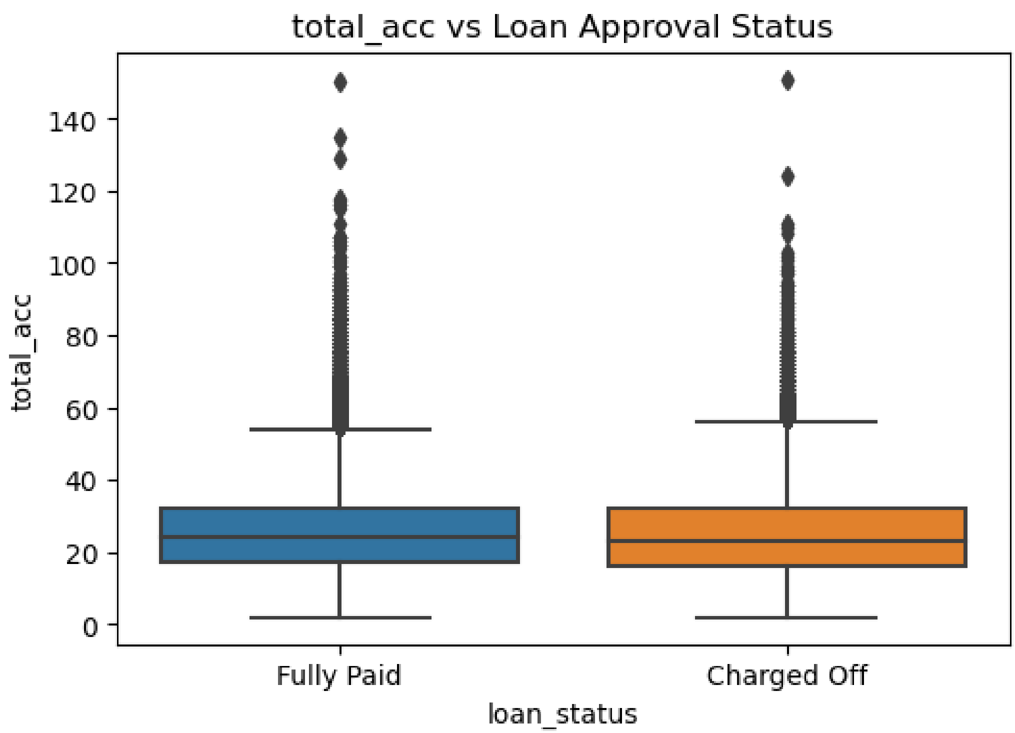

dti vs Loan Approval Status
10000
8000
6000
4000
2000
0
dti
Fully Paid
Charged Off
loan_status

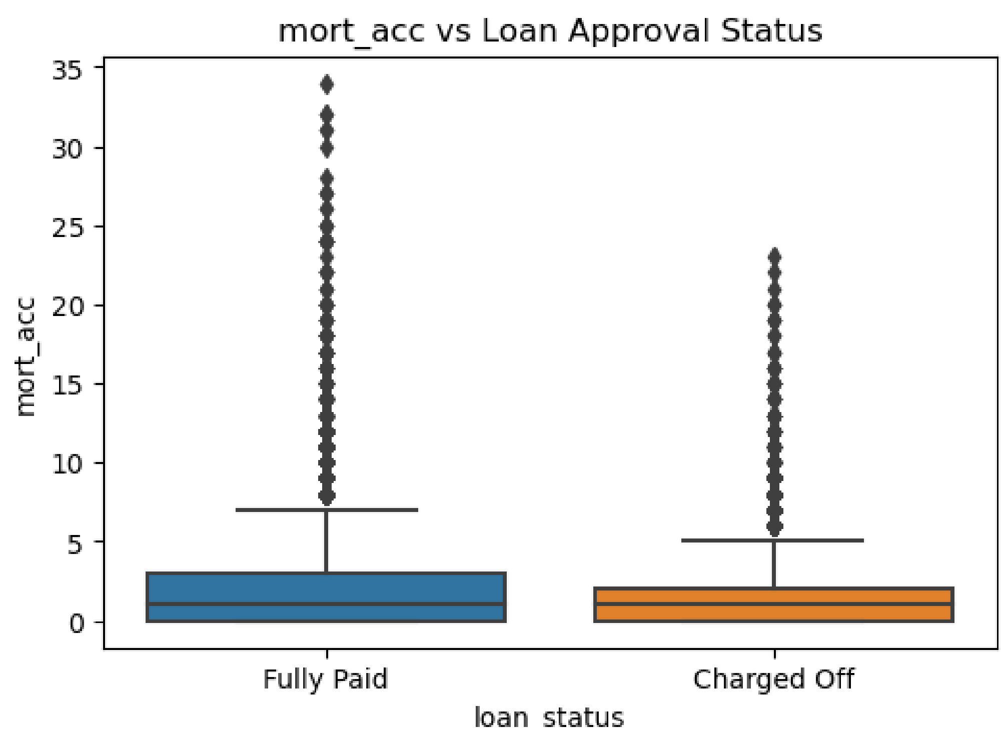

open_acc vs Loan Approval Status
80
60
40
20
0
open_acc
Fully Paid
Charged Off
loan_status

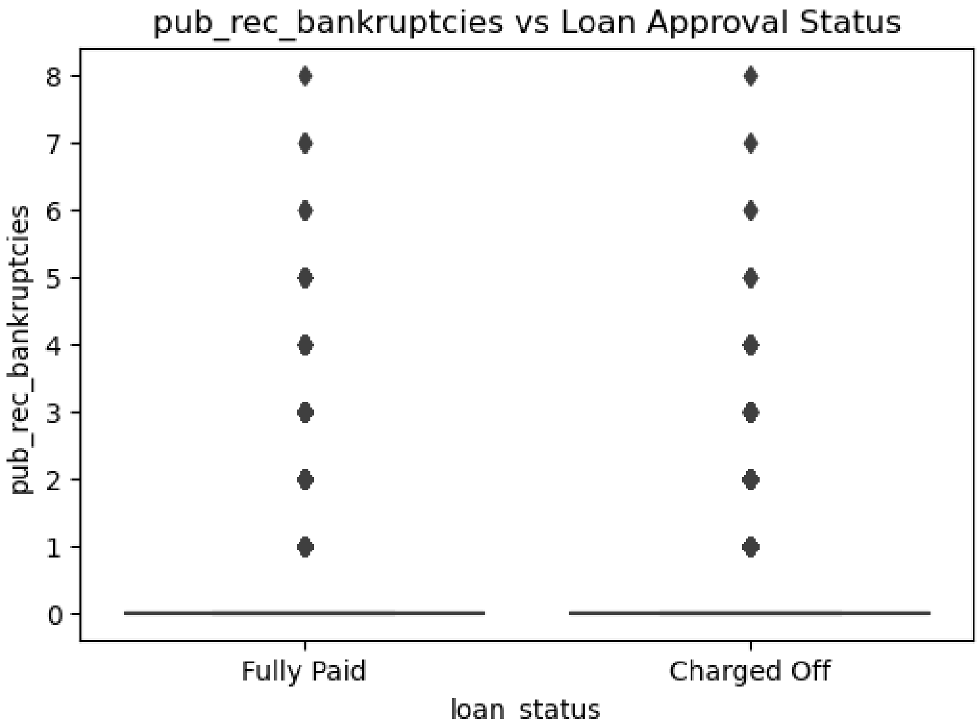

pub_rec vs Loan Approval Status
80
60
40
20
0
pub_rec
Fully Paid
Charged Off
loan_status


revol_bal vs Loan Approval Status
1e6
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
revol_bal
Fully Paid
Charged Off
loan_status

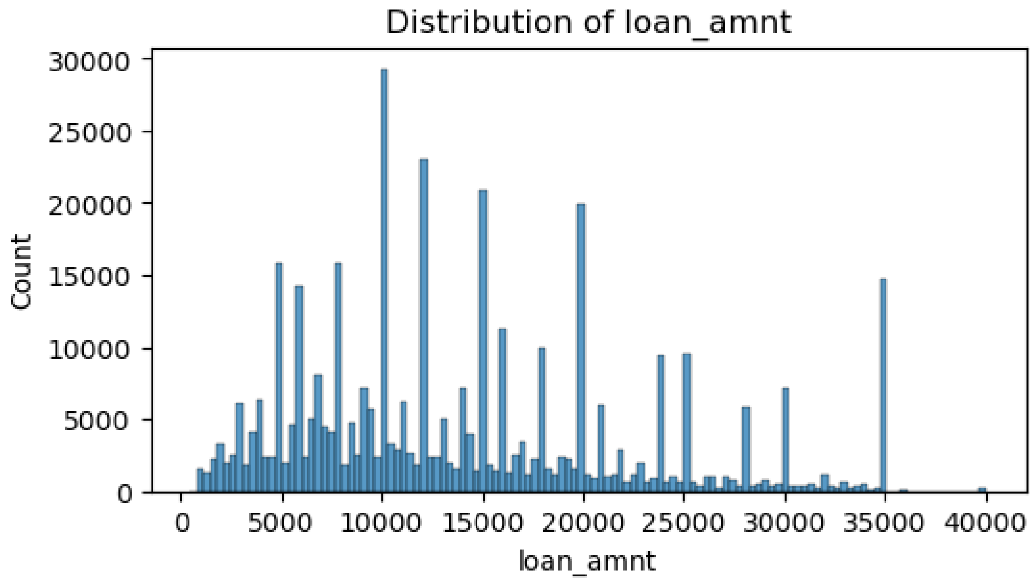

revol_util vs Loan Approval Status
800
600
400
200
0
revol_util
Fully Paid
Charged Off
loan_status

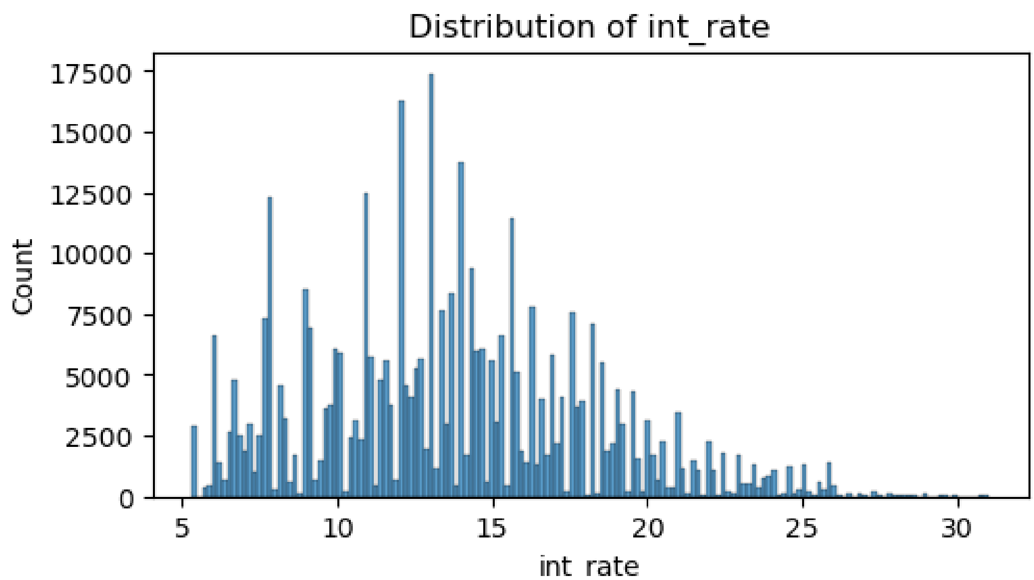

total_acc vs Loan Approval Status
140
120
100
80
60
40
20
0
total_acc
Fully Paid
Charged Off
loan_status

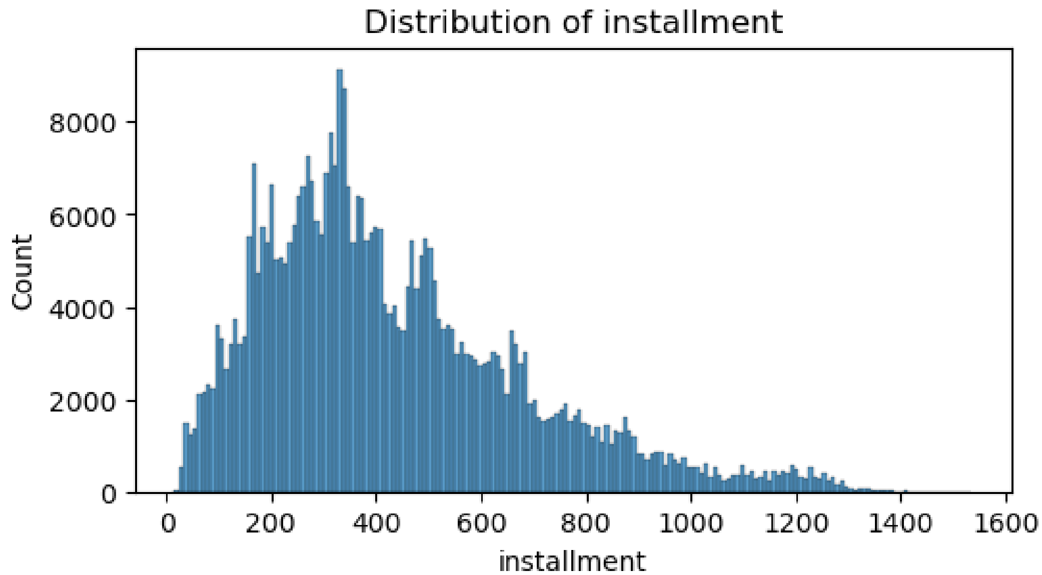

mort_acc vs Loan Approval Status
35
30
25
20
15
10
5
0
mort_acc
Fully Paid
Charged Off
loan_status

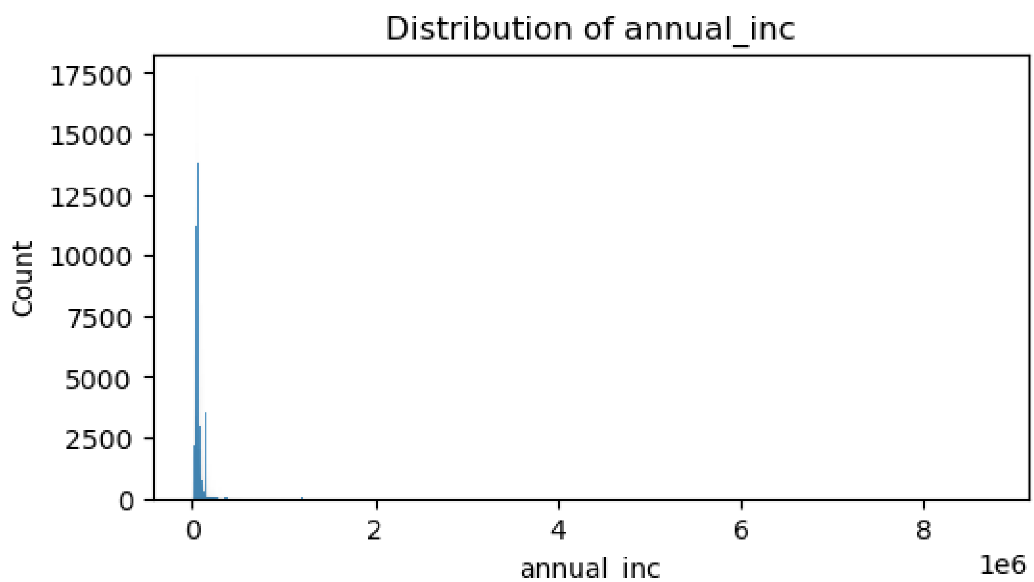


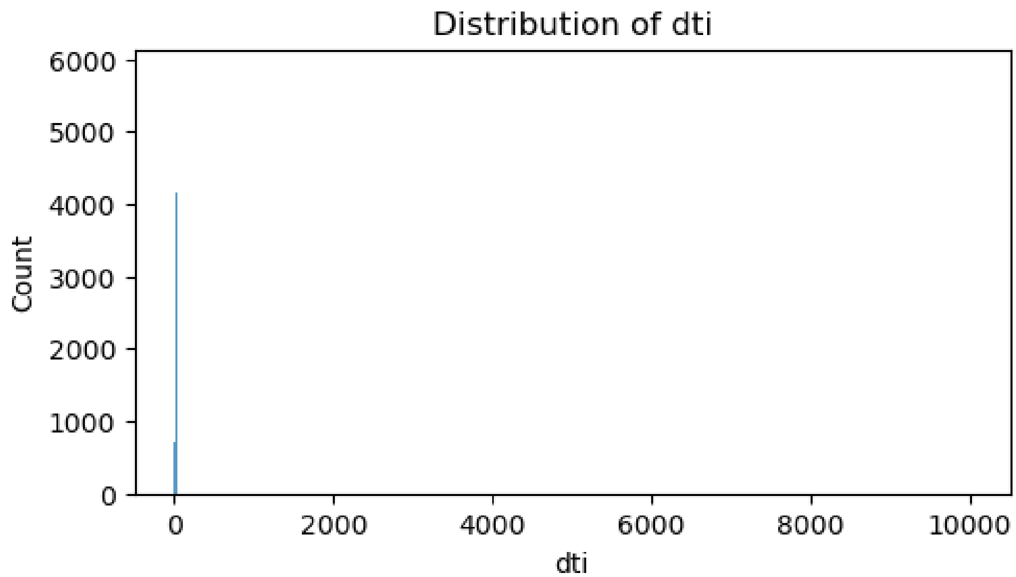
[8]:
```
subplot = 1
for variable in numerical_columns:
    plt.figure(figsize=(20,3))
    plt.subplot(1,3,subplot)
    sns.histplot(df[variable])
    plt.title( 'Distribution of ' + variable)
    subplot += 1
    if(subplot == 4):
        subplot=1

plt.show()
```

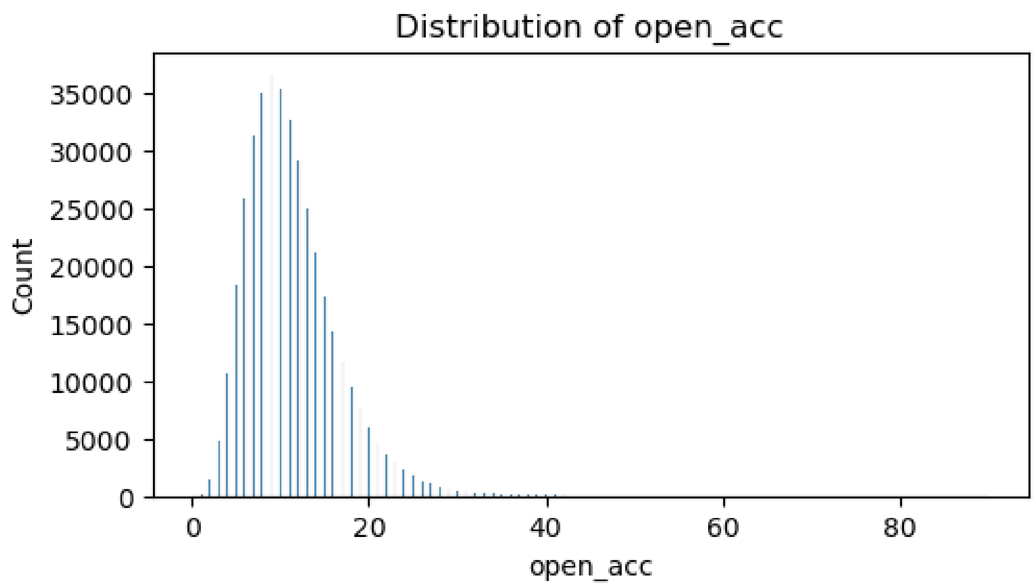

Distribution of loan_amnt
Count
30000
25000
20000
15000
10000
5000
0
0
5000
10000
15000
20000
25000
30000
35000
40000
loan_amnt

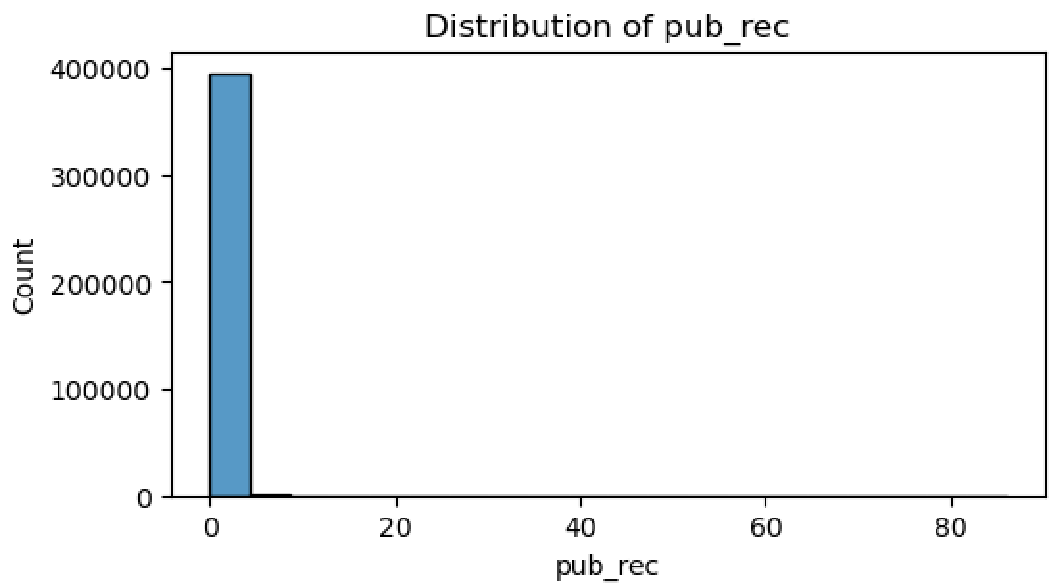

Distribution of int_rate
Count
17500
15000
12500
10000
7500
5000
2500
0
5
10
15
20
25
30
int_rate

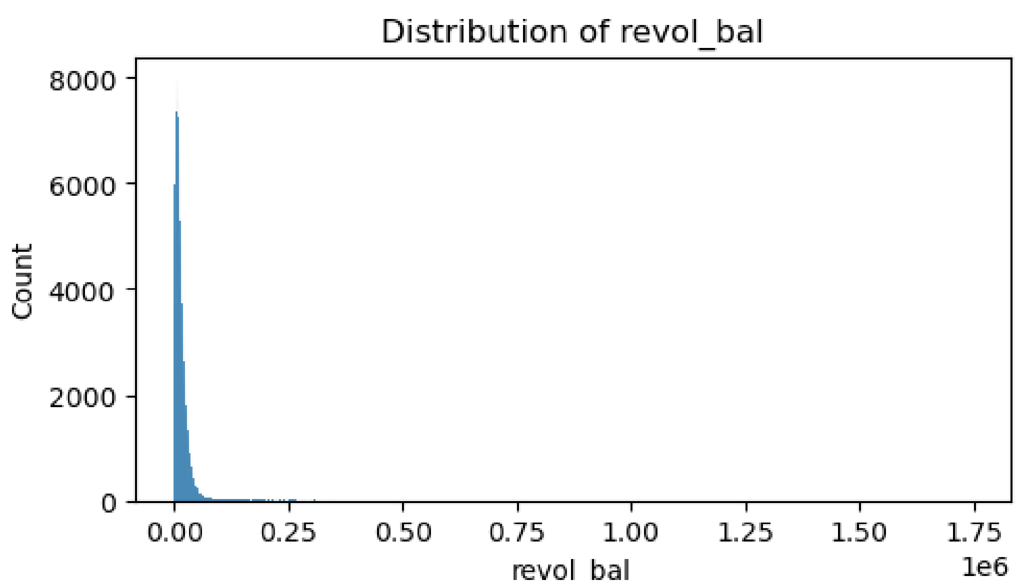

Distribution of installment
Count
8000
6000
4000
2000
0
0
200
400
600
800
1000
1200
1400
1600
installment

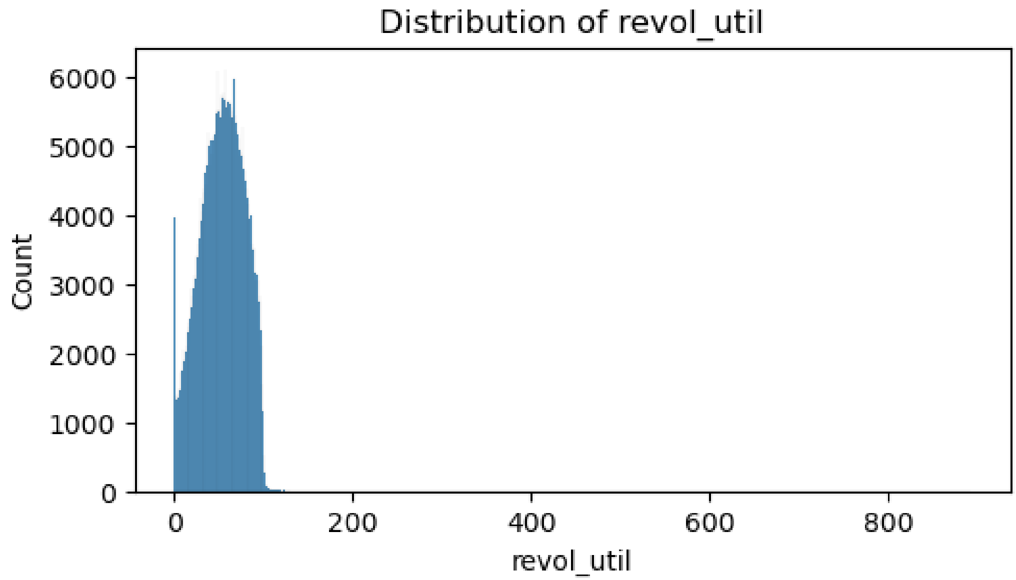

Distribution of annual_inc
Count
17500
15000
12500
10000
7500
5000
2500
0
0
2
4
6
8
1e6
annual_inc

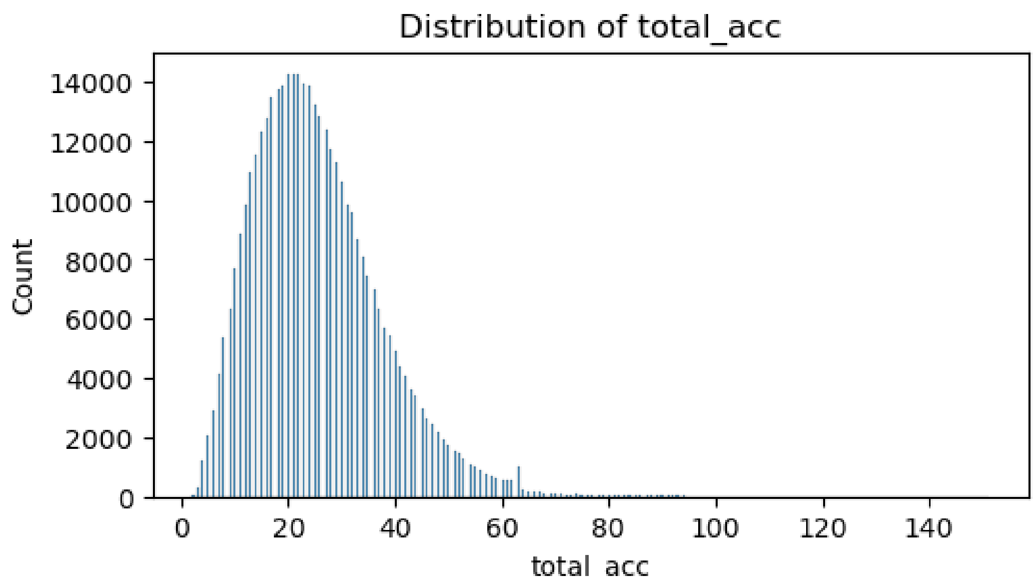

Distribution of dti
6000
5000
4000
3000
2000
1000
0
Count
0
2000
4000
6000
8000
10000
dti

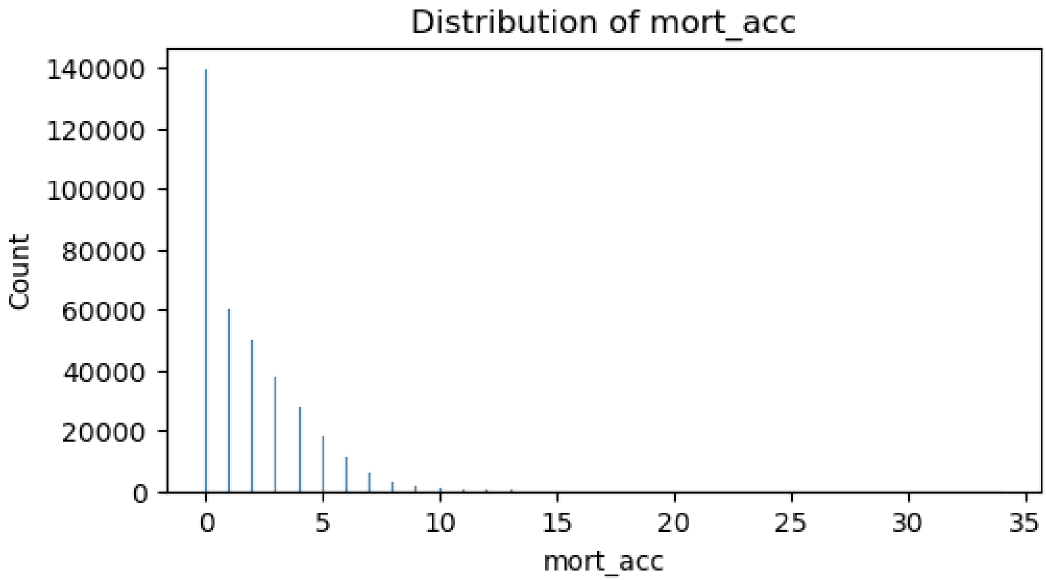

Distribution of open_acc
35000
30000
25000
20000
15000
10000
5000
0
Count
0
20
40
60
80
open_acc

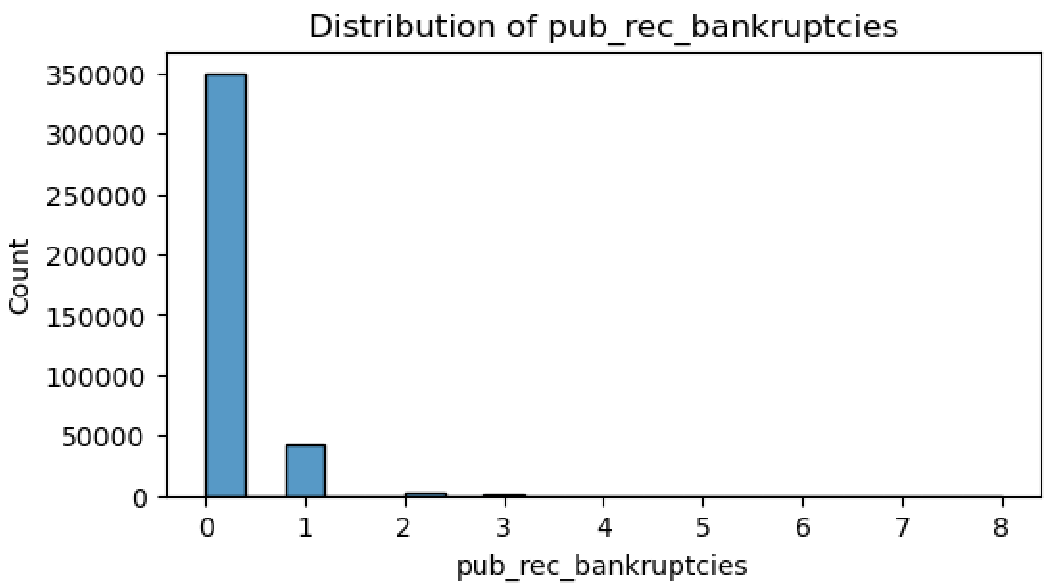

Distribution of pub_rec
Count
400000
300000
200000
100000
0
0
20
40
60
80
pub_rec


Distribution of revol_bal
Count
8000
6000
4000
2000
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
1e6
revol_bal


Distribution of revol_util
Count
revol_util
6000
5000
4000
3000
2000
1000
0
0
200
400
600
800

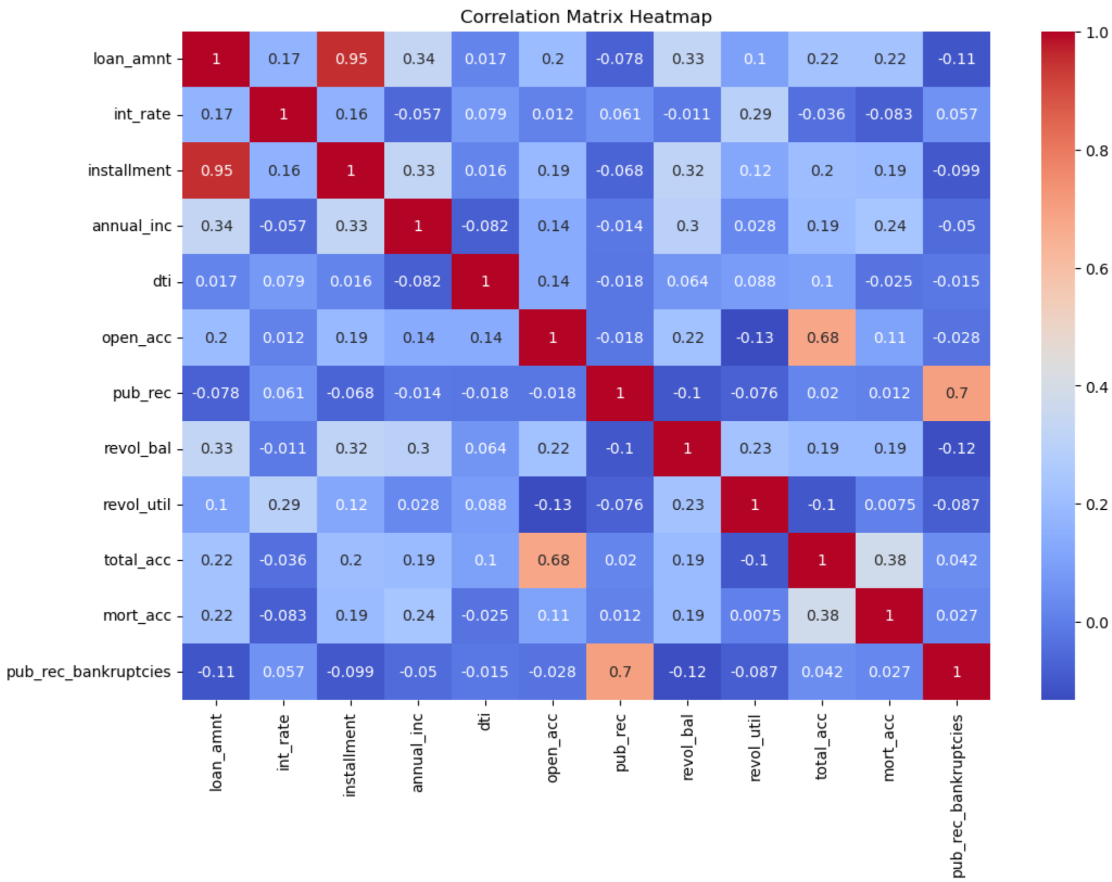

Distribution of total_acc
Count
total_acc
14000
12000
10000
8000
6000
4000
2000
0
0
20
40
60
80
100
120
140





[9]:
```
numerical_columns = df.select_dtypes(include=['float64', 'int64'])
correlation_matrix = numerical_columns.corr()

# Heat map for correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix Heatmap')
```

```
plt.show()
```



1. Missing values and outlier treatment

- Check how much data is missing - is it greater than 10% of all data?
- Do we drop this data or apply some method to fill these empty spaces?
- Are there repeated rows?
- Check how many outliers are there in each column using IQR method
- Can they be ignored/removed? Do we need to scale them?

[10]:
```
df.isnull().sum()
```

[10]:
```
loan_amnt               0
term                    0
int_rate                0
installment             0
grade                   0
sub_grade               0
```

```
emp_title               22927
emp_length              18301
home_ownership              0
annual_inc                  0
verification_status         0
issue_d                     0
loan_status                 0
purpose                     0
title                    1756
dti                         0
earliest_cr_line            0
open_acc                    0
pub_rec                     0
revol_bal                   0
revol_util                276
total_acc                   0
initial_list_status         0
application_type            0
mort_acc                37795
pub_rec_bankruptcies      535
address                     0
dtype: int64
```

[11]:
```python
# pd.get_dummies(df, columns=['grade', 'sub_grade'])
# mask = df.emp_title.isnull() and df.emp_length.isnull()
emp_legth_missing = df[df.emp_length.isnull()]
emp_title_missing = df[df.emp_title.isnull()]
emp_length_title_missing = emp_legth_missing[emp_legth_missing.emp_title.
 ↪isnull()]
emp_title_length_missing = emp_title_missing[emp_title_missing.emp_length.
 ↪isnull()]

print(len(emp_legth_missing))
print(len(emp_length_title_missing))

print(len(emp_title_missing))
print(len(emp_title_length_missing))

mort_acc_missing = df[df.mort_acc.isnull()]
len(mort_acc_missing[mort_acc_missing.pub_rec_bankruptcies.isnull()])

mort_acc_missing.head()
```

```
18301
18123
22927
18123
```

[11]:

```
    loan_amnt        term  int_rate  installment grade sub_grade  \
22     4200.0   36 months      6.99       129.67     A        A3
25     6000.0   36 months     11.36       197.47     B        B5
32     3000.0   36 months      6.03        91.31     A        A1
41    28000.0   60 months     19.91       312.04     E        E4
62     5000.0   36 months     10.39       118.45     B        B4

                    emp_title emp_length home_ownership  annual_inc  …  \
22           midstate steel llc    5 years            OWN     24000.0  …
25              CSU Monterey Bay    2 years           RENT     46680.0  …
32  American Heart Association     1 year            OWN     64000.0  …
41            American Airlines  10+ years           RENT     52000.0  …
62                         self  10+ years           RENT     66000.0  …

   open_acc pub_rec revol_bal revol_util total_acc  initial_list_status  \
22      6.0     0.0       0.0        0.0       7.0                    f
25      9.0     0.0    4370.0       40.1      10.0                    f
32      6.0     0.0    4912.0       13.4      18.0                    f
41     10.0     0.0   29178.0       87.6      16.0                    f
62     12.0     0.0   15807.0       20.0      17.0                    f

   application_type  mort_acc  pub_rec_bankruptcies  \
22       INDIVIDUAL       NaN                   0.0
25       INDIVIDUAL       NaN                   0.0
32       INDIVIDUAL       NaN                   0.0
41       INDIVIDUAL       NaN                   0.0
62       INDIVIDUAL       NaN                   0.0

                                              address
22      54395 Melissa Walks\r\nJenniferbury, AL 05113
25  44130 Powers Course Suite 880\r\nEast Preston,…
32  2722 Smith Branch Suite 131\r\nShaunbury, NH 2…
41  5836 Garcia Falls Apt. 525\r\nMatthewtown, CT …
62                         USS Goodman\r\nFPO AE 22690

[5 rows x 27 columns]
```

[12]:

```
df.drop(emp_length_title_missing.index, inplace=True)
```

[13]:

```
df.emp_title.describe()
(df.emp_title.value_counts() >= 1000).value_counts()
```

[13]:

```
count
False    173092
True         13
Name: count, dtype: int64
```

```
[14]: df.isnull().sum()
```

```
[14]: loan_amnt                   0
      term                        0
      int_rate                    0
      installment                 0
      grade                       0
      sub_grade                   0
      emp_title                4804
      emp_length                178
      home_ownership              0
      annual_inc                  0
      verification_status         0
      issue_d                     0
      loan_status                 0
      purpose                     0
      title                    1544
      dti                         0
      earliest_cr_line            0
      open_acc                    0
      pub_rec                     0
      revol_bal                   0
      revol_util                266
      total_acc                   0
      initial_list_status         0
      application_type            0
      mort_acc                36788
      pub_rec_bankruptcies        535
      address                     0
      dtype: int64
```

```
[15]: # Lets remove rows with empty emp_length
      # Lets remove title column - as 'purpose' gives that same information

      missing_emp_length = df[df.emp_length.isnull()]
      df.drop(missing_emp_length.index, inplace=True)
```

```
[16]: df.isnull().sum()
```

```
[16]: loan_amnt                   0
      term                        0
      int_rate                    0
      installment                 0
      grade                       0
      sub_grade                   0
      emp_title                4804
      emp_length                  0
```

```
home_ownership             0
annual_inc                 0
verification_status        0
issue_d                    0
loan_status                0
purpose                    0
title                   1542
dti                        0
earliest_cr_line           0
open_acc                   0
pub_rec                    0
revol_bal                  0
revol_util               265
total_acc                  0
initial_list_status        0
application_type           0
mort_acc               36739
pub_rec_bankruptcies     535
address                    0
dtype: int64
```

[17]:
```
emp_title_counts = df['emp_title'].value_counts()
df['emp_title_counts'] = df['emp_title'].apply(lambda x: 0 if (type(x)==float)␣
 ↪else emp_title_counts[x])[40]
```

[18]:
```
missing_revol_util = df[df.revol_util.isnull()]
df.drop(missing_revol_util.index, inplace=True)

missing_pub_rec_bankruptcies = df[df.pub_rec_bankruptcies.isnull()]
df.drop(missing_pub_rec_bankruptcies.index, inplace=True)
```

[19]:
```
df.isnull().sum()
```

[19]:
```
loan_amnt                  0
term                       0
int_rate                   0
installment                0
grade                      0
sub_grade                  0
emp_title               4768
emp_length                 0
home_ownership             0
annual_inc                 0
verification_status        0
issue_d                    0
loan_status                0
purpose                    0
```

```
title                    1541
dti                         0
earliest_cr_line            0
open_acc                    0
pub_rec                     0
revol_bal                   0
revol_util                  0
total_acc                   0
initial_list_status         0
application_type            0
mort_acc                36154
pub_rec_bankruptcies        0
address                     0
emp_title_counts            0
dtype: int64
```

We will use MICE imputation to impute values where we have missing mort_acc

Lets first do encoding of categorical variables and also do outlier treatment

[20]:
```
df.describe(include=object)
```

[20]:
```
              term   grade sub_grade emp_title emp_length home_ownership  \
count       376929  376929    376929    372161     376929         376929
unique           2       7        35    172566         11              6
top      36 months       B        B3   Teacher  10+ years       MORTGAGE
freq        285598  110538     25430      4386     125876         189448

       verification_status   issue_d loan_status             purpose  \
count               376929    376929      376929              376929
unique                   3       112           2                  14
top        Source Verified  Oct-2014  Fully Paid  debt_consolidation
freq                127694     14140      304440              224188

                     title earliest_cr_line initial_list_status  \
count               375388           376929              376929
unique               46765              665                   2
top     Debt consolidation         Oct-2000                   f
freq                145223             2918              226586

       application_type                        address
count            376929                         376929
unique                3                         374809
top          INDIVIDUAL  USNS Johnson\r\nFPO AE 05113
freq             376292                              8
```

We can see that sub_grade include information about grade so we can remove that

[ ]:
```

```

Lets first deal with categorical columns. We have total 13 of them - term - grade - sub_grade - emp_length - home_ownership - verification_status - issue_d - loan_status - purpose - earliest_cr_line - initial_list_status - application_type

[21]:
```
# Column = purpose
dummies_purposes = pd.get_dummies(df.purpose, dtype=int)
merged = pd.concat([df, dummies_purposes], axis = 'columns')
merged.drop(columns = ['purpose', 'other'], inplace=True)
merged.drop(columns = ['emp_title'], inplace=True)
merged.drop(columns = ['grade'], inplace=True)
merged.drop(columns = ['address'], inplace=True)
merged.drop(columns = ['title'], inplace=True)
```

[22]:
```
# Column = emp_length
year_values_str = np.sort(list(df['emp_length'].unique()))
years_values_bins = ['0-2 years','10+ years','0-2 years','3-6 years','3-6␣
↪years','3-6 years','3-6 years','7-9 years','7-9 years','7-9 years','0-2␣
↪years']
years_bin_mapping = dict(zip(year_values_str, years_values_bins))

# years_bin_mapping['1 year']

emp_length_bin = df['emp_length'].apply(lambda x: years_bin_mapping[x])
dummies_emp_length = pd.get_dummies(emp_length_bin,dtype=int)
merged = pd.concat([merged, dummies_emp_length], axis = 'columns')
merged.drop(columns = ['emp_length', '0-2 years'], inplace=True)
```

[23]:
```
# column = application_type
dummies_application_type = pd.get_dummies(df['application_type'], dtype=int)
merged = pd.concat([merged, dummies_application_type], axis = 'columns')
merged.drop(columns = ['application_type', 'DIRECT_PAY'], inplace=True)
```

[24]:
```
# Column = issue_d

import calendar

data = np.array(df['issue_d'].apply(lambda x: x.split('-')))
calendar_dict = {month: index for index, month in enumerate(calendar.
↪month_abbr) if month}

def eval_numeric_val(x):
    num_val_arr = []
    for elem in x:
        curr = calendar_dict[elem[0]] + (int(elem[1])-1900)*12
        num_val_arr.append(curr)
    return np.array(num_val_arr)
```

```
date_eval_numeric = eval_numeric_val(data)
merged['issued_date_transformed'] = date_eval_numeric
merged.drop(columns = ['issue_d'], inplace=True)
```

[25]:
```
# Column = earliest_cr_line

data = np.array(df['earliest_cr_line'].apply(lambda x: x.split('-')))
date_eval_numeric = eval_numeric_val(data)
merged['earliest_cr_line_date_transformed'] = date_eval_numeric
merged.drop(columns = ['earliest_cr_line'], inplace=True)
```

[26]:
```
dummies_initial_list_status = pd.get_dummies(df['initial_list_status'],␣
 ↪dtype=int)
dummies_verification_status = pd.get_dummies(df['verification_status'],␣
 ↪dtype=int)
dummies_home_ownership = pd.get_dummies(df['home_ownership'], dtype=int)
dummies_term = pd.get_dummies(df['term'], dtype=int)

merged = pd.concat([merged, dummies_initial_list_status,␣
 ↪dummies_verification_status, dummies_home_ownership, dummies_term], axis =␣
 ↪'columns')
merged.drop(columns = ['initial_list_status', 'verification_status',␣
 ↪'home_ownership', 'term'], inplace=True)
merged.drop(columns=[' 60 months','NONE', 'Not Verified', 'w'], inplace=True)
```

[27]:
```
def get_subgrade_mapping(x):
    mapping = {}
    for item in x:
        mapping[item] = (ord(item[0]) - ord('A'))*5 + ord(item[1]) - ord('0')
    return mapping

grades_unique = np.array(merged.sub_grade.unique())
grade_mapping = get_subgrade_mapping(grades_unique)
```

[28]:
```
merged['sub_grade']
merged['sub_grade'] = merged['sub_grade'].apply(lambda x: grade_mapping[x])

# from sklearn.preprocessing import OrdinalEncoder

# encoder = OrdinalEncoder()
# encoder.fit(pd.DataFrame(merged.sub_grade))
# encoder.categories_

# encoder.transform(pd.DataFrame(merged.sub_grade))
# merged['sub_grade'] = encoder.transform(pd.DataFrame(merged.sub_grade))
```

[29]:
```
merged['loan_status'].replace({'Fully Paid': 1, 'Charged Off': 0}, inplace=True)
```

[30]:
```python
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

merged_copy = merged.copy()
missing_mask = merged_copy.mort_acc.isna()

imputer = IterativeImputer(max_iter=10, random_state=0)
imputed_values = imputer.fit_transform(merged_copy)
```

[31]:
```python
type(imputed_values[missing_mask])
merged.columns.get_loc('mort_acc')
merged['mort_acc'] = imputed_values[:, 12]
```

0.1.1 Outlier Treatment

[32]:
```python
to_scale=['loan_amnt', 'int_rate', 'installment', 'annual_inc',
            'dti', 'open_acc', 'pub_rec', 'revol_bal',
            'revol_util', 'total_acc', 'mort_acc', 'pub_rec_bankruptcies']
```

[33]:
```python
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
merged[to_scale] = scaler.fit_transform(merged[to_scale])
```

0.2 Test Train Split

[34]:
```python
from sklearn.model_selection import train_test_split
train, test = train_test_split(merged, train_size = 0.70, random_state = 100)
```

Let's deal with other data first - we will use that data to predict missing values 1. Encoding data 2. conversion to correct data type

[35]:
```python
X_train = train.drop(columns='loan_status')
X_test = test.drop(columns='loan_status')

Y_train = train['loan_status']
Y_test = test['loan_status']
```

0.3 Logistic Regression modelling

[36]:
```python
from sklearn.linear_model import LogisticRegression
X_train.isnull().sum()

lr = LogisticRegression()
lr.fit(X_train, Y_train)
```

[36]:
```
LogisticRegression()
```

[37]:
```python
Y_pred = lr.predict(X_test)
```

[38]:
```
from sklearn.metrics import accuracy_score, recall_score, precision_score,␣
↪roc_auc_score, f1_score
print('Accuracy: ', accuracy_score(Y_test, lr.predict(X_test)))
print('Recall: ', recall_score(Y_test, lr.predict(X_test)))
print('Precision: ', precision_score(Y_test, lr.predict(X_test)))
print('F1-score: ', f1_score(Y_test, lr.predict(X_test)))
print('ROC-AUC: ', roc_auc_score(Y_test, lr.predict(X_test)))
```

```
Accuracy:  0.8063389311896992
Recall:  0.9847904165389164
Precision:  0.8142967087690706
F1-score:  0.891464992144482
ROC-AUC:  0.5210093199679183
```

[39]:
```
from sklearn.linear_model import LogisticRegression
logReg=LogisticRegression()
from sklearn.feature_selection import RFE
rfe = RFE(logReg, n_features_to_select = 30)
rfe.fit(X_train, Y_train)
```

```
/Users/vaibhavmotwani/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/vaibhavmotwani/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/vaibhavmotwani/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
```

```
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/vaibhavmotwani/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/vaibhavmotwani/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[39]:
```
RFE(estimator=LogisticRegression(), n_features_to_select=30)
```

[40]:
```
selected_features = X_train.columns[rfe.support_]
selected_features
```

[40]:
```
Index(['loan_amnt', 'int_rate', 'installment', 'sub_grade', 'annual_inc',
       'dti', 'open_acc', 'revol_util', 'total_acc', 'mort_acc',
       'pub_rec_bankruptcies', 'car', 'credit_card', 'debt_consolidation',
       'home_improvement', 'house', 'major_purchase', 'medical', 'moving',
       'small_business', 'vacation', 'wedding', '10+ years', 'JOINT',
       'Source Verified', 'Verified', 'MORTGAGE', 'OWN', 'RENT', ' 36 months'],
      dtype='object')
```

[41]:
```
X_train_rfe = X_train[selected_features]
Y_train_rfe = Y_train

X_test_rfe = X_test[selected_features]
Y_test_rfe = Y_test
```

[42]:
```
lr_rfe = LogisticRegression()
lr_rfe.fit(X_train_rfe, Y_train_rfe)
```

```
/Users/vaibhavmotwani/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[42]:
```
LogisticRegression()
```

[43]:
```
print('Accuracy: ', accuracy_score(Y_test_rfe, lr_rfe.predict(X_test_rfe)))
print('Recall: ', recall_score(Y_test_rfe, lr_rfe.predict(X_test_rfe)))
print('Precision: ', precision_score(Y_test_rfe, lr_rfe.predict(X_test_rfe)))
print('F1-score: ', f1_score(Y_test_rfe, lr_rfe.predict(X_test_rfe)))
print('ROC-AUC: ', roc_auc_score(Y_test_rfe, lr_rfe.predict(X_test_rfe)))
```

```
Accuracy:  0.8074266663129317
Recall:  0.9836187639612807
Precision:  0.8158171976604788
F1-score:  0.8918940386830295
ROC-AUC:  0.5257096347960851
```

[44]:
```
import statsmodels.api as sm
X_train_sm = sm.add_constant(X_train_rfe)
X_test_sm = sm.add_constant(X_test_rfe)
logreg = sm.GLM(Y_train, X_train_sm, family = sm.families.Binomial())
res = logreg.fit()
res.summary()
```

[44]:

Dep. Variable:	loan_status	**No. Observations:**	263850
Model:	GLM	**Df Residuals:**	263819
Model Family:	Binomial	**Df Model:**	30
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-1.1764e+05
Date:	Sat, 03 Feb 2024	**Deviance:**	2.3527e+05
Time:	12:54:45	**Pearson chi2:**	4.06e+11
No. Iterations:	6	**Pseudo R-squ. (CS):**	0.08364
Covariance Type:	nonrobust		

	coef	std err	z	P> |z|	[0.025	0.975]
const	2.4382	0.287	8.492	0.000	1.875	3.001
loan_amnt	0.2318	0.171	1.359	0.174	-0.103	0.566
int_rate	1.2746	0.129	9.905	0.000	1.022	1.527
installment	-0.6708	0.205	-3.269	0.001	-1.073	-0.269
sub_grade	-0.1033	0.003	-30.955	0.000	-0.110	-0.097
annual_inc	24.8308	1.459	17.022	0.000	21.972	27.690
dti	-9.2741	0.277	-33.428	0.000	-9.818	-8.730
open_acc	-1.7628	0.130	-13.601	0.000	-2.017	-1.509
revol_util	-2.5237	0.216	-11.700	0.000	-2.946	-2.101
total_acc	1.2520	0.102	12.314	0.000	1.053	1.451
mort_acc	1.2250	0.122	10.009	0.000	0.985	1.465
pub_rec_bankruptcies	-0.2904	0.118	-2.461	0.014	-0.522	-0.059
car	0.2239	0.061	3.693	0.000	0.105	0.343
credit_card	0.0542	0.026	2.108	0.035	0.004	0.105
debt_consolidation	-0.0080	0.023	-0.345	0.730	-0.054	0.038
home_improvement	-0.0709	0.032	-2.223	0.026	-0.133	-0.008
house	0.1230	0.073	1.693	0.091	-0.019	0.265
major_purchase	0.0283	0.044	0.650	0.516	-0.057	0.114
medical	-0.0976	0.053	-1.826	0.068	-0.202	0.007
moving	-0.0583	0.061	-0.949	0.342	-0.179	0.062
small_business	-0.4292	0.044	-9.807	0.000	-0.515	-0.343
vacation	-0.0522	0.069	-0.753	0.451	-0.188	0.084
wedding	0.6476	0.097	6.681	0.000	0.458	0.838
10+ years	0.0336	0.011	2.938	0.003	0.011	0.056
JOINT	1.5417	0.231	6.670	0.000	1.089	1.995
Source Verified	-0.1486	0.014	-10.822	0.000	-0.175	-0.122
Verified	-0.0534	0.014	-3.731	0.000	-0.081	-0.025
MORTGAGE	0.2014	0.283	0.711	0.477	-0.354	0.757
OWN	0.0964	0.284	0.340	0.734	-0.460	0.653
RENT	-0.0449	0.283	-0.158	0.874	-0.600	0.511
36 months	0.4659	0.028	16.399	0.000	0.410	0.522

[46]:
```
X_train_sm.drop(columns = ['loan_amnt', 'debt_consolidation','house', 'moving',
                        'vacation', 'MORTGAGE', 'OWN', 'RENT'], inplace=True)

X_test_sm.drop(columns = ['loan_amnt', 'debt_consolidation','house', 'moving',
                        'vacation', 'MORTGAGE', 'OWN', 'RENT'], inplace=True)
```

[47]:
```
logreg = sm.GLM(Y_train, X_train_sm, family = sm.families.Binomial())
res = logreg.fit()
res.summary()
```

[47]:

Dep. Variable:	loan_status	**No. Observations:**	263850
Model:	GLM	**Df Residuals:**	263827
Model Family:	Binomial	**Df Model:**	22
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-1.1782e+05
Date:	Sat, 03 Feb 2024	**Deviance:**	2.3564e+05
Time:	12:55:32	**Pearson chi2:**	6.62e+11
No. Iterations:	6	**Pseudo R-squ. (CS):**	0.08237
Covariance Type:	nonrobust		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	2.4733	0.029	84.261	0.000	2.416	2.531
int_rate	1.2544	0.127	9.896	0.000	1.006	1.503
installment	-0.3606	0.038	-9.473	0.000	-0.435	-0.286
sub_grade	-0.1050	0.003	-31.543	0.000	-0.111	-0.098
annual_inc	25.4720	1.453	17.527	0.000	22.624	28.320
dti	-9.1425	0.277	-33.037	0.000	-9.685	-8.600
open_acc	-1.5537	0.129	-12.048	0.000	-1.806	-1.301
revol_util	-2.4480	0.214	-11.436	0.000	-2.868	-2.028
total_acc	1.1448	0.101	11.304	0.000	0.946	1.343
mort_acc	2.2363	0.112	20.013	0.000	2.017	2.455
pub_rec_bankruptcies	-0.3022	0.118	-2.566	0.010	-0.533	-0.071
car	0.2254	0.057	3.963	0.000	0.114	0.337
credit_card	0.0578	0.014	4.193	0.000	0.031	0.085
home_improvement	-0.0005	0.024	-0.020	0.984	-0.047	0.046
major_purchase	0.0315	0.038	0.824	0.410	-0.044	0.107
medical	-0.0817	0.049	-1.653	0.098	-0.179	0.015
small_business	-0.4187	0.039	-10.825	0.000	-0.494	-0.343
wedding	0.6384	0.095	6.739	0.000	0.453	0.824
10+ years	0.0492	0.011	4.313	0.000	0.027	0.071
JOINT	1.5786	0.231	6.832	0.000	1.126	2.031
Source Verified	-0.1541	0.014	-11.238	0.000	-0.181	-0.127
Verified	-0.0561	0.014	-3.921	0.000	-0.084	-0.028
36 months	0.4109	0.013	31.596	0.000	0.385	0.436

[50]:
```
X_train_sm.drop(columns = ['home_improvement', 'medical'], inplace=True)

X_test_sm.drop(columns = ['home_improvement', 'medical'], inplace=True)
```

[51]:
```
logreg = sm.GLM(Y_train, X_train_sm, family = sm.families.Binomial())
res = logreg.fit()
res.summary()
```

[51]:

Dep. Variable:	loan_status	**No. Observations:**	263850
Model:	GLM	**Df Residuals:**	263829
Model Family:	Binomial	**Df Model:**	20
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-1.1782e+05
Date:	Sat, 03 Feb 2024	**Deviance:**	2.3564e+05
Time:	12:55:59	**Pearson chi2:**	6.06e+11
No. Iterations:	6	**Pseudo R-squ. (CS):**	0.08236
Covariance Type:	nonrobust		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	2.4726	0.029	84.611	0.000	2.415	2.530
int_rate	1.2547	0.127	9.899	0.000	1.006	1.503
installment	-0.3559	0.038	-9.382	0.000	-0.430	-0.282
sub_grade	-0.1051	0.003	-31.581	0.000	-0.112	-0.099
annual_inc	25.3832	1.449	17.519	0.000	22.543	28.223
dti	-9.1422	0.277	-33.058	0.000	-9.684	-8.600
open_acc	-1.5516	0.129	-12.034	0.000	-1.804	-1.299
revol_util	-2.4379	0.213	-11.434	0.000	-2.856	-2.020
total_acc	1.1439	0.101	11.297	0.000	0.945	1.342
mort_acc	2.2351	0.112	20.042	0.000	2.017	2.454
pub_rec_bankruptcies	-0.2998	0.118	-2.546	0.011	-0.531	-0.069
car	0.2271	0.057	3.995	0.000	0.116	0.338
credit_card	0.0588	0.014	4.289	0.000	0.032	0.086
major_purchase	0.0331	0.038	0.867	0.386	-0.042	0.108
small_business	-0.4170	0.039	-10.803	0.000	-0.493	-0.341
wedding	0.6402	0.095	6.759	0.000	0.455	0.826
10+ years	0.0491	0.011	4.309	0.000	0.027	0.071
JOINT	1.5784	0.231	6.831	0.000	1.126	2.031
Source Verified	-0.1540	0.014	-11.231	0.000	-0.181	-0.127
Verified	-0.0561	0.014	-3.922	0.000	-0.084	-0.028
36 months	0.4100	0.013	31.555	0.000	0.384	0.435

[52]:
```
from sklearn.linear_model import LogisticRegression
logReg=LogisticRegression()
logReg.fit(X_train_sm, Y_train)



print('Accuracy: ', accuracy_score(Y_test, logReg.predict(X_test_sm)))
print('Recall: ', recall_score(Y_test, logReg.predict(X_test_sm)))
print('Precision: ', precision_score(Y_test, logReg.predict(X_test_sm)))
print('F1-score: ', f1_score(Y_test, logReg.predict(X_test_sm)))
print('ROC-AUC: ', roc_auc_score(Y_test, logReg.predict(X_test_sm)))
```

```
/Users/vaibhavmotwani/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
Accuracy:  0.807205581938291
Recall:  0.9835092637203802
Precision:  0.8156893373171197
F1-score:  0.8917726137700621
ROC-AUC:  0.5253101363419185
```

[53]:
```python
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
vif['feature'] = X_train_sm.columns
vif['VIF'] = [variance_inflation_factor(X_train_sm.values, i) for i in␣
 ↪range(X_train_sm.shape[1])]
vif
```

[53]:

	feature	VIF
0	const	29.405449
1	int_rate	20.793923
2	installment	1.335669
3	sub_grade	21.358557
4	annual_inc	1.260856
5	dti	1.302582
6	open_acc	2.118563
7	revol_util	1.258774
8	total_acc	2.352994
9	mort_acc	1.387336
10	pub_rec_bankruptcies	1.048924
11	car	1.015943
12	credit_card	1.065908
13	major_purchase	1.023189
14	small_business	1.024127
15	wedding	1.005803
16	10+ years	1.059076
17	JOINT	1.002547
18	Source Verified	1.465741
19	Verified	1.599920
20	36 months	1.391789

[54]:
```python
X_train_sm.drop(columns = ['int_rate','sub_grade'], inplace=True)

X_test_sm.drop(columns = ['int_rate','sub_grade'], inplace=True)
```

[55]:
```python
vif = pd.DataFrame()
vif['feature'] = X_train_sm.columns
vif['VIF'] = [variance_inflation_factor(X_train_sm.values, i) for i in␣
 ↪range(X_train_sm.shape[1])]
vif
```

[55]:

	feature	VIF
0	const	23.486834
1	installment	1.319067
2	annual_inc	1.249622
3	dti	1.293576
4	open_acc	2.114046
5	revol_util	1.148242
6	total_acc	2.344877
7	mort_acc	1.370813
8	pub_rec_bankruptcies	1.031302
9	car	1.015442
10	credit_card	1.028627
11	major_purchase	1.023058
12	small_business	1.015854
13	wedding	1.005427
14	10+ years	1.058748
15	JOINT	1.002466
16	Source Verified	1.456114
17	Verified	1.575133
18	36 months	1.081333

[56]:
```python
from sklearn.linear_model import LogisticRegression
logReg=LogisticRegression()
logReg.fit(X_train_sm, Y_train)


print('Accuracy: ', accuracy_score(Y_test, logReg.predict(X_test_sm)))
print('Recall: ', recall_score(Y_test, logReg.predict(X_test_sm)))
print('Precision: ', precision_score(Y_test, logReg.predict(X_test_sm)))
print('F1-score: ', f1_score(Y_test, logReg.predict(X_test_sm)))
print('ROC-AUC: ', roc_auc_score(Y_test, logReg.predict(X_test_sm)))
```

```
Accuracy:  0.8082933170615234
Recall:  0.9957623406771495
Precision:  0.8102879851730406
F1-score:  0.8935014148718755
ROC-AUC:  0.5085453854615349
```

```
/Users/vaibhavmotwani/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

0.4 Probability cutoff tuning

[57]:
```
odds = logReg.predict_proba(X_test_sm)
odds[:, 1]
```

[57]:
```
array([0.84080931, 0.84414807, 0.8337672 , …, 0.89376915, 0.89086001,
       0.89345848])
```

[58]:
```
prob=pd.DataFrame()
prob['y_actual'] = Y_test
prob['y=1|x'] = odds[:,1]
prob
```

[58]:
```
        y_actual     y=1|x
63252          1  0.840809
118566         1  0.844148
58362          1  0.833767
174276         1  0.884254
369908         1  0.897064
…            …         …
222942         1  0.770109
167981         1  0.676179
365061         1  0.893769
310160         0  0.890860
220257         1  0.893458

[113079 rows x 2 columns]
```

[59]:
```
cut = [float(x)/20 for x in range(0, 21)]
cut
```

[59]:
```
[0.0,
 0.05,
 0.1,
 0.15,
 0.2,
 0.25,
 0.3,
 0.35,
 0.4,
```

```
 0.45,
 0.5,
 0.55,
 0.6,
 0.65,
 0.7,
 0.75,
 0.8,
 0.85,
 0.9,
 0.95,
 1.0]
```

[60]:
```python
for i in cut:
    prob[i] = prob['y=1|x'].map(lambda x: x>i)
prob
```

[60]:
```
        y_actual     y=1|x   0.0  0.05   0.1  0.15   0.2  0.25   0.3  0.35  \
63252          1  0.840809  True  True  True  True  True  True  True  True
118566         1  0.844148  True  True  True  True  True  True  True  True
58362          1  0.833767  True  True  True  True  True  True  True  True
174276         1  0.884254  True  True  True  True  True  True  True  True
369908         1  0.897064  True  True  True  True  True  True  True  True
…            …       …   …   …   …   …   …   …   …   …
222942         1  0.770109  True  True  True  True  True  True  True  True
167981         1  0.676179  True  True  True  True  True  True  True  True
365061         1  0.893769  True  True  True  True  True  True  True  True
310160         0  0.890860  True  True  True  True  True  True  True  True
220257         1  0.893458  True  True  True  True  True  True  True  True

        …  0.55   0.6  0.65    0.7   0.75    0.8   0.85    0.9   0.95    1.0
63252   …  True  True  True   True   True   True  False  False  False  False
118566  …  True  True  True   True   True   True  False  False  False  False
58362   …  True  True  True   True   True   True  False  False  False  False
174276  …  True  True  True   True   True   True   True  False  False  False
369908  …  True  True  True   True   True   True   True  False  False  False
…     …   …   …   …    …    …    …    …    …    …    …
222942  …  True  True  True   True   True  False  False  False  False  False
167981  …  True  True  True  False  False  False  False  False  False  False
365061  …  True  True  True   True   True   True   True  False  False  False
310160  …  True  True  True   True   True   True   True  False  False  False
220257  …  True  True  True   True   True   True   True  False  False  False

[113079 rows x 23 columns]
```

[61]:
```python
cut_df = pd.DataFrame(columns = ['prob', 'accuracy', 'precision', 'recall',␣
 ↪'f1-score', 'ROC-AUC'])
```

```python
for i in cut:
    accuracy = accuracy_score(prob['y_actual'], prob[i])
    precision = precision_score(prob['y_actual'], prob[i])
    recall = recall_score(prob['y_actual'], prob[i])
    f1 = f1_score(prob['y_actual'], prob[i])
    roc_auc = roc_auc_score(prob['y_actual'], prob[i])
    cut_df.loc[i] = [i, accuracy, precision, recall, f1, roc_auc]

cut_df
```

```
/Users/vaibhavmotwani/anaconda3/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

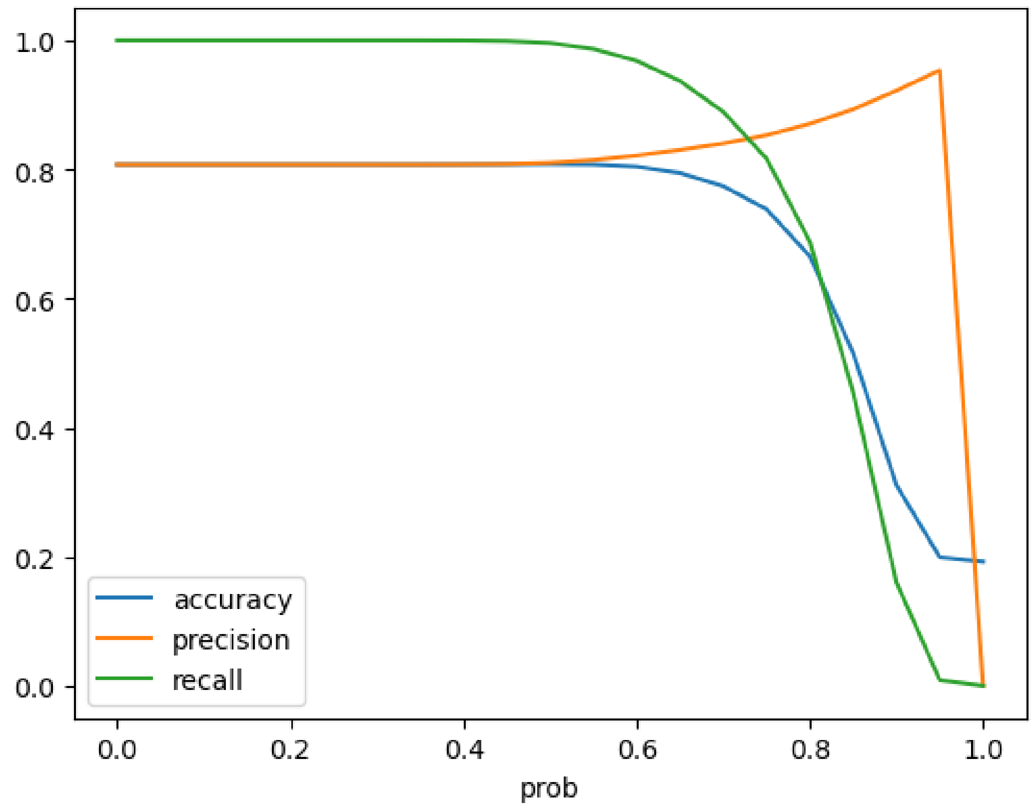
[61]:

	prob	accuracy	precision	recall	f1-score	ROC-AUC
0.00	0.00	0.807612	0.807612	1.000000	0.893568	0.500000
0.05	0.05	0.807612	0.807612	1.000000	0.893568	0.500000
0.10	0.10	0.807612	0.807612	1.000000	0.893568	0.500000
0.15	0.15	0.807612	0.807612	1.000000	0.893568	0.500000
0.20	0.20	0.807612	0.807612	1.000000	0.893568	0.500000
0.25	0.25	0.807604	0.807611	0.999989	0.893563	0.499995
0.30	0.30	0.807612	0.807618	0.999989	0.893567	0.500018
0.35	0.35	0.807612	0.807629	0.999967	0.893565	0.500053
0.40	0.40	0.807718	0.807796	0.999803	0.893602	0.500591
0.45	0.45	0.807895	0.808365	0.998949	0.893608	0.502416
0.50	0.50	0.808293	0.810288	0.995762	0.893501	0.508545
0.55	0.55	0.807807	0.814604	0.986553	0.892371	0.522006
0.60	0.60	0.804393	0.821480	0.968201	0.888826	0.542478
0.65	0.65	0.794383	0.830275	0.936928	0.880383	0.566464
0.70	0.70	0.773990	0.840257	0.889197	0.864035	0.589783
0.75	0.75	0.738298	0.853059	0.816620	0.834442	0.613068
0.80	0.80	0.665490	0.870721	0.687946	0.768617	0.629586
0.85	0.85	0.515224	0.893113	0.454087	0.602065	0.612978
0.90	0.90	0.310756	0.922107	0.160089	0.272815	0.551660
0.95	0.95	0.199135	0.953627	0.008782	0.017404	0.503495
1.00	1.00	0.192388	0.000000	0.000000	0.000000	0.500000

[62]:

```python
cut_df.plot.line(x = 'prob', y = ['accuracy', 'precision', 'recall'])
```

[62]: <Axes: xlabel='prob'>



[63]:
```
prob_precise=pd.DataFrame()
prob_precise['y_actual'] = Y_test
prob_precise['y=1|x'] = odds[:,1]
prob_precise
```

[63]:
```
        y_actual     y=1|x
63252          1  0.840809
118566         1  0.844148
58362          1  0.833767
174276         1  0.884254
369908         1  0.897064
…            …         …
222942         1  0.770109
167981         1  0.676179
365061         1  0.893769
310160         0  0.890860
220257         1  0.893458

[113079 rows x 2 columns]
```

[64]:
```
cut_precise = [float(x)/100 for x in range(70, 80)]
for i in cut_precise:
    prob_precise[i] = prob_precise['y=1|x'].map(lambda x: x>i)
prob_precise
```

[64]:
```
        y_actual     y=1|x    0.7   0.71   0.72   0.73   0.74   0.75   0.76  \
63252          1  0.840809   True   True   True   True   True   True   True
118566         1  0.844148   True   True   True   True   True   True   True
58362          1  0.833767   True   True   True   True   True   True   True
174276         1  0.884254   True   True   True   True   True   True   True
369908         1  0.897064   True   True   True   True   True   True   True
…            …         …    …     …     …     …     …     …     …
222942         1  0.770109   True   True   True   True   True   True   True
167981         1  0.676179  False  False  False  False  False  False  False
365061         1  0.893769   True   True   True   True   True   True   True
310160         0  0.890860   True   True   True   True   True   True   True
220257         1  0.893458   True   True   True   True   True   True   True

         0.77   0.78   0.79
63252    True   True   True
118566   True   True   True
58362    True   True   True
174276   True   True   True
369908   True   True   True
…       …     …     …
222942   True  False  False
167981  False  False  False
365061   True   True   True
310160   True   True   True
220257   True   True   True

[113079 rows x 12 columns]
```

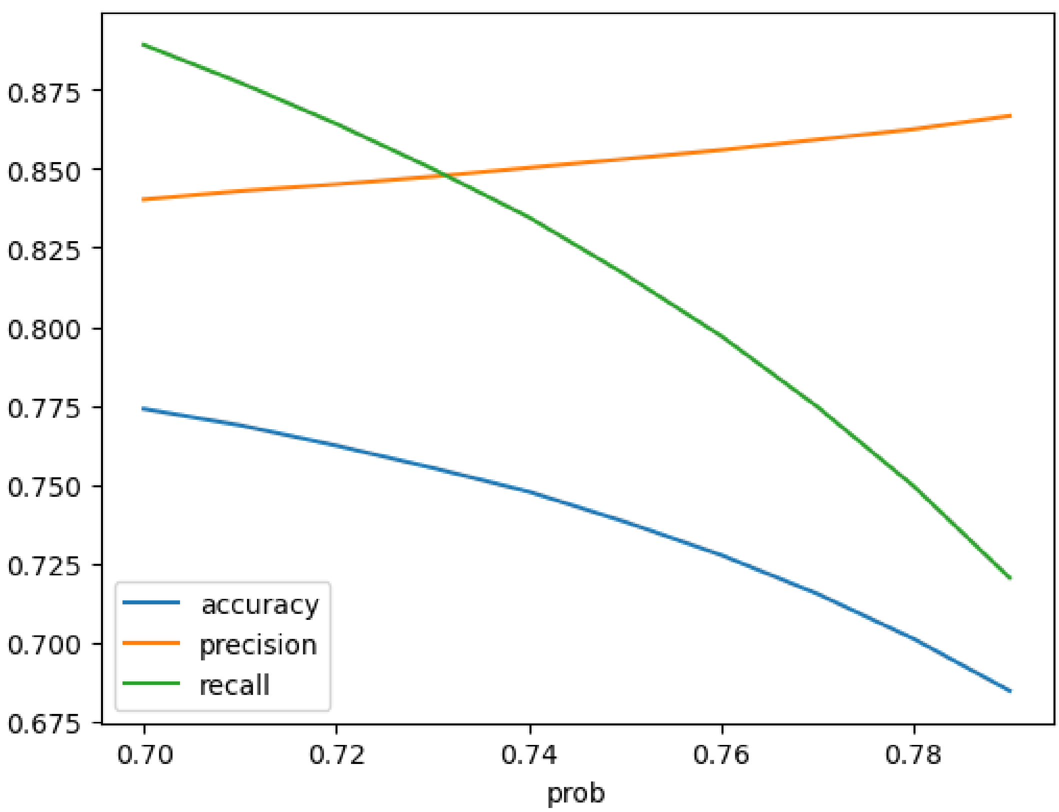
[65]:
```
cut_precise_df = pd.DataFrame(columns = ['prob', 'accuracy', 'precision',␣
↪'recall', 'f1-score', 'ROC-AUC'])

for i in cut_precise:
    accuracy = accuracy_score(prob_precise['y_actual'], prob_precise[i])
    precision = precision_score(prob_precise['y_actual'], prob_precise[i])
    recall = recall_score(prob_precise['y_actual'], prob_precise[i])
    f1 = f1_score(prob_precise['y_actual'], prob_precise[i])
    roc_auc = roc_auc_score(prob_precise['y_actual'], prob_precise[i])
    cut_precise_df.loc[i] = [i, accuracy, precision, recall, f1, roc_auc]

cut_precise_df
```

[65]:

	prob	accuracy	precision	recall	f1-score	ROC-AUC
0.70	0.70	0.773990	0.840257	0.889197	0.864035	0.589783
0.71	0.71	0.768834	0.842908	0.877261	0.859741	0.595468
0.72	0.72	0.762387	0.845067	0.864231	0.854541	0.599548
0.73	0.73	0.755348	0.847503	0.850018	0.848758	0.603979
0.74	0.74	0.747822	0.850303	0.834698	0.842429	0.608914
0.75	0.75	0.738298	0.853059	0.816620	0.834442	0.613068
0.76	0.76	0.727774	0.855918	0.797107	0.825466	0.616917
0.77	0.77	0.715517	0.859199	0.774703	0.814766	0.620884
0.78	0.78	0.701280	0.862497	0.749628	0.802111	0.623975
0.79	0.79	0.684796	0.866684	0.720544	0.786887	0.627636

[66]:
```
cut_precise_df.plot.line(x = 'prob', y = ['accuracy', 'precision', 'recall'])
```

[66]: <Axes: xlabel='prob'>



[67]:
```
y_pred = [0 if i <= 0.73 else 1 for i in prob['y=1|x']]
```

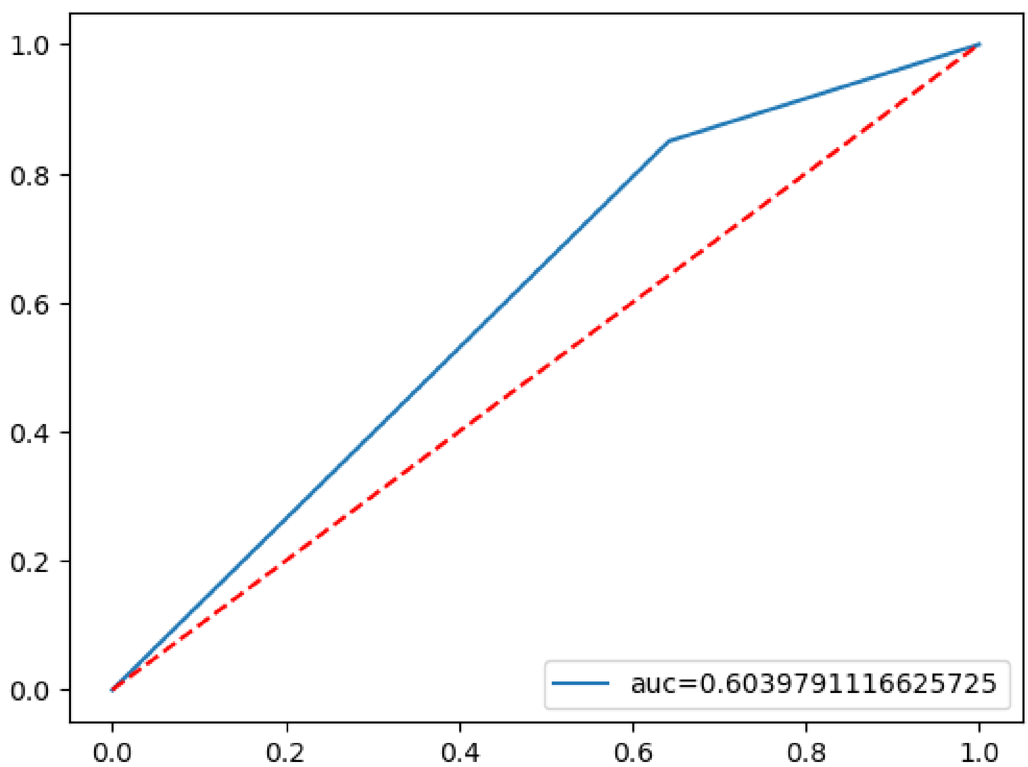
[68]:
```
print('Accuracy: ', accuracy_score(Y_test, y_pred))
print('Recall: ', recall_score(Y_test, y_pred))
print('Precision: ', precision_score(Y_test, y_pred))
```

```
print('F1-score: ', f1_score(Y_test, y_pred))
print('ROC-AUC: ', roc_auc_score(Y_test, y_pred))
```

```
Accuracy:  0.7553480310225594
Recall:  0.8500175200385441
Precision:  0.8475025929362956
F1-score:  0.8487581935173493
ROC-AUC:  0.6039791116625725
```

[69]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score

fpr, tpr, _ = roc_curve(Y_test, y_pred)
auc = roc_auc_score(Y_test, y_pred)
plt.plot(fpr, tpr, label="auc="+str(auc))
plt.plot([0,1],[0,1], 'r--')
plt.legend(loc=4)
plt.show()
```



[ ]:

[ ]: