In [1]:
```
!pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in c:\users\acer\anaconda3\lib\site-packages (0.9.1)
Requirement already satisfied: joblib>=1.0.0 in c:\users\acer\anaconda3\lib\site-packages (from imbalanced-learn) (1.1.
0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\acer\anaconda3\lib\site-packages (from imbalanced-lear
n) (2.2.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\acer\anaconda3\lib\site-packages (from imbalanced-learn) (1.2
0.3)
Requirement already satisfied: scipy>=1.3.2 in c:\users\acer\anaconda3\lib\site-packages (from imbalanced-learn) (1.7.
1)
Requirement already satisfied: scikit-learn>=1.1.0 in c:\users\acer\anaconda3\lib\site-packages (from imbalanced-learn)
(1.1.3)
```

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import precision_recall_curve
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import (
    accuracy_score, confusion_matrix, classification_report,
    roc_auc_score, roc_curve, auc,
    plot_confusion_matrix, plot_roc_curve
)
from statsmodels.stats.outliers_influence import variance_inflation_factor
from imblearn.over_sampling import SMOTE
```

- Here is the information on this particular data set:
  - loan_amnt : The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.

amount, then it will be reflected in this value.

- term : The number of payments on the loan. Values are in months and can be either 36 or 60.
- int_rate : Interest Rate on the loan
- installment : The monthly payment owed by the borrower if the loan originates.
- grade LC : assigned loan grade
- sub_grade LC : assigned loan subgrade
- emp_title : The job title supplied by the Borrower when applying for the loan.
- emp_length : Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.
- home_ownership : The home ownership status provided by the borrower during registration or obtained from the credit report. Our values are: RENT, OWN, MORTGAGE, OTHER
- annual_inc : The self-reported annual income provided by the borrower during registration.
- verification_status : Indicates if income was verified by LC, not verified, or if the income source was verified
- issue_d : The month which the loan was funded
- loan_status : Current status of the loan
- purpose : A category provided by the borrower for the loan request.
- title : The loan title provided by the borrower
- zip_code : The first 3 numbers of the zip code provided by the borrower in the loan application.
- addr_state : The state provided by the borrower in the loan application
- dti : A ratio calculated using the borrower’s total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower’s self-reported monthly income.
- earliest_cr_line : The month the borrower's earliest reported credit line was opened
- open_acc : The number of open credit lines in the borrower's credit file.
- pub_rec : Number of derogatory public records
- revol_bal : Total credit revolving balance
- revol_util : Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.
- total_acc : The total number of credit lines currently in the borrower's credit file
- initial_list_status : The initial listing status of the loan. Possible values are – W, F
- application_type : Indicates whether the loan is an individual application or a joint application with two co-borrowers
- mort_acc : Number of mortgage accounts.
- pub_rec_bankruptcies : Number of public record bankruptcies

1. Import the dataset and do usual exploratory data analysis steps like checking the structure & characteristics of the dataset

In [3]:
```
loantap = pd.read_csv(r"C:\Users\Acer\Downloads\logistic_regression.csv")
```

In [4]:
```
pd.set_option('display.max_columns', 27)

```

In [5]:
```
loantap.head()
```

Out[5]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	verification_status	issue_d
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	RENT	117000.0	Not Verified	Jan-2015
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	Not Verified	Jan-2015
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RENT	43057.0	Source Verified	Jan-2015
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	RENT	54000.0	Not Verified	Nov-2014
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55000.0	Verified	Apr-2013

In [6]:
```
data = loantap.copy()
```

In [7]:
```
# shape of the dataset - the dataset is big
data.shape
```

Out[7]: (396030, 27)

In [8]:

```
# the outcome variable or the dependent variable here would be loan status therefore checking that
data.loan_status.value_counts(normalize=True)
```

Out[8]:

```
Fully Paid     0.803871
Charged Off    0.196129
Name: loan_status, dtype: float64
```

Observation

- it shows that the data is highly imbalanced and we need to balance it in the future but only after carefully doing the initial work

In [9]:

```
# summary of the data
data.describe(include= 'all')
```

Out[9]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	verificatic
count	396030.000000	396030	396030.000000	396030.000000	396030	396030	373103	377729	396030	3.960300e+05	
unique	NaN	2	NaN	NaN	7	35	173105	11	6	NaN	
top	NaN	36 months	NaN	NaN	B	B3	Teacher	10+ years	MORTGAGE	NaN	
freq	NaN	302005	NaN	NaN	116018	26655	4389	126041	198348	NaN	
mean	14113.888089	NaN	13.639400	431.849698	NaN	NaN	NaN	NaN	NaN	7.420318e+04	
std	8357.441341	NaN	4.472157	250.727790	NaN	NaN	NaN	NaN	NaN	6.163762e+04	
min	500.000000	NaN	5.320000	16.080000	NaN	NaN	NaN	NaN	NaN	0.000000e+00	
25%	8000.000000	NaN	10.490000	250.330000	NaN	NaN	NaN	NaN	NaN	4.500000e+04	
50%	12000.000000	NaN	13.330000	375.430000	NaN	NaN	NaN	NaN	NaN	6.400000e+04	
75%	20000.000000	NaN	16.490000	567.300000	NaN	NaN	NaN	NaN	NaN	9.000000e+04	
max	40000.000000	NaN	30.990000	1533.810000	NaN	NaN	NaN	NaN	NaN	8.706582e+06	

Observation from the above and below cell

- Since this is a loan defaulter data set , the most important columns can be (int_rate, term, loan_amnt, annual_inc,pub_rec_bankruptcies , emp_title,purpose
- Nothing found on the int_rae
- term is in object type we need to extract the months from it and change to float / int
- loan_amount can have outliers and therefore the mismatches
- annual income has also difference in the mean and the 50 percent

In [10]:
```
data.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                 394275 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
```

2.Check correlation among independent variables and how they interact with each other

we can check the correlation of various features on each other to know which all features are important on preliminary analysis

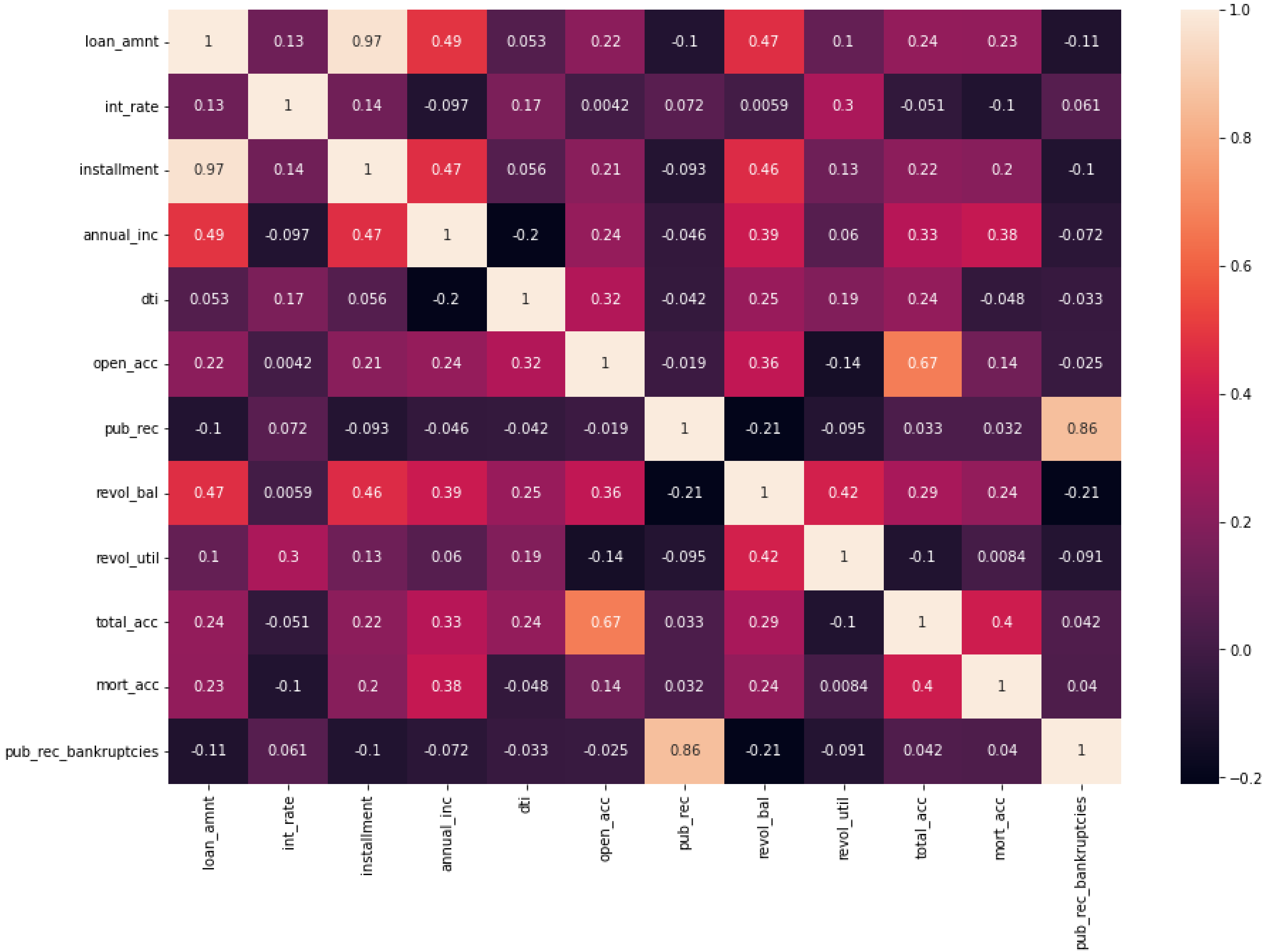
spearman vs pearson - a interview question https://towardsdatascience.com/clearly-explained-pearson-v-s-spearman-correlation-coefficient-ada2f473b8#:~:text=Comparison%20of%20Pearson%20and%20Spearman,with%20monotonic%20relationships%20as%20well (https://towardsdatascience.com/clearly-explained-pearson-v-s-spearman-correlation-coefficient-ada2f473b8#:~:text=Comparison%20of%20Pearson%20and%20Spearman,with%20monotonic%20relationships%20as%20well).

Dont know exactly if the features are linearly related or not so the safer option is to go for spearman than pearson as the person will not capture the details for non linearity

In [11]:

```
plt.figure(figsize=(15,10))
sns.heatmap(data.corr(method= 'spearman'), annot = True)
```

Out[11]: <AxesSubplot:>



Observation

- loan amount and installment are highly correlated
- pub_rec and pub_rec_bankruptcies are also very correlated but it may be the same data (need to veryfy)

Data Analysis

1 . Observation

- the number of people who have fully paid and charged off are 318357 , 77673

In [12]:
```
data['loan_status'].value_counts()
```

Out[12]:
```
Fully Paid     318357
Charged Off     77673
Name: loan_status, dtype: int64
```

Type *Markdown* and LaTeX: α^2

2. Observations

- there are only 2 terms available ie 36 months and 60 minths

In [13]:
```
data['term'].value_counts()
```

Out[13]:
```
 36 months    302005
 60 months     94025
Name: term, dtype: int64
```

In [ ]:
```

```

3 . Observations

- since there are 2 debt consolidation , there might be some issues with the naming which should be taken care of

In [14]:
```
data['title'].value_counts()
```

Out[14]:
```
Debt consolidation           152472
Credit card refinancing       51487
Home improvement              15264
Other                         12930
Debt Consolidation            11608
                              ...  
Graduation/Travel Expenses        1
Daughter's Wedding Bill           1
gotta move                        1
creditcardrefi                    1
Toxic Debt Payoff                 1
Name: title, Length: 48817, dtype: int64
```

In [15]:
```
data['title'] = data.title.str.lower()
```

In [16]:
```
data['title'].value_counts()
```

Out[16]:
```
debt consolidation                  168108
credit card refinancing              51781
home improvement                     17117
other                                12993
consolidation                         5583
                                     ...  
sweet                                    1
mortgage convertion                      1
debt consolidation and relocation        1
1 payment loan plan                      1
toxic debt payoff                        1
Name: title, Length: 41327, dtype: int64
```

4. Observation

- most people applied has house as mortgage and other are living on rented house
- looking at the capacity of repayment with these might help

In [17]:
```
data['home_ownership'].value_counts()
```

Out[17]:
```
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
ANY              3
Name: home_ownership, dtype: int64
```

5. Observation

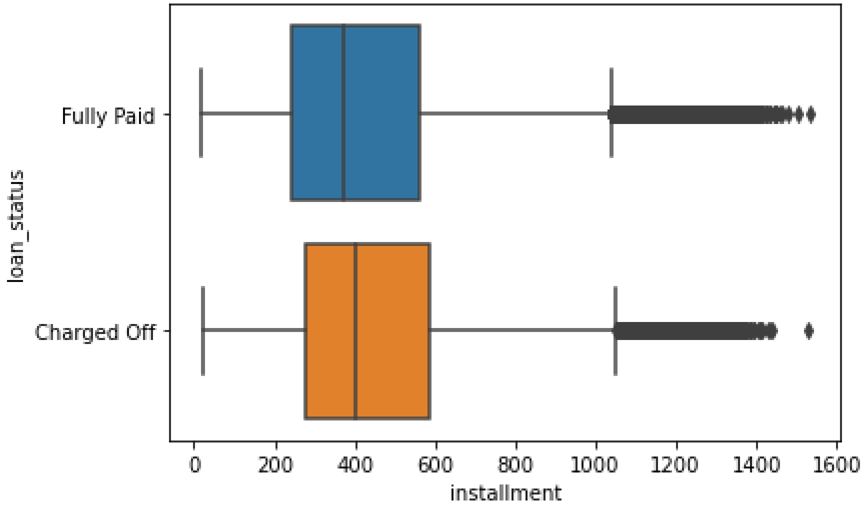
- the Grades and their repaying abilities can be tied as well

In [18]:
```
data['grade'].value_counts()
```

Out[18]:
```
B    116018
C    105987
A     64187
D     63524
E     31488
F     11772
G      3054
Name: grade, dtype: int64
```

In [19]:
```
sns.boxplot(x=data["installment"],
            y=data["loan_status"])
```

Out[19]: <AxesSubplot:xlabel='installment', ylabel='loan_status'>



In [20]:
```
stats.ttest_ind(a = data[data["loan_status"]=="Fully Paid"]["installment"],
               b = data[data["loan_status"]=="Charged Off"]["installment"])
```

Out[20]: Ttest_indResult(statistic=-25.875143861138604, pvalue=1.684401143732544e-147)

Observation

Observation

- from two sample ttest , we can observe the p-value to be < 0.05 , which is not significant ,
- hence we reject null hypothesis
- can conclude that installments for fully_paid loan status and charged_off status is not same.

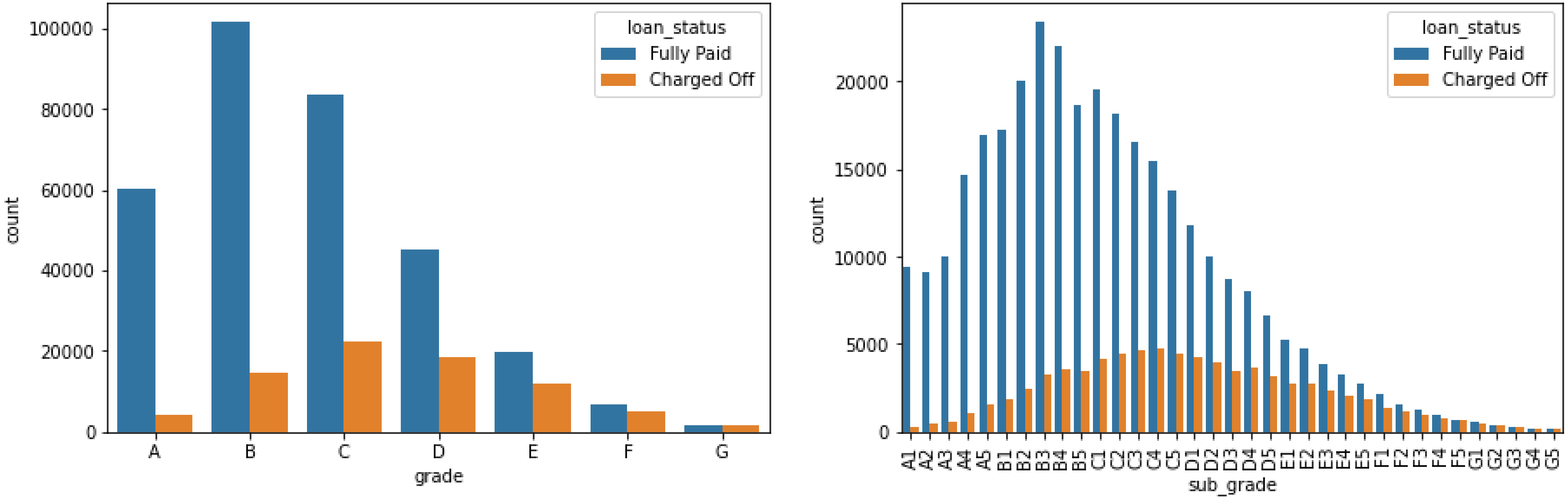
Visualisation

3. Check how much target variable (Loan_Status) depends on different predictor variables (Use count plots, box plots, heat maps etc)

1. Observation

```
- we can see that the group b has the most chance of fullly repaying the loan amount , the repaying of c is the
 worst as well
- the repaying capacity of sub group B2 and B3 is the best
```

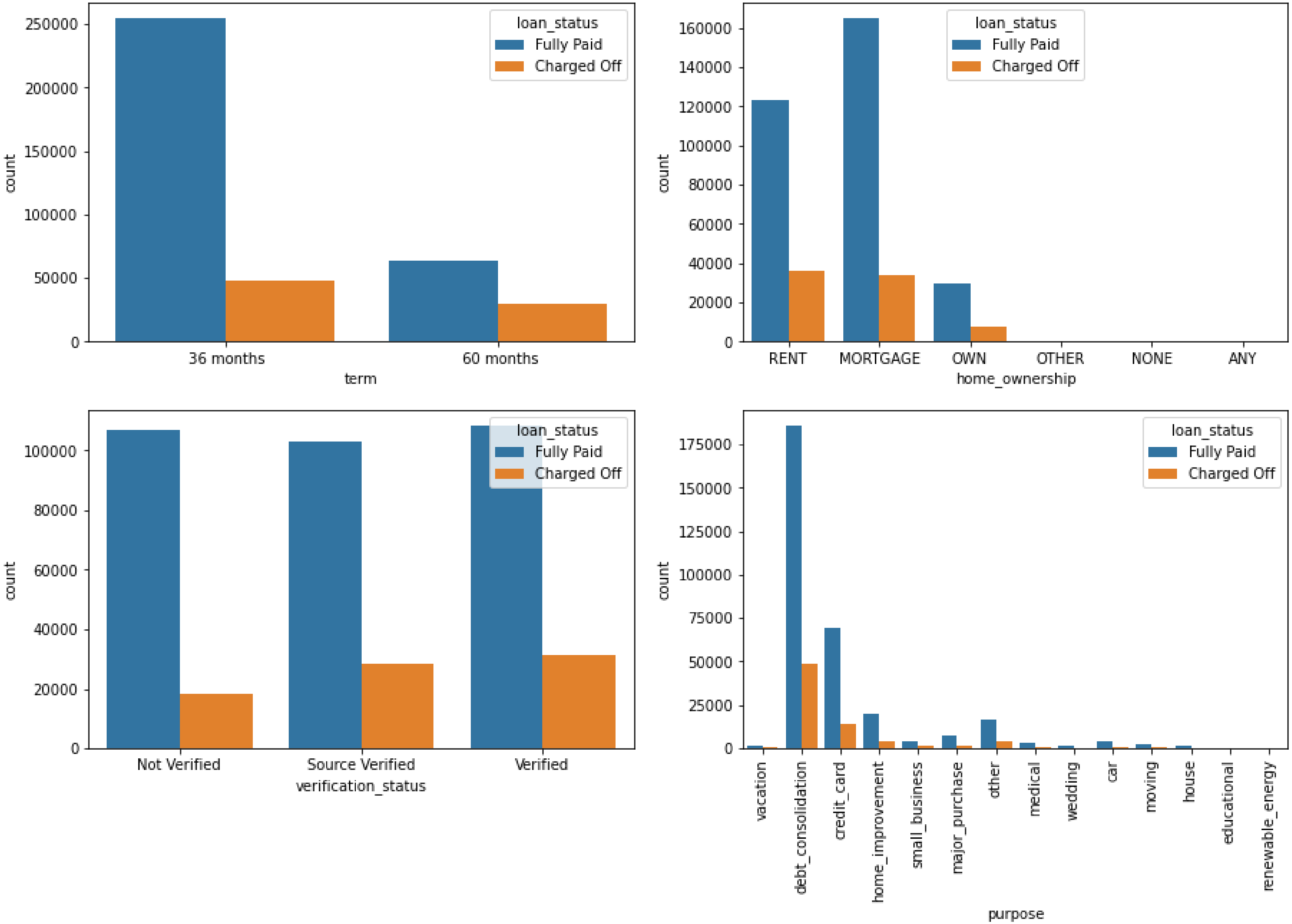
In [22]:
```
plt.figure(figsize= (15,10))

plt.subplot(2 , 2 , 1)
grade = sorted(data.grade.unique().tolist())
sns.countplot(x = 'grade' , hue = 'loan_status' , data= data , order = grade)

plt.subplot(2 , 2 , 2)

sub_grade = sorted(data.sub_grade.unique().tolist())
sns.countplot(x = 'sub_grade' , hue = 'loan_status' , data= data , order = sub_grade)
plt.xticks(rotation = 90)
plt.show()
```



In [23]:
```
# sorted(data.grade.unique().tolist())
```

In [24]:

```
plt.figure(figsize= (15 ,20))

plt.subplot(4 , 2, 1)
sns.countplot(x = 'term' , hue = 'loan_status' , data = data)

plt.subplot(4 , 2, 2)
sns.countplot(x = 'home_ownership' , hue = 'loan_status' , data = data)

plt.subplot(4 , 2, 3)
sns.countplot(x = 'verification_status' , hue = 'loan_status' , data = data)

plt.subplot(4 , 2, 4)
sns.countplot(x = 'purpose' , hue = 'loan_status' , data = data)
plt.xticks(rotation = 90)
plt.show()
```



Observation

- the term of 36 months is the most taken
- most people has owned houses in the mortgage form
- the verification has less variation in 3 categories
- debt consolidation and credit card are the most relieble source of loan approval

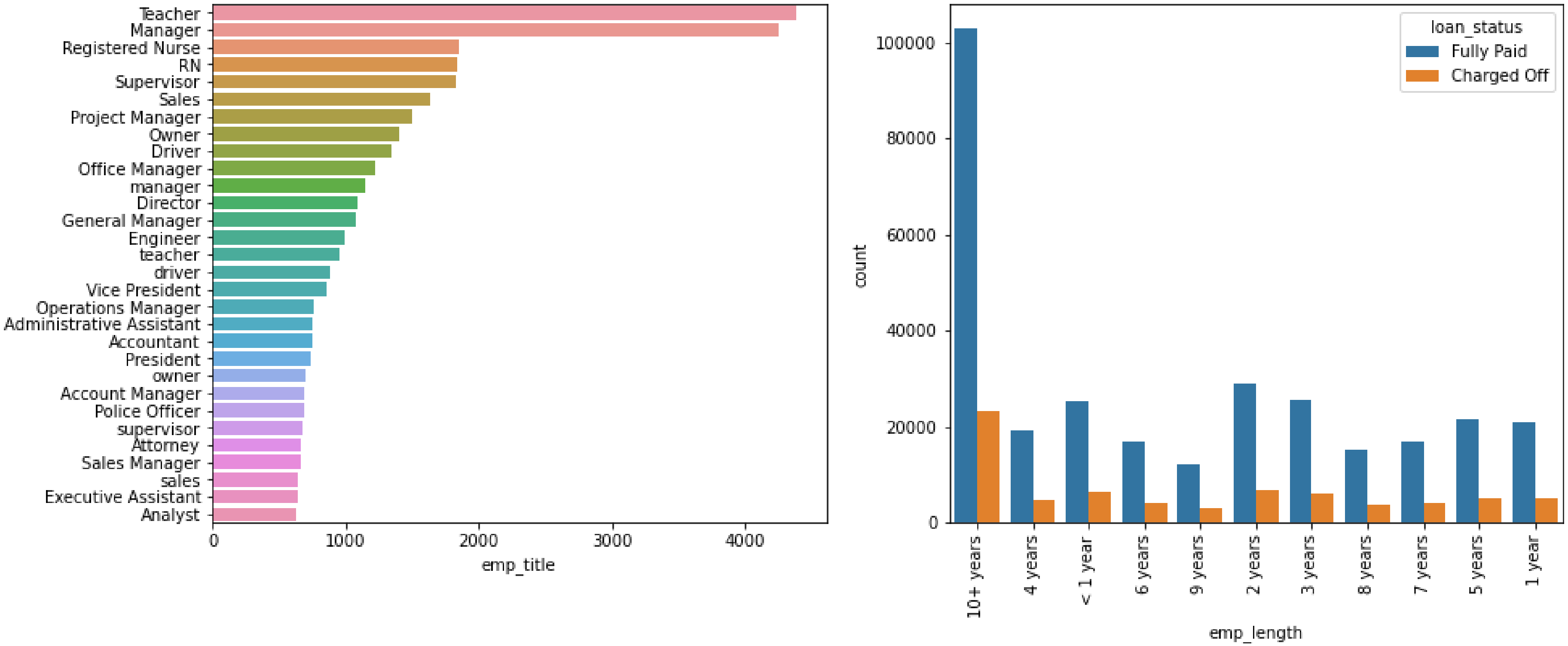
In [25]:
```
data.columns
```

Out[25]:
```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grade',
       'emp_title', 'emp_length', 'home_ownership', 'annual_inc',
       'verification_status', 'issue_d', 'loan_status', 'purpose', 'title',
       'dti', 'earliest_cr_line', 'open_acc', 'pub_rec', 'revol_bal',
       'revol_util', 'total_acc', 'initial_list_status', 'application_type',
       'mort_acc', 'pub_rec_bankruptcies', 'address'],
      dtype='object')
```

In [26]:
```
data['emp_title'].value_counts()[:30].index
```

Out[26]:
```
Index(['Teacher', 'Manager', 'Registered Nurse', 'RN', 'Supervisor', 'Sales',
       'Project Manager', 'Owner', 'Driver', 'Office Manager', 'manager',
       'Director', 'General Manager', 'Engineer', 'teacher', 'driver',
       'Vice President', 'Operations Manager', 'Administrative Assistant',
       'Accountant', 'President', 'owner', 'Account Manager', 'Police Officer',
       'supervisor', 'Attorney', 'Sales Manager', 'sales',
       'Executive Assistant', 'Analyst'],
      dtype='object')
```

In [27]:
```
plt.figure(figsize= (15,13))

plt.subplot(2 , 2, 1)
sns.barplot(y = data['emp_title'].value_counts()[:30].index , x = data['emp_title'].value_counts()[:30])

plt.subplot(2 , 2, 2)
sns.countplot(x='emp_length', data=data, hue='loan_status')
plt.xticks(rotation = 90)
plt.show()
```



In [28]:
```
stats.chi2_contingency(pd.crosstab(index = data["emp_length"],
            columns= data["loan_status"]))
```

Out[28]:
```
(122.11317384460878,
 1.88404995201913e-21,
 10,
 array([[  4976.95191526,  20905.04808474],
        [ 24236.9212716 , 101804.0787284 ],
        [  6889.31521011,  28937.68478989],
        [  6088.98780607,  25576.01219393],
        [  4605.82459912,  19346.17540088],
        [  5094.82810428,  21400.17189572],
        [  4007.59813252,  16833.40186748],
        [  4003.36766571,  16815.63233429],
        [  3685.89036055,  15482.10963945],
        [  2944.78949194,  12369.21050806],
        [  6100.52544284,  25624.47455716]]))
```

Observation

- visually there doent seems to be much correlation between employement length / emp title and loan_status.
- But from chi-sqaure test, we reject that null hypothesis and
- Hence conclude that there is a relationship exists.
- later target encoding or label encoding .

In [29]:
```
data.pub_rec.value_counts()
```

Out[29]:
```
0.0     338272
1.0      49739
2.0       5476
3.0       1521
4.0        527
5.0        237
6.0        122
7.0         56
8.0         34
9.0         12
10.0        11
11.0         8
13.0         4
12.0         4
19.0         2
40.0         1
17.0         1
86.0         1
24.0         1
15.0         1
Name: pub_rec, dtype: int64
```

In [30]:
```
data.pub_rec_bankruptcies.value_counts()
```

Out[30]:
```
0.0    350380
1.0     42790
2.0      1847
3.0       351
4.0        82
5.0        32
6.0         7
7.0         4
8.0         2
Name: pub_rec_bankruptcies, dtype: int64
```

In [31]:
```
data.mort_acc.value_counts()
```

Out[31]:
```
0.0     139777
1.0      60416
2.0      49948
3.0      38049
4.0      27887
5.0      18194
6.0      11069
7.0       6052
8.0       3121
9.0       1656
10.0       865
11.0       479
12.0       264
13.0       146
14.0       107
15.0        61
16.0        37
17.0        22
18.0        18
19.0        15
20.0        13
24.0        10
22.0         7
21.0         4
25.0         4
27.0         3
32.0         2
31.0         2
23.0         2
26.0         2
28.0         1
30.0         1
34.0         1
Name: mort_acc, dtype: int64
```

4. Simple Feature Engineering steps:

- E.g.: Creation of Flags- If value greater than 1.0 then 1 else 0. This can be done on:

- Pub_rec
- Mort_acc
- Pub_rec_bankruptcies

In [32]:

```
def pub_rec(number):
    if number == 0.0:
        return 0
    else:
        return 1

def mort_acc(number):
    if number == 0.0:
        return 0
    elif number >= 1.0:
        return 1
    else:
        return number

def pub_rec_bankruptcies(number):
    if number == 0.0:
        return 0
    elif number >= 1.0:
        return 1
    else:
        return number



# could also use a single fuction for this but for clarity i went with the above method

# def categorization(x):
#     if x >= 1:
#         return 1
#     else:
#         return 0
```

In [33]:
```
data['pub_rec'] = data.pub_rec.apply(pub_rec)
data['mort_acc'] = data.mort_acc.apply(mort_acc)
data['pub_rec_bankruptcies'] = data.pub_rec_bankruptcies.apply(pub_rec_bankruptcies)
```

In [34]:
```
print(data.pub_rec.value_counts())
print(data.pub_rec_bankruptcies.value_counts())
print(data.mort_acc.value_counts())
```

```
0    338272
1     57758
Name: pub_rec, dtype: int64
0.0    350380
1.0     45115
Name: pub_rec_bankruptcies, dtype: int64
1.0    218458
0.0    139777
Name: mort_acc, dtype: int64
```

In [35]:
```
pd.crosstab(data['loan_status'], data['pub_rec'], margins=True,normalize=True)*100
```

Out[35]:

pub_rec	0	1	All
loan_status			
Charged Off	16.498498	3.114411	19.612908
Fully Paid	68.917254	11.469838	80.387092
All	85.415751	14.584249	100.000000

In [36]:
```
pd.crosstab(data['loan_status'], data['mort_acc'], margins=True,normalize=True)*100
```

Out[36]:

mort_acc loan_status	0.0	1.0	All
Charged Off	9.255656	10.877217	20.132874
Fully Paid	29.762586	50.104540	79.867126
All	39.018242	60.981758	100.000000

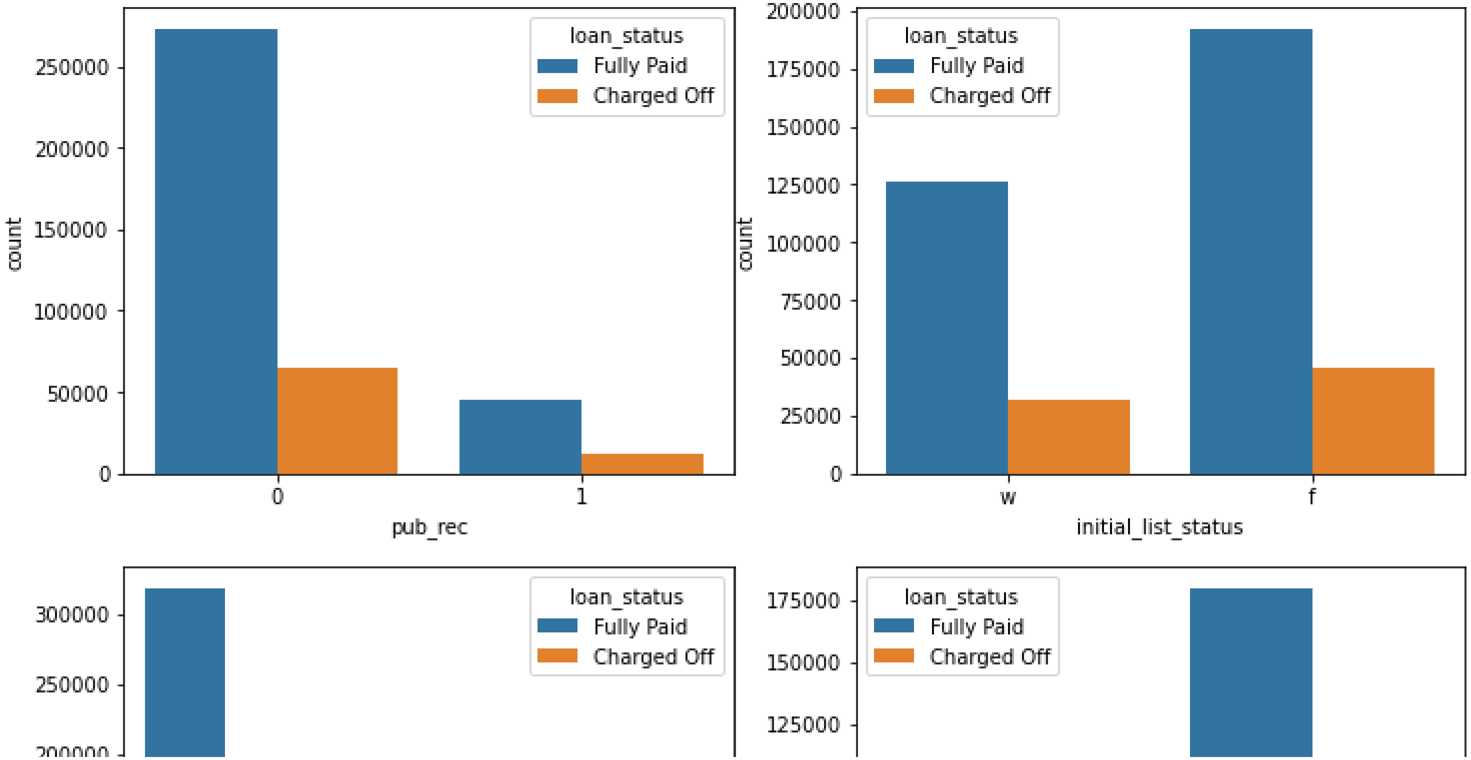
In [37]:
```
pd.crosstab(data['loan_status'], data['pub_rec_bankruptcies'], margins=True,normalize=True)*100
```

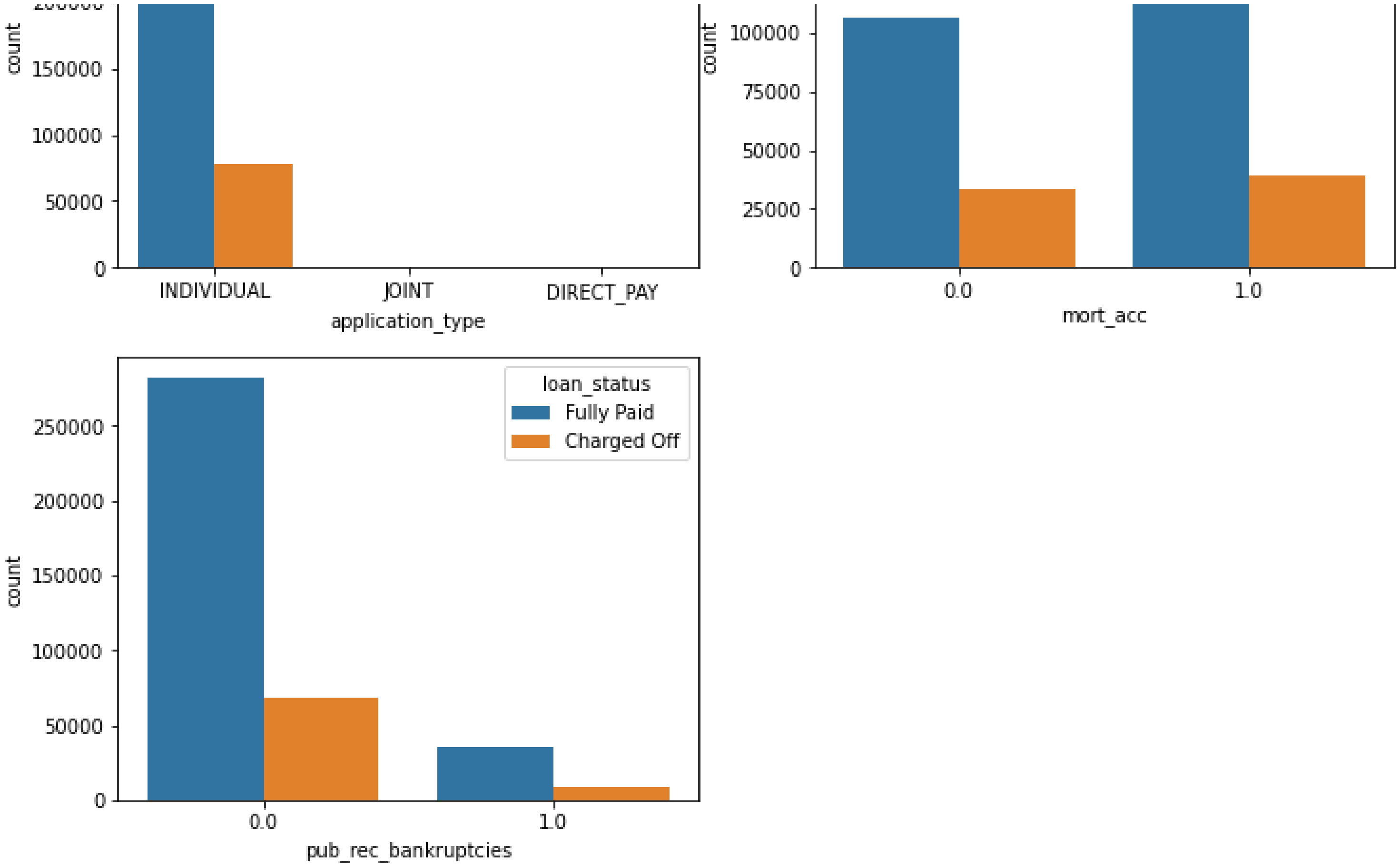
Out[37]:

pub_rec_bankruptcies loan_status	0.0	1.0	All
Charged Off	17.274808	2.342634	19.617441
Fully Paid	71.317969	9.064590	80.382559
All	88.592776	11.407224	100.000000

In [38]:

```
plt.figure(figsize=(12, 30))

plt.subplot(6, 2, 1)
sns.countplot(x='pub_rec', data=data, hue='loan_status')

plt.subplot(6, 2, 2)
sns.countplot(x='initial_list_status', data=data, hue='loan_status')

plt.subplot(6, 2, 3)
sns.countplot(x='application_type', data=data, hue='loan_status')

plt.subplot(6, 2, 4)
sns.countplot(x='mort_acc', data=data, hue='loan_status')

plt.subplot(6, 2, 5)
sns.countplot(x='pub_rec_bankruptcies', data=data, hue='loan_status')

plt.show()
```





5. Missing values and Outlier Treatment

Missing values treatment and dropiing empty

Observation there are alot of missing values in the mort_acc and we have to impute it with mean imutation

In [39]:
```
data.isnull().sum()/len(data)*100
```

Out[39]:
```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
installment             0.000000
grade                   0.000000
sub_grade               0.000000
emp_title               5.789208
emp_length              4.621115
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
title                   0.443148
dti                     0.000000
earliest_cr_line        0.000000
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.069692
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                9.543469
pub_rec_bankruptcies    0.135091
address                 0.000000
dtype: float64
```

In [40]:
```
data.mort_acc.value_counts()
```

Out[40]:
```
1.0    218458
0.0    139777
Name: mort_acc, dtype: int64
```

In [41]:
```
data.mort_acc.isnull().value_counts()
```

Out[41]:
```
False    358235
True      37795
Name: mort_acc, dtype: int64
```

Missing value treatment in mort_acc

In [42]:
```
data.groupby(by='total_acc').mean().head()
```

Out[42]:

total_acc	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_rec	revol_bal	revol_util	mort_acc	pub_rec_bankruptcie
2.0	6672.222222	15.801111	210.881667	64277.777778	2.279444	1.611111	0.000000	2860.166667	53.527778	0.000000	0.00000
3.0	6042.966361	15.615566	198.728318	41270.753884	6.502813	2.611621	0.033639	3382.807339	49.991022	0.046243	0.01548
4.0	7587.399031	15.069491	250.050194	42426.565969	8.411963	3.324717	0.033118	4874.231826	58.477400	0.062140	0.01967
5.0	7845.734714	14.917564	256.190325	44394.098003	10.118328	3.921598	0.055720	5475.253452	56.890311	0.090789	0.03918
6.0	8529.019843	14.651752	278.518228	48470.001156	11.222542	4.511119	0.076634	6546.374957	57.812483	0.121983	0.05094

In [43]:
```
data.groupby(by='total_acc').mean().mort_acc.head()
```

Out[43]:
```
total_acc
2.0    0.000000
3.0    0.046243
4.0    0.062140
5.0    0.090789
6.0    0.121983
Name: mort_acc, dtype: float64
```

In [44]:
```
mort_acc_mean = data.groupby(by='total_acc').mean().mort_acc
```

In [45]:
```
# writing a funtion to do the apply

def mort_acc_imutation (total_acc, mort_acc):

    if np.isnan(mort_acc):

        return mort_acc_mean[total_acc].round()

    else:

        return mort_acc
```

In [46]:
```
data['mort_acc'] = data.apply(lambda x: mort_acc_imutation(x['total_acc'] ,x['mort_acc'] ), axis = 1)
```

In [47]:
```
data.isnull().sum()/len(data)*100
```

Out[47]:
```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
installment             0.000000
grade                   0.000000
sub_grade               0.000000
emp_title               5.789208
emp_length              4.621115
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
title                   0.443148
dti                     0.000000
earliest_cr_line        0.000000
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.069692
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                0.000000
pub_rec_bankruptcies    0.135091
address                 0.000000
dtype: float64
```

In [48]:
```
data.shape
```

Out[48]:
```
(396030, 27)
```

In [49]:
```
data.dropna(inplace = True)
```

In [50]:
```
data.shape
```

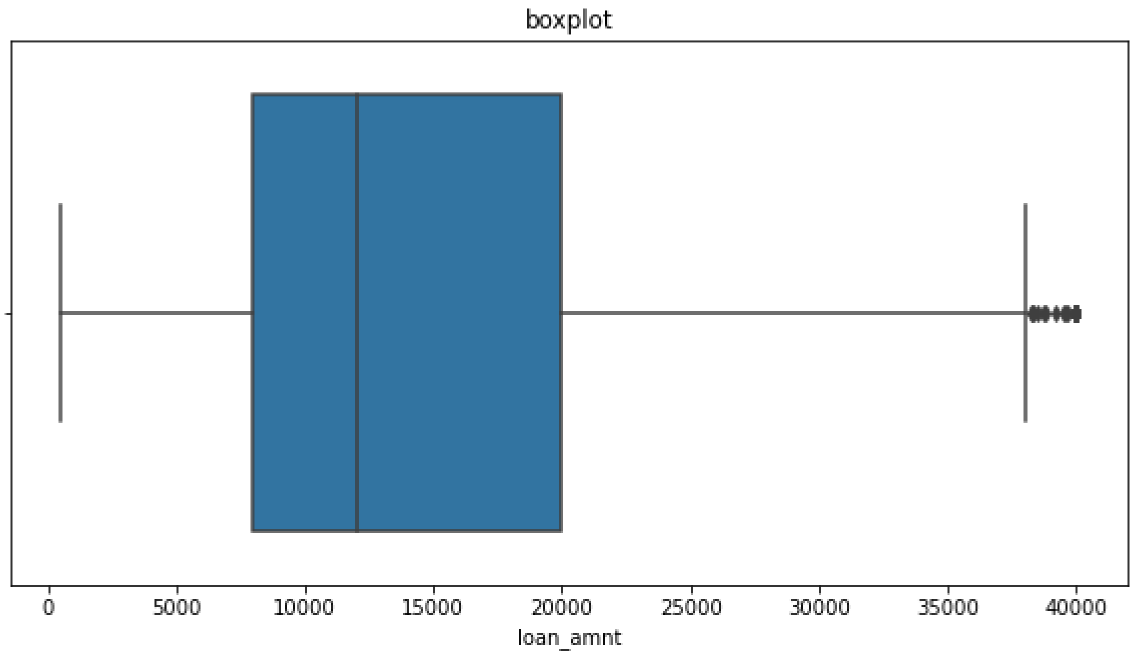
Out[50]: (370622, 27)

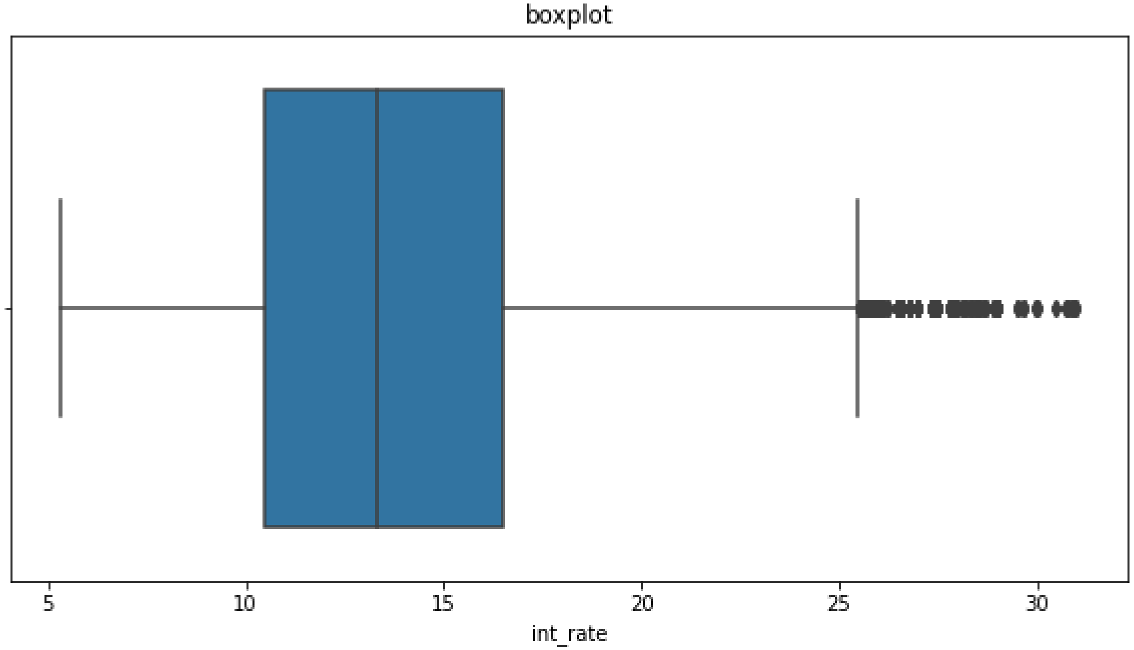
Outlier Treatment

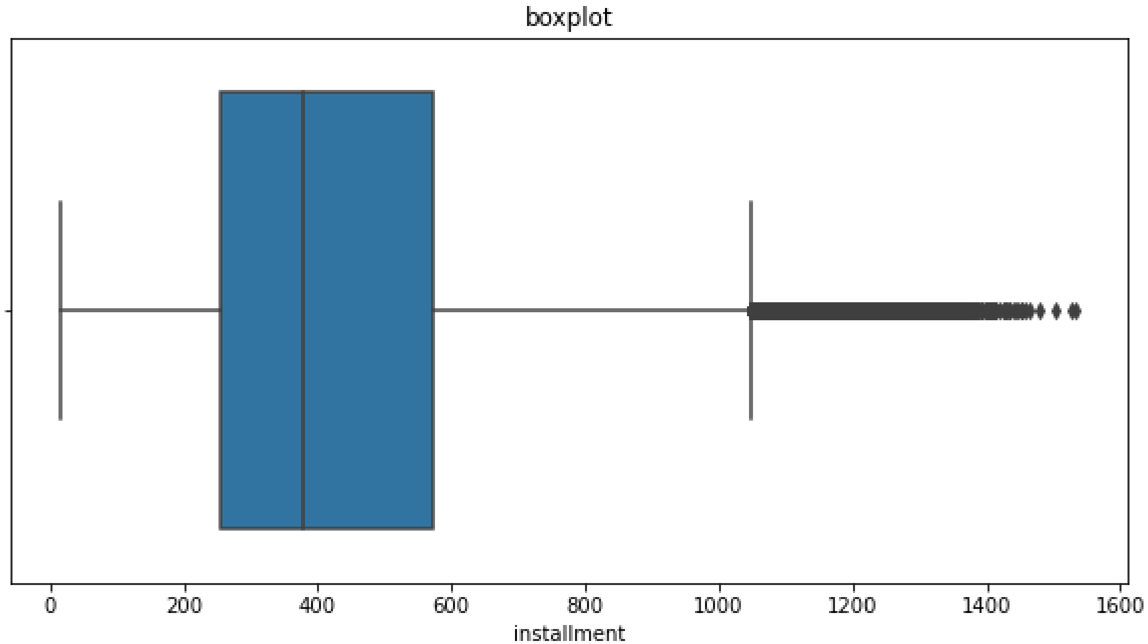
In [51]:
```
numerical_cols = data.select_dtypes(include='number')
columns = numerical_cols.columns
```

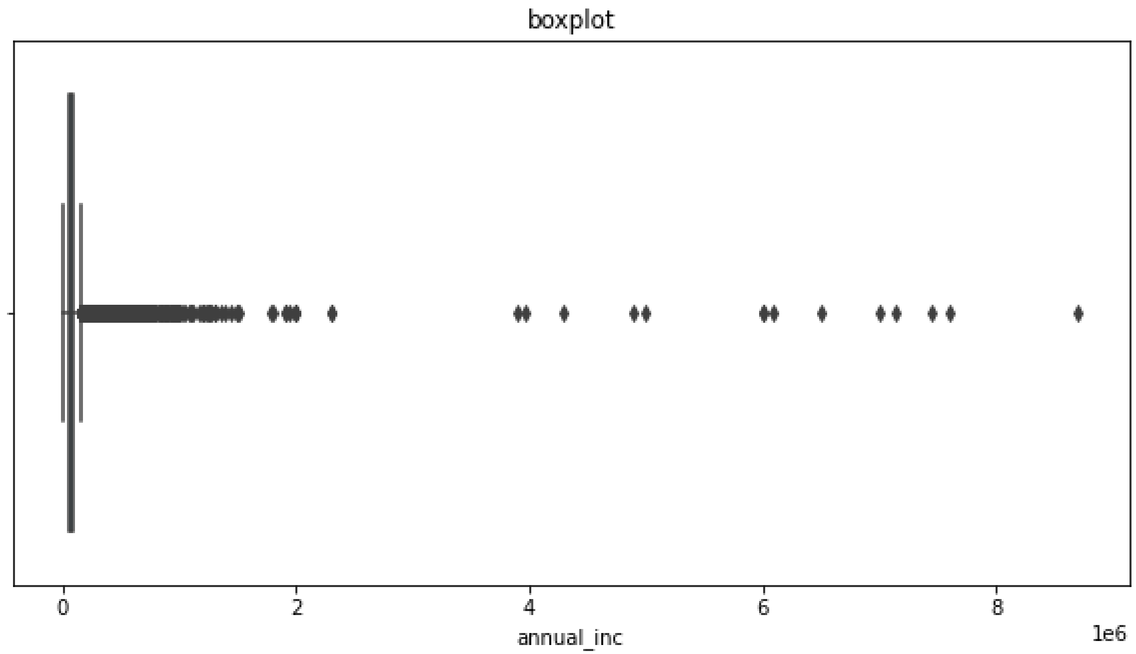
In [52]:

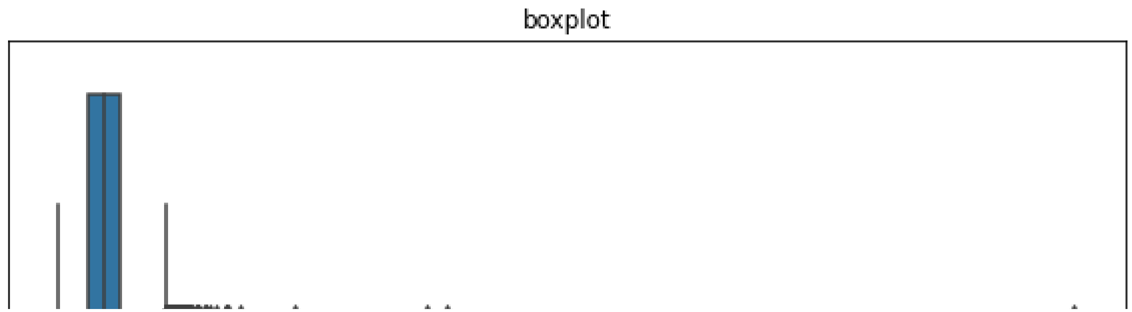
```
def box_plot(columns):
    plt.figure(figsize=(10,5))
    sns.boxplot(x = data[columns])
    plt.title('boxplot')
    plt.show()

for column in columns:
    box_plot(column)
```

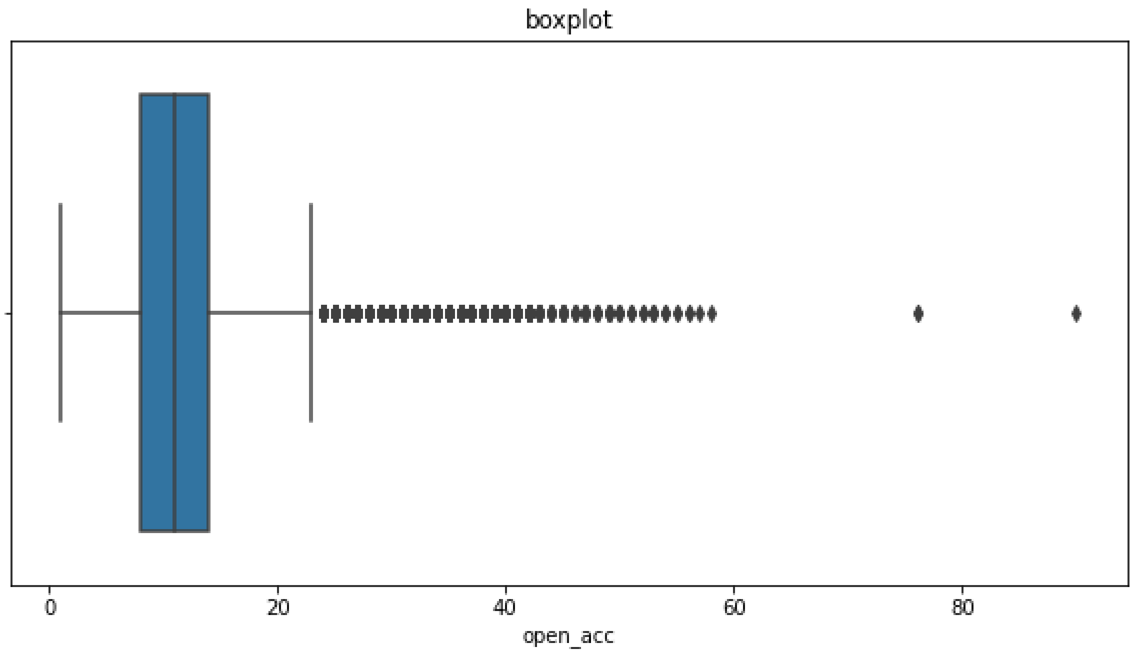


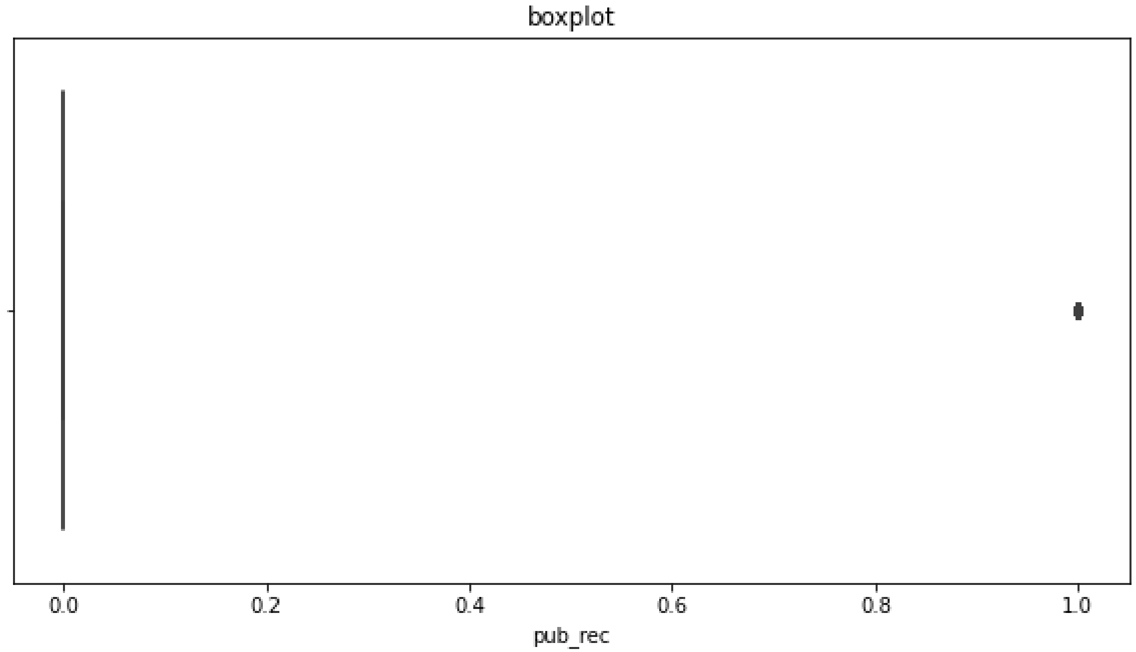

boxplot
5
10
15
20
25
30
int_rate

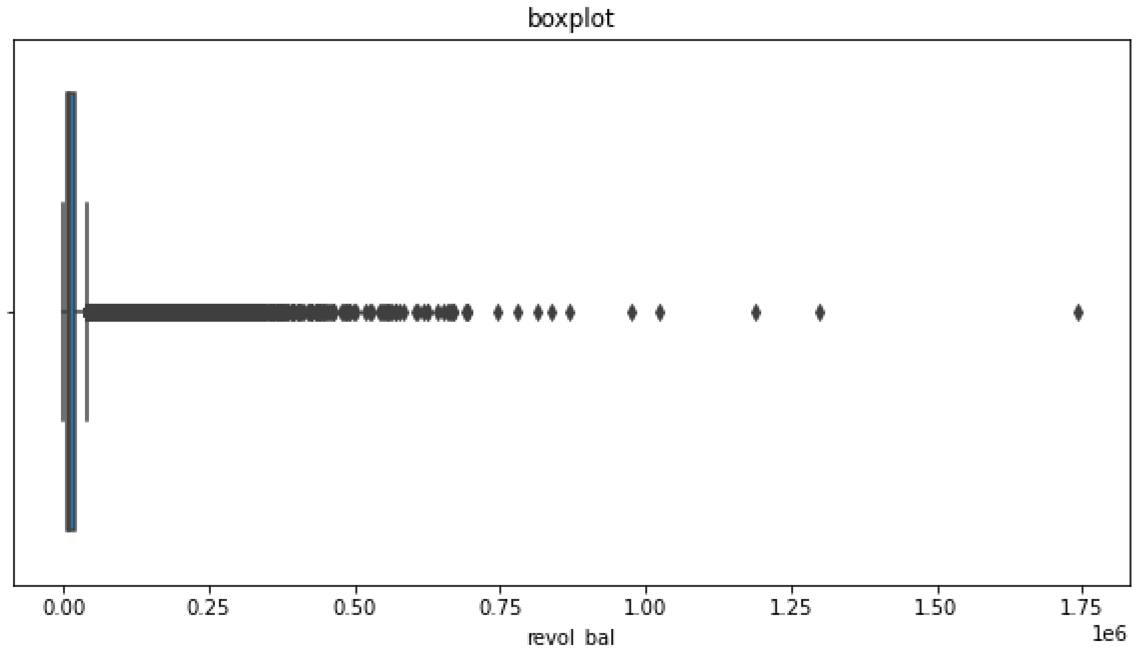

boxplot
0
200
400
600
800
1000
1200
1400
1600
installment

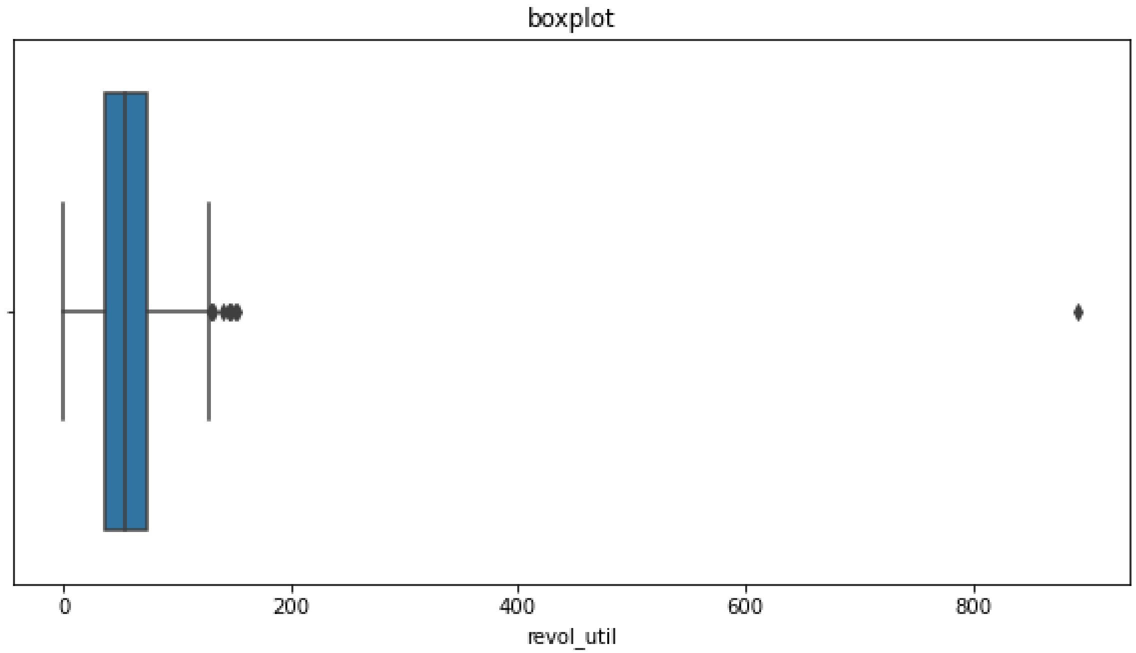

boxplot
0
2
4
6
8
1e6
annual_inc

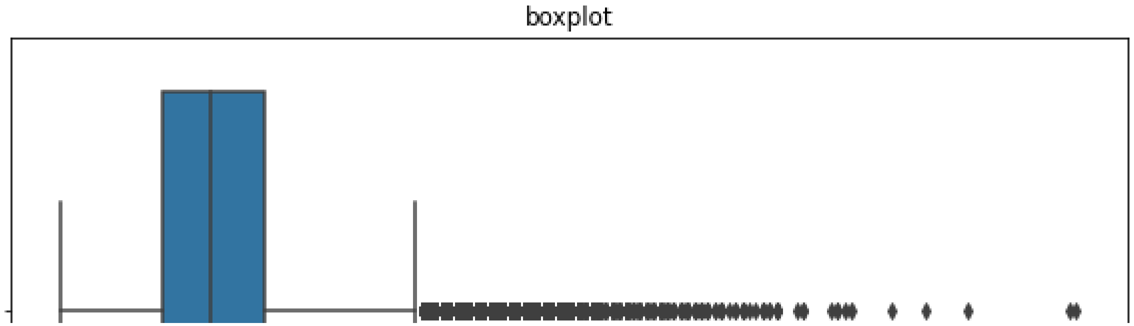

boxplot

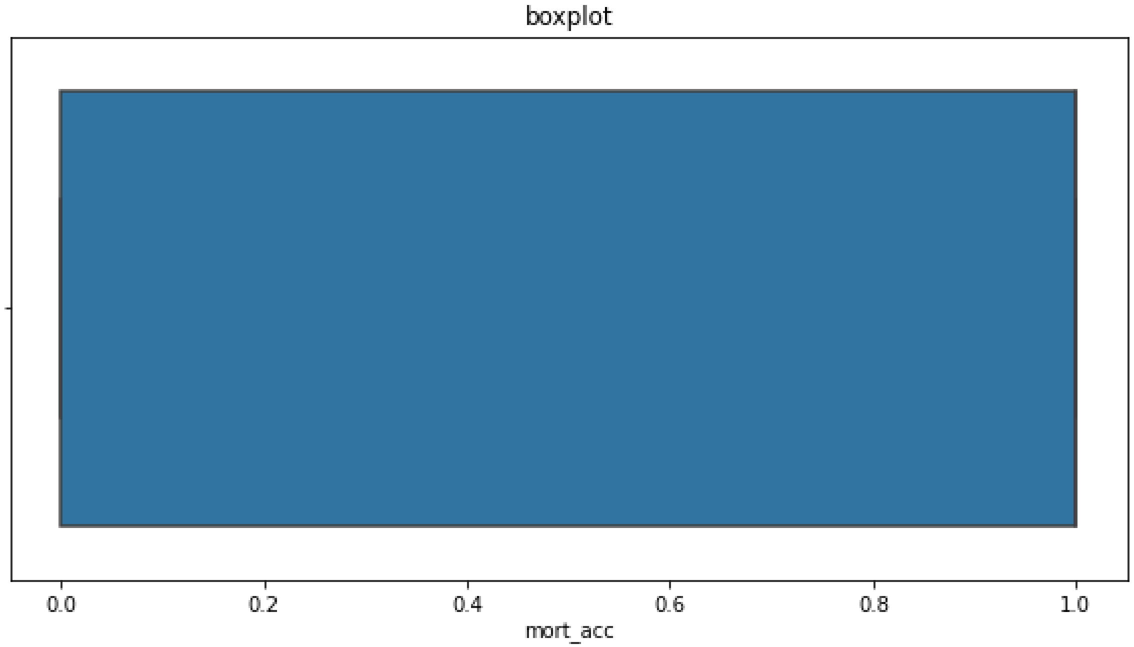

boxplot
0
20
40
60
80
open_acc

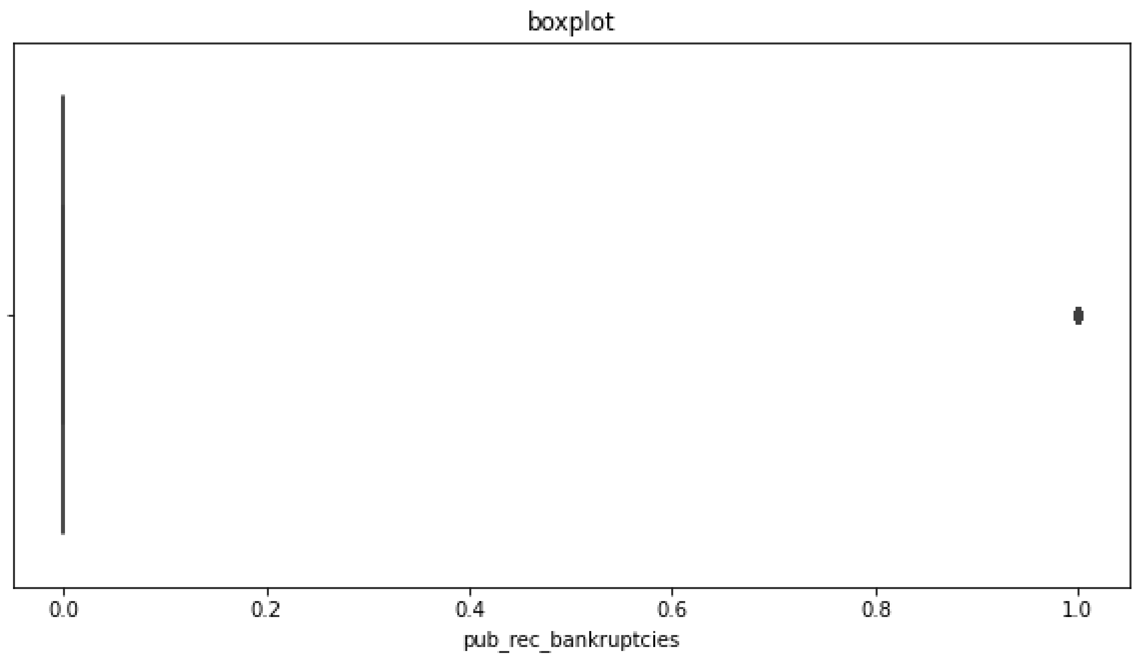

boxplot
0.0
0.2
0.4
0.6
0.8
1.0
pub_rec


boxplot
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
1e6
revol_bal


boxplot
0
200
400
600
800
revol_util


boxplot


boxplot
0.0
0.2
0.4
0.6
0.8
1.0
mort_acc



Observation

- There are alot of outliers and we can treat them with clipping them with iqr

In [53]:

```
for col in columns:
    mean = data[col].mean()
    stdv = data[col].std()

    upper_limit = mean + 3*stdv
    lower_limit = mean - 3*stdv

    data = data[(data[col] < upper_limit) & (data[col] > lower_limit)]

data.shape
```

Out[53]:

```
(350358, 27)
```

In [54]:

```
# data.drop(columns=['issue_d', 'emp_title', 'title', 'sub_grade','address', 'earliest_cr_line', 'emp_length'], axis
```

Data preparation for modeling

In [55]:

```
data
```

Out[55]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	verification_status	issu
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	RENT	117000.0	Not Verified	2
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	Not Verified	2
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RENT	43057.0	Source Verified	2
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	RENT	54000.0	Not Verified	N 2
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55000.0	Verified	2
...	...	...	...	...	...	...	...	...	...	...	...	
396025	10000.0	60 months	10.99	217.38	B	B4	licensed bankere	2 years	RENT	40000.0	Source Verified	2
396026	21000.0	36 months	12.29	700.42	C	C1	Agent	5 years	MORTGAGE	110000.0	Source Verified	F 2
396027	5000.0	36 months	9.99	161.32	B	B1	City Carrier	10+ years	RENT	56500.0	Verified	2
396028	21000.0	60 months	15.31	503.02	C	C2	Gracon Services, Inc	10+ years	MORTGAGE	64000.0	Verified	2
396029	2000.0	36 months	13.61	67.98	C	C2	Internal Revenue Service	10+ years	RENT	42996.0	Verified	2

350358 rows × 27 columns

In [56]:
```
a = '0174 Michelle Gateway\r\nMendozaberg, OK 22690'
a[-5:]
```

Out[56]: '22690'

In [57]:
```
data['zip_code'] = data['address'].apply(lambda x: x[-5:])
```

In [58]:
```
term_values = {' 36 months': 36, ' 60 months': 60}
data['term'] = data.term.map(term_values)
```

In [59]:
```
list_status = {'w': 0, 'f': 1}
data['initial_list_status'] = data.initial_list_status.map(list_status)
```

In [60]:
```
# Dropping some variables which IMO we can let go for now -
data.drop(columns=['issue_d', 'emp_title', 'title', 'sub_grade','address', 'earliest_cr_line', 'emp_length', 'instal
```

In [61]:
```
# Mapping of target variable -
data['loan_status'] = data.loan_status.map({'Fully Paid':0, 'Charged Off':1})
```

In [62]:
```
data
```

Out[62]:

	loan_amnt	term	int_rate	grade	home_ownership	annual_inc	verification_status	loan_status	purpose	dti	open_acc	pub_rec
0	10000.0	36	11.44	B	RENT	117000.0	Not Verified	0	vacation	26.24	16.0	0
1	8000.0	36	11.99	B	MORTGAGE	65000.0	Not Verified	0	debt_consolidation	22.05	17.0	0
2	15600.0	36	10.49	B	RENT	43057.0	Source Verified	0	credit_card	12.79	13.0	0
3	7200.0	36	6.49	A	RENT	54000.0	Not Verified	0	credit_card	2.60	6.0	0
4	24375.0	60	17.27	C	MORTGAGE	55000.0	Verified	1	credit_card	33.95	13.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
396025	10000.0	60	10.99	B	RENT	40000.0	Source Verified	0	debt_consolidation	15.63	6.0	0
396026	21000.0	36	12.29	C	MORTGAGE	110000.0	Source Verified	0	debt_consolidation	21.45	6.0	0
396027	5000.0	36	9.99	B	RENT	56500.0	Verified	0	debt_consolidation	17.56	15.0	0
396028	21000.0	60	15.31	C	MORTGAGE	64000.0	Verified	0	debt_consolidation	15.88	9.0	0
396029	2000.0	36	13.61	C	RENT	42996.0	Verified	0	debt_consolidation	8.32	3.0	0

350358 rows × 20 columns

One - Hot Encoding

In [63]:
```
dummies = ['purpose', 'zip_code', 'grade', 'verification_status', 'application_type', 'home_ownership']
data = pd.get_dummies(data, columns=dummies, drop_first=True)
```

In [64]:
```
# data.drop(columns=['installment'],inplace = True)
# data.head()
```

In [65]:
```
data.shape
```

Out[65]: (350358, 51)

In [66]:
```
pd.DataFrame(data.columns.tolist()).head()

```

Out[66]:

	0
0	loan_amnt
1	term
2	int_rate
3	annual_inc
4	loan_status

Model Building

In [67]:
```
X = data.drop('loan_status', axis = 1)
y = data['loan_status']
```

In [68]:
```
y.shape
```

Out[68]: (350358,)

In [69]:
```
X.shape
```

Out[69]: (350358, 50)

In [70]:
```
X_train, X_test, y_train, y_test = train_test_split(X , y, test_size=0.30 ,random_state=42 , stratify=y)

```

In [71]:
```
print(X_train.shape)
print(X_test.shape)
```

```
(245250, 50)
(105108, 50)
```

6. Scaling - Using MinMaxScaler or StandardScaler

MinMaxScaler

In [72]:
```
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

7. Use Logistic Regression Model from Sklearn/Statsmodel library and explain the results

Logistic Regression

In [73]:
```
Logistic_Regression = LogisticRegression(max_iter = 1000 )
Logistic_Regression.fit(X_train, y_train)
```

Out[73]:
```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

In [74]:
```
y_pred = Logistic_Regression.predict(X_test)
```

In [75]:
```
print('Accuracy of Logistic Regression Classifier on test set: {:.3f}'.format(Logistic_Regression.score(X_test, y_te
```

```
Accuracy of Logistic Regression Classifier on test set: 0.891
```

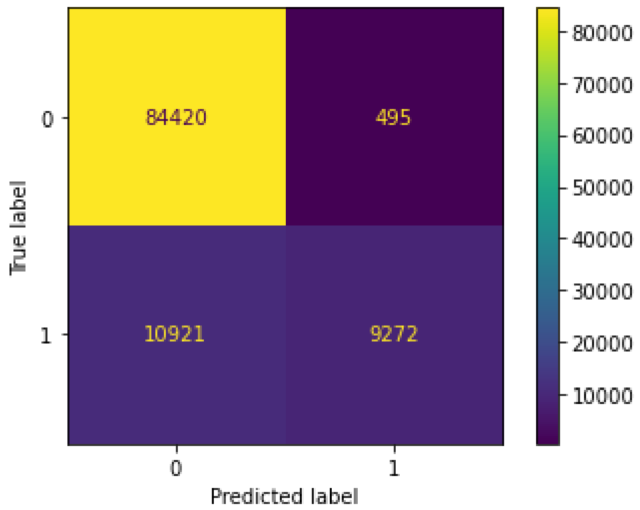
Confusion Matrix

In [76]:
```
confusion_matrix1 = confusion_matrix(y_test, y_pred)
print(confusion_matrix1)
```

```
[[84420   495]
 [10921  9272]]
```

In [77]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

In [78]:
```
ConfusionMatrixDisplay(confusion_matrix1).plot()
plt.show()
```



Results Evaluation:

- Classification Report
- ROC AUC curve
- Precision recall curve

Classification Report

In [79]:
```
from sklearn.metrics import f1_score
f1_score(y_test, y_pred)
```

Out[79]:
```
0.618958611481976
```

In [80]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.99      0.94     84915
           1       0.95      0.46      0.62     20193

    accuracy                           0.89    105108
   macro avg       0.92      0.73      0.78    105108
weighted avg       0.90      0.89      0.88    105108
```

ROC AUC curve

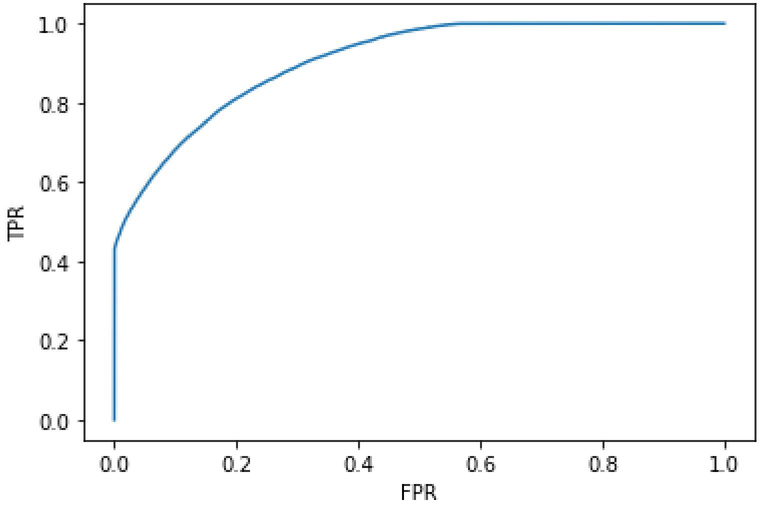
In [81]:
```
Logistic_Regression.predict_proba(X_test)
```

Out[81]:
```
array([[7.52902703e-04, 9.99247097e-01],
       [8.09111591e-01, 1.90888409e-01],
       [7.65894095e-01, 2.34105905e-01],
       ...,
       [7.18296476e-01, 2.81703524e-01],
       [9.99964620e-01, 3.53804188e-05],
       [9.28558189e-04, 9.99071442e-01]])
```

In [82]:
```
Logistic_Regression.predict_proba(X_test)[:,1]
```

Out[82]:
```
array([9.99247097e-01, 1.90888409e-01, 2.34105905e-01, ...,
       2.81703524e-01, 3.53804188e-05, 9.99071442e-01])
```

In [83]:
```
fpr, tpr, thres = roc_curve(y_test, Logistic_Regression.predict_proba(X_test)[:,1])
plt.plot(fpr,tpr)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```



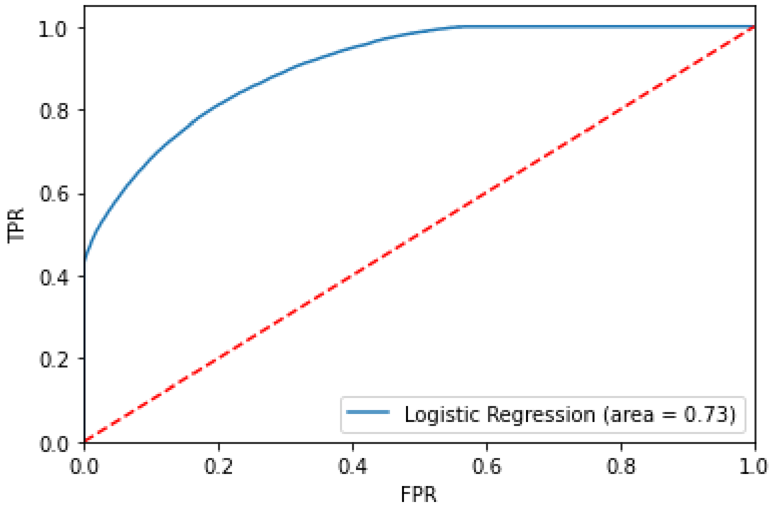
In [84]:
```
roc_aucc = roc_auc_score(y_test, Logistic_Regression.predict(X_test))
roc_aucc
```

Out[84]: 0.7266698300982168

In [85]:
```
auc(fpr , tpr)
```

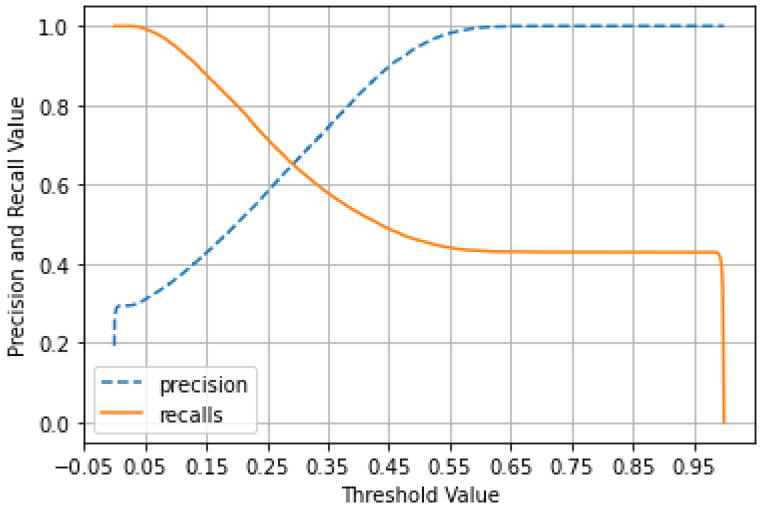
Out[85]: 0.9068505917250823

In [86]:
```
fpr, tpr, thres = roc_curve(y_test, Logistic_Regression.predict_proba(X_test)[:,1])
plt.plot(fpr,tpr , label = 'Logistic Regression (area = %0.2f)' % roc_aucc)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.plot([0, 1], [0, 1],'r--')
plt.legend(loc="lower right")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.show()
```



In [87]:

```
def precision_recall_curve_plot(y_test, pred_prob):
    precisions, recalls, thresholds = precision_recall_curve(y_test, pred_prob)

    threshold_boundary = thresholds.shape[0]
    # plot precision
    plt.plot(thresholds, precisions[0:threshold_boundary], linestyle='--', label='precision')
    # plot recall
    plt.plot(thresholds, recalls[0:threshold_boundary], label='recalls')

    start, end = plt.xlim()
    plt.xticks(np.round(np.arange(start, end, 0.1), 2))

    plt.xlabel('Threshold Value'); plt.ylabel('Precision and Recall Value')
    plt.legend(); plt.grid()
    plt.show()

precision_recall_curve_plot(y_test, Logistic_Regression.predict_proba(X_test)[:,1])
```



In [88]:
```
feature_importance = pd.DataFrame(index = data.drop(["loan_status"], axis = 1).columns, data = Logistic_Regression.c
feature_importance
```

Out[88]:

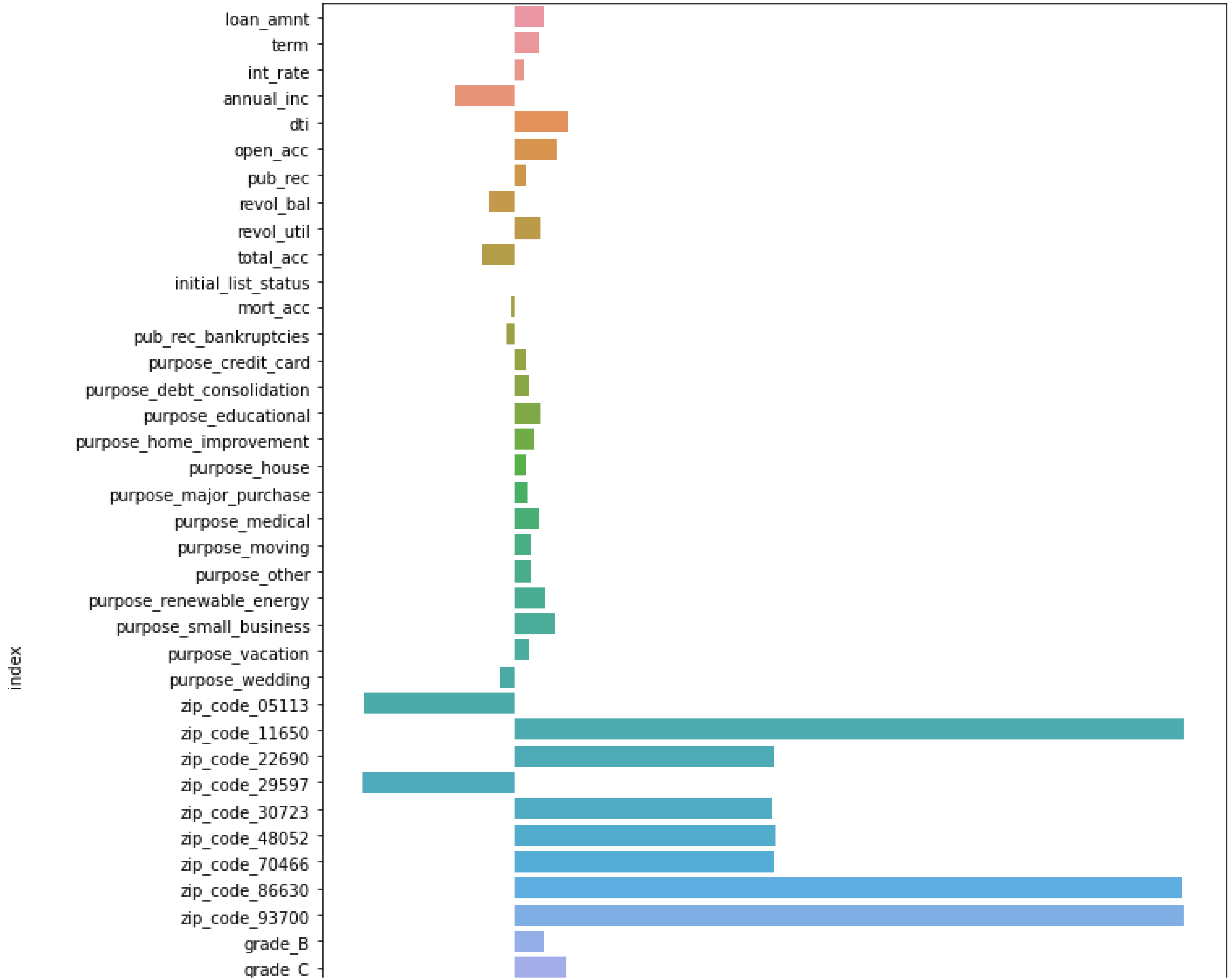
	index	0
0	loan_amnt	0.526715
1	term	0.439817
2	int_rate	0.176492
3	annual_inc	-1.131181
4	dti	1.006044
5	open_acc	0.772007
6	pub_rec	0.199148
7	revol_bal	-0.497641
8	revol_util	0.474739
9	total_acc	-0.608431
10	initial_list_status	-0.018814
11	mort_acc	-0.060903
12	pub_rec_bankruptcies	-0.164527
13	purpose_credit_card	0.190263
14	purpose_debt_consolidation	0.271745
15	purpose_educational	0.482358
16	purpose_home_improvement	0.348791
17	purpose_house	0.191178
18	purpose_major_purchase	0.231802
19	purpose_medical	0.455690
20	purpose_moving	0.308084
21	purpose_other	0.287856

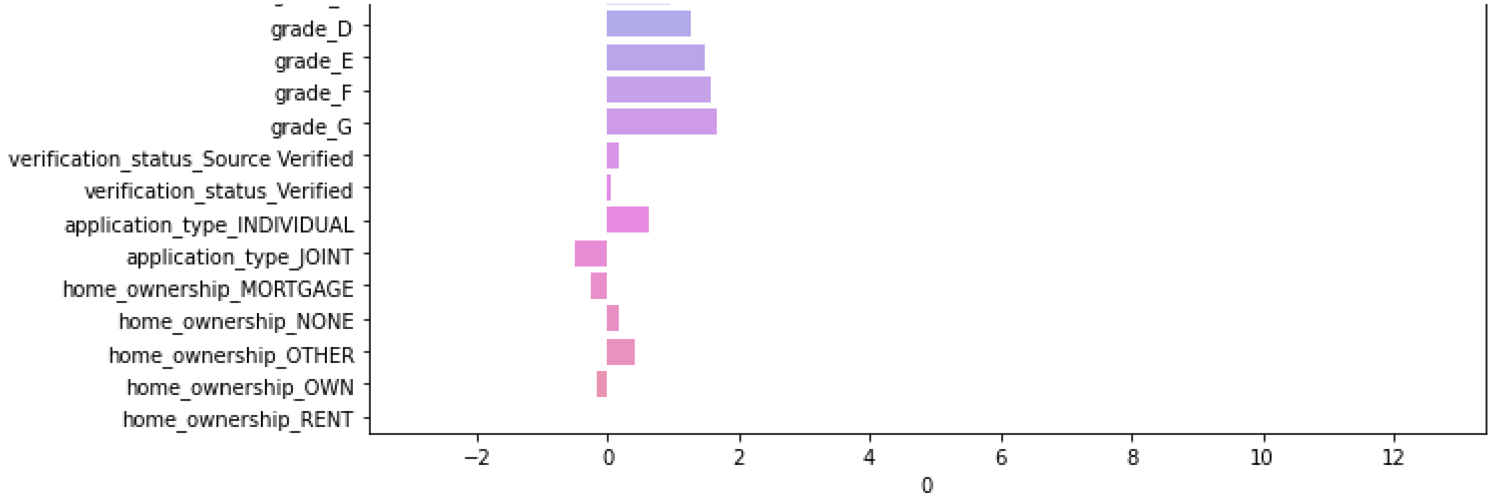
	index	0
22	purpose_renewable_energy	0.568821
23	purpose_small_business	0.743433
24	purpose_vacation	0.254862
25	purpose_wedding	-0.277783
26	zip_code_05113	-2.866603
27	zip_code_11650	12.621129
28	zip_code_22690	4.868247
29	zip_code_29597	-2.880241
30	zip_code_30723	4.853698
31	zip_code_48052	4.901988
32	zip_code_70466	4.884385
33	zip_code_86630	12.594262
34	zip_code_93700	12.620698
35	grade_B	0.535539
36	grade_C	0.973280
37	grade_D	1.266691
38	grade_E	1.474947
39	grade_F	1.583204
40	grade_G	1.677626
41	verification_status_Source Verified	0.185873
42	verification_status_Verified	0.051172
43	application_type_INDIVIDUAL	0.618752
44	application_type_JOINT	-0.490306
45	home_ownership_MORTGAGE	-0.267221
46	home_ownership_NONE	0.164499
47	home_ownership_OTHER	0.427804

	index	0
48	home_ownership_OWN	-0.162157
49	home_ownership_RENT	-0.009303

In [109]:
```
plt.figure(figsize=(10,15))
sns.barplot(y = feature_importance["index"],
            x =  feature_importance[0])
```

Out[109]: <AxesSubplot:xlabel='0', ylabel='index'>





Checking for VIF

In [89]:
```
# for i in range(X.shape[1]):
#     print(i)
```

In [90]:
```
vif = []
for i in range(X_train.shape[1]):
    vif.append(variance_inflation_factor(exog=X_train , exog_idx= i))

vif
```

Out[90]:
```
[7.201411631614606,
 2.1460042427041914,
 52.77981035661532,
 7.53553121880976,
 8.099960313456863,
 11.81108077547681,
 4.886026714377482,
 4.778397833163139,
 9.516873046073286,
 11.015366480352249,
 2.6763459019713114,
 4.724802442941753,
 4.679614090955052,
 18.63336968116256,
 51.26643841198906,
 1.0516601539907633,
 5.8723424315630215,
 1.460902718499696,
 2.8348964615183463,
 1.8618020326341655,
 1.6023123739108234,
 5.411199421804762,
 1.0722552424720642,
 2.0430199017344104,
 1.5289945727770815,
 1.40913587060726,
 1.9836624412059118,
 1.2543923793791707,
 2.2295268114925997,
 1.9947609800126551,
 2.2272336000985313,
 2.2155102589985316,
 2.236695933848133,
 1.2524034592035047,
 1.2531604097822524,
```

```
5.455127753228361,
10.287579338527907,
11.46399922077249,
9.212069142679699,
5.814902785770537,
2.2032671290287817,
2.175461487189248,
2.3239780973056354,
4911.7467640177265,
4.466915434469589,
2487.105698032772,
1.3701012047156547,
2.435347355588868,
451.1477021458173,
2092.3280660917585]
```

In [91]:
```
len(vif)
```

Out[91]: 50

In [92]:
```
len(X.columns)
```

Out[92]: 50

In [ ]:
```

```

In [93]:
```
VIF = pd.DataFrame({'Coeff_name :' : X.columns,
                    'VIFs' : np.round(vif,2)})
VIF = VIF.sort_values(by='VIFs', ascending = False)
VIF
```

Out[93]:

	Coeff_name :	VIFs
43	application_type_INDIVIDUAL	4911.75
45	home_ownership_MORTGAGE	2487.11
49	home_ownership_RENT	2092.33
48	home_ownership_OWN	451.15
2	int_rate	52.78
14	purpose_debt_consolidation	51.27
13	purpose_credit_card	18.63
5	open_acc	11.81
37	grade_D	11.46
9	total_acc	11.02
36	grade_C	10.29
8	revol_util	9.52
38	grade_E	9.21
4	dti	8.10
3	annual_inc	7.54
0	loan_amnt	7.20
16	purpose_home_improvement	5.87
39	grade_F	5.81
35	grade_B	5.46
21	purpose_other	5.41
6	pub_rec	4.89
7	revol_bal	4.78

	Coeff_name :	VIFs
11	mort_acc	4.72
12	pub_rec_bankruptcies	4.68
44	application_type_JOINT	4.47
18	purpose_major_purchase	2.83
10	initial_list_status	2.68
47	home_ownership_OTHER	2.44
42	verification_status_Verified	2.32
32	zip_code_70466	2.24
28	zip_code_22690	2.23
30	zip_code_30723	2.23
31	zip_code_48052	2.22
40	grade_G	2.20
41	verification_status_Source Verified	2.18
1	term	2.15
23	purpose_small_business	2.04
29	zip_code_29597	1.99
26	zip_code_05113	1.98
19	purpose_medical	1.86
20	purpose_moving	1.60
24	purpose_vacation	1.53
17	purpose_house	1.46
25	purpose_wedding	1.41
46	home_ownership_NONE	1.37
34	zip_code_93700	1.25
27	zip_code_11650	1.25
33	zip_code_86630	1.25

	Coeff_name :	VIFs
22	purpose_renewable_energy	1.07
15	purpose_educational	1.05

In [94]:
```
# def Calc_VIF(X):
#     vif = pd.DataFrame()
#     vif['features'] = X.columns
#     vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
#     vif['VIF'] = np.round(vif[VIF],2)
#     vif = vif.sort_values(by='VIF', ascending = False)
#     return vif
```

In [95]:
```
# def calc_vif(X):
#     # Calculating the VIF
#     vif = pd.DataFrame()
#     vif['Feature'] = X.columns
#     vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
#     vif['VIF'] = round(vif['VIF'], 2)
#     vif = vif.sort_values(by='VIF', ascending = False)
#     return vif

# calc_vif(X)
```

In [96]:
```
X.drop(columns=['home_ownership_MORTGAGE' , 'home_ownership_RENT', 'home_ownership_OWN', 'purpose_debt_consolidation
```

In [97]:
```
X = scaler.fit_transform(X)

kfold = KFold(n_splits=5)
accuracy = np.mean(cross_val_score(Logistic_Regression, X, y, cv=kfold, scoring='accuracy', n_jobs=-1))
print("Cross Validation accuracy: {:.3f}".format(accuracy))
```

```
Cross Validation accuracy: 0.891
```

In [98]:
```
y_train.value_counts()
```

Out[98]:
```
0    198134
1     47116
Name: loan_status, dtype: int64
```

Imbalance Data - with Data Imputation (Strategy # 2) [Oversampling vs Undersampling vs SMOTE]

- Undersampling
  - Selecting majority class in equal proportion to minority class
  - Will reduce data points of majority class that causes information loss
  - Hence not a best strategy , specially when we've rich large sample available
- Oversampling
  - Replicating the samples of the -ve labels such that it becomes almost same as the +ve labels
  - It will cause fabrication of data , which will tend to overfitted model
- SMOTE
  - In oversampling, we are simply repeating the data
  - But using SMOTE we are synthetically creating new data
  - Second best strategy to deal with imbalance data

Smote

Since the churn vs Non Churn values is highly imbalanced we can use SMOTE to see if we can increase the REcall and overall score

In [99]:
```
sm = SMOTE(random_state=42)
X_train_sm , y_train_sm = sm.fit_resample(X_train , y_train.ravel())
```

In [100]:
```
X_train_sm.shape
```

Out[100]:
```
(396268, 50)
```

In [101]:
```
y_train_sm.shape
```
Out[101]:
```
(396268,)
```

In [102]:
```
sum(y_train_sm == 1)
```
Out[102]:
```
198134
```

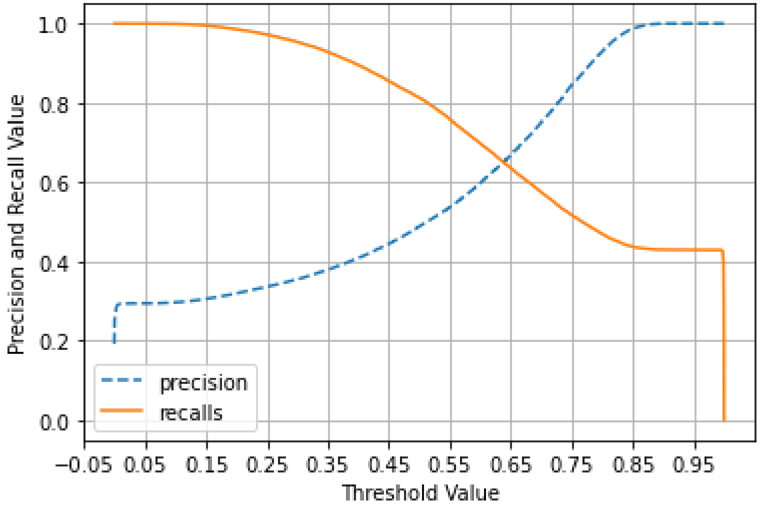
In [103]:
```
sum(y_train_sm == 0)
```
Out[103]:
```
198134
```

In [104]:
```
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train_sm, y_train_sm)
predictions = log_reg.predict(X_test)


```

In [105]:
```
# Classification Report
print(classification_report(y_test, predictions))
```
```
              precision    recall  f1-score   support

           0       0.95      0.80      0.87     84915
           1       0.49      0.81      0.61     20193

    accuracy                           0.80    105108
   macro avg       0.72      0.81      0.74    105108
weighted avg       0.86      0.80      0.82    105108
```

In [106]:

```
def precision_recall_curve_plot(y_test, pred_prob):
    precisions, recalls, thresholds = precision_recall_curve(y_test, pred_prob)

    threshold_boundary = thresholds.shape[0]
    # plot precision
    plt.plot(thresholds, precisions[0:threshold_boundary], linestyle='--', label='precision')
    # plot recall
    plt.plot(thresholds, recalls[0:threshold_boundary], label='recalls')

    start, end = plt.xlim()
    plt.xticks(np.round(np.arange(start, end, 0.1), 2))

    plt.xlabel('Threshold Value'); plt.ylabel('Precision and Recall Value')
    plt.legend(); plt.grid()
    plt.show()

precision_recall_curve_plot(y_test, log_reg.predict_proba(X_test)[:,1])
```



In [112]:
```
# from sklearn.metrics import PrecisionRecallDisplay
# display = PrecisionRecallDisplay.from_estimator(log_reg, X_test, y_test, name="LinearSVC")
# _ = display.ax_.set_title("2-class Precision-Recall curve")
```

Observations

- Observation 1
  - Since this is a loan defaulter data set , the most important columns can be (int_rate, term, loan_amnt, annual_inc,pub_rec_bankruptcies , emp_title,purpose
  - Nothing found on the int_rate
  - term is in object type we need to extract the months from it and change to float / int
  - loan_amount can have outliers and therefore the mismatches
  - annual income has also difference in the mean and the 50 percent
- Observation 2
  - loan amount and installment are highly correlated
  - pub_rec and pub_rec_bankruptcies are also very correlated but it may be the same data (need to veryfy)
- Observation 3
  - most people applied has house as mortgage and other are living on rented house
  - looking at the capacity of repayment with these might help
- Observations 4
  - there are only 2 terms available ie 36 months and 60 minths
- Observation 5
  - most people applied has house as mortgage and other are living on rented house
  - looking at the capacity of repayment with these might help
- Observation 6
  - from two sample ttest , we can observe the p-value to be < 0.05 , which is not significant ,
  - hence we reject null hypothesis
  - can conclude that installments for fully_paid loan status and charged_off status is not same.
- Observation 7

  - we can see that the group b has the most chance of fullly repaying the loan amount , the repaying of c is the worst as well
  - the repaying capacity of sub group B2 and B3 is the best

- Observation 8
  - the term of 36 months is the most taken
  - most people has owned houses in the mortgage form
  - the verification has less variation in 3 categories
  - debt consolidation and credit card are the most relieble source of loan approval

- Observation 9
  - visually there doent seems to be much correlation between employement length / emp title and loan_status.
  - But from chi-sqaure test, we reject that null hypothesis and
  - Hence conclude that there is a relationship exists.
  - later target encoding or label encoding .

Reccomendation

- Loantap should promote low interest loans as much as possible as proability of paying back the loans are above 90%.
- Loantap should focus on betterment of low grade loans as probability of paying back G grade loans are approx 55 %, one out of two loans are going to get charged off.
- Loantap should speially keep track on the loans that has high interest paying loans as there are 40% chance to make a default.
- Loantap should promote joint loans as much as possible as it has the highest probability of getting fully paid among all catagories.
- As there are high numbers of public derogatory records the chances of making default increases, thus loantap should avoid paying loans to such users.

In [ ]:

```
1
```