In [76]:
```
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_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 roc_auc_score
from statsmodels.stats.outliers_influence import variance_inflation_factor
from imblearn.over_sampling import SMOTE
```

In [2]:
```
import pandas as pd
pd.set_option('display.max_columns', 500)
```

In [3]:
```
data = pd.read_csv('Loantap_Business case.csv')
data.head()
```

Out[3]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_owne
0	10000	36 months	11.44	329.48	B	B4	Marketing	10+ years	
1	Mendozaberg	OK 22690"	NaN	NaN	NaN	NaN	NaN	NaN	
2	8000	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORT
3	Loganmouth	SD 05113"	NaN	NaN	NaN	NaN	NaN	NaN	
4	15600	36 months	10.49	506.97	B	B3	Statistician	< 1 year	

In [4]:
```
data = data[data.index % 2 == 0]
data.reset_index(drop=True, inplace=True)
```

In [5]:
```
data
```

Out[5]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home
0	10000	36 months	11.44	329.48	B	B4	Marketing	10+ years	
1	8000	36 months	11.99	265.68	B	B5	Credit analyst	4 years	
2	15600	36 months	10.49	506.97	B	B3	Statistician	< 1 year	
3	7200	36 months	6.49	220.65	A	A2	Client Advocate	6 years	
4	24375	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	
...	...	...	...	...	...	...	...	...	
396025	10000	60 months	10.99	217.38	B	B4	licensed bankere	2 years	
396026	21000	36 months	12.29	700.42	C	C1	Agent	5 years	
396027	5000	36 months	9.99	161.32	B	B1	City Carrier	10+ years	
396028	21000	60 months	15.31	503.02	C	C2	Gracon Services, Inc	10+ years	
396029	2000	36 months	13.61	67.98	C	C2	Internal Revenue Service	10+ years	

396030 rows × 27 columns

In [6]:

```
data.shape
```

Out[6]:

```
(396030, 27)
```

In [7]:

```
# checking the distribution of outcome labels

data.loan_status.value_counts(normalize = True)*100
```

Out[7]:
```
Fully Paid     80.387092
Charged Off    19.612908
Name: loan_status, dtype: float64
```

In [8]:
```
#statistical summary of dataset
data.describe(include = "all")
```

Out[8]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length
count	396030	396030	396030.000000	396030.000000	396030	396030	373103	377729
unique	1397	2	NaN	NaN	7	35	173103	11
top	10000	36 months	NaN	NaN	B	B3	Teacher	10+ years
freq	27668	302005	NaN	NaN	116018	26655	4389	126041
mean	NaN	NaN	13.639400	431.849698	NaN	NaN	NaN	NaN
std	NaN	NaN	4.472157	250.727790	NaN	NaN	NaN	NaN
min	NaN	NaN	5.320000	16.080000	NaN	NaN	NaN	NaN
25%	NaN	NaN	10.490000	250.330000	NaN	NaN	NaN	NaN
50%	NaN	NaN	13.330000	375.430000	NaN	NaN	NaN	NaN
75%	NaN	NaN	16.490000	567.300000	NaN	NaN	NaN	NaN
max	NaN	NaN	30.990000	1533.810000	NaN	NaN	NaN	NaN

In [9]:
```
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  object 
 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(11), object(16)
memory usage: 81.6+ MB
```

In [10]:
```
data['loan_amnt'] = pd.to_numeric(data['loan_amnt'])
```

In [11]:
```
data.info()
```

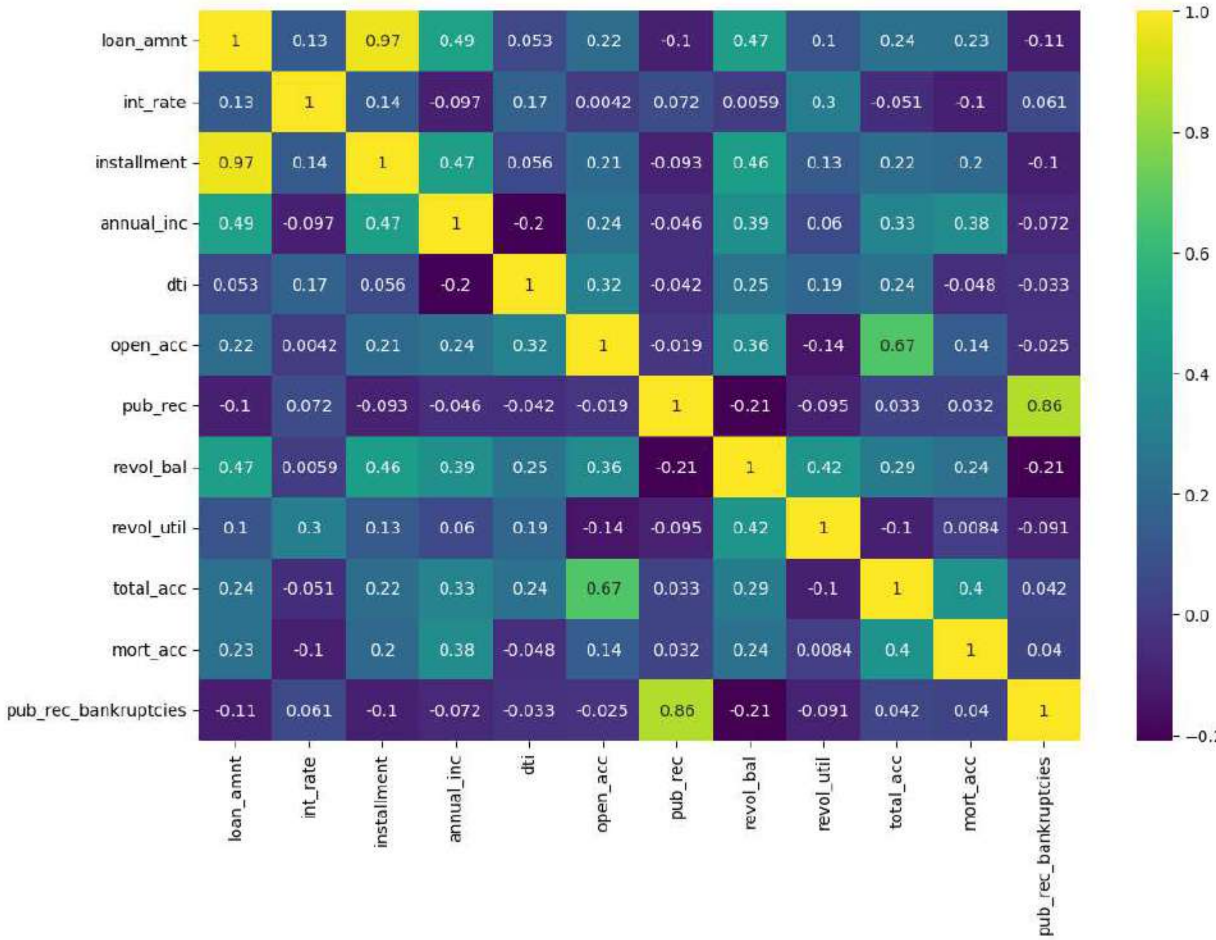
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             396030 non-null  int64  
 1   term                  396030 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(11), int64(1), object(15)
memory usage: 81.6+ MB
```

spearman -- to check relationship other than linear

pearson --- to check linear relationship

In [12]:
```
plt.figure(figsize = (12,8))
sns.heatmap(data.corr(method = 'spearman'), annot = True, cmap = 'viridis')
plt.show()
```

```
C:\Users\bikim\AppData\Local\Temp\ipykernel_23632\3366924295.py:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  sns.heatmap(data.corr(method = 'spearman'), annot = True, cmap = 'viridis')
```



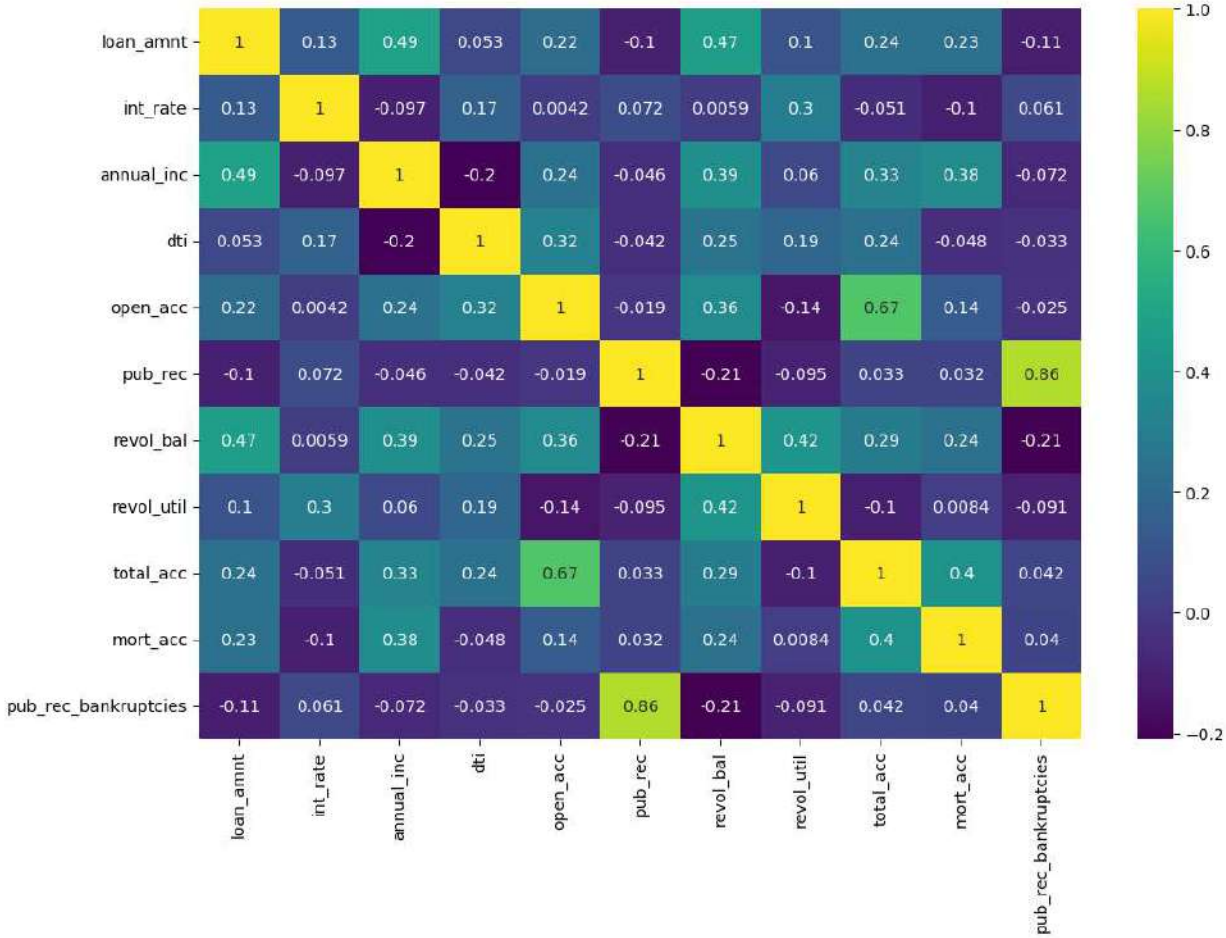
We noticed almost perfect corelation between "Loan amnt" and "installment" feature

so we can drop either one of those columns

In [13]:
```
data.drop(columns = ["installment"], axis = 1, inplace = True)
```

In [14]:
```
plt.figure(figsize = (12,8))
sns.heatmap(data.corr(method = 'spearman'), annot = True, cmap = 'viridis')
plt.show()
```

```
C:\Users\bikim\AppData\Local\Temp\ipykernel_23632\3366924295.py:2: FutureWarning: The
default value of numeric_only in DataFrame.corr is deprecated. In a future version, i
t will default to False. Select only valid columns or specify the value of numeric_on
ly to silence this warning.
  sns.heatmap(data.corr(method = 'spearman'), annot = True, cmap = 'viridis')
```



Data Exploration

In [15]:
```
data.groupby(by = 'loan_status')['loan_amnt'].describe()
```

Out[15]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77673.0	15126.300967	8505.090557	1000.0	8525.0	14000.0	20000.0	40000.0
Fully Paid	318357.0	13866.878771	8302.319699	500.0	7500.0	12000.0	19225.0	40000.0

So we can see if the mean loan amount is 13k $peopleareabletopaybackloanandifthemeanloanamountis15k$ or more than 13k$ people are not able to pay it back

In [16]:
```
data["home_ownership"].value_counts()
```

Out[16]:
```
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
ANY              3
Name: home_ownership, dtype: int64
```

Majority of people have home_ownership as Mortgae and rent

combining the minority classes as others

In [17]:
```
data.loc[(data.home_ownership == 'ANY') | (data.home_ownership == 'NONE'), 'home_owner
data.home_ownership.value_counts()
```

Out[17]:
```
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          146
Name: home_ownership, dtype: int64
```

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

Out[18]:
```
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          146
Name: home_ownership, dtype: int64
```

In [19]:
```
#checking the distribution of others

data.loc[data['home_ownership']== 'OTHER', 'loan_status'].value_counts()
```

Out[19]:
```
Fully Paid     123
Charged Off     23
Name: loan_status, dtype: int64
```

Its look like title column was filled manually and needs some fixing

In [20]:
```
data['title'].value_counts()[:20]
```

Out[20]:
```
Debt consolidation           152472
Credit card refinancing       51487
Home improvement              15264
Other                         12930
Debt Consolidation            11608
Major purchase                 4769
Consolidation                  3852
debt consolidation             3547
Business                       2949
Debt Consolidation Loan        2864
Medical expenses               2742
Car financing                  2139
Credit Card Consolidation      1775
Vacation                       1717
Moving and relocation          1689
consolidation                  1595
Personal Loan                  1591
Consolidation Loan             1299
Home Improvement               1268
Home buying                    1183
Name: title, dtype: int64
```

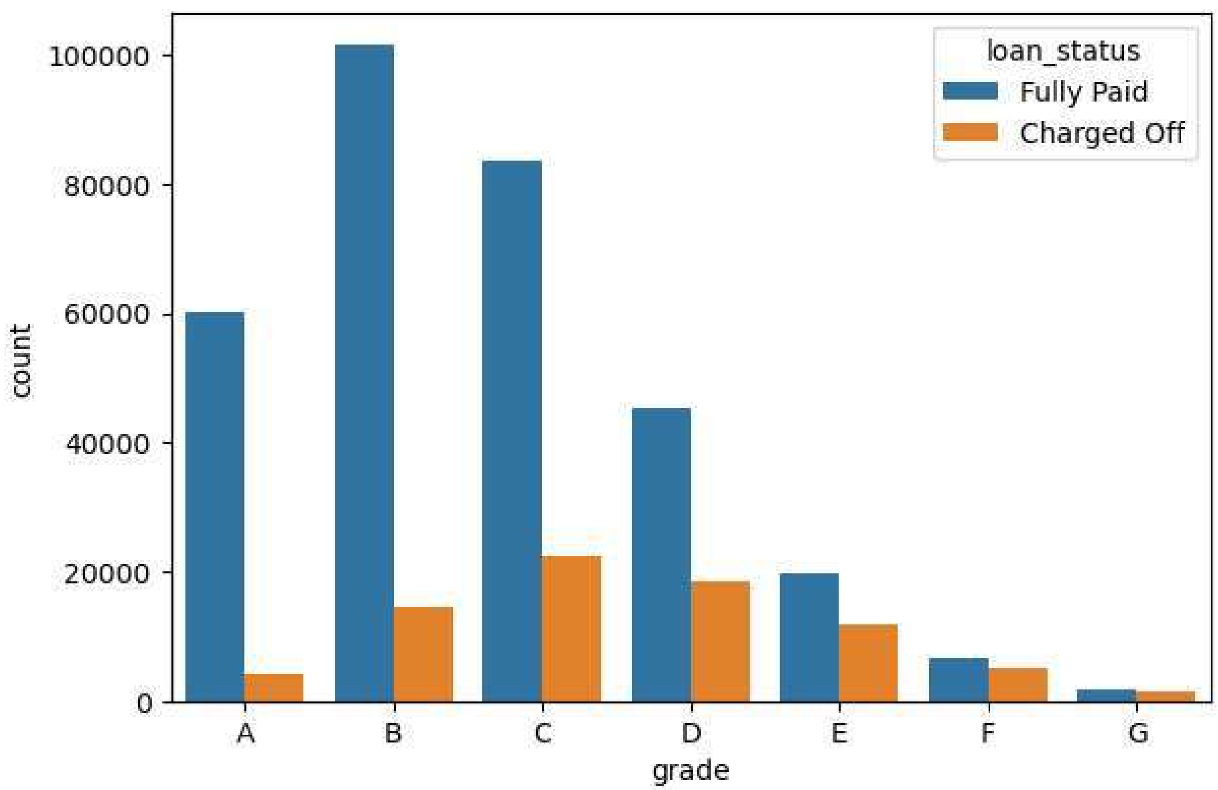
In [21]:
```
data['title'] = data.title.str.lower()
```

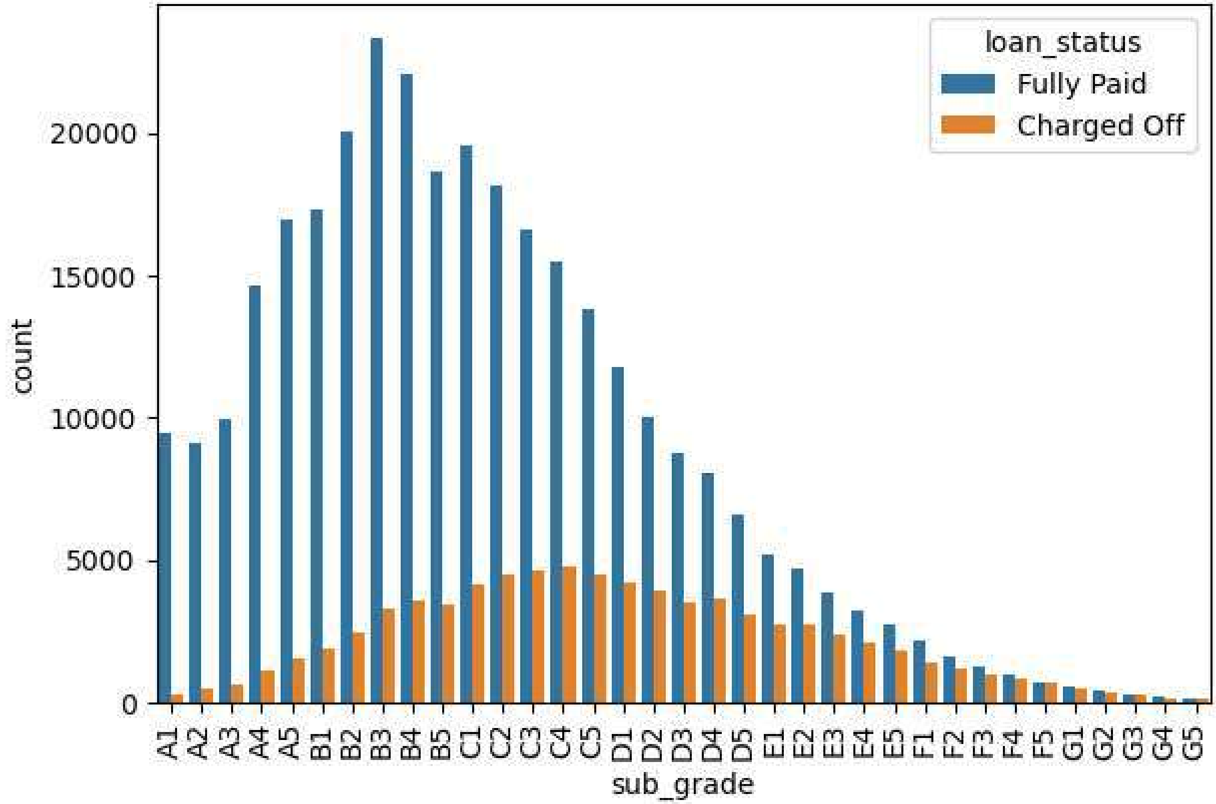
In [22]:
```
data.title.value_counts()[:10]
```

Out[22]:
```
debt consolidation          168108
credit card refinancing      51781
home improvement             17117
other                        12993
consolidation                 5583
major purchase                4998
debt consolidation loan       3513
business                      3017
medical expenses              2820
credit card consolidation     2638
Name: title, dtype: int64
```

Visualization

In [23]:
```python
plt.figure(figsize = (15,10))

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

# plt.subplot(2,2,1)
# sub_grade = sorted(data.sub_grade.unique().tolist())
# g = sns.countplot(x="sub_grade", data = data, hue= 'loan_status', order = sub_grade)
# g.set_xticklabels(g.get_xticklabels(), rotation = 90);


plt.figure(figsize = (15,10))

plt.subplot(2,2,1)
sub_grade = sorted(data.sub_grade.unique().tolist())
g = sns.countplot(x="sub_grade", data = data, hue= 'loan_status', order = sub_grade)
g.set_xticklabels(g.get_xticklabels(), rotation = 90);
```

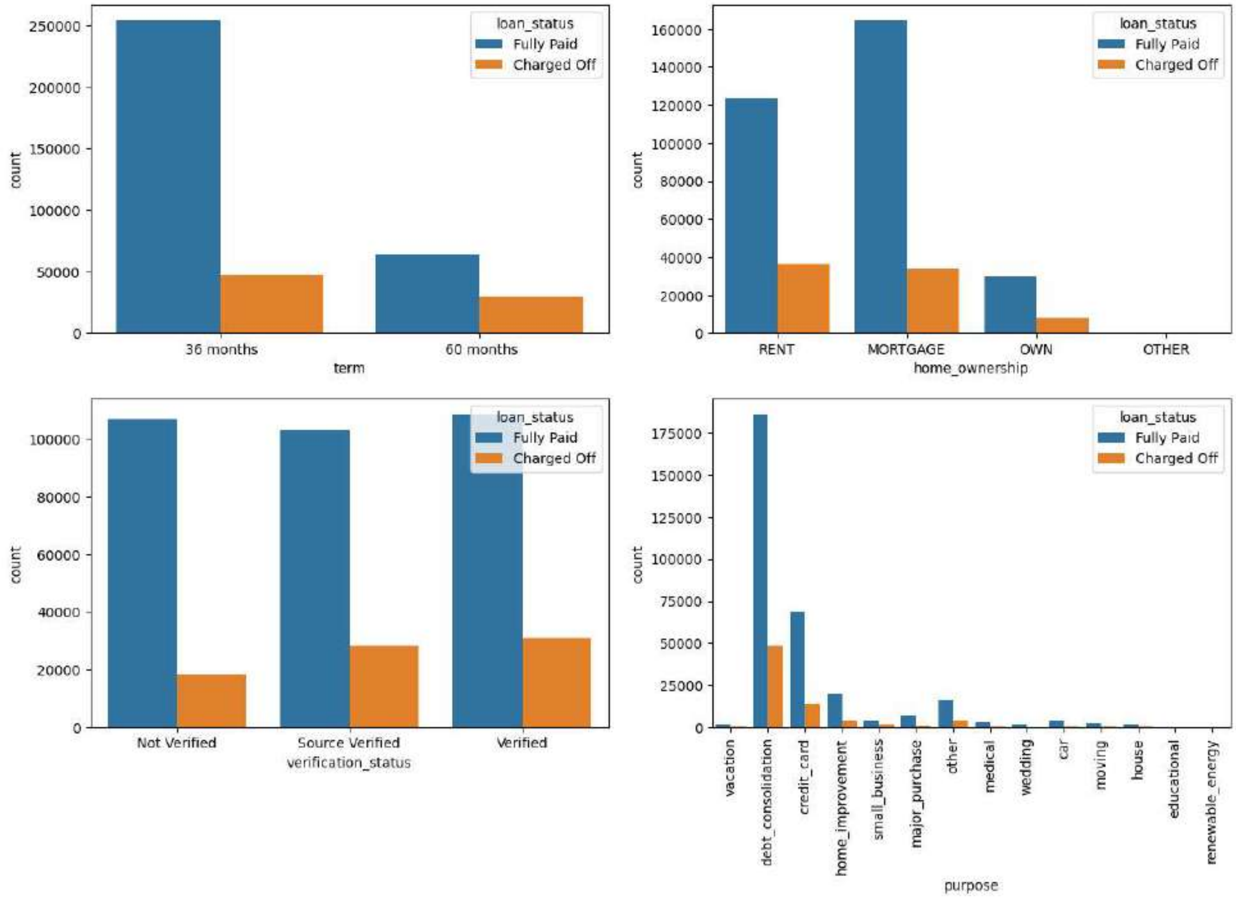




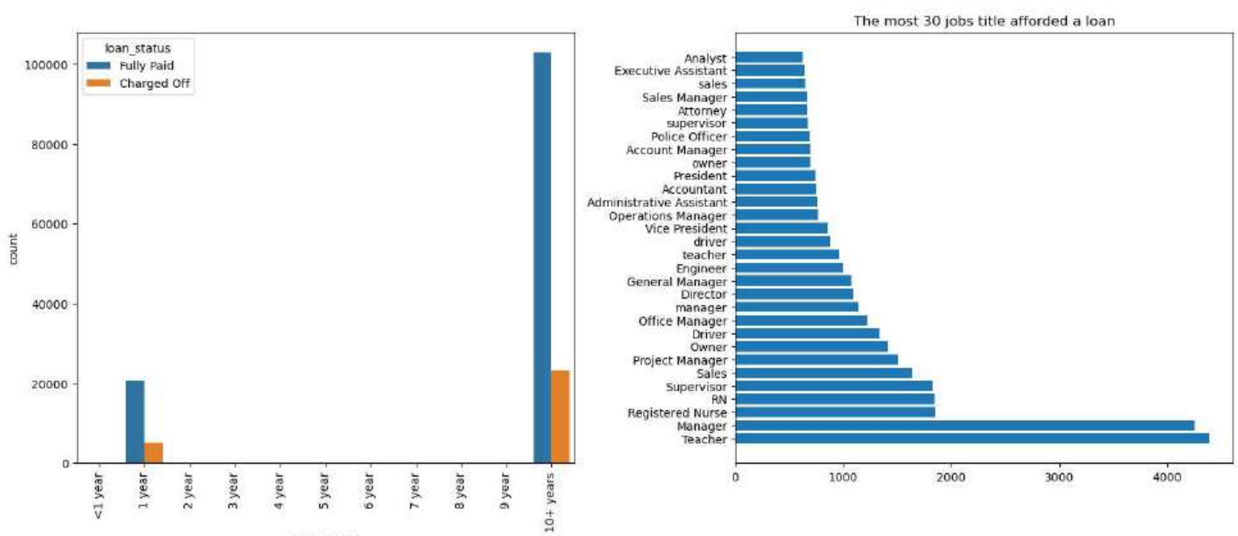
The grade of majority of people those who have fully paid the loan is 'B' and subgrade is 'B3'.So we can say people with grade 'B' and subgrade 'B3' are more likely to fully pay the loan.

G grade people have almost 1:1 ratio of Fullypaid vs charged off. Means their is 50% possibility that they are able to repay loan back. So bank should put more interest rate on them.

In [24]:
```
plt.figure(figsize = (15,20))

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

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

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

plt.subplot(4, 2, 4)
g = sns.countplot(x='purpose', data=data, hue='loan_status')
g.set_xticklabels(g.get_xticklabels(), rotation=90);
```



In [25]:
```
plt.figure(figsize = (15,12))

plt.subplot(2,2,1)
order = ['<1 year', '1 year', '2 year','3 year','4 year','5 year','6 year','7 year','8

g= sns.countplot(x= 'emp_length', data = data, hue = 'loan_status', order = order)
g.set_xticklabels(g.get_xticklabels(), rotation = 90);

plt.subplot(2,2,2)
plt.barh(data.emp_title.value_counts()[:30].index, data.emp_title.value_counts()[:30])
plt.title("The most 30 jobs title afforded a loan")
plt.tight_layout()
```



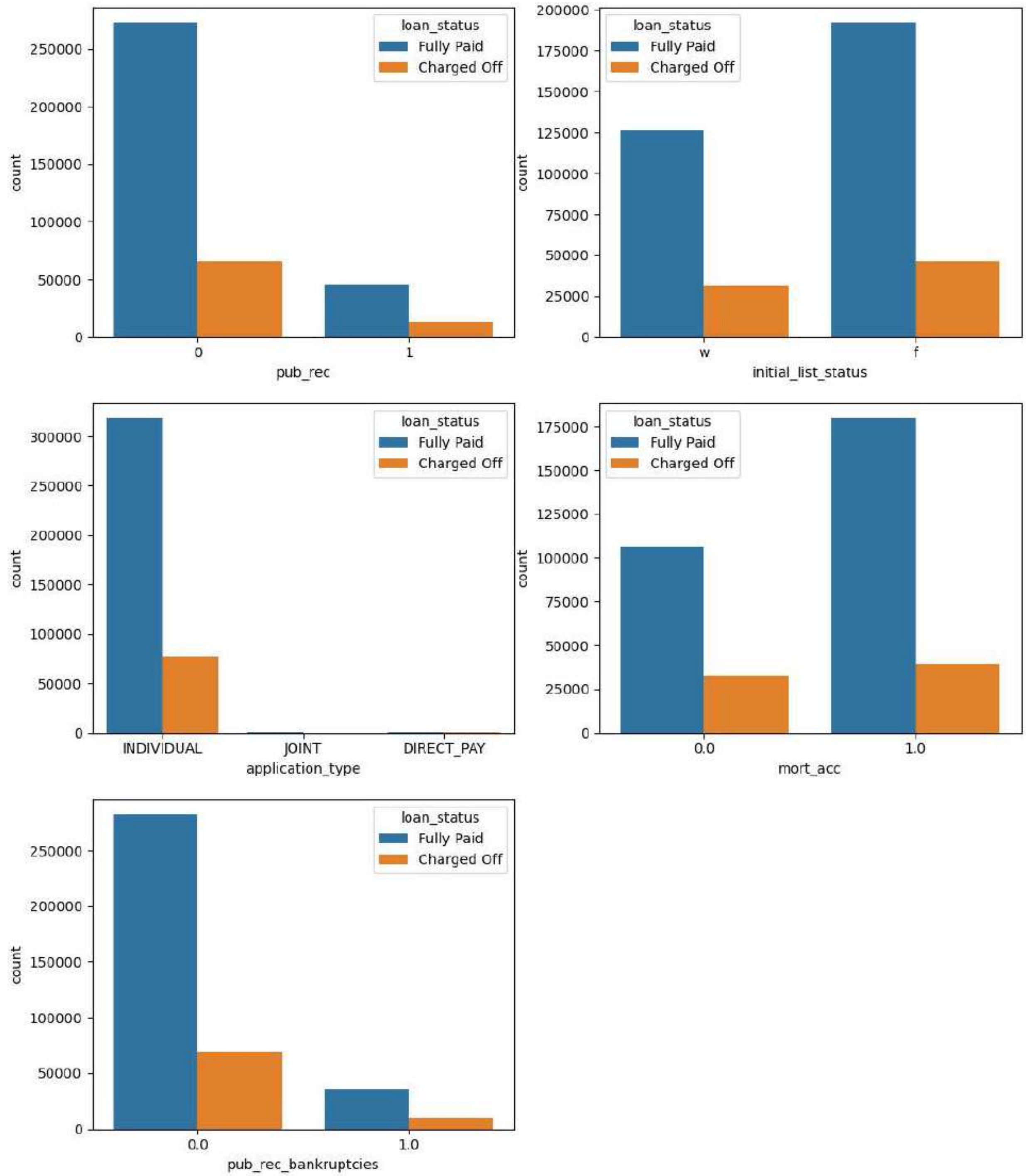
People who have 10 yrs+ experiece are able to repay loan back. Manager and teacher are the most afforded job who can repay loan

Feature Engineering

In [26]:
```
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
```

In [27]:
```
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 [28]:
```
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()
```



In [29]:
```
# Mapping of target variable -
data['loan_status'] = data.loan_status.map({'Fully Paid':0, 'Charged Off':1})
```

In [30]:
```
data.isnull().sum()/len(data)*100
```

Out[30]:
```
loan_amnt               0.000000
term                    0.000000
int_rate                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
```

Mean Imputation

In [31]:
```
data.groupby(by='total_acc')['mort_acc'].median()
```

Out[31]:
```
total_acc
2.0      0.0
3.0      0.0
4.0      0.0
5.0      0.0
6.0      0.0
        ...
124.0    1.0
129.0    1.0
135.0    1.0
150.0    1.0
151.0    0.0
Name: mort_acc, Length: 118, dtype: float64
```

In [32]:
```
total_acc_avg = data.groupby(by = 'total_acc').median().mort_acc
total_acc_avg
# saving mean of mort_acc according to the total_acc_avg (you can pick any variable fc
```

```
C:\Users\bikim\AppData\Local\Temp\ipykernel_23632\3574394951.py:1: FutureWarning: The
default value of numeric_only in DataFrameGroupBy.median is deprecated. In a future v
ersion, numeric_only will default to False. Either specify numeric_only or select onl
y columns which should be valid for the function.
  total_acc_avg = data.groupby(by = 'total_acc').median().mort_acc
```

Out[32]:
```
total_acc
2.0      0.0
3.0      0.0
4.0      0.0
5.0      0.0
6.0      0.0
        ...
124.0    1.0
129.0    1.0
135.0    1.0
150.0    1.0
151.0    0.0
Name: mort_acc, Length: 118, dtype: float64
```

In [33]:
```
def fill_mort_acc(total_acc, mort_acc):
    if np.isnan(mort_acc):
        return total_acc_avg[total_acc].round()
    else:
        return mort_acc
```

In [34]:
```
data['mort_acc'] = data.apply(lambda x: fill_mort_acc(x['total_acc'], x['mort_acc']),
```

In [35]:
```
data.isnull().sum()/len(data)*100
```

Out[35]:
```
loan_amnt               0.000000
term                    0.000000
int_rate                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 [36]:
```
# current no of rows
data.shape
```

Out[36]:
```
(396030, 26)
```

In [37]:
```
#dropping rows with null values
data.dropna(inplace = True)
```

In [38]:
```
data.shape
```

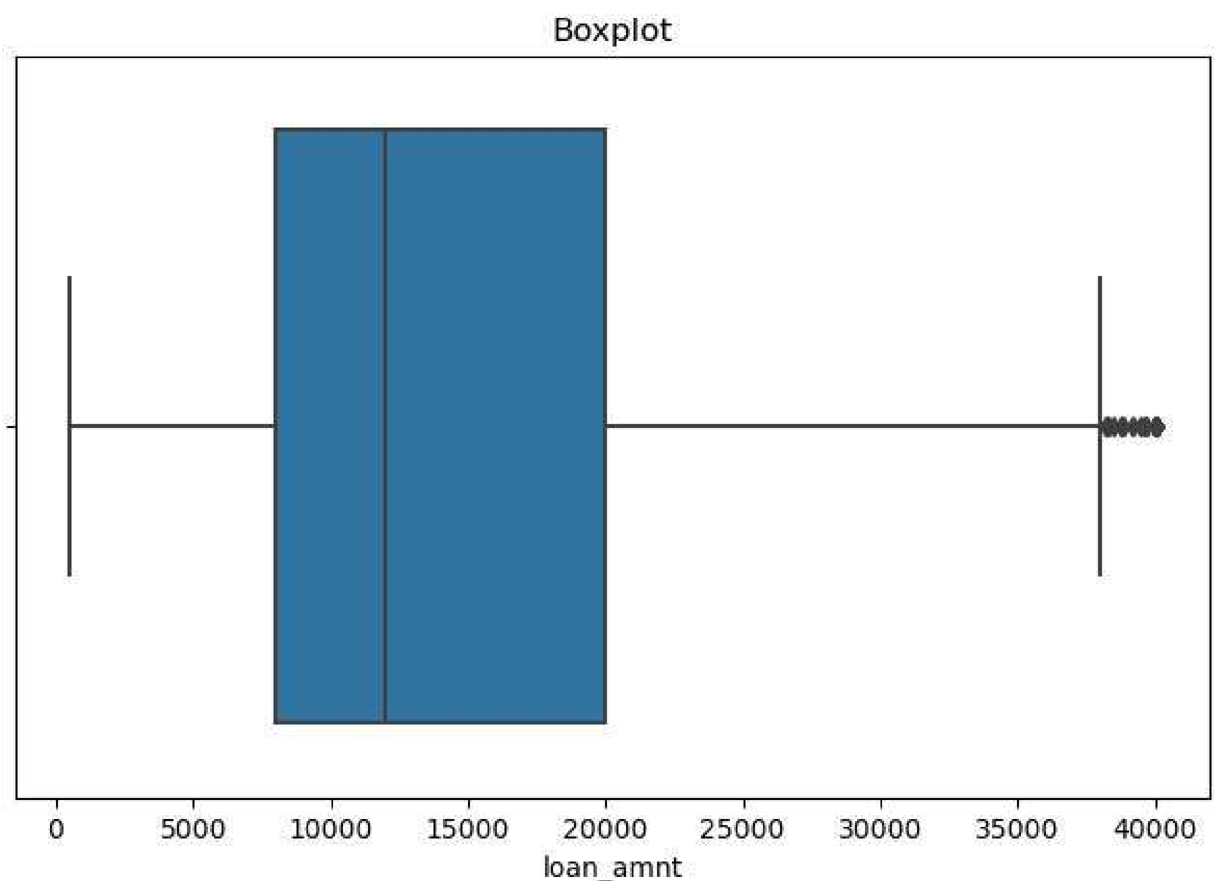
Out[38]: (370622, 26)

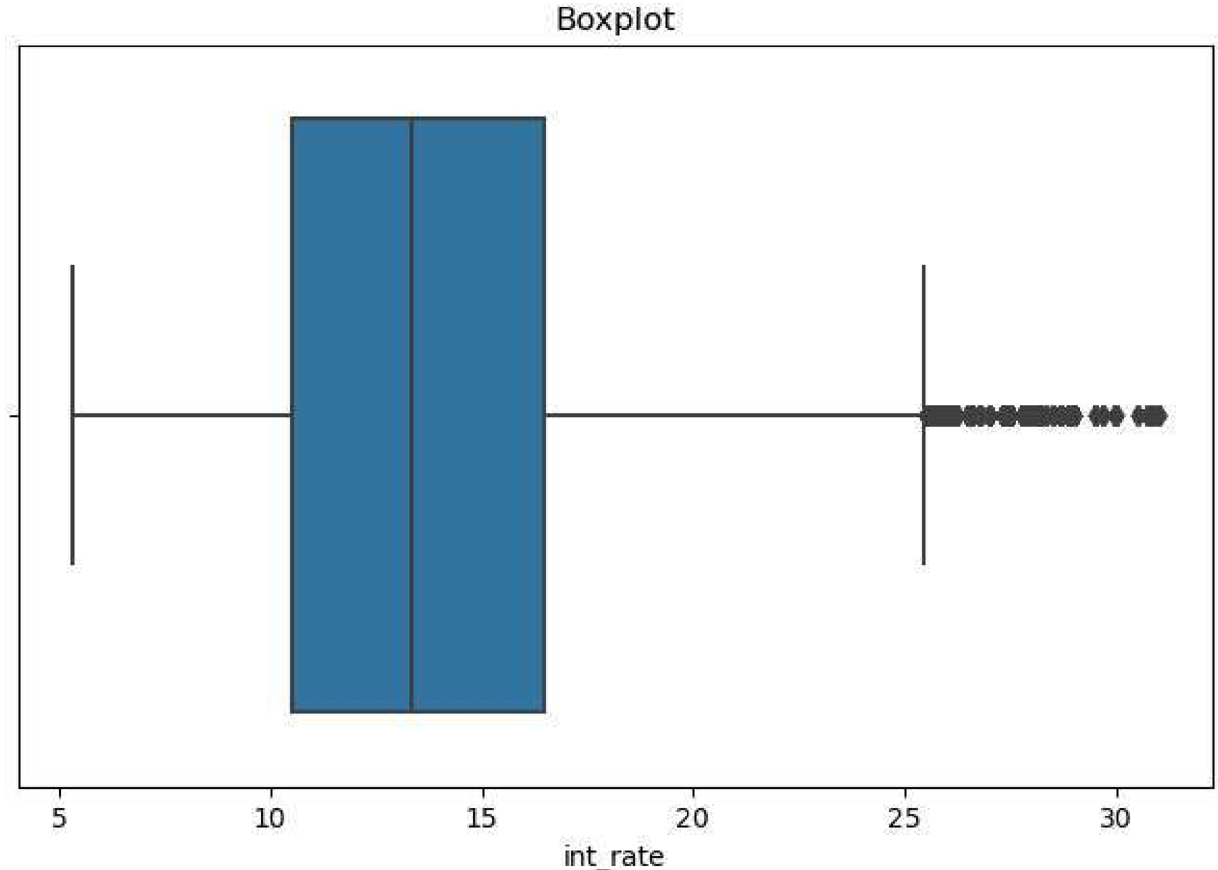
Outlier Detection & Treatment

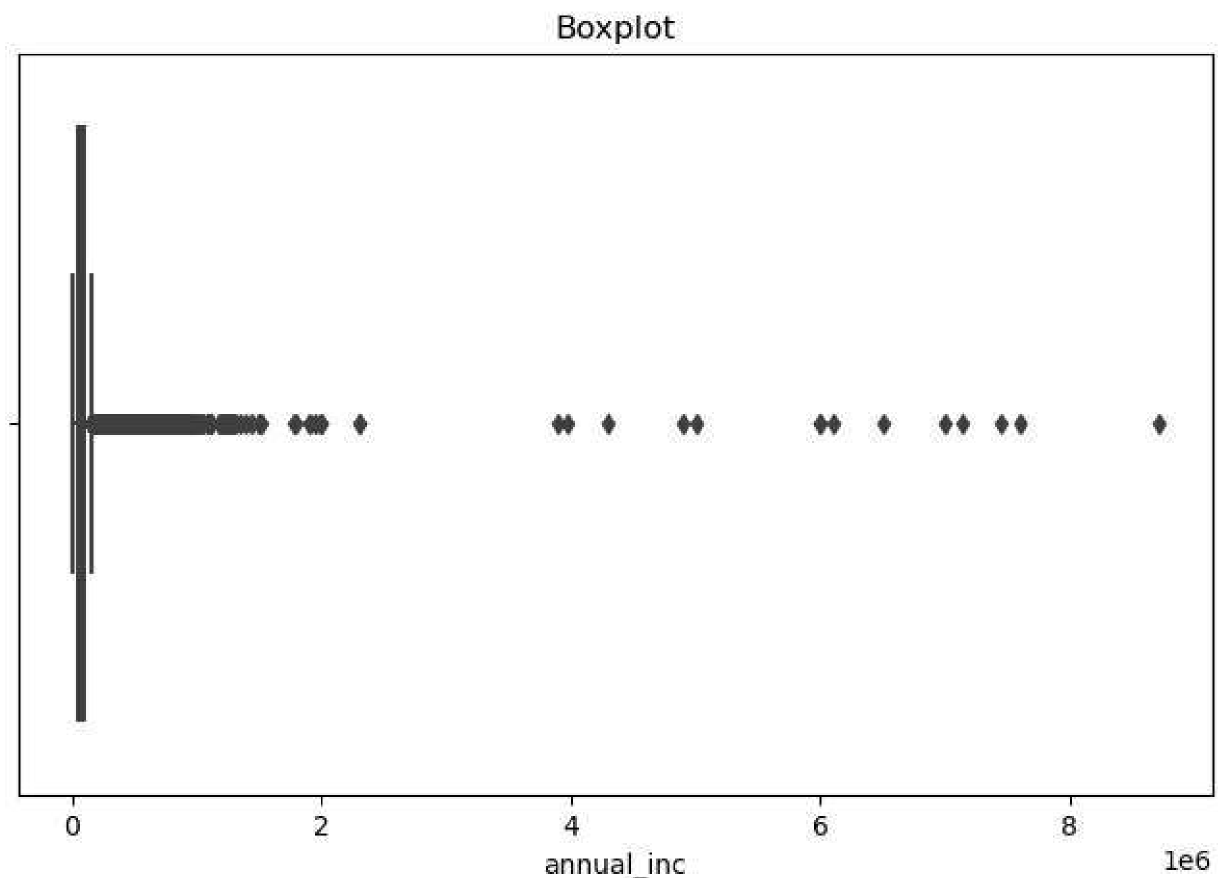
In [39]:
```
numerical_data = data.select_dtypes(include='number')
num_cols = numerical_data.columns
len(num_cols)
```

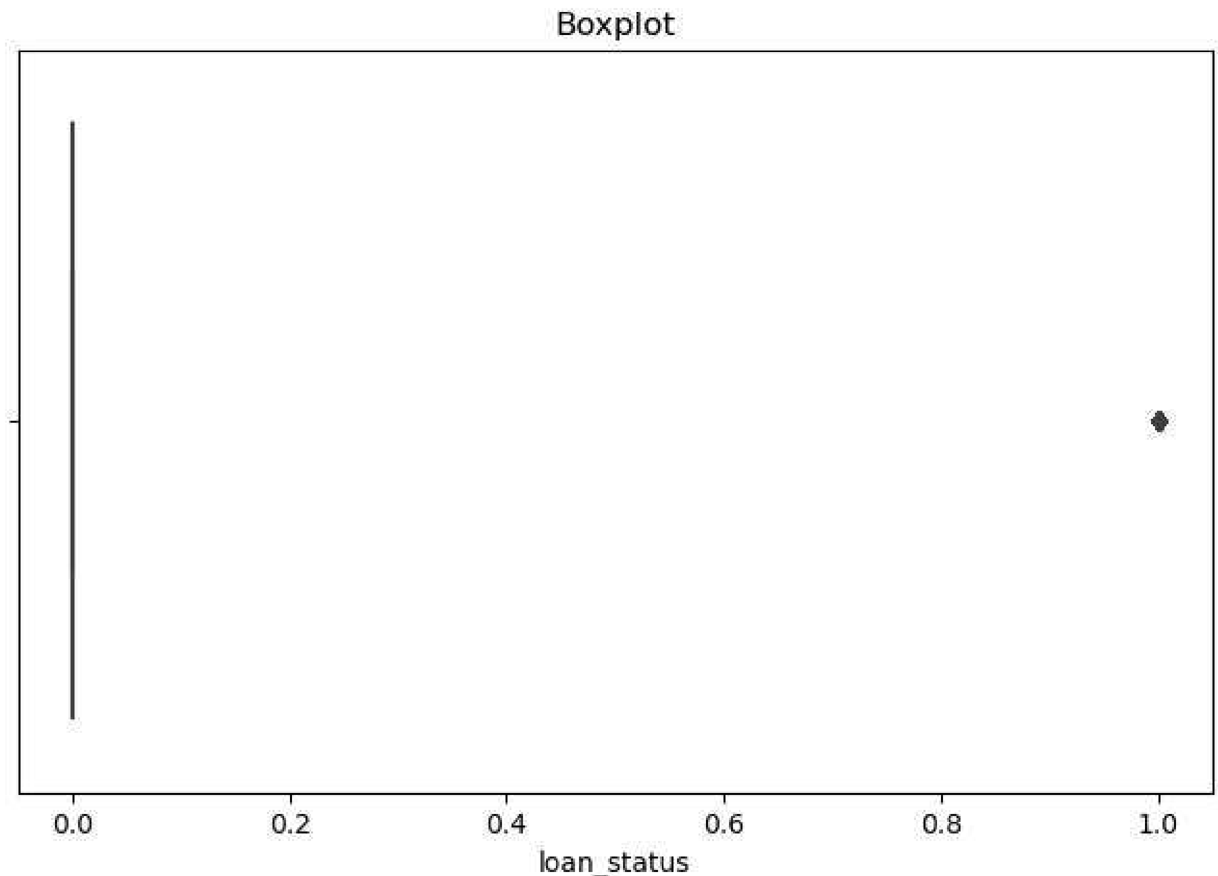
Out[39]: 12

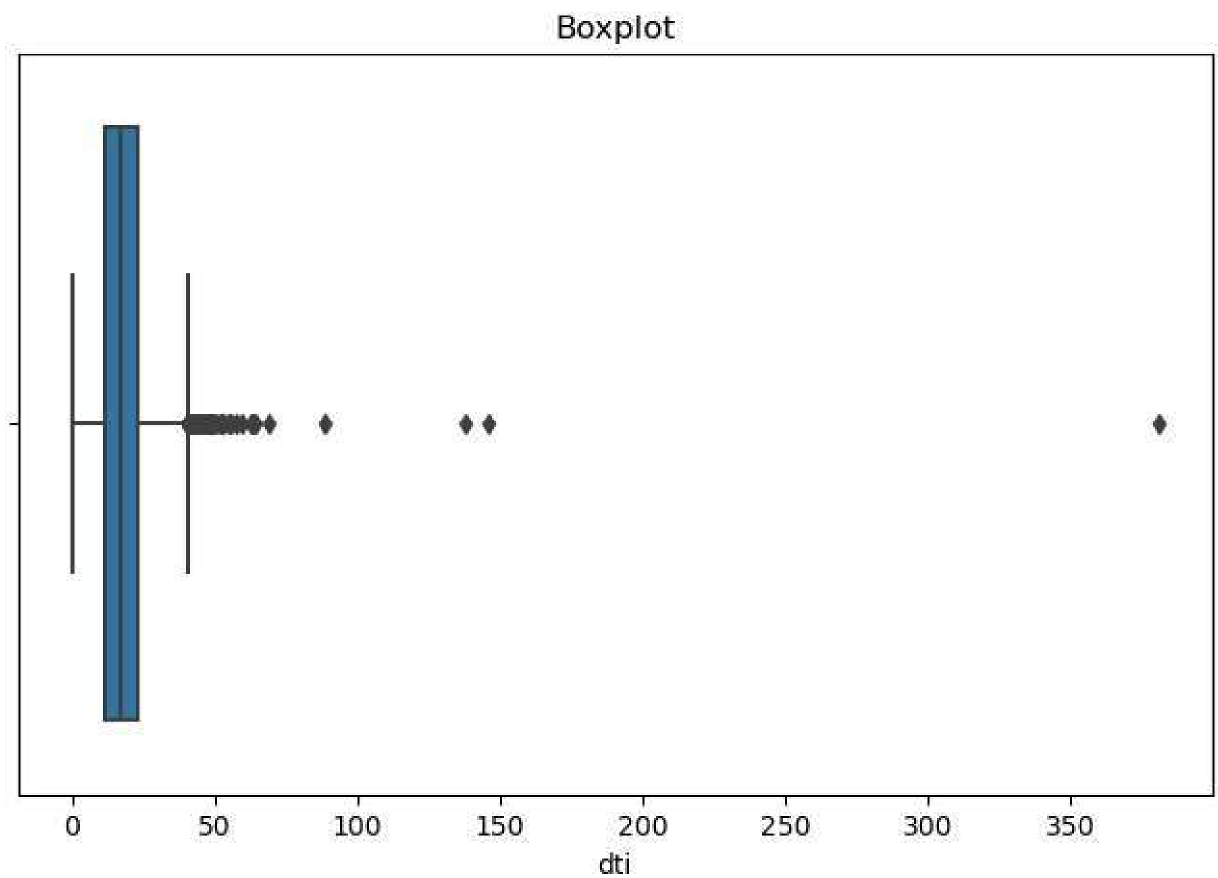
In [40]:
```
def box_plot(col):
    plt.figure(figsize=(8, 5))
    sns.boxplot(x=data[col])
    plt.title('Boxplot')
    plt.show()

for col in num_cols:
    box_plot(col)
```

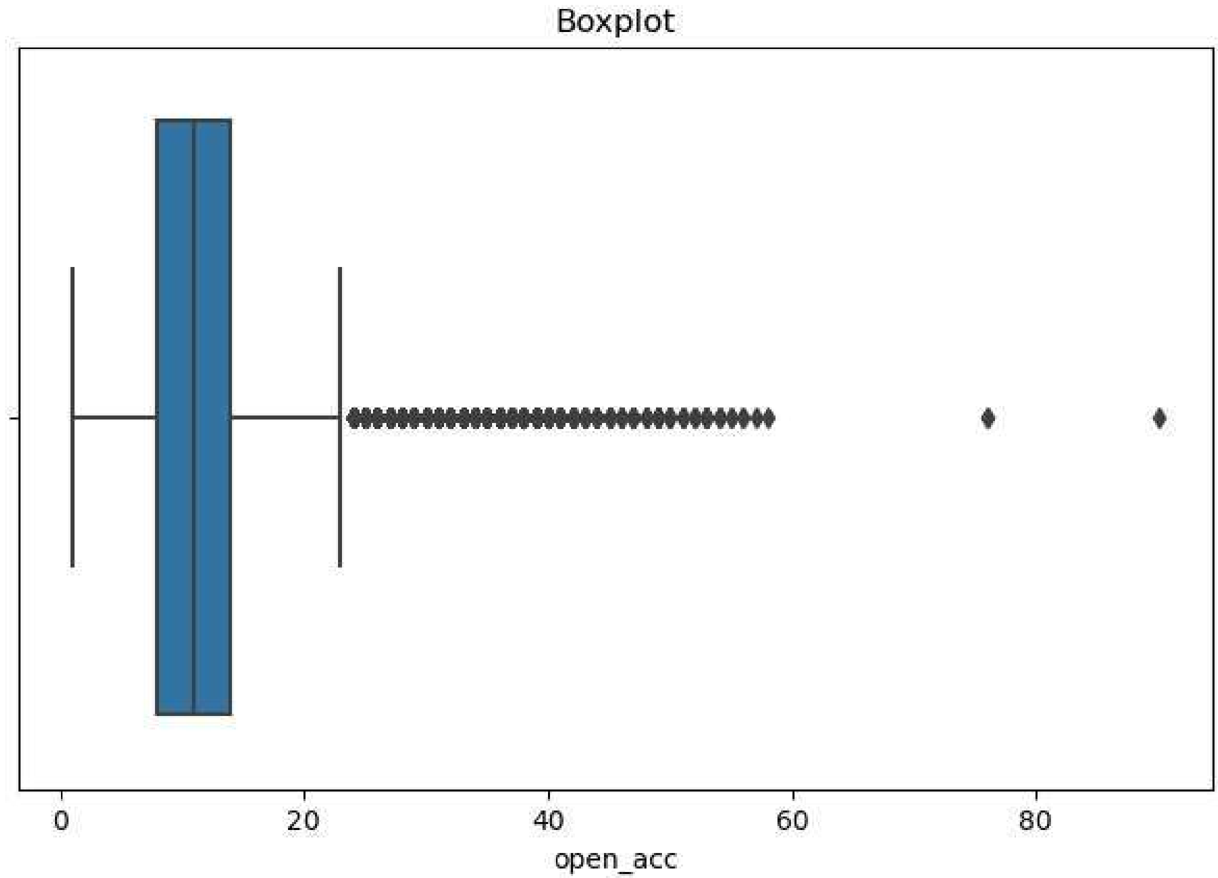


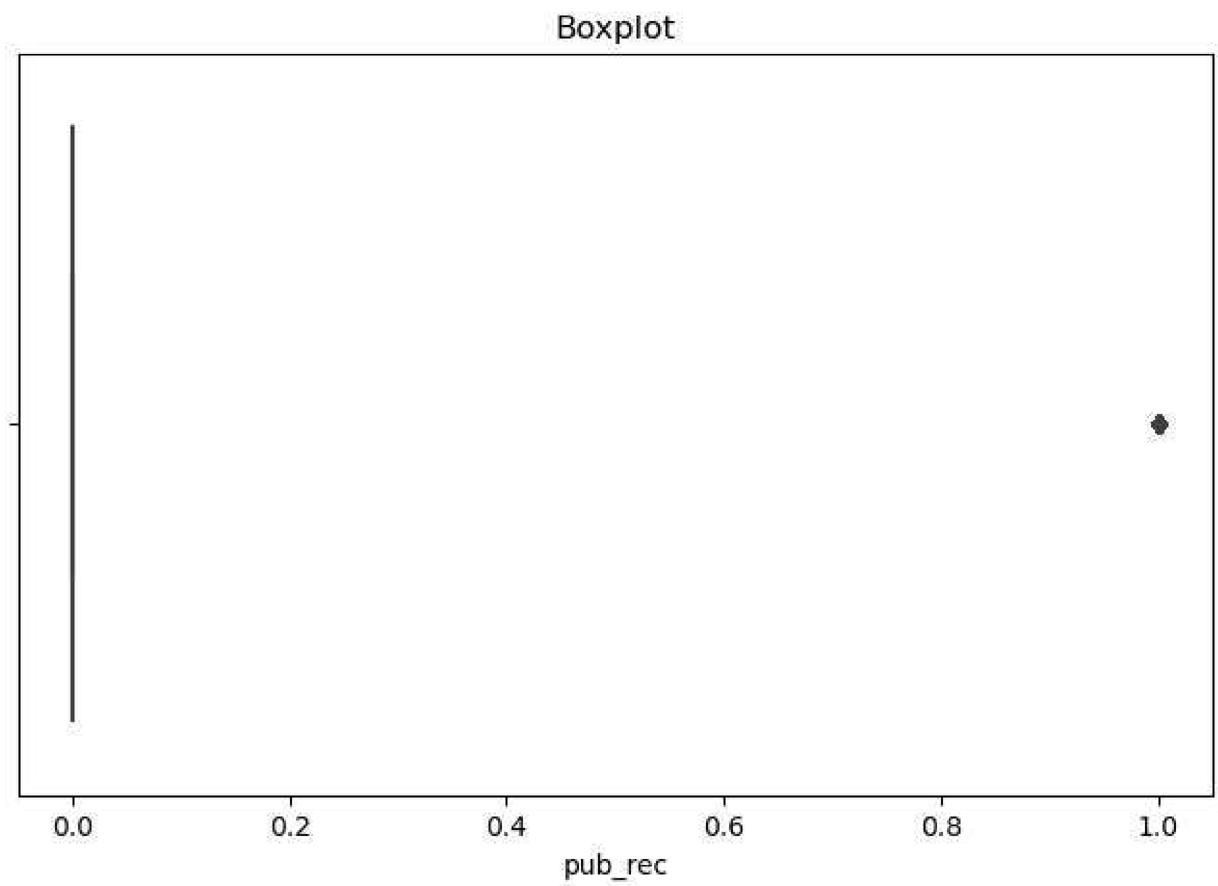

Boxplot
5
10
15
20
25
30
int_rate

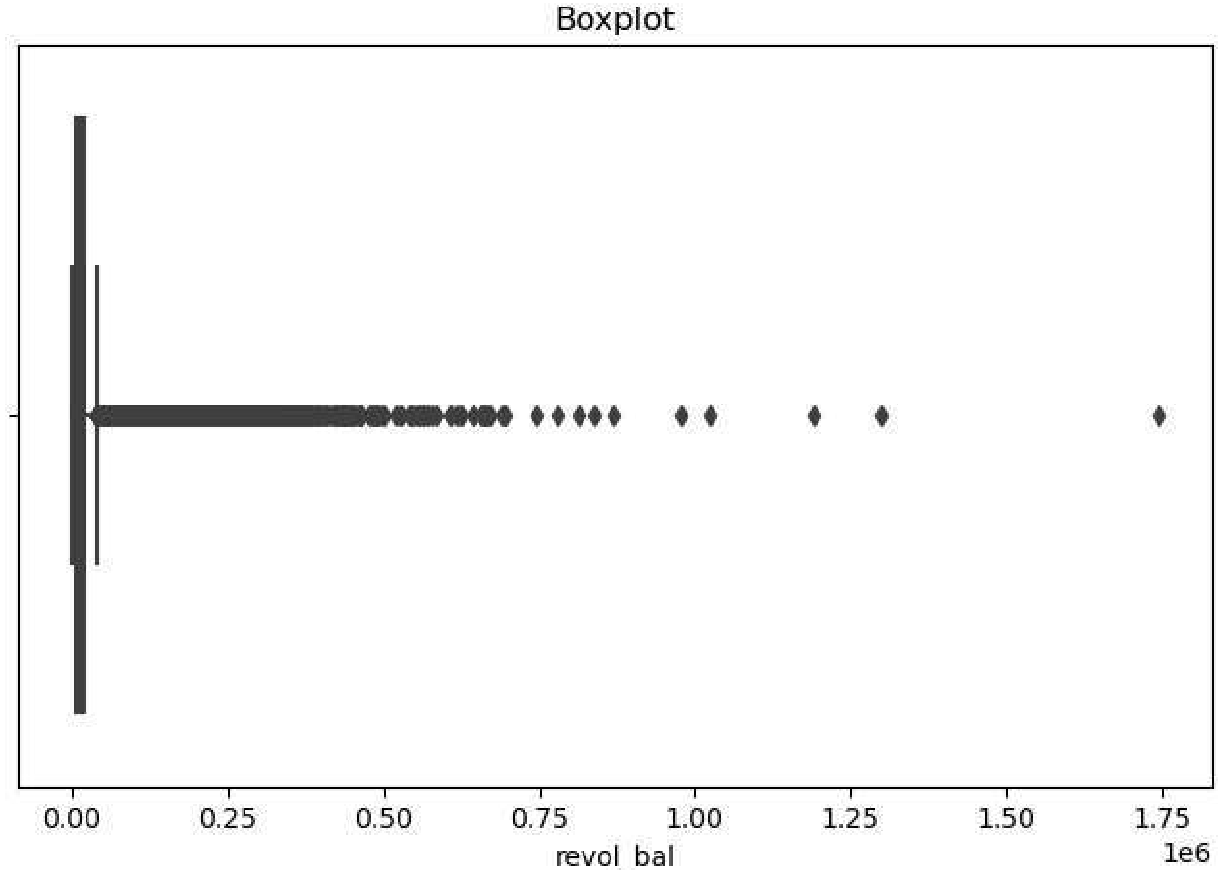

Boxplot
0
2
4
6
8
1e6
annual_inc

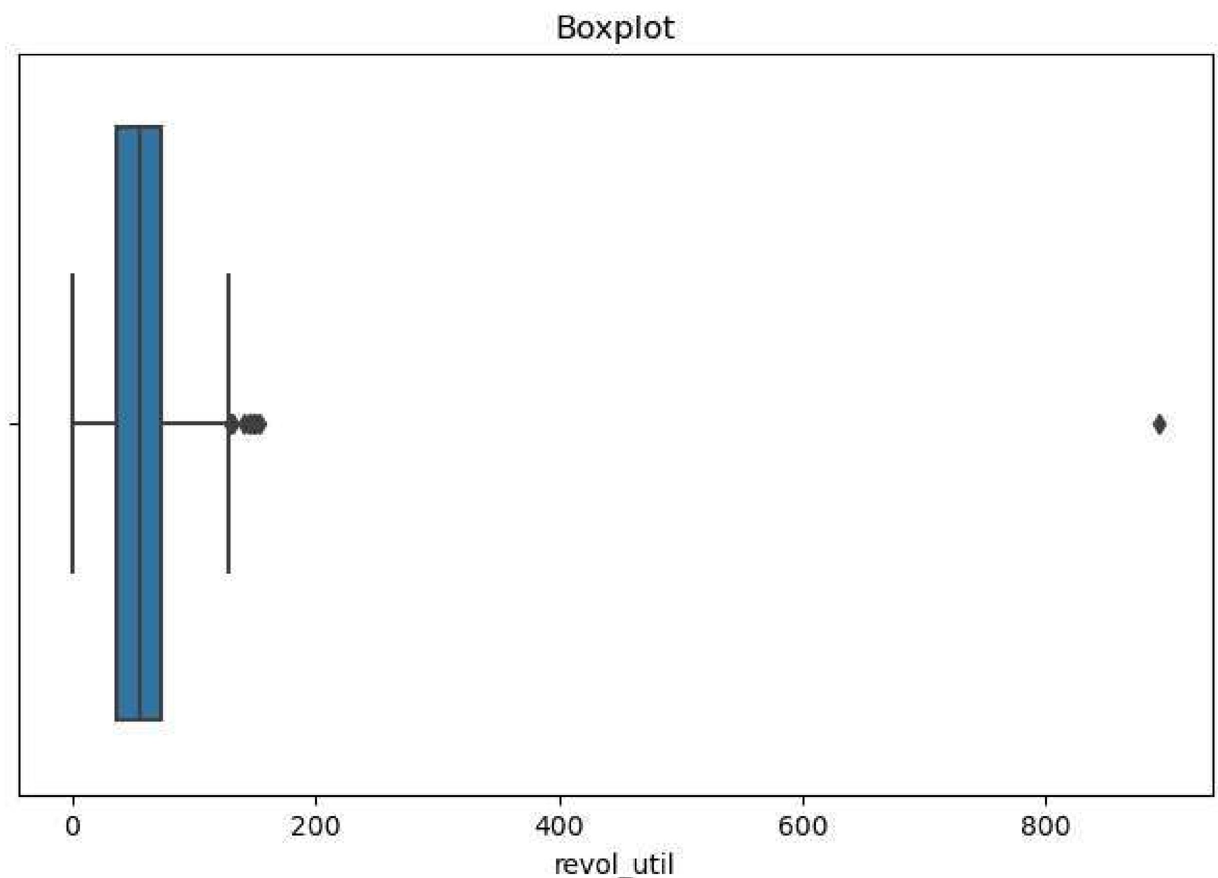

Boxplot
0.0
0.2
0.4
0.6
0.8
1.0
loan_status

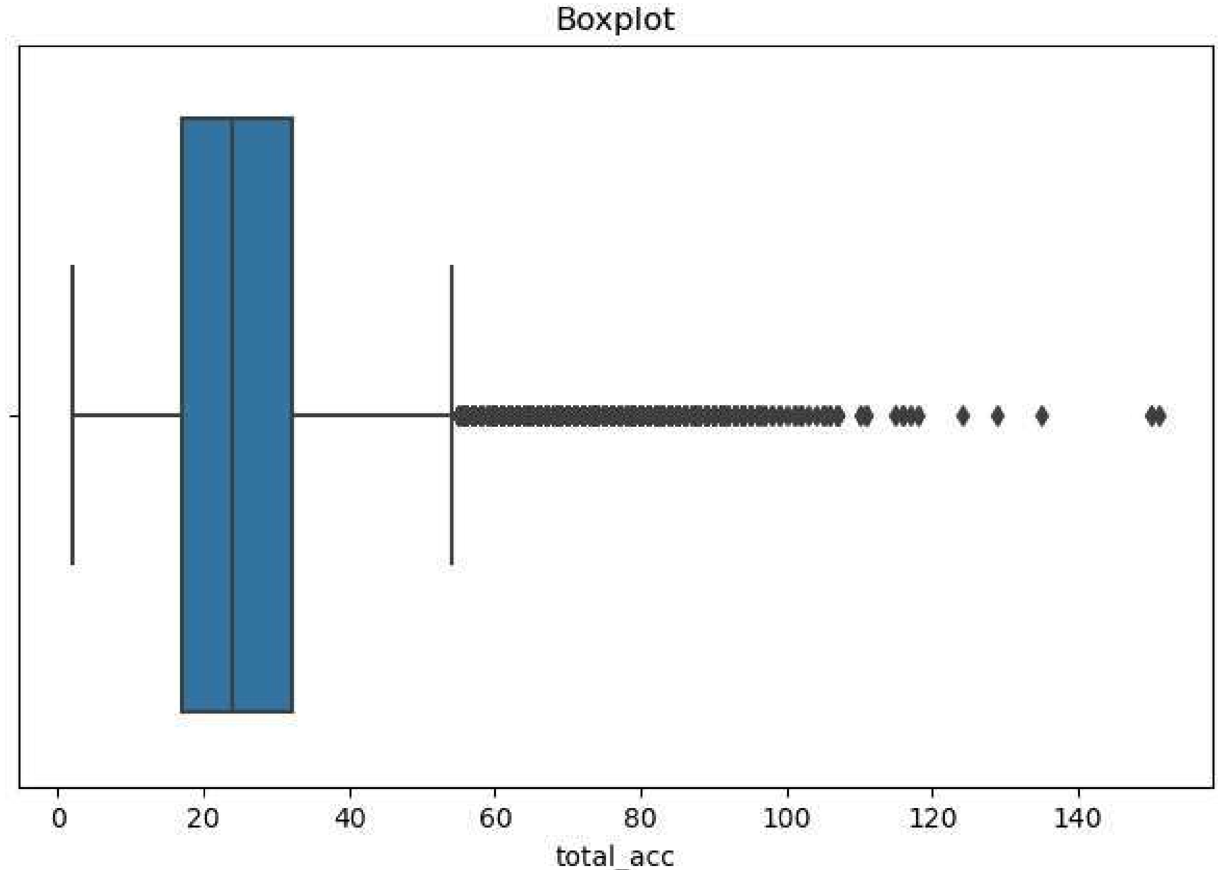

Boxplot
0
50
100
150
200
250
300
350
dti

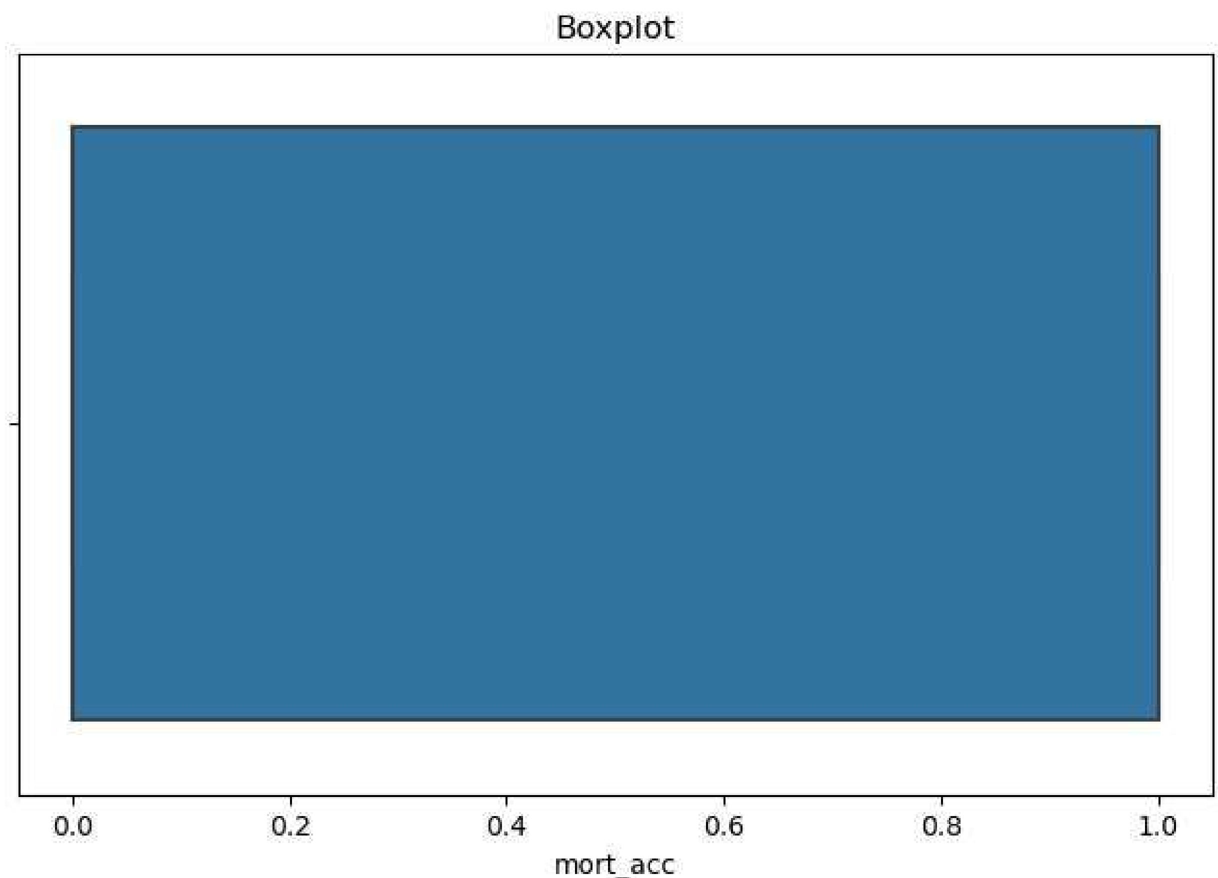

Boxplot
0
20
40
60
80
open_acc

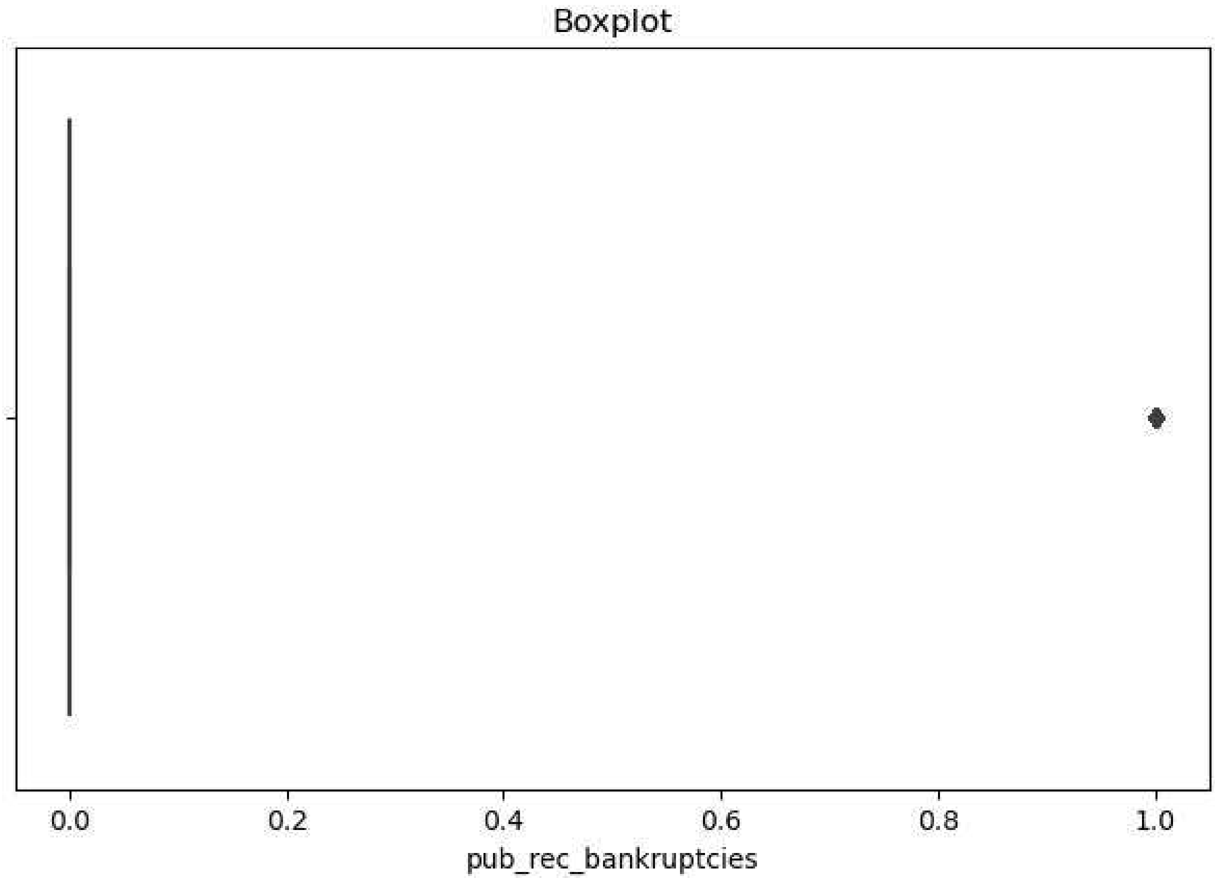

Boxplot
0.0
0.2
0.4
0.6
0.8
1.0
pub_rec


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
0
20
40
60
80
100
120
140
total_acc


Boxplot
0.0
0.2
0.4
0.6
0.8
1.0
mort_acc



In [41]:
```
for col in num_cols:
    mean = data[col].mean()
    std = data[col].std()

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

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

data.shape
```

Out[41]: (354519, 26)

Data Preprocessing

In [42]:
```
# Term -
data.term.unique()
```

Out[42]: array([' 36 months', ' 60 months'], dtype=object)

In [43]:
```
term_values = {' 36 months': 36, ' 60 months': 60}
data['term'] = data.term.map(term_values)
```

In [44]:
```
# Initial List Status -
data['initial_list_status'].unique()
```

Out[44]: array(['w', 'f'], dtype=object)

In [45]:
```
list_status = {'w': 0, 'f': 1}
data['initial_list_status'] = data.initial_list_status.map(list_status)
```

In [46]:
```
# 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'],
                   axis=1, inplace=True)
#target encoding - freqency mapping
#a - number of rows
#b
```

One - Hot Encoding

In [47]:
```
dummies = ['purpose', 'grade', 'verification_status', 'application_type', 'home_owners
data = pd.get_dummies(data, columns=dummies, drop_first=True)
```

In [48]:
```
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

data.head()
```

Out[48]:

	loan_amnt	term	int_rate	annual_inc	loan_status	dti	open_acc	pub_rec	revol_bal	revol_util
0	10000	36	11.44	117000.0	0	26.24	16.0	0	36369.0	41.8
1	8000	36	11.99	65000.0	0	22.05	17.0	0	20131.0	53.3
2	15600	36	10.49	43057.0	0	12.79	13.0	0	11987.0	92.2
3	7200	36	6.49	54000.0	0	2.60	6.0	0	5472.0	21.5
4	24375	60	17.27	55000.0	1	33.95	13.0	0	24584.0	69.8

In [49]:
```
data.shape
```

Out[49]: (354519, 40)

Data preparation for Modelling

In [50]:
```
x = data.drop('loan_status', axis= 1)
y = data['loan_status']
```

In [51]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30,
                                                    stratify=y, random_state=42)
```

In [52]:
```
print(x_train.shape)
print(x_test.shape)
```

```
(248163, 39)
(106356, 39)
```

In [53]:
```
scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

Logistic Regression

In [54]:
```
logreg = LogisticRegression(max_iter=1000)
logreg.fit(x_train, y_train)
```

Out[54]:
```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

In [55]:
```
y_pred = logreg.predict(x_test)
print('Accuracy of Logistic Regression Classifier on test set: {:.3f}'.format(logreg.s
```

```
Accuracy of Logistic Regression Classifier on test set: 0.810
```

Confusion Matrix

In [56]:
```
confusion_matrix = confusion_matrix(y_test,y_pred)
print(confusion_matrix)
```

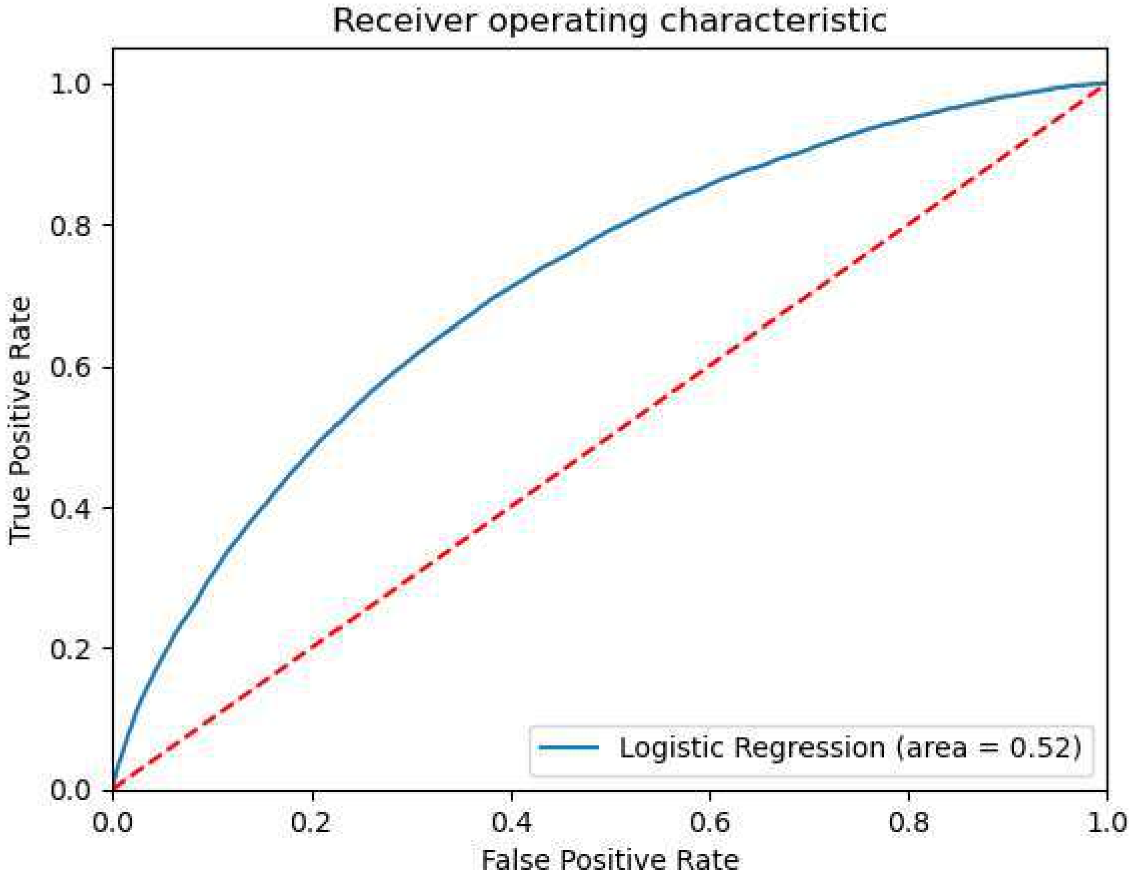
```
[[84992   896]
 [19360  1108]]
```

Classification Report

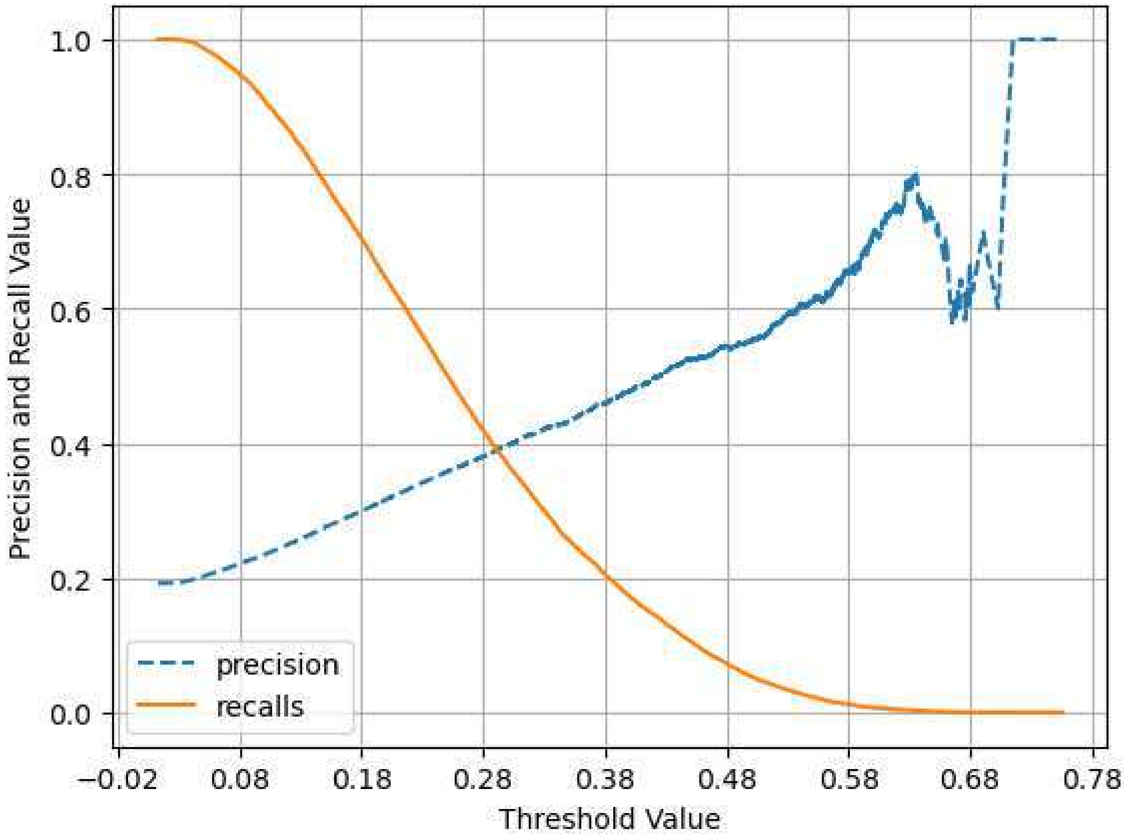
In [57]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.99      0.89     85888
           1       0.55      0.05      0.10     20468

    accuracy                           0.81    106356
   macro avg       0.68      0.52      0.50    106356
weighted avg       0.76      0.81      0.74    106356
```

In [58]:
```
logit_roc_auc = roc_auc_score(y_test, logreg.predict(x_test))
fpr, tpr, thresholds = roc_curve(y_test, logreg.predict_proba(x_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



In [61]:
```
def precision_recall_curve_plot(y_test, pred_proba_c1):
    precisions, recalls, thresholds = precision_recall_curve(y_test, pred_proba_c1)

    threshold_boundary = thresholds.shape[0]
    # plot precision
    plt.plot(thresholds, precisions[0:threshold_boundary], linestyle='--', label='prec
    # 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, logreg.predict_proba(x_test)[:,1])
```



Multicollinearity check using VIF

In [63]:
```
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)[:5]
```

Out[63]:

	Feature	VIF
34	application_type_INDIVIDUAL	150.28
2	int_rate	122.82
14	purpose_debt_consolidation	51.00
1	term	27.24
13	purpose_credit_card	18.48

In [65]:
```
x.drop(columns=['application_type_INDIVIDUAL'], axis=1, inplace=True)
calc_vif(x)[:5]
```

Out[65]:

	Feature	VIF
2	int_rate	102.51
14	purpose_debt_consolidation	26.39
1	term	24.09
5	open_acc	13.74
9	total_acc	12.68

In [67]:
```
x.drop(columns=['int_rate'], axis=1, inplace=True)
calc_vif(x)[:5]
```

Out[67]:

	Feature	VIF
1	term	23.04
13	purpose_debt_consolidation	20.74
4	open_acc	13.62
8	total_acc	12.68
7	revol_util	9.04

In [69]:
```
x.drop(columns=['term'], axis=1, inplace=True)
calc_vif(x)[:5]
```

Out[69]:

	Feature	VIF
12	purpose_debt_consolidation	16.15
3	open_acc	13.62
7	total_acc	12.65
6	revol_util	9.03
1	annual_inc	8.01

In [70]:
```
x.drop(columns=['purpose_debt_consolidation'], axis=1, inplace=True)
calc_vif(x)[:5]
```

Out[70]:

	Feature	VIF
3	open_acc	12.92
7	total_acc	12.63
6	revol_util	8.11
1	annual_inc	7.58
2	dti	7.45

In [72]:
```
x.drop(columns=['open_acc'], axis=1, inplace=True)
calc_vif(x)[:5]
```

Out[72]:

	Feature	VIF
6	total_acc	8.14
5	revol_util	7.85
1	annual_inc	7.46
2	dti	6.81
0	loan_amnt	6.69

In [74]:
```
X = scaler.fit_transform(x)

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

```
Cross Validation accuracy: 0.809
```

Oversampling Using SMOTE

In [77]:
```
sm = SMOTE(random_state=42)
x_train_res, y_train_res = sm.fit_resample(x_train, y_train.ravel())
```

In [79]:
```
print('After OverSampling, the shape of train_X: {}'.format(x_train_res.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train_res.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_train_res == 1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train_res == 0)))
```

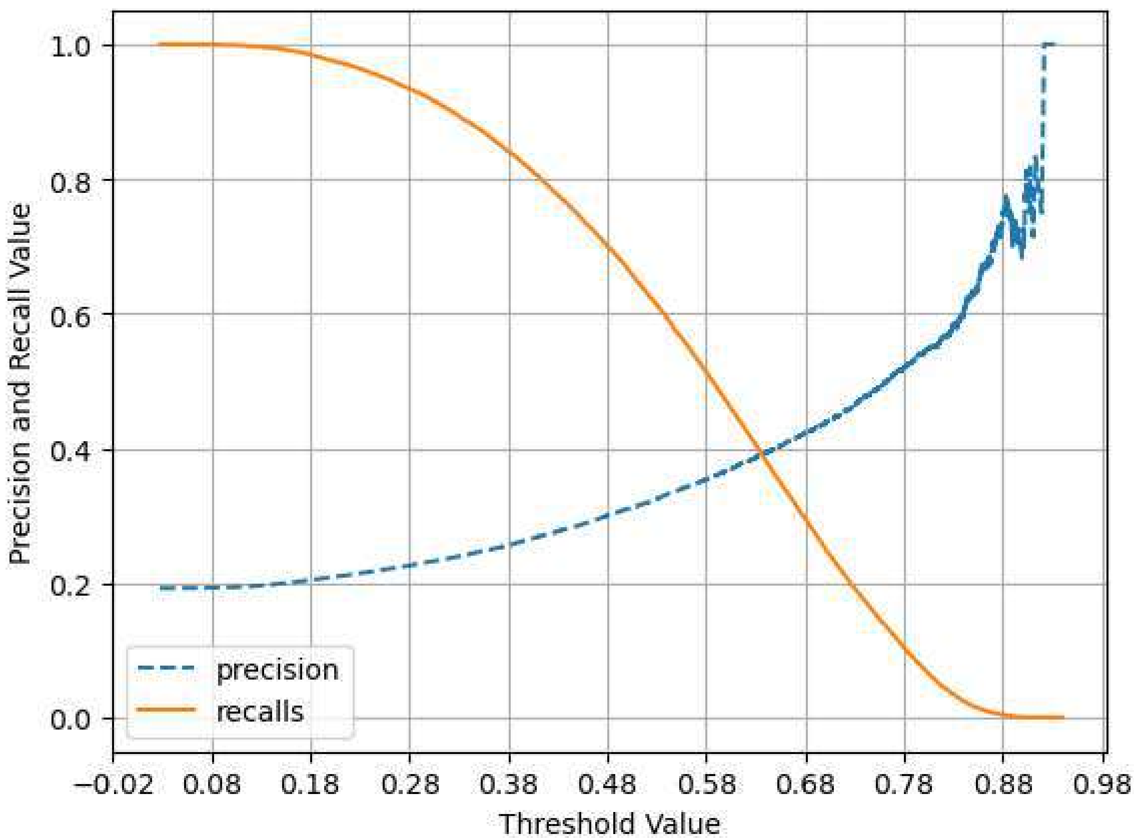
```
After OverSampling, the shape of train_X: (400810, 39)
After OverSampling, the shape of train_y: (400810,)

After OverSampling, counts of label '1': 200405
After OverSampling, counts of label '0': 200405
```

In [82]:
```
lr1 = LogisticRegression(max_iter=1000)
lr1.fit(x_train_res, y_train_res)
predictions = lr1.predict(x_test)

# Classification Report
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.89      0.64      0.75     85888
           1       0.31      0.67      0.42     20468

    accuracy                           0.65    106356
   macro avg       0.60      0.66      0.59    106356
weighted avg       0.78      0.65      0.69    106356
```

In [83]:
```
def precision_recall_curve_plot(y_test, pred_proba_c1):
    precisions, recalls, thresholds = precision_recall_curve(y_test, pred_proba_c1)

    threshold_boundary = thresholds.shape[0]
    # plot precision
```

```
    plt.plot(thresholds, precisions[0:threshold_boundary], linestyle='--', label='prec
    # 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, lr1.predict_proba(x_test)[:,1])
```



In [ ]: