https://drive.google.com/file/d/1JOaZVdZpa-vkBJafSgnLVSEgVD1g0y4e/view?usp=drive_link

In [17]:
```
!gdown 1JOaZVdZpa-vkBJafSgnLVSEgVD1g0y4e
```
...

In [163]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
df  = pd.read_csv("logistic_regression.csv")
```

In [212]:
```
df1=pd.read_csv("logistic_regression.csv")
```

In [20]:
```
df.head()
```

Out[20]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_(
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	M(
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	M(

5 rows × 27 columns

In [3]:
```
df.shape
```

Out[3]:
```
(396030, 27)
```

In [22]:
```
df.info()
```
...

Datatype Conversion

In [164]:
```
df["issue_d"] = pd.to_datetime(df["issue_d"])
df["earliest_cr_line"] = pd.to_datetime(df["earliest_cr_line"])
```

In [6]:
```
df.describe()
```

Out[6]:

	loan_amnt	int_rate	installment	annual_inc	issue_d	
count	396030.000000	396030.000000	396030.000000	3.960300e+05	396030	396030.0(
mean	14113.888089	13.639400	431.849698	7.420318e+04	2014-02-02 15:57:58.045602560	17.3
min	500.000000	5.320000	16.080000	0.000000e+00	2007-06-01 00:00:00	0.0(
25%	8000.000000	10.490000	250.330000	4.500000e+04	2013-05-01 00:00:00	11.2
50%	12000.000000	13.330000	375.430000	6.400000e+04	2014-04-01 00:00:00	16.9
75%	20000.000000	16.490000	567.300000	9.000000e+04	2015-03-01 00:00:00	22.9
max	40000.000000	30.990000	1533.810000	8.706582e+06	2016-12-01 00:00:00	9999.0(
std	8357.441341	4.472157	250.727790	6.163762e+04	NaN	18.0

In [155]:
```
df.isna().sum()
```

...

Feature Engineering

In [18]:
```
# df.drop("mort_acc",axis=1,inplace=True)
```

In [19]:
```
df["loan_status"].value_counts()
```

Out[19]:
```
loan_status
Fully Paid     318357
Charged Off     77673
Name: count, dtype: int64
```

In [35]:
```
df.groupby("loan_status")["loan_amnt"].describe()
```

Out[35]:

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

In [36]:
```
df["home_ownership"].value_counts()
```

Out[36]:
```
home_ownership
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
ANY              3
Name: count, dtype: int64
```

In [165]:
```
df.loc[(df.home_ownership == "NONE") | (df.home_ownership == "ANY"),"home_owne
df["home_ownership"].value_counts()
```

Out[165]:
```
home_ownership
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          146
Name: count, dtype: int64
```

In [40]:
```
df.loc[(df.home_ownership == "OTHER","loan_status")].value_counts()
```

Out[40]:
```
loan_status
Fully Paid     123
Charged Off     23
Name: count, dtype: int64
```

In [49]:
```
df["title"]= df.title.str.lower()
```

In [53]:
```
df.title.value_counts()[:20]
```

...

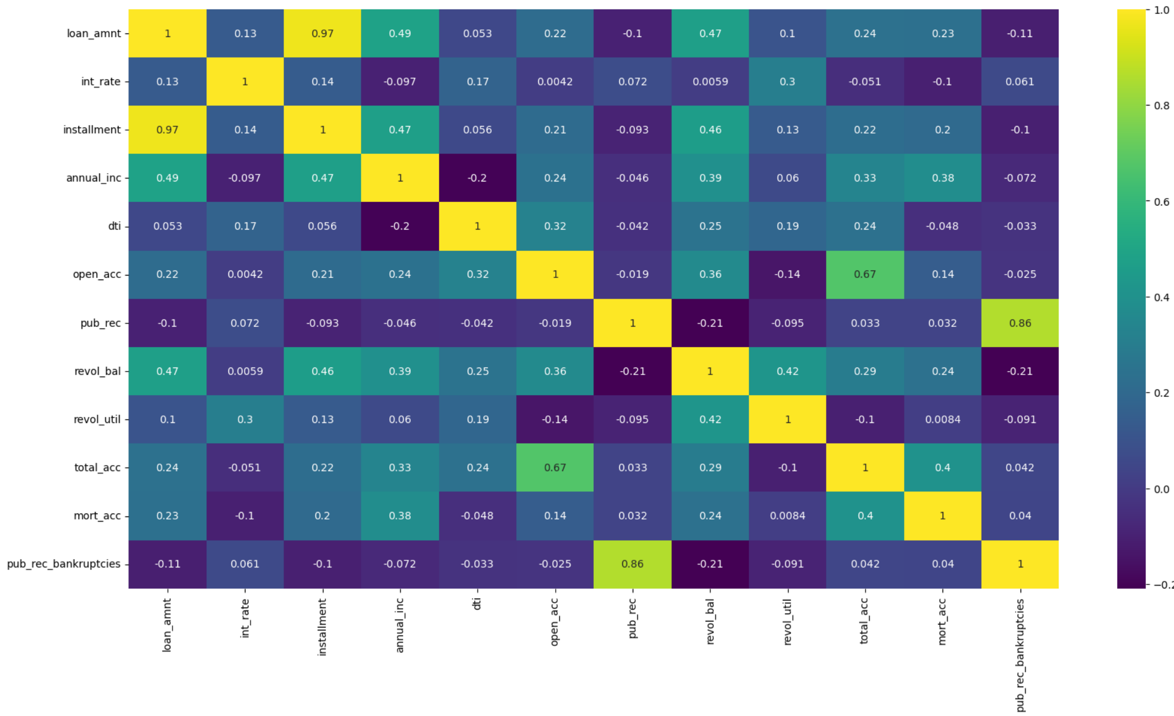
In [48]:
```
df.loc[(df.title == "debt consolidation loan") | (df.title == "consolidation")
df.loc[(df.title == "credit card refinancing") | (df.title == "credit card ref
```

...

In [166]:
```
#since title and purpose columns are both similiar we can drop title column me
df.drop("title", axis=1,inplace=True)
```

In [231]:
```
df_int = df1.select_dtypes(exclude = [object])
```

In [232]:
```
plt.figure(figsize=(20,10))
sns.heatmap(df_int.corr(method="spearman"),annot=True,cmap="viridis")
plt.show()
```



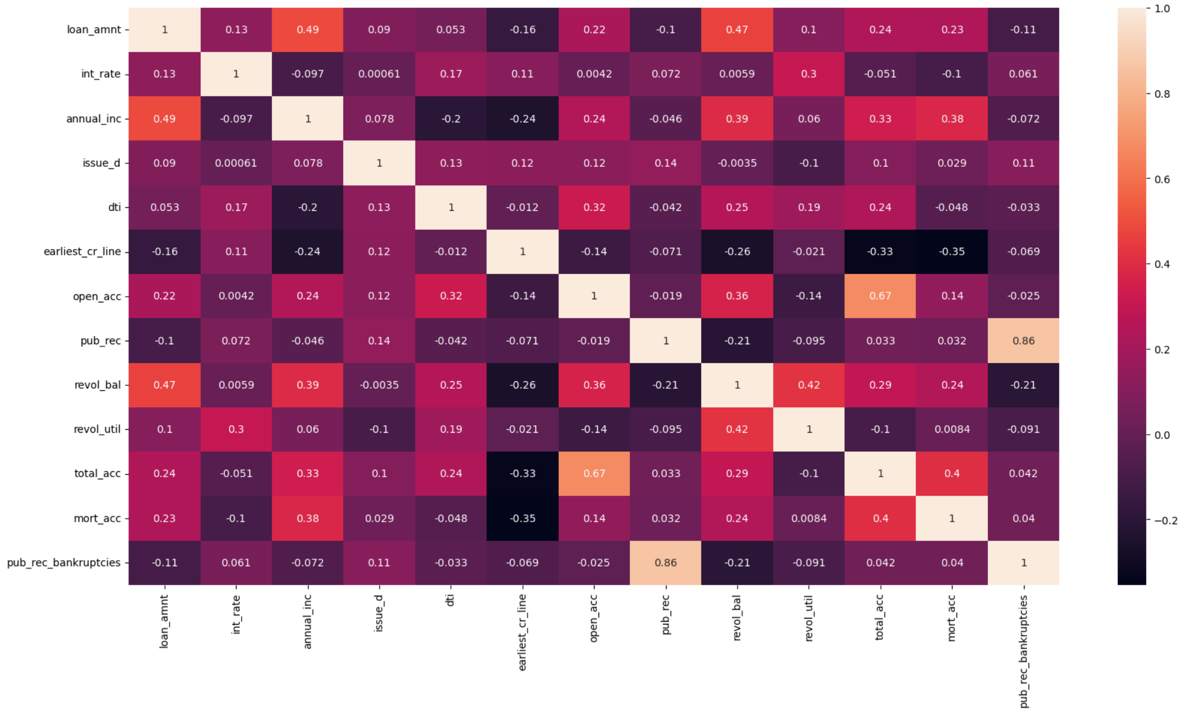
It is observed that correlation betweeen loan_amnt and installment are highly positively correlated.Thus we can remove anyone of them because having both of them increases Dimensionalty of the problem.

In [167]:
```
df.drop("installment",axis = 1,inplace = True)
```

In [80]:
```
df_int.drop("installment",axis = 1,inplace = True)
```

...

In [81]:
```python
plt.figure(figsize=(20,10))
sns.heatmap(df_int.corr(method="spearman"),annot=True)
plt.show()
```

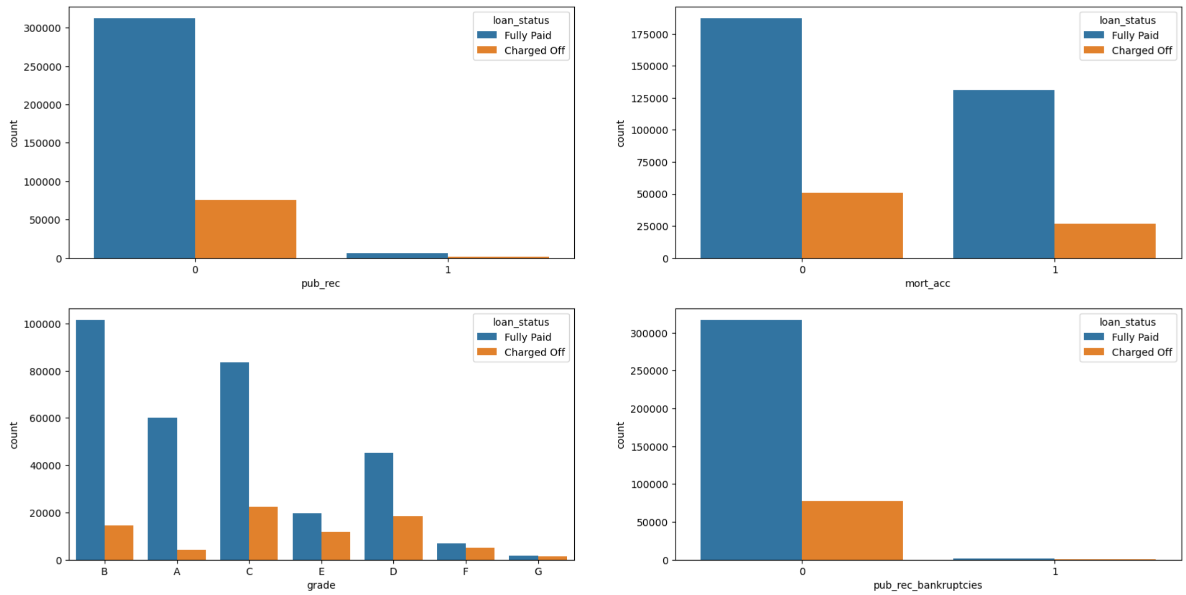


In [168]:
```python
df[["pub_rec","mort_acc","pub_rec_bankruptcies"]] = df[["pub_rec","mort_acc","
```

In [87]:
```python
df.mort_acc.value_counts()
```

Out[87]:
```
mort_acc
0    237988
1    158042
Name: count, dtype: int64
```

In [85]:
```
fig,axs=plt.subplots(nrows=2,ncols=2,figsize=(20,10))

cols=["pub_rec","mort_acc","grade","pub_rec_bankruptcies"]
count=0

for i in range(2):
    for j in range(2):
            sns.countplot(data=df,x=cols[count],ax=axs[i,j],hue="loan_status")
            count +=1

plt.show()
```



In [13]:
```
df.isna().sum()
```

...

In [169]:
```
df.drop("emp_title",axis=1,inplace=True)
```

Dropping emp_title because it seems very complicated column and doesnt yield very important info.

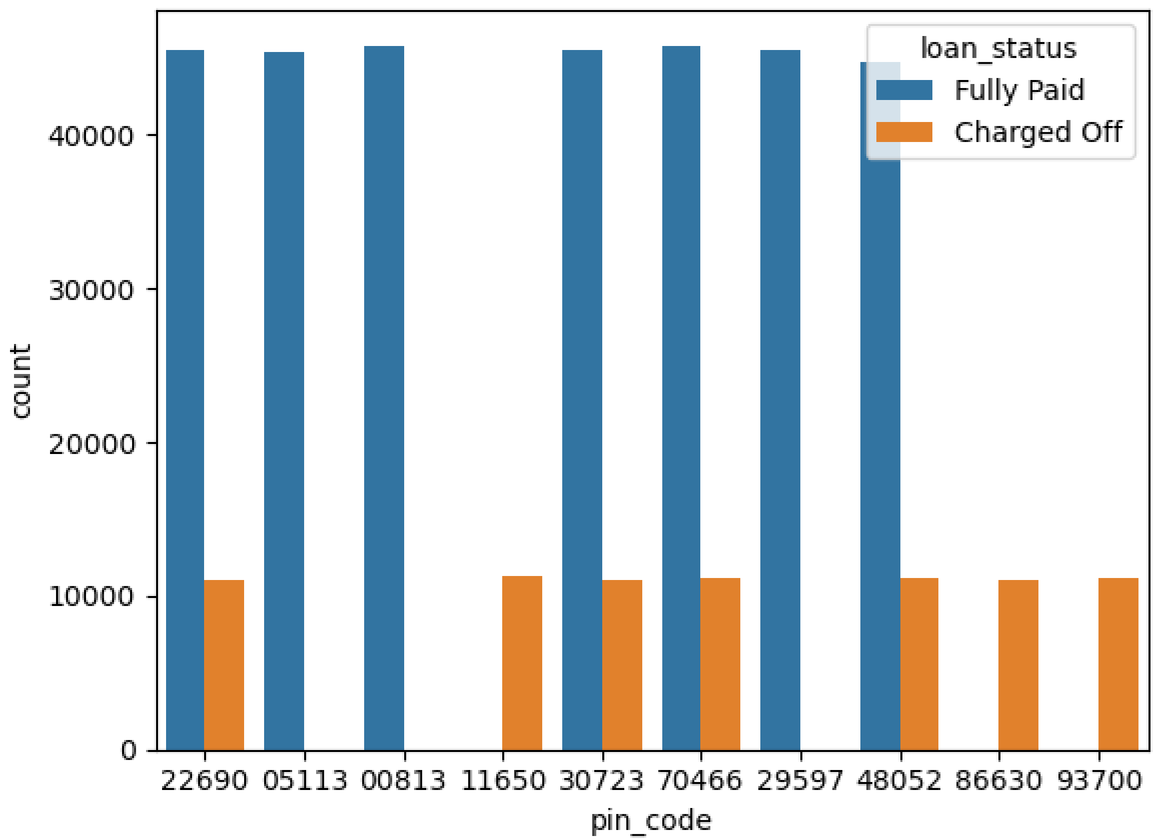
In [233]:
```
df1["pin_code"] = df1["address"].str[-5:]
```

In [234]:
```
df1["pin_code"].value_counts()
```

Out[234]:
```
pin_code
70466    56985
30723    56546
22690    56527
48052    55917
00813    45824
29597    45471
05113    45402
11650    11226
93700    11151
86630    10981
Name: count, dtype: int64
```

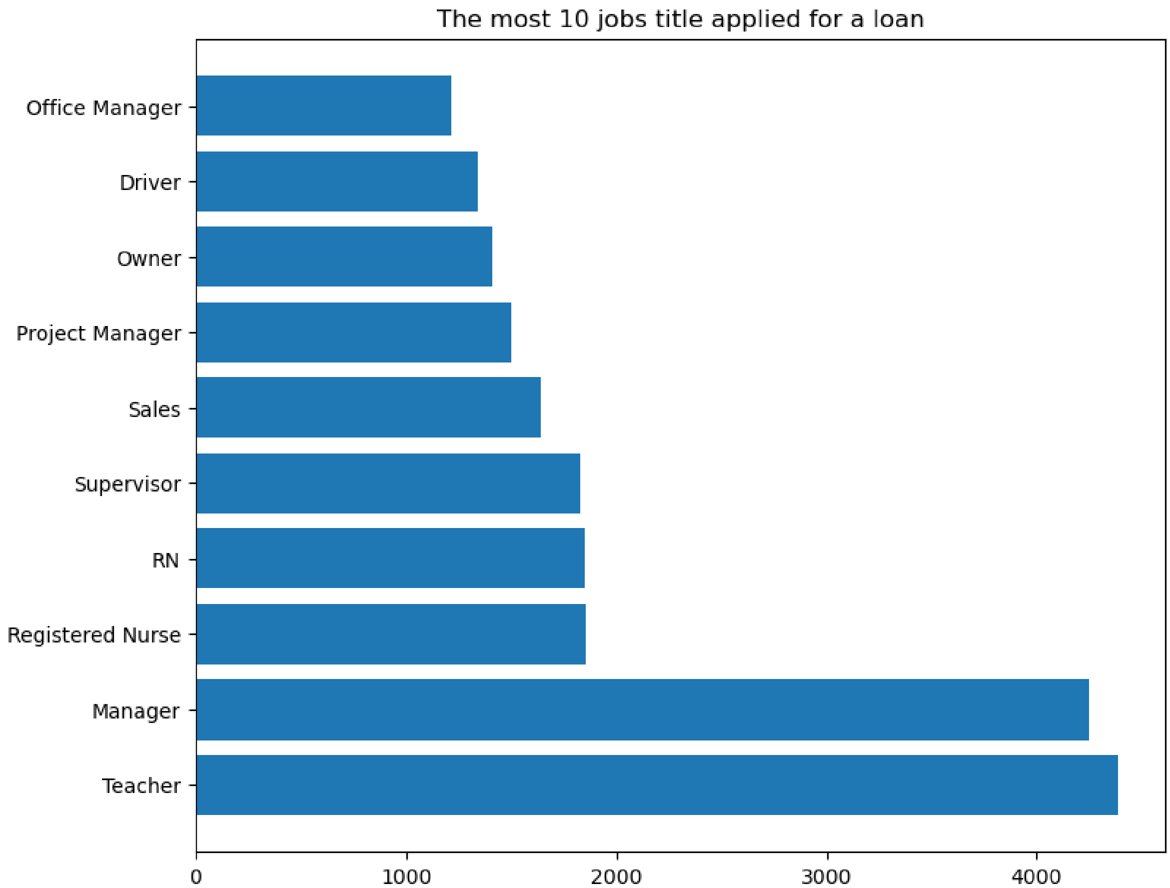
In [236]:
```
sns.countplot(x="pin_code",data=df1,hue="loan_status")
```

Out[236]:
```
<Axes: xlabel='pin_code', ylabel='count'>
```



In [ ]:

In [213]:
```
plt.figure(figsize=(15, 12))
plt.subplot(2, 2, 2)
plt.barh(df1.emp_title.value_counts()[:10].index, df1.emp_title.value_counts()
plt.title("The most 10 jobs title applied for a loan")
plt.tight_layout()
```

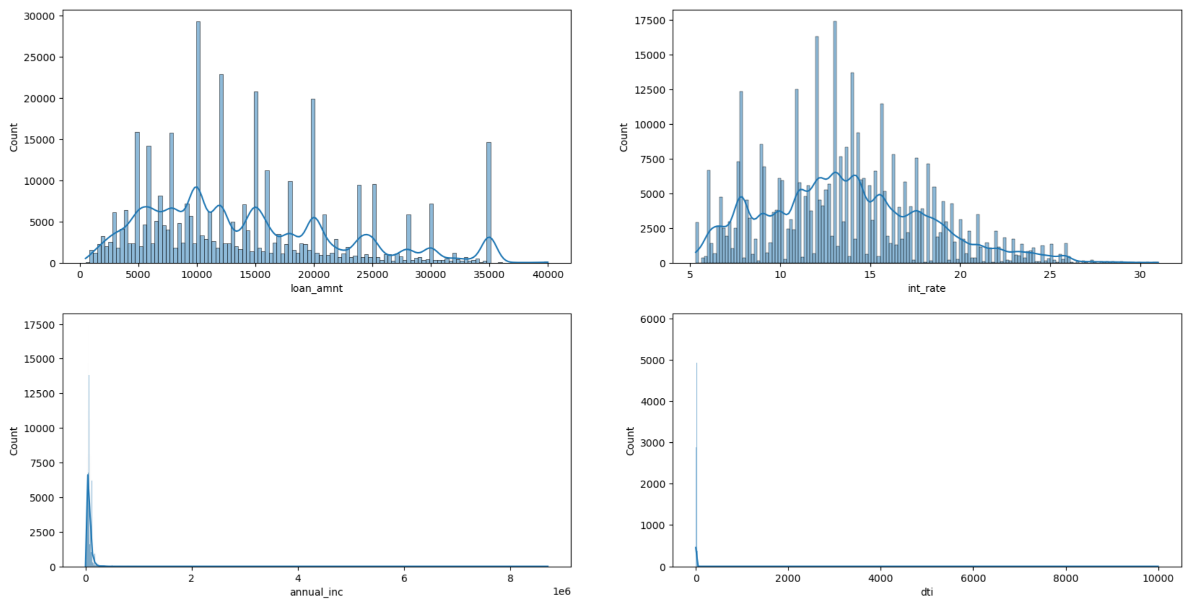


MISSING VALUE IMPUTATION

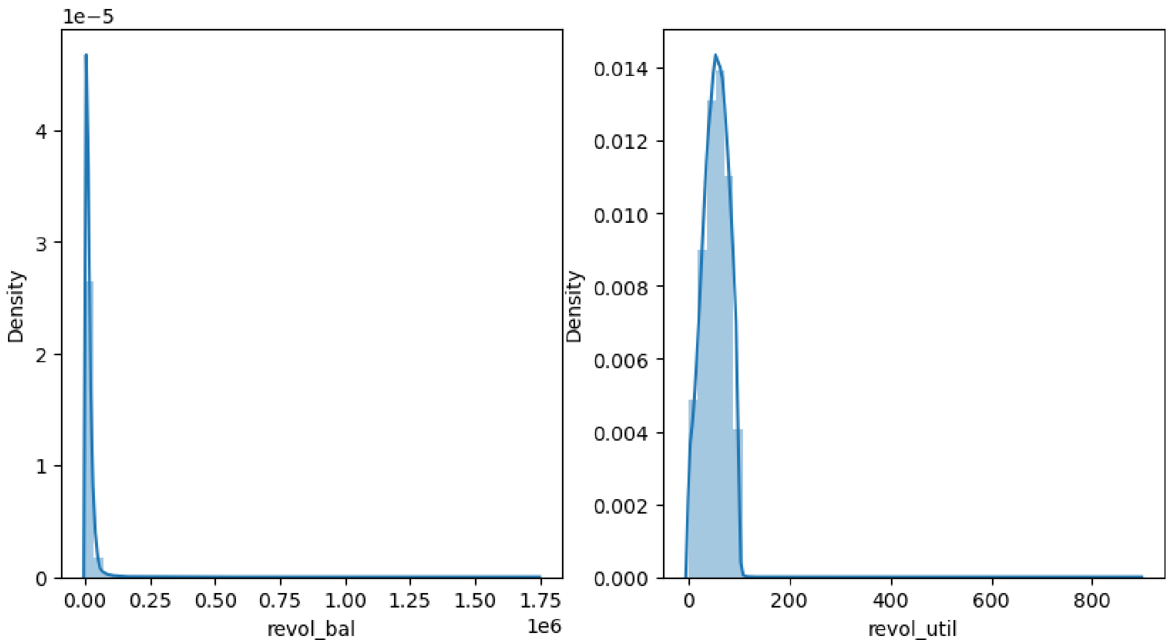
In [170]:
```
from sklearn.impute import SimpleImputer

df[["emp_length"]]=df["emp_length"].values.reshape(-1,1)
df[["emp_length"]]=SimpleImputer(strategy='most_frequent').fit_transform(df[["

df[["revol_util"]]=df["revol_util"].values.reshape(-1,1)
df[["revol_util"]]=SimpleImputer(strategy='mean').fit_transform(df[["revol_uti
```

All the missing values has been imputed using Simple imputer for categorical and numerical features accordingly.

In [282]:
```
fig,axs=plt.subplots(nrows=2,ncols=2,figsize=(20,10))

cols=["loan_amnt","int_rate","annual_inc","dti"]
count=0

for i in range(2):
    for j in range(2):
            sns.histplot(data=df,x=cols[count],kde=True,ax=axs[i,j])
            count +=1

plt.show()


# sns.histplot(data=df,x="int_rate",kde=True)
```

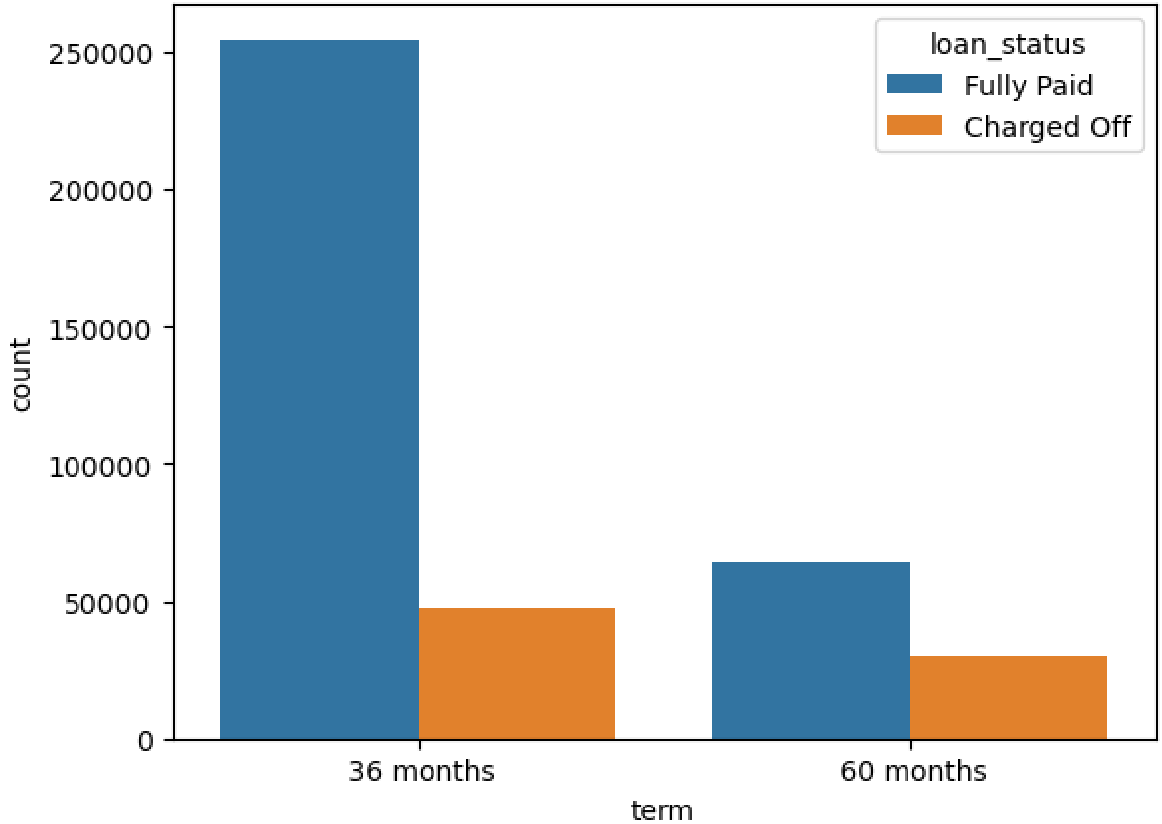


In [68]:
```
fig,axs=plt.subplots(nrows=1,ncols=2,figsize=(10,5))

cols=["revol_bal","revol_util"]
count=0

for i in range(2):
        sns.distplot(df[cols[count]],ax=axs[i])
        count +=1

plt.show()
```



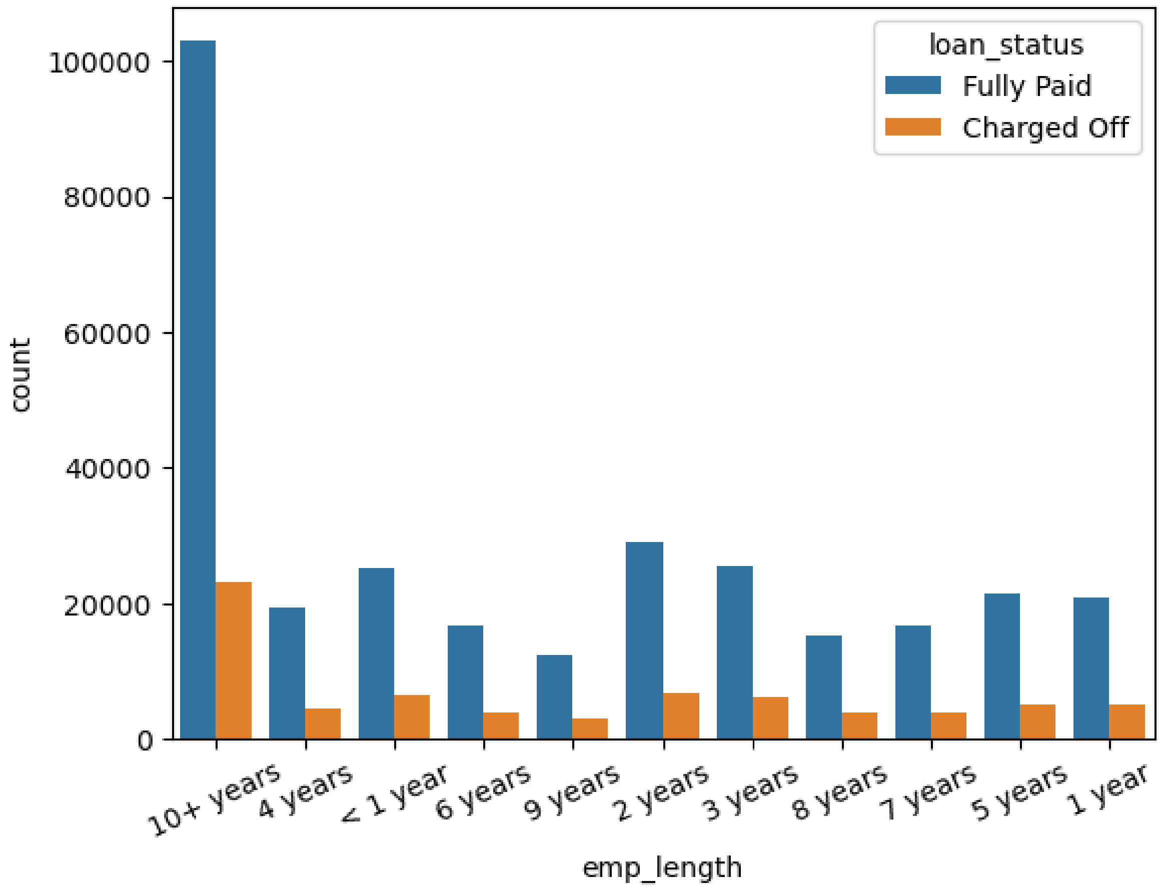
In [60]:
```
sns.countplot(data=df,x="term",hue="loan_status")
```

Out[60]: <Axes: xlabel='term', ylabel='count'>



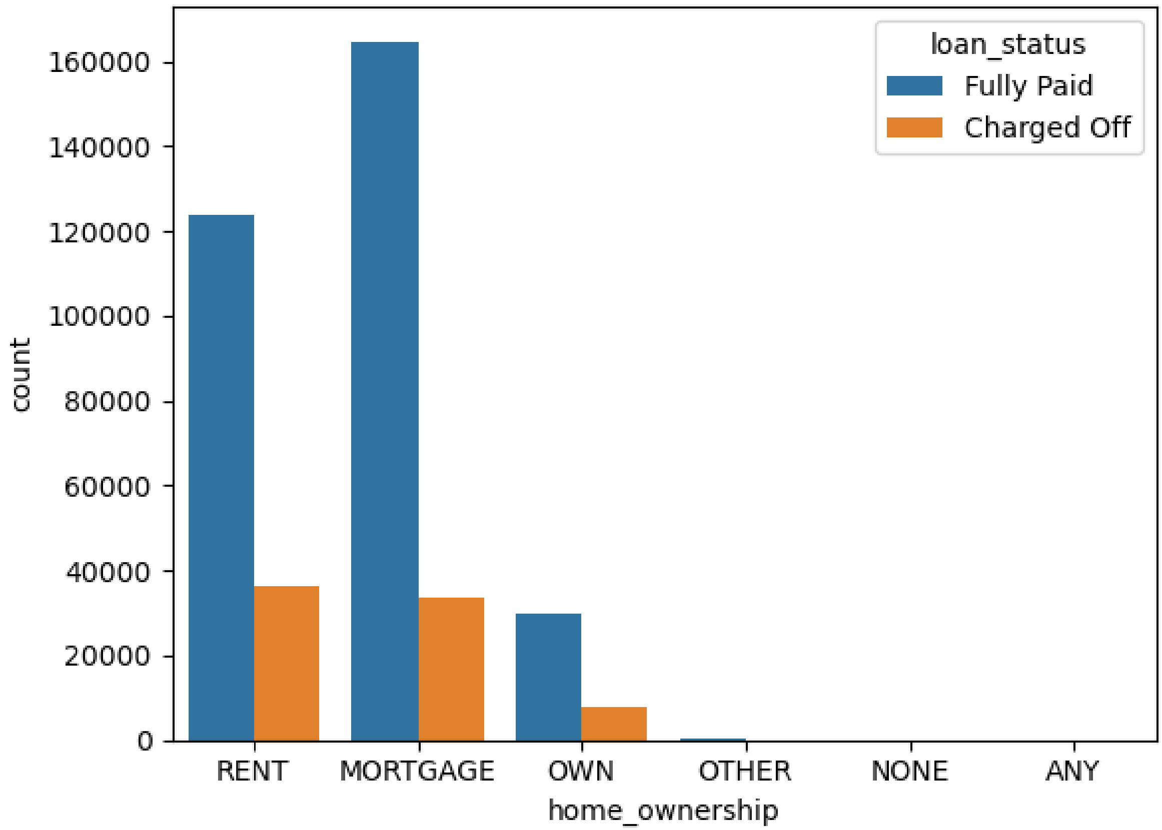
In [63]:
```
sns.countplot(data=df,x="emp_length",hue="loan_status")
plt.xticks(rotation = 25)
```

Out[63]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
 [Text(0, 0, '10+ years'),
  Text(1, 0, '4 years'),
  Text(2, 0, '< 1 year'),
  Text(3, 0, '6 years'),
  Text(4, 0, '9 years'),
  Text(5, 0, '2 years'),
  Text(6, 0, '3 years'),
  Text(7, 0, '8 years'),
  Text(8, 0, '7 years'),
  Text(9, 0, '5 years'),
  Text(10, 0, '1 year')])
```



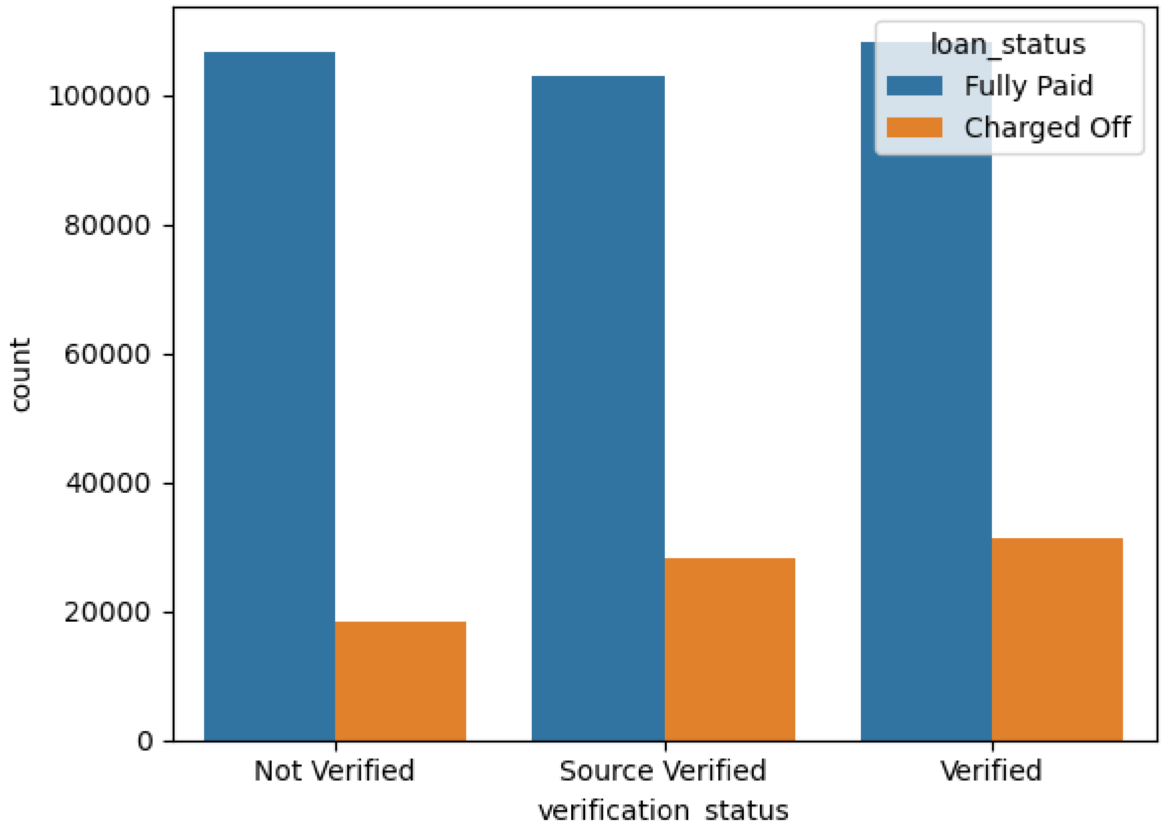
In [156]:
```
sns.countplot(data=df,x="home_ownership",hue="loan_status")
```

Out[156]:
```
<Axes: xlabel='home_ownership', ylabel='count'>
```



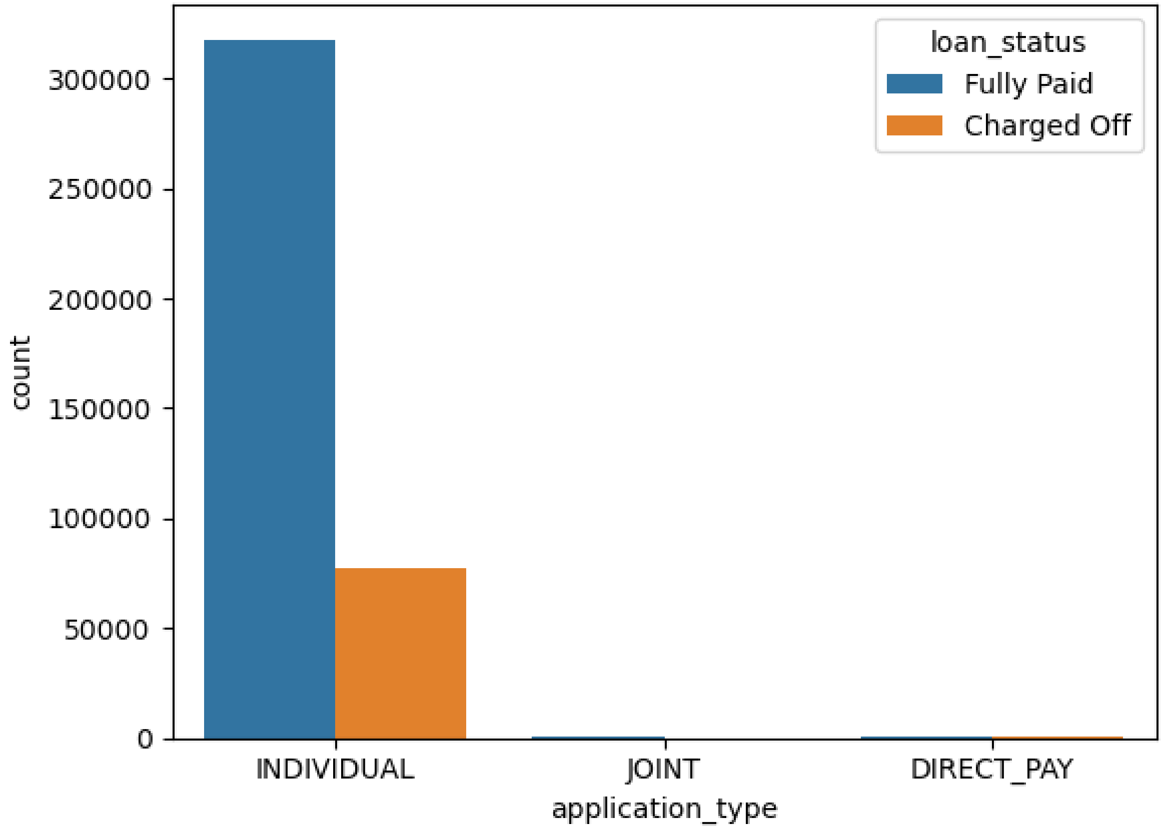
In [65]:
```
sns.countplot(data=df,x="verification_status",hue="loan_status")
```

Out[65]:
```
<Axes: xlabel='verification_status', ylabel='count'>
```



In [14]:
```
sns.countplot(data=df,x="application_type",hue="loan_status")
```

Out[14]: <Axes: xlabel='application_type', ylabel='count'>



In [143]:
```
sns.countplot(data=df,x="loan_status")
```

Out[143]:
```
<Axes: xlabel='loan_status', ylabel='count'>
```

In [148]:
```
df["address"].value_counts()
```

...

There are 2330 duplicate addresses

In [154]:
```
df[df["address"] == "USCGC Williams\r\nFPO AP 00813"]
```

...

Outlier Detection Treatment

In [232]:
```
df_num.columns
```

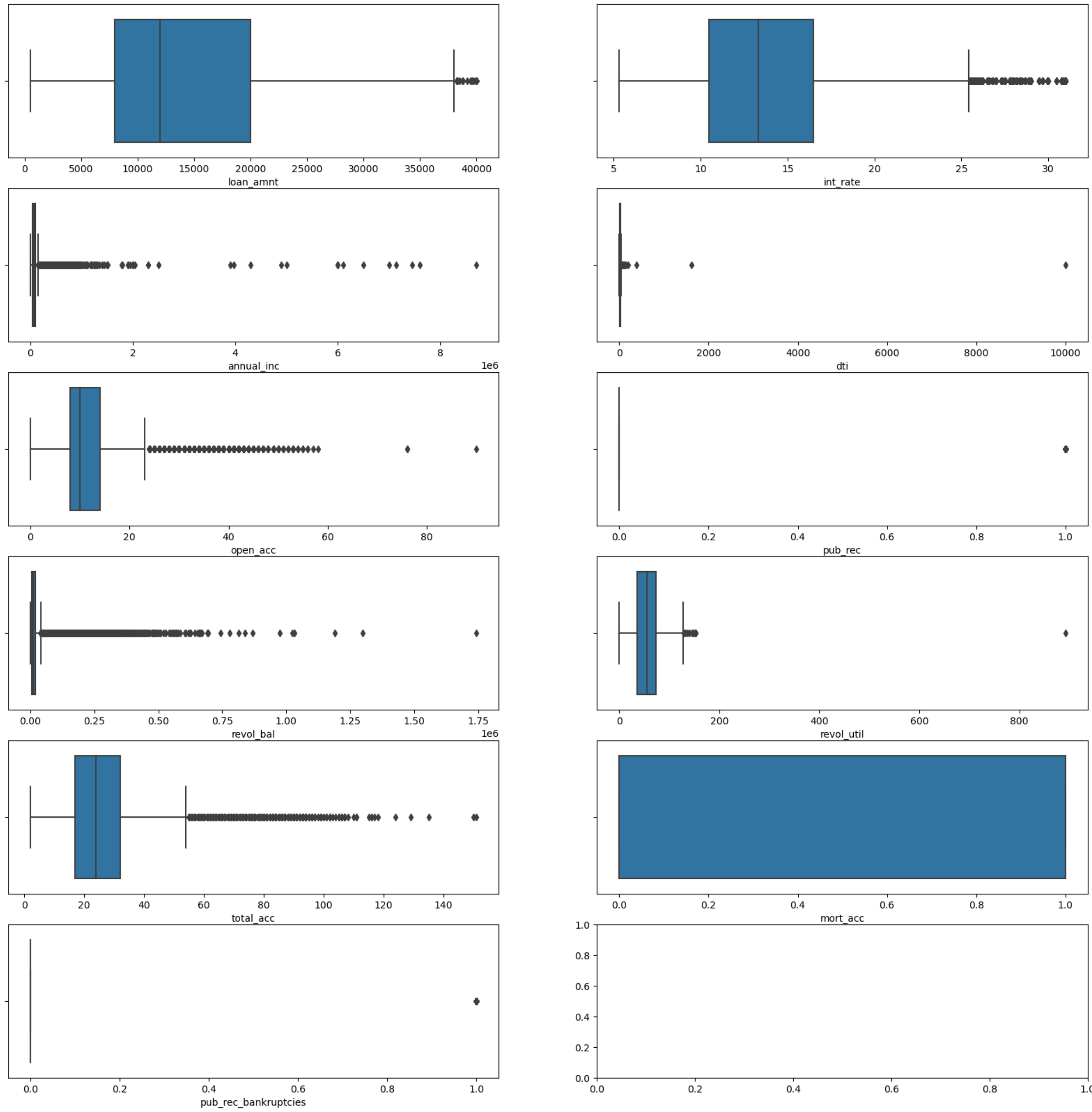
Out[232]:
```
Index(['loan_amnt', 'int_rate', 'annual_inc', 'dti', 'open_acc', 'pub_rec',
       'revol_bal', 'revol_util', 'total_acc', 'mort_acc',
       'pub_rec_bankruptcies'],
      dtype='object')
```

In [171]:
```
df_num = df.select_dtypes(include = "number")
df_num.shape
```

Out[171]:
```
(396030, 11)
```

In [97]:
```
fig,axs=plt.subplots(nrows=6,ncols=2,figsize=(20,20))

cols = df_num.columns
count=0

for i in range(6):
    for j in range(2):
            sns.boxplot(data=df_num,x=cols[count],ax=axs[i,j])
            count +=1

plt.show()
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[97], line 8
      6 for i in range(6):
      7     for j in range(2):
----> 8             sns.boxplot(data=df_num,x=cols[count],ax=axs[i,j])
      9             count +=1
     11 plt.show()

File ~\anaconda3\Lib\site-packages\pandas\core\indexes\base.py:5175, in Inde
x.__getitem__(self, key)
   5172 if is_integer(key) or is_float(key):
   5173     # GH#44051 exclude bool, which would return a 2d ndarray
   5174     key = com.cast_scalar_indexer(key)
-> 5175     return getitem(key)
   5177 if isinstance(key, slice):
   5178     # This case is separated from the conditional above to avoid
   5179     # pessimization com.is_bool_indexer and ndim checks.
   5180     result = getitem(key)

IndexError: index 11 is out of bounds for axis 0 with size 11
```



In [48]:
```python
# outlier_col = ["loan_amnt","int_rate","annual_inc","dti","open_acc","revol_b

for col in df_num.columns:
    Q3 = df[col].quantile(0.75)
    Q1 = df[col].quantile(0.25)

    IQR = Q3 - Q1

    lower_range = Q1 - 10 * IQR
    upper_range = Q3 + 10 * IQR

    df_new = df[(df[col] > lower_range) | (df[col] < upper_range)]

#     df = df.drop(outliers.index)
df_new.shape
```

Out[48]:
```
(2325, 24)
```

In [49]:
```
df.shape
```
Out[49]:
```
(396030, 24)
```

In [50]:
```
df.columns
```
...

In [256]:
```
sns.histplot(x="loan_amnt",data=df_num,kde=True)
```
...

In [343]:
```
df.sub_grade.unique()
```
Out[343]:
```
array(['B4', 'B5', 'B3', 'A2', 'C5', 'C3', 'A1', 'B2', 'C1', 'A5', 'E4',
       'A4', 'A3', 'D1', 'C2', 'B1', 'D3', 'D5', 'D2', 'E1', 'E2', 'E5',
       'F4', 'E3', 'D4', 'G1', 'F5', 'G2', 'C4', 'F1', 'F3', 'G5', 'G4',
       'F2', 'G3'], dtype=object)
```

In [172]:
```
df2 = df.copy()
```

In [130]:
```
df2.head()
```
...

In [131]:
```
df.head()
```
Out[131]:

	loan_amnt	term	int_rate	grade	home_ownership	annual_inc	verification_status	loan_statu
0	10000.0	36	11.44	B	RENT	117000.0	Not Verified	
1	8000.0	36	11.99	B	MORTGAGE	65000.0	Not Verified	
2	15600.0	36	10.49	B	RENT	43057.0	Source Verified	
3	7200.0	36	6.49	A	RENT	54000.0	Not Verified	
4	24375.0	60	17.27	C	MORTGAGE	55000.0	Verified	(

In [210]:
```
fully_paid_percentage = (df['loan_status'] == 1).mean() * 100
fully_paid_percentage
```
Out[210]:
```
80.38709188697825
```

In [211]:
```
df.home_ownership.value_counts()
```
Out[211]:
```
home_ownership
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          146
Name: count, dtype: int64
```

Data preparation for modeling

In [173]:
```
# Removing complex features

df.drop(columns = ["sub_grade","emp_length","issue_d","purpose","earliest_cr_l
```

Converting strings to numerical features for better classification

In [174]:
```
df["term"] = df["term"].str.strip()
df["term"] = df.term.map({"36 months" : 36,"60 months" : 60})
```

In [175]:
```
# Target Variable

df["loan_status"] = df.loan_status.map({"Fully Paid": 1 , "Charged Off" : 0})
```

In [138]:
```
df.head()
```

Out[138]:

	loan_amnt	term	int_rate	grade	home_ownership	annual_inc	verification_status	loan_statu
0	10000.0	36	11.44	B	RENT	117000.0	Not Verified	
1	8000.0	36	11.99	B	MORTGAGE	65000.0	Not Verified	
2	15600.0	36	10.49	B	RENT	43057.0	Source Verified	
3	7200.0	36	6.49	A	RENT	54000.0	Not Verified	
4	24375.0	60	17.27	C	MORTGAGE	55000.0	Verified	

In [139]:
```
df.isna().sum()
```

...

In [160]:
```
df.grade.unique()
```

Out[160]:
```
array(['B', 'A', 'C', 'E', 'D', 'F', 'G'], dtype=object)
```

In [176]:
```
# OHE

cat_cols = ["grade","home_ownership","verification_status","application_type",

df_encod = pd.get_dummies(df,columns = cat_cols,drop_first = True).astype(int)
```

In [ ]:

In [177]:
```
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
df_encod.head()
```

Out[177]:

	loan_amnt	term	int_rate	annual_inc	loan_status	dti	open_acc	pub_rec	revol_bal	revol_ı
0	10000	36	11	117000	1	26	16	0	36369	
1	8000	36	11	65000	1	22	17	0	20131	
2	15600	36	10	43057	1	12	13	0	11987	
3	7200	36	6	54000	1	2	6	0	5472	
4	24375	60	17	55000	0	33	13	0	24584	

In [83]:
```
df_encod.shape
```

Out[83]: (396030, 27)

In [19]:
```
df_encod.initial_list_status_w.unique()
```

Out[19]: array([1, 0])

Train-Test split

In [178]:
```
from sklearn.model_selection import train_test_split

x = df_encod.drop("loan_status",axis=1)
y = df_encod["loan_status"]
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.3,random_st
```

In [90]:
```
x.shape , y.shape
```

Out[90]: ((396030, 26), (396030,))

In [351]:
```
df
```

...

Scaling - Using MinMaxScaler or StandardScaler

In [179]:
```
from sklearn.preprocessing import MinMaxScaler


scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test  = scaler.transform(x_test)
```

In [180]:
```
from sklearn.linear_model import LogisticRegression

log = LogisticRegression()
log.fit(x_train,y_train)
```

Out[180]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [181]:
```
y_pred = log.predict(x_test)
```

Display model coefficients with column names

In [182]:
```
coefficients = pd.DataFrame({'x_Column': x.columns, 'Coefficient': log.coef_[0
print(coefficients)
```

```
                          x_Column  Coefficient
0                        loan_amnt    -0.153768
1                             term    -0.421214
2                         int_rate    -0.226503
3                       annual_inc     9.300367
4                              dti    -2.268563
5                         open_acc    -2.862152
6                          pub_rec    -0.129779
7                        revol_bal     4.364854
8                       revol_util    -3.833235
9                        total_acc     1.252089
10                        mort_acc     0.103169
11            pub_rec_bankruptcies     0.021372
12                         grade_B    -0.599296
13                         grade_C    -1.074705
14                         grade_D    -1.400512
15                         grade_E    -1.628421
16                         grade_F    -1.730096
17                         grade_G    -1.808729
18            home_ownership_OTHER    -0.088492
19              home_ownership_OWN    -0.213878
20             home_ownership_RENT    -0.293033
21  verification_status_Source Verified    -0.173871
22         verification_status_Verified    -0.146263
23          application_type_INDIVIDUAL    -0.814317
24               application_type_JOINT     1.335533
25                initial_list_status_w    -0.054647
```

In [226]:
```
df.corr(method = "spearman")
```

...

In [183]:
```
df.loan_status.value_counts()
```

...

Finding the model coefficient and its corresponding feature importance inorder to understand its contribution to label / output of the model.annual_inc ,revol_bal & total_acc,application_type_JOINT features has high feature importance.

dti,open_acc, revol_util has not much feature importance and can be considered to be removed.

In [184]:
```
from sklearn.metrics import accuracy_score,classification_report,confusion_mat

accuracy_score(y_test,y_pred)
```

Out[184]: 0.8038027422164987

In [219]:
```
conf_matrix = confusion_matrix(y_test, y_pred)
conf_matrix
```

Out[219]:
```
array([[  687, 22683],
       [  627, 94812]], dtype=int64)
```

In [220]:
```
total_instances = conf_matrix.sum()
```

In [223]:
```
percentage_true_positives = (conf_matrix[1, 1] / total_instances) * 100
percentage_false_negatives = (conf_matrix[1, 0] / total_instances) * 100
percentage_false_positives = (conf_matrix[0, 1] / total_instances) * 100
percentage_true_negatives = (conf_matrix[0, 0] / total_instances) * 100
```

In [224]:
```
print(f"Percentage True Positives: {percentage_true_positives:.2f}%")
print(f"Percentage False Negatives: {percentage_false_negatives:.2f}%")
print(f"Percentage False Positives: {percentage_false_positives:.2f}%")
print(f"Percentage True Negatives: {percentage_true_negatives:.2f}%")
```

```
Percentage True Positives: 79.80%
Percentage False Negatives: 0.53%
Percentage False Positives: 19.09%
Percentage True Negatives: 0.58%
```

Though Correct predictions are more than false but having 22680 False Positives and 626 False Negatives is not a acceptable.THus leads to poor model.

In [186]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.52      0.03      0.06     23370
           1       0.81      0.99      0.89     95439

    accuracy                           0.80    118809
   macro avg       0.66      0.51      0.47    118809
weighted avg       0.75      0.80      0.73    118809
```

It is observed that there is class imbalance as class 1 has good precision ,recall & f1-score while class 0 has very poor metric scores.Thus we need to implement Oversampling technique.

In [70]:
```
from sklearn.metrics import roc_curve, roc_auc_score
```

In [189]:
```
probability = log.predict_proba(x_test)
probability
```

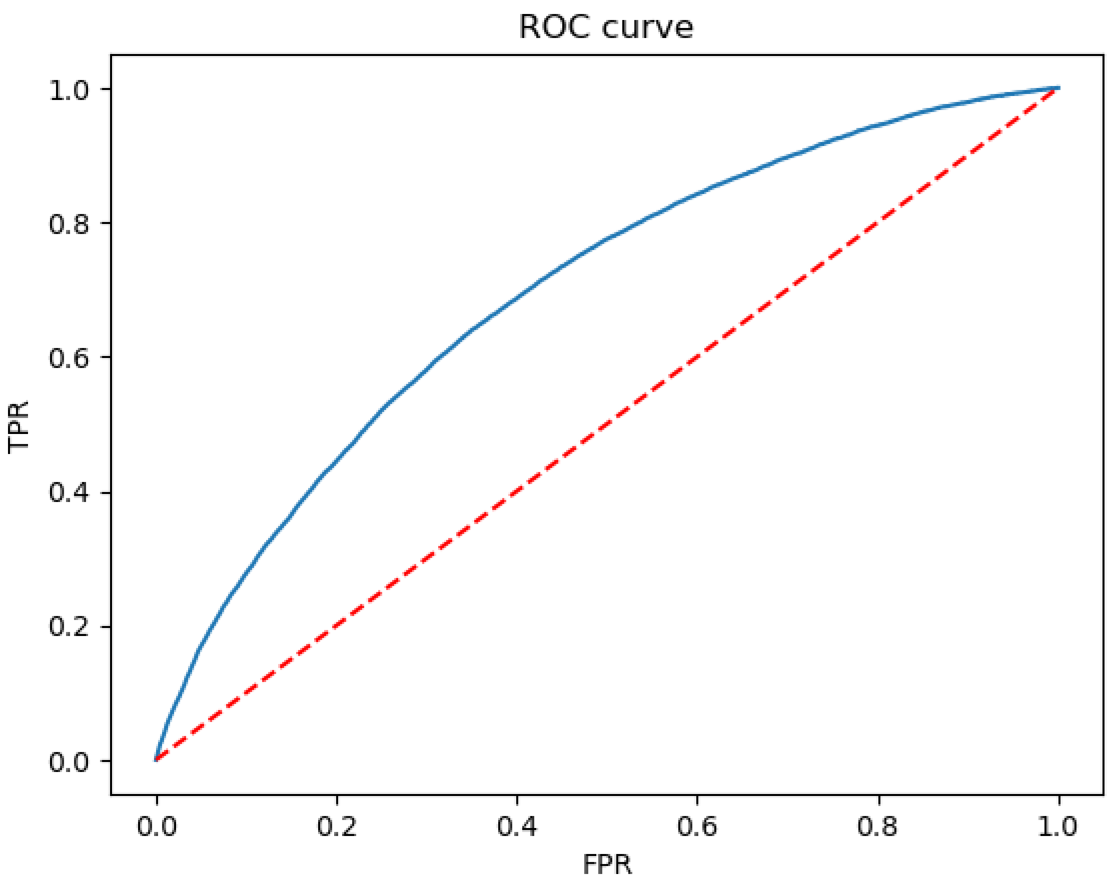
...

Probability variable contains 2 probability P(Y=1|X) and P(Y=0|X)

In [190]:
```
probabilites = probability[:,1]
```

In [191]:
```
fpr, tpr, thr = roc_curve(y_test,probabilites)
```

In [74]:
```
plt.plot(fpr,tpr)

#random model
plt.plot(fpr,fpr,'--',color='red' )
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```



In [192]:
```
roc_auc_score(y_test,probabilites)
```

Out[192]:
```
0.6963554628622602
```

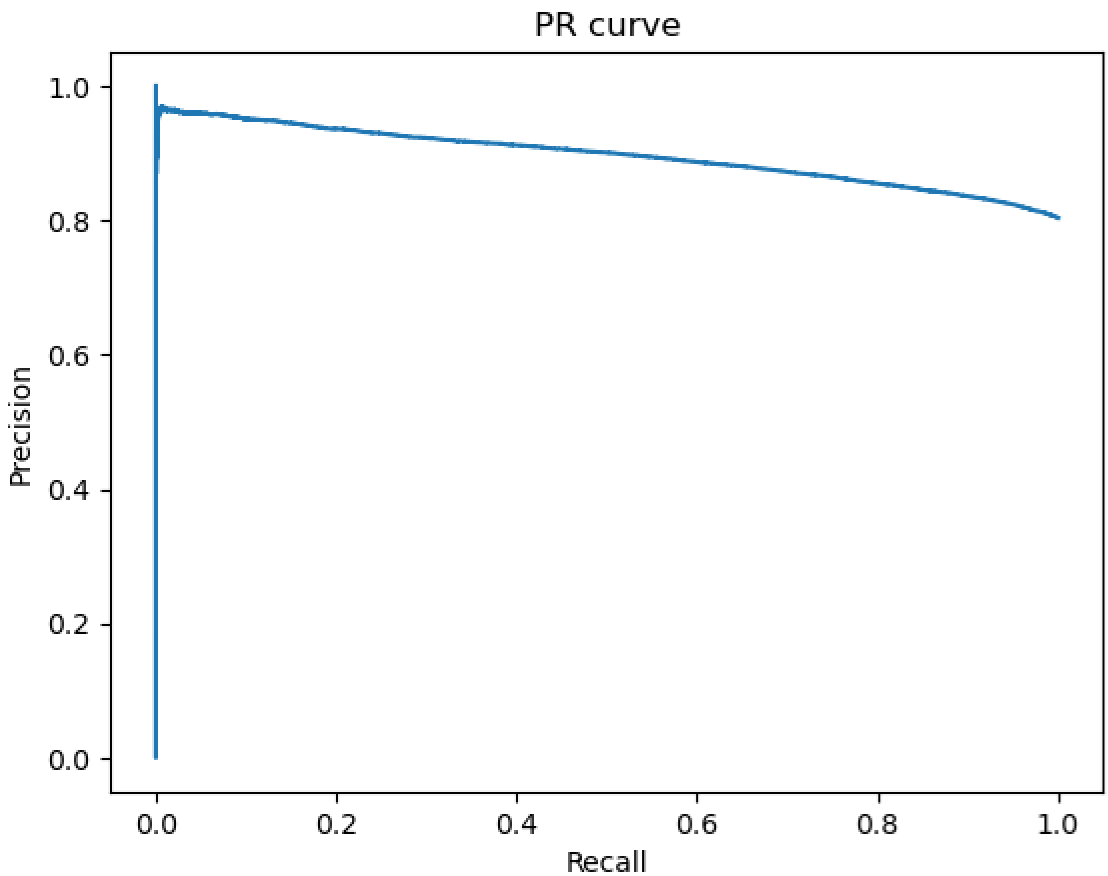
When data is highly imbalanced,

AU-ROC is not prefered since f1_score < AUC_ROC score.

In [76]:
```
from sklearn.metrics import precision_recall_curve,auc
```

In [193]:
```
precision, recall, thr = precision_recall_curve(y_test, probabilites)
```

In [194]:
```
plt.plot(recall, precision)

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('PR curve')
plt.show()
```



In [195]:
```
auc(recall, precision)
```

Out[195]:
```
0.8959216186257071
```

In [ ]:

Now the AU-PRC comes close to F1 score

Showing that PRC worked just fine in imbalanced data

In [ ]:

RandomForestClassifier

In [80]:
```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier()

# fit the predictor and target
rfc.fit(x_train, y_train)

# predict
rfc_predict = rfc.predict(x_test)

# check performance
print('ROCAUC score:',roc_auc_score(y_test, rfc_predict))
print('Accuracy score:',accuracy_score(y_test, rfc_predict))
print('F1 score:',f1_score(y_test, rfc_predict))
```

```
ROCAUC score: 0.5308043194114366
Accuracy score: 0.802666464661768
F1 score: 0.14142893763503867
```

Using Random Forest classifier doeant improve accuracy score etc.

Type *Markdown* and LaTeX: α^2

In [80]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [86]:
```
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).head()
```

Out[86]:

	Feature	VIF
22	application_type_INDIVIDUAL	36.56
1	term	26.56
8	total_acc	11.91
4	open_acc	11.76
7	revol_util	7.18

since VIF of int_rate is very high we can drop it due to multicollinearity.

In [81]:
```
x.drop("int_rate",axis = 1,inplace = True)
```

In [84]:
```
x.shape
```
Out[84]: (396030, 25)

In [87]:
```
calc_vif(x).head()
```
Out[87]:

	Feature	VIF
22	application_type_INDIVIDUAL	36.56
1	term	26.56
8	total_acc	11.91
4	open_acc	11.76
7	revol_util	7.18

In [88]:
```
x.drop("application_type_INDIVIDUAL",axis = 1,inplace = True)
```
In [68]:
```
calc_vif(x).head()
```
Out[68]:

	Feature	VIF
1	term	15.67
8	total_acc	11.80
4	open_acc	11.22
7	revol_util	6.17
12	grade_C	3.06

In [89]:
```
x.drop("term",axis = 1,inplace = True)
```
In [70]:
```
calc_vif(x).head()
```
Out[70]:

	Feature	VIF
7	total_acc	11.56
3	open_acc	11.03
6	revol_util	5.86
11	grade_C	2.72
10	grade_B	2.62

In [90]:
```
x.drop("total_acc",axis = 1,inplace = True)
```

In [72]:
```
calc_vif(x).head()
```

Out[72]:

	Feature	VIF
6	revol_util	5.85
3	open_acc	5.22
10	grade_C	2.72
9	grade_B	2.62
19	verification_status_Verified	2.39

In [92]:
```
df_encod.columns
```

...

In [ ]:
```
VIF score now lies between range 1-5 moderatly collinear and its acceptab
```

model building After eliminating multicollinear features

In [94]:
```
from sklearn.model_selection import train_test_split

x1 = df_encod.drop(columns = ["total_acc","term","application_type_INDIVIDUAL"
y1 = df_encod["loan_status"]
x1_train,x1_test,y1_train,y1_test = train_test_split(x1,y1,test_size = 0.3,ran
```

In [95]:
```
from sklearn.preprocessing import MinMaxScaler


scaler = MinMaxScaler()
x1_train = scaler.fit_transform(x1_train)
x1_test  = scaler.transform(x1_test)
```

In [97]:
```
x1_train.shape,x1_test.shape,y1_train.shape,y1_test.shape
```

Out[97]: ((277221, 22), (118809, 22), (277221,), (118809,))

In [98]:
```
logn = LogisticRegression()
logn.fit(x1_train,y1_train)
```

Out[98]: LogisticRegression()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [99]:
```
y_pred1 = logn.predict(x1_test)
```

In [100]:
```
accuracy_score(y1_test,y_pred1)
```

Out[100]:
```
0.8038532434411535
```

In [101]:
```
print(confusion_matrix(y1_test, y_pred1))
```

```
[[  437 22933]
 [  371 95068]]
```

In [103]:
```
print(classification_report(y1_test, y_pred1))
```

```
              precision    recall  f1-score   support

           0       0.54      0.02      0.04     23370
           1       0.81      1.00      0.89     95439

    accuracy                           0.80    118809
   macro avg       0.67      0.51      0.46    118809
weighted avg       0.75      0.80      0.72    118809
```

In [ ]:
```
Removing Multicollinearity didnot improvise the model performance .
```

In [102]:
```
x.columns
```

...

In [114]:
```
df.loan_status.value_counts()
```

Out[114]:
```
loan_status
1    318357
0     77673
Name: count, dtype: int64
```

In [116]:
```
318357/77673
```

Out[116]:
```
4.098682940017767
```

SMOTE Analysis

In [81]:
```
from imblearn.over_sampling import SMOTE
from collections import Counter

smt = SMOTE()
X_sm, y_sm = smt.fit_resample(x_train, y_train)

print('Resampled dataset shape {}'.format(Counter(y_sm)))
```

```
Resampled dataset shape Counter({0: 222918, 1: 222918})
```

In [82]:
```
log1 = LogisticRegression()

log1.fit(X_sm,y_sm)
```

Out[82]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [83]:
```
predictions = log1.predict(x_test)
```

In [84]:
```
# Classification Report
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.88      0.63      0.73     95439
           1       0.30      0.66      0.42     23370

    accuracy                           0.63    118809
   macro avg       0.59      0.64      0.57    118809
weighted avg       0.77      0.63      0.67    118809
```

In [124]:
```
confusion_matrix(y_test,predictions)
```

Out[124]:
```
array([[15469,  7901],
       [35772, 59667]], dtype=int64)
```

In [82]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.52      0.03      0.06     23370
           1       0.81      0.99      0.89     95439

    accuracy                           0.80    118809
   macro avg       0.66      0.51      0.47    118809
weighted avg       0.75      0.80      0.73    118809
```

In [125]:
```
probability1 = log1.predict_proba(x_test)
probability1
```

...

In [126]:
```
probabilites1 = probability1[:,1]
```

In [128]:
```
fpr1, tpr1, thr1 = roc_curve(y_test,probabilites1)
```

In [127]:
```
precision, recall, thr = precision_recall_curve(y_test, probabilites1)
```

In [129]:
```
plt.plot(fpr1,tpr1)

#random model
plt.plot(fpr1,fpr1,'--',color='red' )
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```

...

In [ ]:

In [ ]:

In [ ]:

In [ ]: