LoanTap_Logistic_Regression

In [ ]:

Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [ ]:

Importing modules or packages for Logistic Regression.

In [2]:
```
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,
    ConfusionMatrixDisplay, RocCurveDisplay
)
from statsmodels.stats.outliers_influence import variance_inflation_factor
from imblearn.over_sampling import SMOTE
```

In [ ]:

Downloading the Dataset

In [3]:
```
df=pd.read_csv('logistic_regression.csv')
df.head()
```

Out[3]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	hom
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	
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	

5 rows × 27 columns

In [ ]:

In [4]:
```
df.shape
```

Out[4]:
```
(396030, 27)
```

In [5]:
```
# Checking the distribution of the outcome labels

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

Out[5]:
```
Fully Paid     80.387092
Charged Off    19.612908
Name: loan_status, dtype: float64
```

In [6]:
```
# Statistical summary of the dataset

df.describe(include='all')
```

Out[6]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title
count	396030.000000	396030	396030.000000	396030.000000	396030	396030	373103
unique	NaN	2	NaN	NaN	7	35	173105
top	NaN	36 months	NaN	NaN	B	B3	Teacher
freq	NaN	302005	NaN	NaN	116018	26655	4389
mean	14113.888089	NaN	13.639400	431.849698	NaN	NaN	NaN
std	8357.441341	NaN	4.472157	250.727790	NaN	NaN	NaN
min	500.000000	NaN	5.320000	16.080000	NaN	NaN	NaN
25%	8000.000000	NaN	10.490000	250.330000	NaN	NaN	NaN
50%	12000.000000	NaN	13.330000	375.430000	NaN	NaN	NaN
75%	20000.000000	NaN	16.490000	567.300000	NaN	NaN	NaN
max	40000.000000	NaN	30.990000	1533.810000	NaN	NaN	NaN

11 rows × 27 columns

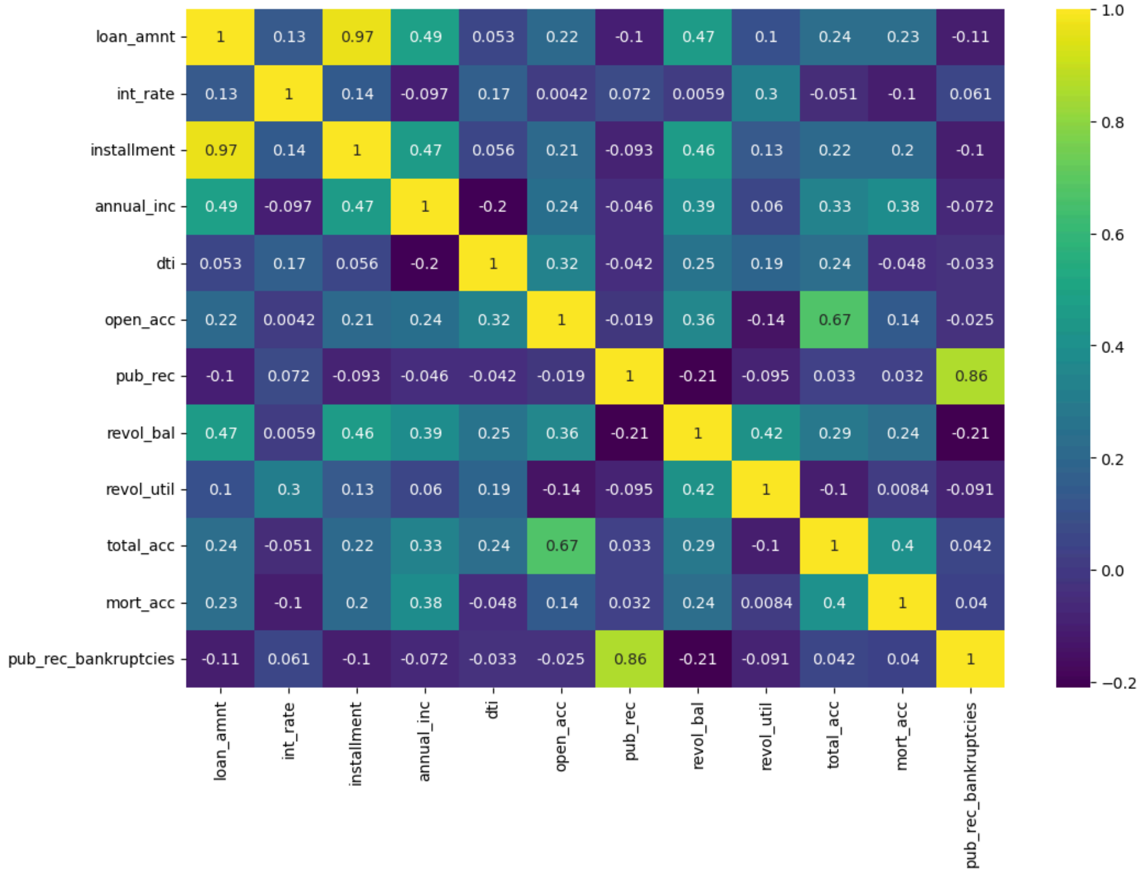
In [ ]:

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  object 
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  object 
 5   sub_grade             396030 non-null  object 
 6   emp_title             373103 non-null  object 
 7   emp_length            377729 non-null  object 
 8   home_ownership        396030 non-null  object 
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  object 
 11  issue_d               396030 non-null  object 
 12  loan_status           396030 non-null  object 
 13  purpose               396030 non-null  object 
 14  title                 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
```

In [ ]:

In [8]:
```
# Correlation Heatmap


plt.figure(figsize=(12,8))
sns.heatmap(df.corr(method='spearman'),annot=True,cmap='viridis')
plt.show()
```



We noticed almost perfect correlation between "loan_amnt" the "installment" feature.

installment: The monthly payment owed by the borrower if the loan originates.

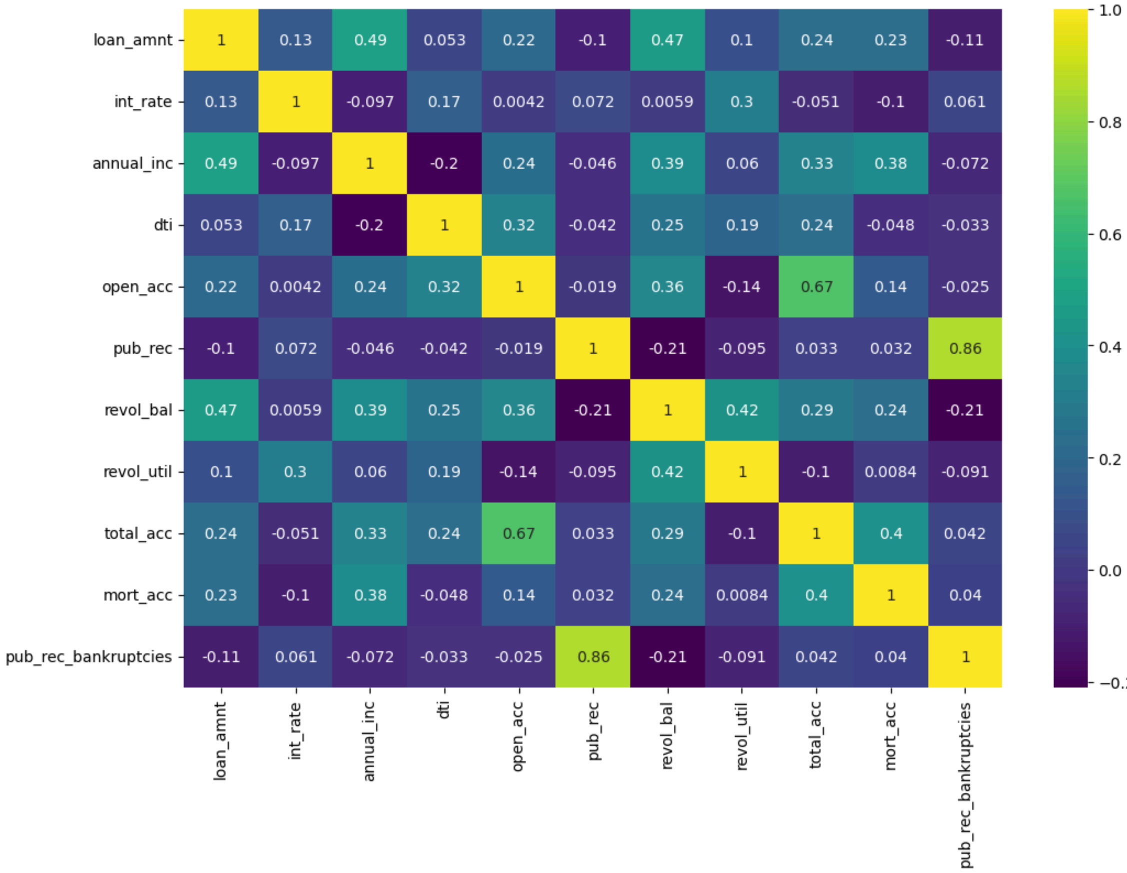
loan_amnt: The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.

So, we can drop either one of those columns.

In [ ]:
```

```

In [9]:
```
## Droppig the Installments column

df.drop(columns=['installment'],axis=1,inplace=True)
```

In [10]:
```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(method='spearman'), annot=True, cmap='viridis')
plt.show()
```



In [ ]:

Data Exploration

In [11]:
```
## No. of people who have paid fully and the no. of people who are charged c

df.groupby(by='loan_status')['loan_amnt'].describe()
```

Out[11]:

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 [ ]:

In [12]:
```
### Majority of ownership


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

Out[12]:
```
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
ANY              3
Name: home_ownership, dtype: int64
```

Majority of ownership is with Mortgage and Rent

In [ ]:

In [14]:
```
## Combining the minority classes as 'OTHERS'

df.loc[(df.home_ownership == 'ANY') | (df.home_ownership == 'NONE'), 'home_o
df.home_ownership.value_counts()
```

Out[14]:
```
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          146
Name: home_ownership, dtype: int64
```

In [15]:
```
df['home_ownership'].value_counts()
```

Out[15]:
```
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          146
Name: home_ownership, dtype: int64
```

In [ ]:

In [16]:
```
## Checking the distribution of 'Other'

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

Out[16]:
```
Fully Paid     123
Charged Off     23
Name: loan_status, dtype: int64
```

In [ ]:

In [17]:
```
### Converting string to date-time format

df['issue_d']=pd.to_datetime(df['issue_d'])
df['earliest_cr_line']=pd.to_datetime(df['earliest_cr_line'])
```

In [ ]:

In [18]:
```
### Some issues in title (It was filled manually and needs some fixing).

df['title'].value_counts()[:20]
```

Out[18]:
```
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 [19]:
```
df['title']=df.title.str.lower()
```

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

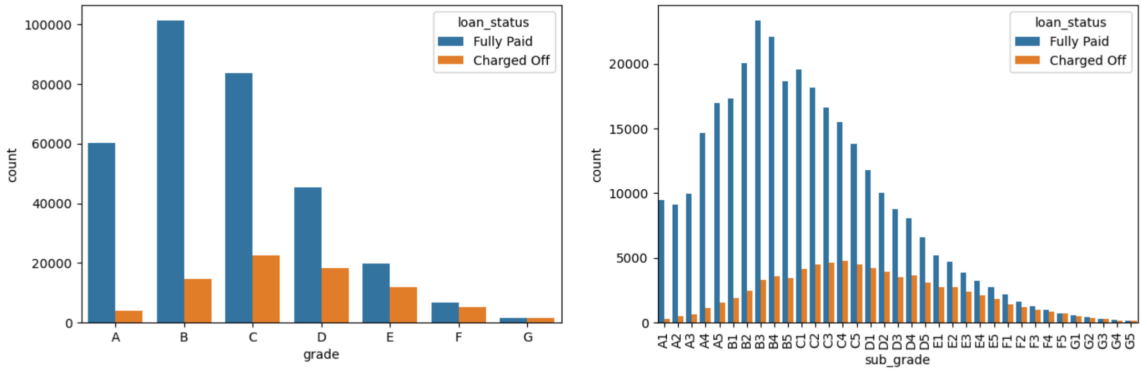
Out[20]:
```
debt consolidation          168108
credit card refinancing      51781
home improvement             17117
other                        12993
consolidation                 5583
major purchase                4998
debt consolidation loan       3513
business                      3017
medical expenses              2820
credit card consolidation     2638
personal loan                 2460
car financing                 2160
credit card payoff            1904
consolidation loan            1887
vacation                      1866
credit card refinance         1832
moving and relocation         1693
consolidate                   1528
personal                      1465
home buying                   1196
Name: title, dtype: int64
```

In [ ]:

Visualisation

In [21]:
```
plt.figure(figsize=(15, 10))

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

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

Out[21]:
```
[Text(0, 0, 'A1'),
 Text(1, 0, 'A2'),
 Text(2, 0, 'A3'),
 Text(3, 0, 'A4'),
 Text(4, 0, 'A5'),
 Text(5, 0, 'B1'),
 Text(6, 0, 'B2'),
 Text(7, 0, 'B3'),
 Text(8, 0, 'B4'),
 Text(9, 0, 'B5'),
 Text(10, 0, 'C1'),
 Text(11, 0, 'C2'),
 Text(12, 0, 'C3'),
 Text(13, 0, 'C4'),
 Text(14, 0, 'C5'),
 Text(15, 0, 'D1'),
 Text(16, 0, 'D2'),
 Text(17, 0, 'D3'),
 Text(18, 0, 'D4'),
 Text(19, 0, 'D5'),
 Text(20, 0, 'E1'),
 Text(21, 0, 'E2'),
 Text(22, 0, 'E3'),
 Text(23, 0, 'E4'),
 Text(24, 0, 'E5'),
 Text(25, 0, 'F1'),
 Text(26, 0, 'F2'),
 Text(27, 0, 'F3'),
 Text(28, 0, 'F4'),
 Text(29, 0, 'F5'),
 Text(30, 0, 'G1'),
 Text(31, 0, 'G2'),
 Text(32, 0, 'G3'),
 Text(33, 0, 'G4'),
 Text(34, 0, 'G5')]
```



The grade of majority of people those who have fully paid the loan is 'B' and have subgrade 'B3'.

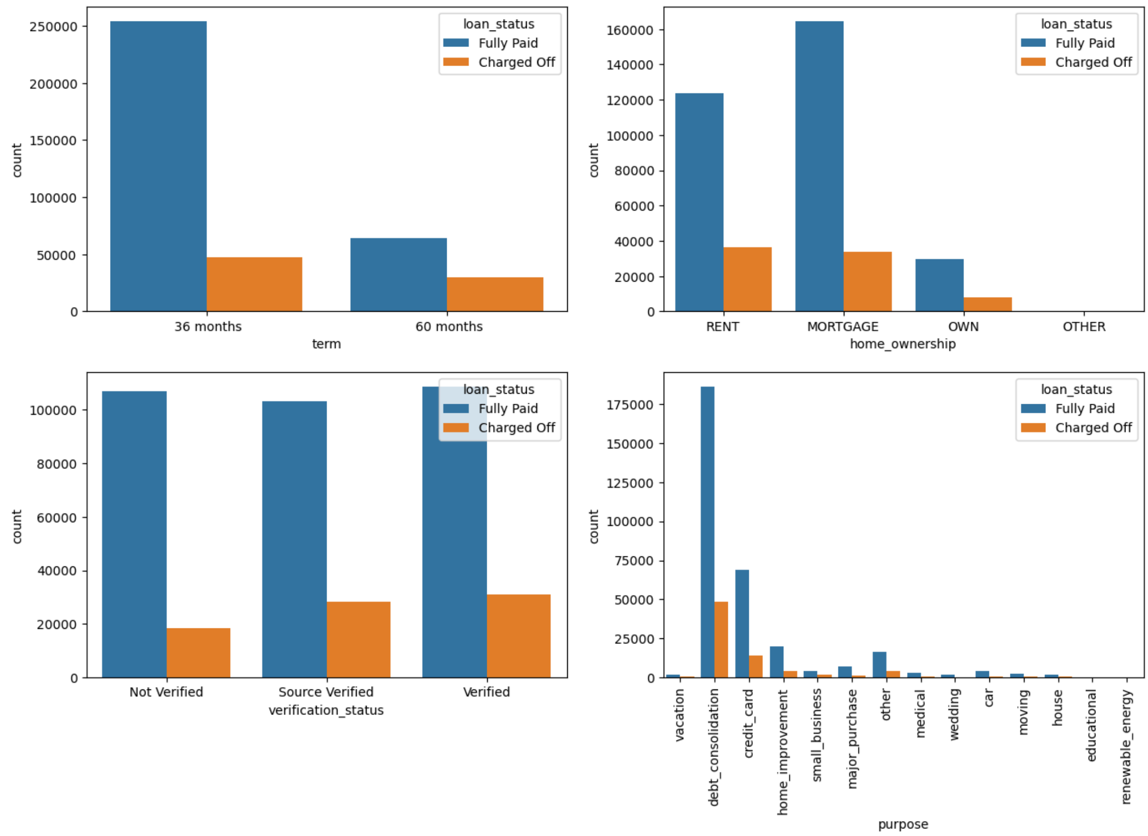
So from that we can infer that people with grade 'B' and subgrade 'B3' are more likely to fully pay the loan.

In [ ]:

```

```

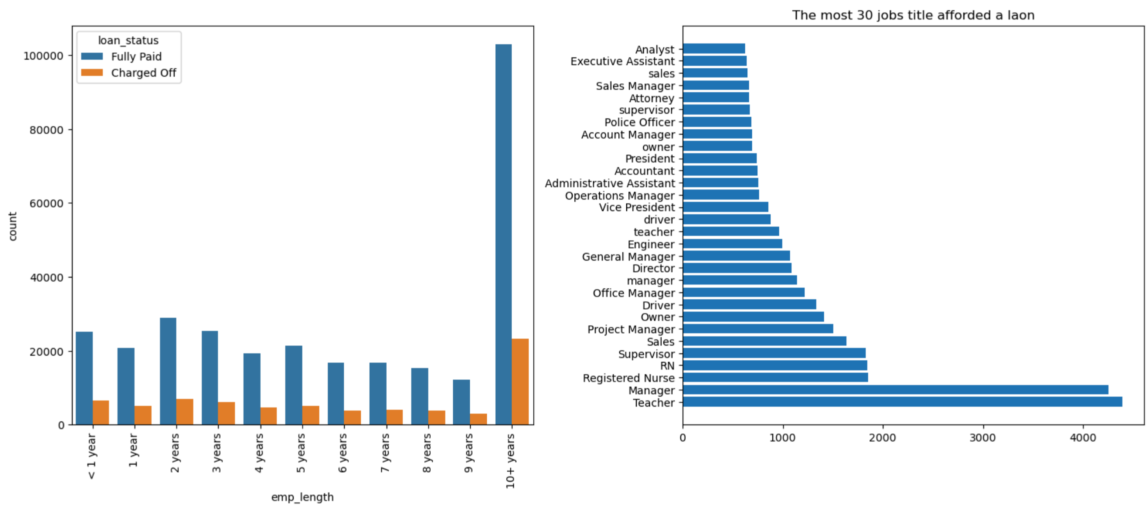
In [22]:
```
plt.figure(figsize=(15,20))

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

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

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

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

Out[22]:
```
[Text(0, 0, 'vacation'),
 Text(1, 0, 'debt_consolidation'),
 Text(2, 0, 'credit_card'),
 Text(3, 0, 'home_improvement'),
 Text(4, 0, 'small_business'),
 Text(5, 0, 'major_purchase'),
 Text(6, 0, 'other'),
 Text(7, 0, 'medical'),
 Text(8, 0, 'wedding'),
 Text(9, 0, 'car'),
 Text(10, 0, 'moving'),
 Text(11, 0, 'house'),
 Text(12, 0, 'educational'),
 Text(13, 0, 'renewable_energy')]
```



In [ ]:

In [23]:
```
plt.figure(figsize=(15,12))

plt.subplot(2,2,1)
order = ['< 1 year', '1 year', '2 years', '3 years', '4 years', '5 years',
          '6 years', '7 years', '8 years', '9 years', '10+ years',]
g=sns.countplot(x='emp_length',data=df,hue='loan_status',order=order)
g.set_xticklabels(g.get_xticklabels(),rotation=90)

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



Manager and Teacher are the most afforded loan on titles

In [ ]:

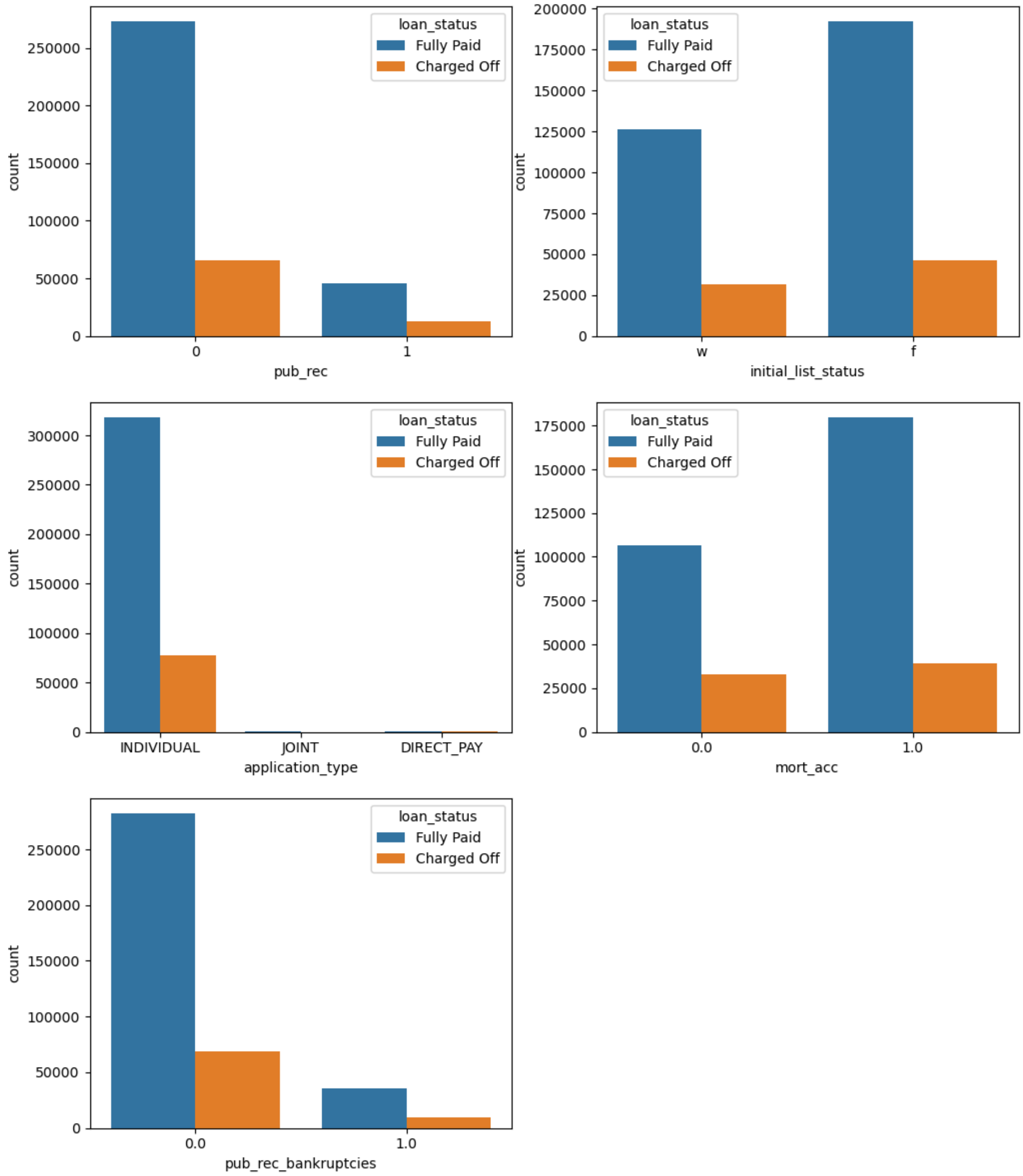
Feature Engineering

In [24]:
```
## Detecting the high outlier columns. (Not deleting these records since som
## have low bankruptcy record.Therefore just flagging anything more than 0
```

In [25]:
```
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 [26]:
```
df['pub_rec']=df.pub_rec.apply(pub_rec)
df['mort_acc']=df.mort_acc.apply(mort_acc)
df['pub_rec_bankruptcies']=df.pub_rec_bankruptcies.apply(pub_rec_bankruptcie
```

In [27]:
```
plt.figure(figsize=(12,30))

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

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

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

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

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

plt.show()
```



In [ ]:

In [28]:
```
# Mapping of target variable

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

In [29]:
```
df.isnull().sum()/len(df)*100
```

Out[29]:
```
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
```

In [ ]:

Mean Target Imputation

In [30]:
```
df.groupby(by='total_acc').mean()
```

Out[30]:

total_acc	loan_amnt	int_rate	annual_inc	loan_status	dti	open_acc	pub_re
2.0	6672.222222	15.801111	64277.777778	0.222222	2.279444	1.611111	0.00000
3.0	6042.966361	15.615566	41270.753884	0.220183	6.502813	2.611621	0.03363
4.0	7587.399031	15.069491	42426.565969	0.214055	8.411963	3.324717	0.03311
5.0	7845.734714	14.917564	44394.098003	0.203156	10.118328	3.921598	0.05572
6.0	8529.019843	14.651752	48470.001156	0.215874	11.222542	4.511119	0.07663
...	...	...	...	...	...	...	.
124.0	23200.000000	17.860000	66000.000000	1.000000	14.040000	43.000000	0.00000
129.0	25000.000000	7.890000	200000.000000	0.000000	8.900000	48.000000	0.00000
135.0	24000.000000	15.410000	82000.000000	0.000000	33.850000	57.000000	0.00000
150.0	35000.000000	8.670000	189000.000000	0.000000	6.630000	40.000000	0.00000
151.0	35000.000000	13.990000	160000.000000	1.000000	12.650000	26.000000	0.00000

118 rows × 11 columns

In [ ]:

In [31]:
```
# saving mean of mort_acc according to total_acc_avg

total_acc_avg=df.groupby(by='total_acc').mean().mort_acc
```

In [32]:
```
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 [33]:
```
df['mort_acc']=df.apply(lambda x: fill_mort_acc(x['total_acc'],x['mort_acc']
```

In [34]:
```
df.isnull().sum()/len(df)*100
```

Out[34]:
```
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 [35]:
```
# Current no. of rows

df.shape
```

Out[35]:
```
(396030, 26)
```

In [36]:
```
# Dropping rows with null values

df.dropna(inplace=True)
```

In [37]:
```
# Remaining no. of rows

df.shape
```

Out[37]:
```
(370622, 26)
```

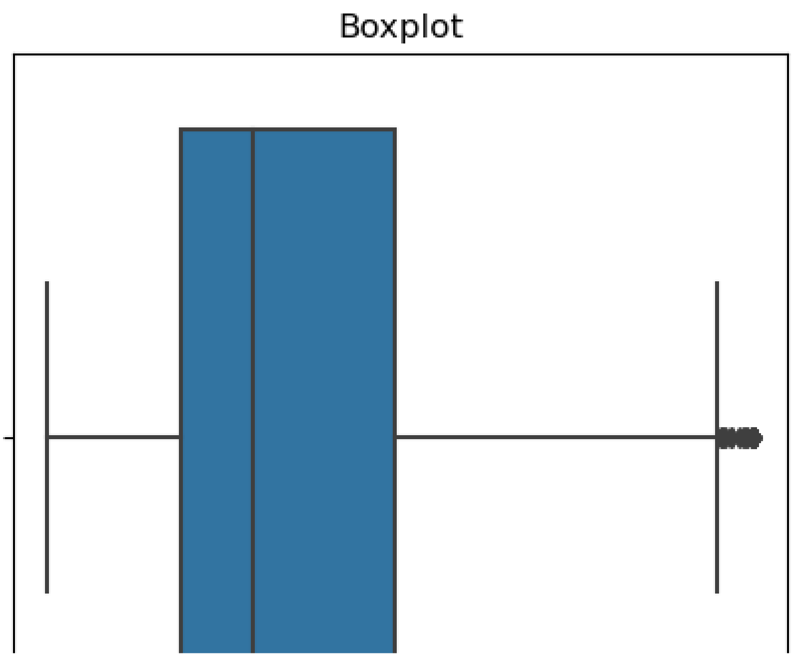
In [ ]:

Outlier Detection & Treatment

In [38]:
```
numerical_data=df.select_dtypes(include='number')
num_cols=numerical_data.columns
len(num_cols)
```

Out[38]:
```
12
```

In [39]:
```
def box_plot(col):
    plt.figure(figsize=(5,5))
    sns.boxplot(x=df[col])
    plt.title('Boxplot')
    plt.show()

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



In [ ]:

In [40]:
```
for col in num_cols:
    mean=df[col].mean()
    std=df[col].std()

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

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

df.shape
```

Out[40]: (354519, 26)

In [ ]:

Data Preprocessing

In [41]:
```
# Term

df.term.unique()
```

Out[41]: array([' 36 months', ' 60 months'], dtype=object)

In [42]:
```
term_values={' 36 months': 36, ' 60 months':60}
df['term'] = df.term.map(term_values)
```

In [43]:
```
# Initial List Status

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

Out[43]:
```
array(['w', 'f'], dtype=object)
```

In [44]:
```
list_status = {'w': 0, 'f': 1}
df['initial_list_status'] = df.initial_list_status.map(list_status)
```

In [45]:
```
# Fetching ZIP from address and then dropping the remaining details -

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

In [46]:
```
df['zip_code'].value_counts(normalize=True)*100
```

Out[46]:
```
70466    14.382022
30723    14.277373
22690    14.268347
48052    14.127028
00813    11.610097
29597    11.537322
05113    11.516731
93700     2.774746
11650     2.772771
86630     2.733563
Name: zip_code, dtype: float64
```

In [ ]:

In [47]:
```
# Dropping few variables() which we can let go for now )

df.drop(columns=['issue_d', 'emp_title', 'title', 'sub_grade','address', 'ea
```

In [ ]:

One-hot Encoding

In [48]:
```
dummies=['purpose', 'zip_code', 'grade', 'verification_status', 'applicatior
df=pd.get_dummies(df,columns=dummies,drop_first=True)
```

In [49]:
```
pd.set_option('display.max_columns',None)
pd.set_option('display.max_rows',None)

df.head()
```

Out[49]:

	loan_amnt	term	int_rate	annual_inc	loan_status	dti	open_acc	pub_rec	revol_bal	r
0	10000.0	36	11.44	117000.0	0	26.24	16.0	0	36369.0	
1	8000.0	36	11.99	65000.0	0	22.05	17.0	0	20131.0	
2	15600.0	36	10.49	43057.0	0	12.79	13.0	0	11987.0	
3	7200.0	36	6.49	54000.0	0	2.60	6.0	0	5472.0	
4	24375.0	60	17.27	55000.0	1	33.95	13.0	0	24584.0	

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

Out[50]:
```
(354519, 49)
```

In [ ]:

Data Preparation for Modelling

In [51]:
```
## Splitting the data in x & y variable

X=df.drop('loan_status',axis=1)
y=df['loan_status']
```

In [52]:
```
X_train, X_test, y_train, y_test =train_test_split(X,y,test_size=0.30,strati
```

In [53]:
```
X_train.shape,X_test.shape
```

Out[53]:
```
((248163, 48), (106356, 48))
```

In [54]:
```
y_train.shape,y_test.shape
```

Out[54]:
```
((248163,), (106356,))
```

In [ ]:

In [55]:
```
## Scaling the data(MinMaxScaler)
```

In [56]:
```
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [ ]:

Logistic Regression

In [57]:
```
## Fitting the model

logreg=LogisticRegression(max_iter=1000)
logreg.fit(X_train,y_train)
```

Out[57]:
```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

In [58]:
```
y_pred = logreg.predict(X_test)
print('Accuracy of Logistic Regression Classifier on test set: {:.3f}'.forma
```

```
Accuracy of Logistic Regression Classifier on test set: 0.890
```

In [ ]:

Confusion Matrix

In [59]:
```
confusion_matrix=confusion_matrix(y_test,y_pred)
print(confusion_matrix)
```

```
[[85367   521]
 [11131  9337]]
```

In [ ]:

Classification Report

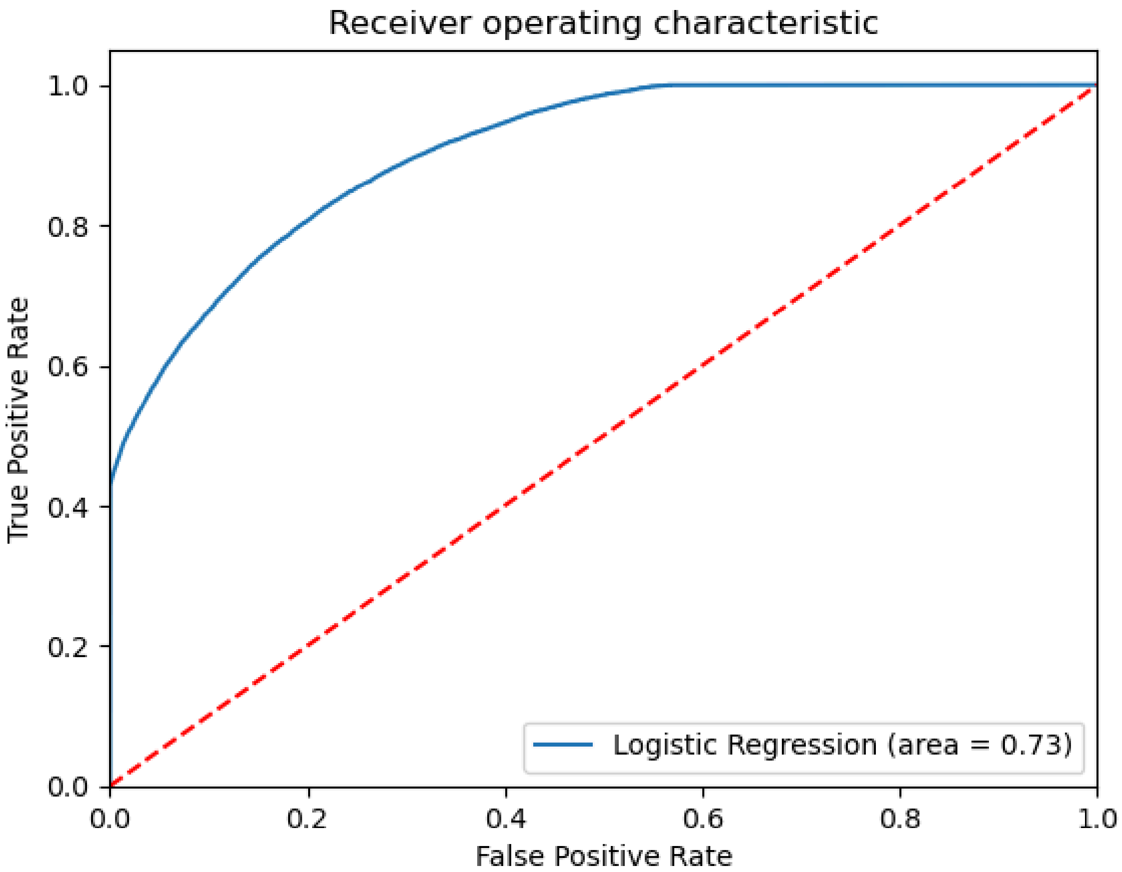
In [60]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.99      0.94     85888
           1       0.95      0.46      0.62     20468

    accuracy                           0.89    106356
   macro avg       0.92      0.73      0.78    106356
weighted avg       0.90      0.89      0.87    106356
```

In [ ]:

ROC(Receiver operating characteristic) Curve and AUC (Area under the ROC) Curve

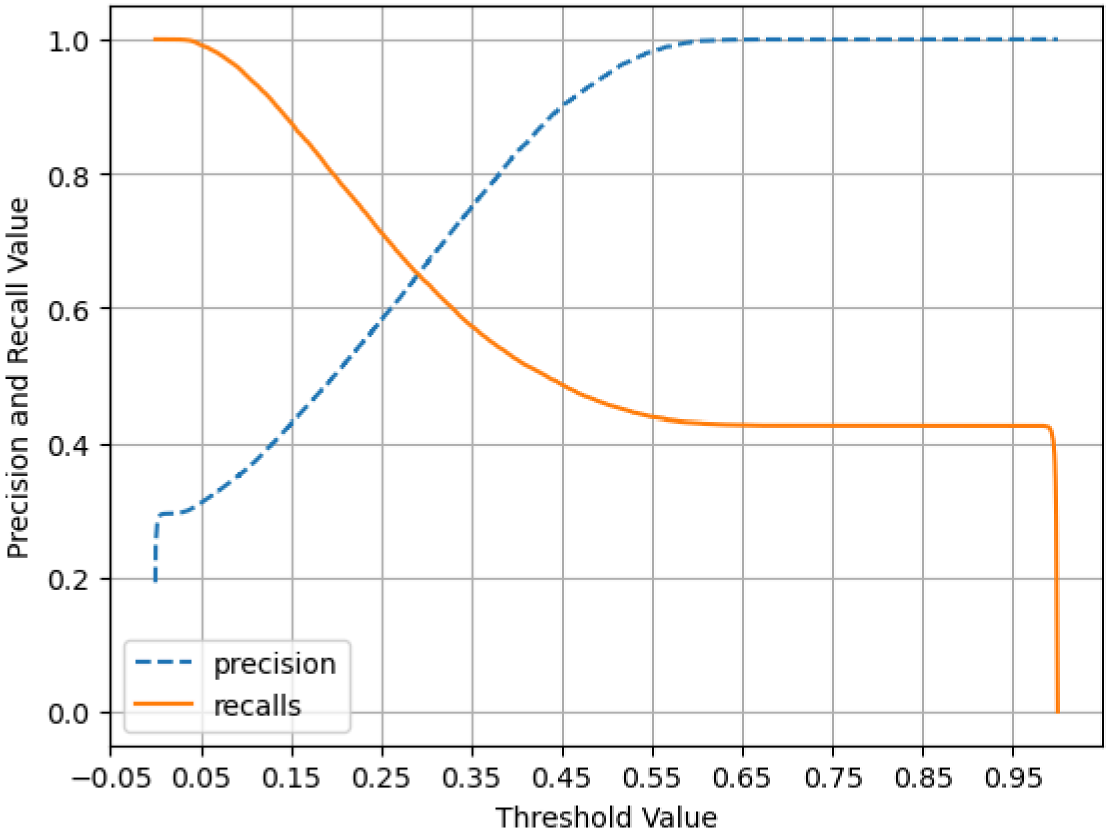
In [61]:
```
logit_roc_auc=roc_auc_score(y_test,logreg.predict(X_test))
fpr,tpr,thresholds=roc_curve(y_test,logreg.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr,tpr,label='Logistic Regression (area = %0.2f)' % logit_roc_auc
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```



In [ ]:

Precission_recall_curve_plot

In [62]:
```
def precission_recall_curve_plot(y_test,pred_proba_c1):
    precisions, recalls, thresholds = precision_recall_curve(y_test,pred_pro

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

precission_recall_curve_plot(y_test,logreg.predict_proba(X_test)[:,1])
```



In [ ]:

Multicollinearity check using Variance Inflation Factor (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
    vif['VIF']=round(vif['VIF'],2)
    vif=vif.sort_values(by='VIF',ascending=False)
    return vif

calc_vif(X)[:5]
```

Out[63]:

	Feature	VIF
43	application_type_INDIVIDUAL	156.97
2	int_rate	122.82
14	purpose_debt_consolidation	51.00
1	term	27.30
13	purpose_credit_card	18.48

In [64]:
```
## Dropping 'application_type_INDIVIDUAL' column

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

Out[64]:

	Feature	VIF
2	int_rate	103.43
14	purpose_debt_consolidation	27.49
1	term	24.31
5	open_acc	13.75
9	total_acc	12.69

In [65]:
```
## Dropping 'int_rate' column

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

Out[65]:

	Feature	VIF
1	term	23.35
13	purpose_debt_consolidation	22.35
4	open_acc	13.64
8	total_acc	12.69
7	revol_util	9.06

In [66]:
```
## Dropping 'term'column

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

Out[66]:

	Feature	VIF
12	purpose_debt_consolidation	18.37
3	open_acc	13.64
7	total_acc	12.65
6	revol_util	9.04
1	annual_inc	8.03

In [67]:
```
## Dropping 'purpose_debt_consolidation' column

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

Out[67]:

	Feature	VIF
3	open_acc	13.09
7	total_acc	12.64
6	revol_util	8.31
1	annual_inc	7.70
2	dti	7.58

In [68]:
```
## Dropping 'open_acc' column

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

Out[68]:

	Feature	VIF
6	total_acc	8.23
5	revol_util	8.00
1	annual_inc	7.60
2	dti	7.02
0	loan_amnt	6.72

In [ ]:

Cross Validation accuracy - (Validation using KFold)

In [69]:
```
X=scaler.fit_transform(X)

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

```
Cross Validation accuracy : 0.891
```

In [70]:
```
## Cross Validation accuracy and testing accuracy is almost same which infer
```

In [ ]:

Oversampling using SMOTE

In [71]:
```
sm=SMOTE(random_state=42)
X_train_res,y_train_res=sm.fit_resample(X_train,y_train.ravel())
```

In [72]:
```
print('After OverSampling, the shape of train_X: {}'.format(X_train_res.shap
print('After OverSampling, the shape of train_y: {} \n'.format(y_train_res.s

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

```
After OverSampling, the shape of train_X: (400810, 48)
After OverSampling, the shape of train_y: (400810,)

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

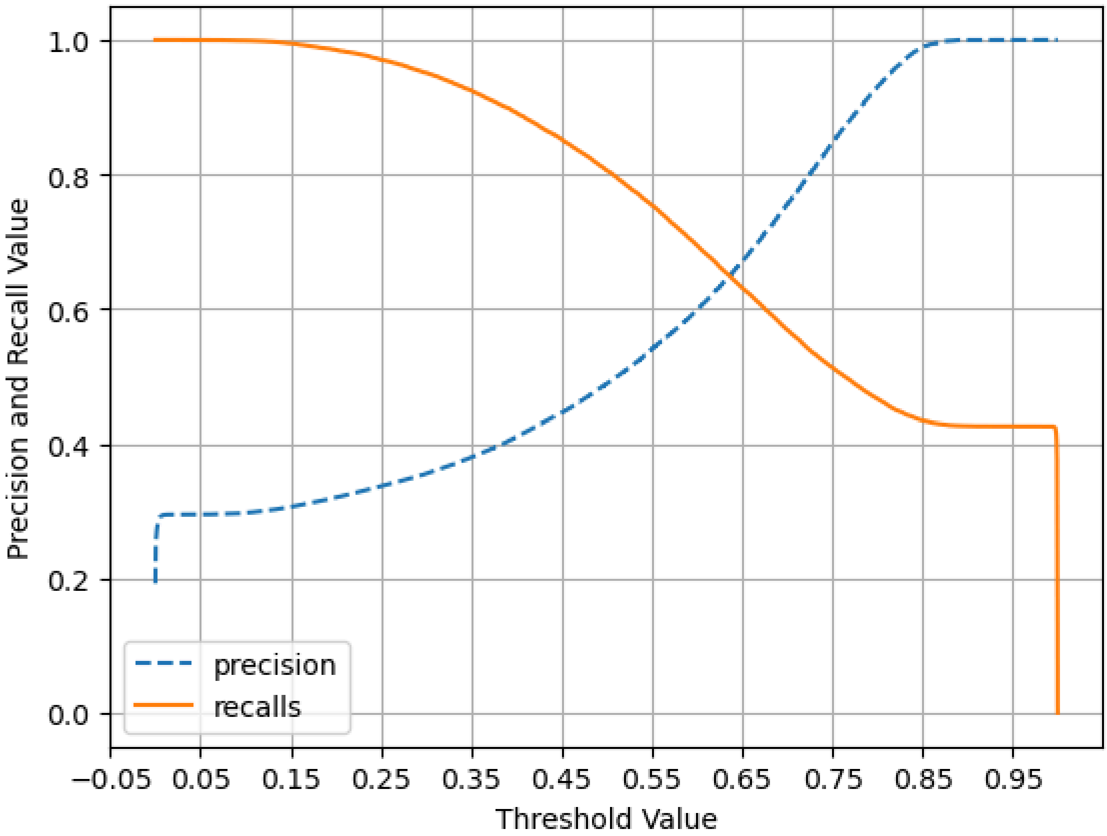
In [73]:
```
lr1 = LogisticRegression(max_iter=1000)
lr1.fit(X_train_res, y_train_res)
predictions = lr1.predict(X_test)

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

```
              precision    recall  f1-score   support

           0       0.95      0.80      0.87     85888
           1       0.49      0.81      0.61     20468

    accuracy                           0.80    106356
   macro avg       0.72      0.80      0.74    106356
weighted avg       0.86      0.80      0.82    106356
```

In [74]:
```
## After making the dataset balanced, the precision and recall score are sam
## There is still room for improvement.
```

In [ ]:

In [75]:
```
## Precision_recall_curve_plot

def precision_recall_curve_plot(y_test, pred_proba_c1):
    precisions, recalls, thresholds = precision_recall_curve(y_test, pred_pr

    threshold_boundary = thresholds.shape[0]
    # plot precision
    plt.plot(thresholds, precisions[0:threshold_boundary], linestyle='--', l
    # 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 [76]:
```
## After balancing the dataset, there is significant change observed in the
## Precision score is .95 and .49 for full paid and charged off respectively
```

In [ ]:

Tradeoff Questions

```
1. How can we make sure that our model can detect real defaulters and there
are less false positives?
```

```
   (This is important as we can lose out on an opportunity to finance more
individuals and earn interest on it.)

Answer - Since data is imbalanced,we can balance the data and can try to
avoid false positives.
         For evaluation metrics, we should be focusing on the macro average
f1-score because we
         don't want to make false positive prediction and at the same we
want to detect the defualters.

2. Since NPA (non-performing asset) is a real problem in this industry,
it’s important we play safe and shouldn’t disburse
   loans to anyone

Answer - Below are the most features and their importance while making the
prediction.
         So these variables can help the managers to identify which are
customers who are more likely to pay the
         loan amount fully.
```

In [85]:
```
coefs = lr1.coef_.tolist()[0]
feature_coef_df = pd.DataFrame({'Variable': range(len(coefs)), 'Coeficient':
feature_coef_df.sort_values(by=['Coeficient'], ascending=False)
```

Out[85]:

	Variable	Coeficient
27	27	13.969023
34	34	13.965646
33	33	13.961654
31	31	6.208743
32	32	6.166388
28	28	6.161188
30	30	6.148002
40	40	1.287924
39	39	1.248181
38	38	1.239945
43	43	1.230186
4	4	1.107999
37	37	1.096884
36	36	0.882708
5	5	0.866036
23	23	0.754095
0	0	0.679283
8	8	0.639293
2	2	0.569135
35	35	0.516077
1	1	0.450707
15	15	0.422355
16	16	0.308721
21	21	0.292368
14	14	0.269560
19	19	0.256942
41	41	0.198143
47	47	0.183624
13	13	0.176496
18	18	0.173316
20	20	0.086791
12	12	0.047302
42	42	0.035817
10	10	0.008257
46	46	-0.009349
6	6	-0.031071
22	22	-0.033563
11	11	-0.048097

	Variable	Coeficient
24	24	-0.064988
17	17	-0.257004
44	44	-0.326725
45	45	-0.401579
7	7	-0.698505
25	25	-0.699964
9	9	-0.777163
3	3	-1.729116
29	29	-2.929288
26	26	-2.931778

In [ ]:

Actional Insights and Recommendations

In [ ]:

```
1. 80% of the customers have paid the loan fully.
2. 20% of the customers are the defaulters.
3. The organization can train model to make prediction for whether a person
   he will be a defaulter.
4. Model achieves the 94% f1-score for the negative class (Fully Paid).
5. Model achieves the 62% f1-score for the positive class (Charged off).
6. Cross Validation accuracy and testing accuracy is almost same which infer
   We can trust this model for unseen data.
7. By collecting more data, using a more complex model, or tuning the hyper
   improve the model's performance.
8. ROC-AUC curve area of 0.73, the model is correctly classifying about 73%
   This is a good performance, but there is still room for improvement.
9. The precision-recall curve allows us to see how the precision and recall
    A higher threshold will result in higher precision, but lower recall, ar
    is the one that best meets the needs of the specific application.
10. After balancing the dataset, there is significant change observed in the
11. Accuracy of Logistic Regression Classifier on test set: 0.891 which is c
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: