

Importing necessary libraries -

```
In [1]: rt pandas as pd
rt numpy as np
rt seaborn as sns
rt scipy import stats
rt matplotlib.pyplot as plt
sklearn.linear_model import LogisticRegression
sklearn import metrics
sklearn.metrics import confusion_matrix
sklearn.metrics import classification_report
sklearn.metrics import roc_curve
sklearn.metrics import precision_recall_curve
sklearn.model_selection import train_test_split, KFold, cross_val_score
sklearn.preprocessing import MinMaxScaler

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

Here is the information on this particular data set:

0. 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.
1. term : The number of payments on the loan. Values are in months and can be either 36 or 60.
2. int_rate : Interest Rate on the loan
3. installment : The monthly payment owed by the borrower if the loan originates.
4. grade LC : assigned loan grade
5. sub_grade LC : assigned loan subgrade
6. emp_title : The job title supplied by the Borrower when applying for the loan.
7. 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.
8. 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
9. annual_inc : The self-reported annual income provided by the borrower during registration.
10. verification_status : Indicates if income was verified by LC, not verified, or if the income source was verified
11. issue_d : The month which the loan was funded
12. loan_status : Current status of the loan
13. purpose : A category provided by the borrower for the loan request.
14. title : The loan title provided by the borrower
15. zip_code : The first 3 numbers of the zip code provided by the borrower in the loan application.
16. addr_state : The state provided by the borrower in the loan application
17. 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.
18. earliest_cr_line : The month the borrower's earliest reported credit line was opened
19. open_acc : The number of open credit lines in the borrower's credit file.

- 20. pub_rec : Number of derogatory public records
- 21. revol_bal : Total credit revolving balance
- 22. revol_util : Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.
- 23. total_acc : The total number of credit lines currently in the borrower's credit file
- 24. initial_list_status : The initial listing status of the loan. Possible values are – W, F
- 25. application_type : Indicates whether the loan is an individual application or a joint application with two co-borrowers
- 26. mort_acc : Number of mortgage accounts.
- 27. pub_rec_bankruptcies : Number of public record bankruptcies

Reading the data file -

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

```
In [4]: data.head()
```

Out [4]:

	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_income
	36 months	11.44	329.48	B	B4	Marketing	10+ years	RENT	117500
	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	65000
	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RENT	45000
	36 months	6.49	220.65	A	A2	Client Advocate	6 years	RENT	54000
	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55000

```
In [6]: # Shape of the dataset -
print("No. of rows: ", data.shape[0])
print("No. of columns: ", data.shape[1])
```

No. of rows: 396030
No. of columns: 27

```
In [8]: # Checking the distribution of outcome labels -
data.loan_status.value_counts(normalize=True)*100
```

Out [8]: Fully Paid 80.387092
Charged Off 19.612908
Name: loan_status, dtype: float64

```
In [ ]: # Statistical summary of the dataset -
data.describe(include='all')
```

Out [14]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_title
count	231746.000000	231746	231746.000000	231746.000000	231746	231746	218320	
unique	NaN	2	NaN	NaN	7	35	109777	
top	NaN	36 months	NaN	NaN	B	B3	Teacher	
freq	NaN	176621	NaN	NaN	67866	15546	2557	
mean	14107.757955	NaN	13.640928	431.586377	NaN	NaN	NaN	
std	8353.939311	NaN	4.466482	250.592582	NaN	NaN	NaN	
min	500.000000	NaN	5.320000	16.250000	NaN	NaN	NaN	
25%	8000.000000	NaN	10.490000	250.340000	NaN	NaN	NaN	
50%	12000.000000	NaN	13.330000	375.430000	NaN	NaN	NaN	
75%	20000.000000	NaN	16.490000	567.040000	NaN	NaN	NaN	
max	40000.000000	NaN	30.990000	1533.810000	NaN	NaN	NaN	

11 rows × 27 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 231746 entries, 0 to 231745
Data columns (total 27 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   loan_amnt                             231746 non-null float64
1   term                                  231746 non-null object
2   int_rate                              231746 non-null float64
3   installment                           231746 non-null float64
4   grade                                 231746 non-null object
5   sub_grade                             231746 non-null object
6   emp_title                             218320 non-null object
7   emp_length                            221005 non-null object
8   home_ownership                        231746 non-null object
9   annual_inc                            231746 non-null float64
10  verification_status                   231746 non-null object
11  issue_d                               231746 non-null object
12  loan_status                           231746 non-null object
13  purpose                               231746 non-null object
14  title                                 230723 non-null object
15  dti                                   231746 non-null float64
16  earliest_cr_line                      231746 non-null object
17  open_acc                              231746 non-null float64
18  pub_rec                               231746 non-null float64
19  revol_bal                             231746 non-null float64
20  revol_util                            231576 non-null float64
21  total_acc                             231746 non-null float64
22  initial_list_status                   231746 non-null object
23  application_type                      231746 non-null object
24  mort_acc                             209558 non-null float64
25  pub_rec_bankruptcies                  231426 non-null float64
26  address                               231745 non-null object
dtypes: float64(12), object(15)
memory usage: 47.7+ MB
```

Correlation Heatmap -

A correlation heatmap is a heatmap that shows a 2D correlation matrix between two discrete dimensions, using colored cells to represent data from usually a monochromatic scale. The values of the first dimension appear as the rows of the table while of the second dimension as a column. The color of the cell is proportional to the number of measurements that match the dimensional value. This makes correlation heatmaps ideal for data analysis since it makes patterns easily readable and highlights the differences and variation in the same data. A correlation heatmap, like a regular heatmap, is assisted by a colorbar making data easily readable and comprehensible.

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



```
In [ ]: you can check this after converting your loan status into 0 and 1x
```

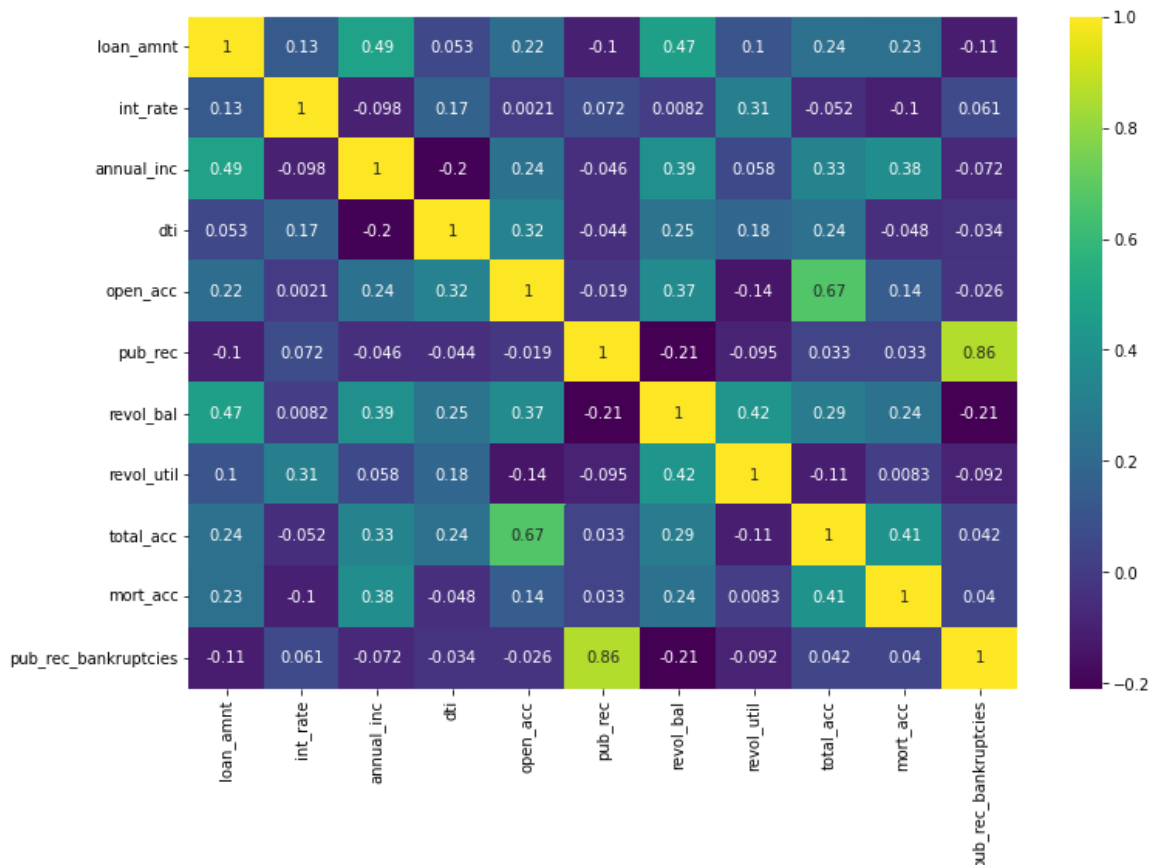
We noticed almost perfect correlation between "loan_amnt" the "installment" feature.

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

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



Data Exploration -

1. The no of people those who have fully paid are 318357 and that of Charged Off are 77673.

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

```
Out[19]:
```

	count	mean	std	min	25%	50%	75%	max
loan_status								
Charged Off	45368.0	15135.654536	8484.848789	1000.0	8700.0	14000.0	20000.0	40000.0
Fully Paid	186378.0	13857.548101	8302.548155	500.0	7500.0	12000.0	19200.0	40000.0

2. The majority of people have home ownership as Mortgage and Rent.

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

```
Out[20]: MORTGAGE      116104
          RENT        93551
          OWN         22010
          OTHER        59
          NONE         19
          ANY          3
          Name: home_ownership, dtype: int64
```

3. Combining the minority classes as 'OTHER'.

```
In [ ]: 'ANY') | (data.home_ownership == 'NONE'), 'home_ownership'] = 'OTHER'
        ()
```

```
Out[21]: MORTGAGE      116104
          RENT        93551
          OWN         22010
          OTHER        81
          Name: home_ownership, dtype: int64
```

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

```
Out[22]: MORTGAGE      116104
          RENT        93551
          OWN         22010
          OTHER        81
          Name: home_ownership, dtype: int64
```

```
In [ ]: # Checking the distribution of 'Other' -
        data.loc[data['home_ownership']=='OTHER', 'loan_status'].value_counts()
```

```
Out[23]: Fully Paid      64
          Charged Off    17
          Name: loan_status, dtype: int64
```

```
In [ ]: data['issue_d'] = pd.to_datetime(data['issue_d'])
        data['earliest_cr_line'] = pd.to_datetime(data['earliest_cr_line'])
```

5. Saw some issues in title (Looks like it was filled manually and needs some fixing).

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

```
Out[25]: Debt consolidation      89183
Credit card refinancing      30086
Home improvement              8932
Other                        7597
Debt Consolidation           6854
Major purchase               2887
Consolidation                2305
debt consolidation           2055
Business                    1701
Debt Consolidation Loan      1673
Medical expenses             1605
Car financing                1242
Credit Card Consolidation    1049
Vacation                     1025
Moving and relocation         986
consolidation                 971
Personal Loan                 913
Home Improvement              746
Consolidation Loan           739
Home buying                   674
Name: title, dtype: int64
```

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

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

```
Out[27]: debt consolidation      98370
credit card refinancing      30256
home improvement             10011
other                        7637
consolidation                3344
major purchase               3024
debt consolidation loan      2059
business                     1749
medical expenses             1652
credit card consolidation    1567
Name: title, dtype: int64
```

Visualization -

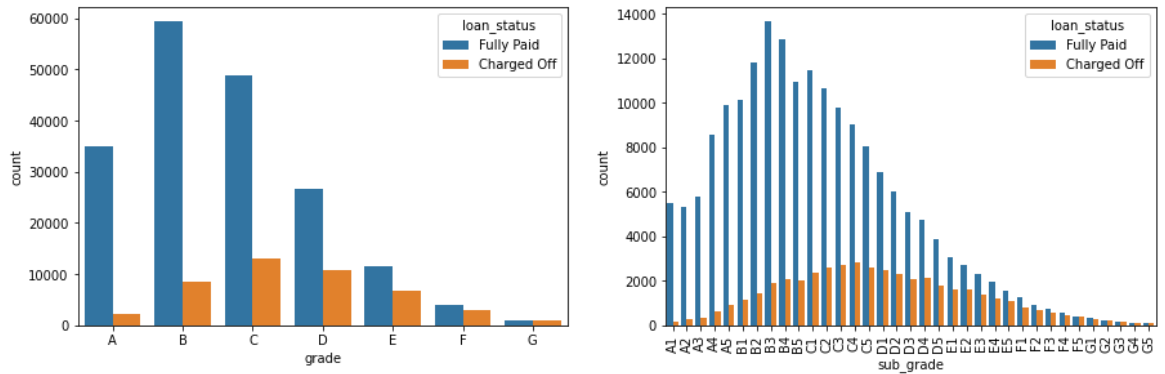
The grade of majority of people those who have fully paid the loan is 'B' and have subgrade 'B3'.

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


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

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

subplot(2, 2, 2)
    rade = sorted(data.sub_grade.unique().tolist())
ns.countplot(x='sub_grade', data=data, hue='loan_status', order=sub_grade)
_xticklabels(g.get_xticklabels(), rotation=90);
```



```
In [ ]: grade
```

```
Out[30]: ['A', 'B', 'C', 'D', 'E', 'F', 'G']
```

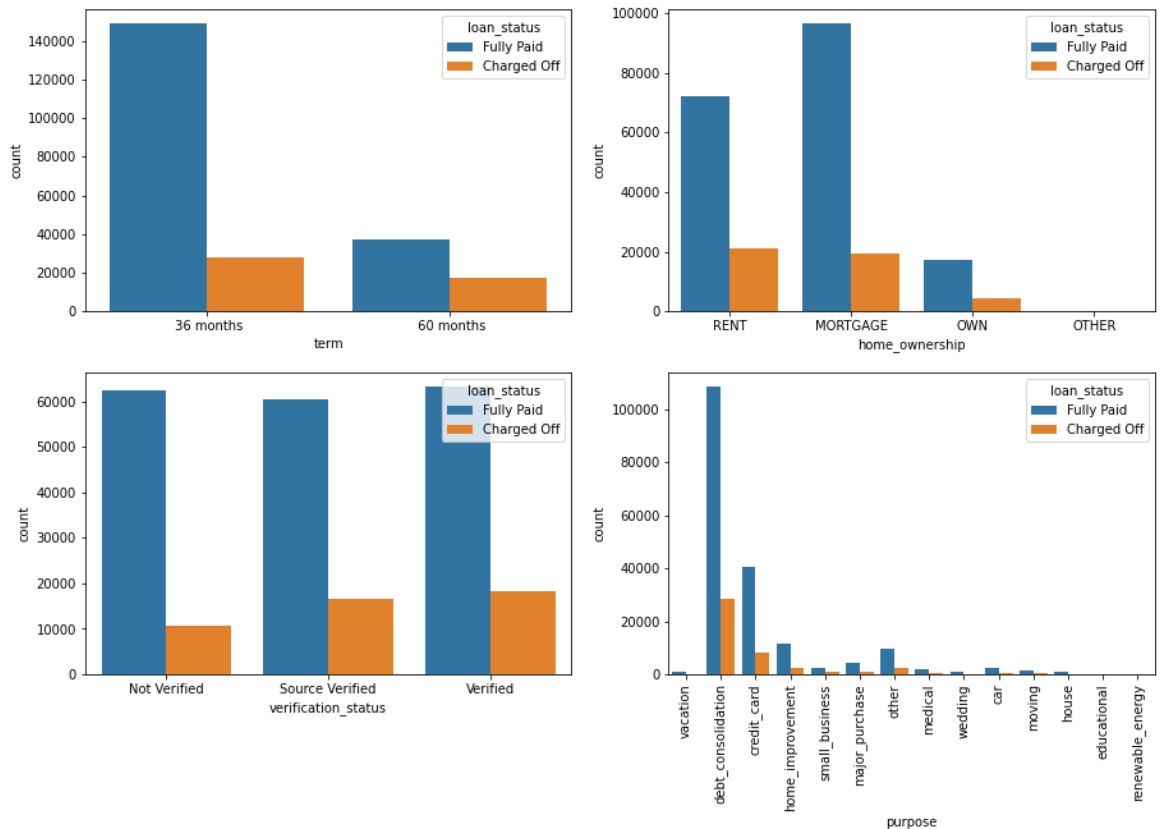
```
In [ ]: plt.figure(figsize=(15, 20))

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);
```



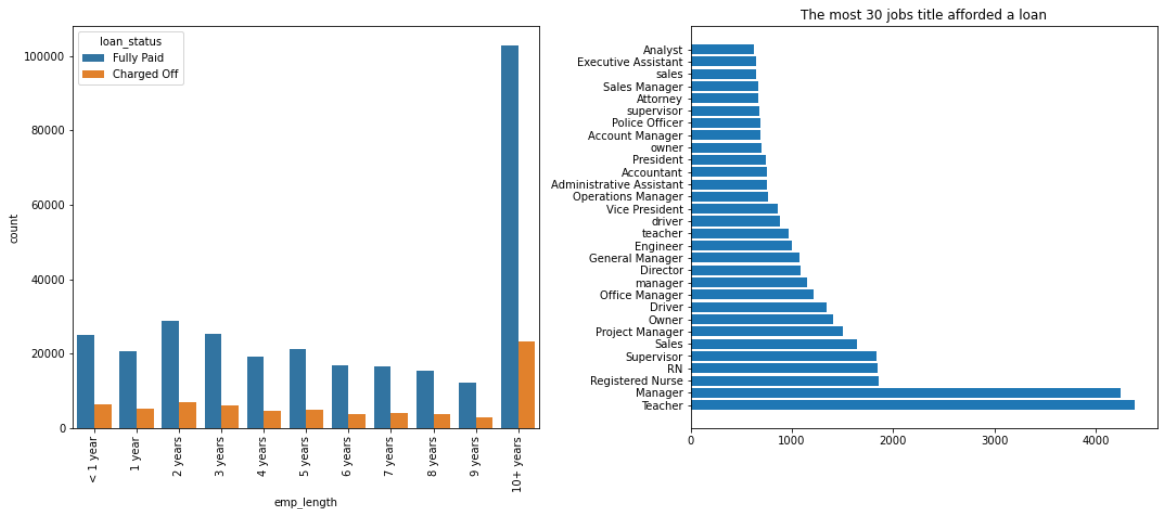
In []:

Manager and Teacher are the most afforded loan job titles.

```
In [ ]: 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=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].values)
plt.title("The most 30 jobs title afforded a loan")
plt.tight_layout()
```



Feature Engineering -

```
In [ ]: 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 [ ]: 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)
```

```

In [ ]: 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')

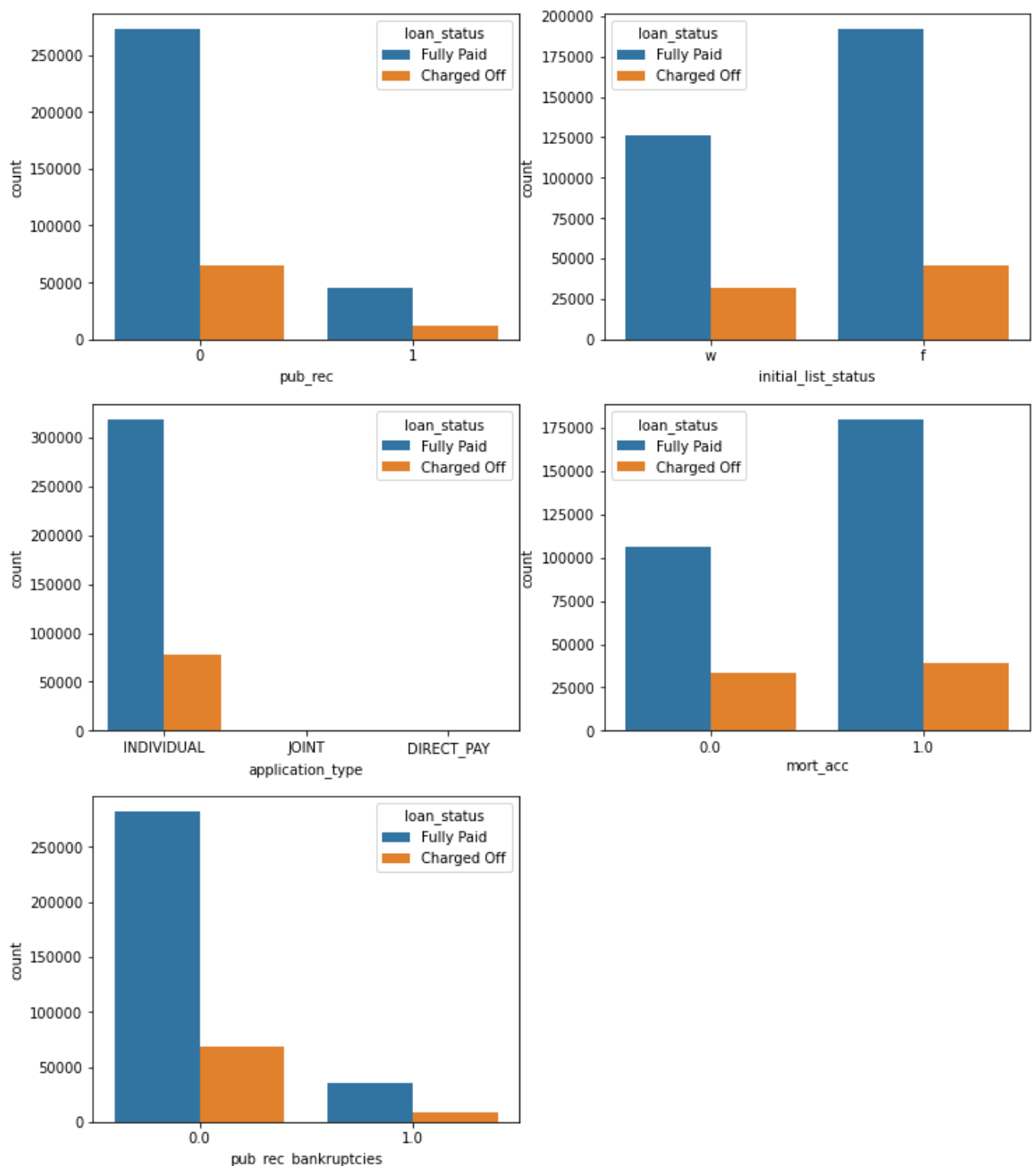
plt.subplot(6, 2, 3)
sns.countplot(x='application_type', data=data, hue='loan_status')

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 [ ]: ping of target variable -
        'loan_status' = data.loan_status.map({'Fully Paid':0, 'Charged Off':1
```

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

```
Out[25]: 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
```

Very Important: Mean Imputation

```
In [12]: data.groupby(by='total_acc')['mort_acc'].median()
```

```
Out[12]: 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    3.0
150.0    2.0
151.0    0.0
Name: mort_acc, Length: 118, dtype: float64
```

```
In [13]: total_acc_avg = data.groupby(by='total_acc').median().mort_acc
        # Saving mean of mort_acc according to total_acc_avg (you can pick any
```

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

```
In [ ]: 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   0.000000  
pub_rec_bankruptcies 0.135091  
address    0.000000  
dtype: float64
```

```
In [ ]: # Current no. of rows -  
data.shape
```

```
Out[31]: (396030, 26)
```

```
In [ ]: # Dropping rows with null values - # do your own research  
data.dropna(inplace=True)
```

```
In [ ]: # Remaining no. of rows -  
data.shape
```

```
Out[33]: (370622, 26)
```

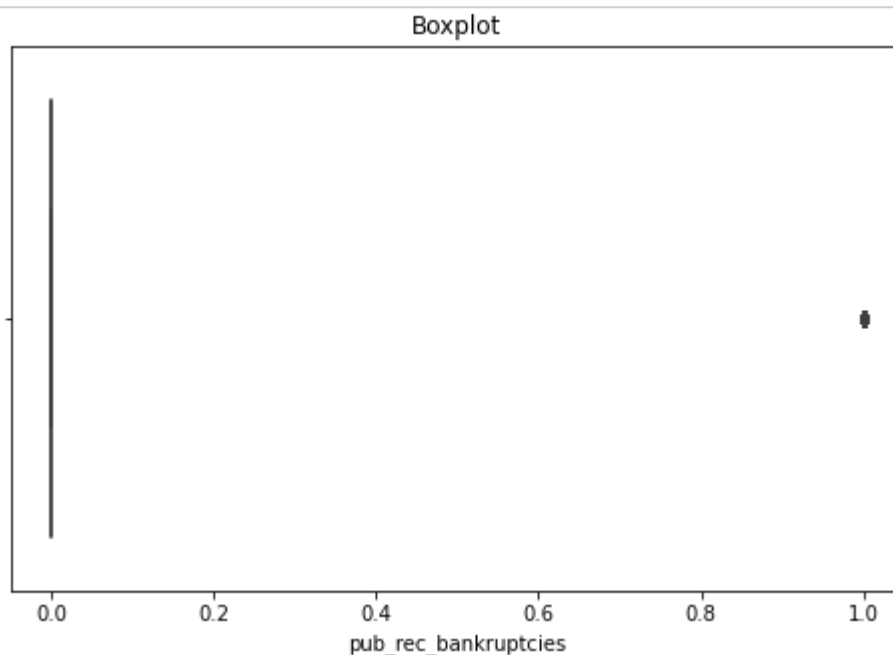
Outlier Detection & Treatment -

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

Out[34]: 12

```
In [ ]: 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)
```



```
In [ ]: 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[36]: (354519, 26)

Data Preprocessing -

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

Out[37]: array([' 36 months', ' 60 months'], dtype=object)


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

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

```
Out[39]: array(['w', 'f'], dtype=object)
```

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

```
In [ ]: # Let's fetch ZIP from address and then drop the remaining details -
data['zip_code'] = data.address.apply(lambda x: x[-5:])
```

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

```
Out[42]: 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 [ ]: # 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 - frequency mapping
#a - number of rows
#b
```

One-hot Encoding -

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

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

data.head()
```

```
Out[45]:
```

	loan_amnt	term	int_rate	annual_inc	loan_status	dti	open_acc	pub_rec	revol_bal	rev
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 [ ]: data.shape
```

```
Out[46]: (354519, 49)
```

Data Preparation for Modeling -

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

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

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

```
(248163, 48)
```

```
(106356, 48)
```

MinMaxScaler -

For each value in a feature, MinMaxScaler subtracts the minimum value in the feature and then divides by the range. The range is the difference between the original maximum and original minimum.

MinMaxScaler preserves the shape of the original distribution. It doesn't meaningfully change the information embedded in the original data.

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

Logistic Regression

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

```
Out[51]: LogisticRegression(max_iter=1000)
```

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

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

Confusion Matrix -

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

[[85365  523]
 [11131  9337]]
```

Classification Report -

```
In [ ]: 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

ROC Curve -

An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

- True Positive Rate
- False Positive Rate

True Positive Rate (TPR) is a synonym for recall and is therefore defined as follows:

- $TPR = (TP) / (TP + FN)$

False Positive Rate (FPR) is defined as follows:

- $FPR = (FP) / (FP + TN)$

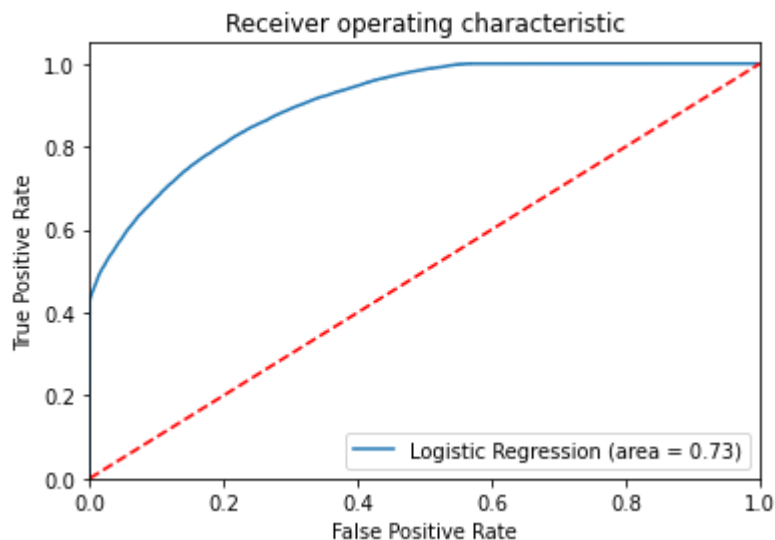
An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure shows a typical ROC curve.

AUC (Area under the ROC Curve) -

AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1).

AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example. For example, given the following examples, which are arranged from left to right in ascending order of logistic regression predictions:

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



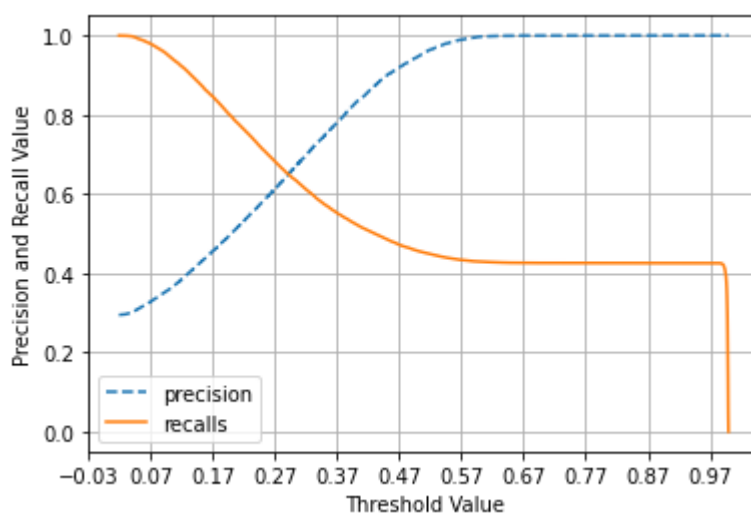
```
In [ ]: def precision_recall_curve_plot(y_test, pred_proba_c1):
    precisions, recalls, thresholds = precision_recall_curve(y_test, p

    threshold_boundary = thresholds.shape[0]
    # plot precision
    plt.plot(thresholds, precisions[0:threshold_boundary], linestyle='
    # 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 Va
    plt.legend(); plt.grid()
    plt.show()

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



Multicollinearity check using Variance Inflation Factor (VIF) -

Multicollinearity occurs when two or more independent variables are highly correlated with one another in a regression model. Multicollinearity can be a problem in a regression model because we would not be able to distinguish between the individual effects of the independent variables on the dependent variable.

Multicollinearity can be detected via various methods. One such method is Variance Inflation Factor aka VIF. In VIF method, we pick each independent feature and regress it against all of the other independent features. VIF score of an independent variable represents how well the variable is explained by other independent variables.

$$\text{VIF} = 1/(1-R^2)$$

```
In [ ]: 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 [57]:

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

Out [58]:

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

Out [59]:

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

Out[60]:

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

Out[61]:

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

Out[62]:

	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 [ ]: X = scaler.fit_transform(X)

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

Cross Validation accuracy: 0.891

Oversampling using SMOTE

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

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

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

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

```
In [ ]: 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

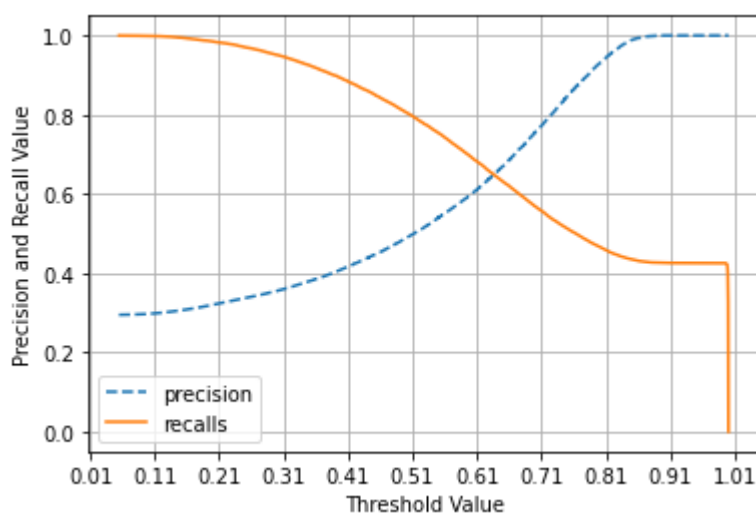

```
In [ ]: def precision_recall_curve_plot(y_test, pred_proba_c1):
    precisions, recalls, thresholds = precision_recall_curve(y_test, p

    threshold_boundary = thresholds.shape[0]
    # plot precision
    plt.plot(thresholds, precisions[0:threshold_boundary], linestyle='
    # 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 Va
    plt.legend(); plt.grid()
    plt.show()

precision_recall_curve_plot(y_test, lr1.predict_proba(X_test)[: ,1])
```



```
In [ ]: #alternative method
#We can try fixing the imbalance using class weights as well
#encoding
#threshold
```