LoanTap Business Case

Problem Statement

LoanTap deploys formal credit to salaried individuals and businesses 4 main financial instruments:

1. Personal Loan
2. EMI Free Loan
3. Personal Overdraft
4. Advance Salary Loan But the main focus is to interpret the underwriting process behind the Personal Loan only

Given a set of attributes for an Individual, determine if a credit line should be extended to them. If so, what should the repayment terms be in business recommendations?

- Additional views
  - We need to track the users previous credit line history and repayment status.
  - Analysing the previous loans tenure and the total liability.
  - As we are focusing more on salaried individual, we need to take salary of the person into consideration.

Installing Dependencies

In [232]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

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
```

Loading Dataset

In [233]:

```
loantap = pd.read_csv('https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/00

loantap.head(5)
```

Out[233]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	hor
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	

In [234]:

```
print(f"The dataset has {loantap.shape[0]} rows and {loantap.shape[1]} columns")
```

```
The dataset has 396030 rows and 27 columns
```

In [235]:

```
loantap.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 [236]:

```
loantap.dtypes
```

Out[236]:

```
loan_amnt               float64
term                     object
int_rate                float64
installment             float64
grade                    object
sub_grade                object
emp_title                object
emp_length               object
home_ownership           object
annual_inc              float64
verification_status      object
issue_d                  object
loan_status              object
purpose                  object
title                    object
dti                     float64
earliest_cr_line         object
open_acc                float64
pub_rec                 float64
revol_bal               float64
revol_util              float64
total_acc               float64
initial_list_status      object
application_type         object
mort_acc                float64
pub_rec_bankruptcies    float64
address                  object
dtype: object
```

In [237]:

```
loantap.duplicated().sum()
```

Out[237]:

```
0
```

Insights

- Dataset has no duplicate values

In [238]:

```
loantap.isnull().sum()
```

Out[238]:

```
loan_amnt                   0
term                        0
int_rate                    0
installment                 0
grade                       0
sub_grade                   0
emp_title               22927
emp_length              18301
home_ownership              0
annual_inc                  0
verification_status         0
issue_d                     0
loan_status                 0
purpose                     0
title                    1755
dti                         0
earliest_cr_line            0
open_acc                    0
pub_rec                     0
revol_bal                   0
revol_util                276
total_acc                   0
initial_list_status         0
application_type            0
mort_acc                37795
pub_rec_bankruptcies      535
address                     0
dtype: int64
```

Insights

- We have bunch of missing value attributes.

In [239]:

```
loantap.describe()
```

Out[239]:

	loan_amnt	int_rate	installment	annual_inc	dti	open
count	396030.000000	396030.000000	396030.000000	3.960300e+05	396030.000000	396030.00
mean	14113.888089	13.639400	431.849698	7.420318e+04	17.379514	11.31
std	8357.441341	4.472157	250.727790	6.163762e+04	18.019092	5.13
min	500.000000	5.320000	16.080000	0.000000e+00	0.000000	0.00
25%	8000.000000	10.490000	250.330000	4.500000e+04	11.280000	8.00
50%	12000.000000	13.330000	375.430000	6.400000e+04	16.910000	10.00
75%	20000.000000	16.490000	567.300000	9.000000e+04	22.980000	14.00
max	40000.000000	30.990000	1533.810000	8.706582e+06	9999.000000	90.00

Insights

- There is significant difference found in the mean and median of the following attributes
  - loan_amnt
  - terms
  - installment
  - revol_bal etc.
- These attributes might contain outliers

In [240]:

```
loantap.describe(include = 'object')
```

Out[240]:

	term	grade	sub_grade	emp_title	emp_length	home_ownership	verification_sta
count	396030	396030	396030	373103	377729	396030	396
unique	2	7	35	173105	11	6	
top	36 months	B	B3	Teacher	10+ years	MORTGAGE	Veri
freq	302005	116018	26655	4389	126041	198348	139

Insights

- Most of the loan disburesed for the 36 months period
- Most of the loan applicant have mortgage the home
- Majority of loans been fully paid off
- Majorily the loans been disbursed for the purpose of debt consolidation
- Most of the applicant is Individual

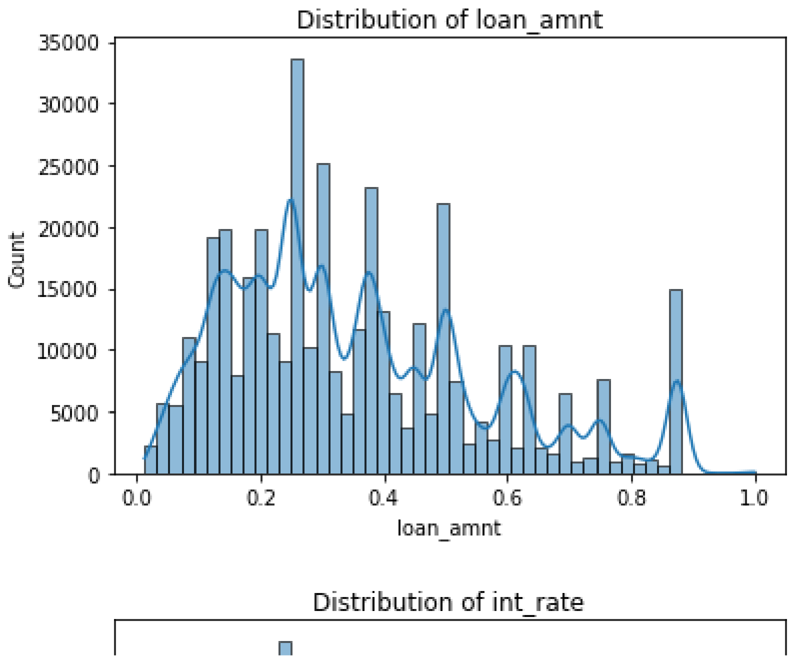
Visualization - Univariate Analysis

In [241]:

```
num_vars = loantap.select_dtypes('float64').columns.tolist()
```

In [242]:

```
for i in num_vars:
#     plt.figure(figsize=(12,5))
    plt.title("Distribution of {}".format(i))
    sns.histplot(loantap[i]/loantap[i].max(), kde=True, bins=50)
    plt.show()
```

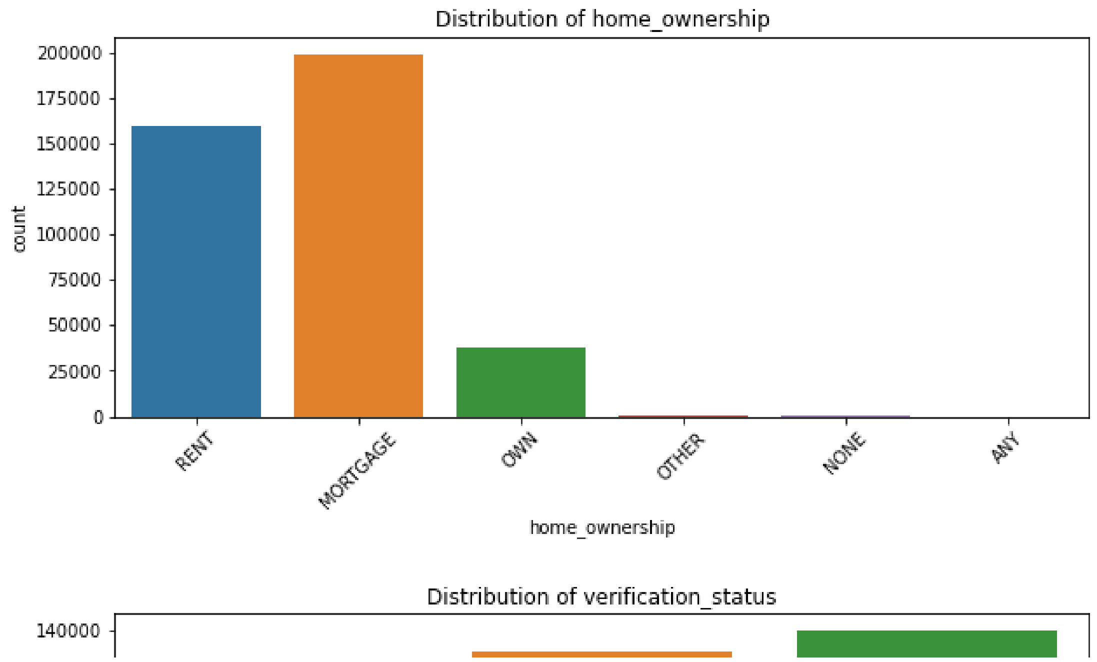


Insights

- Most of the distribution is highly skewed which tells us that they might contain outliers
- Almost all the continuous features have outliers present in the dataset.

In [248]:

```
cat_vars = ['home_ownership', 'verification_status', 'loan_status', 'application_type',
for i in cat_vars:
    plt.figure(figsize=(10, 4))
    plt.title(f'Distribution of {i}')
    sns.countplot(data=loantap, x=i)
    plt.xticks(rotation = 45)
    plt.show()
```



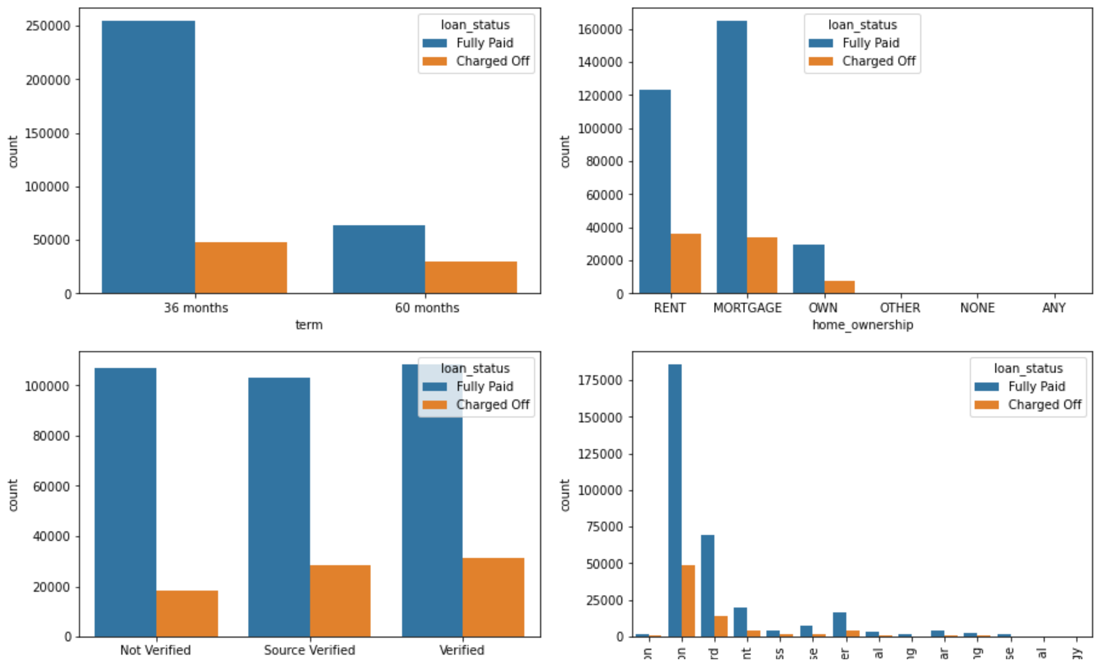
Insights

- All the application type is Individual
- Most of the loan tenure is disbursed for 36 months
- The grade of majority of people those who have took 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.

Visualization - Bivariate Analysis

In [192]:

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

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

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

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

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

plt.show()
```

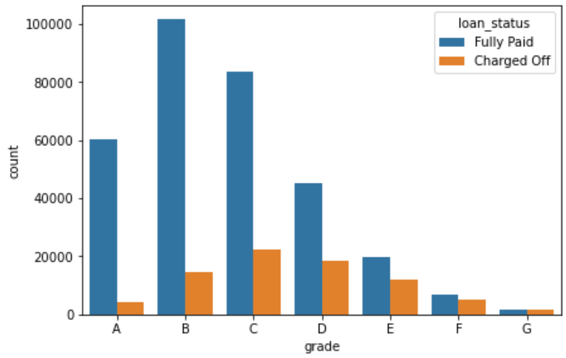


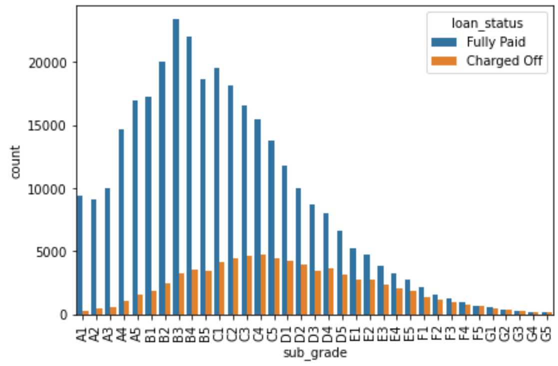
Insights

- Most of the people took loan for 36 months and full paid on time
- Most of people have home ownership as mortgage and rent
- Most of the people took loan for debt consolidations

In [193]:

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

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

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

plt.show()
```



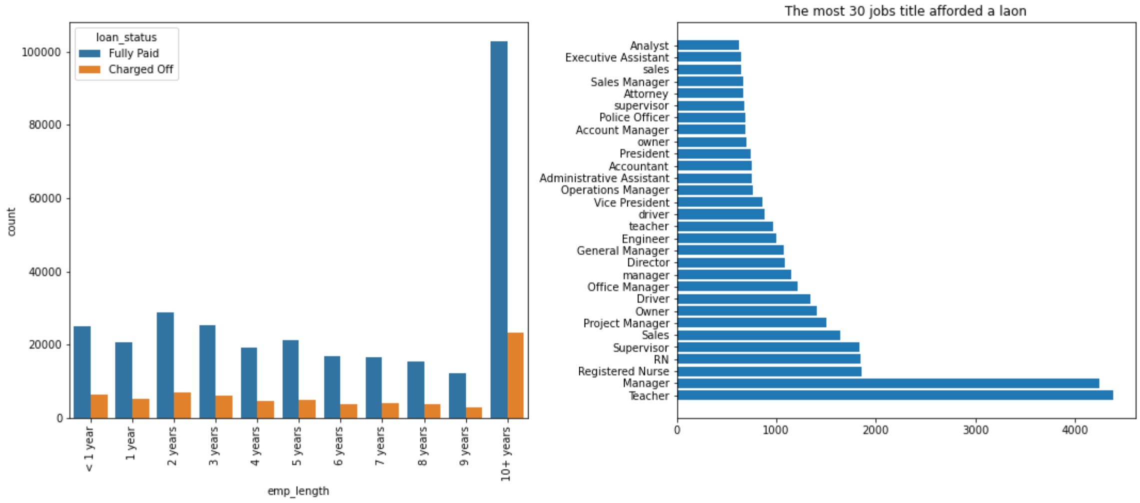


Insights

- 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 [194]:

```
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=loantap,hue='loan_status',order=order)
g.set_xticklabels(g.get_xticklabels(),rotation=90)

plt.subplot(2,2,2)
plt.barh(loantap.emp_title.value_counts()[:30].index,loantap.emp_title.value_counts()[:3
plt.title("The most 30 jobs title afforded a laon")
plt.tight_layout()

plt.show()
```



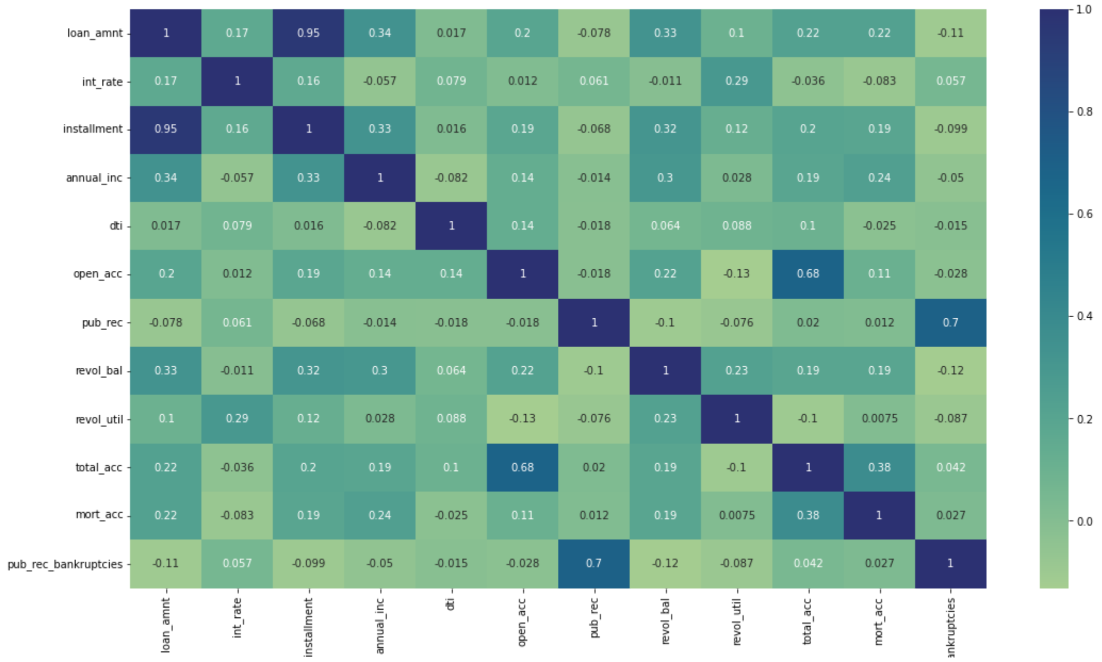
Insights

- Manager and Teacher are the most afforded loan on titles
- Person who employed for more than 10 years has successfully paid of the loan

Correlation Analysis

In [195]:

```
plt.figure(figsize=(18,10))
sns.heatmap(loantap.corr(), cmap = 'crest', annot = True)

plt.show()
```



Insights

- 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.

Action

- So, we can drop either one of those columns.

In [251]:

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

Data Preprocessing

Feature Engineering

In [255]:

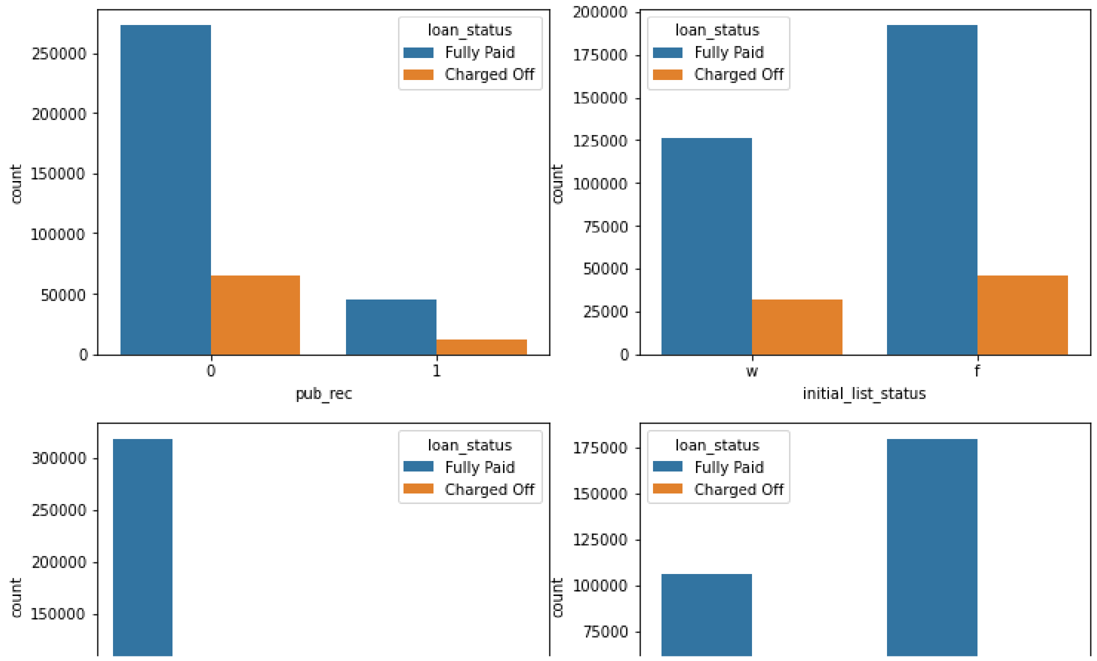
```
def pub_rec(number):
    if number == 0.0:
        return 0
    else:
        return 1

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 [256]:

```
loantap['pub_rec']=loantap.pub_rec.apply(pub_rec)
loantap['mort_acc']=loantap.mort_acc.apply(mort_acc)
loantap['pub_rec_bankruptcies']=loantap.pub_rec_bankruptcies.apply(pub_rec_bankruptcies)
```

In [257]:

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

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

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

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

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

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

plt.show()
```



Insights

- Most the loan disbursed to the people whose do not hold bankrupties record have successfully paid loan

Duplicate Value Check

In [258]:

```
loantap.duplicated().sum()
```

Out[258]:

```
0
```

Missing Value

In [259]:

```
loantap.isnull().sum()
```

Out[259]:

```
loan_amnt                 0
term                      0
int_rate                  0
installment               0
grade                     0
sub_grade                 0
emp_title             22927
emp_length            18301
home_ownership            0
annual_inc                0
verification_status       0
issue_d                   0
loan_status               0
purpose                   0
title                  1755
dti                       0
earliest_cr_line          0
open_acc                  0
```

Missing Value Treatment

In [260]:

```
loantap.groupby(by='total_acc').mean()
```

30.0	15291.652498	13.360931	463.442624	80665.063888	18.457582	12.741948	0.150279	18637.820
31.0	15660.474719	13.443894	472.728868	83198.869078	18.665579	13.006282	0.156146	19155.706
32.0	15951.201319	13.433517	481.732567	83937.167116	18.664672	13.328308	0.152534	19352.960
33.0	15842.878945	13.406691	478.161642	83769.588412	18.838364	13.653766	0.161829	19664.473
34.0	15936.263600	13.422136	478.669446	83687.041424	19.088163	13.845697	0.151954	19279.562
35.0	15837.175938	13.378931	477.717252	85325.993179	19.089714	14.036592	0.149878	19355.149
36.0	16009.535935	13.497613	483.729554	87135.720426	19.149287	14.290489	0.165113	20115.028
37.0	16045.838573	13.458075	483.003717	86854.675868	19.128513	14.620717	0.152153	19945.073
38.0	16128.565796	13.270925	486.242343	87087.704072	19.467156	14.900999	0.143333	21038.224
39.0	16485.390567	13.403869	493.991804	88412.590335	19.413753	15.305822	0.156227	20626.841
40.0	16001.817810	13.351270	483.326595	88271.187825	19.516301	15.585989	0.160539	20239.247
41.0	16568.463903	13.552407	497.469916	89301.841603	19.487146	15.848554	0.158962	21156.345

In [261]:

```
total_acc_avg=loantap.groupby(by='total_acc').mean().mort_acc
# saving mean of mort_acc according to total_acc_avg
def fill_mort_acc(total_acc,mort_acc):
    if np.isnan(mort_acc):
        return total_acc_avg[total_acc].round()
    else:
        return mort_acc
loantap['mort_acc']=loantap.apply(lambda x: fill_mort_acc(x['total_acc'],x['mort_acc']),
```

In [262]:

```
loantap.isnull().sum()
```

Out[262]:

```
loan_amnt                   0
term                        0
int_rate                    0
installment                 0
grade                       0
sub_grade                   0
emp_title               22927
emp_length              18301
home_ownership              0
annual_inc                  0
verification_status         0
issue_d                     0
loan_status                 0
purpose                     0
title                    1755
dti                         0
earliest_cr_line            0
open_acc                    0
```

Insights

- Dataset is very large so we can drop the rows with null values

In [263]:

```
# Dropping rows with null values
loantap.dropna(inplace=True)
# Remaining no. of rows
loantap.shape
```

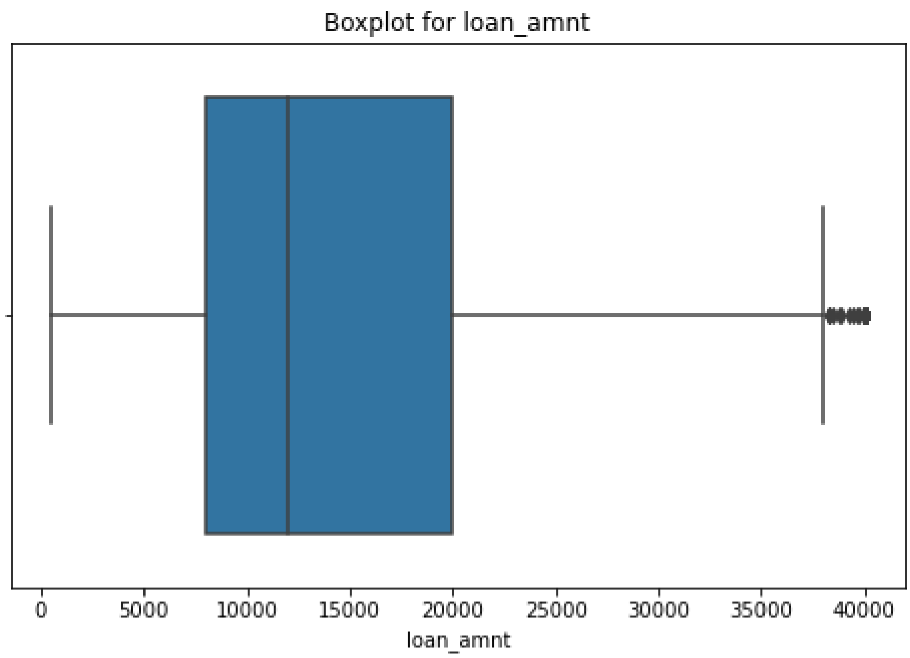
Out[263]:

```
(370622, 27)
```

Outlier Detection

In [264]:

```
def box_plot(col):
    plt.figure(figsize=(8,5))
    sns.boxplot(x=loantap[col])
    plt.title('Boxplot for {}'.format(col))
    plt.show()

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



Outlier Treatment

In [265]:

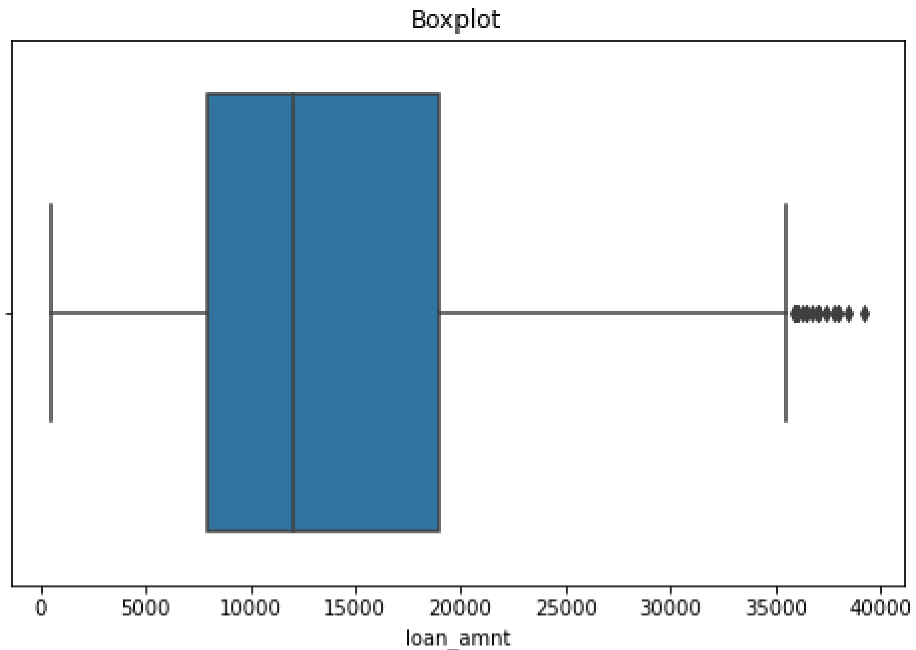
```
for col in num_vars:
    mean=loantap[col].mean()
    std=loantap[col].std()

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

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

loantap.shape
```

Out[265]:

```
(350358, 27)
```

In [266]:

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

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



In [267]:

```
# Converting term values to numerical val
term_values={' 36 months': 36, ' 60 months':60}
loantap['term'] = loantap.term.map(term_values)

# Mapping the target variable
loantap['loan_status']=loantap.loan_status.map({'Fully Paid':0, 'Charged Off':1})

# Initial List Status
loantap['initial_list_status'].unique()
np.array(['w', 'f'], dtype=object)
list_status = {'w': 0, 'f': 1}
loantap['initial_list_status'] = loantap.initial_list_status.map(list_status)

# Let's fetch ZIP from address and then drop the remaining details -
loantap['zip_code'] = loantap.address.apply(lambda x: x[-5:])
loantap['zip_code'].value_counts(normalize=True)*100
```

Out[267]:

```
70466    14.375296
30723    14.289669
22690    14.272259
48052    14.126979
00813    11.605558
29597    11.549044
05113    11.519075
93700     2.768597
11650     2.762888
86630     2.730636
Name: zip_code, dtype: float64
```

In [268]:

```
# Dropping some variables which we can let go for now
loantap.drop(columns=['issue_d', 'emp_title', 'title', 'sub_grade',
                      'address', 'earliest_cr_line', 'emp_length'],
                      axis=1, inplace=True)
```

One hot encoding

In [269]:

```
dummies=['purpose', 'zip_code', 'grade', 'verification_status', 'application_type', 'hom
data=pd.get_dummies(loantap,columns=dummies,drop_first=True)
pd.set_option('display.max_columns',None)
pd.set_option('display.max_rows',None)
```

Data processing for modelling

In [270]:

```
from sklearn.model_selection import train_test_split

X=data.drop('loan_status',axis=1)
y=data['loan_status']
X_train, X_test, y_train, y_test =train_test_split(X,y,test_size=0.30,stratify=y,random_
print(X_train.shape)
print(X_test.shape)
```

```
(245250, 51)
(105108, 51)
```

In [271]:

```
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Model Building

In [272]:

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

Out[272]:

```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

In [273]:

```
# X.columns.shape
# # logreg.coef_[0]
pd.Series((zip(X.columns, logreg.coef_[0])))
```

Out[273]:

```
0                      (loan_amnt, -0.14356459964004828)
1                             (term, 0.5389302062853516)
2                        (int_rate, 0.10172820951150907)
3                      (installment, 0.6706557163327765)
4                      (annual_inc, -1.1264337642098299)
5                              (dti, 1.0067797317649039)
6                         (open_acc, 0.7678516471957975)
7                          (pub_rec, 0.1993203792746295)
8                       (revol_bal, -0.49158538696898174)
9                       (revol_util, 0.46896593976659967)
10                      (total_acc, -0.6106469406833719)
11          (initial_list_status, -0.019547846656039085)
12                     (mort_acc, -0.060255365784687036)
13          (pub_rec_bankruptcies, -0.16521148949218148)
14             (purpose_credit_card, 0.19113889623686453)
15      (purpose_debt_consolidation, 0.2717077616352504)
16              (purpose_educational, 0.4831152902862432)
17         (purpose_home_improvement, 0.349891580984392)
```

In [274]:

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

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

Confusion Matrix

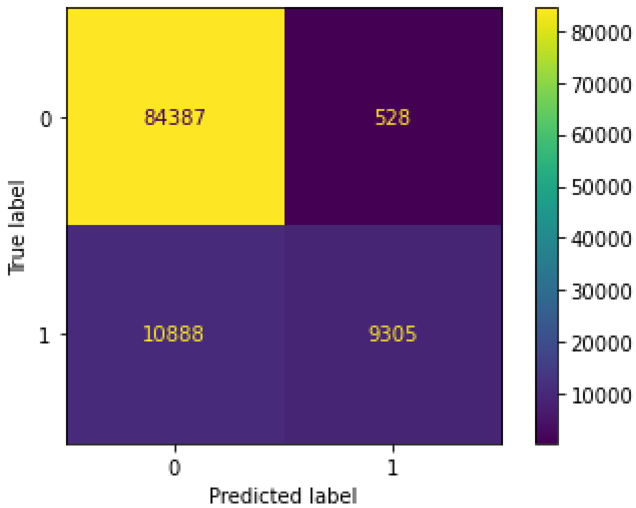
In [276]:

```
confusion_matrix=confusion_matrix(y_test,y_pred)
print(confusion_matrix)
ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=logreg.classes_
```

```
[[84387   528]
 [10888  9305]]
```

Out[276]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1df3d1
db550>
```



Insights

- There is significant value for false negative and false positive. Which will hamper our prediction due to type-1 or type-2 error.

Classification Report

In [277]:

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.99      0.94     84915
           1       0.95      0.46      0.62     20193

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

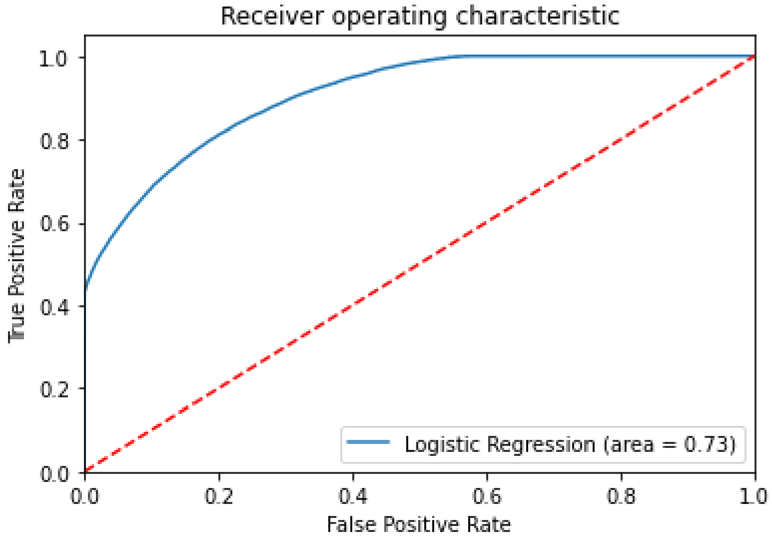
Insights

- Precision score and recall score for full paid status is almost same indicates that model is doing decent job which correctly classified the both of the scenarios

ROC / AUC

In [219]:

```
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()
```



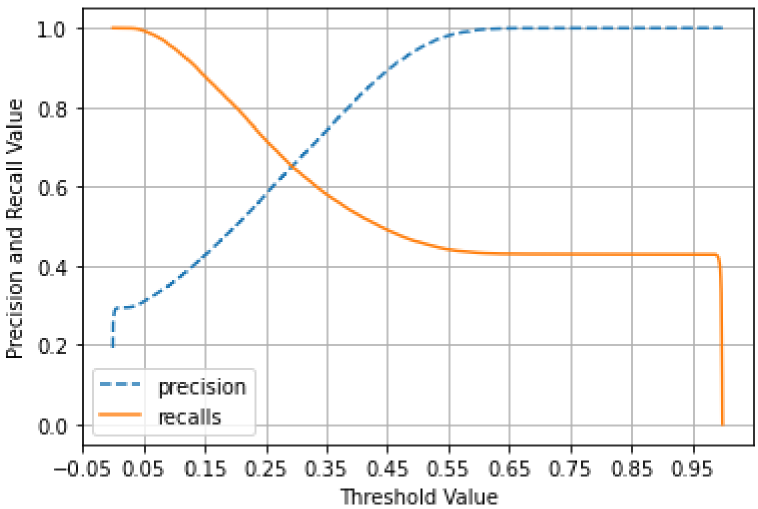
Insights

- `ROC-AUC` curve is grossing the area near about 0.73 which indicates that model is performing well.
- There is still room for some model improvement
- By collecting more data, using a more complex model, or tuning the hyperparameters, it is possible to improve the model's performance.

Precision-Recall Curve

In [220]:

```
def precission_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='precision
    #plot recall
    plt.plot(thresholds,recalls[0:threshold_boundary],label='recalls')

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

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

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



Insights

- Precision score is highest at 0.55 threshold. High precision value indicates that model is positevly predicating the charged off loan status which helps business to take more stable decision.
- Recall score is higher on smaller threshold but after 0.55 the recall value is constant. Model is correctly classifying the actual predicated values as instances.

Assumption of Log. Reg. (Multicollinearity Check)

In [41]:

```
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[41]:

	Feature	VIF
44	application_type_INDIVIDUAL	5012.25
46	home_ownership_MORTGAGE	2576.84
50	home_ownership_RENT	2172.79
49	home_ownership_OWN	468.51
0	loan_amnt	241.19

In [42]:

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

Out[42]:

	Feature	VIF
0	loan_amnt	241.19
3	installment	217.64
2	int_rate	130.35
1	term	126.76
45	home_ownership_MORTGAGE	102.91

In [43]:

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

Out[43]:

	Feature	VIF
1	int_rate	123.55
44	home_ownership_MORTGAGE	78.86
48	home_ownership_RENT	63.78
14	purpose_debt_consolidation	51.20
0	term	25.86

In [44]:

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

Out[44]:

	Feature	VIF
43	home_ownership_MORTGAGE	66.11
47	home_ownership_RENT	53.10
13	purpose_debt_consolidation	51.20
0	term	25.78
12	purpose_credit_card	18.62

In [45]:

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

Out[45]:

	Feature	VIF
13	purpose_debt_consolidation	22.73
0	term	22.21
4	open_acc	13.61
8	total_acc	12.66
7	revol_util	8.99

In [46]:

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

Out[46]:

	Feature	VIF
0	term	17.95
4	open_acc	13.17
8	total_acc	12.65
7	revol_util	8.28
2	annual_inc	7.90

In [47]:

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

Out[47]:

	Feature	VIF
3	open_acc	13.09
7	total_acc	12.60
6	revol_util	8.25
1	annual_inc	7.60
2	dti	7.58

In [48]:

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

Out[48]:

	Feature	VIF
6	total_acc	8.23
5	revol_util	7.94
1	annual_inc	7.52
2	dti	7.02
0	installment	6.64

Validation using KFold

In [49]:

```
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=-1))
print("Cross Validation accuracy : {:.3f}".format(accuracy))
```

```
Cross Validation accuracy : 0.891
```

Insights

- Cross Validation accuracy and testing accuracy is almost same which infers model is performing the decent job.

Oversampling using SMOTE

In [50]:

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

In [51]:

```
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: (396268, 51)
After OverSampling, the shape of train_y: (396268,)

After OverSampling, counts of label '1': 198134
After OverSampling, counts of label '0': 198134
```

In [52]:

```
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))
```

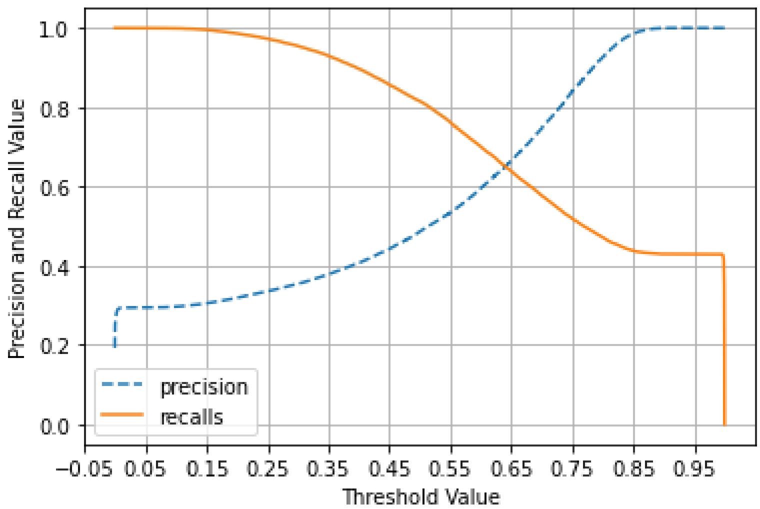
```
              precision    recall  f1-score   support

           0       0.95      0.80      0.86     84915
           1       0.49      0.82      0.61     20193

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

Insights

- After making the dataset balanced, the precision and recall score are same as imbalanced dataset. But the accuracy dropped.
- There is still room for improvement.

In [53]:

```
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='precis
    # 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])
```



Insights

- After balancing the dataset, there is significant change observed in the precion and recall score for both of the classes.
- Precision score is .95 and .49 for full paid and charged off respectively.

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 imbalances by making the data balance we 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 defualers.
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 [283]:

```
coefs = lr1.coef_.tolist()[0]
feature_coef_df = pd.DataFrame({'Variable': X.columns, 'Coeficient': coefs})
feature_coef_df.sort_values(by=['Coeficient'], ascending=False)
```

Out[283]:

	Variable	Coeficient
35	zip_code_93700	14.024452
28	zip_code_11650	13.996881
34	zip_code_86630	13.885951
32	zip_code_48052	6.155181
33	zip_code_70466	6.131976
29	zip_code_22690	6.116895
31	zip_code_30723	6.102021
41	grade_G	1.402764
40	grade_F	1.370776
39	grade_E	1.315705

Actional Insights and Recommendations

1. 80% of the customers have paid the loan fully.
2. 20% of the customers are the defaulters.
3. The organization can the trained model to make prediction for whether a person will likely to pay the loan amount or 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 infers model is performing the decent job. We can trust this model for unseen data
7. By collecting more data, using a more complex model, or tuning the hyperparameters, it is possible to improve the model's performance.
8. ROC AUC curve area of 0.73, the model is correctly classifying about 73% of the instances. 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 trade-off as we vary the threshold. A higher threshold will result in higher precision, but lower recall, and vice versa. The ideal point on the curve is the one that best meets the needs of the specific application.
10. After balancing the dataset, there is significant change observed in the precion and recall score for both of the classes.
11. Accuracy of Logistic Regression Classifier on test set: 0.891 which is decent and not by chance.

In [ ]:

```

```