Problem Statement

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?

Data Dictionary

```
1) loan_amnt : The listed amount of the loan applied for by the borrower. If at so
me point in time, the credit department reduces the loan amount, then it will be r
eflected in this value.
2) term : The number of payments on the loan. Values are in months and can be eith
er 36 or 60.
3) int_rate : Interest Rate on the loan
4) installment : The monthly payment owed by the borrower if the loan originates.
5) grade : LoanTap assigned loan grade
6) sub_grade : LoanTap assigned loan subgrade
7) emp_title :The job title supplied by the Borrower when applying for the loan.*
8) emp_length : Employment length in years. Possible values are between 0 and 10 w
here 0 means less than one year and 10 means ten or more years.
9) home_ownership : The home ownership status provided by the borrower during regi
stration or obtained from the credit report.
10) annual_inc : The self-reported annual income provided by the borrower during r
egistration.
11) verification_status : Indicates if income was verified by LoanTap, not verifie
d, or if the income source was verified
12) issue_d : The month which the loan was funded
13) loan_status : Current status of the loan - Target Variable
14) purpose : A category provided by the borrower for the loan request.
15) title : The loan title provided by the borrower
16) dti : A ratio calculated using the borrower’s total monthly debt payments on t
he total debt obligations, excluding mortgage and the requested LoanTap loan, divi
ded by the borrower’s self-reported monthly income.
17) earliest_cr_line :The month the borrower's earliest reported credit line was o
pened
18) open_acc : The number of open credit lines in the borrower's credit file.
19) pub_rec : Number of derogatory public records
20) revol_bal : Total credit revolving balance
21) revol_util : Revolving line utilization rate, or the amount of credit the borr
ower is using relative to all available revolving credit.
22) total_acc : The total number of credit lines currently in the borrower's credi
t file
23) initial_list_status : The initial listing status of the loan. Possible values
 are – W, F
24) application_type : Indicates whether the loan is an individual application or
 a joint application with two co-borrowers
25) mort_acc : Number of mortgage accounts.
```

26) pub_rec_bankruptcies : Number of public record bankruptcies
27) Address: Address of the individual

Concept Used

- Exploratory Data Analysis
- Feature Engineering
- Logistic Regression
- Precision Vs Recall Tradeoff

Questionnaire

1) What percentage of customers have fully paid their Loan Amount?
2) Comment about the correlation between Loan Amount and Installment features.
3) The majority of people have home ownership as _____.
4) People with grades ‘A’ are more likely to fully pay their loan. (T/F)
5) Name the top 2 afforded job titles.
6) Thinking from a bank's perspective, which metric should our primary focus be o
n..
    1) ROC AUC
    2) Precision
    3) Recall
    4) F1 Score
7) How does the gap in precision and recall affect the bank?
8) Which were the features that heavily affected the outcome?
9) Will the results be affected by geographical location? (Yes/No)

In [ ]:

Exploratory Data Analysis

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style='whitegrid')
from numpy import NaN, nan, NAN
from scipy import stats
import statsmodels.api as sm
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')
from sklearn.pipeline import make_pipeline
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
```

loading data into dataframe

In [2]:

```
loan_data=pd.read_csv('logistic_regression.csv')
```

In [3]:

```
loan_data.head()
```

Out[3]:

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

5 rows × 27 columns

Identification of variables and data types:

In [4]:

```
loan_data.shape
```

Out[4]:

```
(396030, 27)
```

In [5]:

```
loan_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             396030 non-null  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 [6]:

```
def feature_names():

    print(f"Columns with category datatypes (Categorical Features) are : \
    {list(loan_data.select_dtypes('object').columns)}")
    print('-'*100)
    print('-'*100)
    print(f"Columns with integer and float datatypes (Numerical Features) are: \
    {list(loan_data.select_dtypes(['int64','float64']).columns)}")
```

In [7]:

```
feature_names()
```

```
Columns with category datatypes (Categorical Features) are :     ['term', 'g
rade', 'sub_grade', 'emp_title', 'emp_length', 'home_ownership', 'verificati
on_status', 'issue_d', 'loan_status', 'purpose', 'title', 'earliest_cr_lin
e', 'initial_list_status', 'application_type', 'address']
------------------------------------------------------------------------------
-----------------------
------------------------------------------------------------------------------
-----------------------
Columns with integer and float datatypes (Numerical Features) are:     ['loa
n_amnt', 'int_rate', 'installment', 'annual_inc', 'dti', 'open_acc', 'pub_re
c', 'revol_bal', 'revol_util', 'total_acc', 'mort_acc', 'pub_rec_bankruptcie
s']
```

Analysing the basic metrics:

In [8]:

```
loan_data.describe(include=[np.number]).transpose()
```

Out[8]:

	count	mean	std	min	25%	50%	7
loan_amnt	396030.0	14113.888089	8357.441341	500.00	8000.00	12000.00	20000
int_rate	396030.0	13.639400	4.472157	5.32	10.49	13.33	16
installment	396030.0	431.849698	250.727790	16.08	250.33	375.43	567
annual_inc	396030.0	74203.175798	61637.621158	0.00	45000.00	64000.00	90000
dti	396030.0	17.379514	18.019092	0.00	11.28	16.91	22
open_acc	396030.0	11.311153	5.137649	0.00	8.00	10.00	14
pub_rec	396030.0	0.178191	0.530671	0.00	0.00	0.00	0
revol_bal	396030.0	15844.539853	20591.836109	0.00	6025.00	11181.00	19620
revol_util	395754.0	53.791749	24.452193	0.00	35.80	54.80	72
total_acc	396030.0	25.414744	11.886991	2.00	17.00	24.00	32
mort_acc	358235.0	1.813991	2.147930	0.00	0.00	1.00	3
pub_rec_bankruptcies	395495.0	0.121648	0.356174	0.00	0.00	0.00	0

In [9]:

```
loan_data.describe(include=[object]).transpose()
```

Out[9]:

	count	unique	top	freq
term	396030	2	36 months	302005
grade	396030	7	B	116018
sub_grade	396030	35	B3	26655
emp_title	373103	173105	Teacher	4389
emp_length	377729	11	10+ years	126041
home_ownership	396030	6	MORTGAGE	198348
verification_status	396030	3	Verified	139563
issue_d	396030	115	Oct-2014	14846
loan_status	396030	2	Fully Paid	318357
purpose	396030	14	debt_consolidation	234507
title	394275	48817	Debt consolidation	152472
earliest_cr_line	396030	684	Oct-2000	3017
initial_list_status	396030	2	f	238066
application_type	396030	3	INDIVIDUAL	395319
address	396030	393700	USCGC Smith\r\nFPO AE 70466	8

Missing values and outlier treatment:

In [10]:

```
def missingValue(df):
    #Identifying Missing data.
    total_null = df.isnull().sum().sort_values(ascending = False)
    percent = ((df.isnull().sum()/len(df))*100).sort_values(ascending = False)
    print(f"Total records in our data =  {df.shape[0]} where missing values are as follows:

    missing_data = pd.concat([total_null,percent.round(2)],axis=1,keys=['Total Missing','In
    return missing_data
```

In [11]:

```
missing_df = missingValue(loan_data)
missing_df[missing_df['Total Missing'] > 0]
```

Total records in our data = 396030 where missing values are as follows:

Out[11]:

	Total Missing	In Percent
mort_acc	37795	9.54
emp_title	22927	5.79
emp_length	18301	4.62
title	1755	0.44
pub_rec_bankruptcies	535	0.14
revol_util	276	0.07

3 numerical and 3 categorical features are having missing values

Dealing with emp_title:

In [12]:

```
top_10 = loan_data['emp_title'].value_counts().head(10)
top_10
```

Out[12]:

```
Teacher             4389
Manager             4250
Registered Nurse    1856
RN                  1846
Supervisor          1830
Sales               1638
Project Manager     1505
Owner               1410
Driver              1339
Office Manager      1218
Name: emp_title, dtype: int64
```

In [13]:

```
top_10_df = pd.DataFrame({'frequency':top_10})
top10_dfnewest = top_10_df.reset_index()
top10_dflatest = top10_dfnewest.rename(columns = {'index':'emp_title'})
top10_dflatest
```

Out[13]:

	emp_title	frequency
0	Teacher	4389
1	Manager	4250
2	Registered Nurse	1856
3	RN	1846
4	Supervisor	1830
5	Sales	1638
6	Project Manager	1505
7	Owner	1410
8	Driver	1339
9	Office Manager	1218

In [14]:

```
df_latest = pd.merge(top10_dflatest,loan_data, how = 'inner', on = 'emp_title')
df_latest.head()
```

Out[14]:

	emp_title	frequency	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_lengtl
0	Teacher	4389	15000.0	36 months	9.99	483.94	B	B1	10+ years
1	Teacher	4389	35000.0	36 months	14.65	1207.30	C	C5	10+ years
2	Teacher	4389	28625.0	36 months	11.99	950.63	B	B3	10+ years
3	Teacher	4389	11000.0	36 months	11.44	362.43	B	B4	10+ years
4	Teacher	4389	2000.0	36 months	14.98	69.32	C	C3	5 years

5 rows × 28 columns

In [15]:

```
top10_dflatest = df_latest[df_latest['loan_status'] == 'Fully Paid']
top10_dflatest.head()
```

Out[15]:

	emp_title	frequency	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_lengtl
0	Teacher	4389	15000.0	36 months	9.99	483.94	B	B1	10+ year
1	Teacher	4389	35000.0	36 months	14.65	1207.30	C	C5	10+ year
2	Teacher	4389	28625.0	36 months	11.99	950.63	B	B3	10+ year
3	Teacher	4389	11000.0	36 months	11.44	362.43	B	B4	10+ year
4	Teacher	4389	2000.0	36 months	14.98	69.32	C	C3	5 year

5 rows × 28 columns

In [16]:

```
top10_dflatest = top10_dflatest['emp_title'].value_counts()
top10_dflatest
```

Out[16]:

```
Teacher              3532
Manager              3321
Registered Nurse     1476
RN                   1467
Supervisor           1425
Project Manager      1259
Sales                1239
Office Manager        970
Driver                961
Owner                 954
Name: emp_title, dtype: int64
```

In [17]:

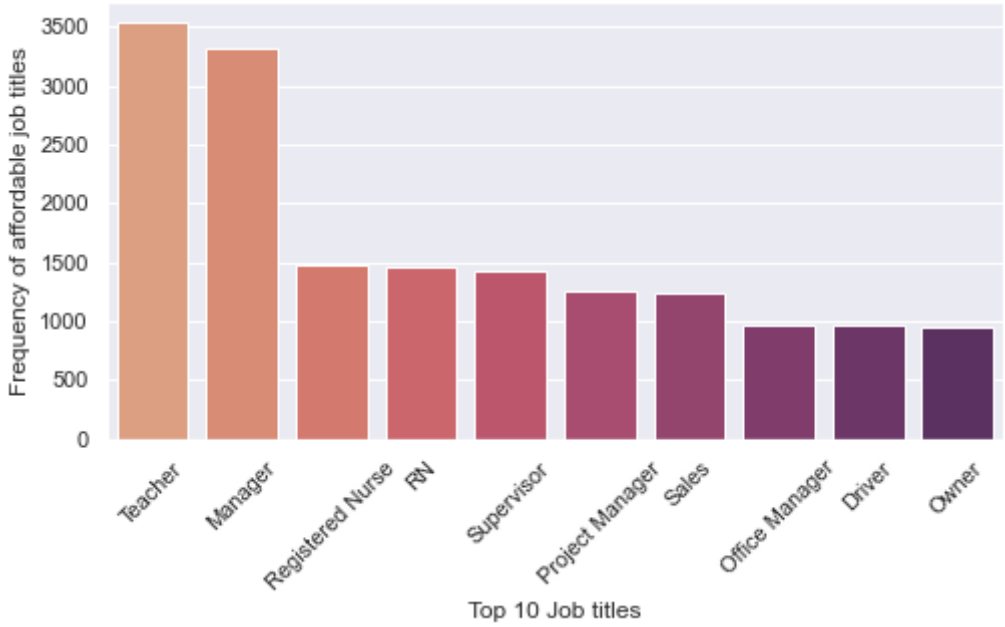
```
top_10_df = pd.DataFrame({'Frequency of affordable job titles':top10_dflatest})
top10_dfnewest = top_10_df.reset_index()
top10_dflatest = top10_dfnewest.rename(columns = {'index':'Job Roles for fully paid'})
top10_dflatest
```

Out[17]:

	Job Roles for fully paid	Frequency of affordable job titles
0	Teacher	3532
1	Manager	3321
2	Registered Nurse	1476
3	RN	1467
4	Supervisor	1425
5	Project Manager	1259
6	Sales	1239
7	Office Manager	970
8	Driver	961
9	Owner	954

In [18]:

```
plt.figure(figsize=(8,4))
sns.set_theme(style="darkgrid")
sns.barplot(data = top10_dflatest,x = 'Job Roles for fully paid', y ='Frequency of affordab
plt.xticks(rotation = 45)
plt.xlabel('Top 10 Job titles')
plt.show()
```



In [19]:

```
# dropping emp_title for model as almost half are unique values
loan_data = loan_data.drop('emp_title', axis = 1)
```

Insights:

* Name the top 2 afforded job titles - Teacher and Manager
* This feature will not be informative because half people (173k) have unique titles, so we have dropped it

Dealing with emp_length:

In [20]:

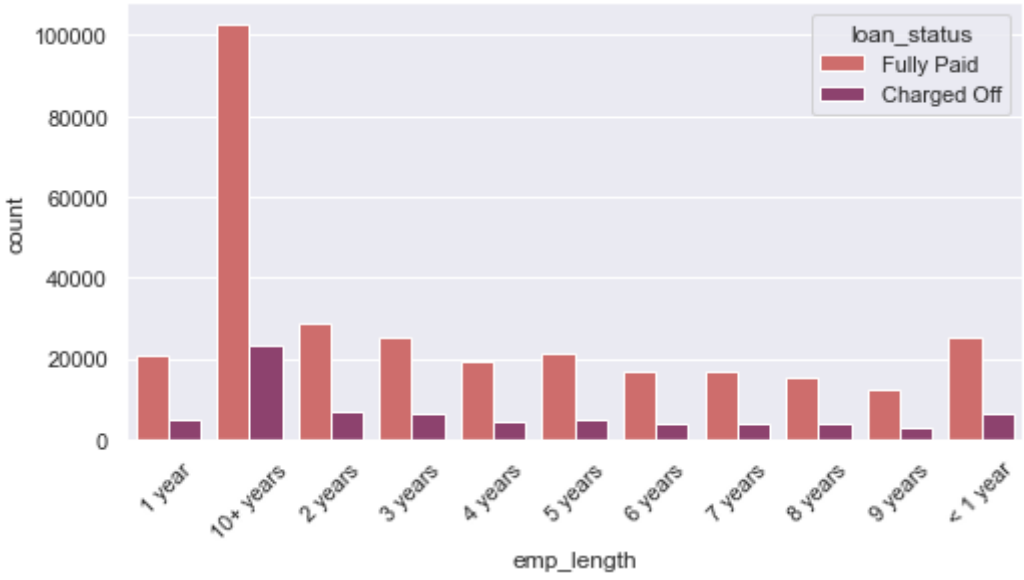
```
emp_length_sorted = sorted(loan_data['emp_length'].dropna().unique())
emp_length_sorted
```

Out[20]:

```
['1 year',
 '10+ years',
 '2 years',
 '3 years',
 '4 years',
 '5 years',
 '6 years',
 '7 years',
 '8 years',
 '9 years',
 '< 1 year']
```

In [21]:

```
plt.figure(figsize=(8,4))
sns.countplot(x='emp_length',data=loan_data,order=emp_length_sorted,hue='loan_status',palet
plt.xticks(rotation = 45)
plt.show()
```



In [22]:

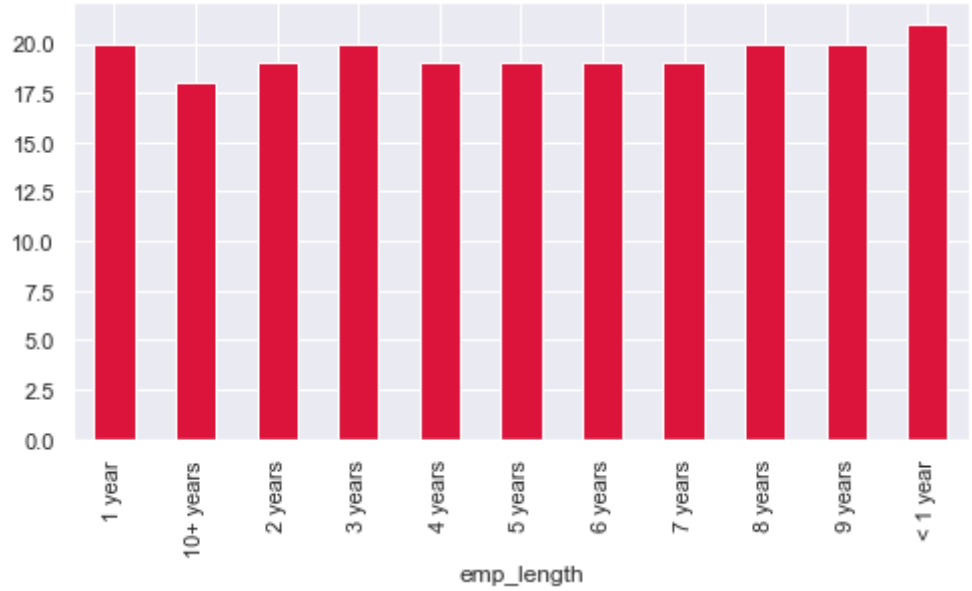
```
emp_co = loan_data[loan_data['loan_status']=='Charged Off'].groupby('emp_length').count()['
emp_fp = loan_data[loan_data['loan_status']=='Fully Paid'].groupby('emp_length').count()['l
emp_len = emp_co/(emp_fp+emp_co) * 100
pd.DataFrame(emp_len.round())
```

Out[22]:

emp_length	loan_status
1 year	20.0
10+ years	18.0
2 years	19.0
3 years	20.0
4 years	19.0
5 years	19.0
6 years	19.0
7 years	19.0
8 years	20.0
9 years	20.0
< 1 year	21.0

In [23]:

```
plt.figure(figsize=(8,4))
emp_len.round().plot(kind='bar',color='crimson')
plt.xticks(rotation=90)
plt.show()
```



In [24]:

```
#drop emp_len
loan_data=loan_data.drop('emp_length',axis=1)
```

Insights:

```
    * Customers majoratily working for 10+ year
    * This still doesnt really inform us if there is a strong reltionship betweem
 employment length and being charge off
      what we want is the percentage of charge offs per category. Essentially info
rming us what percent of people per
      employment category didn't pay back their loan.
    * Charge off rates are extremely similar across all employment lengths.
    * If the ratio between the two bars is the same across all categories, then th
is feature will not be very informative
      and we can safely drop it
```

Dealing with title:

In [25]:

```
loan_data['title'].value_counts().head(10)
```

Out[25]:

```
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
Name: title, dtype: int64
```

In [26]:

```
loan_data['title'].value_counts().nunique()
```

Out[26]:

```
209
```

In [27]:

```
print(loan_data['title'].head(10))
print('-'*30)
print('-'*30)
print(loan_data['purpose'].head(10))
```

```
0                   Vacation
1         Debt consolidation
2    Credit card refinancing
3    Credit card refinancing
4      Credit Card Refinance
5         Debt consolidation
6           Home improvement
7       No More Credit Cards
8         Debt consolidation
9         Debt Consolidation
Name: title, dtype: object
------------------------------
------------------------------
0             vacation
1    debt_consolidation
2           credit_card
3           credit_card
4           credit_card
5    debt_consolidation
6      home_improvement
7           credit_card
8    debt_consolidation
9    debt_consolidation
Name: purpose, dtype: object
```

In [28]:

```
#Dropping title as it is showing similar as purpose
loan_data = loan_data.drop('title',axis=1)
```

Dealing with mort_acc:

In [29]:

```
loan_data_copy1 = loan_data.copy()
```

As we need to do median imputation after the train test split, on train data we won't be able to perform median imputation on overall data.`

Dealing with revol_util and the pub_rec_bankruptcies:

In [30]:

```
# For the other two, just dropping the NAN columns as the data is less than 0.2 %
loan_data = loan_data.dropna(axis = 0,how = 'any',subset = ['revol_util','pub_rec_bankruptc
```

In [31]:

```
loan_data.shape
```

Out[31]:

```
(395219, 24)
```

In [32]:

```
loan_data.columns
```

Out[32]:

```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grade',
       'home_ownership', 'annual_inc', 'verification_status', 'issue_d',
       'loan_status', 'purpose', 'dti', 'earliest_cr_line', 'open_acc',
       'pub_rec', 'revol_bal', 'revol_util', 'total_acc',
       'initial_list_status', 'application_type', 'mort_acc',
       'pub_rec_bankruptcies', 'address'],
      dtype='object')
```

Insights:

revol_util and the pub_rec_bankruptcies have missing data points, but they acc
ount for less than 0.2% of the total
data. ahead Hence we will remove the rows that are missing those values in tho
se columns with dropna().

In [33]:

```
# Cross checking
missingValue(loan_data)
```

```
Total records in our data =  395219 where missing values are as follows:
```

Out[33]:

	Total Missing	In Percent
mort_acc	37205	9.41
loan_amnt	0	0.00
term	0	0.00
pub_rec_bankruptcies	0	0.00
application_type	0	0.00
initial_list_status	0	0.00
total_acc	0	0.00
revol_util	0	0.00
revol_bal	0	0.00
pub_rec	0	0.00
open_acc	0	0.00
earliest_cr_line	0	0.00
dti	0	0.00
purpose	0	0.00
loan_status	0	0.00
issue_d	0	0.00
verification_status	0	0.00
annual_inc	0	0.00
home_ownership	0	0.00
sub_grade	0	0.00
grade	0	0.00
installment	0	0.00
int_rate	0	0.00
address	0	0.00

In [34]:

```
# Checking percentage of Fully paid and defaulted customers

val_counts  = loan_data['loan_status'].value_counts(normalize = True) * 100
pd.DataFrame(val_counts.round())
```

Out[34]:

	loan_status
Fully Paid	80.0
Charged Off	20.0

Insights:

The data is clearly imbalanced. This is a typical scenario when dealing with problems related to loan default, spam
or fraud detection. We can expect to do very well in terms of accuracy, but our precision and recall will be the
true matrix that our model should be evaluated based on. We do not expect the model to perform very well on those
metrics.

Univariate Analysis

In [35]:

```
def numerical_feat(df,colname,nrows=2,mcols=2,width=15,height=15):
    fig , ax = plt.subplots(nrows,mcols,figsize=(width,height))
    fig.set_facecolor("peachpuff")
    rows = 0
    for var in colname:
        ax[rows][0].set_title("Boxplot for Outlier Detection ", fontweight="bold")
        plt.ylabel(var, fontsize=12)
        sns.boxplot(y = df[var],color='crimson',ax=ax[rows][0])

        # plt.subplot(nrows,mcols,pltcounter+1)
        sns.distplot(df[var],color='purple',ax=ax[rows][1])
        ax[rows][1].axvline(df[var].mean(), color='r', linestyle='--', label="Mean")
        ax[rows][1].axvline(df[var].median(), color='m', linestyle='-', label="Median")
        ax[rows][1].axvline(df[var].mode()[0], color='royalblue', linestyle='-', label="Mod
        ax[rows][1].set_title("Outlier Detection ", fontweight="bold")
        ax[rows][1].legend({'Mean':df[var].mean(),'Median':df[var].median(),'Mode':df[var].
        rows += 1
    plt.show()
```

In [36]:

```
feature_names()
```

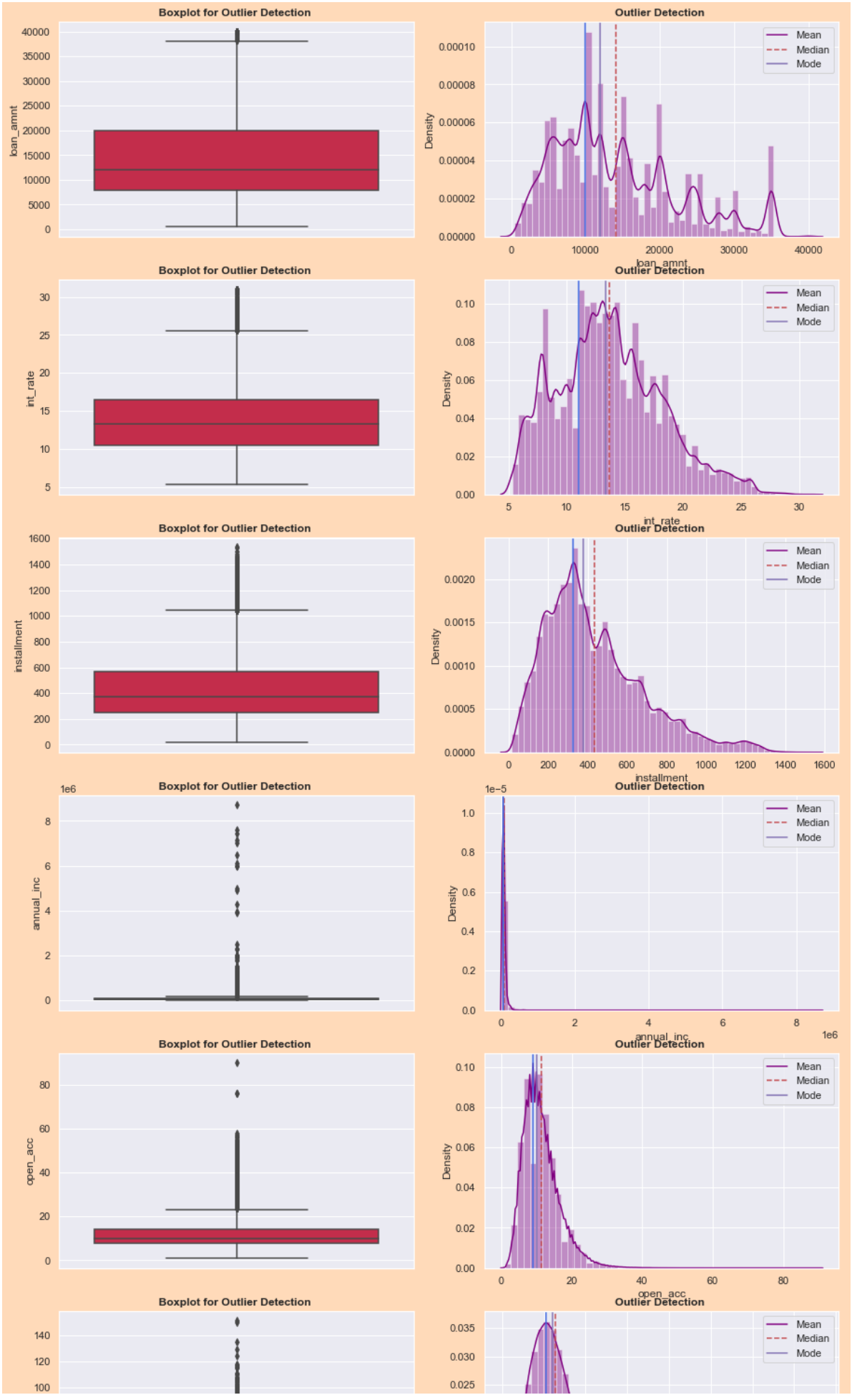
```
Columns with category datatypes (Categorical Features) are :     ['term', 'g
rade', 'sub_grade', 'home_ownership', 'verification_status', 'issue_d', 'loa
n_status', 'purpose', 'earliest_cr_line', 'initial_list_status', 'applicatio
n_type', 'address']
------------------------------------------------------------------------------
------------------------
------------------------------------------------------------------------------
------------------------
Columns with integer and float datatypes (Numerical Features) are:     ['loa
n_amnt', 'int_rate', 'installment', 'annual_inc', 'dti', 'open_acc', 'pub_re
c', 'revol_bal', 'revol_util', 'total_acc', 'mort_acc', 'pub_rec_bankruptcie
s']
```

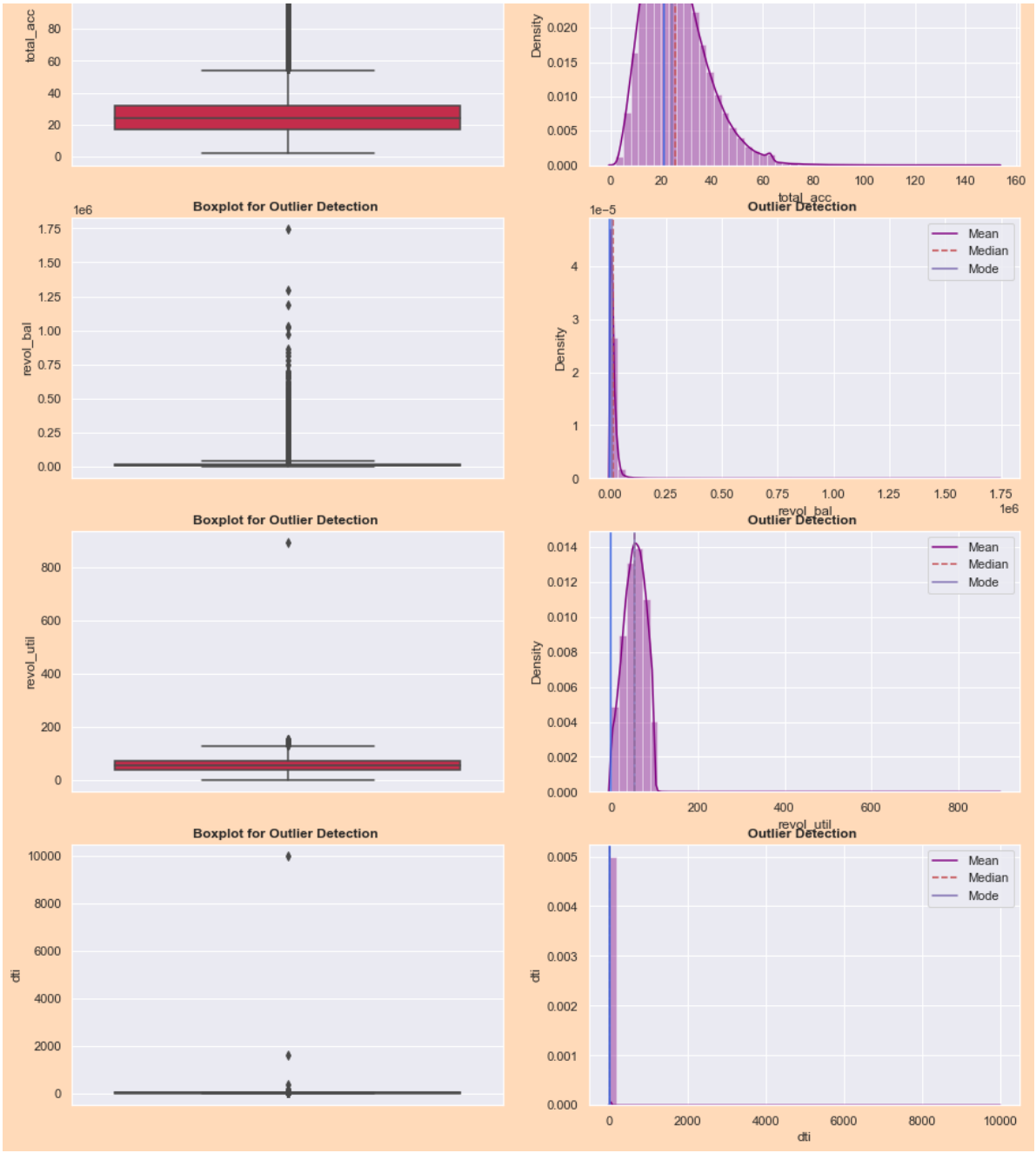
In [37]:

```
numerical_cols = ['loan_amnt', 'int_rate', 'installment', 'annual_inc', 'open_acc', 'total_
```

In [38]:

```
numerical_feat(loan_data,numerical_cols,len(numerical_cols),2,15,45)
```





In [39]:

```
loan_data.columns
```

Out[39]:

```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grade',
       'home_ownership', 'annual_inc', 'verification_status', 'issue_d',
       'loan_status', 'purpose', 'dti', 'earliest_cr_line', 'open_acc',
       'pub_rec', 'revol_bal', 'revol_util', 'total_acc',
       'initial_list_status', 'application_type', 'mort_acc',
       'pub_rec_bankruptcies', 'address'],
      dtype='object')
```

Insights:

* Above plots can be interpreted as we have many outliers and hence we need to handle them post train-test split
* As outliers detection and treatment can be done after train test split we wont fo the iqr based median imputation
for outliers. We will treat after split

Outliers detection and treatment :

In [40]:

```
# creating the new copy of our original data and will use this copy for further processing
loan_data_new = loan_data.copy()
```

In [41]:

```
loan_data.shape
```

Out[41]:

```
(395219, 24)
```

In [42]:

```
q1 = loan_data_new[numerical_cols].quantile(0.25)
q3 = loan_data_new[numerical_cols].quantile(0.75)
iqr = q3 - q1

loan_data_new = loan_data_new[~((loan_data_new[numerical_cols]<q1-1.5*iqr)|(loan_data_new[n
                                .any(axis=1)]
loan_data_new=loan_data_new.reset_index(drop = True)
```

In [43]:

```
loan_data.shape[0]-loan_data_new.shape[0]
```

Out[43]:

```
54757
```

In [44]:

```
percenatge_removal=(39931/396030)*100
percenatge_removal
```

Out[44]:

```
10.082822008433705
```

Insights:

If we remove the outliers of the data using IQR, then around 10% of the data is getting removed. Hence we wont
remove the outliers. We will treat it after train-test split.

In [45]:

```
loan_data.select_dtypes('number').columns
```

Out[45]:

```
Index(['loan_amnt', 'int_rate', 'installment', 'annual_inc', 'dti', 'open_ac
c',
       'pub_rec', 'revol_bal', 'revol_util', 'total_acc', 'mort_acc',
       'pub_rec_bankruptcies'],
      dtype='object')
```

In [ ]:

```

```

Feature Engineering

Creating a flag :
If a values is greater than 0.0 then 1 else 0

This is to be done on-
* Pub_rec
* Mort_acc
* pub_rec_bankruptcies

In [46]:

```
loan_data.head()
```

Out[46]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	home_ownership	annual_inc	ve
0	10000.0	36 months	11.44	329.48	B	B4	RENT	117000.0	
1	8000.0	36 months	11.99	265.68	B	B5	MORTGAGE	65000.0	
2	15600.0	36 months	10.49	506.97	B	B3	RENT	43057.0	
3	7200.0	36 months	6.49	220.65	A	A2	RENT	54000.0	
4	24375.0	60 months	17.27	609.33	C	C5	MORTGAGE	55000.0	

5 rows × 24 columns

In [47]:

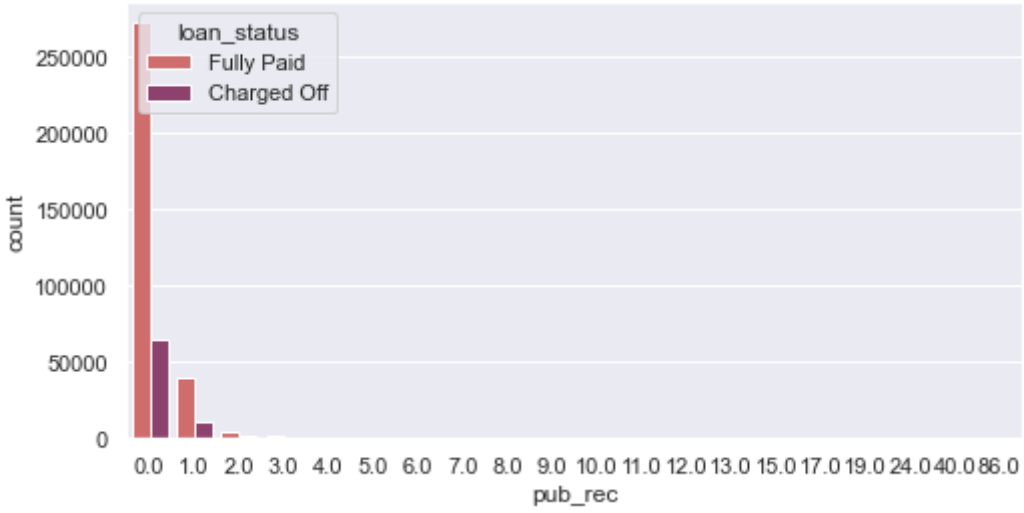
```
#FE on Pub_rec
loan_data['pub_rec'].value_counts()
```

Out[47]:

```
0.0     337489
1.0      49713
2.0       5474
3.0       1521
4.0        527
5.0        237
6.0        122
7.0         56
8.0         34
9.0         12
10.0        11
11.0         8
13.0         4
12.0         4
19.0         2
40.0         1
17.0         1
86.0         1
24.0         1
15.0         1
Name: pub_rec, dtype: int64
```

In [48]:

```
plt.figure(figsize=(8,4))
sns.countplot(x='pub_rec',data=loan_data,hue='loan_status',palette='flare')
plt.show()
```

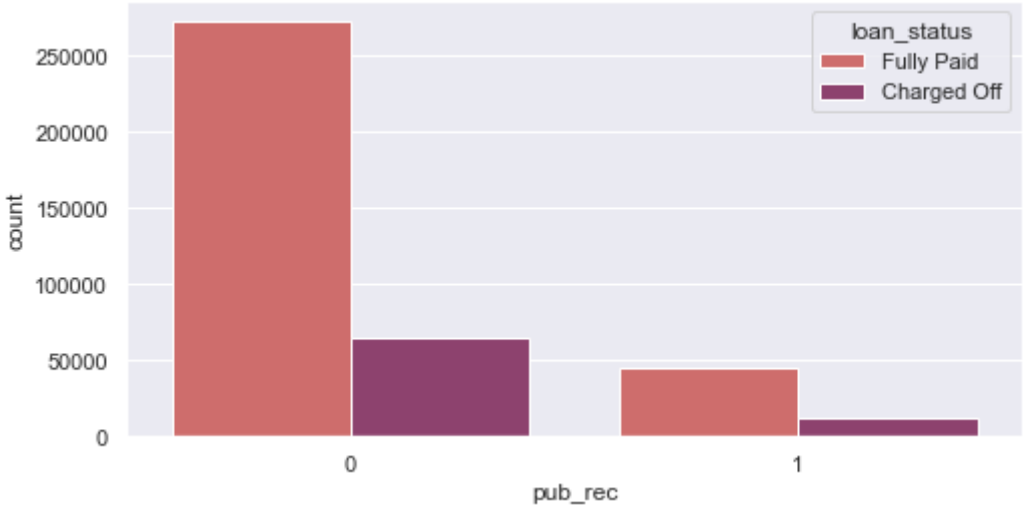


In [49]:

```
loan_data['pub_rec'] = loan_data['pub_rec'].apply(lambda x: 1 if x > 0.0 else 0)
```

In [50]:

```
plt.figure(figsize=(8,4))
sns.countplot(x='pub_rec',data=loan_data,hue='loan_status',palette='flare')
plt.show()
```



In [51]:

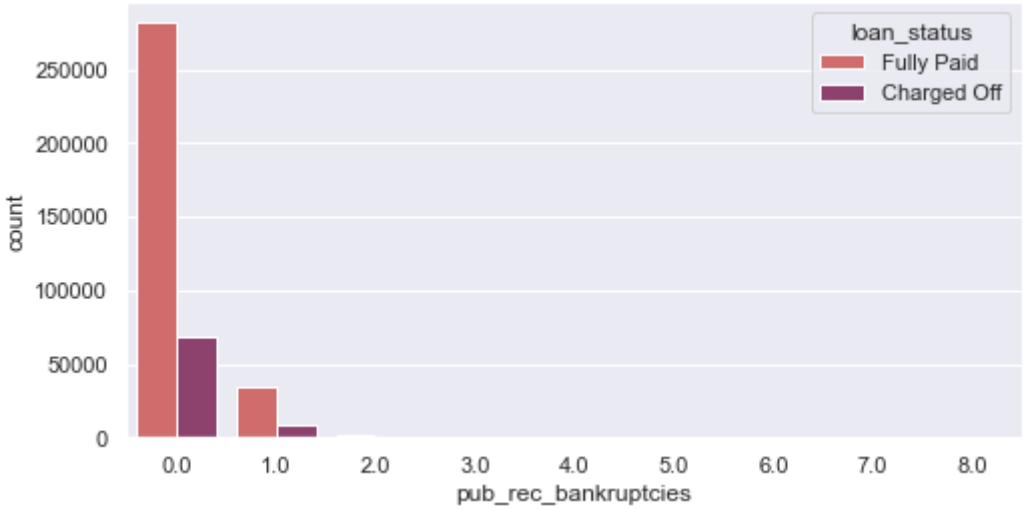
```
# FE on pub_rec_bankruptcies
loan_data['pub_rec_bankruptcies'].value_counts()
```

Out[51]:

```
0.0    350108
1.0     42786
2.0      1847
3.0       351
4.0        82
5.0        32
6.0         7
7.0         4
8.0         2
Name: pub_rec_bankruptcies, dtype: int64
```

In [52]:

```
plt.figure(figsize=(8,4))
sns.countplot(x='pub_rec_bankruptcies',data=loan_data,hue='loan_status',palette='flare')
plt.show()
```

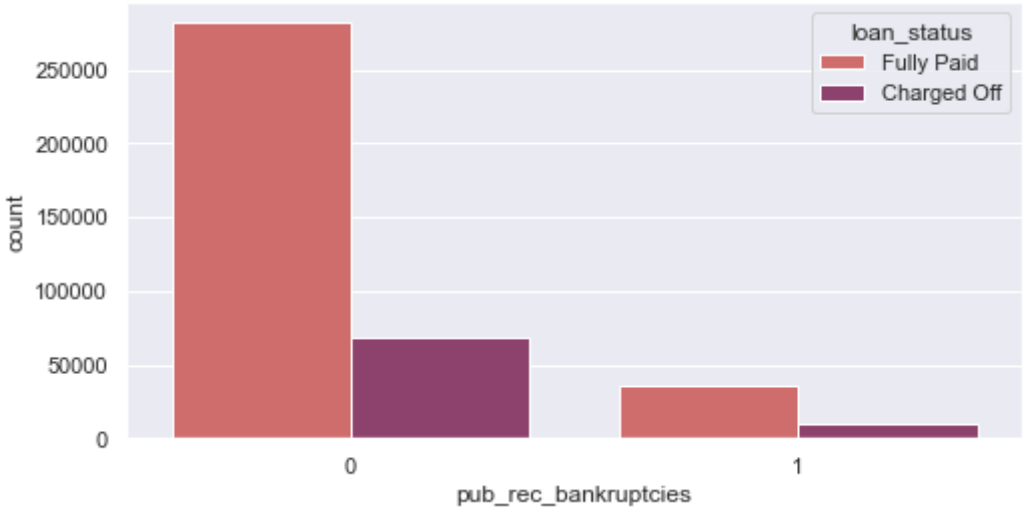


In [53]:

```
loan_data['pub_rec_bankruptcies']=loan_data['pub_rec_bankruptcies'].apply(lambda x: 1 if x
```

In [54]:

```
plt.figure(figsize=(8,4))
sns.countplot(x='pub_rec_bankruptcies',data=loan_data,hue='loan_status',palette='flare')
plt.show()
```



In [55]:

```
# FE on mort_acc will done post median imputation after train-test split.
loan_data['mort_acc'].value_counts()
```

Out[55]:

```
0.0     139708
1.0      60384
2.0      49907
3.0      38022
4.0      27865
5.0      18182
6.0      11059
7.0       6050
8.0       3120
9.0       1653
10.0       863
11.0       479
12.0       264
13.0       146
14.0       107
15.0        61
16.0        37
17.0        22
18.0        18
19.0        15
20.0        13
24.0        10
22.0         7
21.0         4
25.0         4
27.0         3
32.0         2
31.0         2
23.0         2
26.0         2
28.0         1
30.0         1
34.0         1
Name: mort_acc, dtype: int64
```

In [56]:

```
convert_dtype=['pub_rec','pub_rec_bankruptcies']
```

In [57]:

```
for i in convert_dtype:
    loan_data[i] = loan_data[i].astype('object')
print(loan_data[i].dtype)
```

```
object
```

In [58]:

```
# Frequency of each feature in percentage.
def categorical_feat(df, colnames, nrows=2,mcols=2,width=15,height=80, sortbyindex=False):
    fig , ax = plt.subplots(nrows,mcols,figsize=(width,height))
    fig.set_facecolor(color = 'peachpuff')
    string = "Frequency of "
    rows = 0
    for colname in colnames:
        count = (df[colname].value_counts(normalize=True)*100)
        string += colname + ' in (%)'
        if sortbyindex:
                count = count.sort_index()
        count.plot.bar(color=sns.color_palette("flare"),ax=ax[rows][0])
        ax[rows][0].set_ylabel(string, fontsize=14)
        ax[rows][0].set_xlabel(colname, fontsize=14)

        count.plot.pie(colors = sns.color_palette("flare"),autopct='%0.0f%%',
                       textprops={'fontsize': 14},shadow = True, ax=ax[rows][1])
        #explode=[0.2 if colname[i] == min(colname) else 0])
        ax[rows][0].set_title("Frequency wise " + colname, fontweight="bold")
        string = "Frequency of "
        rows += 1
```

In [59]:

```
Cats_cols_all = ['term','grade','sub_grade','home_ownership','verification_status','issue_d
                 'purpose','earliest_cr_line', 'initial_list_status', 'application_type', '
```

In [60]:

```
categorical_cols = ['grade', 'home_ownership', 'verification_status', 'loan_status', 'initi
                    'pub_rec','pub_rec_bankruptcies','term']
```

In [61]:

```
feature_with_large_unique_vals = ['sub_grade','issue_d','earliest_cr_line','address']
```

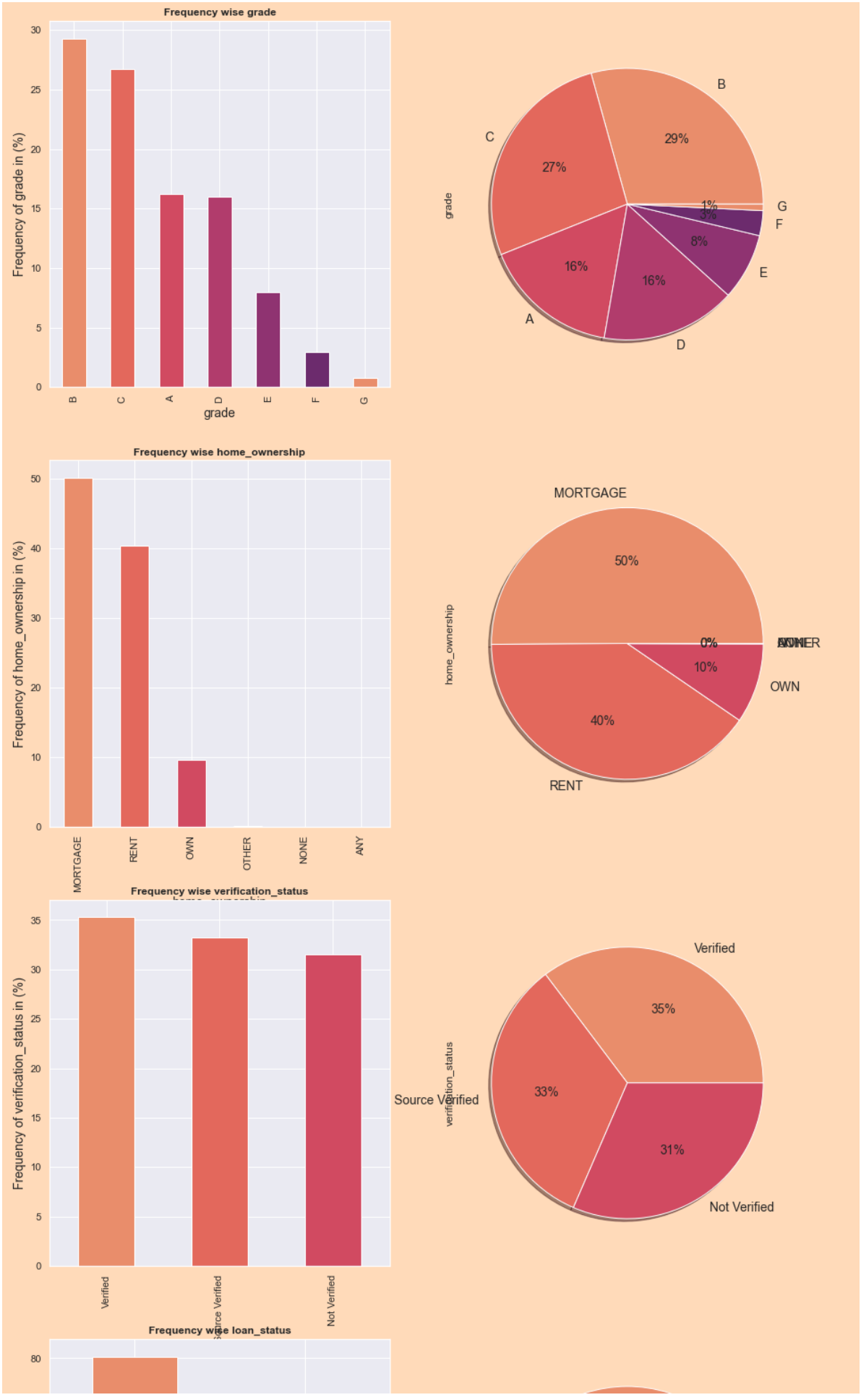
In [62]:

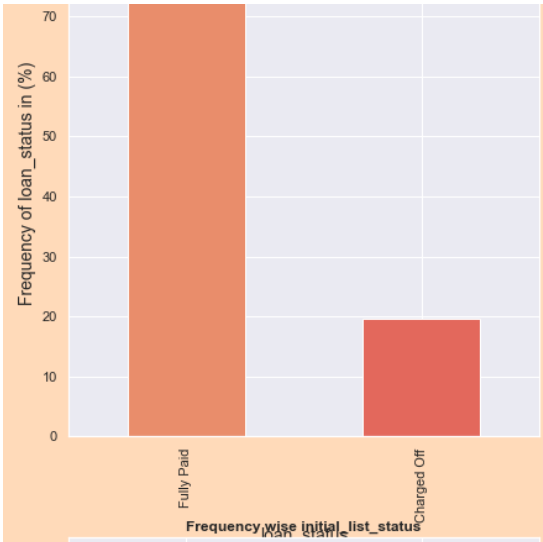
```
for i in Cats_cols_all:
    print(f'Unique values in {i} are {loan_data[i].nunique()}')
```

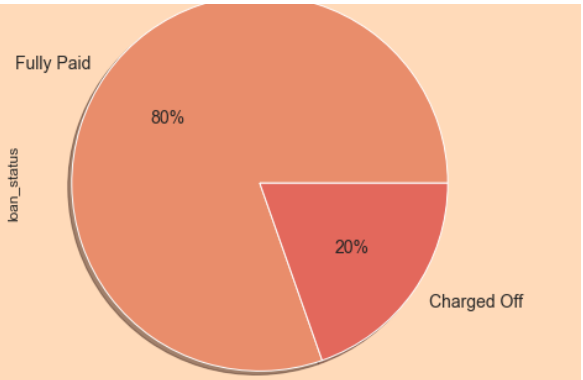
```
Unique values in term are 2
Unique values in grade are 7
Unique values in sub_grade are 35
Unique values in home_ownership are 6
Unique values in verification_status are 3
Unique values in issue_d are 112
Unique values in loan_status are 2
Unique values in purpose are 14
Unique values in earliest_cr_line are 684
Unique values in initial_list_status are 2
Unique values in application_type are 3
Unique values in address are 392898
```

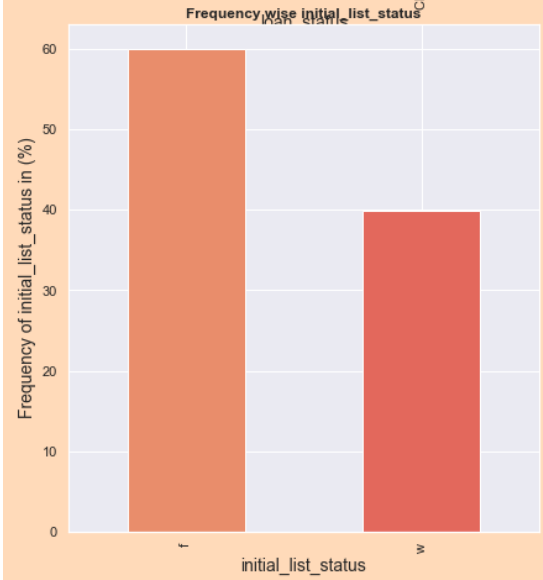
In [63]:

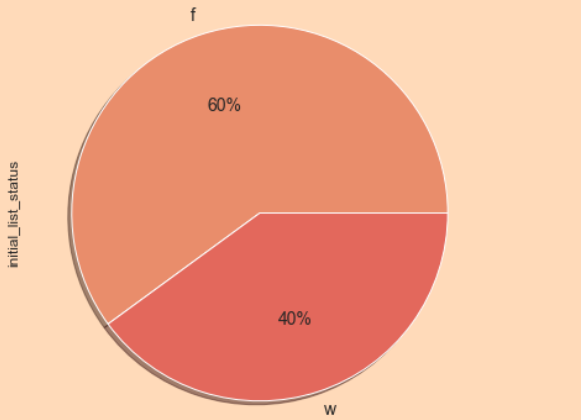
```
categorical_feat(loan_data,categorical_cols,len(categorical_cols),2)
```

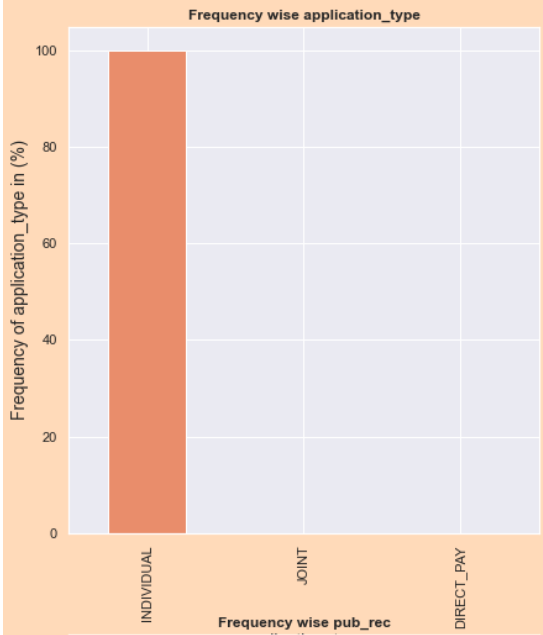


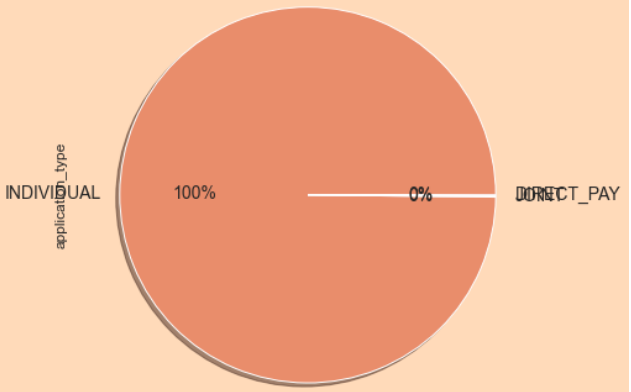

Frequency of loan_status in (%)
Fully Paid
Charged Off

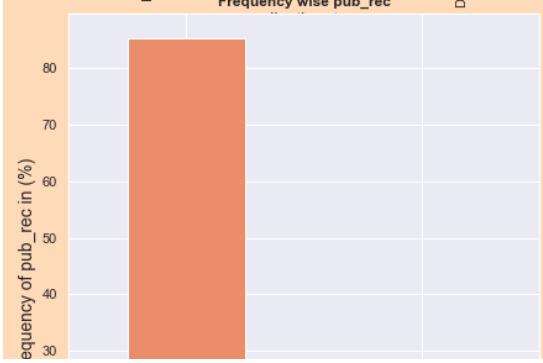

Fully Paid
80%
20%
Charged Off
loan_status

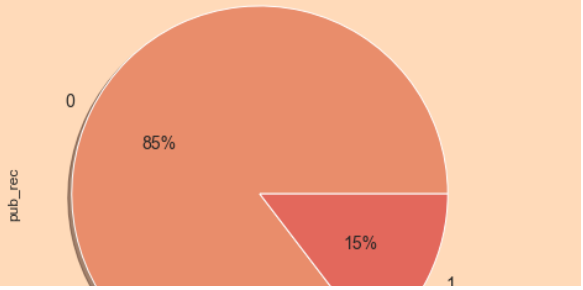

Frequency wise initial_list_status
Frequency of initial_list_status in (%)
f
w
initial_list_status

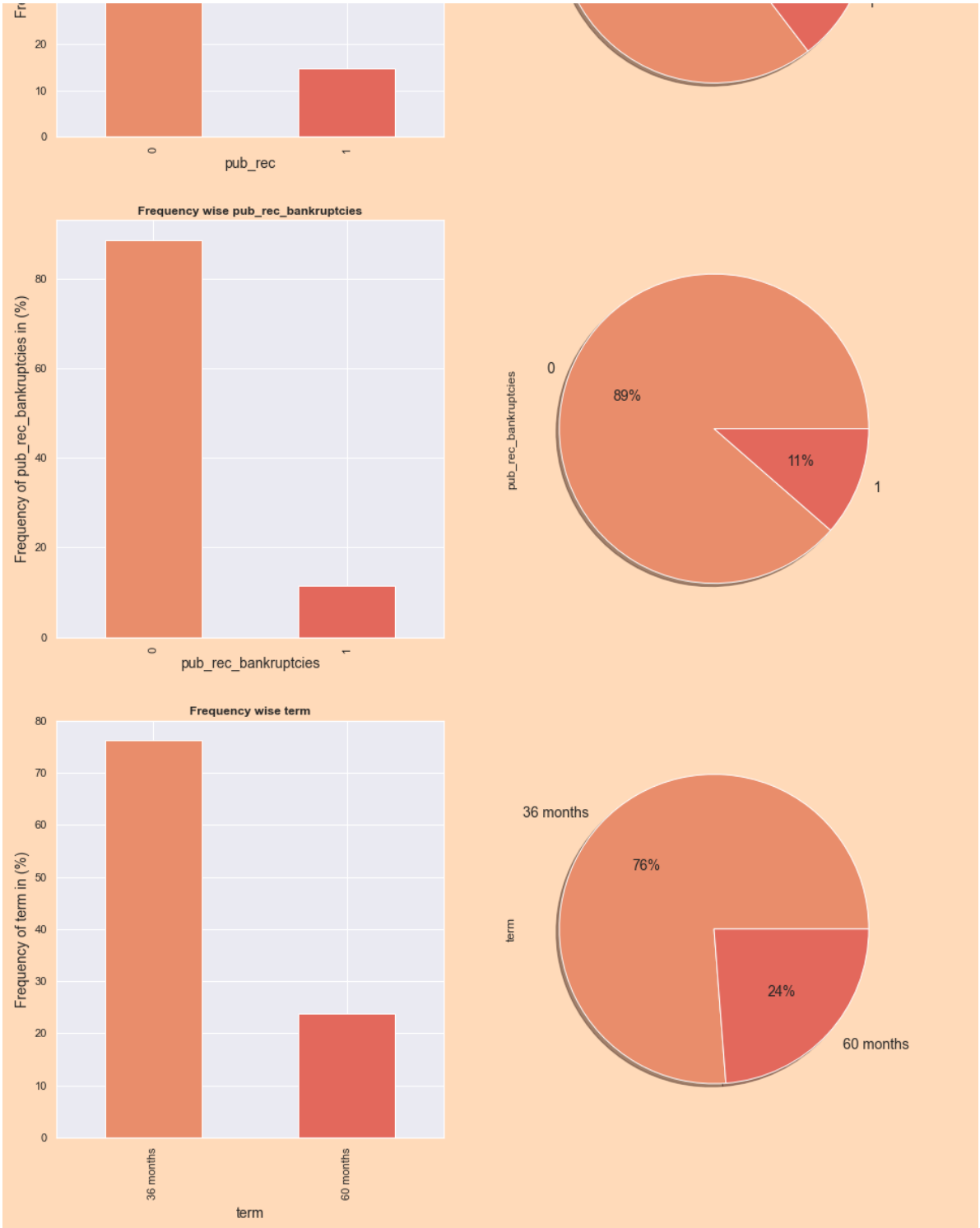

f
60%
40%
w
initial_list_status


Frequency wise application_type
Frequency of application_type in (%)
INDIVIDUAL
JOINT
DIRECT_PAY


INDIVIDUAL
100%
0%
application_type


Frequency wise pub_rec
equency of pub_rec in (%)


0
85%
15%
1
pub_rec



In [64]:

```
for i,j in enumerate(categorical_cols):
    print(i,j)
```

```
0 grade
1 home_ownership
2 verification_status
3 loan_status
4 initial_list_status
5 application_type
6 pub_rec
7 pub_rec_bankruptcies
8 term
```

Checking dependancy of target variable (loan_status):

In [65]:

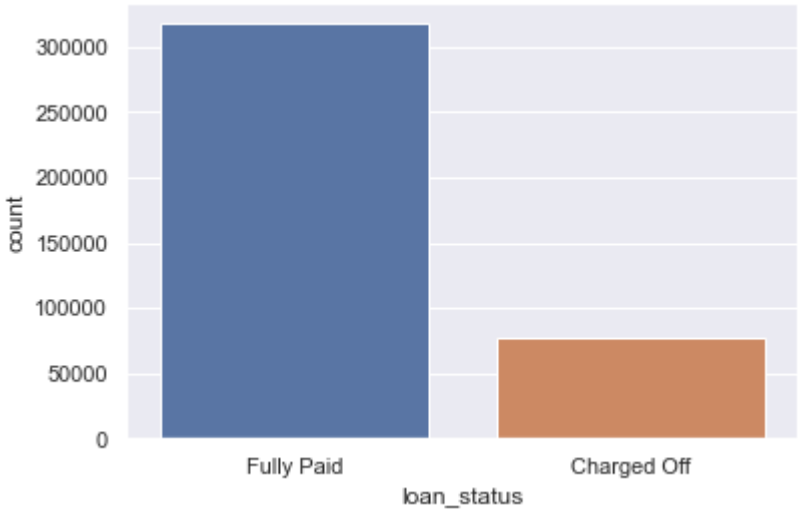
```
# Checking dependency of target variable (Loan_Status) on various predictor variables
vals_counts = loan_data['loan_status'].value_counts(normalize=True) * 100
pd.DataFrame(vals_counts.round())
```

Out[65]:

	loan_status
Fully Paid	80.0
Charged Off	20.0

In [66]:

```
sns.countplot(loan_data['loan_status'])
plt.show()
```



In [67]:

```
categorical_cols = ['term', 'grade', 'home_ownership', 'verification_status', 'loan_status'
                    'application_type','purpose']
```

In [68]:

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

In [69]:

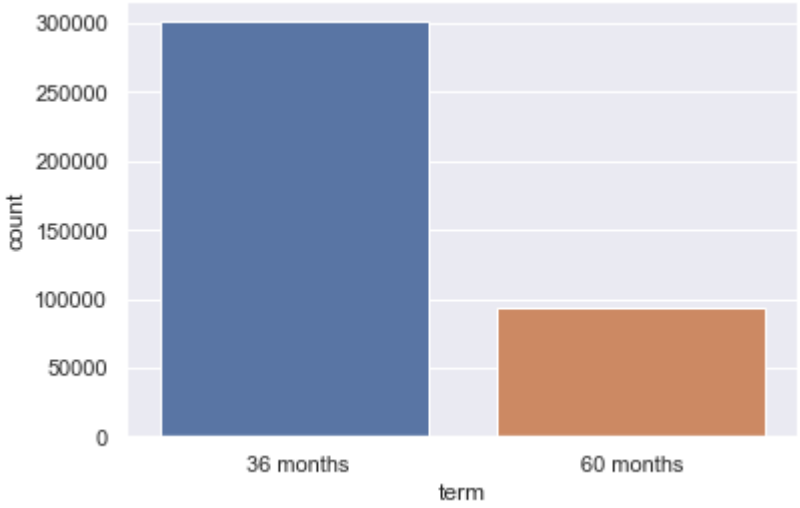
```
val_counts = loan_data['loan_status'].value_counts(normalize=True) * 100
pd.DataFrame(val_counts.round())
```

Out[69]:

	loan_status
0	80.0
1	20.0

In [70]:

```
sns.countplot(loan_data['term'])
plt.show()
```



In [71]:

```
encode_dict = {' 36 months':0, ' 60 months':1}
loan_data['term'] = loan_data['term'].map(encode_dict)
val_counts  = loan_data['term'].value_counts(normalize = True) * 100
pd.DataFrame(val_counts.round())
```

Out[71]:

	term
0	76.0
1	24.0

In [72]:

```
loan_data.columns
```

Out[72]:

```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grade',
       'home_ownership', 'annual_inc', 'verification_status', 'issue_d',
       'loan_status', 'purpose', 'dti', 'earliest_cr_line', 'open_acc',
       'pub_rec', 'revol_bal', 'revol_util', 'total_acc',
       'initial_list_status', 'application_type', 'mort_acc',
       'pub_rec_bankruptcies', 'address'],
      dtype='object')
```

In [73]:

```
loan_data.head(3)
```

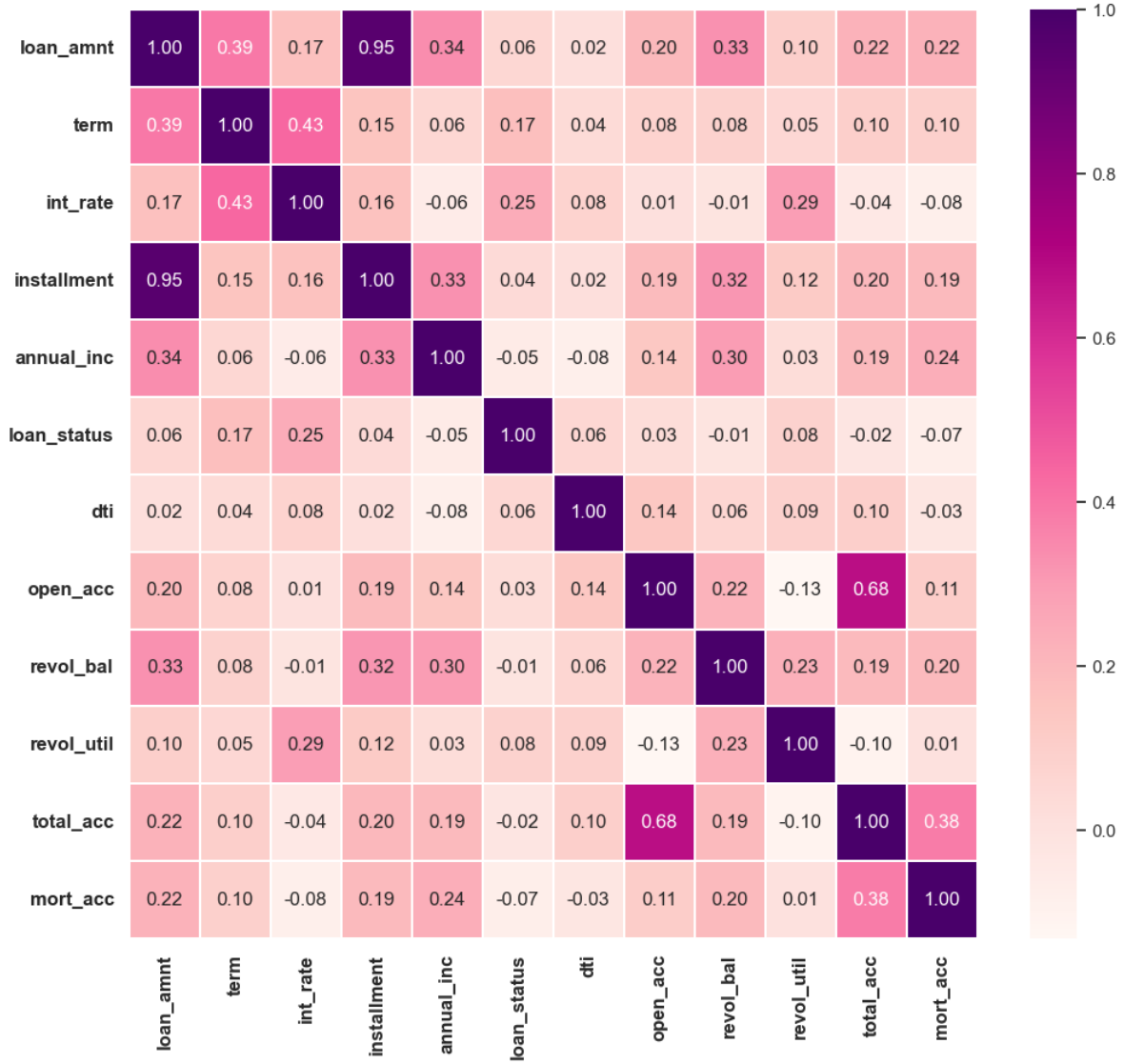
Out[73]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	home_ownership	annual_inc	verif
0	10000.0	0	11.44	329.48	B	B4	RENT	117000.0	
1	8000.0	0	11.99	265.68	B	B5	MORTGAGE	65000.0	
2	15600.0	0	10.49	506.97	B	B3	RENT	43057.0	

3 rows × 24 columns

In [74]:

```
# Checking correlation among independent variables and how they interact with each other.
fig, ax = plt.subplots(figsize=(12, 10),dpi = 100)
fig.subplots_adjust(top=.94)

ax.set_yticklabels(ax.get_yticklabels(), ha="right", fontsize=12, weight='bold')
ax.set_xticklabels(ax.get_xticklabels(), fontsize=12, weight='bold')

sns.heatmap(loan_data.corr(), annot = True, fmt='.2f', linewidths=.3, ax = ax ,cmap='RdPu')
plt.show()
```



Insights:

```
    * loan amount and Installments are highly correlated with each other, more the
number of loan amount more the number
      of installment
    * total_acc and open_acc are highly correlated
    * pub_rec_bankruptcies and pub_rec are higly corelated.
```

EDA w.r.t loan_amnt, loan_installment , annual_inc and loan_status:

In [75]:

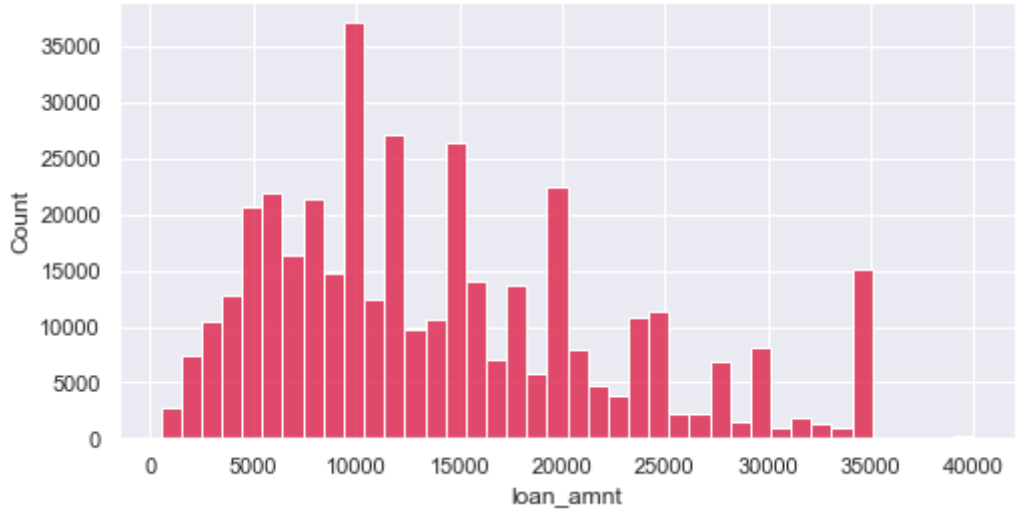
```
loan_data.groupby(['loan_status'])['loan_amnt'].describe()
```

Out[75]:

loan_status	count	mean	std	min	25%	50%	75%	max
0	317696.0	13875.478681	8302.172723	500.0	7500.0	12000.0	19275.0	40000.0
1	77523.0	15132.578074	8503.989092	1000.0	8575.0	14000.0	20000.0	40000.0

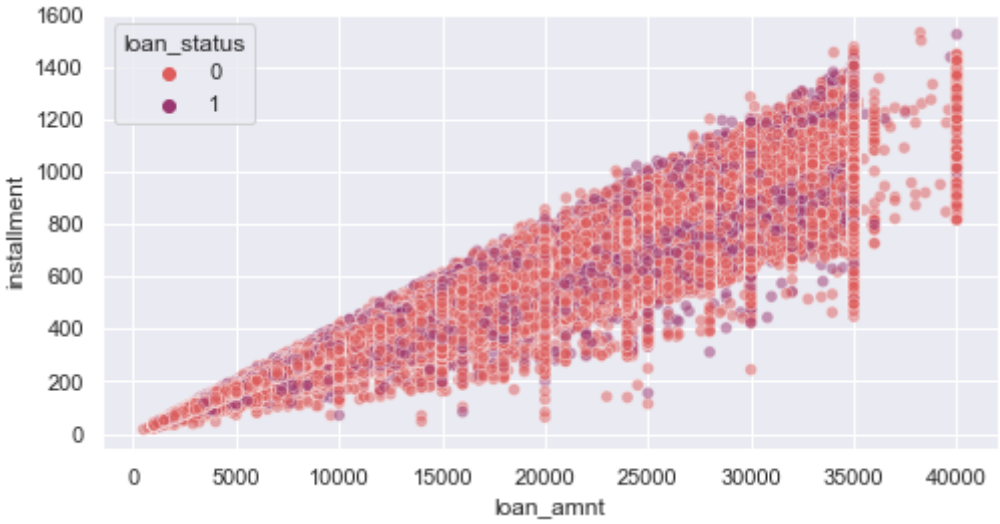
In [76]:

```
plt.figure(figsize=(8,4))
sns.histplot(data=loan_data,x='loan_amnt',bins=40,color='crimson')
plt.show()
```



In [77]:

```
#  relationship between loan amount, installment and loan_status
encode_dist={'Charged Off':1,'Fully Paid':0}
plt.figure(figsize=(8,4))
sns.scatterplot(data=loan_data,x='loan_amnt',y='installment',alpha=0.5,hue='loan_status',pa
plt.show()
```



Insights:

* loan_amnt and installment are highly correlated, more the loan_amnt more the number of installments

In [78]:

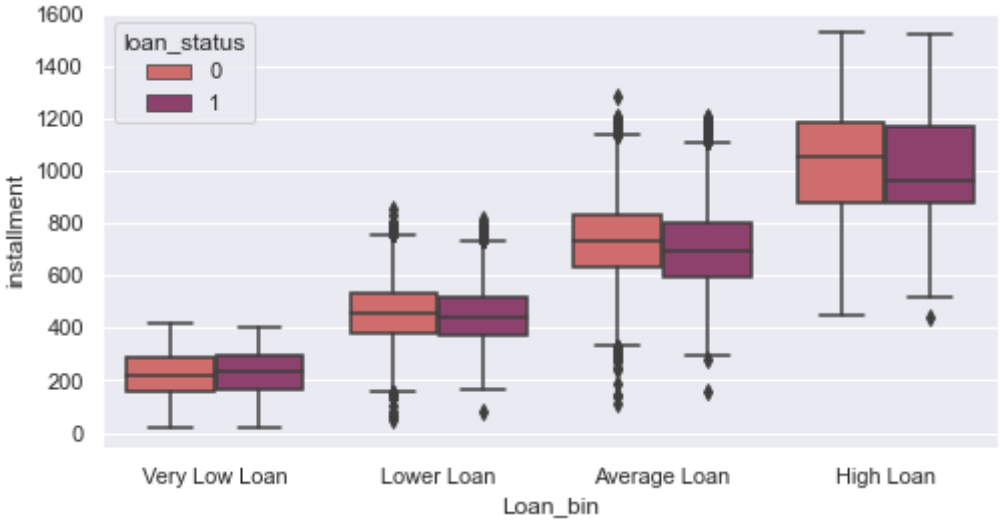
```
bins=[0,10000,20000,30000,40000]
group_name=['Very Low Loan', 'Lower Loan', 'Average Loan', 'High Loan']

loan_data['Loan_bin'] = pd.cut(loan_data['loan_amnt'],bins=bins,labels=group_name)
loan_data['Loan_bin']
```

Out[78]:

```
0         Very Low Loan
1         Very Low Loan
2            Lower Loan
3         Very Low Loan
4          Average Loan
              ...
396025    Very Low Loan
396026     Average Loan
396027    Very Low Loan
396028     Average Loan
396029    Very Low Loan
Name: Loan_bin, Length: 395219, dtype: category
Categories (4, object): ['Very Low Loan' < 'Lower Loan' < 'Average Loan' <
'High Loan']
```

In [79]:

```
#  relationship between loan amount, installment and loan_status
encode_dist={'Charged Off':1,'Fully Paid':0}
plt.figure(figsize=(8,4))
sns.boxplot(data=loan_data,x='Loan_bin',y='installment',hue='loan_status',palette='flare')
plt.show()
```



Insights:

* when the loan taken is less than 20000, then the installment in the form of monthly payment owned by the borrower
originates, are similar to both types of borrower who have defaulted or fully
* But as loan amount increases, the installment owned by the defaulters decreases as compared to the installments owned
by the honest borrowers who are not defaulting

In [80]:

```
# annual_inc
loan_data['annual_inc'].value_counts()
```

Out[80]:

```
60000.00    15288
50000.00    13282
65000.00    11317
70000.00    10662
40000.00    10612
            ...
67842.00        1
72179.00        1
50416.00        1
46820.80        1
31789.88        1
Name: annual_inc, Length: 27155, dtype: int64
```

In [81]:

```
loan_data.groupby(['loan_status'])['annual_inc'].describe()
```

Out[81]:

loan_status	count	mean	std	min	25%	50%	75%	max
0	317696.0	75827.575913	62215.381438	600.0	46171.25	65000.0	90000.0	7600000.0
1	77523.0	67526.852852	58310.628403	0.0	42000.00	59000.0	80000.0	8706582.0

In [82]:

```
bins=[0,25000,50000,100000,500000,1200000,2500000,8706582]
group_name=['BPL','Very Low','Low','Average','Better Off','High','Very High']

loan_data['Income_bin']=pd.cut(loan_data['annual_inc'],bins=bins,labels=group_name)
loan_data[['Income_bin']]
```

Out[82]:

	Income_bin
0	Average
1	Low
2	Very Low
3	Low
4	Low
...	...
396025	Very Low
396026	Average
396027	Low
396028	Low
396029	Very Low

395219 rows × 1 columns

In [83]:

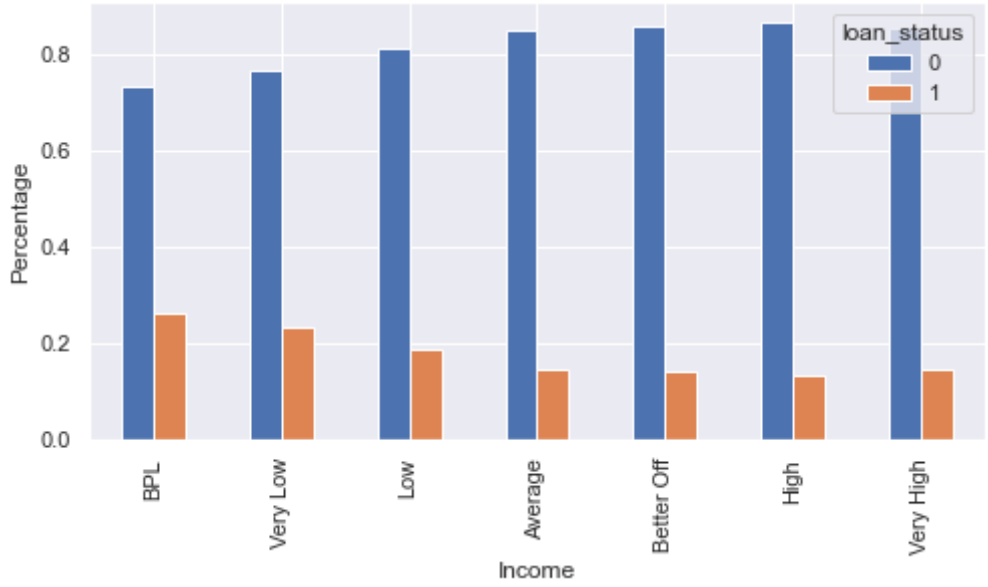
```
Income_bin=pd.crosstab(loan_data['Income_bin'],loan_data['loan_status'])
Income_bin
```

Out[83]:

loan_status	0	1
Income_bin		
BPL	10742	3841
Very Low	89294	26960
Low	159430	36729
Average	57884	9936
Better Off	308	50
High	26	4
Very High	12	2

In [84]:

```
Income_bin.div(Income_bin.sum(axis=1),axis=0).plot(kind='bar',figsize=(8,4))
plt.xlabel('Income')
plt.ylabel('Percentage')
plt.show()
```

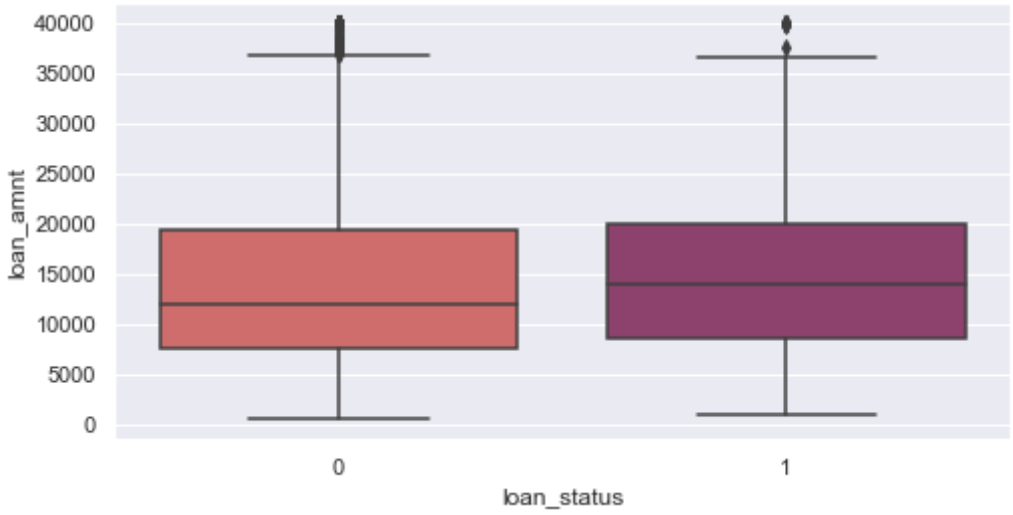


Insights:

* There's a high percentage of defaulting (Charged_off) when the income is below poverty line, very low and low
category. Whereas there are very few cases where customers with high and very high income also have defaulted the
loan.

In [85]:

```
# The relationship between the loan_status and the Loan Amount

encode_doct={'Charged Off':0,'Fully Paid':1}
plt.figure(figsize=(8,4))
sns.boxplot(data=loan_data,x='loan_status',y='loan_amnt',palette='flare')
plt.show()
```



In [86]:

```
# summary statistics for the loan amount, grouped by the loan_status.

loan_data.groupby('loan_status')['loan_amnt'].describe().round()
```

Out[86]:

loan_status	count	mean	std	min	25%	50%	75%	max
0	317696.0	13875.0	8302.0	500.0	7500.0	12000.0	19275.0	40000.0
1	77523.0	15133.0	8504.0	1000.0	8575.0	14000.0	20000.0	40000.0

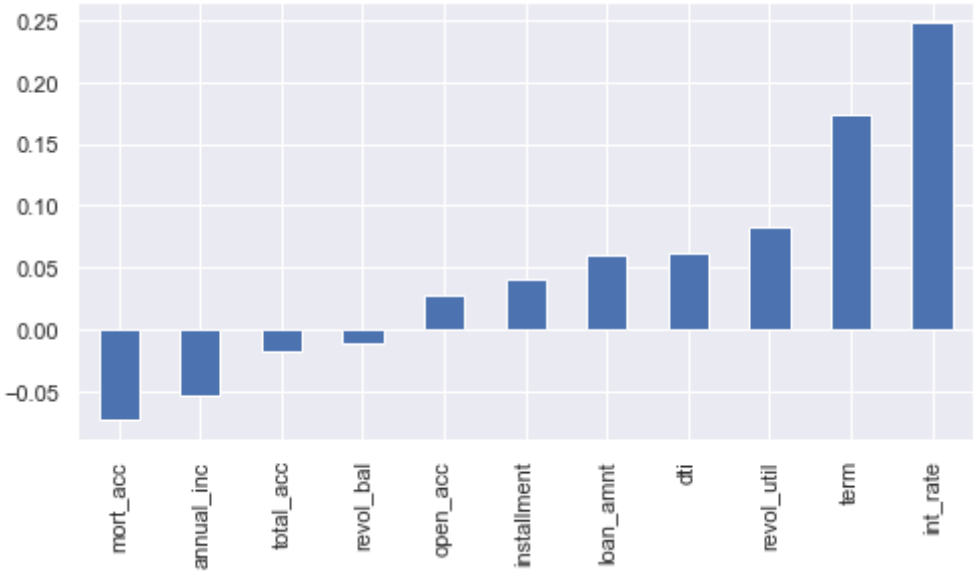
Insights:

* If the loan amount is higher then we have a slight increase in the likelihood of being charged off. This makes
sense it is being harder to repay a larger loans than smaller loans

EDA w.r.t loan_status and int_rate:

In [87]:

```
# Correlation of the numeric features to the loan_status column
plt.figure(figsize=(8,4))
loan_data.corr()['loan_status'].sort_values().drop('loan_status').plot(kind='bar')
plt.show()
```



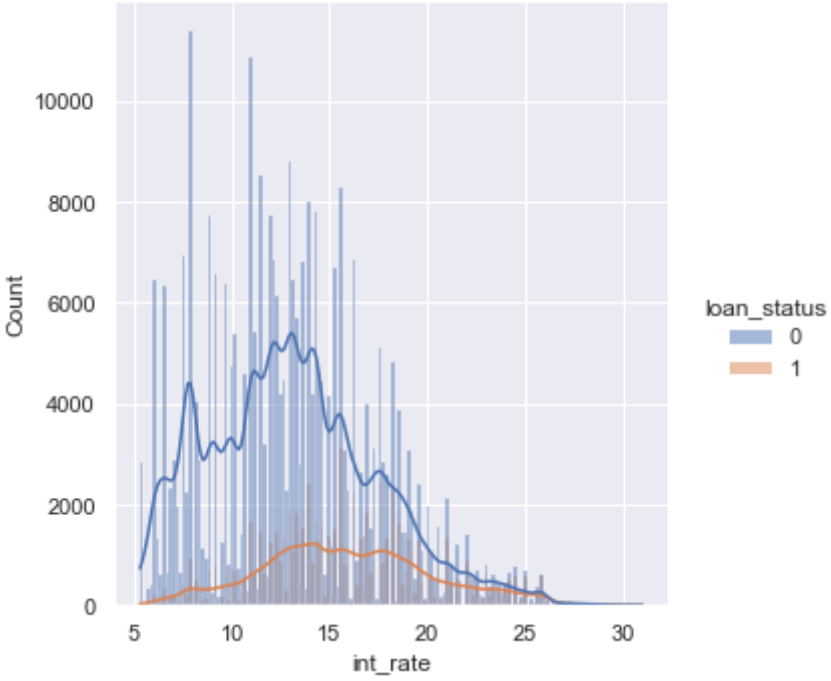
Insights:

```
    * As the interest rate goes down, its easier to pay loan whereas if the intere
  st rate increased there's more
      possibility of charged off
```

In [88]:

```
plt.figure(figsize=(8,4))
sns.displot(data=loan_data,x='int_rate',hue='loan_status',kde=True)
plt.show()
```

```
<Figure size 576x288 with 0 Axes>
```



In [89]:

```
feature_names()
```

```
Columns with category datatypes (Categorical Features) are :     ['grade',
'sub_grade', 'home_ownership', 'verification_status', 'issue_d', 'purpose',
'earliest_cr_line', 'pub_rec', 'initial_list_status', 'application_type', 'p
ub_rec_bankruptcies', 'address']
------------------------------------------------------------------------------
------------------------
------------------------------------------------------------------------------
------------------------
Columns with integer and float datatypes (Numerical Features) are:     ['loa
n_amnt', 'term', 'int_rate', 'installment', 'annual_inc', 'loan_status', 'dt
i', 'open_acc', 'revol_bal', 'revol_util', 'total_acc', 'mort_acc']
```

EDA w.r.t home_ownership:

In [90]:

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

Out[90]:

```
MORTGAGE    198022
RENT        159395
OWN          37660
OTHER          110
NONE            29
ANY              3
Name: home_ownership, dtype: int64
```

Insights:

* Majority of the people have home loan as MORTGAGE

The relationship between Grades and loan_status:

In [91]:

```
# What are the unique possible grades and subgrades?
sorted(loan_data['grade'].unique())
```

Out[91]:

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

In [92]:

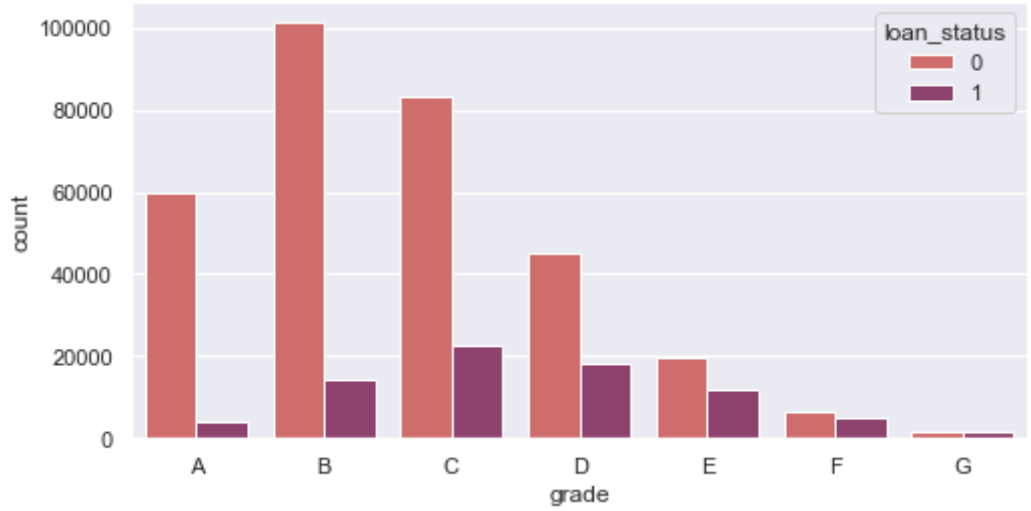
```
loan_data['sub_grade'].unique()
```

Out[92]:

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

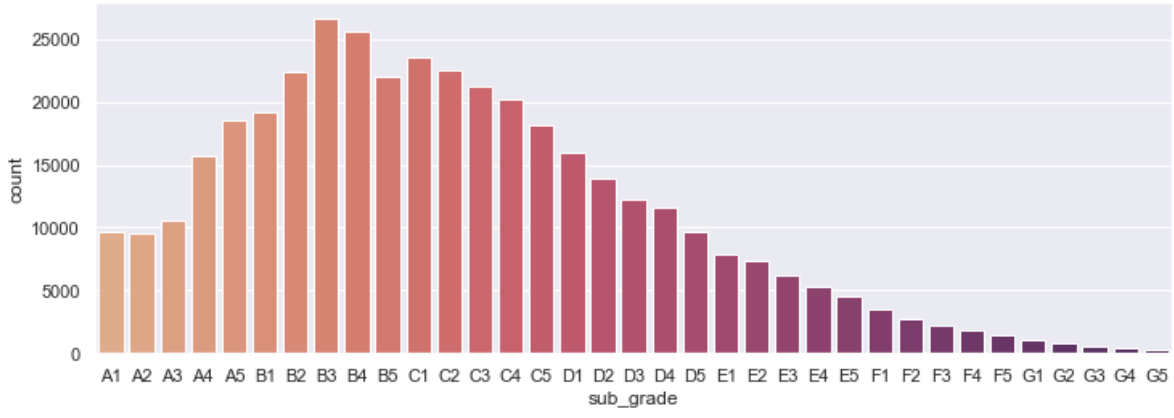
In [93]:

```
# Create a countplot per grade. Set the hue to the loan_status label.
order = sorted(loan_data['grade'].unique())
plt.figure(figsize=(8,4))
sns.countplot(x='grade',data=loan_data,hue='loan_status',order=order,palette='flare')
plt.show()
```



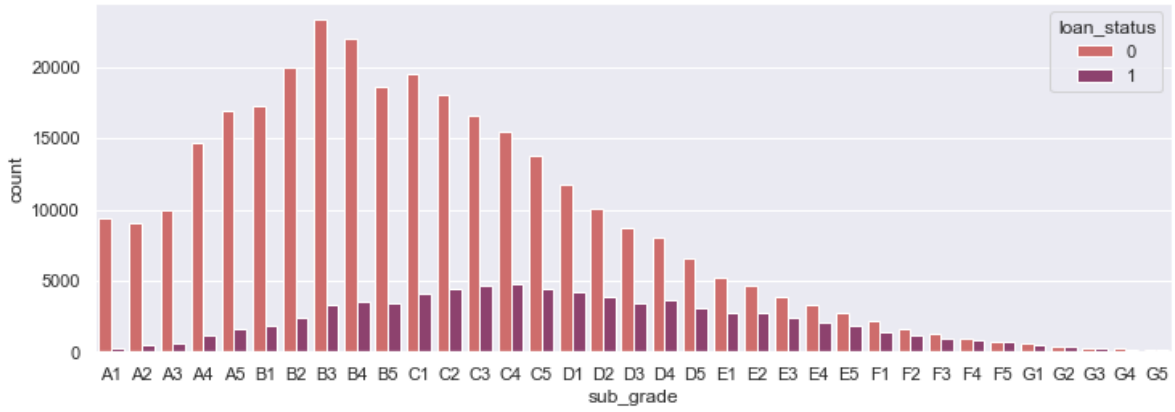
In [94]:

```
#Display a count plot per subgrade
plt.figure(figsize=(12,4))
order=sorted(loan_data['sub_grade'].unique())
sns.countplot(x='sub_grade',data=loan_data,order=order,palette='flare')
plt.show()
```



In [95]:

```
# Create a countplot per sub_grade. Set the hue to the loan_status label.
order = sorted(loan_data['sub_grade'].unique())
plt.figure(figsize = (12,4))
sns.countplot(x='sub_grade',data=loan_data,hue='loan_status', order = order, palette='flare
plt.show()
```



Insights:

* It is obvious that F and G grades have high default rate. We will need to have a closer look at them.

In [96]:

```
# more analysis
filtered_f_g = loan_data[(loan_data.grade == 'F') | (loan_data.grade == 'G')]
order = sorted(filtered_f_g['sub_grade'].unique())
plt.figure(figsize = (8,4))
sns.countplot(x='sub_grade', data=filtered_f_g, hue='loan_status', order = order, palette='
plt.show()
```

Insights:

* If your sub grade is G5: the probability that you will default is almost 50%

Categorical and Dummy Variables:

In [97]:

```
loan_data.select_dtypes(['object']).columns
```

Out[97]:

```
Index(['grade', 'sub_grade', 'home_ownership', 'verification_status',
       'issue_d', 'purpose', 'earliest_cr_line', 'pub_rec',
       'initial_list_status', 'application_type', 'pub_rec_bankruptcies',
       'address'],
      dtype='object')
```

Insights:

* We already know grade is part of sub_grade, so just dropping the grade featu
re

In [98]:

```
loan_data = loan_data.drop('grade', axis = 1)
```

In [99]:

```
# Convert the subgrade into dummy variables.

subgrade_dummies = pd.get_dummies(loan_data['sub_grade'],drop_first=True)
loan_data = pd.concat([loan_data.drop('sub_grade',axis=1),subgrade_dummies],axis=1)
```

In [100]:

```
loan_data.columns
```

Out[100]:

```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'home_ownership',
       'annual_inc', 'verification_status', 'issue_d', 'loan_status',
       'purpose', 'dti', 'earliest_cr_line', 'open_acc', 'pub_rec',
       'revol_bal', 'revol_util', 'total_acc', 'initial_list_status',
       'application_type', 'mort_acc', 'pub_rec_bankruptcies', 'address',
       'Loan_bin', 'Income_bin', 'A2', 'A3', 'A4', 'A5', 'B1', 'B2', 'B3',
       'B4', 'B5', 'C1', 'C2', 'C3', 'C4', 'C5', 'D1', 'D2', 'D3', 'D4', 'D
5',
       'E1', 'E2', 'E3', 'E4', 'E5', 'F1', 'F2', 'F3', 'F4', 'F5', 'G1', 'G
2',
       'G3', 'G4', 'G5'],
      dtype='object')
```

In [101]:

```
loan_data.select_dtypes(['object']).columns
```

Out[101]:

```
Index(['home_ownership', 'verification_status', 'issue_d', 'purpose',
       'earliest_cr_line', 'pub_rec', 'initial_list_status',
       'application_type', 'pub_rec_bankruptcies', 'address'],
      dtype='object')
```

In [102]:

```
dummies = pd.get_dummies(loan_data[['verification_status', 'application_type','initial_list
loan_data = loan_data.drop(['verification_status', 'application_type','initial_list_status'
loan_data = pd.concat([loan_data,dummies],axis=1)
```

In [103]:

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

Out[103]:

```
MORTGAGE    198022
RENT        159395
OWN          37660
OTHER          110
NONE            29
ANY              3
Name: home_ownership, dtype: int64
```

In [104]:

```
# Convert these to dummy variables, but replace NONE and ANY with OTHER,
# so that we end up with just 4 categories, MORTGAGE, RENT, OWN, OTHER

loan_data['home_ownership']=loan_data['home_ownership'].replace(['NONE', 'ANY'], 'OTHER')
```

In [105]:

```
home_ownership_dummies = pd.get_dummies(loan_data['home_ownership'],drop_first=True)
loan_data = loan_data.drop('home_ownership',axis=1)
loan_data = pd.concat([loan_data,home_ownership_dummies],axis=1)
```

In [106]:

```
loan_data['address'][0]
```

Out[106]:

```
'0174 Michelle Gateway\r\nMendozaberg, OK 22690'
```

In [107]:

```
loan_data['address'].value_counts()[0]
```

Out[107]:

```
8
```

In [108]:

```
# Let's feature engineer a zip code column from the address in the data set

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

In [109]:

```
loan_data['zip_code'].value_counts()
```

Out[109]:

```
70466    56880
22690    56413
30723    56402
48052    55811
00813    45725
29597    45393
05113    45300
11650    11210
93700    11126
86630    10959
Name: zip_code, dtype: int64
```

In [110]:

```
zip_code_dummies = pd.get_dummies(loan_data['zip_code'],drop_first=True)
```

In [111]:

```
loan_data = loan_data.drop(['zip_code','address'],axis=1)
```

In [112]:

```
loan_data = pd.concat([loan_data,zip_code_dummies],axis=1)
```

In [113]:

```
# issue_d

loan_data['issue_d'].value_counts()
```

Out[113]:

```
Oct-2014    14838
Jul-2014    12597
Jan-2015    11701
Dec-2013    10609
Nov-2013    10492
            ...
Jan-2008        3
Dec-2007        2
Oct-2007        1
Nov-2007        1
Aug-2007        1
Name: issue_d, Length: 112, dtype: int64
```

In [114]:

```
# Need to drop issue_d feature

loan_data = loan_data.drop('issue_d',axis=1)
```

Insights:

```
    * This would be data leakage as we wouldn't know beforehand whether or not a l
oan would be issued when using our
      model, so in reality we wouldn't have an issue_date and hence droped this fe
ature.
```

In [115]:

```
# FE on earliest_cr_line

loan_data['earliest_cr_line'].value_counts()
```

Out[115]:

```
Oct-2000    3013
Aug-2000    2930
Oct-2001    2890
Aug-2001    2879
Nov-2000    2729
            ...
Jul-1955       1
Oct-1950       1
May-1958       1
Nov-1955       1
Aug-1959       1
Name: earliest_cr_line, Length: 684, dtype: int64
```

Insights:

* This appears to be a historical time stamp feature. Extracting the year from this feature using apply function,
then converting it to a numeric feature.

In [116]:

```
loan_data['earliest_cr_year'] = loan_data['earliest_cr_line'].apply(lambda date:int(date[-4
loan_data = loan_data.drop('earliest_cr_line',axis=1)
```

In [117]:

```
loan_data['earliest_cr_year'].value_counts()
```

Out[117]:

```
2000    29302
2001    29031
1999    26444
2002    25849
2003    23623
        ...
1951        3
1950        3
1953        2
1944        1
1948        1
Name: earliest_cr_year, Length: 65, dtype: int64
```

In [118]:

```
loan_data.select_dtypes(['object']).columns
```

Out[118]:

```
Index(['pub_rec', 'pub_rec_bankruptcies'], dtype='object')
```

In [119]:

```
for i in convert_dtype:
    loan_data[i] =  loan_data[i].astype('int64')
print(loan_data[i].dtype)
```

```
int64
```

In [120]:

```
# Convert the Loan_bin and Income_bin into dummy variables.

loan_data['Loan_bin'].value_counts()
```

Out[120]:

```
Very Low Loan    163818
Lower Loan       150715
Average Loan      60192
High Loan         20494
Name: Loan_bin, dtype: int64
```

In [121]:

```
dummies = pd.get_dummies(loan_data['Loan_bin'],drop_first=True)
```

In [122]:

```
loan_data = loan_data.drop('Loan_bin',axis=1)
```

In [123]:

```
loan_data = pd.concat([loan_data,dummies],axis=1)
```

In [124]:

```
loan_data['Income_bin'].value_counts()
```

Out[124]:

```
Low           196159
Very Low      116254
Average        67820
BPL            14583
Better Off       358
High              30
Very High         14
Name: Income_bin, dtype: int64
```

In [125]:

```
# Convert these to dummy variables, but replace Better off , High and Very High with OTHER,
# so that we end up with just 4 categories, MORTGAGE, RENT, OWN, OTHER

loan_data['Income_bin']=loan_data['Income_bin'].replace([ 'High', 'Very High'], 'OTHER')
```

In [126]:

```
dummies = pd.get_dummies(loan_data['Income_bin'],drop_first=True)
loan_data = loan_data.drop('Income_bin',axis=1)
loan_data = pd.concat([loan_data,dummies],axis=1)
```

In [127]:

```
loan_data.select_dtypes(['object']).columns
```

Out[127]:

```
Index([], dtype='object')
```

In [128]:

```
loan_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 395219 entries, 0 to 396029
Data columns (total 87 columns):
 #   Column                                  Non-Null Count   Dtype  
---  ------                                  --------------   -----  
 0   loan_amnt                               395219 non-null  float64
 1   term                                    395219 non-null  int64  
 2   int_rate                                395219 non-null  float64
 3   installment                             395219 non-null  float64
 4   annual_inc                              395219 non-null  float64
 5   loan_status                             395219 non-null  int64  
 6   dti                                     395219 non-null  float64
 7   open_acc                                395219 non-null  float64
 8   pub_rec                                 395219 non-null  int64  
 9   revol_bal                               395219 non-null  float64
 10  revol_util                              395219 non-null  float64
 11  total_acc                               395219 non-null  float64
 12  mort_acc                                358014 non-null  float64
 13  pub_rec_bankruptcies                    395219 non-null  int64  
 14  A2                                      395219 non-null  uint8  
 15  A3                                      395219 non-null  uint8  
 16  A4                                      395219 non-null  uint8  
 17  A5                                      395219 non-null  uint8  
 18  B1                                      395219 non-null  uint8  
 19  B2                                      395219 non-null  uint8  
 20  B3                                      395219 non-null  uint8  
 21  B4                                      395219 non-null  uint8  
 22  B5                                      395219 non-null  uint8  
 23  C1                                      395219 non-null  uint8  
 24  C2                                      395219 non-null  uint8  
 25  C3                                      395219 non-null  uint8  
 26  C4                                      395219 non-null  uint8  
 27  C5                                      395219 non-null  uint8  
 28  D1                                      395219 non-null  uint8  
 29  D2                                      395219 non-null  uint8  
 30  D3                                      395219 non-null  uint8  
 31  D4                                      395219 non-null  uint8  
 32  D5                                      395219 non-null  uint8  
 33  E1                                      395219 non-null  uint8  
 34  E2                                      395219 non-null  uint8  
 35  E3                                      395219 non-null  uint8  
 36  E4                                      395219 non-null  uint8  
 37  E5                                      395219 non-null  uint8  
 38  F1                                      395219 non-null  uint8  
 39  F2                                      395219 non-null  uint8  
 40  F3                                      395219 non-null  uint8  
 41  F4                                      395219 non-null  uint8  
 42  F5                                      395219 non-null  uint8  
 43  G1                                      395219 non-null  uint8  
 44  G2                                      395219 non-null  uint8  
 45  G3                                      395219 non-null  uint8  
 46  G4                                      395219 non-null  uint8  
 47  G5                                      395219 non-null  uint8  
 48  verification_status_Source Verified     395219 non-null  uint8  
 49  verification_status_Verified            395219 non-null  uint8  
 50  application_type_INDIVIDUAL             395219 non-null  uint8  
 51  application_type_JOINT                  395219 non-null  uint8  
```

```
 52  initial_list_status_w                395219 non-null  uint8
 53  purpose_credit_card                  395219 non-null  uint8
 54  purpose_debt_consolidation           395219 non-null  uint8
 55  purpose_educational                  395219 non-null  uint8
 56  purpose_home_improvement             395219 non-null  uint8
 57  purpose_house                        395219 non-null  uint8
 58  purpose_major_purchase               395219 non-null  uint8
 59  purpose_medical                      395219 non-null  uint8
 60  purpose_moving                       395219 non-null  uint8
 61  purpose_other                        395219 non-null  uint8
 62  purpose_renewable_energy             395219 non-null  uint8
 63  purpose_small_business               395219 non-null  uint8
 64  purpose_vacation                     395219 non-null  uint8
 65  purpose_wedding                      395219 non-null  uint8
 66  OTHER                                395219 non-null  uint8
 67  OWN                                  395219 non-null  uint8
 68  RENT                                 395219 non-null  uint8
 69  05113                                395219 non-null  uint8
 70  11650                                395219 non-null  uint8
 71  22690                                395219 non-null  uint8
 72  29597                                395219 non-null  uint8
 73  30723                                395219 non-null  uint8
 74  48052                                395219 non-null  uint8
 75  70466                                395219 non-null  uint8
 76  86630                                395219 non-null  uint8
 77  93700                                395219 non-null  uint8
 78  earliest_cr_year                     395219 non-null  int64
 79  Lower Loan                           395219 non-null  uint8
 80  Average Loan                         395219 non-null  uint8
 81  High Loan                            395219 non-null  uint8
 82  Very Low                             395219 non-null  uint8
 83  Low                                  395219 non-null  uint8
 84  Average                              395219 non-null  uint8
 85  Better Off                           395219 non-null  uint8
 86  OTHER                                395219 non-null  uint8
dtypes: float64(10), int64(5), uint8(72)
memory usage: 83.4 MB
```

In [129]:

```
loan_data.shape
```

Out[129]:

```
(395219, 87)
```

In [130]:

```
#Creating a copy of the data

loan = loan_data.copy()
```

Insights:

* We have converted all the categorical data into one hot endcoded dummy variables.
* We have now 3,95,219 rows and 86 columns in our data.
* We are reading for model building and training on these numerical features.

* open_acc : The number of open credit lines in the borrower's credit file.
* total_acc : The total number of credit lines currently in the borrower's credit file

Logistics Regression Model Building

In [131]:

```
# Assigning the featurs as X and target as y

X= loan.drop(["loan_status"],axis =1)
y= loan["loan_status"]
```

In [132]:

```
X.shape
```

Out[132]:

```
(395219, 86)
```

Splitting data into train , validation and test:

In [133]:

```
# Train, CV, test split
from sklearn.model_selection import train_test_split
#0.6, 0.2, 0.2 split

X_tr_cv, X_test, y_tr_cv, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

X_train, X_val, y_train, y_val = train_test_split(X_tr_cv, y_tr_cv, test_size=0.2, random_s
```

In [134]:

```
X_train.shape
```

Out[134]:

```
(252940, 86)
```

Missing value and Outlier treatment on Train Data:

* At this point , we will do the missing value handling using median imputation for missing value for 'mort_acc'
* At this point , we will do the outliers handling/treatment using median imputati

```
    on for missing value for all the
      numerical features.
```

In [135]:

```
missingValue(X_train)
```

```
Total records in our data =  252940 where missing values are as follows:
```

Out[135]:

	Total Missing	In Percent
mort_acc	23889	9.44
OTHER	0	0.00
purpose_vacation	0	0.00
purpose_small_business	0	0.00
purpose_renewable_energy	0	0.00
...	...	...
D2	0	0.00
D1	0	0.00
C5	0	0.00
C4	0	0.00
OTHER	0	0.00

86 rows × 2 columns

In [136]:

```
X_train['mort_acc'].value_counts()
```

Out[136]:

```
0.0     89391
1.0     38512
2.0     31881
3.0     24442
4.0     17865
5.0     11556
6.0      7123
7.0      3880
8.0      2016
9.0      1067
10.0      548
11.0      315
12.0      164
13.0       95
14.0       70
15.0       42
16.0       19
19.0       12
18.0       12
20.0        9
17.0        8
24.0        7
22.0        4
25.0        3
32.0        2
26.0        2
27.0        2
31.0        1
34.0        1
30.0        1
21.0        1
Name: mort_acc, dtype: int64
```

In [137]:

```
X_train['mort_acc'].median()
```

Out[137]:

```
1.0
```

In [138]:

```
#Imputing mort_acc with median

X_train['mort_acc'] = X_train['mort_acc'].fillna(X_train['mort_acc'].median())
```

In [139]:

```
X_train['mort_acc'].isnull().sum()
```

Out[139]:

```
0
```

In [140]:

```
missingValue(X_train)
```

```
Total records in our data =  252940 where missing values are as follows:
```

Out[140]:

	Total Missing	In Percent
loan_amnt	0	0.0
purpose_educational	0	0.0
purpose_small_business	0	0.0
purpose_renewable_energy	0	0.0
purpose_other	0	0.0
...	...	...
D1	0	0.0
C5	0	0.0
C4	0	0.0
C3	0	0.0
OTHER	0	0.0

86 rows × 2 columns

In [141]:

```
# Feature Engineering on mort_acc on train data

X_train['mort_acc'] = X_train['mort_acc'].apply(lambda x: 1 if x > 0.0 else 0)
```

In [142]:

```
X_train_copy1 = X_train.copy()
```

In [143]:

```
# Outlier Treatment on numerical cols on train data

numerical_cols = ['loan_amnt', 'int_rate', 'installment', 'annual_inc', 'open_acc', 'total_
```

In [144]:

```
q1 = X_train_copy1[numerical_cols].quantile(0.25)
q3 = X_train_copy1[numerical_cols].quantile(0.75)
iqr = q3 -q1

X_train_copy1 = X_train_copy1[~((X_train_copy1[numerical_cols]<q1-1.5*iqr) | (X_train_copy1
X_train_copy1 = X_train_copy1.reset_index(drop = True)
```

In [145]:

```
X_train.shape[0]
```

Out[145]:

```
252940
```

In [146]:

```
X_train.shape[0] - X_train_copy1.shape[0]
```

Out[146]:

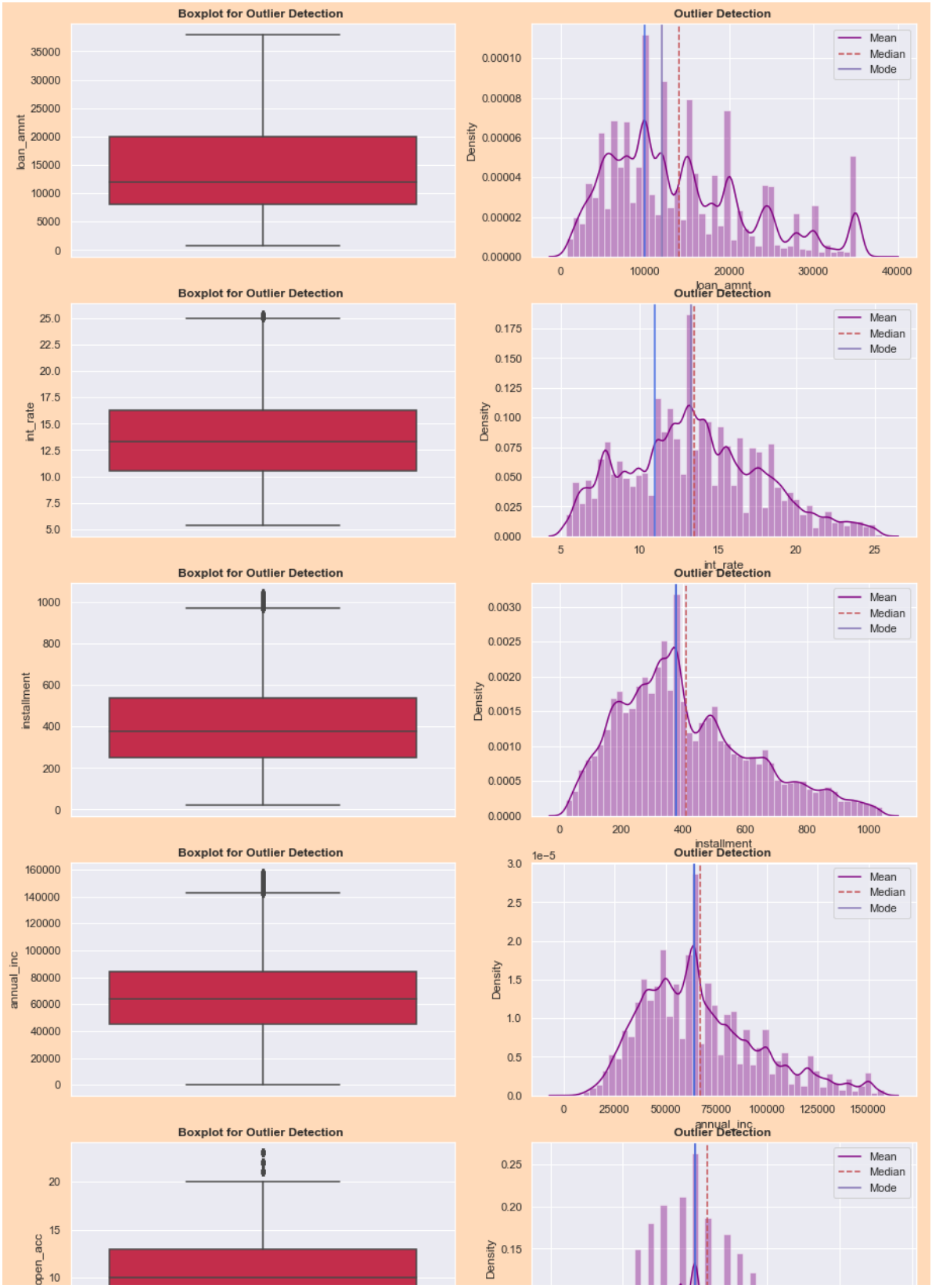
```
35396
```

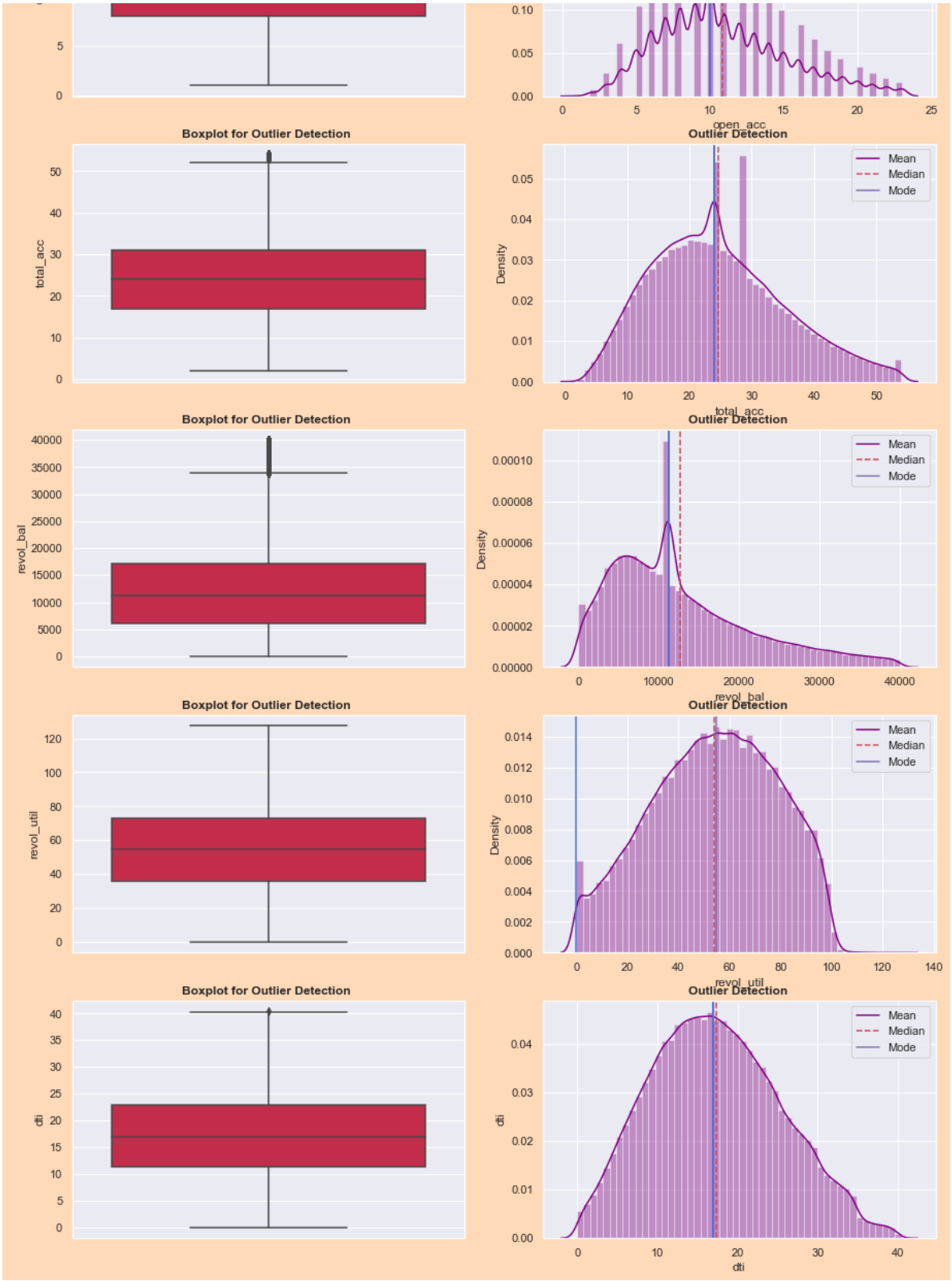
Insights:

* We can treat 52,465 values using iqr based median imputation

In [147]:

```
for i in numerical_cols:
    X_train[i] = np.where(X_train[i] > X_train[i].quantile(0.75) + \
                          (X_train[i].quantile(0.75) - \
                           X_train[i].quantile(0.25))*1.5,
                          X_train[i].quantile(0.5),X_train[i])
    X_train[i] = np.where(X_train[i] < X_train[i].quantile(0.25) - \
                          (X_train[i].quantile(0.75) - \
                           X_train[i].quantile(0.25))*1.5, \
                          X_train[i].quantile(0.5),X_train[i])
numerical_feat(X_train,numerical_cols,len(numerical_cols),2,15,45)
```





Insights:

```
    * As we can see from the above plots, the train data is free from outliers fro
m all the important and required
      numerical features
```

Missing value and Outlier treatment on Cross Validation Data:

In [148]:

```
#Imputing mort_acc with median on cross val data

X_val['mort_acc'] = X_val['mort_acc'].fillna(X_val['mort_acc'].median())
missingValue(X_val)
```

```
Total records in our data =  63235 where missing values are as follows:
```

Out[148]:

	Total Missing	In Percent
loan_amnt	0	0.0
purpose_educational	0	0.0
purpose_small_business	0	0.0
purpose_renewable_energy	0	0.0
purpose_other	0	0.0
...	...	...
D1	0	0.0
C5	0	0.0
C4	0	0.0
C3	0	0.0
OTHER	0	0.0

86 rows × 2 columns

In [149]:

```
# Feature Engineering on mort_acc on cross validation data
X_val_copy1 = X_val.copy()
X_val['mort_acc'] = X_val['mort_acc'].apply(lambda x: 1 if x > 0.0 else 0)
```

Missing value and Outlier treatment on Test Data:

In [150]:

```
#Imputing mort_acc with median on test data

X_test['mort_acc'] = X_test['mort_acc'].fillna(X_test['mort_acc'].median())
missingValue(X_test)
```

```
Total records in our data =  79044 where missing values are as follows:
```

Out[150]:

	Total Missing	In Percent
loan_amnt	0	0.0
purpose_educational	0	0.0
purpose_small_business	0	0.0
purpose_renewable_energy	0	0.0
purpose_other	0	0.0
...	...	...
D1	0	0.0
C5	0	0.0
C4	0	0.0
C3	0	0.0
OTHER	0	0.0

86 rows × 2 columns

In [151]:

```
# Feature Engineering on mort_acc on test data

X_test['mort_acc'] = X_test['mort_acc'].apply(lambda x: 1 if x > 0.0 else 0)
```

In [152]:

```
#scaling the data

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
```

Out[152]:

```
StandardScaler()
```

In [153]:

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train, y_train)
```

Out[153]:

```
LogisticRegression()
```

In [154]:

```
model.coef_
```

Out[154]:

```
array([[ 3.61004127e-05,  3.92709716e-03,  5.07707585e-02,
        -2.71745498e-04, -8.08868219e-06,  4.69113833e-02,
         1.23864384e-02,  6.49652493e-04, -2.04208030e-05,
         6.37223293e-03,  1.05790787e-03, -1.88993543e-03,
         3.54674126e-04, -3.94502941e-04, -4.25669959e-04,
        -6.08268205e-04, -6.56676803e-04, -6.23445208e-04,
        -7.23102676e-04, -7.43349096e-04, -5.42657811e-04,
        -3.47715796e-04, -1.98866989e-04, -1.89663820e-05,
         1.36345370e-04,  2.67388376e-04,  2.69820230e-04,
         3.56287695e-04,  3.95151051e-04,  3.49316916e-04,
         4.51085623e-04,  4.00122523e-04,  3.89803844e-04,
         4.38789808e-04,  3.66060422e-04,  3.51388869e-04,
         3.34187509e-04,  2.21528025e-04,  2.10468838e-04,
         1.76034191e-04,  1.52934565e-04,  1.52683832e-04,
         9.71598702e-05,  7.19755306e-05,  6.75845876e-05,
         3.26910222e-05,  3.12168835e-05,  1.02640924e-03,
         7.32607247e-04, -3.87966266e-06, -2.17152836e-05,
         4.69232489e-04, -1.07147942e-03,  4.36258785e-04,
         1.38127610e-06, -4.84974001e-05,  5.92467471e-06,
        -1.70416582e-06,  9.97738516e-05,  7.28751310e-05,
         2.89185246e-04,  8.16104859e-06,  2.66091666e-04,
         2.80520480e-05, -3.74745421e-05, -2.87815725e-07,
         1.05799222e-04,  1.68379296e-03, -3.72513479e-03,
         3.79995006e-03, -7.75214815e-05, -3.73395045e-03,
         1.62259337e-05,  9.67323919e-05, -2.30188231e-05,
         3.65862751e-03,  3.76613270e-03, -1.56637489e-03,
         7.39100253e-04,  1.07525880e-04, -2.75962036e-04,
         5.22329848e-04, -1.63321525e-04, -4.75095366e-04,
        -2.26515817e-05, -1.68783351e-06]])
```

In [155]:

```
model.intercept_
```

Out[155]:

```
array([-8.05080572e-06])
```

Insights:

* The model intercept is very close to zero , which means , our model has very less noise.

Model 1

In [156]:

```
# Hyper-pram tuning
train_scores = []
val_scores = []
scaler = StandardScaler()
# imputer = SimpleImputer(strategy='median', missing_values=np.nan)

for la in np.arange(0.01, 100.0, 5):
    scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la))
    scaled_lr.fit(X_train, y_train)
    train_score = scaled_lr.score(X_train, y_train)
    val_score = scaled_lr.score(X_val, y_val)
    train_scores.append(train_score)
    val_scores.append(val_score)
```

In [157]:

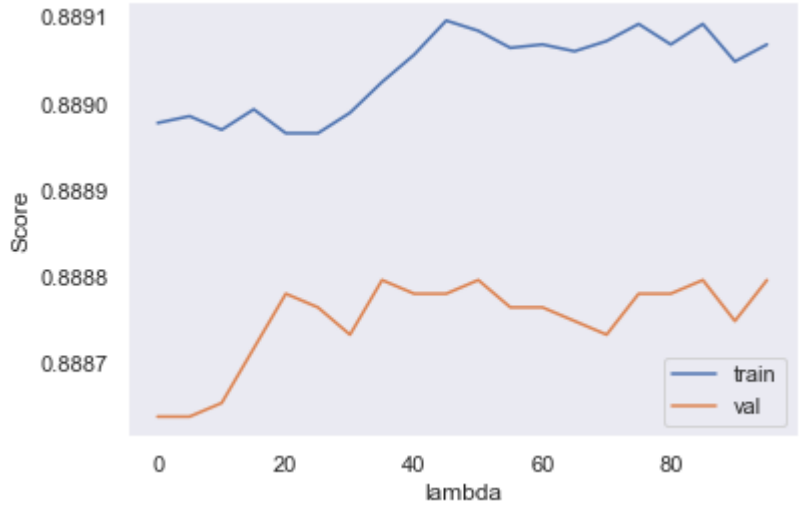
```
len(val_scores)
```

Out[157]:

```
20
```

In [158]:

```
plt.figure()
plt.plot(list(np.arange(0.01, 100.0, 5)), train_scores, label="train")
plt.plot(list(np.arange(0.01, 100.0, 5)), val_scores, label="val")
plt.legend(loc='lower right')
plt.xlabel("lambda")
plt.ylabel("Score")
plt.grid()
plt.show()
```



In [159]:

```
np.argmax(val_scores)
```

Out[159]:

```
7
```

In [160]:

```
val_scores[7]
```

Out[160]:

```
0.8887957618407527
```

In [161]:

```
l_best = 0.01+5*7
l_best
```

Out[161]:

```
35.01
```

In [162]:

```
# Model with lambda= 35.01
scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/l_best))
scaled_lr.fit(X_train, y_train)
```

Out[162]:

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('logisticregression',
                 LogisticRegression(C=0.028563267637817767))])
```

In [163]:

```
test_score = scaled_lr.score(X_test, y_test)
print(test_score)

y_pred = scaled_lr.predict(X_test)
```

```
0.8895804868174687
```

In [164]:

```
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score,confusion_

print(f"Accuracy : {accuracy_score(y_test, y_pred)*100}%")
print(f"recall_score : {recall_score(y_test, y_pred)*100}%")
print(f"precision_score : {precision_score(y_test, y_pred)*100}%")
print(f"f1_score : {f1_score(y_test, y_pred)*100}%")
print(f"AUC score : {metrics.roc_auc_score( y_test, y_pred)*100}%")
print(f"confusion_matrix :")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy : 88.95804868174687%
recall_score : 45.44452370144608%
precision_score : 95.69848422777551%
f1_score : 61.62504396763981%
AUC score : 72.474709864806%
confusion_matrix :
[[63308   315]
 [ 8413  7008]]
```

In [165]:

```
# Let's evaluate the other metrics as well
confusion = confusion_matrix(y_test, y_pred)
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

In [166]:

```
# Calculate the sensitivity

TP/(TP+FN)
```

Out[166]:

```
0.4544452370144608
```

In [167]:

```
# Calculate the sensitivity

TP/(TP+FN)
# Calculate the specificity

TN/(TN+FP)
from sklearn.metrics import classification_report

print(f"{classification_report(y_test, y_pred)}")
```

```
              precision    recall  f1-score   support

           0       0.88      1.00      0.94     63623
           1       0.96      0.45      0.62     15421

    accuracy                           0.89     79044
   macro avg       0.92      0.72      0.78     79044
weighted avg       0.90      0.89      0.87     79044
```

In [168]:

```
from sklearn.metrics import classification_report

print(f"{classification_report(y_test, y_pred)}")
```

```
              precision    recall  f1-score   support

           0       0.88      1.00      0.94     63623
           1       0.96      0.45      0.62     15421

    accuracy                           0.89     79044
   macro avg       0.92      0.72      0.78     79044
weighted avg       0.90      0.89      0.87     79044
```

In [169]:

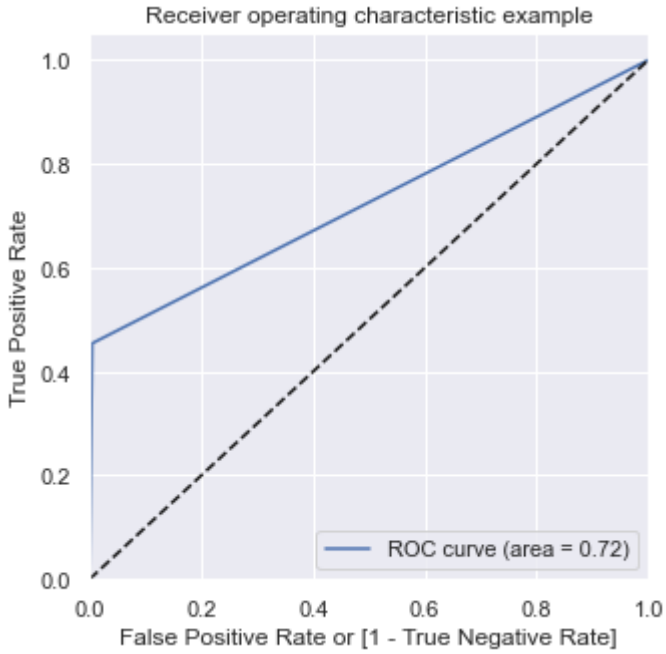
```
# ROC function

from sklearn import metrics
def draw_roc( actual, probs ):
    fpr, tpr, thresholds = metrics.roc_curve( actual, probs,
                                              drop_intermediate = False )
    auc_score = metrics.roc_auc_score( actual, probs )
    plt.figure(figsize=(5, 5))
    plt.plot( fpr, tpr, label='ROC curve (area = %0.2f)' % auc_score )
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()

    return None
```

In [170]:

```
fpr, tpr, thresholds = metrics.roc_curve( y_test, y_pred, drop_intermediate = False )
```

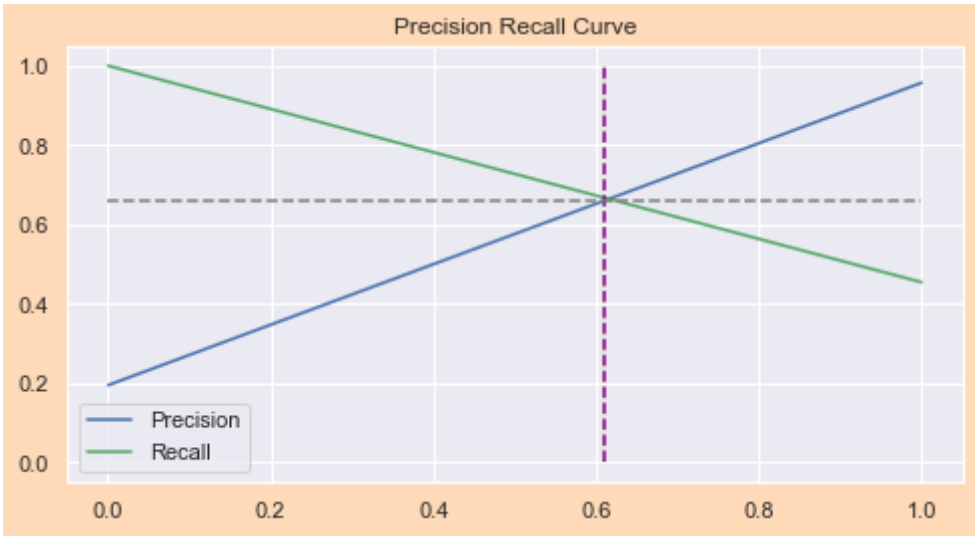
In [171]:

```
# Calling the ROC function

draw_roc(y_test, y_pred)
```



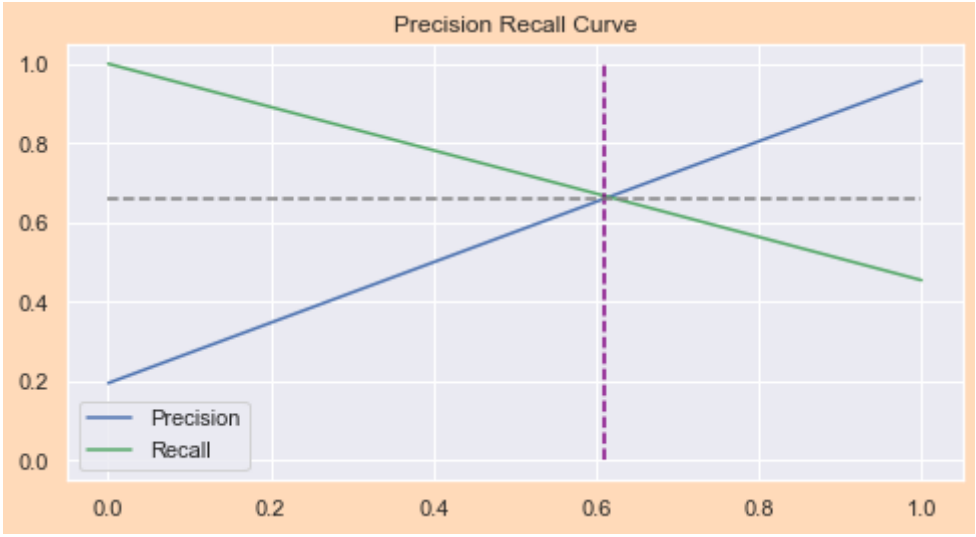
In [172]:

```
# Checking for precision_recall_curve metric

from sklearn.metrics import precision_recall_curve

p, r, thresholds = precision_recall_curve(y_test, y_pred)

fig = plt.figure(figsize = (8,4))
fig.set_facecolor("peachpuff")

# Precision Recall Curve
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)
plt.plot(thresholds, precision[:-1], "b",label='Precision')
plt.plot(thresholds, recall[:-1], "g",label='Recall')
plt.vlines(x=0.61,ymax=1,ymin=0.0,color="purple",linestyles="--")
plt.hlines(y=0.66,xmax=1,xmin=0.0,color="grey",linestyles="--")
plt.title('Precision Recall Curve')
plt.legend()
plt.show()
```



In [173]:

```
fig = plt.figure(figsize = (8,4))
fig.set_facecolor("peachpuff")

# Precision Recall Curve
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)
plt.plot(thresholds, precision[:-1], "b",label='Precision')
plt.plot(thresholds, recall[:-1], "g",label='Recall')
plt.vlines(x=0.61,ymax=1,ymin=0.0,color="purple",linestyles="--")
plt.hlines(y=0.66,xmax=1,xmin=0.0,color="grey",linestyles="--")
plt.title('Precision Recall Curve')
plt.legend()
plt.show()
```



Insgihts From Model 1:

```
    * Defaulters/Irresponsible Business or Customers: For our simplest model 1 , w
e are getting fi-score as 62 % to
      check if the credit line can be extended to the business/customers who have
 a higher chances of defaulting/
      not repaying the loan/credit taken. Since NPA (non-performing asset) is a re
al problem in this industry, it’s
      important we play safe and shouldn’t disburse loans to anyone.
    * Honest/Responsible Business or Customers :For our simplest model 1, we are g
etting fi-score as 94 % to check if
      the credit line can be extended to the business/customers who have a higher
 chances of fully repaying the
      loan/credit taken. This is important as we can lose out on an opportunity to
finance more supply chains and
      earn interest in it.
```

After the preprocessing and encoding steps, we had a total of 87 features and not all of these may be useful in forecasting the sales. Alternatively we can select the top 5 or top 7 features, which had a major contribution in forecasting sales values.

The idea is to have a less complex model without compromising on the overall model performance.

Model 2

Fixing the imbalance issue in target feature:

* Here we will give class weights manually for 0 (Fully paid):0.2, 1 (Charged off) :0.8
* We will also change the range of lambda to see any effect.

In [174]:

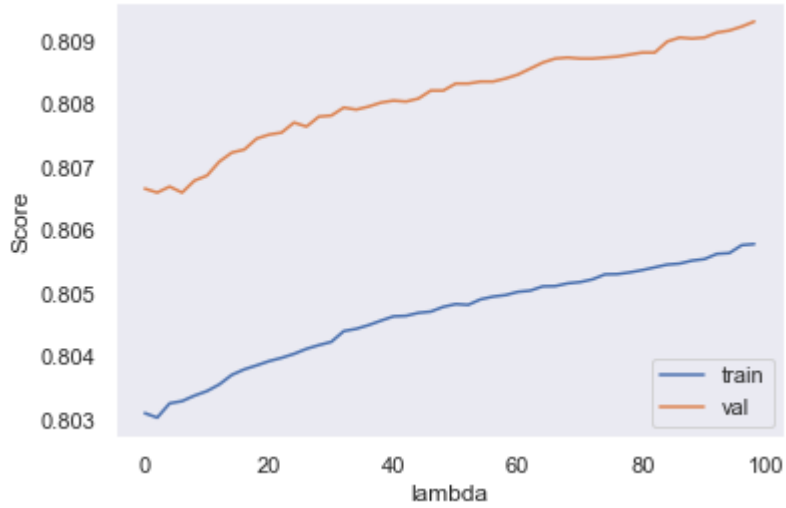
```
# Checking for imbalanced data:

loan['loan_status'].value_counts(normalize = True)*100
```

Out[174]:

```
0    80.384799
1    19.615201
Name: loan_status, dtype: float64
```

In [175]:

```
# Hyper-parameter tuning
train_scores = []
val_scores = []
scaler = StandardScaler()

for la in np.arange(0.01, 100.0, 2):
    scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la , class_weight={ 0:0.2, 1:
    scaled_lr.fit(X_train, y_train)
    train_score = scaled_lr.score(X_train, y_train)
    val_score = scaled_lr.score(X_val, y_val)
    train_scores.append(train_score)
    val_scores.append(val_score)

plt.figure()
plt.plot(list(np.arange(0.01, 100.0, 2)), train_scores, label="train")
plt.plot(list(np.arange(0.01, 100.0, 2)), val_scores, label="val")
plt.legend(loc='lower right')
plt.xlabel("lambda")
plt.ylabel("Score")
plt.grid()
plt.show()
```



In [176]:

```
np.argmax(val_scores)
```

Out[176]:

49

In [177]:

```
val_scores[49]
```

Out[177]:

0.809298647900688

In [178]:

```
l_best2 = 0.01+2*49
l_best2
```

Out[178]:

```
98.01
```

In [179]:

```
# Model with lambda=98.01
scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/l_best2))
scaled_lr.fit(X_train, y_train)
```

Out[179]:

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('logisticregression',
                 LogisticRegression(C=0.010203040506070809))])
```

In [180]:

```
test_score = scaled_lr.score(X_test, y_test)
print(test_score)

y_pred2 = scaled_lr.predict(X_test)
```

```
0.8895804868174687
```

In [181]:

```
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score,confusion_
print(f"Accuracy : {accuracy_score(y_test, y_pred2)*100}%")
print(f"recall_score : {recall_score(y_test, y_pred2)*100}%")
print(f"precision_score : {precision_score(y_test, y_pred2)*100}%")
print(f"f1_score : {f1_score(y_test, y_pred2)*100}%")
print(f"AUC score : {metrics.roc_auc_score( y_test, y_pred2)*100}%")
print(f"confusion_matrix :")
print(confusion_matrix(y_test, y_pred2))
```

```
Accuracy : 88.95804868174687%
recall_score : 45.54827832176901%
precision_score : 95.49966009517335%
f1_score : 61.6789603090973%
AUC score : 72.51401310584153%
confusion_matrix :
[[63292   331]
 [ 8397  7024]]
```

In [182]:

```
# Let's evaluate the other metrics as well
confusion = confusion_matrix(y_test, y_pred2)
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

In [183]:

```
# Calculate the sensitivity

TP/(TP+FN)
```

Out[183]:

```
0.45548278321769015
```

In [184]:

```
# Calculate the specificity

TN/(TN+FP)
```

Out[184]:

```
0.9947974788991403
```

In [185]:

```
# Checking the classification_report

from sklearn.metrics import classification_report

print(f"{classification_report(y_test, y_pred2)}")
```

```
              precision    recall  f1-score   support

           0       0.88      0.99      0.94     63623
           1       0.95      0.46      0.62     15421

    accuracy                           0.89     79044
   macro avg       0.92      0.73      0.78     79044
weighted avg       0.90      0.89      0.87     79044
```

In [186]:

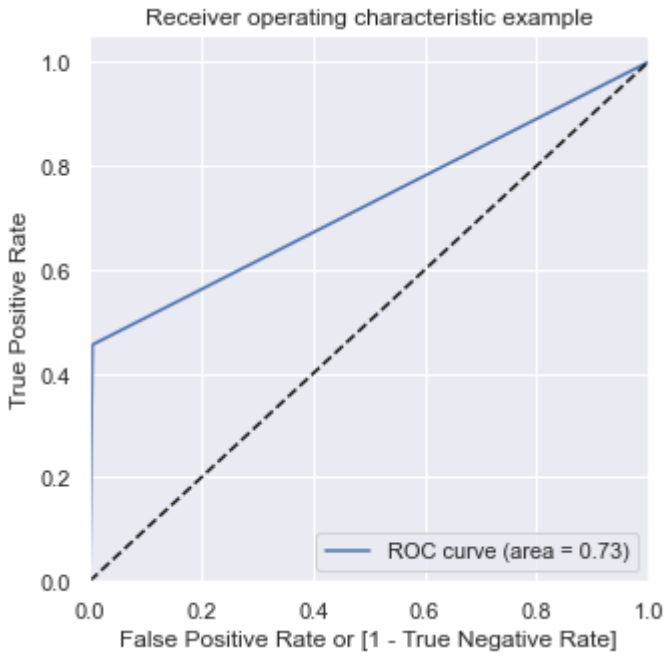
```
fpr2, tpr2, thresholds2 = metrics.roc_curve( y_test, y_pred2, drop_intermediate = False )
fpr2, tpr2, thresholds2
```

Out[186]:

```
(array([0.        , 0.00520252, 1.        ]),
 array([0.        , 0.45548278, 1.        ]),
 array([2, 1, 0], dtype=int64))
```

In [187]:

```
# Calling the ROC function

draw_roc(y_test, y_pred2)
```



In [188]:

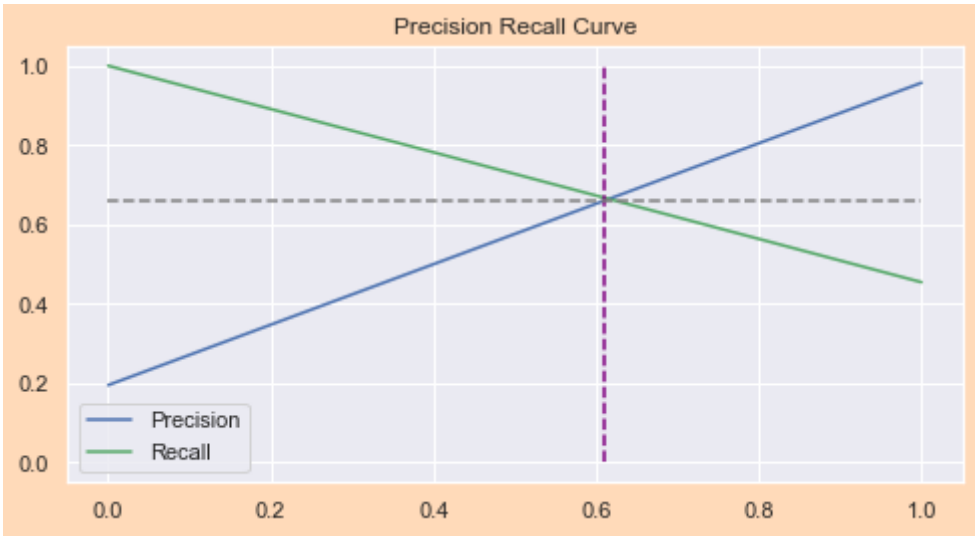
```
# Checking for precision_recall_curve metric

from sklearn.metrics import precision_recall_curve

p2, r2, thresholds2 = precision_recall_curve(y_test, y_pred)
p2, r2, thresholds2
```

Out[188]:

```
(array([0.19509387, 0.95698484, 1.        ]),
 array([1.        , 0.45444524, 0.        ]),
 array([0, 1], dtype=int64))
```

In [189]:

```
fig = plt.figure(figsize = (8,4))
fig.set_facecolor("peachpuff")

# Precision Recall Curve
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)
plt.plot(thresholds, precision[:-1], "b",label='Precision')
plt.plot(thresholds, recall[:-1], "g",label='Recall')
plt.vlines(x=0.61,ymax=1,ymin=0.0,color="purple",linestyles="--")
plt.hlines(y=0.66,xmax=1,xmin=0.0,color="grey",linestyles="--")
plt.title('Precision Recall Curve')
plt.legend()
plt.show()
```



Insights from model 2:

* Defaulters/Irresponsible Business or Customers: For our model 2 (with balancing weights) , we are getting
fi-score as 62 % to check if the credit line can be extended to the business/customers who have a higher chances
of defaulting/ not repaying the loan/credit taken. 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.
* Honest/Responsible Business or Customers :For our model 2 (with balancing weights) , we are getting fi-score as
94 % to check if the credit line can be extended to the business/customers who have a higher chances of fully
repaying the loan/credit taken. This is important as we can lose out on an opportunity to finance more supply
chains and earn interest in it.

Model 3

* Here we will change the class- weights and do re-balancing.
* In addition we will select only the features which are important for our mod
els using RFE (Recurrsive
Feature Elimaination)

In [190]:

```
# Importing 'LogisticRegression' and creating a LogisticRegression object
# For pipeline creation
from sklearn.linear_model import LogisticRegression
imputer = SimpleImputer(strategy='median', missing_values=np.nan)
lr3 = Pipeline(steps=[('imputer',imputer),
                      ('scaler',scaler),
                      ('logistic_model',LogisticRegression())
                     ])
```

In [191]:

```
# Import RFE and select 20 variables

from sklearn.feature_selection import RFE
type(lr3)
```

Out[191]:

```
sklearn.pipeline.Pipeline
```

In [192]:

```
rfe = RFE(lr3['logistic_model'], n_features_to_select=20) # running RFE with 20 variables a
```

In [193]:

```
rfe = rfe.fit(X_train, y_train)
```

In [194]:

```
# Let's take a look at which features have been selected by RFE

list(zip(X_train.columns, rfe.support_, rfe.ranking_))
```

Out[194]:

```
[('loan_amnt', False, 58),
 ('term', False, 12),
 ('int_rate', False, 38),
 ('installment', False, 55),
 ('annual_inc', False, 61),
 ('dti', False, 53),
 ('open_acc', False, 42),
 ('pub_rec', False, 32),
 ('revol_bal', False, 59),
 ('revol_util', False, 49),
 ('total_acc', False, 50),
 ('mort_acc', False, 18),
 ('pub_rec_bankruptcies', False, 33),
 ('A2', True, 1),
 ('A3', True, 1),
 ('A4', True, 1),
 ('A5', True, 1),
 ('B1', True, 1),
 ('B2', True, 1),
 ('B3', False, 6),
 ('B4', False, 10),
 ('B5', False, 14),
 ('C1', False, 22),
 ('C2', False, 60),
 ('C3', False, 46),
 ('C4', False, 34),
 ('C5', False, 45),
 ('D1', False, 26),
 ('D2', False, 23),
 ('D3', False, 56),
 ('D4', False, 19),
 ('D5', False, 20),
 ('E1', False, 16),
 ('E2', False, 5),
 ('E3', False, 4),
 ('E4', False, 3),
 ('E5', False, 2),
 ('F1', False, 44),
 ('F2', True, 1),
 ('F3', False, 41),
 ('F4', True, 1),
 ('F5', True, 1),
 ('G1', True, 1),
 ('G2', True, 1),
 ('G3', True, 1),
 ('G4', True, 1),
 ('G5', True, 1),
 ('verification_status_Source Verified', False, 24),
 ('verification_status_Verified', False, 25),
 ('application_type_INDIVIDUAL', False, 63),
 ('application_type_JOINT', False, 11),
 ('initial_list_status_w', False, 43),
 ('purpose_credit_card', False, 39),
```

```
('purpose_debt_consolidation', False, 36),
('purpose_educational', False, 66),
('purpose_home_improvement', False, 37),
('purpose_house', False, 62),
('purpose_major_purchase', False, 64),
('purpose_medical', False, 48),
('purpose_moving', False, 54),
('purpose_other', False, 47),
('purpose_renewable_energy', False, 52),
('purpose_small_business', False, 17),
('purpose_vacation', False, 40),
('purpose_wedding', False, 15),
('OTHER', False, 67),
('OWN', False, 35),
('RENT', False, 31),
('05113', True, 1),
('11650', True, 1),
('22690', False, 9),
('29597', True, 1),
('30723', False, 57),
('48052', False, 7),
('70466', False, 8),
('86630', True, 1),
('93700', True, 1),
('earliest_cr_year', False, 51),
('Lower Loan', False, 30),
('Average Loan', False, 29),
('High Loan', False, 27),
('Very Low', False, 28),
('Low', False, 21),
('Average', False, 13),
('Better Off', True, 1),
('OTHER', False, 65)]
```

In [195]:

```
# Putting all the columns selected by RFE in the variable 'new_col'

new_col = X_train.columns[rfe.support_]
```

In [196]:

```
# Select only the columns selected by RFE

X_train = X_train[new_col]
```

In [197]:

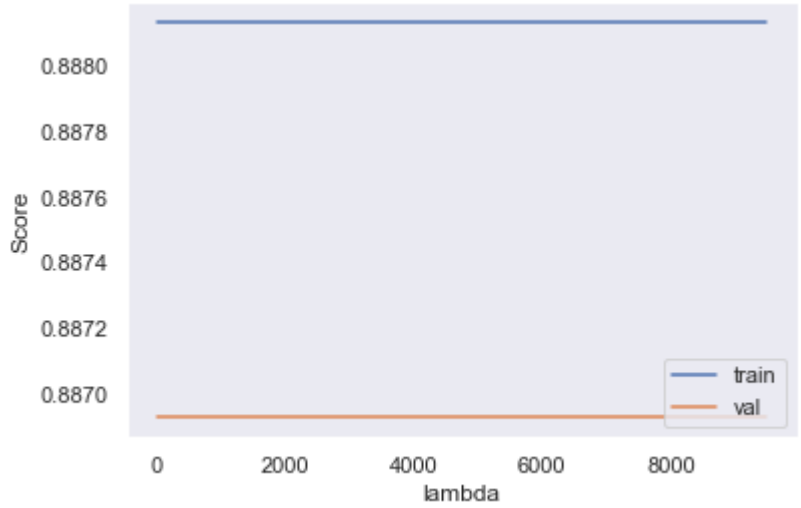
```
X_val = X_val[new_col]
```

In [198]:

```
X_test = X_test[new_col]
```

In [199]:

```
# Hyper-parameter tuning
train_scores = []
val_scores = []
scaler = StandardScaler()

for la in np.arange(0.01, 10000.0,500):
    scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la))
    scaled_lr.fit(X_train, y_train)
    train_score = scaled_lr.score(X_train, y_train)
    val_score = scaled_lr.score(X_val, y_val)
    train_scores.append(train_score)
    val_scores.append(val_score)

plt.figure()
plt.plot(list(np.arange(0.01, 10000.0,500)), train_scores, label="train")
plt.plot(list(np.arange(0.01, 10000.0,500)), val_scores, label="val")
plt.legend(loc='lower right')
plt.xlabel("lambda")
plt.ylabel("Score")
plt.grid()
plt.show()
```



In [200]:

```
np.argmax(val_scores)
```

Out[200]:

```
0
```

In [201]:

```
val_scores[0]
```

Out[201]:

```
0.8869297066497984
```

In [202]:

```
l_best3 = 0.01+500*0
l_best3
```

Out[202]:

```
0.01
```

In [203]:

```
# Model with Lambda=0.01
scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/l_best3))
scaled_lr.fit(X_train, y_train)
```

Out[203]:

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('logisticregression', LogisticRegression(C=100.0))])
```

In [204]:

```
test_score = scaled_lr.score(X_test, y_test)
print(test_score)

y_pred3 = scaled_lr.predict(X_test)
```

```
0.8888973230099692
```

In [205]:

```
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score,confusion_
print(f"Accuracy : {accuracy_score(y_test, y_pred3)*100}%")
print(f"recall_score : {recall_score(y_test, y_pred3)*100}%")
print(f"precision_score : {precision_score(y_test, y_pred3)*100}%")
print(f"f1_score : {f1_score(y_test, y_pred3)*100}%")
print(f"AUC score : {metrics.roc_auc_score( y_test, y_pred3)*100}%")
print(f"confusion_matrix :")
print(confusion_matrix(y_test, y_pred3))
```

```
Accuracy : 88.88973230099691%
recall_score : 43.05168277024836%
precision_score : 100.0%
f1_score : 60.19038984587489%
AUC score : 71.52584138512418%
confusion_matrix :
[[63623     0]
 [ 8782  6639]]
```

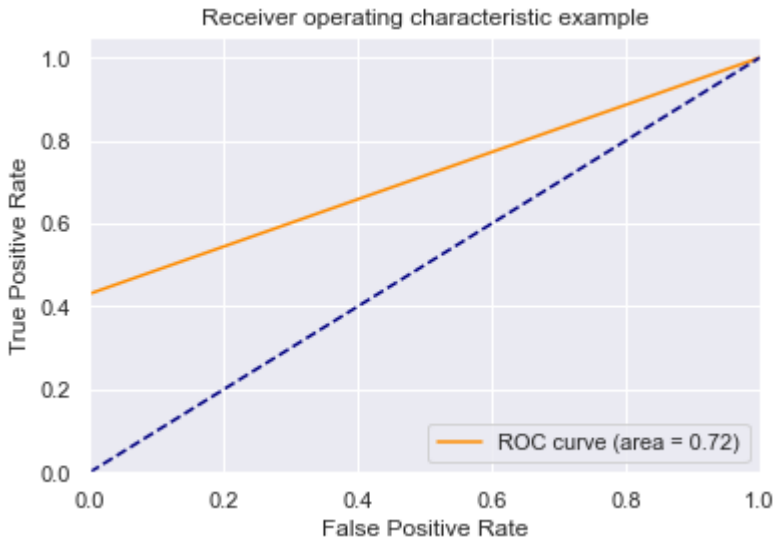
In [206]:

```
# Checking the classification_report

from sklearn.metrics import classification_report

print(f"{classification_report(y_test, y_pred3)}")
```

```
              precision    recall  f1-score   support

           0       0.88      1.00      0.94     63623
           1       1.00      0.43      0.60     15421

    accuracy                           0.89     79044
   macro avg       0.94      0.72      0.77     79044
weighted avg       0.90      0.89      0.87     79044
```

In [207]:

```
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_pred3)
roc_auc= auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange',
         label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
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 example')
plt.legend(loc="lower right")
plt.show()
```



Insights from Model 3 :

* Defaulters/Irresponsible Business or Customers: For our model 3 , we are getting fi-score as 60 % to check if
the credit line can be extended to the business/customers who have a higher chances of defaulting/ not repaying
the loan/credit taken. 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.
* Honest/Responsible Business or Customers :For our model 2 (with balancing weights) , we are getting fi-score as
100 % to check if the credit line can be extended to the business/customers who have a higher chances of
fully repaying the loan/credit taken. This is important as we can lose out on an opportunity to finance more
supply chains and earn interest in it.

Model 4

* Here we will be eliminating featres based on VIF score and model building using stats model.

In [208]:

```
loan.shape
```

Out[208]:

```
(395219, 87)
```

In [209]:

```
# Assigning the featurs as X and target as y

X= loan.drop(["loan_status"],axis =1)
y= loan["loan_status"]
```

In [210]:

```
# Split the dataset into 70% train and 30% test

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, test_size=0.3, ra
```

In [211]:

```
#Imputing mort_acc with median

X_train['mort_acc'] = X_train['mort_acc'].fillna(X_test['mort_acc'].median())
missingValue(X_train)
```

```
Total records in our data =  276653 where missing values are as follows:
```

Out[211]:

	Total Missing	In Percent
loan_amnt	0	0.0
purpose_educational	0	0.0
purpose_small_business	0	0.0
purpose_renewable_energy	0	0.0
purpose_other	0	0.0
...	...	...
D1	0	0.0
C5	0	0.0
C4	0	0.0
C3	0	0.0
OTHER	0	0.0

86 rows × 2 columns

In [212]:

```
# Feature Engineering on mort_acc on train data

X_train['mort_acc'] = X_train['mort_acc'].apply(lambda x: 1 if x > 0.0 else 0)
```

In [213]:

```
# Outliers removal for train data
for i in numerical_cols:
    X_train[i] = np.where(X_train[i] > X_train[i].quantile(0.75) + \
                          (X_train[i].quantile(0.75) - \
                           X_train[i].quantile(0.25))*1.5,
                          X_train[i].quantile(0.5),X_train[i])
    X_train[i] = np.where(X_train[i] < X_train[i].quantile(0.25) - \
                          (X_train[i].quantile(0.75) - \
                           X_train[i].quantile(0.25))*1.5, \
                          X_train[i].quantile(0.5),X_train[i])
```

In [214]:

```
# For test data
X_test['mort_acc'] = X_test['mort_acc'].fillna(X_test['mort_acc'].median())
```

In [215]:

```
# Feature Engineering on mort_acc on test data

X_test['mort_acc'] = X_test['mort_acc'].apply(lambda x: 1 if x > 0.0 else 0)
```

In [216]:

```
# Fit a logistic Regression model on X_train after adding a constant and output the summary

X_train_sm = sm.add_constant(X_train)
logm2 = sm.GLM(y_train, X_train_sm, family = sm.families.Binomial())
res = logm2.fit()
res.summary()
```

Out[216]:

Generalized Linear Model Regression Results

Dep. Variable:	loan_status	**No. Observations:**	276653
Model:	GLM	**Df Residuals:**	276566
Model Family:	Binomial	**Df Model:**	86
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-70810.
Date:	Mon, 29 Aug 2022	**Deviance:**	1.4162e+05
Time:	14:49:31	**Pearson chi2:**	1.58e+05
No. Iterations:	30	**Pseudo R-squ. (CS):**	0.3798
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-26.6017	2.31e+04	-0.001	0.999	-4.53e+04	4.53e+04
loan_amnt	2.775e-05	3.24e-06	8.566	0.000	2.14e-05	3.41e-05
term	0.4158	0.020	21.057	0.000	0.377	0.455
int_rate	-0.0580	0.006	-10.179	0.000	-0.069	-0.047
installment	7.782e-06	7.37e-05	0.106	0.916	-0.000	0.000
annual_inc	-3.537e-06	4.58e-07	-7.716	0.000	-4.43e-06	-2.64e-06
dti	0.0240	0.001	25.516	0.000	0.022	0.026
open_acc	0.0208	0.002	10.124	0.000	0.017	0.025
pub_rec	0.1811	0.036	5.054	0.000	0.111	0.251
revol_bal	-6.594e-06	1e-06	-6.587	0.000	-8.56e-06	-4.63e-06
revol_util	0.0034	0.000	9.880	0.000	0.003	0.004
total_acc	-0.0068	0.001	-7.819	0.000	-0.009	-0.005
mort_acc	-0.1351	0.018	-7.510	0.000	-0.170	-0.100
pub_rec_bankruptcies	-0.1431	0.040	-3.589	0.000	-0.221	-0.065
A2	0.4047	0.119	3.400	0.001	0.171	0.638
A3	0.5973	0.113	5.263	0.000	0.375	0.820
A4	0.7080	0.106	6.667	0.000	0.500	0.916
A5	0.9264	0.103	8.984	0.000	0.724	1.128
B1	1.1786	0.102	11.526	0.000	0.978	1.379
B2	1.2445	0.102	12.168	0.000	1.044	1.445
B3	1.3928	0.102	13.597	0.000	1.192	1.594

B4	1.5523	0.104	14.971	0.000	1.349	1.756
B5	1.6850	0.105	16.048	0.000	1.479	1.891
C1	1.8387	0.106	17.380	0.000	1.631	2.046
C2	1.9533	0.107	18.229	0.000	1.743	2.163
C3	2.1516	0.109	19.827	0.000	1.939	2.364
C4	2.2354	0.110	20.312	0.000	2.020	2.451
C5	2.2747	0.112	20.230	0.000	2.054	2.495
D1	2.4407	0.115	21.272	0.000	2.216	2.666
D2	2.5090	0.117	21.390	0.000	2.279	2.739
D3	2.5809	0.119	21.619	0.000	2.347	2.815
D4	2.6632	0.121	21.933	0.000	2.425	2.901
D5	2.6854	0.124	21.619	0.000	2.442	2.929
E1	2.7584	0.127	21.774	0.000	2.510	3.007
E2	2.9524	0.129	22.895	0.000	2.700	3.205
E3	2.9735	0.132	22.483	0.000	2.714	3.233
E4	3.0269	0.136	22.237	0.000	2.760	3.294
E5	3.0909	0.140	22.066	0.000	2.816	3.365
F1	3.0152	0.145	20.765	0.000	2.731	3.300
F2	3.2964	0.150	21.949	0.000	3.002	3.591
F3	3.3161	0.155	21.427	0.000	3.013	3.619
F4	3.3565	0.159	21.140	0.000	3.045	3.668
F5	3.4003	0.158	21.547	0.000	3.091	3.710
G1	2.8772	0.148	19.395	0.000	2.586	3.168
G2	2.8558	0.159	17.916	0.000	2.543	3.168
G3	3.1430	0.178	17.688	0.000	2.795	3.491
G4	2.9695	0.192	15.447	0.000	2.593	3.346
G5	2.8183	0.218	12.949	0.000	2.392	3.245
verification_status_Source Verified	0.1669	0.018	9.257	0.000	0.132	0.202
verification_status_Verified	0.1012	0.018	5.521	0.000	0.065	0.137
application_type_INDIVIDUAL	-0.2877	0.208	-1.385	0.166	-0.695	0.119
application_type_JOINT	-1.1210	0.307	-3.657	0.000	-1.722	-0.520
initial_list_status_w	-0.0154	0.014	-1.091	0.275	-0.043	0.012
purpose_credit_card	0.2428	0.079	3.081	0.002	0.088	0.397
purpose_debt_consolidation	0.3093	0.078	3.983	0.000	0.157	0.461
purpose_educational	0.4420	0.303	1.461	0.144	-0.151	1.035
purpose_home_improvement	0.3888	0.082	4.727	0.000	0.228	0.550
purpose_house	0.2475	0.115	2.152	0.031	0.022	0.473
purpose_major_purchase	0.3555	0.091	3.924	0.000	0.178	0.533
purpose_medical	0.4656	0.099	4.700	0.000	0.271	0.660
purpose_moving	0.3627	0.107	3.390	0.001	0.153	0.572

purpose_other	0.3286	0.082	4.012	0.000	0.168	0.489
purpose_renewable_energy	0.4955	0.240	2.068	0.039	0.026	0.965
purpose_small_business	0.7251	0.091	7.944	0.000	0.546	0.904
purpose_vacation	0.3595	0.114	3.156	0.002	0.136	0.583
purpose_wedding	-0.0840	0.140	-0.601	0.548	-0.358	0.190
OTHER	0.4864	0.329	1.477	0.140	-0.159	1.132
OWN	0.0879	0.025	3.580	0.000	0.040	0.136
RENT	0.2121	0.018	11.751	0.000	0.177	0.247
05113	-0.0052	3.29e+04	-1.57e-07	1.000	-6.44e+04	6.44e+04
11650	62.9651	5.22e+04	0.001	0.999	-1.02e+05	1.02e+05
22690	30.0493	2.31e+04	0.001	0.999	-4.53e+04	4.53e+04
29597	-0.0124	3.28e+04	-3.79e-07	1.000	-6.44e+04	6.44e+04
30723	30.0518	2.31e+04	0.001	0.999	-4.53e+04	4.53e+04
48052	30.0839	2.31e+04	0.001	0.999	-4.53e+04	4.53e+04
70466	30.0633	2.31e+04	0.001	0.999	-4.53e+04	4.53e+04
86630	62.9395	5.29e+04	0.001	0.999	-1.04e+05	1.04e+05
93700	62.9608	5.25e+04	0.001	0.999	-1.03e+05	1.03e+05
earliest_cr_year	-0.0033	0.001	-3.269	0.001	-0.005	-0.001
Lower Loan	-0.0479	0.026	-1.846	0.065	-0.099	0.003
Average Loan	-0.2388	0.049	-4.903	0.000	-0.334	-0.143
High Loan	-0.4413	0.075	-5.877	0.000	-0.589	-0.294
Very Low	-0.1893	0.035	-5.419	0.000	-0.258	-0.121
Low	-0.3522	0.041	-8.549	0.000	-0.433	-0.271
Average	-0.4751	0.055	-8.669	0.000	-0.583	-0.368
Better Off	-0.7430	0.278	-2.671	0.008	-1.288	-0.198
OTHER	0.0905	0.668	0.136	0.892	-1.218	1.399

There are quite a few variable which have a p-value greater than 0.05. We will need to take care of them. But first, let's also look at the VIFs.

In [217]:

```
# to drop :installment, purpose_wedding, OTHER, Lower Loan
```

In [218]:

```
X_train.columns
```

Out[218]:

```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'annual_inc', 'dti',
       'open_acc', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc',
       'mort_acc', 'pub_rec_bankruptcies', 'A2', 'A3', 'A4', 'A5', 'B1', 'B
2',
       'B3', 'B4', 'B5', 'C1', 'C2', 'C3', 'C4', 'C5', 'D1', 'D2', 'D3', 'D
4',
       'D5', 'E1', 'E2', 'E3', 'E4', 'E5', 'F1', 'F2', 'F3', 'F4', 'F5', 'G
1',
       'G2', 'G3', 'G4', 'G5', 'verification_status_Source Verified',
       'verification_status_Verified', 'application_type_INDIVIDUAL',
       'application_type_JOINT', 'initial_list_status_w',
       'purpose_credit_card', 'purpose_debt_consolidation',
       'purpose_educational', 'purpose_home_improvement', 'purpose_house',
       'purpose_major_purchase', 'purpose_medical', 'purpose_moving',
       'purpose_other', 'purpose_renewable_energy', 'purpose_small_busines
s',
       'purpose_vacation', 'purpose_wedding', 'OTHER', 'OWN', 'RENT', '0511
3',
       '11650', '22690', '29597', '30723', '48052', '70466', '86630', '9370
0',
       'earliest_cr_year', 'Lower Loan', 'Average Loan', 'High Loan',
       'Very Low', 'Low', 'Average', 'Better Off', 'OTHER'],
      dtype='object')
```

In [219]:

```
X_train.drop(['installment','purpose_wedding','OTHER','05113',\
              '11650', '22690', '29597', '30723', '48052', '70466', '86630', '93700','Lower
```

In [220]:

```
# Refit the model with the new set of features

logm1 = sm.GLM(y_train,(sm.add_constant(X_train)), family = sm.families.Binomial())
logm1.fit().summary()
```

Out[220]:

Generalized Linear Model Regression Results

Dep. Variable:	loan_status	**No. Observations:**	276653
Model:	GLM	**Df Residuals:**	276580
Model Family:	Binomial	**Df Model:**	72
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-1.2413e+05
Date:	Mon, 29 Aug 2022	**Deviance:**	2.4827e+05
Time:	14:49:48	**Pearson chi2:**	2.77e+05
No. Iterations:	6	**Pseudo R-squ. (CS):**	0.08810
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	2.6416	1.555	1.698	0.089	-0.407	5.690
loan_amnt	2.403e-05	1.3e-06	18.468	0.000	2.15e-05	2.66e-05
term	0.3920	0.014	28.647	0.000	0.365	0.419
int_rate	-0.0565	0.004	-13.182	0.000	-0.065	-0.048
annual_inc	-3.821e-06	3.45e-07	-11.090	0.000	-4.5e-06	-3.15e-06
dti	0.0237	0.001	33.404	0.000	0.022	0.025
open_acc	0.0220	0.002	14.264	0.000	0.019	0.025
pub_rec	0.2040	0.027	7.582	0.000	0.151	0.257
revol_bal	-6.603e-06	7.56e-07	-8.728	0.000	-8.09e-06	-5.12e-06
revol_util	0.0033	0.000	13.003	0.000	0.003	0.004
total_acc	-0.0068	0.001	-10.407	0.000	-0.008	-0.006
mort_acc	-0.1454	0.014	-10.726	0.000	-0.172	-0.119
pub_rec_bankruptcies	-0.1743	0.030	-5.817	0.000	-0.233	-0.116
A2	0.4306	0.092	4.706	0.000	0.251	0.610
A3	0.6160	0.087	7.052	0.000	0.445	0.787
A4	0.8267	0.081	10.248	0.000	0.669	0.985
A5	1.0131	0.079	12.855	0.000	0.859	1.168
B1	1.1941	0.078	15.232	0.000	1.040	1.348
B2	1.3135	0.078	16.782	0.000	1.160	1.467
B3	1.4729	0.078	18.807	0.000	1.319	1.626
B4	1.6153	0.079	20.370	0.000	1.460	1.771
B5	1.7800	0.080	22.193	0.000	1.623	1.937

C1	1.8819	0.081	23.268	0.000	1.723	2.040
C2	2.0603	0.082	25.192	0.000	1.900	2.221
C3	2.1785	0.083	26.279	0.000	2.016	2.341
C4	2.2614	0.084	26.903	0.000	2.097	2.426
C5	2.3521	0.086	27.431	0.000	2.184	2.520
D1	2.4929	0.088	28.487	0.000	2.321	2.664
D2	2.5577	0.089	28.590	0.000	2.382	2.733
D3	2.6184	0.091	28.768	0.000	2.440	2.797
D4	2.7206	0.093	29.406	0.000	2.539	2.902
D5	2.7751	0.095	29.348	0.000	2.590	2.960
E1	2.8371	0.096	29.420	0.000	2.648	3.026
E2	2.9965	0.098	30.526	0.000	2.804	3.189
E3	3.0269	0.101	30.076	0.000	2.830	3.224
E4	3.1398	0.104	30.329	0.000	2.937	3.343
E5	3.2174	0.106	30.218	0.000	3.009	3.426
F1	3.1246	0.110	28.375	0.000	2.909	3.340
F2	3.3277	0.114	29.161	0.000	3.104	3.551
F3	3.4108	0.118	29.019	0.000	3.180	3.641
F4	3.4694	0.120	28.829	0.000	3.234	3.705
F5	3.4556	0.120	28.843	0.000	3.221	3.690
G1	3.0302	0.112	26.981	0.000	2.810	3.250
G2	2.9537	0.122	24.295	0.000	2.715	3.192
G3	3.1619	0.132	23.999	0.000	2.904	3.420
G4	2.9406	0.148	19.865	0.000	2.651	3.231
G5	3.0543	0.164	18.627	0.000	2.733	3.376
verification_status_Source Verified	0.1439	0.014	10.573	0.000	0.117	0.171
verification_status_Verified	0.0927	0.014	6.711	0.000	0.066	0.120
application_type_INDIVIDUAL	-0.1901	0.154	-1.237	0.216	-0.491	0.111
application_type_JOINT	-1.0405	0.233	-4.457	0.000	-1.498	-0.583
initial_list_status_w	-0.0123	0.011	-1.149	0.251	-0.033	0.009
purpose_credit_card	0.1913	0.049	3.917	0.000	0.096	0.287
purpose_debt_consolidation	0.2314	0.048	4.846	0.000	0.138	0.325
purpose_educational	0.2674	0.229	1.167	0.243	-0.182	0.716
purpose_home_improvement	0.3396	0.052	6.526	0.000	0.238	0.442
purpose_house	0.1082	0.082	1.322	0.186	-0.052	0.269
purpose_major_purchase	0.2642	0.060	4.431	0.000	0.147	0.381
purpose_medical	0.3651	0.067	5.465	0.000	0.234	0.496
purpose_moving	0.3232	0.073	4.454	0.000	0.181	0.465
purpose_other	0.2597	0.052	5.032	0.000	0.159	0.361
purpose_renewable_energy	0.4819	0.168	2.872	0.004	0.153	0.811

purpose_small_business	0.7124	0.060	11.926	0.000	0.595	0.829
purpose_vacation	0.2609	0.080	3.270	0.001	0.105	0.417
OWN	0.0987	0.019	5.330	0.000	0.062	0.135
RENT	0.1989	0.014	14.606	0.000	0.172	0.226
earliest_cr_year	-0.0030	0.001	-3.849	0.000	-0.004	-0.001
Average Loan	-0.1523	0.021	-7.342	0.000	-0.193	-0.112
High Loan	-0.3685	0.034	-10.736	0.000	-0.436	-0.301
Very Low	-0.1833	0.026	-6.999	0.000	-0.235	-0.132
Low	-0.3386	0.031	-10.954	0.000	-0.399	-0.278
Average	-0.4195	0.041	-10.206	0.000	-0.500	-0.339
Better Off	-0.4776	0.190	-2.510	0.012	-0.851	-0.105

In [221]:

```
# Make a VIF dataframe for all the variables present
# Import 'variance_inflation_factor'

from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = pd.DataFrame()
vif['Features'] = X_train.columns
vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[221]:

	Features	VIF
65	earliest_cr_year	1560.81
48	application_type_INDIVIDUAL	1394.20
2	int_rate	189.25
52	purpose_debt_consolidation	38.94
3	annual_inc	21.95
...	...	...
44	G4	1.14
45	G5	1.10
60	purpose_renewable_energy	1.06
53	purpose_educational	1.04
71	Better Off	1.04

72 rows × 2 columns

In [222]:

```
X_train.drop(['earliest_cr_year','application_type_INDIVIDUAL','int_rate'], axis = 1, inpla
```

In [223]:

```
# Refit the model with the new set of features

logm1 = sm.GLM(y_train,(sm.add_constant(X_train)), family = sm.families.Binomial())
logm1.fit().summary()
```

Out[223]:

Generalized Linear Model Regression Results

Dep. Variable:	loan_status	**No. Observations:**	276653
Model:	GLM	**Df Residuals:**	276583
Model Family:	Binomial	**Df Model:**	69
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-1.2423e+05
Date:	Mon, 29 Aug 2022	**Deviance:**	2.4845e+05
Time:	14:57:47	**Pearson chi2:**	2.77e+05
No. Iterations:	6	**Pseudo R-squ. (CS):**	0.08748
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-3.8366	0.090	-42.397	0.000	-4.014	-3.659
loan_amnt	2.391e-05	1.3e-06	18.374	0.000	2.14e-05	2.65e-05
term	0.3990	0.014	29.176	0.000	0.372	0.426
annual_inc	-3.807e-06	3.45e-07	-11.050	0.000	-4.48e-06	-3.13e-06
dti	0.0243	0.001	34.212	0.000	0.023	0.026
open_acc	0.0213	0.002	13.850	0.000	0.018	0.024
pub_rec	0.2229	0.027	8.302	0.000	0.170	0.276
revol_bal	-6.428e-06	7.52e-07	-8.542	0.000	-7.9e-06	-4.95e-06
revol_util	0.0031	0.000	12.003	0.000	0.003	0.004
total_acc	-0.0065	0.001	-10.134	0.000	-0.008	-0.005
mort_acc	-0.1203	0.013	-8.987	0.000	-0.147	-0.094
pub_rec_bankruptcies	-0.1841	0.030	-6.147	0.000	-0.243	-0.125
A2	0.3934	0.091	4.302	0.000	0.214	0.573
A3	0.5352	0.087	6.142	0.000	0.364	0.706
A4	0.7241	0.080	9.017	0.000	0.567	0.882
A5	0.8646	0.078	11.081	0.000	0.712	1.018
B1	0.9986	0.077	12.963	0.000	0.848	1.150
B2	1.0609	0.076	13.965	0.000	0.912	1.210
B3	1.1684	0.075	15.590	0.000	1.022	1.315
B4	1.2641	0.075	16.890	0.000	1.117	1.411
B5	1.4002	0.075	18.661	0.000	1.253	1.547
C1	1.4658	0.075	19.631	0.000	1.319	1.612

C2	1.6109	0.075	21.599	0.000	1.465	1.757
C3	1.6958	0.075	22.721	0.000	1.549	1.842
C4	1.7454	0.075	23.370	0.000	1.599	1.892
C5	1.7959	0.075	23.954	0.000	1.649	1.943
D1	1.8961	0.075	25.199	0.000	1.749	2.044
D2	1.9207	0.076	25.381	0.000	1.772	2.069
D3	1.9531	0.076	25.654	0.000	1.804	2.102
D4	2.0226	0.076	26.514	0.000	1.873	2.172
D5	2.0440	0.077	26.527	0.000	1.893	2.195
E1	2.0815	0.078	26.673	0.000	1.929	2.234
E2	2.2050	0.078	28.196	0.000	2.052	2.358
E3	2.2035	0.079	27.740	0.000	2.048	2.359
E4	2.2713	0.080	28.244	0.000	2.114	2.429
E5	2.3060	0.082	28.274	0.000	2.146	2.466
F1	2.1793	0.084	25.875	0.000	2.014	2.344
F2	2.3431	0.087	26.965	0.000	2.173	2.513
F3	2.4001	0.090	26.761	0.000	2.224	2.576
F4	2.4630	0.094	26.334	0.000	2.280	2.646
F5	2.5525	0.098	25.966	0.000	2.360	2.745
G1	2.4765	0.104	23.784	0.000	2.272	2.681
G2	2.4152	0.114	21.111	0.000	2.191	2.639
G3	2.6361	0.125	21.059	0.000	2.391	2.881
G4	2.4108	0.142	16.972	0.000	2.132	2.689
G5	2.5513	0.159	16.062	0.000	2.240	2.863
verification_status_Source Verified	0.1497	0.014	11.001	0.000	0.123	0.176
verification_status_Verified	0.0853	0.014	6.192	0.000	0.058	0.112
application_type_JOINT	-0.8367	0.176	-4.757	0.000	-1.181	-0.492
initial_list_status_w	0.0078	0.011	0.741	0.459	-0.013	0.028
purpose_credit_card	0.1890	0.049	3.873	0.000	0.093	0.285
purpose_debt_consolidation	0.2314	0.048	4.848	0.000	0.138	0.325
purpose_educational	0.3358	0.227	1.477	0.140	-0.110	0.781
purpose_home_improvement	0.3407	0.052	6.551	0.000	0.239	0.443
purpose_house	0.1107	0.082	1.355	0.175	-0.049	0.271
purpose_major_purchase	0.2681	0.060	4.499	0.000	0.151	0.385
purpose_medical	0.3704	0.067	5.548	0.000	0.240	0.501
purpose_moving	0.3245	0.073	4.473	0.000	0.182	0.467
purpose_other	0.2590	0.052	5.023	0.000	0.158	0.360
purpose_renewable_energy	0.4865	0.168	2.901	0.004	0.158	0.815
purpose_small_business	0.7149	0.060	11.976	0.000	0.598	0.832
purpose_vacation	0.2590	0.080	3.245	0.001	0.103	0.415

OWN	0.1117	0.018	6.050	0.000	0.076	0.148
RENT	0.2087	0.014	15.350	0.000	0.182	0.235
Average Loan	-0.1528	0.021	-7.366	0.000	-0.193	-0.112
High Loan	-0.3621	0.034	-10.557	0.000	-0.429	-0.295
Very Low	-0.1862	0.026	-7.117	0.000	-0.238	-0.135
Low	-0.3381	0.031	-10.944	0.000	-0.399	-0.278
Average	-0.4111	0.041	-10.007	0.000	-0.492	-0.331
Better Off	-0.4528	0.190	-2.382	0.017	-0.825	-0.080

In [224]:

```
# to drop :purpose_educational, purpose_house
```

In [225]:

```
X_train.drop(['purpose_educational','purpose_house'], axis = 1, inplace = True)
```

In [226]:

```
# Refit the model with the new set of features

logm1 = sm.GLM(y_train,(sm.add_constant(X_train)), family = sm.families.Binomial())
logm1.fit().summary()
```

Out[226]:

Generalized Linear Model Regression Results

Dep. Variable:	loan_status	**No. Observations:**	276653
Model:	GLM	**Df Residuals:**	276585
Model Family:	Binomial	**Df Model:**	67
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-1.2423e+05
Date:	Mon, 29 Aug 2022	**Deviance:**	2.4846e+05
Time:	14:58:02	**Pearson chi2:**	2.77e+05
No. Iterations:	6	**Pseudo R-squ. (CS):**	0.08747
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-3.7938	0.086	-43.946	0.000	-3.963	-3.625
loan_amnt	2.396e-05	1.3e-06	18.429	0.000	2.14e-05	2.65e-05
term	0.3983	0.014	29.145	0.000	0.372	0.425
annual_inc	-3.809e-06	3.45e-07	-11.055	0.000	-4.48e-06	-3.13e-06
dti	0.0243	0.001	34.206	0.000	0.023	0.026
open_acc	0.0213	0.002	13.851	0.000	0.018	0.024
pub_rec	0.2230	0.027	8.308	0.000	0.170	0.276
revol_bal	-6.434e-06	7.52e-07	-8.551	0.000	-7.91e-06	-4.96e-06
revol_util	0.0031	0.000	11.984	0.000	0.003	0.004
total_acc	-0.0065	0.001	-10.132	0.000	-0.008	-0.005
mort_acc	-0.1200	0.013	-8.970	0.000	-0.146	-0.094
pub_rec_bankruptcies	-0.1842	0.030	-6.152	0.000	-0.243	-0.126
A2	0.3934	0.091	4.301	0.000	0.214	0.573
A3	0.5357	0.087	6.147	0.000	0.365	0.706
A4	0.7246	0.080	9.023	0.000	0.567	0.882
A5	0.8653	0.078	11.089	0.000	0.712	1.018
B1	0.9992	0.077	12.972	0.000	0.848	1.150
B2	1.0616	0.076	13.975	0.000	0.913	1.210
B3	1.1691	0.075	15.599	0.000	1.022	1.316
B4	1.2649	0.075	16.902	0.000	1.118	1.412
B5	1.4010	0.075	18.672	0.000	1.254	1.548
C1	1.4669	0.075	19.647	0.000	1.321	1.613

C2	1.6118	0.075	21.612	0.000	1.466	1.758
C3	1.6968	0.075	22.735	0.000	1.551	1.843
C4	1.7463	0.075	23.383	0.000	1.600	1.893
C5	1.7970	0.075	23.969	0.000	1.650	1.944
D1	1.8972	0.075	25.215	0.000	1.750	2.045
D2	1.9218	0.076	25.396	0.000	1.773	2.070
D3	1.9540	0.076	25.668	0.000	1.805	2.103
D4	2.0237	0.076	26.530	0.000	1.874	2.173
D5	2.0451	0.077	26.543	0.000	1.894	2.196
E1	2.0827	0.078	26.690	0.000	1.930	2.236
E2	2.2064	0.078	28.215	0.000	2.053	2.360
E3	2.2048	0.079	27.759	0.000	2.049	2.361
E4	2.2729	0.080	28.266	0.000	2.115	2.430
E5	2.3075	0.082	28.294	0.000	2.148	2.467
F1	2.1808	0.084	25.894	0.000	2.016	2.346
F2	2.3456	0.087	26.997	0.000	2.175	2.516
F3	2.4018	0.090	26.782	0.000	2.226	2.578
F4	2.4646	0.094	26.352	0.000	2.281	2.648
F5	2.5549	0.098	25.993	0.000	2.362	2.748
G1	2.4788	0.104	23.808	0.000	2.275	2.683
G2	2.4181	0.114	21.137	0.000	2.194	2.642
G3	2.6388	0.125	21.083	0.000	2.394	2.884
G4	2.4145	0.142	16.999	0.000	2.136	2.693
G5	2.5567	0.159	16.100	0.000	2.245	2.868
verification_status_Source Verified	0.1496	0.014	10.998	0.000	0.123	0.176
verification_status_Verified	0.0852	0.014	6.191	0.000	0.058	0.112
application_type_JOINT	-0.8366	0.176	-4.756	0.000	-1.181	-0.492
initial_list_status_w	0.0078	0.011	0.744	0.457	-0.013	0.028
purpose_credit_card	0.1453	0.040	3.628	0.000	0.067	0.224
purpose_debt_consolidation	0.1876	0.039	4.850	0.000	0.112	0.263
purpose_home_improvement	0.2968	0.044	6.772	0.000	0.211	0.383
purpose_major_purchase	0.2243	0.053	4.260	0.000	0.121	0.327
purpose_medical	0.3266	0.061	5.385	0.000	0.208	0.445
purpose_moving	0.2806	0.067	4.192	0.000	0.149	0.412
purpose_other	0.2152	0.043	4.965	0.000	0.130	0.300
purpose_renewable_energy	0.4426	0.165	2.677	0.007	0.119	0.767
purpose_small_business	0.6707	0.053	12.747	0.000	0.568	0.774
purpose_vacation	0.2153	0.075	2.878	0.004	0.069	0.362
OWN	0.1118	0.018	6.055	0.000	0.076	0.148
RENT	0.2089	0.014	15.369	0.000	0.182	0.236

Average Loan	-0.1530	0.021	-7.376	0.000	-0.194	-0.112
High Loan	-0.3628	0.034	-10.577	0.000	-0.430	-0.296
Very Low	-0.1865	0.026	-7.127	0.000	-0.238	-0.135
Low	-0.3384	0.031	-10.954	0.000	-0.399	-0.278
Average	-0.4113	0.041	-10.012	0.000	-0.492	-0.331
Better Off	-0.4533	0.190	-2.385	0.017	-0.826	-0.081

In [227]:

```
#to drop: initial_list_status_w
```

In [228]:

```
X_train.drop(['initial_list_status_w'], axis = 1, inplace = True)
```

In [229]:

```
# Refit the model with the new set of features

logm1 = sm.GLM(y_train,(sm.add_constant(X_train)), family = sm.families.Binomial())
logm1.fit().summary()
```

Out[229]:

Generalized Linear Model Regression Results

Dep. Variable:	loan_status	**No. Observations:**	276653
Model:	GLM	**Df Residuals:**	276586
Model Family:	Binomial	**Df Model:**	66
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-1.2423e+05
Date:	Mon, 29 Aug 2022	**Deviance:**	2.4846e+05
Time:	14:58:43	**Pearson chi2:**	2.77e+05
No. Iterations:	6	**Pseudo R-squ. (CS):**	0.08747
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-3.7912	0.086	-43.954	0.000	-3.960	-3.622
loan_amnt	2.397e-05	1.3e-06	18.436	0.000	2.14e-05	2.65e-05
term	0.3995	0.014	29.410	0.000	0.373	0.426
annual_inc	-3.805e-06	3.45e-07	-11.043	0.000	-4.48e-06	-3.13e-06
dti	0.0243	0.001	34.340	0.000	0.023	0.026
open_acc	0.0213	0.002	13.845	0.000	0.018	0.024
pub_rec	0.2237	0.027	8.337	0.000	0.171	0.276
revol_bal	-6.438e-06	7.52e-07	-8.556	0.000	-7.91e-06	-4.96e-06
revol_util	0.0031	0.000	11.961	0.000	0.003	0.004
total_acc	-0.0065	0.001	-10.127	0.000	-0.008	-0.005
mort_acc	-0.1206	0.013	-9.037	0.000	-0.147	-0.094
pub_rec_bankruptcies	-0.1843	0.030	-6.156	0.000	-0.243	-0.126
A2	0.3932	0.091	4.299	0.000	0.214	0.572
A3	0.5353	0.087	6.144	0.000	0.365	0.706
A4	0.7242	0.080	9.018	0.000	0.567	0.882
A5	0.8652	0.078	11.089	0.000	0.712	1.018
B1	0.9985	0.077	12.963	0.000	0.848	1.149
B2	1.0609	0.076	13.967	0.000	0.912	1.210
B3	1.1683	0.075	15.589	0.000	1.021	1.315
B4	1.2643	0.075	16.894	0.000	1.118	1.411
B5	1.4004	0.075	18.666	0.000	1.253	1.547
C1	1.4664	0.075	19.641	0.000	1.320	1.613

C2	1.6112	0.075	21.606	0.000	1.465	1.757
C3	1.6964	0.075	22.731	0.000	1.550	1.843
C4	1.7459	0.075	23.379	0.000	1.600	1.892
C5	1.7966	0.075	23.964	0.000	1.650	1.943
D1	1.8963	0.075	25.206	0.000	1.749	2.044
D2	1.9209	0.076	25.388	0.000	1.773	2.069
D3	1.9530	0.076	25.659	0.000	1.804	2.102
D4	2.0228	0.076	26.522	0.000	1.873	2.172
D5	2.0441	0.077	26.534	0.000	1.893	2.195
E1	2.0817	0.078	26.681	0.000	1.929	2.235
E2	2.2054	0.078	28.207	0.000	2.052	2.359
E3	2.2039	0.079	27.750	0.000	2.048	2.360
E4	2.2719	0.080	28.257	0.000	2.114	2.429
E5	2.3066	0.082	28.286	0.000	2.147	2.466
F1	2.1792	0.084	25.883	0.000	2.014	2.344
F2	2.3441	0.087	26.987	0.000	2.174	2.514
F3	2.4004	0.090	26.772	0.000	2.225	2.576
F4	2.4631	0.094	26.342	0.000	2.280	2.646
F5	2.5533	0.098	25.983	0.000	2.361	2.746
G1	2.4773	0.104	23.798	0.000	2.273	2.681
G2	2.4163	0.114	21.126	0.000	2.192	2.640
G3	2.6374	0.125	21.074	0.000	2.392	2.883
G4	2.4135	0.142	16.993	0.000	2.135	2.692
G5	2.5555	0.159	16.093	0.000	2.244	2.867
verification_status_Source Verified	0.1500	0.014	11.029	0.000	0.123	0.177
verification_status_Verified	0.0848	0.014	6.163	0.000	0.058	0.112
application_type_JOINT	-0.8338	0.176	-4.741	0.000	-1.179	-0.489
purpose_credit_card	0.1463	0.040	3.654	0.000	0.068	0.225
purpose_debt_consolidation	0.1884	0.039	4.875	0.000	0.113	0.264
purpose_home_improvement	0.2977	0.044	6.794	0.000	0.212	0.384
purpose_major_purchase	0.2248	0.053	4.272	0.000	0.122	0.328
purpose_medical	0.3275	0.061	5.402	0.000	0.209	0.446
purpose_moving	0.2814	0.067	4.205	0.000	0.150	0.413
purpose_other	0.2159	0.043	4.983	0.000	0.131	0.301
purpose_renewable_energy	0.4430	0.165	2.679	0.007	0.119	0.767
purpose_small_business	0.6707	0.053	12.747	0.000	0.568	0.774
purpose_vacation	0.2161	0.075	2.890	0.004	0.070	0.363
OWN	0.1118	0.018	6.051	0.000	0.076	0.148
RENT	0.2085	0.014	15.352	0.000	0.182	0.235
Average Loan	-0.1530	0.021	-7.375	0.000	-0.194	-0.112

High Loan	-0.3626	0.034	-10.571	0.000	-0.430	-0.295
Very Low	-0.1864	0.026	-7.123	0.000	-0.238	-0.135
Low	-0.3381	0.031	-10.947	0.000	-0.399	-0.278
Average	-0.4109	0.041	-10.003	0.000	-0.491	-0.330
Better Off	-0.4529	0.190	-2.382	0.017	-0.825	-0.080

In [230]:

```
# Make a VIF dataframe for all the variables present

vif = pd.DataFrame()
vif['Features'] = X_train.columns
vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif.head(10)
```

Out[230]:

	Features	VIF
2	annual_inc	21.59
49	purpose_debt_consolidation	20.07
63	Low	19.02
0	loan_amnt	16.83
4	open_acc	12.91
8	total_acc	11.51
64	Average	11.19
7	revol_util	9.16
62	Very Low	8.27
48	purpose_credit_card	7.74

In [231]:

```
vif.head(10)
```

Out[231]:

	Features	VIF
2	annual_inc	21.59
49	purpose_debt_consolidation	20.07
63	Low	19.02
0	loan_amnt	16.83
4	open_acc	12.91
8	total_acc	11.51
64	Average	11.19
7	revol_util	9.16
62	Very Low	8.27
48	purpose_credit_card	7.74

In [232]:

```
X_train.drop(['annual_inc','purpose_debt_consolidation','Low','loan_amnt','open_acc'], axis
```

In [233]:

```
# Refit the model with the new set of features

logm1 = sm.GLM(y_train,(sm.add_constant(X_train)), family = sm.families.Binomial())
logm1.fit().summary()
```

Out[233]:

Generalized Linear Model Regression Results

Dep. Variable:	loan_status	**No. Observations:**	276653
Model:	GLM	**Df Residuals:**	276591
Model Family:	Binomial	**Df Model:**	61
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-1.2466e+05
Date:	Mon, 29 Aug 2022	**Deviance:**	2.4932e+05
Time:	15:09:27	**Pearson chi2:**	2.77e+05
No. Iterations:	6	**Pseudo R-squ. (CS):**	0.08463
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-3.7985	0.075	-50.893	0.000	-3.945	-3.652
term	0.4424	0.013	34.114	0.000	0.417	0.468
dti	0.0280	0.001	40.756	0.000	0.027	0.029
pub_rec	0.2090	0.027	7.815	0.000	0.157	0.261
revol_bal	-3.584e-06	7.16e-07	-5.010	0.000	-4.99e-06	-2.18e-06
revol_util	0.0020	0.000	7.929	0.000	0.001	0.002
total_acc	-0.0036	0.001	-6.518	0.000	-0.005	-0.003
mort_acc	-0.1548	0.013	-11.699	0.000	-0.181	-0.129
pub_rec_bankruptcies	-0.1874	0.030	-6.278	0.000	-0.246	-0.129
A2	0.3984	0.091	4.358	0.000	0.219	0.578
A3	0.5476	0.087	6.288	0.000	0.377	0.718
A4	0.7480	0.080	9.320	0.000	0.591	0.905
A5	0.8883	0.078	11.390	0.000	0.735	1.041
B1	1.0238	0.077	13.299	0.000	0.873	1.175
B2	1.0921	0.076	14.386	0.000	0.943	1.241
B3	1.1997	0.075	16.019	0.000	1.053	1.346
B4	1.2984	0.075	17.363	0.000	1.152	1.445
B5	1.4332	0.075	19.116	0.000	1.286	1.580
C1	1.5021	0.075	20.136	0.000	1.356	1.648
C2	1.6475	0.075	22.112	0.000	1.501	1.794
C3	1.7367	0.075	23.292	0.000	1.591	1.883
C4	1.7896	0.075	23.987	0.000	1.643	1.936

C5	1.8393	0.075	24.558	0.000	1.692	1.986
D1	1.9432	0.075	25.857	0.000	1.796	2.090
D2	1.9685	0.076	26.045	0.000	1.820	2.117
D3	2.0022	0.076	26.334	0.000	1.853	2.151
D4	2.0732	0.076	27.213	0.000	1.924	2.222
D5	2.0972	0.077	27.257	0.000	1.946	2.248
E1	2.1400	0.078	27.464	0.000	1.987	2.293
E2	2.2601	0.078	28.944	0.000	2.107	2.413
E3	2.2630	0.079	28.533	0.000	2.108	2.418
E4	2.3320	0.080	29.044	0.000	2.175	2.489
E5	2.3617	0.081	29.004	0.000	2.202	2.521
F1	2.2372	0.084	26.616	0.000	2.072	2.402
F2	2.3972	0.087	27.641	0.000	2.227	2.567
F3	2.4619	0.090	27.500	0.000	2.286	2.637
F4	2.5266	0.093	27.066	0.000	2.344	2.710
F5	2.6171	0.098	26.679	0.000	2.425	2.809
G1	2.5465	0.104	24.512	0.000	2.343	2.750
G2	2.4862	0.114	21.784	0.000	2.262	2.710
G3	2.7132	0.125	21.719	0.000	2.468	2.958
G4	2.4774	0.142	17.468	0.000	2.199	2.755
G5	2.6140	0.159	16.471	0.000	2.303	2.925
verification_status_Source Verified	0.1691	0.014	12.526	0.000	0.143	0.196
verification_status_Verified	0.1191	0.014	8.749	0.000	0.092	0.146
application_type_JOINT	-0.7698	0.175	-4.388	0.000	-1.114	-0.426
purpose_credit_card	-0.0245	0.013	-1.821	0.069	-0.051	0.002
purpose_home_improvement	0.0815	0.023	3.568	0.000	0.037	0.126
purpose_major_purchase	-0.0084	0.037	-0.225	0.822	-0.081	0.065
purpose_medical	0.0594	0.048	1.247	0.213	-0.034	0.153
purpose_moving	0.0097	0.056	0.174	0.862	-0.099	0.119
purpose_other	-0.0346	0.022	-1.555	0.120	-0.078	0.009
purpose_renewable_energy	0.1760	0.161	1.094	0.274	-0.139	0.491
purpose_small_business	0.4576	0.037	12.254	0.000	0.384	0.531
purpose_vacation	-0.0928	0.065	-1.437	0.151	-0.219	0.034
OWN	0.1232	0.018	6.709	0.000	0.087	0.159
RENT	0.2017	0.013	14.947	0.000	0.175	0.228
Average Loan	0.0711	0.016	4.582	0.000	0.041	0.101
High Loan	0.0567	0.024	2.383	0.017	0.010	0.103
Very Low	0.1603	0.012	13.403	0.000	0.137	0.184
Average	-0.1739	0.016	-10.572	0.000	-0.206	-0.142
Better Off	0.0124	0.187	0.066	0.947	-0.355	0.379

In [234]:

```
# to drop :Better off   ,purpose_major_purchase,purpose_medical ,purpose_moving ,purpose_ot
# purpose_vacation
```

In [ ]:

```
#X_train.drop(['Better off','purpose_major_purchase','purpose_medical','purpose_moving',\
#               'purpose_other','purpose_renewable_energy','purpose_vacation'], axis = 1, in
```

In [236]:

```
# Refit the model with the new set of features

logm1 = sm.GLM(y_train,(sm.add_constant(X_train)), family = sm.families.Binomial())
logm1.fit().summary()
```

Out[236]:

Generalized Linear Model Regression Results

Dep. Variable:	loan_status	**No. Observations:**	276653
Model:	GLM	**Df Residuals:**	276591
Model Family:	Binomial	**Df Model:**	61
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-1.2466e+05
Date:	Mon, 29 Aug 2022	**Deviance:**	2.4932e+05
Time:	15:12:26	**Pearson chi2:**	2.77e+05
No. Iterations:	6	**Pseudo R-squ. (CS):**	0.08463
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-3.7985	0.075	-50.893	0.000	-3.945	-3.652
term	0.4424	0.013	34.114	0.000	0.417	0.468
dti	0.0280	0.001	40.756	0.000	0.027	0.029
pub_rec	0.2090	0.027	7.815	0.000	0.157	0.261
revol_bal	-3.584e-06	7.16e-07	-5.010	0.000	-4.99e-06	-2.18e-06
revol_util	0.0020	0.000	7.929	0.000	0.001	0.002
total_acc	-0.0036	0.001	-6.518	0.000	-0.005	-0.003
mort_acc	-0.1548	0.013	-11.699	0.000	-0.181	-0.129
pub_rec_bankruptcies	-0.1874	0.030	-6.278	0.000	-0.246	-0.129
A2	0.3984	0.091	4.358	0.000	0.219	0.578
A3	0.5476	0.087	6.288	0.000	0.377	0.718
A4	0.7480	0.080	9.320	0.000	0.591	0.905
A5	0.8883	0.078	11.390	0.000	0.735	1.041
B1	1.0238	0.077	13.299	0.000	0.873	1.175
B2	1.0921	0.076	14.386	0.000	0.943	1.241
B3	1.1997	0.075	16.019	0.000	1.053	1.346
B4	1.2984	0.075	17.363	0.000	1.152	1.445
B5	1.4332	0.075	19.116	0.000	1.286	1.580
C1	1.5021	0.075	20.136	0.000	1.356	1.648
C2	1.6475	0.075	22.112	0.000	1.501	1.794
C3	1.7367	0.075	23.292	0.000	1.591	1.883
C4	1.7896	0.075	23.987	0.000	1.643	1.936

C5	1.8393	0.075	24.558	0.000	1.692	1.986
D1	1.9432	0.075	25.857	0.000	1.796	2.090
D2	1.9685	0.076	26.045	0.000	1.820	2.117
D3	2.0022	0.076	26.334	0.000	1.853	2.151
D4	2.0732	0.076	27.213	0.000	1.924	2.222
D5	2.0972	0.077	27.257	0.000	1.946	2.248
E1	2.1400	0.078	27.464	0.000	1.987	2.293
E2	2.2601	0.078	28.944	0.000	2.107	2.413
E3	2.2630	0.079	28.533	0.000	2.108	2.418
E4	2.3320	0.080	29.044	0.000	2.175	2.489
E5	2.3617	0.081	29.004	0.000	2.202	2.521
F1	2.2372	0.084	26.616	0.000	2.072	2.402
F2	2.3972	0.087	27.641	0.000	2.227	2.567
F3	2.4619	0.090	27.500	0.000	2.286	2.637
F4	2.5266	0.093	27.066	0.000	2.344	2.710
F5	2.6171	0.098	26.679	0.000	2.425	2.809
G1	2.5465	0.104	24.512	0.000	2.343	2.750
G2	2.4862	0.114	21.784	0.000	2.262	2.710
G3	2.7132	0.125	21.719	0.000	2.468	2.958
G4	2.4774	0.142	17.468	0.000	2.199	2.755
G5	2.6140	0.159	16.471	0.000	2.303	2.925
verification_status_Source Verified	0.1691	0.014	12.526	0.000	0.143	0.196
verification_status_Verified	0.1191	0.014	8.749	0.000	0.092	0.146
application_type_JOINT	-0.7698	0.175	-4.388	0.000	-1.114	-0.426
purpose_credit_card	-0.0245	0.013	-1.821	0.069	-0.051	0.002
purpose_home_improvement	0.0815	0.023	3.568	0.000	0.037	0.126
purpose_major_purchase	-0.0084	0.037	-0.225	0.822	-0.081	0.065
purpose_medical	0.0594	0.048	1.247	0.213	-0.034	0.153
purpose_moving	0.0097	0.056	0.174	0.862	-0.099	0.119
purpose_other	-0.0346	0.022	-1.555	0.120	-0.078	0.009
purpose_renewable_energy	0.1760	0.161	1.094	0.274	-0.139	0.491
purpose_small_business	0.4576	0.037	12.254	0.000	0.384	0.531
purpose_vacation	-0.0928	0.065	-1.437	0.151	-0.219	0.034
OWN	0.1232	0.018	6.709	0.000	0.087	0.159
RENT	0.2017	0.013	14.947	0.000	0.175	0.228
Average Loan	0.0711	0.016	4.582	0.000	0.041	0.101
High Loan	0.0567	0.024	2.383	0.017	0.010	0.103
Very Low	0.1603	0.012	13.403	0.000	0.137	0.184
Average	-0.1739	0.016	-10.572	0.000	-0.206	-0.142
Better Off	0.0124	0.187	0.066	0.947	-0.355	0.379

In [237]:

```
# to drop :purpose_credit_card
```

In [238]:

```
X_train.drop(['purpose_credit_card'], axis = 1, inplace = True)
```

In [239]:

```
# Refit the model with the new set of features

logm1 = sm.GLM(y_train,(sm.add_constant(X_train)), family = sm.families.Binomial())
logm1.fit().summary()
```

Out[239]:

Generalized Linear Model Regression Results

Dep. Variable:	loan_status	**No. Observations:**	276653
Model:	GLM	**Df Residuals:**	276592
Model Family:	Binomial	**Df Model:**	60
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-1.2466e+05
Date:	Mon, 29 Aug 2022	**Deviance:**	2.4932e+05
Time:	15:13:19	**Pearson chi2:**	2.77e+05
No. Iterations:	6	**Pseudo R-squ. (CS):**	0.08462
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-3.8060	0.075	-51.068	0.000	-3.952	-3.660
term	0.4418	0.013	34.078	0.000	0.416	0.467
dti	0.0280	0.001	40.783	0.000	0.027	0.029
pub_rec	0.2092	0.027	7.821	0.000	0.157	0.262
revol_bal	-3.674e-06	7.14e-07	-5.147	0.000	-5.07e-06	-2.28e-06
revol_util	0.0019	0.000	7.823	0.000	0.001	0.002
total_acc	-0.0036	0.001	-6.498	0.000	-0.005	-0.002
mort_acc	-0.1540	0.013	-11.648	0.000	-0.180	-0.128
pub_rec_bankruptcies	-0.1880	0.030	-6.299	0.000	-0.246	-0.130
A2	0.3983	0.091	4.358	0.000	0.219	0.578
A3	0.5483	0.087	6.296	0.000	0.378	0.719
A4	0.7493	0.080	9.336	0.000	0.592	0.907
A5	0.8896	0.078	11.407	0.000	0.737	1.042
B1	1.0250	0.077	13.316	0.000	0.874	1.176
B2	1.0939	0.076	14.412	0.000	0.945	1.243
B3	1.2021	0.075	16.055	0.000	1.055	1.349
B4	1.3012	0.075	17.405	0.000	1.155	1.448
B5	1.4362	0.075	19.161	0.000	1.289	1.583
C1	1.5055	0.075	20.189	0.000	1.359	1.652
C2	1.6513	0.074	22.172	0.000	1.505	1.797
C3	1.7408	0.075	23.357	0.000	1.595	1.887
C4	1.7941	0.075	24.060	0.000	1.648	1.940

C5	1.8439	0.075	24.634	0.000	1.697	1.991
D1	1.9481	0.075	25.939	0.000	1.801	2.095
D2	1.9736	0.076	26.132	0.000	1.826	2.122
D3	2.0074	0.076	26.423	0.000	1.859	2.156
D4	2.0788	0.076	27.309	0.000	1.930	2.228
D5	2.1029	0.077	27.355	0.000	1.952	2.254
E1	2.1460	0.078	27.564	0.000	1.993	2.299
E2	2.2664	0.078	29.053	0.000	2.113	2.419
E3	2.2694	0.079	28.642	0.000	2.114	2.425
E4	2.3382	0.080	29.146	0.000	2.181	2.495
E5	2.3683	0.081	29.113	0.000	2.209	2.528
F1	2.2441	0.084	26.725	0.000	2.080	2.409
F2	2.4038	0.087	27.742	0.000	2.234	2.574
F3	2.4687	0.089	27.600	0.000	2.293	2.644
F4	2.5339	0.093	27.170	0.000	2.351	2.717
F5	2.6246	0.098	26.779	0.000	2.432	2.817
G1	2.5534	0.104	24.595	0.000	2.350	2.757
G2	2.4936	0.114	21.862	0.000	2.270	2.717
G3	2.7208	0.125	21.792	0.000	2.476	2.965
G4	2.4847	0.142	17.526	0.000	2.207	2.763
G5	2.6217	0.159	16.526	0.000	2.311	2.933
verification_status_Source Verified	0.1693	0.014	12.543	0.000	0.143	0.196
verification_status_Verified	0.1193	0.014	8.767	0.000	0.093	0.146
application_type_JOINT	-0.7703	0.175	-4.390	0.000	-1.114	-0.426
purpose_home_improvement	0.0864	0.023	3.808	0.000	0.042	0.131
purpose_major_purchase	-0.0034	0.037	-0.092	0.927	-0.076	0.069
purpose_medical	0.0640	0.048	1.345	0.179	-0.029	0.157
purpose_moving	0.0140	0.056	0.252	0.801	-0.095	0.123
purpose_other	-0.0301	0.022	-1.361	0.174	-0.074	0.013
purpose_renewable_energy	0.1806	0.161	1.123	0.262	-0.135	0.496
purpose_small_business	0.4617	0.037	12.383	0.000	0.389	0.535
purpose_vacation	-0.0881	0.065	-1.366	0.172	-0.215	0.038
OWN	0.1229	0.018	6.688	0.000	0.087	0.159
RENT	0.2017	0.013	14.942	0.000	0.175	0.228
Average Loan	0.0711	0.016	4.586	0.000	0.041	0.102
High Loan	0.0562	0.024	2.363	0.018	0.010	0.103
Very Low	0.1598	0.012	13.366	0.000	0.136	0.183
Average	-0.1737	0.016	-10.561	0.000	-0.206	-0.141
Better Off	0.0129	0.187	0.069	0.945	-0.354	0.380

In [240]:

```
# Make a VIF dataframe for all the variables present

vif = pd.DataFrame()
vif['Features'] = X_train.columns
vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif.head(10)
```

Out[240]:

	Features	VIF
4	revol_util	8.41
5	total_acc	7.90
1	dti	7.14
2	pub_rec	4.83
3	revol_bal	4.73
7	pub_rec_bankruptcies	4.62
6	mort_acc	4.34
54	RENT	2.67
43	verification_status_Verified	2.43
42	verification_status_Source Verified	2.19

In [241]:

```
X_train.drop(['revol_util','total_acc','dti'], axis = 1, inplace = True)
```

In [242]:

```
# Refit the model with the new set of features

logm1 = sm.GLM(y_train,(sm.add_constant(X_train)), family = sm.families.Binomial())
logm1.fit().summary()
```

Out[242]:

Generalized Linear Model Regression Results

Dep. Variable:	loan_status	**No. Observations:**	276653
Model:	GLM	**Df Residuals:**	276595
Model Family:	Binomial	**Df Model:**	57
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-1.2557e+05
Date:	Mon, 29 Aug 2022	**Deviance:**	2.5113e+05
Time:	15:18:14	**Pearson chi2:**	2.77e+05
No. Iterations:	6	**Pseudo R-squ. (CS):**	0.07860
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-3.4891	0.073	-47.757	0.000	-3.632	-3.346
term	0.4121	0.013	32.266	0.000	0.387	0.437
pub_rec	0.1779	0.027	6.692	0.000	0.126	0.230
revol_bal	4.332e-06	6.37e-07	6.799	0.000	3.08e-06	5.58e-06
mort_acc	-0.1958	0.013	-15.095	0.000	-0.221	-0.170
pub_rec_bankruptcies	-0.1847	0.030	-6.224	0.000	-0.243	-0.127
A2	0.4255	0.091	4.659	0.000	0.246	0.605
A3	0.5987	0.087	6.883	0.000	0.428	0.769
A4	0.8059	0.080	10.059	0.000	0.649	0.963
A5	0.9610	0.078	12.350	0.000	0.809	1.114
B1	1.1114	0.077	14.478	0.000	0.961	1.262
B2	1.1912	0.076	15.749	0.000	1.043	1.339
B3	1.3147	0.075	17.635	0.000	1.169	1.461
B4	1.4226	0.074	19.120	0.000	1.277	1.568
B5	1.5691	0.075	21.046	0.000	1.423	1.715
C1	1.6473	0.074	22.217	0.000	1.502	1.793
C2	1.8019	0.074	24.347	0.000	1.657	1.947
C3	1.8990	0.074	25.647	0.000	1.754	2.044
C4	1.9617	0.074	26.491	0.000	1.817	2.107
C5	2.0206	0.074	27.190	0.000	1.875	2.166
D1	2.1335	0.075	28.632	0.000	1.987	2.280
D2	2.1649	0.075	28.900	0.000	2.018	2.312

D3	2.2001	0.075	29.195	0.000	2.052	2.348
D4	2.2757	0.075	30.155	0.000	2.128	2.424
D5	2.3040	0.076	30.231	0.000	2.155	2.453
E1	2.3539	0.077	30.502	0.000	2.203	2.505
E2	2.4761	0.077	32.031	0.000	2.325	2.628
E3	2.4917	0.078	31.745	0.000	2.338	2.646
E4	2.5556	0.079	32.158	0.000	2.400	2.711
E5	2.5813	0.081	32.028	0.000	2.423	2.739
F1	2.4574	0.083	29.541	0.000	2.294	2.620
F2	2.6319	0.086	30.666	0.000	2.464	2.800
F3	2.6901	0.089	30.361	0.000	2.516	2.864
F4	2.7463	0.092	29.714	0.000	2.565	2.927
F5	2.8488	0.097	29.325	0.000	2.658	3.039
G1	2.7695	0.103	26.882	0.000	2.568	2.971
G2	2.7318	0.113	24.127	0.000	2.510	2.954
G3	2.9396	0.124	23.704	0.000	2.697	3.183
G4	2.7156	0.141	19.266	0.000	2.439	2.992
G5	2.8612	0.158	18.159	0.000	2.552	3.170
verification_status_Source Verified	0.1889	0.013	14.049	0.000	0.163	0.215
verification_status_Verified	0.1626	0.014	12.024	0.000	0.136	0.189
application_type_JOINT	-0.6609	0.175	-3.787	0.000	-1.003	-0.319
purpose_home_improvement	0.0193	0.023	0.855	0.392	-0.025	0.063
purpose_major_purchase	-0.0877	0.037	-2.374	0.018	-0.160	-0.015
purpose_medical	0.0262	0.047	0.552	0.581	-0.067	0.119
purpose_moving	-0.0487	0.055	-0.881	0.379	-0.157	0.060
purpose_other	-0.0840	0.022	-3.819	0.000	-0.127	-0.041
purpose_renewable_energy	0.0841	0.160	0.524	0.600	-0.230	0.399
purpose_small_business	0.3232	0.037	8.747	0.000	0.251	0.396
purpose_vacation	-0.1284	0.064	-1.997	0.046	-0.254	-0.002
OWN	0.1185	0.018	6.484	0.000	0.083	0.154
RENT	0.1757	0.013	13.104	0.000	0.149	0.202
Average Loan	0.0524	0.015	3.402	0.001	0.022	0.083
High Loan	0.0232	0.024	0.982	0.326	-0.023	0.070
Very Low	0.2293	0.012	19.578	0.000	0.206	0.252
Average	-0.2867	0.016	-17.877	0.000	-0.318	-0.255
Better Off	-0.3178	0.187	-1.701	0.089	-0.684	0.048

In [243]:

```
# to drop :purpose_home_improvement,High Loan
```

In [244]:

```
X_train.drop(['purpose_home_improvement','High Loan'], axis = 1, inplace = True)
```

In [245]:

```
# Refit the model with the new set of features

logm1 = sm.GLM(y_train,(sm.add_constant(X_train)), family = sm.families.Binomial())
logm1.fit().summary()
```

Out[245]:

Generalized Linear Model Regression Results

Dep. Variable:	loan_status	**No. Observations:**	276653
Model:	GLM	**Df Residuals:**	276597
Model Family:	Binomial	**Df Model:**	55
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-1.2557e+05
Date:	Mon, 29 Aug 2022	**Deviance:**	2.5113e+05
Time:	15:18:25	**Pearson chi2:**	2.77e+05
No. Iterations:	6	**Pseudo R-squ. (CS):**	0.07860
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-3.4878	0.073	-47.806	0.000	-3.631	-3.345
term	0.4130	0.013	32.429	0.000	0.388	0.438
pub_rec	0.1779	0.027	6.691	0.000	0.126	0.230
revol_bal	4.357e-06	6.28e-07	6.941	0.000	3.13e-06	5.59e-06
mort_acc	-0.1954	0.013	-15.069	0.000	-0.221	-0.170
pub_rec_bankruptcies	-0.1851	0.030	-6.239	0.000	-0.243	-0.127
A2	0.4255	0.091	4.659	0.000	0.247	0.605
A3	0.5987	0.087	6.884	0.000	0.428	0.769
A4	0.8061	0.080	10.062	0.000	0.649	0.963
A5	0.9612	0.078	12.354	0.000	0.809	1.114
B1	1.1115	0.077	14.481	0.000	0.961	1.262
B2	1.1914	0.076	15.754	0.000	1.043	1.340
B3	1.3151	0.075	17.642	0.000	1.169	1.461
B4	1.4230	0.074	19.127	0.000	1.277	1.569
B5	1.5695	0.075	21.053	0.000	1.423	1.716
C1	1.6479	0.074	22.230	0.000	1.503	1.793
C2	1.8028	0.074	24.363	0.000	1.658	1.948
C3	1.8999	0.074	25.664	0.000	1.755	2.045
C4	1.9625	0.074	26.506	0.000	1.817	2.108
C5	2.0216	0.074	27.210	0.000	1.876	2.167
D1	2.1346	0.074	28.654	0.000	1.989	2.281
D2	2.1660	0.075	28.922	0.000	2.019	2.313

D3	2.2014	0.075	29.221	0.000	2.054	2.349
D4	2.2775	0.075	30.192	0.000	2.130	2.425
D5	2.3057	0.076	30.267	0.000	2.156	2.455
E1	2.3559	0.077	30.543	0.000	2.205	2.507
E2	2.4784	0.077	32.079	0.000	2.327	2.630
E3	2.4938	0.078	31.790	0.000	2.340	2.648
E4	2.5579	0.079	32.204	0.000	2.402	2.714
E5	2.5836	0.081	32.074	0.000	2.426	2.742
F1	2.4602	0.083	29.598	0.000	2.297	2.623
F2	2.6346	0.086	30.716	0.000	2.466	2.803
F3	2.6927	0.089	30.408	0.000	2.519	2.866
F4	2.7491	0.092	29.759	0.000	2.568	2.930
F5	2.8521	0.097	29.379	0.000	2.662	3.042
G1	2.7725	0.103	26.925	0.000	2.571	2.974
G2	2.7348	0.113	24.164	0.000	2.513	2.957
G3	2.9431	0.124	23.741	0.000	2.700	3.186
G4	2.7191	0.141	19.296	0.000	2.443	2.995
G5	2.8641	0.158	18.181	0.000	2.555	3.173
verification_status_Source Verified	0.1894	0.013	14.101	0.000	0.163	0.216
verification_status_Verified	0.1643	0.013	12.256	0.000	0.138	0.191
application_type_JOINT	-0.6568	0.174	-3.765	0.000	-0.999	-0.315
purpose_major_purchase	-0.0890	0.037	-2.412	0.016	-0.161	-0.017
purpose_medical	0.0236	0.047	0.498	0.618	-0.069	0.116
purpose_moving	-0.0507	0.055	-0.918	0.358	-0.159	0.058
purpose_other	-0.0862	0.022	-3.933	0.000	-0.129	-0.043
purpose_renewable_energy	0.0814	0.160	0.507	0.612	-0.233	0.396
purpose_small_business	0.3218	0.037	8.717	0.000	0.249	0.394
purpose_vacation	-0.1307	0.064	-2.034	0.042	-0.257	-0.005
OWN	0.1184	0.018	6.483	0.000	0.083	0.154
RENT	0.1740	0.013	13.088	0.000	0.148	0.200
Average Loan	0.0480	0.015	3.252	0.001	0.019	0.077
Very Low	0.2278	0.012	19.577	0.000	0.205	0.251
Average	-0.2819	0.015	-18.267	0.000	-0.312	-0.252
Better Off	-0.3069	0.187	-1.645	0.100	-0.673	0.059

In [246]:

```
# Make a VIF dataframe for all the variables present

vif = pd.DataFrame()
vif['Features'] = X_train.columns
vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif.head(10)
```

Out[246]:

	Features	VIF
1	pub_rec	4.82
4	pub_rec_bankruptcies	4.62
3	mort_acc	4.04
2	revol_bal	3.57
50	RENT	2.63
40	verification_status_Verified	2.36
39	verification_status_Source Verified	2.17
0	term	1.88
11	B3	1.67
52	Very Low	1.66

Model4 Evaluation:

Now, both the p-values and VIFs seem decent enough for all the variables. So let's go ahead and make predictions using this final set of features.

In [247]:

```
# Use 'predict' to predict the probabilities on the train set
res = logm1.fit()
y_train_pred = res.predict(sm.add_constant(X_train))
y_train_pred
```

Out[247]:

```
69562     0.123329
18239     0.062432
316733    0.151587
285026    0.173839
352685    0.166136
            ...
340502    0.125612
254332    0.037904
211195    0.226324
56189     0.070861
301179    0.045452
Length: 276653, dtype: float64
```

In [248]:

```
# Reshaping it into an array

y_train_pred = y_train_pred.values.reshape(-1)
y_train_pred
```

Out[248]:

```
array([0.12332917, 0.0624316 , 0.15158718, ..., 0.22632394, 0.07086149,
       0.04545202])
```

Creating a dataframe with the actual conversion flag and the predicted probabilities

In [249]:

```
# Create a new dataframe containing the actual conversion flag and the probabilities predic

y_train_pred_final = pd.DataFrame({'Loan_Status':y_train.values, 'Loan_Status_Prob':y_train
y_train_pred_final.head()
```

Out[249]:

	Loan_Status	Loan_Status_Prob
0	0	0.123329
1	0	0.062432
2	0	0.151587
3	1	0.173839
4	0	0.166136

Creating new column 'Predicted' with 1 if Prob > 0.5 else 0

In [250]:

```
y_train_pred_final['Predicted'] = y_train_pred_final.Loan_Status_Prob.map(lambda x: 1 if x

# Let's see the head
y_train_pred_final.head()
y_train_pred_final.Loan_Status, y_train_pred_final.Predicted
```

Out[250]:

```
(0         0
 1         0
 2         0
 3         1
 4         0
          ..
 276648    0
 276649    0
 276650    0
 276651    0
 276652    0
 Name: Loan_Status, Length: 276653, dtype: int64,
 0         0
 1         0
 2         0
 3         0
 4         0
          ..
 276648    0
 276649    0
 276650    0
 276651    0
 276652    0
 Name: Predicted, Length: 276653, dtype: int64)
```

Insights:

* Now that we have the probabilities and have also made conversion predictions using them, it's time to evaluate the model.

In [251]:

```
# Import metrics from sklearn for evaluation
# Let's evaluate the all metrics as well
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score,confusion_

print(f"Accuracy : {accuracy_score(y_train_pred_final.Loan_Status, y_train_pred_final.Predi
print(f"recall_score : {recall_score(y_train_pred_final.Loan_Status, y_train_pred_final.Pre
print(f"precision_score : {precision_score(y_train_pred_final.Loan_Status, y_train_pred_fin
print(f"f1_score : {f1_score(y_train_pred_final.Loan_Status, y_train_pred_final.Predicted )
print(f"AUC score : {metrics.roc_auc_score(y_train_pred_final.Loan_Status, y_train_pred_fin
print(f"confusion_matrix :")
print(confusion_matrix(y_train_pred_final.Loan_Status, y_train_pred_final.Predicted ))
```

```
Accuracy : 80.48602400841487%
recall_score : 3.387798760696371%
precision_score : 53.46332945285215%
f1_score : 6.371834894207423%
AUC score : 51.33445883977119%
confusion_matrix :
[[220830   1599]
 [ 52387   1837]]
```

Finding the Optimal Cutoff:

* Now 0.5 was just arbitrary to loosely check the model performace. But in order to get good results, you need
to optimise the threshold. So first let's plot an ROC curve to see what AUC we get.

In [252]:

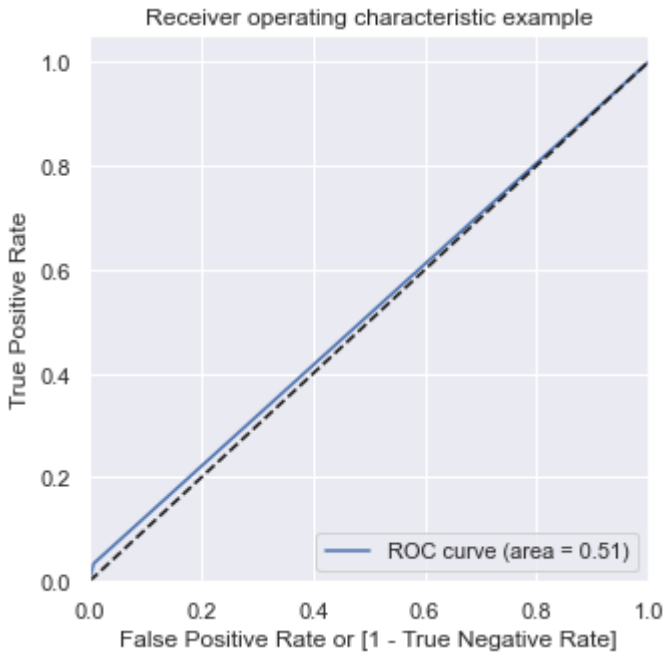
```
# ROC function

def draw_roc( actual, probs ):
    fpr, tpr, thresholds = metrics.roc_curve( actual, probs,
                                              drop_intermediate = False )
    auc_score = metrics.roc_auc_score( actual, probs )
    plt.figure(figsize=(5, 5))
    plt.plot( fpr, tpr, label='ROC curve (area = %0.2f)' % auc_score )
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()

    return None
```

In [253]:

```
fpr, tpr, thresholds = metrics.roc_curve( y_train_pred_final.Loan_Status, y_train_pred_fina
                                          drop_intermediate = False )
```

In [254]:

```
# Call the ROC function

draw_roc(y_train_pred_final.Loan_Status, y_train_pred_final.Predicted)
```



Insights:

```
    * The area under the curve of the ROC is 0.51 which is not that good. Let's al
so check the sensitivity and
      specificity tradeoff to find the optimal cutoff point.
```

In [255]:

```
# Let's create columns with different probability cutoffs

numbers = [float(x)/10 for x in range(10)]
for i in numbers:
    y_train_pred_final[i]= y_train_pred_final.Loan_Status_Prob.map(lambda x: 1 if x > i els
y_train_pred_final.head()
```

Out[255]:

	Loan_Status	Loan_Status_Prob	Predicted	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0	0	0.123329	0	1	1	0	0	0	0	0	0	0	0
1	0	0.062432	0	1	0	0	0	0	0	0	0	0	0
2	0	0.151587	0	1	1	0	0	0	0	0	0	0	0
3	1	0.173839	0	1	1	0	0	0	0	0	0	0	0
4	0	0.166136	0	1	1	0	0	0	0	0	0	0	0

In [256]:

```
y_train_pred_final.shape
```

Out[256]:

```
(276653, 13)
```

In [257]:

```
y_train_pred_final
```

Out[257]:

	Loan_Status	Loan_Status_Prob	Predicted	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0
0	0	0.123329	0	1	1	0	0	0	0	0	0	0	
1	0	0.062432	0	1	0	0	0	0	0	0	0	0	
2	0	0.151587	0	1	1	0	0	0	0	0	0	0	
3	1	0.173839	0	1	1	0	0	0	0	0	0	0	
4	0	0.166136	0	1	1	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
276648	0	0.125612	0	1	1	0	0	0	0	0	0	0	
276649	0	0.037904	0	1	0	0	0	0	0	0	0	0	
276650	0	0.226324	0	1	1	1	0	0	0	0	0	0	
276651	0	0.070861	0	1	0	0	0	0	0	0	0	0	
276652	0	0.045452	0	1	0	0	0	0	0	0	0	0	

276653 rows × 13 columns

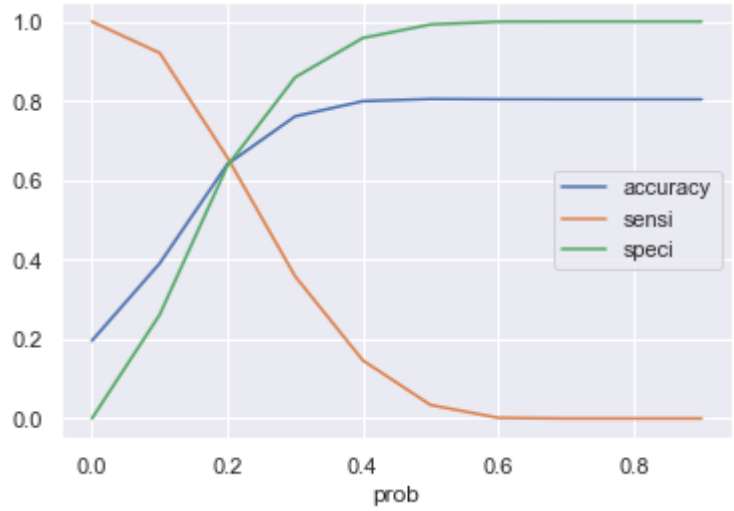
In [258]:

```
# Let's create a dataframe to see the values of accuracy, sensitivity, and specificity at d

cutoff_df = pd.DataFrame( columns = ['prob','accuracy','sensi','speci'])
from sklearn.metrics import confusion_matrix

# TP = confusion[1,1] # true positive
# TN = confusion[0,0] # true negatives
# FP = confusion[0,1] # false positives
# FN = confusion[1,0] # false negatives

num = [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
for i in num:
    cm1 = metrics.confusion_matrix(y_train_pred_final.Loan_Status, y_train_pred_final[i] )
    total1=sum(sum(cm1))
    accuracy = (cm1[0,0]+cm1[1,1])/total1

    speci = cm1[0,0]/(cm1[0,0]+cm1[0,1])
    sensi = cm1[1,1]/(cm1[1,0]+cm1[1,1])
    cutoff_df.loc[i] =[ i ,accuracy,sensi,speci]
print(cutoff_df)
```

```
     prob  accuracy     sensi     speci
0.0   0.0  0.196000  1.000000  0.000000
0.1   0.1  0.390760  0.921050  0.261486
0.2   0.2  0.640842  0.657310  0.636828
0.3   0.3  0.761358  0.358255  0.859627
0.4   0.4  0.799518  0.146319  0.958755
0.5   0.5  0.804860  0.033878  0.992811
0.6   0.6  0.804130  0.001531  0.999789
0.7   0.7  0.804000  0.000000  1.000000
0.8   0.8  0.804000  0.000000  1.000000
0.9   0.9  0.804000  0.000000  1.000000
```

In [259]:

```
# Let's plot it as well

cutoff_df.plot.line(x='prob', y=['accuracy','sensi','speci'])
plt.show()
```



Insight:

* As you can see that around 0.20, you get the optimal values of the three metrics. So let's choose 0.2 as our
cutoff now.

In [260]:

```
y_train_pred_final['final_predicted'] = y_train_pred_final.Loan_Status_Prob.map( lambda x:

y_train_pred_final.head()
```

Out[260]:

	Loan_Status	Loan_Status_Prob	Predicted	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	fir
0	0	0.123329	0	1	1	0	0	0	0	0	0	0	0	
1	0	0.062432	0	1	0	0	0	0	0	0	0	0	0	
2	0	0.151587	0	1	1	0	0	0	0	0	0	0	0	
3	1	0.173839	0	1	1	0	0	0	0	0	0	0	0	
4	0	0.166136	0	1	1	0	0	0	0	0	0	0	0	

In [261]:

```
# Import metrics from sklearn for evaluation
# Let's evaluate the all metrics as well
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score,confusion_

print(f"Accuracy : {accuracy_score(y_train_pred_final.Loan_Status, y_train_pred_final.final
print(f"recall_score : {recall_score(y_train_pred_final.Loan_Status, y_train_pred_final.fin
print(f"precision_score : {precision_score(y_train_pred_final.Loan_Status, y_train_pred_fin
print(f"f1_score : {f1_score(y_train_pred_final.Loan_Status, y_train_pred_final.final_predi
print(f"AUC score : {metrics.roc_auc_score(y_train_pred_final.Loan_Status, y_train_pred_fin
print(f"confusion_matrix :")
print(confusion_matrix(y_train_pred_final.Loan_Status, y_train_pred_final.final_predicted )
```

```
Accuracy : 64.08424994487679%
recall_score : 65.73104160519327%
precision_score : 30.614488670526192%
f1_score : 41.77302720251281%
AUC score : 64.7069173830785%
confusion_matrix :
[[141649  80780]
 [ 18582  35642]]
```

Making Predictions on the Test Set:

In [262]:

```
X_train.columns
```

Out[262]:

```
Index(['term', 'pub_rec', 'revol_bal', 'mort_acc', 'pub_rec_bankruptcies',
       'A2', 'A3', 'A4', 'A5', 'B1', 'B2', 'B3', 'B4', 'B5', 'C1', 'C2', 'C
3',
       'C4', 'C5', 'D1', 'D2', 'D3', 'D4', 'D5', 'E1', 'E2', 'E3', 'E4', 'E
5',
       'F1', 'F2', 'F3', 'F4', 'F5', 'G1', 'G2', 'G3', 'G4', 'G5',
       'verification_status_Source Verified', 'verification_status_Verifie
d',
       'application_type_JOINT', 'purpose_major_purchase', 'purpose_medica
l',
       'purpose_moving', 'purpose_other', 'purpose_renewable_energy',
       'purpose_small_business', 'purpose_vacation', 'OWN', 'RENT',
       'Average Loan', 'Very Low', 'Average', 'Better Off'],
      dtype='object')
```

In [263]:

```
X_test.columns
```

Out[263]:

```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'annual_inc', 'dti',
       'open_acc', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc',
       'mort_acc', 'pub_rec_bankruptcies', 'A2', 'A3', 'A4', 'A5', 'B1', 'B
2',
       'B3', 'B4', 'B5', 'C1', 'C2', 'C3', 'C4', 'C5', 'D1', 'D2', 'D3', 'D
4',
       'D5', 'E1', 'E2', 'E3', 'E4', 'E5', 'F1', 'F2', 'F3', 'F4', 'F5', 'G
1',
       'G2', 'G3', 'G4', 'G5', 'verification_status_Source Verified',
       'verification_status_Verified', 'application_type_INDIVIDUAL',
       'application_type_JOINT', 'initial_list_status_w',
       'purpose_credit_card', 'purpose_debt_consolidation',
       'purpose_educational', 'purpose_home_improvement', 'purpose_house',
       'purpose_major_purchase', 'purpose_medical', 'purpose_moving',
       'purpose_other', 'purpose_renewable_energy', 'purpose_small_busines
s',
       'purpose_vacation', 'purpose_wedding', 'OTHER', 'OWN', 'RENT', '0511
3',
       '11650', '22690', '29597', '30723', '48052', '70466', '86630', '9370
0',
       'earliest_cr_year', 'Lower Loan', 'Average Loan', 'High Loan',
       'Very Low', 'Low', 'Average', 'Better Off', 'OTHER'],
      dtype='object')
```

In [264]:

```
set(X_test.columns) - set(X_train.columns)
```

Out[264]:

```
{'05113',
 '11650',
 '22690',
 '29597',
 '30723',
 '48052',
 '70466',
 '86630',
 '93700',
 'High Loan',
 'Low',
 'Lower Loan',
 'OTHER',
 'annual_inc',
 'application_type_INDIVIDUAL',
 'dti',
 'earliest_cr_year',
 'initial_list_status_w',
 'installment',
 'int_rate',
 'loan_amnt',
 'open_acc',
 'purpose_credit_card',
 'purpose_debt_consolidation',
 'purpose_educational',
 'purpose_home_improvement',
 'purpose_house',
 'purpose_wedding',
 'revol_util',
 'total_acc'}
```

In [273]:

```
X_test.drop(['05113',
 '11650',
 '22690',
 '29597',
 '30723',
 '48052',
 '70466',
 '86630',
 '93700',
 'High Loan',
 'Low',
 'Lower Loan',
 'OTHER',
 'annual_inc',
 'application_type_INDIVIDUAL',
 'dti',
 'earliest_cr_year',
 'initial_list_status_w',
 'installment',
 'int_rate',
 'loan_amnt',
 'open_acc',
 'purpose_credit_card',
 'purpose_debt_consolidation',
 'purpose_educational',
 'purpose_home_improvement',
 'purpose_house',
 'purpose_wedding',
 'revol_util',
 'total_acc'], axis = 1, inplace = True)
```

In [274]:

```
col = X_test.columns
col
```

Out[274]:

```
Index(['term', 'pub_rec', 'revol_bal', 'mort_acc', 'pub_rec_bankruptcies',
       'A2', 'A3', 'A4', 'A5', 'B1', 'B2', 'B3', 'B4', 'B5', 'C1', 'C2', 'C
3',
       'C4', 'C5', 'D1', 'D2', 'D3', 'D4', 'D5', 'E1', 'E2', 'E3', 'E4', 'E
5',
       'F1', 'F2', 'F3', 'F4', 'F5', 'G1', 'G2', 'G3', 'G4', 'G5',
       'verification_status_Source Verified', 'verification_status_Verifie
d',
       'application_type_JOINT', 'purpose_major_purchase', 'purpose_medica
l',
       'purpose_moving', 'purpose_other', 'purpose_renewable_energy',
       'purpose_small_business', 'purpose_vacation', 'OWN', 'RENT',
       'Average Loan', 'Very Low', 'Average', 'Better Off'],
      dtype='object')
```

In [275]:

```
# Add a constant to X_test

X_test_sm = sm.add_constant(X_test[col])
```

In [276]:

```
# Check X_test_sm

X_test_sm
```

Out[276]:

	const	term	pub_rec	revol_bal	mort_acc	pub_rec_bankruptcies	A2	A3	A4	A5	...
346397	1.0	0	0	24499.0	1	0	0	0	0	0	...
78418	1.0	0	1	8381.0	0	1	0	0	0	0	...
356956	1.0	0	0	7721.0	1	0	0	0	0	0	...
327009	1.0	0	0	44810.0	1	0	0	0	0	0	...
124321	1.0	0	0	4901.0	1	0	0	0	0	0	...
...	...	...	...	...	...	...	...	...	...	...	...
139477	1.0	0	0	10425.0	1	0	0	0	1	0	...
371079	1.0	0	0	2298.0	1	0	0	0	0	0	...
303152	1.0	0	0	8966.0	1	0	0	0	0	0	...
108628	1.0	0	0	51680.0	1	0	0	0	0	0	...
129767	1.0	1	0	20068.0	1	0	0	0	0	0	...

118566 rows × 56 columns

In [277]:

```
# Make predictions on the test set and store it in the variable 'y_test_pred'

y_test_pred = res.predict(sm.add_constant(X_test))
```

In [278]:

```
y_test_pred
```

Out[278]:

```
346397    0.238182
78418     0.221999
356956    0.221727
327009    0.084013
124321    0.212949
            ...
139477    0.074208
371079    0.107386
303152    0.073604
108628    0.126679
129767    0.227895
Length: 118566, dtype: float64
```

In [279]:

```
# Converting y_pred to a dataframe

y_pred_1 = pd.DataFrame(y_test_pred)
y_pred_1
```

Out[279]:

	0
346397	0.238182
78418	0.221999
356956	0.221727
327009	0.084013
124321	0.212949
...	...
139477	0.074208
371079	0.107386
303152	0.073604
108628	0.126679
129767	0.227895

118566 rows × 1 columns

In [280]:

```
# Converting y_test to dataframe

y_test_df = pd.DataFrame(y_test)
```

In [281]:

```
# Remove index for both dataframes to append them side by side

y_pred_1.reset_index(drop=True, inplace=True)
y_test_df.reset_index(drop=True, inplace=True)
```

In [282]:

```
# Append y_test_df and y_pred_1

y_pred_final = pd.concat([y_test_df, y_pred_1],axis=1)
```

In [283]:

```
# Check 'y_pred_final'

y_pred_final.head()
```

Out[283]:

	loan_status	0
0	0	0.238182
1	0	0.221999
2	0	0.221727
3	0	0.084013
4	0	0.212949

In [284]:

```
# Rename the column

y_pred_final= y_pred_final.rename(columns = {0 : 'Loan_Status_Prob'})
```

In [285]:

```
# Let's see the head of y_pred_final

y_pred_final.head()
```

Out[285]:

	loan_status	Loan_Status_Prob
0	0	0.238182
1	0	0.221999
2	0	0.221727
3	0	0.084013
4	0	0.212949

In [286]:

```
# Make predictions on the test set using 0.2 as the cutoff

y_pred_final['final_predicted'] = y_pred_final.Loan_Status_Prob.map(lambda x: 1 if x > 0.2
```

In [287]:

```
# Check y_pred_final

y_pred_final.head()
```

Out[287]:

	loan_status	Loan_Status_Prob	final_predicted
0	0	0.238182	1
1	0	0.221999	1
2	0	0.221727	1
3	0	0.084013	0
4	0	0.212949	1

In [288]:

```
# Let's check the overall accuracy

metrics.accuracy_score(y_pred_final['loan_status'], y_pred_final.final_predicted)
```

Out[288]:

```
0.640284735927669
```

In [289]:

```
# Import metrics from sklearn for evaluation
# Let's evaluate the all metrics as well
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score,confusion_

print(f"Accuracy : {accuracy_score(y_pred_final['loan_status'],  y_pred_final.final_predict
print(f"recall_score : {recall_score(y_pred_final['loan_status'],  y_pred_final.final_predi
print(f"precision_score : {precision_score(y_pred_final['loan_status'],  y_pred_final.final
print(f"f1_score : {f1_score(y_pred_final['loan_status'],  y_pred_final.final_predicted )*1
print(f"AUC score : {metrics.roc_auc_score(y_pred_final['loan_status'], y_pred_final.final_
print(f"confusion_matrix :")
print(confusion_matrix(y_pred_final['loan_status'],  y_pred_final.final_predicted))
```

```
Accuracy : 64.0284735927669%
recall_score : 66.50929224430233%
precision_score : 30.780843414178733%
f1_score : 42.084679938078814%
AUC score : 64.96552187136128%
confusion_matrix :
[[60420 34847]
 [ 7803 15496]]
```

In [290]:

```
# Checking the classification_report

from sklearn.metrics import classification_report

print(f"{classification_report(y_pred_final['loan_status'], y_pred_final.final_predicted )}
```

```
              precision    recall  f1-score   support

           0       0.89      0.63      0.74     95267
           1       0.31      0.67      0.42     23299

    accuracy                           0.64    118566
   macro avg       0.60      0.65      0.58    118566
weighted avg       0.77      0.64      0.68    118566
```

Insights from Model 4:

```
    * Defaulters/Irresponsible Business or Customers: For our model 3 , we are get
ting fi-score as 42 % to check if
      the credit line can be extended to the business/customers who have a higher
 chances of defaulting/ not repaying
      the loan/credit taken. 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.
    * Honest/Responsible Business or Customers :For our model 2 ( with balancing w
eights) , we are getting fi-score as
      74 % to check if the credit line can be extended to the business/customers w
ho have a higher chances of fully
      repaying the loan/credit taken. This is important as we can lose out on an o
pportunity to finance more supply
      chains and earn interest in it.
    * We can conclude from the above metrics and f1-score that the Model 4 is the
 worst performer amongst all the 4
      models in terms of Precision and Recal but it's have a best Recall score of
 67 %.
```

In [ ]:

TradeOff Questions:

How can we make sure that our model can detect real defaulters and there are fewer false positives?

```
- As we have seen in our first two models, by keeping precion score higher and the
n assigning priority to fi score and
  then recall, we can make sure that real dafualters ( TP ) and low False positive
s ( FP ) in demominator of Precision.
  This is important as we can lose out on an opportunity to finance more supply ch
ains and earn interest in it.
- Also, we have to handle treat missing values and outliers after train test spli
t. Missing values can be treated on
  all three types of data (train, cv and test but outliers needs to get handled on
ly on train data).
- Even, proper balancing of class weights and findling best hyperparameter can be
 though as one of the key
  consideration.
```

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.¶

```
- We should be having a low recall rate
- Then the priority should be considering less gap between precison and recall and
having a more AUC (area under the
  curve score)
```

In [ ]:

Recommendations: ¶

- As the NPA issues are rising for the Lending Clubs/Banks, it's necessary to think in terms of not loosing money in bad loans, Recall is the parameter which should be more focused. The term amount, purpose for the loan take (if it's small business then there's higher possibility of defaulting). Apart from that, if there are derogatory public records in form of earlier Lending Clubs/Banksruptcies, then alos, the Lending Clubs/Banks should focus on not giving any loan to such customers/business.
- In addition to this, the Lending Clubs/Banks assigned subgrades are also equally important as if the sub grades are of low gradings (F and G) to be specific, then there's almost 50 % probability of the customers defaulting on the loan taken. The Lending Clubs/Banks should be careful in this perspective as well.
- Also if we think in terms of underwriting process behind Supply Chain Financing, in my analysis, I found that the if the income source is not verified by Lending Clubs/Banks, then it's a possibility of higher default as it's a key parameter after VIF and RFE (Recurrsive Feature Elimaination) considerations. In addition if the income of the borrower is very low the Lending Clubs/Banks should rethink twice before disbursind the loan amount.

In [ ]:

Questionnaire Answers:

1. What percentage of customers have fully paid their Loan Amount?

1. What percentage of customers have fully paid their Loan Amount?

* 80 % of the customers have fully paid their Loan Amount

2. Comment about the correlation between Loan Amount and Installment features.

* loan_amnt and installment are highly correlated which means , more the number of loan amount, more are
  the installments.
* When the loan taken is less than 20000, then the installments in form of the monthly payment owed by the borrower
  if the loan originates, are similar to both types of borrowers who have defaulted and who had fully paid the loan.
* But as loan amount increases, the installments owed by the defaulters decreases as compared to the installments
  by the honest borrowers who are not defaulting.

3. The majority of people have home ownership as ___.

* The majority of people have home ownership as MORTGAGE

4. People with grades 'A' are more likely to fully pay their loan. (T/F)

* True

5. Name the top 2 afforded job titles.

* 2 afforded job titles - Teacher and Manager

6. Thinking from a bank's perspective, which metric should our primary focus be on..

1. ROC AUC
2. Precision
3. Recall
4. F1 Score

* If thinking purely in terms of not loosing money in bad loans, Recall is the parameter which should be more focused. In Model 4, I am getting recall as 66 % which is comparatively higher than any of the other three Models.

* If we think in terms of not loosing customers at the cost of loosing some money in bad loans, Precision should be the main parameter that we should focus. Our three models (1,2,3), Model 3 to be specific has highest precision percentage i.e nearly equal to 100 %

* But we have to provide a balanced stance, then F1 score is the metric as it considers both Precision and Recall (harmonically). Model 2 has the highest F1 core in my analysis.

* In order to conclude, since 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. And hence Model 4 is the best model which is having highest Recall percentage

7. How does the gap in precision and recall affect the bank?

* The gap in precision and recall affects the Lending Clubs/Banks in terms if more difference in precision and recall, more are the chances of Lending Clubs/Banks loosing money in form of bad loans. As per my analysis in Model 4 we are getting less precision core then the recall score and the difference is of 15. Whereas in all other models the difference between Precision and Recall is around 50 % (Precision being the highest and recall being in range of 40-50 %. This clearly shows that there's a risk of Lending Clubs/Banks giving bad loans to defaulters and thus the NPA of the Lending Clubs/Banks are currently rising.

8. Which were the features that heavily affected the outcome?

* According to my analysis, Term (36 or 60 months) , Subgrades (A, B ,F5 and G) application_type_JOINT,Number of mortage accounts (Mort_acc) and any of the two - number of public record Lending Clubs/Banksruptcy (pub_rec_bankruptcies) or Number of derogatory public records (pub_rec) .
* These are the top 5 features which are heavily affecting the outcome and are primarily important.

9. Will the results be affected by geographical location? (Yes/No)

* Yes, there are few Geographical locations based on zip codes where the charged off rates are 100 % . Hence the
results are affected by geographical location

In [ ]: