Case Study : Logistic Regrerssion Case Study

Problem Statement:

Design a Logistic Regression model to correctly classify the customer based on the given set of attributes into two categories - whether the customer will be able to repay the loan or will it possibly result into NPA (Non-performing Account). The notion is that bank should not loose good a customer or retain a defaulter customer because of "False Alarm".

Importing Libraries

In [11]:

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

In [12]:

```
pd.set_option('display.max_columns', None)
```

In [13]:

```
warnings.filterwarnings('ignore')
sns.set_style('darkgrid')
```

Import dataset and perform EDA

In [14]:

```
df = pd.read_csv('LoanTapData.csv')
df.shape
```

Out[14]:

```
(33091, 27)
```

In [15]:

```
df.head()
```

Out[15]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	verific
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	RENT	117000.0	
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RENT	43057.0	So
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	RENT	54000.0	

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	verific
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55000.0	

In [16]:

```
#Check for dupliicate values
df.duplicated().sum()
```

Out[16]:

```
0
```

In [17]:

```
#Check for null values
df.isna().sum().sort_values(ascending=False)
```

Out[17]:

```
mort_acc                3034
emp_title               1877
emp_length              1514
title                    134
pub_rec_bankruptcies      46
revol_util                23
address                    1
dti                        0
application_type           0
initial_list_status        0
total_acc                  0
revol_bal                  0
pub_rec                    0
open_acc                   0
earliest_cr_line           0
loan_amnt                  0
term                       0
loan_status                0
issue_d                    0
verification_status        0
annual_inc                 0
home_ownership             0
sub_grade                  0
grade                      0
installment                0
int_rate                   0
purpose                    0
dtype: int64
```

In [18]:

```
#Check for percentage of null values
round(df.isna().sum()/len(df)*100,2).sort_values(ascending=False)
```

Out[18]:

```
mort_acc                9.17
emp_title               5.67
emp_length              4.58
title                   0.40
pub_rec_bankruptcies    0.14
revol_util              0.07
loan_amnt               0.00
dti                     0.00
application_type        0.00
initial_list_status     0.00
total_acc               0.00
revol_bal               0.00
pub_rec                 0.00
open_acc                0.00
earliest_cr_line        0.00
purpose                 0.00
```

```
purpose                0.00
term                   0.00
loan_status            0.00
issue_d                0.00
verification_status    0.00
annual_inc             0.00
home_ownership         0.00
sub_grade              0.00
grade                  0.00
installment            0.00
int_rate               0.00
address                0.00
dtype: float64
```

In [19]:

```
#Check dataframe.info(). Get the null values and datatypes
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33091 entries, 0 to 33090
Data columns (total 27 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   loan_amnt             33091 non-null  float64
 1   term                  33091 non-null  object 
 2   int_rate              33091 non-null  float64
 3   installment           33091 non-null  float64
 4   grade                 33091 non-null  object 
 5   sub_grade             33091 non-null  object 
 6   emp_title             31214 non-null  object 
 7   emp_length            31577 non-null  object 
 8   home_ownership        33091 non-null  object 
 9   annual_inc            33091 non-null  float64
 10  verification_status   33091 non-null  object 
 11  issue_d               33091 non-null  object 
 12  loan_status           33091 non-null  object 
 13  purpose               33091 non-null  object 
 14  title                 32957 non-null  object 
 15  dti                   33091 non-null  float64
 16  earliest_cr_line      33091 non-null  object 
 17  open_acc              33091 non-null  float64
 18  pub_rec               33091 non-null  float64
 19  revol_bal             33091 non-null  float64
 20  revol_util            33068 non-null  float64
 21  total_acc             33091 non-null  float64
 22  initial_list_status   33091 non-null  object 
 23  application_type      33091 non-null  object 
 24  mort_acc              30057 non-null  float64
 25  pub_rec_bankruptcies  33045 non-null  float64
 26  address               33090 non-null  object 
dtypes: float64(12), object(15)
memory usage: 6.8+ MB
```

In [20]:

```
#Missing Value Treatments upon analysis
#NaN values are replaced as below
df.loc[df['emp_title'].isna(),'emp_title'] = 'No Employee Title'
df.loc[df['emp_length'].isna(),'emp_length'] = 'Unavailable'
df.loc[df['title'].isna(),'title'] = 'Unavailable'
df.loc[df['revol_util'].isna(),'revol_util'] = 0.0
df.loc[df['mort_acc'].isna(),'mort_acc'] = 0.0
df.loc[df['pub_rec_bankruptcies'].isna(),'pub_rec_bankruptcies'] = 0.0
```

In [21]:

```
#Check for missing values
df.isna().sum()
```

Out[21]:

```
loan_amnt                0
```

```
term                    0
int_rate                0
installment             0
grade                   0
sub_grade               0
emp_title               0
emp_length              0
home_ownership          0
annual_inc              0
verification_status     0
issue_d                 0
loan_status             0
purpose                 0
title                   0
dti                     0
earliest_cr_line        0
open_acc                0
pub_rec                 0
revol_bal               0
revol_util              0
total_acc               0
initial_list_status     0
application_type        0
mort_acc                0
pub_rec_bankruptcies    0
address                 1
dtype: int64
```

In [22]:

```
#Get stats of numeric/continuous variables
df.describe()
```

Out[22]:

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_rec	revol_bal	
count	33091.000000	33091.000000	33091.000000	3.309100e+04	33091.000000	33091.000000	33091.000000	33091.000000	33
mean	14094.069384	13.647271	431.478318	7.415520e+04	17.360175	11.304191	0.180321	15819.015896	
std	8337.658049	4.459740	249.935889	6.158381e+04	8.201504	5.115333	0.517188	19320.630293	
min	500.000000	5.320000	16.310000	2.500000e+03	0.000000	1.000000	0.000000	0.000000	
25%	8000.000000	10.490000	252.120000	4.500000e+04	11.250000	8.000000	0.000000	6035.500000	
50%	12000.000000	13.330000	376.440000	6.400000e+04	16.850000	10.000000	0.000000	11169.000000	
75%	19800.000000	16.490000	564.665000	9.000000e+04	22.970000	14.000000	0.000000	19631.000000	
max	40000.000000	30.740000	1533.810000	6.100000e+06	189.900000	51.000000	11.000000	617838.000000	

In [23]:

```
#Get stats of categorical variables
df.describe(include='object')
```

Out[23]:

	term	grade	sub_grade	emp_title	emp_length	home_ownership	verification_status	issue_d	loan_status	
count	33091	33091	33091	33091	33091	33091	33091	33091	33091	
unique	2	7	35	20166	12	6	3	115	2	
top	36 months	B	B3	No Employee Title	10+ years	MORTGAGE	Verified	Oct-2014	Fully Paid	debt_cc
freq	25253	9811	2289	1877	10670	16493	11470	1256	26658	

Feature Engineering

In [24]:

```
#Perform Encoding
df.loc[df['pub_rec'] >= 1,'pub_rec'] = 1
df.loc[df['mort_acc'] >= 1, 'mort_acc'] = 1
df.loc[df['pub_rec_bankruptcies'] >= 1, 'pub_rec_bankruptcies'] = 1
df.loc[df['term'] ==' 36 months', 'term'] = 36
df.loc[df['term'] ==' 60 months', 'term'] = 60
df['term'] = df['term'].astype('int64')
```

In [25]:

```
#Split issue_date into month and year
df[['issue_month', 'issue_year']] = df['issue_d'].str.split('-', expand=True)
df.drop(['issue_d'], axis=1, inplace=True)
```

In [26]:

```
#Split er_cr_line date into month and year
df[['er_cr_line_m', 'er_cr_line_y']] = df['earliest_cr_line'].str.split('-', expand=True
)
df.drop(['earliest_cr_line'], axis=1, inplace=True)
```

In [27]:

```
#Split address into State and Zip code
df[['address_state', 'address_zip']] = df['address'].str.split(','
                                                                , expand=True)[1].str.sp
lit(' '
                                                                 ,expand=True)[[1,2]]
df.drop(['address'], axis=1, inplace=True)
```

In [28]:

```
#Make emp_title, purpose and title as uppercase fields
df['emp_title'] = df['emp_title'].str.upper()
df['purpose'] = df['purpose'].str.upper()
df['title'] = df['title'].str.upper()
```

Outliers Detection

In [29]:

```
df1 = df.copy()
```

In [30]:

```
#Removing some extreme outliers values for annual income
print(np.percentile(df['annual_inc'], 50))
print(np.percentile(df['annual_inc'], 99))
print(np.percentile(df['annual_inc'], 99.99))
print(round(df.loc[df['annual_inc'] > 210000.0].shape[0]/len(df),2)*100)
df = df.loc[~(df['annual_inc'] > np.percentile(df['annual_inc'], 99))]
```

```
64000.0
250000.0
1250000.0
2.0
```

In [31]:

```
#Removing some extreme outliers values for pub_rec
print(np.percentile(df['pub_rec'], 50))
print(np.percentile(df['pub_rec'], 99))
print(np.percentile(df['pub_rec'], 99.99))
```

```
print(round(df.loc[df['pub_rec'] > 9.0].shape[0]/len(df),2)*100)
df = df.loc[~(df['pub_rec'] > np.percentile(df['pub_rec'], 99.99))]
```

```
0.0
1.0
1.0
0.0
```

In [32]:

```
#Removing some extreme outliers values for pub_rec_bankruptcies
print(np.percentile(df['pub_rec_bankruptcies'], 50))
print(np.percentile(df['pub_rec_bankruptcies'], 99))
print(np.percentile(df['pub_rec_bankruptcies'], 99.99))
print(round(df.loc[df['pub_rec_bankruptcies'] > 5.0].shape[0]/len(df),2)*100)
df = df.loc[~(df['pub_rec_bankruptcies'] > np.percentile(df['pub_rec_bankruptcies'], 99.
99))]
```

```
0.0
1.0
1.0
0.0
```

In [33]:

```
#Define Outlier Detection function based on IQR and Percentile

def detect_outliers(df,col):
    q1 = np.quantile(df[col],0.25)
    q3 = np.quantile(df[col],0.75)
    iqr = q3-q1
    lb = q1 - 1.5*iqr
    ub = q3 + 1.5*iqr
    outlier = df.loc[(df[col] < lb) | (df[col] > ub)]
    return round(outlier.shape[0]/df.shape[0]*100,2)

def detect_outliers_percentile(df,col):
    q1 = np.quantile(df[col],0.25)
    q3 = np.quantile(df[col],0.75)
    p = np.percentile(df[col],99.99)
    iqr = q3-q1
    lb = q1 - 1.5*iqr
    ub = q3 + 1.5*iqr
    outlier = df.loc[(df[col] < lb) | (df[col] > p)]
    return round(outlier.shape[0]/df.shape[0]*100,2)
```

In [34]:

```
#Print percentage of outliers for each cont. variable
print(f"Outlier Percentage")
print(f"loan_amnt            = {detect_outliers(df,'loan_amnt')}%")
print(f"int_rate             = {detect_outliers(df,'int_rate')}%")
print(f"installment          = {detect_outliers(df,'installment')}%")
print(f"annual_inc           = {detect_outliers(df,'annual_inc')}%")
print(f"dti                  = {detect_outliers(df,'dti')}%")
print(f"open_acc             = {detect_outliers(df,'open_acc')}%")
print(f"pub_rec              = {detect_outliers_percentile(df,'pub_rec')}%")
print(f"revol_bal            = {detect_outliers(df,'revol_bal')}%")
print(f"revol_util           = {detect_outliers(df,'revol_util')}%")
print(f"total_acc            = {detect_outliers(df,'total_acc')}%")
print(f"mort_acc             = {detect_outliers(df,'mort_acc')}%")
print(f"pub_rec_bankruptcies = {detect_outliers(df,'pub_rec_bankruptcies')}%")
```

```
Outlier Percentage
loan_amnt            = 0.05%
int_rate             = 0.97%
installment          = 2.88%
annual_inc           = 3.62%
dti                  = 0.09%
open_acc             = 2.54%
pub_rec              = 0.0%
revol_bal            = 5.46%
```

```
revol_util           = 0.0%
total_acc            = 2.04%
mort_acc             = 0.0%
pub_rec_bankruptcies = 11.5%
```

Outliers Treatment

In [35]:

```
#Define function to remove outliers based on IQR

def remove_outliers(df,col):
    q1 = np.quantile(df[col],0.25)
    q3 = np.quantile(df[col],0.75)
    iqr = q3-q1
    lb = q1 - 1.5*iqr
    ub = q3 + 1.5*iqr
    return df.loc[~((df[col] < lb) | (df[col] > ub))]

# def remove_outliers_percentile(df,col):
#     q1 = np.quantile(df[col],0.25)
#     q3 = np.quantile(df[col],0.75)
#     p = np.percentile(df[col],99.99)
#     iqr = q3-q1
#     lb = q1 - 1.5*iqr
#     ub = q3 + 1.5*iqr
#     return df.loc[~((df[col] < lb) | (df[col] > p))]
```

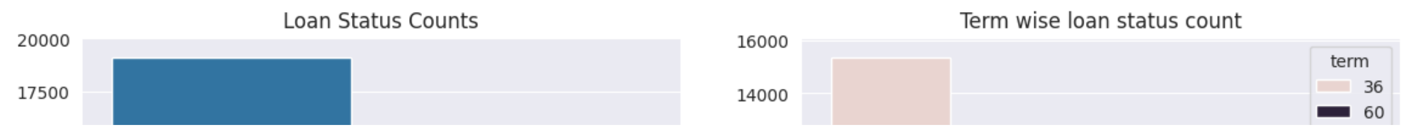
In [36]:

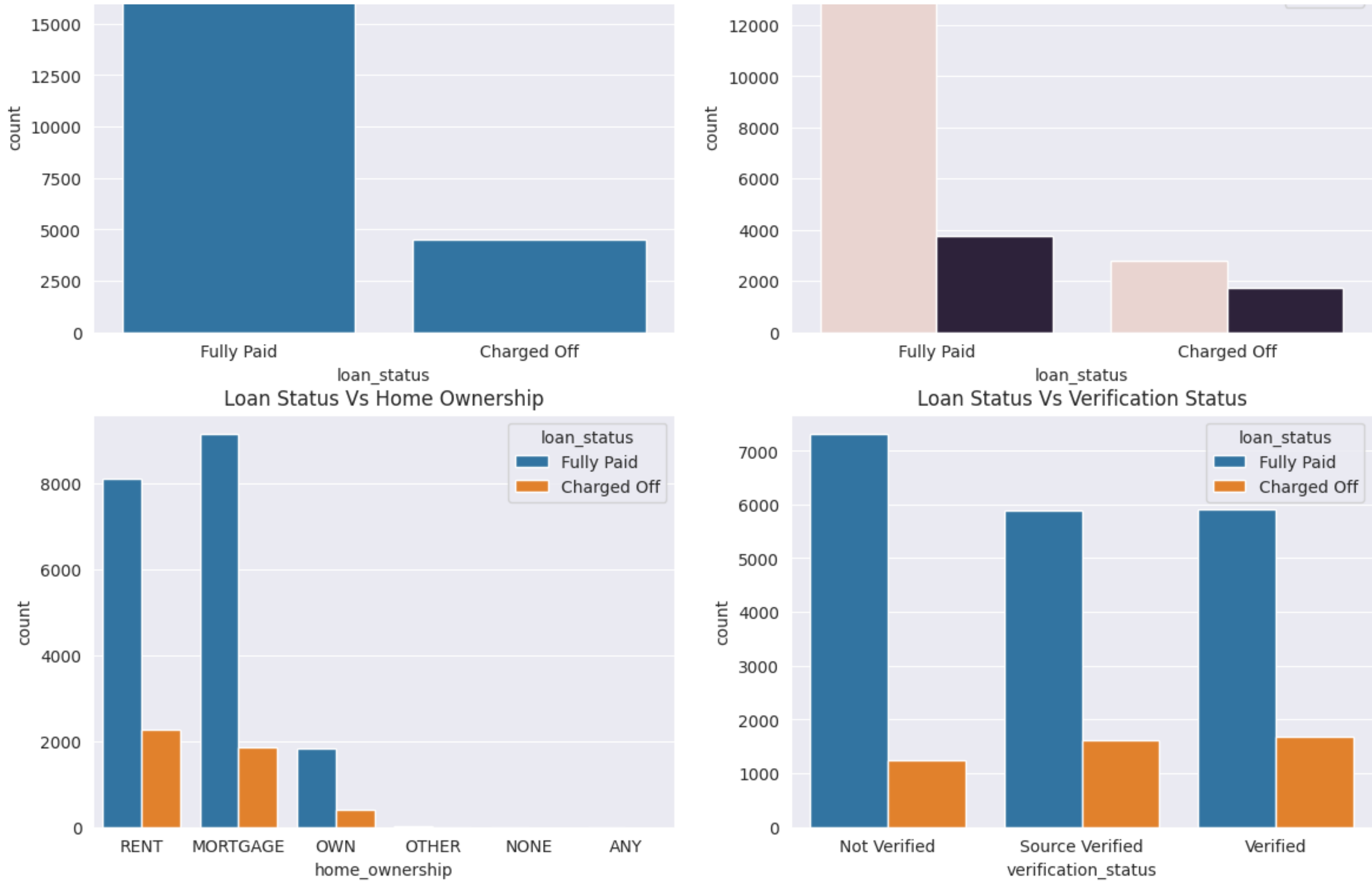
```
#Remove outliers from cont. variables mentioned below
df = remove_outliers(df, 'loan_amnt')
df = remove_outliers(df, 'int_rate')
df = remove_outliers(df, 'installment')
df = remove_outliers(df, 'annual_inc')
df = remove_outliers(df, 'dti')
df = remove_outliers(df, 'pub_rec')
df = remove_outliers(df, 'revol_bal')
df = remove_outliers(df, 'revol_util')
df = remove_outliers(df, 'open_acc')
df = remove_outliers(df, 'total_acc')
df = remove_outliers(df, 'mort_acc')
df = remove_outliers(df, 'pub_rec_bankruptcies')
```

Univariate/Bivariate Analysis

In [37]:

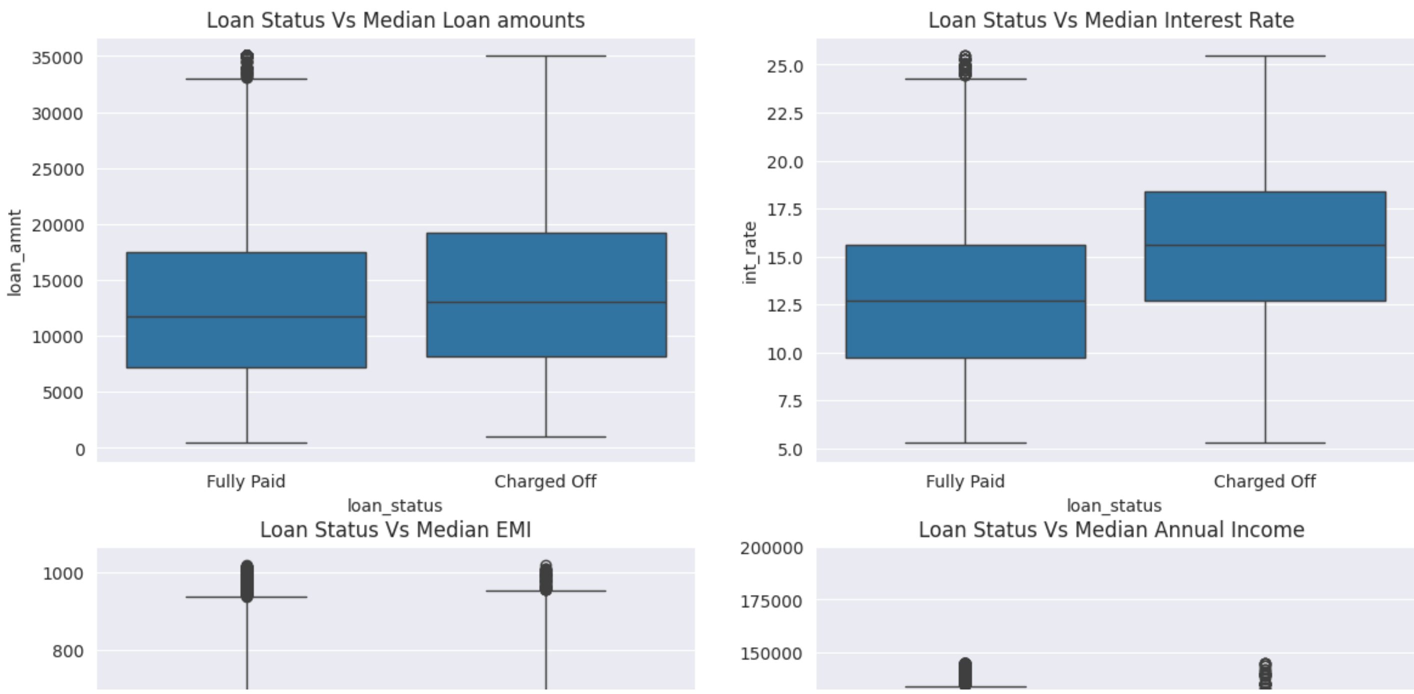
```
#Countplots of various categorical features w.r.t. to target variable loan_status
plt.figure(figsize=(14,10))
plt.subplot(2,2,1)
sns.countplot(data=df, x='loan_status')
plt.title('Loan Status Counts')
plt.subplot(2,2,2)
sns.countplot(data=df, x='loan_status', hue='term')
plt.title('Term wise loan status count')
plt.subplot(2,2,3)
sns.countplot(data=df, x='home_ownership', hue='loan_status')
plt.title('Loan Status Vs Home Ownership')
plt.subplot(2,2,4)
sns.countplot(data=df, x='verification_status', hue='loan_status')
plt.title('Loan Status Vs Verification Status')
plt.show()
```

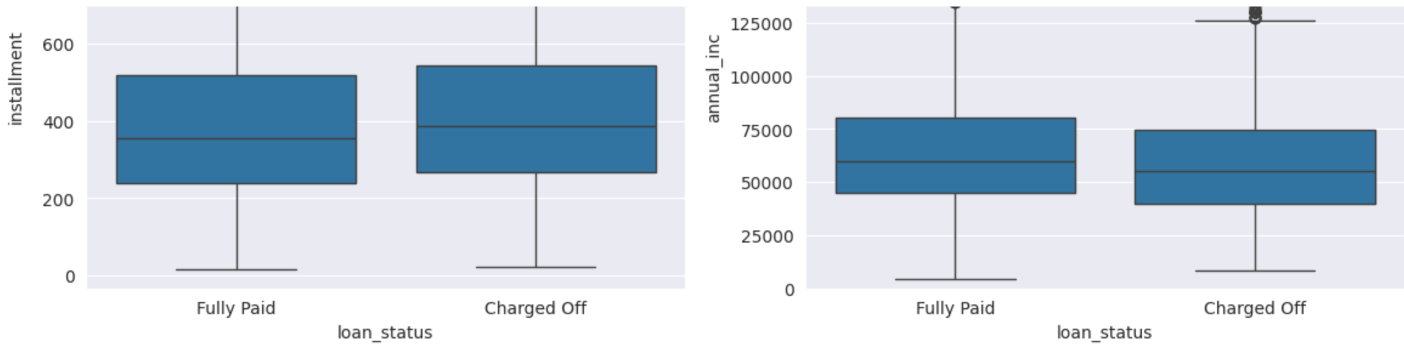




In [38]:

```
#Boxplot of various cont. features w.r.t. target variable loan_status
plt.figure(figsize=(14,10))
plt.subplot(2,2,1)
sns.boxplot(data=df, x='loan_status', y='loan_amnt')
plt.title('Loan Status Vs Median Loan amounts')
plt.subplot(2,2,2)
sns.boxplot(data=df, x='loan_status', y='int_rate')
plt.title('Loan Status Vs Median Interest Rate ')
plt.subplot(2,2,3)
sns.boxplot(data=df, x='loan_status', y='installment')
plt.title('Loan Status Vs Median EMI')
plt.subplot(2,2,4)
sns.boxplot(data=df, x='loan_status', y='annual_inc')
plt.ylim(bottom=0, top=200000)
plt.title('Loan Status Vs Median Annual Income ')
plt.show()
```





Observation 1: Median interest rate of Charged Off customers is significantly higher than those of Fully Paid

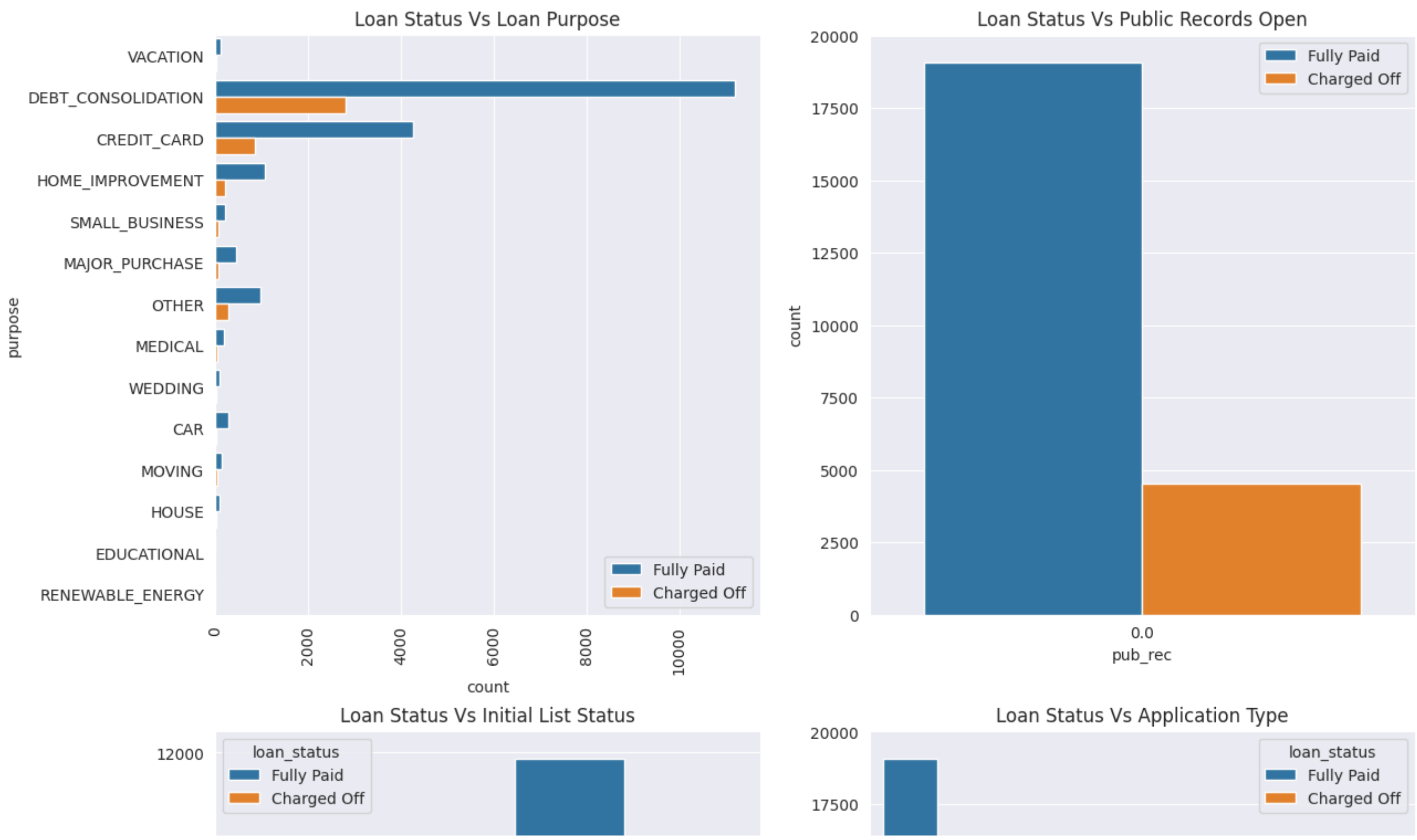
Observation 2: Median annual income of Charged Off customers is lower than those of Fully Paid

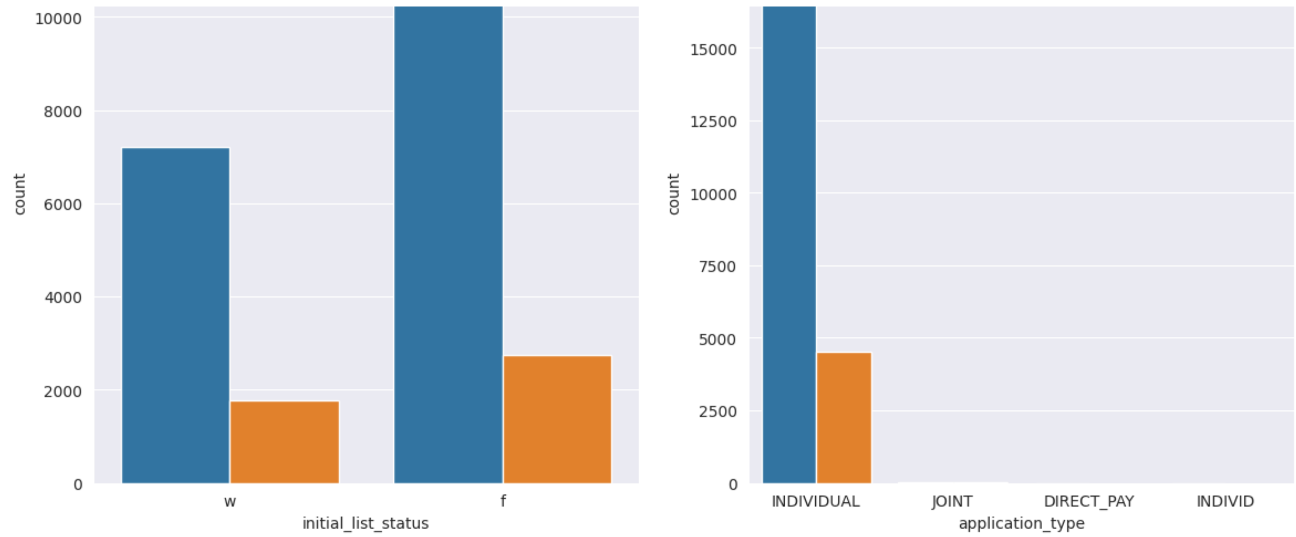
Observation 3: Median EMI of Charged Off is higher than those of Fully Paid

Observation 4: Median loan amount of Charged Off is higher than those of Fully Paid

In [39]:

```
#Countplot of categorical variables w.r.t. target variable loan_status
plt.figure(figsize=(14,15))
plt.subplot(2,2,1)
sns.countplot(data=df, y='purpose', hue='loan_status')
plt.xticks(rotation=90)
plt.title('Loan Status Vs Loan Purpose')
plt.legend(loc=4)
plt.subplot(2,2,2)
sns.countplot(data=df, x='pub_rec',hue='loan_status')
#plt.xlim(left=0,right=10)
plt.title('Loan Status Vs Public Records Open')
plt.legend(loc=1)
plt.subplot(2,2,3)
sns.countplot(data=df, x='initial_list_status', hue='loan_status')
plt.title('Loan Status Vs Initial List Status')
plt.subplot(2,2,4)
sns.countplot(data=df, x='application_type',hue='loan_status')
#plt.xlim(left=0,right=10)
plt.title('Loan Status Vs Application Type')
#plt.legend(loc=1)
plt.show()
```



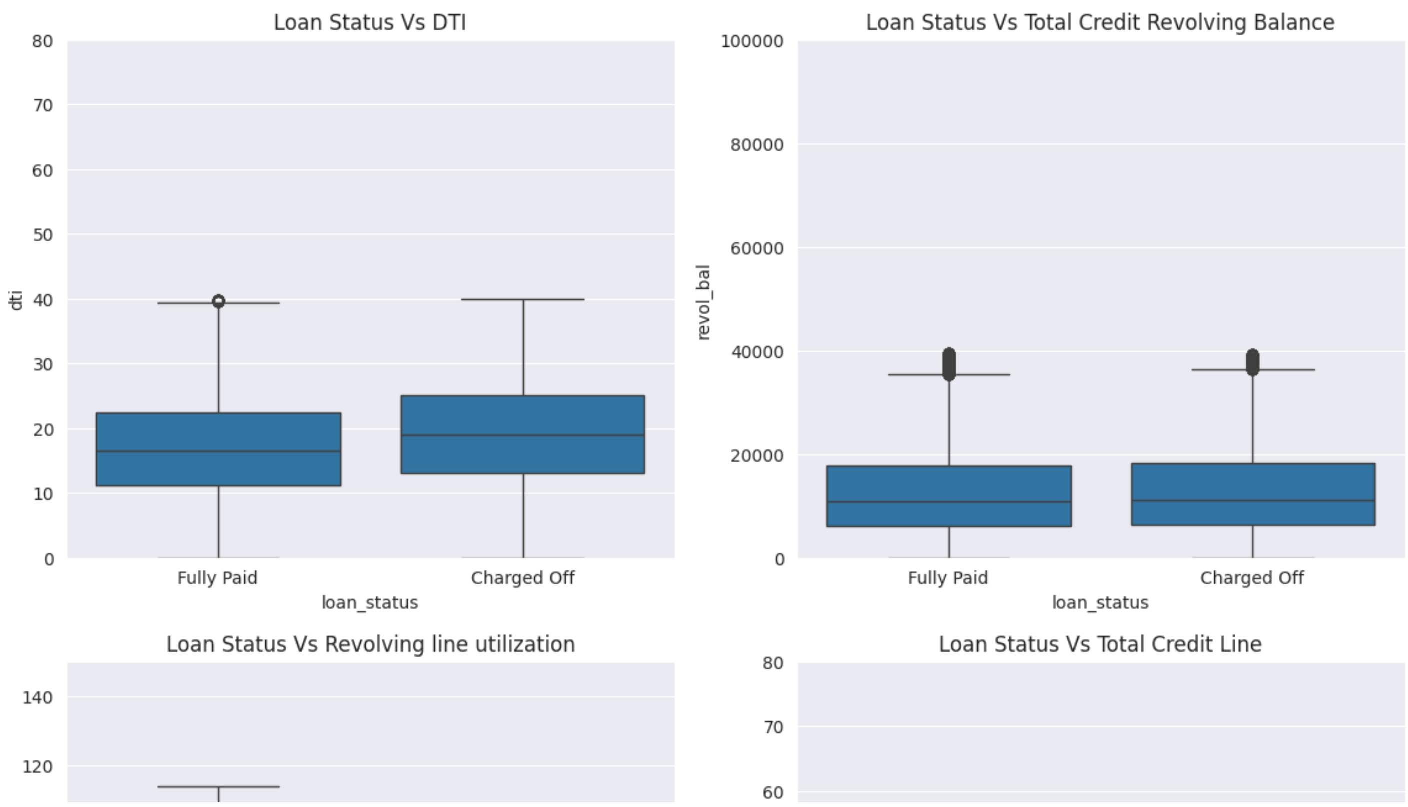


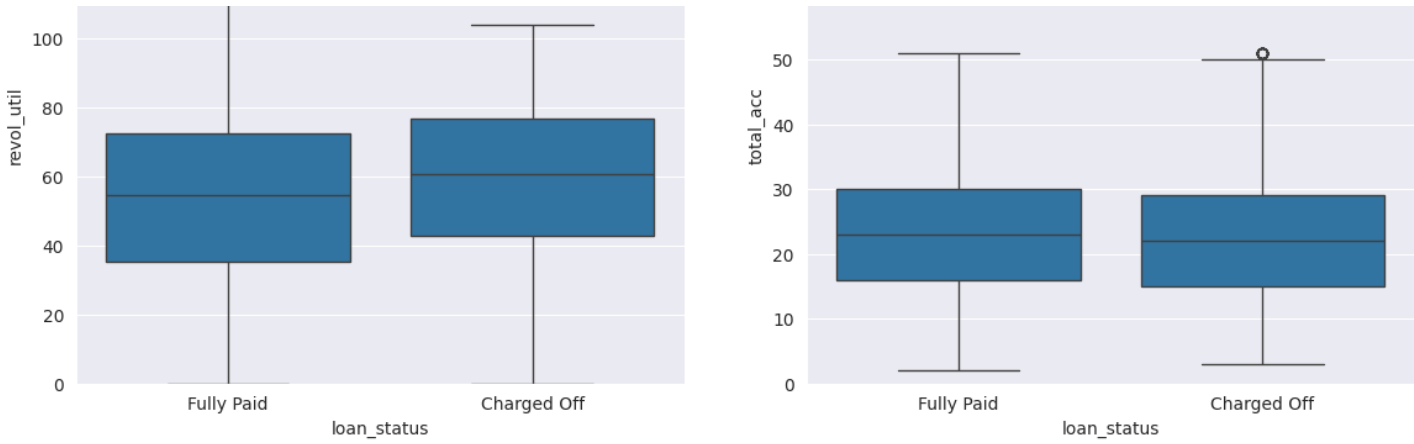
Observation 1: Top 2 loan purpose categories are Debit Consolidation and Credit Card

Observation 2: Topmost loan type application is INDIVIDUAL

In [40]:

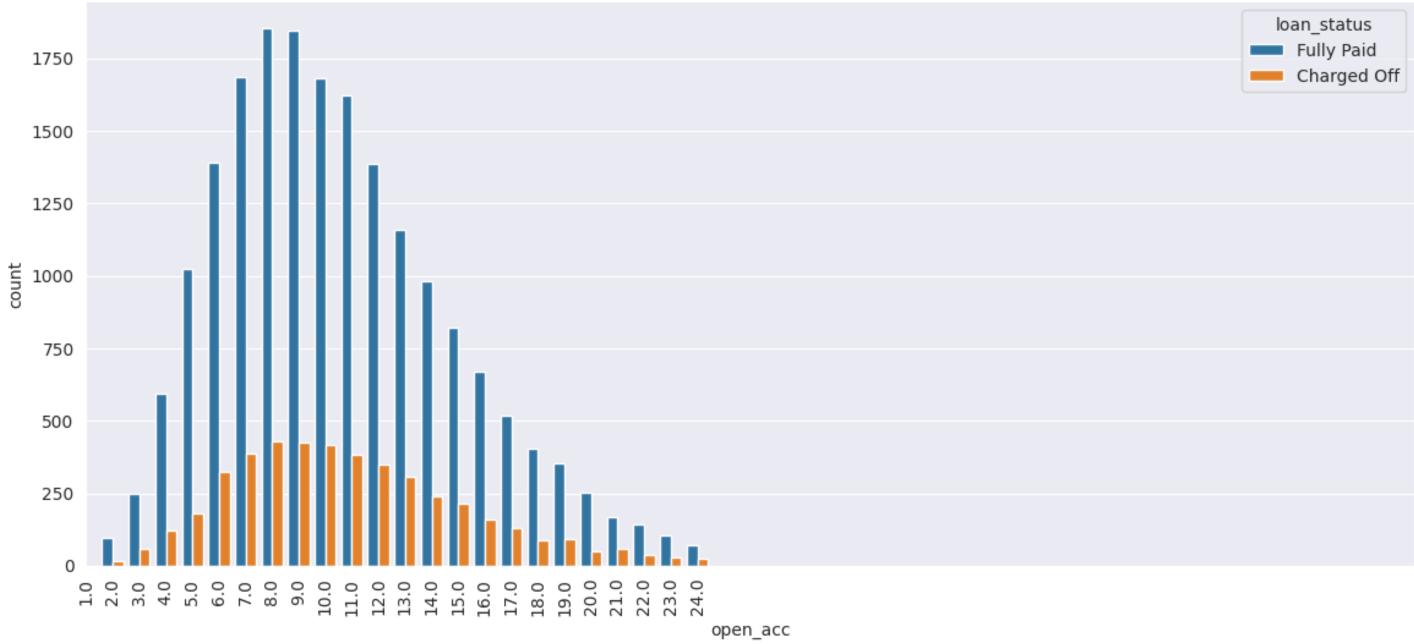
```
#Box plot of various cont. features w.r.t. target variable loan_status

plt.figure(figsize=(14,12))
plt.subplot(2,2,1)
sns.boxplot(data=df, x='loan_status', y='dti')
plt.ylim(bottom=0,top=80)
plt.title('Loan Status Vs DTI')
plt.subplot(2,2,2)
sns.boxplot(data=df, x='loan_status', y='revol_bal')
plt.ylim(bottom=0, top=100000)
plt.title('Loan Status Vs Total Credit Revolving Balance')
plt.subplot(2,2,3)
sns.boxplot(data=df, x='loan_status', y='revol_util')
plt.ylim(bottom=0,top=150)
plt.title('Loan Status Vs Revolving line utilization')
plt.subplot(2,2,4)
sns.boxplot(data=df, x='loan_status', y='total_acc')
plt.ylim(bottom=0, top=80)
plt.title('Loan Status Vs Total Credit Line')
plt.show()
```





In [41]:

```
#Countplot of categorical variable open_acc w.r.t. target variable loan_status

plt.figure(figsize=(14,6))
sns.countplot(data=df, x='open_acc',hue='loan_status')
plt.xlim(left=0,right=50)
plt.xticks(rotation=90)
plt.show()
```

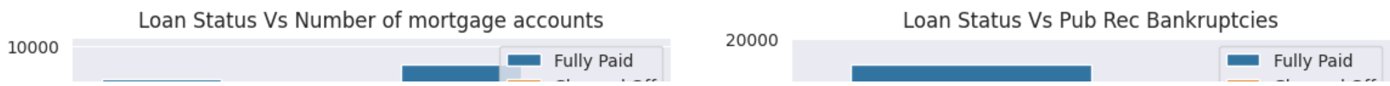


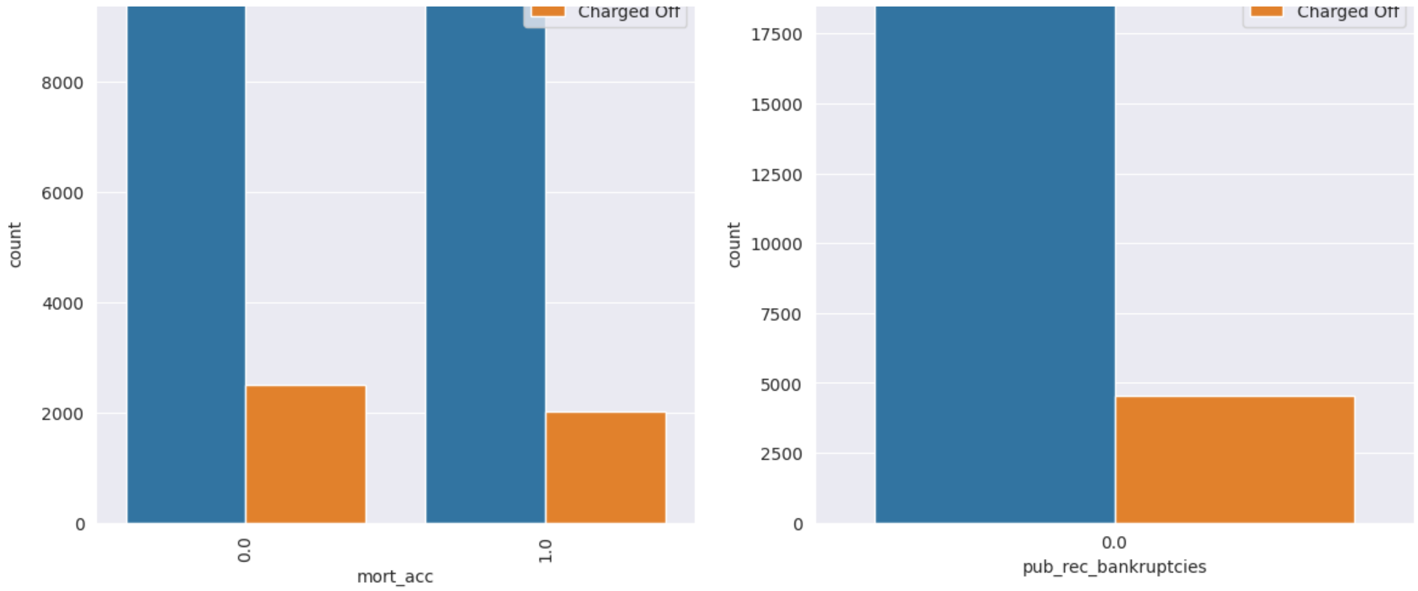
Observation 1: open_acc is fairly graphically normall distributed

Observation 2: Charged Off and Fully Paid have same distribution

In [42]:

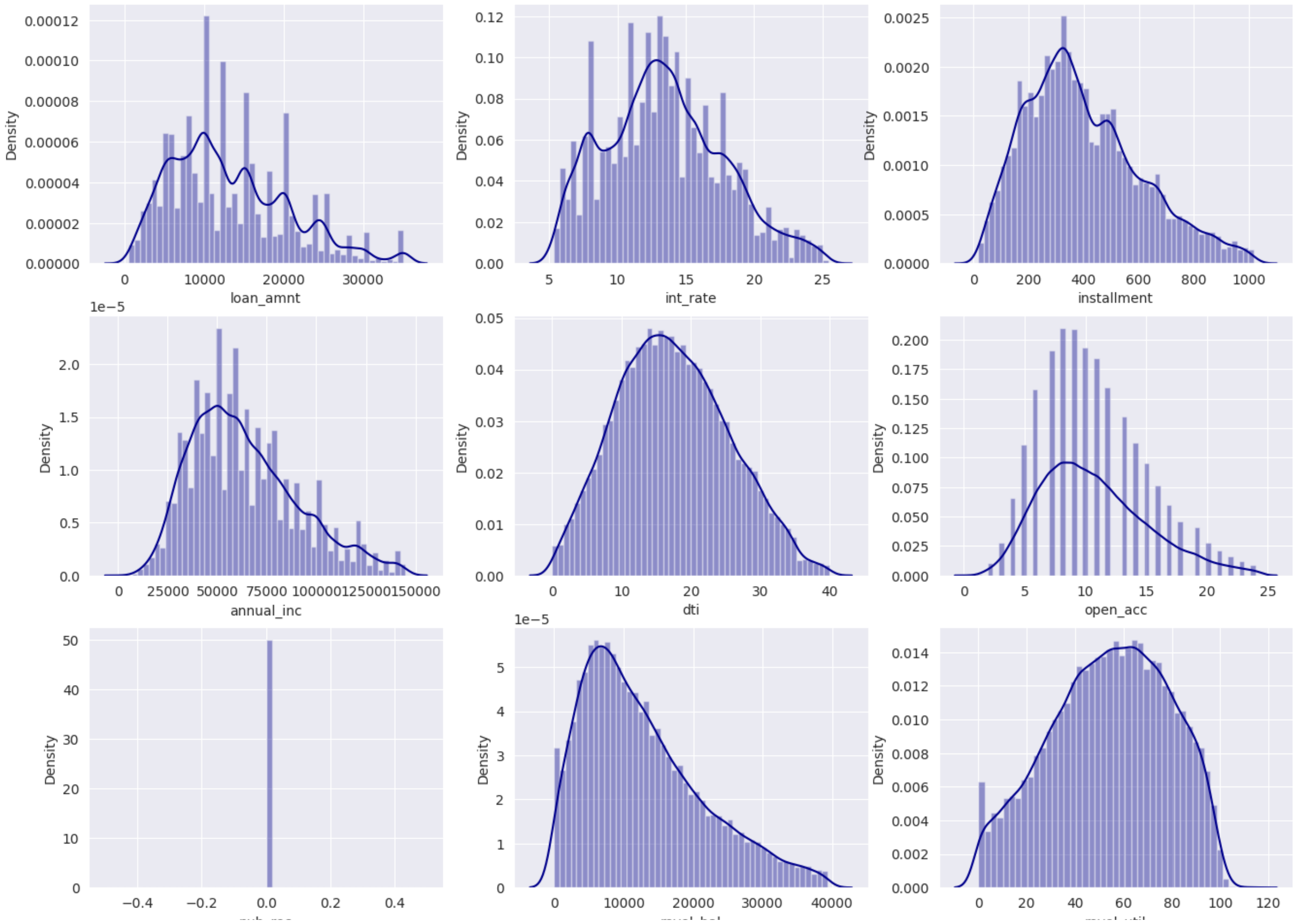
```
#Countplot for various categorical features w.r.t. target variable loan_status

plt.figure(figsize=(14,6))
plt.subplot(1,2,1)
sns.countplot(data=df, x='mort_acc',hue='loan_status')
plt.xticks(rotation=90)
plt.title('Loan Status Vs Number of mortgage accounts')
plt.legend(loc=1)
plt.subplot(1,2,2)
sns.countplot(data=df, x='pub_rec_bankruptcies',hue='loan_status')
#plt.xlim(left=0,right=10)
plt.title('Loan Status Vs Pub Rec Bankruptcies')
plt.legend(loc=1)
plt.show()
```

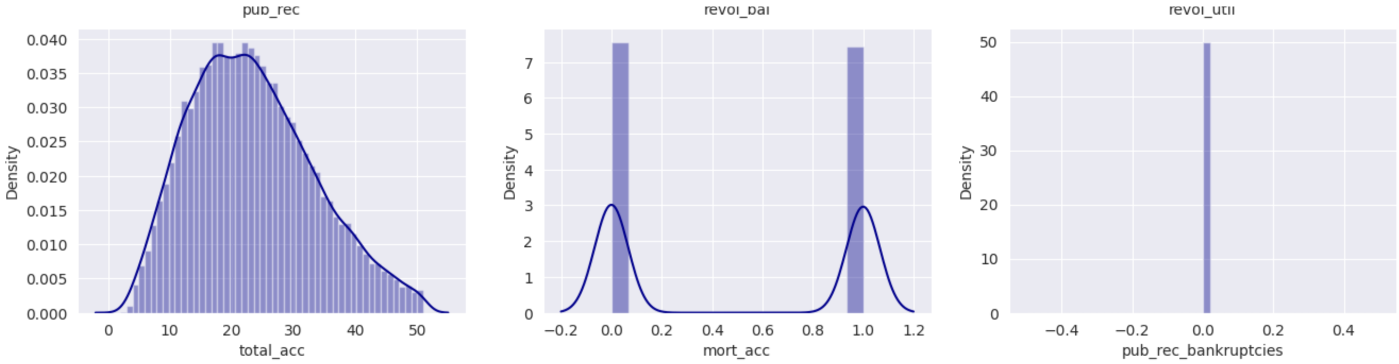




In [43]:

```
#Distribution plot for cont. variables

fig,axes = plt.subplots(4,3,figsize=(16,16))
sns.distplot(df['loan_amnt'],ax=axes[0,0],color='darkblue')
sns.distplot(df['int_rate'],ax=axes[0,1],color='darkblue')
sns.distplot(df['installment'],ax=axes[0,2],color='darkblue')
sns.distplot(df['annual_inc'],ax=axes[1,0],color='darkblue')
sns.distplot(df['dti'],ax=axes[1,1],color='darkblue')
sns.distplot(df['open_acc'],ax=axes[1,2],color='darkblue')
sns.distplot(df['pub_rec'],ax=axes[2,0],color='darkblue')
sns.distplot(df['revol_bal'],ax=axes[2,1],color='darkblue')
sns.distplot(df['revol_util'],ax=axes[2,2],color='darkblue')
sns.distplot(df['total_acc'],ax=axes[3,0],color='darkblue')
sns.distplot(df['mort_acc'],ax=axes[3,1],color='darkblue')
sns.distplot(df['pub_rec_bankruptcies'],ax=axes[3,2],color='darkblue')
plt.show()
```

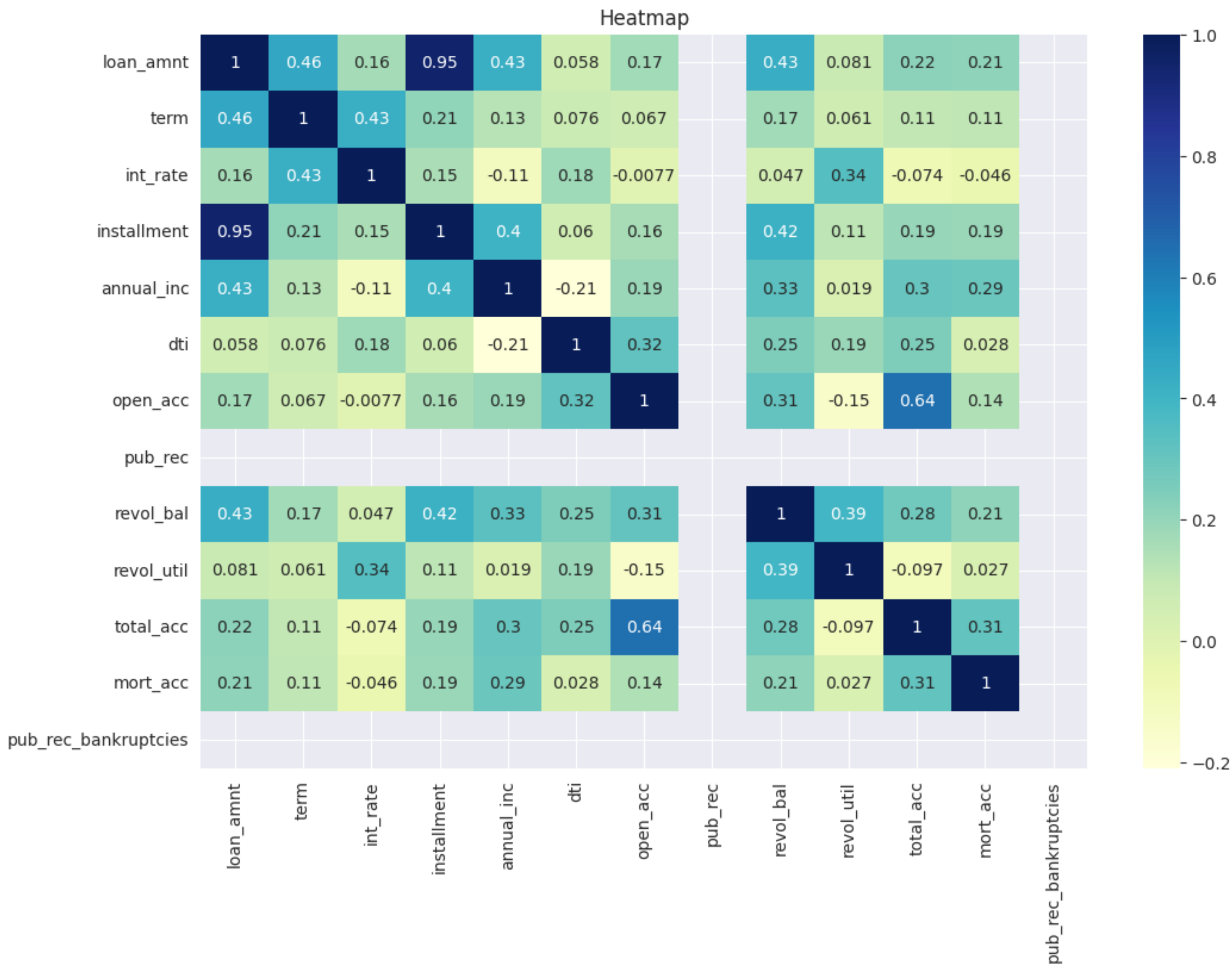




Correlation

In [44]:

```
#Prepare corr dataframe and plot heatmap

# Select only numeric columns
numeric_df = df.select_dtypes(include=['number'])

plt.figure(figsize=(12,8))
df_corr = numeric_df.corr()  # Calculate correlation on numeric data
sns.heatmap(df_corr, cmap='YlGnBu', annot=True)
plt.title('Heatmap')
plt.show()
```



Observation 1: **Installment and loan_amnt are highly correlated**

Observation 2: **Open_acc and total_acc are having fairly good positive correlation**

Data Preparation for ML

In [45]:

```
df1 = df.copy()
```

In [46]:

```
#Drop the variables which didn't show any significant impact on loan_status in above anal
ysis

df1.drop(['emp_title'
          ,'emp_length'
          ,'initial_list_status'
          ,'issue_month'
          ,'issue_year'
          ,'er_cr_line_m'
          ,'er_cr_line_y'
          ,'address_state'
          ,'address_zip'
          ,'application_type'
          ,'verification_status'
          ,'purpose'
          ,'title'
          ,'sub_grade'],axis=1,inplace=True)
```

In [47]:

```
#OneHotEncode variable home_ownership

df1 = pd.get_dummies(df1, prefix=['home_ownership'] ,columns=['home_ownership'])
```

In [48]:

```
#Binary encode target variable loan_status

loan_status_dict = {
    'Fully Paid':1,
    'Charged Off':0
}
df1['loan_status'] = df1['loan_status'].map(loan_status_dict)
```

In [49]:

```
#OneHotEncode variable grade

df1 = pd.get_dummies(df1, prefix=['grade'], columns=['grade'])
```

In [50]:

```
df1.head()
```

Out[50]:

	loan_amnt	term	int_rate	installment	annual_inc	loan_status	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	m
0	10000.0	36	11.44	329.48	117000.0	1	26.24	16.0	0.0	36369.0	41.8	25.0	
1	8000.0	36	11.99	265.68	65000.0	1	22.05	17.0	0.0	20131.0	53.3	27.0	
2	15600.0	36	10.49	506.97	43057.0	1	12.79	13.0	0.0	11987.0	92.2	26.0	
3	7200.0	36	6.49	220.65	54000.0	1	2.60	6.0	0.0	5472.0	21.5	13.0	
4	24375.0	60	17.27	609.33	55000.0	0	33.95	13.0	0.0	24584.0	69.8	43.0	

Building ML model

Importing libraries

In [51]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_repo
rt
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.metrics import precision_recall_curve, auc
from sklearn.model_selection import train_test_split

# Import the ConfusionMatrixDisplay class directly for plotting
from sklearn.metrics import ConfusionMatrixDisplay
```

In [52]:

```
#Prepare X and y dataset i.e. independent and dependent datasets

X = df1.drop(['loan_status'], axis=1)
y = df1['loan_status']
```

In [53]:

```
#Split the data into train and test

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(18889, 26)
(4723, 26)
(18889,)
(4723,)
```

In [54]:

```
#Standardize the data

scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
X_train = pd.DataFrame(X_train, columns=X.columns)
X_test = pd.DataFrame(X_test, columns=X.columns)
```

In [55]:

```
# Fill NaN values in y_train with the most frequent value
y_train_filled = y_train.fillna(y_train.mode()[0])

# Fit the model
model = LogisticRegression()
model.fit(X_train, y_train_filled)
```

Out[55]:

```
▾ LogisticRegression
LogisticRegression()
```

In [56]:

```
#Predit the data on test dataset
```

```
y_pred = model.predict(X_test)
```

In [57]:

```
print(f'Logistic Regression Model Score: ',end='')
print(round(model.score(X_test, y_test)*100,2))
```

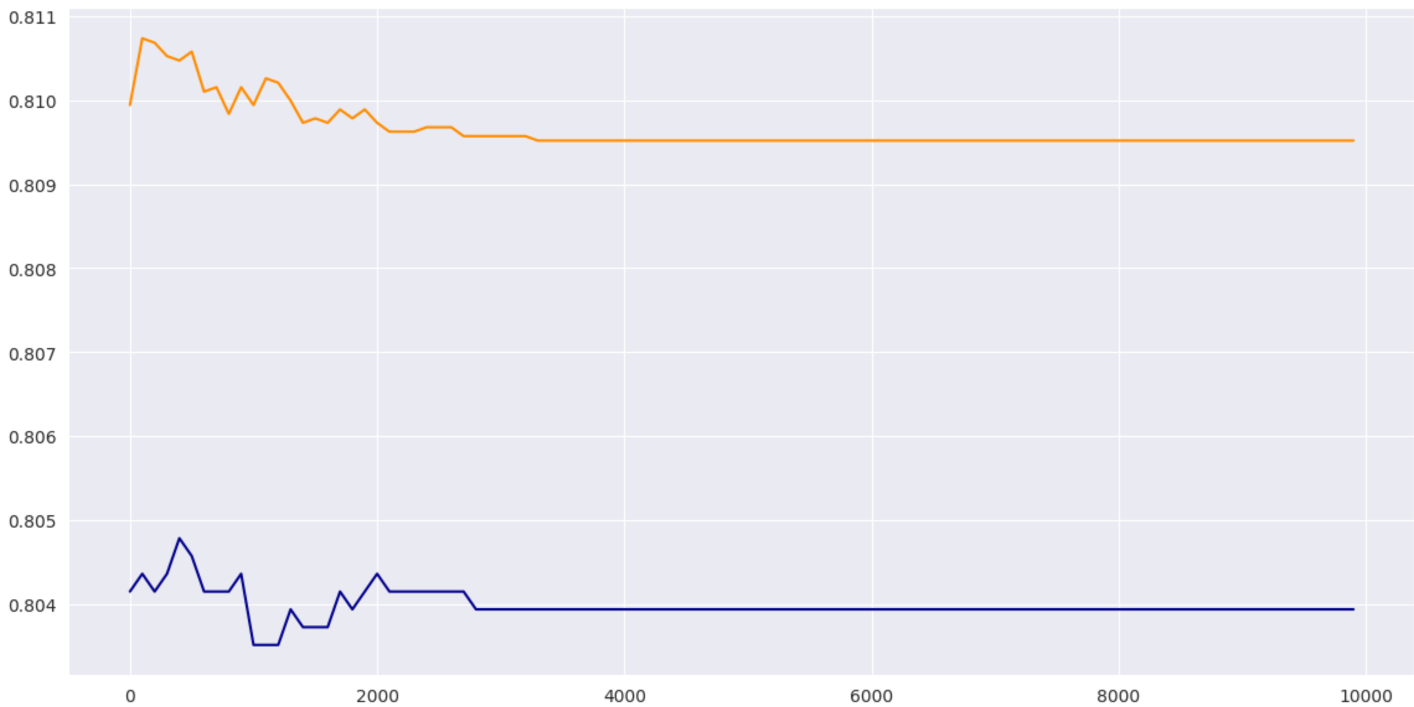
Logistic Regression Model Score: 80.41

In [59]:

```
#Try with different regularization factor lamda and choose the best to build the model

lamb = np.arange(0.01, 10000, 100)

train_scores = []
test_scores = []

for lam in lamb:
    model = LogisticRegression(C = 1/lam)
    model.fit(X_train, y_train)

    tr_score = model.score(X_train, y_train)
    te_score = model.score(X_test, y_test)

    train_scores.append(tr_score)
    test_scores.append(te_score)
```

In [61]:

```
#Plot the train and test scores with respect lambda values i.e. regularization factore

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

plt.figure(figsize=(14,7))
sns.lineplot(x=np.arange(0.01,10000,100), y=test_scores, color='darkblue') # Pass x and
y values as keyword arguments
sns.lineplot(x=np.arange(0.01,10000,100), y=train_scores, color='darkorange') # Pass x a
nd y values as keyword arguments
plt.show()
```



In [62]:

```
#Check the index of best test score and the check the best test score
```

```
print(np.argmax(test_scores))
test_scores[9]
```

4

Out[62]:

0.8043616345543088

In [63]:

```
#Calculate the best lambda value based on the index of best test score

best_lamb = 0.01 + 100*13
```

In [64]:

```
#Fit the model using best lambda

model = LogisticRegression(C=1/best_lamb)
model.fit(X_train, y_train)
```

Out[64]:

```
▾ LogisticRegression
LogisticRegression(C=0.0007692248521165222)
```

In [65]:

```
#Predict the y_values and y_probability values

y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)
```

In [66]:

```
#Print model score

print(f'Logistic Regression Model Score with best lambda: ',end='')
print(round(model.score(X_test, y_test)*100,2))
```

Logistic Regression Model Score with best lambda: 80.39

In [67]:

```
#Collect the model coefficients and print those in dataframe format
coeff_df = pd.DataFrame()
coeff_df['Coefficients'] = X_train.columns
coeff_df['Weights'] = model.coef_[0]
coeff_df['ABS_Weights'] = abs(coeff_df['Weights'])
```

In [68]:

```
#Sort the coeff in the order of their importance
coeff_df = coeff_df.sort_values(['ABS_Weights'], ascending=False)
```

Weights of features (coefficients)

In [69]:

```
#Display variable weights

coeff_df
```

Out[69]:

	Coefficients	Weights	ABS_Weights
2	int_rate	-0.172725	0.172725

	Coefficients	Weights	ABS_Weights
19	grade_A	0.160744	0.160744
1	term	-0.151026	0.151026
5	dti	-0.115873	0.115873
22	grade_D	-0.107174	0.107174
4	annual_inc	0.104148	0.104148
23	grade_E	-0.101850	0.101850
24	grade_F	-0.077018	0.077018
6	open_acc	-0.072491	0.072491
9	revol_util	-0.070610	0.070610
20	grade_B	0.066610	0.066610
18	home_ownership_RENT	-0.046934	0.046934
10	total_acc	0.045026	0.045026
14	home_ownership_MORTGAGE	0.040512	0.040512
0	loan_amnt	-0.039455	0.039455
8	revol_bal	0.038956	0.038956
11	mort_acc	0.030272	0.030272
21	grade_C	-0.028357	0.028357
3	installment	-0.018559	0.018559
17	home_ownership_OWN	0.010564	0.010564
25	grade_G	-0.010023	0.010023
13	home_ownership_ANY	0.007427	0.007427
16	home_ownership_OTHER	-0.003140	0.003140
15	home_ownership_NONE	0.000000	0.000000
12	pub_rec_bankruptcies	0.000000	0.000000
7	pub_rec	0.000000	0.000000

In [70]:

```
#Top 5 important features

coeff_df.head(5)
```

Out[70]:

	Coefficients	Weights	ABS_Weights
2	int_rate	-0.172725	0.172725
19	grade_A	0.160744	0.160744
1	term	-0.151026	0.151026
5	dti	-0.115873	0.115873
22	grade_D	-0.107174	0.107174

In [71]:

```
#Logistic Regression model intercept

model.intercept_
```

Out[71]:

```
array([1.56189014])
```

Confusion Matrix

In [72]:

```
#Create confusion matrix and print the matrix

cm = confusion_matrix(y_test, y_pred)
cm_df = pd.DataFrame(cm, index=np.unique(y_test), columns=np.unique(y_test))
```

In [73]:

```
cm_df
```

Out[73]:

	0	1
0	4	922
1	4	3793

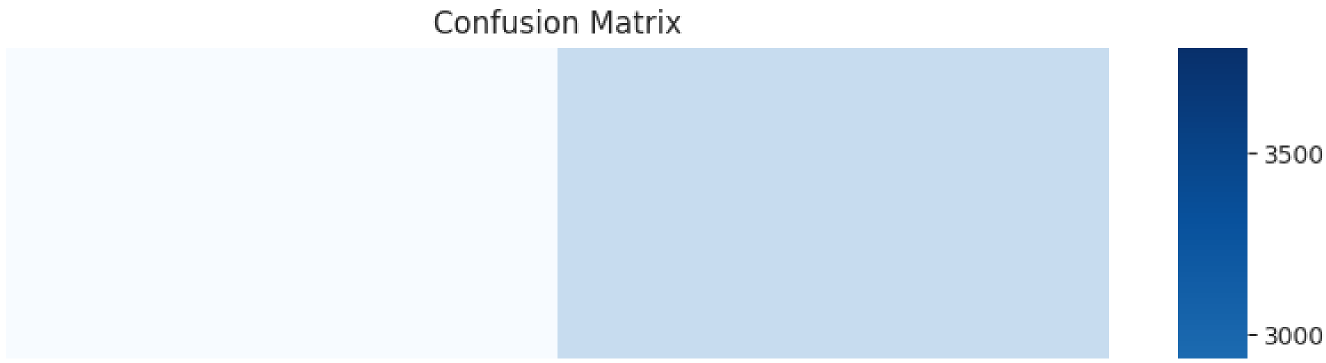
Class 0 : Charged Off (Here considering as negative class)

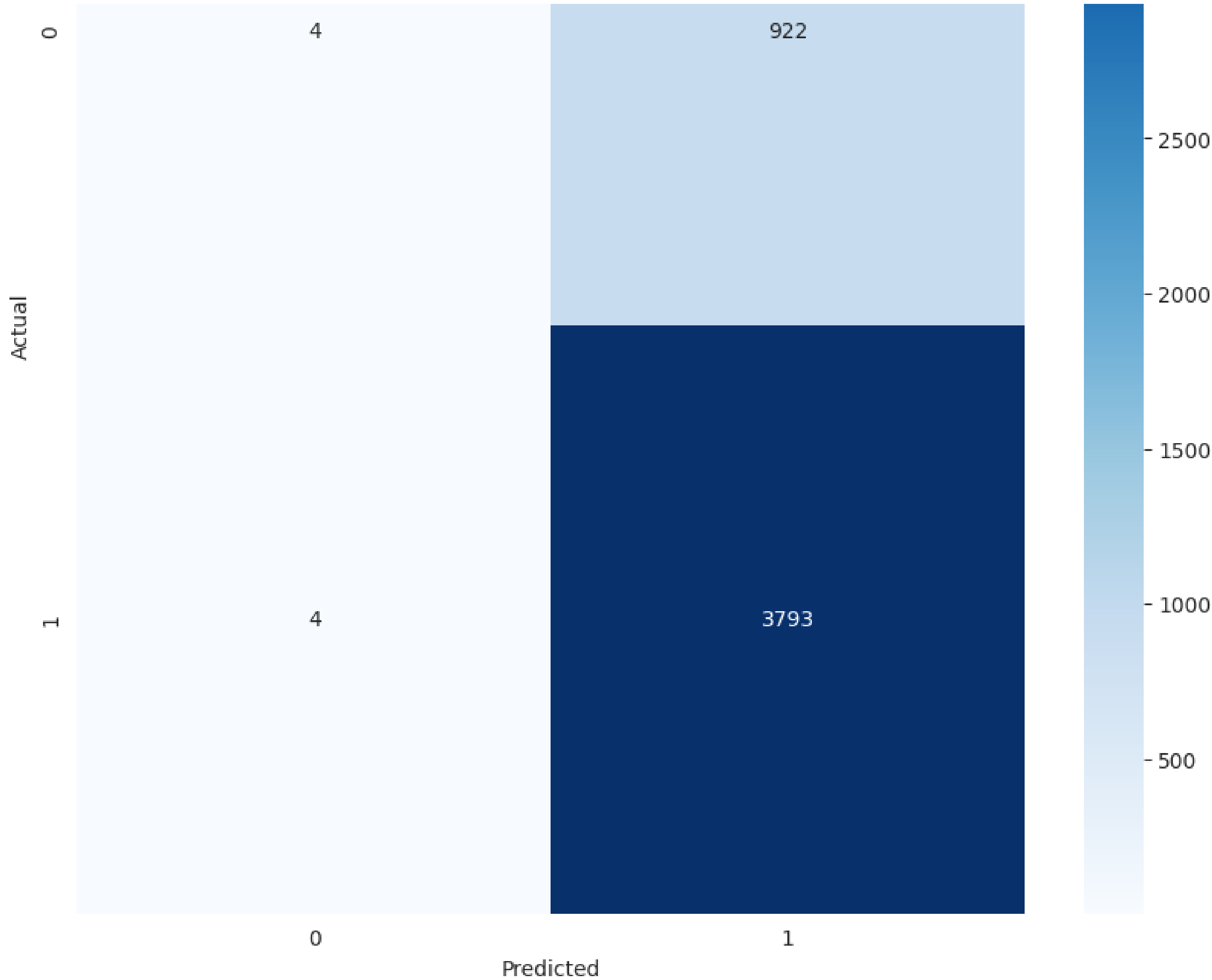
Class 1 : Fully Paid (Here considering as positive class)

1. **TN = 471**
2. **TP = 45212**
3. **FP = 10774**
4. **FN = 337**
5. **Actual Negative (Charged Off) = 471 + 10774 = 11245**
6. **Actual Positive (Fully Paid) = 337 + 45212 = 45549**
7. **Predicted Negative (Charged Off) = 471 + 337 = 808**
8. **Predicted Positive (Fully Paid) = 10774 + 45212 = 55986**

In [80]:

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression

# Assuming you have already fitted the model with the training data
# model.fit(X_train, y_train)

# Predict on the test data
y_pred = model.predict(X_test)

# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix using seaborn
fig, ax = plt.subplots(figsize=(10, 10))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=ax)
ax.set_xlabel('Predicted')
ax.set_ylabel('Actual')
ax.set_title('Confusion Matrix')
plt.show()
```





Classification Report

In [81]:

```
#Print classification report

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

```
              precision    recall  f1-score   support

           0       0.50      0.00      0.01       926
           1       0.80      1.00      0.89      3797

    accuracy                           0.80      4723
   macro avg       0.65      0.50      0.45      4723
weighted avg       0.74      0.80      0.72      4723
```

Observations from classification report:

1. **Precision : 0.81**
2. **Recall : 0.99**
3. **F1-score : 0.89**
4. **Accuracy : 0.80**

In [82]:

```
print('Precision Score:', precision_score(y_test,y_pred).round(2))
print('Recall Score:', recall_score(y_test,y_pred).round(2))
```

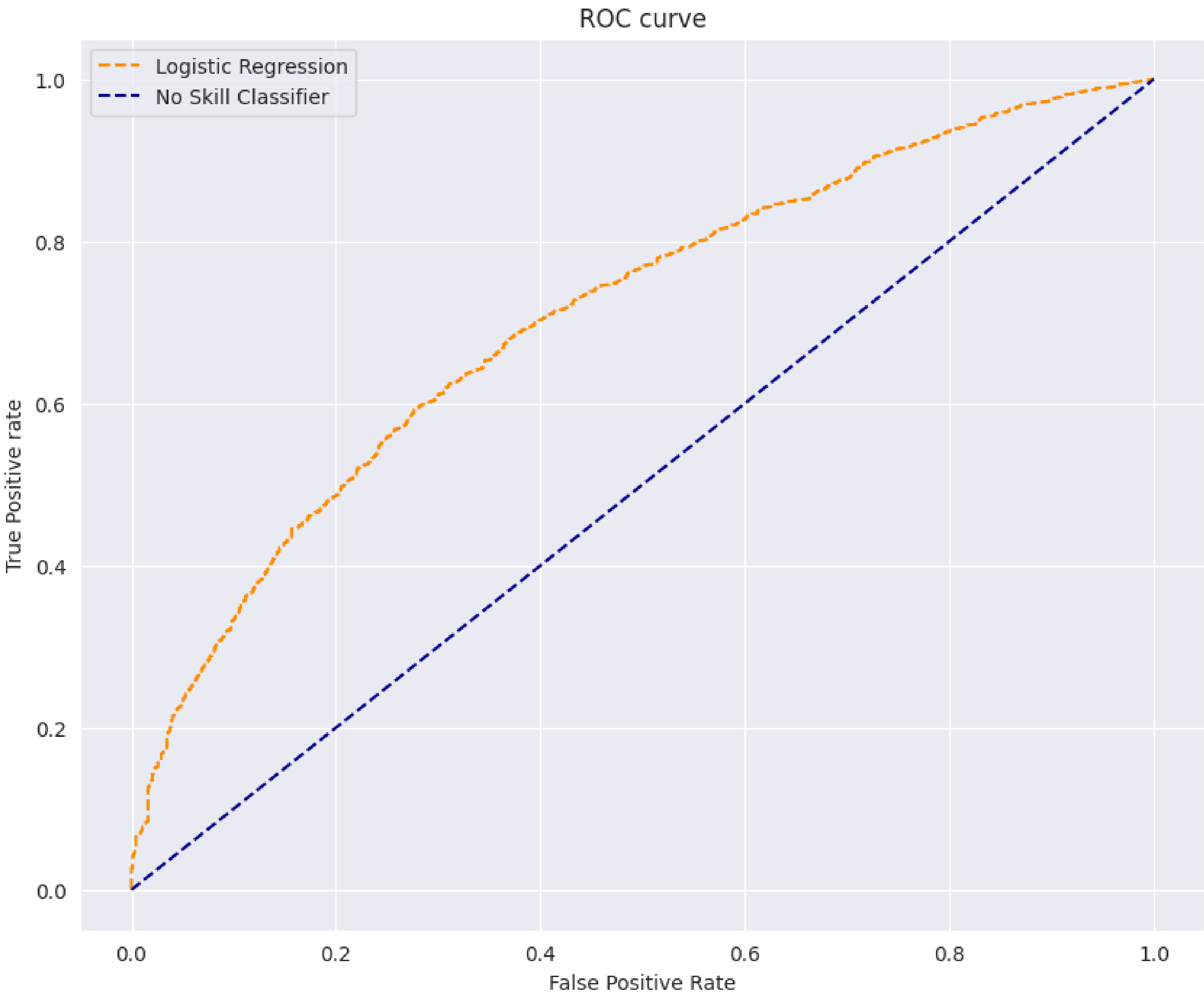
```
print('F1 Score:', f1_score(y_test,y_pred).round(2))
```

```
Precision Score: 0.8
Recall Score: 1.0
F1 Score: 0.89
```

ROC AUC Curve

In [83]:

```
#ROC Curve summarizes trade off between TPR and FPR

random_probs = [0 for i in range(len(y_test))]

p_fpr, p_tpr, _ = roc_curve(y_test, random_probs, pos_label=1)

fpr, tpr, thresh = roc_curve(y_test, y_pred_proba[:,1], pos_label=1)

plt.figure(figsize=(10,8))
plt.plot(fpr, tpr, linestyle='--',color='darkorange', label='Logistic Regression')
plt.plot(p_fpr, p_tpr, linestyle='--', color='darkblue', label='No Skill Classifier')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.show()
```



In [84]:

```
roc_auc_score(y_test, y_pred_proba[:,1]).round(2)
```

Out[84]:

0.71

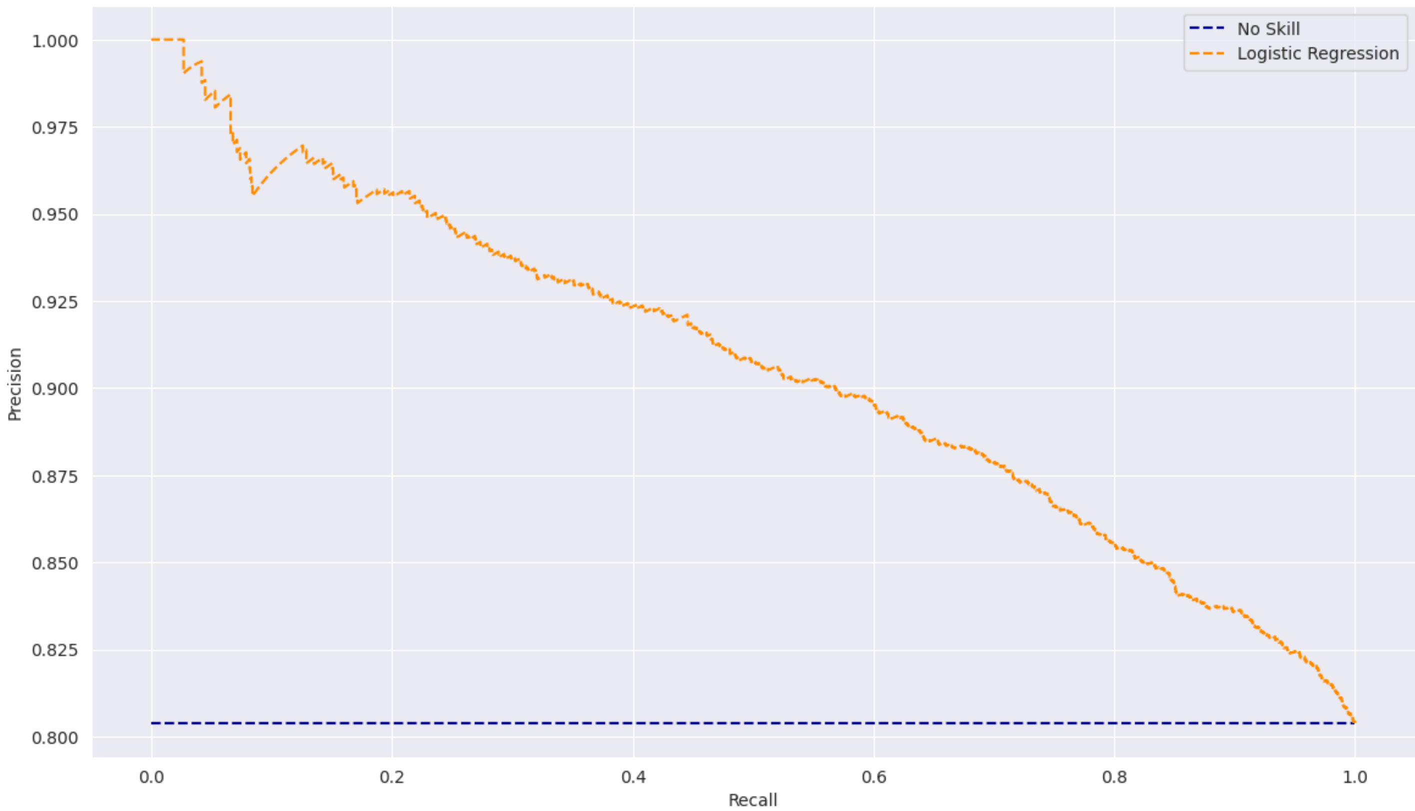
Observations: (Answers to trade off questions)

1. **Area under the ROC curve = 71%. That means we can say that the performance of the model is 0.71**
2. **Ideal scenario would be more TPR and lower FPR**
3. **Plot shows that True Positives increase at the cost of generating more False Positives**
4. **That means in order to find more Fully Paid customers, the model will have more chances of mistakenly classifying Charged Off customers as Fully Paid customers which might result in NPAs.**
5. **To avoid the NPAs, there is a necessity of bringing down the FPR while keeping the TPR in shape.**
6. **The model can detect the real defaulters when FPs (False Positives) are pushed towards left on x-axis**
7. **Once FPs (False Positives) towards left on X-axis the AUC will increase and hence the model performance**
8. **While FPs are moved towards left on X-axis, TPs need to remain high there on Y-axis**

Precision Recall Curve

In [85]:

```
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_proba[:,1])

no_skill = len(y_test[y_test==1]) / len(y_test)

plt.figure(figsize=(14,8))
plt.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill', color='darkblue
')
plt.plot(recall, precision, linestyle='--', label='Logistic Regression', color='darkoran
ge')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend(loc='best')
plt.show()
```



In [86]:

```
auc(recall, precision).round(3)
```

Out[86]:

0.906

0.906

Observations: (Answering Trade Off question)

1. **Precision recall curve is more useful in case of imbalanced data.**
2. **Calculation of precision and recall do not make use of the true negatives. So, it focuses on the correct prediction of one of the class. In our case that class is Class 1 i.e. Fully Paid customers**
3. **If you see the confusion matrix, the upper left box just won't be used in these calculations.**
4. **AUC = 90.4% which is failry good.**
5. **We can see that as the recall increases the precision is falling down.**
6. **For a strong model, both the recall and precision should be high**
7. **For a good trade off the precision needs to stay high on y-axis as recall progress towards right on x-axis**
8. **This shows that in order to increase the performance of model, precision needs to be improved**
9. **Increase precision means, there needs to be low FPs (False Positives)**
10. **So, here we need to focus more on reducing the FPs**

Extra analysis for questionaire and recommendations

In [88]:

```
#Loan Status for term = 60 months
print(df.loc[df['term']==' 60 months']['loan_status'].value_counts(normalize=True))

print('---------------')
#Loan Status for term = 36 months
print(df.loc[df['term']==' 36 months']['loan_status'].value_counts(normalize=True))

print('---------------')
#Loan status for grade A
print(df.loc[df['grade']=='A']['loan_status'].value_counts(normalize=True))

print('---------------')
#Median annual income of defaulters
print(np.percentile(df.loc[df['loan_status'] == 'Charged Off']['annual_inc'],50))

print('---------------')
#Median annual income of fully paid customers
print(np.percentile(df.loc[df['loan_status'] == 'Fully Paid']['annual_inc'],50))

print('---------------')
#Median dti ratio of Charged Off customers
print(np.percentile(df.loc[df['loan_status'] == 'Charged Off']['dti'],50))

print('---------------')
#Median dti ratio of Fully Paid customers
print(np.percentile(df.loc[df['loan_status'] == 'Fully Paid']['dti'],50))

print('---------------')
print(df.loc[df['grade']=='E']['loan_status'].value_counts(normalize=True))

print('---------------')
print(df.loc[df['grade']=='D']['loan_status'].value_counts(normalize=True))

print('---------------')
print(np.percentile(df.loc[df['loan_status'] == 'Charged Off']['int_rate'],50))

print('---------------')
print(np.percentile(df.loc[df['loan_status'] == 'Fully Paid']['int_rate'],50))
```

```
Series([], Name: proportion, dtype: float64)
---------------
Series([], Name: proportion, dtype: float64)
---------------
loan_status
Fully Paid     0.944362
Charged Off    0.055638
Name: proportion, dtype: float64
```

```
--------------
55000.0
--------------
60000.0
--------------
18.98
--------------
16.43
--------------
loan_status
Fully Paid     0.638263
Charged Off    0.361737
Name: proportion, dtype: float64
--------------
loan_status
Fully Paid     0.700766
Charged Off    0.299234
Name: proportion, dtype: float64
--------------
15.58
--------------
12.69
```

Questionaire

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

Answer: 80.38%

Question 2: Comment about the correlation between Loan Amount and Installment features?

Answer: Loan amount and installment has very strong positive correlation of 0.95

Question 3: The majority of people have home ownership as __.

Answer: Mortgage i.e. 50.08%

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

Answer: True. Out of All people with grade A, 93.71% are Fully Paid and only 6.29% are Charged Off

Question 5: Name the top 2 afforded job titles.

Answer: Teacher and Manager.

Question 6: Thinking from a bank's perspective, which metric should our primary focus be on.. a. ROC AUC b. Precision c. Recall d. F1 Score

Answer:

1. **Bank's primary focus should be on ROC AUC**
2. **Because bank needs to reduce FPR (False Positive Rate) and needs to increase the TPR (True Positive Rate).**
3. **In common man's term, Bank should not classify Charged Off customers as Fully Paid i.e. False Positives**
4. **And bank should not classify Fully Paid customers as Charged Off i.e. False Negatives**

Question 7: Which were the features that heavily affected the outcome?

Answer: Top 5 Features that affected the outcome are -

1. **Grade**
2. **Term**
3. **Annual income**
4. **dti**
5. **int_rate Question 8: Will the results be affected by geographical location? (Yes/No)**

Answer: No. The results will not be affected by the geographical locaion. See the bar graph plotted above.

Business Recommendations

1. **Customers with Grade A are the most reliable on the repayments. Bank can extend the credit line to these customers and should focus and adding more new customers to list of borrowers. 93% of these have a track record of repaying their loan.**
2. **The term period of 60 months is a trouble when it comes to Charged Off accounts. 32% of accounts from 60 months term period turned into NPA based on the data available. So, here needs to rethink on the repayment terms.**
3. **The median annual income of Charged Off customers is 59K which is 6K less than median annual income of Fully Paid customers (65K). Please revisit the annual income thresholds while extending the credit lines to the customers.**
4. **The median dti ratio of Charged Off customers is 19.34 which is 3 points higher than the fully paid customers. Please give it a thought. This feature tops in first 5 most impactful features.**
5. **37% of the grade E and 28% of the grade D customers are defaulters from historical data. The needs to put more stringent criteria and the grade E and D customers.**
6. **Median interest rates of defaulter customers are 2.62% higher than those of regular. Median interest rate of regualr customers is 12.99% and for defaulters it's found that median interest rate is 15.61%. If the customer interest rates crawl above the alarming thresholds then that account is more probably more prone to become an NPA**
7. **Apart from this, the bank needs to focus more on improving the precision of correctly identifying the Charged Off customer. Becuase the current historical data trend shows that the bank is not so accurate in classifying the Charged Off customers. However these customers often get the green pass as a result of high FPR (False Positive Rate).**