PROJECT REPORT

BANK LENDING

Submitted towards the partial fulfillment of the criteria for award of PGA by Imarticus

Submitted By:

KAUSHAL KISHORE (IL014489)

Pro Degree Genpact – DSP, MAR07

Abstract

Managing credit risk structure through historical data and behavior of customers and taking critical decisions to fund loans to respective customers. A thorough study through past data would help XYZ organization to infer whom to fund loans and in what amount. The key idea here is to build a robust model by using various past behaviors which would help decide what's is best case scenario and to avoid any fraudulent activity in future.

Acknowledgements

I am using this opportunity to express my gratitude to everyone who supported us throughout the course of this group project. We are thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the project work. I am sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the project.

Further, we were fortunate to have **Gaurav** as our mentor. He has readily shared his immense knowledge in data analytics and guide us in a manner that the outcome resulted in enhancing our data skills.

We wish to thank, all the faculties, as this project utilized knowledge gained from every course that formed the PGA program.

We certify that the work done by us for conceptualizing and completing this project is original and authentic.

Date: April 21, 2020 KAUSHAL KISHORE

Place: Bangalore

Certificate of Completion

I hereby certify that the project titled **Bank Lending** was undertaken and completed under my supervision by KAUSHAL KISHORE from the batch of DSP, Mar 07.

Mentor: Gaurav

Date: April 21, 2020

Place – Bangalore

Contents

CHAPTER 1: INTRODUCTION

1.1 Understanding the problem statement

We have been provided with a task of managing credit risk for XYZ organization who passes loan for customer. for this task we have been given past data containing list of successful payers, defaulters etc. Using all this past & present information we have to build a robust model which would help in predicting the probability of default in future which will help organization in deciding whom to pass loan and whom not to.

1.2 Need of the Study

Whenever any individual/corporation applies for a loan from a bank (or any loan issuer), their credit history undergoes a rigorous check in order to ensure that whether they are capable enough to pay off the loan (in this industry it is referred to as credit-worthiness).
The issuers has to set up models and rules in place which take information regarding their current financial standing, previous credit history and some other variables as input and output a metric which gives a measure of the risk that the issuer will potentially take on issuing the loan. The measure is generally in the form of a probability and is the risk that the person will default on their loan (called the probability of default) in the future.
Based on the amount of risk that the issuer is willing to take (plus some other factors) they decide on a cutoff of that score and use it to take a decision regarding whether to pass the loan or not. This is a way of managing credit risk. The whole process collectively is referred to as underwriting.

1.3 Data Sources

The text file "XYZCorp_LendingData" contains complete loan data for all loans issued by XYZ Corp. through 2007-2015. The data contains the indicator of default, payment information, credit history, etc.

1.4 Tools & Techniques

Tools: Python (Jupyter 6.0.0, Spyder 3.3.6)

Techniques:

The data will be divided into train (June 2007 - May 2015) and test (June 2015 - Dec 2015) data. Will use the training data to build models/analytical solution and finally apply it to test data to measure the performance and robustness using various machine learning algorithms.

Understand the data
EDA and segmentation
Data cleaning
Feature Engineering
Model building
Testing and cross-validation
Final results
Recommendations

CHAPTER 2: DATA PREPARATION

We would now be focusing on preparing the given data to fit to a different model.

To achieve this, we will be looking at sequence of steps

2.1 Exploratory Data Analysis

2.1.1 Data import and Statistical Analysis

2.1.1.1 Importing preliminary Libraries

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

2.1.1.2 Reading data

```
os.chdir(r"C:\Users\ezkiska\Videos\Imarticus\Python\4th Week 28th and 29th Dec\29th Dec Practical  DT, RF & XG")
data = pd.read_csv('XYZCorp_LendingData.txt',sep="\t",low_memory=False)
data.head()
```

2.1.1.3 Analyzing various aspects of Data

```
data.shape
data.dtypes
data.info()
data.describe()
data.describe(include=object)
```

This step will help us give the statistical analysis of raw data like shape of data, data types, min & max values.

2.1.2 Data Cleaning

2.1.2.1 Checking for null values

```
data.isnull().sum()
```

It clearly shows that a lot of columns have huge amount of missing values in them which might impact our result so we will drop the columns which have huge chunk of nulls and treat the ones which have lower count of nulls.

2.1.2.2 Null Value treatment

There are various methods to treat null values like we can simply drop them if that is least important or if huge chunk of data in them is Null. There are other methods like for numerical data we can replace nulls with mean, median or mode of that feature. For object type i.e. date or string data we can see the frequency or previous attribute and replace the nulls.

```
null_cols = [x for x in data.count() < 855969*0.40]
data.drop(data.columns[null_cols], 1, inplace=True)
```

With the above code we are dropping columns which have more than 60% of data as nulls.

Once we do this, we will check the shape of dataframe and observe that now we have 53 columns only with which we will be going ahead.

For columns like - **mths_since_last_delinq, revol_util, tot_coll_amt, tot_cur_bal, total_rev_hi_lim, collections_12_mths_ex_med** We have found statistical parameters and replaced the null entities with mean, median or mode depending upon the nature of feature.
Eg -

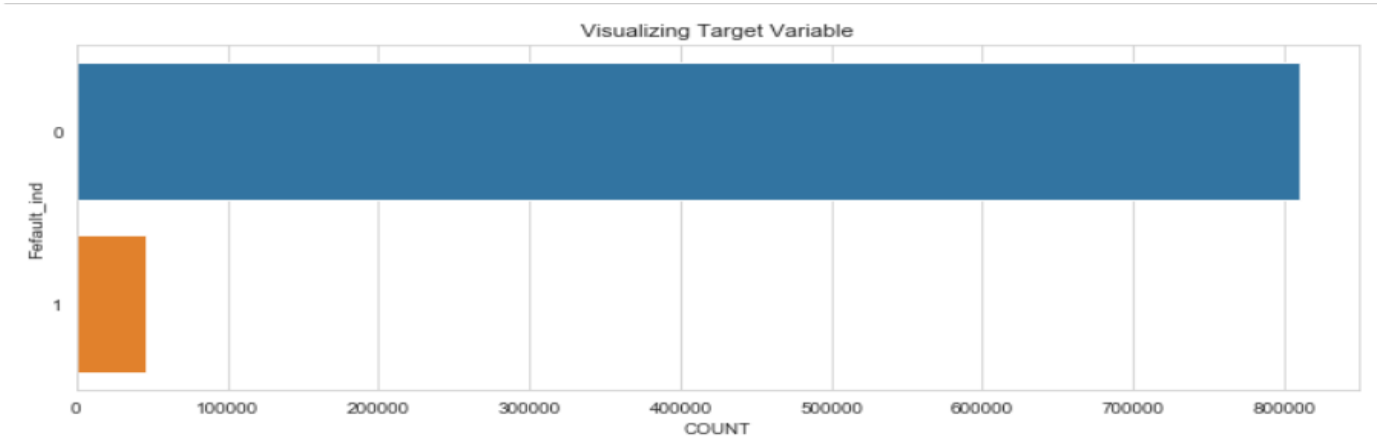
```
print (data.collections_12_mths_ex_med.min(), data.collections_12_mths_ex_med.max())
print(data.collections_12_mths_ex_med.mean())
print(data.collections_12_mths_ex_med.median())
print(data.collections_12_mths_ex_med.mode())
data.collections_12_mths_ex_med =
data.collections_12_mths_ex_med.fillna(data.collections_12_mths_ex_med.median())
```

Coming on to object types **last_pymnt_d, next_pymnt_d**, **title** we have used ffilll method which takes previous attribute and replace the null. Now after this processing we find only emp_title & emp_length to be containing null values as of now which we will treat later.

```
data["title"].fillna( method ='ffill', inplace = True)
```

2.1.3 Checking the target variable

```
plt.figure(figsize= (12,4))
sns.set_style("whitegrid")
sns.countplot(y='default_ind',data=data)
plt.title("Visualizing Target Variable")
plt.xlabel('COUNT')
plt.ylabel('Fefault_ind')
plt.show()
```

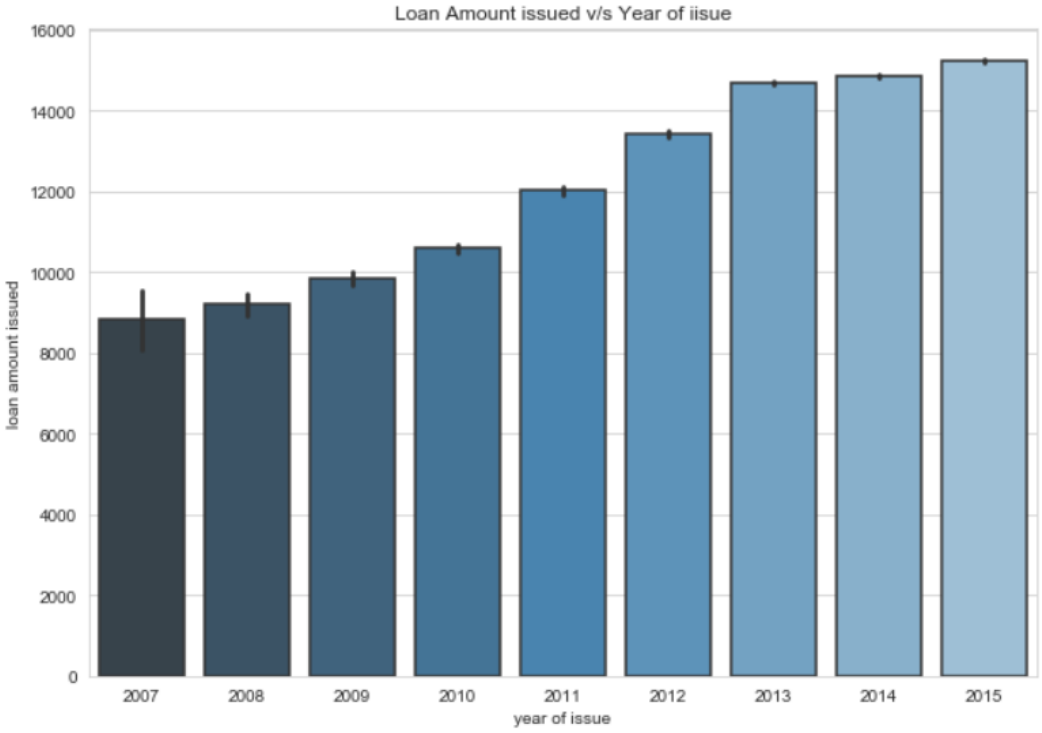


We observe from the target variable that data is highly imbalanced i.e. there is huge difference between count of 1's & 0's.Thus also depicting that we have to used imbalance treatement methods later to reduce chances of bias. Also methodes like cross validation or grid search will be used to proceed further.

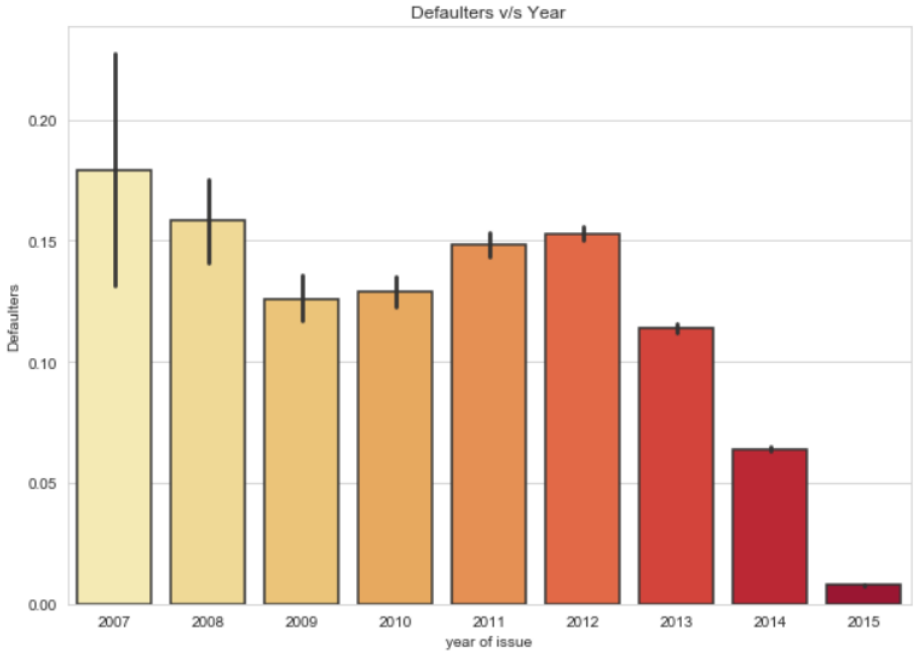
2.1.4 Data Visualization

Since Loan is an important parameter to check if who defaulted hence we will see the relation of loan with some of columns which are important considering the banking domain.

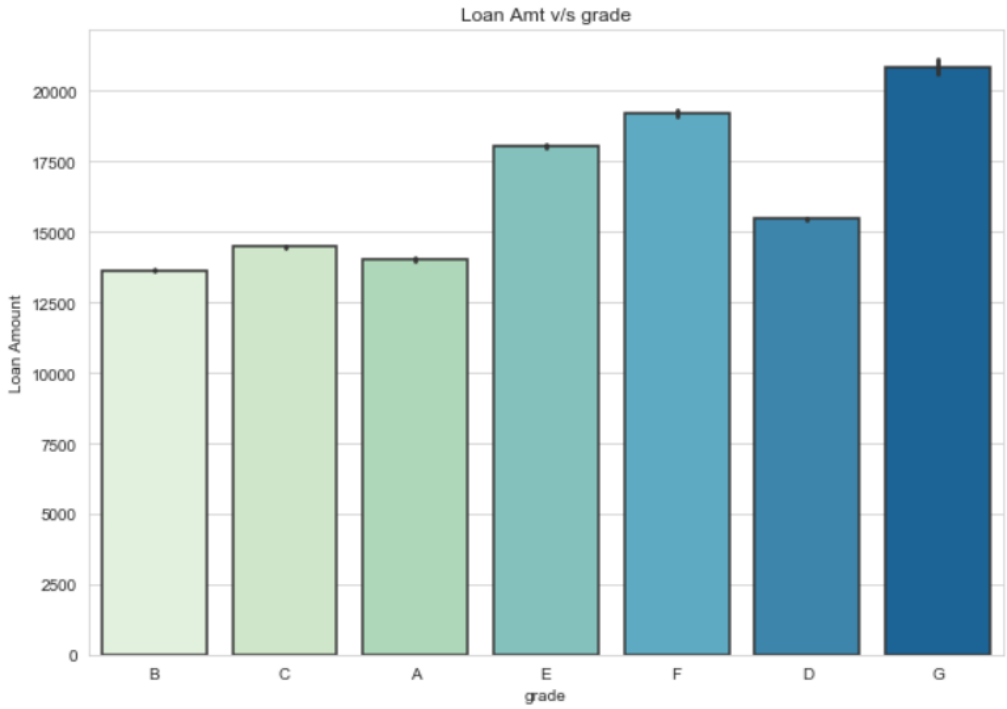
#First we plot **loan issued wrt year** and found out from below graph that with each year the amounty of loan issued increased.



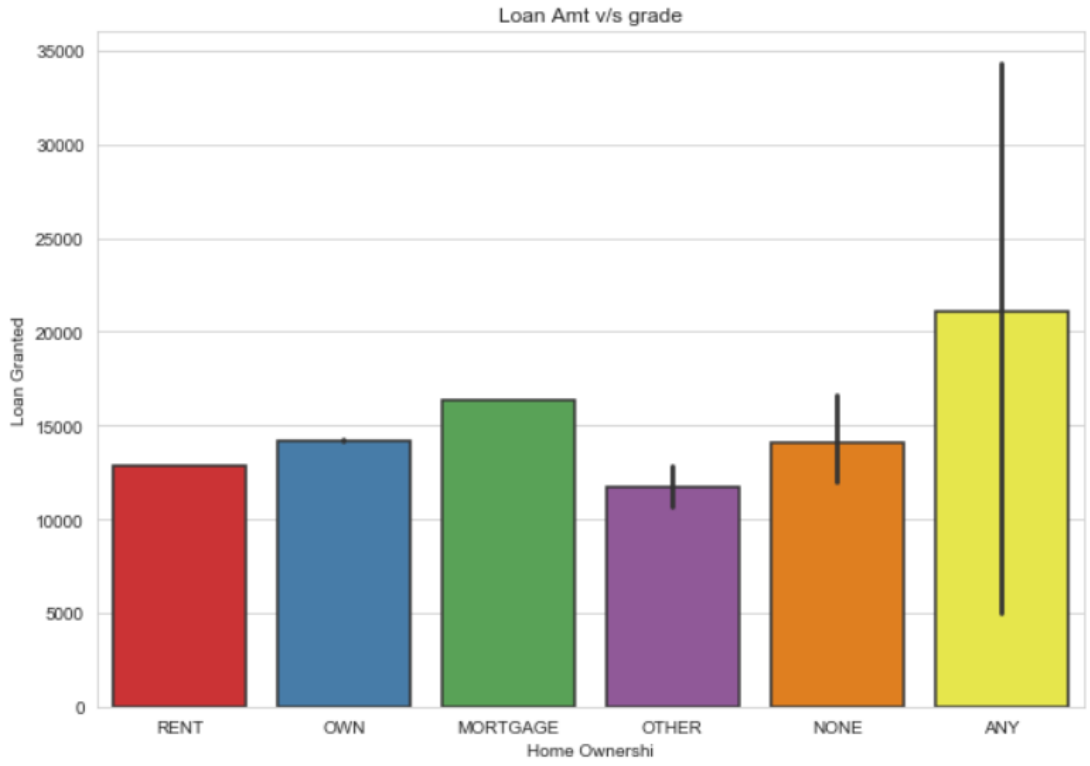
#Now we Check **Defaulter wrt Year** and found somewhat decreasing trend



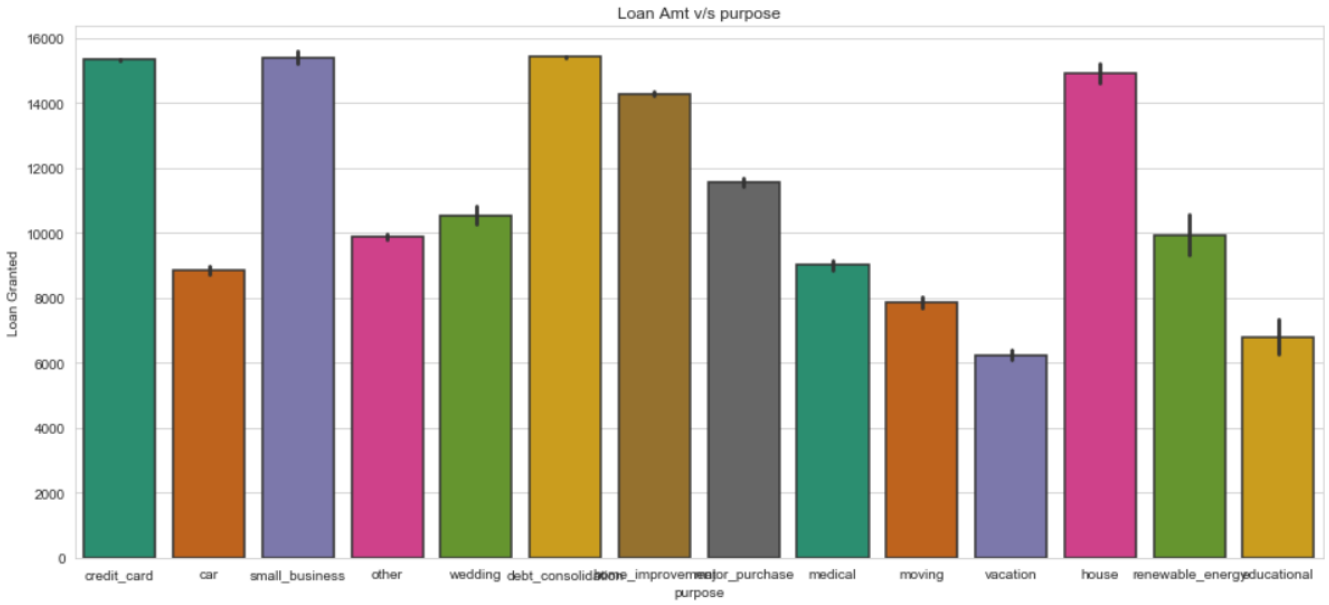
WE observe relation between **Loan amount wrt Grade**, there is no substantial conclusion regarding trend. Least amount was issued to thodse having grade B & highest to grade G



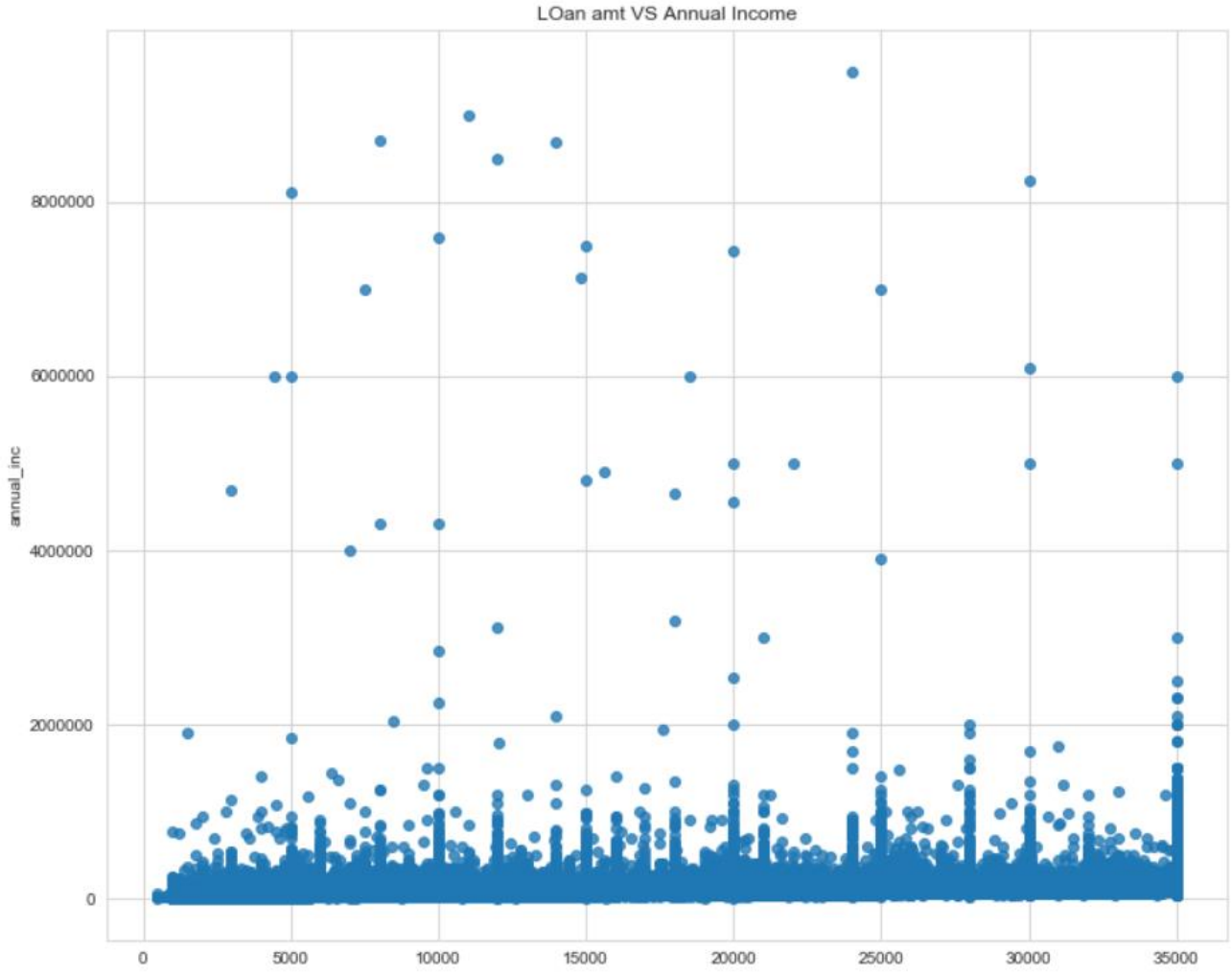
Next we see relation between **Home Ownership & Loan Amount** We see that Any & Mortgage were the resons to have been given maximum loan.



Now We will see Column **Purpose wrt Loan Amount**. We observe that house, credit card, small business & debt consolidation had the highest share. Vacation, education & car had the least share. Now well see graphically the relation.



Lets check **Annual Income wrt Loan amount** via a regression plot. Although it does not signify as all income class were scattered among various loan amounts. Max people who opted for loan had their income below 200000.



2.2 Data Seperation & Visualization

We will be seperating data into numerical and categorical columns so a to process them individually. Basis of seperation would be datatypes. For Numerical data we will be considering data type INT 64 and Float 64 while for categorical we will be taking Object data type.

2.2.1 Getting Numerical Data

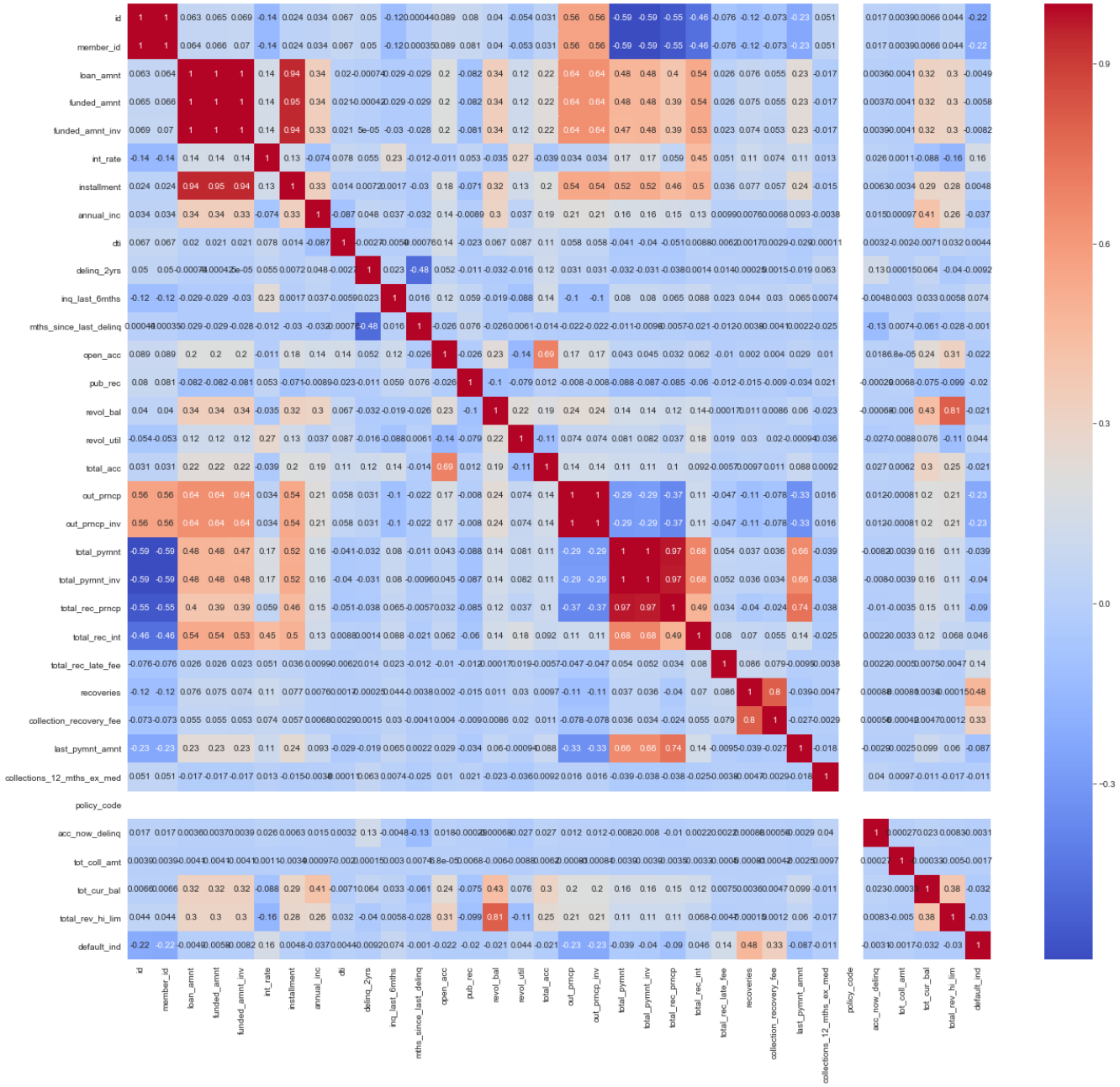
data_num = data.select_dtypes(include = ['float64', 'int64'])
data_num.shape

***o/p* - (855969, 34)**

So we have a total of 34 numerical columns.

2.2.2 Colinearity & Mullticolineraity

We plot heat map for numerical features to check for colinearity.



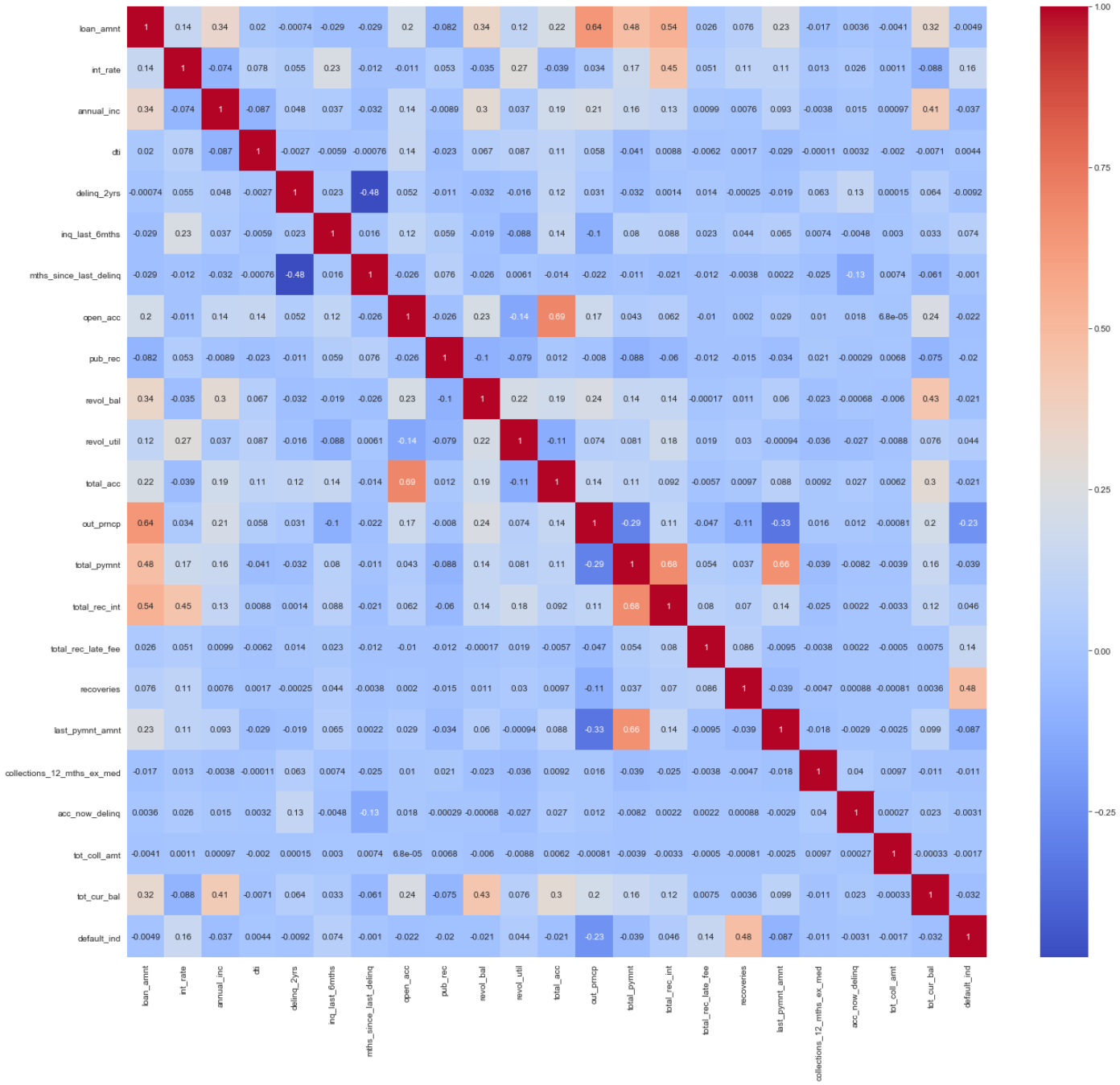
Now we will check for muticolineraity with below code, and drop the columns whose extent of multicolinearity is above 80%

```
corr_matrix = data_num.corr().abs()
# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
# Find index of feature columns with correlation greater than 0.95
to_drop = [column for column in upper.columns if any(upper[column] > 0.8)]
to_drop
data_num=data_num.drop(data_num[to_drop], axis=1)
```

Apart from this columns like ID & Policy Code doesnot infer anything, we will be droping them also

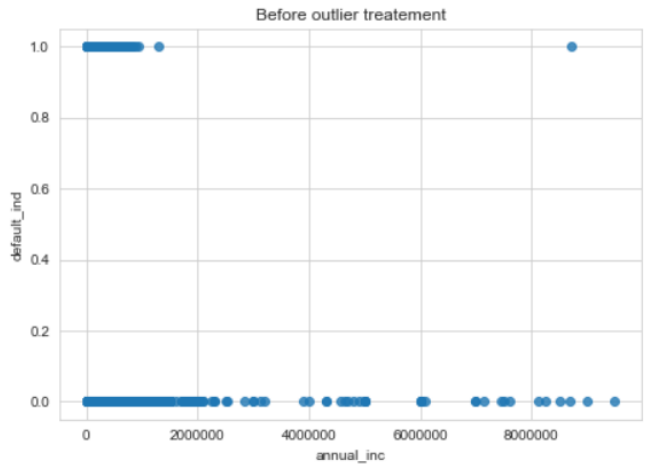
```
data_num.drop(['id','policy_code'], axis = 1, inplace = True)
```

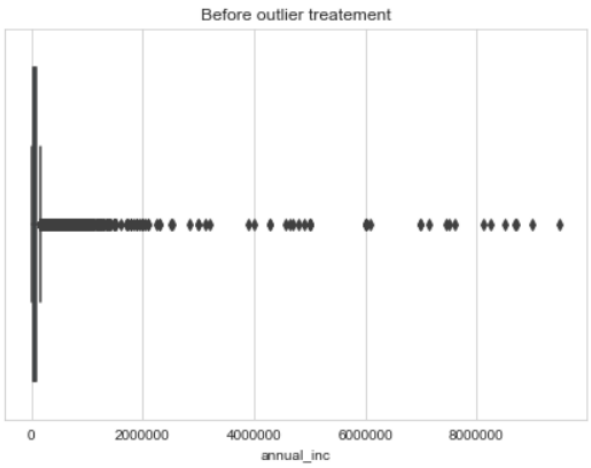
After droping above columns we have now only 23 columns left in numerical data. lets again check the heat map

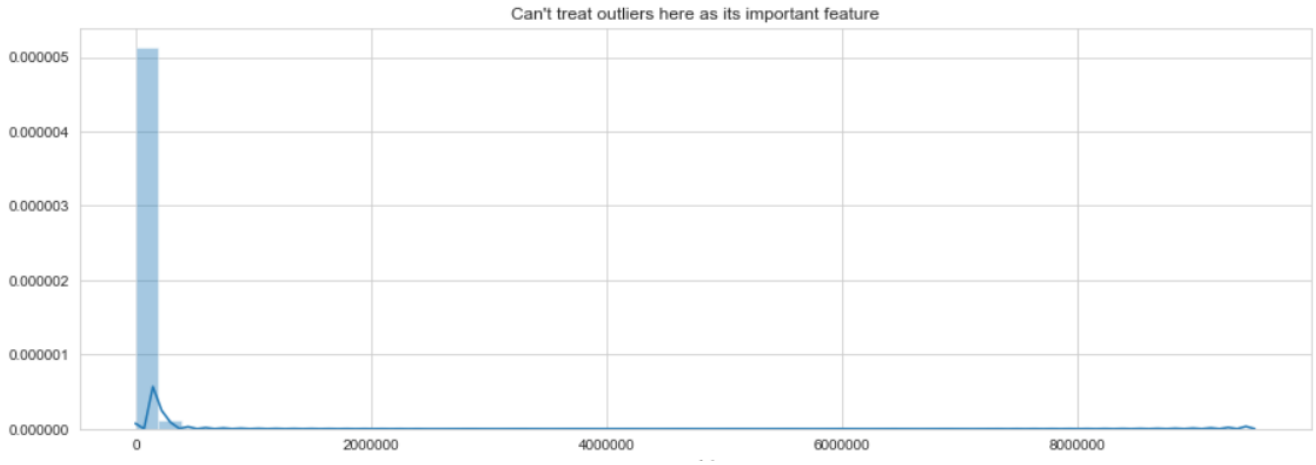


2.2.3 Columnwise visualization of Numerical Columns

#annual_inc: From below 3 graphs its is clear there are a lot of outliers but we wont be removing or treating them simply because that would fabricated result. Annual income is an important feature so we will keep it as it is.

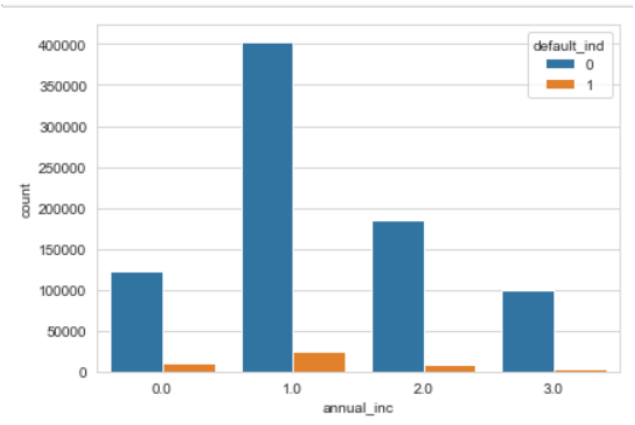




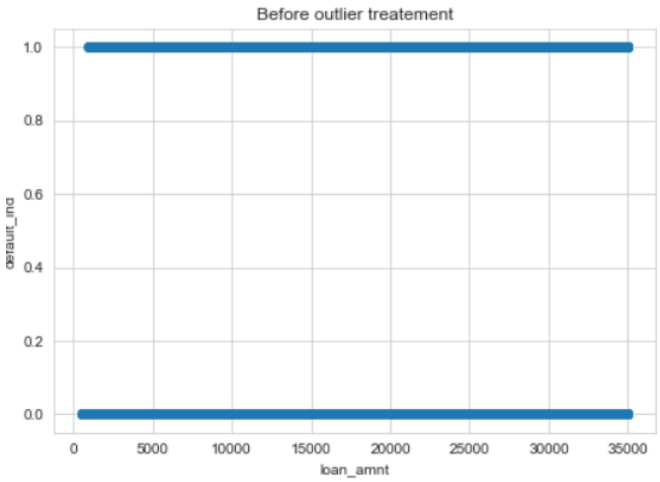


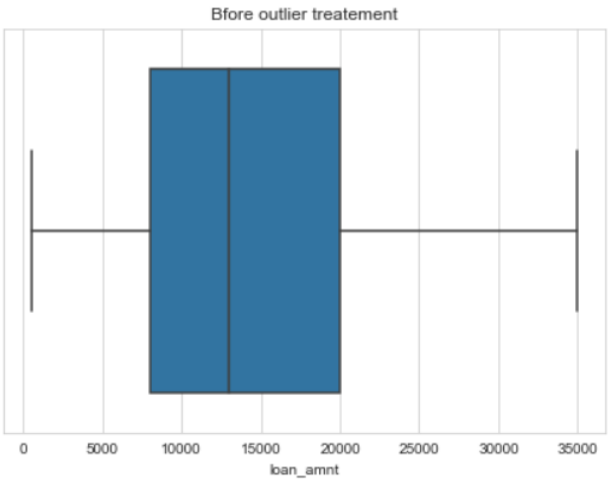
We will didvide it into various bands depending on range and check relation wrt target.

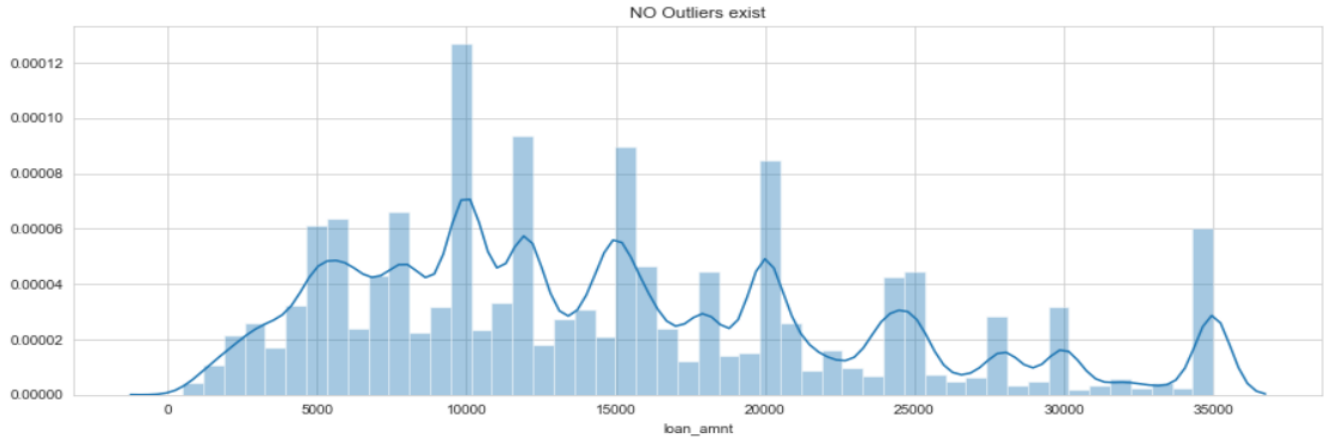
```
#creating bands
data_num.loc[data_num['annual_inc'] <= 39366.925, 'annual_inc'] = 0
data_num.loc[(data_num['annual_inc'] > 39366.925) & (data_num['annual_inc'] <=
78733.85), 'annual_inc'] = 1
data_num.loc[(data_num['annual_inc'] > 78733.85) & (data_num['annual_inc'] <=
118100.775), 'annual_inc'] = 2
data_num.loc[data_num['annual_inc'] > 118100.775, 'annual_inc'] = 3
data_num['annual_inc'].value_counts()
sns.countplot(x='annual_inc',hue='default_ind',data=data_num)
plt.tight_layout()
```



#loan_amnt- We observe from boxplot, reg plot & dist plot that there are no outliers here hence we need no treatement here.

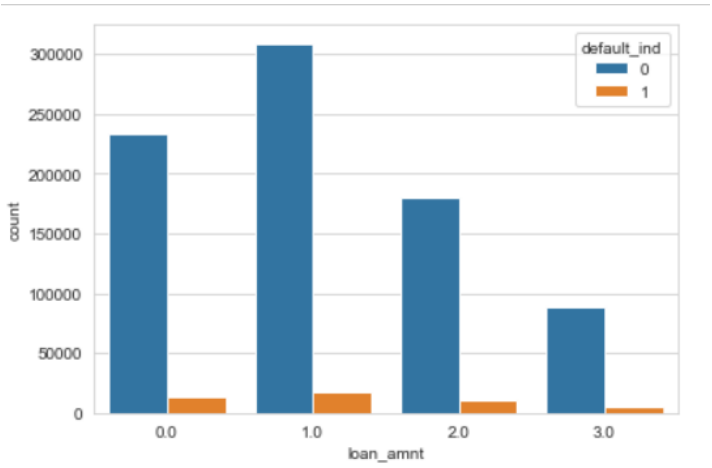




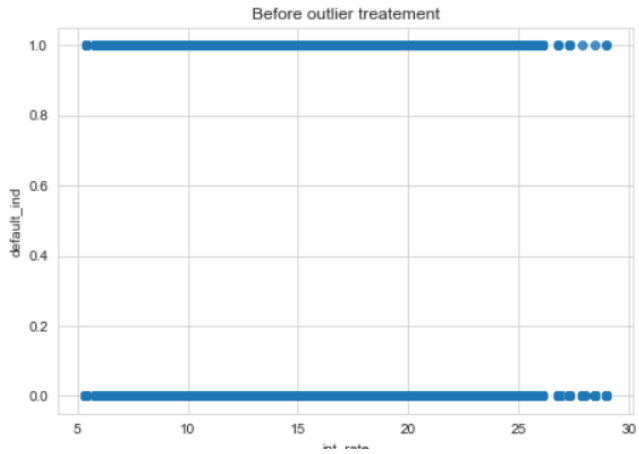


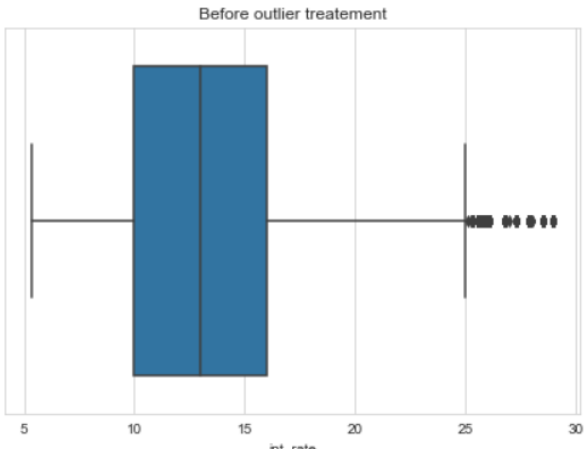
Now we wiill create bands and plot a countplot to see share of target variable in them.

```
data_num.loc[data_num['loan_amnt'] <= 9125.0, 'loan_amnt'] = 0
data_num.loc[(data_num['loan_amnt'] > 9125.0) & (data_num['loan_amnt'] <= 17750.0), 'loan_amnt'] = 1
data_num.loc[(data_num['loan_amnt'] > 17750.0) & (data_num['loan_amnt'] <= 26375.0), 'loan_amnt'] = 2
data_num.loc[data_num['loan_amnt'] > 26375.0, 'loan_amnt'] = 3
data_num['loan_amnt'].value_counts()
sns.countplot(x='loan_amnt',hue='default_ind',data=data_num)
plt.tight_layout()
```



#int_rate: From regression plot and boxplot it is very clear that there are certain outliers here which needs treatement here.



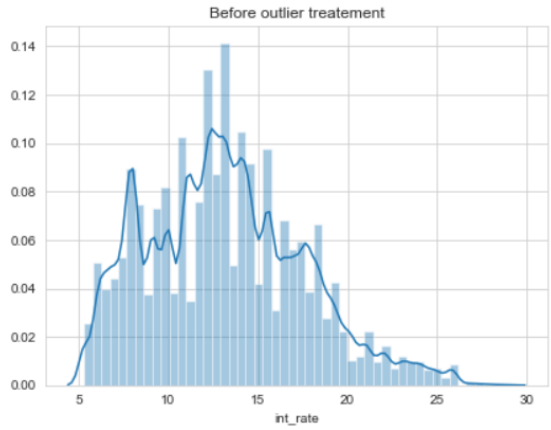


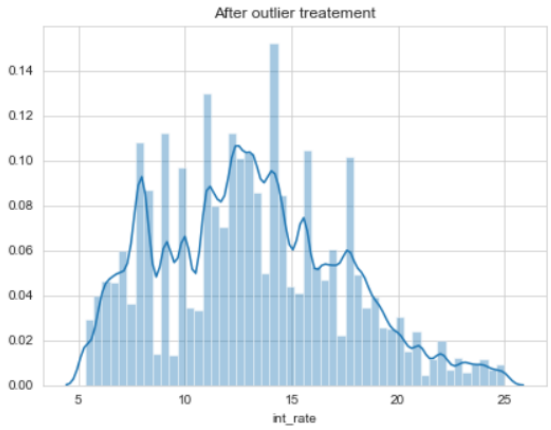
For outlier treatement we have ysed below methods-

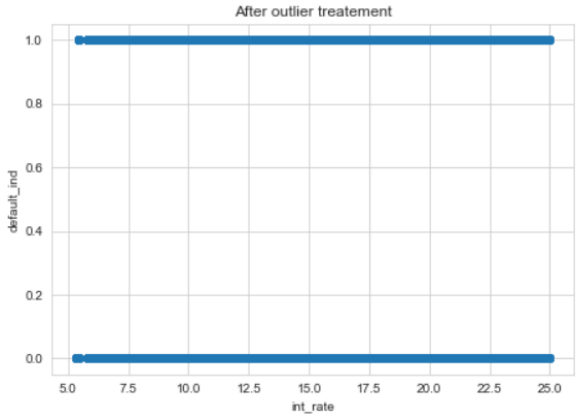
Q1=data_num['int_rate'].quantile(0.25)
Q3=data_num['int_rate'].quantile(0.75)
IQR=Q3-Q1
print(Q1)
print(Q3)
print(IQR)
*Lower_Whisker = Q1 - 1.5*IQR*
*Upper_Whisker = Q3+1.5*IQR*
print(Lower_Whisker, Upper_Whisker)

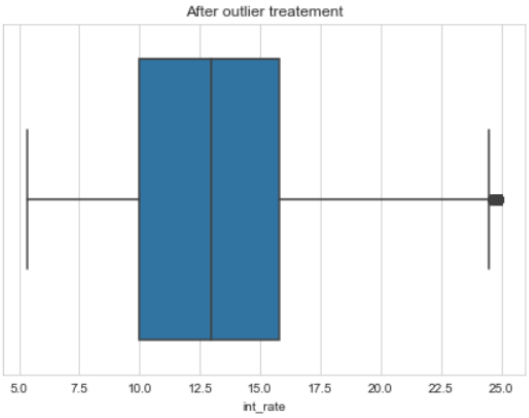
then ***data_num = data_num[data_num['int_rate']< Upper_Whisker]***

below distribution plot compare before and after outlier treatement.

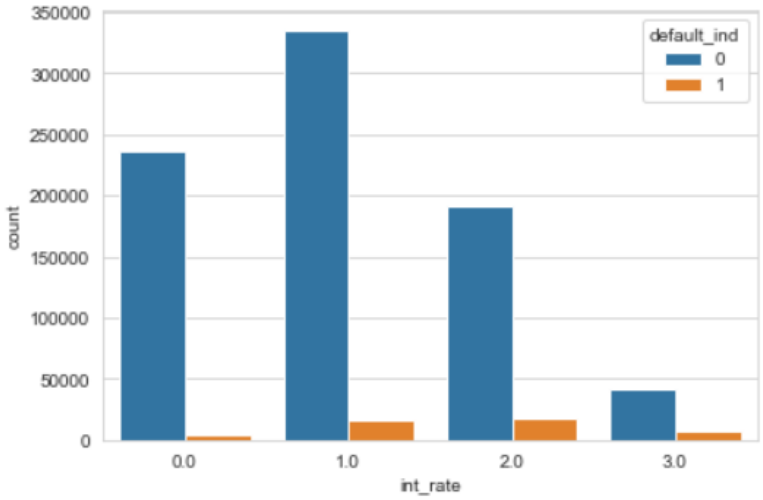




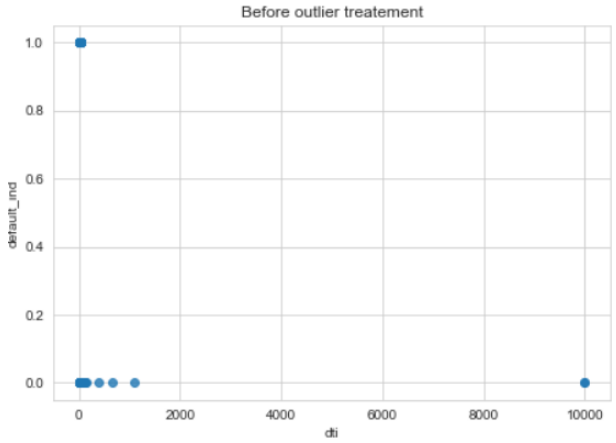


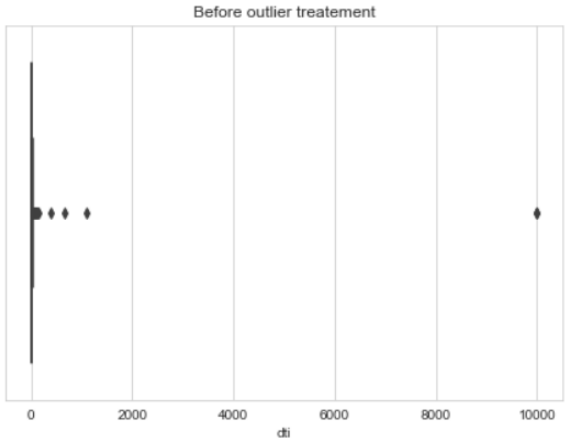


Now will divide it into bands and plot the countplot wrt target variable :

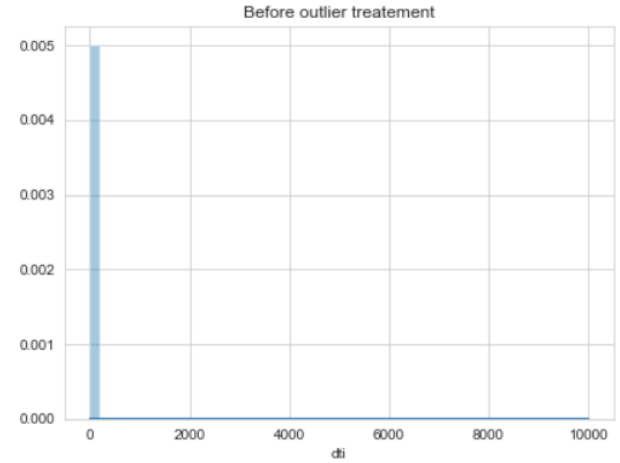


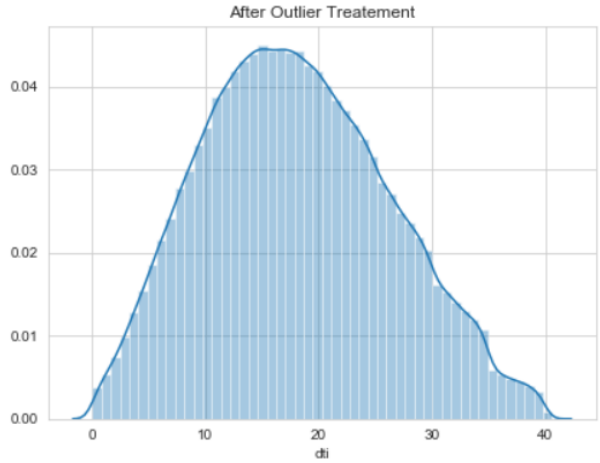
#dti – We observe a lot of outliers, hence would be doing outlier treatement and check for results



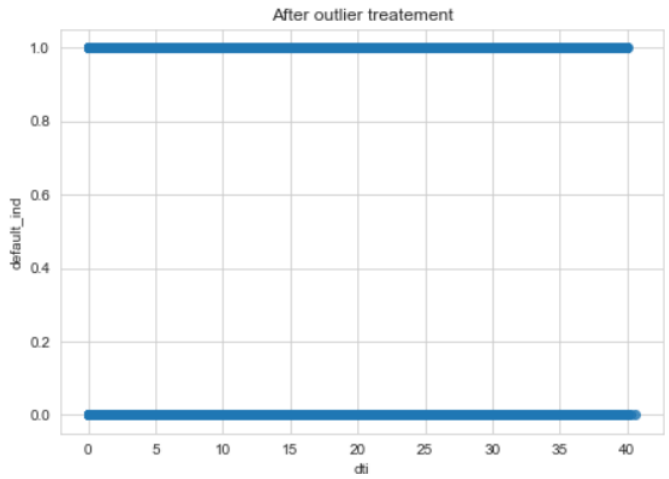


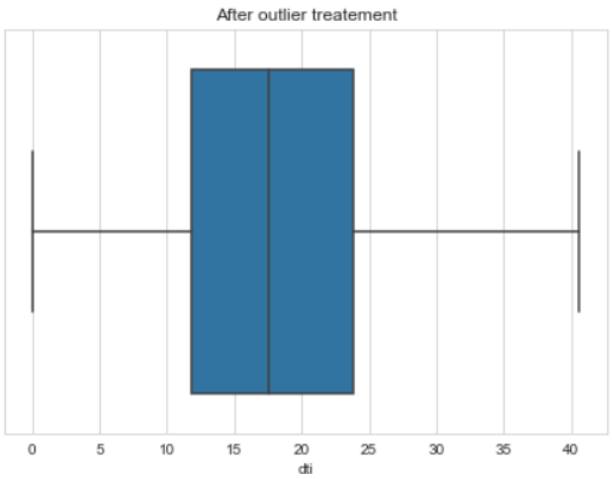

```
# Outliers Treatment
#Find mean of the column "dti"
dti_mean = int(data_num['dti'].mean())
IQR_dti_P75 = data_num['dti'].quantile(q=0.75)
IQR_dti_P25 = data_num['dti'].quantile(q=0.25)
IQR_dti = IQR_dti_P75-IQR_dti_P25
IQR_LL = int(IQR_dti_P25 - 1.5*IQR_dti)
IQR_UL = int(IQR_dti_P75 + 1.5*IQR_dti)
data_num.loc[data_num['dti']>IQR_UL , 'dti'] = dti_mean
data.loc[data['dti']>IQR_UL , 'dti'] = dti_mean
data_num.loc[data_num['dti']<IQR_LL , 'dti'] = dti_mean
data.loc[data['dti']<IQR_LL , 'dti'] = dti_mean
```





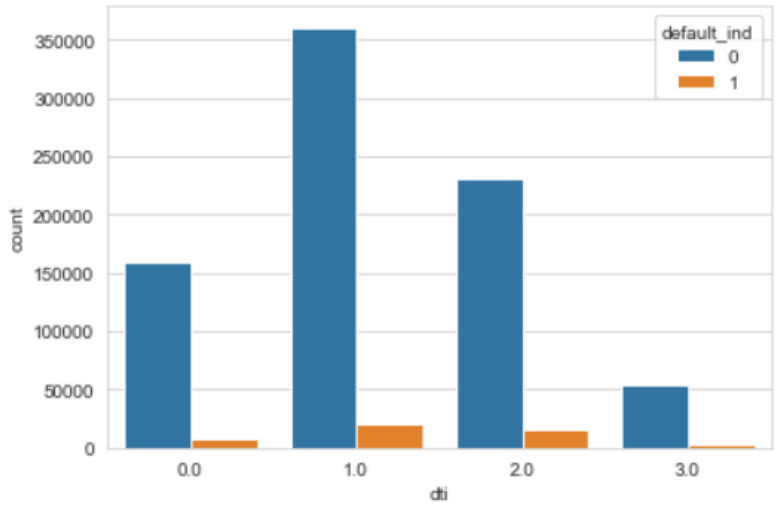
Now we again check reg plot & box plots :





Now we will ceate bands and do a countplot on them wrt target variable –

```
data_num.loc[data_num['dti'] <= 10.51, 'dti'] = 0
data_num.loc[(data_num['dti'] > 10.51) & (data_num['dti'] <= 21.02), 'dti'] = 1
data_num.loc[(data_num['dti'] > 21.02) & (data_num['dti'] <= 31.53), 'dti'] = 2
data_num.loc[data_num['dti'] > 31.53, 'dti'] = 3
data_num['dti'].value_counts()
sns.countplot(x='dti',hue='default_ind',data=data_num)
plt.tight_layout()
```



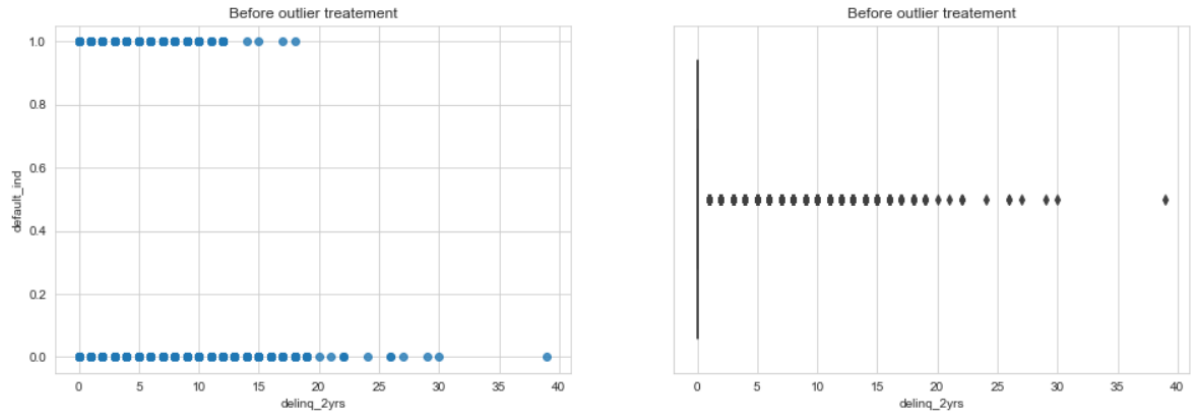
Just like dti we treated various other columns like **inq_last_6mths, mths_since_last_delinq, open_acc, revol_bal, revol_util, total_acc, out_prncp, total_pymnt, total_rec_int, recoveries, last_pymnt_amnt, tot_coll_amt & tot_cur_bal.**
Method of treatement of outliers in abpove columns was similar to what we used in dti. Aftre that we created bands and plotted a countplot wrt target variable as hue in them.

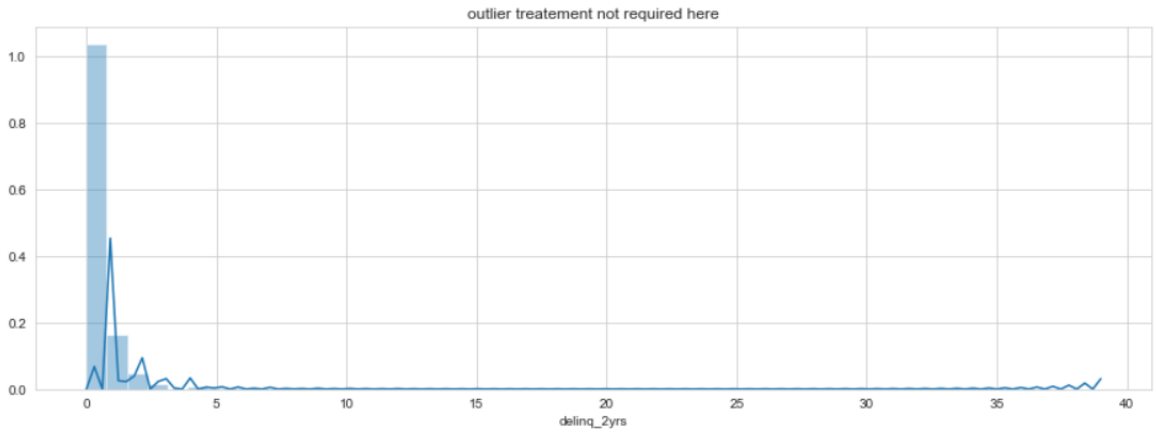
Apart from these columns there waere others too where outliers were there but we choose not to treat them becauise on nature of them and usage later. For eg in some columns binarization could solve the issue as there were not much unique values hence feasible to simply divide them into 0's & 1's.
Whuile some columns were related to fraud so we divided them into no fraud for casees equal to 0 and greater than cases as fraud.

Lets's see some of those columns :

#delinq_2yrs: Let's check the statistics of this particular column :
count 850051.000000, mean 0.311085, std 0.856071, min 0.000000, 25% 0.000000, 50% 0.000000 ,75% 0.000000 & max 39.000000

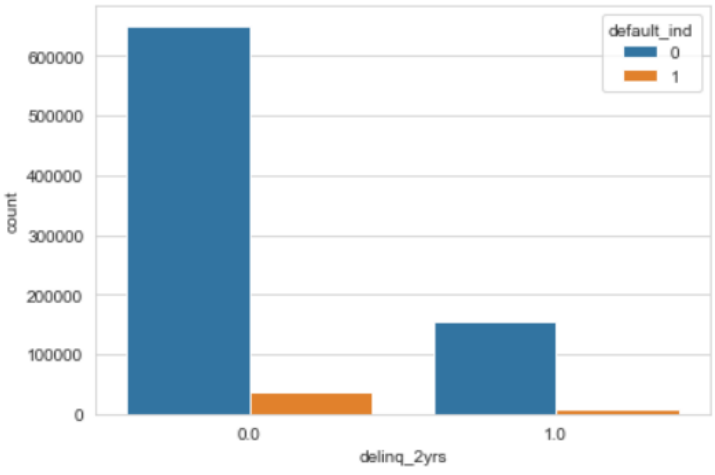
This clearly shows that there are not much unique values here.





So instead of treating outliers here we will simply binarize them i.e. create class of 0's & 1's and plot a count plot wrt target

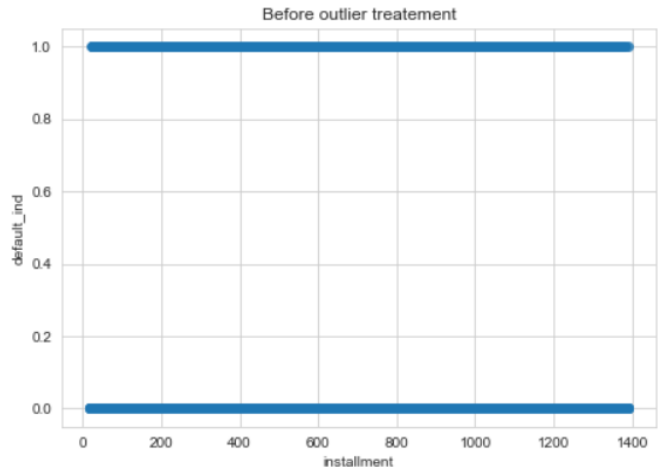
```
data_num.loc[data_num['delinq_2yrs'] <= 0, 'delinq_2yrs'] = 0
data_num.loc[data_num['delinq_2yrs'] > 0, 'delinq_2yrs'] = 1
data_num['delinq_2yrs'].value_counts()
sns.countplot(x='delinq_2yrs',hue='default_ind',data=data_num)
plt.tight_layout()
```

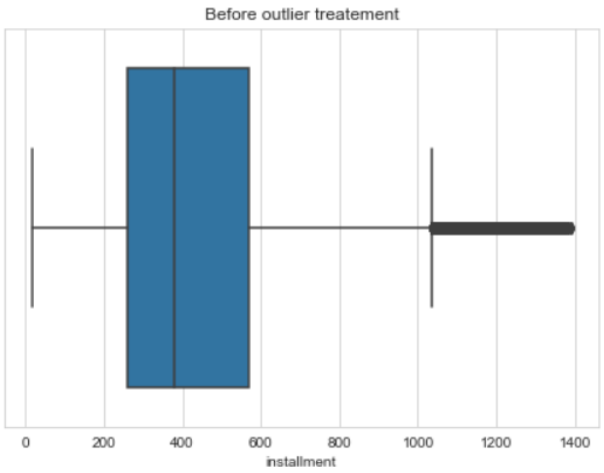


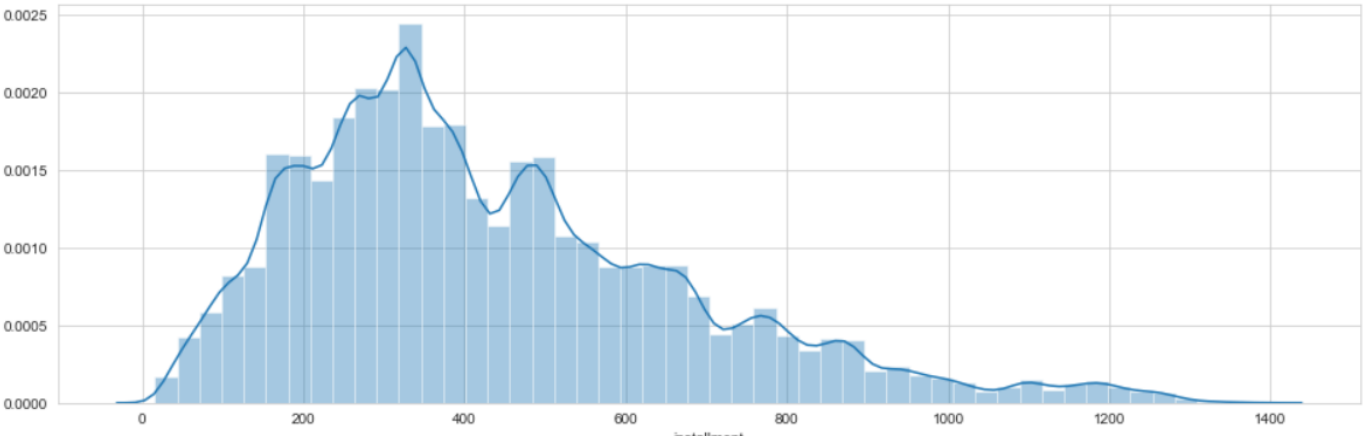
Similar approach will be followed for columns like - pub_rec, total_rec_late_fee, collections_12_mths_ex_med & acc_now_delinq,

Now we will take a look at 1 more column which we initially rejected but we will be considering forr model build i.e. unstallment. From domain knowledge it is preety clear that installment is an important feature while granting loan.

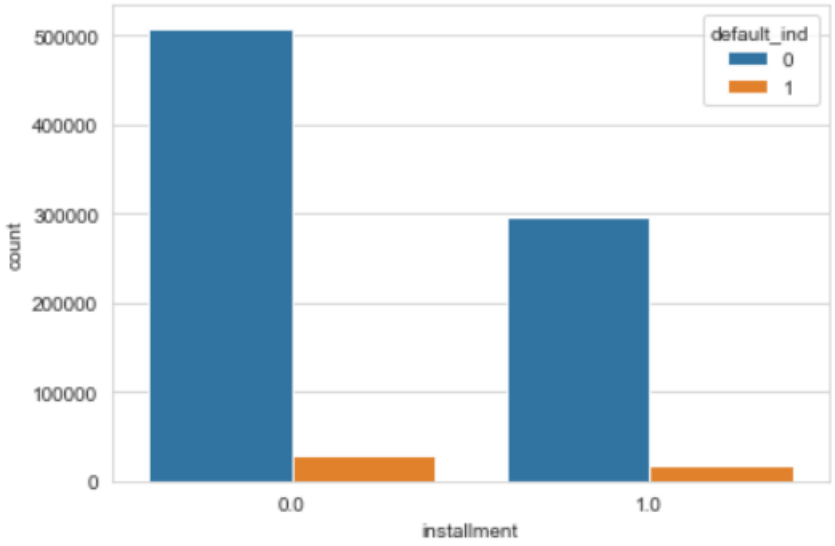
#installment: We oserve through installment box plot that there are few outlier sbut we won't be treating them as that would fabricate results, so simply create bands and plot the count plot wrt target.







```
data_num.loc[data_num['installment'] <= 474.263, 'installment'] = 0
data_num.loc[data_num['installment'] > 474.263, 'installment'] = 1
data_num['installment'].value_counts()
sns.countplot(x='installment',hue='default_ind',data=data_num)
plt.tight_layout()
```



Now we will drop the bands columns created in numerical data frame and check the shape of numerical data frame

```
data_num.drop(['annual_inc_band', 'loan_amnt_band', 'int_rate_band', 'dti_band',
'delinq_2yrs_bandvx', 'inq_last_6mths_band', 'mths_since_last_delinq_band', 'open_acc_ba',
    'pub_rec_ban', 'revol_bal_baad', 'revol_util_ban', 'total_acc_band','out_prncp_ban',
'total_pymnt_ban', 'total_rec_int_band','total_rec_late_fee_band', 'recoveries_band',
'last_pymnt_amnt_ban','collections_12_mths_ex_med_band', 'tot_coll_amt_band',
    'tot_cur_bal_bb', 'installment_bann'], axis = 1, inplace = True)

print(data.shape, data_num.shape)
o/p - (850051, 53) (850051, 24)
```

2.2.4 Columnwise visualization of Categorical Columns –

```
data_cat = data.select_dtypes(include = ['object'])
data_cat.shape
```

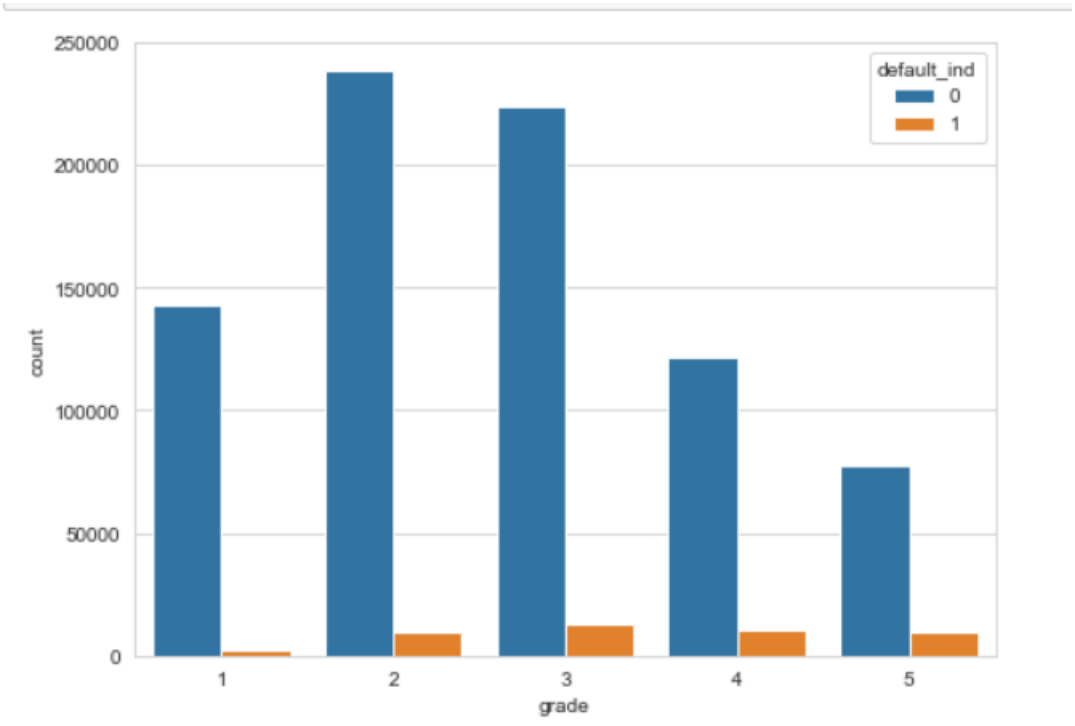
o/p- (850051, 19)

To get categorical columns from data frame we choose those data type which are of type object and then check shape of that. So here we have a total of 19 columns with us.

Lets check column wise –

#grade- So we see alphabetical grades which we convert to numerical which would help us in data prepartioon for model fitting. For this we use map function. Then we plot the count plot wrt target variable.

```
grade_map={'A':1, 'B':2, 'C':3, 'D': 4, 'E':5, 'F':5, 'G':5}
data_cat['grade']=data_cat['grade'].map(grade_map)
data['grade']=data['grade'].map(grade_map)
plt.figure(figsize=(7,5))
sns.countplot(x='grade',hue='default_ind',data=data_cat)
plt.tight_layout()
```



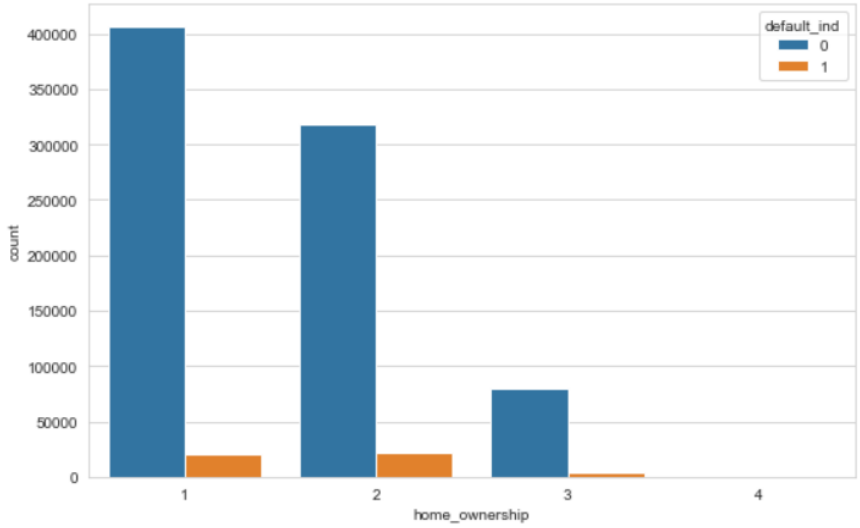
#sub_grade-

We check count of sub_grade and find a lot of unique values which would be difficult to convert to number so we will leave this as of now to be dummified or labelled encoded later.

data_cat['sub_grade'].value_counts()

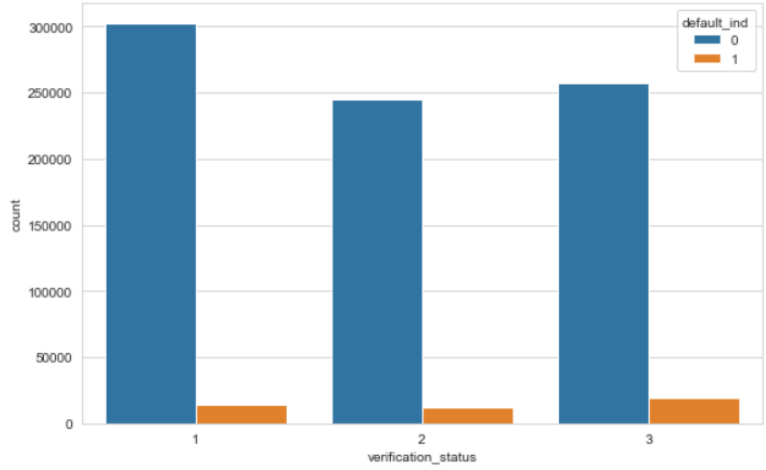
#home_ownership- Just like grade column we will map this column into numerical equivalent and plot the countplot.

home_map={'MORTGAGE':1, 'RENT':2, 'OWN':3, 'OTHER':4, 'NONE':4, 'ANY':4}
data_cat['home_ownership']=data_cat['home_ownership'].map(home_map)
data['home_ownership']=data['home_ownership'].map(home_map)
data_cat['home_ownership'].value_counts()
plt.figure(figsize=(8,5))
sns.countplot(x='home_ownership',hue='default_ind',data=data_cat)
plt.tight_layout()

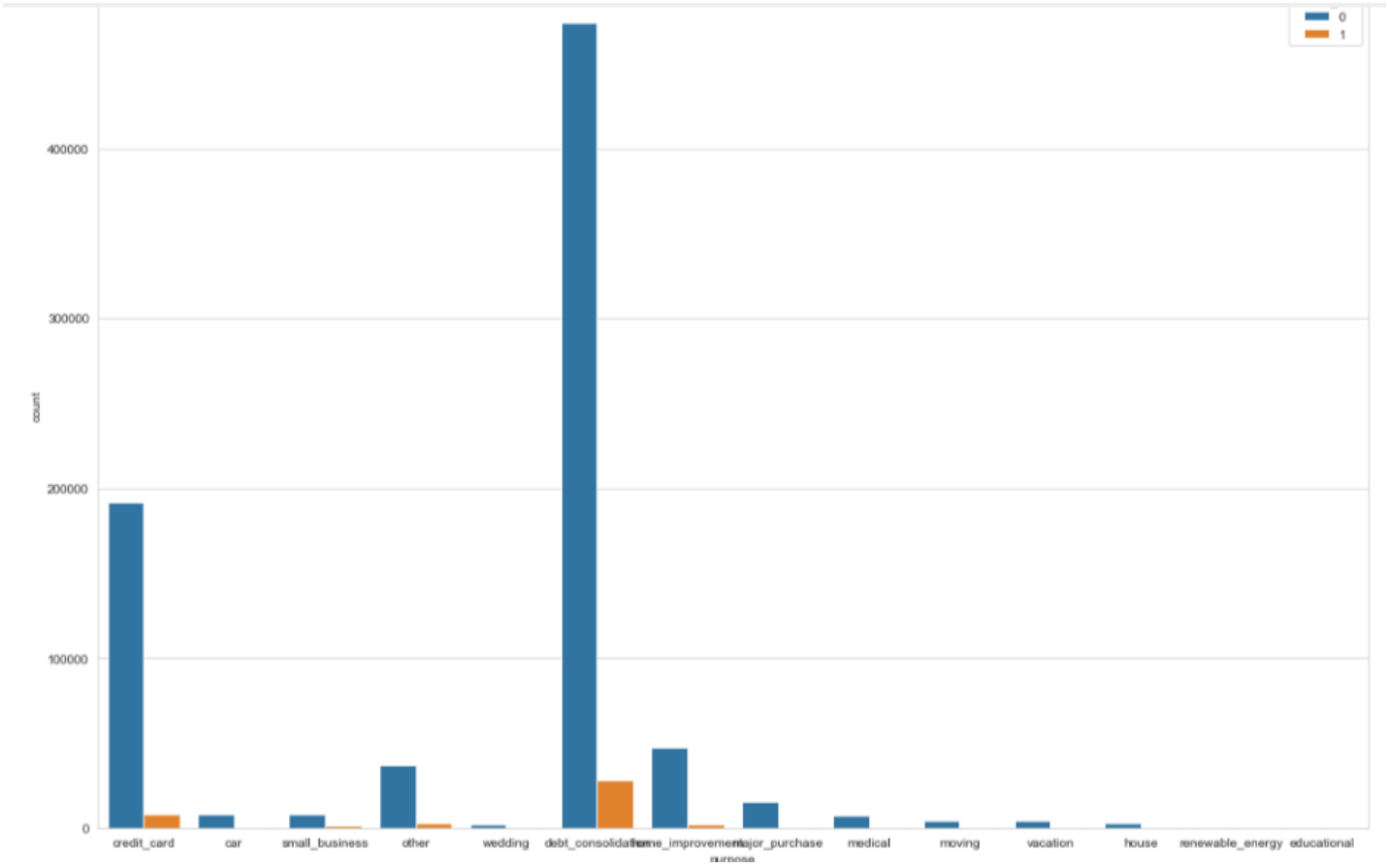


#verification_status- Again we map it into numerical equivalent and plot the countplot for same

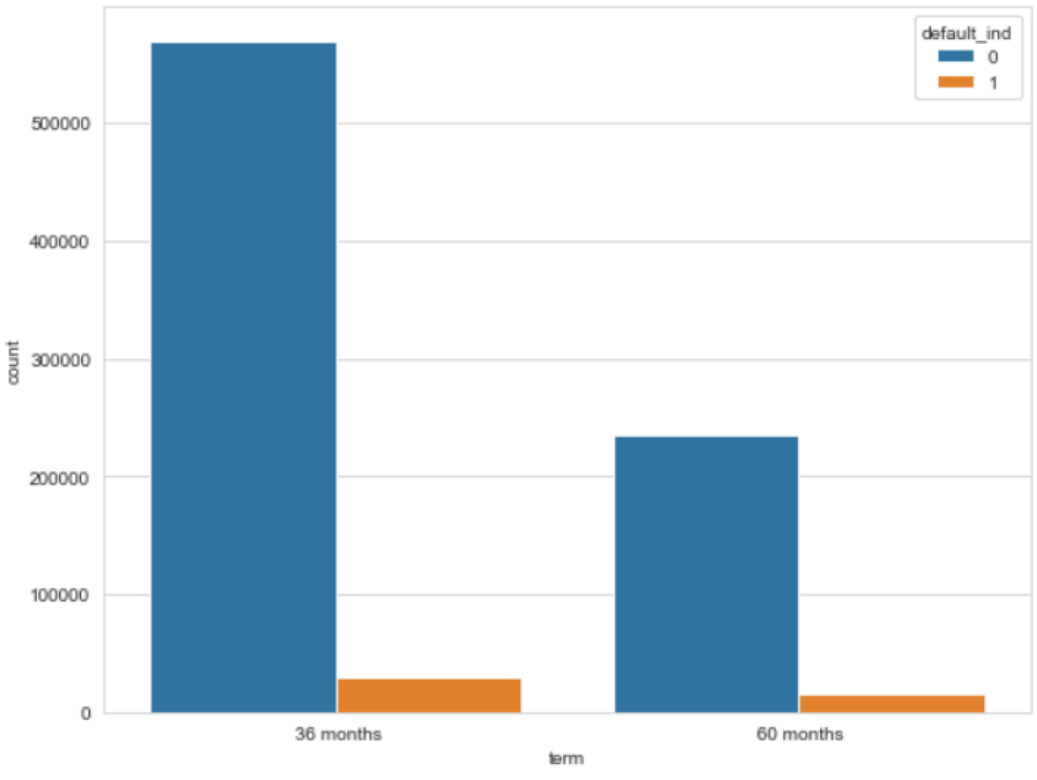
verify_map={'Source Verified':1, 'Not Verified':2, 'Verified':3}
data_cat['verification_status']=data_cat['verification_status'].map(verify_map)
data['verification_status']=data['verification_status'].map(verify_map)
data_cat['verification_status'].value_counts()
plt.figure(figsize=(8,5))
sns.countplot(x='verification_status',hue='default_ind',data=data_cat)
plt.tight_layout()



#purpose- on doing value counts we observe a lot of categories hence we will keep it fur dummy or label encoding rather than mapping.

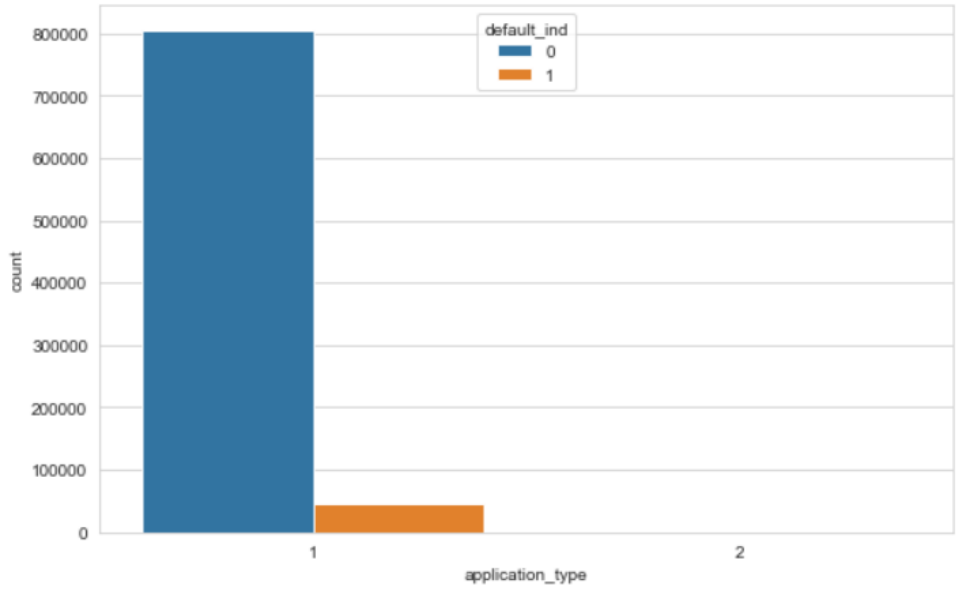


#term- For this column also we will keep it for dummy/label encoding and won't be converting to numerical .



#application_type- We have only 2 categories here, we will map them to numerical and plot the countplolt

app_map={'INDIVIDUAL':1, 'JOINT':2}
data_cat['application_type']=data_cat['application_type'].map(app_map)
data['application_type']=data['application_type'].map(app_map)
data_cat['application_type'].value_counts()
plt.figure(figsize=(8,5))
sns.countplot(x='application_type',hue='default_ind',data=data_cat)
plt.tight_layout()



#zip_code : Contains a huge amount of Unique values which are diffuicult to treat. Also, it does not conclude anything hence we will be dropping this feature.

data_cat['zip_code'].describe()
o/p - count 850051 unique 931 top 945xx freq 9402

#pymnt_plan- It doesnot conclude anything hence we can choose to ignore this for model evaluation

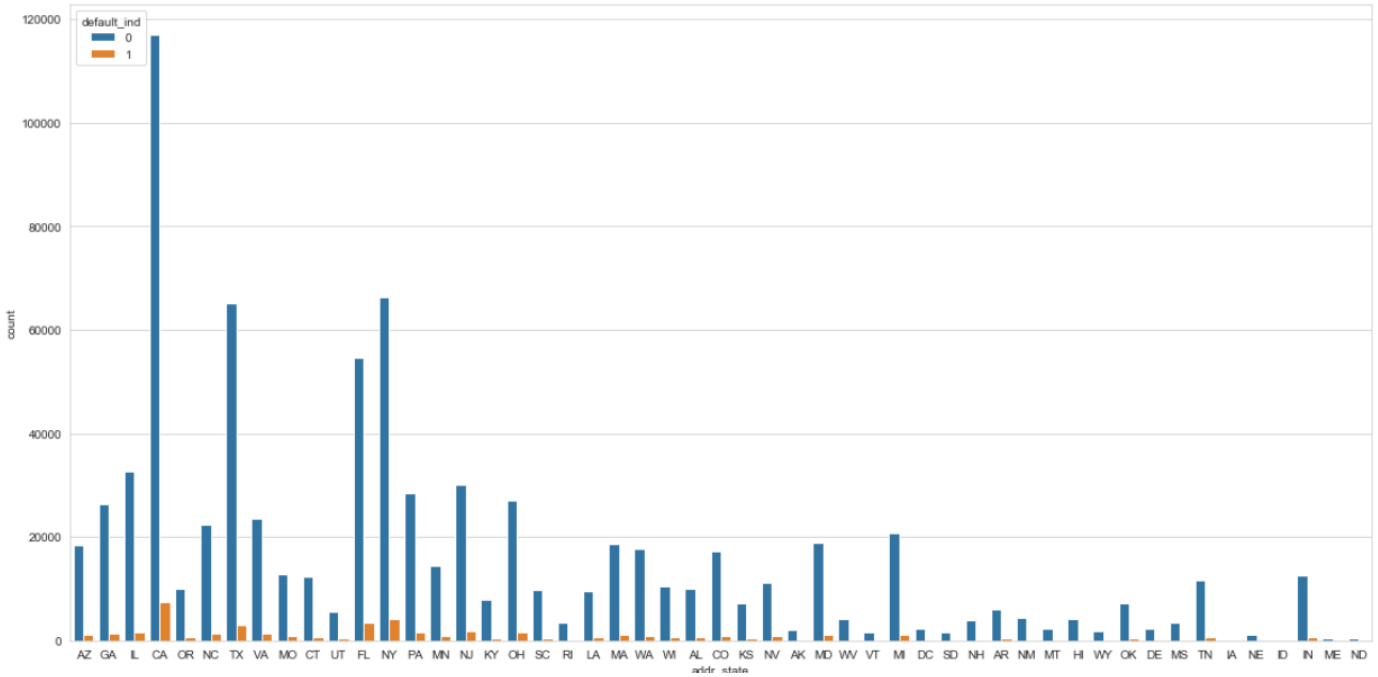
data_cat['pymnt_plan'].describe()
o/p - *count 850051 unique 2 top n freq 850046*

#title- There are as many as 60840 unique which are a lot to handle so we will choosed to drop this column for medel fitting.

data_cat['title'].describe()
o/p - count 850051 unique 60840 top Debt consolidation freq 394838

#addr_state- Since there are 51 uniques here so we will dummify/label encode it later

data_cat['addr_state'].describe()
o/p - count 850051 unique 51 top CA freq 124333
plt.figure(figsize=(16,8))
sns.countplot(x='addr_state',hue='default_ind',data=data_cat)
plt.tight_layout()

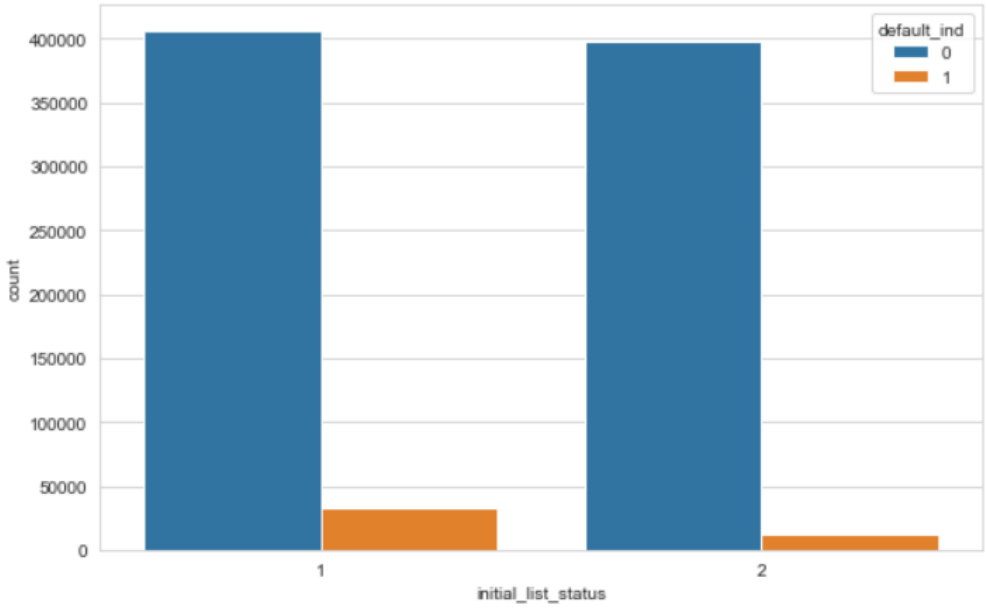


#earliest_cr_line- Has a lot of uniques so we will not consider this for model.

data_cat['earliest_cr_line'].describe()
o/p - count 850051 unique 697 top Aug-2001 freq 6383

#initial_list_status- We will map this into numerical data and plot the count plot.

data_cat['initial_list_status'].value_counts()
list_status_map={'f':1, 'w':2}
data_cat['initial_list_status']=data_cat['initial_list_status'].map(list_status_map)
data['initial_list_status']=data['initial_list_status'].map(list_status_map)
data_cat['initial_list_status'].value_counts()
plt.figure(figsize=(8,5))
sns.countplot(x='initial_list_status',hue='default_ind',data=data_cat)
plt.tight_layout()

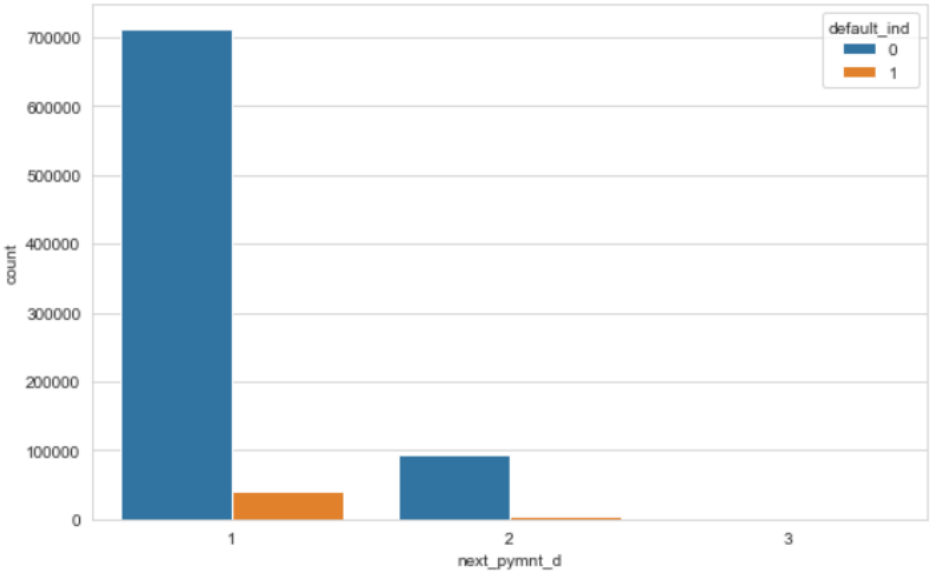


#last_pymnt_d- A lot of categories i.e. 97 uniques so we will keep this for dummy/label encoding

data_cat['last_pymnt_d'].describe()
o/p - count 850051 unique 97 top Jan-2016 freq 472746

#next_pymnt_d- Will map this to numerical and plot a countplot w.r.t. target.

```
next_pymnt_status_map={'Feb-2016':1, 'Jan-2016':2, 'Mar-2016': 3}
data_cat['next_pymnt_d']=data_cat['next_pymnt_d'].map(next_pymnt_status_map)
data['next_pymnt_d']=data['next_pymnt_d'].map(next_pymnt_status_map)
data_cat['next_pymnt_d'].value_counts()
plt.figure(figsize=(8,5))
sns.countplot(x='next_pymnt_d',hue='default_ind',data=data_cat)
plt.tight_layout()
```



#last_credit_pull_d – Too many categories so it has to be dummified / label encoded.

 # emp_title- It has too many uniques so we can choose to ignore this.

data_cat['emp_title'].describe()
o/p - count 800889 unique 289475 top Teacher freq 12890

#emp_length- It contains few null values which needs treatement and then we will go ahead with label encoding/ dummifying it.

data_cat['emp_length'].isnull().sum()
o/p - 4281

```
mapping and null value treatement
emp_range= {'< 1 year':0.5, '1 year':1, '2 years': 2, '3 years':3,
        '4 years':4, '5 years':5,'6 years':6,'7 years':7,
        '8 years':8,'9 years':9, '10+ years':10}
data_cat['emp_length'] = data_cat["emp_length"].map(emp_range)
nullseries=pd.isnull(data_cat).sum()
nullseries[nullseries>0]
data_cat['emplen'] = data_cat['emp_length'].replace(np.nan, 10)
data_cat.drop(['emp_length'],axis=1,inplace=True)
data_cat['emplen'].value_counts() #category
```

2.2.5 Feature Engineering

Since we have already processed both numerical & Categorical data based on datatypes we will now do some feaure engineering in order to make the data frame ready to be fitted.

Lets 1st reset indexes of all the data frames which we processed so far so as to avoid any dimension related issues:

```
#To reset all the indexes
data.reset_index(drop=True, inplace=True)
data_cat.reset_index(drop=True, inplace=True)
data_num.reset_index(drop=True, inplace=True)
```

Lets once check all the categorical columns :

```
data_cat.columns
```
o/p - Index(['term', 'grade', 'sub_grade', 'emp_title', 'home_ownership', 'verification_status', 'issue_d', 'pymnt_plan', 'purpose', 'title', 'zip_code', 'addr_state', 'earliest_cr_line', 'initial_list_status', 'last_pymnt_d', 'next_pymnt_d', 'last_credit_pull_d', 'application_type', 'default_ind', 'emplen'], dtype='object')

Lets check shape of all data frames –

```
print(data.shape,data_num.shape,data_cat.shape,data_ordinal.shape)
```
o/p -(850051, 54) (850051, 24) (850051, 20) (850051, 7)

Now we will plug out the columns from categorical data frame which we mapped to numerical and crea te a new data frame called - **data_numerical**

```
data_numerical = data_cat.loc[:,['grade','home_ownership','verification_status','initial_list_status','next_pymnt_d','emplen', 'application_type']]
```

data_numerical has a total of 7 columns

Now we will concatenate all numerical features i.e. data_num & data_numerical

```
data_num_all = pd.concat([data_num,data_numerical], axis = 1)
data_num_all.shape
```
o/p – 850051, 31

Numerical data has a total of 31 columns/features now.

Now we will drop target variable from numerical data frame

```
data_num_all.drop(['default_ind'], axis =1, inplace = True)
```

Now we will grab all the Numeric columns from original data frame i.e. data and create new data frame called - **scaled_all_numeric**

```
scaled_all_numeric= data.loc[:,['loan_amnt', 'int_rate', 'annual_inc', 'dti', 'delinq_2yrs',
    'inq_last_6mths', 'mths_since_last_delinq', 'open_acc', 'pub_rec', 'revol_bal', 'revol_util',
'total_acc', 'out_prncp', 'total_pymnt', 'total_rec_int', 'total_rec_late_fee', 'recoveries', 'last_py
mnt_amnt', 'collections_12_mths_ex_med', 'acc_now_delinq', 'tot_coll_amt', 'tot_cur_bal', 'inst
allment', 'grade', 'home_ownership', 'verification_status', 'initial_list_status', 'next_pymnt_d', 'e
mplen','application_type' ]]
```

Now we Scale this data frame using Standard Scalar

```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler().fit(scaled_all_numeric)
scaled_numeric_all = scaler.transform(scaled_all_numeric)
Numeric_Scaled = pd.DataFrame(scaled_numeric_all,
columns=scaled_all_numeric.columns.tolist())
```

So basically we have 2 combination of Numerical data – 1st is data_num_all which basically has ben created with divinding all columns into bands using pd.cut. 2nd Numerica as Numeric_Scaled which is scaled version of raw data.

Similarly we will treat the remaining categorical columns with 2 methods i.e. Lable Encoding and dummy encoding.

LabelEncoder_categorical *= data_cat.loc[:,['term','sub_grade','purpose','addr_state', 'last_pymnt_d', 'last_credit_pull_d']]*
dummyEncoder_categorical *= data_cat.loc[:,['term','sub_grade','purpose','addr_state', 'last_pymnt_d', 'last_credit_pull_d']]*

1st lets create label encoded categorical data :

```
from sklearn.preprocessing import LabelEncoder
categorical_feature_mask = LabelEncoder_categorical.dtypes==object
categorical_feature_mask
categorical_cols = LabelEncoder_categorical.columns[categorical_feature_mask].tolist()
le = LabelEncoder()
LabelEncoder_categorical[categorical_cols] =
LabelEncoder_categorical[categorical_cols].apply(lambda col: le.fit_transform(col))
LabelEncoder_categorical.head()
```

This is how output looks like :

	term	sub_grade	purpose	addr_state	last_pymnt_d	last_credit_pull_d
0	0	6	1	3	39	41
1	1	13	0	10	5	99
2	0	14	11	14	55	41
3	0	10	9	4	39	40
4	1	9	9	37	40	41

2nd Lets create Dummy encoded values for categorical data :

```
ummyEncoder_categorical = pd.get_dummies(dummyEncoder_categorical, drop_first=True)
dummyEncoder_categorical.shape
```

o/p: 850051,295
So it has 295 columns now

So going ahead with model fitting we will consider below combinations :

Scaled Numeric	Dummy Categorical
Scaled Numeric	Label categorical
Cut / band Numerical	Dummy Categorical
Cut / band Numerical	Label categorical

CHAPTER 3: MODEL BUILDING & EVALUATION

3.1 Visualizing Recall, Precision, F1 Score & Accuracy

3.1.1 Confusion matrix: shows the actual and predicted labels from a classification problem

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

- True positives: data points labeled as positive that are actually positive
- False positives: data points labeled as positive that are actually negative
- True negatives: data points labeled as negative that are actually negative
- False negatives: data points labeled as negative that are actually positive

Calculating Recall, Precision, F1 Score and Accuracy

- **Recall**: ability of a classification model to identify all relevant instances
- **Precision**: ability of a classification model to return only relevant instances
- **F1 score**: single metric that combines recall and precision using the harmonic mean
- **Accuracy** - Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations.

$$recall = \frac{true\ positives}{true\ positives\ + false\ negatives}$$

$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

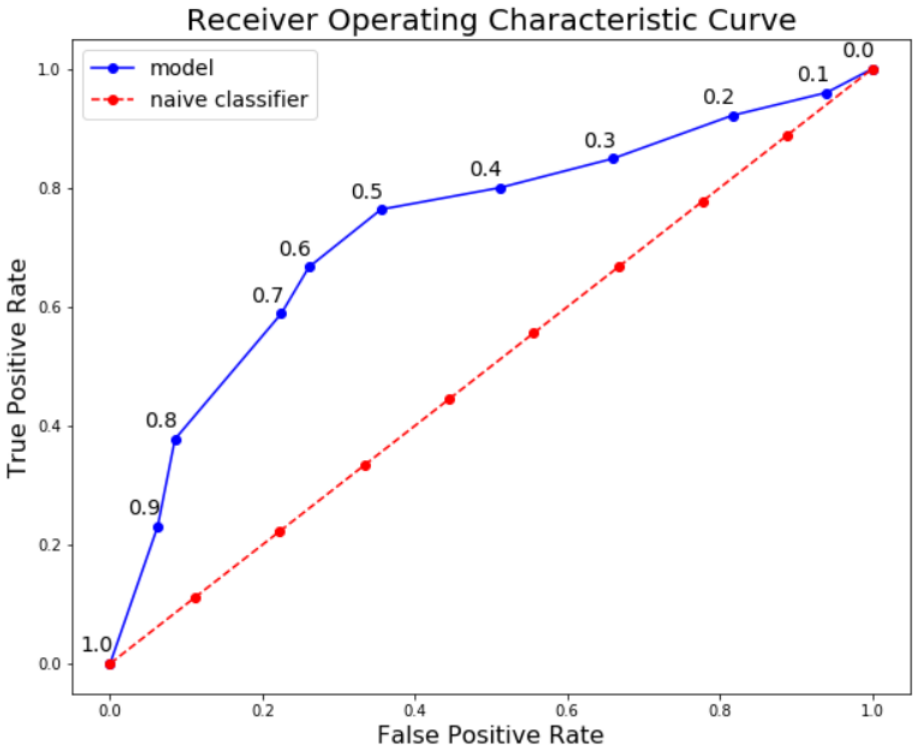
$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

$$Accuracy = \frac{TrueNegatives + TruePositive}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$

3.1.2 Receiver operating characteristic (ROC) & Area under the curve (AUC)

- **Receiver operating characteristic (ROC)**: plots the true positive rate (TPR) versus the false positive rate (FPR) as a function of the model's threshold for classifying a positive

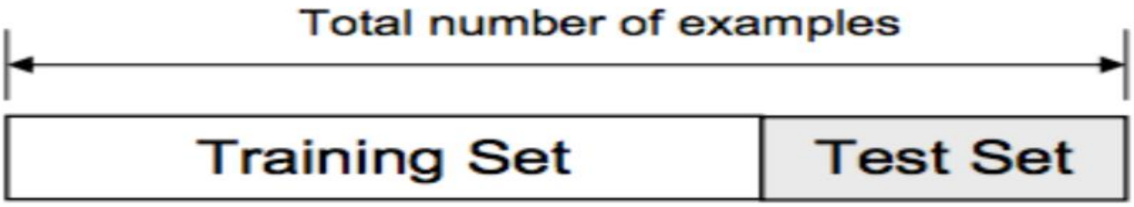
- **Area under the curve (AUC)**: metric to calculate the overall performance of a classification model based on area under the ROC curve



3.2 Data Preparation for model fitting

3.2.1 Train/Test Split

The data we use is usually split into training data and test data. The training set contains a known output and the model learns on this data in order to be generalized to other data later on. We have the test dataset (or subset) in order to test our model's prediction on this subset. Further if we have a large dataset, we can divide train dataset into Sub Train & validation sets and 1st check accuracy on validation and then on test data.




```
com1[['issue_d','default_ind']]= data_cat[['issue_d','default_ind']]
#for splitting purpose'
com1['issue_d'] = pd.to_datetime(com1['issue_d'])
# Creating train and test data set According to problem statement given.
The train data should be from June 2007 - May 2015 & test should be
and out-of-time test ( June 2015 - Dec 2015  from issue_d columns

train1 = com1[com1['issue_d'] < '2015-6-01']
```

```
test1 = com1[com1['issue_d'] >= '2015-6-01']


train1 =train1.drop('issue_d' , axis=1)
test1 =test1.drop('issue_d', axis=1)
"""X = train1.iloc[:,0:-1]
y = train1['default_ind']"""

X_train1 = train1.iloc[:,0:-1]
y_train1 = train1['default_ind']
X_test_com1  = test1.iloc[:,0:-1]
y_test_com1  = test1['default_ind']
print(X_train1.shape, y_train1.shape, X_test_com1.shape, y_test_com1.shape)
```

o/p - **(594741, 325) (594741,) (255310, 325) (255310,)**

Since it's a huge dataset and training the data could actually cause a lot of memory issues so we will divide train data into Sub Train and Validation.

```
from sklearn.model_selection import train_test_split
X_train_com1, X_val1, y_train_com1, y_val1 = train_test_split(X_train1, y_train1, test_size =
0.3, random_state = 0)
print(X_train_com1.shape, y_train_com1.shape, X_val1.shape, y_val1.shape)
```

***o/p* - (416318, 325) (416318,) (178423, 325) (178423,)**

3.2.2 Overfitting/Underfitting a Model for Imbalance Treatment

Overfitting means that model we trained has trained "too well" and is now, well, fit too closely to the training dataset. This usually happens when the model is too complex.

Underfitting means that the model does not fit the training data and therefore misses the trends in the data. It also means the model cannot be generalized to new data.

#Methods

Random Under-Sampling - Random Under sampling aims to balance class distribution by randomly eliminating majority class examples. This is done until the majority and minority class instances are balanced out.

Random Over-Sampling - Over-Sampling increases the number of instances in the minority class by randomly replicating them in order to present a higher representation of the minority class in the sample.

Cluster-Based Over Sampling - K-means clustering algorithm is independently applied to minority and majority class instances. This is to identify clusters in the dataset. Subsequently, each cluster is oversampled such that all clusters of the same class have an equal number of instances and all classes have the same size.

Synthetic Minority Over-sampling Technique- followed when exact replicas of minority instances are added to the main dataset. A subset of data is taken from the minority class as an example and then new synthetic similar instances are created. These synthetic instances are then added to the original dataset. The new dataset is used as a sample to train the classification models.

Bagging- Bagging is used for reducing Overfitting in order to create strong learners for generating accurate predictions. Unlike boosting, bagging allows replacement in the bootstrapped sample.

Boosting-Based Techniques-Itis an ensemble technique to combine weak learners to create a strong learner that can make accurate predictions. Boosting starts out with a base classifier / weak classifier that is prepared on the training data., the new classifier places more weight to those cases which were incorrectly classified in the last round.

XG Boost techniques- It is 10 times faster than the normal Gradient Boosting as it implements parallel processing. It is highly flexible as users can define custom optimization objectives and evaluation criteria, has an inbuilt mechanism to handle missing values.

Performing SMOTE on Train:

```
from imblearn.over_sampling import SMOTE
sm1 = SMOTE(random_state = 2)
X_sm1, y_sm1 = sm1.fit_sample(X_train_com1, y_train_com1.ravel())
print('Before OverSampling, X: {}'.format(X_train_com1.shape))
print('Before OverSampling, y: {}'.format(y_train_com1.shape))
print("Before OverSampling, counts of '1': {}".format(sum(y_train_com1 == 1)))
print("Before OverSampling, counts of '0': {}".format(sum(y_train_com1 == 0)))
print('\n')
print('With imbalance treatment:'.upper())
print('After OverSampling, X: {}'.format(X_sm1.shape))
print('After OverSampling, y: {}'.format(y_sm1.shape))
print("After OverSampling, counts of '1': {}".format(sum(y_sm1 == 1)))
print("After OverSampling, counts of '0': {}".format(sum(y_sm1 == 0)))
print('\n')
```

```
Before OverSampling, X: (416318, 325)
Before OverSampling, y: (416318,)
Before OverSampling, counts of '1': 31616
Before OverSampling, counts of '0': 384702


WITH IMBALANCE TREATMENT:
After OverSampling, X: (769404, 325)
After OverSampling, y: (769404,)
After OverSampling, counts of '1': 384702
After OverSampling, counts of '0': 384702
```

Performing SMOTE on Train & Test both:

```
from imblearn.over_sampling import SMOTE
smt = SMOTE(random_state = 2)
X_smt1, y_smt1 = smt.fit_sample(X_test_com1, y_test_com1.ravel())
print('Before OverSampling, X: {}'.format(X_test_com1.shape))
print('Before OverSampling, y: {}'.format(y_test_com1.shape))
print("Before OverSampling, counts of '1': {}".format(sum(y_test_com1 == 1)))
print("Before OverSampling, counts of '0': {}".format(sum(y_test_com1 == 0)))
print('\n')
print('With imbalance treatment:'.upper())
print('After OverSampling, X: {}'.format(X_smt1.shape))
print('After OverSampling, y: {}'.format(y_smt1.shape))
print("After OverSampling, counts of '1': {}".format(sum(y_smt1 == 1)))
print("After OverSampling, counts of '0': {}".format(sum(y_smt1 == 0)))
print('\n')
```

```
Before OverSampling, X: (255310, 325)
Before OverSampling, y: (255310,)
Before OverSampling, counts of '1': 297
Before OverSampling, counts of '0': 255013


WITH IMBALANCE TREATMENT:
After OverSampling, X: (510026, 325)
After OverSampling, y: (510026,)
After OverSampling, counts of '1': 255013
After OverSampling, counts of '0': 255013
```

Performing Under Sampling on Train:

```
from imblearn.under_sampling import (RandomUnderSampler)
un = RandomUnderSampler(random_state=2)
X_sm3, y_sm3 = un.fit_sample(X_train_com1, y_train_com1.ravel())
print('Before Undersampling, X: {}'.format(X_train_com1.shape))
print('Before Undersampling, y: {}'.format(y_train_com1.shape))
print("Before Undersampling, counts of '1': {}".format(sum(y_train_com1 == 1)))
print("Before Undersampling, counts of '0': {}".format(sum(y_train_com1 == 0)))
print('\n')
print('With imbalance treatment:'.upper())
print('After Undersampling, X: {}'.format(X_sm3.shape))
print('After Undersampling, y: {}'.format(y_sm3.shape))
print("After Undersampling, counts of '1': {}".format(sum(y_sm3 == 1)))
print("After Undersampling, counts of '0': {}".format(sum(y_sm3 == 0)))
print('\n')
```

```
Before Undersampling, X: (416318, 325)
Before Undersampling, y: (416318,)
Before Undersampling, counts of '1': 31616
Before Undersampling, counts of '0': 384702


WITH IMBALANCE TREATMENT:
After Undersampling, X: (63232, 325)
After Undersampling, y: (63232,)
After Undersampling, counts of '1': 31616
After Undersampling, counts of '0': 31616
```

It is quite evident that numbers of ones are way lesser that the number of zeros in both train & test datasets. Hence an imbalance treatment is a must here.

3.3 Modelling Techniques used

For this assignment we will be focussing on Logistic regression, Decision Tree, Random Forest, XG Boost and Support vector Machine.

3.3.1 Logistic Regression

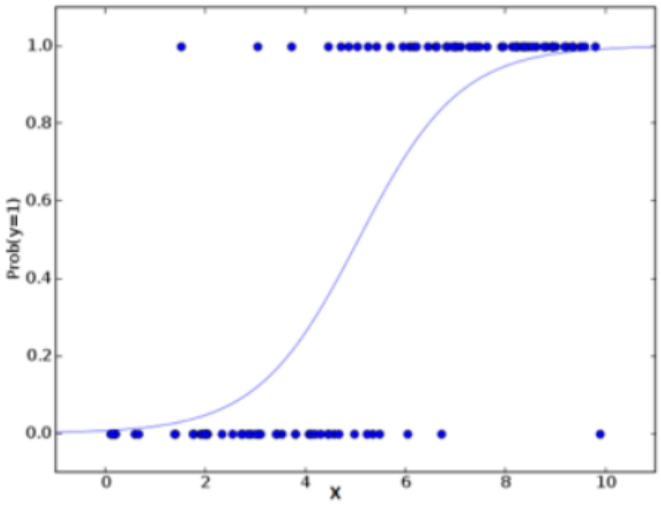
3.3.1.1 Working Principle

Logistic Regression is a classification algorithm. It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables. To represent binary/categorical outcome, we use dummy variables.

Equation for Logistic Regression:

$$\log\left(\frac{p}{1-p}\right) = \beta_o + \beta(\text{Age})$$

Curve for Logistic regression:



3.3.1.2 Model Fitting

Importing Libraries to build model

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import confusion_matrix, classification_report , f1_score, accuracy_score,
roc_auc_score
from confusionMatrix import plotConfusionMatrix
```

 # To treat data imbalance, call SMOTE & UnderSampler

```
sm1 = SMOTE(random_state = 2)
X_sm1, y_sm1 = sm1.fit_sample(X_train_com1, y_train_com1.ravel())

un = RandomUnderSampler(random_state=2)
X_sm3, y_sm3 = un.fit_sample(X_train_com1, y_train_com1.ravel())
```

 # Model fitting without imbalance treatment-

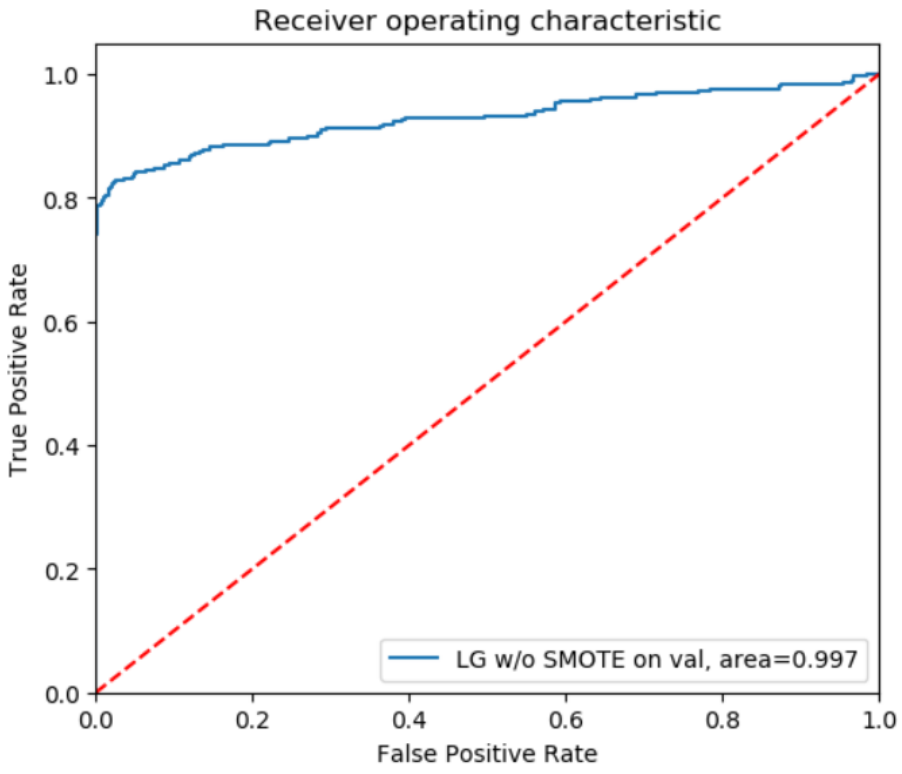
```
lR = LogisticRegression(max_iter=200, C=0.5)
lR.fit(X_train_com1, y_train_com1)
acc_v=lR.score(X_val1, y_val1)
acc = lR.score(X_test_com1, y_test_com1)
predv= lR.predict(X_val1)
preds = lR.predict(X_test_com1)
pred_probav = lR.predict_proba(X_val1)[::,1]
pred_proba = lR.predict_proba(X_test_com1)[::,1]
print('*'*80)
print('Logistic Regression:')
```

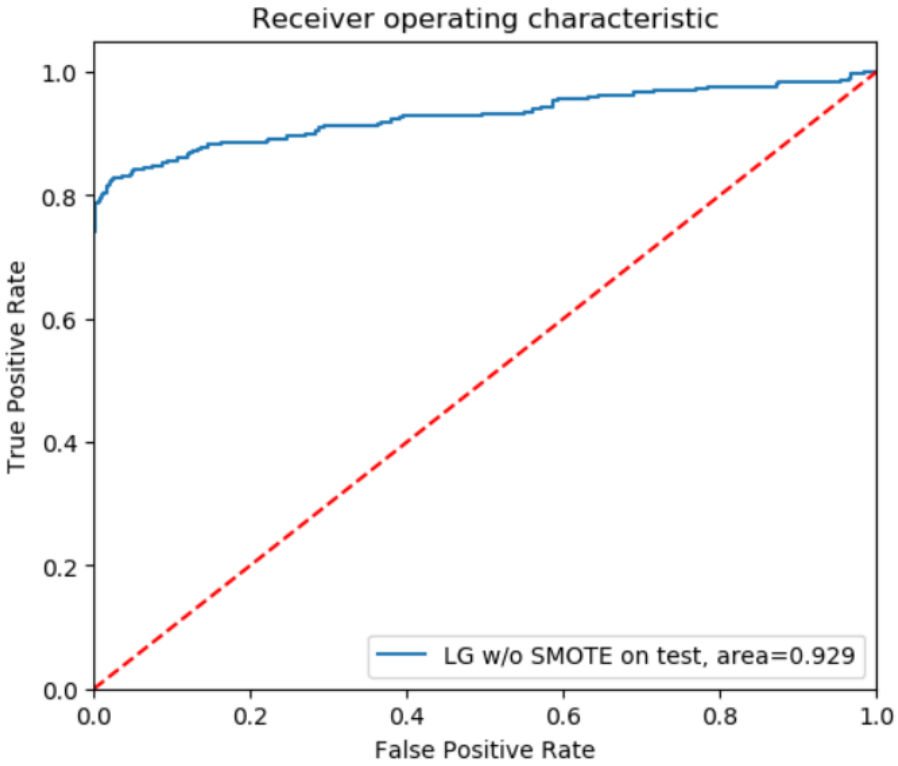
```
print("Accuracy without SNOTE on validation set: %.2f%%" % (acc_v * 100.0))
print("Accuracy without SNOTE on test set: %.2f%%" % (acc * 100.0))
print('F1 score val:\n', classification_report(y_val1, predv))
print('F1 score test:\n', classification_report(y_test_com1, preds))
fpr_G, tpr_G, _G = roc_curve(y_test_com1,  pred_proba)
aucv = roc_auc_score(y_val1, pred_probav)
plt.plot(fpr_G,tpr_G,label="LG w/o SMOTE on val, area="+str(np.round(aucv,3)))
```

- LET's see the o/p we got –

1. Let's check Accuracy for validation & Test along with Precision, Recall & F1 Score
2. ROC/AUC curve.

```
*********************************************************
Logistic Regression:
Accuracy without SNOTE on validation set: 99.00%
Accuracy without SNOTE on test set: 99.96%
*********************************************************
F1 score val:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99    164668
           1       0.94      0.92      0.93     13755

    accuracy                           0.99    178423
   macro avg       0.97      0.96      0.96    178423
weighted avg       0.99      0.99      0.99    178423

*********************************************************
F1 score test:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    255013
           1       0.92      0.70      0.79       297

    accuracy                           1.00    255310
   macro avg       0.96      0.85      0.90    255310
weighted avg       1.00      1.00      1.00    255310

*********************************************************
```



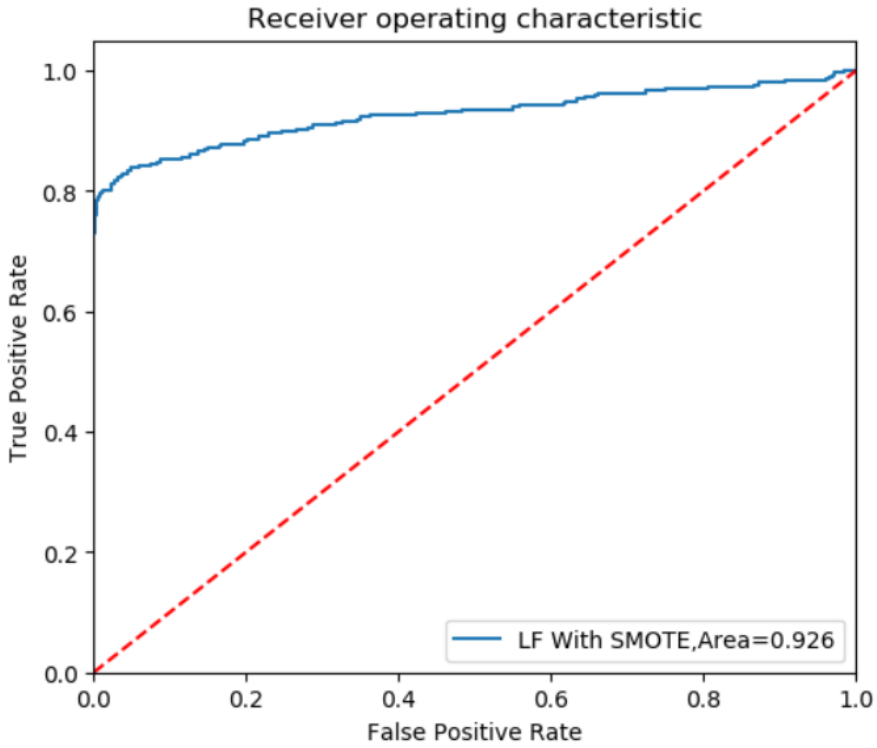


 # Model fitting with imbalance treatment using SMOTE on Train –

```
lR = LogisticRegression(max_iter=200, C=0.5)
lR.fit(X_sm1, y_sm1)
lR.score(X_sm1, y_sm1)
acc = lR.score(X_test_com1, y_test_com1)
preds = lR.predict(X_test_com1)
pred_proba = lR.predict_proba(X_test_com1)[::,1]
print('*'*80)
print('Logistic Regression:')
print("Accuracy with SMOTE: %.2f%%" % (acc * 100.0))
print(classification_report(y_test_com1, preds))
print('*'*80)
#y_pred_proba_G = model_G.predict_proba(X_val_res)[::,1]
fpr_G, tpr_G, _G = roc_curve(y_test_com1,  pred_proba)
auc = roc_auc_score(y_test_com1, pred_proba)
print(confusion_matrix(y_test_com1, preds))
```

```
************************************************************
Logistic Regression:
Accuracy with SMOTE: 99.95%
************************************************************
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    255013
           1       0.87      0.69      0.77       297

    accuracy                           1.00    255310
   macro avg       0.94      0.84      0.88    255310
weighted avg       1.00      1.00      1.00    255310

************************************************************
```

```
************************
[[254983     30]
 [    93    204]]
************************
```



 # Model fitting with imbalance treatment using SMOTE on Train & Test –

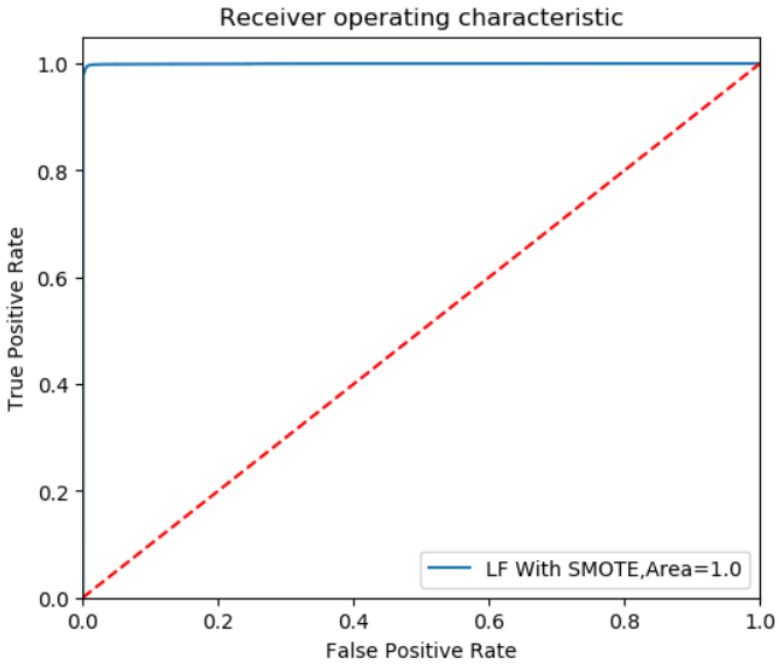
```
lR = LogisticRegression(max_iter=200, C=0.5)
lR.fit(X_sm1, y_sm1)
lR.score(X_sm1, y_sm1)
acct = lR.score(X_smt1, y_smt1)
preds = lR.predict(X_smt1)
pred_proba = lR.predict_proba(X_smt1)[::,1]
print('Logistic Regression:')
print("Accuracy with SMOTE: %.2f%%" % (acct * 100.0))
print(classification_report(y_smt1, preds))
#y_pred_proba_G = model_G.predict_proba(X_val_res)[::,1]
fpr_G, tpr_G, _G = roc_curve(y_smt1,  pred_proba)
auct = roc_auc_score(y_smt1, pred_proba)
plt.figure(figsize = (6,5))
plt.plot(fpr_G,tpr_G,label="LF With SMOTE,Area="+str(np.round(auct,3)))
```

- LET's see the o/p we got –

1. Let's check Accuracy for validation & Test along with Precision, Recall & F1 Score
2. ROC/AUC curve.

```
**********************
[[254983     30]
 [ 10768 244245]]
**********************
```

```
*******************************************************
Logistic Regression:
Accuracy with SMOTE: 97.88%
*******************************************************
              precision    recall  f1-score   support

           0       0.96      1.00      0.98    255013
           1       1.00      0.96      0.98    255013

    accuracy                           0.98    510026
   macro avg       0.98      0.98      0.98    510026
weighted avg       0.98      0.98      0.98    510026

*******************************************************
```



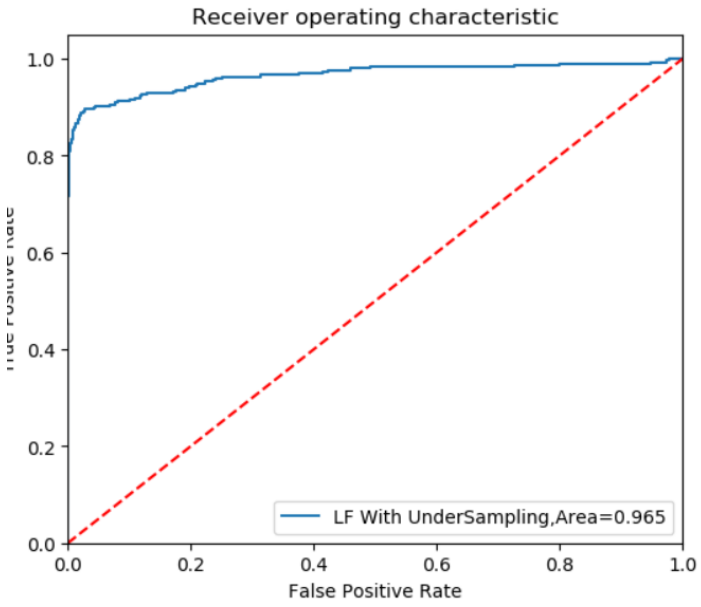
 # Model fitting with imbalance treatment using Under Sampling on Train –

```
lR = LogisticRegression(max_iter=200, C=0.5)
lR.fit(X_sm3, y_sm3)
lR.score(X_sm3, y_sm3)
acc = lR.score(X_test_com1, y_test_com1)
preds = lR.predict(X_test_com1)
pred_proba = lR.predict_proba(X_test_com1)[::,1]
print('Logistic Regression:')
print("Accuracy with UnderSampling: %.2f%%" % (acc * 100.0))
print(classification_report(y_test_com1, preds))
#y_pred_proba_G = model_G.predict_proba(X_val_res)[::,1]
fpr_G, tpr_G, _G = roc_curve(y_test_com1,  pred_proba)
auc = roc_auc_score(y_test_com1, pred_proba)
plt.figure(figsize = (6,5))
plt.plot(fpr_G,tpr_G,label="LF With UnderSampling,Area="+str(np.round(auc,3)))
```

- LET's see the o/p we got –

1. Let's check Accuracy for validation & Test along with Precision, Recall & F1 Score
2. ROC/AUC curve.

```
**********************************************************
Logistic Regression:
Accuracy with UnderSampling: 99.78%
**********************************************************
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    255013
           1       0.32      0.79      0.45       297

    accuracy                           1.00    255310
   macro avg       0.66      0.89      0.73    255310
weighted avg       1.00      1.00      1.00    255310

**********************************************************

**********
[[254507    506]
 [    62    235]]
**********
```



3.3.1.3 Logistic Regression Statistics

	Method	Testing on	ACCURACY(%)	PRECISION	RECALL	F1-SCORE	ROC Score
LR	Without Treatement	Validation	99	0.94	0.92	0.93	0.997
		Test	99.96	0.92	0.7	0.79	0.929
	SMOTE on Train	Test	99.95	0.87	0.69	0.77	0.926
	SMOTE on Train & Test	Test	97.88	1	0.96	0.98	1
	Under Sampling	Test	99.78	0.32	0.79	0.45	0.965

3.3.2 Decision Tree Model

3.3.2.1: Working Principle

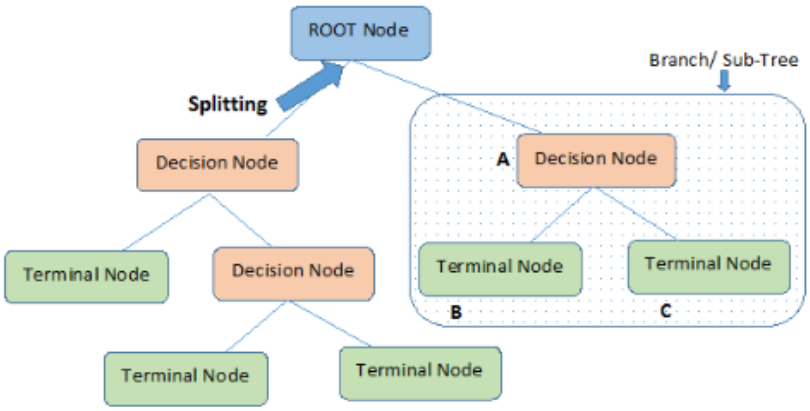
It is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator in input variables.

Types

1. Binary Variable Decision Tree: Decision Tree which has binary target variable then it called as Binary Variable Decision Tree
2. Continuous Variable Decision Tree: Decision Tree has continuous target variable then it is called as Continuous Variable Decision Tree.

Terminologies

ROOT Node: It represents entire population or sample, and this further gets divided into two or more homogeneous sets.
SPLITTING: It is a process of dividing a node into two or more sub-nodes.
Decision Node: When a sub-node splits into further sub-nodes, then it is called decision node.
Leaf/ Terminal Node: Nodes do not split is called Leaf or Terminal node.
Pruning: When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.
Branch / Sub-Tree: A sub section of entire tree is called branch or sub-tree
Parent and Child Node: A node, which is divided into sub-nodes is called parent node of sub-nodes whereas sub-nodes are the child of parent node.



3.3.2.1: Model Fitting

 # Importing Libraries to build model

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, classification_report , f1_score, accuracy_score, roc_auc_score, roc_curve
from confusionMatrix import plotConfusionMatrix
```

 # To treat data imbalance, call SMOTE & UnderSampler

```
sm1 = SMOTE(random_state = 2)
X_sm1, y_sm1 = sm1.fit_sample(X_train_com1, y_train_com1.ravel())
un = RandomUnderSampler(random_state=2)
X_sm3, y_sm3 = un.fit_sample(X_train_com1, y_train_com1.ravel())
```

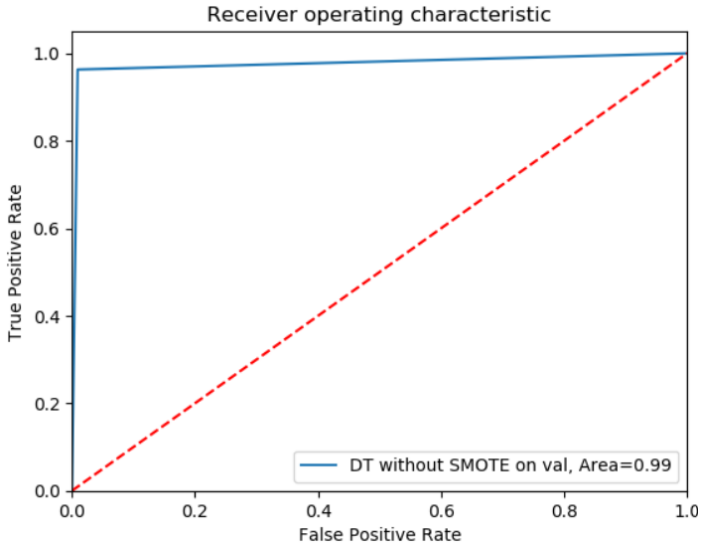
 # Model fitting without imbalance treatment-

```
clf = DecisionTreeClassifier(random_state=0)
clf.fit(X_train_com1, y_train_com1.ravel())
accv = clf.score(X_val1, y_val1)
acc = clf.score(X_test_com1, y_test_com1)
predictions_v = clf.predict(X_val1)
predictions_ = clf.predict(X_test_com1)
y_pred_probav = clf.predict_proba(X_val1)[::,1]
y_pred_proba = clf.predict_proba(X_test_com1)[::,1]
print('Decision Tree:')
print("Accuracy without SMOTE on val: %.2f%%" % (accv * 100.0))
print("Accuracy without SMOTE on test: %.2f%%" % (acc * 100.0))
print(confusion_matrix(y_val1, predictions_v))
```

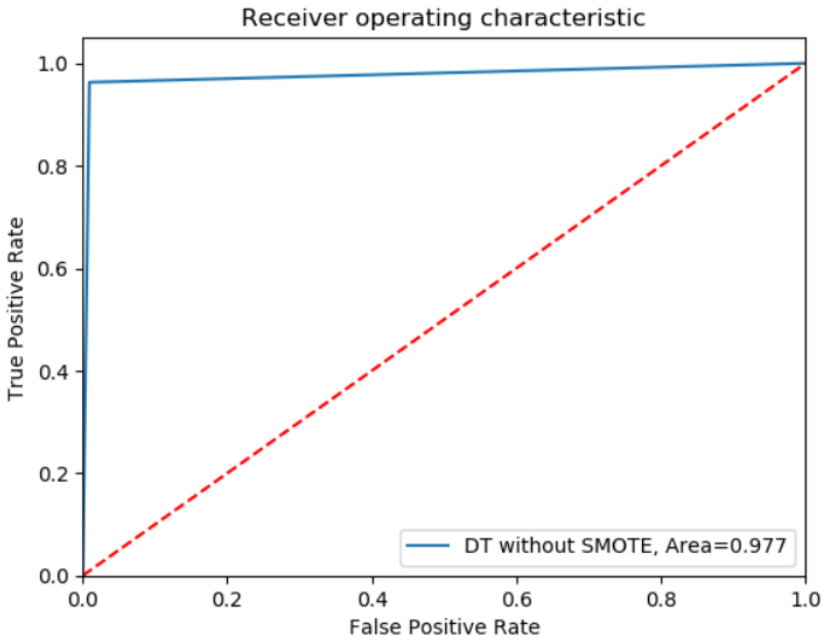
```
print('Without SMOTE on Val:'.upper())
print(classification_report(y_val1, predictions_v))
print(confusion_matrix(y_test_com1, predictions_))
print('Without SMOTE on Test:'.upper())
print(classification_report(y_test_com1, predictions_))
y_pred_probav = clf.predict_proba(X_val1)[::,1]
fpr, tpr, _ = roc_curve(y_test_com1,  y_pred_probav)
aucv = roc_auc_score(y_val1, y_pred_probav)
plt.plot(fpr,tpr,label="DT without SMOTE on val, Area="+str(np.round(aucv,3)))
plt.show()
```

- LET's see the o/p we got –

1. Let's check Accuracy for validation & Test along with Precision, Recall & F1 Score
2. ROC/AUC curve.




```
**********************************************************
Decision Tree:
Accuracy without SMOTE on val: 99.73%
Accuracy without SMOTE on test: 99.08%
**********************************************************
[[164436    232]
 [   257  13498]]
WITHOUT SMOTE ON VAL:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    164668
           1       0.98      0.98      0.98     13755

    accuracy                           1.00    178423
   macro avg       0.99      0.99      0.99    178423
weighted avg       1.00      1.00      1.00    178423

**********************************************************
[[252673   2340]
 [    11    286]]
WITHOUT SMOTE ON TEST:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00    255013
           1       0.11      0.96      0.20       297

    accuracy                           0.99    255310
   macro avg       0.55      0.98      0.60    255310
weighted avg       1.00      0.99      0.99    255310

**********************************************************
```



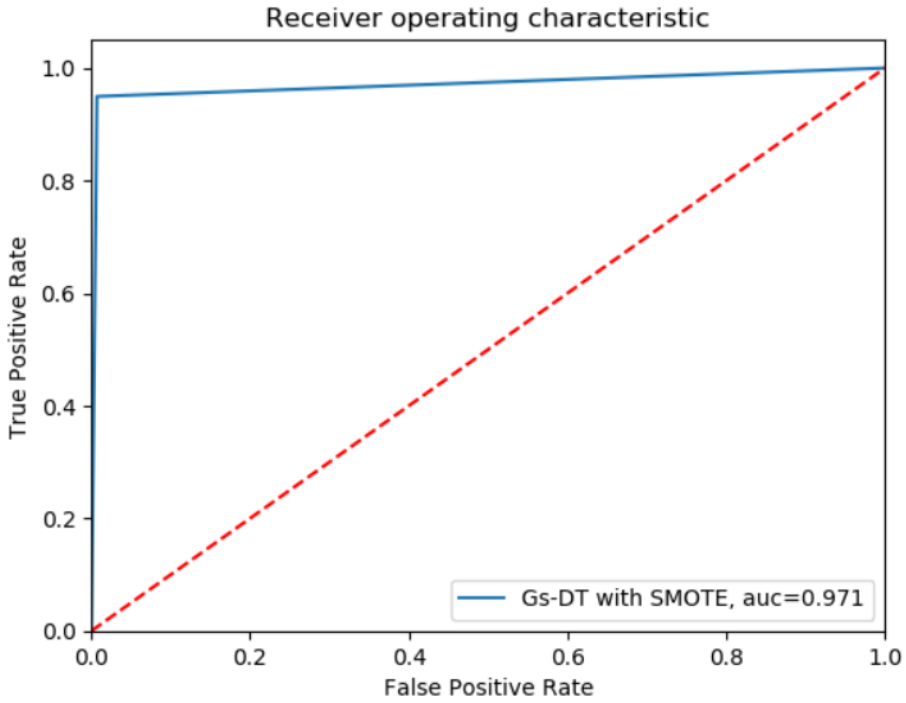
 # Model fitting with imbalance treatment using SMOTE on Train –

```
clfs = DecisionTreeClassifier(random_state=0)
clfs.fit(X_sm1, y_sm1.ravel())
acc = clfs.score(X_test_com1, y_test_com1)
predictions_ = clfs.predict(X_test_com1)
y_pred_proba = clfs.predict_proba(X_test_com1)[::,1]
print('Decision Tree:')
print("Accuracy with SMOTE: %.2f%%" % (acc * 100.0))
print(confusion_matrix(y_test_com1, predictions_))
# print classification report
print('With SMOTE:'.upper())
print(classification_report(y_test_com1, predictions_))
y_pred_proba = clfs.predict_proba(X_test_com1)[::,1]
fpr, tpr, _ = roc_curve(y_test_com1,  y_pred_proba)
auc = roc_auc_score(y_test_com1, y_pred_proba)
plt.plot(fpr,tpr,label="Gs-DT with SMOTE, auc="+str(np.round(auc,3)))
```

- LET's see the o/p we got –

1. Let's check Accuracy for validation & Test along with Precision, Recall & F1 Score
2. ROC/AUC curve.

```
**********************************************************
Decision Tree:
Accuracy with SMOTE: 99.30%
**********************************************************
[[253242   1771]
 [    15    282]]
**********************************************************
WITH SMOTE:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00    255013
           1       0.14      0.95      0.24       297

    accuracy                           0.99    255310
   macro avg       0.57      0.97      0.62    255310
weighted avg       1.00      0.99      1.00    255310

**********************************************************
```

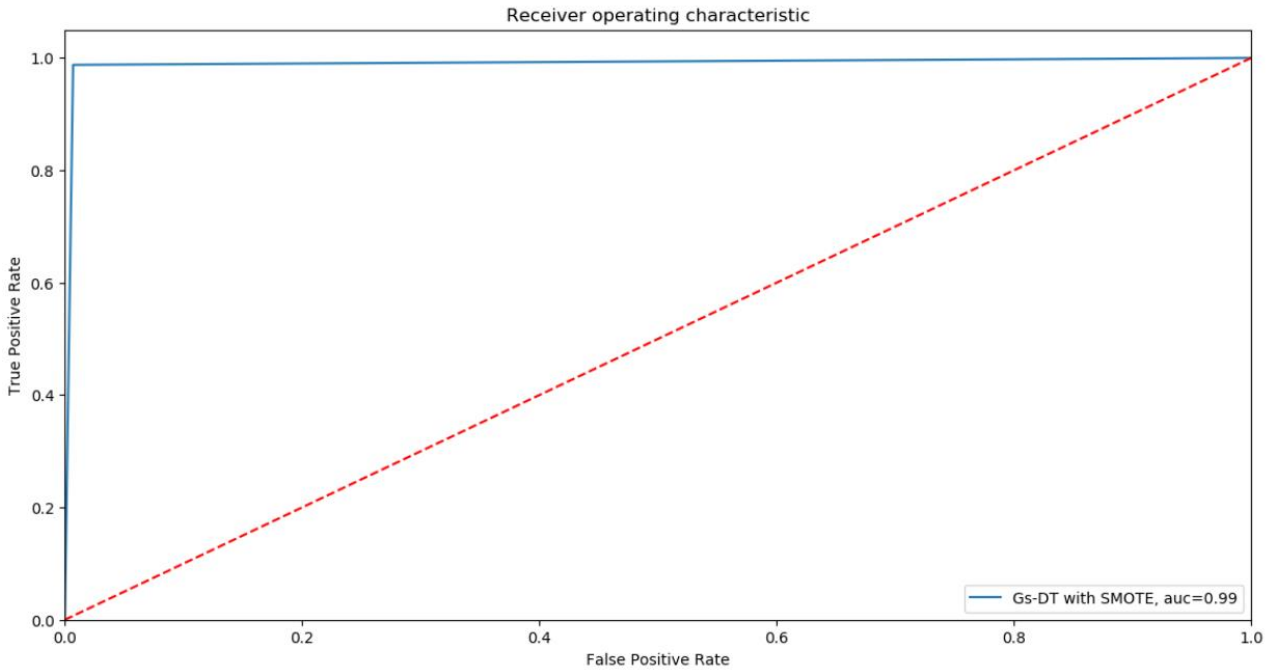


 # Model fitting with imbalance treatment using SMOTE on Train & Test –

```
clfs = DecisionTreeClassifier(random_state=0)
clfs.fit(X_sm1, y_sm1.ravel())
acc = clfs.score(X_smt1, y_smt1)
predictions_ = clfs.predict(X_smt1)
y_pred_proba = clfs.predict_proba(X_smt1)[::,1]
print('Decision Tree:')
print("Accuracy with SMOTE: %.2f%%" % (acc * 100.0))
print(confusion_matrix(y_smt1, predictions_))
print('With SMOTE:'.upper())
print(classification_report(y_smt1, predictions_))
y_pred_proba = clfs.predict_proba(X_smt1)[::,1]
fpr, tpr, _ = roc_curve(y_smt1,  y_pred_proba)
auc = roc_auc_score(y_smt1, y_pred_proba)
plt.plot(fpr,tpr,label="Gs-DT with SMOTE, auc="+str(np.round(auc,3)))
```

- LET's see the o/p we got –
  1. Let's check Accuracy for validation & Test along with Precision, Recall & F1 Score
  2. ROC/AUC curve.

```
**************************************************
Decision Tree:
Accuracy with SMOTE: 99.04%
**************************************************
[[253242   1771]
 [  3141 251872]]
**************************************************
WITH SMOTE:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99    255013
           1       0.99      0.99      0.99    255013

    accuracy                           0.99    510026
   macro avg       0.99      0.99      0.99    510026
weighted avg       0.99      0.99      0.99    510026

**************************************************
```

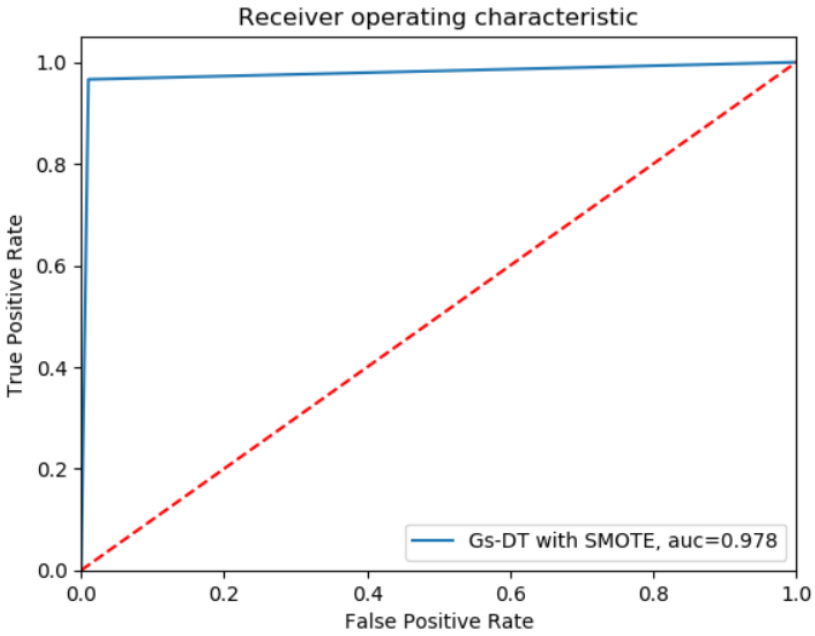


 # Model fitting with imbalance treatment using Undersampling on Train –

```
clfs = DecisionTreeClassifier(random_state=0)
clfs.fit(X_sm3, y_sm3.ravel())
acc = clfs.score(X_test_com1, y_test_com1)
predictions_ = clfs.predict(X_test_com1)
y_pred_proba = clfs.predict_proba(X_test_com1)[::,1]
print('Decision Tree:')
print("Accuracy with UnderSampling: %.2f%%" % (acc * 100.0))
print(confusion_matrix(y_test_com1, predictions_))
print('With SMOTE:'.upper())
print(classification_report(y_test_com1, predictions_))
y_pred_proba = clfs.predict_proba(X_test_com1)[::,1]
fpr, tpr, _ = roc_curve(y_test_com1,  y_pred_proba)
auc = roc_auc_score(y_test_com1, y_pred_proba)
plt.plot(fpr,tpr,label="Gs-DT with SMOTE, auc="+str(np.round(auc,3)))
```

- LET's see the o/p we got –
  1. Let's check Accuracy for validation & Test along with Precision, Recall & F1 Score
  2. ROC/AUC curve.

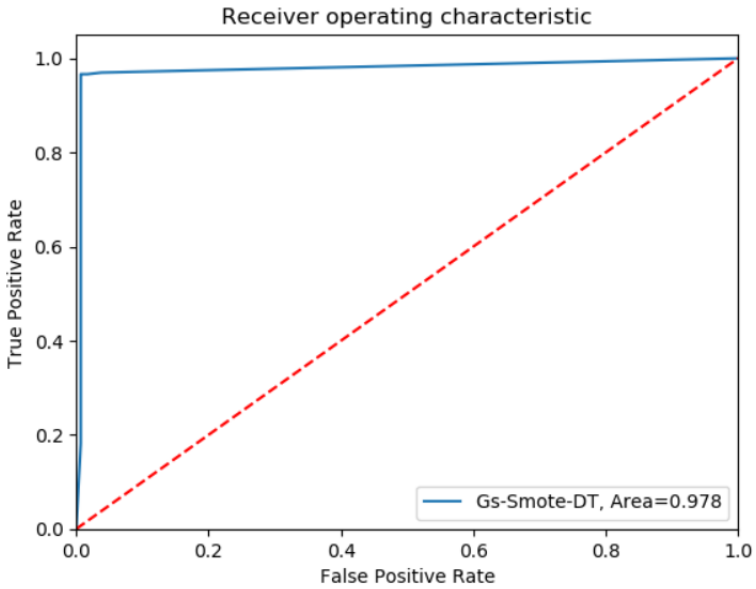
```
**********************************************************
Decision Tree:
Accuracy with UnderSampling: 98.99%
**********************************************************
[[252434   2579]
 [    10    287]]
**********************************************************
WITH SMOTE:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99    255013
           1       0.10      0.97      0.18       297

    accuracy                           0.99    255310
   macro avg       0.55      0.98      0.59    255310
weighted avg       1.00      0.99      0.99    255310

**********************************************************
```



3.3.2.3 Model Fitting using Cross Validation

```
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
from sklearn import metrics
parameters = {'max_depth': np.arange(3, 10)} # pruning
tree = GridSearchCV(clfs,parameters)
tree.fit(X_sm1,y_sm1)
preds = tree.predict(X_test_com1)
accu = tree.score(X_test_com1, y_test_com1)
print('GRID SEARCH WITH SMOTE -- DT:')
print('Using best parameters:',tree.best_params_)
print("Accuracy with SMOTE & Grid Search: %.2f%%" % (acc * 100.0))
y_pred_proba_ = tree.predict_proba(X_test_com1)[::,1]
fpr, tpr, _ = roc_curve(y_test_com1,  y_pred_proba_)
auc = roc_auc_score(y_test_com1, y_pred_proba_)
plt.plot(fpr,tpr,label="Gs-Smote-DT, Area="+str(np.round(auc,3)))
```

```
GRID SEARCH WITH SMOTE -- DT:
Using best parameters: {'max_depth': 9}
Accuracy with SMOTE & Grid Search: 99.30%
```



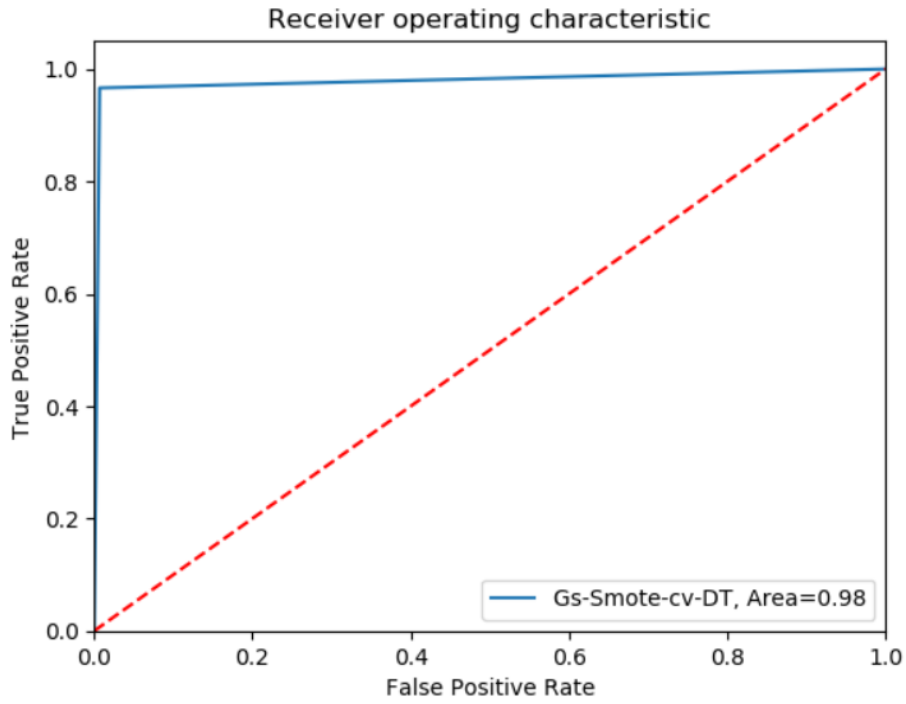
3.3.2.4 Model Fitting using Cross Validation, SMOTE & Grid Search

```
def dtree_grid_search(X,y,nfolds):
#create a dictionary of all values we want to test
param_grid = { 'criterion':['gini','entropy'],'max_depth': np.arange(15, 30)}
# decision tree model
dtree_model = DecisionTreeClassifier()
#use gridsearch to val all values
dtree_gscv = GridSearchCV(dtree_model, param_grid, cv=nfolds)
#fit model to data
dtree_gscv.fit(X, y)
#find score
score = dtree_gscv.score(X, y)
return dtree_gscv.best_params_, score, dtree_gscv

print('GRID SEARCH WITH SMOTE & CROSS VALIDATION -- DT:')
best_param, acc, model = dtree_grid_search(X_sm1,y_sm1, 4)
preds = model.predict(X_test_com1)
acc = model.score(X_test_com1, y_test_com1)
print('Using best parameters:',best_param)
print("Accuracy with SMOTE & Grid Search & CV: %.2f%%" % (acc * 100.0))
print('*'*80)
print(classification_report(y_test_com1, preds))
print(confusion_matrix(y_test_com1, preds))
## ROC curve
y_pred_proba = model.predict_proba(X_test_com1)[::,1]
fpr, tpr, _ = roc_curve(y_test_com1,  y_pred_proba)
auc = metrics.roc_auc_score(y_test_com1, y_pred_proba)
plt.plot(fpr,tpr,label="Gs-Smote-cv-DT, Area="+str(np.round(auc,3)))
```

- LET's see the o/p we got –

1. Let's check Accuracy for validation & Test along with Precision, Recall & F1 Score
2. ROC/AUC curve.

```
GRID SEARCH WITH SMOTE & CROSS VALIDATION -- DT:
Using best parameters: {'criterion': 'entropy', 'max_depth': 28}
Accuracy with SMOTE & Grid Search & CV: 99.31%
**************************************************************
              precision    recall  f1-score   support

           0       1.00      0.99      1.00    255013
           1       0.14      0.97      0.25       297

    accuracy                           0.99    255310
   macro avg       0.57      0.98      0.62    255310
weighted avg       1.00      0.99      1.00    255310

[[253261   1752]
 [    10    287]]
**************************************************************
```



3.3.2.5 Decision Tree Statistics

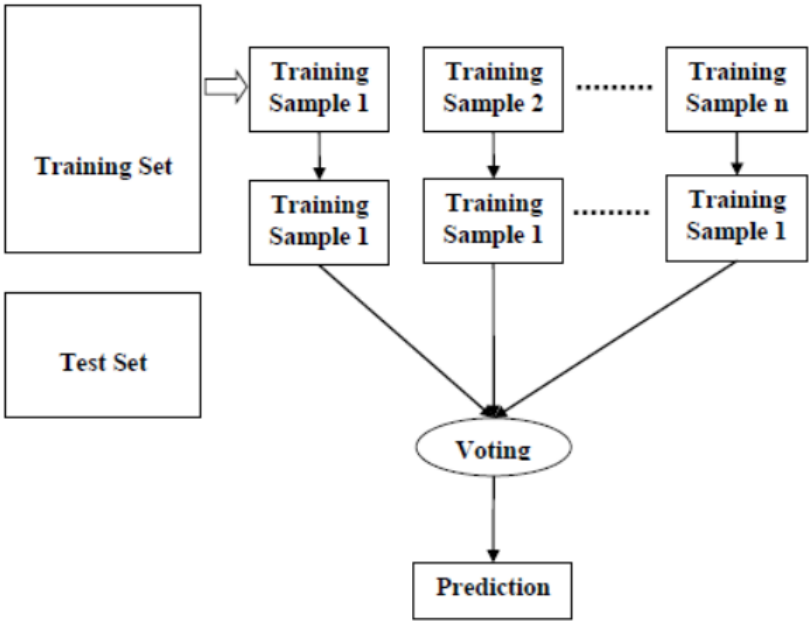
	Method	Testing on	ACCURACY(%)	PRECISION	RECALL	F1-SCORE	ROC Score
DT	Without Treatement	Validation	99.73	0.98	0.98	0.98	0.99
		Test	99.08	0.11	0.96	0.2	0.977
	SMOTE on Train	Test	99.3	0.14	0.95	0.24	0.971
	SMOTE on Train & Test	Test	99.04	0.99	0.99	0.99	0.99
	Under Sampling	Test	98.99	0.1	0.97	0.18	0.978
	Cross Validation	Test	99.3				0.978
	CV with Grid search	Test	99.31	0.14	0.97	0.25	0.98

3.3.3 Random Forest Model

3.3.3.1 Working Principle

Random forest is a supervised learning algorithm which is used for both classification as well as regression. But however, it is mainly used for classification problems. As we know that a forest is made up of trees and more trees means more robust forest. Similarly, random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

We can understand the working of Random Forest algorithm with the help of following steps –
Step 1 – First, start with the selection of random samples from a given dataset.
Step 2 – Next, this algorithm will construct a decision tree for every sample. Then it will get the prediction result from every decision tree.
Step 3 – In this step, voting will be performed for every predicted result.
Step 4 – At last, select the most voted prediction result as the final prediction result.



HYPERPARAMETERS –

#1: **max_depth**- The max_depth of a tree in Random Forest is defined as the longest path between the root node and the leaf node. can limit up to what depth I want every tree in my random forest to grow.
#2: **min_sample_split**- a parameter that tells the decision tree in a random forest the minimum required number of observations in any given node in order to split it.default value of the minimum_sample_split is assigned to 2. This means that if any terminal node has more than two observations and is not a pure node, we can split it further into subnodes.
#3: **max_terminal_nodes**- This hyperparameter sets a condition on the splitting of the nodes in the tree and hence restricts the growth of the tree. If after splitting we have more terminal nodes than the specified number of terminal nodes, it will stop the splitting and the tree will not grow further.
#4: **min_samples_leaf-** This Random Forest hyperparameter specifies the minimum number of samples that should be present in the leaf node after splitting a node.
#5: **n_estimators**- Helps to choose the number of trees. choosing a large number of estimators in a random forest model is not the best idea. Although it will not degrade the model, it can save you the computational complexity.
#6: **max_samples**- determines what fraction of the original dataset is given to any individual tree.
#7: **max_features** - This resembles the number of maximum features provided to each tree in a random forest.

3.3.3.2 Model Fitting

 # Importing Libraries to build model-

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report , f1_score, accuracy_score,
roc_auc_score, roc_curve
from confusionMatrix import plotConfusionMatrix
```

 # To treat data imbalance, call SMOTE & UnderSampler-

```
sm1 = SMOTE(random_state = 2)
X_sm1, y_sm1 = sm1.fit_sample(X_train_com1, y_train_com1.ravel())
un = RandomUnderSampler(random_state=2)
X_sm3, y_sm3 = un.fit_sample(X_train_com1, y_train_com1.ravel())
```

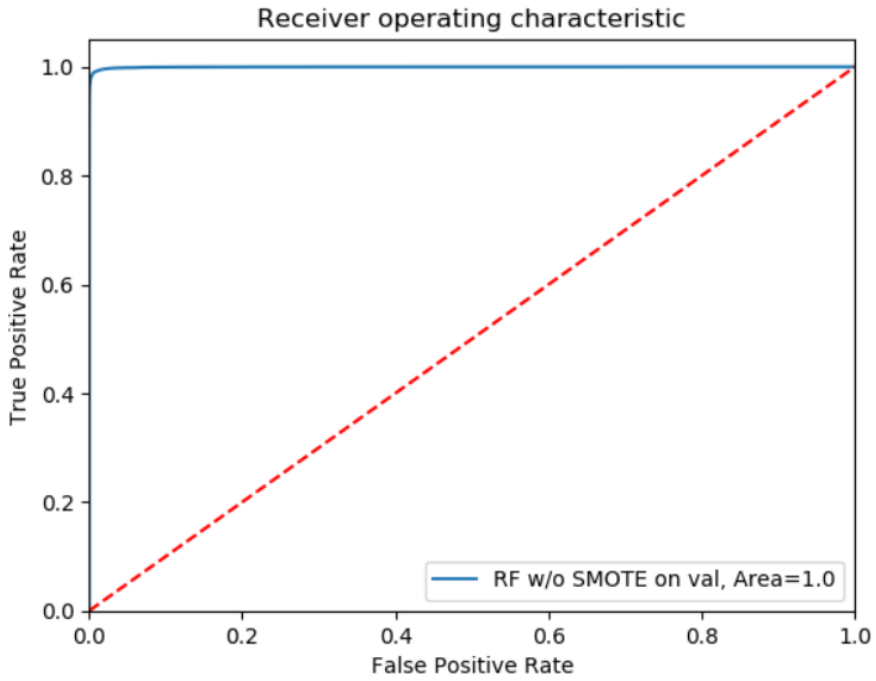
 # Model fitting without imbalance treatment-

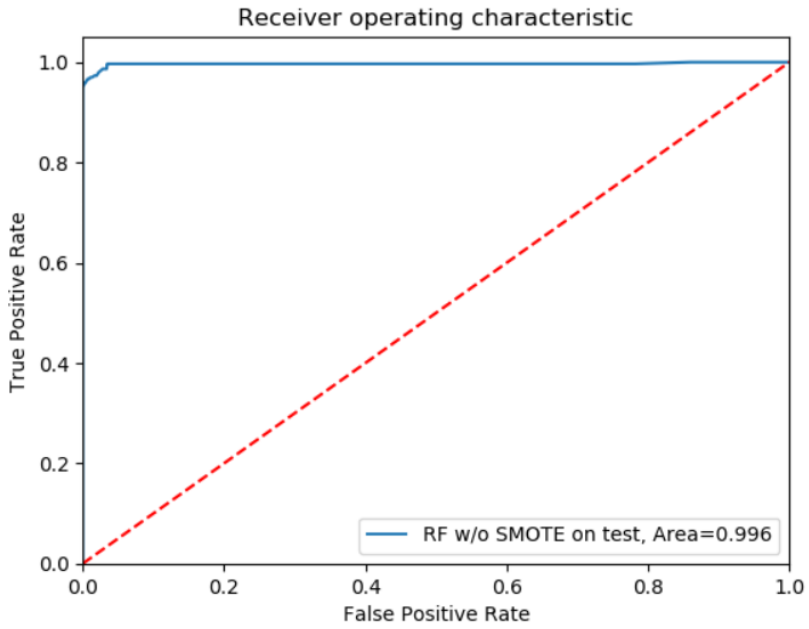
```
# Create the model with 100 trees
```

```
model = RandomForestClassifier(n_estimators=100,
                        bootstrap = True,
                        max_features = 'sqrt')
# Fit on training data
model.fit(X_train_com1, y_train_com1)
print('*'*80)
print('Random Forest:')
# Actual class predictions
probs = model.predict(X_test_com1)
probsv = model.predict(X_val1)
# Probabilities for each class
rf_probsv = model.predict_proba(X_val1)[:, 1]
rf_probs = model.predict_proba(X_test_com1)[:, 1]
acc_rfv = model.score(X_val1, y_val1)
acc_rf = model.score(X_test_com1, y_test_com1)
print("Accuracy with Random forest without SMOTE on val: %.2f%%" % (acc_rfv * 100.0))
print("Accuracy with Random forest without SMOTE on test: %.2f%%" % (acc_rf * 100.0))
print("Classification matrix on Val")
print(classification_report(y_val1, probsv))
print("Classification matrix on Val")
print(classification_report(y_test_com1, probs))
```

- LET's see the o/p we got –
  1. Let's check Accuracy for validation & Test along with Precision, Recall & F1 Score
  2. ROC/AUC curve.

```
Random Forest:
Accuracy with Random forest without SMOTE on val: 99.53%
Accuracy with Random forest without SMOTE on test: 98.92%
************************************************************
Classification matrix on Val
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    164668
           1       1.00      0.94      0.97     13755

    accuracy                           1.00    178423
   macro avg       1.00      0.97      0.98    178423
weighted avg       1.00      1.00      1.00    178423

************************************************************
Classification matrix on Val
              precision    recall  f1-score   support

           0       1.00      0.99      0.99    255013
           1       0.09      0.97      0.17       297

    accuracy                           0.99    255310
   macro avg       0.55      0.98      0.58    255310
weighted avg       1.00      0.99      0.99    255310

************************************************************
```





FEATURE IMPORTANCE

```
import pandas as pd
# Extract feature importances
fin_without_SMOTE = pd.DataFrame({'feature': list(X_train_com1.columns),
                    'importance': model.feature_importances_}).\
                     sort_values('importance', ascending= False)
fin_without_SMOTE.head(5)
```

	feature	importance
16	recoveries	0.262319
17	last_pymnt_amnt	0.133930
13	total_pymnt	0.093959
12	out_prncp	0.070553
22	installment	0.032133

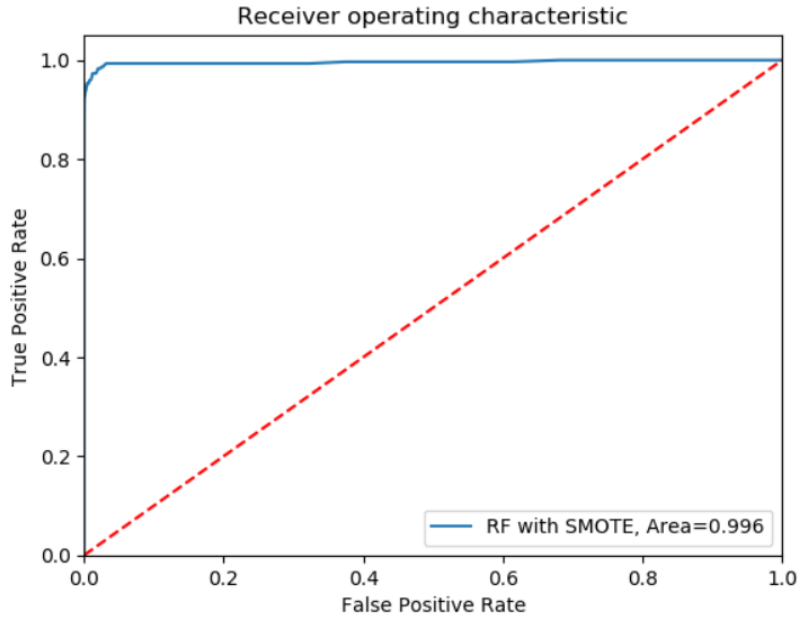
#Model fitting with imbalance treatment using SMOTE on Train –

```
model = RandomForestClassifier(n_estimators=100,
                               bootstrap = True,
                               max_features = 'sqrt')
# Fit on training data
model.fit(X_sm1, y_sm1)
print('Random Forest:')
# Actual class predictions
probs = model.predict(X_test_com1)
# Probabilities for each class
rf_probs = model.predict_proba(X_test_com1)[:, 1]
acc_rf = model.score(X_test_com1, y_test_com1)
print("Accuracy with Random forest: %.2f%%" % (acc * 100.0))
print(confusion_matrix(y_test_com1, probs))
print(classification_report(y_test_com1, probs))
## ROC curve
fpr_rf, tpr_rf, _rf = roc_curve(y_test_com1,  rf_probs)
auc_rf = metrics.roc_auc_score(y_test_com1, rf_probs)
plt.plot(fpr_rf,tpr_rf,label="RF with SMOTE, Area="+str(np.round(auc_rf,3)))
```

- LET's see the o/p we got –

3. Let's check Accuracy for validation & Test along with Precision, Recall & F1 Score
4. ROC/AUC curve.

```
************************************************************
Random Forest:
Accuracy with Random forest: 99.31%
************************************************************
[[254936     77]
 [    25    272]]
************************************************************
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    255013
           1       0.78      0.92      0.84       297

    accuracy                           1.00    255310
   macro avg       0.89      0.96      0.92    255310
weighted avg       1.00      1.00      1.00    255310

************************************************************
```



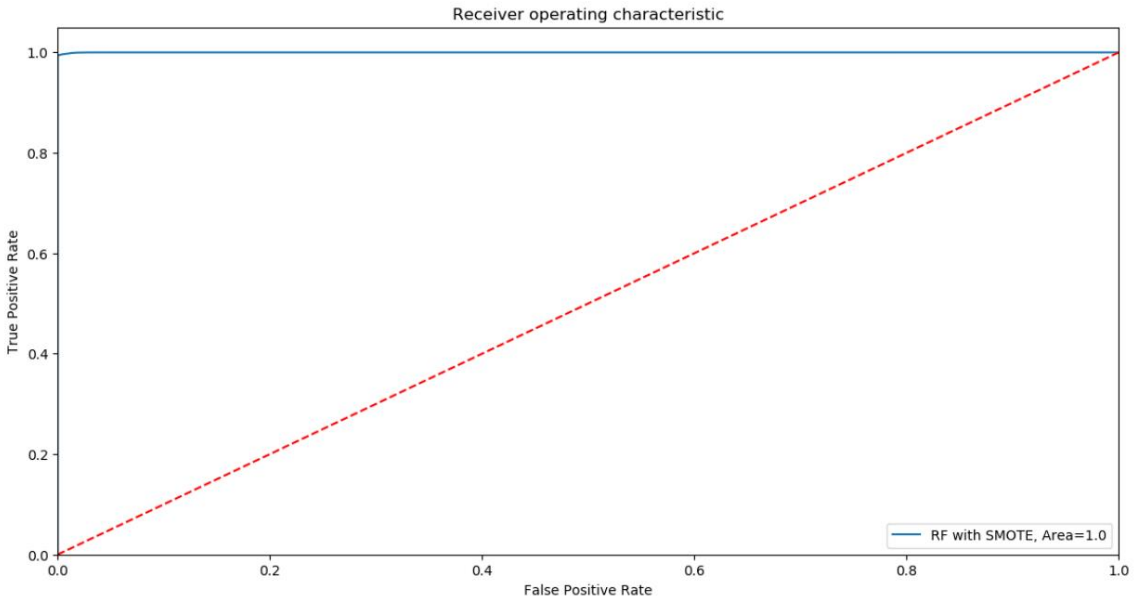
 # Model fitting with imbalance treatment using SMOTE on Train & Test –

```
model = RandomForestClassifier(n_estimators=100,
                               bootstrap = True,
                               max_features = 'sqrt')
# Fit on training data
model.fit(X_sm1, y_sm1)
print('*'*80)
print('Random Forest:')
probs = model.predict(X_smt1)
# Probabilities for each class
rf_probs = model.predict_proba(X_smt1)[:, 1]
acc_rf = model.score(X_smt1, y_smt1)
print("Accuracy with Random forest: %.2f%%" % (acc * 100.0))
print(confusion_matrix(y_smt1, probs))
print(classification_report(y_smt1, probs))
fpr_rf, tpr_rf, _rf = roc_curve(y_smt1,  rf_probs)
auc_rf = metrics.roc_auc_score(y_smt1, rf_probs)
plt.plot(fpr_rf,tpr_rf,label="RF with SMOTE, Area="+str(np.round(auc_rf,3)))
```

- LET's see the o/p we got –

1. Let's check Accuracy for validation & Test along with Precision, Recall & F1 Score
2. ROC/AUC curve.

```
************************************************************
Random Forest:
Accuracy with Random forest: 99.04%
************************************************************
[[254979     34]
 [  1961 253052]]
************************************************************
              precision    recall  f1-score   support

           0       0.99      1.00      1.00    255013
           1       1.00      0.99      1.00    255013

    accuracy                           1.00    510026
   macro avg       1.00      1.00      1.00    510026
weighted avg       1.00      1.00      1.00    510026

************************************************************
```

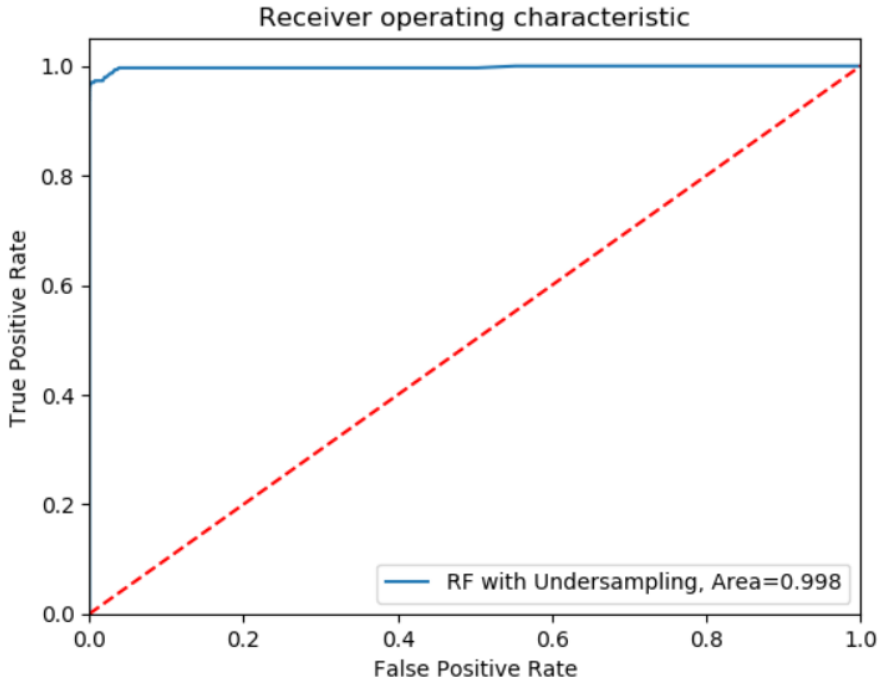


 # Model fitting with imbalance treatment using Undersampling on Train –

```
model = RandomForestClassifier(n_estimators=100,
                               bootstrap = True,
                               max_features = 'sqrt')
# Fit on training data
model.fit(X_sm3, y_sm3)
print('Random Forest:')
probs = model.predict(X_test_com1)
# Probabilities for each class
rf_probs = model.predict_proba(X_test_com1)[:, 1]
acc_rf = model.score(X_test_com1, y_test_com1)
print("Accuracy with Random forest and undersampling: %.2f%%" % (acc * 100.0))
print(confusion_matrix(y_test_com1, probs))
print(classification_report(y_test_com1, probs))
fpr_rf, tpr_rf, _rf = roc_curve(y_test_com1,  rf_probs)
auc_rf = metrics.roc_auc_score(y_test_com1, rf_probs)
plt.plot(fpr_rf,tpr_rf,label="RF with Undersampling, Area="+str(np.round(auc_rf,3)))
```

- LET's see the o/p we got –
  1. Let's check Accuracy for validation & Test along with Precision, Recall & F1 Score
  2. ROC/AUC curve.

```
************************************************************
Random Forest:
Accuracy with Random forest and undersampling: 98.99%
************************************************************
[[254813    200]
 [    10    287]]
************************************************************
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    255013
           1       0.59      0.97      0.73       297

    accuracy                           1.00    255310
   macro avg       0.79      0.98      0.87    255310
weighted avg       1.00      1.00      1.00    255310

************************************************************
```



FEATURE IMPORTANCE-

```
# Extract feature importances
finnimp_with_SMOTE = pd.DataFrame({'feature': list(X_sm1.columns),
                    'importance': model.feature_importances_}).\
                     sort_values('importance', ascending= False)
# Display
finnimp_with_SMOTE.head(5)
```

	feature	importance
12	out_prncp	0.159372
16	recoveries	0.136910
17	last_pymnt_amnt	0.133417
167	last_pymnt_d_Jan-2016	0.127967
13	total_pymnt	0.054666

3.3.3.3 Random Forest Statistics

	Method	Testing on	ACCURACY(%)	PRECISION	RECALL	F1-SCORE	ROC Score
RF	Without Treatement	Validation	99.53	1	0.94	0.97	1
		Test	98.92	0.09	0.97	0.17	0.996
	SMOTE on Train	Test	99.31	0.78	0.92	0.84	0.996
	SMOTE on Train & Test	Test	99.04	1	0.99	1	1
	Under Sampling	Test	98.99	0.59	0.97	0.73	0.998

3.3.4 XG Boost Model

3.3.4.1 Working Principle

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples.

Parameters

booster [default=gbtree)- elect the type of model to run at each iteration. It has 2 options:gbtree: 1. silent [default=0]tree-based models 2.gblinear: linear models
silent [default=0]- Silent mode is activated is set to 1, i.e. no running messages will be printed. It's generally good to keep it 0 as the messages might help in understanding the model.
nthread [default to maximum number of threads available if not set]-his is used for parallel processing and number of cores in the system should be entered
eta [default=0.3]- Makes the model more robust by shrinking the weights on each step. Typical final values to be used: 0.01-0.2
min_child_weight [default=1]- Defines the minimum sum of weights of all observations required in a child. This is similar to min_child_leaf in GBM but not exactly. This refers to min "sum of weights" of observations while GBM has min "number of observations".
max_depth [default=6]- The maximum depth of a tree, same as GBM. Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.

max_leaf_nodes- The maximum number of terminal nodes or leaves in a tree. Can be defined in place of max_depth. Since binary trees are created, a depth of 'n' would produce a maximum of 2^n leaves.
gamma [default=0]- A node is split only when the resulting split gives a positive reduction in the loss function. Gamma specifies the minimum loss reduction required to make a split.
Makes the algorithm conservative. The values can vary depending on the loss function and should be tuned.
max_delta_step [default=0]- In maximum delta step we allow each tree's weight estimation to be. If the value is set to 0, it means there is no constraint. If it is set to a positive value, it can help making the update step more conservative.
subsample [default=1]- Same as the subsample of GBM. Denotes the fraction of observations to be randomly samples for each tree. Typical values: 0.5-1
colsample_bytree [default=1]- Similar to max_features in GBM. Denotes the fraction of columns to be randomly samples for each tree. Typical values: 0.5-1.
colsample_bylevel [default=1]- Denotes the subsample ratio of columns for each split, in each level.
lambda [default=1]- L2 regularization term on weights (analogous to Ridge regression) This used to handle the regularization part of XGBoost. Though many data scientists don't use it often, it should be explored to reduce overfitting.
alpha [default=0]- L1 regularization term on weight (analogous to Lasso regression) Can be used in case of very high dimensionality so that the algorithm runs faster when implemented.
scale_pos_weight [default=1]- A value greater than 0 should be used in case of high class imbalance as it helps in faster convergence.

3.3.5.2 Model Fitting & Prediction

 # Importing Libraries to build model-

```
from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix, classification_report , f1_score, accuracy_score,
roc_auc_score, roc_curve
from confusionMatrix import plotConfusionMatrix
```

 # To treat data imbalance, call SMOTE & UnderSampler-

```
sm1 = SMOTE(random_state = 2)
X_sm1, y_sm1 = sm1.fit_sample(X_train_com1, y_train_com1.ravel())
un = RandomUnderSampler(random_state=2)
X_sm3, y_sm3 = un.fit_sample(X_train_com1, y_train_com1.ravel())
```

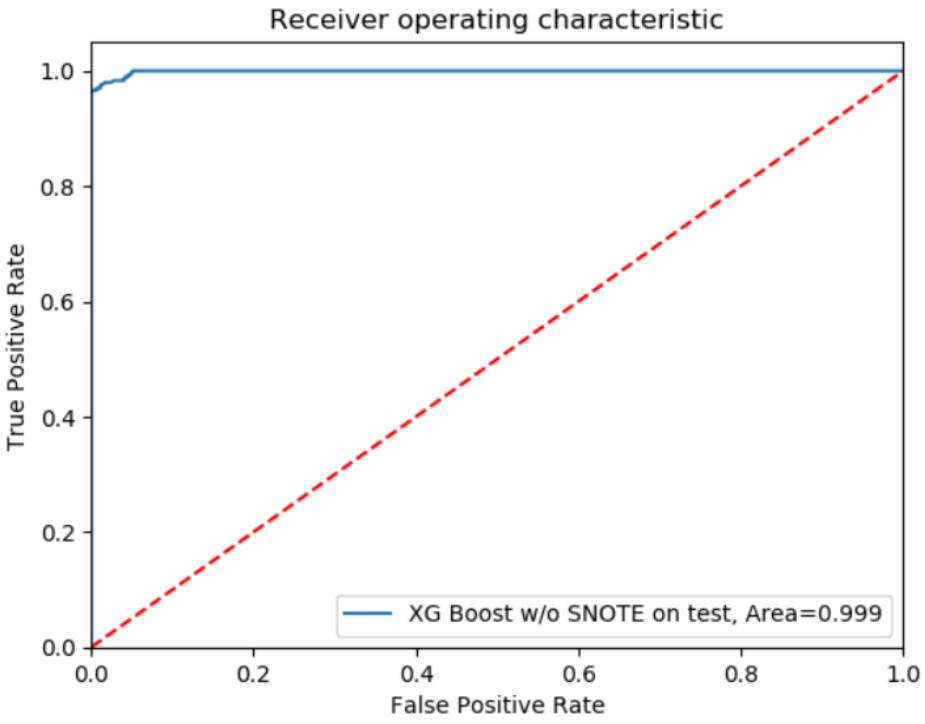
 # Model fitting without imbalance treatment-

```
#instantiate model and train
model_G = XGBClassifier()
model_G.fit(X_train_com1, y_train_com1)
#learning_rate =0.01, n_estimators=5000, max_depth=4,min_child_weight=6, gamma=0,
subsample=0.8,colsample_bytree=0.8,
#reg_alpha=0.005, objective= 'binary:logistic',nthread=4,scale_pos_weight=1,seed=27
y_pred = model_G.predict(X_test_com1)
y_predv = model_G.predict(X_val1)
preds = [round(value) for value in y_pred]
accGv = model_G.score(X_val1,y_val1)
accG = model_G.score(X_test_com1,y_test_com1)
print('XG Bosst:')
print("Accuracy without SMOTE on VAl: %.2f%%" % (accGv * 100.0))
print("Accuracy without SMOTE on Test: %.2f%%" % (accG * 100.0))
print("Calssification report for validation")
print(classification_report(y_val1, y_predv))
```

```
print("Calssification report for test")
print(classification_report(y_test_com1, y_pred))
y_pred_proba_Gv = model_G.predict_proba(X_val1)[::,1]
fpr_G, tpr_G, _G = roc_curve(y_val1,  y_pred_proba_Gv)
auc_Gv = metrics.roc_auc_score(y_val1, y_pred_proba_Gv)
plt.plot(fpr_G,tpr_G,label="XG Boost w/o SNOTE on val, Area="+str(np.round(auc_Gv,3)))
y_pred_proba_G = model_G.predict_proba(X_test_com1)[::,1]
fpr_G, tpr_G, _G = roc_curve(y_test_com1,  y_pred_proba_G)
auc_G = metrics.roc_auc_score(y_test_com1, y_pred_proba_G)
plt.plot(fpr_G,tpr_G,label="XG Boost w/o SNOTE on test, Area="+str(np.round(auc_G,3)))
```

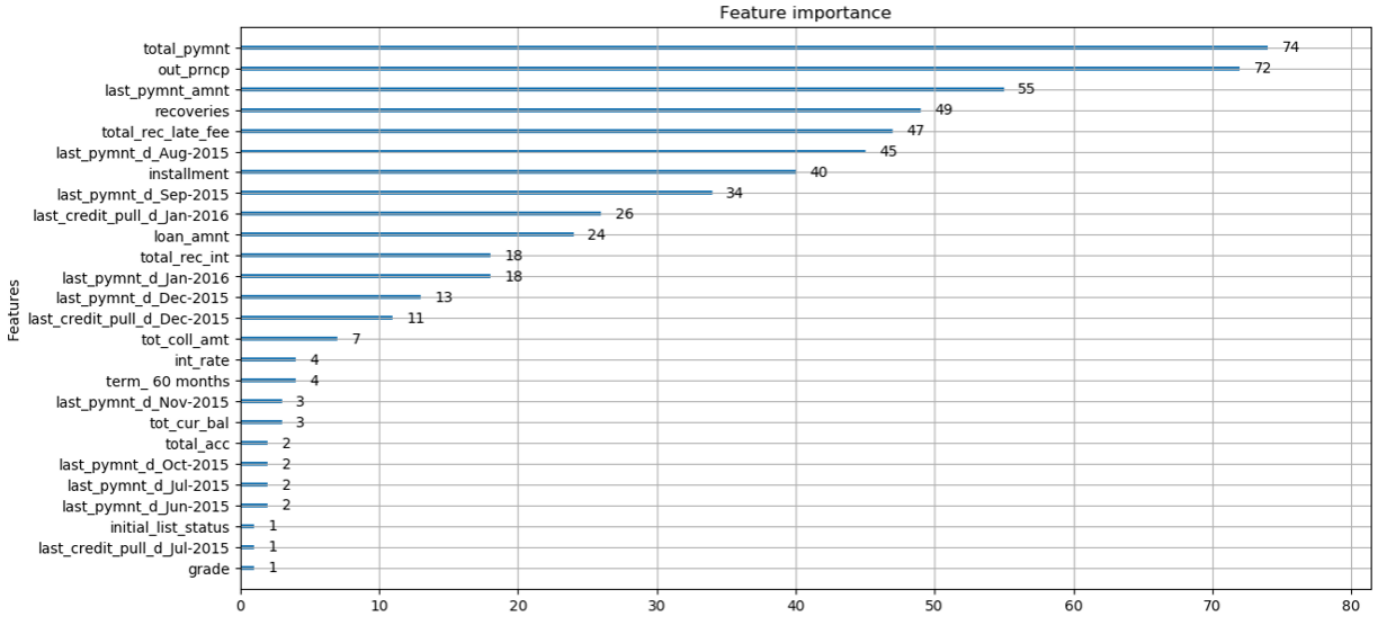
- LET's see the o/p we got –
    1. Let's check Accuracy for validation & Test along with Precision, Recall & F1 Score
    2. ROC/AUC curve.

```
XG Bosst:
Accuracy without SMOTE on VAl: 99.58%
Accuracy without SMOTE on Test: 99.99%
**************************************************************
Calssification report for validation
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    164668
           1       1.00      0.95      0.97     13755

    accuracy                           1.00    178423
   macro avg       1.00      0.97      0.98    178423
weighted avg       1.00      1.00      1.00    178423

**************************************************************
Calssification report for test
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    255013
           1       0.98      0.97      0.97       297

    accuracy                           1.00    255310
   macro avg       0.99      0.98      0.99    255310
weighted avg       1.00      1.00      1.00    255310

**************************************************************
```



Feature Importance

```
from xgboost import plot_importance
# plot feature importance
plt.rcParams["figure.figsize"] = (14, 7)
plot_importance(model_G)
```

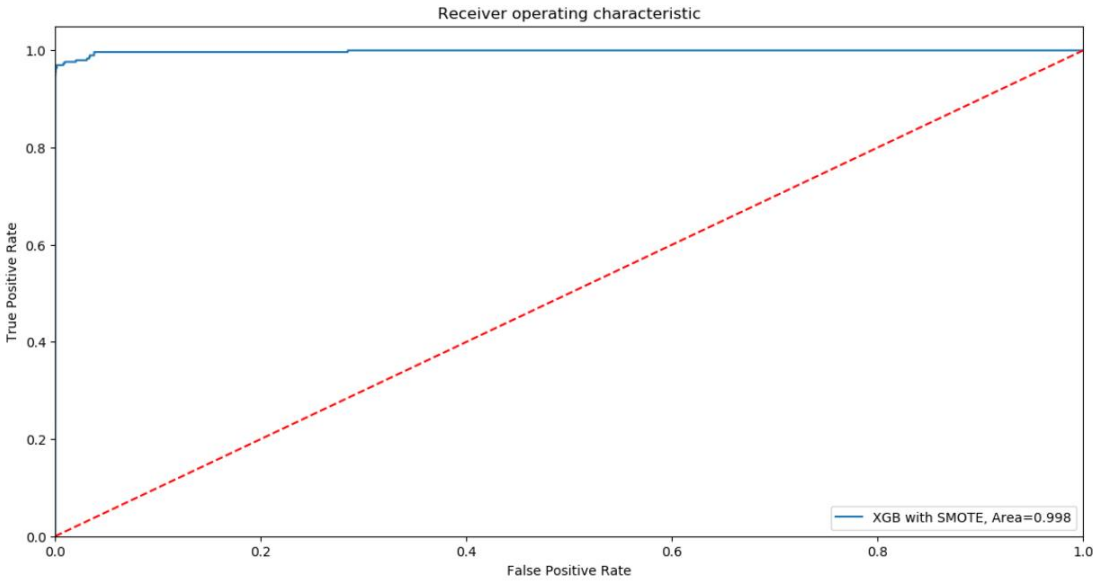


#Model fitting with imbalance treatment using SMOTE on Train –

```
model_G = XGBClassifier()
model_G.fit(X_sm1, y_sm1)
y_pred = model_G.predict(X_test_com1)
preds = [round(value) for value in y_pred]
accG = model_G.score(X_test_com1,y_test_com1)
print('XG Bosst:')
print("Accuracy with SMOTE: %.2f%%" % (accG * 100.0))
print(classification_report(y_test_com1, y_pred))
print(confusion_matrix(y_test_com1, y_pred))
y_pred_proba_G = model_G.predict_proba(X_test_com1)[::,1]
fpr_G, tpr_G, _G = roc_curve(y_test_com1,  y_pred_proba_G)
auc_G = metrics.roc_auc_score(y_test_com1, y_pred_proba_G)
plt.plot(fpr_G,tpr_G,label="XGB with SMOTE, Area="+str(np.round(auc_G,3)))
```

- LET's see the o/p we got –
  1. Let's check Accuracy for validation & Test along with Precision, Recall & F1 Score
  2. ROC/AUC curve.

```
XG Bosst:
Accuracy with SMOTE: 99.98%
**************************************************************************
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    255013
           1       0.89      0.95      0.92       297

    accuracy                           1.00    255310
   macro avg       0.94      0.97      0.96    255310
weighted avg       1.00      1.00      1.00    255310

**************************************************************************
[[254978     35]
 [    15    282]]
**************************************************************************
```

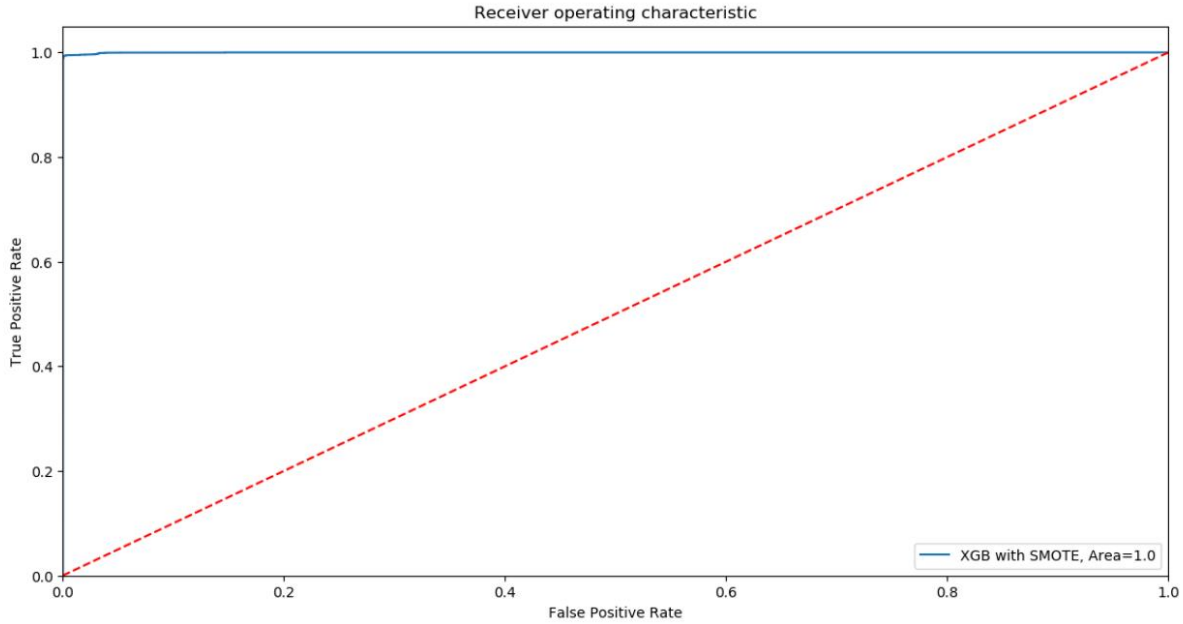


 # Model fitting with imbalance treatment using SMOTE on Train & Test –

```
y_pred = model_G.predict(X_smt1)
preds = [round(value) for value in y_pred]
accG = model_G.score(X_smt1,y_smt1)
print('XG Bosst:')
print("Accuracy with SMOTE: %.2f%%" % (accG * 100.0))
print(classification_report(y_smt1, y_pred))
print(confusion_matrix(y_smt1, y_pred))
y_pred_proba_G = model_G.predict_proba(X_smt1)[::,1]
fpr_G, tpr_G, _G = roc_curve(y_smt1,  y_pred_proba_G)
auc_G = metrics.roc_auc_score(y_smt1, y_pred_proba_G)
plt.plot(fpr_G,tpr_G,label="XGB with SMOTE, Area="+str(np.round(auc_G,3)))
```

- LET's see the o/p we got –
  1. Let's check Accuracy for validation & Test along with Precision, Recall & F1 Score
  2. ROC/AUC curve.

```
************************************************************
XG Bosst:
Accuracy with SMOTE: 99.28%
************************************************************
              precision    recall  f1-score   support

           0       0.99      1.00      0.99    255013
           1       1.00      0.99      0.99    255013

    accuracy                           0.99    510026
   macro avg       0.99      0.99      0.99    510026
weighted avg       0.99      0.99      0.99    510026

************************************************************
[[254978     35]
 [  3645 251368]]
************************************************************
```

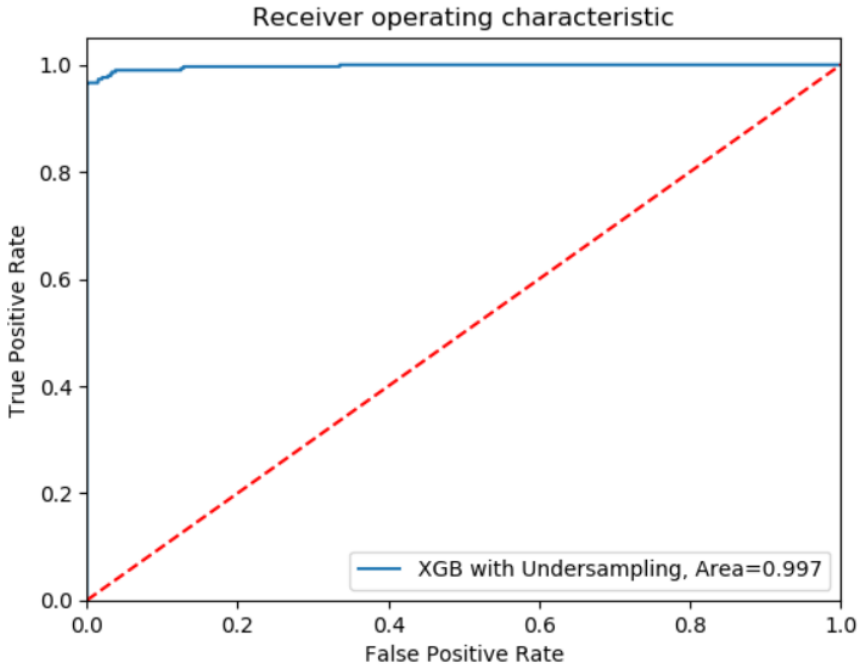


 # Model fitting with imbalance treatment using Undersampling on Train –

```
model_G = XGBClassifier()
model_G.fit(X_sm3, y_sm3)
#learning_rate =0.01, n_estimators=5000, max_depth=4,min_child_weight=6, gamma=0,
subsample=0.8,colsample_bytree=0.8,
#reg_alpha=0.005, objective= 'binary:logistic',nthread=4,scale_pos_weight=1,seed=27

y_pred = model_G.predict(X_test_com1)
preds = [round(value) for value in y_pred]
accG = model_G.score(X_test_com1,y_test_com1)
print('XG Bosst:')
print("Accuracy with UnderSampling: %.2f%%" % (accG * 100.0))
print(classification_report(y_test_com1, y_pred))
print(confusion_matrix(y_test_com1, y_pred))
y_pred_proba_G = model_G.predict_proba(X_test_com1)[::,1]
fpr_G, tpr_G, _G = roc_curve(y_test_com1,  y_pred_proba_G)
auc_G = metrics.roc_auc_score(y_test_com1, y_pred_proba_G)
plt.plot(fpr_G,tpr_G,label="XGB with Undersampling, Area="+str(np.round(auc_G,3)))
```

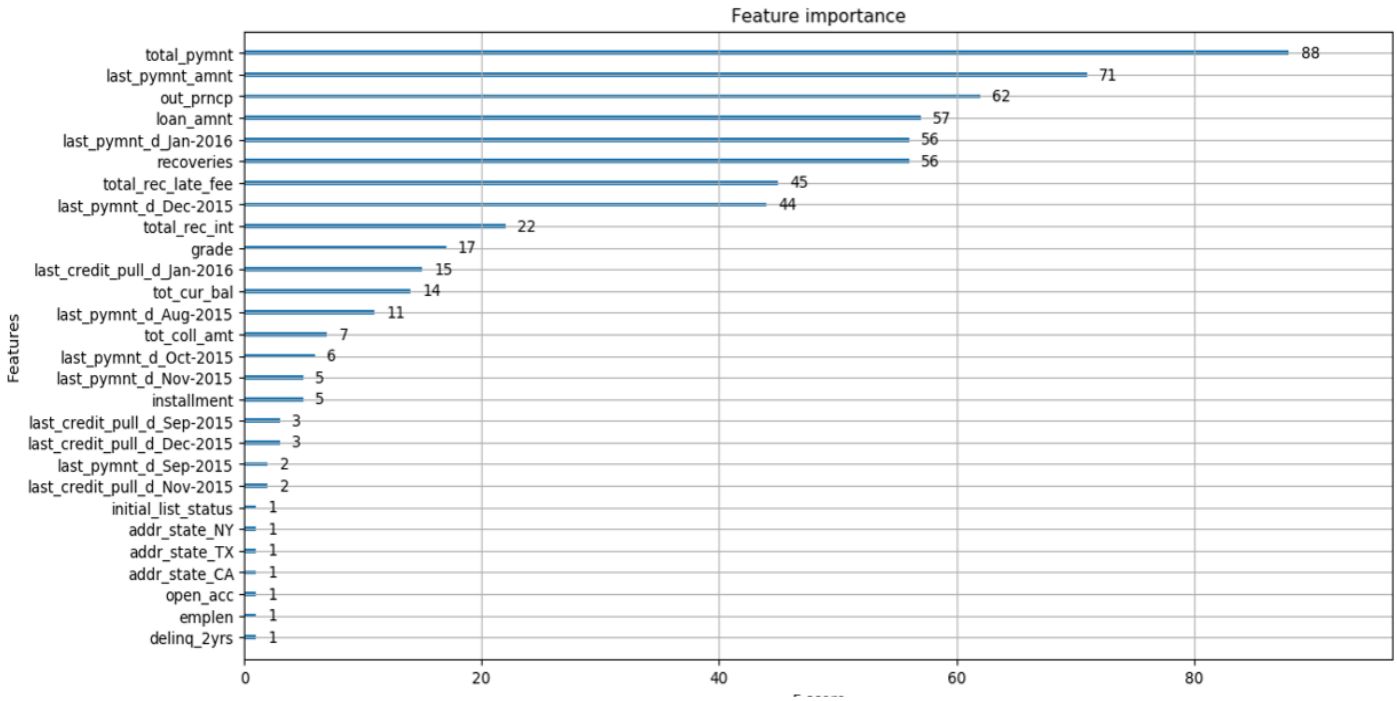
- LET's see the o/p we got –
  1. Let's check Accuracy for validation & Test along with Precision, Recall & F1 Score
  2. ROC/AUC curve.

```
************************************************************
XG Bosst:
Accuracy with UnderSampling: 99.97%
************************************************************
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    255013
           1       0.82      0.97      0.88       297

    accuracy                           1.00    255310
   macro avg       0.91      0.98      0.94    255310
weighted avg       1.00      1.00      1.00    255310

************************************************************
[[254948     65]
 [    10    287]]
************************************************************
```



FEATURE IMPORTANCE using SMOTE-

```
from xgboost import plot_importance
# plot feature importance
plt.rcParams["figure.figsize"] = (14, 7)
plot_importance(model_G)
```



3.3.4.3 XG BOOST Statistics

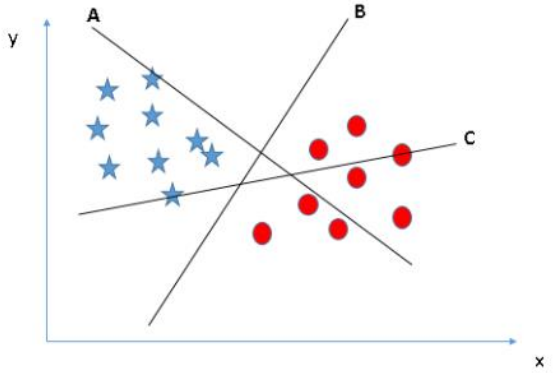
	Method	Testing on	ACCURACY(%)	PRECISION	RECALL	F1-SCORE	ROC Score
XGB	Without Treatement	Validation	99.58	1	0.95	0.97	1
		Test	99.99	0.98	0.97	0.97	0.999
	SMOTE on Train	Test	99.98	0.89	0.95	0.92	0.998
	SMOTE on Train & Test	Test	99.28	1	0.99	0.99	1
	Under Sampling	Test	99.97	0.82	0.97	0.88	0.997

3.3.5 Support Vector Machine

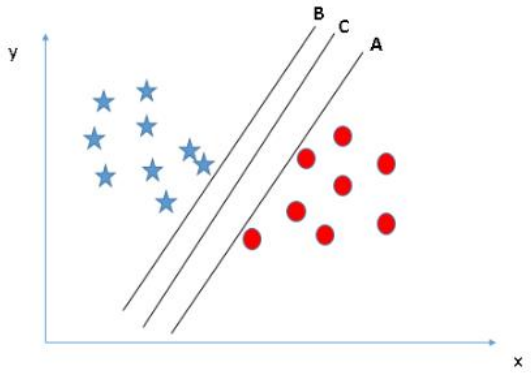
It is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well

3.3.5.1 Working principle

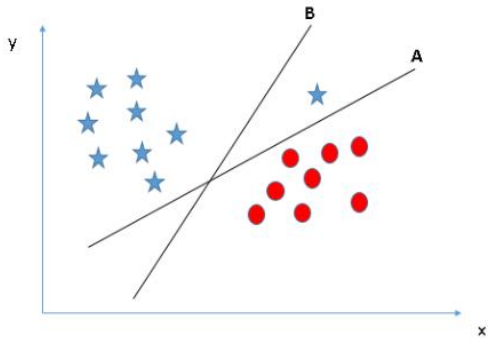
Identify the right hyper-plane (Scenario-1)- Select the hyper-plane which segregates the two classes better". In this scenario, hyper-plane "B" has excellently performed this job.



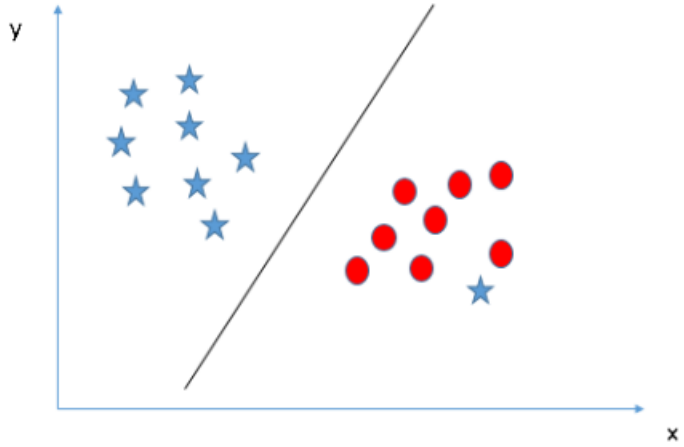
Identify the right hyper-plane (Scenario-2)- maximizing the distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane. This distance is called as Margin.



Identify the right hyper-plane (Scenario-3)- SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin. Here, hyper-plane B has a classification error and A has classified all correctly. Therefore, the right hyper-plane is A.



Can we classify two classes (Scenario-4)- The SVM algorithm has a feature to ignore outliers and find the hyper-plane that has the maximum margin. Hence, we can say, SVM classification is robust to outliers.



3.3.5.2 Model Fitting & Prediction

Note: For operations using SVM we have taken random sample consisting of 1/10th of actual dataset because of memory issues which we faced while running SVM.

 # Importing Libraries to build model-

```
from sklearn.metrics import confusion_matrix, classification_report, f1_score
from imblearn.over_sampling import SMOTE
from sklearn.svm import SVC #Support vector classifier
from sklearn.metrics import roc_auc_score, roc_curve
from confusionMatrix import plotConfusionMatrix
```

 # To treat data imbalance, call SMOTE & UnderSampler-

```
sm1 = SMOTE(random_state = 2)
X_sm1, y_sm1 = sm1.fit_sample(X_train_com1, y_train_com1.ravel())
un = RandomUnderSampler(random_state=2)
X_sm3, y_sm3 = un.fit_sample(X_train_com1, y_train_com1.ravel())
```

After performing SMOTE on train data following were results:

Performing SMOTE on Train -

```
print('Before OverSampling, X: {}'.format(X_train_com11.shape))
print('Before OverSampling, y: {}'.format(y_train_com11.shape))
print("Before OverSampling, counts of '1': {}".format(sum(y_train_com11 == 1)))
print("Before OverSampling, counts of '0': {}".format(sum(y_train_com11 == 0)))
print('\n')
print('With imbalance treatment:'.upper())
print('After OverSampling, X: {}'.format(X_sm11.shape))
print('After OverSampling, y: {}'.format(y_sm11.shape))
print("After OverSampling, counts of '1': {}".format(sum(y_sm11 == 1)))
print("After OverSampling, counts of '0': {}".format(sum(y_sm11 == 0)))
```

```
Before OverSampling, X: (35764, 325)
Before OverSampling, y: (35764,)
Before OverSampling, counts of '1': 2777
Before OverSampling, counts of '0': 32987


WITH IMBALANCE TREATMENT:
After OverSampling, X: (65974, 325)
After OverSampling, y: (65974,)
After OverSampling, counts of '1': 32987
After OverSampling, counts of '0': 32987
```

It clearly indicates the need of treating data as it is highly imbalanced dataset.

After performing SMOTE on Test as well, below are results:
Performing SMOTE on test was required because of huge imbalance between minority & majority classes.

```
from imblearn.over_sampling import SMOTE
smt11 = SMOTE(random_state = 2)
X_smt11, y_smt11 = smt11.fit_sample(X_test_com11, y_test_com11.ravel())

print('Before OverSampling, X: {}'.format(X_test_com11.shape))
print('Before OverSampling, y: {}'.format(y_test_com11.shape))
print("Before OverSampling, counts of '1': {}".format(sum(y_test_com11 == 1)))
print("Before OverSampling, counts of '0': {}".format(sum(y_test_com11 == 0)))
print('\n')
print('With imbalance treatment:'.upper())
print('After OverSampling, X: {}'.format(X_smt11.shape))
print('After OverSampling, y: {}'.format(y_smt11.shape))
print("After OverSampling, counts of '1': {}".format(sum(y_smt11 == 1)))
print("After OverSampling, counts of '0': {}".format(sum(y_smt11 == 0)))
print('\n')
```

```
Before OverSampling, X: (25392, 325)
Before OverSampling, y: (25392,)
Before OverSampling, counts of '1': 29
Before OverSampling, counts of '0': 25363


WITH IMBALANCE TREATMENT:
After OverSampling, X: (50726, 325)
After OverSampling, y: (50726,)
After OverSampling, counts of '1': 25363
After OverSampling, counts of '0': 25363
```

After performing Undersampling on train, below are results:

```
un1 = RandomUnderSampler(random_state=2)
X_sm12, y_sm12 = un1.fit_sample(X_train_com11, y_train_com11.ravel())

print('Before Undersampling, X: {}'.format(X_train_com11.shape))
print('Before Undersampling, y: {}'.format(y_train_com11.shape))
print("Before Undersampling, counts of '1': {}".format(sum(y_train_com11 == 1)))
print("Before Undersampling, counts of '0': {}".format(sum(y_train_com11 == 0)))
print('\n')
print('With imbalance treatment:'.upper())
print('After Undersampling, X: {}'.format(X_sm12.shape))
print('After Undersampling, y: {}'.format(y_sm12.shape))
print("After Undersampling, counts of '1': {}".format(sum(y_sm12 == 1)))
print("After Undersampling, counts of '0': {}".format(sum(y_sm12 == 0)))
print('\n')
```

```
Before Undersampling, X: (35764, 325)
Before Undersampling, y: (35764,)
Before Undersampling, counts of '1': 2777
Before Undersampling, counts of '0': 32987


WITH IMBALANCE TREATMENT:
After Undersampling, X: (5554, 325)
After Undersampling, y: (5554,)
After Undersampling, counts of '1': 2777
After Undersampling, counts of '0': 2777
```

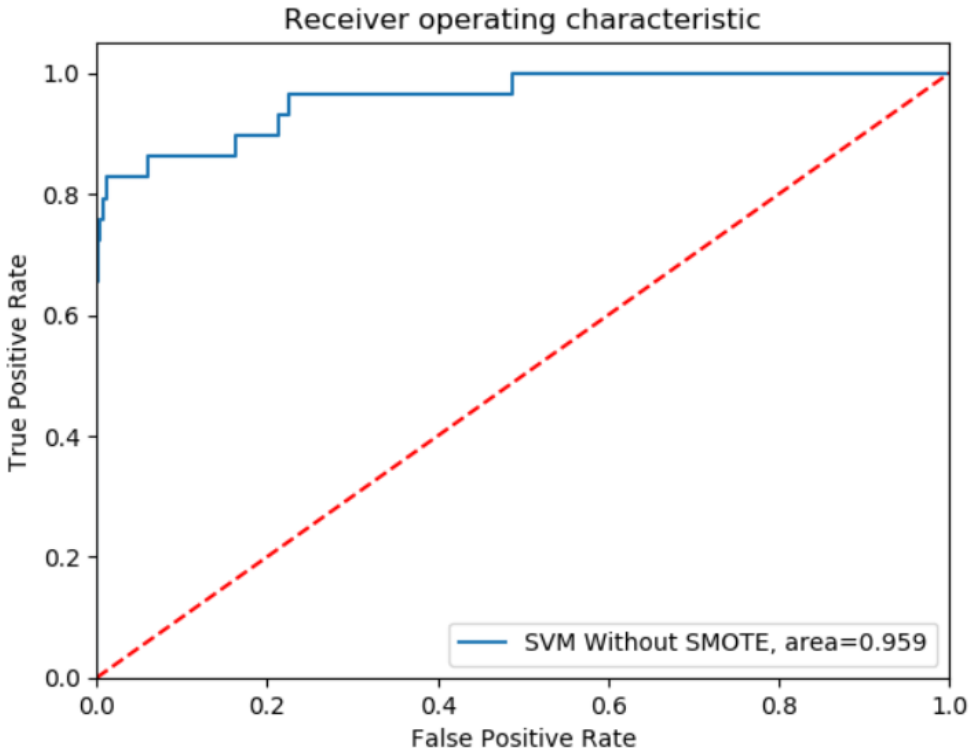
 # Model fitting without imbalance treatment-

```
svm = SVC() #SVC(kernel='linear')

clf = svm.fit(X_train_com11, y_train_com11.ravel())
score1 = svm.score(X_val11, y_val11)
score2 = svm.score(X_test_com11, y_test_com11)
pred1 = svm.predict(X_val11)
pred2 = svm.predict(X_test_com11)
# print classification report
print('Without SMOTE:'.upper())
print('Validation accuracy: ', score1)
print('test accuracy: ', score2)
print(confusion_matrix(y_test_com11, pred2))
print('F1 score val:\n', classification_report(y_val11, pred1))
print('F1 score test:\n', classification_report(y_test_com11, pred2))
# ROC Curve
svm_p = SVC(probability = True) # for probability
clf_p = svm_p.fit(X_train_com11, y_train_com11.ravel()) # for probability
pred_proba = svm_p.predict_proba(X_test_com11)[::,1]
fpr, tpr, _ = roc_curve(y_test_com11,  pred_proba)
auc = roc_auc_score(y_test_com11, pred_proba)
plt.plot(fpr,tpr,label="SVM Without SMOTE, area="+str(np.round(auc,3)))
```

- LET's see the o/p we got –

1. Let's check Accuracy for validation & Test along with Precision, Recall & F1 Score
2. ROC/AUC curve.

```
WITHOUT SMOTE:
Validation accuracy:  0.9912766314376782
test accuracy:  0.9992123503465659
********************************************************************************
[[25353    10]
 [   10    19]]
********************************************************************************
F1 score val:
               precision    recall  f1-score   support

           0       0.99      1.00      1.00     22097
           1       0.97      0.90      0.94      1747

    accuracy                           0.99     23844
   macro avg       0.98      0.95      0.97     23844
weighted avg       0.99      0.99      0.99     23844

********************************************************************************
F1 score test:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     25363
           1       0.66      0.66      0.66        29

    accuracy                           1.00     25392
   macro avg       0.83      0.83      0.83     25392
weighted avg       1.00      1.00      1.00     25392

********************************************************************************
```

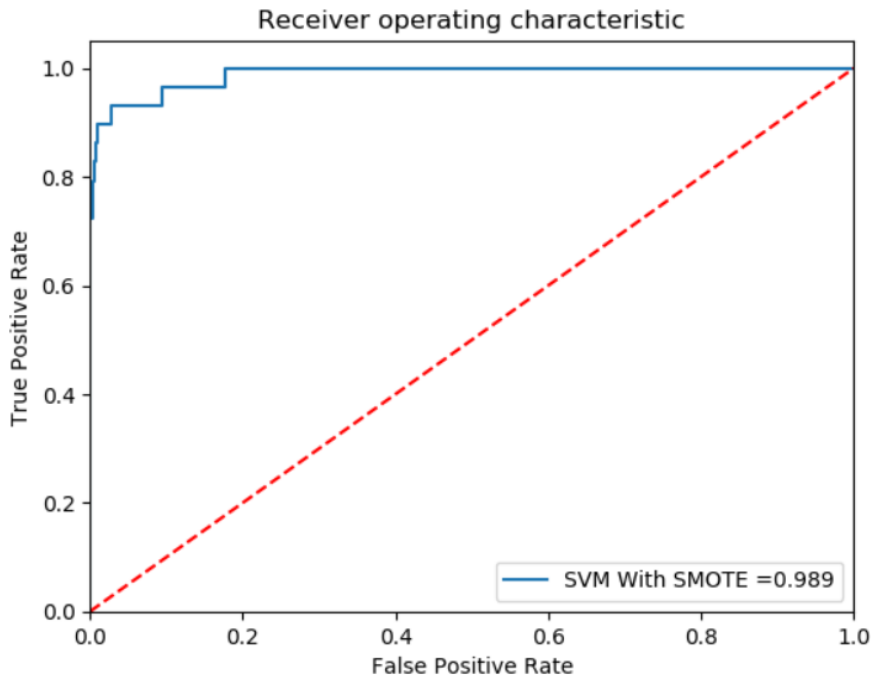


 # Model fitting with imbalance treatment applying SMOTE on train-

```
svm = SVC() #SVC(kernel='linear')
clf = svm.fit(X_sm11, y_sm11.ravel())
score1 = svm.score(X_val11, y_val11)
score2 = svm.score(X_test_com11, y_test_com11)
pred1 = svm.predict(X_val11)
pred2 = svm.predict(X_test_com11)
# print classification report
print('With SMOTE:'.upper())
print('Validation accuracy: ', score1)
print('test accuracy: ', score2)
print(confusion_matrix(y_test_com11, pred2))
print('F1 score val:\n', classification_report(y_val11, pred1))
print('F1 score test:\n', classification_report(y_test_com11, pred2))
svm_p = SVC(probability = True) # for probability
clf_p = svm_p.fit(X_sm11, y_sm11.ravel()) # for probability
pred_proba = svm_p.predict_proba(X_test_com11)[::,1]
fpr, tpr, _ = roc_curve(y_test_com11,  pred_proba)
auc = roc_auc_score(y_test_com11, pred_proba)
plt.plot(fpr,tpr,label="SVM With SMOTE ="+str(np.round(auc,3)))
```

- LET's see the o/p we got –
    1. Let's check Accuracy for validation & Test along with Precision, Recall & F1 Score
    2. ROC/AUC curve.

```
WITH SMOTE:
Validation accuracy:  0.9924928703237712
test accuracy:  0.9994880277252678
********************************************************************************
[[25358     5]
 [    8    21]]
********************************************************************************
F1 score val:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     22097
           1       0.95      0.94      0.95      1747

    accuracy                           0.99     23844
   macro avg       0.97      0.97      0.97     23844
weighted avg       0.99      0.99      0.99     23844

********************************************************************************
F1 score test:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     25363
           1       0.81      0.72      0.76        29

    accuracy                           1.00     25392
   macro avg       0.90      0.86      0.88     25392
weighted avg       1.00      1.00      1.00     25392

********************************************************************************
```



 # Model fitting with imbalance treatment applying SMOTE on train & test-

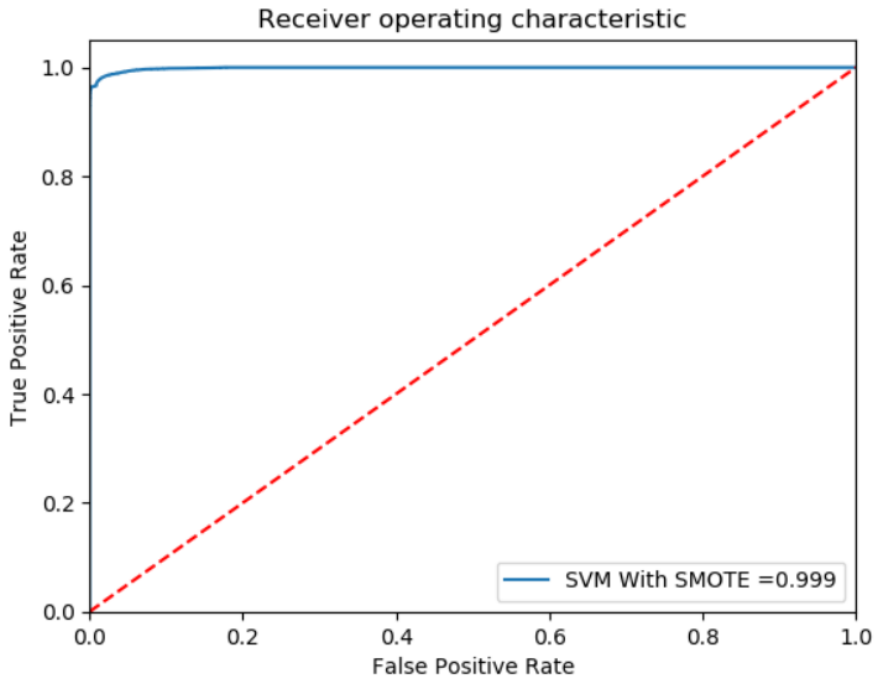
```
svm = SVC() #SVC(kernel='linear')
clf = svm.fit(X_sm11, y_sm11.ravel())
score1 = svm.score(X_val11, y_val11)
score2 = svm.score(X_smt11, y_smt11)
pred1 = svm.predict(X_val11)
pred2 = svm.predict(X_smt11)
# print classification report
print('With SMOTE:'.upper())
print('Validation accuracy: ', score1)
print('test accuracy: ', score2)
```

```
print(confusion_matrix(y_smt11, pred2))
print('F1 score val:\n', classification_report(y_val11, pred1))
print('F1 score test:\n', classification_report(y_smt11, pred2))
svm_p = SVC(probability = True) # for probability
clf_p = svm_p.fit(X_sm11, y_sm11.ravel()) # for probability
pred_proba = svm_p.predict_proba(X_smt11)[::,1]
fpr, tpr, _ = roc_curve(y_smt11,  pred_proba)
auc = roc_auc_score(y_smt11, pred_proba)
plt.plot(fpr,tpr,label="SVM With SMOTE ="+str(np.round(auc,3)))
```

- LET's see the o/p we got –

1. Let's check Accuracy for validation & Test along with Precision, Recall & F1 Score
2. ROC/AUC curve.

```
WITH SMOTE:
Validation accuracy:  0.9924928703237712
test accuracy:  0.9649095138587707
*******************************************************************************
[[25358     5]
 [ 1775 23588]]
*******************************************************************************
F1 score val:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     22097
           1       0.95      0.94      0.95      1747

    accuracy                           0.99     23844
   macro avg       0.97      0.97      0.97     23844
weighted avg       0.99      0.99      0.99     23844

*******************************************************************************
F1 score test:
              precision    recall  f1-score   support

           0       0.93      1.00      0.97     25363
           1       1.00      0.93      0.96     25363

    accuracy                           0.96     50726
   macro avg       0.97      0.96      0.96     50726
weighted avg       0.97      0.96      0.96     50726

*******************************************************************************
```



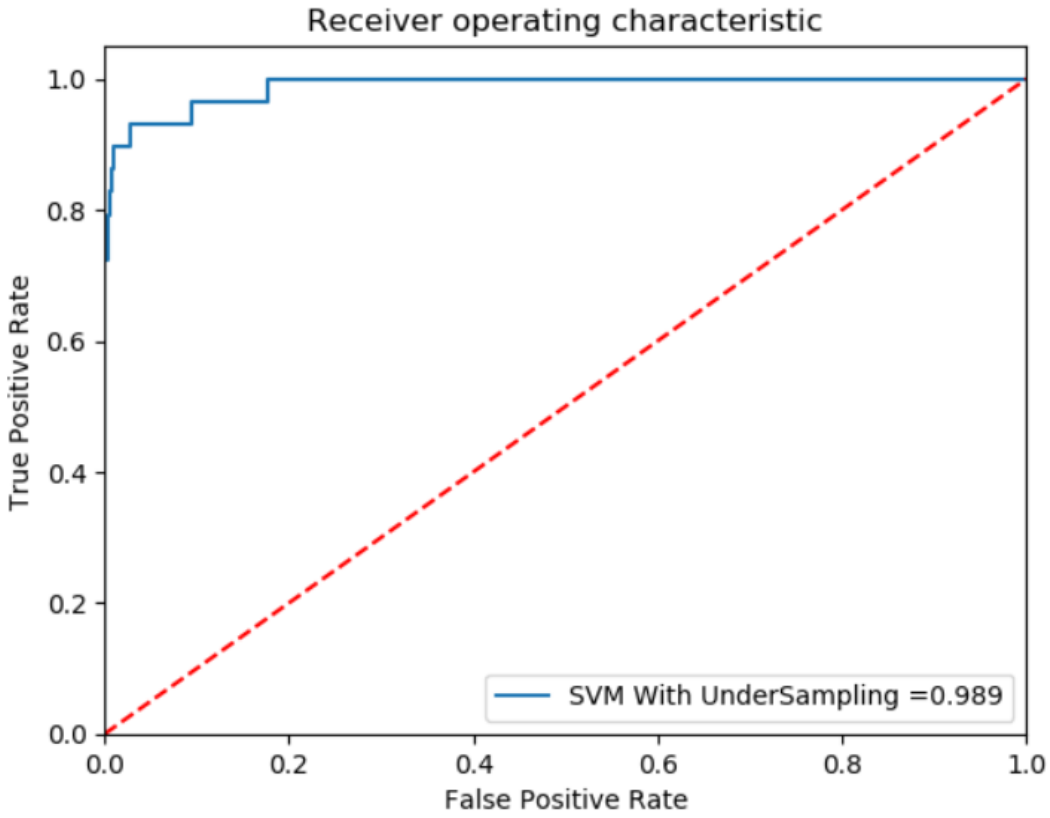
 # Model fitting with imbalance treatment applying Under Sampling on train-

```
svm = SVC() #SVC(kernel='linear')
clf = svm.fit(X_sm12, y_sm12.ravel())
score1 = svm.score(X_val11, y_val11)
score2 = svm.score(X_test_com11, y_test_com11)
pred1 = svm.predict(X_val11)
pred2 = svm.predict(X_test_com11)
# print classification report
print('With UnderSampling:'.upper())
print('Validation accuracy: ', score1)
print('test accuracy: ', score2)
print(confusion_matrix(y_test_com11, pred2))
print('F1 score val:\n', classification_report(y_val11, pred1))
print('F1 score test:\n', classification_report(y_test_com11, pred2))
svm_p = SVC(probability = True) # for probability
clf_p = svm_p.fit(X_sm11, y_sm11.ravel()) # for probability
pred_proba = svm_p.predict_proba(X_test_com11)[::,1]
fpr, tpr, _ = roc_curve(y_test_com11,  pred_proba)
auc = roc_auc_score(y_test_com11, pred_proba)
plt.plot(fpr,tpr,label="SVM With UnderSampling ="+str(np.round(auc,3)))
```

- LET's see the o/p we got –

1. Let's check Accuracy for **validation & Test** along with Precision, Recall & F1 Score
2. ROC/AUC curve.

```
WITH UNDERSAMPLING:
Validation accuracy:  0.9747525582955879
test accuracy:  0.9936594202898551
*******************************************************************************
[[25210   153]
 [    8    21]]
*******************************************************************************
F1 score val:
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     22097
           1       0.75      0.99      0.85      1747

    accuracy                           0.97     23844
   macro avg       0.87      0.98      0.92     23844
weighted avg       0.98      0.97      0.98     23844

*******************************************************************************
F1 score test:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     25363
           1       0.12      0.72      0.21        29

    accuracy                           0.99     25392
   macro avg       0.56      0.86      0.60     25392
weighted avg       1.00      0.99      1.00     25392

*******************************************************************************
```



3.3.5.3 SVM STATISTICS

	Method	Testing on	ACCURACY(%)	PRECISION	RECALL	F1-SCORE	ROC Score
SVM	Without Treatement	Validation	99.12	0.97	0.99	0.94	-
		Test	99.92	0.66	0.66	0.66	0.95
	SMOTE on Train	Validation	99.24	0.95	0.94	0.95	-
		Test	99.95	0.81	0.72	0.76	0.99
	SMOTE on Train & Test	Validation	99.25	0.95	0.94	0.95	-
		Test	96.49	1	0.93	0.96	0.999
	Under Sampling	Validation	97.48	0.75	0.99	0.85	-
		Test	99.37	0.12	0.72	0.21	0.989

CHAPTER 4: MODEL SELECTION & RECOMMENDATION

4.1 Choosing the Best Model

Below is the complete summary of all the models and methods used.
Two fields which we will not consider while declaring the best model would be field marked in Green & blue i.e. SMOTE on both test & Train and Under Sampling.

- Reason for not choosing SMOTE on both test & train is because of the fabricated result it gives on all ML algorithm. Reason why we decided to apply SMOTE test as well because of huge imbalance in target variable. Although it completely depends upon the customer/ XYZ bank if it wants to take into consideration.
- We used another imbalance treatment apart from SMOTE and that was Under Sampling the majority class. However, we observe almost for all algorithm it does not give a very good result. Infact other methods performed better than this.
- Under decision tree algorithm We used Grid search with Cross Validation however it did not return a very good F1 Score. Hence, we would not be considering this while selecting best model. We tried Grid search with CV for RF also, but it caused Memory issues as data frame was extremely large.

MODELS	METHODS	TESTING	ACCURACY(%)	PRECISION	RECALL	F1-SCORE	ROC Score
LR	Without Treatement	Test	99.96	0.92	0.7	0.79	0.929
	SMOTE on Train	Test	99.95	0.87	0.69	0.77	0.926
	SMOTE on Train & Test	Test	97.88	1	0.96	0.98	1
	Under Sampling	Test	99.78	0.32	0.79	0.45	0.965
DT	Without Treatement	Test	99.08	0.11	0.96	0.2	0.977
	SMOTE on Train	Test	99.3	0.14	0.95	0.24	0.971
	SMOTE on Train & Test	Test	99.04	0.99	0.99	0.99	0.99
	Under Sampling	Test	98.99	0.1	0.97	0.18	0.978
	Cross Validation	Test	99.3				0.978
	CV with Grid search	Test	99.31	0.14	0.97	0.25	0.98
RF	Without Treatement	Test	98.92	0.09	0.97	0.17	0.996
	SMOTE on Train	Test	99.31	0.78	0.92	0.84	0.996
	SMOTE on Train & Test	Test	99.04	1	0.99	1	1
	Under Sampling	Test	98.99	0.59	0.97	0.73	0.998
XGB	Without Treatement	Test	99.99	0.98	0.97	0.97	0.999
	SMOTE on Train	Test	99.98	0.89	0.95	0.92	0.998
	SMOTE on Train & Test	Test	99.28	1	0.99	0.99	1
	Under Sampling	Test	99.97	0.82	0.97	0.88	0.997
SVM	Without Treatement	Test	99.92	0.66	0.66	0.66	0.95
	SMOTE on Train	Test	99.95	0.81	0.72	0.76	0.99
	SMOTE on Train & Test	Test	96.49	1	0.93	0.96	0.999
	Under Sampling	Test	99.37	0.12	0.72	0.21	0.989

4.1.1 Main focus

Now we will focus on only those methods which performed best along with permissibility. The two methods under consideration are **Without any treatment** & **SMOTE** on Train data.
The table below shows combination of only those 2 methods.
Need for SMOTE arise because of huge data imbalance and only this method was preferred because it prevents overfitting.

MODELS	METHODS	TESTING	ACCURACY(%)	PRECISION	RECALL	F1-SCORE	ROC Score
LR	Without Treatement	Test	99.96	0.92	0.7	0.79	0.929
LR	SMOTE on Train	Test	99.95	0.87	0.69	0.77	0.926
DT	Without Treatement	Test	99.08	0.11	0.96	0.2	0.977
DT	SMOTE on Train	Test	99.3	0.14	0.95	0.24	0.971
RF	Without Treatement	Test	98.92	0.09	0.97	0.17	0.996
RF	SMOTE on Train	Test	99.31	0.78	0.92	0.84	0.996
XGB	Without Treatement	Test	99.99	0.98	0.97	0.97	0.999
XGB	SMOTE on Train	Test	99.98	0.89	0.95	0.92	0.998
SVM	Without Treatement	Test	99.92	0.66	0.66	0.66	0.95
SVM	SMOTE on Train	Test	99.95	0.81	0.72	0.76	0.99

From above table we arrange rows w.r.t F1 Score column. We understand that accuracy is the one on which focus should be there but given how data is unbalanced we will choose F1 score as the primary criteria.
1 more reason for selecting F! score is because it depends on Precision & Recall both.
Apart from F1 score of 0.97 XG boost also possess an excellent Accuracy of 99.99% & ROC/AUR score of 99.99%

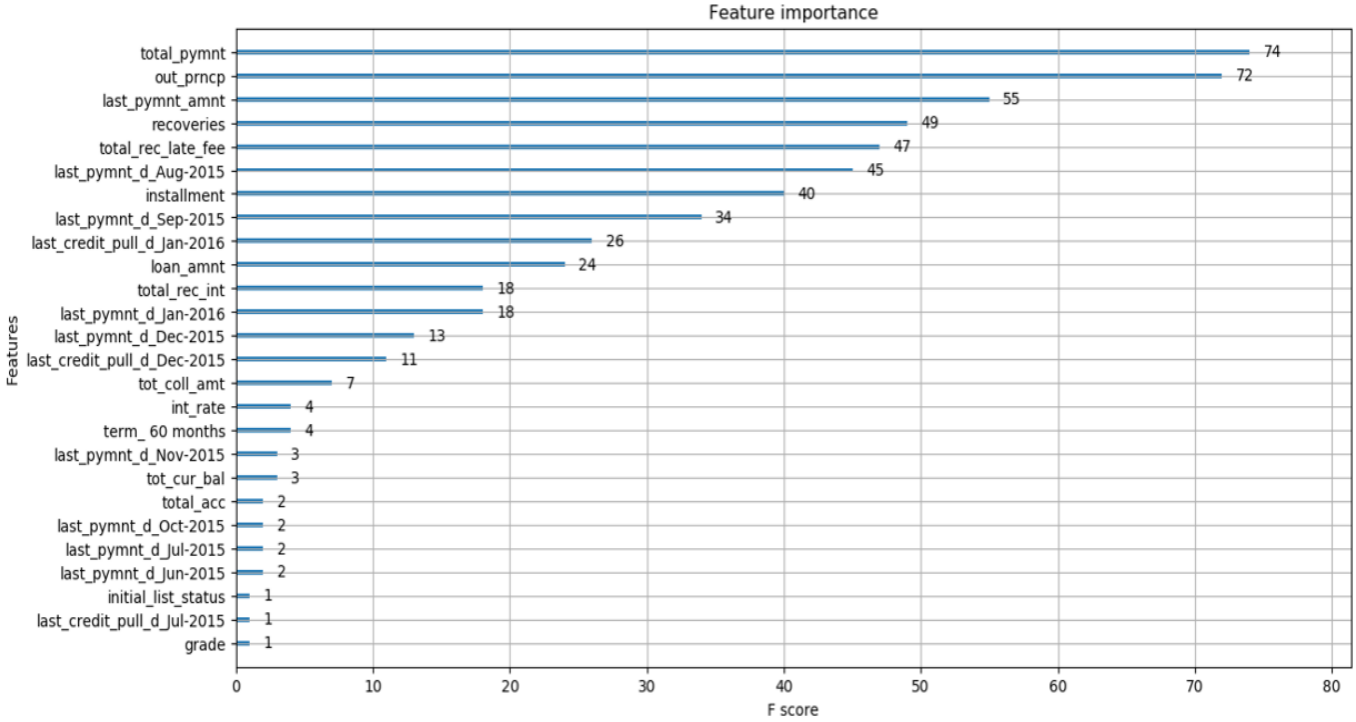
XG Boost stands out from all other algorithms. Without imbalance treatment as well as SMOTE stands at top, hence we will choose XGB as the best model.

MODELS	METHODS	TESTING	ACCURACY(%)	PRECISION	RECALL	F1-SCORE	ROC Score
XGB	Without Treatement	Test	99.99	0.98	0.97	**0.97**	0.999
XGB	SMOTE on Train	Test	99.98	0.89	0.95	**0.92**	0.998
RF	SMOTE on Train	Test	99.31	0.78	0.92	**0.84**	0.996
LR	Without Treatement	Test	99.96	0.92	0.7	**0.79**	0.929
LR	SMOTE on Train	Test	99.95	0.87	0.69	**0.77**	0.926
SVM	SMOTE on Train	Test	99.95	0.81	0.72	**0.76**	0.99
SVM	Without Treatement	Test	99.92	0.66	0.66	**0.66**	0.95
DT	SMOTE on Train	Test	99.3	0.14	0.95	**0.24**	0.971
DT	Without Treatement	Test	99.08	0.11	0.96	**0.2**	0.977
RF	Without Treatement	Test	98.92	0.09	0.97	**0.17**	0.996

4.1.2 Parameters of XG Boost & feature importance:

XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=3, min_child_weight=1, missing=None, n_estimators=100, n_jobs=1, nthread=None, objective='binary:logistic', random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None, silent=None, subsample=1, verbosity=1)

Apart from this XG Boost also gives us list of important features:



4.1.3 Advantages of using XG Boost –

1. **Regularization**: - Standard GBM implementation has no regularization like XGBoost, therefore it also helps to reduce overfitting.
2. **Parallel Processing**: - XGBoost implements parallel processing and is blazingly faster as compared to GBM.
3. **High Flexibility**: -XGBoost allows users to define custom optimization objectives and evaluation criteria.
4. **Handling Missing Values**: - The user is required to supply a different value than other observations and pass that as a parameter. XGBoost tries different things as it encounters a missing value on each node and learns which path to take for missing values in future.
5. **Tree Pruning**: - It splits upto the max_depth specified and then start pruning the tree backwards and remove splits beyond which there is no positive gain.
6. **Built-in Cross-Validation**: - XGBoost allows user to run a cross-validation at each iteration of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run.

4.2 Major Findings from project

We tried 4 combinations of numerical & categorical data to fit the model and below are our observations:

- Label encoded categorical along with cut/scaled numeric data performed really bad for almost all algorithm hence we decided not to include that in our final model build.
- Cut numerical along with dummy encoded categorical also fell short on all algorithm, it did not have a good F1 score for almost all algorithm hence we have not included that combination either.
- Scaled numeric & dummy encoded categorical performed best by far, hence that combination was considered for model fitting.

4.2.1 Best Models:

- XG boost with / without imbalance treatment was by far the best model and it would be our 1st choice considering the least processing time it took and best result it delivered. By playing around with its hyper parameters we can surely improve the results further.
- Random forest with SMOTE can be our next best choice, again processing time was fast but slower than XGB.
- Logistic regression with/without treatment could be our 3rd choice which gave a fairly good result.

4.2.2 Key Recommendations

- Fields like Zip code & policy code barely are of any significance so such fields can be ignored from our study.
- Based on annual income & purpose, lending organization can decide how much loan to finance. For e.g.- if a customer has a higher salary or source of income is fixed, he can be financed loan.
- Columns depicting delinquency can be kept in binarized format by bank. So if someone has no default history it can be taken as zero and if someone has defaulted in 1 or more than 1 month can be simply labelled as 1 i.e. delinquency.
- A higher interest rate can be charged from those who have defaulted as a form of penalty, so they don't default from next time.
- Successful payers should be given priority in terms loan grant.
- Finally, XGB model gave a list of important features for this entire process that can be considered by bank while issuing loan.