In [1]:
```
# Assignment: DSC630 Term Project
# Name: Bezawada, Sashidhar
# Date: 2023-08-09
# Assignment: Milestone 4
```

1. Importing data & libraries

I...
```
# Importing Libraries

import numpy as np
import pandas as pd
from numpy import mean
from numpy import std

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
plt.style.use('ggplot')
from plotly import tools
import plotly.offline as py
import plotly.figure_factory as ff
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.express as px
from plotly.subplots import make_subplots


from scipy.stats import norm
from sklearn.preprocessing import StandardScaler
from scipy import stats

import warnings
warnings.filterwarnings('ignore')

#for displaying 500 results in pandas dataframe
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

import itertools
#import xgboost as xgb
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import precision_score, recall_score, confusion_matrix, class
import warnings
```

In [3]:
```
test = pd.read_csv('datasets/test.csv')
train = pd.read_csv('datasets/train.csv')
```

In [4]:
```
df= pd.concat ([train, test])
```

```
 Shape of training dataframe:  (233154, 41)
 Shape of testing dataframe:  (112392, 40)
(233154, 41)
(112392, 40)
```

2. Variable Inspection

In [6]:
```
print("Names of columns ", list(train.columns))
```

```
Names of columns  ['UNIQUEID', 'DISBURSED_AMOUNT', 'ASSET_COST', 'LTV', 'BRANCH_ID',
'SUPPLIER_ID', 'MANUFACTURER_ID', 'CURRENT_PINCODE_ID', 'DATE_OF_BIRTH', 'EMPLOYMENT_
TYPE', 'DISBURSAL_DATE', 'STATE_ID', 'EMPLOYEE_CODE_ID', 'MOBILENO_AVL_FLAG', 'AADHAR
_FLAG', 'PAN_FLAG', 'VOTERID_FLAG', 'DRIVING_FLAG', 'PASSPORT_FLAG', 'PERFORM_CNS_SCO
RE', 'PERFORM_CNS_SCORE_DESCRIPTION', 'PRI_NO_OF_ACCTS', 'PRI_ACTIVE_ACCTS', 'PRI_OVE
RDUE_ACCTS', 'PRI_CURRENT_BALANCE', 'PRI_SANCTIONED_AMOUNT', 'PRI_DISBURSED_AMOUNT',
'SEC_NO_OF_ACCTS', 'SEC_ACTIVE_ACCTS', 'SEC_OVERDUE_ACCTS', 'SEC_CURRENT_BALANCE', 'S
EC_SANCTIONED_AMOUNT', 'SEC_DISBURSED_AMOUNT', 'PRIMARY_INSTAL_AMT', 'SEC_INSTAL_AM
T', 'NEW_ACCTS_IN_LAST_SIX_MONTHS', 'DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS', 'AVERAGE_A
CCT_AGE', 'CREDIT_HISTORY_LENGTH', 'NO_OF_INQUIRIES', 'LOAN_DEFAULT']
```

In ...
```
#Null values in training dataset

null= train.isnull().sum().sort_values(ascending=False)
total =train.shape[0]
percent_missing= (train.isnull().sum()/total).sort_values(ascending=False)

missing_data= pd.concat([null, percent_missing], axis=1, keys=['Total missing', '

missing_data.reset_index(inplace=True)
missing_data= missing_data.rename(columns= { "index": " column name"})

print ("Null Values in each column:\n", missing_data.sort_values(by ='Total missi
```

```
Null Values in each column:
                            column name  Total missing  Percent missing
0                       EMPLOYMENT_TYPE           7661         0.032858
21          PERFORM_CNS_SCORE_DESCRIPTION            0         0.000000
23                         DISBURSAL_DATE            0         0.000000
24                             ASSET_COST            0         0.000000
25                                    LTV            0         0.000000
26                              BRANCH_ID            0         0.000000
27                            SUPPLIER_ID            0         0.000000
28                        MANUFACTURER_ID            0         0.000000
29                     CURRENT_PINCODE_ID            0         0.000000
30                          DATE_OF_BIRTH            0         0.000000
31                               STATE_ID            0         0.000000
32                      PERFORM_CNS_SCORE            0         0.000000
33                       EMPLOYEE_CODE_ID            0         0.000000
34                      MOBILENO_AVL_FLAG            0         0.000000
35                            AADHAR_FLAG            0         0.000000
36                               PAN_FLAG            0         0.000000
37                           VOTERID_FLAG            0         0.000000
38                           DRIVING_FLAG            0         0.000000
39                          PASSPORT_FLAG            0         0.000000
22                       DISBURSED_AMOUNT            0         0.000000
20                       PRI_ACTIVE_ACCTS            0         0.000000
1                                UNIQUEID            0         0.000000
19                        NO_OF_INQUIRIES            0         0.000000
2                    SEC_SANCTIONED_AMOUNT            0         0.000000
3                       PRI_OVERDUE_ACCTS            0         0.000000
4                     PRI_CURRENT_BALANCE            0         0.000000
5                   PRI_SANCTIONED_AMOUNT            0         0.000000
6                    PRI_DISBURSED_AMOUNT            0         0.000000
7                         SEC_NO_OF_ACCTS            0         0.000000
8                        SEC_ACTIVE_ACCTS            0         0.000000
9                       SEC_OVERDUE_ACCTS            0         0.000000
10                    SEC_CURRENT_BALANCE            0         0.000000
11                   SEC_DISBURSED_AMOUNT            0         0.000000
12                        PRI_NO_OF_ACCTS            0         0.000000
13                     PRIMARY_INSTAL_AMT            0         0.000000
14                         SEC_INSTAL_AMT            0         0.000000
15          NEW_ACCTS_IN_LAST_SIX_MONTHS            0         0.000000
16   DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS            0         0.000000
17                       AVERAGE_ACCT_AGE            0         0.000000
18                  CREDIT_HISTORY_LENGTH            0         0.000000
40                           LOAN_DEFAULT            0         0.000000
```

In …

```
#Null values in test dataset

null= test.isnull().sum().sort_values(ascending=False)
total =test.shape[0]
percent_missing= (test.isnull().sum()/total).sort_values(ascending=False)

missing_data= pd.concat([null, percent_missing], axis=1, keys=['Total missing', '

missing_data.reset_index(inplace=True)
```

```
Null Values in each column:
                          column name  Total missing  Percent missing
0                     EMPLOYMENT_TYPE           3443         0.030634
1                            UNIQUEID              0         0.000000
22                     DISBURSAL_DATE              0         0.000000
23                         ASSET_COST              0         0.000000
24                                LTV              0         0.000000
25                          BRANCH_ID              0         0.000000
26                        SUPPLIER_ID              0         0.000000
27                    MANUFACTURER_ID              0         0.000000
28                 CURRENT_PINCODE_ID              0         0.000000
29                      DATE_OF_BIRTH              0         0.000000
30                           STATE_ID              0         0.000000
31                  PERFORM_CNS_SCORE              0         0.000000
32                   EMPLOYEE_CODE_ID              0         0.000000
33                  MOBILENO_AVL_FLAG              0         0.000000
34                        AADHAR_FLAG              0         0.000000
35                           PAN_FLAG              0         0.000000
36                       VOTERID_FLAG              0         0.000000
37                       DRIVING_FLAG              0         0.000000
38                      PASSPORT_FLAG              0         0.000000
21                   DISBURSED_AMOUNT              0         0.000000
20      PERFORM_CNS_SCORE_DESCRIPTION              0         0.000000
19                   PRI_ACTIVE_ACCTS              0         0.000000
9                   SEC_OVERDUE_ACCTS              0         0.000000
2                 SEC_CURRENT_BALANCE              0         0.000000
3                   PRI_OVERDUE_ACCTS              0         0.000000
4                 PRI_CURRENT_BALANCE              0         0.000000
5               PRI_SANCTIONED_AMOUNT              0         0.000000
6                PRI_DISBURSED_AMOUNT              0         0.000000
7                     SEC_NO_OF_ACCTS              0         0.000000
8                    SEC_ACTIVE_ACCTS              0         0.000000
10              SEC_SANCTIONED_AMOUNT              0         0.000000
18              CREDIT_HISTORY_LENGTH              0         0.000000
11                    PRI_NO_OF_ACCTS              0         0.000000
12               SEC_DISBURSED_AMOUNT              0         0.000000
13                 PRIMARY_INSTAL_AMT              0         0.000000
14                     SEC_INSTAL_AMT              0         0.000000
15         NEW_ACCTS_IN_LAST_SIX_MONTHS              0         0.000000
16  DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS              0         0.000000
17                   AVERAGE_ACCT_AGE              0         0.000000
39                    NO_OF_INQUIRIES              0         0.000000
```

Flag 1: 3443 missing values in employment type

In [9]:

```
train_null_unique= train.EMPLOYMENT_TYPE .unique()
test_null_unique= test.EMPLOYMENT_TYPE .unique()
print(train_null_unique)
print (test_null_unique)
```

```
['Salaried' 'Self employed' nan]
['Salaried' 'Self employed' nan]
```

In [10]:

```
train.EMPLOYMENT_TYPE= train.EMPLOYMENT_TYPE.fillna("Missing")
test.EMPLOYMENT_TYPE= test.EMPLOYMENT_TYPE .fillna("Missing")
```

```
['Salaried' 'Self employed' 'Missing']
['Salaried' 'Self employed' 'Missing']
```

In [11]:
```
print(train.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 233154 entries, 0 to 233153
Data columns (total 41 columns):
 #   Column                               Non-Null Count   Dtype  
---  ------                               --------------   -----  
 0   UNIQUEID                             233154 non-null  int64  
 1   DISBURSED_AMOUNT                     233154 non-null  int64  
 2   ASSET_COST                           233154 non-null  int64  
 3   LTV                                  233154 non-null  float64
 4   BRANCH_ID                            233154 non-null  int64  
 5   SUPPLIER_ID                          233154 non-null  int64  
 6   MANUFACTURER_ID                      233154 non-null  int64  
 7   CURRENT_PINCODE_ID                   233154 non-null  int64  
 8   DATE_OF_BIRTH                        233154 non-null  object 
 9   EMPLOYMENT_TYPE                      233154 non-null  object 
 10  DISBURSAL_DATE                       233154 non-null  object 
 11  STATE_ID                             233154 non-null  int64  
 12  EMPLOYEE_CODE_ID                     233154 non-null  int64  
 13  MOBILENO_AVL_FLAG                    233154 non-null  int64  
 14  AADHAR_FLAG                          233154 non-null  int64  
 15  PAN_FLAG                             233154 non-null  int64  
 16  VOTERID_FLAG                         233154 non-null  int64  
 17  DRIVING_FLAG                         233154 non-null  int64  
 18  PASSPORT_FLAG                        233154 non-null  int64  
 19  PERFORM_CNS_SCORE                    233154 non-null  int64  
 20  PERFORM_CNS_SCORE_DESCRIPTION        233154 non-null  object 
 21  PRI_NO_OF_ACCTS                      233154 non-null  int64  
 22  PRI_ACTIVE_ACCTS                     233154 non-null  int64  
 23  PRI_OVERDUE_ACCTS                    233154 non-null  int64  
 24  PRI_CURRENT_BALANCE                  233154 non-null  int64  
 25  PRI_SANCTIONED_AMOUNT                233154 non-null  int64  
 26  PRI_DISBURSED_AMOUNT                 233154 non-null  int64  
 27  SEC_NO_OF_ACCTS                      233154 non-null  int64  
 28  SEC_ACTIVE_ACCTS                     233154 non-null  int64  
 29  SEC_OVERDUE_ACCTS                    233154 non-null  int64  
 30  SEC_CURRENT_BALANCE                  233154 non-null  int64  
 31  SEC_SANCTIONED_AMOUNT                233154 non-null  int64  
 32  SEC_DISBURSED_AMOUNT                 233154 non-null  int64  
 33  PRIMARY_INSTAL_AMT                   233154 non-null  int64  
 34  SEC_INSTAL_AMT                       233154 non-null  int64  
 35  NEW_ACCTS_IN_LAST_SIX_MONTHS         233154 non-null  int64  
 36  DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS  233154 non-null  int64  
 37  AVERAGE_ACCT_AGE                     233154 non-null  object 
 38  CREDIT_HISTORY_LENGTH                233154 non-null  object 
 39  NO_OF_INQUIRIES                      233154 non-null  int64  
 40  LOAN_DEFAULT                         233154 non-null  int64  
dtypes: float64(1), int64(34), object(6)
memory usage: 72.9+ MB
None
```

Flag 2:
AVERAGE_ACCT_AGE, CREDIT_HISTORY_LENGTH are object, but they should be int.
DATE_OF_BIRTH & DISBURSAL_DATE should be datetime type

In [1...
```
train['DATE_OF_BIRTH'] =  pd.to_datetime(train['DATE_OF_BIRTH'], format='%d-%m-%
#format='%d%b%Y:%H:%M:%S.%f'
test['DATE_OF_BIRTH'] =  pd.to_datetime(test['DATE_OF_BIRTH'], format='%d-%m-%Y
train['DISBURSAL_DATE'] =  pd.to_datetime(train['DISBURSAL_DATE'], format='%d-%m
test['DISBURSAL_DATE'] =  pd.to_datetime(test['DISBURSAL_DATE'], format='%d-%m-%
```

EDA

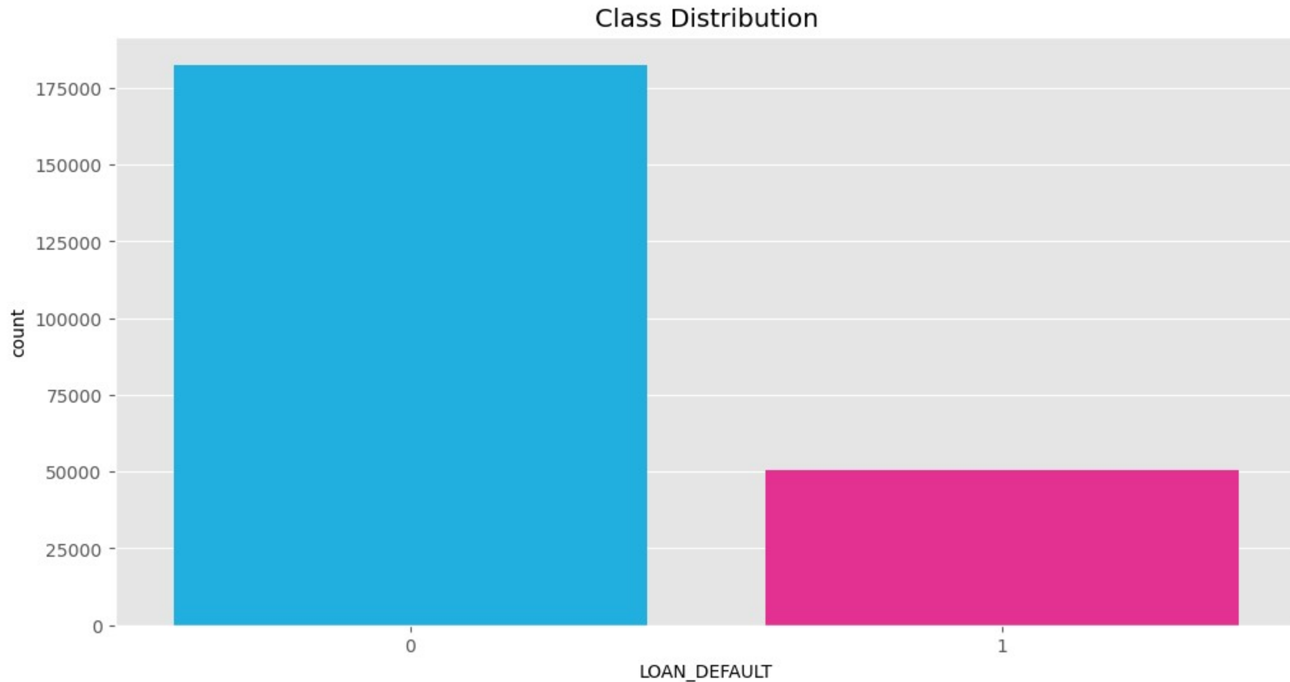
3.1 Class Distribution

In ...
```
class_df = train.groupby('LOAN_DEFAULT').count()['UNIQUEID'].reset_index().sort_v
class_df.style.background_gradient(cmap='winter')
```

Out[14]:

	LOAN_DEFAULT	UNIQUEID
0	0	182543
1	1	50611

In ...
```
#Graph
my_pal = {0: 'deepskyblue', 1: 'deeppink'}

plt.figure(figsize = (12, 6))
ax = sns.countplot(x = 'LOAN_DEFAULT', data = train, palette = my_pal)
plt.title('Class Distribution')
plt.show()

# Count and %
Count_Normal_transacation = len(train[train['LOAN_DEFAULT']==0])
Count_Fraud_transacation = len(train[train['LOAN_DEFAULT']==1])
Percentage_of_Normal_transacation = Count_Normal_transacation/(Count_Normal_trans
print('% of no defaults       :', Percentage_of_Normal_transacation*100)
print('Number of no defaults     :', Count_Normal_transacation)
Percentage_of_Fraud_transacation= Count_Fraud_transacation/(Count_Normal_transaca
print('% of defaults         :',Percentage_of_Fraud_transacation*100)
print('Number of defaults    :', Count_Fraud_transacation)
```




```
% of no defaults       : 78.29288796246257
Number of no defaults     : 182543
% of defaults          : 21.70711203753742
Number of defaults     : 50611
```

Flag : Uneven class

In [1...

```
print("Employment type\n")
print(train.groupby(["EMPLOYMENT_TYPE"]).LOAN_DEFAULT.value_counts(normalize=Tru
print("##############\n")
print("Mobile Flag\n")
print(train.groupby(["MOBILENO_AVL_FLAG"]).LOAN_DEFAULT.value_counts(normalize=T
print("##############\n")
print("Aadhar Flag\n")
print(train.groupby(["AADHAR_FLAG"]).LOAN_DEFAULT.value_counts(normalize=True))
print("##############\n")
print("Pan Flag\n")
print(train.groupby(["PAN_FLAG"]).LOAN_DEFAULT.value_counts(normalize=True))
print("##############\n")
print("Voter ID Flag\n")
print(train.groupby(["VOTERID_FLAG"]).LOAN_DEFAULT.value_counts(normalize=True)
print("##############\n")
print("Driving L Flag\n")
print(train.groupby(["DRIVING_FLAG"]).LOAN_DEFAULT.value_counts(normalize=True)
print("##############\n")
print("Passport\n")
print(train.groupby(["PASSPORT_FLAG"]).LOAN_DEFAULT.value_counts(normalize=True
```

```
Employment type

EMPLOYMENT_TYPE  LOAN_DEFAULT
Missing          0               0.785407
                 1               0.214593
Salaried         0               0.796542
                 1               0.203458
Self employed    0               0.772343
                 1               0.227657
Name: LOAN_DEFAULT, dtype: float64
##############

Mobile Flag

MOBILENO_AVL_FLAG  LOAN_DEFAULT
1                  0               0.782929
                   1               0.217071
Name: LOAN_DEFAULT, dtype: float64
##############

Aadhar Flag

AADHAR_FLAG  LOAN_DEFAULT
0            0               0.743594
             1               0.256406
1            0               0.790403
             1               0.209597
Name: LOAN_DEFAULT, dtype: float64
##############

Pan Flag

PAN_FLAG  LOAN_DEFAULT
0         0               0.783170
          1               0.216830
1         0               0.779978
          1               0.220022
Name: LOAN_DEFAULT, dtype: float64
##############

Voter ID Flag

VOTERID_FLAG  LOAN_DEFAULT
0             0               0.790354
              1               0.209646
1             0               0.739125
              1               0.260875
Name: LOAN_DEFAULT, dtype: float64
##############

Driving L Flag
```

```
DRIVING_FLAG  LOAN_DEFAULT
0             0               0.782559
              1               0.217441
1             0               0.798487
              1               0.201513
Name: LOAN_DEFAULT, dtype: float64
##############

Passport

PASSPORT_FLAG  LOAN_DEFAULT
0              0               0.782784
               1               0.217216
1              0               0.850806
               1               0.149194
Name: LOAN_DEFAULT, dtype: float64
```

In…
```
print(train.groupby(["LOAN_DEFAULT","EMPLOYMENT_TYPE","AADHAR_FLAG","PAN_FLAG","DF
print("##############\n")
```

LOAN_DEFAULT	EMPLOYMENT_TYPE	AADHAR_FLAG	PAN_FLAG	DRIVING_FLAG	PASSPORT_FLAG	VOTERID_FLAG	
0	Missing	0	0	0	0	1	451
					1	0	10
				1	0	0	44
						1	4
			1	0	0	1	16
						0	1
				1	0	0	1
		1	0	0	0	0	5315
						1	41
					1	0	1
				1	0	0	15
						1	1
			1	0	0	0	114
						1	1
				1	0	0	2
	Salaried	0	0	0	0	1	6091
					1	0	148
						1	2
				1	0	0	1419
						1	37
					1	0	2
			1	0	0	1	1540
						0	293
					1	0	9
						1	1

				1	0	0
99						
						1
2						
		1	0	0	0	0
63068						
						1
696						
					1	0
43						
				1	0	0
287						
						1
2						
					1	0
1						
			1	0	0	0
4082						
						1
105						
					1	0
4						
				1	0	0
16						
						1
1						
	Self employed	0	0	0	0	1
12203						
					1	0
156						
						1
2						
				1	0	0
1729						
						1
100						
					1	0
1						
			1	0	0	1
2719						
						0
463						
					1	0
16						
				1	0	0
118						
						1
7						
		1	0	0	0	0
75644						
						1
853						

					1	0
20						
				1	0	0
407						
						1
6						
			1	0	0	0
4007						
						1
95						
					1	0
6						
				1	0	0
24						
						1
2						
1	Missing	0	0	0	0	1
93						
					1	0
3						
				1	0	0
9						
			1	0	0	1
3						
						0
1						
				1	0	0
1						
		1	0	0	0	0
1494						
						1
8						
				1	0	0
1						
			1	0	0	0
31						
	Salaried	0	0	0	0	1
1796						
					1	0
18						
				1	0	0
321						
						1
10						
			1	0	0	1
401						
						0
67						
				1	0	0
19						
						1
1						

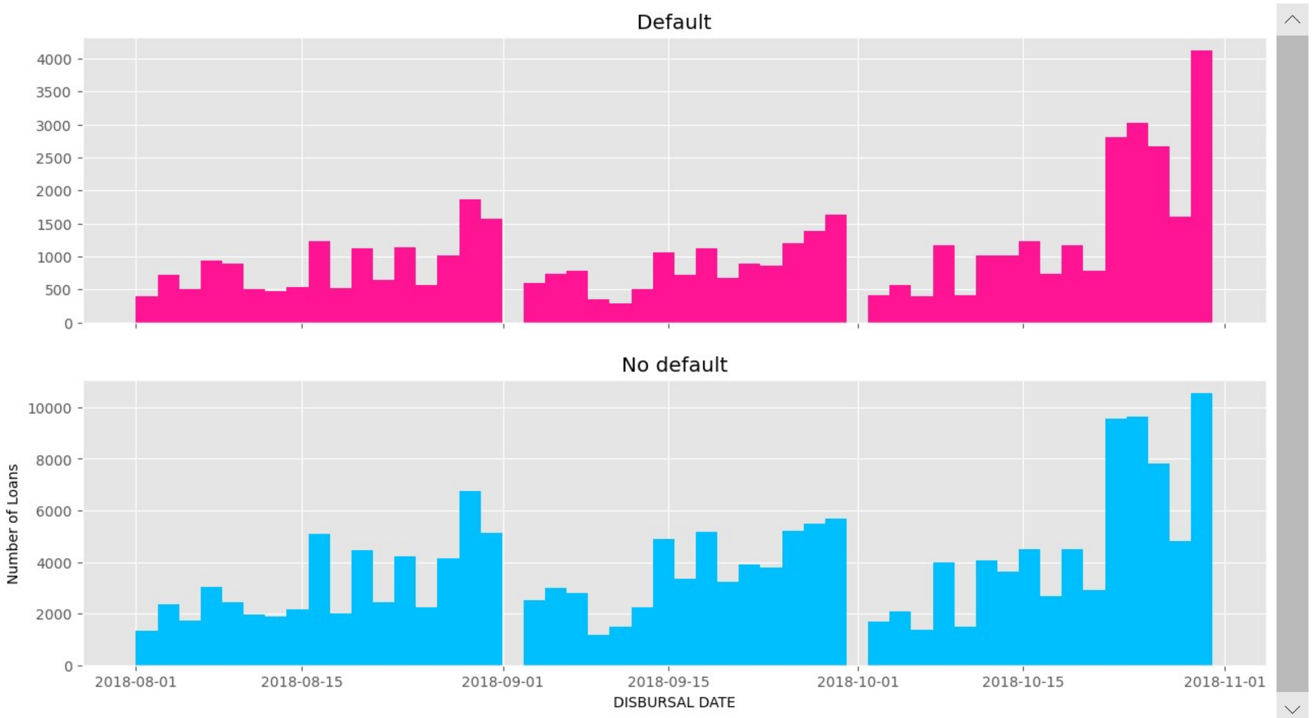
16203		1	0	0	0	0
169						1
6					1	0
56				1	0	0
823			1	0	0	0
16						1
1					1	0
1						1
2				1	0	0
4617	Self employed	0	0	0	0	1
34					1	0
1						1
529				1	0	0
19						1
1416			1	0	0	1
148						0
4					1	0
35				1	0	0
21021		1	0	0	0	0
238						1
6					1	0
79				1	0	0
3						1
876			1	0	0	0
24						1
7				1	0	0

```
Name: VOTERID_FLAG, dtype: int64
##############
```

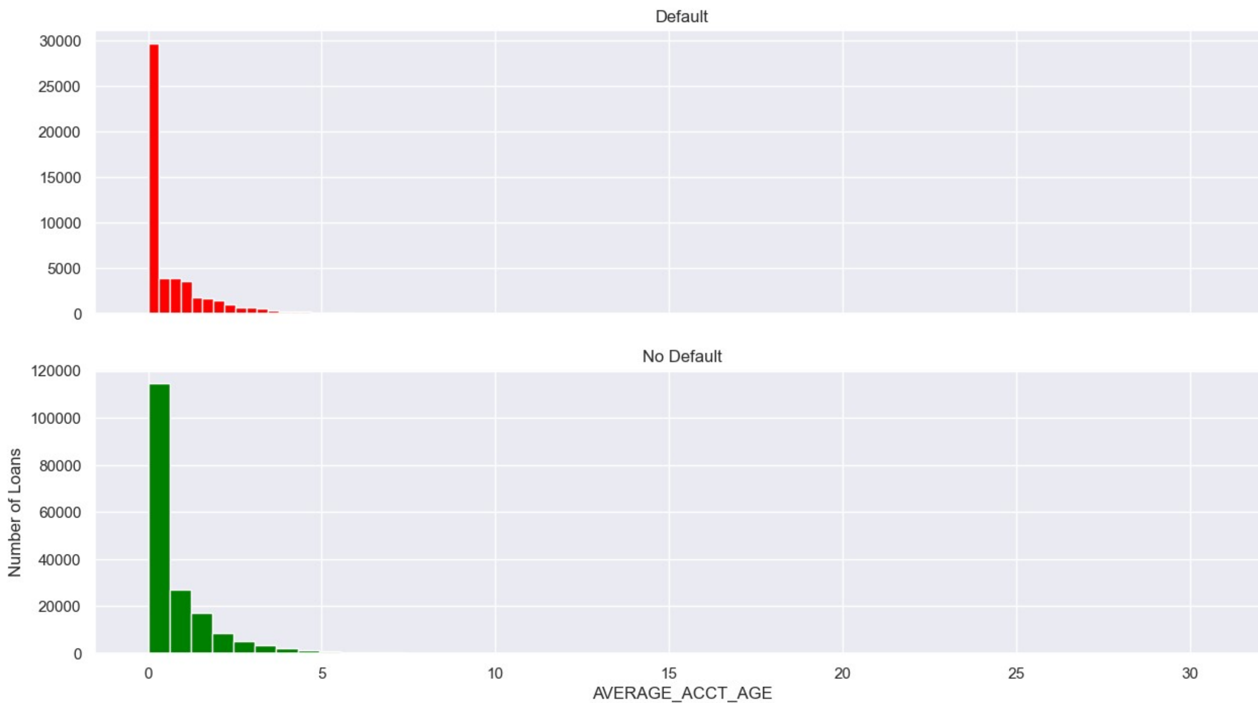
In [18]:
```
train_0 = train[train["LOAN_DEFAULT"]==0]
train_1 = train[train["LOAN_DEFAULT"]==1]
```

3.2 Default vs Disbursal date

In [...
```
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(15,8))

bins = 50

ax1.hist(train.DISBURSAL_DATE[train.LOAN_DEFAULT == 1], bins = bins, color = 'de
ax1.set_title('Default')
ax1.set_title('Default')

ax2.hist(train.DISBURSAL_DATE[train.LOAN_DEFAULT == 0], bins = bins, color = 'de
ax2.set_title('No default')

plt.xlabel('DISBURSAL DATE')
plt.ylabel('Number of Loans')
plt.show()
```

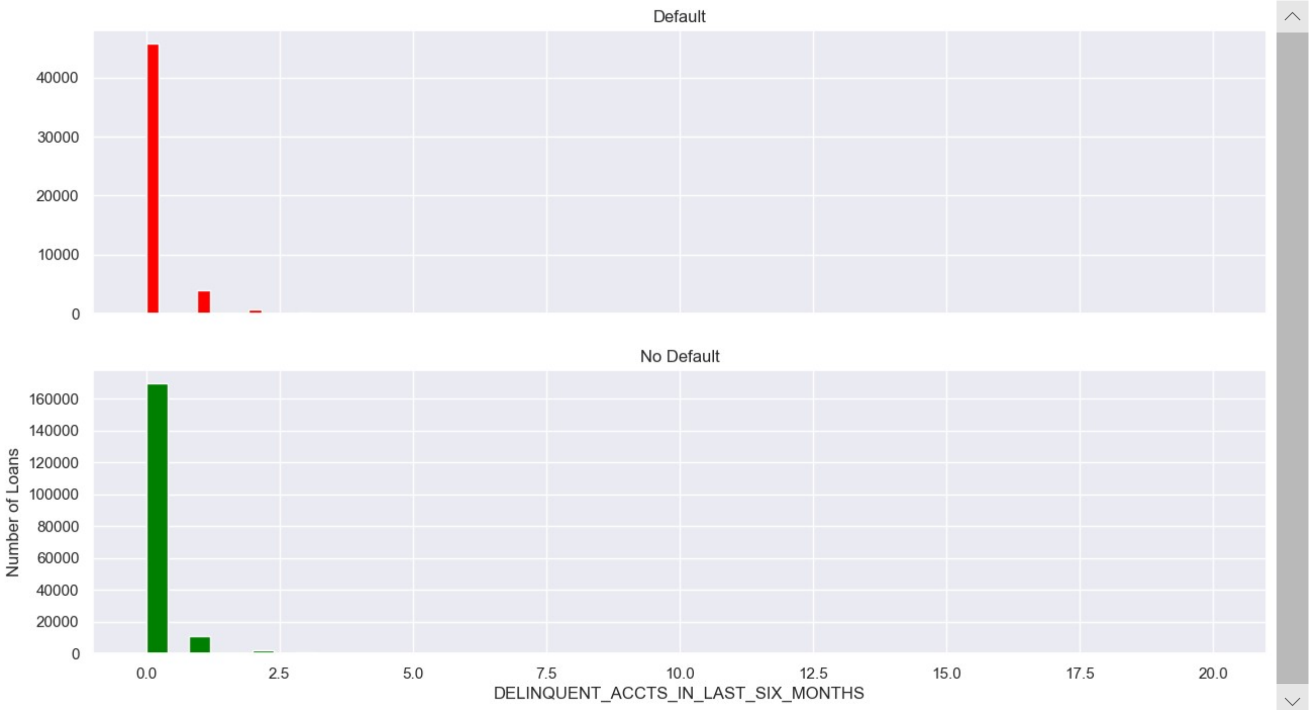


In [1...
```
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(15,8))

bins = 50

ax1.hist(train.AVERAGE_ACCT_AGE[train.LOAN_DEFAULT == 1], bins = bins, color =
ax1.set_title('Default')

ax2.hist(train.AVERAGE_ACCT_AGE[train.LOAN_DEFAULT == 0], bins = bins, color =
ax2.set_title('No Default')
```



In [...

```
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(15,8))

bins = 50

ax1.hist(train.DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS[train.LOAN_DEFAULT == 1], bir
ax1.set_title('Default')

ax2.hist(train.DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS[train.LOAN_DEFAULT == 0], bir
ax2.set_title('No Default')

plt.xlabel('DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS')
plt.ylabel('Number of Loans')
plt.show()
```



3.3 Univariate analysis

In [20]:

```
# Plot distribution of one feature
def plot_distribution(feature,color):
    plt.figure(figsize=(10,6))
    plt.title("Distribution of %s" % feature)
    sns.distplot(train[feature].dropna(),color=color, kde=True,bins=100)
    plt.show()

# Plot distribution of multiple features, with TARGET = 1/0 on the same graph
def plot_distribution_comp(var,nrow=2):

    i = 0
    t1 = train.loc[train['LOAN_DEFAULT'] != 0]
    t0 = train.loc[train['LOAN_DEFAULT'] == 0]

    sns.set_style('whitegrid')
    plt.figure()
    fig, ax = plt.subplots(nrow,2,figsize=(12,6*nrow))

    for feature in var:
        i += 1
        plt.subplot(nrow,2,i)
        sns.kdeplot(t1[feature], bw=0.5,label="LOAN_DEFAULT = 1")
        sns.kdeplot(t0[feature], bw=0.5,label="LOAN_DEFAULT = 0")
        plt.ylabel('Density plot', fontsize=12)
        plt.xlabel(feature, fontsize=12)
        locs, labels = plt.xticks()
        plt.tick_params(axis='both', which='major', labelsize=12)
    plt.show();
```

In [21]:

In [22]:
```
# Box Plot for one feature
def plot_box(feature, color):
    plt.figure(figsize=(10,6))
    plt.title("Box Plot of %s" % feature)
    sns.boxplot(train[feature].dropna(),color=color)
    plt.show()
```

In [23]:
```
# Bar Plot for one feature
def plot_bar(feature):
    plt.figure(figsize=(10,50))
    sns.catplot(y=feature, hue="LOAN_DEFAULT", kind="count",
            palette="pastel", edgecolor=".6",
            data=train);
```

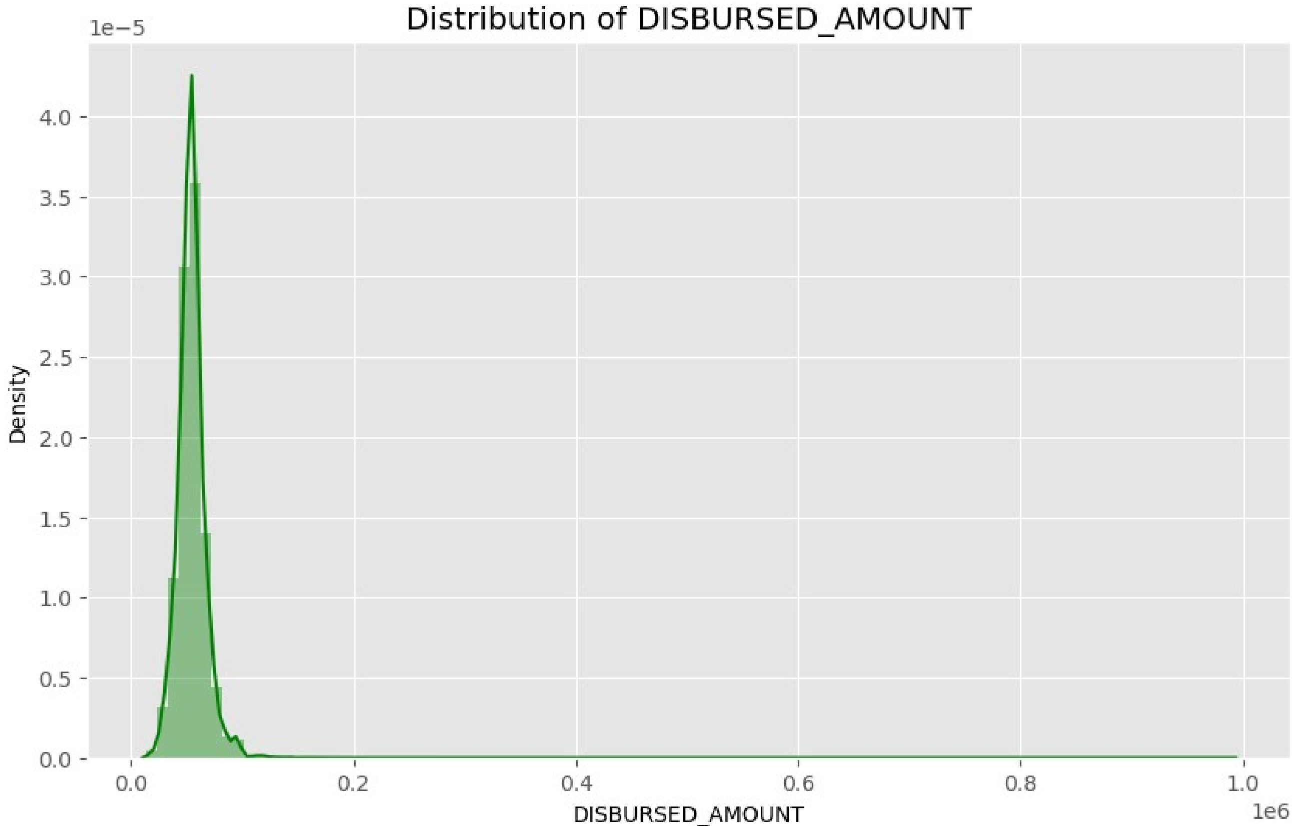
Let's look into variables with high importance

Loan information

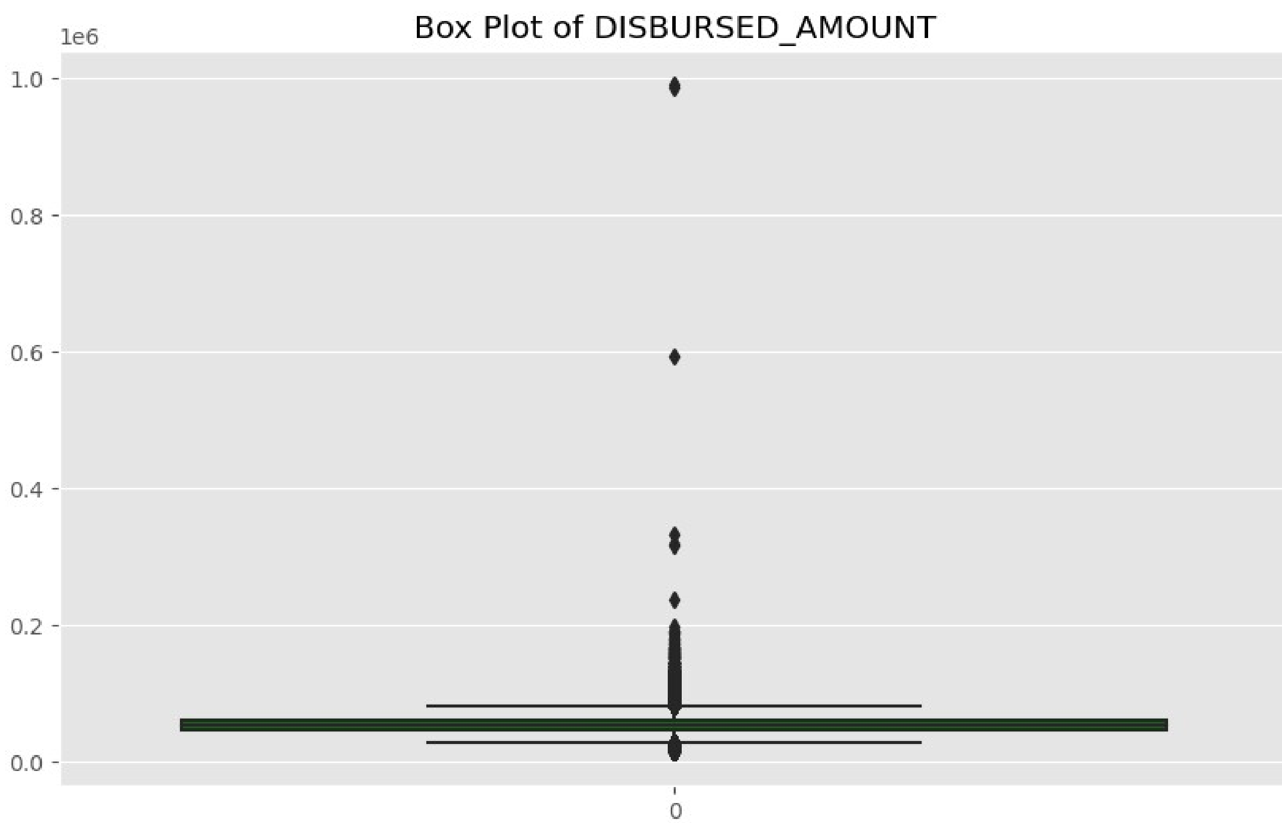
'DISBURSED_AMOUNT' : Amount of Loan disbursed

In [24]:
```
print(train.DISBURSED_AMOUNT.describe())
plot_distribution('DISBURSED_AMOUNT','green')
```

```
count    233154.000000
mean      54356.993528
std       12971.314171
min       13320.000000
25%       47145.000000
50%       53803.000000
75%       60413.000000
max      990572.000000
Name: DISBURSED_AMOUNT, dtype: float64
```



In [25]:
```
plot_box("DISBURSED_AMOUNT", "green")
```



3.4 Outlier Treatment

In [2...

```
#Number of observations in column
obs = len(train.DISBURSED_AMOUNT)
print("No. of observations in column: ",obs)

# calculate summary statistics
data_mean, data_std = mean(train.DISBURSED_AMOUNT), std(train.DISBURSED_AMOUNT)
print('Statistics: Mean=%.3f, Std dev=%.3f' % (data_mean, data_std))
# identify outliers
cut_off = data_std * 3
lower, upper = data_mean - cut_off, data_mean + cut_off
# identify outliers
outliers = [x for x in train.DISBURSED_AMOUNT if x < lower or x > upper]
print('Identified outliers: %d' % len(outliers))
```

```
No. of observations in column:  233154
Statistics: Mean=54356.994, Std dev=12971.286
Identified outliers: 3076
```

In [27]:

```
def impute_outlier(x):
    if x <= lower:
        return(data_mean)
    elif x>= (upper):
        return(data_mean)
    else:
        return(x)
train["DISBURSED_AMOUNT_new"]= train["DISBURSED_AMOUNT"].apply(impute_outlier)
print("No. of observations in column: ",len(train.DISBURSED_AMOUNT_new))
```

```
No. of observations in column:  233154
```

Binning

mean 54356.993528

std 12971.314171

min 13320.000000

25% 47145.000000

50% 53803.000000

75% 60413.000000

max 990572.000000

In [28]:

```
bin_labels = ['Low', 'Medium', 'High', 'Extreme']
train['DISBURSED_AMOUNT_bins'] = pd.qcut(train['DISBURSED_AMOUNT'],
                              q=[0, .25, .5, .75, 1],
                              labels=bin_labels)
train['DISBURSED_AMOUNT_bins'].value_counts()
```

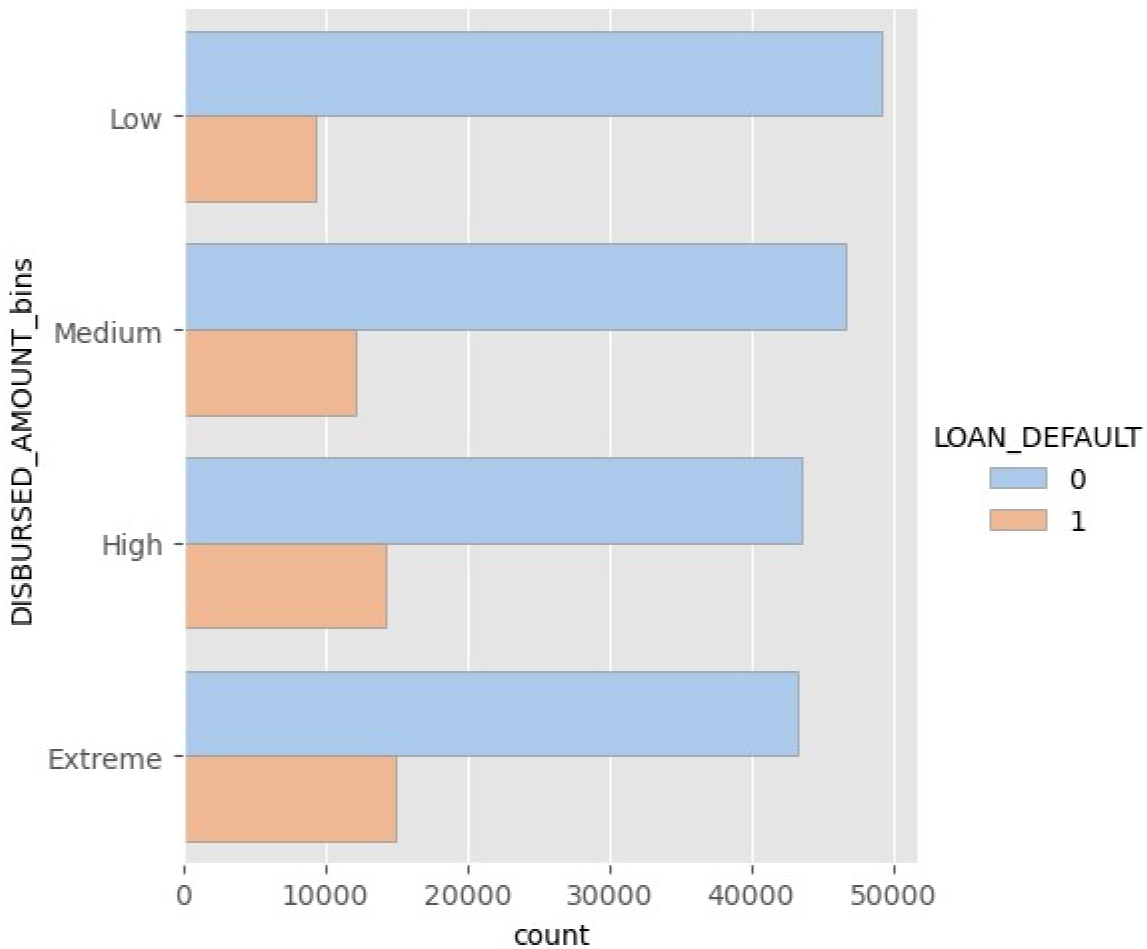
Out[28]:

```
Medium      58676
Low         58537
Extreme     58207
High        57734
Name: DISBURSED_AMOUNT_bins, dtype: int64
```

In [29]:

```
plot_bar("DISBURSED_AMOUNT_bins")
```

```
<Figure size 1000x5000 with 0 Axes>
```

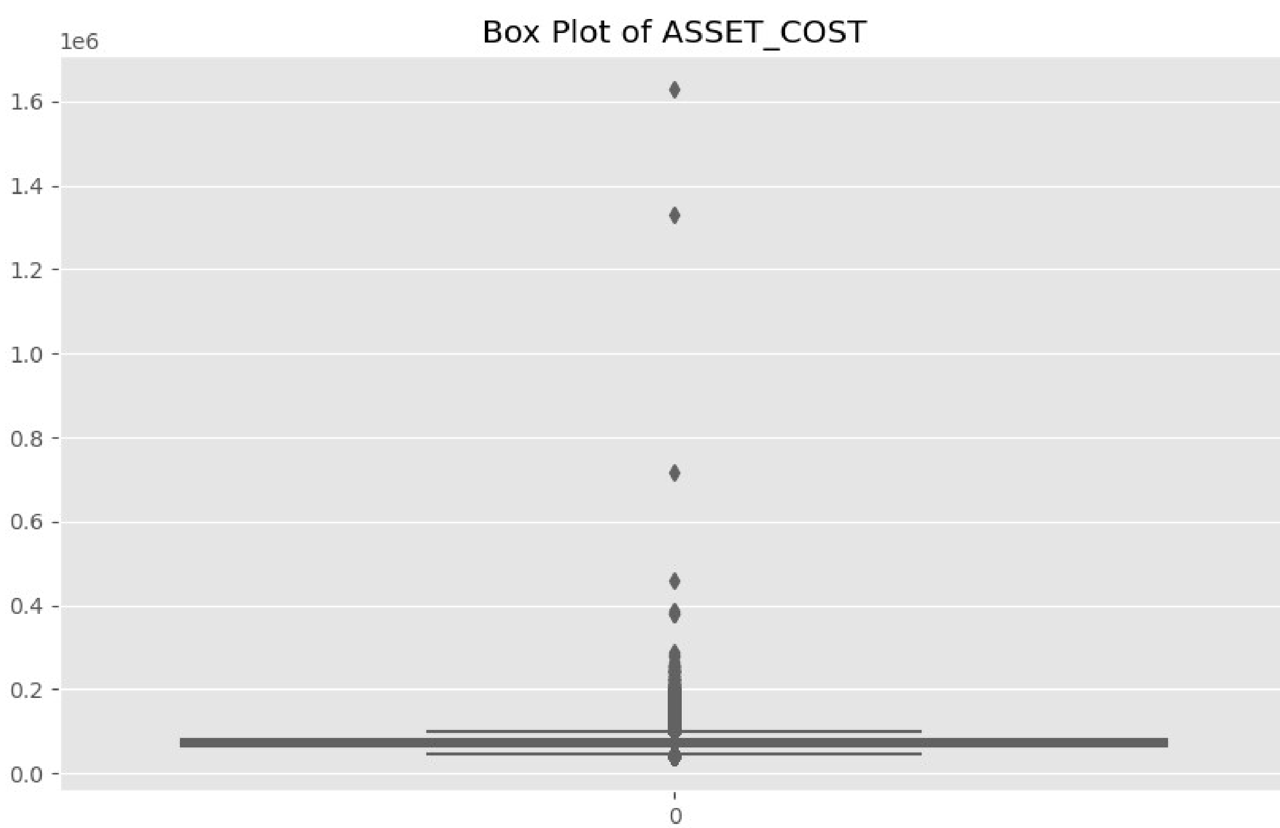


'ASSET_COST' : Payment default in the first EMI on due date

In [30]:
```
print(train.ASSET_COST.describe().astype(str))
plot_distribution('ASSET_COST','tomato')
```

```
count             233154.0
mean     75865.06814380195
std      18944.78128866517
min                37000.0
25%                65717.0
50%                70946.0
75%               79201.75
max              1628992.0
Name: ASSET_COST, dtype: object
```

In [31]:
```
plot_box("ASSET_COST", "tomato")
```



In [32]:

In [33]:
```
outlier_data(train,"ASSET_COST")
```

```
No. of observations in column:  233154
Statistics: Mean=75865.068, Std dev=18944.741
Identified outliers: 4425
```

In [34]:
```
train["ASSET_COST_new"]= train["ASSET_COST"].apply(impute_outlier)
print("No. of observations in column: ",len(train.DISBURSED_AMOUNT_new))
outlier_data(train,"ASSET_COST_new")
```

```
No. of observations in column:  233154
No. of observations in column:  233154
Statistics: Mean=68018.188, Std dev=9598.448
Identified outliers: 60
```

Binning

mean 75865.06814380195

std 18944.78128866517

min 37000.0

25% 65717.0

50% 70946.0

75% 79201.75

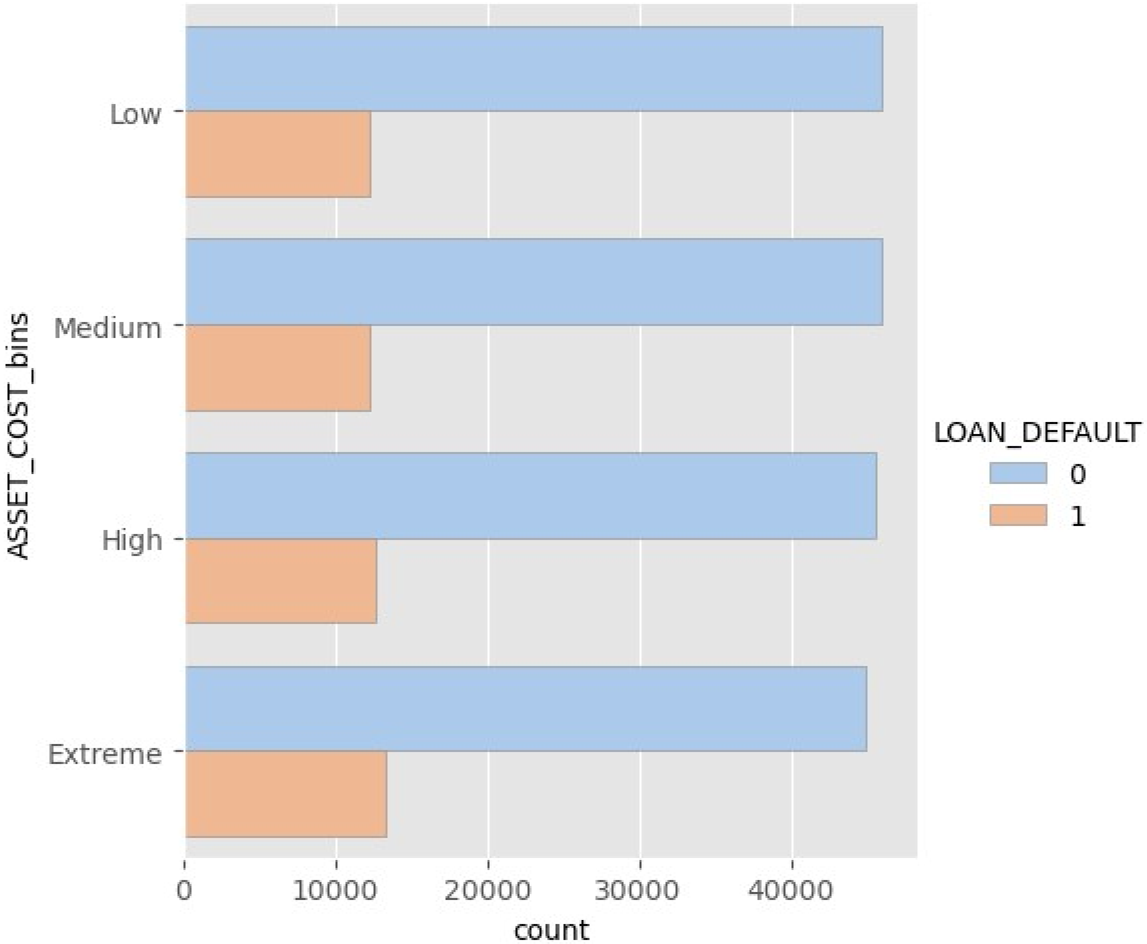
max 1628992.0

In [35]:
```
bin_labels = ['Low', 'Medium', 'High', 'Extreme']
train['ASSET_COST_bins'] = pd.qcut(train['ASSET_COST'],
                              q=[0, .25, .5, .75, 1],
                              labels=bin_labels)
train['ASSET_COST_bins'].value_counts()
```

Out[35]:
```
Low        58290
Extreme    58289
Medium     58288
High       58287
Name: ASSET_COST_bins, dtype: int64
```

In [36]:
```
plot_bar("ASSET_COST_bins")
```

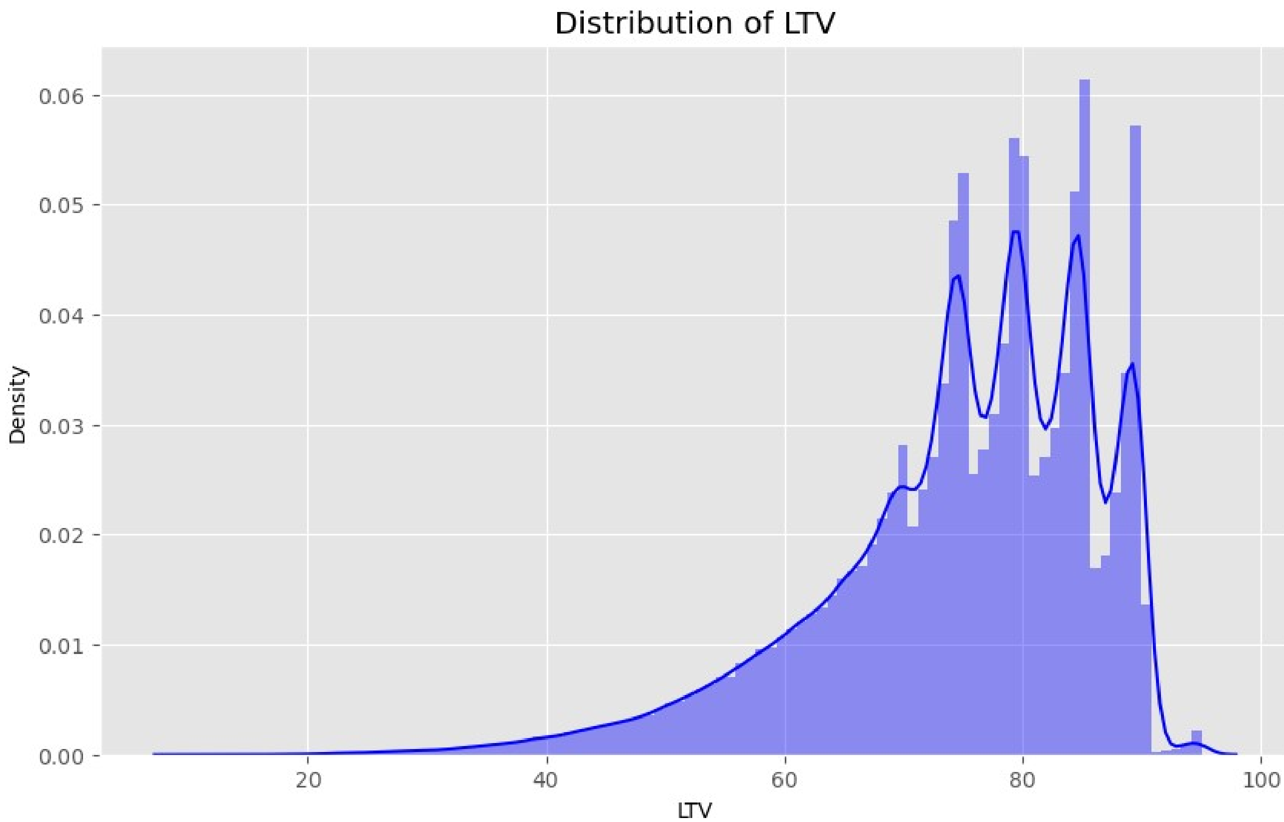
```
<Figure size 1000x5000 with 0 Axes>
```



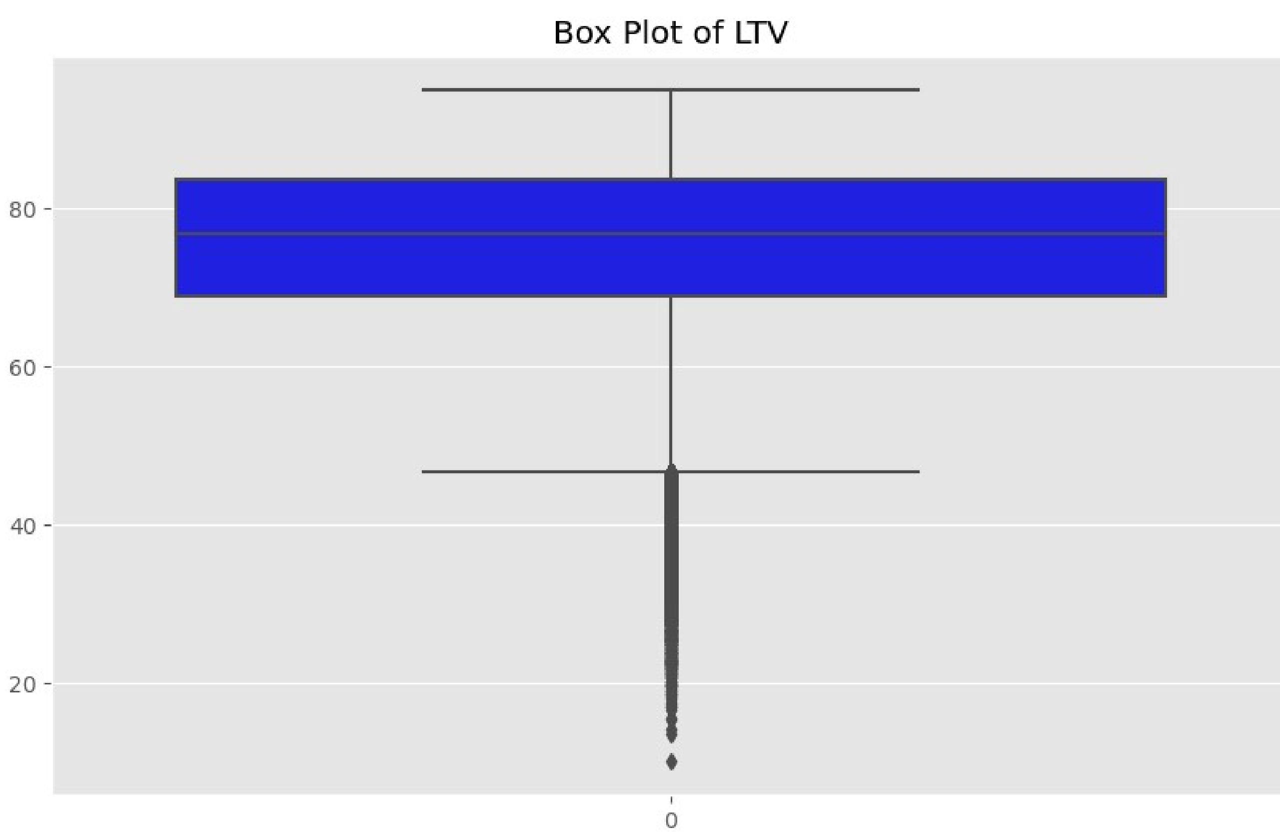
LTV

In [37]:
```
print(train.LTV.describe().astype(str))
plot_distribution('LTV','blue')
```

```
count              233154.0
mean       74.74653001878589
std       11.456635738792304
min                   10.03
25%                   68.88
50%                    76.8
75%                   83.67
max                    95.0
Name: LTV, dtype: object
```



In [38]: `plot_box("LTV", "blue")`



In [39]: `outlier_data(train,"LTV")`

```
No. of observations in column:  233154
Statistics: Mean=74.747, Std dev=11.457
Identified outliers: 2745
```

In [40]:
```
train["LTV_new"]= train["LTV"].apply(impute_outlier)
print("No. of observations in column: ",len(train.LTV_new))
outlier_data(train,"LTV_new")
```

```
No. of observations in column:  233154
No. of observations in column:  233154
Statistics: Mean=54356.994, Std dev=0.000
Identified outliers: 0
```

Binning

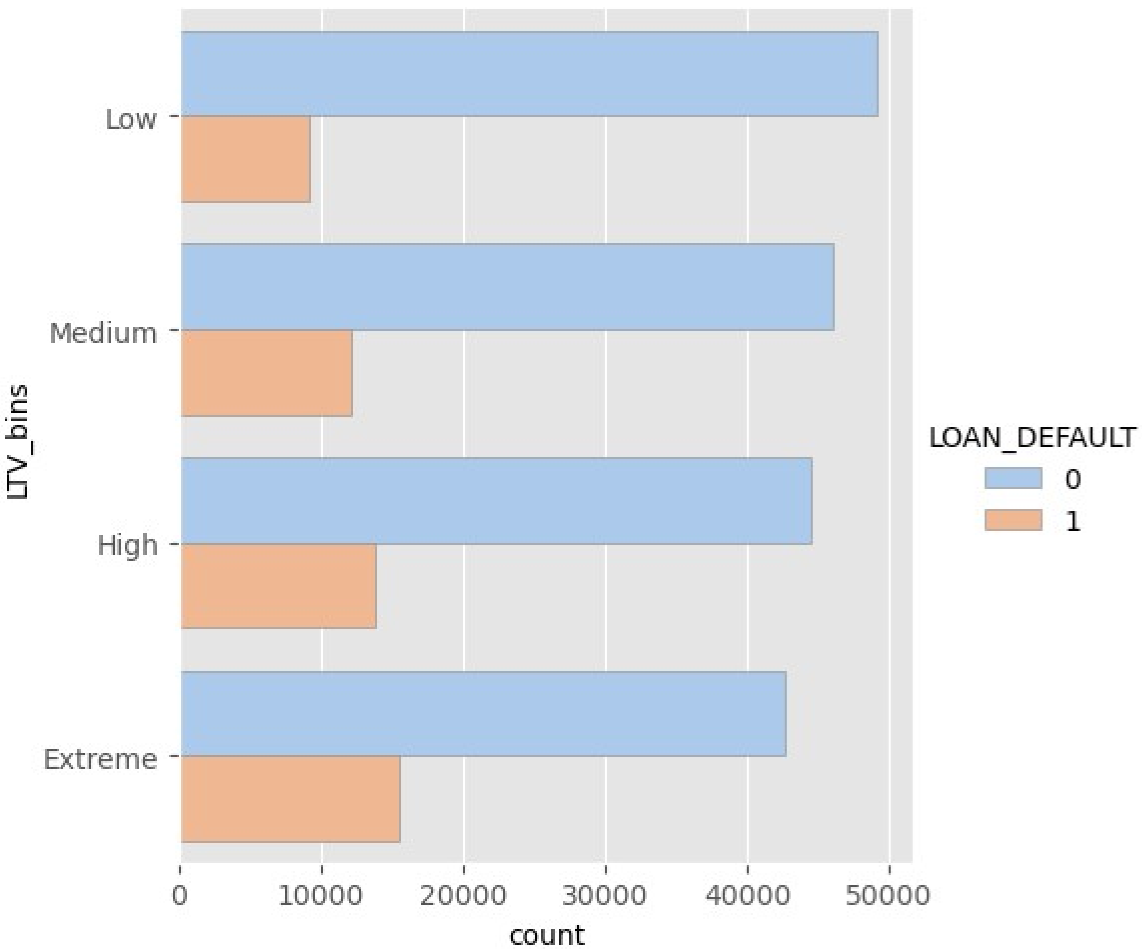
mean 74.74653001879038
std 11.456635738792304
min 10.03
25% 68.88
50% 76.8
75% 83.67
max 95.0

In [41]:
```
bin_labels = ['Low', 'Medium', 'High', 'Extreme']
train['LTV_bins'] = pd.qcut(train['LTV'],
                              q=[0, .25, .5, .75, 1],
                              labels=bin_labels)
train['LTV_bins'].value_counts()
```

Out[41]:
```
Low        58303
Medium     58299
High       58285
Extreme    58267
Name: LTV_bins, dtype: int64
```

In [42]:
```
plot_bar("LTV_bins")
```

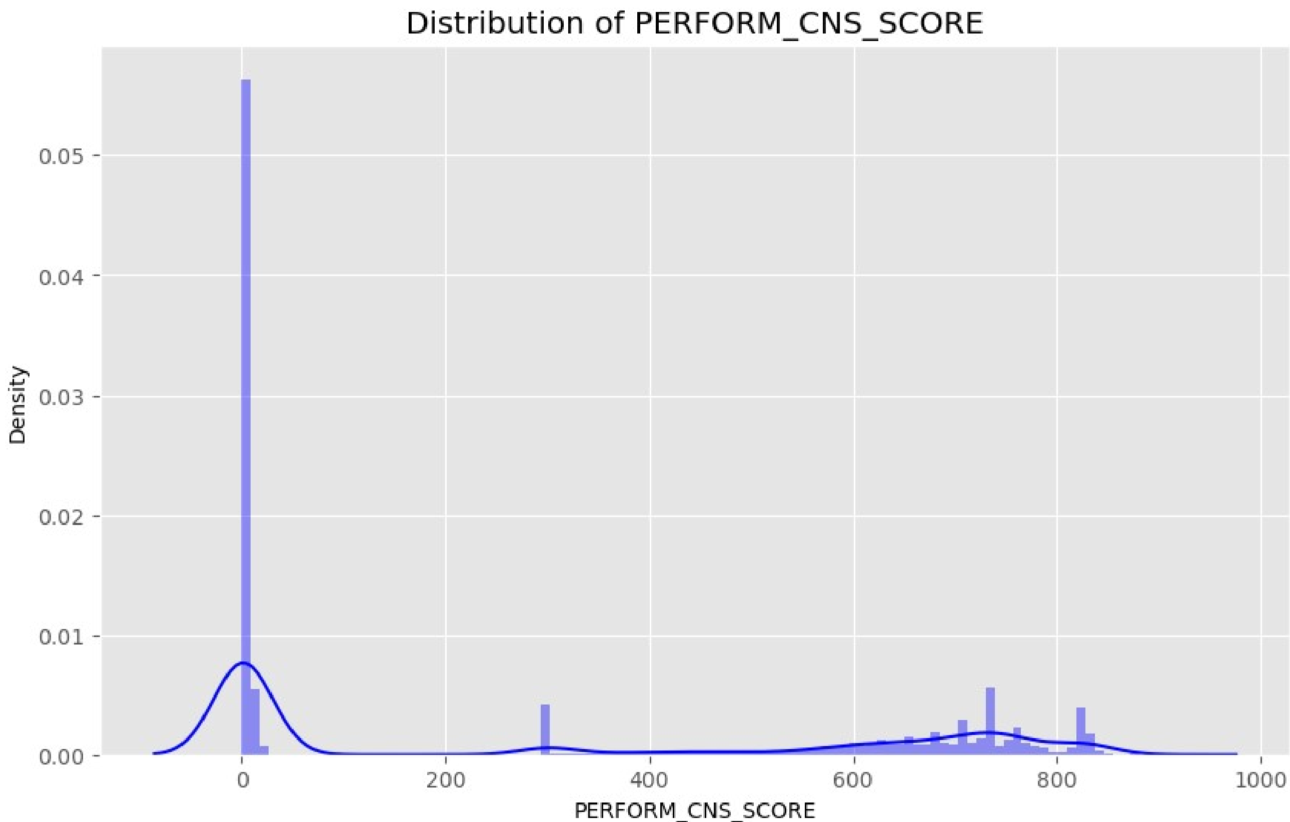
```
<Figure size 1000x5000 with 0 Axes>
```



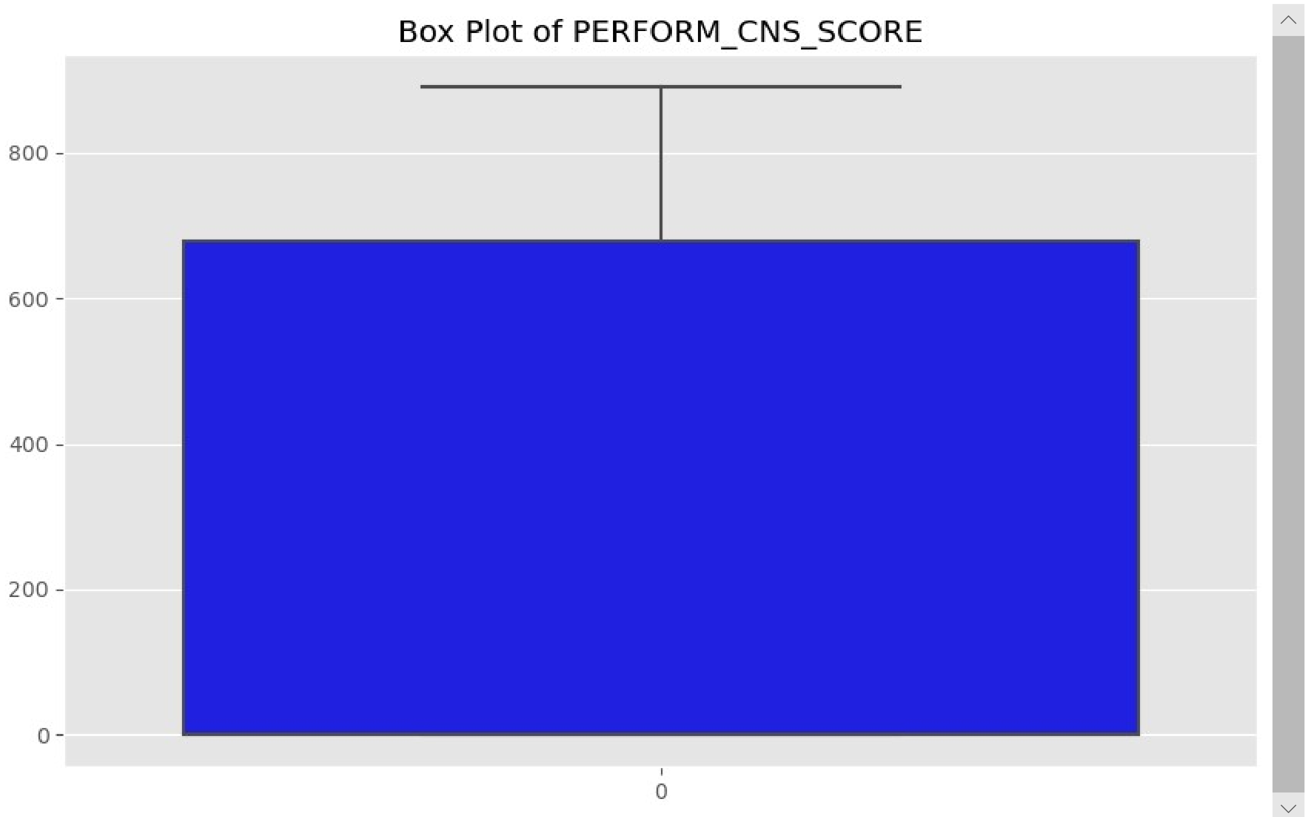
'PERFORM_CNS_SCORE': Bureau Score

In [43]:
```
print(train.PERFORM_CNS_SCORE.describe().astype(str))
plot_distribution('PERFORM_CNS_SCORE','blue')
```

```
count              233154.0
mean      289.46299441570807
std        338.3747790080087
min                     0.0
25%                     0.0
50%                     0.0
75%                   678.0
max                   890.0
Name: PERFORM_CNS_SCORE, dtype: object
```



In [44]:
```
plot_box("PERFORM_CNS_SCORE", "blue")
```



In [45]:
```
outlier_data(train,"PERFORM_CNS_SCORE")
```

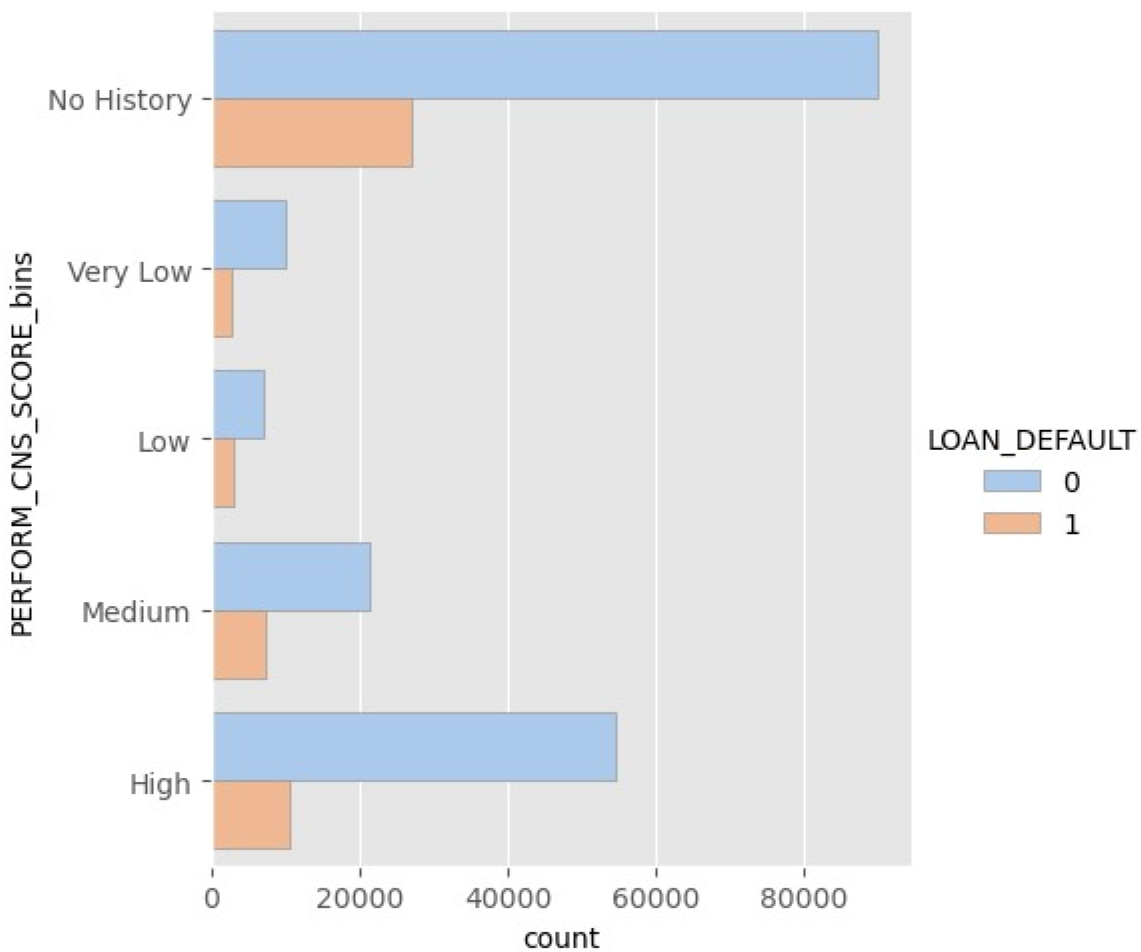
```
No. of observations in column:  233154
Statistics: Mean=289.463, Std dev=338.374
Identified outliers: 0
```

In [46]:
```
bin_labels = ["No History",'Very Low', "Low" ,'Medium', 'High']
cut_bins = [-1,10,150, 350, 650, 1000]

train['PERFORM_CNS_SCORE_bins'] = pd.cut(train['PERFORM_CNS_SCORE'],
                                         bins=cut_bins,
                                         labels=bin_labels)
train['PERFORM_CNS_SCORE_bins'].value_counts()
```

Out[46]:
```
No History    116950
High           65034
Medium         28425
Very Low       12835
Low             9910
Name: PERFORM_CNS_SCORE_bins, dtype: int64
```

In [47]:
```
plot_bar("PERFORM_CNS_SCORE_bins")
```

```
<Figure size 1000x5000 with 0 Axes>
```



In [4...
```
train.groupby(["PERFORM_CNS_SCORE_DESCRIPTION"]).PERFORM_CNS_SCORE_bins.value_co
```

Out[48]:PERFORM_CNS_SCORE_DESCRIPTION PERFORM_CNS_SCORE_bin

s

A-Very Low Risk High

14124

No History

0

Very Low

0

Low

0

Medium

0

B-Very Low Risk High

9201

No History

0

Very Low

0

Low

0

Medium

0

C-Very Low Risk High

16045

No History

0

Very Low

0

Low

0

Medium

0

D-Very Low Risk High

11358

No History

0

Very Low

0

Low

0

Medium

0

E-Low Risk High

5821

No History

0

Very Low

0

Low

0

Medium

0

F-Low Risk High
8485
No History
0
Very Low
0
Low
0
Medium
0
G-Low Risk Medium
3988
No History
0
Very Low
0
Low
0
High
0
H-Medium Risk Medium
6855
No History
0
Very Low
0
Low
0
High
0
I-Medium Risk Medium
5557
No History
0
Very Low
0
Low
0
High
0
J-High Risk Medium
3748
No History
0
Very Low
0
Low
0
High
0
K-High Risk Medium
8277

No History
0
Very Low
0
Low
0
High
0
L-Very High Risk Low
1134
No History
0
Very Low
0
Medium
0
High
0
M-Very High Risk Low
8776
No History
0
Very Low
0
Medium
0
High
0
No Bureau History Available No History
116950
Very Low
0
Low
0
Medium
0
High
0
Not Scored: More than 50 active Accounts found Very Low
3
No History
0
Low
0
Medium
0
High
0
Not Scored: No Activity seen on the customer (Inactive) Very Low
2885
No History
0

```
                                                         Low
0
                                                         Medium
0
                                                         High
0
Not Scored: No Updates available in last 36 months       Very Low
1534
                                                         No History
0
                                                         Low
0
                                                         Medium
0
                                                         High
0
Not Scored: Not Enough Info available on the customer    Very Low
3672
                                                         No History
0
                                                         Low
0
                                                         Medium
0
                                                         High
0
Not Scored: Only a Guarantor                             Very Low
976
                                                         No History
0
                                                         Low
0
                                                         Medium
0
                                                         High
0
Not Scored: Sufficient History Not Available             Very Low
3765
                                                         No History
0
                                                         Low
0
                                                         Medium
0
                                                         High
0
Name: PERFORM_CNS_SCORE_bins, dtype: int64
```

PERFORM_CNS_SCORE_DESCRIPTION

In [49]:
```
train.PERFORM_CNS_SCORE_DESCRIPTION.value_counts()
```

Out[49]:
```
No Bureau History Available                                116950
C-Very Low Risk                                             16045
A-Very Low Risk                                             14124
D-Very Low Risk                                             11358
B-Very Low Risk                                              9201
M-Very High Risk                                             8776
F-Low Risk                                                   8485
K-High Risk                                                  8277
H-Medium Risk                                                6855
E-Low Risk                                                   5821
I-Medium Risk                                                5557
G-Low Risk                                                   3988
Not Scored: Sufficient History Not Available                 3765
J-High Risk                                                  3748
Not Scored: Not Enough Info available on the customer        3672
Not Scored: No Activity seen on the customer (Inactive)      2885
Not Scored: No Updates available in last 36 months           1534
L-Very High Risk                                             1134
Not Scored: Only a Guarantor                                  976
Not Scored: More than 50 active Accounts found                  3
Name: PERFORM_CNS_SCORE_DESCRIPTION, dtype: int64
```

In [...
```
g = train.groupby("PERFORM_CNS_SCORE_DESCRIPTION")['LOAN_DEFAULT']
gg = pd.concat([g.value_counts(),
                g.value_counts(normalize=True).mul(100)],axis=1, keys=('counts',
print (gg)

#train.groupby("PERFORM_CNS_SCORE_DESCRIPTION").LOAN_DEFAULT.value_counts(normal
```

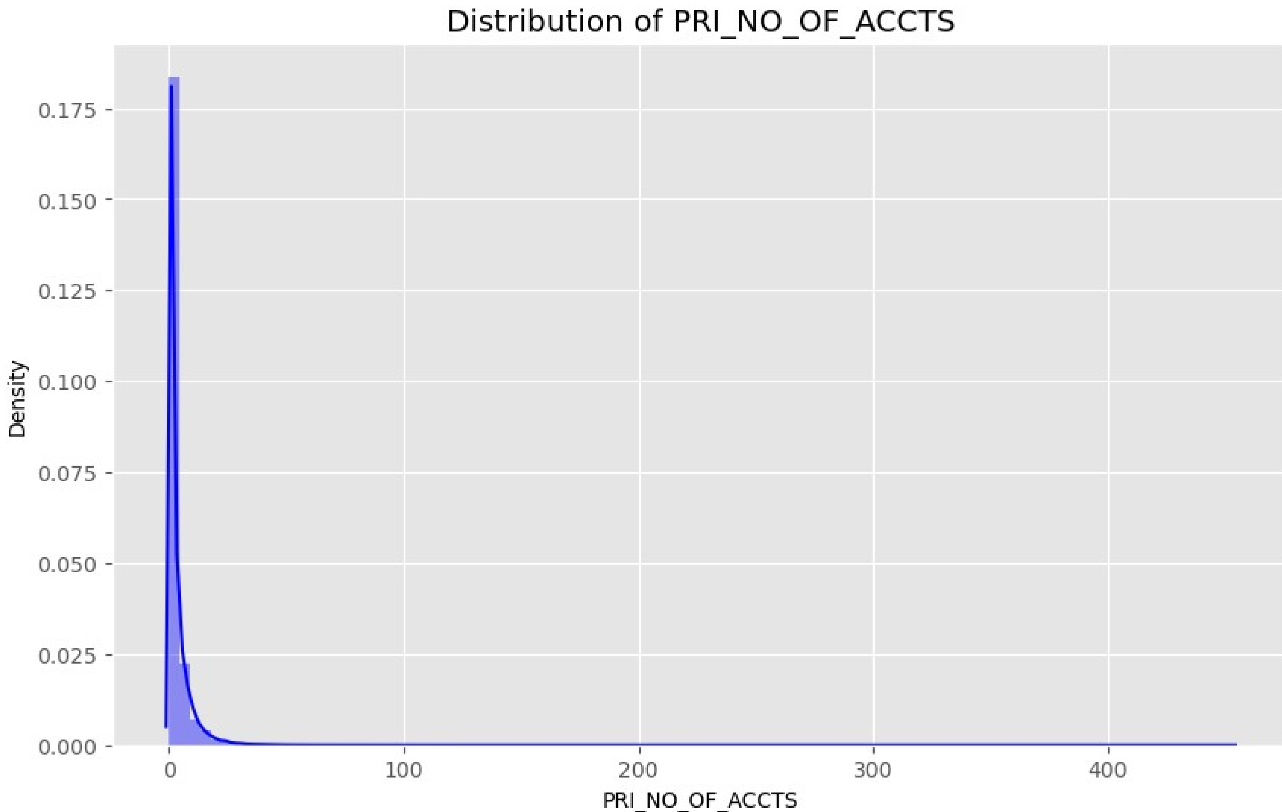
```
                                                                counts  percentage
PERFORM_CNS_SCORE_DESCRIPTION                       LOAN_DEFAULT
A-Very Low Risk                                     0            11783   83.425375
                                                    1             2341   16.574625
B-Very Low Risk                                     0             7993   86.870992
                                                    1             1208   13.129008
C-Very Low Risk                                     0            13275   82.736055
                                                    1             2770   17.263945
D-Very Low Risk                                     0             9659   85.041381
                                                    1             1699   14.958619
E-Low Risk                                          0             4821   82.820821
                                                    1             1000   17.179179
F-Low Risk                                          0             6905   81.378904
                                                    1             1580   18.621096
G-Low Risk                                          0             3202   80.290873
                                                    1              786   19.709127
H-Medium Risk                                       0             5197   75.813275
                                                    1             1658   24.186725
I-Medium Risk                                       0             4042   72.737088
                                                    1             1515   27.262912
J-High Risk                                         0             2802   74.759872
                                                    1              946   25.240128
K-High Risk                                         0             5975   72.187991
                                                    1             2302   27.812009
L-Very High Risk                                    0              816   71.957672
                                                    1              318   28.042328
M-Very High Risk                                    0             6103   69.541933
                                                    1             2673   30.458067
No Bureau History Available                         0            89898   76.868747
                                                    1            27052   23.131253
Not Scored: More than 50 active Accounts found      0                3  100.000000
Not Scored: No Activity seen on the customer (I...  0             2355   81.629116
                                                    1              530   18.370884
Not Scored: No Updates available in last 36 months  0             1242   80.964798
                                                    1              292   19.035202
Not Scored: Not Enough Info available on the cu...  0             2902   79.030501
                                                    1              770   20.969499
Not Scored: Only a Guarantor                        0              768   78.688525
                                                    1              208   21.311475
Not Scored: Sufficient History Not Available        0             2802   74.422311
                                                    1              963   25.577689
```

PRI_NO_OF_ACCTS : count of total loans taken by the customer at the time of disbursement

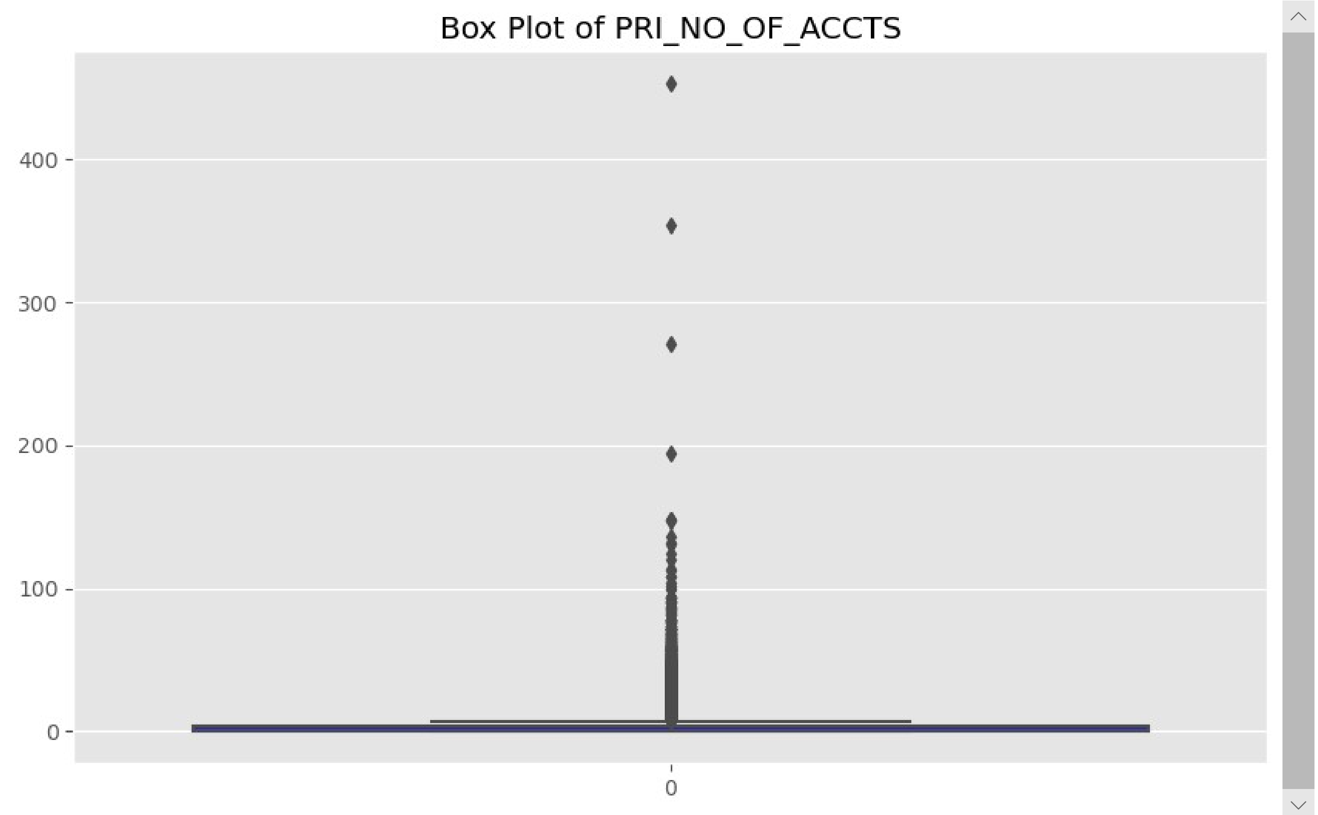
In [51]:

```
print(train.PRI_NO_OF_ACCTS .describe().astype(str))
plot_distribution('PRI_NO_OF_ACCTS','blue')
```

```
count              233154.0
mean     2.4406358029456925
std       5.217233021576896
min                     0.0
25%                     0.0
50%                     0.0
75%                     3.0
max                   453.0
Name: PRI_NO_OF_ACCTS, dtype: object
```



In [52]:
```
plot_box("PRI_NO_OF_ACCTS", "blue")
```



In [53]:
```
outlier_data(train,"PRI_NO_OF_ACCTS")
```

```
No. of observations in column:  233154
Statistics: Mean=2.441, Std dev=5.217
Identified outliers: 4119
```

In [54]:
```
train["PRI_NO_OF_ACCTS_new"]= train["PRI_NO_OF_ACCTS"].apply(impute_outlier)
outlier_data(train,"PRI_NO_OF_ACCTS_new")
```

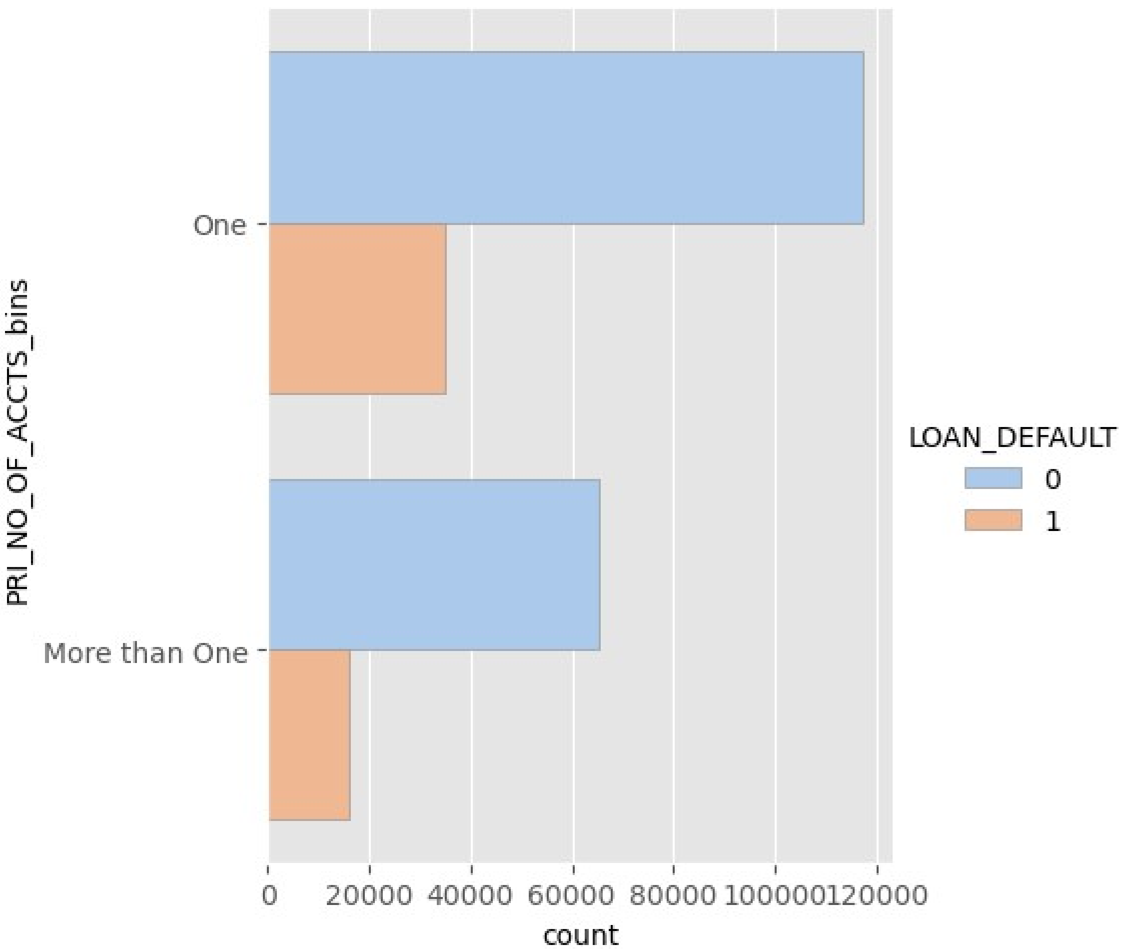
```
No. of observations in column:  233154
Statistics: Mean=54356.994, Std dev=0.000
Identified outliers: 0
```

In [55]:
```
bin_labels = ["One",'More than One']
cut_bins = [-1,1, 1000]

train['PRI_NO_OF_ACCTS_bins'] = pd.cut(train['PRI_NO_OF_ACCTS'],
                              bins=cut_bins,
                              labels=bin_labels)
train['PRI_NO_OF_ACCTS_bins'].value_counts()
```

Out[55]:
```
One              151928
More than One     81226
Name: PRI_NO_OF_ACCTS_bins, dtype: int64
```

In [56]:
```
plot_bar("PRI_NO_OF_ACCTS_bins")
```

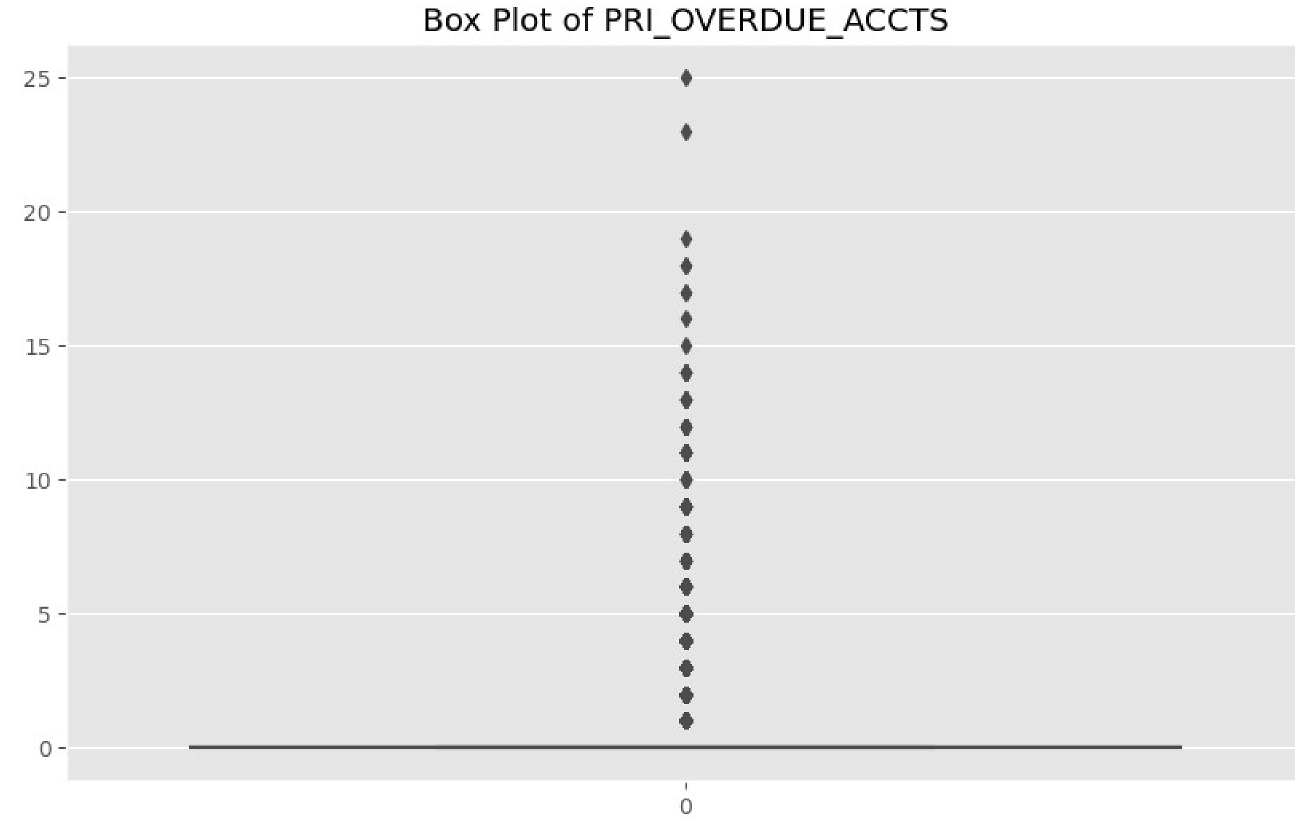
```
<Figure size 1000x5000 with 0 Axes>
```



PRI_OVERDUE_ACCTS: count of default accounts at the time of disbursement

In [57]:
```
print(train.PRI_OVERDUE_ACCTS.describe().astype(str))
plot_box("PRI_OVERDUE_ACCTS", "blue")
```

```
count                233154.0
mean      0.15654889043293274
std        0.5487867498784913
min                       0.0
25%                       0.0
50%                       0.0
75%                       0.0
max                      25.0
Name: PRI_OVERDUE_ACCTS, dtype: object
```



In [58]:
```
outlier_data(train,"PRI_OVERDUE_ACCTS")
```

```
No. of observations in column:  233154
Statistics: Mean=0.157, Std dev=0.549
Identified outliers: 6305
```

In [5…
```
train["PRI_OVERDUE_ACCTS_new"]= train["PRI_OVERDUE_ACCTS"].apply(impute_outlier
outlier_data(train,"PRI_OVERDUE_ACCTS_new")
```

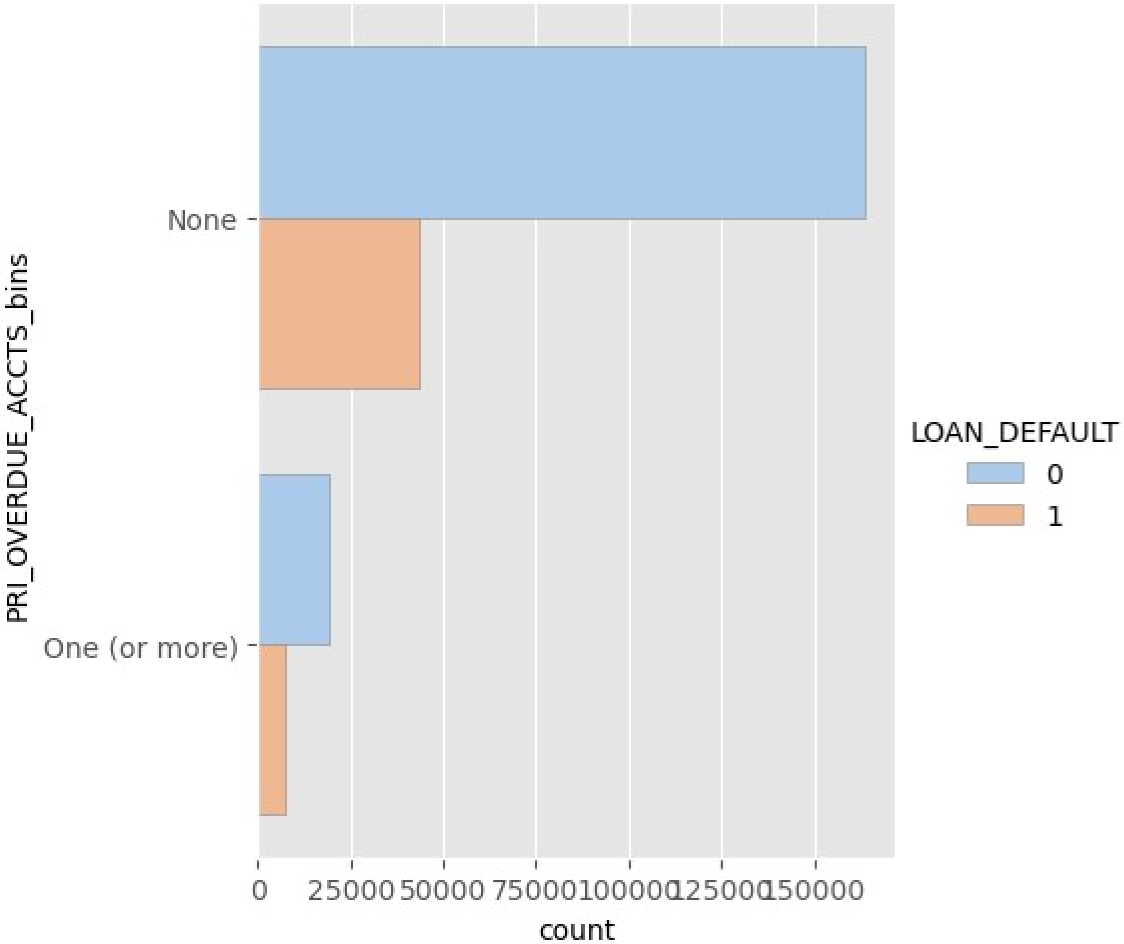
```
No. of observations in column:  233154
Statistics: Mean=54356.994, Std dev=0.000
Identified outliers: 0
```

In [60]:
```
bin_labels = ["None",'One (or more)']
cut_bins = [-1,0, 1000]

train['PRI_OVERDUE_ACCTS_bins'] = pd.cut(train['PRI_OVERDUE_ACCTS'],
                                   bins=cut_bins,
                                   labels=bin_labels)
train['PRI_OVERDUE_ACCTS_bins'].value_counts()
```

Out[60]:
```
None              206879
One (or more)      26275
Name: PRI_OVERDUE_ACCTS_bins, dtype: int64
```

In [61]:
```
plot_bar("PRI_OVERDUE_ACCTS_bins")
```

```
<Figure size 1000x5000 with 0 Axes>
```



Let's look into data with lesser importance

MOBILENO_AVL_FLAG : if Mobile no. was shared by the customer then flagged as 1

AADHAR_FLAG : if aadhar was shared by the customer then flagged as 1

PAN_FLAG : if pan was shared by the customer then flagged as 1

VOTERID_FLAG : if voter was shared by the customer then flagged as 1

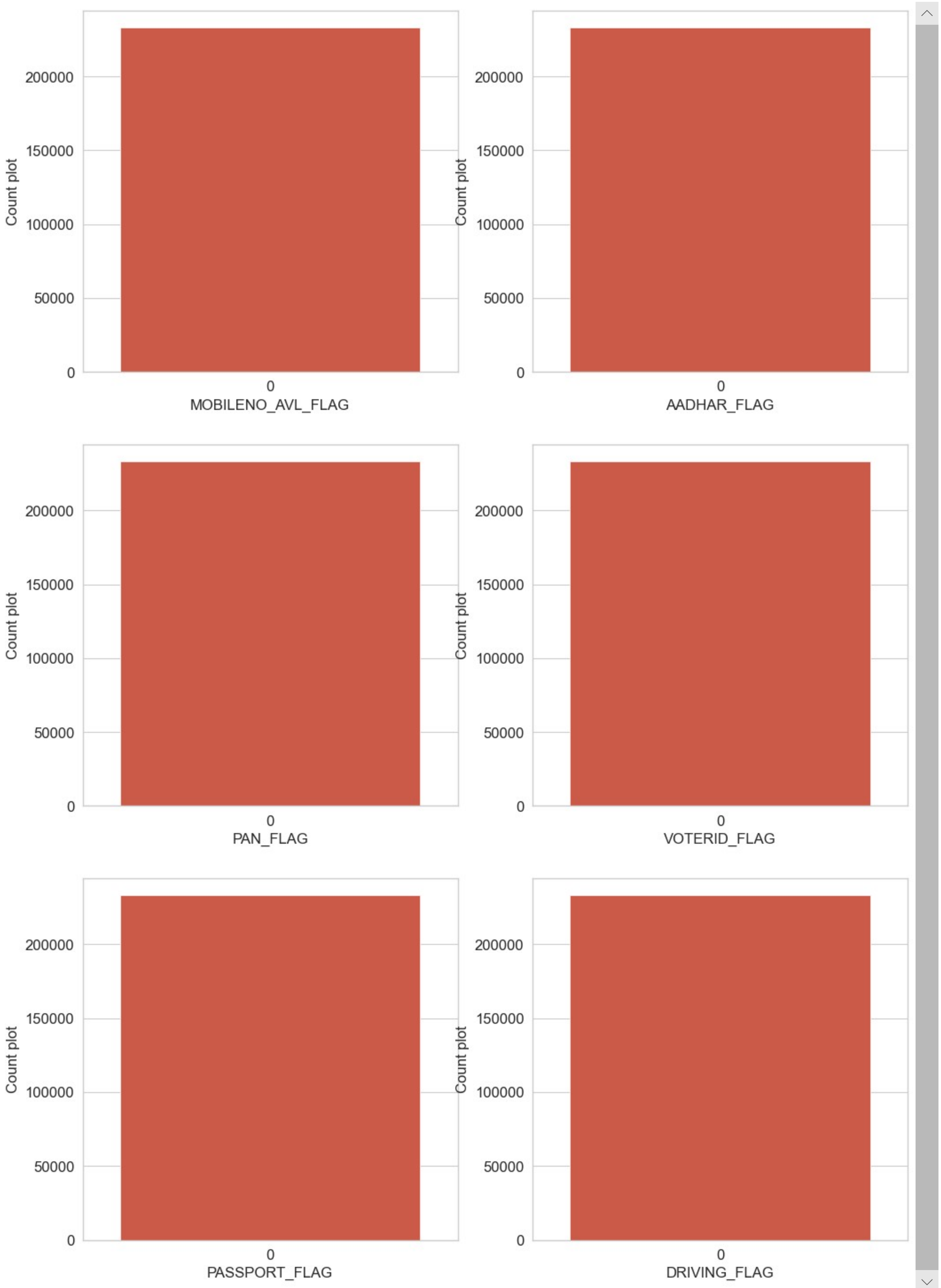
PASSPORT_FLAG : if DL was shared by the customer then flagged as 1

DRIVING_FLAG : if passport was shared by the customer then flagged as 1

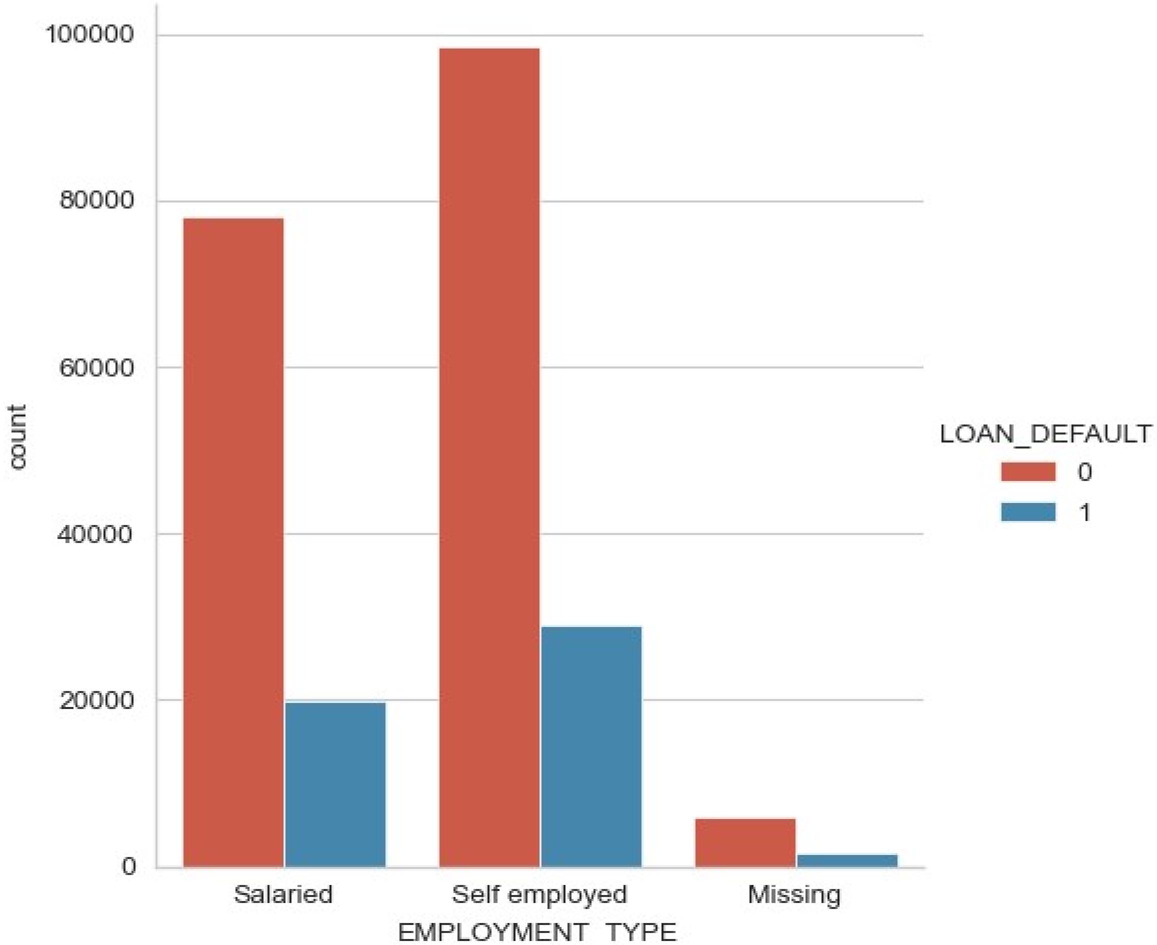
In [...

```
var = ['MOBILENO_AVL_FLAG', 'AADHAR_FLAG', 'PAN_FLAG', 'VOTERID_FLAG', 'PASSPORT_
plot_bar_comp(var,nrow=3)
```

```
<Figure size 640x480 with 0 Axes>
```


Count plot
200000
150000
100000
50000
0
0
MOBILENO_AVL_FLAG
AADHAR_FLAG
PAN_FLAG
VOTERID_FLAG
PASSPORT_FLAG
DRIVING_FLAG

Out[63]:<seaborn.axisgrid.FacetGrid at 0x1e4c7f51150>



Age is in days

In...
```
now = pd.Timestamp('now')
#train['DATE_OF_BIRTH'] = train['DATE_OF_BIRTH'].where(train['DATE_OF_BIRTH'] < no
train['age'] = (now - train['DATE_OF_BIRTH'])

train['age']= train['age'].astype(str)
train[['age','age_waste']] = train['age'].str.split("days",expand=True)
train['age']= train['age'].astype(str).astype(int)
train= train.drop(columns= ['age_waste'])

print(train['age'].head())
```

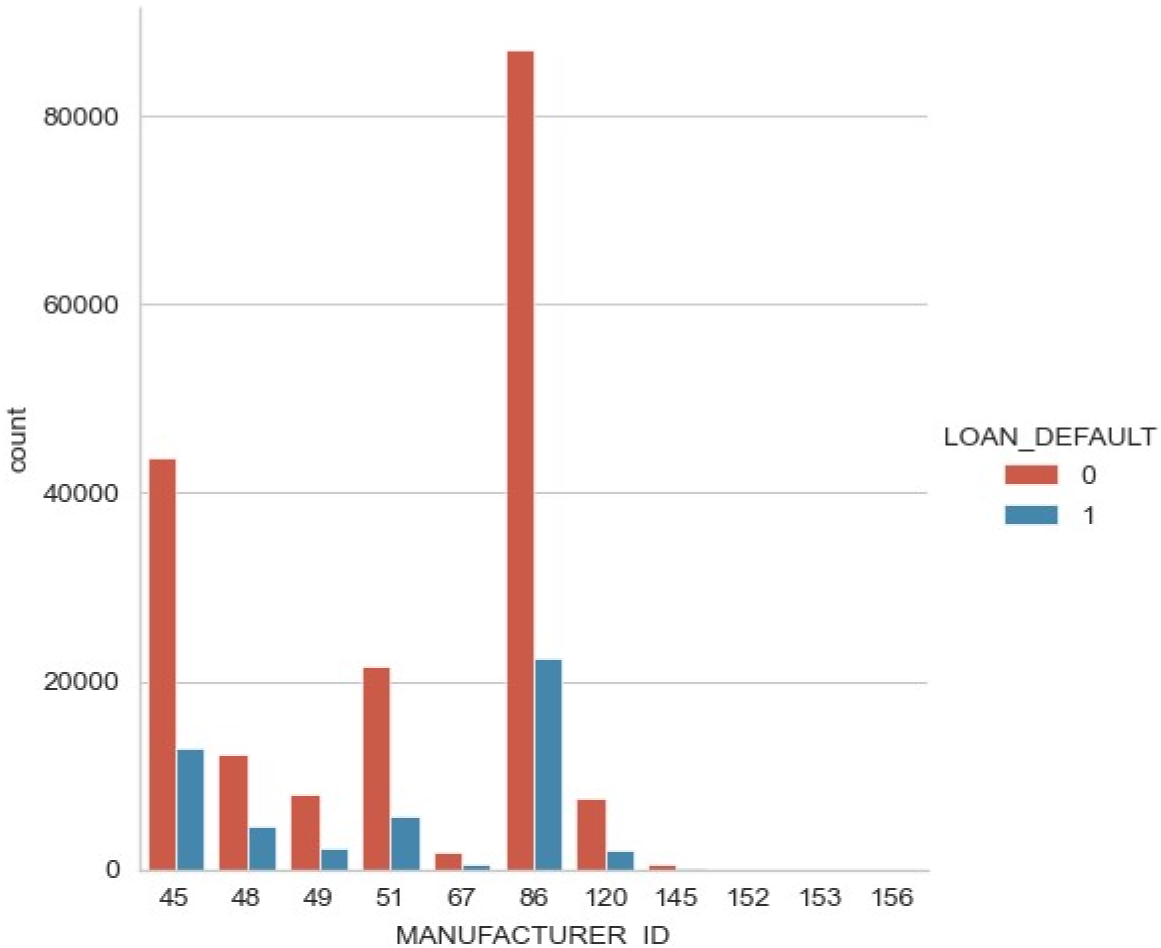
```
0    14466
1    13889
2    13865
3    10815
4    16680
Name: age, dtype: int32
```

In [...
```
train['disbursal_time'] = (now - train['DISBURSAL_DATE'])

train['disbursal_time']= train['disbursal_time'].astype(str)
train[['disbursal_time','disbursal_time_waste']] = train['disbursal_time'].str.s
```

```
0    1833
1    1779
2    1835
3    1749
4    1779
Name: disbursal_time, dtype: int32
```

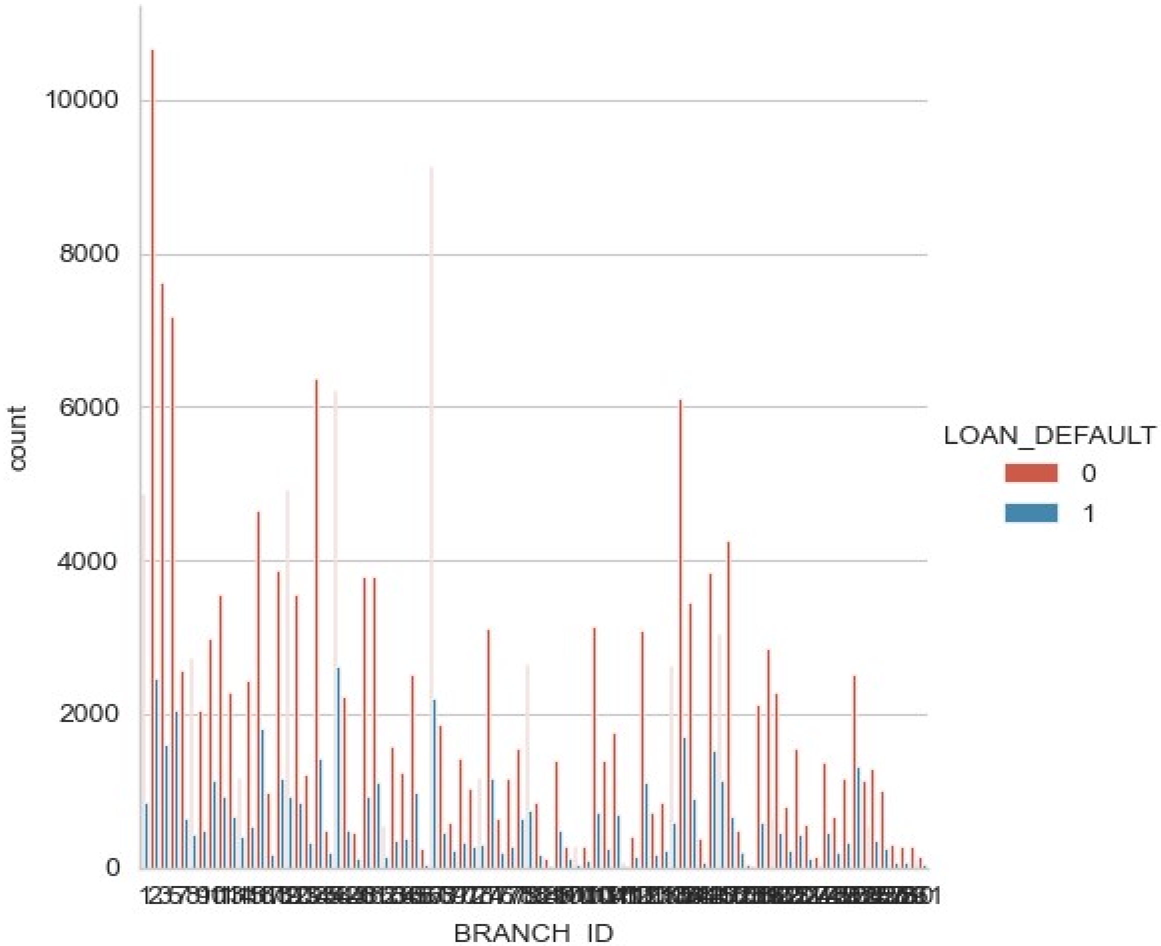
In [66]:
```
# MANUFACTURER_ID
sns.catplot(data=train,kind='count',x='MANUFACTURER_ID',hue='LOAN_DEFAULT')
```

Out[66]:
```
<seaborn.axisgrid.FacetGrid at 0x1e4d3b5b5b0>
```



In [67]:
```
sns.catplot(data=train,kind='count',x='BRANCH_ID',hue='LOAN_DEFAULT')
```

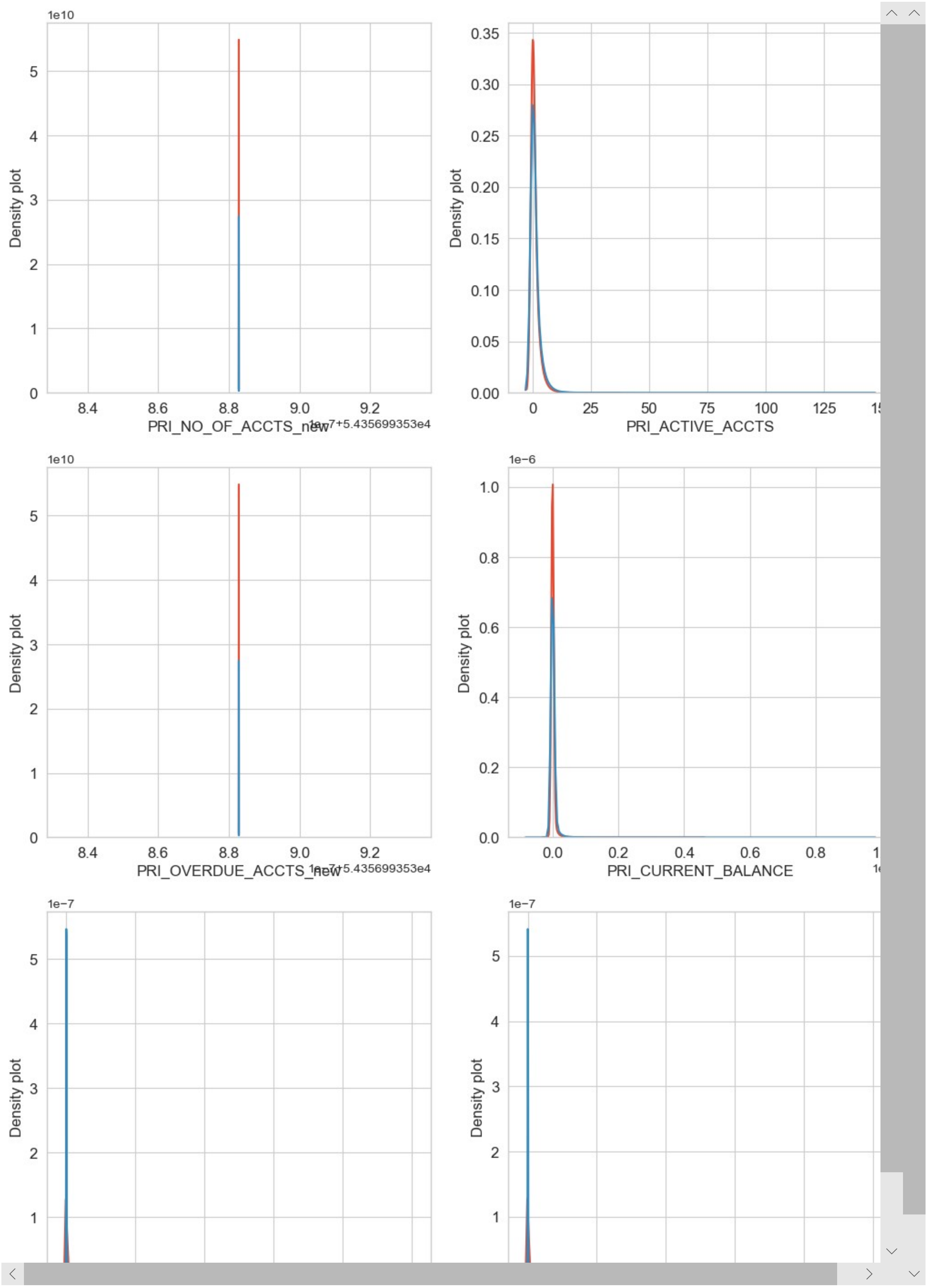
Out[67]:
```
<seaborn.axisgrid.FacetGrid at 0x1e4d6159540>
```



Let's see the new columns along with the less important continous variables

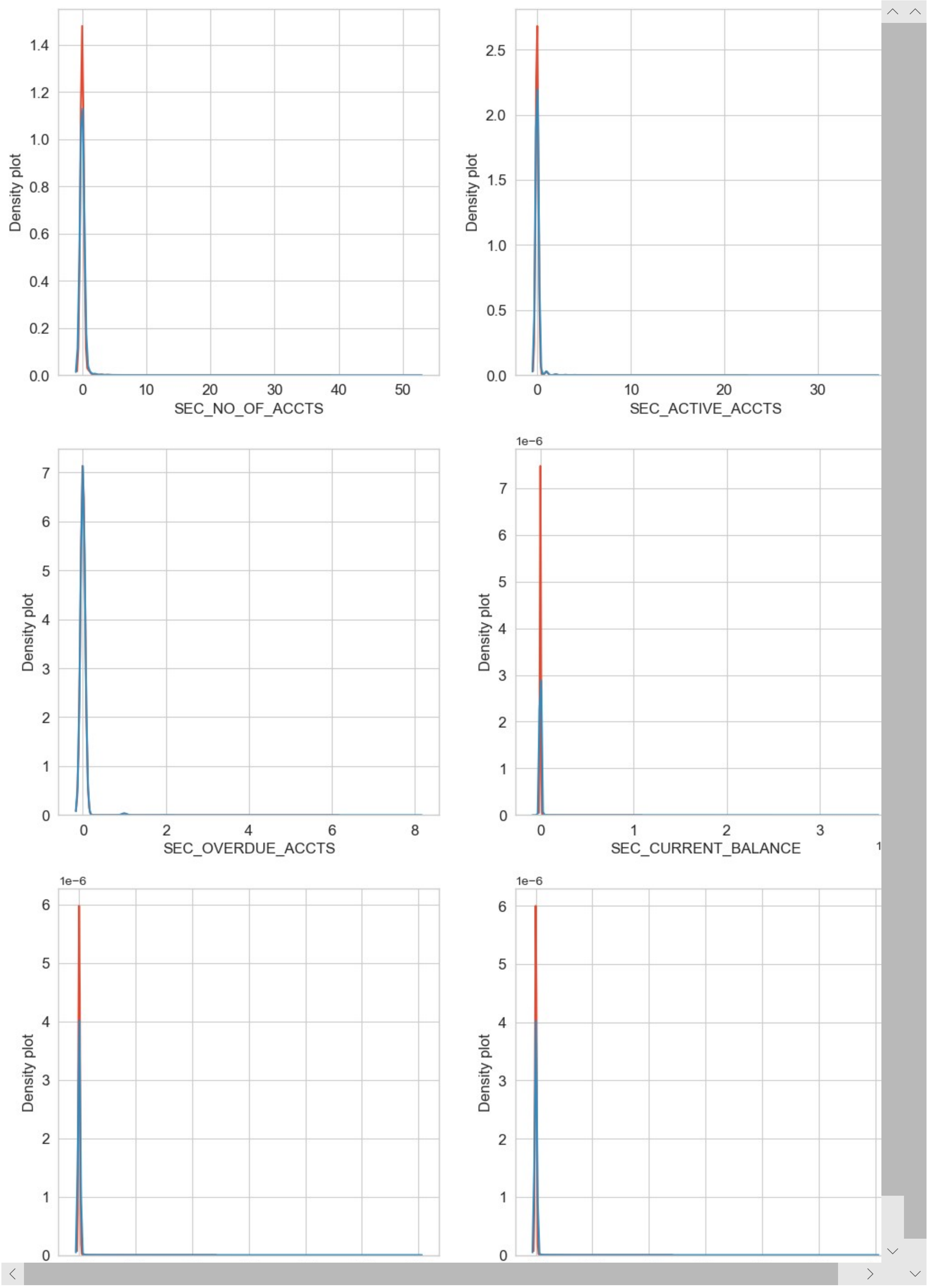
In...
```
var = ['PRI_NO_OF_ACCTS_new', 'PRI_ACTIVE_ACCTS', 'PRI_OVERDUE_ACCTS_new', 'PRI_CU
plot_distribution_comp(var,nrow=3)
```

```
<Figure size 640x480 with 0 Axes>
```


1e10
Density plot
PRI_NO_OF_ACCTS_new
1e-7+5.435699353e4
Density plot
PRI_ACTIVE_ACCTS
1e10
Density plot
PRI_OVERDUE_ACCTS_new
1e-7+5.435699353e4
1e-6
Density plot
PRI_CURRENT_BALANCE
1e-7
Density plot
1e-7
Density plot

In…
```
var = ['SEC_NO_OF_ACCTS', 'SEC_ACTIVE_ACCTS', 'SEC_OVERDUE_ACCTS', 'SEC_CURRENT_B
plot_distribution_comp(var,nrow=3)
```

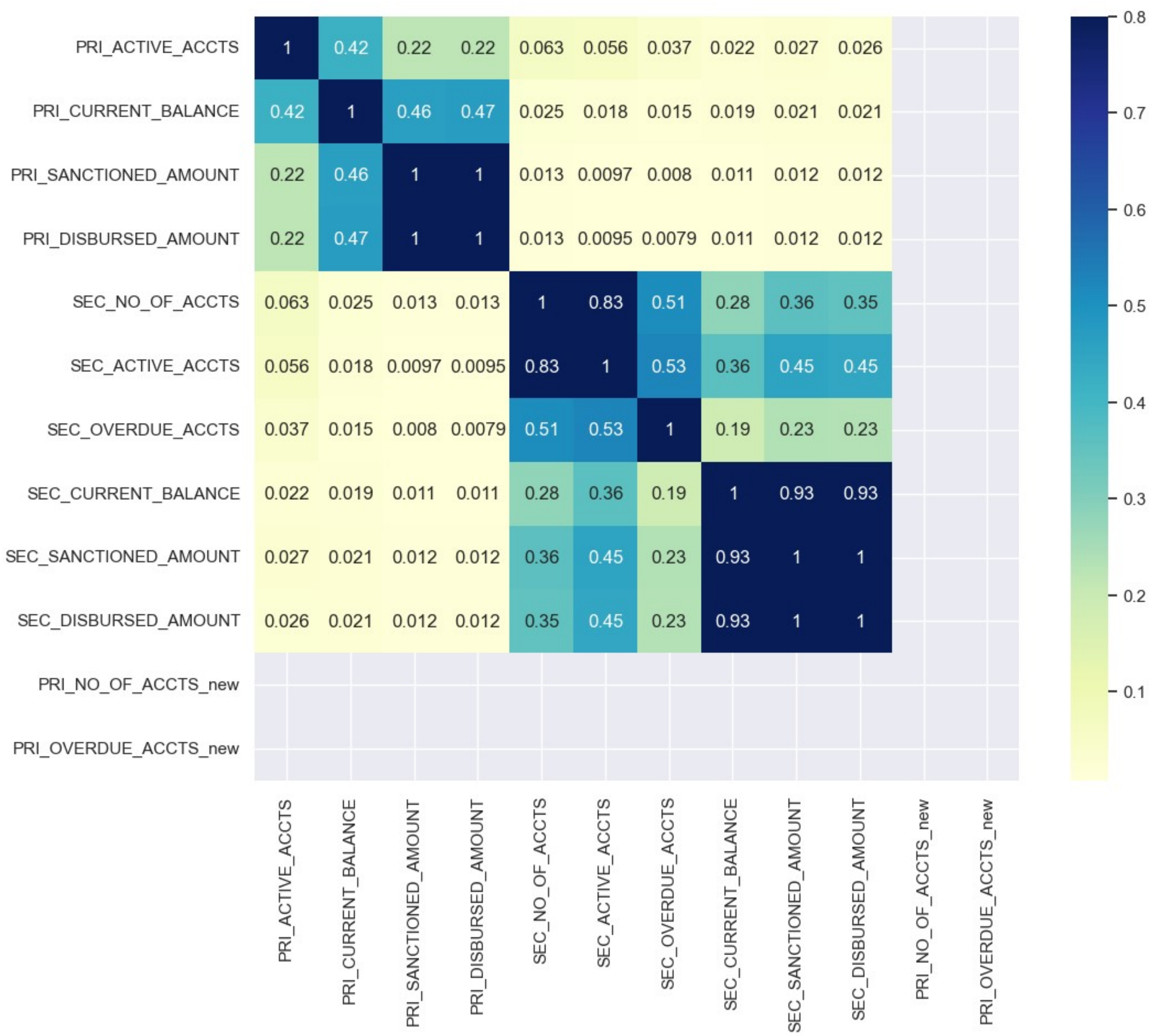
```
<Figure size 640x480 with 0 Axes>
```


Density plot
SEC_NO_OF_ACCTS
SEC_ACTIVE_ACCTS
SEC_OVERDUE_ACCTS
SEC_CURRENT_BALANCE
1e-6

3.5 Feature Selection

I...
```
#Useless features
#train = train.drop(['DISBURSED_AMOUNT','ASSET_COST', 'LTV', 'PRI_NO_OF_ACCTS','PRI
train = train.drop([ 'DATE_OF_BIRTH' , 'STATE_ID', 'EMPLOYEE_CODE_ID', 'SUPPLIER_ID
```

In ...
```
#Highly Correlated
sns.set()

cols = train[['PRI_ACTIVE_ACCTS', 'PRI_CURRENT_BALANCE', 'PRI_SANCTIONED_AMOUNT',
              'SEC_NO_OF_ACCTS', 'SEC_ACTIVE_ACCTS', 'SEC_OVERDUE_ACCTS', 'SEC_CU
              'SEC_DISBURSED_AMOUNT',  'PRI_NO_OF_ACCTS_new', 'PRI_OVERDUE_ACCTS_
corr = cols.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corr, annot=True, vmax=.8, square=True,cmap = 'YlGnBu');
```



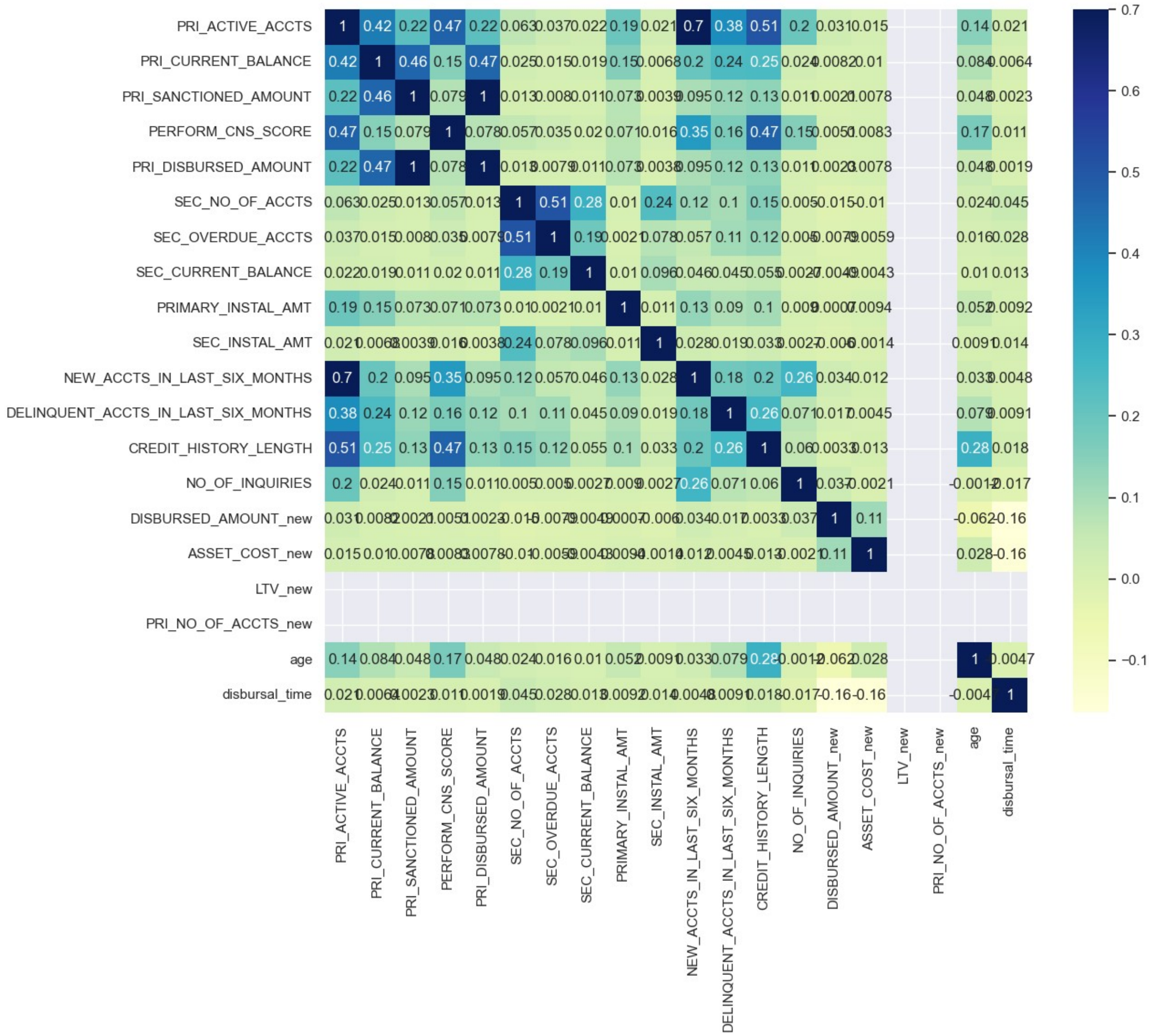
In ...
```
# train = train.drop(['PRI_OVERDUE_ACCTS_new','SEC_ACTIVE_ACCTS','SEC_SANCTIONED_
```

In ...
```
#Highly Correlated
sns.set()

cols = train[['SEC_INSTAL_AMT', 'PERFORM_CNS_SCORE','NEW_ACCTS_IN_LAST_SIX_MONTHS
              'AVERAGE_ACCT_AGE', 'CREDIT_HISTORY_LENGTH', 'NO_OF_INQUIRIES','age
corr = cols.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corr, annot=True, vmax=.8, square=True,cmap = 'YlGnBu');
```

'AVERAGE_ACCT_AGE', 'CREDIT_HISTORY_LENGTH'are highly positively correlated and hence keeping one

In [74]:
```
#train = train.drop(['AVERAGE_ACCT_AGE' ],axis=1)
```

In [...
```
#Highly Correlated
sns.set()

cols = train[['PRI_ACTIVE_ACCTS', 'PRI_CURRENT_BALANCE', 'PRI_SANCTIONED_AMOUNT',
```



One out of 'PRI_SANCTIONED_AMOUNT', 'PRI_DISBURSED_AMOUNT'
One out of 'LTV_new', 'PRI_NO_OF_ACCTS_new'
Eliminate 'NEW_ACCTS_IN_LAST_SIX_MONTHS'

In [...
```
#train = train.drop(['PRI_SANCTIONED_AMOUNT','PRI_NO_OF_ACCTS_new','NEW_ACCTS_IN_
```

Preparing Datasets 1) Binned Variables 2) Continous variables

In [...
```
train_con = train[['EMPLOYMENT_TYPE', 'MOBILENO_AVL_FLAG', 'AADHAR_FLAG', 'PAN_FI
                   'DRIVING_FLAG', 'PASSPORT_FLAG', 'PERFORM_CNS_SCORE', 'PERFORM
                   'PRI_ACTIVE_ACCTS', 'PRI_CURRENT_BALANCE', 'PRI_DISBURSED_AMOU
                   'SEC_OVERDUE_ACCTS', 'SEC_CURRENT_BALANCE', 'PRIMARY_INSTAL_AM
                   'DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS', 'CREDIT_HISTORY_LENGTH
                   'LOAN_DEFAULT', 'DISBURSED_AMOUNT_new', 'ASSET_COST_new',
                   'LTV_new', 'age', 'disbursal_time']]
```

In ...
```
train_bin = train [['UNIQUEID', 'EMPLOYMENT_TYPE', 'MOBILENO_AVL_FLAG', 'AADHAR_F
                    'VOTERID_FLAG', 'DRIVING_FLAG', 'PASSPORT_FLAG', 'PERFORM_CNS
                    'PERFORM_CNS_SCORE_DESCRIPTION', 'PRI_ACTIVE_ACCTS', 'PRI_CUR
                    'PRI_DISBURSED_AMOUNT', 'SEC_NO_OF_ACCTS', 'SEC_OVERDUE_ACCTS
```

In [79]:
```
# Confusion Matrix

def plot_confusion_matrix() :
    sns.heatmap(cm,annot=True,fmt='g',cmap=plt.cm.Blues)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show
```

In [8...
```
# Precision, Recall, F1 Score
def show_metrics():
    tp = cm[1,1]
    fn = cm[1,0]
    fp = cm[0,1]
    tn = cm[0,0]
    print('Precision =     {:.3f}'.format(tp/(tp+fp)))
    print('Recall    =     {:.3f}'.format(tp/(tp+fn)))
    print('F1_score  =     {:.3f}'.format(2*(((tp/(tp+fp))*(tp/(tp+fn)))/
                                              ((tp/(tp+fp))+(tp/(tp+fn))))))
```

In [80]:
```
# ROC curve
def plot_roc(y_test, logpred):
    roc_auc = metrics.roc_auc_score(y_test, logpred)
    fpr, tpr, thresholds = metrics.roc_curve(y_test, logpred)
    plt.figure()

    plt.plot(fpr, tpr, label = 'ROC curve', color ='orange', linewidth = 2)
    plt.plot([0,1],[0,1], 'k--', linewidth = 2)
    plt.xlim([0.0,1.0])
    plt.ylim([0.0,1.0])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.show();
```

3.5.1 Standardization of data

In [...
```
scaler_data = StandardScaler()
def scaleColumns(df, cols_to_scale):

    for col in cols_to_scale:

        df[col] = pd.DataFrame(scaler_data.fit_transform(pd.DataFrame(train_con[

    return df
```

In [...
```
scaled_df = scaleColumns(train_con,['PERFORM_CNS_SCORE','PRI_ACTIVE_ACCTS', 'PRI
                                   'PRI_DISBURSED_AMOUNT', 'SEC_NO_OF_ACCTS', 'SEC
                                   'SEC_CURRENT_BALANCE', 'PRIMARY_INSTAL_AMT', 'S
                                   'DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS', 'CREDIT_
                                   'NO_OF_INQUIRIES', 'DISBURSED_AMOUNT_new',
                                   'ASSET_COST_new', 'LTV_new', 'age', 'disbursal_

scaled_df.head()
```

Out[82]:

	EMPLOYMENT_TYPE	MOBILENO_AVL_FLAG	AADHAR_FLAG	PAN_FLAG	VOTERID_FLAG	DRIVING_
0	Salaried	1	1	0	0	
1	Self employed	1	1	0	0	
2	Self employed	1	1	0	0	
3	Self employed	1	1	0	0	
4	Self employed	1	1	0	0	

3.5.2 Dummy insertion

In [83]:
```
train_dummy = pd.get_dummies(scaled_df, prefix_sep='_', drop_first=True)
train_dummy.head()
```

Out[83]:

	MOBILENO_AVL_FLAG	AADHAR_FLAG	PAN_FLAG	VOTERID_FLAG	DRIVING_FLAG	PASSPORT_FLA
0	1	1	0	0	0	
1	1	1	0	0	0	
2	1	1	0	0	0	
3	1	1	0	0	0	
4	1	1	0	0	0	

In [84]:
```
y = train_dummy[['LOAN_DEFAULT']]
X= train_dummy.loc[:, train_dummy.columns != 'LOAN_DEFAULT']
X.shape
```

Out[84]:
```
(233154, 44)
```

In [85]:
```
np.any(np.isnan(X))
```

Out[85]:
```
False
```

In [86]:
```
X = X.fillna(0)
X.shape
```

Out[86]:
```
(233154, 44)
```

In [...
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, rand
```

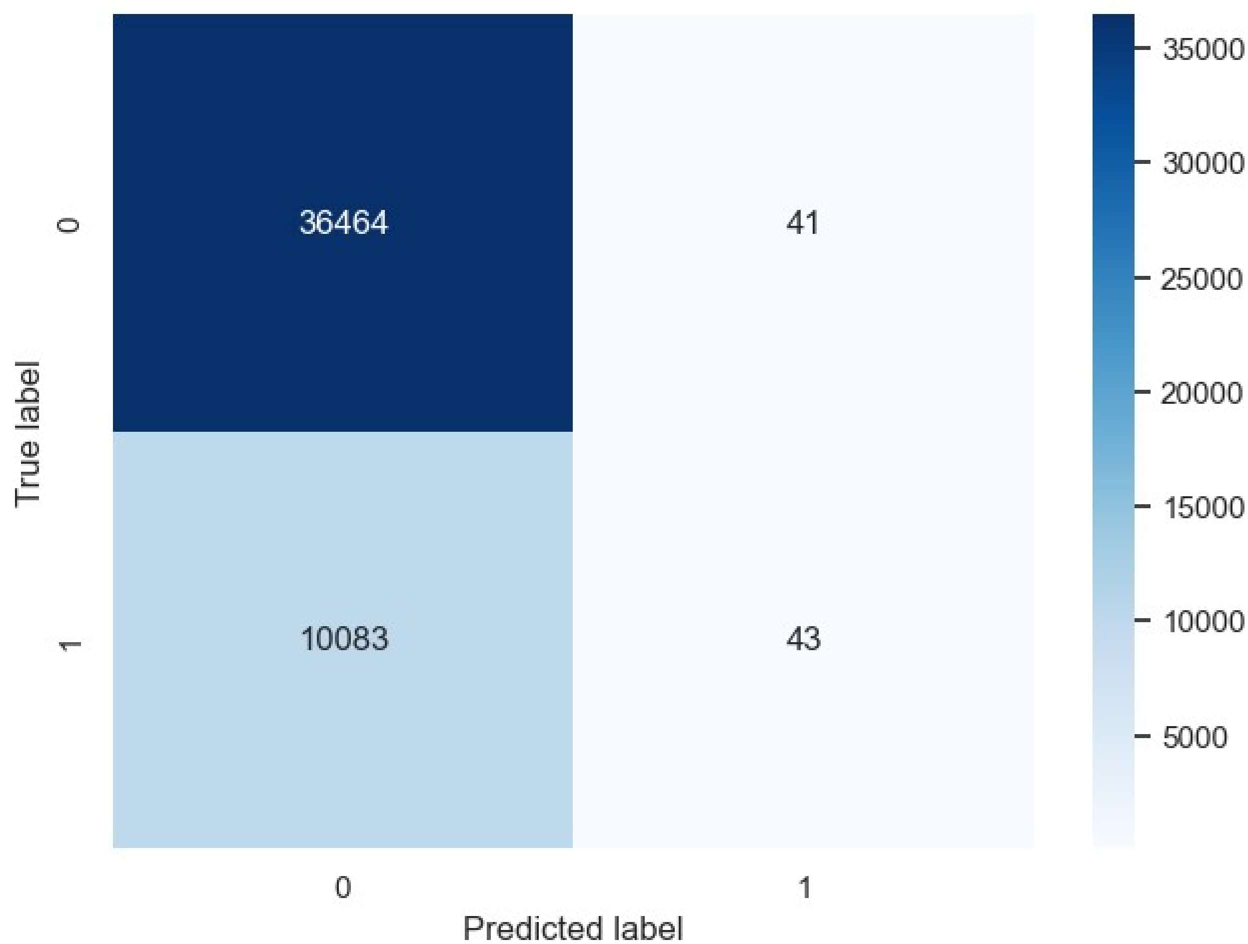
4. Base Line Models

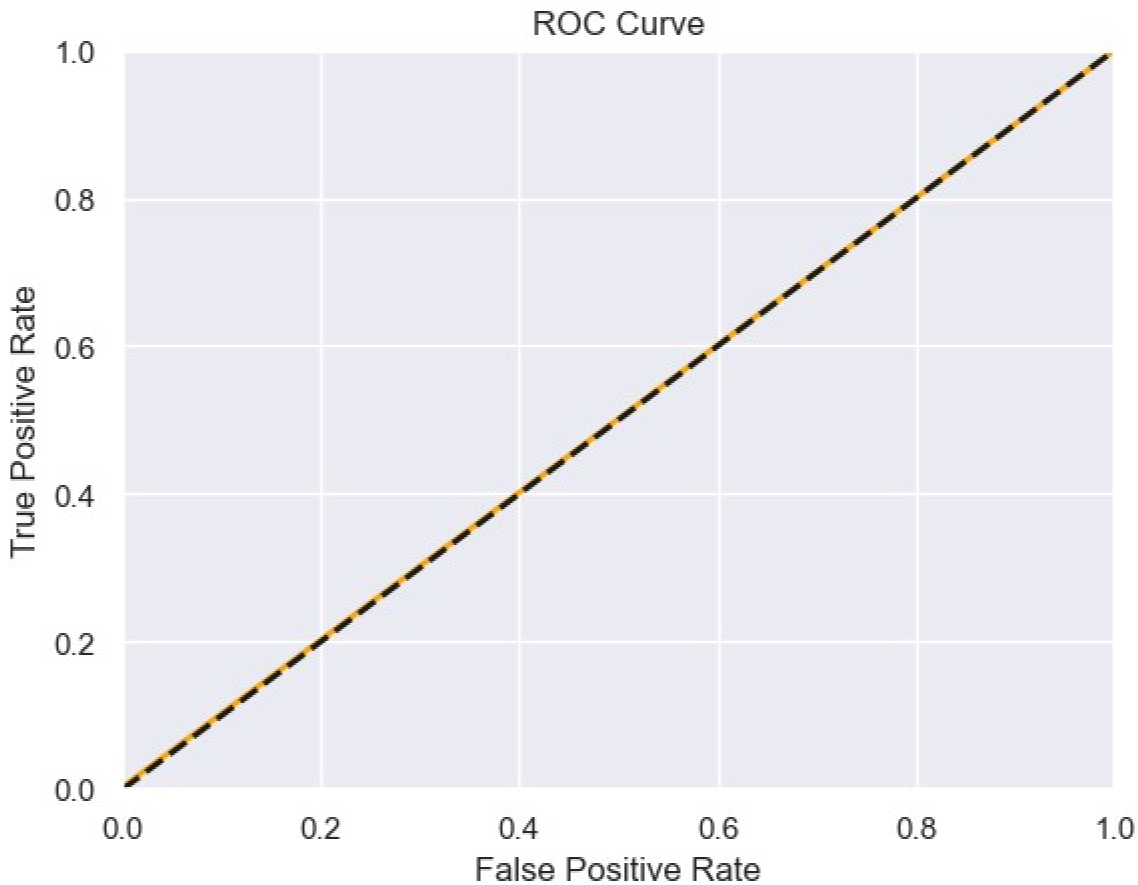
4.1 Logistic Regression

In [...

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
logpred = logmodel.predict(X_test)


print(confusion_matrix(y_test, logpred))
print(round(accuracy_score(y_test, logpred),2)*100)
LOGCV = (cross_val_score(logmodel, X_train, y_train, cv=k_fold, n_jobs=1, scorin
cm=metrics.confusion_matrix(y_test, logpred)
show_metrics()

plot_confusion_matrix()
```

```
[[36464    41]
 [10083    43]]
78.0
Precision =     0.512
Recall    =     0.004
F1_score  =     0.008
```





In [92]:
```
from sklearn.metrics import f1_score
from sklearn.metrics import balanced_accuracy_score
print("Accuracy of model ",accuracy_score(y_test, logpred))
print("F1 Score ",f1_score(y_test, logpred))
print("Recall Score ",recall_score(y_test, logpred))
print("AUC Score ",metrics.roc_auc_score(y_test,logpred))
print("Balanced Accuracy Score ",balanced_accuracy_score(y_test, logpred))
```

```
Accuracy of model  0.7828912097102786
F1 Score  0.00842311459353575
Recall Score  0.004246494173414972
AUC Score  0.5015616801780649
Balanced Accuracy Score  0.5015616801780648
```

Accuracy score is good, however the model is not predicting the Defaults well

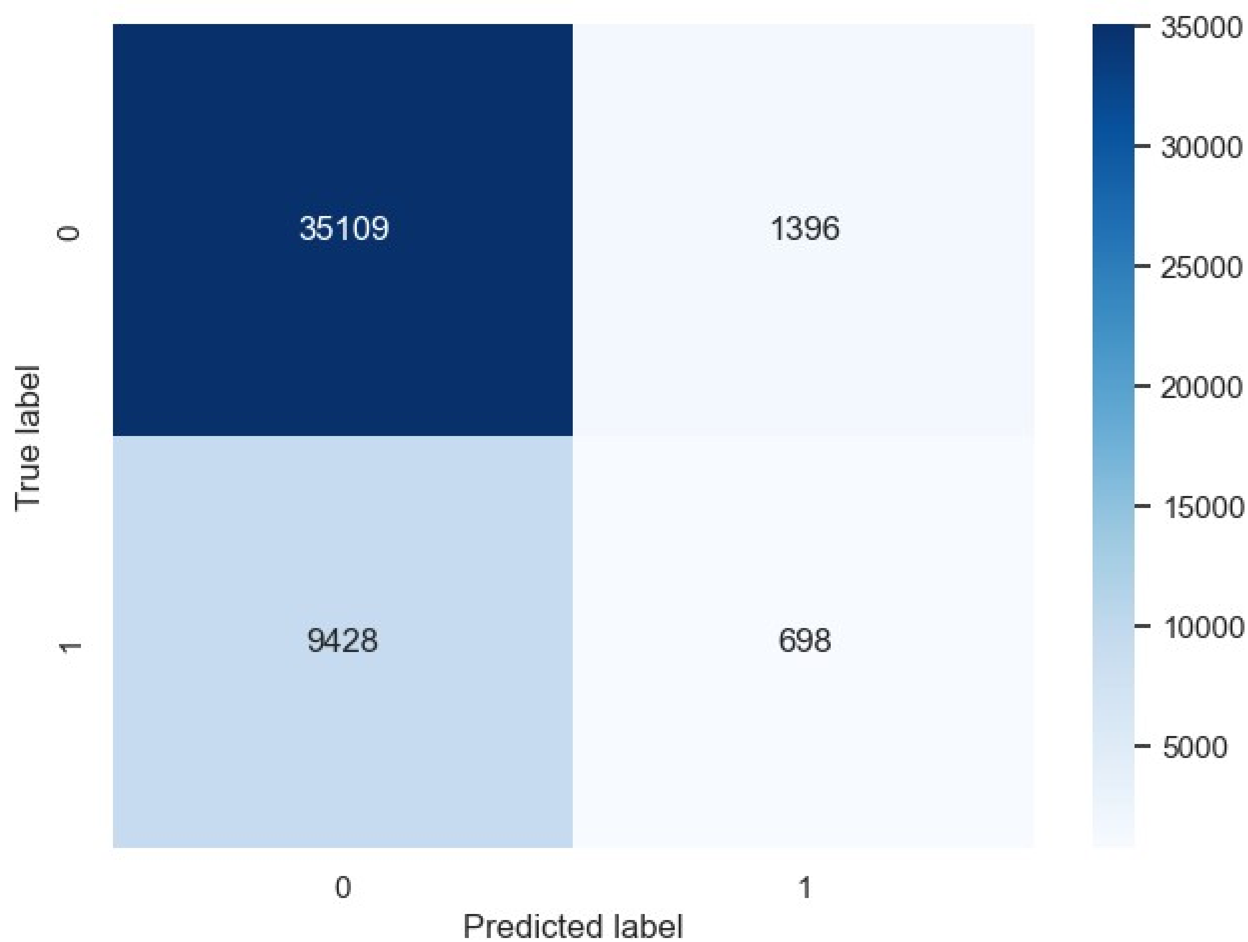
4.2 Random Forest

In [...
```
from sklearn.ensemble import RandomForestClassifier

# train model
rfc = RandomForestClassifier(n_estimators=10).fit(X_train, y_train)

# predict on test set
rfc_pred = rfc.predict(X_test)
print(confusion_matrix(y_test, rfc_pred))
```

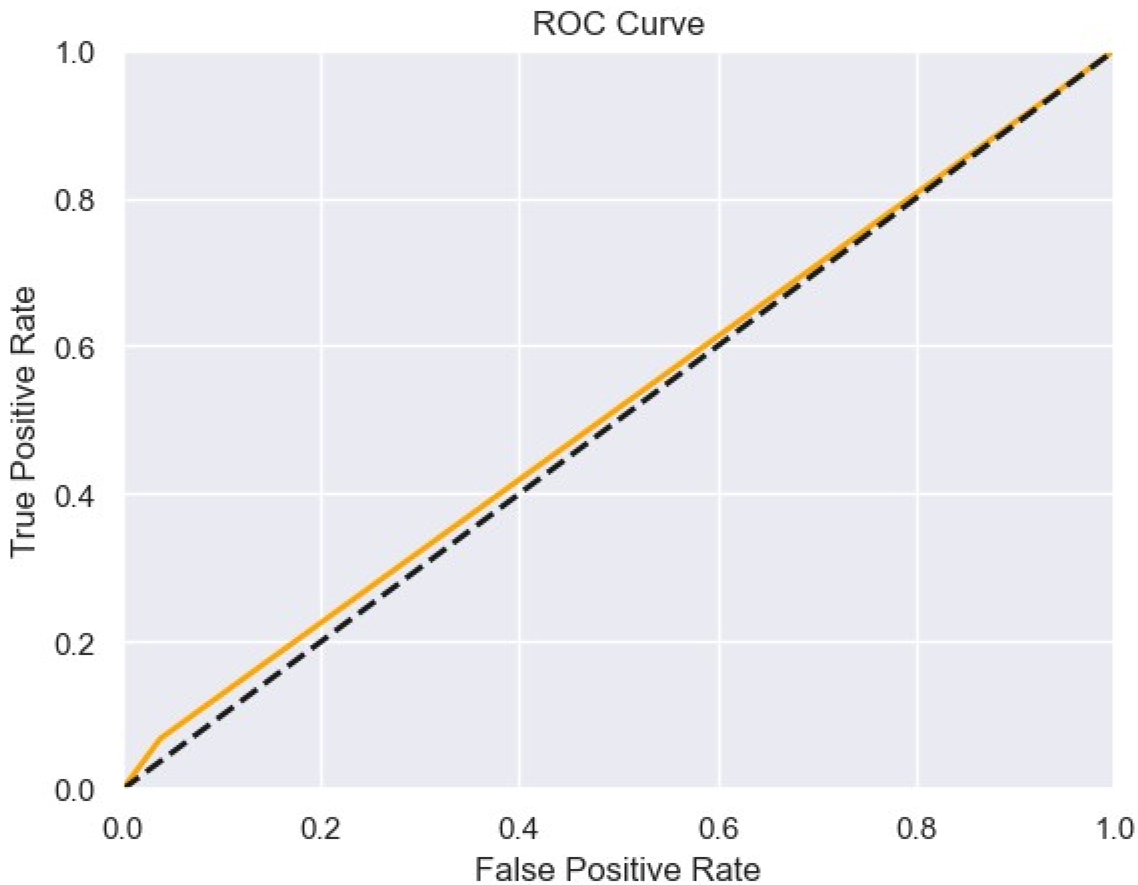
```
[[35109  1396]
 [ 9428   698]]
77.0
```

In [94]:
```
cm=metrics.confusion_matrix(y_test, rfc_pred)
show_metrics()

plot_confusion_matrix()
```

```
Precision =     0.333
Recall    =     0.069
F1_score  =     0.114
```



In [95]:
```
plot_roc(y_test, rfc_pred)
```



In [96]:
```
from sklearn.metrics import f1_score
from sklearn.metrics import balanced_accuracy_score
print("Accuracy of model ",accuracy_score(y_test, rfc_pred))
print("F1 Score ",f1_score(y_test, rfc_pred))
print("Recall Score ",recall_score(y_test, rfc_pred))
print("AUC Score ",metrics.roc_auc_score(y_test,rfc_pred))
print("Balanced Accuracy Score ",balanced_accuracy_score(y_test, rfc_pred))
```

```
Accuracy of model  0.7678797366558727
F1 Score  0.11423895253682488
Recall Score  0.06893146355915465
AUC Score  0.5153450633779886
Balanced Accuracy Score  0.5153450633779885
```

Accuracy score is good, however the model is predicting the Defaults better than Logistic reg

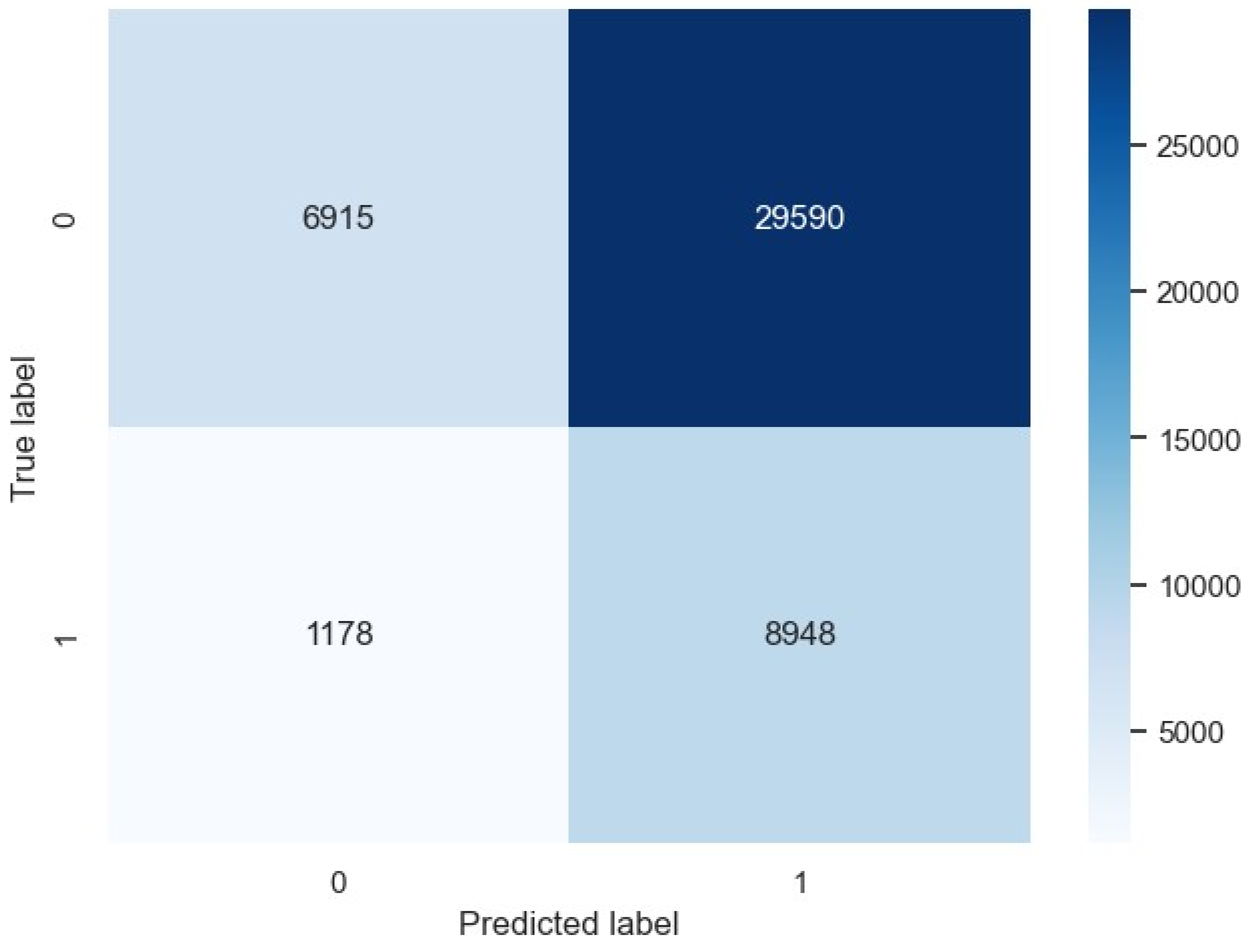
4.3 Naive Bayes

In [...
```
from sklearn.naive_bayes import GaussianNB

# train model
nb = GaussianNB().fit(X_train, y_train)

# predict on test set
nb_pred = nb.predict(X_test)
print(confusion_matrix(y_test, nb_pred))
```

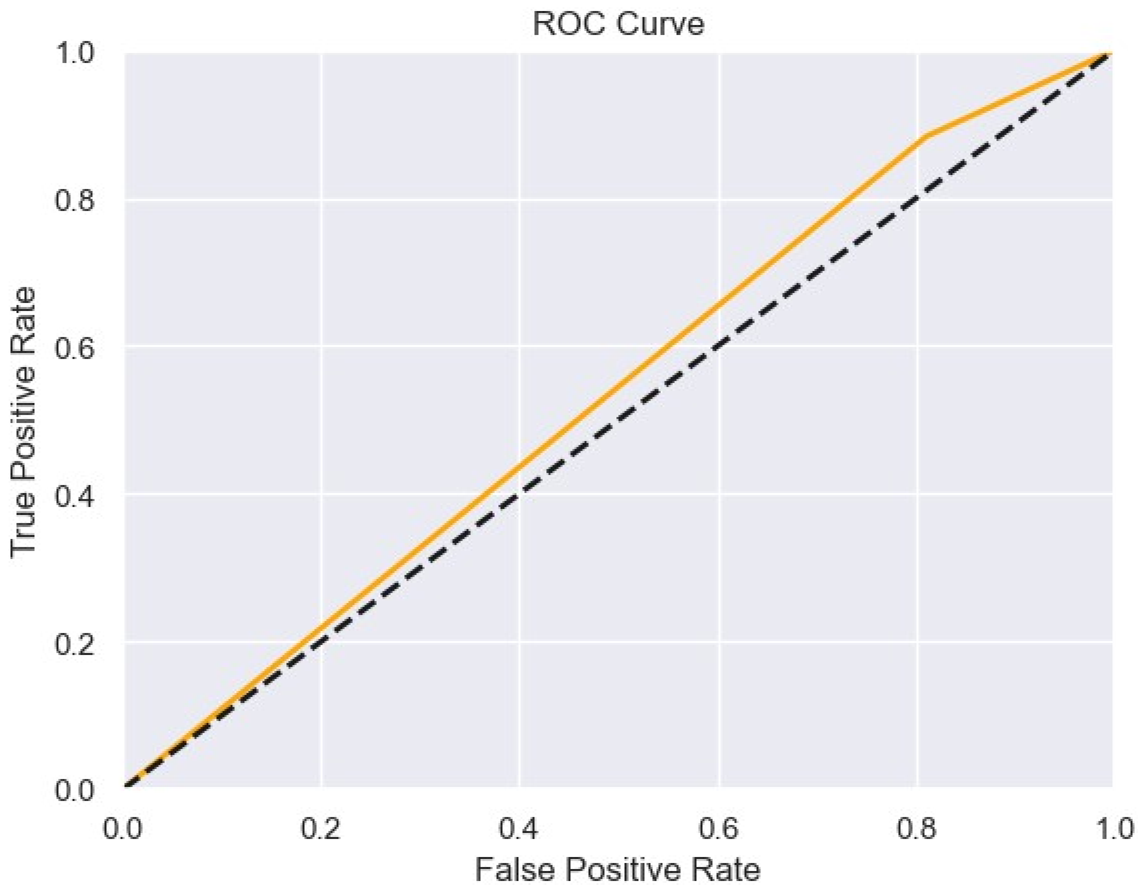
```
[[ 6915 29590]
 [ 1178  8948]]
34.0
```

In [98]:
```
cm=metrics.confusion_matrix(y_test, nb_pred)
show_metrics()

plot_confusion_matrix()
```

```
Precision =     0.232
Recall    =     0.884
F1_score  =     0.368
```



In [99]:
```
plot_roc(y_test, nb_pred)
```



In [100]:
```
from sklearn.metrics import f1_score
from sklearn.metrics import balanced_accuracy_score
print("Accuracy of model ",accuracy_score(y_test, nb_pred))
print("F1 Score ",f1_score(y_test, nb_pred))
print("Recall Score ",recall_score(y_test, nb_pred))
print("AUC Score ",metrics.roc_auc_score(y_test,nb_pred))
print("Balanced Accuracy Score ",balanced_accuracy_score(y_test, nb_pred))
```

```
Accuracy of model  0.34018142437434323
F1 Score  0.36774617787276015
Recall Score  0.8836658107841201
AUC Score  0.5365459583984975
Balanced Accuracy Score  0.5365459583984975
```

Model accuracy is very poor

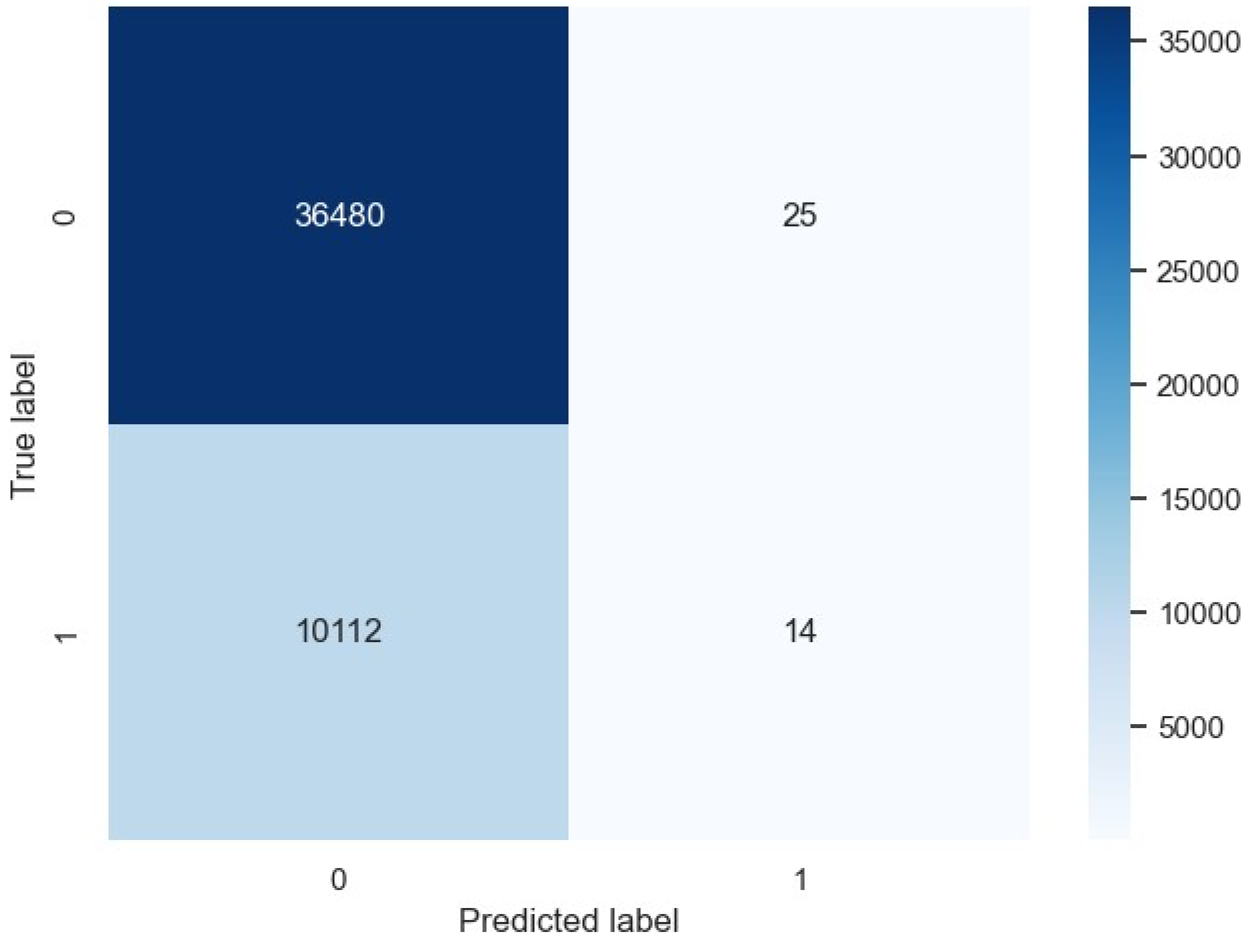
4.4 Stochastic Gradient Descent

In [1...
```
from sklearn.linear_model import SGDClassifier

# train model
sgd = SGDClassifier(loss= "modified_huber", shuffle = True, random_state= 101).f

# predict on test set
sgd_pred = sgd.predict(X_test)
print(confusion_matrix(y_test, sgd_pred))
```

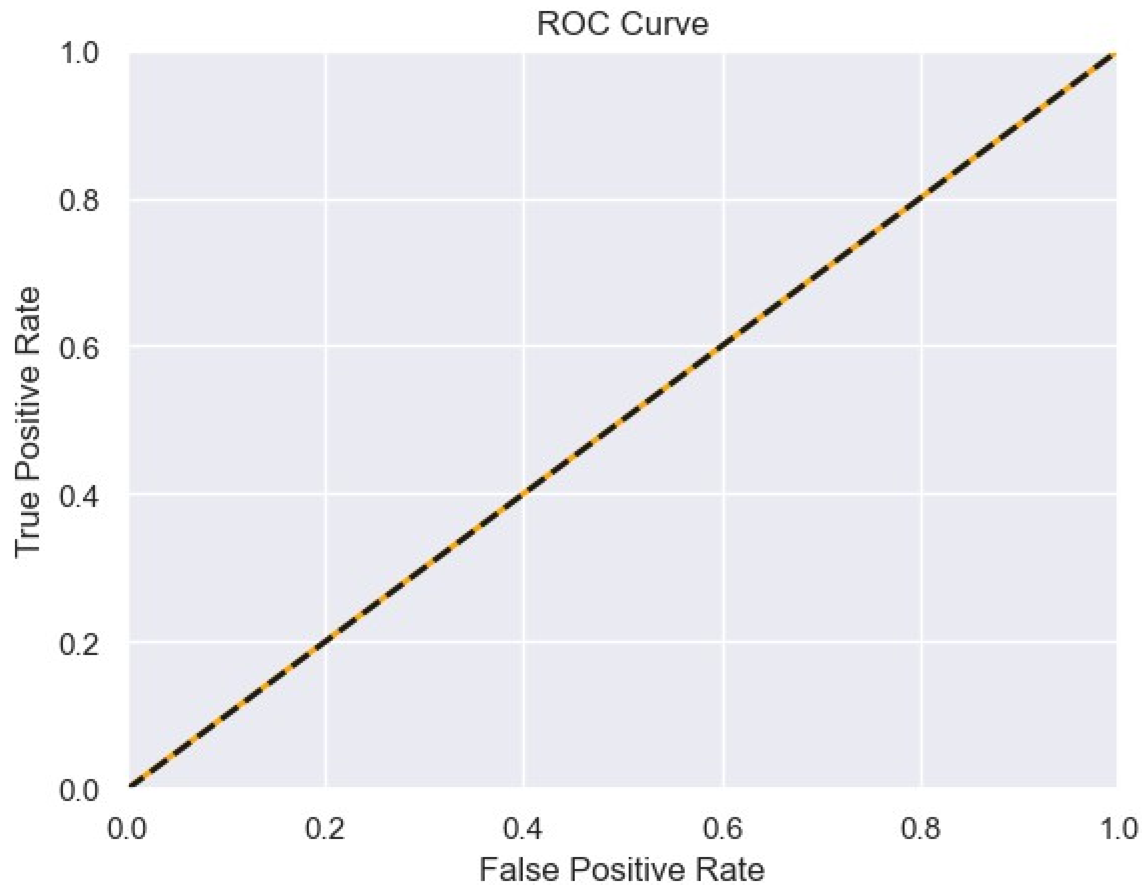
```
[[36480    25]
 [10112    14]]
78.0
```

In [102]:
```
cm=metrics.confusion_matrix(y_test, sgd_pred)
show_metrics()

plot_confusion_matrix()
```

```
Precision =     0.359
Recall    =     0.001
F1_score  =     0.003
```



In [103]:
```
plot_roc(y_test, sgd_pred)
```



In [104]:
```
from sklearn.metrics import f1_score
from sklearn.metrics import balanced_accuracy_score
print("Accuracy of model ",accuracy_score(y_test, sgd_pred))
print("F1 Score ",f1_score(y_test, sgd_pred))
print("Recall Score ",recall_score(y_test, sgd_pred))
print("AUC Score ",metrics.roc_auc_score(y_test,sgd_pred))
print("Balanced Accuracy Score ",balanced_accuracy_score(y_test, sgd_pred))
```

```
Accuracy of model  0.7826124252106967
F1 Score  0.002754549926217413
Recall Score  0.0013825794983211535
AUC Score  0.5003488709024272
Balanced Accuracy Score  0.5003488709024273
```

Accuracy score is good, however the model is not predicting the Defaults well

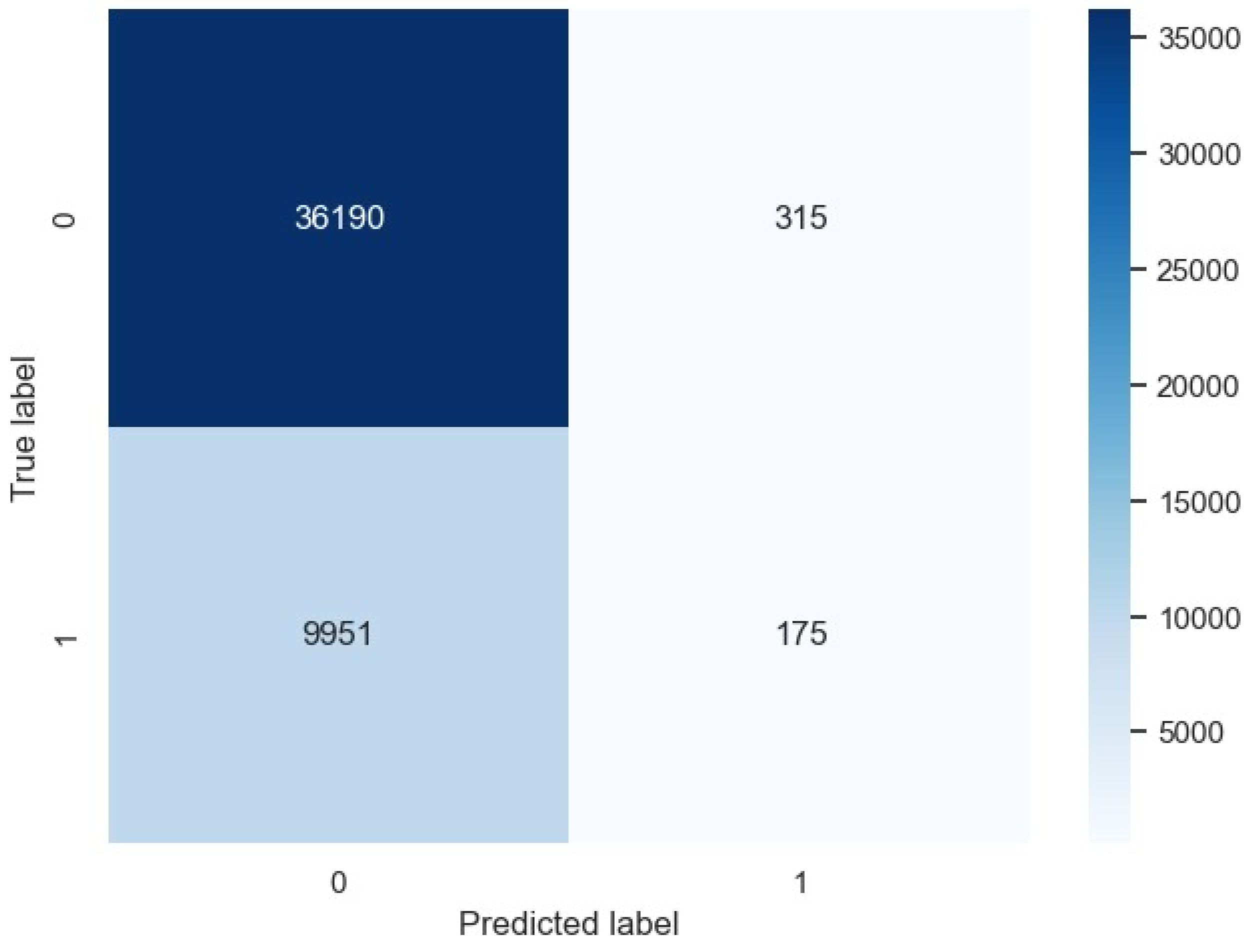
4.5 Decision Tree Classifier

In [...
```
from sklearn.tree import DecisionTreeClassifier

# train model
dtree = DecisionTreeClassifier(max_depth = 10, random_state= 101, max_features =N

# predict on test set
dtree_pred = dtree.predict(X_test)
print(confusion_matrix(y_test, dtree_pred))
```

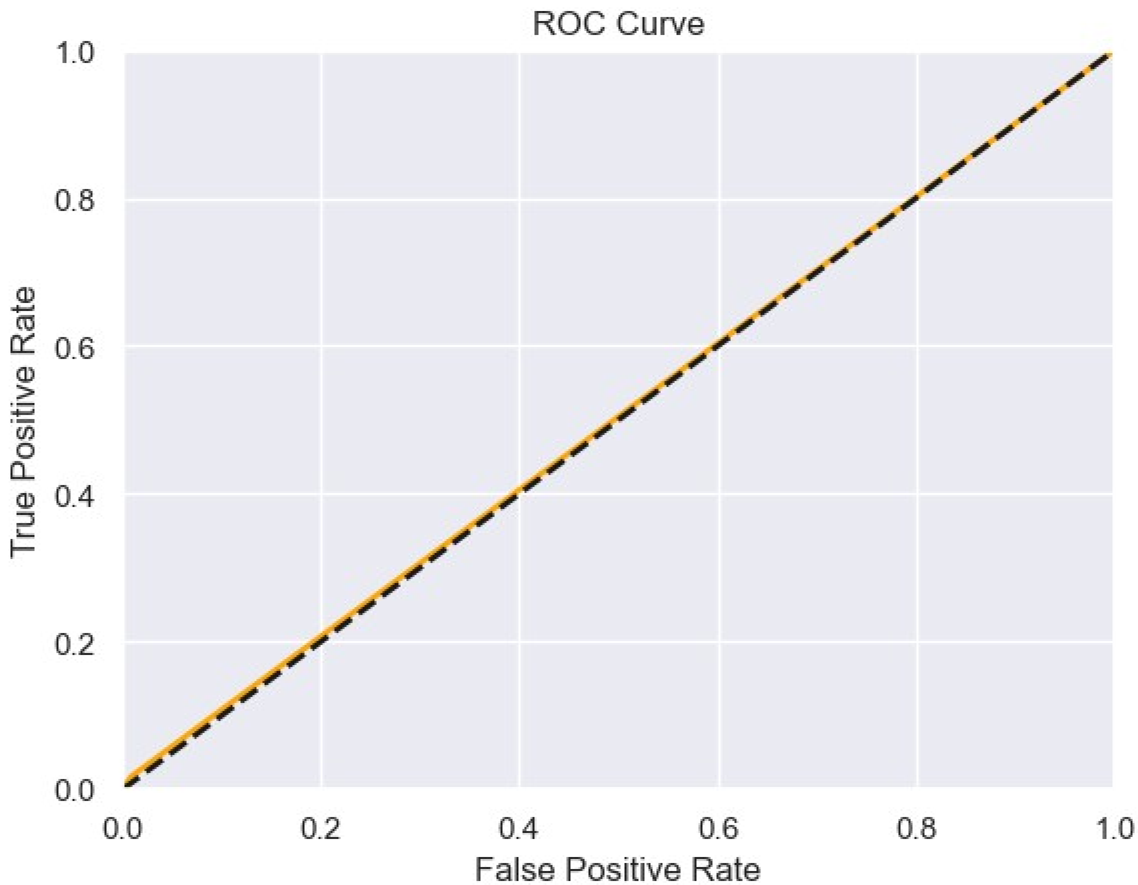
```
[[36190   315]
 [ 9951   175]]
78.0
```

In [106]:
```
cm=metrics.confusion_matrix(y_test, dtree_pred)
show_metrics()

plot_confusion_matrix()
```

```
Precision =     0.357
Recall    =     0.017
F1_score  =     0.033
```



In [107]:
```
plot_roc(y_test, dtree_pred)
```



In [108]:
```
from sklearn.metrics import f1_score
from sklearn.metrics import balanced_accuracy_score
print("Accuracy of model ",accuracy_score(y_test, dtree_pred))
print("F1 Score ",f1_score(y_test, dtree_pred))
print("Recall Score ",recall_score(y_test, dtree_pred))
print("AUC Score ",metrics.roc_auc_score(y_test,dtree_pred))
print("Balanced Accuracy Score ",balanced_accuracy_score(y_test, dtree_pred))
```

```
Accuracy of model  0.7798460251763848
F1 Score  0.03296910324039186
Recall Score  0.017282243729014417
AUC Score  0.5043266443956673
Balanced Accuracy Score  0.5043266443956673
```

Accuracy score is good, however the model is not predicting the Defaults well

Best model is Random Forest till now

5. Dealing with Imbalanced data

5.1 SMOTE

SMOTE or Synthetic Minority Oversampling Technique is used to create synthetic data. SMOTE uses a nearest neighbors algorithm to generate new and synthetic data we can use for training our model.

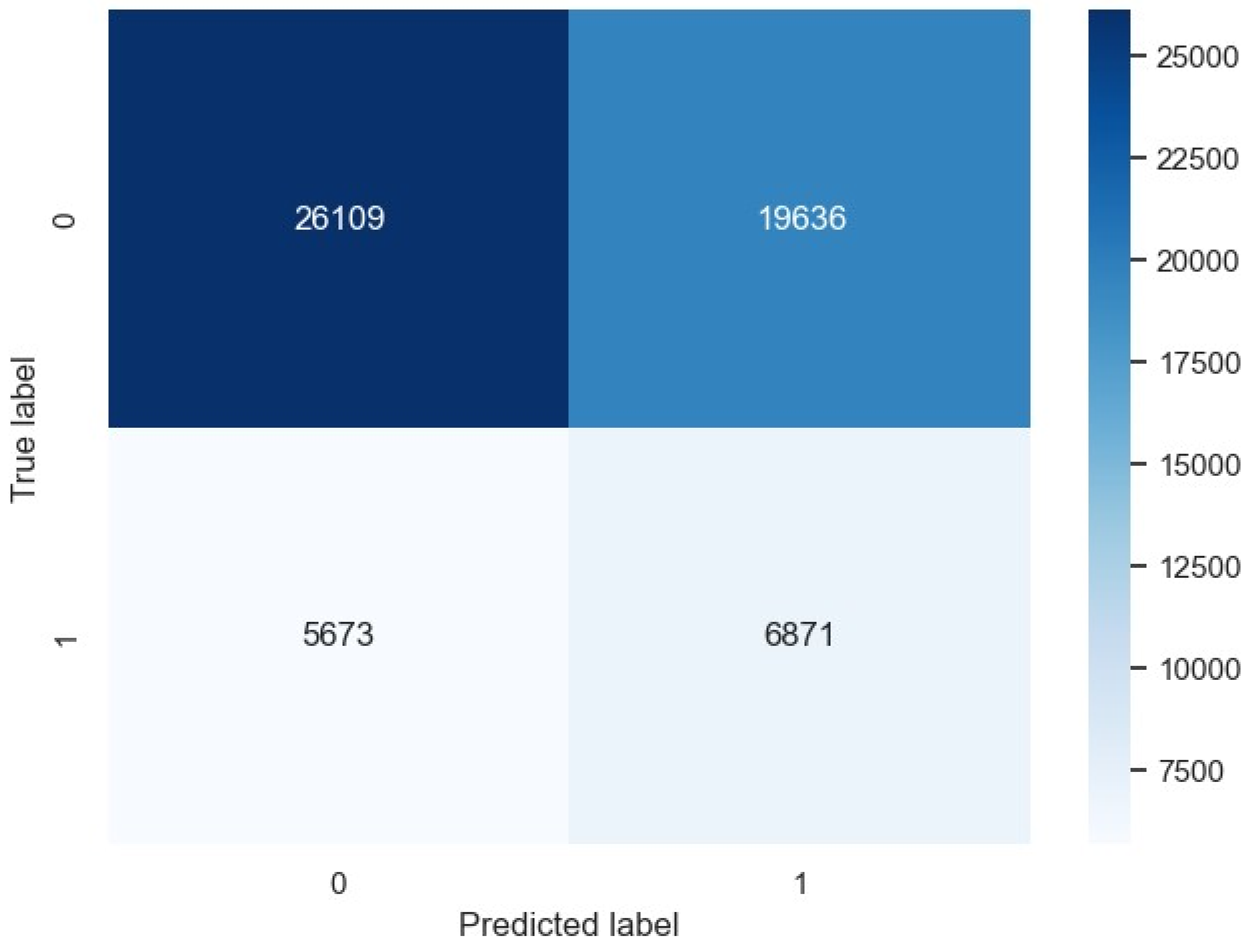
In [1...
```
from imblearn.over_sampling import SMOTE
```

In [1...
```
from sklearn.linear_model import SGDClassifier

# train model
sgd = SGDClassifier(loss= "modified_huber", shuffle = True, random_state= 101).f

# predict on test set
sgd_pred = sgd.predict(X_test)
print(confusion_matrix(y_test, sgd_pred))
print(round(accuracy_score(y_test, sgd_pred),2)*100)
LOGCV = (cross_val_score(logmodel, X_train, y_train, cv=k_fold, n_jobs=1, scorin
```

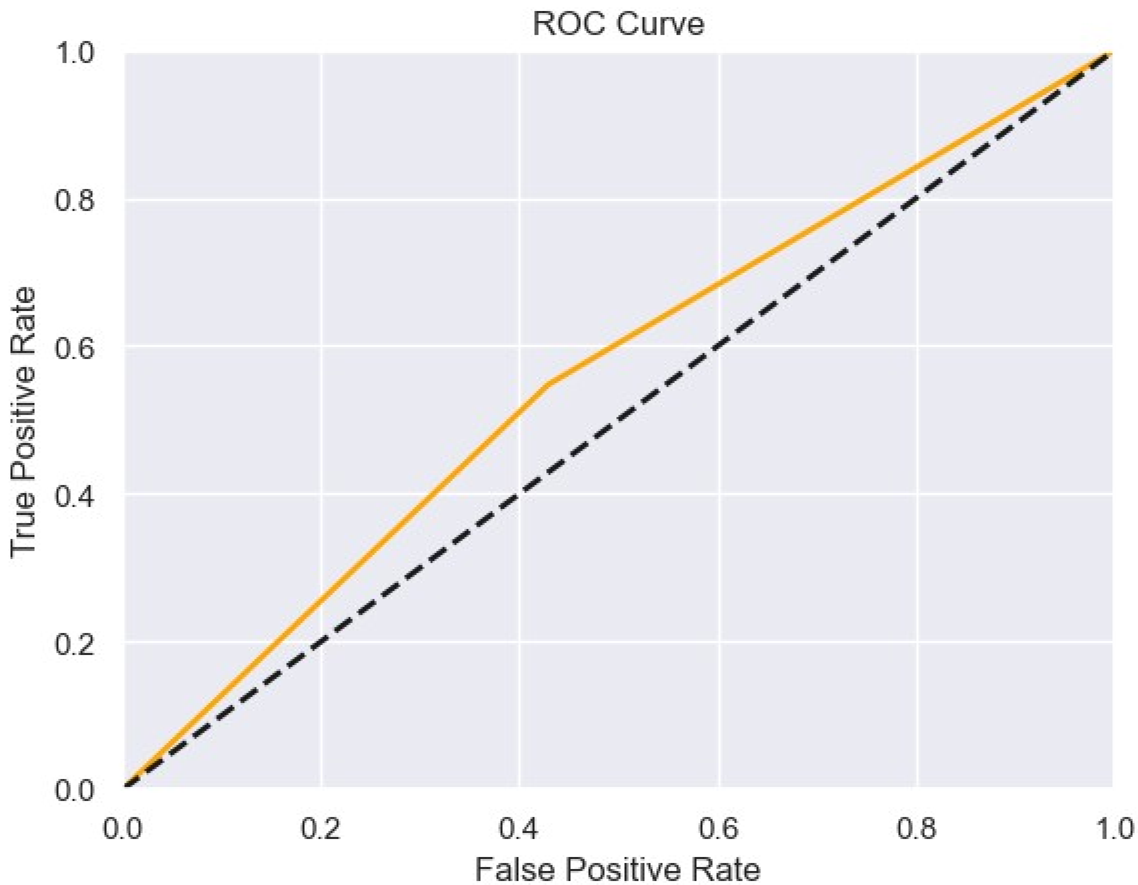
```
[[26109 19636]
 [ 5673  6871]]
56.99999999999999
```

In [111]:
```
cm=metrics.confusion_matrix(y_test, sgd_pred)
show_metrics()

plot_confusion_matrix()
```

```
Precision =     0.259
Recall    =     0.548
F1_score  =     0.352
```



In [112]:
```
plot_roc(y_test, sgd_pred)
```



In [...

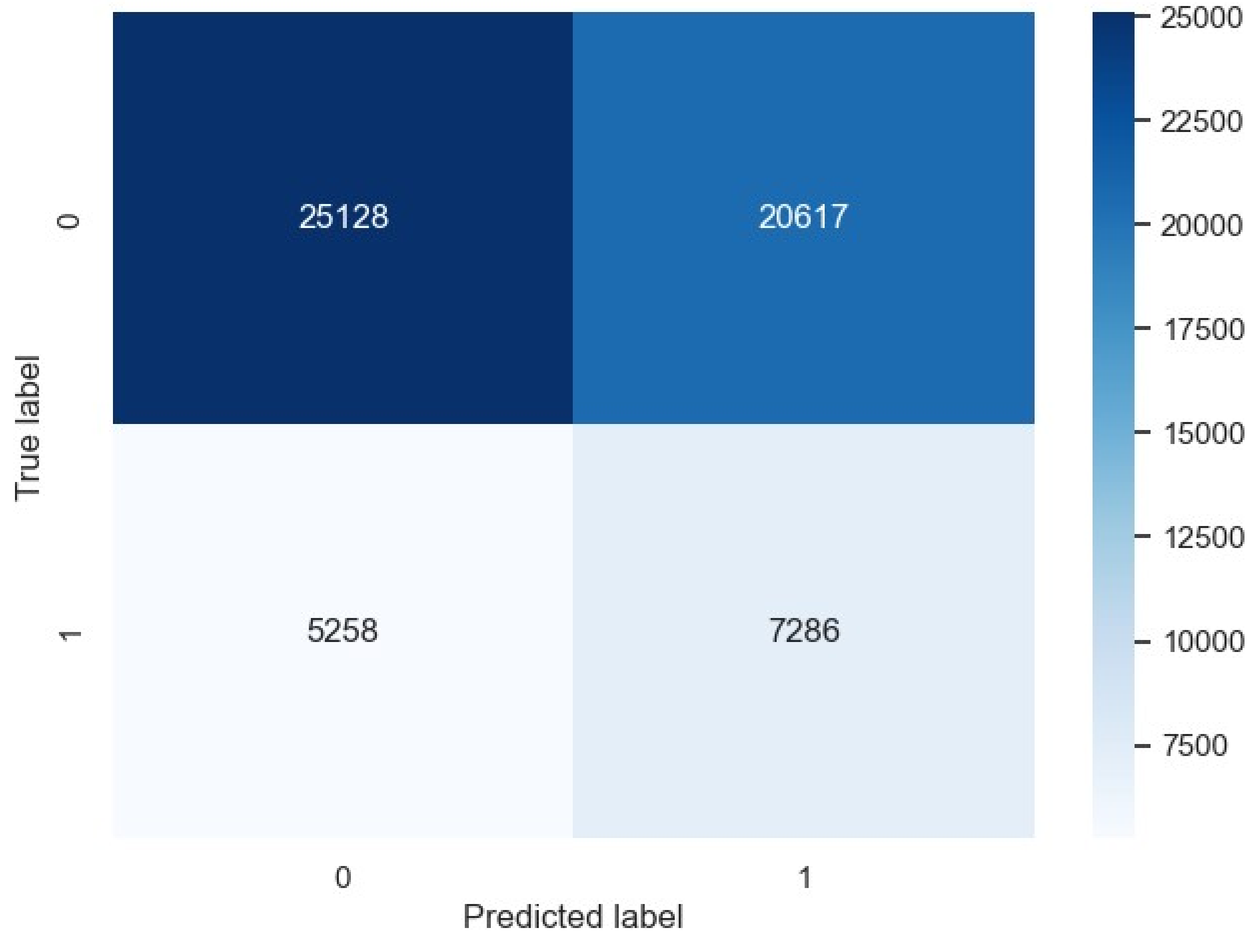
```
from sklearn.tree import DecisionTreeClassifier

# train model
dtree = DecisionTreeClassifier(max_depth = 10, random_state= 101, max_features =N

# predict on test set
dtree_pred = dtree.predict(X_test)
print(confusion_matrix(y_test, dtree_pred))
print(round(accuracy_score(y_test, dtree_pred),2)*100)
LOGCV = (cross_val_score(logmodel, X_train, y_train, cv=k_fold, n_jobs=1, scoring
```

```
[[25128 20617]
 [ 5258  7286]]
56.00000000000001
```

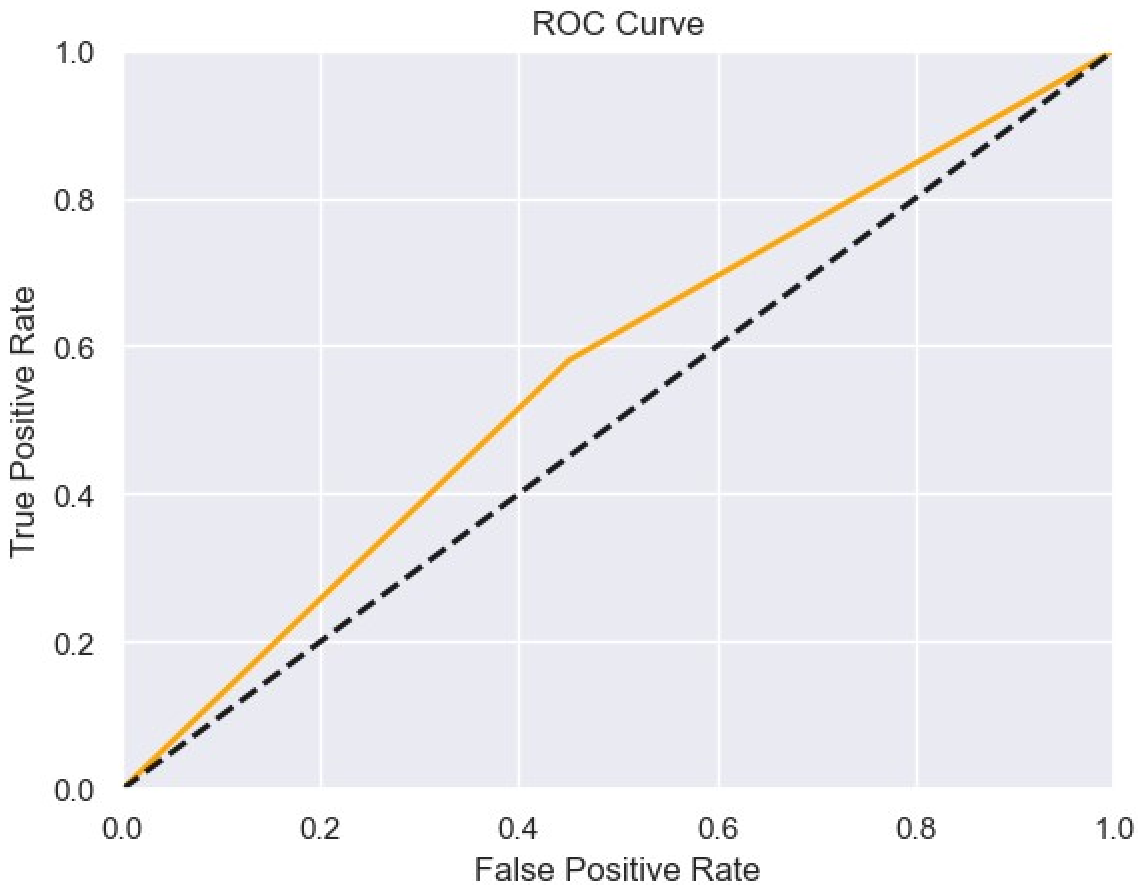
In [114]:

```
cm=metrics.confusion_matrix(y_test, dtree_pred)
show_metrics()

plot_confusion_matrix()
```

```
Precision =     0.261
Recall    =     0.581
F1_score  =     0.360
```



In [115]:
```
plot_roc(y_test, dtree_pred)
```



In [116]:
```
pip install eli5
```

```
Requirement already satisfied: eli5 in c:\users\sashi_000\anaconda3\lib\site-packages
(0.13.0)
Requirement already satisfied: jinja2>=3.0.0 in c:\users\sashi_000\anaconda3\lib\sit
e-packages (from eli5) (3.1.2)
Requirement already satisfied: tabulate>=0.7.7 in c:\users\sashi_000\anaconda3\lib\si
te-packages (from eli5) (0.8.10)
Requirement already satisfied: attrs>17.1.0 in c:\users\sashi_000\anaconda3\lib\site-
packages (from eli5) (22.1.0)
Requirement already satisfied: scikit-learn>=0.20 in c:\users\sashi_000\anaconda3\lib
\site-packages (from eli5) (1.2.1)
Requirement already satisfied: graphviz in c:\users\sashi_000\anaconda3\lib\site-pack
ages (from eli5) (0.20.1)
Requirement already satisfied: six in c:\users\sashi_000\anaconda3\lib\site-packages
(from eli5) (1.16.0)
Requirement already satisfied: numpy>=1.9.0 in c:\users\sashi_000\anaconda3\lib\site-
packages (from eli5) (1.25.0)
Requirement already satisfied: scipy in c:\users\sashi_000\anaconda3\lib\site-package
s (from eli5) (1.10.1)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\sashi_000\anaconda3\lib\si
te-packages (from jinja2>=3.0.0->eli5) (2.1.1)
Requirement already satisfied: joblib>=1.1.1 in c:\users\sashi_000\anaconda3\lib\sit
e-packages (from scikit-learn>=0.20->eli5) (1.1.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\sashi_000\anaconda3\l
ib\site-packages (from scikit-learn>=0.20->eli5) (3.1.0)
Note: you may need to restart the kernel to use updated packages.
```

In [117]:

```python
from sklearn.datasets import make_classification
import eli5
from eli5.sklearn import PermutationImportance

perm = PermutationImportance(rfc, random_state=1).fit(X_test, y_test)
eli5.show_weights(perm, feature_names = X_test.columns.tolist())
```

Out[117]:

Weight	Feature
0.1295 ± 0.0031	DISBURSED_AMOUNT_new
0.1172 ± 0.0015	disbursal_time
0.1014 ± 0.0011	ASSET_COST_new
0.1012 ± 0.0013	age
0.0652 ± 0.0010	PERFORM_CNS_SCORE
0.0587 ± 0.0017	CREDIT_HISTORY_LENGTH
0.0471 ± 0.0011	PRI_DISBURSED_AMOUNT
0.0454 ± 0.0010	PRI_CURRENT_BALANCE
0.0438 ± 0.0010	EMPLOYMENT_TYPE_Self employed
0.0424 ± 0.0009	EMPLOYMENT_TYPE_Salaried
0.0356 ± 0.0009	VOTERID_FLAG
0.0353 ± 0.0011	PRIMARY_INSTAL_AMT
0.0339 ± 0.0013	AADHAR_FLAG
0.0304 ± 0.0004	NO_OF_INQUIRIES
0.0273 ± 0.0008	PRI_ACTIVE_ACCTS
0.0271 ± 0.0012	PERFORM_CNS_SCORE_DESCRIPTION_No Bureau History Available
0.0162 ± 0.0012	DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS
0.0139 ± 0.0003	PAN_FLAG
0.0080 ± 0.0005	PERFORM_CNS_SCORE_DESCRIPTION_M-Very High Risk
0.0060 ± 0.0001	PERFORM_CNS_SCORE_DESCRIPTION_C-Very Low Risk
	… 24 more …

```
Accuracy of model  0.5560912007411347
F1 Score  0.36027393873464036
Recall Score  0.5808354591836735
AUC Score  0.5650706971292726
Balanced Accuracy Score  0.5650706971292725
```

In [1...

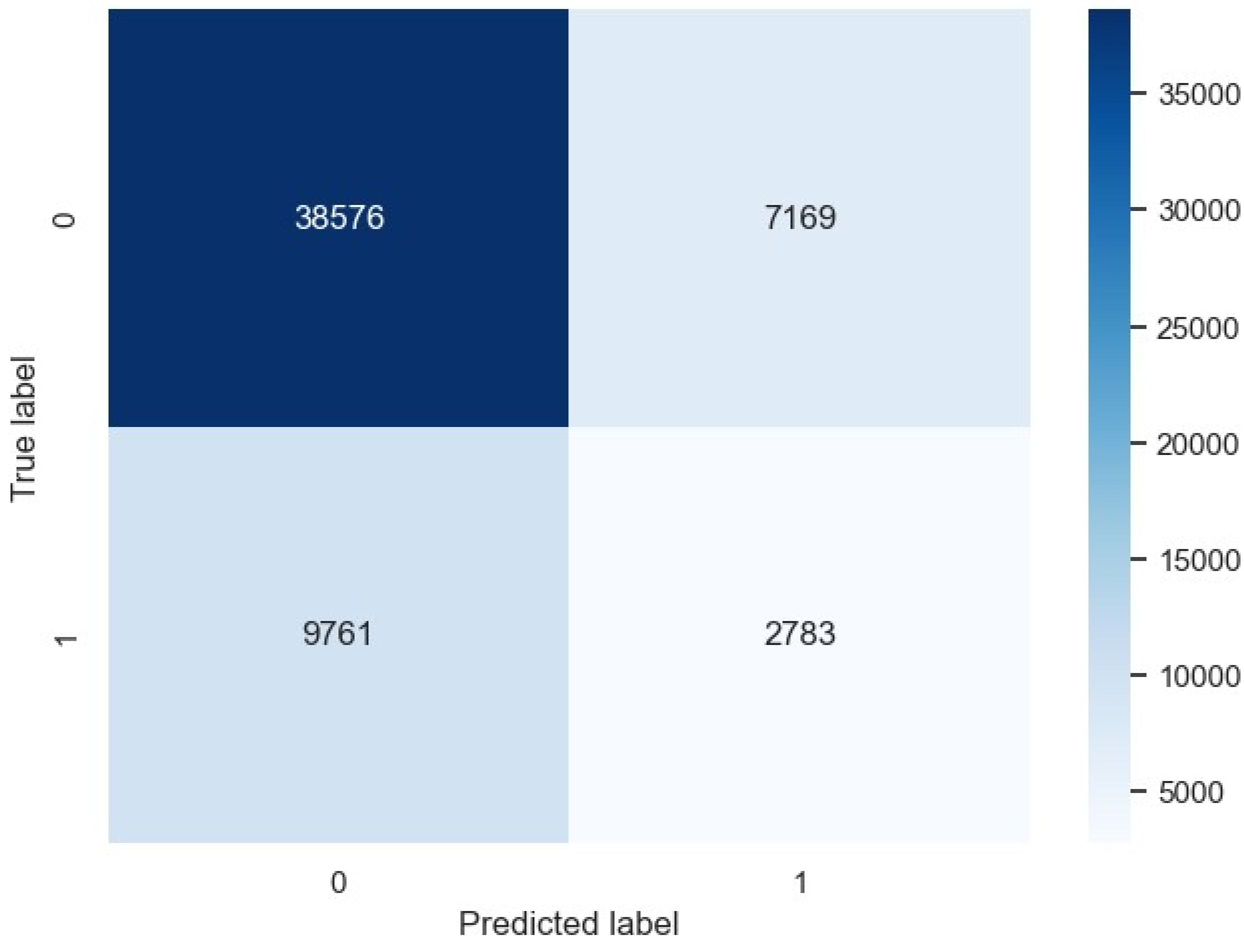
```
from sklearn.ensemble import RandomForestClassifier

# train model
rfc = RandomForestClassifier(n_estimators=10).fit(X_train, y_train)

# predict on test set
rfc_pred = rfc.predict(X_test)
print(confusion_matrix(y_test, rfc_pred))
print(round(accuracy_score(y_test, rfc_pred),2)*100)
LOGCV = (cross_val_score(logmodel, X_train, y_train, cv=k_fold, n_jobs=1, scorir
```

```
[[38576  7169]
 [ 9761  2783]]
71.0
```

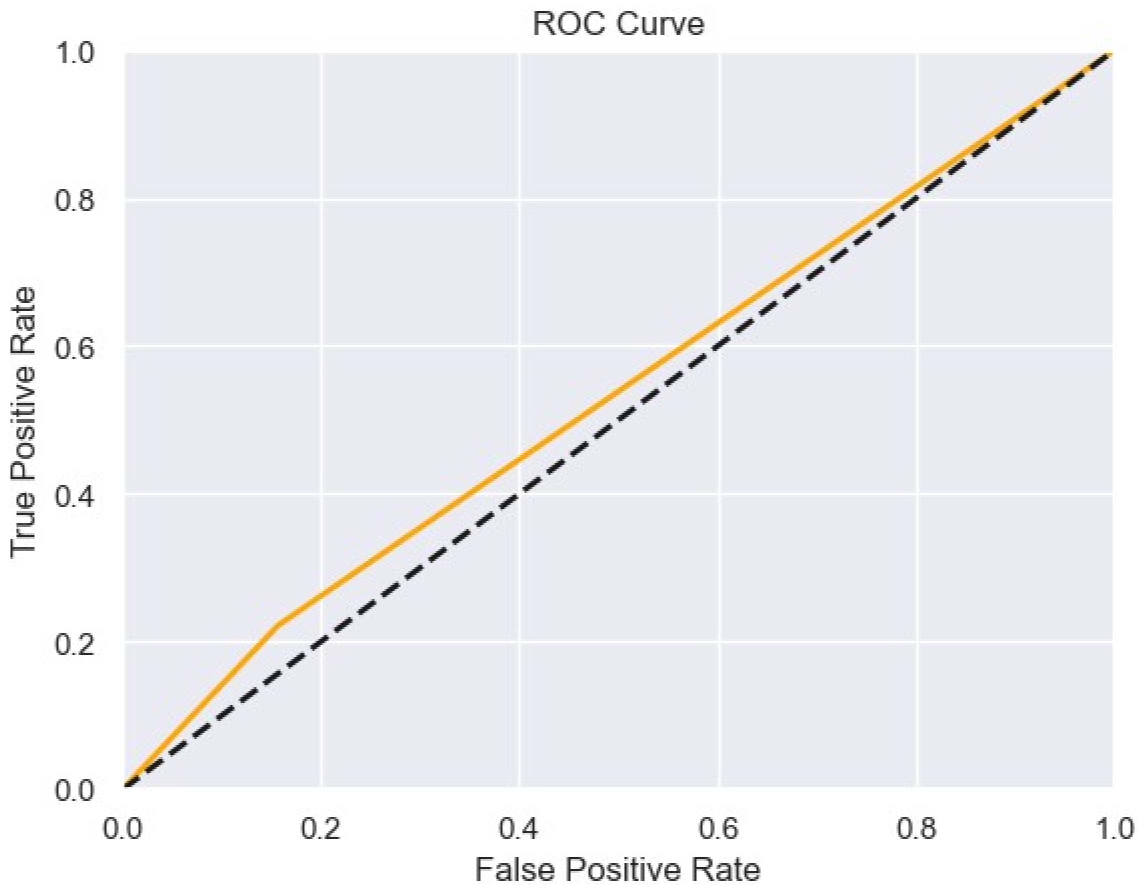
In [120]:

```
cm=metrics.confusion_matrix(y_test, rfc_pred)
show_metrics()

plot_confusion_matrix()
```

```
Precision =     0.280
Recall    =     0.222
F1_score  =     0.247
```



In [121]:
```
plot_roc(y_test, rfc_pred)
```



In [122]:
```
from sklearn.metrics import f1_score
from sklearn.metrics import balanced_accuracy_score
print("Accuracy of model ",accuracy_score(y_test, rfc_pred))
print("F1 Score ",f1_score(y_test, rfc_pred))
print("Recall Score ",recall_score(y_test, rfc_pred))
print("AUC Score ",metrics.roc_auc_score(y_test,rfc_pred))
print("Balanced Accuracy Score ",balanced_accuracy_score(y_test, rfc_pred))
```

```
Accuracy of model  0.7095506870936197
F1 Score  0.24742176386913228
Recall Score  0.22185905612244897
AUC Score  0.53257123753767
Balanced Accuracy Score  0.53257123753767
```

The accuracy of RF might have gone down by 7% but is predicting defaults better now.

5.2 Upsampling

Upsampling can be defined as adding more copies of the minority class. Upsampling can be a good choice when you don’t have a ton of data to work with. (Not a good choice here though)

In [123]:
```
y = train_dummy[['LOAN_DEFAULT']]
X= train_dummy.loc[:, train_dummy.columns != 'LOAN_DEFAULT']
X.shape
```

Out[123]: (233154, 44)

In [124]:
```
from sklearn.utils import resample
# concatenate our training data back together
X = pd.concat([X_train, y_train], axis=1)

# separate minority and majority classes
not_fraud = X[X.LOAN_DEFAULT==0]
fraud = X[X.LOAN_DEFAULT==1]
```

In [12...
```
# upsample minority
fraud_upsampled = resample(fraud,
                          replace=True, # sample with replacement
                          n_samples=len(not_fraud), # match number in majority
                          random_state=27) # reproducible results

# combine majority and upsampled minority
upsampled = pd.concat([not_fraud, fraud_upsampled])

# check new class counts
upsampled.LOAN_DEFAULT.value_counts()

y_train = upsampled.LOAN_DEFAULT
X_train = upsampled.drop('LOAN_DEFAULT', axis=1)
```

In [...
```
from sklearn.tree import DecisionTreeClassifier

# train model
dtree = DecisionTreeClassifier(max_depth = 10, random_state= 101, max_features =N

# predict on test set
dtree_pred = dtree.predict(X_test)
print(confusion_matrix(y_test, dtree_pred))
print(round(accuracy_score(y_test, dtree_pred),2)*100)
LOGCV = (cross_val_score(logmodel, X_train, y_train, cv=k_fold, n_jobs=1, scoring
```

```
[[24438 21307]
 [ 5060  7484]]
55.00000000000001
```

In [1...
```
from sklearn.linear_model import SGDClassifier

# train model
sgd = SGDClassifier(loss= "modified_huber", shuffle = True, random_state= 101).f

# predict on test set
sgd_pred = sgd.predict(X_test)
print(confusion_matrix(y_test, sgd_pred))
print(round(accuracy_score(y_test, sgd_pred),2)*100)
LOGCV = (cross_val_score(logmodel, X_train, y_train, cv=k_fold, n_jobs=1, scorin
```

```
[[27177 18568]
 [ 5721  6823]]
57.99999999999999
```

In [128]:

```
Accuracy of model  0.5833004512000549
F1 Score  0.35972057466719387
Recall Score  0.5439253826530612
AUC Score  0.5690115491251971
Balanced Accuracy Score  0.5690115491251971
```

In [1...
```
from sklearn.ensemble import RandomForestClassifier

# train model
rfc = RandomForestClassifier(n_estimators=10).fit(X_train, y_train)

# predict on test set
rfc_pred = rfc.predict(X_test)
print(confusion_matrix(y_test, rfc_pred))
print(round(accuracy_score(y_test, rfc_pred),2)*100)
LOGCV = (cross_val_score(logmodel, X_train, y_train, cv=k_fold, n_jobs=1, scorir
from sklearn.metrics import f1_score
from sklearn.metrics import balanced_accuracy_score
print("Accuracy of model ",accuracy_score(y_test, rfc_pred))
print("F1 Score ",f1_score(y_test, rfc_pred))
print("Recall Score ",recall_score(y_test, rfc_pred))
print("AUC Score ",metrics.roc_auc_score(y_test,rfc_pred))
print("Balanced Accuracy Score ",balanced_accuracy_score(y_test, rfc_pred))
```

```
[[40705  5040]
 [10582  1962]]
73.0
Accuracy of model  0.7319905985691983
F1 Score  0.20075718817149288
Recall Score  0.1564094387755102
AUC Score  0.5231167316295302
Balanced Accuracy Score  0.5231167316295302
```

5.3 Downsampling

Undersampling can be defined as removing some observations of the majority class. Undersampling can be a good choice when you have a ton of data -think millions of rows. But a drawback is that we are removing information that may be valuable. This could lead to underfitting and poor generalization to the test set.

In [130]:
```
y = train_dummy[['LOAN_DEFAULT']]
X= train_dummy.loc[:, train_dummy.columns != 'LOAN_DEFAULT']
X.shape
```

Out[130]:
```
(233154, 44)
```

In [131]:
```
#Downsample
from sklearn.utils import resample
# concatenate our training data back together
X = pd.concat([X_train, y_train], axis=1)

# separate minority and majority classes
not_fraud = X[X.LOAN_DEFAULT==0]
fraud = X[X.LOAN_DEFAULT==1]
```

In [...

```
from sklearn.tree import DecisionTreeClassifier

# train model
dtree = DecisionTreeClassifier(max_depth = 10, random_state= 101, max_features =N

# predict on test set
dtree_pred = dtree.predict(X_test)
print(confusion_matrix(y_test, dtree_pred))
print(round(accuracy_score(y_test, dtree_pred),2)*100)
LOGCV = (cross_val_score(logmodel, X_train, y_train, cv=k_fold, n_jobs=1, scoring
```

```
[[24438 21307]
 [ 5060  7484]]
55.00000000000001
```

In [1...

```
from sklearn.linear_model import SGDClassifier

# train model
sgd = SGDClassifier(loss= "modified_huber", shuffle = True, random_state= 101).f

# predict on test set
sgd_pred = sgd.predict(X_test)
print(confusion_matrix(y_test, sgd_pred))
print(round(accuracy_score(y_test, sgd_pred),2)*100)
LOGCV = (cross_val_score(logmodel, X_train, y_train, cv=k_fold, n_jobs=1, scorin
```

```
[[28654 17091]
 [ 6249  6295]]
60.0
```

In [134]:

```
from sklearn.metrics import f1_score
from sklearn.metrics import balanced_accuracy_score
print("Accuracy of model ",accuracy_score(y_test, sgd_pred))
print("F1 Score ",f1_score(y_test, sgd_pred))
print("Recall Score ",recall_score(y_test, sgd_pred))
print("AUC Score ",metrics.roc_auc_score(y_test,sgd_pred))
print("Balanced Accuracy Score ",balanced_accuracy_score(y_test, sgd_pred))
```

```
Accuracy of model  0.5995813961467858
F1 Score  0.3504035624826051
Recall Score  0.5018335459183674
AUC Score  0.5641094716147744
Balanced Accuracy Score  0.5641094716147745
```

In [1...

```
from sklearn.ensemble import RandomForestClassifier

# train model
rfc = RandomForestClassifier(n_estimators=10).fit(X_train, y_train)

# predict on test set
rfc_pred = rfc.predict(X_test)
print(confusion_matrix(y_test, rfc_pred))
print(round(accuracy_score(y_test, rfc_pred),2)*100)
LOGCV = (cross_val_score(logmodel, X_train, y_train, cv=k_fold, n_jobs=1, scorin
from sklearn.metrics import f1_score
```

```
[[40782  4963]
 [10537  2007]]
73.0
Accuracy of model  0.7340836178352691
F1 Score  0.20569847289125756
Recall Score  0.1599968112244898
AUC Score  0.5257520398892152
Balanced Accuracy Score  0.525752039889215
```

5.4 PCA

In [136]:
```
y = train_dummy[['LOAN_DEFAULT']]
X= train_dummy.loc[:, train_dummy.columns != 'LOAN_DEFAULT']
X.shape
```

Out[136]:
```
(233154, 44)
```

In [137]:
```
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(X)
```

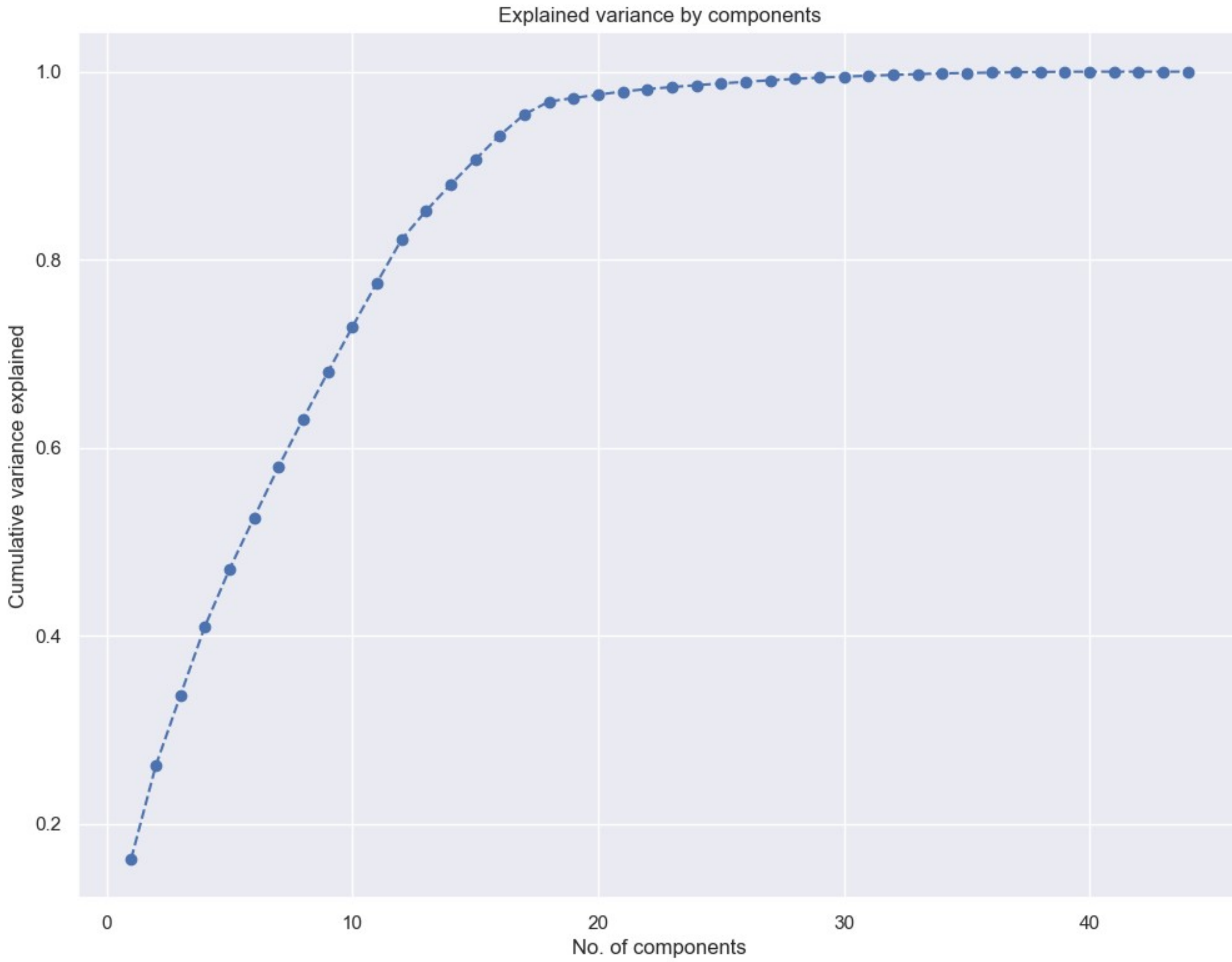
Out[137]:
```
▾ PCA
PCA()
```

In [138]:
```
pca.explained_variance_ratio_.astype(str)
```

Out[138]:
```
array(['0.1626582935070612', '0.09903621412229272', '0.07462430773578396',
       '0.07358981469493256', '0.06085225395809794',
       '0.05492265440886832', '0.05417331740691560 4',
       '0.050860139826426225', '0.04937237829966508',
       '0.048371742087527786', '0.047106943633681356',
       '0.04568138029631013', '0.030792454186137352',
       '0.027879267610843972', '0.026482572171510992',
       '0.025929758169300408', '0.02218006864733693',
       '0.013668751388108457', '0.003768018625633322',
       '0.0035460933355694717', '0.0034366769983810263',
       '0.0025400541925315414', '0.0021101944189278743',
       '0.0020098929032181453', '0.0018506854099092313',
       '0.001781416986869723', '0.0015677351561001781',
       '0.0015197371949381834', '0.0013412339627867558',
       '0.0010616394683435261', '0.0009210425276718671',
       '0.0009079937390059669', '0.00086572210026907 6',
       '0.0007485076826041395', '0.00044451400038288873',
       '0.0003865189912228064', '0.00036704790780511874',
       '0.00028471711592024056', '0.00024746152989298993',
       '0.00010885061441257197', '1.3511217860020275e-06',
       '5.81865016557263e-07', '1.0053488740834725e-33',
       '1.0053192526339164e-33'], dtype='<U32')
```

In [1...
```
plt.figure(figsize= (12,9))
plt.plot(range(1,45), pca.explained_variance_ratio_.cumsum(), marker= 'o', line
plt.title("Explained variance by components")
```

Out[139]:Text(0, 0.5, 'Cumulative variance explained')



In [140]:
```
pca = PCA(n_components = 17)
pca.fit(X)
```

Out[140]:

PCA

PCA(n_components=17)

In [1...
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, ran
k_fold = KFold(n_splits=10, shuffle=True, random_state=0)
```

In [...
```
from sklearn.tree import DecisionTreeClassifier

# train model
dtree = DecisionTreeClassifier(max_depth = 10, random_state= 101, max_features =N

# predict on test set
dtree_pred = dtree.predict(X_test)
print(confusion_matrix(y_test, dtree_pred))
print(round(accuracy_score(y_test, dtree_pred),2)*100)
LOGCV = (cross_val_score(logmodel, X_train, y_train, cv=k_fold, n_jobs=1, scoring
```

```
[[36190   315]
 [ 9951   175]]
78.0
```

In [1...

```python
from sklearn.linear_model import SGDClassifier

# train model
sgd = SGDClassifier(loss= "modified_huber", shuffle = True, random_state= 101).f

# predict on test set
sgd_pred = sgd.predict(X_test)
print(confusion_matrix(y_test, sgd_pred))
print(round(accuracy_score(y_test, sgd_pred),2)*100)
LOGCV = (cross_val_score(logmodel, X_train, y_train, cv=k_fold, n_jobs=1, scorin
```

```
[[36480    25]
 [10112    14]]
78.0
```

In [144]:

```python
from sklearn.metrics import f1_score
from sklearn.metrics import balanced_accuracy_score
print("Accuracy of model ",accuracy_score(y_test, sgd_pred))
print("F1 Score ",f1_score(y_test, sgd_pred))
print("Recall Score ",recall_score(y_test, sgd_pred))
print("AUC Score ",metrics.roc_auc_score(y_test,sgd_pred))
print("Balanced Accuracy Score ",balanced_accuracy_score(y_test, sgd_pred))
```

```
Accuracy of model  0.7826124252106967
F1 Score  0.002754549926217413
Recall Score  0.0013825794983211535
AUC Score  0.5003488709024272
Balanced Accuracy Score  0.5003488709024273
```

In [1...

```python
from sklearn.ensemble import RandomForestClassifier

# train model
rfc = RandomForestClassifier(n_estimators=10).fit(X_train, y_train)

# predict on test set
rfc_pred = rfc.predict(X_test)
print(confusion_matrix(y_test, rfc_pred))
print(round(accuracy_score(y_test, rfc_pred),2)*100)
LOGCV = (cross_val_score(logmodel, X_train, y_train, cv=k_fold, n_jobs=1, scorin
from sklearn.metrics import f1_score
from sklearn.metrics import balanced_accuracy_score
print("Accuracy of model ",accuracy_score(y_test, rfc_pred))
print("F1 Score ",f1_score(y_test, rfc_pred))
print("Recall Score ",recall_score(y_test, rfc_pred))
print("AUC Score ",metrics.roc_auc_score(y_test,rfc_pred))
print("Balanced Accuracy Score ",balanced_accuracy_score(y_test, rfc_pred))
```

```
[[35093  1412]
 [ 9434   692]]
77.0
Accuracy of model  0.7674079475027342
F1 Score  0.11316434995911692
Recall Score  0.06833892948844558
AUC Score  0.5148296482807246
Balanced Accuracy Score  0.5148296482807246
```

5.5 Resampling

In [146]:

```
y = train_dummy[['LOAN_DEFAULT']]
X= train_dummy.loc[:, train_dummy.columns != 'LOAN_DEFAULT']
X.shape
```

Out[146]:

```
(233154, 44)
```

In [147]:

```
from sklearn.utils import resample
# concatenate our training data back together
X = pd.concat([X_train, y_train], axis=1)

# separate minority and majority classes
not_fraud = X[X.LOAN_DEFAULT==0]
fraud = X[X.LOAN_DEFAULT==1]
```

In [14...

```
# upsample minority
fraud_upsampled = resample(fraud,
                          replace=True, # sample with replacement
                          n_samples=len(not_fraud), # match number in majority
                          random_state=27) # reproducible results

# combine majority and upsampled minority
upsampled = pd.concat([not_fraud, fraud_upsampled])
```

In [149]:

```
# check new class counts
upsampled.LOAN_DEFAULT.value_counts()
```

Out[149]:

```
0    146038
1    146038
Name: LOAN_DEFAULT, dtype: int64
```

In [150]:

```
y_train = upsampled.LOAN_DEFAULT
X_train = upsampled.drop('LOAN_DEFAULT', axis=1)
```

In [...

```
from sklearn.tree import DecisionTreeClassifier

# train model
dtree = DecisionTreeClassifier(max_depth = 10, random_state= 101, max_features =N

# predict on test set
dtree_pred = dtree.predict(X_test)
print(confusion_matrix(y_test, dtree_pred))
print(round(accuracy_score(y_test, dtree_pred),2)*100)
LOGCV = (cross_val_score(logmodel, X_train, y_train, cv=k_fold, n_jobs=1, scoring
```

```
[[18007 18498]
 [ 3464  6662]]
53.0
```

In [152]:
```python
from sklearn.metrics import f1_score
from sklearn.metrics import balanced_accuracy_score
print("Accuracy of model ",accuracy_score(y_test, dtree_pred))
print("F1 Score ",f1_score(y_test, dtree_pred))
print("Recall Score ",recall_score(y_test, dtree_pred))
print("AUC Score ",metrics.roc_auc_score(y_test,dtree_pred))
print("Balanced Accuracy Score ",balanced_accuracy_score(y_test, dtree_pred))
```

```
Accuracy of model  0.5290257553987691
F1 Score  0.3776001813750496
Recall Score  0.657910329843966
AUC Score  0.5755926118470618
Balanced Accuracy Score  0.5755926118470618
```

In [1...
```python
from sklearn.linear_model import SGDClassifier

# train model
sgd = SGDClassifier(loss= "modified_huber", shuffle = True, random_state= 101).f

# predict on test set
sgd_pred = sgd.predict(X_test)
print(confusion_matrix(y_test, sgd_pred))
print(round(accuracy_score(y_test, sgd_pred),2)*100)
LOGCV = (cross_val_score(logmodel, X_train, y_train, cv=k_fold, n_jobs=1, scorir
```

```
[[11751 24754]
 [ 1993  8133]]
43.0
```

In [154]:
```python
from sklearn.metrics import f1_score
from sklearn.metrics import balanced_accuracy_score
print("Accuracy of model ",accuracy_score(y_test, sgd_pred))
print("F1 Score ",f1_score(y_test, sgd_pred))
print("Recall Score ",recall_score(y_test, sgd_pred))
print("AUC Score ",metrics.roc_auc_score(y_test,sgd_pred))
print("Balanced Accuracy Score ",balanced_accuracy_score(y_test, sgd_pred))
```

```
Accuracy of model  0.4264116145911518
F1 Score  0.3781647408922884
Recall Score  0.8031799328461386
AUC Score  0.562540521141601
Balanced Accuracy Score  0.562540521141601
```

Comparing all the models based on Model Performance

In [...
```python
comparison_frame = pd.DataFrame({'Model':['Test_accuracy               :',
                                          'Test_F1_Score               :',
                                          'Test_Recall_Score           :',
                                          'Test_AUC_Score              :',
                                          'Test_Balanced_Accuracy_Score:'],
                                          'Logisitic Regression':[0.78289,0.0084
                                          'Random Forest':[0.76787,0.11423,0.068
```

Out[167]:

	Model	Logisitic Regression	Random Forest	Naive Bayes	Stochastic Gradient Descent	Decision Tree	Decision Tree (SMOTE)	Ran Fo (SM
0	Test_accuracy :	0.78289	0.76787	0.34018	0.78261	0.77984	0.55609	0.7
1	Test_F1_Score :	0.00842	0.11423	0.36774	0.00275	0.03296	0.36027	0.2
2	Test_Recall_Score :	0.00424	0.06893	0.88366	0.00138	0.01728	0.58083	0.2
3	Test_AUC_Score :	0.50156	0.51534	0.53654	0.50034	0.50432	0.56507	0.5
4	Test_Balanced_Accuracy_Score:	0.50156	0.51534	0.53654	0.50034	0.50432	0.56507	0.5

Results interpretation :

Logisitic Regression - Accuracy score is good, however the model is not predicting the Defaults well

Random Forest - Accuracy score is good, however the model is predicting the Defaults better than Logistic reg

Naive Bayes - Model accuracy is very poor

Stochastic Gradient Descent & Decision Tree - Accuracy score is good, however the model is not predicting the Defaults well

Random Forest (SMOTE) - The accuracy of RF might have gone down by 7% but is predicting defaults better. (SMOTE uses a nearest neighbors algorithm to generate new and synthetic data we can use for training our model.)

Note: Upsampling & Undersampling results , i haven't considered due to the following reasons

Upsampling can be defined as adding more copies of the minority class. Upsampling can be a good choice when you don’t have a ton of data to work with. (Not a good choice here though)

Undersampling can be defined as removing some observations of the majority class. Undersampling can be a good choice when you have a ton of data -think millions of rows. But a drawback is that we are removing information that may be valuable. This could lead to underfitting and poor generalization to the test set.

Best result is obtained by Random Forest after deploying [SMOTE].

In [ ]: