Importing required Library's

In [1]:
```
import pandas as pd
import numpy as np
from datetime import datetime
import sklearn
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
```

From the Training Data set we first load main_loan_base.

In [2]:
```
# Loanding the main_loan_base
loan_base = pd.read_csv('main_loan_base.csv')
```

In [3]:
```
loan_base.head()
```

Out[3]:

	loan_acc_num	customer_name	customer_address	loan_type	loan_amount	collateral_value
0	LN79307711	Aarna Sura	09/506, Anand Path, Ongole 646592	Consumer-Durable	21916	4929.47
1	LN88987787	Amira Konda	11, Dhaliwal Circle\nRaichur 659460	Two-Wheeler	121184	10254.50
2	LN78096023	Eshani Khosla	H.No. 31\nAtwal Street\nKatihar-037896	Car	487036	116183.86
3	LN56862431	Divij Kala	766, Gulati Marg\nPudukkottai-051396	Two-Wheeler	52125	10310.05
4	LN77262680	Vaibhav Bir	55/73, Sachdev Marg\nDharmavaram-332966	Consumer-Durable	8635	1051.25

In [4]:
```
#For the purpose of predicting LGD we feel that 'customer_name', 'customer_address
# we are dropping those columns.
loan_base = loan_base.drop(['customer_name', 'customer_address'], axis=1)
```

In [5]:
```
#Converting 'default_date' & 'disbursal_date' columns into datetime format.
loan_base['default_date'] = loan_base['default_date'].astype('datetime64[ns]')
loan_base['disbursal_date'] = loan_base['disbursal_date'].astype('datetime64[ns]')
```

In [6]:
```
loan_base.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   loan_acc_num       50000 non-null  object
 1   loan_type          50000 non-null  object
 2   loan_amount        50000 non-null  int64
 3   collateral_value   50000 non-null  float64
 4   cheque_bounces     50000 non-null  int64
 5   number_of_loans    50000 non-null  int64
 6   missed_repayments  50000 non-null  int64
 7   vintage_in_months  50000 non-null  int64
 8   tenure_years       50000 non-null  int64
 9   interest           50000 non-null  float64
 10  monthly_emi        50000 non-null  float64
 11  disbursal_date     50000 non-null  datetime64[ns]
 12  default_date       50000 non-null  datetime64[ns]
dtypes: datetime64[ns](2), float64(3), int64(6), object(2)
memory usage: 5.0+ MB
```

In [7]:
```
# Checking for missing/ na values in the columns.
for column in loan_base:
    btao = loan_base[column].isna().sum()
    btao2 = (btao * 100)/len(loan_base)
    print(column + " -> " + str(btao2) + ' %')
```

```
loan_acc_num -> 0.0 %
loan_type -> 0.0 %
loan_amount -> 0.0 %
collateral_value -> 0.0 %
cheque_bounces -> 0.0 %
number_of_loans -> 0.0 %
missed_repayments -> 0.0 %
vintage_in_months -> 0.0 %
tenure_years -> 0.0 %
interest -> 0.0 %
monthly_emi -> 0.0 %
disbursal_date -> 0.0 %
default_date -> 0.0 %
```

In [8]:
```
loan_base.describe()
```

Out[8]:

	loan_amount	collateral_value	cheque_bounces	number_of_loans	missed_repayments	vinta
count	5.000000e+04	50000.000000	50000.000000	50000.000000	50000.000000	
mean	3.816870e+05	57189.733515	1.764740	1.509540	9.808280	
std	5.037605e+05	93407.376232	1.760175	1.259389	7.788007	
min	2.000000e+03	0.070000	0.000000	0.000000	0.000000	
25%	2.393550e+04	3329.392500	0.000000	0.000000	4.000000	
50%	1.926885e+05	19863.105000	1.000000	1.000000	8.000000	
75%	4.334075e+05	62313.440000	3.000000	2.000000	15.000000	
max	1.999992e+06	592545.710000	11.000000	6.000000	38.000000	

In [9]:
```
num_cols = loan_base.select_dtypes('number').columns
```

In [10]:
```
# Creating a new column 'loan_tenure' which is difference between 'default_date' &
loan_base['loan_tenure'] = (loan_base['default_date'] - loan_base['disbursal_date'
```

In [11]:
```
loan_base['loan_tenure'] = loan_base['loan_tenure'].astype('int')
```

In [12]:
```
# Now we have loan_tenure in months.
# We can drop the both date columns 'default_date', 'disbursal_date'.
loan_base = loan_base.drop(['default_date', 'disbursal_date'], axis = 1)
loan_base.head()
```

Out[12]:

	loan_acc_num	loan_type	loan_amount	collateral_value	cheque_bounces	number_of_loans	mi
0	LN79307711	Consumer-Durable	21916	4929.47	3	0	
1	LN88987787	Two-Wheeler	121184	10254.50	1	0	
2	LN78096023	Car	487036	116183.86	0	1	
3	LN56862431	Two-Wheeler	52125	10310.05	5	2	
4	LN77262680	Consumer-Durable	8635	1051.25	0	1	

Loading Train Repayment Data

In [13]:
```
# Loading the main_repayment_base
repayment_base = pd.read_csv('repayment_base.csv')
```

In [14]:
```
repayment_base.head()
```

Out[14]:

	loan_acc_num	repayment_amount	repayment_date
0	LN79307711	1012.320000	2019-05-18
1	LN79307711	667.987539	2019-06-20
2	LN79307711	1012.320000	2019-07-16
3	LN79307711	1012.320000	2019-08-16
4	LN79307711	1012.320000	2019-09-22

In [15]:
```
repayment_base.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 626601 entries, 0 to 626600
Data columns (total 3 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   loan_acc_num      626601 non-null  object 
 1   repayment_amount  626601 non-null  float64
 2   repayment_date    626601 non-null  object 
dtypes: float64(1), object(2)
memory usage: 14.3+ MB
```

In [16]:
```
# Earlier we saw that number of loan_acc_num is 50k now we are getting a figure of
```

In [17]:
```
repayment_base.nunique(axis = 0)
```

Out[17]:
```
loan_acc_num         46008
repayment_amount    138950
repayment_date        4813
dtype: int64
```

In [18]:
```
# Checking for missing/ na values in the columns.
for column in repayment_base:
    btao = repayment_base[column].isna().sum()
    btao2 = (btao * 100)/len(loan_base)
    print(column + " -> " + str(btao2) + ' %')
```

```
loan_acc_num -> 0.0 %
repayment_amount -> 0.0 %
repayment_date -> 0.0 %
```

In [19]:
```
repayment_base['repayment_amount'] = repayment_base['repayment_amount'].round(2)
repayment_base.head()
```

Out[19]:

	loan_acc_num	repayment_amount	repayment_date
0	LN79307711	1012.32	2019-05-18
1	LN79307711	667.99	2019-06-20
2	LN79307711	1012.32	2019-07-16
3	LN79307711	1012.32	2019-08-16
4	LN79307711	1012.32	2019-09-22

In [20]:
```
# Lets calculate the total amount which has been repayed for each loan.
```

In [21]:
```
repayment_base.drop(['repayment_date'], axis = 1)
```

Out[21]:

	loan_acc_num	repayment_amount
0	LN79307711	1012.32
1	LN79307711	667.99
2	LN79307711	1012.32
3	LN79307711	1012.32
4	LN79307711	1012.32
...	...	...
626596	LN74765572	3771.91
626597	LN74765572	3771.91
626598	LN46546410	21443.47
626599	LN46546410	21443.47
626600	LN46546410	21443.47

626601 rows × 2 columns

In [22]:
```
group_repayment = repayment_base.groupby(by = ['loan_acc_num']).sum()
```

```
C:\Users\amann\AppData\Local\Temp\ipykernel_14360\3896089517.py:1: FutureWarning:
The default value of numeric_only in DataFrameGroupBy.sum is deprecated. In a futu
re version, numeric_only will default to False. Either specify numeric_only or sel
ect only columns which should be valid for the function.
  group_repayment = repayment_base.groupby(by = ['loan_acc_num']).sum()
```

In [23]:
```
group_repayment = group_repayment.reset_index()
```

In [24]:
```
group_repayment.head()
```

Out[24]:

	loan_acc_num	repayment_amount
0	LN10000701	40020.99
1	LN10001077	112218.46
2	LN10004116	290634.94
3	LN10007976	337321.71
4	LN10010204	61290.49

In [25]:
```
group_repayment.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 46008 entries, 0 to 46007
Data columns (total 2 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   loan_acc_num      46008 non-null  object
 1   repayment_amount  46008 non-null  float64
dtypes: float64(1), object(1)
memory usage: 719.0+ KB
```

In [26]:
```
group_repayment.nunique(axis = 0)
```

Out[26]:
```
loan_acc_num        46008
repayment_amount    45943
dtype: int64
```

Loading Train monthly account balance data

In [27]:
```
monthly_balance = pd.read_csv('monthly_balance_base.csv')
```

In [28]:
```
monthly_balance.head()
```

Out[28]:

	loan_acc_num	date	balance_amount
0	LN79307711	2010-03-26	407.343213
1	LN79307711	2010-04-25	545.431227
2	LN79307711	2010-05-25	861.932145
3	LN79307711	2010-06-24	562.082133
4	LN79307711	2010-07-24	37.768861

In [29]:
```
monthly_balance.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4002490 entries, 0 to 4002489
Data columns (total 3 columns):
 #   Column          Dtype  
---  ------          -----  
 0   loan_acc_num    object 
 1   date            object 
 2   balance_amount  float64
dtypes: float64(1), object(2)
memory usage: 91.6+ MB
```

In [30]:
```
# Dropping the date column as we dont need it.
monthly_balance.drop(['date'], axis = 1)
```

Out[30]:

	loan_acc_num	balance_amount
0	LN79307711	407.343213
1	LN79307711	545.431227
2	LN79307711	861.932145
3	LN79307711	562.082133
4	LN79307711	37.768861
...	...	...
4002485	LN46546410	10875.235336
4002486	LN46546410	8479.809099
4002487	LN46546410	9745.974332
4002488	LN46546410	9226.494566
4002489	LN46546410	8851.138461

4002490 rows × 2 columns

In [31]:
```
for column in monthly_balance:
    btao = monthly_balance[column].isna().sum()
    btao2 = (btao * 100)/len(monthly_balance)
    print(column + " -> " + str(btao2) + ' %')
```

```
loan_acc_num -> 0.0 %
date -> 0.0 %
balance_amount -> 0.0 %
```

In [32]:
```
#Consolidating the average balance via 'loan_acc_num'.
grouped_monthly = monthly_balance.groupby(by=['loan_acc_num']).mean()
```

```
C:\Users\amann\AppData\Local\Temp\ipykernel_14360\2974728870.py:2: FutureWarning:
The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a fut
ure version, numeric_only will default to False. Either specify numeric_only or se
lect only columns which should be valid for the function.
  grouped_monthly = monthly_balance.groupby(by=['loan_acc_num']).mean()
```

In [33]:
```
grouped_monthly = grouped_monthly.reset_index()
```

In [34]:
```
grouped_monthly
```

Out[34]:

	loan_acc_num	balance_amount
0	LN10000701	2301.879193
1	LN10001077	2296.279543
2	LN10004116	8887.380832
3	LN10007976	9420.561560
4	LN10010204	6446.205233
...	...	...
49666	LN99991810	20537.816328
49667	LN99992591	263.078287
49668	LN99995043	267.037722
49669	LN99995214	202.563984
49670	LN99995643	45908.804885

49671 rows × 2 columns

In [35]:
```
# Rounding off to two decimal place the 'balance_amount' column.
grouped_monthly['balance_amount'] = grouped_monthly['balance_amount'].round(2)
```

In [36]:
```
grouped_monthly.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49671 entries, 0 to 49670
Data columns (total 2 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   loan_acc_num    49671 non-null  object 
 1   balance_amount  49671 non-null  float64
dtypes: float64(1), object(1)
memory usage: 776.2+ KB
```

In [37]:
```
grouped_monthly.nunique(axis = 0)
```

Out[37]:
```
loan_acc_num      49671
balance_amount    47059
dtype: int64
```

now our monthly balance data is also ready to be mearged with loan_base data

In [38]:
```
# Merging 'loan_base' data with 'group_repayment' data set.
data1 = loan_base.merge(group_repayment, how = 'outer', on = 'loan_acc_num')
```

In [39]:
```
# Merging 'data1' with 'grouped_monthly' data set.
data = data1.merge(grouped_monthly, how = 'outer', on = 'loan_acc_num')
```

In [40]:
```
data.head()
```

Out[40]:

	loan_acc_num	loan_type	loan_amount	collateral_value	cheque_bounces	number_of_loans	mi
0	LN79307711	Consumer-Durable	21916	4929.47	3	0	
1	LN88987787	Two-Wheeler	121184	10254.50	1	0	
2	LN78096023	Car	487036	116183.86	0	1	
3	LN56862431	Two-Wheeler	52125	10310.05	5	2	
4	LN77262680	Consumer-Durable	8635	1051.25	0	1	

In [41]:
```
data.tail()
```

Out[41]:

	loan_acc_num	loan_type	loan_amount	collateral_value	cheque_bounces	number_of_loans
49995	LN82044693	Two-Wheeler	222483	43088.19	2	2
49996	LN37968463	Two-Wheeler	104051	8666.54	6	2
49997	LN87152445	Two-Wheeler	51767	4101.24	1	0
49998	LN74765572	Two-Wheeler	77869	10652.77	0	0
49999	LN46546410	Two-Wheeler	241857	59258.17	0	2

In [42]:
```
data[['repayment_amount', 'balance_amount']].describe()
```

Out[42]:

	repayment_amount	balance_amount
count	4.602300e+04	49686.000000
mean	1.645432e+05	7678.490833
std	2.622382e+05	16119.892191
min	5.228000e+01	0.100000
25%	1.076201e+04	412.062500
50%	6.155783e+04	2151.835000
75%	1.829428e+05	7339.770000
max	1.852111e+06	261799.900000

In [43]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50000 entries, 0 to 49999
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   loan_acc_num       50000 non-null  object 
 1   loan_type          50000 non-null  object 
 2   loan_amount        50000 non-null  int64  
 3   collateral_value   50000 non-null  float64
 4   cheque_bounces     50000 non-null  int64  
 5   number_of_loans    50000 non-null  int64  
 6   missed_repayments  50000 non-null  int64  
 7   vintage_in_months  50000 non-null  int64  
 8   tenure_years       50000 non-null  int64  
 9   interest           50000 non-null  float64
 10  monthly_emi        50000 non-null  float64
 11  loan_tenure        50000 non-null  int32  
 12  repayment_amount   46023 non-null  float64
 13  balance_amount     49686 non-null  float64
dtypes: float64(5), int32(1), int64(6), object(2)
memory usage: 5.5+ MB
```

In [44]:
```
# We find missing values in 'repayment_amount' & 'balance_amount'
```

In [45]:
```
data.isnull().sum().sum()
```

Out[45]:
```
4291
```

In [46]:
```
# Replacing missing values by 0 as we can assume that no repayment has been done ir
# and account balance is also 0.
```

In [47]:
```
data['repayment_amount'] = data['repayment_amount'].fillna(0)
```

In [48]:
```
data['balance_amount'] = data['balance_amount'].fillna(0)
```

In [49]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50000 entries, 0 to 49999
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   loan_acc_num       50000 non-null  object 
 1   loan_type          50000 non-null  object 
 2   loan_amount        50000 non-null  int64  
 3   collateral_value   50000 non-null  float64
 4   cheque_bounces     50000 non-null  int64  
 5   number_of_loans    50000 non-null  int64  
 6   missed_repayments  50000 non-null  int64  
 7   vintage_in_months  50000 non-null  int64  
 8   tenure_years       50000 non-null  int64  
 9   interest           50000 non-null  float64
 10  monthly_emi        50000 non-null  float64
 11  loan_tenure        50000 non-null  int32  
 12  repayment_amount   50000 non-null  float64
 13  balance_amount     50000 non-null  float64
dtypes: float64(5), int32(1), int64(6), object(2)
memory usage: 5.5+ MB
```

In [50]:
```
# now our data is ready and we can go ahead with calculating LGD
# LGD = (loan Amount -(Amount repayed + Collateral Value))/ Loan Amount
```

In [51]:
```
data['lgd_pct'] = (data['loan_amount'] - (data['repayment_amount'] + data['collate
```

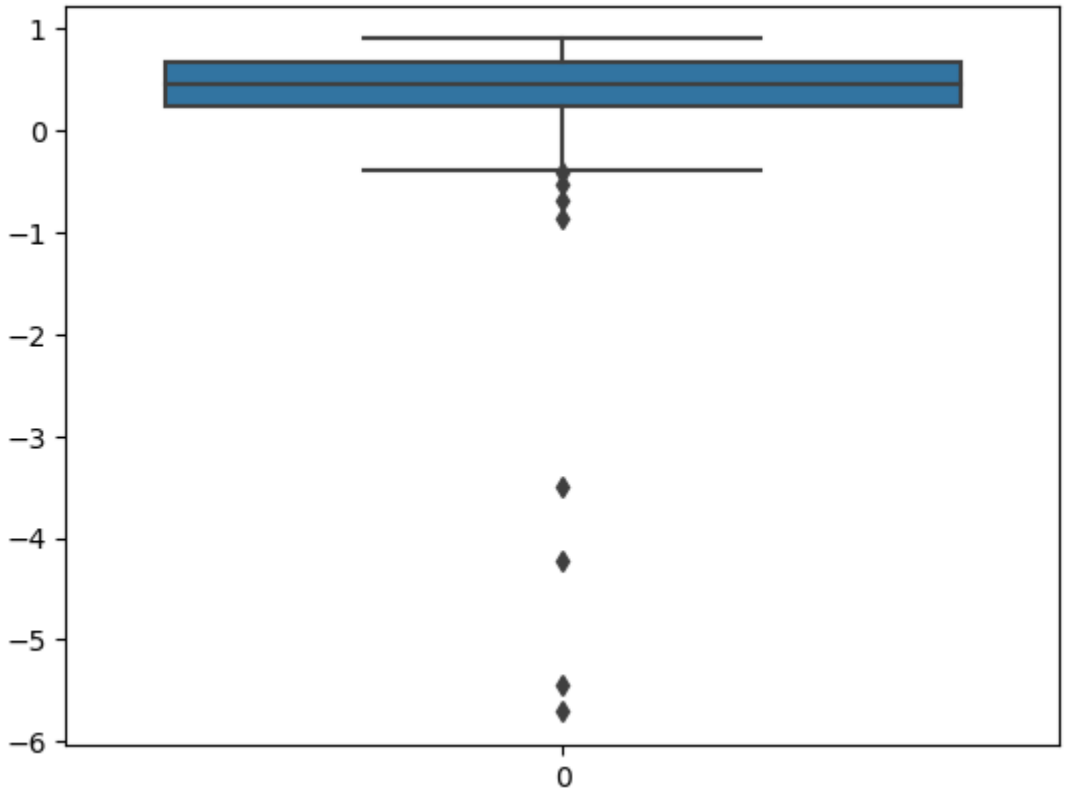
In [52]:
```
data.head()
```

Out[52]:

	loan_acc_num	loan_type	loan_amount	collateral_value	cheque_bounces	number_of_loans	mi
0	LN79307711	Consumer-Durable	21916	4929.47	3	0	
1	LN88987787	Two-Wheeler	121184	10254.50	1	0	
2	LN78096023	Car	487036	116183.86	0	1	
3	LN56862431	Two-Wheeler	52125	10310.05	5	2	
4	LN77262680	Consumer-Durable	8635	1051.25	0	1	

In [53]:
```
sns.boxplot(data['lgd_pct'])
```

Out[53]:
```
<Axes: >
```



In [54]:
```
n_values = data.loc[data['lgd_pct'] < 0]
#df.loc[(df['col1'] == value) & (df['col2'] < value)]
```

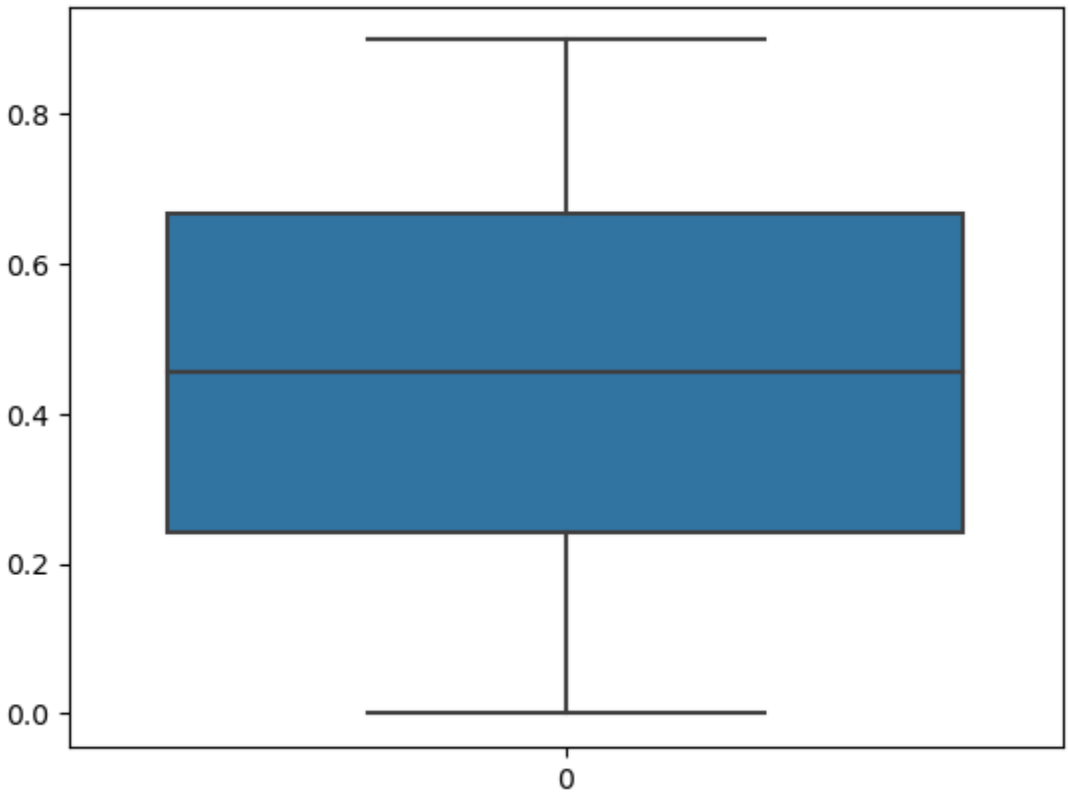
In [55]:
```
n_values.shape
```

Out[55]:
```
(101, 15)
```

In [56]:
```
data['lgd_pct'] = data['lgd_pct'].apply(lambda x: x if x>0 else 0 )
```

In [57]:
```
sns.boxplot(data['lgd_pct'])
```

Out[57]: <Axes: >



In [58]:
```
# Naming the data to be used for training the model as 'train-data'
train_data = data
```

Now lets prepare the test data in a similar way

In [59]:
```
# Loading the data set to be used for testing the model.
test_main_base = pd.read_csv('test_main_loan_base.csv')
```

In [60]:
```
test_main_base.head()
```

Out[60]:

	loan_acc_num	customer_name	customer_address	loan_type	loan_amount	collateral_value	ch
0	LN14086568	Jayesh Kar	83/65, Deo Circle\nBhagalpur-852841	Car	1259062	10184.09	
1	LN37082418	Kaira Chhabra	17\nSule Ganj\nAizawl 491897	Consumer-Durable	21731	2313.21	
2	LN42963368	Anahita Bhargava	51/421\nKannan Chowk\nVaranasi-209999	Car	207660	8308.71	
3	LN54572294	Myra Samra	22\nSubramanian Marg, Bhilai 850327	Two-Wheeler	193528	26432.24	
4	LN65792799	Arhaan Rana	22, Kapoor Road\nJalandhar 667155	Consumer-Durable	5980	1641.66	

In [61]:
```
#For the purpose of predicting LGD we feel that 'customer_name', 'customer_address
# we are dropping those columns.

test_main_base = test_main_base.drop(['customer_name', 'customer_address'], axis=1
```

In [62]:
```
#Converting 'default_date' & 'disbursal_date' columns into datetime format.
test_main_base['default_date'] = test_main_base['default_date'].astype('datetime64
test_main_base['disbursal_date'] = test_main_base['disbursal_date'].astype('dateti
```

In [63]:
```
# Creating a new column 'loan_tenure' which is difference between 'default_date' &
test_main_base['loan_tenure'] = (test_main_base['default_date'] - test_main_base['
```

In [64]:
```
test_main_base['loan_tenure'] = test_main_base['loan_tenure'].astype('int')
```

In [65]:
```
# now we have loan tenure
# we can drop the both date columns
test_main_base = test_main_base.drop(['default_date', 'disbursal_date'], axis = 1)
```

In [66]:
```
test_main_base.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   loan_acc_num       10000 non-null  object
 1   loan_type          10000 non-null  object
 2   loan_amount        10000 non-null  int64
 3   collateral_value   10000 non-null  float64
 4   cheque_bounces     10000 non-null  int64
 5   number_of_loans    10000 non-null  int64
 6   missed_repayments  10000 non-null  int64
 7   vintage_in_months  10000 non-null  int64
 8   tenure_years       10000 non-null  int64
 9   interest           10000 non-null  float64
 10  monthly_emi        10000 non-null  float64
 11  loan_tenure        10000 non-null  int32
dtypes: float64(3), int32(1), int64(6), object(2)
memory usage: 898.6+ KB
```

Loading test repayment data

In [67]:
```
test_repayment = pd.read_csv('test_repayment_base.csv')
```

In [68]:
```
test_repayment.head()
```

Out[68]:

	loan_acc_num	repayment_amount	repayment_date
0	LN14086568	111925.040000	2014-12-19
1	LN14086568	111925.040000	2015-01-26
2	LN14086568	111925.040000	2015-02-21
3	LN14086568	111925.040000	2015-03-20
4	LN14086568	11590.317813	2015-04-26

In [69]:
```
test_repayment.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 125860 entries, 0 to 125859
Data columns (total 3 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   loan_acc_num      125860 non-null  object
 1   repayment_amount  125860 non-null  float64
 2   repayment_date    125860 non-null  object
dtypes: float64(1), object(2)
memory usage: 2.9+ MB
```

In [70]:
```
# Dropping the date column as we dont need it.
test_repayment = test_repayment.drop('repayment_date', axis=1)
```

In [71]:
```
test_repayment.head()
```

Out[71]:

	loan_acc_num	repayment_amount
0	LN14086568	111925.040000
1	LN14086568	111925.040000
2	LN14086568	111925.040000
3	LN14086568	111925.040000
4	LN14086568	11590.317813

In [72]:
```
# rounding off to two decimal place 'repayment_amount' column.
test_repayment['repayment_amount'] = test_repayment['repayment_amount'].round(2)
```

In [73]:
```
test_repayment.head()
```

Out[73]:

	loan_acc_num	repayment_amount
0	LN14086568	111925.04
1	LN14086568	111925.04
2	LN14086568	111925.04
3	LN14086568	111925.04
4	LN14086568	11590.32

In [74]:
```
# calculating the repayment made in each loan account by adding.
group_test_repayment = test_repayment.groupby(by = ['loan_acc_num']).sum()
```

In [75]:
```
group_test_repayment.head()
```

Out[75]:

loan_acc_num	repayment_amount
LN10011015	1725.32
LN10028091	3560.31
LN10033713	11582.17
LN10045654	66181.74
LN10051605	87664.41

In [76]:
```
group_test_repayment = group_test_repayment.reset_index()
```

In [77]:
```
group_test_repayment.head()
```

Out[77]:

	loan_acc_num	repayment_amount
0	LN10011015	1725.32
1	LN10028091	3560.31
2	LN10033713	11582.17
3	LN10045654	66181.74
4	LN10051605	87664.41

In [78]:
```
group_test_repayment.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9229 entries, 0 to 9228
Data columns (total 2 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   loan_acc_num      9229 non-null   object 
 1   repayment_amount  9229 non-null   float64
dtypes: float64(1), object(1)
memory usage: 144.3+ KB
```

Repayment test data is ready to be merged now prepairing account account balance test data

In [79]:
```
# Loading Monthly balance Data
test_balance = pd.read_csv('test_monthly_balance_base.csv')
```

In [80]:
```
test_balance.head()
```

Out[80]:

	loan_acc_num	date	balance_amount
0	LN14086568	2006-12-13	9014.212689
1	LN14086568	2007-01-12	28129.516540
2	LN14086568	2007-02-11	10820.366663
3	LN14086568	2007-03-13	32491.477851
4	LN14086568	2007-04-12	24982.192310

In [81]:
```
# Dropping the date column as we dont need it.
test_balance = test_balance.drop(['date'], axis = 1)
```

In [82]:
```
test_balance.head()
```

Out[82]:

	loan_acc_num	balance_amount
0	LN14086568	9014.212689
1	LN14086568	28129.516540
2	LN14086568	10820.366663
3	LN14086568	32491.477851
4	LN14086568	24982.192310

In [83]:
```
# rounding off to two decimal place 'balance_amount' column.
test_balance['balance_amount'] = test_balance['balance_amount'].round(2)
```

In [84]:
```
test_balance.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 801407 entries, 0 to 801406
Data columns (total 2 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   loan_acc_num    801407 non-null  object 
 1   balance_amount  801407 non-null  float64
dtypes: float64(1), object(1)
memory usage: 12.2+ MB
```

In [85]:
```
# Finding average balance in each loan account.
group_balance = test_balance.groupby(by=['loan_acc_num']).mean()
```

In [86]:
```
group_balance = group_balance.reset_index()
```

In [87]:
```
group_balance.head()
```

Out[87]:

	loan_acc_num	balance_amount
0	LN10011015	25.087949
1	LN10028091	62.524848
2	LN10033713	182.412149
3	LN10045654	1838.350449
4	LN10051605	3374.170137

In [88]:
```
group_balance.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9940 entries, 0 to 9939
Data columns (total 2 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   loan_acc_num    9940 non-null   object 
 1   balance_amount  9940 non-null   float64
dtypes: float64(1), object(1)
memory usage: 155.4+ KB
```

Now all 3 test sets are ready to be merged together

In [89]:
```
# Merging 'test-main_base' with 'group_test_repayment' data set.
test_data1 = test_main_base.merge(group_test_repayment, how = 'outer', on = 'loan_
```

In [90]:
```
## Merging 'test_data1' with 'group_balance' data set.
test_data = test_data1.merge(group_balance, how = 'outer', on = 'loan_acc_num')
```

In [91]:
```
test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   loan_acc_num       10000 non-null  object 
 1   loan_type          10000 non-null  object 
 2   loan_amount        10000 non-null  int64  
 3   collateral_value   10000 non-null  float64
 4   cheque_bounces     10000 non-null  int64  
 5   number_of_loans    10000 non-null  int64  
 6   missed_repayments  10000 non-null  int64  
 7   vintage_in_months  10000 non-null  int64  
 8   tenure_years       10000 non-null  int64  
 9   interest           10000 non-null  float64
 10  monthly_emi        10000 non-null  float64
 11  loan_tenure        10000 non-null  int32  
 12  repayment_amount   9232 non-null   float64
 13  balance_amount     9943 non-null   float64
dtypes: float64(5), int32(1), int64(6), object(2)
memory usage: 1.1+ MB
```

Addressing nul values

In [92]:
```
# Filling nul/ na values in 'repayment_amount'.
test_data['repayment_amount'] = test_data['repayment_amount'].fillna(0)
```

In [93]:
```
# Filling nul/ na values in 'balance_amount'.
test_data['balance_amount'] = test_data['balance_amount'].fillna(0)
```

In [94]:
```
test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   loan_acc_num       10000 non-null  object 
 1   loan_type          10000 non-null  object 
 2   loan_amount        10000 non-null  int64  
 3   collateral_value   10000 non-null  float64
 4   cheque_bounces     10000 non-null  int64  
 5   number_of_loans    10000 non-null  int64  
 6   missed_repayments  10000 non-null  int64  
 7   vintage_in_months  10000 non-null  int64  
 8   tenure_years       10000 non-null  int64  
 9   interest           10000 non-null  float64
 10  monthly_emi        10000 non-null  float64
 11  loan_tenure        10000 non-null  int32  
 12  repayment_amount   10000 non-null  float64
 13  balance_amount     10000 non-null  float64
dtypes: float64(5), int32(1), int64(6), object(2)
memory usage: 1.1+ MB
```

In [95]:
```
# creating LGD column
# LGD = (loan Amount -(Amount repayed + Collateral Value))/ Loan Amount
```

In [96]:
```
test_data['lgd_pct'] = (test_data['loan_amount'] - (test_data['collateral_value'] -
```

In [97]:
```
test_data.head()
```

Out[97]:

	loan_acc_num	loan_type	loan_amount	collateral_value	cheque_bounces	number_of_loans	mi
0	LN14086568	Car	1259062	10184.09	3	1	
1	LN37082418	Consumer-Durable	21731	2313.21	0	3	
2	LN42963368	Car	207660	8308.71	1	2	
3	LN54572294	Two-Wheeler	193528	26432.24	4	3	
4	LN65792799	Consumer-Durable	5980	1641.66	1	1	

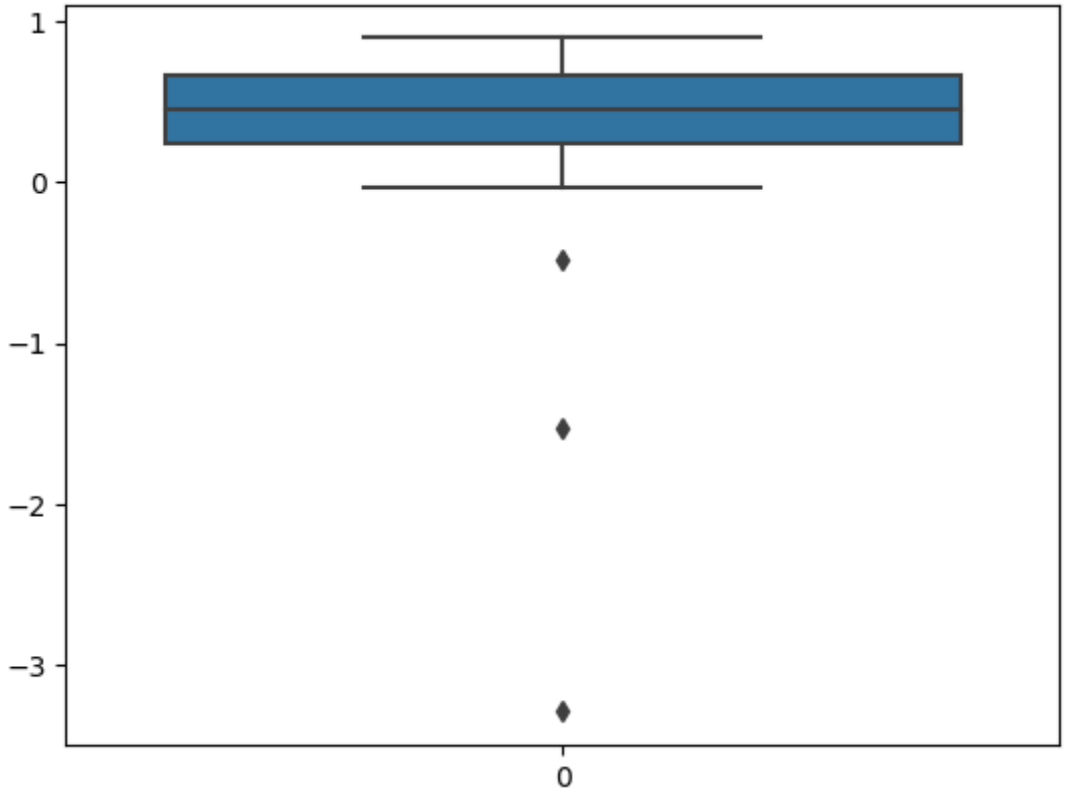
In [98]:
```
test_data.describe()
```

Out[98]:

	loan_amount	collateral_value	cheque_bounces	number_of_loans	missed_repayments	vinta
count	1.000000e+04	10000.000000	10000.00000	10000.00000	10000.000000	
mean	3.825381e+05	57759.914277	1.76000	1.49940	9.716700	
std	5.005845e+05	93155.252125	1.78312	1.26038	7.672374	
min	2.009000e+03	1.160000	0.00000	0.00000	0.000000	
25%	2.405375e+04	3335.285000	0.00000	0.00000	4.000000	
50%	1.960010e+05	20323.325000	1.00000	1.00000	8.000000	
75%	4.347512e+05	64152.422500	3.00000	2.00000	14.000000	
max	1.998735e+06	591419.920000	10.00000	6.00000	35.000000	

In [99]:
```
sns.boxplot(test_data['lgd_pct'])
```

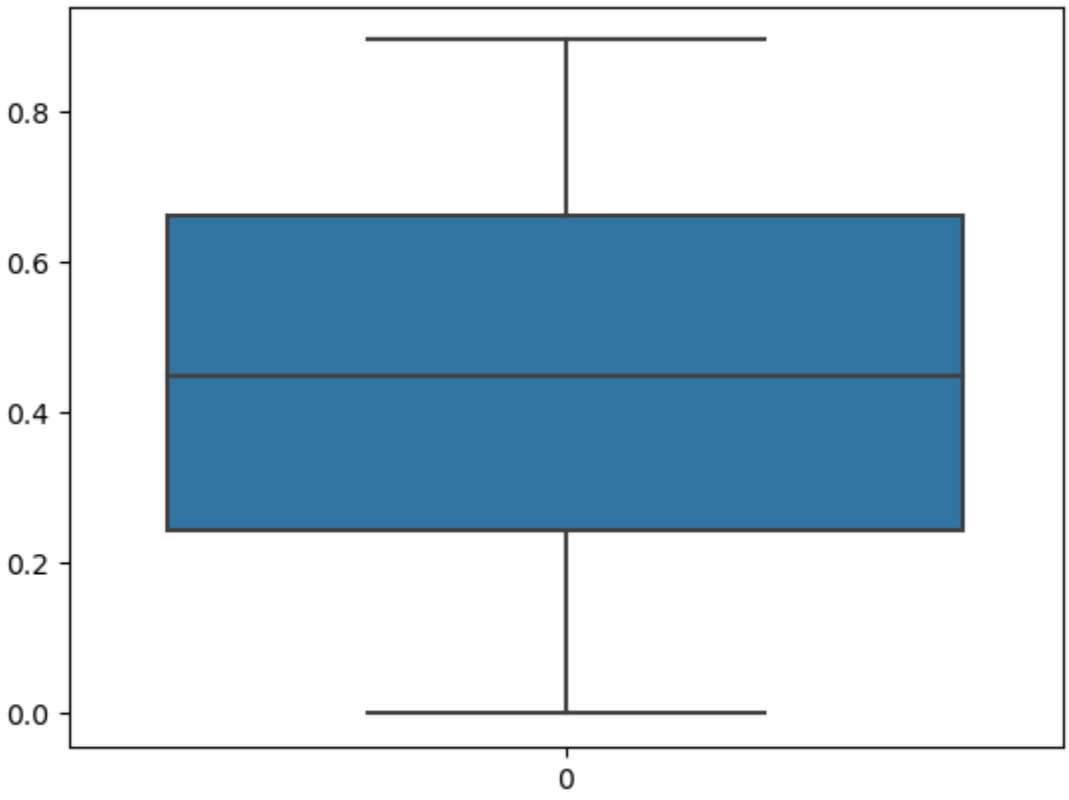
Out[99]:
```
<Axes: >
```



In [100…
```
# Repacing -ve values with 0 as LGD can not be -ve.
test_data['lgd_pct'] = test_data['lgd_pct'].apply(lambda x : x if x > 0 else 0)
```

In [101…
```
sns.boxplot(test_data['lgd_pct'])
```

Out[101]:
```
<Axes: >
```



In [102…
```
# Exploring the various variables
```

In [103…
```
data.loan_type.value_counts(normalize = True)
```

Out[103]:
```
Two-Wheeler         0.25170
Car                 0.25092
Personal            0.24898
Consumer-Durable    0.24840
Name: loan_type, dtype: float64
```

In [104…
```
# We see that the default is quite evenly distributed in all types of loans
```

Making dummy variables for the loan type !!

In [105…
```
dummies = pd.get_dummies(data['loan_type'], drop_first = True)
```

In [106…
```
data = pd.concat([data, dummies], axis=1)
```

In [107…
```
data.shape
```

Out[107]:
```
(50000, 18)
```

In [108…
```
data.drop(['loan_type'], axis=1, inplace = True)
```

In [109…
```
data.head()
```

Out[109]:

	loan_acc_num	loan_amount	collateral_value	cheque_bounces	number_of_loans	missed_repaym
0	LN79307711	21916	4929.47	3	0	
1	LN88987787	121184	10254.50	1	0	
2	LN78096023	487036	116183.86	0	1	
3	LN56862431	52125	10310.05	5	2	
4	LN77262680	8635	1051.25	0	1	

In [110…
```
num_cols = data.select_dtypes('number').columns
num_cols
```

Out[110]:
```
Index(['loan_amount', 'collateral_value', 'cheque_bounces', 'number_of_loans',
       'missed_repayments', 'vintage_in_months', 'tenure_years', 'interest',
       'monthly_emi', 'loan_tenure', 'repayment_amount', 'balance_amount',
       'lgd_pct', 'Consumer-Durable', 'Personal', 'Two-Wheeler'],
      dtype='object')
```

In [111…
```
num_cols = num_cols.drop('Consumer-Durable')
num_cols = num_cols.drop('Personal')
num_cols = num_cols.drop('Two-Wheeler')
num_cols = num_cols.drop('lgd_pct')
num_cols
```

Out[111]:
```
Index(['loan_amount', 'collateral_value', 'cheque_bounces', 'number_of_loans',
       'missed_repayments', 'vintage_in_months', 'tenure_years', 'interest',
       'monthly_emi', 'loan_tenure', 'repayment_amount', 'balance_amount'],
      dtype='object')
```

In [112…
```
#applying MinMaxScaler
scaler = MinMaxScaler()
```

```
data[num_cols] = scaler.fit_transform(data[num_cols])
```

In [113…
```
data.head()
```

Out[113]:

	loan_acc_num	loan_amount	collateral_value	cheque_bounces	number_of_loans	missed_repaym
0	LN79307711	0.009968	0.008319	0.272727	0.000000	0.07
1	LN88987787	0.059652	0.017306	0.090909	0.000000	0.00
2	LN78096023	0.242762	0.196076	0.000000	0.166667	0.26
3	LN56862431	0.025088	0.017399	0.454545	0.333333	0.23
4	LN77262680	0.003321	0.001774	0.000000	0.166667	0.07

In [114…
```
data.describe()
```

Out[114]:

	loan_amount	collateral_value	cheque_bounces	number_of_loans	missed_repayments	vinta
count	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	
mean	0.190034	0.096515	0.160431	0.251590	0.258113	
std	0.252133	0.157637	0.160016	0.209898	0.204948	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.010979	0.005619	0.000000	0.000000	0.105263	
50%	0.095440	0.033522	0.090909	0.166667	0.210526	
75%	0.215921	0.105162	0.272727	0.333333	0.394737	
max	1.000000	1.000000	1.000000	1.000000	1.000000	

Making loan column in both data sets numeric

In [115…
```
test_data['loan_acc_num'] = test_data['loan_acc_num'].str.slice(2)
```

In [116…
```
data['loan_acc_num'] = data['loan_acc_num'].str.slice(2)
```

In [117…
```
test_data['loan_acc_num'] = test_data['loan_acc_num'].apply(pd.to_numeric)
```

In [118…
```
data['loan_acc_num'] = data['loan_acc_num'].apply(pd.to_numeric)
```

Getting Dummies for the test data set.

In [119…
```
test_data.head()
```

Out[119]:

	loan_acc_num	loan_type	loan_amount	collateral_value	cheque_bounces	number_of_loans	mi
0	14086568	Car	1259062	10184.09	3	1	
1	37082418	Consumer-Durable	21731	2313.21	0	3	
2	42963368	Car	207660	8308.71	1	2	
3	54572294	Two-Wheeler	193528	26432.24	4	3	
4	65792799	Consumer-Durable	5980	1641.66	1	1	

In [120…
```
dummies = pd.get_dummies(test_data['loan_type'], drop_first= True)
```

In [121…
```
test_data = pd.concat([test_data, dummies], axis = 1)
```

In [122…
```
# Since we have the dummies column we dont need 'loan_type' column.
test_data.drop(['loan_type'], axis = 1, inplace = True)
```

In [123…
```
test_data[num_cols] = scaler.fit_transform(test_data[num_cols])
```

In [124…
```
test_data.head()
```

Out[124]:

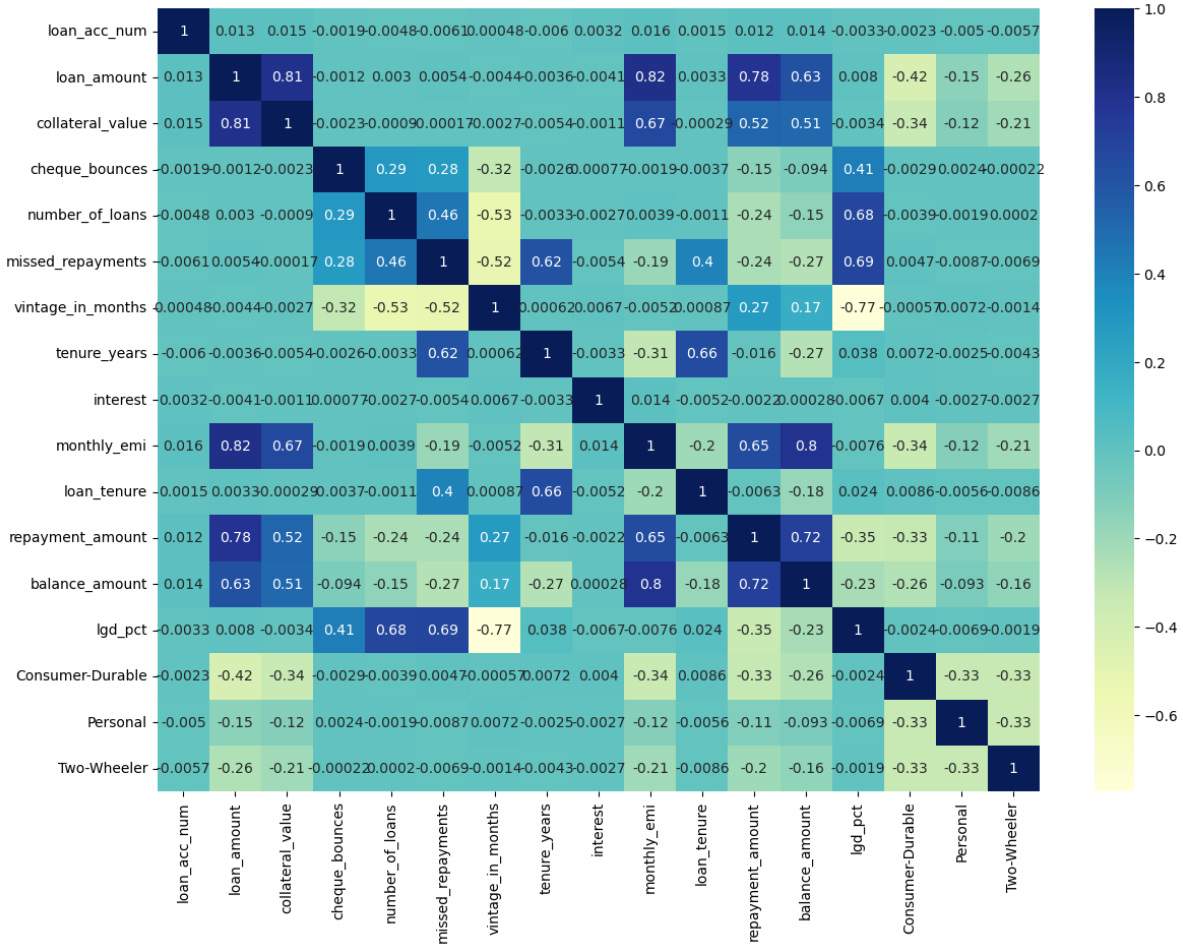
	loan_acc_num	loan_amount	collateral_value	cheque_bounces	number_of_loans	missed_repaym
0	14086568	0.629557	0.017218	0.3	0.166667	0.14
1	37082418	0.009877	0.003909	0.0	0.500000	0.74
2	42963368	0.102994	0.014047	0.1	0.333333	0.00
3	54572294	0.095917	0.044691	0.4	0.500000	0.34
4	65792799	0.001989	0.002774	0.1	0.166667	0.00

In [125…
```
test_data.shape
```

Out[125]:
```
(10000, 17)
```

In [126…
```
# Checking Corelation between variables
```

In [127…
```
plt.figure(figsize = (14, 10))
sns.heatmap(data.corr(), annot = True, cmap="YlGnBu")
plt.show()
```



In [128…
```
# We see that lot of variables, factors have strong relationship with target factor
```

Prepairing Data for Modeling

In [129…
```
X_train = data.drop('lgd_pct', axis = 1)
y_train = data['lgd_pct']
X_test = test_data.drop('lgd_pct', axis = 1)
y_test = test_data['lgd_pct']
```

Building Model

In [130…
```
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train)

lr_1 = sm.OLS(y_train, X_train_lm.astype(float)).fit()

lr_1.params
```

Out[130]:
```
const                4.169843e-01
loan_acc_num         3.041764e-12
loan_amount          1.977541e-01
collateral_value    -1.278627e-01
cheque_bounces       1.074226e-01
number_of_loans      2.069781e-01
missed_repayments    6.802345e-01
vintage_in_months   -3.760851e-01
tenure_years        -2.173195e-01
interest            -2.608713e-03
monthly_emi          1.380111e-01
loan_tenure         -1.644212e-03
repayment_amount    -3.263841e-01
balance_amount      -3.890404e-01
Consumer-Durable    -9.979395e-04
Personal            -2.201339e-04
Two-Wheeler         -5.504809e-05
dtype: float64
```

In [131…
```
print(lr_1.summary())
```

OLS Regression Results

Dep. Variable:	lgd_pct	R-squared:	0.846
Model:	OLS	Adj. R-squared:	0.846
Method:	Least Squares	F-statistic:	1.717e+04
Date:	Tue, 29 Aug 2023	Prob (F-statistic):	0.00
Time:	19:51:44	Log-Likelihood:	47017.
No. Observations:	50000	AIC:	-9.400e+04
Df Residuals:	49983	BIC:	-9.385e+04
Df Model:	16		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.4170	0.003	154.249	0.000	0.412	0.422
loan_acc_num	3.042e-12	1.63e-11	0.187	0.852	-2.88e-11	3.49e-11
loan_amount	0.1978	0.006	30.594	0.000	0.185	0.210
collateral_value	-0.1279	0.005	-25.144	0.000	-0.138	-0.118
cheque_bounces	0.1074	0.003	37.559	0.000	0.102	0.113
number_of_loans	0.2070	0.003	80.286	0.000	0.202	0.212
missed_repayments	0.6802	0.004	171.716	0.000	0.672	0.688
vintage_in_months	-0.3761	0.003	-115.308	0.000	-0.382	-0.370
tenure_years	-0.2173	0.002	-96.272	0.000	-0.222	-0.213
interest	-0.0026	0.001	-1.779	0.075	-0.005	0.000
monthly_emi	0.1380	0.008	17.730	0.000	0.123	0.153
loan_tenure	-0.0016	0.002	-0.707	0.480	-0.006	0.003
repayment_amount	-0.3264	0.007	-43.589	0.000	-0.341	-0.312
balance_amount	-0.3890	0.015	-26.654	0.000	-0.418	-0.360
Consumer-Durable	-0.0010	0.002	-0.485	0.628	-0.005	0.003
Personal	-0.0002	0.002	-0.124	0.901	-0.004	0.003
Two-Wheeler	-5.505e-05	0.002	-0.029	0.977	-0.004	0.004

Omnibus:	300.131	Durbin-Watson:	1.997
Prob(Omnibus):	0.000	Jarque-Bera (JB):	385.010
Skew:	-0.103	Prob(JB):	2.49e-84
Kurtosis:	3.378	Cond. No.	2.41e+09

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.41e+09. This might indicate that there are strong multicollinearity or other numerical problems.

Variance Inflation Factor

In [132…
```
# Check for the VIF values of the feature variables. We want VIF to be less than 5
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [133…
```
vif = pd.DataFrame()
vif['Featue'] = X_train.columns
vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.
vif['VIF'] = round(vif['VIF'],2)
vif = vif.sort_values(by = 'VIF', ascending = False)
vif
```

Out[133]:

	Featue	VIF
1	loan_amount	22.13
7	tenure_years	10.65
9	monthly_emi	10.25
5	missed_repayments	9.29
11	repayment_amount	8.07
12	balance_amount	5.51
6	vintage_in_months	5.02
2	collateral_value	4.95
0	loan_acc_num	4.87
10	loan_tenure	4.19
4	number_of_loans	3.73
8	interest	3.70
13	Consumer-Durable	3.45
15	Two-Wheeler	2.97
14	Personal	2.70
3	cheque_bounces	2.29

In [134…
```
# As we can see the P Value of 'Two+Wheeler' is very high
# we will drop the insignificant variable and test further.
```

In [135…
```
X_train = X_train.drop('Two-Wheeler',1)
```

```
C:\Users\amann\AppData\Local\Temp\ipykernel_14360\1430195737.py:1: FutureWarning:
In a future version of pandas all arguments of DataFrame.drop except for the argum
ent 'labels' will be keyword-only.
  X_train = X_train.drop('Two-Wheeler',1)
```

In [136…
```
# Running the Model again

X_train_lm = sm.add_constant(X_train)

lr_2 = sm.OLS(y_train, X_train_lm.astype(float)).fit()

print(lr_2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                lgd_pct   R-squared:                       0.846
Model:                            OLS   Adj. R-squared:                  0.846
Method:                 Least Squares   F-statistic:                 1.831e+04
Date:                Tue, 29 Aug 2023   Prob (F-statistic):               0.00
Time:                        19:51:46   Log-Likelihood:                 47017.
No. Observations:               50000   AIC:                         -9.400e+04
Df Residuals:                   49984   BIC:                         -9.386e+04
Df Model:                          15
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
const                0.4169      0.002    199.171      0.000       0.413       0.
421
loan_acc_num      3.043e-12   1.63e-11      0.187      0.852   -2.88e-11    3.49e
-11
loan_amount          0.1978      0.006     32.868      0.000       0.186       0.
210
collateral_value    -0.1279      0.005    -25.145      0.000      -0.138      -0.
118
cheque_bounces       0.1074      0.003     37.559      0.000       0.102       0.
113
number_of_loans      0.2070      0.003     80.287      0.000       0.202       0.
212
missed_repayments    0.6802      0.004    171.725      0.000       0.672       0.
688
vintage_in_months   -0.3761      0.003   -115.313      0.000      -0.382      -0.
370
tenure_years        -0.2173      0.002    -96.273      0.000      -0.222      -0.
213
interest            -0.0026      0.001     -1.779      0.075      -0.005       0.
000
monthly_emi          0.1380      0.008     17.731      0.000       0.123       0.
153
loan_tenure         -0.0016      0.002     -0.707      0.480      -0.006       0.
003
repayment_amount    -0.3264      0.007    -43.589      0.000      -0.341      -0.
312
balance_amount      -0.3890      0.015    -26.654      0.000      -0.418      -0.
360
Consumer-Durable    -0.0009      0.001     -0.791      0.429      -0.003       0.
001
Personal            -0.0002      0.001     -0.163      0.870      -0.002       0.
002
==============================================================================
Omnibus:                      300.154   Durbin-Watson:                   1.997
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              385.025
Skew:                          -0.103   Prob(JB):                     2.47e-84
Kurtosis:                       3.378   Cond. No.                     2.41e+09
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly s
pecified.
[2] The condition number is large, 2.41e+09. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [137…

```
## Checking VIF
vif = pd.DataFrame()
```

```
vif['Featue'] = X_train.columns
vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.
vif['VIF'] = round(vif['VIF'],2)
vif = vif.sort_values(by = 'VIF', ascending = False)
vif
```

Out[137]:

	Featue	VIF
1	loan_amount	20.22
7	tenure_years	10.63
9	monthly_emi	10.16
5	missed_repayments	9.15
11	repayment_amount	8.07
12	balance_amount	5.51
2	collateral_value	4.95
0	loan_acc_num	4.48
6	vintage_in_months	4.26
10	loan_tenure	4.18
4	number_of_loans	3.55
8	interest	3.53
3	cheque_bounces	2.25
13	Consumer-Durable	1.84
14	Personal	1.58

In [138…

```
# Dropping personal from the Model as P value is very high.
X_train = X_train.drop('Personal',1)
```

```
C:\Users\amann\AppData\Local\Temp\ipykernel_14360\3873789974.py:2: FutureWarning:
In a future version of pandas all arguments of DataFrame.drop except for the argum
ent 'labels' will be keyword-only.
  X_train = X_train.drop('Personal',1)
```

In [139…

```
# Running the Model again

X_train_lm = sm.add_constant(X_train)

lr_3 = sm.OLS(y_train, X_train_lm.astype(float)).fit()

print(lr_3.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                lgd_pct   R-squared:                       0.846
Model:                            OLS   Adj. R-squared:                  0.846
Method:                 Least Squares   F-statistic:                 1.962e+04
Date:                Tue, 29 Aug 2023   Prob (F-statistic):               0.00
Time:                        19:51:47   Log-Likelihood:                 47017.
No. Observations:               50000   AIC:                         -9.400e+04
Df Residuals:                   49985   BIC:                         -9.387e+04
Df Model:                          14
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
const                0.4168      0.002    205.844      0.000       0.413       0.
421
loan_acc_num      3.047e-12   1.63e-11      0.187      0.851   -2.88e-11    3.49e
-11
loan_amount          0.1979      0.006     33.076      0.000       0.186       0.
210
collateral_value    -0.1279      0.005    -25.144      0.000      -0.138      -0.
118
cheque_bounces       0.1074      0.003     37.559      0.000       0.102       0.
113
number_of_loans      0.2070      0.003     80.287      0.000       0.202       0.
212
missed_repayments    0.6802      0.004    171.729      0.000       0.672       0.
688
vintage_in_months   -0.3761      0.003   -115.315      0.000      -0.382      -0.
370
tenure_years        -0.2173      0.002    -96.274      0.000      -0.222      -0.
213
interest            -0.0026      0.001     -1.779      0.075      -0.005       0.
000
monthly_emi          0.1380      0.008     17.731      0.000       0.123       0.
153
loan_tenure         -0.0016      0.002     -0.706      0.480      -0.006       0.
003
repayment_amount    -0.3264      0.007    -43.590      0.000      -0.341      -0.
312
balance_amount      -0.3890      0.015    -26.655      0.000      -0.418      -0.
360
Consumer-Durable    -0.0009      0.001     -0.801      0.423      -0.003       0.
001
==============================================================================
Omnibus:                      300.160   Durbin-Watson:                   1.997
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              385.045
Skew:                          -0.103   Prob(JB):                     2.45e-84
Kurtosis:                       3.378   Cond. No.                     2.41e+09
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly s
pecified.
[2] The condition number is large, 2.41e+09. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [140…

```
## Checking VIF
vif = pd.DataFrame()
vif['Featue'] = X_train.columns
vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.
```

```
vif['VIF'] = round(vif['VIF'],2)
vif = vif.sort_values(by = 'VIF', ascending = False)
vif
```

Out[140]:

	Featue	VIF
1	loan_amount	19.96
7	tenure_years	10.63
9	monthly_emi	10.15
5	missed_repayments	9.13
11	repayment_amount	8.07
12	balance_amount	5.51
2	collateral_value	4.95
0	loan_acc_num	4.41
10	loan_tenure	4.18
6	vintage_in_months	4.12
4	number_of_loans	3.52
8	interest	3.50
3	cheque_bounces	2.24
13	Consumer-Durable	1.55

In [141…

```
# Dropping 'loan_acc_num' from the Model as P value is very high.
X_train = X_train.drop('loan_acc_num',1)
```

```
C:\Users\amann\AppData\Local\Temp\ipykernel_14360\2224676173.py:2: FutureWarning:
In a future version of pandas all arguments of DataFrame.drop except for the argum
ent 'labels' will be keyword-only.
  X_train = X_train.drop('loan_acc_num',1)
```

In [142…

```
# Running the Model again

X_train_lm = sm.add_constant(X_train)

lr_4 = sm.OLS(y_train, X_train_lm.astype(float)).fit()

print(lr_4.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                lgd_pct   R-squared:                       0.846
Model:                            OLS   Adj. R-squared:                  0.846
Method:                 Least Squares   F-statistic:                 2.113e+04
Date:                Tue, 29 Aug 2023   Prob (F-statistic):               0.00
Time:                        19:51:48   Log-Likelihood:                 47017.
No. Observations:               50000   AIC:                         -9.401e+04
Df Residuals:                   49986   BIC:                         -9.388e+04
Df Model:                          13
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
const                0.4170      0.002    229.416      0.000       0.413       0.
421
loan_amount          0.1979      0.006     33.076      0.000       0.186       0.
210
collateral_value    -0.1279      0.005    -25.144      0.000      -0.138      -0.
118
cheque_bounces       0.1074      0.003     37.559      0.000       0.102       0.
113
number_of_loans      0.2070      0.003     80.288      0.000       0.202       0.
212
missed_repayments    0.6802      0.004    171.730      0.000       0.672       0.
688
vintage_in_months   -0.3761      0.003   -115.317      0.000      -0.382      -0.
370
tenure_years        -0.2173      0.002    -96.277      0.000      -0.222      -0.
213
interest            -0.0026      0.001     -1.778      0.075      -0.005       0.
000
monthly_emi          0.1380      0.008     17.732      0.000       0.123       0.
153
loan_tenure         -0.0016      0.002     -0.705      0.481      -0.006       0.
003
repayment_amount    -0.3264      0.007    -43.590      0.000      -0.341      -0.
312
balance_amount      -0.3890      0.015    -26.655      0.000      -0.418      -0.
360
Consumer-Durable    -0.0009      0.001     -0.800      0.424      -0.003       0.
001
==============================================================================
Omnibus:                      300.261   Durbin-Watson:                   1.997
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              385.202
Skew:                          -0.103   Prob(JB):                     2.26e-84
Kurtosis:                       3.378   Cond. No.                         55.7
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly s
pecified.
```

In [143…

```
## Checking VIF
vif = pd.DataFrame()
vif['Featue'] = X_train.columns
vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.s
vif['VIF'] = round(vif['VIF'],2)
vif = vif.sort_values(by = 'VIF', ascending = False)
vif
```

Out[143]:

	Featue	VIF
0	loan_amount	19.95
6	tenure_years	10.62
8	monthly_emi	10.09
4	missed_repayments	9.03
10	repayment_amount	8.06
11	balance_amount	5.51
1	collateral_value	4.94
9	loan_tenure	4.17
5	vintage_in_months	3.63
3	number_of_loans	3.42
7	interest	3.37
2	cheque_bounces	2.21
12	Consumer-Durable	1.53

In [144…

```
# Dropping 'loan_amount' as VIF is too high.
X_train = X_train.drop('loan_amount',1)
```

```
C:\Users\amann\AppData\Local\Temp\ipykernel_14360\1034633755.py:2: FutureWarning:
In a future version of pandas all arguments of DataFrame.drop except for the argum
ent 'labels' will be keyword-only.
  X_train = X_train.drop('loan_amount',1)
```

In [145…

```
# Running the Model again

X_train_lm = sm.add_constant(X_train)

lr_5 = sm.OLS(y_train, X_train_lm.astype(float)).fit()

print(lr_5.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                lgd_pct   R-squared:                       0.843
Model:                            OLS   Adj. R-squared:                  0.843
Method:                 Least Squares   F-statistic:                 2.231e+04
Date:                Tue, 29 Aug 2023   Prob (F-statistic):               0.00
Time:                        19:51:49   Log-Likelihood:                 46476.
No. Observations:               50000   AIC:                         -9.293e+04
Df Residuals:                   49987   BIC:                         -9.281e+04
Df Model:                          12
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
const                0.4145      0.002    225.778      0.000       0.411       0.
418
collateral_value    -0.0165      0.004     -4.281      0.000      -0.024      -0.
009
cheque_bounces       0.1091      0.003     37.740      0.000       0.103       0.
115
number_of_loans      0.2107      0.003     80.911      0.000       0.206       0.
216
missed_repayments    0.7082      0.004    181.061      0.000       0.701       0.
716
vintage_in_months   -0.3824      0.003   -116.198      0.000      -0.389      -0.
376
tenure_years        -0.2154      0.002    -94.416      0.000      -0.220      -0.
211
interest            -0.0040      0.001     -2.728      0.006      -0.007      -0.
001
monthly_emi          0.2996      0.006     48.918      0.000       0.288       0.
312
loan_tenure         -0.0011      0.002     -0.487      0.626      -0.006       0.
003
repayment_amount    -0.1459      0.005    -28.150      0.000      -0.156      -0.
136
balance_amount      -0.6033      0.013    -45.624      0.000      -0.629      -0.
577
Consumer-Durable    -0.0056      0.001     -5.153      0.000      -0.008      -0.
003
==============================================================================
Omnibus:                      316.145   Durbin-Watson:                   1.996
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              414.463
Skew:                          -0.100   Prob(JB):                     1.00e-90
Kurtosis:                       3.398   Cond. No.                         45.2
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly s
pecified.
```

In [146…

```
## Checking VIF
vif = pd.DataFrame()
vif['Featue'] = X_train.columns
vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.s
vif['VIF'] = round(vif['VIF'],2)
vif = vif.sort_values(by = 'VIF', ascending = False)
vif
```

Out[146]:

	Featue	VIF
5	tenure_years	10.62
3	missed_repayments	8.63
7	monthly_emi	6.08
10	balance_amount	4.43
8	loan_tenure	4.17
9	repayment_amount	3.79
4	vintage_in_months	3.59
2	number_of_loans	3.41
6	interest	3.37
0	collateral_value	2.78
1	cheque_bounces	2.21
11	Consumer-Durable	1.50

In [147…

```
# Dropping 'loan_tenure' as VIF is too high.
X_train = X_train.drop('loan_tenure',1)
```

```
C:\Users\amann\AppData\Local\Temp\ipykernel_14360\213307851.py:2: FutureWarning: I
n a future version of pandas all arguments of DataFrame.drop except for the argume
nt 'labels' will be keyword-only.
  X_train = X_train.drop('loan_tenure',1)
```

In [148…

```
# Running the Model again

X_train_lm = sm.add_constant(X_train)

lr_6 = sm.OLS(y_train, X_train_lm.astype(float)).fit()

print(lr_6.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                lgd_pct   R-squared:                       0.843
Model:                            OLS   Adj. R-squared:                  0.843
Method:                 Least Squares   F-statistic:                 2.434e+04
Date:                Tue, 29 Aug 2023   Prob (F-statistic):               0.00
Time:                        19:51:50   Log-Likelihood:                 46476.
No. Observations:               50000   AIC:                         -9.293e+04
Df Residuals:                   49988   BIC:                         -9.282e+04
Df Model:                          11
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
const                0.4144      0.002    226.417      0.000       0.411       0.
418
collateral_value    -0.0165      0.004     -4.281      0.000      -0.024      -0.
009
cheque_bounces       0.1091      0.003     37.741      0.000       0.103       0.
115
number_of_loans      0.2106      0.003     80.910      0.000       0.206       0.
216
missed_repayments    0.7082      0.004    181.062      0.000       0.701       0.
716
vintage_in_months   -0.3824      0.003   -116.199      0.000      -0.389      -0.
376
tenure_years        -0.2159      0.002   -106.536      0.000      -0.220      -0.
212
interest            -0.0040      0.001     -2.726      0.006      -0.007      -0.
001
monthly_emi          0.2996      0.006     48.918      0.000       0.288       0.
312
repayment_amount    -0.1459      0.005    -28.153      0.000      -0.156      -0.
136
balance_amount      -0.6032      0.013    -45.623      0.000      -0.629      -0.
577
Consumer-Durable    -0.0056      0.001     -5.157      0.000      -0.008      -0.
003
==============================================================================
Omnibus:                      316.246   Durbin-Watson:                   1.996
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              414.606
Skew:                          -0.100   Prob(JB):                     9.32e-91
Kurtosis:                       3.399   Cond. No.                         44.1
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly s
pecified.
```

In [149…

```
## Checking VIF
vif = pd.DataFrame()
vif['Featue'] = X_train.columns
vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.
vif['VIF'] = round(vif['VIF'],2)
vif = vif.sort_values(by = 'VIF', ascending = False)
vif
```

Out[149]:

	Featue	VIF
3	missed_repayments	8.63
5	tenure_years	8.31
7	monthly_emi	6.08
9	balance_amount	4.43
8	repayment_amount	3.79
4	vintage_in_months	3.57
2	number_of_loans	3.41
6	interest	3.36
0	collateral_value	2.78
1	cheque_bounces	2.21
10	Consumer-Durable	1.50

In [150…

```
# Dropping 'missed_repayments' as VIF is too high.
X_train = X_train.drop('missed_repayments',1)
```

```
C:\Users\amann\AppData\Local\Temp\ipykernel_14360\3413836430.py:2: FutureWarning:
In a future version of pandas all arguments of DataFrame.drop except for the argum
ent 'labels' will be keyword-only.
  X_train = X_train.drop('missed_repayments',1)
```

In [151…

```
# Running the Model again

X_train_lm = sm.add_constant(X_train)

lr_7 = sm.OLS(y_train, X_train_lm.astype(float)).fit()

print(lr_7.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                lgd_pct   R-squared:                       0.740
Model:                            OLS   Adj. R-squared:                  0.739
Method:                 Least Squares   F-statistic:                 1.419e+04
Date:                Tue, 29 Aug 2023   Prob (F-statistic):               0.00
Time:                        19:51:50   Log-Likelihood:                 33869.
No. Observations:               50000   AIC:                         -6.772e+04
Df Residuals:                   49989   BIC:                         -6.762e+04
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
const                0.4906      0.002    214.028      0.000       0.486       0.
495
collateral_value    -0.0047      0.005     -0.949      0.343      -0.014       0.
005
cheque_bounces       0.1882      0.004     51.187      0.000       0.181       0.
195
number_of_loans      0.3645      0.003    115.089      0.000       0.358       0.
371
vintage_in_months   -0.6557      0.004   -174.236      0.000      -0.663      -0.
648
tenure_years         0.0516      0.002     28.920      0.000       0.048       0.
055
interest            -0.0048      0.002     -2.512      0.012      -0.009      -0.
001
monthly_emi          0.3494      0.008     44.374      0.000       0.334       0.
365
repayment_amount    -0.3251      0.007    -49.655      0.000      -0.338      -0.
312
balance_amount      -0.4202      0.017    -24.772      0.000      -0.453      -0.
387
Consumer-Durable    -0.0098      0.001     -7.075      0.000      -0.013      -0.
007
==============================================================================
Omnibus:                        4.084   Durbin-Watson:                   2.011
Prob(Omnibus):                  0.130   Jarque-Bera (JB):                4.036
Skew:                          -0.013   Prob(JB):                        0.133
Kurtosis:                       2.964   Cond. No.                         43.1
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly s
pecified.
```

In [152…

```
## Checking VIF
vif = pd.DataFrame()
vif['Featue'] = X_train.columns
vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.s
vif['VIF'] = round(vif['VIF'],2)
vif = vif.sort_values(by = 'VIF', ascending = False)
vif
```

Out[152]:

	Featue	VIF
6	monthly_emi	6.02
8	balance_amount	4.40
7	repayment_amount	3.64
5	interest	3.33
4	tenure_years	3.26
3	vintage_in_months	2.95
0	collateral_value	2.78
2	number_of_loans	2.69
1	cheque_bounces	2.10
9	Consumer-Durable	1.50

In [153…

```
# Dropping 'monthly_emi' as VIF is too high, above 5.
X_train = X_train.drop('monthly_emi',1)
```

```
C:\Users\amann\AppData\Local\Temp\ipykernel_14360\2372411855.py:2: FutureWarning:
In a future version of pandas all arguments of DataFrame.drop except for the argum
ent 'labels' will be keyword-only.
  X_train = X_train.drop('monthly_emi',1)
```

In [154…

```
X_train_lm = sm.add_constant(X_train)

lr_8 = sm.OLS(y_train, X_train_lm.astype(float)).fit()

print(lr_8.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                lgd_pct   R-squared:                       0.729
Model:                            OLS   Adj. R-squared:                  0.729
Method:                 Least Squares   F-statistic:                 1.496e+04
Date:                Tue, 29 Aug 2023   Prob (F-statistic):               0.00
Time:                        19:51:51   Log-Likelihood:                 32903.
No. Observations:               50000   AIC:                        -6.579e+04
Df Residuals:                   49990   BIC:                        -6.570e+04
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
const                0.5112      0.002    223.331      0.000       0.507       0.
516
collateral_value     0.0995      0.004     22.374      0.000       0.091       0.
108
cheque_bounces       0.1950      0.004     52.077      0.000       0.188       0.
202
number_of_loans      0.3798      0.003    118.338      0.000       0.373       0.
386
vintage_in_months   -0.6820      0.004   -179.994      0.000      -0.689      -0.
675
tenure_years         0.0254      0.002     14.793      0.000       0.022       0.
029
interest            -0.0021      0.002     -1.059      0.290      -0.006       0.
002
repayment_amount    -0.2617      0.007    -40.168      0.000      -0.274      -0.
249
balance_amount      -0.0143      0.015     -0.979      0.327      -0.043       0.
014
Consumer-Durable    -0.0164      0.001    -11.616      0.000      -0.019      -0.
014
==============================================================================
Omnibus:                        0.459   Durbin-Watson:                   2.012
Prob(Omnibus):                  0.795   Jarque-Bera (JB):                0.474
Skew:                          -0.001   Prob(JB):                        0.789
Kurtosis:                       2.985   Cond. No.                         36.1
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly s
pecified.
```

In [155…

```
## Checking VIF
vif = pd.DataFrame()
vif['Featue'] = X_train.columns
vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.s
vif['VIF'] = round(vif['VIF'],2)
vif = vif.sort_values(by = 'VIF', ascending = False)
vif
```

Out[155]:

	Featue	VIF
6	repayment_amount	3.47
5	interest	3.28
7	balance_amount	3.06
4	tenure_years	3.01
3	vintage_in_months	2.94
2	number_of_loans	2.52
0	collateral_value	2.13
1	cheque_bounces	2.08
8	Consumer-Durable	1.49

In [156…
```
X_train_lm.head()
```

Out[156]:

	const	collateral_value	cheque_bounces	number_of_loans	vintage_in_months	tenure_years	in
0	1.0	0.008319	0.272727	0.000000	0.460905	0.25	0.3
1	1.0	0.017306	0.090909	0.000000	0.580247	0.25	0.5
2	1.0	0.196076	0.000000	0.166667	0.028807	0.50	0.9
3	1.0	0.017399	0.454545	0.333333	0.061728	0.25	0.2
4	1.0	0.001774	0.000000	0.166667	0.193416	0.25	0.2

In [157…
```
# We can say that our our OLS Regression model is ready to be implemented.
```

In [158…
```
# Prepare Test Data for implementing the model.
X_test = X_test.drop(['Two-Wheeler', 'loan_amount', 'loan_tenure', 'missed_repayme
```

In [159…
```
X_test = X_test.drop(['loan_acc_num','Personal'],1)
```

```
C:\Users\amann\AppData\Local\Temp\ipykernel_14360\4152939079.py:1: FutureWarning:
In a future version of pandas all arguments of DataFrame.drop except for the argum
ent 'labels' will be keyword-only.
  X_test = X_test.drop(['loan_acc_num','Personal'],1)
```

In [160…
```
# Adding Constant variable to test dataframe
X_test_fnl = sm.add_constant(X_test)
```

In [161…
```
X_test_fnl.head()
```

Out[161]:

	const	collateral_value	cheque_bounces	number_of_loans	vintage_in_months	tenure_years	in
0	1.0	0.017218	0.3	0.166667	0.400000	0.00	0.5
1	1.0	0.003909	0.0	0.500000	0.109091	1.00	0.6
2	1.0	0.014047	0.1	0.333333	0.218182	0.00	0.8
3	1.0	0.044691	0.4	0.500000	0.000000	0.25	0.6
4	1.0	0.002774	0.1	0.166667	0.622727	0.00	0.9

In [162…
```
y_pred_fnl = lr_8.predict(X_test_fnl)
```

In [163…
```
from sklearn.metrics import r2_score
```

In [164…
```
r2_score(y_test, y_pred_fnl)
```

Out[164]:
```
0.7187743649739935
```

In [165…
```
sklearn.metrics.mean_squared_error(y_test, y_pred_fnl)
```

Out[165]:
```
0.016133911231650675
```

Now trying Decision Tree Modeling

In [166…
```
X_train = data.drop('lgd_pct', axis = 1)
y_train = data['lgd_pct']
X_test = test_data.drop('lgd_pct', axis = 1)
y_test = test_data['lgd_pct']
```

In [167…
```
X_train.shape
```

Out[167]:
```
(50000, 16)
```

In [168…
```
X_test.shape
```

Out[168]:
```
(10000, 16)
```

In [169…
```
y_train.shape
```

Out[169]:
```
(50000,)
```

In [170…
```
y_test.shape
```

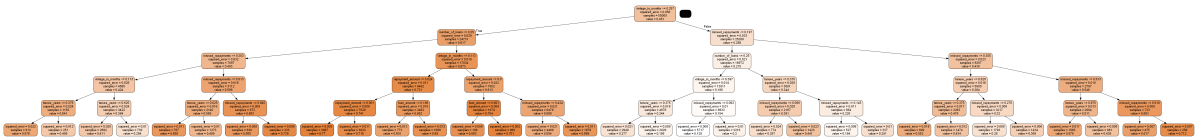
Out[170]:
```
(10000,)
```

In [171…
```
#from sklearn.ensemble import Decision Tree.
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, reca
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import export_graphviz
from IPython.display import Image
from sklearn import tree
```

In [172…
```
dt = DecisionTreeRegressor(random_state=45, max_depth=5, min_samples_leaf = 20)
model = dt.fit(X_train, y_train)

y_pred_dt1 = dt.predict(X_test)
```

In [173…
```
from IPython.display import Image
from six import StringIO
from sklearn.tree import export_graphviz
import pydotplus, graphviz


dot_data = StringIO()

export_graphviz(model, out_file = dot_data, filled = True, rounded = True,
                feature_names = X_train.columns)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[173]:



In [174…
```
r2_score(y_test, y_pred_dt1)
```

Out[174]:
```
0.7814103580444152
```

In [175…
```
sklearn.metrics.mean_squared_error(y_test, y_pred_dt1)
```

Out[175]:
```
0.012540485077555518
```

WE find that our r2 score has improved slighly from 0.71 to 0.78.

Checking performance with Randome Forest

In [176…
```
from sklearn.ensemble import RandomForestRegressor
```

In [177…
```
rfr = RandomForestRegressor(n_estimators = 100, random_state = 42)

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

In [178…
```
y_pred_rfr1 = rfr.predict(X_test)
```

In [179…
```
sample_tree = rfr.estimators_[24]
```

In [180…
```
r2_score(y_test, y_pred_rfr1)
```

Out[180]:
```
0.906434623276438
```

We see a significant improvement in r2Score with adoption of Randome Forest Regressor.

In [181…
```
sklearn.metrics.mean_squared_error(y_test, y_pred_rfr1)
```

Out[181]:
```
0.005367844514865458
```

In [ ]: