JESSICA R HOMBAL

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
data = pd.read_csv('LoanData.csv',parse_dates=['Date.of.Birth','DisbursalDate'])
data.head(10)
```

Out[2]:

	disbursed_amount	asset_cost	ltv	branch_id	Date.of.Birth	Employment.Type	DisbursalDate	MobileNo_Avl_Fla
0	36439	65850	56.19	64	1990-06-14	Self employed	2018-09-28	
1	48749	69303	72.15	67	1991-01-01	Salaried	2018-10-09	
2	55348	66340	85.00	2	1993-08-16	Self employed	2018-08-31	
3	48849	64133	77.96	217	1989-01-01	Self employed	2018-10-13	
4	40394	59386	70.72	74	1974-12-31	Self employed	2018-09-14	
5	51803	67466	79.30	162	2064-11-23	Self employed	2018-08-17	
6	61947	109094	58.21	251	1989-01-10	Self employed	2018-08-16	
7	51301	61815	85.00	67	1995-01-01	Salaried	2018-08-26	
8	65882	80461	84.51	255	1994-06-15	Self employed	2018-10-15	
9	34639	69717	50.49	34	1982-11-23	Self employed	2018-10-26	

Descriptive Statistic

In [3]:

```
data.shape
```

Out[3]:

```
(23315, 18)
```

In [4]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23315 entries, 0 to 23314
Data columns (total 18 columns):
 #   Column                               Non-Null Count  Dtype
---  ------                               --------------  -----
 0   disbursed_amount                     23315 non-null  int64
 1   asset_cost                           23315 non-null  int64
 2   ltv                                  23315 non-null  float64
 3   branch_id                            23315 non-null  int64
 4   Date.of.Birth                        23315 non-null  datetime64[ns]
 5   Employment.Type                      22545 non-null  object
 6   DisbursalDate                        23315 non-null  datetime64[ns]
 7   MobileNo_Avl_Flag                    23315 non-null  int64
 8   Aadhar_flag                          23315 non-null  int64
```

```
 9   PAN_flag                             23315 non-null  int64
 10  VoterID_flag                         23315 non-null  int64
 11  Driving_flag                         23315 non-null  int64
 12  Passport_flag                        23315 non-null  int64
 13  PERFORM_CNS.SCORE                    23315 non-null  int64
 14  DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS  23315 non-null  int64
 15  CREDIT.HISTORY.LENGTH                23315 non-null  object
 16  NO.OF_INQUIRIES                      23315 non-null  int64
 17  loan_default                         23315 non-null  int64
dtypes: datetime64[ns](2), float64(1), int64(13), object(2)
memory usage: 3.2+ MB
```

In [5]:

```
data.columns = data.columns.str.lower()
data.columns
```

Out[5]:

```
Index(['disbursed_amount', 'asset_cost', 'ltv', 'branch_id', 'date.of.birth',
       'employment.type', 'disbursaldate', 'mobileno_avl_flag', 'aadhar_flag',
       'pan_flag', 'voterid_flag', 'driving_flag', 'passport_flag',
       'perform_cns.score', 'delinquent.accts.in.last.six.months',
       'credit.history.length', 'no.of_inquiries', 'loan_default'],
      dtype='object')
```

In [6]:

```
data.isna().sum()
```

Out[6]:

```
disbursed_amount                         0
asset_cost                               0
ltv                                      0
branch_id                                0
date.of.birth                            0
employment.type                        770
disbursaldate                            0
mobileno_avl_flag                        0
aadhar_flag                              0
pan_flag                                 0
voterid_flag                             0
driving_flag                             0
passport_flag                            0
perform_cns.score                        0
delinquent.accts.in.last.six.months      0
credit.history.length                    0
no.of_inquiries                          0
loan_default                             0
dtype: int64
```

Employment.Type attribute has 770 missing values.

Missing values Treatment

In [7]:

```
data['employment.type'].mode()
```

Out[7]:

```
0    Self employed
dtype: object
```

In [8]:

```
data.fillna(data['employment.type'].mode()[0], inplace=True)
```

In [9]:

```
data['employment.type'].isna().sum()
```

Out[9]:

```
0
```

In [10]:

```
data['loan_default'].value_counts(normalize=True)
```

Out[10]:
```
0    0.780142
1    0.219858
Name: loan_default, dtype: float64
```

The data is imbalanced.

Dealing with Dates

In [11]:
```
data.head(10)
```

Out[11]:

	disbursed_amount	asset_cost	ltv	branch_id	date.of.birth	employment.type	disbursaldate	mobileno_avl_flag
0	36439	65850	56.19	64	1990-06-14	Self employed	2018-09-28	1
1	48749	69303	72.15	67	1991-01-01	Salaried	2018-10-09	1
2	55348	66340	85.00	2	1993-08-16	Self employed	2018-08-31	1
3	48849	64133	77.96	217	1989-01-01	Self employed	2018-10-13	1
4	40394	59386	70.72	74	1974-12-31	Self employed	2018-09-14	1
5	51803	67466	79.30	162	2064-11-23	Self employed	2018-08-17	1
6	61947	109094	58.21	251	1989-01-10	Self employed	2018-08-16	1
7	51301	61815	85.00	67	1995-01-01	Salaried	2018-08-26	1
8	65882	80461	84.51	255	1994-06-15	Self employed	2018-10-15	1
9	34639	69717	50.49	34	1982-11-23	Self employed	2018-10-26	1

In Date.of.Birth column, index = 5 shows 2064 instead of 1964 which is misinterpreted by the compiler. There are 1000 such values. And also we have years like 28-06-00, which is correctly interpreted as 2000-06-28. But just by replacing first two characters of all the year by 19 will even change 2000 to 1900. Keeping those in mind I've replaced the years by taking a condition. If year is greater than 1920 then only replace first two characters by 19. Else keeping it 20 so that 2000 or 2003 won't change. So will change them to appropriate values.

In [12]:
```
dob = data['date.of.birth'].apply(lambda x: str(x))
```

In [13]:
```
DOB = []
for dates in dob:
    if int(dates[2:4]) >= 20:
        dates = "19" + dates[2:]
    DOB.append(dates)
DOB[0:10]
```

Out[13]:
```
['1990-06-14 00:00:00',
 '1991-01-01 00:00:00',
 '1993-08-16 00:00:00',
 '1989-01-01 00:00:00',
 '1974-12-31 00:00:00',
 '1964-11-23 00:00:00',
 '1989-01-10 00:00:00',
 '1995-01-01 00:00:00',
 '1994-06-15 00:00:00',
 '1982-11-23 00:00:00']
```

In [14]:
```
dob[5][0:]
```

Out[14]: '2064-11-23 00:00:00'

In [15]:
```
DOB[5][0:]
```

Out[15]: '1964-11-23 00:00:00'

In [16]:
```
dob[152][0:]
```

Out[16]: '2000-06-28 00:00:00'

In [17]:
```
DOB[152][0:]
```

Out[17]: '2000-06-28 00:00:00'

In [18]:
```
data['date.of.birth'] = DOB
```

In [19]:
```
data.head(10)
```

Out[19]:

	disbursed_amount	asset_cost	ltv	branch_id	date.of.birth	employment.type	disbursaldate	mobileno_avl_flag
0	36439	65850	56.19	64	1990-06-14 00:00:00	Self employed	2018-09-28	1
1	48749	69303	72.15	67	1991-01-01 00:00:00	Salaried	2018-10-09	1
2	55348	66340	85.00	2	1993-08-16 00:00:00	Self employed	2018-08-31	1
3	48849	64133	77.96	217	1989-01-01 00:00:00	Self employed	2018-10-13	1
4	40394	59386	70.72	74	1974-12-31 00:00:00	Self employed	2018-09-14	1
5	51803	67466	79.30	162	1964-11-23 00:00:00	Self employed	2018-08-17	1
6	61947	109094	58.21	251	1989-01-10 00:00:00	Self employed	2018-08-16	1
7	51301	61815	85.00	67	1995-01-01 00:00:00	Salaried	2018-08-26	1
8	65882	80461	84.51	255	1994-06-15 00:00:00	Self employed	2018-10-15	1
9	34639	69717	50.49	34	1982-11-23 00:00:00	Self employed	2018-10-26	1

In [20]:
```
data['date.of.birth'] = pd.to_datetime(data['date.of.birth'])
```

In [21]:
```
data.head(10)
```

Out[21]:

	disbursed_amount	asset_cost	ltv	branch_id	date.of.birth	employment.type	disbursaldate	mobileno_avl_flag
0	36439	65850	56.19	64	1990-06-14	Self employed	2018-09-28	1

	disbursed_amount	asset_cost	ltv	branch_id	date.of.birth	employment.type	disbursaldate	mobileno_avl_flag
1	48749	69303	72.15	67	1991-01-01	Salaried	2018-10-09	1
2	55348	66340	85.00	2	1993-08-16	Self employed	2018-08-31	1
3	48849	64133	77.96	217	1989-01-01	Self employed	2018-10-13	1
4	40394	59386	70.72	74	1974-12-31	Self employed	2018-09-14	1
5	51803	67466	79.30	162	1964-11-23	Self employed	2018-08-17	1
6	61947	109094	58.21	251	1989-01-10	Self employed	2018-08-16	1
7	51301	61815	85.00	67	1995-01-01	Salaried	2018-08-26	1
8	65882	80461	84.51	255	1994-06-15	Self employed	2018-10-15	1
9	34639	69717	50.49	34	1982-11-23	Self employed	2018-10-26	1

credit.history.length attribute can be converted to total months.

In [22]:

```
y = data["credit.history.length"].str.extract("(\d+)yrs", expand=False).astype(int)
m = data["credit.history.length"].str.extract("(\d+)mon", expand=False).astype(int)
data["credit.history.length"] = y * 12 + m
data.head(5)
```

Out[22]:

	disbursed_amount	asset_cost	ltv	branch_id	date.of.birth	employment.type	disbursaldate	mobileno_avl_flag
0	36439	65850	56.19	64	1990-06-14	Self employed	2018-09-28	1
1	48749	69303	72.15	67	1991-01-01	Salaried	2018-10-09	1
2	55348	66340	85.00	2	1993-08-16	Self employed	2018-08-31	1
3	48849	64133	77.96	217	1989-01-01	Self employed	2018-10-13	1
4	40394	59386	70.72	74	1974-12-31	Self employed	2018-09-14	1

Age in years when loan was disbursed

In [23]:

```
data['Age'] = (data['disbursaldate'] - data['date.of.birth'])
data['Age'] = data['Age']/np.timedelta64(1,'Y')
```

In [24]:

```
data['Age'] = data['Age'].apply(lambda x: int(x))
```

In [25]:

```
data.head(10)
```

Out[25]:

	disbursed_amount	asset_cost	ltv	branch_id	date.of.birth	employment.type	disbursaldate	mobileno_avl_flag
0	36439	65850	56.19	64	1990-06-14	Self employed	2018-09-28	1
1	48749	69303	72.15	67	1991-01-01	Salaried	2018-10-09	1
2	55348	66340	85.00	2	1993-08-16	Self employed	2018-08-31	1
3	48849	64133	77.96	217	1989-01-01	Self employed	2018-10-13	1
4	40394	59386	70.72	74	1974-12-31	Self employed	2018-09-14	1
5	51803	67466	79.30	162	1964-11-23	Self employed	2018-08-17	1
6	61947	109094	58.21	251	1989-01-10	Self employed	2018-08-16	1

	disbursed_amount	asset_cost	ltv	branch_id	date.of.birth	employment.type	disbursaldate	mobileno_avl_flag
7	51301	61815	85.00	67	1995-01-01	Salaried	2018-08-26	1
8	65882	80461	84.51	255	1994-06-15	Self employed	2018-10-15	1
9	34639	69717	50.49	34	1982-11-23	Self employed	2018-10-26	1

Moving age column to 7th position.

In [26]:

```
ages = data['Age']
data.insert(loc=7, column='age', value=ages)
```

In [27]:

```
data.drop('Age',axis=1,inplace=True)
```

In [28]:

```
data.drop(data[['disbursaldate','date.of.birth']], axis=1,inplace=True)
```

In [29]:

```
data.head()
```

Out[29]:

	disbursed_amount	asset_cost	ltv	branch_id	employment.type	age	mobileno_avl_flag	aadhar_flag	pan_flag
0	36439	65850	56.19	64	Self employed	28	1	1	0
1	48749	69303	72.15	67	Salaried	27	1	1	0
2	55348	66340	85.00	2	Self employed	25	1	1	0
3	48849	64133	77.96	217	Self employed	29	1	1	0
4	40394	59386	70.72	74	Self employed	43	1	1	0

In [30]:

```
data.describe()
```

Out[30]:

	disbursed_amount	asset_cost	ltv	branch_id	age	mobileno_avl_flag	aadhar_flag
count	23315.000000	23315.000000	23315.000000	23315.000000	23315.000000	23315.0	23315.000000
mean	54297.647309	75842.182887	74.701607	72.079262	33.850097	1.0	0.845078
std	13061.877434	18988.525635	11.462722	69.095008	9.809538	0.0	0.361838
min	13369.000000	37230.000000	17.130000	1.000000	17.000000	1.0	0.000000
25%	46949.000000	65629.000000	68.830000	13.000000	26.000000	1.0	1.000000
50%	53759.000000	70929.000000	76.710000	61.000000	32.000000	1.0	1.000000
75%	60379.000000	79354.500000	83.630000	121.000000	41.000000	1.0	1.000000
max	592460.000000	715186.000000	94.980000	261.000000	64.000000	1.0	1.000000

In [31]:

```
df = data.copy()
```

In [32]:

```
df2 = data.copy()
```

In [241…

```
mask1 = df['mobileno_avl_flag']==1
```

```python
mask2 = df['aadhar_flag']==1
mask3 = df['pan_flag']==1
mask4 = df['voterid_flag']==1
mask5 = df['driving_flag']==1
mask6 = df['passport_flag']==1

df.where(mask1 & mask2 & mask3 & mask4 & mask5 & mask6).dropna()
```

Out[241…

disbursed_amount	asset_cost	ltv	branch_id	employment.type	age	mobileno_avl_flag	aadhar_flag	pan_flag	vo

0 rows × 21 columns

In [264…

```python
df[df['mobileno_avl_flag'] + df['aadhar_flag'] + df['pan_flag'] + df['voterid_flag'] + df
```

Out[264…

	disbursed_amount	asset_cost	ltv	branch_id	employment.type	age	mobileno_avl_flag	aadhar_flag	pan_
19	78151	107074	74.25	135	Self employed	31	1	1	
700	38439	73797	54.20	202	Salaried	26	1	1	
1642	112463	167819	69.72	1	Salaried	34	1	1	
3576	137654	213600	67.42	1	Self employed	28	1	1	
4120	48349	71886	69.55	3	Salaried	34	1	1	
4324	57413	67841	88.44	7	Salaried	25	1	1	
4769	45849	62000	75.00	2	Self employed	35	1	1	
5051	49594	68697	74.99	3	Self employed	45	1	1	
5276	72123	99617	74.28	3	Salaried	34	1	1	
6647	50383	65705	82.19	19	Self employed	27	1	1	
6855	48349	64512	77.50	3	Self employed	24	1	1	
6870	56889	69793	84.54	5	Salaried	39	1	1	
7206	42872	63262	72.56	3	Self employed	31	1	1	
7994	37639	68469	58.42	1	Salaried	27	1	1	
8067	45949	64851	74.94	3	Salaried	32	1	1	
8369	43204	63000	74.60	3	Salaried	36	1	1	
8636	29741	62870	49.31	61	Self employed	28	1	1	
8766	49503	69975	74.60	3	Self employed	31	1	1	
8991	52303	67576	79.91	3	Self employed	37	1	1	

	disbursed_amount	asset_cost	ltv	branch_id	employment.type	age	mobileno_avl_flag	aadhar_flag	pan_
9298	50303	70776	73.47	3	Self employed	39	1	1	
9763	37439	49725	78.43	3	Salaried	30	1	1	
10207	50700	66148	78.61	13	Salaried	27	1	1	
10970	93599	125444	75.89	73	Self employed	29	1	1	
12348	47349	66500	73.68	2	Self employed	27	1	1	
12901	40560	56989	73.35	70	Self employed	28	1	1	
13038	30484	64007	48.43	3	Salaried	42	1	1	
13333	88660	188711	49.28	1	Self employed	39	1	1	
13334	77909	134245	60.78	79	Self employed	26	1	1	
13385	59259	71814	83.55	67	Self employed	32	1	1	
15042	55459	63610	89.92	3	Self employed	29	1	1	
15351	37133	50499	79.80	3	Self employed	22	1	1	
15874	111972	168319	69.51	9	Salaried	56	1	1	
16023	119787	170850	73.16	79	Self employed	38	1	1	
16112	36659	61425	63.49	3	Salaried	52	1	1	
17594	54259	77031	72.70	3	Self employed	30	1	1	
20587	53583	74350	75.32	3	Self employed	44	1	1	
20627	44849	58412	79.61	160	Self employed	39	1	1	
20671	47834	66600	76.05	3	Salaried	45	1	1	
20947	47349	58672	81.81	160	Self employed	30	1	1	
20956	50303	65862	78.95	3	Salaried	21	1	1	
21176	28921	63472	48.84	3	Salaried	31	1	1	
21983	39745	63872	64.74	15	Self employed	50	1	1	
22549	124646	188711	68.89	1	Self employed	27	1	1	
22828	53078	76400	71.99	152	Self employed	33	1	1	
23073	34075	44814	79.89	36	Self employed	35	1	1	
23116	52578	62469	86.44	79	Salaried	24	1	1	
23226	48145	70237	69.76	9	Self employed	47	1	1	

47 rows × 21 columns

In [259…

```
df[df['mobileno_avl_flag'] + df['aadhar_flag'] + df['pan_flag'] + df['voterid_flag'] + df
```

Out[259…

```
4
```

In [261…

```
df[df['mobileno_avl_flag'] + df['aadhar_flag'] + df['pan_flag'] + df['voterid_flag'] + df
```

Out[261…

```
Poor    47
Name: credit_history_rating, dtype: int64
```

Inference from LTVs

In [33]:

```
df[df['ltv']>90]['loan_default'].sum()
```

Out[33]:

```
21
```

21 defaulters have LTV rate more than 90%. The higher the LTV, the loan represents more of the value of asset cost and is a bigger risk to the lender. This is something which might make lenders stand to lose.

In [221…

```
df[df['ltv']>60]['loan_default'].sum()
```

Out[221…

```
4741
```

In [222…

```
df[df['ltv']<60]['loan_default'].sum()
```

Out[222…

```
384
```

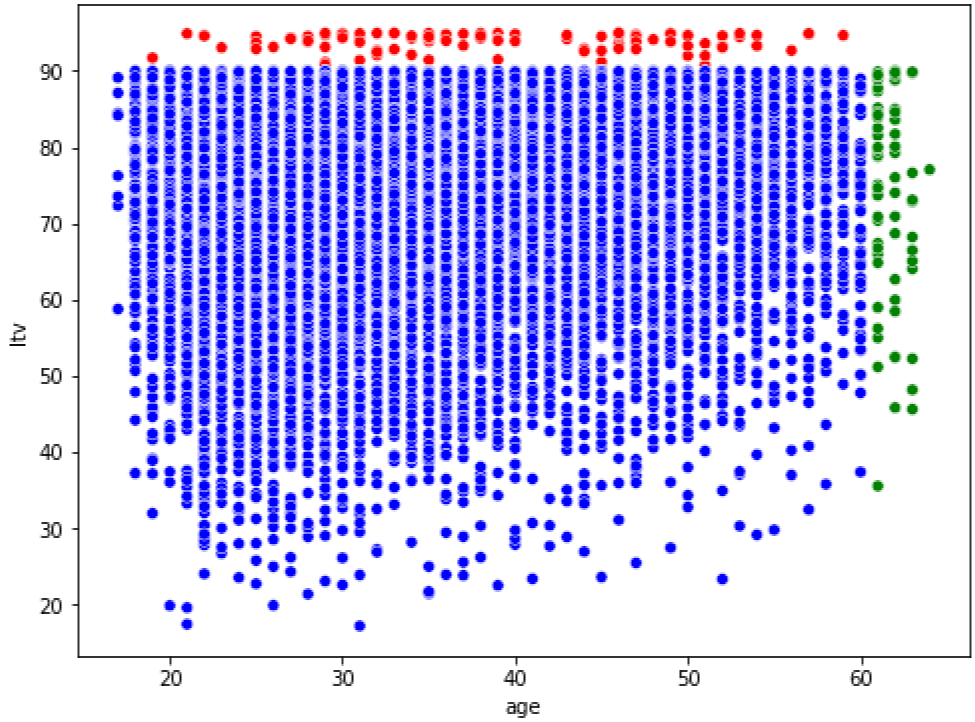
The less the LTV, more the amount contributed towards the asset. And chances of becoming a defaulter too reduce.

In [35]:

```
plt.figure(figsize=(8,6))
col = np.where(df['ltv']>90, 'r', np.where(df['age']>60,'g','b'))
sns.scatterplot(x='age', y='ltv', c=col, data=df)
```

Out[35]:

```
<AxesSubplot:xlabel='age', ylabel='ltv'>
```



Observations:
From the above plot, The 21 Customers with LTV > 90% are shown in red. Also, very few senior citizens tend to have LTV, and none of them fall under those 21 customers.

Inference from Branch ID and Employment type

In [36]:

```
count = df.groupby(['branch_id','employment.type','loan_default']).size().reset_index().re
count.head(10)
```

Out[36]:

	branch_id	employment.type	loan_default	count
0	1	Salaried	0	95
1	1	Salaried	1	11
2	1	Self employed	0	381
3	1	Self employed	1	81
4	2	Salaried	0	663
5	2	Salaried	1	145
6	2	Self employed	0	468
7	2	Self employed	1	96
8	3	Salaried	0	349
9	3	Salaried	1	66

Branch-wise loan default check for each employment type.

In [37]:

```
count.groupby('loan_default')['count'].agg(max)
```

Out[37]:

```
loan_default
0    663
1    205
Name: count, dtype: int64
```

In [38]:
```python
count[count['count']==205]
```

Out[38]:

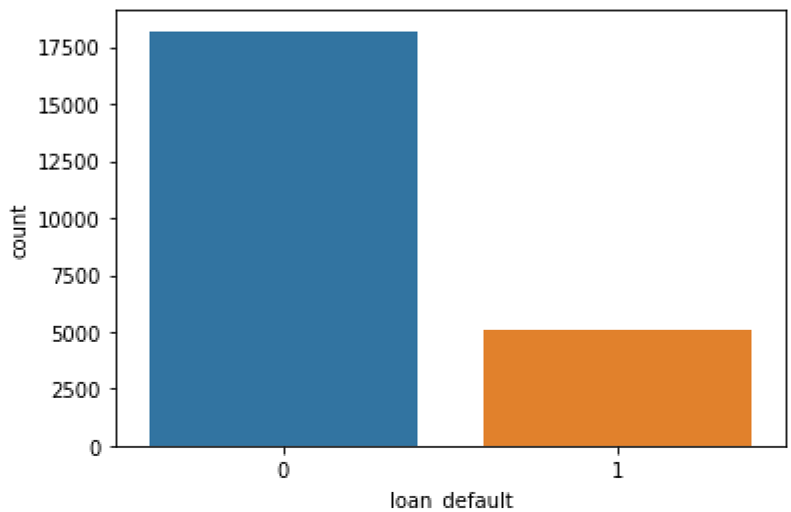
	branch_id	employment.type	loan_default	count
81	36	Self employed	1	205

Observations:

More loan defaults at branch ID = 36.

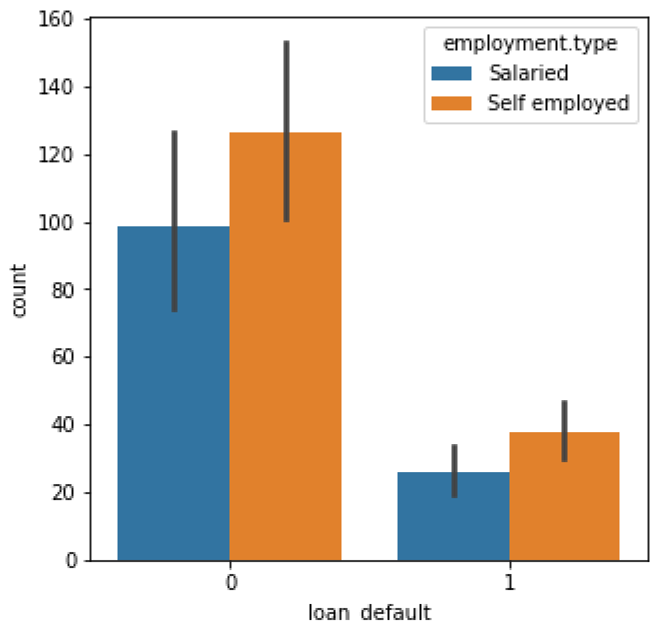
In [39]:
```python
sns.countplot(x="loan_default", data = data)
```

Out[39]:
```
<AxesSubplot:xlabel='loan_default', ylabel='count'>
```



In [40]:
```python
fig, ax = plt.subplots(figsize=(5,5))
sns.barplot(x='loan_default', y='count', hue='employment.type', data=count)
```

Out[40]:
```
<AxesSubplot:xlabel='loan_default', ylabel='count'>
```



Observations:

```
1. Defaulters are less. <br>
2. Self-Employed are more in number. <br>
3. Salaried and Self-Employed being non-defaulter are also more.
```

Inference from Age

In [41]:
```
def age_type(x):
    if x < 30:
        return "Young Adults"
    elif x >= 30 and x < 60:
        return "Adults"
    else:
        return "Old Aged"
```

In [42]:
```
df['age_type'] = df['age'].apply(age_type)
```

In [43]:
```
df.head()
```

Out[43]:

	disbursed_amount	asset_cost	ltv	branch_id	employment.type	age	mobileno_avl_flag	aadhar_flag	pan_flag
0	36439	65850	56.19	64	Self employed	28	1	1	0
1	48749	69303	72.15	67	Salaried	27	1	1	0
2	55348	66340	85.00	2	Self employed	25	1	1	0
3	48849	64133	77.96	217	Self employed	29	1	1	0
4	40394	59386	70.72	74	Self employed	43	1	1	0

In [44]:
```
df['age_type'].value_counts()
```

Out[44]:
```
Adults          13842
Young Adults     9340
Old Aged          133
Name: age_type, dtype: int64
```

In [45]:
```
age_group = df.groupby('age_type').size().reset_index().rename(columns={0:"age_count"})
```

In [46]:
```
age_group['Count of defaulters'] = list(df.groupby('age_type')['loan_default'].agg(sum))
```

In [47]:
```
age_group['% Loan defaults'] = round((age_group['Count of defaulters'] / age_group['age_co
```

In [48]:
```
age_group['Total Delinquent Accounts'] = list(df.groupby('age_type')['delinquent.accts.in.
```

In [49]:
```
age_group['% Delinquent Accounts'] = round((age_group['Total Delinquent Accounts'] / df['c
```

In [226…
```
age_group
```

Out[226…

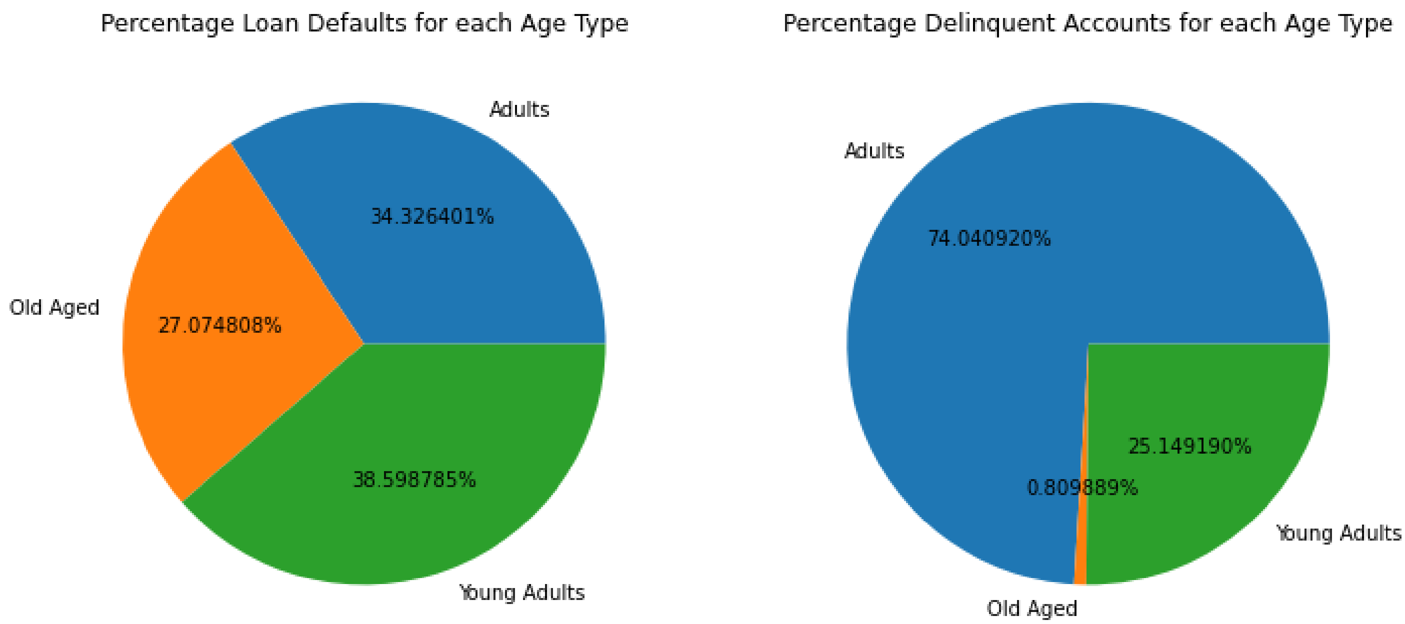
	age_type	age_count	Count of defaulters	% Loan defaults	Total Delinquent Accounts
0	Adults	13842	2902	20.97	1737

	age_type	age_count	Count of defaulters	% Loan defaults	Total Delinquent Accounts
1	Old Aged	133	22	16.54	19
2	Young Adults	9340	2202	23.58	590

In [227…

```
plt.figure(figsize=(12,10))
plt.subplot(1, 2, 1)
plt.pie(age_group['% Loan defaults'], labels=age_group['age_type'], autopct='%2f%%')
plt.title("Percentage Loan Defaults for each Age Type")

plt.subplot(1, 2, 2)
plt.pie(age_group['Total Delinquent Accounts'], labels=age_group['age_type'], autopct='%2f
plt.title("Percentage Delinquent Accounts for each Age Type")
plt.show()
```



Observations:

1. Young Adults are majority as percentage of defaulters.
2. Adults have more Delinquent accounts created in last six months.

Inference from Bureau Score (perform_cns.score)

In [52]:

```
def credit_rating(x):
    if x <= 300:
        return "Poor"
    elif x >= 700:
        return "Very Good"
    else:
        return "Ok"
```

A credit score deemed "very good" or "exceptional" may range from 700 or higher and credit score deemed "very poor" may range from 300 or below. A good credit score helps to get better interest rates on credit cards and loans, an increased likelihood of loan approvals, obtaining an apartment for rent, as well as better car insurance rates.

In [53]:

```
df['credit_score_rating'] = df['perform_cns.score'].apply(credit_rating)
```

In [54]:
```
credit_rate = df.groupby('credit_score_rating').size().reset_index().rename(columns={0:"ra
```

In [55]:
```
credit_rate['total_loan_defaults'] = list(df.groupby('credit_score_rating')['loan_default'
```

In [56]:
```
credit_rate['% loan defaults'] = round(credit_rate['total_loan_defaults'] / (df['loan_defa
```

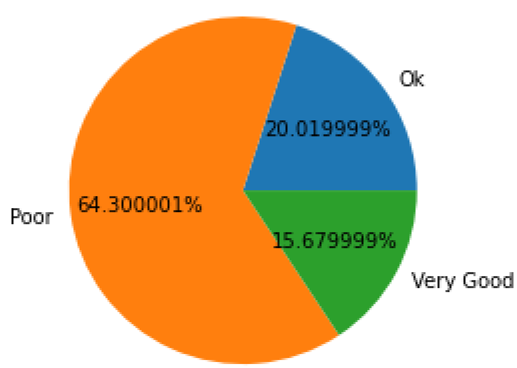
In [57]:
```
credit_rate
```

Out[57]:

	credit_score_rating	rate_count	total_loan_defaults	% loan defaults
0	Ok	4304	1026	20.02
1	Poor	13796	3296	64.30
2	Very Good	5215	804	15.68

In [58]:
```
plt.figure(figsize=(6,4))
plt.pie(credit_rate['% loan defaults'], labels=credit_rate['credit_score_rating'], autopct
```

Out[58]:
```
([<matplotlib.patches.Wedge at 0x2347895ba00>,
  <matplotlib.patches.Wedge at 0x2347894c130>,
  <matplotlib.patches.Wedge at 0x2347894c850>],
 [Text(0.8895122875259667, 0.6471227784125068, 'Ok'),
  Text(-1.089791338875732, -0.14951534272923017, 'Poor'),
  Text(0.9692159490758288, -0.5202119222557674, 'Very Good')],
 [Text(0.48518852046870903, 0.35297606095227635, '20.019999%'),
  Text(-0.5944316393867629, -0.0815538233068528, '64.300001%'),
  Text(0.528663244950452, -0.2837519575940549, '15.679999%')])
```



Observations:
As we can see here too, percentage of loan-defaulters is very less (~16%) among total defaulters for the customers having very good Bureau score. And 64% of total loan-defaulters have Bureau score less than 300. Overall Bureau score is considered as an important parameter. I've also cross-checked this analysis through Logistic Regression.

Inference from Credit History Length

In [59]:
```
def credit_hist_length(x):
    if x >= 144:
        return "Good"
    else:
        return "Poor"
```

Account being actively open for more than 12 years considered to be good for credit score or not falling into being defaulter.

In [60]:
```
df['credit_history_rating'] = df['credit.history.length'].apply(credit_hist_length)
```

In [61]:
```
credit_history_rate = df.groupby('credit_history_rating').size().reset_index().rename(colu
```

In [62]:
```
credit_history_rate['total_loan_defaults'] = list(df.groupby('credit_history_rating')['loa
```

In [63]:
```
def good(x):
    if x == "Very Good":
        return 1
    else:
        return 0
```

Mapping credit scores more than 700 to check with credit history length of more than 12 years.

In [64]:
```
df['Good performers on Credit score'] = df['credit_score_rating'].apply(good)
```

In [65]:
```
credit_history_rate['good_performers'] = list(df.groupby('credit_history_rating')['Good pe
```

In [66]:
```
credit_history_rate['% of Good Performers'] = round(credit_history_rate['good_performers']
```

In [67]:
```
credit_history_rate['% loan defaults'] = round(credit_history_rate['total_loan_defaults']
```

In [68]:
```
credit_history_rate
```

Out[68]:

	credit_history_rating	rate_count	total_loan_defaults	good_performers	% of Good Performers	% loan defaults
0	Good	210	28	59	28.10	0.55
1	Poor	23105	5098	5156	22.32	99.45

Observations:

1. Among customers having credit history length more than 12 years, 28% are having credit score as "Very Good" (>700), which is actually more compared to that of poor credit history rating.
2. Very negligible percentage (0.55%) of loan defaulters fall into the "Good" category. Clearly says that having good credit history length makes one to not become a defaulter.

In [69]:
```
df2['mobileno_avl_flag'].value_counts()
```

Out[69]:
```
1    23315
Name: mobileno_avl_flag, dtype: int64
```

All have given their contact number. So, cannot infer if not sharing contact number makes any difference.

In [70]:
```
sns.pairplot(df2, vars=['disbursed_amount','asset_cost','age','ltv','perform_cns.score','c
plt.show()
```

In [71]:
```python
numerical = data[['disbursed_amount','asset_cost','ltv','branch_id','age','perform_cns.sco
numerical.columns.to_list()
```

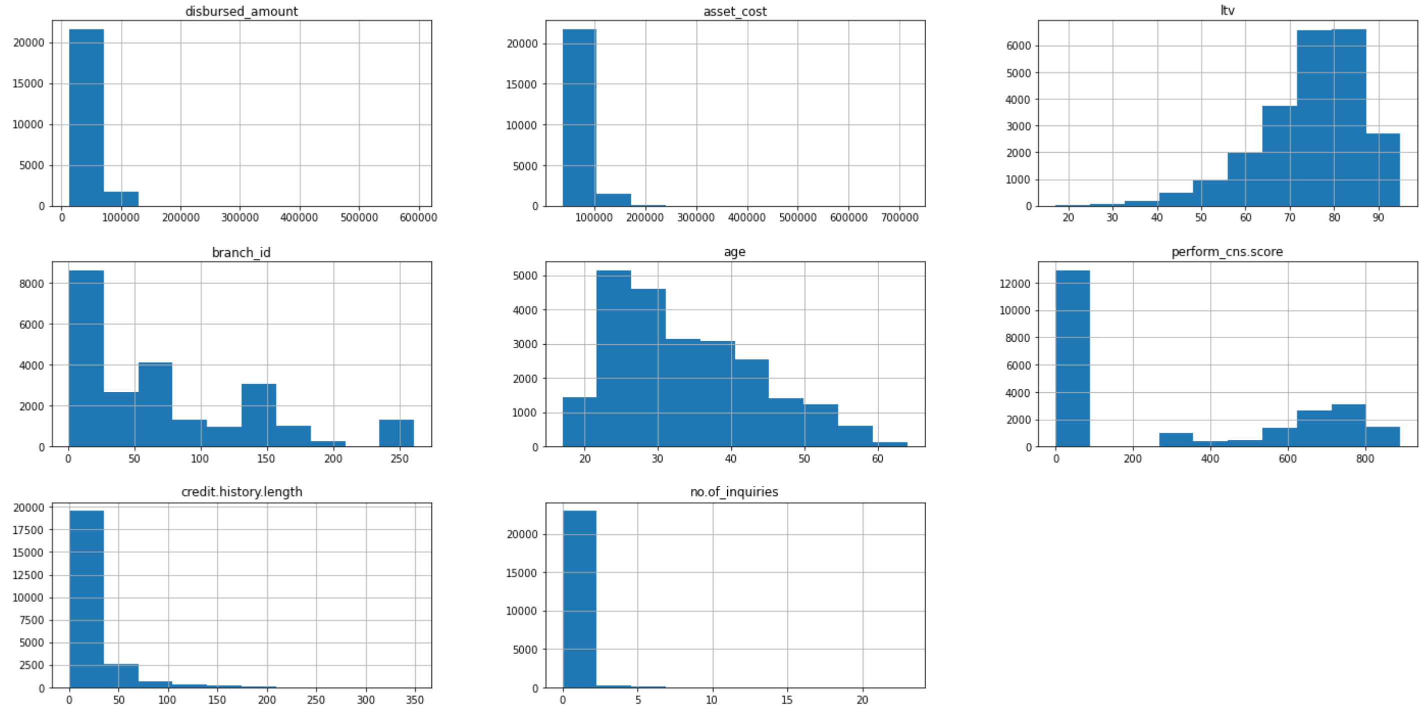
Out[71]:
```
['disbursed_amount',
 'asset_cost',
 'ltv',
 'branch_id',
 'age',
 'perform_cns.score',
 'credit.history.length',
 'no.of_inquiries']
```

Checking Skewness and Outliers

In [72]:
```python
plt.rcParams["figure.figsize"] = (24, 12)
numerical.hist()
```

Out[72]:
```
array([[<AxesSubplot:title={'center':'disbursed_amount'}>,
        <AxesSubplot:title={'center':'asset_cost'}>,
        <AxesSubplot:title={'center':'ltv'}>],
       [<AxesSubplot:title={'center':'branch_id'}>,
```

```
 <AxesSubplot:title={'center':'age'}>,
 <AxesSubplot:title={'center':'perform_cns.score'}>],
[<AxesSubplot:title={'center':'credit.history.length'}>,
 <AxesSubplot:title={'center':'no.of_inquiries'}>, <AxesSubplot:>]],
dtype=object)
```

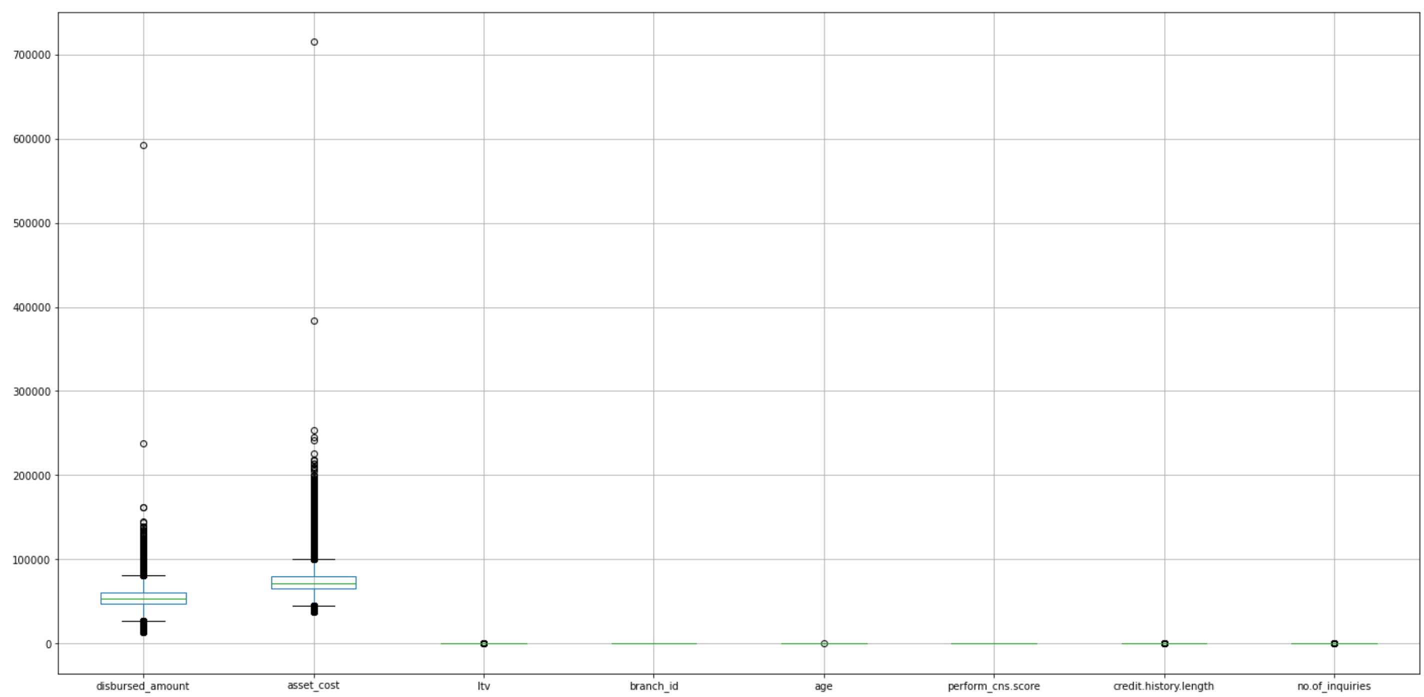


Disbursed_amount, asset_cost, credit.history.length, no.of_inquirires, disbursed_age are left skewed data. And ltv is right skewed data. We can apply log transformation to Disbursed_amount, asset_cost, ltv and disbursed_age so as to reduce the skewness of our data and since we don't have any negative and zero values in them. And for the rest z-score normalization.

In [73]:

```python
plt.rcParams["figure.figsize"] = (24, 12)
numerical.boxplot()
```

Out[73]: `<AxesSubplot:>`



disbursed_amount and asset_cost have too many outliers.

Quartile ranges for each attribute.

In [74]:

```
for i in numerical:
    q75,q25 = np.percentile(numerical.loc[:,i],[75,25])
    iqr = q75-q25
    minimum = q25 - (iqr * 1.5)
    maximum = q75 + (iqr * 1.5)
    print(i)
    print('Q25 = ',q25)
    print('Q75 = ',q75)
    print('Lower-limit = ',minimum)
    print('Upper-limit = ',maximum)
    print('*************************')
```

```
disbursed_amount
Q25 =  46949.0
Q75 =  60379.0
Lower-limit =  26804.0
Upper-limit =  80524.0
*************************
asset_cost
Q25 =  65629.0
Q75 =  79354.5
Lower-limit =  45040.75
Upper-limit =  99942.75
*************************
ltv
Q25 =  68.83
Q75 =  83.63
Lower-limit =  46.63
Upper-limit =  105.82999999999998
*************************
branch_id
Q25 =  13.0
Q75 =  121.0
Lower-limit =  -149.0
Upper-limit =  283.0
*************************
age
Q25 =  26.0
Q75 =  41.0
Lower-limit =  3.5
Upper-limit =  63.5
*************************
perform_cns.score
Q25 =  0.0
Q75 =  679.0
Lower-limit =  -1018.5
Upper-limit =  1697.5
*************************
credit.history.length
Q25 =  0.0
Q75 =  24.0
Lower-limit =  -36.0
Upper-limit =  60.0
*************************
no.of_inquiries
Q25 =  0.0
Q75 =  0.0
Lower-limit =  0.0
Upper-limit =  0.0
*************************
```

Correlation between Numerical values

In [75]:

```python
numerical['loan_default'] = data['loan_default']
corr_matrix = numerical.corr()
corr_matrix
```

Out[75]:

	disbursed_amount	asset_cost	ltv	branch_id	age	perform_cns.score	credit.history.l
disbursed_amount	1.000000	0.752629	0.372385	0.024890	-0.066678	0.009414	0.0
asset_cost	0.752629	1.000000	-0.304288	0.023997	-0.123455	-0.052328	-0.0
ltv	0.372385	-0.304288	1.000000	0.010844	0.077077	0.088537	0.0
branch_id	0.024890	0.023997	0.010844	1.000000	0.001531	-0.016329	-0.0
age	-0.066678	-0.123455	0.077077	0.001531	1.000000	0.167337	0.2
perform_cns.score	0.009414	-0.052328	0.088537	-0.016329	0.167337	1.000000	0.4
credit.history.length	0.013026	-0.020114	0.048224	-0.003680	0.286099	0.467107	1.0
no.of_inquiries	0.035428	-0.018974	0.087960	0.000974	0.011284	0.146122	0.0
loan_default	0.062805	0.002535	0.095423	0.037361	-0.036662	-0.060836	-0.0

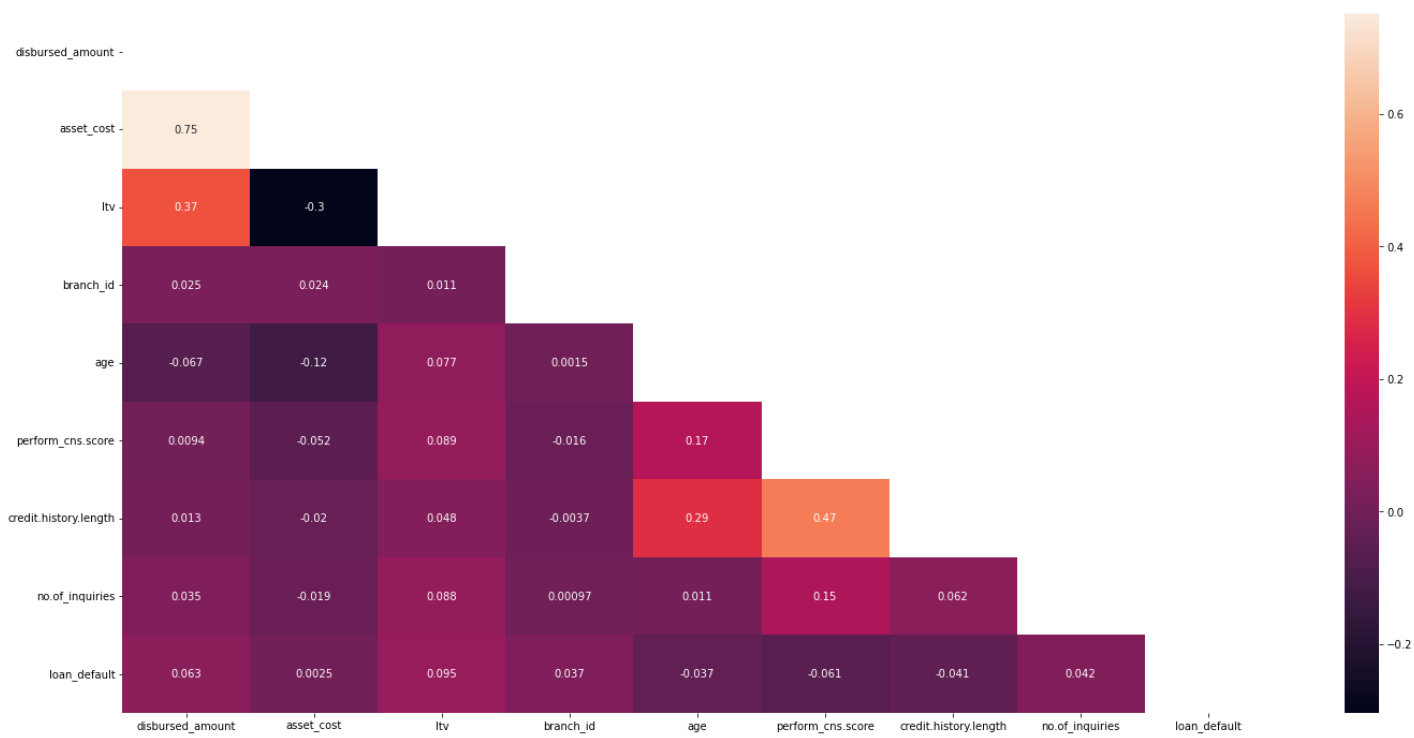
In [76]:

```python
mask = np.zeros_like(corr_matrix)
mask[np.triu_indices_from(mask)] = True
```

In [77]:

```python
sns.heatmap(corr_matrix, annot=True, mask=mask)
```

Out[77]: <AxesSubplot:>



As we can see here, asset_cost and ltv are negatively correlated in moderate. As asset_cost increasing, ltv decreasing. Whereas, ltv and disbursed_amount are positively correlated in moderate. As disbursed_amount increasing, ltv too increasing.

Whereas, asset_cost and disbursed_amount are more positively correlated. Also, perform_cns_score and credit.history.length and positively correlated. Hence, the longer an account has been open and active, the better it is for the credit score (perform_cns.score).

Correlation for categorical variable

Chi-Square testing for Categorical Variable (Employment.Type) and loan_default

Null hypothesis: There is no relationship between Employment.Type and loan_default.

Alternate hypothesis: There is relationship between Employment.Type and loan_default.

In [78]:
```
contingency_tab = pd.crosstab(data['employment.type'],data['loan_default'])
contingency_tab
```

Out[78]:

loan_default	0	1
employment.type		
Salaried	7806	2015
Self employed	10383	3111

In [79]:
```
from scipy.stats import chi2_contingency,chisquare
chi_sq_Stat, p_value, deg_freedom, exp_freq = chi2_contingency(contingency_tab)

print('Chi-square statistic %3.5f    P value %1.6f    Degrees of freedom %d'
      %(chi_sq_Stat, p_value,deg_freedom))
```

```
Chi-square statistic 21.18931    P value 0.000004    Degrees of freedom 1
```

Since P-value is much smaller than the significance = 0.05, therefore **Alternate hypothesis** is accepted. Hence we can say that **there is a relationship between Employment.Type and loan_default.**

Log-Transformation 'disbursed_amount', 'asset_cost', 'ltv', 'age' columns

In [80]:
```
data[['disbursed_amount', 'asset_cost', 'ltv','age']] = data[['disbursed_amount', 'asset_c
```

In [81]:
```
data.head(10)
```

Out[81]:

	disbursed_amount	asset_cost	ltv	branch_id	employment.type	age	mobileno_avl_flag	aadhar_flag	pa
0	10.503395	11.095135	4.028739	64	Self employed	3.332205	1	1	
1	10.794440	11.146243	4.278747	67	Salaried	3.295837	1	1	
2	10.921396	11.102548	4.442651	2	Self employed	3.218876	1	1	
3	10.796489	11.068714	4.356196	217	Self employed	3.367296	1	1	
4	10.606437	10.991814	4.258728	74	Self employed	3.761200	1	1	
5	10.855203	11.119379	4.373238	162	Self employed	3.970292	1	1	
6	11.034034	11.599965	4.064057	251	Self employed	3.367296	1	0	
7	10.845466	11.031901	4.442651	67	Salaried	3.135494	1	1	
8	11.095621	11.295528	4.436870	255	Self employed	3.178054	1	1	
9	10.452735	11.152199	3.921775	34	Self employed	3.555348	1	1	

One-Hot Encoding for categorical variable

In [82]:
```
df_encoded = pd.get_dummies(data)
df_encoded.head()
```

Out[82]:

	disbursed_amount	asset_cost	ltv	branch_id	age	mobileno_avl_flag	aadhar_flag	pan_flag	voterid_fla
0	10.503395	11.095135	4.028739	64	3.332205	1	1	0	
1	10.794440	11.146243	4.278747	67	3.295837	1	1	0	
2	10.921396	11.102548	4.442651	2	3.218876	1	1	0	
3	10.796489	11.068714	4.356196	217	3.367296	1	1	0	
4	10.606437	10.991814	4.258728	74	3.761200	1	1	0	

Z-Score Normalization

In [210…

```
df_encoded[['credit.history.length','no.of_inquiries','perform_cns.score','branch_id']] =
```

In [211…

```
df_encoded.head()
```

Out[211…

	disbursed_amount	asset_cost	ltv	branch_id	age	mobileno_avl_flag	aadhar_flag	pan_flag	voterid_fla
0	10.503395	11.095135	4.028739	-0.116930	3.332205	1	1	0	
1	10.794440	11.146243	4.278747	-0.073511	3.295837	1	1	0	
2	10.921396	11.102548	4.442651	-1.014245	3.218876	1	1	0	
3	10.796489	11.068714	4.356196	2.097413	3.367296	1	1	0	
4	10.606437	10.991814	4.258728	0.027799	3.761200	1	1	0	

In [83]:

```
X = df_encoded.drop('loan_default', axis=1)
y = df_encoded['loan_default']
```

In [85]:

```
X.head()
```

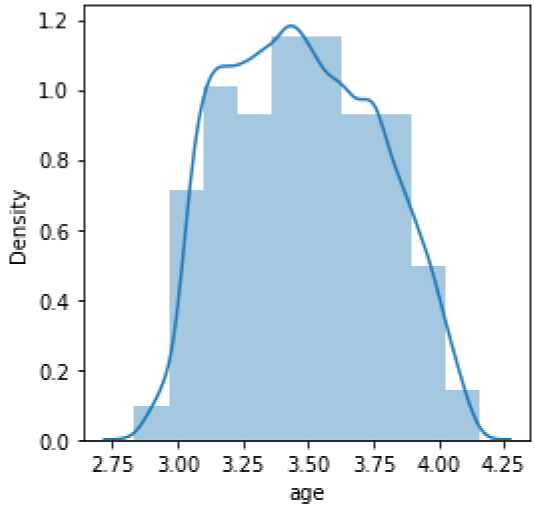
Out[85]:

	disbursed_amount	asset_cost	ltv	branch_id	age	mobileno_avl_flag	aadhar_flag	pan_flag	voterid_fla
0	10.503395	11.095135	4.028739	-0.116930	3.332205	1	1	0	
1	10.794440	11.146243	4.278747	-0.073511	3.295837	1	1	0	
2	10.921396	11.102548	4.442651	-1.014245	3.218876	1	1	0	
3	10.796489	11.068714	4.356196	2.097413	3.367296	1	1	0	
4	10.606437	10.991814	4.258728	0.027799	3.761200	1	1	0	

In [86]:

```
fig, ax = plt.subplots(figsize=(4,4))
sns.distplot(X['age'], bins=10)
```

Out[86]: <AxesSubplot:xlabel='age', ylabel='Density'>



Model Building

Baseline model for Logistic Regression

In [87]:
```
from sklearn.model_selection import train_test_split
```

In [88]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=0)
```

Logistic Regression

In [89]:
```
regressor = LogisticRegression()
regressor = regressor.fit(X_train,y_train)
```

In [90]:
```
y_pred = regressor.predict(X_test)
```

In [91]:
```
y_test[10:20]
```

Out[91]:
```
3254     0
390      0
10472    1
1482     0
7007     0
23267    1
15478    0
4501     0
21047    0
6211     0
Name: loan_default, dtype: int64
```

In [92]:
```
y_pred[10:20]
```

Out[92]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

In [93]:
```
from sklearn.metrics import classification_report
```

In [94]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support
```

```
              0       0.78      1.00      0.88      3640
              1       0.22      0.00      0.00      1023

       accuracy                           0.78      4663
      macro avg       0.50      0.50      0.44      4663
   weighted avg       0.66      0.78      0.68      4663
```

Since it is a class imbalanced problem, we focus on precision, recall and f1-score also AUC score, instead of accuracy. Here, we can clearly see that majority class is being predicted very well but minority class prediction rate is totally awful. And recall for the same is zero, means True Positives are totally zero in number, as we can see from y_pred. So to overcome this problem, SMOTE can be performed.

SMOTE

In [95]:

```
from imblearn.over_sampling import SMOTE
smot = SMOTE(random_state=0)
smot_x, smot_y = smot.fit_resample(X,y)
X = pd.DataFrame(smot_x, columns=X.columns)
y = pd.DataFrame(smot_y, columns=['loan_default'])
```

In [96]:

```
X.shape
```

Out[96]:

```
(36378, 17)
```

In [97]:

```
y.shape
```

Out[97]:

```
(36378, 1)
```

In [98]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=0)
```

Logistic Regression for SMOTE dataframe

In [99]:

```
regressor2 = LogisticRegression()
regressor2 = regressor2.fit(X_train,y_train)
```

In [100…

```
y_predict = regressor2.predict(X_test)
```

In [101…

```
y_test[10:20]
```

Out[101…

	loan_default
25519	1
11686	0
4410	0
30239	1
17000	0
29772	1
25035	1

	loan_default
22309	0
34507	1
11766	1

In [102…

```
y_predict[10:20]
```

Out[102…

```
array([1, 1, 0, 1, 0, 1, 0, 0, 1, 1], dtype=int64)
```

Two misclassifications

In [103…

```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.58      0.54      0.56      3600
           1       0.58      0.62      0.60      3676

    accuracy                           0.58      7276
   macro avg       0.58      0.58      0.58      7276
weighted avg       0.58      0.58      0.58      7276
```

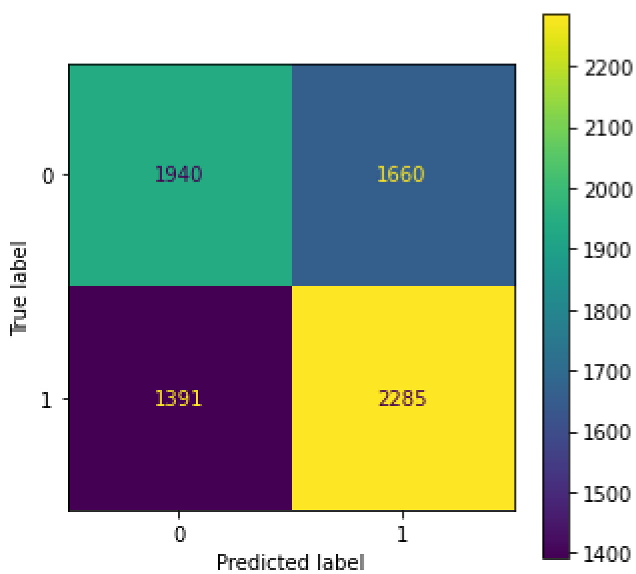
Now we can see that the metrics score of the minority class has been improved a little.

In [104…

```
from sklearn.metrics import plot_confusion_matrix
```

In [105…

```
fig, ax = plt.subplots(figsize=(5,5))
plot_confusion_matrix(regressor2, X_test, y_test, ax=ax)
plt.show()
```



True Positives = 2285 = Person is a Default and same predicted by the model.
True Negatives = 1940 = Person is not a Default and same predicted by the model.
False Positives = 1391 = Person is not a Default, but model predicted the person as Default.
False Negatives = 1660 = Person is a Default, but model predicted the person as not a Default.
In actual case if a person is a Default and prediction goes wrong, then it might makes lenders to stand out without even knowing the things properly about the borrower, since being a default or not considered as an

important parameter for lending them the loan. So, FNs have to be reduced as much as possible. Therefore, recall comes into picture for selecting the right metric.

In [106…

```
probs = regressor2.predict_proba(X_test)
```

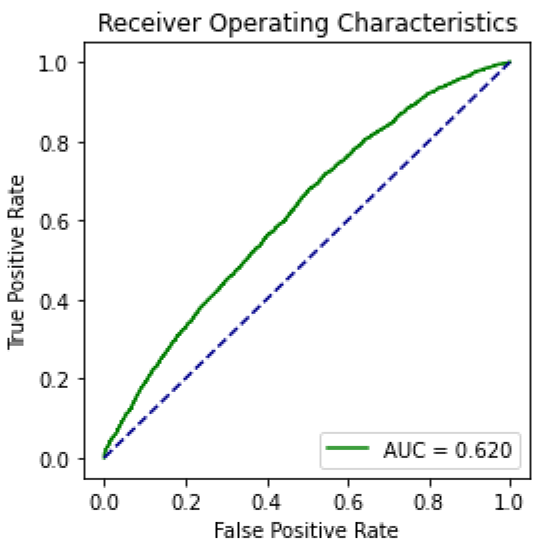
In [107…

```
prob_positive = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, prob_positive)
roc_auc = metrics.auc(fpr, tpr)
print("Area under the curve: ", roc_auc)
```

```
Area under the curve:  0.6202872234312659
```

In [108…

```
plt.rcParams["figure.figsize"] = (4,4)
plt.title("Receiver Operating Characteristics")
plt.plot(fpr,tpr,'green',label='AUC = %0.3f' %roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0,1],[0,1], color='darkblue', linestyle='--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Importance of each predictor by its Coefficient.

In [109…

```
importances = pd.DataFrame(data={
    'Attribute': X_train.columns,
    'Coefficient': regressor2.coef_[0]
})
importances = importances.sort_values(by='Coefficient', ascending=False)
```

In [110…

```
importances.reset_index(drop=True, inplace=True)
```

In [111…

```
importances
```

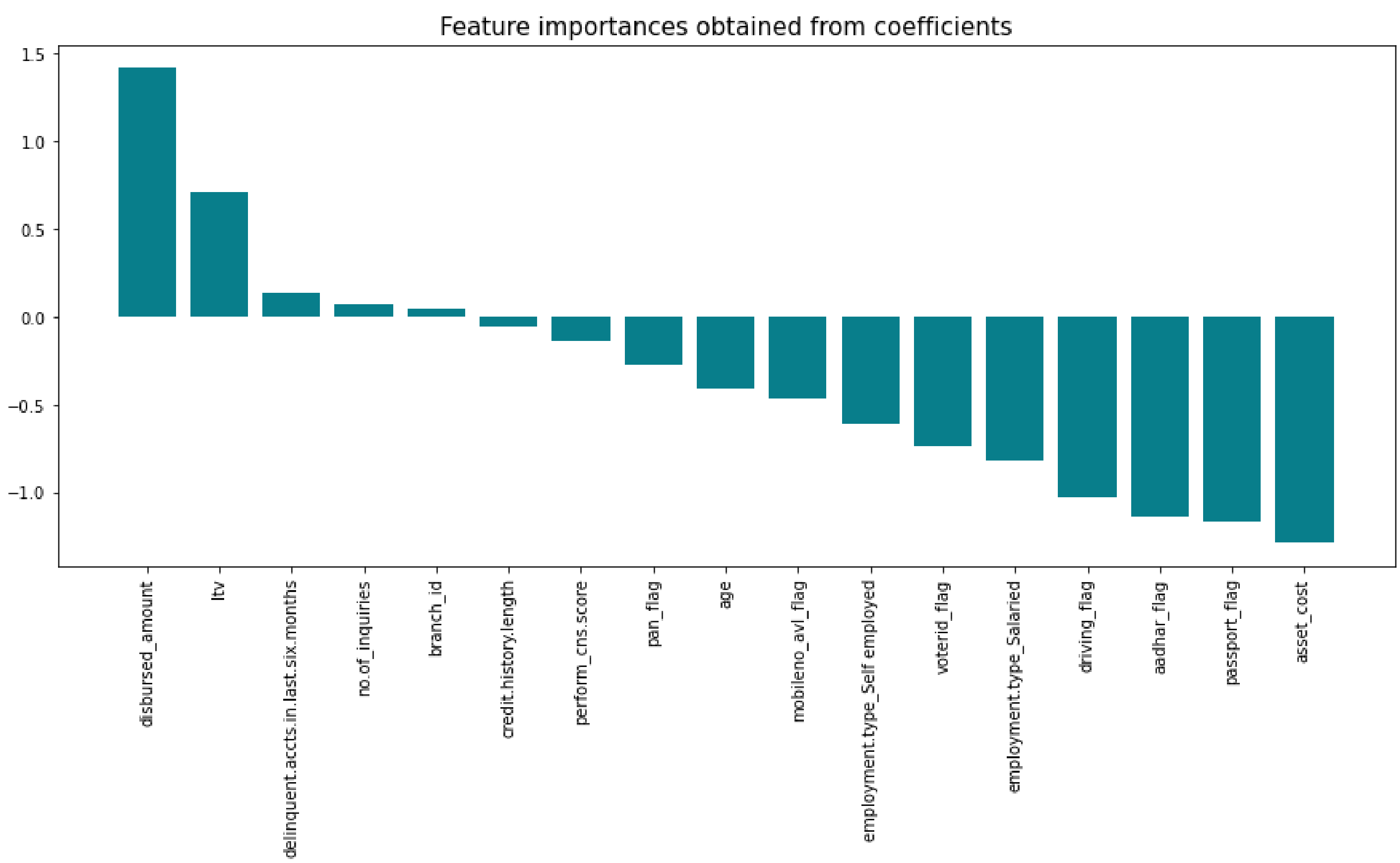
Out[111…

	Attribute	Coefficient
0	disbursed_amount	1.418798
1	ltv	0.712966
2	delinquent.accts.in.last.six.months	0.134613

	Attribute	Coefficient
3	no.of_inquiries	0.074544
4	branch_id	0.047109
5	credit.history.length	-0.052909
6	perform_cns.score	-0.136082
7	pan_flag	-0.271665
8	age	-0.413380
9	mobileno_avl_flag	-0.464699
10	employment.type_Self employed	-0.610245
11	voterid_flag	-0.739503
12	employment.type_Salaried	-0.816998
13	driving_flag	-1.028998
14	aadhar_flag	-1.140330
15	passport_flag	-1.165155
16	asset_cost	-1.283031

In [112…

```
plt.rcParams["figure.figsize"] = (15,6)
plt.bar(x=importances['Attribute'], height=importances['Coefficient'], color='#087E8B')
plt.title('Feature importances obtained from coefficients', size=15)
plt.xticks(rotation='vertical')
plt.show()
```



Business Inference:

Disbursed_amount has positive coefficient. It means, keeping all variables equal, higher the disbursed_amount,

more likely the customer to become a defaulter, which makes sense. Lowest disbursed_amount is 13369, then the effect is 1.42 *13369 = 18984. Highest disbursed_amount is 592460, then the effect is 1.42* 592460 = 841293. Similarly, for LTV, more the LTV rate, more likely the customer to become a defaulter. There are three more attributes like delinquent.accts.in.last.six.months, no.of_inquiries and branch_id which contribute positively towards making the customer defaulter.
Whereas, age is negative. Means, more elder the customer, less likely to become a defaulter, keeping all variables equal. Maximum age is 64, its effect is 64 * -0.41 = -26. Similarly, for perform_cns.score (bureau score), it is definite that more the bureau score, less likely the person to become a defaulter.
And also we can say that the customers who have provided all the necessary documents like PAN, Aadhaar, DL, etc are less likely to become a defaulter.
Let's see how each attribute contribute for probabilistic result of being a defaulter.

Probability of being a Defaulter, $p = \frac{1}{1+e^{-z}}$

where,

$z = Bo + \sum_{i=0}^{n} BiXi$

Bo = Intercept,

Bi = Weight (Coefficient) of each attribute and

Xi = Value of each attribute

In [113…

```
intercept = regressor2.intercept_
```

Considering a random row where loan_default is 1 to check how much is the probability for that particular row to have loan_default as 1.

In [114…

```
df2.iloc[905,:]
```

Out[114…

```
disbursed_amount                          55913
asset_cost                                60407
ltv                                       94.69
branch_id                                    15
employment.type                        Salaried
age                                          50
mobileno_avl_flag                             1
aadhar_flag                                   1
pan_flag                                      0
voterid_flag                                  0
driving_flag                                  0
passport_flag                                 0
perform_cns.score                           300
delinquent.accts.in.last.six.months           0
credit.history.length                        35
no.of_inquiries                               0
loan_default                                  1
Name: 905, dtype: object
```

Considering a random row where loan_default is 0 to check how much is the probability for that particular row to have loan_default as 1.

In [115…

```
df2.iloc[4410,:]
```

Out[115…

```
disbursed_amount                                41494
asset_cost                                      57850
ltv                                              74.5
branch_id                                          34
employment.type                         Self employed
age                                                50
```

```
mobileno_avl_flag                          1
aadhar_flag                                0
pan_flag                                   1
voterid_flag                               1
driving_flag                               0
passport_flag                              0
perform_cns.score                        378
delinquent.accts.in.last.six.months        0
credit.history.length                     44
no.of_inquiries                            0
loan_default                               0
Name: 4410, dtype: object
```

For loan_default = 1

In [116…

```python
l2 = [55913,94.69,0,0,15,35,300,0,50,1,0,0,1,0,1,0,60407]
```

In [117…

```python
importances['Effect'] = importances['Coefficient'] * l2
```

For loan_default = 0

In [118…

```python
l3 = [41494,74.5,0,0,34,44,378,1,50,1,1,1,0,0,0,0,57850]
```

In [119…

```python
importances['Effect2'] = importances['Coefficient'] * l3
```

In [120…

```python
importances
```

Out[120…

	Attribute	Coefficient	Effect	Effect2
0	disbursed_amount	1.418798	79329.231305	58871.588428
1	ltv	0.712966	67.510725	53.115947
2	delinquent.accts.in.last.six.months	0.134613	0.000000	0.000000
3	no.of_inquiries	0.074544	0.000000	0.000000
4	branch_id	0.047109	0.706634	1.601704
5	credit.history.length	-0.052909	-1.851828	-2.328013
6	perform_cns.score	-0.136082	-40.824639	-51.439045
7	pan_flag	-0.271665	-0.000000	-0.271665
8	age	-0.413380	-20.669023	-20.669023
9	mobileno_avl_flag	-0.464699	-0.464699	-0.464699
10	employment.type_Self employed	-0.610245	-0.000000	-0.610245
11	voterid_flag	-0.739503	-0.000000	-0.739503
12	employment.type_Salaried	-0.816998	-0.816998	-0.000000
13	driving_flag	-1.028998	-0.000000	-0.000000
14	aadhar_flag	-1.140330	-1.140330	-0.000000
15	passport_flag	-1.165155	-0.000000	-0.000000
16	asset_cost	-1.283031	-77504.039670	-74223.329994

In [121…
```
importances['Effect'].sum()
```
Out[121…
```
1827.6414754050784
```
In [122…
```
importances['Effect2'].sum()
```
Out[122…
```
-15373.546107364353
```
In [123…
```
z = round((intercept + (importances['Effect'].sum()))[0],2)
z
```
Out[123…
```
1827.16
```
In [124…
```
z2 = round((intercept + (importances['Effect2'].sum()))[0],2)
z2
```
Out[124…
```
-15374.02
```
In [125…
```
import math
```
In [126…
```
probability = 1 / (1 + math.exp(-z))
print("Probability of being a Loan Defaulter is: ",probability*100)
```
```
Probability of being a Loan Defaulter is:  100.0
```
In [127…
```
probability = 1 / (1 + math.exp(-z2))
print("Probability of being a Loan Defaulter is: ",probability*100)
```
```
---------------------------------------------------------------------------
OverflowError                             Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_33096/1490725659.py in <module>
----> 1 probability = 1 / (1 + math.exp(-z2))
      2 print("Probability of being a Loan Defaulter is: ",probability*100)

OverflowError: math range error
```
The above function, 1 / (1 + math.exp(-z2)) can only handle z2<=-230, since it is still more smaller than -230, it is assumed that the probability is 0%.

As expected we got the results correctly. Hence, attributes that make major changes in probability for becoming / not becoming a Defaulter are disbursed_amount, asset_cost, ltv, age,branch_id, credit.history.length and perform_cns.score, whose maximum values are big enough to easily manipulate the probability.

Model buliding with important features

In [128…
```
X2_train = X_train[['disbursed_amount','age','ltv','asset_cost','delinquent.accts.in.last.
                    'employment.type_Self employed','employment.type_Salaried','credit.his
                    'passport_flag','driving_flag','aadhar_flag']]
```
In [129…
```
X2_test = X_test[['disbursed_amount','age','ltv','asset_cost','delinquent.accts.in.last.si
                  'employment.type_Self employed','employment.type_Salaried','credit.histo
                  'passport_flag','driving_flag','aadhar_flag']]
```
In [130…

```
regressor3 = LogisticRegression()
regressor3 = regressor3.fit(X2_train,y_train)
```

In [131…

```
y_pred = regressor3.predict(X2_test)
```

In [132…

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

```
              precision    recall  f1-score   support

           0       0.58      0.54      0.56      3600
           1       0.58      0.62      0.60      3676

    accuracy                           0.58      7276
   macro avg       0.58      0.58      0.58      7276
weighted avg       0.58      0.58      0.58      7276
```

We can improve our model more by using other algorithms too, as the Logistic Regression even after feature engineering gave not so good metric values.

Decision Tree

Base Model

In [133…

```
from sklearn.tree import DecisionTreeClassifier
```

In [134…

```
clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X_train, y_train)
```

Out[134…

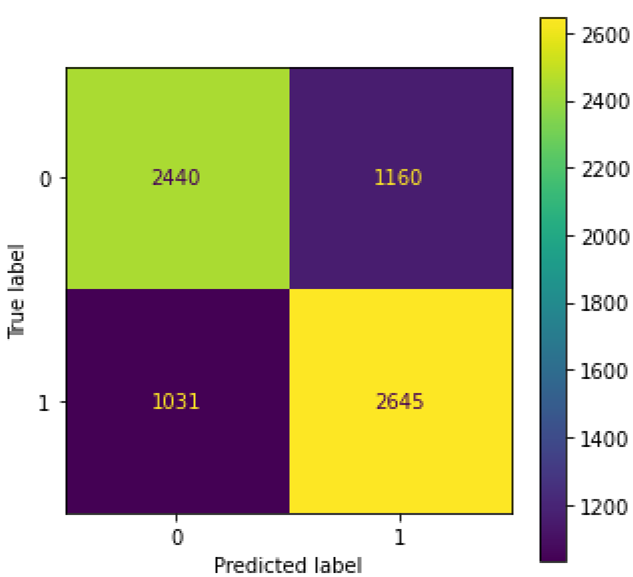
```
DecisionTreeClassifier(criterion='entropy')
```

In [135…

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

In [136…

```
fig, ax = plt.subplots(figsize=(5,5))
plot_confusion_matrix(clf, X_test, y_test, ax=ax)
plt.show()
```



In [137…

```
y_test[10:20]
```

Out[137...

	loan_default
25519	1
11686	0
4410	0
30239	1
17000	0
29772	1
25035	1
22309	0
34507	1
11766	1

In [138...

```
y_pred[10:20]
```

Out[138...

```
array([1, 1, 0, 1, 0, 1, 1, 0, 1, 1], dtype=int64)
```

One Mis-classification

In [139...

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

```
              precision    recall  f1-score   support

           0       0.70      0.68      0.69      3600
           1       0.70      0.72      0.71      3676

    accuracy                           0.70      7276
   macro avg       0.70      0.70      0.70      7276
weighted avg       0.70      0.70      0.70      7276
```

Feature Importance

In [140...

```
feat_imp = clf.feature_importances_
```

In [141...

```
cols = list(X_train.columns.values)
```

In [142...

```
pd.Series(feat_imp*100, index = cols).sort_values(ascending = False)
```

Out[142...

```
branch_id                          18.077817
age                                17.580159
ltv                                16.277165
disbursed_amount                   12.970502
asset_cost                         11.681624
perform_cns.score                   8.372643
credit.history.length               7.456233
no.of_inquiries                     2.133834
employment.type_Salaried            1.266587
employment.type_Self employed       0.996792
pan_flag                            0.809115
aadhar_flag                         0.735779
```

```
delinquent.accts.in.last.six.months     0.672780
voterid_flag                            0.657453
driving_flag                            0.300367
passport_flag                           0.011152
mobileno_avl_flag                       0.000000
dtype: float64
```

In [143…

```
from sklearn import tree
```

In [144…

```
new_X_train = X_train.drop(['mobileno_avl_flag','passport_flag'], axis=1)
```

In [145…

```
new_X_test = X_test.drop(['mobileno_avl_flag','passport_flag'], axis=1)
```

In [146…

```
clf2 = DecisionTreeClassifier(criterion = 'entropy')
clf2.fit(new_X_train, y_train)
```

Out[146…

```
DecisionTreeClassifier(criterion='entropy')
```

In [147…

```
y_pred = clf2.predict(new_X_test)
```

In [148…

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

```
              precision    recall  f1-score   support

           0       0.70      0.68      0.69      3600
           1       0.70      0.72      0.71      3676

    accuracy                           0.70      7276
   macro avg       0.70      0.70      0.70      7276
weighted avg       0.70      0.70      0.70      7276
```

The metric results from Decision Tree model are quite good. Hyper Parameter Pruning can be done to further improve the model.

In [149…

```
from sklearn.model_selection import GridSearchCV
```

In [151…

```
mod = GridSearchCV(clf2, param_grid =
                  {'max_depth':[i for i in range (20,30)],
                  'max_leaf_nodes':[i for i in range (1000,2000,100)],
                  'min_samples_leaf':[i for i in range (40,60,5)]})
mod.fit(new_X_train, y_train)
```

Out[151…

```
GridSearchCV(estimator=DecisionTreeClassifier(criterion='entropy'),
             param_grid={'max_depth': [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
                         'max_leaf_nodes': [1000, 1100, 1200, 1300, 1400, 1500,
                                            1600, 1700, 1800, 1900],
                         'min_samples_leaf': [40, 45, 50, 55]})
```

In [152…

```
mod.best_estimator_
```

Out[152…

```
DecisionTreeClassifier(criterion='entropy', max_depth=25, max_leaf_nodes=1300,
                       min_samples_leaf=45)
```

In [153…

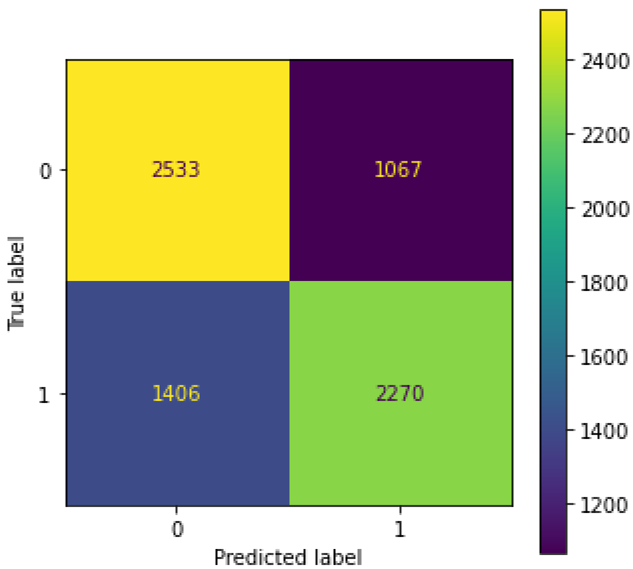
```
clf3 = DecisionTreeClassifier(criterion = 'entropy', splitter = 'best',
                              max_leaf_nodes = 1300, min_samples_leaf = 45, max_depth = 25
clf3.fit(new_X_train, y_train)
```

Out[153…

```
DecisionTreeClassifier(criterion='entropy', max_depth=25, max_leaf_nodes=1300,
                       min_samples_leaf=45)
```

In [154…

```
fig, ax = plt.subplots(figsize=(5,5))
plot_confusion_matrix(clf3, new_X_test, y_test, ax=ax)
plt.show()
```



In [155…

```
y_pred = clf3.predict(new_X_test)
```

In [156…

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

```
              precision    recall  f1-score   support

           0       0.64      0.70      0.67      3600
           1       0.68      0.62      0.65      3676

    accuracy                           0.66      7276
   macro avg       0.66      0.66      0.66      7276
weighted avg       0.66      0.66      0.66      7276
```

In [157…

```
probs = clf3.predict_proba(new_X_test)
```

In [158…

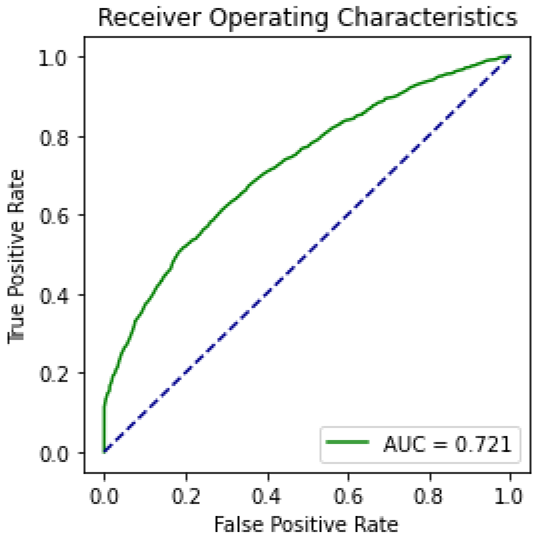
```
prob_positive = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, prob_positive)
roc_auc = metrics.auc(fpr, tpr)
print("Area under the curve: ", roc_auc)
```

```
Area under the curve:  0.7213402626647323
```

In [159…

```
plt.rcParams["figure.figsize"] = (4,4)
plt.title("Receiver Operating Characteristics")
plt.plot(fpr,tpr,'green',label='AUC = %0.3f' %roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0,1],[0,1], color='darkblue', linestyle='--')
plt.ylabel('True Positive Rate')
```

```
plt.xlabel('False Positive Rate')
plt.show()
```



The results are better before pruning the hyper parameters.

Bagging classifier with base estimator as Decision tree

In [160…

```
from sklearn.ensemble import BaggingClassifier
```

In [161…

```
ls = []
for w in range (250, 400, 10):
    clf4 = BaggingClassifier(oob_score = True, n_estimators = w, random_state = 400, max_s
                             base_estimator = DecisionTreeClassifier())
    clf4.fit(new_X_train, y_train)
    oob = clf4.oob_score_
    print("For n_estimators = ", w)
    print("OOB score is ", oob)
    print("************************************")
    ls.append((oob,w))
```

```
For n_estimators =  250
OOB score is  0.7965775548072297
************************************
For n_estimators =  260
OOB score is  0.7957185073190846
************************************
For n_estimators =  270
OOB score is  0.7963026596110233
************************************
For n_estimators =  280
OOB score is  0.7965088310081782
************************************
For n_estimators =  290
OOB score is  0.7955810597209814
************************************
For n_estimators =  300
OOB score is  0.7964744691086523
************************************
For n_estimators =  310
OOB score is  0.7965431929077039
************************************
For n_estimators =  320
OOB score is  0.7971960689986942
************************************
For n_estimators =  330
```

```
OOB score is  0.7969555357020136
***********************************
For n_estimators =  340
OOB score is  0.7969898976015394
***********************************
For n_estimators =  350
OOB score is  0.7962682977114975
***********************************
For n_estimators =  360
OOB score is  0.7962682977114975
***********************************
For n_estimators =  370
OOB score is  0.7964744691086523
***********************************
For n_estimators =  380
OOB score is  0.7968524500034362
***********************************
For n_estimators =  390
OOB score is  0.7967493643048588
***********************************
```

In [162…

```
print(max(ls))
```

```
(0.7971960689986942, 320)
```

In [163…

```
clf4 = BaggingClassifier(oob_score = True, n_estimators = 320, random_state = 400,
                        base_estimator = DecisionTreeClassifier())
clf4.fit(new_X_train, y_train)
```

Out[163…

```
BaggingClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=320,
                  oob_score=True, random_state=400)
```

In [164…

```
y_pred = clf4.predict(new_X_test)
```

In [165…

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

```
              precision    recall  f1-score   support

           0       0.79      0.80      0.80      3600
           1       0.80      0.79      0.80      3676

    accuracy                           0.80      7276
   macro avg       0.80      0.80      0.80      7276
weighted avg       0.80      0.80      0.80      7276
```

In [166…

```
probs = clf4.predict_proba(new_X_test)
```

In [167…

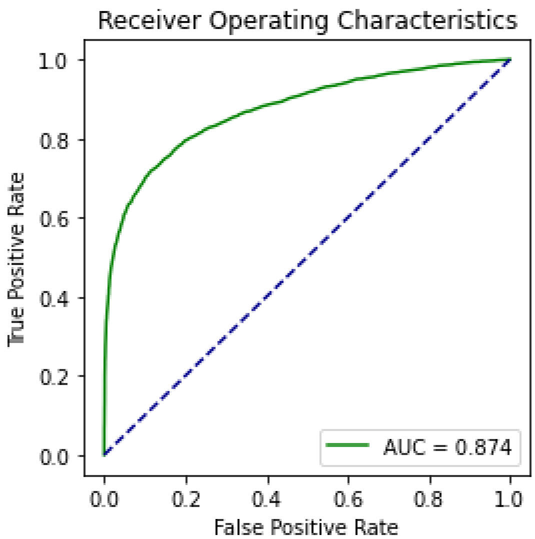
```
prob_positive = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, prob_positive)
roc_auc = metrics.auc(fpr, tpr)
print("Area under the curve: ", roc_auc)
```

```
Area under the curve:  0.8743862592189579
```

In [168…

```
plt.rcParams["figure.figsize"] = (4,4)
plt.title("Receiver Operating Characteristics")
plt.plot(fpr,tpr,'green',label='AUC = %0.3f' %roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0,1],[0,1], color='darkblue', linestyle='--')
```

```
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



The Bagging Classifier gave very good metric results compared to Base Decision Trees.

Random Forest Algorithm

In [169…

```
from sklearn.ensemble import RandomForestClassifier
```

In [170…

```
ls2 = []
for w in range (250, 400, 10):
    clf5 = RandomForestClassifier(oob_score = True, n_estimators = w, random_state = 400)
    clf5.fit(new_X_train, y_train)
    oob = clf5.oob_score_
    print("For n_estimators = ", w)
    print("OOB score is ", oob)
    print("***********************************")
    ls2.append((oob,w))
```

```
For n_estimators =  250
OOB score is  0.7851006803656106
***********************************
For n_estimators =  260
OOB score is  0.7848601470689299
***********************************
For n_estimators =  270
OOB score is  0.7857191945570751
***********************************
For n_estimators =  280
OOB score is  0.7865438801456944
***********************************
For n_estimators =  290
OOB score is  0.7868531372414267
***********************************
For n_estimators =  300
OOB score is  0.7868531372414267
***********************************
For n_estimators =  310
OOB score is  0.7874372895333654
***********************************
For n_estimators =  320
OOB score is  0.7873685657343138
***********************************
For n_estimators =  330
```

```
OOB score is  0.7872998419352621
***********************************
For n_estimators =  340
OOB score is  0.7870593086385815
***********************************
For n_estimators =  350
OOB score is  0.7868187753419009
***********************************
For n_estimators =  360
OOB score is  0.7870249467390558
***********************************
For n_estimators =  370
OOB score is  0.7858222802556525
***********************************
For n_estimators =  380
OOB score is  0.7858910040547041
***********************************
For n_estimators =  390
OOB score is  0.7874029276338396
***********************************
```

In [171…

```
print(max(ls2))
```

```
(0.7874372895333654, 310)
```

In [172…

```
clf5 = RandomForestClassifier(oob_score = True, n_estimators = 310, random_state = 400)
clf5.fit(new_X_train, y_train)
```

Out[172…

```
RandomForestClassifier(n_estimators=310, oob_score=True, random_state=400)
```

In [173…

```
probs = clf5.predict_proba(new_X_test)
```

In [174…

```
prob_positive = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, prob_positive)
roc_auc = metrics.auc(fpr, tpr)
print("Area under the curve: ", roc_auc)
```

```
Area under the curve:  0.8665408505622052
```

In [175…

```
y_pred = clf5.predict(new_X_test)
```

In [176…

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

```
              precision    recall  f1-score   support

           0       0.78      0.78      0.78      3600
           1       0.79      0.79      0.79      3676

    accuracy                           0.78      7276
   macro avg       0.78      0.78      0.78      7276
weighted avg       0.78      0.78      0.78      7276
```

Random Forest Algorithm and Bagging Classifier gave almost similar metric values. Here precision, recall and F1-score are all same and quite impressive. Hence, we can say model is performing very good.

AdaBoost Algorithm

In [177…

```
from sklearn.ensemble import AdaBoostClassifier
```

In [178…

```
ls3 = []
for w in range (1800, 2500, 100):
    clf6 = AdaBoostClassifier(n_estimators = w, random_state = 400)
    clf6.fit(new_X_train, y_train)
    Score = clf6.score(new_X_test, y_test)
    print("For n_estimators = ", w)
    print("Score is ", Score)
    print("***********************************")
    ls3.append((Score,w))
```

```
For n_estimators =  1800
Score is  0.8003023639362287
***********************************
For n_estimators =  1900
Score is  0.8007146783947223
***********************************
For n_estimators =  2000
Score is  0.802913688840022
***********************************
For n_estimators =  2100
Score is  0.804150632215503
***********************************
For n_estimators =  2200
Score is  0.8067619571192963
***********************************
For n_estimators =  2300
Score is  0.8082737768004398
***********************************
For n_estimators =  2400
Score is  0.810197910940077
***********************************
```

In [179…

```
print(max(ls3))
```

```
(0.810197910940077, 2400)
```

In [180…

```
clf6 = AdaBoostClassifier(n_estimators = 2400, random_state = 400)
clf6.fit(new_X_train, y_train)
```

Out[180…

```
AdaBoostClassifier(n_estimators=2400, random_state=400)
```

In [181…

```
probs = clf6.predict_proba(new_X_test)
```

In [182…

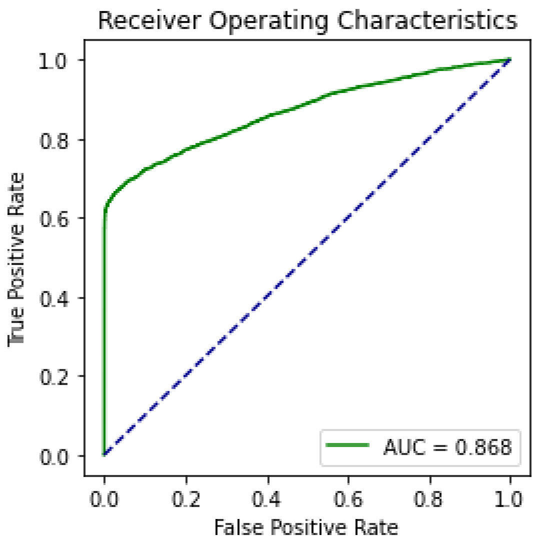
```
prob_positive = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, prob_positive)
roc_auc = metrics.auc(fpr, tpr)
print("Area under the curve: ", roc_auc)
```

```
Area under the curve:  0.8684503460887438
```

In [183…

```
plt.rcParams["figure.figsize"] = (4,4)
plt.title("Receiver Operating Characteristics")
plt.plot(fpr,tpr,'green',label='AUC = %0.3f' %roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0,1],[0,1], color='darkblue', linestyle='--')
plt.ylabel('True Positive Rate')
```

```
plt.xlabel('False Positive Rate')
plt.show()
```



In [184…

```
y_pred = clf6.predict(new_X_test)
```

In [185…

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

```
              precision    recall  f1-score   support

           0       0.75      0.91      0.83      3600
           1       0.89      0.71      0.79      3676

    accuracy                           0.81      7276
   macro avg       0.82      0.81      0.81      7276
weighted avg       0.82      0.81      0.81      7276
```

XG-Boost Algorithm

In [186…

```
import xgboost
```

In [187…

```
from xgboost import XGBClassifier
```

In [188…

```
ls4 = []
for w in range (200, 300, 10):
    xgb = XGBClassifier(n_estimators = w, random_state = 400)
    xgb.fit(new_X_train, y_train)
    Score = xgb.score(new_X_test, y_test)
    print("For n_estimators = ", w)
    print("Score is ", Score)
    print("***********************************")
    ls4.append((Score,w))
```

```
For n_estimators =  200
Score is  0.8228422210005497
***********************************
For n_estimators =  210
Score is  0.8228422210005497
***********************************
For n_estimators =  220
Score is  0.8227047828477185
```

```
**********************************
For n_estimators =  230
Score is  0.8209180868609126
**********************************
For n_estimators =  240
Score is  0.823117097306212
**********************************
For n_estimators =  250
Score is  0.8233919736118747
**********************************
For n_estimators =  260
Score is  0.8247663551401869
**********************************
For n_estimators =  270
Score is  0.8243540406816933
**********************************
For n_estimators =  280
Score is  0.8239417262231996
**********************************
For n_estimators =  290
Score is  0.8236668499175371
**********************************
```

In [189…

```
print(max(ls4))
```

```
(0.8247663551401869, 260)
```

In [190…

```
xgb = XGBClassifier(n_estimators=260, random_state=400)
xgb.fit(new_X_train, y_train)
xgb.score(new_X_test, y_test)
```

Out[190…

```
0.8247663551401869
```

In [191…

```
probs = xgb.predict_proba(new_X_test)
```

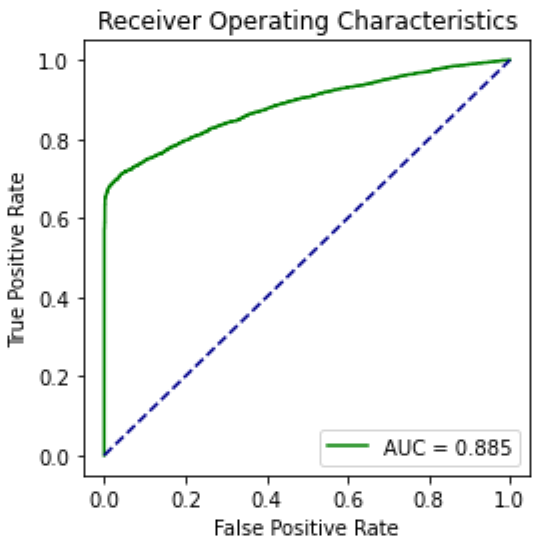
In [192…

```
prob_positive = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, prob_positive)
roc_auc = metrics.auc(fpr, tpr)
print("Area under the curve: ", roc_auc)
```

```
Area under the curve:  0.884682852738484
```

In [193…

```
plt.rcParams["figure.figsize"] = (4,4)
plt.title("Receiver Operating Characteristics")
plt.plot(fpr,tpr,'green',label='AUC = %0.3f' %roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0,1],[0,1], color='darkblue', linestyle='--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```




```
In [194…  y_pred = xgb.predict(new_X_test)
```

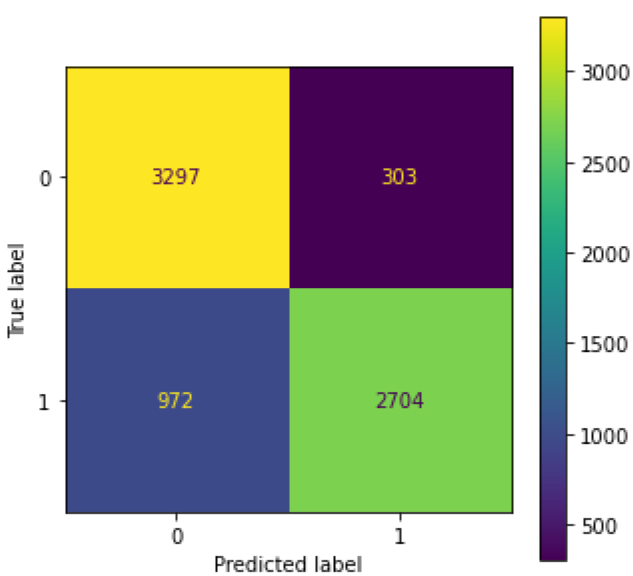
```
In [195…  print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.77      0.92      0.84      3600
           1       0.90      0.74      0.81      3676

    accuracy                           0.82      7276
   macro avg       0.84      0.83      0.82      7276
weighted avg       0.84      0.82      0.82      7276
```

```
In [196…  fig, ax = plt.subplots(figsize=(5,5))
          plot_confusion_matrix(xgb, new_X_test, y_test, ax=ax)
          plt.show()
```



We got very good metrics from XGBoost.
From above confusion matrix, we can say that both TN and TP have increased which resulted in good recall and precision. Its clearly shown that FP drastically got reduced to 303 from 1660 and FN got reduced to 972 from 1391 which was from Logistic Regression after SMOTE. The current model is performing very good.

SVM Classifier

In [212...

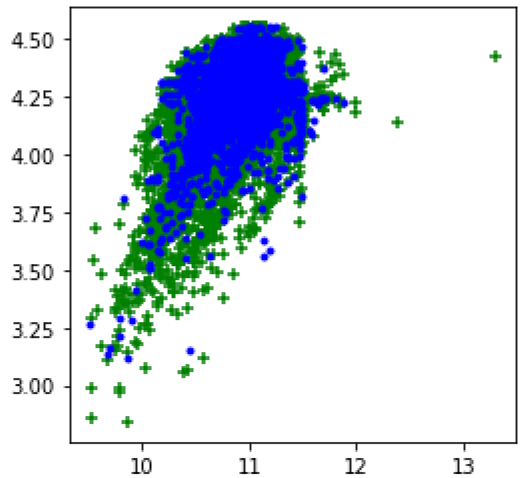
```
df0 = df_encoded[df_encoded.loan_default==0]
df1 = df_encoded[df_encoded.loan_default==1]
```

In [213...

```
plt.scatter(df0['disbursed_amount'], df0['ltv'], color='green', marker="+")
plt.scatter(df1['disbursed_amount'], df1['ltv'], color='blue', marker=".")
```

Out[213...

```
<matplotlib.collections.PathCollection at 0x23478c9dca0>
```

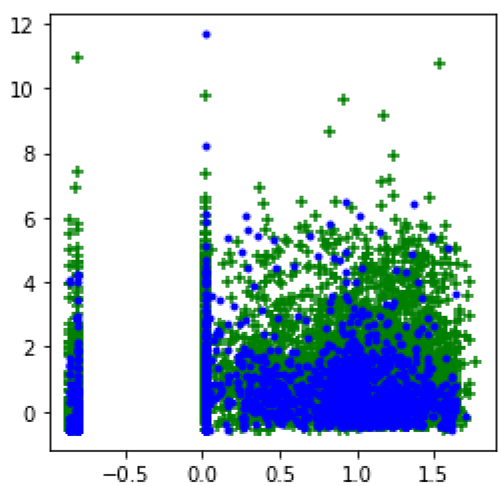


In [214...

```
plt.scatter(df0['perform_cns.score'], df0['credit.history.length'], color='green', marker=
plt.scatter(df1['perform_cns.score'], df1['credit.history.length'], color='blue', marker='
```

Out[214...

```
<matplotlib.collections.PathCollection at 0x234793ee280>
```



Complexity for drawing a boundary to classify the classes will be too high for SVM.

In [215...

```
from sklearn.svm import SVC
model = SVC()
```

In [216...

```
model.fit(new_X_train, y_train)
```

Out[216...

```
SVC()
```

In [217...

```
model.score(new_X_test,y_test)
```

Out[217...

```
0.5835623969213853
```

In [218…

```
y_pred2 = model.predict(new_X_test)
```

In [220…

```
print(classification_report(y_test,y_pred2))
```

```
              precision    recall  f1-score   support

           0       0.61      0.45      0.52      3600
           1       0.57      0.72      0.63      3676

    accuracy                           0.58      7276
   macro avg       0.59      0.58      0.58      7276
weighted avg       0.59      0.58      0.58      7276
```

Recall for minority class is quite good, but SVM classifier for this dataset is not really classifying majority class well. F1-score is also not impressive, almost 50% for majority class, just like random guessing. Hence, we can go ahead with **XGBoost classifier** which is performing very good on our model for both classes.