How to predict Loan Default using Machine Learning Models

Author: Keyura Vadlamani
Submitted for: Analytics Vidhya Machine Learning Summer Training

Can you predict if an applicant will default the loan or not in the future?

Problem

MyHom is a finance company that lends housing loans at the best and most affordable interest rates to customers. In recent times, the company incurred heavy losses due to loan defaults. Most applicants failed to repay the loan as per the promissory note.

In order to avoid such losses, the company has decided to build a system for identifying the loan defaulters automatically based on data. This will help the company to identify the potential applicants and ensure the smooth running of the entire process.

Now, the company challenges the Data Science community to build a smart AI system to predict the probability of an applicant defaulting the loan or not in the future.

To summarize: In Data Science terms ,this problem falls into the category of "Binary Classification" problem in which the target variable (Loan Default) (Which we are supposed to predict) can have two values ("Yes" OR "No")

Training dataset : It consists of details of the customer (like Age, Asset Cost , No.of loans, No.of Current loans,Education, Loan Amount and others.) along with the target column which is Loan_Default with values : 0 & 1 **Test Dataset :** It consists of similar details for new customers (like Age, Asset Cost , No.of loans,

No.of Current loans,Education, Loan Amount and others), however without the target column (i.e. Loan_Default) Using the training dataset we need to prepare a Machine Learning model which will predict the Loan_Default for the customers in the test dataset (i.e. For each customer in the test dataset predict whether the loan a will get defaulted or not)

****What would be the Strategy? ****

- Divide the solution approach in simple 8-steps and execute them one after the other

1. Load essential Python Libraries
2. Load Training/Test datasets in Python environment.
3. Exploratory data analysis (EDA).
4. Impute Missing values.
5. Feature Engineering.
6. Build Machine Learning Model
7. Make a prediction on test dataset & Prepare submission file with the final prediction
8. Conclusion

1)Importing Necessary Libraries and Loading Datasets

- Firstly we will import the necessary python libraries and load datasets- train, test, submission

```
In [1]: #Importing Necessary Libraries
        import numpy as np # linear algebra
        import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
        import matplotlib.pyplot as plt
        import seaborn as sns #importing seaborn module
        import warnings
        import os
        from sklearn.model_selection import train_test_split
        warnings.filterwarnings('ignore')  #this will ignore the warnings.it wont display warnings in notebook
```

```
In [4]: train = pd.read_csv("train_LZV4RXX.csv") # Load training dataset
        test = pd.read_csv("test_4zJg83n.csv") # Load test dataset
        submission = pd.read_csv("sample_submission_tbPU9qQ.csv") # Load final submission dataset
```

2) Understanding Train, Test and Submission files Data

- The train and test set contains the different attributes related to demographic and loan information of the applicants such as age, profession, no. of active loans, loan default in previous loans, and so on. The training set contains the target variable *loan_default* and you need to predict the target variable in the test set.

Variable	**Description**
loan_id	Unique identifier of a loan
age	Age of the Applicant
Education	Applicant Education
proof_submitted	Type of proof submitted
loan_amount	Loan Amount Disbursed
asset_cost	The total asset value of the applicant
no_of_loans	No. of the loans taken by the applicant
no_of_curr_loans	No. of active loans held by the applicant
last_delinq_none	The loan defaulted in at least one of the past loans
loan_default (Target Variable)	*0/1 indicating if an applicant will default the loan or not*

```
In [5]: print("Data Types in Train Data:-\n")
        print(train.info(),'\n')

        print("Shape of Train Data:-\n")
        print(train.shape,'\n')

        print("-------------------------------------------------------------------------------------------------------")

        print("\nData Types in Test Data:-\n")
        print(test.info(),'\n')

        print("\nShape of Test Data:-\n")
        print(test.shape,'\n')

        print("-------------------------------------------------------------------------------------------------------")

        print("\nData Types in Submission Data:-\n")
        print(submission.info(),'\n')

        print("\nShape of Submission Data:-\n")
        print(submission.shape,'\n')

        print("-------------------------------------------------------------------------------------------------------")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7000 entries, 0 to 6999
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   loan_id           7000 non-null   int64  
 1   age               7000 non-null   int64  
 2   education         6755 non-null   float64
 3   proof_submitted   7000 non-null   object 
 4   loan_amount       7000 non-null   int64  
 5   asset_cost        7000 non-null   int64  
 6   no_of_loans       7000 non-null   int64  
 7   no_of_curr_loans  7000 non-null   int64  
 8   last_delinq_none  7000 non-null   int64  
 9   loan_default      7000 non-null   int64  
dtypes: float64(1), int64(8), object(1)
memory usage: 547.0+ KB
None

Shape of Train Data:-

(7000, 10)


Data Types in Test Data:-

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   loan_id           3000 non-null   int64  
 1   age               3000 non-null   int64  
 2   education         2908 non-null   float64
 3   proof_submitted   3000 non-null   object 
 4   loan_amount       3000 non-null   int64  
 5   asset_cost        3000 non-null   int64  
 6   no_of_loans       3000 non-null   int64  
 7   no_of_curr_loans  3000 non-null   int64  
 8   last_delinq_none  3000 non-null   int64  
dtypes: float64(1), int64(7), object(1)
memory usage: 211.1+ KB
None


Shape of Test Data:-

(3000, 9)
```

```
----------------------------------------------------------

Data Types in Submission Data:-

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 2 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   loan_id       3000 non-null   int64
 1   loan_default  3000 non-null   int64
dtypes: int64(2)
memory usage: 47.0 KB
None


Shape of Submission Data:-

(3000, 2)
```

In [6]:

Slide Type: Sub-Slide

```
print("About Train Data:-\n")
print(train.head(),'\n')
print("-------------------------------------------------------------------------------------------------------------------")

print("\nAbout Test Data:-\n")
print(test.head(),'\n')
print("-------------------------------------------------------------------------------------------------------------------")

print("\nAbout Submission Data:-\n")
print(submission.head(),'\n')
```

```
About Train Data:-

   loan_id  age  education proof_submitted  loan_amount  asset_cost  \
0        1   27        1.0          Aadhar       504264      820920
1        2   48        1.0          Aadhar       728556      831444
2        3   30        2.0         VoterID       642936      826092
3        4   28        1.0          Aadhar       746556      930924
4        5   29        1.0          Aadhar      1139880     1902000

   no_of_loans  no_of_curr_loans  last_delinq_none  loan_default
0            2                 2                 0             0
1            6                 2                 0             0
2            0                 0                 0             1
3            0                 0                 0             0
4            0                 0                 0             0

-------------------------------------------------------------------------------------------------------------------

About Test Data:-

   loan_id  age  education proof_submitted  loan_amount  asset_cost  \
0     7001   29        1.0          Aadhar       636936      768240
1     7002   28        1.0          Aadhar       548988      693060
2     7003   28        1.0          Aadhar       651756      936600
3     7004   45        2.0          Aadhar       614676      744840
4     7005   48        1.0          Aadhar       625236      839400

   no_of_loans  no_of_curr_loans  last_delinq_none
0            2                 2                 0
1            3                 3                 0
2            0                 0                 0
3            4                 3                 0
4            0                 0                 0

-------------------------------------------------------------------------------------------------------------------
```

About Submission Data:-

	loan_id	loan_default
0	7001	1
1	7002	1
2	7003	1
3	7004	1
4	7005	1

- Then we perform statistical analysis on Train, Test Data Sets

In [8]:
```
train.describe(include='all')
# Below Line will provide Statistical Insight of data for Numerical Variable Only
# train.describe()
```

Out[8]:

	loan_id	age	education	proof_submitted	loan_amount	asset_cost	no_of_loans	no_of_curr_loans	last_delinq_none	loan_default
count	7000.000000	7000.000000	6755.000000	7000	7.000000e+03	7.000000e+03	7000.000000	7000.000000	7000.000000	7000.000000
unique	NaN	NaN	NaN	5	NaN	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	NaN	Aadhar	NaN	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	NaN	5931	NaN	NaN	NaN	NaN	NaN	NaN
mean	3500.500000	36.096571	1.561954	NaN	6.633552e+05	9.162998e+05	2.853286	1.371143	0.013286	0.400000
std	2020.870275	7.587700	0.496184	NaN	1.498128e+05	2.144922e+05	5.471932	2.189278	0.114504	0.489933
min	1.000000	21.000000	1.000000	NaN	1.678800e+05	4.733520e+05	0.000000	0.000000	0.000000	0.000000
25%	1750.750000	29.000000	1.000000	NaN	5.777880e+05	7.979010e+05	0.000000	0.000000	0.000000	0.000000

In [9]:
```
test.describe(include='all')

# Below Line will provide Statistical Insight of data for Numerical Variable Only
# test.describe()
```

Out[9]:

	loan_id	age	education	proof_submitted	loan_amount	asset_cost	no_of_loans	no_of_curr_loans	last_delinq_none
count	3000.000000	3000.000000	2908.000000	3000	3.000000e+03	3.000000e+03	3000.000000	3000.000000	3000.000000
unique	NaN	NaN	NaN	5	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	NaN	Aadhar	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	NaN	2545	NaN	NaN	NaN	NaN	NaN
mean	8500.500000	36.111000	1.563618	NaN	6.629294e+05	9.106000e+05	2.819000	1.374333	0.014333
std	866.169729	7.467347	0.496022	NaN	1.451009e+05	2.070278e+05	5.329575	2.218825	0.118881
min	7001.000000	21.000000	1.000000	NaN	1.696800e+05	4.706040e+05	0.000000	0.000000	0.000000
25%	7750.750000	30.000000	1.000000	NaN	5.801880e+05	7.930710e+05	0.000000	0.000000	0.000000
50%	8500.500000	36.000000	2.000000	NaN	6.541560e+05	8.569080e+05	0.000000	0.000000	0.000000
75%	9250.250000	42.000000	2.000000	NaN	7.315590e+05	9.501360e+05	4.000000	2.000000	0.000000
max	10000.000000	50.000000	2.000000	NaN	1.576920e+06	2.358180e+06	92.000000	32.000000	1.000000

3) Checking for Null values & their Percentage

- We will check for missing values in both train and test datasets and calculate their percentage

In [10]:
```
print("Checking Null entries & their Percentage in Train Data:-\n\n")
missing_val_train=pd.DataFrame(zip(train.isnull().sum(),train.isnull().sum()*100/len(train)),columns=['Missing Values','Percentag
missing_val_train
```

Checking Null entries & their Percentage in Train Data:-

Out[10]:

	Missing Values	Percentage Missing Values
loan_id	0	0.0
age	0	0.0
education	245	3.5
proof_submitted	0	0.0
loan_amount	0	0.0
asset_cost	0	0.0
no_of_loans	0	0.0

In [11]:
```
print("Checking Null entries & their Percentage in Test Data:-\n\n")
missing_val_test=pd.DataFrame(zip(test.isnull().sum(),test.isnull().
missing_val_test
```

```
Checking Null entries & their Percentage in Test Data:-
```

Out[11]:

	Missing Values	Percentage Missing Values
loan_id	0	0.000000
age	0	0.000000
education	92	3.066667
proof_submitted	0	0.000000
loan_amount	0	0.000000
asset_cost	0	0.000000
no_of_loans	0	0.000000
no_of_curr_loans	0	0.000000
last_delinq_none	0	0.000000

In [12]:
```
#EDA
```

4) Exploratory Data Analysis

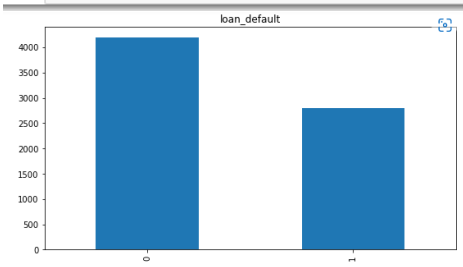
Training Dataset columns can be categorized into:

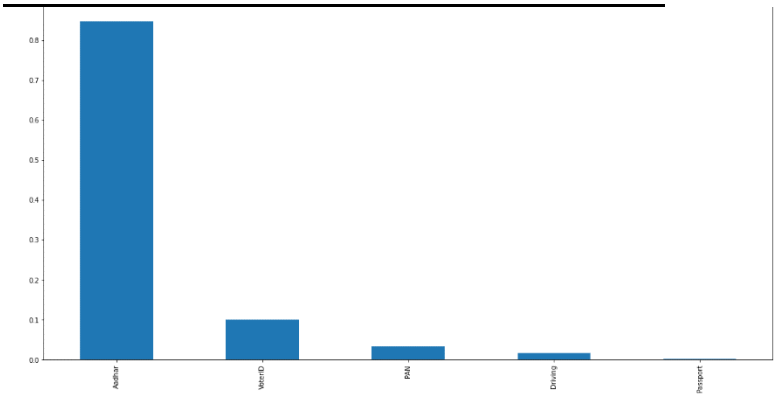
***Categorical Columns**: Age, Education (1.0 / 2.0), Loan Default (0/1)(i. e. Target variable),No.of loans, No.of Current loans, last_delinq none*

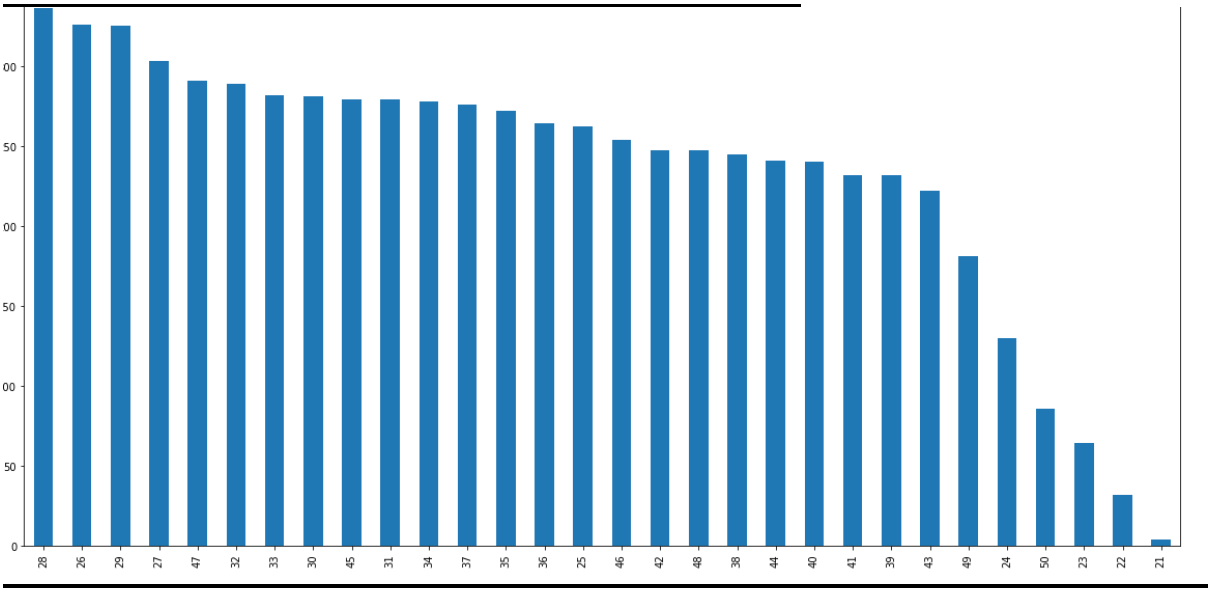
***Numerical Columns:** Loan ID, Loan Amount , Asset Cost*

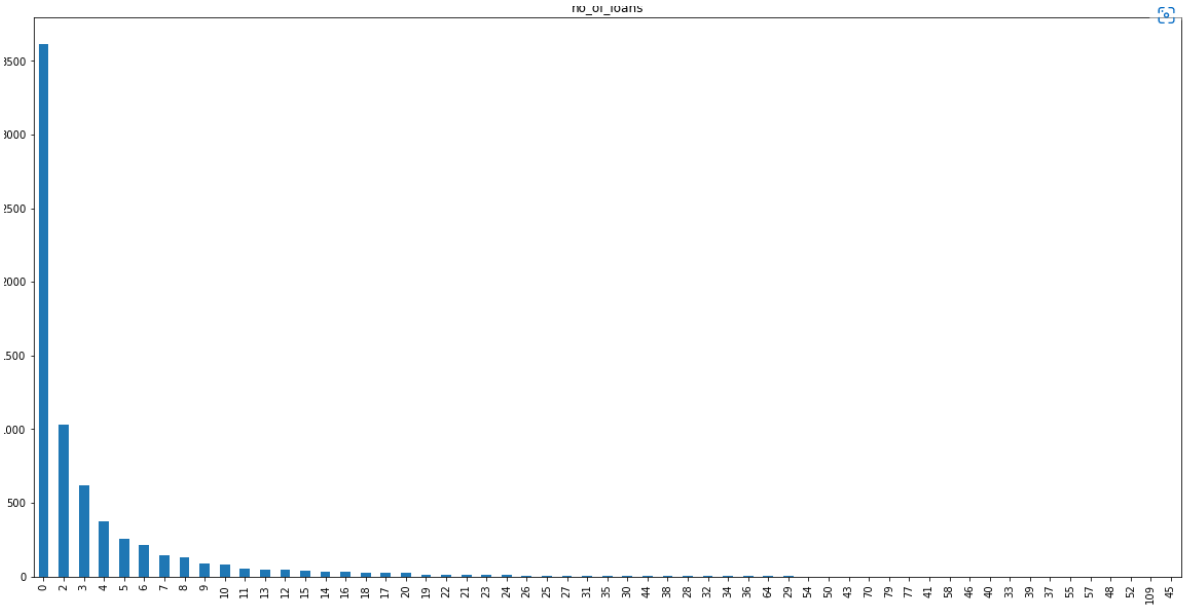
- Lets start with Univariate Analysis

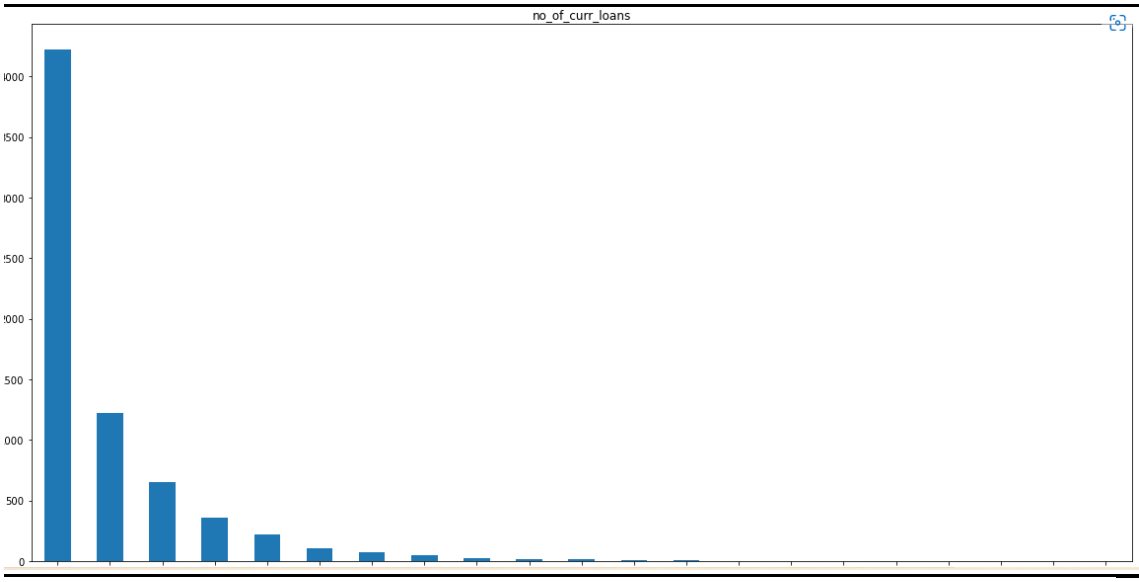
12]:
```
#EDA

train['loan_default'].value_counts().plot.bar(figsize=(9,5), title='loan_default')
plt.show()
train['proof_submitted'].value_counts(normalize=True).plot.bar(figsize=(20,10), title='proof_submitted')
plt.show()
train['age'].value_counts().plot.bar(figsize=(20,10), title='age')
plt.show()
train['no_of_loans'].value_counts().plot.bar(figsize=(20,10), title='no_of_loans')
plt.show()
train['no_of_curr_loans'].value_counts().plot.bar(figsize=(20,10), title='no_of_curr_loans')
plt.show()
train['education'].value_counts().plot.bar(title='education')
plt.show()
train['last_delinq_none'].value_counts().plot.bar(title='last_delinq_none')
plt.show()
sns.distplot(train['asset_cost'])
plt.show()
sns.distplot(train['loan_amount'])
plt.show()
```

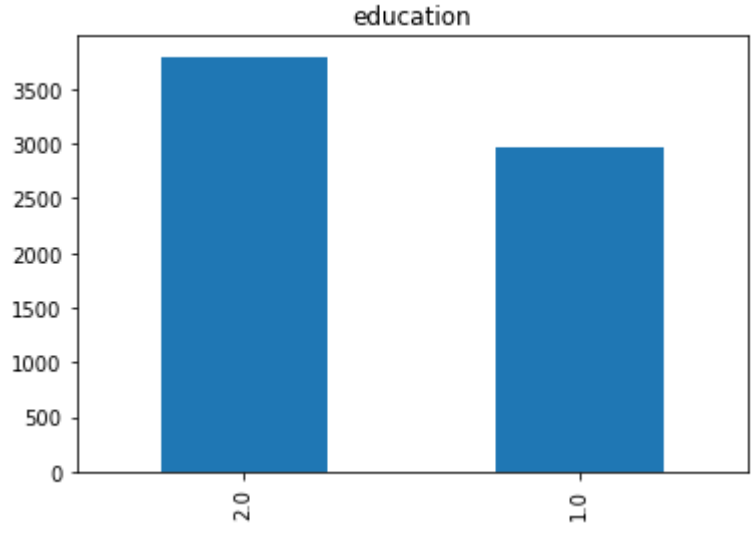


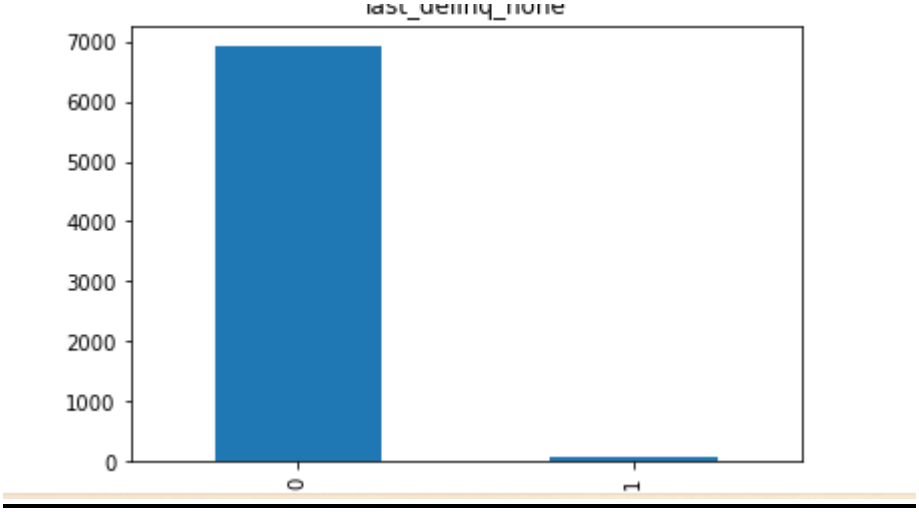


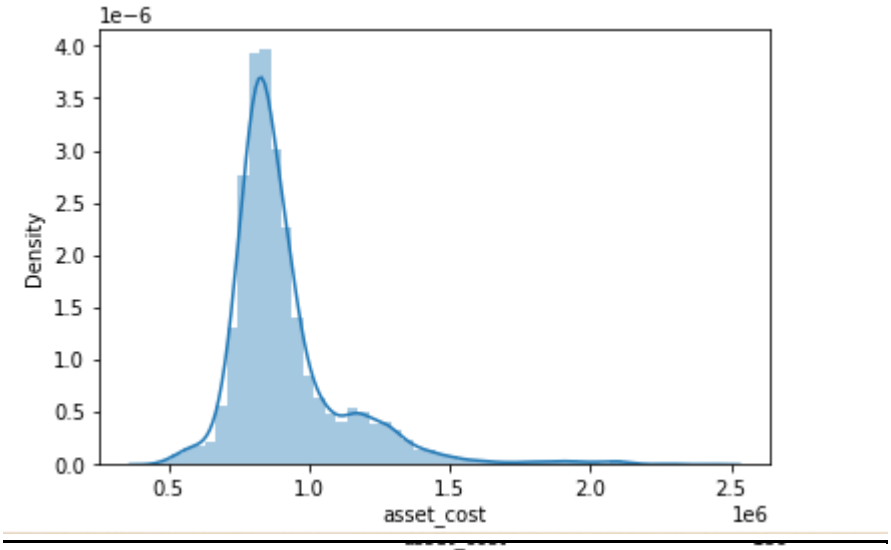


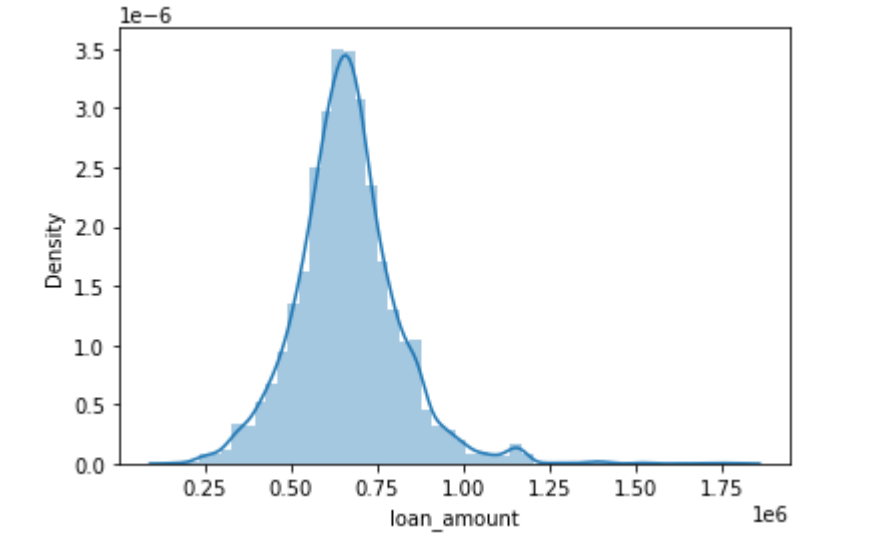

no_of_loans


no_of_curr_loans


education
3500
3000
2500
2000
1500
1000
500
0
2.0
1.0







Univariate analysis Observations:

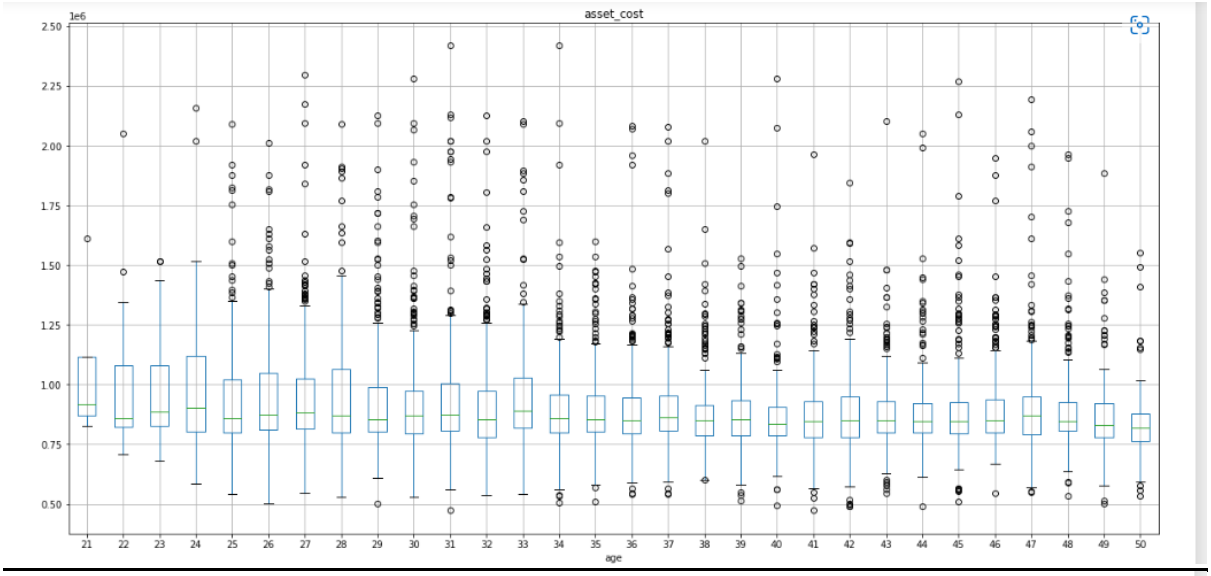
1. More Loans are not defaulted Vs Defaulted
2. Majority of the customers submitted Aadhar as proof
3. Age group of customers is between 21-50 with 21 being the least customers and 28 being the highest number of customers
4. Count of customers who didn't take a loan before is more than that of those who took a loan before
5. Count of customers who does not have any current loans is more than that of those who have current loans
6. Maximum customers are educated
7. Loan has not defaulted for most customers in the past
8. Asset cost and loan amount are proportional

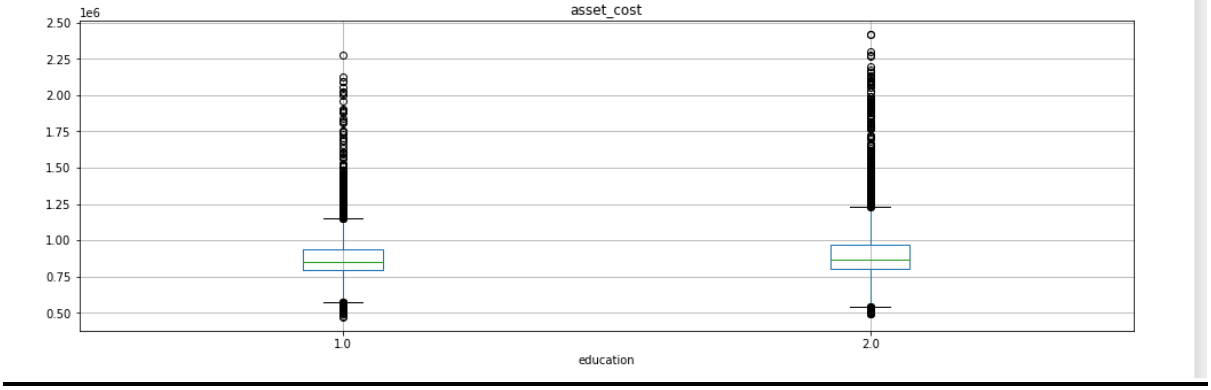
- **Bivariate Analysis using Catplot, Box Plot and Crosstab**

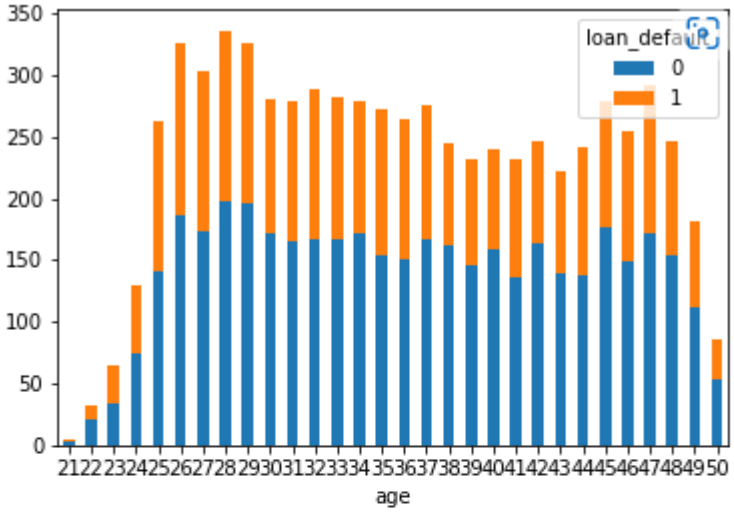
In [13]:

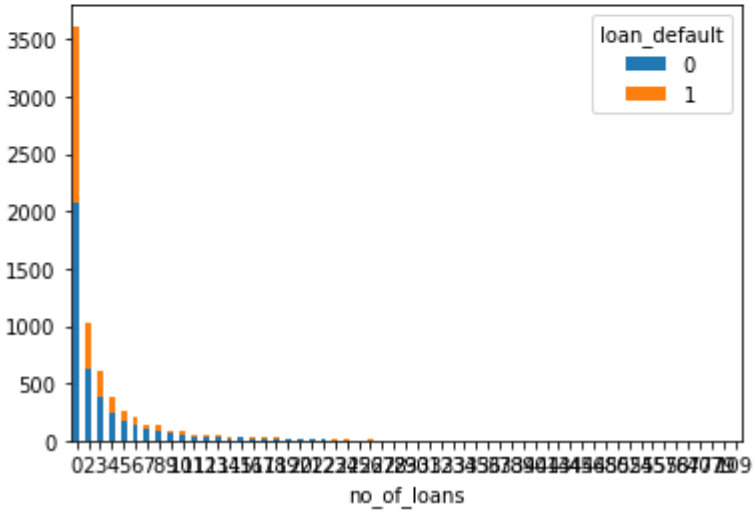
Slide Type

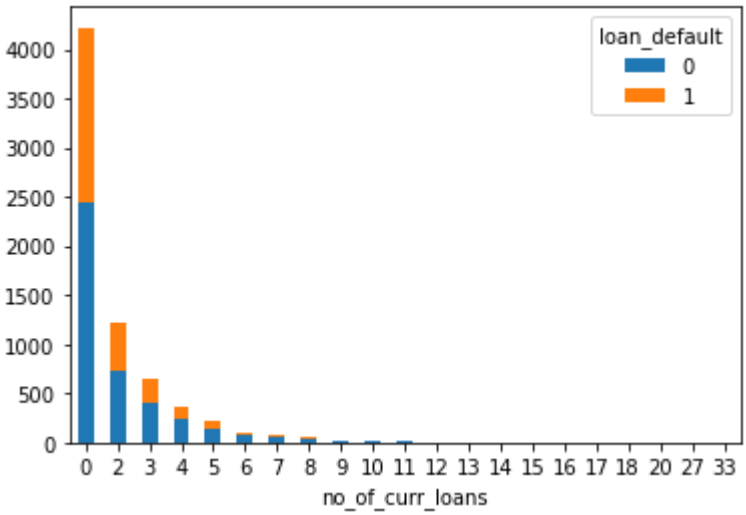
```
#Exploring independent variables vs target variables
train.boxplot(column='asset_cost', by = 'age',figsize=(20,10))
plt.suptitle('')
train.boxplot(column='asset_cost', by = 'education',figsize=(16,5))
plt.suptitle('')
ct=pd.crosstab(train['age'],train['loan_default'])
ct.plot(kind="bar",stacked=True,rot=0)
plt.show()
nlt=pd.crosstab(train['no_of_loans'],train['loan_default'])
nlt.plot(kind="bar",stacked=True,rot=0)
plt.show()
nlct=pd.crosstab(train['no_of_curr_loans'],train['loan_default'])
nlct.plot(kind="bar",stacked=True,rot=0)
plt.show()
```


asset_cost
1e6
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
age


asset_cost
1e6
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
1.0
2.0
education


loan_default
0
1
350
300
250
200
150
100
50
0
age

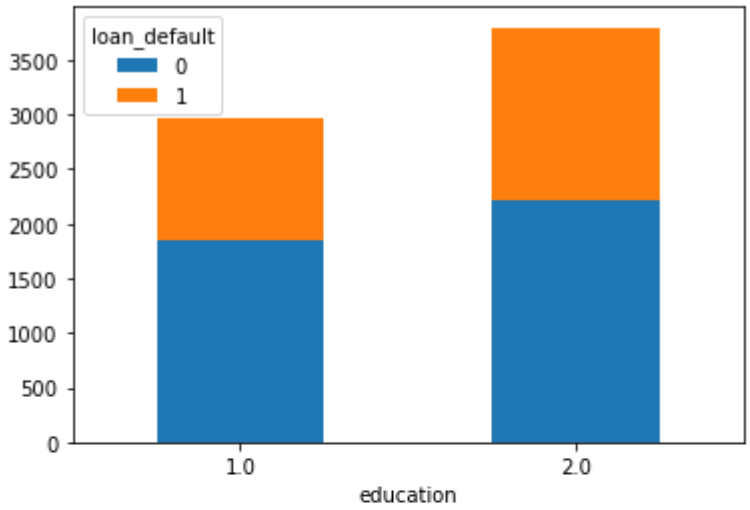

loan_default
0
1
3500
3000
2500
2000
1500
1000
500
0
no_of_loans

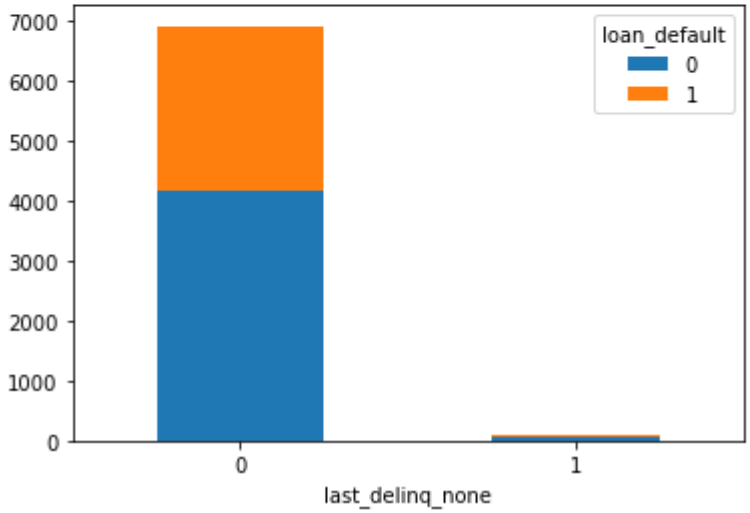

loan_default
0
1
4000
3500
3000
2500
2000
1500
1000
500
0
0 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 20 27 33
no_of_curr_loans

In [14]:

Slide Type Sub-Slide

```
et=pd.crosstab(train['education'],train['loan_default'])
et.plot(kind="bar",stacked=True,rot=0)
plt.show()
lt=pd.crosstab(train['last_delinq_none'],train['loan_default'])
lt.plot(kind="bar",stacked=True,rot=0)
plt.show()
```

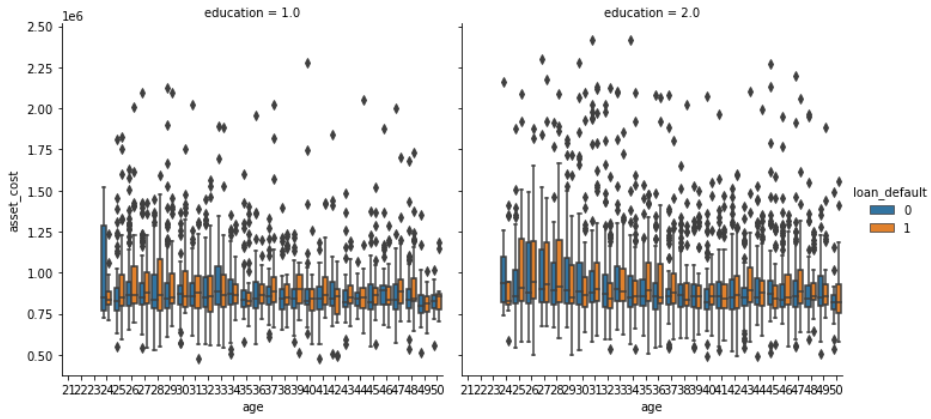




In [15]:

```
#Variable - 'Asset_cost' Vs 'Loan_Default'
sns.catplot(x='age',y='asset_cost',data=train,kind='box',hue='loan_default', col='education')
```

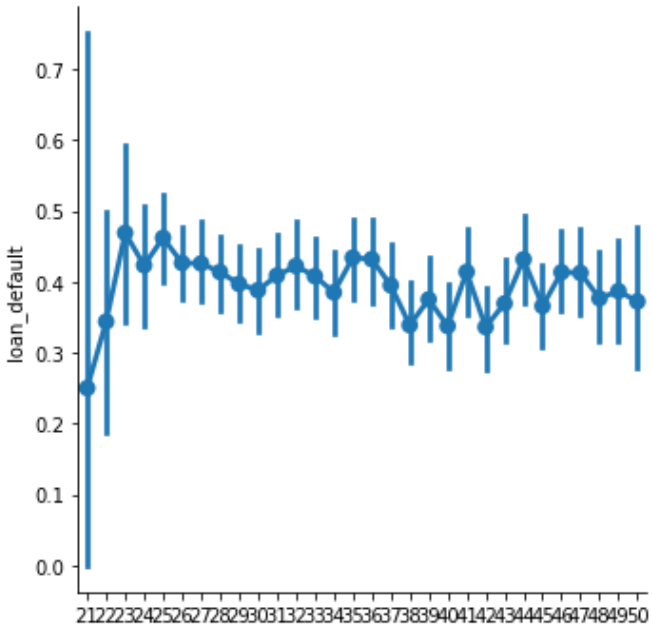
Out[15]: <seaborn.axisgrid.FacetGrid at 0x28330932d30>



In [16]:

```
#Catplot - 'Education' Vs 'Loan_Default'
sns.catplot(x='age',y='loan_default',kind='point',data=train)
```

Out[16]: <seaborn.axisgrid.FacetGrid at 0x28330aa9820>



I think using the crosstab function was very helpful as it provided valuable insights which are self-explanatory

5)Impute Missing values

We will fill the missing values in the education column with count value in both train and test datasets

In [18]:
```
#filling missing values
train['education'].fillna(train['education'].value_counts().index[0], inplace=True)
test['education'].fillna(test['education'].value_counts().index[0], inplace=True)
```

In [19]:
```
train.isnull().sum()
```

Out[19]:
```
loan_id            0
age                0
education          0
proof_submitted    0
loan_amount        0
asset_cost         0
no_of_loans        0
no_of_curr_loans   0
last_delinq_none   0
loan_default       0
dtype: int64
```

In [20]:
```
test.isnull().sum()
```

Out[20]:
```
loan_id            0
age                0
education          0
proof_submitted    0
loan_amount        0
asset_cost         0
no_of_loans        0
no_of_curr_loans   0
last_delinq_none   0
dtype: int64
```

6) Feature Engineering

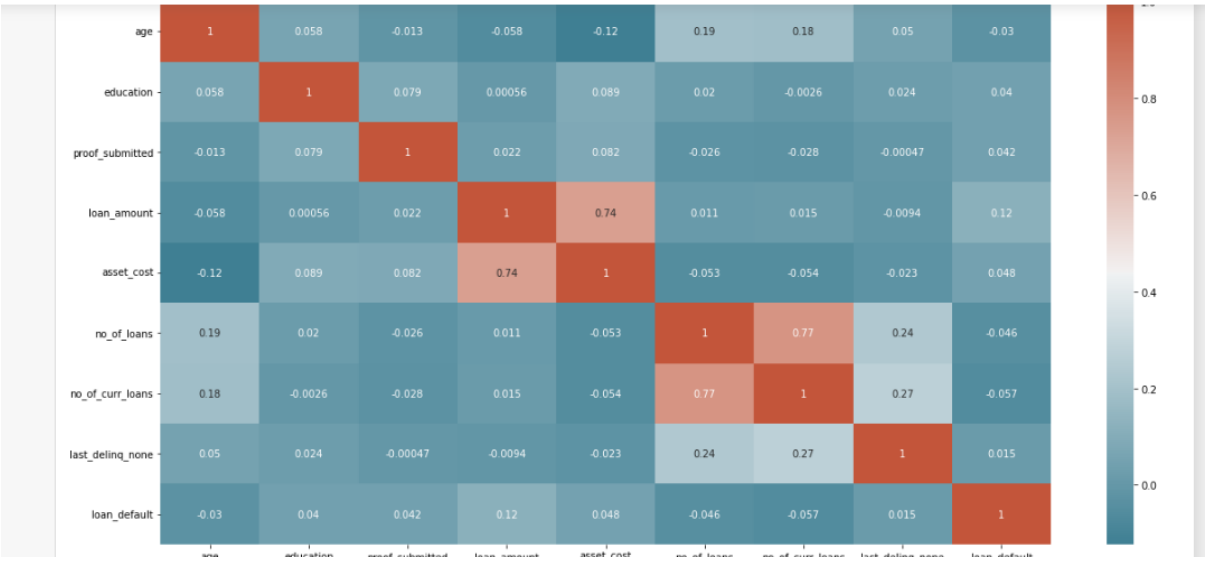
I have done Label Encoding, splitting test(40:60) & train datasets, dropping unwanted columns and plotting a correlation plot. Additionally, I have prepared the submission file and copied the loan id column into it. We dropped the loan id column because it won't add any valuable insights to data

In [22]:
```
train["proof_submitted"] = train["proof_submitted"].map({"Aadhar": 0, "VoterID": 1,"PAN": 2, "Driving": 3, "passport":4})
test["proof_submitted"] = test["proof_submitted"].map({"Aadhar": 0, "VoterID": 1,"PAN": 2, "Driving": 3, "passport":4})
```

In [23]:
```
submission = pd.DataFrame()
submission['loan_id'] = test['loan_id']
```

In [24]:
```
train.drop(['loan_id'],axis=1,inplace=True)
test.drop(['loan_id'],axis=1,inplace=True)
```

In [25]:
```
plt.figure(figsize=(20,10))
corr = train.corr()
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, annot=True, cmap=sns.diverging_palette(220, 20, as_cmap=Tru
```



Train & test datasets:

In [29]:

Slide Type Sub-Slide

```
train_temp,test_temp=train_test_split(train,test_size=0.20,random_state=0)
print("Length of Original Data:-",len(train),"\nLength of Train Data:-",len(train_temp),"\nLength of Test Data:-",len(test_temp))

features=list(train.columns)
label='loan_default'

features.remove(label)

print("-----------------------------------------------------------------------------------------------------\n\n")
print("\nFEATURES ARE:-",features)

print("-----------------------------------------------------------------------------------------------------\n\n")
print("\nLABELS ARE:-",label)

print("-----------------------------------------------------------------------------------------------------\n\n")

# To check the data points in "Train" & "Test" datasets after split
X_train=train_temp[features]
y_train=train_temp[label]

X_test=test_temp[features]
y_test=test_temp[label]
```

```
Length of Original Data:- 7000
Length of Train Data:- 5600
Length of Test Data:- 1400
```

7: Build Machine Learning Model & Make a prediction on test dataset

In [31]:

Slide Type Sub-Slide

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import xgboost as xgb
from sklearn.ensemble import VotingClassifier
from catboost import CatBoostClassifier
```

I have imported the necessary classifiers from the sklearn library

Machine Learning Algorithm: I have tried multiple machine learning algorithms including Logistic regression & Random Forest classifier however, I got the best accuracy with "CATBOOST"

In [32]:

Slide Type Sub-Slide

```
catb = CatBoostClassifier(max_depth=8, n_estimators=3000)

catb.fit(X_train,y_train)

print('Accuracy of CatBoost Classifier on training set: {:.2f}'
     .format(catb.score(X_train, y_train)))

print('Accuracy of CatBoost Classifier on test set: {:.2f}'
     .format(catb.score(X_test, y_test)))

print("--------------------------------------------------------------------------------------------------------------------------------")

print('CatBoost Classification Report of the training data:\n\n',classification_report(y_test,catb.predict(X_test)),'\n')
```

```
2996:   learn: 0.4771400        total: 32.9s    remaining: 33ms
2997:   learn: 0.4771002        total: 33s      remaining: 22ms
2998:   learn: 0.4770208        total: 33s      remaining: 11ms
2999:   learn: 0.4769798        total: 33s      remaining: 0us
Accuracy of CatBoost Classifier on training set: 0.82
Accuracy of CatBoost Classifier on test set: 0.59
```

click to unscroll output; double click to hide

```
--------------------------------------------------------------------------------------------------------------------------------
CatBoost Classification Report of the training data:

              precision    recall  f1-score   support

           0       0.64      0.79      0.71       861
           1       0.46      0.28      0.35       539

    accuracy                           0.59      1400
   macro avg       0.55      0.54      0.53      1400
weighted avg       0.57      0.59      0.57      1400
```

F1 Score for loan default (0) is 0.71 and loan default(1) is 0.35. The overall macro score is 0.53

8: Prepare submission file with final prediction

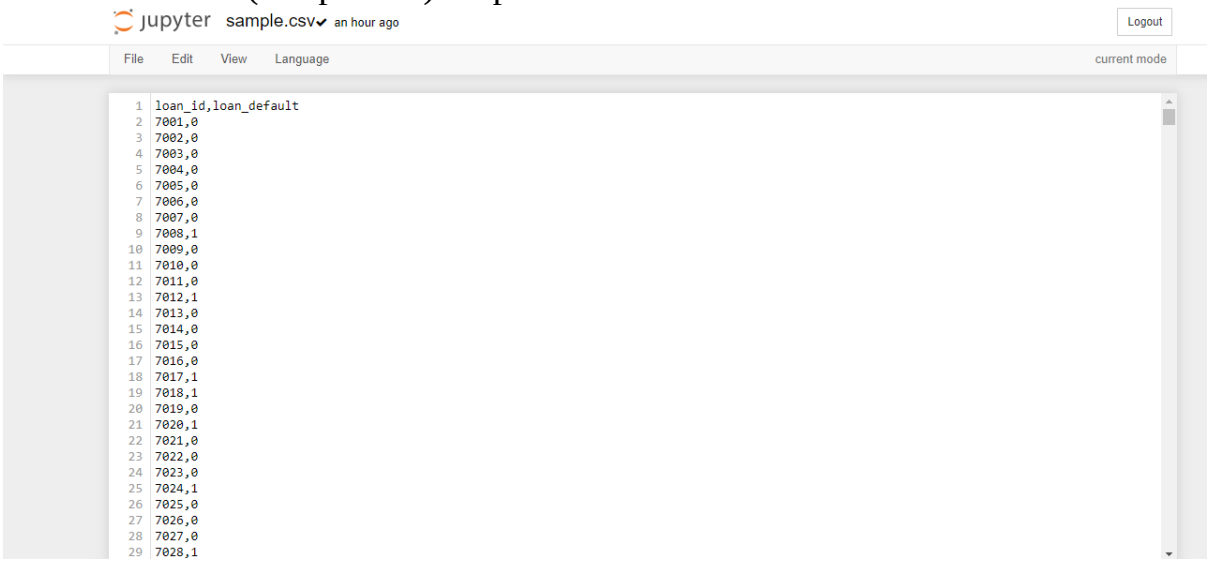
In [33]:
```
# Import Test data for the prediction of the Target Variable
x = np.array(test)

# Prediction using above Tuned Mode

y_pred = catb.predict(x)

print(y_pred)
```

```
[0 0 0 ... 0 0 0]
```

In [34]:
```
# Save in Dataframe
df1=pd.DataFrame(y_pred,columns=['loan_default'])

a=pd.Series(submission['loan_id'],name='loan_id')

final_pred = pd.concat([a,df1], axis=1)

final_pred.head()

final_pred.to_csv('sample.csv',index=False)


print("Process Completed")
```

```
Process Completed
```

Submission file (sample.csv) output:



Conclusion:

Using the Catboost classifier is best and model prediction results will be correct only if the data parameter with feature values contains all the features used in the model. Typically, the order of these features must match the order of the corresponding columns that are provided during the training. But if feature names are provided both during the training and when applying the model, they can be matched by names instead of columns order.