Imports

In [107]:

```
#Base
import pandas as pd
import numpy as np
import os
import datetime

# Data preparation
from sklearn.model_selection import train_test_split, cross_val_score, Stratifiec
from sklearn.preprocessing import OneHotEncoder, StandardScaler, LabelEncoder
from sklearn.decomposition import TruncatedSVD
```

Models

In [108]:

```
#Models
from sklearn.svm import SVC, LinearSVC
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
# from tensorflow import keras
# from tensorflow.keras import layers

# from keras.models import Sequential
# from keras.layers import Dense
# from sklearn.pipeline import Pipeline
# from scikeras.wrappers import KerasClassifier
```

Read Data

In [ ]:

```

```

In [109]:

```
data = pd.read_csv("train.csv")
print(f"The shape of Training data is {data.shape}")

target= data.iloc[:,-1]
data=data.iloc[:,:-1]
# target = data[target_col]
# data = data.drop(target_col, axis=1)
data.head()
```

```
The shape of Training data is (233154, 41)
```

Out[109]:

	UniqueID	disbursed_amount	asset_cost	ltv	branch_id	supplier_id	manufacturer_id	Curren
0	420825	50578	58400	89.55	67	22807	45	
1	537409	47145	65550	73.23	67	22807	45	
2	417566	53278	61360	89.63	67	22807	45	
3	624493	57513	66113	88.48	67	22807	45	
4	539055	52378	60300	88.39	67	22807	45	

5 rows × 40 columns

EDA

In [110]:

```
data.columns[data.isna().any()].tolist()
```

Out[110]:

```
['Employment.Type']
```

In [111]:

```
cat = """
asset_cost
Current_pincode_ID
DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS
disbursed_amount
ltv
NEW.ACCTS.IN.LAST.SIX.MONTHS
NO.OF_INQUIRIES
PERFORM_CNS.SCORE
PRI.ACTIVE.ACCTS
PRI.CURRENT.BALANCE
PRI.DISBURSED.AMOUNT
PRI.NO.OF.ACCTS
PRI.OVERDUE.ACCTS
PRI.SANCTIONED.AMOUNT
PRIMARY.INSTAL.AMT
SEC.ACTIVE.ACCTS
SEC.CURRENT.BALANCE
SEC.DISBURSED.AMOUNT
SEC.INSTAL.AMT
SEC.NO.OF.ACCTS
SEC.OVERDUE.ACCTS
SEC.SANCTIONED.AMOUNT
"""

print("','".join(cat.split("\n")))
```

```
','asset_cost','Current_pincode_ID','DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS','disb
ursed_amount','ltv','NEW.ACCTS.IN.LAST.SIX.MONTHS','NO.OF_INQUIRIES','PERFORM_C
NS.SCORE','PRI.ACTIVE.ACCTS','PRI.CURRENT.BALANCE','PRI.DISBURSED.AMOUNT','PRI.
NO.OF.ACCTS','PRI.OVERDUE.ACCTS','PRI.SANCTIONED.AMOUNT','PRIMARY.INSTAL.AM
T','SEC.ACTIVE.ACCTS','SEC.CURRENT.BALANCE','SEC.DISBURSED.AMOUNT','SEC.INSTAL.
AMT','SEC.NO.OF.ACCTS','SEC.OVERDUE.ACCTS','SEC.SANCTIONED.AMOUNT','
```

Removing columns

In [112]:

```
remove_cols = ['branch_id','Current_pincode_ID','Employee_code_ID','UniqueID',"su

data = data.drop(remove_cols, axis=1)

print("Shape of data after removing non-necessary columns: ", data.shape)
```

```
Shape of data after removing non-necessary columns:  (233154, 33)
```

Feature Extraction & Cleaning

```
In [113]: def get_age_from_dob(row):
            dob = row['Date.of.Birth']
            disb = row['DisbursalDate']

            dob_year = int(dob.split("-")[2])
            if dob_year < 49:
              dob_year = 2000 + dob_year
            else:
              dob_year = 1900 + dob_year

            disb_year = 2000 + int(disb.split("-")[2])

            age_at_disbursement = disb_year - dob_year
            return age_at_disbursement


          def get_avg_acc_age_in_months(row):
            avg_age = row['AVERAGE.ACCT.AGE']
            yrs, mth = avg_age.split(" ")[0], avg_age.split(" ")[1]
            yrs = int(yrs.replace("yrs",""))
            mth = int(mth.replace("mon",""))

            total_months = yrs*12 + mth
            return total_months

          def get_credit_age_in_months(row):
            avg_age = row['CREDIT.HISTORY.LENGTH']
            yrs, mth = avg_age.split(" ")[0], avg_age.split(" ")[1]
            yrs = int(yrs.replace("yrs",""))
            mth = int(mth.replace("mon",""))

            total_months = yrs*12 + mth
            return total_months

          def clean_employment_type(row):
            if row['Employment.Type'] not in ['Self employed', 'Salaried']:
              return 'NOT Defined'
            else:
              return row['Employment.Type']

            yrs, mth = avg_age.split(" ")[0], avg_age.split(" ")[1]
            yrs = int(yrs.replace("yrs",""))
            mth = int(mth.replace("mon",""))

            total_months = yrs*12 + mth
            return total_months

          data["age_at_disbursement"] = data.apply (lambda row: get_age_from_dob(row), axis
          data["AVERAGE.ACCT.AGE_Months"] = data.apply(lambda row: get_avg_acc_age_in_month
          data["credit_history_in_months"] = data.apply (lambda row: get_credit_age_in_mont
          data["Employment.Type"] = data.apply (lambda row: clean_employment_type(row), axi

          data = data.drop(['DisbursalDate','Date.of.Birth','AVERAGE.ACCT.AGE','CREDIT.HIST

          print("Shape of data after feature extraction: ", data.shape)
          data.head()
```

```
Shape of data after feature extraction:  (233154, 32)
```

Out[113]:

	disbursed_amount	asset_cost	ltv	Employment.Type	MobileNo_Avl_Flag	Aadhar_flag	PAN_f
0	50578	58400	89.55	Salaried	1	1	
1	47145	65550	73.23	Self employed	1	1	
2	53278	61360	89.63	Self employed	1	1	
3	57513	66113	88.48	Self employed	1	1	
4	52378	60300	88.39	Self employed	1	1	

5 rows × 32 columns

In [114]:

```
#print(data.tail())
```

One Hot Encoding

In [115]:

```
categorical_cols = ['Aadhar_flag','Driving_flag','Employment.Type',
                    'MobileNo_Avl_Flag','PAN_flag','Passport_flag',
                    'PERFORM_CNS.SCORE.DESCRIPTION','VoterID_flag'
]
for i in categorical_cols:
    cols = pd.get_dummies(data[i], prefix=i, drop_first=False)
    data = pd.concat([data, cols], axis=1)

data = data.drop(categorical_cols, axis=1)
print("Shape of data after one-hot encoding: ", data.shape)
data.head()
```

```
Shape of data after one-hot encoding:  (233154, 58)
```

Out[115]:

	disbursed_amount	asset_cost	ltv	PERFORM_CNS.SCORE	PRI.NO.OF.ACCTS	PRI.ACTIVE.AC
0	50578	58400	89.55	0	0	
1	47145	65550	73.23	598	1	
2	53278	61360	89.63	0	0	
3	57513	66113	88.48	305	3	
4	52378	60300	88.39	0	0	

5 rows × 58 columns

Scaling

In [99]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scale_cols = ['asset_cost','DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS','disbursed_amour
              'NEW.ACCTS.IN.LAST.SIX.MONTHS','NO.OF_INQUIRIES','PERFORM_CNS.SCORE
              'PRI.DISBURSED.AMOUNT','PRI.NO.OF.ACCTS','PRI.OVERDUE.ACCTS','PRI.S
              'SEC.ACTIVE.ACCTS','SEC.CURRENT.BALANCE',
              'SEC.DISBURSED.AMOUNT','SEC.INSTAL.AMT','SEC.NO.OF.ACCTS','SEC.OVEF
# scaler = StandardScaler()
col_list = data.columns.values.tolist()
print(col_list)
data[scale_cols] = scaler.fit_transform(data[scale_cols])
data_n = scaler.fit_transform(data)
data_n = pd.DataFrame(data_n, columns = col_list)
data = data_n
```

```
['disbursed_amount', 'asset_cost', 'ltv', 'PERFORM_CNS.SCORE', 'PRI.NO.OF.ACCT
S', 'PRI.ACTIVE.ACCTS', 'PRI.OVERDUE.ACCTS', 'PRI.CURRENT.BALANCE', 'PRI.SANCTI
ONED.AMOUNT', 'PRI.DISBURSED.AMOUNT', 'SEC.NO.OF.ACCTS', 'SEC.ACTIVE.ACCTS', 'S
EC.OVERDUE.ACCTS', 'SEC.CURRENT.BALANCE', 'SEC.SANCTIONED.AMOUNT', 'SEC.DISBURS
ED.AMOUNT', 'PRIMARY.INSTAL.AMT', 'SEC.INSTAL.AMT', 'NEW.ACCTS.IN.LAST.SIX.MONT
HS', 'DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS', 'NO.OF_INQUIRIES', 'age_at_disburse
ment', 'AVERAGE.ACCT.AGE_Months', 'credit_history_in_months', 'Aadhar_flag_0',
'Aadhar_flag_1', 'Driving_flag_0', 'Driving_flag_1', 'Employment.Type_NOT Defin
ed', 'Employment.Type_Salaried', 'Employment.Type_Self employed', 'MobileNo_Avl
_Flag_1', 'PAN_flag_0', 'PAN_flag_1', 'Passport_flag_0', 'Passport_flag_1', 'PE
RFORM_CNS.SCORE.DESCRIPTION_A-Very Low Risk', 'PERFORM_CNS.SCORE.DESCRIPTION_B-
Very Low Risk', 'PERFORM_CNS.SCORE.DESCRIPTION_C-Very Low Risk', 'PERFORM_CNS.S
CORE.DESCRIPTION_D-Very Low Risk', 'PERFORM_CNS.SCORE.DESCRIPTION_E-Low Risk',
'PERFORM_CNS.SCORE.DESCRIPTION_F-Low Risk', 'PERFORM_CNS.SCORE.DESCRIPTION_G-Lo
w Risk', 'PERFORM_CNS.SCORE.DESCRIPTION_H-Medium Risk', 'PERFORM_CNS.SCORE.DESC
RIPTION_I-Medium Risk', 'PERFORM_CNS.SCORE.DESCRIPTION_J-High Risk', 'PERFORM_C
NS.SCORE.DESCRIPTION_K-High Risk', 'PERFORM_CNS.SCORE.DESCRIPTION_L-Very High R
isk', 'PERFORM_CNS.SCORE.DESCRIPTION_M-Very High Risk', 'PERFORM_CNS.SCORE.DESC
RIPTION_No Bureau History Available', 'PERFORM_CNS.SCORE.DESCRIPTION_Not Score
d: More than 50 active Accounts found', 'PERFORM_CNS.SCORE.DESCRIPTION_Not Scor
ed: No Activity seen on the customer (Inactive)', 'PERFORM_CNS.SCORE.DESCRIPTIO
N_Not Scored: No Updates available in last 36 months', 'PERFORM_CNS.SCORE.DESCR
IPTION_Not Scored: Not Enough Info available on the customer', 'PERFORM_CNS.SCO
RE.DESCRIPTION_Not Scored: Only a Guarantor', 'PERFORM_CNS.SCORE.DESCRIPTION_No
t Scored: Sufficient History Not Available', 'VoterID_flag_0', 'VoterID_flag_
1']
```

In [100]:

```
# data_n.head()
```

Splitting Data

In [116]:

```
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2,

print("Shape of data:", data.shape)
print("Shape of target", target.shape)
print("Shape of Xtrain and ytrain:", X_train.shape, y_train.shape)
print("Shape of Xtest and ytest:", X_test.shape, y_test.shape)
```

```
Shape of data: (233154, 58)
Shape of target (233154,)
Shape of Xtrain and ytrain: (186523, 58) (186523,)
Shape of Xtest and ytest: (46631, 58) (46631,)
```

In [117]:

```
X_train.head()
```

Out[117]:

	disbursed_amount	asset_cost	ltv	PERFORM_CNS.SCORE	PRI.NO.OF.ACCTS	PRI.ACTI
192380	57159	67730	88.59	680	5	
43291	46645	76706	65.18	0	0	
210644	30484	74642	41.53	0	0	
50259	62447	92130	70.01	749	3	
158140	54515	64500	87.60	300	1	

5 rows × 58 columns

XGBoost

In [118]:

```
xgb = XGBClassifier()
xgb.fit(X_train, y_train)

xgb_y_pred = xgb.predict(X_test)

cnf_matrix = metrics.confusion_matrix(y_test, xgb_y_pred)
print(cnf_matrix)


print("Accuracy:",metrics.accuracy_score(y_test, xgb_y_pred))
print("Precision:",metrics.precision_score(y_test, xgb_y_pred))
print("Recall:",metrics.recall_score(y_test, xgb_y_pred))
```

```
C:\Users\Checkout\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarn
ing: The use of label encoder in XGBClassifier is deprecated and will be remove
d in a future release. To remove this warning, do the following: 1) Pass option
use_label_encoder=False when constructing XGBClassifier object; and 2) Encode y
our labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[14:02:23] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
0/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
[[36258   251]
 [ 9924   198]]
Accuracy: 0.7817975166734575
Precision: 0.44097995545657015
Recall: 0.01956135151155898
```

In [119]:

```
from sklearn.ensemble import RandomForestClassifier
model=RandomForestClassifier(n_estimators=1200,max_depth=7)
model.fit(X_train,y_train)
ypred=model.predict(X_test)
```

In [120]:

```
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, ypred)
print("Accuracy:",metrics.accuracy_score(y_test, ypred))
```

```
Accuracy: 0.7829340996332912
```

In [121]:

```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=51)
neigh.fit(X_train,y_train)
ypred=neigh.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, ypred)
print("Accuracy:",metrics.accuracy_score(y_test, ypred))
```

```
Accuracy: 0.7828054298642534
```

In [122]:

```
print(X_train.shape,X_test.shape)
```

```
(186523, 58) (46631, 58)
```

In [123]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
```

In [124]:

```
ann=Sequential()
nodes=20
ann.add(Dense(units=nodes, activation='relu'))
ann.add(Dense(units=nodes, activation='relu'))
ann.add(Dense(units=nodes, activation='relu'))
ann.add(Dense(units=nodes, activation='relu'))
ann.add(Dense(units=nodes, activation='relu'))
ann.add(Dense(units=nodes, activation='relu'))
ann.add(Dense(units=nodes, activation='relu'))
ann.add(Dense(units=nodes, activation='relu'))
ann.add(Dense(units=nodes, activation='relu'))
ann.add(Dense(units=nodes, activation='relu'))
ann.add(Dense(units=nodes, activation='relu'))
ann.add(Dense(units=nodes, activation='relu'))
ann.add(Dense(units=nodes, activation='relu'))
ann.add(Dense(units=nodes, activation='relu'))

ann.add(Dense(units=1, activation='sigmoid'))
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accura
ann.fit(X_train, y_train, batch_size = 500, epochs = 1000)
```

```
Epoch 638/1000
374/374 [==============================] - 1s 3ms/step - loss: 0.5232 - accur
acy: 0.7829
Epoch 639/1000
374/374 [==============================] - 1s 3ms/step - loss: 0.5232 - accur
acy: 0.7829: 0s - l
Epoch 640/1000
374/374 [==============================] - 1s 3ms/step - loss: 0.5232 - accur
acy: 0.7829: 0s
Epoch 641/1000
374/374 [==============================] - 1s 3ms/step - loss: 0.5232 - accur
acy: 0.7829
Epoch 642/1000
374/374 [==============================] - 1s 3ms/step - loss: 0.5232 - accur
acy: 0.7829: 0s - loss: 0.5232 - accuracy: 0.
Epoch 643/1000
374/374 [==============================] - 1s 3ms/step - loss: 0.5232 - accur
acy: 0.7829
Epoch 644/1000
374/374 [==============================] - 1s 3ms/step - loss: 0.5232 - accur
```

In [125]:

```
ypred=ann.predict(X_test)
```

In [126]:

```
print(ypred>0.5)
ypred = [1 if i[0]==True else 0 for i in ypred>0.5]
```

```
[[False]
 [False]
 [False]
 ...
 [False]
 [False]
 [False]]
```

In [129]:

```
cnf_matrix = metrics.confusion_matrix(y_test, ypred)
print("accuracy",metrics.accuracy_score(y_test, ypred))
```

```
accuracy 0.7829340996332912
```

In [ ]: