Imports

```
#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, StratifiedKFold
from sklearn.preprocessing import OneHotEncoder, StandardScaler, LabelEncoder
from sklearn.decomposition import TruncatedSVD
```

Models

```
#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 sklearn.ensemble import RandomForestClassifier
```

Read Data

```
target_col = ["loan_default"]
```

```
data = pd.read_csv("train.csv")
print(f"The shape of Training data is {data.shape}")
# target = data[target_col]
# data = data.drop(target_col, axis=1)
data.head()
```

EDA

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

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

Removing columns

```
remove_cols = ['branch_id','Current_pincode_ID','Employee_code_ID','UniqueID'
               ,'State_ID', 'supplier_id', 'MobileNo_Avl_Flag'
               ]

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

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

```
Shape of data after removing non-necessary columns:  (233154, 34)
```

	disbursed_amount	asset_cost	ltv	manufacturer_id	Date.of.Birth	Employment.Type
0	50578	58400	89.55	45	01-01-84	Salaried
1	47145	65550	73.23	45	31-07-85	Self employed
2	53278	61360	89.63	45	24-08-85	Self employed
3	57513	66113	88.48	45	30-12-93	Self employed
4	52378	60300	88.39	45	09-12-77	Self employed

Feature Extraction & Cleaning

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

  dob_year = int(dob.split("-")[2])
  if dob_year < 19:
    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'] == 'Self employed':
    return 0
  elif row['Employment.Type'] == 'Salaried':
    return 1
  else:
    return 2

def clean_credit_risk(row):
  risk_category = {'unknown':-1, 'A':13, 'B':12, 'C':11,'D':10,'E':9,'F':8,'G':7,'H':6,'I':5,
  value = row['PERFORM_CNS.SCORE.DESCRIPTION'].split("-")
  if len(value) == 1:
    # No Bureau History Available
    return -1
  else:
    rc = risk_category[value[0]]
    return rc


data["age_at_disbursement"] = data.apply (lambda row: get_age_from_dob(row), axis=1)
data["AVERAGE.ACCT.AGE_Months"] = data.apply(lambda row: get_avg_acc_age_in_months(row), axis
data["credit_history_in_months"] = data.apply (lambda row: get_credit_age_in_months(row), axi
data["Employment.Type"] = data.apply (lambda row: clean_employment_type(row), axis=1)
data["credit_risk_cleaned"] = data.apply(lambda row: clean_credit_risk(row), axis=1)

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

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

```
Shape of data after feature extraction:  (233154, 33)
   disbursed_amount  asset_cost    ltv  manufacturer_id  Employment.Type  Aadhar_flag  F

0             50578       58400  89.55               45                1            1
1             47145       65550  73.23               45                0            1
2             53278       61360  89.63               45                0            1
3             57513       66113  88.48               45                0            1
4             52378       60300  88.39               45                0            1
```

Combining Primary and Secondary Accounts

```
data['all_accounts'] = data['PRI.NO.OF.ACCTS'] + data['SEC.NO.OF.ACCTS']
data['primary_inactive_accounts'] = data['PRI.NO.OF.ACCTS'] - data['PRI.ACTIVE.ACCTS']
data['secondary_innactive_accounts'] = data['SEC.NO.OF.ACCTS'] - data['SEC.ACTIVE.ACCTS']
data['total_inactive_accounts'] = data['primary_inactive_accounts'] + data['secondary_innacti
data['total_overdue_Accounts'] = data['PRI.OVERDUE.ACCTS'] + data['SEC.OVERDUE.ACCTS']
data['total_balance'] = data['PRI.CURRENT.BALANCE'] + data['SEC.CURRENT.BALANCE']
data['total_sanctioned_amount'] = data['PRI.SANCTIONED.AMOUNT'] + data['SEC.SANCTIONED.AMOUNT
data['total_disbursed_amount'] = data['PRI.DISBURSED.AMOUNT'] + data['SEC.DISBURSED.AMOUNT']
data['total_installment'] = data['PRIMARY.INSTAL.AMT'] + data['SEC.INSTAL.AMT']

data=data.drop(['PRI.NO.OF.ACCTS','SEC.NO.OF.ACCTS','PRI.CURRENT.BALANCE',
                'primary_inactive_accounts','secondary_innactive_accounts',
            'PRI.SANCTIONED.AMOUNT','SEC.NO.OF.ACCTS','PRI.NO.OF.ACCTS',
            'PRI.DISBURSED.AMOUNT','PRI.ACTIVE.ACCTS', 'PRI.OVERDUE.ACCTS',
            'SEC.CURRENT.BALANCE','SEC.SANCTIONED.AMOUNT', 'SEC.OVERDUE.ACCTS',
            'SEC.DISBURSED.AMOUNT','PRIMARY.INSTAL.AMT','SEC.INSTAL.AMT',
            'disbursed_amount','SEC.ACTIVE.ACCTS'],axis=1)
```

Scaling all numerical columns

```
numerical_cols = ['asset_cost', 'ltv','PERFORM_CNS.SCORE',
       'NEW.ACCTS.IN.LAST.SIX.MONTHS', 'DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS',
       'AVERAGE.ACCT.AGE_Months', 'credit_history_in_months', 'NO.OF_INQUIRIES','all_accounts
       'total_overdue_Accounts', 'total_balance', 'total_sanctioned_amount',
       'total_disbursed_amount', 'total_installment','age_at_disbursement']

from sklearn.preprocessing import  RobustScaler
rob_scaler = RobustScaler()

data[numerical_cols] = rob_scaler.fit_transform(data[numerical_cols])
data.head()
```

	asset_cost	ltv	manufacturer_id	Employment.Type	Aadhar_flag	PAN_flag	Voter
0	-0.930384	0.862069	45	1	1	0	
1	-0.400156	-0.241379	45	0	1	0	
2	-0.710877	0.867478	45	0	1	0	
3	-0.358405	0.789723	45	0	1	0	
4	-0.789484	0.783638	45	0	1	0	

Adding a new column to the dataset for those records that have zeros in important feature columns

```
data['NA_Feature'] = (data == 0).astype(int).sum(axis=1)
data.head()
```

	asset_cost	ltv	manufacturer_id	Employment.Type	Aadhar_flag	PAN_flag	Voter
0	-0.930384	0.862069	45	1	1	0	
1	-0.400156	-0.241379	45	0	1	0	
2	-0.710877	0.867478	45	0	1	0	
3	-0.358405	0.789723	45	0	1	0	
4	-0.789484	0.783638	45	0	1	0	

Dimensionality Reduction

```
# Not required as number of features is low
```

Splitting Data

```
from sklearn.model_selection import train_test_split,KFold,cross_val_score

target = data.loan_default
X = data.drop("loan_default",axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, target, test_size=0.33, random_state=1
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, 26)
Shape of target (233154,)
Shape of Xtrain and ytrain: (156213, 25) (156213,)
Shape of Xtest and ytest: (76941, 25) (76941,)
```

XGBoost Classifier

```
from sklearn import metrics
from sklearn.metrics import f1_score

xgb = XGBClassifier()
xgb.fit(X_train, y_train.values.flatten())
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))
print("Weighted f1 score:", f1_score(y_test, xgb_y_pred, average='weighted'))
```

```
     Accuracy: 0.9006121573673335
     Precision: 0.9493721347418776
     Recall: 0.5716171617161716
     Weighted f1 score: 0.890861022785194
```

Random Forest Classifier

```
forest = RandomForestClassifier(criterion='gini',
                                n_estimators=5,
                                random_state=1,
                                n_jobs=2)
forest.fit(X_train, y_train)
forest_y_pred = forest.predict(X_test)
print('Accuracy: %.3f' % accuracy_score(y_test, forest_y_pred))
print("Precision:",metrics.precision_score(y_test, forest_y_pred))
print("Recall:",metrics.recall_score(y_test, forest_y_pred))
print("Weighted f1 score:", f1_score(y_test, forest_y_pred, average='weighted'))
```

```
     Accuracy: 0.910
     Precision: 0.8553568832679168
     Recall: 0.7061506150615061
     Weighted f1 score: 0.9072672029751047
```

Neural Network Classifier

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000)
mlp.fit(X_train, y_train.values.ravel())
mlp_y_pred = mlp.predict(X_test)
print('Accuracy: %.3f' % accuracy_score(y_test, mlp_y_pred))
print("Precision:",metrics.precision_score(y_test, mlp_y_pred))
print("Recall:",metrics.recall_score(y_test, mlp_y_pred))
print("Weighted f1 score:", f1_score(y_test, mlp_y_pred, average='weighted'))
```

```
     Accuracy: 0.861
     Precision: 0.6959079618250752
     Recall: 0.6388238823882388
     Weighted f1 score: 0.859121064504987
```

KNN Classifier

```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X_train, y_train.values.ravel())
knn_y_pred = neigh.predict(X_test)
print('Accuracy: %.3f' % accuracy_score(y_test, knn_y_pred))
print("Precision:",metrics.precision_score(y_test, knn_y_pred))
print("Recall:",metrics.recall_score(y_test, knn_y_pred))
print("Weighted f1 score:", f1_score(y_test, knn_y_pred, average='weighted'))
```

```
     Accuracy: 0.883
     Precision: 0.7760957324106112
     Recall: 0.6460246024602461
     Weighted f1 score: 0.8789350593325763
```