Home Credit Default Risk (HCDR)-Predicting Borrower Behavior-Phase3

This course project is based on a Kaggle competition. It tries to predict whether a borrower will pay or will not pay for his/her loan. The dataset of this project is from Home Credit company, a finacial institution in Asia and East Europe. In phase 2 of this project, we are doing feature engineering, experiments with ensemble methods, and hyperparameter tuning.

Dataset

Background on the dataset

Home Credit is a consumer finance provider which focuses mainly on providing financial needs to customers with little or no credit history. Founded in 1999, Home Credit has branches internationally and spread across over 9 countries. It offers financial needs to the customers such as point of sale loans, cash loans and loans for products through online and physical distribution network. Generally, customers start with POS loans and satisfied customers adopt themselves to entire business model.

Its main aim is to:

- Focus on mass-retail lending with innovative financial methods.
- Fulfill client's dreams and ambitions in a financially responsible manner.
- Provide long term and stable employment opportunities to the employers.
- Increase the standard of living through economic development supporting domestic consumption.

It offers retail banking services such as current and deposit accounts in countries where it has banking license. It focuses on customer retention by offering cross-selling opportunities across its product base.

Data files overview

application_{train|test}.csv: This is the crucial table which handles all static details about the applicants. Each row in this dataset conveys details about each loan applicant. This is furtherly classified into two different data files for training and testing. The training data includes our binary target variable (1 or 0) and the training data obviously doesn't contain the target variable.

bureau.csv: This data file contains information regarding the borrower's previous credit which was borrowed from other financial institutions and was reported to the Credit Bureau. It contains every detail of the number of credits the client previously had in the Credit Bureau before applying to loan at the Home Credit Group.

bureau_balance.csv: This dataset has information of borrower's monthly credits in the previous credits. Every row in the data represents the history of previous credit reported to the Credit Bureau for every month.

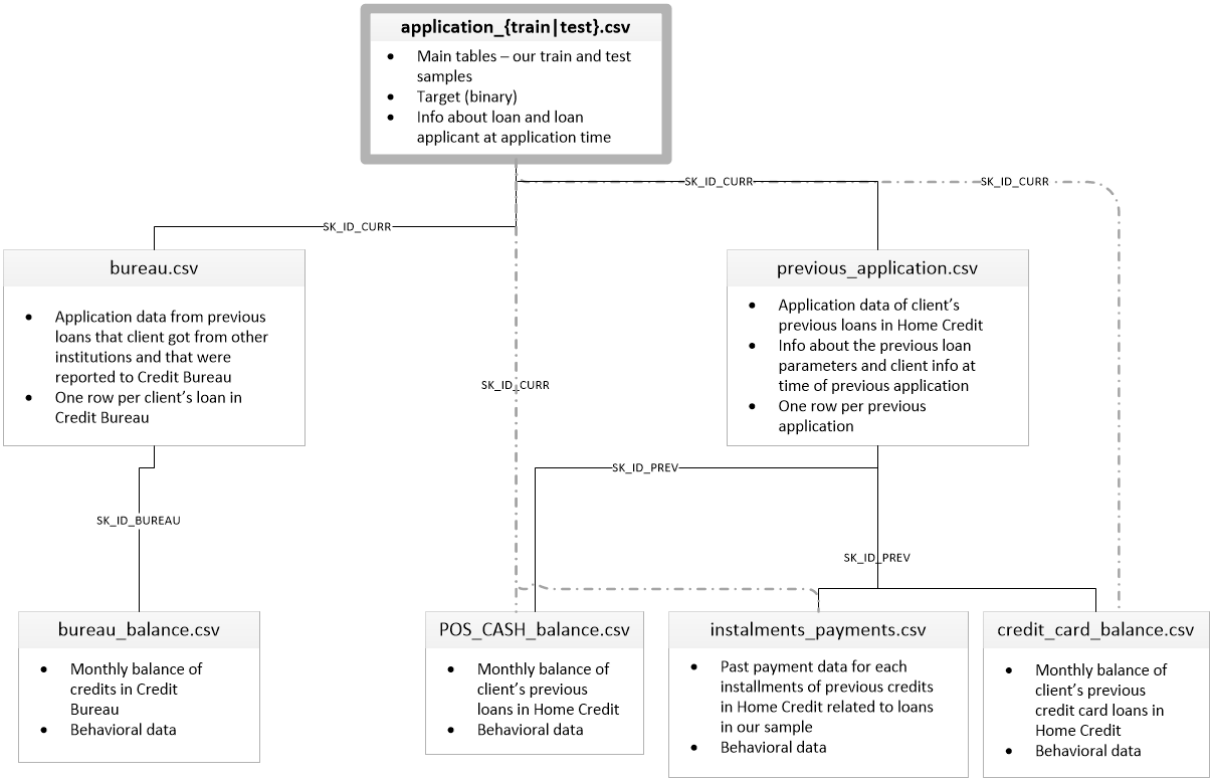
POS_CASH_balance.csv: This data file contains monthly balance snapshots of cash loans that borrower had with Home Credit and previous point of sales. Each row has each month of history of every previous credit in Home Credit.

credit_card_balance.csv: This data file contains the snapshots of the history of credit cards that the borrower has with the Home credit. Each row has each month of history of every previous credit card in Home Credit.

previous_application.csv: This data file has all the applications that the borrower had applied for in the past.

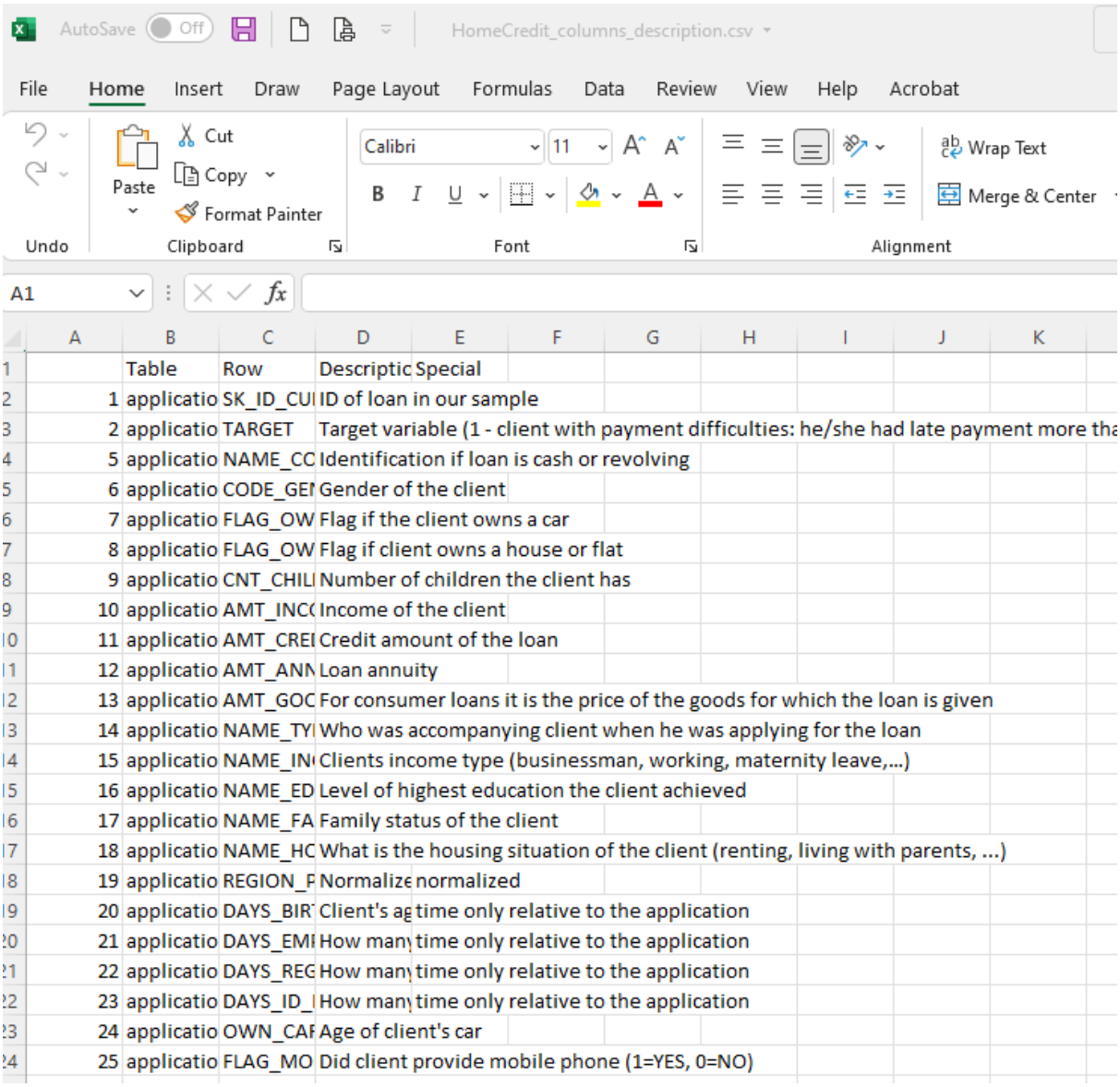
installments_payments.csv: This data file has the history of the repayment that has been done for the disbursed amount in Home Credit. It has one row for every repayment that has been done and also the missed ones.

HomeCredit_columns_description.csv: This data file has the information about the columns in the other data files given.



Data Dictionary

`HomeCredit_columns_description.csv` is a data dictionary of this dataset



	A	B	C	D	E	F	G	H	I	J	K
1		Table	Row	Descriptic	Special						
2	1	applicatio	SK_ID_CU	ID of loan in our sample							
3	2	applicatio	TARGET	Target variable (1 - client with payment difficulties: he/she had late payment more tha							
4	5	applicatio	NAME_CC	Identification if loan is cash or revolving							
5	6	applicatio	CODE_GEI	Gender of the client							
6	7	applicatio	FLAG_OW	Flag if the client owns a car							
7	8	applicatio	FLAG_OW	Flag if client owns a house or flat							
8	9	applicatio	CNT_CHIL	Number of children the client has							
9	10	applicatio	AMT_INC	Income of the client							
10	11	applicatio	AMT_CRE	Credit amount of the loan							
11	12	applicatio	AMT_ANN	Loan annuity							
12	13	applicatio	AMT_GOC	For consumer loans it is the price of the goods for which the loan is given							
13	14	applicatio	NAME_TY	Who was accompanying client when he was applying for the loan							
14	15	applicatio	NAME_IN	Clients income type (businessman, working, maternity leave,...)							
15	16	applicatio	NAME_ED	Level of highest education the client achieved							
16	17	applicatio	NAME_FA	Family status of the client							
17	18	applicatio	NAME_HC	What is the housing situation of the client (renting, living with parents, ...)							
18	19	applicatio	REGION_F	Normalize	normalized						
19	20	applicatio	DAYS_BIR	Client's ag	time only relative to the application						
20	21	applicatio	DAYS_EMI	How many	time only relative to the application						
21	22	applicatio	DAYS_REG	How many	time only relative to the application						
22	23	applicatio	DAYS_ID_	How many	time only relative to the application						
23	24	applicatio	OWN_CAF	Age of client's car							
24	25	applicatio	FLAG_MO	Did client provide mobile phone (1=YES, 0=NO)							

Application train

- **application_train:** the main training and testing data with information about each loan application at Home Credit. Every loan has its own row and is identified by the feature SK_ID_CURR. The training application data comes with the TARGET indicating **0: the loan was repaid** or **1: the loan was not repaid**. The target variable defines if the client had payment difficulties meaning he/she had late payment more than X days on at least one of the first Y installments of the loan. Such case is marked as 1 while other all other cases as 0.

In [140…

```
import pandas as pd
application_train = pd.read_csv('../input/application_train.csv')
print(app_train_data.shape)
app_train_data.describe()
```

```
(307511, 122)
```

Out[140]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANN
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.0(
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.57
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.73
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.5(
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.0(
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.0(
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.0(
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.5(

8 rows × 106 columns

Application test

- **application_test:** has the same features as application_train, except TARGET column. Kaggle does not provide the TARGET column in the test file. It is used for scoring.

In [141…

```
import pandas as pd
application_test = pd.read_csv('../input/application_test.csv')
print(app_test.shape)
app_test.describe()
```

```
(48744, 129)
```

Out[141]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GO(
count	48744.000000	48744.000000	4.874400e+04	4.874400e+04	48720.000000	4.8
mean	277796.676350	0.397054	1.784318e+05	5.167404e+05	29426.240209	4.6
std	103169.547296	0.709047	1.015226e+05	3.653970e+05	16016.368315	3.3
min	100001.000000	0.000000	2.694150e+04	4.500000e+04	2295.000000	4.5
25%	188557.750000	0.000000	1.125000e+05	2.606400e+05	17973.000000	2.2
50%	277549.000000	0.000000	1.575000e+05	4.500000e+05	26199.000000	3.9
75%	367555.500000	1.000000	2.250000e+05	6.750000e+05	37390.500000	6.3
max	456250.000000	20.000000	4.410000e+06	2.245500e+06	180576.000000	2.2

8 rows × 113 columns

Other tables

bureau.csv: This data file contains information regarding the borrower's previous credit which was borrowed from other financial institutions and was reported to the Credit Bureau. It contains

every detail of the number of credits the client previously had in the Credit Bureau before applying to loan at the Home Credit Group.

bureau_balance.csv: This dataset has information of borrower's monthly credits in the previous credits. Every row in the data represents the history of previous credit reported to the Credit Bureau for every month.

POS_CASH_balance.csv: This data file contains monthly balance snapshots of cash loans that borrower had with Home Credit and previous point of sales. Each row has each month of history of every previous credit in Home Credit.

credit_card_balance.csv: This data file contains the snapshots of the history of credit cards that the borrower has with the Home credit. Each row has each month of history of every previous credit card in Home Credit.

previous_application.csv: This data file has all the applications that the borrower had applied for in the past.

installments_payments.csv: This data file has the history of the repayment that has been done for the disbursed amount in Home Credit. It has one row for every repayment that has been done and also the missed ones.

Feature Engineering - New Features

In this section, we create 6 new features derived from tables: application_train, previous_application, and installments_payments.

Work history ('work_history'): days employed divided by days birth; implies how long an applicant's work history is as a ratio of their total age. The higher the ratio, the longer the applicant has been working in their life.

Inc_loan ratio ('inc_loan_ratio'): applicant's income divided by the granted loan amount. We use this to gauge if the applicant is borrowing too much – the smaller the ratio, the more the applicant is borrowing compared to their income.

Previous loans ('prev_loans'): a count of previous loans recorded by Home Credit bureau. We are curious to see the impact (positive or negative) to prediction using this feature.

Down payment percentage ('dwn_pymt_pct'): Average down payment on previous Home Credit loans divided by average amount of previous Home Credit Loans received. We use this feature to gauge if higher down payments on loan will impact if an applicant defaults or not.

'DAYS_LATE_PAYMENT': the total amount of late payment of each borrower on previous Home Credit loans from the installments_payment table. The higher this number, the more likely that the borrower get default.

'AMT_LATE_PAYMENT':count the number of late payment of each borrower on previous Home Credit loans from the installments_payment table. The higher this number, the more likely that

the borrower struggles with the repayment

With these new features, we created our new training and test set: application_train_joined and application_test_joined

In [37]:
```
application_train = pd.read_csv('../input/application_train.csv')
app_train_data = application_train
print(app_train_data.shape)
```

```
(307511, 122)
```

Reading Data from Previous Application File

In [39]:
```
prev_apps = pd.read_csv('../input/previous_application.csv')
```

In [ ]:
```
prev_apps.head()
```

In [ ]:
```
prev_apps.info()
```

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

In [40]:
```
## filter previous applications to NAME_CONTRACT_STATUS = 'Approved'

prev_apps = prev_apps[prev_apps['NAME_CONTRACT_STATUS']=='Approved']


## filter prev applications to NAME_CLIENT_TYPE != XNA
prev_apps = prev_apps[prev_apps['NAME_CLIENT_TYPE']!='XNA'] ## drops 355 rows

## remove records where loan amount = 0
prev_apps = prev_apps[prev_apps['AMT_CREDIT']!=0] ## drops 980 rows ## note: AMT_APPL

## impute blank down payment rows with 0
prev_apps['AMT_DOWN_PAYMENT'] = prev_apps['AMT_DOWN_PAYMENT'].fillna(value=0.0)
prev_apps["AMT_DOWN_PAYMENT"] = np.where(prev_apps["AMT_DOWN_PAYMENT"] < 0,
                                         0, prev_apps["AMT_DOWN_PAYMENT"])


#prev_apps[prev_apps['AMT_DOWN_PAYMENT'].isnull()].head()
prev_apps #108129
```

Out[40]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AM
0	1369693	100001	Consumer loans	3951.000	24835.5	
1	1038818	100002	Consumer loans	9251.775	179055.0	
2	1810518	100003	Cash loans	98356.995	900000.0	
3	2636178	100003	Consumer loans	64567.665	337500.0	
4	1564014	100004	Consumer loans	5357.250	24282.0	
...	...	...	...	...	...	
1048568	1792910	456254	Consumer loans	2296.440	18846.0	
1048569	1743609	456255	Consumer loans	11090.835	102037.5	
1048570	1359084	456255	Consumer loans	11314.170	82404.0	
1048571	1179690	456255	Cash loans	28873.170	450000.0	
1048572	2073384	456255	Cash loans	16400.610	229500.0	

651151 rows × 37 columns

In [41]:

```
##get average previous downpayment per current application
m = prev_apps.groupby('SK_ID_CURR')['AMT_DOWN_PAYMENT'].mean().to_frame()
sks = list(m.index.values) ##store indexes (SK_ID's) in a list
avg_dwn_pymt = list(m['AMT_DOWN_PAYMENT'].values)


##get count of previous applications per current application
m = prev_apps.groupby('SK_ID_CURR').size().to_frame()
m.rename(columns = {0:'Count'}, inplace = True) ## rename column to "Count"
prev_loans = list(m['Count'].values)


##get average amount of loans from previous applications per current application
m = prev_apps.groupby('SK_ID_CURR')['AMT_CREDIT'].mean().to_frame()
avg_loan_amt = list(m['AMT_CREDIT'].values)

## create new dataframe with features engineered from the previous_application datase
dict = {'SK_ID_prev_app': sks, 'avg_dwn_pymt': avg_dwn_pymt, 'prev_loans': prev_loans

prev_apps_features = pd.DataFrame(dict)

prev_apps_features['dwn_pymt_pct'] = prev_apps_features['avg_dwn_pymt'] / prev_apps_f

prev_apps_features['SK_ID_CURR'] = prev_apps_features['SK_ID_prev_app']
prev_apps_features = prev_apps_features.drop('SK_ID_prev_app', axis =1)

prev_apps_features
```

Out[41]:

	avg_dwn_pymt	prev_loans	prev_avg_loan_amt	dwn_pymt_pct	SK_ID_CURR
0	2520.0	1	23787.000	0.105940	100001
1	0.0	1	179055.000	0.000000	100002
2	0.0	2	692259.750	0.000000	100003
3	4860.0	1	20106.000	0.241719	100004
4	22329.0	3	404310.000	0.055227	100006
...	...	...	...	...	...
289962	0.0	1	40455.000	0.000000	456251
289963	3456.0	1	56821.500	0.060822	456252
289964	2893.5	1	27306.000	0.105966	456253
289965	0.0	2	134439.750	0.000000	456254
289966	2250.0	4	238008.375	0.009453	456255

289967 rows × 5 columns

In [ ]:
```
prev_apps_features.describe()
```

In [43]:
```
#Load application test data
app_test = pd.read_csv('../input/application_test.csv')
```

In [44]:
```
##standardize DAYS variables to year
app_train_data["DAYS_BIRTH"] = app_train_data["DAYS_BIRTH"]/-365
app_train_data["DAYS_EMPLOYED"] = app_train_data["DAYS_EMPLOYED"]/-365 ##55,374 entri
app_train_data["DAYS_EMPLOYED"] = np.where(app_train_data["DAYS_EMPLOYED"] == 365243/
                                           app_train_data["DAYS_EMPLOYED"].median(),

app_train_data["DAYS_REGISTRATION"] = app_train_data["DAYS_REGISTRATION"]/-365
app_train_data["DAYS_ID_PUBLISH"] = app_train_data["DAYS_ID_PUBLISH"]/-365
```

In [47]:
```
print(app_train_data["DAYS_ID_PUBLISH"])
```

```
0          5.808219
1          0.797260
2          6.934247
3          6.676712
4          9.473973
             ...
307506     5.430137
307507    11.205479
307508    14.109589
307509     2.550685
307510     1.123288
Name: DAYS_ID_PUBLISH, Length: 307511, dtype: float64
```

In [48]:
```
app_test["DAYS_BIRTH"] = app_test["DAYS_BIRTH"]/-365
```

```
app_test["DAYS_EMPLOYED"] = app_test["DAYS_EMPLOYED"]/-365 ##55,374 entries with valu
app_test["DAYS_EMPLOYED"] = np.where(app_test["DAYS_EMPLOYED"] == 365243/-365,
                                     app_test["DAYS_EMPLOYED"].median(), app_te

app_test["DAYS_REGISTRATION"] = app_test["DAYS_REGISTRATION"]/-365
app_test["DAYS_ID_PUBLISH"] = app_test["DAYS_ID_PUBLISH"]/-365
```

In [51]:

```
print(app_test["DAYS_ID_PUBLISH"])
```

```
0         2.224658
1         4.446575
2         9.597260
3        11.528767
4        11.676712
            ...
48739     9.312329
48740     8.227397
48741     4.120548
48742     3.736986
48743    11.561644
Name: DAYS_ID_PUBLISH, Length: 48744, dtype: float64
```

In [52]:

```
app_train_data['work_history'] = app_train_data['DAYS_EMPLOYED'] / app_train_data['DA
app_train_data["work_history"] = np.where(app_train_data["work_history"] == -0,
                                          0, app_train_data["work_history"])

app_train_data['inc_loan_ratio'] = app_train_data['AMT_INCOME_TOTAL'] / app_train_dat

app_train_data.describe()
```

Out[52]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANN
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.00
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.57
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.73
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.50
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.00
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.00
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.00
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.50

8 rows × 108 columns

In [53]:

```
app_test['work_history'] = app_test['DAYS_EMPLOYED'] / app_test['DAYS_BIRTH']
app_test["work_history"] = np.where(app_test["work_history"] == -0,
                                    0, app_test["work_history"])

app_test['inc_loan_ratio'] = app_test['AMT_INCOME_TOTAL'] / app_test['AMT_CREDIT']

app_test.describe()
```

Out[53]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GO
count	48744.000000	48744.000000	4.874400e+04	4.874400e+04	48720.000000	4.8
mean	277796.676350	0.397054	1.784318e+05	5.167404e+05	29426.240209	4.6
std	103169.547296	0.709047	1.015226e+05	3.653970e+05	16016.368315	3.3
min	100001.000000	0.000000	2.694150e+04	4.500000e+04	2295.000000	4.5
25%	188557.750000	0.000000	1.125000e+05	2.606400e+05	17973.000000	2.2
50%	277549.000000	0.000000	1.575000e+05	4.500000e+05	26199.000000	3.9
75%	367555.500000	1.000000	2.250000e+05	6.750000e+05	37390.500000	6.3
max	456250.000000	20.000000	4.410000e+06	2.245500e+06	180576.000000	2.2

8 rows × 107 columns

In [54]:
```
app_train_data.loc[app_train_data['work_history']<0,["DAYS_BIRTH","DAYS_EMPLOYED"]]
```

Out[54]:

	DAYS_BIRTH	DAYS_EMPLOYED

In [55]:
```
app_test.loc[app_test['work_history']<0,["DAYS_BIRTH","DAYS_EMPLOYED"]]
```

Out[55]:

	DAYS_BIRTH	DAYS_EMPLOYED

Reading Data from Credit Card Balance

In [57]:
```
credit_card_bal = pd.read_csv('../input/credit_card_balance.csv')
```

In [58]:
```
credit_card_bal.head()
```

Out[58]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AMT_I
0	2562384	378907	-6	56.970	135000	
1	2582071	363914	-1	63975.555	45000	
2	1740877	371185	-7	31815.225	450000	
3	1389973	337855	-4	236572.110	225000	
4	1891521	126868	-1	453919.455	450000	

5 rows × 23 columns

In [59]:
```
credit_card_bal['credit_util'] = credit_card_bal['AMT_BALANCE']/ credit_card_bal['AMT
```

In [60]:
```
#credit_card_bal[-np.isinf(credit_card_bal['credit_util'])==True]

#credit_card_bal[credit_card_bal['credit_util']==np.inf].head() #find rows with +ve i
#credit_card_bal[credit_card_bal['credit_util']==-np.inf] #find rows with -ve infinit

#impute
#credit_card_bal['credit_util'] = credit_card_bal['credit_util'].replace(-np.Inf, cre
#credit_card_bal['credit_util'] = credit_card_bal['credit_util'].replace(np.Inf, cred

credit_card_bal["credit_util"] = np.where(credit_card_bal["credit_util"] == -np.inf,
                                          credit_card_bal["credit_util"].median(), c
credit_card_bal["credit_util"] = np.where(credit_card_bal["credit_util"] == np.inf,
                                          credit_card_bal["credit_util"].median(), c
```

Merge Credit Card Features

In [61]:
```
tbl = credit_card_bal.groupby('SK_ID_CURR')['credit_util'].mean().reset_index()

app_train_data = app_train_data.merge(tbl, on='SK_ID_CURR', how='left')

app_train_data.describe()
```

Out[61]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANN
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.00
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.57
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.73
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.50
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.00
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.00
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.00
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.50

8 rows × 109 columns

In [62]:
```
# Merge Previous Application Features

app_train_data = app_train_data.merge(prev_apps_features, on='SK_ID_CURR', how='left'

app_train_data.describe()
```

Out[62]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANN
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.00
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.57
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.73
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.50
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.00
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.00
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.00
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.50

8 rows × 113 columns

In [63]:

```
#application test
app_test = app_test.merge(prev_apps_features, on='SK_ID_CURR', how='left')

app_test.describe()
```

Out[63]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GO
count	48744.000000	48744.000000	4.874400e+04	4.874400e+04	48720.000000	4.8
mean	277796.676350	0.397054	1.784318e+05	5.167404e+05	29426.240209	4.6
std	103169.547296	0.709047	1.015226e+05	3.653970e+05	16016.368315	3.3
min	100001.000000	0.000000	2.694150e+04	4.500000e+04	2295.000000	4.5
25%	188557.750000	0.000000	1.125000e+05	2.606400e+05	17973.000000	2.2
50%	277549.000000	0.000000	1.575000e+05	4.500000e+05	26199.000000	3.9
75%	367555.500000	1.000000	2.250000e+05	6.750000e+05	37390.500000	6.3
max	456250.000000	20.000000	4.410000e+06	2.245500e+06	180576.000000	2.2

8 rows × 111 columns

Reading Data from Bureau File

In [65]:

```
# Reading the data from Bureau

bureau_data = pd.read_csv('../input/bureau.csv')
```

In [66]:

```
tbl1 = bureau_data[['SK_ID_CURR', 'CREDIT_TYPE']].groupby(by = ['SK_ID_CURR'])['CREDI


app_train_data = app_train_data.merge(tbl1, on='SK_ID_CURR', how='left')
app_train_data.describe()
```

Out[66]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANN
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.00
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.57
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.73
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.50
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.00
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.00
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.00
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.50

8 rows × 114 columns

Reading Data from Installment Payments File

In [68]:
```
# Read the installment Data

ins_payments_data = pd.read_csv('../input/installments_payments.csv')
```

In [69]:
```
#DAYS_LATE_PAYMENT>0: pay late; DAYS_LATE_PAYMENT<0: pay early
ins_payments_data['DAYS_LATE_PAYMENT']=ins_payments_data['DAYS_ENTRY_PAYMENT']-ins_pa
ins_payments_data['AMT_LATE_PAYMENT']=ins_payments_data['AMT_PAYMENT']-ins_payments_d
```

In [70]:
```
days_late_payment = ins_payments_data.groupby('SK_ID_CURR')['DAYS_LATE_PAYMENT'].appl
```

In [71]:
```
late_payment = days_late_payment
```

In [72]:
```
amt_late_payment = ins_payments_data.groupby('SK_ID_CURR')['AMT_LATE_PAYMENT'].sum().
amt_late_payment.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 92334 entries, 0 to 92333
Data columns (total 2 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   SK_ID_CURR        92334 non-null  int64  
 1   AMT_LATE_PAYMENT  92334 non-null  float64
dtypes: float64(1), int64(1)
memory usage: 1.4 MB
```

In [73]:
```
# if AMT_LATE_PAYMENT<0 --> late payment
late_payment['AMT_LATE_PAYMENT']=amt_late_payment['AMT_LATE_PAYMENT']
late_payment.head()
```

Out[73]:

	SK_ID_CURR	DAYS_LATE_PAYMENT	AMT_LATE_PAYMENT
0	100002	0	0.0
1	100003	0	0.0
2	100005	0	0.0
3	100006	0	0.0
4	100007	2	0.0

In [74]:
```
app_train_data = app_train_data.merge(late_payment, on='SK_ID_CURR', how='left')
```

In [75]:
```
app_test = app_test.merge(late_payment, on='SK_ID_CURR', how='left')
```

In [76]:
```
app_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 48744 entries, 0 to 48743
Columns: 129 entries, SK_ID_CURR to AMT_LATE_PAYMENT
dtypes: float64(76), int64(37), object(16)
memory usage: 48.3+ MB
```

In [77]:
```
app_train_data.shape
```

Out[77]:
```
(307511, 132)
```

In [80]:
```
app_train_data.describe()
```

Out[80]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANN
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.00
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.57
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.73
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.50
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.00
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.00
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.00
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.50

8 rows × 116 columns

In [81]:
```
app_test.describe()
```

Out[81]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GO
count	48744.000000	48744.000000	4.874400e+04	4.874400e+04	48720.000000	4.8
mean	277796.676350	0.397054	1.784318e+05	5.167404e+05	29426.240209	4.6
std	103169.547296	0.709047	1.015226e+05	3.653970e+05	16016.368315	3.3
min	100001.000000	0.000000	2.694150e+04	4.500000e+04	2295.000000	4.5
25%	188557.750000	0.000000	1.125000e+05	2.606400e+05	17973.000000	2.2
50%	277549.000000	0.000000	1.575000e+05	4.500000e+05	26199.000000	3.9
75%	367555.500000	1.000000	2.250000e+05	6.750000e+05	37390.500000	6.3
max	456250.000000	20.000000	4.410000e+06	2.245500e+06	180576.000000	2.2

8 rows × 113 columns

In [82]:
```
app_train_data.to_csv("../input/app_train_joined.csv",index=False)
```

In [83]:
```
app_test.to_csv("../input/app_test_joined.csv",index=False)
```

Building preprocessing pipeline - phase 3

In [1]:
```
from sklearn.pipeline import Pipeline, FeatureUnion, make_pipeline
from sklearn.compose import ColumnTransformer
from sklearn.datasets import load_iris
import numpy as np
import pandas as pd
from time import time
from sklearn.model_selection import train_test_split  # sklearn.cross_validation in o
from sklearn.preprocessing import StandardScaler, OneHotEncoder, MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score

# Convert a number to a percent.
def pct(x):
    return round(100*x,1)
```

In [2]:
```
app_train_joined = pd.read_csv('../input/app_train_joined.csv')
new_features = ['work_history',
'inc_loan_ratio',
'prev_loans',
'dwn_pymt_pct',
'DAYS_LATE_PAYMENT',
'AMT_LATE_PAYMENT' ]
app_train_joined[new_features].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 6 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   work_history       307511 non-null  float64
 1   inc_loan_ratio     307511 non-null  float64
 2   prev_loans         248722 non-null  float64
 3   dwn_pymt_pct       248722 non-null  float64
 4   DAYS_LATE_PAYMENT  79173 non-null   float64
 5   AMT_LATE_PAYMENT   79173 non-null   float64
dtypes: float64(6)
memory usage: 14.1 MB
```

In [4]:

```
app_train_joined.describe()
```

Out[4]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANN
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.00
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.57
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.73
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.50
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.00
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.00
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.00
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.50

8 rows × 116 columns

In [5]:

```
data = app_train_joined
y = data['TARGET']
selected_features = ['CNT_CHILDREN','AMT_INCOME_TOTAL','AMT_CREDIT','AMT_ANNUITY','AM
                     'DAYS_REGISTRATION','DAYS_ID_PUBLISH','OWN_CAR_AGE','CNT_FAM_ME
                     'EXT_SOURCE_1','EXT_SOURCE_2','EXT_SOURCE_3',
                     'OBS_30_CNT_SOCIAL_CIRCLE','DEF_30_CNT_SOCIAL_CIRCLE','DAYS_LAS
                     'NAME_CONTRACT_TYPE','CODE_GENDER','FLAG_OWN_CAR','FLAG_OWN_REA
                     'NAME_EDUCATION_TYPE','NAME_FAMILY_STATUS','NAME_HOUSING_TYPE',
                     'FLAG_CONT_MOBILE','FLAG_PHONE','FLAG_EMAIL','OCCUPATION_TYPE',
                     'REGION_RATING_CLIENT_W_CITY','REG_REGION_NOT_LIVE_REGION','REG
                     'REG_CITY_NOT_LIVE_CITY','REG_CITY_NOT_WORK_CITY','LIVE_CITY_NO
                     'work_history','inc_loan_ratio','prev_loans','dwn_pymt_pct','DA
X = data[selected_features]

# Split the provided training data into training and validationa and test

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15,shuffle=True
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.2

print(f"X train           shape: {X_train.shape}")
print(f"X validation      shape: {X_valid.shape}")
print(f"X test            shape: {X_test.shape}")
```

```
X train          shape: (209107, 47)
X validation     shape: (52277, 47)
X test           shape: (46127, 47)
```

In [122…

```
X_train.describe()
```

Out[122]:

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REG
count	209107.000000	2.091070e+05	2.091070e+05	209098.000000	2.089100e+05	
mean	0.418035	1.685036e+05	5.995655e+05	27125.263035	5.390325e+05	
std	0.723662	1.062519e+05	4.032270e+05	14486.631596	3.702801e+05	
min	0.000000	2.565000e+04	4.500000e+04	1615.500000	4.500000e+04	
25%	0.000000	1.125000e+05	2.700000e+05	16537.500000	2.385000e+05	
50%	0.000000	1.444500e+05	5.147775e+05	24907.500000	4.500000e+05	
75%	1.000000	2.025000e+05	8.086500e+05	34641.000000	6.795000e+05	
max	19.000000	1.350000e+07	4.050000e+06	258025.500000	4.050000e+06	

8 rows × 36 columns

In [6]:

```
# Identify the numeric features we wish to consider.
numerical_selected_features = ['CNT_CHILDREN','AMT_INCOME_TOTAL','AMT_CREDIT','AMT_AN
                      'DAYS_REGISTRATION','DAYS_ID_PUBLISH','OWN_CAR_AGE','CNT_FAM_ME
                      'EXT_SOURCE_1','EXT_SOURCE_2','EXT_SOURCE_3',
                      'OBS_30_CNT_SOCIAL_CIRCLE','DEF_30_CNT_SOCIAL_CIRCLE','DAYS_LAS
]

num_pipeline =Pipeline([
    ('imputer',SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler())
])

numerical_features_fill0 = ['prev_loans','dwn_pymt_pct','DAYS_LATE_PAYMENT','AMT_LATE

num_features_fill0_pipeline = Pipeline([
    ('imputer',SimpleImputer(strategy="constant")),
    ('std_scaler', StandardScaler())
])

# Identify the categorical features we wish to consider.
categorical_selected_features = ['NAME_CONTRACT_TYPE','CODE_GENDER','FLAG_OWN_CAR','F
                       'NAME_EDUCATION_TYPE','NAME_FAMILY_STATUS','NAME_HOUSING_TYPE
                       'FLAG_CONT_MOBILE','FLAG_PHONE','FLAG_EMAIL','OCCUPATION_TYPE
                       'REGION_RATING_CLIENT_W_CITY','REG_REGION_NOT_LIVE_REGION','R
                       'REG_CITY_NOT_LIVE_CITY','REG_CITY_NOT_WORK_CITY','LIVE_CITY_
]

# Create a pipelne for the categorical features.
# Entries with missing values or values that don't exist in the range
# defined above will be one hot encoded as zeroes.
cat_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
```

```
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
    ])


data_pipeline = ColumnTransformer(transformers=[
        ("num_pipeline", num_pipeline, numerical_selected_features),
        ("num_features_fill0_pipeline", num_features_fill0_pipeline, numerical_featur
        ("cat_pipeline", cat_pipeline, categorical_selected_features)],
        remainder='drop',
        n_jobs=-1
    )
```

In [7]:

```
# X_train_transformed, X_train_transformed, X_train_transformed might have different
# Solution: use fit_transform(X_train), transform(X_valid), transform(X_test)
X_train_transformed = data_pipeline.fit_transform(X_train)
print(f"X train after transform          shape: {X_train_transformed.shape}")
column_names = numerical_selected_features + numerical_features_fill0  + \
              list(data_pipeline.transformers_[2][1].named_steps["ohe"].get_feature_n
display(pd.DataFrame(X_train_transformed,  columns=column_names).head())
```

```
X train after transform          shape: (209107, 168)
/usr/local/lib/python3.9/site-packages/sklearn/utils/deprecation.py:87: FutureWarnin
g: Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 a
nd will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_
0	2.186062	-0.230619	-0.147722	0.149785	0.002841	
1	0.804197	0.531722	2.883109	1.336114	2.799270	
2	-0.577667	-0.315323	-1.085832	-1.086556	-1.091414	
3	-0.577667	-0.950607	-0.817322	-0.910735	-0.726663	
4	-0.577667	0.319961	1.938237	0.758326	1.802282	

5 rows × 168 columns

In [8]:

```
pd.DataFrame(X_train_transformed,  columns=column_names).describe()
```

Out[8]:

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	RE
count	2.091070e+05	2.091070e+05	2.091070e+05	2.091070e+05	2.091070e+05	
mean	-1.156675e-16	2.086534e-16	3.933169e-17	-1.133228e-16	-6.214917e-17	
std	1.000002e+00	1.000002e+00	1.000002e+00	1.000002e+00	1.000002e+00	
min	-5.776675e-01	-1.344484e+00	-1.375322e+00	-1.760952e+00	-1.334582e+00	
25%	-5.776675e-01	-5.270845e-01	-8.173221e-01	-7.308749e-01	-8.117713e-01	
50%	-5.776675e-01	-2.263833e-01	-2.102742e-01	-1.530873e-01	-2.403271e-01	
75%	8.041972e-01	3.199611e-01	5.185292e-01	5.188237e-01	3.797507e-01	
max	2.567776e+01	1.254709e+02	8.557073e+00	1.593923e+01	9.486383e+00	

8 rows × 168 columns

In [9]:

```
X_valid_transformed = data_pipeline.transform(X_valid)
print(f"X valid after transform          shape: {X_valid_transformed.shape}")
column_names = numerical_selected_features + numerical_features_fill0  + \
              list(data_pipeline.transformers_[2][1].named_steps["ohe"].get_feature_n
display(pd.DataFrame(X_valid_transformed,  columns=column_names).head())
```

```
X valid after transform          shape: (52277, 168)
/usr/local/lib/python3.9/site-packages/sklearn/utils/deprecation.py:87: FutureWarnin
g: Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 a
nd will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_
0	-0.577667	-0.527085	-0.213366	-0.158058	-0.216010	
1	-0.577667	-0.103562	-0.370922	-0.319279	-0.240327	
2	-0.577667	0.955245	1.506323	0.572564	1.097096	
3	2.186062	-0.103562	-0.282535	0.643079	-0.240327	
4	2.186062	0.108200	-0.258787	0.558585	-0.301119	

5 rows × 168 columns

In [10]:

```
pd.DataFrame(X_valid_transformed,  columns=column_names).describe()
```

Out[10]:

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REG
count	52277.000000	52277.000000	52277.000000	52277.000000	52277.000000	
mean	-0.007021	-0.000578	-0.003099	-0.004517	-0.003860	
std	0.983147	0.950038	0.996419	1.002952	0.995775	
min	-0.577667	-1.344484	-1.375322	-1.735790	-1.346740	
25%	-0.577667	-0.527085	-0.817322	-0.736466	-0.811771	
50%	-0.577667	-0.188266	-0.214582	-0.154641	-0.240327	
75%	0.804197	0.319961	0.518529	0.515096	0.379751	
max	7.713520	40.766388	8.510033	13.659453	8.574504	

8 rows × 168 columns

In [11]:

```
X_test_transformed = data_pipeline.transform(X_test)
print(f"X test after transform          shape: {X_test_transformed.shape}")
column_names = numerical_selected_features + numerical_features_fill0  + \
             list(data_pipeline.transformers_[2][1].named_steps["ohe"].get_feature_n
display(pd.DataFrame(X_test_transformed,  columns=column_names).head())
```

```
X test after transform          shape: (46127, 168)
/usr/local/lib/python3.9/site-packages/sklearn/utils/deprecation.py:87: FutureWarnin
g: Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 a
nd will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_
0	2.186062	0.362313	-0.332588	1.761378	-0.325436	
1	-0.577667	0.743484	1.691724	1.506343	1.729332	
2	-0.577667	0.319961	-0.259322	0.827288	-0.118743	
3	-0.577667	0.743484	-0.855266	-0.150913	-0.848246	
4	-0.577667	-0.527085	-0.722752	-0.777471	-0.823930	

5 rows × 168 columns

In [12]:

```
pd.DataFrame(X_test_transformed,  columns=column_names).describe()
```

Out[12]:

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REG
count	46127.000000	46127.000000	46127.000000	46127.000000	46127.000000	
mean	-0.001099	0.019122	-0.005409	-0.002557	-0.007015	
std	1.004757	5.258075	0.991870	0.999950	0.989855	
min	-0.577667	-1.340248	-1.375322	-1.735790	-1.334582	
25%	-0.577667	-0.527085	-0.817322	-0.728390	-0.811771	
50%	-0.577667	-0.188266	-0.217004	-0.153087	-0.240327	
75%	0.804197	0.319961	0.518529	0.510747	0.379751	
max	16.004708	1099.573374	8.557073	14.015755	9.486383	

8 rows × 168 columns

Multi-layer Perception (MLP) model - phase 3

We use Tensorboard to visual the traning accuracy, training AUC, and training loss.

In [144…

```
from time import time
from torch.utils.tensorboard import SummaryWriter
from sklearn.metrics import accuracy_score
import torch
#import torchvision
import torch.utils.data
#import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from sklearn.metrics import roc_curve

# is there a GPU availabale. If available use it
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# Assuming that we are on a CUDA machine, this should print a CUDA device:
print(f"We are working on a {device} device")

writer = SummaryWriter()
print(y_train.dtypes)
# convert numpy arrays to tensors
X_train_tensor = torch.from_numpy(X_train_transformed)
X_validation_tensor = torch.from_numpy(X_valid_transformed)
X_test_tensor = torch.from_numpy(X_test_transformed)
y_train_tensor = torch.from_numpy(y_train.values)
y_test_tensor = torch.from_numpy(y_test.values)
y_validation_tensor = torch.from_numpy(y_valid.values)

# create TensorDataset in PyTorch
hcrd_train = torch.utils.data.TensorDataset(X_train_tensor, y_train_tensor)
hcrd_validation = torch.utils.data.TensorDataset(X_validation_tensor, y_validation_te
hcrd_test = torch.utils.data.TensorDataset(X_test_tensor, y_test_tensor)
```

```
# create dataloader
batch_size = 96
trainloader_hcrd = torch.utils.data.DataLoader(hcrd_train, batch_size=batch_size, shu
validloader_hcrd = torch.utils.data.DataLoader(hcrd_validation, batch_size=X_valid_tr
testloader_hcrd = torch.utils.data.DataLoader(hcrd_test, batch_size=X_test_transforme

# Neural Network architecture
D_in = X_train_transformed.shape[1]
print(D_in)
D_hidden = 512
D_out = 2
# Use the nn package to define our model and loss function.
# use the sequential API makes things simple
model = torch.nn.Sequential(
    torch.nn.Linear(D_in, D_hidden),   # X.matmul(W1)
    nn.ReLU(),
    nn.Linear(in_features=D_hidden, out_features=D_out)       # Relu( X.matmul(W1)).m

)

model.to(device)

# loss scaffolding layer
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)


print('-'*50)
print('Model:')
print(model)
print('-'*50)


'''
Training Process:
    Load a batch of data.
    Zero the grad.
    Predict the batch of the data through net i.e forward pass.
    Calculate the loss value by predict value and true value.
    Backprop i.e get the gradient with respect to parameters
    Update optimizer i.e gradient update
'''
start = time()
epochs = range(20)
train_losses = {}
train_accuracy = {}
for epoch in epochs:
    running_loss = []
    y_pred = []
    epoch_target = []
    #probs_np = []
    for batch, data in enumerate(trainloader_hcrd):
        inputs_train, target_train = data[0].to(device), data[1].to(device)

        # Clear gradient buffers because we don't want any gradient from previous epo
        optimizer.zero_grad()

        # do forward pass
        output_train = model(inputs_train.float())
```

```
        # compute loss and gradients
        loss = loss_fn(output_train, target_train)

        # get gradients w.r.t to parameters
        loss.backward()
        # perform gradient update
        optimizer.step()

        y_pred.extend(torch.argmax(output_train, dim=1).tolist())
        epoch_target.extend(target_train.tolist())
        running_loss.append(loss.item())

    epoch_training_loss = np.mean(running_loss)

    train_losses[epoch+1] = epoch_training_loss
    print(f"Epoch {epoch+1}, Training loss: {np.round(epoch_training_loss, 3)}")

    #accuracy
    correct = (np.array(y_pred) == np.array(epoch_target))
    accuracy = correct.sum()/correct.size
    train_accuracy[epoch+1] = accuracy
    print(f"Epoch {epoch+1}, Training accuracy: {np.round(accuracy, 3)}")
    print(f"Train AUC: {np.round(roc_auc_score(y_pred, epoch_target), 3)}")
    writer.add_scalar("Training Loss",np.round(epoch_training_loss, 3),epoch+1)
    writer.add_scalar("Training accuracy",np.round(accuracy, 3),epoch+1)
    writer.add_scalar("Training AUC",np.round(roc_auc_score(y_pred, epoch_target), 3)

writer.flush()

print('Finished Training')
print('-'*50)
train_time = np.round(time() - start, 4)

validation_batch_losses = []
validation_y_pred = []
validation_target = []
for batch, data in enumerate(validloader_hcrd):
    inputs_2, target_2 = data[0].to(device), data[1].to(device)
    # do forward pass
    output_2 = model(inputs_2.float())

    # compute loss
    loss = loss_fn(output_2, target_2)

    validation_batch_losses.append(loss.item())

    validation_y_pred.extend(torch.argmax(output_2, dim=1).tolist())
    validation_target.extend(target_2.tolist())

print(f"Validation loss: {np.round(np.mean(validation_batch_losses), 3)}")

#accuracy
validation_correct = (np.array(validation_y_pred) == np.array(validation_target))
validation_accuracy = validation_correct.sum()/ validation_correct.size
print(f"validation accuracy: {np.round(validation_accuracy, 3)}")
print(f"validation AUC: {np.round(roc_auc_score(validation_y_pred,validation_target),

start = time()
test_batch_losses = []
```

```
test_y_pred = []
test_target = []
for batch, data in enumerate(testloader_hcrd):
    inputs_3, target_3 = data[0].to(device), data[1].to(device)
    # do forward pass
    output_3 = model(inputs_3.float())

    # compute loss
    loss = loss_fn(output_3, target_3)

    test_batch_losses.append(loss.item())

    test_y_pred.extend(torch.argmax(output_3, dim=1).tolist())
    test_target.extend(target_3.tolist())

print(f"Test loss: {np.round(np.mean(test_batch_losses), 3)}")

#accuracy
test_correct = (np.array(test_y_pred) == np.array(test_target))
test_accuracy = test_correct.sum()/ test_correct.size
print(f"Test accuracy: {np.round(test_accuracy, 3)}")
print(f"Test AUC: {np.round(roc_auc_score(test_y_pred,test_target), 3)}")
test_time = np.round(time() - start, 4)
writer.close()
```

```
We are working on a cpu device
int64
168
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=168, out_features=512, bias=True)
  (1): ReLU()
  (2): Linear(in_features=512, out_features=2, bias=True)
)
--------------------------------------------------
Epoch 1, Training loss: 0.259
Epoch 1, Training accuracy: 0.919
Train AUC: 0.514
Epoch 2, Training loss: 0.25
Epoch 2, Training accuracy: 0.92
Train AUC: 0.686
Epoch 3, Training loss: 0.25
Epoch 3, Training accuracy: 0.92
Train AUC: 0.693
Epoch 4, Training loss: 0.249
Epoch 4, Training accuracy: 0.92
Train AUC: 0.699
Epoch 5, Training loss: 0.249
Epoch 5, Training accuracy: 0.92
Train AUC: 0.718
Epoch 6, Training loss: 0.248
Epoch 6, Training accuracy: 0.92
Train AUC: 0.72
Epoch 7, Training loss: 0.248
Epoch 7, Training accuracy: 0.92
Train AUC: 0.725
Epoch 8, Training loss: 0.248
Epoch 8, Training accuracy: 0.92
Train AUC: 0.741
Epoch 9, Training loss: 0.248
Epoch 9, Training accuracy: 0.92
Train AUC: 0.747
Epoch 10, Training loss: 0.248
Epoch 10, Training accuracy: 0.92
Train AUC: 0.749
Epoch 11, Training loss: 0.247
Epoch 11, Training accuracy: 0.92
Train AUC: 0.752
Epoch 12, Training loss: 0.247
Epoch 12, Training accuracy: 0.92
Train AUC: 0.755
Epoch 13, Training loss: 0.247
Epoch 13, Training accuracy: 0.92
Train AUC: 0.757
Epoch 14, Training loss: 0.247
Epoch 14, Training accuracy: 0.92
Train AUC: 0.759
Epoch 15, Training loss: 0.247
Epoch 15, Training accuracy: 0.92
Train AUC: 0.761
Epoch 16, Training loss: 0.247
Epoch 16, Training accuracy: 0.92
Train AUC: 0.758
Epoch 17, Training loss: 0.246
```

```
Epoch 17, Training accuracy: 0.92
Train AUC: 0.765
Epoch 18, Training loss: 0.246
Epoch 18, Training accuracy: 0.92
Train AUC: 0.766
Epoch 19, Training loss: 0.246
Epoch 19, Training accuracy: 0.92
Train AUC: 0.773
Epoch 20, Training loss: 0.246
Epoch 20, Training accuracy: 0.92
Train AUC: 0.773
Finished Training
--------------------------------------------------
Validation loss: 0.255
validation accuracy: 0.916
validation AUC: 0.717
Test loss: 0.248
Test accuracy: 0.92
Test AUC: 0.778
```

In [146…

```
try:
    result1
except NameError:
    result1 = pd.DataFrame(columns=["exp_name", "TrainAcc", "ValidAcc", "TestAcc", "T
                                    "TestAUC", "Train Time(s)", "Test Time(s)", "Descr

result1.loc[0] = ["MLP model", np.round(accuracy, 3), np.round(validation_accuracy, 3
                  np.round(roc_auc_score(y_pred, epoch_target), 3), np.round(roc_auc_
                 np.round(roc_auc_score(test_y_pred,test_target), 3), train_time, tes
result1
```

Out[146]:

	exp_name	TrainAcc	ValidAcc	TestAcc	TrainAUC	ValidAUC	TestAUC	Train Time(s)	Test Time(s)	Descripti
0	MLP model	0.92	0.916	0.92	0.773	0.717	0.778	79.1007	1.4473	MLP w one hidc la

Kaggle Submission - phase 3

In [98]:

```
import pandas as pd
app_test_joined = pd.read_csv('../input/app_test_joined.csv')
```

In [99]:

```
app_test_joined.describe()
```

Out[99]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GO
count	48744.000000	48744.000000	4.874400e+04	4.874400e+04	48720.000000	4.8
mean	277796.676350	0.397054	1.784318e+05	5.167404e+05	29426.240209	4.6
std	103169.547296	0.709047	1.015226e+05	3.653970e+05	16016.368315	3.3
min	100001.000000	0.000000	2.694150e+04	4.500000e+04	2295.000000	4.5
25%	188557.750000	0.000000	1.125000e+05	2.606400e+05	17973.000000	2.2
50%	277549.000000	0.000000	1.575000e+05	4.500000e+05	26199.000000	3.9
75%	367555.500000	1.000000	2.250000e+05	6.750000e+05	37390.500000	6.3
max	456250.000000	20.000000	4.410000e+06	2.245500e+06	180576.000000	2.2

8 rows × 113 columns

In [100…

```
X_Kaggle_test = app_test_joined[selected_features]
X_Kaggle_test.describe()
```

Out[100]:

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REG
count	48744.000000	4.874400e+04	4.874400e+04	48720.000000	4.874400e+04	
mean	0.397054	1.784318e+05	5.167404e+05	29426.240209	4.626188e+05	
std	0.709047	1.015226e+05	3.653970e+05	16016.368315	3.367102e+05	
min	0.000000	2.694150e+04	4.500000e+04	2295.000000	4.500000e+04	
25%	0.000000	1.125000e+05	2.606400e+05	17973.000000	2.250000e+05	
50%	0.000000	1.575000e+05	4.500000e+05	26199.000000	3.960000e+05	
75%	1.000000	2.250000e+05	6.750000e+05	37390.500000	6.300000e+05	
max	20.000000	4.410000e+06	2.245500e+06	180576.000000	2.245500e+06	

8 rows × 36 columns

In [139…

```
X_Kaggle_test_transformed = data_pipeline.transform(X_Kaggle_test)
print(f"X_Kaggle_test after transform          shape: {X_Kaggle_test_transformed.shap
column_names = numerical_selected_features + numerical_features_fill0  + \
              list(data_pipeline.transformers_[2][1].named_steps["ohe"].get_feature_n
display(pd.DataFrame(X_Kaggle_test_transformed,  columns=column_names).head())
```

```
X_Kaggle_test after transform          shape: (48744, 168)
```

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGIC
0	-0.577667	-0.315323	-0.076299	-0.453164	-0.240327	
1	-0.577667	-0.654141	-0.934457	-0.673407	-0.969830	
2	-0.577667	0.319961	0.157972	2.944289	0.246008	
3	2.186062	1.378768	2.419076	1.511314	2.799270	
4	0.804197	0.108200	0.064317	0.341139	0.233850	

5 rows × 168 columns

In [102…

```
pd.DataFrame(X_Kaggle_test_transformed,  columns=column_names).describe()
```

Out[102]:

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REG
count	48744.000000	48744.000000	48744.000000	48744.000000	48744.000000	
mean	-0.028993	0.093440	-0.205406	0.158691	-0.206233	
std	0.979807	0.955492	0.906184	1.105372	0.909745	
min	-0.577667	-1.332328	-1.375322	-1.714046	-1.334582	
25%	-0.577667	-0.527085	-0.840535	-0.631781	-0.848246	
50%	-0.577667	-0.103562	-0.370922	-0.064245	-0.386228	
75%	0.804197	0.531722	0.187077	0.708624	0.246008	
max	27.059626	39.919342	4.081915	10.592830	4.610870	

8 rows × 168 columns

In [135…

```
X_Kaggle_test_tensor = torch.from_numpy(X_Kaggle_test_transformed)
# Disable grad
with torch.no_grad():

    prediction = model(X_Kaggle_test_tensor.float())
    # Predicted class value
    sm = torch.nn.Softmax(dim=1)
    probs = sm(prediction)
    print(type(probs[:,1].tolist()))
    submit_df = app_test_joined[['SK_ID_CURR']]
    submit_df['TARGET'] = probs[:,1].tolist()

    print(submit_df.head())
```

```
<class 'list'>
   SK_ID_CURR    TARGET
0      100001  0.053653
1      100005  0.199241
2      100013  0.050457
3      100028  0.043939
4      100038  0.114051
```

In [136…

```
submit_df.to_csv("MLP1_submission.csv",index=False)
```

Write-up - phase 3

Group members

Sai Varun Datta Vemavarapu savemava@iu.edu

Binh Bui binbui@iu.edu

Kevaun Stewart kevstew@iu.edu

Prasad Yacham pyacham@iu.edu

Abstract

In Phase 3, we take the features engineered from phase 2 and build a model with multi-layer perceptron (MLP) using Pytorch. We also integrate our model with Tensorboard to view the training performance real time.

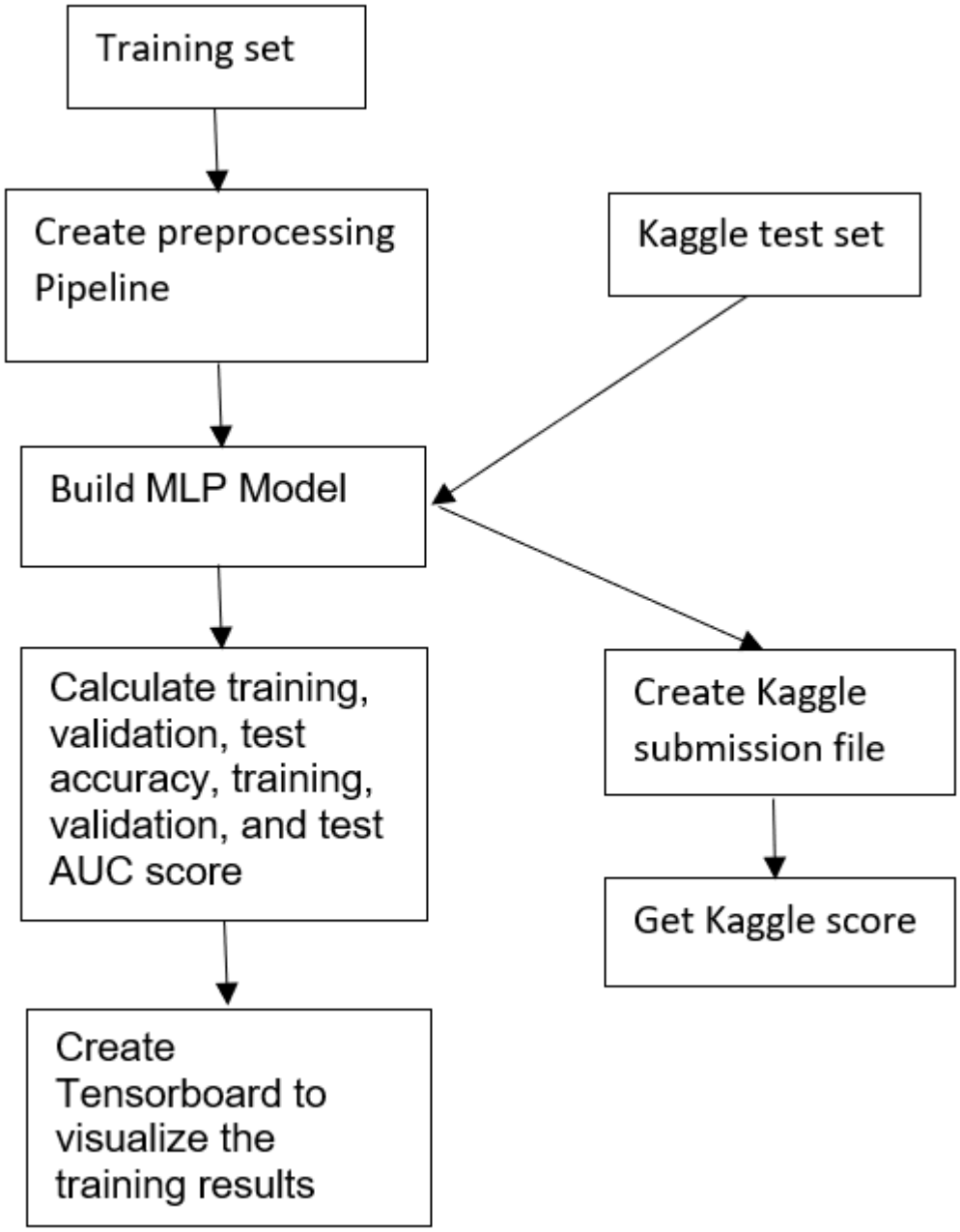
Data leakage analysis was performed on the machine learning pipeline and found this risk is minimized as the available data is split into training, test, and validation sets. The imputing strategy, used on features, leverages constants instead of summary statistics of the data, which avoids sharing data between the 3 highlighted sets. However, we are mindful not to overinterpret our results as the improvements using the MLP model are minor, and worth significance testing.

The MLP model was implemented with one hidden layer comprising 512 neurons. Using this MLP model we see that our Test AUC score has increased from 0.76 to 0.778. Also, the Kaggle submission public score has increased from 0.725 to 0.7408.

Project Description

- **Data description**: See section *1.1 Dataset*
- **Task to be tackled**:
    - Create preprocessing pipeline for training set and test set
    - Build a multi-layer perception model
    - Calculate training, validation, test accuracy, training, validation, and test AUC score
    - Create Tensorboard to visualize the training results
    - Create a submission file on Kaggle

- **Workflow**:



Leakage detection

In phase 2, we filled missing value of 4 features 'prev_loans', 'dwn_pymt_pct', 'DAYS_LATE_PAYMENT', 'AMT_LATE_PAYMENT' with 0 right after loading data from file 'app_train_joined.csv'. Then we splited the data in to traing, validation and test sets.

After reviewing the pipeline steps taken in phase 2, we have addressed Data Leakage concerns. Mainly in the below areas:

- We are not at risk of leakage from Feature Engineering as we impute values of zero instead of characteristic metrics, such as mean and median. We are not risking leakage between

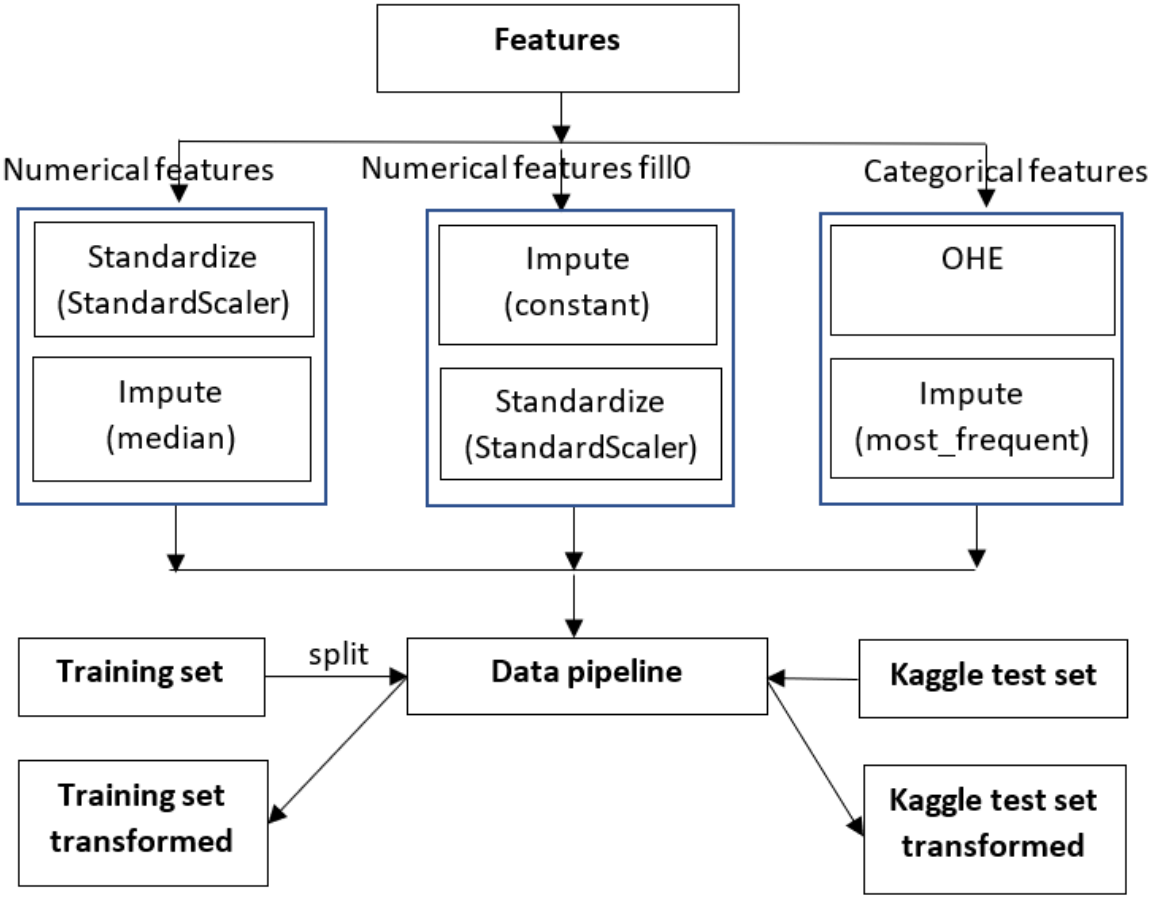
training, test and validation datasets from Feature Engineering

- We avoid fitting our models with test data; we use only the training dataset. The test dataset is used for prediction purposes.
- To perform clean tests of our machine learning models, we avoid fitting our models with the entire dataset. We maintain a holdout (test) dataset that is used for pure tests of our model

In phase 3, we splited the data right after loading file 'app_train_joined.csv' to be safe.

Preprocessing Pipeline

Here is our pipeline diagram:



There are 47 raw features in our training set.

Numerical features: We use StanardScaler to scale these features and impute the missing value with median.

```
numerical_selected_features = ['CNT_CHILDREN', 'AMT_INCOME_TOTAL',
'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
'REGION_POPULATION_RELATIVE', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH',
'OWN_CAR_AGE', 'CNT_FAM_MEMBERS',
 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'OBS_30_CNT_SOCIAL_CIRCLE',
'DEF_30_CNT_SOCIAL_CIRCLE', 'DAYS_LAST_PHONE_CHANGE', 'work_history',
'inc_loan_ratio']
```

The numerical features fill0: We use StanardScaler to scale these features and impute the missing value with 0.

```
numerical_features_fill0 =
['prev_loans','dwn_pymt_pct','DAYS_LATE_PAYMENT','AMT_LATE_PAYMENT']
```

The categorical features: We use OneHotEncoder to scale these features and impute the missing value with 'most_frequent' strategy.

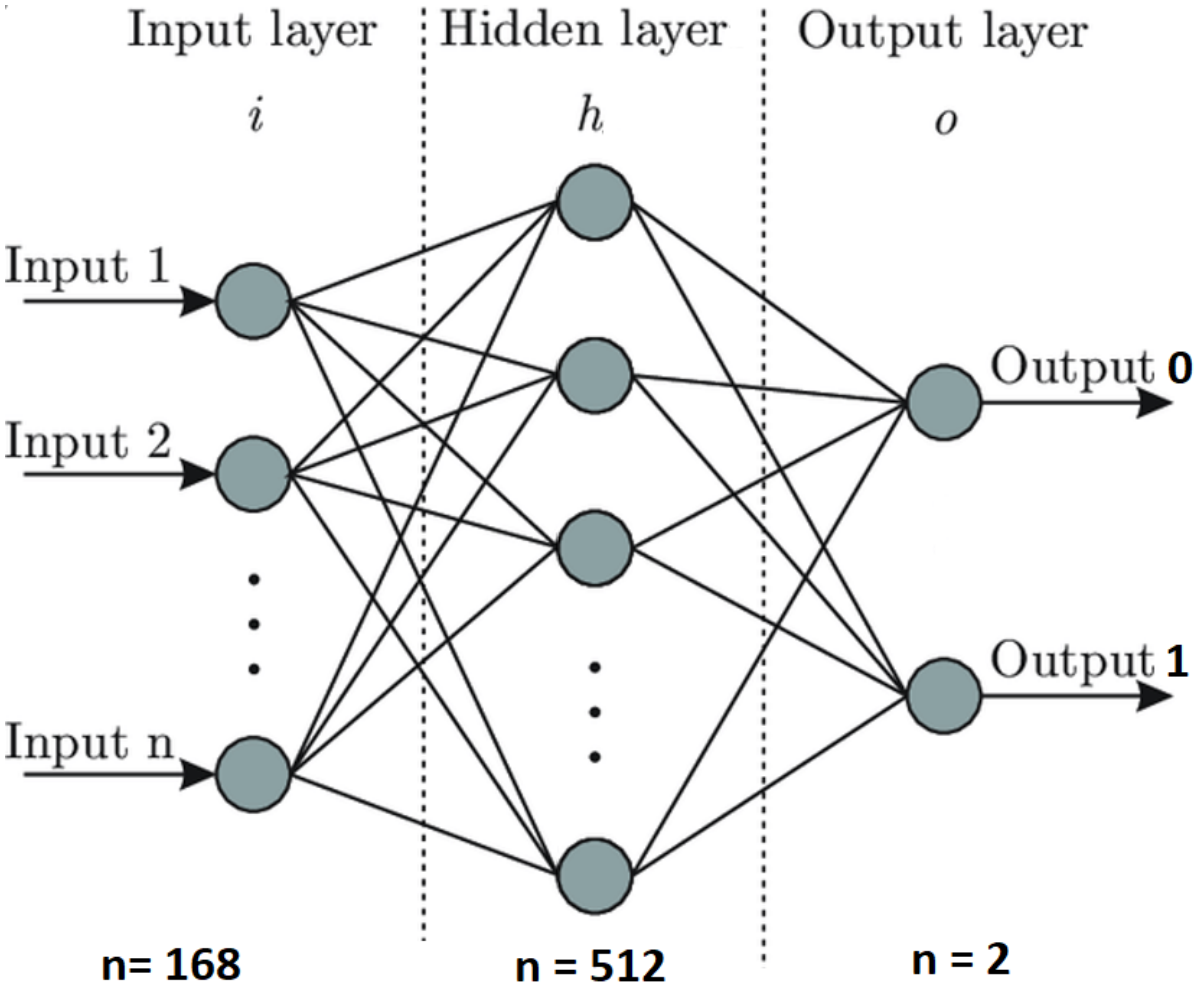
```
categorical_selected_features = ['NAME_CONTRACT_TYPE', 'CODE_GENDER',
'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE',
'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE',
'FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE',
'FLAG_PHONE', 'FLAG_EMAIL', 'OCCUPATION_TYPE' ,'REGION_RATING_CLIENT',
'REGION_RATING_CLIENT_W_CITY', 'REG_REGION_NOT_LIVE_REGION',
'REG_REGION_NOT_WORK_REGION', 'LIVE_REGION_NOT_WORK_REGION',
'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WORK_CITY',
'LIVE_CITY_NOT_WORK_CITY', 'ORGANIZATION_TYPE']
```

MLP Modeling

We use CrossEntropyLoss for this model. More detail about this loss function can be seen here

Our Multilayer Perception model has one hidden layer with 512 neurons. The input layer has 168 features (after transforming), and the output layer has 2 outputs, 0 and 1.

Here is the MLP modeling architecture:




```
model = torch.nn.Sequential(
    torch.nn.Linear(168, 512),
    nn.ReLU(),
    nn.Linear(in_features=512, out_features=2)

)

loss_fn = torch.nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

Evaluation metric

Area Under the Curve-Receiver Operating Characteristics (AUC-ROC) score will be used to evaluate the accuracy of various logistic regression models (based on varying hyperparameters and thresholds). The AUC score represents the area under the ROC graph. The ROC graph represents a plot of the True Positive Rate (TPR) versus the False Positive Rate (FPR) – where each point on the graph represents a different logistic regression model. The higher the AUC-ROC score, the better. TPR and FPR are defined in equations 1 and 2, respectively.

Equation 1:

True Positive Rate, TPR = True Positives / (True Positives + False Negatives)

Where:

```
True Positives = samples accurately classified as class 1 (class 1 =
will default on loan)

False Negatives = samples incorrectly classified as class 0 (class 0 =
will not default on loan)
```

Equation 2:

False Positive Rate, FPR = False Positives / (True Negatives + False Positives)

Where:

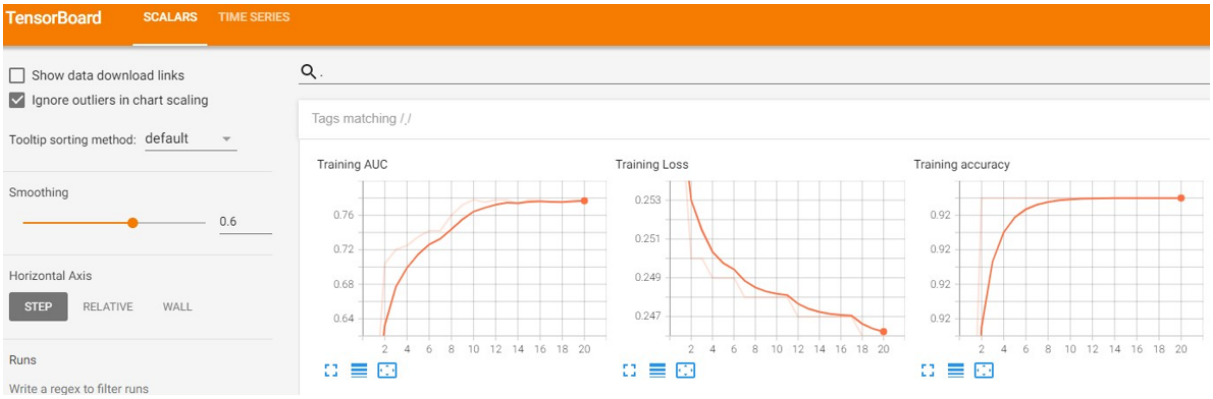
```
False Positives = samples incorrectly classified as class 1

True Negatives = samples accurately classified as class 0
```

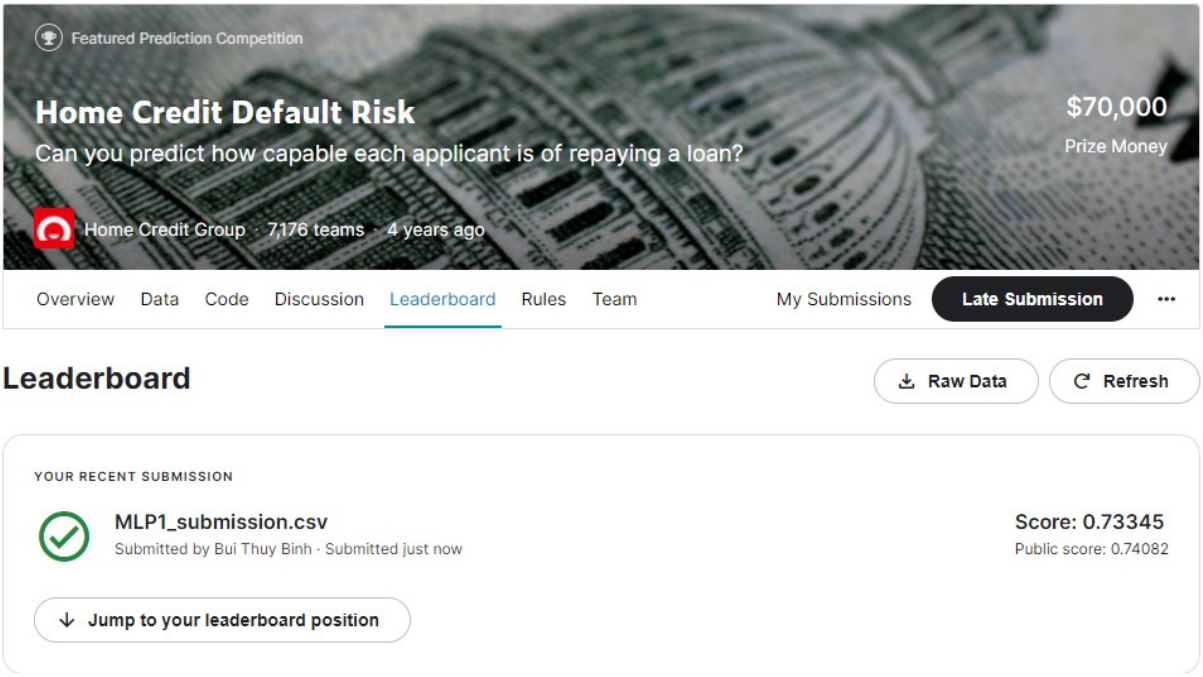
Results

	exp_name	TrainAcc	ValidAcc	TestAcc	TrainAUC	ValidAUC	TestAUC	Train Time(s)	Test Time(s)	Description
0	MLP model	0.92	0.916	0.92	0.773	0.717	0.778	79.1007	1.4473	MLP with one hidden layer

Tensorboard result:



Kaggle Submission



Discussion

We have a Multi-layer Perception model that has one hidden layer with 512 neurons. The input layer has 168 features, and the output layer has 2 outputs, 0 and 1.

The training AUC and test AUC of our model in phase 3 are 0.773 and 0.778. These results are better than the results of our best model (LightGBM) in phase 2 in overall. The traing AUC of the LightGBM model in phase 2 was 0.826, bit the test AUC was 0.76 which is far from 0.826.

The tensorboard results demonstrates that the training accuracy of the MLP model is from 0.919 to 0.92 which is similar to models in phase 1 and 2.

After 20 eporchs, the training loss descreases gradually from 0.259 to 0.246.

Kaggle Submission:

We applied the MLP model to create Kaggle submission file and got public score at 0.74082 and private scores at 0.73345. Our Kaggle scores in this phase are highest among 3 phases.

Overall:

With all our experiments we have created, MLP model has the highest test AUC in our notebook (0.778) as well as in Kaggle (public score at 0.74082 and private scores at 0.73345).

Conclusion

Our main aim for this project is to efficiently provide financial needs to customers with little or no credit history.

In the third phase, we have addressed the data leakage concerns on the machine learning pipeline and found this risk is minimized as the available data is split into training, test, and

validation sets, we build a multi-layer perceptron model which gave us a test AUC about 77.8%.

The model was implemented with one hidden layer comprising 512 neurons. Using this MLP model we see that our Test AUC score has increased from 0.76 to 0.778.

Also we have implemented tensor board to visualize the modeling pipelines to view the training performance real time using PyTorch.

The idea is to improve the accuracy of our models, and Kaggle submission public score has increased from 0.725 to 0.7408.

For future scope, we can further do feature Engineering on other tables, expand the grid search and build MLP with more hidden layers, and experiment with different activation functions.