Home Credit Default Risk

Group 23

TEAM AND PROJECT META INFORMATION

Email IDs:

Raj Chavan : rchavan@iu.edu

Sanket Bailmare: sbailmar@iu.edu

Shefali Luley: sluley@iu.edu

Tanay Kulkarni : tankulk@iu.edu

Group : 23

Members: Raj Chavan Sanket Bailmare Shefali Luley Tanay Kulkarni

PROJECT ABSTRACT

In today's world, many people struggle to get loans due to insufficient credit histories or even non-existing credit records, which often tend to untrustworthy lenders who exploit them.Home Credit acts towards expanding the financial inclusion for the unbanked by providing a secure borrowing experience. Home credit utilizes several alternative data and methods to predict their repayment abilities. To ensure that this underserved demographic has a favorable loan experience, we will be using machine learning and statistical methods to determine these predictions. This ensures that the clients who are capable of repayment will be granted a loan and are not rejected by any means. Also, they will be given a loan maturity plan and a repayment calendar that will accredit our clients to be more successful. Our goal in this phase is to work on the Home Credit Default Risk (HCDR) data and perform some cleaning and preprocessing techniques and modeling techniques with pipelining to predict whether a person receives a loan or not. After merging all subordinate files, cleaning the data, performing some exploratory data analysis, we apply some preprocessing steps which also include dividing the data into numerical and categorical values. The next step involved creating new features

from the existing features that could add on as a good predictor in the existing dataset. The project included 3 phases where, the first phase included training the model without feature engineering and hyperparameter tuning, the second phase included creation of 11 new features from the existing important features and creating a modeling pipeline where we determined the best hyperparameters (hyperparameter tuning) for the models to select the best model. The testing data was then parsed through the best model to get the test results. The best model for the second phase was of the Random Forest Classifier with a training accuracy of 92.4% and a test roc score of 0.72814. The third and the final phase included creation of a Multi Layer Perceptron (MLP) neural network using Pytorch.The Artificial Neural Network model yielded a training score of 91.95% and a test roc score of 0.71225.

In [ ]:

PROJECT DESCRIPTION

DATA DESCRIPTION

- The Home Credit Group released the 'Home Credit Default Risk' dataset three years ago. This dataset may be used to forecast how successfully a customer with no credit history would repay a loan. This data is divided into two files: application_train/test.csv, which is the primary table, and Train (with TARGET) and Test (without TARGET).
- All prior credits given by other financial institutions that were reported to Credit Bureau are contained in Bureau.csv.
- Bureau balance.csv is a spreadsheet that provides monthly credit bureau balances and other information. There are a total of ten CSV files. The overall file size is 2.68 GB.
- POS_CASH_balance.csv is a monthly balance snapshot of the applicant's prior POS (point of sale) and cash loans with Home Credit. Each month of history of every prior credit in Home Credit (consumer credit and cash loans) connected to loans in our sample has one row in this table.
- Monthly balance snapshots of prior credit cards that the applicant holds with Home Credit are contained in Credit_card_balance.csv. This table contains one row for each month of history for every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – that is, the table contains (#loans in sample *# of relative previous credit cards* # of months where we have some history observable for the previous credit card) rows.
- All prior applications for Home Credit loans of consumers who have loans in our sample are contained in the previous_application.csv. In our data set, there is one entry for each previous loan application.
- Repayment history for previously disbursed credits in Home Credit connected to the loans in our sample is contained in the Installments_payments.csv file. There is a row for each payment paid, as well as a row for each missed payment. In our sample, one row is one payment of one installment OR one installment equals one payment of one prior Home Credit credit connected to loans.

- The file HomeCredit_columns_description.csv offers descriptions for the different data files' columns.

DataSet Link: https://www.kaggle.com/c/home-credit-default-risk/data

Tasks to be tackled :

Phase 3

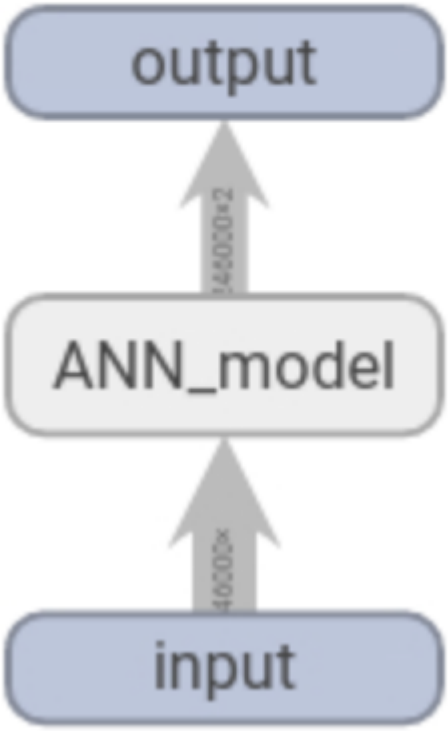
- Use a Multi Layer Perceptron (MLP) Model using Pytorch for loan default classification
- Use Tensorboard to visualize the results of training and modeling

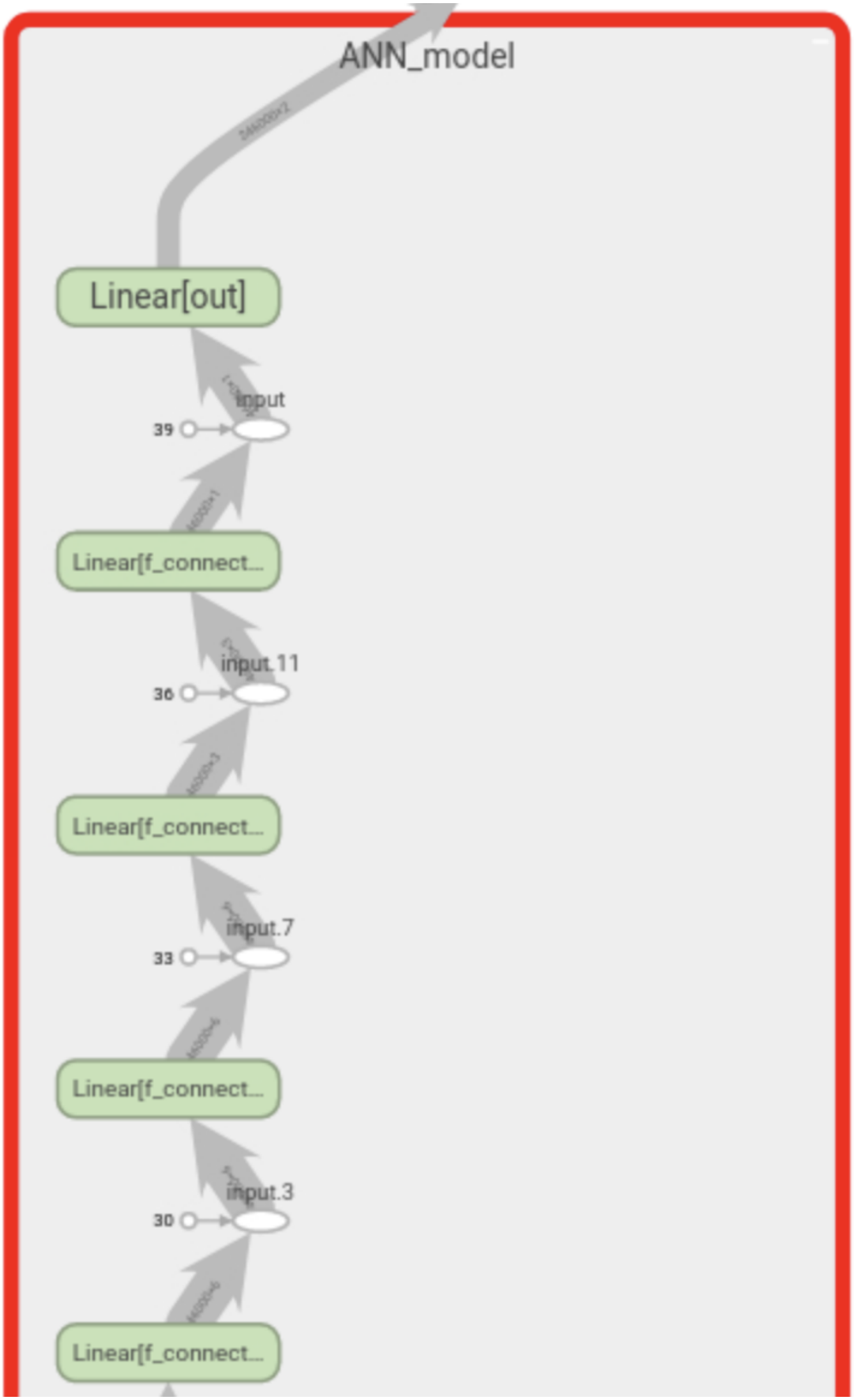
`Approach`

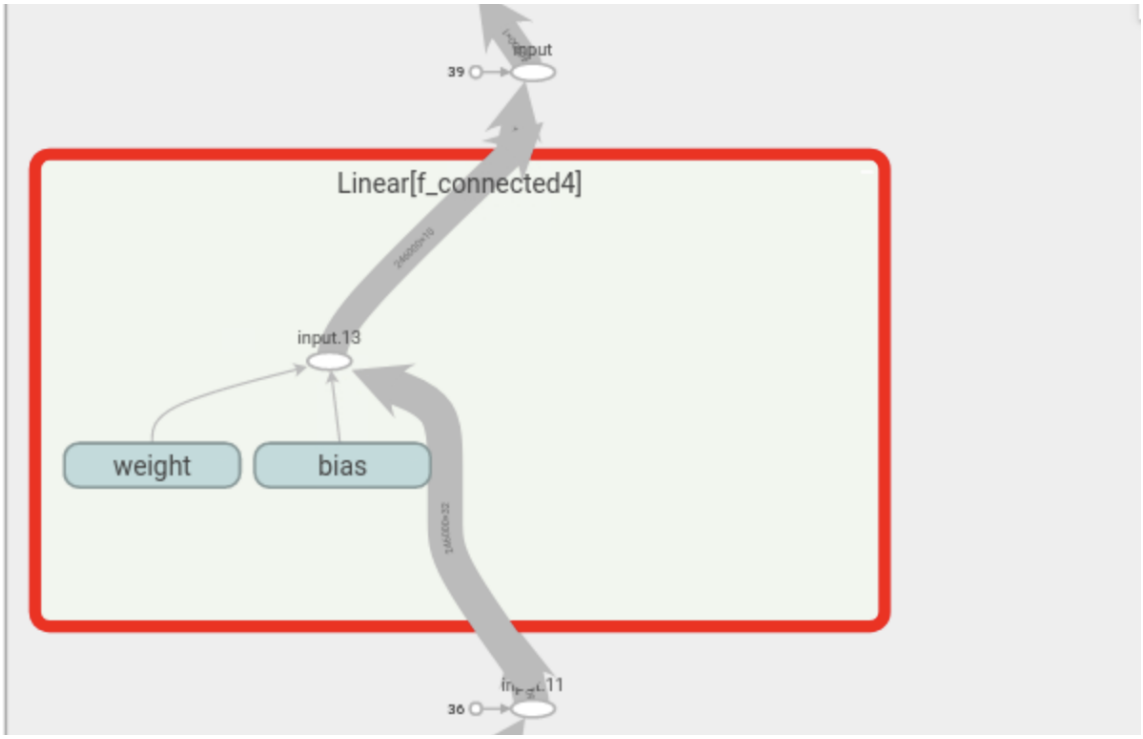
- We used Pytorch to build a MLP model and built an Artificial Neural Network with 4 hidden layers consisting of 128, 64, 32, 10 neurons in each layer respectively with an output layer having 2 neurons
- The activation function that we used was the leaky relu activation function which is an upgrade on the relu activation function having more resilience to the vanishing gradient problem
- The loss function that we used for this model is the cross entropy loss and the optimizer (after experimenting different optimizers) we used for modeling is the Adam optimizer
- The learning rate (after trying different learning rates) that we decided for this particular model was 0.001 and a dropout layer with dropout rate of 0.5 was added in the network to regularize the overfitting of model
- Lastly, the neural network model was trained for 1000 epochs and a softmax function was applied on the final predictions to get the final answer in the form of probabilities

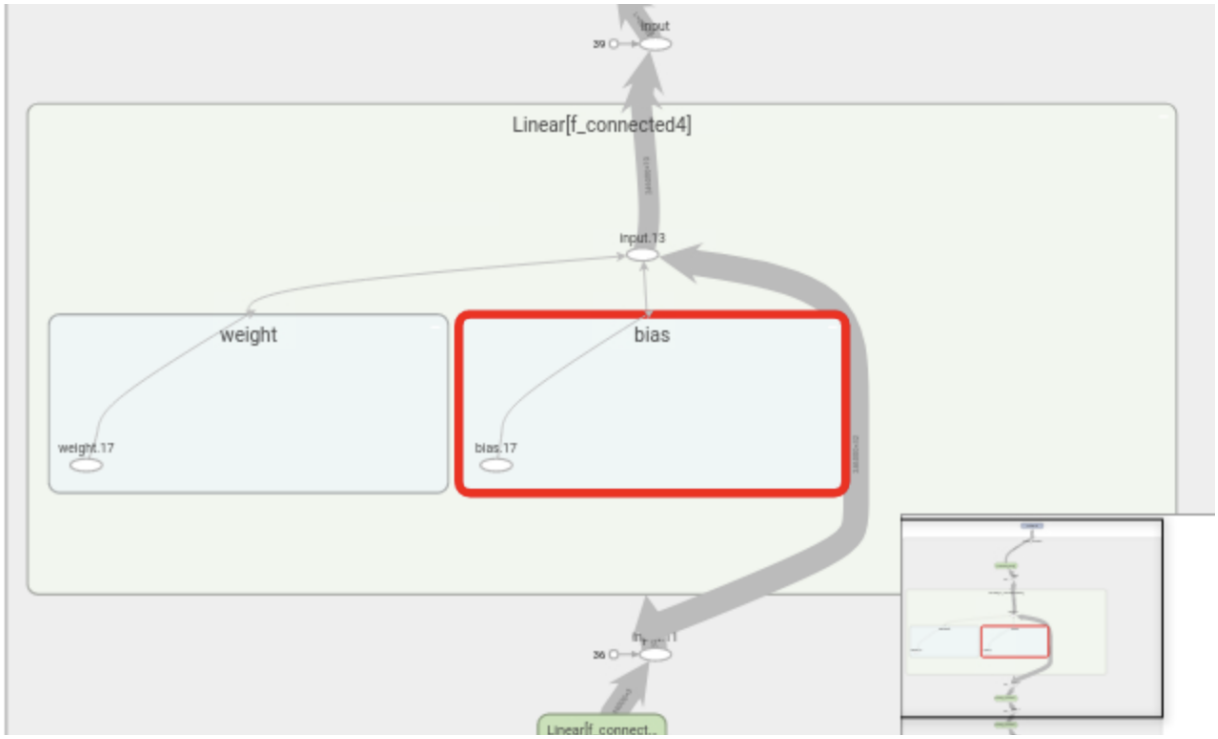
Here are some screenshots from `Tensorboard` that we got for the model

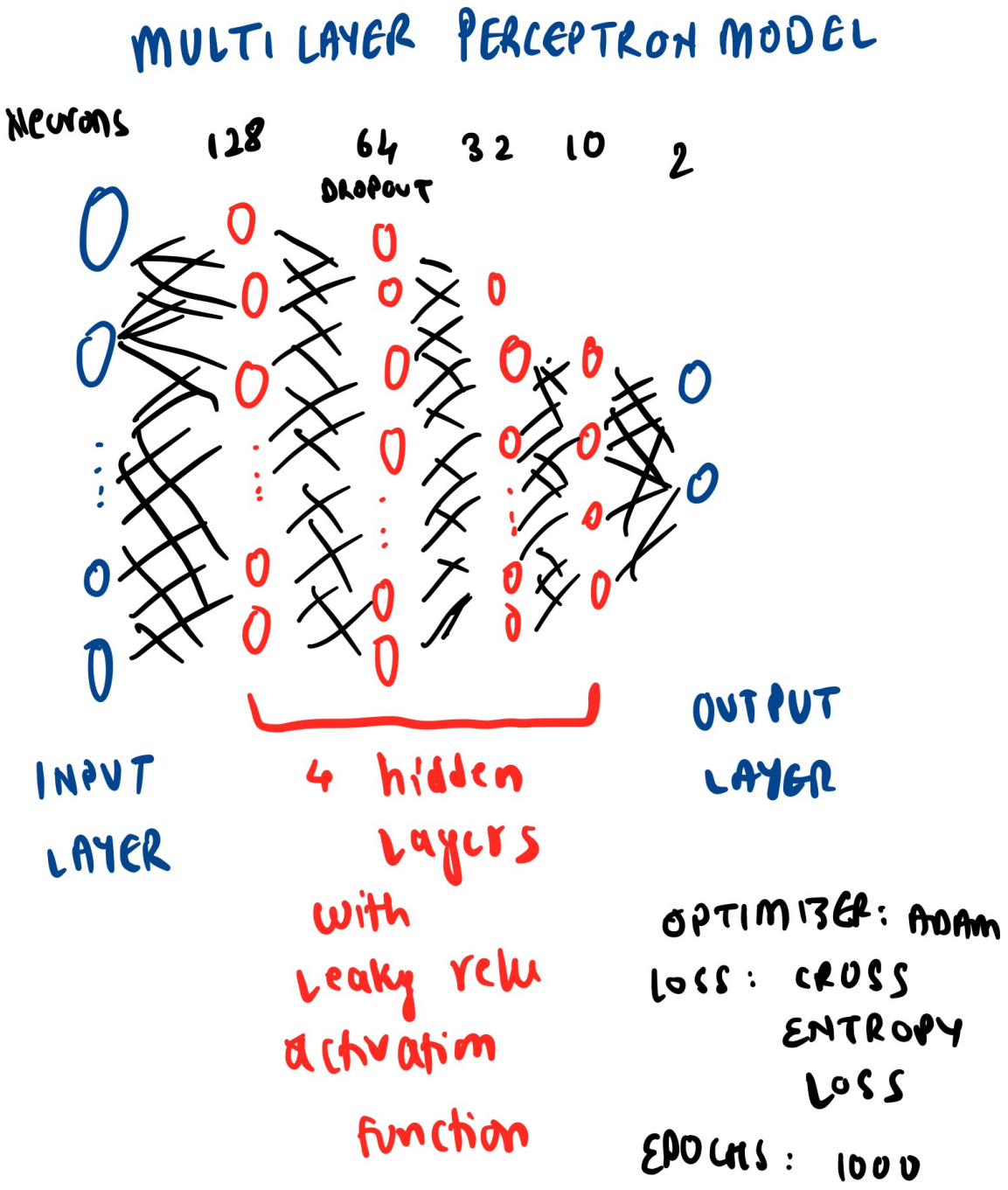
Main Graph




ANN_model
Linear[out]
input
39
Linear[f_connect...
input.11
36
Linear[f_connect...
input.7
33
Linear[f_connect...
input.3
30
Linear[f_connect...


input
39
Linear[f_connected4]
input.13
weight
bias
36


input
39
Linear[f_connected4]
input.13
weight
bias
weight.17
bias.17
36



Phase 2

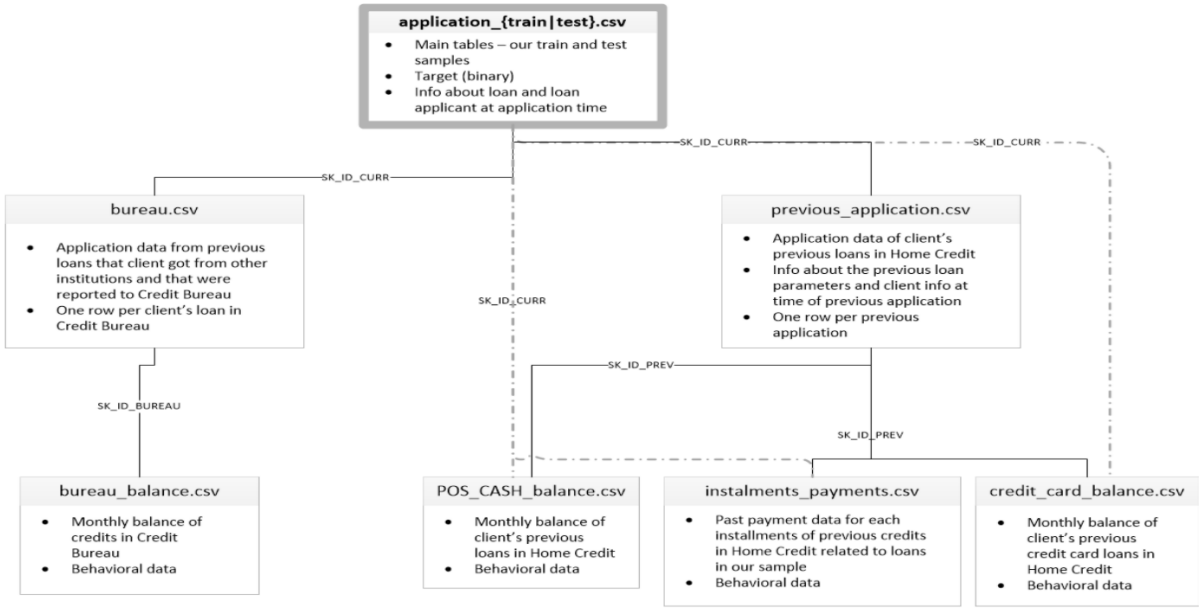
- Use external files like bureau.csv, bureau_balance.csv, credit_card_balance.csv as features in our current training dataset from phase 1, so that there are more features for the model to be trained on and carefully deal with the joining and preprocessing of the data.
- Perform feature engineering by creating new features and use the existing features in a way that helps the model accuracy

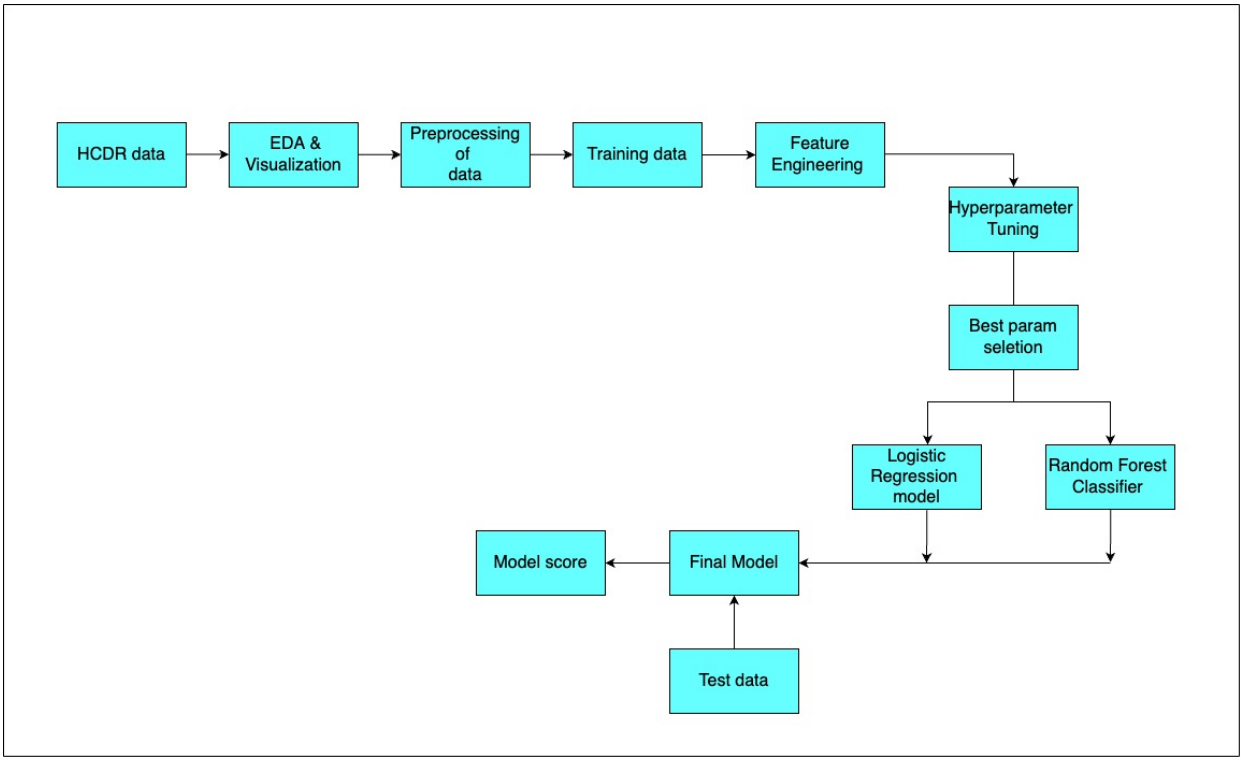
- Perform Hyperparameter tuning for the Logistic Regression and Random Forest Classifier Models to get the best parameters for the models.
- Select the best model out of these models and generate predictions for the test model to get the ROC score for the model.

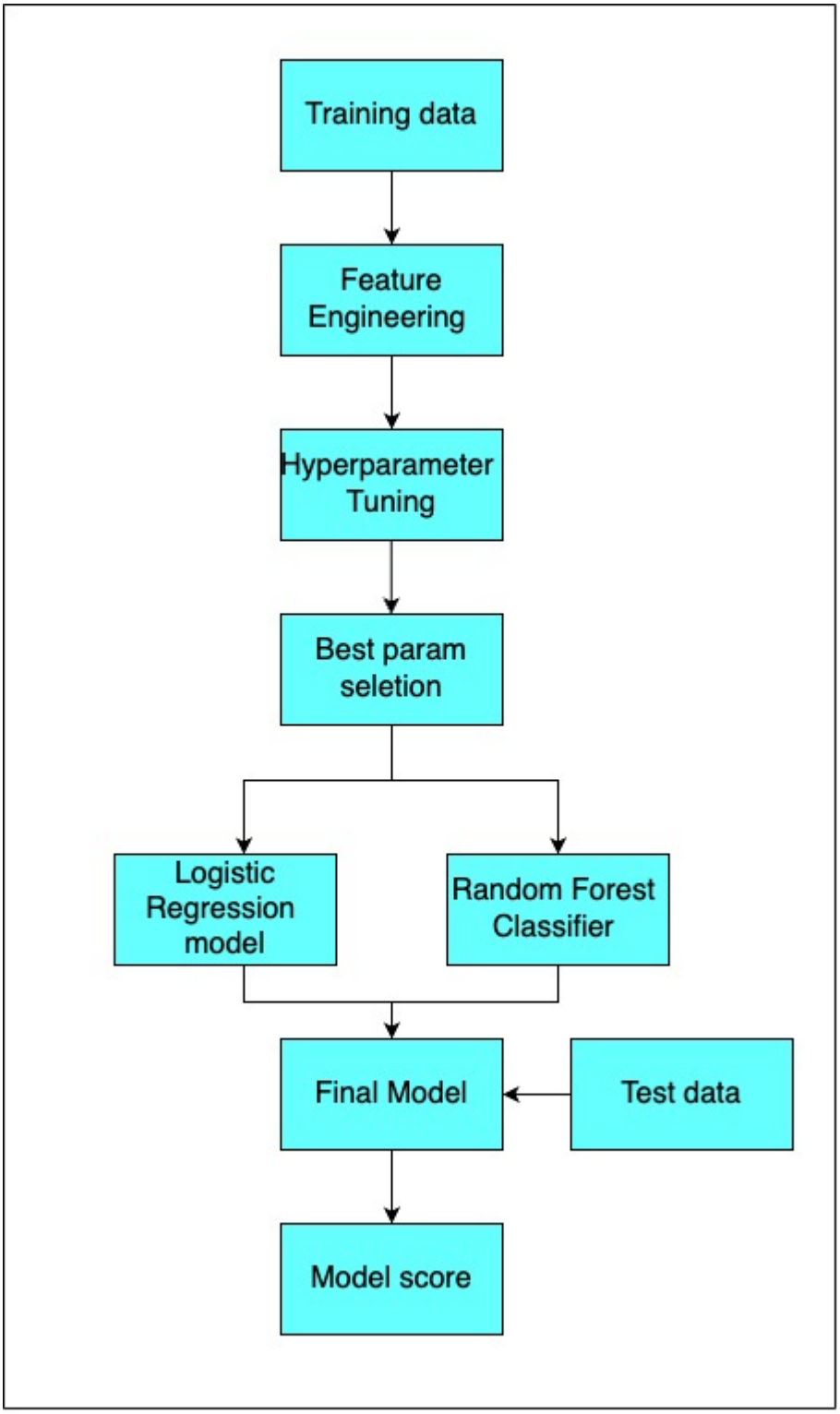
Approach

- We joined the data with the application train and test file. Keeping the imputing strategy intact for the original train and test files, we removed the low correlation categorical columns and imputed the numerical data with their medians for the missing values as we observed that certain values exploded the means.
- We created new features and experimented with the modeling where there were a few newly created features that gave a high correlation with the target variable
- Performed multiple hyperparameter selection loops to get the best values for respective models
- Generated results using the best model.

Diagram : This is a block diagram to understand the workflow of the data.




HCDR data
EDA & Visualization
Preprocessing of data
Training data
Feature Engineering
Hyperparameter Tuning
Best param seletion
Logistic Regression model
Random Forest Classifier
Model score
Final Model
Test data



In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
df = pd.read_csv('application_train.csv')
df_test = pd.read_csv('application_test.csv')
```

In [3]:
```
bureau = pd.read_csv('bureau.csv')
```

```
bureau_balance = pd.read_csv('bureau_balance.csv')
```

In [4]:
```
df_inst = pd.read_csv('installments_payments.csv')
df_pos_c_bal = pd.read_csv('POS_CASH_balance.csv')
df_prev_app=pd.read_csv('previous_application.csv')
credit_card_bal = pd.read_csv('credit_card_balance.csv')
```

In [5]:
```
df_inst = df_inst.select_dtypes(exclude='object')
df_pos_c_bal = df_pos_c_bal.select_dtypes(exclude='object')
df_prev_app = df_prev_app.select_dtypes(exclude='object')
credit_card_bal = credit_card_bal.select_dtypes(exclude='object')
```

In [6]:
```
df_inst= df_inst.groupby('SK_ID_CURR').median()
df_pos_c_bal = df_pos_c_bal.groupby('SK_ID_CURR').median()
credit_card_bal = credit_card_bal.groupby('SK_ID_CURR').median()
df_prev_app = df_prev_app.groupby('SK_ID_CURR').median()
```

In [6]:
```
# df_inst =df_inst.drop(columns=['NUM_INSTALMENT_VERSION','NUM_INSTALMENT_NUMBER'])
# df_pos_c_bal= df_pos_c_bal.drop(columns=['SK_DPD','SK_DPD_DEF','MONTHS_BALANCE'])
# m_pos_ins = pd.merge(left=df_pos_c_bal, right=df_inst, how='inner', left_on=['SK_ID_P
# df_prev_app = df_prev_app.drop(columns=['WEEKDAY_APPR_PROCESS_START','HOUR_APPR_PROCE
#         'NAME_CONTRACT_STATUS', 'DAYS_DECISION', 'NAME_PAYMENT_TYPE',
#         'CODE_REJECT_REASON', 'NAME_TYPE_SUITE', 'NAME_CLIENT_TYPE',
#         'NAME_GOODS_CATEGORY', 'NAME_PORTFOLIO', 'NAME_PRODUCT_TYPE',
#         'CHANNEL_TYPE', 'SELLERPLACE_AREA', 'NAME_SELLER_INDUSTRY',
#         'CNT_PAYMENT', 'NAME_YIELD_GROUP', 'PRODUCT_COMBINATION','DAYS_LAST_DUE_1ST_VE
#         'FLAG_LAST_APPL_PER_CONTRACT','AMT_GOODS_PRICE','NFLAG_INSURED_ON_APPROVAL','RA
#     'NFLAG_LAST_APPL_IN_DAY','NAME_CONTRACT_TYPE','DAYS_FIRST_DRAWING','DAYS_TERMINAT
# m_pos_ins_prev_app = pd.merge(left=m_pos_ins, right=df_prev_app, how='inner', left_on
# m_pos_ins_prev_app = m_pos_ins_prev_app.drop(columns=['RATE_INTEREST_PRIVILEGED','RAT
# m_pos_ins_prev_app= m_pos_ins_prev_app.groupby('SK_ID_CURR').median()
```

In [8]:
```
#from google.colab import drive
#drive.mount('/content/drive')
m_a_bu_bal = pd.merge(left=bureau, right=bureau_balance, how='left', left_on='SK_ID_BUR
```

In [159…
```
m_a_bu_bal = m_a_bu_bal.drop_duplicates()

m_a_bu_bal.shape


m_a_bu_bal = m_a_bu_bal.groupby(['SK_ID_CURR','SK_ID_BUREAU']).min()

new_bb = m_a_bu_bal[['DAYS_CREDIT','DAYS_ENDDATE_FACT','AMT_CREDIT_SUM','DAYS_CREDIT_UP

new_bb = new_bb.reset_index()
# new_bb.drop(columns = ['SK_ID_BUREAU'], inplace = True)

new_bb = new_bb.groupby('SK_ID_CURR').median()

new_bb = new_bb.reset_index()
```

In [20]:
```
#new_bb = pd.read_csv('bureau_and_bureau_balance.csv')
```

In [18]:
```
new_bb.drop(columns = ['Unnamed: 0'],inplace=True)
```

In [19]:
```
new_bb = new_bb.select_dtypes(exclude='object')
```

In [196…
```
# appl_train_num.apply(lambda x: x.fillna(x.median()),axis=0)
```

EXPLORATORY DATA ANALYSIS + FEATURE ENGINEERING AND TRANSFORMERS

Feature Engineering:

Phase1

- The first step to deal with the data was to remove the columns which would act as redundant as it would not contribute in prediction.We explored the data and saw the number of missing values. We removed the columns which had more than 50% of missing values. We checked the columns for the number of 0's distribution and removed the columns which has 90% rows with only values as 0's.
- Further more we divided the data to identify if it is Numerical and categorical. The numerical data was dealt by creating an intermediate Imputer pipleine where the numerical missing values were replaced with the mean of the the data and the missing values in Categorical missing data was dealt by performing OHE(One hot Encoding) and replacing the missing values with Mode of the columns.

Phase2

- We found out the important features for this home credit risk prediction by finding out their correlation with the target variable. These were the steps we followed as an updation to the phase 1 feature engineering:
- We merged the following files to the main appication_train and application_test file - bureau_balance, bureau, installments_payments, POS_CASH_balance, previous_application
- After merging these files with the application_train file we experimented with the added features to generate new features
- The following are the new features that we added

  Feature 1 - AMT_TOTAL_RECEIVABLE/AMT_BALANCE

  Assuming that the total amount receivable is the expected loan amount by the client and the amount receivable is the final amount the individual is going to get,we take in the consideration that the ratio of amount balance receivable by amount will give us the final result.

  Feature 2 : AMT_TOTAL_RECEIVABLE/AMT_RECIVABLE

  Assuming that the total amount receivable is the expected loan amount by the client and the amount receivable is the final amount the individual is going to get,we take in the consideration that the ratio of amount balance receivable by amount will give us the final result.

Feature 3 : AMT_TOTAL_RECEIVABLE/AMT_RECIVABLE_PRINCIPAL

Assuming that the total amount receivable is the expected loan amount by the client and the amount receivable principal is the principal amount the individual is going to get,we take in the consideration that the ratio receivable by principal amount will give us the final result.

Feature 4 - AMT_BALANCE/AMT_RECEIVABLE

Assuming that the amount balance is receivable is the expected loan amount by the client and the amount receivable is the final amount the individual is going to get,we take in the consideration that the ratio of amount balance receivable by amount will give us the final result.

Feature 5 : AMT_BALANCE/AMT_RECEIVABLE_PRINCIPAL

Assuming that the amount balance is the amount left by the client and the amount receivable is the principal amount the individual is going to get,we take in the consideration that the ratio will give us the final result.

Feature 6 : AMT_RECIVABLE/AMT_RECEIVABLE_PRINCIPAL

Assuming that the total amount receivable is the expected loan amount by the client and the amount receivable principal is the principal amount the individual is going to get,we take in the consideration that the ratio of amount receivable receivable by principal amount will give us the final result.

- We had noticed that the features of EXT_SOURCE had a high correlation with the Target, and looked discriminatory between the Defaulters and Non-Defaulters too, hence we created new features from that features as:

Feature 7 - (EXT_SOURCE_1+EXT_SOURCE_2+EXT_SOURCE_3)/AMT_RECIVABLE

Sum of the sources divided by the amount receivable by the customer.

Feature 8 - EXT_SOURCE_1+EXT_SOURCE_2+EXT_SOURCE_3

Sum of the external sources

Feature 9 - EXT_SOURCE_1*EXT_SOURCE_2*EXT_SOURCE_3

Multiplication of the external resources

Feature 10 - EXT_SOURCE_1*2*+*EXT_SOURCE_2*3+EXT_SOURCE_3*4

Weighted average of the external resources.

Feature 11 - Max of (EXT_SOURCE_1,EXT_SOURCE_2,EXT_SOURCE_3)

Maximum of the 3 columns of external resources.

- We along with the features now select all those columns that have a correlation of more than 5% or 0.05 with the target variable.
- After obtaining these columns we dealt with their missing values by replacing them with the median of that column.
- Finally we get the columns which we are going to include in our final application train dataframe.
- Since Naive Bayes did not yield better results than the last phase and had a really low accuracy we discarded it in this phase. This phase we will consider only 2 models namely a Logistic Regression model and a Random Forest Classifier model.
- After getting the final dataset we put this data in the pipeline and the pipeline alongside the step of StandardScaler this time consists of GridSearch cross validation to get the best

parameters in Logistic Regression(ie C) and Random Forest Classifier(ie max_features, max_depth, n_estimators, criterion).

- After the search for best parameters, we fit the training data in the best models and get the accuracy for the training set and the validation set.
- The next step would be to put the merged test data into the best model amongst the two models that we got and submit it to Kaggle.

- Here is a small snippet representing the experiments we performed with the features:

FEATURE NO.	FEATURES	LOGISTIC REGRESSION	RANDOM FOREST CLASSIFIER
F1	Amt_credit/Amt_annuity	Training accuracy: 86.227	Training accuracy : 87.227
F2	Amt_instalment/Amt_credit	Validation accuracy: 84.526	Validation accuracy: 85.526
F3	Amt_payment/ Amt_application	Testing accuracy :66.453	Testing accuracy :68.453
F4	Amt_credit/ Amt_credit_sum		
F5	Amt_instalment/ Days_instalment		

- The above features have a low correlation with the target variable.

FEATURE NO.	FEATURES	LOGISTIC REGRESSION	RANDOM FOREST CLASSIFIER
F1	Amt_credit_sum/ Days_credit	Training Test Accuracy : 87.242	Training Test Accuracy: 88.117
F2	Days_credit_update/ Days_credit	Validation set accuracy : 85.546	Validation set accuracy : 86.126
F3	Amt_installment/ Amt_payment	Testing Test Accuracy :66.453	Testing Test Accuracy :67.412
F4	Amt_instalment/ Amt_annuity		
F5	Days_credit/ Amt_annuity		

- The above features have a low correlation with the target variable.

FEATURE NO.	FEATURES	LOGISTIC REGRESSION	RANDOM FOREST CLASSIFIER
F1	Amt_instalment/ Amt_payment	Training Accuracy: 87.598	Training Accuracy: 89.101
F2	Amt_credit_sum/Days_credit_update	Validation set accuracy: 85.926	Validation set accuracy: 86.140
F3	Days_credit_update/ Amt_instalment	Testing Test Accuracy :66.643	Testing Test Accuracy :67.482
F4	Amt_application/Amt_payment		
F5	Days_credit/ Days_instalment		

FEATURE NO.	FEATURES	LOGISTIC REGRESSION	RANDOM FOREST CLASSIFIER
F1	Amt_instalment/ Amt_payment	Training Accuracy: 87.598	Training Accuracy: 89.101
F2	Amt_credit_sum/Days_credit_update	Validation set accuracy: 85.926	Validation set accuracy: 86.140
F3	Days_credit_update/ Amt_instalment	Testing Test Accuracy :66.643	Testing Test Accuracy :67.482
F4	Amt_application/Amt_payment		
F5	Days_credit/ Days_instalment		

The above features have a low correlation with the target variable.

FEATURE NO.	FEATURES	LOGISTIC REGRESSION	RANDOM FOREST CLASSIFIER
F1	Days_credit/ Days_endate_fact	Training Test Accuracy: 87.340	Training Test Accuracy: 88.220
F2	Days_credit_update/ Months_balance	Validation set accuracy: 85.612	Validation set accuracy: 86.150
F3	Cnt_instalment/ Cnt_instalment_future	Testing Test Accuracy :66.431	Testing Test Accuracy :67.521
F4	Days_first_due/ Days_last_due		
F5	Days_credit/Cnt_instalment		

- Finally, these are the 8 new features we selected with their following accuracies :

FEATURE NO.	FEATURES	LOGISTIC REGRESSION	RANDOM FOREST CLASSIFIER
F1	AMT_TOTAL_RECEIVABLE/ AMT_BALANCE	Training Accuracy : 91.928	Training Accuracy : 99.995
F2	AMT_TOTAL_RECEIVABLE/ AMT_RECIVABLE	Validation accuracy : 91.893	Validation accuracy : 91.921
F3	AMT_TOTAL_RECEIVABLE/ AMT_RECIVABLE_PRINCIPAL	Testing Accuracy : 72.388	Testing accuracy :70.039

FEATURE NO.	FEATURES	LOGISTIC REGRESSION	RANDOM FOREST CLASSIFIER
F4	AMT_BALANCE/ AMT_RECEIVABLE	Training Accuracy : 91.928	Training Accuracy : 99.995
F5	AMT_BALANCE/ AMT_RECEIVABLE_PRINCIPAL	Validation accuracy : 91.893	Validation accuracy : 91.921
F6	AMT_RECIVABLE/ AMT_RECEIVABLE_PRINCIPAL	Testing Accuracy : 72.388	Testing accuracy :70.039

FEATURE NO.	FEATURES	LOGISTIC REGRESSION	RANDOM FOREST CLASSIFIER
F7	(EXT_SOURCE_1+EXT_SOURCE_2+EXT_SOURCE_3)/ AMT_RECIVABLE	Training Accuracy : 91.928	Training Accuracy : 99.995
F8	EXT_SOURCE_1+EXT_SOURCE_2+EXT_SOURCE_3	Validation accuracy : 91.893	Validation accuracy : 91.921
F9	EXT_SOURCE_1*EXT_SOURCE_2*EXT_SOURCE_3	Testing Accuracy : 72.388	Testing accuracy :70.039
F10	EXT_SOURCE_1*2+EXT_SOURCE_2*3+EXT_SOURCE_3*4		
F11	Max of (EXT_SOURCE_1,EXT_SOURCE_2,EXT_SOURCE_3)		

In [31]:
```
df.head()
```

Out[31]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY
0	100002	1	Cash loans	M	N	Y
1	100003	0	Cash loans	F	N	N
2	100004	0	Revolving loans	M	Y	Y
3	100006	0	Cash loans	F	N	Y
4	100007	0	Cash loans	M	N	Y

5 rows × 122 columns

In [32]:
```
df.describe()
```

Out[32]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUIT
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.0000
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.5739
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.7373
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.5000

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUI
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.0000
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.0000
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.0000
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.5000

8 rows × 106 columns

In [33]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
```

In [34]:
```
df_test.head()
```

Out[34]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHI
0	100001	Cash loans	F	N	Y	
1	100005	Cash loans	M	N	Y	
2	100013	Cash loans	M	Y	Y	
3	100028	Cash loans	F	N	Y	
4	100038	Cash loans	M	Y	N	

5 rows × 121 columns

In [35]:
```
df_test.describe()
```

Out[35]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOOD
count	48744.000000	48744.000000	4.874400e+04	4.874400e+04	48720.000000	4.874
mean	277796.676350	0.397054	1.784318e+05	5.167404e+05	29426.240209	4.626
std	103169.547296	0.709047	1.015226e+05	3.653970e+05	16016.368315	3.367
min	100001.000000	0.000000	2.694150e+04	4.500000e+04	2295.000000	4.500
25%	188557.750000	0.000000	1.125000e+05	2.606400e+05	17973.000000	2.250
50%	277549.000000	0.000000	1.575000e+05	4.500000e+05	26199.000000	3.960
75%	367555.500000	1.000000	2.250000e+05	6.750000e+05	37390.500000	6.300
max	456250.000000	20.000000	4.410000e+06	2.245500e+06	180576.000000	2.245

8 rows × 105 columns

In [36]:
```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48744 entries, 0 to 48743
Columns: 121 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(40), object(16)
memory usage: 45.0+ MB
```

Declaring some functions

In [37]:
```
def missing_per(df):
    missing_vals = df.isnull().sum(axis=0)*100/len(df)
    return missing_vals.sort_values(ascending=False)
```

In [38]:
```
def remove_cols(df,rem_cols):
    df.drop(columns = rem_cols,inplace=True)
    return df
```

In [39]:
```
# p = missing_per(df)
# cols_to_remove = p.reset_index()
# cols_to_remove.columns = ['col_name','flag']
# cols_to_remove_ = cols_to_remove[cols_to_remove['flag']>50]
# rem_cols = list(cols_to_remove_['col_name'])
```

In [40]:
```
def get_unique_in_a_column(df,n):
    u = []
    for cols in df.columns:
        if cols=='TARGET':
            continue
        u.append([cols,df[cols].nunique()])
    df_temp = pd.DataFrame(u,columns=['col_name','unique_values'])
    return pd.DataFrame(df_temp[df_temp['unique_values']<n])
```

In [41]:
```
def seg_num_cat(df_phase3):
    dfp3_numerical= df_phase3.select_dtypes(exclude='object')
    dfp3_numerical['TARGET'] = df_phase3['TARGET']
    dfp3_categorical= df.select_dtypes(include='object')
    return dfp3_numerical,dfp3_categorical
```

In [42]:
```
# def mean_mode(df_phase3):
#   cols_req_work=[]
#   mean_cols = ['YEARS_BEGINEXPLUATATION_AVG','YEARS_BEGINEXPLUATATION_MEDI','YEARS_BE
#   mode_cols = ['NAME_TYPE_SUITE','DEF_60_CNT_SOCIAL_CIRCLE', 'OBS_30_CNT_SOCIAL_CIRCL
#   print(df_phase3['NAME_TYPE_SUITE'].mode()[0])
#   for i in mean_cols:
#     column_means = df_phase3[i].mean()
#     df_phase3[i] = df_phase3[i].fillna(column_means)
#   for i in mode_cols:
```

```
#     col_mode = df_phase3[i].mode()[0]
#     df_phase3[i] = df_phase3[i].fillna(col_mode)
#   return df_phase3
```

In [43]:
```
def zeros_df(df,per):
    dfp3_n0s = pd.DataFrame()
    columns = []
    percentage =[]
    for col in df.columns:
        if col == 'TARGET':
          continue
        count = (df[col] == 0).sum()
        columns.append(col)
        percentage.append(count/len(df[col]))
    dfp3_n0s['Column'] = columns
    dfp3_n0s['Percentage'] = percentage
    per = per/100
    dfp3_n0s = dfp3_n0s[dfp3_n0s['Percentage']>per]

    return dfp3_n0s
```

In [44]:
```
def to_consider_as_cat(df,thres_val):
    return list(get_unique_in_a_column(df,thres_val)['col_name'])
```

In [45]:
```
def corr_target(df,cor):
    cor_matrix = df.corr()['TARGET'].sort_values(key=abs,ascending=False).reset_index()
    cor_matrix.columns = ['col_name','Correlation']
    column_after_corr_filter = cor_matrix[abs(cor_matrix['Correlation'])>cor]
    return column_after_corr_filter
```

In [46]:
```
def missing(df,n):
    new_df = missing_per(df).reset_index()
    categ_ = []
    new_df.columns = ['index','flag']
    fin_df = []
    for row in new_df.itertuples():
        try:
          fin_df.append([row.index,row.flag,df[row.index].median(),df[row.index].mean()
        except:
          fin_df.append([row.index,row.flag,df[row.index].mode(),'NA',df[row.index].nun
    cols = ['col_name','percentage_missing','median/Mode','mean','no_of_unique_values']
    temp = pd.DataFrame(fin_df,columns=cols)
    return  temp[temp['percentage_missing']>n]
```

Columns with more than 90% zero values in them

In [47]:
```
more_than_90_zero = zeros_df(df,90)
more_than_90_zero
```

Out[47]:

	Column	Percentage
26	FLAG_EMAIL	0.943280

	Column	Percentage
33	REG_REGION_NOT_LIVE_REGION	0.984856
34	REG_REGION_NOT_WORK_REGION	0.949231
35	LIVE_REGION_NOT_WORK_REGION	0.959341
36	REG_CITY_NOT_LIVE_CITY	0.921827
93	DEF_60_CNT_SOCIAL_CIRCLE	0.912881
95	FLAG_DOCUMENT_2	0.999958
97	FLAG_DOCUMENT_4	0.999919
98	FLAG_DOCUMENT_5	0.984885
99	FLAG_DOCUMENT_6	0.911945
100	FLAG_DOCUMENT_7	0.999808
101	FLAG_DOCUMENT_8	0.918624
102	FLAG_DOCUMENT_9	0.996104
103	FLAG_DOCUMENT_10	0.999977
104	FLAG_DOCUMENT_11	0.996088
105	FLAG_DOCUMENT_12	0.999993
106	FLAG_DOCUMENT_13	0.996475
107	FLAG_DOCUMENT_14	0.997064
108	FLAG_DOCUMENT_15	0.998790
109	FLAG_DOCUMENT_16	0.990072
110	FLAG_DOCUMENT_17	0.999733
111	FLAG_DOCUMENT_18	0.991870
112	FLAG_DOCUMENT_19	0.999405
113	FLAG_DOCUMENT_20	0.999493
114	FLAG_DOCUMENT_21	0.999665

Dropping the above columns from the dataset

In [48]:
```
df.drop(columns = more_than_90_zero['Column'],inplace = True)
```

Columns in training set having more than 30% of missing data, along with their median/mode and unique values in the column

In [49]:
```
missing(df,30)
```

Out[49]:

	col_name	percentage_missing	median/Mode	mean	no_of_unique_values
0	COMMONAREA_MEDI	69.872297	0.0208	0.044595	3202

	col_name	percentage_missing	median/Mode	mean	no_of_unique_values
1	COMMONAREA_MODE	69.872297	0.019	0.042553	3128
2	COMMONAREA_AVG	69.872297	0.0211	0.044621	3181
3	NONLIVINGAPARTMENTS_MEDI	69.432963	0.0	0.008651	214
4	NONLIVINGAPARTMENTS_AVG	69.432963	0.0	0.008809	386
5	NONLIVINGAPARTMENTS_MODE	69.432963	0.0	0.008076	167
6	FONDKAPREMONT_MODE	68.386172	0 reg oper account dtype: object	NA	4
7	LIVINGAPARTMENTS_MODE	68.354953	0.0771	0.105645	736
8	LIVINGAPARTMENTS_AVG	68.354953	0.0756	0.100775	1868
9	LIVINGAPARTMENTS_MEDI	68.354953	0.0761	0.101954	1097
10	FLOORSMIN_MEDI	67.848630	0.2083	0.231625	47
11	FLOORSMIN_AVG	67.848630	0.2083	0.231894	305
12	FLOORSMIN_MODE	67.848630	0.2083	0.228058	25
13	YEARS_BUILD_MODE	66.497784	0.7648	0.759637	154
14	YEARS_BUILD_MEDI	66.497784	0.7585	0.755746	151
15	YEARS_BUILD_AVG	66.497784	0.7552	0.752471	149
16	OWN_CAR_AGE	65.990810	9.0	12.061091	62
17	LANDAREA_MODE	59.376738	0.0458	0.064958	3563
18	LANDAREA_AVG	59.376738	0.0481	0.066333	3527
19	LANDAREA_MEDI	59.376738	0.0487	0.067169	3560
20	BASEMENTAREA_MEDI	58.515956	0.0758	0.087955	3772
21	BASEMENTAREA_AVG	58.515956	0.0763	0.088442	3780
22	BASEMENTAREA_MODE	58.515956	0.0746	0.087543	3841
23	EXT_SOURCE_1	56.381073	0.505998	0.50213	114584
24	NONLIVINGAREA_AVG	55.179164	0.0036	0.028358	3290
25	NONLIVINGAREA_MODE	55.179164	0.0011	0.027022	3327
26	NONLIVINGAREA_MEDI	55.179164	0.0031	0.028236	3323
27	ELEVATORS_AVG	53.295980	0.0	0.078942	257
28	ELEVATORS_MEDI	53.295980	0.0	0.078078	46
29	ELEVATORS_MODE	53.295980	0.0	0.07449	26
30	WALLSMATERIAL_MODE	50.840783	0 Panel dtype: object	NA	7
31	APARTMENTS_AVG	50.749729	0.0876	0.11744	2339
32	APARTMENTS_MEDI	50.749729	0.0864	0.11785	1148

	col_name	percentage_missing	median/Mode	mean	no_of_unique_values
33	APARTMENTS_MODE	50.749729	0.084	0.114231	760
34	ENTRANCES_MEDI	50.348768	0.1379	0.149213	46
35	ENTRANCES_AVG	50.348768	0.1379	0.149725	285
36	ENTRANCES_MODE	50.348768	0.1379	0.145193	30
37	LIVINGAREA_MEDI	50.193326	0.0749	0.108607	5281
38	LIVINGAREA_MODE	50.193326	0.0731	0.105975	5301
39	LIVINGAREA_AVG	50.193326	0.0745	0.107399	5199
40	HOUSETYPE_MODE	50.176091	0 block of flats dtype: object	NA	3
41	FLOORSMAX_MODE	49.760822	0.1667	0.222315	25
42	FLOORSMAX_MEDI	49.760822	0.1667	0.225897	49
43	FLOORSMAX_AVG	49.760822	0.1667	0.226282	403
44	YEARS_BEGINEXPLUATATION_AVG	48.781019	0.9816	0.977735	285
45	YEARS_BEGINEXPLUATATION_MEDI	48.781019	0.9816	0.977752	245
46	YEARS_BEGINEXPLUATATION_MODE	48.781019	0.9816	0.977065	221
47	TOTALAREA_MODE	48.268517	0.0688	0.102547	5116
48	EMERGENCYSTATE_MODE	47.398304	0 No dtype: object	NA	2
49	OCCUPATION_TYPE	31.345545	0 Laborers dtype: object	NA	18

Segregating the Dataset in numerical and categorical dataframes

In [50]:
```
df_num, df_cat = seg_num_cat(df)
```

```
/N/u/tankulk/Carbonate/miniconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports until
```

In [51]:
```
df_num.describe().T
```

Out[51]:

	count	mean	std	min	25%	50%
SK_ID_CURR	307511.0	278180.518577	102790.175348	100002.0	189145.5	278202.0
TARGET	307511.0	0.080729	0.272419	0.0	0.0	0.0
CNT_CHILDREN	307511.0	0.417052	0.722121	0.0	0.0	0.0

	count	mean	std	min	25%	50%
AMT_INCOME_TOTAL	307511.0	168797.919297	237123.146279	25650.0	112500.0	147150.0
AMT_CREDIT	307511.0	599025.999706	402490.776996	45000.0	270000.0	513531.0
...	...	...	...	...	...	...
AMT_REQ_CREDIT_BUREAU_DAY	265992.0	0.007000	0.110757	0.0	0.0	0.0
AMT_REQ_CREDIT_BUREAU_WEEK	265992.0	0.034362	0.204685	0.0	0.0	0.0
AMT_REQ_CREDIT_BUREAU_MON	265992.0	0.267395	0.916002	0.0	0.0	0.0
AMT_REQ_CREDIT_BUREAU_QRT	265992.0	0.265474	0.794056	0.0	0.0	0.0
AMT_REQ_CREDIT_BUREAU_YEAR	265992.0	1.899974	1.869295	0.0	0.0	1.0

81 rows × 8 columns

In [52]:
```
df_cat.describe().T
```

Out[52]:

	count	unique	top	freq
NAME_CONTRACT_TYPE	307511	2	Cash loans	278232
CODE_GENDER	307511	3	F	202448
FLAG_OWN_CAR	307511	2	N	202924
FLAG_OWN_REALTY	307511	2	Y	213312
NAME_TYPE_SUITE	306219	7	Unaccompanied	248526
NAME_INCOME_TYPE	307511	8	Working	158774
NAME_EDUCATION_TYPE	307511	5	Secondary / secondary special	218391
NAME_FAMILY_STATUS	307511	6	Married	196432
NAME_HOUSING_TYPE	307511	6	House / apartment	272868
OCCUPATION_TYPE	211120	18	Laborers	55186
WEEKDAY_APPR_PROCESS_START	307511	7	TUESDAY	53901
ORGANIZATION_TYPE	307511	58	Business Entity Type 3	67992
FONDKAPREMONT_MODE	97216	4	reg oper account	73830
HOUSETYPE_MODE	153214	3	block of flats	150503
WALLSMATERIAL_MODE	151170	7	Panel	66040
EMERGENCYSTATE_MODE	161756	2	No	159428

Numerical Columns and their correlation with the TARGET column in descending order

In [53]:
```
corr_target(df_num,0.00)
```

Out[53]:

	col_name	Correlation
0	TARGET	1.000000
1	EXT_SOURCE_3	-0.178919
2	EXT_SOURCE_2	-0.160472
3	EXT_SOURCE_1	-0.155317
4	DAYS_BIRTH	0.078239
...	...	...
76	NONLIVINGAPARTMENTS_MODE	-0.001557
77	AMT_REQ_CREDIT_BUREAU_HOUR	0.000930
78	AMT_REQ_CREDIT_BUREAU_WEEK	0.000788
79	FLAG_MOBIL	0.000534
80	FLAG_CONT_MOBILE	0.000370

81 rows × 2 columns

The Columns named "NAME_FAMILY_STATUS, CODE_GENDER, NAME_INCOME_TYPE"does not have values 'Unknown','XNA' and 'Maternity Leave' in the test dataset thus these rows are removed from the training dataset and there are a total of 11 rows that are removed.

In [54]:
```
df =df[df['NAME_FAMILY_STATUS']!='Unknown']
df =df[df['CODE_GENDER']!='XNA']
df =df[df['NAME_INCOME_TYPE']!='Maternity leave']
```

Considering Columns that have more than 2% correlation with the TARGET variable

In [55]:
```
temp_df = corr_target(df_num,0.02)
# temp_df['SK_ID_CURR'] = df_num['SK_ID_CURR']
temp_df
```

Out[55]:

	col_name	Correlation
0	TARGET	1.000000
1	EXT_SOURCE_3	-0.178919
2	EXT_SOURCE_2	-0.160472
3	EXT_SOURCE_1	-0.155317
4	DAYS_BIRTH	0.078239
5	REGION_RATING_CLIENT_W_CITY	0.060893
6	REGION_RATING_CLIENT	0.058899
7	DAYS_LAST_PHONE_CHANGE	0.055218
8	DAYS_ID_PUBLISH	0.051457
9	REG_CITY_NOT_WORK_CITY	0.050994

	col_name	Correlation
10	FLAG_EMP_PHONE	0.045982
11	DAYS_EMPLOYED	-0.044932
12	FLAG_DOCUMENT_3	0.044346
13	FLOORSMAX_AVG	-0.044003
14	FLOORSMAX_MEDI	-0.043768
15	FLOORSMAX_MODE	-0.043226
16	DAYS_REGISTRATION	0.041975
17	AMT_GOODS_PRICE	-0.039645
18	OWN_CAR_AGE	0.037612
19	REGION_POPULATION_RELATIVE	-0.037227
20	ELEVATORS_AVG	-0.034199
21	ELEVATORS_MEDI	-0.033863
22	FLOORSMIN_AVG	-0.033614
23	FLOORSMIN_MEDI	-0.033394
24	LIVINGAREA_AVG	-0.032997
25	LIVINGAREA_MEDI	-0.032739
26	FLOORSMIN_MODE	-0.032698
27	TOTALAREA_MODE	-0.032596
28	LIVE_CITY_NOT_WORK_CITY	0.032518
29	DEF_30_CNT_SOCIAL_CIRCLE	0.032248
30	ELEVATORS_MODE	-0.032131
31	LIVINGAREA_MODE	-0.030685
32	AMT_CREDIT	-0.030369
33	APARTMENTS_AVG	-0.029498
34	APARTMENTS_MEDI	-0.029184
35	FLAG_WORK_PHONE	0.028524
36	APARTMENTS_MODE	-0.027284
37	LIVINGAPARTMENTS_AVG	-0.025031
38	LIVINGAPARTMENTS_MEDI	-0.024621
39	HOUR_APPR_PROCESS_START	-0.024166
40	FLAG_PHONE	-0.023806
41	LIVINGAPARTMENTS_MODE	-0.023393
42	BASEMENTAREA_AVG	-0.022746

	col_name	Correlation
43	YEARS_BUILD_MEDI	-0.022326
44	YEARS_BUILD_AVG	-0.022149
45	BASEMENTAREA_MEDI	-0.022081
46	YEARS_BUILD_MODE	-0.022068

Out of these columns we first need to deal with missing values of the columns. Thus we get all columns which have missing values

In [56]:
```
temp_df_missing = missing(df[temp_df['col_name']],0)
```

Here in cols_with_no_missing we keep columns not having missing values

In [57]:
```
cols_with_no_missing = list(set(temp_df.col_name).difference(set(temp_df_missing.col_na
```

We set a threshold of 10 for unique values in a column, if it is 10 or less we do not consider them as numerical but discrete and replace the missing values with the mode of the column

In [58]:
```
less_than_10 = temp_df_missing[temp_df_missing['no_of_unique_values']<=10]
temp_df_missing = temp_df_missing[temp_df_missing['no_of_unique_values']>10]
```

In [59]:
```
for i in less_than_10.col_name:
    df_num[i] = df_num[i].fillna(df_num[i].mode()[0])

for i in less_than_10.col_name:
    df_test[i] = df_test[i].fillna(df_test[i].mode()[0])
```

```
/N/u/tankulk/Carbonate/miniconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
```

Finally, we get all the column names for the numerical columns that we will consider in our modeling and also imputation

In [60]:
```
final_cols_in_num = list(less_than_10.col_name) + list(temp_df_missing.col_name) +cols_
```

This gives us the final numerical dataframe

In [61]:
```
df_num_temp = df_num
df_num = df_num[final_cols_in_num]
df_num.insert(0,"SK_ID_CURR",df_num_temp['SK_ID_CURR'])
df_num.head()
```

Out[61]:

	SK_ID_CURR	DEF_30_CNT_SOCIAL_CIRCLE	LIVINGAPARTMENTS_MODE	LIVINGAPARTMENTS_MEDI	LIV
0	100002	2.0	0.022	0.0205	
1	100003	0.0	0.079	0.0787	
2	100004	0.0	NaN	NaN	
3	100006	0.0	NaN	NaN	
4	100007	0.0	NaN	NaN	

5 rows × 48 columns

Here, we get the missing value of columns for the categorical columns, we see that out of 16 columns we get 6 columns that have missing values

In [62]:
```
df_temp = missing(df_cat,0)
df_temp
```

Out[62]:

	col_name	percentage_missing	median/Mode	mean	no_of_unique_values
0	FONDKAPREMONT_MODE	68.386172	0 reg oper account dtype: object	NA	4
1	WALLSMATERIAL_MODE	50.840783	0 Panel dtype: object	NA	7
2	HOUSETYPE_MODE	50.176091	0 block of flats dtype: object	NA	3
3	EMERGENCYSTATE_MODE	47.398304	0 No dtype: object	NA	2
4	OCCUPATION_TYPE	31.345545	0 Laborers dtype: object	NA	18
5	NAME_TYPE_SUITE	0.420148	0 Unaccompanied dtype: object	NA	7

Getting the counts of each category in these columns

Here we observe that for columns :

1)FONDKAPREMONT_MODE there is 68% missing data which makes it not ideal to impute missing values with the mode

2)WALLSMATERIAL_MODE, here the difference between the first and the second most occuring values is not much thus imputation with Mode is ambiguous

3)OCCUPATION_TYPE, this is a column that has 31% missing value and again there can not be one specific value that can be decided to impute the missing data with

Thus we remove these columns from the categorical part of the dataframe

In [63]:
```
cat_cols_to_rem = ['FONDKAPREMONT_MODE','WALLSMATERIAL_MODE','OCCUPATION_TYPE']
```

In [64]:

```
df_cat.drop(columns = cat_cols_to_rem,inplace=True)
```

```
/N/u/tankulk/Carbonate/miniconda3/lib/python3.7/site-packages/pandas/core/frame.py:4913:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  errors=errors,
```

VISUAL EXPLORATORY DATA ANALYSIS

Visualizing the categorical columns to understand the data more efficiently

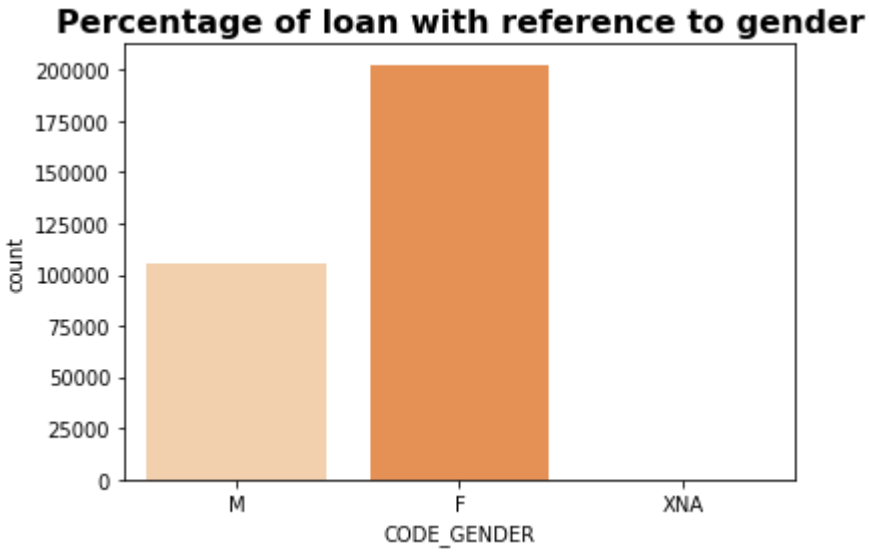
In [65]:
```
def col(cat):
      plt.figure(figsize=(10,10))
      plt.title("Loan Default with respect to "+cat,fontweight='bold' , fontsize =16)
      sns.countplot(x=df[cat],hue='TARGET',data=df, palette = 'Blues')
      plt.xticks(rotation=90)
```

How is the distribution of of loan according to gender?

In [66]:
```
print(df_cat['CODE_GENDER'].value_counts())
sns.countplot(df_cat['CODE_GENDER'], palette = 'Oranges')
plt.title("Percentage of loan with reference to gender", fontweight = 'bold', fontsize
```

```
/N/u/tankulk/Carbonate/miniconda3/lib/python3.7/site-packages/seaborn/_decorators.py:43:
FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments without an exp
licit keyword will result in an error or misinterpretation.
  FutureWarning
F      202448
M      105059
XNA         4
Name: CODE_GENDER, dtype: int64
```

Out[66]:
```
Text(0.5, 1.0, 'Percentage of loan with reference to gender')
```



Inference: The number of female borrowing the loan and who haven't paid is comparatively higher than men.

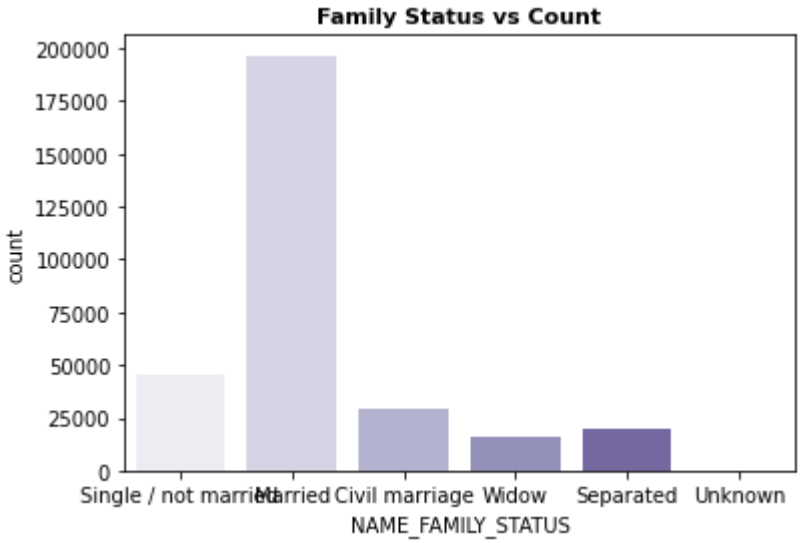
What is the marital status of client?

In [67]:
```
print(df_cat['NAME_FAMILY_STATUS'].value_counts())
sns.countplot(df_cat['NAME_FAMILY_STATUS'], palette = 'Purples')
plt.title("Family Status vs Count", fontweight = 'bold', fontsize = 11)
```

```
Married                 196432
Single / not married     45444
Civil marriage           29775
Separated                19770
Widow                    16088
Unknown                      2
Name: NAME_FAMILY_STATUS, dtype: int64
```

```
/N/u/tankulk/Carbonate/miniconda3/lib/python3.7/site-packages/seaborn/_decorators.py:43:
FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments without an exp
licit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[67]: Text(0.5, 1.0, 'Family Status vs Count')



Inference: The majority of client who are Married have paid the least loan amount while the status of unknown is negligible.

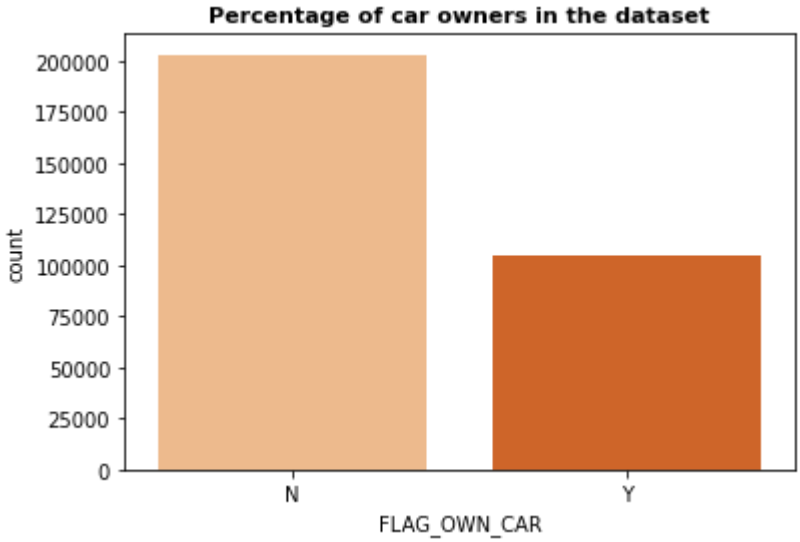
How many percent of client own a car?

In [68]:
```
print(df_cat['FLAG_OWN_CAR'].value_counts())
sns.countplot(df_cat['FLAG_OWN_CAR'], palette = 'Oranges')
plt.title("Percentage of car owners in the dataset", fontweight = 'bold', fontsize = 11
```

```
N    202924
Y    104587
Name: FLAG_OWN_CAR, dtype: int64
```

```
/N/u/tankulk/Carbonate/miniconda3/lib/python3.7/site-packages/seaborn/_decorators.py:43:
FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments without an exp
licit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[68]: Text(0.5, 1.0, 'Percentage of car owners in the dataset')



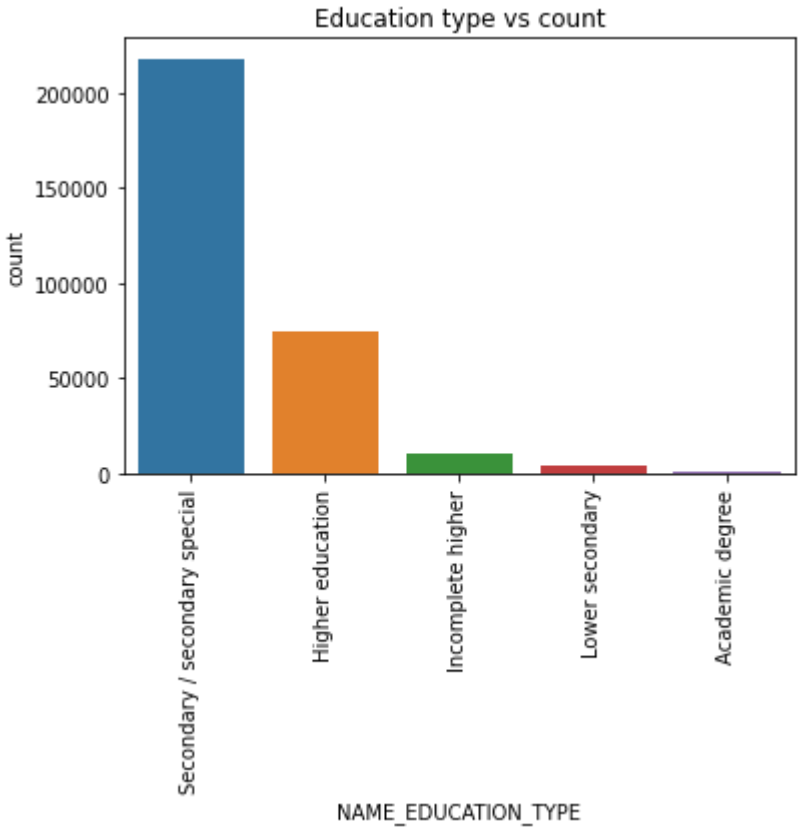
Inference: About 50 % of people own's a car, but there's majority of client (more than 50%) who doesn't possess a car and most of them are likely who haven't paid the loan.

What type of educational background does the clients have?

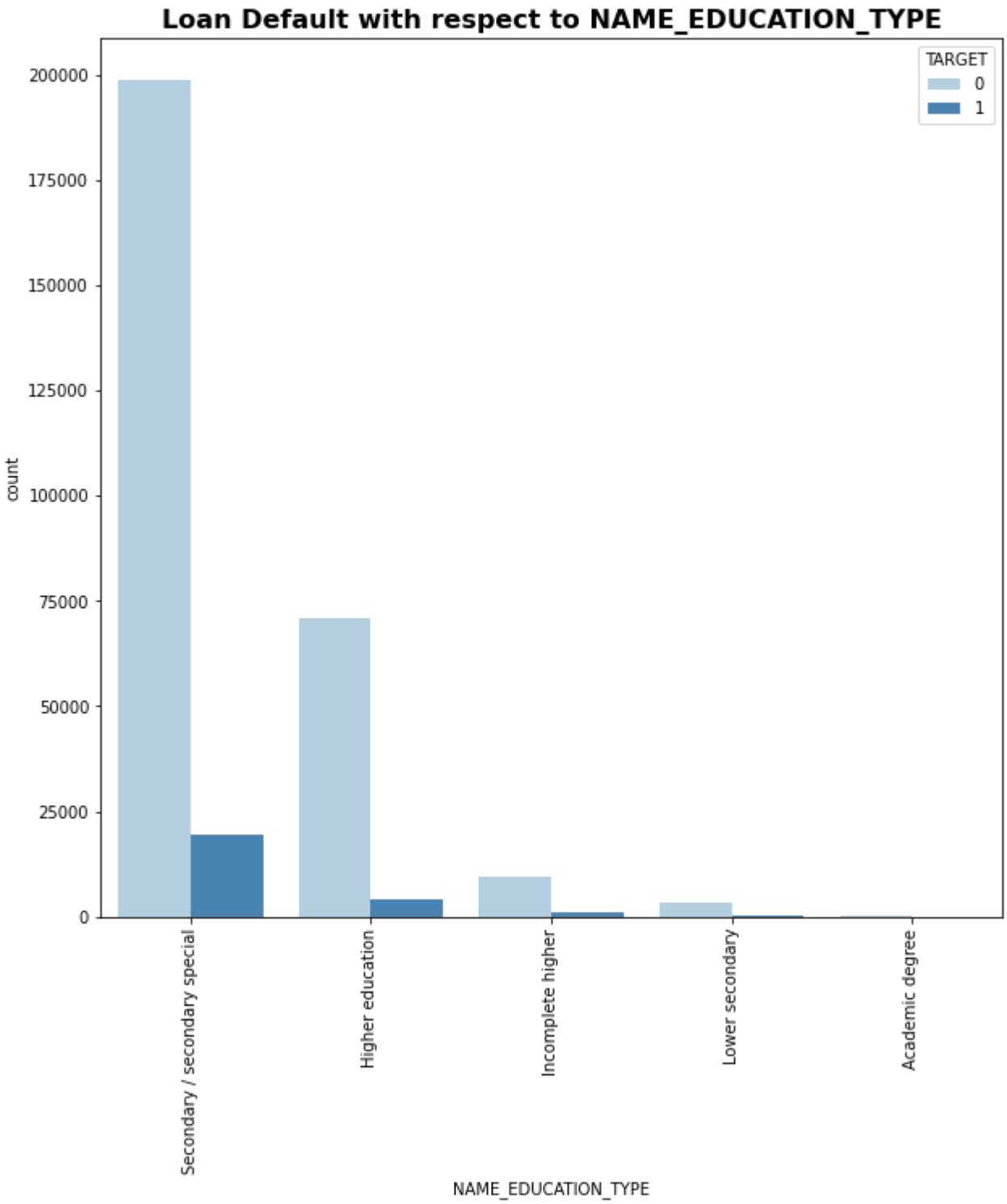
In [69]:
```
print(df_cat['NAME_EDUCATION_TYPE'].value_counts())
sns.countplot(df_cat['NAME_EDUCATION_TYPE'])
plt.title("Education type vs count")
plt.xticks(rotation=90)
```

```
Secondary / secondary special    218391
Higher education                  74863
Incomplete higher                 10277
Lower secondary                    3816
Academic degree                     164
Name: NAME_EDUCATION_TYPE, dtype: int64
/N/u/tankulk/Carbonate/miniconda3/lib/python3.7/site-packages/seaborn/_decorators.py:43:
FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments without an exp
licit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[69]:
```
(array([0, 1, 2, 3, 4]),
 [Text(0, 0, 'Secondary / secondary special'),
  Text(1, 0, 'Higher education'),
  Text(2, 0, 'Incomplete higher'),
  Text(3, 0, 'Lower secondary'),
  Text(4, 0, 'Academic degree')])
```



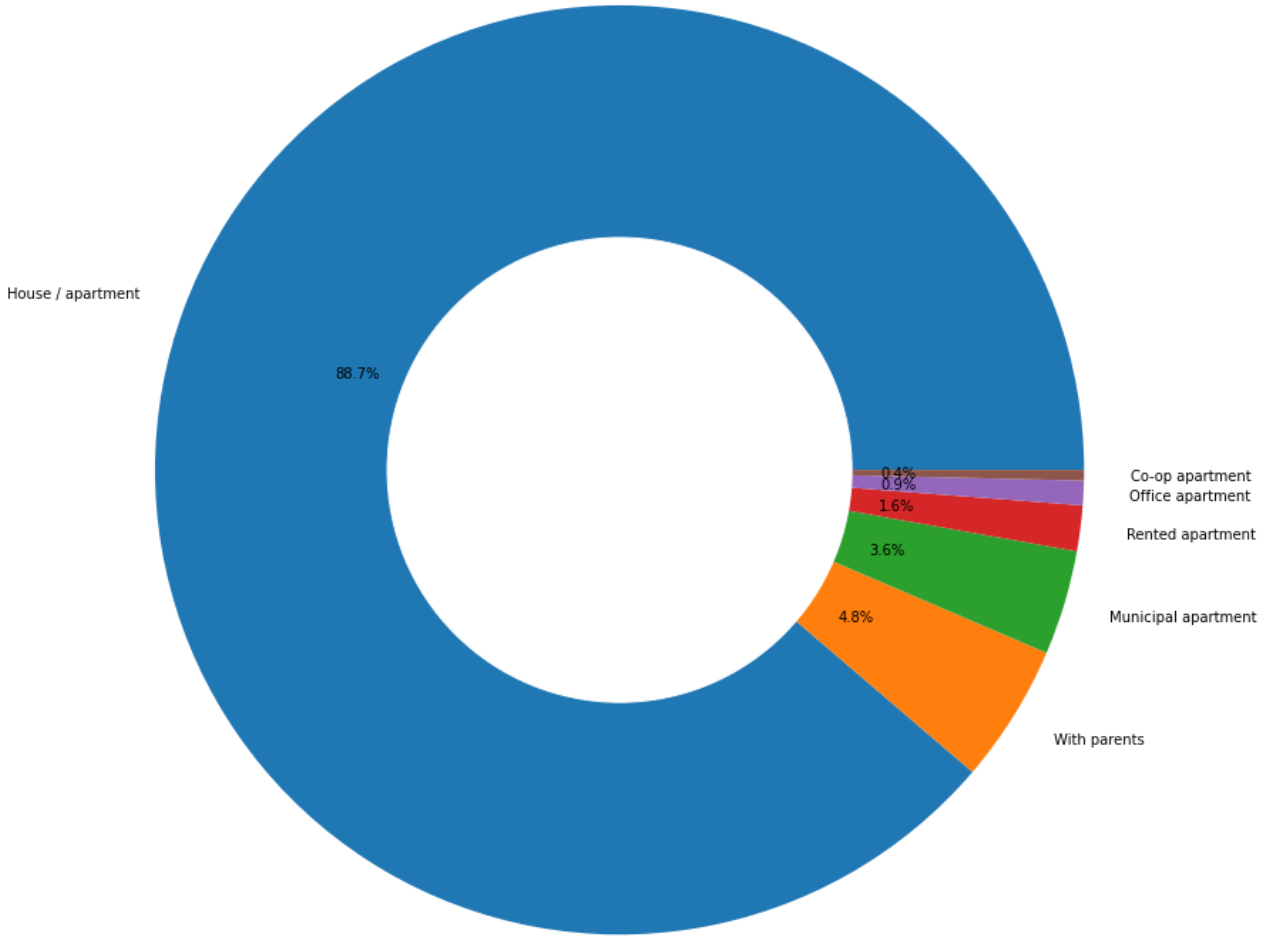
In [70]:
```
col('NAME_EDUCATION_TYPE')
```



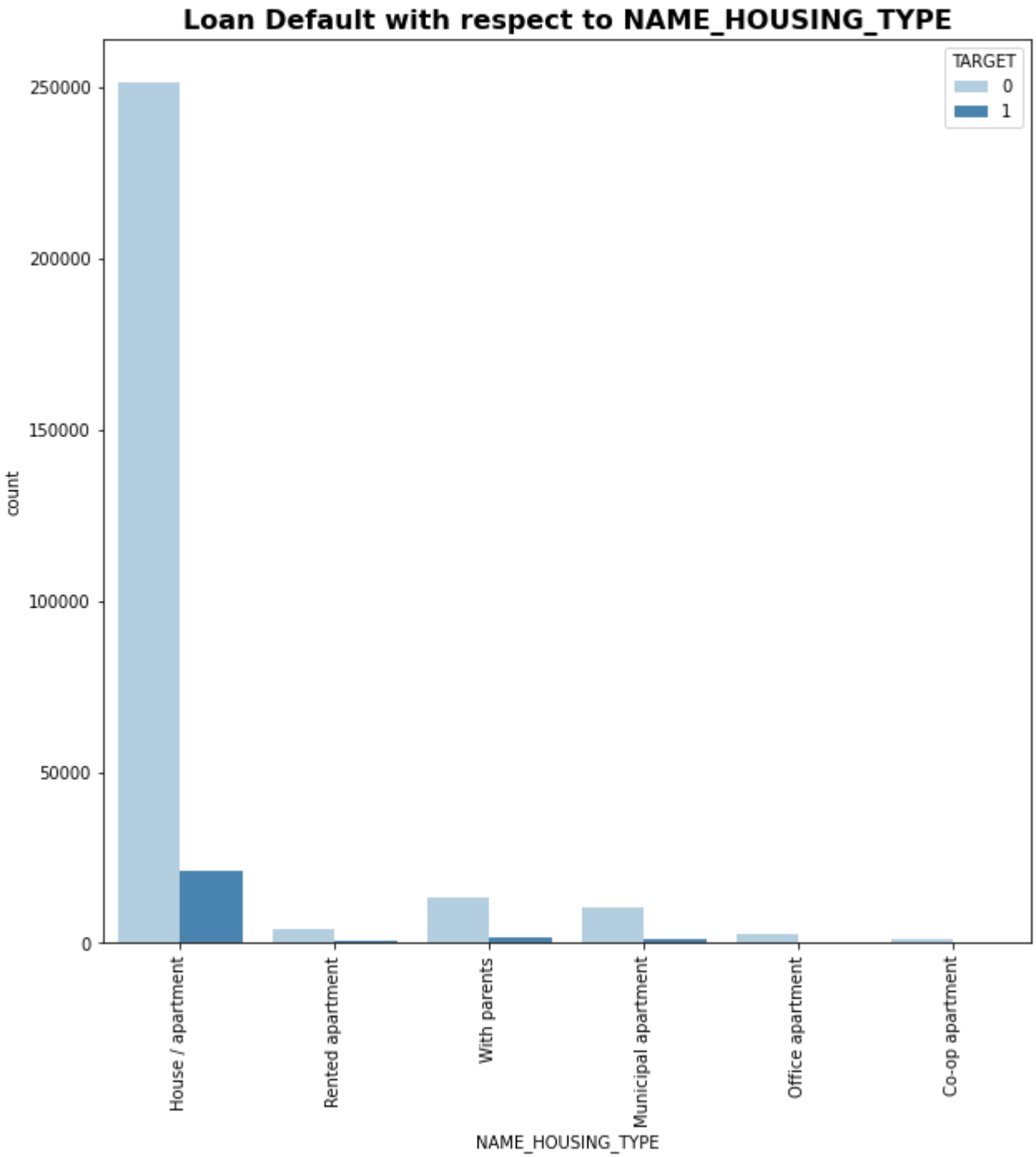
Inference: Clients with Academic Degree are more likely to repay the loan compared to others.

What are the types of housing does the clients stay in?

In [71]:
```
plt.figure(figsize=[20,15])
plt.pie(df_cat['NAME_HOUSING_TYPE'].value_counts(),labels = df_cat['NAME_HOUSING_TYPE']
my_circle=plt.Circle( (0,0), 0.5, color='white')
p=plt.gcf()
p.gca().add_artist(my_circle)
# plt.title('Percentage of User Types')
plt.show()
#plt.ticks(rotation=90)
```



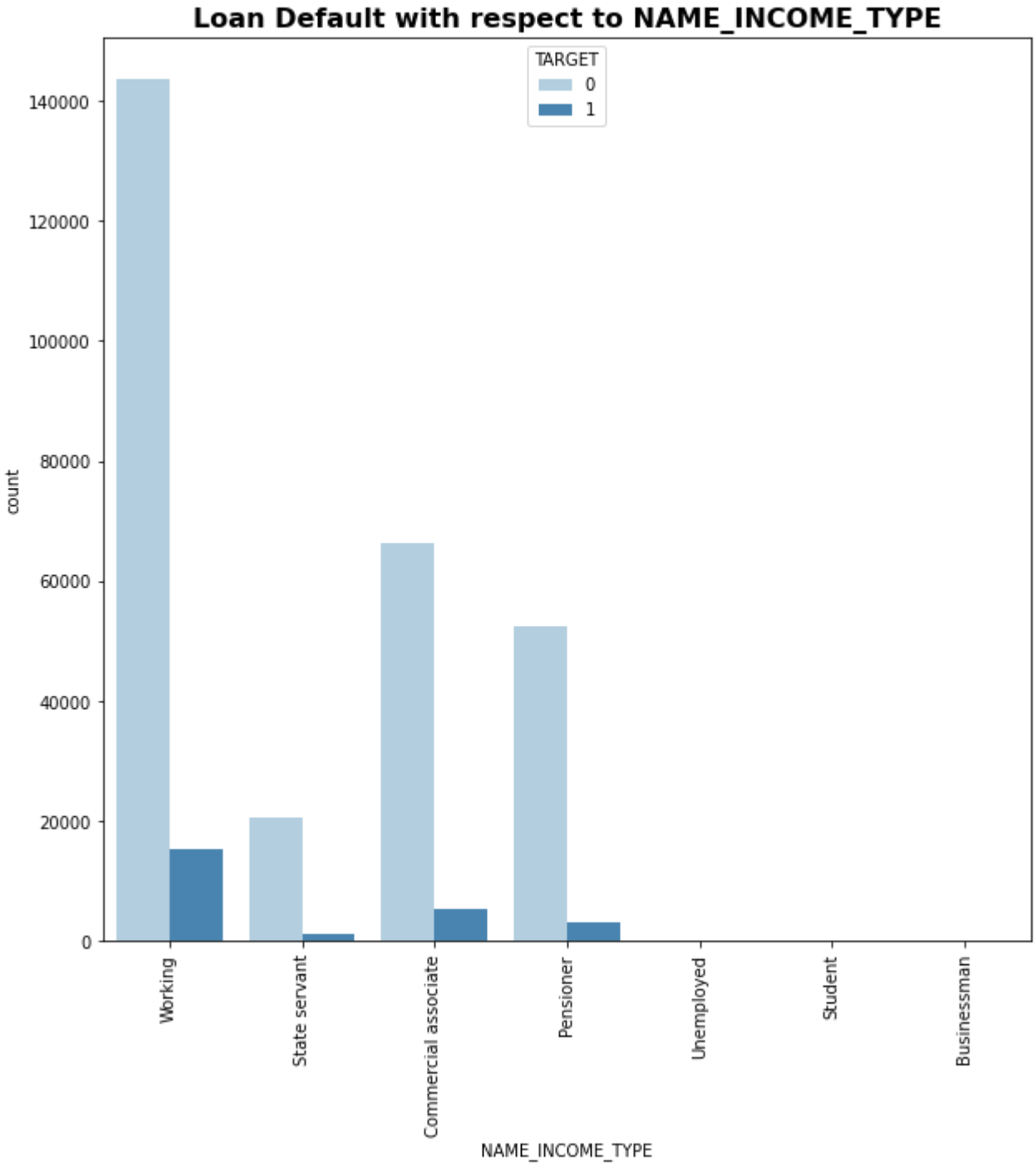
In [72]:
```
col("NAME_HOUSING_TYPE")
```



Inference: From the above graphical presentation, we can see that majority of the clients stay in apartment/House haven't paid the loan amount, while the least amount of them stay in office apartment and co-op apartment are negligible.

What are income type of applicant in terms of loan does the clients have?

In [73]:
```
col("NAME_INCOME_TYPE")
```



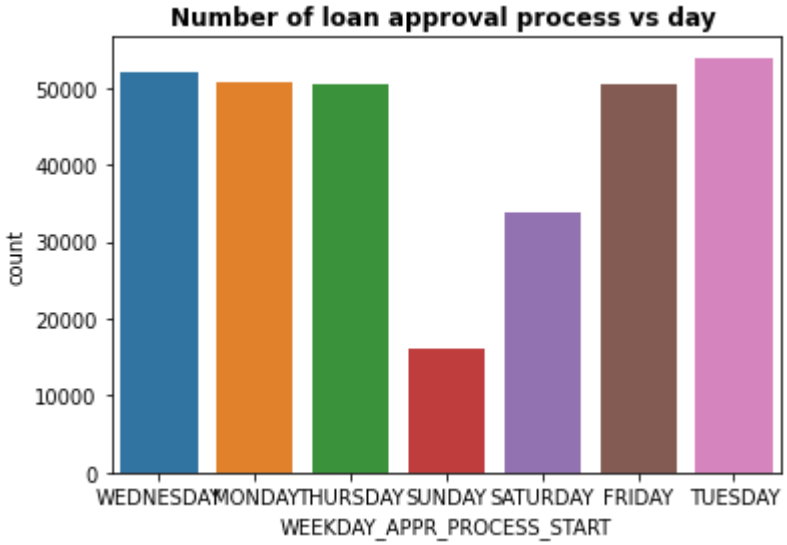
Inference: All the Students and Businessman are neglibile, here we can see that majority of working people are hardly paying the loan.

In [74]:
```
print(df_cat['WEEKDAY_APPR_PROCESS_START'].value_counts())
sns.countplot(df_cat['WEEKDAY_APPR_PROCESS_START'])
plt.title("Number of loan approval process vs day ", fontweight = 'bold', fontsize = 12
```

```
TUESDAY      53901
WEDNESDAY    51934
MONDAY       50714
THURSDAY     50591
FRIDAY       50338
SATURDAY     33852
SUNDAY       16181
Name: WEEKDAY_APPR_PROCESS_START, dtype: int64
```

```
/N/u/tankulk/Carbonate/miniconda3/lib/python3.7/site-packages/seaborn/_decorators.py:43:
FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments without an exp
licit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[74]: Text(0.5, 1.0, 'Number of loan approval process vs day ')



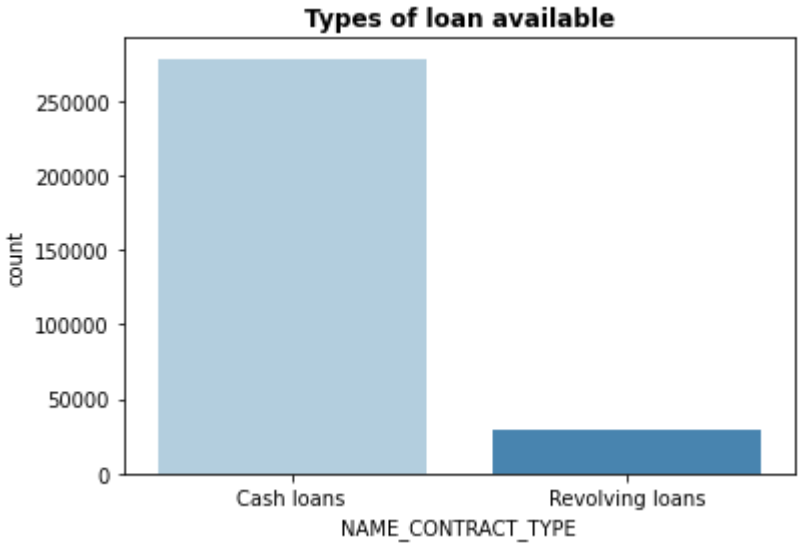
Inference: The loan approval process has the highest count starting tuesday ,while the lowest count can be clearly seen on the weekends.

What type of loan are available?

In [75]:
```
print(df_cat['NAME_CONTRACT_TYPE'].value_counts())
sns.countplot(df_cat['NAME_CONTRACT_TYPE'], palette = 'Blues')
plt.title("Types of loan available", fontweight = 'bold', fontsize = 12)
```

```
Cash loans         278232
Revolving loans     29279
Name: NAME_CONTRACT_TYPE, dtype: int64
/N/u/tankulk/Carbonate/miniconda3/lib/python3.7/site-packages/seaborn/_decorators.py:43:
FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments without an exp
licit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[75]: Text(0.5, 1.0, 'Types of loan available')

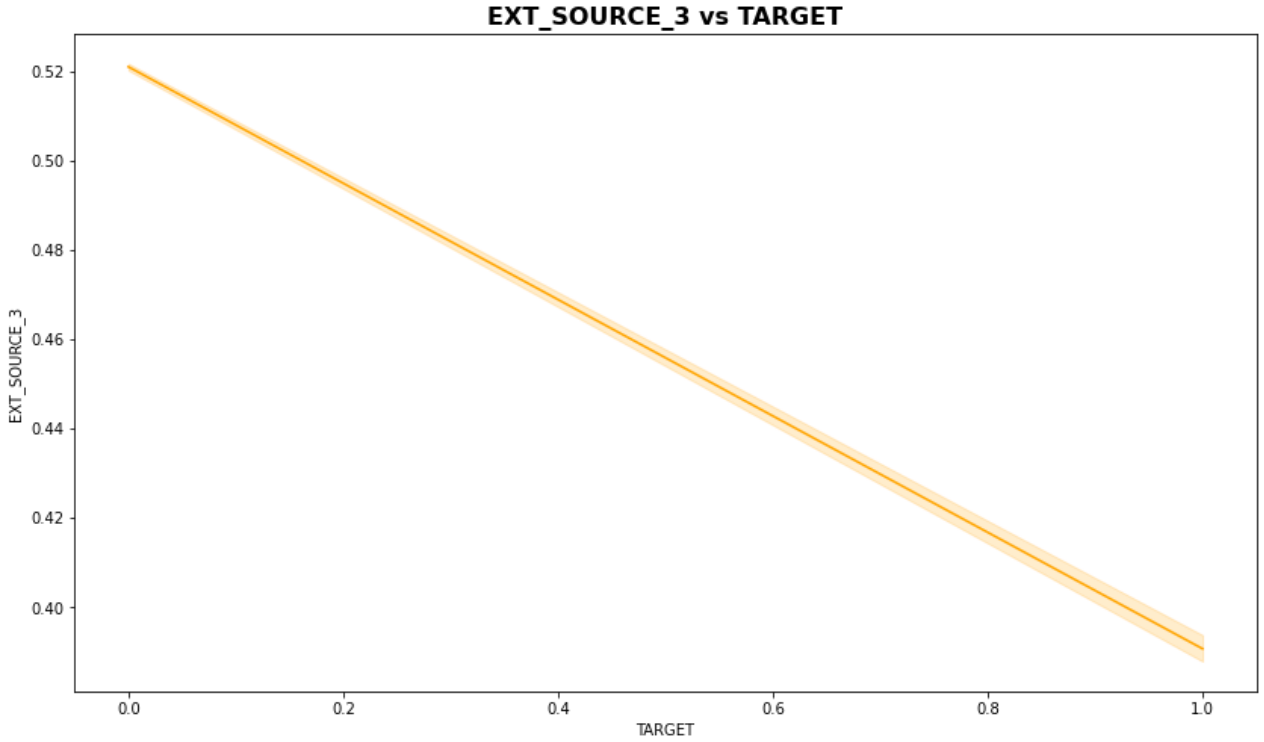


Inference: Many people are willing to take cash loan than revolving loan

Here, we plot some graphs for the columns with highest correlations with the Target variable and observe the trend with respect to the target variable.

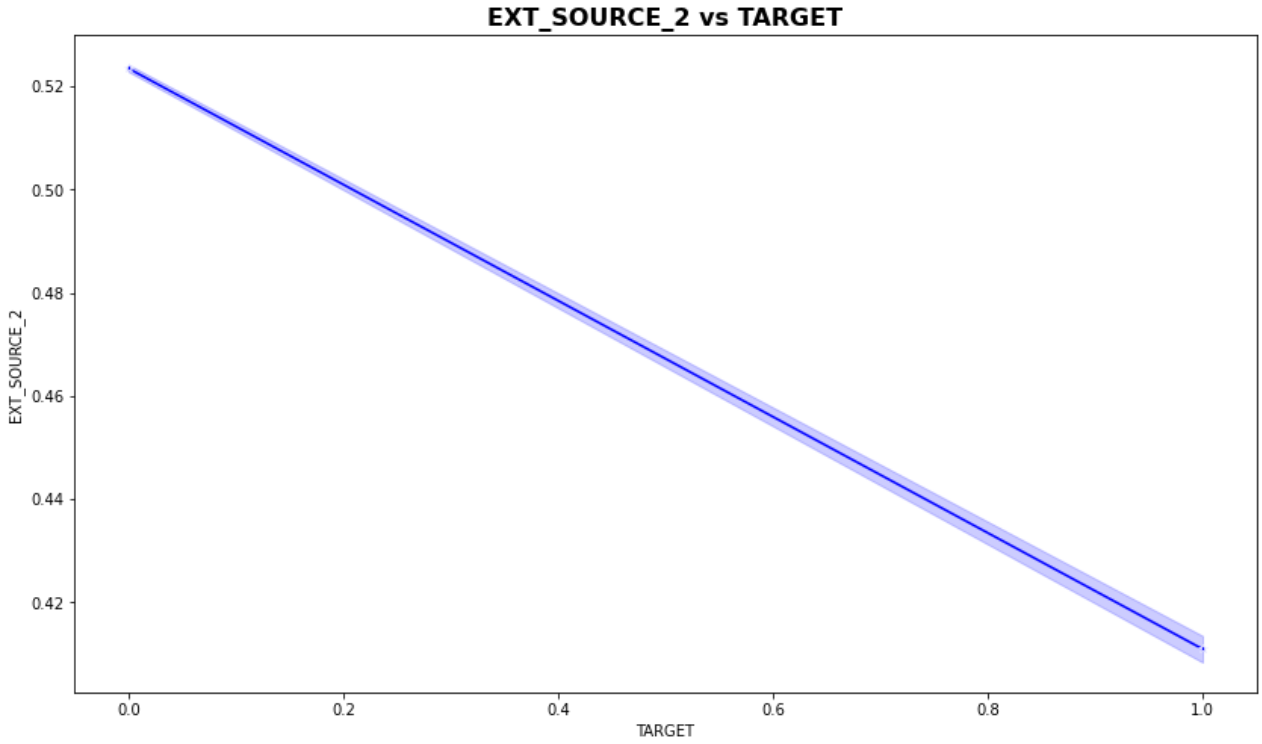
In [76]:
```
#Numerical Plot
plt.figure(figsize=[14,8])
sns.lineplot(x='TARGET',y='EXT_SOURCE_3',data=df,color= 'orange')
plt.title("EXT_SOURCE_3 vs TARGET", fontweight = 'bold', fontsize = 16)
```

Out[76]:
```
Text(0.5, 1.0, 'EXT_SOURCE_3 vs TARGET')
```



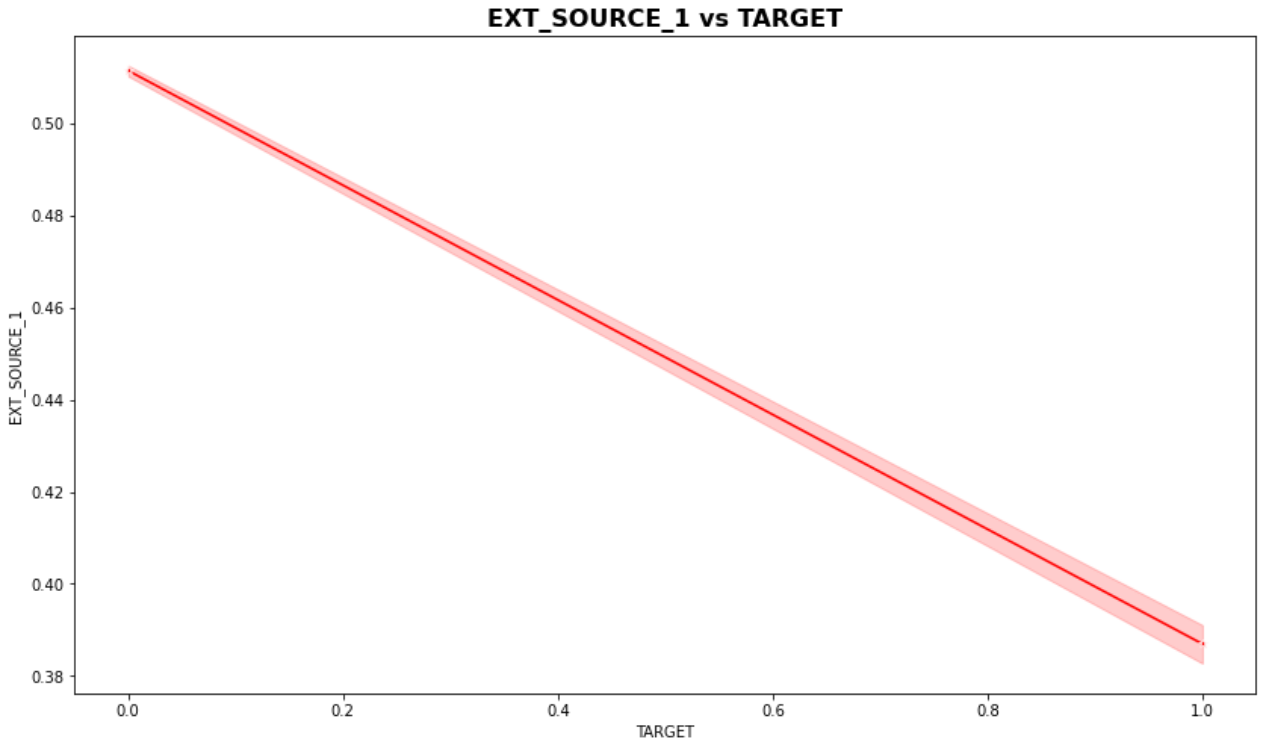
In [77]:
```
plt.figure(figsize=[14,8])
sns.lineplot(x='TARGET',y='EXT_SOURCE_2',data=df,marker='*',color= 'blue')
plt.title("EXT_SOURCE_2 vs TARGET", fontweight = 'bold', fontsize = 16)
```

Out[77]: Text(0.5, 1.0, 'EXT_SOURCE_2 vs TARGET')



In [78]:
```
plt.figure(figsize=[14,8])
sns.lineplot(x='TARGET',y='EXT_SOURCE_1',data=df,marker='*',color= 'red')
plt.title("EXT_SOURCE_1 vs TARGET", fontweight = 'bold', fontsize = 16)
```

Out[78]: Text(0.5, 1.0, 'EXT_SOURCE_1 vs TARGET')

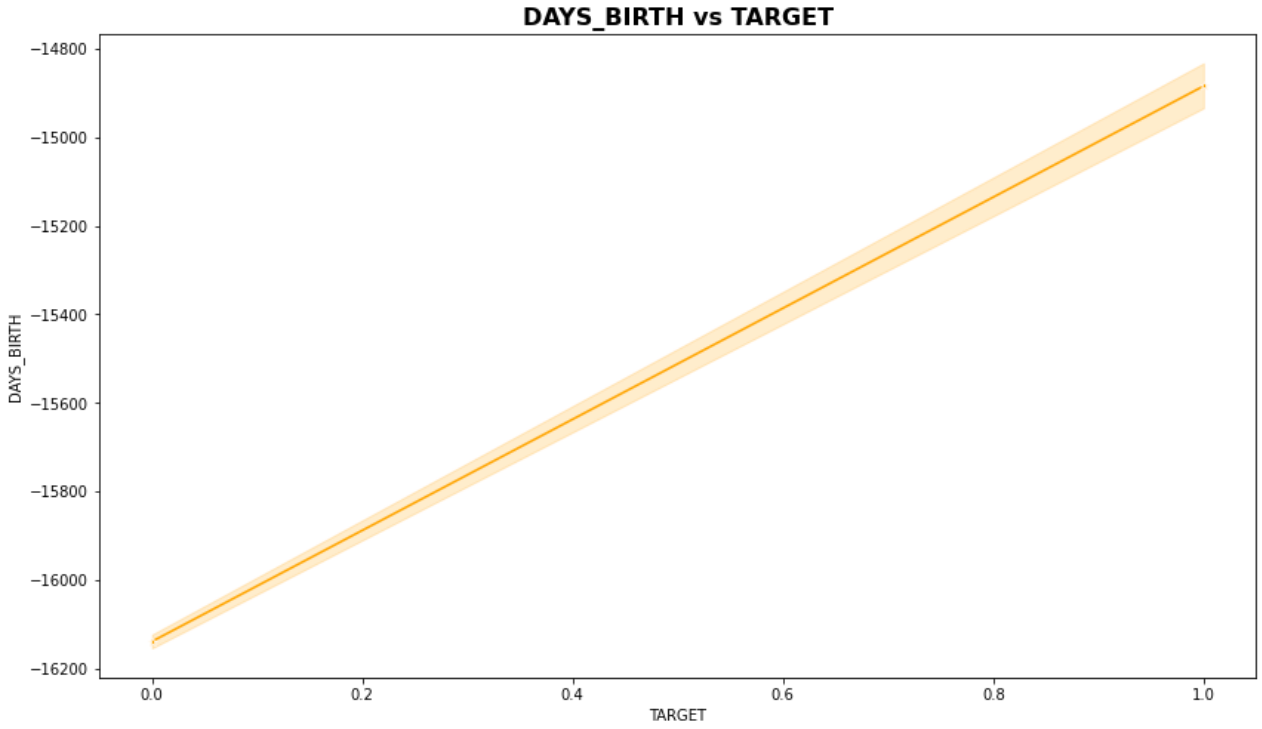


Inference: We see these for the columns EXT_SOURCE_3, EXT_SOURCE_2 and EXT_SOURCE_1 and can observe a clear strong negative correlation

What type of correlation does the columns DAYS_BIRTH and DAY_LAST_PHONE_CHANGE have with respect to target?

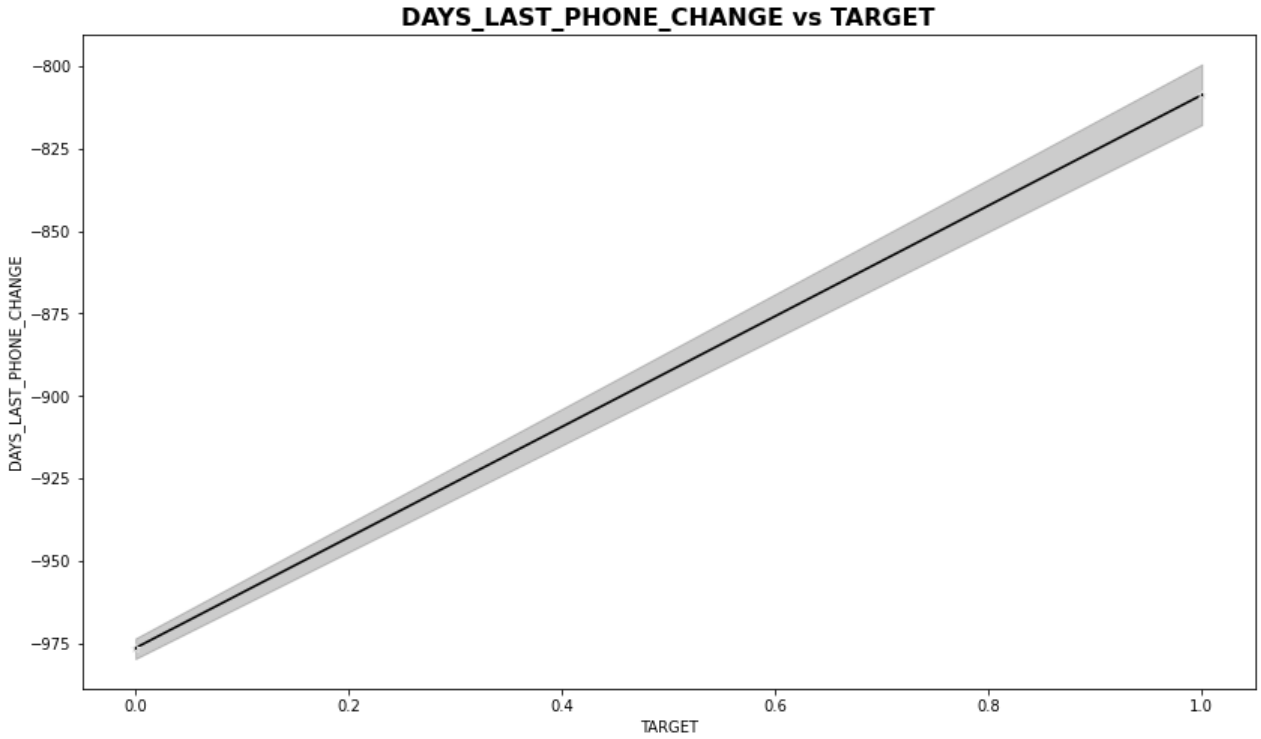
In [79]:
```
plt.figure(figsize=[14,8])
sns.lineplot(x='TARGET',y='DAYS_BIRTH',data=df,marker='*',color= 'orange')
plt.title("DAYS_BIRTH vs TARGET", fontweight = 'bold', fontsize = 16)
```

Out[79]: Text(0.5, 1.0, 'DAYS_BIRTH vs TARGET')



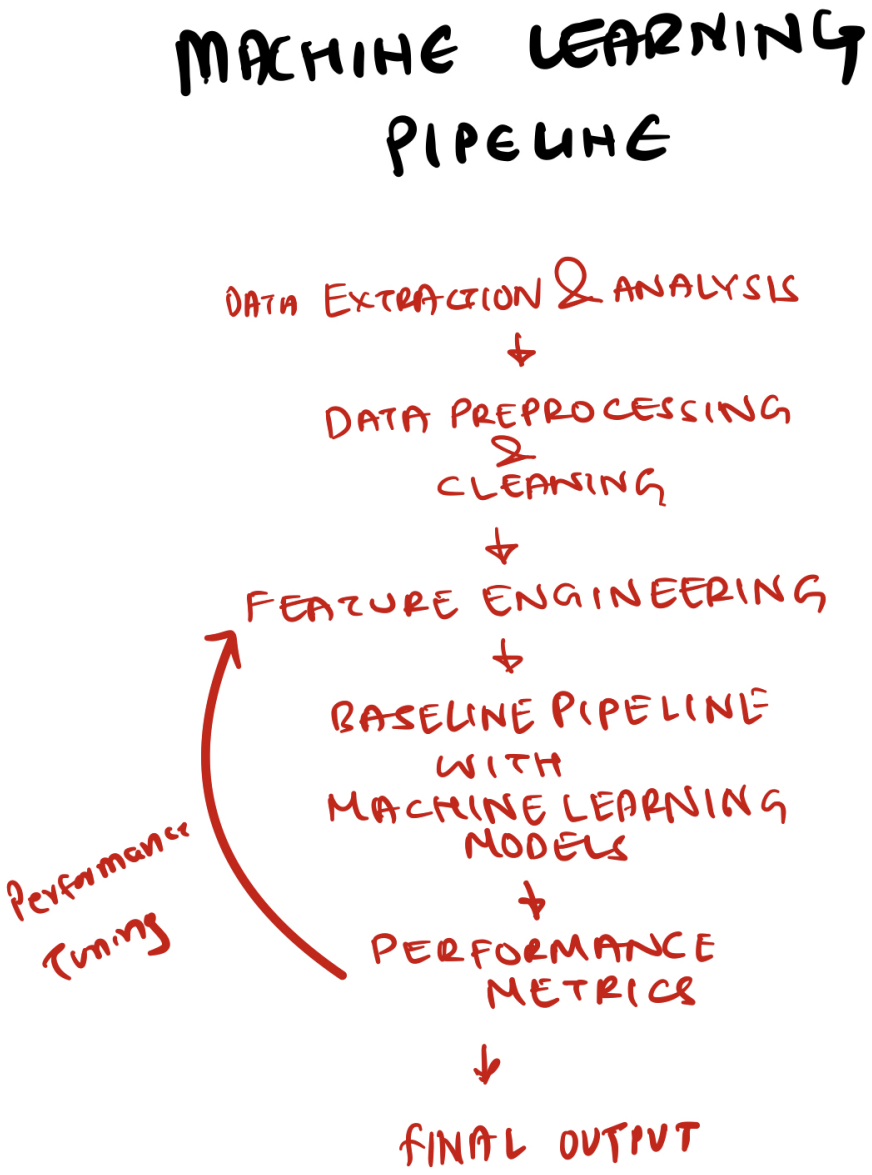
In [80]:
```
plt.figure(figsize=[14,8])
sns.lineplot(x='TARGET',y='DAYS_LAST_PHONE_CHANGE',data=df,marker='*',color= 'black')
plt.title("DAYS_LAST_PHONE_CHANGE vs TARGET", fontweight = 'bold', fontsize = 16)
```

Out[80]: Text(0.5, 1.0, 'DAYS_LAST_PHONE_CHANGE vs TARGET')



Inference: From the above plots,we can see that for columns of DAYS_BIRTH and DAYS_LAST_PHONE_CHANGE we see a strong positive correlation with respect to the target.

MODELING PIPELINES



In this project, we are creating three pipelines, one for numerical data, one for categorical data and finally a pipeline to combine the data.

(i).Numerical data pipeline: For the pipeline with numerical data also called 'num_pipeline', we impute the missing values by the mean of the columns.

(ii). Categorical data pipeline: For the pipeline with categorical data also called 'cat_pipeline', we impute the missing values by the mode or the most frequent data.

(iii)Final pipeline: We create a pipeline to merge the numerical and categorical columns that have no missing values. The categorical columns are also one hot encoded.

Importing necessary packages

In [81]:

```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

In [82]:
```
df_num.shape
```

Out[82]: (307511, 48)

Finally selecting only the columns that we have finally decided for the numerical and the categorical part

In [83]:
```
# df_num.drop(columns = ['TARGET'],inplace=True)
df_cols = list(df_num.columns)+ list(df_cat.columns)
```

In [84]:
```
df_i = df[df_cols]
```

Making two pipelines one for the numerical data where we impute the missing values by the mean of the columns, and the other for categorical data where we impute the missing data in the categorical columns using the Mode or the most frequent data.

In [69]:
```
num_pipeline = Pipeline(steps=[('imputer', SimpleImputer(strategy='mean'))])

cat_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
    ])
```

Here we create a pipeline to merge the numerical and categorical columns that have no missing values and the categorical columns are one hot encoded.

In [70]:
```
data_pipeline = ColumnTransformer([
    ("num_pipeline", num_pipeline, df_num.columns),
    ("cat_pipeline", cat_pipeline, df_cat.columns)], n_jobs = -1)
df_transformed = data_pipeline.fit_transform(df_i)
column_names = list(df_num.columns)  + \
               list(data_pipeline.transformers_[1][1].named_steps["ohe"].get_feature_na
```

```
/N/u/tankulk/Carbonate/miniconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.
py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names is dep
recated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

This is the final dataset that we get for training our model

In [71]:
```
df_n = pd.DataFrame(df_transformed,  columns=column_names)
```

In [7]:

```
df_n = pd.read_csv('bhai_appl_train_bhai.csv')
```

In [8]:
```
df_n.drop(columns=['Unnamed: 0'],inplace=True)
```

In [9]:
```
df_n.head()
```

Out[9]:

	SK_ID_CURR	DEF_30_CNT_SOCIAL_CIRCLE	LIVINGAPARTMENTS_MODE	LIVINGAPARTMENTS_MEDI	LIV
0	100002.0	2.0	0.022000	0.020500	
1	100003.0	0.0	0.079000	0.078700	
2	100004.0	0.0	0.105646	0.101956	
3	100006.0	0.0	0.105646	0.101956	
4	100007.0	0.0	0.105646	0.101956	

5 rows × 156 columns

In [10]:
```
df_temp = df_n
```

In [11]:
```
# df_n = df_temp
```

In [ ]:
```

```

In [12]:
```
df_n = pd.merge(left=df_n, right=df_inst, how='left', left_on='SK_ID_CURR', right_on='S
```

In [13]:
```
df_n = pd.merge(left=df_n, right=df_pos_c_bal, how='left', left_on='SK_ID_CURR', right_
```

In [14]:
```
df_n = pd.merge(left=df_n, right=df_prev_app, how='left', left_on='SK_ID_CURR', right_o
```

In [15]:
```
df_n = pd.merge(left=df_n, right=credit_card_bal, how='left', left_on='SK_ID_CURR', rig
```

```
/N/u/tankulk/Carbonate/miniconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: F
utureWarning: Passing 'suffixes' which cause duplicate columns {'SK_ID_PREV_x'} in the r
esult is deprecated and will raise a MergeError in a future version.
  """Entry point for launching an IPython kernel.
```

In [21]:
```
df_n = pd.merge(left=df_n, right=new_bb, how='left', left_on='SK_ID_CURR', right_on='SK
```

In [22]:
```
df_final_app_imputer = df_n
```

In [ ]:

```
```

In [23]:

```
df_final_app_imputer['FEATURE1']= df_final_app_imputer['AMT_TOTAL_RECEIVABLE']/(df_fina
df_final_app_imputer['FEATURE2'] = df_final_app_imputer['AMT_TOTAL_RECEIVABLE']/(df_fin
df_final_app_imputer['FEATURE3'] = df_final_app_imputer['AMT_TOTAL_RECEIVABLE']/(df_fin


df_final_app_imputer['FEATURE4']=df_final_app_imputer['AMT_BALANCE']/(df_final_app_impu
df_final_app_imputer['FEATURE5']=df_final_app_imputer['AMT_BALANCE']/(df_final_app_impu
df_final_app_imputer['FEATURE6']=df_final_app_imputer['AMT_RECIVABLE']/(df_final_app_im
df_final_app_imputer['FEATURE7']=(df_final_app_imputer['EXT_SOURCE_1']+df_final_app_imp

df_final_app_imputer['FEATURE8']=(df_final_app_imputer['EXT_SOURCE_1']+df_final_app_imp
```

In [24]:

```
df_final_app_imputer['FEATURE9']=(df_final_app_imputer['EXT_SOURCE_1']*df_final_app_imp

df_final_app_imputer['FEATURE10'] =  (df_final_app_imputer['EXT_SOURCE_1']*2+df_final_a

df_final_app_imputer['FEATURE11'] = [max(ele1,ele2,ele3) for ele1, ele2, ele3 in zip(df
```

In [36]:

```
df_final_app_imputer.shape
correlation=corr_target(df_final_app_imputer[['FEATURE1','FEATURE2','FEATURE3','FEATURE
```

In [675…

```
correlation
```

Out[675…

	col_name	Correlation
0	TARGET	1.000000
1	FEATURE10	-0.223086
2	FEATURE8	-0.221463
3	FEATURE9	-0.189587
4	FEATURE11	-0.175531
5	FEATURE7	-0.071049
6	FEATURE2	0.050901
7	FEATURE5	0.001060
8	FEATURE6	0.000817
9	FEATURE3	0.000764
10	FEATURE4	0.000594
11	FEATURE1	0.000026

In [26]:

```
df_final_app_imputer['TARGET'].head()
```

Out[26]:

```
0    1.0
1    0.0
```

```
2    0.0
3    0.0
4    0.0
Name: TARGET, dtype: float64
```

1)Amt credit/Amt annuity, 2)Amt installment /Amt credit, 3)Day_first_due - Day_last_due, 4)Amt application/Amt payment 5)Amt credit sum/amt credit 6)Amt installment /Days installment 7))Amt credit sum/days credit

In [27]:
```
df_final_app_imputer = df_final_app_imputer.rename(columns = {'AMT_CREDIT_y':'AMT_CREDI
```

In [28]:
```
df_final_app_imputer = df_final_app_imputer.apply(lambda x: x.fillna(x.median()),axis=0
```

Including features only with 5% and above correlation

In [678…
```
five_percent_se_jyada = corr_target(df_final_app_imputer,0.05)
```

In [679…
```
five_percent_se_jyada = five_percent_se_jyada['col_name'][1:]
```

In [680…
```
#five_percent_se_jyada
```

Leakage

Data leakage in Machine Learning modeling pipelines usually occurs when the data that is available during training or feature engineering is unavailable on the ime of inference or testing the accuracy of model for untested/unseen data. For the pipeline that we have used, we see that there is no data leakage as we have dealt with all NaN values appropriately, and the data type has been set uniform across columns. We have also ensured that the new features that we have generated during feature engineering have been used appropriately during training and is available at the time of inference.

The most common cardinal sins that could map to this project that we understood were:

- Data and Model abuse: where there is no split for training and testing, also there is a high possibility of overfitting. Here we have sufficiently dealt with the above point by splitting the data and training it for best fit.
- Bad Data: We have checked appropriately for bad data and have removed that if any.
- Cross-validation chaos: we have used less number of cross-validations in our grid search to get the best average and not the overrated one.
- Overinterpretation of results: We have stuck to the evidence that we found through results and did not attempt to overmap tp pollute the understanding of results.
- Sticking to one score metric: We did not stick to just one score metric to understand our results but used additional metrics like roc score and confusion matrix to unterpret the results.

Thus we believe that we have not committed any cardinal sins of Machine Learning in this project.

RESULTS AND DISCUSSION OF RESULTS

In [681…
```
from sklearn.metrics import log_loss
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

In [705…
```
df_temp_5_percent.columns
```

Out[705…
```
Index(['Feature10', 'FEATURE10', 'FEATURE8', 'FEATURE9', 'EXT_SOURCE_MAX',
       'FEATURE11', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'EXT_SOURCE_1',
       'DAYS_CREDIT', 'DAYS_BIRTH', 'FEATURE7', 'DAYS_CREDIT_UPDATE',
       'REGION_RATING_CLIENT_W_CITY', 'REGION_RATING_CLIENT',
       'NAME_INCOME_TYPE_Working', 'NAME_EDUCATION_TYPE_Higher education',
       'DAYS_LAST_PHONE_CHANGE', 'CODE_GENDER_M', 'CODE_GENDER_F',
       'DAYS_ID_PUBLISH', 'REG_CITY_NOT_WORK_CITY', 'FEATURE2'],
      dtype='object')
```

In [682…
```
models_results = []
Target = df_final_app_imputer['TARGET']
df_temp_5_percent = df_final_app_imputer[five_percent_se_jyada]
```

In [ ]:

In [683…
```
X = df_temp_5_percent.values
y = Target.values
```

In [684…
```
# X = df_final_app_imputer.drop(columns = ['TARGET']).values
# y = Target.values
```

In [ ]:

In [ ]:

In [686…
```
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.15, random_stat
```

Here we will be using the lbfgs solver which is a Limited-memory BFGS (L-BFGS or LM-BFGS) optimization algorithm in the family of quasi-Newton methods that approximates the Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS) using a limited amount of computer memory

Pipeline+GridSearchCV

Logistic Regression

Hyperparameter tuning

- After selecting the optimal features for our models, we use hyperparameter tuning to find the most optimal setting for running our model.
- For hyperparameter tuning, we have decided to use GridSearchCV. The GridSearchCV is a library of sklearn's model selection package. It helps to fit our model on our data by using the best running conditions and the best parameters possible.
- By just specifying the model, input parameters, and accuracy we want, we can easily obtain the best running conditions and features using GridSearchCV.
- In our code, for logistic regression, we have created a function for GridSearchCV with 3 cross-validations for the hyperparameters and 1000 max iterations.

In [507…

```
# for i in ['FEATURE1','FEATURE2','FEATURE3','FEATURE4','FEATURE5','FEATURE6','FEATURE7
#     print(i)
#     print(df_final_app_imputer[i].min(),df_final_app_imputer[i].max())
#     print('***********')
```

In [687…

```
pipe = Pipeline([
('scaler', StandardScaler()),
('classifier', GridSearchCV(LogisticRegression(solver='lbfgs', max_iter=1000),
                                param_grid={'C': [ 0.1,1,5,10.]},
                                cv=5,
                                refit=True))
])
pipe.fit(X_train, y_train)
```

Out[687…

```
Pipeline(steps=[('scaler', StandardScaler()),
                ('classifier',
                 GridSearchCV(cv=5, estimator=LogisticRegression(max_iter=1000),
                              param_grid={'C': [0.1, 1, 5, 10.0]}))])
```

In [688…

```
pipe.named_steps['classifier'].best_params_
```

Out[688…

```
{'C': 0.1}
```

In [689…

```
print('Training set accuracy score: ' + str(pipe.score(X_train,y_train)))
y_pred = pipe.predict(X_valid)
print('Validation set accuracy score: ' + str(accuracy_score(y_valid,y_pred)))
print('Log loss: ',log_loss(y_valid,y_pred))
print('Confusion Matrix: ','\n',confusion_matrix(y_valid, y_pred))
print('ROC_AUC: ',roc_auc_score(y_valid, pipe.predict_proba(X_valid)[:, 1]))
```

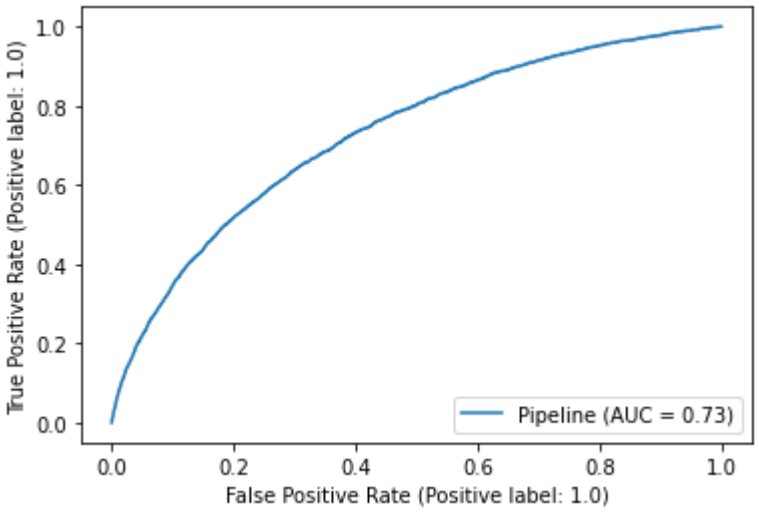
```
Training set accuracy score: 0.9192883787661406
Validation set accuracy score: 0.9189376693766937
Log loss:  2.79979385013226
Confusion Matrix:
 [[42372     8]
 [ 3731    14]]
ROC_AUC:  0.7293815633366117
```

In [690…
```
models_results.append(['Logistic Regression',pipe.score(X_train,y_train),accuracy_score
```

In [691…
```
metrics.plot_roc_curve(pipe, X_valid, y_valid)
```

```
/N/u/tankulk/Carbonate/miniconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.
py:87: FutureWarning: Function plot_roc_curve is deprecated; Function `plot_roc_curve` i
s deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: RocCurveDi
splay.from_predictions or RocCurveDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[691… <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f785d482ad0>



In [ ]:

RandomForestClassifer

In [692…
```
# param_grid = {
#     'n_estimators': [200, 500],
#     'max_features': ['auto', 'sqrt', 'log2'],
#     'max_depth' : [4,5,6,7,8],
#     'criterion' :['gini', 'entropy']
# }

param_grid = {
    'n_estimators': [40,50,60],
    'max_features': ['auto'],
    'max_depth' : [15,20,25],
    'criterion' :['entropy']
}
```

In [693…
```
pipe1 = Pipeline([
('scaler', StandardScaler()),
('classifier', GridSearchCV(RandomForestClassifier(),
                                param_grid=param_grid,
                                cv=4,
                                refit=True))
])
pipe1.fit(X_train, y_train)
```

Out[693…
```
Pipeline(steps=[('scaler', StandardScaler()),
                ('classifier',
                 GridSearchCV(cv=4, estimator=RandomForestClassifier(),
                              param_grid={'criterion': ['entropy'],
                                          'max_depth': [15, 20, 25],
                                          'max_features': ['auto'],
                                          'n_estimators': [40, 50, 60]}))])
```

In [694…
```
pipe1.named_steps['classifier'].best_params_
```

Out[694…
```
{'criterion': 'entropy',
 'max_depth': 15,
 'max_features': 'auto',
 'n_estimators': 40}
```

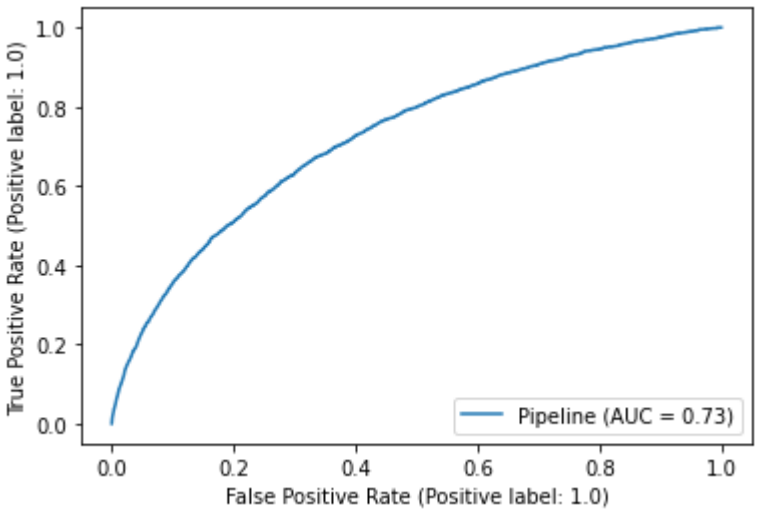
In [697…
```
print('Training set accuracy score: ' + str(pipe1.score(X_train,y_train)))
y_pred = pipe1.predict(X_valid)
print('Validation set accuracy score: ' + str(accuracy_score(y_valid,y_pred)))
print('Log loss: ',log_loss(y_valid,y_pred))
print('Confusion Matrix: ','\n',confusion_matrix(y_valid, y_pred))
print('ROC_AUC: ',roc_auc_score(y_valid, pipe1.predict_proba(X_valid)[:, 1]))
```

```
Training set accuracy score: 0.9240593017694883
Validation set accuracy score: 0.9191761517615176
Log loss:  2.791557324413063
Confusion Matrix:
 [[42351    29]
 [ 3699    46]]
ROC_AUC:  0.7269508408568669
```

In [696…
```
metrics.plot_roc_curve(pipe1, X_valid, y_valid)
```

```
/N/u/tankulk/Carbonate/miniconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.
py:87: FutureWarning: Function plot_roc_curve is deprecated; Function `plot_roc_curve` i
s deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: RocCurveDi
splay.from_predictions or RocCurveDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[696…
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f785cfd5310>
```



In [601…

In [698…
```
models_results.append(['Random Regression Classifier',pipe1.score(X_train,y_train),accu
```

In [699…
```
pd.DataFrame(models_results,columns = ['Model Name', 'Training Accuracy', 'Validation A
```

Out[699…

	Model Name	Training Accuracy	Validation Accuracy
0	Logistic Regression	0.919288	0.918938
1	Random Regression Classifier	0.924059	0.919176

In [700…
```
# make_sub(pipe)
```

In [ ]:

In [ ]:

Pipeline

In [636…
```
pipe_lr = Pipeline([
('scaler', StandardScaler()),
('classifier', LogisticRegression(solver='lbfgs', max_iter=1000))
])

pipe.fit(X_train, y_train)
```

Out[636…
```
Pipeline(steps=[('scaler', StandardScaler()),
                ('classifier', LogisticRegression(max_iter=1000))])
```

In [637…
```
print('Training set accuracy score: ' + str(pipe_lr.score(X_train,y_train)))
y_pred = pipe_lr.predict(X_valid)
print('Validation set accuracy score: ' + str(accuracy_score(y_valid,y_pred)))
print('Log loss: ',log_loss(y_valid,y_pred))
print('Confusion Matrix: ','\n',confusion_matrix(y_valid, y_pred))
print('ROC_AUC: ',roc_auc_score(y_valid, pipe_lr.predict_proba(X_valid)[:, 1]))
```

```
Training set accuracy score: 0.9192156862745098
Validation set accuracy score: 0.9188509485094851
Log loss:  2.8027894988242488
Confusion Matrix:
 [[42348    32]
 [ 3711    34]]
ROC_AUC:  0.729216630511281
```

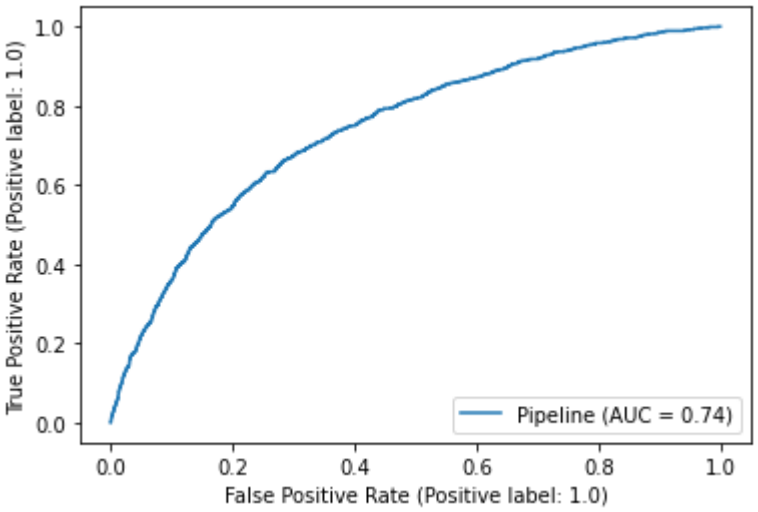
In [638…

In [140…
```
models_results.append(['Logistic Regression',pipe_lr.score(X_train,y_train),accuracy_sc
```

In [141…
```
metrics.plot_roc_curve(pipe_lr, X_valid, y_valid)
```

```
/N/u/tankulk/Carbonate/miniconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.
py:87: FutureWarning: Function plot_roc_curve is deprecated; Function `plot_roc_curve` i
s deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: RocCurveDi
splay.from_predictions or RocCurveDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[141… `<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f528f8a35d0>`



In [132…

```
# pipe_naive_bayes = Pipeline([
# ('scaler', StandardScaler()),
# ('classifier', GaussianNB())
# ])

# pipe_naive_bayes.fit(X_train, y_train)



# print('Training set accuracy score: ' + str(pipe_naive_bayes.score(X_train,y_train)))
# y_pred = pipe_naive_bayes.predict(X_valid)
# print('Validation set accuracy score: ' + str(accuracy_score(y_valid,y_pred)))
# print('Log loss: ',log_loss(y_valid,y_pred))
# print('Confusion Matrix: ','\n',confusion_matrix(y_valid, y_pred))
# print('ROC_AUC: ',roc_auc_score(y_valid, pipe_naive_bayes.predict_proba(X_valid)[:, 1

# models_results.append(['Naive Bayes',pipe_naive_bayes.score(X_train,y_train),accuracy_

# metrics.plot_roc_curve(pipe_naive_bayes, X_valid, y_valid)
```

In [654…

```
pipe_rf = Pipeline([
('scaler', StandardScaler()),
('classifier', RandomForestClassifier())
])

pipe_rf.fit(X_train, y_train)
```

Out[654… `Pipeline(steps=[('scaler', StandardScaler()),`
`('classifier', RandomForestClassifier())])`

In [655…

```
print('Training set accuracy score: ' + str(pipe_rf.score(X_train,y_train)))
y_pred = pipe_rf.predict(X_valid)
```

```
print('Validation set accuracy score: ' + str(accuracy_score(y_valid,y_pred)))
print('Log loss: ',log_loss(y_valid,y_pred))
print('Confusion Matrix: ','\n',confusion_matrix(y_valid, y_pred))
print('ROC_AUC: ',roc_auc_score(y_valid, pipe_rf.predict_proba(X_valid)[:, 1]))
```

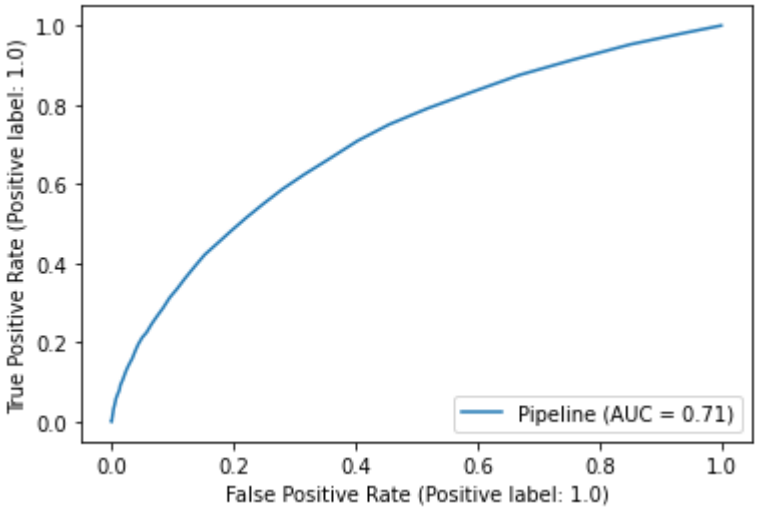
```
Training set accuracy score: 0.999950263032042
Validation set accuracy score: 0.9189593495934959
Log loss:  2.799045926079665
Confusion Matrix:
 [[42321    59]
 [ 3679    66]]
ROC_AUC:  0.7078786754212475
```

In [656…
```
models_results.append(['Random Forest Classifier',pipe_rf.score(X_train,y_train),accura
```

In [657…
```
metrics.plot_roc_curve(pipe_rf, X_valid, y_valid)
```

```
/N/u/tankulk/Carbonate/miniconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.
py:87: FutureWarning: Function plot_roc_curve is deprecated; Function `plot_roc_curve` i
s deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: RocCurveDi
splay.from_predictions or RocCurveDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[657… `<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f785d4e4710>`



In [200…
```
pd.DataFrame(models_results,columns = ['Model Name', 'Training Accuracy', 'Validation A
```

Out[200…

	Model Name	Training Accuracy	Validation Accuracy
0	Logistic Regression	0.922662	0.920591
1	Naive Bayes	0.114888	0.115725
2	Random Forest Classifier	1.000000	0.920591

Here we see that Random Forest Classifier has the best training accuracy of around 99% but an accuracy like this runs a risk of overfitting. The logistic Regression Model also gives an accuracy of 91% which is descent enough and seems to be a consdierable model. The ROC Area Under Curve value for Random Forest Classifier and the Logistic regression models are 0.704 and 0.735 respectively and both of them show a significant amount of True Positive values which shows that

this is a good model fit. We cannot consider Naive Bayes as our model as it is evidently underfitting the data. We need to check how Random Forest classifier and the Logistic Regression model works on the test data to confirm if Random forest is really overfitting.

For Kaggle submission

In [ ]:
```
X = df_n.values
y = df['TARGET'].values
```

In the following pipeline we use standard scaler to normalize the data with mean being zero and the standard deviation being 1, and use Logistic Regression as our modeling algorithm with solver being lbfgs and interations for convergence being 1000

In [ ]:
```
pipe = Pipeline([
('scaler', StandardScaler()),
('classifier', LogisticRegression(solver='lbfgs', max_iter=1000))
])

pipe.fit(X, y)

print('Training set score: ' + str(pipe.score(X,y)))
```

```
Training set score: 0.919190243902439
```

In [ ]:
```
pipe_rf = Pipeline([
('scaler', StandardScaler()),
('classifier', RandomForestClassifier())
])

pipe_rf.fit(X, y)
```

Out[ ]:
```
Pipeline(steps=[('scaler', StandardScaler()),
                ('classifier', RandomForestClassifier())])
```

In [ ]:
```
print('Training set score: ' + str(pipe_rf.score(X_train,y_train)))
```

```
Training set score: 0.9999464371114299
```

Test Dataset

Here we get the same final columns as for the training dataset in the test dataset

In [85]:
```
df_cols = list(df_num.columns)+ list(df_cat.columns)
```

In [88]:
```
# df_cols.remove('TARGET')
```

In [89]:
```
df_test_n= df_test[df_cols]
```

In [90]:
```
num_pipeline1 = Pipeline(steps=[('imputer', SimpleImputer(strategy='mean'))])
cat_pipeline1 = Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
    ])
```

In [91]:
```
df_test_n_num = df_test_n.select_dtypes(exclude='object')
df_test_n_cat = df_test_n.select_dtypes(include='object')
```

In [92]:
```
data_pipeline1 = ColumnTransformer([
    ("num_pipeline", num_pipeline1, df_test_n_num.columns),
    ("cat_pipeline", cat_pipeline1, df_test_n_cat.columns)], n_jobs = -1)
df_transformed1 = data_pipeline1.fit_transform(df_test_n)
column_names = list(df_test_n_num.columns)  + \
               list(data_pipeline1.transformers_[1][1].named_steps["ohe"].get_feature_n
```

```
/N/u/tankulk/Carbonate/miniconda3/lib/python3.7/site-packages/sklearn/utils/deprecation.
py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names is dep
recated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

In [93]:
```
df_n_test = pd.DataFrame(df_transformed1,  columns=column_names)
```

In [94]:
```
##Yaaha add karna hai left joins
```

In [95]:
```
df_n_test = pd.merge(left=df_n_test, right=df_pos_c_bal, how='left', left_on='SK_ID_CUR
df_n_test = pd.merge(left=df_n_test, right=df_prev_app, how='left', left_on='SK_ID_CURR
df_n_test = pd.merge(left=df_n_test, right=credit_card_bal, how='left', left_on='SK_ID_
df_n_test = pd.merge(left=df_n_test, right=new_bb, how='left', left_on='SK_ID_CURR', ri
df_n_test = pd.merge(left=df_n_test, right=df_inst, how='left', left_on='SK_ID_CURR', r
```

```
/N/u/tankulk/Carbonate/miniconda3/lib/python3.7/site-packages/ipykernel_launcher.py:5: F
utureWarning: Passing 'suffixes' which cause duplicate columns {'SK_ID_PREV_x'} in the r
esult is deprecated and will raise a MergeError in a future version.
  """
```

In [96]:
```
df_n_test['FEATURE1']= df_n_test['AMT_TOTAL_RECEIVABLE']/(df_n_test['AMT_BALANCE']+1)
df_n_test['FEATURE2'] = df_n_test['AMT_TOTAL_RECEIVABLE']/(df_n_test['AMT_RECIVABLE']+1
df_n_test['FEATURE3'] = df_n_test['AMT_TOTAL_RECEIVABLE']/(df_n_test['AMT_RECEIVABLE_PR


df_n_test['FEATURE4']=df_n_test['AMT_BALANCE']/(df_n_test['AMT_RECIVABLE']+1)
df_n_test['FEATURE5']=df_n_test['AMT_BALANCE']/(df_n_test['AMT_RECEIVABLE_PRINCIPAL']+1
df_n_test['FEATURE6']=df_n_test['AMT_RECIVABLE']/(df_n_test['AMT_RECEIVABLE_PRINCIPAL']
df_n_test['FEATURE7']=(df_n_test['EXT_SOURCE_1']+df_n_test['EXT_SOURCE_2']+df_n_test['E

df_n_test['FEATURE8']=(df_n_test['EXT_SOURCE_1']+df_n_test['EXT_SOURCE_2']+df_n_test['E
```

In [713…
```
# df_n_test['Feature10'] =  (df_n_test['EXT_SOURCE_1']*2+df_n_test['EXT_SOURCE_2']*3+df_
# df_n_test['EXT_SOURCE_MAX'] = [max(ele1,ele2,ele3) for ele1, ele2, ele3 in zip(df_n_t
```

In [97]:
```
df_n_test['FEATURE9']=(df_n_test['EXT_SOURCE_1']*df_n_test['EXT_SOURCE_2']*df_n_test['E

df_n_test['FEATURE10'] =  (df_n_test['EXT_SOURCE_1']*2+df_n_test['EXT_SOURCE_2']*3+df_n

df_n_test['FEATURE11'] = [max(ele1,ele2,ele3) for ele1, ele2, ele3 in zip(df_n_test['EX
```

In [98]:
```
df_n_test.head()
```

Out[98]:

	SK_ID_CURR	DEF_30_CNT_SOCIAL_CIRCLE	LIVINGAPARTMENTS_MODE	LIVINGAPARTMENTS_MEDI	LIV
0	100001.0	0.0	0.110874	0.107063	
1	100005.0	0.0	0.110874	0.107063	
2	100013.0	0.0	0.110874	0.107063	
3	100028.0	0.0	0.262600	0.244600	
4	100038.0	0.0	0.110874	0.107063	

5 rows × 226 columns

In [101…
```
# df_n_test.to_csv('final_application_test.csv')
```

In [627…
```
df_n_test.rename(columns={'AMT_CREDIT_y':'AMT_CREDIT'},inplace=True)
```

In [628…
```
# r = set(df_final_app_imputer.columns).difference(set(df_n_test.columns))
```

In [703…
```
df_n_test = df_n_test.apply(lambda x: x.fillna(x.median()),axis=0)
```

In [707…
```
df_n_test = df_n_test[five_percent_se_jyada]
```

In [708…
```
df_n_test
```

Out[708…

	Feature10	FEATURE10	FEATURE8	FEATURE9	EXT_SOURCE_MAX	FEATURE11	EXT_SOURCE_2	I
0	4.512270	4.512270	1.701788	0.094803	0.789654	0.789654	0.789654	
1	3.736794	3.736794	1.289607	0.071345	0.564990	0.564990	0.291656	
2	5.545685	5.545685	1.811958	0.214286	0.699787	0.699787	0.699787	
3	5.031316	5.031316	1.648115	0.164177	0.612704	0.612704	0.509677	
4	3.681774	3.681774	1.127938	0.043034	0.500106	0.500106	0.425687	
...	...	...	...	...	...	...	...	
48739	5.520187	5.520187	1.792780	0.209017	0.648575	0.648575	0.648575	

	Feature10	FEATURE10	FEATURE8	FEATURE9	EXT_SOURCE_MAX	FEATURE11	EXT_SOURCE_2	I
48740	5.056569	5.056569	1.685881	0.171589	0.684596	0.684596	0.684596	
48741	4.500163	4.500163	1.649985	0.131682	0.733503	0.733503	0.632770	
48742	4.465108	4.465108	1.414247	0.099016	0.595456	0.595456	0.445701	
48743	3.460516	3.460516	1.229854	0.062267	0.501180	0.501180	0.456541	

48744 rows × 23 columns

In [709…
```
X = df_n_test.values
```

In [ ]:
```
```

In [710…
```
def make_sub(p):
    result = p.predict(X)
    result_prob = p.predict_proba(X)
    r = pd.DataFrame(result,columns=['result'])
    r[['class_0_prob','class_1_prob']] = result_prob
    final_sub = pd.DataFrame()
    final_sub['SK_ID_CURR'] = df_test['SK_ID_CURR']
    final_sub['TARGET'] = r['class_1_prob']
    final_sub = final_sub.set_index('SK_ID_CURR')
    final_sub.to_csv('submission.csv')
```

In [712…
```
```

In [445…
```
# result = pipe_rf.predict(X)

# result_prob = pipe_rf.predict_proba(X)

# r = pd.DataFrame(result,columns=['result'])

# r[['class_0_prob','class_1_prob']] = result_prob



# final_sub = pd.DataFrame()

# final_sub['SK_ID_CURR'] = df_test['SK_ID_CURR']
# final_sub['TARGET'] = r['class_1_prob']

# final_sub = final_sub.set_index('SK_ID_CURR')

# final_sub
```

Neural Network

In [1]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader
import torchvision
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter("runs/")
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

In [3]:
```
def corr_target(df,cor):
    cor_matrix = df.corr()['TARGET'].sort_values(key=abs,ascending=False).reset_index()
    cor_matrix.columns = ['col_name','Correlation']
    column_after_corr_filter = cor_matrix[abs(cor_matrix['Correlation'])>cor]
    return column_after_corr_filter
```

In [4]:
```
train_df = pd.read_csv('final_application_train.csv')
train_df = train_df.iloc[:,1:]
```

In [5]:
```
train_df_cols = corr_target(train_df,0.10)
```

In [6]:
```
train_df = train_df[train_df_cols['col_name']]
```

In [7]:
```
train_df.shape
```

Out[7]: (307500, 7)

In [21]:
```
X = train_df.drop('TARGET',axis=1).values
y = train_df['TARGET'].values
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)

X_train = torch.FloatTensor(X_train)
X_test = torch.FloatTensor(X_test)
y_train = torch.tensor(y_train, dtype=torch.long, device=device)
y_test = torch.tensor(y_test, dtype=torch.long, device=device)
```

In [22]:
```
# X_train = train_df.drop('TARGET',axis=1).values
# y_train = train_df['TARGET'].values
# X_train = torch.FloatTensor(X_train)
# y_train = torch.tensor(y_train, dtype=torch.long, device=device)
```

In [23]:
```
class ANN_model(nn.Module):
    def __init__(self,input_features=6,hidden1=128,hidden2=64,hidden3=32,hidden4=10,out
```

```
        super().__init__()
        self.f_connected1 = nn.Linear(input_features,hidden1)
        self.f_connected2 = nn.Linear(hidden1,hidden2)
        self.f_connected3 = nn.Linear(hidden2,hidden3)
        self.f_connected4 = nn.Linear(hidden3,hidden4)
        self.out = nn.Linear(hidden4,out_features)
    def forward(self,x):
        x = F.leaky_relu(self.f_connected1(x))
        x = F.dropout(x,p=.4)
        x = F.leaky_relu(self.f_connected2(x))
        x = F.leaky_relu(self.f_connected3(x))
        x = F.leaky_relu(self.f_connected4(x))
        x = self.out(x)
        return x
```

In [25]:

```
torch.manual_seed(20)
model= ANN_model().to(device)
```

In [26]:

```
loss_function = nn.CrossEntropyLoss()
#optimizer = torch.optim.Adadelta(model.parameters(),lr=0.001)

optimizer = torch.optim.Adam(model.parameters(),lr=0.001)
# optimizer = torch.optim.Adamax(model.parameters(),lr=0.001)
```

In [27]:

```
size_ = y_train.shape[0]
epochs = 1000
final_losses = []
for i in range(epochs):
    i += 1
    y_pred = model.forward(X_train)
    loss = loss_function(y_pred,y_train)
    final_losses.append(loss.item())
    _,predicted = torch.max(y_pred,1)
    running_correct = (predicted==y_train).sum().item()
    if i%100 == 1:
        print("Epoch Number: {} and the loss: {} accuracy : {}".format(i,loss.item(),ru
        writer.add_scalar('Training loss',loss.item(),i)
        writer.add_scalar('Accuracy',running_correct/size_,i)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```
Epoch Number: 1 and the loss: 0.5532795786857605 accuracy : 0.9195243902439024
Epoch Number: 101 and the loss: 0.25988706946372986 accuracy : 0.9195243902439024
Epoch Number: 201 and the loss: 0.2582740783691406 accuracy : 0.9195243902439024
Epoch Number: 301 and the loss: 0.2573358416557312 accuracy : 0.9195243902439024
Epoch Number: 401 and the loss: 0.25660866498947144 accuracy : 0.9195243902439024
Epoch Number: 501 and the loss: 0.25646641850471497 accuracy : 0.9195243902439024
Epoch Number: 601 and the loss: 0.2560791075229645 accuracy : 0.9195243902439024
Epoch Number: 701 and the loss: 0.2559730112552643 accuracy : 0.9195325203252033
Epoch Number: 801 and the loss: 0.2557643949985504 accuracy : 0.9195447154471544
Epoch Number: 901 and the loss: 0.25565895438194275 accuracy : 0.9195325203252033
```

In [28]:

```
predictions = []
with torch.no_grad():
    for i,data in enumerate(X_test):
```

```
        y_pred = model(data)
#         print(F.softmax(y_pred))
        predictions.append(y_pred.argmax().item())
#         print(y_pred.argmax().item())
```

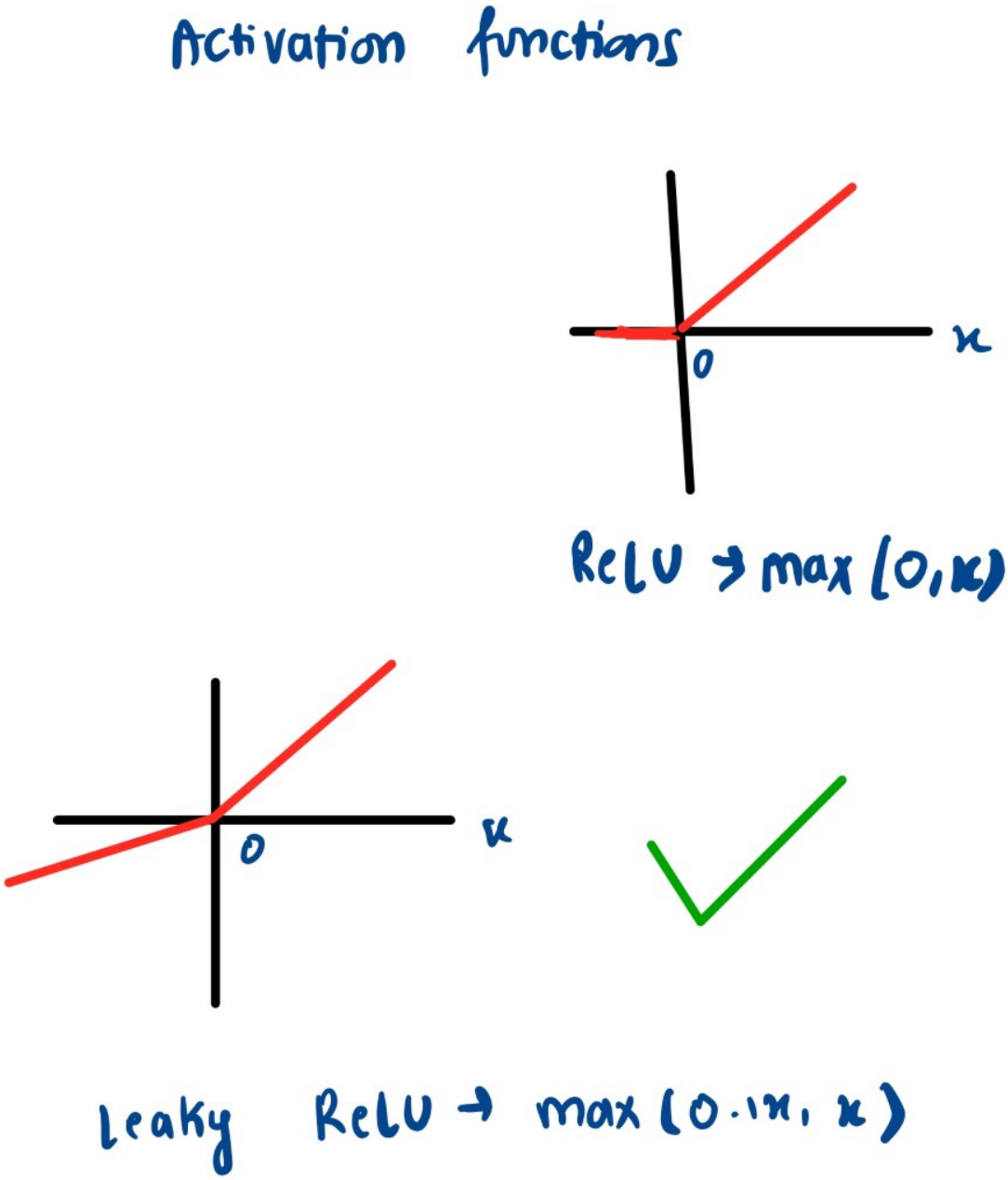
In [29]:
```
from sklearn.metrics import confusion_matrix, roc_auc_score
cm = confusion_matrix(y_test,predictions)
cm
```

Out[29]:
```
array([[56470,     4],
       [ 5021,     5]])
```

In [30]:
```
from sklearn.metrics import accuracy_score
score = accuracy_score(y_test,predictions)
score
```

Out[30]:
```
0.9182926829268293
```

RESULTS AND DICUSSION: NEURAL NETWORK



In [31]:
```
test_df = pd.read_csv('final_application_test.csv')
test_df = test_df.iloc[:,1:]
```

In [32]:
```
test_cols = list(train_df.columns)
test_cols.remove('TARGET')
```

In [33]:
```
testing = torch.FloatTensor(test_df[test_cols].values)
```

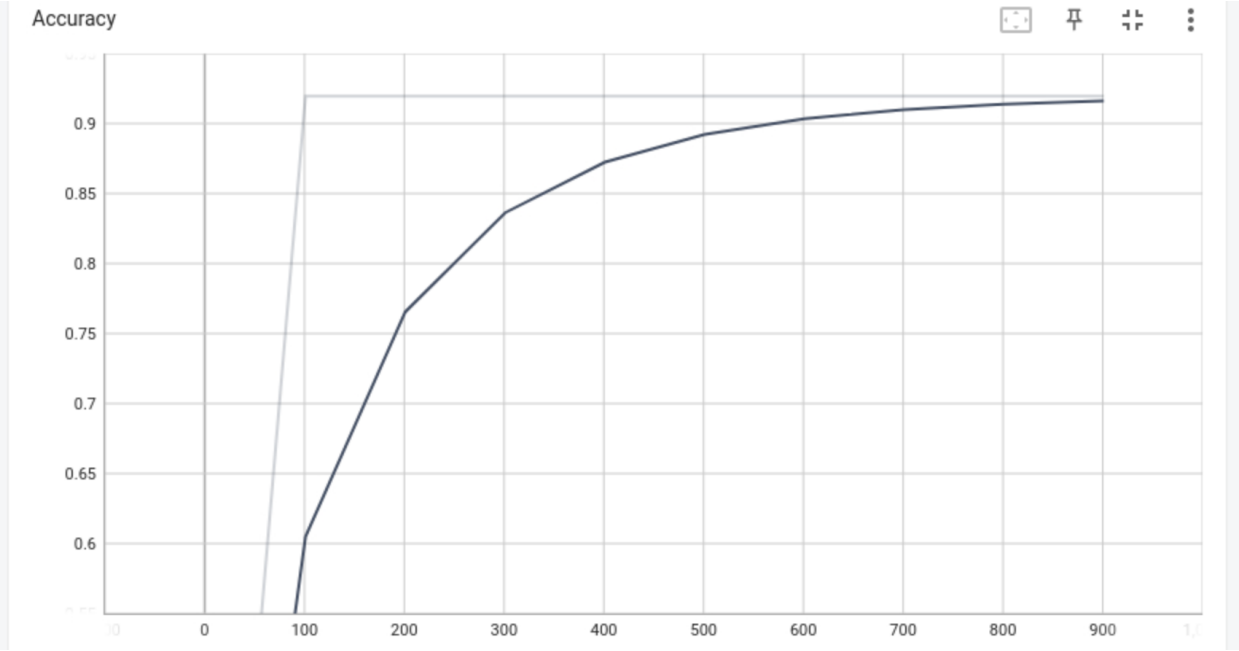
In [34]:
```
predictions = []
probs = []
with torch.no_grad():
    for i,data in enumerate(testing):
        y_pred = model(data)
#         print(y_pred)
#         print(F.softmax(y_pred)[1].item())
        probs.append(F.softmax(y_pred)[1].item())
        predictions.append(y_pred.argmax().item())
#         print(y_pred.argmax().item())
```

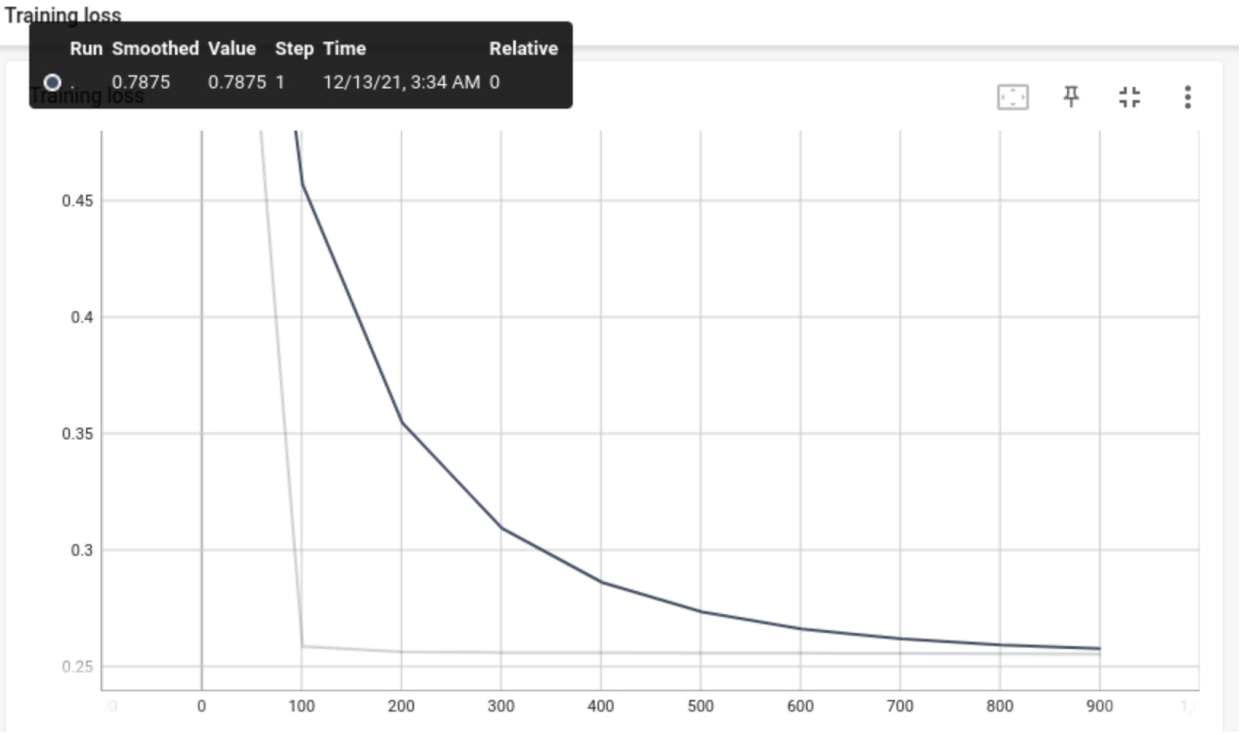
```
/N/u/tankulk/Carbonate/miniconda3/lib/python3.7/site-packages/ipykernel_launcher.py:8: U
serWarning: Implicit dimension choice for softmax has been deprecated. Change the call t
o include dim=X as an argument.
```

In [35]:
```
sub = pd.DataFrame()

sub["SK_ID_CURR"]=test_df['SK_ID_CURR'].astype(int)
sub['TARGET'] = probs
sub.set_index('SK_ID_CURR')


sub.to_csv('submission.csv',index=False)
```

On implementation of Neural Network for modeling the HCDR dataset we got a trainng accuracy of 91.95% and a test roc score of 0.71225. Here, we used the features that we created in the previous phase of the project and selected features just with high correlation as strong predictors that we would train our neural network with. After trying different combinations of hiddem layers and the number of neurons in them we found that 4 hidden layers with 128,64,32,10 neurons respectively worked the best with an output layer having 2 neurons. We tried out 3 optimizers namely AdaDelta, Adam and RMSprop where we found that the Adam optimizer worked the best for us. Since we did not want to overtrain and overfit the model we limited the epochs to 1000 after which we did not find any significant difference for the increase in accuracy.

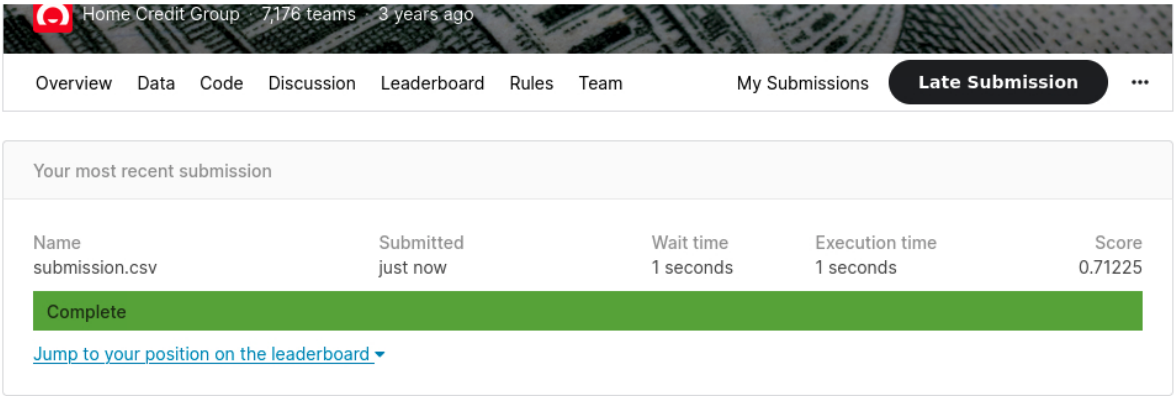
Here are some Tensorboard visualizations for Accuracy and Loss while training



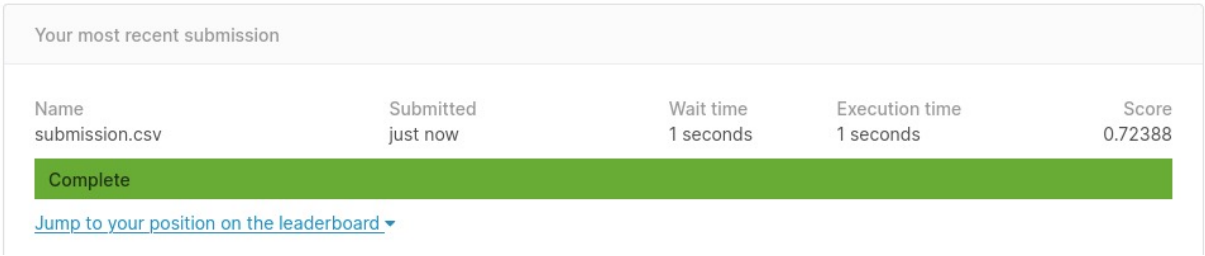


Kaggle Test Accuracy

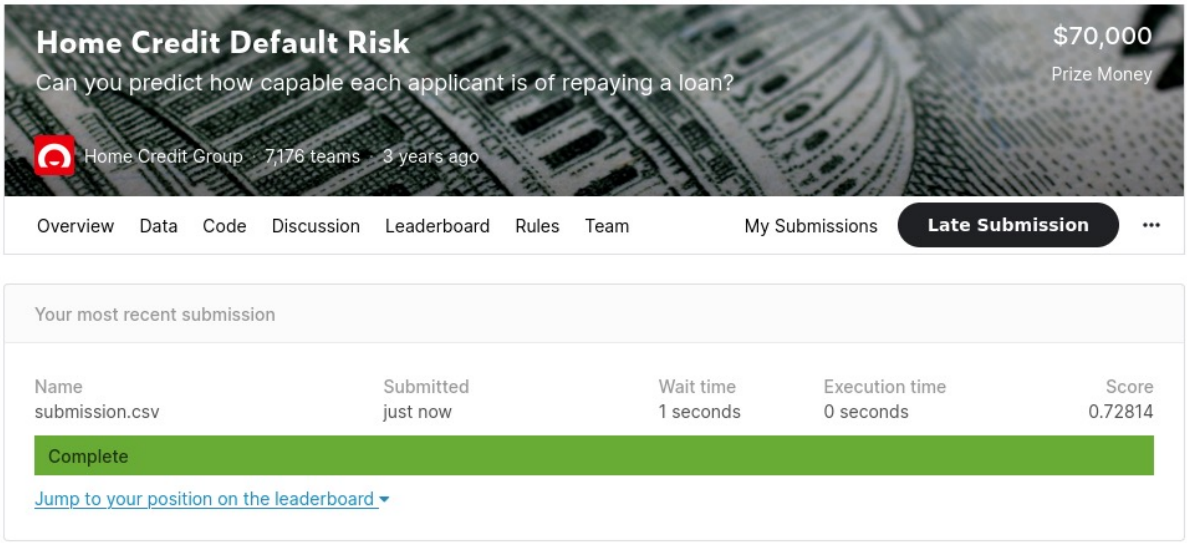
For Neural Network



For Logistic Regression



For Random Forest Classifier



With high training accuracy and a low test accuracy we can see that the Random forest classifier seems to be over fitting and thus the ideal choice of model would be Logistic Regression

CONCLUSION

Our project focuses on predicting whether the credit-less population are able to pay back their loans or not. In order to make this possible we source our data from the Home Credit dataset. It is very important that this population also gets a fair chance of getting a loan and as we being students can highly relate to this. Hence we decide to pursue this project. In previous phases, we understood the data by performing exploratory data analysis and visualization. After which we perform pre-processing and clean the data accordingly.We featured the data performing OHE and applied imputing methods to fix the data before feeding it to the model. In phase 2 of this project we had performed feature engineering by adding new columns to the model for improved accuracy and performed hyperparameter tuning to find out the best setting and parameters for the models and were able to achieve the test ROC Score of 0.728 using random forest classifier and training accuracy of 92.4%. We have used Logistic Regression and Random Forests Classifier models. Finally we have evaluated the results using accuracy score, log loss, confusion matrix and ROC AUC scores. In this last phase we implemented a Multilayer Perceptron(MLP) model using Pytorch for loan default classification. We found out the accuracy for the MLP model to be 91.95% and a test roc score of 0.71225 which is pretty close to our previous non deep learning models. Deep Learning models require huge amount of data to train itself and thus on a longer run Deep Learning models would work best for HCDR classification as compared to usual supervised models. The future scope for this project can include using embeddings in deep learning models or using some advanced classification models like lightGBM/other boosting models that can produce better results.

References:https://www.kaggle.com/c/home-credit-default-risk/data