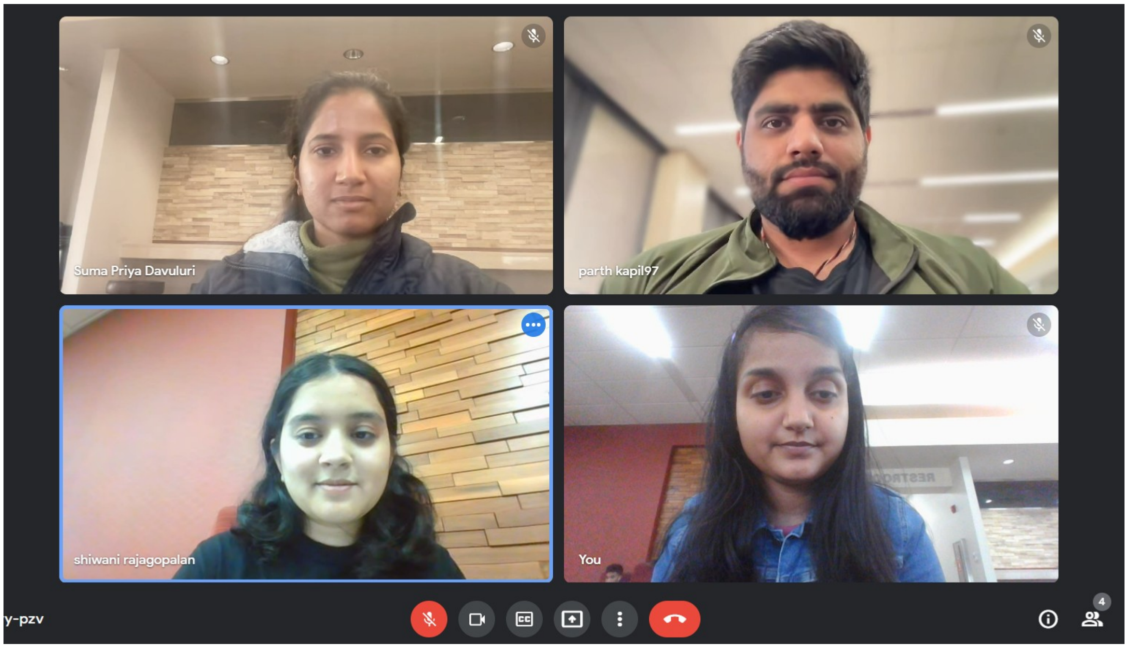
Home Credit Default Risk (HCDR)

Group Members:

- Parth Kapil(pkapil@iu.edu)
- Shiwani Rajagopalan(srajago@iu.edu)
- Shubhangi Mishra(shubmish@iu.edu)
- Suma Priya Davuluri(sdavulu@iu.edu



The course project is based on the Home Credit Default Risk (HCDR) Kaggle Competition. The goal of this project is to predict whether or not a client will repay a loan. In order to make sure that people who struggle to get loans due to insufficient or non-existent credit histories have a positive loan experience, Home Credit makes use of a variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities.

Some of the challenges

1. Dataset size
   - (688 meg uncompressed) with millions of rows of data
   - 2.71 Gig of data uncompressed
- Dealing with missing data
- Imbalanced datasets
- Summarizing transaction data

Kaggle API setup

Kaggle is a Data Science Competition Platform which shares a lot of datasets. In the past, it was troublesome to submit your result as your have to go through the console in your browser and drag your files there. Now you can interact with Kaggle via the command line. E.g.,

```
! kaggle competitions files home-credit-default-risk
```

It is quite easy to setup, it takes me less than 15 minutes to finish a submission.

1. Install library

- Create a API Token (edit your profile on Kaggle.com); this produces `kaggle.json` file
- Put your JSON `kaggle.json` in the right place
- Access competition files; make submissions via the command (see examples below)
- Submit result

For more detailed information on setting the Kaggle API see here and here.

```
!pip install kaggle
```

```
Requirement already satisfied: kaggle in
/usr/local/lib/python3.9/site-packages (1.5.12)
Requirement already satisfied: requests in
/usr/local/lib/python3.9/site-packages (from kaggle) (2.26.0)
Requirement already satisfied: six>=1.10 in
/usr/local/lib/python3.9/site-packages (from kaggle) (1.15.0)
Requirement already satisfied: urllib3 in
/usr/local/lib/python3.9/site-packages (from kaggle) (1.26.7)
Requirement already satisfied: python-slugify in
/usr/local/lib/python3.9/site-packages (from kaggle) (5.0.2)
Requirement already satisfied: python-dateutil in
/usr/local/lib/python3.9/site-packages (from kaggle) (2.8.2)
Requirement already satisfied: certifi in
/usr/local/lib/python3.9/site-packages (from kaggle) (2021.10.8)
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/site-
packages (from kaggle) (4.62.3)
Requirement already satisfied: text-unidecode>=1.3 in
/usr/local/lib/python3.9/site-packages (from python-slugify->kaggle)
(1.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in
/usr/local/lib/python3.9/site-packages (from requests->kaggle) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.9/site-packages (from requests->kaggle) (3.3)
WARNING: Running pip as the 'root' user can result in broken
permissions and conflicting behaviour with the system package manager.
It is recommended to use a virtual environment instead:
https://pip.pypa.io/warnings/venv
WARNING: You are using pip version 21.3.1; however, version 22.0.4 is
```

```
available.
You should consider upgrading via the '/usr/local/bin/python -m pip
install --upgrade pip' command.
```

```
!pwd
```

```
/root/shared/Downloads
```

```
!mkdir ~/.kaggle
!cp /root/shared/Downloads/kaggle.json ~/.kaggle
!chmod 600 ~/.kaggle/kaggle.json
```

```
cp: cannot stat '/root/shared/Downloads/kaggle.json': No such file or
directory
chmod: cannot access '/root/.kaggle/kaggle.json': No such file or
directory
```

```
! kaggle competitions files home-credit-default-risk
```

```
Traceback (most recent call last):
  File "/usr/local/bin/kaggle", line 5, in <module>
    from kaggle.cli import main
  File "/usr/local/lib/python3.9/site-packages/kaggle/__init__.py",
line 23, in <module>
    api.authenticate()
  File
"/usr/local/lib/python3.9/site-packages/kaggle/api/kaggle_api_extended
.py", line 164, in authenticate
    raise IOError('Could not find {}. Make sure it\'s located in'
OSError: Could not find kaggle.json. Make sure it's located in
/root/.kaggle. Or use the environment method.
```

Dataset and how to download

Back ground Home Credit Group

Many people struggle to get loans due to insufficient or non-existent credit histories. And, unfortunately, this population is often taken advantage of by untrustworthy lenders.

Home Credit Group

Home Credit strives to broaden financial inclusion for the unbanked population by providing a positive and safe borrowing experience. In order to make sure this underserved population has a positive loan experience, Home Credit makes use of a variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities.

While Home Credit is currently using various statistical and machine learning methods to make these predictions, they're challenging Kagglers to help them unlock the full potential of their data. Doing so will ensure that clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful.

Background on the dataset

Home Credit is a non-banking financial institution, founded in 1997 in the Czech Republic.

The company operates in 14 countries (including United States, Russia, Kazahstan, Belarus, China, India) and focuses on lending primarily to people with little or no credit history which will either not obtain loans or became victims of untrustworthly lenders.

Home Credit group has over 29 million customers, total assests of 21 billions Euro, over 160 millions loans, with the majority in Asia and and almost half of them in China (as of 19-05-2018).

While Home Credit is currently using various statistical and machine learning methods to make these predictions, they're challenging Kagglers to help them unlock the full potential of their data. Doing so will ensure that clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful.

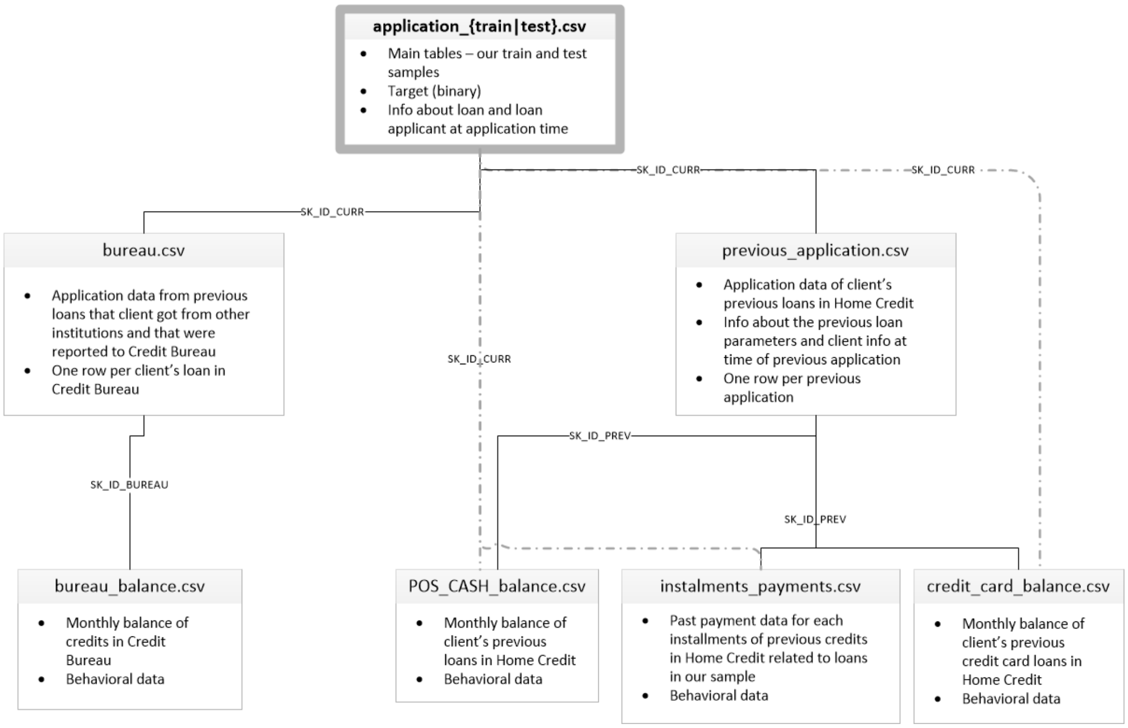
Data files overview

There are 7 different sources of data:

- **application_train/application_test:** the main training and testing data with information about each loan application at Home Credit. Every loan has its own row and is identified by the feature SK_ID_CURR. The training application data comes with the TARGET indicating **0: the loan was repaid** or **1: the loan was not repaid**. The target variable defines if the client had payment difficulties meaning he/she had late payment more than X days on at least one of the first Y installments of the loan. Such case is marked as 1 while other all other cases as 0.
- **bureau:** data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau, but one loan in the application data can have multiple previous credits.
- **bureau_balance:** monthly data about the previous credits in bureau. Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length.
- **previous_application:** previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV.
- **POS_CASH_BALANCE:** monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.

- credit_card_balance: monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.
- **installments_payment:** payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

```
# ![alt](home_credit.png "Home credit")
```



Downloading the files via Kaggle API

Create a base directory:

```
DATA_DIR = "../../../Data/home-credit-default-risk"   #same level as course repo in the data directory
```

Please download the project data files and data dictionary and unzip them using either of the following approaches:

1. Click on the `Download` button on the following Data Webpage and unzip the zip file to the `BASE_DIR`
2. If you plan to use the Kaggle API, please use the following steps.

Home Credit Default Risk (HCDR)

The course project is based on the Home Credit Default Risk (HCDR) Kaggle Competition. The goal of this project is to predict whether or not a client will repay a loan. In order to make sure that people who struggle to get loans due to insufficient or non-existent credit histories have a positive loan experience, Home Credit makes use of a variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities.

Some of the challenges

1. Dataset size
   - (688 meg uncompressed) with millions of rows of data
   - 2.71 Gig of data uncompressed
- Dealing with missing data
- Imbalanced datasets
- Summarizing transaction data

Kaggle API setup

Kaggle is a Data Science Competition Platform which shares a lot of datasets. In the past, it was troublesome to submit your result as your have to go through the console in your browser and drag your files there. Now you can interact with Kaggle via the command line. E.g.,

```
! kaggle competitions files home-credit-default-risk
```

It is quite easy to setup, it takes me less than 15 minutes to finish a submission.

1. Install library
- Create a API Token (edit your profile on Kaggle.com); this produces `kaggle.json` file
- Put your JSON `kaggle.json` in the right place
- Access competition files; make submissions via the command (see examples below)
- Submit result

For more detailed information on setting the Kaggle API see here and here.

!pip install kaggle

!pwd

!mkdir ~/.kaggle !cp /root/shared/Downloads/kaggle.json ~/.kaggle !chmod 600 ~/.kaggle/kaggle.json

! kaggle competitions files home-credit-default-risk

Dataset and how to download

Back ground Home Credit Group

Many people struggle to get loans due to insufficient or non-existent credit histories. And, unfortunately, this population is often taken advantage of by untrustworthy lenders.

Home Credit Group

Home Credit strives to broaden financial inclusion for the unbanked population by providing a positive and safe borrowing experience. In order to make sure this underserved population has a positive loan experience, Home Credit makes use of a variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities.

While Home Credit is currently using various statistical and machine learning methods to make these predictions, they're challenging Kagglers to help them unlock the full potential of their data. Doing so will ensure that clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful.

Background on the dataset

Home Credit is a non-banking financial institution, founded in 1997 in the Czech Republic.

The company operates in 14 countries (including United States, Russia, Kazahstan, Belarus, China, India) and focuses on lending primarily to people with little or no credit history which will either not obtain loans or became victims of untrustworthly lenders.

Home Credit group has over 29 million customers, total assests of 21 billions Euro, over 160 millions loans, with the majority in Asia and and almost half of them in China (as of 19-05-2018).

While Home Credit is currently using various statistical and machine learning methods to make these predictions, they're challenging Kagglers to help them unlock the full potential of their data. Doing so will ensure that clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful.

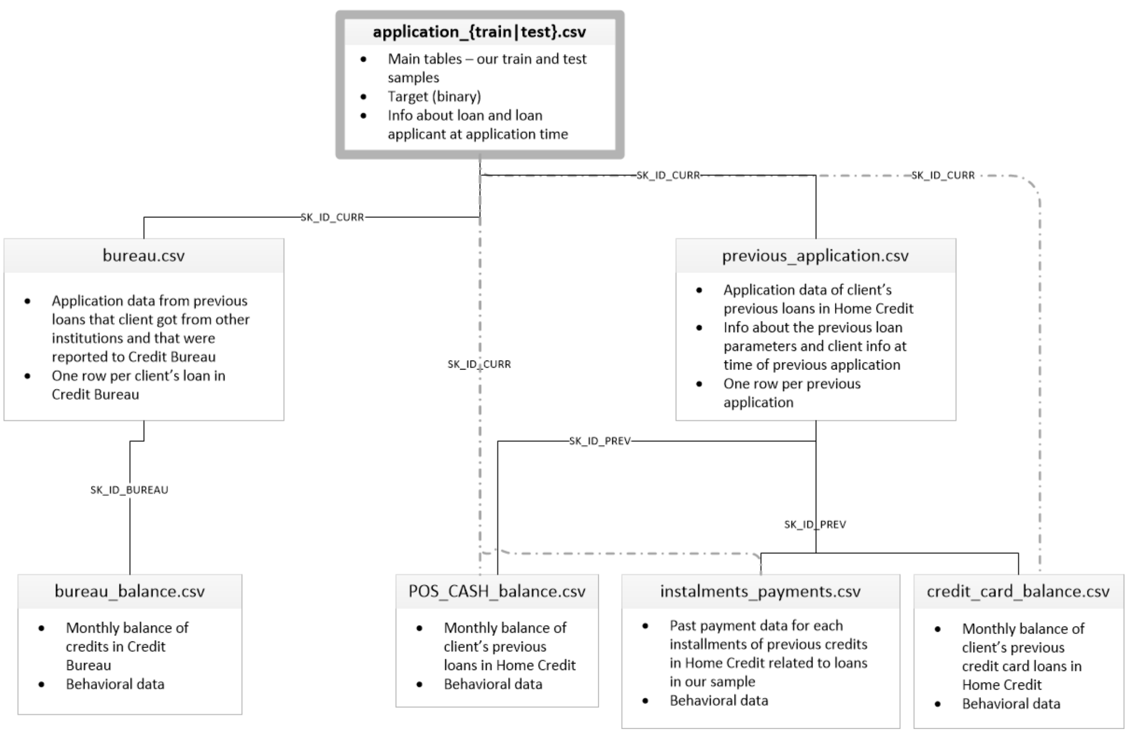
Data files overview

There are 7 different sources of data:

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

- **bureau:** data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau, but one loan in the application data can have multiple previous credits.
- **bureau_balance:** monthly data about the previous credits in bureau. Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length.
- **previous_application:** previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV.
- **POS_CASH_BALANCE:** monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.
- credit_card_balance: monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.
- **installments_payment:** payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

alt



Downloading the files via Kaggle API

Create a base directory:

```
DATA_DIR = "../../../Data/home-credit-default-risk"   #same level as course repo in the data directory
```

Please download the project data files and data dictionary and unzip them using either of the following approaches:

1. Click on the `Download` button on the following Data Webpage and unzip the zip file to the `BASE_DIR`
2. If you plan to use the Kaggle API, please use the following steps. import numpy as np import pandas as pd from sklearn.preprocessing import LabelEncoder import missingno as msno import os import zipfile from sklearn.base import BaseEstimator, TransformerMixin import matplotlib.pyplot as plt import seaborn as sns from sklearn.linear_model import LogisticRegression from sklearn.model_selection import train_test_split from sklearn.model_selection import KFold from sklearn.model_selection import cross_val_score from sklearn.model_selection import GridSearchCV from sklearn.impute import SimpleImputer from sklearn.preprocessing import MinMaxScaler from sklearn.pipeline import Pipeline, FeatureUnion from pandas.plotting import scatter_matrix from sklearn.preprocessing import StandardScaler from sklearn.ensemble import RandomForestClassifier from sklearn.preprocessing import OneHotEncoder import warnings import gc warnings.filterwarnings('ignore')

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
!pip install missingno
import missingno as msno
import os
import zipfile
from sklearn.base import BaseEstimator, TransformerMixin
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline, FeatureUnion
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import OneHotEncoder
import warnings
```

```
import gc
warnings.filterwarnings('ignore')
```

```
Collecting missingno
  Downloading missingno-0.5.1-py3-none-any.whl (8.7 kB)
Requirement already satisfied: matplotlib in c:\users\shiwani\anaconda3\lib\site-packages (from missingno) (3.4.2)
Requirement already satisfied: numpy in c:\users\shiwani\anaconda3\lib\site-packages (from missingno) (1.20.3)
Requirement already satisfied: seaborn in c:\users\shiwani\anaconda3\lib\site-packages (from missingno) (0.11.2)
Requirement already satisfied: scipy in c:\users\shiwani\anaconda3\lib\site-packages (from missingno) (1.6.2)
Requirement already satisfied: cycler>=0.10 in c:\users\shiwani\anaconda3\lib\site-packages (from matplotlib->missingno) (0.10.0)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\shiwani\anaconda3\lib\site-packages (from matplotlib->missingno) (2.4.7)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\shiwani\anaconda3\lib\site-packages (from matplotlib->missingno) (2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\shiwani\anaconda3\lib\site-packages (from matplotlib->missingno) (1.3.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\shiwani\anaconda3\lib\site-packages (from matplotlib->missingno) (8.4.0)
Requirement already satisfied: six in c:\users\shiwani\anaconda3\lib\site-packages (from cycler>=0.10->matplotlib->missingno) (1.16.0)
Requirement already satisfied: pandas>=0.23 in c:\users\shiwani\anaconda3\lib\site-packages (from seaborn->missingno) (1.3.4)
Requirement already satisfied: pytz>=2017.3 in c:\users\shiwani\anaconda3\lib\site-packages (from pandas>=0.23->seaborn->missingno) (2021.1)
Installing collected packages: missingno
Successfully installed missingno-0.5.1
```

```
WARNING: Ignoring invalid distribution -ywin32 (c:\users\shiwani\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -ywin32 (c:\users\shiwani\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -ywin32 (c:\users\shiwani\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -ywin32 (c:\users\shiwani\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -ywin32 (c:\users\shiwani\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -ywin32 (c:\users\shiwani\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -ywin32 (c:\users\shiwani\anaconda3\lib\site-packages)
```

Helper functions for EDA

```
#To make it easier to get results quicker we tried to optimize the
memory of dataset, for that we are
# using this amazing solution we found on the kaggle
# https://www.kaggle.com/rinnqd/reduce-memory-usage
def optimize_memory(df):

    mem_before = df.memory_usage().sum() / 1024**2
    print("Before Optimization : DataFrame Memory "+ str(mem_before))

    for col in df.columns:
        col_type = df[col].dtype

        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                #Check if  Column can be interpreted using int8
                if c_min > np.iinfo(np.int8).min and c_max <
np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                #Check if  Column can be interpreted using int16
                elif c_min > np.iinfo(np.int16).min and c_max <
np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                #Check if  Column can be interpreted using int32
                elif c_min > np.iinfo(np.int32).min and c_max <
np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                #Use  Int64 if no conditions match
                else:
                    df[col] = df[col].astype(np.int64)
            else:
                #Check if  Column can be interpreted using Float 16
                if c_min > np.finfo(np.float16).min and c_max <
np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                #Check if  Column can be interpreted using float32
                elif c_min > np.finfo(np.float32).min and c_max <
np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                #Use float64 instead
                else:
                    df[col] = df[col].astype(np.float64)

    mem_after = df.memory_usage().sum() / 1024**2
    print("After Optimization : DataFrame Memory "+ str(mem_after))
    return df
```

```python
# For one hot encoding categorical features
def ohe(df):
    cat_feature=df.select_dtypes(include='object')
    cat_fetature_cols=cat_feature.columns
    df=pd.get_dummies(df,columns=cat_fetature_cols,dummy_na=False)
    return df

# rename columns in the dataframe
def rename(df,name):
  df.columns=pd.Index([name + "_"+ col for col in list(df.columns)])
  df.rename(columns={name+"_SK_ID_CURR":"SK_ID_CURR"},inplace=True)


#function for loading data
def load_csv(path, name):
    df = optimize_memory(pd.read_csv(path))
    print(f"{name}: shape: {df.shape}")
    return df

#function for checking the feature types in the data frame'
def feature_type(data):
    cat_feat = data.select_dtypes(include = ["object"]).columns
    num_feat = data.select_dtypes(exclude = ["object"]).columns


    print("numerical features:",num_feat)
    print('*'*100)
    print( "categorical features :",cat_feat)


#finding missing values and their percentage in the dataframe
def missingFeatures(data):
    total = data.isnull().sum().sort_values(ascending = False)
    percent =
(data.isnull().sum()/data.isnull().count()*100).sort_values(ascending
= False)
    ms=pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
    ms= ms[ms["Percent"] > 0]
    f,ax =plt.subplots(figsize=(15,10))
    plt.xticks(rotation='90')
    fig=sns.barplot(ms.index, ms["Percent"],color="red",alpha=0.8)
    plt.xlabel('Features', fontsize=15)
    plt.ylabel('Percent of missing values', fontsize=15)
    plt.title('Percent missing data by feature', fontsize=15)
    #ms= ms[ms["Percent"] > 0]
    return ms

#function for plotting relationship between features
def getRelationship(df,val1='',val2=''):
    f,ax=plt.subplots(1,2,figsize=(10,6))
```

```
df[[val1,val2]].groupby([val1]).count().plot.bar(ax=ax[0],color='red')
    ax[0].set_title('Customer counts Based on '+val1)
    sns.countplot(val1,hue=val2,data=df,ax=ax[1],palette="bright")
    ax[1].set_title(val1+': Unpaid vs Paid')
    plt.xticks(rotation=-90)
    plot=plt.show()
    return plot
```

Description about data

```
#dataset names
dataset_names = ["POS_CASH_balance", "application_train",
"application_test", "bureau",
"bureau_balance","credit_card_balance","installments_payments",
            "previous_application"]

#Reading all the data
DATA_DIR = "./hcdr"
datasets={}
for name in dataset_names:
    datasets[name] = load_csv(os.path.join(DATA_DIR, f'{name}.csv'),
name)
```

```
Before Optimization : DataFrame Memory 610.4345703125
After Optimization : DataFrame Memory 238.451078414917
POS_CASH_balance: shape: (10001358, 8)
Before Optimization : DataFrame Memory 286.2270965576172
After Optimization : DataFrame Memory 92.37870502471924
application_train: shape: (307511, 122)
Before Optimization : DataFrame Memory 44.99847412109375
After Optimization : DataFrame Memory 14.596694946289062
application_test: shape: (48744, 121)
Before Optimization : DataFrame Memory 222.62033081054688
After Optimization : DataFrame Memory 112.94713973999023
bureau: shape: (1716428, 17)
Before Optimization : DataFrame Memory 624.845817565918
After Optimization : DataFrame Memory 338.45820713043213
bureau_balance: shape: (27299925, 3)
Before Optimization : DataFrame Memory 673.8829956054688
After Optimization : DataFrame Memory 289.3302688598633
credit_card_balance: shape: (3840312, 23)
Before Optimization : DataFrame Memory 830.4078979492188
After Optimization : DataFrame Memory 311.40303802490234
installments_payments: shape: (13605401, 8)
Before Optimization : DataFrame Memory 471.48081970214844
After Optimization : DataFrame Memory 309.0111198425293
previous_application: shape: (1670214, 37)
```

Exploratory Data Analysis

POS_CASH_balance

```
datasets["POS_CASH_balance"].describe()
```

```
         SK_ID_PREV    SK_ID_CURR  MONTHS_BALANCE  CNT_INSTALMENT  \
count  1.000136e+07  1.000136e+07    1.000136e+07       9975287.0
mean   1.903217e+06  2.784039e+05   -3.501259e+01             NaN
std    5.358465e+05  1.027637e+05    2.606657e+01             0.0
min    1.000001e+06  1.000010e+05   -9.600000e+01             1.0
25%    1.434405e+06  1.895500e+05   -5.400000e+01            10.0
50%    1.896565e+06  2.786540e+05   -2.800000e+01            12.0
75%    2.368963e+06  3.674290e+05   -1.300000e+01            24.0
max    2.843499e+06  4.562550e+05   -1.000000e+00            92.0

       CNT_INSTALMENT_FUTURE        SK_DPD    SK_DPD_DEF
count              9975271.0  1.000136e+07  1.000136e+07
mean                     NaN  1.160693e+01  6.544684e-01
std                      0.0  1.327140e+02  3.276249e+01
min                      0.0  0.000000e+00  0.000000e+00
25%                      3.0  0.000000e+00  0.000000e+00
50%                      7.0  0.000000e+00  0.000000e+00
75%                     14.0  0.000000e+00  0.000000e+00
max                     85.0  4.231000e+03  3.595000e+03
```

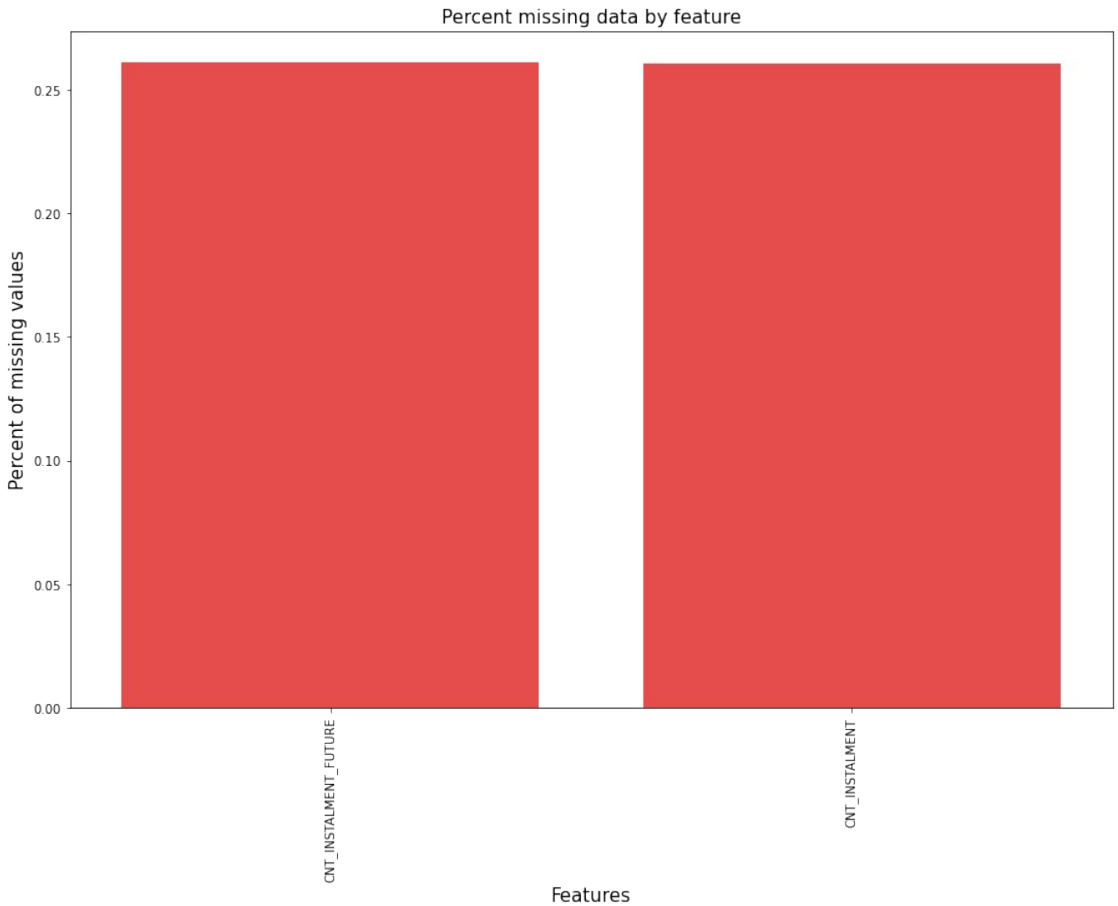
Grouping featrues by type

```
feature_type(datasets["POS_CASH_balance"])
```

```
numerical features: Index(['SK_ID_PREV', 'SK_ID_CURR',
'MONTHS_BALANCE', 'CNT_INSTALMENT',
       'CNT_INSTALMENT_FUTURE', 'SK_DPD', 'SK_DPD_DEF'],
      dtype='object')
**********************************************************************
******************************
categorical features : Index(['NAME_CONTRACT_STATUS'], dtype='object')
```

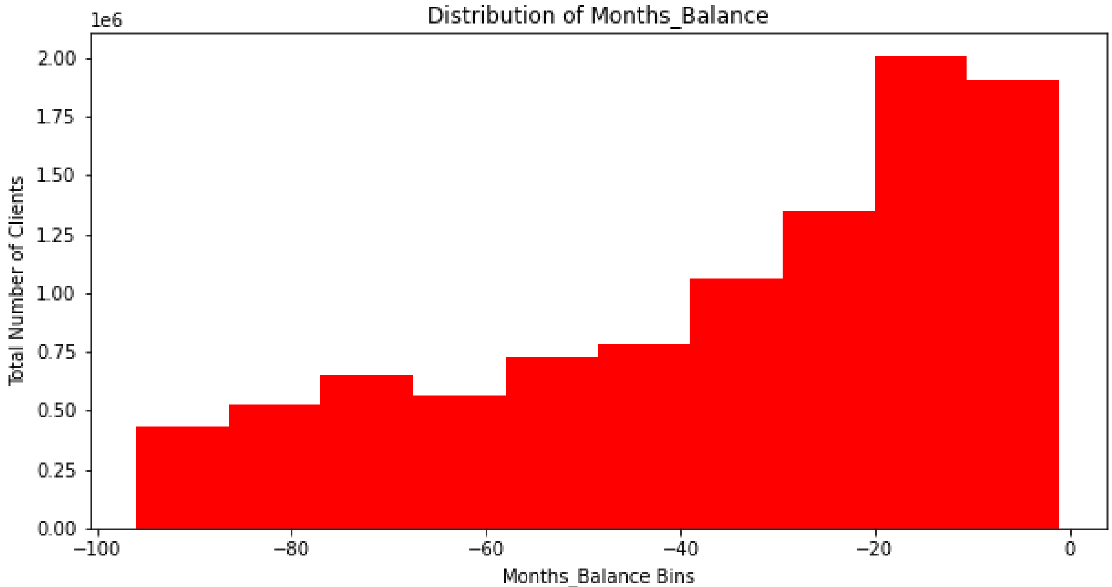
```
missingFeatures(datasets["POS_CASH_balance"])
```

```
                       Total   Percent
CNT_INSTALMENT_FUTURE  26087  0.260835
CNT_INSTALMENT         26071  0.260675
```



Realtionship between cash balance with months balance

```
plt.figure(figsize=(10,5))
plt.hist(datasets['POS_CASH_balance'][['MONTHS_BALANCE']].values,
bins=10,color='red',label=True)
plt.title('Distribution of Months_Balance')
plt.xlabel('Months_Balance Bins')
plt.ylabel('Total Number of Clients')
plt.show()
```

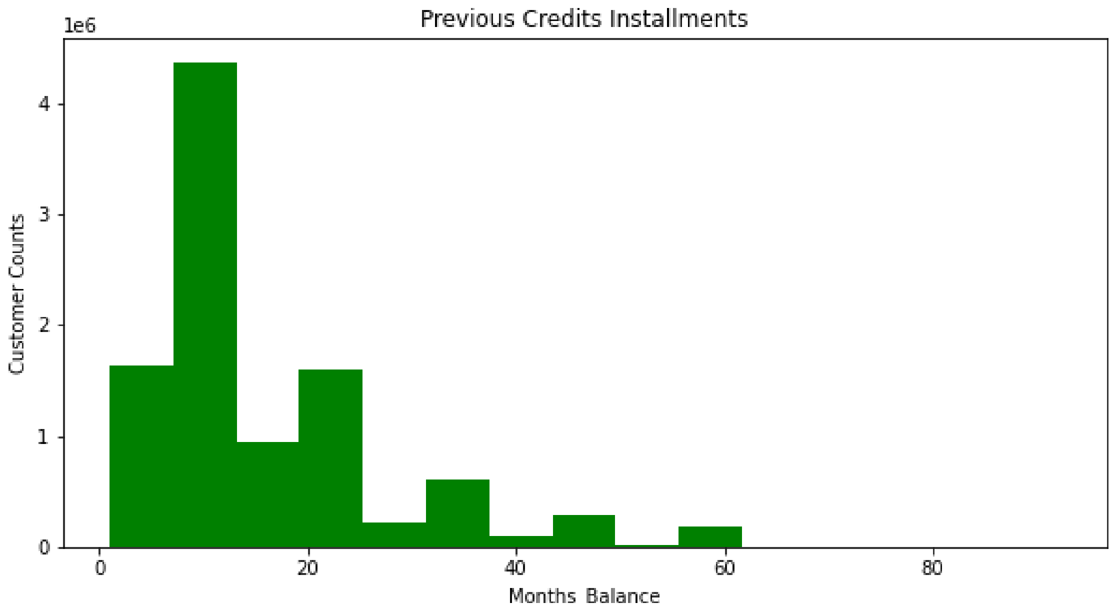


Observation

Most of the customers have non zero month balance.

Customer counts on the baiss of Instalment counts

```
plt.figure(figsize=(10,5))
plt.hist(datasets['POS_CASH_balance'][['CNT_INSTALMENT']].values,
bins=15,color='green',label=True)
plt.title('Previous Credits Installments')
plt.xlabel('Months_Balance')
plt.ylabel('Customer Counts')
plt.show()
```



Observation

Most of the customers with previous credit installments have positive month balance.

application_train EDA

```
datasets['application_train'].describe()
```

```
          SK_ID_CURR         TARGET   CNT_CHILDREN
AMT_INCOME_TOTAL  \
count  307511.000000  307511.000000  307511.000000      3.075110e+05

mean   278180.518577       0.080729       0.417052      1.687391e+05

std    102790.175348       0.272419       0.722121      2.371759e+05

min    100002.000000       0.000000       0.000000      2.565000e+04

25%    189145.500000       0.000000       0.000000      1.125000e+05

50%    278202.000000       0.000000       0.000000      1.471500e+05

75%    367142.500000       0.000000       1.000000      2.025000e+05

max    456255.000000       1.000000      19.000000      1.170000e+08


         AMT_CREDIT    AMT_ANNUITY  AMT_GOODS_PRICE  \
count  3.075110e+05  307499.000000     3.072330e+05
mean   5.988308e+05   27110.958984     5.379796e+05
std    4.024795e+05   14493.233398     3.695427e+05
```

```
min    4.500000e+04    1615.500000     4.050000e+04
25%    2.700000e+05   16524.000000     2.385000e+05
50%    5.135310e+05   24903.000000     4.500000e+05
75%    8.086500e+05   34596.000000     6.795000e+05
max    4.050000e+06  258025.500000     4.050000e+06

       REGION_POPULATION_RELATIVE     DAYS_BIRTH
DAYS_EMPLOYED  ...  \
count                307511.000000  307511.000000  307511.000000  ...

mean                      0.020859  -16036.995067   63815.045904  ...

std                       0.013824    4363.988632  141275.766519  ...

min                       0.000290  -25229.000000  -17912.000000  ...

25%                       0.010010  -19682.000000   -2760.000000  ...

50%                       0.018845  -15750.000000   -1213.000000  ...

75%                       0.028656  -12413.000000    -289.000000  ...

max                       0.072510   -7489.000000  365243.000000  ...


       FLAG_DOCUMENT_18  FLAG_DOCUMENT_19  FLAG_DOCUMENT_20
FLAG_DOCUMENT_21  \
count     307511.000000     307511.000000     307511.000000
307511.000000
mean           0.008130          0.000595          0.000507
0.000335
std            0.089798          0.024387          0.022518
0.018299
min            0.000000          0.000000          0.000000
0.000000
25%            0.000000          0.000000          0.000000
0.000000
50%            0.000000          0.000000          0.000000
0.000000
75%            0.000000          0.000000          0.000000
0.000000
max            1.000000          1.000000          1.000000
1.000000

       AMT_REQ_CREDIT_BUREAU_HOUR  AMT_REQ_CREDIT_BUREAU_DAY  \
count                265992.000000              265992.000000
mean                      0.006401                   0.007000
std                       0.083984                   0.110718
min                       0.000000                   0.000000
```

```
25%                        0.000000                     0.000000
50%                        0.000000                     0.000000
75%                        0.000000                     0.000000
max                        4.000000                     9.000000

       AMT_REQ_CREDIT_BUREAU_WEEK  AMT_REQ_CREDIT_BUREAU_MON  \
count                 265992.000000                   265992.0
mean                       0.034302                        NaN
std                        0.204712                        0.0
min                        0.000000                        0.0
25%                        0.000000                        0.0
50%                        0.000000                        0.0
75%                        0.000000                        0.0
max                        8.000000                       27.0

       AMT_REQ_CREDIT_BUREAU_QRT  AMT_REQ_CREDIT_BUREAU_YEAR
count                   265992.0                    265992.0
mean                         NaN                         NaN
std                          NaN                         0.0
min                          0.0                         0.0
25%                          0.0                         0.0
50%                          0.0                         1.0
75%                          0.0                         3.0
max                        261.0                        25.0

[8 rows x 106 columns]
```

Grouping featrues by type

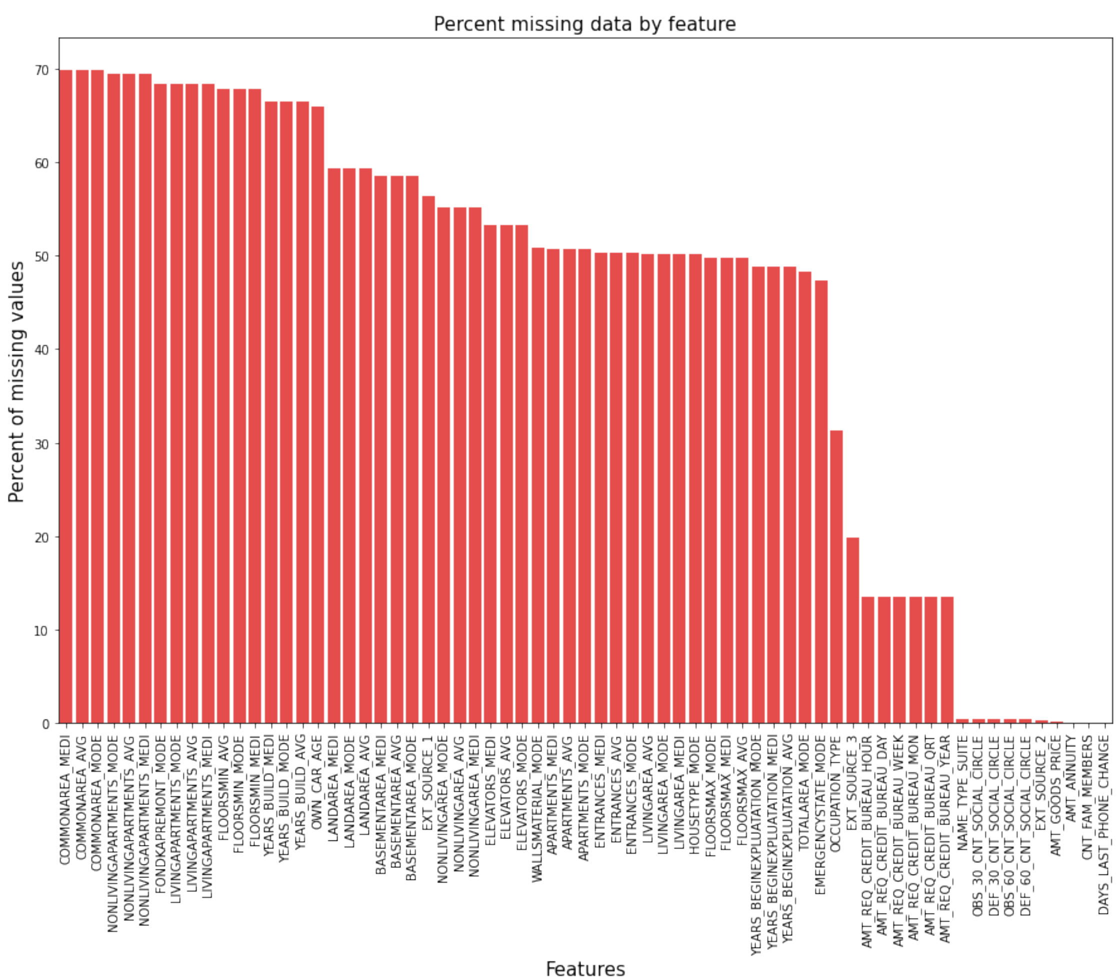
```
feature_type(datasets["application_train"])
```

```
numerical features: Index(['SK_ID_CURR', 'TARGET', 'CNT_CHILDREN',
'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
       'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED',
       ...
       'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20',
       'FLAG_DOCUMENT_21', 'AMT_REQ_CREDIT_BUREAU_HOUR',
       'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_YEAR'],
      dtype='object', length=106)
**********************************************************************
******************************
categorical features : Index(['NAME_CONTRACT_TYPE', 'CODE_GENDER',
'FLAG_OWN_CAR', 'FLAG_OWN_REALTY',
       'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE',
       'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'OCCUPATION_TYPE',
       'WEEKDAY_APPR_PROCESS_START', 'ORGANIZATION_TYPE',
'FONDKAPREMONT_MODE',
```

```
       'HOUSETYPE_MODE', 'WALLSMATERIAL_MODE', 'EMERGENCYSTATE_MODE'],
      dtype='object')
```

```
missingFeatures(datasets["application_train"])
```

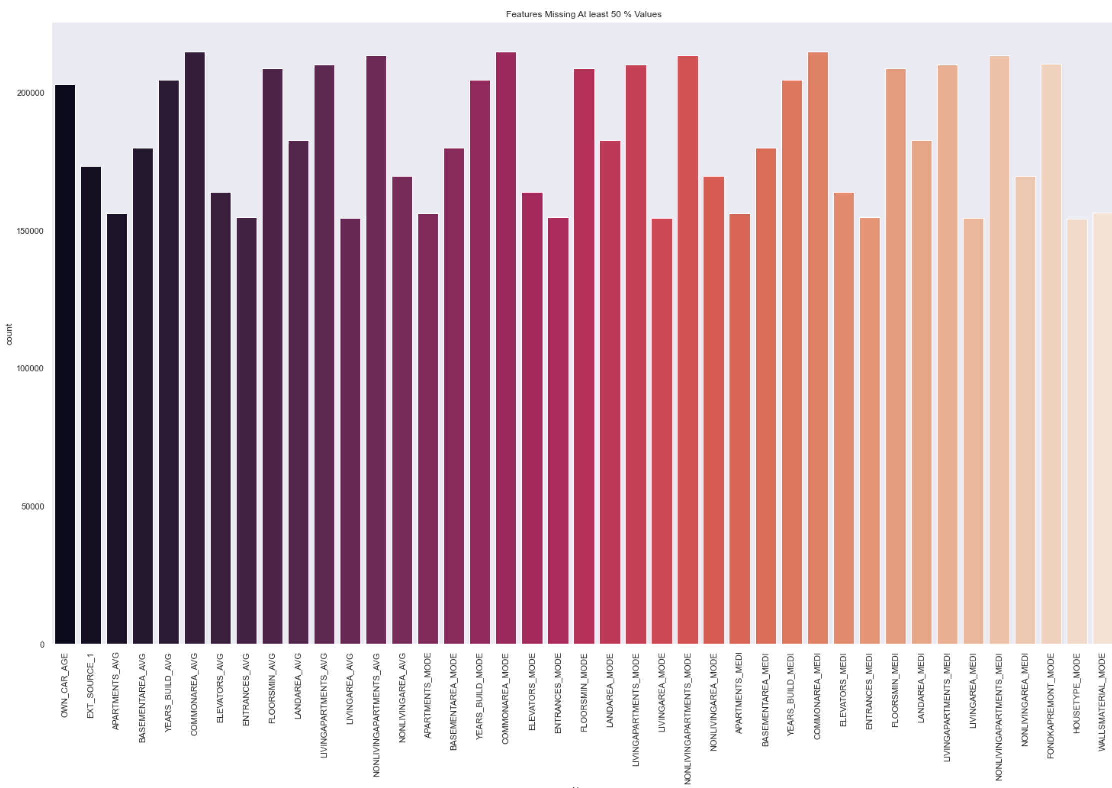
```
                             Total    Percent
COMMONAREA_MEDI             214865  69.872297
COMMONAREA_AVG              214865  69.872297
COMMONAREA_MODE             214865  69.872297
NONLIVINGAPARTMENTS_MODE    213514  69.432963
NONLIVINGAPARTMENTS_AVG     213514  69.432963
...                            ...        ...
EXT_SOURCE_2                   660   0.214626
AMT_GOODS_PRICE                278   0.090403
AMT_ANNUITY                     12   0.003902
CNT_FAM_MEMBERS                  2   0.000650
DAYS_LAST_PHONE_CHANGE           1   0.000325

[67 rows x 2 columns]
```



```python
# Missing value in dataframe
missing_vals = (datasets['application_train'].isna().sum())

print('Missing values in dataframe ',missing_vals[missing_vals > 
0].count())
```

```
Missing values in dataframe  67
```

```python
missing_vals = pd.DataFrame(missing_vals)
missing_vals.columns = ['count']
missing_vals.index.names = ['Name']
missing_vals['Name'] = missing_vals.index

sns.set(style="dark", color_codes=False,rc={'figure.figsize':(25,15)})
sns.barplot(x = 'Name', y = 'count', 
data=missing_vals[missing_vals['count']>len(datasets['application_trai
n'])/2], palette="rocket").set(title='Features Missing At least 50 % 
Values')
plt.xticks(rotation = 90)
plt.show()
```



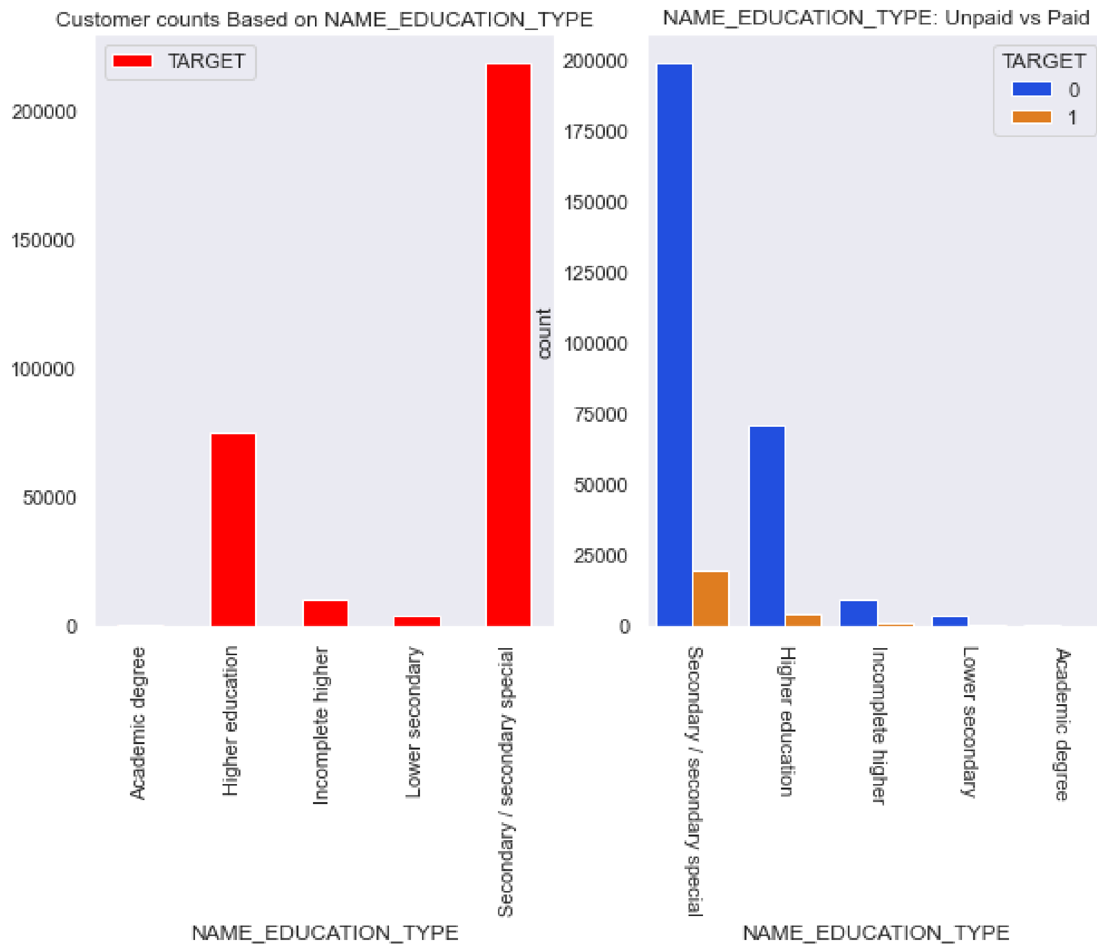
Observation

About 67 features in application train have missing data. Among those features most of them have more than 50% missing data.

Lets have a look at relationship between features and Target features

Relationship between NAME_EDUCATION_TYPE with Target

```
getRelationship(datasets['application_train'],'NAME_EDUCATION_TYPE','TARGET')
```

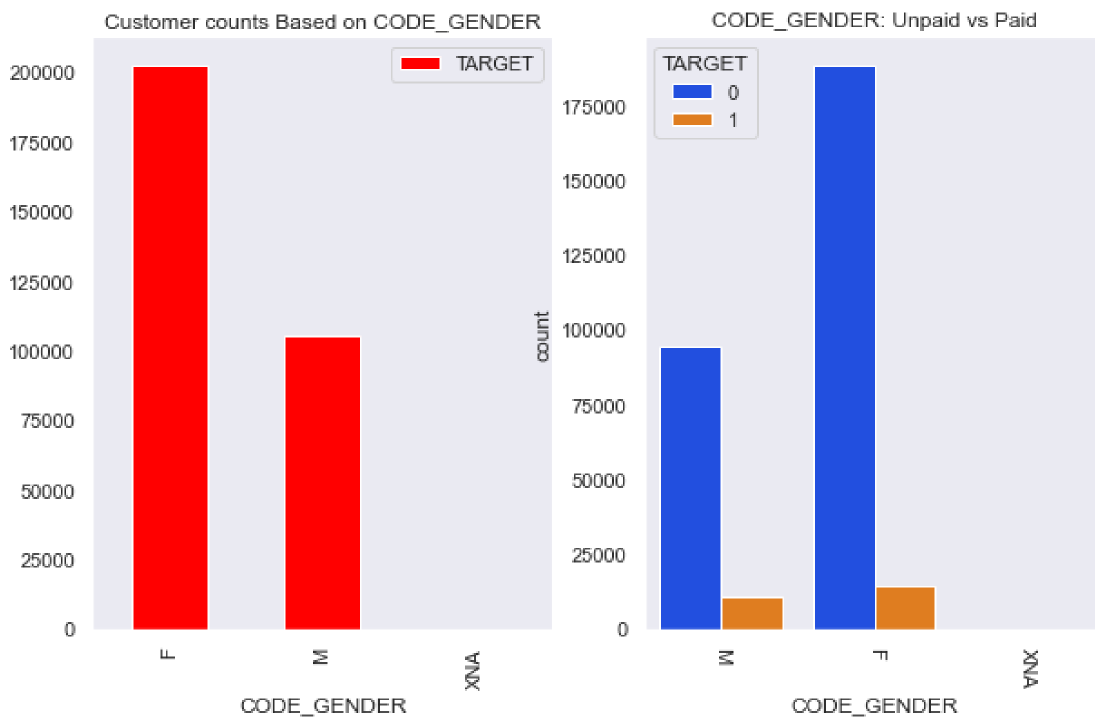


Observation

This is evidant from the plot that a customers with Secondary/Secondary special had a high rate of not paying back as compared with customers with other education types

Relationship between Gender with Target

```
getRelationship(datasets['application_train'],'CODE_GENDER','TARGET')
```

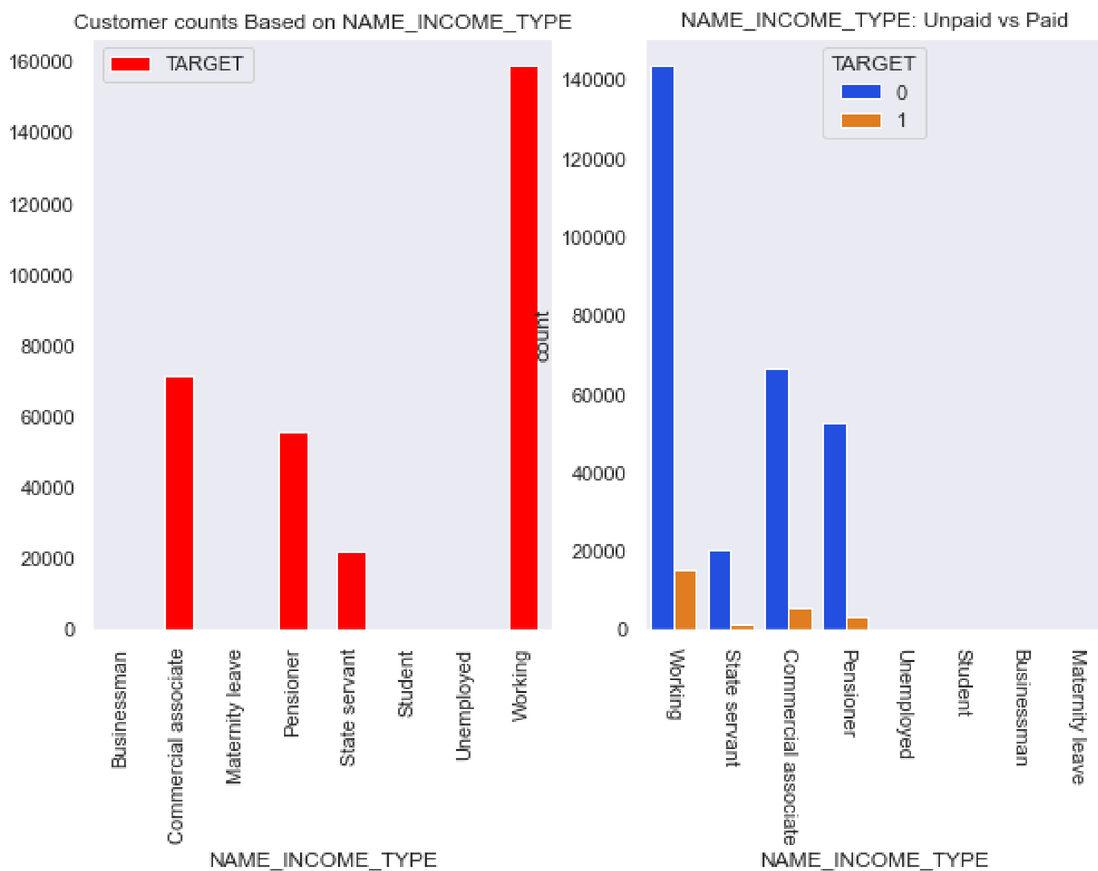


Observation

From the plot, we can see that most of the customers are Females and that is the reason they are defaulting more on paying back than males.

Relationship between Customer income with Target

```
getRelationship(datasets['application_train'],'NAME_INCOME_TYPE','TARG
ET')
```

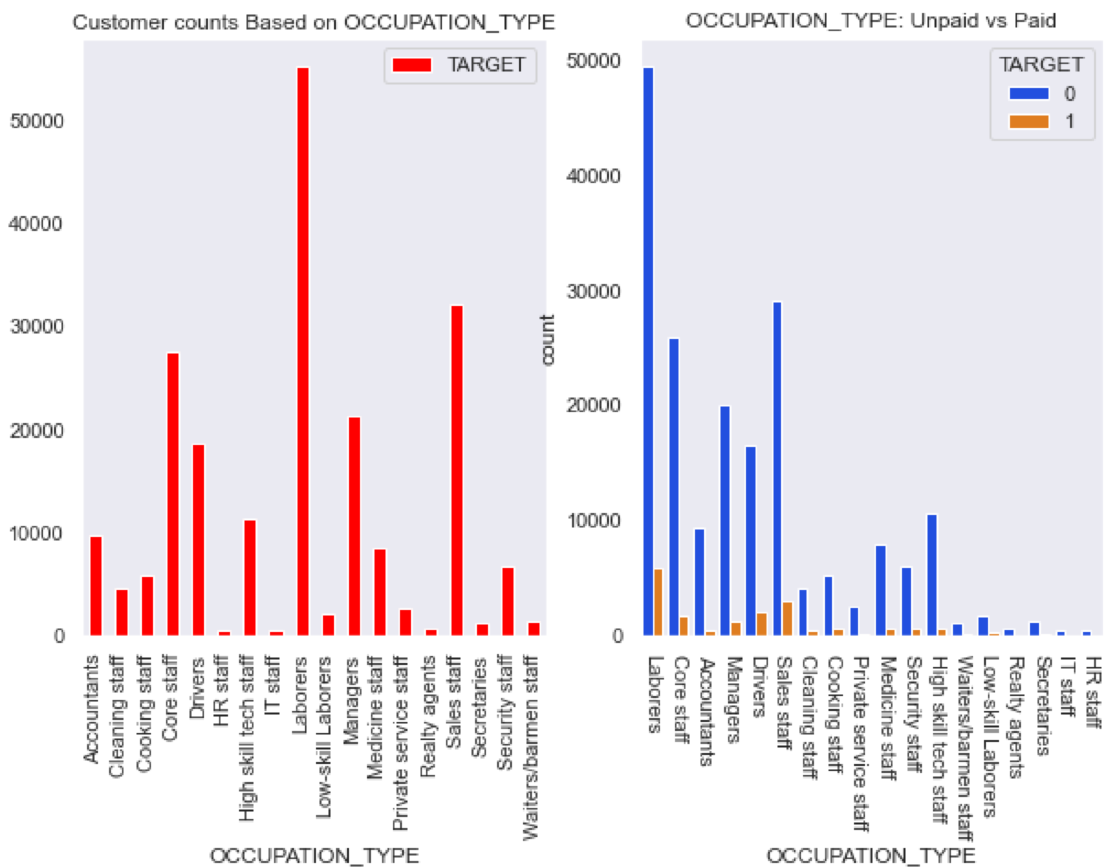


Observation

From the plot, we can see that most of the customers are working and still they are defaulting on paying back. Let's drill more on this and check why is that.

Relationship between Customer Occupation with Target

```
getRelationship(datasets['application_train'],'OCCUPATION_TYPE','TARGE
T')
```

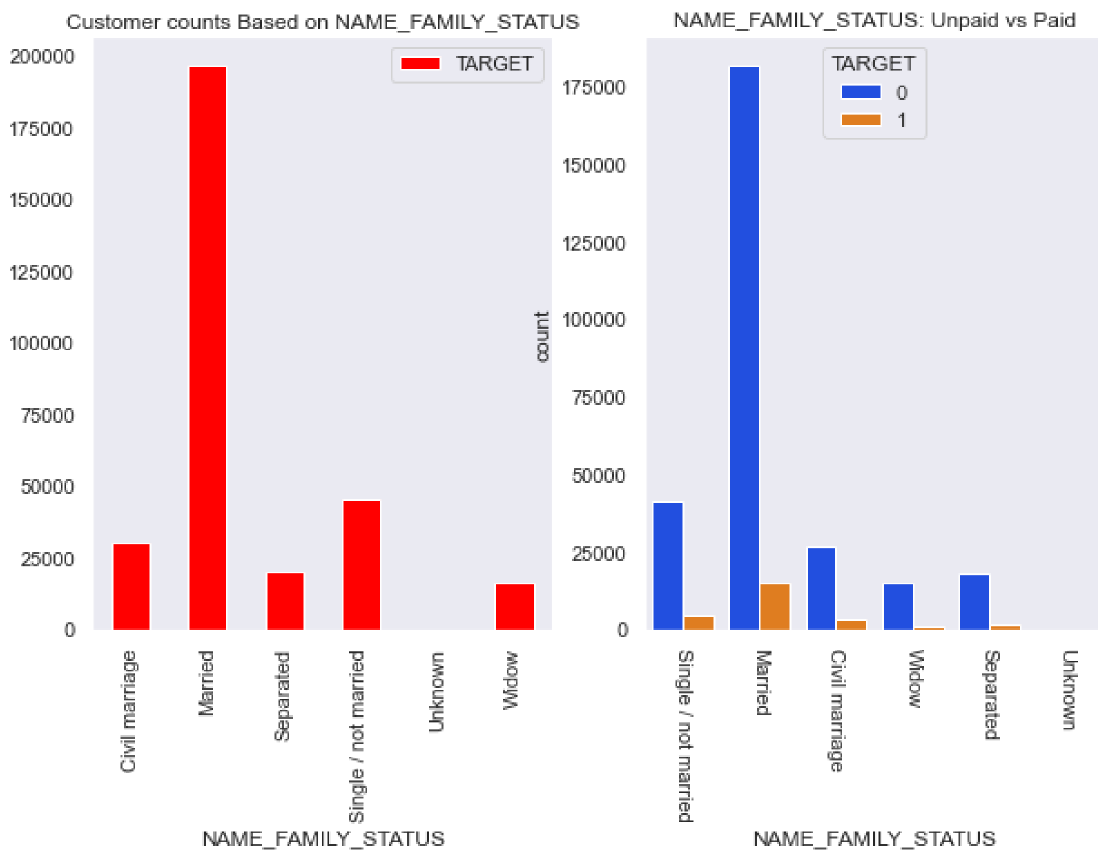


Observation

From the plot, we can see that most of the customers are laborers and they are the customers who are defaulting the most as compared to customers with other occupations. This makes sense because laborers don't make that much money and may be thats the reason thety are defaulting more.

Relationship between Customer Family Status with Target

```
getRelationship(datasets['application_train'],'NAME_FAMILY_STATUS','TA
RGET')
```

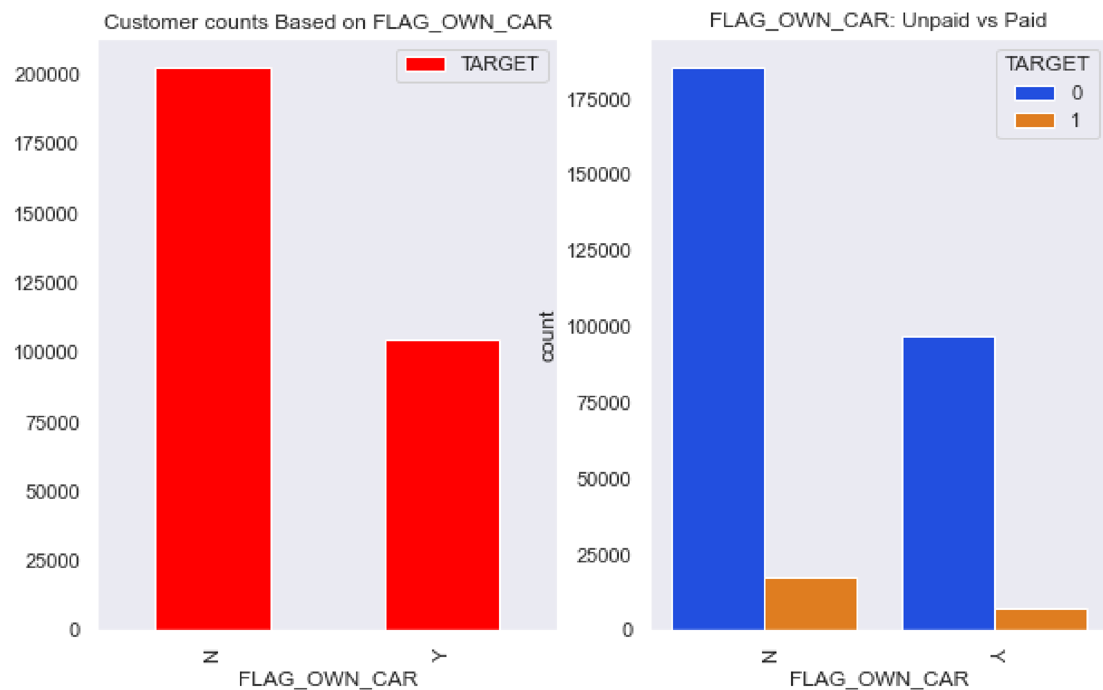


Observation

From the plot, we can see that most of the customers are married. Customers who are marrid have the highest rate of defaulting as compared to other customers with differnt family status.

Relationship between Customer Owning a car with Target

```
getRelationship(datasets['application_train'],'FLAG_OWN_CAR','TARGET')
```

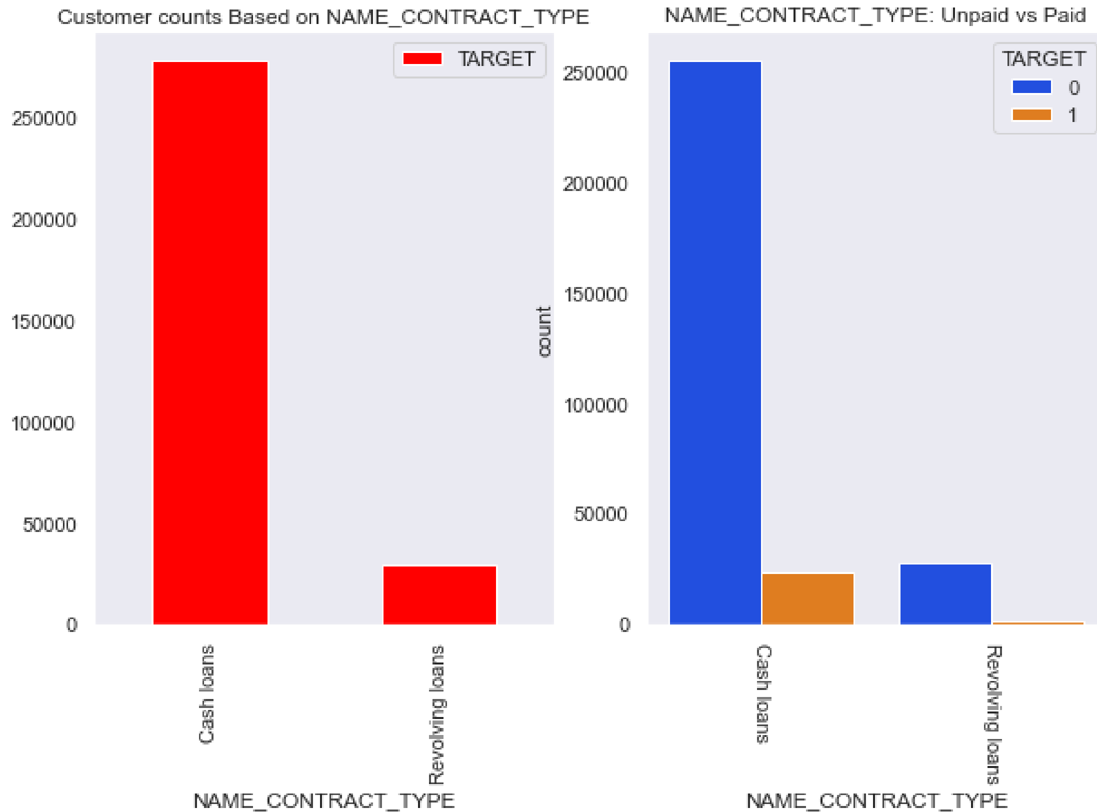


Observation

Most of the vcustomers don't own a car and that is also the chunk of customers which is not paying back.

Relationship between Customer contract type with Target

```
getRelationship(datasets['application_train'],'NAME_CONTRACT_TYPE','TA
RGET')
```

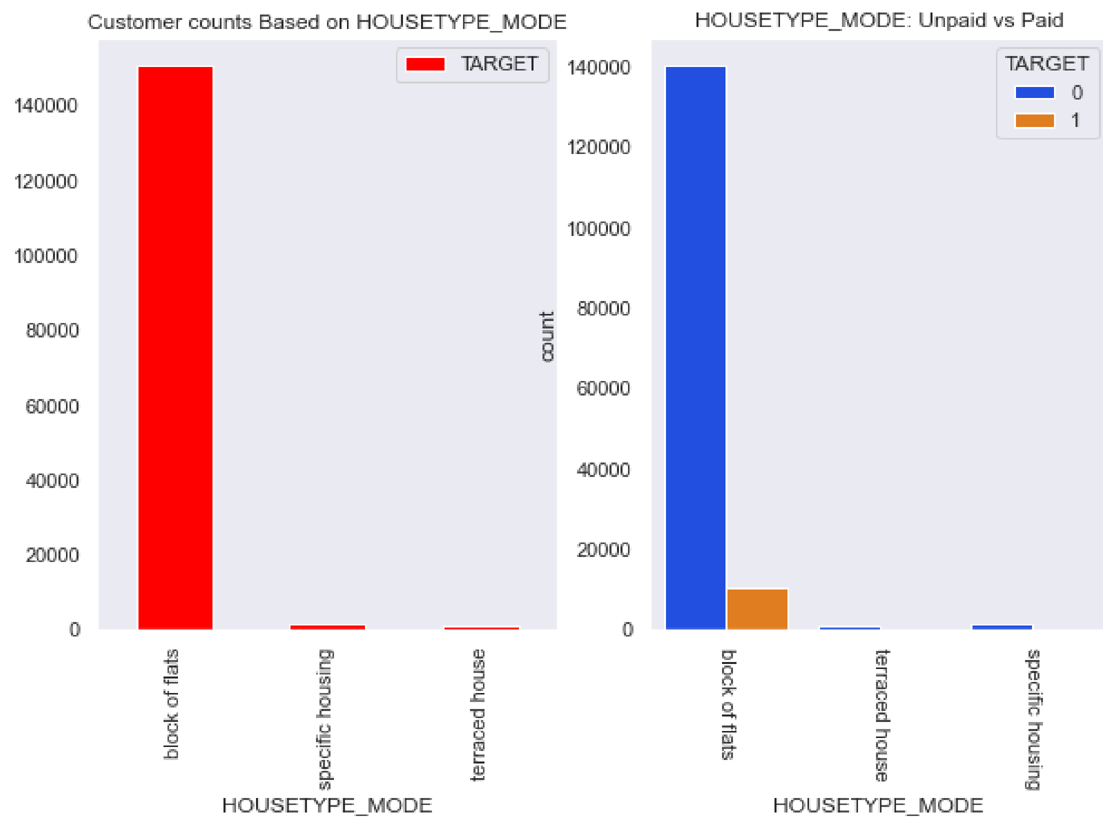


Observation

Most of the customers took cash loans. Customers with cash loans defaulted more on the loan as compared to customers with recieving loan contract ype.

Relationship between Customer House type with Target

```
getRelationship(datasets['application_train'],'HOUSETYPE_MODE','TARGET
')
```

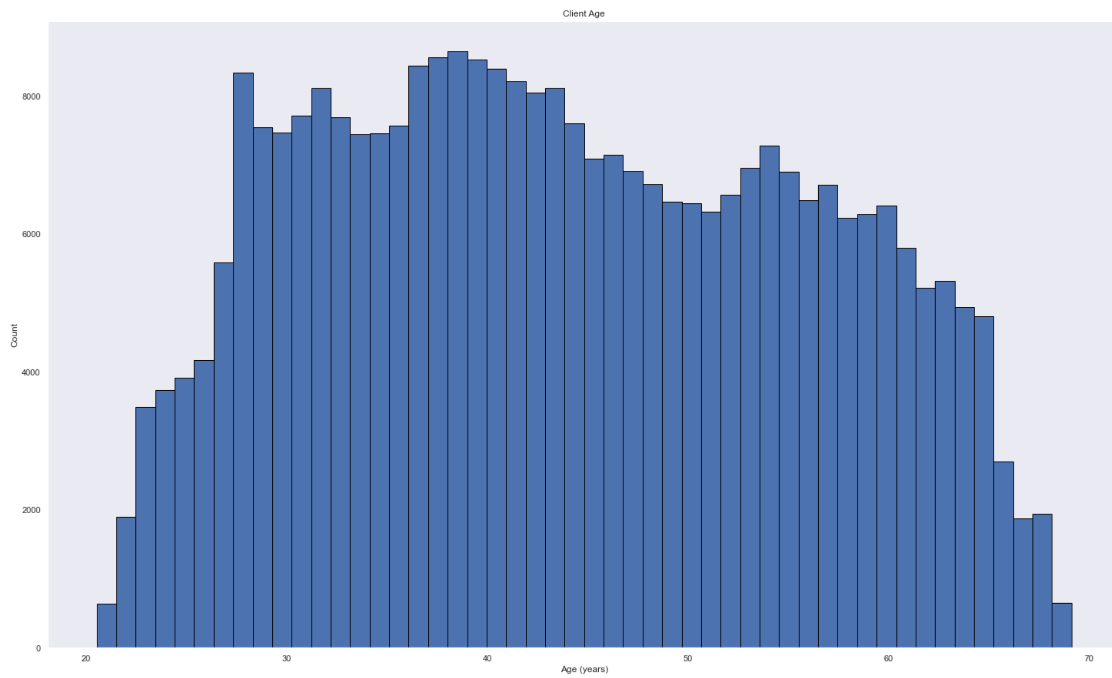


Observation

From the plot it is clear that the data is highly skewed towards customers with block of flats.

Client Age Distribution

```
plt.hist(datasets["application_train"]['DAYS_BIRTH'] /-365, edgecolor
= 'k', bins = 50)
plt.title('Client Age'); plt.xlabel('Age (years)');
plt.ylabel('Count');
plt.show()
```



Observation

Most of the customers are between the ages 30-60

Bureau EDA

```
datasets['bureau'].describe()

         SK_ID_CURR  SK_ID_BUREAU   DAYS_CREDIT  CREDIT_DAY_OVERDUE  \
count  1.716428e+06  1.716428e+06  1.716428e+06        1.716428e+06
mean   2.782149e+05  5.924434e+06 -1.142108e+03        8.181666e-01
std    1.029386e+05  5.322657e+05  7.951649e+02        3.654443e+01
min    1.000010e+05  5.000000e+06 -2.922000e+03        0.000000e+00
25%    1.888668e+05  5.463954e+06 -1.666000e+03        0.000000e+00
50%    2.780550e+05  5.926304e+06 -9.870000e+02        0.000000e+00
75%    3.674260e+05  6.385681e+06 -4.740000e+02        0.000000e+00
max    4.562550e+05  6.843457e+06  0.000000e+00        2.792000e+03

       DAYS_CREDIT_ENDDATE  DAYS_ENDDATE_FACT  AMT_CREDIT_MAX_OVERDUE
\
count            1610875.0          1082775.0            5.919400e+05

mean                   NaN                NaN            3.825358e+03

std                    NaN                NaN            2.059873e+05

min               -42048.0           -42016.0            0.000000e+00
```

```
25%                  -1138.0            -1489.0            0.000000e+00

50%                   -330.0             -897.0            0.000000e+00

75%                    474.0             -425.0            0.000000e+00

max                  31200.0                0.0            1.159872e+08
```

```
       CNT_CREDIT_PROLONG  AMT_CREDIT_SUM  AMT_CREDIT_SUM_DEBT  \
count        1.716428e+06    1.716415e+06         1.458759e+06
mean         6.410406e-03    3.545773e+05         1.370818e+05
std          9.622391e-02    1.150277e+06         6.790749e+05
min          0.000000e+00    0.000000e+00        -4.705600e+06
25%          0.000000e+00    5.130000e+04         0.000000e+00
50%          0.000000e+00    1.255185e+05         0.000000e+00
75%          0.000000e+00    3.150000e+05         4.015350e+04
max          9.000000e+00    5.850000e+08         1.701000e+08

       AMT_CREDIT_SUM_LIMIT  AMT_CREDIT_SUM_OVERDUE
DAYS_CREDIT_UPDATE  \
count          1.124648e+06            1.716428e+06
1.716428e+06
mean           6.229781e+03            3.791263e+01       -
5.937483e+02
std            4.489666e+04            5.937519e+03
7.207473e+02
min           -5.864061e+05            0.000000e+00       -
4.194700e+04
25%            0.000000e+00            0.000000e+00       -
9.080000e+02
50%            0.000000e+00            0.000000e+00       -
3.950000e+02
75%            0.000000e+00            0.000000e+00       -
3.300000e+01
max            4.705600e+06            3.756681e+06
3.720000e+02

        AMT_ANNUITY
count  4.896370e+05
mean   1.571327e+04
std    3.256556e+05
min    0.000000e+00
25%    0.000000e+00
50%    0.000000e+00
75%    1.350000e+04
max    1.184534e+08
```

Grouping featrues by type

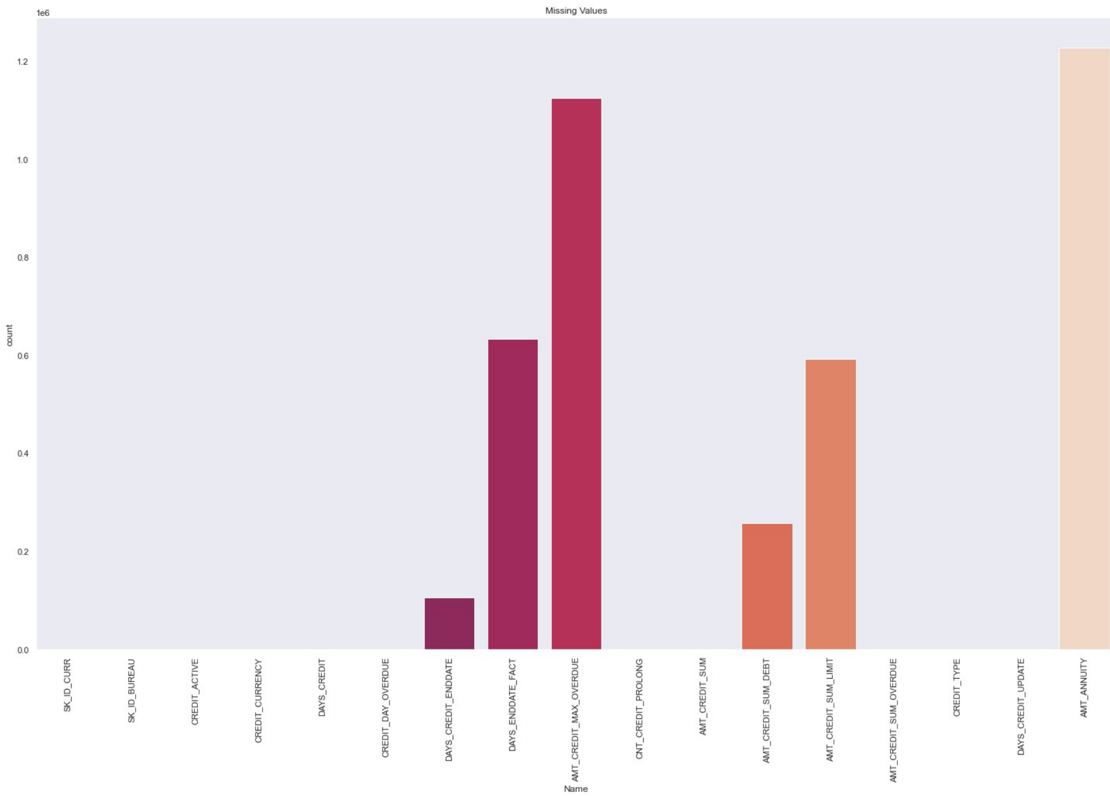
```
feature_type(datasets["bureau"])
```

```
numerical features: Index(['SK_ID_CURR', 'SK_ID_BUREAU',
'DAYS_CREDIT', 'CREDIT_DAY_OVERDUE',
       'DAYS_CREDIT_ENDDATE', 'DAYS_ENDDATE_FACT',
'AMT_CREDIT_MAX_OVERDUE',
       'CNT_CREDIT_PROLONG', 'AMT_CREDIT_SUM', 'AMT_CREDIT_SUM_DEBT',
       'AMT_CREDIT_SUM_LIMIT', 'AMT_CREDIT_SUM_OVERDUE',
'DAYS_CREDIT_UPDATE',
       'AMT_ANNUITY'],
      dtype='object')
**********************************************************************
******************************
categorical features : Index(['CREDIT_ACTIVE', 'CREDIT_CURRENCY',
'CREDIT_TYPE'], dtype='object')
```

```
# Missing value in dataframe
missing_vals = (datasets['bureau'].isna().sum())

print('Missing values in dataframe ',missing_vals[missing_vals >
0].count())
```

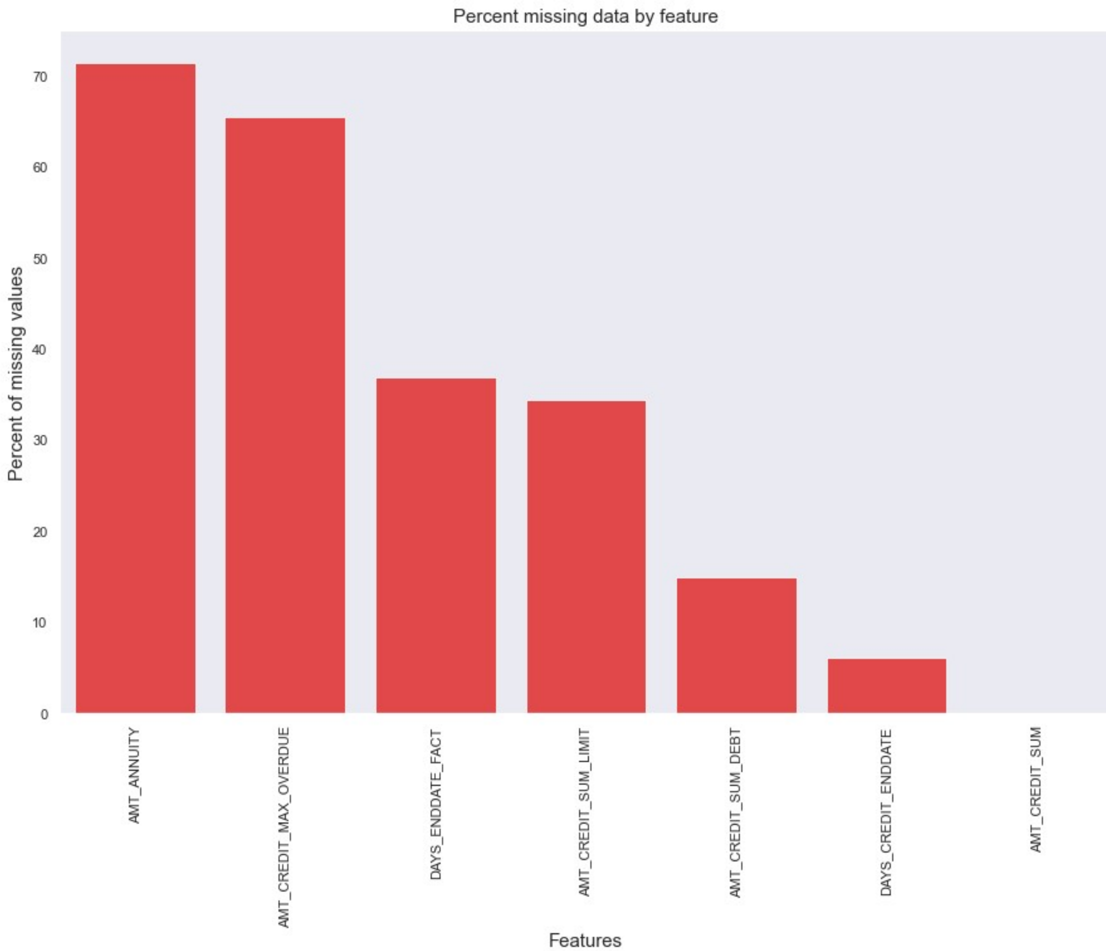
```
Missing values in dataframe  7
```

```
missing_vals = pd.DataFrame(missing_vals)
missing_vals.columns = ['count']
missing_vals.index.names = ['Name']
missing_vals['Name'] = missing_vals.index

sns.set(style="dark", color_codes=True,rc={'figure.figsize':(25,15)})
sns.barplot(x = 'Name', y = 'count', data=missing_vals,
palette="rocket").set(title='Missing Values')
plt.xticks(rotation = 90)
plt.show()
```




```
missingFeatures(datasets["bureau"])
```

	Total	Percent
AMT_ANNUITY	1226791	71.473490
AMT_CREDIT_MAX_OVERDUE	1124488	65.513264
DAYS_ENDDATE_FACT	633653	36.916958
AMT_CREDIT_SUM_LIMIT	591780	34.477415
AMT_CREDIT_SUM_DEBT	257669	15.011932
DAYS_CREDIT_ENDDATE	105553	6.149573
AMT_CREDIT_SUM	13	0.000757

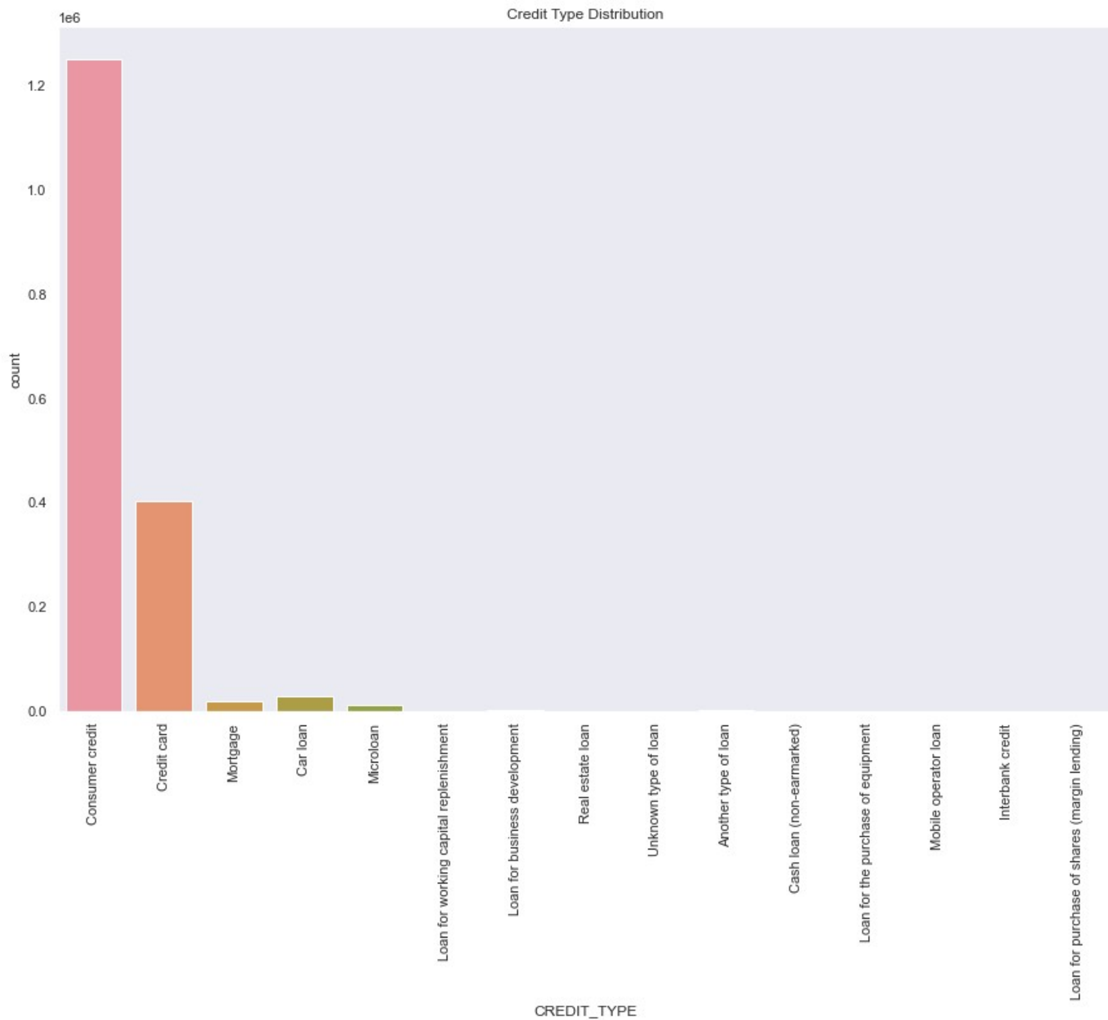


Observations

7 features in Bureau are missing values 4 of them have more than 30% missing data.

CREDIT TYPE Analysis

```
plt.figure(figsize=(15,10))
sns.countplot(x='CREDIT_TYPE', data=datasets["bureau"]);
plt.title('Credit Type Distribution');
plt.xticks(rotation=90);
plt.show()
```

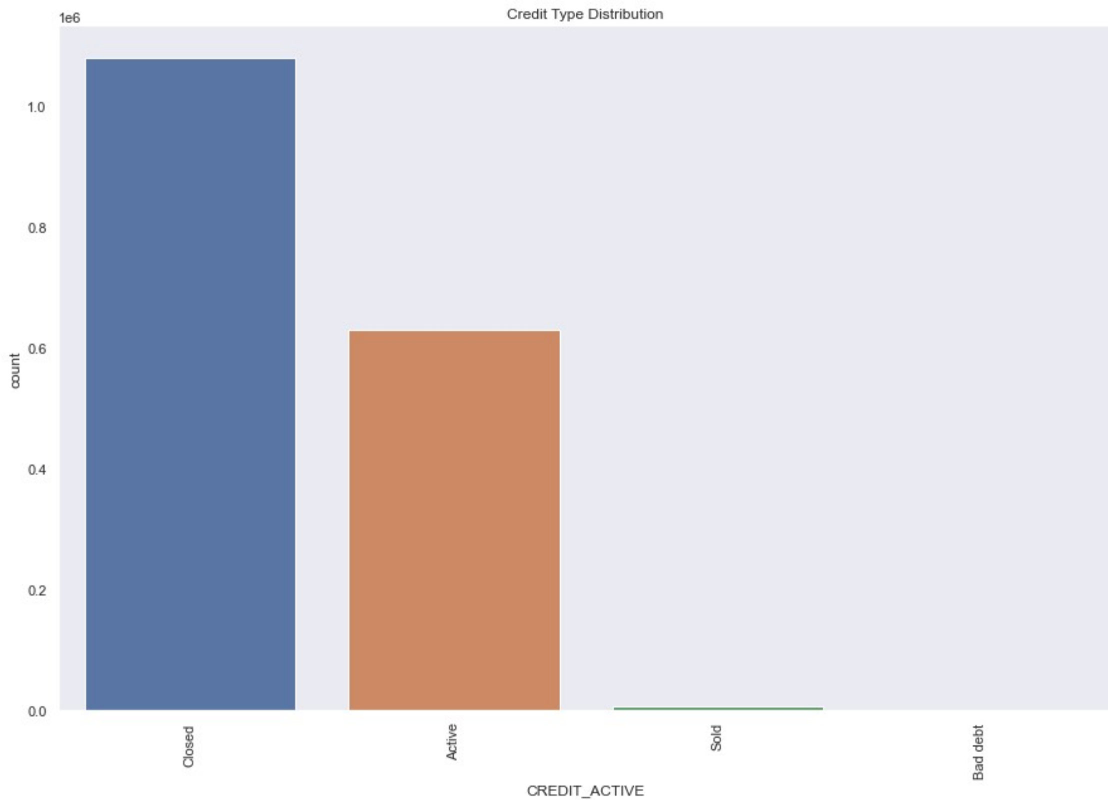


Observation

Majority of the customers have Consumer Credits

Credit Active analysis

```
plt.figure(figsize=(15,10))
sns.countplot(x='CREDIT_ACTIVE', data=datasets["bureau"]);
plt.title('Credit Type Distribution');
plt.xticks(rotation=90);
plt.show()
```



Observation

Majority of the custormers have Closed credit. There is no customer with bad debt

Bureau Balance EDA

```
datasets['bureau_balance'].describe()
```

```
       SK_ID_BUREAU  MONTHS_BALANCE
count  2.729992e+07    2.729992e+07
mean   6.036297e+06   -3.074169e+01
std    4.923489e+05    2.386451e+01
min    5.001709e+06   -9.600000e+01
25%    5.730933e+06   -4.600000e+01
50%    6.070821e+06   -2.500000e+01
75%    6.431951e+06   -1.100000e+01
max    6.842888e+06    0.000000e+00
```

Grouping featrues by type

```
feature_type(datasets["bureau_balance"])
```

```
numerical features: Index(['SK_ID_BUREAU', 'MONTHS_BALANCE'],
dtype='object')
**********************************************************************
******************************
categorical features : Index(['STATUS'], dtype='object')
```

```python
# Missing value in dataframe
missing_vals = (datasets['bureau_balance'].isna().sum())

print('Missing values in dataframe ',missing_vals[missing_vals >
0].count())
```

```
Missing values in dataframe  0
```

Previous Application EDA

```python
datasets['previous_application'].describe()
```

```
         SK_ID_PREV    SK_ID_CURR   AMT_ANNUITY  AMT_APPLICATION  \
count  1.670214e+06  1.670214e+06  1.297979e+06     1.670214e+06
mean   1.923089e+06  2.783572e+05  1.594889e+04     1.749806e+05
std    5.325980e+05  1.028148e+05  1.477695e+04     2.933005e+05
min    1.000001e+06  1.000010e+05  0.000000e+00     0.000000e+00
25%    1.461857e+06  1.893290e+05  6.321780e+03     1.872000e+04
50%    1.923110e+06  2.787145e+05  1.125000e+04     7.104600e+04
75%    2.384280e+06  3.675140e+05  2.065842e+04     1.803600e+05
max    2.845382e+06  4.562550e+05  4.180582e+05     6.905160e+06

         AMT_CREDIT  AMT_DOWN_PAYMENT  AMT_GOODS_PRICE  \
count  1.670213e+06      7.743700e+05     1.284699e+06
mean   1.960131e+05      6.699080e+03     2.275182e+05
std    3.177837e+05      2.090572e+04     3.154605e+05
min    0.000000e+00     -9.000000e-01     0.000000e+00
25%    2.416050e+04      0.000000e+00     5.084100e+04
50%    8.054100e+04      1.638000e+03     1.123200e+05
75%    2.164185e+05      7.740000e+03     2.340000e+05
max    6.905160e+06      3.060045e+06     6.905160e+06

       HOUR_APPR_PROCESS_START  NFLAG_LAST_APPL_IN_DAY
RATE_DOWN_PAYMENT  \
count             1.670214e+06            1.670214e+06
774370.000000
mean              1.248418e+01            9.964675e-01
0.079651
std               3.334028e+00            5.932963e-02
0.107788
min               0.000000e+00            0.000000e+00          -
0.000015
25%               1.000000e+01            1.000000e+00
0.000000
50%               1.200000e+01            1.000000e+00
0.051605
75%               1.500000e+01            1.000000e+00
0.108887
max               2.300000e+01            1.000000e+00
1.000000
```

```
       ...  RATE_INTEREST_PRIVILEGED  DAYS_DECISION  SELLERPLACE_AREA  \
count  ...               5951.000000   1.670214e+06      1.670214e+06

mean   ...                  0.774902  -8.806797e+02      3.139511e+02

std    ...                  0.100952   7.790997e+02      7.127443e+03

min    ...                  0.373047  -2.922000e+03     -1.000000e+00

25%    ...                  0.715820  -1.300000e+03     -1.000000e+00

50%    ...                  0.834961  -5.810000e+02      3.000000e+00

75%    ...                  0.852539  -2.800000e+02      8.200000e+01

max    ...                  1.000000  -1.000000e+00      4.000000e+06


       CNT_PAYMENT  DAYS_FIRST_DRAWING  DAYS_FIRST_DUE  \
count    1297984.0       997149.000000   997149.000000
mean           NaN       340114.343750    13838.132812
std            0.0        88611.609375    72421.296875
min            0.0        -2922.000000    -2892.000000
25%            6.0       365243.000000    -1628.000000
50%           12.0       365243.000000     -831.000000
75%           24.0       365243.000000     -411.000000
max           84.0       365243.000000   365243.000000

       DAYS_LAST_DUE_1ST_VERSION  DAYS_LAST_DUE  DAYS_TERMINATION  \
count              997149.000000  997149.000000      997149.00000
mean                33764.871094   76829.148438       82314.84375
std                106544.812500  150155.109375      152926.93750
min                 -2801.000000   -2889.000000       -2874.00000
25%                 -1242.000000   -1314.000000       -1270.00000
50%                  -361.000000    -537.000000        -499.00000
75%                   129.000000     -74.000000         -44.00000
max                365243.000000  365243.000000      365243.00000

       NFLAG_INSURED_ON_APPROVAL
count                   997149.0
mean                         NaN
std                          0.0
min                          0.0
25%                          0.0
50%                          0.0
75%                          1.0
max                          1.0
```

```
[8 rows x 21 columns]
```

Grouping featrues by type

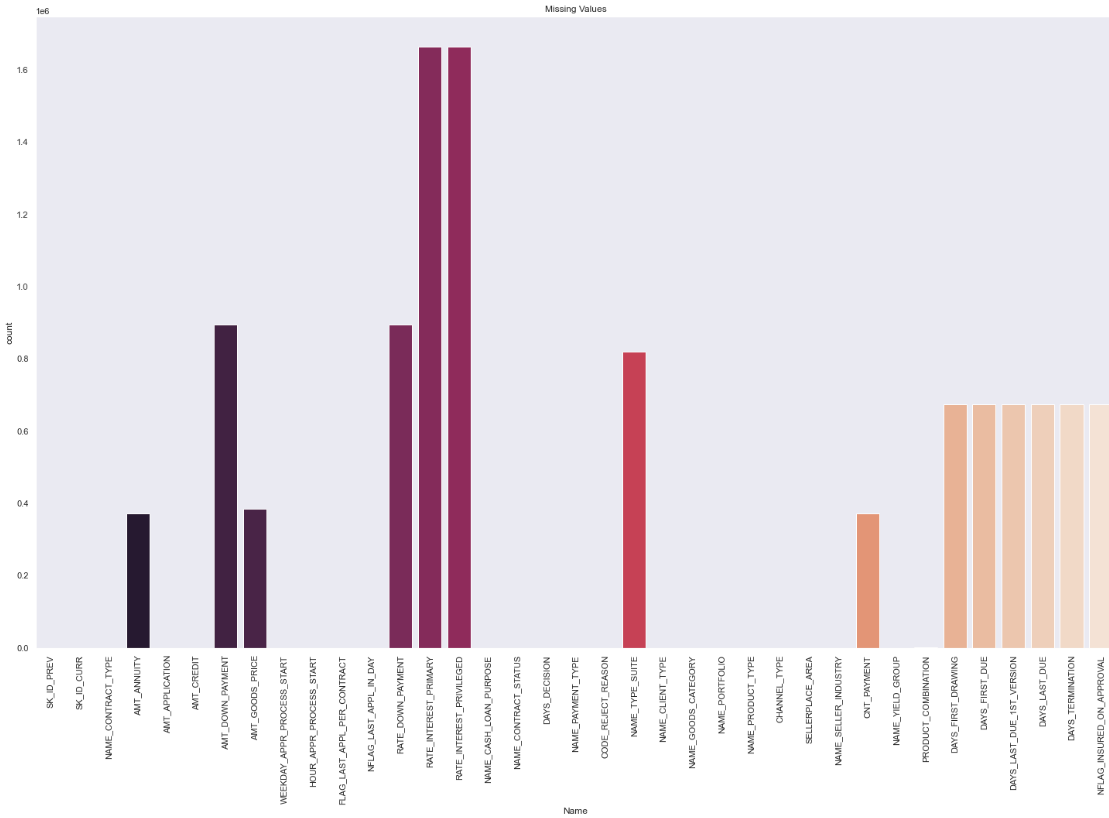
```
feature_type(datasets["previous_application"])
```

```
numerical features: Index(['SK_ID_PREV', 'SK_ID_CURR', 'AMT_ANNUITY',
'AMT_APPLICATION',
       'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AMT_GOODS_PRICE',
       'HOUR_APPR_PROCESS_START', 'NFLAG_LAST_APPL_IN_DAY',
       'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY',
       'RATE_INTEREST_PRIVILEGED', 'DAYS_DECISION',
'SELLERPLACE_AREA',
       'CNT_PAYMENT', 'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE',
       'DAYS_LAST_DUE_1ST_VERSION', 'DAYS_LAST_DUE',
'DAYS_TERMINATION',
       'NFLAG_INSURED_ON_APPROVAL'],
      dtype='object')
**********************************************************************
******************************
categorical features : Index(['NAME_CONTRACT_TYPE',
'WEEKDAY_APPR_PROCESS_START',
       'FLAG_LAST_APPL_PER_CONTRACT', 'NAME_CASH_LOAN_PURPOSE',
       'NAME_CONTRACT_STATUS', 'NAME_PAYMENT_TYPE',
'CODE_REJECT_REASON',
       'NAME_TYPE_SUITE', 'NAME_CLIENT_TYPE', 'NAME_GOODS_CATEGORY',
       'NAME_PORTFOLIO', 'NAME_PRODUCT_TYPE', 'CHANNEL_TYPE',
       'NAME_SELLER_INDUSTRY', 'NAME_YIELD_GROUP',
'PRODUCT_COMBINATION'],
      dtype='object')
```

```
# Missing value in dataframe
missing_vals = (datasets['previous_application'].isna().sum())
print('Missing values in dataframe ',missing_vals[missing_vals >
0].count())
```

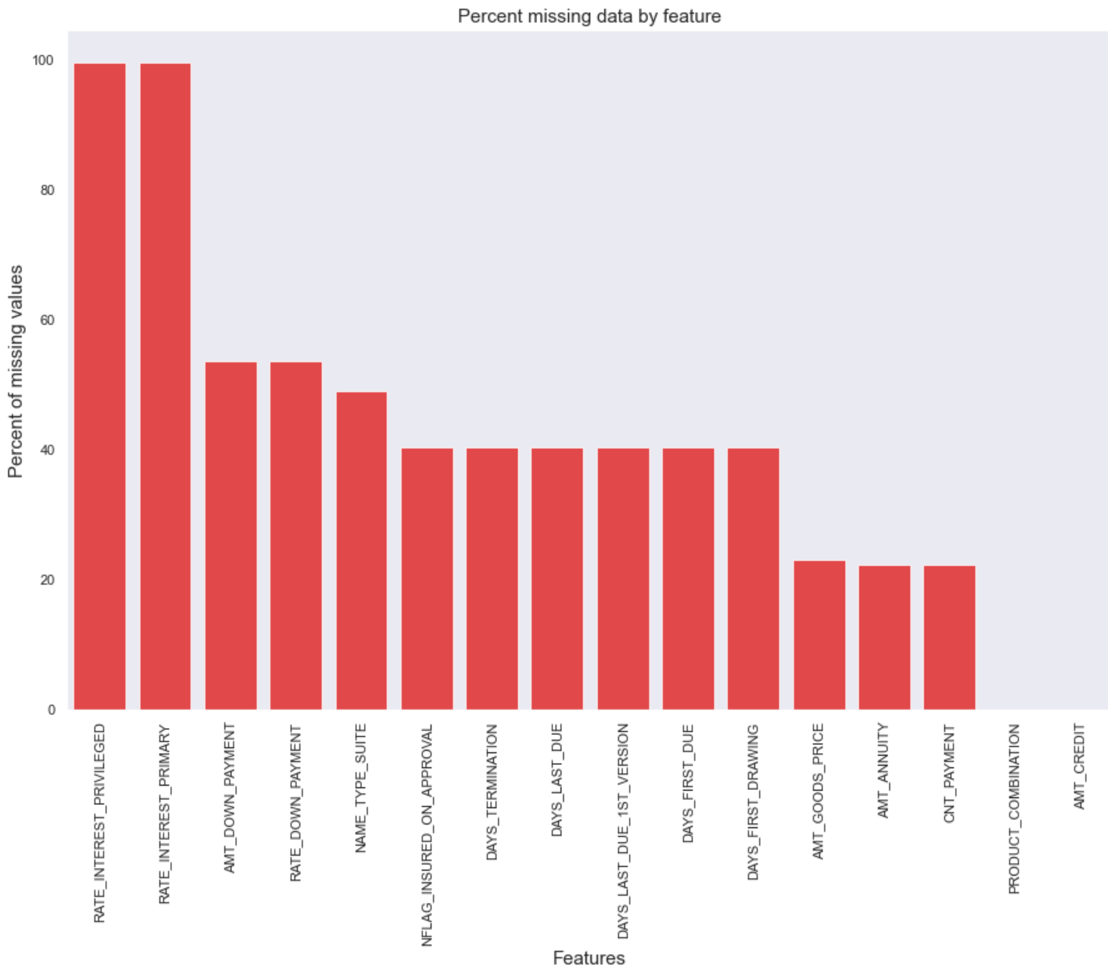
```
Missing values in dataframe  16
```

```
missing_vals = pd.DataFrame(missing_vals)
missing_vals.columns = ['count']
missing_vals.index.names = ['Name']
missing_vals['Name'] = missing_vals.index

sns.set(style="dark", color_codes=True,rc={'figure.figsize':(25,15)})
sns.barplot(x = 'Name', y = 'count', data=missing_vals,
palette="rocket").set(title='Missing Values')
plt.xticks(rotation = 90)
plt.show()
```




```
missingFeatures(datasets["previous_application"])
```

	Total	Percent
RATE_INTEREST_PRIVILEGED	1664263	99.643698
RATE_INTEREST_PRIMARY	1664263	99.643698
AMT_DOWN_PAYMENT	895844	53.636480
RATE_DOWN_PAYMENT	895844	53.636480
NAME_TYPE_SUITE	820405	49.119754
NFLAG_INSURED_ON_APPROVAL	673065	40.298129
DAYS_TERMINATION	673065	40.298129
DAYS_LAST_DUE	673065	40.298129
DAYS_LAST_DUE_1ST_VERSION	673065	40.298129
DAYS_FIRST_DUE	673065	40.298129
DAYS_FIRST_DRAWING	673065	40.298129
AMT_GOODS_PRICE	385515	23.081773
AMT_ANNUITY	372235	22.286665
CNT_PAYMENT	372230	22.286366
PRODUCT_COMBINATION	346	0.020716
AMT_CREDIT	1	0.000060

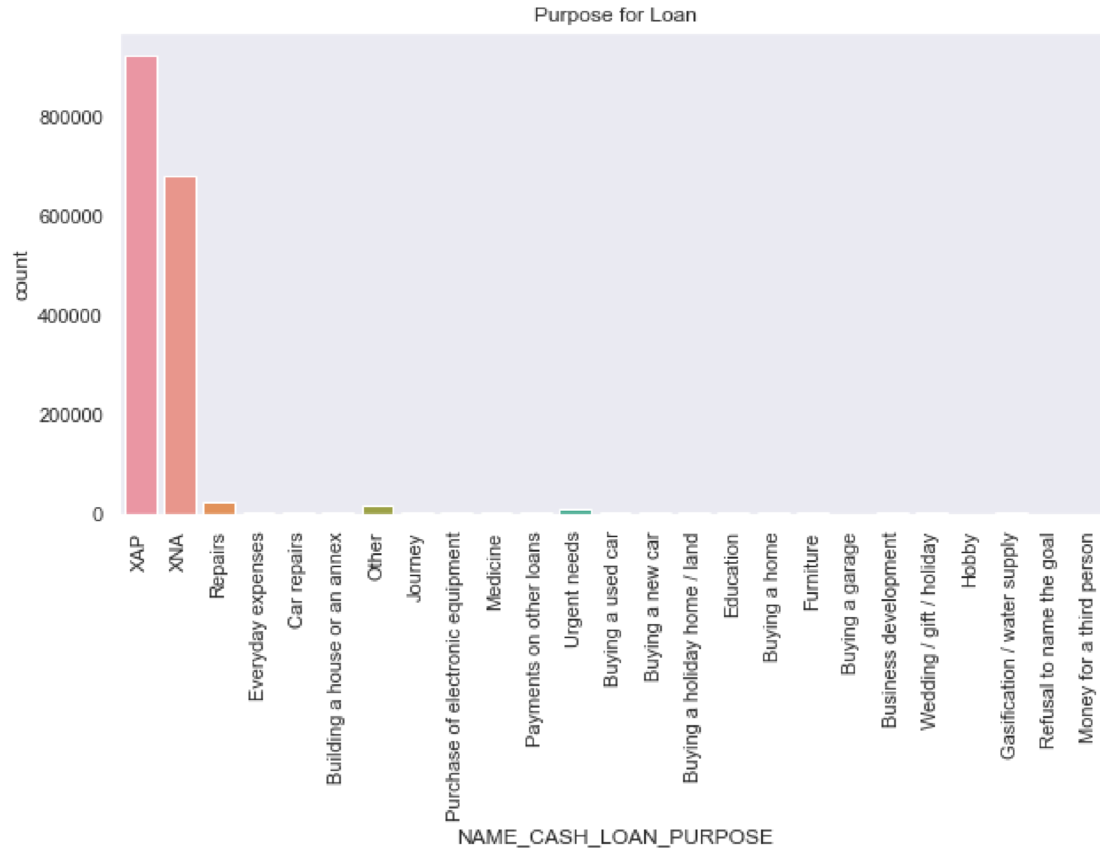


Observation

16 features have missing values. RATE_INTEREST_PRIVILEGED and RATE_INTEREST_PRIMARY have 99% missing data. These features are almost of no use to us. Other than these 2 features, 9 features have more than 40% missing data.

previous_application Analysis

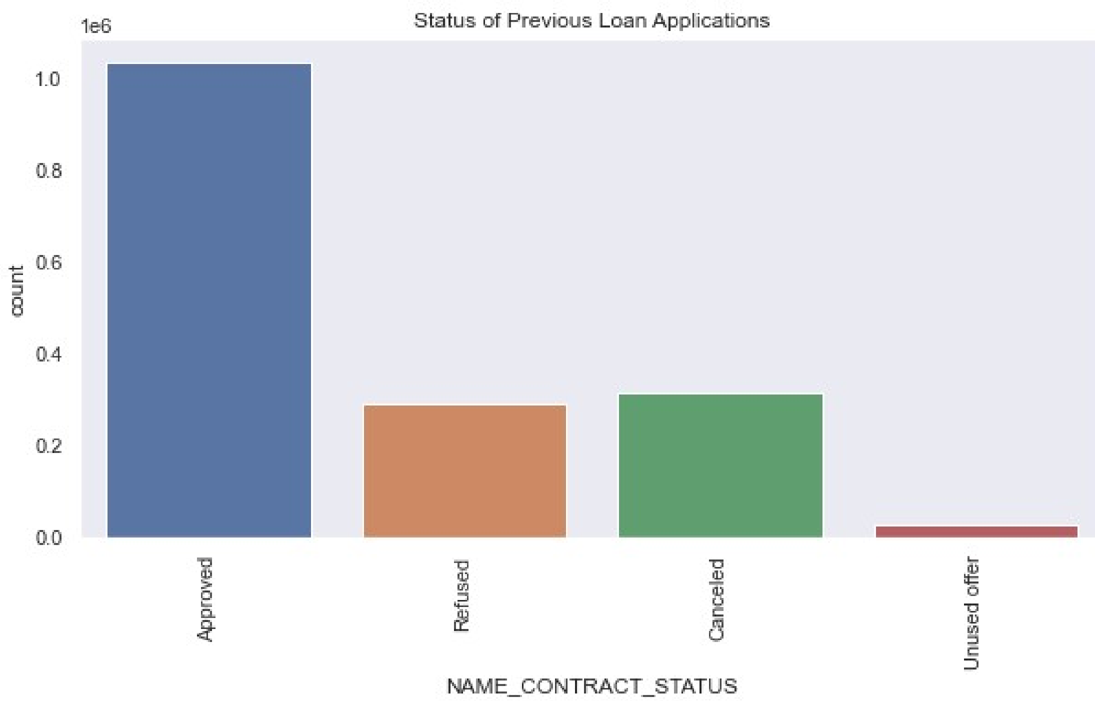
```
plt.figure(figsize=(10,5))
sns.countplot(x='NAME_CASH_LOAN_PURPOSE',
data=datasets["previous_application"]);
plt.title('Purpose for Loan');
plt.xticks(rotation=90);
plt.show()
```



Observation

Almost all the customers took loans for 2 use cases (XAP & XNA)

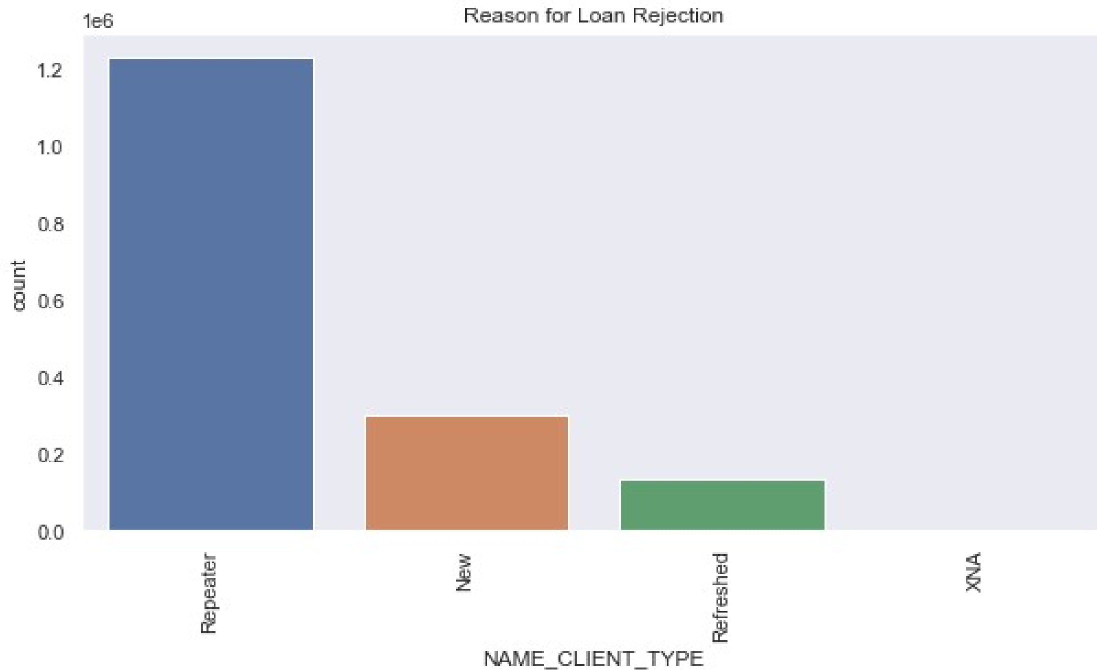
```
plt.figure(figsize=(10,5))
sns.countplot(x='NAME_CONTRACT_STATUS',
data=datasets["previous_application"]);
plt.title('Status of Previous Loan Applications');
plt.xticks(rotation=90);
plt.show()
```



Observation

Most of the customer's previous applications were approved.

```
plt.figure(figsize=(10,5))
sns.countplot(x='NAME_CLIENT_TYPE',
data=datasets["previous_application"]);
plt.title('Reason for Loan Rejection');
plt.xticks(rotation=90);
plt.show()
```



Observation

Customers who were repeaters, got their application rejected the most.

CREDIT CARD BALANCE EDA

```
datasets['credit_card_balance'].describe()
```

```
         SK_ID_PREV    SK_ID_CURR  MONTHS_BALANCE   AMT_BALANCE  \
count  3.840312e+06  3.840312e+06    3.840312e+06  3.840312e+06
mean   1.904504e+06  2.783242e+05   -3.452192e+01  5.827686e+04
std    5.364695e+05  1.027045e+05    2.666775e+01  1.074641e+05
min    1.000018e+06  1.000060e+05   -9.600000e+01 -4.202502e+05
25%    1.434385e+06  1.895170e+05   -5.500000e+01  0.000000e+00
50%    1.897122e+06  2.783960e+05   -2.800000e+01  0.000000e+00
75%    2.369328e+06  3.675800e+05   -1.100000e+01  8.904669e+04
max    2.843496e+06  4.562500e+05   -1.000000e+00  1.505902e+06

       AMT_CREDIT_LIMIT_ACTUAL  AMT_DRAWINGS_ATM_CURRENT  \
count             3.840312e+06              3.090496e+06
mean              1.538080e+05              5.962299e+03
std               1.651457e+05              2.803397e+04
min               0.000000e+00             -6.827310e+03
25%               4.500000e+04              0.000000e+00
50%               1.125000e+05              0.000000e+00
75%               1.800000e+05              0.000000e+00
max               1.350000e+06              2.115000e+06

       AMT_DRAWINGS_CURRENT  AMT_DRAWINGS_OTHER_CURRENT  \
```

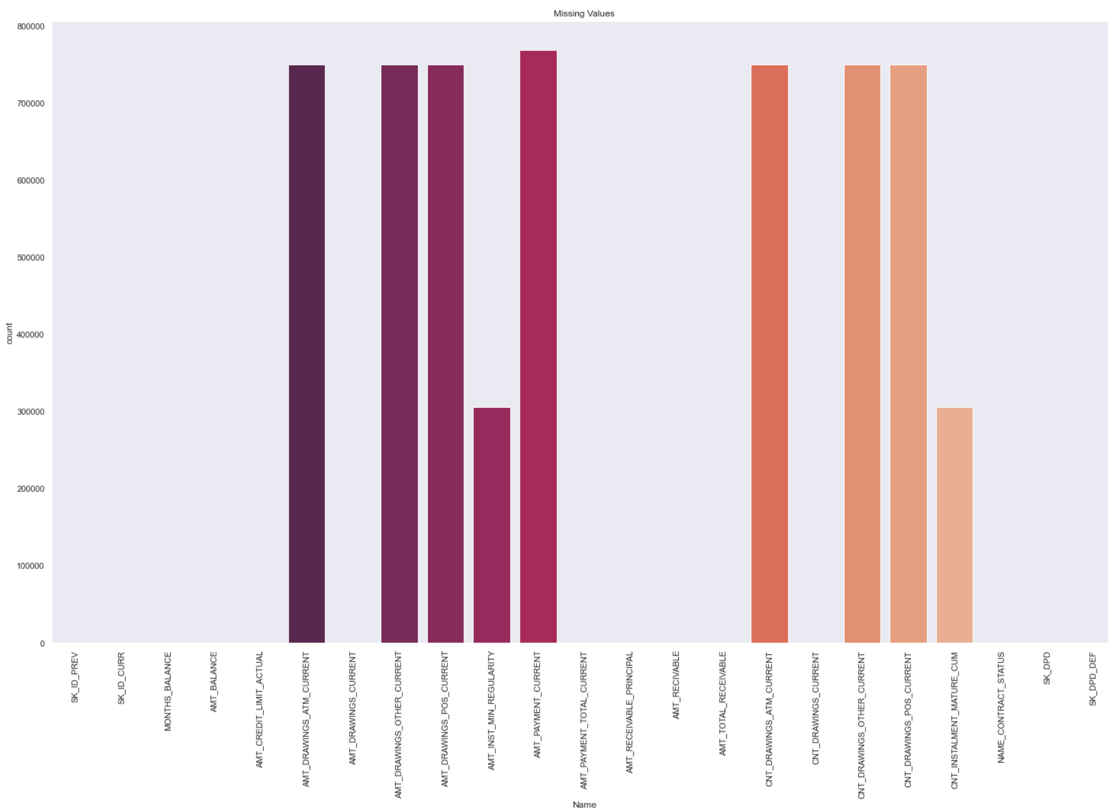
```
count        3.840312e+06                3.090496e+06
mean         7.432263e+03                2.881647e+02
std          3.336682e+04                8.197021e+03
min         -6.211620e+03                0.000000e+00
25%          0.000000e+00                0.000000e+00
50%          0.000000e+00                0.000000e+00
75%          0.000000e+00                0.000000e+00
max          2.287098e+06                1.529847e+06

       AMT_DRAWINGS_POS_CURRENT  AMT_INST_MIN_REGULARITY  ...  \
count              3.090496e+06             3.535076e+06  ...
mean               2.968840e+03             3.541778e+03  ...
std                2.066321e+04             5.525350e+03  ...
min                0.000000e+00             0.000000e+00  ...
25%                0.000000e+00             0.000000e+00  ...
50%                0.000000e+00             0.000000e+00  ...
75%                0.000000e+00             6.633911e+03  ...
max                2.239274e+06             2.028820e+05  ...

       AMT_RECEIVABLE_PRINCIPAL  AMT_RECIVABLE
AMT_TOTAL_RECEIVABLE  \
count              3.840312e+06   3.840312e+06          3.840312e+06

mean               5.595036e+04   5.808502e+04          5.809459e+04

std                1.015177e+05   1.071769e+05          1.071802e+05

min               -4.233058e+05  -4.202502e+05         -4.202502e+05

25%                0.000000e+00   0.000000e+00          0.000000e+00

50%                0.000000e+00   0.000000e+00          0.000000e+00

75%                8.535924e+04   8.889949e+04          8.891451e+04

max                1.472317e+06   1.493338e+06          1.493338e+06


       CNT_DRAWINGS_ATM_CURRENT  CNT_DRAWINGS_CURRENT  \
count                 3090496.0          3.840312e+06
mean                        NaN          7.031439e-01
std                         0.0          3.190347e+00
min                         0.0          0.000000e+00
25%                         0.0          0.000000e+00
50%                         0.0          0.000000e+00
75%                         0.0          0.000000e+00
max                        51.0          1.650000e+02

       CNT_DRAWINGS_OTHER_CURRENT  CNT_DRAWINGS_POS_CURRENT  \
```

```
count                3.090496e+06                 3090496.0
mean                 4.810333e-03                       NaN
std                  8.239746e-02                       0.0
min                  0.000000e+00                       0.0
25%                  0.000000e+00                       0.0
50%                  0.000000e+00                       0.0
75%                  0.000000e+00                       0.0
max                  1.200000e+01                     165.0

       CNT_INSTALMENT_MATURE_CUM        SK_DPD    SK_DPD_DEF
count                  3535076.0  3.840312e+06  3.840312e+06
mean                         NaN  9.283667e+00  3.316220e-01
std                          0.0  9.751570e+01  2.147923e+01
min                          0.0  0.000000e+00  0.000000e+00
25%                          4.0  0.000000e+00  0.000000e+00
50%                         15.0  0.000000e+00  0.000000e+00
75%                         32.0  0.000000e+00  0.000000e+00
max                        120.0  3.260000e+03  3.260000e+03

[8 rows x 22 columns]
```

Grouping featrues by type

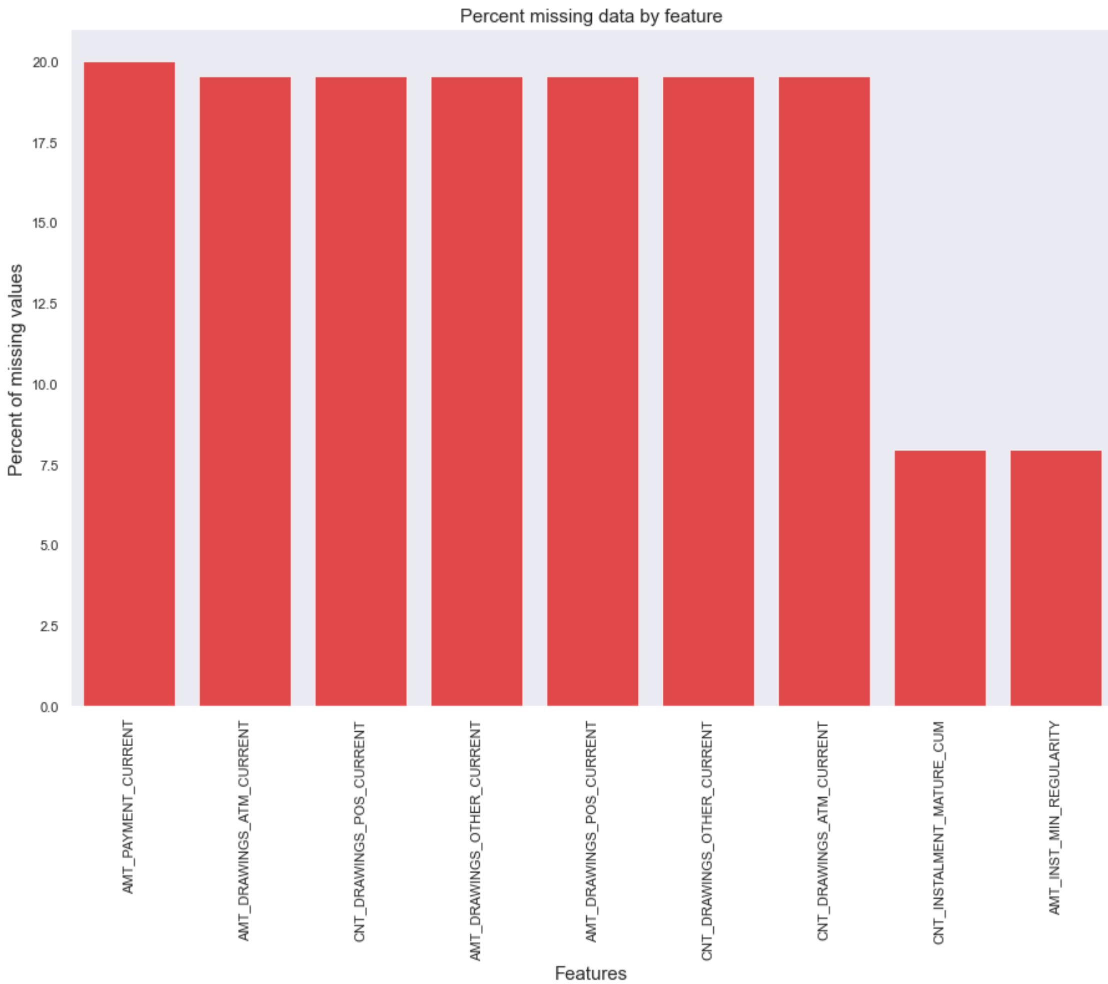
```
feature_type(datasets["credit_card_balance"])

numerical features: Index(['SK_ID_PREV', 'SK_ID_CURR',
'MONTHS_BALANCE', 'AMT_BALANCE',
       'AMT_CREDIT_LIMIT_ACTUAL', 'AMT_DRAWINGS_ATM_CURRENT',
       'AMT_DRAWINGS_CURRENT', 'AMT_DRAWINGS_OTHER_CURRENT',
       'AMT_DRAWINGS_POS_CURRENT', 'AMT_INST_MIN_REGULARITY',
       'AMT_PAYMENT_CURRENT', 'AMT_PAYMENT_TOTAL_CURRENT',
       'AMT_RECEIVABLE_PRINCIPAL', 'AMT_RECIVABLE',
'AMT_TOTAL_RECEIVABLE',
       'CNT_DRAWINGS_ATM_CURRENT', 'CNT_DRAWINGS_CURRENT',
       'CNT_DRAWINGS_OTHER_CURRENT', 'CNT_DRAWINGS_POS_CURRENT',
       'CNT_INSTALMENT_MATURE_CUM', 'SK_DPD', 'SK_DPD_DEF'],
      dtype='object')
**********************************************************************
******************************
categorical features : Index(['NAME_CONTRACT_STATUS'], dtype='object')

# Missing value in dataframe
missing_vals = (datasets['credit_card_balance'].isna().sum())
print('Missing values in dataframe ',missing_vals[missing_vals >
0].count())

Missing values in dataframe  9

missing_vals = pd.DataFrame(missing_vals)
missing_vals.columns = ['count']
missing_vals.index.names = ['Name']
missing_vals['Name'] = missing_vals.index
```

```
sns.set(style="dark", color_codes=True,rc={'figure.figsize':(25,15)})
sns.barplot(x = 'Name', y = 'count', data=missing_vals,
palette="rocket").set(title='Missing Values')
plt.xticks(rotation = 90)
plt.show()
```




```
missingFeatures(datasets["credit_card_balance"])
```

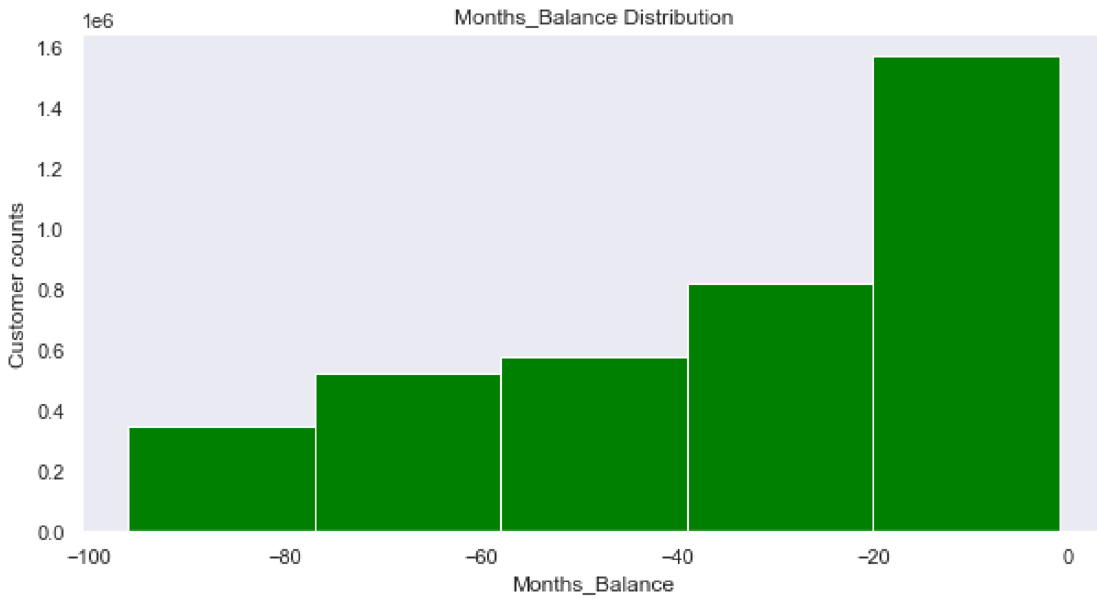
```
                             Total    Percent
AMT_PAYMENT_CURRENT         767988  19.998063
AMT_DRAWINGS_ATM_CURRENT    749816  19.524872
CNT_DRAWINGS_POS_CURRENT    749816  19.524872
AMT_DRAWINGS_OTHER_CURRENT  749816  19.524872
AMT_DRAWINGS_POS_CURRENT    749816  19.524872
CNT_DRAWINGS_OTHER_CURRENT  749816  19.524872
CNT_DRAWINGS_ATM_CURRENT    749816  19.524872
CNT_INSTALMENT_MATURE_CUM   305236   7.948208
AMT_INST_MIN_REGULARITY     305236   7.948208
```



Observation

9 features have missing data with most of them having more than 19% missing data.

```
plt.figure(figsize=(10,5))
plt.hist(datasets['credit_card_balance'][['MONTHS_BALANCE']].values,
bins=5,color='green',label=True)
plt.title('Months_Balance Distribution')
plt.xlabel('Months_Balance')
plt.ylabel('Customer counts')
plt.show()
```



Observation

Majority of customers have negative month balance

Installment Payments EDA

```
datasets['installments_payments'].describe()
```

```
         SK_ID_PREV    SK_ID_CURR  NUM_INSTALMENT_VERSION  \
count  1.360540e+07  1.360540e+07              13605401.0
mean   1.903365e+06  2.784449e+05                     NaN
std    5.362029e+05  1.027183e+05                     0.0
min    1.000001e+06  1.000010e+05                     0.0
25%    1.434191e+06  1.896390e+05                     0.0
50%    1.896520e+06  2.786850e+05                     1.0
75%    2.369094e+06  3.675300e+05                     1.0
max    2.843499e+06  4.562550e+05                   178.0

       NUM_INSTALMENT_NUMBER  DAYS_INSTALMENT  DAYS_ENTRY_PAYMENT  \
count           1.360540e+07       13605401.0          13602496.0
mean            1.887090e+01              NaN                 NaN
std             2.666407e+01              NaN                 NaN
min             1.000000e+00          -2922.0             -4920.0
25%             4.000000e+00          -1654.0             -1662.0
50%             8.000000e+00           -818.0              -827.0
75%             1.900000e+01           -361.0              -370.0
max             2.770000e+02             -1.0                -1.0

       AMT_INSTALMENT    AMT_PAYMENT
```

```
count    1.360540e+07  1.360250e+07
mean     1.675076e+04  1.691504e+04
std      4.964295e+04  5.375981e+04
min      0.000000e+00  0.000000e+00
25%      4.226085e+03  3.398265e+03
50%      8.884080e+03  8.125515e+03
75%      1.671021e+04  1.610842e+04
max      3.771488e+06  3.771488e+06
```

Grouping featrues by type

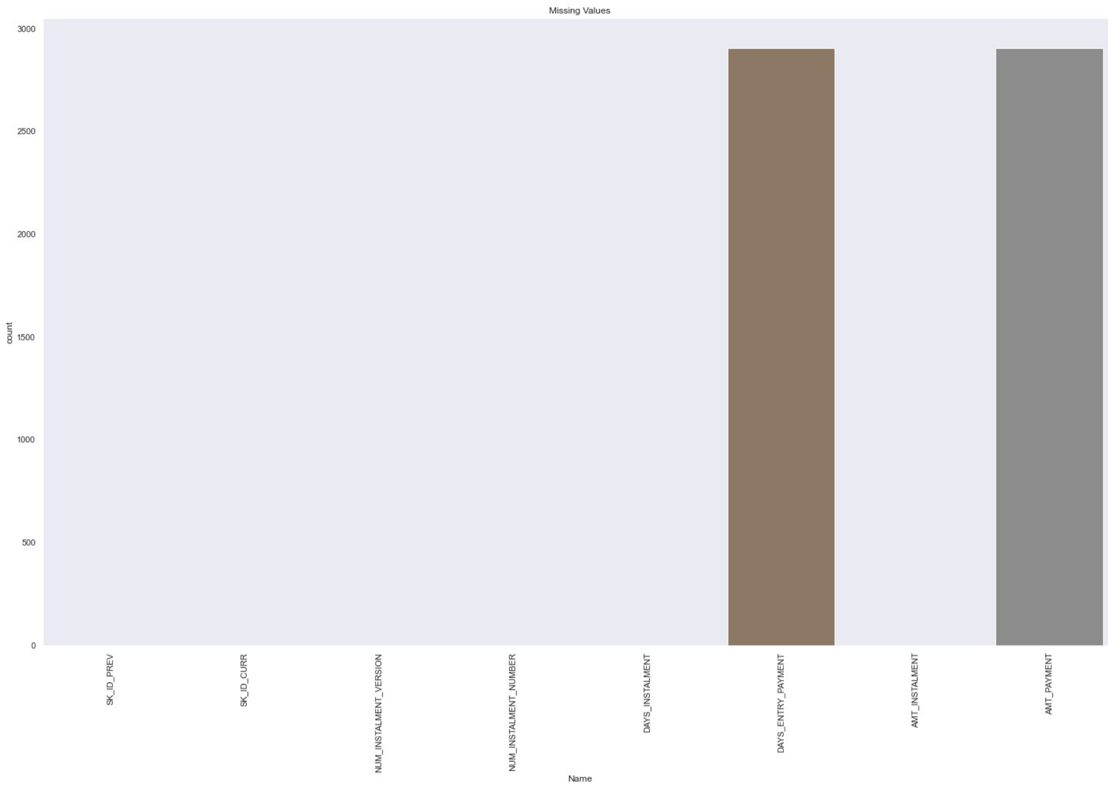
```
feature_type(datasets["installments_payments"])
```

```
numerical features: Index(['SK_ID_PREV', 'SK_ID_CURR',
'NUM_INSTALMENT_VERSION',
       'NUM_INSTALMENT_NUMBER', 'DAYS_INSTALMENT',
'DAYS_ENTRY_PAYMENT',
       'AMT_INSTALMENT', 'AMT_PAYMENT'],
      dtype='object')
**********************************************************************
******************************
categorical features : Index([], dtype='object')
```

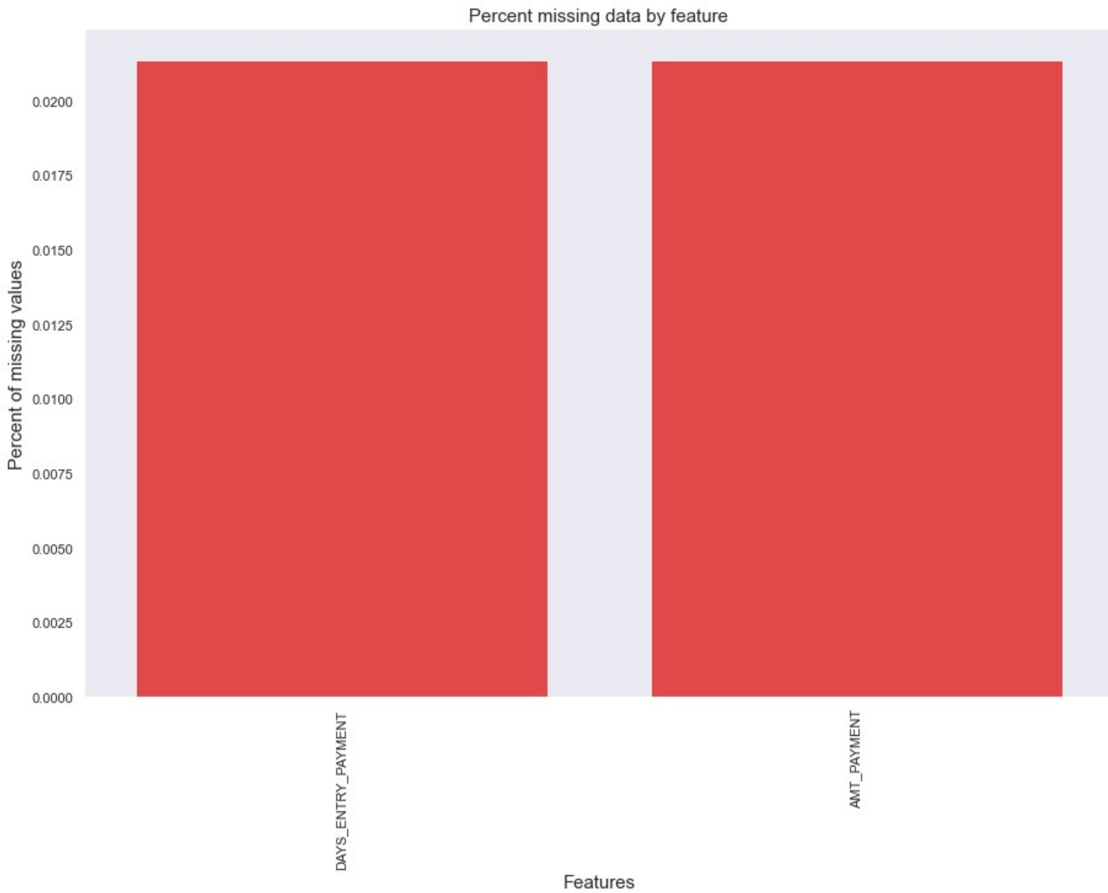
```
# Missing value in dataframe
missing_vals = (datasets['installments_payments'].isna().sum())
print('Missing values in dataframe ',missing_vals[missing_vals >
0].count())
```

```
Missing values in dataframe  2
```

```
missing_vals = pd.DataFrame(missing_vals)
missing_vals.columns = ['count']
missing_vals.index.names = ['Name']
missing_vals['Name'] = missing_vals.index

sns.set(style="dark", color_codes=True,rc={'figure.figsize':(25,15)})
sns.barplot(x = 'Name', y = 'count',
data=missing_vals).set(title='Missing Values')
plt.xticks(rotation = 90)
plt.show()
```




```
missingFeatures(datasets["installments_payments"])

                    Total   Percent
DAYS_ENTRY_PAYMENT   2905  0.021352
AMT_PAYMENT          2905  0.021352
```



Observation

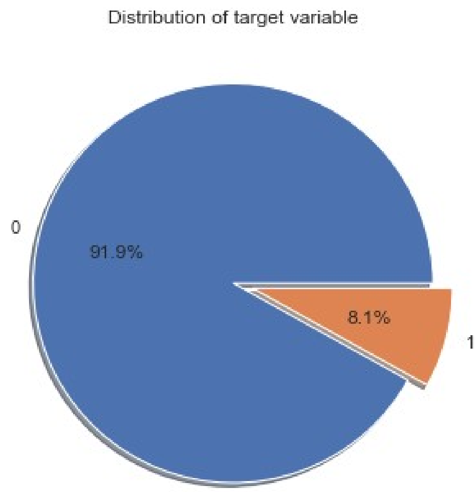
Only 2 features have missing values. Both the columns have less than .2% missing data

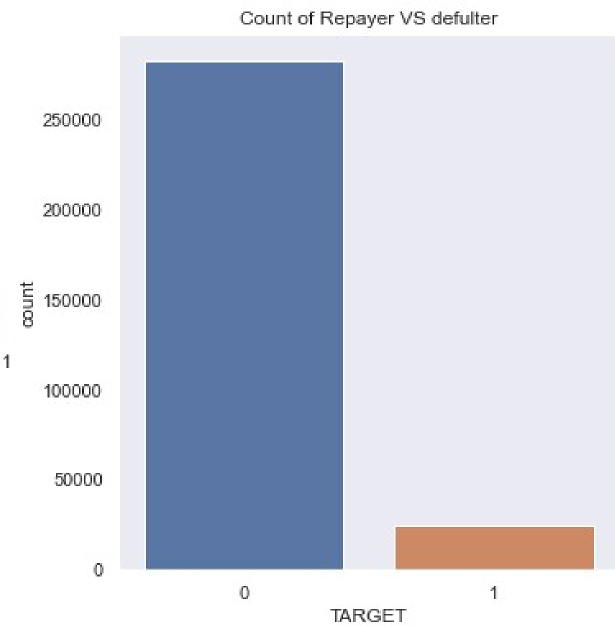
Lets Check if the Data is Balanced or Not

```
datasets['application_train'].TARGET.value_counts()
```

```
0    282686
1     24825
Name: TARGET, dtype: int64
```

```
f,ax=plt.subplots(1,2,figsize=(12,6))
datasets['application_train'].TARGET.value_counts().plot.pie(explode=[
0,0.1],autopct='%1.1f%%',ax=ax[0],shadow=True)
ax[0].set_title('Distribution of target variable')
ax[0].set_ylabel('')
sns.countplot('TARGET',data=datasets['application_train'],ax=ax[1])
ax[1].set_title('Count of Repayer VS defulter')
plt.show()
```





We see that a large chunk of customers (about 92%) payed back on time and about 8% did not.This shows that the data is highly imbalanced. We need to choose a correct metric while evaluating our models, so that imbalnced data doesn't give us a false evaluation our model.

Merging data and building Baseline model

Removing Null Values from Aplication train

```
#A total of 278 datasets['application_train']points are there where Annuity Amount is null.
datasets['application_train']['NAME_TYPE_SUITE'].fillna('NA', inplace=True)

datasets['application_train']['EXT_SOURCE_3'].fillna(0, inplace=True)

#A total of 36 datasets['application_train']points are there where Annuity Amount is null.
datasets['application_train']['AMT_GOODS_PRICE'].fillna(0, inplace=True)
datasets['application_train']['OCCUPATION_TYPE'].fillna('NA', inplace=True)
datasets['application_train']['EXT_SOURCE_1'].fillna(0, inplace=True)
datasets['application_train']['EXT_SOURCE_2'].fillna(0, inplace=True)
datasets['application_train']['NAME_FAMILY_STATUS'].fillna('NA', inplace=True)
datasets['application_train']['NAME_HOUSING_TYPE'].fillna('NA', inplace=True)
```

```
#Days Employed value for 1 row has been filled in wrong.
datasets['application_train'].replace(max(datasets['application_train'
]['DAYS_EMPLOYED'].values), np.nan, inplace=True)
datasets['application_train']
['CODE_GENDER'].replace('XNA','M',inplace=True)
#There are a total of 4 applicants with Gender provided as 'XNA'
datasets['application_train']['AMT_ANNUITY'].fillna(0, inplace=True)
datasets['application_train']['FLAG_MOBIL'].fillna('NA', inplace=True)
datasets['application_train']['FLAG_EMP_PHONE'].fillna('NA',
inplace=True)
datasets['application_train']['FLAG_CONT_MOBILE'].fillna('NA',
inplace=True)
datasets['application_train']['FLAG_EMAIL'].fillna('NA', inplace=True)
datasets['application_train']['OCCUPATION_TYPE'].fillna('NA',
inplace=True)
datasets['application_train']
['CNT_FAM_MEMBERS'].fillna(0,inplace=True)
datasets['previous_application']
['DAYS_TERMINATION'].replace(max(datasets['previous_application']
['DAYS_TERMINATION'].values),np.nan, inplace=True)
datasets['application_train']=
datasets['application_train'].drop(['FLAG_DOCUMENT_2','FLAG_DOCUMENT_4
','FLAG_DOCUMENT_5','FLAG_DOCUMENT_6','FLAG_DOCUMENT_7',
    'FLAG_DOCUMENT_8','FLAG_DOCUMENT_9','FLAG_DOCUMENT_10',
'FLAG_DOCUMENT_11','FLAG_DOCUMENT_12','FLAG_DOCUMENT_13',

'FLAG_DOCUMENT_14','FLAG_DOCUMENT_15','FLAG_DOCUMENT_16','FLAG_DOCUMEN
T_17','FLAG_DOCUMENT_18','FLAG_DOCUMENT_19',
    'FLAG_DOCUMENT_20','FLAG_DOCUMENT_21'],axis=1)
```

Feature Aggregation Class for Aggregation in Pipeline

```
class FeatureAggregator(BaseEstimator,TransformerMixin):
    def __init__(self,dataset,features):
        self.features=features
        self.dataset=dataset
        self.agg_ops=['min','max','mean','sum']

    def fit(self,X,y=None):
        return self

    def transform(self,X,y=None):
        result=X.groupby(['SK_ID_CURR']).agg(self.agg_ops)
        result.columns=["_".join(x) for x in result.columns.ravel()]
        result=result.reset_index(level=["SK_ID_CURR"])
        return result
```

Merging Datasets together

Merging Credit Card Balance Dataset with Application Train|Test

```
creditC_df=datasets['credit_card_balance']

#one hot encoding credit card data
creditC_df=ohe(creditC_df)


creditC_features=["MONTHS_BALANCE","AMT_BALANCE","CNT_INSTALMENT_MATUR
E_CUM"]
creditC_df =
creditC_df.groupby(["SK_ID_CURR"],as_index=False).agg("mean")
creditC_bal_pipeline=Pipeline([

("creditC_aggregator",FeatureAggregator(creditC_df,creditC_features))
])

creditC_bal_agg=creditC_bal_pipeline.transform(creditC_df)
creditC_df=creditC_df.merge(creditC_bal_agg,how='left',on=['SK_ID_CURR
'])
rename(creditC_df,"creditC")

creditC_df.shape

(103558, 141)

creditC_df['SK_ID_CURR'].nunique()

103558
```

Merging POS_CASH_Balance Dataset with Application Train|Test

```
pos_cash_df=datasets['POS_CASH_balance']

#One hot encoding
pos_cash_df=ohe(pos_cash_df)

pos_cash_features=['SK_DPD_DEF','SK_DPD','MONTHS_BALANCE','CNT_INSTALM
ENT','CNT_INSTALMENT_FUTURE']
pos_cash_df=pos_cash_df.groupby(["SK_ID_CURR"],as_index=False).agg("me
an")
pos_cash_pipeline=Pipeline([

("pos_cash_cash_aggregator",FeatureAggregator(pos_cash_df,pos_cash_fea
tures))
])

pos_cash_agg=pos_cash_pipeline.transform(pos_cash_df)
pos_cash_df=pos_cash_df.merge(pos_cash_agg,how='left',on=['SK_ID_CURR'
```

```
])
rename(pos_cash_df,"pos_cash")

pos_cash_df.shape
```

```
(337252, 76)
```

```
pos_cash_df['SK_ID_CURR'].nunique()
```

```
337252
```

Preparing Installment Paymets for Merging

```
ins_pay_df=datasets['installments_payments']

#onehot encoding
ins_pay_df=ohe(ins_pay_df)

ins_pay_features=['AMT_INSTALMENT','DAYS_ENTRY_PAYMENT','AMT_PAYMENT']
ins_pay_df=ins_pay_df.groupby(["SK_ID_CURR"],as_index=False).agg("mean
")
ins_pay_pipeline=Pipeline([

("ins_pay_aggregator",FeatureAggregator(ins_pay_df,ins_pay_features))
])

ins_pay_agg=ins_pay_pipeline.transform(ins_pay_df)
ins_pay_df=ins_pay_df.merge(ins_pay_agg,how='left',on=['SK_ID_CURR'])
rename(ins_pay_df,"ins_pay")

ins_pay_df.shape
```

```
(339587, 36)
```

Preparing Bureau and Bureau Balance for merging

```
bur_df=datasets['bureau']

#onehot encoding
bur_df=ohe(bur_df)
bur_df2=bur_df[['SK_ID_CURR']]
bur_df2['appcount']=1
bur_df2=bur_df2.groupby(['SK_ID_CURR'],as_index=False).agg("sum")
bur_features=["AMT_ANNUITY","AMT_CREDIT_SUM","AMT_CREDIT_SUM_DEBT","AM
T_CREDIT_SUM_OVERDUE","AMT_CREDIT_SUM_LIMIT","CNT_CREDIT_PROLONG","DAY
S_CREDIT_UPDATE","DAYS_CREDIT_ENDDATE","CREDIT_DAY_OVERDUE","AMT_CREDI
T_MAX_OVERDUE","DAYS_CREDIT"]
bur_df=bur_df.groupby(["SK_ID_CURR"],as_index=False).agg("mean")

bur_pipeline=Pipeline([

('bur_aggregator',FeatureAggregator(bur_df,bur_features))
```

```
                        ])


bur_agg=bur_pipeline.transform(bur_df)
bur_df=bur_df.merge(bur_agg,how='left',on='SK_ID_CURR')
rename(bur_df,"bur")
bur_df=bur_df.merge(bur_df2,how="left",on="SK_ID_CURR")

bur_df.shape
```

```
(305811, 182)
```

```
bur_bal_df=datasets['bureau_balance']

#onehot encoding
bur_bal_df=ohe(bur_bal_df)

bur_bal_features=["MONTHS_BALANCE"]
bur_bal_df=bur_bal_df.groupby(["SK_ID_BUREAU"],as_index=False).agg("me
an")
bur_bal_df=bur_bal_df.groupby(["SK_ID_BUREAU"],as_index=False).agg({f"
{feature}":["min","max","mean","sum"] for feature in
["MONTHS_BALANCE"]})
bur_bal_df.columns=["_".join(x) for x in bur_bal_df.columns.ravel()]
bur_bal_df.columns=pd.Index(['bur_bal_'+col for col in
list(bur_bal_df.columns)])


bur_bal_df.rename(columns={"bur_bal_SK_ID_BUREAU_":"SK_ID_BUREAU"},inp
lace=True)
bur_bal_df.rename(columns={"SK_ID_BUREAU":"SK_ID_CURR"},inplace=True)
bur_df.shape
```

```
(305811, 182)
```

```
bur_df=bur_df.merge(bur_bal_df,how='left',on='SK_ID_CURR')
bur_df.shape
```

```
(305811, 186)
```

Preparing Application Dataset for merging

```
prev_app_df=datasets['previous_application']

#onehot encoding
prev_app_df=ohe(prev_app_df)

prev_app_features=['AMT_ANNUITY','AMT_APPLICATION','AMT_CREDIT','AMT_D
OWN_PAYMENT','AMT_GOODS_PRICE','CNT_PAYMENT','DAYS_DECISION','HOUR_APP
R_PROCESS_START','RATE_DOWN_PAYMENT']
```

```
prev_app_df=prev_app_df.groupby(["SK_ID_CURR"],as_index=False).agg('me
an')
prev_app_pipeline=Pipeline([
("prev_app_aggregator",FeatureAggregator(prev_app_df,prev_app_features
))
])

prev_app_agg=prev_app_pipeline.transform(prev_app_df)
prev_app_df=prev_app_df.merge(prev_app_agg,how='left',on=['SK_ID_CURR'
])
rename(prev_app_df,"pa")
```

```
prev_app_df.shape
```

```
(338857, 816)
```

```
prev_app_df['SK_ID_CURR'].nunique()
```

```
338857
```

Merging all Sub-dataframes together

```
app_df=datasets['application_train']
app_test_df=datasets['application_test']


app_df=app_df.merge(bur_df,how='left',on='SK_ID_CURR')
app_test_df=app_test_df.merge(bur_df,how='left',on='SK_ID_CURR')


app_df=app_df.merge(prev_app_df,how='left',on='SK_ID_CURR')
app_test_df=app_test_df.merge(prev_app_df,how='left',on='SK_ID_CURR')


app_df=app_df.merge(creditC_df,how='left',on='SK_ID_CURR')
app_test_df=app_test_df.merge(creditC_df,how='left',on='SK_ID_CURR')

app_df=app_df.merge(ins_pay_df,how='left',on="SK_ID_CURR")
app_test_df=app_test_df.merge(ins_pay_df,how='left',on='SK_ID_CURR')

app_df=app_df.merge(pos_cash_df,how='left',on='SK_ID_CURR')
app_test_df=app_test_df.merge(pos_cash_df,how='left',on='SK_ID_CURR')
```

```
%%time
print("Optimizing memory After Merging")
app_test_df=optimize_memory(app_test_df)
app_df=optimize_memory(app_df)
```

```
Optimizing memory After Merging
Before Optimization : DataFrame Memory 430.5524139404297
After Optimization : DataFrame Memory 148.1040802001953
Before Optimization : DataFrame Memory 2713.2909507751465
After Optimization : DataFrame Memory 928.4780750274658
Wall time: 3min 1s
```

Saving final dataframe

```
import pickle
print("SAVING: trainig dataframe.....")
with open('app_df.pkl', 'wb') as file:
    pickle.dump(app_df, file)

print("SAVED: trainig dataframe")

print("SAVING: test dataframe.....")
with open('app_test_df.pkl', 'wb') as file:
    pickle.dump(app_test_df, file)
print("SAVED: test dataframe")
```

```
SAVING: trainig dataframe.....
SAVED: trainig dataframe
SAVING: test dataframe.....
SAVED: test dataframe
```

Finding Correlation between Merged data and Target feature.

```
# %%time
# correlations=np.abs(app_df.corr()['TARGET'])
```

Seperating categorical and numerical features

```
numerical_feat=list(app_df.loc[:,
~app_df.columns.isin(['TARGET'])]._get_numeric_data().columns)
print("number of numerical features: ", len(numerical_feat))


categorical_feat=list(app_df.select_dtypes(include="object").columns.v
alues)
print("number of categorical features: ", len(categorical_feat))
```

```
number of numerical features:  1336
number of categorical features:  16
```

Top50 numerical features which are highly correlated to the Target feature.

```
# corr_num=np.abs(app_df.loc[:,
app_df.columns.isin(numerical_feat)].corr()
['TARGET']).sort_values(ascending=False)
```

```
# trainer_data=pd.read_pickle("app_df.pkl")

# numvar_top50=list(corr_num.index[1:51])
# corr_num=np.abs(trainer_data.loc[:,
trainer_data.columns.isin(numvar_top50+['TARGET'])].corr()
['TARGET']).sort_values(ascending=False)
# corr_num
```

Top50 categorical features which are highly correlated to the Target feature.

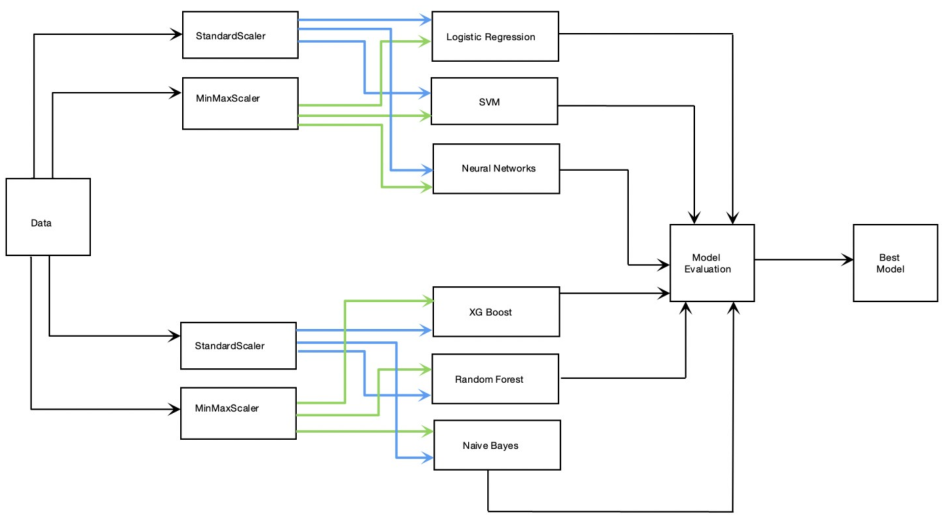
```
# cat_corr_num=np.abs(app_df.loc[:,
app_df.columns.isin(categorical_feat)].corr()
['TARGET']).sort_values(ascending=False)

# trainer_data=pd.read_pickle("app_df.pkl")

# catvar_top50=list(corr_num.index[1:51])
# cat_corr_num=np.abs(trainer_data.loc[:,
trainer_data.columns.isin(catvar_top50+['TARGET'])].corr()
['TARGET']).sort_values(ascending=False)
# cat_corr_num
```

Separting train and test data

```
selected_features = ['AMT_INCOME_TOTAL',
'AMT_CREDIT','DAYS_EMPLOYED','DAYS_BIRTH','EXT_SOURCE_1',
        'EXT_SOURCE_2','EXT_SOURCE_3','CODE_GENDER',
'FLAG_OWN_REALTY','FLAG_OWN_CAR','NAME_CONTRACT_TYPE',

'NAME_EDUCATION_TYPE','OCCUPATION_TYPE','NAME_INCOME_TYPE']

X=app_df.drop(['TARGET'],axis=1)
y=app_df['TARGET']
X_kaggle_test= datasets["application_test"][selected_features]

X_train, X_valid, y_train, y_valid = train_test_split(X, y,
test_size=0.15, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,
test_size=0.15, random_state=42)
X_kaggle_test= X_kaggle_test[selected_features]

print("Train data shape: ", X_train.shape)
print("Test data shape: ", X_valid.shape)
print("Test data shape: ", X_test.shape)
```

```
Train data shape:  (222176, 1352)
Test data shape:  (46127, 1352)
Test data shape:  (39208, 1352)
```

Pipleine



We will use pipeline to prepare our data for the predictions.

Pipleines consist of:

1. custom DataFrameSelector which slelcts the given features
2. Imputer for imputing missing values
3. MinMax scaler for bringing all the values on the same scale.

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import log_loss

#custom dataframeSelector for preparing numerical and categorical
pipelines
class DataFrameSelector(BaseEstimator, TransformerMixin):
#     initialize with given ferature names
    def __init__(self, feature_name):
        self.feature_name = feature_name

    #fit function that will return the object
    def fit(self, X, y=None):
```

```
        return self

    #Trnasform function that will return the requested features
    def transform(self, X):
        return X[self.feature_name].values


#pipeline for preparing numerical features
numerical_pipeline = Pipeline([
        ('selector', DataFrameSelector(numerical_feat)),
        ('imputer', SimpleImputer(strategy='mean')),
        ('min_max_scaler', MinMaxScaler()),
    ])


#Pipoeline for preparing categorical features
catagorical_pipeline = Pipeline([
        ('selector', DataFrameSelector(categorical_feat)),
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
    ])


#Pipeline combining numerical and categorical pipelines
data_pipeline = FeatureUnion(transformer_list=[
        ("numerical_pipeline", numerical_pipeline),
        ("catagorical_pipeline", catagorical_pipeline),
    ])
```

Creating the full pipeline with data preparation and base classifier

1. Naive Bayes

```
from sklearn.naive_bayes import MultinomialNB

np.random.seed(42)

#Creating full pipeline
full_pipeline = Pipeline([
        ("data_pipeline", data_pipeline),
        ("MNB", MultinomialNB())
    ])
model = full_pipeline.fit(X_train, y_train)

y_pred = model.predict(X_train)

print("Acuracy is : ",np.round(accuracy_score(y_train, y_pred), 3))
print("ROC is : ",np.round(roc_auc_score(y_train,
model.predict_proba(X_train)[:, 1]),3))
```

```
print("F1 score is : ",np.round(f1_score(y_train,
y_pred,average='weighted'), 3))
print("Precision is : ",np.round(precision_score(y_train, y_pred), 3))
print("Recall is : ",np.round(recall_score(y_train, y_pred), 3))
print("Log loss is : ",np.round(log_loss(y_train, y_pred), 3))
```

```
Acuracy is :  0.8
ROC is :  0.652
F1 score is :  0.832
Precision is :  0.16
Recall is :  0.352
Log loss is :  6.922
```

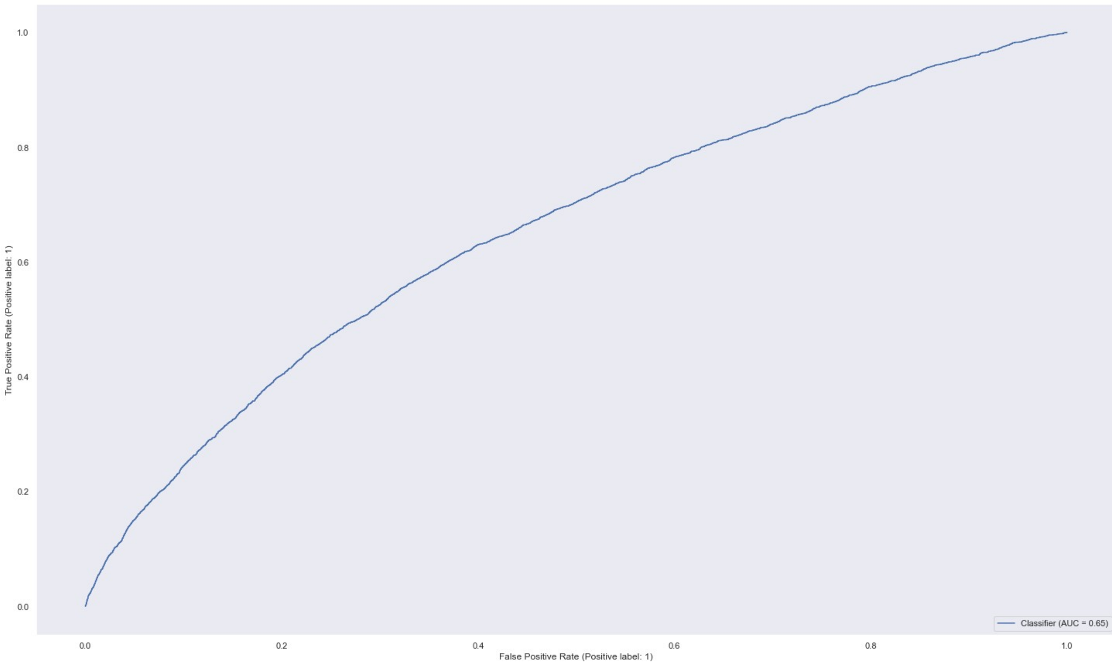
Logging results

```
data = {'Model': [],
       'Accuracy': [],
       'ROC_AUC': [],
       'F1-Score': [],
       'Precision': [],
       'Recall': [],
       'Log-Loss': []}

model_score = pd.DataFrame(data)

y_pred = model.predict(X_test)

model_score.loc[len(model_score)] = ["(Base)Naive Bayes",
                                    np.round(accuracy_score(y_test,
y_pred), 3),
                                     np.round(roc_auc_score(y_test,
model.predict_proba(X_test)[:, 1]),3),
                                     np.round(f1_score(y_test,
y_pred,average='weighted'), 3),
                                     np.round(precision_score(y_test,
y_pred), 3),
                                     np.round(recall_score(y_test,
y_pred), 3),
                                     np.round(log_loss(y_test,
y_pred), 3)

                                    ]

model_score
```

```
               Model  Accuracy  ROC_AUC  F1-Score  Precision  Recall
Log-Loss
0  (Base)Naive Bayes     0.799    0.652     0.829      0.164    0.34
6.945
```

Plotting ROC Curve

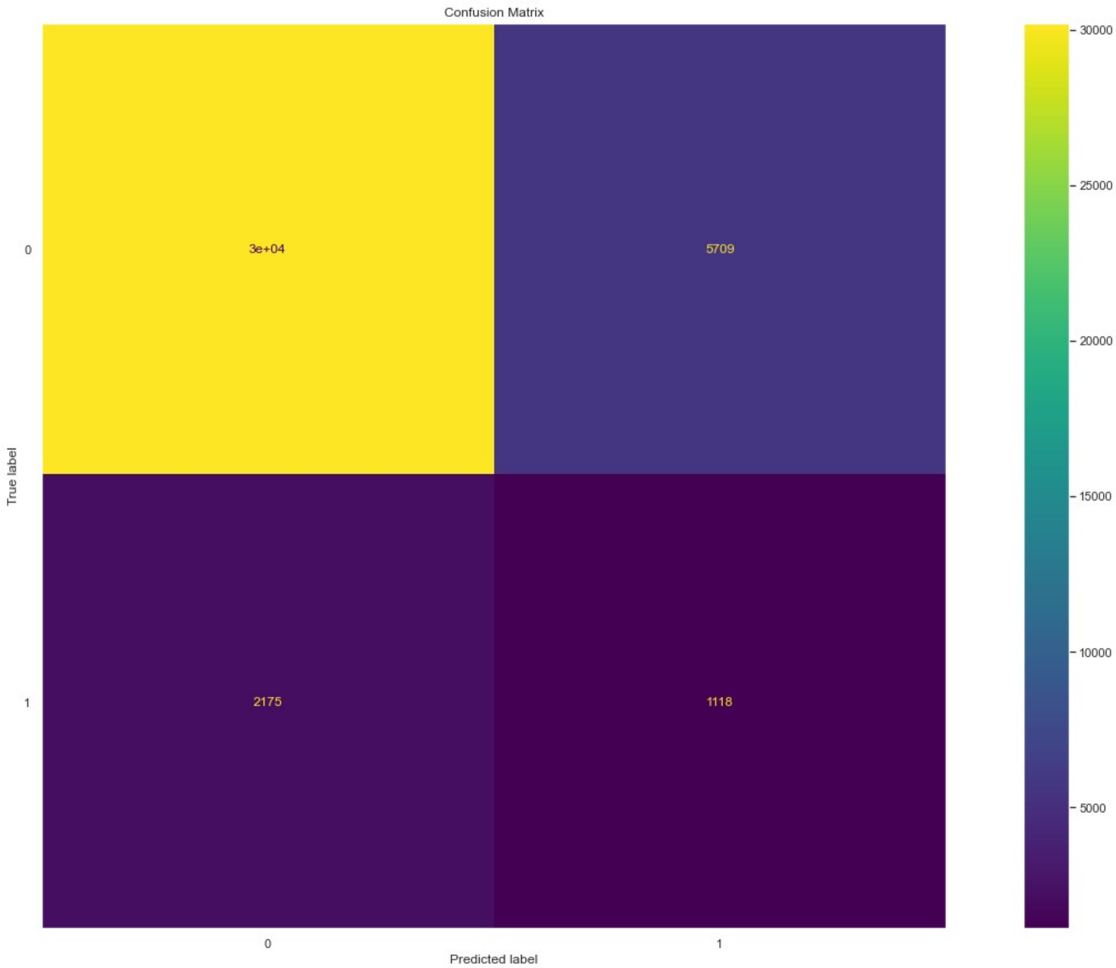
```
import matplotlib.pyplot as plt
from sklearn.metrics import RocCurveDisplay
RocCurveDisplay.from_predictions(y_test, model.predict_proba(X_test)
[:, 1])
plt.show()
```



Confusion matrix for testing data

```
from sklearn.metrics import plot_confusion_matrix
plt.clf()
plot_confusion_matrix(model, X_test, y_test)
plt.title('Confusion Matrix')
plt.show()
```

```
<Figure size 1800x1080 with 0 Axes>
```



Logistic Regression

```
from sklearn.naive_bayes import MultinomialNB

np.random.seed(42)

#Creating full pipeline
full_pipeline = Pipeline([
        ("data_pipeline", data_pipeline),
        ("linear", LogisticRegression())
    ])
model = full_pipeline.fit(X_train, y_train)

y_pred = model.predict(X_train)

print("Acuracy is : ",np.round(accuracy_score(y_train, y_pred), 3))
print("ROC is : ",np.round(roc_auc_score(y_train,
model.predict_proba(X_train)[:, 1]),3))
print("F1 score is : ",np.round(f1_score(y_train,
y_pred,average='weighted'), 3))
print("Precision is : ",np.round(precision_score(y_train, y_pred), 3))
```

```
print("Recall is : ",np.round(recall_score(y_train, y_pred), 3))
print("Log loss is : ",np.round(log_loss(y_train, y_pred), 3))

---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_11068/2906165806.py in <module>
----> 1 y_pred = model.predict(X_train)
      2 
      3 print("Acuracy is : ",np.round(accuracy_score(y_train, y_pred), 3))
      4 print("ROC is : ",np.round(roc_auc_score(y_train, model.predict_proba(X_train)[:, 1]),3))
      5 print("F1 score is : ",np.round(f1_score(y_train, y_pred,average='weighted'), 3))

~\Anaconda3\lib\site-packages\sklearn\utils\metaestimators.py in <lambda>(*args, **kwargs)
    111 
    112             # lambda, but not partial, allows help() to work with update_wrapper
--> 113             out = lambda *args, **kwargs: self.fn(obj, *args, **kwargs)  # noqa
    114         else:
    115 

~\Anaconda3\lib\site-packages\sklearn\pipeline.py in predict(self, X, **predict_params)
    467         Xt = X
    468         for _, name, transform in self._iter(with_final=False):
--> 469             Xt = transform.transform(Xt)
    470         return self.steps[-1][1].predict(Xt, **predict_params)
    471 

~\Anaconda3\lib\site-packages\sklearn\pipeline.py in transform(self, X)
   1222         Xs = Parallel(n_jobs=self.n_jobs)(
   1223             delayed(_transform_one)(trans, X, None, weight)
-> 1224             for name, trans, weight in self._iter()
   1225         )
   1226         if not Xs:

~\Anaconda3\lib\site-packages\joblib\parallel.py in __call__(self, iterable)
   1041             # remaining jobs.
   1042             self._iterating = False
-> 1043             if self.dispatch_one_batch(iterator):
   1044                 self._iterating = self._original_iterator is
```

```
not None
   1045

~\Anaconda3\lib\site-packages\joblib\parallel.py in
dispatch_one_batch(self, iterator)
    859                 return False
    860             else:
--> 861                 self._dispatch(tasks)
    862                 return True
    863

~\Anaconda3\lib\site-packages\joblib\parallel.py in _dispatch(self,
batch)
    777         with self._lock:
    778             job_idx = len(self._jobs)
--> 779             job = self._backend.apply_async(batch,
callback=cb)
    780             # A job can complete so quickly than its callback
is
    781             # called before we get here, causing self._jobs to

~\Anaconda3\lib\site-packages\joblib\_parallel_backends.py in
apply_async(self, func, callback)
    206     def apply_async(self, func, callback=None):
    207         """Schedule a func to be run"""
--> 208         result = ImmediateResult(func)
    209         if callback:
    210             callback(result)

~\Anaconda3\lib\site-packages\joblib\_parallel_backends.py in
__init__(self, batch)
    570         # Don't delay the application, to avoid keeping the
input
    571         # arguments in memory
--> 572         self.results = batch()
    573
    574     def get(self):

~\Anaconda3\lib\site-packages\joblib\parallel.py in __call__(self)
    261         with parallel_backend(self._backend,
n_jobs=self._n_jobs):
    262             return [func(*args, **kwargs)
--> 263                     for func, args, kwargs in self.items]
    264
    265     def __reduce__(self):

~\Anaconda3\lib\site-packages\joblib\parallel.py in <listcomp>(.0)
    261         with parallel_backend(self._backend,
n_jobs=self._n_jobs):
    262             return [func(*args, **kwargs)
```

```
--> 263                     for func, args, kwargs in self.items]
    264
    265     def __reduce__(self):

~\Anaconda3\lib\site-packages\sklearn\utils\fixes.py in __call__(self,
*args, **kwargs)
    214     def __call__(self, *args, **kwargs):
    215         with config_context(**self.config):
--> 216             return self.function(*args, **kwargs)
    217
    218

~\Anaconda3\lib\site-packages\sklearn\pipeline.py in
_transform_one(transformer, X, y, weight, **fit_params)
    874
    875 def _transform_one(transformer, X, y, weight, **fit_params):
--> 876     res = transformer.transform(X)
    877     # if we have a weight for this transformer, multiply
output
    878     if weight is None:

~\Anaconda3\lib\site-packages\sklearn\utils\metaestimators.py in
<lambda>(*args, **kwargs)
    111
    112             # lambda, but not partial, allows help() to work
with update_wrapper
--> 113             out = lambda *args, **kwargs: self.fn(obj, *args,
**kwargs)  # noqa
    114         else:
    115

~\Anaconda3\lib\site-packages\sklearn\pipeline.py in transform(self,
X)
    645         Xt = X
    646         for _, _, transform in self._iter():
--> 647             Xt = transform.transform(Xt)
    648         return Xt
    649

~\AppData\Local\Temp/ipykernel_11068/3612033971.py in transform(self,
X)
     19     #Trnasform function that will return the requested
features
     20     def transform(self, X):
---> 21         return X[self.feature_name].values
     22
     23

~\Anaconda3\lib\site-packages\pandas\core\frame.py in values(self)
  10662         """
```

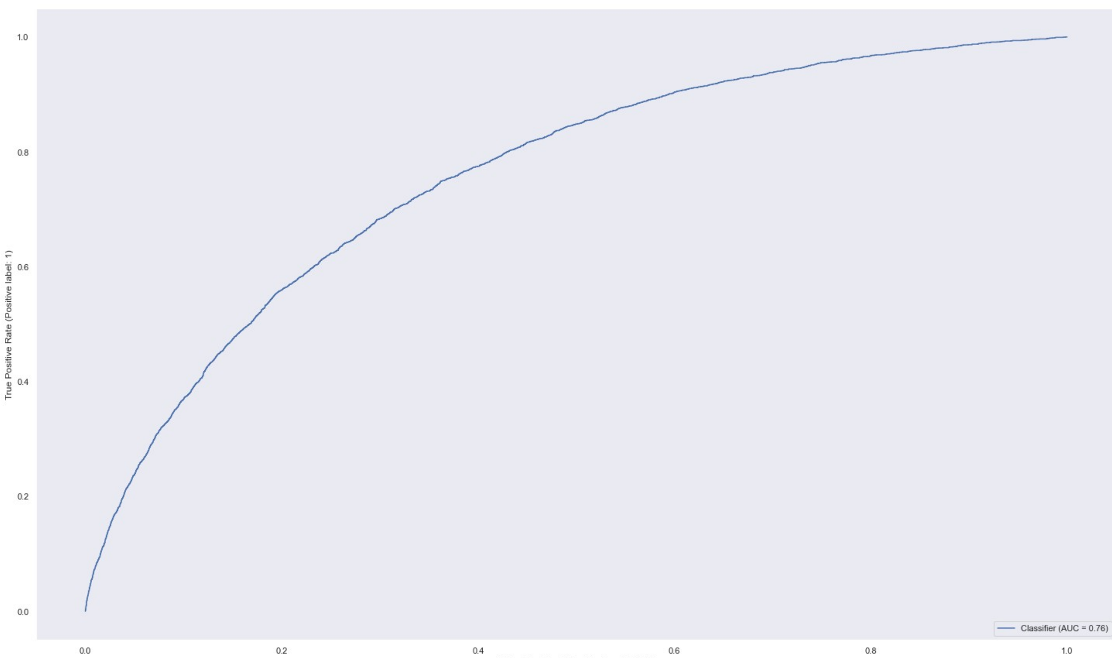
```
  10663         self._consolidate_inplace()
> 10664         return self._mgr.as_array(transpose=True)
  10665
  10666     @deprecate_nonkeyword_arguments(version=None,
allowed_args=["self"])

~\Anaconda3\lib\site-packages\pandas\core\internals\managers.py in
as_array(self, transpose, dtype, copy, na_value)
   1464                     arr = arr.astype(dtype, copy=False)  #
type: ignore[arg-type]
   1465         else:
-> 1466             arr = self._interleave(dtype=dtype,
na_value=na_value)
   1467             # The underlying data was copied within
_interleave
   1468             copy = False

~\Anaconda3\lib\site-packages\pandas\core\internals\managers.py in
_interleave(self, dtype, na_value)
   1500         # Tuple[Any, Union[int, Sequence[int]]], List[Any],
_DTypeDict,
   1501         # Tuple[Any, Any]]]"
-> 1502         result = np.empty(self.shape, dtype=dtype)  # type:
ignore[arg-type]
   1503
   1504         itemmask = np.zeros(self.shape[0])

MemoryError: Unable to allocate 2.21 GiB for an array with shape
(1336, 222176) and data type float64
```

```
y_pred = model.predict(X_test)
model_score.loc[len(model_score)] = ["(Base)Logistic Regression",
                                    np.round(accuracy_score(y_test,
y_pred), 3),
                                     np.round(roc_auc_score(y_test,
model.predict_proba(X_test)[:, 1]),3),
                                     np.round(f1_score(y_test,
y_pred,average='weighted'), 3),
                                     np.round(precision_score(y_test,
y_pred), 3),
                                     np.round(recall_score(y_test,
y_pred), 3),
                                     np.round(log_loss(y_test,
y_pred), 3)

                                    ]
```

```
model_score

                        Model  Accuracy  ROC_AUC  F1-Score  Precision
Recall  \
0          (Base)Naive Bayes     0.799    0.652     0.829      0.164
0.340
1  (Base)Logistic Regression     0.916    0.758     0.878      0.525
0.016

   Log-Loss
0     6.945
1     2.896
```

Plotting ROC Curve

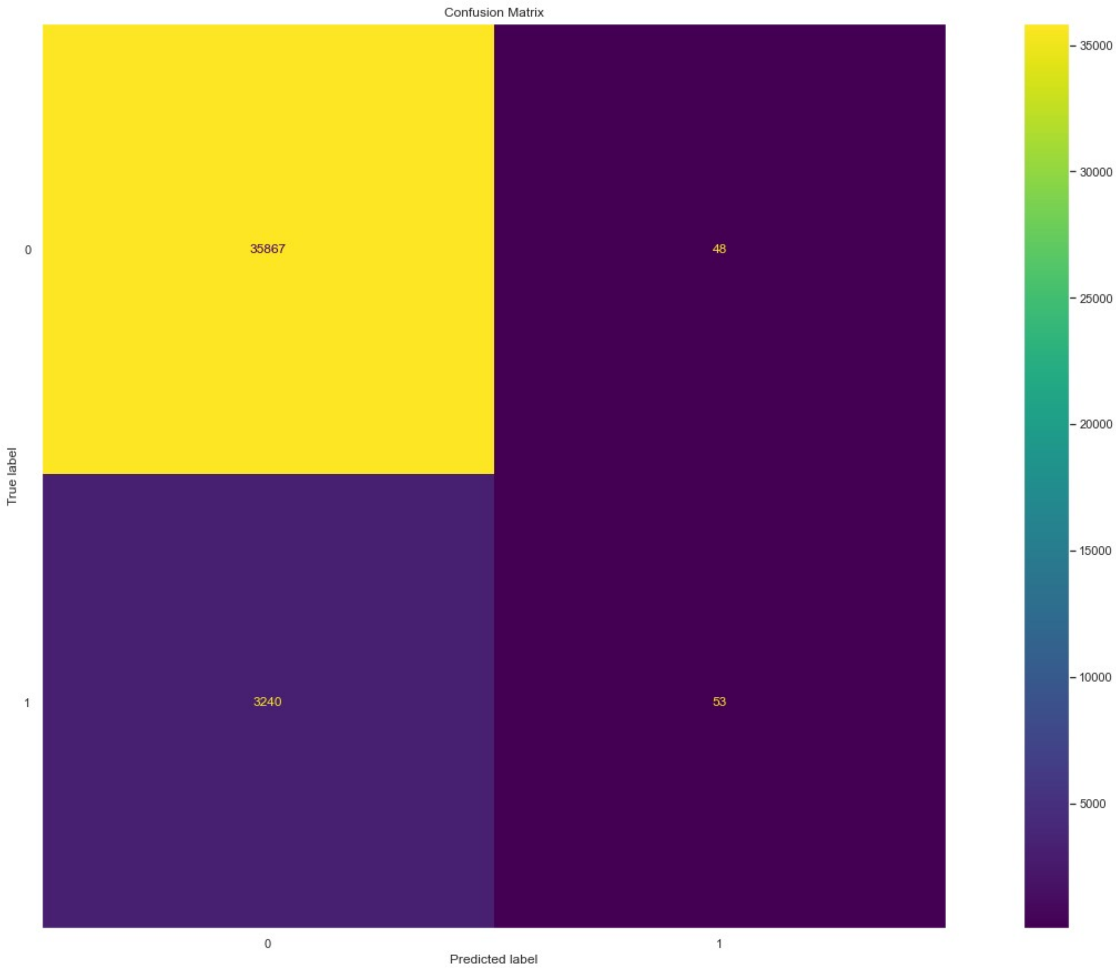
```
import matplotlib.pyplot as plt
from sklearn.metrics import RocCurveDisplay
RocCurveDisplay.from_predictions(y_test, model.predict_proba(X_test)
[:, 1])
plt.show()
```



Confusion matrix for testing data

```
from sklearn.metrics import plot_confusion_matrix
plt.clf()
plot_confusion_matrix(model, X_test, y_test)
plt.title('Confusion Matrix')
plt.show()
```

```
<Figure size 1800x1080 with 0 Axes>
```

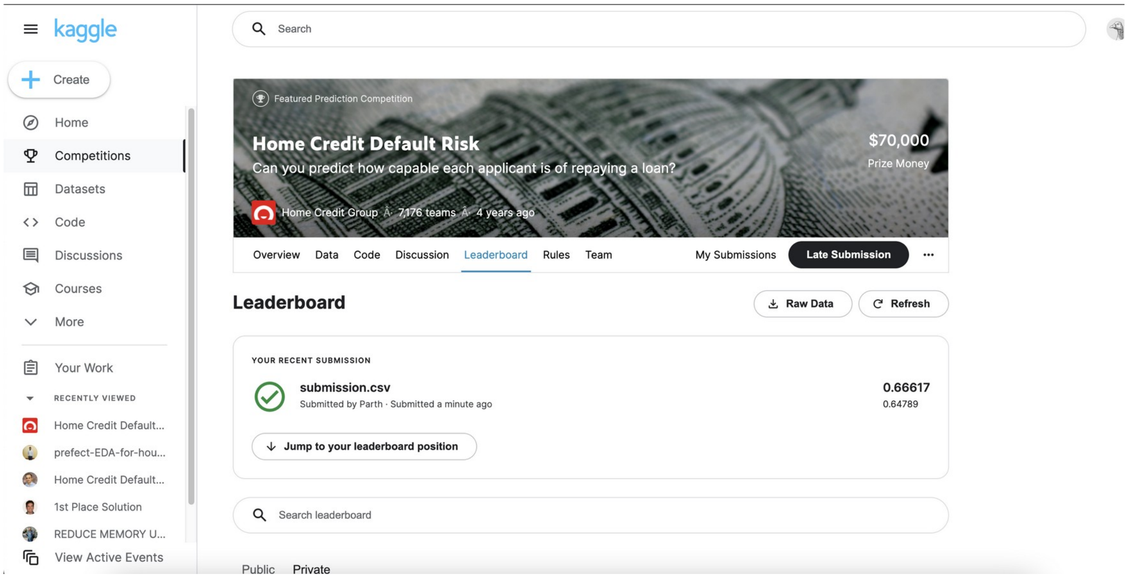


Result Analysis

As we can see from the result log above logistic regression performed better than Naive bayes. So we'll go ahead and submit the logistic regression model as our baseline model on Kaggle.

Kaggle Submission

```
# test_class_scores = model.predict_proba(X_kaggle_test)[:, 1]

# # Submission dataframe
# submit_df = datasets["application_test"][['SK_ID_CURR']]
# submit_df['TARGET'] = test_class_scores

# submit_df.head()
# submit_df.to_csv("submission.csv",index=False)
```



Phase 2: Feature Engineering

```
import pandas as pd
import seaborn as sns
import plotly.express as px
import numpy as np
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import gc
sns.set_style("whitegrid");
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
import pickle as pkl
import tqdm as tqdm
from random import choices
from sklearn.impute import SimpleImputer

from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import log_loss
from sklearn.metrics import roc_auc_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from tqdm import tqdm
# from bayes_opt import BayesianOptimization
from lightgbm import LGBMClassifier
from sklearn.svm import SVC
```

```
from sklearn import metrics
from sklearn.model_selection import KFold,StratifiedKFold
from sklearn.linear_model import Ridge

import warnings
warnings.filterwarnings('ignore')

def df_OHE(df):
    all_col = list(df.columns)
    #getting catagorical features
    cat_col=[x for x in df.columns if df[x].dtype == 'object']
    #using pandas get dummies on categorical features
    df=pd.get_dummies(df,columns=cat_col,dummy_na= False)

    #gewtting new columns
    new_col=list(set(df.columns).difference(set(all_col)))
    return df,new_col,df.columns
```

Feature Engineering on Application Train and Test

```
def train_test_feature_engineering():

    train=pd.read_csv('./Data/application_train.csv')
    test=pd.read_csv('./Data/application_test.csv')


    #merging both dataframes so that we can perform Feature
engineering on it.
    # At the end we will seperate both datas on the basis if Target
variable
    full_data=train.append(test).reset_index()

    del train
    del test
    gc.collect()


    #The XNA value doesn't mean any thing so it is removed from train
data
    full_data=full_data[full_data['CODE_GENDER']!='XNA']

    #we remove this because 365243 is an outlier
    full_data["DAYS_EMPLOYED"].replace({365243: np.nan}, inplace =
True)

    #meadm median, min, max for EXT features
    full_data['EXT_SOURCE_MEAN']=(full_data[['EXT_SOURCE_1',
'EXT_SOURCE_2',
       'EXT_SOURCE_3']]).mean(axis=1)
```

```
    full_data['EXT_SOURCE_MEDIAN']=(full_data[['EXT_SOURCE_1',
'EXT_SOURCE_2',
       'EXT_SOURCE_3']]).median(axis=1)

    full_data['EXT_SOURCE_MIN']=(full_data[['EXT_SOURCE_1',
'EXT_SOURCE_2',
       'EXT_SOURCE_3']]).min(axis=1)

    full_data['EXT_SOURCE_MAX']=(full_data[['EXT_SOURCE_1',
'EXT_SOURCE_2',
       'EXT_SOURCE_3']]).max(axis=1)

    #There is an outlier in the train data where AMT_INCOME_TOTAL of a
person having highest income had difficulty in paying loan.
    full_data=full_data[full_data['AMT_INCOME_TOTAL']<(0.2*1e8)]

    #merging credit bureu hour, day, week, month, quater and year

full_data['AMT_REQ_CREDIT_BUREAU_DATE_DATA']=(full_data[['AMT_REQ_CRED
IT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY',
       'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_MON',
'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_YEAR']]).sum(axis=1)

    # income left after paying annual cost on credit
    full_data['INCOME_LEFT'] = full_data['AMT_INCOME_TOTAL']-
full_data['AMT_ANNUITY']


    #feature engineerin gon the some usefull features
    full_data['DEFAULT_MEAN']=((full_data[['OBS_30_CNT_SOCIAL_CIRCLE',
'DEF_30_CNT_SOCIAL_CIRCLE',
       'OBS_60_CNT_SOCIAL_CIRCLE',
'DEF_60_CNT_SOCIAL_CIRCLE']]).sum(axis=1))//4

    #number of enquiries per customer

full_data['MEAN_ENQUIRIES']=((full_data[['AMT_REQ_CREDIT_BUREAU_HOUR',
'AMT_REQ_CREDIT_BUREAU_DAY',
       'AMT_REQ_CREDIT_BUREAU_WEEK',
'AMT_REQ_CREDIT_BUREAU_MON','AMT_REQ_CREDIT_BUREAU_QRT','AMT_REQ_CREDI
T_BUREAU_YEAR']]).mean(axis=1))

    #how much contact info provided
    full_data['CONTACT_INFO']=((full_data[[
       'FLAG_WORK_PHONE', 'FLAG_MOBIL',
'FLAG_EMP_PHONE','FLAG_CONT_MOBILE','FLAG_PHONE','FLAG_EMAIL']]).sum(a
xis=1))

    #numbber of dayes employed
```

```
    full_data['WORKING_DAYS_NUM']=full_data['DAYS_EMPLOYED'] / 
full_data['DAYS_BIRTH']

    #how much was the goods price compared to the income of that 
customer

full_data['PRICE_PER_INCOME']=full_data['AMT_INCOME_TOTAL']/full_data[
'AMT_GOODS_PRICE']

     #Income divided by family members
    full_data['INCOME_PER_PERSON'] = full_data['AMT_INCOME_TOTAL'] / 
(full_data['CNT_FAM_MEMBERS']+1)

    #Amount paid anually based on credit taken
    full_data['PAYMENT_RATE'] = full_data['AMT_ANNUITY'] / 
full_data['AMT_CREDIT']

    #how much was the goods price compared to the credit amount

full_data['PRICE_PER_CREDIT']=full_data['AMT_CREDIT']/full_data['AMT_G
OODS_PRICE']

    #Income versus the credit amount
    full_data['INCOME_PER_CREDIT'] = full_data['AMT_INCOME_TOTAL'] / 
full_data['AMT_CREDIT']

    # cr4eating new feature with number of colyumns
    full_data['DOC_NUM']=(full_data[[
                                     'FLAG_DOCUMENT_13', 
'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15',
                                     'FLAG_DOCUMENT_16', 
'FLAG_DOCUMENT_17', 'FLAG_DOCUMENT_18',
       'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6',
       'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12',
       'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8', 'FLAG_DOCUMENT_9',
        'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21',
        'FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3'


      ]]==1).sum(axis=1)


    full_data,cat_col,all_col=df_OHE(full_data)
    with open("train_test_FE_data.pkl", "wb") as f:
        pkl.dump(all_col, f)

    return full_data
```

Feature Engineering on Bureau and Bureau Blance datasets

```
def fe_of_bureau_balance():

    bureau_balance = pd.read_csv('./Data/bureau_balance.csv')

bureau_balance,bureau_balance_data_cat_columns,all_columns=df_OHE(bure
au_balance)
    with open("all_columns_bureau_data_balance.pkl", "wb") as f:
        pkl.dump(all_columns, f)

    #Applying aggregate function on numerical column
    bureau_balance_agg = {'MONTHS_BALANCE': ['min', 'max','sum']}

    #Applying Aggregate function on cat column
    for column in bureau_balance_data_cat_columns:
        if (column!='SK_BUREAU_ID'):
            bureau_balance_agg[column] = ['mean']

    bureau_balance_agg =
bureau_balance.groupby(['SK_ID_BUREAU']).agg(bureau_balance_agg)


    month = -12
    bureau_balance_temp = bureau_balance[bureau_balance.MONTHS_BALANCE
>= month].copy()
    bureau_balance_agg['STATUS_C_12'] =
bureau_balance_temp.groupby('SK_ID_BUREAU')['STATUS_C'].sum()

    month = -24
    bureau_balance_temp = bureau_balance[bureau_balance.MONTHS_BALANCE
>= month].copy()
    bureau_balance_agg['STATUS_C_24'] =
bureau_balance_temp.groupby('SK_ID_BUREAU')['STATUS_C'].sum()

    month = -36
    bureau_balance_temp = bureau_balance[bureau_balance.MONTHS_BALANCE
>= month].copy()
    bureau_balance_agg['STATUS_C_36'] =
bureau_balance_temp.groupby('SK_ID_BUREAU')['STATUS_C'].sum()

    month = -48
    bureau_balance_temp = bureau_balance[bureau_balance.MONTHS_BALANCE
```

```
>= month].copy()
    bureau_balance_agg['STATUS_C_48'] =
bureau_balance_temp.groupby('SK_ID_BUREAU')['STATUS_C'].sum()

    month = -60
    bureau_balance_temp = bureau_balance[bureau_balance.MONTHS_BALANCE
>= month].copy()
    bureau_balance_agg['STATUS_C_60'] =
bureau_balance_temp.groupby('SK_ID_BUREAU')['STATUS_C'].sum()

    del bureau_balance_temp

    gc.collect()



    modified_col=[]
    for column in list(bureau_balance_agg.columns):
        if (column!='SK_BUREAU_ID'):
            modified_col.append(column[0]+"_"+column[1].upper())
    bureau_balance_agg.columns=modified_col


    bureau_data = pd.read_csv('./Data/bureau.csv')


bureau_data,bureau_data_cat_columns,all_columns=df_OHE(bureau_data)
    with open("all_columns_bureau_data.pkl", "wb") as f:
        pkl.dump(all_columns, f)

    bureau_data['SEC_LOAN_COUNT']=(bureau_data[['CREDIT_TYPE_Car
loan','CREDIT_TYPE_Loan for the purchase of
equipment','CREDIT_TYPE_Mortgage','CREDIT_TYPE_Real estate
loan','CREDIT_TYPE_Loan for purchase of shares (margin lending)'
                                                ]]==1).sum(axis=1)

    bureau_data['UNSEC_LOAN_COUNT']=(bureau_data[[
'CREDIT_TYPE_Another type of loan',
                                                   'CREDIT_TYPE_Cash
loan (non-earmarked)', 'CREDIT_TYPE_Consumer credit',
                                                   'CREDIT_TYPE_Credit
card', 'CREDIT_TYPE_Interbank credit',
                                                   'CREDIT_TYPE_Loan
for business development',
                                                   'CREDIT_TYPE_Loan
for working capital replenishment',

'CREDIT_TYPE_Microloan', 'CREDIT_TYPE_Mobile operator loan',
```

```
'CREDIT_TYPE_Unknown type of loan']]==1).sum(axis=1)



bureau_data['DEBT_PER']=bureau_data['AMT_CREDIT_SUM_DEBT']/bureau_data
['AMT_CREDIT_SUM']


bureau_data['AMT_ANNUITY_AMT_CREDIT_SUM_PER']=bureau_data['AMT_ANNUITY
']/bureau_data['AMT_CREDIT_SUM']


bureau_data['DEBT_LIMIT_PER']=bureau_data['AMT_CREDIT_SUM_DEBT']/burea
u_data['AMT_CREDIT_SUM_LIMIT']

    bureau_data['B_EXTRA_PAY'] = bureau_data['AMT_ANNUITY']-
bureau_data['AMT_CREDIT_SUM']

    for column in bureau_data.columns:
        if column.startswith('DAYS'):
            bureau_data[column].replace(365243, np.nan, inplace= True)


    bureau_data = bureau_data.join(bureau_balance_agg, how='left',
on=['SK_ID_BUREAU'])



    bureau_data_agg={}
    for column in bureau_data.columns:
        if (column!='SK_ID_CURR' or column!='SK_BUREAU_ID'):
            bureau_data_agg[column]=['mean']
            if (column=='AMT_CREDIT_SUM_OVERDUE') |
(column=='SEC_LOAN_COUNT') | (column=='UNSEC_LOAN_COUNT') |
(column=='AMT_CREDIT_SUM_DEBT'):
                bureau_data_agg[column]=['sum']
            elif column=='DAYS_CREDIT':
                bureau_data_agg[column]=['min']
            elif column=='DEBT_PER':
                bureau_data_agg[column]=['mean']





    bureau_agg =
bureau_data.groupby('SK_ID_CURR').agg(bureau_data_agg)
```

```
    modified_col=[]
    for column in list(bureau_agg.columns):
        modified_col.append(column[0]+"_"+column[1].upper())
    bureau_agg.columns=modified_col

    bureau_agg['AMT_CREDIT_MAX_OVERDUE_MAX'] =
bureau_data.groupby('SK_ID_CURR')['AMT_CREDIT_MAX_OVERDUE'].max()

    bureau_agg['BEAU_COUNT'] = bureau_data.groupby('SK_ID_CURR')
['SK_ID_BUREAU'].count()


bureau_agg['ABS_YEAR_CREDIT_MAX']=abs(bureau_agg['DAYS_CREDIT_MIN']/
365)




    bureau_data_active =
bureau_data[bureau_data['CREDIT_ACTIVE_Active'] == 1]

    bureau_agg['BUREAU_ACT_AMT_CREDIT_SUME_MIN'] =
bureau_data_active.groupby('SK_ID_CURR')['AMT_CREDIT_SUM'].min()
    bureau_agg['BUREAU_ACT_AMT_CREDIT_SUM_MAX'] =
bureau_data_active.groupby('SK_ID_CURR')['AMT_CREDIT_SUM'].max()
    bureau_agg['BUREAU_ACT_AMT_CREDIT_SUM_MEAN'] =
bureau_data_active.groupby('SK_ID_CURR')['AMT_CREDIT_SUM'].mean()

    bureau_agg['BUREAU_ACT_AMT_CREDIT_SUM_OVERDUE_MIN'] =
bureau_data_active.groupby('SK_ID_CURR')
['AMT_CREDIT_SUM_OVERDUE'].min()
    bureau_agg['BUREAU_ACT_AMT_CREDIT_SUM_OVERDUE_MAX'] =
bureau_data_active.groupby('SK_ID_CURR')
['AMT_CREDIT_SUM_OVERDUE'].max()
    bureau_agg['BUREAU_ACT_AMT_CREDIT_SUM_OVERDUE_MEAN'] =
bureau_data_active.groupby('SK_ID_CURR')
['AMT_CREDIT_SUM_OVERDUE'].mean()

    bureau_agg['BUREAU_ACT_AMT_CREDIT_MAX_OVERDUE_MIN'] =
bureau_data_active.groupby('SK_ID_CURR')
['AMT_CREDIT_MAX_OVERDUE'].min()
    bureau_agg['BUREAU_ACT_AMT_CREDIT_MAX_OVERDUE_MAX'] =
bureau_data_active.groupby('SK_ID_CURR')
['AMT_CREDIT_MAX_OVERDUE'].max()
    bureau_agg['BUREAU_ACT_AMT_CREDIT_MAX_OVERDUE_MEAN'] =
bureau_data_active.groupby('SK_ID_CURR')
['AMT_CREDIT_MAX_OVERDUE'].mean()

    bureau_agg['BUREAU_ACT_AMT_CREDIT_SUM_LIMIT_MIN'] =
```

```
bureau_data_active.groupby('SK_ID_CURR')['AMT_CREDIT_SUM_LIMIT'].min()
    bureau_agg['BUREAU_ACT_AMT_CREDIT_SUM_LIMIT_MAX'] =
bureau_data_active.groupby('SK_ID_CURR')['AMT_CREDIT_SUM_LIMIT'].max()
    bureau_agg['BUREAU_ACT_AMT_CREDIT_SUM_LIMIT_MEAN'] =
bureau_data_active.groupby('SK_ID_CURR')
['AMT_CREDIT_SUM_LIMIT'].mean()


    bureau_agg['BUREAU_ACT_AMT_CREDIT_SUM_DEBT_MIN'] =
bureau_data_active.groupby('SK_ID_CURR')['AMT_CREDIT_SUM_DEBT'].min()
    bureau_agg['BUREAU_ACT_AMT_CREDIT_SUM_DEBT_MAX'] =
bureau_data_active.groupby('SK_ID_CURR')['AMT_CREDIT_SUM_DEBT'].max()
    bureau_agg['BUREAU_ACT_AMT_CREDIT_SUM_DEBT_MEAN'] =
bureau_data_active.groupby('SK_ID_CURR')['AMT_CREDIT_SUM_DEBT'].mean()



    bureau_agg['BUREAU_ACT_DAYS_CREDIT_ENDDATE_MIN'] =
bureau_data_active.groupby('SK_ID_CURR')['DAYS_CREDIT_ENDDATE'].min()
    bureau_agg['BUREAU_ACT_DAYS_CREDIT_ENDDATE_MAX'] =
bureau_data_active.groupby('SK_ID_CURR')['DAYS_CREDIT_ENDDATE'].max()
    bureau_agg['BUREAU_ACT_DAYS_CREDIT_ENDDATE_MEAN'] =
bureau_data_active.groupby('SK_ID_CURR')['DAYS_CREDIT_ENDDATE'].mean()


    bureau_agg['BUREAU_ACT_DAYS_ENDDATE_FACT_MIN'] =
bureau_data_active.groupby('SK_ID_CURR')['DAYS_ENDDATE_FACT'].min()
    bureau_agg['BUREAU_ACT_DAYS_ENDDATE_FACT_MAX'] =
bureau_data_active.groupby('SK_ID_CURR')['DAYS_ENDDATE_FACT'].max()
    bureau_agg['BUREAU_ACT_DAYS_ENDDATE_FACT_MEAN'] =
bureau_data_active.groupby('SK_ID_CURR')['DAYS_ENDDATE_FACT'].mean()


    bureau_agg['BUREAU_ACT_CREDIT_DAY_OVERDUE_MIN'] =
bureau_data_active.groupby('SK_ID_CURR')['CREDIT_DAY_OVERDUE'].min()
    bureau_agg['BUREAU_ACT_CREDIT_DAY_OVERDUE_MAX'] =
bureau_data_active.groupby('SK_ID_CURR')['CREDIT_DAY_OVERDUE'].max()
    bureau_agg['BUREAU_ACT_CREDIT_DAY_OVERDUE_MEAN'] =
bureau_data_active.groupby('SK_ID_CURR')['CREDIT_DAY_OVERDUE'].mean()


    bureau_agg['BUREAU_ACT_DAYS_CREDIT_MIN'] =
bureau_data_active.groupby('SK_ID_CURR')['DAYS_CREDIT'].min()
    bureau_agg['BUREAU_ACT_DAYS_CREDIT_MAX'] =
bureau_data_active.groupby('SK_ID_CURR')['DAYS_CREDIT'].max()
    bureau_agg['BUREAU_ACT_DAYS_CREDIT_MEAN'] =
bureau_data_active.groupby('SK_ID_CURR')['DAYS_CREDIT'].mean()
```

```
    bureau_data_closed =
bureau_data[bureau_data['CREDIT_ACTIVE_Closed'] == 1]

    bureau_agg['BUREAU_CLO_DAYS_CREDIT_ENDDATE_MIN'] =
bureau_data_closed.groupby('SK_ID_CURR')['DAYS_CREDIT_ENDDATE'].min()
    bureau_agg['BUREAU_CLO_DAYS_CREDIT_ENDDATE_MAX'] =
bureau_data_closed.groupby('SK_ID_CURR')['DAYS_CREDIT_ENDDATE'].max()
    bureau_agg['BUREAU_CLO_DAYS_CREDIT_ENDDATE_MEAN'] =
bureau_data_closed.groupby('SK_ID_CURR')['DAYS_CREDIT_ENDDATE'].mean()

    bureau_agg['BUREAU_CLO_CREDIT_DAY_OVERDUE_MIN'] =
bureau_data_closed.groupby('SK_ID_CURR')['CREDIT_DAY_OVERDUE'].min()
    bureau_agg['BUREAU_CLO_CREDIT_DAY_OVERDUE_MAX'] =
bureau_data_closed.groupby('SK_ID_CURR')['CREDIT_DAY_OVERDUE'].max()
    bureau_agg['BUREAU_CLO_CREDIT_DAY_OVERDUE_MEAN'] =
bureau_data_closed.groupby('SK_ID_CURR')['CREDIT_DAY_OVERDUE'].mean()


    bureau_agg['BUREAU_CLO_DAYS_CREDIT_MIN'] =
bureau_data_closed.groupby('SK_ID_CURR')['DAYS_CREDIT'].min()
    bureau_agg['BUREAU_CLO_DAYS_CREDIT_MAX'] =
bureau_data_closed.groupby('SK_ID_CURR')['DAYS_CREDIT'].max()
    bureau_agg['BUREAU_CLO_DAYS_CREDIT_MEAN'] =
bureau_data_closed.groupby('SK_ID_CURR')['DAYS_CREDIT'].mean()


    bureau_agg['BUREAU_CLO_DAYS_ENDDATE_FACT_MIN'] =
bureau_data_closed.groupby('SK_ID_CURR')['DAYS_ENDDATE_FACT'].min()
    bureau_agg['BUREAU_CLO_DAYS_ENDDATE_FACT_MAX'] =
bureau_data_closed.groupby('SK_ID_CURR')['DAYS_ENDDATE_FACT'].max()
    bureau_agg['BUREAU_CLO_DAYS_ENDDATE_FACT_MEAN'] =
bureau_data_closed.groupby('SK_ID_CURR')['DAYS_ENDDATE_FACT'].mean()

    bureau_agg['BUREAU_CLO_AMT_CREDIT_SUM_MIN'] =
bureau_data_closed.groupby('SK_ID_CURR')['AMT_CREDIT_SUM'].min()
    bureau_agg['B_CLO_AMT_CREDIT_SUM_MAX'] =
bureau_data_closed.groupby('SK_ID_CURR')['AMT_CREDIT_SUM'].max()
    bureau_agg['BUREAU_CLO_AMT_CREDIT_SUM_MEAN'] =
bureau_data_closed.groupby('SK_ID_CURR')['AMT_CREDIT_SUM'].mean()

    bureau_agg['BUREAU_CLO_AMT_CREDIT_MAX_OVERDUE_MIN'] =
bureau_data_closed.groupby('SK_ID_CURR')
['AMT_CREDIT_MAX_OVERDUE'].min()
    bureau_agg['BUREAU_CLO_AMT_CREDIT_MAX_OVERDUE_MAX'] =
bureau_data_closed.groupby('SK_ID_CURR')
['AMT_CREDIT_MAX_OVERDUE'].max()
    bureau_agg['BUREAU_CLO_AMT_CREDIT_MAX_OVERDUE_MEAN'] =
```

```
bureau_data_closed.groupby('SK_ID_CURR')
['AMT_CREDIT_MAX_OVERDUE'].mean()

    bureau_agg['BUREAU_CLO_AMT_CREDIT_SUM_DEBT_MIN'] =
bureau_data_closed.groupby('SK_ID_CURR')['AMT_CREDIT_SUM_DEBT'].min()
    bureau_agg['BUREAU_CLO_AMT_CREDIT_SUM_DEBT_MAX'] =
bureau_data_closed.groupby('SK_ID_CURR')['AMT_CREDIT_SUM_DEBT'].max()
    bureau_agg['BUREAU_CLO_AMT_CREDIT_SUM_DEBT_MEAN'] =
bureau_data_closed.groupby('SK_ID_CURR')['AMT_CREDIT_SUM_DEBT'].mean()

    bureau_agg['BUREAU_CLO_AMT_CREDIT_SUM_LIMIT_MIN'] =
bureau_data_closed.groupby('SK_ID_CURR')['AMT_CREDIT_SUM_LIMIT'].min()
    bureau_agg['BUREAU_CLO_AMT_CREDIT_SUM_LIMIT_MAX'] =
bureau_data_closed.groupby('SK_ID_CURR')['AMT_CREDIT_SUM_LIMIT'].max()
    bureau_agg['BUREAU_CLO_AMT_CREDIT_SUM_LIMIT_MEAN'] =
bureau_data_closed.groupby('SK_ID_CURR')
['AMT_CREDIT_SUM_LIMIT'].mean()

    bureau_agg['BUREAU_CLO_AMT_CREDIT_SUM_OVERDUE_MIN'] =
bureau_data_closed.groupby('SK_ID_CURR')
['AMT_CREDIT_SUM_OVERDUE'].min()
    bureau_agg['BUREAU_CLO_AMT_CREDIT_SUM_OVERDUE_MAX'] =
bureau_data_closed.groupby('SK_ID_CURR')
['AMT_CREDIT_SUM_OVERDUE'].max()
    bureau_agg['BUREAU_CLO_AMT_CREDIT_SUM_OVERDUE_MEAN'] =
bureau_data_closed.groupby('SK_ID_CURR')
['AMT_CREDIT_SUM_OVERDUE'].mean()

    bureau_agg.drop(['SK_ID_CURR_MEAN'], axis=1, inplace= True)
    bureau_agg.drop(['SK_ID_BUREAU_MEAN'], axis=1, inplace= True)


    del bureau_data,bureau_data_closed,bureau_data_active
    del bureau_balance

    gc.collect()

    return bureau_agg
```

Feature Engineering on Previous application

```
def fe_previous_application():

    prev_data = pd.read_csv('./DATA/previous_application.csv')

    prev_data['EXTRA_AMT_PAID'] =
prev_data['CNT_PAYMENT']*prev_data['AMT_ANNUITY']-
```

```
prev_data['AMT_CREDIT']

    prev_data['AMT_LEFT_TO_PAY'] =
prev_data['CNT_PAYMENT']*prev_data['AMT_ANNUITY']-
prev_data['AMT_DOWN_PAYMENT']


    #https://www.calculatorsoup.com/calculators/financial/simple-
interest-plus-principal-calculator.php
    prev_data['RATE_OF_INTREST'] =
(1/prev_data['CNT_PAYMENT'])*(((prev_data['CNT_PAYMENT']*prev_data['AM
T_ANNUITY'])/prev_data['AMT_CREDIT'])-1)

    prev_data['SIMPLE_INTREST']=
(prev_data['AMT_CREDIT']*prev_data['RATE_OF_INTREST']*prev_data['CNT_P
AYMENT'])/100


prev_data['PREV_APP_XAP']=((prev_data['CODE_REJECT_REASON']=='XAP')).a
stype(int)



prev_data['AMT_DOWN_PAYMENT_L_40']=(prev_data['AMT_DOWN_PAYMENT']<=(0.
40*prev_data['AMT_CREDIT'])).astype(int)

    prev_data,prev_data_cat_columns,all_columns=df_OHE(prev_data)
    with open("all_columns_prev_data.pkl", "wb") as f:
        pkl.dump(all_columns, f)

    for col in prev_data.columns:
        if col.startswith('DAYS'):
            prev_data[col].replace(365243, np.nan, inplace= True)


    prev_data_agg={}
    for col in prev_data.columns:
        if col!='SK_ID_CURR' and col !='SK_ID_PREV':
            prev_data_agg[col]=['mean']
        if (col=='DAYS_TERMINATION') | (col=='DAYS_FIRST_DUE') |
(col=='DAYS_LAST_DUE') | (col=='AMT_CREDIT') | (col=='AMT_ANNUITY') |
(col=='AMT_DOWN_PAYMENT') | (col=='DAYS_LAST_DUE_1ST_VERSION') |
(col=='HOUR_APPR_PROCESS_START') :
            prev_data_agg[col]=['min','max','mean']

    prev_agg = prev_data.groupby('SK_ID_CURR').agg(prev_data_agg)

    modified_col=[]
    for c in list(prev_agg.columns):
        modified_col.append("PREV_"+c[0]+"_"+c[1].upper())
```

```
    prev_agg.columns=modified_col

    canceled_refused =
prev_data[(prev_data['NAME_CONTRACT_STATUS_Refused'] == 1) |
(prev_data['NAME_CONTRACT_STATUS_Canceled'] == 1)]
    prev_agg['PREVCR_AMT_CREDIT_MEAN'] =
canceled_refused.groupby('SK_ID_CURR')['AMT_CREDIT'].mean()
    prev_agg['PREVCR_AMT_CREDIT_MIN'] =
canceled_refused.groupby('SK_ID_CURR')['AMT_CREDIT'].min()
    prev_agg['PREVCR_AMT_CREDIT_MAX'] =
canceled_refused.groupby('SK_ID_CURR')['AMT_CREDIT'].max()

    prev_agg['PREVCR_AMT_ANNUITY_MEAN'] =
canceled_refused.groupby('SK_ID_CURR')['AMT_ANNUITY'].mean()
    prev_agg['PREVCR_AMT_ANNUITY_MIN'] =
canceled_refused.groupby('SK_ID_CURR')['AMT_ANNUITY'].min()
    prev_agg['PREVCR_AMT_ANNUITY_MAX'] =
canceled_refused.groupby('SK_ID_CURR')['AMT_ANNUITY'].max()


    prev_agg['PREVCR_AMT_DOWN_PAYMENT_MEAN'] =
canceled_refused.groupby('SK_ID_CURR')['AMT_DOWN_PAYMENT'].mean()
    prev_agg['PREVCR_AMT_DOWN_PAYMENT_MIN'] =
canceled_refused.groupby('SK_ID_CURR')['AMT_DOWN_PAYMENT'].min()
    prev_agg['PREVCR_AMT_DOWN_PAYMENT_MAX'] =
canceled_refused.groupby('SK_ID_CURR')['AMT_DOWN_PAYMENT'].max()

    prev_agg['PREVCR_AMT_LEFT_TO_PAY_MEAN'] =
canceled_refused.groupby('SK_ID_CURR')['AMT_LEFT_TO_PAY'].mean()
    prev_agg['PREVCR_AMT_LEFT_TO_PAY_MIN'] =
canceled_refused.groupby('SK_ID_CURR')['AMT_LEFT_TO_PAY'].min()
    prev_agg['PREVCR_AMT_LEFT_TO_PAY_MAX'] =
canceled_refused.groupby('SK_ID_CURR')['AMT_LEFT_TO_PAY'].max()


    prev_agg['PREVCR_RATE_OF_INTREST_MEAN'] =
canceled_refused.groupby('SK_ID_CURR')['RATE_OF_INTREST'].mean()
    prev_agg['PREVCR_RATE_OF_INTREST_MIN'] =
canceled_refused.groupby('SK_ID_CURR')['RATE_OF_INTREST'].min()
    prev_agg['PREVCR_RATE_OF_INTREST_MAX'] =
canceled_refused.groupby('SK_ID_CURR')['RATE_OF_INTREST'].max()

    approved = prev_data[(prev_data['NAME_CONTRACT_STATUS_Approved']
== 1)]

    prev_agg['PREVA_AMT_CREDIT_MEAN'] = approved.groupby('SK_ID_CURR')
['AMT_CREDIT'].mean()
    prev_agg['PREVA_AMT_CREDIT_MIN'] = approved.groupby('SK_ID_CURR')
['AMT_CREDIT'].min()
```

```
    prev_agg['PREVA_AMT_CREDIT_MAX'] = approved.groupby('SK_ID_CURR')
['AMT_CREDIT'].max()

    prev_agg['PREVA_AMT_ANNUITY_MEAN'] =
approved.groupby('SK_ID_CURR')['AMT_ANNUITY'].mean()
    prev_agg['PREVA_AMT_ANNUITY_MIN'] = approved.groupby('SK_ID_CURR')
['AMT_ANNUITY'].min()
    prev_agg['PREVA_AMT_ANNUITY_MAX'] = approved.groupby('SK_ID_CURR')
['AMT_ANNUITY'].max()

    prev_agg['PREVA_AMT_DOWN_PAYMENT_MEAN'] =
approved.groupby('SK_ID_CURR')['AMT_DOWN_PAYMENT'].mean()
    prev_agg['PREVA_AMT_DOWN_PAYMENT_MIN'] =
approved.groupby('SK_ID_CURR')['AMT_DOWN_PAYMENT'].min()
    prev_agg['PREVA_AMT_DOWN_PAYMENT_MAX'] =
approved.groupby('SK_ID_CURR')['AMT_DOWN_PAYMENT'].max()

    prev_agg['PREVA_AMT_LEFT_TO_PAY_MEAN'] =
approved.groupby('SK_ID_CURR')['AMT_LEFT_TO_PAY'].mean()
    prev_agg['PREVA_AMT_LEFT_TO_PAY_MIN'] =
approved.groupby('SK_ID_CURR')['AMT_LEFT_TO_PAY'].min()
    prev_agg['PREVA_AMT_LEFT_TO_PAY_MAX'] =
approved.groupby('SK_ID_CURR')['AMT_LEFT_TO_PAY'].max()

    prev_agg['PREVA_RATE_OF_INTREST_MEAN'] =
approved.groupby('SK_ID_CURR')['RATE_OF_INTREST'].mean()
    prev_agg['PREVA_RATE_OF_INTREST_MIN'] =
approved.groupby('SK_ID_CURR')['RATE_OF_INTREST'].min()
    prev_agg['PREVA_RATE_OF_INTREST_MAX'] =
approved.groupby('SK_ID_CURR')['RATE_OF_INTREST'].max()

    del prev_data,approved,canceled_refused

    gc.collect()


    return prev_agg
```

Feature Engineering on POS application

```
def fe_pos_application():

    pos_data = pd.read_csv('./DATA/POS_CASH_balance.csv')

    pos_data=pos_data[pos_data['NAME_CONTRACT_STATUS']!='XNA']

    pos_data,pos_data_cat_columns,all_columns=df_OHE(pos_data)
    with open("all_columns_pos_data.pkl", "wb") as f:
        pkl.dump(all_columns, f)
```

```
    pos_data_agg={}
    for col in pos_data.columns:
        if col!='SK_ID_CURR' and col !='SK_ID_PREV':
            pos_data_agg[col]=['mean']
        if col=='MONTHS_BALANCE':
            pos_data_agg[col]=['sum','mean','max','min']


    pos_agg = pos_data.groupby('SK_ID_CURR').agg(pos_data_agg)

    modified_col=[]
    for c in list(pos_agg.columns):
        modified_col.append("POS_"+c[0]+"_"+c[1].upper())
    pos_agg.columns=modified_col

    pos_agg['COUNT_OF_POS'] = pos_data.groupby('SK_ID_CURR')
['SK_ID_PREV'].count()

    month = -24
    pos_temp = pos_data[pos_data.MONTHS_BALANCE >= month].copy()
    pos_agg['24_MON_CNT_INSTALMENT_FUTURE_MEAN'] =
pos_temp.groupby('SK_ID_CURR')['CNT_INSTALMENT_FUTURE'].mean()
    pos_agg['24_MON_CNT_INSTALMENT_FUTURE_MIN'] =
pos_temp.groupby('SK_ID_CURR')['CNT_INSTALMENT_FUTURE'].min()
    pos_agg['24_MON_CNT_INSTALMENT_FUTURE_MAX'] =
pos_temp.groupby('SK_ID_CURR')['CNT_INSTALMENT_FUTURE'].max()

    month = -12
    pos_temp = pos_data[pos_data.MONTHS_BALANCE >= month].copy()
    pos_agg['12_MON_CNT_INSTALMENT_FUTURE_MEAN'] =
pos_temp.groupby('SK_ID_CURR')['CNT_INSTALMENT_FUTURE'].mean()
    pos_agg['12_MON_CNT_INSTALMENT_FUTURE_MIN'] =
pos_temp.groupby('SK_ID_CURR')['CNT_INSTALMENT_FUTURE'].min()
    pos_agg['12_MON_CNT_INSTALMENT_FUTURE_MAX'] =
pos_temp.groupby('SK_ID_CURR')
['CNT_INSTALMENT_FUTURE'].max()#######################################
3

    month = -24
    pos_temp = pos_data[pos_data.MONTHS_BALANCE >= month].copy()
    pos_agg['24_SK_DPD_MEAN'] = pos_temp.groupby('SK_ID_CURR')
['SK_DPD'].mean()
    pos_agg['24_SK_DPD_MIN'] = pos_temp.groupby('SK_ID_CURR')
['SK_DPD'].min()
    pos_agg['24_SK_DPD_MAX'] = pos_temp.groupby('SK_ID_CURR')
['SK_DPD'].max()

    month = -12
```

```
    pos_temp = pos_data[pos_data.MONTHS_BALANCE >= month].copy()
    pos_agg['12_SK_DPD_MEAN'] = pos_temp.groupby('SK_ID_CURR')
['SK_DPD'].mean()
    pos_agg['12_SK_DPD_MIN'] = pos_temp.groupby('SK_ID_CURR')
['SK_DPD'].min()
    pos_agg['12_SK_DPD_MAX'] = pos_temp.groupby('SK_ID_CURR')
['SK_DPD'].max()

    month = -24
    pos_temp = pos_data[pos_data.MONTHS_BALANCE >= month].copy()
    pos_agg['24_SK_DPD_DEF_MEAN'] = pos_temp.groupby('SK_ID_CURR')
['SK_DPD_DEF'].mean()
    pos_agg['24_SK_DPD_DEF_MAX'] = pos_temp.groupby('SK_ID_CURR')
['SK_DPD_DEF'].max()

    month = -12
    pos_temp = pos_data[pos_data.MONTHS_BALANCE >= month].copy()
    pos_agg['12_SK_DPD_DEF_MEAN'] = pos_temp.groupby('SK_ID_CURR')
['SK_DPD_DEF'].mean()
    pos_agg['12_SK_DPD_DEF_MAX'] = pos_temp.groupby('SK_ID_CURR')
['SK_DPD_DEF'].max()

    active = pos_data[pos_data['NAME_CONTRACT_STATUS_Active'] == 1]
    pos_agg['POSACT_CNT_INSTALMENT_FUTURE_MEAN'] =
active.groupby('SK_ID_CURR')['CNT_INSTALMENT_FUTURE'].mean()
    pos_agg['POSACT_CNT_INSTALMENT_FUTURE_MIN'] =
active.groupby('SK_ID_CURR')['CNT_INSTALMENT_FUTURE'].min()
    pos_agg['POSACT_CNT_INSTALMENT_FUTURE_MAX'] =
active.groupby('SK_ID_CURR')['CNT_INSTALMENT_FUTURE'].max()

    pos_agg['POSACT_CNT_INSTALMENT_MEAN'] =
active.groupby('SK_ID_CURR')['CNT_INSTALMENT'].mean()
    pos_agg['POSACT_CNT_INSTALMENT_MIN'] =
active.groupby('SK_ID_CURR')['CNT_INSTALMENT'].min()
    pos_agg['POSACT_CNT_INSTALMENT_MAX'] =
active.groupby('SK_ID_CURR')['CNT_INSTALMENT'].max()

    pos_agg['POSACT_SK_DPD_MEAN'] = active.groupby('SK_ID_CURR')
['SK_DPD'].mean()
    pos_agg['POSACT_SK_DPD_MIN'] = active.groupby('SK_ID_CURR')
['SK_DPD'].min()
    pos_agg['POSACT_SK_DPD_MAX'] = active.groupby('SK_ID_CURR')
['SK_DPD'].max()

    pos_agg['POSACT_SK_DPD_DEF_MEAN'] = active.groupby('SK_ID_CURR')
['SK_DPD_DEF'].mean()
    pos_agg['POSACT_SK_DPD_DEF_MIN'] = active.groupby('SK_ID_CURR')
['SK_DPD_DEF'].min()
    pos_agg['POSACT_SK_DPD_DEF_MAX'] = active.groupby('SK_ID_CURR')
['SK_DPD_DEF'].max()
```

```
    completed = pos_data[pos_data['NAME_CONTRACT_STATUS_Completed'] ==
1]

    pos_agg['POSCOMP_CNT_INSTALMENT_FUTURE_MEAN'] =
completed.groupby('SK_ID_CURR')['CNT_INSTALMENT_FUTURE'].mean()
    pos_agg['POSCOMP_CNT_INSTALMENT_FUTURE_MIN'] =
completed.groupby('SK_ID_CURR')['CNT_INSTALMENT_FUTURE'].min()
    pos_agg['POSCOMP_CNT_INSTALMENT_FUTURE_MAX'] =
completed.groupby('SK_ID_CURR')['CNT_INSTALMENT_FUTURE'].max()

    pos_agg['POSCOMP_CNT_INSTALMENT_MEAN'] =
completed.groupby('SK_ID_CURR')['CNT_INSTALMENT'].mean()
    pos_agg['POSCOMP_CNT_INSTALMENT_MIN'] =
completed.groupby('SK_ID_CURR')['CNT_INSTALMENT'].min()
    pos_agg['POSCOMP_CNT_INSTALMENT_MAX'] =
completed.groupby('SK_ID_CURR')['CNT_INSTALMENT'].max()

    pos_agg['POSCOMP_SK_DPD_MEAN'] = completed.groupby('SK_ID_CURR')
['SK_DPD'].mean()
    pos_agg['POSCOMP_SK_DPD_MIN'] = completed.groupby('SK_ID_CURR')
['SK_DPD'].min()
    pos_agg['POSCOMP_SK_DPD_MAX'] = completed.groupby('SK_ID_CURR')
['SK_DPD'].max()

    pos_agg['POSCOMP_SK_DPD_DEF_MEAN'] =
completed.groupby('SK_ID_CURR')['SK_DPD_DEF'].mean()
    pos_agg['POSCOMP_SK_DPD_DEF_MIN'] =
completed.groupby('SK_ID_CURR')['SK_DPD_DEF'].min()
    pos_agg['POSCOMP_SK_DPD_DEF_MAX'] =
completed.groupby('SK_ID_CURR')['SK_DPD_DEF'].max()


    del pos_data

    gc.collect()

    return pos_agg
```

Feature Engineering credit card balance

```
def fe_credit_card_balance():

    credit_data = pd.read_csv('./DATA/credit_card_balance.csv')


credit_data['FLAG_LESS_30']=(credit_data['AMT_DRAWINGS_CURRENT']<(0.30
*credit_data['AMT_CREDIT_LIMIT_ACTUAL'])).astype(int)
```

```
credit_data['FLAG_LESS_60']=(credit_data['AMT_DRAWINGS_CURRENT']<(0.60
*credit_data['AMT_CREDIT_LIMIT_ACTUAL'])).astype(int)

credit_data['FLAG_LESS_90']=(credit_data['AMT_DRAWINGS_CURRENT']<(0.90
*credit_data['AMT_CREDIT_LIMIT_ACTUAL'])).astype(int)

credit_data,credit_data_cat_columns,all_columns=df_OHE(credit_data)
    with open("all_columns_credit_data.pkl", "wb") as f:
        pkl.dump(all_columns, f)

    credit_data_agg={}
    for col in credit_data.columns:
        if col!='SK_ID_CURR' and col !='SK_ID_PREV':
            credit_data_agg[col]=['mean']
            if (col=='FLAG_GRT_30')|(col=='NAME_CONTRACT_STATUS') :
                credit_data_agg[col]=['sum']
            if col=='MONTH_BALANCE':
                credit_data_agg[col]=['min','mean']


    credit_agg =
credit_data.groupby('SK_ID_CURR').agg(credit_data_agg)

    modified_col=[]
    for c in list(credit_agg.columns):
        modified_col.append("CRED_"+c[0]+"_"+c[1].upper())
    credit_agg.columns=modified_col

    month = -3
    cred_temp = credit_data[credit_data.MONTHS_BALANCE >=
month].copy()
    cred_temp['CRED_UTIL_PER'] = (cred_temp['AMT_BALANCE'])/
cred_temp['AMT_CREDIT_LIMIT_ACTUAL']
    credit_agg['3_CREDIT_UTIL_PER_MEAN'] =
cred_temp.groupby('SK_ID_CURR')['CRED_UTIL_PER'].mean()

    month = -6
    cred_temp = credit_data[credit_data.MONTHS_BALANCE >=
month].copy()
    cred_temp['CRED_UTIL_PER'] = (cred_temp['AMT_BALANCE'])/
cred_temp['AMT_CREDIT_LIMIT_ACTUAL']
    credit_agg['6_CREDIT_UTIL_PER_MEAN'] =
cred_temp.groupby('SK_ID_CURR')['CRED_UTIL_PER'].mean()

    month = -12
    cred_temp = credit_data[credit_data.MONTHS_BALANCE >=
month].copy()
    cred_temp['CRED_UTIL_PER'] = (cred_temp['AMT_BALANCE'])/
cred_temp['AMT_CREDIT_LIMIT_ACTUAL']
    credit_agg['12_CREDIT_UTIL_PER_MEAN'] =
```

```
cred_temp.groupby('SK_ID_CURR')['CRED_UTIL_PER'].mean()

    month = -24
    cred_temp = credit_data[credit_data.MONTHS_BALANCE >=
month].copy()
    cred_temp['CRED_UTIL_PER'] = (cred_temp['AMT_BALANCE'])/
cred_temp['AMT_CREDIT_LIMIT_ACTUAL']
    credit_agg['24_CREDIT_UTIL_PER_MEAN'] =
cred_temp.groupby('SK_ID_CURR')['CRED_UTIL_PER'].mean()
    credit_agg['COUNT_OF_CREDITS'] = credit_data.groupby('SK_ID_CURR')
['SK_ID_PREV'].count()

    return credit_agg
```

Feature engineering on Installments payments balance

```
def fe_installments_payments_balance():

    installments_payments_data =
pd.read_csv('./DATA/installments_payments.csv')



installments_payments_data['LATE_PAYMENT']=installments_payments_data[
'DAYS_INSTALMENT']-installments_payments_data['DAYS_ENTRY_PAYMENT']


installments_payments_data['LESS_PAYMENT']=installments_payments_data[
'AMT_INSTALMENT']-installments_payments_data['AMT_PAYMENT']


installments_payments_data['LATE_LESS_PAYMENT']=0.5*installments_payme
nts_data['LATE_PAYMENT']
+0.5*installments_payments_data['LESS_PAYMENT']


installments_payments_data['LATE_PAYMENT_FLAG']=((installments_payment
s_data['DAYS_INSTALMENT']-
installments_payments_data['DAYS_ENTRY_PAYMENT'])>0).astype(int)


installments_payments_data['LESS_PAYMENT_FLAG']=((installments_payment
s_data['AMT_INSTALMENT']-
installments_payments_data['AMT_PAYMENT'])>0).astype(int)

    for col in installments_payments_data.columns:
        if col.startswith('DAYS'):
            installments_payments_data[col].replace(365243, np.nan,
inplace= True)
```

```
installments_payments_data,installments_payments_cat_columns,all_colum
ns=df_OHE(installments_payments_data)
    with open("all_columns_installments_payments_data.pkl", "wb") as
f:
        pkl.dump(all_columns, f)

    installments_payments_data_agg={}
    for col in installments_payments_data.columns:
        if col!='SK_ID_CURR' and col !='SK_ID_PREV':
            installments_payments_data_agg[col]=['mean']
            if (col=='LATE_PAYMENT') |  (col=='LESS_PAYMENT') |
(col=='NUM_INSTALMENT_VERSION') | (col=='NUM_INSTALMENT_NUMBER'):

installments_payments_data_agg[col]=['mean','sum','max','min']

    installments_payments_agg =
installments_payments_data.groupby('SK_ID_CURR').agg(installments_paym
ents_data_agg)

    modified_col=[]
    for c in list(installments_payments_agg.columns):
        modified_col.append("INST_"+c[0]+"_"+c[1].upper())
    installments_payments_agg.columns=modified_col


    installments_payments_agg['COUNT_OF_INST'] =
installments_payments_data.groupby('SK_ID_CURR')['SK_ID_PREV'].count()



    no = -365*3
    installments_payments_agg_temp =
installments_payments_data[installments_payments_data.DAYS_ENTRY_PAYME
NT >=no].copy()
    installments_payments_agg['3365_LATE_PAYMENT_FLAG_MEAN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT'].mean()
    installments_payments_agg['3365_LATE_PAYMENT_FLAG_MIN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT'].min()
    installments_payments_agg['3365_LATE_PAYMENT_FLAG_MAX'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT'].max()
    installments_payments_agg['3365_LATE_PAYMENT_FLAG_COUNT'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT_FLAG'].sum()



    no = -365*2
```

```
    installments_payments_agg_temp =
installments_payments_data[installments_payments_data.DAYS_ENTRY_PAYME
NT >=no].copy()
    installments_payments_agg['2365_LATE_PAYMENT_FLAG_MEAN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT'].mean()
    installments_payments_agg['2365_LATE_PAYMENT_FLAG_MIN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT'].min()
    installments_payments_agg['2365_LATE_PAYMENT_FLAG_MAX'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT'].max()
    installments_payments_agg['2365_LATE_PAYMENT_FLAG_COUNT'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT_FLAG'].sum()


    no = -365
    installments_payments_agg_temp =
installments_payments_data[installments_payments_data.DAYS_ENTRY_PAYME
NT >=no].copy()
    installments_payments_agg['365_LATE_PAYMENT_FLAG_MEAN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT'].mean()
    installments_payments_agg['365_LATE_PAYMENT_FLAG_MIN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT'].min()
    installments_payments_agg['365_LATE_PAYMENT_FLAG_MAX'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT'].max()
    installments_payments_agg['365_LATE_PAYMENT_FLAG_COUNT'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT_FLAG'].sum()

    no = -180
    installments_payments_agg_temp =
installments_payments_data[installments_payments_data.DAYS_ENTRY_PAYME
NT >= no].copy()
    installments_payments_agg['180_LATE_PAYMENT_FLAG_MEAN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT'].mean()
    installments_payments_agg['180_LATE_PAYMENT_FLAG_MIN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT'].min()
    installments_payments_agg['180_LATE_PAYMENT_FLAG_MAX'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT'].max()
    installments_payments_agg['180_LATE_PAYMENT_FLAG_COUNT'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT_FLAG'].sum()
```

```
    no = -90
    installments_payments_agg_temp =
installments_payments_data[installments_payments_data.DAYS_ENTRY_PAYME
NT >= no].copy()
    installments_payments_agg['90_LATE_PAYMENT_FLAG_MEAN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT'].mean()
    installments_payments_agg['90_LATE_PAYMENT_FLAG_MIN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT'].min()
    installments_payments_agg['90_LATE_PAYMENT_FLAG_MAX'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT'].max()
    # installments_payments_agg['90_LATE_PAYMENT_FLAG_COUNT'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT_FLAG'].sum()

    no = -30
    installments_payments_agg_temp =
installments_payments_data[installments_payments_data.DAYS_ENTRY_PAYME
NT >= no].copy()
    installments_payments_agg['30_LATE_PAYMENT_FLAG_MEAN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT'].mean()
    installments_payments_agg['30_LATE_PAYMENT_FLAG_MIN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT'].min()
    installments_payments_agg['30_LATE_PAYMENT_FLAG_MAX'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LATE_PAYMENT'].max()

    no = -365*2
    installments_payments_agg_temp =
installments_payments_data[installments_payments_data.DAYS_ENTRY_PAYME
NT >=no].copy()
    installments_payments_agg['2365_LESS_PAYMENT_FLAG_MEAN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LESS_PAYMENT'].mean()
    installments_payments_agg['2365_LESS_PAYMENT_FLAG_MIN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LESS_PAYMENT'].min()
    installments_payments_agg['2365_LESS_PAYMENT_FLAG_MAX'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LESS_PAYMENT'].max()
    installments_payments_agg['2365_LESS_PAYMENT_FLAG_COUNT'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LESS_PAYMENT_FLAG'].sum()

    no = -365
```

```
    installments_payments_agg_temp =
installments_payments_data[installments_payments_data.DAYS_ENTRY_PAYME
NT >=no].copy()
    installments_payments_agg['365_LESS_PAYMENT_FLAG_MEAN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LESS_PAYMENT'].mean()
    installments_payments_agg['365_LESS_PAYMENT_FLAG_MIN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LESS_PAYMENT'].min()
    installments_payments_agg['365_LESS_PAYMENT_FLAG_MAX'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LESS_PAYMENT'].max()
    installments_payments_agg['365_LESS_PAYMENT_FLAG_COUNT'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LESS_PAYMENT_FLAG'].sum()

    no = -180
    installments_payments_agg_temp =
installments_payments_data[installments_payments_data.DAYS_ENTRY_PAYME
NT >= no].copy()
    installments_payments_agg['180_LESS_PAYMENT_FLAG_MEAN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LESS_PAYMENT'].mean()
    installments_payments_agg['180_LESS_PAYMENT_FLAG_MIN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LESS_PAYMENT'].min()
    installments_payments_agg['180_LESS_PAYMENT_FLAG_MAX'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LESS_PAYMENT'].max()
    no = -90
    installments_payments_agg_temp =
installments_payments_data[installments_payments_data.DAYS_ENTRY_PAYME
NT >= no].copy()
    installments_payments_agg['90_LESS_PAYMENT_FLAG_MEAN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LESS_PAYMENT'].mean()
    installments_payments_agg['90_LESS_PAYMENT_FLAG_MIN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LESS_PAYMENT'].min()
    installments_payments_agg['90_LESS_PAYMENT_FLAG_MAX'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LESS_PAYMENT'].max()

    no = -30
    installments_payments_agg_temp =
installments_payments_data[installments_payments_data.DAYS_ENTRY_PAYME
NT >= no].copy()
    installments_payments_agg['30_LESS_PAYMENT_FLAG_MEAN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LESS_PAYMENT'].mean()
```

```
    installments_payments_agg['30_LESS_PAYMENT_FLAG_MIN'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LESS_PAYMENT'].min()
    installments_payments_agg['30_LESS_PAYMENT_FLAG_MAX'] =
installments_payments_agg_temp.groupby('SK_ID_CURR')
['LESS_PAYMENT'].max()



    return installments_payments_agg
```

Merging Data

```
df=train_test_feature_engineering()

df_bureau_agg=fe_of_bureau_balance()
df=df.join(df_bureau_agg,how='left', on='SK_ID_CURR')
del df_bureau_agg
gc.collect()


df_prev_agg=fe_previous_application()
df=df.join(df_prev_agg,how='left', on='SK_ID_CURR')
del df_prev_agg
gc.collect()

df_pos_agg=fe_pos_application()
df=df.join(df_pos_agg,how='left', on='SK_ID_CURR')
del df_pos_agg
gc.collect()

df_credit_agg=fe_credit_card_balance()
df=df.join(df_credit_agg,how='left', on='SK_ID_CURR')
del df_credit_agg
gc.collect()

df_payments_agg=fe_installments_payments_balance()
df=df.join(df_payments_agg,how='left', on='SK_ID_CURR')
del df_payments_agg
gc.collect()
```

```
0
```

```
df.head()
```

```
   index  SK_ID_CURR  TARGET  CNT_CHILDREN  AMT_INCOME_TOTAL
AMT_CREDIT  \
0      0      100002     1.0             0          202500.0
406597.5
1      1      100003     0.0             0          270000.0
```

```
1293502.5
2      2      100004     0.0              0          67500.0
135000.0
3      3      100006     0.0              0         135000.0
312682.5
4      4      100007     0.0              0         121500.0
513000.0

   AMT_ANNUITY  AMT_GOODS_PRICE  REGION_POPULATION_RELATIVE
DAYS_BIRTH  ...  \
0      24700.5         351000.0                    0.018801       -
9461  ...
1      35698.5        1129500.0                    0.003541      -
16765  ...
2       6750.0         135000.0                    0.010032      -
19046  ...
3      29686.5         297000.0                    0.008019      -
19005  ...
4      21865.5         513000.0                    0.028663      -
19932  ...

   365_LESS_PAYMENT_FLAG_COUNT  180_LESS_PAYMENT_FLAG_MEAN  \
0                          0.0                         0.0
1                          NaN                         NaN
2                          NaN                         NaN
3                          0.0                         0.0
4                          0.0                         0.0

   180_LESS_PAYMENT_FLAG_MIN  180_LESS_PAYMENT_FLAG_MAX  \
0                        0.0                        0.0
1                        NaN                        NaN
2                        NaN                        NaN
3                        0.0                        0.0
4                        0.0                        0.0

   90_LESS_PAYMENT_FLAG_MEAN  90_LESS_PAYMENT_FLAG_MIN  \
0                        0.0                       0.0
1                        NaN                       NaN
2                        NaN                       NaN
3                        0.0                       0.0
4                        0.0                       0.0

   90_LESS_PAYMENT_FLAG_MAX  30_LESS_PAYMENT_FLAG_MEAN  \
0                       0.0                        NaN
1                       NaN                        NaN
2                       NaN                        NaN
3                       0.0                        0.0
4                       0.0                        0.0

   30_LESS_PAYMENT_FLAG_MIN  30_LESS_PAYMENT_FLAG_MAX
```

```
0                       NaN                       NaN
1                       NaN                       NaN
2                       NaN                       NaN
3                       0.0                       0.0
4                       0.0                       0.0

[5 rows x 745 columns]
```

Separating train and test data

```
train_df=df[df['TARGET'].notnull()]
test_df=df[df['TARGET'].isnull()]
```

Handeling missing data

replacing infinity with null values

```
train_df.replace([np.inf, -np.inf], np.nan,inplace=True)
test_df.replace([np.inf, -np.inf], np.nan,inplace=True)

def find_missing_data_features(df):
    missing_data=df.isnull().sum()
    missing_data_percent =100 * df.isnull().sum() / len(df)
    missing_data_table = pd.concat([missing_data, 
missing_data_percent], axis=1)
    missing_data_table_ren_columns = missing_data_table.rename(columns
= {0 : 'Missing Values', 1 : '% of Total Values'})
    missing_data_table_ren_columns = 
missing_data_table_ren_columns[missing_data_table_ren_columns.iloc[:,1
] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)
    print ("Totals Features " + str(df.shape[1]) + "\n"
              + str(missing_data_table_ren_columns.shape[0]) + 
              " features have missing values.")

    return missing_data_table_ren_columns

missing_values_data=find_missing_data_features(train_df)

Totals Features 745
556 features have missing values.
```

Dropping Features with more than 60% null data

```
missing_6=missing_values_data[(missing_values_data['% of Total 
Values']>60)]
len(list(missing_6.index))

98
```

```
train_df=train_df.drop(columns=list(missing_3.index)).copy()
test_df=test_df.drop(columns=list(missing_3.index)).copy()

train_df.shape, test_df.shape

((307506, 647), (48744, 647))
```

Imputing median for features with less than 60% null data

```
miss_data_60=missing_values_data[missing_values_data['% of Total
Values']<=60]
miss_data_60

                        Missing Values  % of Total Values
LANDAREA_MEDI                   182588               59.4
LANDAREA_AVG                    182588               59.4
LANDAREA_MODE                   182588               59.4
BASEMENTAREA_AVG                179942               58.5
BASEMENTAREA_MODE               179942               58.5
...                                ...                ...
INCOME_LEFT                         12                0.0
AMT_ANNUITY                         12                0.0
INCOME_PER_PERSON                    2                0.0
CNT_FAM_MEMBERS                      2                0.0
DAYS_LAST_PHONE_CHANGE               1                0.0

[458 rows x 2 columns]
```

We are using median because it is less prone to outliers.

```
from sklearn.impute import SimpleImputer
imputer_60 = SimpleImputer(missing_values=np.nan, strategy='median')

imputer_60.fit(train_df[list(miss_data_60.index)])

SimpleImputer(strategy='median')

%%time
train_df_median_df =
imputer_60.transform(train_df[list(miss_data_60.index)])
test_df_median_df  =
imputer_60.transform(test_df[list(miss_data_60.index)])

miss_data_60_col=list(miss_data_60.index)

train_df.loc[:,miss_data_60_col]=train_df_median_df.copy()
test_df.loc[:,miss_data_60_col]=test_df_median_df.copy()

CPU times: user 3min 50s, sys: 49min 24s, total: 53min 14s
Wall time: 1h 19min 25s
```

Separating train, target and test data

```
train_col=set(train_df.columns)-{'TARGET','SK_ID_CURR','index'}
y_train=train_df['TARGET']
X_train=train_df[train_col]
kaggle_test=test_df[train_col]
```

Standardizing data using MINMAXSCALER

```
scaler = MinMaxScaler(feature_range = (0, 1))
X_train = scaler.fit_transform(X_train)
kaggle_test = scaler.transform(kaggle_test)

with open("train_final_data.pkl", "wb") as f:
    pkl.dump([X_train,y_train], f)
with open("kaggle_test_data.pkl", "wb") as f:
    pkl.dump(kaggle_test, f)

X_train.shape, y_train.shape

((307506, 644), (307506,))

kaggle_test.shape

(48744, 644)
```

Hyperparameter tuning

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

Splitting data into train and cross validation data

```
X_train,X_cv,y_train,y_cv = train_test_split(X_train,
y_train,stratify=y_train,test_size=0.20)
```

Evaluating and Tuning Models

Logging scores

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import log_loss
```

```
data = {'Model': [],
       'Accuracy': [],
       'ROC_AUC': [],
       'F1-Score': [],
       'Precision': [],
       'Recall': [],
       'Log-Loss': []}

model_score = pd.DataFrame(data)
```

```
model_score.head()
```

```
Empty DataFrame
Columns: [Model, Accuracy, ROC_AUC, F1-Score, Precision, Recall, Log-
Loss]
Index: []
```

```
from sklearn.metrics import roc_auc_score
```

Logistic Regression

```
pipe = Pipeline([('scaler', StandardScaler()),
                 ('lr', LogisticRegression())])

param_grid = {'lr__penalty':['l1', 'l2','none'],
              'lr__C': [1, 10, 100, 1000, 10000]
             }

gs_logistic=RandomizedSearchCV(pipe,param_grid,cv=5,n_jobs=-
1,verbose=1, refit = True, scoring='roc_auc',)

gs_logistic.fit(X_train,y_train)
```

```
gs_logistic.best_params_
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:1483: UserWarning: Setting penalty='none'
will ignore the C and l1_ratio parameters
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:1483: UserWarning: Setting penalty='none' will
ignore the C and l1_ratio parameters
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:1483: UserWarning: Setting penalty='none' will
```

```
ignore the C and l1_ratio parameters
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:1483: UserWarning: Setting penalty='none' will
ignore the C and l1_ratio parameters
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:1483: UserWarning: Setting penalty='none' will
ignore the C and l1_ratio parameters
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
```

```
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
```

```
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:1483: UserWarning: Setting penalty='none' will
ignore the C and l1_ratio parameters
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:1483: UserWarning: Setting penalty='none' will
ignore the C and l1_ratio parameters
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:1483: UserWarning: Setting penalty='none' will
ignore the C and l1_ratio parameters
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:1483: UserWarning: Setting penalty='none' will
ignore the C and l1_ratio parameters
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:1483: UserWarning: Setting penalty='none' will
ignore the C and l1_ratio parameters
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
```

```
https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
```

```
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
```

```
https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
{'lr__penalty': 'l2', 'lr__C': 1}
```

```python
# find best model score
gs_logistic.score(X_cv, y_cv)
```

```
0.7720790176334007
```

```
roc_auc_score(y_cv, gs_logistic.predict_proba(X_cv)[:, 1])
```

```
0.7720790176334007
```

Training with best parameters

```
log_clf = LogisticRegression(penalty='l2', C=1000)
log_clf.fit(X_train, y_train)

y_pred = log_clf.predict(X_cv)
# y_prob = log_clf.predict_proba(X_cv)[:, 1]

model_score.loc[len(model_score)] = ["Logistic Regression",
                                     np.round(accuracy_score(y_cv, 
y_pred), 3),
                                      np.round(roc_auc_score(y_cv, 
log_clf.predict_proba(X_cv)[:, 1]), 3),
                                      np.round(f1_score(y_cv, 
y_pred,average='weighted'), 3),
                                      np.round(precision_score(y_cv, 
y_pred), 3),
                                      np.round(recall_score(y_cv, 
y_pred), 3),
                                      np.round(log_loss(y_cv, y_pred), 
3)
                                     
                                     ]

model_score.head()
```

```
                 Model  Accuracy  ROC_AUC  F1-Score  Precision  Recall  
\
0  Logistic Regression      0.92    0.769     0.885      0.545    0.03   


   Log-Loss  
0     2.774
```

SVM

```
svc_pipeline = Pipeline([('scaler', StandardScaler()),
                 ('svm', SVC(max_iter=50))])

parameters={'svm__C': (.1, 1, 10),        ## access parameters for the 
estimator inside
            'svm__kernel': ('linear', 'poly', 'rbf'),
            'svm__gamma': [1, 0.1, 0.01, 0.001, 0.0001],
            'svm__degree': (1,2,3) #since poly is used as a kernel
           }

gs_SVM=RandomizedSearchCV(svc_pipeline,parameters,cv=5,n_jobs=-
```

```
1,verbose=2, refit = True, scoring='roc_auc',)
gs_SVM.fit(X_train,y_train)
gs_SVM.best_params_
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
```

```
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
```

```
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
```

```
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
```

```
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
```

```
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/svm/
_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=50).  Consider pre-processing your data with StandardScaler
or MinMaxScaler.
  warnings.warn(
```

```
{'svm__kernel': 'rbf', 'svm__gamma': 1, 'svm__degree': 2, 'svm__C':
10}
```

```
# find best model score
gs_SVM.score(X_cv, y_cv)
```

```
0.5778639610050658
```

```
# roc_auc_score(y_cv, gs_SVM.predict_proba(X_cv)[:, 1])
```

Training with best parameters

```
SVM_clf = SVC(kernel='rbf', gamma=0.01, degree=2, C=1,
probability=True)
SVM_clf.fit(X_train, y_train)

roc_auc_score(y_cv, SVM_clf.predict_proba(X_cv)[:, 1])
```

```
[CV] END svm__C=10, svm__degree=2, svm__gamma=0.001, svm__kernel=rbf;
total time= 1.1min
[CV] END svm__C=1, svm__degree=3, svm__gamma=0.001,
svm__kernel=linear; total time= 1.3min
[CV] END svm__C=10, svm__degree=2, svm__gamma=1, svm__kernel=rbf;
total time=  57.1s
[CV] END svm__C=10, svm__degree=3, svm__gamma=1, svm__kernel=linear;
total time= 1.7min
[CV] END svm__C=0.1, svm__degree=3, svm__gamma=0.01, svm__kernel=poly;
total time= 1.0min
[CV] END svm__C=10, svm__degree=2, svm__gamma=1, svm__kernel=poly;
total time=  51.8s
[CV] END svm__C=10, svm__degree=3, svm__gamma=0.01, svm__kernel=poly;
total time=  28.6s
[CV] END svm__C=1, svm__degree=3, svm__gamma=0.001,
svm__kernel=linear; total time= 1.0min
[CV] END svm__C=10, svm__degree=2, svm__gamma=1, svm__kernel=rbf;
total time=  56.9s
[CV] END svm__C=10, svm__degree=3, svm__gamma=1, svm__kernel=linear;
total time= 1.7min
[CV] END svm__C=0.1, svm__degree=3, svm__gamma=0.01, svm__kernel=poly;
total time= 1.0min
[CV] END svm__C=10, svm__degree=2, svm__gamma=0.001, svm__kernel=rbf;
total time= 1.1min
[CV] END svm__C=10, svm__degree=3, svm__gamma=0.01, svm__kernel=poly;
total time=  40.9s
[CV] END svm__C=1, svm__degree=3, svm__gamma=0.001,
svm__kernel=linear; total time=  50.1s
```

```
[CV] END svm__C=10, svm__degree=1, svm__gamma=0.0001,
svm__kernel=linear; total time=  39.8s
[CV] END svm__C=10, svm__degree=3, svm__gamma=1, svm__kernel=linear;
total time= 1.1min
[CV] END svm__C=1, svm__degree=2, svm__gamma=0.01, svm__kernel=rbf;
total time= 1.6min
[CV] END svm__C=0.1, svm__degree=3, svm__gamma=0.01, svm__kernel=poly;
total time=  11.8s
[CV] END svm__C=10, svm__degree=2, svm__gamma=0.001, svm__kernel=rbf;
total time= 1.1min
[CV] END svm__C=10, svm__degree=3, svm__gamma=0.01, svm__kernel=poly;
total time= 1.2min
[CV] END svm__C=10, svm__degree=2, svm__gamma=1, svm__kernel=rbf;
total time=  59.0s
[CV] END svm__C=10, svm__degree=1, svm__gamma=0.0001,
svm__kernel=linear; total time=  55.8s
[CV] END svm__C=1, svm__degree=2, svm__gamma=0.01, svm__kernel=rbf;
total time= 1.8min
[CV] END svm__C=0.1, svm__degree=2, svm__gamma=1, svm__kernel=rbf;
total time=  15.6s
[CV] END svm__C=10, svm__degree=2, svm__gamma=0.001, svm__kernel=rbf;
total time= 1.1min
[CV] END svm__C=10, svm__degree=3, svm__gamma=0.01, svm__kernel=poly;
total time= 1.2min
[CV] END svm__C=10, svm__degree=2, svm__gamma=1, svm__kernel=rbf;
total time=  59.7s
[CV] END svm__C=10, svm__degree=1, svm__gamma=0.0001,
svm__kernel=linear; total time= 1.3min
[CV] END svm__C=1, svm__degree=2, svm__gamma=0.01, svm__kernel=rbf;
total time= 1.4min
[CV] END svm__C=0.1, svm__degree=2, svm__gamma=1, svm__kernel=rbf;
total time=  14.2s
[CV] END svm__C=10, svm__degree=2, svm__gamma=1, svm__kernel=poly;
total time=  50.9s
[CV] END svm__C=10, svm__degree=2, svm__gamma=1, svm__kernel=poly;
total time= 1.2min
[CV] END svm__C=1, svm__degree=3, svm__gamma=0.001,
svm__kernel=linear; total time=  39.8s
[CV] END svm__C=10, svm__degree=1, svm__gamma=0.0001,
svm__kernel=linear; total time=  37.3s
[CV] END svm__C=10, svm__degree=3, svm__gamma=1, svm__kernel=linear;
total time= 1.7min
[CV] END svm__C=0.1, svm__degree=3, svm__gamma=0.01, svm__kernel=poly;
total time= 1.0min
[CV] END svm__C=0.1, svm__degree=2, svm__gamma=1, svm__kernel=rbf;
total time=  15.0s
[CV] END svm__C=10, svm__degree=2, svm__gamma=0.001, svm__kernel=rbf;
total time= 1.1min
[CV] END svm__C=10, svm__degree=3, svm__gamma=0.01, svm__kernel=poly;
total time= 1.2min
```

```
[CV] END svm__C=10, svm__degree=2, svm__gamma=1, svm__kernel=rbf;
total time=  58.4s
[CV] END svm__C=10, svm__degree=3, svm__gamma=1, svm__kernel=linear;
total time= 1.4min
[CV] END svm__C=1, svm__degree=2, svm__gamma=0.01, svm__kernel=rbf;
total time= 1.4min
[CV] END svm__C=0.1, svm__degree=2, svm__gamma=1, svm__kernel=rbf;
total time=  15.7s
[CV] END svm__C=10, svm__degree=2, svm__gamma=1, svm__kernel=poly;
total time=  51.3s
[CV] END svm__C=10, svm__degree=2, svm__gamma=1, svm__kernel=poly;
total time= 1.2min
[CV] END svm__C=1, svm__degree=3, svm__gamma=0.001,
svm__kernel=linear; total time=  55.2s
[CV] END svm__C=10, svm__degree=1, svm__gamma=0.0001,
svm__kernel=linear; total time=  38.0s
[CV] END svm__C=1, svm__degree=2, svm__gamma=0.01, svm__kernel=rbf;
total time= 1.6min
[CV] END svm__C=0.1, svm__degree=3, svm__gamma=0.01, svm__kernel=poly;
total time=  54.4s
[CV] END svm__C=0.1, svm__degree=2, svm__gamma=1, svm__kernel=rbf;
total time=  15.6s
```

```
y_pred = SVC_clf.predict(X_cv)
model_score.loc[len(model_score)] = ["SVM",
                                    np.round(accuracy_score(y_cv,
y_pred), 3),

                                     np.round(roc_auc_score(y_cv,
SVM_clf.predict_proba(X_cv)[:, 1]), 3),
                                     np.round(f1_score(y_cv,
y_pred,average='weighted'), 3),

                                     np.round(precision_score(y_cv,
y_pred), 3),

                                     np.round(recall_score(y_cv,
y_pred), 3),

                                   np.round(log_loss(y_cv, y_pred), 3)
                                    ]
```

Random Forest

```
rf_pipeline = Pipeline([('scaler', StandardScaler()),
                 ('rf', RandomForestClassifier())])

parameters = {
    'rf__n_estimators': [200, 500],
    'rf__max_features': ['auto', 'sqrt', 'log2'],
    'rf__max_depth' : [4,5,6,7,8],
    'rf__criterion' :['gini', 'entropy']
```

```
}

gs_RF=RandomizedSearchCV(rf_pipeline,parameters,cv=5,n_jobs=-
1,verbose=2, refit = True, scoring='roc_auc',)
gs_RF.fit(X_train,y_train)
gs_RF.best_params_

# find best model score
gs_RF.score(X_cv, y_cv)

# roc_auc_score(y_cv, gs_RF.predict_proba(X_cv)[:, 1])
```

Training with best parameters

```
RF_clf = RandomizedSearchCV(n_estimators=500, max_features=0.01,
max_depth=5, criterion='entropy')
RF_clf.fit(X_train, y_train)

y_pred = RF_clf.predict(X_cv)
model_score.loc[len(model_score)] = ["Random forrest",
                                    np.round(accuracy_score(y_cv,
y_pred), 3),
                                     np.round(roc_auc_score(y_cv,
RF_clf.predict_proba(X_cv)[:, 1]), 3),
                                     np.round(f1_score(y_cv,
y_pred,average='weighted'), 3),
                                     np.round(precision_score(y_cv,
y_pred), 3),
                                     np.round(recall_score(y_cv,
y_pred), 3),
                                     np.round(log_loss(y_cv, y_pred),
3)

                                    ]
```

XGBOOST

```
import xgboost
pipeline = Pipeline([
        ('scaler', StandardScaler()),
        ("xgb", xgboost.XGBClassifier())
    ])

params={
 "xgb__learning_rate"    : [0.05, 0.10] ,
 "xgb__max_depth"        : [ 3,5, 7],
 "xgb__min_child_weight" : [ 1, 3, 5]
}
```

```
rs_XGBOOST=RandomizedSearchCV(pipeline,
                                 param_distributions=params,
                                 n_iter=10,
                                 scoring='accuracy',
                                 n_jobs=-1,
                                 cv=5,
                                 verbose=1
                             , refit = True,
                             scoring='roc_auc',)
rs_XGBOOST.fit(X_train,y_train)
rs_XGBOOST.best_params_

# find best model score
rs_XGBOOST.score(X_cv, y_cv) ### Training with best parameters
```

Training with best parameters

```
XGB_clf = xgboost.XGBClassifier(min_child_weight=1, max_depth=7,
learning_rate=0.1)
XGB_clf.fit(X_train, y_train)

y_pred = XGB_clf.predict(X_cv)
model_score.loc[len(model_score)] = ["XGBOOST",
                                    np.round(accuracy_score(y_cv,
y_pred), 3),
                                     np.round(roc_auc_score(y_cv,
XGB_clf.predict_proba(X_cv)[:, 1]), 3),
                                     np.round(f1_score(y_cv,
y_pred,average='weighted'), 3),
                                     np.round(precision_score(y_cv,
y_pred), 3),
                                     np.round(recall_score(y_cv,
y_pred), 3),
                                     np.round(log_loss(y_cv, y_pred),
3)

                                    ]
```

Naive Bayes

```
from sklearn.naive_bayes import MultinomialNB

nb_pipeline = Pipeline([
        ("nb", MultinomialNB())
    ])


parameters = {
    'nb__alpha': [0.01, 0.1, 1, 10]
```

```
}

rs_NB=RandomizedSearchCV(nb_pipeline,parameters,cv=5,n_jobs=-
1,verbose=2, refit = True, scoring='roc_auc',)
rs_NB.fit(X_train,y_train)
rs_NB.best_params_

# find best model score
rs_NB.score(X_cv, y_cv)
```

Training with best parameters

```
NB_clf = xgboost.XGBClassifier(min_child_weight=1, max_depth=7,
learning_rate=0.1)
NB_clf.fit(X_train, y_train)

y_pred = NB_clf.predict(X_cv)
model_score.loc[len(model_score)] = ["Naive Bayes",
                                    np.round(accuracy_score(y_cv,
y_pred), 3),
                                     np.round(roc_auc_score(y_cv,
NB_clf.predict_proba(X_cv)[:, 1]), 3),
                                     np.round(f1_score(y_cv,
y_pred,average='weighted'), 3),
                                     np.round(precision_score(y_cv,
y_pred), 3),
                                     np.round(recall_score(y_cv,
y_pred), 3),
                                     np.round(log_loss(y_cv, y_pred),
3)

                                    ]
```

Evaluating all models

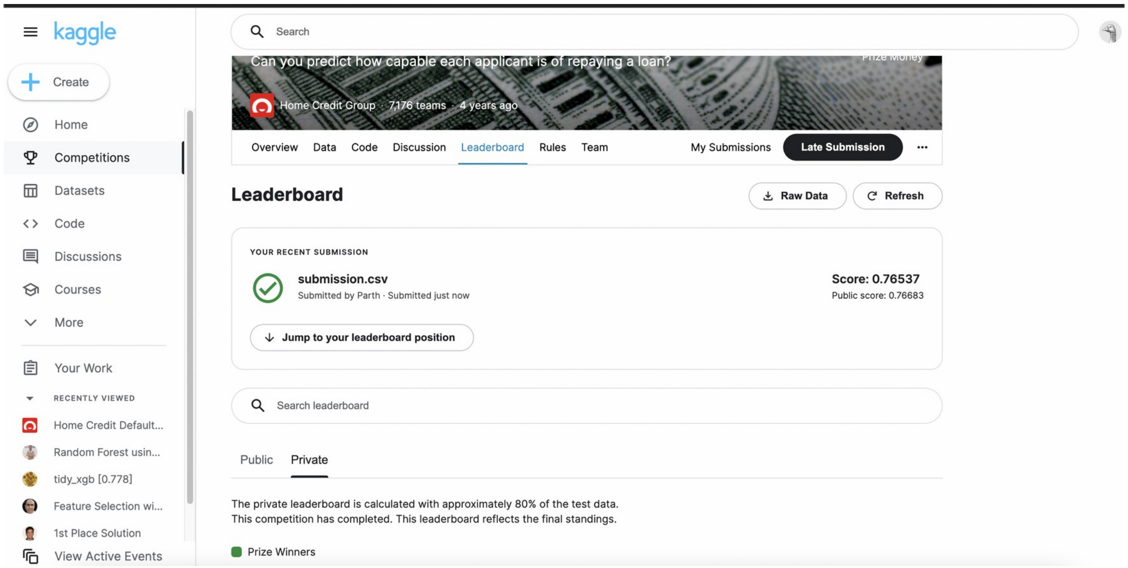
```
model_score.head()
```

from all of our experiments XGBOOST performed the best. We will use this model to submit our kaggle submission

Kaggle Submission

```
import pandas as pd
import seaborn as sns
import plotly.express as px
import numpy as np
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
```

```python
import gc
sns.set_style("whitegrid");
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
import pickle as pkl
import tqdm as tqdm
from random import choices
from sklearn.impute import SimpleImputer

from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import log_loss
from sklearn.metrics import roc_auc_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from tqdm import tqdm
# from bayes_opt import BayesianOptimization
from lightgbm import LGBMClassifier
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.model_selection import KFold,StratifiedKFold
from sklearn.linear_model import Ridge

import warnings
warnings.filterwarnings('ignore')

with open("train_final_data.pkl", "rb") as f:
    X_train,y_train = pkl.load(f)
with open("kaggle_test_data.pkl", "rb") as f:
    kaggle_test_data = pkl.load(f)

from sklearn.linear_model import LogisticRegression
log_clf = LogisticRegression(penalty='l2', C=1000)
log_clf.fit(X_train, y_train)

test_class_scores = log_clf.predict_proba(kaggle_test_data)[:, 1]

# Submission dataframe
submit_df = datasets["application_test"][['SK_ID_CURR']]
submit_df['TARGET'] = test_class_scores

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



References

1. https://www.kaggle.com/competitions/home-credit-default-risk/discussion?sort=votes
2. https://www.youtube.com/watch?v=3-0g84_nGnw&ab_channel=XinZhao
3. https://medium.com/analytics-vidhya/home-credit-loan-default-risk-7d660ce22942
4. https://medium.com/@praveenkotha/home-credit-default-risk-end-to-end-machine-learning-project-1871f52e3ef2

Abstract

Credit Rating is one of the most important parameters for availing home loan.In our project, we will address all the factors to accurately predict the ability of an applicant to repay loan.To do so our team intends towards building different machine learning models and choose the best of the lot. The data used for building these ML models is referred from the competition project, Home Credit Default Risk on Kaggle. In this phase(Phase 2) we implemented concepts like feature engineering and hyperparameter tuning with the aim of increasing the overall accuracy of the models. For feature engineering we combined the domain knowledge(read about the domain) with the input obtained from data visualization(or EDA performed in the previous phase). While performing feature engineering we focused towards making the features highly correlated with the target variable. After this we ran multiple model pipelines namely Logistic Regression, Random Forest, XGBoost, Naive Bayes and Support Vector Machine. Subsequently we evaluated the F1 and AUC-ROC score for the mentioned models using cross validation technique. Other than this we performed hyperparameter tuning using GridSearchCV and RandomSearchCV to optimize the results. Among all models XGBoost performs the best with the ROC-AUC score of 0.77. In our next and final phase(Phase 3) we will implement a neural network (MultiLayer Perceptron) using Pytorch besides creating a tensor board to carry out experimental analysis of our obtained accuracy. Neural Networks will boost the accuracy

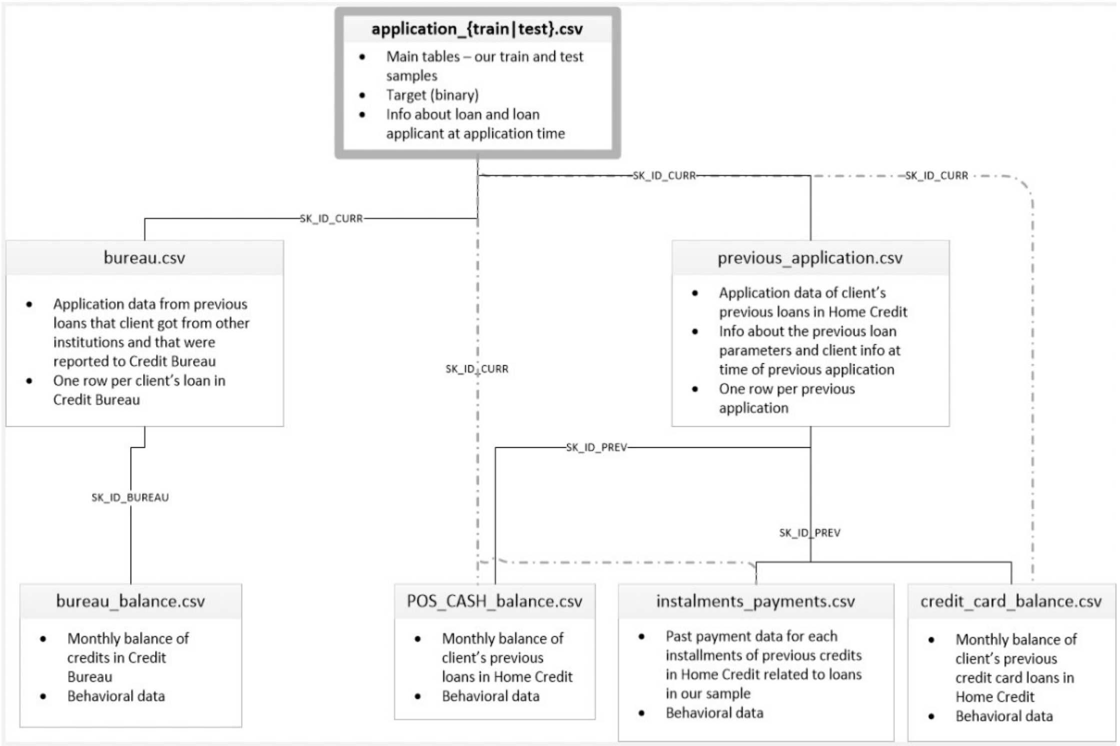
and enable us to obtain the most desired accuracy required to predict the ability of the applicant to repay the loan (meet our final goal).

Project Description

Data Description

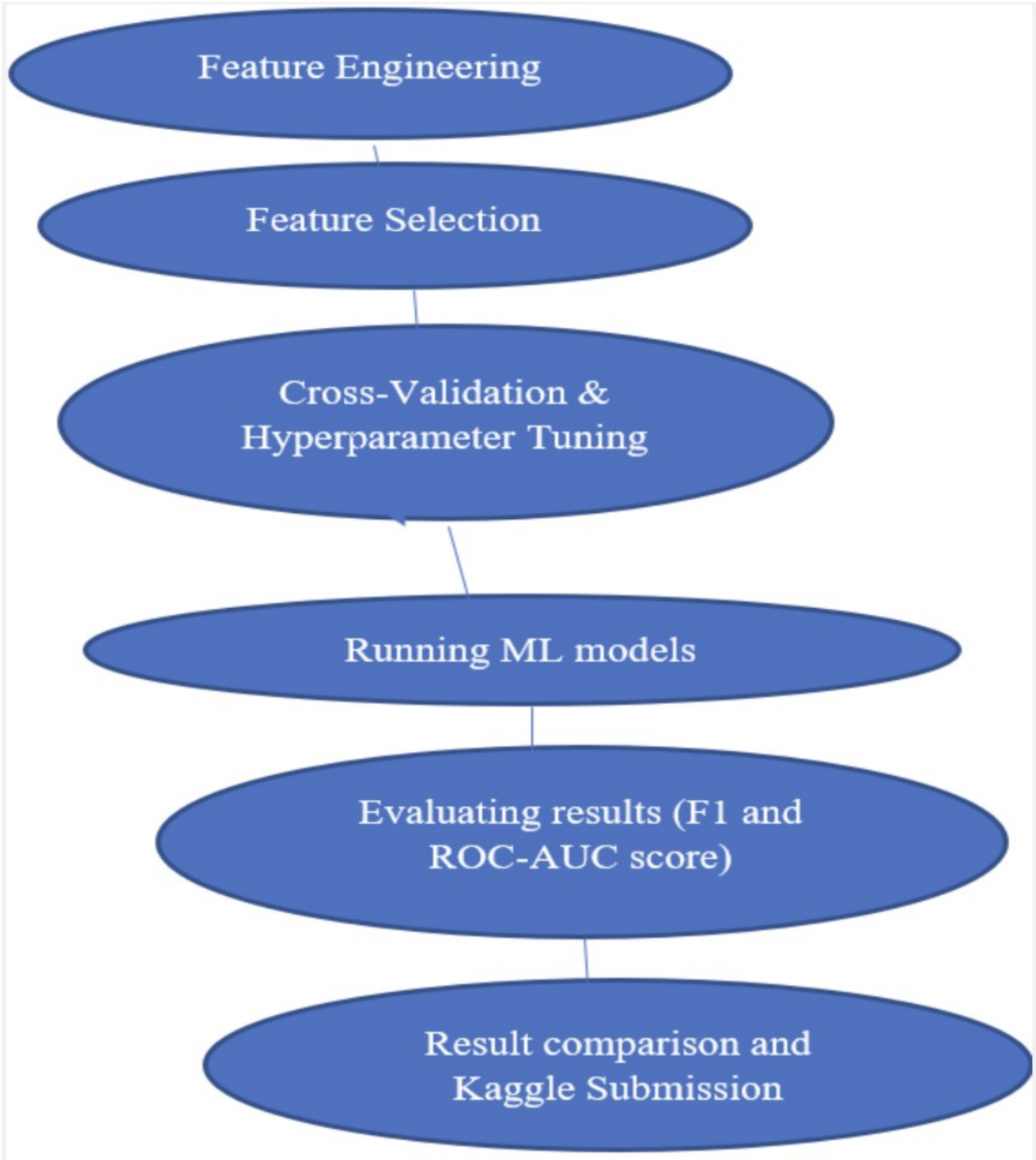
Many people struggle to get loans due to insufficient or non-existent credit histories. And, unfortunately, this population is often taken advantage of by untrustworthy lenders.Home Credit strives to broaden financial inclusion for the unbanked population by providing a positive and safe borrowing experience. In order to make sure this underserved population has a positive loan experience, Home Credit makes use of a variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities.While Home Credit is currently using various statistical and machine learning methods to make these predictions, they're challenging Kagglers to help them unlock the full potential of their data. Doing so will ensure that clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful. In this project, we will be working on the Home Credit Default Risk dataset which is taken and adapted from the dataset hosted on Kaggle. There are eight important tables (or CSV files) present in the dataset which needs to be used for analyzing the results. The tables are as follows:

1. application_{train|test}.csv • This is the primary table split into two files for Train (with TARGET) and Test (without TARGET) • Each row represents a single loan
2. bureau.csv • It consists of data concerning the client's previous credits from other financial institutions. • For every loan there are as many rows as the number of credits the client had in the Credit Bureau before the application date.
3. bureau_balance.csv • It consists of monthly data about the previous credits in the bureau table. • Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length.
4. POS_CASH_balance.csv • It consists of monthly data about the previous point of sale. • Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.
5. credit_card_balance.csv • It consists of the monthly data about previous credit cards customers for Home Credit. • Each row is one month of a credit card balance, and a single credit card can have multiple rows.
6. previous_application.csv • It consists of all previous applications for Home Credit loans for the customers who have loans in the given sample. • There is one row for each previous application related to loans in our data sample.
7. installments_payments.csv • It consists of the data related to payment history for previous loans at Home Credit. • There are two rows one for every payment made while the other for every missed payment.
8. HomeCredit_columns_description.csv • This table (or files) contains the description for each column(or feature) present in each of the above-mentioned data files.

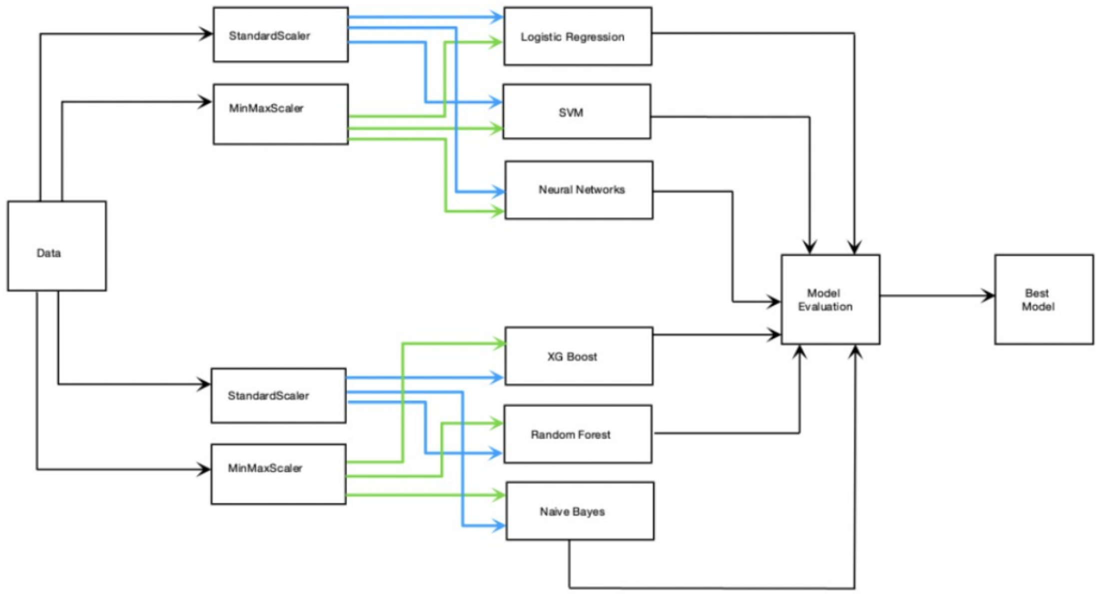


Tasks accomplished in Phase 2

1. Feature Engineering: Combined the input obtained from EDA with domain knowledge to create new features.
2. Feature Selection: Aims at selecting features which are highly correlated to the target variable.
3. Cross Validation & Hyperparameter Tuning: These techniques are implemented to optimize the accuracy. Moreover, Tuning is performed using GridSearchCV and RandomSearchCV.
4. Running ML models: Pipeline for different models is run.
5. Evaluating Results: ROC-AUC score is evaluated for all the ml models.
6. Result Comparison and Kaggle Submission: The metrics primarily used to compare the accuracy results are F1 score and ROC-AUC score while the submission made on Kaggle uses ROC-AUC score as the metric(ROC-AUC is used to due to the imbalance nature of data).


Feature Engineering
Feature Selection
Cross-Validation &
Hyperparameter Tuning
Running ML models
Evaluating results (F1 and
ROC-AUC score)
Result comparison and
Kaggle Submission

Pipelines (Feature Engineering & HyperParameter



To incorporate feature engineering we have combined the input obtained from Exploratory Data Analysis with domain knowledge. We have created new features after merging several important(correlated tables) data files. Newly created features are highly correlated with the target variable. The number input features used for the same are 647. To optimize the obtained results we have implemented hyperparameter tuning. We have implemented it using both GridSearchCV and RandomSearchCV. We have attached code snippets to understand the number of parameters assigned to different ML models. The crossfold validation parameter used for all the models is 5.

The parameters used to obtain result for different models after performing hyperparameter tuning are as follows:

Logistic Regression

best params: {'lr_penalty': 'l2', 'lr_C': 1000}

SVM

best params: {'svm_kernel': 'rbf', 'svmgamma': 0.01, 'svmdegree': 2, 'svm_C': 1}

XGBoost

best params: {'xgb_min_child_weight': 1, 'xgbmax_depth': 7, 'xgb_learning_rate': 0.1}

Random Forest

best params: {'rf_n_estimators': 500, 'rfmax_features': 'log2', 'rfmax_depth': 5, 'rf_criterion': 'entropy'}

Naive Bayes

best params: {'nb__alpha': 1}

XGBoost gives the best result with ROC-AUC score of 0.77 after applying hyperparameter tuning while the validation accuracy for the same is 0.91.

Results and Discussion of Results

	Model	Accuracy	ROC_AUC	F1-Score	Precision	Recall	Log-Loss
0	Logistic Regression	0.919	0.77	0.84	0.483	0.02	2.793
1	SVM	0.921	0.75	0.82	0.423	0.04	2.930
2	Random Forrest	0.920	0.73	0.79	0.389	0.04	2.820
3	XGBOOST	0.930	0.77	0.85	0.480	0.02	2.793
4	Naive Bayes	0.660	0.65	0.62	0.365	0.07	3.820

Conclusion

The main aim of our project is to predict the likelihood of loan repayment for people who are seeking to buy a home.A good credit rating increases the chances of approval for all the above-mentioned scenarios. Still, in many cases, we see that the customers tend to not have a credit rating which makes them less competitive in loan approval. Thereby, in our project, we will address all the factors which are important for an individual to acquire a loan some of which are monthly income, previous loan applications, previous loan history, and loan repayment history among others. The Kaggle competition was started with the hypothesis that machine learning can be used to mine through the large amount of data and features to accurately predict whether a buyer should be approved or not.This phase was an extension of the EDA, preliminary feature engineering and baseline model selection. In this phase we performed feature engineering and hyperparameter tuning. New features were found and tables were merged. In our next and final phase(Phase 3) we will implement a neural network (MultiLayer Perceptron) using Pytorch besides creating a tensor board to carry out experimental analysis of our obtained accuracy. Neural Networks will boost the accuracy and enable us to obtain the most desired accuracy required to predict the ability of the applicant to repay the loan (meet our final goal). Checked correlation, maintain high correlation between newly created features and target variables. Performed hyperparameter tuning using GridSearchCV and RandomSearchCV for the models: Logistic Regression, Naïve Bayes, Support Vector Machine, XGBoost, Random Forest. XGBoost performed well with the ROC-AUC score of 0.77.

Phase 3

```
import torch
import torch.nn as nn
import torch.nn.functional as F

from torchvision import datasets, transforms
from torch.utils.data import DataLoader
from sklearn.metrics import confusion_matrix
from torch.utils.tensorboard import SummaryWriter

import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split

import pandas as pd
import pickle as pkl

from sklearn.metrics import roc_auc_score
```

Loading Data

```
with open("train_final_data.pkl", "rb") as f:
    X_train,y_train = pkl.load(f)
with open("kaggle_test_data.pkl", "rb") as f:
    kaggle_test_data = pkl.load(f)

X_train.shape, y_train.shape

((307506, 644), (307506,))

kaggle_test_data.shape

(48744, 644)

y_train[0]

1.0

from tensorflow.keras.callbacks import TensorBoard
from torch.utils.tensorboard import SummaryWriter
```

Building MLP

```
device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")

input_layer = X_train.shape[0]
```

We are going to have 3 hidden layers

- IN * 256 * 128 * 64 * OUT

```
def MLP(hidden1,hidden2,hidden3):
    model= torch.nn.Sequential(
            torch.nn.LazyLinear(hidden1),
            torch.nn.Sigmoid(),
            torch.nn.Linear(hidden1,hidden2),
            torch.nn.Sigmoid(),
            torch.nn.Linear(hidden2,hidden3),
            torch.nn.Sigmoid(),
            torch.nn.Linear(hidden3,1),
            torch.nn.Sigmoid()
            )


    return model
```

Intitializing the model

```
device=torch.device('cpu')

model = MLP(256,128, 64)


model.to(device=device)
```

```
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/torch/nn/
modules/lazy.py:178: UserWarning: Lazy modules are a new feature under
heavy development so changes to the API or functionality can happen at
any moment.
  warnings.warn('Lazy modules are a new feature under heavy
development '
```

```
Sequential(
  (0): LazyLinear(in_features=0, out_features=256, bias=True)
  (1): Sigmoid()
  (2): Linear(in_features=256, out_features=128, bias=True)
  (3): Sigmoid()
  (4): Linear(in_features=128, out_features=64, bias=True)
  (5): Sigmoid()
  (6): Linear(in_features=64, out_features=1, bias=True)
  (7): Sigmoid()
)
```

```
model
```

```
Sequential(
  (0): LazyLinear(in_features=0, out_features=256, bias=True)
  (1): Sigmoid()
  (2): Linear(in_features=256, out_features=128, bias=True)
```

```
    (3): Sigmoid()
    (4): Linear(in_features=128, out_features=64, bias=True)
    (5): Sigmoid()
    (6): Linear(in_features=64, out_features=1, bias=True)
    (7): Sigmoid()
)
```

Preparing Dataset for Neural Network

Splitting data

```
X_train,X_cv,y_train,y_cv = train_test_split(X_train,
y_train,stratify=y_train,test_size=0.20)
```

```
X_train.shape, y_train.shape
```

```
((246004, 644), (246004,))
```

```
X_cv.shape, y_cv.shape
```

```
((61502, 644), (61502,))
```

Transforming data to tensors

```
X_train_transformed = torch.FloatTensor(X_train).to(device=device)
y_train_transformed =
torch.FloatTensor(y_train.values).to(device=device)
```

Putting Train data in DataLoader for Training

```
train_data = []
for ind in range(len(X_train_transformed)):
    train_data.append([X_train_transformed[ind],
y_train_transformed[ind]])

print(len(train_data))
```

```
246004
```

```
train_loader = torch.utils.data.DataLoader(train_data, shuffle=True,
batch_size=500)
```

Initializing loss and optimization algotithms

```
criterion = nn.BCEWithLogitsLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

Training Model

```
%reload_ext tensorboard
writer = SummaryWriter('runs/working_directory')

prev_loss = float("-inf")
EPOCHS=500
```

```
for ind in range(EPOCHS):
    running_loss=0.0

    for b_iter, (x_train_data, y_train_data) in
enumerate(train_loader):
        b_iter += 1
        y_pred = model(x_train_data)

        loss = criterion(y_pred,
y_train_data.reshape(y_pred.shape[0],1))


        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    print(f"Epoch {ind}, loss: {loss.item()}")

    if (abs(prev_loss - loss.item()) < 0.000001):
        break

    prev_loss = loss.item()

    # to add graph for training loss into tensorboard
    writer.add_scalar('Training loss', running_loss, ind+1)

print("*******Finished Trianing*******")
```

```
Epoch 0, loss: 0.7062119841575623
Epoch 1, loss: 0.6961959600448608
Epoch 2, loss: 0.6951020956039429
Epoch 3, loss: 0.6959899663925171
Epoch 4, loss: 0.6953731775283813
Epoch 5, loss: 0.6949716806411743
Epoch 6, loss: 0.6939195394515991
Epoch 7, loss: 0.6931484937667847
Epoch 8, loss: 0.6943195462226868
Epoch 9, loss: 0.6941933035850525
Epoch 10, loss: 0.6940888166427612
Epoch 11, loss: 0.6940033435821533
Epoch 12, loss: 0.6939330697059631
Epoch 13, loss: 0.6935098767280579
Epoch 14, loss: 0.6934830546379089
Epoch 15, loss: 0.6937738656997681
Epoch 16, loss: 0.6937345266342163
Epoch 17, loss: 0.6936987638473511
```

```
Epoch 18, loss: 0.6936672925949097
Epoch 19, loss: 0.6936390995979309
Epoch 20, loss: 0.6936138868331909
Epoch 21, loss: 0.6935909986495972
Epoch 22, loss: 0.6935698390007019
Epoch 23, loss: 0.693550705909729
Epoch 24, loss: 0.693533182144165
Epoch 25, loss: 0.6935176253318787
Epoch 26, loss: 0.6933247447013855
Epoch 27, loss: 0.693489134311676
Epoch 28, loss: 0.6934759616851807
Epoch 29, loss: 0.6934643387794495
Epoch 30, loss: 0.6934530138969421
Epoch 31, loss: 0.6932949423789978
Epoch 32, loss: 0.6934328079223633
Epoch 33, loss: 0.693423867225647
Epoch 34, loss: 0.6934155225753784
Epoch 35, loss: 0.6932771801948547
Epoch 36, loss: 0.6933998465538025
Epoch 37, loss: 0.693269670009613
Epoch 38, loss: 0.6933858394622803
Epoch 39, loss: 0.6933794617652893
Epoch 40, loss: 0.6933733224868774
Epoch 41, loss: 0.6933677792549133
Epoch 42, loss: 0.6932546496391296
Epoch 43, loss: 0.6933568716049194
Epoch 44, loss: 0.6933515071868896
Epoch 45, loss: 0.6933469176292419
Epoch 46, loss: 0.6933422684669495
Epoch 47, loss: 0.6933376789093018
Epoch 48, loss: 0.6932407021522522
Epoch 49, loss: 0.6933297514915466
Epoch 50, loss: 0.6932364702224731
Epoch 51, loss: 0.6933221817016602
Epoch 52, loss: 0.6933186650276184
Epoch 53, loss: 0.6933151483535767
Epoch 54, loss: 0.6932294368743896
Epoch 55, loss: 0.6933087706565857
Epoch 56, loss: 0.693226158618927
Epoch 57, loss: 0.6933026313781738
Epoch 58, loss: 0.6932997703552246
Epoch 59, loss: 0.6932969689369202
Epoch 60, loss: 0.6932945251464844
Epoch 61, loss: 0.6932917833328247
Epoch 62, loss: 0.6932892799377441
Epoch 63, loss: 0.6932170987129211
Epoch 64, loss: 0.6932845711708069
Epoch 65, loss: 0.6932148337364197
Epoch 66, loss: 0.6932802796363831
Epoch 67, loss: 0.6932781934738159
```

```
Epoch 68, loss: 0.693211555480957
Epoch 69, loss: 0.693210780620575
*******Finished Trianing*******
```

Testing model on test data

Transforming cross validation data to Tensors

```
X_cv_transformed = torch.FloatTensor(X_cv).to(device=device)
```

Getting prediction on Cross Validation Data

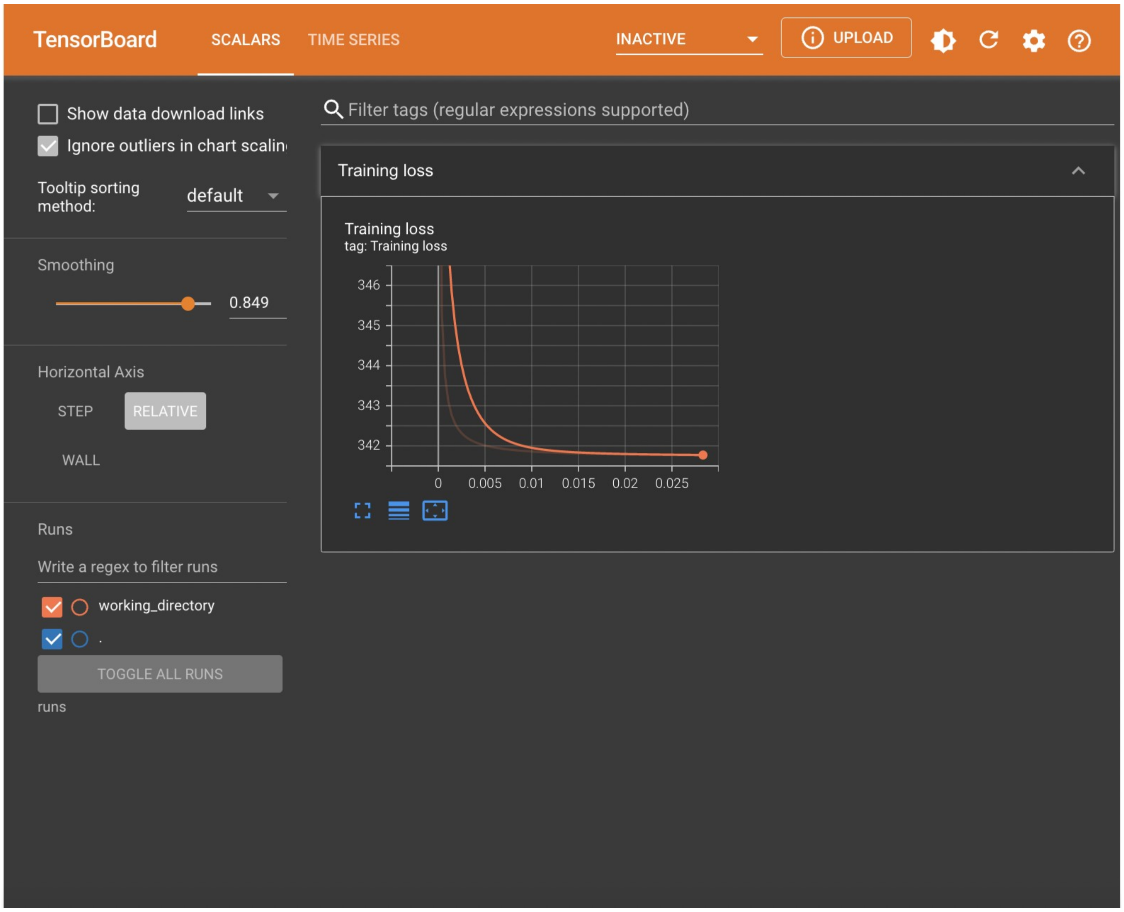
```
preds = model(X_cv_transformed)
predictions = torch.max(preds.data, 1)[1]
```

Tensorboard

```
%tensorboard --logdir=runs
```

```
Reusing TensorBoard on port 6006 (pid 3891), started 4:33:14 ago. (Use
'!kill 3891' to kill it.)
```

```
<IPython.core.display.HTML object>
```



Evaluating model

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import log_loss
```

```
model_score.loc[len(model_score)] = ["MLP (in*128*64*out)",
                                     np.round(accuracy_score(y_cv,
predictions), 3),
                                      np.round(roc_auc_score(y_cv,
preds.cpu().detach().numpy()), 3),
                                      np.round(f1_score(y_cv,
predictions,average='weighted'), 3),
                                      np.round(precision_score(y_cv,
predictions), 3),
                                      np.round(recall_score(y_cv,
predictions), 3),
```

```
                                    np.round(log_loss(y_cv, 
predictions), 3)
                                   ]
```

```
/Users/parthkapil/miniforge3/lib/python3.9/site-packages/sklearn/
metrics/_classification.py:1318: UndefinedMetricWarning: Precision is 
ill-defined and being set to 0.0 due to no predicted samples. Use 
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
writer.flush()
writer.close()
```

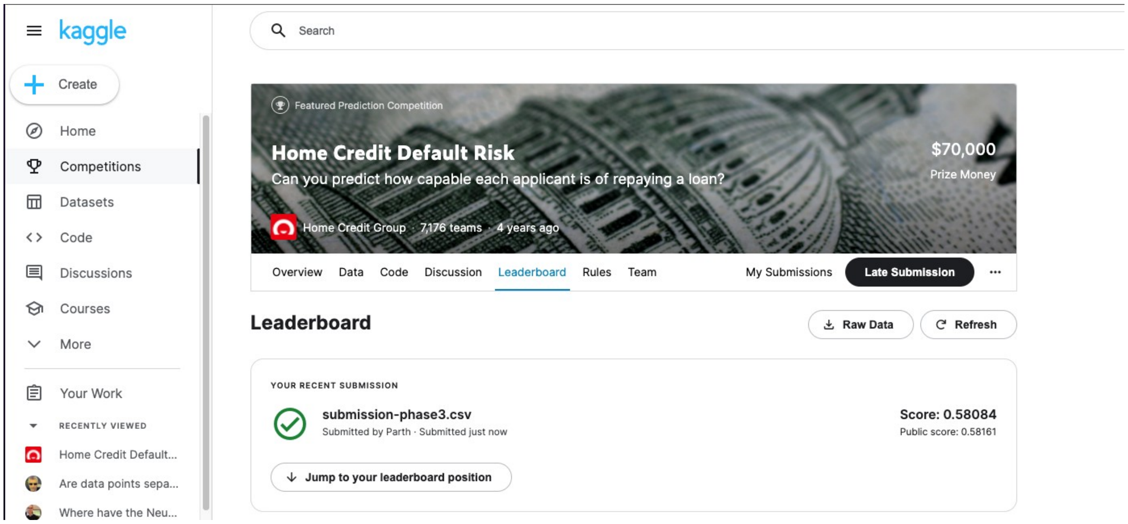
```
model_score
```

```
                 Model  CV_Accuracy  CV_ROC_AUC  CV_F1-Score  
CV_Precision  \
0  Logistic Regression        0.919       0.770        0.840   
0.483   
1                  SVM        0.921       0.750        0.820   
0.423   
2       Random Forrest        0.920       0.730        0.790   
0.389   
3              XGBoost        0.930       0.770        0.850   
0.480   
4          Naive Bayes        0.660       0.650        0.620   
0.365   
5  MLP (in*128*64*out)        0.919       0.507        0.881   
0.000   

   CV_Recall  CV_Log-Loss  
0       0.02        2.793  
1       0.04        2.930  
2       0.04        2.820  
3       0.02        2.793  
4       0.07        3.820  
5       0.00        2.788  
```

Submitting on Kaggle

```
kaggle_test_data_transformed = 
torch.FloatTensor(kaggle_test_data).to(device=device)

test_class_scores = model(kaggle_test_data_transformed)
test_class_scores = test_class_scores.cpu().detach().numpy()

# Submission dataframe
submit_df = datasets["application_test"][['SK_ID_CURR']]
submit_df['TARGET'] = test_class_scores
```

```
submit_df.head()
submit_df.to_csv("submission-phase3.csv",index=False)
```



Write-Up

Abstract

Credit Rating is one of the most important parameters for availing home loan.In our project, we will address all the factors to accurately predict the ability of an applicant to repay loan.To do so our team intends towards building different machine learning models and choose the best of the lot. The data used for building these ML models is referred from the competition project, Home Credit Default Risk on Kaggle.

In this phase(Phase 3) we built a multi-layer perception (MLP) classification model using Pytorch and used Tensorboard to evaluate and monitor real-time training results.To achieve this,we have implemented Feedforward MLP with three hidden layers of 256,128 and 64 neurons each. Data(present in the form of NumPy array) is converted to tensors using PyTorch FloatTensor. After this we put the train data into DataLoader for training the dataset.Further we initialized our optimization algorithm using Stochastic Gradient Descent(SGD) while loss algorithm is initialized using Log Loss. We trained our model for 500 epochs and tested our model on Cross Validation data.To visualize the real-time training results we have used TensorBoard to monitor the loss (LogLoss) and accuracy achieved after each epoch. The ROC_AUC score achieved for the MLP model is 0.61 while the validation accuracy achieved for the same is 0.919.The public score obtained for Kaggle submission for the MLP model is 0.58161.

After evaluating the results we can see that XGBoost has the highest ROC_AUC score of 0.77 while the highest Kaggle submission score achieved till now is 0.76683 which is for XGBoost. This shows that XGBoost gives the best result for predicting the ability of the applicant to repay loan with ROC_AUC score of 0.76683 and validation accuracy of 0.93.

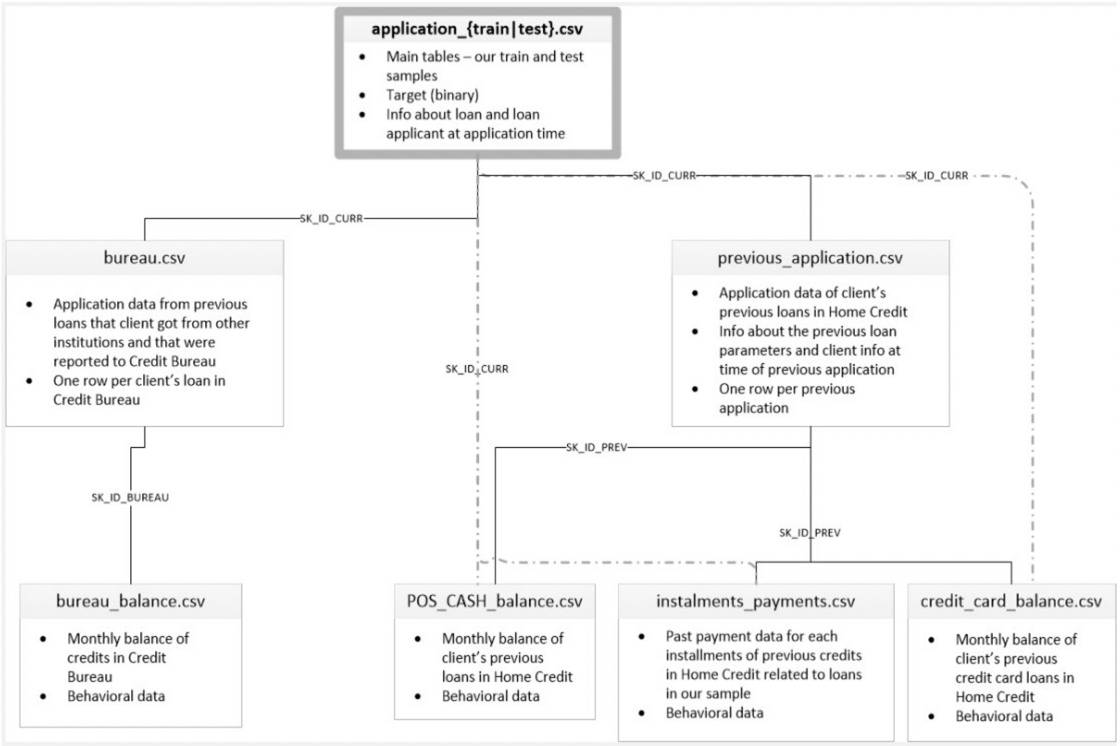
Project Description

Data Description

Many people struggle to get loans due to insufficient or non-existent credit histories. And, unfortunately, this population is often taken advantage of by untrustworthy lenders.Home Credit strives to broaden financial inclusion for the unbanked population by providing a positive and safe borrowing experience. In order to make sure this underserved population has a positive loan experience, Home Credit makes use of a variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities.While Home Credit is currently using various statistical and machine learning methods to make these predictions, they're challenging Kagglers to help them unlock the full potential of their data. Doing so will ensure that clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful. In this project, we will be working on the Home Credit Default Risk dataset which is taken and adapted from the dataset hosted on Kaggle.

There are eight important tables (or CSV files) present in the dataset which needs to be used for analyzing the results. The tables are as follows:

1. application_{train|test}.csv • This is the primary table split into two files for Train (with TARGET) and Test (without TARGET) • Each row represents a single loan
2. bureau.csv • It consists of data concerning the client's previous credits from other financial institutions. • For every loan there are as many rows as the number of credits the client had in the Credit Bureau before the application date.
3. bureau_balance.csv • It consists of monthly data about the previous credits in the bureau table. • Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length.
4. POS_CASH_balance.csv • It consists of monthly data about the previous point of sale. • Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.
5. credit_card_balance.csv • It consists of the monthly data about previous credit card customers for Home Credit. • Each row is one month of a credit card balance, and a single credit card can have multiple rows.
6. previous_application.csv • It consists of all previous applications for Home Credit loans for the customers who have loans in the given sample. • There is one row for each previous application related to loans in our data sample.
7. installments_payments.csv • It consists of the data related to payment history for previous loans at Home Credit. • There are two rows one for every payment made while the other for every missed payment.
8. HomeCredit_columns_description.csv • This table (or files) contains the description for each column(or feature) present in each of the above-mentioned data files.

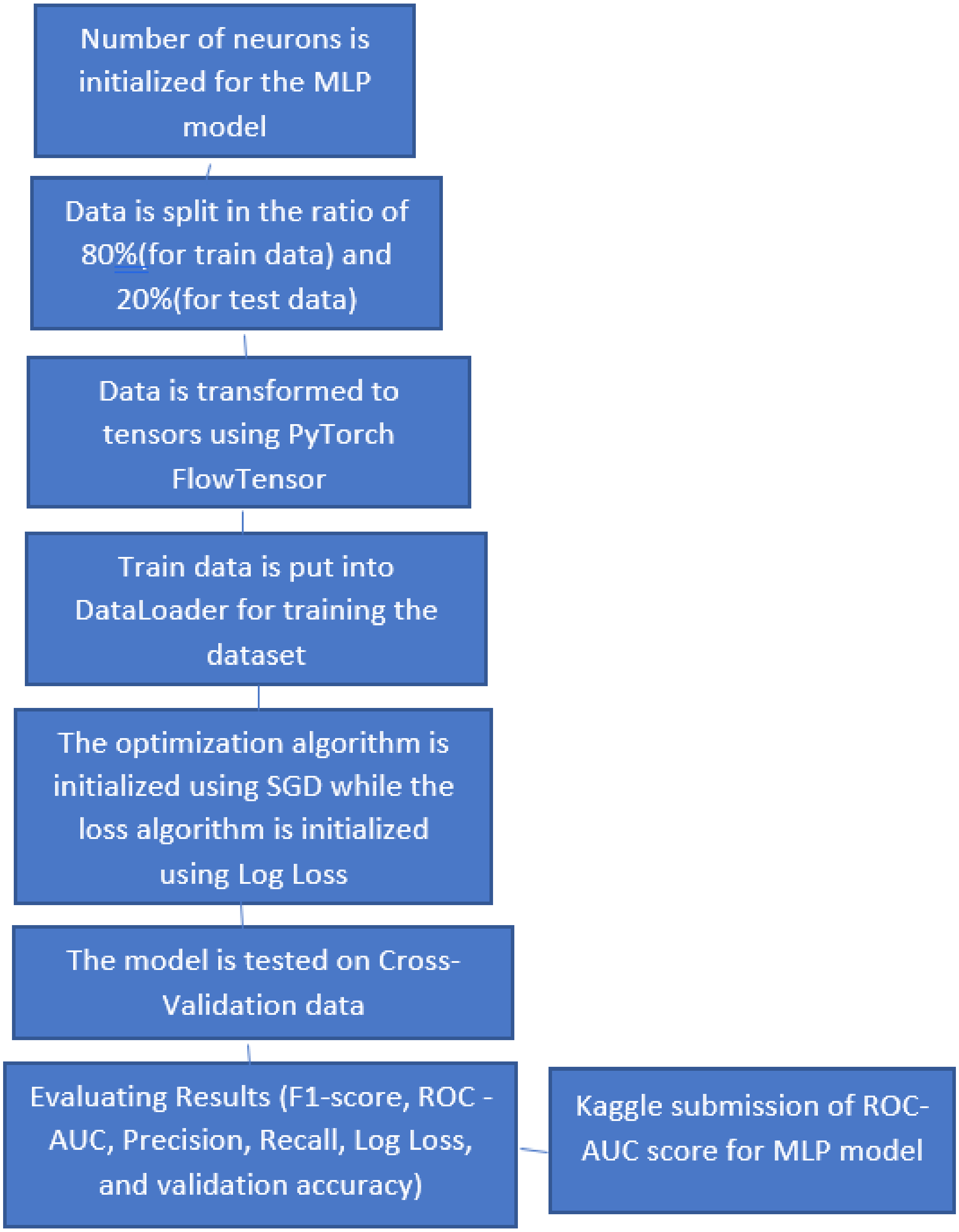


Tasks tackled in Phase 3

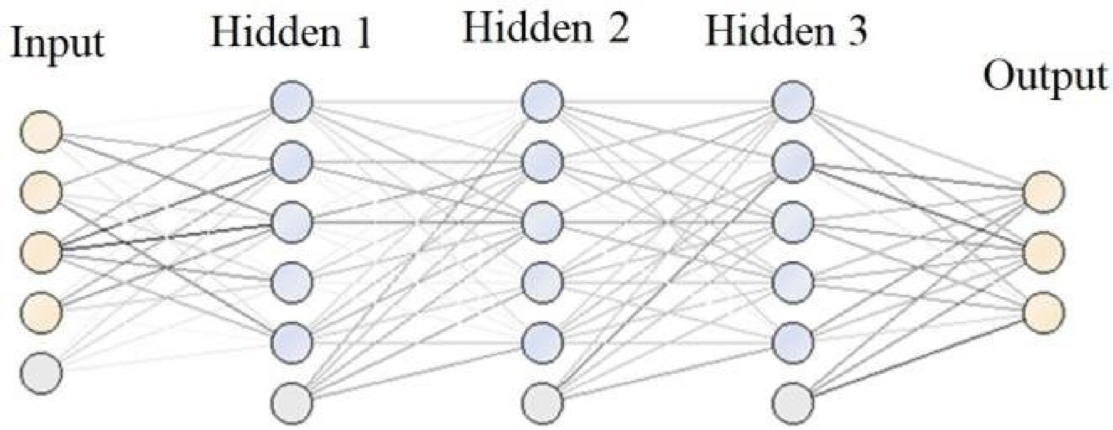
1. Building Multilayer Perceptron(MLP) model using PyTorch
- Implemented feed forward MLP with three layers having 256,128 and 64 neurons respectively.
- Optimization algorithm is initialized using Stochastic Gradient Descent(SGD) while loss algorithm is initialized using Log Loss
- Trained model for 500 epochs
- Testing is done on Cross Validation data
- Evaluated performance using metrics like F1-score,Recall,Precision,ROC AUC and Log Loss

1. Creating TensorBoard to monitor real time training results
2. Kaggle submission: Made a submission for the ROC AUC score for MLP model
3. Selected the model having highest ROC AUC score till now(From Phase1 up till Phase3)

WorkFlow Diagram for building MultiLayer Perceptron Model



Neural Network



In this phase, we have used Neural Networks to calculate the ability of an applicant to repay loan.For this we have used PyTorch to implement Neural Networks. We have implemented a Feedforward MultiLayer Perceptron model with three layers having 256,128 and 64 neurons each. 80 % of the data is used for training while the remaining 20% is used for testing. To train the data we have converted it to tensors using PyTorch FloatTensor while the optimization algorithm and loss function is initialized using Stochastic Gradient Descent(SGD) function and Log Loss respectively. Testing is done on Cross Validation data while the model is trained for 500 epochs.

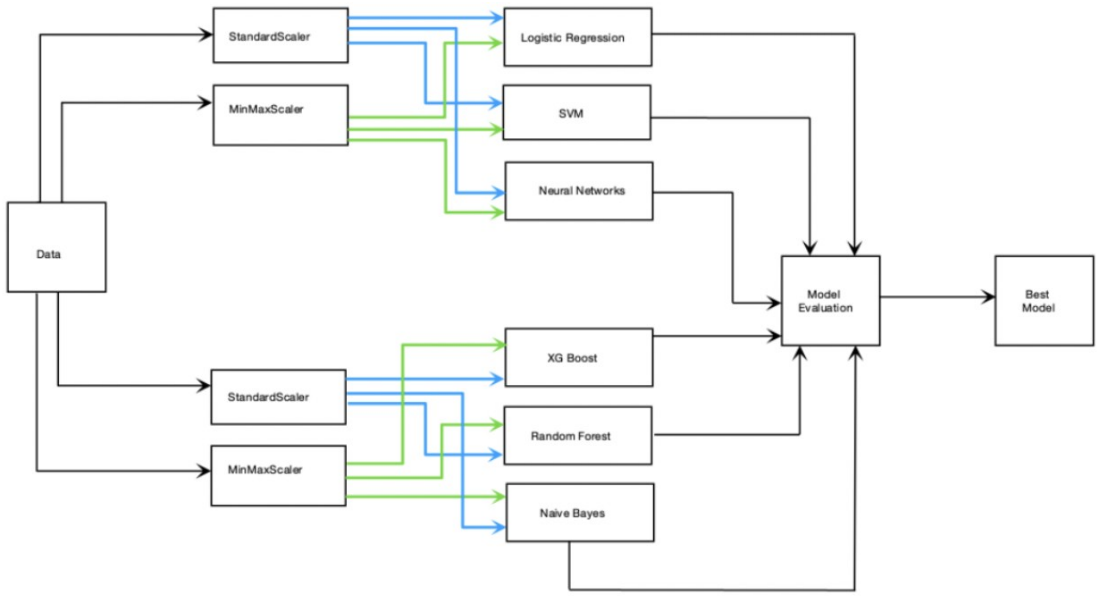
The validation accuracy and ROC-AUC score obtained for the MLP model is 0.919 and 0.61 respectively while the public score for Kaggle submission is 0.58084.

Leakage

Data leakage takes place when the training data is used before training the data. To elaborate, if we perform imputations using feature engineering on test data way before the data is trained then it is a case of data leakage.

In our project we have performed one hot encoding and imputed the missing values present in the training data. Additionally we have fitted StandardScaler and MinMaxScaler using train data after which the models are run using train data. Following which the models are tested on the test data. Therefore we can see that we don't have any data leakage while implementing the models.

Modeling Pipelines



The main aim of pipeline is to prepare the data for prediction. From the above diagram we can observe that our pipeline consists of custom dataframe selector which selects the number of input features. The number of input features used by us is 647.

Secondly, the pipeline has an imputer for imputing the missing values(which is used in Feature engineering mentioned in the above codes) besides having a minmax scaler to bring all the values on the same scale. To incorporate feature engineering we have combined the input obtained from Exploratory Data Analysis with domain knowledge. We have created new features after merging several important(correlated tables) data files. Newly created features are highly correlated with the target variable. Moreover, to optimize the obtained results we have implemented hyperparameter tuning. We have implemented it using both GridSearchCV and RandomSearchCV.

All the above mentioned concepts are implemented on different machine learning models namely RandomForest,Logistic Regression,Naive Bayes,XGBoost among others. Following this these concepts are extended by implementing Feedforward MultiLayer Perceptron model using PyTorch (having three layers consisting of 256,128 and 64 neurons each). After this the results are evaluated based on several metrics like ROC-AUC, F1 score, Log Loss etc. with ROC-AUC being the metric (due to imbalanced nature of data). Finally the model having the best ROC-AUC score which is XGBoost in this case (having ROC-AUC 0.77) is selected as the model to predict the ability of an applicant to repay loan.

Results and Discussion of Results

	Model	CV_Accuracy	CV_ROC_AUC	CV_F1-Score	CV_Precision	CV_Recall	CV_Log-Loss
0	Logistic Regression	0.919	0.770	0.840	0.483	0.02	2.793
1	SVM	0.921	0.750	0.820	0.423	0.04	2.930
2	Random Forrest	0.920	0.730	0.790	0.389	0.04	2.820
3	XGBoost	0.930	0.770	0.850	0.480	0.02	2.793
4	Naive Bayes	0.660	0.650	0.620	0.365	0.07	3.820
5	MLP (in*128*64*out)	0.919	0.507	0.881	0.000	0.00	2.788

After implementing concepts like Exploratory Data Analysis, Feature engineering and hyperparameter tuning in the previous phases along with running the model pipeline and evaluating the results for different ML models like Logistic Regression,SVM,Random Forest,XGBoost and Naive Bayes, in the final phase we extended the concepts by implementing MultiLayer Perceptron model (or Neural Networks).

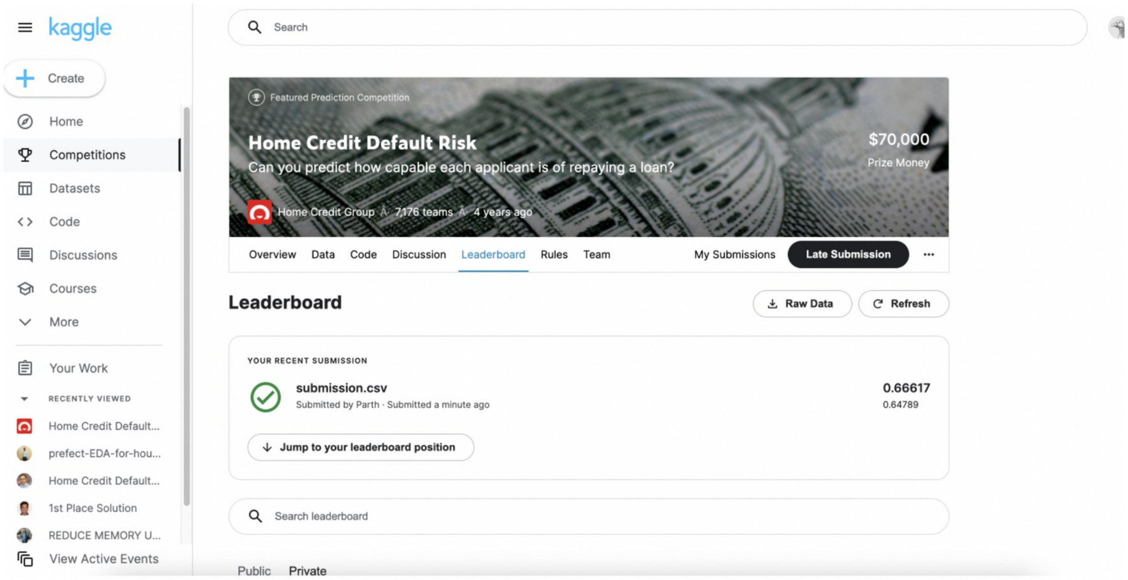
For evaluating the result several metrics like ROC-AUC score,F1-score,Log loss among others have been used.In the very beginning we observed that the given data is highly imbalanced in nature, thereby in this case ROC-AUC(and F1-score) performs the best. From the above table we can see that the ROC-AUC score and validation accuracy obtained for MLP is 0.61 and 0.919 respectively.

We also observe that XGBoost gives the best performance with ROC-AUC score of 0.77 while the validation accuracy for the same is 0.93. After experimental analysis Kaggle submission is made.

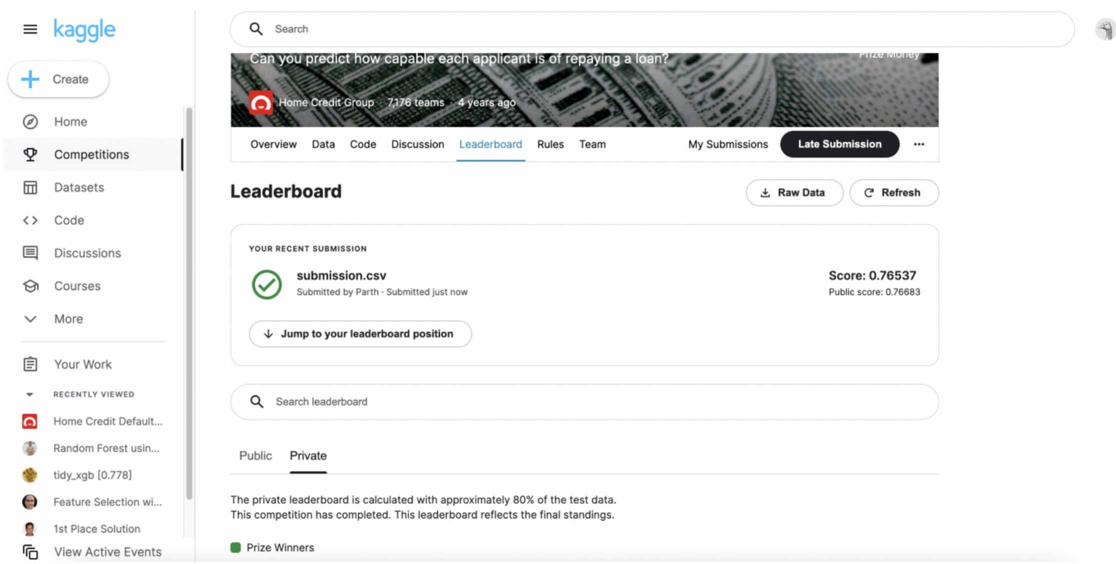
Highest public score achieved for the submission is 0.76683 which is for XGBoost model.

Here are some Kaggle Submission to provide more clarity in the public scores obtained across all the phases.

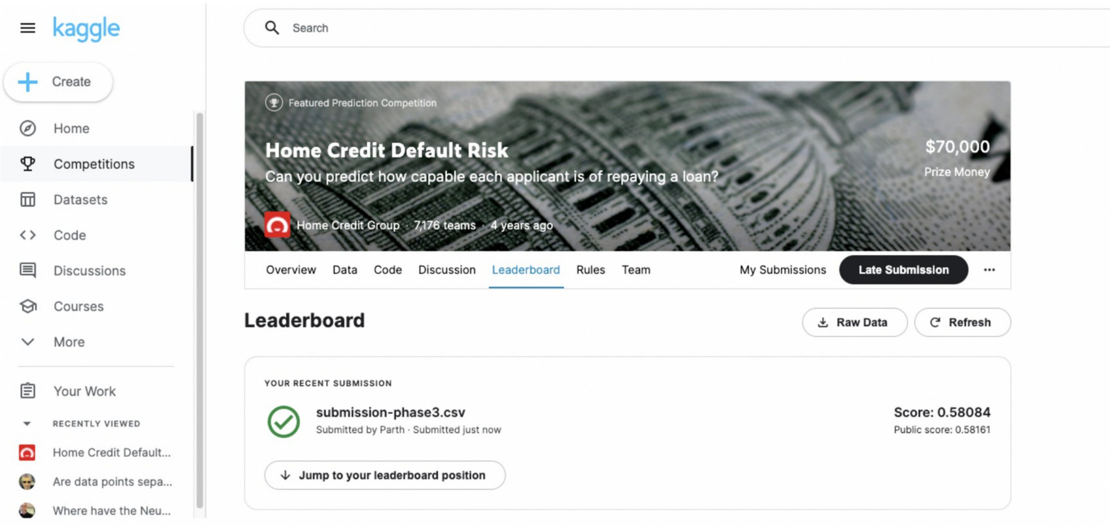
Kaggle submission for selected baseline model(Phase 1). Logistic Regression is selected as the baseline model.



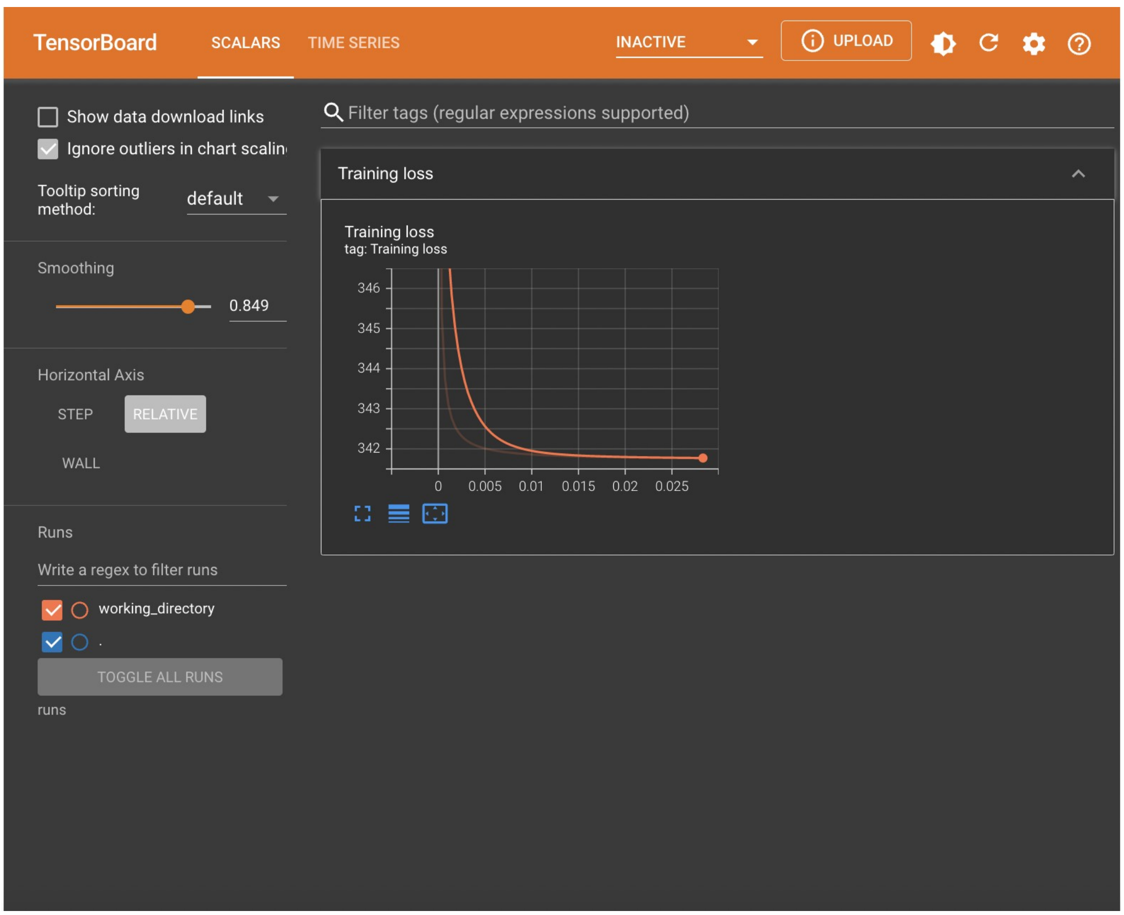
Kaggle Submission for the highest accuracy achieved by XGBoost after hyperparameter tuning and feature engineering.(Phase 2)



Kaggle Submission for the ROC-AUC score obtained for Neural Networks. (Phase 3)



Adding on we have also implemented TensorBoard to monitor the loss (LogLoss) and accuracy achieved after each epoch.



Conclusion

The main aim of our project is to predict the likelihood of loan repayment for people who are seeking to buy a home.A good credit rating increases the chances of approval for all the above-mentioned scenarios. Still, in many cases, we see that the customers tend to not have a credit rating which makes them less competitive in loan approval. Thereby, in our project, we will address all the factors which are important for an individual to acquire a loan some of which are monthly income, previous loan applications, previous loan history, and loan repayment history among others.

The Kaggle competition was started with the hypothesis that machine learning can be used to mine through the large amount of data and features to accurately predict whether a buyer should be approved or not.

We started the project with exploratory data analysis, preliminary feature engineering and baseline model selection(of the previous phase) following which we implemented concepts like robust feature engineering and hyperparameter tuning to boost the accuracy of machine learning models.Several models like Naive Bayes,Logistic Regression,XGBoost were trained.Towards the end of the project (Phase 3) we implemented Feedforward MultiLayer Perceptron model(Neural Networks) using PyTorch.The MLP model has three layers consisting of 256,128 and 64 neurons each.

For evaluating the results we used different metrics score like F1-score,ROC-AUC score,Precision,Recall and Log-Loss, yet for the Kggle submission we have considered ROC-AUC for reason being that the data present is highly imbalanced.After evaluation we observed that XGBoost performs the best with ROC-AUC score of 0.77 and validation accuracy of 0.93 while the MLP model was not at par with other (traditional machine learning) models with ROC-AUC score of 0.61.

Therefore we infer that XGBoost is our best model(predicts with highest accuracy) to predict the ability of an applicant to repay loan with Kaggle public score as 0.58161.