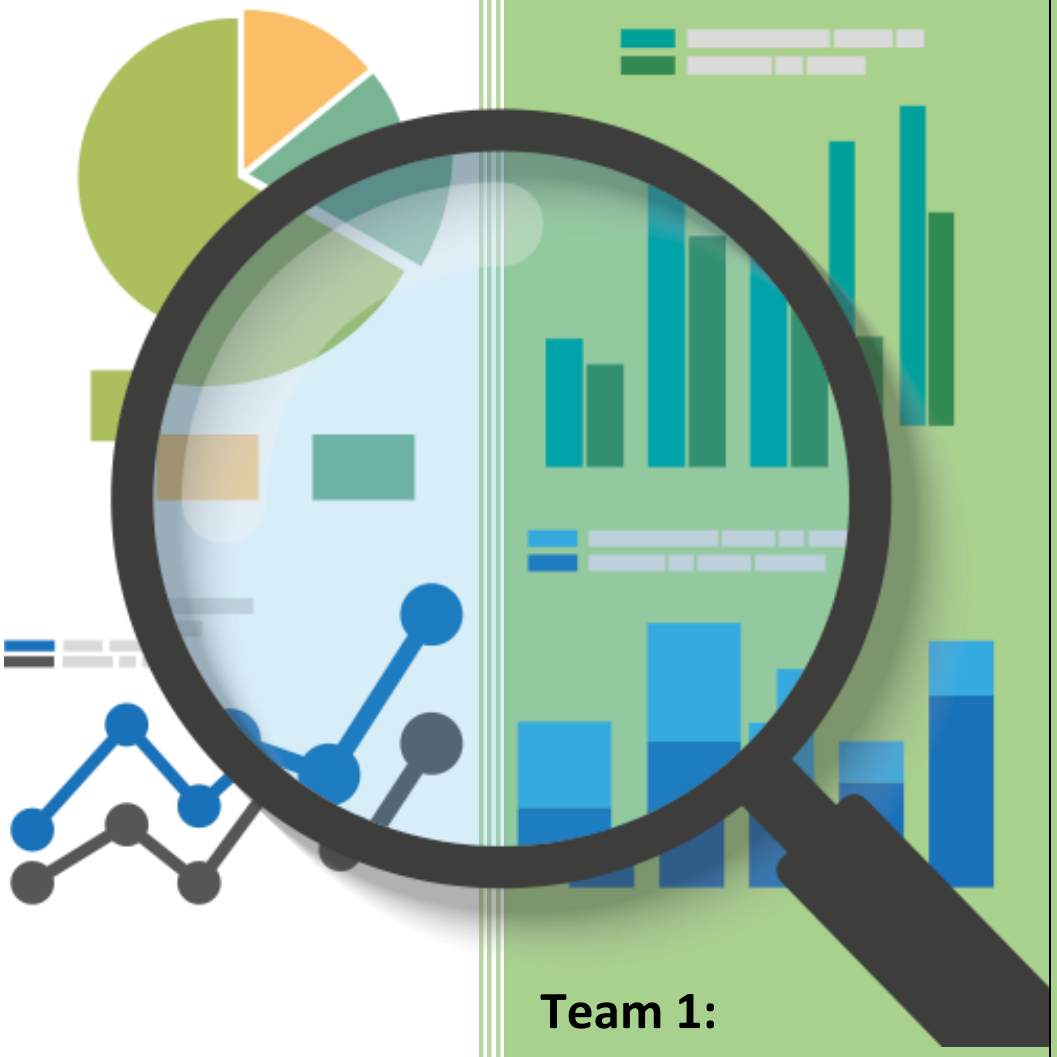
2022

# Home Credit Default Risk Prediction




**Team 1:**

1. Abhishek Raghorte
2. Abubakar Shaikh
3. Aditya More
4. Akshay Sutar
5. Amey Koli

# Table of Contents

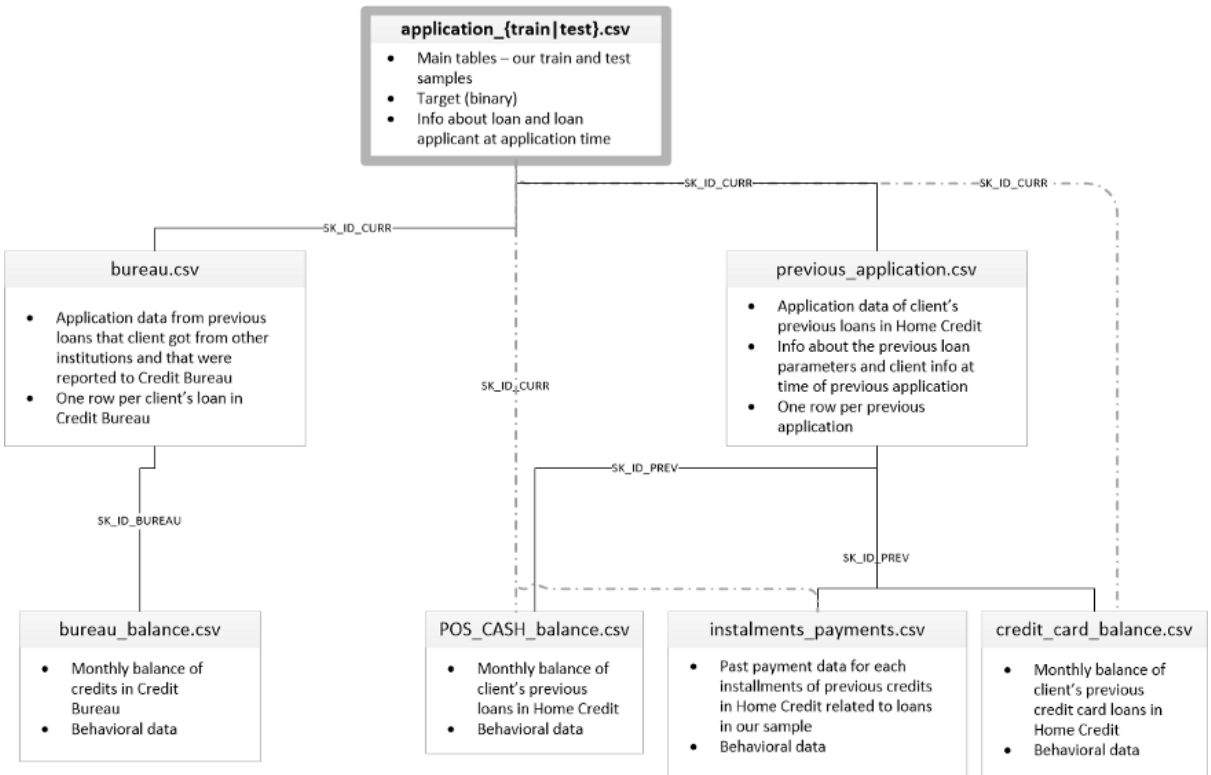
# 1. Problem Statement:

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**.**

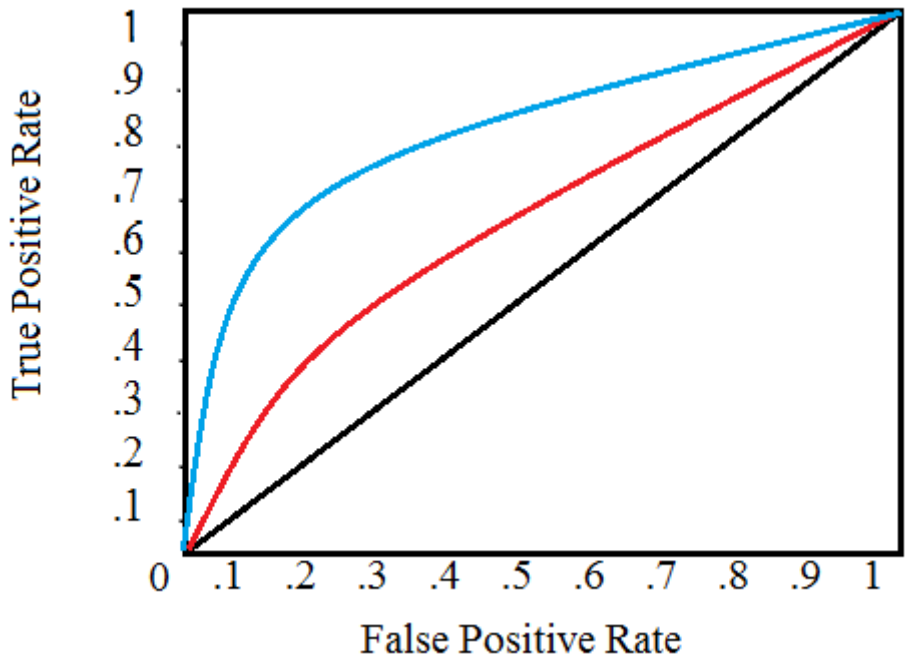
# 2. Dataset Description:

The dataset provided contains lots of details about the borrower. It contains numerical, categorical and character data. It is segregated into multiple relational tables, which contain applicants' static data such as their gender, age, number of family members, occupation, and other related fields, applicant's previous credit history obtained from the credit bureau department, and the applicant's past credit history within the Home Credit Group itself. The dataset is an imbalanced dataset, where the Negative class dominates the Positive class, as there are only a few number of defaulters among all the applicants.



# 3. Evaluation Metric:

Once we have a grasp of the data (reading through the column descriptions helps immensely), we need to understand the metric by which our submission is judged. In this case, it is a common classification metric known as the Receiver Operating Characteristic Area Under the Curve (ROC AUC, also sometimes called AUROC). The ROC AUC may sound intimidating, but it is relatively straightforward once you can get your head around the two individual concepts.



# 4. Importing Libraries:

We will first start with importing relevant Python Libraries.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import lightgbm as lgb
import xgboost as xgb
```

# 5. Exploratory Data Analysis:

## A. Basic Statistics:

We will load each table, and print their shapes and a few of their rows.

1. Loading the Dataset

```
application_train = pd.read_csv('D:/case_study Exam/application_train.csv')
application_test= pd.read_csv('D:/case_study Exam/application_test.csv')
bureau = pd.read_csv('D:/case_study Exam/bureau.csv')
bureau_balance = pd.read_csv('D:/case_study Exam/bureau_balance.csv')
POS_CASH_balance = pd.read_csv('D:/case_study Exam/POS_CASH_balance.csv')
credit_card_balance = pd.read_csv('D:/case_study Exam/credit_card_balance.csv')
previous_application = pd.read_csv('D:/case_study Exam/previous_application.csv')
installments_payments = pd.read_csv('D:/case_study Exam/installments_payments.csv')
```

2. Checking shape of Main Dataframes:

```
print('------------main------------')
print(application_train.shape)
print(application_test.shape)
print('      ')
print('------------others------------')
print(POS_CASH_balance.shape)
print(bureau.shape)
print( bureau_balance.shape)
print(previous_application.shape)
print( installments_payments.shape)
print(credit_card_balance.shape)
```

```
------------main------------
(307511, 122)
(48744, 121)

------------others------------
(10001358, 8)
(1716428, 17)
(27299925, 3)
(1670214, 37)
(13605401, 8)
(3840312, 23)
```

## 3. Statistics of Main tables:

The training data has 307511 observations (each one a separate loan) and 122 features (variables) including the TARGET (the label we want to predict).

```
application_train.head()
```

Python

| | SK_ID_CURR | TARGET | NAME_CONTRACT_TYPE | CODE_GENDER | FLAG_OWN_CAR | FLAG_OWN_REALTY | CNT_CHILDREN | AMT_INCOME_TOTAL | AMT_CREDIT | AMT_ANNUITY | ... | FLAG_DOCUMENT_18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 100002 | 1 | Cash loans | M | N | Y | 0 | 202500.0 | 406597.5 | 24700.5 | ... | 0 |
| 1 | 100003 | 0 | Cash loans | F | N | N | 0 | 270000.0 | 1293502.5 | 35698.5 | ... | 0 |
| 2 | 100004 | 0 | Revolving loans | M | Y | Y | 0 | 67500.0 | 135000.0 | 6750.0 | ... | 0 |
| 3 | 100006 | 0 | Cash loans | F | N | Y | 0 | 135000.0 | 312682.5 | 29686.5 | ... | 0 |
| 4 | 100007 | 0 | Cash loans | M | N | Y | 0 | 121500.0 | 513000.0 | 21865.5 | ... | 0 |

5 rows × 122 columns

```
application_train.describe()
```

| | SK_ID_CURR | TARGET | CNT_CHILDREN | AMT_INCOME_TOTAL | AMT_CREDIT | AMT_ANNUITY | AMT_GOODS_PRICE | REGION_POPULATION_RELATIVE | DAYS_BIRTH | DAYS_EMPLOYED |
|---|---|---|---|---|---|---|---|---|---|---|
| count | 307511.000000 | 307511.000000 | 307511.000000 | 3.075110e+05 | 3.075110e+05 | 307499.000000 | 3.072330e+05 | 307511.000000 | 307511.000000 | 307511.000000 |
| mean | 278180.518577 | 0.080729 | 0.417052 | 1.687979e+05 | 5.990260e+05 | 27108.573909 | 5.383962e+05 | 0.020868 | -16036.995067 | 63815.045904 |
| std | 102790.175348 | 0.272419 | 0.722121 | 2.371231e+05 | 4.024908e+05 | 14493.737315 | 3.694465e+05 | 0.013831 | 4363.988632 | 141275.766519 |
| min | 100002.000000 | 0.000000 | 0.000000 | 2.565000e+04 | 4.500000e+04 | 1615.500000 | 4.050000e+04 | 0.000290 | -25229.000000 | -17912.000000 |
| 25% | 189145.500000 | 0.000000 | 0.000000 | 1.125000e+05 | 2.700000e+05 | 16524.000000 | 2.385000e+05 | 0.010006 | -19682.000000 | -2760.000000 |
| 50% | 278202.000000 | 0.000000 | 0.000000 | 1.471500e+05 | 5.135310e+05 | 24903.000000 | 4.500000e+05 | 0.018850 | -15750.000000 | -1213.000000 |
| 75% | 367142.500000 | 0.000000 | 1.000000 | 2.025000e+05 | 8.086500e+05 | 34596.000000 | 6.795000e+05 | 0.028663 | -12413.000000 | -289.000000 |
| max | 456255.000000 | 1.000000 | 19.000000 | 1.170000e+08 | 4.050000e+06 | 258025.500000 | 4.050000e+06 | 0.072508 | -7489.000000 | 365243.000000 |

Training Dataset

```
application_test.head()
```

Python

| | SK_ID_CURR | NAME_CONTRACT_TYPE | CODE_GENDER | FLAG_OWN_CAR | FLAG_OWN_REALTY | CNT_CHILDREN | AMT_INCOME_TOTAL | AMT_CREDIT | AMT_ANNUITY | AMT_GOODS_PRICE | ... | FLAG_DO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 100001 | Cash loans | F | N | Y | 0 | 135000.0 | 568800.0 | 20560.5 | 450000.0 | ... | |
| 1 | 100005 | Cash loans | M | N | Y | 0 | 99000.0 | 222768.0 | 17370.0 | 180000.0 | ... | |
| 2 | 100013 | Cash loans | M | Y | Y | 0 | 202500.0 | 663264.0 | 69777.0 | 630000.0 | ... | |
| 3 | 100028 | Cash loans | F | N | Y | 2 | 315000.0 | 1575000.0 | 49018.5 | 1575000.0 | ... | |
| 4 | 100038 | Cash loans | M | Y | N | 1 | 180000.0 | 625500.0 | 32067.0 | 625500.0 | ... | |

5 rows × 121 columns

```
application_test.describe()
```

| | SK_ID_CURR | CNT_CHILDREN | AMT_INCOME_TOTAL | AMT_CREDIT | AMT_ANNUITY | AMT_GOODS_PRICE | REGION_POPULATION_RELATIVE | DAYS_BIRTH | D |
|---|---|---|---|---|---|---|---|---|---|
| count | 48744.000000 | 48744.000000 | 4.874400e+04 | 4.874400e+04 | 48720.000000 | 4.874400e+04 | 48744.000000 | 48744.000000 | |
| mean | 277796.676350 | 0.397054 | 1.784318e+05 | 5.167404e+05 | 29426.240209 | 4.626188e+05 | 0.021226 | -16068.084605 | |
| std | 103169.547296 | 0.709047 | 1.015226e+05 | 3.653970e+05 | 16016.368315 | 3.367102e+05 | 0.014428 | 4325.900393 | |
| min | 100001.000000 | 0.000000 | 2.694150e+04 | 4.500000e+04 | 2295.000000 | 4.500000e+04 | 0.000253 | -25195.000000 | |
| 25% | 188557.750000 | 0.000000 | 1.125000e+05 | 2.606400e+05 | 17973.000000 | 2.250000e+05 | 0.010006 | -19637.000000 | |
| 50% | 277549.000000 | 0.000000 | 1.575000e+05 | 4.500000e+05 | 26199.000000 | 3.960000e+05 | 0.018850 | -15785.000000 | |
| 75% | 367555.500000 | 1.000000 | 2.250000e+05 | 6.750000e+05 | 37390.500000 | 6.300000e+05 | 0.028663 | -12496.000000 | |
| max | 456250.000000 | 20.000000 | 4.410000e+06 | 2.245500e+06 | 180576.000000 | 2.245500e+06 | 0.072508 | -7338.000000 | |

8 rows × 105 columns

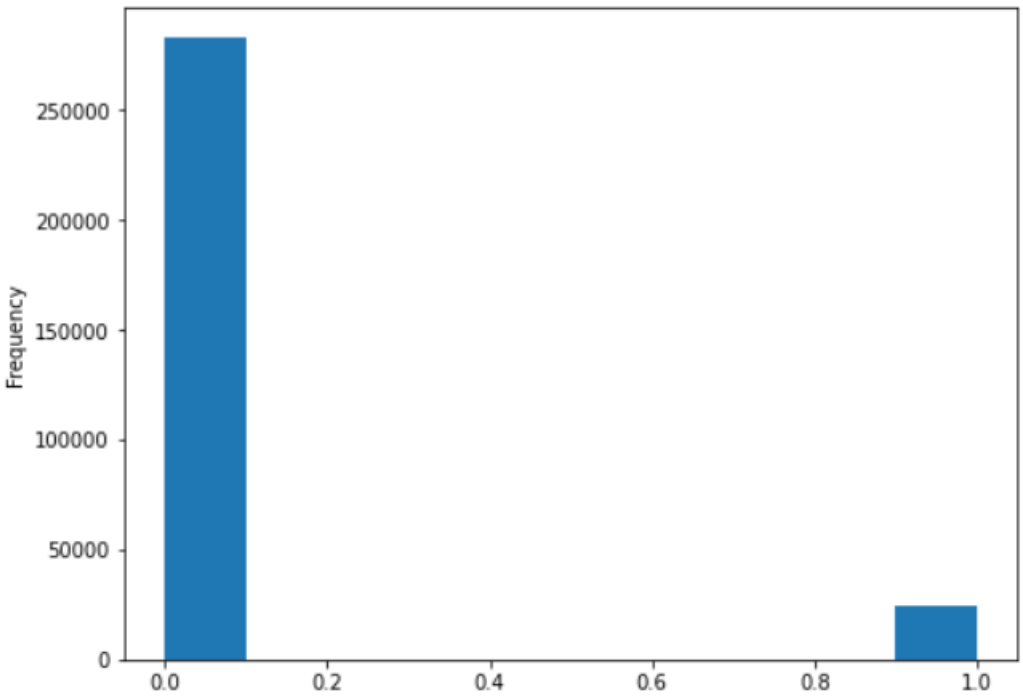
Testing Dataset

## B. Examine the Distribution for the Target Column:

The target is what we are asked to predict: either a 0 for loan was repaid on time, or a 1 indicating the client had payment difficulties. We can first examine the number of loans falling into each category.

```
application_train.TARGET.value_counts()
✓ 0.7s

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




From this information, we see this is an imbalanced class problem. There are far more loans that were repaid on time than loans that were not repaid.

## C. Examine the Missing Values:

When it comes time to build our machine learning models, we will have to fill in these missing values (known as imputation). Later, we will use machine learning models that can handle missing values with no need for imputation. Therefore, we will keep all of the columns for now. Figure below shows missing count and percentage of missing values in Training Dataset.

| | missing_count | missing_perc |
|---|---|---|
| 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 × 2 columns

There can be an option would be to drop columns with a high percentage of missing values, but it is not possible to know ahead of time if these columns will be helpful to our model.

Similarly missing counts and percentage of missing values of all datasets were found out.

## D. Column Types:

Let's look at the number of columns of each data type. int64 and float64 are numeric variables (which can be either discrete or continuous). object columns contain strings and are categorical features.

```
SK_ID_CURR                    int64
TARGET                        int64
NAME_CONTRACT_TYPE           object
CODE_GENDER                  object
FLAG_OWN_CAR                 object
                               ...
AMT_REQ_CREDIT_BUREAU_DAY    float64
AMT_REQ_CREDIT_BUREAU_WEEK   float64
AMT_REQ_CREDIT_BUREAU_MON    float64
AMT_REQ_CREDIT_BUREAU_QRT    float64
AMT_REQ_CREDIT_BUREAU_YEAR   float64
Length: 122, dtype: object
```

Training Dataset

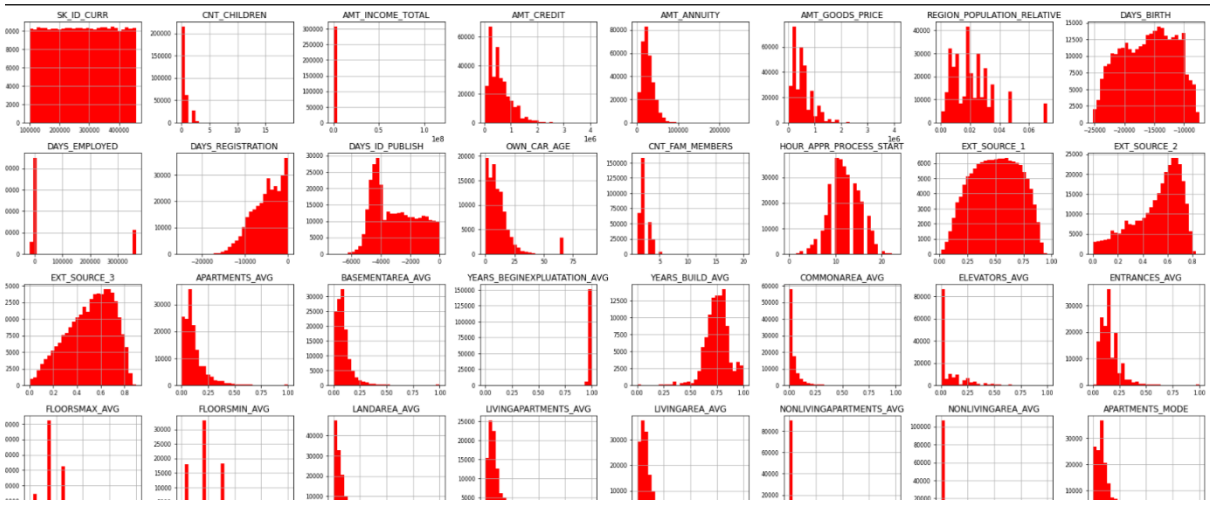
```
SK_ID_CURR                    int64
NAME_CONTRACT_TYPE           object
CODE_GENDER                  object
FLAG_OWN_CAR                 object
FLAG_OWN_REALTY              object
                               ...
AMT_REQ_CREDIT_BUREAU_DAY    float64
AMT_REQ_CREDIT_BUREAU_WEEK   float64
AMT_REQ_CREDIT_BUREAU_MON    float64
AMT_REQ_CREDIT_BUREAU_QRT    float64
AMT_REQ_CREDIT_BUREAU_YEAR   float64
Length: 121, dtype: object
```

Testing Dataset

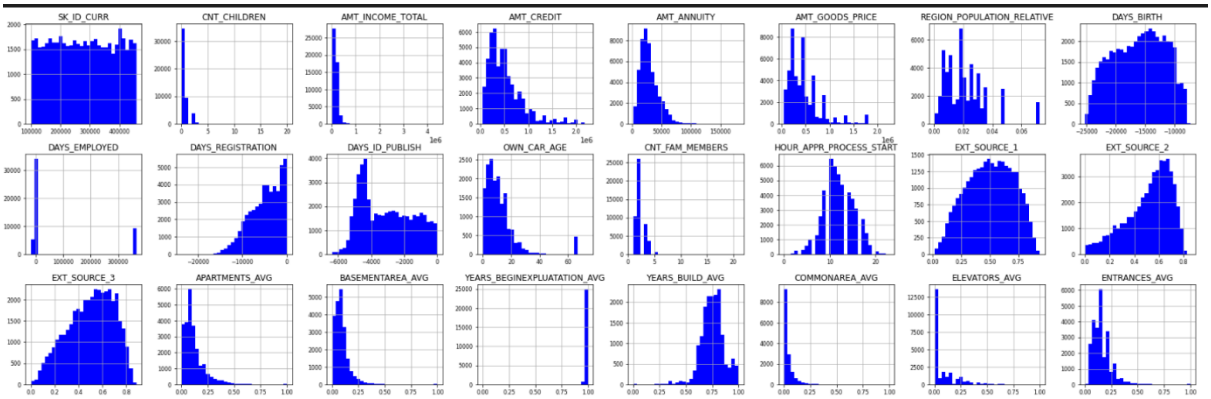
## E. Distribution (before Pre-processing and Cleansing):

### E.1. Outliers:

Finding out the distribution of all numeric features of Training and Testing Dataset before Data Cleansing and Pre-processing. Following are some of the histograms of numeric features.

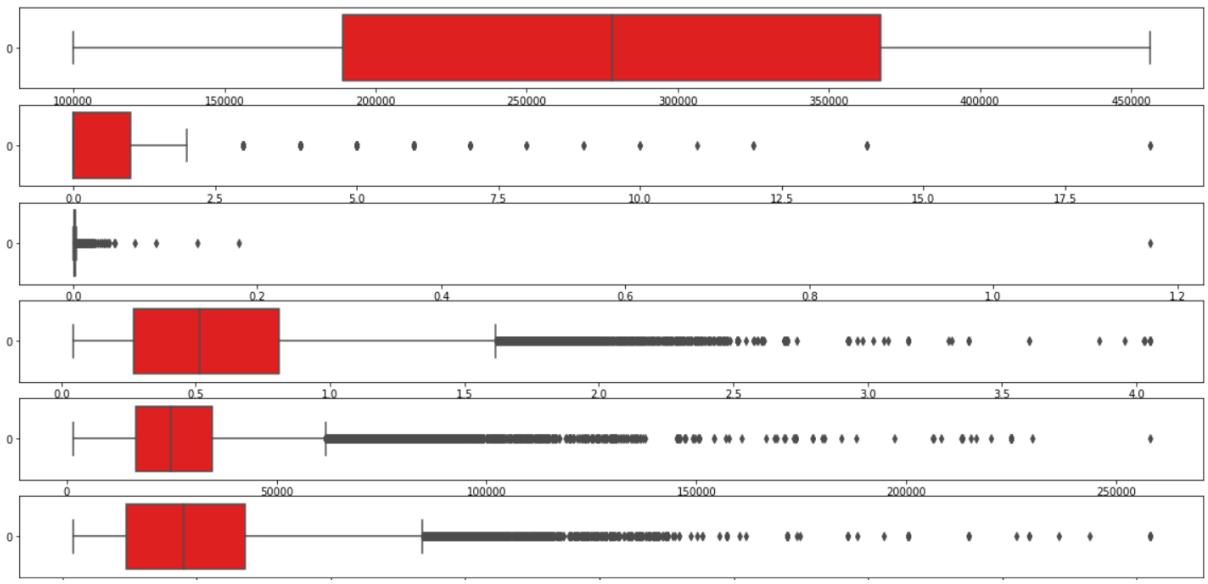



Training Dataset (1st Iteration)

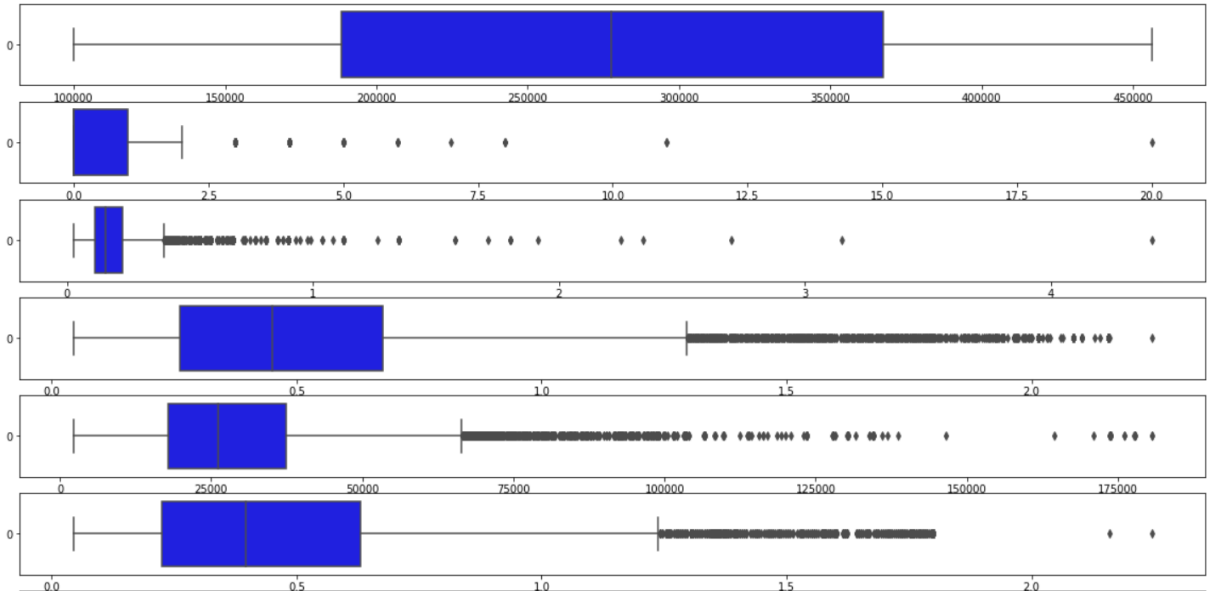



Testing Dataset (1st Iteration)

Observing from above graphs, it seems the distribution is not uniform for some columns. We will now look for outliers with the help of boxplots.



Training Dataset




Testing Dataset

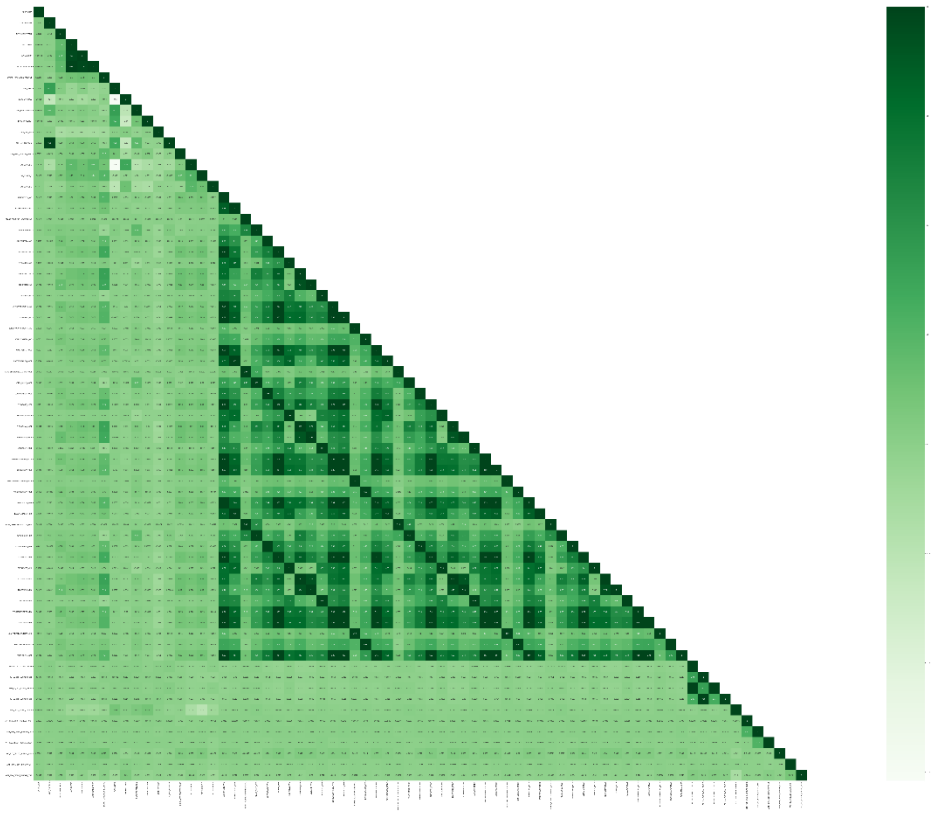
From the above boxplots, there are several outliers in features which needs to be handled. We shall handle those at a later stage. Let's now take a look at correlation between features.

## E.2. Correlations:

Now that we have dealt with the categorical variables and the outliers, let's continue with the EDA. One way to try and understand the data is by looking for correlations between the features and the target. We can calculate the Pearson correlation coefficient between every variable and the target using the .corr() dataframe method and visualize it using Heatmap method from seaborn library.

Some general interpretations of the absolute value of the correlation coefficient are:

- .00-.19 “very weak”
- .20-.39 “weak”
- .40-.59 “moderate”
- .60-.79 “strong”
- .80-1.0 “very strong”

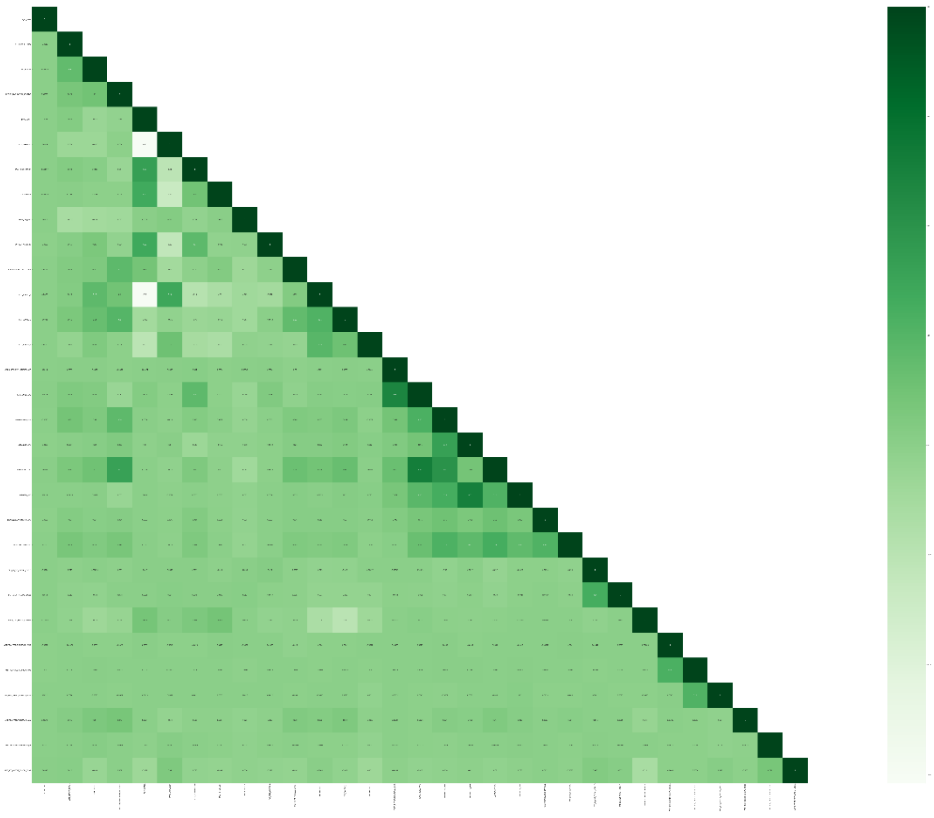


Training Dataset (1st Iteration)

Let's take a look at some of more significant correlations: the DAYS_BIRTH is the most positive correlation. Looking at the documentation, DAYS_BIRTH is the age in days of the client at the time of the loan in negative days. The correlation is positive, but the values of this feature is actually negative, meaning that as the client gets older, they

are less likely to default on their loan (i.e. the target == 0). The 3 variables with the strongest negative correlations with the target are EXT_SOURCE_1, EXT_SOURCE_2, and EXT_SOURCE_3. All three EXT_SOURCE features have negative correlations with the target, indicating that as the value of the EXT_SOURCE increases, the clients is more likely to repay the loan. We can also see that DAYS_BIRTH is positively correlated with EXT_SOURCE_1 indicating that maybe one of the factors in this score is the client age.

We will delete the features with correlation less than 2% and plot the Heatmap again.



Training Dataset (2nd Iteration)

## F. Handling Missing Values:

### F.1. Numeric Features:

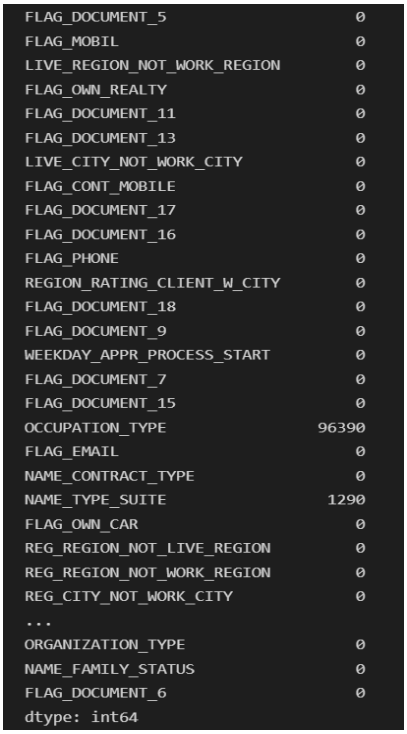
Firstly, we will be removing the un-necessary numeric features i.e. 'CNT_FAM_MEMBERS' and 'DAYS_LAST_PHONE_CHANGE'. Then we will impute numeric values with the median of respective features in both training and testing datasets.

```
col_del_candidate = { 'CNT_FAM_MEMBERS',
                      'DAYS_LAST_PHONE_CHANGE'
                    }

application_train = application_train.dropna(subset=col_del_candidate)
application_train.fillna((application_train.median()), inplace=True)

application_test = application_test.dropna(subset=col_del_candidate)
application_test.fillna((application_test.median()), inplace=True)
```

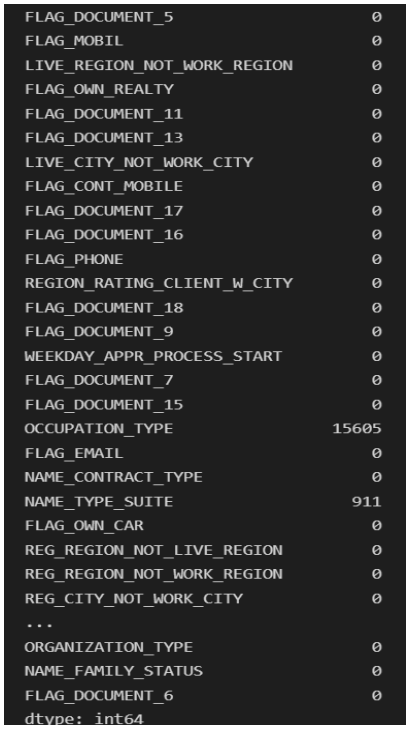
### F.2. Categorical Features:



| | |
|---|---|
| FLAG_DOCUMENT_5 | 0 |
| FLAG_MOBIL | 0 |
| LIVE_REGION_NOT_WORK_REGION | 0 |
| FLAG_OWN_REALTY | 0 |
| FLAG_DOCUMENT_11 | 0 |
| FLAG_DOCUMENT_13 | 0 |
| LIVE_CITY_NOT_WORK_CITY | 0 |
| FLAG_CONT_MOBILE | 0 |
| FLAG_DOCUMENT_17 | 0 |
| FLAG_DOCUMENT_16 | 0 |
| FLAG_PHONE | 0 |
| REGION_RATING_CLIENT_W_CITY | 0 |
| FLAG_DOCUMENT_18 | 0 |
| FLAG_DOCUMENT_9 | 0 |
| WEEKDAY_APPR_PROCESS_START | 0 |
| FLAG_DOCUMENT_7 | 0 |
| FLAG_DOCUMENT_15 | 0 |
| OCCUPATION_TYPE | 96390 |
| FLAG_EMAIL | 0 |
| NAME_CONTRACT_TYPE | 0 |
| NAME_TYPE_SUITE | 1290 |
| FLAG_OWN_CAR | 0 |
| REG_REGION_NOT_LIVE_REGION | 0 |
| REG_REGION_NOT_WORK_REGION | 0 |
| REG_CITY_NOT_WORK_CITY | 0 |
| ... | |
| ORGANIZATION_TYPE | 0 |
| NAME_FAMILY_STATUS | 0 |
| FLAG_DOCUMENT_6 | 0 |

dtype: int64

Training Dataset



| | |
|---|---|
| FLAG_DOCUMENT_5 | 0 |
| FLAG_MOBIL | 0 |
| LIVE_REGION_NOT_WORK_REGION | 0 |
| FLAG_OWN_REALTY | 0 |
| FLAG_DOCUMENT_11 | 0 |
| FLAG_DOCUMENT_13 | 0 |
| LIVE_CITY_NOT_WORK_CITY | 0 |
| FLAG_CONT_MOBILE | 0 |
| FLAG_DOCUMENT_17 | 0 |
| FLAG_DOCUMENT_16 | 0 |
| FLAG_PHONE | 0 |
| REGION_RATING_CLIENT_W_CITY | 0 |
| FLAG_DOCUMENT_18 | 0 |
| FLAG_DOCUMENT_9 | 0 |
| WEEKDAY_APPR_PROCESS_START | 0 |
| FLAG_DOCUMENT_7 | 0 |
| FLAG_DOCUMENT_15 | 0 |
| OCCUPATION_TYPE | 15605 |
| FLAG_EMAIL | 0 |
| NAME_CONTRACT_TYPE | 0 |
| NAME_TYPE_SUITE | 911 |
| FLAG_OWN_CAR | 0 |
| REG_REGION_NOT_LIVE_REGION | 0 |
| REG_REGION_NOT_WORK_REGION | 0 |
| REG_CITY_NOT_WORK_CITY | 0 |
| ... | |
| ORGANIZATION_TYPE | 0 |
| NAME_FAMILY_STATUS | 0 |
| FLAG_DOCUMENT_6 | 0 |

dtype: int64

Testing Dataset

Looking at the above list of categorical features, we can deduce that some categorical features such as 'OCCUPATION_TYPE' and 'NAME_TYPE_SUITE' have missing values. However, we can't drop those columns as those columns can be significant and impact accuracy of the model. Thus, we will first need to add 'Unknown' as a new category for those features and impute null values with the same.

```
cat_to_add = { 'OCCUPATION_TYPE', 'NAME_TYPE_SUITE' }
for c in cat_to_add:
    application_train[c] = application_train[c].cat.add_categories('Unkown')
    application_train[c].fillna("Unkown")
    application_test[c] = application_test[c].cat.add_categories('Unkown')
    application_test[c].fillna("Unkown")
```

Similarly, we will also remove other categorical features from Training and Testing Dataset based on Domain Knowledge.

## G. Categorical Features to Numeric:

A machine learning model unfortunately cannot deal with categorical variables (except for some models such as LightGBM). Therefore, we have to find a way to encode these variables as numbers before handing them off to the model. We will use Label Encoding for any categorical variables with only 2 categories and One-Hot Encoding for any categorical variables with more than 2 categories.

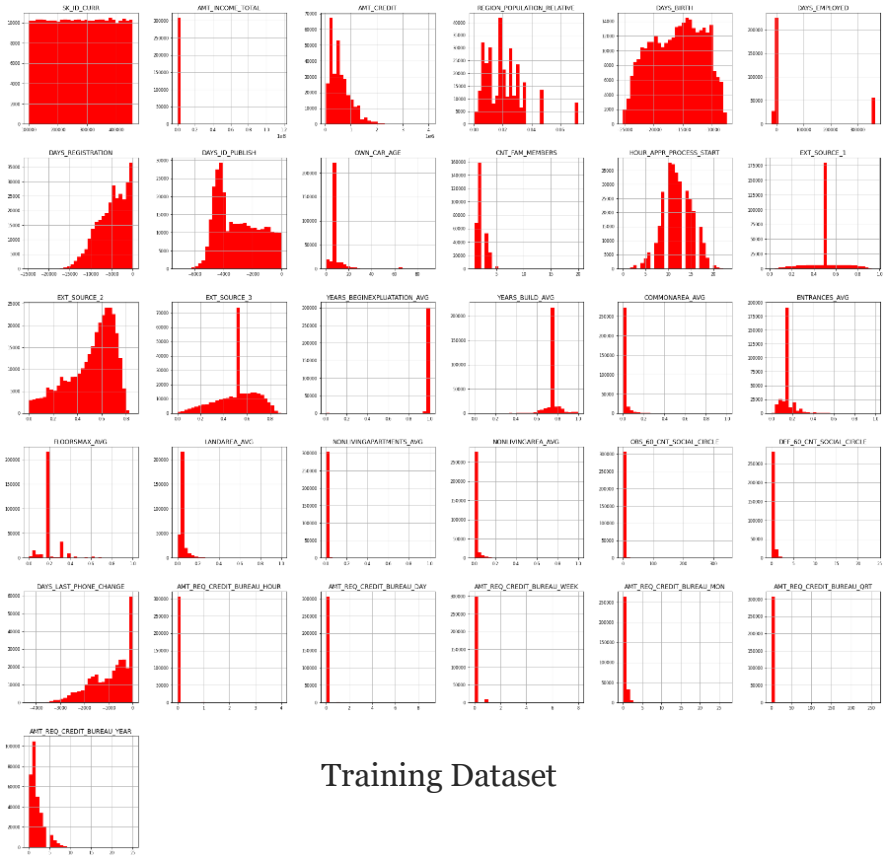
## G.1. One-Hot Encoding:

Creating a new column for each unique category in a categorical variable. Each observation receives a 1 in the column for its corresponding category and a 0 in all other new columns.

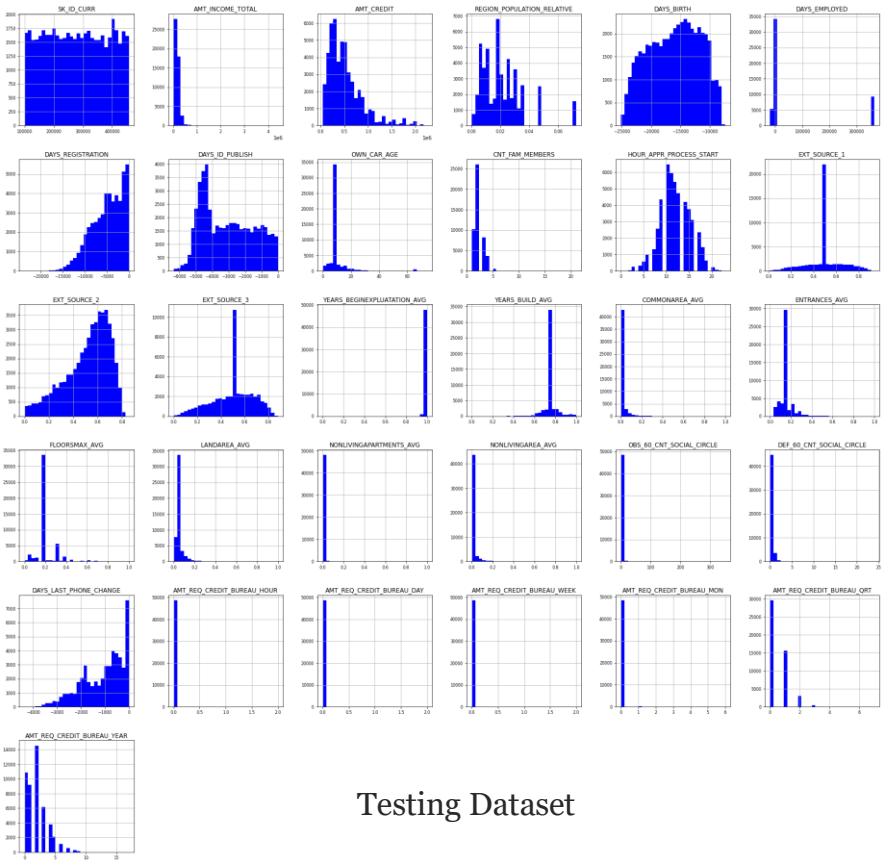
## G.2. Label Encoding:

Assign each unique category in a categorical variable with an integer. No new columns are created.

## H. New Distribution:




Training Dataset




Testing Dataset

Thus, by observing above plots we can deduce that the distribution across almost all numeric features is appropriate. However, some features are looking imbalanced which we shall be balancing. Before that, we removed some invalid numeric features that were non-relevant.

## I. Data Balancing:

In order to balance data in 'TARGET' variable, we will perform under-sampling technique. Since there is a high count of '0' category in 'TARGET' variable, we will be taking 50,000 sample records with '0' category using .sample() method and append it to records with '1' category within the training dataset.

```
n1=application_train.drop(application_train[application_train.TARGET == 0].index)
application_train=n1.append(application_train.drop(application_train[application_train.TARGET == 1].index).sample(50_000))
```

✓ 0.1s

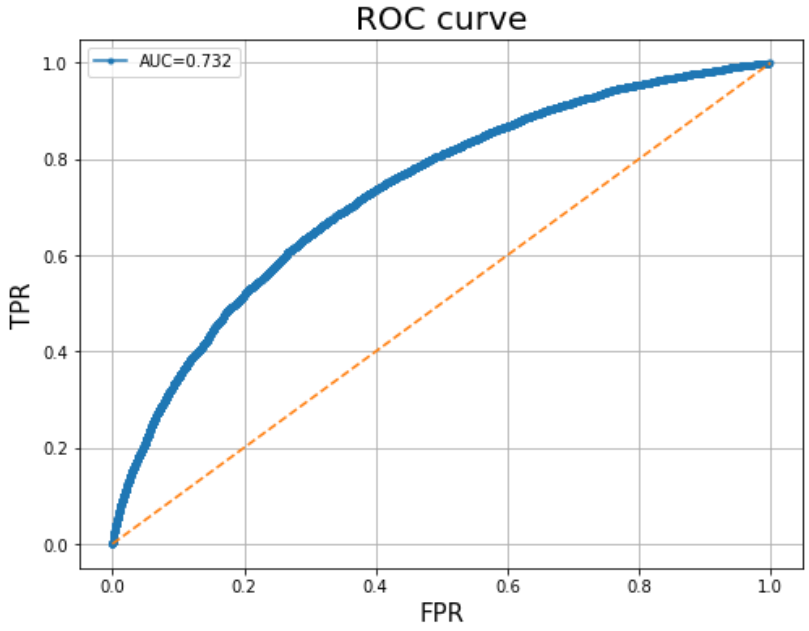
# 6. Baseline Model:

Prior to implementing machine learning we will normalize the data using StandardScaler method. This will change the values of numeric features in the dataset to a common scale, without distorting differences in the ranges of values. Also, we will split the data using train_test_split method since 'TARGET' variable is not present in testing data.

## Logistic Regression Model:

We will use Logistic Regression from Scikit-Learn for our first model. We first create the model, then we train the model using .fit and then we make predictions on the testing data using .predict_proba method (since we want probabilities and not a 0 or 1).

ROC_AUC_SCORE = 0.7318

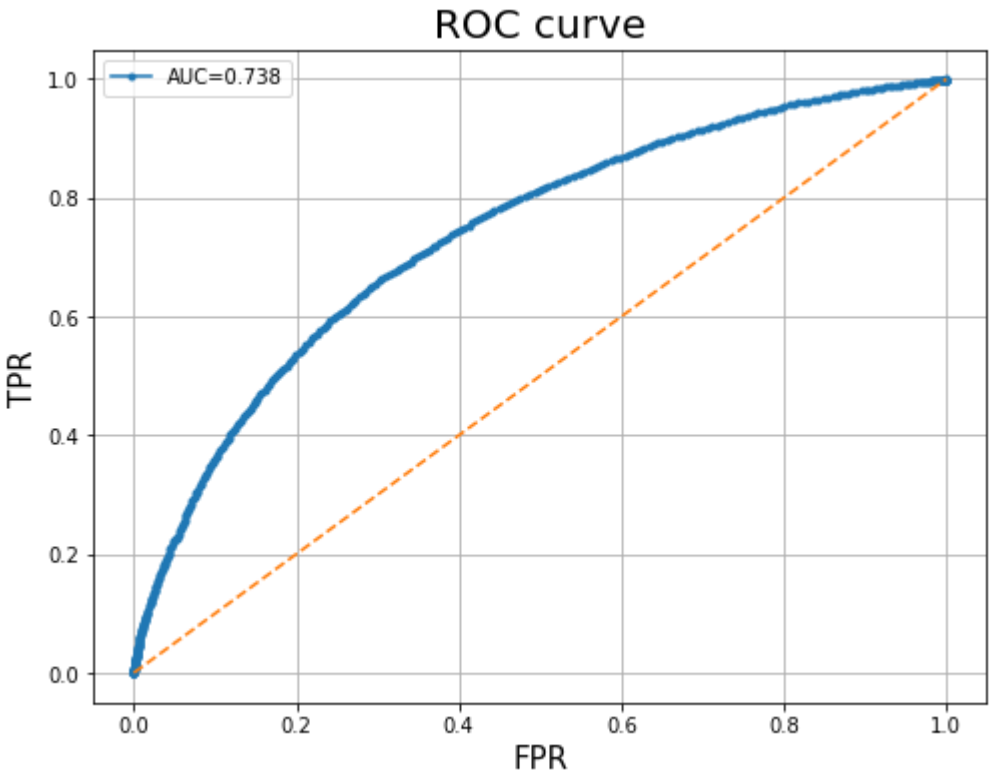


# 7. Improved Models:

## 1. Random Forest Classifier:

To try and beat the poor performance of our baseline, we can update the algorithm. Let's try using a Random Forest on the same training data to see how that affects performance. The Random Forest is a much more powerful model especially when we use hundreds of trees. We will use 500 trees in the random forest.

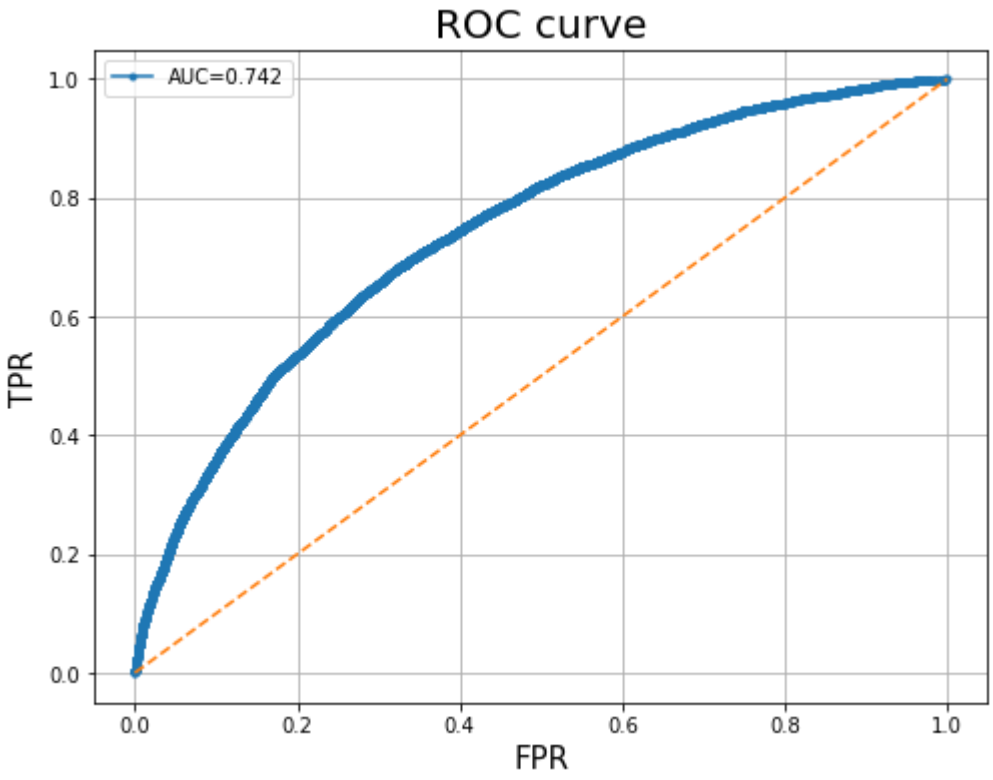
ROC_AUC_SCORE = 0.737



## 2. Light GBM Classifier:

We will improve our performance and update the algorithm. Let's try with LGBM Classifier on the same training data to see how that affects performance. This model accepts categorical features and doesn't require One-Hot Encoding or Label Encoding.

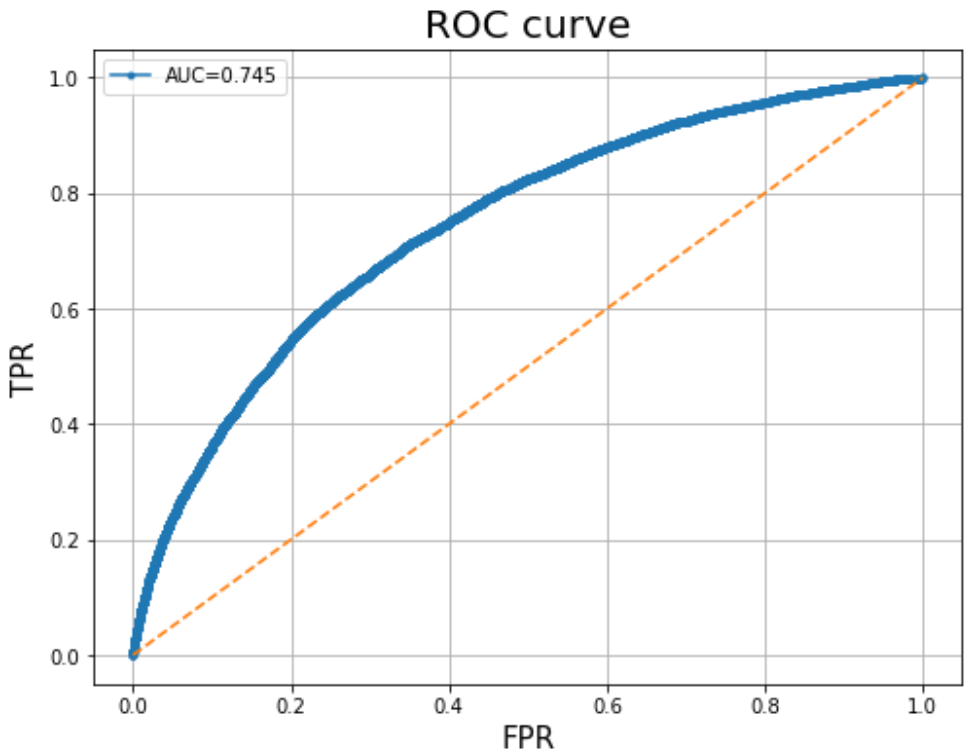
ROC_AUC_SCORE = 0.742



### 3. XGBoost Classifier:

We will further improve our performance and update the algorithm again. It provides parallel tree boosting and is the leading machine learning library for regression, classification, and ranking problems. It also accepts NULL values and doesn't require imputation. It attempts to accurately predict a target variable by combining the estimates of a set of simpler, weaker models.

ROC_AUC_SCORE = 0.7446



# 7. Conclusion:

We first made sure to understand the data, our task, and the metric by which our submissions will be judged. Then, we performed a fairly simple EDA to try and identify relationships and trends that may help our modelling. Along the way, we performed necessary pre-processing steps such as encoding categorical variables, imputing missing values, and scaling features to a range. Then, we constructed new features out of the existing data to see if doing so could help our model.

Once the data exploration, data preparation, and feature engineering were complete, we implemented a baseline model upon which we hope to improve. Then we built slightly more complicated models to improve our performance. Below table shows our machine learning model scores.

| ***Machine Learning Model*** | ***AUC Score*** |
|---|---|
| *Logistic Regression* | 0.732 |
| *Random Forest Classifier* | 0.737 |
| *Light GBM Classifier* | 0.742 |
| *XGBoost Classifier* | 0.745 |

Thus, we conclude that XGBoost Classifier model turned out to be the best performer overall.