Home Credit Default Risk

Data Exploration and Initial Analysis

Introduction

Taking loans has been a crucial part of many of our lives. Whether it's for education, or buying a new house or supporting a business, people prefer taking a loan over spending liquid cash. However, sometimes people have a disadvantage in obtaining loans due to nonexistent or low credit history. Home Credit is a financial provider that is trying to make borrowing a positive experience and ensure that clients capable of repayment are not rejected.

In this study, we try to utilize alternative data such as transactional history in order to predict the client's repaying ability and thus allow people with repayment ability to obtain the loan they need.

Data Preprocessing and Missing Values

Our dataset is already split into Train and Test sets. We work on the Train data to train and evaluate the model before deploying it on the Test set.We start with basic EDA, where we explore the following:

- Missing or null values.
- Distribution of target variable over numerical and categorical variables.
- Correlation between features and the label.

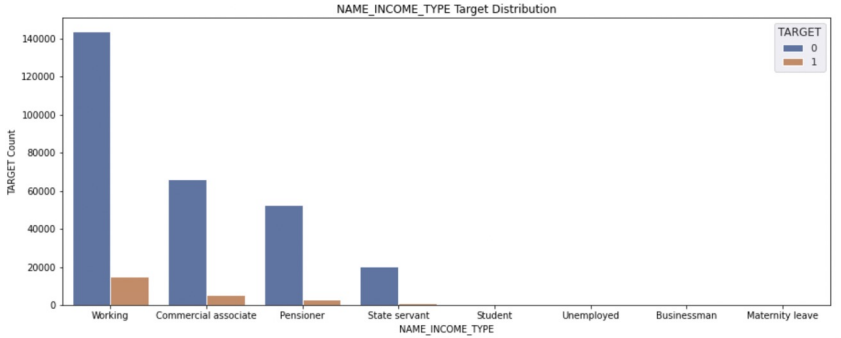
```
app_train.isna().sum()*100/len(app_train)
```

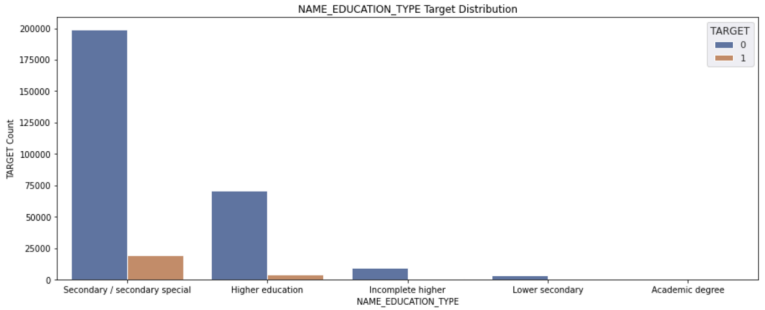
EXT_SOURCE_1	56.381073
EXT_SOURCE_2	0.214626
EXT_SOURCE_3	19.825307
APARTMENTS_AVG	50.749729
BASEMENTAREA_AVG	58.515956
YEARS_BEGINEXPLUATATION_AVG	48.781019
YEARS_BUILD_AVG	66.497784
COMMONAREA_AVG	69.872297
ELEVATORS_AVG	53.295980
ENTRANCES_AVG	50.348768
FLOORSMAX_AVG	49.760822

Percentage of Missing Values

Categorical Variables

- We plot the relationship between the target variable and categorical features using bar charts.
- We observe that the distribution of the target variable is highly imbalanced with more 0's than 1's across all categories.
- The feature OCCUPATION_TYPE had missing values. We filled these with the entry ‘NA’.
- We perform One Hot Encoding on the categorical data to convert it into numerical data.
- We perform target encoding on the features OCCUPATION_TYPE and ORGANIZATION_TYPE.






```
## Editing Occupation Type
app_train.loc[app_train['OCCUPATION_TYPE'].isna()==True,'OCCUPATION_TYPE'] = 'NA'
#app_test.loc[app_test['OCCUPATION_TYPE'].isna()==True,'OCCUPATION_TYPE'] = 'NA'
```

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

Categorical Features

```
!pip install category_encoders
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from category_encoders import TargetEncoder
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder, OrdinalEncoder

mixed_df = X_train[num + cat + te]
# target = X_train["TARGET"]
preprocess = make_column_transformer((StandardScaler(), num),
                                     (TargetEncoder(), te),
                                     (OneHotEncoder(), cat),
                                     remainder = "passthrough")
```

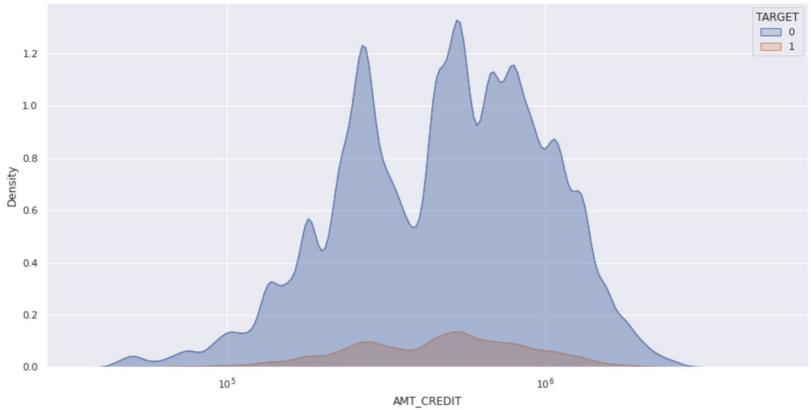
Numerical Variables

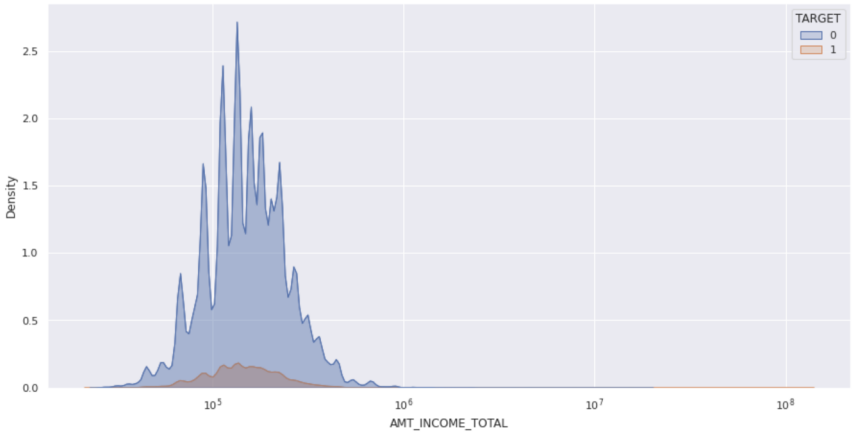
- We analyze the distribution of target variable over the numerical variables and observe a class imbalance.
- We drop columns with more than 35% missing values. For others, we used imputation techniques such as median to fill them.
- We scale the numerical variables using StandardScaler.

```
simple_imputer7 = SimpleImputer(missing_values=np.nan, strategy='median')
X_train['EXT_SOURCE_3'] = simple_imputer7.fit_transform(np.array(X_train['EXT_SOURCE_3']).reshape(-1,1))
X_val['EXT_SOURCE_3'] = simple_imputer7.transform(np.array(X_val['EXT_SOURCE_3']).reshape(-1,1))
X_test['EXT_SOURCE_3'] = simple_imputer7.transform(np.array(X_test['EXT_SOURCE_3']).reshape(-1,1))

simple_imputer8 = SimpleImputer(missing_values=np.nan, strategy='median')
X_train['EXT_SOURCE_2'] = simple_imputer8.fit_transform(np.array(X_train['EXT_SOURCE_2']).reshape(-1,1))
X_val['EXT_SOURCE_2'] = simple_imputer8.transform(np.array(X_val['EXT_SOURCE_2']).reshape(-1,1))
X_test['EXT_SOURCE_2'] = simple_imputer8.transform(np.array(X_test['EXT_SOURCE_2']).reshape(-1,1))

simple_imputer9 = SimpleImputer(missing_values=np.nan, strategy='median')
X_train['AMT_ANNUITY'] = simple_imputer9.fit_transform(np.array(X_train['AMT_ANNUITY']).reshape(-1,1))
X_val['AMT_ANNUITY'] = simple_imputer9.transform(np.array(X_val['AMT_ANNUITY']).reshape(-1,1))
X_test['AMT_ANNUITY'] = simple_imputer9.transform(np.array(X_test['AMT_ANNUITY']).reshape(-1,1))

simple_imputer10 = SimpleImputer(missing_values=np.nan, strategy='median')
X_train['OWN_CAR_AGE'] = simple_imputer10.fit_transform(np.array(X_train['OWN_CAR_AGE']).reshape(-1,1))
X_val['OWN_CAR_AGE'] = simple_imputer10.transform(np.array(X_val['OWN_CAR_AGE']).reshape(-1,1))
X_test['OWN_CAR_AGE'] = simple_imputer10.transform(np.array(X_test['OWN_CAR_AGE']).reshape(-1,1))

simple_imputer11 = SimpleImputer(missing_values=np.nan, strategy='median')
X_train['OBS_30_CNT_SOCIAL_CIRCLE'] = simple_imputer11.fit_transform(np.array(X_train['OBS_30_CNT_SOCIAL_CIRCLE']).reshape(-1,1))
X_val['OBS_30_CNT_SOCIAL_CIRCLE'] = simple_imputer11.transform(np.array(X_val['OBS_30_CNT_SOCIAL_CIRCLE']).reshape(-1,1))
X_test['OBS_30_CNT_SOCIAL_CIRCLE'] = simple_imputer11.transform(np.array(X_test['OBS_30_CNT_SOCIAL_CIRCLE']).reshape(-1,1))

simple_imputer12 = SimpleImputer(missing_values=np.nan, strategy='median')
X_train['DAYS_LAST_PHONE_CHANGE'] = simple_imputer12.fit_transform(np.array(X_train['DAYS_LAST_PHONE_CHANGE']).reshape(-1,1))
X_val['DAYS_LAST_PHONE_CHANGE'] = simple_imputer12.transform(np.array(X_val['DAYS_LAST_PHONE_CHANGE']).reshape(-1,1))
X_test['DAYS_LAST_PHONE_CHANGE'] = simple_imputer12.transform(np.array(X_test['DAYS_LAST_PHONE_CHANGE']).reshape(-1,1))
```

```
num = [
    'CNT_CHILDREN',
    'AMT_INCOME_TOTAL',
    'AMT_CREDIT',
    'AMT_ANNUITY',
    'AMT_GOODS_PRICE',
    'REGION_POPULATION_RELATIVE',
    'DAYS_BIRTH',
    'DAYS_EMPLOYED',
    'DAYS_REGISTRATION',
    'DAYS_ID_PUBLISH',
    'CNT_FAM_MEMBERS',
    'OWN_CAR_AGE',
    'OBS_30_CNT_SOCIAL_CIRCLE',
    'DEF_30_CNT_SOCIAL_CIRCLE',
    'OBS_60_CNT_SOCIAL_CIRCLE',
    'DEF_60_CNT_SOCIAL_CIRCLE',
    'AMT_REQ_CREDIT_BUREAU_HOUR_T',
    'AMT_REQ_CREDIT_BUREAU_DAY_T',
    'AMT_REQ_CREDIT_BUREAU_WEEK_T',
    'AMT_REQ_CREDIT_BUREAU_MON_T',
    'AMT_REQ_CREDIT_BUREAU_QRT_T',
    'AMT_REQ_CREDIT_BUREAU_YEAR_T'
    ]
```

Numerical Features



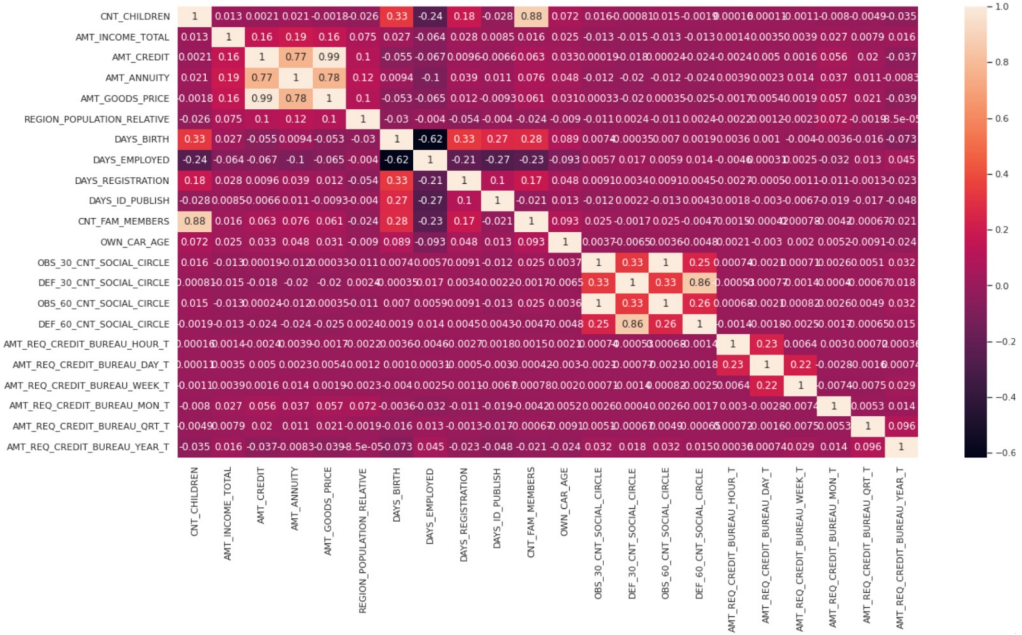


Multicollinearity and Correlation

We check correlation between the numerous features and observe that the following pairs are highly correlated:

- **AMT_CREDIT and AMT_GOODS_PRICE:** If the house is valued at $200,000 then the credit is likely to be equivalent or close to that.
- **CNT_CHILDREN and CNT_FAM_MEMBERS**: If a family has 3 kids then the number of members are most likely to be 2+3.
- **DEF_30_CNT_SOCIAL and DEF_60_CNT_SOCIAL:** If people in someone's social circle have a payment due for 60 days then it's obvious they have it due for the past 30 days as well.

With a threshold limit set to 0.85 we drop one feature from each pair.



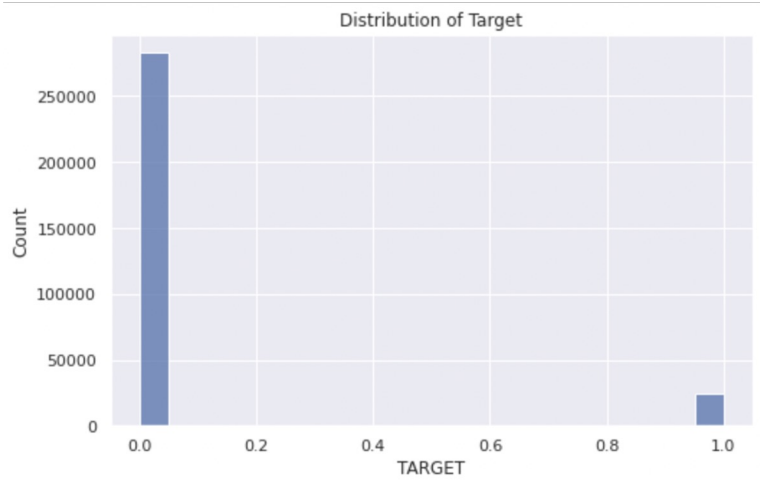

```
corr_matrix = X_train.corr().abs()

# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))

# Find features with correlation greater than 0.85
to_drop = [column for column in upper.columns if any(upper[column] >= 0.85)]

# Drop features
X_train.drop(to_drop, axis=1, inplace=True)
X_val.drop(to_drop, axis=1, inplace=True)
X_test.drop(to_drop, axis=1, inplace=True)
```

Splitting Methodology and Sampling

Since the dataset is highly imbalance, we do **stratified sampling** to ensure that both validation and training set get the same ratio of both classes.

We split the training dataset such that 20% of it is allocated for validation.

We further use SMOTE to increase the number of cases of 1s in our dataset in a balanced way.




```
## Sampling using SMOTE
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
x_smote, y_smote = oversample.fit_resample(x_train, y_train)
```

```
from sklearn.model_selection import train_test_split
app_train_y_train = app_train["TARGET"]
X_dev, X_val, y_dev, y_val = train_test_split(app_train.drop('TARGET',axis = 1), app_train_y_train, test_size=0.2, random_state=0, stratify = app_train_y_train)
X_train, X_test, y_train, y_test = train_test_split(X_dev, y_dev, test_size=0.25, random_state=0)
```

Baseline model: Logistic Regression

We train a baseline model in order to understand model performance with default parameters and observe how much complex models may improve when compared to the baselines.

We chose to use logistic regression instead of linear regression as the latter predicts continuous values whereas the former is used in the binary classification of values.

The confusion matrix shows the Logistic Regression model predicts “True” to all results (all predictions are either true positives or false positives). This could be related to imbalanced dataset and we plan to alter our data processing method to attain a more accurate baseline model performance.

Code Implementation

```
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression()
lr_model.fit(x_smote,y_smote)
lr_predict = lr_model.predict(x_val)
```

Confusion Matrix

```
from sklearn.metrics import (classification_report, accuracy_score, f1_score,
precision_score, recall_score, roc_auc_score, average_precision_score, confusion_matrix)

confusion_matrix(y_true = y_val, y_pred = lr_predict)
```

```
array([[56648,     0],
       [ 4855,     0]])
```

Baseline model: Gradient Boosting (XGBoost)

We chose to utilize XGBoost, an ensemble boosting model as it supports numerous hyper-parameter feature tuning, as well as its inbuilt support for missing and sparse data.

From the confusion matrix below we see the xgboost model is predicting “True” to most data points, and there are approximately the same number of False negatives and True negatives. This shows the XGBoost model is not much better. We plan to tune the XGBoost model and re-evaluate our preprocessing methods.

Code Implementation

```
from xgboost import XGBClassifier
xgb = XGBClassifier()
xgb.fit(x_smote, y_smote)
xgb_predict = xgb.predict(x_val)
```

Confusion Matrix

```
confusion_matrix(y_true = y_val, y_pred = xgb_predict)
```

```
array([[56513,   135],
       [ 4732,   123]])
```

Further Work

We plan to improve upon our feature engineering, add additional data features, and use models like Logistic Regression, Random Forests, LGBM, and XGBoost. Neural networks have been outperforming other techniques in many machine learning problems, so we hope to explore Deep Neural Networks on our dataset. We will compare model performances to find the best performing model based on Average Precision. Since this may require significantly more computing power and hardware support, we will use Amazon SageMaker for it.

Thank you!

Applied Machine Learning

Group 27:

- Bora Elci (be2246)
- Mark Wu (rw2921)
- Parth Batra (pb2882)
- Shruti Agarwal (sa4136)