

## Project Phase 2

Home Credit Default Risk(HCDR)  
FP\_GroupN\_11



Luddy School of Informatics, Computing, and Engineering

Developed by-  
Kumud Sharma([kumsharm@iu.edu](mailto:kumsharm@iu.edu))  
Kamna Chaudhary ([kamchau@iu.edu](mailto:kamchau@iu.edu))  
Bhavya Mistry([brmistry@iu.edu](mailto:brmistry@iu.edu))  
Jaydeep Patel([jp157@iu.edu](mailto:jp157@iu.edu))



## **Abstract**

Using the "Home Credit Default Risk" dataset, we attempted to develop a machine learning model that forecasts a borrower's repayment capacity. To do this, we used application, demographic, and credit behavior history data to perform basic exploratory data analysis, and we created a baseline pipeline that also dealt with missing, aberrant, and uncommon data.

The metrics Accuracy, F1 Score, and Confusion Matrix are among the ones we want to publish. In addition to Logistic Regression, Decision Tree Classifier, Random Forest, NaiveBayes Classifier, and XGBoost Classifier, we also used other baseline models.

Logistic Regression and XGBoost, our basic model, had the highest training accuracy, of 91%. DecisionTreeClassifier, GaussianNB test accuracy were found to be 53.8%, 72.25%. After submitting to Kaggle, we received an AUC private score of 71.54% and a public score of 71.49%. Our tests revealed that XGBoost and Logistic Regression had the highest accuracy, both at 73.85%.

## **Introduction:**

Using the "Home Credit Default Risk" dataset, the goal of this study was to create a machine learning model that forecasts a borrower's capacity to repay a loan. We carried out exploratory data analysis and built a foundational pipeline to manage missing, abnormal, and unusual data. We tested with a number of baseline models, including Logistic Regression, Decision Tree Classifier, Random Forest, NaiveBayes Classifier, and XGBoost Classifier.

## **Data Description:**

The task to hand is to build a machine learning model that can predict if a client will repay the loan or not based on the credit history and many other features. The primary data for the task is stored in the csv files: "application\_train.csv" and "application\_test.csv". The training data contains 121 feature columns, which have categorical and numerical variables. The target column indicates whether the client repaid the loan or not and is present in training dataset.

Other than the primary dataset, there are six subsidiary datasets that contain credit history information for each client. These datasets include "bureau.csv," which contains any credit history for the customer prior to the application date, and "bureau\_balance.csv," which includes data for the prior credits stated in the bureau dataset for each month in history. These datasets interact with one another and the application dataset to provide a complete picture of each client's credit history.

To build the machine learning model, we will first need to perform exploratory data analysis (EDA) to understand the relationships between the features and the target variable. We can use various diagrams, such as box plot, scatter plots, heatmaps, and histograms, to visualize the data and identify any patterns or correlations. Once we have a good understanding of the data, we can start pre-processing it by handling missing values, encoding categorical variables, and scaling numerical features.

Next, we will select a suitable machine learning algorithm. We can use cross-validation to evaluate the model's performance and fine-tune its hyperparameters to optimize its accuracy.

Finally, we can use the trained model to make predictions on the test dataset and submit them to the competition platform to see how well our model performs compared to other participants. Overall, this is a challenging task that requires a thorough understanding of the data and careful selection and optimization of machine learning algorithms.

### **1. application\_{train|test}.csv**

This dataset is the primary dataset and serves as the base for connecting with other datasets. It contains information on the loan applicants' personal characteristics and uses the SK\_ID\_CURR property as the primary key.

### **2. previous\_application.csv**

This dataset contains details regarding prior Home Credit loans made by the applicant. It includes characteristics such as loan status, down payment, and loan type. The SK\_ID\_CURR property from the application\_{train|test}.csv dataset serves as the primary key for connecting with this dataset.

### **3. instalments\_payments.csv**

This dataset contains information on loan repayment from the past. It may be connected to the SK\_ID\_CURR property from the application\_{train|test}.csv dataset or the SK\_ID\_PREV property from the previous\_application.csv dataset, depending on the context.

#### **4. credit\_card\_balance.csv**

This dataset contains information on Home Credit credit card transactions. It may be connected to the SK\_ID\_CURR property from the application\_{train|test}.csv dataset or the SK\_ID\_PREV property from the previous\_application.csv dataset, depending on the context.

#### **5. POS\_CASH\_balance.csv**

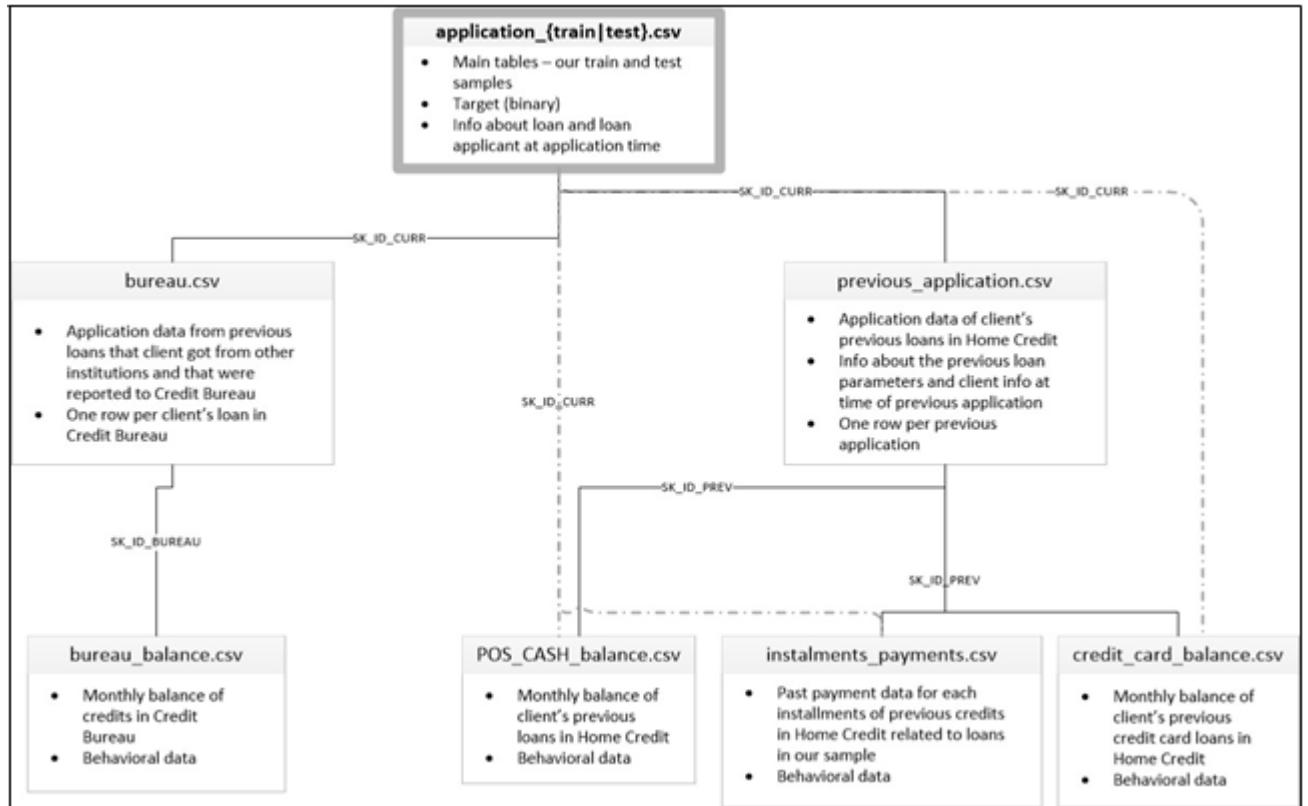
This dataset describes the individual's past credit history at Home Credit, including personal loans and consumer credit. It may be connected to the SK\_ID\_CURR property from the application\_{train|test}.csv dataset or the SK\_ID\_PREV property from the previous\_application.csv dataset, depending on the context.

#### **6. bureau.csv**

This dataset contains details about a person's prior credit history at other financial institutions that were reported to a credit agency. It may be connected to the SK\_ID\_CURR property from the application\_{train|test}.csv dataset or the SK\_ID\_BUREAU property from the bureau.csv dataset, depending on the context.

#### **7. Previous\_application.csv**

This dataset includes the credit bureau's monthly credit balance. It may be connected to the SK\_ID\_CURR property from the application\_{train|test}.csv dataset or the SK\_ID\_BUREAU property from the bureau.csv dataset, depending on the context.



## Tasks to be accomplished in Phase-2:

### **1. Data Visualization and Exploratory Data Analysis:**

Data visualization is the graphic representation of data and information. Charts, graphs, and maps are used as visualizations to help show patterns, trends, and connections in the data. EDA, on the other hand, is the procedure of using graphical and statistical tools to assess and list the salient aspects of a dataset. Outlier detection, correlation analysis, and summary statistics are a few of the methods used in EDA. EDA is a critical step in developing a machine learning model that is effective for our problem statement and will play a significant part in verifying the correctness and quality of our data.

### **2. Data cleaning and preprocessing:**

Several tables in the HCDR dataset have missing values, outliers, and inconsistencies. Imputation, normalization, and feature scaling will be used to address these problems in order to get the data ready for modeling.

### **3. Selection of Model:**

In order to choose the best model for the task, a number of machine learning algorithms will be compared, taking into account performance indicators including accuracy, AUC-ROC,F-1 score. Logistic regression, decision trees, random forests, gradient boosting, and neural networks are a few examples of potential models.

### **4. Evaluation and Interpretation:**

To examine the generalization performance of the final model and spot any potential biases or constraints, it will be tested on a holdout set of data. To obtain understanding of the underlying connections between features and outcomes, interpretability techniques like feature importance analysis and partial dependence plots may be applied.

# Experiments:

## Models:

We are aiming to anticipate the client's ability to repay the loan, as explained in the abstract and data description module, and this ability is decided by a single binary output variable called TARGET (0|1). Under supervised learning, this is a binary classification issue because the goal has only two discrete possibilities that could occur. We will compare various categorization models using the chosen metrics to find the model that best fits the situation.

We will use Logistic Regression (with Lasso, Ridge, and No Regularization) as our basic model because this is a binary classification problem. Based on the Logistic Regression model, we will evaluate the effectiveness of different models. The likelihood of various classes or clusters occurring in the dataset will be predicted using a Naive Bayes classifier. We will use the Decision Tree Classifier model by developing rules based on the application train.csv dataset, notably on the income and credit columns such as amt income total, amt credit, etc. We will also investigate Support Vector Machines (SVM), Random Forest Classifier, K-Nearest Neighbors, and XGBoost Classifier. Later on, we'll concentrate on dimensionality reduction and hyperparameter tweaking, setting the stage for a thrilling algorithmic race.

A brief introduction of the models is as below:

**1. Logistic Regression** - This statistical model type is employed to investigate the relationship between a binary or categorical dependent variable and one or more independent variables. To simulate the possibility that the dependent variable will, a logistic function is used. Any input is transformed by the logistic function into a value between 0 and 1, which represents the likelihood of the binary outcome. Log Loss is a tool used in logistic regression.

**2. Decision Tree Classification** - Using the features of the input data, a tree-like model of decisions and possible outcomes is constructed in this model. In order to partition the data according to a specific criterion, such as Gini Impurity or Information Gain, the approach starts with the entire dataset and selects the best feature. Depending on the chosen characteristic, the data are then separated into subsets, and the operation is then repeated repeatedly for each subset until all of the data in each subset belong to the same class, or a stopping condition is satisfied.

**3. Random Forest Classification** - This model constructs a number of decision trees by randomly selecting features for each split and using subsets of the input data. Since each decision tree is trained using a bootstrapped sample of the data, it only sees a portion of the data. The outcome of the algorithm is determined by the majority vote of the decision trees.

**4. Neural Networks** - Neural networks are a category of models that learn by varying the weights of the input data to produce a specific output. The input features are supplied into a network of linked neurons, which then manipulates the input using a number of mathematical operations. These changes are controlled by a set of weights that are developed during training. The weights are then modified using a loss function, such as cross-entropy loss, when the output is compared to the actual goal output.

$$\text{BinaryCrossEntropy} = H_p(q) = -\frac{1}{N} \sum_{i=1}^n y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

## Metrics:

### 1. Confusion Matrix:

A confusion matrix is used to evaluate the effectiveness of a classification model, and the resulting matrix reveals the degree of classification accuracy and potential prediction mistakes for each record. It is used to gauge AUC-ROC curve, recall, accuracy, and precision.

		Predicted			
		Positive	Negative		
Actual	Positive	True Positive (TP) Correct objects	False Negative (FN) Missed objects	Recall	$\frac{(TP+TN)}{(TP+FN)}$
	Negative	False Positive (FP) Extra objects	True Negative (TN) No objects	Accuracy	
		Precision $\frac{TP}{(TP+FP)}$	Negative predictive value $\frac{TN}{(TN+FP)}$	Specificity	$\frac{TN}{(TN+FP)}$

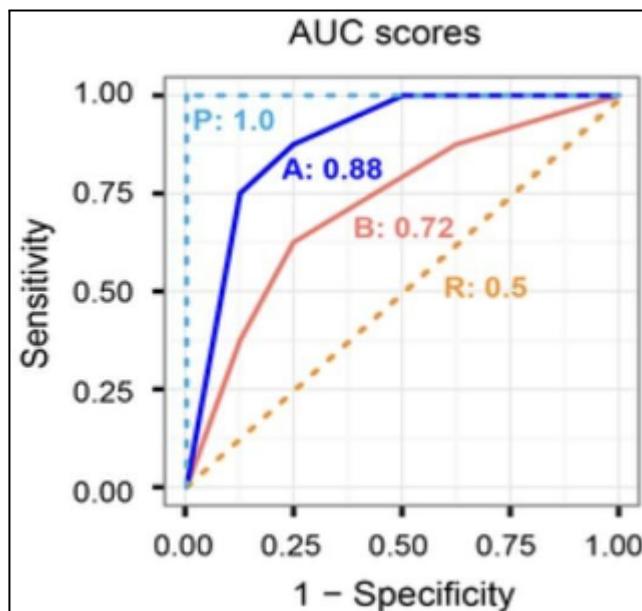
### 2. F-1 Score:

It will be used to compare the effectiveness of two classifiers based on their precision and recall values. It will be crucial in deciding which model fits the issue more effectively.

$$F_1 = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

### **3.AUC-ROC Curve:**

When the actual outcome is positive, the ROC curve will assist us in calculating the likelihood of correctly predicting the positive class. Its performance is distilled into a single value using the AUC curve.



## **EDA**

- The Test dataset have following **numerical columns** such as:

{'SK\_ID\_CURR', 'TARGET', 'CNT\_CHILDREN', 'AMT\_INCOME\_TOTAL', 'AMT\_CREDIT',  
'AMT\_ANNUITY', 'AMT\_GOODS\_PRICE', 'REGION\_POPULATION\_RELATIVE',  
'DAYS\_BIRTH', 'DAYS\_EMPLOYED', 'DAYS\_REGISTRATION', 'DAYS\_ID\_PUBLISH',  
'OWN\_CAR\_AGE', 'FLAG\_MOBIL', 'FLAG\_EMP\_PHONE', 'FLAG\_WORK\_PHONE',  
'FLAG\_CONT\_MOBILE', 'FLAG\_PHONE', 'FLAG\_EMAIL', 'CNT\_FAM\_MEMBERS',  
'REGION\_RATING\_CLIENT', 'REGION\_RATING\_CLIENT\_W\_CITY',  
'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', 'EXT\_SOURCE\_1', 'EXT\_SOURCE\_2', 'EXT\_SOURCE\_3',  
'APARTMENTS\_AVG', 'BASEMENTAREA\_AVG', 'YEARS\_BEGINEXPLUATATION\_AVG',

```

'YEARS_BUILD_AVG', 'COMMONAREA_AVG', 'ELEVATORS_AVG', 'ENTRANCES_AVG',
'FLOORSMAX_AVG', 'FLOORSMIN_AVG', 'LANDAREA_AVG',
'LIVINGAPARTMENTS_AVG', 'LIVINGAREA_AVG', 'NONLIVINGAPARTMENTS_AVG',
'NONLIVINGAREA_AVG', 'APARTMENTS_MODE', 'BASEMENTAREA_MODE',
'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_BUILD_MODE',
'COMMONAREA_MODE', 'ELEVATORS_MODE', 'ENTRANCES_MODE',
'FLOORSMAX_MODE', 'FLOORSMIN_MODE', 'LANDAREA_MODE',
'LIVINGAPARTMENTS_MODE', 'LIVINGAREA_MODE',
'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAREA_MODE', 'APARTMENTS_MEDI',
'BASEMENTAREA_MEDI', 'YEARS_BEGINEXPLUATATION_MEDI',
'YEARS_BUILD_MEDI', 'COMMONAREA_MEDI', 'ELEVATORS_MEDI',
'ENTRANCES_MEDI', 'FLOORSMAX_MEDI', 'FLOORSMIN_MEDI', 'LANDAREA_MEDI',
'LIVINGAPARTMENTS_MEDI', 'LIVINGAREA_MEDI', 'NONLIVINGAPARTMENTS_MEDI',
'NONLIVINGAREA_MEDI', 'TOTALAREA_MODE', 'OBS_30_CNT_SOCIAL_CIRCLE',
'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE',
'DEF_60_CNT_SOCIAL_CIRCLE', 'DAYS_LAST_PHONE_CHANGE',
'FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3', '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_DOCUMENT_17', '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'}
.

```

### Categorical Columns:

```

{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.}

```

Dataset size (rows columns, train, test, validation)

Application\_train : Rows: 307511, Columns: 122

Application\_test : Rows: 48744 , Columns: 121

Bureau : Rows: 1716428 , Columns: 17

Bureau Balance : Rows: 27299925 , Columns: 3

Credit Card Balance : Rows: 3840312 , Columns: 23

Installments Payments : Rows: 13605401 , Columns: 8

Previous Application : Rows: 1670214, Columns: 37

POS CASH Balance : Rows: 10001358 , Columns: 8

Validation: Rows: , Columns:

## Summary statistics

Application Test:

datasets["application_test"].describe(include='all') #Look at all categorical and numerical										
<b>count</b>	48744.000000	48744	48744	48744	48744	48744.000000	4.874400e+04	4.874400e+04	48720.000000	4
<b>unique</b>	NaN	2	2	2	2	NaN	NaN	NaN	NaN	
<b>top</b>	NaN	Cash loans	F	N	Y	NaN	NaN	NaN	NaN	
<b>freq</b>	NaN	48305	32678	32311	33658	NaN	NaN	NaN	NaN	
<b>mean</b>	277796.676350	NaN	NaN	NaN	NaN	0.397054	1.784318e+05	5.167404e+05	29426.240209	4
<b>std</b>	103169.547296	NaN	NaN	NaN	NaN	0.709047	1.015226e+04	3.653970e+05	16016.368315	3
<b>min</b>	100001.000000	NaN	NaN	NaN	NaN	0.000000	2.694150e+04	4.500000e+04	2295.000000	4
<b>25%</b>	188557.750000	NaN	NaN	NaN	NaN	0.000000	1.125000e+05	2.606400e+05	17973.000000	2
<b>50%</b>	277549.000000	NaN	NaN	NaN	NaN	0.000000	1.575000e+05	4.500000e+05	26199.000000	3
<b>75%</b>	367555.500000	NaN	NaN	NaN	NaN	1.000000	2.250000e+05	6.750000e+05	37390.500000	6
<b>max</b>	456250.000000	NaN	NaN	NaN	NaN	20.000000	4.410000e+06	2.245500e+06	180576.000000	2

11 rows × 121 columns

Application Train:

datasets["application_train"].describe(include='all') #Look at all categorical and numerical										
<b>count</b>	307511.000000	307511.000000	307511	307511	307511	307511	307511.000000	3.075110e+05	3.075110e+05	307499
<b>unique</b>	NaN	NaN	2	3	2	2	NaN	NaN	NaN	NaN
<b>top</b>	NaN	NaN	Cash loans	F	N	Y	NaN	NaN	NaN	NaN
<b>freq</b>	NaN	NaN	278232	202448	202924	213312	NaN	NaN	NaN	NaN
<b>mean</b>	278180.518577	0.080729	NaN	NaN	NaN	NaN	0.417052	1.687979e+05	5.990260e+05	27108
<b>std</b>	102790.175348	0.272419	NaN	NaN	NaN	NaN	0.722121	2.371231e+05	4.024908e+05	14493
<b>min</b>	100002.000000	0.000000	NaN	NaN	NaN	NaN	0.000000	2.565000e+04	4.500000e+04	1615
<b>25%</b>	189145.500000	0.000000	NaN	NaN	NaN	NaN	0.000000	1.125000e+05	2.700000e+05	16524
<b>50%</b>	278202.000000	0.000000	NaN	NaN	NaN	NaN	0.000000	1.471500e+05	5.135310e+05	24903
<b>75%</b>	367142.500000	0.000000	NaN	NaN	NaN	NaN	1.000000	2.025000e+05	8.086500e+05	34596
<b>max</b>	456255.000000	1.000000	NaN	NaN	NaN	NaN	19.000000	1.170000e+08	4.050000e+06	258025

11 rows × 122 columns

## Correlation with the target column

```
correlations = datasets["application_train"].corr()['TARGET'].sort_values()
print('Most Positive Correlations:\n', correlations.tail(10))
print('\nMost Negative Correlations:\n', correlations.head(10))

Most Positive Correlations:
FLAG_DOCUMENT_3           0.044346
REG_CITY_NOT_LIVE_CITY    0.044395
FLAG_EMP_PHONE             0.045982
REG_CITY_NOT_WORK_CITY     0.050994
DAYS_ID_PUBLISH            0.051457
DAYS_LAST_PHONE_CHANGE     0.055218
REGION_RATING_CLIENT        0.058899
REGION_RATING_CLIENT_W_CITY 0.060893
DAYS_BIRTH                  0.078239
TARGET                      1.000000
Name: TARGET, dtype: float64

Most Negative Correlations:
EXT_SOURCE_3                -0.178919
EXT_SOURCE_2                 -0.160472
EXT_SOURCE_1                 -0.155317
DAYS_EMPLOYED                 -0.044932
FLOORSMAX_AVG                 -0.044003
FLOORSMAX_MEDI                 -0.043768
FLOORSMAX_MODE                 -0.043226
AMT_GOODS_PRICE                 -0.039645
REGION_POPULATION_RELATIVE      -0.037227
ELEVATORS_AVG                 -0.034199
Name: TARGET, dtype: float64
```

### Observations from the above data:

1. Maximum positive correlation for the TARGET feature is 0.078239 which is observed with the DAYS\_BIRTH feature.
2. Following this are the features REGION\_RATING\_CLIENT\_W\_CITY, REGION\_RATING\_CLIENT, DAYS\_LAST\_PHONE\_CHANGE, DAYS\_ID\_PUBLISH,DAYS\_ID\_PUBLISH.
3. Another observation is a high value of indirect correlation between TARGET and FLOORS features,AMT\_GOODS\_PRICE and relative population features .

## Applicant's House Wall Material Type

```
ap_train_data['WALLSMATERIAL_MODE'].value_counts()
```

Panel	66040
Stone, brick	64815
Block	9253
Wooden	5362
Mixed	2296
Monolithic	1779
Others	1625

Name: WALLSMATERIAL\_MODE, dtype: int64

## Applicant's House Type Part 2

```
ap_train_data['HOUSETYPE_MODE'].value_counts()
```

block of flats	150503
specific housing	1499
terraced house	1212

Name: HOUSETYPE\_MODE, dtype: int64

## Applicants already own Realty?

```
ap_train_data['FLAG_OWN_REALTY'].value_counts()
```

Y	213312
N	94199

Name: FLAG\_OWN\_REALTY, dtype: int64

## For previous Application DB:

### Intersection of SK\_ID\_CURR for Previous Application and application\_train

```
#Find the intersection of two arrays.
print(f'Number of train applicants with previous applications is {len(np.intersect1d(datasets["previous_application"]["SK_ID_CURR"], datasets["application_train"]["SK_ID_CURR"]))}')
Number of train applicants with previous applications is 291,057
```

### Intersection of SK\_ID\_CURR for Previous Application and application\_train

```
#Find the intersection of two arrays.
print(f'Number of train applicants with previous applications is {len(np.intersect1d(datasets["previous_application"]的文化_ID_CURR"], datasets["application_test"]的文化_SK_ID_CURR"])}
Number of train applicants with previous applications is 47,800
```

# Visual EDA:

## Missing data for application train

```
percent = (datasets["application_train"].isnull().sum()/datasets["application_train"].count()*100).sort_values(ascending = False).round(2)
sum_missing = datasets["application_train"].isna().sum().sort_values(ascending = False)
missing_application_train_data = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Train Missing Count"])
missing_application_train_data.head(20)
```

	Percent	Train Missing Count
COMMONAREA_MEDI	69.87	214865
COMMONAREA_AVG	69.87	214865
COMMONAREA_MODE	69.87	214865
NONLIVINGAPARTMENTS_MODE	69.43	213514
NONLIVINGAPARTMENTS_AVG	69.43	213514
NONLIVINGAPARTMENTS_MEDI	69.43	213514
FONDKAPREMONT_MODE	68.39	210295
LIVINGAPARTMENTS_MODE	68.35	210199
LIVINGAPARTMENTS_AVG	68.35	210199
LIVINGAPARTMENTS_MEDI	68.35	210199
FLOORSMIN_AVG	67.85	208642
FLOORSMIN_MODE	67.85	208642
FLOORSMIN_MEDI	67.85	208642
YEARS_BUILD_MEDI	66.50	204488
YEARS_BUILD_MODE	66.50	204488
YEARS_BUILD_AVG	66.50	204488
OWN_CAR_AGE	65.99	202929
LANDAREA_MEDI	59.38	182590
LANDAREA_MODE	59.38	182590
LANDAREA_AVG	59.38	182590

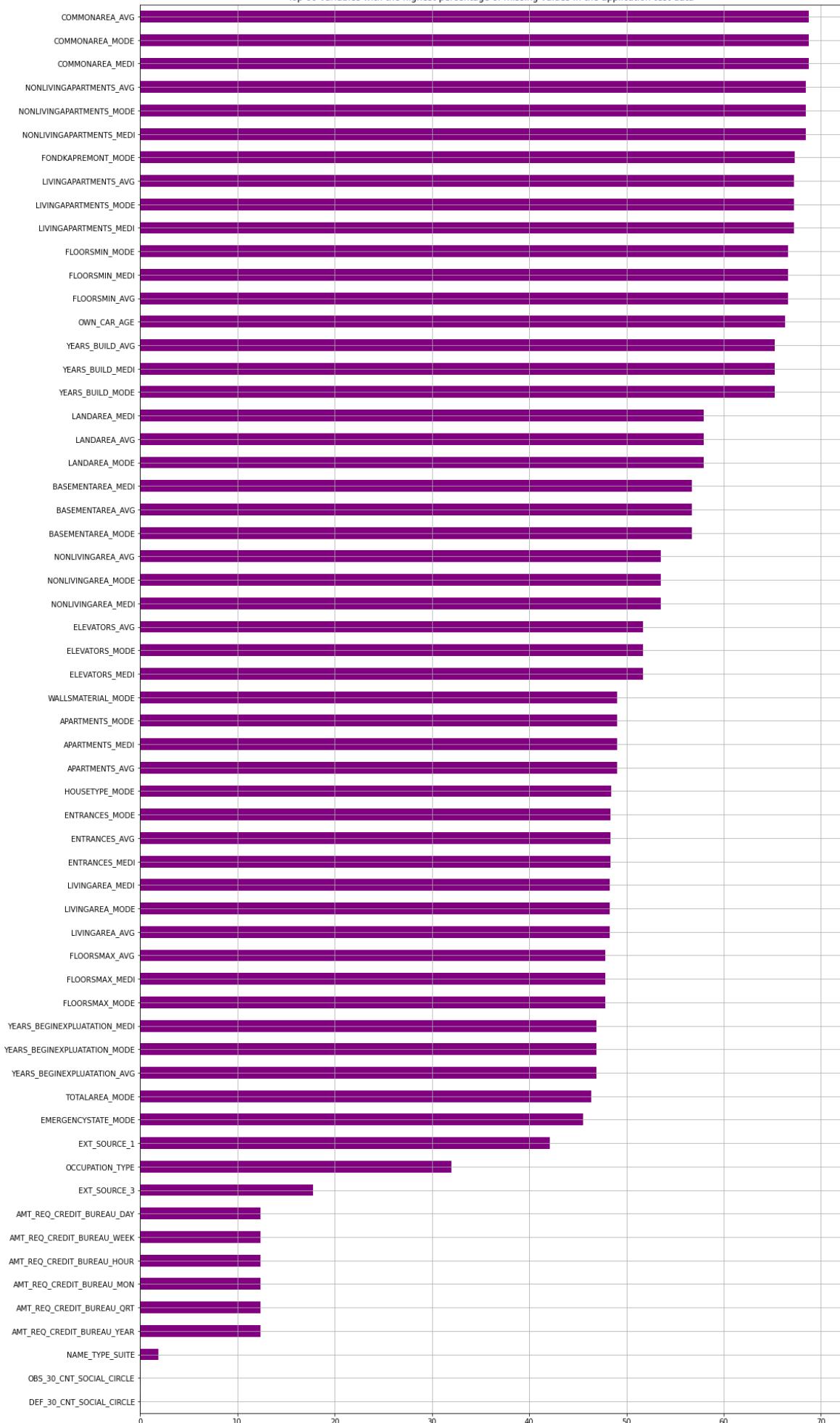
## Missing data for application test

```
percent = (datasets["application_test"].isnull().sum()/datasets["application_test"].count())*100.sort_values(ascending = False).round(2)
sum_missing = datasets["application_test"].isna().sum().sort_values(ascending = False)
missing_application_test_data = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Test Missing Count"])
missing_application_test_data.head(20)
```

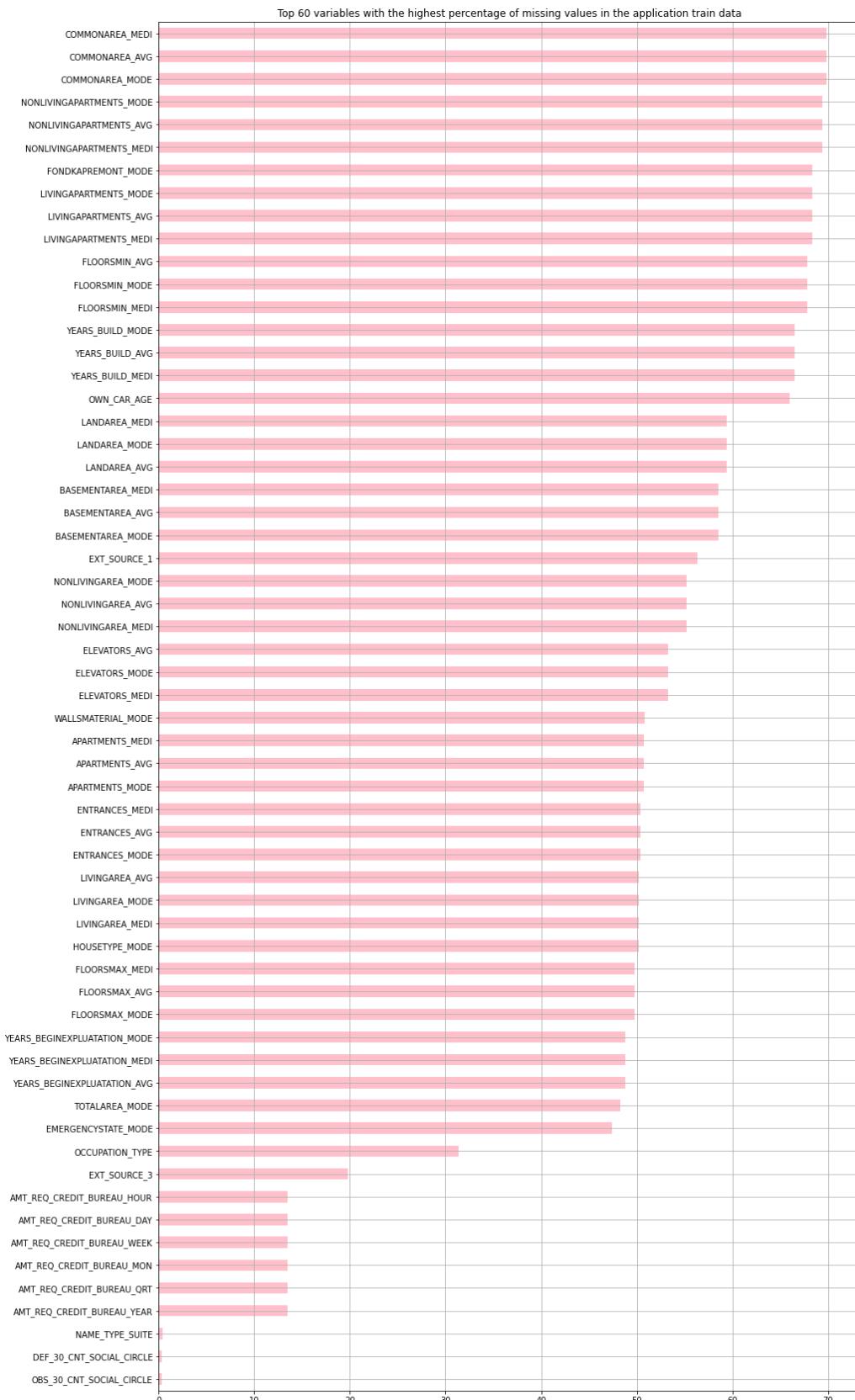
	Percent	Test Missing Count
COMMONAREA_AVG	68.72	33495
COMMONAREA_MODE	68.72	33495
COMMONAREA_MEDI	68.72	33495
NONLIVINGAPARTMENTS_AVG	68.41	33347
NONLIVINGAPARTMENTS_MODE	68.41	33347
NONLIVINGAPARTMENTS_MEDI	68.41	33347
FONDKAPREMONT_MODE	67.28	32797
LIVINGAPARTMENTS_AVG	67.25	32780
LIVINGAPARTMENTS_MODE	67.25	32780
LIVINGAPARTMENTS_MEDI	67.25	32780
FLOORSMIN_MEDI	66.61	32466
FLOORSMIN_AVG	66.61	32466
FLOORSMIN_MODE	66.61	32466
OWN_CAR_AGE	66.29	32312
YEARS_BUILD_AVG	65.28	31818
YEARS_BUILD_MEDI	65.28	31818
YEARS_BUILD_MODE	65.28	31818
LANDAREA_MEDI	57.96	28254
LANDAREA_AVG	57.96	28254
LANDAREA_MODE	57.96	28254

**Top 60 variables with the highest percentage of missing values in the application test data**

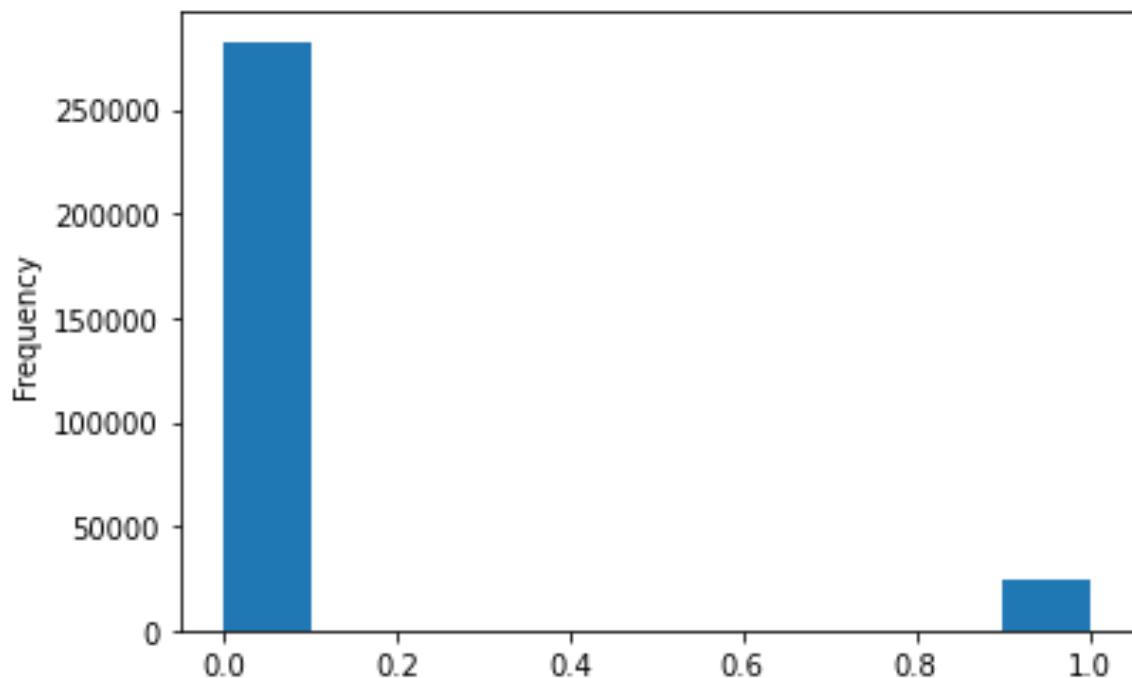
Top 60 variables with the highest percentage of missing values in the application test data



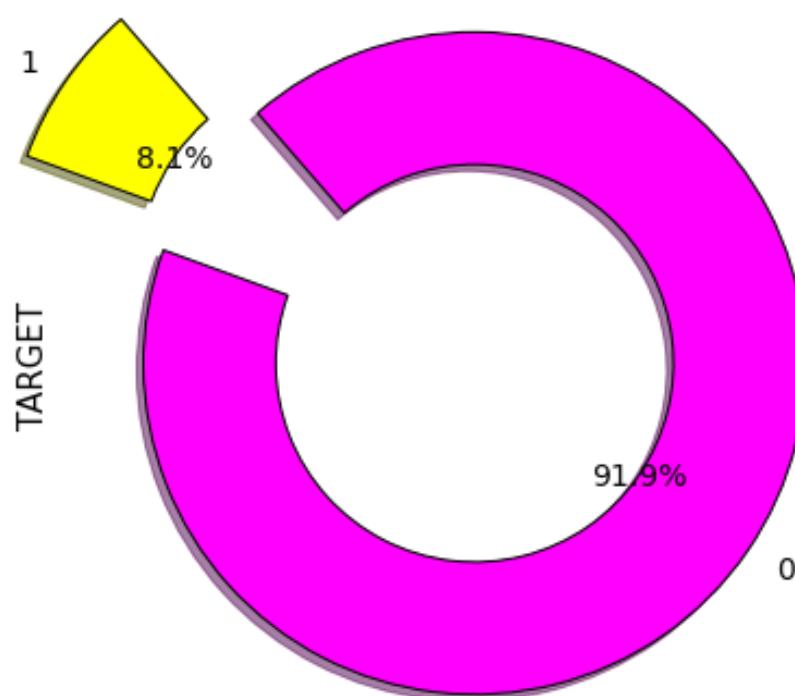
## Top 60 variables with the highest percentage of missing values in the application train data



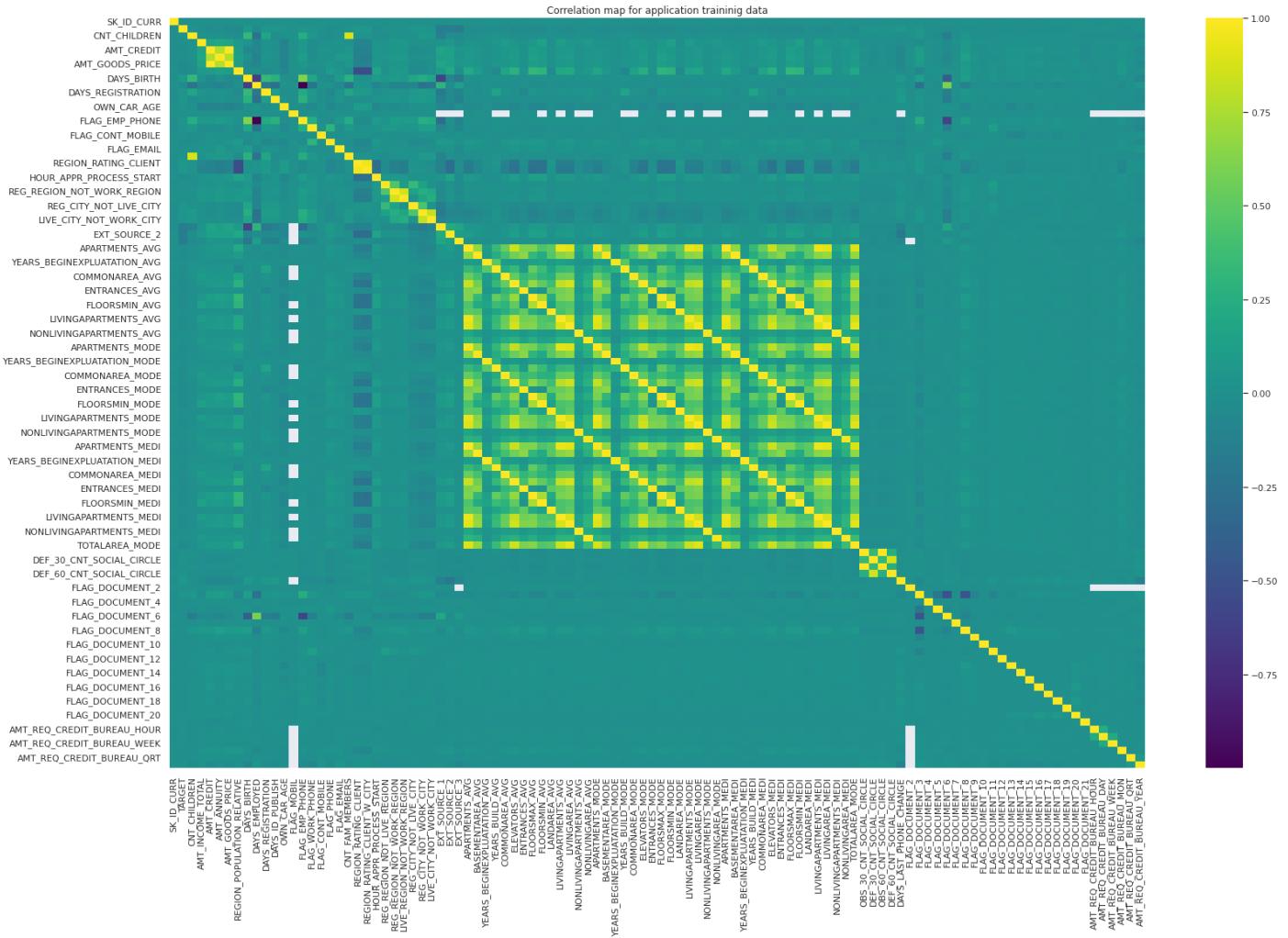
## Distribution of the target column



Plot for distribution of TARGET feature

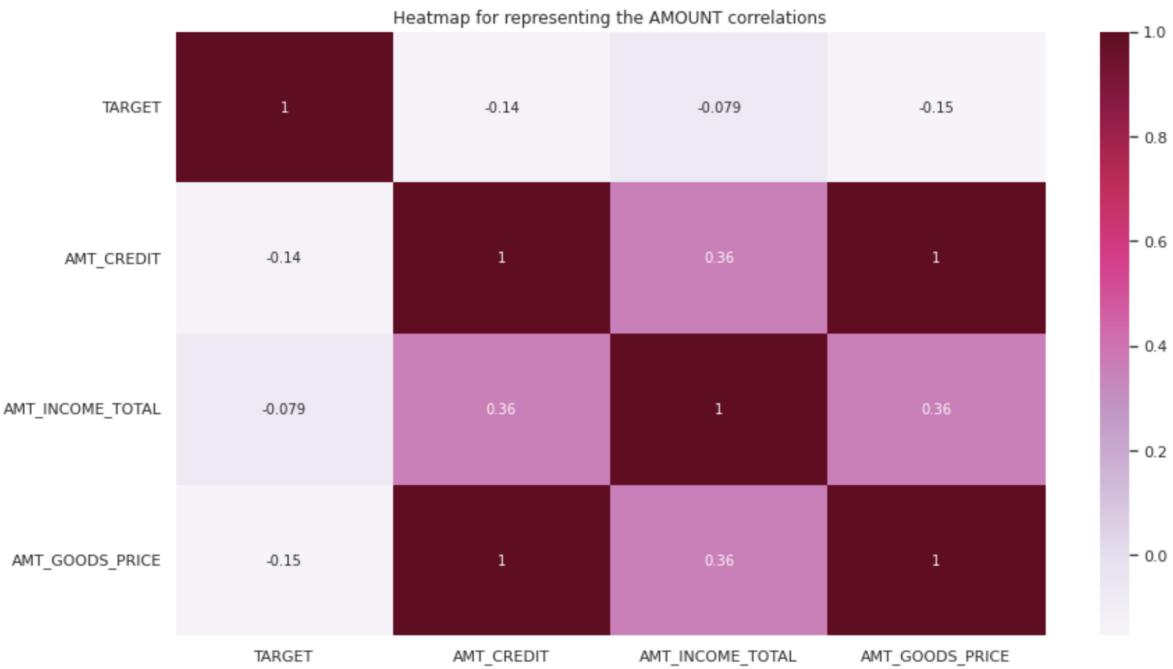


Correlation with the Application Train Data columns



**Observations from the above plot:** -The heatmap is a bit ambiguous to read as we have 122 columns to compare from.

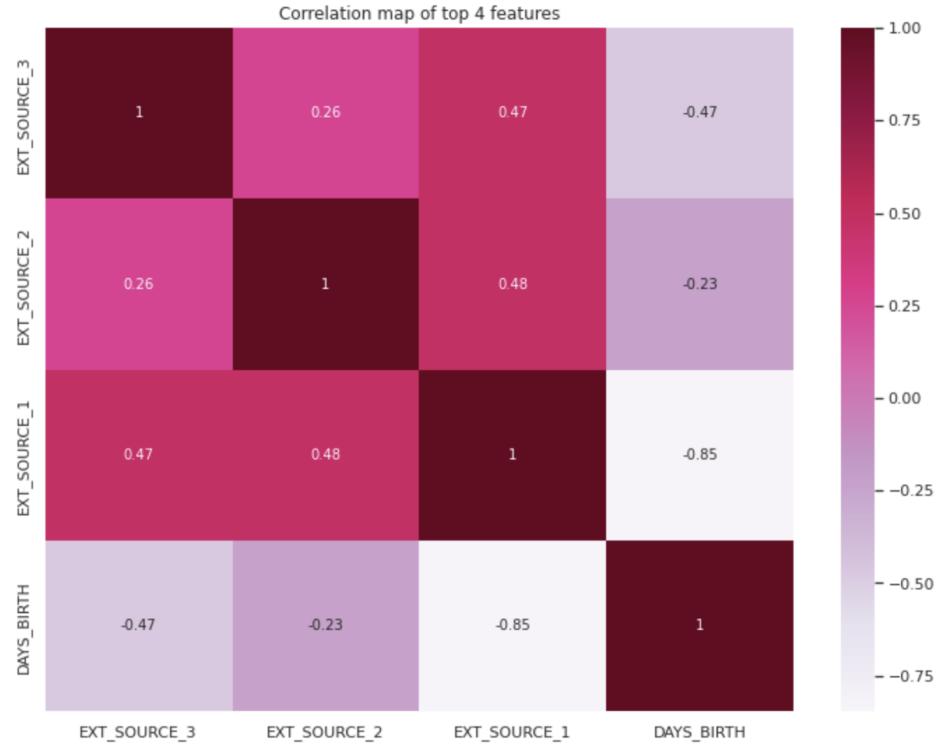
## Correlation Map for target column with the Amount Columns



### Observations from the above data: -

The above plot is the correlation between the target column and the amount columns, with the plot we can clearly depict from the plot that target column has very low correlation with the amount columns.

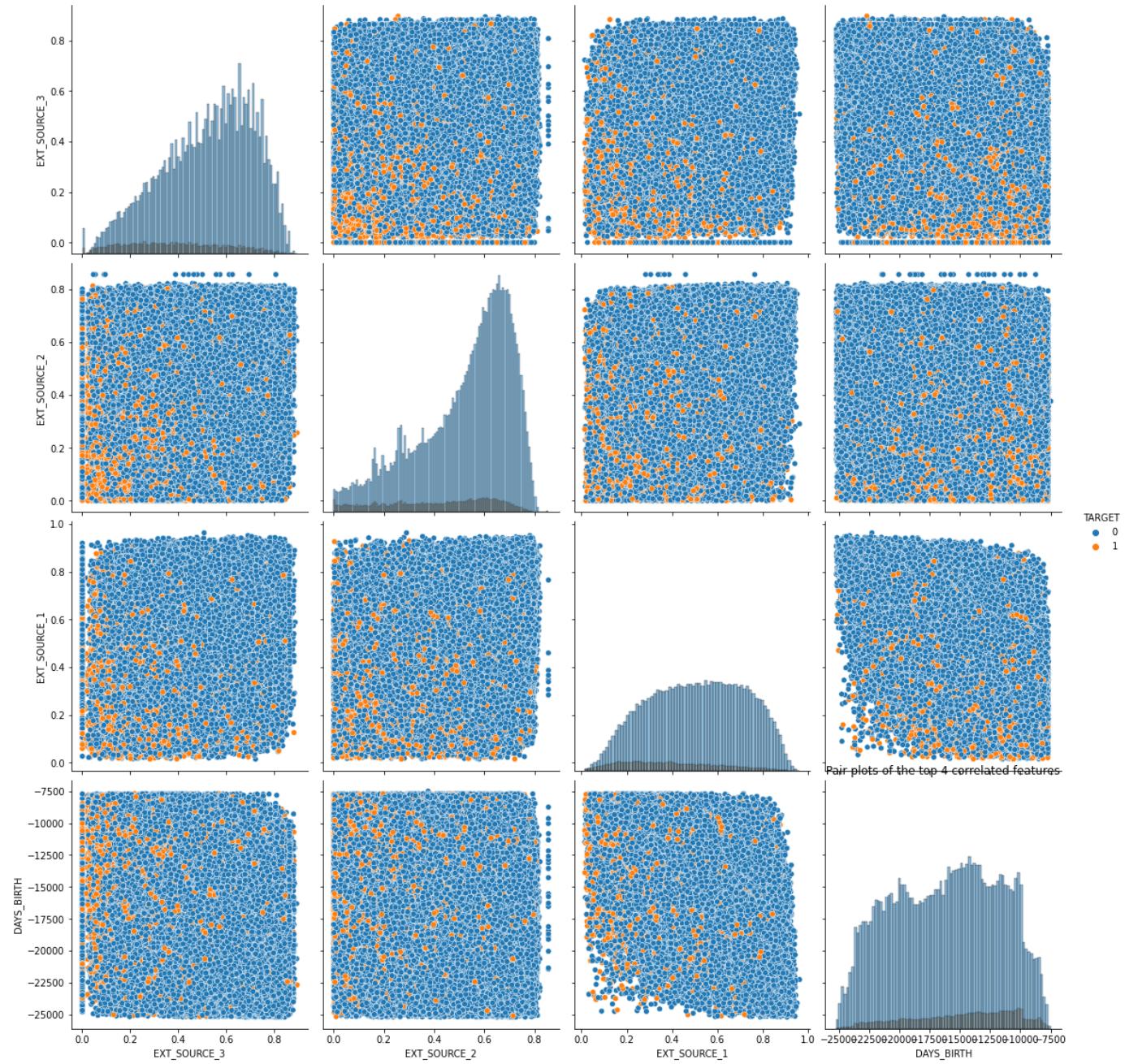
### Correlation Map for top 4 features:



### Observation based on above

- The heatmap demonstrates how extrinsic factors have a secondary impact on the TARGET characteristic.
- But, it is also clear from the correlations between them that multicollinearity exists.

## Pair plots of the top 4 correlated features

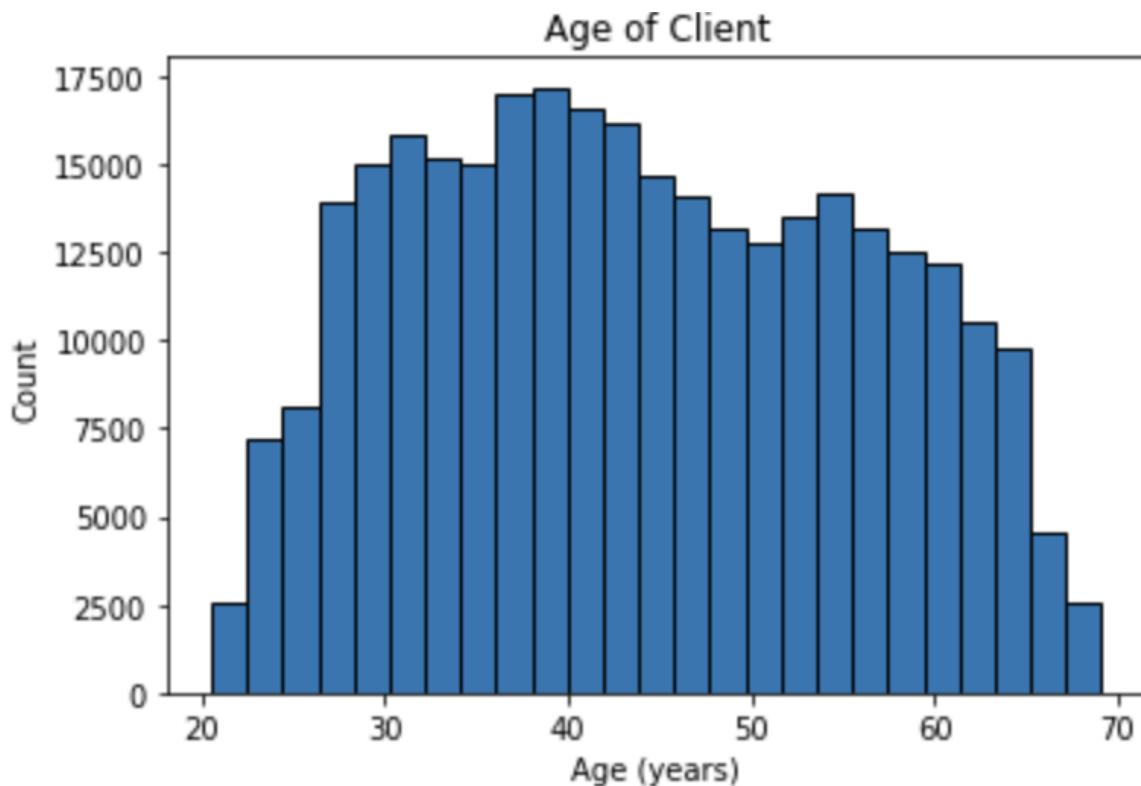


## Observation based on above

- The pair plot matrix demonstrates how extrinsic factors have a secondary impact on the TARGET characteristic, which is used \

- But, it is also clear from the correlations between them that multicollinearity exists.

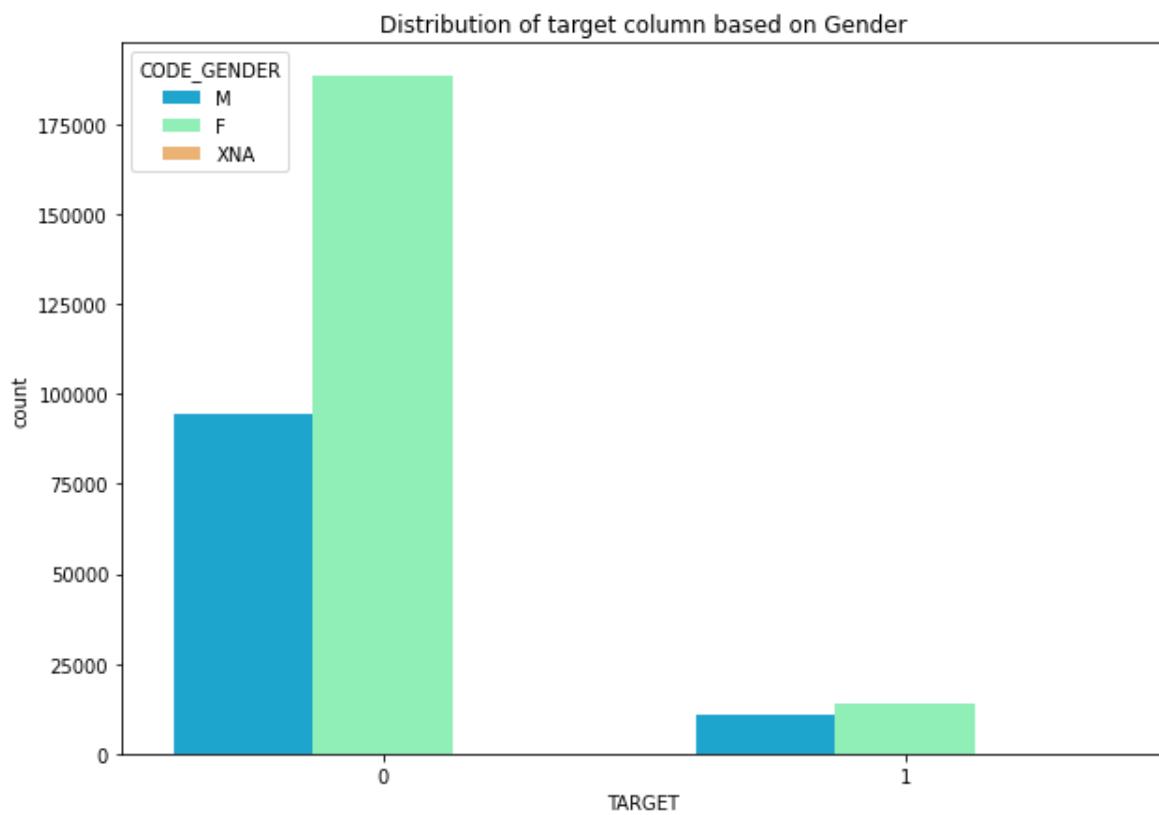
## Distribution for Applicants Age



### Observations from the above plot

- The DAYS\_BIRTH feature, which contains negative values, is used to calculate age. This is inconsistent and needs to be fixed.
- When we plot age as a function of years, we observe a pretty normal distribution, which is encouraging in a challenging dataset where the DAYS BIRTH feature is substantially linked with the TARGET feature.

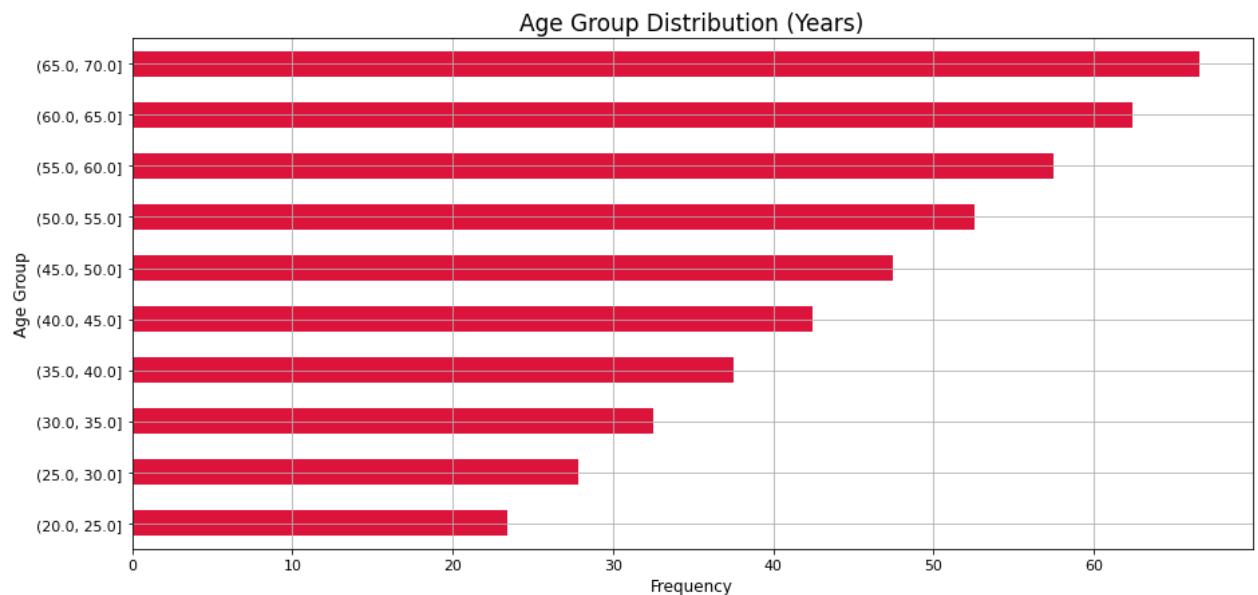
## Distribution for Target Column based on Gender



## Observation from the above plot

The purpose of this code is to group the 'YEARS\_BIRTH' column into different age groups and analyze the distribution of 'TARGET' variable in each age group.

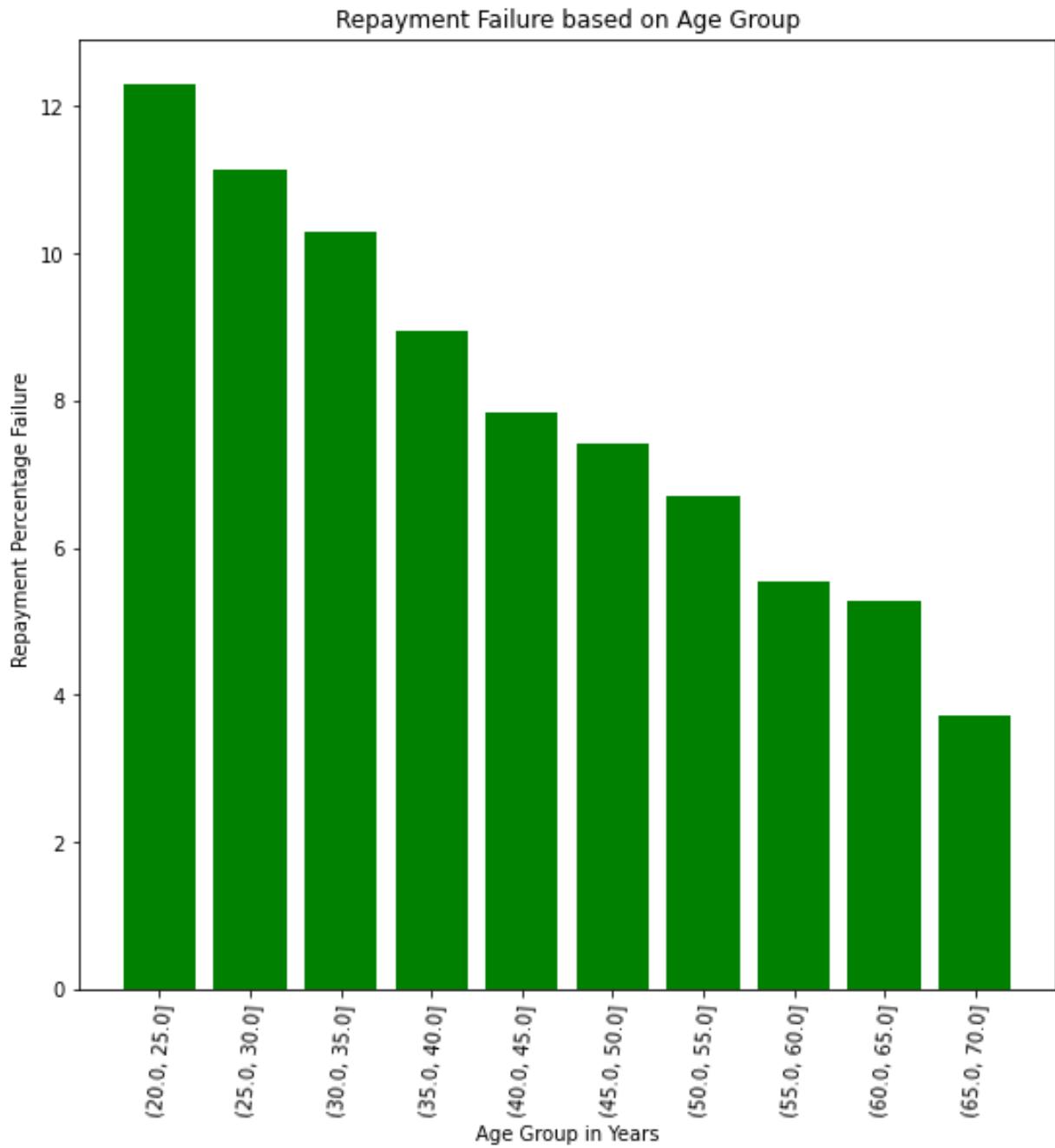
## Age Group Distribution (Years)



### Observation from the above plot-

Following binning, we see that older persons have a propensity to take out more loans than younger people.

## Repayment Failure based on Age Group

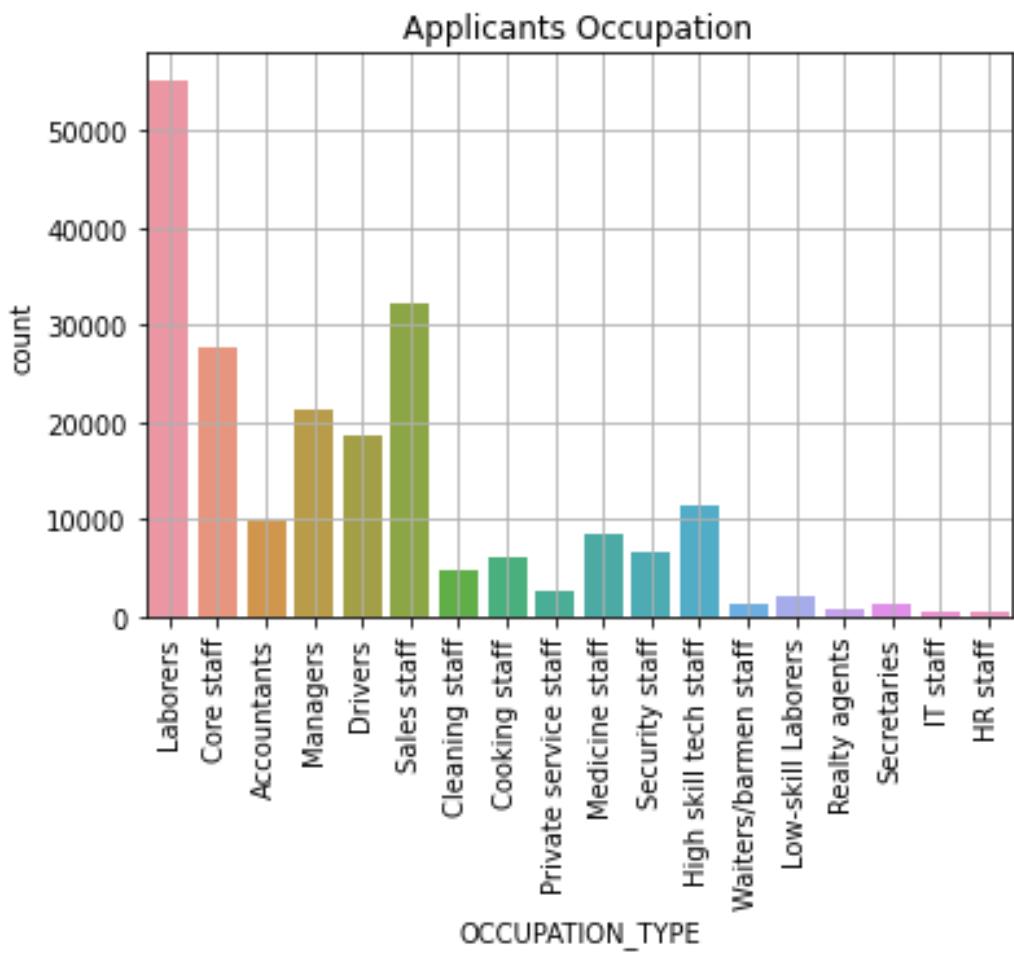


**Observations from the above plot:** The Binning method is utilized to provide precise information regarding age.

Now, it can generally be deduced that the age range of 20 to 25 years is where the most majority of people who are essentially unable to essentially repay the loan on time for the most part reside. Contrary to popular opinion, a decline in the number of failures is typically observed as age

increases. Younger applicants are actually more likely to default on the loan, showing that it can now be generally deduced that the majority of borrowers who normally are unable to return the loan on time type of fall into the age range of 20–25 years, which is actually quite substantial. Younger applicants are generally more likely to default on a loan, so banks should generally be advised against doing so. This proves that, contrary to popular belief, the age group of 20 to 25 years is actually where the majority of borrowers fail to repay loans in a timely manner.

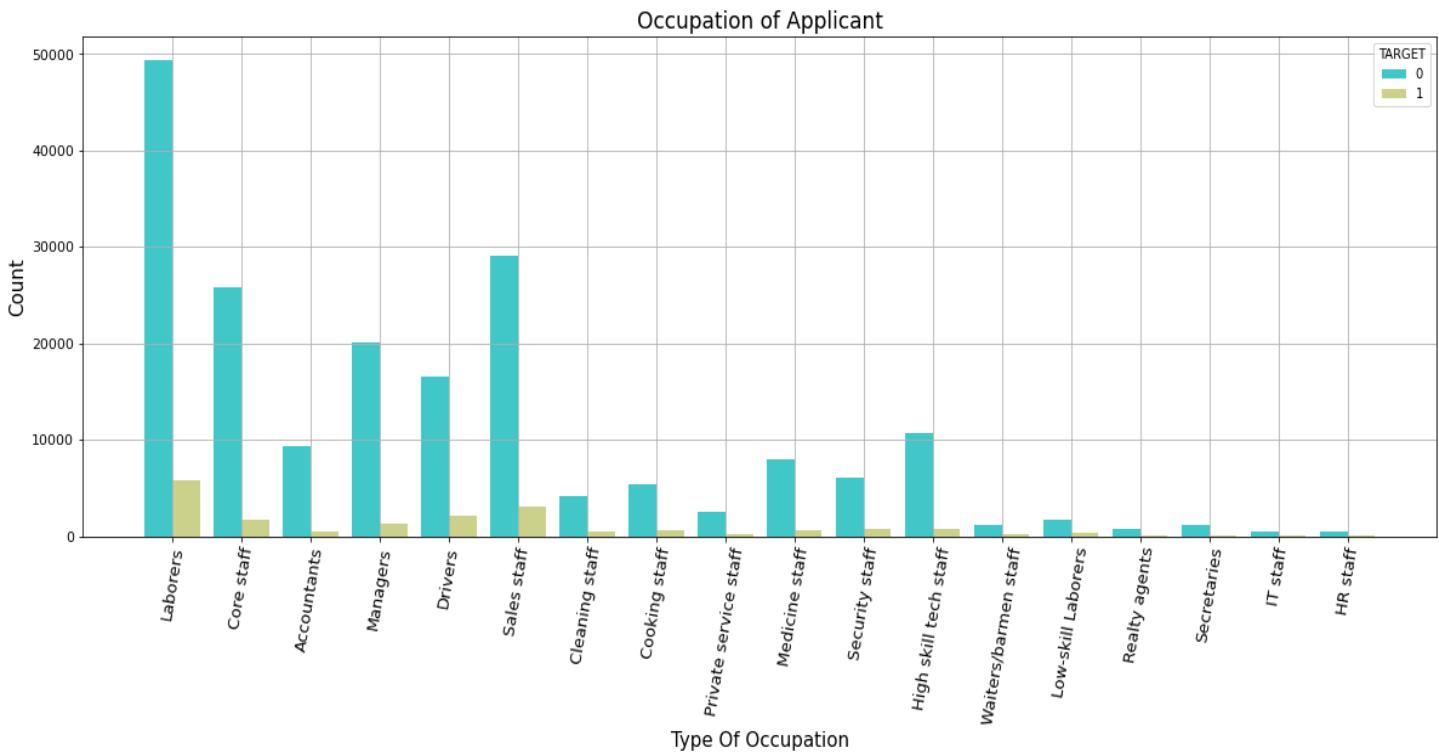
## Applicants occupations



## Observation based on plot:

Contrary to popular opinion, the majority of the candidates' occupations are studied here. It is generally accepted that laborers kind of tend to kind of take the loan most fundamentally followed by sales staff, demonstrating how this is generally accepted that laborers kind of tend to kind of take the loan most fundamentally followed by sales staff in a subtle way.

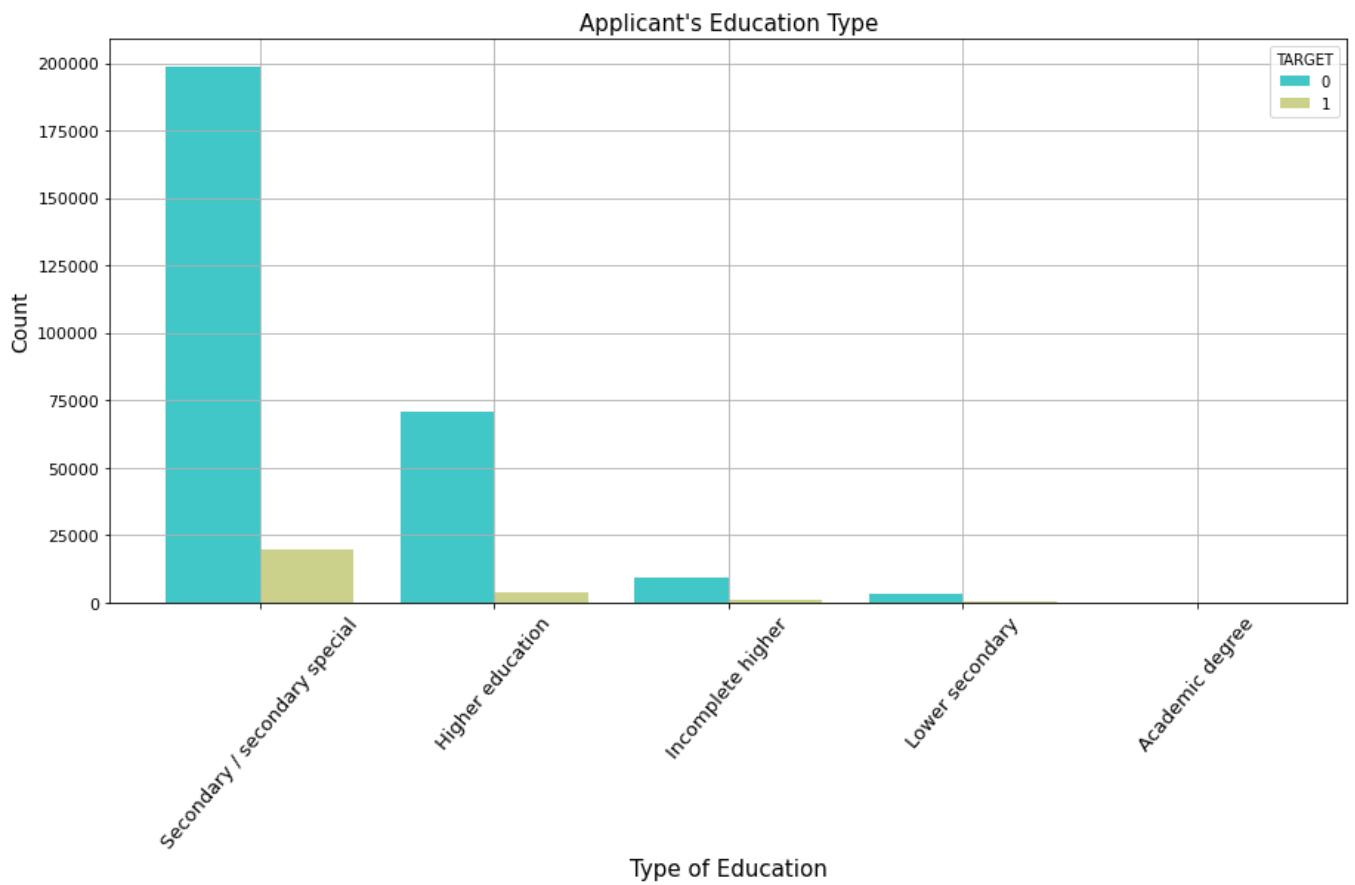
## Occupation of Applicant:



## Observation of the above plot

- There are 18 distinct occupations represented among the borrowers, with laborers, salespeople, core employees, managers, and drivers at the top.
- There is no clear pattern in the occupation classes whose borrowers effectively repay their loans.

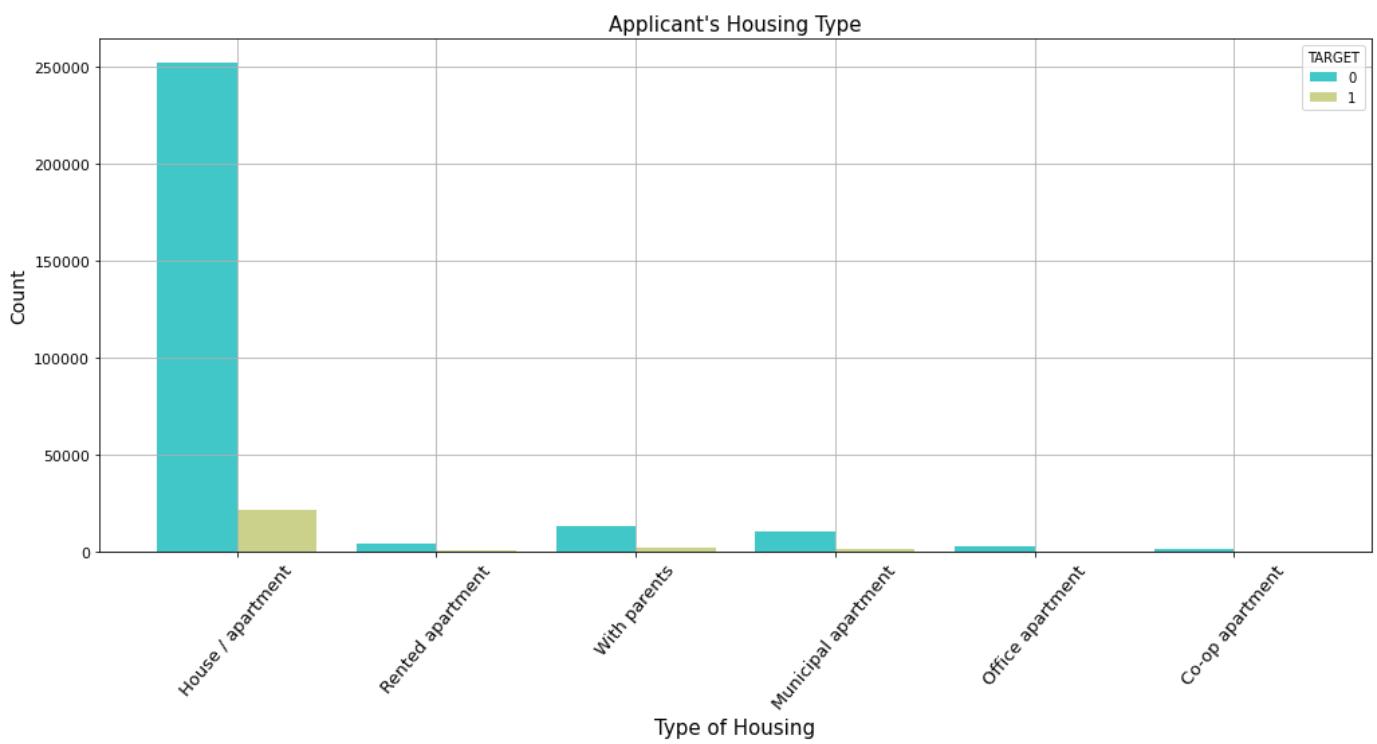
## Applicant's Education Type



### **Observation from the plot**

- The majority of candidates have the greatest levels of secondary and higher education, with academic degrees being the lowest.

## **Applicant Housing Type versus TARGET**

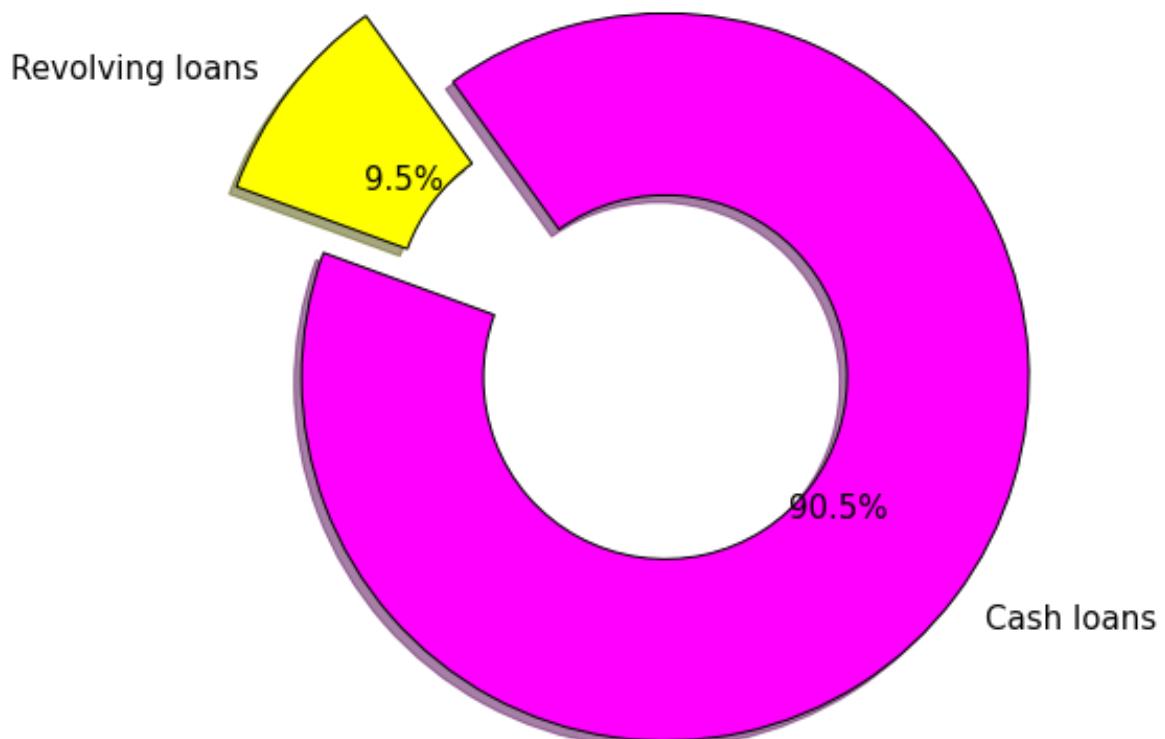


### **Observation from the plot**

- The applicant now lives in houses or apartments the majority of the time, followed by municipal apartments and living with parents while looking for a new place for themselves.

### **Applicant Contract Type**

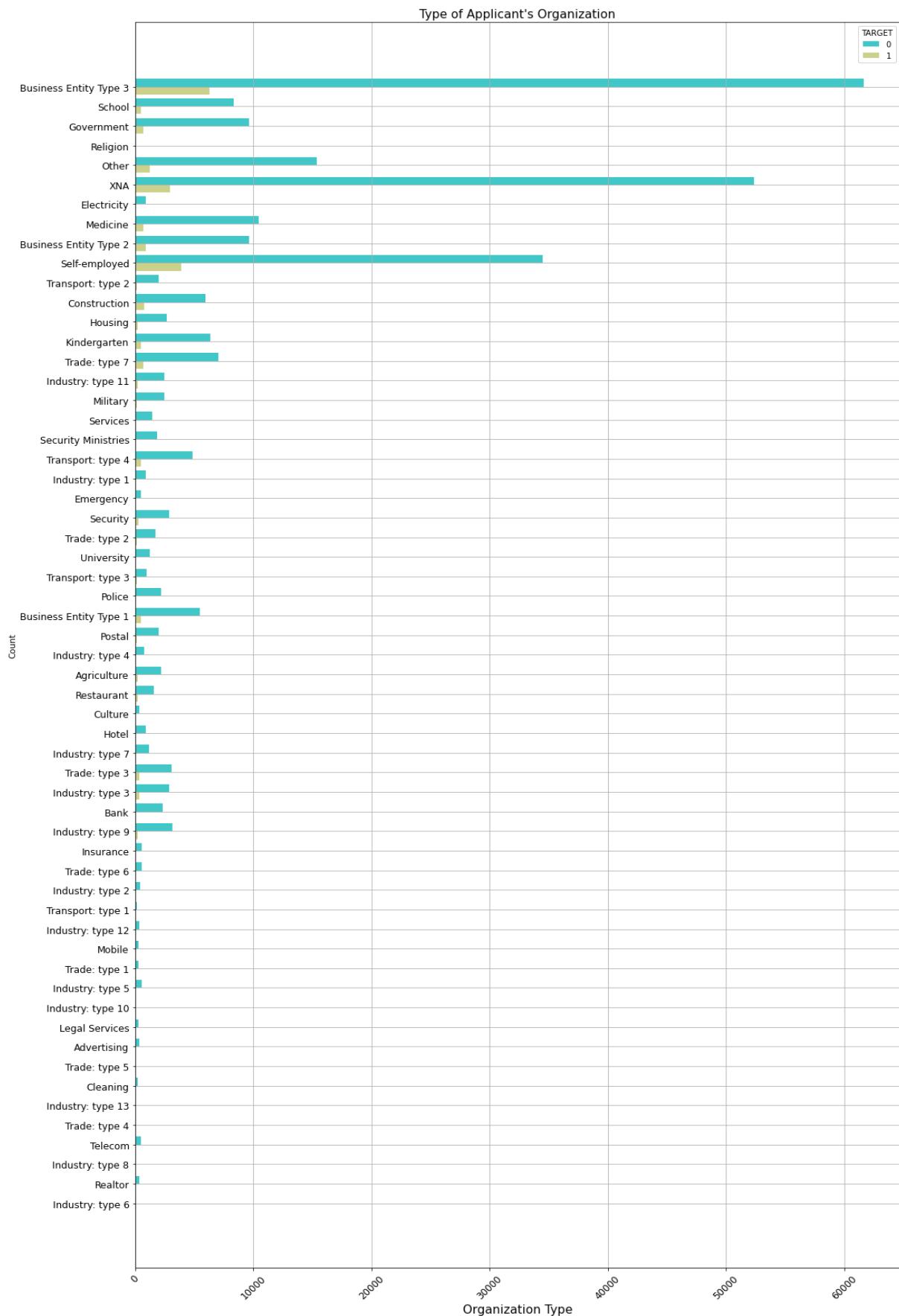
Applicant's Contract Type



**Observation from the plot**

- Cash loans makeup 90.5% of all loan contracts, while revolving loans, which can be repaid and refinanced repeatedly, account for the remaining 9.5%.

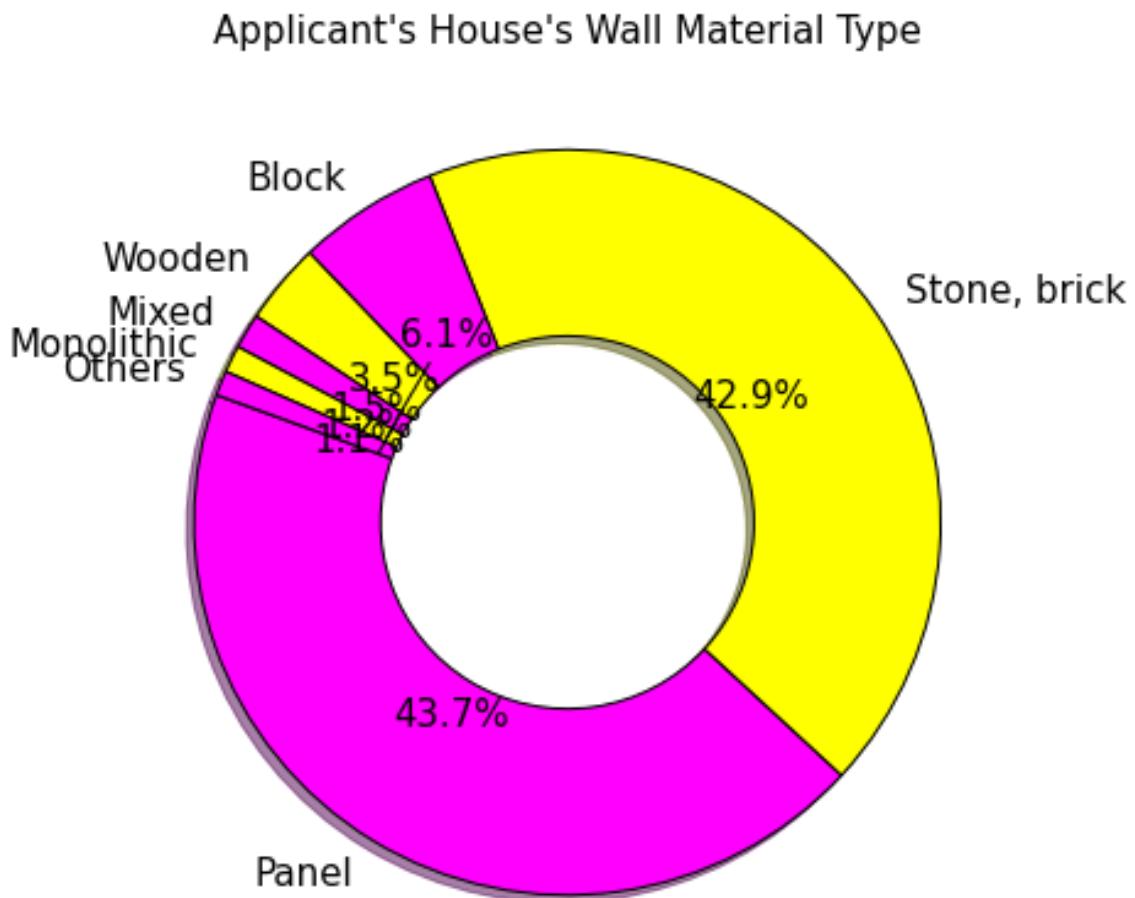
# Applicant Organization Type



### **Observation from the plot**

- The application pool includes a variety of organization kinds, with the majority coming from Type 3 commercial entities and self-employment.

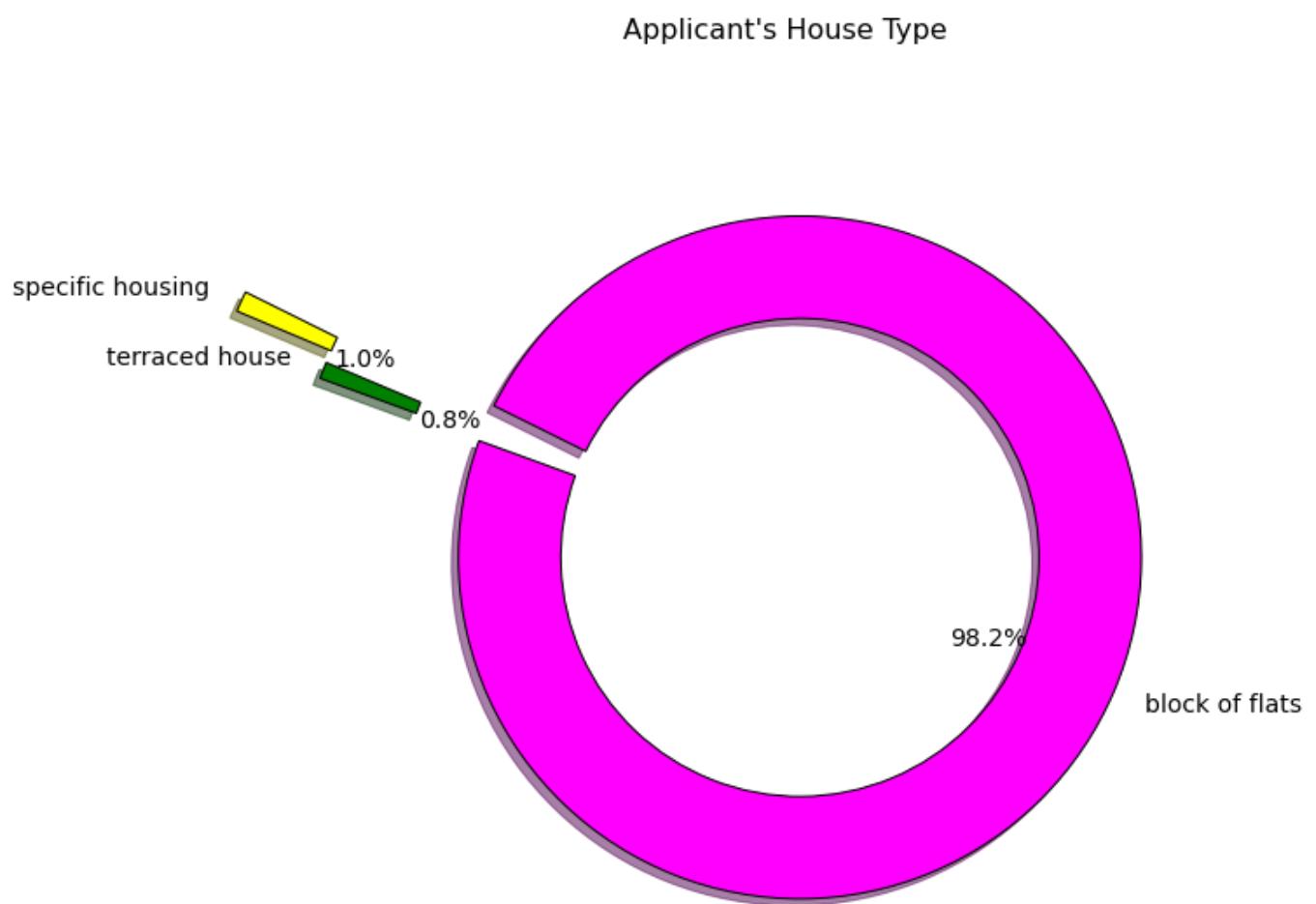
## **Applicant's House Wall Material Type**



### **Observation from the plot**

- The most common wall materials in applicant's homes are panels, stones, and bricks, followed by cement blocks, wood, or a combination of the aforementioned.

## Applicant's House Type Part 2

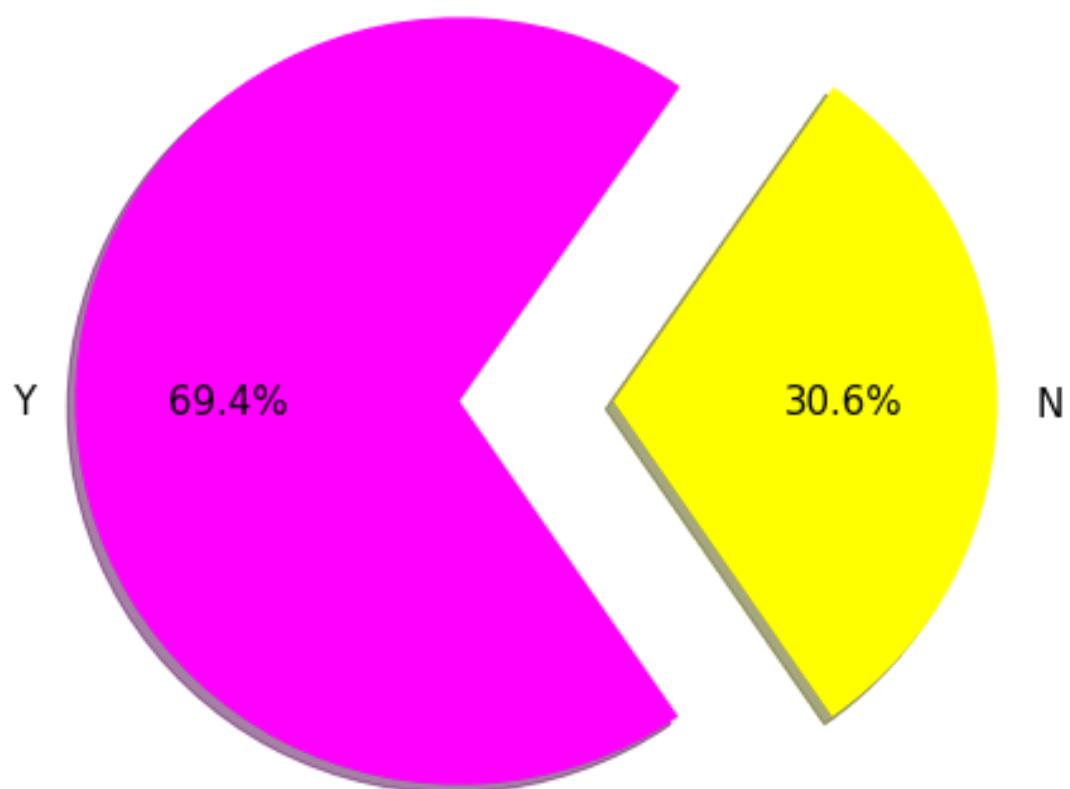


### Observation from the plot

- Applicants mostly reside in flats (more than 98%) while the remaining either live in terraced or other specific house types.

## Applicant already own Realty?

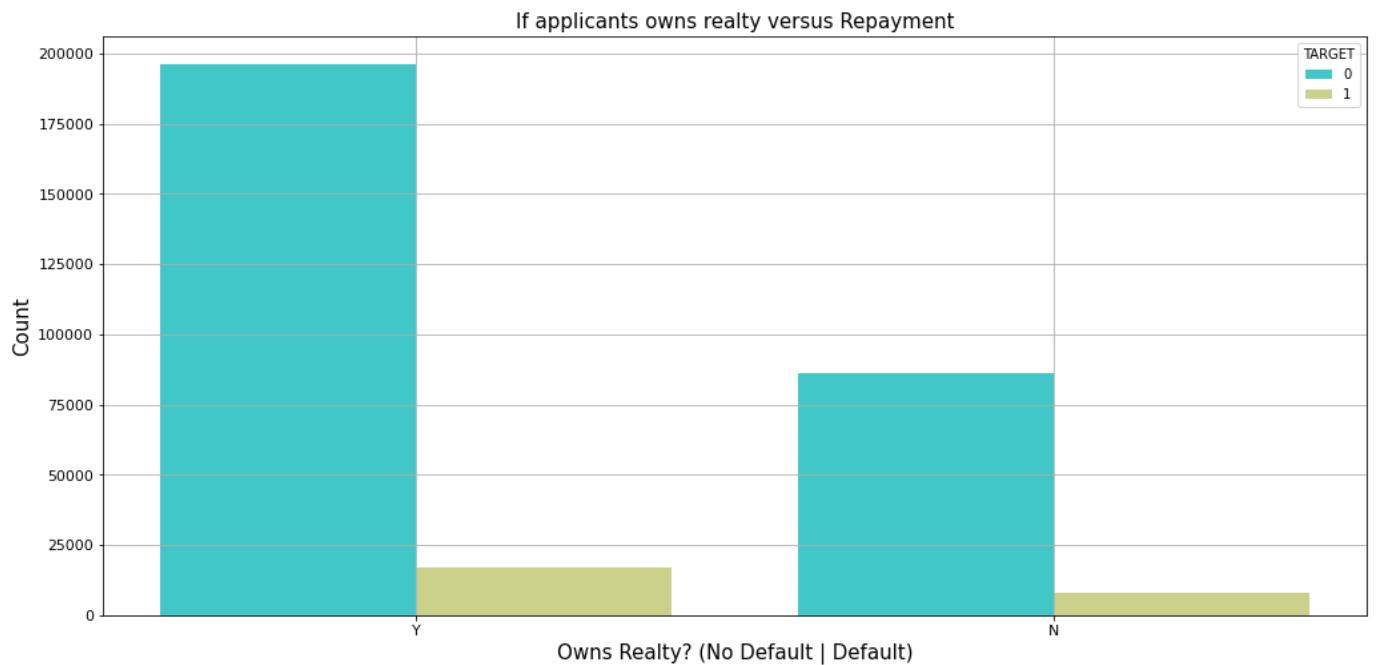
Does the applicant already own a Realty?



### Observation from the plot

- 69.4% of applicants have a home of their own. Check out the repayment's distribution.

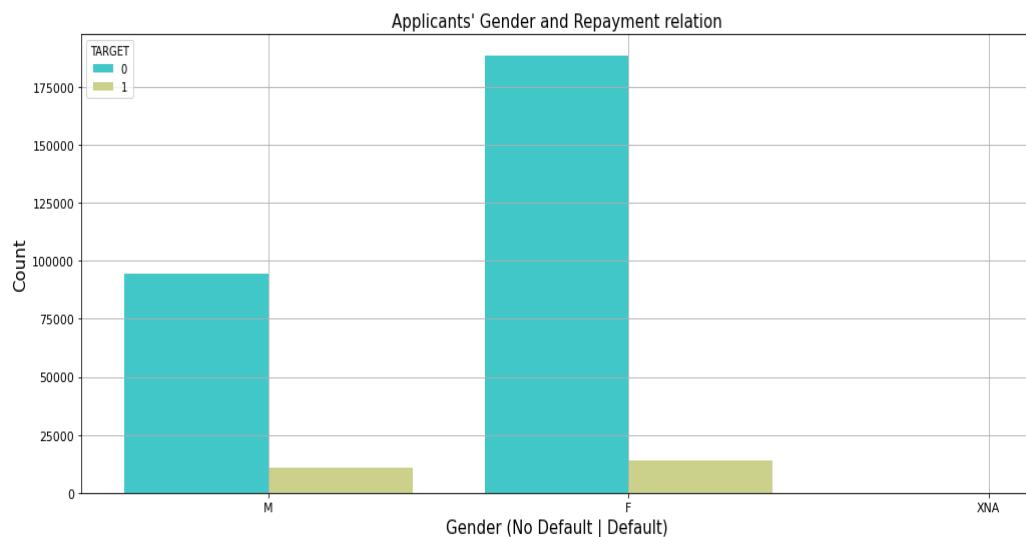
## If applicants owns realty versus Repayment



### Observation from the above plot:

-The majority of applicants in either class are not defaulters. Less than 25000 applicants have real estate and are behind on their payments.

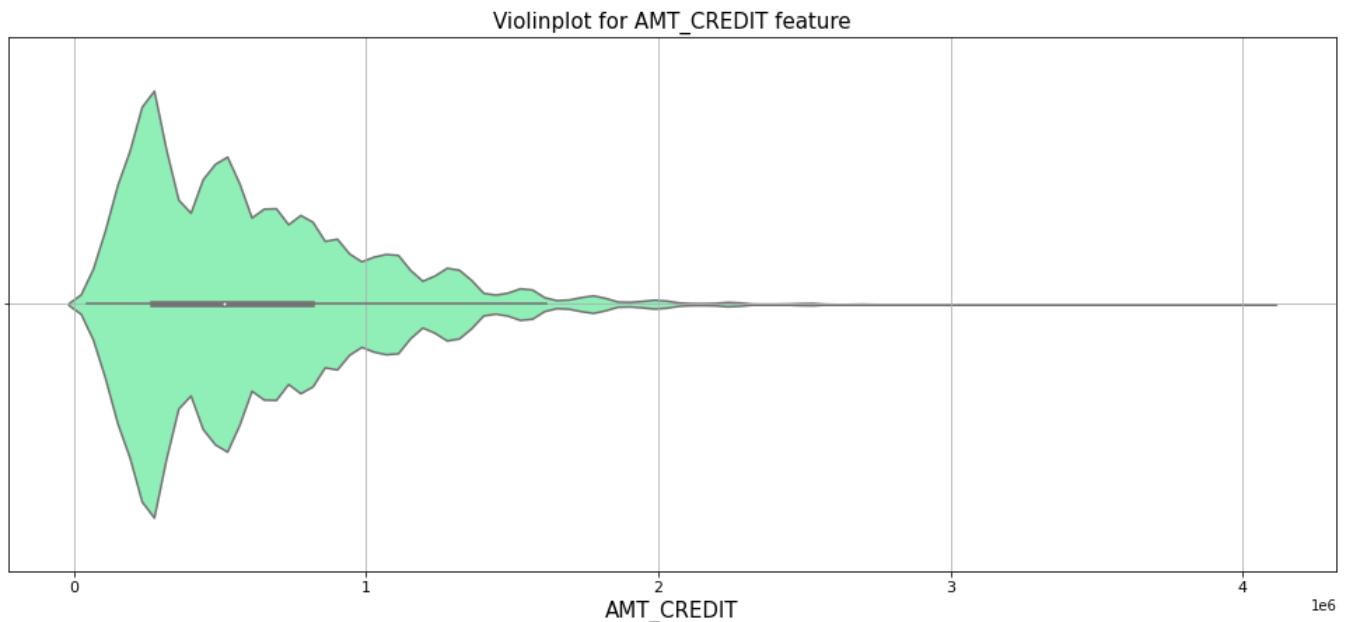
## Gender wise repayment analysis



### Observations from the above plot:

- Female applicants make up the majority, and the majority of them have no history of defaults.
- In the case of male candidates, it is clear that a disproportionately higher percentage of applicants default.

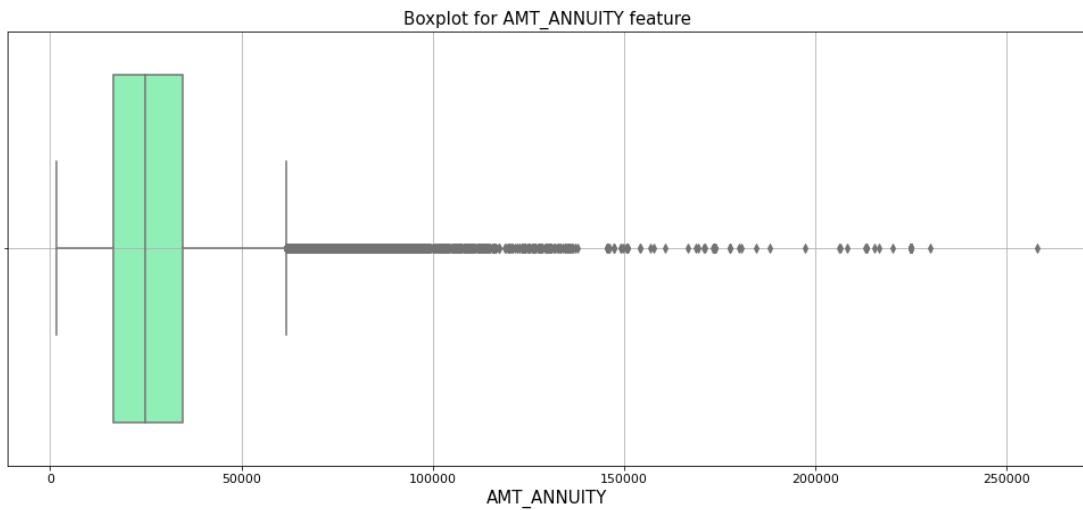
## **AMT\_CREDIT feature distribution Plot**



### **Observations from the plot:**

- The feature is right-skewed, as is evident.
- Sizing could aid in implementing this functionality effectively.

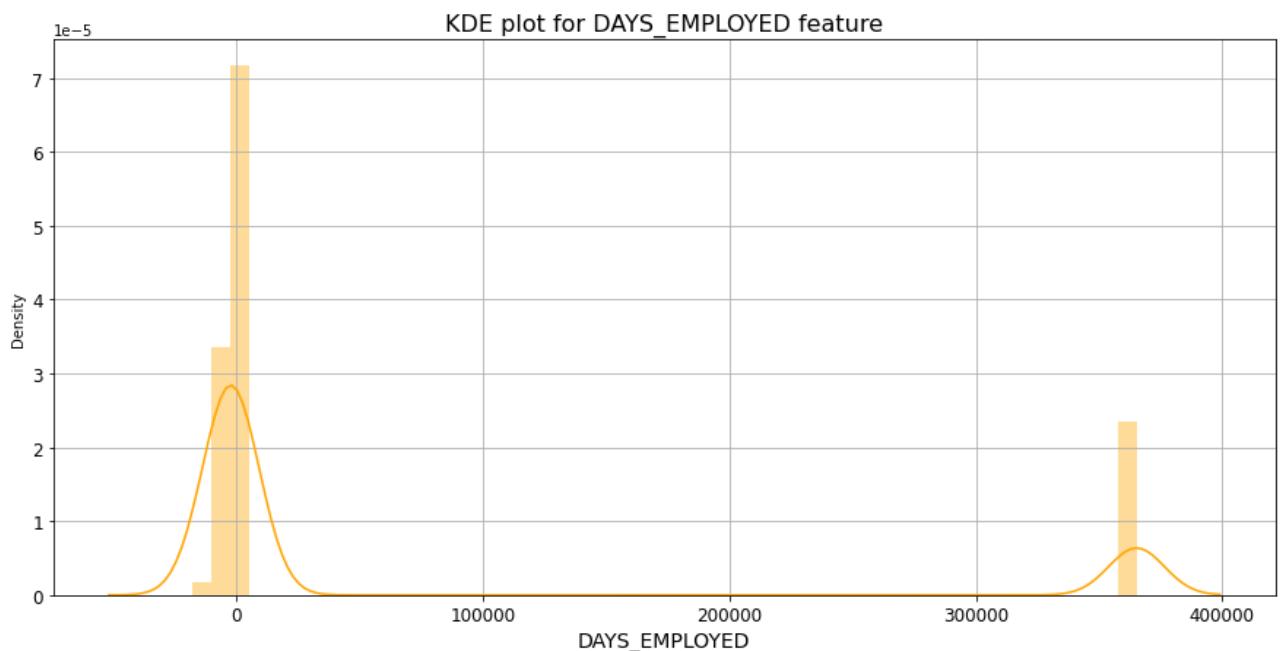
## **AMT\_ANNUITY feature distribution Plot**



### **Observations from the plot**

- A right-skewed feature with numerous outliers is once again visible.
- We can't get rid of these outliers because doing so could mean losing crucial data.

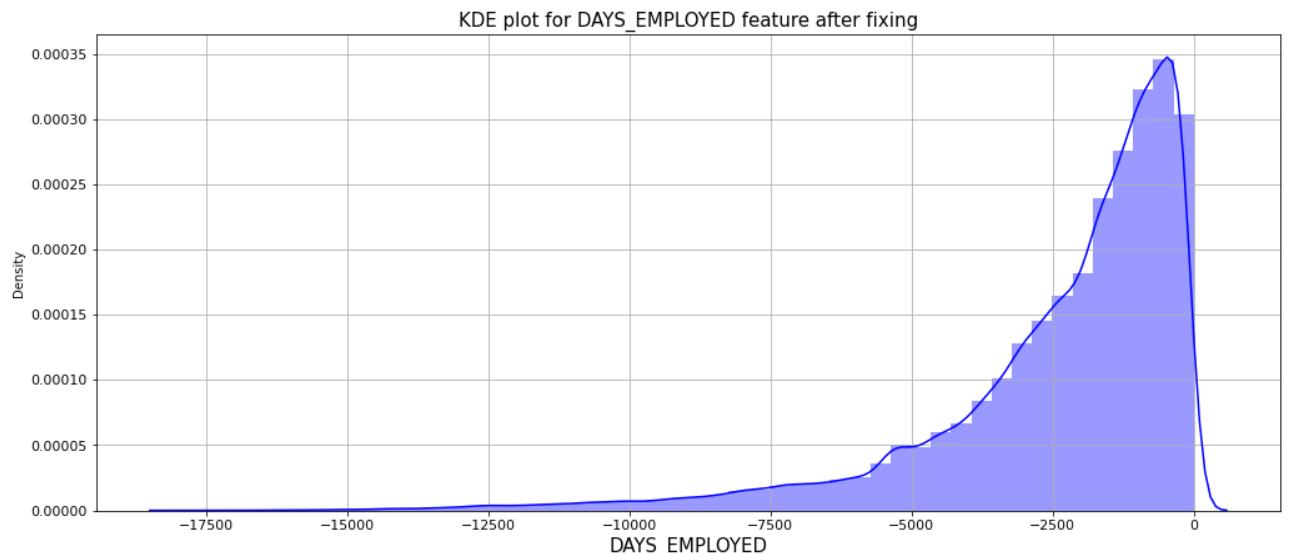
## **DAYS\_EMPLOYED feature distribution Plot**



### Observations from the plot

- The negative days values for this feature are the same as those for DAYS\_BIRTH.
- But, we notice a strange anomaly in this situation: the maximum number of days that can be worked is 365243, or a thousand years.
- We won't analyze the anomaly, we'll just ignore it and go ahead and examine the feature distribution once more.

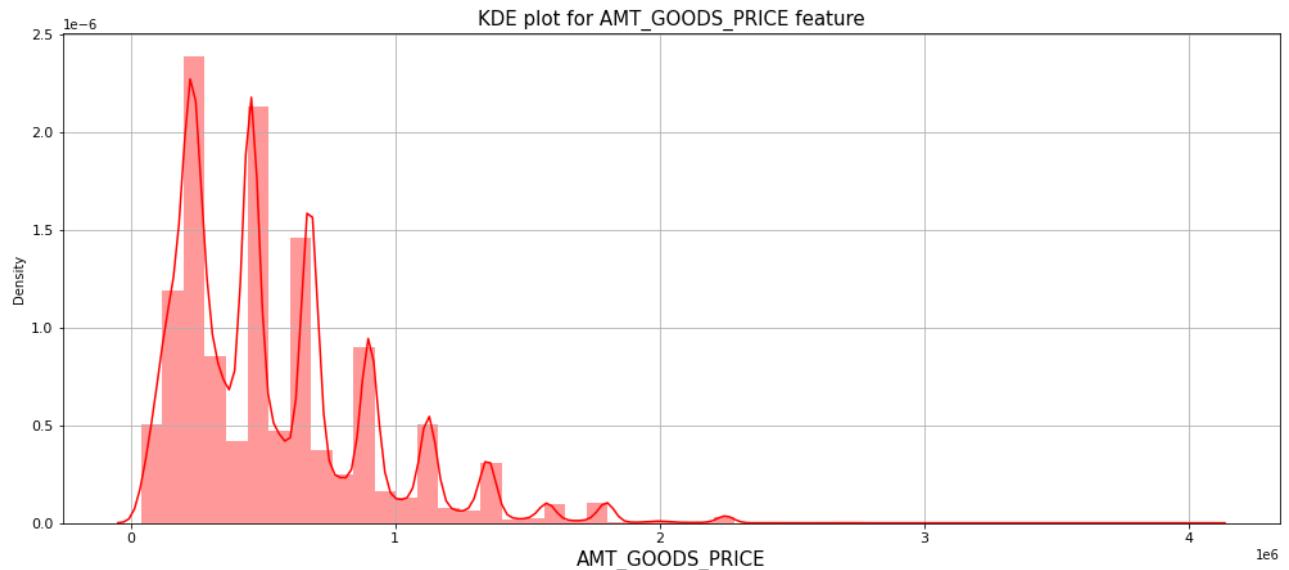
## Distribution of DAYS\_EMPLOYED feature after removing the inconsistent value



### Observations from the plot

- In this graphic, we see left-skewed data that would become right-skewed if the days were flipped to the positive side

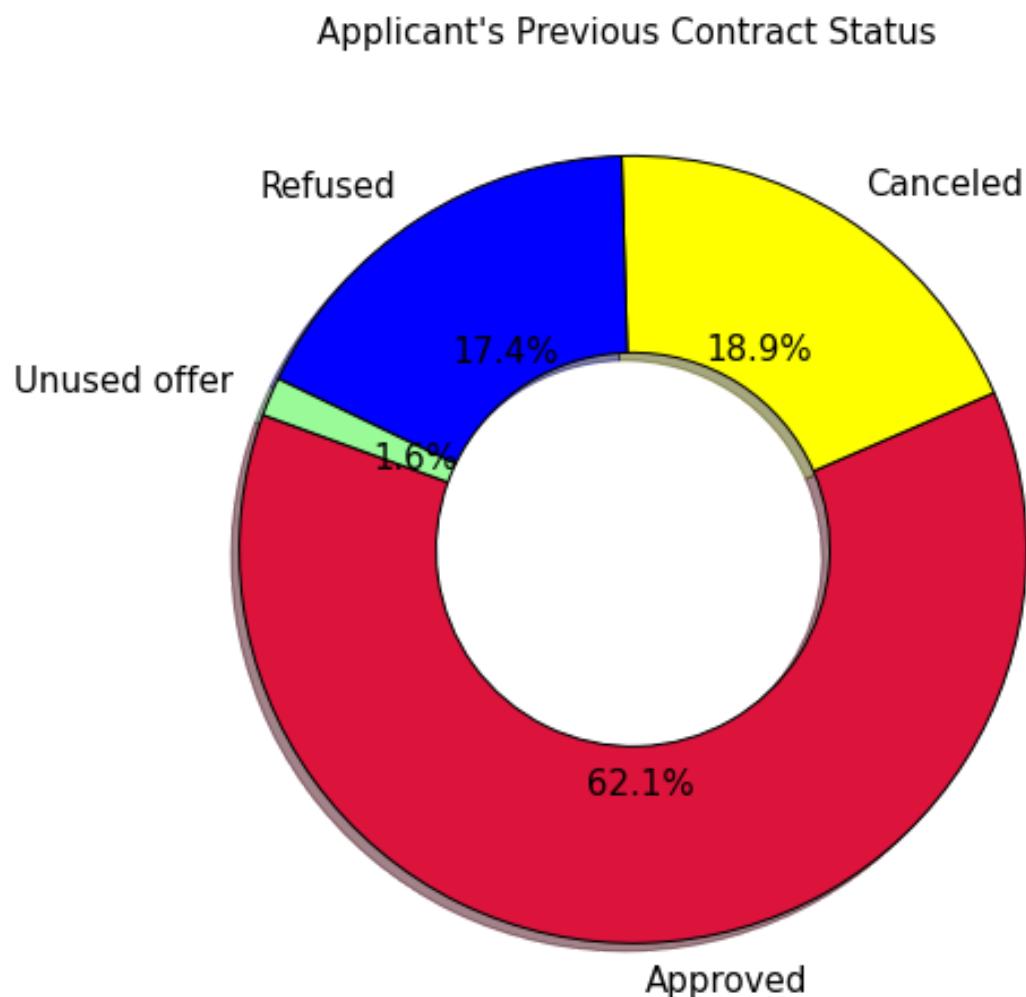
## Distribution of AMT\_GOODS\_PRICE



### Observations from the plot

- We see yet another multi-modal skewed distribution.
- Binning might make better use of this functionality.

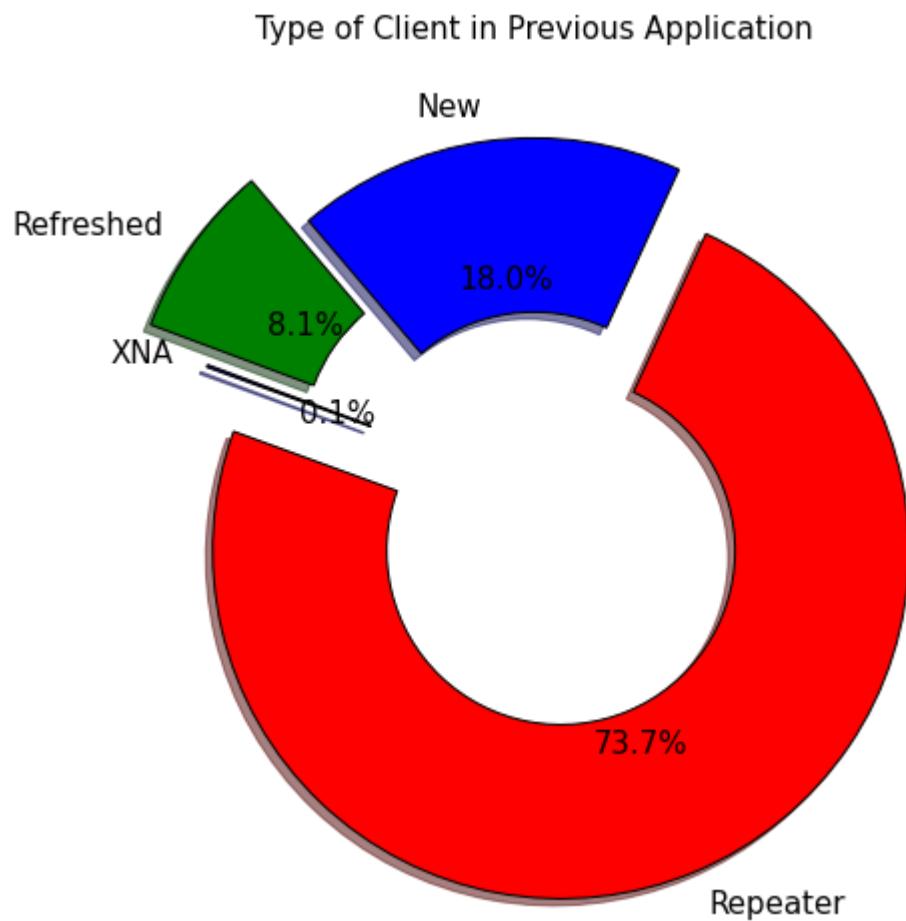
## **Applicant's Previous Contract Status from Prev Application DB:**



### **Observation from above**

- The majority of the applicants had their contracts approved in prior applications.
- 36% of candidates had their contracts refused or terminated, and the remaining 1.6% never used their contracts.

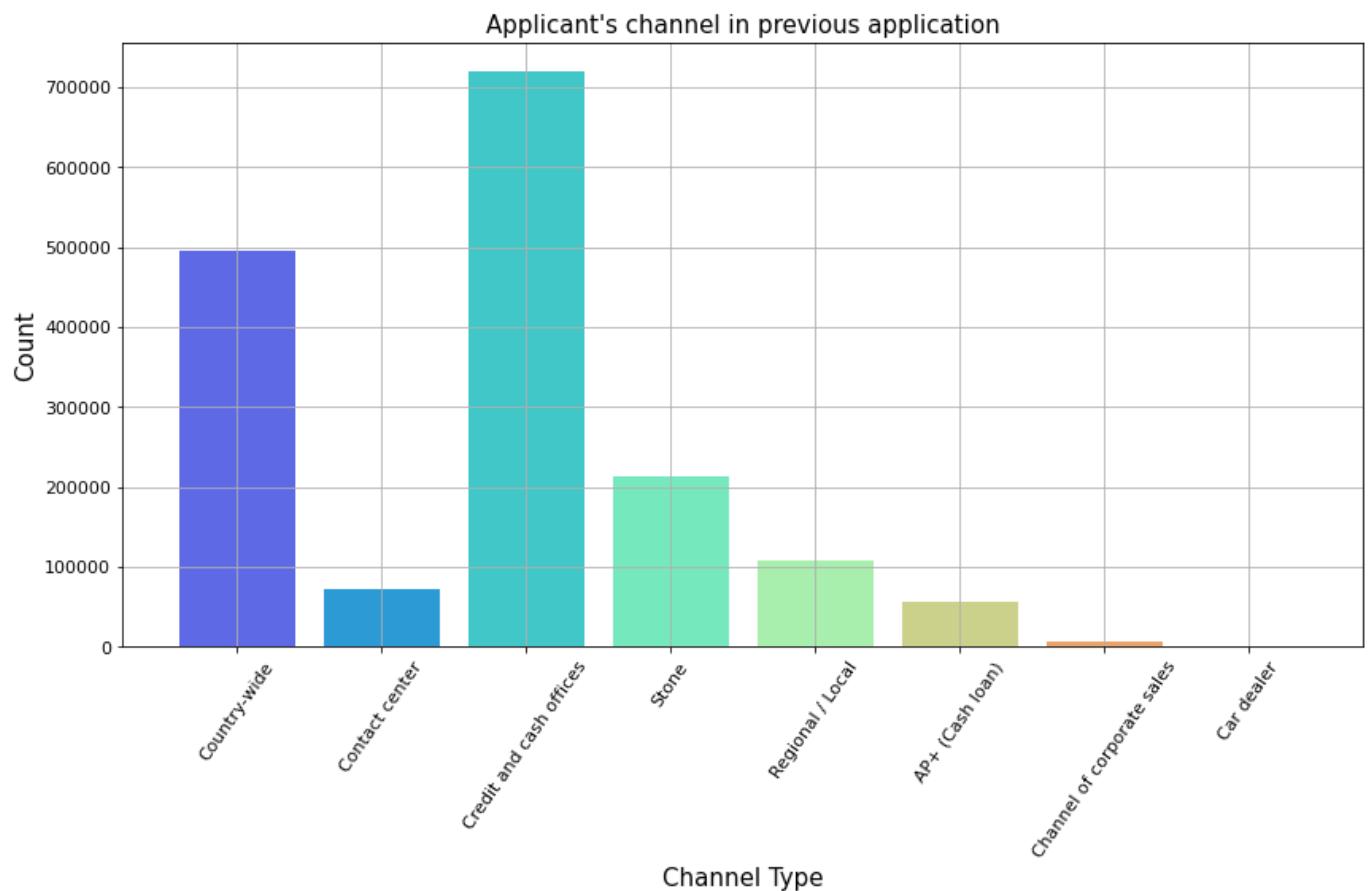
## Type of Client in Previous Application



### Observation from above

- The majority of applicants are returning candidates, followed by fresh and new candidates.

## Applicant's channel in previous application



### Observation from above

- The majority of prior applicants were referred by credit and cash offices, and vehicle dealers were the least common.

## Modeling Pipelines

- We have designed two distinct pipelines, one for numerical features and the other for categorical features. We can conduct scaling on numerical features and fill in missing values by building different pipelines.
- Both of these pipelines use an imputer to fill in the missing data, with the numerical pipeline using the median and the categorical pipeline using the most frequent value.
- To enhance the efficiency of our models, the categorical pipeline uses One Hot Encoding to convert categorical variables into integers.
- The numerical pipeline uses the Standard Scaler as a scaling tool.

```

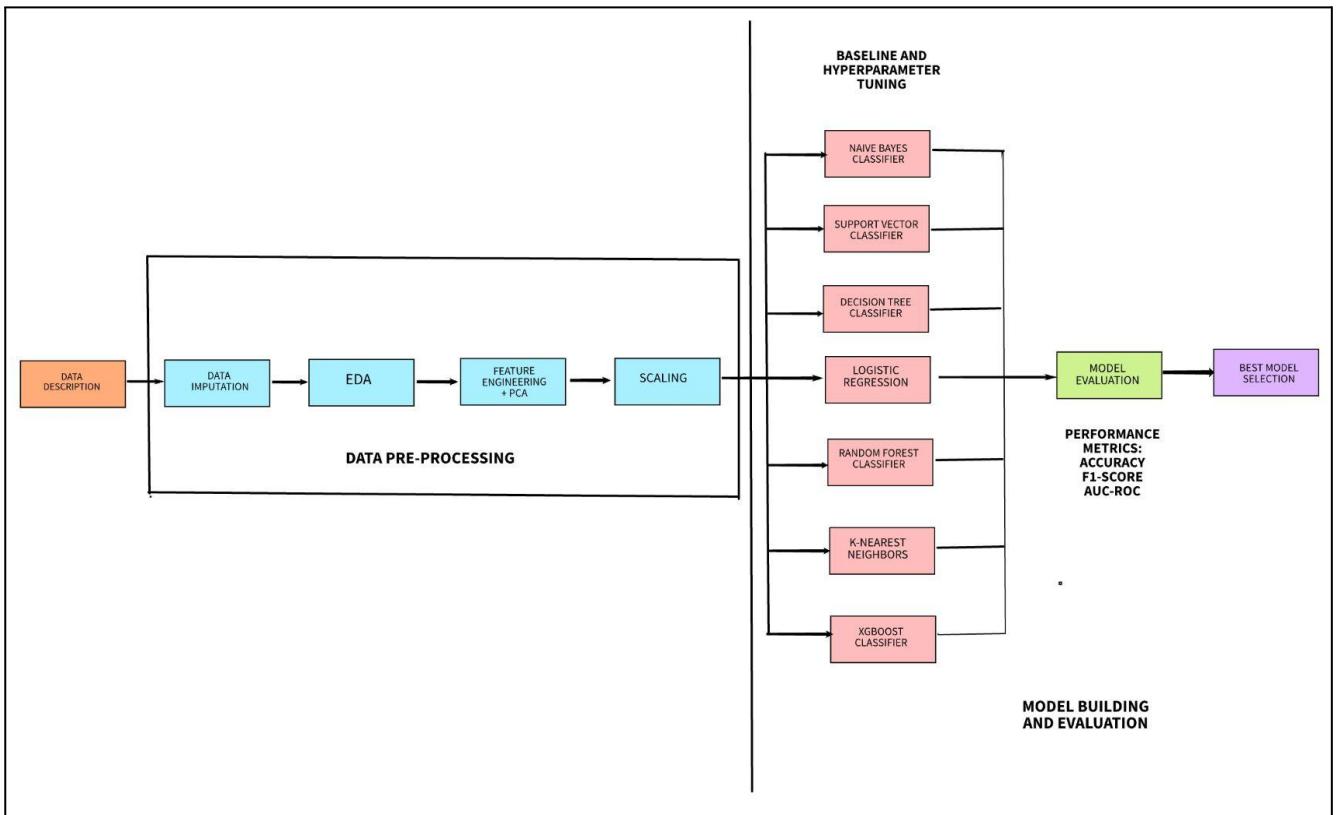
num_pipeline = Pipeline([
    ('selector', DataFrameSelector(num_attributes)),
    ('imputer', SimpleImputer(strategy='median')),
    ('std_scaler', StandardScaler()),
])
# Identify the categorical features we wish to consider.
cat_attributes = ['FLAG_OWN_REALTY', 'FLAG_OWN_CAR', 'NAME_CONTRACT_TYPE', 'FLAG_OWN_CAR']

# Notice handle_unknown="ignore" in OHE which ignore values from the validation/test that
# do NOT occur in the training set
cat_pipeline = Pipeline([
    ('selector', DataFrameSelector(cat_attributes)),
    ('imputer', SimpleImputer(strategy='most_frequent')),
    # ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
])

data_prep_pipeline = FeatureUnion(transformer_list=[
    ("num_pipeline", num_pipeline),
    ("cat_pipeline", cat_pipeline),
])
selected_features = numerical_attributes + categorical_attributes
tot_features = f'{len(selected_features)}: Num:{len(numerical_attributes)}, Cat:{len(categorical_attributes)}'
tot_features

```

'67: Num:63, Cat:4'



Following that, the two pipelines are integrated into a single "data prep pipeline" that is utilized for modeling and experiments.

## Baseline Model

To get a baseline, we will use some of the features after being preprocessed through the pipeline. The baseline model is a logistic regression model

$$\underset{\theta}{\operatorname{argmin}} [\text{CCE}] = \underset{\theta}{\operatorname{argmin}} \left[ -\frac{1}{m} \sum_{i=1}^m \left[ y^{(i)} \log(\hat{p}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{p}^{(i)}) \right] \right]$$

```
print('The Accuracy Score for Train Dataset:', np.round(accuracy_score(y_train, model.predict(X_train)), 3))
print('The F1 Score for Train Dataset:', np.round(f1_score(y_train, model.predict(X_train)), 3))
con_train = confusion_matrix(y_train, model.predict(X_train))
con_val = confusion_matrix(y_valid, model.predict(X_valid))
con_test = confusion_matrix(y_test, model.predict(X_test))
plt.figure(figsize=(7,6))
print('The Confusion Matrix for Training Set')
sns.heatmap(con_train, annot=True, fmt='g', cmap='viridis')
plt.show()
plt.figure(figsize=(7,6))
print('The Confusion Matrix for Validation Set')
sns.heatmap(con_val, annot=True, fmt='g', cmap='viridis')
plt.show()
plt.figure(figsize=(7,6))
print('The Confusion Matrix for Test Set')
sns.heatmap(con_test, annot=True, fmt='g', cmap='viridis')
plt.show()
plt.figure(figsize=(9,6))
print('The AUC-ROC for Train Set')
plot_roc_curve(model, X_train, y_train);
plt.show()
plt.figure(figsize=(9,6))
print('The AUC-ROC for Valid Set')
plot_roc_curve(model, X_valid, y_valid);
plt.show()
plt.figure(figsize=(10,8))
print('AUC-ROC for Test Set')
plot_roc_curve(model, X_test, y_test);
plt.show()
```

The Accuracy Score for Train Dataset: 0.919  
The F1 Score for Train Dataset: 0.008

## Feature Engineering and Transformers

1.

- a) For both the test and train datasets, categorical features were subjected to one-hot encoding (OHE). In addition to making our models' testing process simpler, this will help to increase their accuracy.

```
[150]: 1 # one-hot encoding of features
2 ap_train_data = pd.get_dummies(ap_train_data)
3 ap_test_data = pd.get_dummies(ap_test_data)
4
5 print('Training Features shape: ', ap_train_data.shape)
6 print('Testing Features shape: ', ap_test_data.shape)

Training Features shape: (307511, 243)
Testing Features shape: (48744, 239)
```

- b) Our pipeline for processing categorical characteristics also makes advantage of OHE.

```
[76] # do NOT occur in the training set
77 cat_pipeline = Pipeline([
78     ('selector', DataFrameSelector(cat_atrrib)),
79     ('imputer', SimpleImputer(strategy='most_frequent')),
80     # ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
81     ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
82 ])
83
```

2. Using the PolynomialFeatures class, we performed some feature engineering to obtain 3-degree features on the test and train datasets. This will enhance the functionality of our models and be more beneficial while modifying the hyperparameters.

```
[158]: 1 from sklearn.preprocessing import PolynomialFeatures
2 from sklearn.impute import SimpleImputer
3
4 # Make a new dataframe for polynomial features
5 poly_features_df = ap_train_data[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAYS_BIRTH', 'TARGET']]
6 poly_features_test_df = ap_test_data[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAYS_BIRTH']]
7
8 # imputer for handling missing values
9 # from sklearn.preprocessing import Imputer
10 imputer = SimpleImputer(strategy = 'median')
11
12 pn_target = poly_features_df['TARGET']
13
14 poly_features_df = poly_features_df.drop(columns = ['TARGET'])
15
16 # Need to impute missing values
17 poly_features_df = imputer.fit_transform(poly_features_df)
18 poly_features_test_df = imputer.transform(poly_features_test_df)
19
20 # Create the polynomial object with specified degree
21 pn_transformer = PolynomialFeatures(degree = 3)
```

```
[159]: 1 # Train the polynomial features
2 pn_transformer.fit(poly_features_df)
3
4 # Transform the features
5 poly_features_df = pn_transformer.transform(poly_features_df)
6 poly_features_test_df = pn_transformer.transform(poly_features_test_df)
7 print('Shape of polynomial features: ', poly_features_df.shape)
```

Shape of polynomial features: (307511, 35)

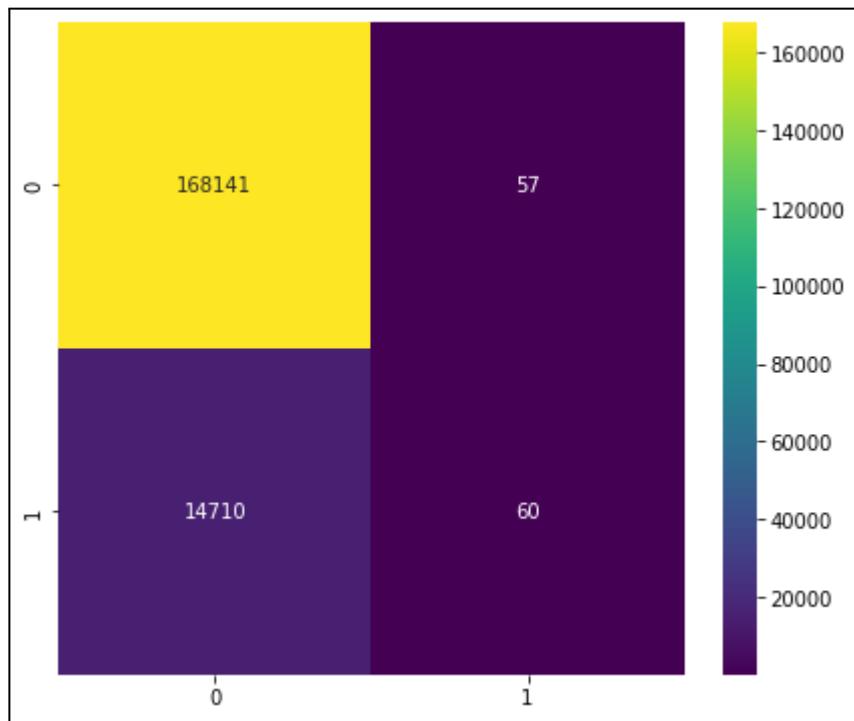
### 3. Categorical features from the test and train datasets have also undergone label encoding.

```
[149]: 1 # Label Encoding
2 # Create a Label encoder object
3 le = LabelEncoder()
4 le_count = 0
5
6 # Iterating through the columns
7 for col in ap_train_data:
8     if ap_train_data[col].dtype == 'object':
9         # If two or fewer unique categories
10        if len(list(ap_train_data[col].unique())) <= 2:
11            # Train on the training data
12            le.fit(ap_train_data[col])
13            # Transforming both training and testing data
14            ap_train_data[col] = le.transform(ap_train_data[col])
15            ap_test_data[col] = le.transform(ap_test_data[col])
16
17            # Tracking of how many columns were label encoded
18            le_count += 1
19
20 print('%d columns were label encoded.' % le_count)
```

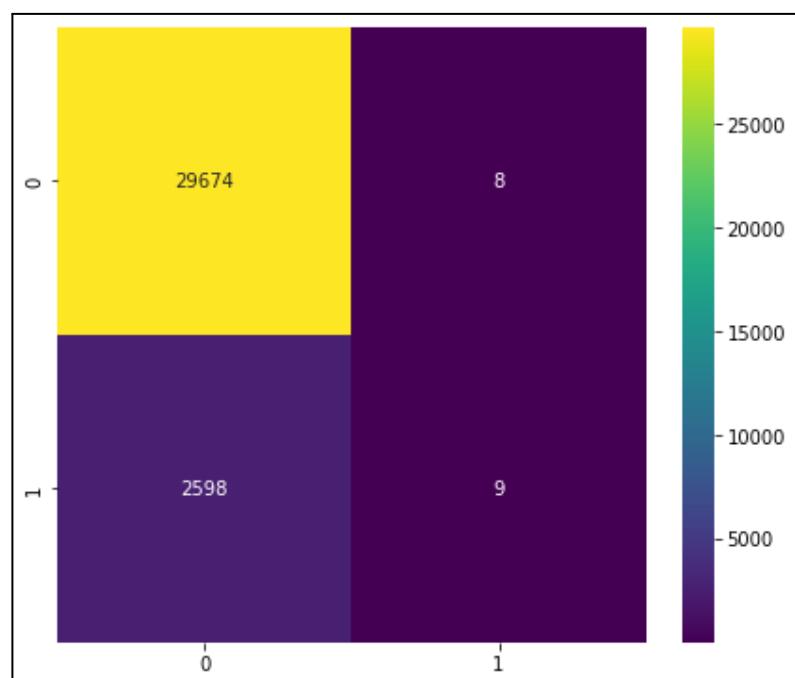
3 columns were label encoded.

## Baseline Logistic Regression

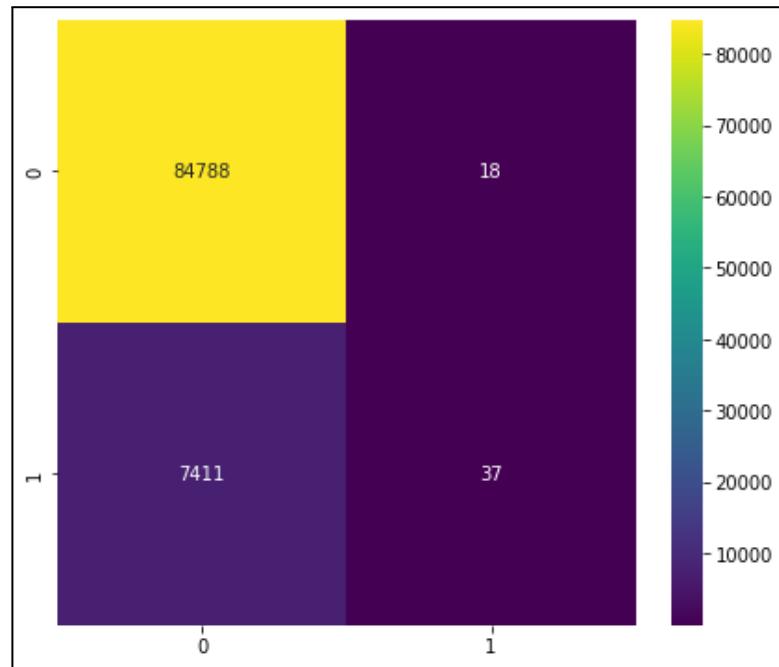
### Training Set Confusion Matrix



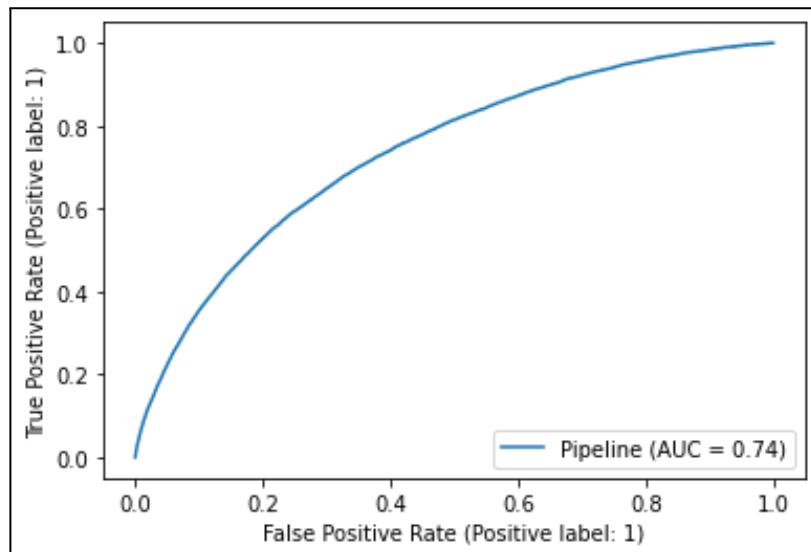
### Validation Set Confusion matrix



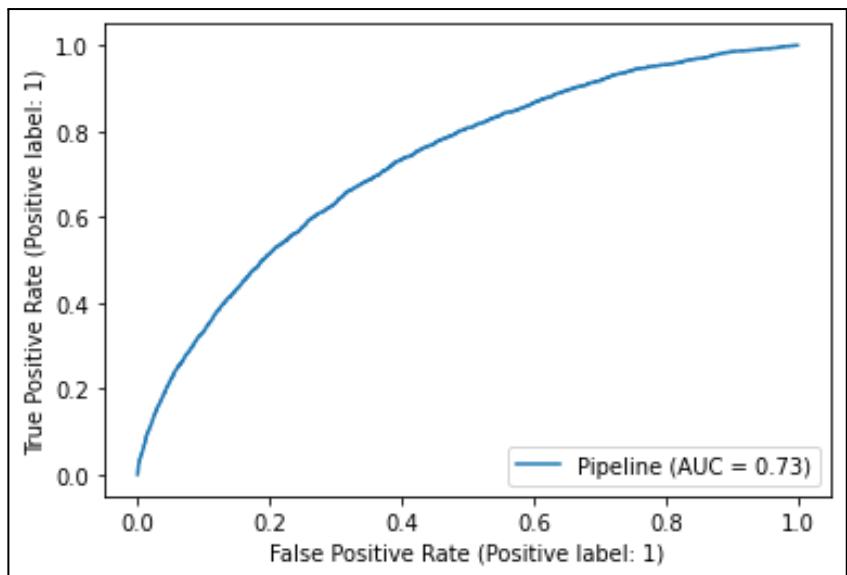
## Test Set Confusion Matrix



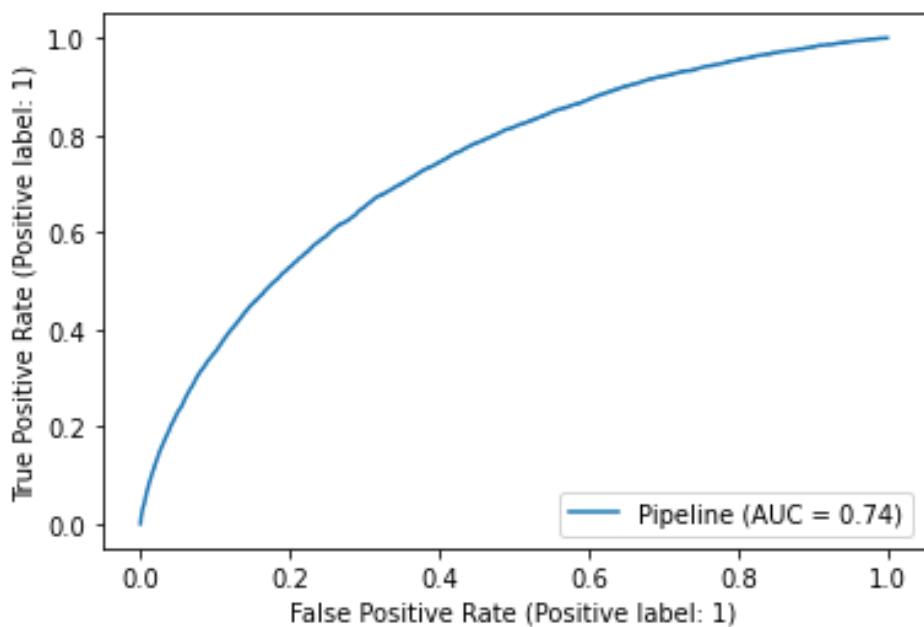
## AUC-ROC for Train Set



## AUC-ROC for Valid Set

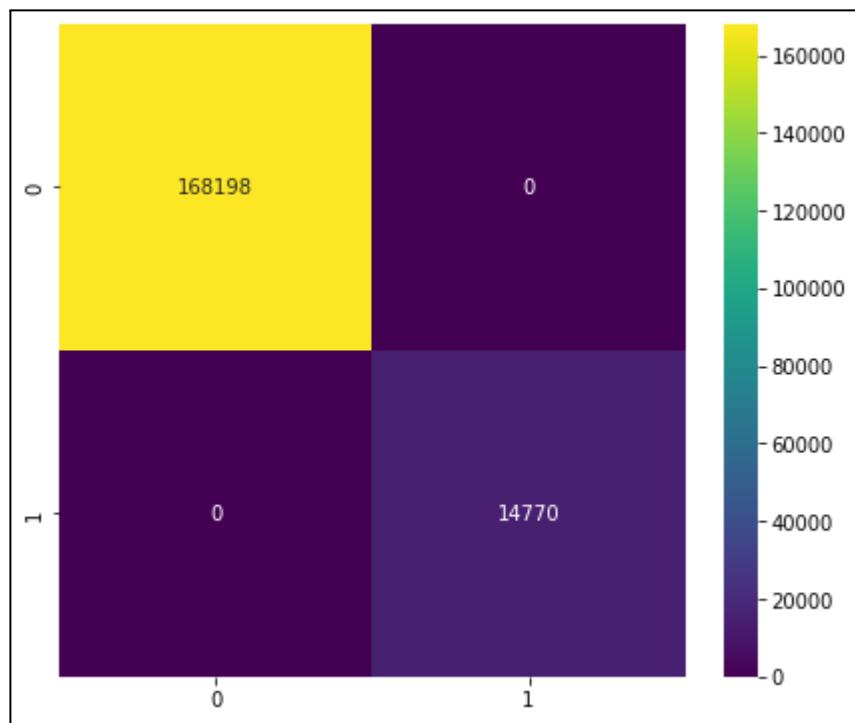


## AUC-ROC for Test Set

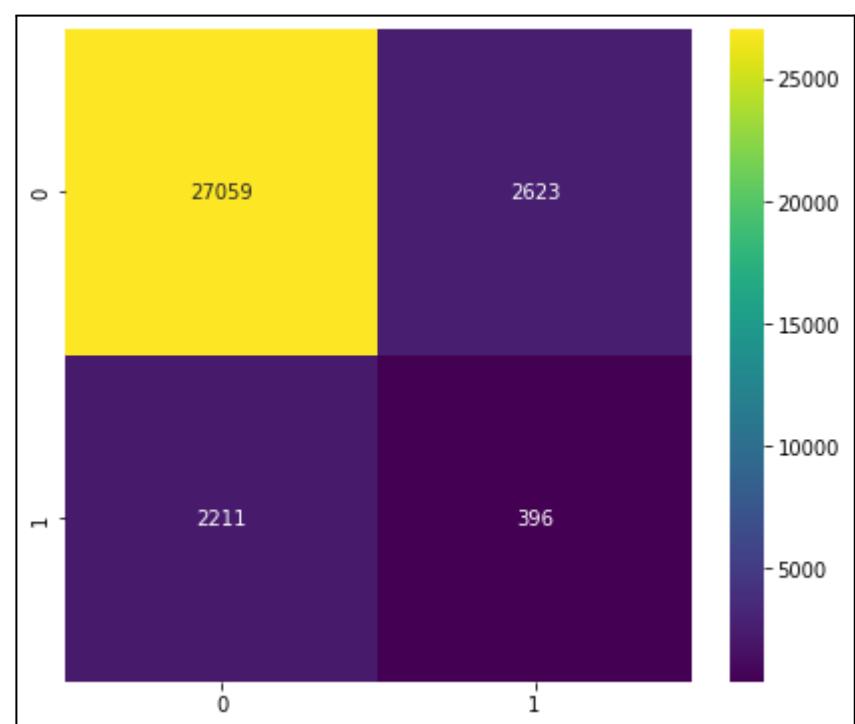


## Baseline DecisionTreeClassifier

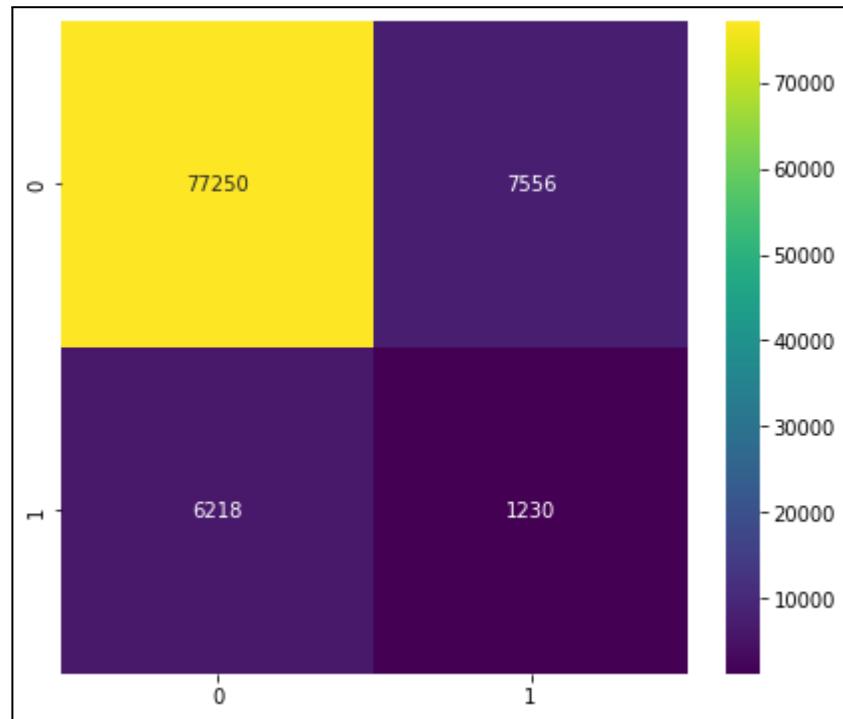
### Training Set Confusion Matrix



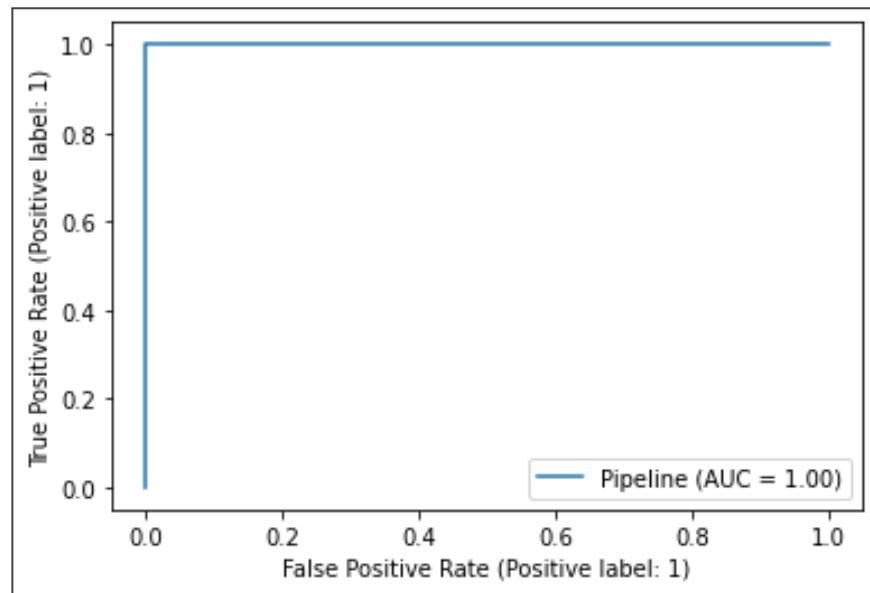
### Validation Set Confusion matrix



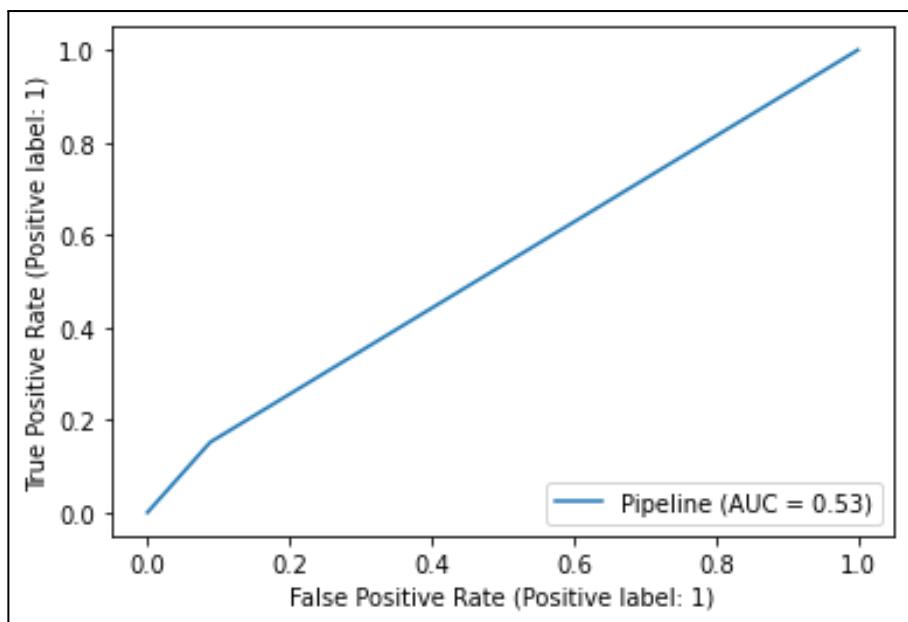
## Test Set Confusion Matrix



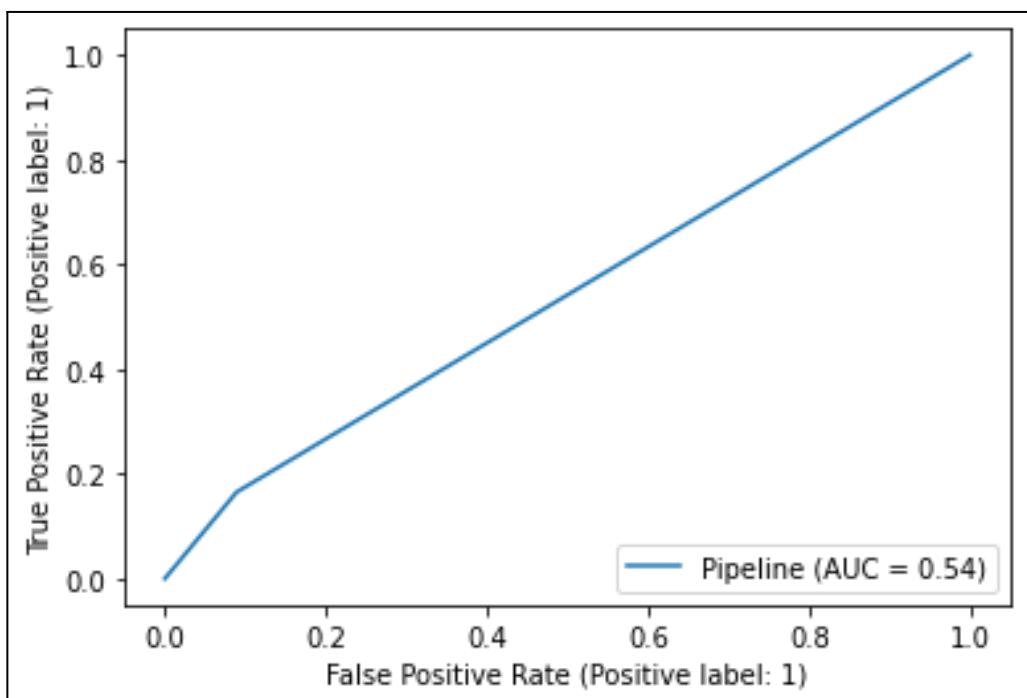
## AUC-ROC for Train Set



## AUC-ROC for Valid Set

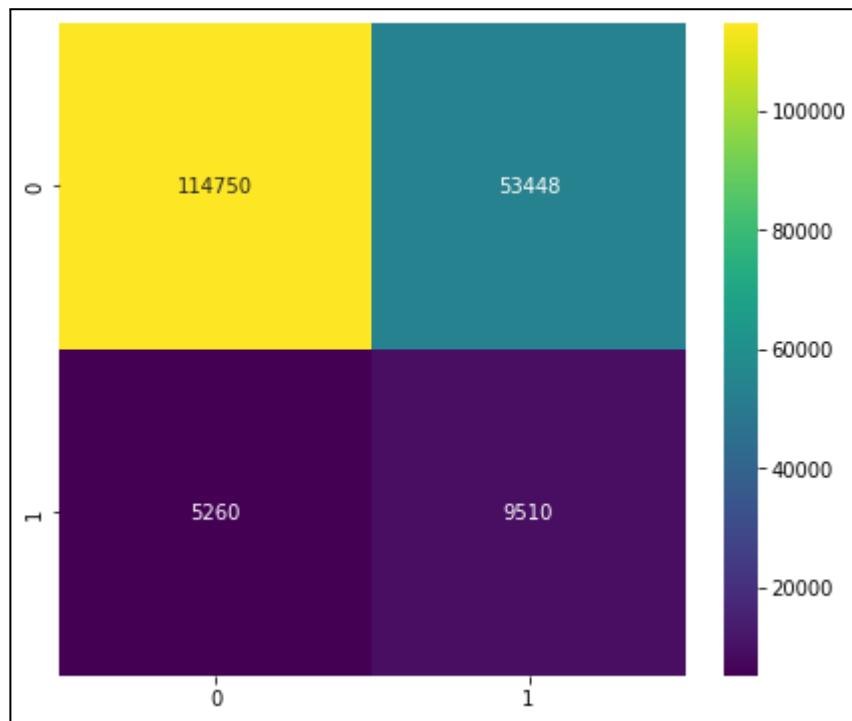


## AUC-ROC for Test Set

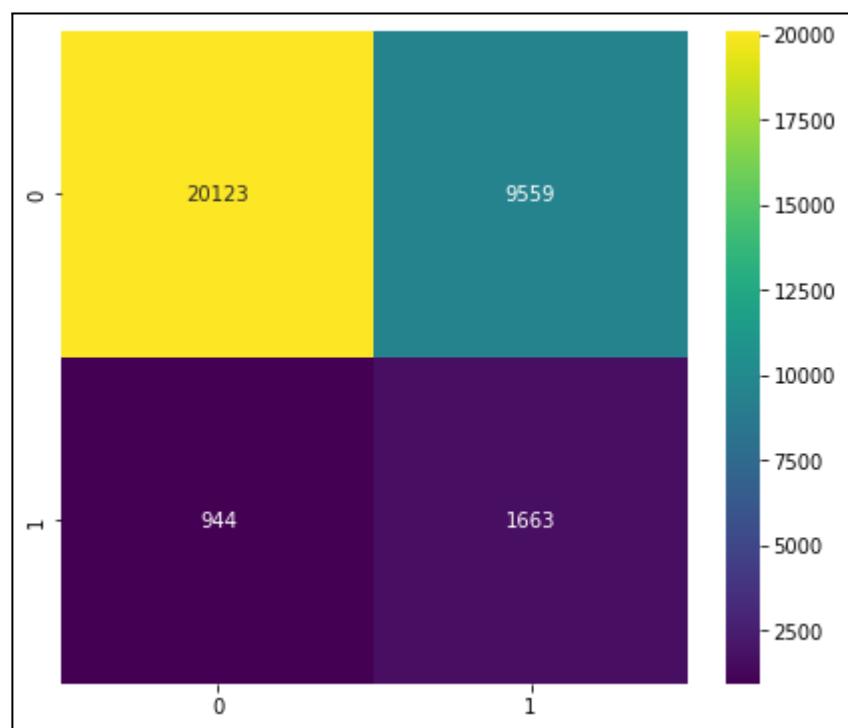


## Baseline GaussianNB

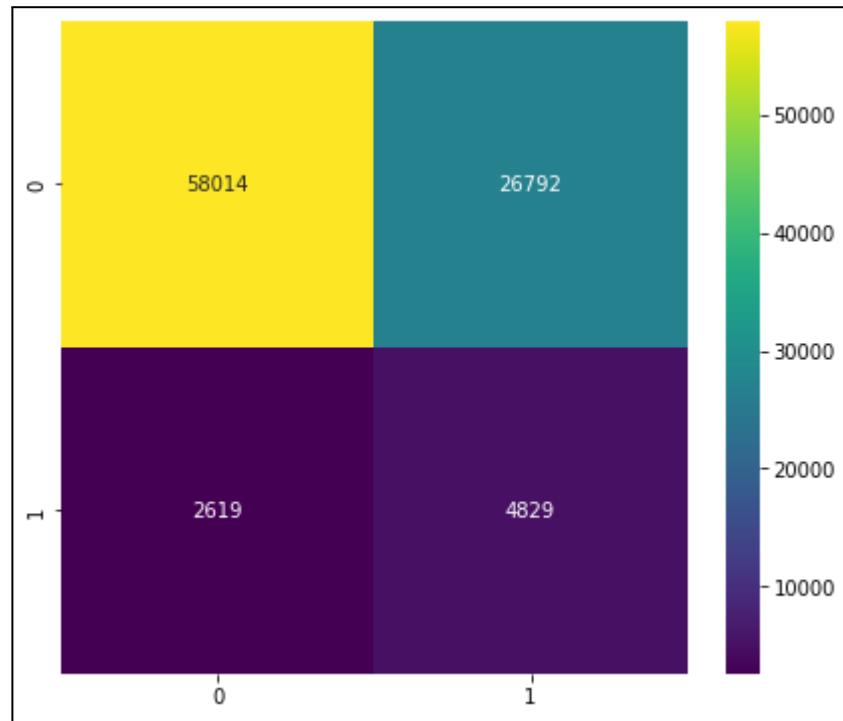
### Training Set Confusion Matrix



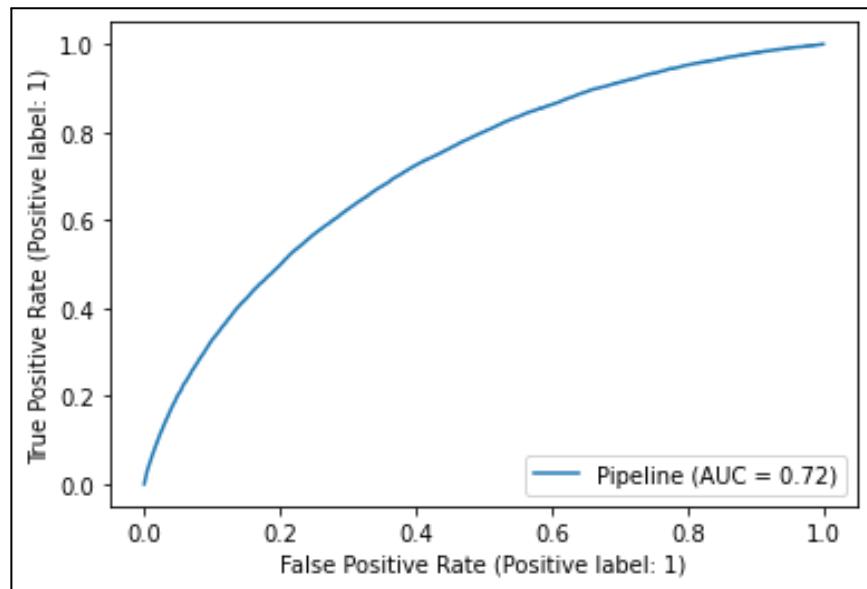
### Validation Set Confusion matrix



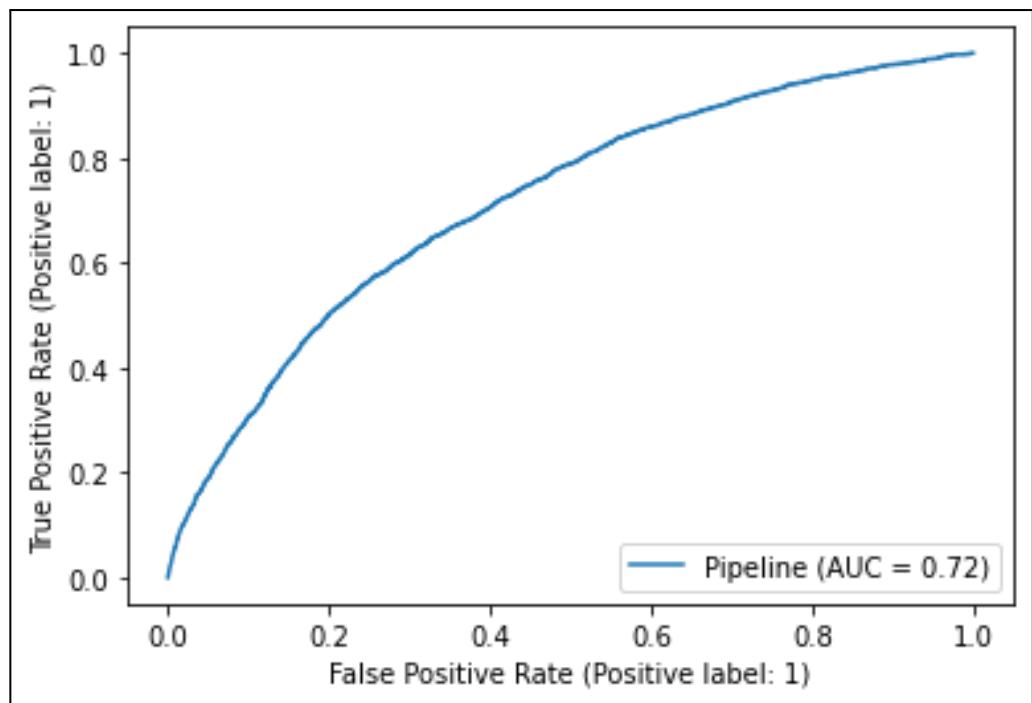
## Test Set Confusion Matrix



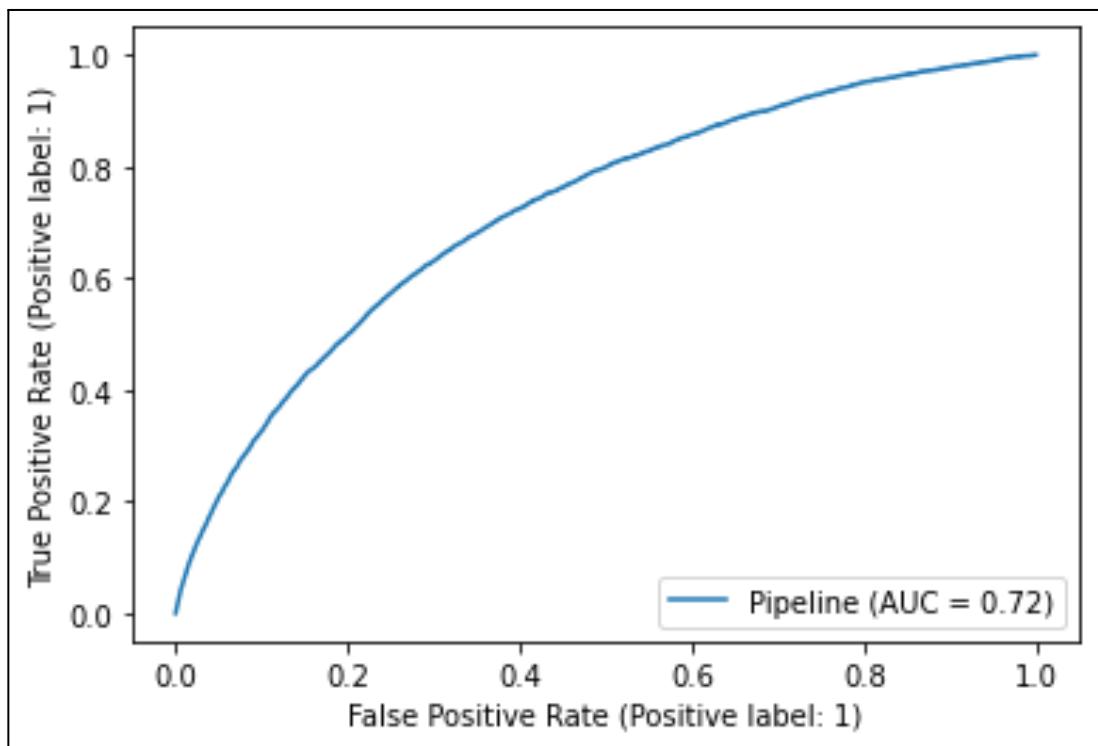
## AUC-ROC for Train Set



## AUC-ROC for Valid Set

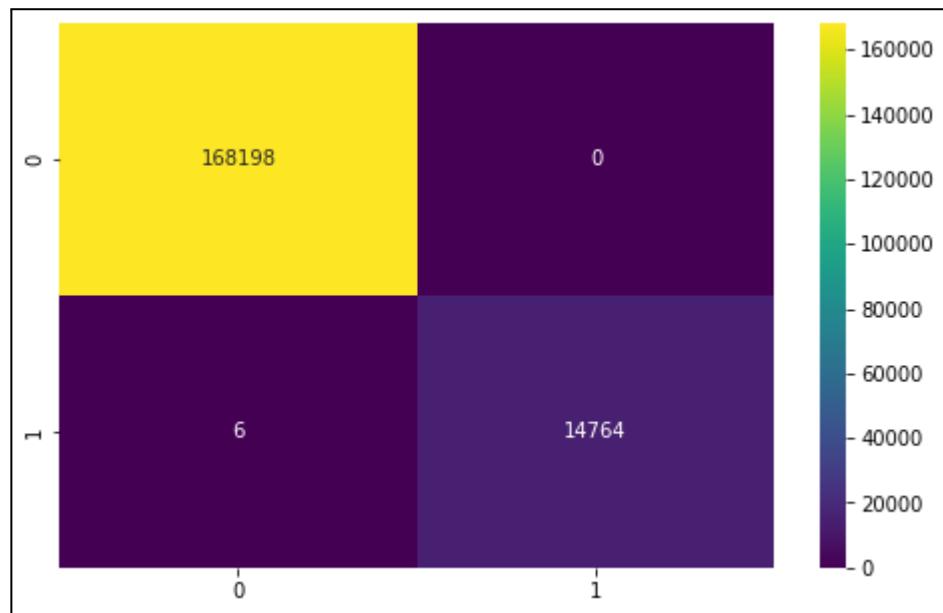


**AUC-ROC for Test Set**

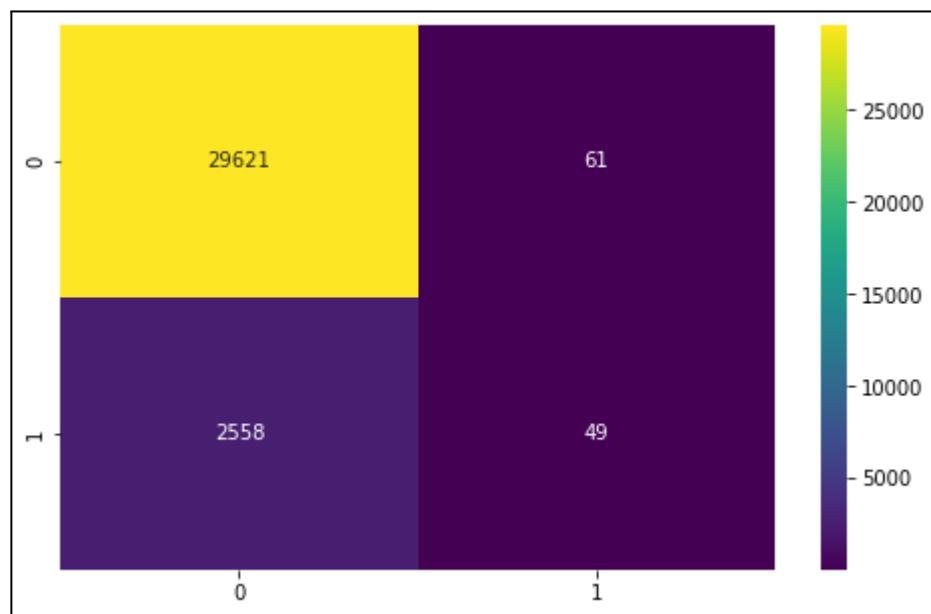


## Baseline RandomForestClassifier

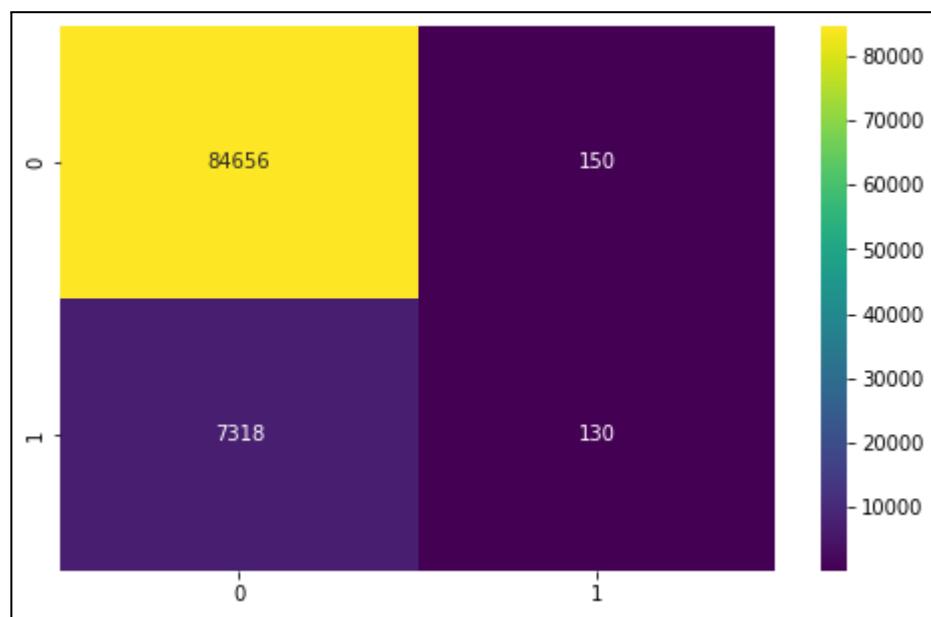
### Training Set Confusion Matrix



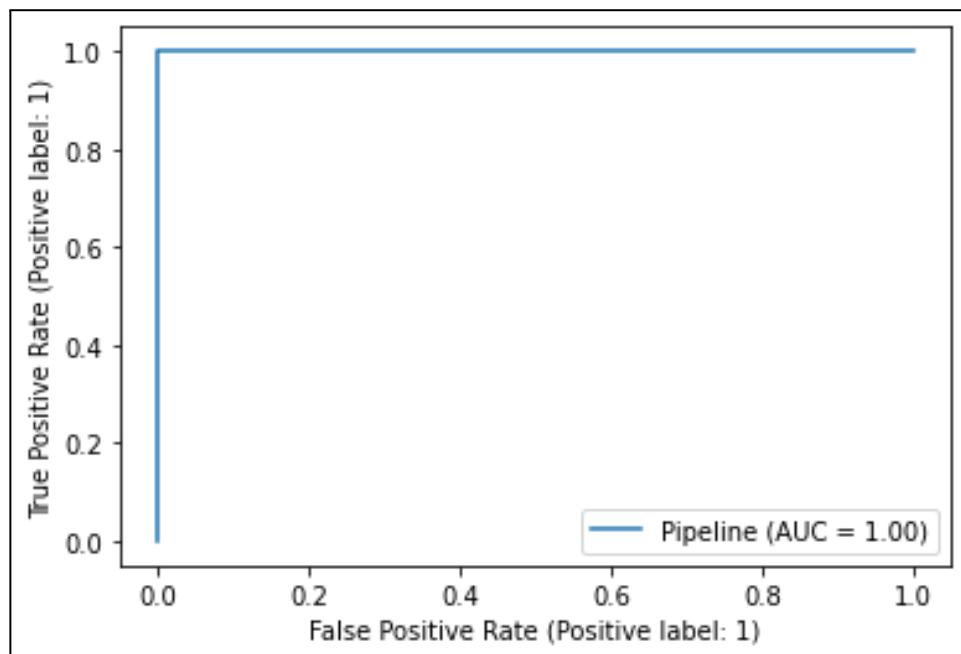
### Validation Set Confusion matrix



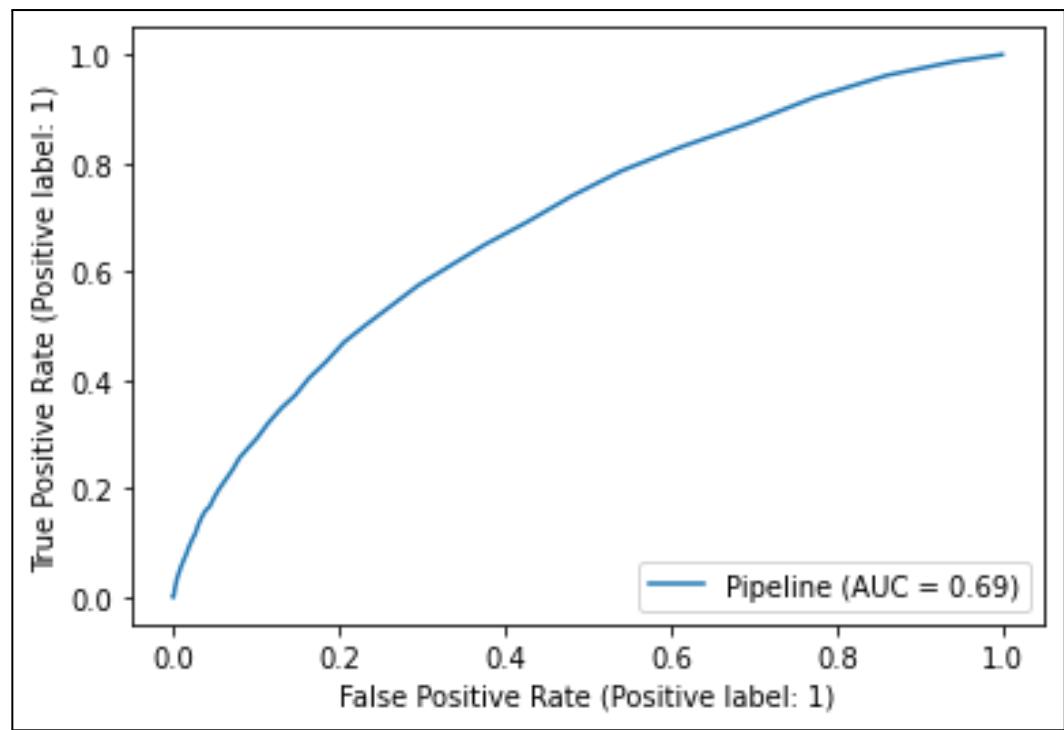
### Test Set Confusion Matrix



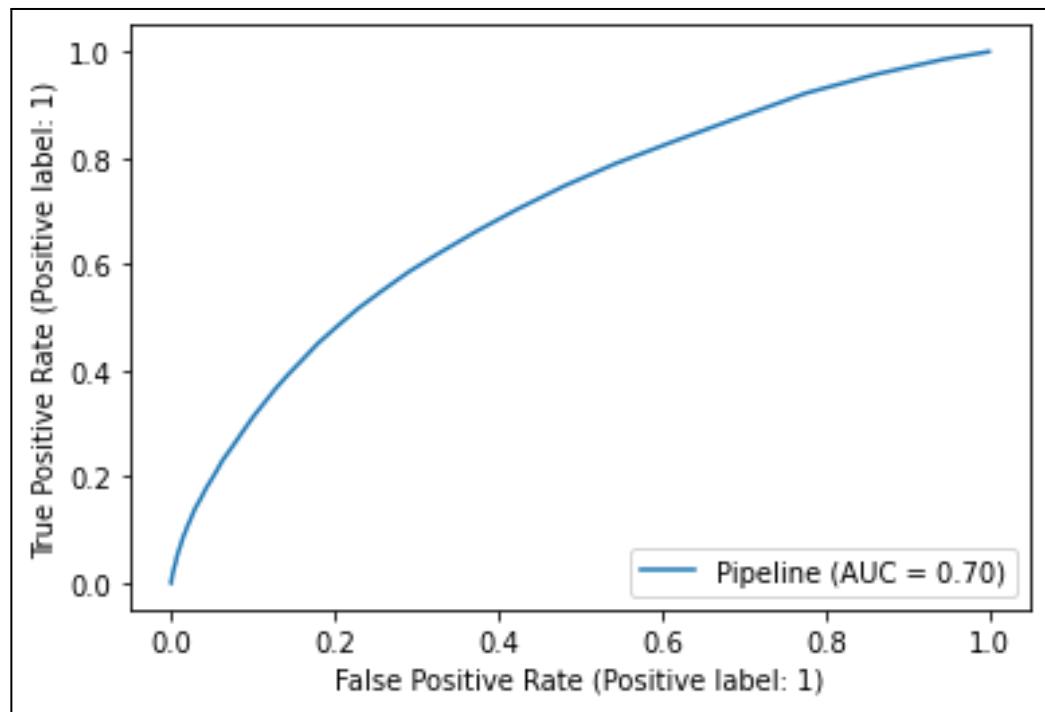
**AUC-ROC for Train Set**



**AUC-ROC for Valid Set**

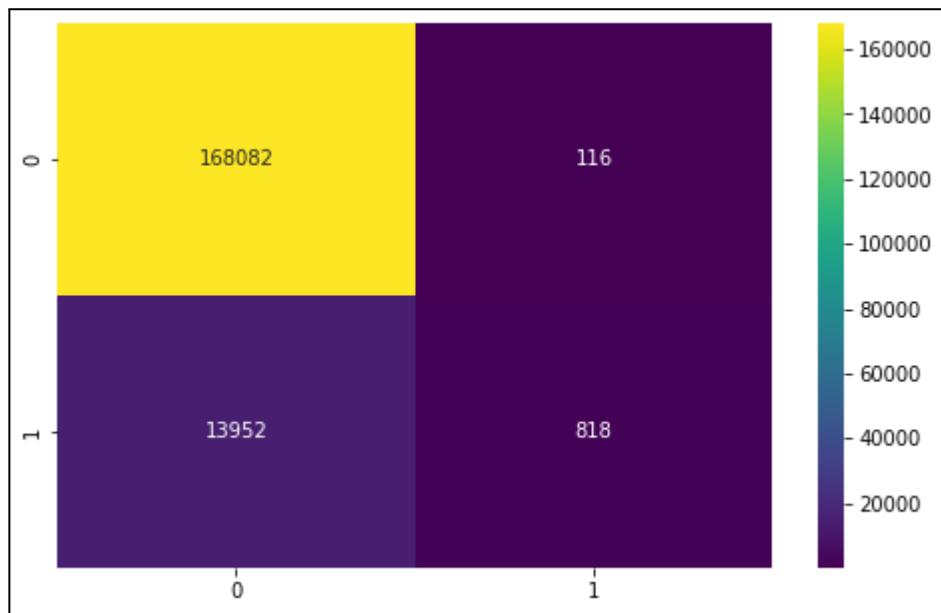


## AUC-ROC for Test Set

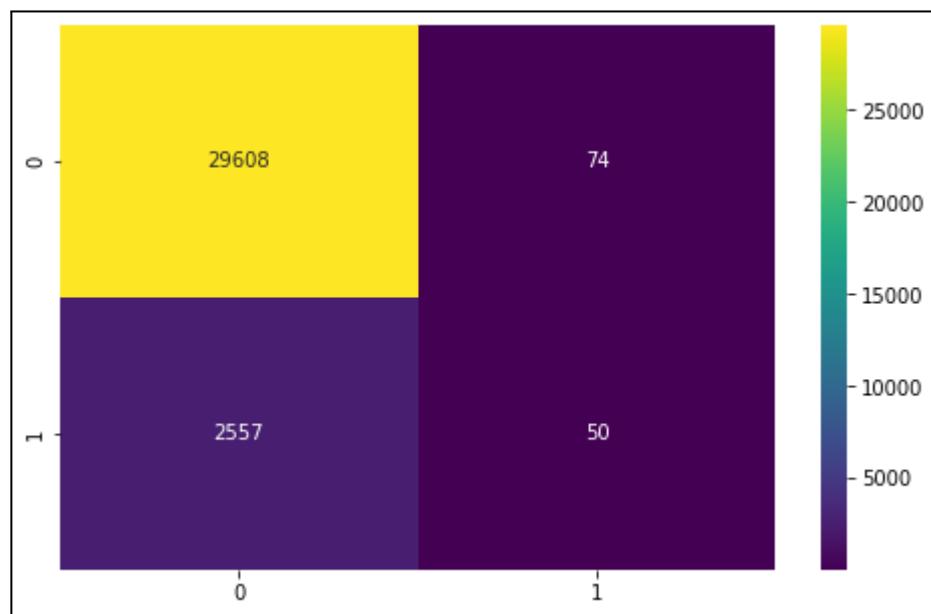


## Baseline XGBClassifier

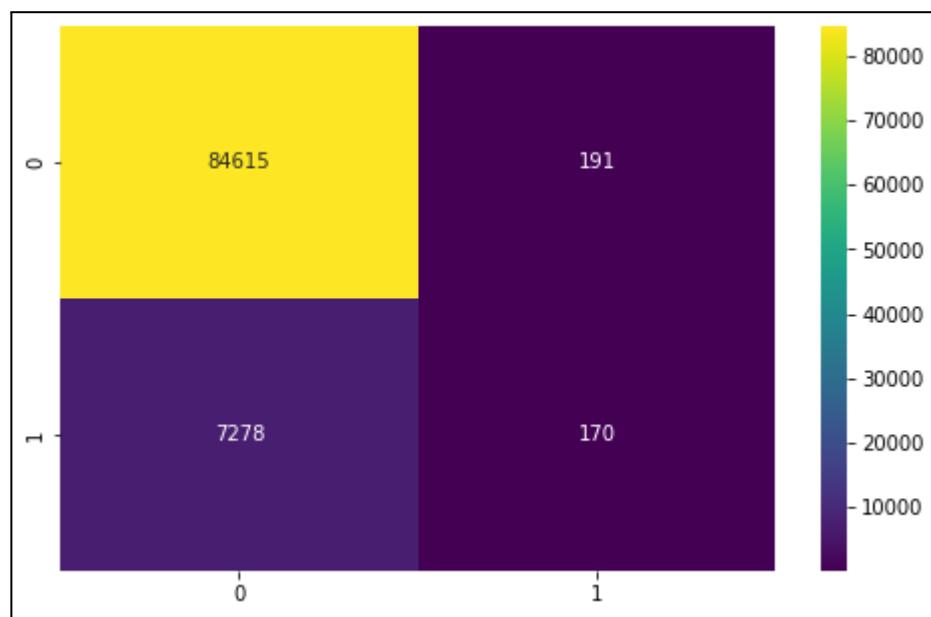
### Training Set Confusion Matrix



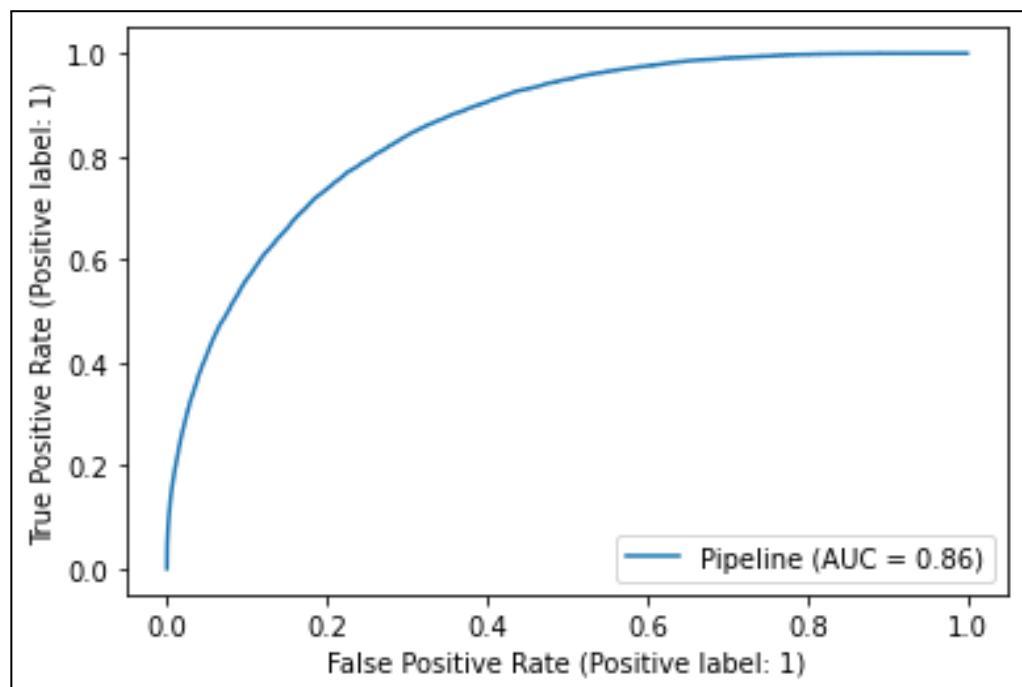
### Validation Set Confusion matrix



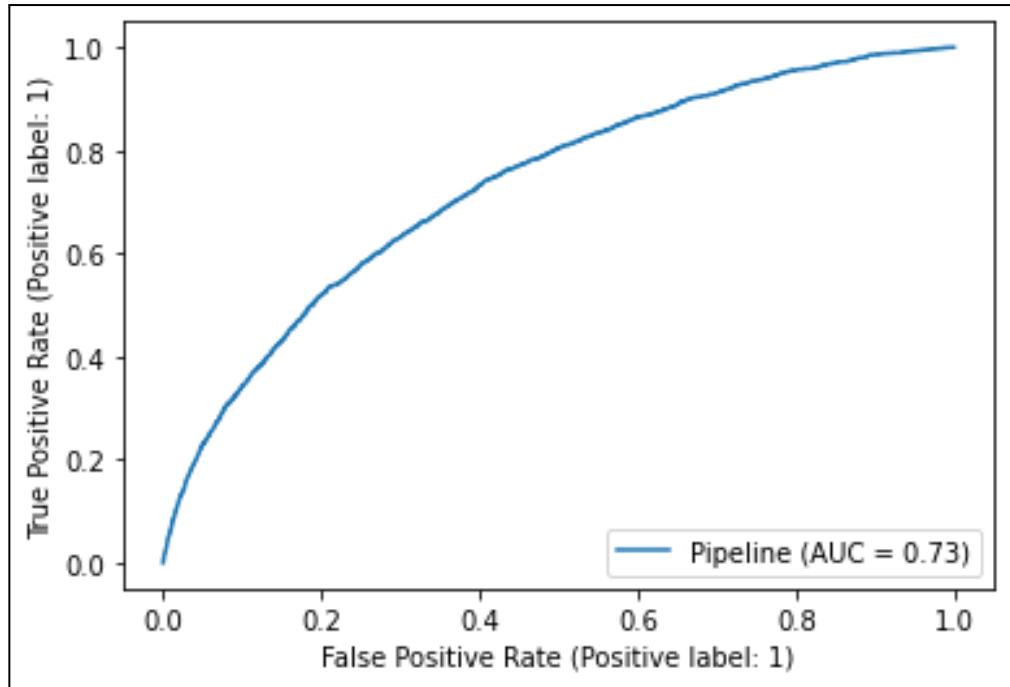
### Test Set Confusion Matrix



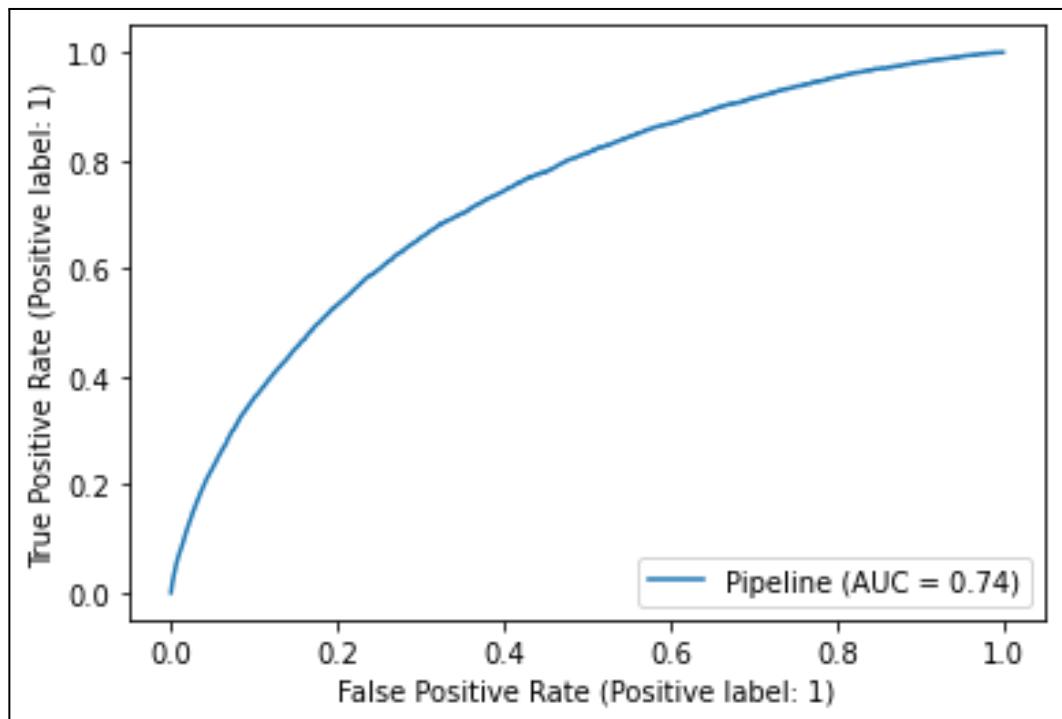
**AUC-ROC for Train Set**



**AUC-ROC for Valid Set**



**AUC-ROC for Test Set**



## Results and discussion of results

In [252...]	expLog											
Out [252]:	exp_name	Model_name	Train Acc	Valid Acc	Test Acc	Train F1	Valid F1	Test F1	Train AUC	Valid AUC	Test AUC	Fit Time
0	Baseline_67_features	Baseline XGBClassifier	0.9193	0.9193	0.9195	0.0081	0.0069	0.0099	0.7377	0.7300	0.7385	49.3522
1	Baseline_67_features	Baseline XGBClassifier	0.9193	0.9193	0.9195	0.0081	0.0069	0.0099	0.7377	0.7300	0.7385	49.3522
2	Baseline_67_features	Baseline LogisticRegression	0.9193	0.9194	0.9195	0.0083	0.0076	0.0093	0.7378	0.7299	0.7386	5.8745
3	Baseline_67_features	Baseline LogisticRegression	0.9193	0.9193	0.9195	0.0081	0.0069	0.0099	0.7377	0.7300	0.7385	6.3473
4	Baseline_67_features	Baseline DecisionTreeClassifier	1.0000	0.8503	0.8507	1.0000	0.1408	0.1515	1.0000	0.5318	0.5380	22.4071
5	Baseline_67_features	Baseline GaussianNB	0.6791	0.6747	0.6812	0.2447	0.2405	0.2472	0.7229	0.7156	0.7225	1.8760
6	Baseline_67_features	Baseline RandomForestClassifier	1.0000	0.9189	0.9190	0.9998	0.0361	0.0336	1.0000	0.6903	0.6980	136.4937
7	Baseline_67_features	Baseline XGBClassifier	0.9231	0.9185	0.9190	0.1042	0.0366	0.0435	0.8586	0.7290	0.7381	51.3018

Our results showed that Logistic Regression and XGBoost had the highest training accuracy, achieving an accuracy of 91%. The test accuracy for Decision Tree Classifier and GaussianNB were found to be 53.8% and 72.25% respectively. After submitting our predictions to Kaggle, we obtained AUC private and public scores of 71.54% and 71.49% respectively.

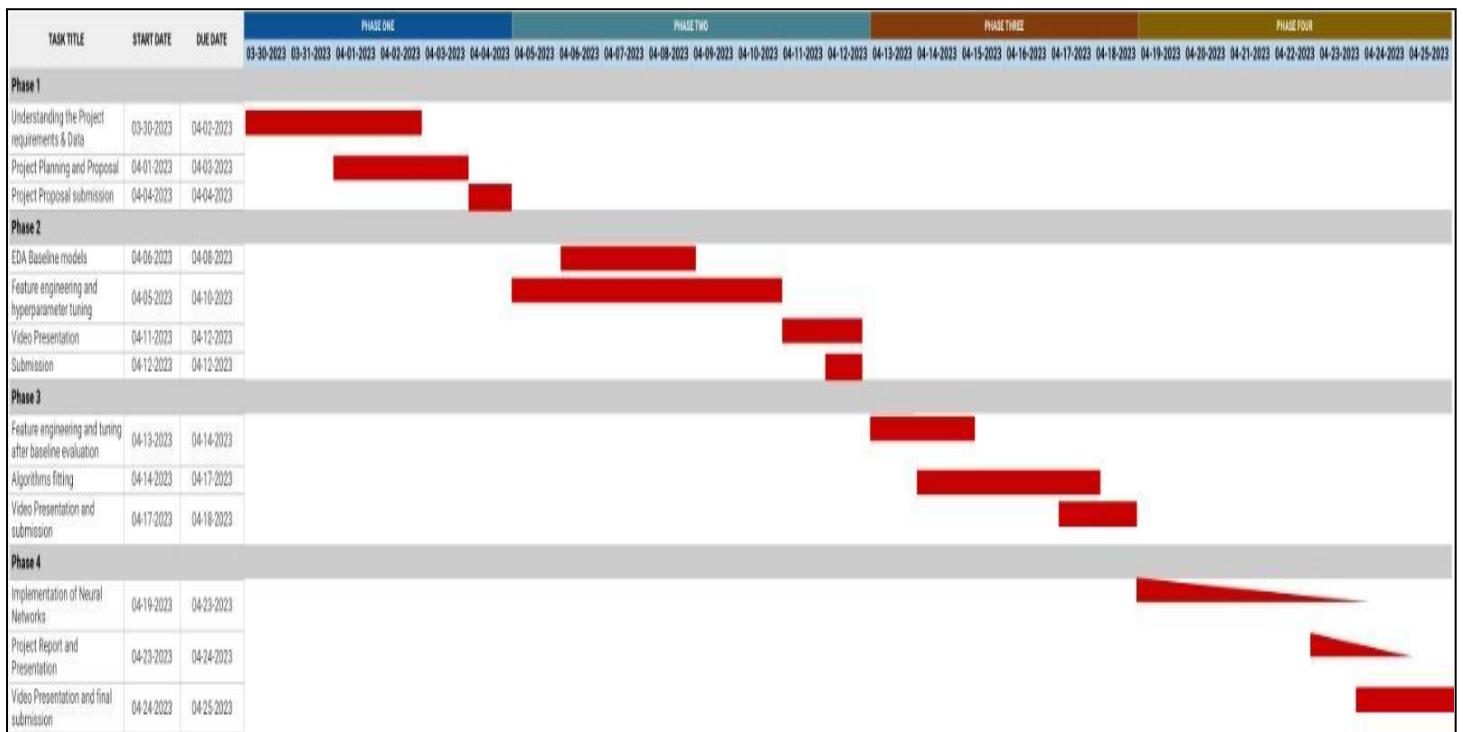
Further tests revealed that XGBoost and Logistic Regression also had the highest accuracy in the test set, both achieving an accuracy of 73.85%. These results suggest that Logistic Regression and XGBoost are promising models for predicting borrower's repayment capacity using the "Home Credit Default Risk" dataset.

And the Other models, Random ForestClassifier and The Decision Tree Classifier had low accuracy of 53.8% and 69.8% which is comparatively low, and may be improved if selected some other columns.

Overall, our experiment demonstrated the effectiveness of machine learning models in predicting borrower's repayment capacity and highlighted the potential of Logistic Regression and XGBoost as suitable models for this task. However, further research and fine-tuning of the models may be necessary to improve their performance and applicability in real-world scenarios.

## Timeline

The proposed timeline for the Home Credit Default Risk Project:



## Phase Leader Plan

Final Project Phase	Phase Leader	Phase Plan
Phase 0	Team	Team Formation
Phase 1: Project Proposal	Kumud Sharma	There will be a project proposal created outlining all of the project's components. Each team member is given a task to do. The dataset is investigated, and potential machine learning techniques that could be used in the project are described. A base pipeline for the project is chosen, and appropriate evaluation metrics are determined.

Phase 2: EDA + Baseline	Bhavya Mistry	With the chosen machine learning methods, this phase focuses on baseline model training and evaluation as well as exploratory data analysis, data imputation, and baseline model evaluation. The EDA and baseline performances of several models will be used to derive conclusions, and then judgments will be taken regarding hyperparameter tuning and feature selection.
Phase 3: Feature Engineering + Hyperparameter Tuning	Kamna Chaudhary	Here, we'll concentrate on the feature of choosing the most suitable features based on several strategies, including correlation, developing new features, and dimensionality reduction, as well as our comprehension of the data from earlier phases. To help determine the ideal set of parameters for each machine learning model, we will also begin experimenting with the different parameters of the models. This will pave the stage for an exciting race between the various algorithms.
Phase 4: Final Submission	Jaydeep Patel	Deep neural networks will be used in this step, and we will compare their performance to the existing fine-tuned models from our previous phase. We will choose the best model and submit it in our final submission based on our numerous performance indicators.

### Credit Assignment Plan:

Phase	Member Name	Task Name	Description
Phase 1	Kumud Sharma Bhavya Mistry Kamna Chaudhary Jaydeep Patel	Understanding the Data and problemset	Understanding the Dataset given and problemset provided.
	Kumud Sharma	Assigning tasks and phase planning	Setting the timeline for the project and discussing with teammates the tasks to be done.
	Jaydeep Patel	Abstract, Data Description	Writing a detailed abstract and data description including all

			the tables and columns.
	Kamna Chaudhary	ML pipeline	Describing the flow of the project with the ML models we will be using.
	Bhavya Mistry	ML algorithms and metrices	Defing the different metrics to see how well the model is performing.
Phase 2	Kumud Sharma	EDA and data visualization	Performed data visualisation and analysed datasets.
	Jaydeep Patel	Feature engineering and model evaluation	Transformed data to remove or add irrelevant or relevant and performed model evaluation to see how well model is performing.
	Kamna Chaudhary	Creating pipeline and describing hyperparameters	Created a pipeline for different features like numeric and categorical and collected hyperparameterd for model.
	Bhavya Mistry	Training model, discussing results	Trained the model using different ML algorithms with the pipeline we created. Compared the accuracy and result of different baseline models we created.
Phase 3	Jaydeep Patel	Feature Engineering/ NN	Will Provide additional features to be added to training data and implementing neural networks.
	Kamna Chaudhary	Hyperparameter tuning/NN	Tuning model to find optimal parameters.
	Bhavya Mistry	Modeling pipelines	Visualization of modeling pipelines
	Kumud Sharma	Model evaluation and discussing results	Analysing and comparing the results
Phase 4	Kumud Sharma	Fina model evualtion	Analysing and comparing the results achieved on our final model

	Jaydeep Patel	Final project presentation	Preparing final presentation for our project
	Kamna Chaudhary	Final project report	Preparing final report for our project
	Bhavya Mistry	Hyperparameter tuning and baseline models revised	Conducting hyperparameter tuning and baseline models if needed.

## **Bibliography:**

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

<https://seaborn.pydata.org/>

<https://scikit-learn.org>

<https://pandas.pydata.org/>

<https://numpy.org/>

<https://www.kaggle.com/code/flavioblondeau/home-credit-default-risk-notebook>

## **Conclusion:**

The focus of our project is to develop a machine learning model that predicts a borrower's repayment capacity using the "Home Credit Default Risk" dataset. This is important because financial institutions need to assess whether a borrower has the ability to repay a loan. By conducting basic Exploratory Data Analysis on application, demographic, and credit behavior history data, and developing a baseline pipeline that handles missing, anomalous, and unusual data, we can create an accurate model that assesses risk and categorizes clients as Defaulters.

Conducted basic Exploratory Data Analysis on the "Home Credit Default Risk" dataset. Developed a baseline pipeline that handles missing, anomalous, and unusual data. Employed several baseline models, including Logistic Regression, Decision Tree Classifier, Random Forest, Naive Bayes Classifier, and XGBoost Classifier

When submitted to Kaggle, the baseline Ridge Logistic Regression model generated private scores of 71.54% and roughly 71.49% for public. Constructed and examined baseline models - Random Forest Logistic Regression. In the future, we plan to further refine our model by incorporating more advanced machine learning techniques and exploring additional parameters that can improve its accuracy. Overall, our project demonstrates the potential of machine learning to improve lending practices and reduce the risk of loan defaults.

# 1 Home Credit Default Risk (HCDR)

The course project is based on the [Home Credit Default Risk \(HCDR\) Kaggle Competition](https://www.kaggle.com/c/home-credit-default-risk/) (<https://www.kaggle.com/c/home-credit-default-risk/>). 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.

## 1.1 Some of the challenges

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

## 2 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 you 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](https://www.kaggle.com/) (<https://www.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](#) (<https://medium.com/@nokkk/make-your-kaggle-submissions-with-kaggle-official-api-f49093c04f8a>) and [here](#) (<https://github.com/Kaggle/kaggle-api>).

```
In [ ]: !pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.7/site-packages (1.5.12)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/site-packages (from kaggle) (1.15.0)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/site-packages (from kaggle) (1.26.6)
Requirement already satisfied: requests in /usr/local/lib/python3.7/site-packages (from kaggle) (2.25.1)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/site-packages (from kaggle) (2.8.2)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/site-packages (from kaggle) (2021.5.30)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/site-packages (from kaggle) (5.0.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/site-packages (from kaggle) (4.62.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/site-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.7/site-packages (from requests->kaggle) (4.0.0)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/site-packages (from requests->kaggle) (2.10)
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 21.2.4; however, version 21.3.1 is available.
You should consider upgrading via the '/usr/local/bin/python -m pip install --upgrade pip' command.
```

```
In [ ]: !pwd
```

```
/root/shared/AML/I526_AML_Student/Assignments/Unit-Project-Home-Credit-Default-Risk/Phase2
```

```
In [ ]: !pwd
```

```
/root/shared/AML/I526_AML_Student/Assignments/Unit-Project-Home-Credit-Default-Risk/Phase2
```

```
In [ ]: !ls -l ~/.kaggle/kaggle.json
```

```
-rw----- 1 root root 62 Nov 22 17:40 /root/.kaggle/kaggle.json
```

```
In [ ]: # !mkdir ~/.kaggle
# !cp kaggle.json ~/.kaggle
!chmod 600 ~/.kaggle/kaggle.json
```

```
In [ ]: ! kaggle competitions files home-credit-default-risk
```

name	size	creationDate
installments_payments.csv	690MB	2019-12-11 02:55:35
POS_CASH_balance.csv	375MB	2019-12-11 02:55:35
previous_application.csv	386MB	2019-12-11 02:55:35
application_train.csv	158MB	2019-12-11 02:55:35
HomeCredit_columns_description.csv	37KB	2019-12-11 02:55:35
credit_card_balance.csv	405MB	2019-12-11 02:55:35
sample_submission.csv	524KB	2019-12-11 02:55:35
bureau.csv	162MB	2019-12-11 02:55:35
bureau_balance.csv	358MB	2019-12-11 02:55:35
application_test.csv	25MB	2019-12-11 02:55:35

## 3 Dataset and how to download

### 3.1 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.

#### 3.1.1 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.

### 3.2 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 untrustworthy lenders.

Home Credit group has over 29 million customers, total assets of 21 billions Euro, over 160 millions loans, with the majority in Asia 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.

### 3.3 Data files overview

The `HomeCredit_columns_description.csv` acts as a data dictionary.

There are 7 different sources of data:

- **application\_train/application\_test (307k rows, and 48k rows):** 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 (1.7 Million rows):** 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 (27 Million rows):** 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 (1.6 Million rows):** 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 (10 Million rows):** 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 (13.6 Million rows):** payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

### 3.3.1 Table sizes

name	[ rows cols]	MegaBytes
application_train	: [ 307,511, 122]:	158MB
application_test	: [ 48,744, 121]:	25MB
bureau	: [ 1,716,428, 17]	162MB
bureau_balance	: [ 27,299,925, 3]:	358MB
credit_card_balance	: [ 3,840,312, 23]	405MB
installments_payments	: [ 13,605,401, 8]	690MB
previous_application	: [ 1,670,214, 37]	386MB
POS_CASH_balance	: [ 10,001,358, 8]	375MB



## 3.4 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](https://www.kaggle.com/c/home-credit-default-risk/data) (<https://www.kaggle.com/c/home-credit-default-risk/data>) and unzip the zip file to the BASE\_DIR
2. If you plan to use the Kaggle API, please use the following steps.

```
In [7]: DATA_DIR = "/root/shared/AML/I526_AML_Student/Assignments/Unit-Project"
#DATA_DIR = os.path.join('./ddddd/')
# !mkdir DATA_DIR
```

In [8]: `!ls -l DATA_DIR`

```
total 2621364
-rw-r--r-- 1 root root      37383 Nov 25 21:45 HomeCredit_columns_desc
ription.csv
-rw-r--r-- 1 root root  392703158 Nov 25 21:45 POS_CASH_balance.csv
-rw-r--r-- 1 root root  26567651 Nov 25 21:45 application_test.csv
-rw-r--r-- 1 root root 166133370 Nov 25 21:45 application_train.csv
-rw-r--r-- 1 root root 170016717 Nov 25 21:45 bureau.csv
-rw-r--r-- 1 root root 375592889 Nov 25 21:45 bureau_balance.csv
-rw-r--r-- 1 root root 424582605 Nov 25 21:45 credit_card_balance.csv
-rw-r--r-- 1 root root 723118349 Nov 25 21:45 installments_payments.c
sv
-rw-r--r-- 1 root root 404973293 Nov 25 21:45 previous_application.cs
v
-rw-r--r-- 1 root root     536202 Nov 25 21:45 sample_submission.csv
```

In [9]: `# ! kaggle competitions download home-credit-default-risk -p $DATA_DIR`

```
Downloading home-credit-default-risk.zip to /root/shared/AML/I526_AML
_Student/Assignments/Unit-Project-Home-Credit-Default-Risk/Phase2
13%|███████| 87.0M/688M [00:04<00:33
, 18.6MB/s]^C
13%|███████| 87.0M/688M [00:04<00:34
, 18.4MB/s]
User cancelled operation
```

In [10]: `!pwd`

```
/root/shared/AML/I526_AML_Student/Assignments/Unit-Project-Home-Credi
t-Default-Risk/Phase2
```

In [11]: `!ls -l $DATA_DIR`

```
total 831304
drwxr-xr-x 1 root root      4096 Nov 25 21:45 DATA_DIR
-rwxrwxrwx 1 root root  4988449 Nov 29 20:34 HCDR_baseLine_submissio
n_with_numerical_and_cat_features_to_kaggle_v1.ipynb
-rwxrwxrwx 1 root root  2215207 Nov 29 20:47 HCDR_baseLine_submissio
n_with_numerical_and_cat_features_to_kaggle_v2.ipynb
-rwxrwxrwx 1 root root       10 Nov 25 21:34 Phase2.md
-rw-r--r-- 1 root root  91226112 Nov 29 20:47 home-credit-default-ris
k.zip
-rwxrwxrwx 1 root root   66899 Nov 25 21:34 home_credit.png
-rw-r--r-- 1 root root 375168662 Nov 29 00:41 merged_data_test.csv
-rw-r--r-- 1 root root 375168662 Nov 29 00:41 merged_data_train.csv
-rwxrwxrwx 1 root root  1320236 Nov 25 21:34 submission.csv
-rwxrwxrwx 1 root root 1091396 Nov 25 21:35 submission.png
```

```
In [12]: #!rm -r DATA_DIR
```

### 3.4.1 Imports

```
In [4]: import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
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.preprocessing import OneHotEncoder
import warnings
warnings.filterwarnings('ignore')
```

```
In [ ]: unzippingReq = True #True
if unzippingReq: #please modify this code
    zip_ref = zipfile.ZipFile(f'{DATA_DIR}/home-credit-default-risk.zip')
    # extractall(): Extract all members from the archive to the current directory
    zip_ref.extractall('DATA_DIR')
    zip_ref.close()
```

## 3.5 Data files overview

### 3.5.1 Data Dictionary

As part of the data download comes a Data Dictionary. It named `HomeCredit_columns_description.csv`

### 3.5.2 Application train

In [13]: `ls -l /root/shared/AML/I526_AML_Student/Assignments/Unit-Project-Home-`

```
-rw-r--r-- 1 root root 166133370 Nov 25 21:45 /root/shared/AML/I526_AML_Student/Assignments/Unit-Project-Home-Credit-Default-Risk/Phase2/DIR/application_train.csv
```

In [ ]:

In [1]:

```

import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
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.preprocessing import OneHotEncoder
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline

def load_data(in_path, name):
    df = pd.read_csv(in_path)
    print(f'{name}: shape is {df.shape}')
    print(df.info())
    display(df.head(5))
    return df

datasets={} # lets store the datasets in a dictionary so we can keep
ds_name = 'application_train'
DATA_DIR='/root/shared/AML/I526_AML_Student/Assignments/Unit-Project-'
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'))

datasets['application_train'].shape

```

application\_train: shape is (307511, 122)  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 307511 entries, 0 to 307510  
Columns: 122 entries, SK\_ID\_CURR to AMT\_REQ\_CREDIT\_BUREAU\_YEAR  
dtypes: float64(65), int64(41), object(16)  
memory usage: 286.2+ MB  
None

SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_C
0	100002	1	Cash loans	M	N
1	100003	0	Cash loans	F	N

2	100004	0	Revolving loans	M	Y
3	100006	0	Cash loans	F	N
4	100007	0	Cash loans	M	N

5 rows × 122 columns

Out[1]: (307511, 122)

In [2]: datasets.keys()

Out[2]: dict\_keys(['application\_train'])

In [3]: DATA\_DIR

Out[3]: '/root/shared/AML/I526\_AML\_Student/Assignments/Unit-Project-Home-Credit-Default-Risk/Phase2/DATA\_DIR/'

### 3.5.3 Application test

- **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.

```
In [41]: ds_name = 'application_test'  
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'))  
  
application_test: shape is (48744, 121)  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 48744 entries, 0 to 48743  
Columns: 121 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR  
dtypes: float64(65), int64(40), object(16)  
memory usage: 45.0+ MB  
None
```

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

5 rows × 121 columns

The application dataset has the most information about the client: Gender, income, family status, education ...

### 3.5.4 The Other datasets

- **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.

```
In [42]: ds_names = ("application_train", "application_test", "bureau", "bureau_previous_application", "POS_CASH_balance")
```

In [43]:

```
%time
ds_names = ("application_train", "application_test", "bureau", "bureau_"
             "previous_application", "POS_CASH_balance")

for ds_name in ds_names:
    datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.c
```

#	Column	Dtype
0	SK_ID_PREV	int64
1	SK_ID_CURR	int64
2	MONTHS_BALANCE	int64
3	CNT_INSTALMENT	float64
4	CNT_INSTALMENT_FUTURE	float64
5	NAME_CONTRACT_STATUS	object
6	SK_DPD	int64
7	SK_DPD_DEF	int64

dtypes: float64(2), int64(5), object(1)  
memory usage: 610.4+ MB  
None

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FL
0	1803195	182943		-31	48.0
1	1715348	367990		-33	36.0
	-----	-----	-----	--	--

In [44]:

```
for ds_name in datasets.keys():
    print(f'dataset {ds_name}: {datasets[ds_name].shape[0]}:{10},',
```

dataset application_train	:	[ 307,511, 122]
dataset application_test	:	[ 48,744, 121]
dataset bureau	:	[ 1,716,428, 17]
dataset bureau_balance	:	[ 27,299,925, 3]
dataset credit_card_balance	:	[ 3,840,312, 23]
dataset installments_payments	:	[ 13,605,401, 8]
dataset previous_application	:	[ 1,670,214, 37]
dataset POS_CASH_balance	:	[ 10,001,358, 8]

$$\text{BinaryCrossEntropy} = H_p(q) = -\frac{1}{N} \sum_{i=1}^n y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

- We primarily focus on these two performance metrics and loss functions:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN}$$

$$Sensitivity = Recall = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{FP + TN}$$

## 4 Exploratory Data Analysis

### 4.1 Summary of Application train

In [22]: `datasets["application_train"].info()`

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

In [23]: `datasets["application_train"].describe() #numerical only features`

Out[23]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AI
<b>count</b>	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307511.000000	307511.000000
<b>mean</b>	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	2.700000e+05	2.700000e+05
<b>std</b>	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	1.6016368315	1.6016368315
<b>min</b>	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	2.250000e+04	2.250000e+04
<b>25%</b>	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	1.7973000000	1.7973000000
<b>50%</b>	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	3.7390500000	3.7390500000
<b>75%</b>	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	6.750000e+05	6.750000e+05
<b>max</b>	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	2.245500e+06	2.245500e+06

8 rows × 106 columns

In [24]: `datasets["application_test"].describe() #numerical only features`

Out[24]:

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

8 rows × 105 columns

```
In [25]: datasets["application_train"].describe(include='all') #look at all cat
```

Out[25]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_C	FLAG_OWN_C_
<b>count</b>	307511.000000	307511.000000		307511	307511	307511
<b>unique</b>		NaN	NaN		2	3
<b>top</b>		NaN	NaN	Cash loans	F	F
<b>freq</b>		NaN	NaN	278232	202448	202448
<b>mean</b>	278180.518577	0.080729		NaN	NaN	NaN
<b>std</b>	102790.175348	0.272419		NaN	NaN	NaN
<b>min</b>	100002.000000	0.000000		NaN	NaN	NaN
<b>25%</b>	189145.500000	0.000000		NaN	NaN	NaN
<b>50%</b>	278202.000000	0.000000		NaN	NaN	NaN
<b>75%</b>	367142.500000	0.000000		NaN	NaN	NaN
<b>max</b>	456255.000000	1.000000		NaN	NaN	NaN

11 rows × 122 columns

## 4.2 Missing data for application train

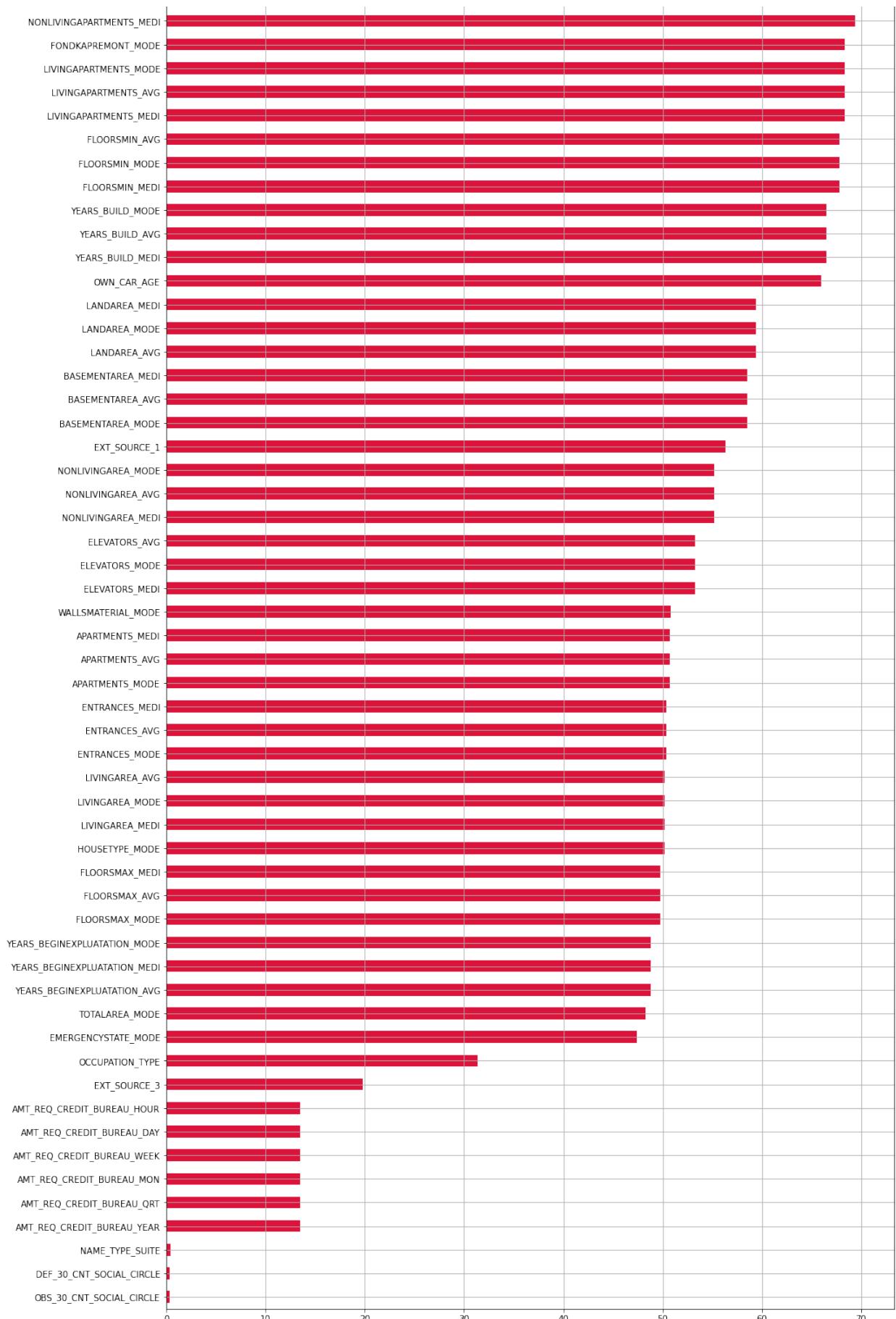
```
In [26]: percent = (datasets["application_train"].isnull().sum()/datasets["appl
sum_missing = datasets["application_train"].isna().sum().sort_values(a
missing_application_train_data = pd.concat([percent, sum_missing], ax
missing_application_train_data.head(20)
```

Out[26]:

	Percent	Train Missing	Count
COMMONAREA_MEDI	69.87	214865	
COMMONAREA_AVG	69.87	214865	
COMMONAREA_MODE	69.87	214865	
NONLIVINGAPARTMENTS_MODE	69.43	213514	
NONLIVINGAPARTMENTS_AVG	69.43	213514	
NONLIVINGAPARTMENTS_MEDI	69.43	213514	
FONDKAPREMONT_MODE	68.39	210295	
LIVINGAPARTMENTS_MODE	68.35	210199	
LIVINGAPARTMENTS_AVG	68.35	210199	
LIVINGAPARTMENTS_MEDI	68.35	210199	
FLOORSMIN_AVG	67.85	208642	
FLOORSMIN_MODE	67.85	208642	
FLOORSMIN_MEDI	67.85	208642	
YEARS_BUILD_MEDI	66.50	204488	
YEARS_BUILD_MODE	66.50	204488	
YEARS_BUILD_AVG	66.50	204488	
OWN_CAR_AGE	65.99	202929	
LANDAREA_MEDI	59.38	182590	
LANDAREA_MODE	59.38	182590	
LANDAREA_AVG	59.38	182590	

```
In [27]: plt.figure(figsize=(15, 7))
missing_application_train_data['Percent'].sort_values().tail(60).plot.
plt.grid(b=True)
plt.show();
```



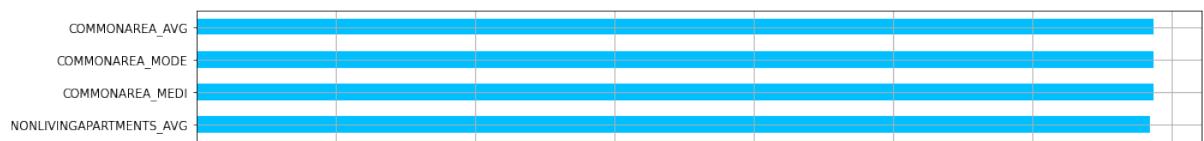


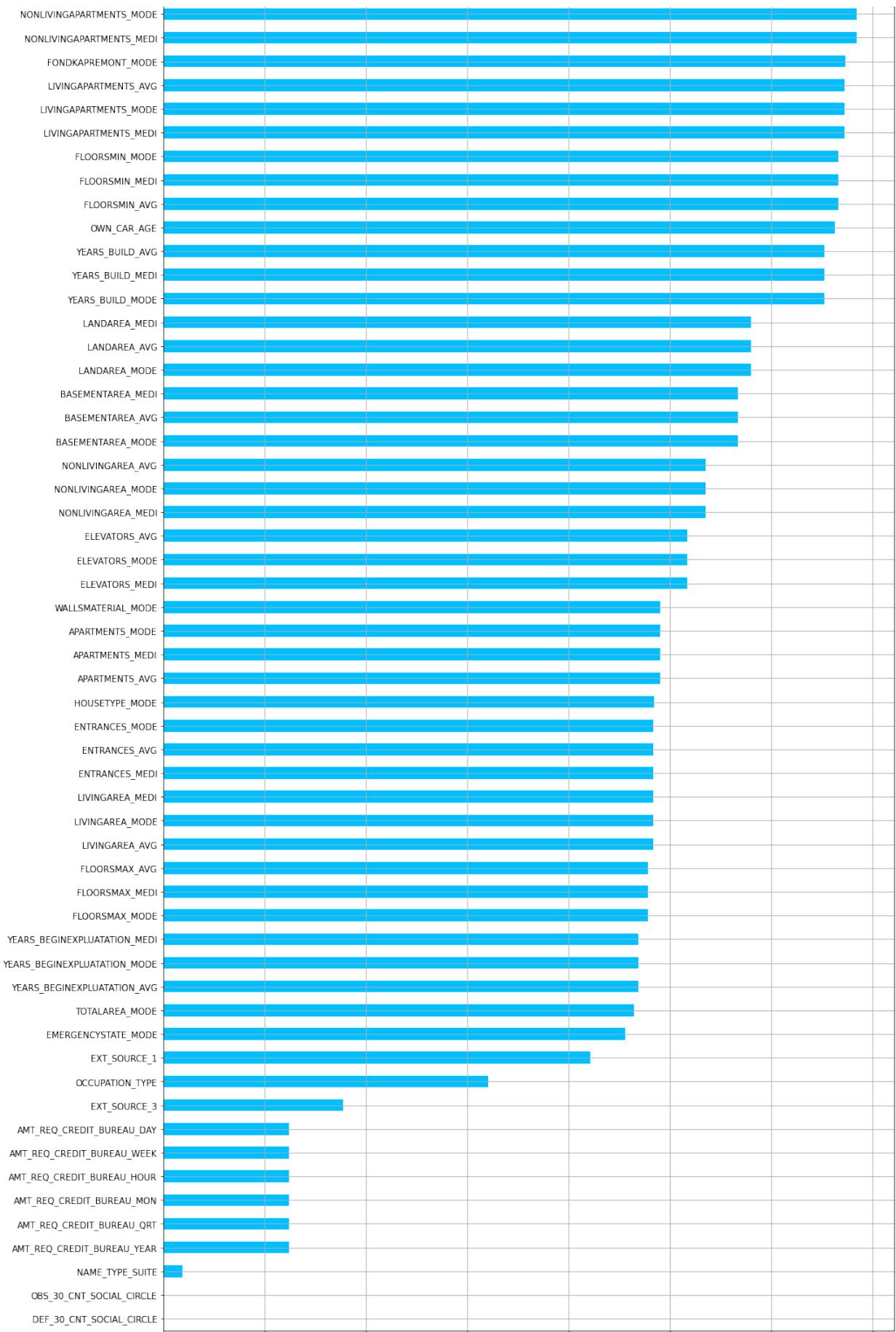
```
In [28]: percent = (datasets["application_test"].isnull().sum()/datasets["appli
sum_missing = datasets["application_test"].isna().sum().sort_values(as
missing_application_test_data = pd.concat([percent, sum_missing], axis=1)
missing_application_test_data.head(20)
```

Out[28]:

	Percent	Test Missing Count
COMMONAREA_AVG	68.72	33495
COMMONAREA_MODE	68.72	33495
COMMONAREA_MEDI	68.72	33495
NONLIVINGAPARTMENTS_AVG	68.41	33347
NONLIVINGAPARTMENTS_MODE	68.41	33347
NONLIVINGAPARTMENTS_MEDI	68.41	33347
FONDKAPREMONT_MODE	67.28	32797
LIVINGAPARTMENTS_AVG	67.25	32780
LIVINGAPARTMENTS_MODE	67.25	32780
LIVINGAPARTMENTS_MEDI	67.25	32780
FLOORSMIN_MEDI	66.61	32466
FLOORSMIN_AVG	66.61	32466
FLOORSMIN_MODE	66.61	32466
OWN_CAR_AGE	66.29	32312
YEARS_BUILD_AVG	65.28	31818
YEARS_BUILD_MEDI	65.28	31818
YEARS_BUILD_MODE	65.28	31818
LANDAREA_MEDI	57.96	28254
LANDAREA_AVG	57.96	28254
LANDAREA_MODE	57.96	28254

```
In [29]: plt.figure(figsize=(15, 7))
missing_application_test_data['Percent'].sort_values().tail(60).plot.bar()
plt.grid(b=True)
plt.show();
```





0 10 20 30 40 50 60 70

#### 4.2.1 Observation:

- We can see that a large portion of the data is missing from train and test sets

```
In [30]: # Setting up the train and test datasets
```

```
app_train = datasets["application_train"]
app_test = datasets["application_test"]
```

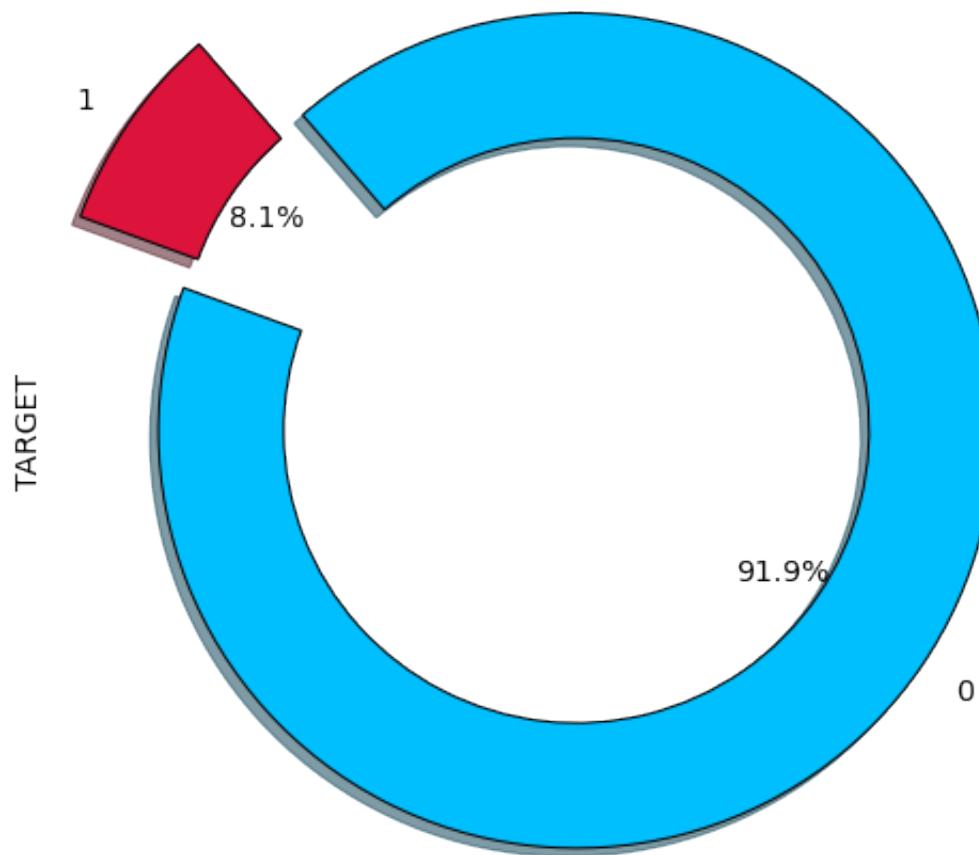
### 4.3 Distribution of the target column

```
In [31]: app_train['TARGET'].value_counts()
```

```
Out[31]: 0    282686
         1    24825
Name: TARGET, dtype: int64
```

```
In [32]: plt.figure(figsize=(9, 9))
plt.pie(x=app_train['TARGET'].value_counts(),
        radius=1.3-0.3,
        labels=app_train['TARGET'].value_counts().index,
        autopct='%1.1f%%',
        colors=['deepskyblue', 'crimson'],
        explode=[0,0.3],
        wedgeprops={"edgecolor":"0", "width":0.3},
        startangle=160,
        shadow=True,
        textprops={'fontsize': 14})
plt.ylabel('TARGET', fontsize=14)
plt.title('Distribution of TARGET feature', fontsize=16)
plt.show()
```

Distribution of TARGET feature



### 4.3.1 Observation:

- We can observe a high amount of imbalance in the TARGET feature.
- This will cause issues when measuring the accuracy performance metric.

## 4.4 Correlation with the target column

```
In [33]: correlations = datasets["application_train"].corr()["TARGET"].sort_values()
print('Most Positive Correlations:\n', correlations.tail(10))
print('\nMost Negative Correlations:\n', correlations.head(10))
```

Most Positive Correlations:

ELEVATORS_AVG	-0.034199
REGION_POPULATION_RELATIVE	-0.037227
AMT_GOODS_PRICE	-0.039645
FLOORSMAX_MODE	-0.043226
FLOORSMAX_MEDI	-0.043768
FLOORSMAX_AVG	-0.044003
DAYS_EMPLOYED	-0.044932
EXT_SOURCE_1	-0.155317
EXT_SOURCE_2	-0.160472
EXT_SOURCE_3	-0.178919

Name: TARGET, dtype: float64

Most Negative Correlations:

TARGET	1.000000
DAYS_BIRTH	0.078239
REGION_RATING_CLIENT_W_CITY	0.060893
REGION_RATING_CLIENT	0.058899
DAYS_LAST_PHONE_CHANGE	0.055218
DAYS_ID_PUBLISH	0.051457
REG_CITY_NOT_WORK_CITY	0.050994
FLAG_EMP_PHONE	0.045982
REG_CITY_NOT_LIVE_CITY	0.044395
FLAG_DOCUMENT_3	0.044346

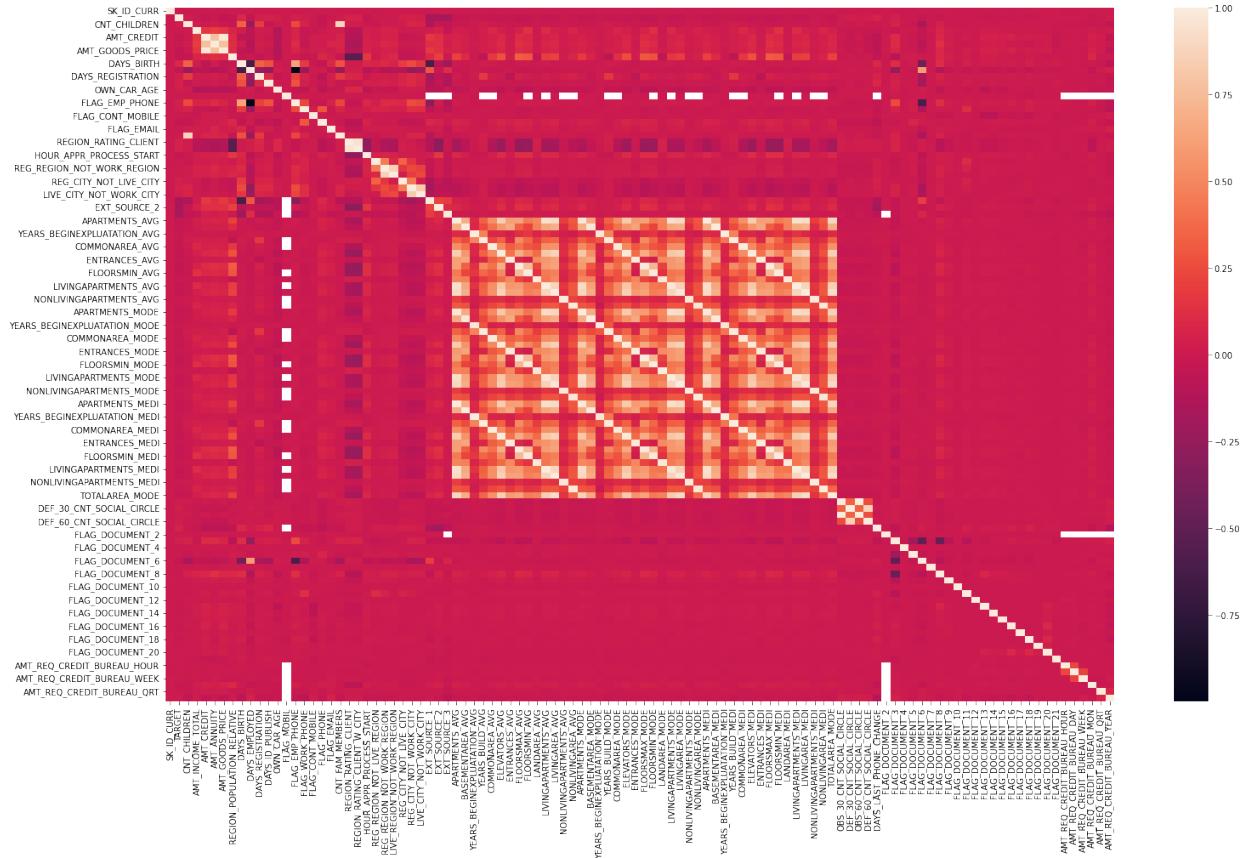
Name: TARGET, dtype: float64

#### 4.4.1 Observation:

- The maximum positive correlation with TARGET feature is observed as 0.0782 with DAYS\_BIRTH feature. We will observe that in the coming sections.
- This is followed by REGION\_RATING, DAYS\_LAST\_PHONE\_CHANGE, DAYS\_ID\_PUBLISH, and REG\_CITY\_NOT\_WORK\_CITY features.
- High indirect correlation is observed between TARGET and FLOORS features, External Sources, AMT\_GOODS\_PRICE, and relative population features.

```
In [34]: train_corr = datasets["application_train"].corr()
```

```
In [35]: plt.figure(figsize=(25, 15))
sns.heatmap(train_corr, cmap='rocket')
plt.plot();
```



#### 4.4.2 Observation:

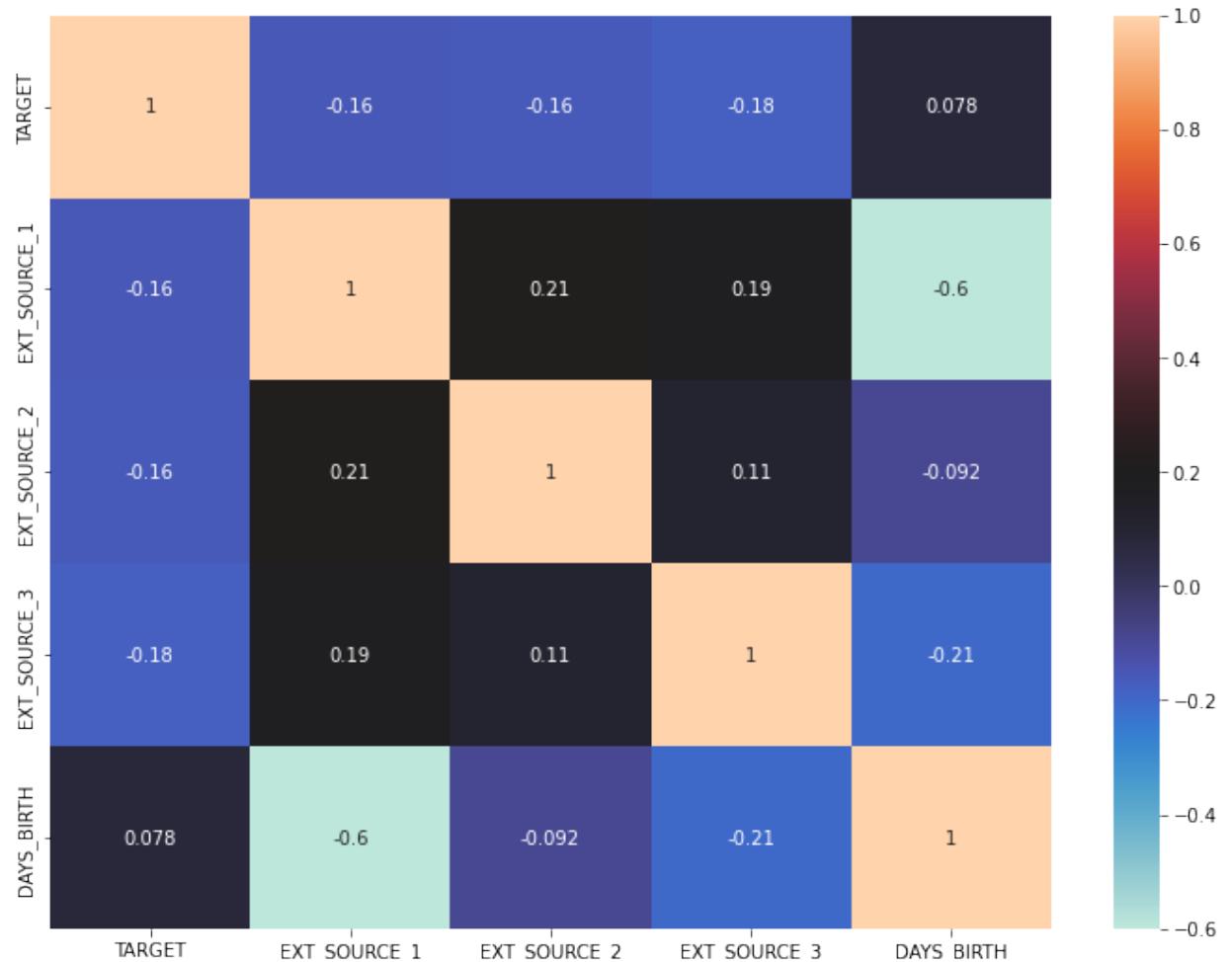
- The heatmap can't be made sense of as of now since we have 120+ columns to compare from.

```
In [36]: # Extract the EXT_SOURCE variables and show correlations
ext_source_data = app_train[['TARGET', 'EXT_SOURCE_1', 'EXT_SOURCE_2',
ext_source_data_corrs = ext_source_data.corr()
ext_source_data_corrs
```

Out [36]:

	TARGET	EXT_SOURCE_1	EXT_SOURCE_2	EXT_SOURCE_3	DAYS_BIRTH
TARGET	1.000000	-0.155317	-0.160472	-0.178919	0.078239
EXT_SOURCE_1	-0.155317	1.000000	0.213982	0.186846	-0.600610
EXT_SOURCE_2	-0.160472	0.213982	1.000000	0.109167	-0.091996
EXT_SOURCE_3	-0.178919	0.186846	0.109167	1.000000	-0.205478
DAYS_BIRTH	0.078239	-0.600610	-0.091996	-0.205478	1.000000

```
In [37]: plt.figure(figsize=(12, 9))
sns.heatmap(ext_source_data_corrs, annot=True, cmap='icefire')
plt.plot();
```



#### 4.4.3 Observation:

- The heatmap shows us that external sources indirectly affect the TARGET feature.
- But we can also see that they are correlated to each other as well i.e. multicollinearity is present.

In [38]: `app_train.describe()`

Out [38]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AM...
<b>count</b>	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	30...
<b>mean</b>	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	2...
<b>std</b>	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	1...
<b>min</b>	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	...
<b>25%</b>	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	1...
<b>50%</b>	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	2...
<b>75%</b>	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	3...
<b>max</b>	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	25...

8 rows × 106 columns

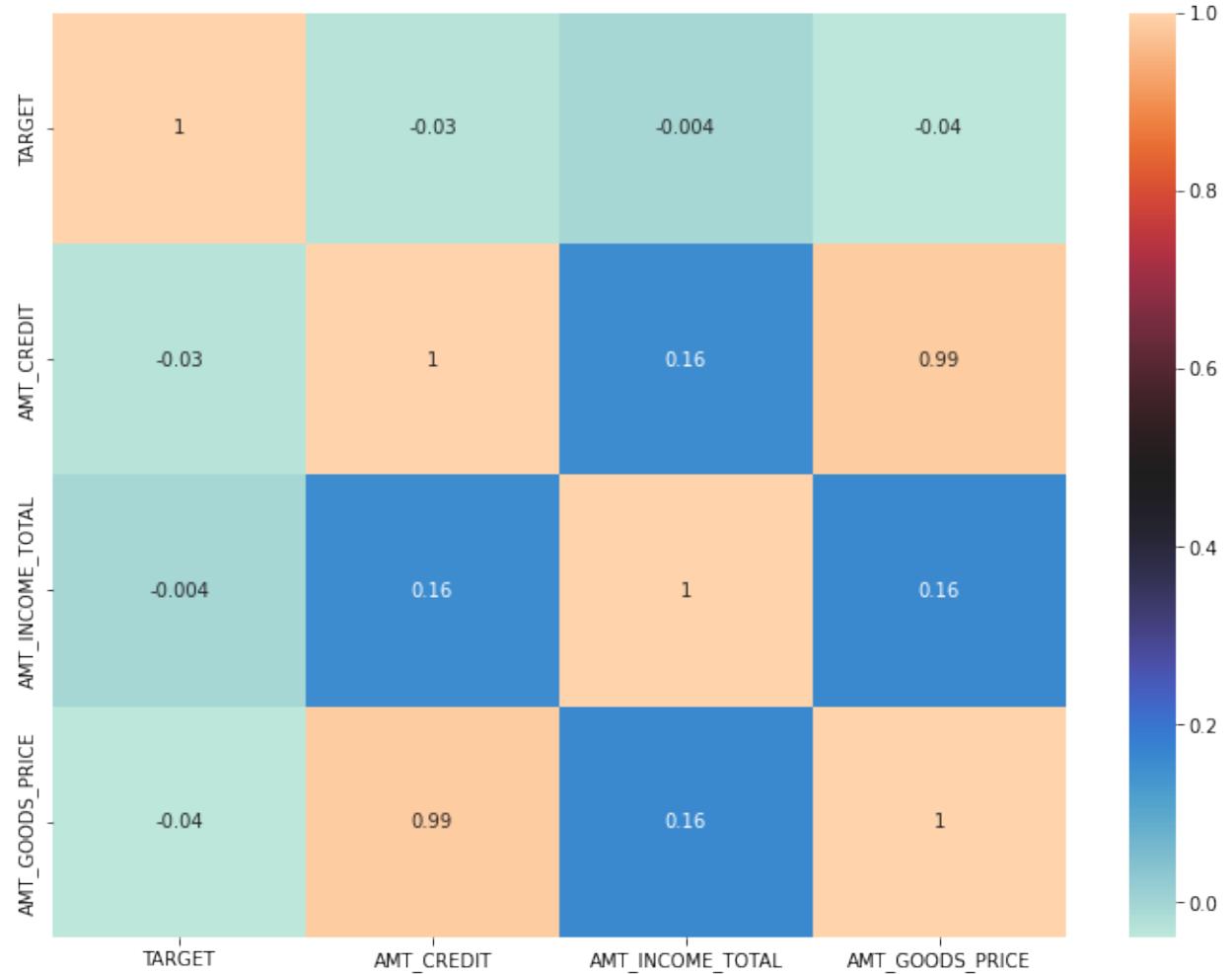
In [39]: `# Extract the AMOUNT variables and show correlations`

```
amount_data = app_train[['TARGET', 'AMT_CREDIT', 'AMT_INCOME_TOTAL', 'AMT_GOODS_PRICE']]
amount_data_corrs = amount_data.corr()
amount_data_corrs
```

Out [39]:

	TARGET	AMT_CREDIT	AMT_INCOME_TOTAL	AMT_GOODS_PRICE
<b>TARGET</b>	1.000000	-0.030369	-0.003982	-0.039645
<b>AMT_CREDIT</b>	-0.030369	1.000000	0.156870	0.986968
<b>AMT_INCOME_TOTAL</b>	-0.003982	0.156870	1.000000	0.159610
<b>AMT_GOODS_PRICE</b>	-0.039645	0.986968	0.159610	1.000000

```
In [40]: plt.figure(figsize=(12, 9))
sns.heatmap(amount_data_corrs, annot=True, cmap='icefire')
plt.plot();
```



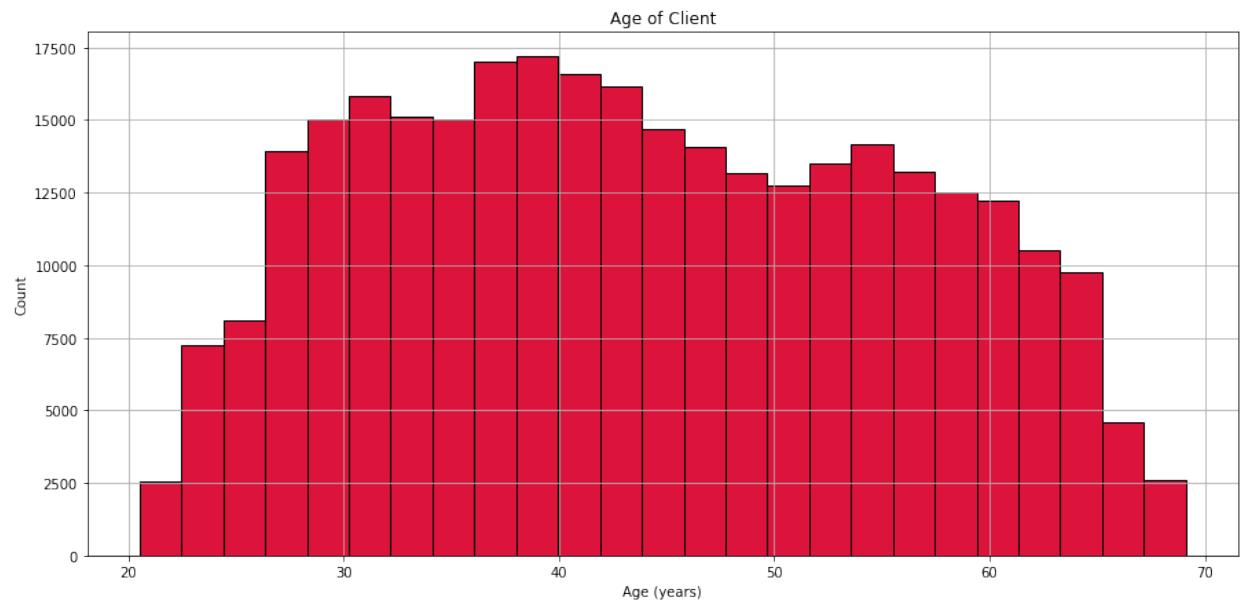
#### 4.4.4 Observation:

- The heatmap shows us yet again a case of multicollinearity among the AMT features.
- We may have to deal with these features while proceeding with the modelling.

## 4.5 Exploratory Data Analysis on Categorical Features

## 4.6 Applicants Age

```
In [41]: plt.figure(figsize=(15, 7))
plt.hist(datasets["application_train"]['DAYS_BIRTH'] / -365, edgecolor='black', bins=50)
plt.title('Age of Client')
plt.xlabel('Age (years)')
plt.ylabel('Count')
plt.grid(b=True)
plt.show()
```



#### 4.6.1 Observation:

- Age is obtained by the DAYS\_BIRTH feature which has negative values. This is inconsistent and should be taken care of.
- On plotting the age as number of years, we see a fairly standard distribution which is a good sign in such a complicated dataset as we have DAYS\_BIRTH highly correlated with TARGET feature.

```
In [42]: # Age information into a separate dataframe
age_data = app_train[['TARGET', 'DAYS_BIRTH']]
age_data['YEARS_BIRTH'] = age_data['DAYS_BIRTH'] / -365

# Bin the age data
age_data['GROUPED_YEARS_BIRTH'] = pd.cut(age_data['YEARS_BIRTH'], bins)
age_data.head(10)
```

Out [42]:

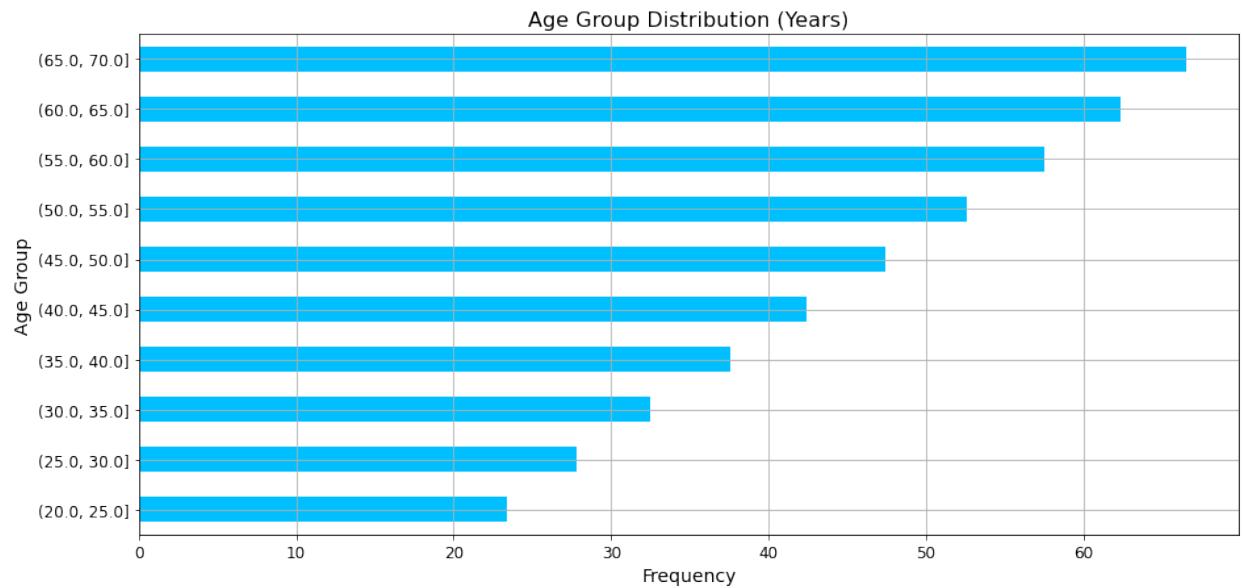
	TARGET	DAYS_BIRTH	YEARS_BIRTH	GROUPED_YEARS_BIRTH
0	1	-9461	25.920548	(25.0, 30.0]
1	0	-16765	45.931507	(45.0, 50.0]
2	0	-19046	52.180822	(50.0, 55.0]
3	0	-19005	52.068493	(50.0, 55.0]
4	0	-19932	54.608219	(50.0, 55.0]
5	0	-16941	46.413699	(45.0, 50.0]
6	0	-13778	37.747945	(35.0, 40.0]
7	0	-18850	51.643836	(50.0, 55.0]
8	0	-20099	55.065753	(55.0, 60.0]
9	0	-14469	39.641096	(35.0, 40.0]

In [43]: age\_groups = age\_data.groupby('GROUPED\_YEARS\_BIRTH').mean()

Out [43]:

GROUPED_YEARS_BIRTH	TARGET	DAYS_BIRTH	YEARS_BIRTH
(20.0, 25.0]	0.123036	-8532.795625	23.377522
(25.0, 30.0]	0.111436	-10155.219250	27.822518
(30.0, 35.0]	0.102814	-11854.848377	32.479037
(35.0, 40.0]	0.089414	-13707.908253	37.555913
(40.0, 45.0]	0.078491	-15497.661233	42.459346
(45.0, 50.0]	0.074171	-17323.900441	47.462741
(50.0, 55.0]	0.066968	-19196.494791	52.593136
(55.0, 60.0]	0.055314	-20984.262742	57.491131
(60.0, 65.0]	0.052737	-22780.547460	62.412459
(65.0, 70.0]	0.037270	-24292.614340	66.555108

```
In [44]: age_groups['YEARS_BIRTH'].plot.barh(figsize=(15, 7), color='deepskyblue'
plt.title('Age Group Distribution (Years)', fontsize=16)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.xlabel('Frequency', fontsize=14)
plt.ylabel('Age Group', fontsize=14)
plt.grid(b=True)
plt.show()
```

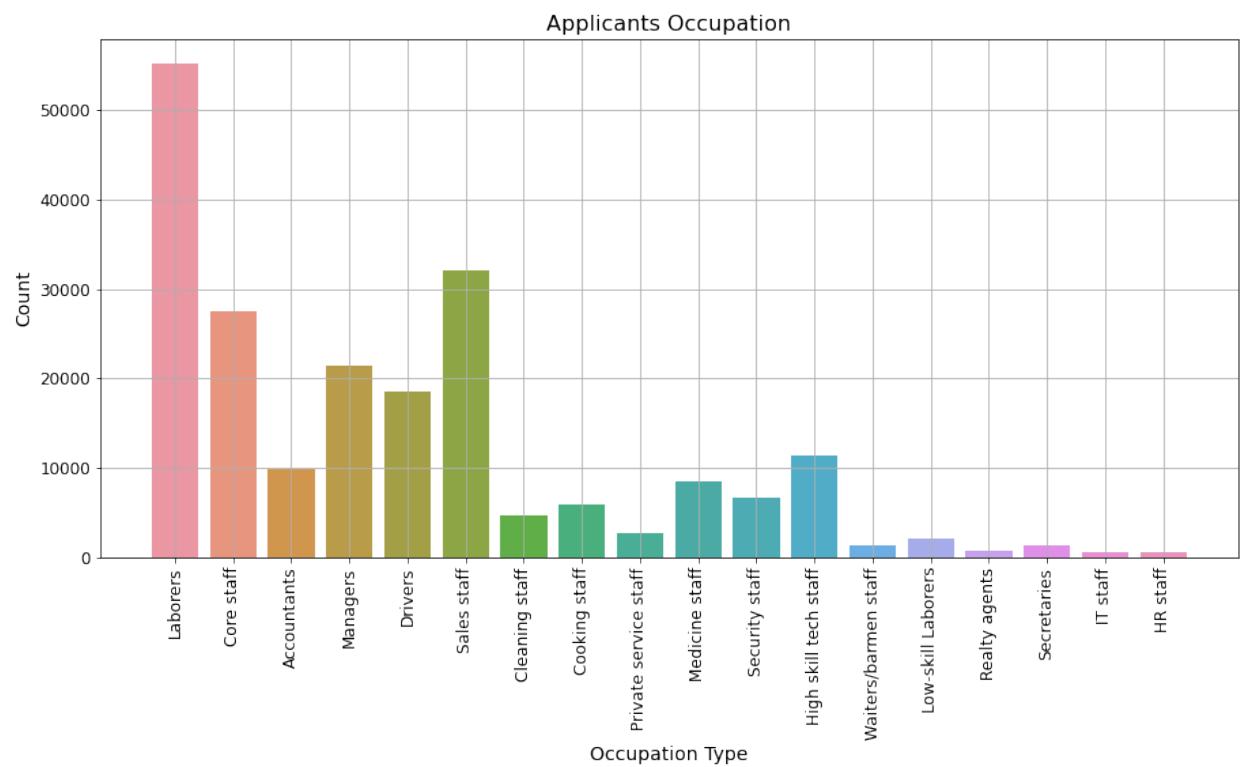


#### 4.6.2 Observation:

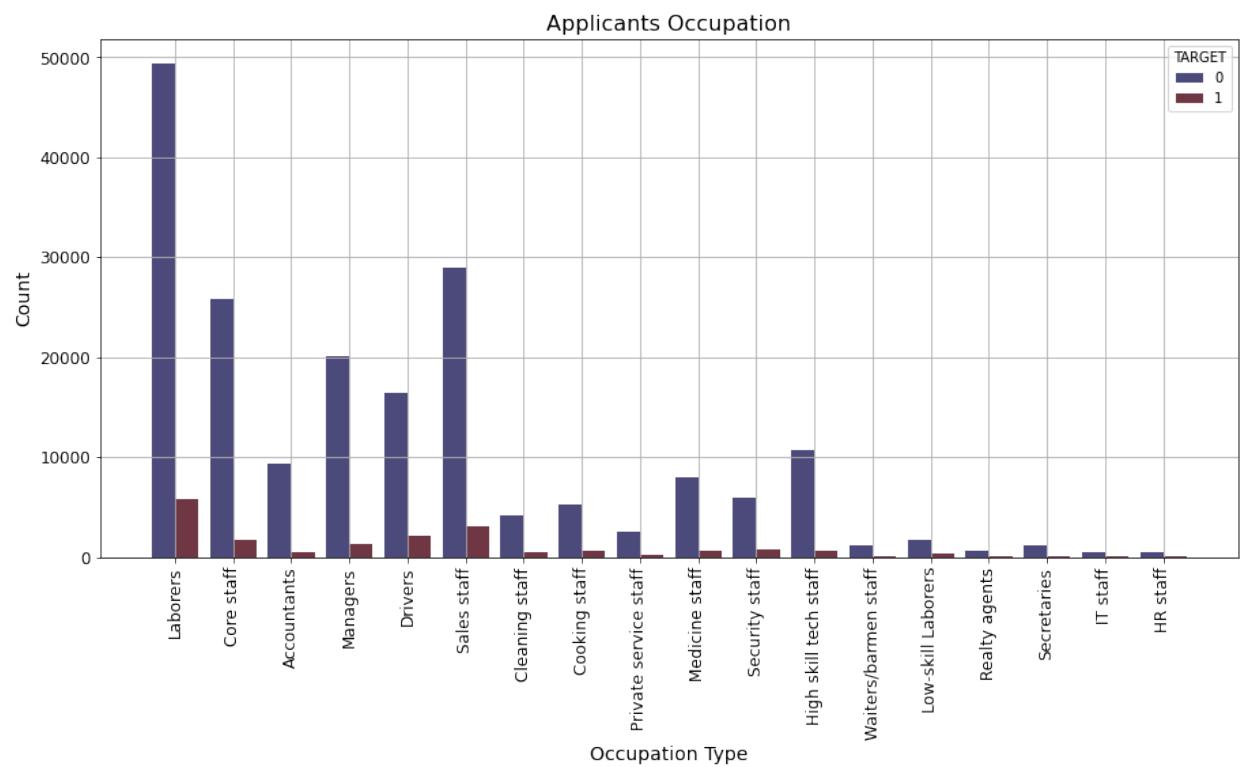
- After performing binning, we can observe that older people tend to take more loans than younger people.

### 4.7 Applicants occupations

```
In [45]: plt.figure(figsize=(15, 7))
sns.countplot(x='OCCUPATION_TYPE', data=datasets["application_train"])
plt.title('Applicants Occupation', fontsize=16)
plt.xlabel('Occupation Type', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(rotation=90, fontsize=12)
plt.yticks(fontsize=12)
plt.grid(b=True)
plt.plot();
```



```
In [46]: plt.figure(figsize=(15, 7))
sns.countplot(x='OCCUPATION_TYPE', data=datasets["application_train"],
plt.title('Applicants Occupation', fontsize=16)
plt.xlabel('Occupation Type', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(rotation=90, fontsize=12)
plt.yticks(fontsize=12)
plt.grid(b=True)
plt.plot();
```



#### 4.7.1 Observation:

- We see 18 different occupations among the borrowers, led by Laborers, Sales staff, Core staff, Managers and Drivers.
- We can't observe any specific trend as to which occupation class successfully pays back their loan.

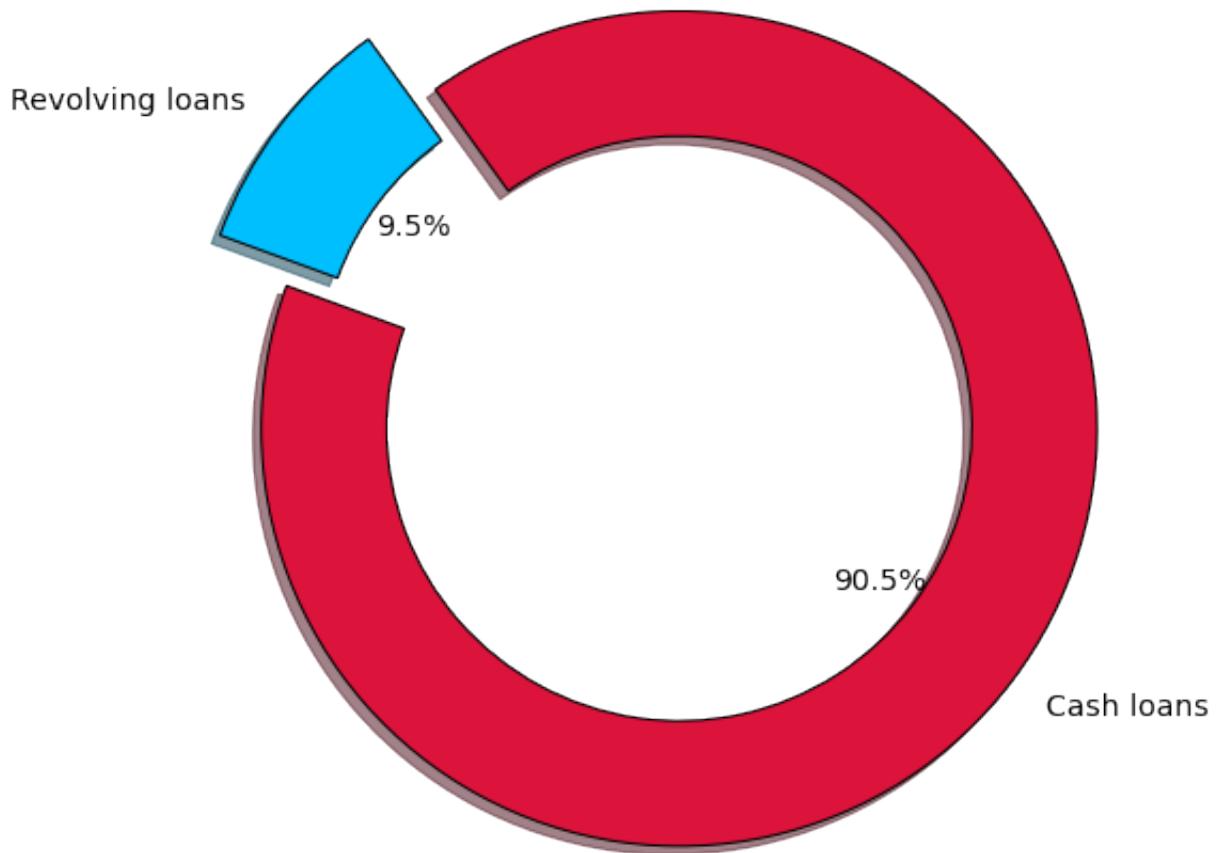
## 4.8 Applicant Contract Type

```
In [47]: app_train['NAME_CONTRACT_TYPE'].value_counts()
```

```
Out[47]: Cash loans      278232  
Revolving loans     29279  
Name: NAME_CONTRACT_TYPE, dtype: int64
```

```
In [48]: plt.figure(figsize=(9, 9))
plt.pie(x=app_train['NAME_CONTRACT_TYPE'].value_counts(),
        radius=1.3-0.3,
        labels=app_train['NAME_CONTRACT_TYPE'].value_counts().index,
        autopct='%.1f%%',
        colors=['crimson', 'deepskyblue'],
        explode=[0,0.2],
        wedgeprops={"edgecolor":"0", "width":0.3},
        startangle=160,
        shadow=True,
        textprops={'fontsize': 14})
plt.ylabel('', fontsize=14)
plt.title("Applicant's Contract Type", fontsize=16)
plt.show()
```

Applicant's Contract Type



#### 4.8.1 Observation:

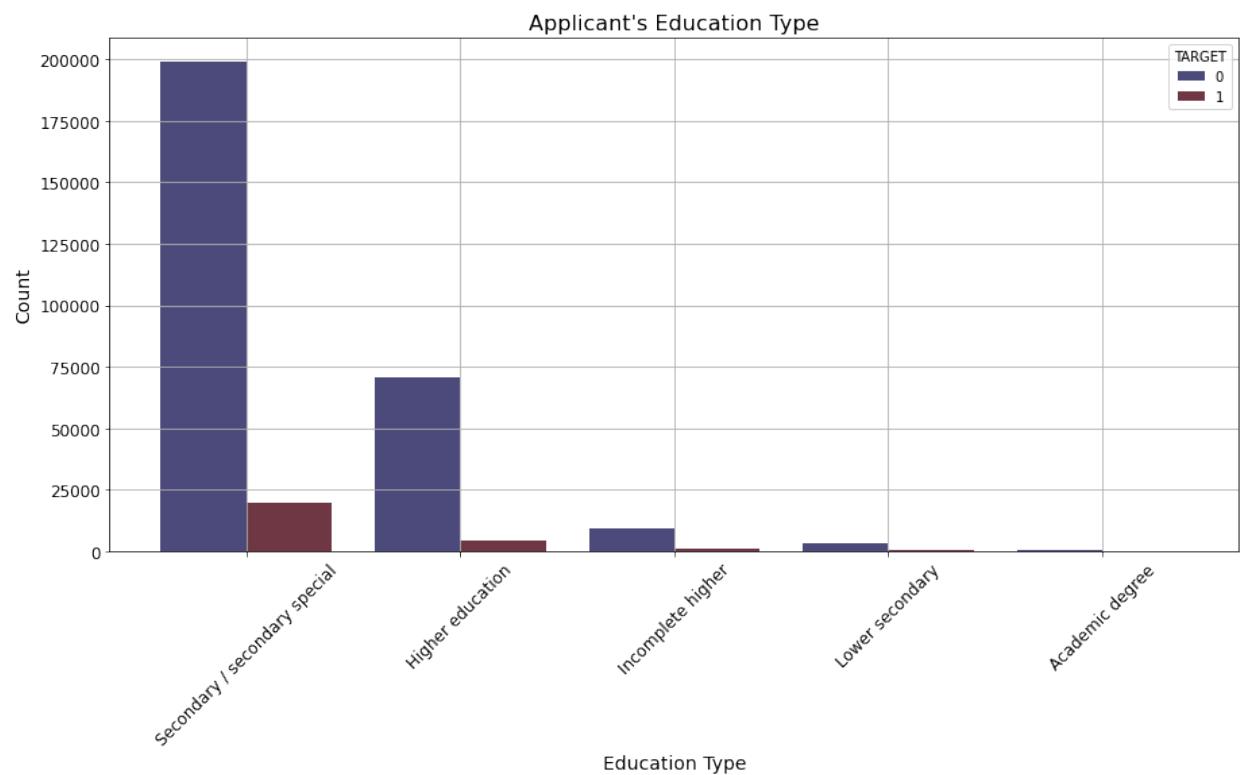
- There are two type of loan contracts - Cash loans (90.5%) and Revolving (re-pay and re-borrow again and again) loans (9.5%)

### 4.9 Applicant Education Type v/s TARGET

```
In [49]: app_train['NAME_EDUCATION_TYPE'].value_counts()
```

```
Out[49]: Secondary / secondary special    218391
Higher education                      74863
Incomplete higher                      10277
Lower secondary                         3816
Academic degree                         164
Name: NAME_EDUCATION_TYPE, dtype: int64
```

```
In [50]: plt.figure(figsize=(15, 7))
sns.countplot(x='NAME_EDUCATION_TYPE', data=app_train, palette='icefire'
plt.title("Applicant's Education Type", fontsize=16)
plt.xlabel('Education Type', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(rotation=45, fontsize=12)
plt.yticks(fontsize=12)
plt.grid(b=True)
plt.plot();
```



#### 4.9.1 Observation:

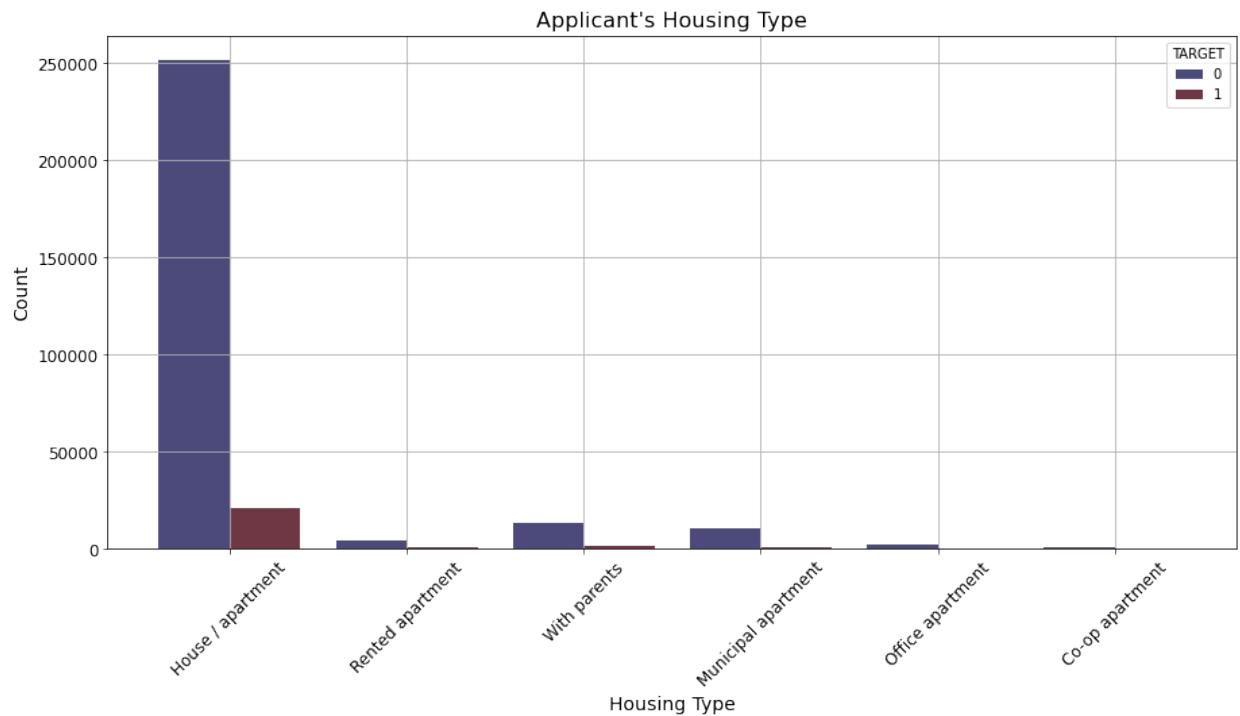
- We see most of the applicants have highest education of secondary and higher education with the lowest being Academic degree.

## 4.10 Applicant Housing Type v/s TARGET

```
In [51]: app_train['NAME_HOUSING_TYPE'].value_counts()
```

```
Out[51]: House / apartment    272868
With parents        14840
Municipal apartment  11183
Rented apartment     4881
Office apartment      2617
Co-op apartment       1122
Name: NAME_HOUSING_TYPE, dtype: int64
```

```
In [52]: plt.figure(figsize=(15, 7))
sns.countplot(x='NAME_HOUSING_TYPE', data=app_train, palette='icefire')
plt.title("Applicant's Housing Type", fontsize=16)
plt.xlabel('Housing Type', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(rotation=45, fontsize=12)
plt.yticks(fontsize=12)
plt.grid(b=True)
plt.plot();
```



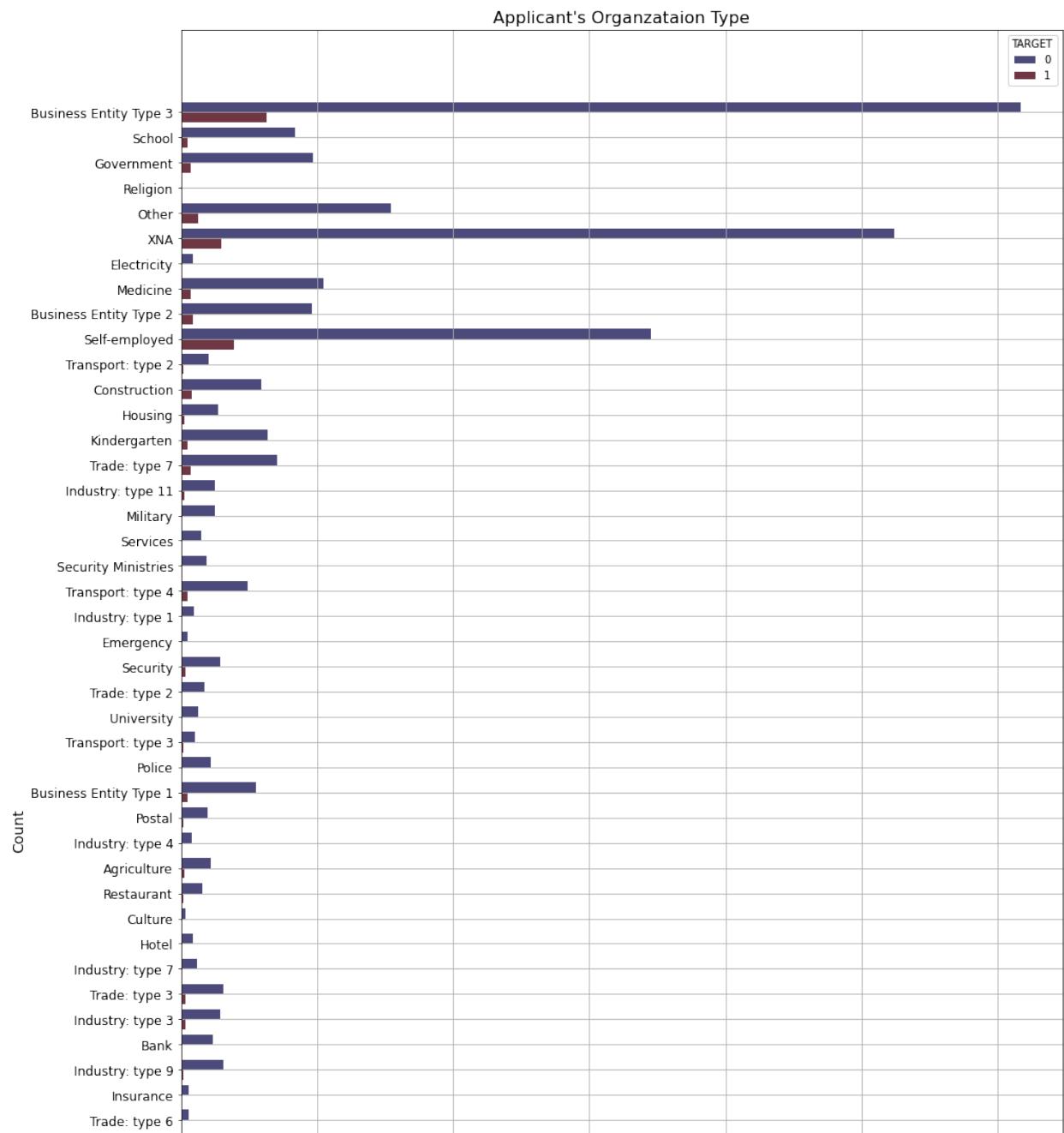
#### 4.10.1 Observation:

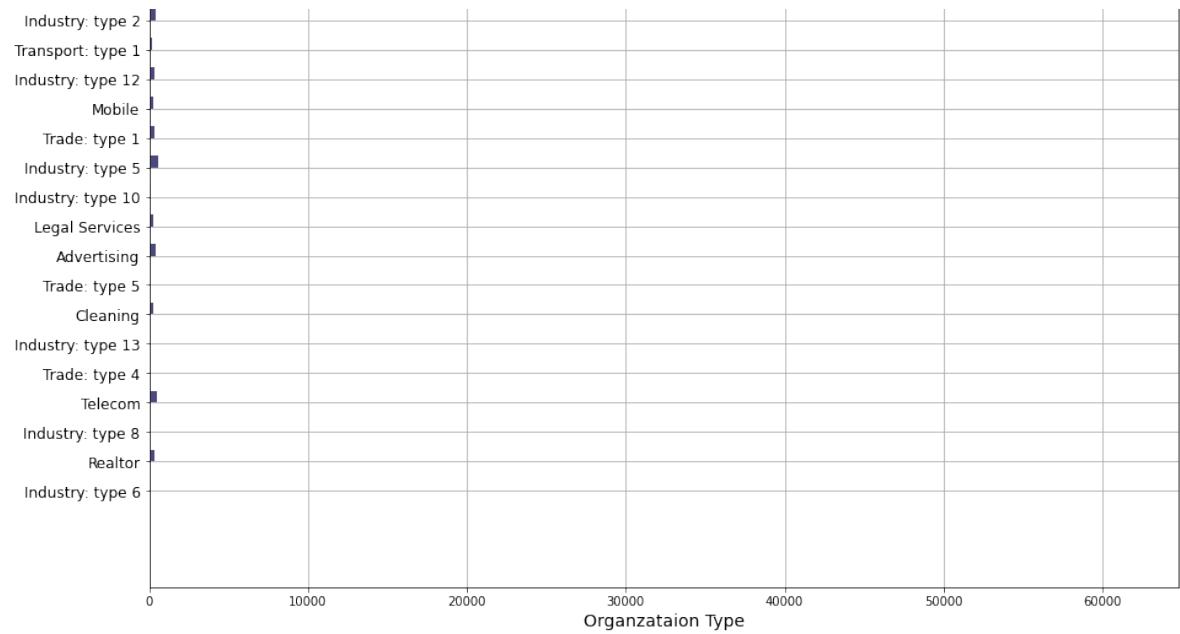
- Most of the applicant's current house type is a house or an apartment followed by living with parents (searching for a new home for themselves) and municipal apartment.

## 4.11 Applicant Organization Type

In [53]:

```
plt.figure(figsize=(15, 28))
sns.countplot(y='ORGANIZATION_TYPE', data=app_train, palette='icefire'
plt.title("Applicant's Organzataion Type", fontsize=16)
plt.xlabel('Organzataion Type', fontsize=14)
plt.ylabel('Count', fontsize=14)
# plt.xticks(rotation=45, fontsize=12)
plt.yticks(fontsize=12)
plt.grid(b=True)
plt.plot();
```





#### 4.11.1 Observation:

- There are several organization types across various applicants predominantly from Type 3 business entities and self-employment.

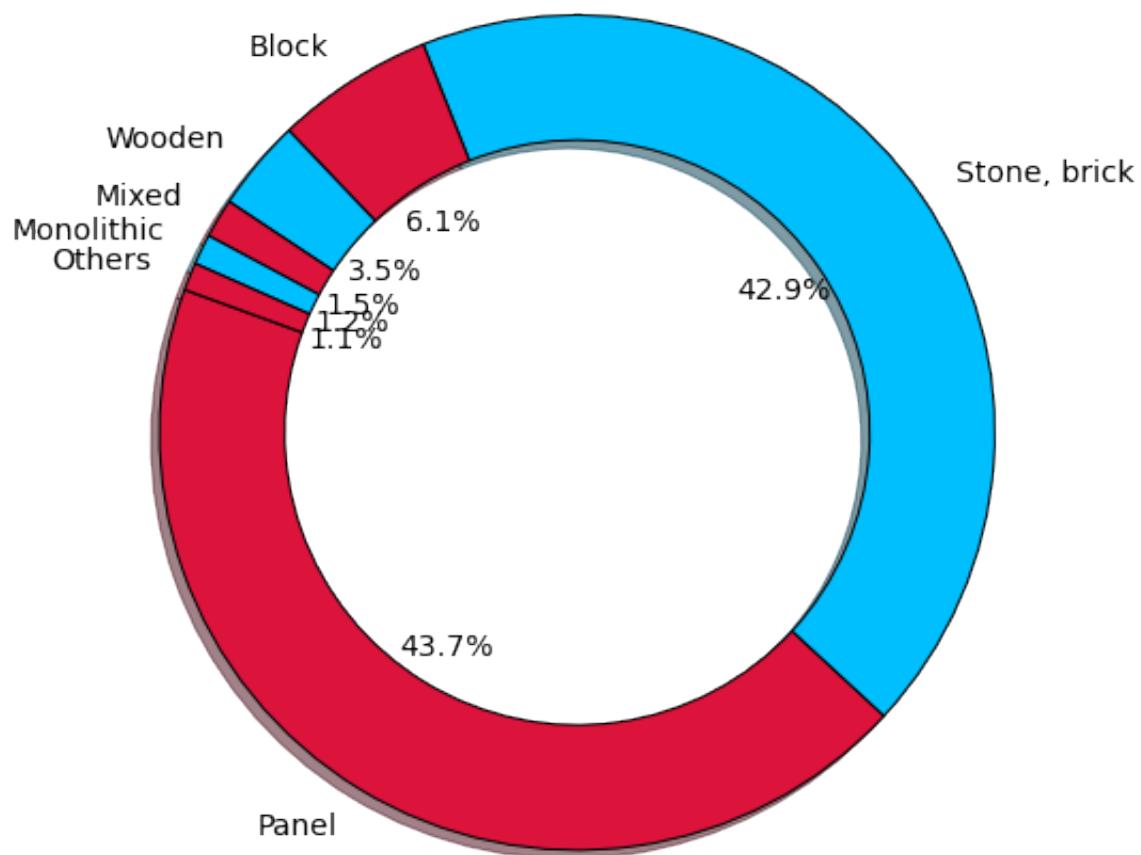
## 4.12 Applicant's House Wall Material Type

```
In [54]: app_train['WALLSMATERIAL_MODE'].value_counts()
```

```
Out[54]: Panel      66040
Stone, brick    64815
Block          9253
Wooden         5362
Mixed          2296
Monolithic     1779
Others          1625
Name: WALLSMATERIAL_MODE, dtype: int64
```

```
In [55]: plt.figure(figsize=(9, 9))
plt.pie(x=app_train['WALLSMATERIAL_MODE'].value_counts(),
        radius=1.3-0.3,
        labels=app_train['WALLSMATERIAL_MODE'].value_counts().index,
        autopct='%.1f%%',
        colors=['crimson', 'deepskyblue'],
        # explode=[0.2,0.2,0,0.2,0,0.2,0],
        wedgeprops={"edgecolor":"0", "width":0.3},
        startangle=160,
        shadow=True,
        textprops={'fontsize': 14})
plt.ylabel('', fontsize=14)
plt.title("Applicant's House's Wall Material Type", fontsize=16)
plt.show()
```

Applicant's House's Wall Material Type



#### 4.12.1 Observation:

- Most Applicant's house's wall material are made of panels or stones and bricks, followed by cement blocks, wood or a mix of the earlier mentioned.

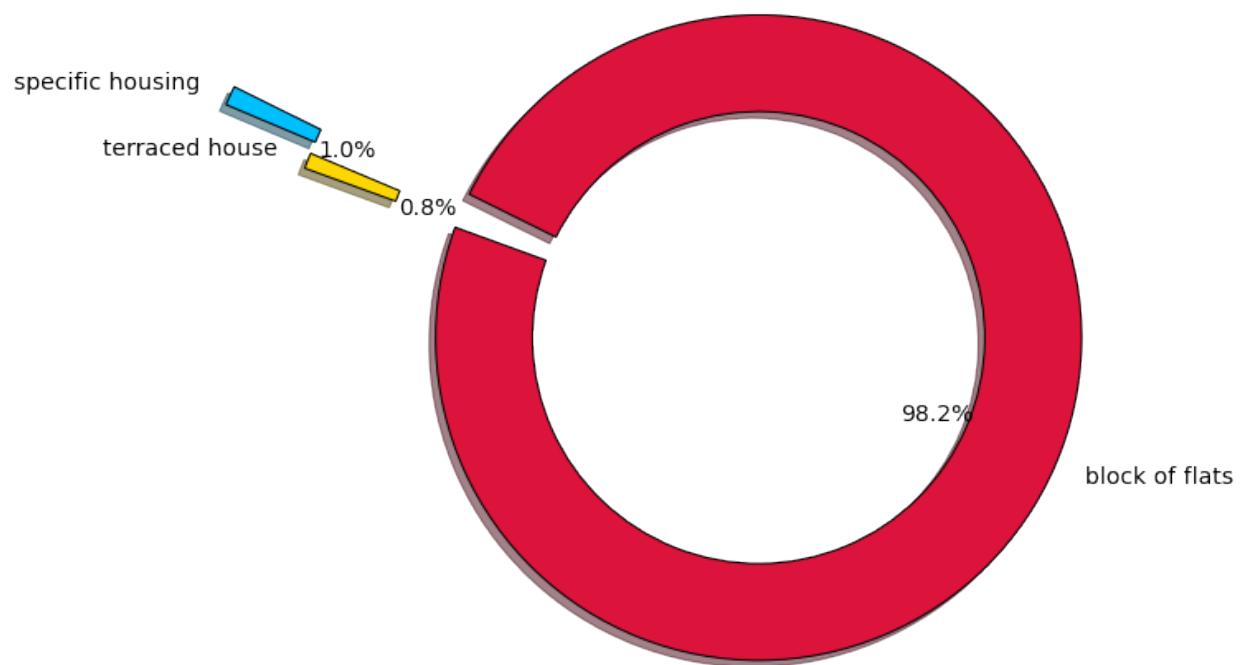
### 4.13 Applicant's House Type Part 2

```
In [56]: app_train['HOUSETYPE_MODE'].value_counts()
```

```
Out[56]: block of flats      150503  
specific housing          1499  
terraced house            1212  
Name: HOUSETYPE_MODE, dtype: int64
```

```
In [57]: plt.figure(figsize=(9, 9))
plt.pie(x=app_train['HOUSETYPE_MODE'].value_counts(),
        radius=1.3-0.3,
        labels=app_train['HOUSETYPE_MODE'].value_counts().index,
        autopct='%.1f%%',
        colors=['crimson', 'deepskyblue', 'gold'],
        explode=[0, 0.8, 0.5],
        wedgeprops={"edgecolor": "0", "width": 0.3},
        startangle=160,
        shadow=True,
        textprops={'fontsize': 14})
plt.ylabel('', fontsize=14)
plt.suptitle("Applicant's House Type", fontsize=16)
plt.show()
```

Applicant's House Type



#### 4.13.1 Observation:

- Applicants mostly reside in flats (more than 98%) while the remaining either live in terraced or other specific house types.

## 4.14 Does an Applicant already own Realty?

In [58]: `app_train['FLAG OWN REALTY'].value_counts()`

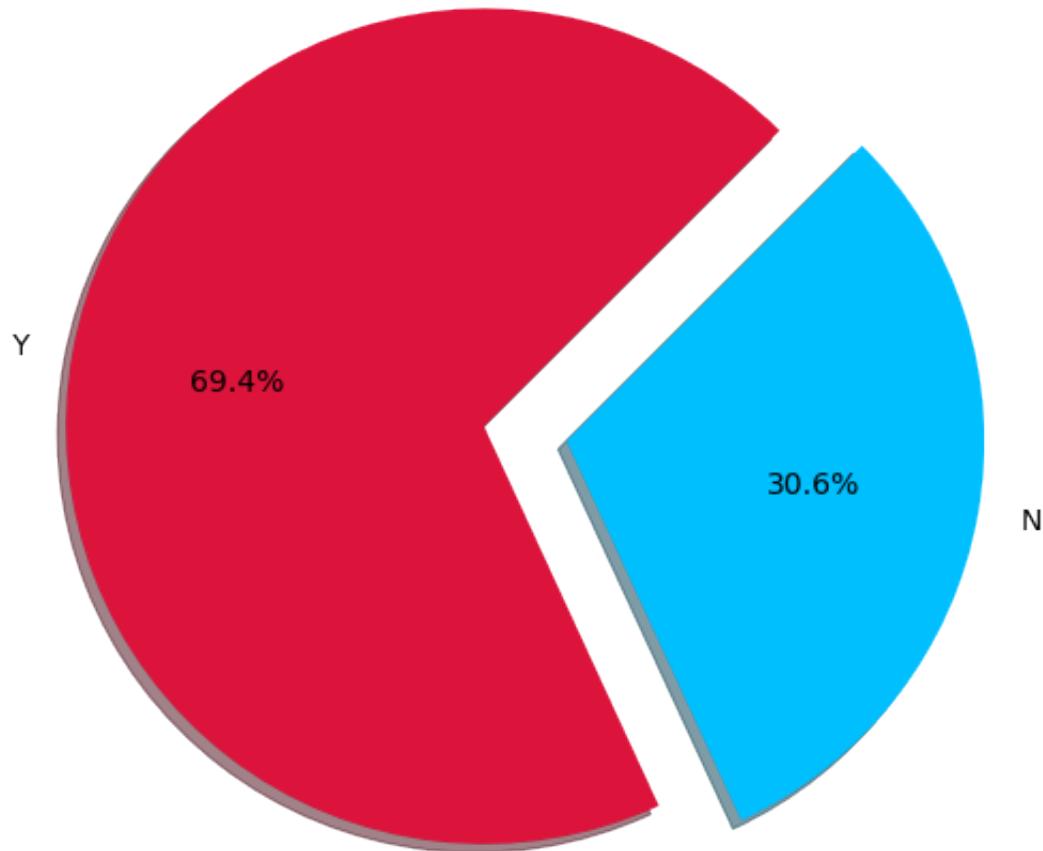
Out[58]:

Y	213312
N	94199

Name: FLAG OWN REALTY, dtype: int64

```
In [59]: plt.figure(figsize=(9, 9))
plt.pie(x=app_train['FLAG_OWN_REALTY'].value_counts(),
        radius=1.3-0.3,
        labels=app_train['FLAG_OWN_REALTY'].value_counts().index,
        autopct='%.1f%%',
        colors=['crimson', 'deepskyblue'],
        explode=[0,0.2],
        # wedgeprops={"edgecolor": "0", "width":0.3},
        startangle=45,
        shadow=True,
        textprops={'fontsize': 14})
plt.ylabel(' ', fontsize=14)
plt.suptitle("Does the Application Own a Realty?", fontsize=16)
plt.show()
```

Does the Application Own a Realty?

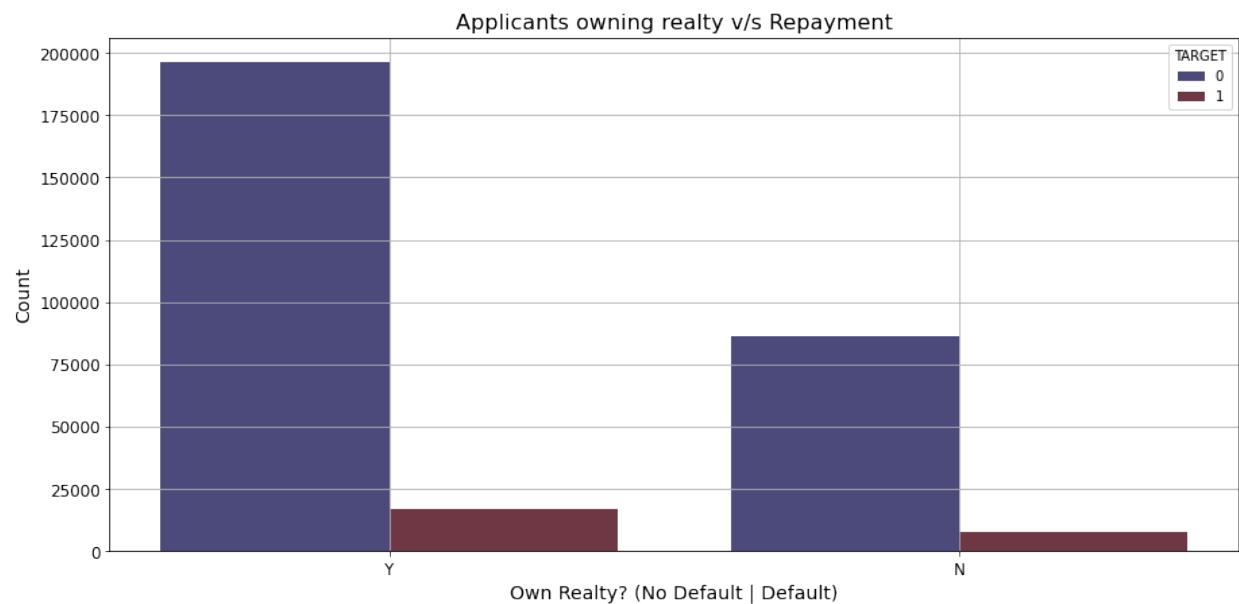


#### 4.14.1 Observation:

- 69.4% Applicants already own a realty. Let's see the distribution of the repayment.

In [60]:

```
plt.figure(figsize=(15, 7))
sns.countplot(x='FLAG_OWN_REALTY', data=app_train, palette='icefire',
plt.title("Applicants owning realty v/s Repayment", fontsize=16)
plt.xlabel('Own Realty? (No Default | Default)', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.grid(b=True)
plt.plot();
```



#### 4.14.2 Observation:

- Most of the applicants in either class are not in default. Less than 25000 applicants own a realty and are in default for their repayment.

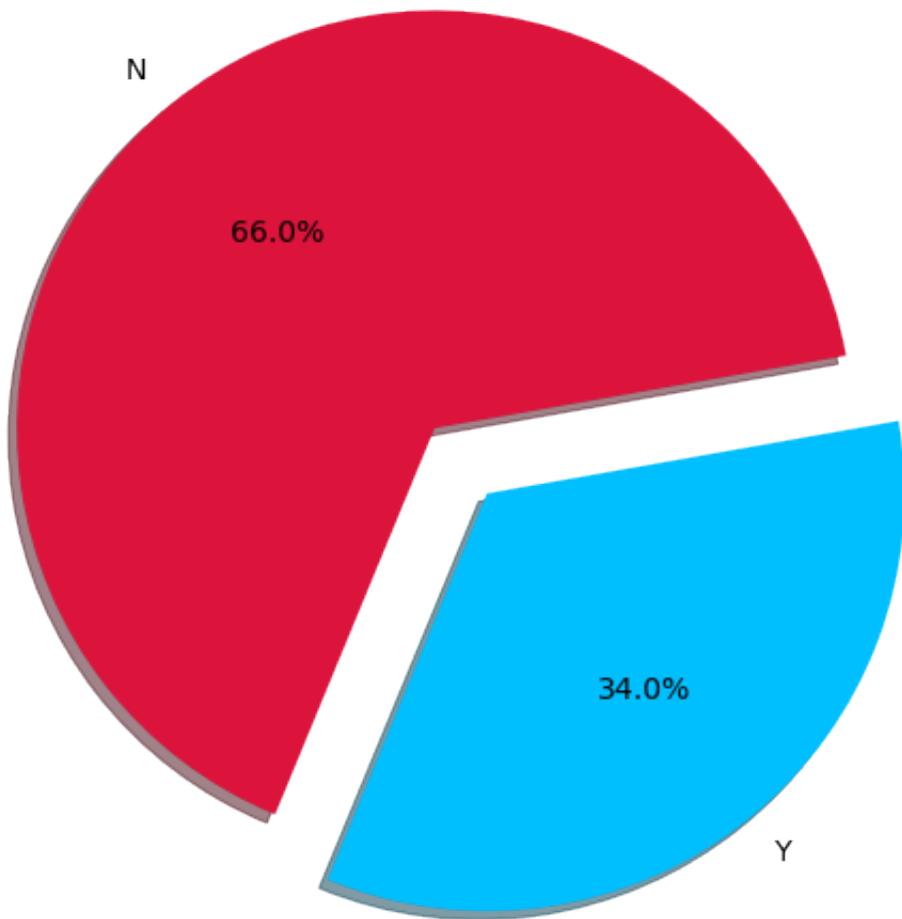
### 4.15 Does an Applicant own cars?

```
In [61]: app_train['FLAG OWN CAR'].value_counts()
```

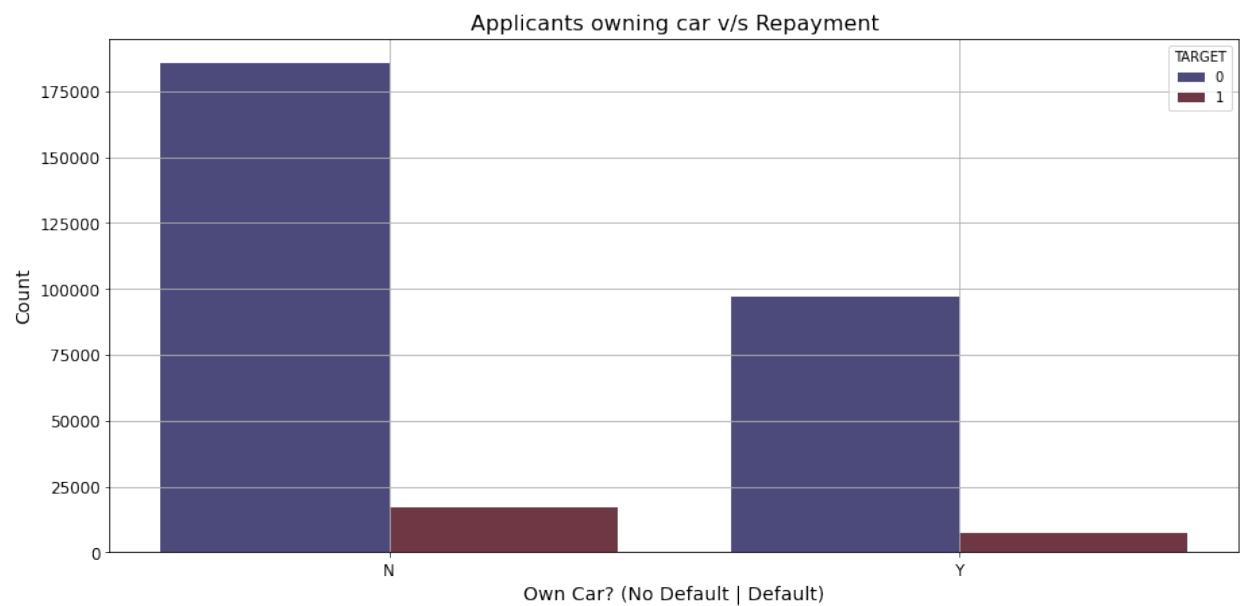
```
Out[61]: N    202924  
Y    104587  
Name: FLAG OWN CAR, dtype: int64
```

```
In [62]: plt.figure(figsize=(9, 9))
plt.pie(x=app_train['FLAG_OWN_CAR'].value_counts(),
        radius=1.3-0.3,
        labels=app_train['FLAG_OWN_CAR'].value_counts().index,
        autopct='%.1f%%',
        colors=['crimson', 'deepskyblue'],
        explode=[0,0.2],
        # wedgeprops={"edgecolor": "0", "width":0.3},
        startangle=10,
        shadow=True,
        textprops={'fontsize': 14})
plt.ylabel('', fontsize=14)
plt.suptitle("Does the Application Own a Car?", fontsize=16)
plt.show()
```

Does the Application Own a Car?



```
In [63]: plt.figure(figsize=(15, 7))
sns.countplot(x='FLAG_own_car', data=app_train, palette='icefire', hue
plt.title("Applicants owning car v/s Repayment", fontsize=16)
plt.xlabel('Own Car? (No Default | Default)', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.grid(b=True)
plt.plot();
```

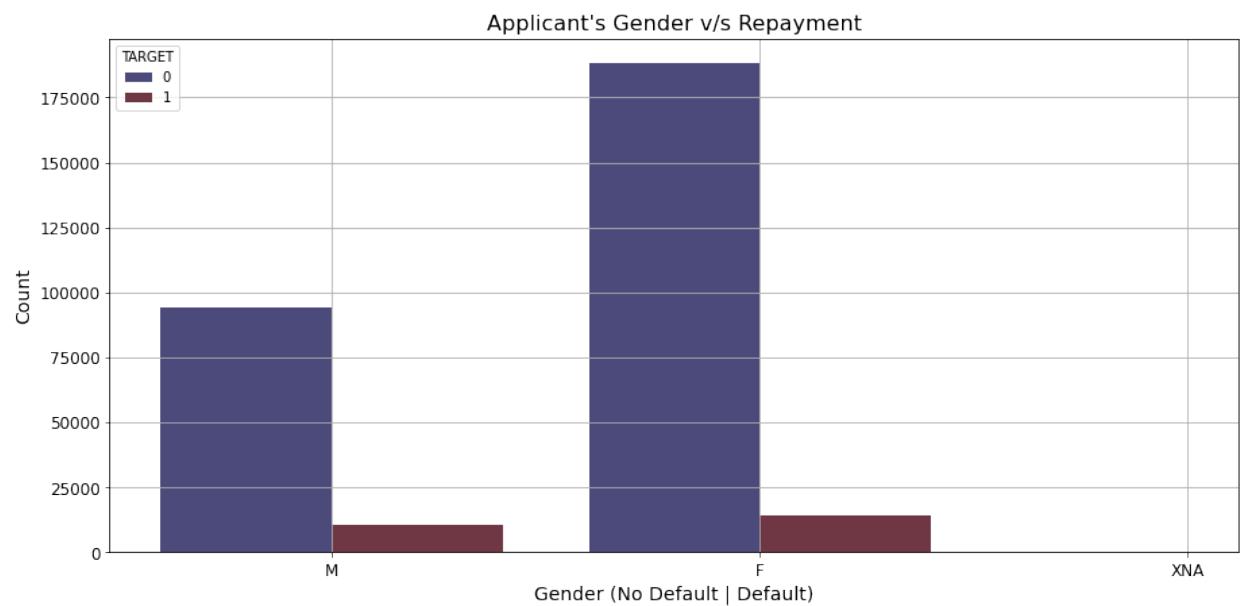


#### 4.15.1 Observation:

- 34% applicants own atleast one car.
- No specific relation to be noted here when it comes to finding a relation between car ownership and loan repayment.

### 4.16 Which Gender seems more likely to take and repay loans?

```
In [64]: plt.figure(figsize=(15, 7))
sns.countplot(x='CODE_GENDER', data=app_train, palette='icefire', hue=
plt.title("Applicant's Gender v/s Repayment", fontsize=16)
plt.xlabel('Gender (No Default | Default)', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.grid(b=True)
plt.plot();
```



#### 4.16.1 Observation:

- Most of the applicants are females and most of them have no default in their history.
- In case of male applicants, we can see that relatively higher number of applicants are in default.

### 4.17 Exploratory Data Analysis on Numeric/Continuous Features

### 4.18 Target v/s Age (in years)

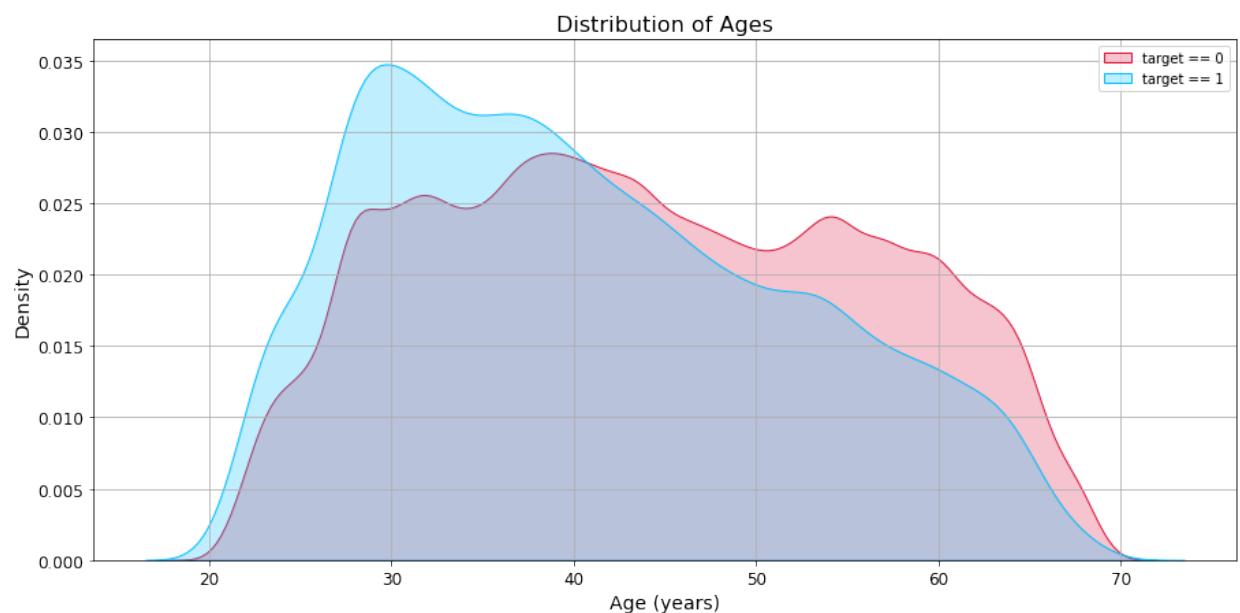
```
In [65]: plt.figure(figsize = (15, 7))

# KDE plot of loans that were repaid on time
sns.kdeplot(app_train.loc[app_train['TARGET'] == 0, 'DAYS_BIRTH'] / -3)

# KDE plot of loans which were not repaid on time
sns.kdeplot(app_train.loc[app_train['TARGET'] == 1, 'DAYS_BIRTH'] / -3

# Labeling of plot
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.xlabel('Age (years)', fontsize=14)
plt.ylabel('Density', fontsize=14)
plt.title('Distribution of Ages', fontsize=16)

plt.legend()
plt.grid(b=True)
plt.show()
```

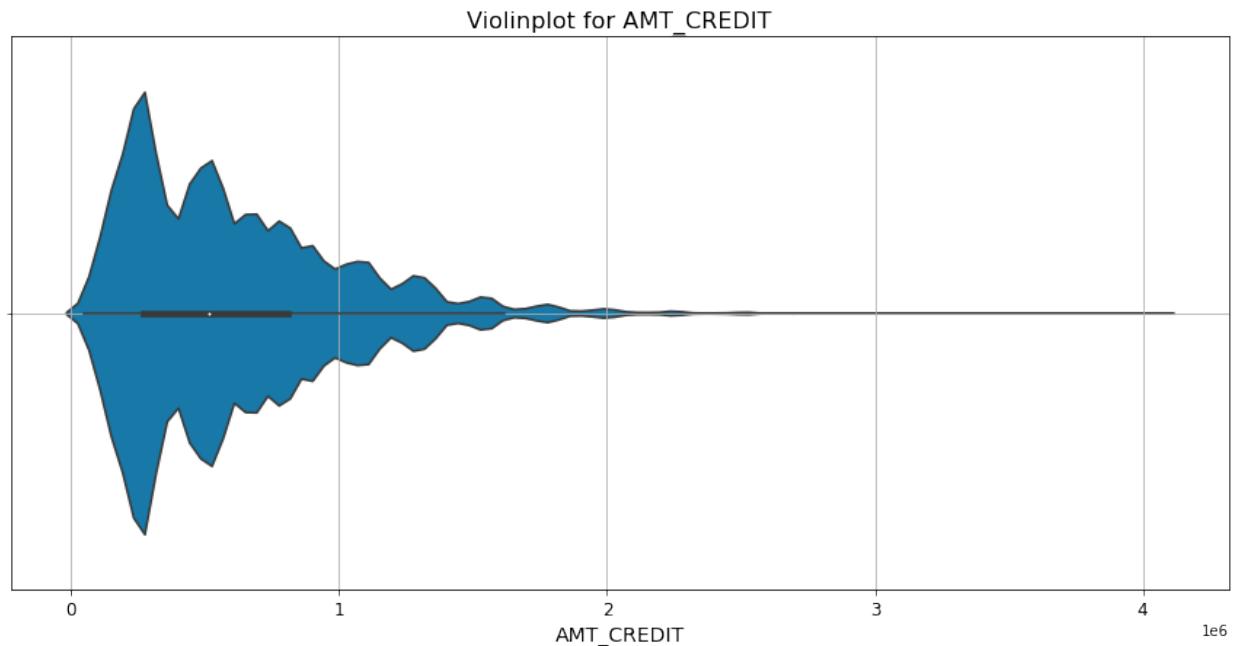


#### 4.18.1 Observation

- We can observe a skew of defaults towards the younger applicants.
- This indicates that older applicants repaid their loans in a more timely/efficient manner.

#### 4.18.2 Checking the distribution of AMT\_CREDIT feature

```
In [66]: plt.figure(figsize=(15, 7))
sns.violinplot(x=app_train['AMT_CREDIT'], palette='winter')
plt.xticks(size=12)
plt.yticks(size=12)
plt.xlabel('AMT_CREDIT', size=14)
plt.title('Violinplot for AMT_CREDIT', size=16)
plt.grid(b=True)
```

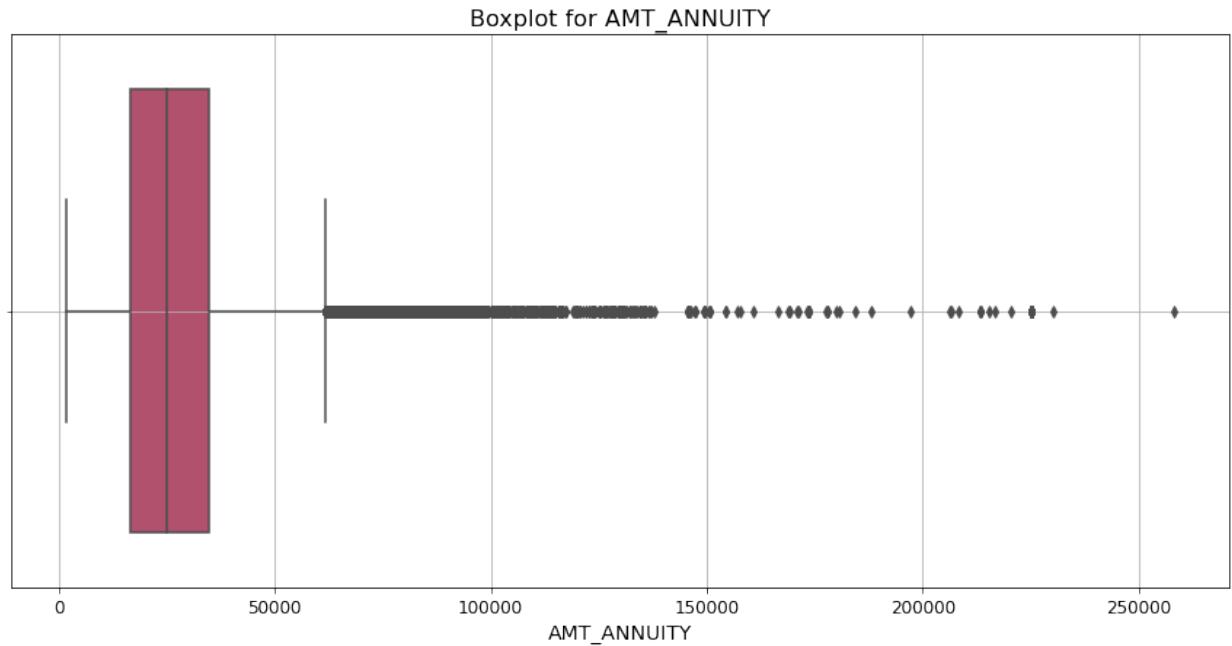


### 4.18.3 Observation

- We can observe that the feature is right skewed.
- Scaling might help us use this feature appropriately

### 4.18.4 Checking the distribution of AMT\_ANNUITY feature

```
In [67]: plt.figure(figsize=(15, 7))
sns.boxplot(x=app_train['AMT_ANNUITY'], palette='flare')
plt.xticks(size=12)
plt.yticks(size=12)
plt.xlabel('AMT_ANNUITY', size=14)
plt.title('Boxplot for AMT_ANNUITY', size=16)
plt.grid(b=True)
```

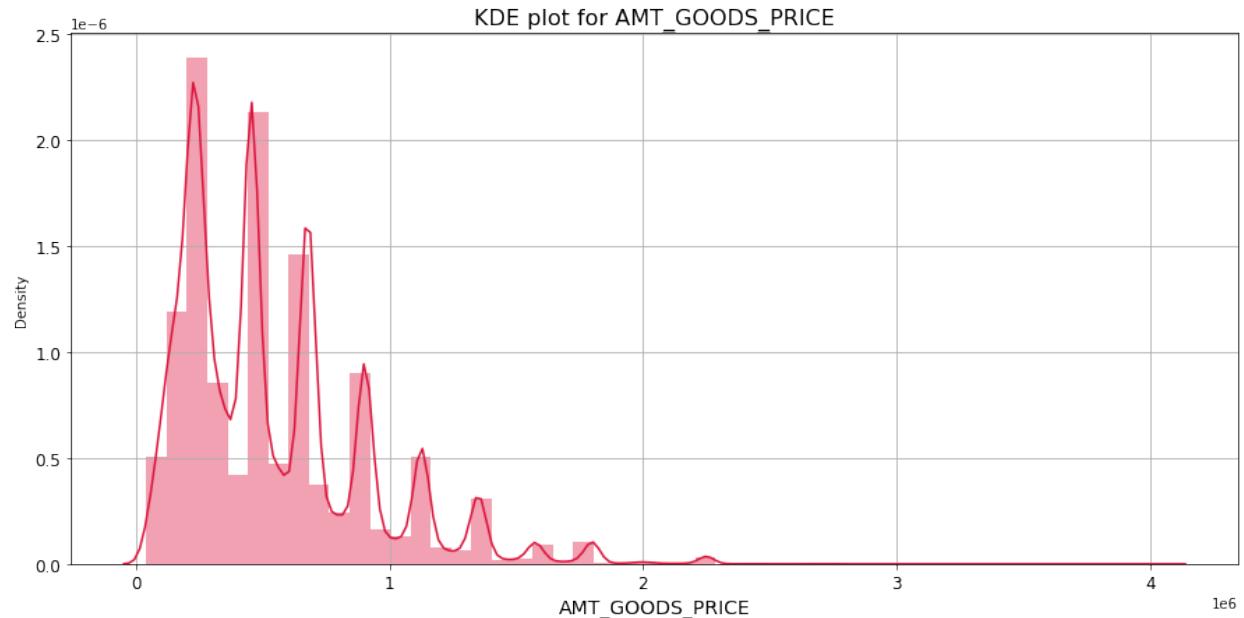


#### 4.18.5 Observation

- We observe yet again a right skewed feature with a lot of outliers.
- We can't remove these outliers since we might lose important information.

#### 4.18.6 Checking the distribution of AMT\_GOODS\_PRICE feature

```
In [68]: plt.figure(figsize=(15, 7))
sns.distplot(x=app_train['AMT_GOODS_PRICE'], color='crimson')
plt.xticks(size=12)
plt.yticks(size=12)
plt.xlabel('AMT_GOODS_PRICE', size=14)
plt.title('KDE plot for AMT_GOODS_PRICE', size=16)
plt.grid(b=True)
```



#### 4.18.7 Observation

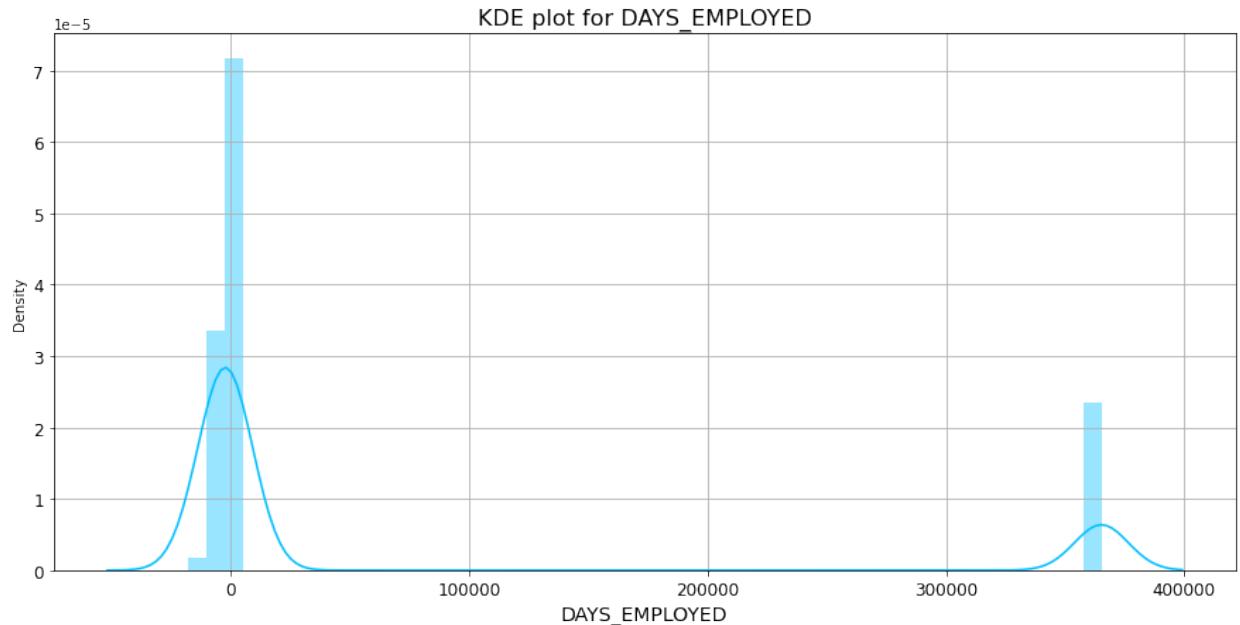
- We see yet another skewed distribution which is multi-modal in nature.
- Binning might help to make an efficient use of this feature.

#### 4.18.8 Checking the distribution of DAYS\_EMPLOYED feature

```
In [69]: app_train['DAYS_EMPLOYED'].describe()
```

```
Out[69]: count    307511.000000
mean      63815.045904
std       141275.766519
min     -17912.000000
25%     -2760.000000
50%     -1213.000000
75%     -289.000000
max      365243.000000
Name: DAYS_EMPLOYED, dtype: float64
```

```
In [70]: plt.figure(figsize=(15, 7))
sns.distplot(x=app_train['DAYS_EMPLOYED'], color='deepskyblue')
plt.xticks(size=12)
plt.yticks(size=12)
plt.xlabel('DAYS_EMPLOYED', size=14)
plt.title('KDE plot for DAYS_EMPLOYED', size=16)
plt.grid(b=True)
```

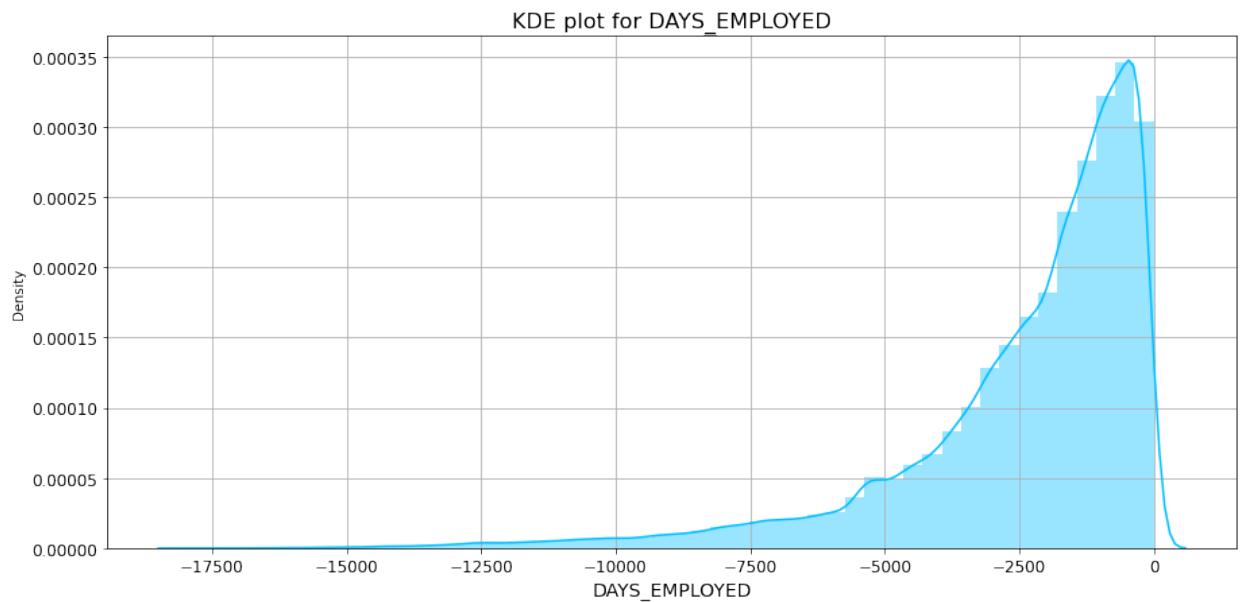


#### 4.18.9 Observation

- Just like DAYS\_BIRTH, this feature has negative days values.
- But we observe a weird anomaly here - max days employed is 365243 days which is a thousand years.
- We will simply ignore this anomaly (replace with appropriate values) and check the distribution of the feature again.

#### 4.18.10 Checking the distribution of DAYS\_EMPLOYED feature after removing the inconsistent value

```
In [71]: days_employed = app_train['DAYS_EMPLOYED']
days_employed = days_employed[days_employed<365243]
plt.figure(figsize=(15, 7))
sns.distplot(x=days_employed, color='deepskyblue')
plt.xticks(size=12)
plt.yticks(size=12)
plt.xlabel('DAYS_EMPLOYED', size=14)
plt.title('KDE plot for DAYS_EMPLOYED', size=16)
plt.grid(b=True)
```



#### 4.18.11 Observation

- We observe a left skewed data in this plot which in turn would form a right skewed distribution if we flip the days to the positive side.

#### 4.18.12 Fixing DAYS\_EMPLOYES and DAYS\_BIRTH features

```
In [21]: app_train['DAYS_BIRTH'] = app_train['DAYS_BIRTH'] / -1
app_test['DAYS_BIRTH'] = app_test['DAYS_BIRTH'] / -1

app_train['DAYS_EMPLOYED'] = app_train['DAYS_EMPLOYED'][app_train['DAY
app_test['DAYS_EMPLOYED'] = app_test['DAYS_EMPLOYED'][app_test['DAYS_E
app_train['DAYS_EMPLOYED'] = app_train['DAYS_EMPLOYED']/-1
app_test['DAYS_EMPLOYED'] = app_test['DAYS_EMPLOYED']/-1

app_train['DAYS_BIRTH'].head()
```

```
Out[21]: 0    -9461.0
1    -16765.0
2    -19046.0
3    -19005.0
4    -19932.0
Name: DAYS_BIRTH, dtype: float64
```

```
In [22]: app_test['DAYS_BIRTH'].head()
```

```
Out[22]: 0    -19241.0
1    -18064.0
2    -20038.0
3    -13976.0
4    -13040.0
Name: DAYS_BIRTH, dtype: float64
```

```
In [23]: app_train['DAYS_EMPLOYED'].head()
```

```
Out[23]: 0    -637.0
1    -1188.0
2    -225.0
3    -3039.0
4    -3038.0
Name: DAYS_EMPLOYED, dtype: float64
```

```
In [24]: app_test['DAYS_EMPLOYED'].head()
```

```
Out[24]: 0    -2329.0
1    -4469.0
2    -4458.0
3    -1866.0
4    -2191.0
Name: DAYS_EMPLOYED, dtype: float64
```

## 5 Dataset questions

### 5.1 Unique record for each SK\_ID\_CURR

```
In [25]: list(datasets.keys())
```

```
Out[25]: ['application_train',
           'application_test',
           'bureau',
           'bureau_balance',
           'credit_card_balance',
           'installments_payments',
           'previous_application',
           'POS_CASH_balance']
```

```
In [26]: len(datasets["application_train"]["SK_ID_CURR"].unique()) == datasets[
```

```
Out[26]: True
```

```
In [27]: # is there an overlap between the test and train customers
          np.intersect1d(datasets["application_train"]["SK_ID_CURR"], datasets["
```

```
Out[27]: array([], dtype=int64)
```

```
In [28]: datasets["application_test"].shape
```

```
Out[28]: (48744, 121)
```

```
In [29]: datasets["application_train"].shape
```

```
Out[29]: (307511, 122)
```

### 5.2 previous applications for the submission file

The persons in the kaggle submission file have had previous applications in the `previous_application.csv`. 47,800 out 48,744 people have had previous applications.

```
In [4]: appsDF = pd.read_csv('/root/shared/AML/I526_AML_Student/Assignments/Ur  
display(appsDF.head())  
print(f'{appsDF.shape[0]}:{} rows, {appsDF.shape[1]}:{} columns')
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	A
0	2030495	271877	Consumer loans	1730.430	17145.0	
1	2802425	108129	Cash loans	25188.615	607500.0	
2	2523466	122040	Cash loans	15060.735	112500.0	
3	2819243	176158	Cash loans	47041.335	450000.0	
4	1784265	202054	Cash loans	31924.395	337500.0	

5 rows × 37 columns

1,670,214 rows, 37 columns

```
In [4]: print(f'There are {appsDF.shape[0]}:{} previous applications')
```

There are 1,670,214 previous applications

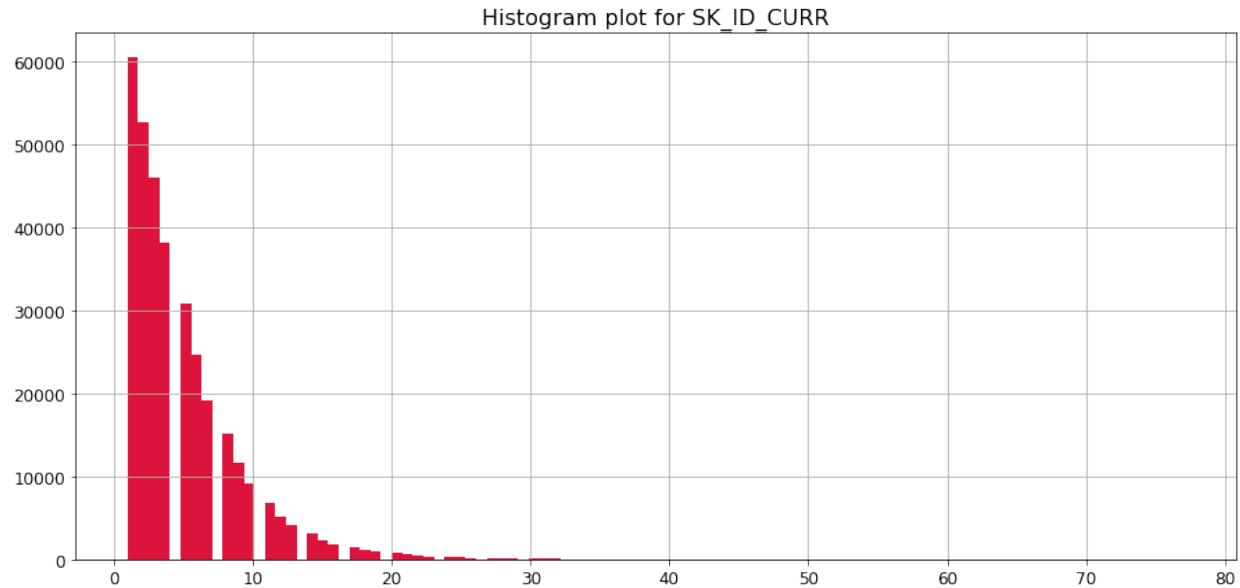
```
In [12]: #Find the intersection of two arrays.  
print(f'Number of train applicants with previous applications is {len(  
    )}
```

Number of train applicants with previous applications is 291,057

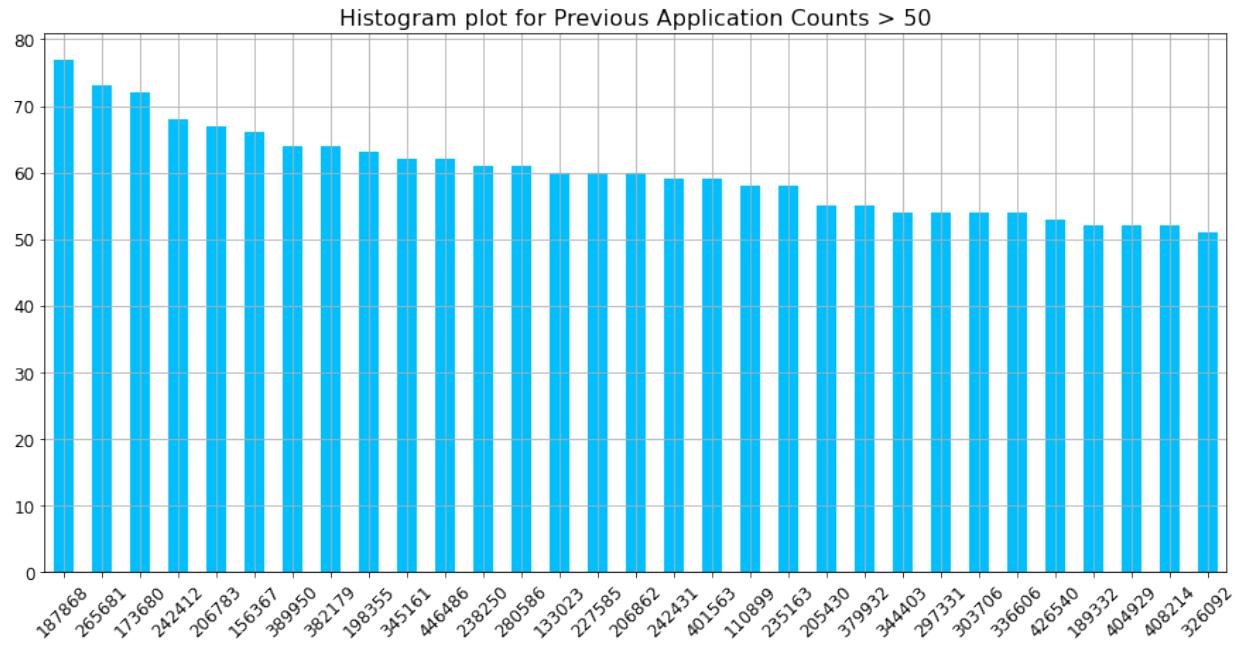
```
In [13]: #Find the intersection of two arrays.  
print(f'Number of test applicants with previous applications is {len(  
    )}
```

Number of test applicants with previous applications is 47,800

```
In [14]: # How many previous appliciations per applicant in the previous_applications
plt.figure(figsize=(15,7))
prevAppCounts = appsDF['SK_ID_CURR'].value_counts(dropna=False)
len(prevAppCounts[prevAppCounts >40]) #more than 40 previous applications
plt.hist(prevAppCounts[prevAppCounts>=0], bins=100, color='crimson')
plt.xticks(size=12)
plt.yticks(size=12)
plt.xlabel('', size=14)
plt.ylabel('', size=14)
plt.title('Histogram plot for SK_ID_CURR', size=16)
plt.grid(b=True)
```



```
In [15]: plt.figure(figsize=(15,7))
prevAppCounts[prevAppCounts > 50].plot(kind='bar', color='deepskyblue')
plt.xticks(size=12, rotation=45)
plt.yticks(size=12)
plt.xlabel('', size=14)
plt.ylabel('', size=14)
plt.title('Histogram plot for Previous Application Counts > 50', size=14)
plt.grid(b=True)
```

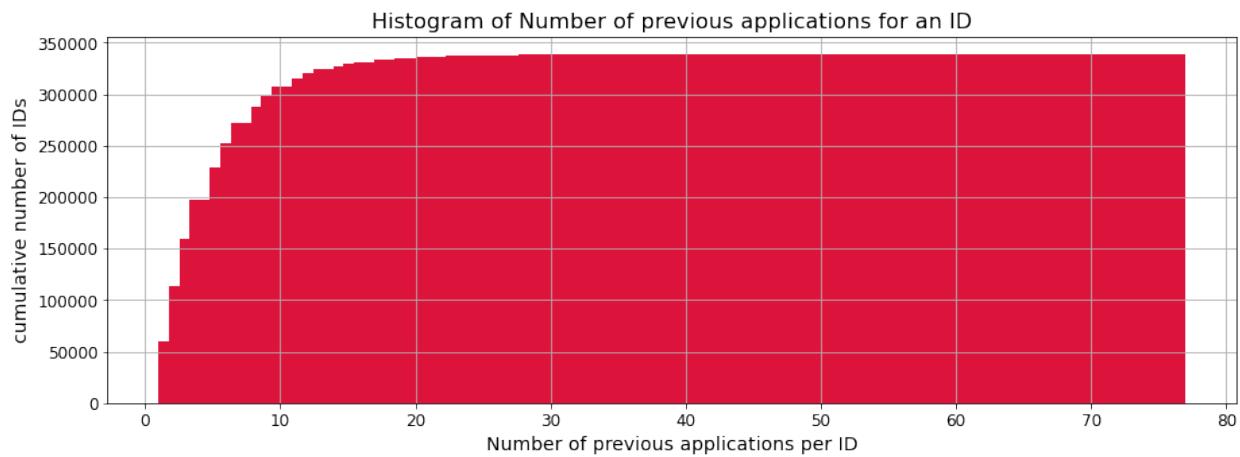


### 5.2.1 Histogram of Number of previous applications for an ID

```
In [16]: sum(appsDF['SK_ID_CURR'].value_counts()==1)
```

Out[16]: 60458

```
In [17]: plt.figure(figsize=(15,5))
plt.hist(appsDF['SK_ID_CURR'].value_counts(), cumulative =True, bins =
plt.xticks(size=12)
plt.yticks(size=12)
plt.ylabel('cumulative number of IDs', size=14)
plt.xlabel('Number of previous applications per ID', size=14)
plt.title('Histogram of Number of previous applications for an ID', si
plt.grid()
plt.show()
```



### **Can we differentiate applications by low, medium and high previous apps?**

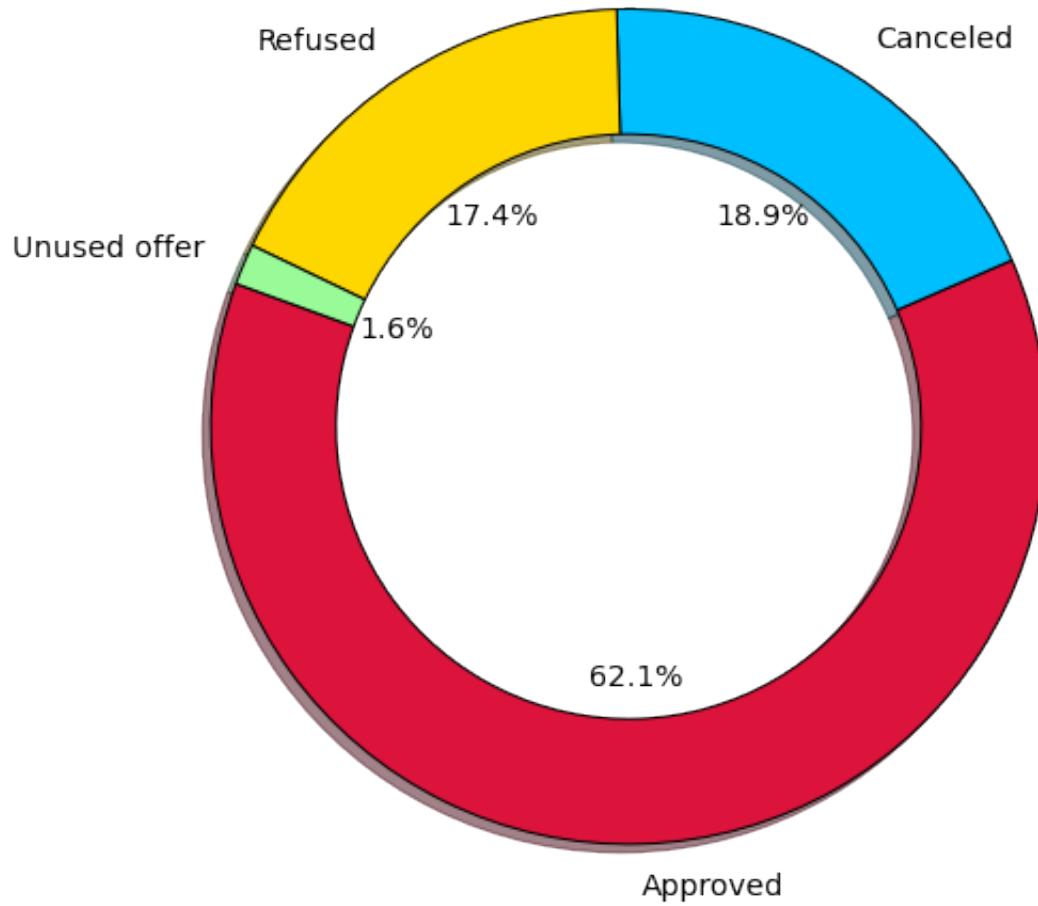
- \* Low = <5 claims (22%)
- \* Medium = 10 to 39 claims (58%)
- \* High = 40 or more claims (20%)

```
In [38]: apps_all = appsDF['SK_ID_CURR'].nunique()
apps_5plus = appsDF['SK_ID_CURR'].value_counts()>=5
apps_40plus = appsDF['SK_ID_CURR'].value_counts()>=40
print('Percentage with 10 or more previous apps:', np.round(100.*sum(
print('Percentage with 40 or more previous apps:', np.round(100.*sum(
```

Percentage with 10 or more previous apps: 41.76895  
 Percentage with 40 or more previous apps: 0.03453

```
In [39]: plt.figure(figsize=(9, 9))
plt.pie(x=appsDF['NAME_CONTRACT_STATUS'].value_counts(),
        radius=1.3-0.3,
        labels=appsDF['NAME_CONTRACT_STATUS'].value_counts().index,
        autopct='%.1f%%',
        colors=['crimson', 'deepskyblue', 'gold', 'palegreen'],
        wedgeprops={"edgecolor":"0", "width":0.3},
        startangle=160,
        shadow=True,
        textprops={'fontsize': 14})
plt.ylabel('', fontsize=14)
plt.title("Applicant's Previous Contract Status", fontsize=16)
plt.show()
```

Applicant's Previous Contract Status

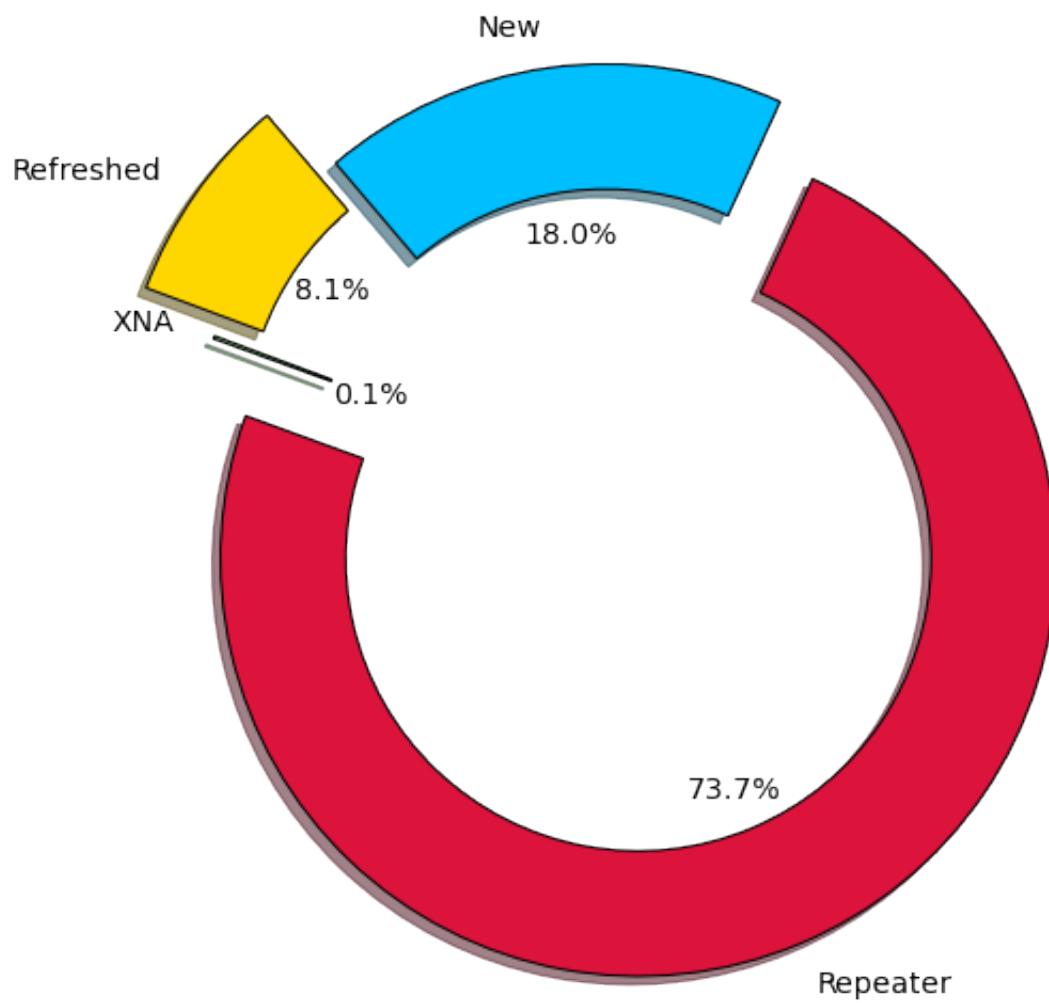


### 5.2.2 Observation

- In previous applications, most of the applicants had their contracts approved.
- 36% of applicants had their contracts either rejected or cancelled and the rest 1.6% didn't use their contracts at all.

```
In [ ]: plt.figure(figsize=(9, 9))
plt.pie(x=appsDF['NAME_CLIENT_TYPE'].value_counts(),
        radius=1.3-0.3,
        labels=appsDF['NAME_CLIENT_TYPE'].value_counts().index,
        autopct='%1.1f%%',
        colors=['crimson', 'deepskyblue', 'gold', 'palegreen'],
        explode=[0.2, 0, 0.2, 0],
        wedgeprops={"edgecolor":"0", "width":0.3},
        startangle=160,
        shadow=True,
        textprops={'fontsize': 14})
plt.ylabel('', fontsize=14)
plt.title("Type of Client in Previous Application", fontsize=16)
plt.show()
```

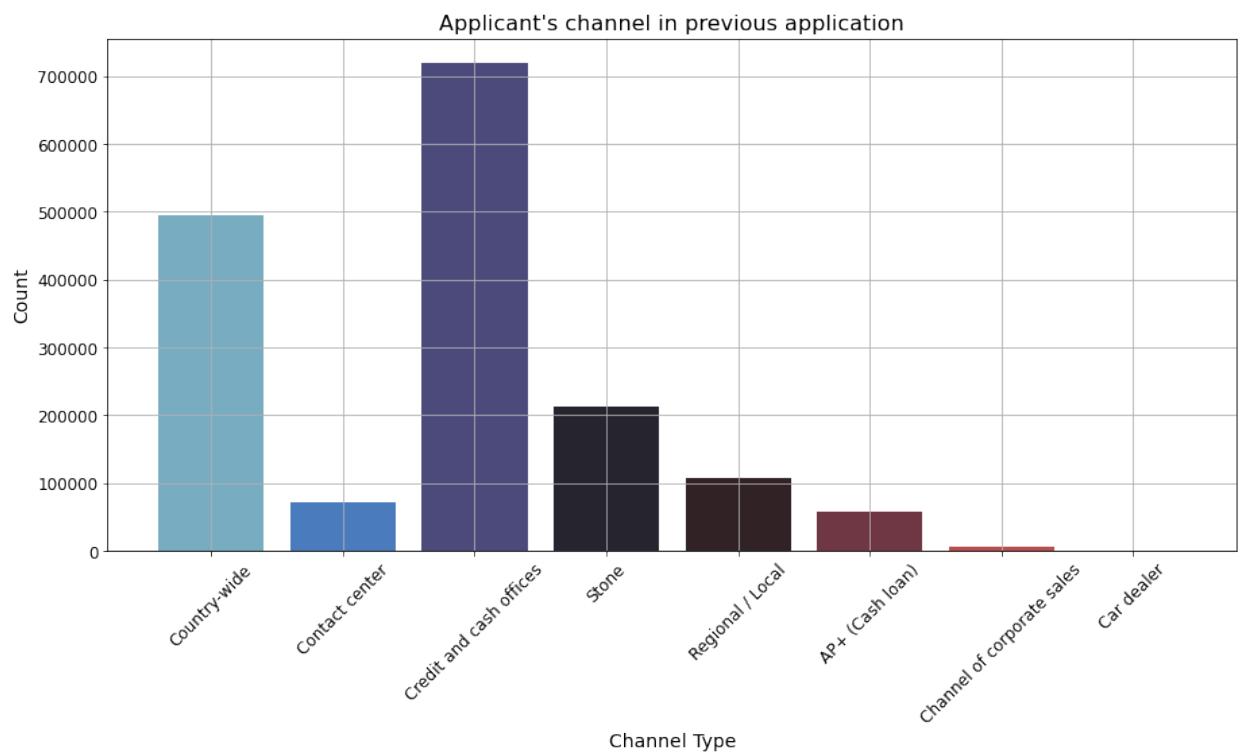
Type of Client in Previous Application



### 5.2.3 Observation

- Most of the applicants are repeaters, followed by new applicants and refreshed applicants.

```
In [ ]: plt.figure(figsize=(15, 7))
sns.countplot(x='CHANNEL_TYPE', data=appsDF, palette='icefire')
plt.title("Applicant's channel in previous application", fontsize=16)
plt.xlabel('Channel Type', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(fontsize=12, rotation=45)
plt.yticks(fontsize=12)
plt.grid(b=True)
plt.plot();
```



### 5.2.4 Observation

- Previous applicants primarily came through credit and cash offices and least via car dealers.

## 5.3 Feature Engineering

### 5.3.1 Performing Encoding on the Categorical Features of application\_train and application\_test

```
In [40]: # Label Encoding
# Create a label encoder object
le = LabelEncoder()
le_count = 0

# Iterate through the columns
for col in app_train:
    if app_train[col].dtype == 'object':
        # If 2 or fewer unique categories
        if len(list(app_train[col].unique())) <= 2:
            # Train on the training data
            le.fit(app_train[col])
            # Transform both training and testing data
            app_train[col] = le.transform(app_train[col])
            app_test[col] = le.transform(app_test[col])

        # Keep track of how many columns were label encoded
        le_count += 1

print('%d columns were label encoded.' % le_count)
```

0 columns were label encoded.

```
In [43]: # one-hot encoding of features
app_train = pd.get_dummies(app_train)
app_test = pd.get_dummies(app_test)

print('Training Features shape: ', app_train.shape)
print('Testing Features shape: ', app_test.shape)
```

Training Features shape: (307511, 240)  
Testing Features shape: (48744, 239)

```
In [44]: train_labels = app_train['TARGET']

# Align the training and testing data, keep only columns present in both
app_train, app_test = app_train.align(app_test, join = 'inner', axis = 1)

# Add the target back in
app_train['TARGET'] = train_labels

print('Training Features shape: ', app_train.shape)
print('Testing Features shape: ', app_test.shape)
```

```
Training Features shape: (307511, 240)
Testing Features shape: (48744, 239)
```

## 5.4 Saving the cleaned train and test file for easy future access

```
In [91]: # app_train.to_csv('app_train.csv', index=False)
```

```
In [92]: # app_test.to_csv('app_test.csv', index=False)
```

# 6 Joining secondary tables with the primary table

In the case of the HCDR competition (and many other machine learning problems that involve multiple tables in 3NF or not) we need to join these datasets (denormalize) when using a machine learning pipeline. Joining the secondary tables with the primary table will lead to lots of new features about each loan application; these features will tend to be aggregate type features or meta data about the loan or its application. How can we do this when using Machine Learning Pipelines?

## 6.1 Joining previous\_application with application\_x

We refer to the `application_train` data (and also `application_test` data also) as the **primary table** and the other files as the **secondary tables** (e.g., `previous_application` dataset). All tables can be joined using the primary key `SK_ID_PREV`.

Let's assume we wish to generate a feature based on previous application attempts. In this case, possible features here could be:

- A simple feature could be the number of previous applications.
- Other summary features of original features such as `AMT_APPLICATION` , `AMT_CREDIT` could be based on average, min, max, median, etc.

To build such features, we need to join the `application_train` data (and also `application_test` data also) with the 'previous\_application' dataset (and the other available datasets).

When joining this data in the context of pipelines, different strategies come to mind with various tradeoffs:

1. Preprocess each of the non-application data sets, thereby generating many new (derived) features, and then joining (aka merge) the results with the `application_train` data (the labeled dataset) and with the `application_test` data (the unlabeled submission dataset) prior to processing the data (in a train, valid, test partition) via your machine learning pipeline. [This approach is recommended for this HCDR competition. WHY?]
- Do the joins as part of the transformation steps. [Not recommended here. WHY?]. How can this be done? Will it work?
  - This would be necessary if we had dataset wide features such as IDF (inverse document frequency) which depend on the entire subset of data as opposed to a single loan application (e.g., a feature about the relative amount applied for such as the percentile of the loan amount being applied for).

I want you to think about this section and build on this.

## 6.2 Roadmap for secondary table processing

1. Transform all the secondary tables to features that can be joined into the main table the `application` table (labeled and unlabeled)
  - 'bureau', 'bureau\_balance', 'credit\_card\_balance', 'installments\_payments',
  - 'previous\_application', 'POS\_CASH\_balance'
- Merge the transformed secondary tables with the primary tables (i.e., the `application_train` data (the labeled dataset) and with the `application_test` data (the unlabeled submission dataset)), thereby leading to `X_train`, `y_train`, `X_valid`, etc.
- Proceed with the learning pipeline using `X_train`, `y_train`, `X_valid`, etc.
- Generate a submission file using the learnt model

In [45]: !pwd

```
/root/shared/AML/I526_AML_Student/Assignments/Unit-Project-Home-Credit-Default-Risk/Phase2
```

In [5]: app\_train = pd.read\_csv('/root/shared/AML/I526\_AML\_Student/Assignments/app\_train.head()')

Out[5]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN
0	100002		0	0	1
1	100003		0	0	0
2	100004		1	1	1
3	100006		0	0	1
4	100007		0	0	1

5 rows × 240 columns

In [6]: app\_test = pd.read\_csv('/root/shared/AML/I526\_AML\_Student/Assignments/app\_test.head()')

Out[6]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN
0	100001		0	0	1
1	100005		0	0	1
2	100013		0	1	1
3	100028		0	0	1
4	100038		0	1	0

5 rows × 239 columns

In [46]: `appsDF.columns`

Out[46]: `Index(['SK_ID_PREV', 'SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'AMT_ANNUITY', 'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AMT_GOODS_PRICE', 'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START', 'FLAG_LAST_APPL_PER_CONTRACT', 'NFLAG_LAST_APPL_IN_DAY', 'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY', 'RATE_INTEREST_PRIVILEGED', 'NAME_CASH_LOAN_PURPOSE', 'NAME_CONTRACT_STATUS', 'DAYS_DECISION', 'NAME_PAYMENT_TYPE', 'CODE_REJECT_REASON', 'NAME_TYPE_SUITE', 'NAME_CLIENT_TYPE', 'NAME_GOODS_CATEGORY', 'NAME_PORTFOLIO', 'NAME_PRODUCT_TYPE', 'CHANNEL_TYPE', 'SELLERPLACE_AREA', 'NAME_SELLER_INDUSTRY', 'CNT_PAYMENT', 'NAME_YIELD_GROUP', 'PRODUCT_COMBINATION', 'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION', 'DAYS_LAST_DUE', 'DAYS_TERMINATION', 'NFLAG_INSURED_ON_APPROVAL'], dtype='object')`

In [47]: `appsDF[0:50][(appsDF["SK_ID_CURR"]==175704)]`

Out[47]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	A
6	2315218	175704	Cash loans	NaN	0.0	

In [48]: `appsDF[0:50][(appsDF["SK_ID_CURR"]==175704)]["AMT_CREDIT"]`

Out[48]: `6 0.0  
Name: AMT_CREDIT, dtype: float64`

In [49]: `appsDF[0:50][(appsDF["SK_ID_CURR"]==175704) & ~(appsDF["AMT_CREDIT"]==0)]`

Out[49]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	A
6	2315218	175704	Cash loans	NaN	0.0	

## 6.3 Missing values in prevApps

In [7]: `appsDF.isna().sum()`

Out[7]:

SK_ID_PREV	0
SK_ID_CURR	0
NAME_CONTRACT_TYPE	0
AMT_ANNUITY	372235
AMT_APPLICATION	0
AMT_CREDIT	1
AMT_DOWN_PAYMENT	895844
AMT_GOODS_PRICE	385515
WEEKDAY_APPR_PROCESS_START	0
HOUR_APPR_PROCESS_START	0
FLAG_LAST_APPL_PER_CONTRACT	0
NFLAG_LAST_APPL_IN_DAY	0
RATE_DOWN_PAYMENT	895844
RATE_INTEREST_PRIMARY	1664263
RATE_INTEREST_PRIVILEGED	1664263
NAME_CASH_LOAN_PURPOSE	0
NAME_CONTRACT_STATUS	0
DAYS_DECISION	0
NAME_PAYMENT_TYPE	0
CODE_REJECT_REASON	0
NAME_TYPE_SUITE	820405
NAME_CLIENT_TYPE	0
NAME_GOODS_CATEGORY	0
NAME_PORTFOLIO	0
NAME_PRODUCT_TYPE	0
CHANNEL_TYPE	0
SELLERPLACE_AREA	0
NAME_SELLER_INDUSTRY	0
CNT_PAYMENT	372230
NAME_YIELD_GROUP	0
PRODUCT_COMBINATION	346
DAYS_FIRST_DRAWING	673065
DAYS_FIRST_DUE	673065
DAYS_LAST_DUE_1ST_VERSION	673065
DAYS_LAST_DUE	673065
DAYS_TERMINATION	673065
NFLAG_INSURED_ON_APPROVAL	673065

dtype: int64

In [51]: `appsDF.columns`

```
Out[51]: Index(['SK_ID_PREV', 'SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'AMT_ANNUITY',
       'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AMT_GOODS_PRICE',
       'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START',
       'FLAG_LAST_APPL_PER_CONTRACT', 'NFLAG_LAST_APPL_IN_DAY',
       'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY',
       'RATE_INTEREST_PRIVILEGED', 'NAME_CASH_LOAN_PURPOSE',
       'NAME_CONTRACT_STATUS', 'DAYS_DECISION', 'NAME_PAYMENT_TYPE',
       'CODE_REJECT_REASON', 'NAME_TYPE_SUITE', 'NAME_CLIENT_TYPE',
       'NAME_GOODS_CATEGORY', 'NAME_PORTFOLIO', 'NAME_PRODUCT_TYPE',
       'CHANNEL_TYPE', 'SELLERPLACE_AREA', 'NAME_SELLER_INDUSTRY',
       'CNT_PAYMENT', 'NAME_YIELD_GROUP', 'PRODUCT_COMBINATION',
       'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION',
       'DAYS_LAST_DUE', 'DAYS_TERMINATION', 'NFLAG_INSURED_ON_APPROVAL'],
      dtype='object')
```

## 6.4 feature engineering for prevApp table

```
In [8]: features = ['AMT_ANNUITY', 'AMT_APPLICATION']
print(f"{appsDF[features].describe()}")
agg_ops = ["min", "max", "mean"]
result = appsDF.groupby(["SK_ID_CURR"], as_index=False).agg("mean") #groupby
display(result.head())
print("-"*50)
result = appsDF.groupby(["SK_ID_CURR"], as_index=False).agg({'AMT_ANNUITY': 'range_AMT_APPLICATION' : result['AMT_APPLICATION_max'] - result['AMT_APPLICATION_min']})
result.columns = result.columns.map('_'.join)
display(result)
result['range_AMT_APPLICATION'] = result['AMT_APPLICATION_max'] - result['AMT_APPLICATION_min']
print(f"result.shape: {result.shape}")
result[0:10]
```

	AMT_ANNUITY	AMT_APPLICATION
count	1.297979e+06	1.670214e+06
mean	1.595512e+04	1.752339e+05
std	1.478214e+04	2.927798e+05
min	0.000000e+00	0.000000e+00
25%	6.321780e+03	1.872000e+04
50%	1.125000e+04	7.104600e+04
75%	2.065842e+04	1.803600e+05
max	4.180581e+05	6.905160e+06

`SK_ID_CURR SK_ID_PREV AMT_ANNUITY AMT_APPLICATION AMT_CREDIT AMT_DOWN_PAYMENT`

0	100001	1.369693e+06	3951.000	24835.50	23787.00
1	100002	1.038818e+06	9251.775	179055.00	179055.00
2	100003	2.281150e+06	56553.990	435436.50	484191.00
3	100004	1.564014e+06	5357.250	24282.00	20106.00
4	100005	2.176837e+06	4813.200	22308.75	20076.75

5 rows × 21 columns

	SK_ID_CURR_	AMT_ANNUITY_min	AMT_ANNUITY_max	AMT_ANNUITY_mean	AMT_AF
0	100001	3951.000	3951.000	3951.000000	
1	100002	9251.775	9251.775	9251.775000	
2	100003	6737.310	98356.995	56553.990000	
3	100004	5357.250	5357.250	5357.250000	
4	100005	4813.200	4813.200	4813.200000	
...	...	...	...	...	...
338852	456251	6605.910	6605.910	6605.910000	
338853	456252	10074.465	10074.465	10074.465000	
338854	456253	3973.095	5567.715	4770.405000	
338855	456254	2296.440	19065.825	10681.132500	
338856	456255	2250.000	54022.140	20775.391875	

338857 rows × 7 columns

result.shape: (338857, 8)

Out[8]:

	SK_ID_CURR_	AMT_ANNUITY_min	AMT_ANNUITY_max	AMT_ANNUITY_mean	AMT_APPLICABILITY
0	100001	3951.000	3951.000	3951.000000	
1	100002	9251.775	9251.775	9251.775000	
2	100003	6737.310	98356.995	56553.990000	
3	100004	5357.250	5357.250	5357.250000	
4	100005	4813.200	4813.200	4813.200000	
5	100006	2482.920	39954.510	23651.175000	
6	100007	1834.290	22678.785	12278.805000	

7	100008	8019.090	25309.575	15839.696250
8	100009	7435.845	17341.605	10051.412143
9	100010	27463.410	27463.410	27463.410000

```
In [19]: result.isna().sum()
```

```
Out[19]: SK_ID_CURR_          0  
AMT_ANNUITY_min      480  
AMT_ANNUITY_max      480  
AMT_ANNUITY_mean     480  
AMT_APPLICATION_min   0  
AMT_APPLICATION_max   0  
AMT_APPLICATION_mean  0  
range_AMT_APPLICATION 0  
dtype: int64
```

## 6.5 feature transformer for prevApp table

```
In [98]: # class prevAppsFeaturesAggregater(BaseEstimator, TransformerMixin):
#     def __init__(self, features=None): # no *args or **kargs
#         self.features = features
#         self.agg_op_features = {}
#         for f in features:
#             self.agg_op_features[f] = {f"{{f}}_{{func}}":func for func in ["min", "max", "mean"]}

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

#     def transform(self, X, y=None):
#         #from IPython.core.debugger import Pdb;    pdb().set_trace()
#         result = X.groupby(["SK_ID_CURR"]).agg(self.agg_op_features)
#         result.columns = result.columns.droplevel()
#         result.columns = ["_".join(x) for x in result.columns.ravel()]

#         result = result.reset_index(level=["SK_ID_CURR"])
#         result['range_AMT_APPLICATION'] = result['AMT_APPLICATION'].map(lambda x: x if x > 0 else 0)
#         return result # return dataframe with the join key "SK_ID_CURR"

# from sklearn.pipeline import make_pipeline
# def test_driver_prevAppsFeaturesAggregater(df, features):
#     print(f"df.shape: {df.shape}\n")
#     print(f"df[{features}][0:5]: \n{df[features][0:5]}")
#     test_pipeline = make_pipeline(prevAppsFeaturesAggregater(features))
#     return(test_pipeline.fit_transform(df))

# features = ['AMT_ANNUITY', 'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PAYMENT',
#             'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY',
#             'RATE_INTEREST_PRIVILEGED', 'DAYS_DECISION', 'NAME_PAYMENT_TYPE',
#             'CNT_PAYMENT', 'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE',
#             'DAYS_LAST_DUE', 'DAYS_TERMINATION']

# res = test_driver_prevAppsFeaturesAggregater(appsDF, features)
# print(f"Hello")
# print(f"Test driver: \n{res[0:10]}")
# print(f"input[features][0:10]: \n{appsDF[0:10]}")
```

```
In [9]: from sklearn.preprocessing import PolynomialFeatures
from sklearn.impute import SimpleImputer

# Make a new dataframe for polynomial features
poly_features = app_train[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3',
                           'EXT_SOURCE_4', 'EXT_SOURCE_5', 'EXT_SOURCE_6',
                           'EXT_SOURCE_7']]
poly_features_test = app_test[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3',
                               'EXT_SOURCE_4', 'EXT_SOURCE_5', 'EXT_SOURCE_6',
                               'EXT_SOURCE_7']]

# imputer for handling missing values
# from sklearn.preprocessing import Imputer
imputer = SimpleImputer(strategy = 'median')

poly_target = poly_features['TARGET']

poly_features = poly_features.drop(columns = ['TARGET'])

# Need to impute missing values
poly_features = imputer.fit_transform(poly_features)
poly_features_test = imputer.transform(poly_features_test)

# Create the polynomial object with specified degree
poly_transformer = PolynomialFeatures(degree = 3)
```

```
In [10]: # Train the polynomial features
poly_transformer.fit(poly_features)

# Transform the features
poly_features = poly_transformer.transform(poly_features)
poly_features_test = poly_transformer.transform(poly_features_test)
print('Polynomial Features shape: ', poly_features.shape)
```

Polynomial Features shape: (307511, 35)

```
In [11]: poly_transformer.get_feature_names(input_features = ['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'EXT_SOURCE_1 EXT_SOURCE_2', 'EXT_SOURCE_1 EXT_SOURCE_3', 'EXT_SOURCE_1 DAYS_BIRTH', 'EXT_SOURCE_2 EXT_SOURCE_3', 'EXT_SOURCE_2 DAYS_BIRTH', 'EXT_SOURCE_3 EXT_SOURCE_1', 'EXT_SOURCE_3 EXT_SOURCE_2', 'EXT_SOURCE_3 DAYS_BIRTH', 'EXT_SOURCE_1^2', 'EXT_SOURCE_2^2', 'EXT_SOURCE_3^2', 'EXT_SOURCE_1^2 EXT_SOURCE_2', 'EXT_SOURCE_1^2 EXT_SOURCE_3', 'EXT_SOURCE_1^2 DAYS_BIRTH', 'EXT_SOURCE_1 EXT_SOURCE_2^2'])
```

```
Out[11]: ['1',  
          'EXT_SOURCE_1',  
          'EXT_SOURCE_2',  
          'EXT_SOURCE_3',  
          'EXT_SOURCE_1',  
          'EXT_SOURCE_2',  
          'EXT_SOURCE_3',  
          'EXT_SOURCE_1 EXT_SOURCE_2',  
          'EXT_SOURCE_1 EXT_SOURCE_3',  
          'EXT_SOURCE_1 DAYS_BIRTH',  
          'EXT_SOURCE_2 EXT_SOURCE_3',  
          'EXT_SOURCE_2 DAYS_BIRTH',  
          'EXT_SOURCE_3 EXT_SOURCE_1',  
          'EXT_SOURCE_3 EXT_SOURCE_2',  
          'EXT_SOURCE_3 DAYS_BIRTH',  
          'EXT_SOURCE_1^2',  
          'EXT_SOURCE_2^2',  
          'EXT_SOURCE_3^2',  
          'EXT_SOURCE_1^2 EXT_SOURCE_2',  
          'EXT_SOURCE_1^2 EXT_SOURCE_3',  
          'EXT_SOURCE_1^2 DAYS_BIRTH',  
          'EXT_SOURCE_1 EXT_SOURCE_2^2']
```

```
In [12]: # Create a dataframe of the features
poly_features = pd.DataFrame(poly_features,
                               columns = poly_transformer.get_feature_na

# Add in the target
poly_features['TARGET'] = poly_target

# Find the correlations with the target
poly_corrs = poly_features.corr()['TARGET'].sort_values()

# Display most negative and most positive
print(poly_corrs.head(10))
print(poly_corrs.tail(5))
```

```
EXT_SOURCE_2 EXT_SOURCE_3      -0.193939
EXT_SOURCE_1 EXT_SOURCE_2 EXT_SOURCE_3      -0.189605
EXT_SOURCE_2 EXT_SOURCE_3 DAYS_BIRTH      -0.181283
EXT_SOURCE_2^2 EXT_SOURCE_3      -0.176428
EXT_SOURCE_2 EXT_SOURCE_3^2      -0.172282
EXT_SOURCE_1 EXT_SOURCE_2      -0.166625
EXT_SOURCE_1 EXT_SOURCE_3      -0.164065
EXT_SOURCE_2      -0.160295
EXT_SOURCE_2 DAYS_BIRTH      -0.156873
EXT_SOURCE_1 EXT_SOURCE_2^2      -0.156867
Name: TARGET, dtype: float64
DAYS_BIRTH      -0.078239
DAYS_BIRTH^2     -0.076672
DAYS_BIRTH^3     -0.074273
TARGET          1.000000
1              NaN
Name: TARGET, dtype: float64
```

```
In [13]: # Put test features into dataframe
poly_features_test = pd.DataFrame(poly_features_test,
                                    columns = poly_transformer.get_featu

# Merge polynomial features into training dataframe
poly_features['SK_ID_CURR'] = app_train['SK_ID_CURR']
app_train_poly = app_train.merge(poly_features, on = 'SK_ID_CURR', how = 'left')

# Merge polynomial features into testing dataframe
poly_features_test['SK_ID_CURR'] = app_test['SK_ID_CURR']
app_test_poly = app_test.merge(poly_features_test, on = 'SK_ID_CURR', how = 'left')

# Align the dataframes
app_train_poly, app_test_poly = app_train_poly.align(app_test_poly, join = 'inner')

# Print out the new shapes
print('Training data with polynomial features shape: ', app_train_poly.shape)
print('Testing data with polynomial features shape: ', app_test_poly.shape)

Training data with polynomial features shape: (307511, 274)
Testing data with polynomial features shape: (48744, 274)
```

```
In [14]: app_train_poly['TARGET'] = app_train['TARGET']
app_train_poly.head()
```

Out[14]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN
0	100002		0	0	1
1	100003		0	0	0
2	100004		1	1	1
3	100006		0	0	1
4	100007		0	0	1

0	100002		0	0	1
1	100003		0	0	0
2	100004		1	1	1
3	100006		0	0	1
4	100007		0	0	1

5 rows × 275 columns

```
In [15]: app_train_poly['TARGET'].head()
```

```
Out[15]: 0    1  
1    0  
2    0  
3    0  
4    0  
Name: TARGET, dtype: int64
```

```
In [16]: app_test_poly.head()
```

```
Out[16]:
```

	SK_ID_CURR	NAME_CONTRACT_TYPE	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN
0	100001		0	0	1
1	100005		0	0	1
2	100013		0	1	1
3	100028		0	0	1
4	100038		0	1	0

5 rows × 274 columns

## 6.6 Feature Aggregating

```
In [ ]:
```

## 6.7 Join the labeled dataset

```
In [51]: datasets.keys()
```

```
Out[51]: dict_keys(['application_train', 'application_test', 'bureau', 'bureau_balance', 'credit_card_balance', 'installments_payments', 'previous_application', 'POS_CASH_balance'])
```

```
In [52]:
```

```

# features = ['AMT_ANNUITY', 'AMT_APPLICATION']
# prevApps_feature_pipeline = Pipeline([
#     ('prevApps_add_features1', prevApps_add_features1()), # add
#     ('prevApps_add_features2', prevApps_add_features2()), # add
#     ('prevApps_aggregator', prevAppsFeaturesAggregator()), # Agg
# ])
# ]]

merged_data = app_train_poly #primary dataset
appsDF = datasets["previous_application"] #prev app

merge_all_data = False

# transform all the secondary tables
# 'bureau', 'bureau_balance', 'credit_card_balance', 'installments_pay'
# 'previous_application', 'POS_CASH_balance'

if merge_all_data:
    prevApps_aggregated = prevApps_feature_pipeline.transform(appsDF)

    #'bureau', 'bureau_balance', 'credit_card_balance', 'installments_
    # 'previous_application', 'POS_CASH_balance'

# merge primary table and secondary tables using features based on met
if merge_all_data:
    # 1. Join/Merge in prevApps Data
    merged_data = merged_data.merge(prevApps_aggregated, how='left', on="SK_ID_CURR")
    # merged_data = merged_data.merge(bureau_aggregated, how='left', on="SK_ID_CURR")
    # merged_data = merged_data.merge(ccblance_aggregated, how='left', on="SK_ID_CURR")
    # merged_data = merged_data.merge(installments_pmnts_aggregated, how='left', on="SK_ID_CURR")

    # 2. Join/Merge in ..... Data
    #X_train = X_train.merge(...._aggregated, how='left', on="SK_ID_CURR")

    # 3. Join/Merge in .....Data
    #dX_train = X_train.merge(...._aggregated, how='left', on="SK_ID_CURR")

    # 4. Join/Merge in Aggregated ..... Data
    #X_train = X_train.merge(...._aggregated, how='left', on="SK_ID_CURR")

    #
    # .....
merged_data.head()

```

Out[52]:

SK_ID_CURR	NAME_CONTRACT_TYPE	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN
------------	--------------------	--------------	-----------------	--------------

0	100002	0	0	1
---	--------	---	---	---

1	100003	0	0	0
2	100004	1	1	1
3	100006	0	0	1
4	100007	0	0	1

5 rows × 275 columns

## 6.8 Join the unlabeled dataset (i.e., the submission file)

```
In [17]: # X_kaggle_test= app_test_poly
# X_kaggle_test = apln_feature_pipeline.fit_transform(X_kaggle_test)

# merge_all_data = True
# if merge_all_data:
#     X_kaggle_test = X_kaggle_test.merge(prevApps_aggregated, how='le
#     # X_kaggle_test = X_kaggle_test.merge(bureau_aggregated, how='le
#     # X_kaggle_test = X_kaggle_test.merge(ccblance_aggregated, how='
#     # X_kaggle_test = X_kaggle_test.merge(installments_pmnts_aggrega
```

```
In [31]: app_train_poly.to_csv('app_train_poly.csv', index=False)
app_test_poly.to_csv('app_test_poly.csv', index=False)
```

## 7 Processing pipeline

```
In [3]: # merged_data = pd.read_csv('merged_data_train.csv')
# merged_data.head()
```

Out[3]:

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

5 rows × 160 columns

```
In [4]: # pd.set_option('display.max_columns', None)
# pd.set_option('display.max_rows', None)
# list(merged_data.columns)
```

```
Out[4]: ['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',
 'NAME_TYPE_SUITE',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE',
 'REGION_POPULATION_RELATIVE',
 'DAYS_BIRTH_x',
 'DAYS_EMPLOYED',
 'DAYS_REGISTRATION',
 'DAYS_ID_PUBLISH']
```

Please [this blog](https://medium.com/hugo-ferreiras-blog/dealing-with-categorical-features-in-machine-learning-1bb70f07262d) (<https://medium.com/hugo-ferreiras-blog/dealing-with-categorical-features-in-machine-learning-1bb70f07262d>) for more details of OHE when the validation/test have previously unseen unique values.

```
In [18]: class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

```
In [22]: # # Split the provided training data into training and validation and
# # The kaggle evaluation test set has no labels

train_dataset=app_train_poly
class_labels = ["No Default", "Default"]

from sklearn.model_selection import train_test_split

num_attribs = [
    'AMT_INCOME_TOTAL',
    'AMT_CREDIT',
    'EXT_SOURCE_3_x',
```

```
'EXT_SOURCE_2_x',
'EXT_SOURCE_1_x',
'EXT_SOURCE_3_y',
'EXT_SOURCE_2_y',
'EXT_SOURCE_1_y',
'DAYS_EMPLOYED',
'FLOORSMAX_AVG',
'FLOORSMAX_MEDI',
'FLOORSMAX_MODE',
'AMT_GOODS_PRICE',
'REGION_POPULATION_RELATIVE',
'ELEVATORS_AVG',
'REG_CITY_NOT_LIVE_CITY',
'FLAG_EMP_PHONE',
'REG_CITY_NOT_WORK_CITY',
'DAYS_ID_PUBLISH',
'DAYS_LAST_PHONE_CHANGE',
'REGION_RATING_CLIENT',
'REGION_RATING_CLIENT_W_CITY',
'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',
'1',
'EXT_SOURCE_1_y',
'EXT_SOURCE_2_y',
'EXT_SOURCE_3_y',
'DAYS_BIRTH_y',
'EXT_SOURCE_1^2',
'EXT_SOURCE_1 EXT_SOURCE_2',
'EXT_SOURCE_1 EXT_SOURCE_3',
'EXT_SOURCE_1 DAYS_BIRTH',
'EXT_SOURCE_2^2',
'EXT_SOURCE_2 EXT_SOURCE_3',
'EXT_SOURCE_2 DAYS_BIRTH',
'EXT_SOURCE_3^2',
'EXT_SOURCE_3 DAYS_BIRTH',
'DAYS_BIRTH^2',
'EXT_SOURCE_1^3',
'EXT_SOURCE_1^2 EXT_SOURCE_2',
'EXT_SOURCE_1^2 EXT_SOURCE_3',
'EXT_SOURCE_1^2 DAYS_BIRTH',
'EXT_SOURCE_1 EXT_SOURCE_2^2',
'EXT_SOURCE_1 EXT_SOURCE_2 EXT_SOURCE_3',
'EXT_SOURCE_1 EXT_SOURCE_2 DAYS_BIRTH',
'EXT_SOURCE_1 EXT_SOURCE_3^2',
'EXT_SOURCE_1 EXT_SOURCE_3 DAYS_BIRTH',
'EXT_SOURCE_1 DAYS_BIRTH^2',
'EXT_SOURCE_1^2
```

```

        'EXT_SOURCE_2'],
        'EXT_SOURCE_2^2 EXT_SOURCE_3',
        'EXT_SOURCE_2^2 DAYS_BIRTH',
        'EXT_SOURCE_2 EXT_SOURCE_3^2',
        'EXT_SOURCE_2 EXT_SOURCE_3 DAYS_BIRTH',
        'EXT_SOURCE_2 DAYS_BIRTH^2',
        'EXT_SOURCE_3^3',
        'EXT_SOURCE_3^2 DAYS_BIRTH',
        'EXT_SOURCE_3 DAYS_BIRTH^2',
        'DAYS_BIRTH^3']

num_pipeline = Pipeline([
    ('selector', DataFrameSelector(num_attrbs)),
    ('imputer', SimpleImputer(strategy='median')),
    ('std_scaler', StandardScaler()),
])

cat_attrbs = ['FLAG_OWN_REALTY', 'FLAG_OWN_CAR', 'NAME_CONTRACT_TYPE',
              'NAME_FAMILY_STATUS', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS',
              'NAME_HOUSING_TYPE', 'NAME_INCOME_TYPE', 'NAME_FAMILY_STATUS',
              'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE',
              'NAME_INCOME_TYPE', 'NAME_FAMILY_STATUS', 'NAME_EDUCATION_TYPE',
              'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'NAME_INCOME_TYPE']

cat_pipeline = Pipeline([
    ('selector', DataFrameSelector(cat_attrbs)),
    ('imputer', SimpleImputer(strategy='most_frequent')),
    #('imputer', SimpleImputer(strategy='constant', fill_value='mi'),
    ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
])

data_prep_pipeline = FeatureUnion(transformer_list=[
    ("num_pipeline", num_pipeline),
    ("cat_pipeline", cat_pipeline),
])

selected_features = num_attrbs + cat_attrbs
total_features = f'{len(selected_features)}: Num:{len(num_attrbs)}, Total Feature selected for processing {len(cat_attrbs)}'
total_features

# use_application_data_ONLY = False #use joined data
# if use_application_data_ONLY:
#     # just selected a few features for a baseline experiment
#     selected_features = ['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'DAYS_EMPLOYED',
#                           'EXT_SOURCE_2', 'EXT_SOURCE_3', 'CODE_GENDER', 'FLAG_OWN_REALTY',
#                           'NAME_EDUCATION_TYPE', 'OCCUPATION_TYPE', 'NAME_FAMILY_STATUS',
#                           'X_train = datasets["application_train"][selected_features]
#                           y_train = datasets["application_train"]['TARGET']
#                           X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train)
#                           X_train, X_test, y_train, y_test = train_test_split(X_train, y_train)
#                           X_kaggle_test= datasets["application_test"]的文化
#                           # y_test = datasets["application_test"]['TARGET'] #why no TARGET
#                           # selected_features = ['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'DAYS_EMPLOYED',
#                           #                       'EXT_SOURCE_2', 'EXT_SOURCE_3', 'CODE_GENDER', 'FLAG_OWN_REALTY',
#                           #                       'NAME_EDUCATION_TYPE', 'OCCUPATION_TYPE', 'NAME_FAMILY_STATUS',
#                           #                       'NAME_HOUSING_TYPE', 'NAME_INCOME_TYPE', 'NAME_FAMILY_STATUS',
#                           #                       'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE',
#                           #                       'NAME_INCOME_TYPE', 'NAME_FAMILY_STATUS', 'NAME_EDUCATION_TYPE',
#                           #                       'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'NAME_INCOME_TYPE']
```

```

#           'EXT_SOURCE_2','EXT_SOURCE_3','CODE_GENDER', 'FLAG_OWN_REALT
#           'NAME_EDUCATION_TYPE','OCCUPATION_TYPE','NAME_INC
# y_train = X_train['TARGET']
# X_train = X_train[selected_features]
# X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train)
# X_train, X_test, y_train, y_test = train_test_split(X_train, y_train)
# X_kaggle_test= X_kaggle_test[selected_features]
# # y_test = datasets["application_test"]['TARGET']    #why no TARGET?

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

```

Out[22]: '67: Num:63, Cat:4'

In [23]: `# list(train_dataset.columns)`

```

In [26]: X_train = train_dataset[selected_features]
y_train = app_train_poly["TARGET"]
X_kaggle_test = app_test_poly[selected_features]

subsample_rate = 0.3

X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,
                                                    test_size=subsample_rate)

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train,
                                                      test_size=0.15)

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

X train      shape: (182968, 67)
X validation  shape: (32289, 67)
X test        shape: (92254, 67)
X X_kaggle_test shape: (48744, 67)

```

In [ ]:

In [ ]:

```
In [53]: # from sklearn.base import BaseEstimator, TransformerMixin
# import re

# # Creates the following date features
# # But could do so much more with these features
# # E.g.,
# #     extract the domain address of the homepage and OneHotEncode it
# #
# # ['release_month', 'release_day', 'release_year', 'release_dayofweek']
# class prep_OCCUPATION_TYPE(BaseEstimator, TransformerMixin):
#     def __init__(self, features="OCCUPATION_TYPE"): # no *args or **
#         self.features = features
#     def fit(self, X, y=None):
#         return self # nothing else to do
#     def transform(self, X):
#         df = pd.DataFrame(X, columns=self.features)
#         #from IPython.core.debugger import Pdb as pdb;    pdb().set_trace()
#         df['OCCUPATION_TYPE'] = df['OCCUPATION_TYPE'].apply(lambda x: str(x))
#         #df.drop(self.features, axis=1, inplace=True)
#         return np.array(df.values) #return a Numpy Array to observe

# from sklearn.pipeline import make_pipeline
# features = ["OCCUPATION_TYPE"]
# def test_driver_prep_OCCUPATION_TYPE():
#     print(f"X_train.shape: {X_train.shape}\n")
#     print(f"X_train['name'][0:5]: \n{X_train[features][0:5]}")
#     test_pipeline = make_pipeline(prep_OCCUPATION_TYPE(features))
#     return(test_pipeline.fit_transform(X_train))

# x = test_driver_prep_OCCUPATION_TYPE()
# print(f"Test driver: \n{test_driver_prep_OCCUPATION_TYPE()[0:10, :]}")
# print(f"X_train['name'][0:10]: \n{X_train[features][0:10]}")

# # QUESTION, should we lower case df['OCCUPATION_TYPE'] as Sales staff
```

```
In [54]: # # Create a class to select numerical or categorical columns
# # since Scikit-Learn doesn't handle DataFrames yet
# class DataFrameSelector(BaseEstimator, TransformerMixin):
#     def __init__(self, attribute_names):
#         self.attribute_names = attribute_names
#     def fit(self, X, y=None):
#         return self
#     def transform(self, X):
#         return X[self.attribute_names].values
```

```
In [55]: # # Identify the numeric features we wish to consider.  
# num_attribs = [  
#     'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'DAYS_EMPLOYED', 'DAYS_BIRTH', '  
#     'EXT_SOURCE_2', 'EXT_SOURCE_3']  
  
# num_pipeline = Pipeline([  
#     ('selector', DataFrameSelector(num_attribs)),  
#     ('imputer', SimpleImputer(strategy='mean')),  
#     ('std_scaler', StandardScaler()),  
# ])  
# # Identify the categorical features we wish to consider.  
# cat_attribs = ['CODE_GENDER', 'FLAG_OWN_REALTY', 'FLAG_OWN_CAR', 'NAME_  
#                 'NAME_EDUCATION_TYPE', 'OCCUPATION_TYPE', 'NAME_INCOME_  
  
# # Notice handle_unknown="ignore" in OHE which ignore values from the  
# # do NOT occur in the training set  
# cat_pipeline = Pipeline([  
#     ('selector', DataFrameSelector(cat_attribs)),  
#     #('imputer', SimpleImputer(strategy='most_frequent')),  
#     ('imputer', SimpleImputer(strategy='constant', fill_value='n  
#     ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))  
# ])  
  
# data_prep_pipeline = FeatureUnion(transformer_list=[  
#     ("num_pipeline", num_pipeline),  
#     ("cat_pipeline", cat_pipeline),  
# ])
```

```
In [56]: # list(datasets["application_train"].columns)
```

## 8 Baseline Model

To get a baseline, we will use some of the features after being preprocessed through the pipeline. The baseline model is a logistic regression model

```
In [27]: def pct(x):  
    return round(100*x,3)
```

```
In [28]: try:  
    del expLog  
    expLog  
except NameError:  
    expLog = pd.DataFrame(columns=["exp_name",  
                                    "Model name",  
                                    "Train Acc",  
                                    "Valid Acc",  
                                    "Test Acc",  
                                    "Train AUC",  
                                    "Valid AUC",  
                                    "Test AUC",  
                                    "Train F1",  
                                    "Valid F1",  
                                    "Test F1",  
                                    "Fit Time"  
                                ])  
expLog
```

Out[28]:

exp_name	Model name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Train F1	Valid F1	Test F1	Fit Time
----------	------------	-----------	-----------	----------	-----------	-----------	----------	----------	----------	---------	----------

```
In [29]: import time  
np.random.seed(42)  
start_time = time.time()  
full_pipeline_with_predictor = Pipeline([  
    ("preparation", data_prep_pipeline),  
    ("linear", LogisticRegression())  
])  
model = full_pipeline_with_predictor.fit(X_train, y_train)  
fit_time = time.time() - start_time
```

In [30]:

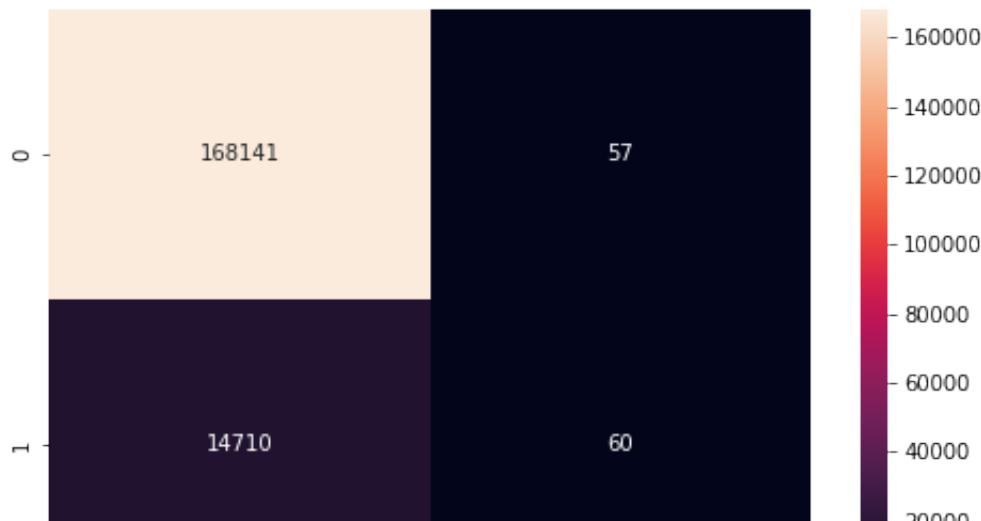
```
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix

print('Accuracy Score on Train Dataset:', np.round(accuracy_score(y_tr
print('F1 Score on Train Dataset:', np.round(f1_score(y_train, model.p
cf_train = confusion_matrix(y_train, model.predict(X_train))
cf_val = confusion_matrix(y_valid, model.predict(X_valid))
cf_test = confusion_matrix(y_test, model.predict(X_test))
plt.figure(figsize=(8,5))
print('Confusion Matrix for Training Set')
sns.heatmap(cf_train, annot=True, fmt='g')
plt.show()
plt.figure(figsize=(8,5))
print('Confusion Matrix for Validation Set')
sns.heatmap(cf_val, annot=True, fmt='g')
plt.show()
plt.figure(figsize=(8,5))
print('Confusion Matrix for Test Set')
sns.heatmap(cf_test, annot=True, fmt='g')
plt.show()
plt.figure(figsize=(10,8))
print('AUC-ROC for Train Set')
plot_roc_curve(model, X_train, y_train);
plt.show()
plt.figure(figsize=(10,8))
print('AUC-ROC for Valid Set')
plot_roc_curve(model, X_valid, y_valid);
plt.show()
plt.figure(figsize=(10,8))
print('AUC-ROC for Test Set')
plot_roc_curve(model, X_test, y_test);
plt.show()
```

Accuracy Score on Train Dataset: 0.919

F1 Score on Train Dataset: 0.008

Confusion Matrix for Training Set



## 8.1 Evaluation metrics

Submissions are evaluated on [area under the ROC curve](#) ([http://en.wikipedia.org/wiki/Receiver\\_operating\\_characteristic](http://en.wikipedia.org/wiki/Receiver_operating_characteristic)) between the predicted probability and the observed target.

The SkLearn `roc_auc_score` function computes the area under the receiver operating characteristic (ROC) curve, which is also denoted by AUC or AUROC. By computing the area under the roc curve, the curve information is summarized in one number.

```
from sklearn.metrics import roc_auc_score
>>> y_true = np.array([0, 0, 1, 1])
>>> y_scores = np.array([0.1, 0.4, 0.35, 0.8])
>>> roc_auc_score(y_true, y_scores)
0.75
```

In [32]: `from sklearn.metrics import roc_auc_score  
print('Accuracy Score on Train Dataset:', roc_auc_score(y_train, model`

Accuracy Score on Train Dataset: 0.7376770017713672

In [33]: `roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])`

Out[33]: 0.7384610162759039

In [34]: `expLog`

Out[34]:

exp_name	Model name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Train F1	Valid F1	Test F1	Fit Time
----------	------------	-----------	-----------	----------	-----------	-----------	----------	----------	----------	---------	----------

### 8.1.1 THE BIG RACE (Baseline Models)

```
In [52]: del expLog
try:
    expLog
except NameError:
    expLog = pd.DataFrame(columns=["exp_name",
                                    "Model name",
                                    "Train Acc",
                                    "Valid Acc",
                                    "Test Acc",
                                    "Train AUC",
                                    "Valid AUC",
                                    "Test AUC",
                                    "Train F1",
                                    "Valid F1",
                                    "Test F1",
                                    "Fit Time (seconds)"])
expLog
```

Out[52]:

exp_name	Model name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Train F1	Valid F1	Test F1	Fit Time (seconds)
----------	------------	-----------	-----------	----------	-----------	-----------	----------	----------	----------	---------	--------------------

```
In [53]: from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.naive_bayes import GaussianNB
```

### 8.1.1.1 Using Non-Ensemble Models

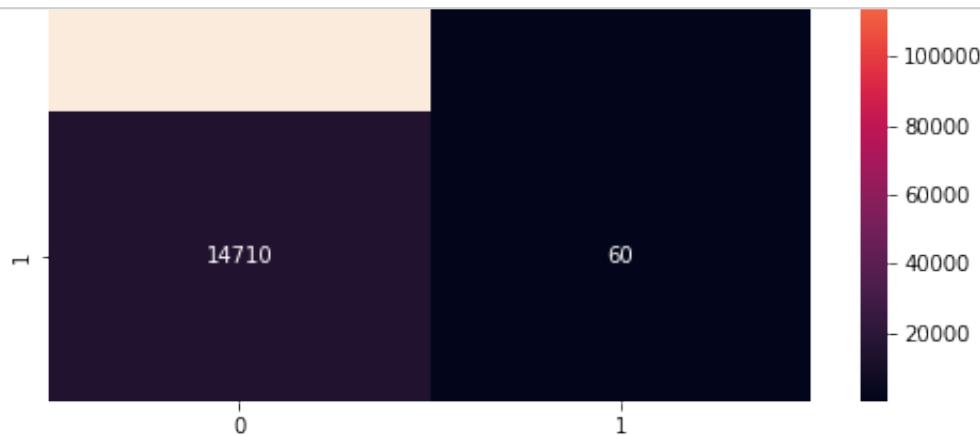
```
In [54]: clfs = [LogisticRegression(penalty='none'),
            LogisticRegression(penalty='l2'),
            DecisionTreeClassifier(),
            GaussianNB()]

for clf in clfs:
    start_time = time.time()
    full_pipeline_with_predictor = Pipeline([
        ("preparation", data_prep_pipeline),
        ("model", clf)
    ])
    model_name = "Baseline {}".format(type(full_pipeline_with_predictor))
    model = full_pipeline_with_predictor.fit(X_train, y_train)
    fit_time = time.time() - start_time
    print('Fit Time for {} is: {} seconds'.format(model_name, fit_time))
    exp_name = f"Baseline_{len(selected_features)}_features"
    print(model_name)
    expLog.loc[len(expLog)] = [f"{{exp_name}}"] + [model_name] + list(np
        accuracy_score(y_train, model.predict(X_train)))
```

```
accuracy_score(y_train, model.predict(X_train)),
accuracy_score(y_valid, model.predict(X_valid)),
accuracy_score(y_test, model.predict(X_test)),
roc_auc_score(y_train, model.predict_proba(X_train),
roc_auc_score(y_valid, model.predict_proba(X_valid),
roc_auc_score(y_test, model.predict_proba(X_test)),
f1_score(y_train, model.predict(X_train)),
f1_score(y_valid, model.predict(X_valid)),
f1_score(y_test, model.predict(X_test)),
fit_time], 4))

cf_train = confusion_matrix(y_train, model.predict(X_train))
cf_val = confusion_matrix(y_valid, model.predict(X_valid))
cf_test = confusion_matrix(y_test, model.predict(X_test))
plt.figure(figsize=(8,5))
print('Confusion Matrix for Training Set')
sns.heatmap(cf_train, annot=True, fmt='g')
plt.show()
plt.figure(figsize=(8,5))
print('Confusion Matrix for Validation Set')
sns.heatmap(cf_val, annot=True, fmt='g')
plt.show()
plt.figure(figsize=(8,5))
print('Confusion Matrix for Test Set')
sns.heatmap(cf_test, annot=True, fmt='g')
plt.show()
plt.figure(figsize=(10,8))
print('AUC-ROC for Train Set')
plot_roc_curve(model, X_train, y_train);
plt.show()
plt.figure(figsize=(10,8))
print('AUC-ROC for Valid Set')
plot_roc_curve(model, X_valid, y_valid);
plt.show()
plt.figure(figsize=(10,8))
print('AUC-ROC for Test Set')
plot_roc_curve(model, X_test, y_test);
plt.show()
```

expLog



### Confusion Matrix for Validation Set



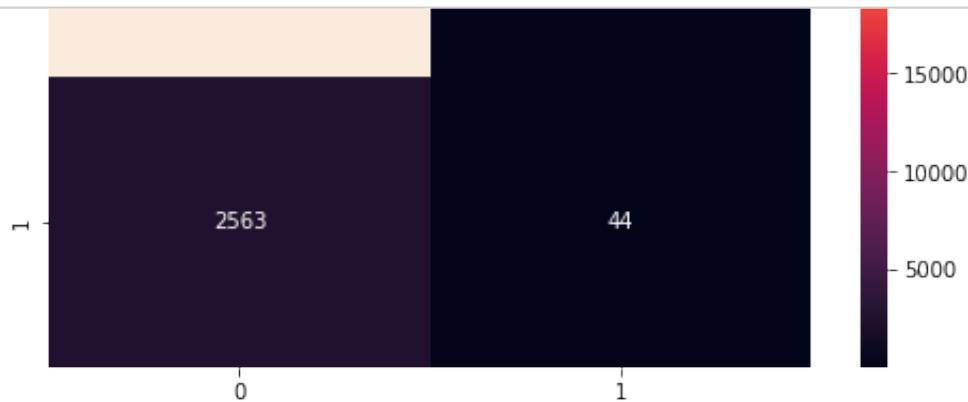
#### 8.1.1.2 Using Ensemble Models

```
In [55]: clfs = [RandomForestClassifier(), XGBClassifier()]

for clf in clfs:
    start_time = time.time()
    full_pipeline_with_predictor = Pipeline([
        ("preparation", data_prep_pipeline),
        ("model", clf)
    ])
    model_name = "Baseline {}".format(type(full_pipeline_with_predictor))
    model = full_pipeline_with_predictor.fit(X_train, y_train)
    fit_time = time.time() - start_time
    print('Fit Time for {} is: {} seconds'.format(model_name, fit_time))
    exp_name = f"Baseline_{len(selected_features)}_features"
    print(model_name)
    expLog.loc[len(expLog)] = [f"{{exp_name}}"] + [model_name] + list(np
        [accuracy_score(y_train, model.predict(X_train)),
         accuracy_score(y_valid, model.predict(X_valid)),
         accuracy_score(y_test, model.predict(X_test)),
         roc_auc_score(y_train, model.predict_proba(X_train)),
         roc_auc_score(y_valid, model.predict_proba(X_valid)),
         roc_auc_score(y_test, model.predict_proba(X_test)),
         f1_score(y_train, model.predict(X_train)),
         f1_score(y_valid, model.predict(X_valid)),
         f1_score(y_test, model.predict(X_test)),
         fit_time], 4))

    cf_train = confusion_matrix(y_train, model.predict(X_train))
    cf_val = confusion_matrix(y_valid, model.predict(X_valid))
    cf_test = confusion_matrix(y_test, model.predict(X_test))
    plt.figure(figsize=(8,5))
    print('Confusion Matrix for Training Set')
    sns.heatmap(cf_train, annot=True, fmt='g')
    plt.show()
    plt.figure(figsize=(8,5))
    print('Confusion Matrix for Validation Set')
    sns.heatmap(cf_val, annot=True, fmt='g')
    plt.show()
    plt.figure(figsize=(8,5))
    print('Confusion Matrix for Test Set')
    sns.heatmap(cf_test, annot=True, fmt='g')
    plt.show()
```

```
sns.heatmap(cf_test, annot=True, fmt='g')
plt.show()
plt.figure(figsize=(10,8))
print('AUC-ROC for Train Set')
plot_roc_curve(model, X_train, y_train);
plt.show()
plt.figure(figsize=(10,8))
print('AUC-ROC for Valid Set')
plot_roc_curve(model, X_valid, y_valid);
plt.show()
plt.figure(figsize=(10,8))
print('AUC-ROC for Test Set')
plot_roc_curve(model, X_test, y_test);
plt.show()
```



Confusion Matrix for Test Set



In [56]: expLog

Out[56]:

	exp_name	Model name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	Baseline_67_features	Baseline LogisticRegression	0.9193	0.9194	0.9195	0.7378	0.7299	0.7386
1	Baseline_67_features	Baseline LogisticRegression	0.9193	0.9193	0.9195	0.7377	0.7300	0.7385
2	Baseline_67_features	Baseline DecisionTreeClassifier	1.0000	0.8503	0.8512	1.0000	0.5318	0.5387
3	Baseline_67_features	Baseline GaussianNB	0.6791	0.6747	0.6812	0.7229	0.7156	0.7225
4	Baseline_67_features	Baseline RandomForestClassifier	1.0000	0.9188	0.9192	1.0000	0.6858	0.6958
5	Baseline_67_features	Baseline XGBClassifier	0.9231	0.9185	0.9190	0.8586	0.7290	0.7381

**8.1.1.3 KNeighborsClassifier, SVC, and Logistic Regression with L1 regularization with solver='liblinear' take a lot of time to run and crash the kernel. So we were unable to train the dataset on these models.**

In [47]:

```

# from sklearn.svm import SVC
# from sklearn.neighbors import KNeighborsClassifier
# clfs = [SVC(), KNeighborsClassifier()]

# for clf in clfs:
#     start_time = time.time()
#     full_pipeline_with_predictor = Pipeline([
#         ("preparation", data_prep_pipeline),
#         ("model", clf)
#     ])
#     model_name = "Baseline {}".format(type(full_pipeline_with_predictor))
#     model = full_pipeline_with_predictor.fit(X_train, y_train)
#     fit_time = time.time() - start_time
#     print('Fit Time for {} is: {} seconds'.format(model_name, fit_time))
#     exp_name = f"Baseline_{len(selected_features)}_features"

#     expLog.loc[len(expLog)] = [f"{{exp_name}}"] + [model_name] + list([
#         accuracy_score(y_train, model.predict(X_train)),
#         accuracy_score(y_valid, model.predict(X_valid)),
#         accuracy_score(y_test, model.predict(X_test)),
#         roc_auc_score(y_train, model.predict_proba(X_train)),
#         roc_auc_score(y_valid, model.predict_proba(X_valid)),
#         roc_auc_score(y_test, model.predict_proba(X_test)),
#         f1_score(y_train, model.predict(X_train)),
#         f1_score(y_valid, model.predict(X_valid)),
#         f1_score(y_test, model.predict(X_test)),
#         fit_time], 4))
# expLog

```

## 8.2 Submission File Prep

For each SK\_ID\_CURR in the test set, you must predict a probability for the TARGET variable. The file should contain a header and have the following format:

```

SK_ID_CURR,TARGET
100001,0.1
100005,0.9
100013,0.2
etc.

```

### 8.2.1 Using Logistic Regression (Ridge) for our baseline submission

```
In [57]: full_pipeline_with_predictor = Pipeline([
    ("preparation", data_prep_pipeline),
    ("linear", LogisticRegression(penalty='l2'))
])
model = full_pipeline_with_predictor.fit(X_train, y_train)
```

```
In [58]: test_class_scores = model.predict_proba(X_kaggle_test)[:, 1]
```

```
In [59]: test_class_scores[0:10]
```

```
Out[59]: array([0.06787096, 0.13054041, 0.02292311, 0.03216032, 0.08954263,
       0.02947744, 0.0363267 , 0.08377857, 0.01796927, 0.16895726])
```

```
In [61]: # Submission dataframe
submit_df = app_test[['SK_ID_CURR']]
submit_df['TARGET'] = test_class_scores

submit_df.head()
```

```
Out[61]: SK_ID_CURR  TARGET
_____
0      100001  0.067871
1      100005  0.130540
2      100013  0.022923
3      100028  0.032160
4      100038  0.089543
```

```
In [62]: submit_df.to_csv("submission.csv", index=False)
```

## 9 Kaggle submission via the command line API

```
In [63]: ! kaggle competitions submit -c home-credit-default-risk -f submission
100%|██████████| 1.26M/1.26M [00:01<00:00
, 1.10MB/s]
Successfully submitted to Home Credit Default Risk
```

## 9.1 report submission

Click on this [link \(<https://www.kaggle.com/c/home-credit-default-risk/submissions?sortBy=date&group=all&page=1>\)](https://www.kaggle.com/c/home-credit-default-risk/submissions?sortBy=date&group=all&page=1)

# 10 Write-up

For this phase of the project, you will need to submit a write-up summarizing the work you did. The write-up form is available on Canvas (Modules-> Module 12.1 - Course Project - Home Credit Default Risk (HCDR)-> FP Phase 2 (HCDR) : write-up form ). It has the following sections:

## 10.1 Abstract

Please provide an abstract summarizing the work you did (150 words)

## 10.2 Introduction

## 10.3 Feature Engineering and transformers

Please explain the work you conducted on feature engineering and transformers. Please include code sections when necessary as well as images or any relevant material

## 10.4 Pipelines

Please explain the pipelines you created for this project and how you used them Please include code sections when necessary as well as images or any relevant material

## 10.5 Experimental results

Please present the results of the various experiments that you conducted. The results should be shown in a table or image. Try to include the different details for each experiment.

Please include code sections when necessary as well as images or any relevant material

## 10.6 Discussion

Discuss & analyze your different experimental results

Please include code sections when necessary as well as images or any relevant material

## 10.7 Conclusion

## 10.8 Kaggle Submission

Please provide a screenshot of your best kaggle submission.

The screenshot should show the different details of the submission and not just the score.

## 11 References

Some of the material in this notebook has been adopted from [here](https://www.kaggle.com/willkoehrsen/start-here-a-gentle-introduction/notebook) (<https://www.kaggle.com/willkoehrsen/start-here-a-gentle-introduction/notebook>)

In [ ]:

## 12 TODO: Predicting Loan Repayment with Automated Feature Engineering in Featuretools

Read the following:

- feature engineering via Featuretools library:
  - <https://github.com/Featuretools/predict-loan-repayment/blob/master/Automated%20Loan%20Repayment.ipynb> (<https://github.com/Featuretools/predict-loan-repayment/blob/master/Automated%20Loan%20Repayment.ipynb>)
  - <https://www.analyticsvidhya.com/blog/2018/08/guide-automated-feature-engineering-featuretools-python/> (<https://www.analyticsvidhya.com/blog/2018/08/guide-automated-feature-engineering-featuretools-python/>)
- feature engineering paper: [https://dai.lids.mit.edu/wp-content/uploads/2017/10/DSAA\\_DSM\\_2015.pdf](https://dai.lids.mit.edu/wp-content/uploads/2017/10/DSAA_DSM_2015.pdf) ([https://dai.lids.mit.edu/wp-content/uploads/2017/10/DSAA\\_DSM\\_2015.pdf](https://dai.lids.mit.edu/wp-content/uploads/2017/10/DSAA_DSM_2015.pdf))
- <https://www.analyticsvidhya.com/blog/2017/08/catboost-automated-categorical-data/> (<https://www.analyticsvidhya.com/blog/2017/08/catboost-automated-categorical-data/>)

