Credit Risk Scoring

**Industry Partner: Mr Nick Jonker (Senior Modelling Manager, Retail, ANZ)**

**Course Coordinator: Dr Yan Wang**

# Table of Contents

[Table of Contents 5](#_Toc528657871)

[Executive Summary 6](#_Toc528657872)

[Introduction 7](#_Toc528657873)

[Project Overview 7](#_Toc528657874)

[Significance of the project 7](#_Toc528657875)

[Materials and Methods 8](#_Toc528657876)

[Data 8](#_Toc528657877)

[Data tables used: 9](#_Toc528657878)

[Data pre-processing: 10](#_Toc528657879)

[Sample visualizations and data explorations: 10](#_Toc528657880)

[Validation Metric: 19](#_Toc528657881)

[Modelling approaches 19](#_Toc528657882)

[Overview of the various modelling approaches 19](#_Toc528657883)

[Feature selection and engineering: 22](#_Toc528657884)

[Results 23](#_Toc528657885)

[Discussion/recommendation 29](#_Toc528657886)

[Recommended model & model comparison notes: 29](#_Toc528657887)

[Comment on Feature importance 29](#_Toc528657888)

[Limitations of our approach 29](#_Toc528657889)

[Conclusion/Reflection 30](#_Toc528657890)

[References 31](#_Toc528657891)

[Appendix 32](#_Toc528657892)

# Executive Summary

The context and purpose of the project was to determine which customers are likely to default from a Consumer Finance company to minimize non-performing loans that have a high risk of crystallizing into bad debts. Consumer Finance or Lending Institutions have had more scrutiny in recent years especially after

This project was approached by understanding what conventional methods lending institutions use to minimize risky loans to customers by undertaking and reading research journals.

Researching Academic Journals provided invaluable insight to gain more domain knowledge on what Algorithms to use. In context there are many Algorithms to use for a variety of data sources, but focus was on using algorithms that have a proven track record in Consumer Finance or Financial Institutions.

The most effective algorithm discovered was the Light Gradient boosting method (LGBM) with an Area under the curve Receiver operating Characteristic score of 0.784, the higher the score to 1 the better, the other models were Random Forest, Logit and a simple Artificial neural network.

# Introduction

## Project Overview

This credit risk scoring project has been undertaken on the data made public by

Home Credit (2018), a company in the Czech Republic primarily concerned with providing credit to individuals who do not have a proper credit history and essentially aims to develop methods to predict customers most likely to default on their loan application so that their applications can be identified and rejected in time to minimize any financial strain that the company may suffer due to future bad debts

For these purposes Home Credit organized a data competition on a major data science competitions website called Kaggle (2018) from where the relevant data has been retrieved. The data provided was both internally and externally sourced i.e. internally collected data from Home Credit as well as relevant external sources to determine credit worthiness of customers on obtaining a loan.

## Significance of the project

Financial Institutions ideally want to minimize a metric called Non-Performing Loans as this may force the Financial Institution to raise more capital or be given an undertaking to raise more Tier 1; buffer for credit shocks. As a note a Non-Performing Loan is defined as a scheduled payment that is at least 90 days overdue for commercial loan or in this case 180 days for a Consumer Loan for Home Credit but may be subject to change depending upon the country or legal jurisdiction.

In addition to the monetary considerations of the financial companies, efficient allocation of capital as credit to those who can repay will assist many those who may have been overlooked by the mainstream credit systems.

# Materials and Methods

## Data

Data for the project has been provided by Home Credit itself and comprises the tables below, the relationship of these table can also be seen in Fig. 1

There are 7 tables with over 220 features.

**Application Data** - This is the main data that is broken into training and test data which provides information of the customer for the current application.

This has the ‘Target’ feature where 0 means no default and 1 is default

**Bureau** - Data is the customer’s credit history from other financial institutions that were reported to a credit bureau.

**Bureau Balance** - This data is monthly balance with credit bureau of previous loans.

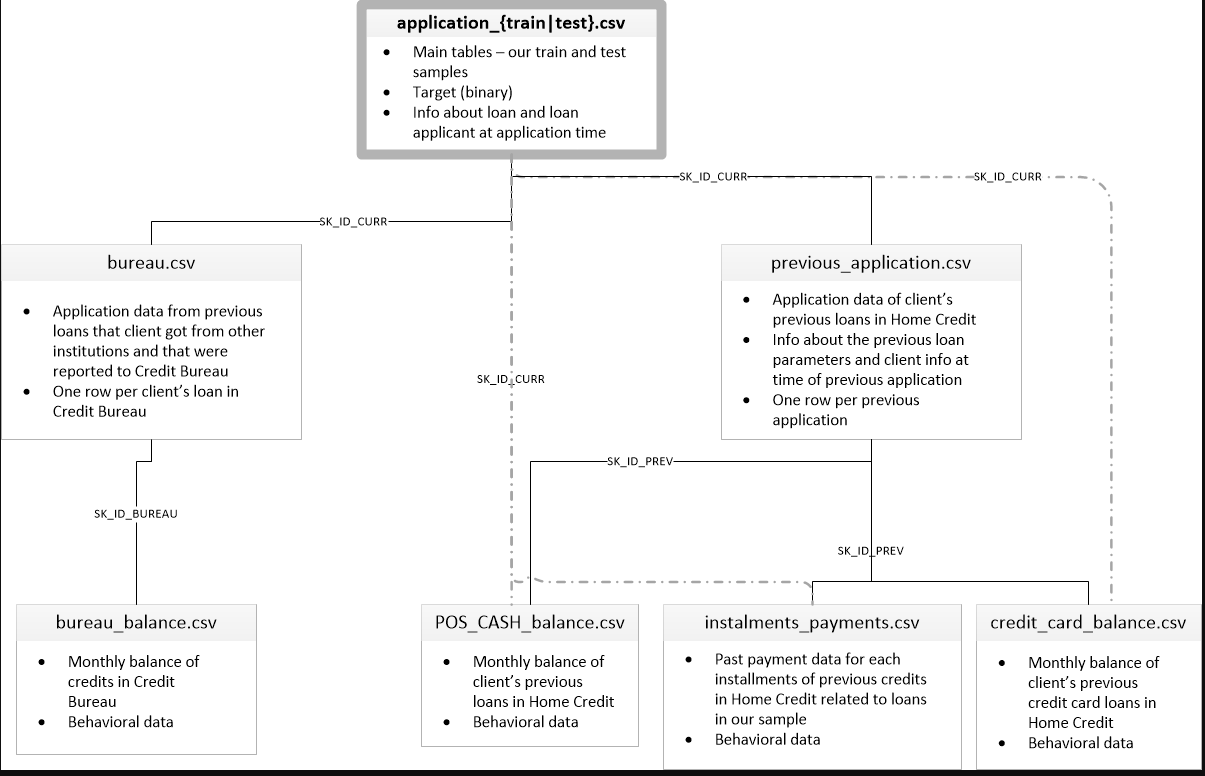
**POS CASH Balance** - Snapshot monthly data of spending pattern and cash loans customer had with Home Credit.

**Credit Card Balance** – Data is monthly credit card balance of previous credit cards customer has had with Home Credit.

**Previous Application** – All previous applications of loans customer has had with Home Credit.

**Instalments Payments**- Loan repayment history Customer has had with Home Credit for Previous Loans.

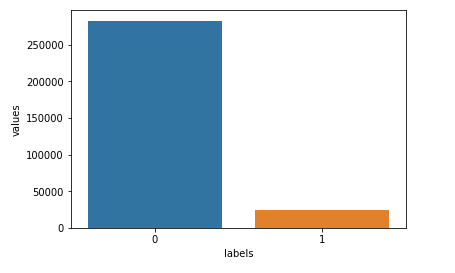
**HomeCredit columns description** – This file is a reference list to give a definition on all the variables of all the files.



**Fig 1**

### Data tables used:

Prior to performing our analysis and working towards a final Analytical Base Table, to get an idea of the tables at hand and get a sense of the quantum of missing values for our training ids, as it is common for such datasets to have these issues. Because of this analysis, we had to discard the ‘Credit Card Balance’ table from joining it with the main ‘application train’ table owing to the high missing values as seen in Fig 2



**Fig 2 No-Default i.e. 0 and Default (1) are imbalanced**

### Data pre-processing:

The Kaggle data was clean and though there were issues surrounding missing values we had very minor pre-processing tasks to consider.

Some pre-processing was needed as follows:

* XAP, XNP values were NaNs and had to be factored
* Day columns had NaNs as 365243
* Binning and grouping were done for certain variables such as age and income etc.
* For data imputation we have used the mean values for numeric and special values for categorical data.

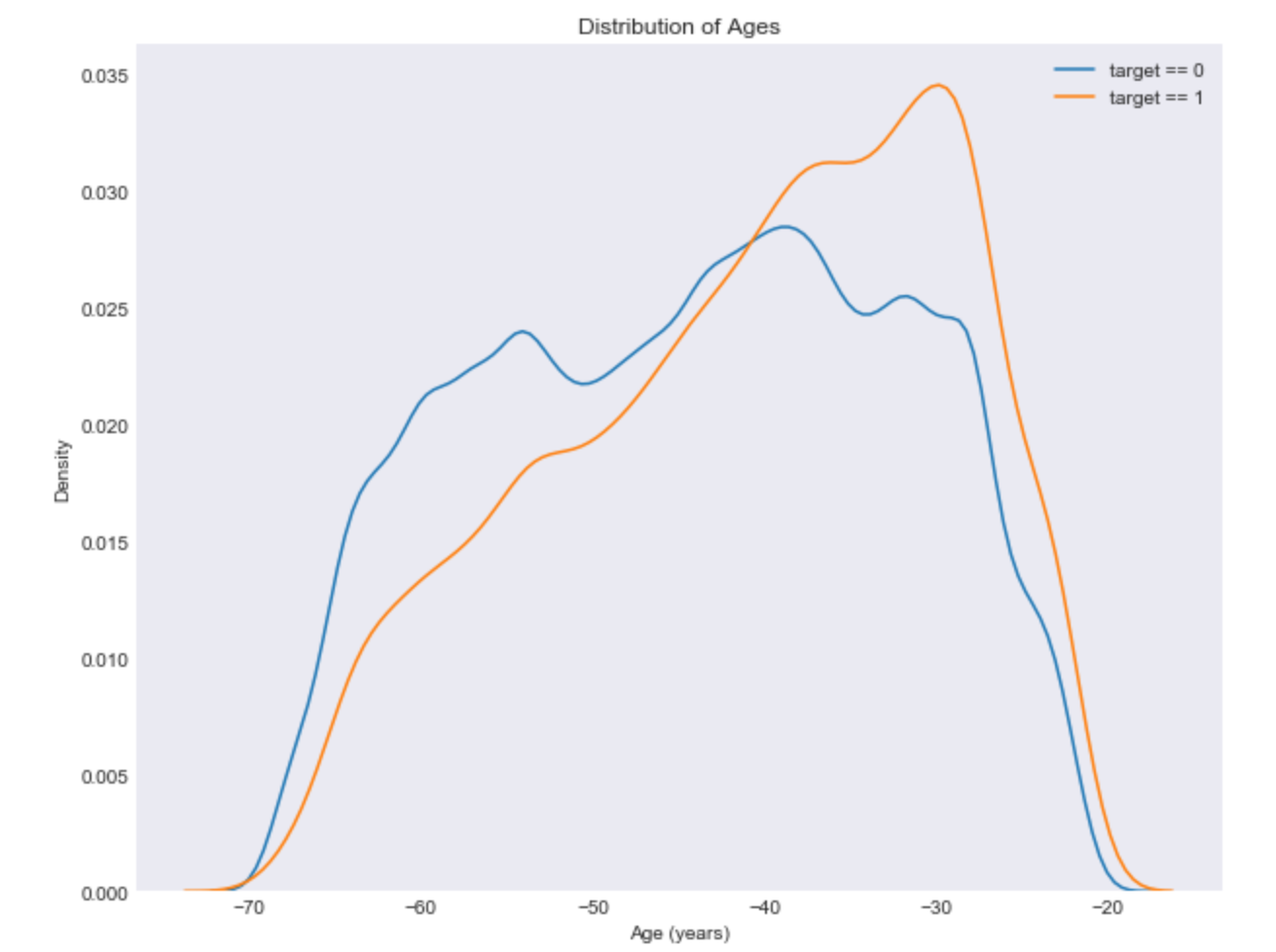
***All codes have been included in the Appendix section***

### Sample visualizations and data explorations:

Below are few of the sample visualizations and explorations performed to aid us in working out the best way to move forward in our analysis.

**Age Distributions:**

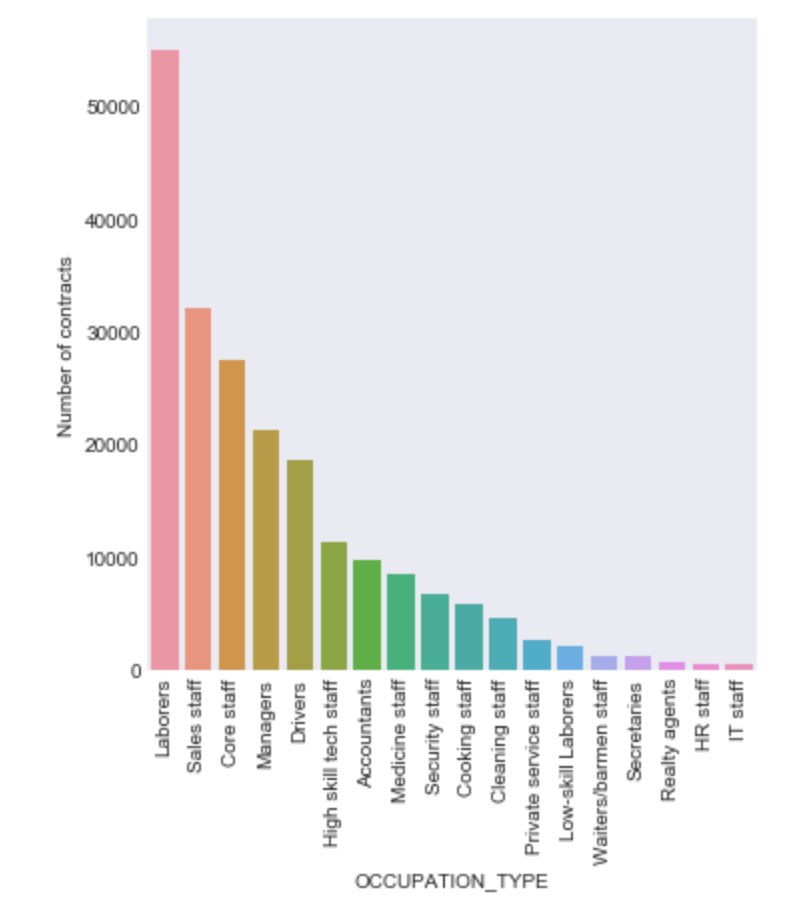
We can clearly see from the two density plots that lower the age , more the chances of default.



**Occupation**

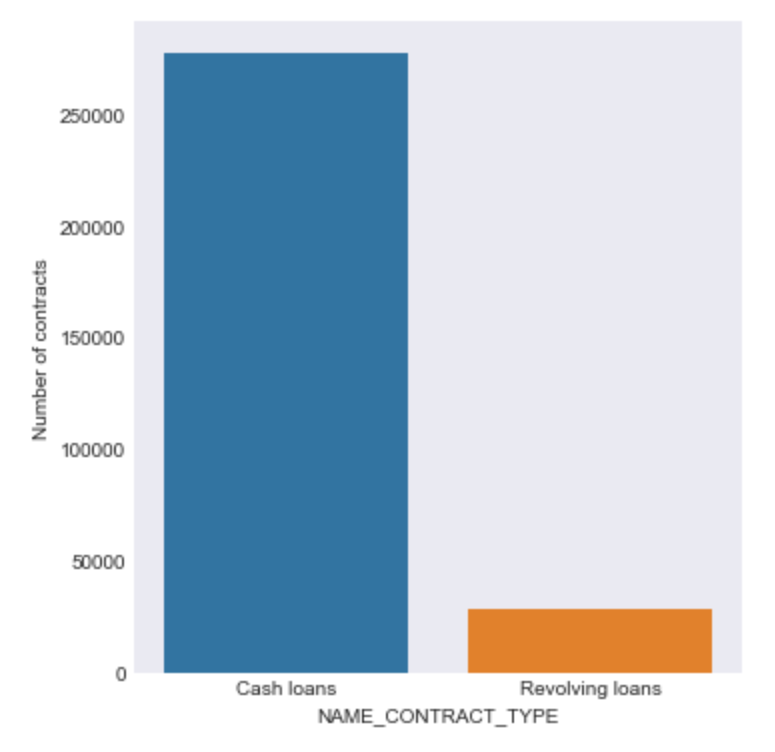
An observation of the occupations is that a large portion are classified in blue collar or low skills occupations. Some of the most highly skilled staff being Managers, High Skill tech staff, Accountants are the most notable of the skilled occupations.

Due to the nature of lending to customers for Home Credit with little or no credit history.



**Marital Status**

The majority of customers are married by a significant margin over other types of customers such as single and those classified as civil marriage.

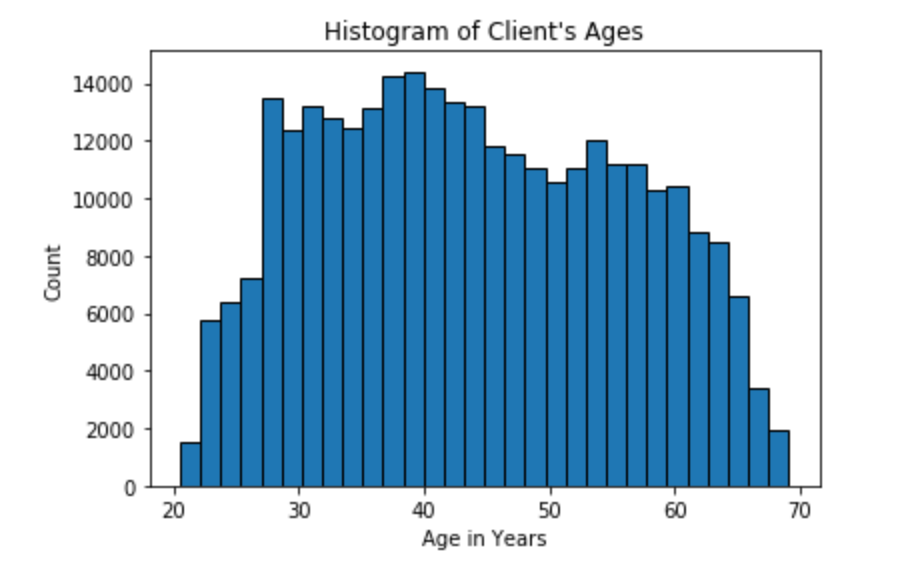


**Loan Type**

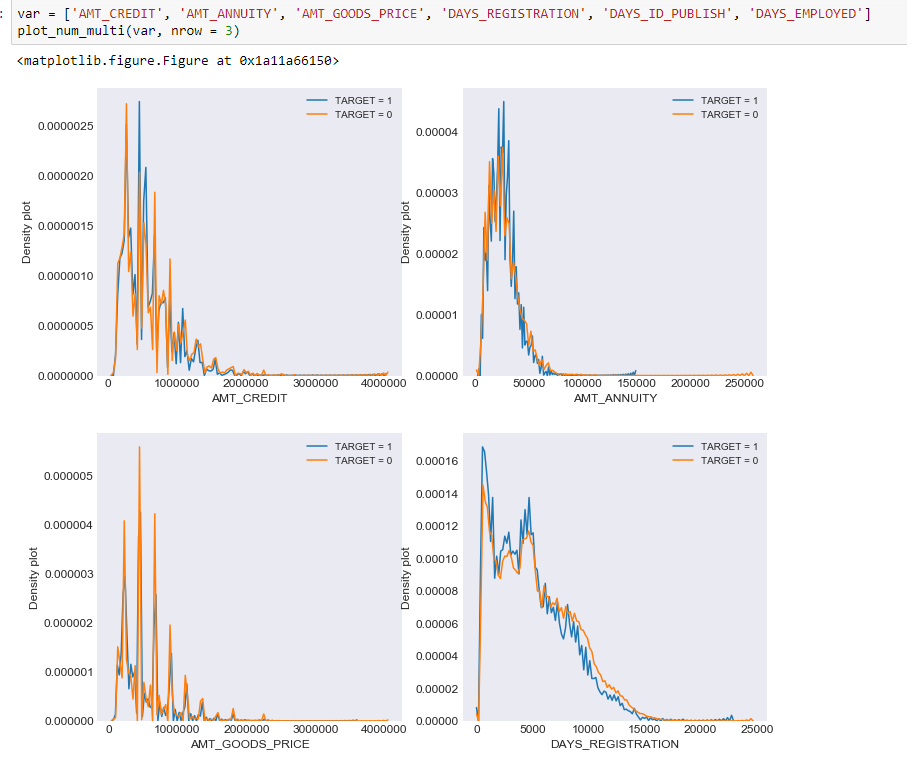
The vast majority of customers applied for one cash loans with the remaining customers applying for revolving loans with credit cards being the most common type of revolving loan.

**Population Distribution**

The population has slight normal distribution from ages 20 – 70, there are no customers below age 20 or over age 70.

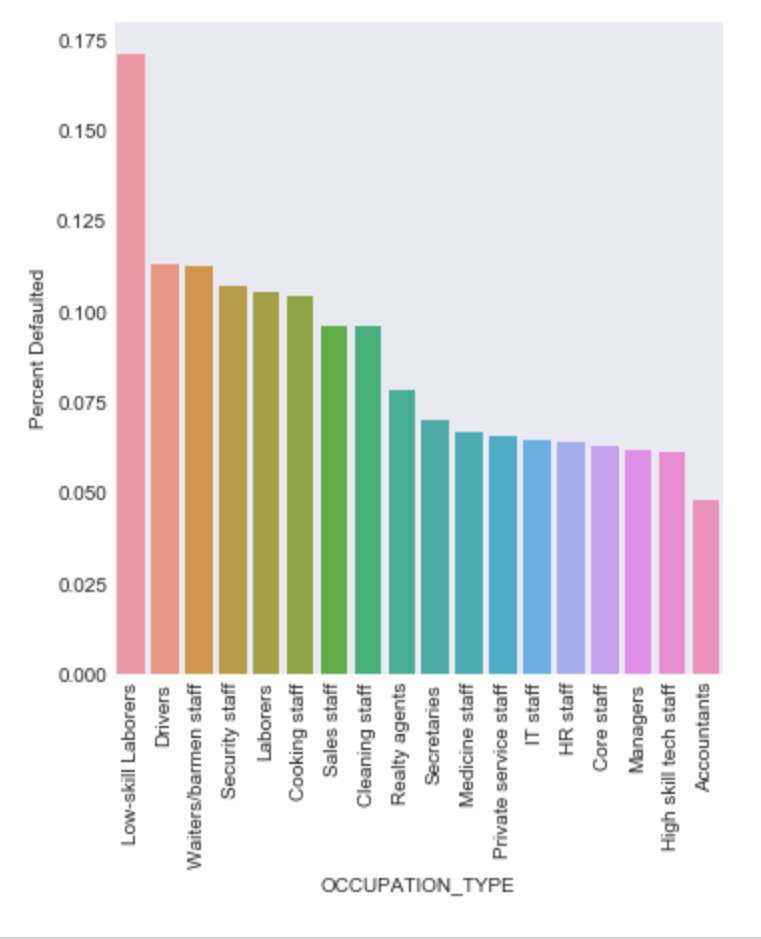


**Distributions:**



**Income Type Default**

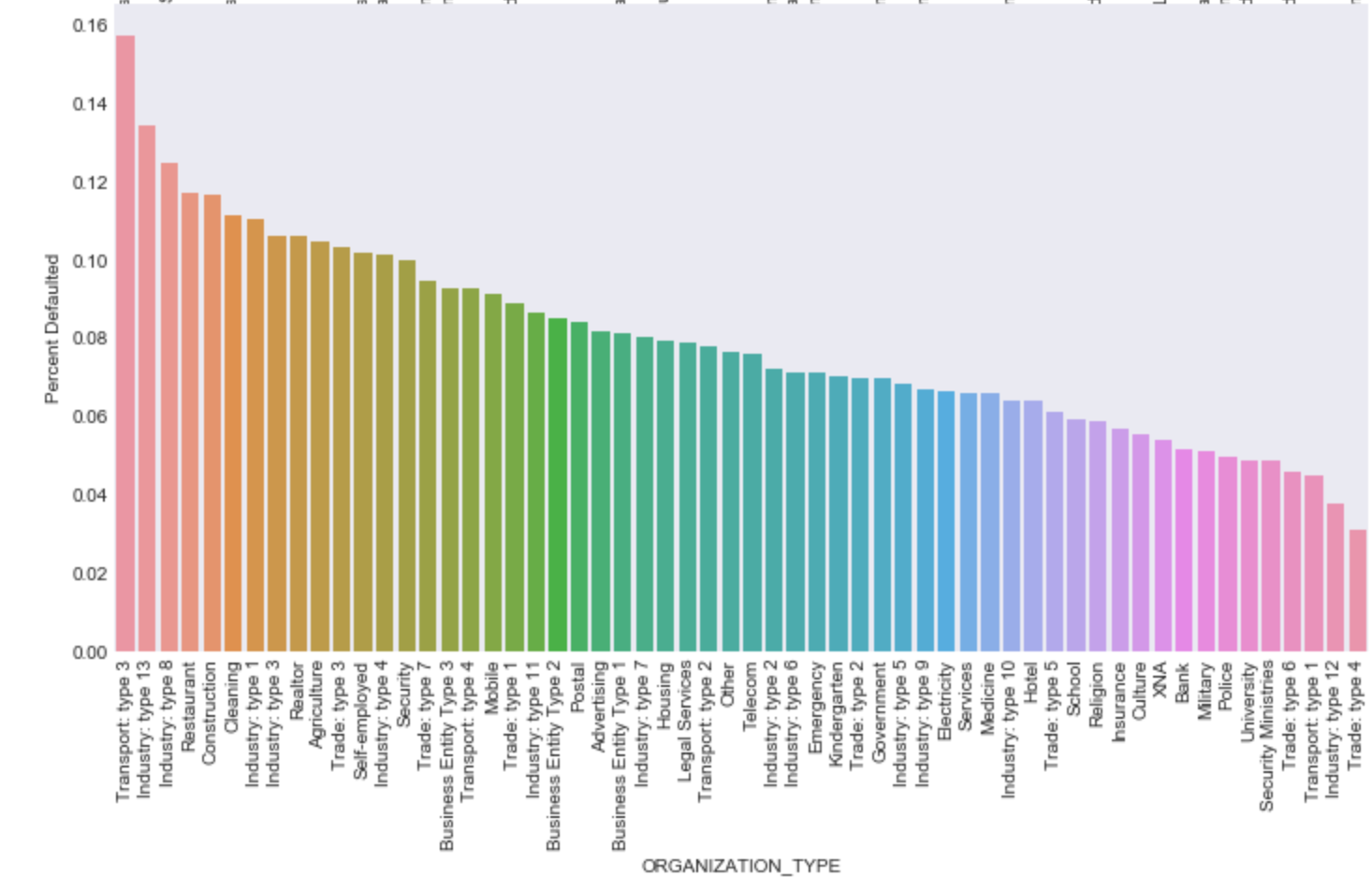
Customers that are out of work are the most likely to default with approximately 40% of customers on Maternity Leave likely to default followed by approximately 36% of unemployed customers defaulting. Customers that were working had a default rate of approximately of 9%, Pensioners had a default rate of approximately 5%.

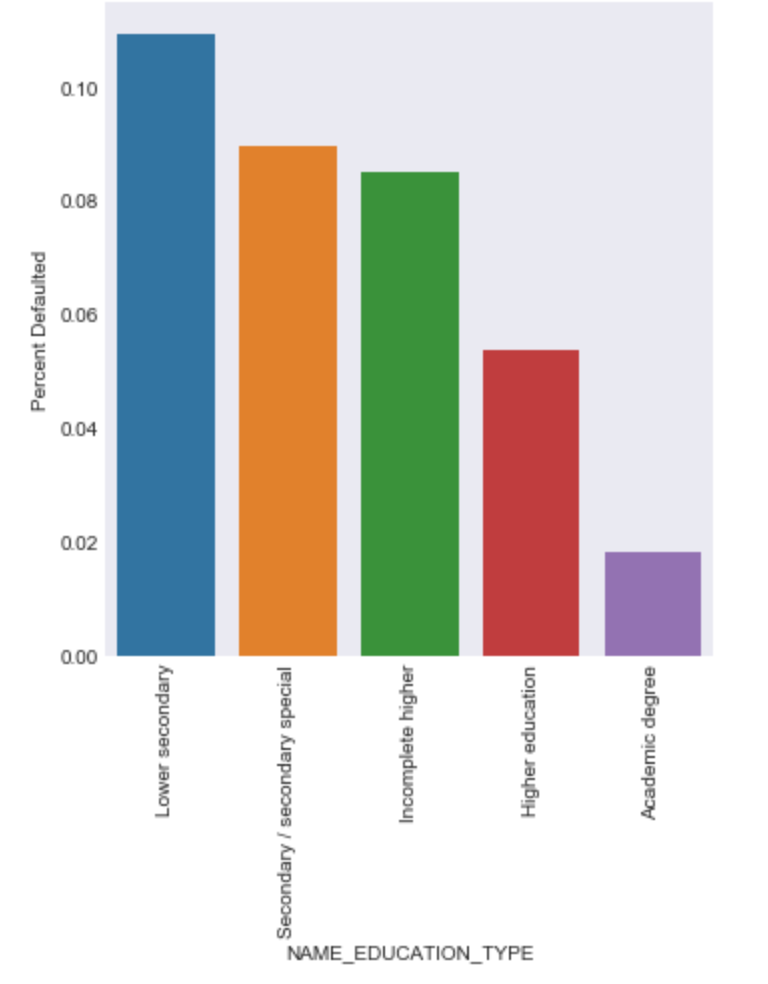


**Default based on Occupation Type**

Low skill occupations are the most likely to default with Low-skill Labourers at around 17% followed by Driver and Waiters/barmen staff at around 11%.

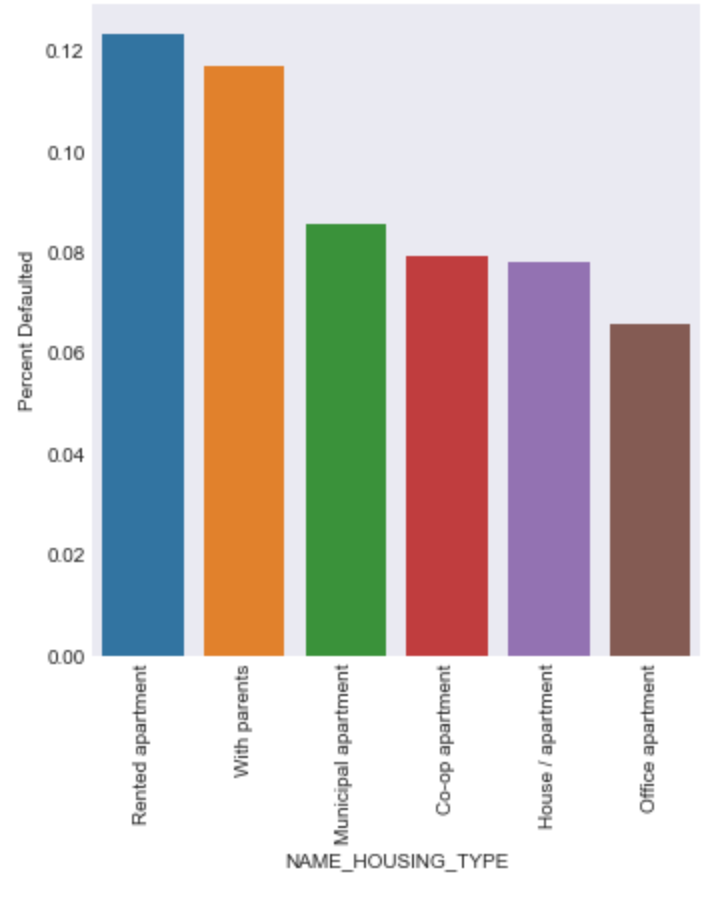
Customers with a higher skill occupation such as Accountant or High skill tech staff are the least likely to default.





**Education Type Default**

The most educated customers are the least likely to default with either an Academic degree or Higher Education were the least likely to default at approximately 2% and 5% respectively. Customers with Lower Secondary were the most likely to default followed by Secondary/Secondary Special at approximately 10.6% and 8.2% respectively.



**Housing Type Default**

Customers with a Rented Apartment or Living with Parents were most likely to default with Rented apartment at approximately 12.1% and 11.8% respectively. Customers living in an office apartment were least likely to default at approximately 6.2%.

***More samples have been included in the appendix***

## Validation Metric:

Before we delve into the methods and other rationales and steps undertaken as part of the project, it is important to establish the quantitative validation metric that will be used to ascertain how our models perform and the desired metric as per the Kaggle competition as well as standard practice for our given data set is the AUROC i.e. the Area under the curve for the Receiver Operating Characteristic which looks to factor the errors that would generate from classifying i.e. false negatives and false positives and assists in determining the goodness of the algorithm when it comes to its performance on the given task,

The area limit ranges from a minimum value of 0, to a maximum of 1 with the area under the curve being the quantified value of how good a classifier performs. An area of 1 indicates that the classiﬁer is perfect while 0 means that everything was misclassified. And as logically follows 0.5 i.e. half is if the classifier has randomly classified the targets. (Streiner & Cairney, 2007).

The ROC is also a better measure when there is a significant imbalance in the target class distributions, which we can see from Fig 2 is the case for our task.

## Modelling approaches

At its core this was a logistic regression task as even though were predicting the probabilities of default instead of the binary 0 or 1, so naturally logistic regression became our base model. Other than that, the Random Forest, Gradient Boosting Method using light gbm and an MLP i.e. Multilayer perceptron were explored.

### Overview of the various modelling approaches

#### Logistic Regression:

Logistic Regression is a very popular statistical method with different types of variables (ordinal, nominal, interval, ratio-level) in this case there are over 220+ to determine one of two outcome (Dichotomous), this case it’ s to predict on whether a customer of Home Credit will likely default or not. Logistic Regression has many advantages such as not assuming a linear relationship between the independent variables and dependent variable (Feature Target Variable for default), handling of imbalanced data well as variables including the feature target are imbalanced. There can be some disadvantages with Logistic Regression particularly around sensitivity to outliers (the need for data pre-processing), handling a of categorical features (the dataset has many important categorical variables). Independent variables that are not well correlated or to each other to the target feature (likely to default) do not generally perform well. (Dong et al, 2010)

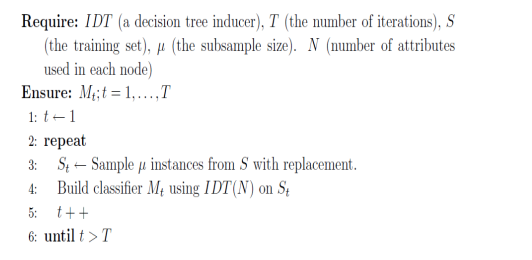
#### Random Forest:

Random Forest is an ensemble method which picks random variables to group up into trees with the feature target variable to come up with a variety of models to determine which variables have a correlated relationship with target variable.

The Fig 3 below also shows a pseudo code for the Random Forest algorithm.

In simple terms a Random Forest randomly selects variables or clusters them to narrow down the pertinent variables to come up with a useful prediction.

An advantage of a random forest is the reduction in over-fitting trees are averaged and by using multiple trees the probability of selecting a variable that does not perform well is reduced and in this case with the dataset having many variables the random selection of variables the probability of picking an incorrect variable is reduced due to random selection of many variables. (Akbhari & Mardukhi, 2014). A disadvantage of a random forest is the large number of trees can make random forest slow to make a prediction and in this case with many variables (220+) there is a challenge in making a fast prediction of which customers are likely to default especially where aggregation methods may be used to combine various datasets.

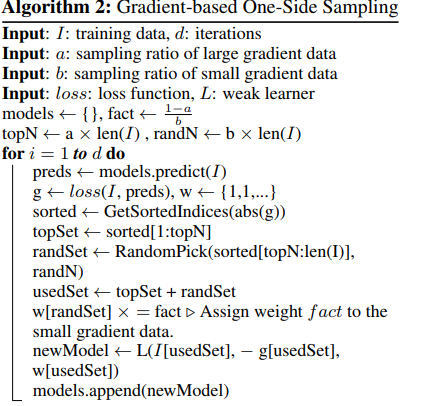


**Fig 3 Pseudo code for a Random Forest algorithm**

#### Light-Gradient boosting method (LightGBM):

The gradient boosting decision tree models have been very successful recently mainly due to their ability to handle sparse data along with robustness to outliers. Like Random Forests discussed above these are also tree based models with a key difference that these are trained in sequence by fitting the negative gradients or residual errors at each iteration.

This method has shown them to be able to split the data at points maximizing the information gain for each split thus making the classification process fast which is very useful for large datasets and variants for the boosting such as Gradient-based One-Side Sampling [GOSS], as seen in Fig 4, also ensure that the instances with small gradients are not discarded which may lead to change in the data distributions, thus improving the performance of the model. (Ke Et al, 2017)



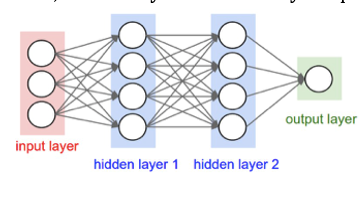
**Fig 4 LGBM GOSS algorithm**

#### Multilayer perceptron (MLP)- ANN

We also ran a simple artificial neural network i.e. a feed forward MLP model

multilayer perceptron is a logistic regressor where instead of feeding the input to the logistic regression an intermediate layer, called the hidden layer, is inserted and usually has an activation function which non-linear to which can make the models more robust than mere linear approaches.

Neurons and Neural networks:To get a basic appreciation of a neuron and an artificial neural network we need to understand that they essentially try to mimic the real-life biological processes i.e. a single neuron takes input signals from its dendrites to produce a single output signal, which has first been worked upon by some hidden layer(s) to produce an output (Jiang Y et al., 2017) as can be seen from Fig. 5 below, and these layers are



**Fig. 5 Simple neural network**

## Feature selection and engineering:

Feature selection:

This is a process in machine learning, in which each feature in the dataset is analysed and only those features which are relevant are filtered through in the model building phase. In the feature selection process, no new variables are created and so it is only a subset of features from the main dataset that is then processed.

The idea behind doing feature selection is that we’d like to filter out as much ‘garbage’ i.e. unwanted features from our dataset, so that a model’s learning/training rate is much faster and more accurate. This is a very critical step as features contain information about the target variable. However, having more features doesn’t always mean that we’d have more information and better classification performance or discriminative power to train the model. These irrelevant features introduce ‘noise’ which can trick the learning algorithm or model by steering it in the wrong direction. We can also have redundant features also do not contribute any additional information and leads to degradation in the model’s accuracy or performance.

These irrelevant and redundant features, combined with limited number of training observations or examples (when compared to large, real world datasets) and limited computational resources available to most people, results in a model that will not yield good results, no matter how robust or complex the model, leading to inexperienced people making wrong inferences.

One of the other reasons to apply feature selection techniques is to avoid overfitting by increasing more generalisation into the mix, which is essentially reducing the variance. When a model is over-fitted, it will have an unrealistically high validation accuracy, but it will not perform as well enough, when the model is asked to predict on an unseen or test dataset.(Scikit 2018)

With our dataset, we started out with 121 features in the main table and after joining another 5 tables to it, along with engineering a few new features as a result of those joins, we ended up with almost 400 features in total. By throwing all these features into a model and expecting it to train effectively to be able to predict accurately would be impossible as there is just too many unwanted information in there that would do more harm to model’s learning ability, than good. (Scikit 2018)

Python’s sklearn package has a few built-in feature selection techniques that can be imported and be used on datasets as needed:

f\_classif is one of the methods that is used over quantitative or numeric features. It computes the ANOVA F-value of groups based on the ratio of mean of squares. Here, the F-value score is checking to see if there is any significant difference in the means of numerical features that ae grouped up, against the target class. Hence, when dealing with numerical features, it is a good benchmark to detect the best features for classification. (Scikit 2018)

Chi-Square based feature selection technique is more used for categorical or qualitative features. In this statistical test, the idea is to determine the dependency among features. Based on the scores of the chi-squared test of independence, we can set a threshold value to ensure that all features that below a certain cap are filtered out, before feeding the data into the model. The chi-square test is performed against each categorical feature against the target class to check for dependency or the existence of any relationship between the two variables. If the test shows that the target class has no dependency (or is independent) on a categorical feature, is it then subsequently dropped and the test moves on to test the next feature.

mutual\_classif is another technique applied in the feature selection process. This method estimates the mutual information (MI) between two variables and must be non-negative, measuring their dependencies. If the MI value is zero, it indicates that the variables are completely independent and higher the value, represents higher dependency. (Scikit 2018)

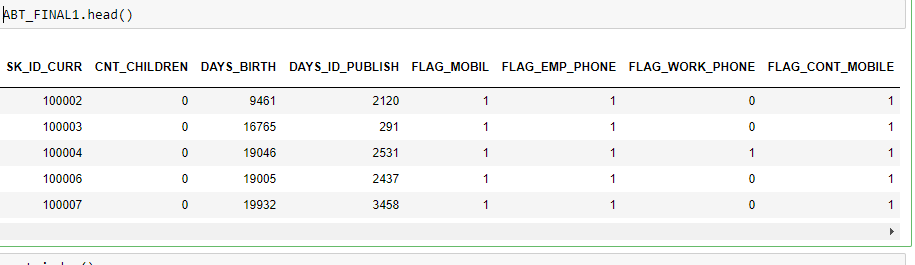
# Results

In this section we will include the best AUROC values for the different models and other ancillary outputs and result which were critical to the project.

**Final ABT:**

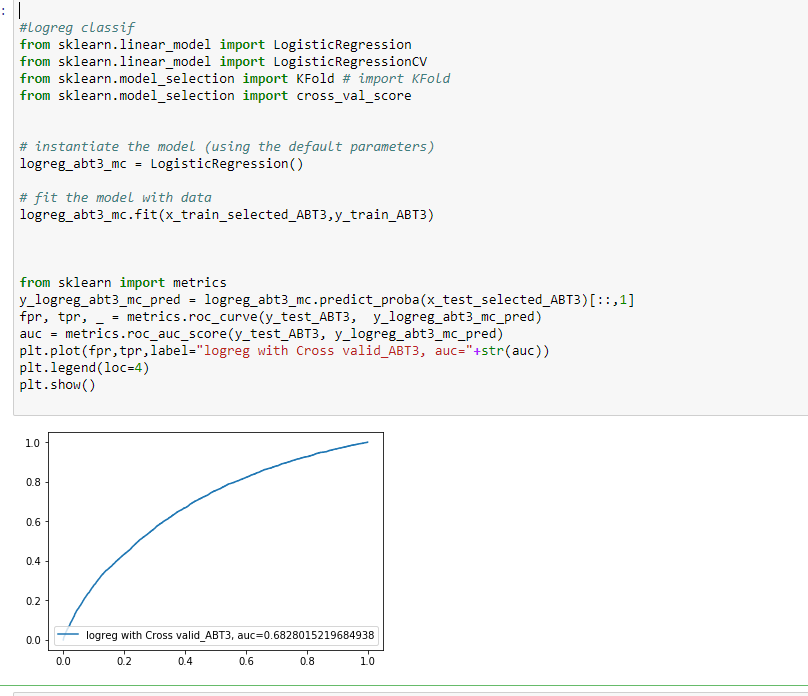
The final analytical base table after feature generation, grouping/joining was as





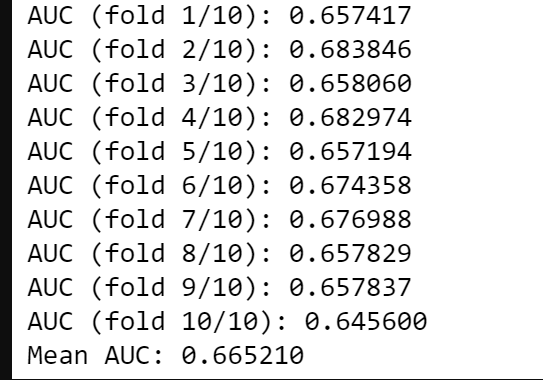
**Fig 6**

**Logistic regression:** The logit model formed our base model and was run using different feature selection, statistical criteria and percentage selection parameters and the best outcome was as below in Fig 7



**Fig 7 Logit best model AUROC 0.6828**

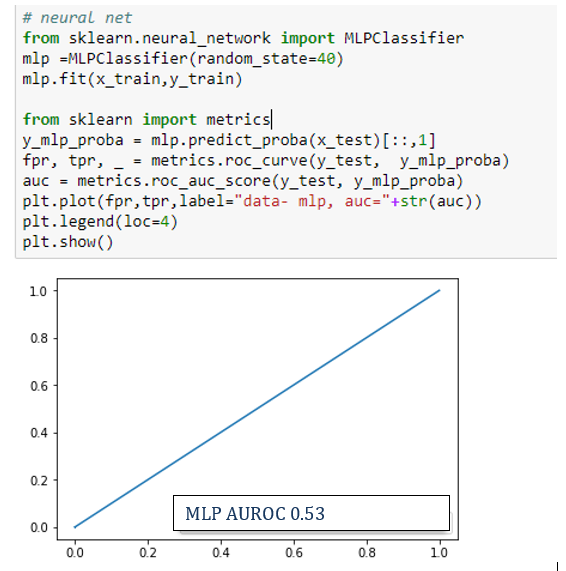
The logit model without Feature selection had a **CV AUROC score of 0.665** as seen from Fig 8



**Fig 8 Logit without Feature selection**

**MLP ANN:**

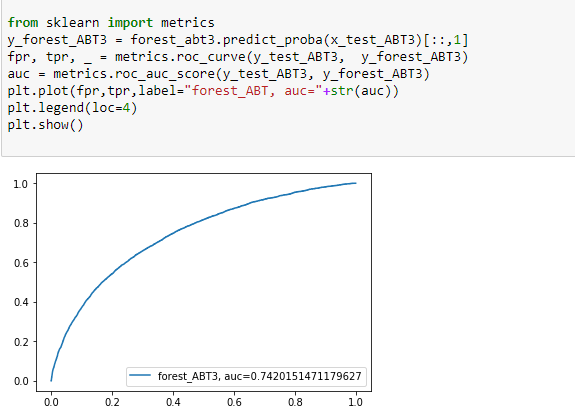
The neural network model which was based on a simple MLP had poor performance and the best score was 0.53



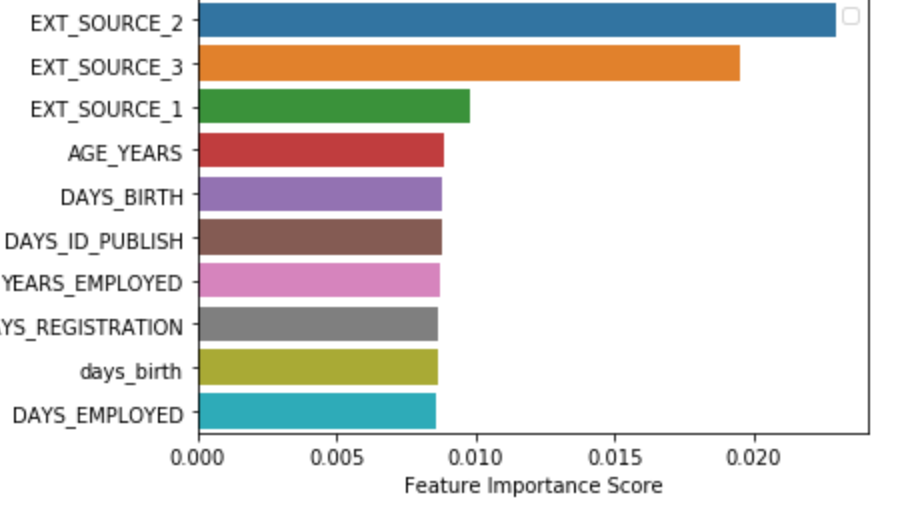
**Fig 9**

**Random Forest:**

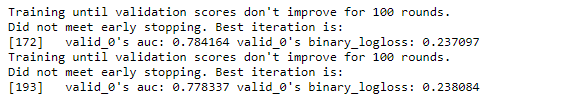
The best AUROC for random forest was 0.74 and



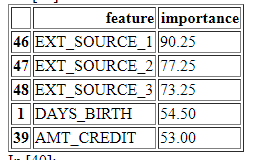
**Selected top features for Random Forest:**



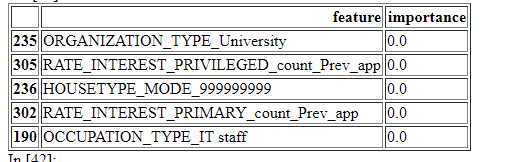
**Light-GBM: From Light GBM we received the best outcome AUROC of 0.784 as below.**

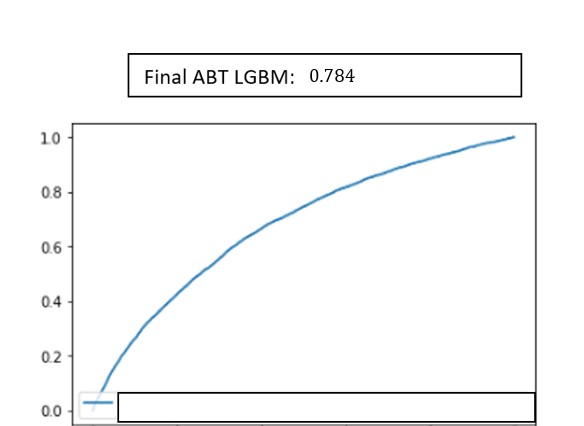


**The most important features from LGBM were:**



**The least important were:**





**Overall model standings:**

|  |  |
| --- | --- |
| **Model** | **Best AUROC** |
| LOGIT | 0.6828 |
| MLP | 0.53 |
| RF | 0.742 |
| LGBM | 0.784 |

# Discussion/recommendation

## Recommended model & model comparison notes:

We recommend Light-GBM as the model of choice since the AUROC of the model is over 4 percent better than the next best which is Random Forest and about 10 percent more than the base model i.e. Logit with 06828.

This is a significant difference and would translate to material amounts in the real world.

**Note on model comparison:**

Most of our time was devoted to improving the main 3 models i.e. the base Logit and RF & LGBM. Due to a limitation in time and computing power the Artificial Neural Network model that was used was basic and only a feed forward model with few hidden layers and no backpropagation. Therefore, the model performance of 0.53 for MLP cannot be considered gospel and would require more research.

## Comment on Feature importance

From the results section the ‘3 ext’ features which were provided by Home Credit and their true origins were kept hidden by the company along with the age and amount of credit featured high on the list and in our own exploratory analysis, these were found to hold sway.

## Limitations of our approach

We feel that more time could have been devoted towards feature engineering and making minor improvements for small but significant gains would have been ideal but owing to resource i.e. computing power and time constraints these aspects remained out of reach and should form basis of any further analysis to be done on the topic.

## Conclusion/Reflection

To conclude we would like to reflect upon the initial discussion about the various models and the feature selection, imputation approaches used in the project and can say that the overall Light-GBM score of 0.784 though reasonable could be improved given more time, computing power and a more in-depth analysis.

# References

Yiming Jiang, Chenguang Yang, Jing Na, Guang Li, Yanan Li, and Junpei Zhong, “A Brief Review of Neural Networks Based Learning and Control and Their Applications for Robots,” Complexity, vol. 2017, Article ID 1895897, 14 pages, 2017. Sourced from <https://doi.org/10.1155/2017/1895897> on 23rd October 28, 2018

Streiner, D. L., & Cairney, J. (2007). What’s under the ROC? An Introduction to Receiver Operating Characteristics Curves. The Canadian Journal of Psychiatry, 52(2), 121–128. <https://doi.org/10.1177/070674370705200210>

Dong G, Lai K K, Chen J, Credit scorecard based on logistic regression with random coefficients, Science Direct, Procedia Computer Science 1 (2012) 2463–2468, 2010

Akbhari S, Mardukhi F, Classification of Bank Customers Using the Random Forest Algorithm, International Journal of Mathematics and Computer Sciences (IJMCS), Vol 29, 2014

Ke et al, LightGBM: A Highly Efficient Gradient Boosting Decision Tree, 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

# Appendix

**Code:**

Import packages & data sets

In [ ]:

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **os**

**from** **sklearn.preprocessing** **import** LabelEncoder

**import** **seaborn** **as** **sb**

**import** **matplotlib.pyplot** **as** **plt**

**from** **sklearn.preprocessing** **import** Imputer

**import** **warnings**

warnings.filterwarnings('ignore')

pd.options.display.max\_columns = **None**

pd.options.display.max\_rows = 500

In [ ]:

*# change the working directory*

**import** **os**

os.chdir('C:/Users/Kumar .LAPTOP-C9MKQM9I/Desktop/MS Analytics and Grad dip math/MS Analytics/Applied Research Project')

In [ ]:

apptrain= pd.read\_csv('application\_train\_raw.csv')

In [ ]:

bureau= pd.read\_csv('bureau.csv')

In [ ]:

bureau\_bal= pd.read\_csv('bureau\_balance.csv')

In [ ]:

prevApp = pd.read\_csv('previous\_application.csv')

In [ ]:

inst\_pay = pd.read\_csv('installments\_payments.csv')

In [ ]:

POS\_CASH\_balance = pd.read\_csv('POS\_CASH\_balance.csv')

In [ ]:

credit\_card\_balance = pd.read\_csv("credit\_card\_balance.csv")*# too many missing values not taken*

Functions etc.

In [ ]:

*# to find missing values of columns in a table*

**def** missingValues(df):

missing\_values = df.isnull().sum()

missing\_values\_percent = 100 \* df.isnull().sum() / len(df)

missing\_values\_table = pd.concat([missing\_values, missing\_values\_percent], axis=1)

missing\_values\_table.columns = ['Count', 'Percentage']

missing\_values\_table['Percentage'] = missing\_values\_table['Percentage'].round(1)

missing\_values\_table\_sorted = missing\_values\_table.sort\_values('Percentage',ascending = **False**)

**return** missing\_values\_table\_sorted

In [ ]:

**def** uniqueCategories(df):

df\_obj = df.select\_dtypes(include='object')

col\_name = []

unique\_categories = []

**for** col **in** df\_obj:

uniqCat = len(list((df\_obj[col].unique())))

col\_name.append(col)

unique\_categories.append(uniqCat)

uniqueCat = pd.DataFrame([col\_name, unique\_categories])

**return** uniqueCat.transpose()

In [ ]:

uniqueCategories(apptrain)

In [ ]:

df\_obj = apptrain.select\_dtypes(include='object')

In [ ]:

*# to get the counts and normalised counts for the grouping column*

**def** cat\_group(df, grouping\_col, dfName):

categorical\_cols = pd.get\_dummies(df.select\_dtypes(include ='object'))

categorical\_cols[grouping\_col] = df[grouping\_col]

categorical\_cols = categorical\_cols.groupby(grouping\_col).agg(['sum', 'mean'])

col\_names = []

**for** col **in** categorical\_cols.columns.levels[0]:

**for** i **in** ['count', 'count\_norm']:

col\_names.append('**%s**\_**%s**\_**%s**' % (dfName, col, i))

categorical\_cols.columns = col\_names

**return** categorical\_cols

In [ ]:

test= apptrain

In [ ]:

test.head()

In [ ]:

missingValues(test)

In [ ]:

*#df.select\_dtypes(include = 'int64').fillna(df.mean(),inplace=True)*

*#test.select\_dtypes(include = 'int64').fillna(test.mean(),inplace=True)*

In [ ]:

*# # to impute based on dtypes*

*# def imputation(df):*

*# for col in df:*

*# if df[col].dtype == 'int64':*

*# df[[col]] = df[[col]].fillna(df.mean())*

*# elif df[col].dtype == 'float64':*

*# df[[col]] = df[[col]].fillna(df.mean())*

*# elif df[col].dtype == 'object':*

*# df[[col]] = df[[col]].fillna(df(value='99999'))#special value*

*# elif df[col].dtype == 'object':*

*# df[[col]] = df[[col]].fillna(df(value='99999'))#special value*

*# return df.head()*

In [ ]:

*#setting XNA to nan*

apptrain.replace('XNA', np.nan, inplace = **True**)

bureau.replace('XNA', np.nan, inplace = **True**)

bureau\_bal.replace('XNA', np.nan, inplace = **True**)

prevApp.replace('XNA', np.nan, inplace = **True**)

inst\_pay.replace('XNA', np.nan, inplace = **True**)

POS\_CASH\_balance.replace('XNA', np.nan, inplace = **True**)

*#credit\_card\_balance.replace('XNA', np.nan, inplace = True)*

In [ ]:

*#setting XAP to nan*

apptrain.replace('XAP', np.nan, inplace = **True**)

bureau.replace('XAP', np.nan, inplace = **True**)

bureau\_bal.replace('XAP', np.nan, inplace = **True**)

prevApp.replace('XAP', np.nan, inplace = **True**)

inst\_pay.replace('XAP', np.nan, inplace = **True**)

POS\_CASH\_balance.replace('XAP', np.nan, inplace = **True**)

*#credit\_card\_balance.replace('XAP', np.nan, inplace = True)*

In [ ]:

*# function to extract name and append aggregate type:*

**def** aggName(df):

origNames = list(df.columns)

names = []

**for** col **in** origNames:

**for** aggType **in** ['count', 'min', 'max', 'mean', 'sum']:

names.append('**%s**\_**%s**' % ( col, aggType))

**return** names

Target

In [ ]:

target = apptrain['TARGET']

SK ID CURR

In [ ]:

SK\_ID\_CURR = apptrain['SK\_ID\_CURR']

CURR\_TARGET

In [ ]:

CURR\_TARGET =pd.DataFrame(zip(SK\_ID\_CURR, target))

Data joining and other pre-processing,visualisations, features etc.

In [ ]:

*# Changing to absolute and addressing age, days employed "365243" values*

apptrain['DAYS\_BIRTH'] = abs(apptrain['DAYS\_BIRTH'])

apptrain['DAYS\_EMPLOYED'] = abs(apptrain['DAYS\_EMPLOYED'])

apptrain['DAYS\_REGISTRATION'] = abs(apptrain['DAYS\_REGISTRATION'])

apptrain['DAYS\_ID\_PUBLISH'] = abs(apptrain['DAYS\_ID\_PUBLISH'])

apptrain['AGE\_YEARS'] = apptrain['DAYS\_BIRTH']/365

apptrain['AGE\_YEARS\_BINS'] = pd.cut(apptrain['AGE\_YEARS'], bins = np.linspace(20, 70, num = 11))

apptrain['DAYS\_EMPLOYED'] = apptrain['DAYS\_EMPLOYED'].astype(np.int64)

np.count\_nonzero(apptrain['DAYS\_EMPLOYED'] == 365243)

apptrain['DAYS\_EMPLOYED'].replace(365243, np.nan, inplace = **True**)

apptrain['DAYS\_EMPLOYED'].replace({np.nan: apptrain['DAYS\_EMPLOYED'].mean()}, inplace = **True**)

print(apptrain['DAYS\_EMPLOYED'].describe()) *# DAYS\_EMPLOYED is now rectified*

apptrain['YEARS\_EMPLOYED'] = apptrain['DAYS\_EMPLOYED']/365

In [ ]:

*# grouping education categories*

apptrain['NAME\_EDUCATION\_TYPE'].value\_counts()

apptrain['NAME\_EDUCATION\_TYPE']=np.where(apptrain['NAME\_EDUCATION\_TYPE'] =='Secondary / secondary special', 'Secondary', apptrain['NAME\_EDUCATION\_TYPE'])

apptrain['NAME\_EDUCATION\_TYPE']=np.where(apptrain['NAME\_EDUCATION\_TYPE'] =='Lower secondary', 'Secondary', apptrain['NAME\_EDUCATION\_TYPE'])

In [ ]:

*#grouping organisation types*

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Business Entity Type 1', 'Business', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Business Entity Type 2', 'Business', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Business Entity Type 3', 'Business', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Industry: type 1', 'Industry', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Industry: type 2', 'Industry', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Industry: type 3', 'Industry', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Industry: type 4', 'Industry', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Industry: type 5', 'Industry', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Industry: type 6', 'Industry', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Industry: type 7', 'Industry', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Industry: type 8', 'Industry', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Industry: type 9', 'Industry', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Industry: type 10', 'Industry', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Industry: type 11', 'Industry', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Industry: type 12', 'Industry', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Industry: type 13', 'Industry', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Trade: type 1', 'Trade', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Trade: type 2', 'Trade', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Trade: type 3', 'Trade', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Trade: type 4', 'Trade', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Trade: type 5', 'Trade', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Trade: type 6', 'Trade', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Trade: type 7', 'Trade', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Transport: type 1', 'Transport', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Transport: type 2', 'Transport', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Transport: type 3', 'Transport', apptrain['ORGANIZATION\_TYPE'])

apptrain['ORGANIZATION\_TYPE']=np.where(apptrain['ORGANIZATION\_TYPE'] =='Transport: type 4', 'Transport', apptrain['ORGANIZATION\_TYPE'])

In [ ]:

missingValues(apptrain)

In [ ]:

*# dropping columns with missing values greater than 60%*

thresh = len(apptrain)\* .4

apptrain.dropna(thresh = thresh, axis = 1, inplace = **True**)

In [ ]:

apptrain.shape

In [ ]:

apptrain.dtypes.value\_counts()

In [ ]:

*#apptrain.dtypes*

Imputation apptrain

In [ ]:

*# imputer for handling missing values*

**from** **sklearn.preprocessing.imputation** **import** Imputer

imputer\_0 = Imputer(strategy = 'mean')

imputer\_1 = Imputer(strategy = 'median')

In [ ]:

*# now that we gave dropped those columns with 60% or greater , we will impute missing values based on mean for float or int dttpes*

*# and special value of 99999 for categorical/object dtypes*

*#imputation(apptrain)*

In [ ]:

apptrain\_object = apptrain.select\_dtypes(include='object')

apptrain\_object = apptrain\_object.replace(np.nan, '99999', regex=**True**)

In [ ]:

apptrain\_int64 = apptrain.select\_dtypes(include='int64')

apptrain\_int64 = apptrain\_int64.fillna(apptrain\_int64.mean())

In [ ]:

apptrain\_float64 = apptrain.select\_dtypes(include='float64')

apptrain\_float64 = apptrain\_float64.fillna(apptrain\_float64.mean())

In [ ]:

apptrain\_imputed = pd.concat([apptrain\_int64, apptrain\_object,apptrain\_float64], axis=1)

In [ ]:

*# dropping target column*

apptrain\_imputed =apptrain\_imputed.drop(['TARGET'], axis=1)

In [ ]:

*# Create a label encoder object*

le = LabelEncoder()

le\_count = 0

**for** col **in** apptrain\_imputed:

**if** apptrain\_imputed[col].dtype == 'object':

**if** len(list(apptrain\_imputed[col].unique())) <= 2:

*# Train on the training data*

le.fit(apptrain\_imputed[col])

*# Transform both training and testing data*

apptrain\_imputed[col] = le.transform(apptrain\_imputed[col])

*# Keep track of how many columns were label encoded*

le\_count += 1

print('**%d** columns were label encoded.' % le\_count)

In [ ]:

apptrain\_imputed.shape

In [ ]:

apptrain\_imputed.head()

In [ ]:

apptrain\_imputed1 = pd.get\_dummies(apptrain\_imputed)

In [ ]:

*#feature creation apptrain: ratios*

apptrain\_imputed1['cred\_inc'] =apptrain\_imputed1['AMT\_CREDIT'] /apptrain\_imputed1['AMT\_INCOME\_TOTAL']

apptrain\_imputed1['annuity\_inc'] =apptrain\_imputed1['AMT\_ANNUITY'] /apptrain\_imputed1['AMT\_INCOME\_TOTAL']

apptrain\_imputed1['cred\_annuity'] =apptrain\_imputed1['AMT\_CREDIT'] /apptrain\_imputed1['AMT\_ANNUITY']

apptrain\_imputed1['days\_birth'] = apptrain\_imputed1['DAYS\_EMPLOYED']/apptrain\_imputed1['DAYS\_BIRTH']

In [ ]:

apptrain\_imputed1.shape

In [ ]:

ABT1 = apptrain\_imputed1

In [ ]:

missingValues(ABT1)

In [ ]:

ABT1.to\_csv('ABT1.csv', sep=',')

In [ ]:

ABT1.shape

Bureau

In [ ]:

missingValues(bureau)

In [ ]:

bureau.shape

In [ ]:

bureau.head()

In [ ]:

uniqueCategories(bureau)

In [ ]:

print(bureau['CREDIT\_ACTIVE'].unique())

print(bureau['CREDIT\_CURRENCY'].unique())

print(bureau['CREDIT\_TYPE'].unique())

In [ ]:

*#everything after mortgage has been clubbed together*

bureau['CREDIT\_TYPE'].value\_counts()

In [ ]:

*#everything after mortgage has been clubbed together*

bureau['CREDIT\_TYPE']=np.where(bureau['CREDIT\_TYPE'] =='Microloan', 'Other\_credit', bureau['CREDIT\_TYPE'])

bureau['CREDIT\_TYPE']=np.where(bureau['CREDIT\_TYPE'] =='Loan for business development', 'Other\_credit', bureau['CREDIT\_TYPE'])

bureau['CREDIT\_TYPE']=np.where(bureau['CREDIT\_TYPE'] =='Another type of loan', 'Other\_credit', bureau['CREDIT\_TYPE'])

bureau['CREDIT\_TYPE']=np.where(bureau['CREDIT\_TYPE'] =='Unknown type of loan', 'Other\_credit', bureau['CREDIT\_TYPE'])

bureau['CREDIT\_TYPE']=np.where(bureau['CREDIT\_TYPE'] =='Loan for working capital replenishment', 'Other\_credit', bureau['CREDIT\_TYPE'])

bureau['CREDIT\_TYPE']=np.where(bureau['CREDIT\_TYPE'] =='Cash loan (non-earmarked)', 'Other\_credit', bureau['CREDIT\_TYPE'])

bureau['CREDIT\_TYPE']=np.where(bureau['CREDIT\_TYPE'] =='Real estate loan', 'Other\_credit', bureau['CREDIT\_TYPE'])

bureau['CREDIT\_TYPE']=np.where(bureau['CREDIT\_TYPE'] =='Loan for the purchase of equipment', 'Other\_credit', bureau['CREDIT\_TYPE'])

bureau['CREDIT\_TYPE']=np.where(bureau['CREDIT\_TYPE'] =='Loan for purchase of shares (margin lending)', 'Other\_credit', bureau['CREDIT\_TYPE'])

bureau['CREDIT\_TYPE']=np.where(bureau['CREDIT\_TYPE'] =='Interbank credit', 'Other\_credit', bureau['CREDIT\_TYPE'])

bureau['CREDIT\_TYPE']=np.where(bureau['CREDIT\_TYPE'] =='Mobile operator loan', 'Other\_credit', bureau['CREDIT\_TYPE'])

In [ ]:

bureau['CREDIT\_TYPE'].value\_counts()

In [ ]:

missingValues(bureau)

In [ ]:

*# over 60% missing dropped & sk id bureau which is just a unique id*

bureau = bureau.drop(columns=['AMT\_ANNUITY','SK\_ID\_BUREAU','AMT\_CREDIT\_MAX\_OVERDUE'])

In [ ]:

bureau.shape

In [ ]:

missingValues(bureau)

In [ ]:

bureau\_int = bureau.select\_dtypes(include= 'int64')

bureau\_float = bureau.select\_dtypes(include= 'float64')

bureau\_object = bureau.select\_dtypes(include= 'object')

In [ ]:

missingValues(bureau\_object)

In [ ]:

bureau\_object['SK\_ID\_CURR'] = bureau\_int['SK\_ID\_CURR']

In [ ]:

bureau\_float.shape

In [ ]:

bureau\_float['SK\_ID\_CURR'] = bureau\_int['SK\_ID\_CURR']

In [ ]:

*#imputation for bureau*

bureau\_object\_1 = bureau\_object.replace(np.nan, '99999', regex=**True**)

bureau\_int\_1 = bureau\_int.fillna(bureau\_int.mean())

bureau\_float\_1 = bureau\_float.fillna(bureau\_float.mean())

In [ ]:

*#bureau\_float64 = bureau.select\_dtypes(include='float64')*

In [ ]:

bureau\_float\_1.head()

In [ ]:

bureau\_float\_agg = bureau\_float\_1.groupby('SK\_ID\_CURR', as\_index = **False**).agg(['count', 'min', 'max', 'mean', 'sum']).reset\_index()

bureau\_int\_agg = bureau\_int\_1.groupby('SK\_ID\_CURR', as\_index = **False**).agg(['count', 'min', 'max', 'mean', 'sum']).reset\_index()

In [ ]:

bureau\_int\_agg.shape

In [ ]:

bureau\_float\_agg.columns = bureau\_float\_agg.columns.droplevel(0)

In [ ]:

bureau\_float\_agg.head()

In [ ]:

bureau\_float\_agg.columns = ['SK\_ID\_CURR','DAYS\_CREDIT\_ENDDATE\_count', 'DAYS\_CREDIT\_ENDDATE\_min', 'DAYS\_CREDIT\_ENDDATE\_max', 'DAYS\_CREDIT\_ENDDATE\_mean', 'DAYS\_CREDIT\_ENDDATE\_sum', 'DAYS\_ENDDATE\_FACT\_count', 'DAYS\_ENDDATE\_FACT\_min', 'DAYS\_ENDDATE\_FACT\_max', 'DAYS\_ENDDATE\_FACT\_mean', 'DAYS\_ENDDATE\_FACT\_sum', 'AMT\_CREDIT\_SUM\_count', 'AMT\_CREDIT\_SUM\_min', 'AMT\_CREDIT\_SUM\_max', 'AMT\_CREDIT\_SUM\_mean', 'AMT\_CREDIT\_SUM\_sum', 'AMT\_CREDIT\_SUM\_DEBT\_count', 'AMT\_CREDIT\_SUM\_DEBT\_min', 'AMT\_CREDIT\_SUM\_DEBT\_max', 'AMT\_CREDIT\_SUM\_DEBT\_mean', 'AMT\_CREDIT\_SUM\_DEBT\_sum', 'AMT\_CREDIT\_SUM\_LIMIT\_count', 'AMT\_CREDIT\_SUM\_LIMIT\_min', 'AMT\_CREDIT\_SUM\_LIMIT\_max', 'AMT\_CREDIT\_SUM\_LIMIT\_mean', 'AMT\_CREDIT\_SUM\_LIMIT\_sum', 'AMT\_CREDIT\_SUM\_OVERDUE\_count', 'AMT\_CREDIT\_SUM\_OVERDUE\_min', 'AMT\_CREDIT\_SUM\_OVERDUE\_max', 'AMT\_CREDIT\_SUM\_OVERDUE\_mean', 'AMT\_CREDIT\_SUM\_OVERDUE\_sum']

In [ ]:

bureau\_float\_agg.head()

In [ ]:

bureau\_int\_agg.columns = bureau\_int\_agg.columns.droplevel(0)

In [ ]:

bureau\_int\_agg.columns = ['SK\_ID\_CURR','DAYS\_CREDIT\_count','DAYS\_CREDIT\_min','DAYS\_CREDIT\_UPDATE\_ENDDATE\_max','DAYS\_ENDDATE\_FACT\_mean','AMT\_CREDIT\_MAX\_OVERDUE\_sum',

'CREDIT\_DAY\_OVERDUE\_count','CREDIT\_DAY\_OVERDUE\_min','CREDIT\_DAY\_OVERDUE\_ENDDATE\_max','CREDIT\_DAY\_OVERDUE\_mean','CREDIT\_DAY\_OVERDUE\_OVERDUE\_sum','CNT\_CREDIT\_PROLONG\_count','CNT\_CREDIT\_PROLONG\_min','CNT\_CREDIT\_PROLONG\_ENDDATE\_max','CNT\_CREDIT\_PROLONG\_mean','CNT\_CREDIT\_PROLONG\_sum',

'DAYS\_CREDIT\_UPDATE\_count','DAYS\_CREDIT\_UPDATE\_min','DAYS\_CREDIT\_UPDATE\_ENDDATE\_max','DAYS\_CREDIT\_UPDATE\_mean','DAYS\_CREDIT\_UPDATE\_sum']

In [ ]:

bureau\_object.shape

In [ ]:

bureau\_object\_ggr=cat\_group(df=bureau\_object, grouping\_col='SK\_ID\_CURR', dfName='bureau\_object').reset\_index()

In [ ]:

bureau\_float\_agg.to\_csv('bureau\_float\_agg.csv')

In [ ]:

bureau\_float\_agg.shape

In [ ]:

ABT2.shape

In [ ]:

ABT2= ABT1.merge(bureau\_float\_agg, on = 'SK\_ID\_CURR', how = 'left')

In [ ]:

ABT2=ABT2.fillna(ABT2.mean())

In [ ]:

ABT2= ABT2.merge(bureau\_int\_agg, on = 'SK\_ID\_CURR', how = 'left')

In [ ]:

missingValues(ABT2)

In [ ]:

ABT2=ABT2.fillna(ABT2.mean())

In [ ]:

ABT2= ABT1.merge(bureau\_object\_ggr, on = 'SK\_ID\_CURR', how = 'left')

In [ ]:

ABT2=ABT2.fillna(ABT2.mean())

In [ ]:

ABT2.to\_csv('ABT2.csv', sep=',')

Bureau Bal

In [ ]:

missingValues(bureau\_bal)

In [ ]:

uniqueCategories(bureau\_bal)

In [ ]:

print(bureau\_bal['STATUS'].unique())

In [ ]:

bureau\_bal.head()

In [ ]:

bureau\_count = pd.read\_csv('bureau.csv')

In [ ]:

bureau1= pd.read\_csv('bureau.csv')

In [ ]:

*#merge bureau, bureau\_balance*

mod\_bureau\_bal = bureau\_bal.merge(bureau1, on = 'SK\_ID\_BUREAU', how = 'left')

In [ ]:

mod\_bureau\_bal.head()

In [ ]:

bureau\_bal\_curr = mod\_bureau\_bal.drop(columns = ['SK\_ID\_BUREAU','CREDIT\_ACTIVE','CREDIT\_CURRENCY','DAYS\_CREDIT','CREDIT\_DAY\_OVERDUE','DAYS\_CREDIT\_ENDDATE', 'DAYS\_ENDDATE\_FACT','AMT\_CREDIT\_MAX\_OVERDUE','CNT\_CREDIT\_PROLONG','AMT\_CREDIT\_SUM','AMT\_CREDIT\_SUM\_DEBT','AMT\_CREDIT\_SUM\_LIMIT','AMT\_CREDIT\_SUM\_OVERDUE','CREDIT\_TYPE','DAYS\_CREDIT\_UPDATE','AMT\_ANNUITY'])

In [ ]:

bureau\_bal\_curr['SK\_ID\_CURR'] = bureau\_bal\_curr['SK\_ID\_CURR'].fillna(0).astype(np.int64)

In [ ]:

bureau\_bal\_curr.head()

In [ ]:

bureau\_\_bal\_object=cat\_group(df=bureau\_bal\_curr, grouping\_col='SK\_ID\_CURR', dfName='bureau\_bal\_curr').reset\_index()

In [ ]:

bureau\_\_bal\_int = bureau\_bal\_curr.groupby('SK\_ID\_CURR', as\_index = **False**).agg(['count', 'min', 'max', 'mean', 'sum']).reset\_index()

In [ ]:

bureau\_\_bal\_object.head()

In [ ]:

bureau\_\_bal\_int.head()

In [ ]:

bureau\_\_bal\_int.columns = bureau\_\_bal\_int.columns.droplevel(0)

In [ ]:

bureau\_\_bal\_int.columns = ['SK\_ID\_CURR','MONTHS\_BALANCE\_count', 'MONTHS\_BALANCE\_min', 'MONTHS\_BALANCE\_max', 'MONTHS\_BALANCE\_mean', 'MONTHS\_BALANCE\_sum']

In [ ]:

ABT3 = ABT2.merge(bureau\_\_bal\_int, on = 'SK\_ID\_CURR', how = 'left')

In [ ]:

missingValues(ABT3)

In [ ]:

ABT3 = ABT3.drop(columns = ['MONTHS\_BALANCE\_sum','MONTHS\_BALANCE\_mean','MONTHS\_BALANCE\_max','MONTHS\_BALANCE\_min','MONTHS\_BALANCE\_count'])

In [ ]:

prev\_loan\_count = bureau\_count.groupby('SK\_ID\_CURR', as\_index = **False**)['SK\_ID\_BUREAU'].count().rename(columns = {'SK\_ID\_BUREAU': 'PREV\_LOAN\_COUNT'})

In [ ]:

prev\_loan\_count.head()

In [ ]:

ABT3 = ABT3.merge(bureau\_\_bal\_object, on = 'SK\_ID\_CURR', how = 'left')

In [ ]:

ABT3 = ABT3.drop(columns = ['bureau\_bal\_curr\_STATUS\_X\_count\_norm','bureau\_bal\_curr\_STATUS\_3\_count\_norm','bureau\_bal\_curr\_STATUS\_0\_count','bureau\_bal\_curr\_STATUS\_0\_count\_norm','bureau\_bal\_curr\_STATUS\_1\_count\_norm','bureau\_bal\_curr\_STATUS\_2\_count','bureau\_bal\_curr\_STATUS\_2\_count\_norm','bureau\_bal\_curr\_STATUS\_3\_count','bureau\_bal\_curr\_STATUS\_1\_count','bureau\_bal\_curr\_STATUS\_4\_count','bureau\_bal\_curr\_STATUS\_5\_count','bureau\_bal\_curr\_STATUS\_5\_count\_norm','bureau\_bal\_curr\_STATUS\_C\_count','bureau\_bal\_curr\_STATUS\_C\_count\_norm','bureau\_bal\_curr\_STATUS\_X\_count','bureau\_bal\_curr\_STATUS\_4\_count\_norm'])

In [ ]:

ABT3.shape

In [ ]:

missingValues(ABT3)

In [ ]:

ABT3.to\_csv('ABT3.csv', sep=',')

Previous Application

In [ ]:

prevApp.shape

In [ ]:

prevApp.head()

In [ ]:

missingValues(prevApp)

In [ ]:

*# if a code has nan for the CODE\_REJECT\_REASON column, it is assumed that the application has not been rejected*

prevApp['CODE\_REJECT\_REASON'].unique()

*# rest above 60% nans dropped & sk id prev*

prev\_app\_dr = prevApp.drop(columns = ['RATE\_INTEREST\_PRIMARY','SK\_ID\_PREV','RATE\_INTEREST\_PRIVILEGED','NAME\_PRODUCT\_TYPE','NAME\_CASH\_LOAN\_PURPOSE'])

In [ ]:

prev\_app\_dr.shape

In [ ]:

uniqueCategories(prev\_app\_dr)

In [ ]:

prev\_app\_dr\_int = prev\_app\_dr.select\_dtypes(include= 'int64')

prev\_app\_dr\_float = prev\_app\_dr.select\_dtypes(include= 'float64')

prev\_app\_dr\_object = prev\_app\_dr.select\_dtypes(include= 'object')

In [ ]:

*#imputation for prev app*

prev\_app\_dr\_object = prev\_app\_dr\_object.replace(np.nan, '99999', regex=**True**)

prev\_app\_dr\_int = prev\_app\_dr\_int.fillna(prev\_app\_dr\_int.mean())

prev\_app\_dr\_float = prev\_app\_dr\_float.fillna(prev\_app\_dr\_float.mean())

In [ ]:

uniqueCategories(prev\_app\_dr\_object)

In [ ]:

prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'].unique()

In [ ]:

prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'].value\_counts()

In [ ]:

*#NAME\_GOODS\_CATEGORY clubbing from auto accessorties onwards into Other\_goods*

prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY']=np.where(prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'] =='Auto Accessorie', 'Other\_goods', prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'])

prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY']=np.where(prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'] =='Jewelry', 'Other\_goods', prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'])

prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY']=np.where(prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'] =='Homewares', 'Other\_goods', prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'])

prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY']=np.where(prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'] =='Medical Supplies', 'Other\_goods', prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'])

prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY']=np.where(prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'] =='Vehicles', 'Other\_goods', prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'])

prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY']=np.where(prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'] =='Sport and Leisure', 'Other\_goods', prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'])

prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY']=np.where(prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'] =='Gardening', 'Other\_goods', prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'])

prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY']=np.where(prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'] =='Other', 'Other\_goods', prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'])

prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY']=np.where(prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'] =='Office Appliances', 'Other\_goods', prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'])

prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY']=np.where(prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'] =='Medicine', 'Other\_goods', prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'])

prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY']=np.where(prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'] =='Tourism', 'Other\_goods', prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'])

prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY']=np.where(prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'] =='Direct Sales', 'Other\_goods', prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'])

prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY']=np.where(prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'] =='Fitness', 'Other\_goods', prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'])

prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY']=np.where(prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'] =='Additional Service', 'Other\_goods', prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'])

prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY']=np.where(prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'] =='Education', 'Other\_goods', prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'])

prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY']=np.where(prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'] =='Weapon', 'Other\_goods', prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'])

prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY']=np.where(prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'] =='Insurance', 'Other\_goods', prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'])

prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY']=np.where(prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'] =='Animals', 'Other\_goods', prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'])

prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY']=np.where(prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'] =='House Construction', 'Other\_goods', prev\_app\_dr\_object['NAME\_GOODS\_CATEGORY'])

In [ ]:

prev\_app\_dr\_object['PRODUCT\_COMBINATION'].value\_counts()

In [ ]:

prev\_app\_dr\_object['NAME\_SELLER\_INDUSTRY'].value\_counts()

In [ ]:

prev\_app\_dr\_float.dtypes

In [ ]:

prev\_app\_dr\_float['SK\_ID\_CURR'] = prev\_app\_dr['SK\_ID\_CURR']

In [ ]:

prev\_app\_dr['SK\_ID\_CURR'].nunique()

In [ ]:

prev\_app\_dr\_object['SK\_ID\_CURR'] = prev\_app\_dr['SK\_ID\_CURR']

In [ ]:

prev\_app\_dr\_float['SK\_ID\_CURR'].nunique()

In [ ]:

prev\_app\_float = prev\_app\_dr\_float.groupby('SK\_ID\_CURR', as\_index = **False**).agg(['count', 'min', 'max', 'mean', 'sum']).reset\_index()

prev\_app\_int = prev\_app\_dr\_int.groupby('SK\_ID\_CURR', as\_index = **False**).agg(['count', 'min', 'max', 'mean', 'sum']).reset\_index()

In [ ]:

prev\_app\_float.head()

In [ ]:

origNames = ['AMT\_ANNUITY', 'AMT\_APPLICATION','AMT\_CREDIT','AMT\_DOWN\_PAYMENT','AMT\_GOODS\_PRICE','RATE\_DOWN\_PAYMENT','CNT\_PAYMENT','DAYS\_FIRST\_DRAWING','DAYS\_FIRST\_DUE','DAYS\_LAST\_DUE\_1ST\_VERSION','DAYS\_LAST\_DUE','DAYS\_TERMINATION','NFLAG\_INSURED\_ON\_APPROVAL']

names = []

**for** col **in** origNames:

**for** aggType **in** ['count', 'min', 'max', 'mean', 'sum']:

names.append('**%s**\_**%s**' % ( col, aggType))

print(names)

In [ ]:

prev\_app\_float.columns = prev\_app\_float.columns.droplevel(0)

In [ ]:

prev\_app\_float.columns = ['SK\_ID\_CURR','AMT\_ANNUITY\_count', 'AMT\_ANNUITY\_min', 'AMT\_ANNUITY\_max', 'AMT\_ANNUITY\_mean', 'AMT\_ANNUITY\_sum', 'AMT\_APPLICATION\_count', 'AMT\_APPLICATION\_min', 'AMT\_APPLICATION\_max', 'AMT\_APPLICATION\_mean', 'AMT\_APPLICATION\_sum', 'AMT\_CREDIT\_count', 'AMT\_CREDIT\_min', 'AMT\_CREDIT\_max', 'AMT\_CREDIT\_mean', 'AMT\_CREDIT\_sum', 'AMT\_DOWN\_PAYMENT\_count', 'AMT\_DOWN\_PAYMENT\_min', 'AMT\_DOWN\_PAYMENT\_max', 'AMT\_DOWN\_PAYMENT\_mean', 'AMT\_DOWN\_PAYMENT\_sum', 'AMT\_GOODS\_PRICE\_count', 'AMT\_GOODS\_PRICE\_min', 'AMT\_GOODS\_PRICE\_max', 'AMT\_GOODS\_PRICE\_mean', 'AMT\_GOODS\_PRICE\_sum', 'RATE\_DOWN\_PAYMENT\_count', 'RATE\_DOWN\_PAYMENT\_min', 'RATE\_DOWN\_PAYMENT\_max', 'RATE\_DOWN\_PAYMENT\_mean', 'RATE\_DOWN\_PAYMENT\_sum', 'CNT\_PAYMENT\_count', 'CNT\_PAYMENT\_min', 'CNT\_PAYMENT\_max', 'CNT\_PAYMENT\_mean', 'CNT\_PAYMENT\_sum', 'DAYS\_FIRST\_DRAWING\_count', 'DAYS\_FIRST\_DRAWING\_min', 'DAYS\_FIRST\_DRAWING\_max', 'DAYS\_FIRST\_DRAWING\_mean', 'DAYS\_FIRST\_DRAWING\_sum', 'DAYS\_FIRST\_DUE\_count', 'DAYS\_FIRST\_DUE\_min', 'DAYS\_FIRST\_DUE\_max', 'DAYS\_FIRST\_DUE\_mean', 'DAYS\_FIRST\_DUE\_sum', 'DAYS\_LAST\_DUE\_1ST\_VERSION\_count', 'DAYS\_LAST\_DUE\_1ST\_VERSION\_min', 'DAYS\_LAST\_DUE\_1ST\_VERSION\_max', 'DAYS\_LAST\_DUE\_1ST\_VERSION\_mean', 'DAYS\_LAST\_DUE\_1ST\_VERSION\_sum', 'DAYS\_LAST\_DUE\_count', 'DAYS\_LAST\_DUE\_min', 'DAYS\_LAST\_DUE\_max', 'DAYS\_LAST\_DUE\_mean', 'DAYS\_LAST\_DUE\_sum', 'DAYS\_TERMINATION\_count', 'DAYS\_TERMINATION\_min', 'DAYS\_TERMINATION\_max', 'DAYS\_TERMINATION\_mean', 'DAYS\_TERMINATION\_sum', 'NFLAG\_INSURED\_ON\_APPROVAL\_count', 'NFLAG\_INSURED\_ON\_APPROVAL\_min', 'NFLAG\_INSURED\_ON\_APPROVAL\_max', 'NFLAG\_INSURED\_ON\_APPROVAL\_mean', 'NFLAG\_INSURED\_ON\_APPROVAL\_sum']

In [ ]:

prev\_app\_int.head()

In [ ]:

origNames = ['HOUR\_APPR\_PROCESS\_START', 'NFLAG\_LAST\_APPL\_IN\_DAY','DAYS\_DECISION','SELLERPLACE\_AREA']

names = []

**for** col **in** origNames:

**for** aggType **in** ['count', 'min', 'max', 'mean', 'sum']:

names.append('**%s**\_**%s**' % ( col, aggType))

print(names)

In [ ]:

prev\_app\_int.columns = prev\_app\_int.columns.droplevel(0)

In [ ]:

prev\_app\_int.columns =['SK\_ID\_CURR','HOUR\_APPR\_PROCESS\_START\_count', 'HOUR\_APPR\_PROCESS\_START\_min', 'HOUR\_APPR\_PROCESS\_START\_max', 'HOUR\_APPR\_PROCESS\_START\_mean', 'HOUR\_APPR\_PROCESS\_START\_sum', 'NFLAG\_LAST\_APPL\_IN\_DAY\_count', 'NFLAG\_LAST\_APPL\_IN\_DAY\_min', 'NFLAG\_LAST\_APPL\_IN\_DAY\_max', 'NFLAG\_LAST\_APPL\_IN\_DAY\_mean', 'NFLAG\_LAST\_APPL\_IN\_DAY\_sum', 'DAYS\_DECISION\_count', 'DAYS\_DECISION\_min', 'DAYS\_DECISION\_max', 'DAYS\_DECISION\_mean', 'DAYS\_DECISION\_sum', 'SELLERPLACE\_AREA\_count', 'SELLERPLACE\_AREA\_min', 'SELLERPLACE\_AREA\_max', 'SELLERPLACE\_AREA\_mean', 'SELLERPLACE\_AREA\_sum']

In [ ]:

prev\_app\_dr\_object\_aggr =cat\_group(df=prev\_app\_dr\_object, grouping\_col='SK\_ID\_CURR', dfName='prev\_app\_dr\_object').reset\_index()

In [ ]:

prev\_app\_dr\_object\_aggr.shape

In [ ]:

ABT3.shape

In [ ]:

ABT4 = ABT3.merge(prev\_app\_dr\_object\_aggr, on = 'SK\_ID\_CURR', how = 'left')

In [ ]:

ABT4 = ABT4.merge(prev\_app\_int, on = 'SK\_ID\_CURR', how = 'left')

In [ ]:

ABT4 = ABT3.merge(prev\_app\_float, on = 'SK\_ID\_CURR', how = 'left')

In [ ]:

ABT4.shape

In [ ]:

missingValues(ABT4)

In [ ]:

ABT4 = ABT4.fillna(ABT4.mean())

In [ ]:

ABT4.to\_csv('ABT4.csv', sep=',')

Instalment payments

In [ ]:

inst\_pay.head()

In [ ]:

inst\_pay.shape

In [ ]:

missingValues(inst\_pay)

In [ ]:

inst\_pay.dtypes

In [ ]:

inst\_pay['NET\_DAYS\_PAID'] = inst\_pay['DAYS\_INSTALMENT'] - inst\_pay['DAYS\_ENTRY\_PAYMENT']

inst\_pay['NET\_AMT\_PAID'] = inst\_pay['AMT\_INSTALMENT'] - inst\_pay['AMT\_PAYMENT']

In [ ]:

inst\_pay['NET\_DAYS\_PAID\_INDICATOR'] =np.where(inst\_pay['NET\_DAYS\_PAID']>0, **True**, **False**)

In [ ]:

*# anything greater than 10 to avoid including trivial amounts*

inst\_pay['NET\_AMT\_PAID\_INDICATOR'] = np.where(inst\_pay['NET\_AMT\_PAID']>10, **True**, **False**)

In [ ]:

inst\_pay['NET\_DAYS\_PAID\_INDICATOR'].replace(**False**, 0, inplace=**True**)

In [ ]:

inst\_pay['NET\_AMT\_PAID\_INDICATOR'].replace(**False**, 0, inplace=**True**)

In [ ]:

inst\_pay.head()

In [ ]:

*# SK ID PREV DROPPED*

inst\_pay = inst\_pay.drop(columns=['SK\_ID\_PREV'])

In [ ]:

inst\_pay\_int = inst\_pay.select\_dtypes(include= 'int64')

inst\_pay\_float = inst\_pay.select\_dtypes(include= 'float64')

In [ ]:

inst\_pay\_float['SK\_ID\_CURR'] = inst\_pay['SK\_ID\_CURR']

In [ ]:

inst\_pay\_float = inst\_pay\_float.fillna(inst\_pay\_float.mean())

In [ ]:

inst\_pay\_float.head()

In [ ]:

inst\_pay\_float\_agg = inst\_pay\_float.groupby('SK\_ID\_CURR', as\_index = **False**).agg(['count', 'min', 'max', 'mean', 'sum']).reset\_index()

In [ ]:

inst\_pay\_int\_agg = inst\_pay\_int.groupby('SK\_ID\_CURR', as\_index = **False**).agg(['count', 'min', 'max', 'mean', 'sum']).reset\_index()

In [ ]:

inst\_pay\_float\_agg.columns = inst\_pay\_float\_agg.columns.droplevel(0)

In [ ]:

inst\_pay\_float\_agg.head()

In [ ]:

origNames = ['NUM\_INSTALMENT\_VERSION', 'DAYS\_INSTALMENT','DAYS\_ENTRY\_PAYMENT','AMT\_INSTALMENT','AMT\_PAYMENT','NET\_DAYS\_PAID','NET\_AMT\_PAID','NET\_DAYS\_PAID\_INDICATOR','NET\_AMT\_PAID\_INDICATOR']

names = []

**for** col **in** origNames:

**for** aggType **in** ['count', 'min', 'max', 'mean', 'sum']:

names.append('**%s**\_**%s**' % ( col, aggType))

print(names)

In [ ]:

inst\_pay\_float\_agg.columns = ['SK\_ID\_CURR','NUM\_INSTALMENT\_VERSION\_count', 'NUM\_INSTALMENT\_VERSION\_min', 'NUM\_INSTALMENT\_VERSION\_max', 'NUM\_INSTALMENT\_VERSION\_mean', 'NUM\_INSTALMENT\_VERSION\_sum', 'DAYS\_INSTALMENT\_count', 'DAYS\_INSTALMENT\_min', 'DAYS\_INSTALMENT\_max', 'DAYS\_INSTALMENT\_mean', 'DAYS\_INSTALMENT\_sum', 'DAYS\_ENTRY\_PAYMENT\_count', 'DAYS\_ENTRY\_PAYMENT\_min', 'DAYS\_ENTRY\_PAYMENT\_max', 'DAYS\_ENTRY\_PAYMENT\_mean', 'DAYS\_ENTRY\_PAYMENT\_sum', 'AMT\_INSTALMENT\_count', 'AMT\_INSTALMENT\_min', 'AMT\_INSTALMENT\_max', 'AMT\_INSTALMENT\_mean', 'AMT\_INSTALMENT\_sum', 'AMT\_PAYMENT\_count', 'AMT\_PAYMENT\_min', 'AMT\_PAYMENT\_max', 'AMT\_PAYMENT\_mean', 'AMT\_PAYMENT\_sum', 'NET\_DAYS\_PAID\_count', 'NET\_DAYS\_PAID\_min', 'NET\_DAYS\_PAID\_max', 'NET\_DAYS\_PAID\_mean', 'NET\_DAYS\_PAID\_sum', 'NET\_AMT\_PAID\_count', 'NET\_AMT\_PAID\_min', 'NET\_AMT\_PAID\_max', 'NET\_AMT\_PAID\_mean', 'NET\_AMT\_PAID\_sum', 'NET\_DAYS\_PAID\_INDICATOR\_count', 'NET\_DAYS\_PAID\_INDICATOR\_min', 'NET\_DAYS\_PAID\_INDICATOR\_max', 'NET\_DAYS\_PAID\_INDICATOR\_mean', 'NET\_DAYS\_PAID\_INDICATOR\_sum', 'NET\_AMT\_PAID\_INDICATOR\_count', 'NET\_AMT\_PAID\_INDICATOR\_min', 'NET\_AMT\_PAID\_INDICATOR\_max', 'NET\_AMT\_PAID\_INDICATOR\_mean', 'NET\_AMT\_PAID\_INDICATOR\_sum']

In [ ]:

inst\_pay\_int\_agg = inst\_pay\_int.groupby('SK\_ID\_CURR', as\_index = **False**).agg(['count', 'min', 'max', 'mean', 'sum']).reset\_index()

In [ ]:

inst\_pay\_int\_agg.head()

In [ ]:

inst\_pay\_int\_agg.columns = inst\_pay\_int\_agg.columns.droplevel(0)

In [ ]:

origNames = ['NUM\_INSTALMENT\_NUMBER']

names = []

**for** col **in** origNames:

**for** aggType **in** ['count', 'min', 'max', 'mean', 'sum']:

names.append('**%s**\_**%s**' % ( col, aggType))

print(names)

In [ ]:

inst\_pay\_int\_agg.columns = ['SK\_ID\_CURR','NUM\_INSTALMENT\_NUMBER\_count', 'NUM\_INSTALMENT\_NUMBER\_min', 'NUM\_INSTALMENT\_NUMBER\_max', 'NUM\_INSTALMENT\_NUMBER\_mean', 'NUM\_INSTALMENT\_NUMBER\_sum']

In [ ]:

ABT5 = ABT4.merge(inst\_pay\_int\_agg, on = 'SK\_ID\_CURR', how = 'left')

In [ ]:

ABT5 =ABT5.merge(inst\_pay\_float\_agg, on = 'SK\_ID\_CURR', how = 'left')

In [ ]:

ABT5.shape

In [ ]:

missingValues(ABT5)

In [ ]:

ABT5=ABT5.fillna(ABT5.mean())

In [ ]:

ABT5.to\_csv('ABT5.csv', sep=',')

POS\_cash\_balance

In [ ]:

POS\_CASH\_balance.head()

In [ ]:

POS\_CASH\_balance.shape

In [ ]:

missingValues(POS\_CASH\_balance)

In [ ]:

POS\_CASH\_balance = POS\_CASH\_balance.drop(columns=['SK\_ID\_PREV'])

In [ ]:

POS\_CASH\_balance.dtypes

In [ ]:

uniqueCategories(POS\_CASH\_balance)

In [ ]:

POS\_CASH\_balance['NAME\_CONTRACT\_STATUS'].value\_counts()

In [ ]:

POS\_CASH\_balance\_int = POS\_CASH\_balance.select\_dtypes(include= 'int64')

POS\_CASH\_balance\_float = POS\_CASH\_balance.select\_dtypes(include= 'float64')

POS\_CASH\_balance\_object = POS\_CASH\_balance.select\_dtypes(include= 'object')

In [ ]:

POS\_CASH\_balance\_float['SK\_ID\_CURR'] = POS\_CASH\_balance['SK\_ID\_CURR']

In [ ]:

POS\_CASH\_balance\_object['SK\_ID\_CURR'] = POS\_CASH\_balance['SK\_ID\_CURR']

In [ ]:

POS\_CASH\_balance\_float\_ = POS\_CASH\_balance\_float.fillna(POS\_CASH\_balance\_float.mean())

In [ ]:

POS\_CASH\_balance\_object\_ = POS\_CASH\_balance\_object.replace(np.nan, '99999', regex=**True**)

In [ ]:

missingValues(POS\_CASH\_balance\_float\_)

In [ ]:

POS\_CASH\_balance\_float\_['FUTURE\_CNT\_INST\_RATIO'] = POS\_CASH\_balance\_float\_['CNT\_INSTALMENT\_FUTURE']/POS\_CASH\_balance\_float\_['CNT\_INSTALMENT']

In [ ]:

POS\_CASH\_balance\_float\_.head()

In [ ]:

POS\_CASH\_balance\_float\_agg = POS\_CASH\_balance\_float\_.groupby('SK\_ID\_CURR', as\_index = **False**).agg(['count', 'min', 'max', 'mean', 'sum']).reset\_index()

POS\_CASH\_balance\_int\_agg = POS\_CASH\_balance\_int.groupby('SK\_ID\_CURR', as\_index = **False**).agg(['count', 'min', 'max', 'mean', 'sum']).reset\_index()

In [ ]:

POS\_CASH\_balance\_object\_agg=cat\_group(df=POS\_CASH\_balance\_object\_, grouping\_col='SK\_ID\_CURR', dfName='POS\_CASH\_balance\_object\_').reset\_index()

In [ ]:

POS\_CASH\_balance\_object\_agg.head()

In [ ]:

POS\_CASH\_balance\_float\_agg.head()

In [ ]:

POS\_CASH\_balance\_float\_agg.columns = POS\_CASH\_balance\_float\_agg.columns.droplevel(0)

In [ ]:

origNames = ['CNT\_INSTALMENT', 'CNT\_INSTALMENT\_FUTURE','FUTURE\_CNT\_INST\_RATIO']

names = []

**for** col **in** origNames:

**for** aggType **in** ['count', 'min', 'max', 'mean', 'sum']:

names.append('**%s**\_**%s**' % ( col, aggType))

print(names)

In [ ]:

POS\_CASH\_balance\_float\_agg.columns = ['SK\_ID\_CURR','CNT\_INSTALMENT\_count', 'CNT\_INSTALMENT\_min', 'CNT\_INSTALMENT\_max', 'CNT\_INSTALMENT\_mean', 'CNT\_INSTALMENT\_sum', 'CNT\_INSTALMENT\_FUTURE\_count', 'CNT\_INSTALMENT\_FUTURE\_min', 'CNT\_INSTALMENT\_FUTURE\_max', 'CNT\_INSTALMENT\_FUTURE\_mean', 'CNT\_INSTALMENT\_FUTURE\_sum', 'FUTURE\_CNT\_INST\_RATIO\_count', 'FUTURE\_CNT\_INST\_RATIO\_min', 'FUTURE\_CNT\_INST\_RATIO\_max', 'FUTURE\_CNT\_INST\_RATIO\_mean', 'FUTURE\_CNT\_INST\_RATIO\_sum']

In [ ]:

POS\_CASH\_balance\_float\_agg.shape

In [ ]:

POS\_CASH\_balance\_int\_agg.head()

In [ ]:

origNames = ['MONTHS\_BALANCE', 'SK\_DPD','SK\_DPD\_DEF']

names = []

**for** col **in** origNames:

**for** aggType **in** ['count', 'min', 'max', 'mean', 'sum']:

names.append('**%s**\_**%s**' % ( col, aggType))

print(names)

In [ ]:

POS\_CASH\_balance\_int\_agg.columns = POS\_CASH\_balance\_int\_agg.columns.droplevel(0)

In [ ]:

POS\_CASH\_balance\_int\_agg.columns = ['SK\_ID\_CURR','MONTHS\_BALANCE\_count', 'MONTHS\_BALANCE\_min', 'MONTHS\_BALANCE\_max', 'MONTHS\_BALANCE\_mean', 'MONTHS\_BALANCE\_sum', 'SK\_DPD\_count', 'SK\_DPD\_min', 'SK\_DPD\_max', 'SK\_DPD\_mean', 'SK\_DPD\_sum', 'SK\_DPD\_DEF\_count', 'SK\_DPD\_DEF\_min', 'SK\_DPD\_DEF\_max', 'SK\_DPD\_DEF\_mean', 'SK\_DPD\_DEF\_sum']

In [ ]:

ABT5.shape

In [ ]:

interim = POS\_CASH\_balance\_int\_agg.merge(POS\_CASH\_balance\_object\_agg, on = 'SK\_ID\_CURR', how = 'left')

In [ ]:

interim = interim.merge(POS\_CASH\_balance\_float\_agg, on = 'SK\_ID\_CURR', how = 'left')

In [ ]:

interim.shape

In [ ]:

target.shape

In [ ]:

SK\_ID =pd.DataFrame(zip(SK\_ID\_CURR, target))

In [ ]:

SK\_ID.shape

In [ ]:

SK\_ID.columns = ['SK\_ID\_CURR','Tar']

In [ ]:

SK\_ID.head()

In [ ]:

interim\_abt6 = SK\_ID.merge(interim, on = 'SK\_ID\_CURR', how = 'left')

In [ ]:

interim\_abt6 = interim\_abt6.drop(columns = ['Tar'])

In [ ]:

interim\_abt6.head()

In [ ]:

ABT5.head()

In [ ]:

print(ABT5.shape)

print(interim\_abt6.shape)

In [ ]:

ABT5\_FIN = ABT5.copy()

In [ ]:

INTERIM\_FIN = pd.read\_csv("interim\_abt6.csv")

In [ ]:

ABT\_FINAL = ABT5\_FIN.merge(INTERIM\_FIN, on = 'SK\_ID\_CURR', how = 'left')

In [ ]:

ABT\_FINAL.shape

In [ ]:

missingValues(ABT\_FINAL)

In [ ]:

ABT\_FINAL= ABT\_FINAL.fillna(ABT\_FINAL.mean())

In [ ]:

ABT\_FINAL.to\_csv('ABT\_FINAL.csv', sep=',')

In [ ]:

ABT\_FINAL1 = pd.read\_csv('ABT\_FINAL.csv', sep=',')

In [ ]:

ABT\_FINAL1.reset\_index()

In [ ]:

ABT\_FINAL1.head()

In [ ]:

ABT\_FINAL\_1=ABT\_FINAL1.drop(ABT\_FINAL1.columns[0], axis=1)

In [ ]:

ABT\_FINAL\_MODELS = ABT\_FINAL\_1.drop(columns = ['SK\_ID\_CURR'])

In [ ]:

ABT\_FINAL\_MODELS.head()

In [ ]:

*#logreg w CROSS VALIDATION*

**from** **sklearn.linear\_model** **import** LogisticRegression

**from** **sklearn.linear\_model** **import** LogisticRegressionCV

**from** **sklearn.model\_selection** **import** KFold *# import KFold*

**from** **sklearn.model\_selection** **import** cross\_val\_score

**from** **sklearn** **import** metrics

*# Evaluate the model using 5-fold cross-validation*

clf=LogisticRegression(C=1.0, class\_weight=**None**, dual=**False**, fit\_intercept=**True**,

intercept\_scaling=1, penalty='l2', random\_state=**None**, tol=0.0001)

*#trainXEncoded = encoder.transform(trainX) # Returns a sparse matrix (see numpy.sparse)*

SEED=42

**from** **sklearn** **import** (metrics, cross\_validation, linear\_model, preprocessing)

mean\_auc = 0.0

n = 5 *# repeat the CV procedure 10 times to get more precise results*

**for** i **in** range(n):

*# for each iteration, randomly hold out 20% of the data as CV set*

X\_train, X\_cv, y\_train, y\_cv = cross\_validation.train\_test\_split(

ABT\_FINAL\_MODELS,target, test\_size=.20, random\_state=i\*SEED)

*# train model and make predictions*

clf.fit(X\_train, y\_train)

preds = clf.predict\_proba(X\_cv)[:, 1]

*# compute AUC metric for this CV fold*

fpr, tpr, thresholds = metrics.roc\_curve(y\_cv, preds)

roc\_auc = metrics.auc(fpr, tpr)

print("AUC (fold **%d**/**%d**): **%f**" % (i + 1, n, roc\_auc))

mean\_auc += roc\_auc

print("Mean AUC: **%f**" % (mean\_auc/n))

In [ ]:

**from** **sklearn.model\_selection** **import** train\_test\_split

x\_train, x\_test\_ABT3, y\_train\_ABT3, y\_test\_ABT3 = train\_test\_split(ABT3\_1, Target\_to\_use, test\_size = 0.20, stratify = Target\_to\_use)

In [ ]:

**from** **sklearn.feature\_selection** **import** SelectPercentile *# using default f\_classif, percentile 55*

select = SelectPercentile(percentile = 55)

select.fit(ABT\_FINAL\_MODELS, target)

x\_train\_selected = select.transform(x\_train\_sel)

x\_test\_selected = select.transform(x\_test\_sel)

print('x\_train\_sel.shape is: **{}**'.format(x\_train\_ABT3.shape))a

print('x\_train\_selected.shape is: **{}**'.format(x\_train\_selected\_ABT3.shape))

print('x\_test\_ABT3.shape is: **{}**'.format(x\_test\_ABT3.shape))

print('x\_test\_selected.shape is: **{}**'.format(x\_test\_selected\_ABT3.shape))

In [ ]:

*#logreg w CROSS VALIDATION + feature selection*

**from** **sklearn.linear\_model** **import** LogisticRegression

**from** **sklearn.linear\_model** **import** LogisticRegressionCV

**from** **sklearn.model\_selection** **import** KFold *# import KFold*

**from** **sklearn.model\_selection** **import** cross\_val\_score

**from** **sklearn** **import** metrics

**from** **sklearn.feature\_selection** **import** SelectPercentile *# using default f\_classif, percentile 55*

select = SelectPercentile(percentile = 55)

select.fit(ABT\_FINAL\_MODELS, target)

x\_train\_selected = select.transform(x\_train\_sel)

x\_test\_selected = select.transform(x\_test\_sel)

*# Evaluate the model using 5-fold cross-validation*

clf=LogisticRegression(C=1.0, class\_weight=**None**, dual=**False**, fit\_intercept=**True**,

intercept\_scaling=1, penalty='l2', random\_state=**None**, tol=0.0001)

*#trainXEncoded = encoder.transform(trainX) # Returns a sparse matrix (see numpy.sparse)*

SEED=42

**from** **sklearn** **import** (metrics, cross\_validation, linear\_model, preprocessing)

mean\_auc = 0.0

n = 5 *# repeat the CV procedure 10 times to get more precise results*

**for** i **in** range(n):

*# for each iteration, randomly hold out 20% of the data as CV set*

*# X\_train, X\_cv, y\_train, y\_cv = cross\_validation.train\_test\_split(*

*#ABT\_FINAL\_MODELS,target, test\_size=.20, random\_state=i\*SEED)*

select = SelectPercentile(percentile = 55)

select.fit(ABT\_FINAL\_MODELS, target)

x\_train\_selected = select.transform(x\_train\_sel)

x\_test\_selected = select.transform(x\_test\_sel)

*# train model and make predictions*

clf.fit(X\_train, y\_train)

preds = clf.predict\_proba(X\_cv)[:, 1]

*# compute AUC metric for this CV fold*

fpr, tpr, thresholds = metrics.roc\_curve(y\_cv, preds)

roc\_auc = metrics.auc(fpr, tpr)

print("AUC (fold **%d**/**%d**): **%f**" % (i + 1, n, roc\_auc))

mean\_auc += roc\_auc

print("Mean AUC: **%f**" % (mean\_auc/n))

In [ ]:

len(target)

In [ ]:

*#without cross val*

**from** **sklearn.model\_selection** **import** train\_test\_split

x\_train\_ABT, x\_test\_ABT, y\_train\_ABT, y\_test\_ABT = train\_test\_split(ABT\_FINAL\_MODELS, target, test\_size = 0.20, stratify = target)

In [ ]:

**from** **sklearn.feature\_selection** **import** SelectPercentile *# using default f\_classif, percentile 70*

select = SelectPercentile(percentile = 30)

select.fit(x\_train\_ABT3, y\_train\_ABT3)

x\_train\_selected\_ABT3 = select.transform(x\_train\_ABT3)

x\_test\_selected\_ABT3 = select.transform(x\_test\_ABT3)

print('x\_train\_ABT3.shape is: **{}**'.format(x\_train\_ABT3.shape))

print('x\_train\_selected.shape is: **{}**'.format(x\_train\_selected\_ABT3.shape))

print('x\_test\_ABT3.shape is: **{}**'.format(x\_test\_ABT3.shape))

print('x\_test\_selected.shape is: **{}**'.format(x\_test\_selected\_ABT3.shape))

In [ ]:

*# mutual\_info\_classif- .6304*

**from** **sklearn.feature\_selection** **import** SelectPercentile

**from** **sklearn.feature\_selection** **import** mutual\_info\_classif

select = SelectPercentile(percentile = 55)

select.fit(x\_train\_ABT3, y\_train\_ABT3)

x\_train\_selected\_ABT3 = select.transform(x\_train\_ABT3)

x\_test\_selected\_ABT3 = select.transform(x\_test\_ABT3)

print('x\_train\_ABT3.shape is: **{}**'.format(x\_train\_ABT3.shape))

print('x\_train\_selected.shape is: **{}**'.format(x\_train\_selected\_ABT3.shape))

print('x\_test\_ABT3.shape is: **{}**'.format(x\_test\_ABT3.shape))

print('x\_test\_selected.shape is: **{}**'.format(x\_test\_selected\_ABT3.shape))

*# mutual\_info\_classif*

In [ ]:

*#logreg w mututal iNfo*

**from** **sklearn.linear\_model** **import** LogisticRegression

**from** **sklearn.linear\_model** **import** LogisticRegressionCV

**from** **sklearn.model\_selection** **import** KFold *# import KFold*

**from** **sklearn.model\_selection** **import** cross\_val\_score

*# instantiate the model (using the default parameters)*

logreg\_abt3\_mc = LogisticRegression()

*# fit the model with data*

logreg\_abt3\_mc.fit(x\_train\_selected\_ABT3,y\_train\_ABT3)

**from** **sklearn** **import** metrics

y\_logreg\_abt3\_mc\_pred = logreg\_abt3\_mc.predict\_proba(x\_test\_selected\_ABT3)[::,1]

fpr, tpr, \_ = metrics.roc\_curve(y\_test\_ABT3, y\_logreg\_abt3\_mc\_pred)

auc = metrics.roc\_auc\_score(y\_test\_ABT3, y\_logreg\_abt3\_mc\_pred)

plt.plot(fpr,tpr,label="logreg with Cross valid\_ABT3, auc="+str(auc))

plt.legend(loc=4)

plt.show()

In [ ]:

**from** **sklearn.model\_selection** **import** train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(ABT\_FINAL\_MODELS, target, test\_size = 0.20, stratify = target, random\_state = 33)

In [ ]:

*# neural net*

**from** **sklearn.neural\_network** **import** MLPClassifier

mlp =MLPClassifier(random\_state=33)

mlp.fit(x\_train,y\_train)

**from** **sklearn** **import** metrics

y\_mlp\_proba = mlp.predict\_proba(x\_test)[::,1]

fpr, tpr, \_ = metrics.roc\_curve(y\_test, y\_mlp\_proba)

auc = metrics.roc\_auc\_score(y\_test, y\_mlp\_proba)

plt.plot(fpr,tpr,label="data- mlp, auc="+str(auc))

plt.legend(loc=4)

plt.show()

In [ ]:

5+5

In [ ]:

**import** **lightgbm** **as** **lgb**

features = ABT\_FINAL\_MODELS.copy()

*# Only numeric features*

features = features.select\_dtypes('number')

*# Extract the labels*

labels = np.array(features['TARGET'].astype(np.int32)).reshape((-1, ))

features = features.drop(columns = ['TARGET', 'SK\_ID\_CURR'])

*# Split into training and testing data*

train\_features, test\_features, train\_labels, test\_labels = train\_test\_split(features, labels, test\_size = 0.3, random\_state = 50)

*# Create a training and testing dataset*

train\_set = lgb.Dataset(data = train\_features, label = train\_labels)

test\_set = lgb.Dataset(data = test\_features, label = test\_labels)

*# Get default hyperparameters*

model = lgb.LGBMClassifier()

default\_params = model.get\_params()

**del** default\_params['n\_estimators']

cv\_results = lgb.cv(default\_params, train\_set, num\_boost\_round = 300000, early\_stopping\_rounds = 100,

metrics = 'auc', nfold = 5, seed = 42)

print(format(cv\_results['auc-mean'][-1], cv\_results['auc-stdv'][-1]))

*# Optimal number of esimators found in cv*

model.n\_estimators = len(cv\_results['auc-mean'])

*# Train and make predicions with model*

model.fit(train\_features, train\_labels)

preds = model.predict\_proba(test\_features)[:, 1]

baseline\_auc = roc\_auc\_score(test\_labels, preds)

print(format(baseline\_auc))

apptrain.describe()

In [ ]:

apptrain['TARGET'].value\_counts()

*# 0 = Loan was repaid*

*# 1 = Loan was not repaid*

In [ ]:

temp = apptrain['TARGET'].value\_counts()

df = pd.DataFrame({'labels': temp.index,

'values': temp.values

})

sb.barplot(x = 'labels', y="values", data=df)

locs, labels = plt.xticks()

plt.show()

In [ ]:

apptrain['DAYS\_BIRTH'] = abs(apptrain['DAYS\_BIRTH'])

apptrain['DAYS\_EMPLOYED'] = abs(apptrain['DAYS\_EMPLOYED'])

apptrain['DAYS\_REGISTRATION'] = abs(apptrain['DAYS\_REGISTRATION'])

apptrain['DAYS\_ID\_PUBLISH'] = abs(apptrain['DAYS\_ID\_PUBLISH'])

In [ ]:

(apptrain['DAYS\_BIRTH']/365).describe()

In [ ]:

np.count\_nonzero(apptrain['DAYS\_EMPLOYED'] == 365243)

In [ ]:

apptrain['DAYS\_EMPLOYED'].replace({365243: apptrain['DAYS\_EMPLOYED'].mean()}, inplace = True)

In [ ]:

df = apptrain.groupby('CNT\_CHILDREN')['SK\_ID\_CURR'].nunique()

**print**(df)

In [ ]:

null\_col = apptrain.columns[apptrain.isnull().any()]

apptrain[null\_col].isnull().sum().sort\_values(ascending = False)

In [ ]:

apptrain['AMT\_INCOME\_TOTAL'].describe()

In [ ]:

apptrain['AMT\_CREDIT'].describe()

In [ ]:

apptrain['AMT\_ANNUITY'].describe()

In [ ]:

apptrain['AMT\_GOODS\_PRICE'].describe()

In [ ]:

plt.hist(apptrain['DAYS\_BIRTH']/365, bins = 30, edgecolor = 'black')

plt.xlabel('Age in Years'); plt.title("Histogram of Client's Ages"); plt.ylabel('Count')

In [ ]:

corr\_test = apptrain.corr()['TARGET'].sort\_values()

**print**(corr\_test)

In [ ]:

apptrain['DAYS\_BIRTH'].corr(apptrain['TARGET'])

*# Inverse linear relationship i.e. as age increases, clients are more likely to repay loan*

In [ ]:

apptrain['DAYS\_EMPLOYED'].corr(apptrain['TARGET'])

*# Inverse linear relationship i.e. as employment tenure increases, clients are more likely to repay loan*

In [ ]:

apptrain['AMT\_INCOME\_TOTAL'].corr(apptrain['TARGET'])

In [ ]:

ext\_source = apptrain[['TARGET','EXT\_SOURCE\_1', 'EXT\_SOURCE\_2', 'EXT\_SOURCE\_3']]

**print**(ext\_source.corr())

*# As ext\_source increases, clients are less likely to default*

In [ ]:

ages = apptrain[['TARGET', 'DAYS\_BIRTH']]

ages['AGE\_YEARS'] = ages['DAYS\_BIRTH']/365

ages['AGE\_BINS'] = pd.cut(ages['AGE\_YEARS'], bins = np.linspace(20, 70, num = 11))

ages.head()

In [ ]:

ages\_mean = ages.groupby('AGE\_BINS').mean()

ages\_mean

In [ ]:

plt.bar(ages\_mean.index.astype(str), 100 \* ages\_mean['TARGET'])

plt.title('Default Rate by Age Bands'); plt.ylabel('Default Rate'); plt.xlabel('Age Bands'); plt.xticks(rotation = 90)

In [ ]:

**def** plot\_stats(feature, label\_rotation = False, horizontal\_layout = True):

temp = apptrain[feature].value\_counts()

df1 = pd.DataFrame({feature: temp.index,'Number of contracts': temp.values})

*# Calculate the percentage of target=1 per category value*

cat\_perc = apptrain[[feature, 'TARGET']].groupby([feature], as\_index = False).mean()

cat\_perc.sort\_values(by = 'TARGET', ascending = False, inplace = True)

**if**(horizontal\_layout):

fig, (ax1, ax2) = plt.subplots(ncols = 2, figsize = (12,6))

**else**:

fig, (ax1, ax2) = plt.subplots(nrows = 2, figsize = (12,14))

s = sb.barplot(ax = ax1, x = feature, y = "Number of contracts", data = df1)

**if**(label\_rotation):

s.set\_xticklabels(s.get\_xticklabels(), rotation = 90)

s = sb.barplot(ax = ax2, x = feature, y ='TARGET', order = cat\_perc[feature], data = cat\_perc)

**if**(label\_rotation):

s.set\_xticklabels(s.get\_xticklabels(), rotation = 90)

plt.ylabel('Percent Defaulted', fontsize = 10)

plt.tick\_params(axis ='both', which = 'major', labelsize = 10)

plt.show();

In [ ]:

**def** plot\_distribution(var):

i = 0

t1 = apptrain.loc[apptrain['TARGET'] == 1]

t0 = apptrain.loc[apptrain['TARGET'] == 0]

plt.figure()

fig, ax = plt.subplots(2, 2, figsize = (12,12))

**for** feature **in** var:

i += 1

plt.subplot(2,2,i)

sb.kdeplot(t1[feature], bw = 0.5,label = "TARGET = 1")

sb.kdeplot(t0[feature], bw = 0.5,label = "TARGET = 0")

plt.ylabel('Density plot', fontsize = 12)

plt.xlabel(feature, fontsize = 12)

locs, labels = plt.xticks()

plt.tick\_params(axis = 'both', which = 'major', labelsize = 12)

plt.show();

In [ ]:

viz\_stats('NAME\_CONTRACT\_TYPE')

In [ ]:

viz\_stats('CODE\_GENDER')

In [ ]:

viz\_stats('NAME\_INCOME\_TYPE', True)

In [ ]:

viz\_stats('OCCUPATION\_TYPE', True)

In [ ]:

viz\_stats('ORGANIZATION\_TYPE', True, False)

In [ ]:

viz\_stats('NAME\_EDUCATION\_TYPE', True)

In [ ]:

viz\_stats('NAME\_HOUSING\_TYPE', True)

In [ ]:

viz\_stats('FLAG\_OWN\_REALTY')

In [ ]:

viz\_stats('NAME\_FAMILY\_STATUS',True, True)

In [ ]:

viz\_stats('CNT\_CHILDREN')

In [ ]:

viz\_stats('CNT\_FAM\_MEMBERS', True)

In [ ]:

viz\_stats('REG\_CITY\_NOT\_LIVE\_CITY')

In [ ]:

viz\_stats('REG\_CITY\_NOT\_WORK\_CITY')

In [ ]:

**def** plot\_num(feature):

plt.figure(figsize = (10, 6))

plt.title("Distribution of **%s**" % feature)

sb.distplot(apptrain[feature].dropna(), kde = True, bins = 100)

plt.show()

In [ ]:

**def** plot\_num\_multi(var, nrow = 2):

i = 0

t1 = apptrain.loc[apptrain['TARGET'] == 1]

t0 = apptrain.loc[apptrain['TARGET'] == 0]

sb.set\_style('dark')

plt.figure()

fig, ax = plt.subplots(nrow, 2, figsize = (12, 6\*nrow))

**for** feature **in** var:

i += 1

plt.subplot(nrow, 2, i)

sb.kdeplot(t1[feature], bw = 0.5, label = "TARGET = 1")

sb.kdeplot(t0[feature], bw = 0.5, label = "TARGET = 0")

plt.ylabel('Density plot', fontsize = 12)

plt.xlabel(feature, fontsize = 12)

locs, labels = plt.xticks()

plt.tick\_params(axis = 'both', which = 'major', labelsize = 12)

plt.show();

In [ ]:

plot\_num('AMT\_INCOME\_TOTAL')

In [ ]:

plot\_num('AMT\_CREDIT')

In [ ]:

plot\_num('AMT\_ANNUITY')

In [ ]:

plot\_num('AMT\_GOODS\_PRICE')

In [ ]:

plot\_num('DAYS\_REGISTRATION')

In [ ]:

plot\_num('DAYS\_ID\_PUBLISH')

In [ ]:

var = ['AMT\_CREDIT', 'AMT\_ANNUITY', 'AMT\_GOODS\_PRICE', 'DAYS\_REGISTRATION', 'DAYS\_ID\_PUBLISH', 'DAYS\_EMPLOYED']

plot\_num\_multi(var, nrow = 3)

In [ ]:

*#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

bureau = pd.read\_csv('bureau.csv')

*#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

In [ ]:

bureau.head()

In [ ]:

bureau.shape

In [ ]:

prev\_loan\_count = bureau.groupby('SK\_ID\_CURR', as\_index = False)['SK\_ID\_BUREAU'].count().rename(columns = {'SK\_ID\_BUREAU': 'PREV\_LOAN\_COUNT'})

prev\_loan\_count.head()

In [ ]:

*# Join prev\_loan\_count to a copy of the apptrain data*

train\_prev\_loan\_count = apptrain.copy()

train\_prev\_loan\_count = train\_prev\_loan\_count.merge(prev\_loan\_count, on = 'SK\_ID\_CURR', how = 'left')

*# Replace missing values with 0*

train\_prev\_loan\_count['PREV\_LOAN\_COUNT'] = train\_prev\_loan\_count['PREV\_LOAN\_COUNT'].fillna(0)

train\_prev\_loan\_count.head()

In [ ]:

**def** kde\_target(var\_name, df):

corr = df['TARGET'].corr(df[var\_name])

mean\_repaid = df.ix[df['TARGET'] == 0, var\_name].mean()

mean\_not\_repaid = df.ix[df['TARGET'] == 1, var\_name].mean()

plt.figure(figsize = (12,6))

sb.kdeplot(df.ix[df['TARGET'] == 0, var\_name], label = 'Target = 0')

sb.kdeplot(df.ix[df['TARGET'] == 1, var\_name], label = 'Target = 1')

plt.xlabel(var\_name); plt.ylabel('Density'); plt.title('**%s** Distribution' % var\_name)

plt.legend()

**print**('**%s** **%0.4f** correlation' % (var\_name, corr))

**print**('average repaid: **%0.4f**' % mean\_repaid)

**print**('average not repaid: **%0.4f**' % mean\_not\_repaid)

In [ ]:

kde\_target('EXT\_SOURCE\_3', train\_prev\_loan\_count)

In [ ]:

kde\_target('PREV\_LOAN\_COUNT', train\_prev\_loan\_count)

In [ ]:

agr\_bureau = bureau.drop(columns = ['SK\_ID\_BUREAU']).groupby('SK\_ID\_CURR', as\_index = False).agg(['count', 'min', 'max', 'mean', 'sum']).reset\_index()

agr\_bureau.head()

**Sample visualizations**

