HKUST Business School
ISOM 3360 Data Mining for Business Analytics

Project Status Report

Group 10

Names	SID
Chan Pui Yiu	20505800
Law Pui Ka, Kelly	20432376
Wong Tsz Yiu L	20515049
LEE Yeonju (Ashley)	20319108

1. Introduction of the project

The project is based on a data mining competition from Kaggle[1] - “Home Credit Default Risk”. It is a featured prediction problem that aims to predict how capable each applicant is of repaying a loan from a given number of features.

The importance of the task to the commercial bank is that a safer lending experience can be reaped by allowing the bank to spot the potential client that may have the chance to be default and stop lending them money OR take some remedial measure before they fail, further decrease the chance of lending money to risky customers that may resulting in a bad debt and increase the profit by maintaining a lower level of bad debt lending.

2. Data Exploration and Data Preprocessing

2.1 Data Exploration:

```
In [5]: # to view training set information
        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
```

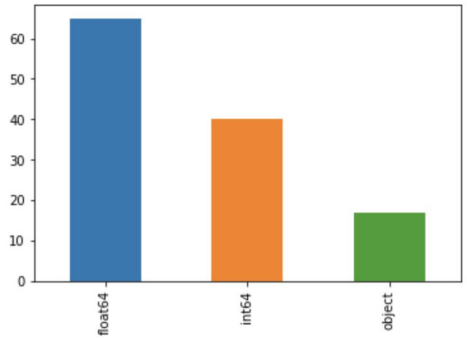
There are around 307500 records(instances) & 122 attributes. There are categorical features such as Gender, car & realty ownership, education level, **marital status** that have option such as Single, Married, Civil Marriage and **types of occupation:** Laborer, Core Staff, Accountants, Managers, Medicine staff, Driver, Sales Stuff, Private Service Staff**,**. & Numerical attributes such as amount of credit annuity and income level.

The Target variable we are going to predict is “TARGET” (the loan default status)

1 means the transaction turn to default
0 means the transaction didn’t turns to default

Nature of data mining: Classification and supervised learning

2.2 Data type




```
In [54]: # Number of each type of column
         app_train.dtypes.value_counts()

Out[54]: float64    65
         int64      40
         object     17
         dtype: int64
```

For the overview of the number of columns of each data type. int64 and float64 are for numeric variables(which can be either discrete or continuous). Object data type contains strings and are for categorical features

[1] Kaggle Featured Prediction Competition <Home Credit Default Risk>
https://www.kaggle.com/c/home-credit-default-risk#description

```
In [13]: app_train['TARGET'].describe()

Out[13]: count    307511.000000
         mean          0.080729
         std           0.272419
         min           0.000000
         25%           0.000000
         50%           0.000000
         75%           0.000000
         max           1.000000
         Name: TARGET, dtype: float64
```

It is found out that the data type for some variables are not truly represent their meanings. “TARGET”: The target variable is what we would like to predict: where a 0 for the loan was repaid on time, and 1 indicating the client had payment difficulties, but “float64” is for numeric variables. As It doesn’t carry any numerical meaning, so we would like to change its data type for a categorical variables

```
In [15]: #Use .astype() to change the data type for a variable
         app_train['TARGET'] = app_train['TARGET'].astype(str)

In [16]: app_train['TARGET'].describe()

Out[16]: count     307511
         unique         2
         top            0
         freq      282686
         Name: TARGET, dtype: object
```

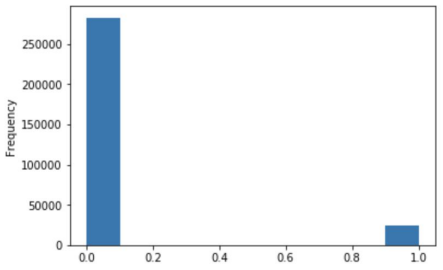
2.3 Class imbalance

When we examine the distribution of the target variable:

```
In [86]: app_train['TARGET'].value_counts()

Out[86]: 0    282686
         1     24825
         Name: TARGET, dtype: int64
```

```
In [87]: app_train['TARGET'].astype(int).plot.hist();
```



It is observed that, there is an significant imbalanced class problem where there are far more people repaid the loan on time than those who did not. We may bear this fact in mind when we do the model evaluation such as the accuracy rate and TPR and FPR.

2.4 Missing Value Handling

An objective decision threshold is when the total number of data with missing value constitute

	Missing Values	% of Total Values
COMMONAREA_MEDI	214865	69.9
COMMONAREA_AVG	214865	69.9
COMMONAREA_MODE	214865	69.9
NONLIVINGAPARTMENTS_MEDI	213514	69.4
NONLIVINGAPARTMENTS_MODE	213514	69.4
NONLIVINGAPARTMENTS_AVG	213514	69.4
FONDKAPREMONT_MODE	210295	68.4
LIVINGAPARTMENTS_MODE	210199	68.4
LIVINGAPARTMENTS_MEDI	210199	68.4
LIVINGAPARTMENTS_AVG	210199	68.4
AMT_REQ_CREDIT_BUREAU_YEAR	41519	13.5
NAME_TYPE_SUITE	1292	0.4
DEF_30_CNT_SOCIAL_CIRCLE	1021	0.3
OBS_60_CNT_SOCIAL_CIRCLE	1021	0.3
DEF_60_CNT_SOCIAL_CIRCLE	1021	0.3
OBS_30_CNT_SOCIAL_CIRCLE	1021	0.3
EXT_SOURCE_2	660	0.2
AMT_GOODS_PRICE	278	0.1
AMT_ANNUITY	12	0.0
CNT_FAM_MEMBERS	2	0.0

2.4.1 Dropping the columns and rows: If the features contain more than 30% missing values, we will drop the whole column e.g. the column of “COMMONAREA_MEDI”

Rows: If there is only <1% missing values, we will just drop those instances -e.g. The rows in ”EXT_SOURCE_2”

2.4.2 Replacing with mean, median & mode

YEARS_BUILD_AVG	204488	66.5
OWN_CAR_AGE	202929	66.0
LANDAREA_AVG	182590	59.4
LANDAREA_MEDI	182590	59.4

It is found that the OWN_CAR_AGE has a lot of missing value and which draw our attention. (OWN_CAR_AGE: An individual owns his car for how long)

And it is believed that it can reference back to the attributes - ‘FLAG_OWN_CAR’, that indicating if one own a car OR not. (Y: Yes, N:No)

```
In [18]: app_train['FLAG_OWN_CAR'].value_counts()

Out[18]: N    202924
         Y    104587
         Name: FLAG_OWN_CAR, dtype: int64
```

It turns out that there are 202924 observations with N: not owning a car, which highly match the numbers of missing value above(202929), so it is reasonable to replace those missing value exhibited in attributes -’OWN_CAR_AGE’ with 0, indicating that the time period for them owning a car is 0.

```
In [21]: app_train['OWN_CAR_AGE'].fillna("0", inplace=True)

In [23]: app_train['OWN_CAR_AGE'].value_counts()

Out[23]: 0       202929
         7.0       7424
         6.0       6382
         3.0       6370
         8.0       5887
```

‘OCCUPATION_TYPE’ is categorical variable, we would like to fill the missing value with the mode.

```
In [17]: app_train['OCCUPATION_TYPE'].describe()

Out[17]: count        211120
         unique           18
         top       Laborers
         freq          55186
         Name: OCCUPATION_TYPE, dtype: object

Out[17]: count        211120
         unique           18
         top       Laborers
         freq          55186
         Name: OCCUPATION_TYPE, dtype: object

In [19]: app_train['OCCUPATION_TYPE'].fillna("Laborers",inplace=True)
```

Numerical variables:
Fill them all with Mean

```
In [21]: app_train['EXT_SOURCE_3'].fillna(app_train['EXT_SOURCE_3'].mean(),inplace=True)
         app_train['AMT_REQ_CREDIT_BUREAU_YEAR'].fillna(app_train['AMT_REQ_CREDIT_BUREAU_YEAR'].mean(),inplace=True)
         app_train['AMT_REQ_CREDIT_BUREAU_QRT'].fillna(app_train['AMT_REQ_CREDIT_BUREAU_QRT'].mean(),inplace=True)
         app_train['AMT_REQ_CREDIT_BUREAU_MON'].fillna(app_train['AMT_REQ_CREDIT_BUREAU_MON'].mean(),inplace=True)
         app_train['AMT_REQ_CREDIT_BUREAU_WEEK'].fillna(app_train['AMT_REQ_CREDIT_BUREAU_WEEK'].mean(),inplace=True)
         app_train['AMT_REQ_CREDIT_BUREAU_DAY'].fillna(app_train['AMT_REQ_CREDIT_BUREAU_DAY'].mean(),inplace=True)
         app_train['AMT_REQ_CREDIT_BUREAU_HOUR'].fillna(app_train['AMT_REQ_CREDIT_BUREAU_HOUR'].mean(),inplace=True)
```

2.5 Outlier Detection

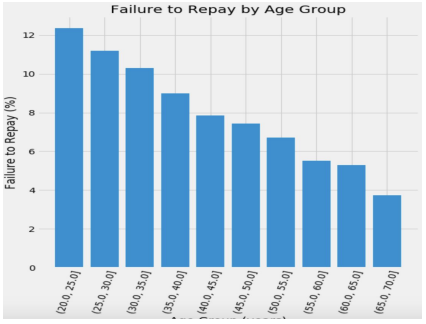
The numbers in the **DAYS_EMPLOYED** attributes are negative because they are recorded relative to the current loan application. To see these statistics in years, we can multiply it by -1 and divide by the number of days in a year:

```
In [88]: (app_train['DAYS_EMPLOYED'] / -365).describe()

Out[88]: count    307511.000000
         mean       -174.835742
         std         387.056895
         min       -1000.665753
         25%           0.791781
         50%           3.323288
         75%           7.561644
         max          49.073973
         Name: DAYS_EMPLOYED, dtype: float64
```

It can be seen that the statistics look reasonable does not look sensible! The minimum value (besides being negative) is about 1000 years! It is believed to be a outlier (OR input by mistake)

2.6 Data Visualization

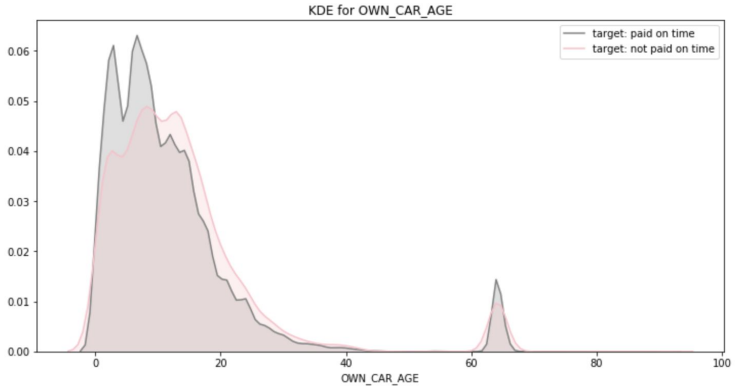


We try visualize the correlation between the age of the client and the target variable.
Given the age of the client and target variable exhibit negative relationship, we try to classify the client based the age group. An obvious relationship is shown: younger clients are less likely to repay the loan. The rate of failure to repay is above 10% for the youngest three age groups and below 5% for the oldest age group. Therefore, the bank can provide more financial planning tips to help the younger clients to repay on time.

3. Data Exploration analysis (Feature Selection)

3.1 KDE

We would like to capture the effect of the attributes that have on the target, a kernel density estimation plot (KDE) colored by the value of the target will be used. The kernel density estimate plot shows the distribution of a single variable and can be thought of as a smoothed histogram.



It can be observed that the variable -"OWN_CAR_AGE" is likely going to be useful in a machine learning model because it does affect the target and we regarded it as a **significant** attributes that have certain correlation against the target variable.

Others significant **float** attributes including “EXT_SOURCE_1”, “EXT_SOURCE_2”, “EXT_SOURCE_3”,”BASEMENTAREA_AVG” and “BASEMENTAREA_MODE”

The significant **integer** attributes are “DAYS_BIRTH”, “DAYS_ID_PUBLISH”, “HOUR_APPR_PROCESS_START”, “FLAG_DOCUMENT_3” and “REG_CITY_NOT_WORK_CITY”.

The significant **categorical** attributes are “NAME_CONTRACT_TYPE”, “GENDER”, “NAME_INCOME_TYPE”, “ACADEMIC_DEGREE”, “OCCUPATION_TYPE”
(Graph and meaning of the the attributes please see Appendix 1, 2 and 3)

3.2 Correlations of independent variables and the TARGET

Most Positive Correlations:

TARGET	1.000000
DAYS_BIRTH	0.078239
REGION_RATING_CLIENT_W_CITY	0.060893
REGION_RATING_CLIENT	0.058899
NAME_INCOME_TYPE_Working	0.057481
DAYS_LAST_PHONE_CHANGE	0.055218
CODE_GENDER_M	0.054713
DAYS_ID_PUBLISH	0.051457
REG_CITY_NOT_WORK_CITY	0.050994
NAME_EDUCATION_TYPE_Secondary / secondary special	0.049824
FLAG_EMP_PHONE	0.045982
REG_CITY_NOT_LIVE_CITY	0.044395
FLAG_DOCUMENT_3	0.044346
OCCUPATION_TYPE_Laborers	0.043019
DAYS_REGISTRATION	0.041975
OWN_CAR_AGE	0.037612
LIVE_CITY_NOT_WORK_CITY	0.032518
DEF_30_CNT_SOCIAL_CIRCLE	0.032248
DEF_60_CNT_SOCIAL_CIRCLE	0.031276
OCCUPATION_TYPE_Drivers	0.030303

Name: TARGET, dtype: float64

Most Negative Correlations:

EXT_SOURCE_3	-0.178919
EXT_SOURCE_2	-0.160472
EXT_SOURCE_1	-0.155317
NAME_EDUCATION_TYPE_Higher education	-0.056593
NAME_INCOME_TYPE_Pensioner	-0.046209
ORGANIZATION_TYPE_XNA	-0.045987
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
ELEVATORS_MEDI	-0.033863
FLOORSMIN_AVG	-0.033614
FLOORSMIN_MEDI	-0.033394
WALLSMATERIAL_MODE_Panel	-0.033119
LIVINGAREA_AVG	-0.032997
LIVINGAREA_MEDI	-0.032739
FLOORSMIN_MODE	-0.032698

Name: TARGET, dtype: float64

3.3 Identify Correlated Variables

We are going to have a deeper look at these variables by investigating the correlation of features with the target variable and with each other

	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	REGION_POPULATION_RELATIVE
TARGET	1.000000	0.019187	0.003982	0.030369	0.012817	0.037227
CNT_CHILDREN	0.019187	1.000000	0.012882	0.002145	0.021374	0.025573
AMT_INCOME_TOTAL	0.003982	0.012882	1.000000	0.156870	0.191657	0.074796
AMT_CREDIT	0.030369	0.002145	0.156870	1.000000	0.770138	0.099738
AMT_ANNUITY	0.012817	0.021374	0.191657	0.770138	1.000000	0.118429
REGION_POPULATION_RELATIVE	0.037227	0.025573	0.074796	0.099738	0.118429	1.000000
DAYS_BIRTH	0.078239	0.330938	0.027261	0.055436	0.009445	0.029582
DAYS_EMPLOYED	0.044932	0.239818	0.064223	0.066838	0.104332	0.003980
DAYS_REGISTRATION	0.041975	0.183395	0.027805	0.009621	0.038514	0.053820
DAYS_ID_PUBLISH	0.051457	0.028019	0.008506	0.006575	0.011268	0.003993

The threshold for dropping the columns with correlations above is 0.9

3.4 Feature Engineering (Binning / Discretization of numeric variable)

We have chosen some numeric attributes that we think it is applicable to apply data normalization. E.g. **‘AMT_INCOME_TOTAL’** to avoid some outliers with extreme income level that can distract our model

```
train['AMT_INCOME_TOTAL_scaled'].head()
```

```
0    0.141610
1    0.425526
2   -0.426220
3   -0.142305
4   -0.199088
Name: AMT_INCOME_TOTAL_scaled, dtype: float64
```

3.5 One hot encoding for categorical variable

NAME_CONTRACT_TYPE
Cash loans
Cash loans
Revolving loans
Cash loans
Cash loans

→

```
contractdummy
```

SK_ID_CURR	Cash loans	Revolving loans
100002	1	0
100003	1	0
100004	0	1
100006	1	0
100007	1	0
100008	1	0

→

NAME_CONTRACT_TYPE
0
0
1
0
0

Now we can see the the NAME_CONTRACT_TYPE column has been encoded, where 0 mean cash loans, and 1 means revolving loans for the client

4. Model Building

```
In [37]: train.shape
Out[37]: (305548, 57)
```

Model Building

Finalized Set for model building :

```
In [40]: features = train.columns[2:57]
         target = ['TARGET']
```

We will have 56 attributes and 1 target

4.1 Naive Bayes: predict the binary value of TARGET given known attributes
4.2 Evaluation of naive bayes model

```
In [25]: train['TARGET'].value_counts('0')
Out[25]: 0    0.919462
         1    0.080538
         Name: TARGET, dtype: float64
```

The benchmark for us to evaluate our model would be the random guess based on the TARGET in the training model. As we can see 91.9462% instances with TARGET = 0. Therefore it is reasonable for us to guess the coming data point would have 92% chance to be 0 - repaid the loan on time and this 92% will become the benchmark to evaluate the naive bayes model.

```
# print the accurary rate
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
print("Accuracy of Benchmark Model:","\n", accuracy_score(y, pred_val_maj, normalize=True, sample_weight=None))
print("Accuracy of Naive Bayes Model:","\n", accuracy_score(y, pred_y, normalize=True, sample_weight=None))
```

```
Accuracy of Benchmark Model:
 0.9191518190267978
Accuracy of Naive Bayes Model:
 0.9191223637529946
```

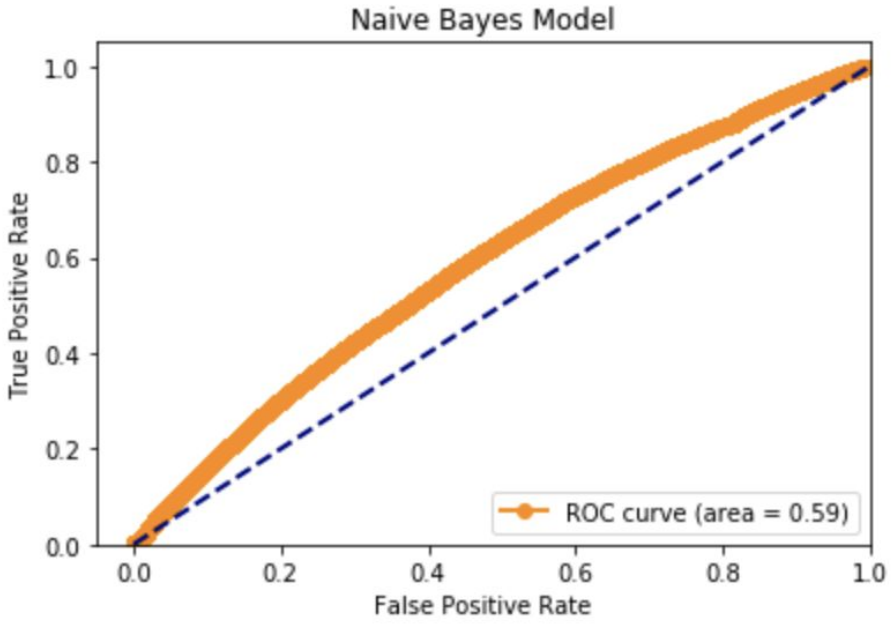
Although we got a lower accuracy rate when doing the naive bayes model compare with the benchmark model, but it is because the benchmark model is a Majority-class classifier that following the majority class vote principal, and just predict every client to be repaid on time (target = 0). Therefore, the benchmark model did not predict anything in others word. We can have a better view of this issue with a confusion matrix

```
Confusion Matrix of Benchmark Model:
 [[280845      0]
 [ 24703      0]]
Confusion Matrix of Naive Bayes Model:
 [[280835     10]
 [ 24702      1]]
```

	Precision	Recall
Benchmark Model	91.91518 19%	1
Naive Bayes Model	91.91521 81%	99.9964393%

Precision is the number of correctly classified positive examples divided by the total number of examples that are classified as positive. It can be seen that Naive Bayes Model's net is more **pure** than the benchmark model.

Afterwards, a ROC analysis will be conducted to have a better view on the classification performance

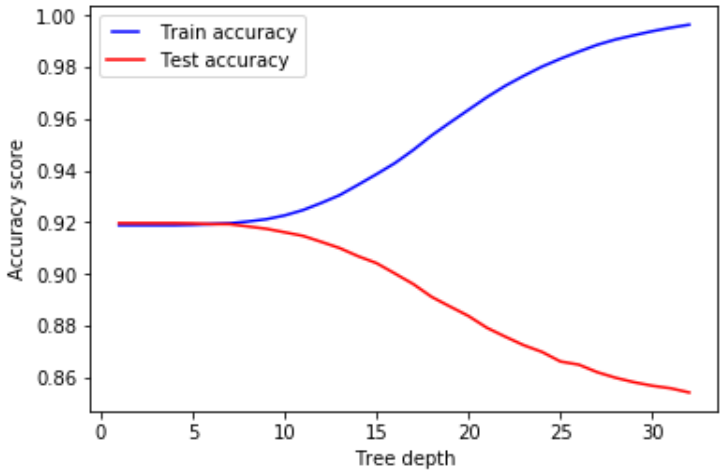


Using Accuracy Rate simply assume the benefit of receiving the interest from the client and the cost of resulting in bad debt is the same, but in reality it is not, a bad debt can cost a huge loss to the finance company. A ROC analysis can help the company to evaluate the performance of the model, and change the threshold in order to alter the confusion matrix.

The bigger the area under ROC curve (AUC), the better the model is, we can see the model did not perform very well.

4.3 Decision Tree Model:

We fit the decision tree with max_depths ranging from 1 to 32 and plot the train and test accuracy scores. It shows that max_depths ranging from 2 to 8 have the highest accuracy scores of 0.919.



Max_ depth	2	3	4	5	6	7	8	9	10
Accuracy	0.919152	0.919152	0.919152	0.919044	0.919034	0.918919	0.918582	0.918009	0.917099

	Max Depth	Mini_samples_ leaf =1	Mini_samples_ leaf =100	Mini_samples_ leaf =200
Mini_sample_split = 2	4	0.919151819	0.919151819	0.919151819
	8	0.918595440	0.919115818	**0.919207459**
	16	0.904322729	0.918621625	0.919207456
Mini_sample_split = 100	4	0.919151819	0.919151819	0.919151819
	8	0.918762354	0.919115818	**0.919207459**
	16	0.913120034	0.918615079	**0.919207459**
Mini_sample_split = 500	4	0.919151819	0.919151819	0.919151819
	8	0.918870357	0.919096181	0.919164911
	16	0.917253596	0.919014363	0.919164911

From the above data, we conclude that hyperparameters with max_depth = 8, mini_sample_split = 100 and mini_samples_ leaf =200 in the decision tree will give us the most accurate scores of 0.9192.

4.4 Evaluation of Decision Tree

```
Confusion Matrix of Decision Tree Model:
[[112397      0]
 [  9823      0]]
```

Precision	Recall
91.96285%	1

4.5 Logistic Regression: Predict the probability of the client to be default on the loan

4.6 Evaluation for Logistic Regression Model

```
In [94]: score_cv.mean()
Out[94]: 0.9190078179064758

In [95]: #predict value of target based on cross validation
         pred_y = cross_val_predict(lr, X, yact, cv=10)

In [96]: print(confusion_matrix(y, pred_y))
         [[280591    254]
          [ 24493    210]]

In [97]: print(classification_report(yact, pred_y))
                      precision    recall  f1-score   support

                   0       0.92      1.00      0.96    280845
                   1       0.45      0.01      0.02     24703

           micro avg       0.92      0.92      0.92    305548
           macro avg       0.69      0.50      0.49    305548
        weighted avg       0.88      0.92      0.88    305548
```

4.7 KNN

In order to find the K with the highest accuracy, we rendered KNeighborsClassifier for a range of 1 to 12. We selected 9 as the number of clusters as once K reaches the level of 8, the accuracy remains at 0.919.

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

print('Accuracy:', cross_val_score(model_f,X,y,cv=10).mean())

print("Classification Report:", classification_report(y_pred_2, y))

print("Confusion Matrix:", confusion_matrix(y_pred_2, y))
```

```
Accuracy: 0.9170310388383409
Classification Report:              precision    recall  f1-score   support

          0       1.00      0.92      0.96    304720
          1       0.00      0.11      0.01       828

avg / total       0.99      0.92      0.95    305548

Confusion Matrix: [[280107  24613]
 [    738     90]]
```

4.5. Conclusion

	Benchmark model	Decision Tree (max_depth = 8, mini_sample_ split = 100 and mini_samples _ leaf =200)	Naive bayes model	Logistic Regression model	KNN model
Accuracy rate	91.9462%	91.92%	91.912236375 %	91.90078%	91.703%
Precision	91.9151819%	91.96285%	91.9152181%	91.9717192%	91.9227487%
Recall	1	1	99.9964393%	99.9095586%	99.7372216%

When we try to plot the relationship of different attributes with the target variable, it can also draw some general observation, e.g. younger client tend to have a higher default risk OR female client are less likely to be default. It suggests that some independent attributes can have a significant correlation with the target variable.

Given several model, as the original dataset has a significant imbalanced problem, and it requires special handling, but in our project, it will remain as a limitation.

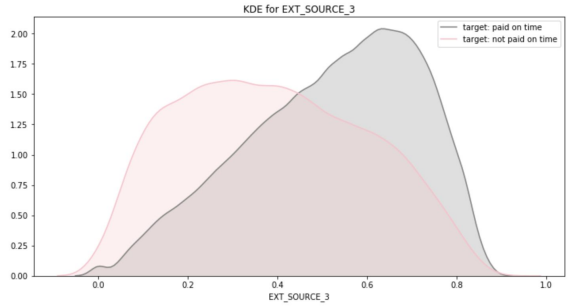
Future work:
Feature Selection and feature engineering: Due to time limitation, we did not try the advanced feature engineering in doing the feature selection, such as the **Polynomial Features** One simple feature construction method is called polynomial features. In this method, we can make features that are powers of existing features as well as interaction terms between existing features. For example, we can create variables EXT_SOURCE_1^2 and EXT_SOURCE_2^2 and also variables such as EXT_SOURCE_1 x EXT_SOURCE_2, EXT_SOURCE_1 x EXT_SOURCE_2^2, EXT_SOURCE_1^2 x EXT_SOURCE_2^2, and so on. These features that are a combination of multiple individual variables are called interaction terms because they capture the interactions between variables. In other words, while two variables by themselves may not have a strong influence on the target, combining them together into a single interaction variable might show a relationship with the target. Interaction terms are commonly used in statistical models to capture the effects of multiple variables.

Model: Logistic regression decision tree and other possible model

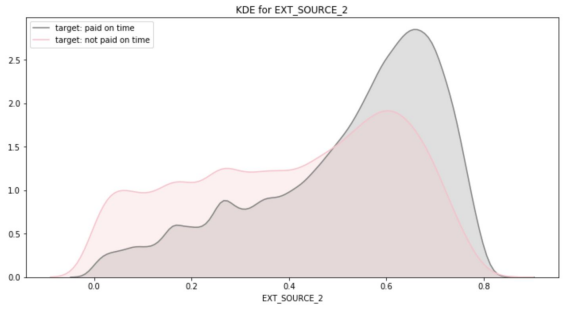
Ensemble Learning can also be applied in the future, Ensemble learning can combines multiple supervised models (the decision tree, logistic regression and naive bayes) into a huge one. instead of choosing a single predictive model, we combine several models to achieve improved performance.

1. Appendix for KDE graph with significant float attribute

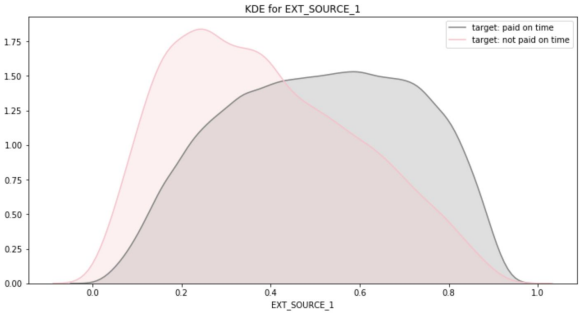
1.A: “**EXT_SOURCE_3**”: Normalized score from external data source,normalized[2]



1.B: “**EXT_SOURCE_2**”: Normalized score from external data source,normalized



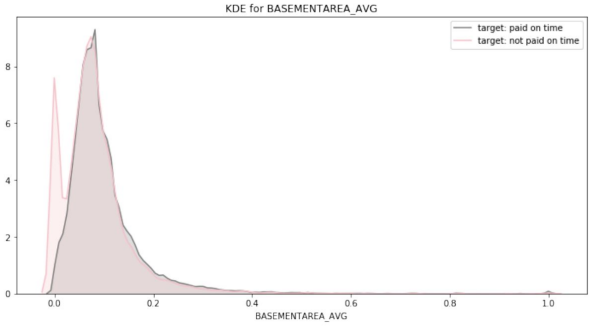
1.C: “**EXT_SOURCE_1**”: Normalized score from external data source,normalized



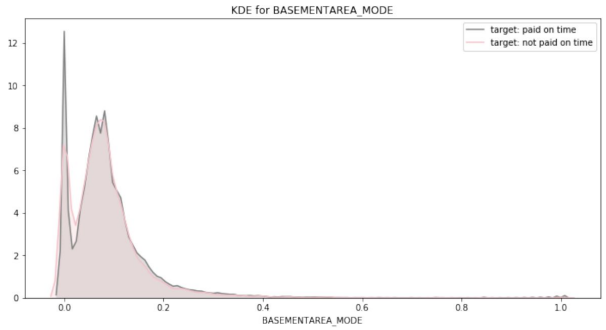
1.D: “**BASEMENTAREA_AVG**”: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor",normalized

2

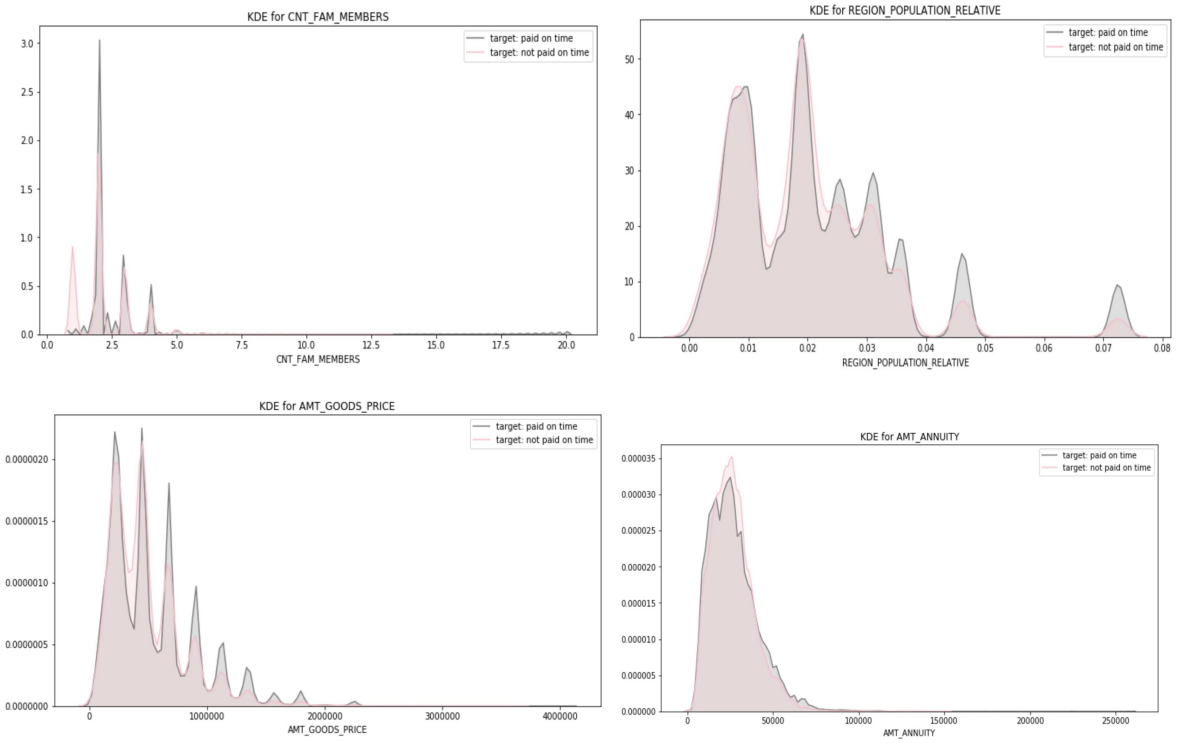
https://storage.googleapis.com/kaggle-competitions-data/kaggle/9120/HomeCredit_columns_description.csv?GoogleAccessId=web-data@kaggle-161607.iam.gserviceaccount.com&Expires=1557365282&Signature=CZA1YKclZvxVab8xS%2Fp%2Fn1S8Y%2FVtArzK80PxJwF1eNgGbPsEOQkPeJQjcyEL9jt8ATlkoYZ5%2FuSDEOW%2FhZwjc7YF1Y67cw7PPzTBLQ0TmYpGPRSsIy%2BpzVYxuDuDzZgItcFErFT%2FPt8FYrVxReWM4F33wtyPbkTDIr13RSQNZHteVU2v3OqrOinJRuAVd%2F0UInuJ6CUsn2xwpyNP%2BUhbtFg5zPzIw%2FdJjF2wcUC3jJ5gzElhhEgs3jF9B7R12vd3QeiBRyQSEBUQII3jdANg578HqNVvICIz%2BmbXSFBIijOXDR0XA3v0YkkTlarkxxQrwi%2Fztt5RQ%2F5JCPkcb2pfwg%3D%3D



1.E: “**BASEMENTAREA_MODE**”: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor",normalized

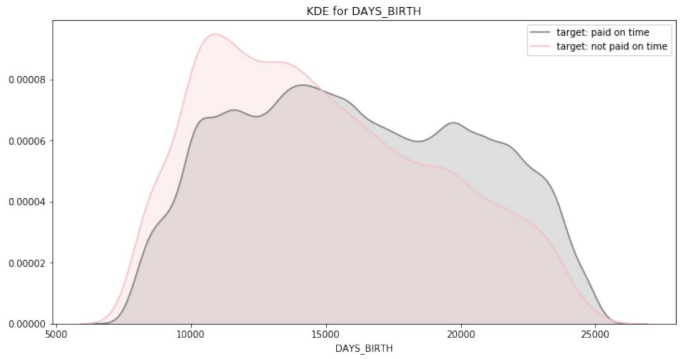


1.F: some KDE plot for others float attribute

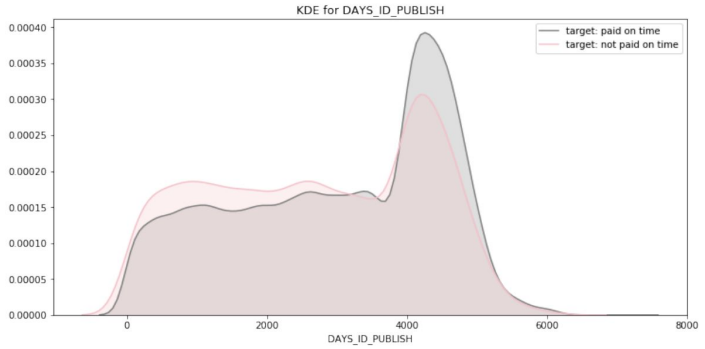


2. Appendix for KDE graph with significant integer attribute

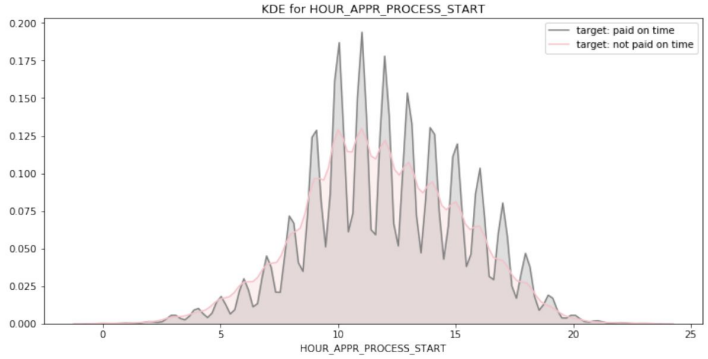
2.A: “**DAYS_BIRTH**”: Client's age in days at the time of application,time only relative to the application



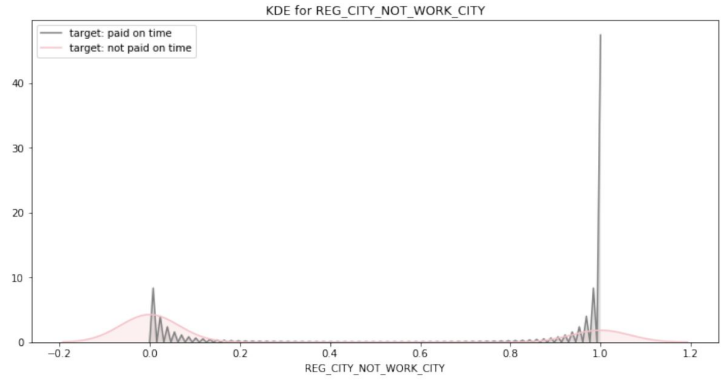
2.B: "**DAYS_ID_PUBLISH**": How many days before the application did client change the identity document with which he applied for the loan,time only relative to the application



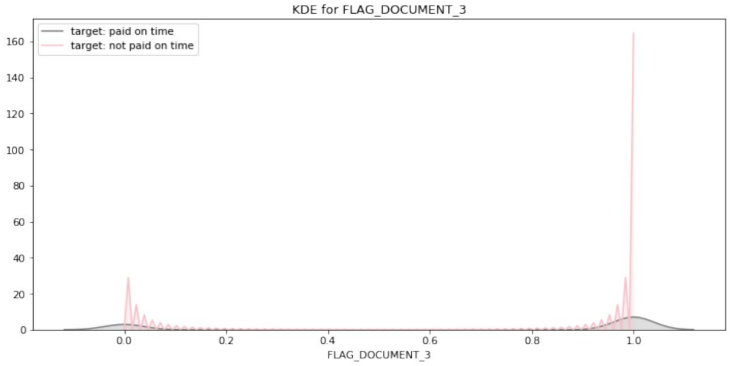
2.C: "**HOUR_APPR_PROCESS_START**": at what hour did client apply for loan



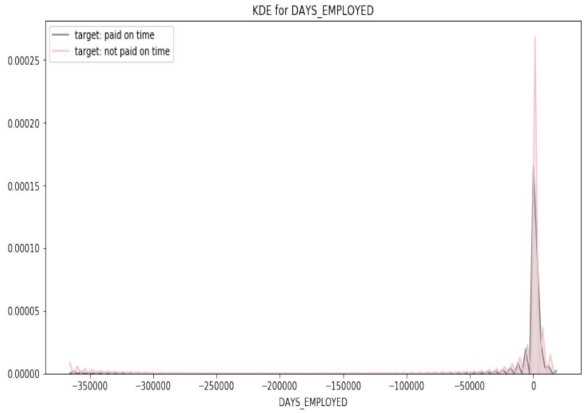
2.D: "**REG_CITY_NOT_WORK_CITY**": if client's permanent address does not match the work address (1=different, 0=same)

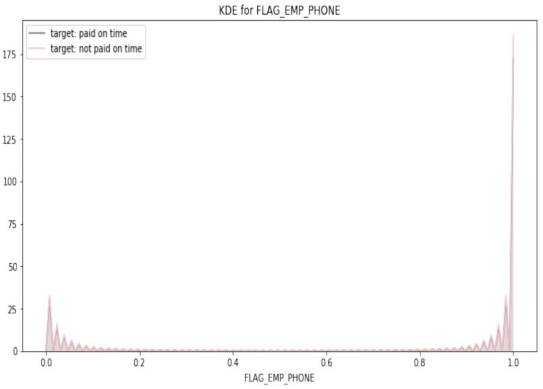


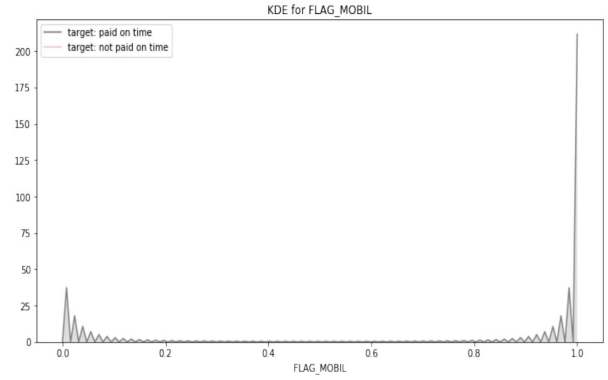
2.E: "**FLAG_DOCUMENT_3**": Did client provide document 3



2.F: some KDE plot for others integer attribute

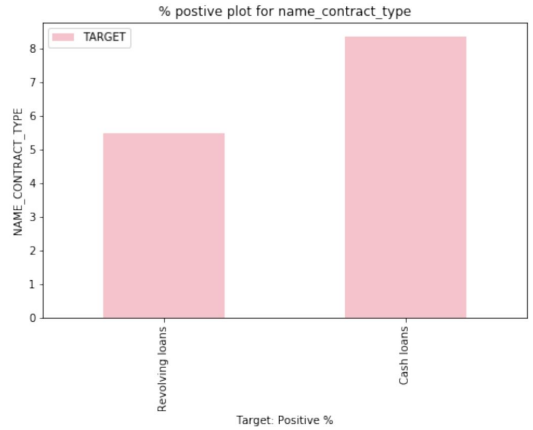






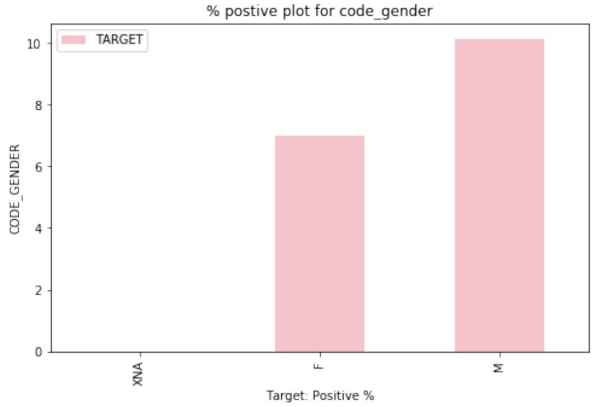
3. Appendix for KDE graph with significant categorical attribute

3.A: "**NAME_CONTRACT_TYPE**": Identification if loan is cash or revolving



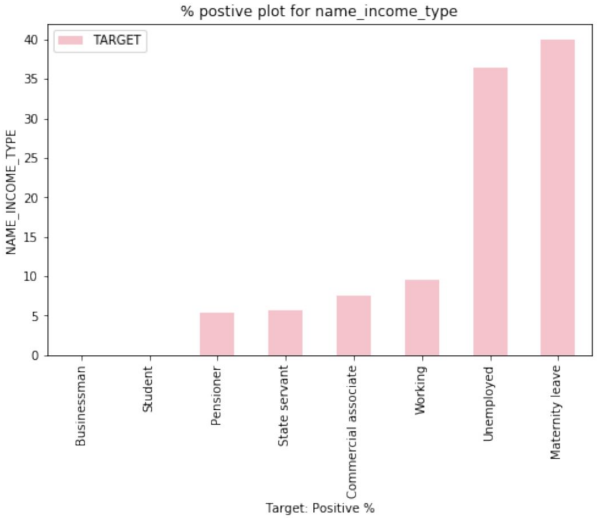
revolving (5.5), Cash (8)

3.B: "**CODE_GENDER**": Gender of the client



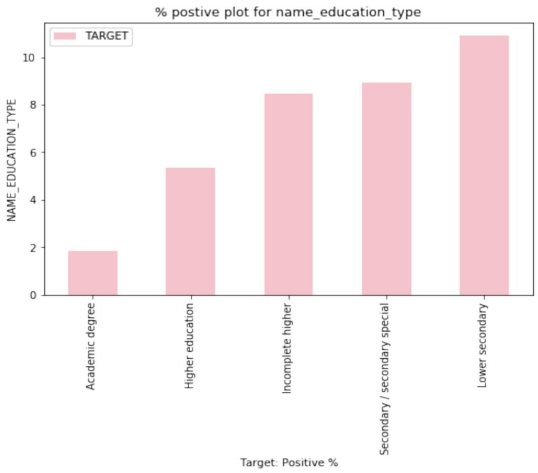
Male (10), Female (7)

3.C "**NAME_INCOME_TYPE**": Clients income type (businessman, working, maternity leave,…)"

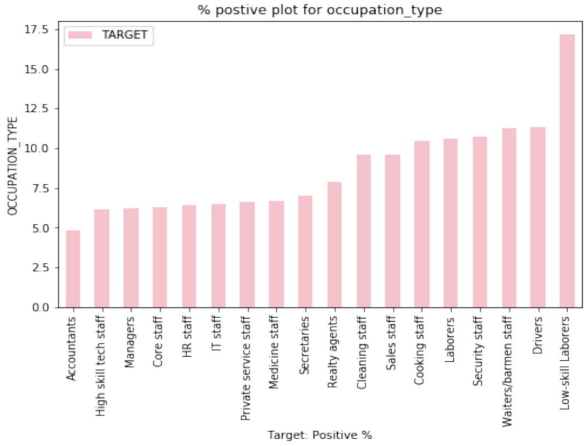


maternity leave (40), unemployed (35), working (10)

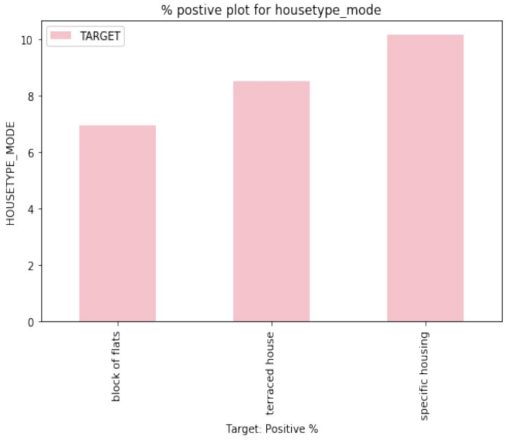
3.D "**NAME_EDUCATION_TYPE**": Level of highest education the client achieved

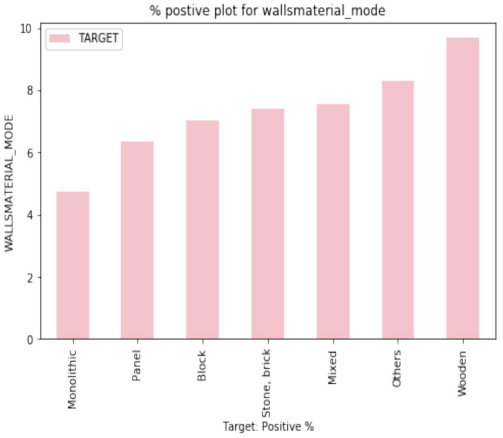


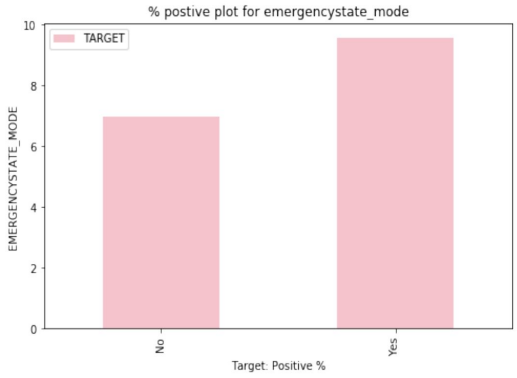
3.E "**OCCUPATION_TYPE**": What kind of occupation does the client have



3.F: some KDE plot for others categorical attribute







Reference

2. Description of the attributes:

https://storage.googleapis.com/kaggle-competitions-data/kaggle/9120/HomeCredit_columns_description.csv?GoogleAccessId=web-data@kaggle-161607.iam.gserviceaccount.com&Expires=1557365282&Signature=CZA1YKclZvxVab8xS%2Fp%2Fn1S8Y%2FVtArzK80PxJwF1eNgGbPsEOQkPeJQjcyEL9jt8ATlkoYZ5%2FuSDEOW%2FhZwjc7YF1Y67cw7PPzTBLQ0TmYpGPRSsIy%2BpzVYxuDuDzZgItcFErFT%2FPt8FYrVxReWM4F33wtyPbkTDIr13RSQNZHteVU2v3OqrOinJRuAVd%2F0UInuJ6CUsn2xwpyNP%2BUhbtFg5zPzIw%2FdJjF2wcUC3jJ5gzElhhEgs3jF9B7R12vd3QeiBRyQSEBUQII3jdANg578HqNVvICIz%2BmbXSFBIijOXDR0XA3v0YkkTIarkxxQrwi%2Fztt5RQ%2F5JCPkcb2pfwg%3D%3D

3.3 Logistic Regression Model:

We will use all the features after encoding the categorical variables in prefer to get the baseline. Preprocess has done by filling in the missing values with mean and normalising the range of the features by feature scaling.

```
In [97]: from sklearn.preprocessing import MinMaxScaler, Imputer

         # Drop the target from the training data
         if 'TARGET' in app_train:
             train = app_train.drop(columns = ['TARGET'])
         else:
             train = app_train.copy()

         # Feature names
         features = list(train.columns)

         # Copy of the testing data
         test = app_test.copy()

         # Median imputation of missing values
         imputer = Imputer(strategy = 'median')

         # Scale each feature to 0-1
         scaler = MinMaxScaler(feature_range = (0, 1))

         # Fit on the training data
         imputer.fit(train)

         # Transform both training and testing data
         train = imputer.transform(train)
         test = imputer.transform(app_test)

         # Repeat with the scaler
         scaler.fit(train)
         train = scaler.transform(train)
         #test = scaler.transform(test)

         print('Training data shape: ', train.shape)
         print('Testing data shape: ', test.shape)

Training data shape:  (307511, 237)
Testing data shape:  (48744, 237)
```

Logistic Regression from Scikit-Learn has used for the logistic regression model and lower the regularisation parameter which controls the amount of overfitting. After creating the model, ".fit" was used to train the model and ".predict_proba" was used to make prediction.

In [98]:
```
from sklearn.linear_model import LogisticRegression

# Make the model with the specified regularization parameter
log_reg = LogisticRegression(C = 0.0001)

# Train on the training data
log_reg.fit(train, train_labels)
```

Out[98]:
```
LogisticRegression(C=0.0001, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
```

“Predict.proba” was used in order to predict the probabilities of not paying a loan

In [99]:
```
# Make predictionsl
# Make sure to select the second column only
log_reg_pred = log_reg.predict_proba(test)[:, 1]
```

We create a dataframe for storing the result of the prediction so as to submit the prediction and to ensure the score of the baseline.

In [100]:
```
# Submission dataframe
submit = app_test[['SK_ID_CURR']]
submit['TARGET'] = log_reg_pred

submit.head()
```

Out[100]:

	SK_ID_CURR	TARGET
0	100001	0.0
1	100005	0.0
2	100013	0.0
3	100028	0.0
4	100038	0.0

In [101]:
```
# Save the submission to a csv file
submit.to_csv('log_reg_baseline.csv', index = False)
```

The logistic model scored around 0.67 for this baseline.