HOME LOAN DEFAULTER PREDICTION

PROJECT BY:-

- GEET JAIN
- RUCHIKA KASAR
- HITESH BARDE
- PRAVEEN RANKAWAT
- PRUTHVI RATHOD
- SHUBHAM CHOPADE

ABSTRACT

For safe and secure lending experience, it is important to analyze the past data. In this project, we have build a machine learning model on a dataset which has around 3 lakh plus rows and 122 columns, to predict the chance of default for future loans using the historical data. In this report different possible solutions for the Home Credit Default Risk has been explored. Data cleaning, exploratory data analysis, have been performed . Based on the type of each feature (categorical or numerical), the abundance of missing entries has been compensated. Several machine learning algorithms like Logistic Regression, Random Forest, Decision Tree, Gradient boosting, Adaboost and XGBoost have been trained and hyperparameters were tuned to get the best performances. Since the Target variable was very unbalanced, the area under the receiver operating characteristic curve and average precision-recall score was used for learner's evaluation. To increase the performance of the model and to make the target variable balanced we have done down-sampling of the data in the ratio of around 70:30. To reduce down the number of features we have also applied PCA on our dataset. After a lot of modifications we were able to achieve some models which were performing better than the others.

BUSINESS OBJECTIVE

The business objective for home loan defaulter analysis for machine learning model is to develop a model that can accurately predict which borrowers are likely to default on their home loans. This model can then be used to make better lending decisions, reduce risk, and improve profitability.

- Reduce loan losses: By identifying borrowers who are at high risk of default, lenders can avoid making loans to these borrowers and reduce their overall loan loss rate.
- Improve profitability: By reducing loan losses, lenders can improve their profitability.

- Make better lending decisions: Lenders can use the model to make more informed lending decisions, such as setting interest rates and loan terms.
- Improve customer satisfaction: By avoiding making loans to borrowers who are likely to default, lenders can help to ensure that their customers are successful in repaying their loans.

DATA

The dataset as it is mentioned above was provided by Home Credit Group's data scientists. It contains a wide variety of personal and financial information belonging to 307,511 individuals who had previously been recipients of loans from Home Credit.

(https://www.kaggle.com/c/home-credit-default-risk)

Home Credit works around this obstacle by looking at an extensive and diverse array of personal and financial information for each of its applicants. These features range from common characteristics, such as marital status, age, type of housing, a region of residence, job type, and education level, to some incredible niche characteristics, such as how many elevators are in an applicant's apartment building

Home Credit also looks at aspects of applicants' financial backgrounds, including month-by-month payment performance on any loans or credit card balances that the applicant has previously had with Home Credit, as well as the amount and monthly repayment balances of any loans that the applicant may have received from other lenders.

WORKFLOW

1. Collecting data
2. Starting exploratory data analysis to find trends and Storytelling
3. Conduct further data analysis to identify relationships between different variables
4. Feature Selection

5. Implement learning algorithms: Logistic Regression, Decision Tree , Random Forest, Gradient Boosting,

Adaboost Classifier, Extreme Gradient Boosting

6. Model Evaluation and Validation (AUC, Log-Loss, Precision_Recall score)

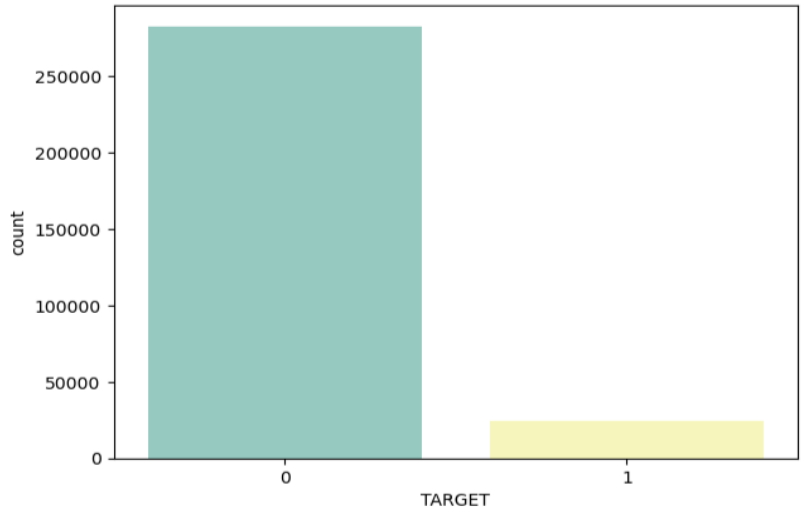
IMPLEMENTING STEPS

1.Collecting Data

After importing necessary libraries, application_train.csv data with 307,511 observations and 122 features have been read.

The Target value is unbalanced since most clients are getting cleared for getting the loan.

```
0    91.927118
1     8.072882
Name: TARGET, dtype: float64
```

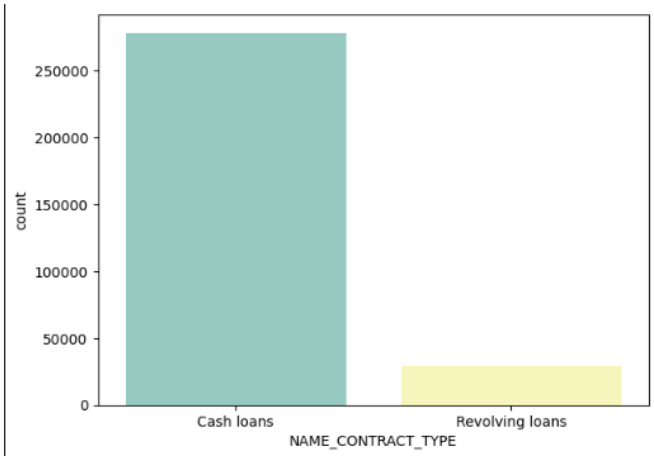


2.Exploratory Data Analysis(EDA)

Exploratory data analysis (EDA) is a process of analyzing and exploring data to discover patterns, identify outliers, and gain insights into the data. EDA is an essential step in any machine learning project, as it helps you to understand the data that you are working with and to identify any potential problems.

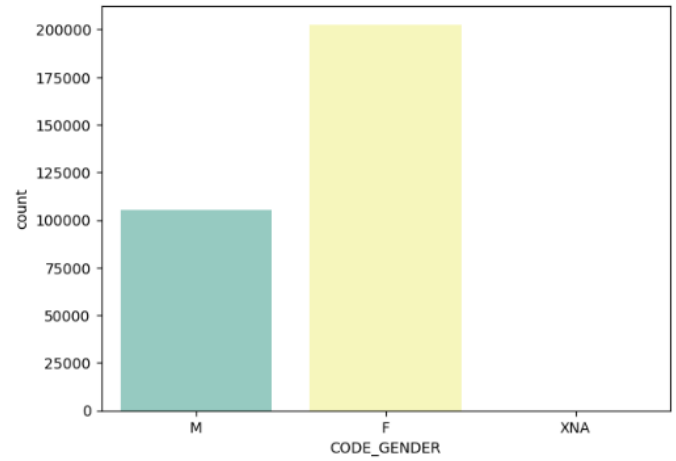
UNIVARIATE ANALYSIS *(only IMPORTANT FEATURES are chosen)*

```
Cash loans         90.478715
Revolving loans     9.521285
Name: NAME_CONTRACT_TYPE, dtype: float64
```



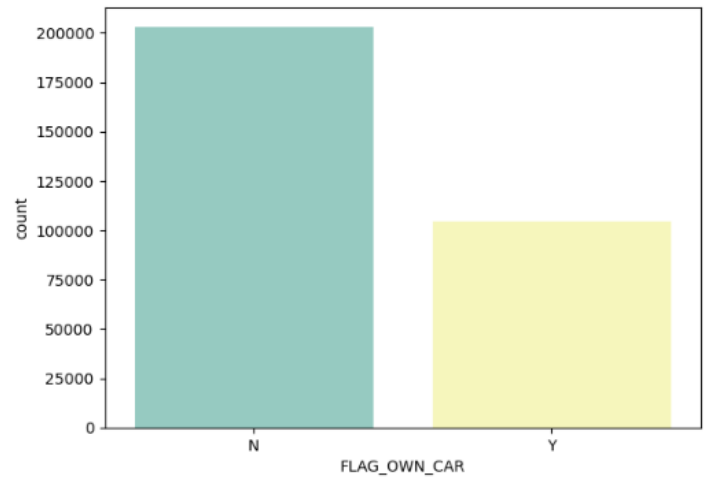
Around 90% of the contracts are 'Cash loans' and less than 10% are 'Revolving loans'.

```
F      0.658344
M      0.341643
XNA    0.000013
Name: CODE_GENDER, dtype: float64
```



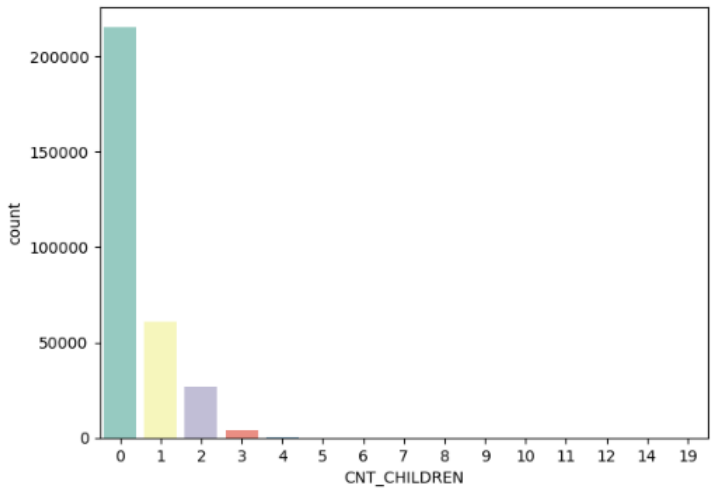
More than 66% of applicants are females.

```
Name: FLAG_OWN_CAR, dtype: float64
```



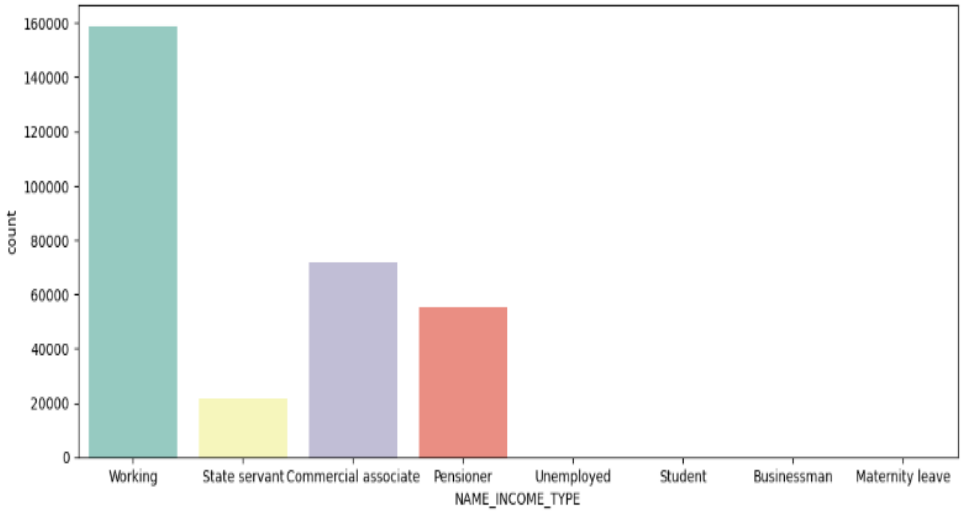
Around 65% of applicants do not have their own car.

```
Name: CNT_CHILDREN, dtype: float64
```



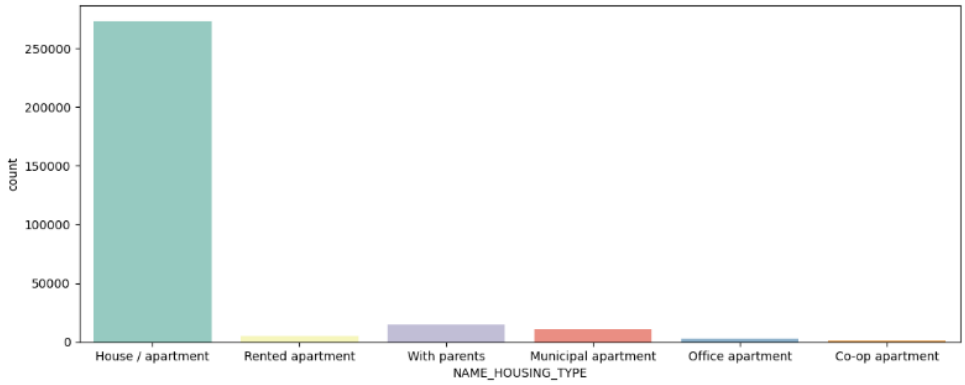
70 % of the customers dont have children, and approx 20 % of customer have single child.

```
<Axes: xlabel='NAME_INCOME_TYPE', ylabel='count'>
```



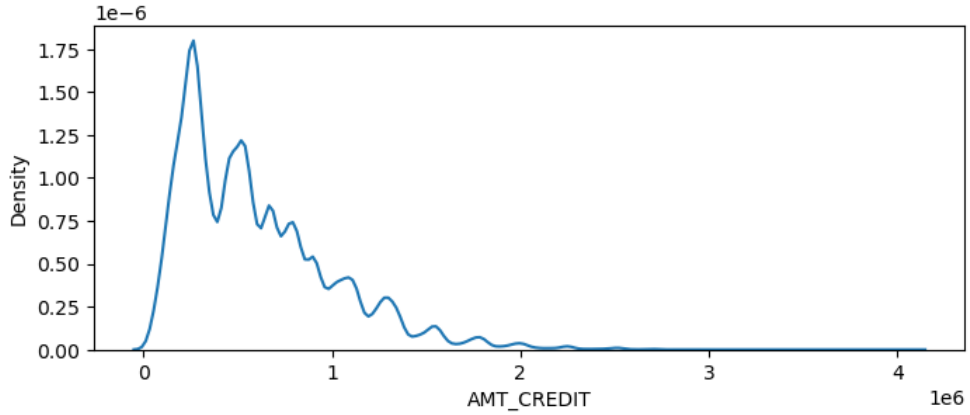
Income type for most applicants is from Working.

```
<Axes: xlabel='NAME_HOUSING_TYPE', ylabel='count'>
```

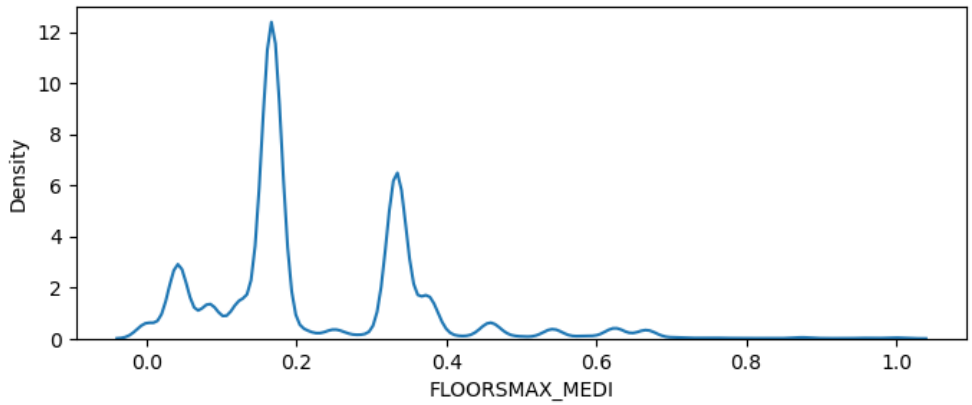


The housing type is mostly House/Apartment.

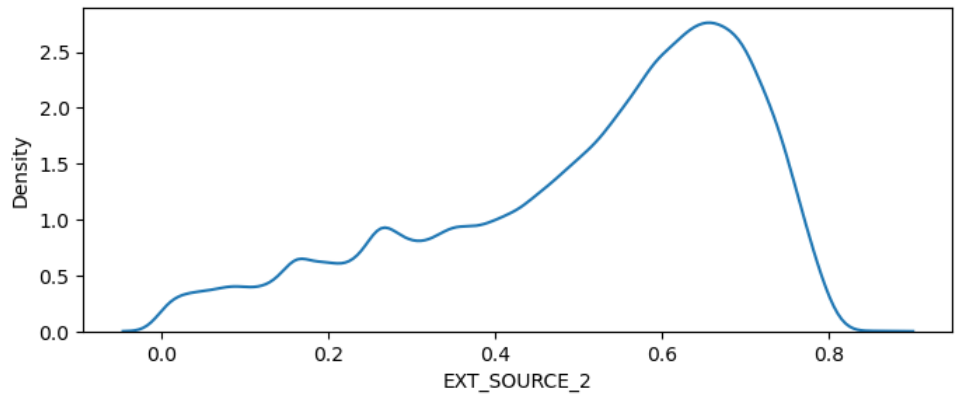
<Axes: xlabel='AMT_CREDIT', ylabel='Density'>



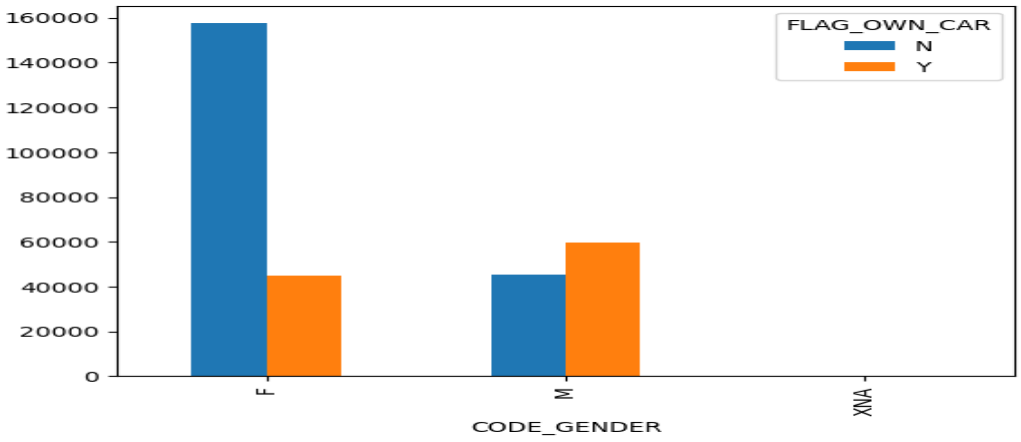
<Axes: xlabel='FLOORSMAX_MEDI', ylabel='Density'>

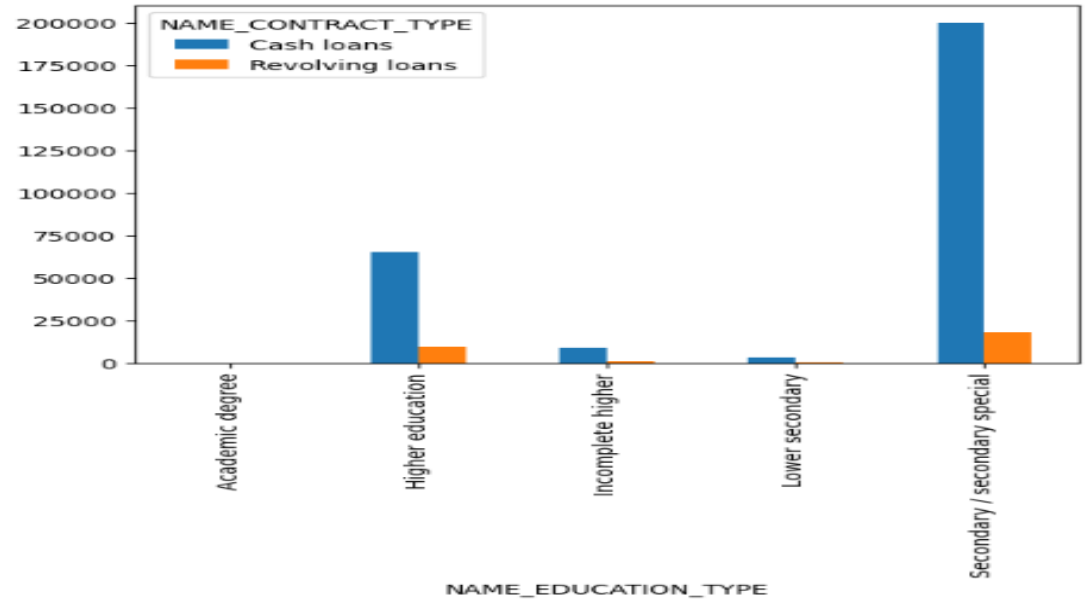


<Axes: xlabel='EXT_SOURCE_2', ylabel='Density'>

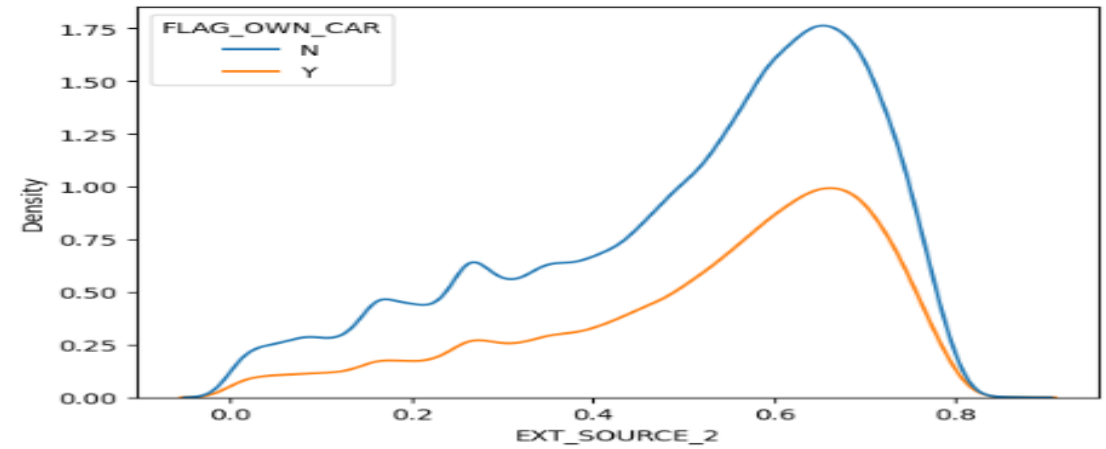


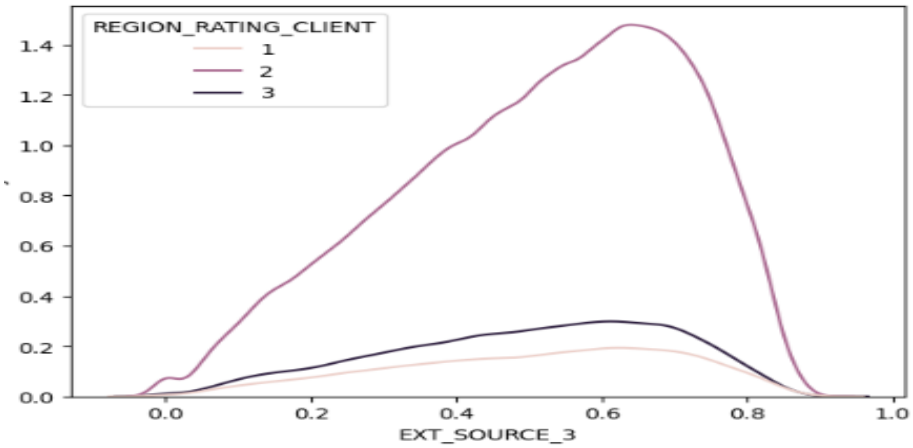
BIVARIATE ANALYSIS *(only IMPORTANT FEATURES are chosen)*





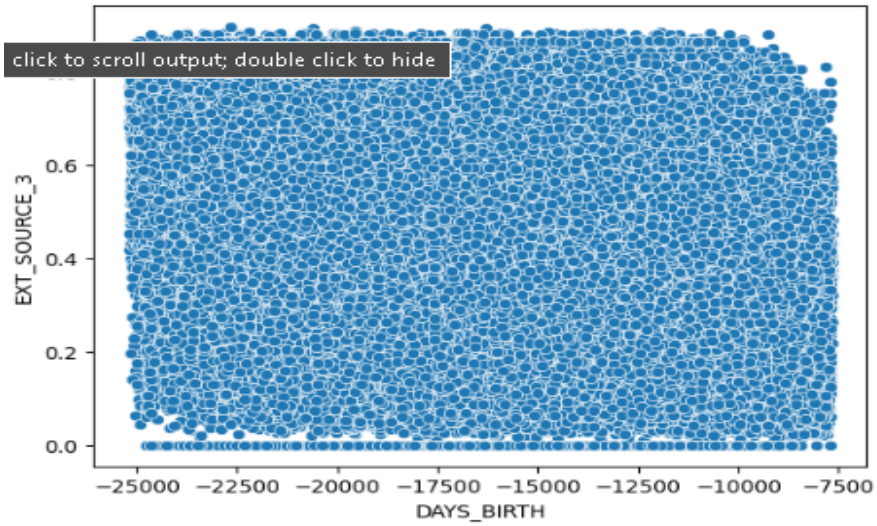

```
<Axes: xlabel='EXT_SOURCE_2', ylabel='Density'>
```

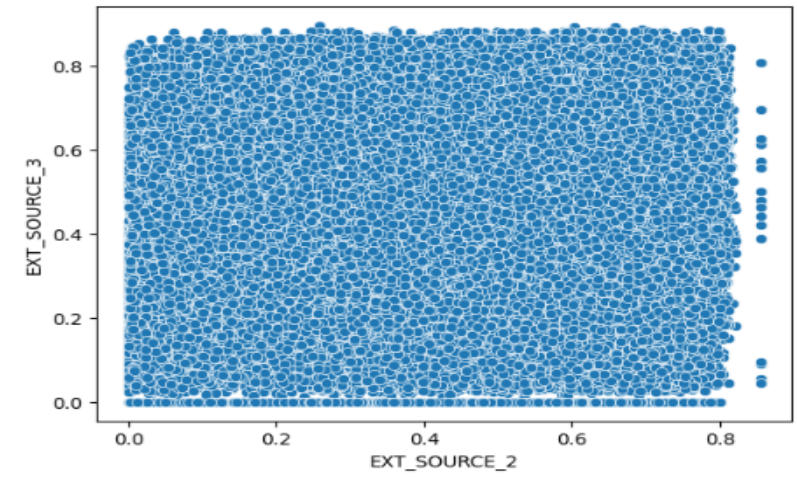


<Axes: xlabel='DAYS_BIRTH', ylabel='EXT_SOURCE_3'>

click to scroll output; double click to hide



<Axes: xlabel='EXT_SOURCE_2', ylabel='EXT_SOURCE_3'>



MULTIVARIATE ANALYSIS

KINDLY REFER TO THE .ipynb FILE FOR MULTIVARIATE ANALYSIS

MISSING VALUE TREATMENT

There are more than 65 columns which have NULL ENTRIES

Out of which 17 columns have more than 65% of NULL ENTRIES

SO WE DROPPED ALL THE COLUMNS WHICH ARE HAVING MORE THAN 65% NULL ENTRIES

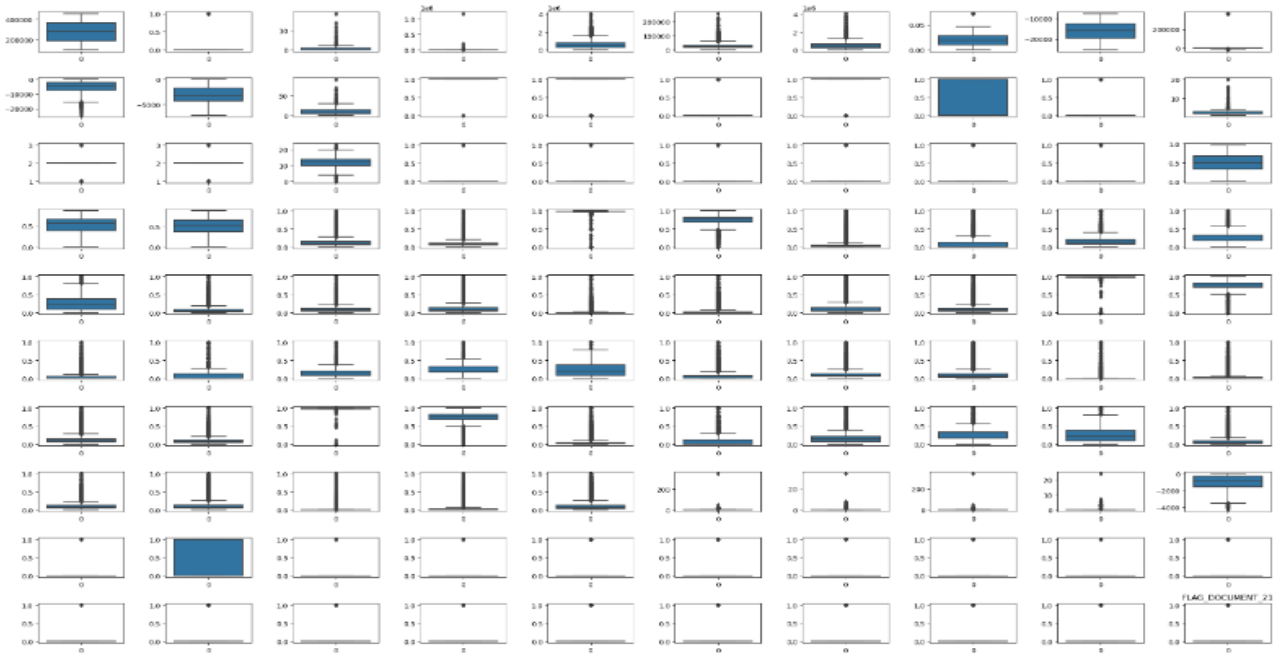
MISSING VALUE IMPUTATION:

For Categorical Features we have imputed them with MODE

For Numerical Features we have imputed them with MEDIAN

OUTLIERS DETECTION

Some of the Features have OUTLIERS but as it is a real time dataset outliers WEREN'T TREATED.



NOTE:-

We have DROPPED some features which were IRRELEVANT for the Model building.

Columns were chosen as per our Domain Knowledge and

Redundancy in the dataset.

SCALING

Scaling in the context of machine learning refers to the process of transforming feature values to a similar scale or range. Scaling is important because it can impact the performance of many machine learning algorithms.

We have used STANDARD SCALING to scale all the columns except

the Target Variable.

ENCODING

Encoding in the context of machine learning refers to the process of converting categorical data (non-numeric data) into a numerical format that can be used by machine learning algorithms.

We have used LABEL ENCODING and MAPPING to convert all the

Categorical columns into NUMERICAL COLUMN

TRAIN TEST SPLIT

The train-test split is a critical step in machine learning for evaluating the performance of your model. It involves dividing your dataset into two subsets: one for training the model and the other for testing or evaluating its performance. The train set is used to train the model, while the test set is used to assess its generalization to unseen data.

We have splitted our data into 70:30 Ratio

70-TRAINING 30- TESTING

CLASSIFICATION MODEL BUILDING

Different classification techniques have been implemented for this capstone project. The details about each one are presented below.

Before implementing each learner, the dataset split for training process we used train test split from 'sklearn.model_selection' to create a test validation set that is 30% of the size of the total training set and used StandardScaler() to normalize the data.

• LOGISTIC REGRESSION

Logistic Regression is a widely used statistical and machine learning algorithm for binary classification tasks, where the goal is to predict one of two possible outcomes (typically labeled as 0 and 1).

The Area Under the Receiver Operating Characteristic Curve (AUC-ROC score or simply AUC-ROC) is a commonly used metric for evaluating the performance of binary classification models. It assesses the model's ability to distinguish between the positive class and the negative class across different thresholds. Here's when and why you would typically use the AUC-ROC score in model building:

1. **Binary Classification Problems: AUC-ROC is most relevant for binary classification tasks, where you are classifying data points into one of two classes, typically referred to as the positive class (e.g., "1") and the negative class (e.g., "0").**
2. **Imbalanced Datasets: When dealing with imbalanced datasets, where one class significantly outnumbers the other, accuracy alone can be a misleading metric. AUC-ROC provides a more comprehensive view of a model's performance because it considers both true positive rate (recall) and false positive rate.**

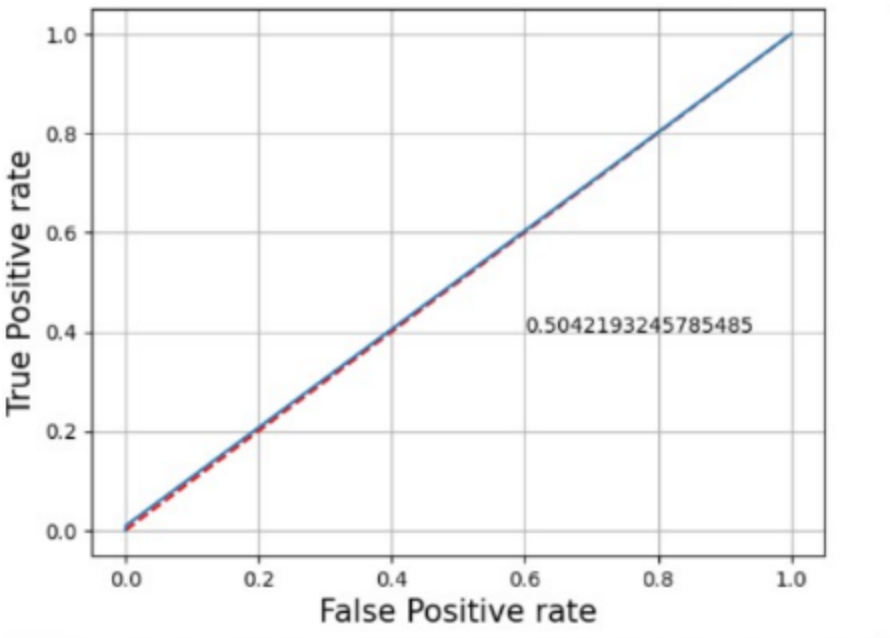
PERFORMANCE OF THE MODEL:

TRAIN	precision	recall	f1-score	support
0	0.92	1.00	0.96	197899
1	0.43	0.01	0.01	17358
accuracy			0.92	215257
macro avg	0.67	0.50	0.49	215257
weighted avg	0.88	0.92	0.88	215257

TEST	precision	recall	f1-score	support
0	0.92	1.00	0.96	84787
1	0.50	0.01	0.02	7467
accuracy			0.92	92254
macro avg	0.71	0.50	0.49	92254
weighted avg	0.89	0.92	0.88	92254

THE MODEL IS NOT PREDICTING FOR CLASS 1

THIS IS PROBABLY BECAUSE OF IMBALANCE IN TARGET VARIABLE.



THE ROC-AUC SCORE IS 0.504

WHICH IS NOT ACCEPTABLE

• DECISION TREE

Decision trees are a versatile and widely used machine learning algorithm that can be applied to a variety of problems where we need to make decisions or predictions based on input features. They are the building blocks for more advanced tree-based ensemble methods like Random Forests and Gradient Boosting.

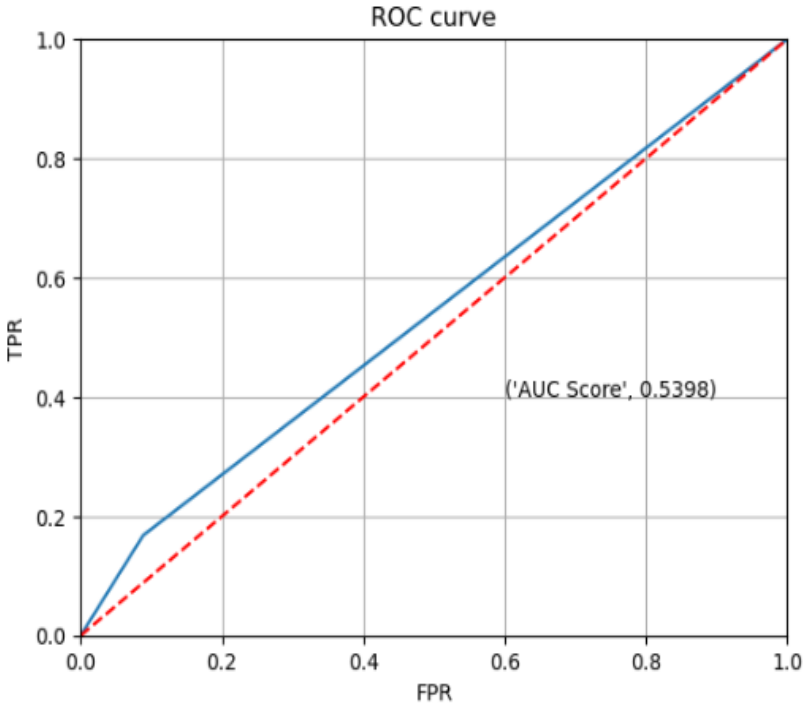
	precision	recall	f1-score	support
0	1.00	1.00	1.00	197899
1	1.00	1.00	1.00	17358
accuracy			1.00	215257
macro avg	1.00	1.00	1.00	215257
weighted avg	1.00	1.00	1.00	215257

	precision	recall	f1-score	support
0	0.93	0.91	0.92	84787
1	0.14	0.17	0.15	7467
accuracy			0.85	92254
macro avg	0.53	0.54	0.54	92254
weighted avg	0.86	0.85	0.86	92254

THE MODEL IS OVERFITTING IN TRAINING PHASE

THE MODEL IS NOT PREDICTING FOR CLASS 1

THIS IS PROBABLY BECAUSE OF IMBALANCE IN TARGET VARIABLE.



THE ROC-AUC SCORE IS 0.539

WHICH IS BETTER THAN THE LR MODEL

THE MAJOR PROBLEMS IN THE ABOVE MODELS ARE THAT THEY WERE NOT ABLE TO PREDICT THE CLASS 1 LABEL BECAUSE THERE IS IMBALANCE IN THE TARGET VARIABLE. SO NOW WE WILL TRY OVERSAMPLING , PCA OR UNDERSAMPLING TO FIND OUT WHICH MODEL WILL GIVE US BEST RESULTS.

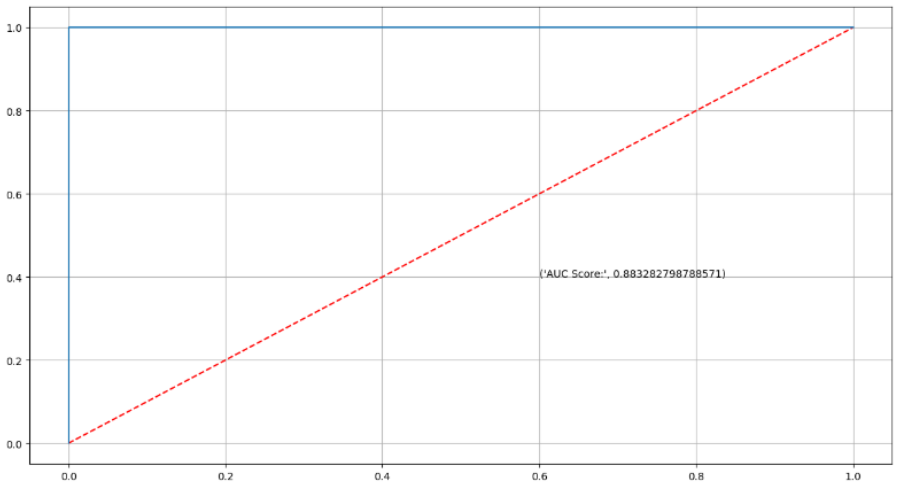
LET US TRY OVER SAMPLING

After doing oversampling of the dataset the Number of rows increased above 4 Lakhs

This is the result of a Decision tree Model .

```
Train:              precision    recall  f1-score   support

           0       1.00      1.00      1.00    197867
           1       1.00      1.00      1.00    197893

    accuracy                           1.00    395760
   macro avg       1.00      1.00      1.00    395760
weighted avg       1.00      1.00      1.00    395760

Test:               precision    recall  f1-score   support

           0       0.89      0.87      0.88     84819
           1       0.87      0.90      0.88     84793

    accuracy                           0.88    169612
   macro avg       0.88      0.88      0.88    169612
weighted avg       0.88      0.88      0.88    169612
```

The decision tree model with oversampled data showed overfitting in training as well as in testing phase.



The ROC-AUC Score is 0.8312.

Tried build a ADA Boost Model

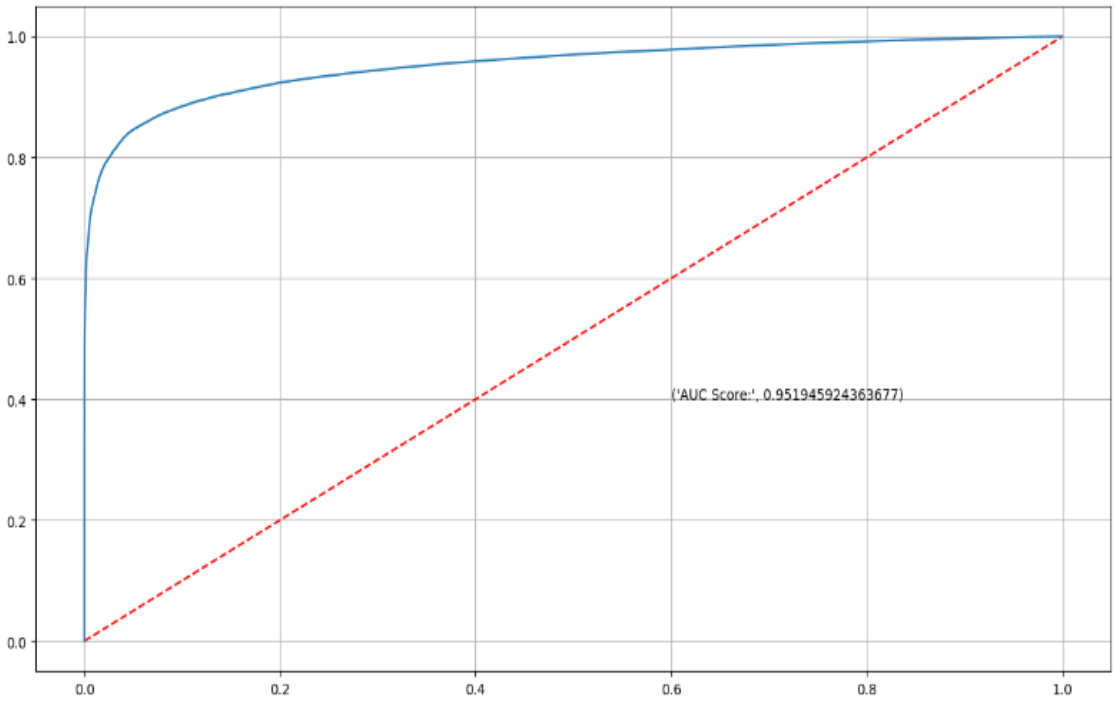
Train:

	precision	recall	f1-score	support
0	0.89	0.89	0.89	197867
1	0.89	0.89	0.89	197893
accuracy			0.89	395760
macro avg	0.89	0.89	0.89	395760
weighted avg	0.89	0.89	0.89	395760

Test:

	precision	recall	f1-score	support
0	0.89	0.89	0.89	84819
1	0.89	0.89	0.89	84793
accuracy			0.89	169612
macro avg	0.89	0.89	0.89	169612
weighted avg	0.89	0.89	0.89	169612

After building an adaboost model the results improved but are not up to the mark in both training and testing.



The ROC-AUC score for the adaptive boosting model is 0.951.

AS THE OVERSAMPLING METHOD HAS FAILED

NOW WE ARE GOING FOR THE PCA METHOD

• PRINCIPAL COMPONENTS ANALYSIS(PCA)

Principal Component Analysis (PCA) is a dimensionality reduction technique used in machine learning and data analysis to reduce the number of features (variables) in a dataset while preserving as much of the original data's variance as possible. PCA accomplishes this by transforming the original features into a new set of linearly uncorrelated variables called principal components.

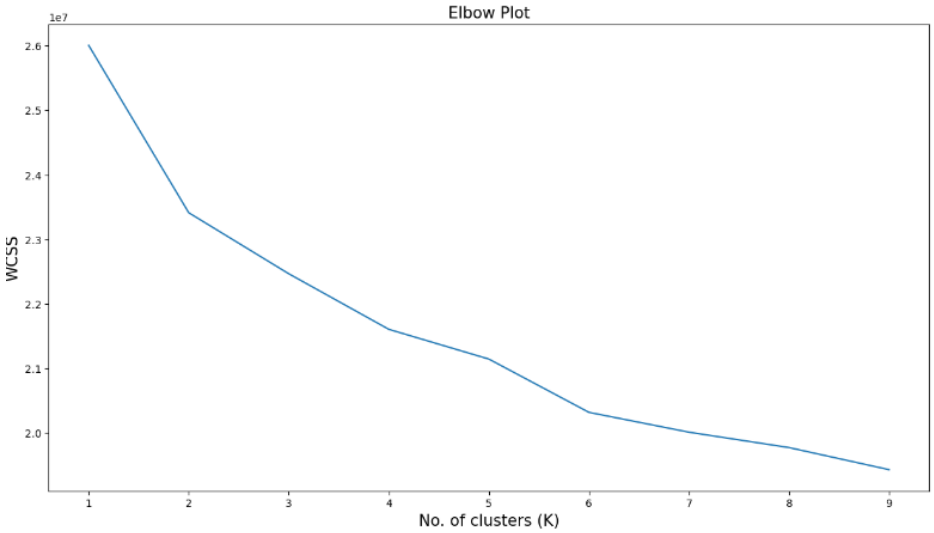
```
# with inbuild function

from sklearn.decomposition import PCA

pca = PCA(n_components=55)
model = pca.fit(df1)

print('Top 55 Eigen values :',model.explained_variance_)
print('Top 55 Eigen vectore :',model.components_)
print('Explained Variation by top 55 Components :',model.explained_variance_ratio_.sum()*100)
```

```
Top 55 Eigen values : [13.06677168  4.25766571  3.68349486  2.97797013  2.85743433  2.45572675
  2.33875675  1.88523361  1.74838732  1.64622908  1.59855134  1.46179728
  1.32292256  1.28149473  1.26588773  1.24206216  1.14183792  1.11496962
  1.09334682  1.07638731  1.05024842  1.03535769  1.02585613  1.02174353
  1.01678193  1.01391801  1.00616846  1.0045299   1.00360826  1.00208711
  1.0010942   0.99990163  0.99962369  0.99919477  0.99538541  0.99194869
  0.99087471  0.98685018  0.98174224  0.96888073  0.96340577  0.95784866
  0.94032497  0.92609602  0.90837637  0.89273963  0.87341055  0.85619634
  0.85394269  0.83316903  0.82272857  0.81172363  0.7970271   0.77712948
  0.73831572]
Top 55 Eigen vectore : [[ 4.70475710e-03 -2.45711795e-03  5.79675693e-03 ...  1.36677961e-02
   1.97325890e-04 -4.08257421e-03]
 [ 9.23135217e-04 -1.49968971e-01  1.35499269e-01 ...  1.53097422e-02
  -8.39844826e-03 -3.50840554e-02]
 [-3.41877017e-02 -9.40160081e-03  4.01329822e-02 ...  3.62022374e-02
   3.53013257e-03 -8.54041956e-03]
 ...
 [-1.25722837e-01  1.27869645e-01 -1.96119919e-01 ... -2.09096479e-02
   1.63434713e-01 -5.22477659e-01]
 [ 6.06157203e-02  1.02953656e-03 -1.09458971e-02 ...  3.19904355e-02
   2.51312909e-02 -6.07410550e-02]
 [-5.55692166e-02  5.16969423e-02  6.76175649e-03 ... -1.41694841e-02
   9.65394096e-03 -1.14807796e-01]]
Explained Variation by top 55 Components : 92.7695386963443
```

```
pca = PCA(0.95)
model = pca.fit(df)

model.n_components_
```

```
58
```

FOR 95% EXPLAINED VARIANCE THE N0. OF COMPONENTS ARE 58



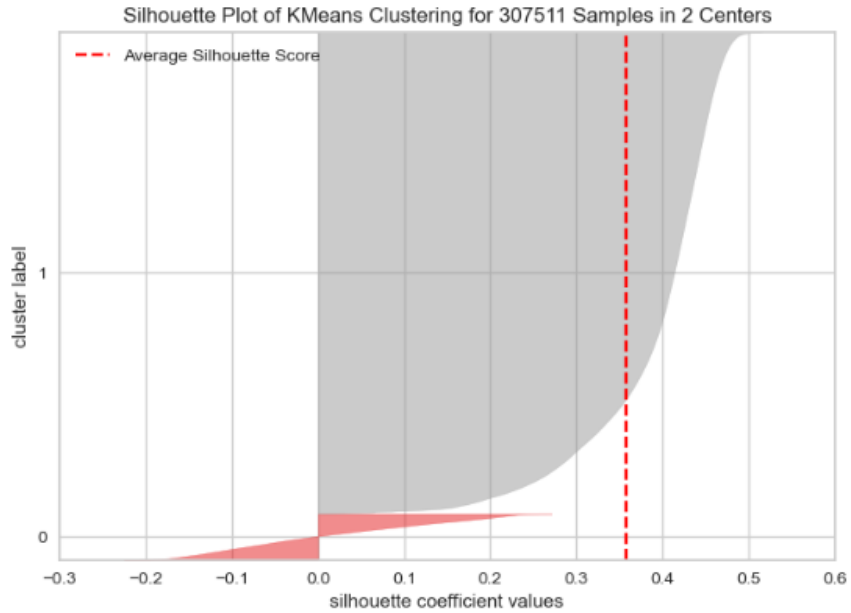
The Elbow Plot shows that the value of n_clusters could be either 2 or 4.

To check the best values amongst these we will check the silhouette score.

For 2 clusters the silhouette score is 0.35712552754309085

For 4 clusters the silhouette score is 0.05779693386381549

Out of these the most promising value of n_cluster is 2.



The Knife plot for the n_cluster = 2 value.

We can clearly see that there are no outliers in the knife pot so we can select the value of n_cluster as 2.

	precision	recall	f1-score	support
0	0.93	0.90	0.91	18757
1	0.99	0.99	0.99	196500
accuracy			0.99	215257
macro avg	0.96	0.95	0.95	215257
weighted avg	0.99	0.99	0.99	215257

	precision	recall	f1-score	support
0	0.93	0.90	0.91	8183
1	0.99	0.99	0.99	84071
accuracy			0.98	92254
macro avg	0.96	0.94	0.95	92254
weighted avg	0.98	0.98	0.98	92254

The result after doing PCA on the actual data frame showed us that again the model is tending towards getting over fit in train and testing.

Train data classification report :

	precision	recall	f1-score	support
0	0.99	1.00	1.00	18757
1	1.00	1.00	1.00	196500
accuracy			1.00	215257
macro avg	1.00	1.00	1.00	215257
weighted avg	1.00	1.00	1.00	215257

Test data classification report :

	precision	recall	f1-score	support
0	1.00	1.00	1.00	8183
1	1.00	1.00	1.00	84071
accuracy			1.00	92254
macro avg	1.00	1.00	1.00	92254
weighted avg	1.00	1.00	1.00	92254

The result after doing PCA on a dataframe which has all the PCA components(58).

Again the model is getting overfitting in training and testing.

AS THE PCA METHOD HAS ALSO FAILED

NOW WE ARE GOING FOR THE UNDERSAMPLING METHOD

LET US TRY UNDER SAMPLING

```
# Separate majority and minority classes
majority_class = df[df['TARGET'] == 0]
minority_class = df[df['TARGET'] == 1]
```

```
from sklearn.utils import resample

# Downsample the majority class
n_samples = len(minority_class)
downsampled_majority = resample(majority_class, replace=False, n_samples=80000, random_state=42)

# Combine the downsampled majority class and the minority class
balanced_data = pd.concat([downsampled_majority, minority_class])

# Your balanced dataset is now in 'balanced_data' variable
```

```
balanced_data.shape
```

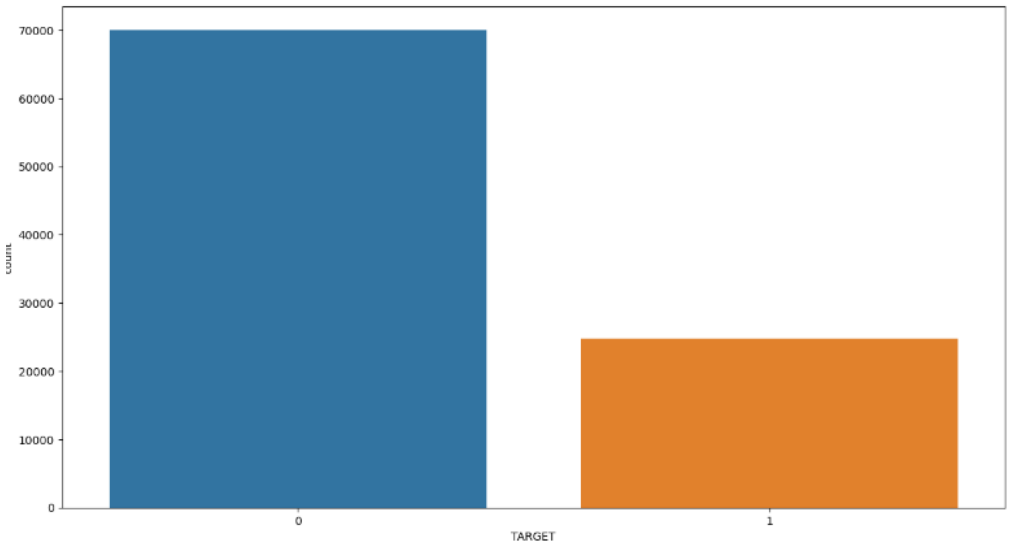
```
(104825, 90)
```

THE SHAPE OF THE DATASET IS REDUCED TO 104,825

OUT OF WHICH 80000 ARE CLASS 0

AND 24825 ARE CLASS 1

WHICH IS IN 76:23 RATIO



LET US NOW BUILT A MODEL ON THIS BALANCED DATA SET.

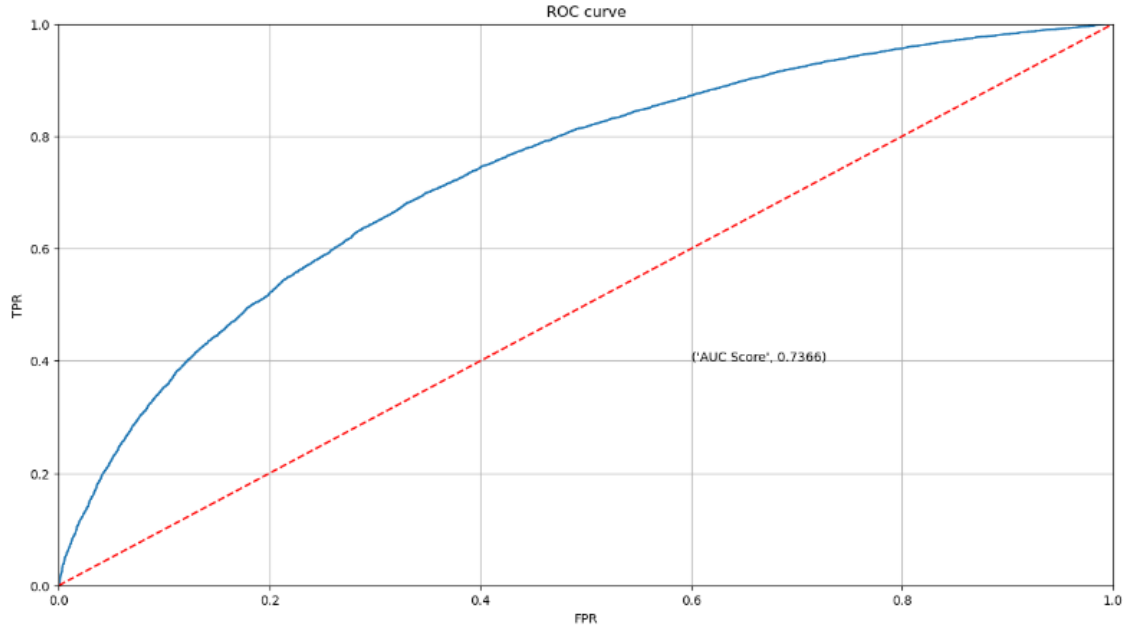
	precision	recall	f1-score	support
0	0.79	0.96	0.87	55926
1	0.59	0.21	0.31	17451
accuracy			0.78	73377
macro avg	0.69	0.58	0.59	73377
weighted avg	0.75	0.78	0.73	73377

	precision	recall	f1-score	support
0	0.80	0.96	0.87	24074
1	0.59	0.20	0.30	7374
accuracy			0.78	31448
macro avg	0.69	0.58	0.58	31448
weighted avg	0.75	0.78	0.74	31448

The result of the logistic regression model on the balanced data set looks promising.

The model has performed equally in training as well as testing.

The model is predicting well for 0 but not for class 1.



The ROC-AUC Score is 0.7366

LET US USE GRID SEARCH CV TO GET THE BEST PARAMETERS FOR XGBOOST MODEL.

GridSearchCV is useful for automatically tuning hyperparameters. Although it can be more straightforward, and less computationally expensive, to manually tune a classier's hyperparameters one parameter at a time, this can sometimes lead to an algorithm getting stuck at a local performance maxima. Indeed, there are situations where one particular combination of hyperparameter values may be superior to a different combination, but the chances of discovering the superior combination by adjusting one parameter at a time could be very remote. GridSearchCV generates a grid of hyperparameter combinations after the user species ranges of discrete values for each hyperparameter that they wish to tune. GridSearchCV then methodically trains a given classier using each unique combination of hyperparameter values, and uses K-Fold cross validation and a scoring function specified by the user to compare the hyperparameter combinations and return the combination belonging to the classier that scored highest.

```
%%time
xgb_grid = GridSearchCV(estimator =xgb_model,
                        param_grid=tuned_parameters,
                        cv=3,scoring='roc_auc')

xgb_grid_model= xgb_grid.fit(x_train , y_train)

print('Best parameters for xgb classifier:',xgb_grid_model.best_params_, '\n')
```

```
Best parameters for xgb classifier: {'gamma': 0, 'learning_rate': 0.3, 'max_depth': 2, 'n_estimators': 150, 'reg_lambda': 1}

CPU times: total: 19h 9min 1s
Wall time: 5h 1min 1s
```

```
train_pred = xgb_grid_model.predict(x_train)
print(classification_report(y_train,train_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.95      0.87     55926
           1       0.63      0.25      0.35     17451

    accuracy                           0.79     73377
   macro avg       0.72      0.60      0.61     73377
weighted avg       0.76      0.79      0.75     73377
```

```
## testingg summary
train_pred = xgb_grid_model.predict(x_test)
print(classification_report(y_test,train_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.96      0.87     24074
           1       0.61      0.23      0.33      7374

    accuracy                           0.79     31448
   macro avg       0.71      0.59      0.60     31448
weighted avg       0.76      0.79      0.75     31448
```

The result after building a model with best parameters showed good result but again for class 1 the f1-score is very less.

We will again try to tune our hyper parameters to get better scores.

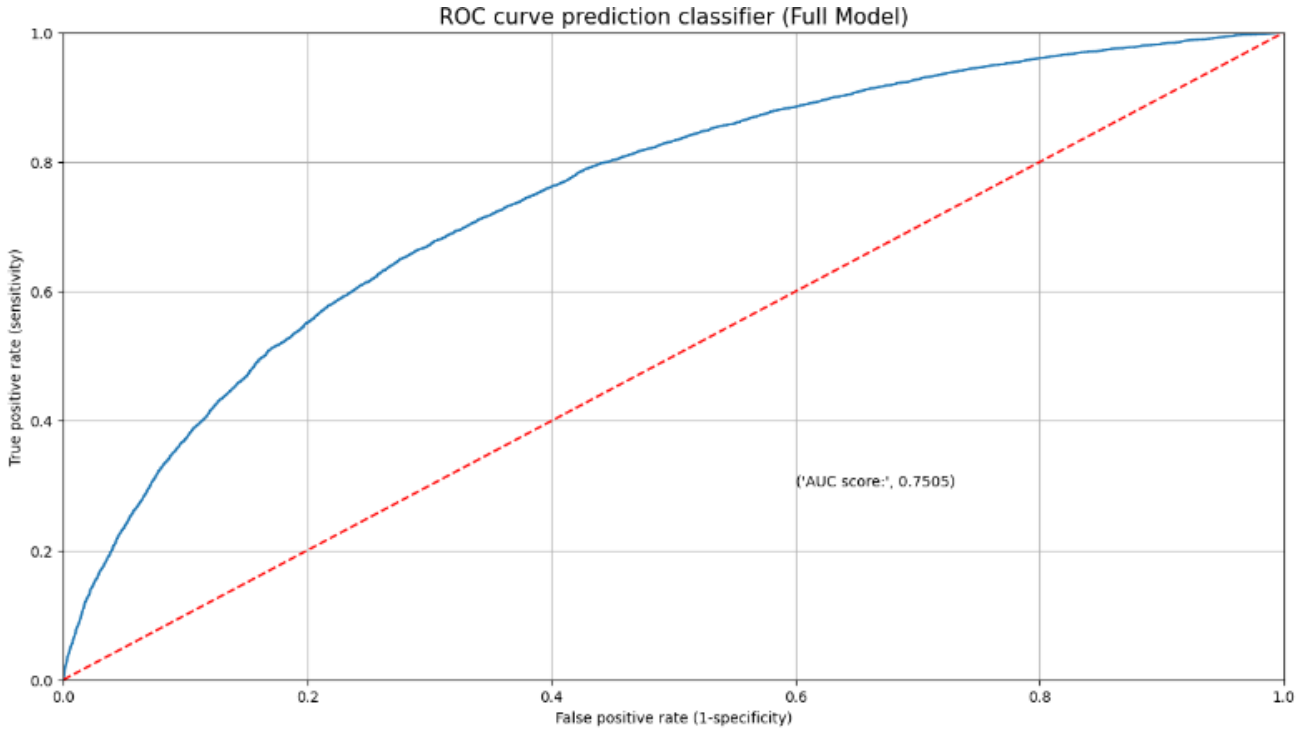
LET US AGAIN BUILT A MODEL ON THIS BALANCED DATA SET BUT WE WILL NOW REDUCE THE N_SAMPLES VALUE TO GET BETTER ACCURACY, F-1 SCORE.

SETTING N_SAMPLES = 70000

```
xgb_model = XGBClassifier(max_depth =4 , gamma = 1.3,n_estimators=170,learning_rate=0.5, reg_lambda=3)

xg_mod = xgb_model.fit(x_train,y_train)

y_pred_test = xgb_model.predict(x_test)
print('TEST',classification_report(y_test , y_pred_test))

y_pred_train = xgb_model.predict(x_train)
print('TRAIN',classification_report(y_train , y_pred_train))
```

```
TEST              precision    recall  f1-score   support

           0       0.79      0.93      0.85     20970
           1       0.60      0.31      0.41      7478

    accuracy                           0.76     28448
   macro avg       0.69      0.62      0.63     28448
weighted avg       0.74      0.76      0.74     28448

TRAIN              precision    recall  f1-score   support

           0       0.80      0.94      0.87     49030
           1       0.67      0.35      0.46     17347

    accuracy                           0.79     66377
   macro avg       0.74      0.65      0.66     66377
weighted avg       0.77      0.79      0.76     66377
```

This is the best performing model until now.



THIS IS THE BEST MODEL WHICH SHOWS THE RESULT AS FOLLOWS:-

ACCURACY TRAIN - 0.79

ACCURACY TEST - 0.76

F1 SCORE (MACRO AVG) - 0.63

AUC-ROC - 0.7505

CONCLUSION

The most challenging part of this project was understanding the dataset and its various features. We budgeted the majority of ourtime toward understanding what information each feature contained, how this information was distributed, and what sort of preprocessing techniques would need to be applied. Dealing with the dataset's overabundance of missing values, or `NaN' entries, proved to be trickiest part of the data preprocessing phase and since most machine learning algorithms aren't designed to overlook missing entries, we had to overcome this challenge.

Several Classifier techniques have been trained and the XGBOOST MODEL was the best model which was showing BEST ROC-AUC SCORE.

However the TEST accuracy of XGradient Boosting with hyperparameter tuning with 80000 samples of 1's provided the best accuracy test score.

The Logistic Regression model was also performing fine but was not able to Predict 1's.