Capstone Project Customer Churn – Notes 2

TEJAS PADEKAR
PGP-DSBA Online
Date: 15/01/2023

CONTENTS:

List of Figures

Summary of Project Note 1

- There was some bad data which was converted to Nan values and then imputed appropriately.
- The dataset contained null and missing values which were imputed appropriately by imputing the median values for the continuous variables and mode for the categorical variables.
- Thorough EDA was performed along with Univariate and Bivariate analysis of the variables.
- Accurate data type was provided to each variable post imputation and all the variables in our dataset are now either integer or float type which is essential to build our classification models. We used Label encoding on the categorical variables to convert them from object data type to Integers.
- The outliers for the continuous variables were also treated basis lower limit of 5% and upper limit of 95% quantile.
- We dropped the variable 'AccountID' as it was not going to add any significance to the prediction. Hence, now we have 17 predictor variables and 1 target variable in our dataset.
- We split the data into two sets:
  - i. xo and yo split contains unscaled observations and contains outliers.
  - ii. x and y split contains unscaled observations but with outliers treated.
- xo/x captures all predictor variables and yo/y captures the target variable 'Churn'.
- We observed the proportion of our target variable is 83% belonging to class 0 (customers who have not churned) and 17% belonging to class 1 (customers who have churned).
- Objective is to develop a model through which the business can do churn prediction of the accounts and provide segmented offers to the potential churners.

Prerequisites to Model Building

i. Train Test Split:

- We have split both xo yo and x y datasets into 70% training dataset and 30% testing dataset with random_state which makes the random split results reproducible and we end up using the same data for every training and testing for all the models built.
- We also stratify the dataset yo and y having the target variable because this data seems imbalanced and could possibly result into different proportions in the yo/y variable between train and test sets.
- Post split we have 7882 observations in our train dataset and 3378 observations in test dataset for each of the two sets.

```
xo_train:  (7882, 17)        x_train:  (7882, 17)
xo_test:  (3378, 17)         x_test:  (3378, 17)
yo_train:  (7882,)           y_train:  (7882,)
yo_test:  (3378,)            y_test:  (3378,)
```

Figure 1: Train and Test Split Proportions

ii. Variable Scaling:

- Some of the model building techniques especially distance based techniques such as KNN and Support Vector Machines need the dataset to be scaled in order to avoid performance issues for their models and also a few other modelling techniques even though do not require scaling but their performances benefit from having a scaled dataset. Hence, we have scaled one of our datasets (x y) only.
- We used StandardScaler to change the features to the same scale. After standardization, each feature has zero mean and standard deviation of 1.
- Standardization is fit on the training dataset only (x_train). Then, the test dataset (x_test) is standardized using the fitting results from the training dataset.

iii. SMOTE:

- We may face a data imbalance problem during our model building due to lower no of class 1 observations. In case of this, we will use SMOTE technique to create a balanced dataset in which the minority class ie; class 1 for our dataset would be over sampled by generating synthesized data and perform model building.
- We have kept the sampling_strategy at 50% ie; for every two observations of class 0, one synthesized observation of class 1 would be generated.
- We have also created a Train Test Split on scaled dataset of x dataset and fitted it to SMOTE as shown below to build models using SMOTE requiring scaled dataset on both train and test for eg; KNN models and SVM Models

```
xs_train:  (7882, 17)
xs_test:  (3378, 17)
ys_train:  (7882,)
ys_test:  (3378,)
```

1). Model building and interpretation

Model Building Approach:

- Prominent Classification Problem related modelling techniques such as Logistic Regression, Linear Discriminant Analysis (LDA), CART, Random Forest, Naïve Bayes, KNN, SVM and ANN were built and predicted on both train and test datasets after which the performances were tabulated using necessary performance measures such as accuracy, recall, precision, f-1 score and AUC.
- Prime emphasis was to build models which can achieve high recall value on test datasets for class 1 observations (customers who have churned).
- First a base model for one of the above techniques was created and evaluated.
- Then, the same model was tuned and hyper tuned multiple times until either a satisfactory result w.r.t the performance measures was achieved or we had to discard the model due to under performance.
- Similarly, base models were built on various ensemble techniques such as Bagging, AdaBoost, GradientBoosting and XGBoost as well and later hyper tuned.
- Post building the models, we have performed 10 fold cross validation to check if our models are valid or are over-fitting.
- Finally, SMOTE was applied to the models which had both under performed or had satisfactory results to check whether their performance improves and also to check if these models are effected by imbalance in the dataset or not.

a) Build various models (You can choose to build models for either or all of descriptive, predictive or prescriptive purposes)

- Below is a chart for all the built and maintained models. We have excluded ANN, Naïve Bayes and LDA models, and other models post multiple tuning and hyper tuning attempts as they had under-fitted on both train and test or/and under-performed. A few of these are however retained in the jupyter notebook attached along with this file for reference.

Model Name	Accuracy		Recall		Precision		F1		AUC		Remarks
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	
Ada Boost (unscaled data without Outliers)	0.90	0.90	0.61	0.60	0.76	0.77	0.67	0.68	0.93	0.93	Poor Recall
Gradient Boosting (unscaled data and without outliers)	0.92	0.91	0.64	0.61	0.84	0.83	0.73	0.70	0.95	0.93	Poor Recall
SVM (scaled dataset without Outliers)	0.94	0.94	0.69	0.65	0.95	0.95	0.80	0.77	0.97	0.94	Poor Recall
Logistic Regression_model_3 with 0.236 optimal threshold	0.83	0.82	0.77	0.72	0.50	0.48	0.61	0.58	0.80	0.80	Poor Recall
Random Forest_model 4 - Hyper Tuned (unscaled data and with outliers)	0.97	0.95	0.83	0.72	0.98	0.94	0.90	0.82	0.99	0.98	Poor Recall
Random Forest_model 3 - Hyper Tuned (unscaled data and with outliers)	0.97	0.95	0.86	0.74	0.99	0.95	0.92	0.83	0.99	0.98	Poor Recall
Logistic Regression_model_4 with 0.219 optimal threshold	0.81	0.81	0.77	0.75	0.46	0.45	0.58	0.56	0.79	0.79	Poor Precision
Random Forest_model 5 - Hyper Tuned (unscaled data and with outliers)	0.98	0.95	0.87	0.75	0.99	0.95	0.93	0.84	0.99	0.98	Slight Overfit
Logistic Regression_model_3 with 0.19 threshold	0.80	0.79	0.81	0.76	0.45	0.42	0.57	0.55	0.80	0.80	Poor Precision
Logistic Regression_model_4 with 0.19 threshold	0.78	0.78	0.79	0.77	0.42	0.42	0.55	0.54	0.79	0.79	Poor Precision
CART Tuned (Unscaled Dataset without Outliers)	0.97	0.94	0.89	0.78	0.95	0.83	0.92	0.81	0.99	0.99	Slight Overfit
CART Tuned with SMOTE (Unscaled Dataset without Outliers)	0.97	0.92	0.93	0.78	0.97	0.77	0.95	0.77	0.99	0.99	Overfit
Bagging (unscaled data and without outliers)	0.98	0.95	0.92	0.79	0.99	0.92	0.95	0.85	0.99	0.98	Overfit
KNN_model (scaled data without outliers)	0.98	0.95	0.90	0.80	0.97	0.89	0.93	0.84	0.99	0.99	Good Model
SVM with SMOTE (scaled dataset without Outliers)	0.95	0.93	0.92	0.82	0.93	0.76	0.92	0.79	0.99	0.99	Good Model
Random Forest_model (unscaled data with outliers)	1.00	0.97	1.00	0.85	1.00	0.98	1.00	0.91	1.00	0.99	Overfit
Random Forest_model 5 - Hyper Tuned with SMOTE (unscaled data and with outliers)	0.98	0.95	0.96	0.85	0.98	0.84	0.97	0.85	0.99	0.98	Slight Overfit
CART (Unscaled Dataset without Outliers)	1.00	0.95	1.00	0.85	1.00	0.85	1.00	0.85	1.00	1.00	Overfit
Bagging with SMOTE (unscaled data and without outliers)	0.99	0.95	0.97	0.85	0.99	0.85	0.98	0.85	0.99	0.98	Slight Overfit
XGBoost (unscaled data and without outliers)	1.00	0.97	1.00	0.85	1.00	0.96	1.00	0.90	1.00	0.99	Overfit
Ada Boost Tuned (unscaled data without Outliers)	1.00	0.98	1.00	0.87	1.00	0.98	1.00	0.92	1.00	1.00	Overfit
Gradient Boosting Tuned (unscaled data and without outliers)	1.00	0.98	1.00	0.87	1.00	0.99	1.00	0.92	1.00	0.99	Overfit
Random Forest_model 6 - Hyper Tuned (unscaled data and with outliers)	1.00	0.98	1.00	0.88	1.00	0.99	1.00	0.93	1.00	0.99	Overfit
XGBoost Tuned (unscaled data and without outliers)	1.00	0.97	0.99	0.88	0.99	0.92	0.99	0.90	1.00	0.99	Slight Overfit

SVM Tuned (scaled dataset without Outliers)	1.00	0.98	0.99	0.91	1.00	0.95	0.99	0.93	0.99	0.99	Good Model
KNN_model with SMOTE (scaled data without outliers)	0.98	0.94	0.99	0.93	0.95	0.76	0.97	0.84	0.99	0.99	Good Model

Figure 2: All models comparison chart

- We have arranged the models in ascending order basis their respective recall scores on test dataset and have mentioned necessary remarks in front of them as per the model's performance.
- Cells highlighted in green consists of the models which have performed overall well on both train and test datasets compared to the other models.
- While there are a few models which have a better recall score than the models highlighted in green, they are not considered as well performed models as they have overfit on the test dataset. Eg; Random Forest_model 6 - Hyper Tuned (4th from bottom) has a test recall of 88% which is higher than 9 out of the 12 models we have highlighted in green. However, we have not considered it as well performed as it has completely over-fitted ie; having good performance on train dataset but poor performance on test dataset. Similarly, we have eliminated all the models which have over-fitted completely ie; 100% result on train dataset but poor result (less than 90%) on test dataset to make the analysis more relevant and focused.
- We will now reflect on the various models built. However, we will select only specific models from the above models highlighted in green which have not over-fitted and have well performed on both train and test datasets across all performance measures such as accuracy, recall, precision, f-1 score and AUC for class 1.
- Below is the chart of the models eliminated from the above models with reasons in remarks column.

Model Name	Accuracy		Recall		Precision		F1		AUC		Remarks
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	
Ada Boost (unscaled data without Outliers)	0.90	0.90	0.61	0.60	0.76	0.77	0.67	0.68	0.93	0.93	Poor Recall
Gradient Boosting (unscaled data and without outliers)	0.92	0.91	0.64	0.61	0.84	0.83	0.73	0.70	0.95	0.93	Poor Recall
SVM (scaled dataset without Outliers)	0.94	0.94	0.69	0.65	0.95	0.95	0.80	0.77	0.97	0.94	Poor Recall
Logistic Regression_model_3 with 0.236 optimal threshold	0.83	0.82	0.77	0.72	0.50	0.48	0.61	0.58	0.80	0.80	Poor Recall
Random Forest_model 4 - Hyper Tuned (unscaled data and with outliers)	0.97	0.95	0.83	0.72	0.98	0.94	0.90	0.82	0.99	0.98	Poor Recall
Random Forest_model 3 - Hyper Tuned (unscaled data and with outliers)	0.97	0.95	0.86	0.74	0.99	0.95	0.92	0.83	0.99	0.98	Poor Recall
Logistic Regression_model_4 with 0.219 optimal threshold	0.81	0.81	0.77	0.75	0.46	0.45	0.58	0.56	0.79	0.79	Poor Precision
Random Forest_model 5 - Hyper Tuned (unscaled data and with outliers)	0.98	0.95	0.87	0.75	0.99	0.95	0.93	0.84	0.99	0.98	Slight Overfit
Logistic Regression_model_3 with 0.19 threshold	0.80	0.79	0.81	0.76	0.45	0.42	0.57	0.55	0.80	0.80	Poor Precision
Logistic Regression_model_4 with 0.19 threshold	0.78	0.78	0.79	0.77	0.42	0.42	0.55	0.54	0.79	0.79	Poor Precision
CART Tuned with SMOTE (Unscaled Dataset without Outliers)	0.97	0.92	0.93	0.78	0.97	0.77	0.95	0.77	0.99	0.99	Overfit
Bagging (unscaled data and without outliers)	0.98	0.95	0.92	0.79	0.99	0.92	0.95	0.85	0.99	0.98	Overfit
Random Forest_model (unscaled data with outliers)	1.00	0.97	1.00	0.85	1.00	0.98	1.00	0.91	1.00	0.99	Overfit
CART (Unscaled Dataset without Outliers)	1.00	0.95	1.00	0.85	1.00	0.85	1.00	0.85	1.00	1.00	Overfit
Bagging with SMOTE (unscaled data and without outliers)	0.99	0.95	0.97	0.85	0.99	0.85	0.98	0.85	0.99	0.98	Slight Overfit
XGBoost (unscaled data and without outliers)	1.00	0.97	1.00	0.85	1.00	0.96	1.00	0.90	1.00	0.99	Overfit

Ada Boost Tuned (unscaled data without Outliers)	**1.00**	**0.98**	**1.00**	**0.87**	**1.00**	**0.98**	**1.00**	**0.92**	**1.00**	**1.00**	**Overfit**
Gradient Boosting Tuned (unscaled data and without outliers)	**1.00**	**0.98**	**1.00**	**0.87**	**1.00**	**0.99**	**1.00**	**0.92**	**1.00**	**0.99**	**Overfit**
Random Forest_model 6 - Hyper Tuned (unscaled data and with outliers)	**1.00**	**0.98**	**1.00**	**0.88**	**1.00**	**0.99**	**1.00**	**0.93**	**1.00**	**0.99**	**Overfit**

Figure 3: Models eliminated from evaluation

- Below is the chart of the selected models for evaluation and interpretation. We have purposefully included a few models which have a diff of +/- 11% of recall between train and test as the important features extracted from analysis of these models can also be crucial for business recommendations.

Model Name	**Accuracy**		**Recall**		**Precision**		**F1**		**AUC**		**Remarks**
	Train	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**	
CART Tuned (Unscaled Dataset without Outliers)	**0.97**	**0.94**	**0.89**	**0.78**	**0.95**	**0.83**	**0.92**	**0.81**	**0.99**	**0.99**	**Slight Overfit**
KNN_model (scaled data without outliers)	**0.98**	**0.95**	**0.90**	**0.80**	**0.97**	**0.89**	**0.93**	**0.84**	**0.99**	**0.99**	**Good Model**
SVM with SMOTE (scaled dataset without Outliers)	**0.95**	**0.93**	**0.92**	**0.82**	**0.93**	**0.76**	**0.92**	**0.79**	**0.99**	**0.99**	**Good Model**
Random Forest_model 5 - Hyper Tuned with SMOTE (unscaled data and with outliers)	**0.98**	**0.95**	**0.96**	**0.85**	**0.98**	**0.84**	**0.97**	**0.85**	**0.99**	**0.98**	**Slight Overfit**
XGBoost Tuned (unscaled data and without outliers)	**1.00**	**0.97**	**0.99**	**0.88**	**0.99**	**0.92**	**0.99**	**0.90**	**1.00**	**0.99**	**Slight Overfit**
SVM Tuned (scaled dataset without Outliers)	**1.00**	**0.98**	**0.99**	**0.91**	**1.00**	**0.95**	**0.99**	**0.93**	**0.99**	**0.99**	**Good Model**
KNN_model with SMOTE (scaled data without outliers)	**0.98**	**0.94**	**0.99**	**0.93**	**0.95**	**0.76**	**0.97**	**0.84**	**0.99**	**0.99**	**Good Model**

Figure 4: Models selected for evaluation

b) Test your predictive model against the test set using various appropriate performance metrics

- Before we get into model evaluation, there are a few basic information related to the performance measures that we should be aware of to interpret the results of the predictions made on train and test datasets as given below:

Understanding Confusion Matrix

- **True Negative (TN)** : A data point belongs to negative class (class 0) and is correctly predicted as negative. Displayed at top left of confusion matrix.
- **True Positive (TP)**: A data point belongs to positive class (class 1) and is correctly predicted as positive. Displayed at bottom right of confusion matrix.
- **False Negative (FN)**: A data point belongs to positive class (class 1) but is incorrectly predicted negative (class 0). Displayed at bottom left of confusion matrix.
- **False Positive (FP)**: A data point belongs to negative class (class 0) but is incorrectly predicted as positive (class 1). Displayed at top right of confusion matrix.

Understanding Classification Report

- Precision is how accurately does the model predict a data point to its respective class correctly.
- Recall is how many data points were correctly identified to their respective class.
- F1 score represents how many percentage of positive predictions were correct.
- Accuracy is how well the model is predicting all the data points to their respective classes correctly.
- AUC (Area under Curve) denotes if the model is capable of distinguishing between class 0 and class 1.

I. CART Tuned (Unscaled Dataset without Outliers - x_train & x_test)

- The Cart Tuned Model built was tuned as per the below:
    a) criterion = 'gini'
    b) min_samples_leaf = 4

Predictions on Train Dataset

```
Accuracy Score is 0.9723420451662015

Confusion Matrix
 [[6487   68]
 [ 150 1177]]

Classification Report
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      6555
           1       0.95      0.89      0.92      1327

    accuracy                           0.97      7882
   macro avg       0.96      0.94      0.95      7882
weighted avg       0.97      0.97      0.97      7882
```

```
AUC: 0.997
```

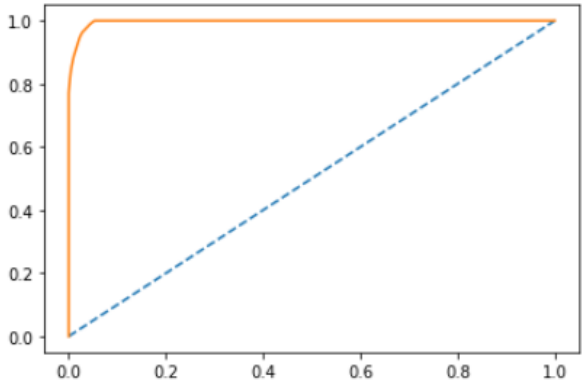


Figure 5: Performance Measure of CART Tuned Model on Train Dataset

- We can see that the Accuracy score is high which states that the model has correctly predicted 97% of total cases including both class 0 and class 1 as well.
- The confusion matrix suggests that the model has:
  a) correctly identified 1177 data points for class 1 (True Positives) of the total 1327
  b) 150 class 1 data points have been incorrectly identified as belonging to class 0 (False Negatives)
  c) correctly identified 6487 data points for class 0 (True Negatives) out of 6555
  d) 68 class 0 data points (False Positives) have been incorrectly identified as belonging to class 1
- The classification report suggests that the model has:
  a) Identified 89% of real True Positives of all data points (Recall for Class 1) and 99% of real True Negatives of all data points (Recall for Class 0)
  b) Identified real True Positives as True Positive correctly 95% of the times of all correctly identified data points (Precision for Class 1) and real True Negatives as True Negative 98% of the times of all correctly identified data points (Precision for Class 0)
- AUC (Area under Curve) is high at 99.7% which means that the model is very highly capable of distinguishing between class 0 and class 1.

Predictions on Test Dataset

```
Accuracy Score is 0.9363528715216104

Confusion Matrix
 [[2718   91]
 [ 124  445]]

Classification Report
              precision    recall  f1-score   support

           0       0.96      0.97      0.96      2809
           1       0.83      0.78      0.81       569

    accuracy                           0.94      3378
   macro avg       0.89      0.87      0.88      3378
weighted avg       0.94      0.94      0.94      3378
```

```
AUC: 0.997
```

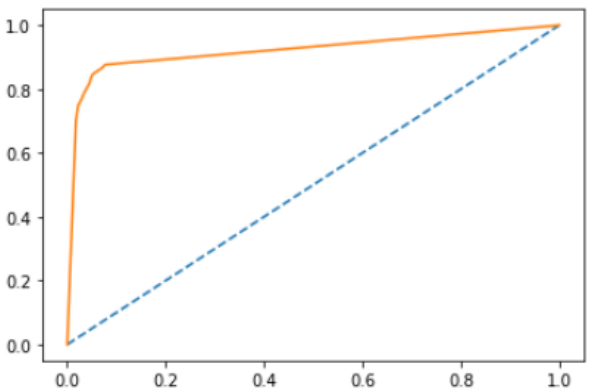


Figure 6: Performance Measure of CART Tuned Model on Test Dataset

- We can see that the Accuracy score is high which states that the model has correctly predicted 94% of total cases including both class 0 and class 1 as well.
- The confusion matrix suggests that the model has:
  a) correctly identified 445 data points for class 1 (True Positives) of the total 569
  b) 124 class 1 data points have been incorrectly identified as belonging to class 0 (False Negatives)
  c) correctly identified 2718 data points for class 0 (True Negatives) out of 2809
  d) 91 class 0 data points have been incorrectly identified as belonging to class 1 (False Positives)
- The classification report suggests that the model has:
  a) Identified 78% of real True Positives of all data points (Recall for Class 1) and 97% of real True Negatives of all data points (Recall for Class 0)
  b) Identified real True Positives as True Positive correctly 83% of the times of all correctly identified data points (Precision for Class 1) and real True Negatives as True Negative 96% of the times of all correctly identified data points (Precision for Class 0)
- AUC (Area under Curve) is high at 99.7% which means that the model is very highly capable of distinguishing between class 0 and class 1.

Model Name	Accuracy		Recall		Precision		F1		AUC	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
CART Tuned (Unscaled Dataset without Outliers)	0.97	0.94	0.89	0.78	0.95	0.83	0.92	0.81	0.99	0.99

Figure 7: CART Tuned Model Train and Test Comparison

Cross Validation Score for Train Dataset:

```
array([0.9252218 , 0.91634981, 0.92258883, 0.94035533, 0.92893401,
       0.93527919, 0.90736041, 0.91751269, 0.93020305, 0.92766497])
```

Cross Validation Score for Test Dataset:

```
array([0.8964497 , 0.86982249, 0.86686391, 0.87869822, 0.86390533,
       0.91420118, 0.90828402, 0.87869822, 0.87240356, 0.88130564])
```

- Recall has dropped from 89% on Train dataset to 78% on Test dataset.
- Accuracy is nearly similar with 97% for Train dataset and 94% on Test dataset.
- Precision has dropped from 95% on Train dataset to 83% on Test dataset.
- The F-1 score has dropped from 92% on Train dataset to 81% on Test dataset.
- AUC is equal for both train and test datasets at 99% each.
- After 10 fold cross validation, scores on train is almost same but vary for the test dataset.

II. KNN_model (scaled data without outliers - x_scaled_train & x_test)

Predictions on Train Dataset

```
Accuracy Score is 0.9775437706165948

Confusion Matrix
 [[6512   43]
 [ 134 1193]]

Classification Report
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      6555
           1       0.97      0.90      0.93      1327

    accuracy                           0.98      7882
   macro avg       0.97      0.95      0.96      7882
weighted avg       0.98      0.98      0.98      7882
```

AUC: 0.996

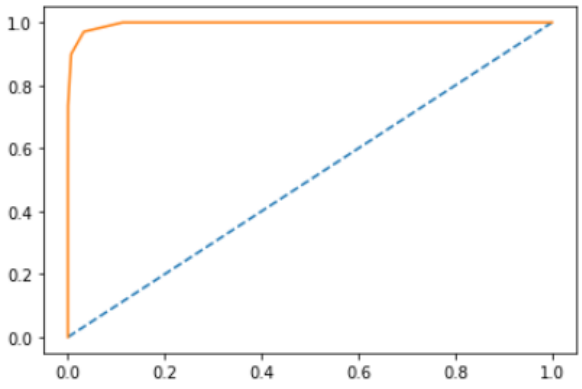


Figure 8: Performance Measure of KNN_Model on Train Dataset

- We can see that the Accuracy score is high which states that the model has correctly predicted 98% of total cases including both class 0 and class 1 as well.
- The confusion matrix suggests that the model has:
  a) correctly identified 1193 data points for class 1 (True Positives) of the total 1327.
  b) 134 class 1 data points have been incorrectly identified as belonging to class 0. (False Negatives)
  c) correctly identified 6512 data points for class 0 (True Negatives) out of 6555.
  d) 43 class 0 data points have been incorrectly identified as belonging to class 1. (False Positives)
- The classification report suggests that the model has:
  a) Identified 90% of real True Positives of all data points (Recall for Class 1) and 99% of real True Negatives of all data points (Recall for Class 0)
  b) Identified real True Positives as True Positive correctly 97% of the times of all correctly identified data points (Precision for Class 1) and real True Negatives as True Negative 98% of the times of all correctly identified data points (Precision for Class 0)
- AUC (Area under Curve) is high at 99.6% which means that the model is very highly capable of distinguishing between class 0 and class 1.

Predictions on Test Dataset

```
Accuracy Score is 0.9490822972172883

Confusion Matrix
 [[2753   56]
 [ 116  453]]

Classification Report
              precision    recall  f1-score   support

           0       0.96      0.98      0.97      2809
           1       0.89      0.80      0.84       569

    accuracy                           0.95      3378
   macro avg       0.92      0.89      0.91      3378
weighted avg       0.95      0.95      0.95      3378
```

AUC: 0.996

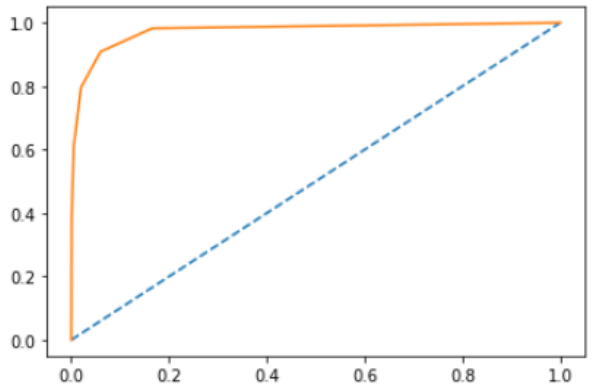


Figure 9: Performance Measure of KNN_Model on Test Dataset

- We can see that the Accuracy score is high which states that the model has correctly predicted 95% of total cases including both class 0 and class 1 as well.
- The confusion matrix suggests that the model has:
  a) correctly identified 453 data points for class 1 (True Positives) of the total 569

b) 116 class 1 data points have been incorrectly identified as belonging to class 0 (False Negatives)
c) correctly identified 2753 data points for class 0 (True Negatives) out of 2809
d) 56 class 0 data points have been incorrectly identified as belonging to class 1 (False Positives)

- The classification report suggests that the model has:
  a) Identified 80% of real True Positives of all data points (Recall for Class 1) and 98% of real True Negatives of all data points (Recall for Class 0)
  b) Identified real True Positives as True Positive correctly 89% of the times of all correctly identified data points (Precision for Class 1) and real True Negatives as True Negative 96% of the times of all correctly identified data points (Precision for Class 0)
- AUC (Area under Curve) is high at 99.6% which means that the model is very highly capable of distinguishing between class 0 and class 1.

Model Name	Accuracy		Recall		Precision		F1		AUC	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
KNN_model (scaled data without outliers)	0.98	0.95	0.90	0.80	0.97	0.89	0.93	0.84	0.99	0.99

Figure 10: KNN_Model Train and Test Comparison

Cross Validation Score for Train Dataset:

```
array([0.9391635 , 0.9531052 , 0.94162437, 0.94923858, 0.94416244,
       0.95939086, 0.93020305, 0.95177665, 0.95050761, 0.95812183])
```

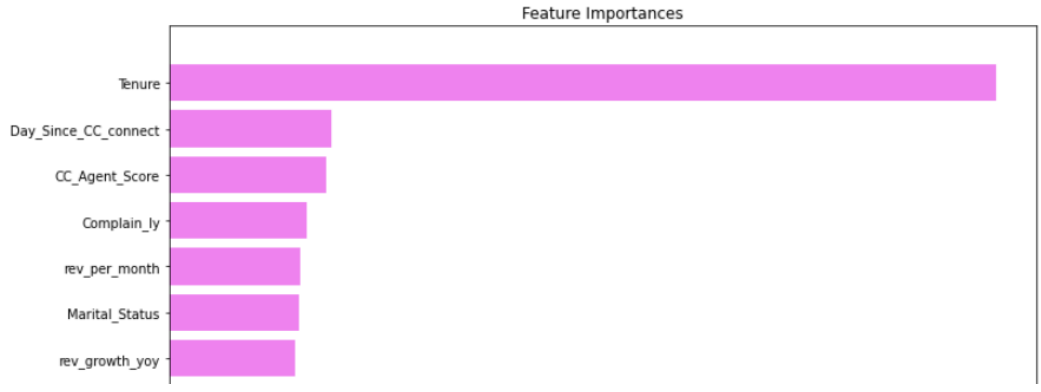
Cross Validation Score for Test Dataset:

```
array([0.8816568 , 0.86390533, 0.88461538, 0.90828402, 0.86094675,
       0.8964497 , 0.8964497 , 0.86390533, 0.87240356, 0.92284866])
```

- Recall is 90% on Train dataset and 80% on Test dataset.
- Accuracy is nearly similar with 98% for Train dataset and 95% on Test dataset.
- Precision is 97% on Train and 89% on Test dataset.
- The F-1 score has dropped from 93% on Train dataset to 84% on Test dataset.
- AUC is equal for both train and test datasets at 99% each.
- After 10 fold cross validation, scores both on train and test data set respectively for all 10 folds are almost same.

c) Interpretation of the model (s)

- We will now check the important features identified by the CART Tuned model below.
- However, we will not be able to check important features for KNN model as it is distance based algorithm and under this method a particular data point is classified to a particular class 0 or 1 basis its closeness to other data points known as neighbours. KNN models do not have any parameters or coefficients that can be learned, without which we can't extract important features.



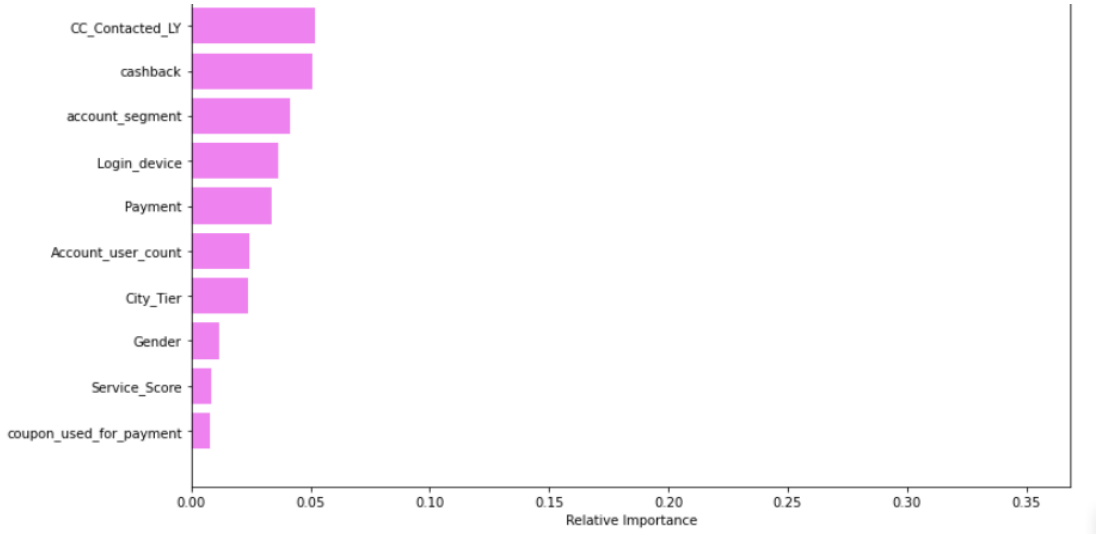


Figure 11: Important features for CART Tuned Model

Below are the most Important Features for CART Model (descending order)

- Tenure – Duration of a customer with business depicting his/her loyalty has the highest weightage of importance for this model. The longer a customer is associated with business the less likely he/she would Churn.
- Day_Since_CC_connect – Gap between customer's last connect with customer care agent. Higher the gap indicates that an active customer is not having major issues with the services and is able to make purchases seamlessly.
- CC_Agent_Score – A better satisfaction score given to customer care is preferred. Focus needs to be to get the highest rating of 5 from as many customers as possible in a genuine way. For this the company can design specific surveys, request for product reviews and
- Complain_ly - It is extremely necessary to avoid any complaints from customers or/and if any complaints are received then to resolve it in a swift and smooth manner.
- rev_per_month – Customer who are churning are contributing higher to revenue generated per month.

2). Model Tuning and business implication

a) Ensemble modelling

Model Name	Accuracy		Recall		Precision		F1		AUC		Remarks
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	
Ada Boost (unscaled data without Outliers)	0.90	0.90	0.61	0.60	0.76	0.77	0.67	0.68	0.93	0.93	Poor Recall
Gradient Boosting (unscaled data and without outliers)	0.92	0.91	0.64	0.61	0.84	0.83	0.73	0.70	0.95	0.93	Poor Recall
Bagging (unscaled data and without outliers)	0.98	0.95	0.92	0.79	0.99	0.92	0.95	0.85	0.99	0.98	Overfit
Bagging with SMOTE (unscaled data and without outliers)	0.99	0.95	0.97	0.85	0.99	0.85	0.98	0.85	0.99	0.98	Slight Overfit
XGBoost (unscaled data and without outliers)	1.00	0.97	1.00	0.85	1.00	0.96	1.00	0.90	1.00	0.99	Overfit
Ada Boost Tuned (unscaled data without Outliers)	1.00	0.98	1.00	0.87	1.00	0.98	1.00	0.92	1.00	1.00	Overfit
Gradient Boosting Tuned (unscaled data and without outliers)	1.00	0.98	1.00	0.87	1.00	0.99	1.00	0.92	1.00	0.99	Overfit
XGBoost Tuned (unscaled data and without outliers)	1.00	0.97	0.99	0.88	0.99	0.92	0.99	0.90	1.00	0.99	Slight Overfit

Figure 12: List of Ensemble Models Built

- We has built several of the above ensemble techniques. However, most of these models have overfit or returned a poor recall as per mentioned under remarks column. Hence, we would evaluate only the XGBoost model which is highlighted in green.

III. XGBoost Tuned (unscaled data and without outliers - x_train & x_test)

- The XGBoost Tuned built was tuned as per the below parameters using GridSearchCV:

```
param_grid={'colsample_bylevel': [0.5, 0.7, 0.9, 1],
            'colsample_bytree': [0.5, 0.7, 0.9, 1],
            'gamma': [0, 1, 3],
            'learning_rate': [0.01, 0.1, 0.2, 0.05],
            'n_estimators': array([10, 30, 50, 70, 90]),
            'scale_pos_weight': [0, 1, 2, 5],
            'subsample': [0.5, 0.7, 0.9, 1]})
```

- Best Parameters Identified:

```
{'colsample_bylevel': 0.9,
 'colsample_bytree': 1,
 'gamma': 0,
 'learning_rate': 0.2,
 'n_estimators': 90,
 'scale_pos_weight': 2,
 'subsample': 1}
```

Predictions on Train Dataset

```
Accuracy Score is 0.9963207307789901

Confusion Matrix
 [[6535   20]
 [   9 1318]]

Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6555
           1       0.99      0.99      0.99      1327

    accuracy                           1.00      7882
   macro avg       0.99      1.00      0.99      7882
weighted avg       1.00      1.00      1.00      7882
```

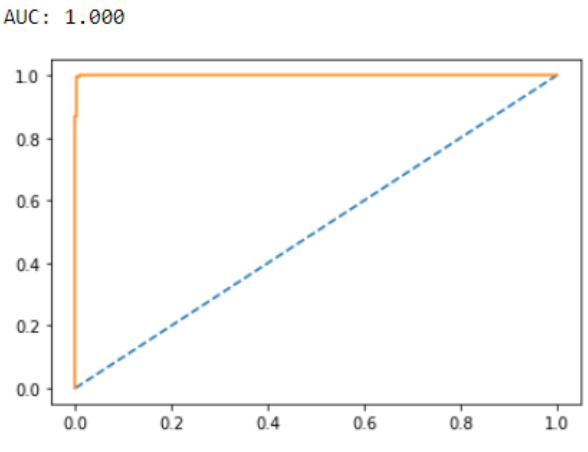


Figure 13: Performance Measure of XGBoost Tuned Model on Train Dataset

- We can see that the Accuracy score is high which states that the model has correctly predicted 99.6% of total cases including both class 0 and class 1 as well.
- The confusion matrix suggests that the model has:
  a) correctly identified 1318 data points for class 1 (True Positives) belonging to churners of the total 1327
  b) Only 9 class 1 data points have been incorrectly identified as belonging to class 0. (False Negatives)
  c) correctly identified 6535 data points for class 0 (True Negatives) out of 6555.
  d) 20 class 0 data points have been incorrectly identified as belonging to class 1 (False Positives)
- The classification report suggests that the model has:

a) Identified 99% of real True Positives of all data points (Recall for Class 1) and almost 100% of real True Negatives of all data points (Recall for Class 0)
b) Identified real True Positives as True Positive correctly 99% of the times of all correctly identified data points (Precision for Class 1) and real True Negatives as True Negative almost 100% of the times of all correctly identified data points (Precision for Class 0)

- AUC (Area under Curve) is high at almost 100% which means that the model is very highly capable of distinguishing between class 0 and class 1.

Predictions on Test Dataset

```
Accuracy Score is 0.9671403197158082

Confusion Matrix
 [[2766   43]
 [  68  501]]

Classification Report
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      2809
           1       0.92      0.88      0.90       569

    accuracy                           0.97      3378
   macro avg       0.95      0.93      0.94      3378
weighted avg       0.97      0.97      0.97      3378
```

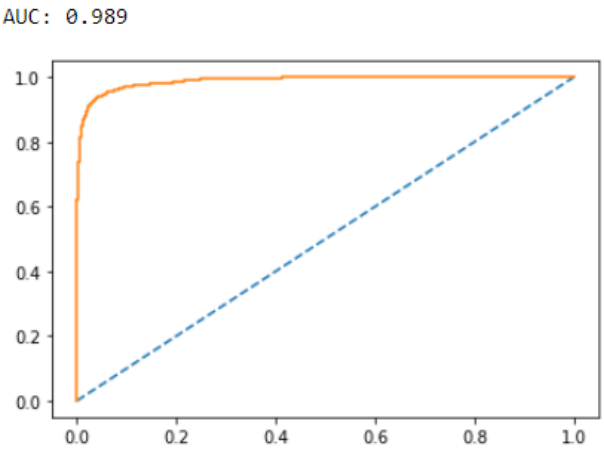


Figure 14: Performance Measure of XGBoost Tuned Model on Test Dataset

- We can see that the Accuracy score is high which states that the model has correctly predicted 96.7% of total cases including both class 0 and class 1 as well.
- The confusion matrix suggests that the model has:
  a) correctly identified 501 data points for class 1 (True Positives) of the total 569
  b) 68 class 1 data points have been incorrectly identified as belonging to class 0 (False Negatives)
  c) correctly identified 2766 data points for class 0 (True Negatives) out of 2809
  d) 43 class 0 data points have been incorrectly identified as belonging to class 1 (False Positives)
- The classification report suggests that the model has:
  a) Identified 88% of real True Positives of all data points (Recall for Class 1) and 98% of real True Negatives of all data points (Recall for Class 0)
  b) Identified real True Positives as True Positive correctly 92% of the times of all correctly identified data points (Precision for Class 1) and real True Negatives as True Negative 98% of the times of all correctly identified data points (Precision for Class 0)
- AUC (Area under Curve) is high at 99% which means that the model is very capable of distinguishing between class 0 and class 1.

Model Name	Accuracy		Recall		Precision		F1		AUC	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
XGBoost Tuned (unscaled data and without outliers)	**1.00**	**0.97**	**0.99**	**0.88**	**0.99**	**0.92**	**0.99**	**0.90**	**1.00**	**0.99**

Figure 15: XGBoost Tuned Model Train and Test Comparison

Cross Validation Score for Train Dataset:

```
array([0.95690748, 0.96451204, 0.95939086, 0.96573604, 0.95558376,
       0.96573604, 0.94796954, 0.95939086, 0.95177665, 0.96827411])
```

Cross Validation Score for Test Dataset:

```
array([0.94674556, 0.91715976, 0.91420118, 0.93195266, 0.91715976,
       0.92899408, 0.93491124, 0.9408284 , 0.91097923, 0.9495549 ])
```

- Recall is 99% on Train dataset and 88% on Test dataset.
- Accuracy is nearly similar with 100% for Train dataset and 97% on Test dataset.
- Precision is 99% on Train and 92% on Test dataset.
- The F-1 score has dropped from 99% on Train dataset to 90% on Test dataset.
- AUC is near equal with 100% on Train dataset and 99% on Test dataset.
- After 10 fold cross validation, scores both on train and test data set respectively for all 10 folds are almost same.

b) Any other model tuning measures (if applicable)

IV. SVM Tuned Model (scaled dataset without Outliers - x_scaled_train & x_test)

- The SVM Tuned built was tuned as per the below parameters using GridSearchCV:

```
param_grid = {
    'C': [0.1,1, 10, 100],
    'gamma': [1,0.1,0.01,0.001],
    'kernel': ['poly','rbf','linear','sigmoid']
}
```

- Best Parameters Identified:

```
{'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
```

Predictions on Train Dataset

```
Accuracy Score is 0.9982238010657194

Confusion Matrix
 [[6552    3]
 [  11 1316]]

Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6555
           1       1.00      0.99      0.99      1327

    accuracy                           1.00      7882
   macro avg       1.00      1.00      1.00      7882
weighted avg       1.00      1.00      1.00      7882
```

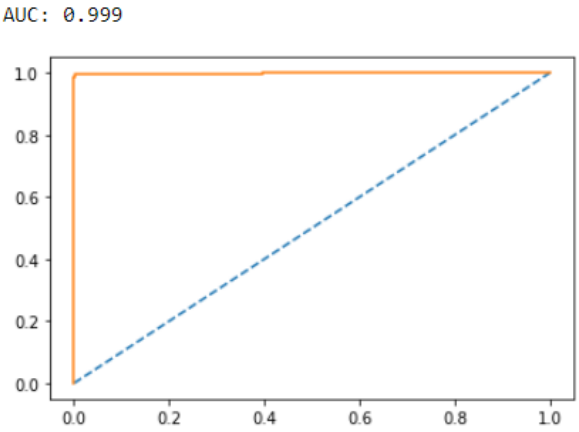


Figure 16: Performance Measure of SVM Tuned Model on Train Dataset

- We can see that the Accuracy score is high which states that the model has correctly predicted 99.8% of total cases including both class 0 and class 1 as well.
- The confusion matrix suggests that the model has:
  a) correctly identified 1316 data points for class 1 (True Positives) belonging to churners of the total 1327
  b) Only 11 class 1 data points have been incorrectly identified as belonging to class 0. (False Negatives)
  c) correctly identified 6552 data points for class 0 (True Negatives) out of 6555.
  d) Only 3 class 0 data points have been incorrectly identified as belonging to class 1 (False Positives)
- The classification report suggests that the model has:

a) Identified 99% of real True Positives of all data points (Recall for Class 1) and almost 100% of real True Negatives of all data points (Recall for Class 0)
b) Identified real True Positives as True Positive correctly almost 100% of the times of all correctly identified data points (Precision for Class 1) and real True Negatives as True Negative almost 100% of the times of all correctly identified data points (Precision for Class 0)

- AUC (Area under Curve) is high at almost 100% which means that the model is very highly capable of distinguishing between class 0 and class 1.

Predictions on Test Dataset

```
Accuracy Score is 0.9763173475429248

Confusion Matrix
 [[2782   27]
 [  53  516]]

Classification Report
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      2809
           1       0.95      0.91      0.93       569

    accuracy                           0.98      3378
   macro avg       0.97      0.95      0.96      3378
weighted avg       0.98      0.98      0.98      3378
```

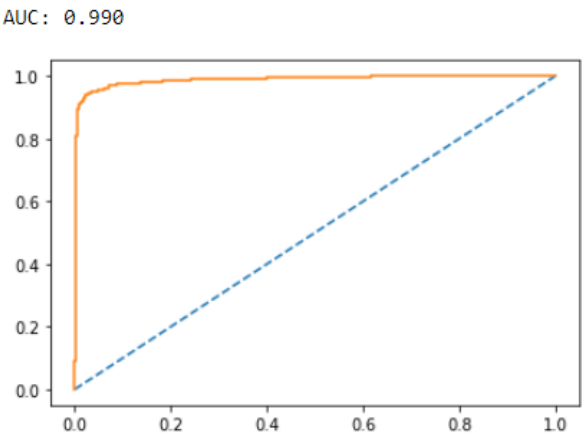


Figure 17: Performance Measure of SVM Tuned Model on Test Dataset

- We can see that the Accuracy score is high which states that the model has correctly predicted 97.6% of total cases including both class 0 and class 1 as well.
- The confusion matrix suggests that the model has:
  a) correctly identified 516 data points for class 1 (True Positives) of the total 569
  b) 53 class 1 data points have been incorrectly identified as belonging to class 0 (False Negatives)
  c) correctly identified 2782 data points for class 0 (True Negatives) out of 2809
  d) 27 class 0 data points have been incorrectly identified as belonging to class 1 (False Negatives)
- The classification report suggests that the model has:
  a) Identified 91% of real True Positives of all data points (Recall for Class 1) and 99% of real True Negatives of all data points (Recall for Class 0)
  b) Identified real True Positives as True Positive correctly 95% of the times of all correctly identified data points (Precision for Class 1) and real True Negatives as True Negative 98% of the times of all correctly identified data points (Precision for Class 0)
- AUC (Area under Curve) is high at 99% which means that the model is very capable of distinguishing between class 0 and class 1.

Model Name	Accuracy		Recall		Precision		F1		AUC	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
SVM Tuned (scaled dataset without Outliers)	**1.00**	**0.98**	**0.99**	**0.91**	**1.00**	**0.95**	**0.99**	**0.93**	**0.99**	**0.99**

Figure 18: SVM Tuned Model Train and Test Comparison

Cross Validation Score for Train Dataset:

```
array([0.96831432, 0.97338403, 0.96827411, 0.96319797, 0.98350254,
       0.9784264 , 0.96827411, 0.9822335 , 0.97461929, 0.98604061])
```

Cross Validation Score for Test Dataset:

```
array([0.94970414, 0.9260355 , 0.89940828, 0.94674556, 0.9112426 ,
       0.93786982, 0.94378698, 0.93195266, 0.91097923, 0.96142433])
```

- Recall is 99% on Train dataset and 91% on Test dataset.
- Accuracy is nearly similar with 100% for Train dataset and 98% on Test dataset.
- Precision is almost 100% on Train and 95% on Test dataset.
- The F-1 score is 99% on Train dataset and 93% on Test dataset.
- AUC is equal with 99% on Train dataset as well as Test dataset.
- After 10 fold cross validation, scores on train is same but vary for test data set.

Models built using SMOTE

```
x_train.shape
(7882, 17)

x_train_res.shape
(9832, 17)

xo_train.shape
(7882, 17)

xo_train_res.shape
(9832, 17)

y_train.shape
(7882,)

y_train_res.shape
(9832,)

yo_train.shape
(7882,)

yo_train_res.shape
(9832,)
```

Figure 19: Pre and Post SMOTE Train Dataset Shape

- sampling_strategy was kept at 50% as this produced the best result and which also indicates that not much synthesized data points were necessary to be added. A total of 1950 synthesized data points for class 1 has been added. Hence, now the dataset contains 6555 class 0 and 3277 class 1 data points.
- Smote train dataset now has 9832 data points against 7882 in the original train datasets. Data points for test remains same at 3378.
- It was observed that the performance of the models on test results drastically reduced at higher sampling_strategy's of 75% and 100%.

V. SVM with SMOTE (Scaled Dataset without Outliers - xs_train_res & xs_test)

Predictions on Train Dataset

```
Accuracy Score is 0.9493490642799024

Confusion Matrix
 [[6312  243]
 [ 255 3022]]

Classification Report
              precision    recall  f1-score   support

           0       0.96      0.96      0.96      6555
           1       0.93      0.92      0.92      3277

    accuracy                           0.95      9832
   macro avg       0.94      0.94      0.94      9832
weighted avg       0.95      0.95      0.95      9832
```

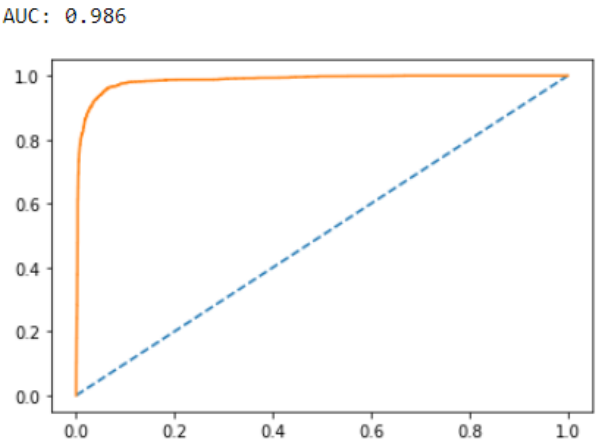


Figure 20: Performance Measure of SVM Model with SMOTE on Train Dataset

- We can see that the Accuracy score is high which states that the model has correctly predicted 95% of total cases including both class 0 and class 1 as well.
- The confusion matrix suggests that the model has:
  a) correctly identified 3022 data points for class 1 (True Positives) of the total 3277.
  b) 255 class 1 data points (False Negatives) have been incorrectly identified as belonging to class 0.
  c) correctly identified 6312 data points for class 0 (True Negatives) out of 6555.
  d) 243 class 0 data points (False Positives) have been incorrectly identified as belonging to class 1
- The classification report suggests that the model has:
  a) Identified 92% of real True Positives of all data points (Recall for Class 1) and almost 96% of real True Negatives of all data points (Recall for Class 0)
  b) Identified real True Positives as True Positive correctly almost 93% of the times of all correctly identified data points (Precision for Class 1) and real True Negatives as True Negative almost 96% of the times of all correctly identified data points (Precision for Class 0)
- AUC (Area under Curve) is high at 98.6% which means that the model is very capable of distinguishing between class 0 and class 1.

Predictions on Test Dataset

```
Accuracy Score is 0.92599171107164

Confusion Matrix
 [[2664  145]
 [ 105  464]]

Classification Report
              precision    recall  f1-score   support

           0       0.96      0.95      0.96      2809
           1       0.76      0.82      0.79       569

    accuracy                           0.93      3378
   macro avg       0.86      0.88      0.87      3378
weighted avg       0.93      0.93      0.93      3378
```

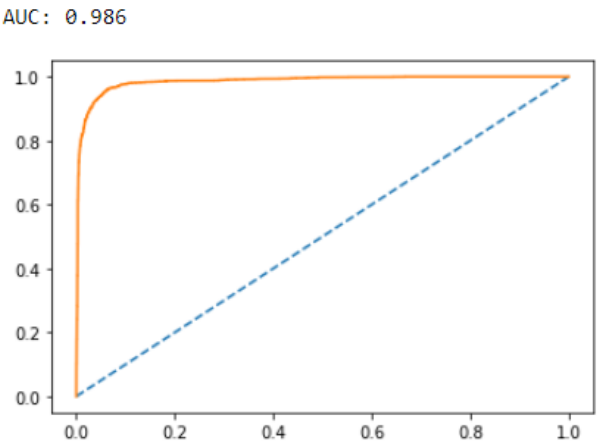


Figure 21: Performance Measure of SVM Model with SMOTE on Test Dataset

- We can see that the Accuracy score is good which states that the model has correctly predicted 92.5% of total cases including both class 0 and class 1 as well.
- The confusion matrix suggests that the model has:
  a) correctly identified 464 data points for class 1 (True Positives) of the total 569
  b) 105 class 1 data points have been incorrectly identified as belonging to class 0 (False Negatives)

c) correctly identified 2664 data points for class 0 (True Negatives) out of 2809
d) 145 class 0 data points have been incorrectly identified as belonging to class 1 (False Negatives)

- The classification report suggests that the model has:
  a) Identified 82% of real True Positives of all data points (Recall for Class 1) and 95% of real True Negatives of all data points (Recall for Class 0)
  b) Identified real True Positives as True Positive correctly 76% of the times of all correctly identified data points (Precision for Class 1) and real True Negatives as True Negative 96% of the times of all correctly identified data points (Precision for Class 0)
- AUC (Area under Curve) is high at 98.6% which means that the model is very capable of distinguishing between class 0 and class 1.

Model Name	Accuracy		Recall		Precision		F1		AUC	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
SVM with SMOTE (scaled dataset without Outliers)	0.95	0.93	0.92	0.82	0.93	0.76	0.92	0.79	0.99	0.99

Figure 22: SVM Model with SMOTE Train and Test Comparison

Cross Validation Score for Train Dataset:

```
array([0.91666667, 0.92682927, 0.92370295, 0.92777213, 0.92675483,
       0.94303154, 0.93692777, 0.94608342, 0.92980671, 0.95015259])
```

Cross Validation Score for Test Dataset:

```
array([0.90828402, 0.89940828, 0.88757396, 0.92011834, 0.90236686,
       0.90828402, 0.90236686, 0.89349112, 0.89614243, 0.91394659])
```

- Recall is 92% on Train dataset and 82% on Test dataset.
- Accuracy is nearly similar with 95% for Train dataset and 93% on Test dataset.
- Precision has dropped from 93% on Train to 76% on Test dataset.
- The F-1 score has dropped from 92% on Train dataset to 79% on Test dataset.
- AUC is equal with 99% on Train dataset and 99% on Test dataset.
- After 10 fold cross validation, scores both on train and test data set respectively for all 10 folds are almost same.

VI. Random Forest Model5 with SMOTE (unscaled data and with outliers - xo_train_res & xo_test)

- The Random Forest Model 5 built was tuned as per the below parameters using GridSearchCV:

```
GridSearchCV(estimator=RandomForestClassifier(random_state=1),
             param_grid={'max_depth': [30, 40, 50],
                         'min_samples_leaf': [4, 5, 7],
                         'min_samples_split': [4, 7, 10],
                         'n_estimators': [40, 45, 50]})
```

- Best Parameters Identified:

```
{'max_depth': 30,
 'min_samples_leaf': 4,
 'min_samples_split': 4,
 'n_estimators': 50}
```

Predictions on Train Dataset

```
Accuracy Score is 0.9811838893409276

Confusion Matrix
 [[6490   65]
 [ 120 3157]]

Classification Report
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      6555
           1       0.98      0.96      0.97      3277

    accuracy                           0.98      9832
   macro avg       0.98      0.98      0.98      9832
weighted avg       0.98      0.98      0.98      9832
```

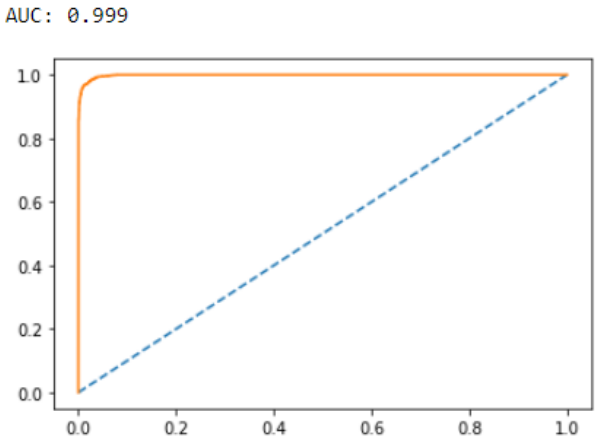


Figure 23: Performance Measure of Random Forest Model 5 with SMOTE on Train Dataset

- We can see that the Accuracy score is high which states that the model has correctly predicted 98% of total cases including both class 0 and class 1 as well.
- The confusion matrix suggests that the model has:
  a) correctly identified 3157 data points for class 1 (True Positives) of the total 3277.
  b) 120 class 1 data points (False Negatives) have been incorrectly identified as belonging to class 0.
  c) correctly identified 6490 data points for class 0 (True Negatives) out of 6555.
  d) 65 class 0 data points (False Positives) have been incorrectly identified as belonging to class 1
- The classification report suggests that the model has:
  a) Identified 96% of real True Positives of all data points (Recall for Class 1) and 99% of real True Negatives of all data points (Recall for Class 0)
  b) Identified real True Positives as True Positive correctly 98% of the times of all correctly identified data points (Precision for Class 1) and real True Negatives as True Negative also 98% of the times of all correctly identified data points (Precision for Class 0)
- AUC (Area under Curve) is high at 99.9% which means that the model is very highly capable of distinguishing between class 0 and class 1.

Predictions on Test Dataset

```
Accuracy Score is 0.9487862640615748

Confusion Matrix
 [[2719   90]
 [  83  486]]

Classification Report
              precision    recall  f1-score   support

           0       0.97      0.97      0.97      2809
           1       0.84      0.85      0.85       569

    accuracy                           0.95      3378
   macro avg       0.91      0.91      0.91      3378
weighted avg       0.95      0.95      0.95      3378
```

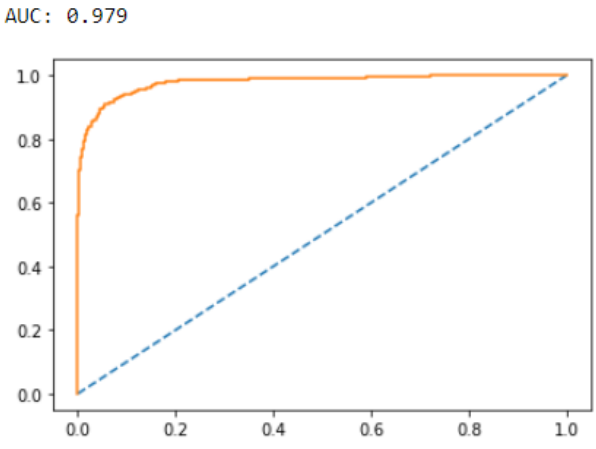


Figure 24: Performance Measure of Random Forest Model 5 with SMOTE on Test Dataset

- We can see that the Accuracy score is good which states that the model has correctly predicted 95% of total cases including both class 0 and class 1 as well.
- The confusion matrix suggests that the model has:
  a) correctly identified 486 data points for class 1 (True Positives) of the total 569

b) 83 class 1 data points have been incorrectly identified as belonging to class 0 (False Negatives)
c) correctly identified 2719 data points for class 0 (True Negatives) out of 2809
d) 90 class 0 data points have been incorrectly identified as belonging to class 1 (False Negatives)

- The classification report suggests that the model has:
  a) Identified 85% of real True Positives of all data points (Recall for Class 1) and 97% of real True Negatives of all data points (Recall for Class 0)
  b) Identified real True Positives as True Positive correctly 84% of the times of all correctly identified data points (Precision for Class 1) and real True Negatives as True Negative 97% of the times of all correctly identified data points (Precision for Class 0)
- AUC (Area under Curve) is high at 97.9% which means that the model is very capable of distinguishing between class 0 and class 1.

Model Name	Accuracy		Recall		Precision		F1		AUC	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Random Forest_model 5 - Hyper Tuned with SMOTE (unscaled data and with outliers)	**0.98**	**0.95**	**0.96**	**0.85**	**0.98**	**0.84**	**0.97**	**0.85**	**0.99**	**0.98**

Figure 25: Random Forest Model 5 with SMOTE Train and Test Comparison

Cross Validation Score for Train Dataset:

```
array([0.93191057, 0.94105691, 0.93997965, 0.94303154, 0.93489318,
       0.93489318, 0.92573754, 0.93997965, 0.91963377, 0.95523906])
```

Cross Validation Score for Test Dataset:

```
array([0.93195266, 0.91420118, 0.90532544, 0.91715976, 0.9112426 ,
       0.91715976, 0.92011834, 0.91715976, 0.90207715, 0.9347181 ])
```

- Recall is 96% on Train dataset and 85% on Test dataset.
- Accuracy is nearly similar with 98% for Train dataset and 95% on Test dataset.
- Precision has dropped from 98% on Train to 84% on Test dataset.
- The F-1 score has dropped from 97% on Train dataset to 85% on Test dataset.
- AUC is almost equal with 99% on Train dataset and 99% on Test dataset.
- After 10 fold cross validation, scores both on train and test data set respectively for all 10 folds are almost same.

VII. KNN with SMOTE (Scaled Dataset without Outliers - xs_train_res & xs_test)

Predictions on Train Dataset

```
Accuracy Score is 0.9792514239218877

Confusion Matrix
 [[6376  179]
 [  25 3252]]

Classification Report
              precision    recall  f1-score   support

           0       1.00      0.97      0.98      6555
           1       0.95      0.99      0.97      3277

    accuracy                           0.98      9832
   macro avg       0.97      0.98      0.98      9832
weighted avg       0.98      0.98      0.98      9832
```

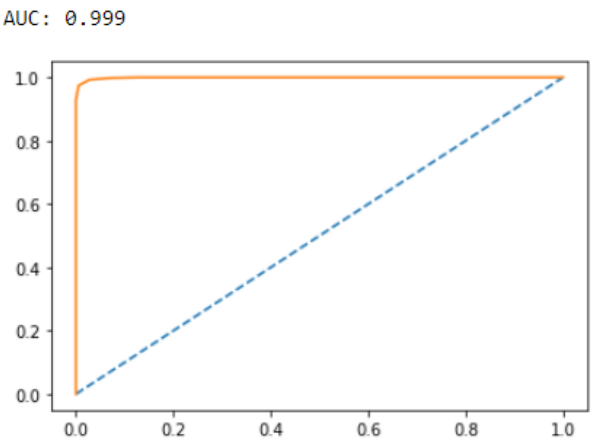


Figure 26: Performance Measure of KNN Model 5 with SMOTE on Train Dataset

- We can see that the Accuracy score is very high which states that the model has correctly predicted 98% of total cases including both class 0 and class 1 as well.
- The confusion matrix suggests that the model has:
  a) correctly identified 3252 data points for class 1 (True Positives) of the total 3277.
  b) 25 class 1 data points have been incorrectly identified as belonging to class 0. (False Negatives)
  c) correctly identified 6376 data points for class 0 (True Negatives) out of 6555.
  d) 179 class 0 data points (False Positives) have been incorrectly identified as belonging to class 1
- The classification report suggests that the model has:
  a) Identified 99% of real True Positives of all data points (Recall for Class 1) and 97% of real True Negatives of all data points (Recall for Class 0)
  b) Identified real True Positives as True Positive correctly 95% of the times of all correctly identified data points (Precision for Class 1) and real True Negatives as True Negative almost 100% of the times of all correctly identified data points (Precision for Class 0)
- AUC (Area under Curve) is high at 99.9% which means that the model is very highly capable of distinguishing between class 0 and class 1.

Predictions on Test Dataset

```
Accuracy Score is 0.9393132030787448

Confusion Matrix
 [[2644  165]
 [  40  529]]

Classification Report
              precision    recall  f1-score   support

           0       0.99      0.94      0.96      2809
           1       0.76      0.93      0.84       569

    accuracy                           0.94      3378
   macro avg       0.87      0.94      0.90      3378
weighted avg       0.95      0.94      0.94      3378
```

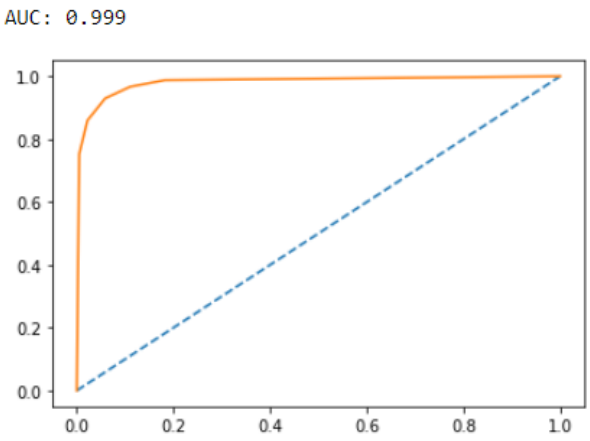


Figure 27: Performance Measure of KNN Model with SMOTE on Test Dataset

- We can see that the Accuracy score is good which states that the model has correctly predicted 94% of total cases including both class 0 and class 1 as well.
- The confusion matrix suggests that the model has:
  a) correctly identified 529 data points for class 1 (True Positives) of the total 569
  b) 40 class 1 data points have been incorrectly identified as belonging to class 0 (False Negatives)
  c) correctly identified 2644 data points for class 0 (True Negatives) out of 2809
  d) 165 class 0 data points have been incorrectly identified as belonging to class 1 (False Negatives)
- The classification report suggests that the model has:
  a) Identified 93% of real True Positives of all data points (Recall for Class 1) and 94% of real True Negatives of all data points (Recall for Class 0)
  b) Identified real True Positives as True Positive correctly 76% of the times of all correctly identified data points (Precision for Class 1) and real True Negatives as True Negative 99% of the times of all correctly identified data points (Precision for Class 0)
- AUC (Area under Curve) is high at 99.9% which means that the model is very highly capable of distinguishing between class 0 and class 1.

Model Name	Accuracy		Recall		Precision		F1		AUC	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test

KNN_model with SMOTE (scaled data without outliers)	**0.98**	**0.94**	**0.99**	**0.93**	**0.95**	**0.76**	**0.97**	**0.84**	**0.99**	**0.99**

Figure 28: KNN Model with SMOTE Train and Test Comparison

Cross Validation Score for Train Dataset:

```
array([0.94308943, 0.95020325, 0.95116989, 0.95218718, 0.95523906,
       0.9664293 , 0.95727365, 0.95422177, 0.95320448, 0.95523906])
```

Cross Validation Score for Test Dataset:

```
array([0.8816568 , 0.86390533, 0.88461538, 0.9112426 , 0.86094675,
       0.89940828, 0.89940828, 0.86390533, 0.87240356, 0.92284866])
```

- Recall is 99% on Train dataset and 93% on Test dataset.
- Accuracy is 98% for Train dataset and 94% on Test dataset.
- Precision has dropped from 95% on Train to 76% on Test dataset.
- The F-1 score has dropped from 97% on Train dataset to 84% on Test dataset.
- AUC is equal with 99% on Train dataset and 99% on Test dataset.
- After 10 fold cross validation, scores both on train and test data set respectively for all 10 folds are almost same.

c) Interpretation of the most optimum model and its implication on the business

Most optimum model:

Model Name	Accuracy		Recall		Precision		F1		AUC	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
CART Tuned (Unscaled Dataset without Outliers)	**0.97**	**0.94**	**0.89**	**0.78**	**0.95**	**0.83**	**0.92**	**0.81**	**0.99**	**0.99**
KNN_model (scaled data without outliers)	**0.98**	**0.95**	**0.90**	**0.80**	**0.97**	**0.89**	**0.93**	**0.84**	**0.99**	**0.99**
SVM with SMOTE (scaled dataset without Outliers)	**0.95**	**0.93**	**0.92**	**0.82**	**0.93**	**0.76**	**0.92**	**0.79**	**0.99**	**0.99**
Random Forest_model 5 - Hyper Tuned with SMOTE (unscaled data and with outliers)	**0.98**	**0.95**	**0.96**	**0.85**	**0.98**	**0.84**	**0.97**	**0.85**	**0.99**	**0.98**
XGBoost Tuned (unscaled data and without outliers)	**1.00**	**0.97**	**0.99**	**0.88**	**0.99**	**0.92**	**0.99**	**0.90**	**1.00**	**0.99**
SVM Tuned (scaled dataset without Outliers)	**1.00**	**0.98**	**0.99**	**0.91**	**1.00**	**0.95**	**0.99**	**0.93**	**0.99**	**0.99**
KNN_model with SMOTE (scaled data without outliers)	**0.98**	**0.94**	**0.99**	**0.93**	**0.95**	**0.76**	**0.97**	**0.84**	**0.99**	**0.99**

Figure 29: Models selected for evaluation

- CART Tuned model and KNN_model did seem to be reliable but had lower recall values for class 1 and we aim to build a model that would maximize the chances of predicting customers for class 1.
- Random Forest_model 5 - Hyper Tuned with SMOTE had a good recall of 85% for class 1 on test dataset but it is not the best model as it too seemed unstable with lower precision and F-1 scores on test dataset compared to the train dataset.
- SVM with SMOTE also had lower precision on test dataset of only 76% and also had a lower F-1 score of 79% compared to the train dataset making the model less reliable.
- KNN_model with SMOTE had the best recall of 93% on test dataset for class 1 (correctly identified 529 true positives of 569 observations). However, the precision on test dataset was only 76% and also F-1 score of 84% was drastically lower compared to the train dataset making the model less reliable and unstable.
- XGBoost Tuned model performed extremely well on both the train and test dataset (incorrectly identified only 68 observations for class 1 and 43 observations for class 0 on test dataset). It has also performed well in predicting class 0. It is also a reliable and stable model with good precision, F-1, accuracy and AUC scores.

- SVM Tuned has better scores than all the other models on train and test datasets across all performance metrics only except for recall for test dataset and AUC on train dataset. However, it was seen during cross validation that scores were varying on the test dataset making the model inconsistent. Also, It is not possible to extract feature importance for SVM models.
- Hence, from all the above observations we can determine that XGBoost Tuned model is the most optimal model.

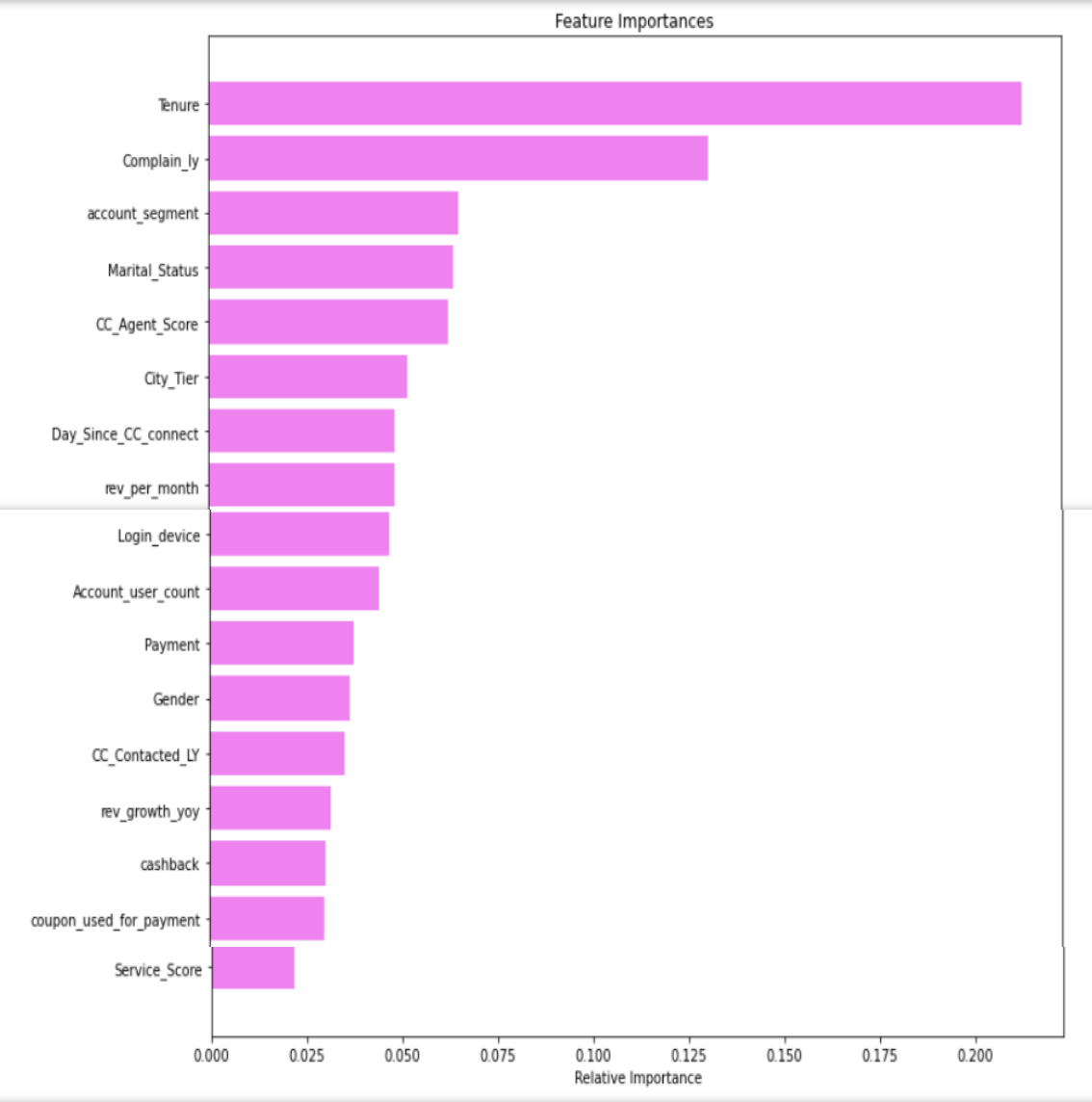


Figure 30: Important Features for XGBoost Tuned Model

Below are the most Important Features for XGBoost Tuned Model (descending order)

- Tenure
- Complain_ly
- account_segment
- Marital_Status
- CC_Agent_Score
- City Tier
- Day_Since_CC_connect
- rev_per_month
- Login_device
- Account_user_count
- Payment

- Gender
- CC_Contacted_LY

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-3.4338	0.320	-10.740	0.000	-4.060	-2.807
Tenure	-0.1658	0.007	-22.907	0.000	-0.180	-0.152
City_Tier	0.3321	0.040	8.223	0.000	0.253	0.411
CC_Contacted_LY	0.0299	0.004	7.036	0.000	0.022	0.038
Payment	-0.0565	0.036	-1.555	0.120	-0.128	0.015
Gender	0.3120	0.076	4.123	0.000	0.164	0.460
Service_Score	-0.0707	0.055	-1.288	0.198	-0.178	0.037
Account_user_count	0.3615	0.040	8.984	0.000	0.283	0.440
account_segment	-0.3470	0.037	-9.337	0.000	-0.420	-0.274
CC_Agent_Score	0.2763	0.027	10.156	0.000	0.223	0.330
Marital_Status	0.6103	0.057	10.704	0.000	0.499	0.722
rev_per_month	0.0908	0.010	8.821	0.000	0.071	0.111
Complain_ly	1.5941	0.076	20.968	0.000	1.445	1.743
rev_growth_yoy	-0.0264	0.010	-2.643	0.008	-0.046	-0.007
coupon_used_for_payment	0.1288	0.021	6.014	0.000	0.087	0.171
Day_Since_CC_connect	-0.1153	0.013	-8.777	0.000	-0.141	-0.090
cashback	-0.0026	0.001	-3.012	0.003	-0.004	-0.001
Login_device	-0.4197	0.079	-5.290	0.000	-0.575	-0.264

Figure 31: Coefficients of all 17 Predictor Variables

We see that P values are > 0.05 for Payment and Service_Score which clearly indicates that these are not significant variables to determine probability of Churn. Hence, we will exclude inferencing on these variables below.

Implication on the business:

- **Tenure:**
  - This variable was observed as important across all models built. Hence, it is extremely necessary to convert customers into long term loyal customers.
  - Coefficient for Tenure is negative, which means increase in tenure decreases the possibility of churn.
  - The business can bring in loyalty programs for rewarding customers with long term association with the business.
  - Introduce referral benefits for adding new users to the account or even offer redeem points to primary account holder on purchases made by the other users in the account.
  - Ensure maximum customers are regularly seeing your products, offers, etc through emails, push notifications, social media posts, youtube advertisements etc
  - Ensure that the business does not get involved in any controversy which hampers its reputation. Customers like to be associated with clean brands.
- **Complain_ly:**
  - This variable was observed as important across 85% of the all models we built and in 100% of all the models we selected to evaluate. Hence, it is extremely necessary to avoid any complaints from customers or/and if any complaints are received then to resolve it in a swift and smooth manner. This highlights the efficiency of a business to provide excellent customer service.
  - Coefficient for Complain_ly is positive, which means increase in complaints increases the possibility of churn.
  - E-commerce website, mobile application, etc needs to be in an extremely good functional condition which can offer a seamless purchasing and transacting experience for the users.
  - User interface to be effortless for even customers without technological know-how.
  - All partners/vendors/supportive businesses such as cargo and shipping, packaging, freight and transport, IT, services and communication etc should be in sync to avoid any loopholes. Hence, need to choose them diligently and incentivise them for excellent service and also can provide them reward and recognition in exchange.

- Arrange for feedback and surveys to understand customer sentiments and try and find areas which can create opportunity for complaints to tackle them beforehand.
- Try and dedicate a small team to exclusively handle customer, vendor and inter-company complaints.

- **account_segment:**
  - This variable was observed as important across 80% of the all models we built and in 100% in all the models we selected to evaluate.
  - Coefficient for account_segment is negative, which means increase in account_segment decreases the possibility of churn. Increase in spending prevents chances of churn.
  - Maximum accounts are under Regular Plus which can be said to be the average category. Second maximum customers belong to Super category which is 2nd best category. Even maximum customers who have churned belong to these two categories.
  - Specific targeted offers/combos can be designed for customer belonging the Regular Plus which can move them to Super and similarly, strategies can be formed for Super category customers to move them to Super Plus.
  - Most purchased products/categories for Regular Plus customers can be evaluated to create such combos/offers. Even newer, in demand products in these categories can be introduced.
  - Similar products/supplementary products based on interests, past purchases can be run under product recommendations and shared via email, notifications etc.
  - Also, pick and choose premium products from those categories and also suggest to these customers to increase possibility of more spends and offer them at discounts whenever possible for eg; festival, New Year shopping.
- **Marital_Status:**
  - This variable was observed as important across 65% of the all models we built and in 75% in all the models we selected to evaluate.
  - Most of the Primary customers are Married. However, maximum customer who are churning are Single. Hence, focus should be on Single customers as well.
  - Shopping interest and past product purchase history based recommendations need to be set for both Single and Married Customers.
  - Inventory should include more trending and in fashion/style products.
  - For Married, anniversary greetings, offers, couple combos can be curated. Products for couples can be recommended along with Valentine's day offers.
  - Special offers for birthdays can be released.
- **CC_Agent_Score:**
  - This variable was observed as important across 90% of the all models we built and in 75% in all the models we selected to evaluate.
  - Coefficient for CC_Agent_Score is positive which is odd but it means customers who are giving a higher rating are churning.
  - Business needs to analyse how this data of Satisfaction score to customer care representative is collected and if it is collected in a legitimate way.
  - Special training sessions for customer service representatives can be arranged once in a while as well.
  - Track customer service performances and reward employees doing well.
- **City Tier:**
  - This variable was observed as important across 60% of the all models we built and in 75% in all the models we selected to evaluate.
  - Business is losing more customers from tier 3 compared to customers they are able to retain.
  - Also the count of accounts in tier 2 is very less.
  - Frequency of reaching customers to these cities need to be improved by increasing partners/vendors in these areas and also distributors as well.
- **Day_Since_CC_connect:**
  - This is variable was observed as important across all models built.
  - Coefficient for this feature is negative, which means increase in gap between customer and customer service agent decreases the possibility of churn.

    - Higher the gap indicates that an active customer is not having major issues with the services and is able to make purchases seamlessly.
    - Hence, better customer service, products, ease of purchase, better the possibility of retention.
    - Make tutorials available to user the website, mobile application. Provide quick solutions through auto bots, show steps and suggestions to improve the purchase experience.
- **Login_device:**
    - This variable was observed as important across 60% of the all models we built.
    - However, we would emphasis on only that the more customers login the better. Hence, ensure that the website and mobile application is maintained, up and running smoothly and invest on the software to make them stronger, robust and richer.
    - Even Sign Up and subscriptions should be easy to execute.
    - Payment System Security to be also be secured completely.
- **Account_user_count:**
    - This variable was observed as important across 70% of the all models we built.
    - Too many users in an account can also be a problem. Hence, it is advisable to limit users per account
    - Also, as mentioned in the problem itself, in case of one account churn, multiple users are lost.
    - Hence, provide opportunity for users to create at least two accounts.
    - If possible avoid having multiple users tagged to one account. Average users per account are 4 for those who have churned.
- **Gender:**
    - This variable was observed as important across only 30% of the all models we built.
    - Major chuck of primary users are male (60%).
    - Business to increase the variety of products offered for both sections as both sections are churning at almost equal proportions.
    - Also, there is a need to add more female primary users or users as there are a wide variety of products they purchase and the frequency of purchases is generally higher.
- **CC_Contacted_LY:**
    - This variable was observed as important across 80% of the all models we built.
    - Coefficient for this feature is positive, which means more contact with customer care, higher chances of churn. Hence, can follow steps provided under CC_Agent_Score and Day_Since_CC_connect for the same.
    - Share customer connect experiences and feedbacks received with other departments to create synergies through collaborations which can ensure in building a great ecommerce platform for the users.
- Rev_growth_yoy, Cashback and Coupon_used_for_payment were not observed as important features for this and rest of the models built as well.

Appendix

Raw Codes & Outputs

PROJECT NOTE 2

```
data.Churn.value_counts()
```

```
0    9364
1    1896
Name: Churn, dtype: int64
```

```
data.Churn.value_counts(normalize = True)
```

Train Test Split (xo & yo)

```
from sklearn.model_selection import train_test_split
```

```
xo_train, xo_test, yo_train, yo_test = train_test_split(xo,yo, test_size= 0.30, random_state=1, stratify= data['Churn'])
```

```
# Checking dimensions on the train and test data
print('xo_train: ',xo_train.shape)
print('xo_test: ',xo_test.shape)
print('yo_train: ',yo_train.shape)
print('yo_test: ',yo_test.shape)
```

```
df_train = pd.concat([xo_train,yo_train], axis=1)
df_test = pd.concat([xo_test,yo_test], axis=1)
```

Train Test Split (x & y)

```
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size= 0.30, random_state=1, stratify= data['Churn'])
```

```
df1_train = pd.concat([x_train,y_train], axis=1)
df1_test = pd.concat([x_test,y_test], axis=1)
```

```
# Checking dimensions on the train and test data
print('x_train: ',x_train.shape)
print('x_test: ',x_test.shape)
print('y_train: ',y_train.shape)
print('y_test: ',y_test.shape)
```

Scaling the variables

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
x_scaled_train = sc.fit_transform(x_train)
```

```
x_scaled_train
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
pd.set_option('display.max_rows', None)
pd.set_option('display.max_colwidth', None)
pd.set_option("display.max_columns", None)
```

```
data = pd.read_excel('E:\GL\Course Content\Capstone\Capstone Business Project\CC_EDTH_02_Customer Churn\Customer Churn Data.xlsx'
                     sheet_name='Data for DSBA')
data.head()
```

```
x_scaled_test = sc.transform(x_test)
```

```
x_scaled_test
```

SMOTE

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=1, sampling_strategy = .50)
x_train_res, y_train_res = sm.fit_resample(x_train, y_train)
xo_train_res, yo_train_res = sm.fit_resample(xo_train, yo_train)
```

Model Building

Logistic Regression (Scaled Dataset without Outliers - x_scaled_train & x_scaled_test)

```
from sklearn.linear_model import LogisticRegression
import statsmodels.formula.api as SM
from sklearn import metrics
from sklearn.metrics import roc_auc_score,roc_curve,classification_report,confusion_matrix,plot_confusion_matrix
```

```
lr = LogisticRegression()
lr.fit(x_scaled_train, y_train)
```

Prediction on Train set ¶

```
y_train_predict = lr.predict(x_scaled_train)
model_score = lr.score(x_scaled_train, y_train)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_train, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_train, y_train_predict))
```

```
# predict probabilities
probs = lr.predict_proba(x_scaled_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = lr.predict(x_scaled_test)
model_score = lr.score(x_scaled_test, y_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n',metrics.confusion_matrix(y_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_test, y_test_predict))
```

Logistic Regression (Unscaled Dataset without Outliers - x_train & x_test)

```
lr2 = LogisticRegression()
lr2.fit(x_train, y_train)
```

Prediction on Train set

```
y_train_predict = lr2.predict(x_train)
model_score = lr2.score(x_train, y_train)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_train, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_train, y_train_predict))
```

```
# predict probabilities
probs = lr2.predict_proba(x_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = lr2.predict(x_test)
model_score = lr2.score(x_test, y_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n',metrics.confusion_matrix(y_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_test, y_test_predict))
```

```
# predict probabilities
probs = lr2.predict_proba(x_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

Logistic Regression SM models (Unscaled Dataset without Outliers - x_train & x_test)

```
x.columns
```

```
f_1 = 'Churn ~ Tenure + City_Tier + CC_Contacted_LY + Payment + Gender + Service_Score + Account_user_count + account_segment + C
```

```
model_1 = SM.logit(formula = f_1, data=df1_train).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.311567
         Iterations 8
```

```
model_1.summary()
```

```
print('The adjusted pseudo R-square value is',1 - ((model_1.llf - model_1.df_model)/model_1.llnull))
```

```
f_2 = 'Churn ~ Tenure + City_Tier + CC_Contacted_LY + Payment + Gender + Account_user_count + account_segment + CC_Agent_Score +
```

```
model_2 = SM.logit(formula = f_2, data=df1_train).fit()
model_2.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.311673
         Iterations 8
```

```
f_3 = 'Churn ~ Tenure + City_Tier + CC_Contacted_LY + Gender + Account_user_count + account_segment + CC_Agent_Score + Marital_St
```

```
model_3 = SM.logit(formula = f_3, data=df1_train).fit()
model_3.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.311825
         Iterations 8
```

Checking the Variance Inflation Factor

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

    return(vif)
```

```
calc_vif(x[['Tenure', 'City_Tier', 'CC_Contacted_LY', 'Gender', 'Account_user_count', 'account_segment', 'CC_Agent_Score',
            'Marital_Status', 'rev_per_month', 'Complain_ly', 'rev_growth_yoy', 'coupon_used_for_payment',
            'Day_Since_CC_connect', 'cashback', 'Login_device']]).sort_values(by='VIF', ascending = False)
```

Prediction on the Train Set

Now, let us see the predicted probability values on unscaled data:

```
y_train_predict = model_3.predict(x_train)
y_train_predict
```

```
sns.boxplot(x=data['Churn'],y=y_train_predict)
plt.xlabel('Churn');
```

```
y_class_pred=[]
for i in range(0,len(y_train_predict)):
    if np.array(y_train_predict)[i]>0.05:
        a=1
    else:
        a=0
    y_class_pred.append(a)
```

```
from sklearn.metrics import accuracy_score
```

```
score =accuracy_score(y_train,y_class_pred)
print('Accuracy Score is', score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_train, y_class_pred),'\n')
print('Classification Report','\n', metrics.classification_report(y_train,y_class_pred))
```

```
# calculate AUC
auc = roc_auc_score(y_train,y_class_pred)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train,y_class_pred)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = model_3.predict(x_test)
y_test_predict
```

```
y_class_pred=[]
for i in range(0,len(y_test_predict)):
    if np.array(y_test_predict)[i]>0.05:
        a=1
    else:
        a=0
    y_class_pred.append(a)
```

```
score =accuracy_score(y_test,y_class_pred)
print('Accuracy Score is', score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_test,y_class_pred),'\n')
print('Classification Report','\n', metrics.classification_report(y_test,y_class_pred))
```

```
# calculate AUC
test_auc = roc_auc_score(y_test,y_class_pred)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test,y_class_pred)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

Choosing the optimal threshold

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_train,y_train_predict)
```

```
optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds[optimal_idx]
optimal_threshold
```

Validating on the train set with revised threshold

```
y_class_pred=[]
for i in range(0,len(y_train_predict)):
    if np.array(y_train_predict)[i]>0.236:
        a=1
    else:
        a=0
    y_class_pred.append(a)
```

```
score =accuracy_score(y_train,y_class_pred)
print('Accuracy Score is', score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_train, y_class_pred),'\n')
print('Classification Report','\n', metrics.classification_report(y_train,y_class_pred))
```

```
# calculate AUC
auc = roc_auc_score(y_train,y_class_pred)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train,y_class_pred)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Validating on the test set

```
y_class_pred=[]
for i in range(0,len(y_test_predict)):
    if np.array(y_test_predict)[i]>0.236:
        a=1
    else:
        a=0
    y_class_pred.append(a)
```

```
score =accuracy_score(y_test,y_class_pred)
print('Accuracy Score is', score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_test,y_class_pred),'\n')
print('Classification Report','\n', metrics.classification_report(y_test,y_class_pred))
```

```
# calculate AUC
test_auc = roc_auc_score(y_test,y_class_pred)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test,y_class_pred)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

Logistic Regression Model_3 with 0.19 threshold

Validating on the train set with revised threshold - 0.19

```
y_class_pred=[]
for i in range(0,len(y_train_predict)):
    if np.array(y_train_predict)[i]>0.19:
        a=1
    else:
        a=0
    y_class_pred.append(a)
```

```
score =accuracy_score(y_train,y_class_pred)
print('Accuracy Score is', score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_train, y_class_pred),'\n')
print('Classification Report','\n', metrics.classification_report(y_train,y_class_pred))
```

```
# calculate AUC
auc = roc_auc_score(y_train,y_class_pred)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train,y_class_pred)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Validating on the test set

```
y_class_pred=[]
for i in range(0,len(y_test_predict)):
    if np.array(y_test_predict)[i]>0.19:
        a=1
    else:
        a=0
    y_class_pred.append(a)
```

```
score =accuracy_score(y_test,y_class_pred)
print('Accuracy Score is', score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_test,y_class_pred),'\n')
print('Classification Report','\n', metrics.classification_report(y_test,y_class_pred))
```

```
# calculate AUC
test_auc = roc_auc_score(y_test,y_class_pred)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test,y_class_pred)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

```
calc_vif(x).sort_values(by = 'VIF', ascending = False)
```

```
a = x.drop('Service_Score', axis = 1)
calc_vif(a).sort_values(by = 'VIF', ascending = False)
```

```
a = a.drop('Payment', axis = 1)
calc_vif(a).sort_values(by = 'VIF', ascending = False)
```

```
a = a.drop('rev_growth_yoy', axis = 1)
calc_vif(a).sort_values(by = 'VIF', ascending = False)
```

```
a = a.drop('Account_user_count', axis = 1)
calc_vif(a).sort_values(by = 'VIF', ascending = False)
```

```
a = a.drop('cashback', axis = 1)
calc_vif(a).sort_values(by = 'VIF', ascending = False)
```

```
a = a.drop('CC_Agent_Score', axis = 1)
calc_vif(a).sort_values(by = 'VIF', ascending = False)
```

```
a = a.drop('CC_Contacted_LY', axis = 1)
calc_vif(a).sort_values(by = 'VIF', ascending = False)
```

```
a.columns
```

```
Index(['Tenure', 'City_Tier', 'Gender', 'account_segment', 'Marital_Status',
       'rev_per_month', 'Complain_ly', 'coupon_used_for_payment',
       'Day_Since_CC_connect', 'Login_device'],
      dtype='object')
```

```
a.shape
```

```
(11260, 10)
```

```
x.shape
```

```
(11260, 17)
```

```
f_4 = 'Churn ~ Tenure + City_Tier + Gender + account_segment + Marital_Status + rev_per_month + Complain_ly + Day_Since_CC_connec
```

```
model_4 = SM.logit(formula = f_4, data=df1_train).fit()
model_4.summary()
```

```
print('The adjusted pseudo R-square value is',1 - ((model_4.llf - model_4.df_model)/model_4.llnull))
```

Prediction on Logistic Regression Model_4 Train Set with 0.19 threshold

Now, let us see the predicted probability values on unscaled data:

```
y_train_predict = model_4.predict(x_train)
y_train_predict
```

```
y_class_pred=[]
for i in range(0,len(y_train_predict)):
    if np.array(y_train_predict)[i]>0.19:
        a=1
    else:
        a=0
    y_class_pred.append(a)
```

```
from sklearn.metrics import accuracy_score

score =accuracy_score(y_train,y_class_pred)
```

```
print('Accuracy Score is', score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_train, y_class_pred),'\n')
print('Classification Report','\n', metrics.classification_report(y_train,y_class_pred))
```

```
# calculate AUC
auc = roc_auc_score(y_train,y_class_pred)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train,y_class_pred)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = model_4.predict(x_test)
y_test_predict
```

```
y_class_pred=[]
for i in range(0,len(y_test_predict)):
    if np.array(y_test_predict)[i]>0.19:
        a=1
    else:
        a=0
    y_class_pred.append(a)
```

```
from sklearn.metrics import accuracy_score

score =accuracy_score(y_test,y_class_pred)
```

```
print('Accuracy Score is', score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_test,y_class_pred),'\n')
print('Classification Report','\n', metrics.classification_report(y_test,y_class_pred))
```

```
# calculate AUC
test_auc = roc_auc_score(y_test,y_class_pred)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test,y_class_pred)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

Choosing the optimal threshold

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_train,y_train_predict)
```

```
optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds[optimal_idx]
optimal_threshold
```

```
0.21996947594614671
```

Logistic Regression Model_4 with 0.219 threshold

Validating on the train set with revised threshold

```
y_class_pred=[]
for i in range(0,len(y_train_predict)):
    if np.array(y_train_predict)[i]>0.219:
        a=1
    else:
        a=0
    y_class_pred.append(a)
```

```
score =accuracy_score(y_train,y_class_pred)
print('Accuracy Score is', score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_train, y_class_pred),'\n')
print('Classification Report','\n', metrics.classification_report(y_train,y_class_pred))
```

```
# calculate AUC
auc = roc_auc_score(y_train,y_class_pred)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train,y_class_pred)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Validating on the test set

```
y_class_pred=[]
for i in range(0,len(y_test_predict)):
    if np.array(y_test_predict)[i]>0.219:
        a=1
    else:
        a=0
    y_class_pred.append(a)
```

```
score =accuracy_score(y_test,y_class_pred)
print('Accuracy Score is', score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_test,y_class_pred),'\n')
print('Classification Report','\n', metrics.classification_report(y_test,y_class_pred))
```

```
# calculate AUC
test_auc = roc_auc_score(y_test,y_class_pred)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test,y_class_pred)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

Random Forest Model (unscaled data and with outliers - xo_train & xo_test)

```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=501, random_state=1)
rf_model.fit(xo_train, yo_train)
```

Prediction on Train set

```
y_train_predict = rf_model.predict(xo_train)
model_score = rf_model.score(xo_train, yo_train)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(yo_train, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(yo_train, y_train_predict))
```

```
# predict probabilities
probs = rf_model.predict_proba(xo_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(yo_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(yo_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set ¶

```
y_test_predict = rf_model.predict(xo_test)
model_score = rf_model.score(xo_test, yo_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(yo_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(yo_test, y_test_predict))
```

```
# predict probabilities
probs = rf_model.predict_proba(xo_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(yo_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(yo_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

```
importances = rf_model.feature_importances_
indices = np.argsort(importances)
feature_names = list(xo.columns)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```

```
from sklearn.model_selection import GridSearchCV
```

Random Forest Model3 (unscaled data and with outliers - xo_train & xo_test)

```
param_grid = {
    'max_depth': [50,60],
    'min_samples_leaf': [4,5,7],
    'min_samples_split': [10,15,20],
    'n_estimators': [20,30,40]
}

rf_model3 = RandomForestClassifier(random_state=1)

grid_search = GridSearchCV(estimator = rf_model3, param_grid = param_grid)
```

```
grid_search.fit(xo_train, yo_train)
```

```
GridSearchCV(estimator=RandomForestClassifier(random_state=1),
             param_grid={'max_depth': [50, 60], 'min_samples_leaf': [4, 5, 7],
                         'min_samples_split': [10, 15, 20],
                         'n_estimators': [20, 30, 40]})
```

```
grid_search.best_params_
```

```
{'max_depth': 50,
 'min_samples_leaf': 4,
 'min_samples_split': 10,
 'n_estimators': 40}
```

```
best_grid = grid_search.best_estimator_
```

Prediction on Train set

```
y_train_predict = best_grid.predict(xo_train)
model_score = best_grid.score(xo_train, yo_train)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(yo_train, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(yo_train, y_train_predict))
```

```
# predict probabilities
probs = best_grid.predict_proba(xo_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(yo_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(yo_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = best_grid.predict(xo_test)
model_score = best_grid.score(xo_test, yo_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(yo_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(yo_test, y_test_predict))
```

```
importances = best_grid.feature_importances_
indices = np.argsort(importances)
feature_names = list(xo.columns)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```

Random Forest Model4 (unscaled data and with outliers - xo_train & xo_test)

```
param_grid = {
    'max_depth': [30,45,50],
    'min_samples_leaf': [5,10,15],
    'min_samples_split': [10,15,20],
    'n_estimators': [30,40,50]
}

rf_model4 = RandomForestClassifier(random_state=1)

grid_search = GridSearchCV(estimator = rf_model4, param_grid = param_grid)
```

```
grid_search.fit(xo_train, yo_train)
```

```
grid_search.best_params_
```

```
{'max_depth': 30,
 'min_samples_leaf': 5,
 'min_samples_split': 10,
 'n_estimators': 40}
```

```
best_grid = grid_search.best_estimator_
```

Prediction on Train set

```
y_train_predict = best_grid.predict(xo_train)
model_score = best_grid.score(xo_train, yo_train)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(yo_train, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(yo_train, y_train_predict))
```

```
# predict probabilities
probs = best_grid.predict_proba(xo_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(yo_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(yo_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = best_grid.predict(xo_test)
model_score = best_grid.score(xo_test, yo_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(yo_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(yo_test, y_test_predict))
```

```
# predict probabilities
probs = best_grid.predict_proba(xo_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(yo_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(yo_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

```
importances = best_grid.feature_importances_
indices = np.argsort(importances)
feature_names = list(xo.columns)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```

Random Forest Model5 (unscaled data and with outliers - xo_train & xo_test)

```
param_grid = {
    'max_depth': [30,40,50],
    'min_samples_leaf': [4,5,7],
    'min_samples_split': [4,7,10],
    'n_estimators': [40,45,50]
}

rf_model5 = RandomForestClassifier(random_state=1)

grid_search = GridSearchCV(estimator = rf_model5, param_grid = param_grid)
```

```
grid_search.fit(xo_train, yo_train)
```

```
grid_search.best_params_
```

```
{'max_depth': 30,
 'min_samples_leaf': 4,
 'min_samples_split': 4,
 'n_estimators': 50}
```

```
best_grid = grid_search.best_estimator_
```

Prediction on Train set

```
y_train_predict = best_grid.predict(xo_train)
model_score = best_grid.score(xo_train, yo_train)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(yo_train, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(yo_train, y_train_predict))
```

```
# predict probabilities
probs = best_grid.predict_proba(xo_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(yo_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(yo_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = best_grid.predict(xo_test)
model_score = best_grid.score(xo_test, yo_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(yo_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(yo_test, y_test_predict))
```

```
# predict probabilities
probs = best_grid.predict_proba(xo_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(yo_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(yo_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

```
importances = best_grid.feature_importances_
indices = np.argsort(importances)
feature_names = list(xo.columns)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(best_grid, xo_train, yo_train, cv=10)
scores
```

```
array([0.92395437, 0.94043093, 0.94923858, 0.93020305, 0.92893401,
       0.94796954, 0.93401015, 0.94162437, 0.93654822, 0.94416244])
```

```
scores = cross_val_score(best_grid, xo_test, yo_test, cv=10)
scores
```

Random Forest Model5 with SMOTE (unscaled data and with outliers - xo_train_res & xo_test)

```
best_grid.fit(xo_train_res, yo_train_res)
```

```
RandomForestClassifier(max_depth=30, min_samples_leaf=4, min_samples_split=4,
                       n_estimators=50, random_state=1)
```

Prediction on Train set

```
y_train_predict = best_grid.predict(xo_train_res)
model_score = best_grid.score(xo_train_res, yo_train_res)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(yo_train_res, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(yo_train_res, y_train_predict))
```

```
# predict probabilities
probs = best_grid.predict_proba(xo_train_res)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(yo_train_res, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(yo_train_res, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = best_grid.predict(xo_test)
model_score = best_grid.score(xo_test, yo_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(yo_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(yo_test, y_test_predict))
```

```
# predict probabilities
probs = best_grid.predict_proba(xo_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(yo_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(yo_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

```
importances = best_grid.feature_importances_
indices = np.argsort(importances)
feature_names = list(xo_train_res.columns)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```

```
scores = cross_val_score(best_grid, xo_train_res, yo_train_res, cv=10)
scores
```

```
array([0.93191057, 0.94105691, 0.93997965, 0.94303154, 0.93489318,
       0.93489318, 0.92573754, 0.93997965, 0.91963377, 0.95523906])
```

```
scores = cross_val_score(best_grid, xo_test, yo_test, cv=10)
scores
```

Random Forest Model6 (unscaled data and with outliers - xo_train & xo_test)

```
param_grid = {
    'max_depth': [60, 70, 80, 90, 100],
    'min_samples_leaf': [2,4,6,8,10],
    'min_samples_split': [2,4,6],
    'n_estimators': [50,60,70,80,100],
    'bootstrap': [True, False],
}

rf_model6 = RandomForestClassifier(random_state=1)

grid_search = GridSearchCV(estimator = rf_model6, param_grid = param_grid, cv = 3)
```

```
grid_search.fit(xo_train, yo_train)
```

```
grid_search.best_params_
```

```
{'bootstrap': False,
 'max_depth': 60,
 'min_samples_leaf': 2,
 'min_samples_split': 2,
 'n_estimators': 70}
```

```
best_grid = grid_search.best_estimator_
```

Prediction on Train set ¶

```
y_train_predict = best_grid.predict(xo_train)
model_score = best_grid.score(xo_train, yo_train)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(yo_train, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(yo_train, y_train_predict))
```

```
# predict probabilities
probs = best_grid.predict_proba(xo_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(yo_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(yo_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = best_grid.predict(xo_test)
model_score = best_grid.score(xo_test, yo_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(yo_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(yo_test, y_test_predict))
```

```
# predict probabilities
probs = best_grid.predict_proba(xo_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(yo_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(yo_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

```
importances = best_grid.feature_importances_
indices = np.argsort(importances)
feature_names = list(xo.columns)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(best_grid, xo_train, yo_train, cv=10)
scores
```

```
array([0.96451204, 0.97718631, 0.97969543, 0.96827411, 0.96954315,
       0.9822335 , 0.95431472, 0.97081218, 0.96573604, 0.97715736])
```

```
scores = cross_val_score(best_grid, xo_test, yo_test, cv=10)
scores
```

```
array([0.94970414, 0.9112426 , 0.90236686, 0.9408284 , 0.92011834,
       0.93195266, 0.94674556, 0.9408284 , 0.90207715, 0.94065282])
```

Creating a Train Test Split on scaled datatset of x and fitting it to SMOTE

```
xscaled = sc.fit_transform(x)
```

```
xs_train, xs_test, ys_train, ys_test = train_test_split(xscaled, y, test_size= 0.30, random_state=1, stratify= data['Churn'])
```

```
# Checking dimensions on the train and test data
print('xs_train: ',xs_train.shape)
print('xs_test: ',xs_test.shape)
print('ys_train: ',ys_train.shape)
print('ys_test: ',ys_test.shape)
```

```
xs_train:  (7882, 17)
xs_test:  (3378, 17)
ys_train:  (7882,)
ys_test:  (3378,)
```

```
xs_train_res, ys_train_res = sm.fit_resample(xs_train, ys_train)
```

KNN Model (scaled dataset without Outliers - x_scaled_train & x_test)

```
from sklearn.neighbors import KNeighborsClassifier

KNN_model=KNeighborsClassifier()
KNN_model.fit(x_scaled_train,y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

Prediction on Train set

```
y_train_predict = KNN_model.predict(x_scaled_train)
model_score = KNN_model.score(x_scaled_train, y_train)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_train, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_train, y_train_predict))
```

```
# predict probabilities
probs = KNN_model.predict_proba(x_scaled_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = KNN_model.predict(x_scaled_test)
model_score = KNN_model.score(x_scaled_test, y_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n',metrics.confusion_matrix(y_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_test, y_test_predict))
```

```
# predict probabilities
probs = KNN_model.predict_proba(x_scaled_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(KNN_model, x_scaled_train, y_train, cv=10)
scores
```

```
array([0.9391635 , 0.9531052 , 0.94162437, 0.94923858, 0.94416244,
       0.95939086, 0.93020305, 0.95177665, 0.95050761, 0.95812183])
```

```
scores = cross_val_score(KNN_model, x_scaled_test, y_test, cv=10)
scores
```

KNN with SMOTE (Scaled Dataset without Outliers - xs_train_res & xs_test)

```
KNN_SMOTE = KNN_model.fit(xs_train_res, ys_train_res)
```

Prediction on Train set

```
y_train_predict = KNN_SMOTE.predict(xs_train_res)
model_score = KNN_SMOTE.score(xs_train_res, ys_train_res)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(ys_train_res, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(ys_train_res, y_train_predict))
```

```
# predict probabilities
probs = KNN_SMOTE.predict_proba(xs_train_res)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(ys_train_res, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(ys_train_res, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = KNN_SMOTE.predict(xs_test)
model_score = KNN_SMOTE.score(xs_test, ys_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n',metrics.confusion_matrix(ys_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(ys_test, y_test_predict))
```

```
# predict probabilities
probs = KNN_SMOTE.predict_proba(xs_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(ys_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(ys_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

```
scores = cross_val_score(KNN_SMOTE, xs_train_res, ys_train_res, cv=10)
scores
```

```
array([0.94308943, 0.95020325, 0.95116989, 0.95218718, 0.95523906,
       0.9664293 , 0.95727365, 0.95422177, 0.95320448, 0.95523906])
```

```
scores = cross_val_score(KNN_SMOTE, xs_test, ys_test, cv=10)
scores
```

SVM Model (scaled dataset without Outliers - x_scaled_train & x_test)

```
from sklearn import svm
```

```
svm = svm.SVC(probability=True, random_state = 1)

svm.fit(x_scaled_train, y_train)
```

```
SVC(probability=True, random_state=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Prediction on Train set

```
y_train_predict = svm.predict(x_scaled_train)
model_score = svm.score(x_scaled_train, y_train)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_train, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_train, y_train_predict))
```

```
# predict probabilities
probs = svm.predict_proba(x_scaled_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = svm.predict(x_scaled_test)
model_score = svm.score(x_scaled_test, y_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_test, y_test_predict))
```

```
# predict probabilities
probs = svm.predict_proba(x_scaled_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

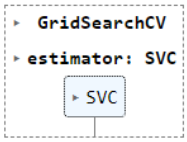
SVM Tuned Model (scaled dataset without Outliers - x_scaled_train & x_test) ¶

```
param_grid = {
    'C': [0.1,1, 10, 100],
    'gamma': [1,0.1,0.01,0.001],
    'kernel': ['poly','rbf','linear','sigmoid']
}

svm = svm.SVC(probability=True, random_state = 1)

grid_search = GridSearchCV(estimator = svm, param_grid = param_grid)
```

```
grid_search.fit(x_scaled_train, y_train)
```




```
grid_search.best_params_
```

```
{'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
```

```
best_grid = grid_search.best_estimator_
```

Prediction on Train set

```
y_train_predict = best_grid.predict(x_scaled_train)
model_score = best_grid.score(x_scaled_train, y_train)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_train, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_train, y_train_predict))
```

```
# predict probabilities
probs = best_grid.predict_proba(x_scaled_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = best_grid.predict(x_scaled_test)
model_score = best_grid.score(x_scaled_test, y_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_test, y_test_predict))
```

```
# predict probabilities
probs = best_grid.predict_proba(x_scaled_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

```
scores = cross_val_score(best_grid, x_scaled_train, y_train, cv=10)
scores
```

```
array([0.96831432, 0.97338403, 0.96827411, 0.96319797, 0.98350254,
       0.9784264 , 0.96827411, 0.9822335 , 0.97461929, 0.98604061])
```

```
scores = cross_val_score(best_grid, x_scaled_test, y_test, cv=10)
scores
```

SVM with SMOTE (Scaled Dataset without Outliers - xs_train_res & xs_test)

```
svm_SMOTE = svm.fit(xs_train_res, ys_train_res)
```

Prediction on Train set

```
y_train_predict = svm_SMOTE.predict(xs_train_res)
model_score = svm_SMOTE.score(xs_train_res, ys_train_res)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(ys_train_res, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(ys_train_res, y_train_predict))
```

```
# predict probabilities
probs = svm_SMOTE.predict_proba(xs_train_res)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(ys_train_res, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(ys_train_res, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = svm_SMOTE.predict(xs_test)
model_score = svm_SMOTE.score(xs_test, ys_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n',metrics.confusion_matrix(ys_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(ys_test, y_test_predict))
```

```
# predict probabilities
probs = svm_SMOTE.predict_proba(xs_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(ys_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(ys_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

```
scores = cross_val_score(svm_SMOTE, xs_train_res, ys_train_res, cv=10)
scores
```

```
array([0.91666667, 0.92682927, 0.92370295, 0.92777213, 0.92675483,
       0.94303154, 0.93692777, 0.94608342, 0.92980671, 0.95015259])
```

```
scores = cross_val_score(svm_SMOTE, xs_test, ys_test, cv=10)
scores
```

CART (Unscaled Dataset without Outliers - x_train & x_test)

```
from sklearn import tree


dt = tree.DecisionTreeClassifier(random_state=1)
dt.fit(x_train, y_train)
```

```
DecisionTreeClassifier(random_state=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Prediction on Train set

```
y_train_predict = dt.predict(x_train)
model_score = dt.score(x_train, y_train)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_train, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_train, y_train_predict))
```

```
# predict probabilities
probs = dt.predict_proba(x_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = dt.predict(x_test)
model_score = dt.score(x_test, y_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n',metrics.confusion_matrix(y_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_test, y_test_predict))
```

```
# predict probabilities
probs = dt.predict_proba(x_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

CART Tuned (Unscaled Dataset without Outliers - x_train & x_test)

```
from sklearn import tree

    dt_tuned = tree.DecisionTreeClassifier(criterion = 'gini', min_samples_leaf = 4, random_state=1)
dt_tuned.fit(x_train, y_train)
```

```
DecisionTreeClassifier(min_samples_leaf=4, random_state=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Prediction on Train set

```
y_train_predict = dt_tuned.predict(x_train)
model_score = dt_tuned.score(x_train, y_train)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_train, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_train, y_train_predict))
```

```
# predict probabilities
probs = dt_tuned.predict_proba(x_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set ¶

```
y_test_predict = dt_tuned.predict(x_test)
model_score = dt_tuned.score(x_test, y_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n',metrics.confusion_matrix(y_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_test, y_test_predict))
```

```
# predict probabilities
probs = dt_tuned.predict_proba(x_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

```
importances = dt_tuned.feature_importances_
indices = np.argsort(importances)
feature_names = list(x.columns)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```

```
scores = cross_val_score(dt_tuned, x_train, y_train, cv=10)
scores
```

```
array([0.9252218 , 0.91634981, 0.92258883, 0.94035533, 0.92893401,
       0.93527919, 0.90736041, 0.91751269, 0.93020305, 0.92766497])
```

```
scores = cross_val_score(dt_tuned, x_test, y_test, cv=10)
scores
```

Emsemble Techniques

Bagging (unscaled data and without outliers - x_train & x_test)

```
from sklearn.ensemble import BaggingClassifier
Bagging_model=BaggingClassifier(base_estimator=dt_tuned,n_estimators=100,random_state=1)
Bagging_model.fit(x_train, y_train)
```

```
BaggingClassifier(base_estimator=DecisionTreeClassifier(min_samples_leaf=4,
                                                        random_state=1),
                  n_estimators=100, random_state=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Prediction on Train set

```
y_train_predict = Bagging_model.predict(x_train)
model_score = Bagging_model.score(x_train, y_train)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_train, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_train, y_train_predict))
```

```
# predict probabilities
probs = Bagging_model.predict_proba(x_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = Bagging_model.predict(x_test)
model_score = Bagging_model.score(x_test, y_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_test, y_test_predict))
```

```
# predict probabilities
probs = Bagging_model.predict_proba(x_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

```
scores = cross_val_score(Bagging_model, x_train, y_train, cv=10)
scores
```

```
array([0.93409379, 0.94930292, 0.9606599 , 0.94543147, 0.93401015,
       0.95431472, 0.93401015, 0.94416244, 0.9428934 , 0.95177665])
```

```
scores = cross_val_score(Bagging_model, x_test, y_test, cv=10)
scores
```

```
feature_importances = np.mean([
    tree.feature_importances_ for tree in Bagging_model.estimators_
], axis=0)
importances = feature_importances
indices = np.argsort(importances)
feature_names = list(x.columns)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```

Bagging with SMOTE (unscaled data and without outliers - x_train_res & x_test)

```
Bagging_model_SMOTE = Bagging_model.fit(x_train_res, y_train_res)
```

Prediction on Train set

```
y_train_predict = Bagging_model_SMOTE.predict(x_train_res)
model_score = Bagging_model_SMOTE.score(x_train_res, y_train_res)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_train_res, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_train_res, y_train_predict))
```

```
# predict probabilities
probs = Bagging_model_SMOTE.predict_proba(x_train_res)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train_res, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train_res, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = Bagging_model_SMOTE.predict(x_test)
model_score = Bagging_model_SMOTE.score(x_test, y_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_test, y_test_predict))
```

```
# predict probabilities
probs = Bagging_model_SMOTE.predict_proba(x_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

```
scores = cross_val_score(Bagging_model_SMOTE, x_train_res, y_train_res, cv=10)
scores
```

```
array([0.93191057, 0.93597561, 0.94404883, 0.93997965, 0.93591048,
       0.96032553, 0.93591048, 0.95422177, 0.93489318, 0.965412  ])
```

```
scores = cross_val_score(Bagging_model_SMOTE, x_test, y_test, cv=10)
scores
```

```
feature_importances = np.mean([
    tree.feature_importances_ for tree in Bagging_model_SMOTE.estimators_
], axis=0)
importances = feature_importances
indices = np.argsort(importances)
feature_names = list(x.columns)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```

Ada Boost (Unscaled Dataset without Outliers - x_train & x_test)

```
from sklearn.ensemble import AdaBoostClassifier

ADB_model = AdaBoostClassifier(n_estimators=300,random_state=1)
ADB_model.fit(x_train,y_train)
```

Prediction on Train set

```
y_train_predict = ADB_model.predict(x_train)
model_score = ADB_model.score(x_train, y_train)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_train, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_train, y_train_predict))
```

```
# predict probabilities
probs = ADB_model.predict_proba(x_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = ADB_model.predict(x_test)
model_score = ADB_model.score(x_test, y_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n',metrics.confusion_matrix(y_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_test, y_test_predict))
```

```
# predict probabilities
probs = ADB_model.predict_proba(x_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

AdaBoost Tuned (Unscaled Dataset without Outliers - x_train & x_test)

```
param_grid = {
    "base_estimator":[DecisionTreeClassifier(max_depth=7),DecisionTreeClassifier(max_depth=8),DecisionTreeClassifier(max_depth=9)
    "n_estimators": [110,210,310],
    "learning_rate":[0.01,2,0.1]
}

ADB_tuned = AdaBoostClassifier(random_state=1)

grid_search = GridSearchCV(estimator = ADB_tuned, param_grid = param_grid)
```

```
grid_search.fit(x_train, y_train)
```

```
grid_search.best_params_
```

```
{'base_estimator': DecisionTreeClassifier(max_depth=7),
 'learning_rate': 0.1,
 'n_estimators': 310}
```

```
best_grid = grid_search.best_estimator_
```

Prediction on Train set

```
y_train_predict = best_grid.predict(x_train)
model_score = best_grid.score(x_train, y_train)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_train, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_train, y_train_predict))
```

```
# predict probabilities
probs = best_grid.predict_proba(x_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = best_grid.predict(x_test)
model_score = best_grid.score(x_test, y_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n',metrics.confusion_matrix(y_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_test, y_test_predict))
```

```
# predict probabilities
probs = best_grid.predict_proba(x_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr);
```

```
importances = best_grid.feature_importances_
indices = np.argsort(importances)
feature_names = list(x.columns)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```

Gradient Boosting (unscaled data and without outliers - x_train & x_test)

```
from sklearn.ensemble import GradientBoostingClassifier
gbcl = GradientBoostingClassifier(random_state=1)
gbcl = gbcl.fit(x_train, y_train)
```

Prediction on Train set

```
y_train_predict = gbcl.predict(x_train)
model_score = gbcl.score(x_train, y_train)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_train, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_train, y_train_predict))
```

```
# predict probabilities
probs = gbcl.predict_proba(x_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = gbcl.predict(x_test)
model_score = gbcl.score(x_test, y_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_test, y_test_predict))
```

```
# predict probabilities
probs = gbcl.predict_proba(x_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Gradient Boosting Tuned (unscaled data and without outliers - x_train & x_test)

```
gbcl_tuned = GradientBoostingClassifier(init=RandomForestClassifier(random_state=1),random_state=1)
gbcl_tuned.fit(x_train,y_train)
```

```
GradientBoostingClassifier(init=RandomForestClassifier(random_state=1),
                           random_state=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
param_grid = {
    'max_depth': [30,40,50],
    "subsample":[0.8,0.9,1],
    "max_features":[0.7,0.8,0.9,1],
    'n_estimators': [40,45,50]
}
gbcl_tuned = GradientBoostingClassifier(init=RandomForestClassifier(random_state=1),random_state=1)

grid_search = GridSearchCV(estimator = gbcl_tuned, param_grid = param_grid)
```

```
grid_search.fit(x_train, y_train)
```

```
GridSearchCV(estimator=GradientBoostingClassifier(init=RandomForestClassifier(random_state=1),
                                                  random_state=1),
             param_grid={'max_depth': [30, 40, 50],
                         'max_features': [0.7, 0.8, 0.9, 1],
                         'n_estimators': [40, 45, 50],
                         'subsample': [0.8, 0.9, 1]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
grid_search.best_params_
```

```
{'max_depth': 50, 'max_features': 0.8, 'n_estimators': 45, 'subsample': 0.9}
```

```
best_grid = grid_search.best_estimator_
```

Prediction on Train set

```
y_train_predict = best_grid.predict(x_train)
model_score = best_grid.score(x_train, y_train)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_train, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_train, y_train_predict))
```

```
# predict probabilities
probs = best_grid.predict_proba(x_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = best_grid.predict(x_test)
model_score = best_grid.score(x_test, y_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_test, y_test_predict))
```

```
# predict probabilities
probs = best_grid.predict_proba(x_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

```
importances = best_grid.feature_importances_
indices = np.argsort(importances)
feature_names = list(x.columns)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```

XGBoost (unscaled data and without outliers - x_train & x_test)

```
from xgboost import XGBClassifier
xgb = XGBClassifier(random_state=1)
xgb.fit(x_train,y_train)
```

Prediction on Train set

```
y_train_predict = xgb.predict(x_train)
model_score = xgb.score(x_train, y_train)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_train, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_train, y_train_predict))
```

```
# predict probabilities
probs = xgb.predict_proba(x_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set ¶

```
y_test_predict = xgb.predict(x_test)
model_score = xgb.score(x_test, y_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_test, y_test_predict))
```

```
# predict probabilities
probs = xgb.predict_proba(x_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

```
importances = xgb.feature_importances_
indices = np.argsort(importances)
feature_names = list(x.columns)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```

XGBoost Tuned (unscaled data and without outliers - x_train & x_test)

```
param_grid  = {
    "n_estimators": np.arange(10,100,20),
    "scale_pos_weight":[0,1,2,5],
    "subsample":[0.5,0.7,0.9,1],
    "learning_rate":[0.01,0.1,0.2,0.05],
    "gamma":[0,1,3],
    "colsample_bytree":[0.5,0.7,0.9,1],
    "colsample_bylevel":[0.5,0.7,0.9,1]
}

xgb_tuned = XGBClassifier(random_state=1)

grid_search = GridSearchCV(estimator = xgb_tuned, param_grid = param_grid)
```

```
grid_search.fit(x_train, y_train)
```

```
grid_search.best_params_
```

```
{'colsample_bylevel': 0.9,
 'colsample_bytree': 1,
 'gamma': 0,
 'learning_rate': 0.2,
 'n_estimators': 90,
 'scale_pos_weight': 2,
 'subsample': 1}
```

```
best_grid = grid_search.best_estimator_
```

Prediction on Train set

```
y_train_predict = best_grid.predict(x_train)
model_score = best_grid.score(x_train, y_train)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_train, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_train, y_train_predict))
```

```
# predict probabilities
probs = best_grid.predict_proba(x_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = best_grid.predict(x_test)
model_score = best_grid.score(x_test, y_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_test, y_test_predict))
```

```
# predict probabilities
probs = best_grid.predict_proba(x_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

```
importances = best_grid.feature_importances_
indices = np.argsort(importances)
feature_names = list(x.columns)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```

```
scores = cross_val_score(best_grid, x_train, y_train, cv=10)
scores
```

```
array([0.95690748, 0.96451204, 0.95939086, 0.96573604, 0.95558376,
       0.96573604, 0.94796954, 0.95939086, 0.95177665, 0.96827411])
```

```
scores = cross_val_score(best_grid, x_test, y_test, cv=10)
scores
```

K-Means Clustering

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_samples, silhouette_score
```

```
df = data.drop(['AccountID'], axis = 1)
```

```
df_scaled = StandardScaler().fit_transform(df)
```

```
df_scaled
```

Calculating WSS for values of K - Elbow Method

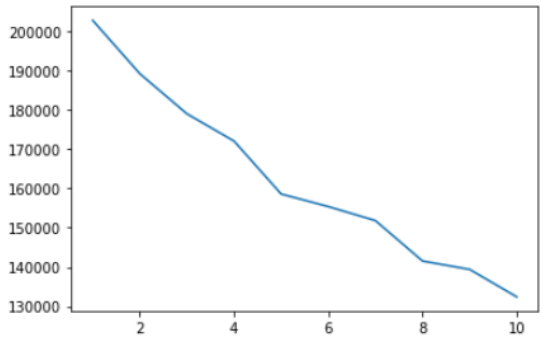
```
wss =[]
```

```
for i in range(1,11):
    KM = KMeans(n_clusters=i,random_state=1)
    KM.fit(df_scaled)
    wss.append(KM.inertia_)
```

```
wss
```

```
plt.plot(range(1,11), wss)
```

```
[<matplotlib.lines.Line2D at 0x12402333c40>]
```




```
k_means = KMeans(n_clusters = 2,random_state=1)
k_means.fit(df_scaled)
labels = k_means.labels_
```

```
silhouette_score(df_scaled,labels)
```

```
0.12336437604603798
```

```
silhouette_samples(df_scaled,labels).min()
```

```
0.0013580240285513245
```

```
k_means = KMeans(n_clusters = 4,random_state=1)
k_means.fit(df_scaled)
labels = k_means.labels_
```

```
silhouette_score(df_scaled,labels)
```

```
0.07424420609512918
```

```
silhouette_samples(df_scaled,labels).min()
```

```
-0.17533947454568208
```

```
k_means = KMeans(n_clusters = 3,random_state=1)
k_means.fit(df_scaled)
labels = k_means.labels_
```

```
silhouette_score(df_scaled,labels)
```

```
0.0742940376682887
```

```
silhouette_samples(df_scaled,labels).min()
```

```
-0.07283738766512156
```

```
df["df_kmeans3"] = labels
df.head()
```

```
df.df_kmeans3.value_counts().sort_index()
```

```
0    1841
1    4535
2    4884
Name: df_kmeans3, dtype: int64
```

```
clust_profile=df
clust_profile=clust_profile.groupby('df_kmeans3').mean()
clust_profile['df_cust_segment']=df.df_kmeans3.value_counts().sort_index()
np.round(clust_profile,2)
```

```
fig, axes = plt.subplots(nrows=5,ncols=2)
fig.set_size_inches(20,24)
a = sns.countplot(x='City_Tier', hue='df_kmeans3', data=df, ax = axes[0][0])
a = sns.countplot(x='Payment', hue='df_kmeans3', data=df, ax = axes[0][1])
a = sns.countplot(x='Gender', hue='df_kmeans3', data=df, ax=axes[1][0])
a = sns.countplot(x='Service_Score', hue='df_kmeans3', data=df, ax=axes[1][1])
a = sns.countplot(x='account_segment', hue='df_kmeans3', data=df, ax = axes[2][0])
a = sns.countplot(x='CC_Agent_Score', hue='df_kmeans3', data=df, ax = axes[2][1])
a = sns.countplot(x='Marital_Status', hue='df_kmeans3', data=df, ax = axes[3][0])
a = sns.countplot(x='Complain_ly', hue='df_kmeans3', data=df, ax = axes[3][1])
a = sns.countplot(x='Login_device', hue='df_kmeans3', data=df, ax = axes[4][0])
a = sns.countplot(x='Churn', hue='df_kmeans3', data=df, ax = axes[4][1])
```

```
fig, axes = plt.subplots(nrows=4,ncols=2)
fig.set_size_inches(20,20)
a = sns.boxplot(x='df_kmeans3', y='Tenure', data=df, ax = axes[0][0])
a = sns.boxplot(x='df_kmeans3', y='CC_Contacted_LY', data=df, ax = axes[0][1])
a = sns.boxplot(x='df_kmeans3', y='Account_user_count', data=df, ax=axes[1][0])
a = sns.boxplot(x='df_kmeans3', y='rev_per_month', data=df, ax=axes[1][1])
a = sns.boxplot(x='df_kmeans3', y='rev_growth_yoy', data=df, ax = axes[2][0])
a = sns.boxplot(x='df_kmeans3', y='coupon_used_for_payment', data=df, ax = axes[2][1])
a = sns.boxplot(x='df_kmeans3', y='Day_Since_CC_connect', data=df, ax = axes[3][0])
a = sns.boxplot(x='df_kmeans3', y='cashback', data=df, ax = axes[3][1])
```

THE END