

## 1) Introduction:

### Problem Statement:

Prediction whether a customer is going to churn based on the various usage parameters of the online E-Commerce company.

In the digital world, people prefer to do online shopping where they purchase becomes easy, fast and quite affordable. Competition is tough in the online market where customers are free to choose from plenty of providers even within one product category. Hence satisfying customers are very important, as unsatisfied customers churn at a clip.

### Need for Study:

#### ➤ **Customer Churn:**

**Customer Churn or Customer Attrition** is the tendency of customers to abandon the products and stop interacting with the site or service of the company. Churn rate is a health indicator for businesses, whose customers pay them on regular basis. Churn rate often correlate with revenue lost and amount spent on acquisition of new customers.

#### ➤ **Importance of predicting Customer Churn:**

The cost of churn includes both revenue lost and marketing costs spent on acquisition of new customers. Businesses with high churn rate will face huge financial loss, as acquiring a new customer is far expensive than retaining an existing customers. It is also identified that companies with high churn rates besides ceasing their relationship with ex-customers, they damage their future acquisitions by creating negative impact about them and their products. Hence, even a small percent of churn rate shouldn't be overlooked and **reducing churn is the key factor of online business**.

### Understanding the business opportunity:

- In order to retain the customers and reduce churn rates, businesses should monitor:
  - customer relationship with products and their service
  - encourage clients on sharing their experience
  - on solving issues promptly to satisfy the customers
- Businesses should be able to
  - Predict in advance about customers' churn
  - Formulate the marketing actions necessary to have higher retention rate on the customers.

Hence, the behavioural pattern of each customer should be analysed, segment the customers, identify the potential churners and take appropriate actions to garner their trust and retain them in the business. Various **churn prediction modelling techniques** are used to understand the customer behaviour and indicate the chances of customer churn.

## Data Report:

Reduction of churn rates involves active and continuous monitoring of customer purchase behaviour. Companies mostly collect customer demographics, user behavioural, support and contextual features typically for most of the business model.

- **Customer demographic features** - primary customer information (e.g., age, location, income)
- **User behavioural features** – customers use of service & product (e.g., Tenure of customer in organization, Number of hours spend on mobile application or website, Preferred order category of customer, Percentage increases in order)
- **Support features** – interactions with customer support (e.g., Complaints raised, Satisfactory score of customer on service)
- **Contextual features** – contextual information about customers

The above information of the customers could be collected by tracking their online usage and purchase details. Besides their use of service, it is very important to understand what value customers are generating from the product and what is the likelihood of churn.

To build a churn prediction model, firstly data science specialist must find and collect the data to work. Once the data is selected, it should be prepared, pre-processed and converted for building prediction models. In the prediction of whether customer is going to churn based on the various usage parameters of the online E-Commerce company, we have a dataset with purchase information of about **5630 customers** explained with the below **20 attributes or variables**.

Data Dictionary:	
Variable	Description
CustomerID	Unique customer ID
Churn	Churn Flag
Tenure	Tenure of customer in organization
PreferredLoginDevice	Preferred login device of customer
CityTier	City tier
WarehouseToHome	Distance in between warehouse to home of customer
PreferredPaymentMode	Preferred payment method of customer
Gender	Gender of customer
HourSpendOnApp	Number of hours spend on mobile application or website
NumberOfDeviceRegistered	Total number of deceives is registered on particular customer
PreferedOrderCat	Preferred order category of customer in last month
SatisfactionScore	Satisfactory score of customer on service
MaritalStatus	Marital status of customer
NumberOfAddress	Total number of added on particular customer
Complain	Any complaint has been raised in last month
OrderAmountHikeFromlastYear	Percentage increases in order from last year
CouponUsed	Total number of coupon has been used in last month
OrderCount	Total number of orders has been places in last month
DaySinceLastOrder	Day Since last order by customer
CashbackAmount	Average cashback received in last month

**Fig 1. Data Dictionary**

## 2. EDA and Business Implication:

### ➤ Duplicate & Missing Values

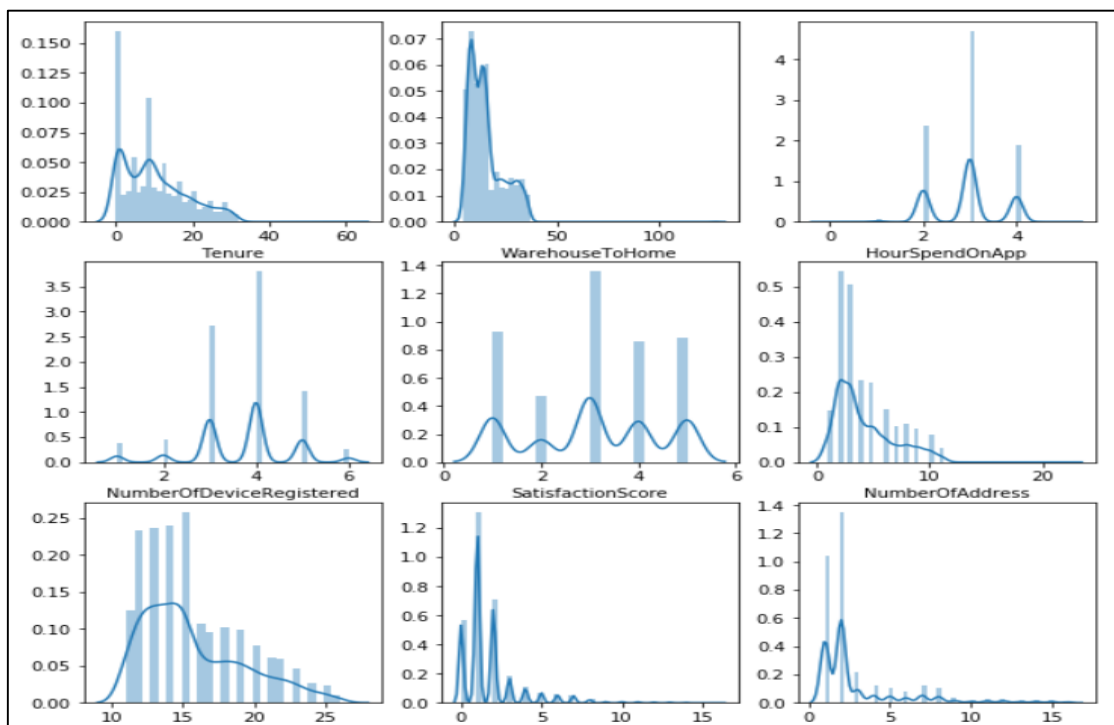
- There are no duplicate values in the given dataset.
- We could see there are missing values only in the 'DaySinceLastOrder', 'OrderAmountHikeFromlastYear', 'Tenure', 'OrderCount', 'CouponUsed', 'HourSpentOnApp', 'WarehouseToHome' variables.
- Since the percentage of missing values are very less ranging from 4.4 % to 5.4%, we could drop or impute the missing values.
- For further analysis, we would recommend to **impute the missing values with median** of the respective variable, as **outliers** are present.
- We could infer that some of the numerical variables have only few distinct values. Hence, these **numerical variables are converted into object types** with appropriate categories for each variable.
- It could be found that some of the values for the categorical variable are repeated. That is, 'Mobile Phone' & 'Phone' are 2 different values in the 'PreferredLoginDevice' variable, but they represent same device.
- Similarly, 'Credit Card' & 'CC' in the 'PreferredPaymentMode' and 'Mobile Phone' & 'Mobile' in the 'PreferredOrderedCat' variables are repeated. Hence, we need to align these categories.

### ➤ Univariate Analysis - Analysis Over A Single Variable

We can describe patterns found in univariate data including **central tendency** (mean, mode and median) and **dispersion**: range, variance, maximum, minimum, quartiles (including the interquartile range), and standard deviation.

#### DISTPLOT

- **Histogram** could be drawn for the analysis of the numerical variables. Distplot() method is used in Seaborn to draw the histogram.
- **Since some of the numerical variables contains only few distinct data, they don't show much distribution among the data available in the distplot()**



**Fig 2. Univariate Analysis of numerical variables using DISTPLOT**

- From the above plot, we can see that there are some numerical variables that are having very few unique values, we would need to explore these variables more.

### HourSpendOnApp:

There are only 6 distinct values in the '**HourSpendOnApp**' variable. Hence, based on the hours spent on App, customers are grouped into 3 groups:

- **Low Usage** – Less than 2 hours
- **Medium Usage** – 2 to 4 hours
- **High Usage** – Greater than 4 hours

### SatisfactionScore, NumberOfDeviceRegistered, NumberOfAddress, CouponUsed, OrderCount:

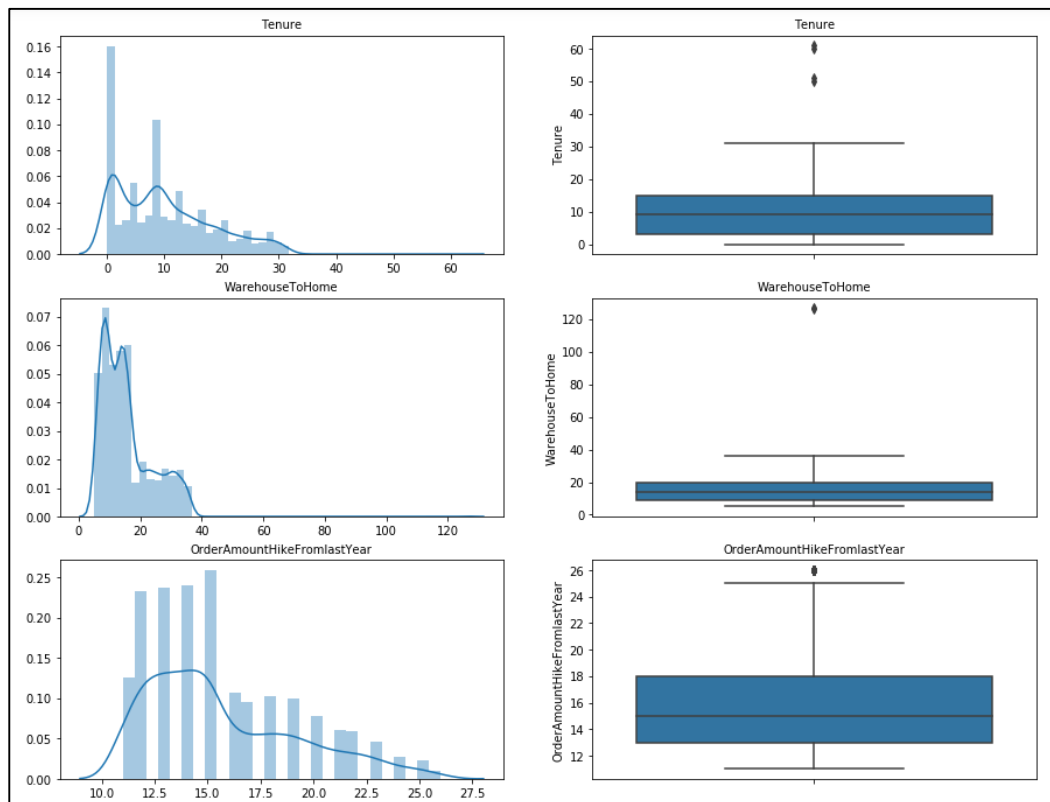
- '**SatisfactionScore**' of the customers ranges from 1 to 5, hence converting this variable into **object** type for better analysis.
- '**NumberOfDeviceRegistered**' has 6 distinct values from 1 to 6, hence grouping 2 groups as '**One or Two**' & '**Three or more**'
- '**NumberOfAddress**' have 15 distinct values and we have grouped them into 5 bins such as 'Less than 2', '2 to 4', '4 to 6', '6 to 8' & '8 plus'
- '**CouponUsed**' have 17 distinct values from 1 to 17, hence grouping them to 3 groups such as '**Less than 3**', '**3 to 8**' & '**8 plus**'
- '**OrderCount**' have 16 distinct values and we have grouped them into 4 bins such as '**Less than 3**', '**3 to 6**', '**6 to 10**' & '**10 plus**'

### Analysis of numerical variables

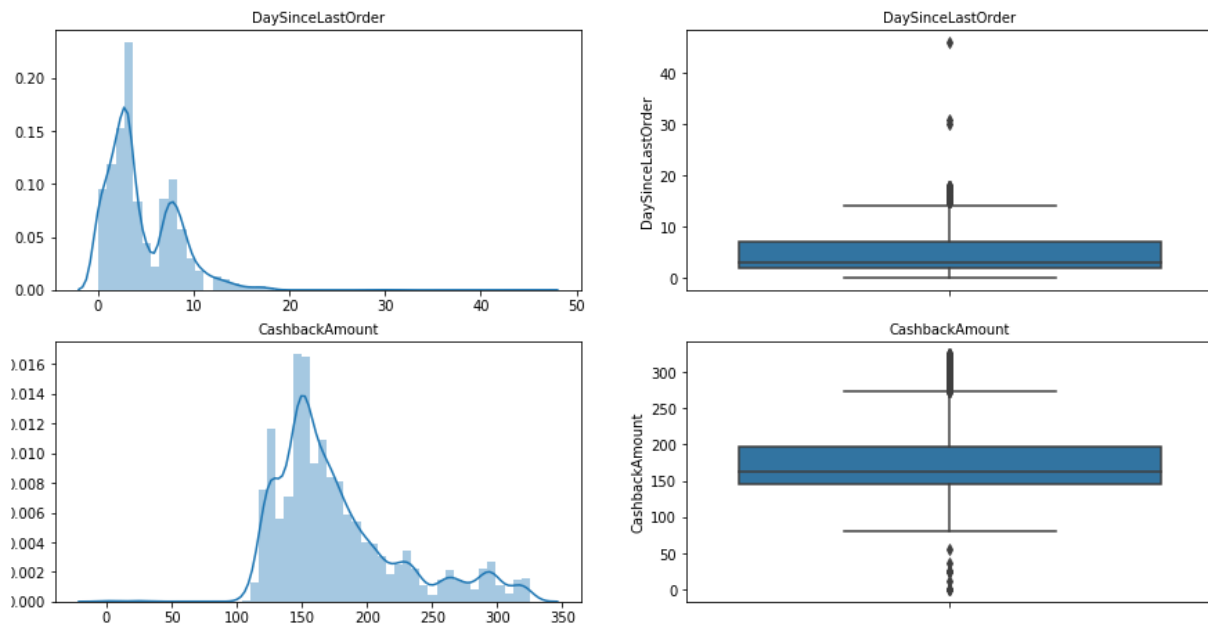
After the transformation of the above variables, we have the following numerical variables for further analysis.

- Tenure
- WarehouseToHome
- OrderAmountHikeFromLastYear
- DaySinceLastOrder
- CashbackAmount

The distribution and spread of the above continuous variables are visualized with the **DISTPLOT & BOXPLOT**.



**Fig 3. Analysis of variables - Tenure, WarehouseToHome, OrderAmountFromLastYear using DISTPLOT & BOXPLOT**



**Fig 4. Analysis of variables – DaySinceLastOrder, CashbackAmount using DISTPLOT & BOXPLOT**

From the above plots we can observe, the following points;

- Most of the customers have spent between 0 to 10 months with the vendor, however there are some customers who have been with the vendor for 30 to 40 months as well.
- Most of the Customers are within 20 km of the vendors warehouses. The number of customers within 20 to 40 km of the warehouses are also appreciable but very few customers are beyond 40 km from the customer warehouses.
- For most customers the hike in order amount has been between 10 to 17.5% however for few customers the hike has been more than 20%.
- Most of the customers have transacted with the vendor recently, between 0 to 10 days from when the data was collected. There are very few customers who have not ordered since 20 days or more.
- Most of the customers have received a cashback amount of 140 to 160, there are many customers who have received more than 250 in cashback as well, and there are very few customers who have received cash back less than 75.
- **All variables have outliers on the higher side, whereas Cashback Amount variable have outliers beyond the upper quartile region and below the lower quartile region. This is in conformity of the understanding of the distributions.**

#### **Analysis of categorical variables:**

After the transformation of the certain variables, we have the following categorical variables for further analysis.

- Churn
- PreferredLoginDevice
- CityTier
- PreferredPaymentMode
- Gender
- HourSpendOnApp
- NumberOfDeviceRegistered
- PreferredOrderCat
- SatisfactionScore
- MaritalStatus
- NumberOfAddress
- Complain
- CouponUsed
- OrderCount

## COUNTPLOT ( )

Count plot is used to graphically visualise the categorical data. It shows the bar chart to display the number of occurrence for each categorical data present in the dataset.

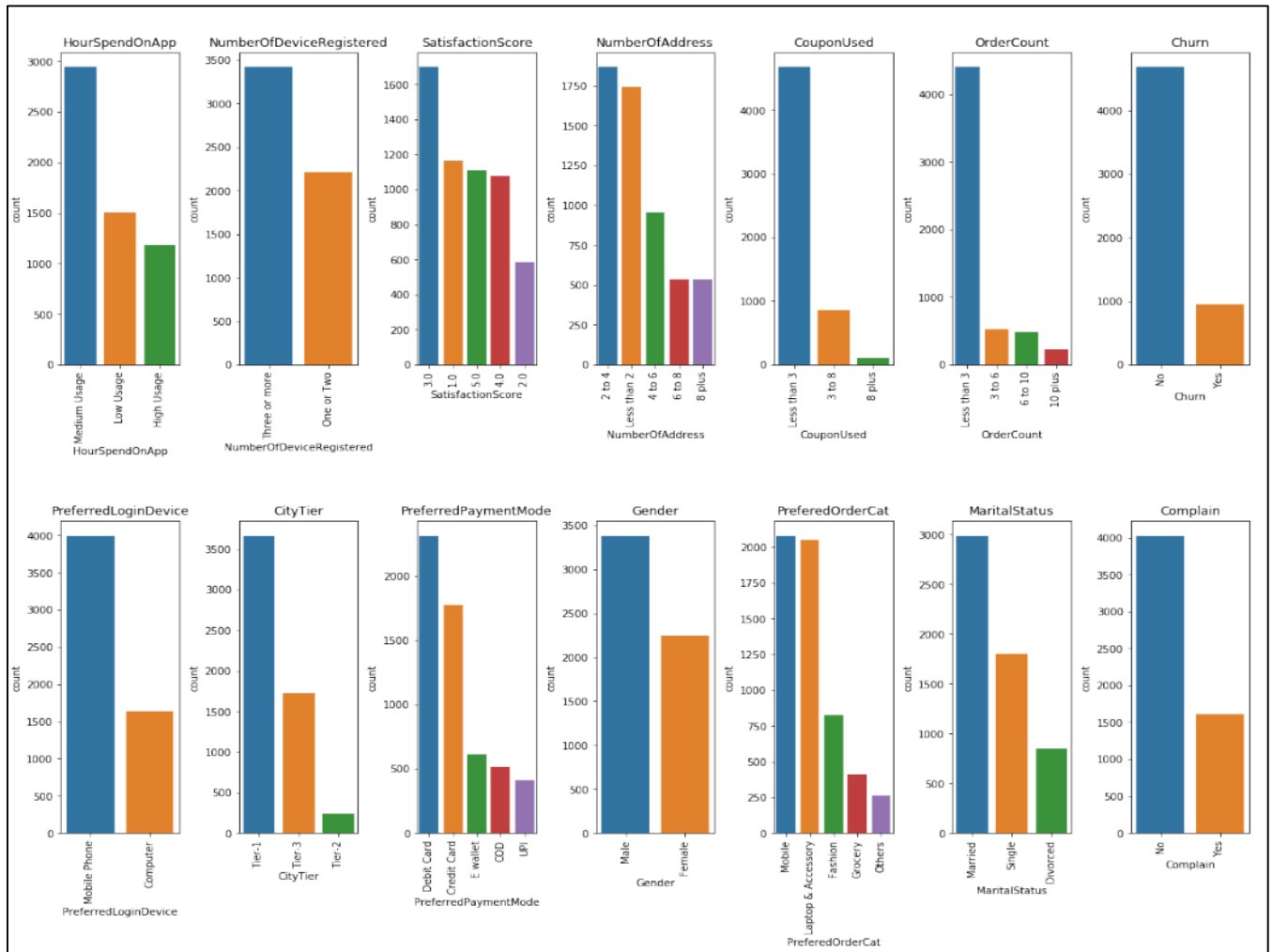


Fig 5. Proportion of Customers under each categorical variable.

From the above countplot, we could visualize the following points:

- Most of the customers are '**Medium Usage**' customers and they spend between 2 to 4 hours on the App. There are almost equal number of 'Low Usage' and 'High Usage' customers who spend 'Less than 2 hours' & 'Greater than 4 hours' in the App or website.
- Most of the customers have registered **three or more devices** on the App or website.
- Among the Satisfaction Score of 1 to 5, most of the customers have given a score of **3**. Considerable amount of customers have scored 1, 4 and 5 and least customers given a score of 2.
- Most of the customers have used between **2 to 4** addresses on the App or website. And there are almost equivalent number of customers who have registered **Less than 2** addresses
- Most of the customers have used **Less than 3 coupons in the last month** on the App or website.
- Most of the customers have placed **Less than 3 orders in the last month**.
- In the given dataset, there are more **non churners** compared to churners.

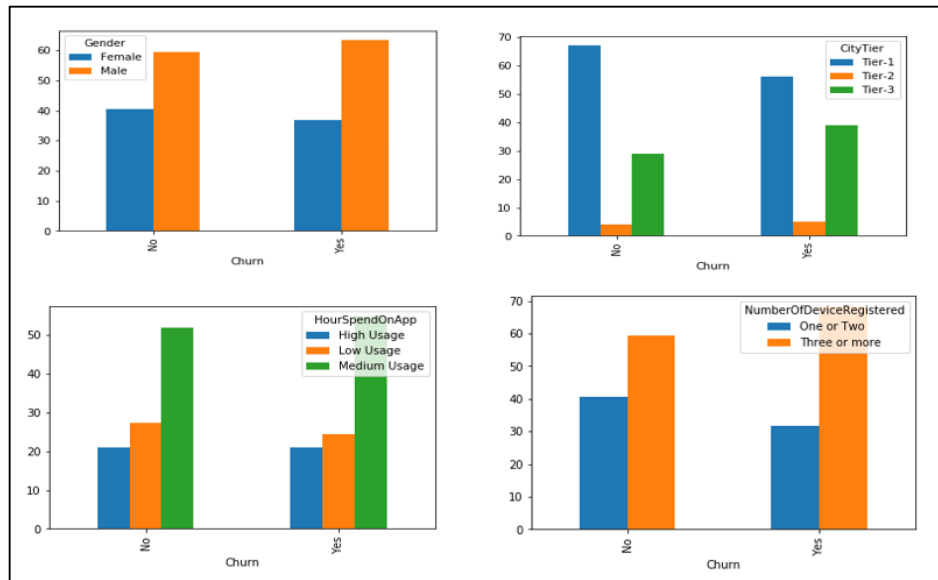
Proportion of Customers as per Churn	
No	0.831616
Yes	0.168384

Fig 6. Proportion of Churned & Non-Churned Customers.

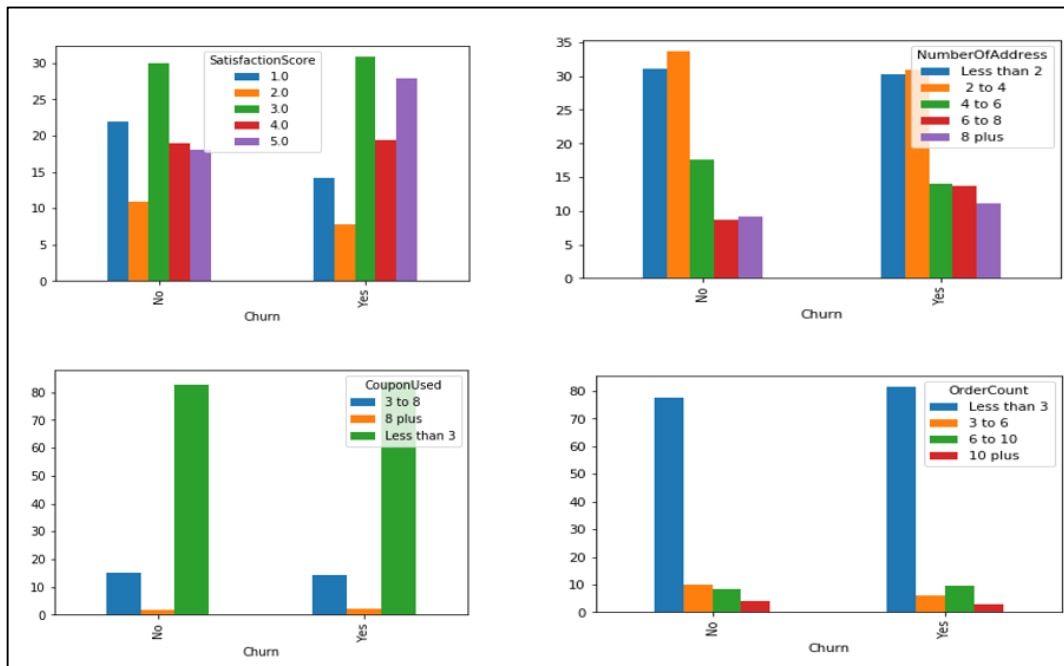
- **Mobile Phones** are the most preferred Login Device for the customers.
- Customers who belong to **Tier - 1** purchase more compared to Tier – 2 & Tier – 3.
- **Debit Card** is the most preferred Mode of Payment of the customers. Credit Card is also used by frequently by the customers as their Mode of Payment.
- Most of the customers are **Male** in the given dataset. We could also see most of the customers are **Married**.
- **'Mobile'** and **'Laptop & Accessory'** are the two most preferred categories by the customers compared to 'Fashion', 'Grocery' and 'Others'.
- Most of the customers have not raised complain in the last month.

#### ➤ Bivariate Analysis - Analysis Over A Two Variable:

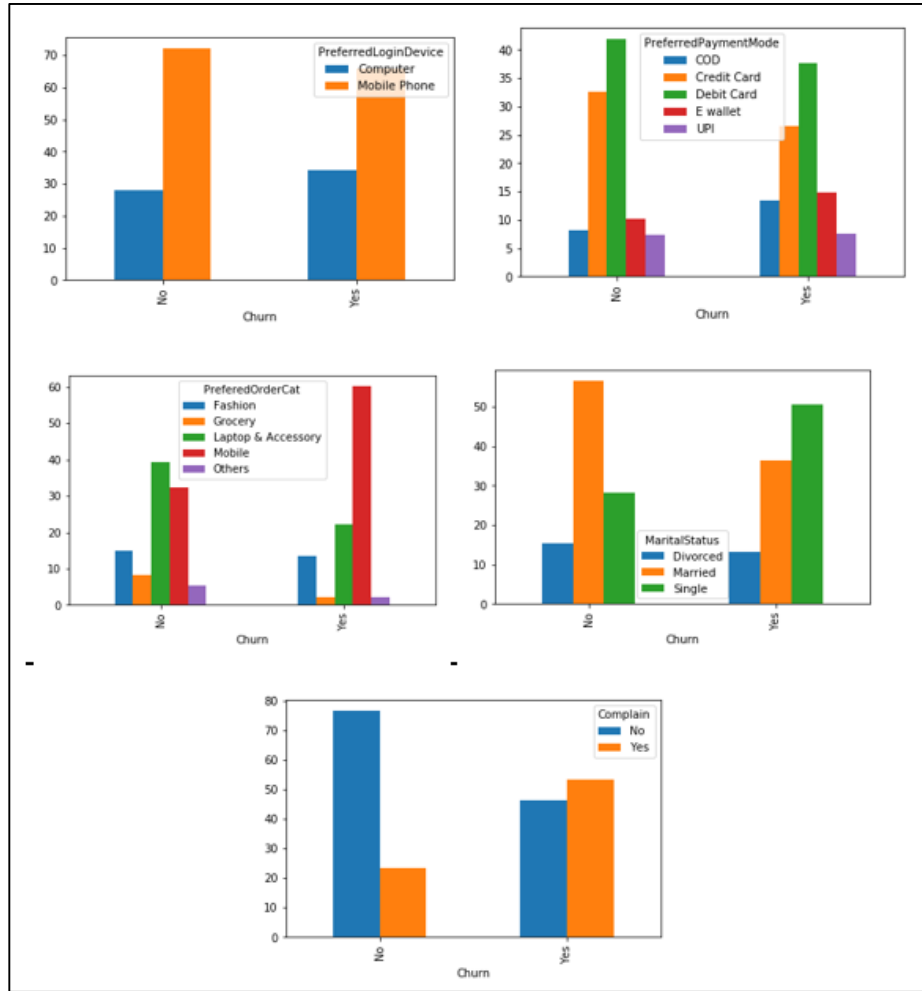
As we are trying to predict whether customers are going to churn or not, we will try to analyse all the variables based on the 'Churn' variable.



**Fig 7. Analysis of Gender, CityTier, HourSpendOnApp, NumberOfDevicesRegistered with 'Churn'**



**Fig 8. Analysis of SatisfactionScore, NumberOfAddress, CouponsUsed, OrderCount with 'Churn'**



**Fig 9. Analysis of PreferredLoginDevice, PreferredPaymentMode, PreferredOrderCat, MaritalStatus, Complain with 'Churn'**

From the above plots, we could observe the following points:

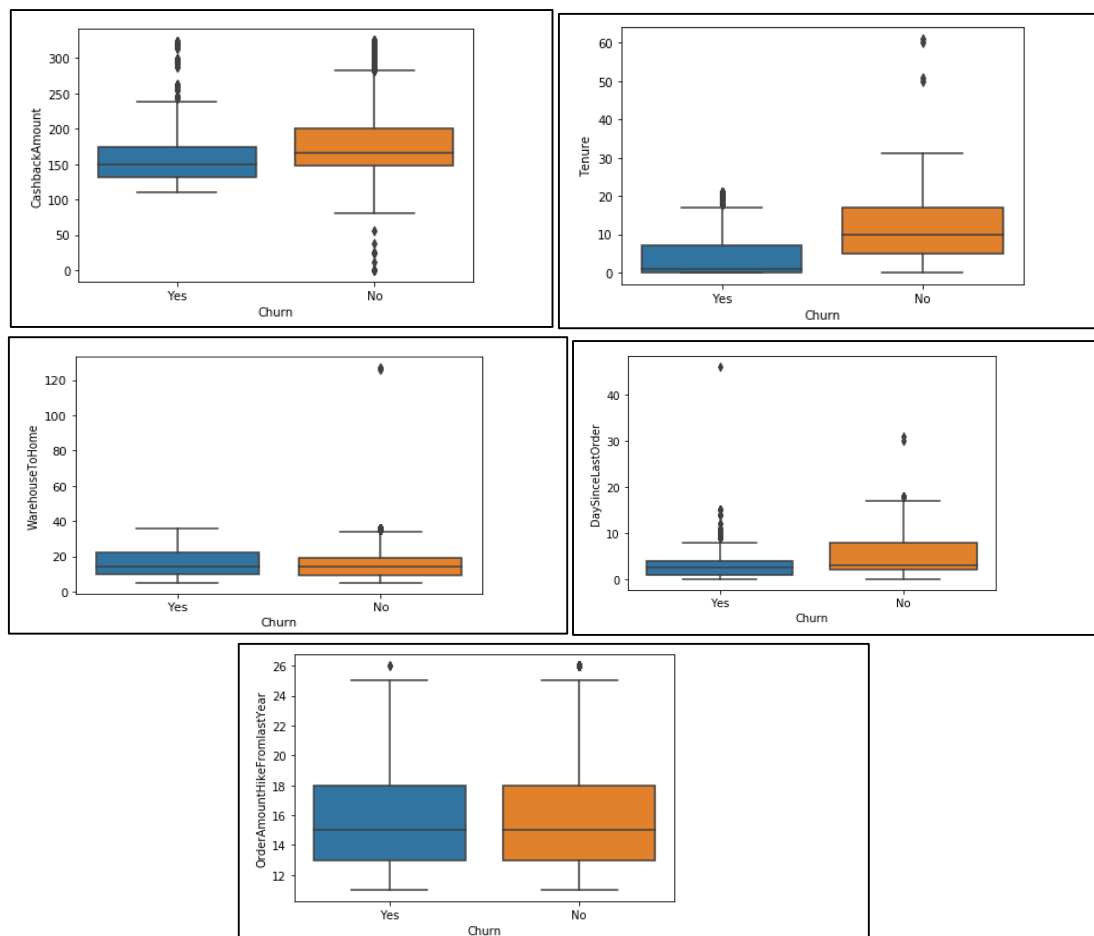
- **Male** customers are predominant in both Churners & Non-Churners.
- **Tier-1** customers are predominant in both Churners & Non-Churners.
- Customers who spend between 2 to 4 hours on the App i.e **Medium Usage** customers are predominant in both Churners & Non-Churners.
- Both Churners & Non-Churners have **three or more devices registered** on the App or website.
- Most of the Non-Churners have the Satisfaction Score of 3, whereas Churners have the Satisfaction Score of 3 and 5. It is inferred that most of the customers tend to churn even after having Satisfaction Score of 5.
- Both Churners & Non-Churners have added **zero to four addresses** on the App or website.
- Both Churners & Non-Churners have used **less than 3 coupons** in the last month.
- It is observed that both Churner & Non-Churners have placed **less than 3 orders** in the last month.
- **Mobile Phones** are the preferred Login Device for both Churners & Non-Churners.
- **Debit Card** is the preferred payment method for both Churners & Non-Churners followed by Credit Card.
- **'Laptop & Accessory'** is the most ordered category of the Non-Churners, whereas the most ordered category of the Churners is **'Mobile'**.
- Most of the **Married** customers are Non-Churners, whereas Most of the **Single** customers are Churners.
- Most of the Non-Churners haven't raised complains in the last month, whereas the **Churners have raised Complaints in the last month.**



	Churners_Mode	Non_Churners_Mode
HourSpendOnApp	Medium Usage	Medium Usage
NumberOfDeviceRegistered	Three or more	Three or more
SatisfactionScore	3	3
NumberOfAddress	2 to 4	2 to 4
PreferredLoginDevice	Mobile Phone	Mobile Phone
CouponUsed	Less than 3	Less than 3
OrderCount	Less than 3	Less than 3
CityTier	Tier-1	Tier-1
PreferredPaymentMode	Debit Card	Debit Card
Gender	Male	Male
PreferedOrderCat	Mobile	Laptop & Accessory
MaritalStatus	Single	Married
Complain	Yes	No

**Fig 10. Mode of Churner & Non-Churners**

**Analysis of categorical variables with ‘Churn’:**



**Fig 11. Analysis of categorical variables with ‘Churn’**

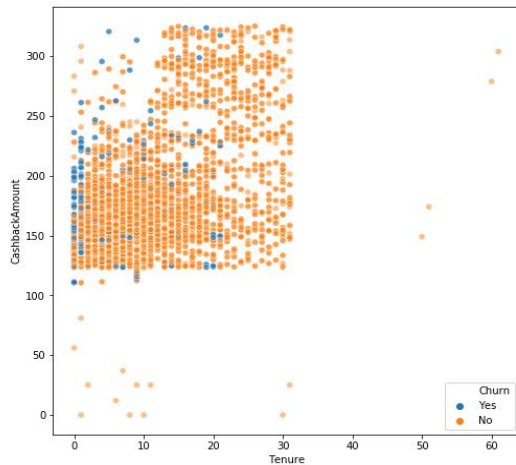
- Most of the non-churned customers have received a cashback amount of **75 to 280**, there are many non-churned customers who have received more than 280 in cashback as well. **Even there are few customers who have received cash back less than 75 and they haven't churned at all.**
- Whereas, **most of churned customers have received a cashback amount of 100 to 250.** Even there are customers who have received more than 250 cash back and churned.
- **Tenure of most of the churned customers are between 0 and 10.** Few churners have a tenure of 10 to 25 also, whereas most of the non-churners have a tenure of 0 and 35, few of them have 50 & 60.
- Most of the churned & Non-churned Customers are within 20 km of the vendors warehouses. The number of customers within 20 to 40 km of the warehouses are also appreciable. Few Non-Churned customers are beyond 40 km from the customer warehouses.
- **Most of the churned customers have transacted with the vendor recently**, between 0 to 10 days from when the data was collected. Most of the Non-Churners have last transacted within one month.
- **Both churned & Non-churned customers have received almost similar amount of hike from last year.**

**HEATMAP( )** - This shows a view of all variables and their relationship with all other variables. As almost all of the variables are with categorical data, there is hardly any correlation or distribution between any variables .

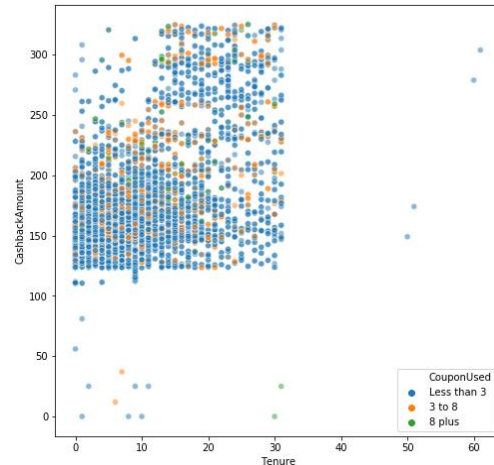


**Fig 11. Correlation of numerical variables using HEATMAP( )**

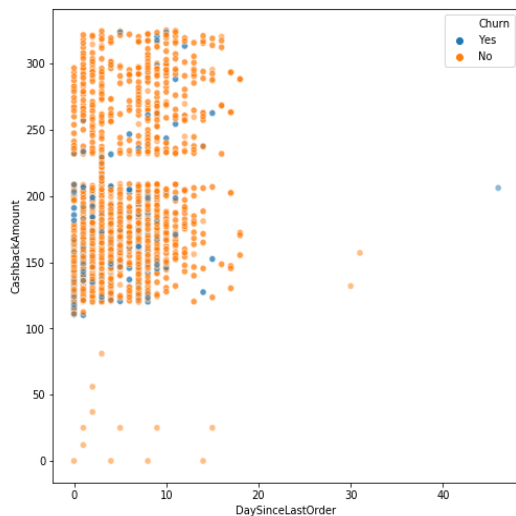
We can see from the above high-level analysis that there is not much evidence of relationship between the numeric variables. The highest degree of correlation occurs between **cashback amount and tenure**, **Days since last order**. There is also some correlation between **DaysSinceLastOrder and Tenure** as well. Let us explore these relationships a more.



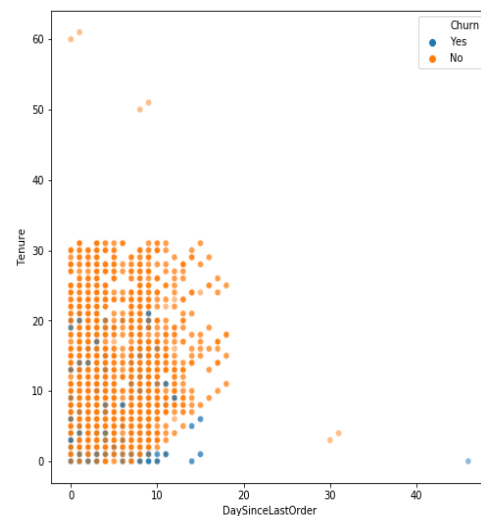
**a. Cashbackamount Vs Tenure Vs Churn**



**b. Cashbackamount Vs Tenure Vs CouponsUsed**



**c. Cashbackamount Vs DaySinceLastOrder Vs Churn**



**d. Tenure Vs DaySinceLastOrder Vs Churn**

**Fig 13. Multivariate analysis of numerical variables.**

- Most of the churned & Non-churned customers receive good cashback amount even in a shorter tenure. We can observe that quite a few customers who churned, have received cashback amount appreciable cashback amount.
- Most of the customers have used less than 3 coupons and also received cashback with a tenure of 0 to 30 days.
- We can see that there are quite a few customers who have churned after ordering recently and even after receiving Cashback.
- There are few short stay customers with shorter tenure, who have ordered recently and churned.

➤ **General Inference from EDA:**

- Customers with Order Counts of 6 to 10, have received huge cashback amount compared to other order count less than 6.
- Most of the customer with 'Other' as Preferred Ordered Category receive higher cashback.
- Most of the customers who churned, have purchased very recently between 0 and 10 days.
- Even customers with longer distance from warehouse to home, have not churned.
- Customers will least tenure predominantly churn. Higher number of orders placed in the last month i.e 6 to 10 orders, are by customers with longer tenure.
- Customers with longer tenure have used more number of coupons in the last month. i.e 8 plus coupons.
- Customers with longer tenure have 'Others' as their most preferred order Category.

### 3. Data Cleaning and Pre-processing:

#### Data Cleaning:

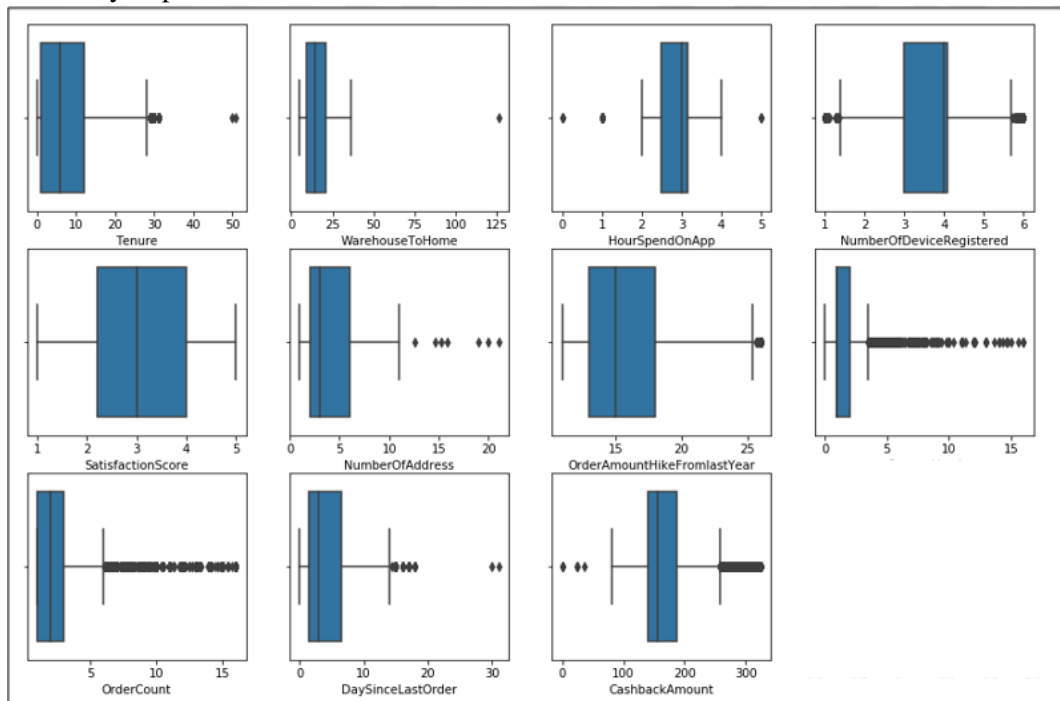
➤ Removal of Unwanted Variable:

**CustomerID** is unique identifier in the given dataset. Hence, it could be removed before going into further analysis.

➤ Checking for Outliers:

Many machine learning models are easily impacted by the outliers in the training data. Models using classification algorithm increase the weights of misclassified points on every iteration and therefore might put high weights on these outliers as they tend to be often misclassified. This can become an issue if that outlier is an error of some type, or if we want our model to generalize well and not care for extreme values. Hence, we check for the outliers in the given data and treat them before model building. Outliers can affect both the result and assumptions.

Boxplot( ) shows that there are outliers in all the numerical variables of the dataset. Hence, before building the model, it is very important to treat the outliers.



**Fig 14. Checking for Outliers using BOXPLOT**

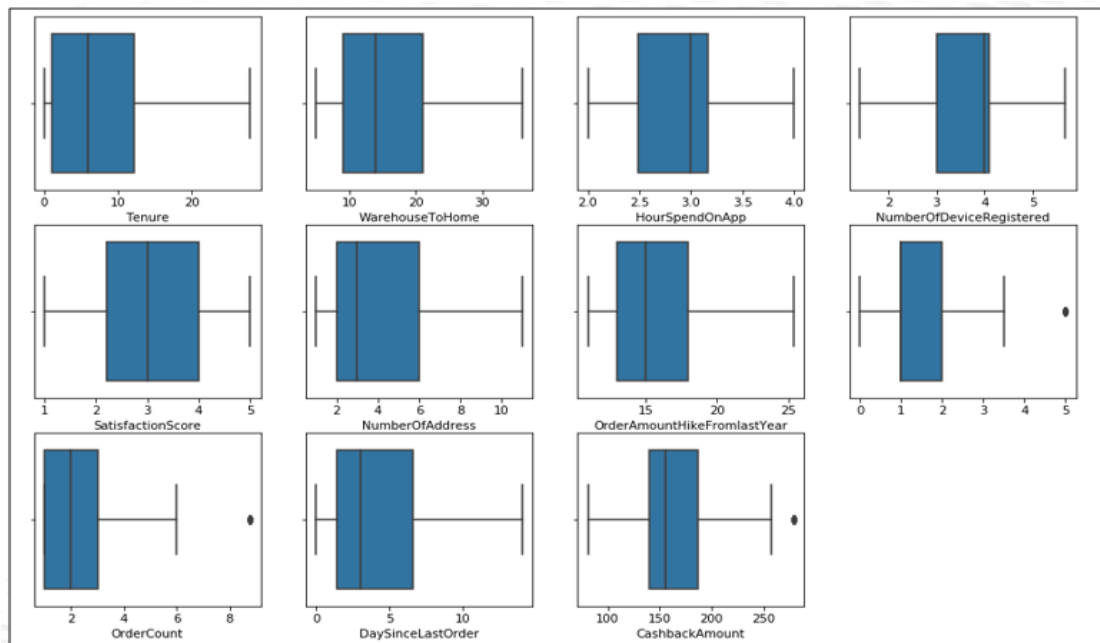
We could see from above visualization that a lot of outliers have come up in the numeric variables, this has happened because of the oversampling technique that we have done. These outliers will have to be treated.

➤ Outlier Treatment using winsorization:

Winsorization is a way to minimize the influence of outliers in your data by either:

- Assigning the outlier a lower weight,
- Changing the value so that it is close to other values in the set.

The data points are modified, but not trimmed/removed.



**Fig 15. Outlier Treatment using winsorization**

Using winsorization, the outliers in the upper range are replaced with 95<sup>th</sup> quantile and outliers in the lower range are replaced with 5<sup>th</sup> quantile.

#### **Data Pre-processing:**

- We can see from that there is a good amount of class imbalance in the data w.r.t the target variable i.e. churn. To take care of this imbalance we will have to apply **SMOTE**.

Churn	Proportion of Class
Yes	83.16
No	16.83

**Fig 16. Proportion of class under Churn**

- Before applying SMOTE we will split the data into training and testing sets to avoid introducing bias in the test data set. Convert the datasets into a sparse matrix before applying SMOTE.
- **SMOTE**  
With SMOTE, we will synthesize new samples from the minority class rather than just duplicating samples. SMOTE works by selecting samples that are close in the feature space, drawing a line between the samples in the feature space and drawing a new sample at a point along that line.
- Hence, the proportion of the minority class i.e. “Churn\_yes” has been increased from approximately 17% to 50% using SMOTE.

#### **4. Model building:**

The classification algorithms such as **Logistic Regression, Linear Discriminant Analysis, Naïve Bayes, K-Nearest Neighbors, Decision Tree, Random Forest and Support Vector Machine** are implemented to build a model to predict the churners and non churners.

#### **Baseline model – Logit:**

The baseline model is built using logit for comparison with other models. Coefficients and Pvalue of each variable is analysed for its impact in the target variable ‘Churn’.

We could see that following variables have Pvalue > 0.5, hence these variables are identified as non significant in predicting the customer churn and further models are built eliminating these variables.

- HourSpendOnApp
- OrderAmountHikeFromlastYear
- CityTier\_Tier-2
- Gender\_Male
- PreferredOrderCat\_Grocery

The baseline model is built again using the significant variables and the coefficients are interpreted as follows:

- As there is an increase in the Tenure of the customer, probability of churn decreases by 45.6%
- If a customer is in a tier-3 city the probability for churning increases by 63%.
- Also for every unit increase in the cashback amount, the probability reduces by 49.7%
- Probability of customer churning increases by 46.5%, if there number of days since last order is very less.
- Customers who prefer the order category 'Others', tend to churn by 86%
- Customers who have raised complain tend to churn by 76.7%

### **Logistic Regression**

Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.). In other words, the logistic regression model predicts  $P(Y=1)$  as a function of  $X$ .

### **Linear Discriminant Analysis**

Linear Discriminant Analysis is a simple and effective method for classification. The model consists of statistical properties of your data, calculated for each class. For multiple variables, this is the mean and the variance of the all the variables for each class is calculated over the multivariate Gaussian, namely the means and the covariance matrix.

These statistical properties are estimated from your data and plug into the LDA equation to make predictions. LDA makes predictions by estimating the probability that a new set of inputs belongs to each class. The class that gets the highest probability is the output class and a prediction is made. The model uses Bayes Theorem to estimate the probabilities.

### **KNN**

Neighbors-based classification is a type of instance-based learning or non-generalizing learning: it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point.

### **Naïve Bayes Model**

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable.

- In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering.
- **They require a small amount of training data to estimate the necessary parameters.**

## Support Vector Classifier (SVC) Model

An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible.

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection

The advantages of support vector machines are:

- It is effective in **high dimensional spaces**.
- They are still effective in cases where **number of dimensions is greater than the number of samples**.
- This model uses a subset of training points in the decision function (called **support vectors**), so it is also memory efficient.
- **Versatile**: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels

## Decision tree Model

Decision trees are used in the data mining with the objective of creating a model that predicts the value of target (dependent variable) from the set of input values (independent variables). CART model constructs decision trees where the target variable is categorical and the tree is used to identify the class ('Yes' or 'No' in this case study) within which a target variable is likely to fall.

The main elements of CART are:

- Rules for splitting data at a node based on the value of one variable
- Stopping rules for deciding when a branch is terminal and can be split no more
- Finally, a prediction for the target variable in each terminal node

## Random Forest Model

The random forest is a classification algorithm consisting of many decisions trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees.

### Improving overall performance:

#### ENSEMBLE MODELLING

Besides the models built with basic classification algorithms, decisions from multiple models could be combined to **improve the overall performance**. This could be achieved using **Ensemble modelling**.

#### ➤ **Bagging:**

Random forest is applied for bagging which also randomly selects subsets of features used in each data sample.

#### ➤ **Boosting:**

AdABOOST, Gradient Boosting and Extreme Gradient Boosting are the Boosting Techniques used to implement the predictive model.

#### HYPER-PARAMETER OPTIMIZATION

**One algorithm isn't the best across all datasets or for all use-cases. To find the best solution, we conduct a lot of experiments, evaluating different machine learning algorithms and tuning their hyper-parameters.**

**Hyper-parameter optimization** is the process of finding the best possible values for these hyper-parameters to optimize your performance metric (e.g. highest accuracy, lowest RMSE, etc.). To do this, we train a model for

different combinations of values and evaluate which find the best solution. **Grid Search could be used to search for best combinations.**

### **Grid Search**

Grid search is the most basic hyperparameter tuning method. Here, build a model for each possible combination of all the hyperparameter values provided, evaluating each model and select the model which produces best result. It is very simple to use. We will find the best combination of the values from the hyper parameters we've provided. In the following segment, we would perform Bagging and Boosting Techniques **with Hyper-parameter Tuning.**

Models built using the classification algorithms such as **K-Nearest Neighbors, Decision Tree, Random Forest and Support Vector Machine** are tuned by **Grid Search.**

**Model Tuning is also performed for all the ensemble models built using Bagging and Boosting Techniques to check the overall performance.**

## **5. Model validation:**

**Performance Metrics - Checking the performance of Predictions on Train and Test sets using Accuracy, Confusion Matrix, Plot ROC curve and get ROC AUC score for each model.**

### **i) Accuracy**

- It is the ratio of number of correct predictions to the total number of input samples  
 **$\text{Accuracy} = \text{No. of correct predictions} / \text{Total no. of predictions made}$**
- It tells us how accurately / cleanly does the model classify the data points.
- Lesser the false predictions , greater the accuracy.

### **ii) Confusion Matrix**

- Confusion Matrix as the name suggests gives us a matrix as output and describes the complete performance of the model.
- It is a 2x2 matrix reflecting the performance of the model built.

	Predicted Negative	Predicted Positive
Actual Negative	True Negative (TN)	False Positive (FP)
Actual Positive	False Negative (FN)	True Positive (TP)

**Fig 17. Confusion Matrix**

- **Accuracy :**  **$(TP + TN) / (TP + TN + FP + FN)$**
- **Sensitivity or Recall or True Positive Rate (TPR):** It describes how many of the actual true data points are identified as true data points by the model.  
 **$\text{Recall} = TP / (TP + FN)$**
- **Specificity :** How many of the actual negative data points are identified as negative data points.  
 **$\text{Specificity} = TN / (TN + FP)$**
- **Precision :** It tells that among the data points identified as positive, how many are really positive.  
 **$\text{Precision} = TP / (FP + TP)$**

### **iii) ROC Curve**

AUC - ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. Higher the AUC, the model predicts the target classes correctly. **The ROC curve is plotted with TPR against the FPR where TPR is on y-axis and FPR is on the x-axis.**

**$\text{True Positive Rate (TPR)} = TP / (TP + FN)$**

**$\text{False Positive Rate (FPR)} = 1 - \text{Specificity} = FP / (FP + TN)$**

TPR and FPR both have values in the range [0,1] and are computed at different thresholds such as (0.00, 0.02, 0.04, ..., 1.00) and a graph is drawn. AUC is the Area Under the Curve of plot FPR vs TPR at different points between [0,1].



**Comparison & Inference of all the models - LG, LDA, KNN, NB, SVC, Decision Tree, Random Forest, Bagging, Boosting (ADA, GB, XGB) with & without Grid Search (GS)**

	Model	Accuracy	Precision	Recall	AUC
0	Logreg	0.835721	0.826952	0.849131	0.835721
1	LDA	0.836788	0.819549	0.863761	0.836788
2	KNN	0.924566	0.874060	0.992076	0.924566
3	NB	0.758001	0.736255	0.804023	0.758001
4	SVC	0.837245	0.825345	0.855532	0.837245
5	Decision-Tree	0.859494	0.856884	0.863151	0.859494
6	Random Forest	0.990247	0.992046	0.988418	0.990247
7	Bagging	0.998476	0.997869	0.999086	0.998476
8	AdaBoost	0.897440	0.902718	0.890887	0.897440
9	Gradient Boosting	0.912069	0.906006	0.919537	0.912069
10	Extreme Gardient Boosting	0.924566	0.927563	0.921061	0.924566
11	Simple Ensemble	0.835721	0.826952	0.849131	0.835721
12	Decision Tree With GS	0.983389	0.979734	0.987199	0.983389
13	Random Forest With GS	0.999390	0.999390	0.999390	0.999390
14	KNN With GS	0.986589	1.000000	0.973179	0.986589
15	SVC With GS	0.985218	0.978653	0.992076	0.985218
16	Bagging With GS	0.913898	0.916054	0.911308	0.913898
17	AdaBoost With GS	0.880829	0.874438	0.889363	0.880829
18	Gradient Boosting With GS	0.999848	0.999695	1.000000	0.999848
19	Extreme Gardient Boosting With GS	0.987199	0.985424	0.989028	0.987199

	Model	Accuracy	Precision	Recall	AUC
0	Logreg	0.793961	0.440711	0.774306	0.786153
1	LDA	0.780935	0.423507	0.788194	0.783819
2	KNN	0.793369	0.446585	0.885417	0.829932
3	NB	0.684429	0.298188	0.628472	0.662202
4	SVC	0.793961	0.441860	0.791667	0.793050
5	Decision-Tree	0.790409	0.429487	0.697917	0.753669
6	Random Forest	0.928952	0.778146	0.815972	0.884075
7	Bagging	0.940793	0.803226	0.864583	0.910522
8	AdaBoost	0.822380	0.486547	0.753472	0.795009
9	Gradient Boosting	0.872114	0.592308	0.802083	0.844296
10	Extreme Gardient Boosting	0.887507	0.633880	0.805556	0.854955
11	Simple Ensemble	0.793961	0.440711	0.774306	0.786153
12	Decision Tree With GS	0.870930	0.588832	0.805556	0.844962
13	Random Forest With GS	0.946714	0.823529	0.875000	0.918228
14	KNN With GS	0.934873	0.796667	0.829861	0.893160
15	SVC With GS	0.876850	0.583333	0.972222	0.914734
16	Bagging With GS	0.864417	0.576227	0.774306	0.828623
17	AdaBoost With GS	0.818236	0.480000	0.791667	0.807682
18	Gradient Boosting With GS	0.968028	0.890000	0.927083	0.951764
19	Extreme Gardient Boosting With GS	0.931912	0.774603	0.847222	0.898272

**Fig 18. Comparison of Training Test set of all the models**

**Interpretation based on Accuracy, Precision, Recall & AUC Score**

**Accuracy** - Accuracy is a ratio of correctly predicted observation to the total observations.

- The train data of Logistic Regression and LDA has accuracy of **83%** whereas test data shows accuracy of **79%**, hence the model performs fairly average.
- The train and test data of KNN has the accuracy of about **92% and 79%** respectively. This shows that the model built is over fitting.
- SVC model has the train and test accuracy of about **83% and 79%** respectively. Hence the model behaves fairly well and not over fitting or under fitting.
- The performance of Naïve Bayes model is comparatively less, as the train and test accuracy of Naïve Bayes model is **75% and 68%**
- Decision Tree model has performs fairly averagely with the train and test accuracy of **85% and 79% respectively**.
- **Random Forest model has the best train and test accuracy of about 99% and 93% respectively with not much over fitting or under fitting.**

**Precision:**

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. This metrics tells us how many of the customers who churn are correctly predicted. That is **among the customers who are predicted to churn, how many actually churned and among the customers who are not predicted to churn, how many actually does not churn.**

**Among the various models built, Random Forest has the most optimal precision score of 78% in predicting the customer 'Churn'.**

### Recall:

Recall is the ratio of correctly predicted positive observations to the all observations in actual class. This shows out of the customer churn predicted, how many did we predict correctly?

**Among the models, Random Forest has the best recall score in correctly predicting customer churn. But the model seems to be over fitting. Hence with respect to other models built, Logistic Regression, LDA and SVC have fairly good recall score and model is not over fitting under fitting.**

### Comparison based on AUC Score:

- The train ROC\_AUC score and test ROC\_AUC score for Logistic Regression model is **0.84 and 0.79** . As the training data and testing data behaves similarly, we conclude that the model built is fairly good and is not over fitted.
- Similarly for LDA model, both the train and test ROC\_AUC score is **0.84 and 0.78**. In this model too, the train and test set behaves similarly and not over fitted or under fitted.
- In KNN model, the train and test ROC\_AUC score is **0.92 and 0.82** respectively. The scores are fairly good and could see that the model is slightly over fitted.
- In NB model, the train and test ROC\_AUC score is almost **0.75 and 0.66**. Hence the scores are average and model is slightly under fitted.
- The train and test ROC\_AUC score for the SVC model is **0.84 and 0.79** respectively. The model behaves fairly well and could see that the model is not over fitted or under fitted.

### Interpretations with Model Tuning

#### Decision Tree, Random Forest, KNN, SVC – Model Tunning

- The performance of the decision tree model is improved with model tuning with respect to Accuracy, Precision & Recall. But the train tuned model train data exceeds the threshold difference (5 % – 10 % of the train value) with resulted test value. The above metrics clearly shows that model built is over fitting, as the train and test data does not behave similarly.
- **Both KNN & SVC models** are tuned to improve the **Accuracy, Precision, Recall and AUC scores. The** resulting tuned scores are improved but model is over fitting in terms of **Accuracy & Recall**.
- **Random Forest model performs well compared to other models in terms of** Accuracy, Recall & AUC score but with low precision. Hence the Random Forest model is tuned to improve the Precision score to **80%. Hence, Random Forest model with model tuning performs well in terms of Accuracy, Precision, Recall and AUC score.**

### Bagging:

- In Bagging, the train ROC\_AUC is **0.99** and test score is **0.91**. The model behaves well and could see that the model is slightly over fitted.
- In Bagging model (without model tuning) - Accuracy, Precision and Recall of train data exceeds the threshold difference (5 % – 10 % of the train value) with resulted test value. The above metrics clearly shows that model built is over fitting, as the train and test data does not behave similarly.
- To solve the issue of over fitting, we tune the Bagging model with hyper parameters. Hence, we have built a best model using Grid Search on Bagging Random Forest model.
- Bagging with model tuning also resulted in a slightly over fitting model as Accuracy, Precision, Recall & AUC score of the train and test data exceeds the threshold difference (5 % – 10 % of the train value) with resulted test value.

## Boosting:

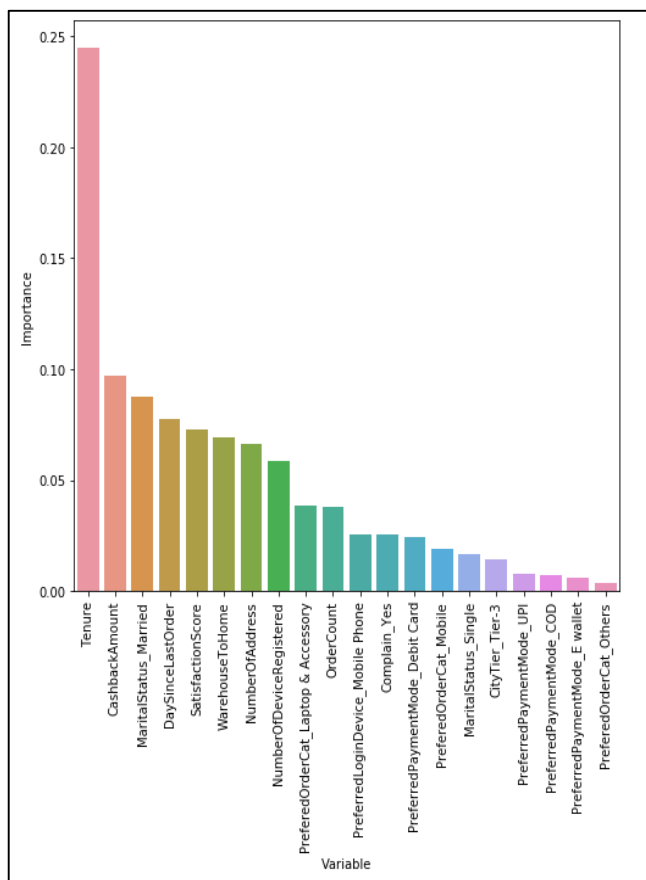
- In Boosting (ADA, GB, XG) models (without model tuning) - Accuracy, Precision and Recall of train and test data exceeds the threshold difference (5 % – 10 % of the train value) with resulted test value. The above metrics clearly shows that model built is **over fitting**, as the train and test data does not behave similarly.
- To solve the issue of overfitting, we tune the Boosting model with hyper parameters.
- AdaBoot model with model tuning has the train ROC\_AUC score **88%** and test ROC\_AUC is score is **80%**. This shows that the model performs fairly well but slightly overfitting.
- **Gradient Boosting with model tuning has the best Accuracy, Precision, Recall & AUC score of about 95% which is the best score of all the model and model behaves similarly for train and test data, which is not over fitting or under fitting.**

## Best/Optimal Models:

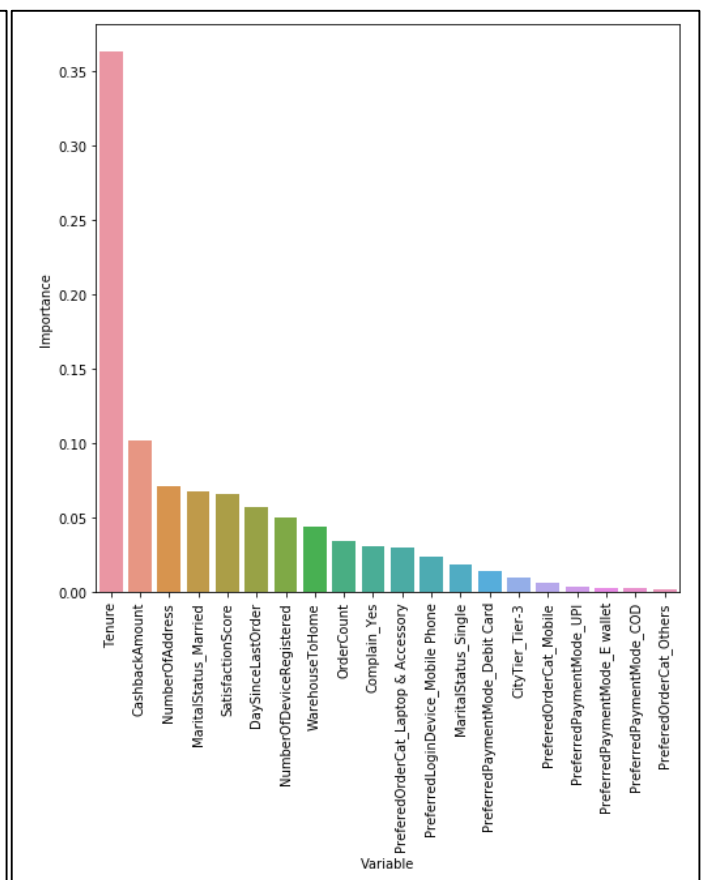
- **Random Forest & Gradient Boosting with model tuning has the best Accuracy, Precision, Recall & AUC score of about 95% which is the best score of all the model and model behaves similarly for train and test data, which is not over fitting or under fitting.**

## 6. Final interpretation / recommendation:

Based on overall performance metrics, it is evident that **Random Forest & Gradient Boosting with model tuning performs comparatively well with other models** to predict customer churn on the basis of the given information. The feature\_importances can give us insight into a problem by telling us what variables are the most discerning between classes.



**Fig 19a. Feature\_importances with Random Forest with Model Tuning**



**Fig 19b. Feature\_importances with Gradient Boosting with Model Tuning**

The above feature\_importances are extracted the Random Forrest Classifier with the Model Tuning. Using, Random Forest it is possible to extract predictive power of each attribute. **We could see “Tenure” and ‘CashbackAmount’ attribute contributes more to the prediction of customer ‘Churn’.** As we have already inferred from the coefficients of each variable, customers with shorter tenure tend and customers who receive huge cashback amount predominantly churn.

### **Characteristics of Churners:**

- ☐ **Male** customers tend to churn more than female customers.
- ☐ **Tier-3** customers are predominant under Churners.
- ☐ Customers who spend between 2 to 4 hours on the App i.e **Medium Usage** customers predominantly Churn.
- ☐ Most of the Churners have **three or more devices registered** on the App or website.
- ☐ Customers tend to churn even after having the **Satisfaction Score of 5**. The ratio of Churners are more in the group of customers with a score of 5, compared to other scores.
- ☐ Customers with **6 to 8 Addresses** registered in the App or website are prone to churn.
- ☐ It is observed that both Churner & Non-Churners have placed **less than 3 orders** in the last month.
- ☐ Customers with **Computers** as their Preferred Login Devices churn more than Mobile users.
- ☐ **Debit Card** is the preferred payment method for both Churners & Non-Churners followed by Credit Card.
- ☐ Customers with **COD & E wallet** as their preferred payment method, have higher probability to churn.
- ☐ Majority of the customers who buys **Mobiles** have the higher probability to churn.
- ☐ Most of the **Married** customers are Non-Churners, whereas Most of the **Single** customers are Churners.
- ☐ **Single** customers are prone to churn compared to Married customers.
- ☐ Customers who raise **complaints** are more likely to churn.

### **Recommendations:**

#### **➤ Tenure & Cashback:**

- ☐ **Subscriptions** could be introduced to customers to increase their tenure.
- ☐ **Since most of the customers churn even after receiving appreciable cashback amount, Cashbacks could be linked to subscribed users. This will increase their tenure in turn reducing churn.**
- ☐ Encourage customers to **order multiple items** in single order by setting slab for cashbacks.
- ☐ As multiple items deliver in different dates, by default tenure of the customer increases.

#### **➤ Tier 3:**

- ☐ Concentrate more on Tier 3 customers, because there where more ratio of people likely to churn.
- ☐ The tenure of Tier 3 customers are the very low, hence increase the tenure to reduce the churn on Tier 3.
- ☐ “Groceries” and “Others” were the most ordered categories in Tier 3. Hence, Target campaigns could be done in those region on their most preferred order category.

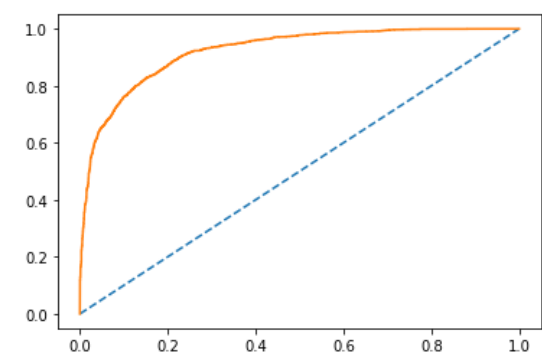
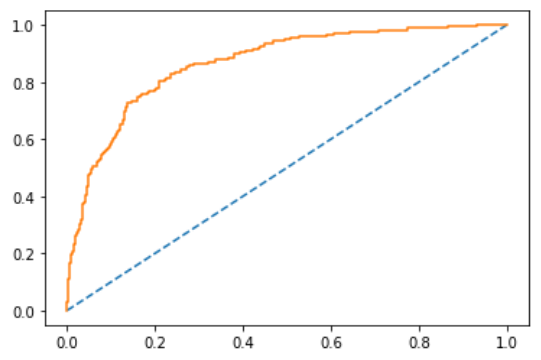
➤ **General Recommendations:**

- ❑ Customers paying through COD & E wallet tend to churn more. But the tenure of these customers are logically high, hence we could take measures to increase these payment methods by bundling with offers and cashbacks.
- ❑ Investigate the type of complaints registered by the customers and focus more on maintaining their trust on the company.
- ❑ Promote specific categories of products based on Gender and Marital Status.
- ❑ As customers who login through Computers tend to churn more, Incentive schemes could be introduced to customers for logging through Mobile App.
- ❑ Customers who have Mobile as their preferred Ordered category tend to churn more as there might be offers on these items only for limited duration.
- ❑ Hence, bundle other items related to Mobile which do not have offer with them. Also, provide coupons for customers who buys only Mobile with shorter tenure to increase their stay.

## APPENDIX

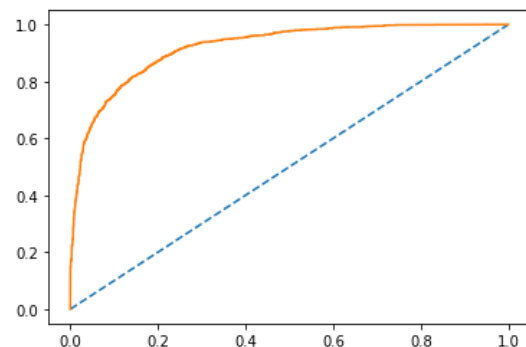
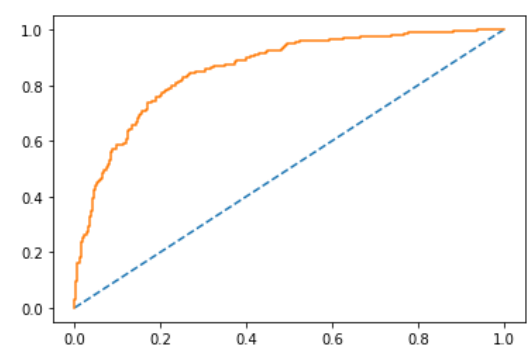
### PERFORMANCE METRICS OF ALL THE MODELS BUILT

#### LOGISTIC REGRESSION MODEL

TRAIN DATA		TEST DATA																																																													
ACCURACY																																																															
<pre>logreg1.score(X_train, Y_train)</pre> 0.835720816824139		<pre>logreg1.score(X_test, Y_test)</pre> 0.7939609236234458																																																													
CONFUSION MATRIX																																																															
<pre>confusion_matrix(Y_train, Y_ptr_lr1)</pre> array([[2698, 583], [ 495, 2786]], dtype=int64)		<pre>confusion_matrix(Y_test, Y_pte_lr1)</pre> array([[1118, 283], [ 65, 223]], dtype=int64)																																																													
CLASSIFICATION REPORT																																																															
<pre>print(classification_report(Y_train, Y_ptr_lr1))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.84</td><td>0.82</td><td>0.83</td><td>3281</td></tr><tr><td>1</td><td>0.83</td><td>0.85</td><td>0.84</td><td>3281</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.84</td><td>6562</td></tr><tr><td>macro avg</td><td>0.84</td><td>0.84</td><td>0.84</td><td>6562</td></tr><tr><td>weighted avg</td><td>0.84</td><td>0.84</td><td>0.84</td><td>6562</td></tr></tbody></table>			precision	recall	f1-score	support	0	0.84	0.82	0.83	3281	1	0.83	0.85	0.84	3281	accuracy			0.84	6562	macro avg	0.84	0.84	0.84	6562	weighted avg	0.84	0.84	0.84	6562	<pre>print(classification_report(Y_test, Y_pte_lr1))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.95</td><td>0.80</td><td>0.87</td><td>1401</td></tr><tr><td>1</td><td>0.44</td><td>0.77</td><td>0.56</td><td>288</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.79</td><td>1689</td></tr><tr><td>macro avg</td><td>0.69</td><td>0.79</td><td>0.71</td><td>1689</td></tr><tr><td>weighted avg</td><td>0.86</td><td>0.79</td><td>0.81</td><td>1689</td></tr></tbody></table>			precision	recall	f1-score	support	0	0.95	0.80	0.87	1401	1	0.44	0.77	0.56	288	accuracy			0.79	1689	macro avg	0.69	0.79	0.71	1689	weighted avg	0.86	0.79	0.81	1689
	precision	recall	f1-score	support																																																											
0	0.84	0.82	0.83	3281																																																											
1	0.83	0.85	0.84	3281																																																											
accuracy			0.84	6562																																																											
macro avg	0.84	0.84	0.84	6562																																																											
weighted avg	0.84	0.84	0.84	6562																																																											
	precision	recall	f1-score	support																																																											
0	0.95	0.80	0.87	1401																																																											
1	0.44	0.77	0.56	288																																																											
accuracy			0.79	1689																																																											
macro avg	0.69	0.79	0.71	1689																																																											
weighted avg	0.86	0.79	0.81	1689																																																											
ROC_CURVE & ROC_AUC SCORE																																																															
<p>AUC: 0.922</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac3a5eab08&gt;]</p> 		<p>AUC: 0.866</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac3a906348&gt;]</p> 																																																													

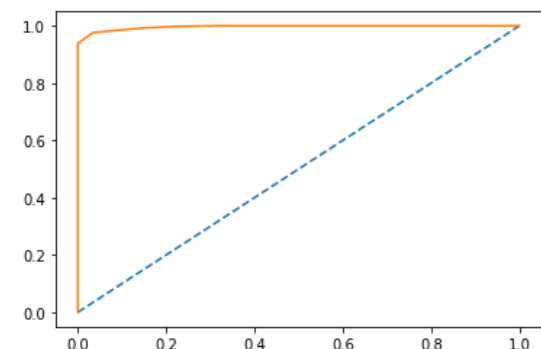
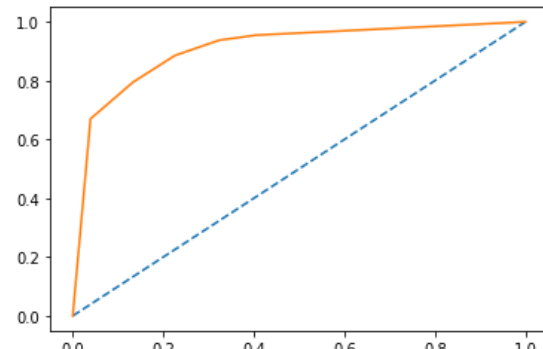
**TABLE 1. PERFORMANCE METRICS : LOGISTIC REGRESSION MODEL**

## LINEAR DISCRIMINANT ANALYSIS MODEL

TRAIN DATA		TEST DATA																																																													
ACCURACY																																																															
<pre>clfLDA.score(X_train, Y_train)</pre> 0.8367875647668394		<pre>clfLDA.score(X_test, Y_test)</pre> 0.7809354647720544																																																													
CONFUSION MATRIX																																																															
<pre>confusion_matrix(Y_train, Y_ptr_ld)</pre> array([[2657, 624], [ 447, 2834]], dtype=int64)		<pre>confusion_matrix(Y_test, Y_pte_ld)</pre> array([[1092, 309], [ 61, 227]], dtype=int64)																																																													
CLASSIFICATION REPORT																																																															
<pre>print(classification_report(Y_train, Y_ptr_ld))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.86</td><td>0.81</td><td>0.83</td><td>3281</td></tr><tr><td>1</td><td>0.82</td><td>0.86</td><td>0.84</td><td>3281</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.84</td><td>6562</td></tr><tr><td>macro avg</td><td>0.84</td><td>0.84</td><td>0.84</td><td>6562</td></tr><tr><td>weighted avg</td><td>0.84</td><td>0.84</td><td>0.84</td><td>6562</td></tr></tbody></table>			precision	recall	f1-score	support	0	0.86	0.81	0.83	3281	1	0.82	0.86	0.84	3281	accuracy			0.84	6562	macro avg	0.84	0.84	0.84	6562	weighted avg	0.84	0.84	0.84	6562	<pre>print(classification_report(Y_test, Y_pte_ld))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.95</td><td>0.78</td><td>0.86</td><td>1401</td></tr><tr><td>1</td><td>0.42</td><td>0.79</td><td>0.55</td><td>288</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.78</td><td>1689</td></tr><tr><td>macro avg</td><td>0.69</td><td>0.78</td><td>0.70</td><td>1689</td></tr><tr><td>weighted avg</td><td>0.86</td><td>0.78</td><td>0.80</td><td>1689</td></tr></tbody></table>			precision	recall	f1-score	support	0	0.95	0.78	0.86	1401	1	0.42	0.79	0.55	288	accuracy			0.78	1689	macro avg	0.69	0.78	0.70	1689	weighted avg	0.86	0.78	0.80	1689
	precision	recall	f1-score	support																																																											
0	0.86	0.81	0.83	3281																																																											
1	0.82	0.86	0.84	3281																																																											
accuracy			0.84	6562																																																											
macro avg	0.84	0.84	0.84	6562																																																											
weighted avg	0.84	0.84	0.84	6562																																																											
	precision	recall	f1-score	support																																																											
0	0.95	0.78	0.86	1401																																																											
1	0.42	0.79	0.55	288																																																											
accuracy			0.78	1689																																																											
macro avg	0.69	0.78	0.70	1689																																																											
weighted avg	0.86	0.78	0.80	1689																																																											
ROC_CURVE & ROC_AUC SCORE																																																															
<p>AUC: 0.921</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac429fc408&gt;]</p> 		<p>AUC: 0.858</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac42a52d48&gt;]</p> 																																																													

**TABLE 2. PERFORMANCE METRICS : LINEAR DISCRIMINANT ANALYSIS MODEL**

## KNN MODEL

TRAIN DATA		TEST DATA																																																													
ACCURACY																																																															
<pre>clfKNN.score(X_train, Y_train)</pre> 0.9245656811947577		<pre>clfKNN.score(X_test, Y_test)</pre> 0.7933688573120189																																																													
CONFUSION MATRIX																																																															
<pre>confusion_matrix(Y_train, Y_ptr_knn)</pre> array([[2812, 469], [ 26, 3255]], dtype=int64)		<pre>confusion_matrix(Y_test, Y_pte_knn)</pre> array([[1085, 316], [ 33, 255]], dtype=int64)																																																													
CLASSIFICATION REPORT																																																															
<pre>print(classification_report(Y_train, Y_ptr_knn))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.99</td><td>0.86</td><td>0.92</td><td>3281</td></tr><tr><td>1</td><td>0.87</td><td>0.99</td><td>0.93</td><td>3281</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.92</td><td>6562</td></tr><tr><td>macro avg</td><td>0.93</td><td>0.92</td><td>0.92</td><td>6562</td></tr><tr><td>weighted avg</td><td>0.93</td><td>0.92</td><td>0.92</td><td>6562</td></tr></tbody></table>			precision	recall	f1-score	support	0	0.99	0.86	0.92	3281	1	0.87	0.99	0.93	3281	accuracy			0.92	6562	macro avg	0.93	0.92	0.92	6562	weighted avg	0.93	0.92	0.92	6562	<pre>print(classification_report(Y_test, Y_pte_knn))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.97</td><td>0.77</td><td>0.86</td><td>1401</td></tr><tr><td>1</td><td>0.45</td><td>0.89</td><td>0.59</td><td>288</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.79</td><td>1689</td></tr><tr><td>macro avg</td><td>0.71</td><td>0.83</td><td>0.73</td><td>1689</td></tr><tr><td>weighted avg</td><td>0.88</td><td>0.79</td><td>0.82</td><td>1689</td></tr></tbody></table>			precision	recall	f1-score	support	0	0.97	0.77	0.86	1401	1	0.45	0.89	0.59	288	accuracy			0.79	1689	macro avg	0.71	0.83	0.73	1689	weighted avg	0.88	0.79	0.82	1689
	precision	recall	f1-score	support																																																											
0	0.99	0.86	0.92	3281																																																											
1	0.87	0.99	0.93	3281																																																											
accuracy			0.92	6562																																																											
macro avg	0.93	0.92	0.92	6562																																																											
weighted avg	0.93	0.92	0.92	6562																																																											
	precision	recall	f1-score	support																																																											
0	0.97	0.77	0.86	1401																																																											
1	0.45	0.89	0.59	288																																																											
accuracy			0.79	1689																																																											
macro avg	0.71	0.83	0.73	1689																																																											
weighted avg	0.88	0.79	0.82	1689																																																											
ROC_CURVE & ROC_AUC SCORE																																																															
<p>AUC: 0.996</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac42ac0448&gt;]</p> 		<p>AUC: 0.908</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac42f5e808&gt;]</p> 																																																													

**TABLE 3. PERFORMANCE METRICS : KNN**



## NAÏVE BAYES MODEL

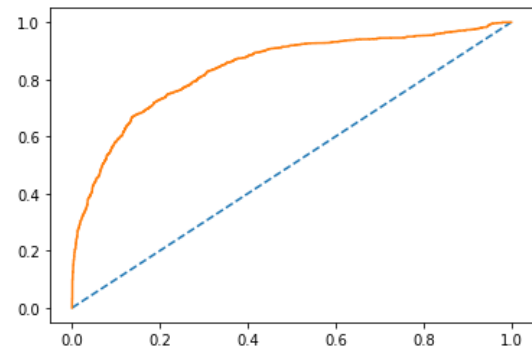
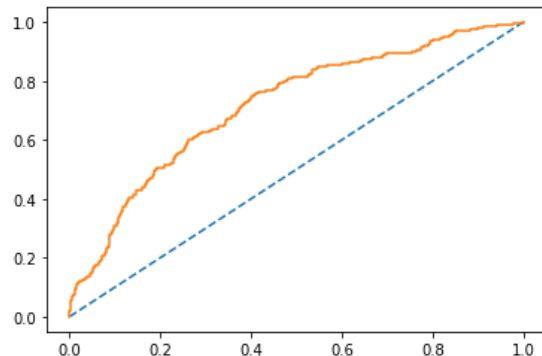
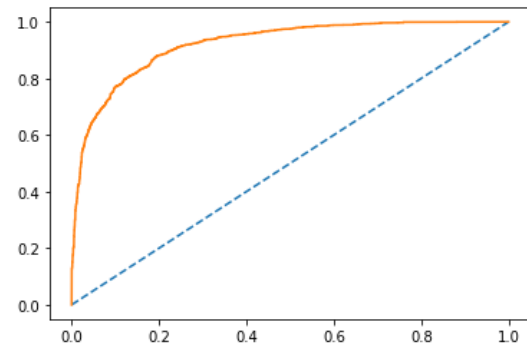
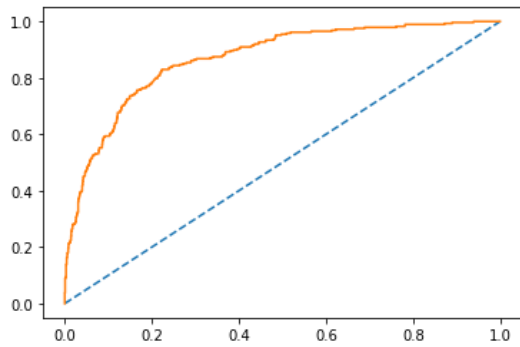
TRAIN DATA		TEST DATA																																																													
ACCURACY																																																															
<pre>modelNB.score(X_train, Y_train)</pre> 0.758000609570253		<pre>modelNB.score(X_test, Y_test)</pre> 0.6844286560094731																																																													
CONFUSION MATRIX																																																															
<pre>confusion_matrix(Y_train, Y_ptr_nb)</pre> array([[2336, 945], [ 643, 2638]], dtype=int64)		<pre>confusion_matrix(Y_test, Y_pte_nb)</pre> array([[975, 426], [107, 181]], dtype=int64)																																																													
CLASSIFICATION REPORT																																																															
<pre>print(classification_report(Y_train, Y_ptr_nb))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.78</td><td>0.71</td><td>0.75</td><td>3281</td></tr><tr><td>1</td><td>0.74</td><td>0.80</td><td>0.77</td><td>3281</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.76</td><td>6562</td></tr><tr><td>macro avg</td><td>0.76</td><td>0.76</td><td>0.76</td><td>6562</td></tr><tr><td>weighted avg</td><td>0.76</td><td>0.76</td><td>0.76</td><td>6562</td></tr></tbody></table>			precision	recall	f1-score	support	0	0.78	0.71	0.75	3281	1	0.74	0.80	0.77	3281	accuracy			0.76	6562	macro avg	0.76	0.76	0.76	6562	weighted avg	0.76	0.76	0.76	6562	<pre>print(classification_report(Y_test, Y_pte_nb))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.90</td><td>0.70</td><td>0.79</td><td>1401</td></tr><tr><td>1</td><td>0.30</td><td>0.63</td><td>0.40</td><td>288</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.68</td><td>1689</td></tr><tr><td>macro avg</td><td>0.60</td><td>0.66</td><td>0.59</td><td>1689</td></tr><tr><td>weighted avg</td><td>0.80</td><td>0.68</td><td>0.72</td><td>1689</td></tr></tbody></table>			precision	recall	f1-score	support	0	0.90	0.70	0.79	1401	1	0.30	0.63	0.40	288	accuracy			0.68	1689	macro avg	0.60	0.66	0.59	1689	weighted avg	0.80	0.68	0.72	1689
	precision	recall	f1-score	support																																																											
0	0.78	0.71	0.75	3281																																																											
1	0.74	0.80	0.77	3281																																																											
accuracy			0.76	6562																																																											
macro avg	0.76	0.76	0.76	6562																																																											
weighted avg	0.76	0.76	0.76	6562																																																											
	precision	recall	f1-score	support																																																											
0	0.90	0.70	0.79	1401																																																											
1	0.30	0.63	0.40	288																																																											
accuracy			0.68	1689																																																											
macro avg	0.60	0.66	0.59	1689																																																											
weighted avg	0.80	0.68	0.72	1689																																																											
ROC_CURVE & ROC_AUC SCORE																																																															
<p>AUC: 0.837</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac42fca2c8&gt;]</p> 		<p>AUC: 0.717</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac43049f08&gt;]</p> 																																																													

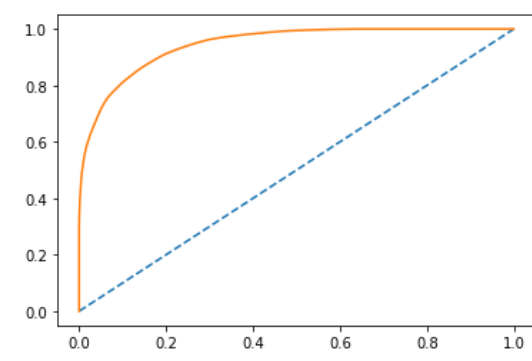
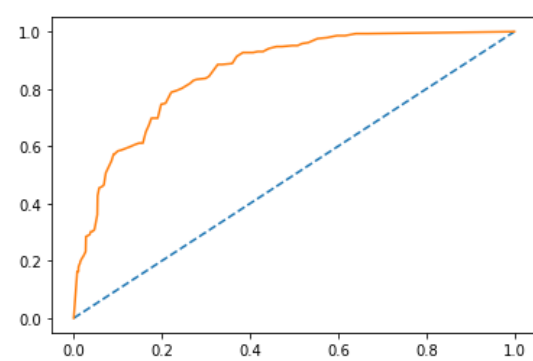
TABLE 4. PERFORMANCE METRICS : NAÏVE BAYES MODEL

## SVC MODEL

TRAIN DATA	TEST DATA																																																												
ACCURACY																																																													
<pre>clfSVM.score(X_train, Y_train)</pre> 0.8372447424565681	<pre>clfSVM.score(X_test, Y_test)</pre> 0.7939609236234458																																																												
CONFUSION MATRIX																																																													
<pre>confusion_matrix(Y_train, Y_ptr_svc)</pre> array([[2687, 594], [ 474, 2807]], dtype=int64)	<pre>confusion_matrix(Y_test, Y_pte_svc)</pre> array([[1113, 288], [ 60, 228]], dtype=int64)																																																												
CLASSIFICATION REPORT																																																													
<pre>print(classification_report(Y_train, Y_ptr_svc))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.85</td><td>0.82</td><td>0.83</td><td>3281</td></tr><tr><td>1</td><td>0.83</td><td>0.86</td><td>0.84</td><td>3281</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.84</td><td>6562</td></tr><tr><td>macro avg</td><td>0.84</td><td>0.84</td><td>0.84</td><td>6562</td></tr><tr><td>weighted avg</td><td>0.84</td><td>0.84</td><td>0.84</td><td>6562</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.85	0.82	0.83	3281	1	0.83	0.86	0.84	3281	accuracy			0.84	6562	macro avg	0.84	0.84	0.84	6562	weighted avg	0.84	0.84	0.84	6562	<pre>print(classification_report(Y_test, Y_pte_svc))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.95</td><td>0.79</td><td>0.86</td><td>1401</td></tr><tr><td>1</td><td>0.44</td><td>0.79</td><td>0.57</td><td>288</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.79</td><td>1689</td></tr><tr><td>macro avg</td><td>0.70</td><td>0.79</td><td>0.72</td><td>1689</td></tr><tr><td>weighted avg</td><td>0.86</td><td>0.79</td><td>0.81</td><td>1689</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.95	0.79	0.86	1401	1	0.44	0.79	0.57	288	accuracy			0.79	1689	macro avg	0.70	0.79	0.72	1689	weighted avg	0.86	0.79	0.81	1689
	precision	recall	f1-score	support																																																									
0	0.85	0.82	0.83	3281																																																									
1	0.83	0.86	0.84	3281																																																									
accuracy			0.84	6562																																																									
macro avg	0.84	0.84	0.84	6562																																																									
weighted avg	0.84	0.84	0.84	6562																																																									
	precision	recall	f1-score	support																																																									
0	0.95	0.79	0.86	1401																																																									
1	0.44	0.79	0.57	288																																																									
accuracy			0.79	1689																																																									
macro avg	0.70	0.79	0.72	1689																																																									
weighted avg	0.86	0.79	0.81	1689																																																									
ROC_CURVE & ROC_AUC SCORE																																																													
AUC: 0.921  [<matplotlib.lines.Line2D at 0x1ac430a78c8>]  	AUC: 0.868  [<matplotlib.lines.Line2D at 0x1ac4310f3c8>]  																																																												

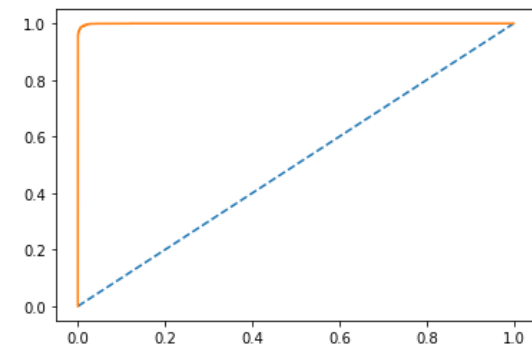
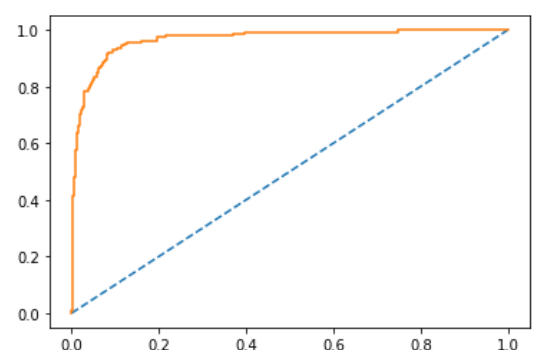
**TABLE 5. PERFORMANCE METRICS : SVC MODEL**

## DECISION TREE

TRAIN DATA	TEST DATA																																																												
ACCURACY																																																													
<pre>DT_model.score(X_train, Y_train)</pre> 0.8594940566900335	<pre>DT_model.score(X_test, Y_test)</pre> 0.7904085257548845																																																												
CONFUSION MATRIX																																																													
<pre>confusion_matrix(Y_train, Y_ptr_dt)</pre> array([[2808, 473], [ 449, 2832]], dtype=int64)	<pre>confusion_matrix(Y_test, Y_pte_dt)</pre> array([[1134, 267], [ 87, 201]], dtype=int64)																																																												
CLASSIFICATION REPORT																																																													
<pre>print(classification_report(Y_train, Y_ptr_dt))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.86</td><td>0.86</td><td>0.86</td><td>3281</td></tr><tr><td>1</td><td>0.86</td><td>0.86</td><td>0.86</td><td>3281</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.86</td><td>6562</td></tr><tr><td>macro avg</td><td>0.86</td><td>0.86</td><td>0.86</td><td>6562</td></tr><tr><td>weighted avg</td><td>0.86</td><td>0.86</td><td>0.86</td><td>6562</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.86	0.86	0.86	3281	1	0.86	0.86	0.86	3281	accuracy			0.86	6562	macro avg	0.86	0.86	0.86	6562	weighted avg	0.86	0.86	0.86	6562	<pre>print(metrics.classification_report(Y_test, Y_pte_dt))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.93</td><td>0.81</td><td>0.86</td><td>1401</td></tr><tr><td>1</td><td>0.43</td><td>0.70</td><td>0.53</td><td>288</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.79</td><td>1689</td></tr><tr><td>macro avg</td><td>0.68</td><td>0.75</td><td>0.70</td><td>1689</td></tr><tr><td>weighted avg</td><td>0.84</td><td>0.79</td><td>0.81</td><td>1689</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.93	0.81	0.86	1401	1	0.43	0.70	0.53	288	accuracy			0.79	1689	macro avg	0.68	0.75	0.70	1689	weighted avg	0.84	0.79	0.81	1689
	precision	recall	f1-score	support																																																									
0	0.86	0.86	0.86	3281																																																									
1	0.86	0.86	0.86	3281																																																									
accuracy			0.86	6562																																																									
macro avg	0.86	0.86	0.86	6562																																																									
weighted avg	0.86	0.86	0.86	6562																																																									
	precision	recall	f1-score	support																																																									
0	0.93	0.81	0.86	1401																																																									
1	0.43	0.70	0.53	288																																																									
accuracy			0.79	1689																																																									
macro avg	0.68	0.75	0.70	1689																																																									
weighted avg	0.84	0.79	0.81	1689																																																									
ROC_CURVE & ROC_AUC SCORE																																																													
AUC: 0.945  [<matplotlib.lines.Line2D at 0x1ac42fb8288>]  	AUC: 0.859  [<matplotlib.lines.Line2D at 0x1ac431d6108>]  																																																												

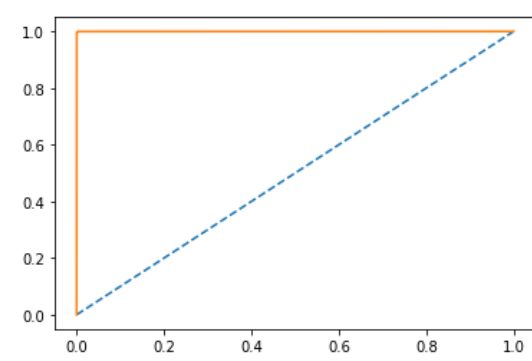
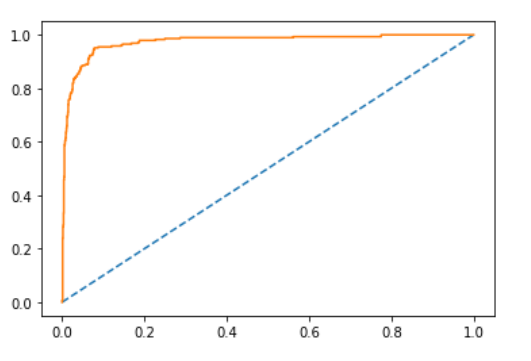
**TABLE 6. PERFORMANCE METRICS : DECISION TREE MODEL**

## RANDOM FOREST

TRAIN DATA	TEST DATA																																																												
ACCURACY																																																													
<pre>RF_model.score(X_train, Y_train)</pre> 0.9902468759524535	<pre>RF_model.score(X_test, Y_test)</pre> 0.9289520426287744																																																												
CONFUSION MATRIX																																																													
<pre>confusion_matrix(Y_train, Y_ptr_rf)</pre> array([[3255, 26], [ 38, 3243]], dtype=int64)	<pre>confusion_matrix(Y_test, Y_pte_rf)</pre> array([[1334, 67], [ 53, 235]], dtype=int64)																																																												
CLASSIFICATION REPORT																																																													
<pre>print(metrics.classification_report(Y_train, Y_ptr_rf))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.99</td><td>0.99</td><td>0.99</td><td>3281</td></tr><tr><td>1</td><td>0.99</td><td>0.99</td><td>0.99</td><td>3281</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.99</td><td>6562</td></tr><tr><td>macro avg</td><td>0.99</td><td>0.99</td><td>0.99</td><td>6562</td></tr><tr><td>weighted avg</td><td>0.99</td><td>0.99</td><td>0.99</td><td>6562</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.99	0.99	0.99	3281	1	0.99	0.99	0.99	3281	accuracy			0.99	6562	macro avg	0.99	0.99	0.99	6562	weighted avg	0.99	0.99	0.99	6562	<pre>print(metrics.classification_report(Y_test, Y_pte_rf))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.96</td><td>0.95</td><td>0.96</td><td>1401</td></tr><tr><td>1</td><td>0.78</td><td>0.82</td><td>0.80</td><td>288</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.93</td><td>1689</td></tr><tr><td>macro avg</td><td>0.87</td><td>0.88</td><td>0.88</td><td>1689</td></tr><tr><td>weighted avg</td><td>0.93</td><td>0.93</td><td>0.93</td><td>1689</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.96	0.95	0.96	1401	1	0.78	0.82	0.80	288	accuracy			0.93	1689	macro avg	0.87	0.88	0.88	1689	weighted avg	0.93	0.93	0.93	1689
	precision	recall	f1-score	support																																																									
0	0.99	0.99	0.99	3281																																																									
1	0.99	0.99	0.99	3281																																																									
accuracy			0.99	6562																																																									
macro avg	0.99	0.99	0.99	6562																																																									
weighted avg	0.99	0.99	0.99	6562																																																									
	precision	recall	f1-score	support																																																									
0	0.96	0.95	0.96	1401																																																									
1	0.78	0.82	0.80	288																																																									
accuracy			0.93	1689																																																									
macro avg	0.87	0.88	0.88	1689																																																									
weighted avg	0.93	0.93	0.93	1689																																																									
ROC_CURVE & ROC_AUC SCORE																																																													
<p>AUC: 1.000</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac4323a488&gt;]</p> 	<p>AUC: 0.966</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac432b7308&gt;]</p> 																																																												

**TABLE 7. PERFORMANCE METRICS : RANDOM FOREST**

### BAGGING MODEL (WITHOUT MODEL TUNNING)

TRAIN DATA		TEST DATA																																																													
ACCURACY																																																															
<div>Bagging_model.score(X_train, Y_train)</div> <div>0.9984760743675709</div>		<div>Bagging_model.score(X_test, Y_test)</div> <div>0.940793368857312</div>																																																													
CONFUSION MATRIX																																																															
<div>confusion_matrix(Y_train, Y_ptr_bag)</div> <div>array([[3274, 7], [ 3, 3278]], dtype=int64)</div>		<div>confusion_matrix(Y_test, Y_pte_bag)</div> <div>array([[1340, 61], [ 39, 249]], dtype=int64)</div>																																																													
CLASSIFICATION REPORT																																																															
<div>print(metrics.classification_report(Y_train, Y_ptr_bag))</div> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>1.00</td><td>1.00</td><td>1.00</td><td>3281</td></tr><tr><td>1</td><td>1.00</td><td>1.00</td><td>1.00</td><td>3281</td></tr><tr><td>accuracy</td><td></td><td></td><td>1.00</td><td>6562</td></tr><tr><td>macro avg</td><td>1.00</td><td>1.00</td><td>1.00</td><td>6562</td></tr><tr><td>weighted avg</td><td>1.00</td><td>1.00</td><td>1.00</td><td>6562</td></tr></tbody></table>			precision	recall	f1-score	support	0	1.00	1.00	1.00	3281	1	1.00	1.00	1.00	3281	accuracy			1.00	6562	macro avg	1.00	1.00	1.00	6562	weighted avg	1.00	1.00	1.00	6562	<div>print(metrics.classification_report(Y_test, Y_pte_bag))</div> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.97</td><td>0.96</td><td>0.96</td><td>1401</td></tr><tr><td>1</td><td>0.80</td><td>0.86</td><td>0.83</td><td>288</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.94</td><td>1689</td></tr><tr><td>macro avg</td><td>0.89</td><td>0.91</td><td>0.90</td><td>1689</td></tr><tr><td>weighted avg</td><td>0.94</td><td>0.94</td><td>0.94</td><td>1689</td></tr></tbody></table>			precision	recall	f1-score	support	0	0.97	0.96	0.96	1401	1	0.80	0.86	0.83	288	accuracy			0.94	1689	macro avg	0.89	0.91	0.90	1689	weighted avg	0.94	0.94	0.94	1689
	precision	recall	f1-score	support																																																											
0	1.00	1.00	1.00	3281																																																											
1	1.00	1.00	1.00	3281																																																											
accuracy			1.00	6562																																																											
macro avg	1.00	1.00	1.00	6562																																																											
weighted avg	1.00	1.00	1.00	6562																																																											
	precision	recall	f1-score	support																																																											
0	0.97	0.96	0.96	1401																																																											
1	0.80	0.86	0.83	288																																																											
accuracy			0.94	1689																																																											
macro avg	0.89	0.91	0.90	1689																																																											
weighted avg	0.94	0.94	0.94	1689																																																											
ROC_CURVE & ROC_AUC SCORE																																																															
<div>AUC: 1.000</div> <div>[&lt;matplotlib.lines.Line2D at 0x1ac442ed448&gt;]</div> 		<div>AUC: 0.973</div> <div>[&lt;matplotlib.lines.Line2D at 0x1ac443bfc48&gt;]</div> 																																																													

**TABLE 8. PERFORMANCE METRICS : BAGGING MODEL (WITHOUT MODEL TUNNING)**

### ADABOOST MODEL (WITHOUT MODEL TUNNING)

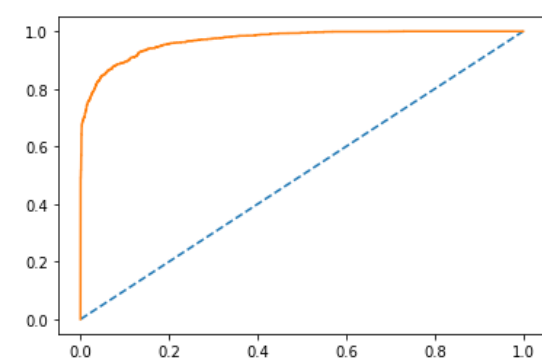
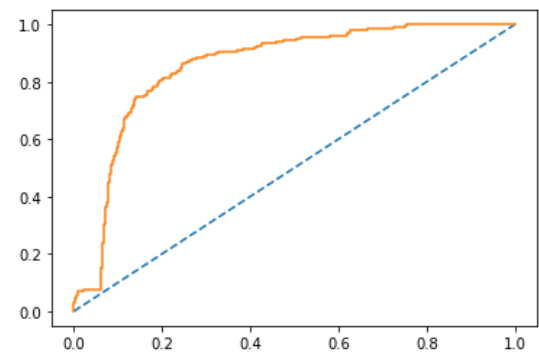
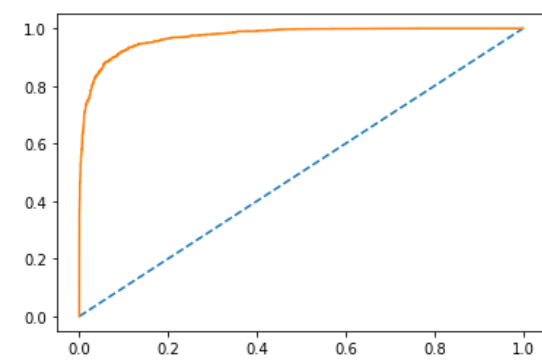
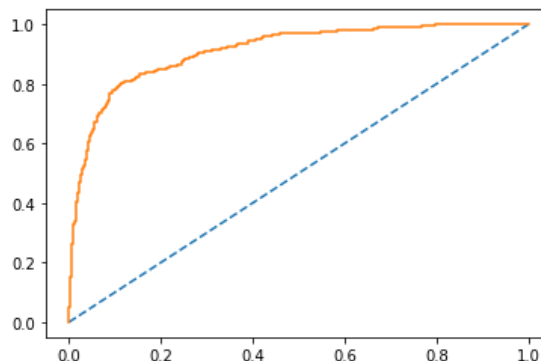
TRAIN DATA	TEST DATA																																																												
ACCURACY																																																													
<pre>ADB_model.score(X_train, Y_train)</pre> 0.8974398049375191	<pre>ADB_model.score(X_test, Y_test)</pre> 0.822380106571936																																																												
CONFUSION MATRIX																																																													
<pre>confusion_matrix(Y_train, Y_ptr_ADA)</pre> array([[2966, 315], [ 358, 2923]], dtype=int64)	<pre>confusion_matrix(Y_test, Y_pte_ADA)</pre> array([[1172, 229], [ 71, 217]], dtype=int64)																																																												
CLASSIFICATION REPORT																																																													
<pre>print(metrics.classification_report(Y_train, Y_ptr_ADA))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.89</td><td>0.90</td><td>0.90</td><td>3281</td></tr><tr><td>1</td><td>0.90</td><td>0.89</td><td>0.90</td><td>3281</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.90</td><td>6562</td></tr><tr><td>macro avg</td><td>0.90</td><td>0.90</td><td>0.90</td><td>6562</td></tr><tr><td>weighted avg</td><td>0.90</td><td>0.90</td><td>0.90</td><td>6562</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.89	0.90	0.90	3281	1	0.90	0.89	0.90	3281	accuracy			0.90	6562	macro avg	0.90	0.90	0.90	6562	weighted avg	0.90	0.90	0.90	6562	<pre>print(metrics.classification_report(Y_test, Y_pte_ADA))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.94</td><td>0.84</td><td>0.89</td><td>1401</td></tr><tr><td>1</td><td>0.49</td><td>0.75</td><td>0.59</td><td>288</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.82</td><td>1689</td></tr><tr><td>macro avg</td><td>0.71</td><td>0.80</td><td>0.74</td><td>1689</td></tr><tr><td>weighted avg</td><td>0.87</td><td>0.82</td><td>0.84</td><td>1689</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.94	0.84	0.89	1401	1	0.49	0.75	0.59	288	accuracy			0.82	1689	macro avg	0.71	0.80	0.74	1689	weighted avg	0.87	0.82	0.84	1689
	precision	recall	f1-score	support																																																									
0	0.89	0.90	0.90	3281																																																									
1	0.90	0.89	0.90	3281																																																									
accuracy			0.90	6562																																																									
macro avg	0.90	0.90	0.90	6562																																																									
weighted avg	0.90	0.90	0.90	6562																																																									
	precision	recall	f1-score	support																																																									
0	0.94	0.84	0.89	1401																																																									
1	0.49	0.75	0.59	288																																																									
accuracy			0.82	1689																																																									
macro avg	0.71	0.80	0.74	1689																																																									
weighted avg	0.87	0.82	0.84	1689																																																									
ROC_CURVE & ROC_AUC SCORE																																																													
AUC: 0.969 [<matplotlib.lines.Line2D at 0x1ac44413cc8>] 	AUC: 0.856 [<matplotlib.lines.Line2D at 0x1ac444a0bc8>] 																																																												

TABLE 9. PERFORMANCE METRICS : ADABOOST MODEL (WITHOUT MODEL TUNNING)

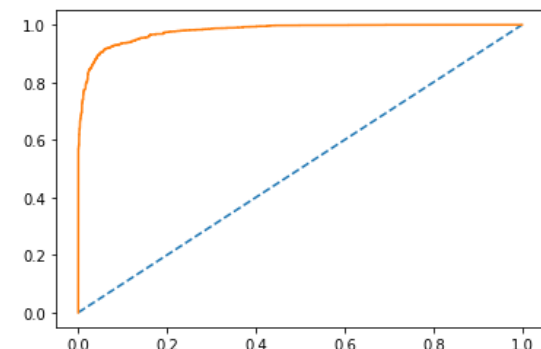
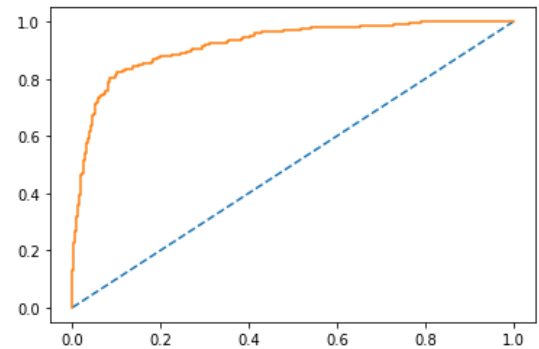
## GRADIENT BOOSTING MODEL (WITHOUT MODEL TUNNING)

TRAIN DATA		TEST DATA																																																													
ACCURACY																																																															
<pre>GB_model.score(X_train, Y_train)</pre> 0.9120694910088387		<pre>GB_model.score(X_test, Y_test)</pre> 0.872113676731794																																																													
CONFUSION MATRIX																																																															
<pre>confusion_matrix(Y_train, Y_ptr_GB)</pre> array([[2968, 313], [ 264, 3017]], dtype=int64)		<pre>confusion_matrix(Y_test, Y_pte_GB)</pre> array([[1242, 159], [ 57, 231]], dtype=int64)																																																													
CLASSIFICATION REPORT																																																															
<pre>print(metrics.classification_report(Y_train, Y_ptr_GB))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.92</td><td>0.90</td><td>0.91</td><td>3281</td></tr><tr><td>1</td><td>0.91</td><td>0.92</td><td>0.91</td><td>3281</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.91</td><td>6562</td></tr><tr><td>macro avg</td><td>0.91</td><td>0.91</td><td>0.91</td><td>6562</td></tr><tr><td>weighted avg</td><td>0.91</td><td>0.91</td><td>0.91</td><td>6562</td></tr></tbody></table>			precision	recall	f1-score	support	0	0.92	0.90	0.91	3281	1	0.91	0.92	0.91	3281	accuracy			0.91	6562	macro avg	0.91	0.91	0.91	6562	weighted avg	0.91	0.91	0.91	6562	<pre>print(metrics.classification_report(Y_test, Y_pte_GB))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.96</td><td>0.89</td><td>0.92</td><td>1401</td></tr><tr><td>1</td><td>0.59</td><td>0.80</td><td>0.68</td><td>288</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.87</td><td>1689</td></tr><tr><td>macro avg</td><td>0.77</td><td>0.84</td><td>0.80</td><td>1689</td></tr><tr><td>weighted avg</td><td>0.89</td><td>0.87</td><td>0.88</td><td>1689</td></tr></tbody></table>			precision	recall	f1-score	support	0	0.96	0.89	0.92	1401	1	0.59	0.80	0.68	288	accuracy			0.87	1689	macro avg	0.77	0.84	0.80	1689	weighted avg	0.89	0.87	0.88	1689
	precision	recall	f1-score	support																																																											
0	0.92	0.90	0.91	3281																																																											
1	0.91	0.92	0.91	3281																																																											
accuracy			0.91	6562																																																											
macro avg	0.91	0.91	0.91	6562																																																											
weighted avg	0.91	0.91	0.91	6562																																																											
	precision	recall	f1-score	support																																																											
0	0.96	0.89	0.92	1401																																																											
1	0.59	0.80	0.68	288																																																											
accuracy			0.87	1689																																																											
macro avg	0.77	0.84	0.80	1689																																																											
weighted avg	0.89	0.87	0.88	1689																																																											
ROC_CURVE & ROC_AUC SCORE																																																															
<p>AUC: 0.972</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac444f49c8&gt;]</p> 		<p>AUC: 0.912</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac6b49ea48&gt;]</p> 																																																													

**TABLE 10. PERFORMANCE METRICS : GRADIENT BOOSTING (WITHOUT MODEL TUNNING)**

## EXTREME GRADIENT BOOSTING MODEL

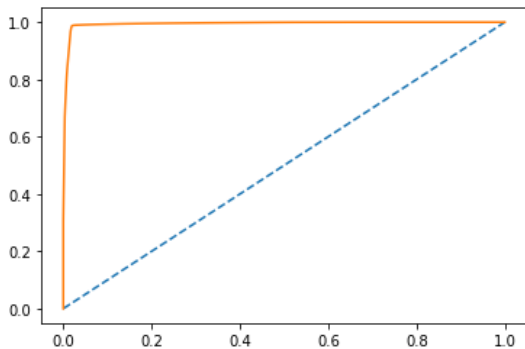
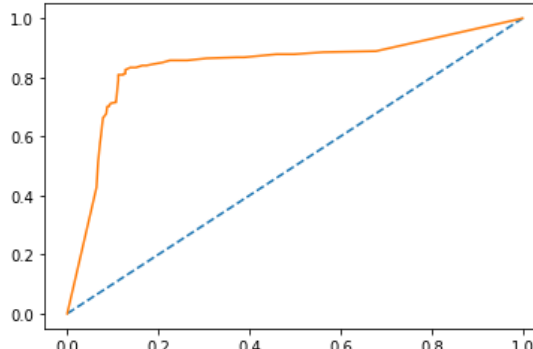
(WITHOUT MODEL TUNNING)

TRAIN DATA		TEST DATA																																																													
ACCURACY																																																															
<pre>XGB_model.score(X_train, Y_train)</pre> 0.9245656811947577		<pre>XGB_model.score(X_test, Y_test)</pre> 0.8875074008288928																																																													
CONFUSION MATRIX																																																															
<pre>confusion_matrix(Y_train, Y_ptr_XGB)</pre> array([[3045, 236], [ 259, 3022]], dtype=int64)		<pre>confusion_matrix(Y_test, Y_pte_XGB)</pre> array([[1267, 134], [ 56, 232]], dtype=int64)																																																													
CLASSIFICATION REPORT																																																															
<pre>print(metrics.classification_report(Y_train, Y_ptr_XGB))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.92</td><td>0.93</td><td>0.92</td><td>3281</td></tr><tr><td>1</td><td>0.93</td><td>0.92</td><td>0.92</td><td>3281</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.92</td><td>6562</td></tr><tr><td>macro avg</td><td>0.92</td><td>0.92</td><td>0.92</td><td>6562</td></tr><tr><td>weighted avg</td><td>0.92</td><td>0.92</td><td>0.92</td><td>6562</td></tr></tbody></table>			precision	recall	f1-score	support	0	0.92	0.93	0.92	3281	1	0.93	0.92	0.92	3281	accuracy			0.92	6562	macro avg	0.92	0.92	0.92	6562	weighted avg	0.92	0.92	0.92	6562	<pre>print(metrics.classification_report(Y_test, Y_pte_XGB))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.96</td><td>0.90</td><td>0.93</td><td>1401</td></tr><tr><td>1</td><td>0.63</td><td>0.81</td><td>0.71</td><td>288</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.89</td><td>1689</td></tr><tr><td>macro avg</td><td>0.80</td><td>0.85</td><td>0.82</td><td>1689</td></tr><tr><td>weighted avg</td><td>0.90</td><td>0.89</td><td>0.89</td><td>1689</td></tr></tbody></table>			precision	recall	f1-score	support	0	0.96	0.90	0.93	1401	1	0.63	0.81	0.71	288	accuracy			0.89	1689	macro avg	0.80	0.85	0.82	1689	weighted avg	0.90	0.89	0.89	1689
	precision	recall	f1-score	support																																																											
0	0.92	0.93	0.92	3281																																																											
1	0.93	0.92	0.92	3281																																																											
accuracy			0.92	6562																																																											
macro avg	0.92	0.92	0.92	6562																																																											
weighted avg	0.92	0.92	0.92	6562																																																											
	precision	recall	f1-score	support																																																											
0	0.96	0.90	0.93	1401																																																											
1	0.63	0.81	0.71	288																																																											
accuracy			0.89	1689																																																											
macro avg	0.80	0.85	0.82	1689																																																											
weighted avg	0.90	0.89	0.89	1689																																																											
ROC_CURVE & ROC_AUC SCORE																																																															
<p>AUC: 0.979</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac6b520188&gt;]</p> 		<p>AUC: 0.920</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac6b6c0048&gt;]</p> 																																																													

**TABLE 11. PERFORMANCE METRICS : EXTREME GRADIENT BOOSTING  
(WITHOUT MODEL TUNNING)**

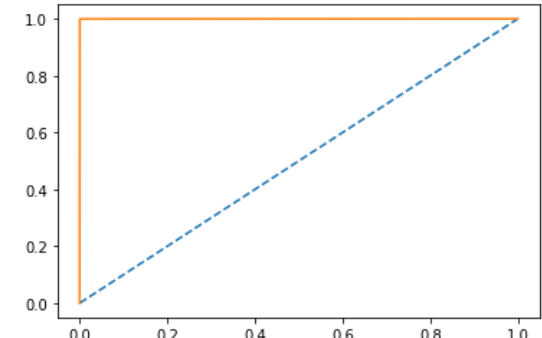
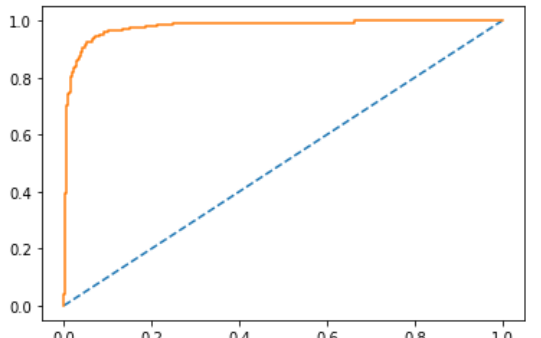


## DECISION TREE (WITH MODEL TUNNING)

TRAIN DATA		TEST DATA																																																													
ACCURACY																																																															
<pre>dt2_train_acc=DT_model_2.score(X_train,Y_train) dt2_train_acc</pre> 0.9833892106065224		<pre>dt2_test_acc=DT_model_2.score(X_test,Y_test) dt2_test_acc</pre> 0.8709295441089402																																																													
CONFUSION MATRIX																																																															
<pre>confusion_matrix(Y_train,Y_ptr_dt2) array([[3214,  67],        [ 42, 3239]], dtype=int64)</pre>		<pre>confusion_matrix(Y_test,Y_pte_dt2) array([[1239, 162],        [ 56, 232]], dtype=int64)</pre>																																																													
CLASSIFICATION REPORT																																																															
<pre>print(classification_report(Y_train,Y_ptr_dt2))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.99</td><td>0.98</td><td>0.98</td><td>3281</td></tr><tr><td>1</td><td>0.98</td><td>0.99</td><td>0.98</td><td>3281</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.98</td><td>6562</td></tr><tr><td>macro avg</td><td>0.98</td><td>0.98</td><td>0.98</td><td>6562</td></tr><tr><td>weighted avg</td><td>0.98</td><td>0.98</td><td>0.98</td><td>6562</td></tr></tbody></table>			precision	recall	f1-score	support	0	0.99	0.98	0.98	3281	1	0.98	0.99	0.98	3281	accuracy			0.98	6562	macro avg	0.98	0.98	0.98	6562	weighted avg	0.98	0.98	0.98	6562	<pre>print(classification_report(Y_test,Y_pte_dt2))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.96</td><td>0.88</td><td>0.92</td><td>1401</td></tr><tr><td>1</td><td>0.59</td><td>0.81</td><td>0.68</td><td>288</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.87</td><td>1689</td></tr><tr><td>macro avg</td><td>0.77</td><td>0.84</td><td>0.80</td><td>1689</td></tr><tr><td>weighted avg</td><td>0.89</td><td>0.87</td><td>0.88</td><td>1689</td></tr></tbody></table>			precision	recall	f1-score	support	0	0.96	0.88	0.92	1401	1	0.59	0.81	0.68	288	accuracy			0.87	1689	macro avg	0.77	0.84	0.80	1689	weighted avg	0.89	0.87	0.88	1689
	precision	recall	f1-score	support																																																											
0	0.99	0.98	0.98	3281																																																											
1	0.98	0.99	0.98	3281																																																											
accuracy			0.98	6562																																																											
macro avg	0.98	0.98	0.98	6562																																																											
weighted avg	0.98	0.98	0.98	6562																																																											
	precision	recall	f1-score	support																																																											
0	0.96	0.88	0.92	1401																																																											
1	0.59	0.81	0.68	288																																																											
accuracy			0.87	1689																																																											
macro avg	0.77	0.84	0.80	1689																																																											
weighted avg	0.89	0.87	0.88	1689																																																											
ROC_CURVE & ROC_AUC SCORE																																																															
<p>AUC: 0.994</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac6bd36048&gt;]</p> 		<p>AUC: 0.840</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac6bd5b488&gt;]</p> 																																																													

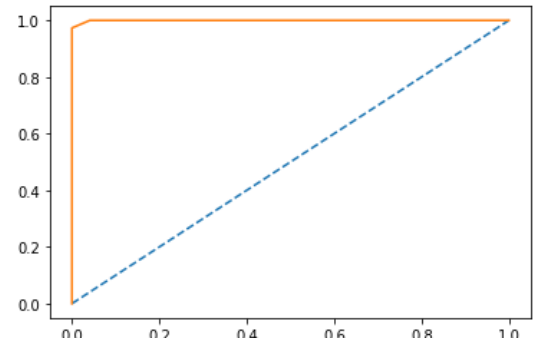
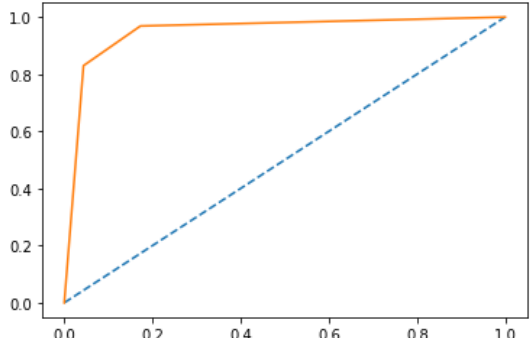
**TABLE 12. PERFORMANCE METRICS : DECISION TREE (WITH MODEL TUNNING)**

## RANDOM FOREST (WITH MODEL TUNNING)

TRAIN DATA	TEST DATA																																																												
ACCURACY																																																													
<pre>rf2_train_acc=rf2.score(X_train,Y_train) rf2_train_acc</pre> 0.9993904297470283	<pre>rf2_test_acc=rf2.score(X_test,Y_test) rf2_test_acc</pre> 0.9467140319715808																																																												
CONFUSION MATRIX																																																													
<pre>confusion_matrix(Y_train,Y_ptr_rf2)</pre> array([[3279, 2], [ 2, 3279]], dtype=int64)	<pre>confusion_matrix(Y_test,Y_pte_rf2)</pre> array([[1347, 54], [ 36, 252]], dtype=int64)																																																												
CLASSIFICATION REPORT																																																													
<pre>print(classification_report(Y_train,Y_ptr_rf2))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>1.00</td><td>1.00</td><td>1.00</td><td>3281</td></tr><tr><td>1</td><td>1.00</td><td>1.00</td><td>1.00</td><td>3281</td></tr><tr><td>accuracy</td><td></td><td></td><td>1.00</td><td>6562</td></tr><tr><td>macro avg</td><td>1.00</td><td>1.00</td><td>1.00</td><td>6562</td></tr><tr><td>weighted avg</td><td>1.00</td><td>1.00</td><td>1.00</td><td>6562</td></tr></tbody></table>		precision	recall	f1-score	support	0	1.00	1.00	1.00	3281	1	1.00	1.00	1.00	3281	accuracy			1.00	6562	macro avg	1.00	1.00	1.00	6562	weighted avg	1.00	1.00	1.00	6562	<pre>print(classification_report(Y_test,Y_pte_rf2))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.97</td><td>0.96</td><td>0.97</td><td>1401</td></tr><tr><td>1</td><td>0.82</td><td>0.88</td><td>0.85</td><td>288</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.95</td><td>1689</td></tr><tr><td>macro avg</td><td>0.90</td><td>0.92</td><td>0.91</td><td>1689</td></tr><tr><td>weighted avg</td><td>0.95</td><td>0.95</td><td>0.95</td><td>1689</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.97	0.96	0.97	1401	1	0.82	0.88	0.85	288	accuracy			0.95	1689	macro avg	0.90	0.92	0.91	1689	weighted avg	0.95	0.95	0.95	1689
	precision	recall	f1-score	support																																																									
0	1.00	1.00	1.00	3281																																																									
1	1.00	1.00	1.00	3281																																																									
accuracy			1.00	6562																																																									
macro avg	1.00	1.00	1.00	6562																																																									
weighted avg	1.00	1.00	1.00	6562																																																									
	precision	recall	f1-score	support																																																									
0	0.97	0.96	0.97	1401																																																									
1	0.82	0.88	0.85	288																																																									
accuracy			0.95	1689																																																									
macro avg	0.90	0.92	0.91	1689																																																									
weighted avg	0.95	0.95	0.95	1689																																																									
ROC_CURVE & ROC_AUC SCORE																																																													
<p>AUC: 1.000</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac6c098a08&gt;]</p> 	<p>AUC: 0.979</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac6baa73c8&gt;]</p> 																																																												

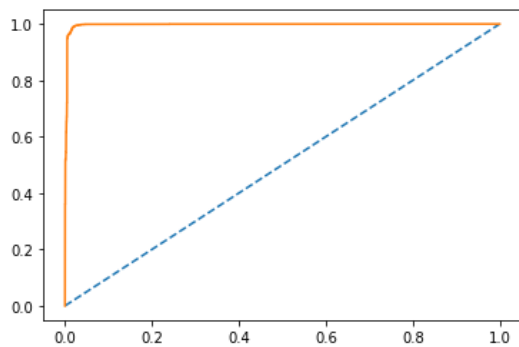
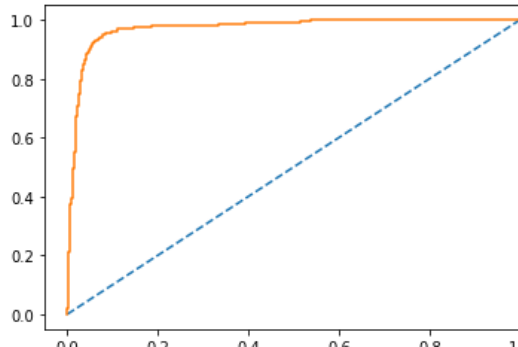
**TABLE 13. PERFORMANCE METRICS : RANDOM FOREST (WITH MODEL TUNNING)**

## KNN (WITH MODEL TUNNING)

TRAIN DATA	TEST DATA																																																												
ACCURACY																																																													
<pre>knn2_train_acc=knn2.score(X_train_std,Y_train) knn2_train_acc  0.9865894544346235</pre>	<pre>knn2_test_acc=knn2.score(X_test_std,Y_test) knn2_test_acc  0.9348727057430433</pre>																																																												
CONFUSION MATRIX																																																													
<pre>confusion_matrix(Y_train,Y_ptr_knn2)  array([[3281,   0],        [ 88, 3193]], dtype=int64)</pre>	<pre>confusion_matrix(Y_test,Y_pte_knn2)  array([[1340,   61],        [ 49, 239]], dtype=int64)</pre>																																																												
CLASSIFICATION REPORT																																																													
<pre>print(classification_report(Y_train,Y_ptr_knn2))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.97</td><td>1.00</td><td>0.99</td><td>3281</td></tr><tr><td>1</td><td>1.00</td><td>0.97</td><td>0.99</td><td>3281</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.99</td><td>6562</td></tr><tr><td>macro avg</td><td>0.99</td><td>0.99</td><td>0.99</td><td>6562</td></tr><tr><td>weighted avg</td><td>0.99</td><td>0.99</td><td>0.99</td><td>6562</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.97	1.00	0.99	3281	1	1.00	0.97	0.99	3281	accuracy			0.99	6562	macro avg	0.99	0.99	0.99	6562	weighted avg	0.99	0.99	0.99	6562	<pre>print(classification_report(Y_test,Y_pte_knn2))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.96</td><td>0.96</td><td>0.96</td><td>1401</td></tr><tr><td>1</td><td>0.80</td><td>0.83</td><td>0.81</td><td>288</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.93</td><td>1689</td></tr><tr><td>macro avg</td><td>0.88</td><td>0.89</td><td>0.89</td><td>1689</td></tr><tr><td>weighted avg</td><td>0.94</td><td>0.93</td><td>0.94</td><td>1689</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.96	0.96	0.96	1401	1	0.80	0.83	0.81	288	accuracy			0.93	1689	macro avg	0.88	0.89	0.89	1689	weighted avg	0.94	0.93	0.94	1689
	precision	recall	f1-score	support																																																									
0	0.97	1.00	0.99	3281																																																									
1	1.00	0.97	0.99	3281																																																									
accuracy			0.99	6562																																																									
macro avg	0.99	0.99	0.99	6562																																																									
weighted avg	0.99	0.99	0.99	6562																																																									
	precision	recall	f1-score	support																																																									
0	0.96	0.96	0.96	1401																																																									
1	0.80	0.83	0.81	288																																																									
accuracy			0.93	1689																																																									
macro avg	0.88	0.89	0.89	1689																																																									
weighted avg	0.94	0.93	0.94	1689																																																									
ROC_CURVE & ROC_AUC SCORE																																																													
<p>AUC: 0.999</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac6e575b88&gt;]</p> 	<p>AUC: 0.949</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac6e6e5988&gt;]</p> 																																																												

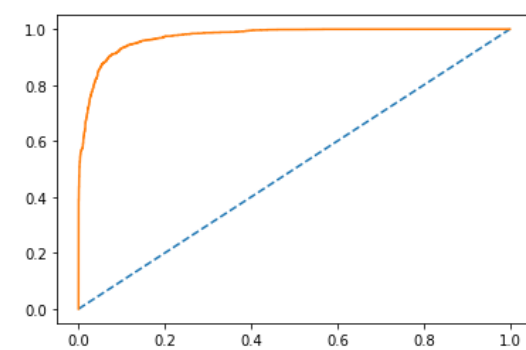
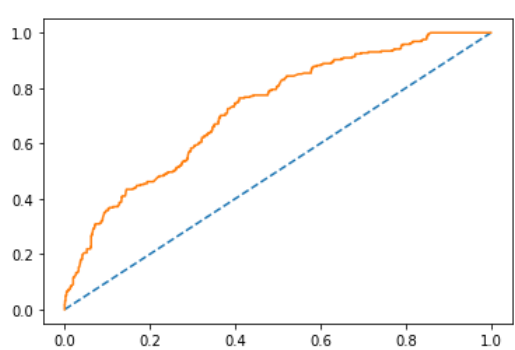
**TABLE 14. PERFORMANCE METRICS : KNN (WITH MODEL TUNNING)**

### SVC (WITH MODEL TUNNING)

TRAIN DATA	TEST DATA																																																												
ACCURACY																																																													
<pre>svc2_train_acc=svc2.score(X_train_std,Y_train) svc2_train_acc</pre> 0.9852179213654374	<pre>svc2_test_acc=svc2.score(X_test_std,Y_test) svc2_test_acc</pre> 0.8768502072232089																																																												
CONFUSION MATRIX																																																													
<pre>confusion_matrix(Y_train,Y_ptr_svc2)</pre> <pre>array([[3210, 71],        [ 26, 3255]], dtype=int64)</pre>	<pre>confusion_matrix(Y_test,Y_pte_svc2)</pre> <pre>array([[1201, 200],        [ 8, 280]], dtype=int64)</pre>																																																												
CLASSIFICATION REPORT																																																													
<pre>print(classification_report(Y_train,Y_ptr_svc2))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.99</td><td>0.98</td><td>0.99</td><td>3281</td></tr><tr><td>1</td><td>0.98</td><td>0.99</td><td>0.99</td><td>3281</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.99</td><td>6562</td></tr><tr><td>macro avg</td><td>0.99</td><td>0.99</td><td>0.99</td><td>6562</td></tr><tr><td>weighted avg</td><td>0.99</td><td>0.99</td><td>0.99</td><td>6562</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.99	0.98	0.99	3281	1	0.98	0.99	0.99	3281	accuracy			0.99	6562	macro avg	0.99	0.99	0.99	6562	weighted avg	0.99	0.99	0.99	6562	<pre>print(classification_report(Y_test,Y_pte_svc2))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.99</td><td>0.86</td><td>0.92</td><td>1401</td></tr><tr><td>1</td><td>0.58</td><td>0.97</td><td>0.73</td><td>288</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.88</td><td>1689</td></tr><tr><td>macro avg</td><td>0.79</td><td>0.91</td><td>0.82</td><td>1689</td></tr><tr><td>weighted avg</td><td>0.92</td><td>0.88</td><td>0.89</td><td>1689</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.99	0.86	0.92	1401	1	0.58	0.97	0.73	288	accuracy			0.88	1689	macro avg	0.79	0.91	0.82	1689	weighted avg	0.92	0.88	0.89	1689
	precision	recall	f1-score	support																																																									
0	0.99	0.98	0.99	3281																																																									
1	0.98	0.99	0.99	3281																																																									
accuracy			0.99	6562																																																									
macro avg	0.99	0.99	0.99	6562																																																									
weighted avg	0.99	0.99	0.99	6562																																																									
	precision	recall	f1-score	support																																																									
0	0.99	0.86	0.92	1401																																																									
1	0.58	0.97	0.73	288																																																									
accuracy			0.88	1689																																																									
macro avg	0.79	0.91	0.82	1689																																																									
weighted avg	0.92	0.88	0.89	1689																																																									
ROC_CURVE & ROC_AUC SCORE																																																													
AUC: 0.997 [<matplotlib.lines.Line2D at 0x1ac717ff708>] 	AUC: 0.972 [<matplotlib.lines.Line2D at 0x1ac71863748>] 																																																												

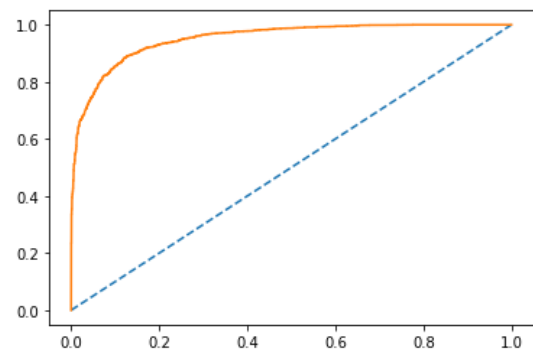
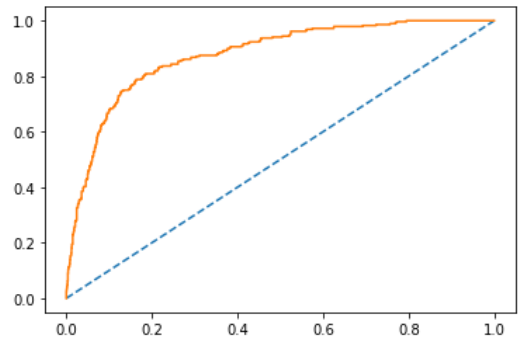
**TABLE 15. PERFORMANCE METRICS : SVC (WITH MODEL TUNNING)**

### BAGGING MODEL (WITH MODEL TUNNING)

TRAIN DATA	TEST DATA																																																												
ACCURACY																																																													
<pre>bag2_train_acc=bag2.score(X_train,Y_train) bag2_train_acc</pre> 0.9138982017677537	<pre>bag2_test_acc=bag2.score(X_test,Y_test) bag2_test_acc</pre> 0.8644168146832445																																																												
CONFUSION MATRIX																																																													
<pre>confusion_matrix(Y_train,Y_ptr_bag2)</pre> array([[3007, 274], [ 291, 2990]], dtype=int64)	<pre>confusion_matrix(Y_test,Y_pte_bag2)</pre> array([[1237, 164], [ 65, 223]], dtype=int64)																																																												
CLASSIFICATION REPORT																																																													
<pre>print(classification_report(Y_train,Y_ptr_bag2))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.91</td><td>0.92</td><td>0.91</td><td>3281</td></tr><tr><td>1</td><td>0.92</td><td>0.91</td><td>0.91</td><td>3281</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.91</td><td>6562</td></tr><tr><td>macro avg</td><td>0.91</td><td>0.91</td><td>0.91</td><td>6562</td></tr><tr><td>weighted avg</td><td>0.91</td><td>0.91</td><td>0.91</td><td>6562</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.91	0.92	0.91	3281	1	0.92	0.91	0.91	3281	accuracy			0.91	6562	macro avg	0.91	0.91	0.91	6562	weighted avg	0.91	0.91	0.91	6562	<pre>print(classification_report(Y_test,Y_pte_bag2))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.95</td><td>0.88</td><td>0.92</td><td>1401</td></tr><tr><td>1</td><td>0.58</td><td>0.77</td><td>0.66</td><td>288</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.86</td><td>1689</td></tr><tr><td>macro avg</td><td>0.76</td><td>0.83</td><td>0.79</td><td>1689</td></tr><tr><td>weighted avg</td><td>0.89</td><td>0.86</td><td>0.87</td><td>1689</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.95	0.88	0.92	1401	1	0.58	0.77	0.66	288	accuracy			0.86	1689	macro avg	0.76	0.83	0.79	1689	weighted avg	0.89	0.86	0.87	1689
	precision	recall	f1-score	support																																																									
0	0.91	0.92	0.91	3281																																																									
1	0.92	0.91	0.91	3281																																																									
accuracy			0.91	6562																																																									
macro avg	0.91	0.91	0.91	6562																																																									
weighted avg	0.91	0.91	0.91	6562																																																									
	precision	recall	f1-score	support																																																									
0	0.95	0.88	0.92	1401																																																									
1	0.58	0.77	0.66	288																																																									
accuracy			0.86	1689																																																									
macro avg	0.76	0.83	0.79	1689																																																									
weighted avg	0.89	0.86	0.87	1689																																																									
ROC_CURVE & ROC_AUC SCORE																																																													
<p>AUC: 0.973</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac6e5f5488&gt;]</p> 	<p>AUC: 0.724</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac71a29ec8&gt;]</p> 																																																												

**TABLE 16. PERFORMANCE METRICS : BAGGING MODEL (WITH MODEL TUNNING)**

### ADABOOST MODEL (WITH MODEL TUNNING)

TRAIN DATA	TEST DATA																																																												
ACCURACY																																																													
<pre>ada2_train_acc=ada2.score(X_train,Y_train) ada2_train_acc</pre> 0.8808290155440415	<pre>ada2_test_acc=best_grid.score(X_test,Y_test) ada2_test_acc</pre> 0.8709295441089402																																																												
CONFUSION MATRIX																																																													
<pre>confusion_matrix(Y_train,Y_ptr_ada2)</pre> array([[2862, 419], [ 363, 2918]], dtype=int64)	<pre>confusion_matrix(Y_test,Y_pte_ada2)</pre> array([[1154, 247], [ 60, 228]], dtype=int64)																																																												
CLASSIFICATION REPORT																																																													
<pre>print(classification_report(Y_train,Y_ptr_ada2))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.89</td><td>0.87</td><td>0.88</td><td>3281</td></tr><tr><td>1</td><td>0.87</td><td>0.89</td><td>0.88</td><td>3281</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.88</td><td>6562</td></tr><tr><td>macro avg</td><td>0.88</td><td>0.88</td><td>0.88</td><td>6562</td></tr><tr><td>weighted avg</td><td>0.88</td><td>0.88</td><td>0.88</td><td>6562</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.89	0.87	0.88	3281	1	0.87	0.89	0.88	3281	accuracy			0.88	6562	macro avg	0.88	0.88	0.88	6562	weighted avg	0.88	0.88	0.88	6562	<pre>print(classification_report(Y_test,Y_pte_ada2))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.95</td><td>0.82</td><td>0.88</td><td>1401</td></tr><tr><td>1</td><td>0.48</td><td>0.79</td><td>0.60</td><td>288</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.82</td><td>1689</td></tr><tr><td>macro avg</td><td>0.72</td><td>0.81</td><td>0.74</td><td>1689</td></tr><tr><td>weighted avg</td><td>0.87</td><td>0.82</td><td>0.83</td><td>1689</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.95	0.82	0.88	1401	1	0.48	0.79	0.60	288	accuracy			0.82	1689	macro avg	0.72	0.81	0.74	1689	weighted avg	0.87	0.82	0.83	1689
	precision	recall	f1-score	support																																																									
0	0.89	0.87	0.88	3281																																																									
1	0.87	0.89	0.88	3281																																																									
accuracy			0.88	6562																																																									
macro avg	0.88	0.88	0.88	6562																																																									
weighted avg	0.88	0.88	0.88	6562																																																									
	precision	recall	f1-score	support																																																									
0	0.95	0.82	0.88	1401																																																									
1	0.48	0.79	0.60	288																																																									
accuracy			0.82	1689																																																									
macro avg	0.72	0.81	0.74	1689																																																									
weighted avg	0.87	0.82	0.83	1689																																																									
ROC_CURVE & ROC_AUC SCORE																																																													
AUC: 0.952 [<matplotlib.lines.Line2D at 0x1ac72cd6608>] 	AUC: 0.877 [<matplotlib.lines.Line2D at 0x1ac72bcd5c8>] 																																																												

**TABLE 17. PERFORMANCE METRICS : ADABOOST MODEL (WITH MODEL TUNNING)**

### GRADIENT BOOSTING MODEL (WITH MODEL TUNNING)

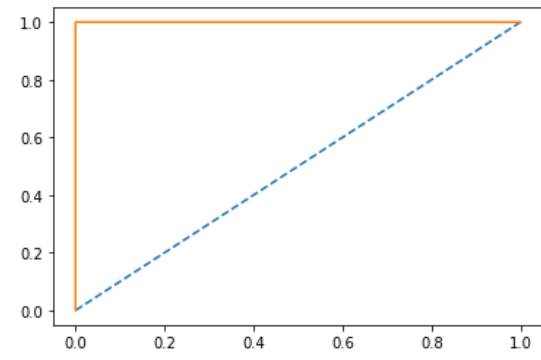
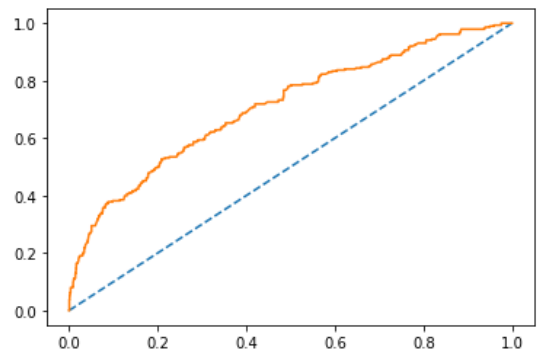
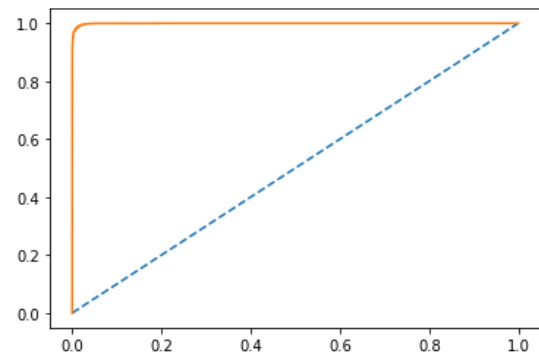
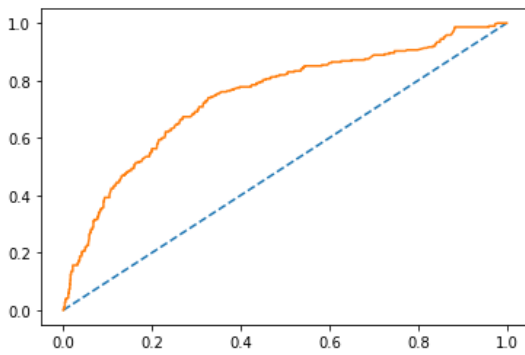
TRAIN DATA	TEST DATA																																																												
ACCURACY																																																													
<pre>gb2_train_acc=gb2.score(X_train,Y_train) gb2_train_acc  0.999847607436757</pre>	<pre>gb2_test_acc=gb2.score(X_test,Y_test) gb2_test_acc  0.9680284191829485</pre>																																																												
CONFUSION MATRIX																																																													
<pre>confusion_matrix(Y_train,Y_ptr_gb2)  array([[3280, 1],        [ 0, 3281]], dtype=int64)</pre>	<pre>confusion_matrix(Y_test,Y_pte_gb2)  array([[1368, 33],        [ 21, 267]], dtype=int64)</pre>																																																												
CLASSIFICATION REPORT																																																													
<pre>print(classification_report(Y_train,Y_ptr_gb2))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>1.00</td><td>1.00</td><td>1.00</td><td>3281</td></tr><tr><td>1</td><td>1.00</td><td>1.00</td><td>1.00</td><td>3281</td></tr><tr><td>accuracy</td><td></td><td></td><td>1.00</td><td>6562</td></tr><tr><td>macro avg</td><td>1.00</td><td>1.00</td><td>1.00</td><td>6562</td></tr><tr><td>weighted avg</td><td>1.00</td><td>1.00</td><td>1.00</td><td>6562</td></tr></tbody></table>		precision	recall	f1-score	support	0	1.00	1.00	1.00	3281	1	1.00	1.00	1.00	3281	accuracy			1.00	6562	macro avg	1.00	1.00	1.00	6562	weighted avg	1.00	1.00	1.00	6562	<pre>print(classification_report(Y_test,Y_pte_gb2))</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.98</td><td>0.98</td><td>0.98</td><td>1401</td></tr><tr><td>1</td><td>0.89</td><td>0.93</td><td>0.91</td><td>288</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.97</td><td>1689</td></tr><tr><td>macro avg</td><td>0.94</td><td>0.95</td><td>0.94</td><td>1689</td></tr><tr><td>weighted avg</td><td>0.97</td><td>0.97</td><td>0.97</td><td>1689</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.98	0.98	0.98	1401	1	0.89	0.93	0.91	288	accuracy			0.97	1689	macro avg	0.94	0.95	0.94	1689	weighted avg	0.97	0.97	0.97	1689
	precision	recall	f1-score	support																																																									
0	1.00	1.00	1.00	3281																																																									
1	1.00	1.00	1.00	3281																																																									
accuracy			1.00	6562																																																									
macro avg	1.00	1.00	1.00	6562																																																									
weighted avg	1.00	1.00	1.00	6562																																																									
	precision	recall	f1-score	support																																																									
0	0.98	0.98	0.98	1401																																																									
1	0.89	0.93	0.91	288																																																									
accuracy			0.97	1689																																																									
macro avg	0.94	0.95	0.94	1689																																																									
weighted avg	0.97	0.97	0.97	1689																																																									
ROC_CURVE & ROC_AUC SCORE																																																													
<p>AUC: 1.000</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac72ed8dc8&gt;]</p> 	<p>AUC: 0.710</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac72f33888&gt;]</p> 																																																												

TABLE 18. PERFORMANCE METRICS : GRADIENT BOOSTING MODEL (WITH MODEL TUNNING)

## EXTREME GRADIENT BOOSTING MODEL

(WITH MODEL TUNNING)

TRAIN DATA	TEST DATA																																																												
ACCURACY																																																													
<pre>xgb2_train_acc=xgb2.score(X_train,Y_train) xgb2_train_acc  0.9871990246875952</pre>	<pre>xgb2_test_acc=best_grid.score(X_test,Y_test) xgb2_test_acc  0.8709295441089402</pre>																																																												
CONFUSION MATRIX																																																													
<pre>confusion_matrix(Y_train,Y_ptr_xgb2)  array([[3233, 48],        [ 36, 3245]], dtype=int64)</pre>	<pre>confusion_matrix(Y_test,Y_pte_xgb2)  array([[1330, 71],        [ 44, 244]], dtype=int64)</pre>																																																												
CLASSIFICATION REPORT																																																													
<pre>print(classification_report(Y_train,Y_ptr_xgb2))</pre> <table><tr><td></td><td>precision</td><td>recall</td><td>f1-score</td><td>support</td></tr><tr><td>0</td><td>0.99</td><td>0.99</td><td>0.99</td><td>3281</td></tr><tr><td>1</td><td>0.99</td><td>0.99</td><td>0.99</td><td>3281</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.99</td><td>6562</td></tr><tr><td>macro avg</td><td>0.99</td><td>0.99</td><td>0.99</td><td>6562</td></tr><tr><td>weighted avg</td><td>0.99</td><td>0.99</td><td>0.99</td><td>6562</td></tr></table>		precision	recall	f1-score	support	0	0.99	0.99	0.99	3281	1	0.99	0.99	0.99	3281	accuracy			0.99	6562	macro avg	0.99	0.99	0.99	6562	weighted avg	0.99	0.99	0.99	6562	<pre>print(classification_report(Y_test,Y_pte_xgb2))</pre> <table><tr><td></td><td>precision</td><td>recall</td><td>f1-score</td><td>support</td></tr><tr><td>0</td><td>0.97</td><td>0.95</td><td>0.96</td><td>1401</td></tr><tr><td>1</td><td>0.77</td><td>0.85</td><td>0.81</td><td>288</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.93</td><td>1689</td></tr><tr><td>macro avg</td><td>0.87</td><td>0.90</td><td>0.88</td><td>1689</td></tr><tr><td>weighted avg</td><td>0.94</td><td>0.93</td><td>0.93</td><td>1689</td></tr></table>		precision	recall	f1-score	support	0	0.97	0.95	0.96	1401	1	0.77	0.85	0.81	288	accuracy			0.93	1689	macro avg	0.87	0.90	0.88	1689	weighted avg	0.94	0.93	0.93	1689
	precision	recall	f1-score	support																																																									
0	0.99	0.99	0.99	3281																																																									
1	0.99	0.99	0.99	3281																																																									
accuracy			0.99	6562																																																									
macro avg	0.99	0.99	0.99	6562																																																									
weighted avg	0.99	0.99	0.99	6562																																																									
	precision	recall	f1-score	support																																																									
0	0.97	0.95	0.96	1401																																																									
1	0.77	0.85	0.81	288																																																									
accuracy			0.93	1689																																																									
macro avg	0.87	0.90	0.88	1689																																																									
weighted avg	0.94	0.93	0.93	1689																																																									
ROC_CURVE & ROC_AUC SCORE																																																													
<p>AUC: 0.999</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac743d4448&gt;]</p> 	<p>AUC: 0.743</p> <p>[&lt;matplotlib.lines.Line2D at 0x1ac7418a308&gt;]</p> 																																																												

**TABLE 19. PERFORMANCE METRICS : EXTREME GRADIENT BOOSTING MODEL**

(WITH MODEL TUNNING)