CAPSTONE PROJECT

ON

Customer Churn Prediction

BY

Santan Chakraborty

PGP-DSBA

(June-2020 B)

FINAL
PROJECT REPORT

Contents

- ## Introduction - What did you wish to achieve while doing the project?

Ans. The ability to predict that a particular customer is at a high risk of churning, while there is still time to do something about it, represents a huge additional potential revenue source for every business. Besides the direct loss of revenue that results from a customer abandoning the business, the costs of initially acquiring that customer may not have already been covered by the customer's spending to date. (In other words, acquiring that customer may have been a losing investment.) Furthermore, it is always more difficult and expensive to acquire a new customer than it is to retain a current paying customer. **My goal is to design a model which can effectively helps to find out a potential churner.**

- ## EDA - Univariate / Bi-variate / multi-variate analysis to understand relationship b/w variables. - Both visual and non-visual understanding of the data.

Ans. In statistics, **exploratory data analysis** is an approach of analysing data sets to summarize their main characteristics, often using statistical graphics and other data visualization methods.

Descriptive Details:

	count	mean	std	min	25%	50%	75%	max
AccountID	11260.0	25629.500000	3250.626350	20000.0	22814.75	25629.5	28444.25	31259.0
Churn	11260.0	0.168384	0.374223	0.0	0.00	0.0	0.00	1.0
City_Tier	11148.0	1.653929	0.915015	1.0	1.00	1.0	3.00	3.0
CC_Contacted_LY	11158.0	17.867091	8.853269	4.0	11.00	16.0	23.00	132.0
Service_Score	11162.0	2.902526	0.725584	0.0	2.00	3.0	3.00	5.0
CC_Agent_Score	11144.0	3.066493	1.379772	1.0	2.00	3.0	4.00	5.0
Complain_ly	10903.0	0.285334	0.451594	0.0	0.00	0.0	1.00	1.0

Numerical

	count	unique	top	freq
Tenure	11158	38	1	1351
Payment	11151	5	Debit Card	4587
Gender	11152	4	Male	6328
Account_user_count	11148	7	4	4569
account_segment	11163	7	Super	4062
Marital_Status	11048	3	Married	5860
rev_per_month	11158	59	3	1746
rev_growth_yoy	11260	20	14	1524
coupon_used_for_payment	11260	20	1	4373
Day_Since_CC_connect	10903	24	3	1816
cashback	10789	5693	155.62	10
Login_device	11039	3	Mobile	7482

Categorical

Univariate Analysis:

To get a proper visual demonstration of the dataset and the correlations between the features we import the libraries. import **matplotlib.pyplot as plt %Matplotlib inline import seaborn as sns.**

Graphs:

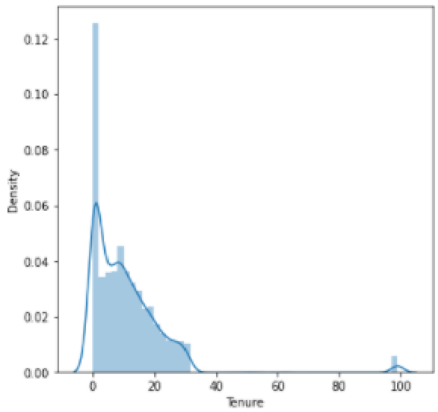


Fig: 1 Tenure

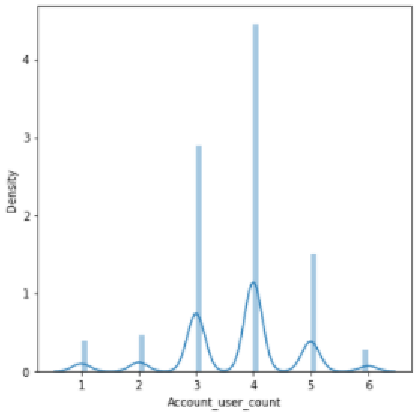


Fig: 2 Account_user_count

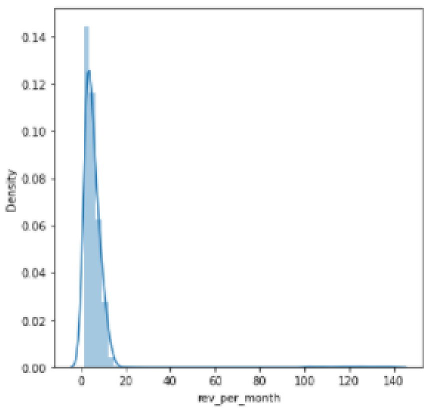


Fig: 3 Rev_per_month

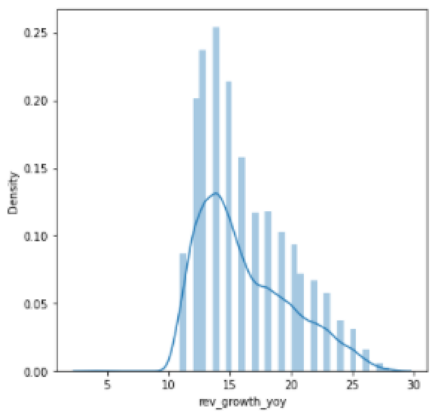


Fig: 4 Rrv_growth_yoy

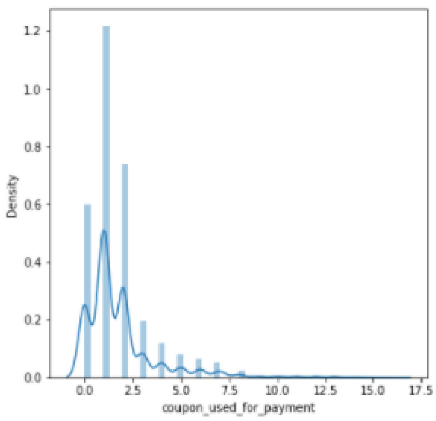


Fig: 5 Coupon_used_for_payment

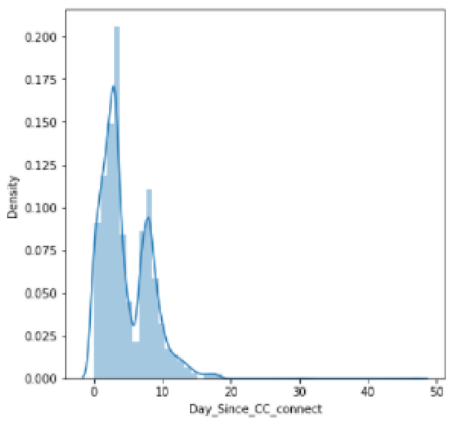


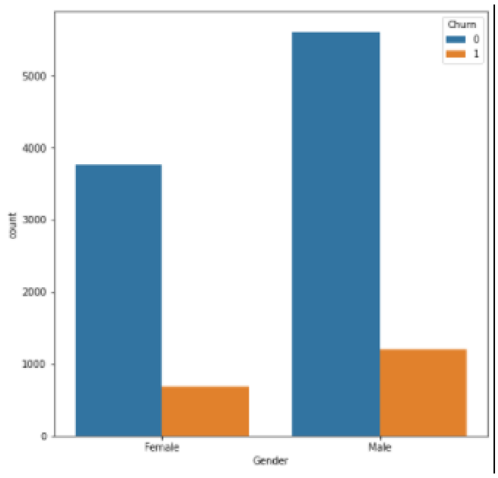
Fig: 6 Day_since_cc_connect

From Fig:1 it is clear that ‘Tenure’ is right skewed
Fig:2 ‘Account_user_count’ is multimodal variable.
Fig:3 ‘Rev_per_month’ highly skewed variable.
Fig:4 ‘Rrv_growth_yoy’ right skewed.
Fig:5 ‘Coupon_used_for_payment’ is multimodal right skewed.
Fig:6 ‘Day_since_cc_connect’ is multimodal.

Bivariate Analysis:

To get a proper visual demonstration of the dataset and the correlations between the features we import the libraries. import **matplotlib.pyplot as plt %Matplotlib inline import seaborn as sns.**

Graphs:



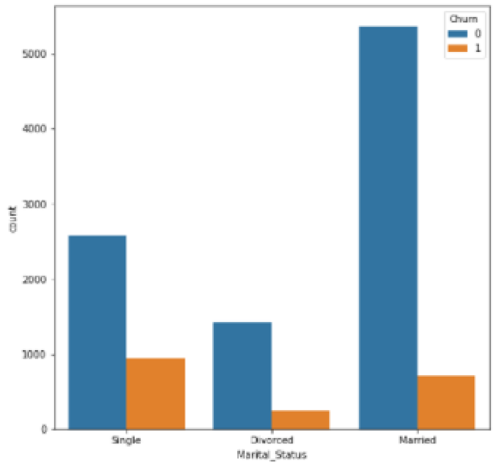
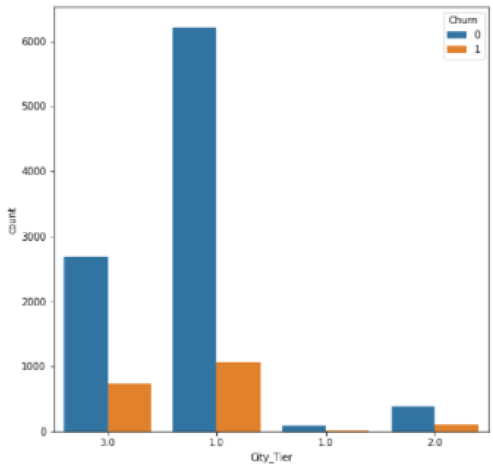


Fig:7 Churn-Gender

Fig:8 Churn-Marital_status



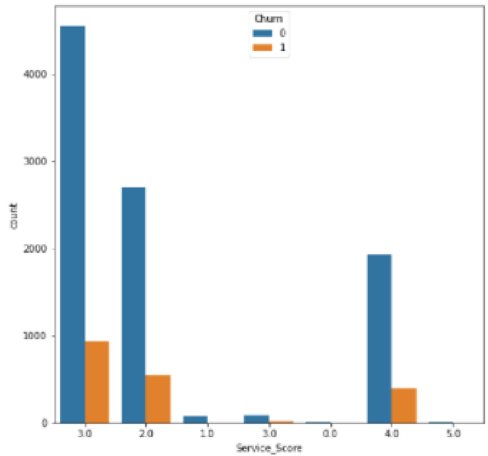


Fig:9 Churn-City_Tire

Fig:10 Churn-Service_Score

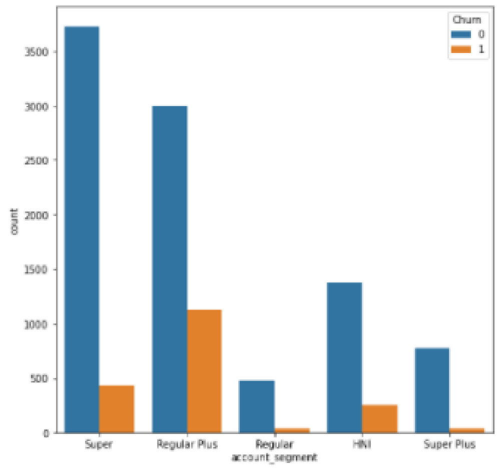


Fig:11 Churn-Account_segment

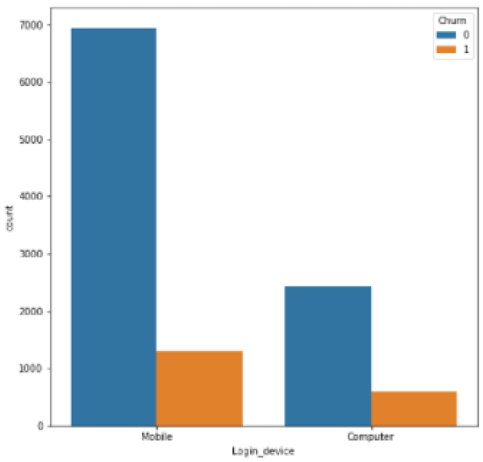


Fig:12 Churn-Login_device

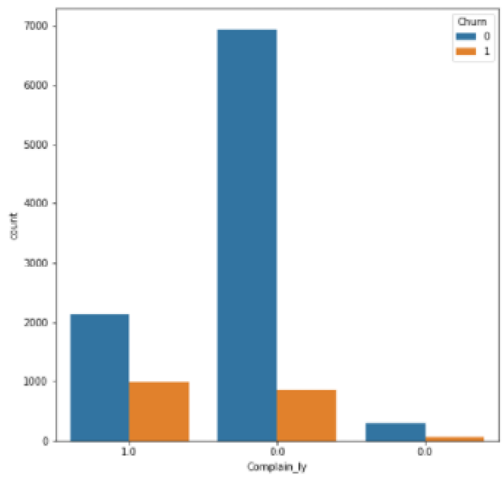


Fig:13 Churn-Complain_ly

From Fig:7 we can conclude that number of churns in Male is more than the number of Female.
Fig:8 We can conclude no. of churns are more in Singles in compared to others.
Fig:9 Tire 1 and Tire 2 city customers are Churning more than the others.
Fig:10 The customers with Service_score 3.0 Churned more than the others.
Fig:11 Regular plus subscribers churned more than the others.
Fig:12 Mobile users churned more than the computer users.
Fig:13 Complain_ly=1.0 Churned more than the others.

Multivariate Analysis:

To get a proper visual demonstration of the dataset and the correlations between the features we import the libraries. import **matplotlib.pyplot as plt %Matplotlib inline import seaborn as sns.**

Graphs:

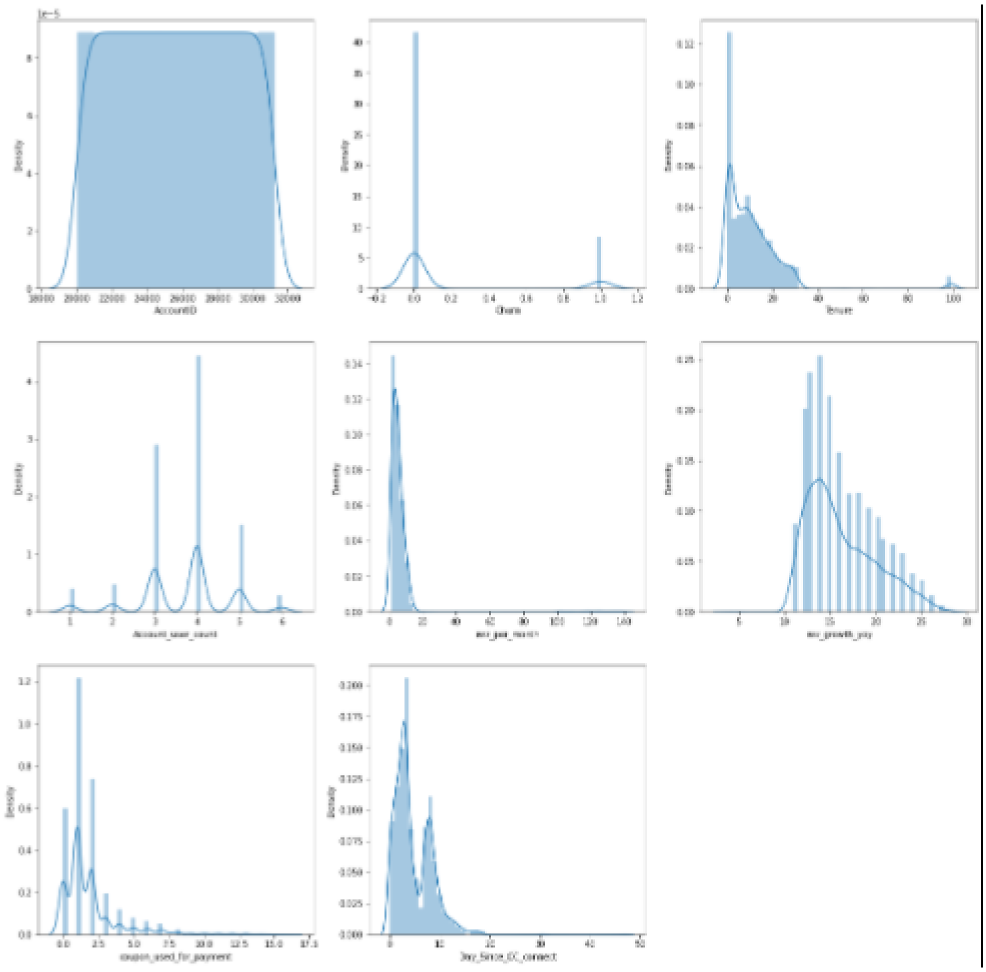


Fig:14 Distplot

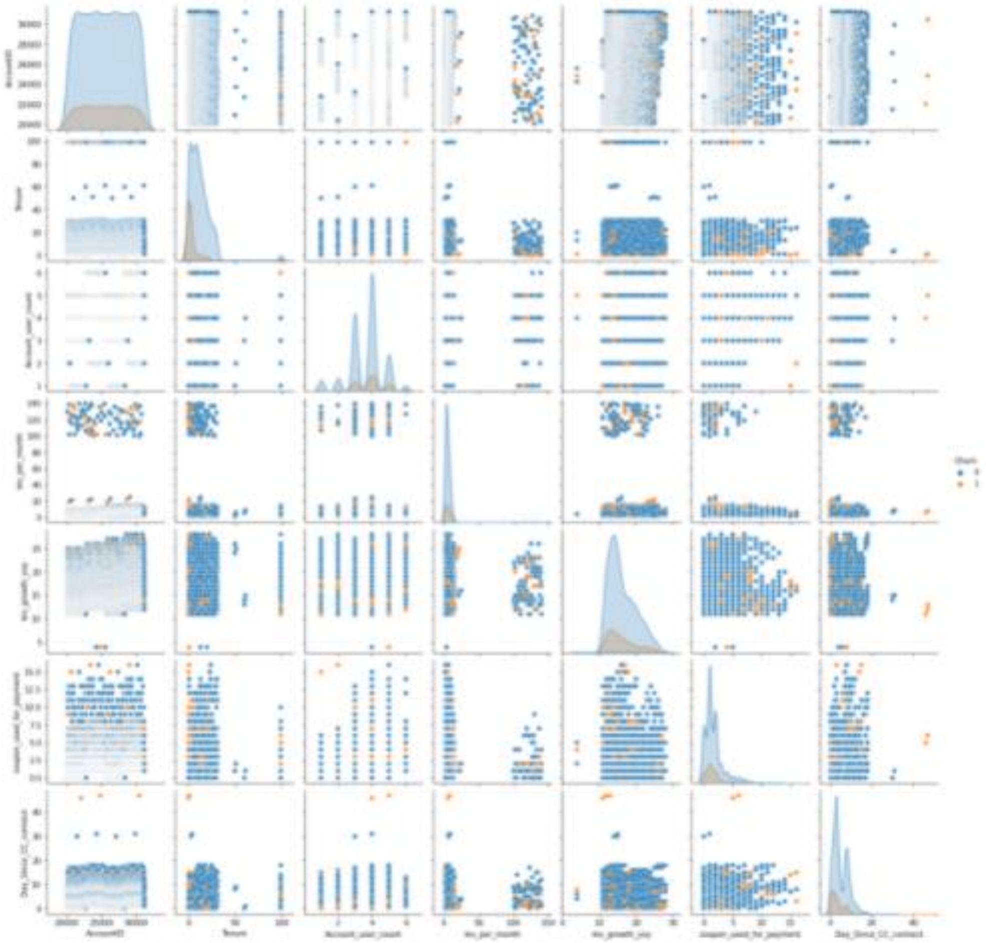


Fig:15 Pairplot

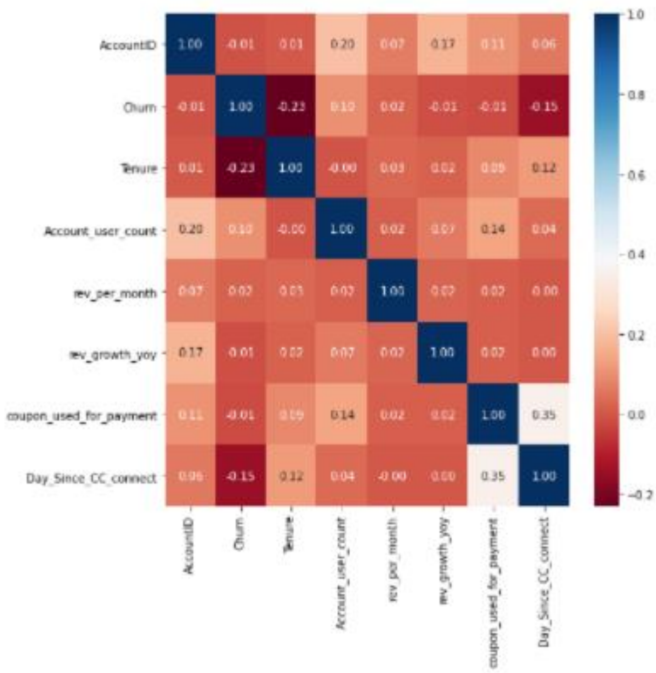


Fig:16 Heat Map

First, I checked From the Figure-14 Distplot, Most of Features are showing Multimodal right skewed distribution plot, so the features contain huge number of outliers in it.
We must apply outlier removal technique to clean the data to process further.
From Figure-15 Pairplot from the pair plot we can see that data points are highly overlapped and convoluted as well so the Linear Regression Model will not work much so we will go for classification techniques such as Random Forest, Decision Tree or **K-Nearest Neighbours (KNN)**. **From Figure-16 Heat Map**,
I am using heat map to get a visualization of the correlations among the features of the dataset dark colour depicts high correlation between two features and light colour shows no correlation.

- ### Data Cleaning and Pre-processing - Approach used for identifying and treating missing values and outlier treatment (and why) - Need for variable transformation (if any) - Variables removed or added and why (if any)

Ans. Data Cleaning and Pre-processing:

- My first job is treating the null vales are present in the dataset.

To check the total Null values in each Features I used **isnull(). sum ()** and from **OutPut-2** I can see that the columns contain a lot of null values in it.

```
AccountID                    0
Churn                        0
Tenure                     102
City_Tier                  112
CC_Contacted_LY            102
Payment                    109
Gender                     108
Service_Score               98
Account_user_count         112
account_segment             97
CC_Agent_Score             116
Marital_Status             212
rev_per_month              102
Complain_ly                357
rev_growth_yoy               0
coupon_used_for_payment      0
Day_Since_CC_connect       357
cashback                   471
Login_device               221
dtype: int64
```

Fig:17 Null values

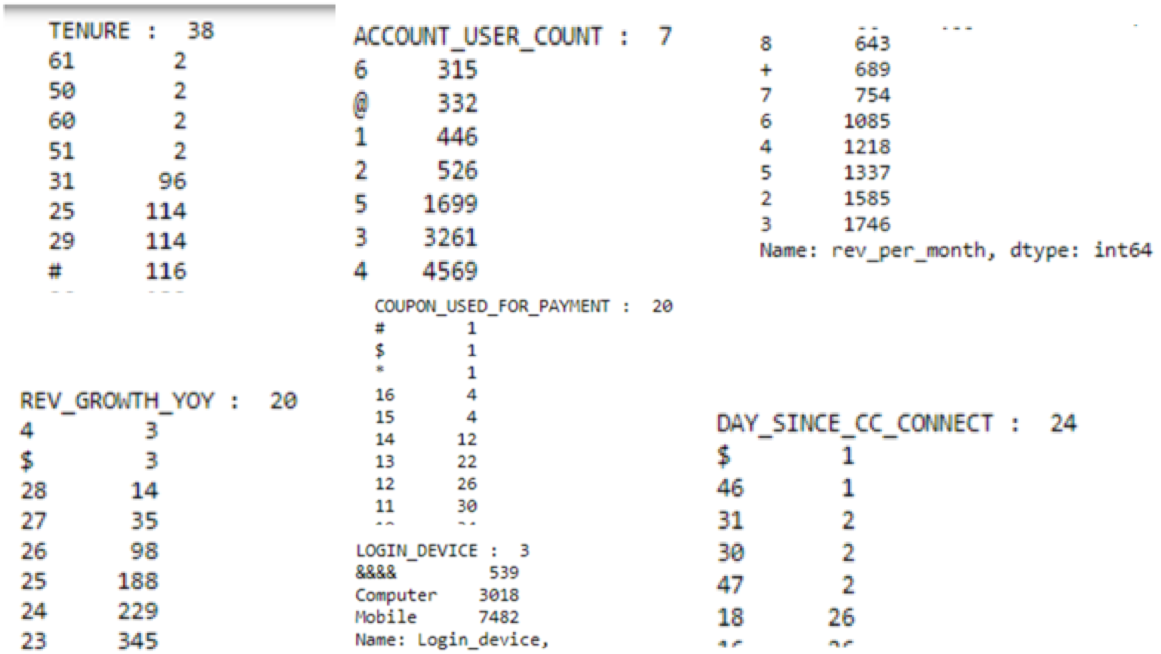
Now, my next job is treating the Null values by imputing the values with relevant replacements. In case of **Numerical Features**, I Imputed the Null values with **Medians** and **Categorical Features** with **Mode**.

```
AccountID                  0
Churn                      0
Tenure                     0
City_Tier                  0
CC_Contacted_LY            0
Payment                    0
Gender                     0
Service_Score              0
Account_user_count         0
account_segment            0
CC_Agent_Score             0
Marital_Status             0
rev_per_month              0
Complain_ly                0
rev_growth_yoy             0
coupon_used_for_payment    0
Day_Since_CC_connect       0
cashback                   0
Login_device               0
dtype: int64
```

Fig:18 After treating Null values

Before proceeding I will check whether there are some duplicate values present or not. To check that I will use **duplicated ().** From the results, no duplicate rows are present there.

- My next job is treating the special symbols present in the dataset.
- First, I will check if any of the Feature contains any unwanted signs like **[@, #, %, ggg, &&&, *] (Except numbers or alphabets)**
- From **Figure-19** it is visible that some of the features contain unwanted signs.
- Now I come across some problems which is related to data collection such as **(RegularPlus as Regular + or Male as M).**
- To get rid of such interruptions I replace the made-up (**Regular + or M**) data points with the original one (**Regular Plus or Male**) by using **relace ()**



```
TENURE :   38
61          2
50          2
60          2
51          2
31         96
25        114
29        114
#         116

ACCOUNT_USER_COUNT :   7
6      315
@      332
1      446
2      526
5     1699
3     3261
4     4569

8       643
+       689
7       754
6      1085
4      1218
5      1337
2      1585
3      1746
Name: rev_per_month, dtype: int64

COUPON_USED_FOR_PAYMENT :   20
#       1
$       1
*       1
16      4
15      4
14     12
13     22
12     26
11     30

REV_GROWTH_YOY :   20
4          3
$          3
28        14
27        35
26        98
25       188
24       229
23       345

LOGIN_DEVICE :   3
&&&&         539
Computer    3018
Mobile      7482
Name: Login_device,

DAY_SINCE_CC_CONNECT :   24
$        1
46       1
31       2
30       2
47       2
18      26
```

Fig:19 Unwanted Symbols

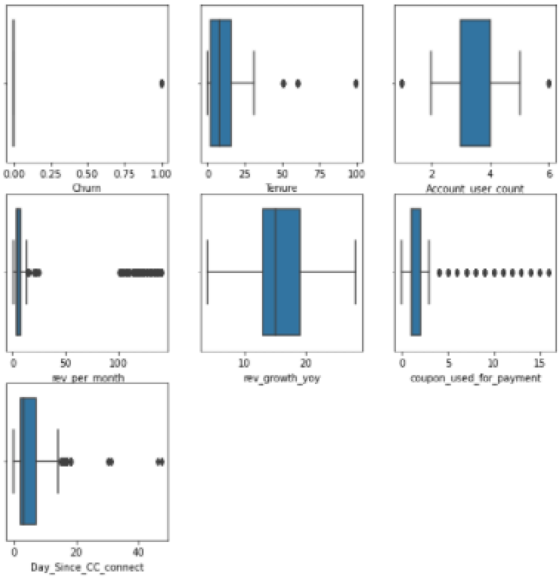
- **Outlier treatment (if required) e) Variable transformation (if applicable) f) Addition of new variables (if required)**

Now, to check the outliers, I am using Boxplots for all the numerical features of the dataset. I predicted earlier the features hold a lot of outliers. It is visible from the **Figure-20 Boxplot**. To remove these outliers, I used IQR method from the **SciPy library of Python**.

IQR method:

I replace all the upper outlier values and bring them to the upper whisker level. Similarly, all the lower outlier values and bring down to the lower whisker levels. This will not affect the shape of the dataset. If I only need the high value and low value of the dataset then this method is useful. But if we need the difference of the values then this value is not useful. Now I am creating a user-defined function for finding the lower and upper range for a variable so that outlier can be treated. Here I identify the **IQR**, lower whisker, and upper whisker**. lower range= Q1-(1.5 * IQR)** and **upper range= Q3+(1.5 * IQR).**

Using the value for all the columns.**From Figure-21 BoxPlot (Outliers Removed)** the outliers are removed, and my dataset is cleaned to proceed further.



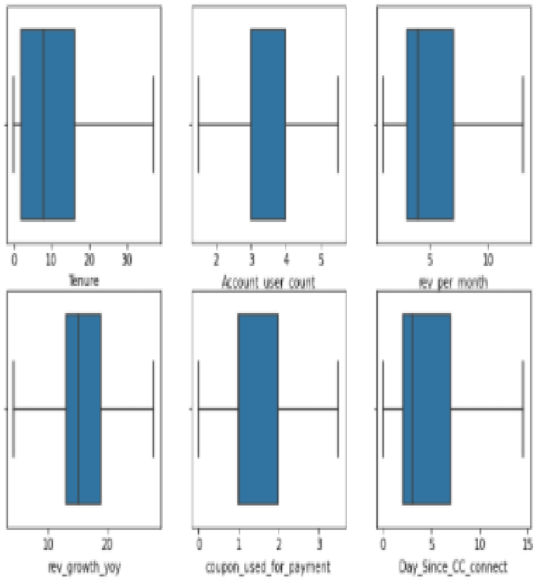


Fig:20 Boxplot

Figure:21 Boxplot (Outliers Removed)

Variable transformation:

Before applying machine learning techniques, we need to pre-process our data by converting all the categorical variables to numerical variables. To perform that I am using **Feature Engineering technique**. It will convert all the categorical Features into unique codes.

```
feature: City_Tier
[3, 1, '1.0', 2]
Categories (4, object): [1, 2, 3, '1.0']
[2 0 3 1]


feature: Payment
['Debit Card', 'UPI', 'Credit Card', 'Cash on Delivery', 'E wallet']
Categories (5, object): ['Cash on Delivery', 'Credit Card', 'Debit Card', 'E wallet', 'UPI']
[2 4 1 0 3]


feature: Gender
['Female', 'Male']
Categories (2, object): ['Female', 'Male']
[0 1]


feature: Service_Score
[3, 2, 1, '3.0', 0, 4, 5]
Categories (7, object): [0, 1, 2, 3, 4, 5, '3.0']
[3 2 1 6 0 4 5]


feature: account_segment
['Super', 'Regular Plus', 'Regular', 'HNI', 'Super Plus']
Categories (5, object): ['HNI', 'Regular', 'Regular Plus', 'Super', 'Super Plus']
[3 2 1 0 4]


feature: CC_Agent_Score
[2, 3, 5, 4, '3.0', 1]
Categories (6, object): [1, 2, 3, 4, 5, '3.0']
[1 2 4 3 5 0]


feature: Marital_Status
['Single', 'Divorced', 'Married']
Categories (3, object): ['Divorced', 'Married', 'Single']
[2 0 1]


feature: Complain_ly
[1, 0, '0.0']
Categories (3, object): [0, 1, '0.0']
[1 0 2]


feature: Login_device
['Mobile', 'Computer']
Categories (2, object): ['Computer', 'Mobile']
[1 0]
```

Fig:22 Encoded Features

- **Model building - Clear on why was a particular model(s) chosen. - Effort to improve model performance.**

Ans. Now I am ready to apply Predictive Models on my both train and test datasets. The problem is a classification problem so I will give priority to the models such as

- Decision Tree
- Random Forest
- K-Nearest Neighbours
- Bagging
- Ada-Boost
- Gradient-Boost
- Logistic-Regression
- LDA
- Naïve Bays

Before applying Machine learning technique, I need to understand **Confusion Matrix.**

	Actual value 1	**Actual value 0**
Predicted value 1	**True Positive**	**False Positive**
Predicted value 0	**False Negative**	**True Negative**

true positives (TP): These are cases in which I predicted yes **(Churn=1)**, and customers **churns**.
true negatives (TN): I predicted **(Churn=0)**, and they do not **churn**.
false positives (FP): I predicted **(Churn=1)**, but customers do not **churn**. **(Also known as a "Type I error.")**
false negatives (FN): I predicted **(Churn=0)**, but customer **churns**. **(Also known as a "Type II error.")**

It is a classification problem so I will give importance to

- Decision Tree
- Random Forest
- K-Nearest Neighbours
- Bagging
- Ada-Boost
- Gradient-Boost

K-Nearest Neighbours:

Results:

```
[0.04203670811130844,
 0.05476613380698636,
 0.06305506216696266,
 0.07519242155121375,
 0.08496151515156897573,
 0.09532267613972767,
 0.10213143872113672,
 0.10834813499111906,
 0.11130846654825344,
 0.11841326228537596]
```

Fig:23 Misclassification Error Values for specific K-Values

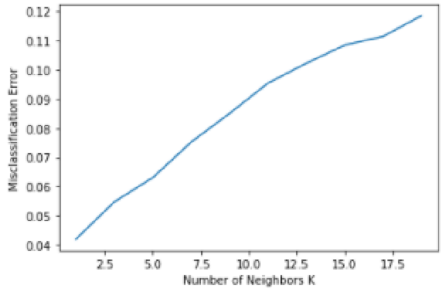


Fig:24 Misclassification Error Values forspecific K-Values graph

```
1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6555
           1       1.00      1.00      1.00      1327

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

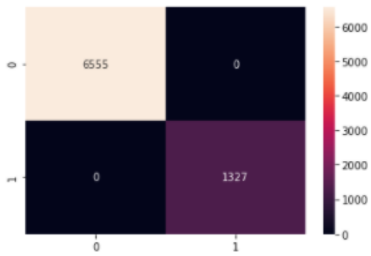


Fig:25 Train Performance and ConfusionMatrix

```
0.9579632918886916
              precision    recall  f1-score   support

           0       0.98      0.97      0.97      2809
           1       0.87      0.88      0.88       569

    accuracy                           0.96      3378
   macro avg       0.92      0.93      0.93      3378
weighted avg       0.96      0.96      0.96      3378
```

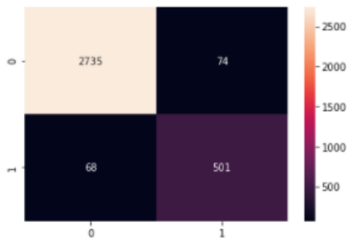


Fig:26 Test Performance and ConfusionMatrix

```
AUC: 1.000
[<matplotlib.lines.Line2D at 0x1c654d7c3a0>]
```

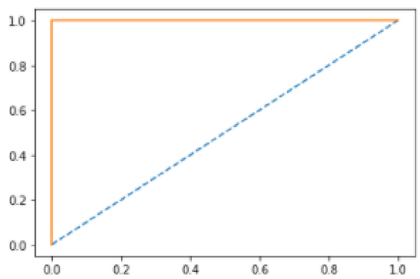


Fig:27 Train ROC-AUC Curve

```
AUC: 0.927
[<matplotlib.lines.Line2D at 0x1c654e45ac0>]
```

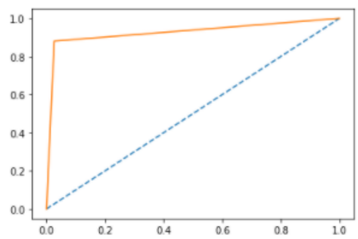


Fig:28 Test ROC-AUC Curve

Dataset	Recall	f-1 score	ROC-AUC
Train Data	1.00	1.00	1.00
Test Data	0.88	0.88	0.93

Decision Tree:

Results:

```
0.8919056077137782
              precision    recall  f1-score   support

           0       0.92      0.95      0.94      6555
           1       0.72      0.58      0.65      1327

    accuracy                           0.89      7882
   macro avg       0.82      0.77      0.79      7882
weighted avg       0.89      0.89      0.89      7882
```

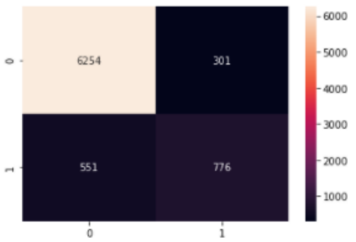


Fig:29 Train Performance and ConfusionMatrix

```
0.8845470692717584
              precision    recall  f1-score   support

           0       0.91      0.95      0.93      2809
           1       0.70      0.56      0.62       569

    accuracy                           0.88      3378
   macro avg       0.81      0.75      0.78      3378
weighted avg       0.88      0.88      0.88      3378
```

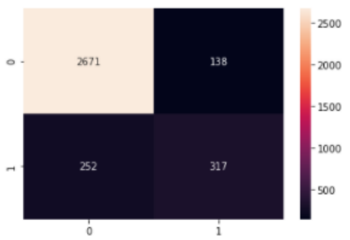


Fig:30 Test Performance and ConfusionMatrix

```
AUC: 0.929
[<matplotlib.lines.Line2D at 0x1c654f36a90>]
```

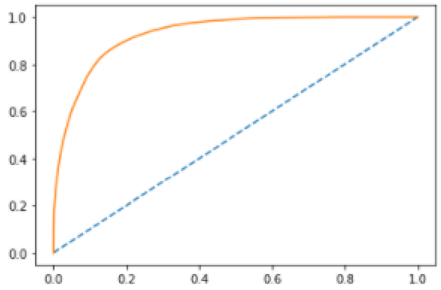


Fig:31 Train ROC-AUC Curve

```
AUC: 0.906
[<matplotlib.lines.Line2D at 0x1c6550041f0>]
```

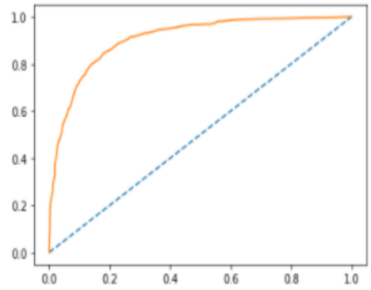


Fig:32 Test ROC-AUC Curve

Dataset	Recall	f-1 score	ROC-AUC
Train Data	0.58	0.65	0.93
Test Data	0.56	0.62	0.91

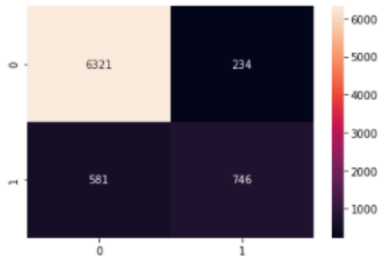
Random Forest Classifier:

Results:

```
0.8965998477543771
              precision    recall  f1-score   support

           0       0.92      0.96      0.94      6555
           1       0.76      0.56      0.65      1327

    accuracy                           0.90      7882
   macro avg       0.84      0.76      0.79      7882
weighted avg       0.89      0.90      0.89      7882
```




```
0.8919478981645944
              precision    recall  f1-score   support

           0       0.91      0.97      0.94      2809
           1       0.76      0.52      0.62       569

    accuracy                           0.89      3378
   macro avg       0.84      0.74      0.78      3378
weighted avg       0.88      0.89      0.88      3378
```

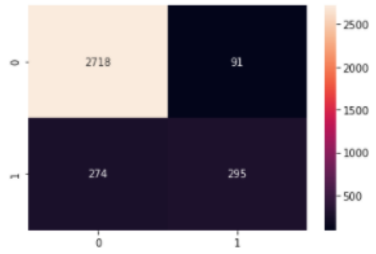
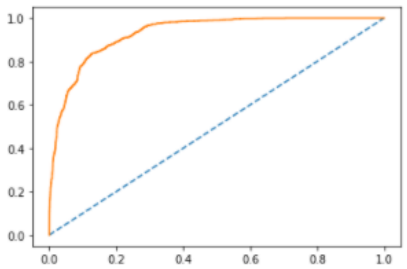


Fig:33 Train Performance and ConfusionMatrix

Fig:34 Test Performance and ConfusionMatrix

```
AUC: 0.931
[<matplotlib.lines.Line2D at 0x1c654b88310>]
```




```
AUC: 0.916
[<matplotlib.lines.Line2D at 0x1c654fc6640>]
```

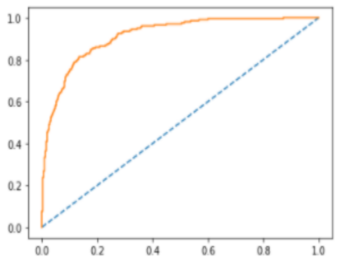


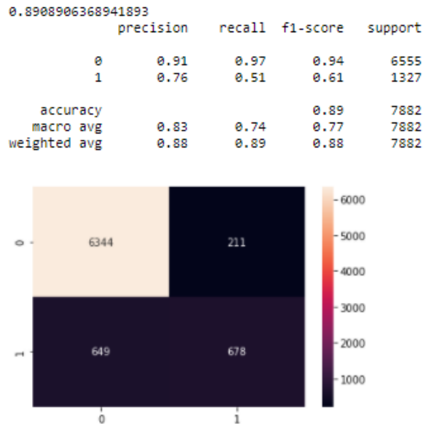
Fig:35 Train ROC-AUC Curve

Fig:36 Test ROC-AUC Curve

Dataset	Recall	f-1 score	ROC-AUC
Train Data	0.56	0.65	0.93
Test Data	0.52	0.62	0.91

Bagging:

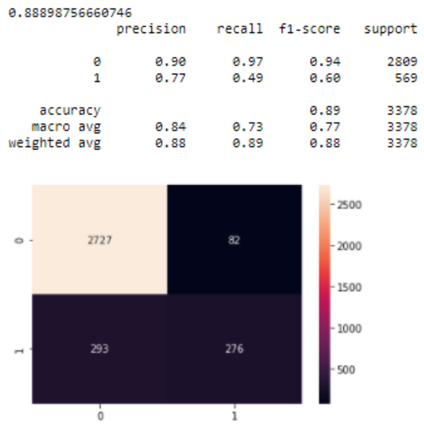
Result:



```
0.8908906368941893
              precision    recall  f1-score   support

           0       0.91      0.97      0.94      6555
           1       0.76      0.51      0.61      1327

    accuracy                           0.89      7882
   macro avg       0.83      0.74      0.77      7882
weighted avg       0.88      0.89      0.88      7882
```

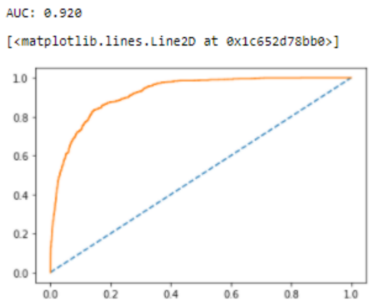
```
0.88898756660746
              precision    recall  f1-score   support

           0       0.90      0.97      0.94      2809
           1       0.77      0.49      0.60       569

    accuracy                           0.89      3378
   macro avg       0.84      0.73      0.77      3378
weighted avg       0.88      0.89      0.88      3378
```


Fig:37 Train Performance and ConfusionMatrix

Fig:38 Test Performance and ConfusionMatrix



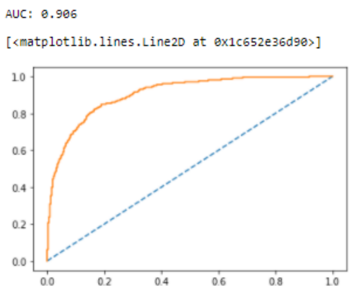


Fig:39 Train ROC-AUC Curve

Fig:40 Test ROC-AUC Curve

Dataset	Recall	f-1 score	ROC-AUC
Train Data	0.51	0.61	0.92
Test Data	0.49	0.60	0.91

From the results the Recall and f-1score is low.

Ada Boost:

Result:

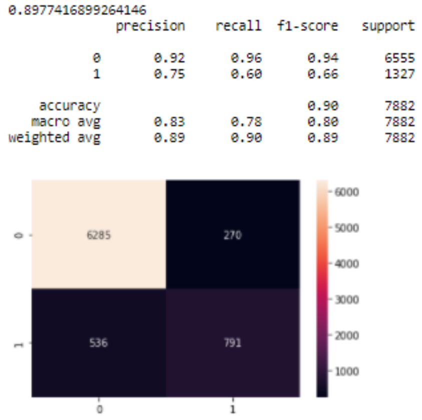


Fig:41 Train Performance and ConfusionMatrix

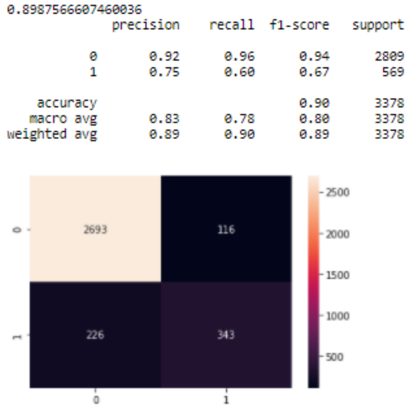


Fig:42 Test Performance and ConfusionMatrix

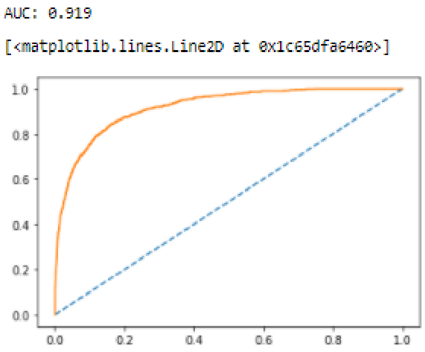


Fig:43 Train ROC-AUC Curve

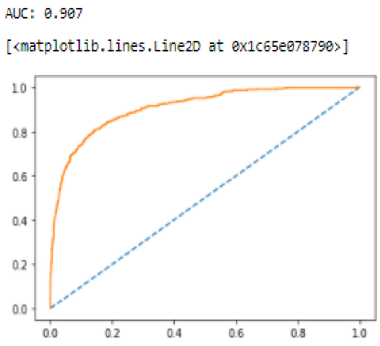


Fig:44 Test ROC-AUC Curve

Dataset	Recall	f-1 score	ROC-AUC
Train Data	0.60	0.66	0.92
Test Data	0.60	0.67	0.91

From the results the Recall and f-1score is low.

Gradient Boost:

Result:

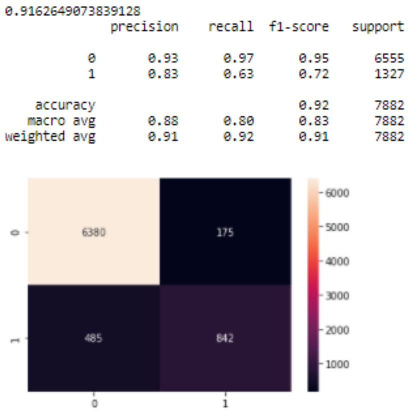


Fig:45 Train Performance and ConfusionMatrix

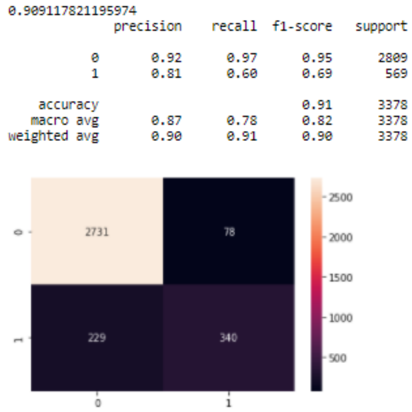


Fig:46 Test Performance and ConfusionMatrix

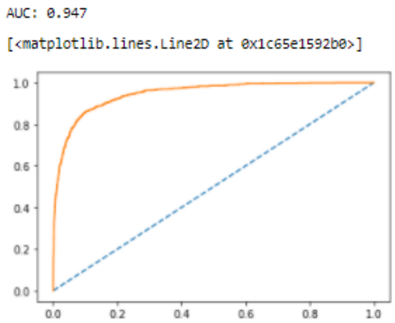


Fig:47 Train ROC-AUC Curve

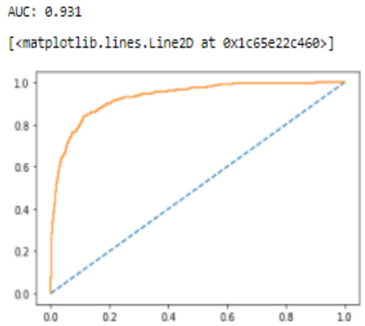


Fig:48 Test ROC-AUC Curve

Dataset	Recall	f-1 score	ROC-AUC
Train Data	0.63	0.72	0.95
Test Data	0.60	0.69	0.93

From the results the Recall and f-1score is low.

model tuning measures:

I used the technique called **Model Tuner (tunning the hyper parameters)** it increases the Model performance. I am applying **gridSearchCV** from **SKlearn.**

It improves the **Recall** and **f1score** of the models.

Results:

	Train_Recall	Test_Recall	Train_F1_score	Test_F1_score	Train_precision	Test_precision
KNN	1.00	0.88	1.00	0.88	1.00	0.87
DECISION_TREE	0.58	0.56	0.65	0.62	0.72	0.70
RANDOM_FOREST	0.56	0.52	0.65	0.62	0.76	0.76
BAGGING	0.51	0.49	0.61	0.60	0.76	0.77
ADABOOST	0.60	0.60	0.66	0.67	0.75	0.75
GRADIENT_BOOST	0.63	0.60	0.72	0.69	0.83	0.81

From the results the KNN showing best result but **Train_Recall** = 1.00, **Triain_f1_score** = 1.00 and **Train_Precision** = 1.00 which means the model is overfitted

I use **SMOTE** to clear this problem.

Results:

```
0.909117821195974
              precision    recall  f1-score   support

           0       0.99      0.83      0.90      6555
           1       0.85      0.99      0.92      6555

    accuracy                           0.91     13110
   macro avg       0.92      0.91      0.91     13110
weighted avg       0.92      0.91      0.91     13110
```

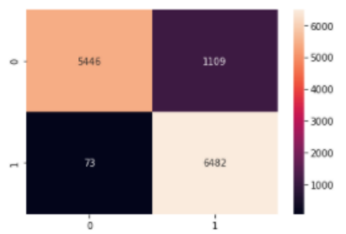


Fig:49 Train Performance and ConfusionMatrix

```
0.909117821195974
              precision    recall  f1-score   support

           0       0.98      0.80      0.89      2809
           1       0.49      0.93      0.64       569

    accuracy                           0.83      3378
   macro avg       0.74      0.87      0.77      3378
weighted avg       0.90      0.83      0.84      3378
```

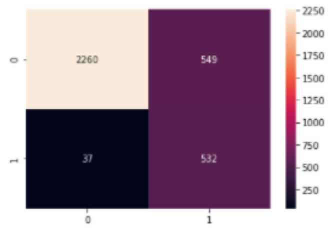
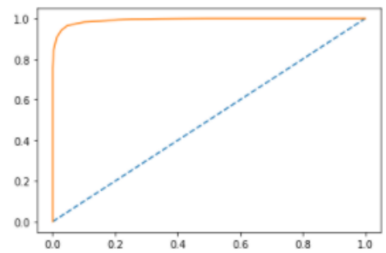


Fig:50 Test Performance and ConfusionMatrix

```
AUC: 0.993
[<matplotlib.lines.Line2D at 0x1c65e814a90>]
```



FiFig:51 Train ROC-AUC Curve

```
AUC: 0.957
[<matplotlib.lines.Line2D at 0x1c65f8c64f0>]
```

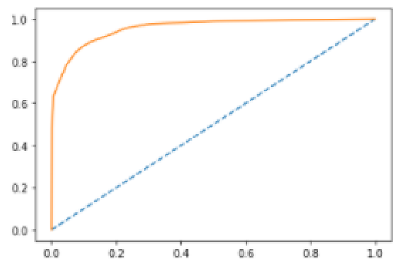


Fig:52 Test ROC-AUC Curve

Dataset	Recall	f-1 score	ROC-AUC
Train Data	0.99	0.92	0.99
Test Data	0.93	0.64	0.96

- **Model validation - How was the model validated? Just accuracy, or anything else too?**

Ans. To check whether the model is valid for this problem or not I will apply the models on test data and check the parameters.

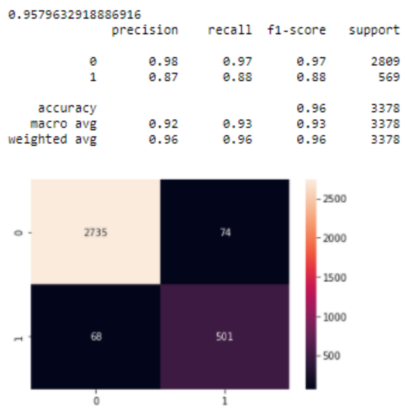


Fig:53 KNN

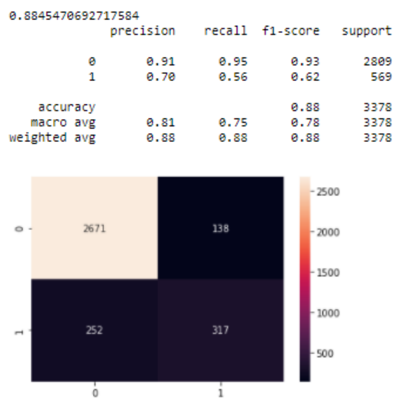


Fig:54 Decision Tree

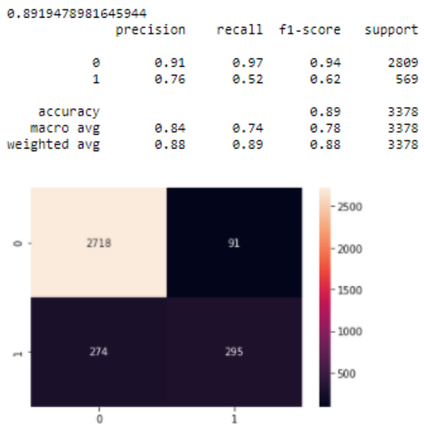


Fig:55 Random Forest

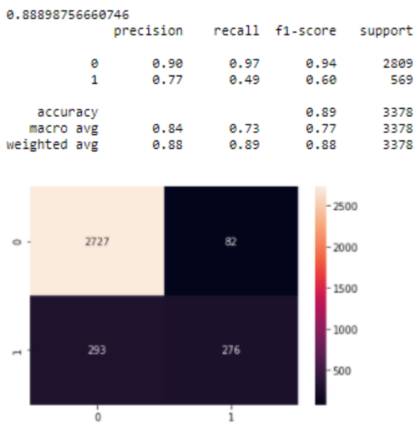


Fig:56 Bagging

```
0.8987566607460036
              precision    recall  f1-score   support

           0       0.92      0.96      0.94      2809
           1       0.75      0.60      0.67       569

    accuracy                           0.90      3378
   macro avg       0.83      0.78      0.80      3378
weighted avg       0.89      0.90      0.89      3378
```

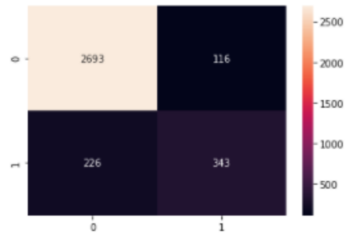


Fig: 57 ADA Boost

```
0.909117821195974
              precision    recall  f1-score   support

           0       0.92      0.97      0.95      2809
           1       0.81      0.60      0.69       569

    accuracy                           0.91      3378
   macro avg       0.87      0.78      0.82      3378
weighted avg       0.90      0.91      0.90      3378
```

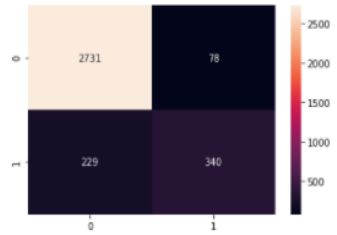


Fig:58 Gradient Boost

	Train_Recall	Test_Recall	Train_F1_score	Test_F1_score	Train_precision	Test_precision
KNN	1.00	0.88	1.00	0.88	1.00	0.87
DECISION_TREE	0.58	0.56	0.65	0.62	0.72	0.70
RANDOM_FOREST	0.56	0.52	0.65	0.62	0.78	0.76
BAGGING	0.51	0.49	0.61	0.60	0.78	0.77
ADABOOST	0.60	0.60	0.66	0.67	0.75	0.75
GRADIENT_BOOST	0.63	0.60	0.72	0.69	0.83	0.81

From the results the KNN showing best result but **Train_Recall** = 1.00, **Triain_f1_score** = 1.00 and **Train_Precision** = 1.00 which means the model is overfitted

I use **SMOTE** to clear this problem.

Results:

```
0.909117821195974
              precision    recall  f1-score   support

           0       0.99      0.83      0.90      6555
           1       0.85      0.99      0.92      6555

    accuracy                           0.91     13110
   macro avg       0.92      0.91      0.91     13110
weighted avg       0.92      0.91      0.91     13110
```

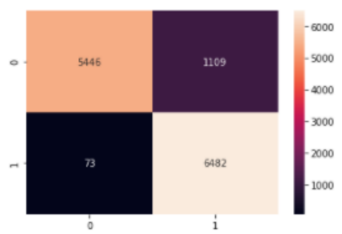


Fig:59 Train Performance and ConfusionMatrix

```
0.909117821195974
              precision    recall  f1-score   support

           0       0.98      0.80      0.89      2809
           1       0.49      0.93      0.64       569

    accuracy                           0.83      3378
   macro avg       0.74      0.87      0.77      3378
weighted avg       0.90      0.83      0.84      3378
```

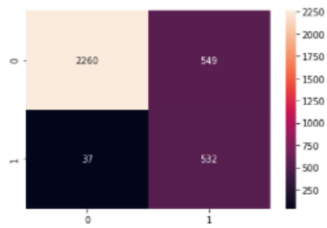
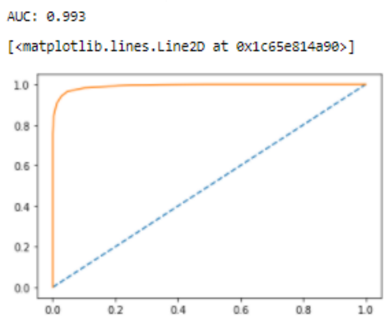
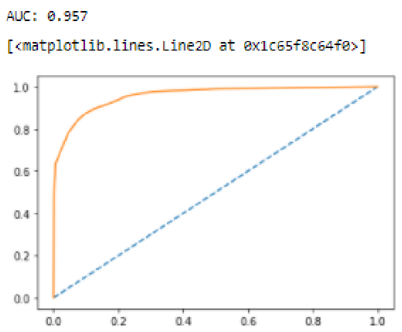


Fig:60 Test Performance and ConfusionMatrix





FiFig:61 Train ROC-AUC Curve **Fig:62 Test ROC-AUC Curve**

Dataset	Recall	f-1 score	ROC-AUC
Train Data	0.99	0.92	0.99
Test Data	0.93	0.64	0.96

From the result **KNN(SMOTE)** is the best model for this classification problem and it is the most valid model.

- **Final interpretation / recommendation - Very clear and crisp on what recommendations do you want to give to the management / client.**

Ans. From **OutPut-4, 0=9364 & 1=1896 so the** difference between datapoints **Churn=1**and **Churn=0** is very high which indicates that the dataset is highly unbalanced, and it may lead to a very high no. of false positive results. That is why I used **ENSEMBLE TECHNIQUES** & **SMOTE**

- As the dataset is imbalanced so I will investigate both Accuracy and Performance table.
- From Accuracy table it is clear that '**Decision Tree**', '**Random Forest**' and '**KNN**' are showing best results.
- Now I will move to **Performance Table** to get some deeper insights.

$$(10.1)\quad Accuracy = \frac{T_p + T_n}{T_p + T_n + F_p + F_n}$$

$$(10.2)\quad Precision = \frac{T_p}{T_p + F_p}$$

$$(10.3)\quad Recall = \frac{T_p}{T_p + T_n}$$

$$(10.4)\quad F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

	Train_Recall	Test_Recall	Train_F1_score	Test_F1_score	Train_precision	Test_precision
KNN	1.00	0.88	1.00	0.88	1.00	0.87
DECISION_TREE	0.58	0.56	0.65	0.62	0.72	0.70
RANDOM_FOREST	0.56	0.52	0.65	0.62	0.76	0.76
BAGGING	0.51	0.49	0.61	0.60	0.76	0.77
ADABOOST	0.60	0.60	0.66	0.67	0.75	0.75
GRADIENT_BOOST	0.63	0.60	0.72	0.69	0.83	0.81

From the results the KNN showing best result but **Train_Recall** = 1.00, **Triain_f1_score** = 1.00 and **Train_Precision** = 1.00 which means the model is overfitted

I use **SMOTE** to clear this problem.

Dataset	Recall	f-1 score	ROC-AUC
Train Data	0.99	0.92	0.99
Test Data	0.93	0.64	0.96

Business Insights:

1. **Test_Recall(SMOTE)= 0.93** , **Test_f1_score(SMOTE)=0.64** and **Test_AUC-ROC(SMOTE)=0.96** it means it is **93% accurately** filtering out the Potential Churners.
2. The company can segregate the Potential Churners from the customers list with **93% accuracy**. Company can design packages for a specific customer with a confidence of **93% accuracy** that he or she might Churn.
3. It will also help the company to design products based on customer demand for future purposes. Company will successfully retain existing customers, which is the main purpose.
4. In the end the company will generate a good revenue.

Other Business Recommendations:

- Check Demographics and behavioural data. Is this user a single user or using your product on behalf of their company?
- Check Revenue information for each customer.
- Check Contract terms.
- Check High number of customers **churning** after sign-up.
- Check Long-time customer **churn**.
- Check Frequent **churn** spikes following product updates.