Capstone Project

Customer Churn – E-Commerce Business

TEJAS PADEKAR
PGP-DSBA
Date: 05/02/2023

CONTENTS:

List of Figures

1. Introduction

a) Defining problem statement

Problem:

An E Commerce company provider is facing a lot of competition in the current market and it has become a challenge to retain the existing customers in the current situation. Hence, the company wants to develop a model through which they can do churn prediction of the accounts and provide segmented offers to the potential churners. In this company, account churn is a major thing because 1 account can have multiple customers. hence by losing one account the company might be losing more than one customer.

Problem Statement:

We have been assigned to develop a churn prediction model for this company and provide business recommendations on the campaign. Our campaign suggestion should be unique and be very clear on the campaign offer because our recommendations will go through the revenue assurance team. If they find that we are giving a lot of free (or subsidized) stuff thereby making a loss to the company; they are not going to approve our recommendations. Hence we need to be very careful while providing campaign recommendations.

b) Need of the study/project

Customer Churn is one of the biggest global problems faced by businesses across various industries which also includes the e-commerce service industry. It is an extremely difficult problem to tackle due to intense competition in the E-commerce industry as E-Commerce business has no entry barriers, works on a light business model, can be easily replicated and has a diverse customer base having difference in generations, different income groups, different geographies with different customs and traditions, different likes and dislikes etc.

Moreover, affordable data services and internet connectivity along with more awareness of information has made customers smarter and more conscious. Thus, impacting their decision making when choosing their loyalty towards a particular brand or business.

Rapidly changing business environment and technological advancements create many opportunities for e-commerce businesses to draft and roll out plans to improve service quality, provide quick delivery solutions, resolve queries quickly, track and collect customer information, track and collect competition information, create ideas and innovations which can help them in customer retention.

In our case, one account has multiple users. Hence, it becomes even more important to focus on retention as churning would impact revenue and profits tremendously. Also, generally customer retention is less expensive for any business than customer acquisition. Hence, it is both necessary as well as beneficial/profitable. It provides stability for any business. Keeps investors and shareholders happy and helps business gain their trust. Gaining trust enables businesses to innovate and explore growth and expansion plans. Customer retention is also necessary for maintaining continued demand and even market gain and it also helps in brand building and brand awareness.

Hence, with this study we aim to deliver a reliable Customer Churn Prediction model which can identify such customers which are likely to churn well in advance so that the company can try and retain them. Also, we would provide insights and recommendations, which help them identify issues and take necessary actions to achieve this objective of retention.

c) Data Report

Data Dictionary

- **AccountID –** Unique Account ID of primary holder.
- **Churn –** Binary Target Variable where 0 within the row represents accounts which have not churned and 1 represents accounts which have churned.
- **Tenure** – Period since primary holder is a customer of the company in months.
- **City_Tier –** Primary customer's city tier from 1, 2 and 3.
- **CC_Contacted_LY –** No of times all the customers of the account has contacted customer care in last 12 months.
- **Payment –** Preferred Payment mode of the customers in the account.
- **Gender** – Gender of the primary customer.
- **Service_Score –** Satisfaction score given by customers to the E-commerce company.
- **Account_user_count –** Number of customers tagged with an account.
- **account_segment –** Account segmentation on the basis of spend.
- **CC_Agent_Score –** Satisfaction score given by customers to customer care service.
- **Marital_Status –** Marital status of the primary customer of the account.
- **rev_per_month –** Monthly average revenue generated by account in last 12 months.
- **Complain_ly –** If complaint has been raised by account in last 12 months
- **rev_growth_yoy –** revenue growth percentage of the account (last 12 months vs last 13 to 24 to month).
- **coupon_used_for_payment –** How many times customers have used coupons to do the payment in last 12 months.
- **Day_Since_CC_connect –** Number of days since none of the customers from an account has contacted the customer care.
- **Cashback –** Monthly average cashback generated by account in last 12 months.
- **Login_device –** Preferred login device of the customers in the account.

	AccountID	Churn	Tenure	City_Tier	CC_Contacted_LY	Payment	Gender	Service_Score	Account_user_count	account_segment	CC_Agent_Score
0	20000	1	4	3.0	6.0	Debit Card	Female	3.0	3	Super	2.0
1	20001	1	0	1.0	8.0	UPI	Male	3.0	4	Regular Plus	3.0
2	20002	1	0	1.0	30.0	Debit Card	Male	2.0	4	Regular Plus	3.0
3	20003	1	0	3.0	15.0	Debit Card	Male	2.0	4	Super	5.0
4	20004	1	0	1.0	12.0	Credit Card	Male	2.0	3	Regular Plus	5.0

Figure 1: Dataset Head (1 to 11 variables and top 5 rows)

Marital_Status	rev_per_month	Complain_ly	rev_growth_yoy	coupon_used_for_payment	Day_Since_CC_connect	cashback	Login_device
Single	9	1.0	11	1	5	159.93	Mobile
Single	7	1.0	15	0	0	120.9	Mobile
Single	6	1.0	14	0	3	NaN	Mobile
Single	8	0.0	23	0	3	134.07	Mobile
Single	3	0.0	11	1	3	129.6	Mobile

Figure 2: Dataset Head (12 to 19 variables and top 5 rows)

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
AccountID	11260.00	NaN	NaN	NaN	25629.50	3250.63	20000.00	22814.75	25629.50	28444.25	31259.00
Churn	11260.00	NaN	NaN	NaN	0.17	0.37	0.00	0.00	0.00	0.00	1.00
Tenure	11158.00	38.00	1.00	1351.00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
City_Tier	11148.00	NaN	NaN	NaN	1.65	0.92	1.00	1.00	1.00	3.00	3.00
CC_Contacted_LY	11158.00	NaN	NaN	NaN	17.87	8.85	4.00	11.00	16.00	23.00	132.00
Payment	11151	5	Debit Card	4587	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Gender	11152	4	Male	6328	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Service_Score	11162.00	NaN	NaN	NaN	2.90	0.73	0.00	2.00	3.00	3.00	5.00
Account_user_count	11148.00	7.00	4.00	4569.00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
account_segment	11163	7	Super	4062	NaN	NaN	NaN	NaN	NaN	NaN	NaN
CC_Agent_Score	11144.00	NaN	NaN	NaN	3.07	1.38	1.00	2.00	3.00	4.00	5.00
Marital_Status	11048	3	Married	5860	NaN	NaN	NaN	NaN	NaN	NaN	NaN
rev_per_month	11158.00	59.00	3.00	1746.00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Complain_ly	10903.00	NaN	NaN	NaN	0.29	0.45	0.00	0.00	0.00	1.00	1.00
rev_growth_yoy	11260.00	20.00	14.00	1524.00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
coupon_used_for_payment	11260.00	20.00	1.00	4373.00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Day_Since_CC_connect	10903.00	24.00	3.00	1816.00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
cashback	10789.00	5693.00	155.62	10.00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Login_device	11039	3	Mobile	7482	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Figure 3: Data Description

- Dataset contains 11260 rows with a total of 18 predictor variables and 1 binary target variable. The predictor variables are a mix of categorical and continuous variables where some of continuous variables such as City_Tier, Service_Score, CC_Agent_Score and Complain_ly are already provided in encoded format.
- Observations across variables are in different scales such as text, decimal values, numbers and percentages.
- There are inconsistencies in the data (M, Male, Premium Plus, Premium +) and presence of bad data with incorrect characters assigned such as $, +, 99, #, etc. Hence, will need to perform data preprocessing and cleaning. It is recommended that the company maintain a more standardized, uniform and accurate data as much as possible to achieve best results.

```
RangeIndex: 11260 entries, 0 to 11259
Data columns (total 19 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   AccountID                11260 non-null  int64
 1   Churn                    11260 non-null  int64
 2   Tenure                   11158 non-null  object
 3   City_Tier                11148 non-null  float64
 4   CC_Contacted_LY          11158 non-null  float64
 5   Payment                  11151 non-null  object
 6   Gender                   11152 non-null  object
 7   Service_Score            11162 non-null  float64
 8   Account_user_count       11148 non-null  object
 9   account_segment          11163 non-null  object
 10  CC_Agent_Score           11144 non-null  float64
 11  Marital_Status           11048 non-null  object
 12  rev_per_month            11158 non-null  object
 13  Complain_ly              10903 non-null  float64
 14  rev_growth_yoy           11260 non-null  object
 15  coupon_used_for_payment  11260 non-null  object
 16  Day_Since_CC_connect     10903 non-null  object
 17  cashback                 10789 non-null  object
 18  Login_device             11039 non-null  object
dtypes: float64(5), int64(2), object(12)
```

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

Figure 4: Data Information with Missing Value Count for each variable

- AccountID variable is not of much significance for prediction. Hence, we will be dropping it.
- City_Tier, Service_Score, CC_Agent_Score and Complain_ly are all actually Categorical variables which have been encoded already and hence, they are of 'float' data type and it will not make sense to analyse the measures of central tendencies such as mean, median or mode for the same.
- Actual categorical variables are 10: Churn, City_Tier, Payment, Gender, Service_Score, account_segment, CC_Agent_Score, Marital_Status, Complain_ly and Login_device.
- Actual Numeric variables are 9: AccountID, Tenure, CC_Contacted_LY, Account_user_count, rev_per_month, rev_growth_yoy, coupon_used_for_payment, Day_Since_CC_connect and cashback.
- There are no duplicate values. However, there are missing values and null values.
- Apart from the variables of 'AccountID', 'Churn' (Target Variable), 'rev_growth_yoy' and 'coupon_used_for_payment' all other variables have null values where ‘cashback’ has the highest missing values (471).
- Once, we have imputed the missing values and Nan values, transformed the data set with correct variable data type, we would analyse the variables again.

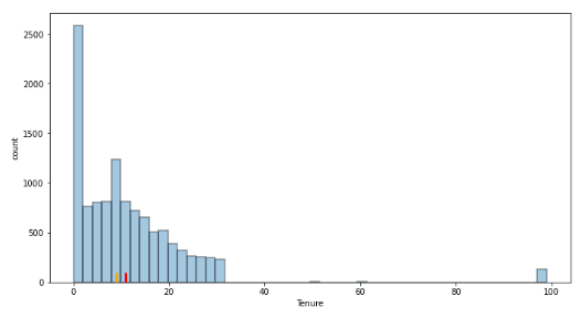
2. Exploratory Data Analysis and Business Implication

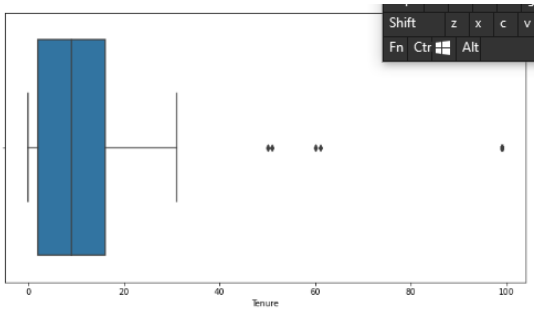
a) Univariate analysis

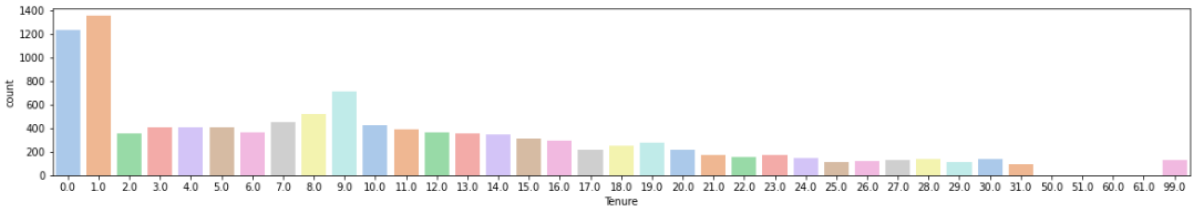
i. Univariate analysis of continuous variables (This analysis is post imputing the missing and null values to make the report more relevant)

	count	mean	std	min	25%	50%	75%	max
AccountID	11260.00	25629.50	3250.63	20000.00	22814.75	25629.50	28444.25	31259.00
Churn	11260.00	0.17	0.37	0.00	0.00	0.00	0.00	1.00
Tenure	11260.00	10.99	12.76	0.00	2.00	9.00	16.00	99.00
City_Tier	11260.00	1.65	0.91	1.00	1.00	1.00	3.00	3.00
CC_Contacted_LY	11260.00	17.85	8.81	4.00	11.00	16.00	23.00	132.00
Service_Score	11260.00	2.90	0.72	0.00	2.00	3.00	3.00	5.00
Account_user_count	11260.00	3.70	1.00	1.00	3.00	4.00	4.00	6.00
CC_Agent_Score	11260.00	3.07	1.37	1.00	2.00	3.00	4.00	5.00
rev_per_month	11260.00	6.27	11.49	1.00	3.00	5.00	7.00	140.00
Complain_ly	11260.00	0.28	0.45	0.00	0.00	0.00	1.00	1.00
rev_growth_yoy	11260.00	16.19	3.76	4.00	13.00	15.00	19.00	28.00
coupon_used_for_payment	11260.00	1.79	1.97	0.00	1.00	1.00	2.00	16.00
Day_Since_CC_connect	11260.00	4.58	3.65	0.00	2.00	3.00	7.00	47.00
cashback	11260.00	194.93	174.98	0.00	147.89	165.25	197.31	1997.00

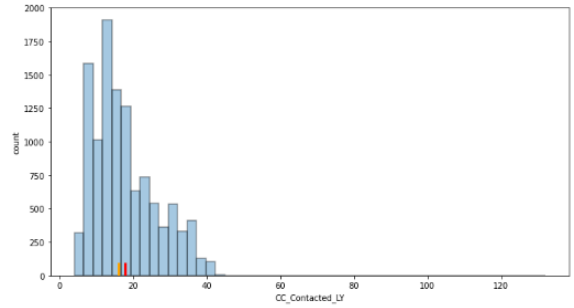
Tenure
Skew: 3.94

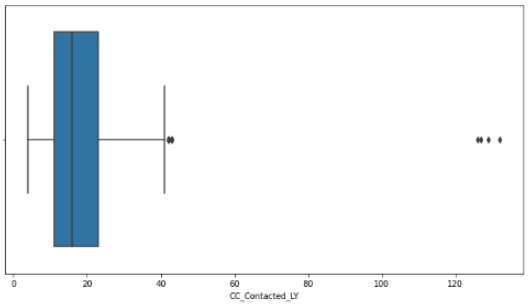


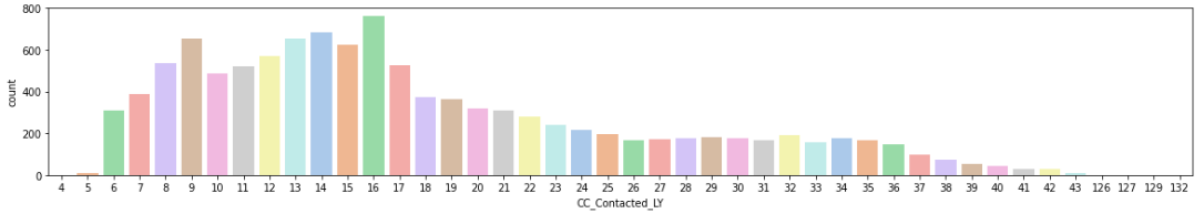




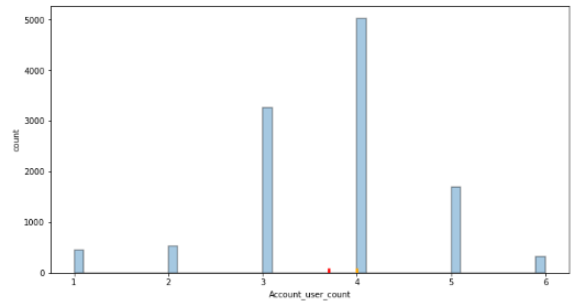
CC_Contacted_LY
Skew: 1.43

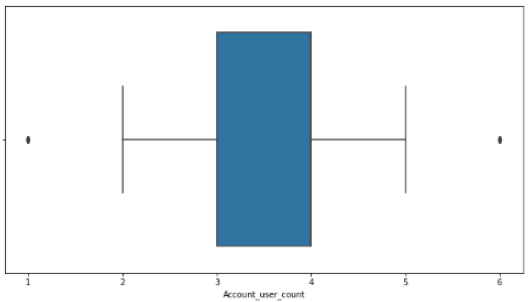


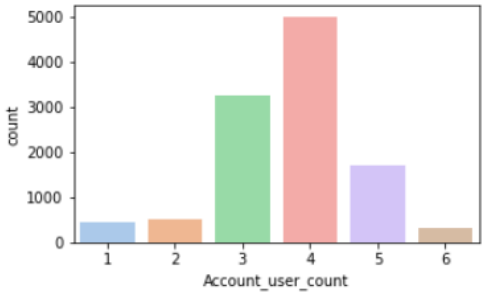




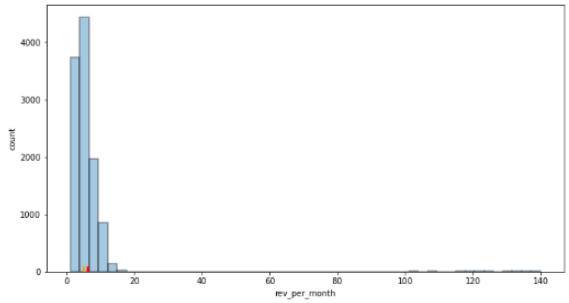
Account_user_count
Skew: -0.43

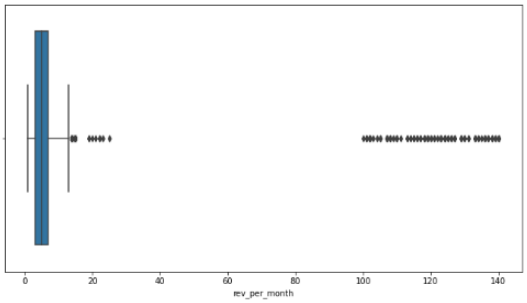


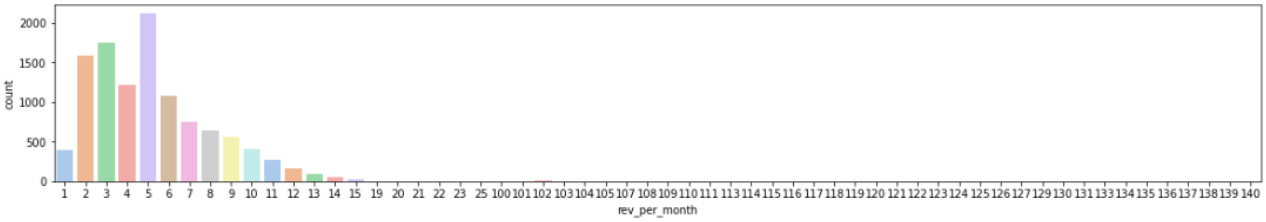




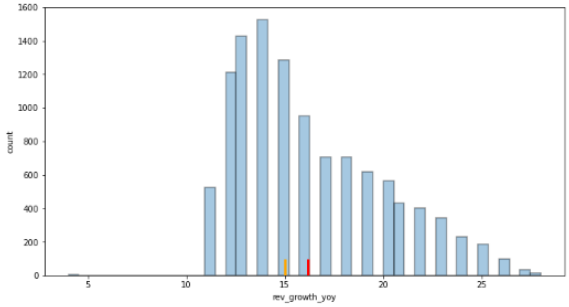
rev_per_month
Skew: 9.44

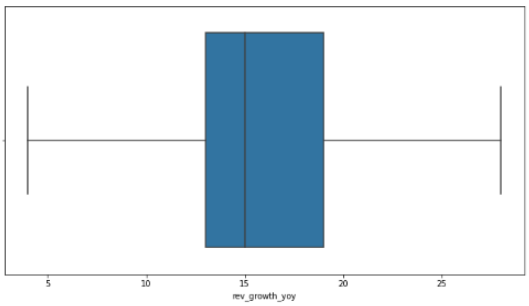


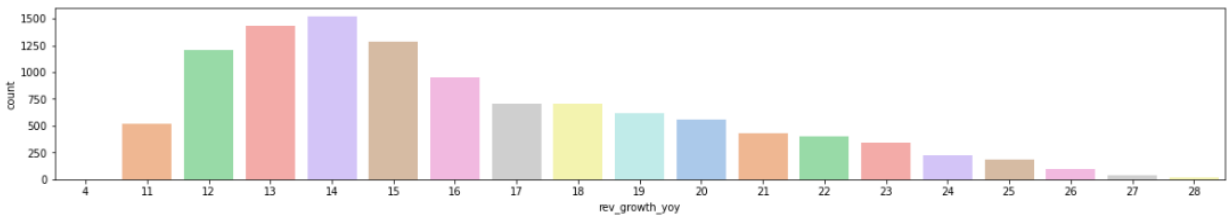




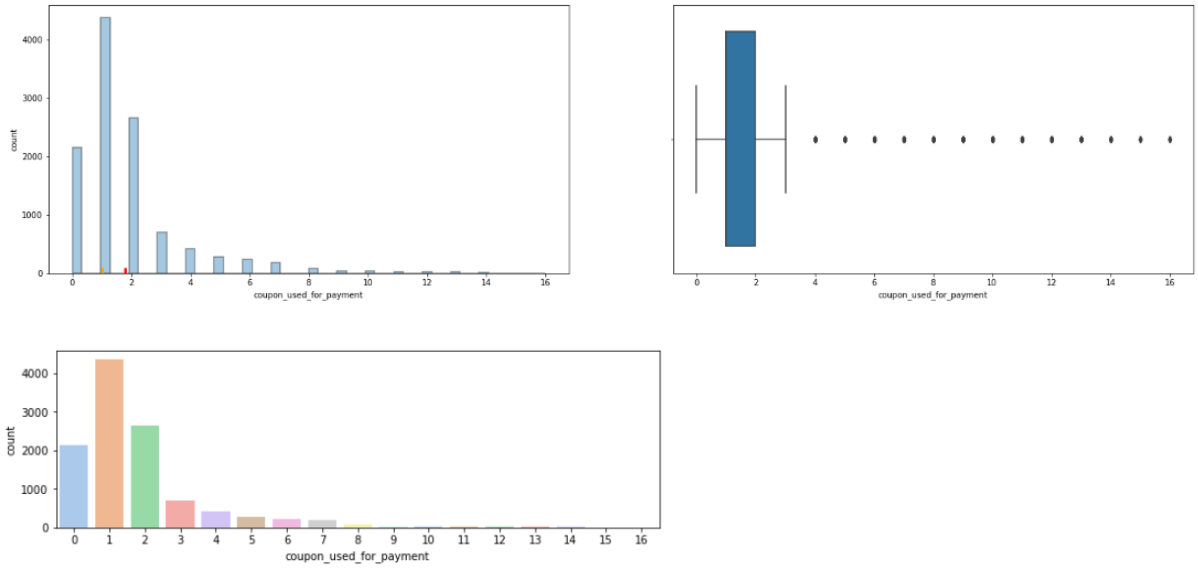
rev_growth_yoy
Skew: 0.75



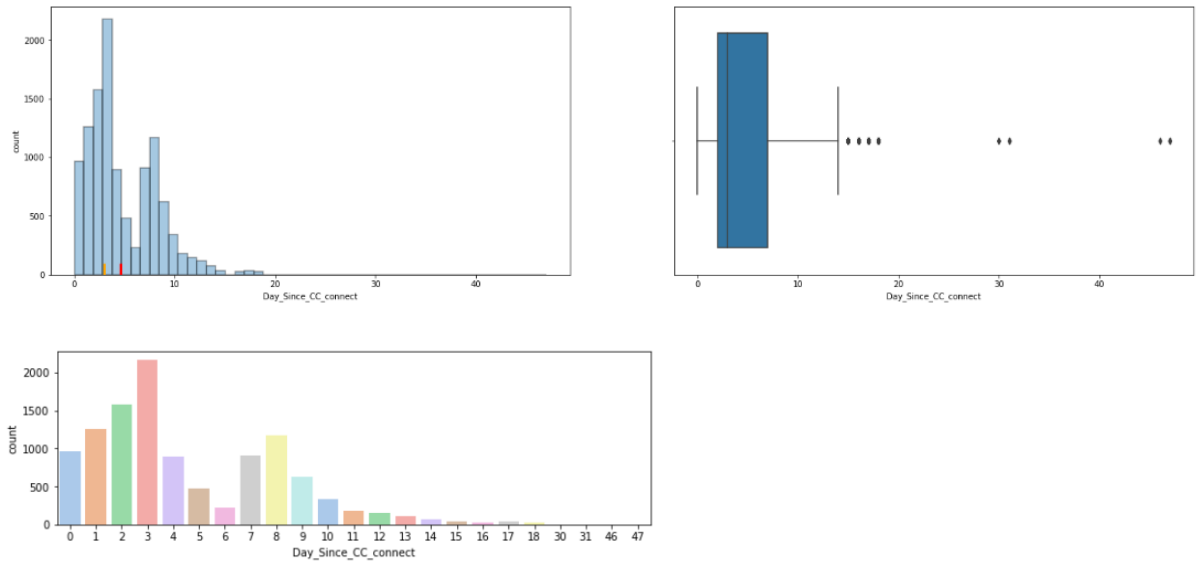




coupon_used_for_payment
Skew: 2.58



Day_Since_CC_connect
Skew: 1.32



cashback
Skew: 8.97

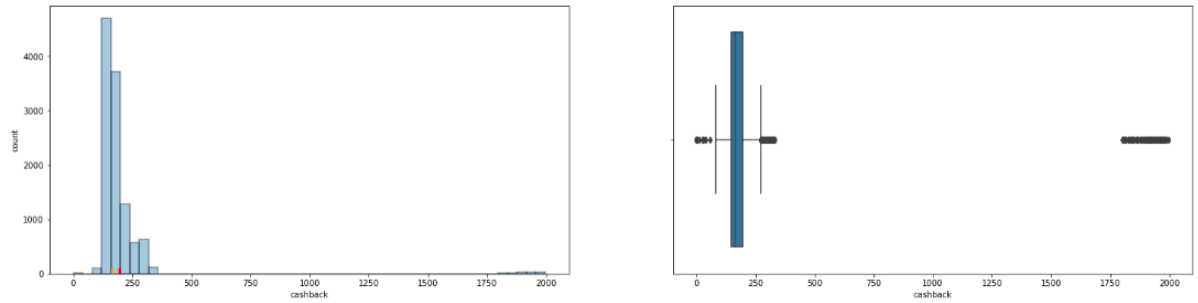


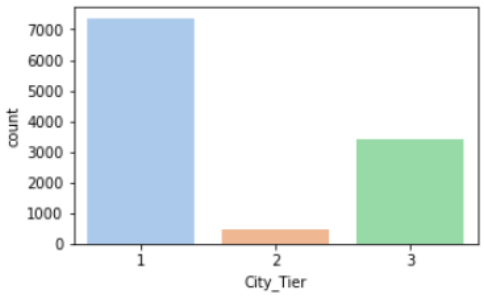
Figure 5: Univariate Analysis of Truly Continuous Variables

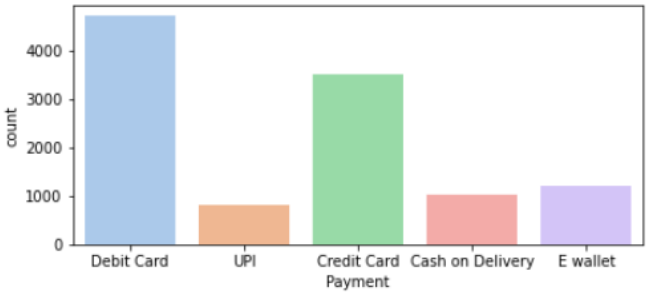
- None of the truly continuous variables in above figure are normally distributed and have presence of right skewness in most of them. Left skewness is seen only in Account_user_count variable.
- rev_per_month has an extremely high right skewness of 9.44 and cashback has an extremely high right skewness of 8.97.
- Tenure has a very high right skewness of 3.94 and coupon_used_for_payment has a very high right skewness of 2.58.

- CC_Contacted_LY has a high right skewness of 1.43, and Day_Since_CC_connect has a high right skewness of 1.32.
- rev_growth_yoy has a high right skewness of 0.75 and Account_user_count has a medium left skewness of 0.43.
- We can see presence of outliers for the truly continuous variables Tenure, CC_Contacted_LY, Account_user_count, rev_per_month, coupon_used_for_payment, Day_Since_CC_connect and cashback.
- Only rev_growth_yoy variable does not contain outliers. However, outliers presence in Account_user_count also does not seem justifiable as there could be many users registered under one account and this condition should not be applicable as an outlier.
- As per Tenure, maximum accounts have been added in the last couple of months (approx 23%) with highest accounts added in the previous month. Also, average tenure for an account is 11 months approx. which is a good sign and suggests that the customers do have the potential to have a high life cycle association with them.
- There is a sign of bad data '99' which could have been assigned to customers for whom the Tenure is not clearly known by the company and hence, it is also shown as an outlier. If so, it is recommended that they try and find this information if possible and reanalyse the data.
- As per Account_user_count, maximum accounts have 4 users and there are approx. 44% of such accounts. This can be a concern as churning of one account will lose multiple users for the company, thus multiplying proportion of losses tremendously. Hence, it is a very severe threat for the longevity of the company.
- As per rev_per_month, approx 2000 accounts have generated the highest average monthly revenue of 5000 units in last 12 months. Thus, churning will have a major impact on generating consistent revenue for the company and can hurt its growth and expansion plans.
- Average contribution is 16% to revenue growth yoy which is good but an increase to this is necessary to capture more market share by impressing investors, customers, vendors and partners.
- As per coupon_used_for_payment, approx. 40% accounts have used 1 coupon to make payment in last 12 months.
- As per cashback, approx. 1997 units is the highest average monthly cashback received by an account. Also, monthly average cashback generated is approx. 195 units, ie; cashback of approx. 2340 units annually.

ii. Univariate Analysis of Categorical Variables including encoded variables (This analysis is post imputing the missing and null values to make the report more relevant)

	Payment	Gender	account_segment	Marital_Status	Login_device
count	11260	11260	11260	11260	11260
unique	5	2	5	3	2
top	Debit Card	Male	Regular Plus	Married	Mobile
freq	4696	6812	4221	6072	8242





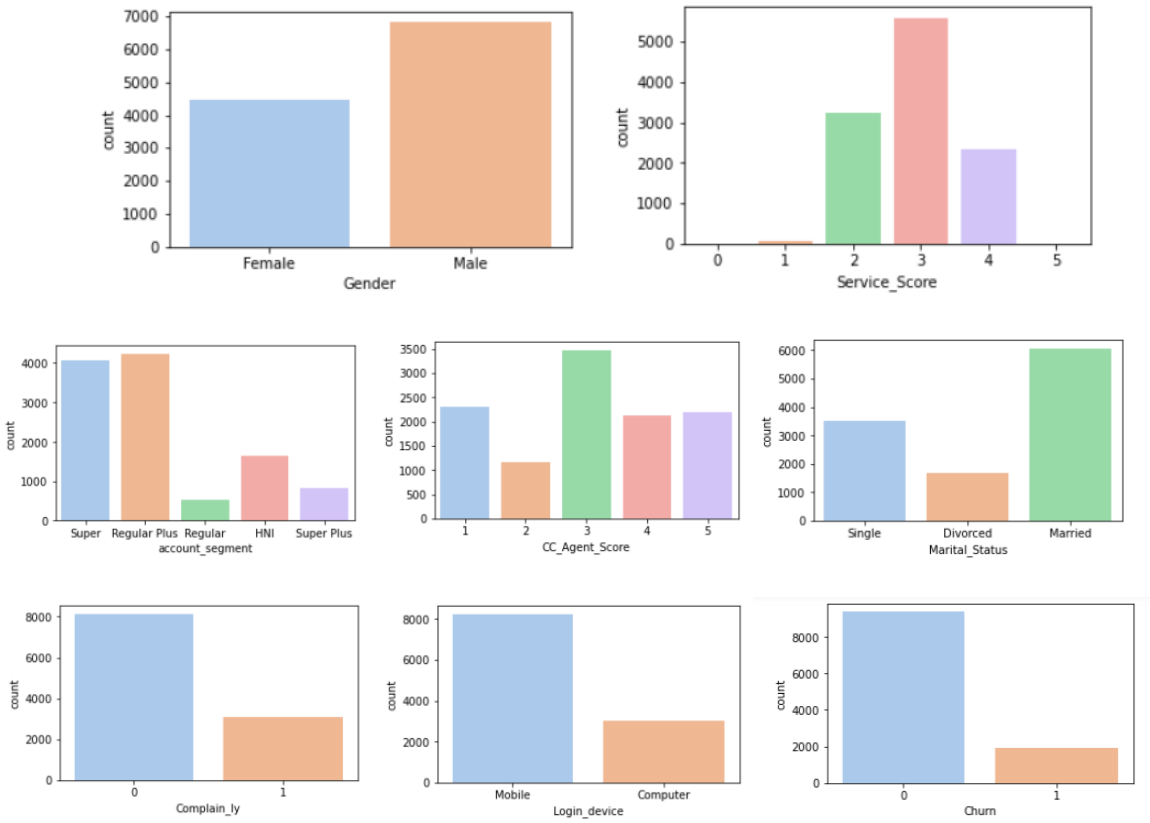


Figure 6: Univariate Analysis of Truly Categorical Variables

- Maximum accounts belong to tier 1 city around 62% accounts. Overall accounts for City Tier 2 is extremely low of only 4%. Company needs to add more accounts in City Tier 2.
- Debit cards are the most preferred payment option for the maximum accounts. Approx. 4500 or 40% users pay using a debit card followed by credit card with approx. 31% users. UPI is the least preferred payment mode.
- Most of the primary customers are Male (approx 7000 or 62% of total).
- As per Service_Score, satisfaction score of 3 has been given to the company by maximum accounts which is 49% of the total accounts indicating an average rating overall.
- The Regular Plus segment tops the list among 4 other options available (earlier summary in the analysis had identified Super as the top spending segment. However, post data cleaning (correctly replacing M as Male and F as Female) we are now able to determine the actual top spending segment). It comprises of approx 4500 of the total accounts. Super is 2nd with approx. 4000 accounts. Lowest no of accounts belong to 'Regular' category.
- As per CC_Agent_Score, Satisfaction score of 3 has been given by maximum accounts for customer care service indicating an average performance overall.
- Most of the primary customers are Married, around 6000 or 53% of the total.
- 27.6% of accounts had raised complaints in the last one year (approx. 3000 accounts) which is cautiously very high and needs to be lowered significantly going forward.
- 73% accounts prefer using mobile to login.
- 17% of the accounts have already churned which is quite high and bothersome for the business.

b) Bivariate analysis

i. Bivariate analysis of continuous variables (This analysis is post imputing the missing and null values to make the report more relevant)

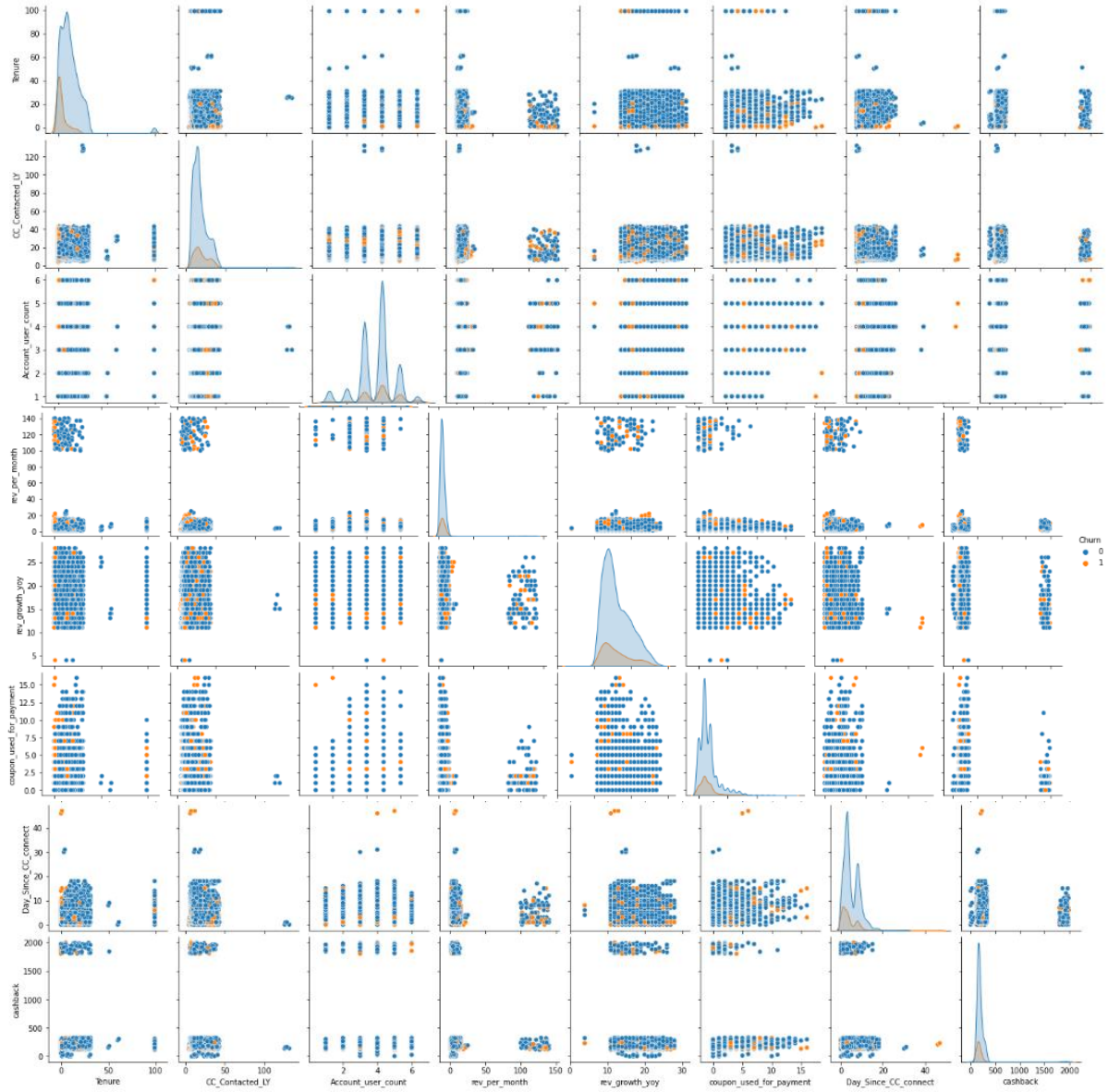
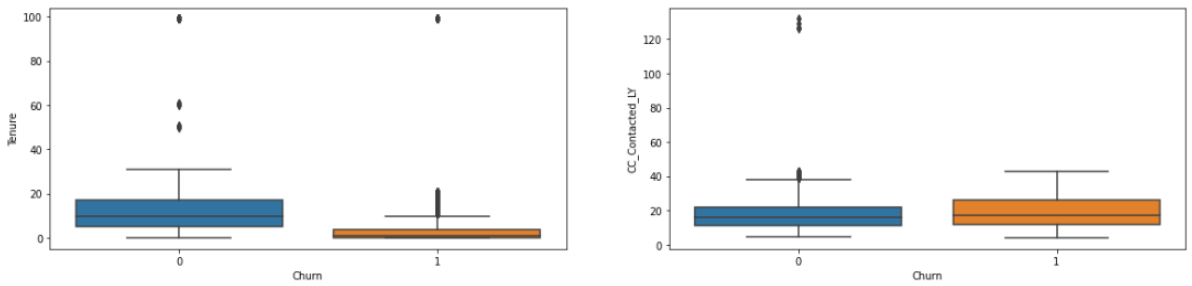


Figure 7: Bivariate Analysis of Truly Continuous Variables

- We do not see any signs of strong correlation among the truly continuous variables with each other.
- We do not see any pattern suggesting any variable being individually strong in predicting the accounts that are likely to churn. This can be seen as the distribution of accounts that have churned are stacked within the range of the accounts not likely to churn.



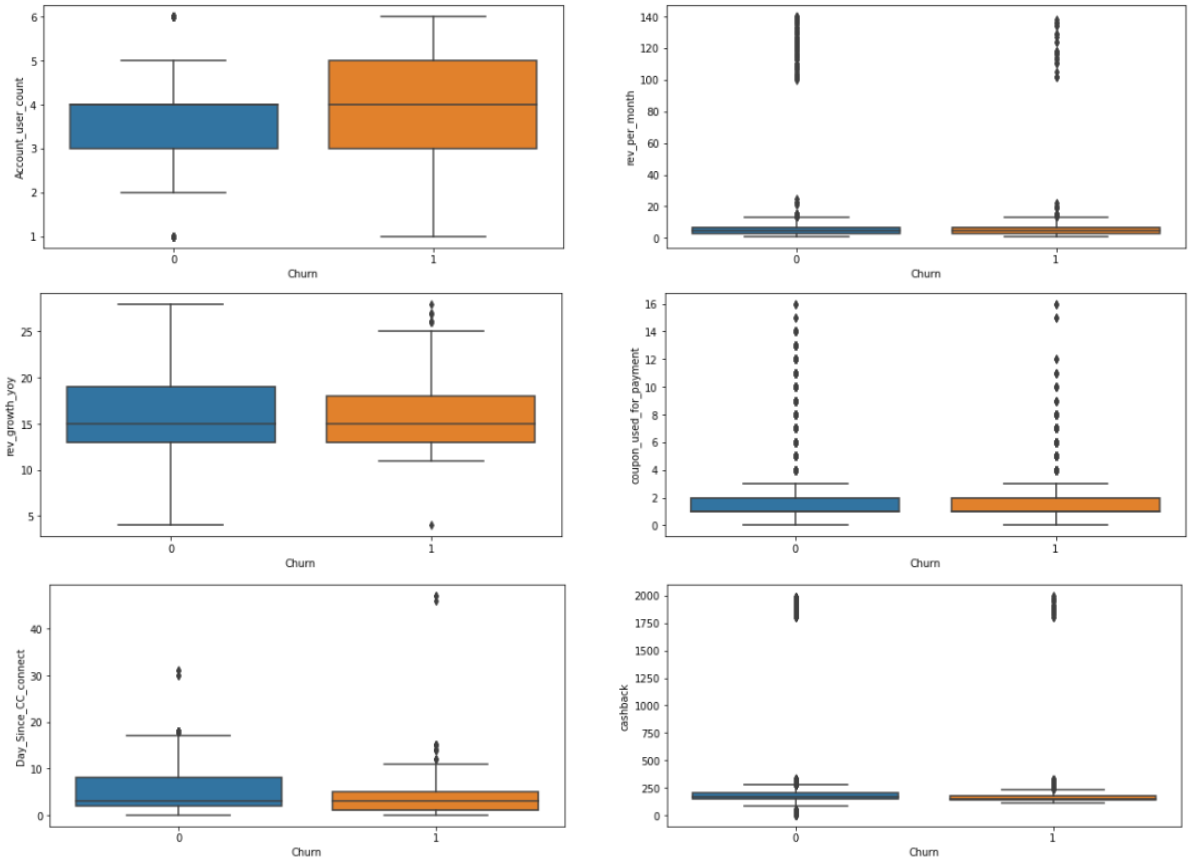
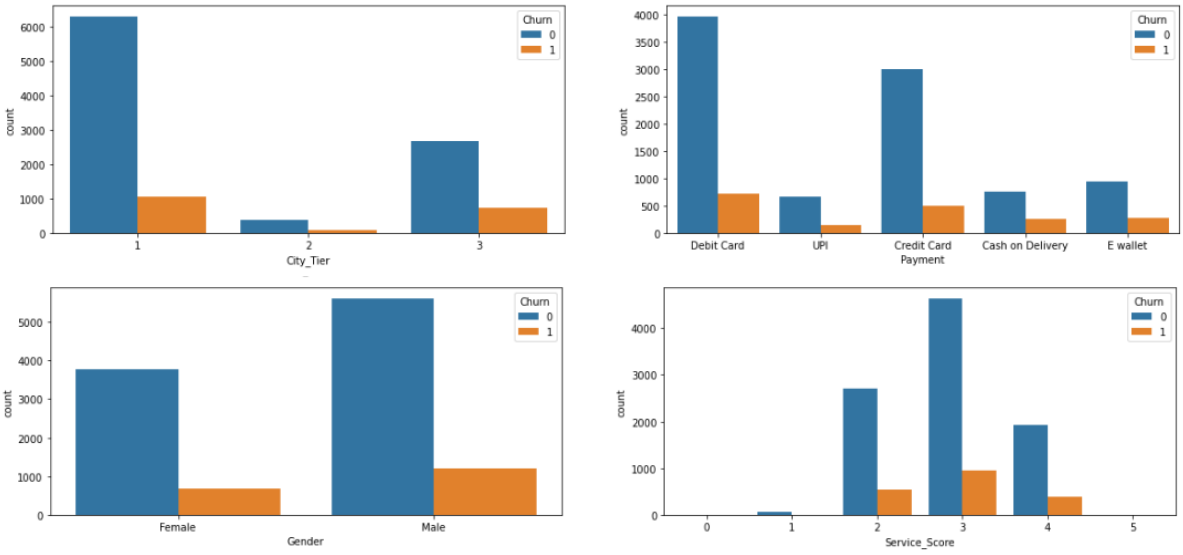


Figure 8: Bivariate Analysis of Truly Continuous Variables II

- Again looking at the above boxplots where we are comparing the continuous variables distribution w.r.t to target variable of Churn, we do not see any major differentiation between 0 and 1 for all variables as none of the classes are completely above or below from each other.
- Only Tenure variables is to able distinguish between class 0 and class 1 of the target variable as the median line of class 0 is outside from the other class.
- Median line of class 0 is slightly outside of class 1 for Account_user_count and perhaps can be a good predictor. Once we build the models, we will be able to evaluate the same.

ii. Bivariate analysis of categorical variables (This analysis is post imputing the missing and null values to make the report more relevant)



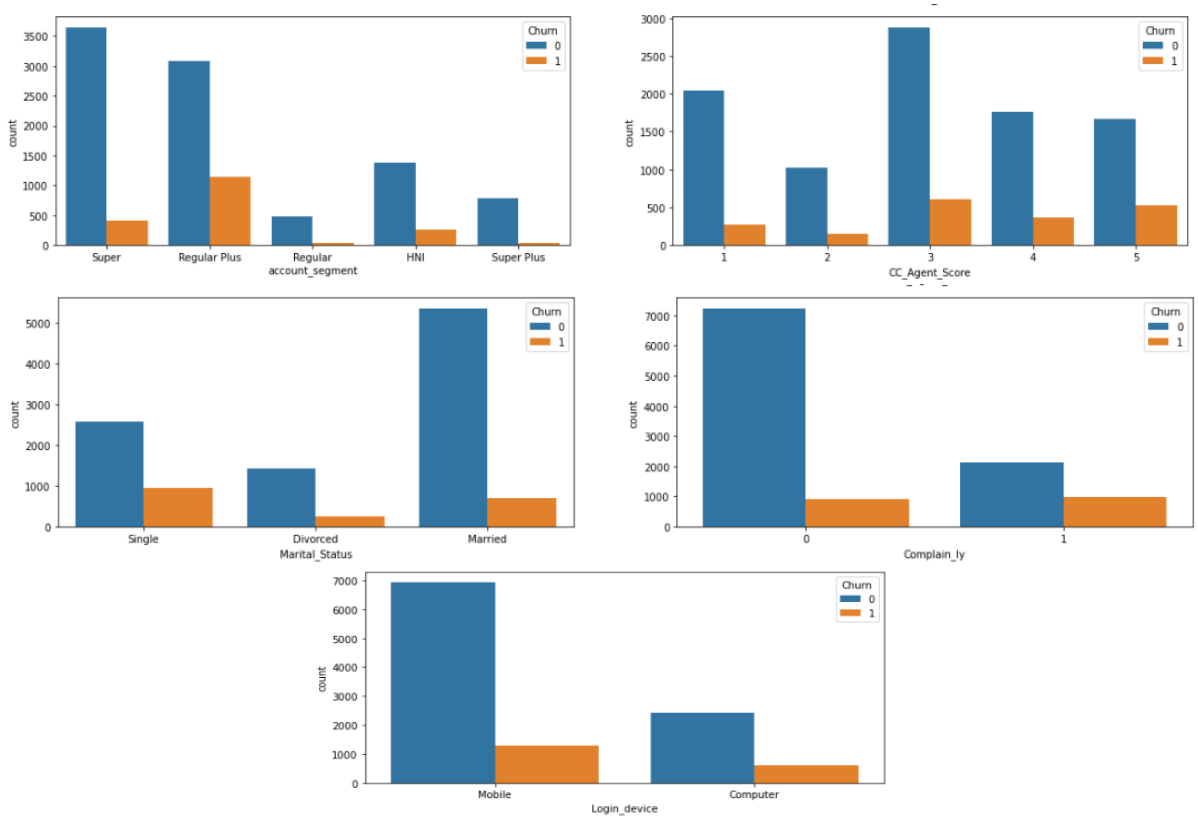


Figure 9: Bivariate Analysis of Truly Categorical Variables

- Even though Tier 1 city has highest no of customers who have churned across all 3 tier cities, the similarity of proportion of class 0 and class 1 is more significant for Tier 3 city. Accounts from City Tier 3 can be seen churning at a faster rate of 17.6% compared to other city tiers indicating that business is losing more customers from City Tier 3 than they are able to retain here.
- Accounts using cash on delivery and e-wallet are churning at a faster rate. Company needs to focus on improving in these areas of payment mode tremendously.
- More female account holders need to be added.
- Surprisingly, customers who are giving a higher rating of 3 and above for the company as well as customer care agents are churning which is a grave concern for the company. Also, the lowest score of 1 for customer care agents, is the second highest rating given by a huge 20% of total accounts indicating customer care service is not up to the mark.
- Regular Plus, Super and HNI accounts are churning more than others and maximum accounts belong to these segments. Hence, it can seriously hurt revenue growth.
- Proportion of customers who have registered a complaint are less compared to who have not registered but they are churning at a faster rate. Comparatively, 27.6% of accounts have raised complaints in the last one year (approx. 3000) and approx. 1/3rd of these have already churned which is quite high and needs immediate attention to turn things around here.
- Proportion of customers churning using both mobile and laptop seem same whereas only 27% use laptops to login. Company needs to pass on the analysis to their website design and development team to update and fix issues on immediate basis as the business heavily depends on the these two mediums.

c) Multivariate analysis

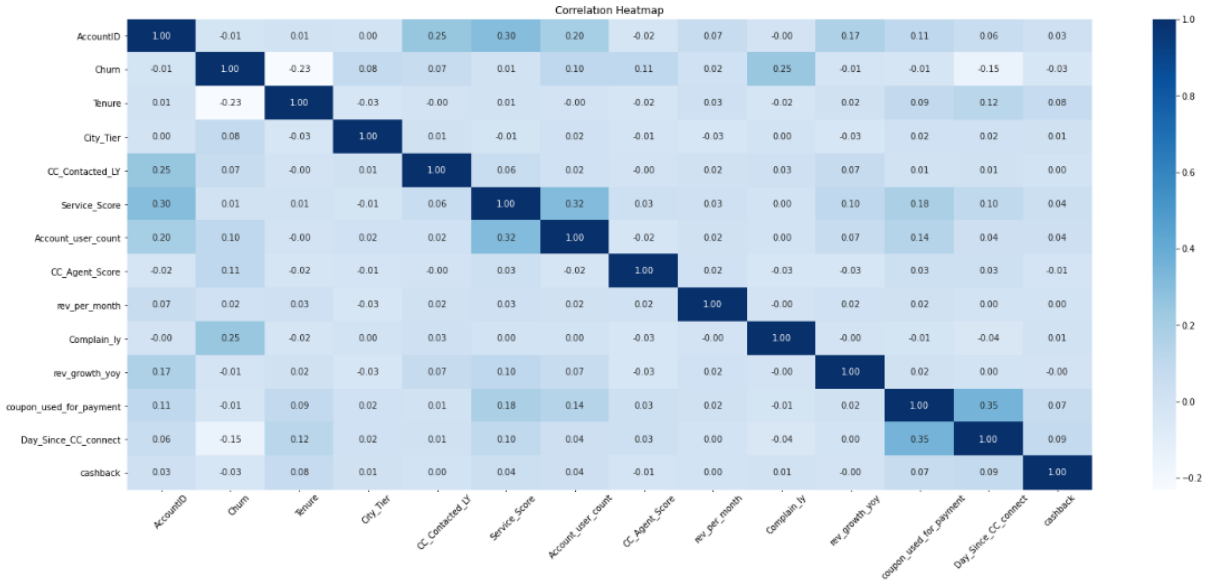


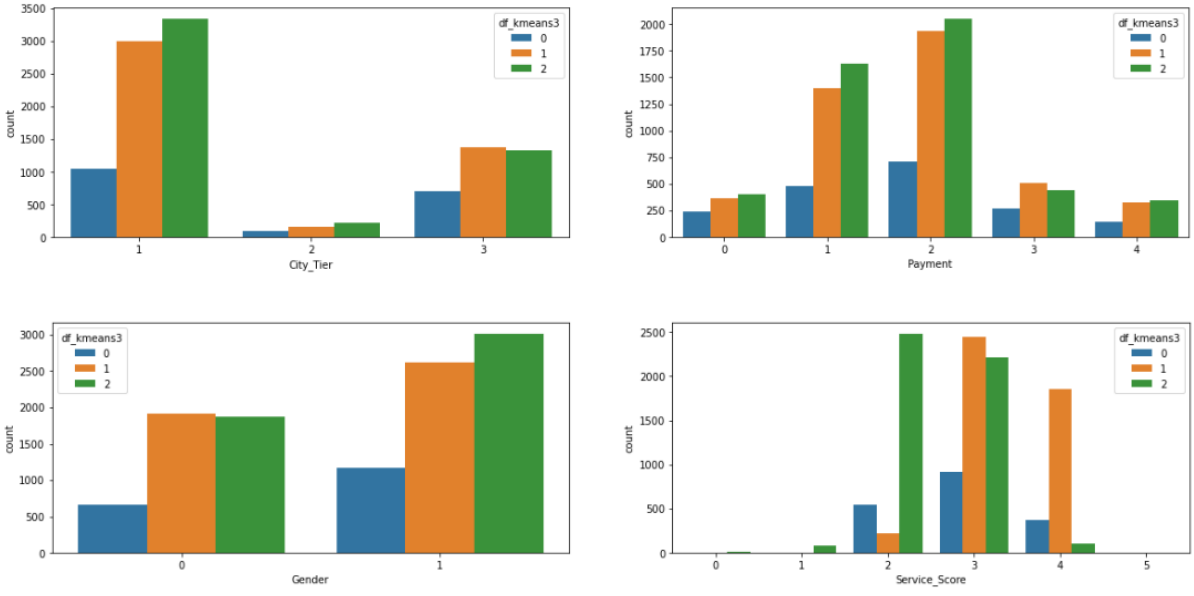
Figure 10: Correlation Heatmap

- Tenure is slightly negatively correlated to Churn (-0.23). Hence, higher the tenure better the chances for retention.
- Complain_ly is slightly positively correlated to Churn (0.25). Hence, complaints encourage probability to churn.
- Day_Since_CC_connect is slightly negatively correlated to Churn (-0.15). Hence, lesser contact with customer care can prevent customer from churning.
- Account_user_count is slightly positively correlated to Service_Score to company (0.32). Better Satisfaction Score to company results in more users being added to primary account.
- coupon_used_for_payment is slightly positively correlated to Day_Since_CC_connect (0.35). Customers using coupons to make payments tend to connect with customer care more frequently.

d) Business implications using clustering

- The clusters have divided all 11260 accounts in 3 segments (0,1,2) with segment 0 having 1841 accounts, segment 1 having 4535 accounts and segment 2 having highest accounts of total 4884.

i. Business implications using clustering for truly categorical variables



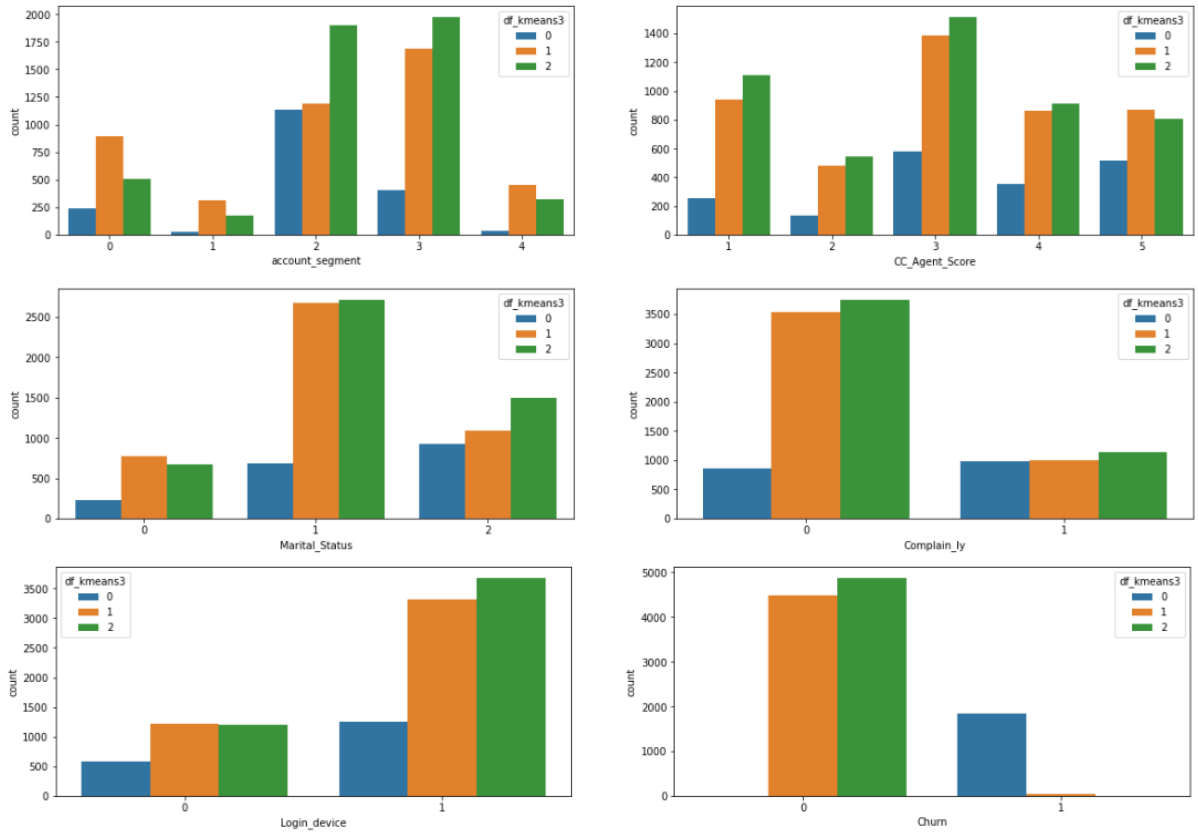


Figure 11: Business insights using clustering for truly categorical variables

- **<u>Segment 0 has highest no of churners and where all 1841 accounts are churners</u>** despite having only 50% of account holders compared to the other 2 segments. Hence, we will try and focus more on patterns for this segment in our analysis.
- Segment 0 accounts are lowest for all tiers but significantly lower for tier 1. So, accounts belonging to the other two city tiers are to be focused to be retained.
- Segment 0 accounts prefer 'Debit Card', 'Credit Card', 'Cash on Delivery' in the same order. Hence, improvements to offer seamless transactions in these areas are to be done.
- Segment 0 accounts have higher no of Males but both genders are churning almost equally which is not a good sign for the company and its product catalogue design team.
- Segment 0 accounts have given Service Score 3, 2 and 4 to company in the same order.
- Segment 0 accounts belong mainly to 'Regular Plus', 'Super', 'HNI' in the same order. Hence, focus is to increase spendings in these categories by introducing a good product portfolio mix to choose and select from.
- Segment 0 accounts have given Service Score 3, 5 and 4 to customer care service in the same order which is bothersome and needs intervention to analyse the legitimacy of data collection for the same.
- Segment 0 accounts are higher for Single primary customers indicating need to improve products for this category.
- 50% of accounts for segment 0 have raised complaints in last 12 months which is very significant and concerning.
- Segment 0 accounts who prefer Mobile are twice than the accounts who prefer Laptop.

ii. Business implications using clustering for truly continuous variables

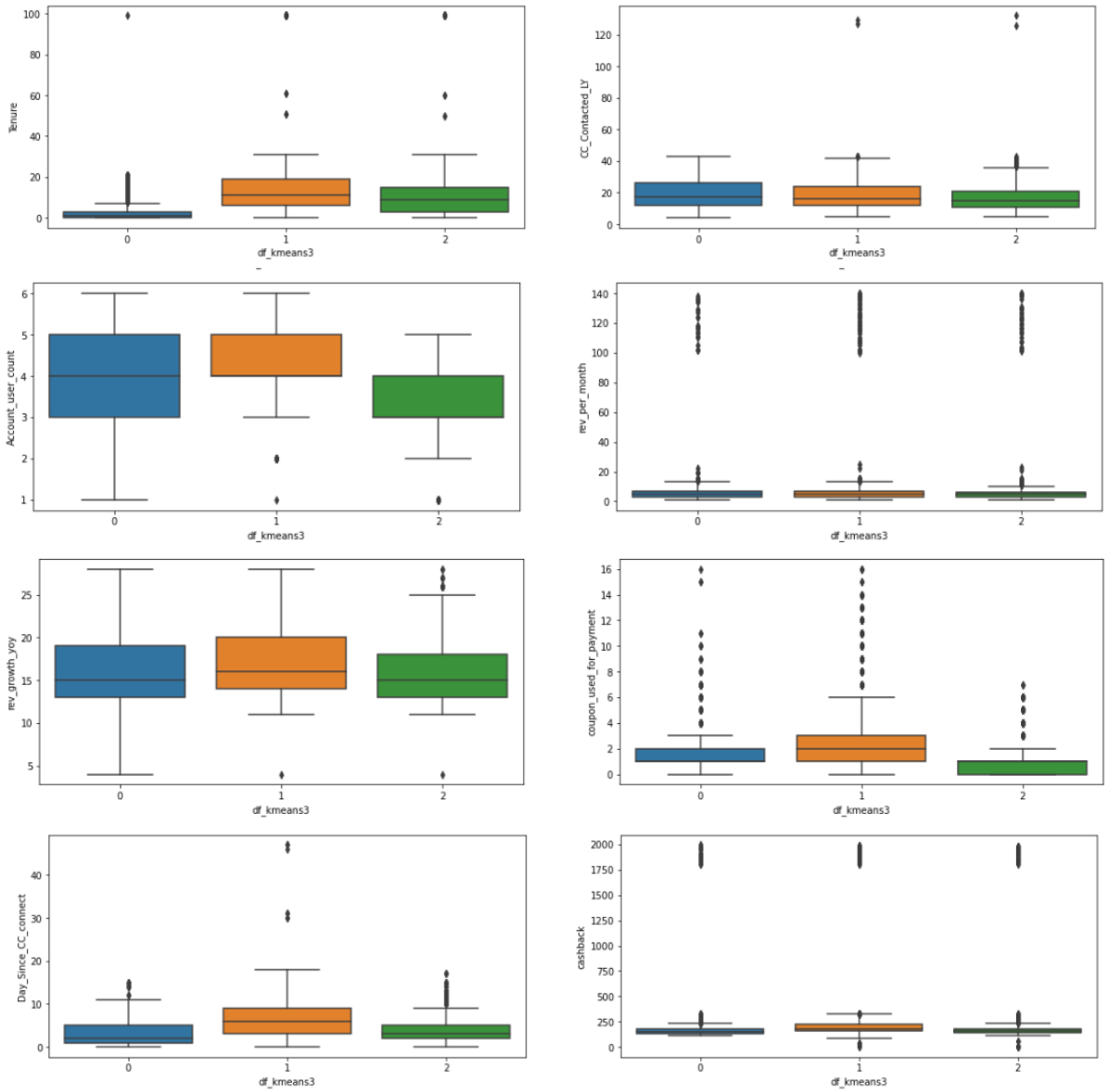


Figure 12: Business insights using clustering for truly continuous variables

- As per Tenure, segment 0 has accounts that are acquired newly compared to the other segments and ranges between 0 to 20 months with an average tenure of approx 3 months. Hence, it is necessary for company to design schemes to retain new customers.
- Segment 0 accounts have contacted customer care agents a median 19 times in last 12 months. Hence, company needs to reduce this count as well as reduce the frequency of contact between the two.
- Segment 0 accounts have a median 4 users per account which is very high and losing these many users in future as well can be a big threat to the company's survival.
- Segment 0 accounts contributed the highest average revenue per month of 6800 units. Overall average is approx. 6300 units generated in last 12 months. This too is a severe threat to company if higher revenue generating accounts are churning more.
- Segment 0 customers contributes a median of 15% growth yoy which is almost equal to the other segments. Hence, retaining any account likely to churn is great for the company.
- Segment 0 customers connect with customer care more often and reduction in this area will be a key factor in maximizing retention of accounts.
- Frequency of cashbacks offered is low which is good as this variable does not seem to affect probability of churn as well.

3. Data Cleaning and Pre-processing

a) Missing Value treatment

AccountID	0
Churn	0
Tenure	218
City_Tier	112
CC_Contacted_LY	102
Payment	109
Gender	108
Service_Score	98
Account_user_count	444
account_segment	97
CC_Agent_Score	116
Marital_Status	212
rev_per_month	791
Complain_ly	357
rev_growth_yoy	3
coupon_used_for_payment	3
Day_Since_CC_connect	358
cashback	473
Login_device	760

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11260 entries, 0 to 11259
Data columns (total 19 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   AccountID                11260 non-null  int64
 1   Churn                    11260 non-null  int64
 2   Tenure                   11260 non-null  int64
 3   City_Tier                11260 non-null  int64
 4   CC_Contacted_LY          11260 non-null  int64
 5   Payment                  11260 non-null  int8
 6   Gender                   11260 non-null  int8
 7   Service_Score            11260 non-null  int64
 8   Account_user_count       11260 non-null  int64
 9   account_segment          11260 non-null  int8
 10  CC_Agent_Score           11260 non-null  int64
 11  Marital_Status           11260 non-null  int8
 12  rev_per_month            11260 non-null  int64
 13  Complain_ly              11260 non-null  int64
 14  rev_growth_yoy           11260 non-null  int64
 15  coupon_used_for_payment  11260 non-null  int64
 16  Day_Since_CC_connect     11260 non-null  int64
 17  cashback                 11260 non-null  float64
 18  Login_device             11260 non-null  int8
dtypes: float64(1), int64(13), int8(5)
```

Figure 13: Missing Values including Null Values and Treated Missing Values

- Most Tree based models, Naive Bayes, KNN, boosting techniques etc can handle missing values. However, there a few models such as neural network, Logistic Regression, SVM which require to impute the missing values. Also, generally it is a good practice to either impute missing values appropriately or drop them if insignificant whenever possible to achieve better modelling results. For this study we have imputed the missing values as we did not wanted to lose the other observations within the same rows.
- Also, the total amount of missing values was only 1.25% excluding null values and hence, imputing them would not have changed the data significantly. Even when added with the count of null values to the missing values, the total values requiring imputation came to only 2.04%.
- Already encoded variables in original dataset such as City_Tier, Service_Score, CC_Agent_Score and Complain_ly were imputed them using their respective mode values (as it is the preferred way of imputing categorical variables)
- Balance categorical variables such as Payment, Gender, account_segment, Marital_Status and Login_device were also imputed using their respective mode values.
- Continuous variables of Tenure, CC_Contacted_LY, Account_user_count, rev_per_month, coupon_used_for_payment, Day_Since_CC_connect and cashback were imputed by their respective median values as they had outliers present in them as imputation with median values is preferred for continuous variables having outliers.
- We also imputed rev_growth_yoy with its respective median value instead of mean value despite not having any outliers as their wasn't much difference between the two values and also wanted to have uniformity in data type for most of the variables as much as possible.

b) Outlier treatment

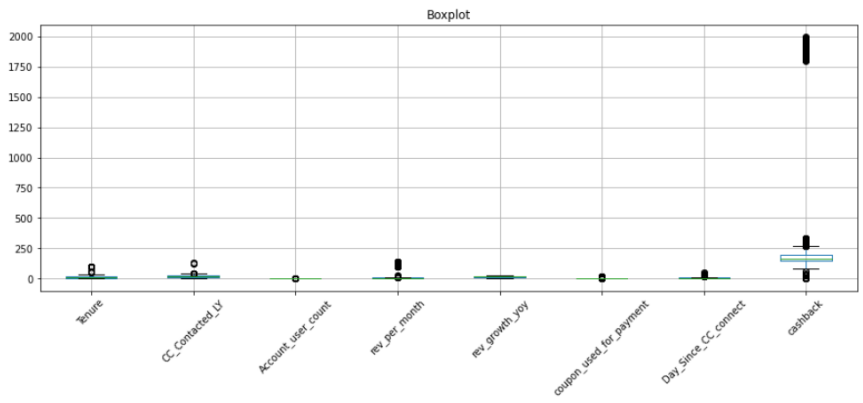


Figure 14: Boxplot with Outliers

- We can see presence of outliers for the continuous variables Tenure, CC_Contacted_LY,Account_user_count, rev_per_month, coupon_used_for_payment, Day_Since_CC_connect and cashback.
- Only rev_growth_yoy variable does not contain outliers. However, outlier presence in Account_user_count also does not seem justified as there could be many users registered under one account and this condition should not be applicable as an outlier.
- Most of the tree based models such as Cart, Random Forest, Neural Network, Bagging are not affected by outliers. However, regression models such as Logistic Regression and boosting techniques such as AdaBoost, XG Boost etc are affected. Hence, we will split our original data in two subsets one which includes the outliers and another one without outliers and build models using either or both the subsets whichever applicable appropriate to the model we are building and check the predictions and performances for these models.

AccountID	0.00
Churn	0.00
Tenure	1.16
City_Tier	0.00
CC_Contacted_LY	0.04
Payment	0.00
Gender	0.00
Service_Score	0.00
Account_user_count	0.00
account_segment	0.00
CC_Agent_Score	0.00
Marital_Status	0.00
rev_per_month	0.94
Complain_ly	0.00
rev_growth_yoy	0.00
coupon_used_for_payment	0.04
Day_Since_CC_connect	0.06
cashback	0.96
Login_device	0.00

Figure 15: Percentage of Outliers basis lower limit of 5% and upper limit of 95% quantiles

- From above we see that the variables Tenure (1.16%), CC_Contacted_LY (0.04%), rev_per_month (0.94%), coupon_used_for_payment (0.04%), Day_Since_CC_connect (0.06%) and cashback (0.96%) have presence of outliers w.r.t 5% and 95% quantile limits. We will treat the outliers basis these above mentioned quantile limits. The quantile limits are kept as such so that not a large chuck of data is manipulated when treating the outliers keeping in mind the best interest of the business as well.

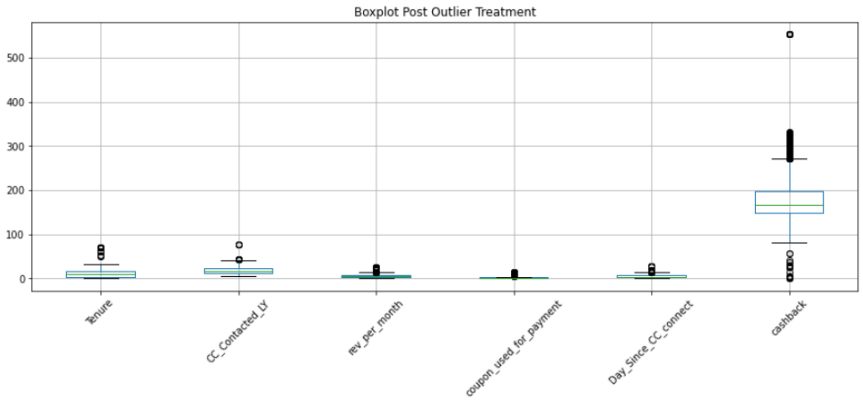


Figure 16: Boxplot post Treating Outliers basis lower limit of 5% and upper limit of 95% quantiles

c) Need for Variable transformation (if any)

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

Figure 17: Categorical Variable Transformation using Label Encoding

- The dataset has variables as given in the above figure which are object data type. These needs to be converted to integer or float data type as predictive modelling for supervised learning techniques such as Cart, Random Forest, Logistic Regression, SVM etc require so to build models and predict the results accurately. Hence, we have used label encoding to transform the variables Payment, Gender, account_segment, Marital_status and Login_device having object data type to numerical values as seen in the above figure.
- We used label encoding to transform the variables as the dataset already had a few variables encoded using the same technique when the dataset was handed over for the analysis.
- The variables were encoded only after appropriately treating the bad data observed in some of the variables with incorrect characters assigned such as $, +, 99, #, etc to make the dataset more robust and meaningful to perform various model building techniques.
- We have also assigned correct data type of integer to already encoded variables City_Tier, Service_Score, CC_Agent_Score and Complain_ly present in original dataset and numerical variable CC_Contacted_LY and Account_user_count has been changed from float to integer as well in order to maintain the correct data type across all variables.

d) Variables removed or added and why (if any)

- AccountID variable was not of much significance for prediction. Hence, we removed it.
- We did not see any requirement to create new variables which would add significance to our model building and/or predicting the customers probability to churn.

4. Model Building

a) Prerequisites to Model Building

i. Train Test Split:

- We split the data into two sets:
  - i. xo and yo split contains unscaled observations and contains outliers.

- ii. x and y split contains unscaled observations but with outliers treated.

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

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

Figure 18: Train and Test Split Proportions

ii. Variable Scaling:

- Some of the model building techniques especially distance based techniques such as KNN and Support Vector Machines need the dataset to be scaled in order to avoid performance issues and also a few other modelling techniques even though do not require scaling but their performances benefit from having a scaled dataset. Hence, we have scaled one of our datasets (x y).
- We used StandardScaler to change the features to the same scale. After standardization, each feature has zero mean and standard deviation of 1. We used StandardScaler as we had some variables with very high variances compared to others and also having different scales such as text, decimal values, numbers and percentages and we needed to bring them to similar scales and also as it is most widely used and recommended technique to standardize the data.
- Standardization is fit on the training dataset only (x_train). Then, the test dataset (x_test) is standardized using the fitting results from the training dataset.

iii. SMOTE:

- We may face a data imbalance problem during our model building process due to lower no of class 1 observations. In case of this, we would use SMOTE technique to create a balanced dataset in which the minority class ie; class 1 for our dataset would be over sampled by generating synthesized data.
- We have kept sampling_strategy at 50% ie; for every two observations of class 0, one synthesized observation of class 1 would be generated as it produced the best results and which also indicates that not much synthesized data points were necessary to be added.
- A total of 1950 synthesized data points for class 1 has been added. Hence, SMOTE train dataset now has 9832 data points against 7882 in the original train datasets. Data points for test remains same at 3378.
- We have also created a Train Test Split on scaled dataset of x dataset and fitted it to SMOTE as shown below to build models using SMOTE requiring scaled dataset on both train and test for eg; KNN models and SVM Models

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

b) Model Building Approach and Efforts to improve model performance

- This Customer Churn prediction problem is a 'Classification Problem' having categorical target variable with binary values (0 and 1) where 0 represents customers who have not churned and 1 represents customers who have churned

- Prime emphasis was to build models which can achieve high recall value on test datasets for class 1 observations (ie; customers who have churned).
- Prominent Classification Problem related modelling techniques such as Logistic Regression, Linear Discriminant Analysis (LDA), CART, Random Forest, Naïve Bayes, KNN, SVM and ANN were built and predicted on both train and test datasets after which their performances were tabulated using necessary performance measures such as accuracy, recall, precision, f-1 and AUC.
- We started by creating a base model for one of the above techniques and evaluated its performance.
- Then, the same model was tuned and hyper tuned multiple times until either a satisfactory result was achieved or we had to discard the model due to under performance.
- Finally, SMOTE was applied to the models which had either under-performed or had satisfactory results to check whether their performance improved and also to check if these models were effected by imbalance in the dataset or not.
- We then followed up by extracting important features for models which performed well and also ran 10 fold cross validation to check if our models were stable and consistent.
- The above process was continued on ensemble techniques as well such as Bagging, AdaBoost, GradientBoosting and XGBoost as well.
- Below is a chart for all the built and maintained models. We have excluded ANN, Naïve Bayes and LDA models, and other models post multiple tuning and hyper tuning attempts as they had under-fitted on both train and test or/and under-performed. A few of these are however retained in the jupyter notebook attached along with this report for reference.

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

Random Forest_model 5 - Hyper Tuned with SMOTE (unscaled data and with outliers)	0.98	0.95	0.96	0.85	0.98	0.84	0.97	0.85	0.99	0.98	Slight Overfit
CART (Unscaled Dataset without Outliers)	1.00	0.95	1.00	0.85	1.00	0.85	1.00	0.85	1.00	1.00	Overfit
Bagging with SMOTE (unscaled data and without outliers)	0.99	0.95	0.97	0.85	0.99	0.85	0.98	0.85	0.99	0.98	Slight Overfit
XGBoost (unscaled data and without outliers)	1.00	0.97	1.00	0.85	1.00	0.96	1.00	0.90	1.00	0.99	Overfit
Ada Boost Tuned (unscaled data without Outliers)	1.00	0.98	1.00	0.87	1.00	0.98	1.00	0.92	1.00	1.00	Overfit
Gradient Boosting Tuned (unscaled data and without outliers)	1.00	0.98	1.00	0.87	1.00	0.99	1.00	0.92	1.00	0.99	Overfit
Random Forest_model 6 - Hyper Tuned (unscaled data and with outliers)	1.00	0.98	1.00	0.88	1.00	0.99	1.00	0.93	1.00	0.99	Overfit
XGBoost Tuned (unscaled data and without outliers)	1.00	0.97	0.99	0.88	0.99	0.92	0.99	0.90	1.00	0.99	Slight Overfit
SVM Tuned (scaled dataset without Outliers)	1.00	0.98	0.99	0.91	1.00	0.95	0.99	0.93	0.99	0.99	Good Model
KNN_model with SMOTE (scaled data without outliers)	0.98	0.94	0.99	0.93	0.95	0.76	0.97	0.84	0.99	0.99	Good Model

Figure 19: All models comparison chart

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

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

Random Forest_model 5 - Hyper Tuned (unscaled data and with outliers)	0.98	0.95	0.87	0.75	0.99	0.95	0.93	0.84	0.99	0.98	Slight Overfit
Logistic Regression_model_3 with 0.19 threshold	0.80	0.79	0.81	0.76	0.45	0.42	0.57	0.55	0.80	0.80	Poor Precision
Logistic Regression_model_4 with 0.19 threshold	0.78	0.78	0.79	0.77	0.42	0.42	0.55	0.54	0.79	0.79	Poor Precision
CART Tuned with SMOTE (Unscaled Dataset without Outliers)	0.97	0.92	0.93	0.78	0.97	0.77	0.95	0.77	0.99	0.99	Overfit
Bagging (unscaled data and without outliers)	0.98	0.95	0.92	0.79	0.99	0.92	0.95	0.85	0.99	0.98	Overfit
Random Forest_model (unscaled data with outliers)	1.00	0.97	1.00	0.85	1.00	0.98	1.00	0.91	1.00	0.99	Overfit
CART (Unscaled Dataset without Outliers)	1.00	0.95	1.00	0.85	1.00	0.85	1.00	0.85	1.00	1.00	Overfit
Bagging with SMOTE (unscaled data and without outliers)	0.99	0.95	0.97	0.85	0.99	0.85	0.98	0.85	0.99	0.98	Slight Overfit
XGBoost (unscaled data and without outliers)	1.00	0.97	1.00	0.85	1.00	0.96	1.00	0.90	1.00	0.99	Overfit
Ada Boost Tuned (unscaled data without Outliers)	1.00	0.98	1.00	0.87	1.00	0.98	1.00	0.92	1.00	1.00	Overfit
Gradient Boosting Tuned (unscaled data and without outliers)	1.00	0.98	1.00	0.87	1.00	0.99	1.00	0.92	1.00	0.99	Overfit
Random Forest_model 6 - Hyper Tuned (unscaled data and with outliers)	1.00	0.98	1.00	0.88	1.00	0.99	1.00	0.93	1.00	0.99	Overfit

Figure 20: Models eliminated from evaluation

c) Most optimum model & why was it chosen

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

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

Figure 21: Models selected for evaluation

- CART Tuned model and KNN_model did seem reliable but had lower recall values for class 1 compared to the other models and we aim to build a model that would maximize the chances of predicting customers for class 1.
- Random Forest_model 5 - Hyper Tuned with SMOTE had a good recall of 85% for class 1 on test dataset but too seemed unstable with lower precision and F-1 scores on test dataset compared to the train dataset.
- SVM with SMOTE also had the same issue of lower precision on test dataset of only 76% and also had a lower F-1 score of 79% compared to the train dataset making the model less reliable.

- KNN_model with SMOTE had the best recall of 93% on test dataset for class 1 (correctly identified 529 true positives of 569 observations). However, the precision on test dataset was only 76% and F-1 score of 84% were drastically lower compared to the train dataset making the model less reliable and unstable.
- XGBoost Tuned model performed extremely well on both the train and test dataset (incorrectly identified only 68 observations for class 1 and 43 observations for class 0 on test dataset). It has also performed well in predicting class 0. It is also a reliable and stable model with good accuracy, precision, F-1 and AUC scores.
- SVM Tuned has better scores than all the other models on train and test datasets across all performance metrics only except for recall for test dataset and AUC on train dataset. However, it was seen during cross validation that scores were varying on the test dataset making the model inconsistent. Also, It is not possible to extract feature importance for SVM models as they rely on a linear model that is not easily interpretable and hence it becomes a difficult to determine which features are most important for them.
- Hence, from all the above observations we can determine that XGBoost Tuned model is the most optimal model.

5. Model Validation

- The XBoost Tuned model was evaluated basis its overall performance across all the performance measures such as accuracy, recall, precision, f-1 and AUC. Below we have shared a gist of the parameters used to tune the model, prediction performance on train and test datasets both using the confusion matrix and classification report and 10 fold cross validation results.
- Before we get into model evaluation and validation, there are a few basic information related to the performance measures that we should be aware of to interpret the results of the predictions made on train and test datasets as given below:

Understanding Confusion Matrix

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

Understanding Classification Report

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

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

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

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

- Best Parameters Identified:

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

Predictions on Train Dataset

```
Accuracy Score is 0.9963207307789901

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

Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6555
           1       0.99      0.99      0.99      1327

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

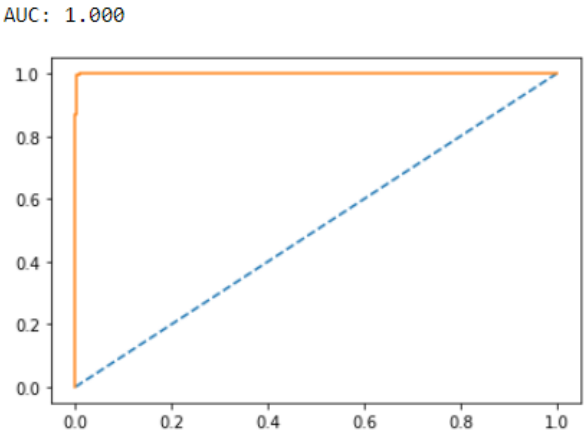


Figure 22: Performance Measure of XGBoost Tuned Model on Train Dataset

- We can see that the Accuracy score is high which states that the model has correctly predicted 99.6% of total cases including both class 0 and class 1 as well.
- The confusion matrix suggests that the model has:
  a) correctly identified 1318 data points for class 1 (True Positives) belonging to churners of the total 1327
  b) Only 9 class 1 data points have been incorrectly identified as belonging to class 0. (False Negatives)
  c) correctly identified 6535 data points for class 0 (True Negatives) out of 6555.
  d) 20 class 0 data points have been incorrectly identified as belonging to class 1 (False Positives)
- The classification report suggests that the model has:
  a) Identified 99% of real True Positives of all data points (Recall for Class 1) and almost 100% of real True Negatives of all data points (Recall for Class 0)
  b) Identified real True Positives as True Positive correctly 99% of the times of all correctly identified data points (Precision for Class 1) and real True Negatives as True Negative almost 100% of the times of all correctly identified data points (Precision for Class 0)
- AUC (Area under Curve) is high at almost 100% which means that the model is very highly capable of distinguishing between class 0 and class 1.

Predictions on Test Dataset

```
Accuracy Score is 0.9671403197158082

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

Classification Report
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      2809
           1       0.92      0.88      0.90       569

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

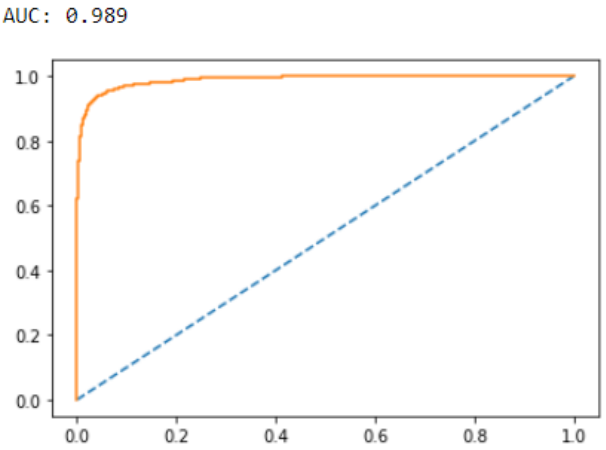


Figure 23: Performance Measure of XGBoost Tuned Model on Test Dataset

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

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

Figure 24: XGBoost Tuned Model Train and Test Comparison

Cross Validation Score for Train Dataset:

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

Cross Validation Score for Test Dataset:

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

Summary for XGBoost Tuned model

- Recall is 99% on Train dataset and 88% on Test dataset.
- Accuracy is nearly similar with 100% for Train dataset and 97% on Test dataset.
- Precision is 99% on Train and 92% on Test dataset.

- The F-1 score dropped from 99% on Train dataset to 90% on Test dataset.
- AUC is near equal with 100% on Train dataset and 99% on Test dataset.
- After 10 fold cross validation, scores both on train and test data set respectively for all 10 folds are almost same. Hence, the model is reliable and valid.

6. Final interpretation / recommendation

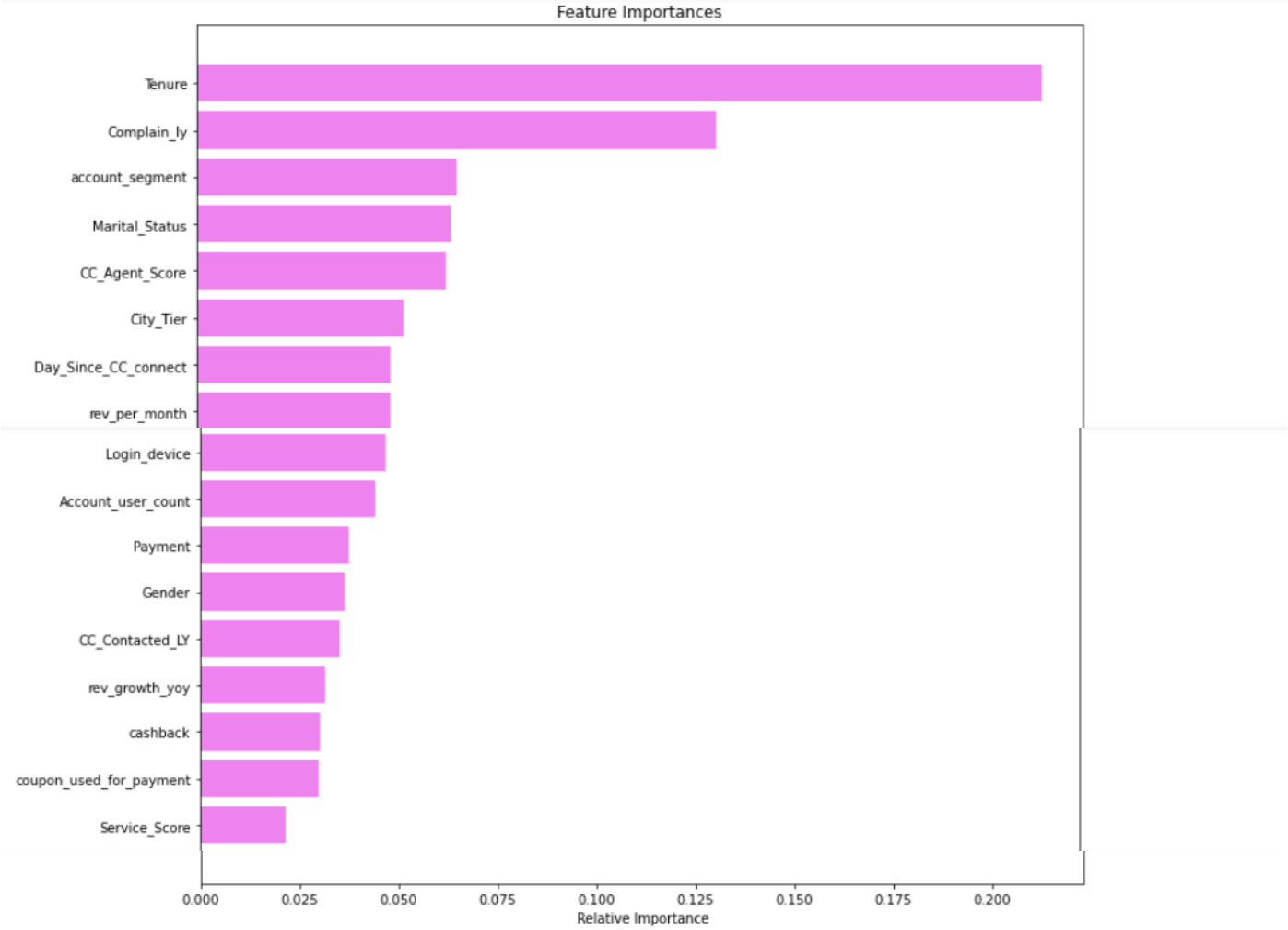


Figure 25: Important Features for XGBoost Tuned Model

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

- **Tenure** - Tenure of an account in months.
- **Complain_ly** - Complaints Raised in Last Year.
- **account_segment** - Account Segment basis of spends.
- **Marital_Status** - Marital Status of primary account holder.
- **CC_Agent_Score** - Satisfaction Score given to Customer Care Agents.
- **City Tier** - City Tier of primary account holder.
- **Day_Since_CC_connect** - Days Since Last Customer Care Connect.
- **rev_per_month** - Average Revenue Per Month for last year.
- **Login_device** - Login Device preferred.
- **Account_user_count** – No of Users per account.

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

Figure 26: Coefficients of all 17 Predictor Variables

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

Insights & Recommendations

- **Tenure:** Tenure of an account in months.
  - This variable was observed as important across all models built. Hence, it is extremely necessary to convert customers into long term loyal customers.
  - Coefficient for Tenure is negative, which means increase in tenure decreases the possibility of churn.
  - Maximum accounts were added in the last couple of months (approx 23%) and the concern is that customers who are churning also have low average tenure of approx 3 months.
  - To tackle this, the business can introduce diverse loyalty programs rewarding customers with different tenure slabs starting from 3 months and above like slabs of 3/6/9/12 months etc.
  - Introduce referral benefits for adding new users to the account or even offer redeem points to primary account holder on purchases made by the other users in the account.
  - Ensure maximum customers are regularly seeing their products, offers, etc through emails, push notifications, social media posts, youtube advertisements etc.
  - Ensure that the business or the management does not get involved in any controversy which hampers its reputation. Customers like to be associated with clean brands.

- **Complain_ly:** Complaints Raised in Last Year.
  - This variable was observed as important across 85% of the all models we built and in all the 7 models we selected to evaluate. Hence, it is extremely necessary to avoid any complaints from customers or/and if any complaints are received then to resolve it in a swift and smooth manner. This highlights the efficiency of a business to provide excellent customer service.
  - Coefficient for Complain_ly is positive, which means increase in complaints increases the possibility of churn.
  - 27.6% of accounts had raised complaints in the last one year (approx. 3000 accounts) and nearly 1/3rd of these had churned which is quite high.
  - E-commerce website, mobile application, etc needs to be in an extremely good functional condition which can offer a seamless purchasing and transacting experience for the users.
  - User interface to be effortless for even customers without technological know-how.
  - All partners/vendors/support businesses such as cargo and shipping, packaging, freight and transport, IT, services and communication etc should be in sync to avoid any loopholes. Hence, need to choose them

diligently and incentivise them for excellent service and also can provide them reward and recognition in exchange.
    - Arrange for feedback and surveys to understand customer sentiments and try and find areas which can create opportunity for complaints to tackle them beforehand.
    - Try and dedicate a small team to exclusively handle customer, vendor and inter-company complaints.
    - Share customer connect experiences and feedbacks received with other departments to create synergies through collaborations which can ensure in building a great ecommerce platform for its users.

- **account_segment:** Account Segment basis of spends.
    - This variable was observed as important across 80% of the all models we built and in all the 7 models we selected to evaluate
    - Coefficient for account_segment is negative, which means increase in account_segment decreases the possibility of churn or increase in spending prevents churning.
    - Maximum accounts are under 'Regular Plus' which can be said to be the average segment. Second maximum customers belong to 'Super' segment which is 2nd best segment. Even maximum customers who have churned belonged to these two categories and 'HNI' segment as well.
    - Specific targeted offers/combos can be designed for customer belonging the 'Regular Plus' which can move them to 'Super'. Similar strategies can be formed for accounts of 'Super' segment to move them up to 'Super Plus'.
    - Most purchased products/categories for customers belonging to these three segments can be evaluated to create such combos/offers. Even newer, in demand products in these categories can be introduced.
    - Similar products/supplementary products based on interests, past purchases can be run under product recommendations and shared via email, push notifications etc for these specific segments.

- **Marital_Status:** Marital Status of primary account holder.
    - This variable was observed as important across 65% of the all models we built and in 75% in all the models we selected to evaluate.
    - Most of the Primary customers are Married. However, maximum customer who are churning are Single. Approx 28.5% of all Single primary customers have churned which is very high. Hence, focus should be on Single customers mainly.
    - Shopping interest and past product purchase history based recommendations need to be set for both Single and Married Customers. Inventory should include more trending and in fashion/style products.
    - For Married, anniversary greetings, offers, couple combos can be curated. Products for couples can be recommended along with Valentine's day offers. Special offers for birthdays can be released.
    - Can offer heavy discounts on select products as well to make it more attractive and affordable to youth. Can also run mega clearance sale during Festivals, New Year's, etc on outdated and unsold inventory to try and trigger impulse buying.

- **CC_Agent_Score:** Satisfaction Score given to Customer Care Agents.
    - This variable was observed as important across 90% of the all models we built and in 75% in all the models we selected to evaluate.
    - Coefficient for CC_Agent_Score is positive which is odd but it means users who are giving a higher rating are churning and we can see this as users who are giving a higher rating of 3 and above are churning.
    - The lowest score of 1, is the second highest rating given by a huge 20% of total accounts which is very concerning.
    - Business needs to analyse how this data of satisfaction score to customer care representative is collected and if it is collected in a legitimate way to take necessary actions.
    - Record and Track customer service performances and reward employees doing well. Special training sessions for customer service representatives can be arranged once in a while as well to hone their skills.

- **City Tier:** City Tier of primary account holder.
    - This variable was observed as important across 60% of the all models we built and in 75% in all the models we selected to evaluate.
    - Accounts from City Tier 3 are churning at a faster rate at 17.6% compared to other city tiers. Overall accounts for City Tier 2 is extremely low of only 4%.

- Frequency of reaching customers to these city tiers need to be improved by increasing partners/vendors/distributors in these city tiers. Choose them diligently and incentivise them for excellent service.
- Last mile connectivity and delivery to be improved for the same. Can tie-up with top logistic companies offering end to end delivery solutions for City Tier 3. Same can be looked into for City Tier 2 once a decent no of accounts are established their as well.

- **Day_Since_CC_connect:** Days Since Last Customer Care Connect.
  - This variable was observed as important across all models built.
  - Coefficient for this feature is negative, which means increase in gap between customer and customer service agent decreases the possibility of churn.
  - Approx 50% customers have contacted customer care between 0 to 3 days. However, higher the gap better are the chances of retention.
  - Hence, better customer service, product quality and portfolio, richer user experience, etc would increase the possibility of retention.
  - It was observed that customers using coupons to make payments connected with customer care more. So, need to smoothen the process to use coupons reducing the need for assistance and train staff for seamless assistance when contacted.
  - Make tutorials available to use the website, mobile application. Provide quick solutions through auto bots, show steps and suggestions to improve the purchase experience and can also hire more customer service staff to meet the demand.

- **rev_per_month** - Average Revenue Per Month for last year
  - This variable was observed as important across all models built.
  - Accounts that have churned contributed the highest average revenue per month of 6800 units. Whereas the overall average for all accounts was lower at approx. 6300 units. Hence, accounts contributing more to revenue are churning at a faster rate which can impact revenue growth and stability tremendously.
  - The business needs to analyse the reviews for accounts that have churned. They can also request them for feedback to understand the pain points and in exchange, reward them with discounted offers/feedback points/free coupons/free gifts.
  - Business can also try to give them a complimentary upgrade to a higher account segment and benefits that come with it to lure them back.
  - Ensure maximum of these churned customers too are regularly seeing your products, offers, etc through emails, push notifications, social media posts, youtube advertisements etc

- **Login_device:** Login Device preferred.
  - This variable was observed as important across only 60% of the all models we built.
  - 73% accounts prefer using mobile to login. However, we would emphasize on only that the more customers login the better. Hence, ensure that the website and mobile application is maintained, up and running smoothly and invest in technology and software to make them stronger, robust and richer.
  - Even Sign Up and subscriptions should be easy to execute.
  - Ensure all payment modes can be used safely and securely and keep customers informed about the steps the business is taking to build trust and confidence. Train staff to guide customers with any queries.

- **Account_user_count:** No of Users per account
  - This is variable was observed as important across 70% of the all models we built.
  - Accounts that have churned had an average of 4 users per account and most accounts have minimum 4 users itself.
  - Too many users in an account can also be a problem. Hence, it is advisable to limit users per account or avoid having multiple users tagged to one account completely to maximize chances of retention.
  - Also, as mentioned in the problem itself, in case of one account churn, multiple users are lost. Hence, provide opportunity for non-primary users to create at least two accounts to use as back up or re-join.

- Gender, CC_Contacted_LY, Rev_growth_yoy, Cashback and Coupon_used_for_payment were not observed as important features for this and rest of the models built as well.

Appendix

Raw Codes & Outputs

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

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

```
data.tail()
```

```
print('The number of rows (observations) is',data.shape[0],'\n''The number of columns (variables) is',data.shape[1])
```

```
data.info()
```

Check Missing Values in dataset

```
data.isnull().sum().sort_values(ascending=False)
```

```
data.isnull().sum().sum()
```

```
2676
```

```
data.size
```

```
213940
```

```
round((2676/213940)*100,2)
```

```
1.25
```

Checking Duplicates

```
data.duplicated().sum()
```

```
0
```

Summary

```
pd.options.display.float_format = '{:.2f}'.format
data.describe(include='all').T
```

Check Proportion of Target Variable (Churn)

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

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

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

```
0   0.83
1   0.17
Name: Churn, dtype: float64
```

EDA

```
data.Tenure.value_counts()
```

```
data.Tenure.unique()
```

```
array([4, 0, 2, 13, 11, '#', 9, 99, 19, 20, 14, 8, 26, 18, 5, 30, 7, 1,
       23, 3, 29, 6, 28, 24, 25, 16, 10, 15, 22, nan, 27, 12, 21, 17, 50,
       60, 31, 51, 61], dtype=object)
```

```
data.Tenure = data.Tenure.replace('#', np.NaN)
```

```
data.Tenure.unique()
```

```
array([ 4.,  0.,  2., 13., 11., nan,  9., 99., 19., 20., 14.,  8., 26.,
       18.,  5., 30.,  7.,  1., 23.,  3., 29.,  6., 28., 24., 25., 16.,
       10., 15., 22., 27., 12., 21., 17., 50., 60., 31., 51., 61.])
```

```
data.Tenure.isnull().sum()
```

```
218
```

```
data.City_Tier.value_counts()
```

```
1.00    7263
3.00    3405
2.00     480
```

```
data.City_Tier.unique()
```

```
array([ 3.,  1., nan,  2.])
```

```
data.City_Tier.isnull().sum()
```

```
112
```

```
data.CC_Contacted_LY.value_counts()
```

```
data.CC_Contacted_LY.unique()
```

```
array([  6.,   8.,  30.,  15.,  12.,  22.,  11.,   9.,  31.,  18.,  13.,
        20.,  29.,  28.,  26.,  14.,  10.,  25.,  27.,  17.,  23.,  33.,
        19.,  35.,  24.,  16.,  32.,  21.,  nan,  34.,   5.,   4., 126.,
         7.,  36., 127.,  42.,  38.,  37.,  39.,  40.,  41., 132.,  43.,
       129.])
```

```
data.CC_Contacted_LY.isnull().sum()
```

```
102
```

```
data.Payment.value_counts()
```

```
Debit Card          4587
Credit Card         3511
E wallet            1217
Cash on Delivery    1014
UPI                  822
Name: Payment, dtype: int64
```

```
data.Payment.unique()
```

```
array(['Debit Card', 'UPI', 'Credit Card', 'Cash on Delivery', 'E wallet',
       nan], dtype=object)
```

```
data.Payment.isnull().sum()
```

```
109
```

```
data.Gender.value_counts()
```

```
Male      6328
Female    4178
M          376
F          270
Name: Gender, dtype: int64
```

```
data.Gender = data.Gender.replace("M", 'Male').replace("F", 'Female')
```

```
data.Gender.unique()
```

```
array(['Female', 'Male', nan], dtype=object)
```

```
data.Gender.isnull().sum()
```

```
108
```

```
data.Service_Score.value_counts()
```

```
3.00    5490
2.00    3251
4.00    2331
1.00      77
0.00       8
5.00       5
Name: Service_Score, dtype: int64
```

```
data.Service_Score.unique()
```

```
array([ 3.,  2.,  1., nan,  0.,  4.,  5.])
```

```
data.Service_Score.isnull().sum()
```

```
98
```

```
data.Account_user_count.value_counts()
```

```
4    4569
3    3261
5    1699
2     526
1     446
@     332
6     315
Name: Account_user_count, dtype: int64
```

```
data.Account_user_count.unique()
```

```
array([3, 4, nan, 5, 2, '@', 1, 6], dtype=object)
```

```
data.Account_user_count.isnull().sum()
```

```
112
```

```
data.Account_user_count = data.Account_user_count.replace('@', np.NaN)
```

```
data.Account_user_count.unique()
```

```
array([ 3.,  4., nan,  5.,  2.,  1.,  6.])
```

```
data.Account_user_count.isnull().sum()
```

```
444
```

```
data.account_segment.value_counts()
```

```
Super           4062
Regular Plus    3862
HNI             1639
Super Plus       771
Regular          520
Regular +        262
Super +           47
Name: account_segment, dtype: int64
```

```
data.account_segment = data.account_segment.replace("Regular +", 'Regular Plus').replace("Super +", 'Super Plus')
```

```
data.account_segment.unique()
```

```
array(['Super', 'Regular Plus', 'Regular', 'HNI', nan, 'Super Plus'],
      dtype=object)
```

```
data.account_segment.isnull().sum()
```

```
97
```

```
data.CC_Agent_Score.value_counts()
```

```
3.00    3360
1.00    2302
5.00    2191
4.00    2127
2.00    1164
Name: CC_Agent_Score, dtype: int64
```

```
data.CC_Agent_Score.unique()
```

```
array([ 2.,  3.,  5.,  4., nan,  1.])
```

```
data.CC_Agent_Score.isnull().sum()
```

```
116
```

```
data.Marital_Status.value_counts()
```

```
Married     5860
Single      3520
Divorced    1668
Name: Marital_Status, dtype: int64
```

```
data.Marital_Status.unique()
```

```
array(['Single', 'Divorced', 'Married', nan], dtype=object)
```

```
data.Marital_Status.isnull().sum()
```

```
212
```

```
data.rev_per_month.value_counts()
```

```
data.rev_per_month.unique()
```

```
array([9, 7, 6, 8, 3, 2, 4, 10, 1, 5, '+', 130, nan, 19, 139, 102, 120,
       138, 127, 123, 124, 116, 21, 126, 134, 113, 114, 108, 140, 133,
       129, 107, 118, 11, 105, 20, 119, 121, 137, 110, 22, 101, 136, 125,
       14, 13, 12, 115, 23, 122, 117, 131, 104, 15, 25, 135, 111, 109,
       100, 103], dtype=object)
```

```
data.rev_per_month.isnull().sum()
```

```
102
```

```
data.rev_per_month = data.rev_per_month.replace('+', np.NaN)
```

```
data.rev_per_month.unique()
```

```
array([  9.,   7.,   6.,   8.,   3.,   2.,   4.,  10.,   1.,   5.,  nan,
       130.,  19., 139., 102., 120., 138., 127., 123., 124., 116.,  21.,
       126., 134., 113., 114., 108., 140., 133., 129., 107., 118.,  11.,
       105.,  20., 119., 121., 137., 110.,  22., 101., 136., 125.,  14.,
        13.,  12., 115.,  23., 122., 117., 131., 104.,  15.,  25., 135.,
       111., 109., 100., 103.])
```

```
data.rev_per_month.isnull().sum()
```

```
791
```

```
data.Complain_ly.value_counts()
```

```
0.00    7792
1.00    3111
Name: Complain_ly, dtype: int64
```

```
data.Complain_ly.unique()
```

```
array([ 1.,  0., nan])
```

```
data.Complain_ly.isnull().sum()
```

```
357
```

```
data.rev_growth_yoy.value_counts()
```

```
data.rev_growth_yoy.unique()
```

```
array([11, 15, 14, 23, 22, 16, 12, 13, 17, 18, 24, 19, 20, 21, 25, 26,
       '$', 4, 27, 28], dtype=object)
```

```
data.rev_growth_yoy = data.rev_growth_yoy.replace('$', np.NaN)
```

```
data.rev_growth_yoy.unique()
```

```
array([11., 15., 14., 23., 22., 16., 12., 13., 17., 18., 24., 19., 20.,
       21., 25., 26., nan,  4., 27., 28.])
```

```
data.rev_growth_yoy.isnull().sum()
```

```
3
```

```
data.coupon_used_for_payment.value_counts()
```

```
data.coupon_used_for_payment.unique()
```

```
array([1, 0, 4, 2, 9, 6, 11, 7, 12, 10, 5, 3, 13, 15, 8, '#', '$', 14,
       '*', 16], dtype=object)
```

```
data.coupon_used_for_payment.isnull().sum()
```

```
0
```

```
data.coupon_used_for_payment = data.coupon_used_for_payment.replace('#', np.NaN).replace('$', np.NaN).replace('*', np.NaN)
```

```
data.coupon_used_for_payment.unique()
```

```
array([ 1.,  0.,  4.,  2.,  9.,  6., 11.,  7., 12., 10.,  5.,  3., 13.,
       15.,  8., nan, 14., 16.])
```

```
data.rev_growth_yoy.isnull().sum()
```

```
3
```

```
data.Day_Since_CC_connect.value_counts()
```

```
data.Day_Since_CC_connect.unique()
```

```
array([5, 0, 3, 7, 2, 1, 8, 6, 4, 15, nan, 11, 10, 9, 13, 12, 17, 16, 14,
       30, '$', 46, 18, 31, 47], dtype=object)
```

```
data.Day_Since_CC_connect.isnull().sum()
```

```
357
```

```
data.Day_Since_CC_connect = data.Day_Since_CC_connect.replace('$', np.NaN)
```

```
data.Day_Since_CC_connect.unique()
```

```
array([ 5.,  0.,  3.,  7.,  2.,  1.,  8.,  6.,  4., 15., nan, 11., 10.,
        9., 13., 12., 17., 16., 14., 30., 46., 18., 31., 47.])
```

```
data.Day_Since_CC_connect.isnull().sum()
```

```
358
```

```
data.cashback.value_counts()
```

```
data.cashback.unique()
```

```
array([159.93, 120.9, nan, ..., 227.36, 226.91, 191.42], dtype=object)
```

```
data.cashback.isnull().sum()
```

```
471
```

```
data.cashback = data.cashback.replace('$', np.NaN)
```

```
data.cashback.unique()
```

```
array([159.93, 120.9 ,    nan, ..., 227.36, 226.91, 191.42])
```

```
data.cashback.isnull().sum()
```

```
473
```

```
data.Login_device.value_counts()
```

```
Mobile      7482
Computer    3018
&&&&         539
Name: Login_device, dtype: int64
```

```
data.Login_device.unique()
```

```
array(['Mobile', 'Computer', '&&&&', nan], dtype=object)
```

```
data.Login_device.isnull().sum()
```

```
221
```

```
data.Login_device = data.Login_device.replace('&&&&', np.NaN)
```

```
data.Login_device.unique()
```

```
array(['Mobile', 'Computer', nan], dtype=object)
```

```
data.Login_device.isnull().sum()
```

```
760
```

```
data.isnull().sum().sum()
```

```
4361
```

```
data.size
```

```
213940
```

```
round((4361/213940)*100,2)
```

```
2.04
```

```
data.isnull().sum().sort_values(ascending = False)/data.index.size
```

```
rev_per_month             0.07
Login_device              0.07
cashback                  0.04
Account_user_count        0.04
Day_Since_CC_connect      0.03
Complain_ly               0.03
Tenure                    0.02
Marital_Status            0.02
CC_Agent_Score            0.01
City_Tier                 0.01
Payment                   0.01
Gender                    0.01
CC_Contacted_LY           0.01
Service_Score             0.01
account_segment           0.01
rev_growth_yoy            0.00
coupon_used_for_payment   0.00
Churn                     0.00
AccountID                 0.00
dtype: float64
```

Impute Missing Values

```
sns.boxplot(x='Tenure', data=data)
```

```
data.Tenure.median()
```

```
9.0
```

```
data[data.Tenure == 9].shape[0]
```

```
496
```

```
data.Tenure.isnull().sum()
```

```
218
```

```
data.Tenure = data.Tenure.fillna(data.Tenure.median())
data[data.Tenure.isnull()]
```

AccountID	Churn	Tenure	City_Tier	CC_Contacted_LY	Payment	Gender	Service_Score	Account_user_count	account_segment	CC_Agent_Score	Marital_St

```
data[data.Tenure == 9].shape[0]
```

```
714
```

```
data.Tenure = data.Tenure.astype('int64')
```

```
data.Tenure.dtype
```

```
dtype('int64')
```

```
sns.boxplot(x='CC_Contacted_LY', data=data)
```

```
data.Account_user_count.median()
```

```
4.0
```

```
data[data.Account_user_count == 4].shape[0]
```

```
4569
```

```
data.Account_user_count.isnull().sum()
```

```
444
```

```
data.Account_user_count = data.Account_user_count.fillna(data.Account_user_count.median())
data[data.Account_user_count.isnull()]
```

```
data[data.Account_user_count == 4].shape[0]
```

5013

```
data.Account_user_count = data.Account_user_count.astype('int64')
```

```
data.Account_user_count.dtype
```

dtype('int64')

```
sns.boxplot(x='rev_per_month', data=data)
```

```
data.rev_per_month.median()
```

5.0

```
data[data.rev_per_month == 5].shape[0]
```

1337

```
data.rev_per_month.isnull().sum()
```

791

```
data.rev_per_month = data.rev_per_month.fillna(data.rev_per_month.median())
data[data.rev_per_month.isnull()]
```

```
data[data.rev_per_month == 5].shape[0]
```

2128

```
data.rev_per_month = data.rev_per_month.astype('int64')
```

```
data.rev_per_month.dtype
```

dtype('int64')

```
sns.boxplot(x='rev_growth_yoy', data=data)
```

```
data.rev_growth_yoy.median()
```

15.0

```
data[data.rev_growth_yoy == 15].shape[0]
```

1283

```
data.rev_growth_yoy.isnull().sum()
```

3

```
data.rev_growth_yoy = data.rev_growth_yoy.fillna(data.rev_growth_yoy.median())
data[data.rev_growth_yoy.isnull()]
```

AccountID	Churn	Tenure	City_Tier	CC_Contacted_LY	Payment	Gender	Service_Score	Account_user_count	account_segment	CC_Agent_Score	Marital_St

```
data[data.rev_growth_yoy == 15].shape[0]
```

1286

```
data.rev_growth_yoy = data.rev_growth_yoy.astype('int64')
```

```
data.rev_growth_yoy.dtype
```

dtype('int64')

```
sns.boxplot(x='coupon_used_for_payment', data=data)
```

```
data.coupon_used_for_payment.median()
```

```
1.0
```

```
data[data.coupon_used_for_payment == 1].shape[0]
```

```
4373
```

```
data.coupon_used_for_payment.isnull().sum()
```

```
3
```

```
data.coupon_used_for_payment = data.coupon_used_for_payment.fillna(data.coupon_used_for_payment.median())
data[data.coupon_used_for_payment.isnull()]
```

AccountID	Churn	Tenure	City_Tier	CC_Contacted_LY	Payment	Gender	Service_Score	Account_user_count	account_segment	CC_Agent_Score	Marital_St

```
data[data.coupon_used_for_payment == 1].shape[0]
```

```
4376
```

```
data.coupon_used_for_payment = data.coupon_used_for_payment.astype('int64')
```

```
data.coupon_used_for_payment.dtype
```

```
dtype('int64')
```

```
sns.boxplot(x='Day_Since_CC_connect', data=data)
```

```
data.Day_Since_CC_connect.median()
```

```
3.0
```

```
data[data.Day_Since_CC_connect == 3].shape[0]
```

```
1816
```

```
data.Day_Since_CC_connect.isnull().sum()
```

```
358
```

```
data.Day_Since_CC_connect = data.Day_Since_CC_connect.fillna(data.Day_Since_CC_connect.median())
data[data.Day_Since_CC_connect.isnull()]
```

AccountID	Churn	Tenure	City_Tier	CC_Contacted_LY	Payment	Gender	Service_Score	Account_user_count	account_segment	CC_Agent_Score	Marital_St

```
data[data.Day_Since_CC_connect == 3].shape[0]
```

```
2174
```

```
data.Day_Since_CC_connect = data.Day_Since_CC_connect.astype('int64')
```

```
data.Day_Since_CC_connect.dtype
```

```
dtype('int64')
```

```
sns.boxplot(x='cashback', data=data)
```

```
data.cashback.median()
```

```
165.25
```

```
data.cashback.isnull().sum()
```

```
473
```

```
data.cashback = data.cashback.fillna(data.cashback.median())
data[data.cashback.isnull()]
```

AccountID	Churn	Tenure	City_Tier	CC_Contacted_LY	Payment	Gender	Service_Score	Account_user_count	account_segment	CC_Agent_Score	Marital_St

```
data.cashback.dtype
```

```
dtype('float64')
```

```
data.info()
```

```
data.City_Tier.isnull().sum()
```

```
112
```

```
data.City_Tier.mode()
```

```
0   1.00
dtype: float64
```

```
data[data.City_Tier == 1].shape[0]
```

```
7263
```

```
data.City_Tier = data.City_Tier.fillna(data.City_Tier.mode()[0])
data[data.City_Tier.isnull()]
```

AccountID	Churn	Tenure	City_Tier	CC_Contacted_LY	Payment	Gender	Service_Score	Account_user_count	account_segment	CC_Agent_Score	Marital_St

```
data[data.City_Tier == 1].shape[0]
```

```
7375
```

```
data.City_Tier = data.City_Tier.astype('int64')
```

```
data.City_Tier.dtype
```

```
dtype('int64')
```

```
data.Service_Score.isnull().sum()
```

```
98
```

```
data.Service_Score.mode()
```

```
0   3.00
dtype: float64
```

```
data[data.Service_Score == 3].shape[0]
```

```
5490
```

```
data.Service_Score = data.Service_Score.fillna(data.Service_Score.mode()[0])
data[data.Service_Score.isnull()]
```

```
data[data.Service_Score == 3].shape[0]
```

```
5588
```

```
data.Service_Score = data.Service_Score.astype('int64')
```

```
data.Service_Score.dtype
```

```
dtype('int64')
```

```
data.CC_Agent_Score.isnull().sum()
```

```
116
```

```
data.CC_Agent_Score.mode()
```

```
0   3.00
dtype: float64
```

```
data[data.CC_Agent_Score == 3].shape[0]
```

```
3360
```

```
data.CC_Agent_Score = data.CC_Agent_Score.fillna(data.CC_Agent_Score.mode()[0])
data[data.CC_Agent_Score.isnull()]
```

AccountID	Churn	Tenure	City_Tier	CC_Contacted_LY	Payment	Gender	Service_Score	Account_user_count	account_segment	CC_Agent_Score	Marital_St

```
data[data.CC_Agent_Score == 3].shape[0]
```

```
3476
```

```
data.CC_Agent_Score = data.CC_Agent_Score.astype('int64')
```

```
data.CC_Agent_Score.dtype
```

```
dtype('int64')
```

```
data.Complain_ly.isnull().sum()
```

```
357
```

```
data.Complain_ly.mode()
```

```
0   0.00
dtype: float64
```

```
data[data.Complain_ly == 0].shape[0]
```

```
7792
```

```
data.Complain_ly = data.Complain_ly.fillna(data.Complain_ly.mode()[0])
data[data.Complain_ly.isnull()]
```

AccountID	Churn	Tenure	City_Tier	CC_Contacted_LY	Payment	Gender	Service_Score	Account_user_count	account_segment	CC_Agent_Score	Marital_St

```
data[data.Complain_ly == 0].shape[0]
```

```
8149
```

```
data.Complain_ly = data.Complain_ly.astype('int64')
```

```
data.Complain_ly.dtype
```

```
dtype('int64')
```

```
data.Payment.isnull().sum()
```

```
109
```

```
data.Payment.mode()
```

```
0    Debit Card
dtype: object
```

```
data.Payment.describe()
```

```
data.Payment = data.Payment.fillna(data.Payment.mode()[0])
data[data.Payment.isnull()]
```

```
data.Gender.isnull().sum()
```

```
108
```

```
data.Gender.mode()
```

```
0    Male
dtype: object
```

```
data.Gender.describe()
```

```
data.Gender = data.Gender.fillna(data.Gender.mode()[0])
data[data.Gender.isnull()]
```

```
data.account_segment.isnull().sum()
```

```
97
```

```
data.account_segment.mode()
```

```
0    Regular Plus
dtype: object
```

```
data.account_segment.describe()
```

```
data.account_segment = data.account_segment.fillna(data.account_segment.mode()[0])
data[data.account_segment.isnull()]
```

```
data.Marital_Status.isnull().sum()
```

```
212
```

```
data.Marital_Status.mode()
```

```
0    Married
dtype: object
```

```
data.Marital_Status = data.Marital_Status.fillna(data.Marital_Status.mode()[0])
data[data.Marital_Status.isnull()]
```

```
data.Login_device.isnull().sum()
```

```
760
```

```
data.Login_device.mode()
```

```
0    Mobile
dtype: object
```

```
data.Login_device.describe()
```

```
data.Login_device = data.Login_device.fillna(data.Login_device.mode()[0])
data[data.Login_device.isnull()]
```

```
data.Login_device.describe()
```

```
data.info()
```

```
data.isnull().sum()
```

```
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
```

Univariate Analysis (Numerical Variables)

```
cont = data.select_dtypes(include = ['float64', 'int64'])
lstnumericcolumns = list(cont.columns.values)
len(lstnumericcolumns)
```

```
14
```

```
cont = data.select_dtypes(include = ['float64', 'int64'])
cols = list(cont.columns)
for col in cols:
    print(col)
    print('Skew:', np.round(data[col].skew(),2))
    plt.figure(figsize=(25,6))
    plt.subplot(1,2,1)
    sns.distplot(data[col],norm_hist=False,kde=False,hist_kws=dict(edgecolor='black',linewidth=1.5))
    plt.vlines(data[col].mean(),ymin=0, ymax=100, color = 'red', linewidth=3)
    plt.vlines(data[col].median(),ymin=0, ymax=100, color = 'orange', linewidth=3)
    plt.ylabel('count')
    plt.subplot(1,2,2)
    sns.boxplot(data[col])
    plt.show()
```

```
plt.figure(figsize=(20,3))
sns.countplot(x='Tenure', data=data, palette='pastel')
```

```
plt.figure(figsize=(20,3))
sns.countplot(x='CC_Contacted_LY', data=data, palette='pastel')
```

```
plt.figure(figsize=(5,3))
sns.countplot(x='Account_user_count', data=data, palette='pastel')
```

```
plt.figure(figsize=(20,3))
sns.countplot(x='rev_per_month', data=data, palette='pastel')
```

```
plt.figure(figsize=(20,3))
sns.countplot(x='rev_growth_yoy', data=data, palette='pastel')
```

```
plt.figure(figsize=(10,3))
sns.countplot(x='coupon_used_for_payment', data=data, palette='pastel')
```

```
plt.figure(figsize=(10,3))
sns.countplot(x='Day_Since_CC_connect', data=data, palette='pastel')
```

```
plt.figure(figsize=(5,3))
sns.countplot(x='City_Tier', data=data, palette='pastel')
```

```
plt.figure(figsize=(7,3))
sns.countplot(x='Payment', data=data, palette='pastel')
```

```
plt.figure(figsize=(5,3))
sns.countplot(x='Gender', data=data, palette='pastel')
```

```
plt.figure(figsize=(5,3))
sns.countplot(x='Service_Score', data=data, palette='pastel')
```

```
plt.figure(figsize=(5,3))
sns.countplot(x='account_segment', data=data, palette='pastel')
```

```
plt.figure(figsize=(5,3))
sns.countplot(x='CC_Agent_Score', data=data, palette='pastel')
```

```
plt.figure(figsize=(5,3))
sns.countplot(x='Marital_Status', data=data, palette='pastel')
```

```
plt.figure(figsize=(5,3))
sns.countplot(x='Complain_ly', data=data, palette='pastel')
```

```
plt.figure(figsize=(5,3))
sns.countplot(x='Login_device', data=data, palette='pastel')
```

Bivariate Analysis (Numerical Variables)

```
sns.pairplot(data, hue ='Churn')
```

```
data_plot = data[['Tenure', 'Churn', 'CC_Contacted_LY', 'Account_user_count', 'rev_per_month',
                  'rev_growth_yoy', 'coupon_used_for_payment','Day_Since_CC_connect','cashback']]
data_plot.head()
```

```
sns.pairplot(data_plot, hue ='Churn')
```

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

Bivariate Analysis (Categorical Variables)

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

Multivariate Analysis (HeatMap)

```
plt.figure(figsize=(25,10))
sns.heatmap(data.corr(),annot=True,fmt=".2f", cmap='Blues')
plt.title("Correlation Heatmap")
plt.xticks(rotation=45)
plt.show()
```

Encoding the Object Variables (using Label Encoding)

```
for feature in data.columns:
    if data[feature].dtype == 'object':
        print('\n')
        print('feature:',feature)
        print(pd.Categorical(data[feature].unique()))
        print(pd.Categorical(data[feature].unique()).codes)
        data[feature] = pd.Categorical(data[feature]).codes
```

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

```
data.info()
```

Outlier Treatment

```
data[['Tenure', 'CC_Contacted_LY', 'Account_user_count', 'rev_per_month', 'rev_growth_yoy', 'coupon_used_for_payment',
      'Day_Since_CC_connect','cashback']].boxplot(figsize=(15,5))
plt.title("Boxplot")
plt.xticks(rotation=45)
plt.show()
```

```
Q1 = data.quantile(0.05)
Q3 = data.quantile(0.95)
IQR = Q3 - Q1
pd.DataFrame((((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR))).sum()/data.shape[0]*100))
```

Split Data Keeping Outliers (xo & yo)

```
xo = data.drop(['Churn','AccountID'], axis = 1)
yo = data['Churn']
```

```
xo.shape
```

```
(11260, 17)
```

```
yo.shape
```

```
(11260,)
```

Split Data to Remove Outliers (x & y)

```
x = data.drop(['Churn','AccountID'], axis = 1)
y = data['Churn']
```

```
x.shape
```

```
(11260, 17)
```

```
y.shape
```

```
(11260,)
```

```
def remove_outlier(col):
    sorted(col)
    Q1,Q3=col.quantile([0.05,0.95])
    IQR=Q3-Q1
    lower_range= Q1-(1.5 * IQR)
    upper_range= Q3+(1.5 * IQR)
    return lower_range, upper_range
```

```
lw,up=remove_outlier(x['Tenure'])
x['Tenure']=np.where(x['Tenure']>up,up,x['Tenure'])
x['Tenure']=np.where(x['Tenure']<lw,lw,x['Tenure'])

lw,up=remove_outlier(x['CC_Contacted_LY'])
x['CC_Contacted_LY']=np.where(x['CC_Contacted_LY']>up,up,x['CC_Contacted_LY'])
x['CC_Contacted_LY']=np.where(x['CC_Contacted_LY']<lw,lw,x['CC_Contacted_LY'])

lw,up=remove_outlier(x['rev_per_month'])
x['rev_per_month']=np.where(x['rev_per_month']>up,up,x['rev_per_month'])
x['rev_per_month']=np.where(x['rev_per_month']<lw,lw,x['rev_per_month'])

lw,up=remove_outlier(x['coupon_used_for_payment'])
x['coupon_used_for_payment']=np.where(x['coupon_used_for_payment']>up,up,x['coupon_used_for_payment'])
x['coupon_used_for_payment']=np.where(x['coupon_used_for_payment']<lw,lw,x['coupon_used_for_payment'])

lw,up=remove_outlier(x['Day_Since_CC_connect'])
x['Day_Since_CC_connect']=np.where(x['Day_Since_CC_connect']>up,up,x['Day_Since_CC_connect'])
x['Day_Since_CC_connect']=np.where(x['Day_Since_CC_connect']<lw,lw,x['Day_Since_CC_connect'])

lw,up=remove_outlier(x['cashback'])
x['cashback']=np.where(x['cashback']>up,up,x['cashback'])
x['cashback']=np.where(x['cashback']<lw,lw,x['cashback'])

x.shape
```

```
(11260, 17)
```

```
x[['Tenure', 'CC_Contacted_LY', 'rev_per_month', 'coupon_used_for_payment','Day_Since_CC_connect',
   'cashback']].boxplot(figsize=(15,5))
plt.title("Boxplot Post Outlier Treatment")
plt.xticks(rotation=45)
plt.show()
```

K-Means Clustering

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

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

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

```
df_scaled
```

Calculating WSS for values of K - Elbow Method

```
wss =[]
```

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

```
wss
```

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

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

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

```
0.12336437604603798
```

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

```
0.0013580240285513245
```

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

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

```
0.0742940376682887
```

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

```
-0.07283738766512156
```

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

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

```
0.07424420609512918
```

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

```
-0.17533947454568208
```

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

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

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

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

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

```
df.to_csv('df_Capstone_Kmeans.csv',index=False)
```

PROJECT NOTE 2

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

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

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

Train Test Split (xo & yo)

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

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

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

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

Train Test Split (x & y)

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

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

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

Scaling the variables

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

```
sc = StandardScaler()
```

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

```
x_scaled_train
```

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

```
x_scaled_test
```

SMOTE

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

Model Building

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

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

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

Prediction on Train set ¶

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

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

Prediction on Test set

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

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

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

Prediction on Train set

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

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

Prediction on Test set

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

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

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

```
x.columns
```

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

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

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

```
model_1.summary()
```

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

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

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

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

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

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

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

Checking the Variance Inflation Factor

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(X):

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

    return(vif)
```

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

Prediction on the Train Set

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

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

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

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

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

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

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

Prediction on Test set

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

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

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

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

Choosing the optimal threshold

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

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

Validating on the train set with revised threshold

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

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

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

Validating on the test set

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

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

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

Logistic Regression Model_3 with 0.19 threshold

Validating on the train set with revised threshold - 0.19

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

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

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

Validating on the test set

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

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

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

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

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

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

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

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

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

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

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

```
a.columns
```

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

```
a.shape
```

```
(11260, 10)
```

```
x.shape
```

```
(11260, 17)
```

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

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

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

Prediction on Logistic Regression Model_4 Train Set with 0.19 threshold

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

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

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

```
from sklearn.metrics import accuracy_score

score =accuracy_score(y_train,y_class_pred)
```

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

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

Prediction on Test set

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

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

```
from sklearn.metrics import accuracy_score

score =accuracy_score(y_test,y_class_pred)
```

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

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

Choosing the optimal threshold

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

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

```
0.21996947594614671
```

Logistic Regression Model_4 with 0.219 threshold

Validating on the train set with revised threshold

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

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

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

Validating on the test set

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

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

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

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

```
from sklearn.ensemble import RandomForestClassifier

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

Prediction on Train set

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

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

Prediction on Test set ¶

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

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

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

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

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

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

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

rf_model3 = RandomForestClassifier(random_state=1)

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

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

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

```
grid_search.best_params_
```

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

```
best_grid = grid_search.best_estimator_
```

Prediction on Train set

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

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

Prediction on Test set

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

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

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

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

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

rf_model4 = RandomForestClassifier(random_state=1)

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

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

```
grid_search.best_params_
```

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

```
best_grid = grid_search.best_estimator_
```

Prediction on Train set

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

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

Prediction on Test set

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

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

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

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

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

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

rf_model5 = RandomForestClassifier(random_state=1)

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

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

```
grid_search.best_params_
```

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

```
best_grid = grid_search.best_estimator_
```

Prediction on Train set

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

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

Prediction on Test set

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

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

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

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

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

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

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

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

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

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

Prediction on Train set

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

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

Prediction on Test set

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

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

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

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

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

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

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

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

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

rf_model6 = RandomForestClassifier(random_state=1)

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

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

```
grid_search.best_params_
```

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

```
best_grid = grid_search.best_estimator_
```

Prediction on Train set ¶

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

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

Prediction on Test set

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
from sklearn.neighbors import KNeighborsClassifier

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

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

Prediction on Train set

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

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

Prediction on Test set

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

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

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

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

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

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

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

Prediction on Train set

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

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

Prediction on Test set

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

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

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

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

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

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

```
from sklearn import svm
```

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

svm.fit(x_scaled_train, y_train)
```

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

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

Prediction on Train set

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

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

Prediction on Test set

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

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

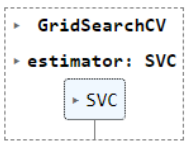
SVM Tuned Model (scaled dataset without Outliers - x_scaled_train & x_test) ¶

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

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

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

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




```
grid_search.best_params_
```

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

```
best_grid = grid_search.best_estimator_
```

Prediction on Train set

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

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

Prediction on Test set

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

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

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

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

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

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

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

Prediction on Train set

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

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

Prediction on Test set

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

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

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

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

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

CART (Unscaled Dataset without Outliers - x_train & x_test)

```
from sklearn import tree


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

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

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

Prediction on Train set

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

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

Prediction on Test set

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

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

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

```
from sklearn import tree

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

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

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

Prediction on Train set

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

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

Prediction on Test set ¶

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

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

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

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

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

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

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

Emsemble Techniques

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

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

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

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

Prediction on Train set

```
y_train_predict = Bagging_model.predict(x_train)
model_score = Bagging_model.score(x_train, y_train)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_train, y_train_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_train, y_train_predict))
# predict probabilities
probs = Bagging_model.predict_proba(x_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

Prediction on Test set

```
y_test_predict = Bagging_model.predict(x_test)
model_score = Bagging_model.score(x_test, y_test)
print('Accuracy Score is', model_score,'\n')
print('Confusion Matrix','\n', metrics.confusion_matrix(y_test, y_test_predict),'\n')
print('Classification Report','\n', metrics.classification_report(y_test, y_test_predict))
# predict probabilities
probs = Bagging_model.predict_proba(x_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr);
```

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

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

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

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

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

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

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

Prediction on Train set

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

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

Prediction on Test set

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

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

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

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

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

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

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

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

```
from sklearn.ensemble import AdaBoostClassifier

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

Prediction on Train set

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

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

Prediction on Test set

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

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

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

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

ADB_tuned = AdaBoostClassifier(random_state=1)

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

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

```
grid_search.best_params_
```

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

```
best_grid = grid_search.best_estimator_
```

Prediction on Train set

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

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

Prediction on Test set

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

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

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

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

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

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

Prediction on Train set

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

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

Prediction on Test set

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

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

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

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

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

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

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

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

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

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

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

```
grid_search.best_params_
```

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

```
best_grid = grid_search.best_estimator_
```

Prediction on Train set

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

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

Prediction on Test set

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

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

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

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

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

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

Prediction on Train set

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

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

Prediction on Test set ¶

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

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

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

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

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

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

xgb_tuned = XGBClassifier(random_state=1)

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

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

```
grid_search.best_params_
```

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

```
best_grid = grid_search.best_estimator_
```

Prediction on Train set

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

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

Prediction on Test set

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

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

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

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

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

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

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

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

THE END