CAPSTONE PROJECT

TITLE: Telecom Churn Analysis

<u>Submitted by :</u>
Group - 4

Bharath N Raju
Niranjan
Vishal Choudhary
Nikhil kodapalli
Reshma RA

<u>Guided By :</u>
Mr. Jatinder Bedi

Brief Overview

- ➢ Churn Prediction is one of the most popular Big Data use cases in Business. It consists of detecting customers who are likely to cancel a subscription a service.

- ➢ Churn is a problem for telecom industries because it is more expensive to acquire a new customer than to keep your existing from leaving.

NOTE :- Telecom Industry today measure voluntary churn by a monthly figure, such as 1.9 or 2.1 percent.

Project Objective

- To predict customer churn.

- Highlighting the main variables/factors influencing the customer churn.

- Use of various ML Classification algorithms to build prediction models, evaluate the accuracy and performance of these models.

- Finding out the best model for the given dataset.

Dataset Information

- ➢Data is taken from Kaggle (Telecom churn Dataset)
- ➢No. of features: 58
- ➢No. of records: 51047
- ➢Target Column: Churn
- ➢Redundant columns: Customer Id, Service area.
- ➢No. categorical columns : 21

Variable Name	**Variable Description**
Customer ID	Primary key of the record.
Churn	Information about Churn of the Customers.
Monthly Revenue	Revenue of each Customer
Monthly Minutes	Number of Minutes call spoken by Customer
Total Recurring Charge	The Charges for the Service
Director Assisted Calls	When we call an operator to request a telephone number
Overage Minutes	Count of Call used over duration to particular post-paid cell phone plan
Roaming Calls	The ability to get access to the Internet when away from home at the price of a local call or at a charge considerably less than the regular long-distance charges.
Three way Calls	A way of adding a third party to your conversation without the assistance of a telephone operator.
Dropped Calls	Count of Phone calls gets disconnected somehow from the cellular network.
Blocked Calls	Count of Telephone call that is unable to connect to an intended recipient.
Unanswered Calls	Count of Calling that an individual perceives but is not currently pursuing.

Variable Name	Variable Description
Received Calls	Number of calls received by the customer.
Out bound Calls	Call initiated by the call centre agent to customer on behalf of client to know the target customer behaviour and needs.
Inbound Calls	In inbound calls, call-centre or customer-care receives call from customer with issues and questions.
Peak Calls In Out	Amount of time period with fewer calls than are handled in a busy period.
Call Forwarding Calls	Count of Calls Forwarded by user.
Dropped Blocked Calls	Number of VM messages customer currently has on the server.
Call Waiting Calls	Duration of call-in waiting period
Months In Service	Number of months customer using service.
Unique Subs	subscription of different networks
Active Subs	subscriptions of the networks that are active or in usage.
Service Area	Network service area
Handset Models	Count of Handsets are used to Contact one to one.

Age HH1	User aged below 45
Age HH2	User aged above 45
Children in HH	Whether there are Children in House hold
Handset Refurbished	Are the handsets refurbished or not
Handset Web Capable	Are the handsets capable of internet connectivity
Truck Owner	Is the user a Truck Owner
RV Owner	Is the user an RV owner
Home Ownership	Is the house the user is staying, his own
Buys Visa Mail Order	Does the user buy Visa Mail order
Responds to Mail Offers	Does the user respond to Mail offers
Opt-out Mailings	Did he opt out of the mail offers sent to him

Non-US-Travel	Does the user travel to other countries
Owns-Computer	Does he have a computer or not
Has-Credit Card	Does he have a credit card or not
Retention Calls	No of Retention Calls
Retention Offers Accepted	Customers accepting retaining the retaining offers given by the company.
New Cell phone User	Number of customers buying new cell phone.
Referrals Made By Subscriber	Referrals made by the existing customer to the other customer.
Income Group	The column talks about the customer saying to which category the customer belongs to.
Owns Motorcycle	The columns ask about the customer weather the customer owns a motorcycle or not.
Adjustments To Credit Rating	Rating Scale
Handset Price	Its amount paid by the customer for his cell phone.

Exploratory Data Analysis(EDA)

i. Duplicate Values

```
#checking for duplicate values
print(df12.duplicated().sum())
print(' ')
print(f'Dataset have {df1.duplicated().sum()} duplicate values.')
```

```
0

Dataset have 0 duplicate values.
```

ii. Treating few columns which are having inappropriate data

Handset price:

```
df1['HandsetPrice'].unique()
```

```
array(['30', 'Unknown', '10', '80', '150', '300', '40', '200', '100',
       '130', '60', '400', '240', '250', '180', '500'], dtype=object)
```

```
df1[df1['HandsetPrice']=='Unknown'].shape[0]
```

```
28981
```

```
# replacing unknown with zeros
df1['HandsetPrice']=df1['HandsetPrice'].replace(to_replace='Unknown',value= 0)
```

```
df1['HandsetPrice']=df1['HandsetPrice'].replace(to_replace=0,value= np.nan)
```

```
# calculating null value percentage for Handsetprice variable
df1['HandsetPrice'].isnull().sum()/df1.shape[0]*100
```

```
56.7742820201387
```

```
# as we can see that we have more than 56% of null values ,
# hence we are dropping the column,instead of filling 56% values which are not true.
```

```
df1.drop(columns='HandsetPrice',inplace=True)
```

```
df1.shape
```

```
(51046, 54)
```

- Handset price is having unknown value , so we replace it as nan.
- 56% of null values are there in handset price.
- Hence we are dropping column instead of filling it with 56% values which are not true.

<u>Credit rating:</u>

Treating credit rating variable

```
]: df1['CreditRating'].unique()

]: array(['1-Highest', '4-Medium', '3-Good', '6-VeryLow', '2-High', '5-Low',
          '7-Lowest'], dtype=object)

]: # binning the variable with high,medium and low

]: dict_rating={ '1-Highest':'High',
           '2-High':'High',
           '3-Good':'Medium',
           '4-Medium':'Medium',
           '5-Low':'Low',
           '6-VeryLow':'Low',
           '7-Lowest':'Low'
            }

]: df1['CreditRating']=df1['CreditRating'].map(dict_rating)

]: df1['CreditRating'].unique()

]: array(['High', 'Medium', 'Low'], dtype=object)

]: # now we have only 3 categories high , medium and low
```

- In credit rating variable we binned it with high, medium and
  class labels.

Age HH1 and Age HH2:

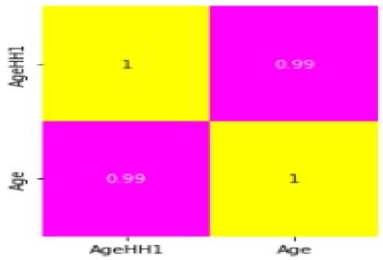
```
df1[df1['AgeHH1']==0].shape
```

(13916, 54)

```
df1[df1['AgeHH2']==0].shape
```

(26086, 54)

AgeHH1	AgeHH2	Age
62.000000	0.000000	62.000000
40.000000	42.000000	41.000000
26.000000	26.000000	26.000000
30.000000	0.000000	30.000000
46.000000	54.000000	50.000000



- We are dropping AgeHH1 and AgeHH2 since they are redundant, having the new column Age.

iii. Missing value

- Out of 56 features 13 had missing values. Age having 29.04% null values.

- 11 of them had less than 1% of missing values, hence the rows were deleted directly.

- In age null values are imputed by KNN imputer.

```
# calculating percentage of null values.
n/df1.shape[0]*100
```

```
MonthlyRevenue            0.305607
MonthlyMinutes            0.305607
TotalRecurringCharge      0.305607
DirectorAssistedCalls     0.305607
OverageMinutes            0.305607
RoamingCalls              0.305607
PercChangeMinutes         0.718959
PercChangeRevenues        0.718959
Age                      29.042432
dtype: float64
```

Imputing null values in age column using KNN Imputers

```
from sklearn.impute import KNNImputer
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
num_scaled=ss.fit_transform(num_cols)
```

```
imputer=KNNImputer(n_neighbors=1000)
df_filled=imputer.fit_transform(num_scaled)
```

```
df_filled=pd.DataFrame(df_filled,columns=num_cols.columns)
df_filled
```

- In marital status null values are replaced by KNN classifier.

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train,y_train)
y_pred = knn.predict(x_test)
```

```
y_pred.shape
```

```
(19556,)
```

```
index_list=df_mar[df_mar['MaritalStatus'].isnull()==True].index
```

```
df_mar['MaritalStatus'][index_list]=y_pred
```

```
df_mar.shape
```

```
(50679, 34)
```

```
dict_2={1:'Yes',0:'No' }
```

```
df_mar['MaritalStatus']=df_mar['MaritalStatus'].map(dict_2)
```

```
df_mar['MaritalStatus'].value_counts()
```

```
No      25388
Yes     25291
```

iv. Outlier Treatment

Inference: By Visualizing the boxplot, we can see that all the Features have potential outliers and some features there are extreme values as well.

Outliers: Outliers is an observation which deviates so much from the other observations, that it become suspicious that it was generated by different mechanism or simply by error

Extreme Values: Extreme Values is an observation with value at the boundaries of the domain

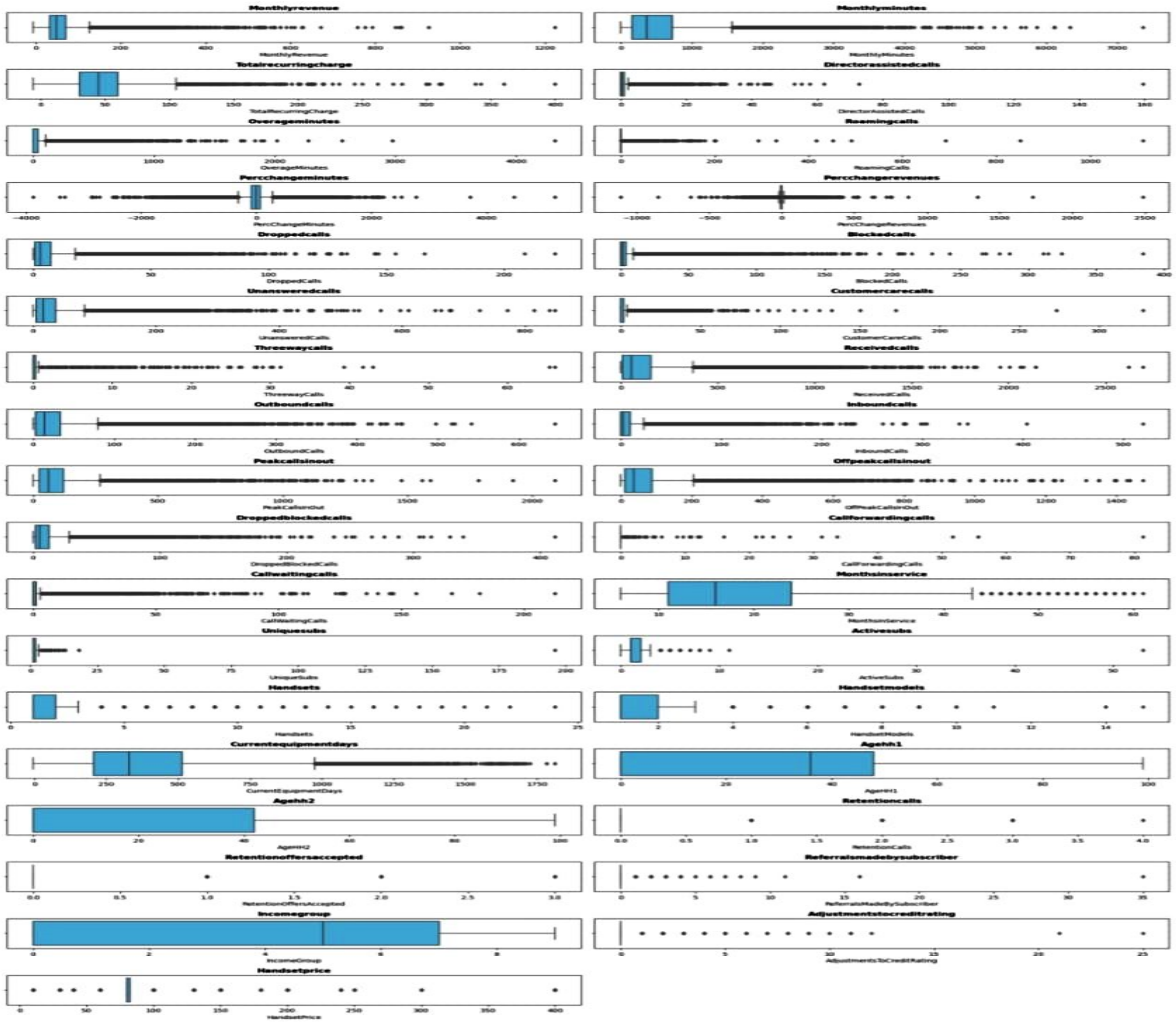
Reason for outliers exist in the data:

1. Variability in the Data
2. An experimental measurement errors

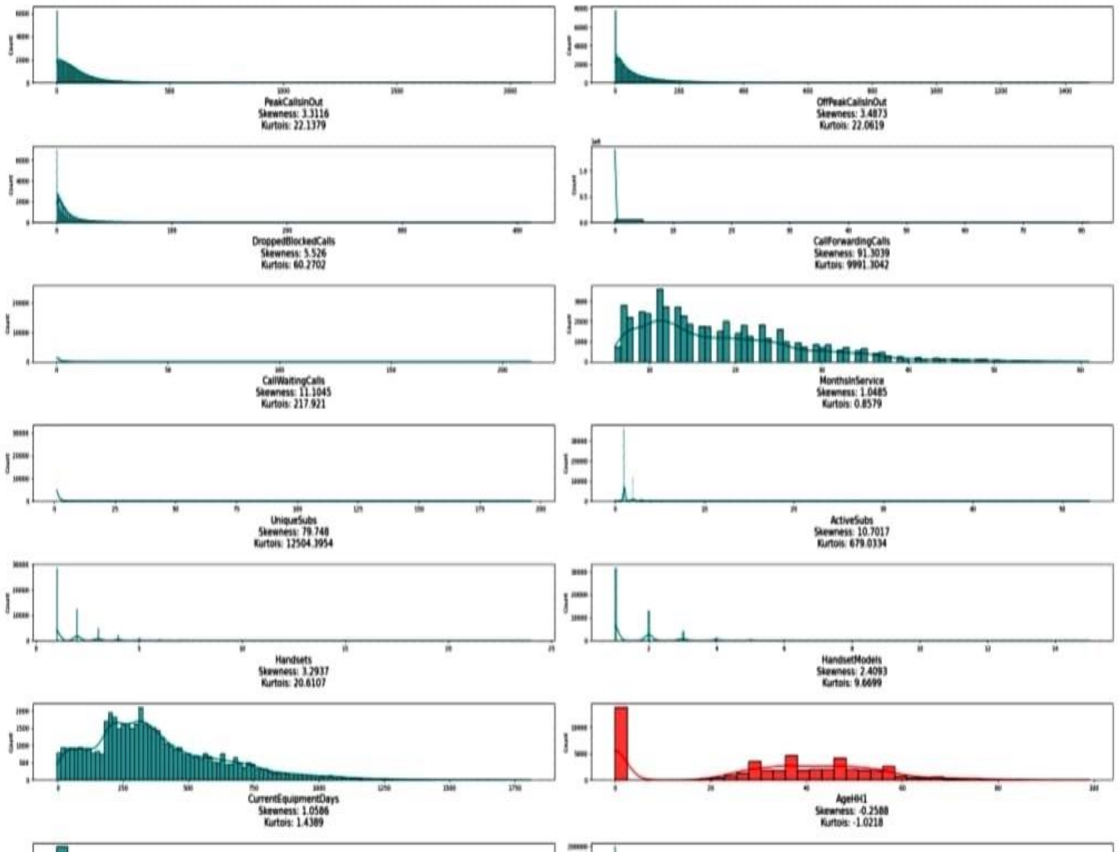
Impact of outliers on Dataset:

1. It causes various problem during statistical analysis.

It effects the mean and standard deviation



iv. Skewness

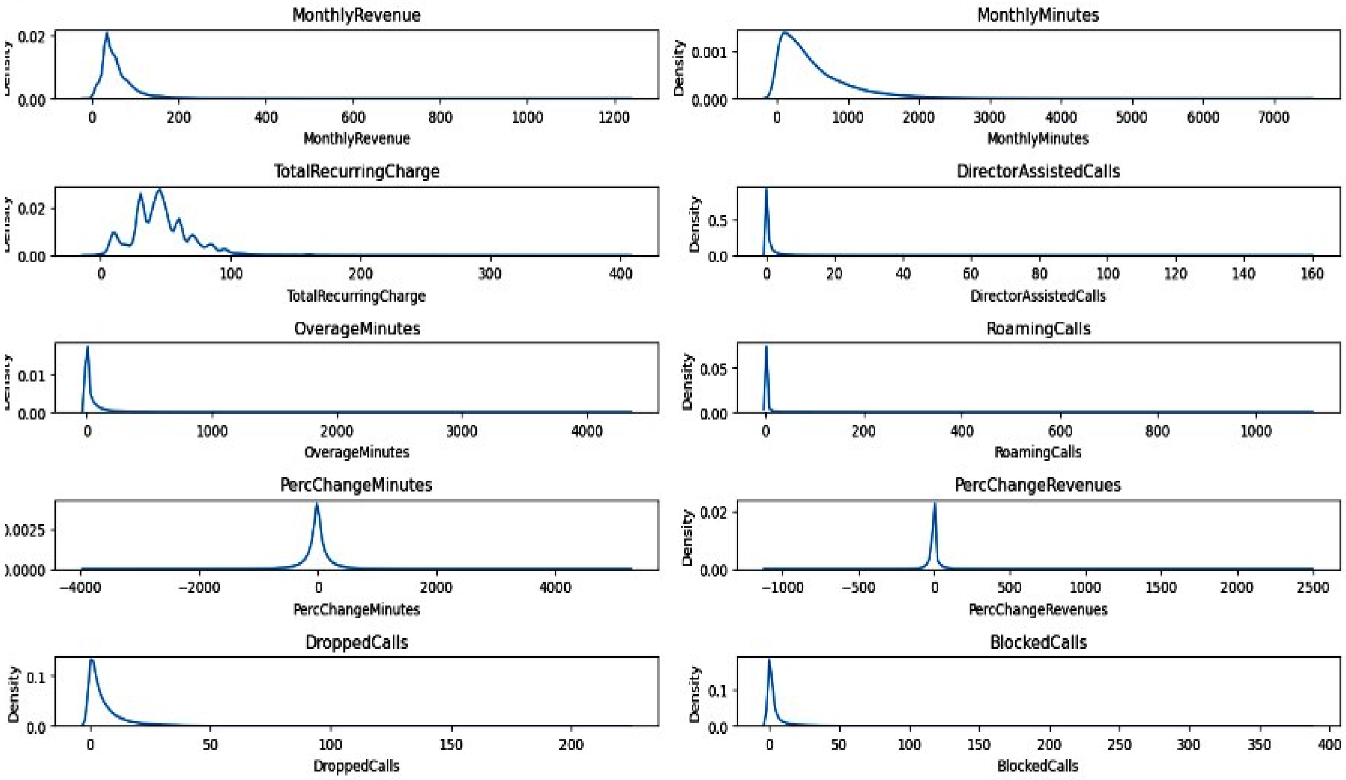


Inference: *Here by visualizing distplot we can see that the Features plotted in Teal colour are positively skewed and Features plotted in red colour are Negatively Skewed.*

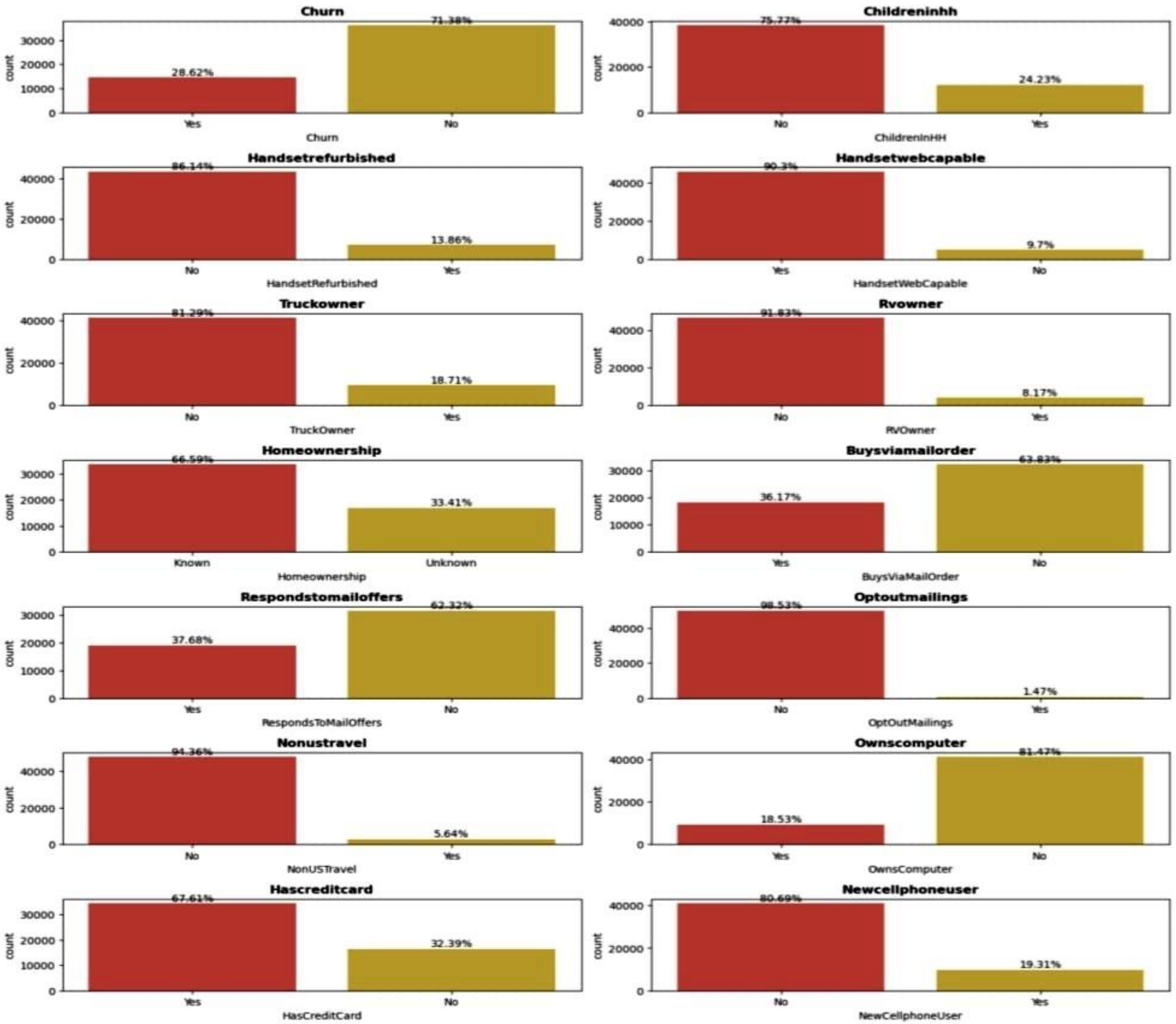
v. Univariate Analysis:

For numerical columns

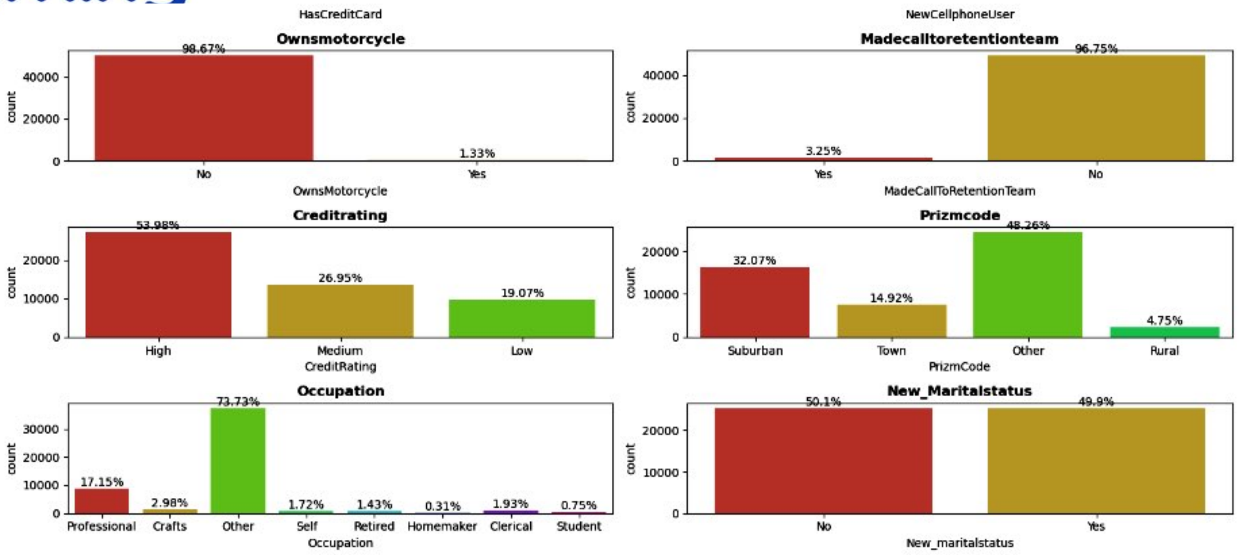
```
# Kde PLot for Numerical features

plt.figure(figsize=(15,25),dpi=100)
n=1
for i in df_num:
    plot=plt.subplot(18,2,n)
    n+=1
    plt.title(i)
    sns.kdeplot(data=df1[i])
    plt.tight_layout()
    annot_percent(plot)
```



For categorical columns




```
#plotting countplot for some categorical variable

plt.figure(figsize=(15,25),dpi=100)
n=1
for i in df_cat:
    plot=plt.subplot(12,2,n)
    n+=1
    sns.countplot(df1[i] ,palette=sns.color_palette("hls", 8))
    plt.title(f'{i.title()}',weight='bold')
    plt.tight_layout()
    annot_percent(plot)
```



Observations:

1) Churn Over 28 percent of people in the data have churned.

2) Handsetwebcapable More than 90 percent of the people in the data have internet support on their phone.

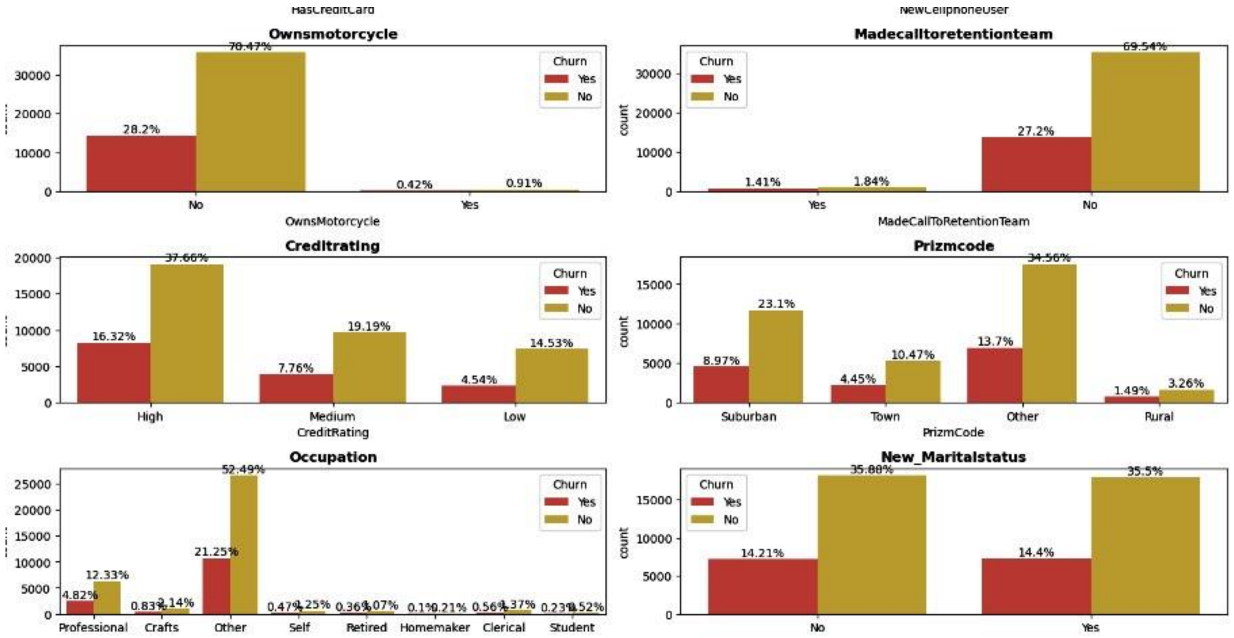
3) More than 65 percent of them don't have a credit card

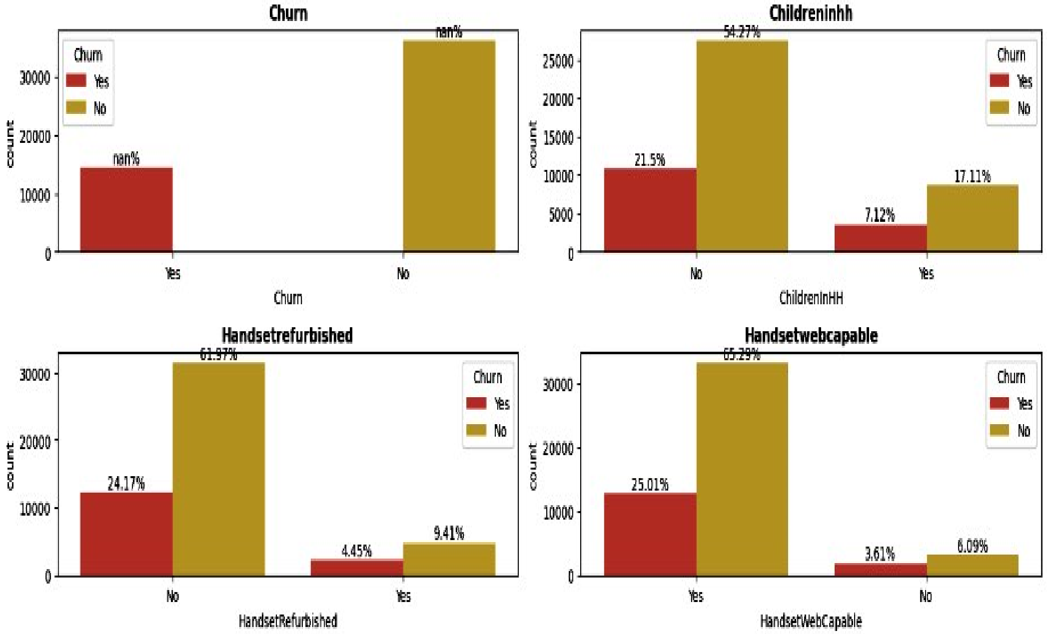
4) Less than 2 percent of them own a motorcycle

5) Over 70 percent of the data has occupations other than the ones mentioned.

Great Learning

vi. Bivariate Analysis





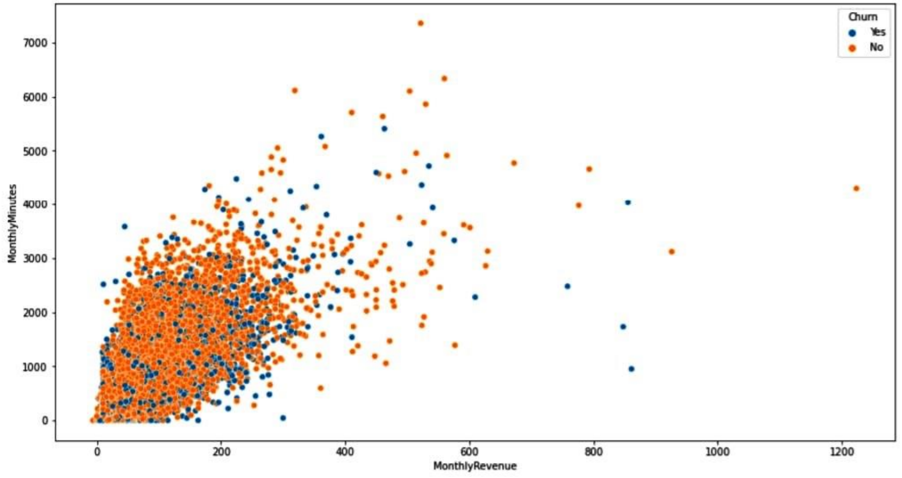
observation:

1. In Handset web capability over 25% of people who have churned has more than 90% of Internet capability on their phone.
2. Less than 6% of people who own new phone have churned.
3. Data shows that people who have Credit Cards are more likely to Churn
4. Marital Status of people churning is independent
5. People who have responded mail offer are less likely to churn

vii. Multivariate analysis

```
sns.scatterplot(x="MonthlyRevenue", y="MonthlyMinutes",hue='Churn', data=df1)
```

```
<AxesSubplot:xlabel='MonthlyRevenue', ylabel='MonthlyMinutes'>
```



Observation:

According to plot, as Monthly Revenue Increases, then number of Monthly Minutes increases, but we could not draw any conclusion on churn.

viii. Statistical Tests

Feature	Statistical Test	P-Value	Inference
MonthlyRevenue	kruskal wallis test	0.000000	Dependent numerical variable found after H-tes...
MonthlyMinutes	kruskal wallis test	0.000000	Dependent numerical variable found after H-tes...
TotalRecurringCharge	kruskal wallis test	0.000000	Dependent numerical variable found after H-tes...
DirectorAssistedCalls	kruskal wallis test	0.000000	Dependent numerical variable found after H-tes...
OverageMinutes	kruskal wallis test	0.000009	Dependent numerical variable found after H-tes...
RoamingCalls	kruskal wallis test	0.922785	Independent numerical variable found after H-t...
PercChangeMinutes	kruskal wallis test	0.000000	Dependent numerical variable found after H-tes...
PercChangeRevenues	kruskal wallis test	0.308102	Independent numerical variable found after H-t...
DroppedCalls	kruskal wallis test	0.000000	Dependent numerical variable found after H-tes...
BlockedCalls	kruskal wallis test	0.000650	Dependent numerical variable found after H-tes...
UnansweredCalls	kruskal wallis test	0.000000	Dependent numerical variable found after H-tes...
CustomerCareCalls	kruskal wallis test	0.000000	Dependent numerical variable found after H-tes...
ThreewayCalls	kruskal wallis test	0.000000	Dependent numerical variable found after H-tes...
ReceivedCalls	kruskal wallis test	0.000000	Dependent numerical variable found after H-tes...
OutboundCalls	kruskal wallis test	0.000000	Dependent numerical variable found after H-tes...
InboundCalls	kruskal wallis test	0.000000	Dependent numerical variable found after H-tes...
PeakCallsInOut	kruskal wallis test	0.000000	Dependent numerical variable found after H-tes...
OffPeakCallsInOut	kruskal wallis test	0.000000	Dependent numerical variable found after H-tes...
DroppedBlockedCalls	kruskal wallis test	0.000000	Dependent numerical variable found after H-tes...
CallForwardingCalls	kruskal wallis test	0.311887	Independent numerical variable found after H-t...
CallWaitingCalls	kruskal wallis test	0.000000	Dependent numerical variable found after H-tes...

RetentionCalls	kruskal wallis test	0.000000	Dependent numerical variable found after H-tes...
RetentionOffersAccepted	kruskal wallis test	0.000000	Dependent numerical variable found after H-tes...
ReferralsMadeBySubscriber	kruskal wallis test	0.024863	Dependent numerical variable found after H-tes...
IncomeGroup	kruskal wallis test	0.026027	Dependent numerical variable found after H-tes...
AdjustmentsToCreditRating	kruskal wallis test	0.000646	Dependent numerical variable found after H-tes...
HandsetPrice	kruskal wallis test	0.242433	Independent numerical variable found after H-t...
ChildrenInHH	Chi-Square Test for Independence	0.030195	Dependent categorical variable found after Chi...
HandsetRefurbished	Chi-Square Test for Independence	0.000000	Dependent categorical variable found after Chi...
HandsetWebCapable	Chi-Square Test for Independence	0.000000	Dependent categorical variable found after Chi...
TruckOwner	Chi-Square Test for Independence	0.324832	Independent categorical variable found after C...
RVOwner	Chi-Square Test for Independence	0.500851	Independent categorical variable found after C...
Homeownership	Chi-Square Test for Independence	0.004931	Dependent categorical variable found after Chi...

	Feature	Statistical Test	P-Value	Inference
41	BuysViaMailOrder	Chi-Square Test for Independence	0.000002	Dependent categorical variable found after Chi...
42	RespondsToMailOffers	Chi-Square Test for Independence	0.000000	Dependent categorical variable found after Chi...
43	OptOutMailings	Chi-Square Test for Independence	0.837419	Independent categorical variable found after C...
44	NonUSTravel	Chi-Square Test for Independence	0.562279	Independent categorical variable found after C...
45	OwnsComputer	Chi-Square Test for Independence	0.810924	Independent categorical variable found after C...
46	HasCreditCard	Chi-Square Test for Independence	0.071275	Independent categorical variable found after C...
47	NewCellphoneUser	Chi-Square Test for Independence	0.141394	Independent categorical variable found after C...
48	NotNewCellphoneUser	Chi-Square Test for Independence	0.106749	Independent categorical variable found after C...
49	OwnsMotorcycle	Chi-Square Test for Independence	0.089071	Independent categorical variable found after C...
50	MadeCallToRetentionTeam	Chi-Square Test for Independence	0.000000	Dependent categorical variable found after Chi...
51	CreditRating	Chi-Square Test for Independence	0.000000	Dependent categorical variable found after Chi...
52	PrizmCode	Chi-Square Test for Independence	0.000295	Dependent categorical variable found after Chi...
53	Occupation	Chi-Square Test for Independence	0.253384	Independent categorical variable found after C...
54	MaritalStatus	Chi-Square Test for Independence	0.000000	Dependent categorical variable found after Chi...

for categorical columns:

we used ***Chi-Square Test for Independence*** to check whether the categorical variables are independent or not.

$H0$: The variables are independent.

$H1$: The variables are not independent (i.e., variables are dependent).

To check normality:

We have used ***Jarque-bera*** test to check the normality of data.

$H0$: The data is normally distributed.

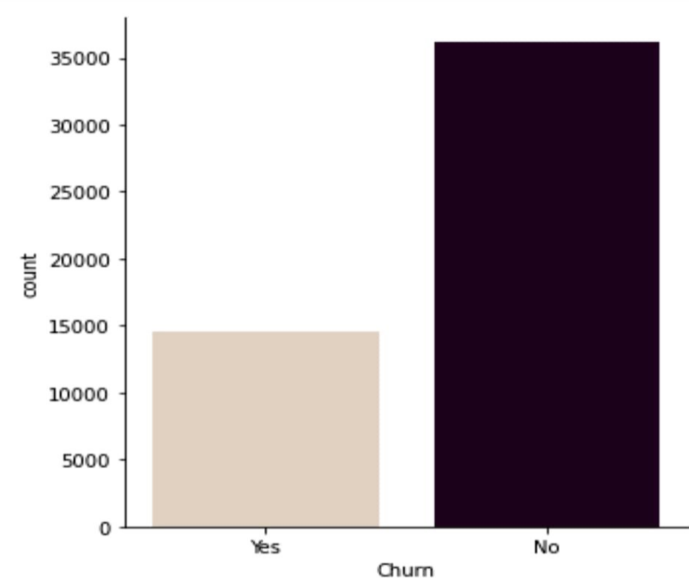
$H1$: The data is not normally distributed.

*We found that data is not normal therefore we use **Kruskal Wallis** test to check its dependency on the target variable*

```
insignificant columns=
      ['RoamingCalls','PercChangeRevenues','CallForwardingCalls','TruckOwner','RVOwner',
       'OptOutMailings','NonUSTravel','OwnsComputer','HasCreditCard','NewCellphoneUser',
       'OwnsMotorcycle','Occupation','New_maritalstatus']
```

- These are the 13 insignificant variables we found.

ix. Class imbalance and treatment



Here we can see that our target variable is too imbalanced, and to treat that we are going to use oversampling techniques like smote.

x. Checking for multicollinearity

	VIF_Factor	Features
0	263.089483	DroppedBlockedCalls
1	133.129373	BlockedCalls
2	91.210629	DroppedCalls
3	11.198276	MonthlyRevenue
4	6.651223	OverageMinutes
5	6.217595	MonthlyMinutes
6	5.514426	HandsetModels
7	5.181104	OffPeakCallsInOut
8	4.951826	Handsets
9	4.506253	PeakCallsInOut
10	4.261220	ReceivedCalls
11	4.122065	TotalRecurringCharge
12	3.557968	OutboundCalls
13	2.661700	MonthsInService
14	2.622790	UnansweredCalls
15	2.614363	ActiveSubs
16	2.572323	InboundCalls
17	2.531890	UniqueSubs
18	2.492452	CallWaitingCalls
19	2.481722	CurrentEquipmentDays
20	2.282738	RetentionCalls
21	2.272071	RetentionOffersAccepted
22	1.632257	PercChangeMinutes
23	1.620044	PercChangeRevenues
24	1.601234	RoamingCalls
25	1.344201	CustomerCareCalls
26	1.343533	DirectorAssistedCalls
27	1.188184	ThreewayCalls
28	1.075708	IncomeGroup
29	1.071277	AdjustmentsToCreditRating
30	1.045947	Age
31	1.013083	ReferralsMadeBySubscriber
32	1.001862	CallForwardingCalls

The variable named dropped blocked calls have high multicollinearity.

xi. Transformation

```python
from sklearn.preprocessing import PowerTransformer
PT=PowerTransformer()
for i in num_f.columns:
    num_f[i]=PT.fit_transform(num_f[[i]])
```

```python
num_f.head()
```

sistedCalls	OverageMinutes	RoamingCalls	PercChangeMinutes	PercChangeRevenues	DroppedCalls	BlockedCalls	UnansweredCalls	CustomerCareCa
-0.044197	-0.995504	-0.621914	-0.566549	-0.450622	-0.889336	-0.306586	-0.568253	-0.824
-0.912074	-0.995504	-0.621914	0.018886	0.088432	-1.200278	-1.216280	-1.013639	-0.824
-0.912074	-0.995504	-0.621914	0.026624	0.088432	-1.515686	-1.216280	-1.760480	-0.824
1.170112	-0.995504	-0.621914	0.655061	0.284291	2.228625	1.290204	1.349529	1.501
-0.912074	-0.995504	-0.621914	0.034384	0.083251	-1.515686	-1.216280	-1.760480	-0.824

- *We used Power transformation, as we can see that there is large number of outliers present so we use Yeo-Johnson transformation technique to reduce the outliers and make the data more normally distributed.*

xii. Base Model(Logistic Regression)

```
# build the model on train data (x_train and y_train)
# use fit() to fit the logistic regression model
logreg = sm.Logit(y_train,x_train).fit()

# print the summary of the model
print(logreg.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:                  Churn   No. Observations:                35475
Model:                          Logit   Df Residuals:                    35415
Method:                           MLE   Df Model:                           59
Date:                Tue, 27 Dec 2022   Pseudo R-squ.:                 0.03218
Time:                        10:23:01   Log-Likelihood:                -20484.
converged:                      False   LL-Null:                       -21165.
Covariance Type:            nonrobust   LLR p-value:                5.454e-246
```

- *The LLR p-value is less than 0.05, implies that the model is significant.*

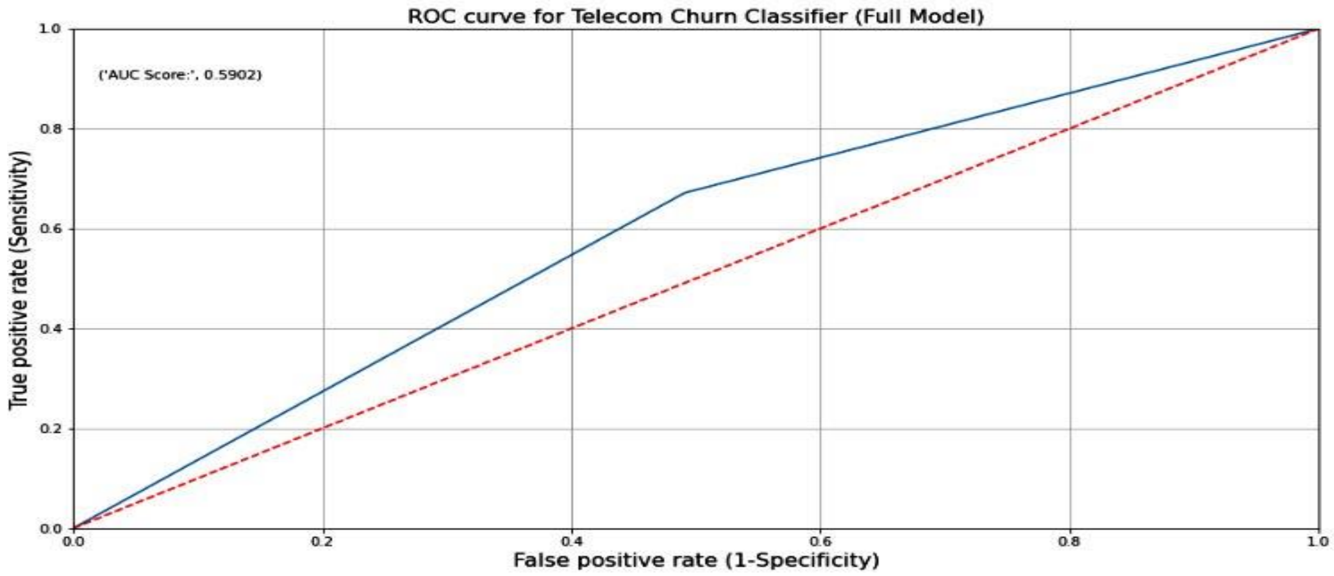
- *The maximum of Cox & Snell R-squared is always less than 1. By above model Cox & Snell R-squared is less than 1 i.e. (0.03456).*

Best Threshold Selection Report

	Probability Cutoff	AUC Score	Precision Score	Recall Score	Accuracy Score	Kappa Score	f1-score
0	0.100000	0.501590	0.292096	0.997517	0.294725	0.001859	0.451873
1	0.100000	0.501590	0.292096	0.997517	0.294725	0.001859	0.451873
2	0.100000	0.501590	0.292096	0.997517	0.294725	0.001859	0.451873
3	0.150000	0.517890	0.299186	0.978786	0.325638	0.021443	0.458287
4	0.150000	0.517890	0.299186	0.978786	0.325638	0.021443	0.458287
5	0.150000	0.517890	0.299186	0.978786	0.325638	0.021443	0.458287
6	0.200000	0.547673	0.314922	0.905890	0.398250	0.061420	0.467369
7	0.200000	0.547673	0.314922	0.905890	0.398250	0.061420	0.467369
8	0.200000	0.547673	0.314922	0.905890	0.398250	0.061420	0.467369
9	0.250000	0.582277	0.344974	0.751298	0.511773	0.122190	0.472836
10	0.260000	0.584886	0.350932	0.709546	0.532886	0.130514	0.469604
11	0.270000	0.590162	0.359961	0.671180	0.556367	0.143743	0.468605
12	0.300000	0.589415	0.382568	0.531934	0.613391	0.160396	0.445053
13	0.300000	0.589415	0.382568	0.531934	0.613391	0.160396	0.445053
14	0.300000	0.589415	0.382568	0.531934	0.613391	0.160396	0.445053
15	0.350000	0.573750	0.430589	0.323403	0.678177	0.159163	0.369377

Threshold 0.27 is giving highest roc-auc score 0.59.

Roc-Curve:



- *The red dotted line represents the ROC curve of a pure random classifier; a good classifier stays as far away from that line as possible (towards top-left corner).*
- *From the above plot, we can see that our classifier (logistic regression) is away from the dotted line; with the AUC score 0.5902.*

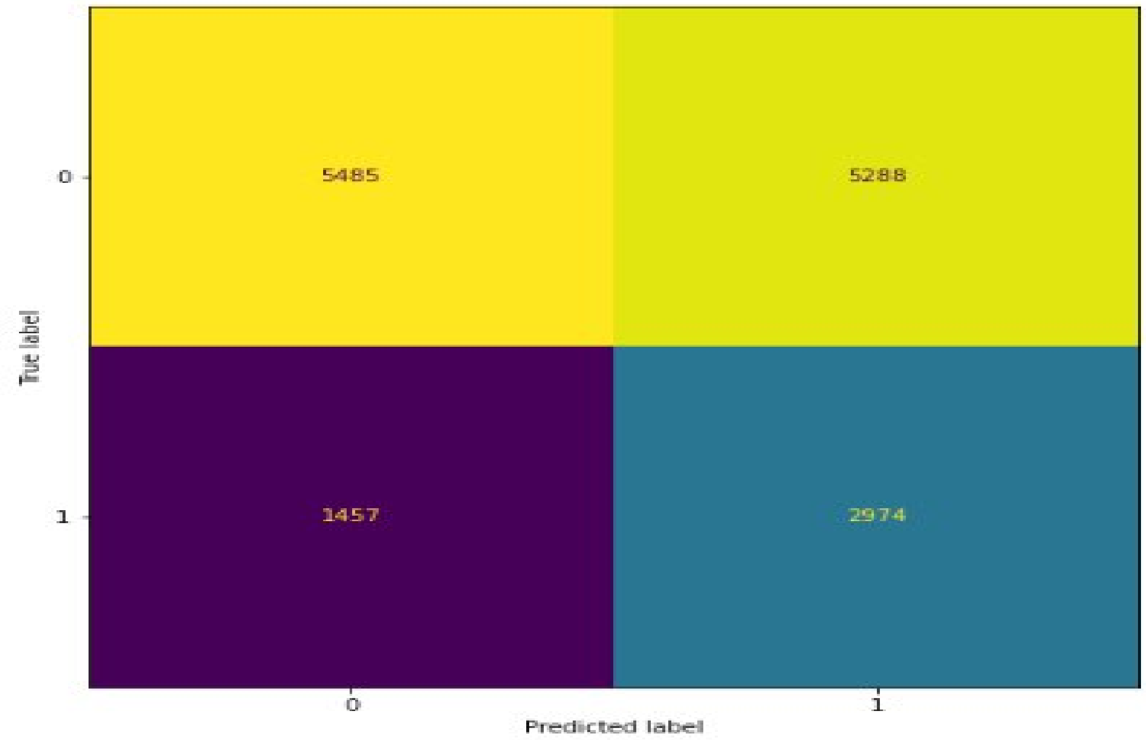
Report for 0.5 cutoff and best cutoff (0.27)according to Auc-score:-

```
print(classification_report(y_test,y_pred))

              precision    recall  f1-score   support

           0       0.71      0.99      0.83     10773
           1       0.54      0.03      0.06      4431

    accuracy                           0.71     15204
   macro avg       0.63      0.51      0.44     15204
weighted avg       0.66      0.71      0.60     15204
```

```
print(classification_report(y_test,y_pred))

              precision    recall  f1-score   support

           0       0.79      0.51      0.62     10773
           1       0.36      0.67      0.47      4431

    accuracy                           0.56     15204
   macro avg       0.58      0.59      0.54     15204
weighted avg       0.66      0.56      0.58     15204
```

- *From the above output, we can infer that the recall of the positive class is known as sensitivity and the recall of the negative class is specificity.*
- *support is the number of observations in the corresponding class.*
- *The macro average in the output is obtained by averaging the unweighted mean per label and the weighted average is given by averaging the support-weighted mean per label.*

Confusion Matrix:-



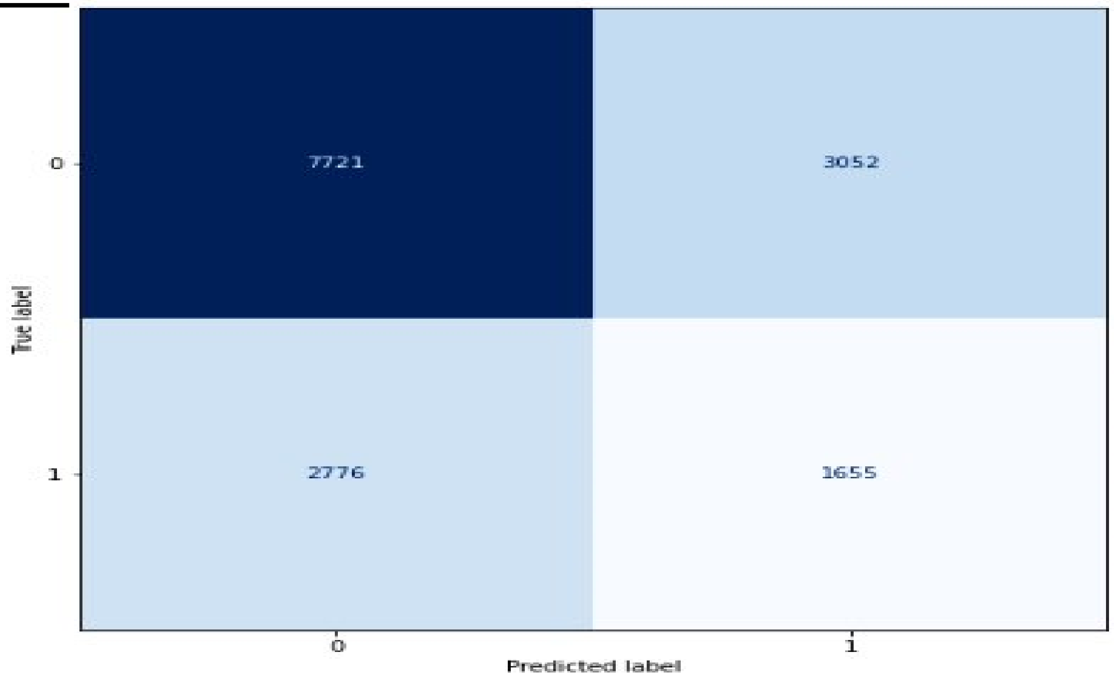
- *By the logistic regression model the maximum roc_auc score obtained by 0.27 cutoff .*
- *The accuracy for the model for the 0.27 Threshold is 0.56.*

xiii. Decision Tree

Build a full decision tree model on a train dataset using 'gini'.

```
# instantiate the 'DecisionTreeClassifier' object using 'gini' criterion
# pass the 'random_state' to obtain the same samples for each time you run the code
decision_tree_classification = DecisionTreeClassifier(criterion = 'gini', random_state = 10)

# fit the model using fit() on train data
decision_tree = decision_tree_classification.fit(x_train, y_train)
```

Model Performance: -



Calculate performance measures on the train set.

```
# compute the performance measures on train data
# call the function 'get_train_report'
# pass the decision tree to the function
train_report = get_train_report(decision_tree)

# print the performance measures
print(train_report)
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	25448
1	1.00	1.00	1.00	10284
accuracy			1.00	35732
macro avg	1.00	1.00	1.00	35732
weighted avg	1.00	1.00	1.00	35732

Calculate performance measures on the test set.

```
# compute the performance measures on test data
# call the function 'get_test_report'
# pass the decision tree to the function
test_report = get_test_report(decision_tree)

# print the performance measures
print(test_report)
```

	precision	recall	f1-score	support
0	0.74	0.72	0.73	10888
1	0.35	0.36	0.35	4427
accuracy			0.62	15315
macro avg	0.54	0.54	0.54	15315
weighted avg	0.62	0.62	0.62	15315

- From The above model, our train accuracy is 1 and test accuracy is 0.62, result is Overfitting
- As the model is over fitted, our false Negative and false Positive is inaccurate
- In the next step we tuned the Hyper parameters and rebuild the model.

Tune the Hyper parameters using GridSearchCV (Decision Tree)

```
# create a dictionary with hyperparameters and its values
# pass the criteria 'entropy' and 'gini' to the parameter, 'criterion'
tuned_paramaters = [{'criterion': ['entropy', 'gini'],
                     'max_depth': range(10, 20),
                     'max_features': ["sqrt", "log2"],
                    }]
```

```
kf=KFold(n_splits=5,shuffle=True, random_state=0)
```

```
DT=DecisionTreeClassifier(random_state=0)
```

```
gr_model=GridSearchCV(estimator=DT,
    param_grid=tuned_paramaters,cv=kf)
```

```
tree_grid_model=gr_model.fit(x_train,y_train)
print('Best parameters for decision tree classifier: ', tree_grid_model.best_params_, '\n')
```

```
Best parameters for decision tree classifier:  {'criterion': 'entropy', 'max_depth': 10, 'max_features': 'sqrt'}
```

Model Performance after Tuning:

performance measures on train model

```
y_pred=dt_model.predict(x_train)
```

```
print(classification_report(y_train,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.74      0.97      0.84     25403
           1       0.67      0.14      0.23     10072

    accuracy                           0.74     35475
   macro avg       0.71      0.56      0.53     35475
weighted avg       0.72      0.74      0.67     35475
```

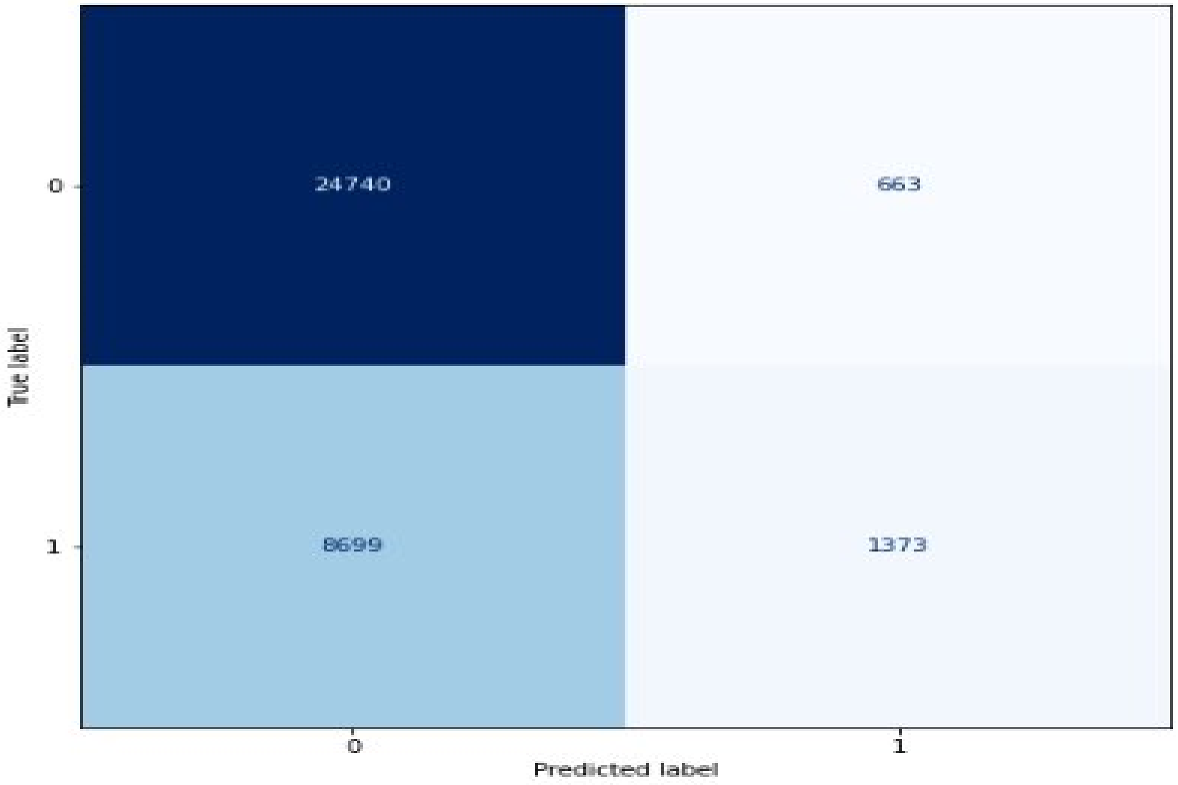
performance measures on test model

```
y_test_pred=dt_model.predict(x_test)
```

```
print(classification_report(y_test,y_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.72      0.96      0.82     10773
           1       0.47      0.10      0.16      4431

    accuracy                           0.70     15204
   macro avg       0.59      0.53      0.49     15204
weighted avg       0.65      0.70      0.63     15204
```

Confusion Matrix:



- The train and test Accuracy are comparable, which shows the reduction in overfitting.
- In this case the false negative and false positive values can be trusted and the FN value are quite High, but as our focus is on reduction of False negative values

xiv. Random forest for classification

```
from sklearn.ensemble import RandomForestClassifier
rnd=RandomForestClassifier(random_state=0)
random_model=rnd.fit(x_train,y_train)
```

results for the train data.

```
y_pred=random_model.predict(x_train)
```

```
print(classification_report(y_train,y_pred))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	25403
1	1.00	1.00	1.00	10072
accuracy			1.00	35475
macro avg	1.00	1.00	1.00	35475
weighted avg	1.00	1.00	1.00	35475

results for the test data

```
y_test_pred=random_model.predict(x_test)
```

```
print(classification_report(y_test,y_test_pred))
```

	precision	recall	f1-score	support
0	0.72	0.98	0.83	10773
1	0.56	0.07	0.13	4431
accuracy			0.71	15204
macro avg	0.64	0.53	0.48	15204
weighted avg	0.67	0.71	0.63	15204

- From The above model, our train accuracy is 0.1 and test accuracy is 0.71, result is Overfitting
- As the model is over fitted, our false Negative and false Positive is inaccurate
- In the next step we tuned the Hyper parameters and rebuild the model.

Tuned the Hyper parameters using GridSearchCV (Random Forest)

Hyper parameters tuning by Gridsearch cv

```
]: grid={'n_estimators':range(10,100,10),'criterion':['gini','entropy'],'max_depth':range(2,25)}
```

```
]: from sklearn.model_selection import GridSearchCV,KFold
kf=KFold(n_splits=5,shuffle=True, random_state=0)
```

```
]: grid_model=GridSearchCV( estimator=rnd,
    param_grid=grid,
    cv=kf)
```

```
]: forest_model=grid_model.fit(x_train,y_train)
print('Best parameters for random forest classifier: ', forest_model.best_params_, '\n')
```

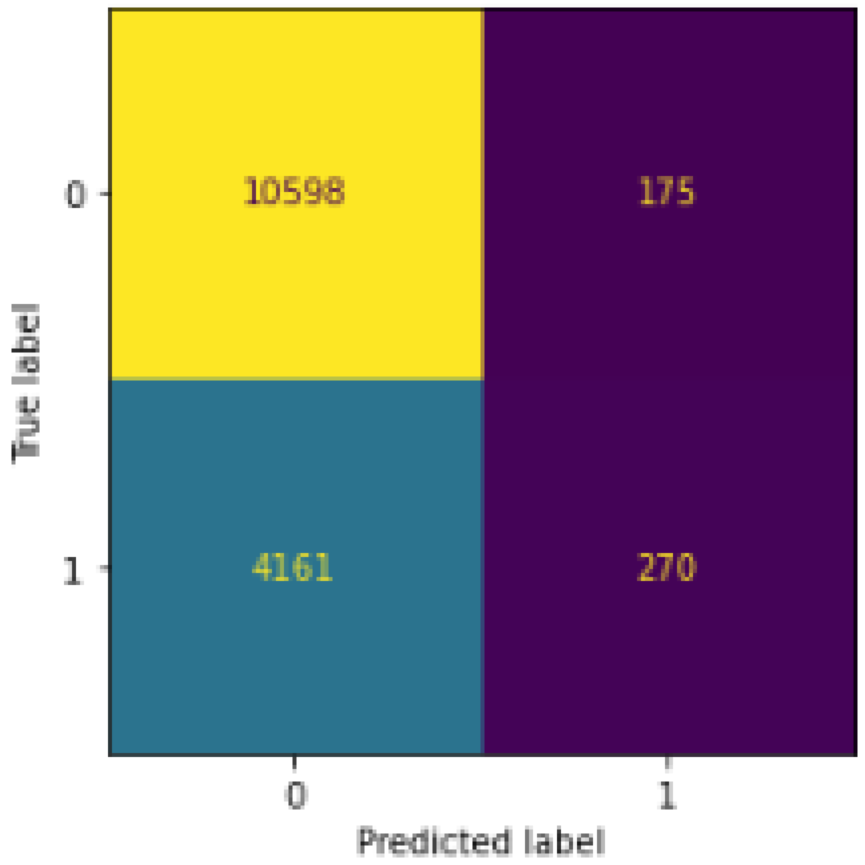
```
Best parameters for random forest classifier:  {'criterion': 'entropy', 'max_depth': 20, 'n_estimators': 70}
```

Model performance after tuning:

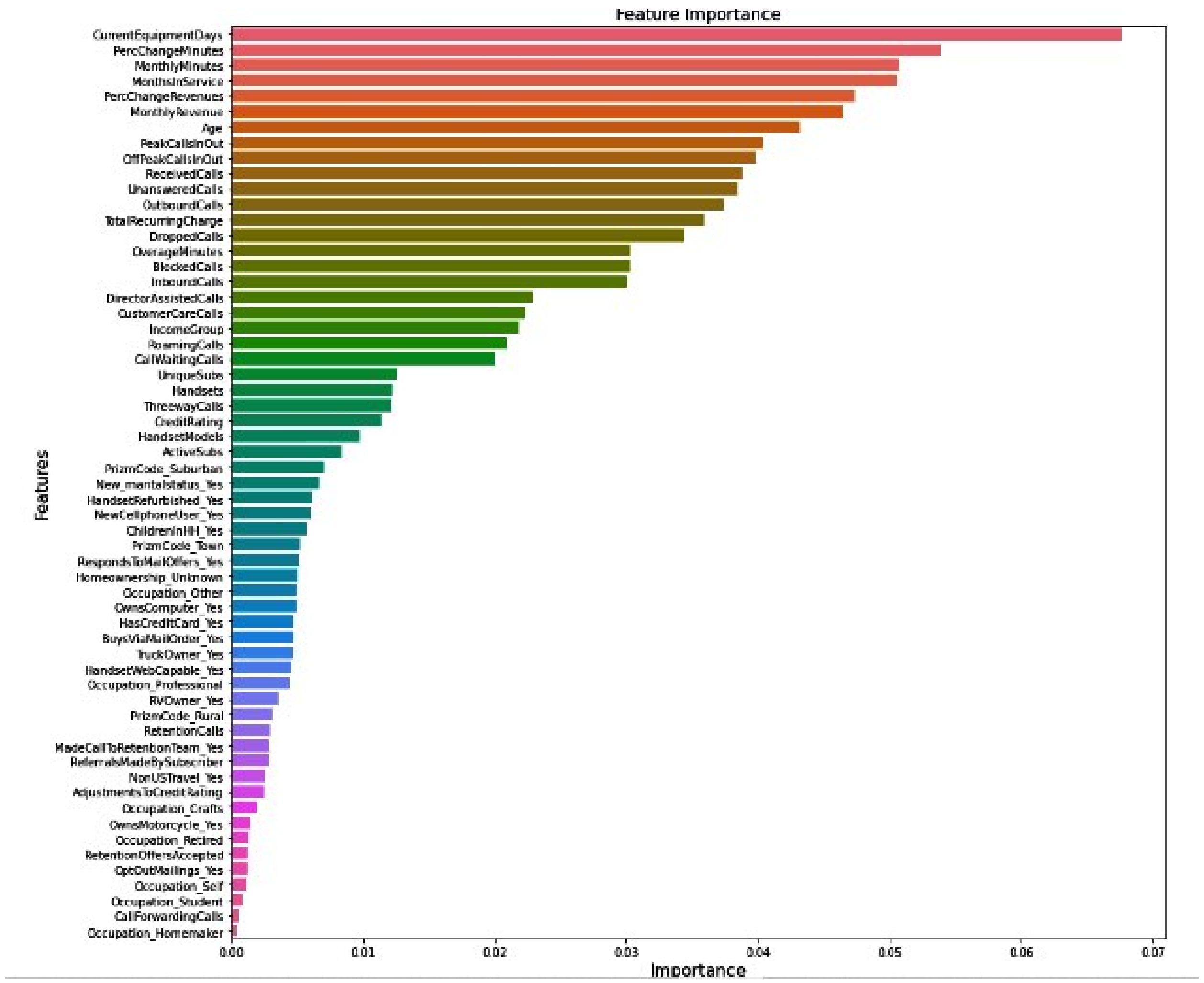
```
print(classification_report(ytest,test_predict))
```

```
              precision    recall  f1-score   support

           0       0.72      0.95      0.82     10773
           1       0.48      0.12      0.19      4431

    accuracy                           0.71     15204
   macro avg       0.60      0.53      0.51     15204
weighted avg       0.65      0.71      0.64     15204
```

Confusion matrix:



Feature importance:



The train and test Accuracy are comparable, which shows the reduction in overfitting.

- In this case the false negative and false positive values can be trusted and the FN value are quite High, but as our focus is on reduction of False negative values.

- Typically, Random Forest classifier is more accurate than a single decision tree, we rebuild the model using the same to reduce the FN and increase the accuracy.

xv. KNN-CLASSIFIER

After Parameter Tuning:-

```
Best parameters for KNN Classifier:  {'metric': 'manhattan', 'n_neighbors': 23}

CPU times: total: 59min 21s
Wall time: 1h 8min 9s
```

```
knn_class = KNeighborsClassifier(n_neighbors = 23,metric= 'manhattan')
knn_model_1 = knn_class.fit(xtrain, ytrain)
```

Report for test:-

```
# test report
```

```
print(classification_report(ytest,test_predict))
```

	precision	recall	f1-score	support
0	0.71	0.97	0.82	10773
1	0.45	0.05	0.10	4431
accuracy			0.71	15204
macro avg	0.58	0.51	0.46	15204
weighted avg	0.64	0.71	0.61	15204

- The accuracy is 71 % which is increased compared to previous models.
- But the Recall score for Churners is reduced to 0.05.
- We try with boosting Techniques to increase the recall score.

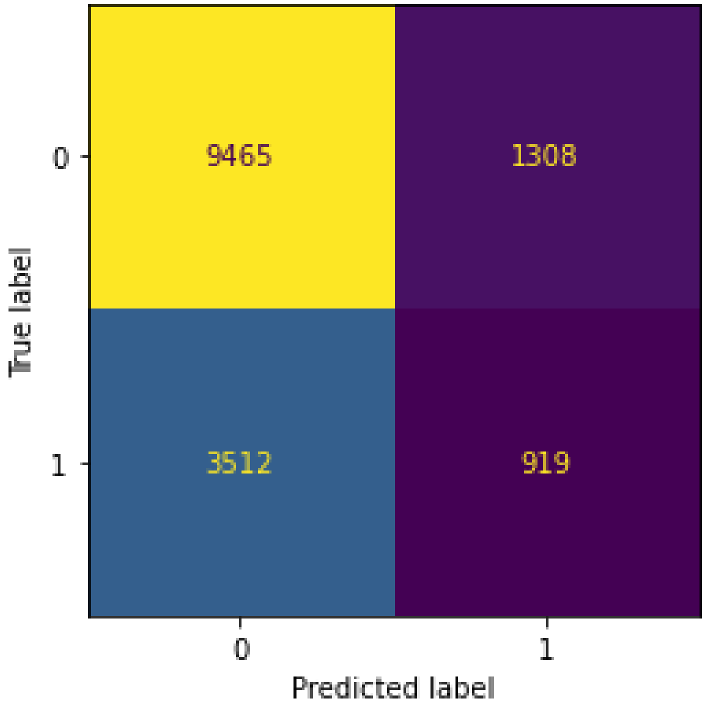
xvi. Naive Bayes –Classifier

Train and test report:-

```
#train report

print(classification_report(ytrain,train_predict))

              precision    recall  f1-score   support

           0       0.73      0.88      0.80     25403
           1       0.39      0.19      0.26     10072

    accuracy                           0.68     35475
   macro avg       0.56      0.54      0.53     35475
weighted avg       0.64      0.68      0.65     35475
```

```
#test report

print(classification_report(ytest,test_predict))

              precision    recall  f1-score   support

           0       0.73      0.88      0.80     10773
           1       0.41      0.21      0.28      4431

    accuracy                           0.68     15204
   macro avg       0.57      0.54      0.54     15204
weighted avg       0.64      0.68      0.65     15204
```

Confusion Matrix:-



- Compare to previous models the Naïve Bayes is giving good recall score for churners.
- But there is slight decrease in the accuracy of the model.
- Boosting models would give good results.

xvii. Boosting Models

1. AdaBoost classifier:-

Test report:-

```
print(classification_report(ytest,test_predict))
```

	precision	recall	f1-score	support
0	0.72	0.97	0.83	10773
1	0.57	0.09	0.16	4431
accuracy			0.71	15204
macro avg	0.64	0.53	0.49	15204
weighted avg	0.68	0.71	0.63	15204

2. GradientBoost Classifier:-

Test report:-

```
# test report
```

```
print(classification_report(ytest,test_predict))
```

```
              precision    recall  f1-score   support

           0       0.73      0.92      0.82     10773
           1       0.49      0.19      0.27      4431

    accuracy                           0.71     15204
   macro avg       0.61      0.55      0.54     15204
weighted avg       0.66      0.71      0.66     15204
```

3. XGBoost Classifier:-

Test report:-

```
print(classification_report(ytest,test_pred))
```

	precision	recall	f1-score	support
0	0.74	0.94	0.83	10773
1	0.55	0.18	0.27	4431
accuracy			0.72	15204
macro avg	0.64	0.56	0.55	15204
weighted avg	0.68	0.72	0.66	15204

- Compare to all Boosting models XGBoost model gave good Accuracy and Recall score.
- We will use Stacking Technique to increase the Recall score.

xviii. Stacking Technique

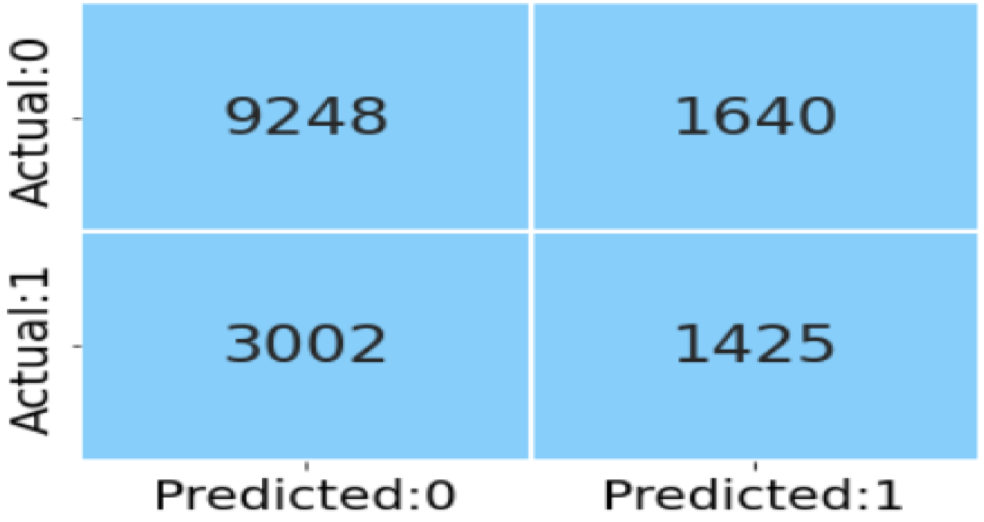
```
%%time
# consider the various algorithms as base learners
base_learners = [('Grad_model',GradientBoostingClassifier(n_estimators = 150, max_depth = 10, random_state = 10)),
                 ('xgb_model',XGBClassifier(colsample_bytree= 1, gamma= 1, learning_rate= 0.4,
                              max_depth=4, min_child_weight= 4, subsample= 1, tree_method= 'hist' )),
                 ('NB_model', GaussianNB())]

# initialize stacking classifier
# pass the base learners to the parameter, 'estimators'
# pass the Naive Bayes model as the 'final_estimator'/ meta model
stack_model = StackingClassifier(estimators = base_learners, final_estimator =GaussianNB())
```

Test Report:-

```
print(classification_report(ytest,test_pred))
```

```
              precision    recall  f1-score   support

           0       0.75      0.86      0.80     10773
           1       0.47      0.31      0.37      4431

    accuracy                           0.70     15204
   macro avg       0.61      0.58      0.59     15204
weighted avg       0.67      0.70      0.68     15204
```

Confusion Matrix:-



- Compare to all models Stacking technique gave good accuracy of 70%.
- Recall score for churners as 0.31
- Auc score as 0.59.
- **Compare to all models this model is best.**

Limitations:-

- The data which we have is highly imbalanced this might lead to inaccurate predictions.
- To enhance the data quality and to reduce errors we have transformed the data using power transformer, getting Business insights out of this would be difficult.
- To proceed with Feature Engineering, we need to have domain knowledge

Conclusion:-

- At first, we dealt with the null value imputation and then we proceeded with Exploratory data analysis to analyse the univariate and bivariant features to understand why the customers are churning.
- As the data was not normal, we use non parametrical statistical test **Kruskal Wallis test**
- This test is used to check features are dependent or independent to Target variables.
- We have built various classification algorithms and final outcomes are as follows
- Compare to base logistic model, the overfitting is reduced and FN errors are reduced by nearly 32%
- Comparatively the recall value has been boosted from 4% to 31%
- Compare to base Decision model, the overfitting is reduced and FN errors are reduced by nearly 30%

Report Card for all models:-

ALGORITHMS	Remark	Train Set							Test Set						
		Recall		Precision		F1 Score		Accuracy	Recall		Precision		F1 Score		Accuracy
		0	1	0	1	0	1		0	1	0	1	0	1	
Logistic Regresion	Threashold as 0.5								0.98	0.04	0.72	0.49	0.83	0.07	0.71
	Threashold as 0.3								0.51	0.67	0.79	0.36	0.79	0.31	0.57
Decision Tree	Overfit	1	1	1	1	1	1	1	0.72	0.36	0.74	0.35	0.73	0.35	0.71
Decision Tree	After Hyper tunning	0.99	0.03	0.72	0.58	0.83	0.06	0.71	0.96	0.1	0.71	0.56	0.83	0.05	0.7
Random Forest	Overfit (n-estimator=70)	1	0.98	0.99	1	1	0.99	0.99	0.92	0.17	0.7	0.45	0.81	0.24	0.7
Random Forest	After Hyper tunning	1	0	0.71	0	0.83	0	0.71	0.95	0.12	0.71	0.75	0.83	0	0.71
KNN	Only with numerical variables							0.71	0.96	0.05	0.79	0.25	0.83	0.17	0.71
Navie Bayes								0.71	0.87	0.22	0.73	0.39	0.79	0.28	0.68
XG Boost	max_depth = 10, gamma = 1							0.99	0.89	0.24	0.74	0.48	0.81	0.32	0.7
Stack Model	XGBoost,Naive Bayes,Gradient Boost							**0.72**	**0.85**	**0.31**	**0.75**	**0.46**	**0.8**	**0.38**	**0.7**

Great Learning

THANK YOU !!