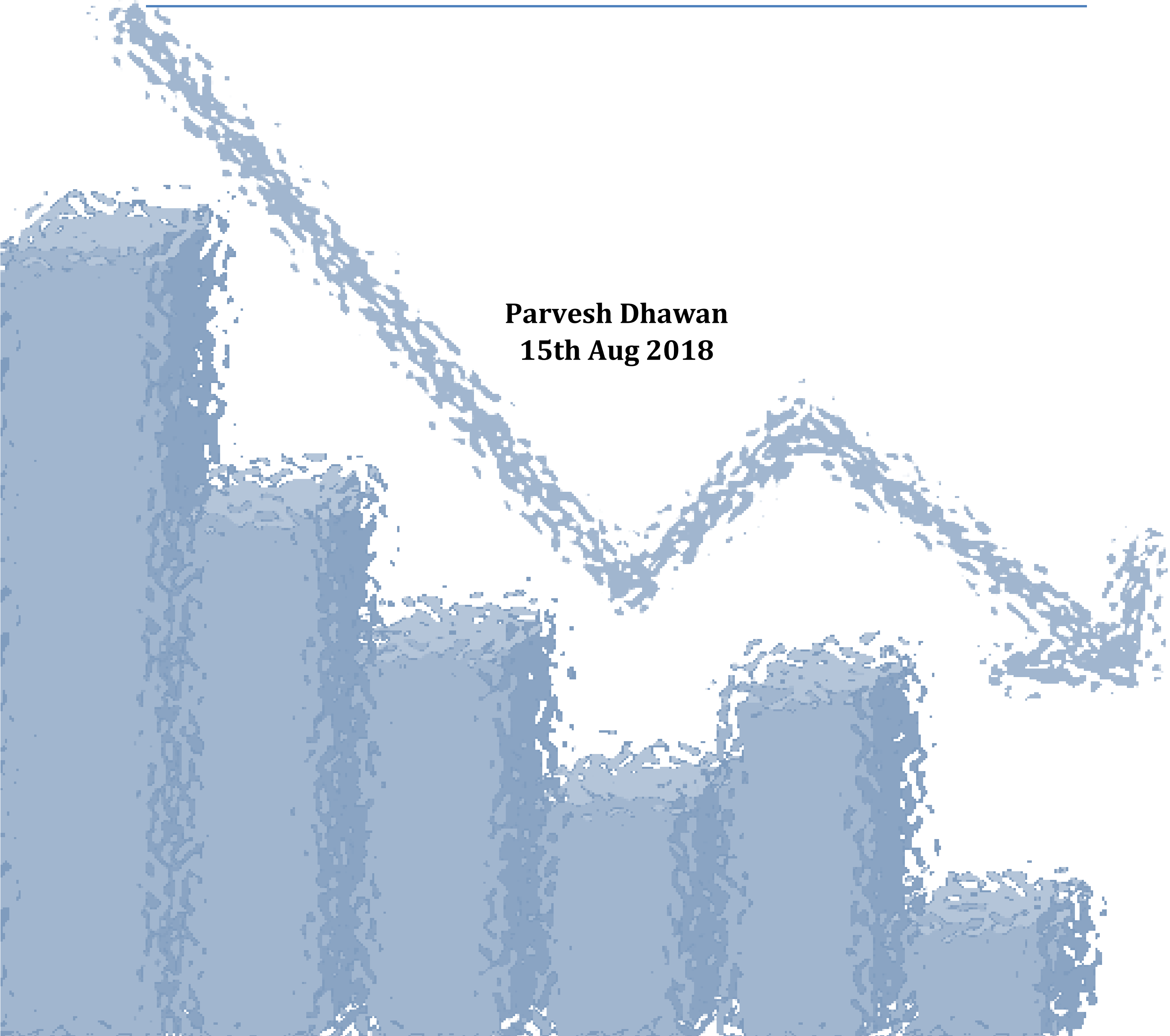
Report
On Project
Customer Churn Reduction



Parvesh Dhawan
15th Aug 2018

Contents

Chapter 1

Introduction

Customer churn refers to when a customer (player, subscriber, user, etc.) ceases his or her relationship with a company. Businesses typically treat a customer as churned once a particular amount of time has elapsed since the customer's last interaction with the site or service. The full cost of customer churn includes both lost revenue and the marketing costs involved with replacing those customers with new ones. Reduction customer churn is important because cost of acquiring a new customer is higher than retaining an existing one. Reducing customer churn is a key business goal of every business. This case is related to telecom industry where particular organizations want to know that for given certain parameters whether a person will churn or not.

1.1 Problem Statement

In this problem statement, we were provided with a train and test dataset of a telecom company. The data set consists of 20 variables describing various services and charges associated with them, duration and any service calls made by customer. The dataset also contains geographic location of customers in form of state and a particular area code. **'Churn'** is the target variable, which tells us weather the customer has churned out or not.

1.2 Data

Two different (files as Test.csv and Train.csv) datasets were provided as train data and test data. Data contains 20 predictor variables and 1 target variables.

Variables		**Description**
State *	:	State to which customer belongs
Account length	:	Service usage period
Area code	:	Telephone area code
Phone number *	:	Customer's phone number
International plan *	:	'yes' if customer opted for international plan else 'no'
Voice mail plan *	:	'yes' if customer opted for voice mail plan else 'no'
Number vmail messages	:	Number of voice messages stored or received by customer
Total day minutes	:	Total minutes in day time usage
Total day calls	:	Total calls made in day time
Total day charges	:	Charges for services used during day time
Total eve minutes	:	Total minutes in evening time usage
Total eve calls	:	Total calls made in evening time
Total eve charges	:	Charges for services used during evening time
Total night minutes	:	Total minutes in night time usage
Total night calls	:	Total calls made in night time
Total night charges	:	Charges for services used during night time

Total intl minutes	:	Total international minutes used
Total intl calls	:	Total international calls made
Total intl charges	:	Charges for international calls
Number customer services call	:	Services call made by customer
Churn *	:	Target - 'True.' If customer churned else 'False'

Size of Dataset Provided :-

Train.csv = 3333 rows, 21 Columns

Test.csv = 1667 rows, 21 Columns

State	account length	area code	phone number	international plan	voice mail plan	number vmail messages
KS	128	415	382-4657	no	yes	25
OH	107	415	371-7191	no	yes	26
NJ	137	415	358-1921	no	no	0
OH	84	408	375-9999	yes	no	0
OK	75	415	330-6626	yes	no	0

total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes
265.1	110	45.07	197.4	99	16.78	244.7
161.6	123	27.47	195.5	103	16.62	254.4
243.4	114	41.38	121.2	110	10.3	162.6
299.4	71	50.9	61.9	88	5.26	196.9
166.7	113	28.34	148.3	122	12.61	186.9

total night calls	total night charge	total intl minutes	total intl calls	total intl charge	number customer service calls	Churn
91	11.01	10	3	2.7	1	False.
103	11.45	13.7	3	3.7	1	False.
104	7.32	12.2	5	3.29	0	False.
89	8.86	6.6	7	1.78	2	False.
121	8.41	10.1	3	2.73	3	False.

***Churn is Our Target Variable**

Chapter 2

Methodology

Customer churn reduction is a business scenario in which a company is trying to retain a customer which is more likely to leave the services. For reducing churn rate, we need to identify which customers are most likely to churn and which are not. Also we have some data to train our model which makes our problem as **Supervised Classification problem**.

- **EDA (Exploratory Data Analysis)**
  It includes looking into the data analyzing various variables, visualization, missing value analysis, correlation analysis, chi-square test, scaling of features, Sampling.

- **Basic Modeling**
  Will try different model over preprocessed data (Random forest, Logistic regression, KNN, Naïve Bayes).

- **Model Evaluation & Optimization**
  Evaluating model performances, select the best model fit for our data, optimizing hyper parameters tuning, Cost effectiveness of model.

- **Implementation model on Final test data**

2.1 Exploratory Data Analysis (EDA)

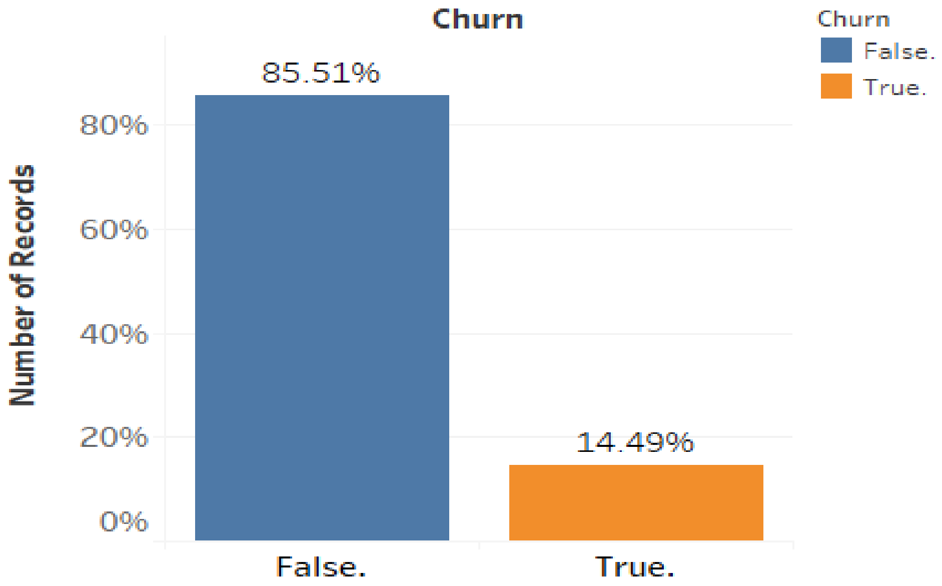
Exploratory Data Analysis is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task.

2.1.1 Target Variable - Churn

Our target variable has two categories which include True and False values.

True = Customer will move or churn out.

False = Customer won’t move



We can clearly see that our data is highly imbalanced. The occurrence of false is higher than True. There are 2850 (85.51%) customers who churn out and 483 (14.49%) customers retained.

2.1.2 Uniqueness in Variable

Let's have a look on amount of uniqueness our variables

Variables		Unique Counts
state	:	51
account length	:	212
area code	:	3
phone number	:	3333
international plan	:	2
voice mail plan	:	2
number vmail messages	:	46
total day minutes	:	1667
total day calls	:	119
total day charge	:	1667
total eve minutes	:	1611
total eve calls	:	123
total eve charge	:	1440
total night minutes	:	1591
total night calls	:	120
total night charge	:	933
total intl minutes	:	162
total intl calls	:	21
total intl charge	:	162
number customer service calls	:	10
Churn	:	2

- **area code** has only 3 values, So will convert it to categorical variable.
- **phone number** has 3333, which makes it a full unique variable. Will remove it because it doesn't contain any important information.

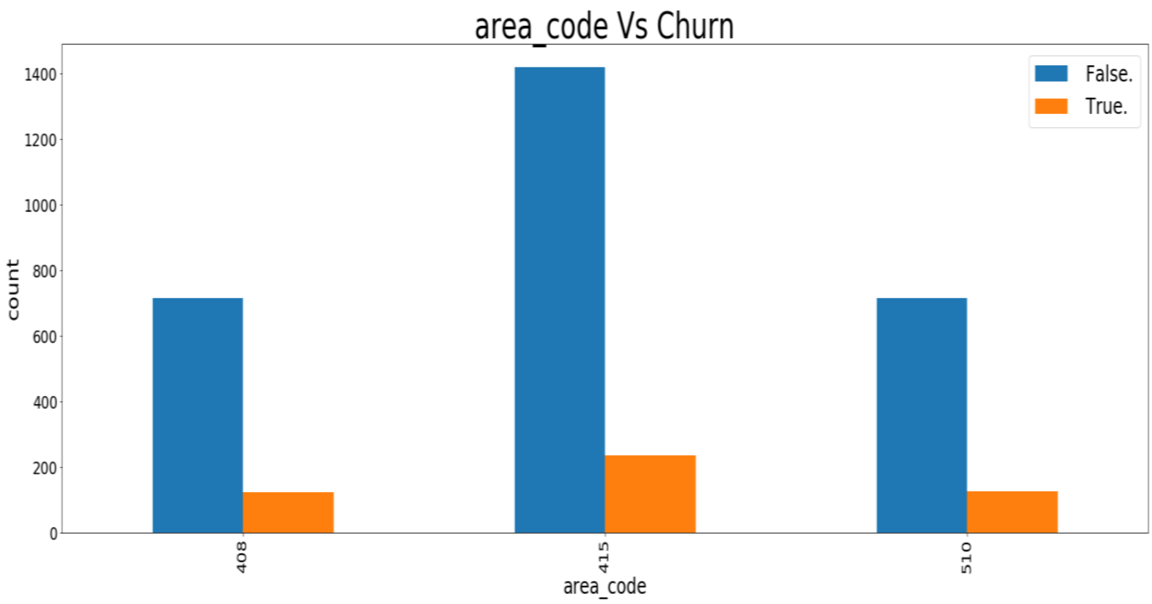
2.1.3 Missing Value Analysis

Variables	Values	Variables	Values
State	0	**total eve calls**	0
account length	0	**total eve charge**	0
area code	0	**total night minutes**	0
international plan	0	**total night calls**	0
voice mail plan	0	**total night charge**	0
number vmail messages	0	**total intl minutes**	0
total day minutes	0	**total intl calls**	0
total day calls	0	**total intl charge**	0
total day charge	0	**number customer service calls**	0
total eve minutes	0	**Churn**	0

- No missing values were present in the training and test dataset.

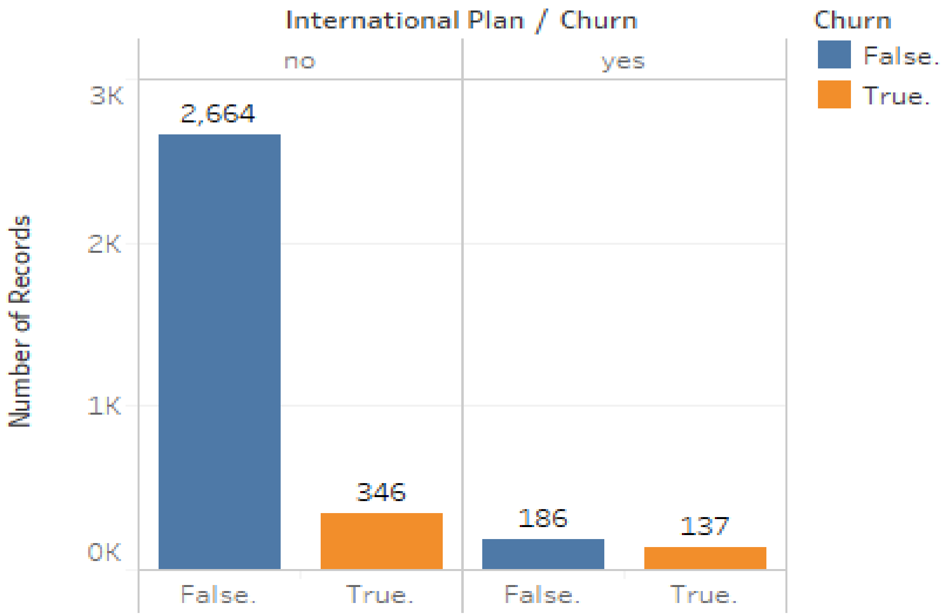
2.1.4 Churning of Customers according to different variables

- ❖ **area code**



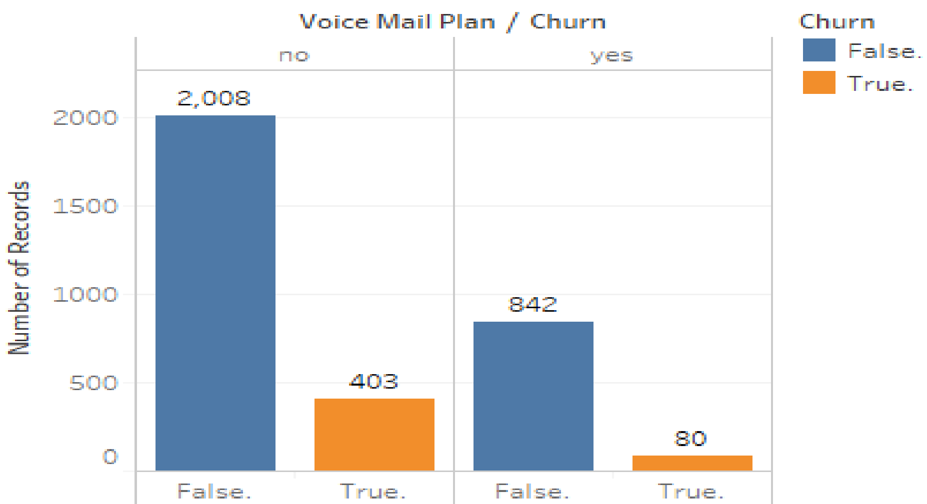
- Most of the churned customers are from 415 area.

❖ **International plan**



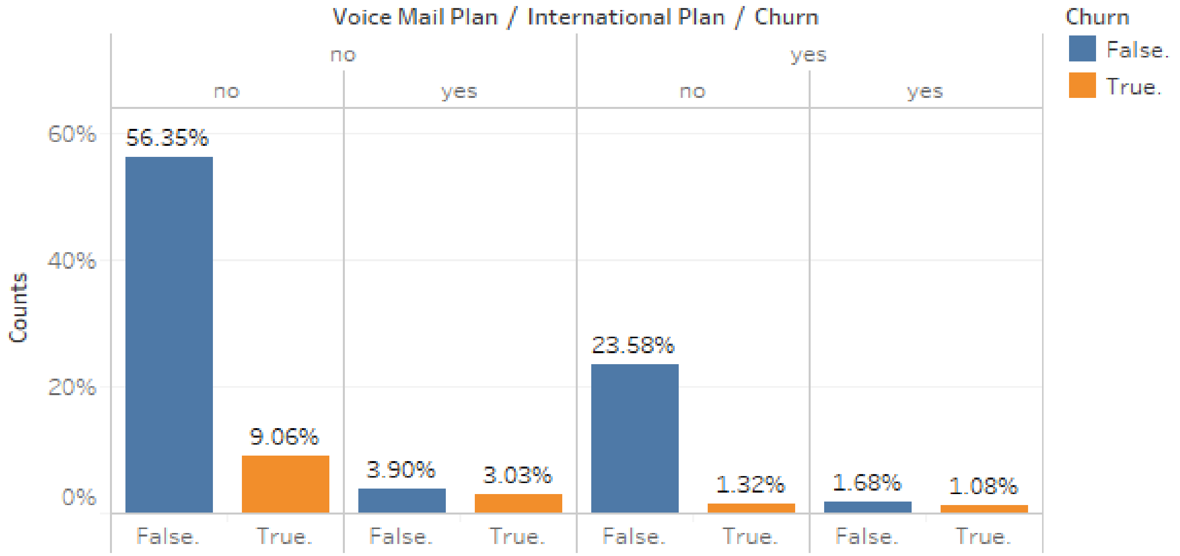
- Churn rate is more with customer using international plan. As only 323 customer using International plan and 137 churning out of them.

❖ **Voice Mail Plan**



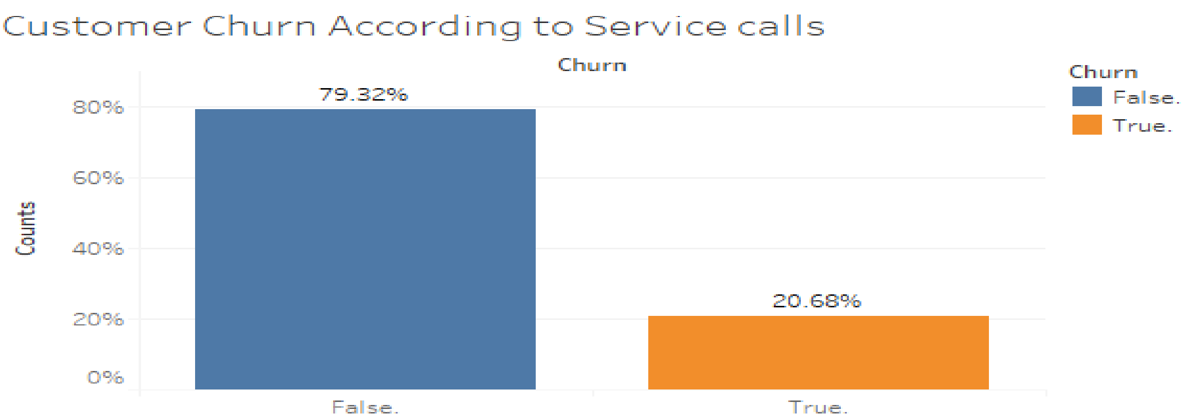
- 922 customer using voice mail plan and 80 out of them are churning.

- Churned Customers who has or not (**voice mail plan and international plan**)

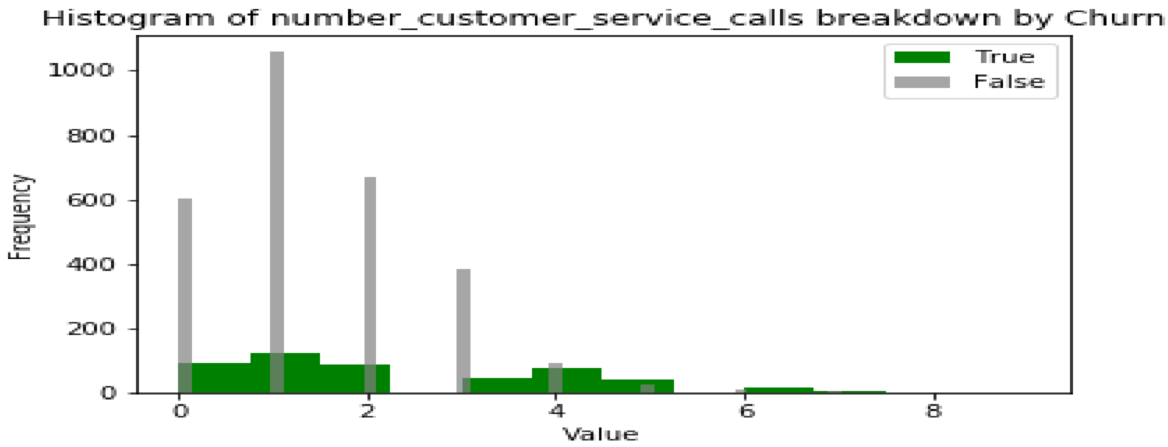


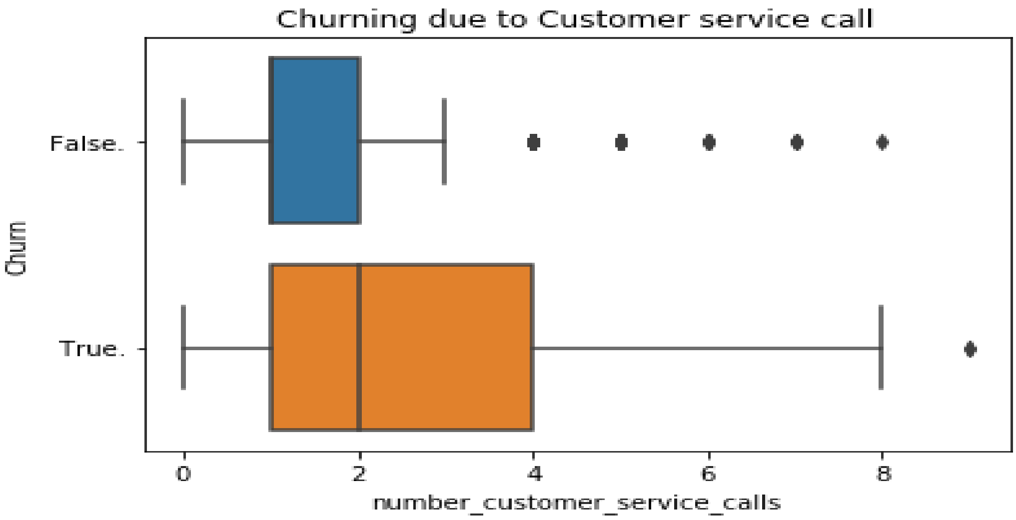
- Churn rate for Customer neither having voice mail plan nor international plan is **9.06%.**
- Churning rate for customer having International plan but don't have voice mail plan is **3.03%** out of **6.93%** customers.
- Churning of customer having both voice mail plan & international plan is **1.08%** out of **2.76%** .

❖ **Customer Service Calls impact to Churn**



- **20.68% customer Churn due to customer service calls**

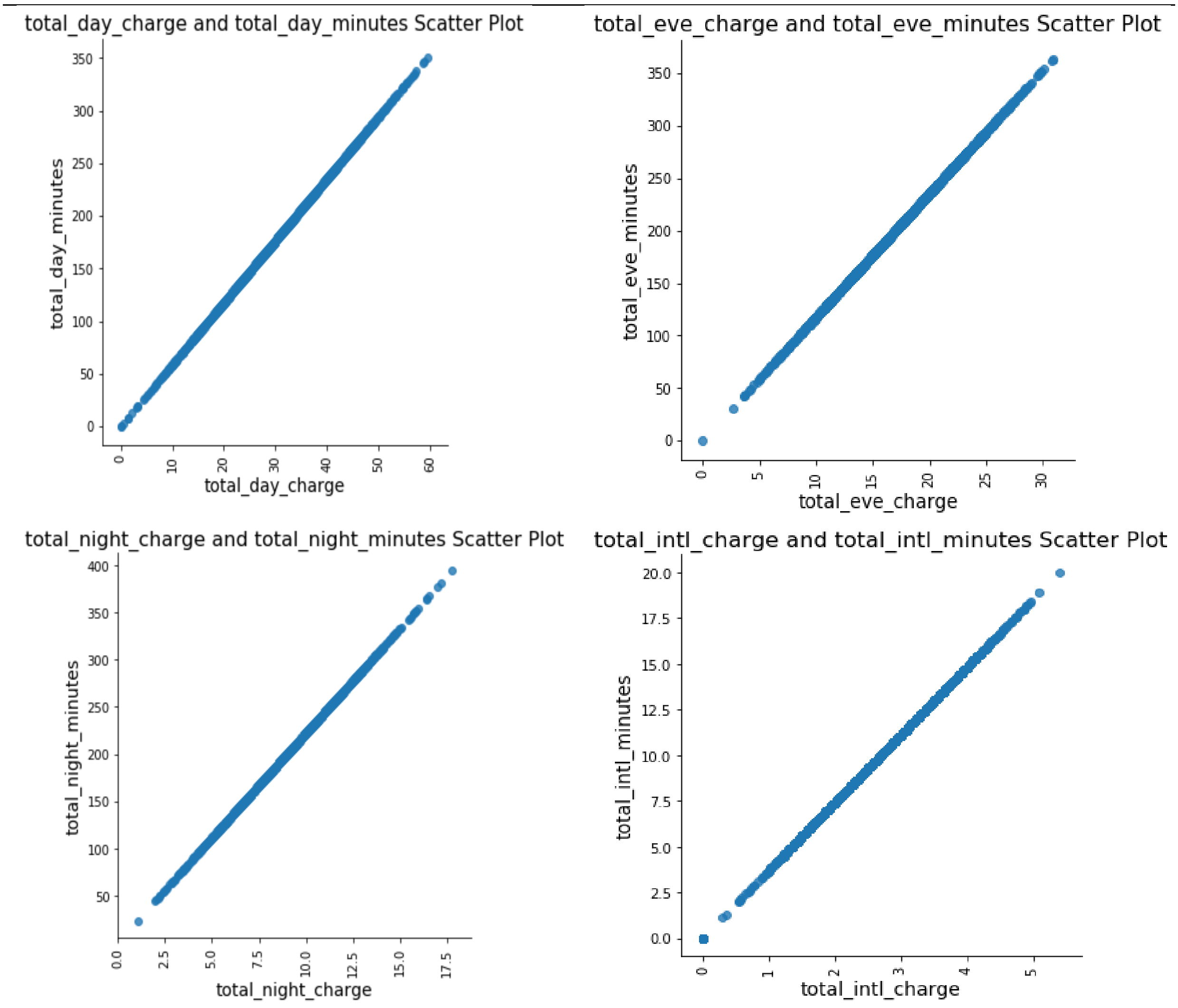




- Churn rate increasing with increase in customer service call frequency

❖ **Some of the Variable are highly correlated :-**

- total day charge & total day minute
- total eve charge & total eve minute
- total night charge & total night minute
- total intl charge & total intl minute



2.1.5 Feature Selection

Feature selection, also known as variable selection, attribute selection or variable subset selection, is the process of selecting a subset of relevant features (variables, predictors) for use in model construction. Feature selection techniques are used for four reasons:

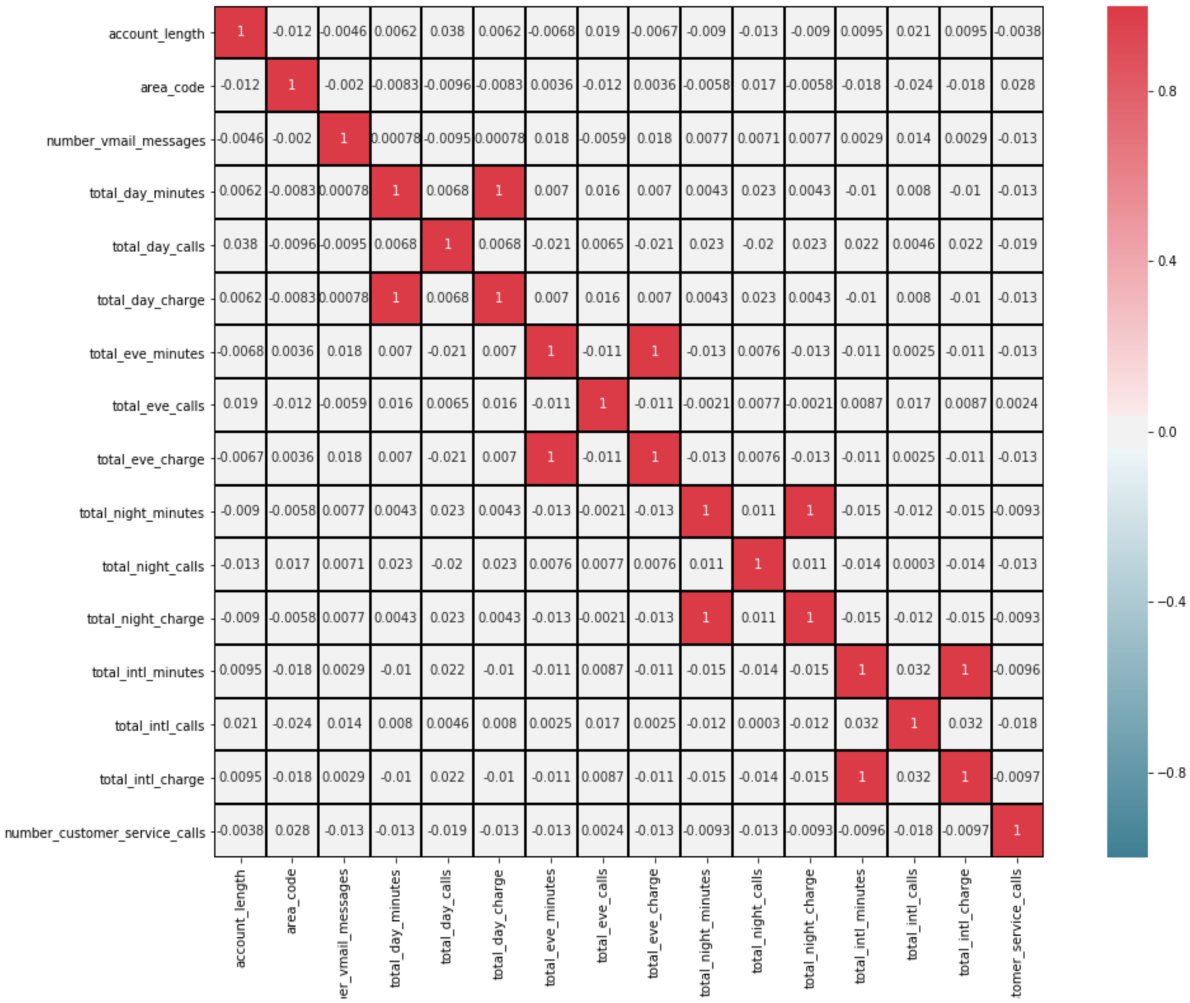
- Simplification of models to make them easier to interpret by researchers.
- Shorter training times.
- to avoid the curse of dimensionality,
- Enhanced generalization by reducing over fitting (reduction of variance).

For Continuous variable using **Correlation Matrix**

For categorical variable using **Chi Square test**

➢ Correlation Analysis

Correlation is used to test relationships between quantitative variables or categorical variables. In other words, it's a measure of how things are related



Variables are highly correlated are highlighted with red color with their corresponding score.

From the correlation plot we can see that –

- 'Total day minutes' and 'total day charges' are highly correlated
- 'Total eve minutes' and 'total eve charges' are highly correlated
- 'Total night minutes' and 'total night charges' are highly correlated
- 'Total intl minutes' and 'total intl charges' are highly correlated

- **Chi Square test – Categorical Variables**

  Chi-square test is a statistical test and mainly use for getting relation between two categorical variables. Chi-square will give us a p-value and if p value is less than 0.05 we will remove the variable because if p-value is less than 0.05 means that variable is independent and not contributing much information in explaining to our target variable.

Variables		p-values
State	:	0.002296221552011188
area_code	:	0.9150556960243712
international_plan	:	2.4931077033159556e-50
voice_mail_plan	:	5.15063965903898e-09

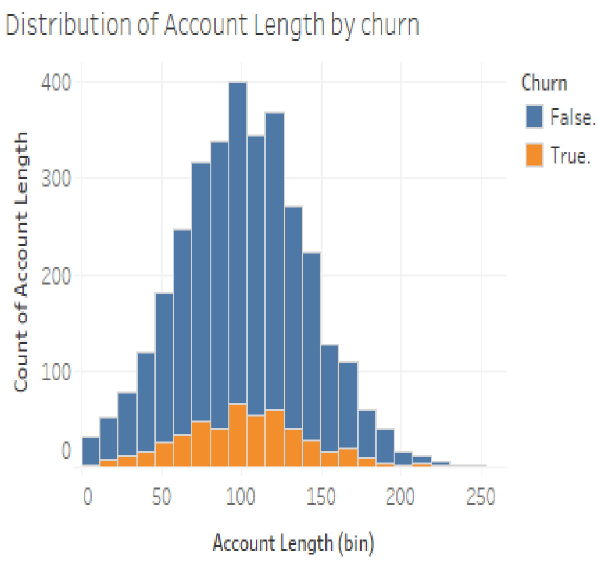
- **Removing all the redundant variables :-**

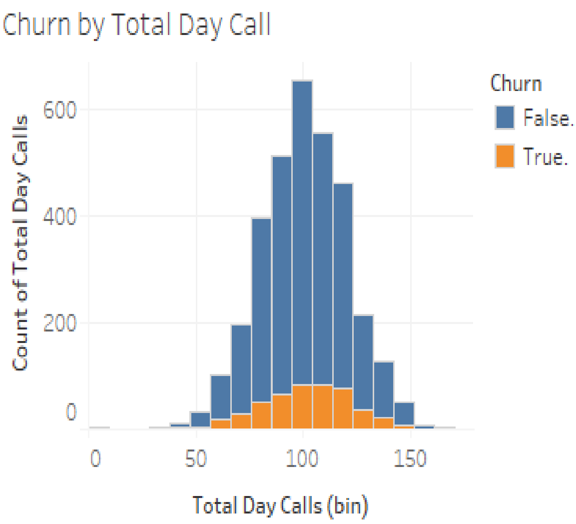
  - 'state','total_day_charge'
  - 'total_eve_charge'
  - 'total_night_charge'
  - 'total_intl_charge'

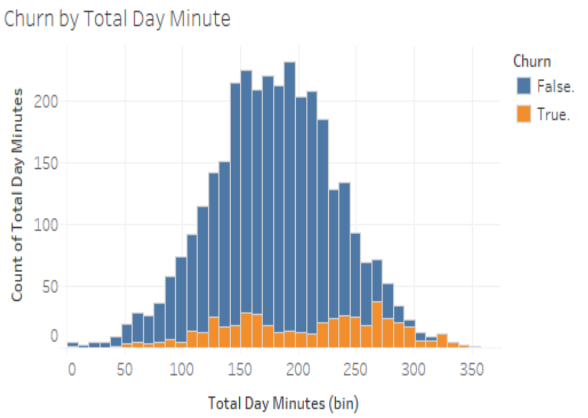
2.1.6 Feature Scaling

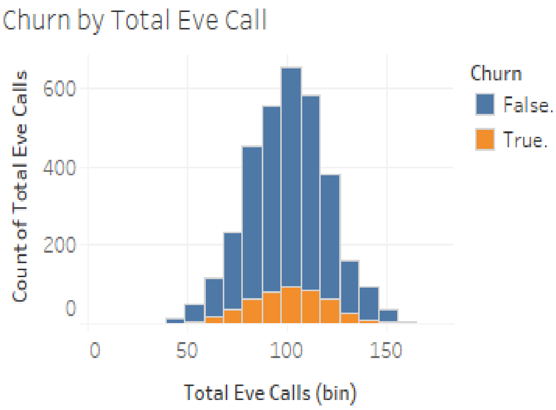
Feature scaling is a method used to standardize the range of independent variables or features of data.

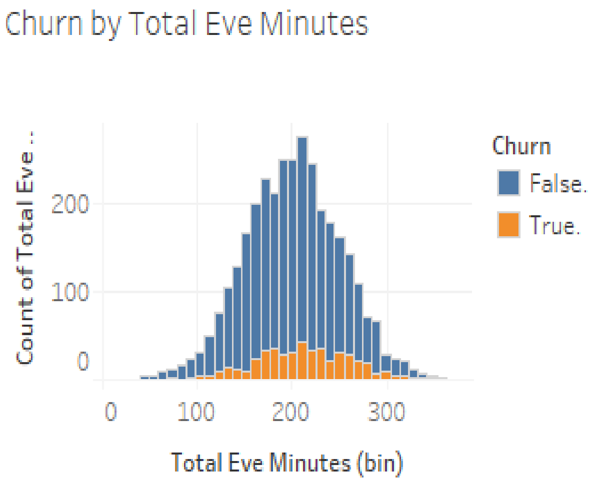
Visualizing Distribution of Continuous variables :-

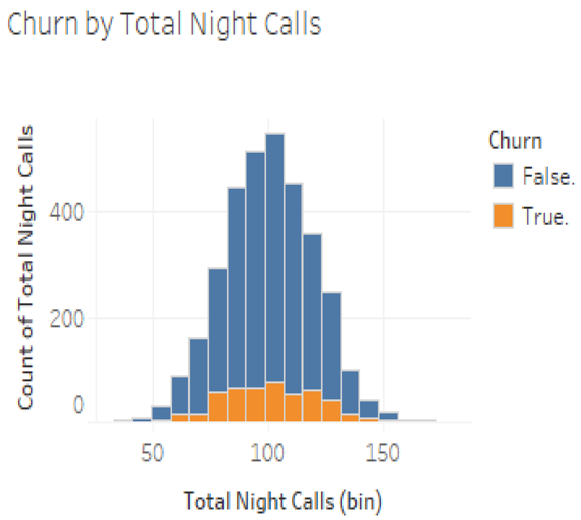


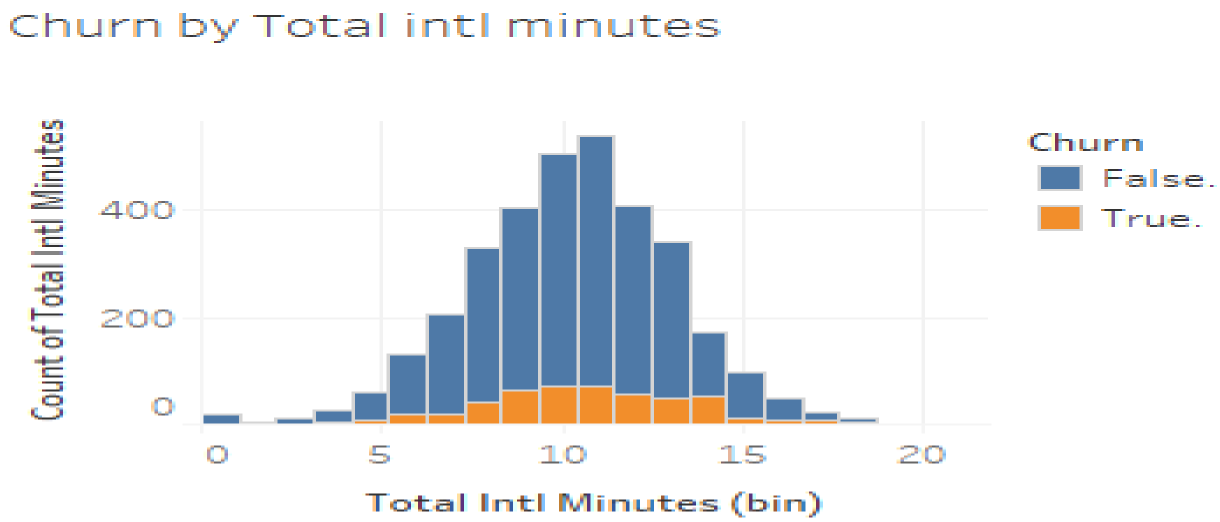












We can see that most of our continuous data distribution is uniformly.

Will use Standardization \ Z - Score here.

- **Standardization :-**

  It will convert mean or Average of each variable to Zero and the each value of variable will convert to unique standard Deviation.

  Data after Scaling

account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total eve minutes
0.676388	1	0	1	1.234697	1.566532	0.476572	-0.070599
0.149043	1	0	1	1.307752	-0.333688	1.124334	-0.108064
0.902393	1	0	0	-0.591671	1.168128	0.675883	-1.573147
-0.428526	0	1	0	-0.591671	2.196267	-1.466716	-2.742453
-0.654531	1	1	0	-0.591671	-0.240054	0.626055	-1.038776

total night minutes	total night calls	total intl minutes	total intl calls	number customer service calls	Churn
0.866613	-0.465425	-0.084995	-0.601105	-0.427868	0
1.058412	0.147802	1.240296	-0.601105	-0.427868	0
-0.756756	0.198905	0.703015	0.211502	-1.18804	0
-0.078539	-0.567629	-1.302831	1.024109	0.332305	0
-0.27627	1.067643	-0.049177	-0.601105	1.092477	0

Churn = 0 = False

Churn = 1 = True

2.1.7 Sampling (Train = Train + Validation)

Under sampling we will divide train data we have into train test split.

In Python we have used **train_test_split()** for sampling the train.csv data into train and validation data.

In R we use createDataPartition() for randomly chosen values from each class.

Both methods using stratified sampling technique to cut the data into train and validation set.

Our target variable class is imbalanced and after split of data in our train set we get

False True

1881 319

If we train our model in this data then our model training will get biased and will accurately predicting target class False more than True.

To overcome the imbalanced data problem we will go for over sampling of training data.

There are multiple approaches to do over sampling but here we will use synthetic over sampling.

In Python we have used **SMOTE**.

In R we are using **ROSE**.

2.1.7.1 SMOTE Oversampling in Python:-

SMOTE synthesize new minority instances between existing real minority instances. Imagine that SMOTE Draw lines between existing minority instances.

Smote then imagine new synthetic minority instance somewhere on that lines. Like it will generate the synthetics of two real minority cases or data points. Applying synthetic minority oversampling technique to overcome the challenge of imbalance dataset as having an imbalance dataset will have negative impact over machine learning model predictions.

```
In python we use SMOTE

Before :-

False = 1895    //     True   =  338
```

After Smote

False = 1895 // True = 1895

2.1.7.2 ROSE Oversampling in R:-

IN R we have used ROSE sampling technique. Which is similar to SMOTE, It also generating the synthetic data points and also it will under sample some random points from majority class.

R before sample

False = 1881 // True = 319

After ROSE we get :-

False = 1101 // True = 1019

Finally our data is ready to feed to the machine learning model.

Chapter 3

Modeling

Customer churn reduction is a binary classification problem. Here we have to build a model which can classify if a customer will move (churn out) or not. So for deal with particular problem we will use an Classification Model here.

There are lots of classification model present in the market.

Here we will test four particular algorithms on our train data.

1- Random Forest
2- Logistic Regression
3- K- Nearest Neighbors
4- Naïve Bayes

We will implement all four models on our preprocessed data in both Python and R in this chapter and then later on will select the final model.

3.1 Random Forest

Random Forest build multiple decision trees and merge them together to get a more accurate and stable prediction.

Summary of Random forest model:-

Python

CONFUSION MATRIX ------->>

	False	**True**
False	**925**	**30**
True	**28**	**117**

Classification paradox :------->>

Accuracy :- 94.73 %

Specificity // True Negative Rate :- 96.86 %

Sensitivity//True Positive Rate // Recall :- 80.69%

False Negative Rate :- 19.31 %

False Positive Rate :- 3.14 %

AUC -: 0.91

	precision	recall	f1-score	support
False	0.97	0.97	0.97	955
True	0.80	0.81	0.80	145
avg / total	0.95	0.95	0.95	1100

Random forest in R

We have used a little different over sampling technique in R so its result will be different not as same as python.

Summary of R model

R result according to our understanding of problem

CONFUSION MATRIX ------->>	Classification paradox :------->>	
False **True**	Accuracy :- 86.23 %	
False **842** **127**	False Negative Rate :- 17.68 %	
True **29** **135**	False Positive Rate :- 13.11 %	
AUC -: 0.8914		
95% CI	:	(0.8409, 0.8819)
No Information Rate	:	0.7688
P-Value[Acc > NIR]	:	8.087e-15
Kappa	:	0.5545
Mcnemar's Test P-Value	:	8.087e-15
Sensitivity	:	0.9667
Specificity	:	0.5153
Pos Pred Value	:	0.8689
Neg Pred Value	:	0.8232
Prevalence	:	0.7688
Detection Rate	:	0.7432
Detection Prevalence	:	0.8553
Balanced Accuracy	:	0.7410
'Positive' Class	:	1

3.2 Logistic Regression

Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes).

In logistic regression, the dependent variable is binary or dichotomous.

The goal of logistic regression is to find the best fitting (yet biologically reasonable) model to describe the relationship between the dichotomous characteristic of interest (dependent variable = response or outcome variable) and a set of independent (predictor or explanatory) variables. Logistic regression generates the coefficients (and its standard errors and significance levels) of a formula to predict a logit transformation of the probability of presence of the characteristic of interest.

We have implemented Logistic regression in R and Python both.

Summary of Logistic Regression model:-

Python

CONFUSION MATRIX ------->>			Classification paradox :------->>
	False	**True**	Accuracy :- 78.18 %
False	**752**	**203**	Specificity // True Negative Rate :- 78.74 %
True	**37**	**108**	Sensitivity//True Positive Rate // Recall :- 74.48%
			False Negative Rate :- 25.52 %
			False Positive Rate :- 21.26 %

AUC -: 0.81

	precision	recall	f1-score	support
False	0.95	0.79	0.86	955
True	0.35	0.74	0.47	145
avg / total	0.87	0.78	0.81	1100

Logistic Regression in R

R result according to our understanding of problem

CONFUSION MATRIX ------->>			Classification paradox :------->>
	False	**True**	Accuracy :- 78.20 %
False	**842**	**127**	False Negative Rate :- 28.65 %
True	**29**	**135**	False Positive Rate :- 20.63 %

AUC -: 0.8017

95% CI	:	(0.7568, 0.8057)
No Information Rate	:	0.7202
P-Value[Acc > NIR]	:	1.229e-06
Kappa	:	0.3654
Mcnemar's Test P-Value	:	< 2.2e-16
Sensitivity	:	0.9424
Specificity	:	0.3691
Pos Pred Value	:	0.7936
Neg Pred Value	:	0.7134
Prevalence	:	0.7202
Detection Rate	:	0.6787
Detection Prevalence	:	0.8553
Balanced Accuracy	:	0.6557
'Positive' Class	:	1

3.3 K- Nearest Neighbor

K-Nearest Neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

- **In k-NN classification**, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.
- **In k-NN regression**, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

We have implemented KNN in R and Python both.

Summary of KNN model:-

Python

CONFUSION MATRIX ------->>			Classification paradox :------->>
	False	**True**	Accuracy :- 78.09 %
False	**759**	**196**	Specificity // True Negative Rate :- 79.48 %
True	**45**	**100**	Sensitivity//True Positive Rate // Recall :- 68.97%
			False Negative Rate :- 31.03 %
			False Positive Rate :- 20.52 %
AUC -: 0.80			

	precision	recall	f1-score	support
False	0.94	0.79	0.86	955
True	0.34	0.69	0.45	145
avg / total	0.86	0.78	0.81	1100

KNN in R

R result according to our understanding of problem

CONFUSION MATRIX ------->>			Classification paradox :------->>
	False	**True**	Accuracy :- 78.82 %
False	**805**	**164**	False Negative Rate :- 46.34 %
True	**76**	**88**	False Positive Rate :- 16.92 %

95% CI	:	(0.7632, 0.8116)
No Information Rate	:	0.7776
P-Value[Acc > NIR]	:	0.2063
Kappa	:	0.3004
Mcnemar's Test P-Value	:	1.956e-08
Sensitivity	:	0.9137
Specificity	:	0.3492
Pos Pred Value	:	0.8308
Neg Pred Value	:	0.5366
Prevalence	:	0.7776
Detection Rate	:	0.7105
Detection Prevalence	:	0.8553
Balanced Accuracy	:	0.6315
'Positive' Class	:	1

3.4 Naïve Bayesian

The Naive Bayesian classifier is based on Bayes' theorem with the independence assumptions between predictors. A Naive Bayesian model is easy to build, with no complicated iterative parameter estimation which makes it particularly useful for very large datasets. Despite its simplicity, the Naive Bayesian classifier often does surprisingly well and is widely used because it often outperforms more sophisticated classification methods

Algo :-

Bayes theorem provides a way of calculating the posterior probability, P(c|x), from P(c), P(x), and P(x|c). Naive Bayes classifier assumes that the effect of the value of a predictor (x) on a given class (c) is independent of the values of other predictors. This assumption is called class conditional independence.

We have implemented KNN in R and Python both.

Summary of Naïve Bayesian model:-

Python

CONFUSION MATRIX ------->>		
	False	**True**
False	**754**	**201**
True	**34**	**111**

AUC -: 0.82

Classification paradox :------->>

Accuracy :- 78.64 %

Specificity // True Negative Rate :- 78.95 %

Sensitivity//True Positive Rate // Recall :- 76.55%

False Negative Rate :- 23.45 %

False Positive Rate :- 21.05 %

	precision	recall	f1-score	support
False	0.94	0.79	0.86	955
True	0.34	0.69	0.45	145
avg / total	0.86	0.78	0.81	1100

Naïve Bayesian in R

R result according to our understanding of problem

CONFUSION MATRIX ------->>		
	False	**True**
False	**826**	**143**
True	**48**	**116**

Classification paradox :------->>

Accuracy :- 83.14 %

False Negative Rate :- 26.27 %

False Positive Rate :- 14.76 %

95% CI	:	(0.8083, 0.8528)
No Information Rate	:	0.7714
P-Value[Acc > NIR]	:	3.972e-07
Kappa	:	0.4512
Mcnemar's Test P-Value	:	1.035e-11
Sensitivity	:	0.9451
Specificity	:	0.4479
Pos Pred Value	:	0.8524
Neg Pred Value	:	0.7073
Prevalence	:	0.7714
Detection Rate	:	0.7290
Detection Prevalence	:	0.8553
Balanced Accuracy	:	0.6965
'Positive' Class	:	1

Chapter 4

Model Evaluation

Model evaluation is the process of choosing between models, different model types, tuning parameters, and features. Better evaluation processes lead to better, more accurate models in applications.

In previous chapter we have built :-

- Random Forest
- Logistic Regression
- KNN
- Naïve Bayesian

We have chosen Classification Accuracy Matrix (Accuracy // False Negative Rate // False Positive Rate // Precision // Sensitivity or Recall // Specificity) matrix as our evaluation matrix.

Before evaluating the final model our of our all model let's get a brief about classification matrix.

- **Accuracy** : the proportion of the total number of predictions that were correct.
- **Positive Predictive Value or Precision** : the proportion of positive cases that were correctly identified.
- **Negative Predictive Value** : the proportion of negative cases that were correctly identified.
- **Sensitivity or Recall** : the proportion of actual positive cases which are correctly identified.
- **Specificity** : the proportion of actual negative cases which are correctly identified.
- **False Positive Rate (Type –I error):** False positive, commonly called a "false alarm", is a result that indicates a given condition exists, when it does not .
- **False Negative Rate (Type – II error)**: false negative, is a test result that indicates that a condition does not hold, while in fact it does.

Let's see False Negative and False Positive according to our problem statement.

If in actual any customer is not churning and our model predict that he /she will churn. Then it's okay we can deal with it , like on prior we will start putting more effort on the customer so that he/she won't churn out.

But in other case if our model predict that this particular customer won't churn out and in actual he will churn out, then there might be a big problem. Because in this scenario our client will lose some important clients .

From both cases it's important to make a good trade off that my model would predict more accurate and have low false negative rate and also not much of false positive rate.

Let's check results of R and Python

Python Results				
Models	**Random Forest**	**Logistic Regression**	**KNN**	**Naïve Bayes**
Accuracy	94.73%	78.18%	78.09%	78.64%
False Positive Rate	19.31%	25.52%	31.03%	23.45%
False Negative Rate	3.14	21.16%	20.52%	21.05%

As in Python Random forest has the best results for our problem.

R – Results				
Models	**Random Forest**	**Logistic Regression**	**KNN**	**Naïve Bayes**
Accuracy	86.23%	78.20%	78.82%	83.14%
False Positive Rate	17.68%	28.65%	46.34%	26.27%
False Negative Rate	13.11%	20.63%	16.92%	14.76%

In R Random forest has the best results for our problem.

Random Forest has the best accuracy and lowest false negative rate and also lowest false positive rate.

This was just the basic model as to test which model work best with our preprocessed data. Let's make it better by cross validating and tuning the parameters.

4.1 Performance Tuning of Random forest

In performance tuning we apply different combination over our model and try to enhance the performance by applying different combination of hyper parameters. Random forest has a lot of parameters so we will tune few of them which make a vast impact over accuracy and all result of model.

We have tuned random forest model in both R and Python using grid search and randomsearchCV.

Python :

Hyper Parameter optimization with RandomSearch CV :

Under randomsearchCV we pass some parameters so that it will fit those parameters in the model and get the result.

We get (**ntree = 500, criterion = gini, max_features = auto**)

Results after parameter optimization

CONFUSION MATRIX ------->>			Classification paradox :------->>				
	False	**True**	Accuracy :- 95.09 %				
False	**927**	**28**	Specificity // True Negative Rate :- 97.07 %				
True	**26**	**119**	Sensitivity//True Positive Rate // Recall :- 82.07%				
			False Negative Rate :- 17.93 %				
			False Positive Rate :- 2.93 %				
AUC -: 0.91							
				precision	recall	f1-score	support
			False	0.97	0.97	0.97	955
			True	0.81	0.82	0.82	145
			avg / total	0.95	0.95	0.95	1100

In R :

We use grid search CV to train our model and for searching hyper parameter tuning.

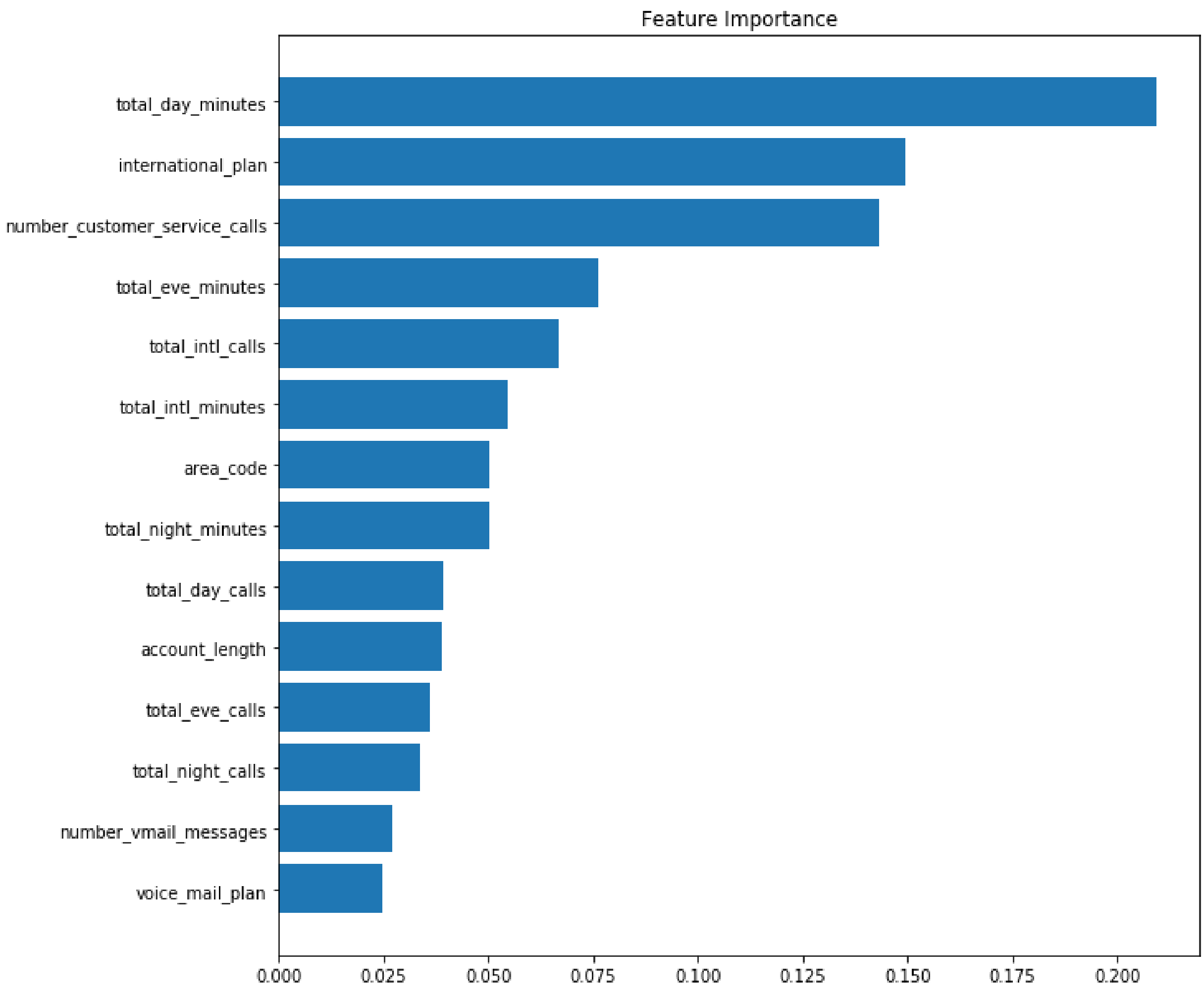
We get (mtry = 4, ntree = 800)

R result :

CONFUSION MATRIX ------->>			Classification paradox :------->>	
	False	**True**	Accuracy :- 86.5 %	
False	**844**	**125**	False Negative Rate :- 17.07 %	
True	**28**	**136**	False Positive Rate :- 12.90 %	
AUC = 89.47				
95% CI			:	(0.8437, 0.8843)

No Information Rate	:	0.7696
P-Value[Acc > NIR]	:	4.453e-16
Kappa	:	0.5622
Mcnemar's Test P-Value	:	8.147e-15
Sensitivity	:	0.9679
Specificity	:	0.5211
Pos Pred Value	:	0.8710
Neg Pred Value	:	0.8293
Prevalence	:	0.7696
Detection Rate	:	0.7449
Detection Prevalence	:	0.8553
Balanced Accuracy	:	0.7445
'Positive' Class	:	1

4.2 Best Parameters we get after tuning the model :



Variables :- total_dat_minutes, International_plan, number of customer service calls which makes major impact on performance of our predictions.

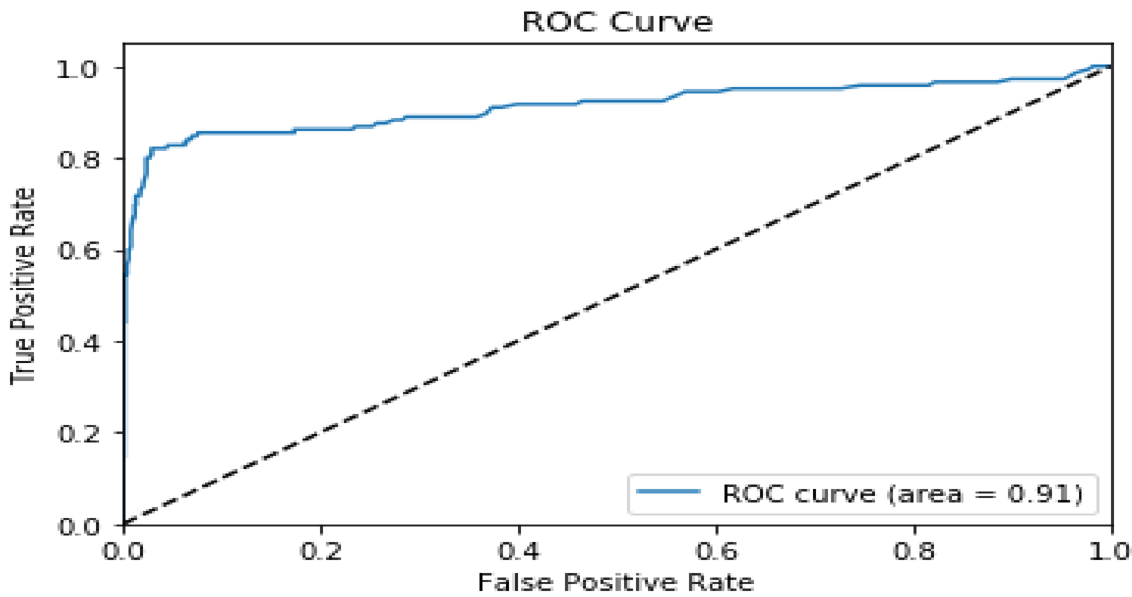
4.3 Cost Effective :

In case of cost effecting we can play a lot more with our model. As of till now we have used default threshold in R and Python both and get a good result. But we can enhance it more by making changes in the model threshold measure. While making any prediction that whether he/she will churn or not, there always single events occurs that either he will churn out or he will not.

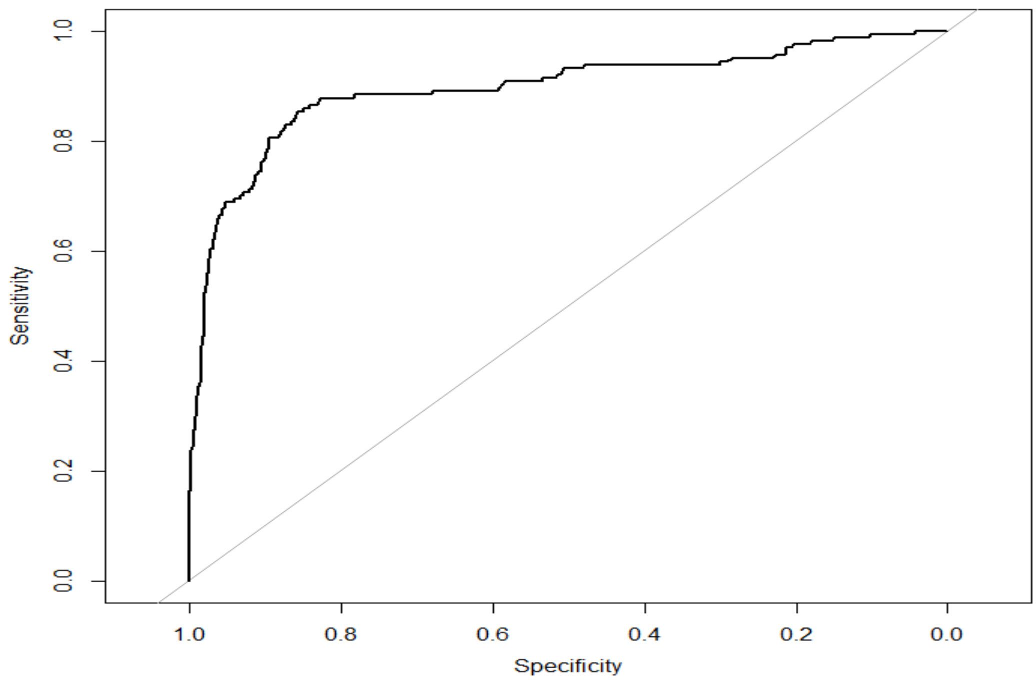
We can change that threshold so that then our model will become better. Overall still with imbalanced data our model is doing well.

ROC curve:

Python model :-



R model : **Auc = 89.47**



4.4 Final Test Data Prediction :

As in our test.csv file Churn target is given, So let's Check the prediction accuracy on our final test data.

We have 1667 observations and 21 columns in test.csv

Python Result:-

CONFUSION MATRIX ------->>

	False	True
False	1331	112
True	37	187

AUC -: 0.92

Classification paradox :------->>

Accuracy :- 91.06 %

Specificity // True Negative Rate :- 92.24 %

Sensitivity//True Positive Rate // Recall :- 83.48%

False Negative Rate :- 16.52 %

False Positive Rate :- 7.76 %

	precision	recall	f1-score	support
False	0.97	0.92	0.95	1443
True	0.63	0.83	0.72	224
avg / total	0.93	0.91	0.92	1667

R result :

CONFUSION MATRIX ------->>

	False	True
False	1243	200
True	35	189

Classification paradox :------->> AUC = 91.74

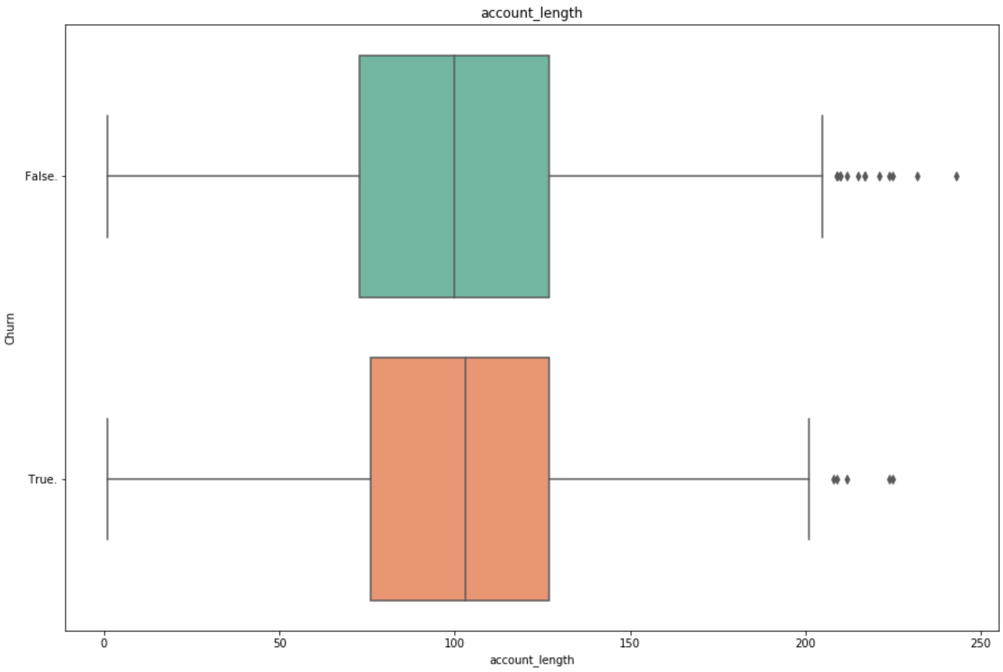
Accuracy :- 85.92 %

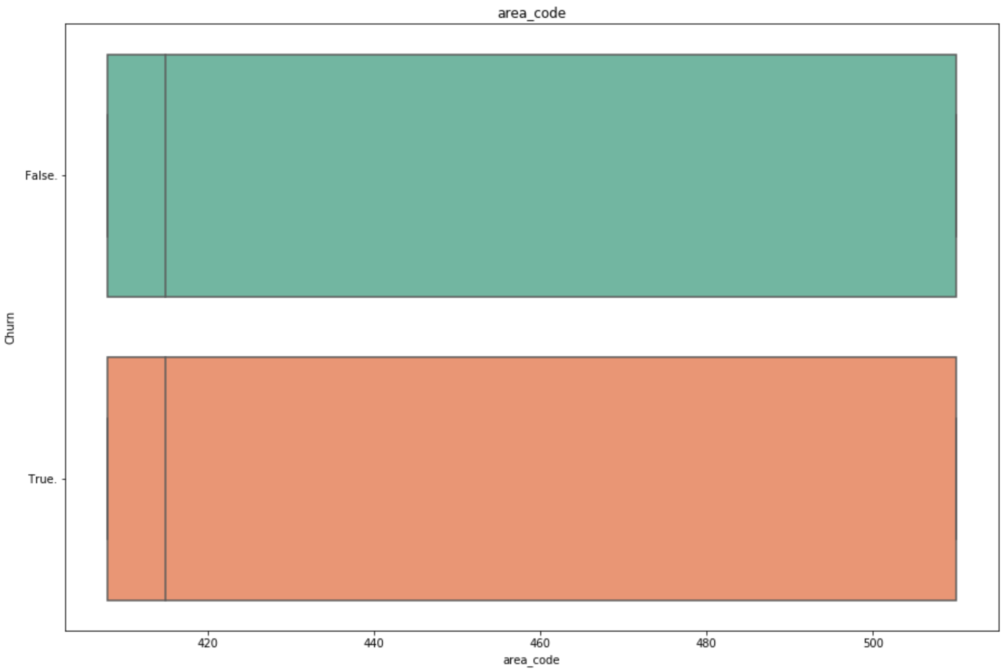
False Negative Rate :- 15.63 %

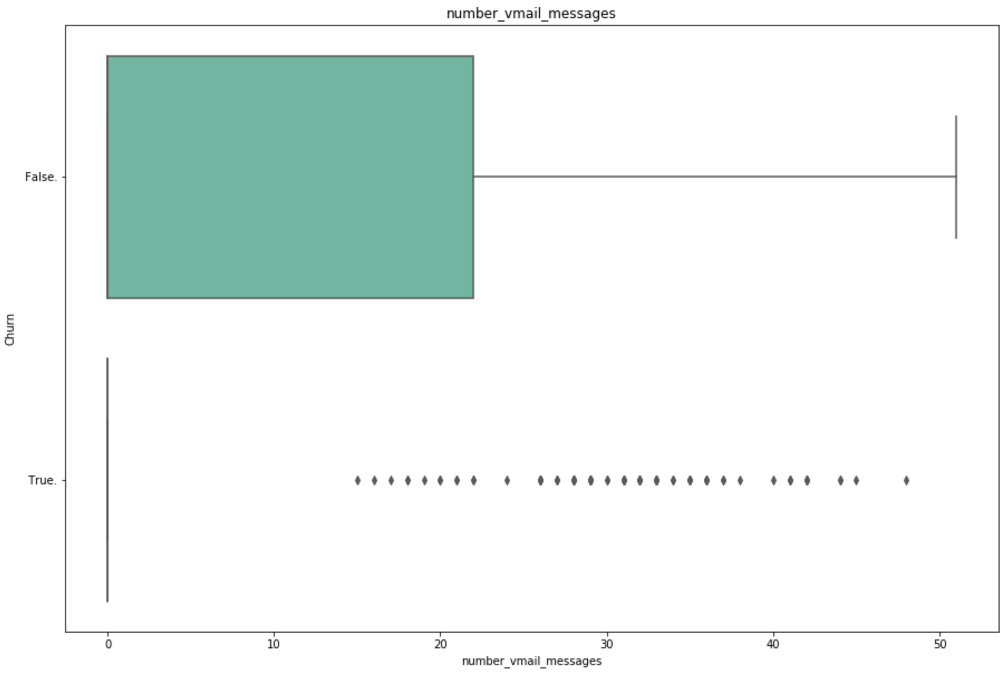
False Positive Rate :- 13.86 %

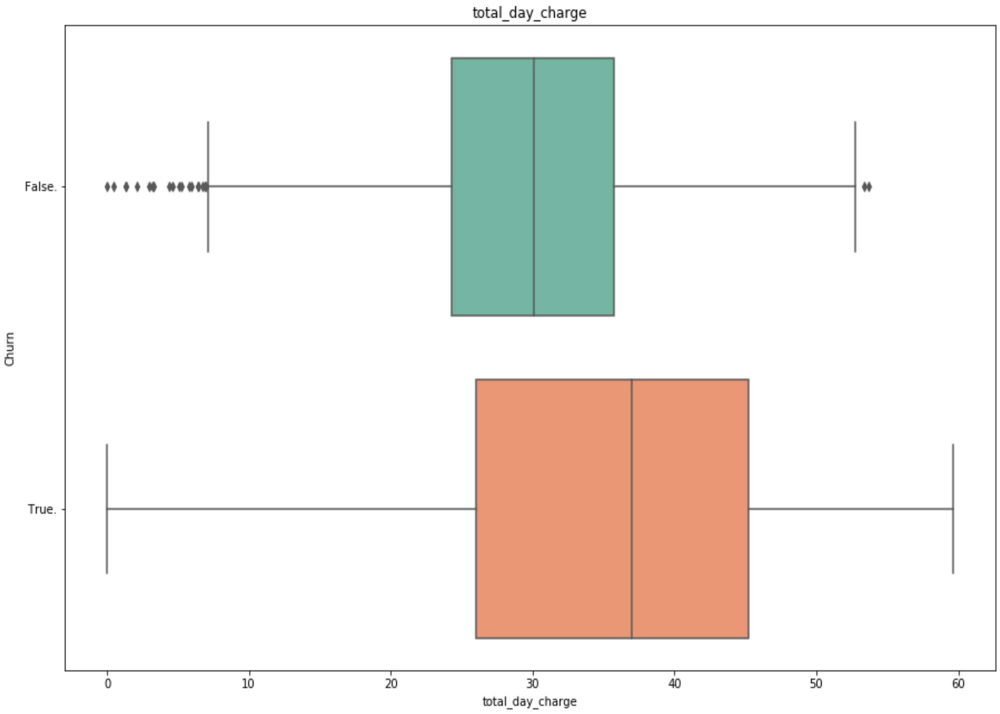
95% CI	:	(0.8414, 0.8754)
No Information Rate	:	0.7666
P-Value[Acc > NIR]	:	< 2.2e-16
Kappa	:	0.5378
Mcnemar's Test P-Value	:	< 2.2e-16
Sensitivity	:	0.9726
Specificity	:	0.4859
Pos Pred Value	:	0.8614
Neg Pred Value	:	0.8438
Prevalence	:	0.7666
Detection Rate	:	0.7457
Detection Prevalence	:	0.8656
Balanced Accuracy	:	0.7292
'Positive' Class	:	1

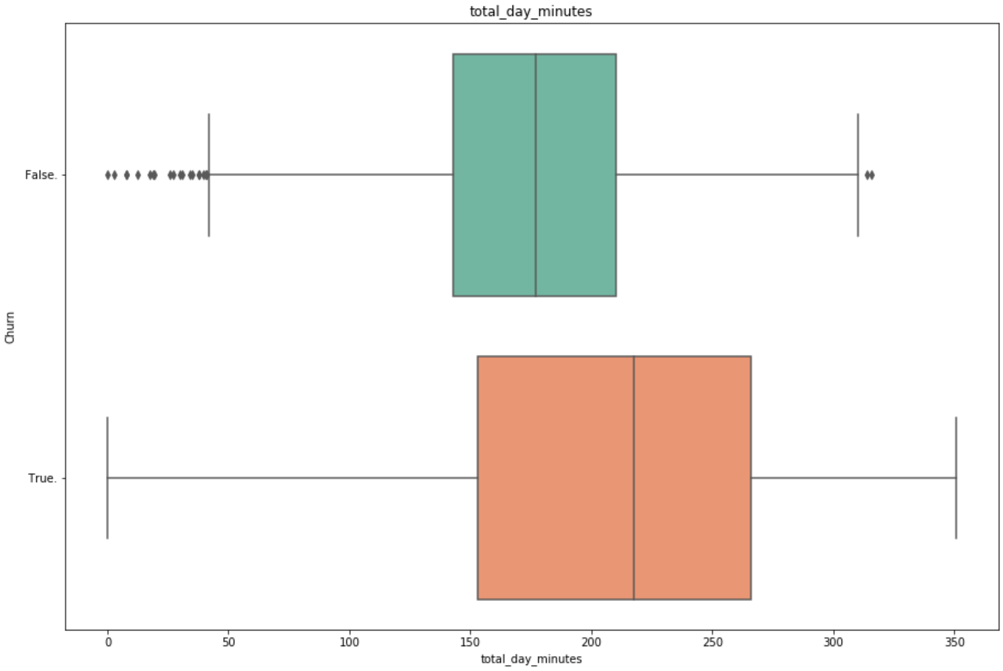
Appendix A : Extra Figures

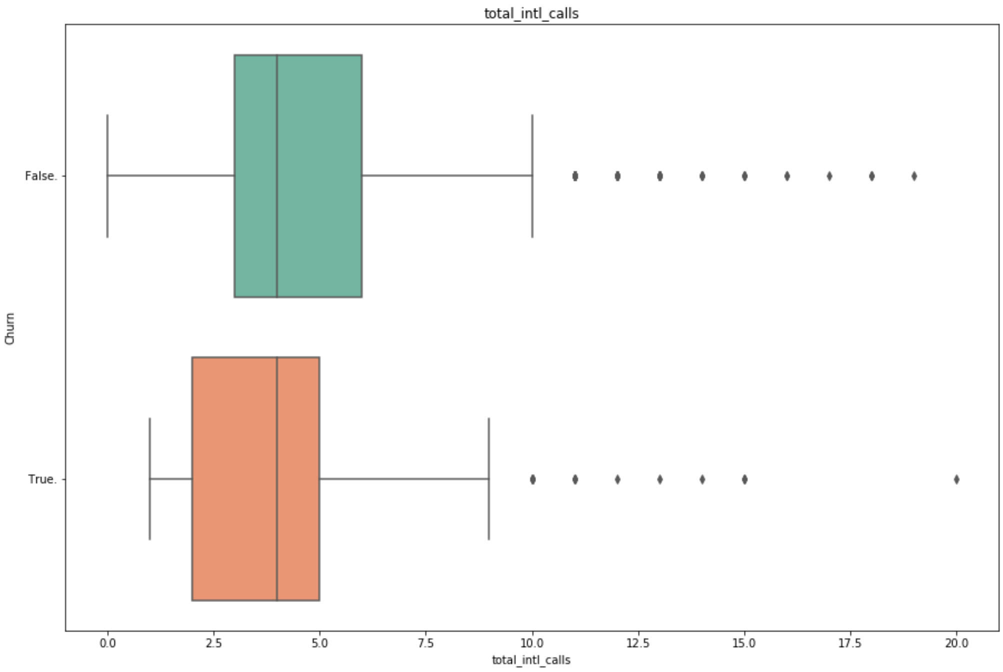


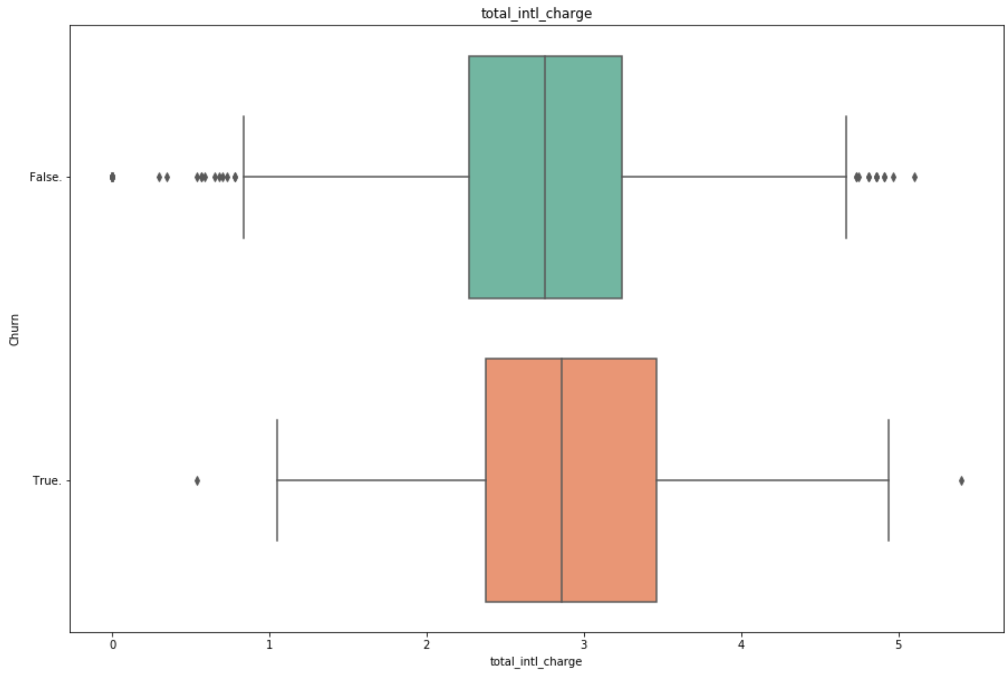


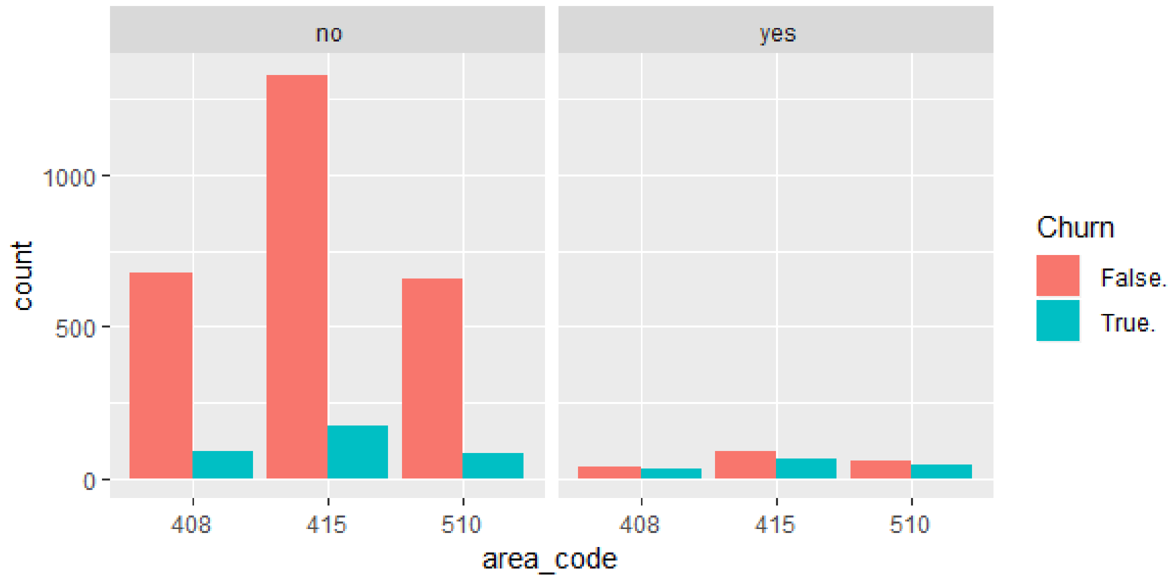

number_vmail_messages
Churn
False.
True.
0
10
20
30
40
50
number_vmail_messages


total_day_charge
Churn
False.
True.
0
10
20
30
40
50
60
total_day_charge


total_day_minutes
False.
True.
Churn
0
50
100
150
200
250
300
350
total_day_minutes

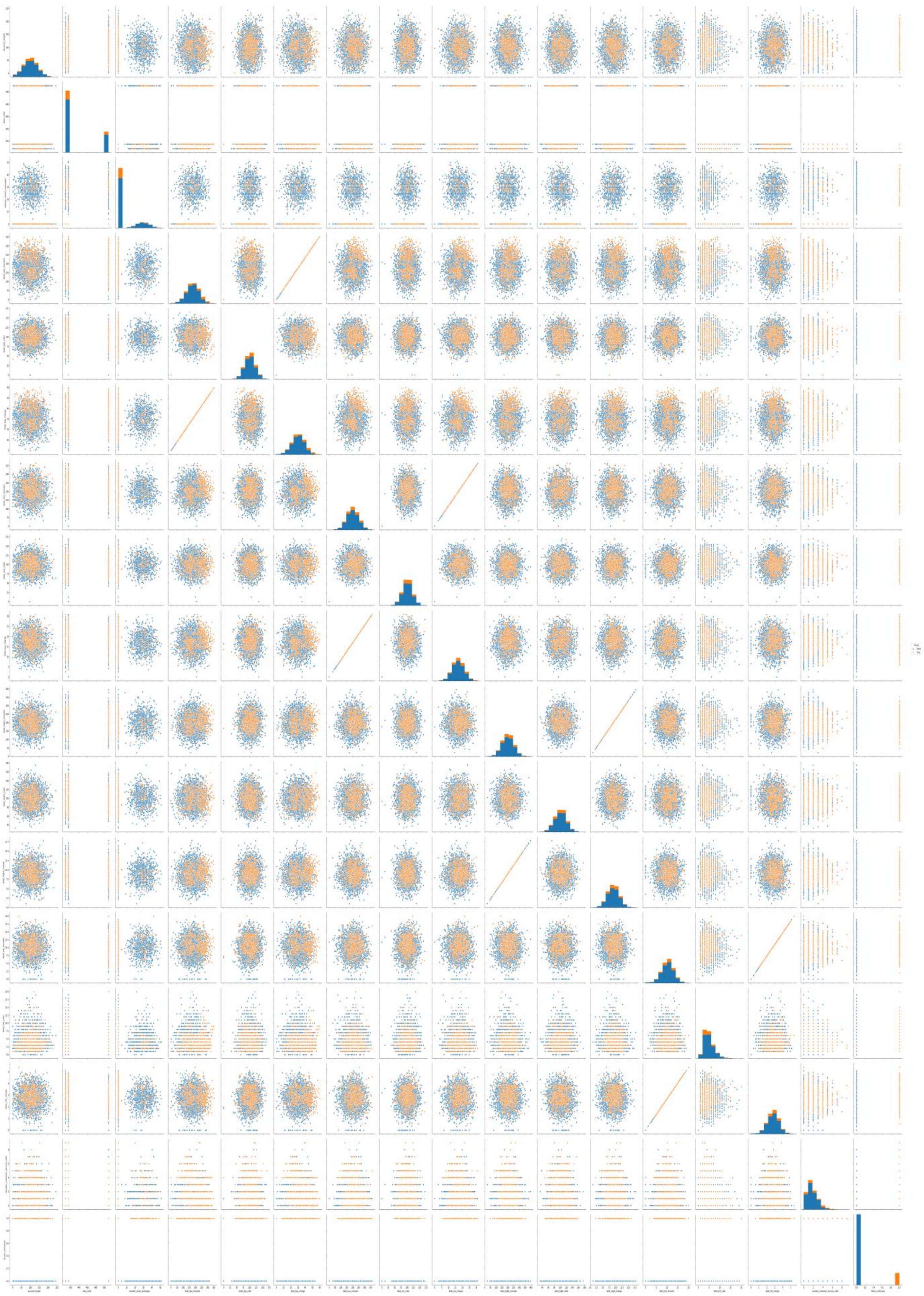

total_intl_calls
False.
True.
Churn
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
total_intl_calls

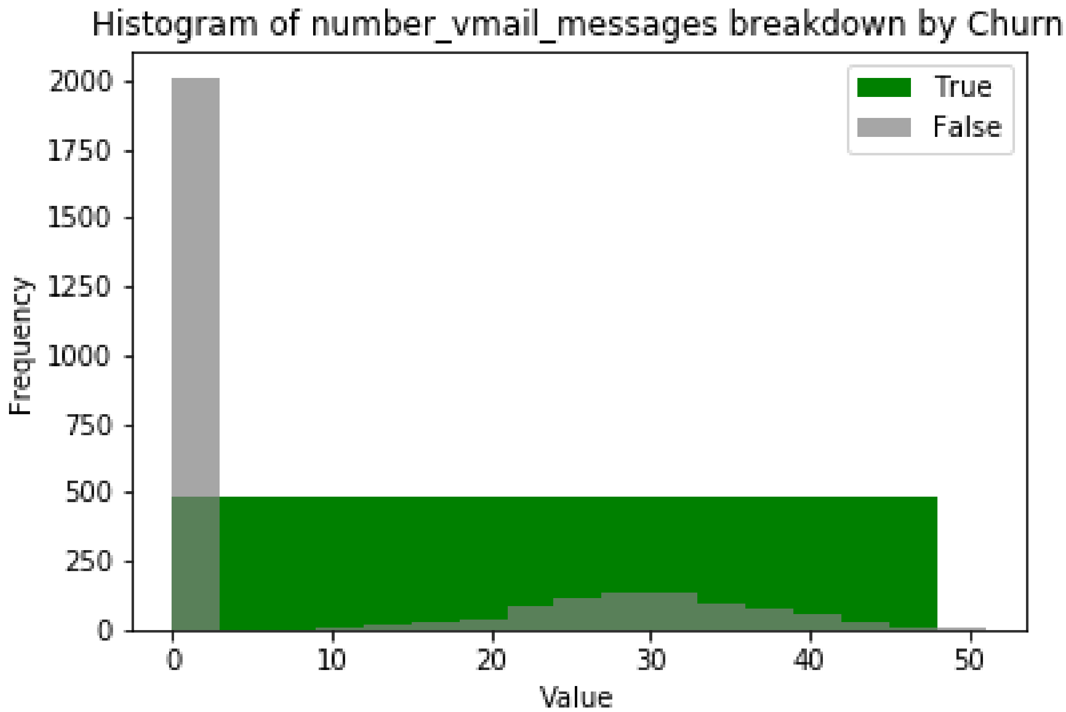


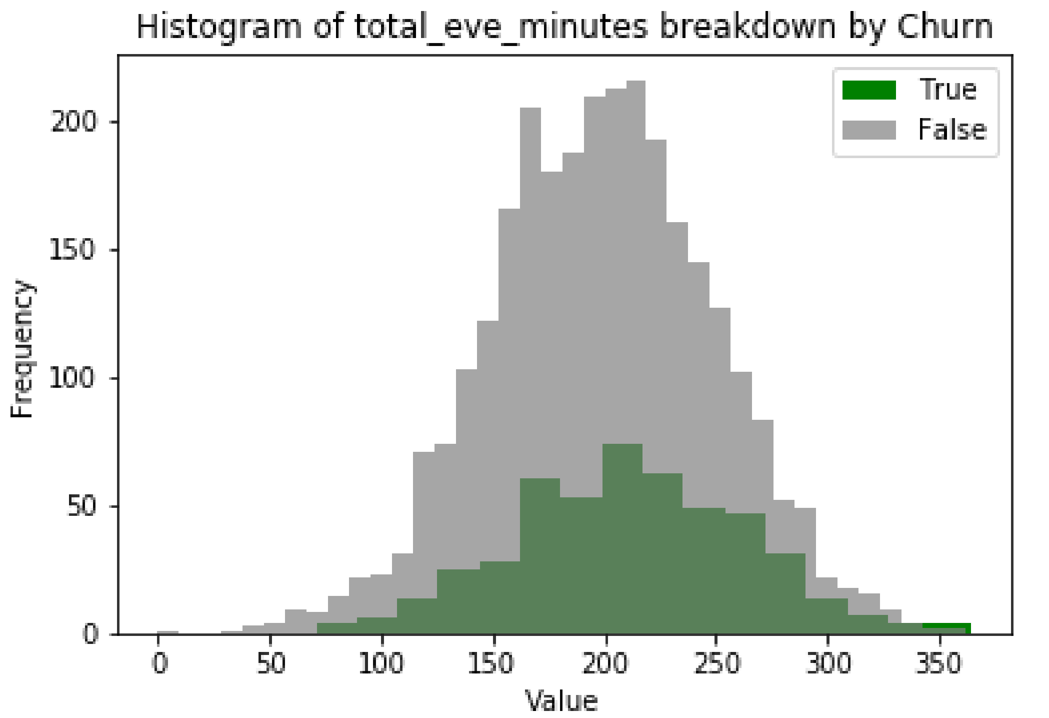


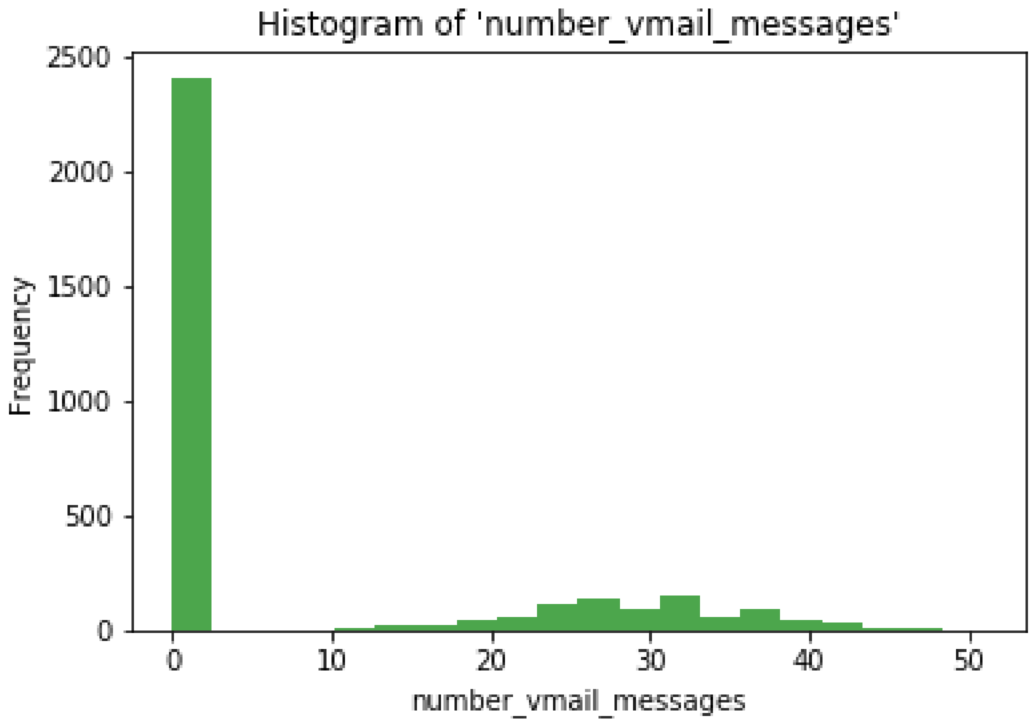
International plan // Area code // Churn

Pair Plot of Entire Test Data set

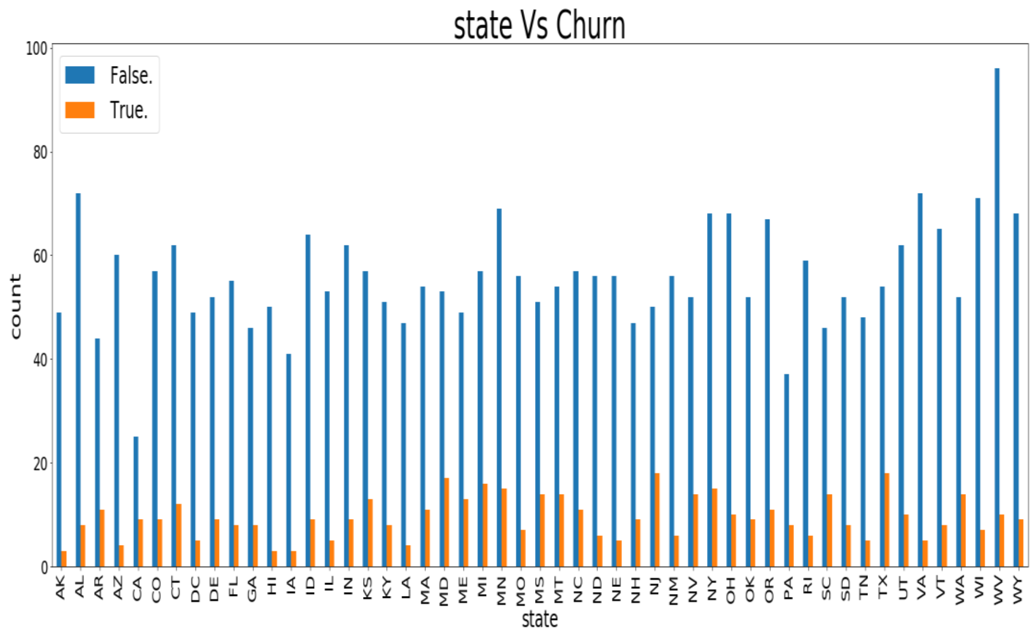



Histogram of number_vmail_messages breakdown by Churn
True
False
Frequency
Value
2000
1750
1500
1250
1000
750
500
250
0
0
10
20
30
40
50


Histogram of total_eve_minutes breakdown by Churn
True
False
Frequency
Value
200
150
100
50
0
0
50
100
150
200
250
300
350



Histogram of 'number_vmail_messages'
Frequency
number_vmail_messages



state Vs Churn
False.
True.
count
state

Appendix B : Python Code

```
# Importing Libraries

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split,RandomizedSearchCV
from imblearn.over_sampling import SMOTE

from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from fancyimpute import KNN

from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve,auc,roc_auc_score
%matplotlib inline

#Setting working directory
os.chdir("C://Users/parve/Documents/Data_Science_Project/")
print(os.getcwd())

#Loading Dataset
train_original = pd.read_csv("./Data/Train_data.csv")
test_original = pd.read_csv("./Data/Test_data.csv")
#Creating Duplicate instances of data for Preprocessing and exploration
train = train_original.copy()
test = test_original.copy()

#Exploring data
train.head(5)

#Checking info of data as data types and rows n cols
train.info()

train.describe()

#calculating all the unique values for all df columns
for i in train.columns:
    print(i,' --------> ',len(train[i].value_counts()))

#Replacing spaces from columns name with underscore
train.columns = train.columns.str.replace(" ","_")
test.columns = test.columns.str.replace(" ","_")  #for test set also ch
anging names

#Changing area_code type to categorical in both test and train data set
train['area_code'] = train['area_code'].astype('object')
test['area_code'] = test['area_code'].astype('object')

#Droping phone_number
```

```
train = train.drop('phone_number',axis=1)
test = test.drop('phone_number',axis=1)

#All continous var list
cname = train.columns[(train.dtypes=="float64")|(train.dtypes=="int64")
].tolist()
print(cname)

#All categorical var and removing target var
cat_names = train.select_dtypes(exclude=np.number).columns.tolist()
cat_names.remove('Churn')
cat_names

#Checking missing Values

#Checking missing values in train dataset
print(train.isnull().sum())  #no missing value present in the train dat
a

#Checking missing values in test data set
print(test.isnull().sum())  #no missing value present in the test data

#Viszualizing data
#Target Variable data distribution
plt.figure(figsize=(8,6))
sns.countplot(x = train.Churn,palette='muted')
plt.xlabel('Customer churn', fontsize= 15)
plt.ylabel('Count', fontsize= 15)
plt.title("Distribution of Churning Vs Not Churning Customer",fontsize=
20)
plt.show()
```

#We can see that it's a target class imbalance problem¶

```
#Groupby --> size to represent ---> unstack the category
#train.groupby(["state", "Churn"]).size().unstack(level=-1).head()

#Relationational bar graph for checking data distribution with respect
to target variable
def diff_bar(x,y):
    train.groupby([x,y]).size().unstack(level=-1).plot(kind='bar', figs
ize=(30,10))
    plt.xlabel(x,fontsize= 25)
    plt.ylabel('count',fontsize= 25)
    plt.legend(loc=0,fontsize= 25)
    plt.xticks(fontsize=20, rotation=90)
    plt.yticks(fontsize=20)
    plt.title("{X} Vs {Y}".format(X=x,Y=y),fontsize = 40)
    #plt.savefig("{X}_Vs_{Y}.png".format(X=x,Y=y))
    plt.show()

#State Wise Churning of customer
diff_bar('state','Churn')


#area_code Wise Churning of customer
diff_bar('area_code','Churn')
```

```
#International_Plan Wise Churning of customer
diff_bar('international_plan','Churn')

#Number of Customer_Service Call Wise Churning of customer
diff_bar('number_customer_service_calls','Churn')

#No. of Customer Churning and had a Voice mail plan
diff_bar('voice_mail_plan','Churn')

#fig = plt.figure()
#fig = sns.pairplot(train,hue='Churn',size=2.5)
#plt.show()
#fig.savefig('pairplot.png')

#Scatter plot function
def diff_scattr(x,y):
    fig = plt.figure()
    fig = sns.lmplot(x,y, data=train,fit_reg=False)
    plt.xlabel(x,fontsize= 14)
    plt.ylabel(y,fontsize= 14)
    plt.xticks(fontsize=10, rotation=90)
    plt.yticks(fontsize=10)
    plt.title("{X} and {Y} Scatter Plot".format(X=x,Y=y),fontsize = 16)
    #fig.savefig("{X}_and_{Y}_Scatter_Plot..png".format(X=x,Y=y))
    plt.show()


#Total intl charge and Total intl Minute
diff_scattr('total_intl_charge','total_intl_minutes')


## Total night charge and Total night Minute
diff_scattr('total_night_charge','total_night_minutes')

#Total eve charge and Total eve Minute
diff_scattr('total_eve_charge','total_eve_minutes')

#Total day charge and Total Day Minute
diff_scattr('total_day_charge','total_day_minutes')

#Changing Categorical colum values to numeric codes


#function for converting cat to num codes
def cat_to_num(df):
    for i in range(0, df.shape[1]):
        #print(i)
        if(df.iloc[:,i].dtypes == 'object'):
            df.iloc[:,i] = pd.Categorical(df.iloc[:,i])
            df.iloc[:,i] = df.iloc[:,i].cat.codes
            df.iloc[:,i] = df.iloc[:,i].astype('object')
    return df

train = cat_to_num(train)
test = cat_to_num(test)

#Anomaly Detections or Outlier Analysis¶
```

```
#Skipping outlier analysis as Their is already an class imbalance impac
#t over data.
#Also we have finalize Random forest as our final model and it can easi
#ly deal with outliers. :-)


# # #Plotting Box Plot
# for i in cname:
#     plt.figure()
#     plt.clf() #clearing the figure
#     sns.boxplot(train[i],palette="Set2")
#     plt.title(i)
#     plt.show()

# #Treating Out Liers and Converting them to nan
# for i in cname:
#     #print(i)
#     q75, q25 = np.percentile(train.loc[:,i], [75 ,25])
#     iqr = q75 - q25
#     minn = q25 - (iqr*1.5)
#     maxx = q75 + (iqr*1.5)
# #Converting to nan
#     train.loc[train.loc[:,i] < minn,i] = np.nan
#     train.loc[train.loc[:,i] > maxx,i] = np.nan
#     print('{var} --------- :- {X}   Missing'.format(var = i, X = (tra
in.loc[:,i].isnull().sum())))


# #Apply KNN imputation algorithm for imputing missing values
# train = pd.DataFrame(KNN(k = 3).complete(train), columns = train.colu
mns)


# for i in cat_names:
#     #print(i)
#     train[i] = train[i].astype('object')

# #Checking Missing value
# print(train[cname].isnull().sum())


# Feature Selections

#Setting up the pane or matrix size
f, ax = plt.subplots(figsize=(18,12))  #Width,height

#Generating Corelation Matrix
corr = train[cname].corr()

#Plot using Seaborn library
sns.heatmap(corr,mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.dive
rging_palette(220,10, as_cmap=True),\
            square=True, ax=ax,annot=True,linewidths=1 , linecolor= 'bl
ack',vmin = -1, vmax = 1)

plt.show()
#f.savefig('heatmap.png')
```

#Chi-Square for Categorical variables

```
#checking Relation b/w categorical variables with respect to target var
from scipy.stats import chi2_contingency
for i in cat_names:
    print(i)
    #As we know imput to chi square is always a contiguency table so we
generating it using crostab function present in pd
    chi2, p, dof, ex =chi2_contingency(pd.crosstab(train['Churn'],train
[i]))
    #as above pd.crosstab(dependent variable , independent variable)
    print(p)

#chi2 = Actual chi square test value
#p = pvalue
#dof = degree of freedom
#ex = expected value

# As if p value is less than 0.05 then we will reject null hypothesis
#Null = both the variables are independent
#Alternate = Both the variables are not independent

#Removing correlated variable & the variable which doesn't contain any
meaning full info
rmev = ['state','total_day_charge','total_eve_charge','total_night_char
ge','total_intl_charge']
train = train.drop(rmev,axis=1)
test = test.drop(rmev,axis=1)


#Updating values _after removal of var
cname = ['account_length', 'number_vmail_messages', 'total_day_minutes'
, 'total_day_calls', 'total_eve_minutes',
         'total_eve_calls', 'total_night_minutes', 'total_night_calls',
'total_intl_minutes', 'total_intl_calls',
         'number_customer_service_calls']

#All categorical var and removing target var
cat_names = ['area_code', 'international_plan', 'voice_mail_plan']

print('cname :- {}'.format(cname))
print()
print('cat_name :- {}'.format(cat_names))
```

Feature Scaling

Checking Distribution of data

```
#Checking distribution of data via pandas visualization
train[cname].hist(figsize=(20,20),color='g',alpha = 0.7)
#plt.savefig('distribution.png')
plt.show()


# #Histogram breaks down by target variable
def plot_hist_y(x,y):
```

```
    plt.hist(list(x[y == 1]),color='green',label='True',bins='auto')
    plt.hist(list(x[y == 0]),color='grey', alpha = 0.7, label='False',b
ins='auto')
    plt.title("Histogram of {var} breakdown by {Y}".format(var = x.name
,Y=y.name))
    plt.xlabel("Value")
    plt.ylabel("Frequency")
    plt.legend(loc="upper right")
    plt.savefig("Histogram of {var} breakdown by {Y}.png".format(var =
x.name,Y=y.name))
    plt.show()


for i in cname:
    #print(i)
    plot_hist_y(train[i],train.Churn)
```

As most of the data is uniformally distributed , Hence Using data Standardization/Z-Score here

Scalling

```
#Applying standarization as most of the variables are normalized distri
buted
def scale_standard(df):
    for i in cname:
        #print(i)
        df[i] = (df[i] - df[i].mean())/df[i].std()
    return df

#Standardizing Scale
train = scale_standard(train)
test = scale_standard(test)
```

Sampling Data For Train and Test¶

Stratified Sampling

```
#Using train test split functionality for creatuing sampling
X = train.iloc[:,:14]
y = train.iloc[:,14]
y=y.astype('int')

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3
3, random_state=101)

# Before smote y_train
# 0    1895
# 1     338

(X_train.shape),(y_train.shape)
```

Using SMOTE (SMOTE: Synthetic Minority Over-sampling Technique)¶

```
# from imblearn.over_sampling import SMOTE

Smo = SMOTE(random_state=101)
X_train_res, y_train_res = Smo.fit_sample(X_train,y_train)


(X_train_res.shape,y_train_res.shape)
```

Prediction function¶

```
#Predicting & Stats Function

def pred(model_object,predictors,compare):
    """1.model_object = model name
       2.predictors = data to be predicted
       3.compare = y_train"""
    predicted = model_object.predict(predictors)
    # Determine the false positive and true positive rates
    fpr, tpr, _ = roc_curve(compare, model_object.predict_proba(predict
ors)[:,1])
    cm = pd.crosstab(compare,predicted)
    TN = cm.iloc[0,0]
    FN = cm.iloc[1,0]
    TP = cm.iloc[1,1]
    FP = cm.iloc[0,1]
    print("CONFUSION MATRIX ------->> ")
    print(cm)
    print()

    ##check accuracy of model
    print('Classification paradox :------->>')
    print('Accuracy :- ', round(((TP+TN)*100)/(TP+TN+FP+FN),2))
    print()
    print('Specificity //  True Negative Rate :- ',round((TN*100)/(TN+F
P),2))
    print()
    print('Sensivity // True Positive Rate // Recall :- ',round((TP*100
)/(FN+TP),2))
    print()
    print('False Negative Rate :- ',round((FN*100)/(FN+TP),2))
    print()
    print('False Postive Rate :- ',round((FP*100)/(FP+TN),2))
    print()
    print(classification_report(compare,predicted))
    print()
    # Calculate the AUC
    print ('AUC -: %0.2f' % auc(fpr, tpr))
```

Model Level Approach

RandomForest¶

```
#Random Forest Model
rf_model = RandomForestClassifier(n_estimators=100,random_state=101).fi
t(X_train_res,y_train_res)

#Model Score on Valdation Data Set
pred(rf_model,X_test,y_test)

# Accuracy :-  94.73
# Specificity //  True Negative Rate :-  96.86
# Sensivity // True Positive Rate // Recall :-  80.69
# False Negative Rate :-  19.31
# False Postive Rate :-  3.14
# AUC -: 0.91
```

Logistic Regression

```
#logistic without binaries
logit_model = LogisticRegression(random_state=101).fit(X_train_res,y_tr
ain_res)

#Model Score on Valdation Data Set
pred(logit_model,X_test,y_test)

# Classification paradox :-------->>
# Accuracy :-  78.18
# Specificity //  True Negative Rate :-  78.74
# Sensivity // True Positive Rate // Recall :-  74.48
# False Negative Rate :-  25.52
# False Postive Rate :-  21.26
#AUC -: 0.81
```

KNN¶

```
#KNN Model Development
KNN_Model = KNeighborsClassifier(n_neighbors=5).fit(X_train_res,y_train
_res)

#Model Score on Valdation Data Set
pred(KNN_Model,X_test,y_test)

# Classification paradox :-------->>
# Accuracy :-  78.09
# Specificity //  True Negative Rate :-  79.48
# Sensivity // True Positive Rate // Recall :-  68.97
# False Negative Rate :-  31.03
# False Postive Rate :-  20.52
# AUC = 0.80
```

Navie Bayes

```
#Navie Model Development
```

```
Naive_model = GaussianNB().fit(X_train_res,y_train_res)

#Model Score on Valdation Data Set
pred(Naive_model,X_test,y_test)

# Classification paradox :-------->>
# Accuracy :-  78.64
# Specificity //  True Negative Rate :-  78.95
# Sensivity // True Positive Rate // Recall :-  76.55
# False Negative Rate :-  23.45
# False Postive Rate :-  21.05
# AUC = 0.82
```

Final Model :- Random Forest¶

As above random forest fits best for out dataset out of our tested models

Hyper Parameter Optimization with RandomSeacrhCV

below code will take time to execute so just made it commented

```
# #Creating RF Instance
# rf_model = RandomForestClassifier(random_state=101)

# pdict = {"n_estimators" : [100,200,250,300,400,500,700,800,1000],
#           "criterion": ["gini", "entropy"],
#           "max_depth": [2,4,6,8,10,None],
#           "max_features": ["auto", "sqrt", "log2",None],
#         }

# Random_CV = RandomizedSearchCV(rf_model, cv = 10,param_distributions
= pdict,n_iter = 10)
# Random_CV.fit(X_train_res, y_train_res)
# print('Best Parameters ===>  {}  '.format(Random_CV.best_params_)

# Training Final Model With Optimum Parameters
final_Model = RandomForestClassifier(random_state=101, n_estimators = 5
00,n_jobs=-1)
final_Model.fit(X_train_res,y_train_res)

#Validating Predictions
pred(final_Model,X_test,y_test)
```

Features Importance

```
#Calculating feature importances
importances = final_Model.feature_importances_

# Sort feature importances in descending order
indices = np.argsort(importances)[::1]

# Rearrange feature names so they match the sorted feature importances
names = [train.columns[i] for i in indices]

# Creating plot
```

```
fig = plt.figure(figsize=(10,10))
plt.title("Feature Importance")

# Add horizontal bars
plt.barh(range(X.shape[1]),importances[indices],align = 'center')
plt.yticks(range(X.shape[1]), names)
plt.show()
#fig.savefig('feature_importance.png')
```

AUC & ROC Curve

```
#from sklearn.metrics import roc_curve,auc,roc_auc_score
# Determine the false positive and true positive rates
fpr, tpr, _ = roc_curve(y_test, final_Model.predict_proba(X_test)[:,1])
# Calculate the AUC
roc_auc = auc(fpr, tpr)
print ('ROC AUC: %0.2f' % roc_auc)

# Plot of a ROC curve for a specific class
plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```

Final Test Data Predictions

```
# #Test Data Spliting parts target and Predictors
XX = test.iloc[:,:14].values  #predictors
yy = test.iloc[:,14].values   #target
yy=yy.astype('int')


#Predicting test data
#pred(model_object=final_Model,predictors=XX,compare=yy)

Churn_Pred = final_Model.predict(XX)
cm = pd.crosstab(yy,Churn_Pred)
TN = cm.iloc[0,0]
FN = cm.iloc[1,0]
TP = cm.iloc[1,1]
FP = cm.iloc[0,1]
print("CONFUSION MATRIX ----->> ")
print(cm)
print()
##check accuracy of model
print('Accuracy :- ', round(((TP+TN)*100)/(TP+TN+FP+FN),2))
print('False Negative Rate :- ',round((FN*100)/(FN+TP),2))
print('False Postive Rate :- ',round((FP*100)/(FP+TN),2))
```

```
print(classification_report(yy,Churn_Pred))
```

AUC & ROC over Test Data¶

```
from sklearn.metrics import roc_curve,auc,roc_auc_score
# Determine the false positive and true positive rates
fpr, tpr, _ = roc_curve(yy, final_Model.predict_proba(XX)[:,1])
# Calculate the AUC
roc_auc = auc(fpr, tpr)
print ('ROC AUC: %0.2f' % roc_auc)

# Plot of a ROC curve for a specific class
plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```

Saving the OutPut

```
#output
test_original['Churn Prediction'] = Churn_Pred
test_original['Churn Prediction'] = test_original['Churn Prediction'].m
ap({1 : 'True', 0 : 'False'})

#Predicted _Output
prob_output = pd.DataFrame(data=final_Model.predict_proba(XX),columns=(
"False_Probability","True_Probability"))
prob_output.head()

output = test_original[['state','area code','phone number','internation
al plan','voice mail plan','Churn Prediction']]

#Saving Result with Class
output.to_csv('Sample_Output.csv',sep='\t', encoding='utf-8')
#Saving with Class and Probabilities
output.join(prob_output).to_csv('Sample_Probabilistic_Churn_output.csv'
,sep='\t', encoding='utf-8')
del(output)


#Cleaning by resetting everything
#%who
#%reset -f
```

Appendix C : R Code-

```
rm(list = ls())

setwd("C://Users/parve/Documents/Data_Science_Project/R_Code/")

# #loading Libraries

x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "e1071",

    "DataCombine", "pROC", "doSNOW", "class", "readxl","ROSE")

# #install.packages if not

# #lapply(x, install.packages)

#

# #load Packages

lapply(x, require, character.only = TRUE)

rm(x)

#Input Train & Test Data Source

train_Original =
read.csv('C:/Users/parve/Documents/Data_Science_Project/Data/Train_data.csv',header =
T,na.strings = c(""," ","NA"))

test_Original =
read.csv('C:/Users/parve/Documents/Data_Science_Project/Data/Test_data.csv',header =
T,na.strings = c(""," ","NA"))

#Creating backup of orginal data

train = train_Original

test = test_Original

##############################################################################
EXPLORING DATA

##############################################################################

#viewing the data

head(train,4)

dim(train)


#structure of data or data types

str(train)
```

```
#Summary of data
summary(train)

#unique value of each count
apply(train, 2,function(x) length(table(x)))

#Replacing the dot b/w collumn name to underscore for easy to use
names(train) <- gsub('\\.','_',names(train))
names(test) <- gsub('\\.','_',names(test))

#Convertiung area code as factor
train$area_code <- as.factor(train$area_code)
test$area_code <- as.factor(test$area_code)

#Removing phone number
train$phone_number <- NULL
test$phone_number <- NULL

#Let's see the percentage of our target variable
round(prop.table(table(train$Churn))*100,2)
# False.   True.
# 85.51   14.49

#Our target Class is suffering from target imbalance
############################################################################ Checking Missing data ############################################################################
apply(train, 2, function(x) {sum(is.na(x))}) #2 for columns as in R 1 = Row & 2 = Col
apply(test, 2, function(x) {sum(is.na(x))})
#Hence no missing data found
```

```
##########################################################################
Visualizing the data
##########################################################################

#library(ggplot2)

#Target class distribution

ggplot(data = train,aes(x = Churn,fill = Churn))+
  geom_bar() +  labs(y='Churn Count', title = 'Customer Churn or Not')


# Churning of customer according to State

ggplot(train, aes(fill=Churn, x=state)) +
  geom_bar(position="dodge") + labs(title="Churning ~ State")


# Churning of customer according to Voice Mail Plan

ggplot(train, aes(fill=Churn, x=voice_mail_plan)) +
  geom_bar(position="dodge") + labs(title="Churning ~ Voice Mail Plan")


# Churning of customer according to international_plan

ggplot(train, aes(fill=Churn, x=international_plan)) +
  geom_bar(position="dodge") + labs(title="Churning ~ international_plan")


# Churning of customer according to area_code

ggplot(train, aes(fill=Churn, x=area_code)) +
  geom_bar(position="dodge") + labs(title="Churning ~ Area Code")


# Churning of customer according to area_code by international_plan

ggplot(train, aes(fill=Churn, x=area_code)) +
  geom_bar(position="dodge") + facet_wrap(~international_plan)+
  labs(title="Churning ~ Area Code by International Plan")
```

```
# Churning of customer according to area_code by voicemail_plan

ggplot(train, aes(fill=Churn, x=area_code)) +

  geom_bar(position="dodge") + facet_wrap(~voice_mail_plan)+

  labs(title="Churning ~ Area Code by Voice Mail Plan")


# Churn of international_plan by voice_mail_plan and Area Code

ggplot(train, aes(fill=Churn, x=international_plan)) +

  geom_bar(position="dodge") + facet_wrap(area_code~voice_mail_plan)+

  labs(title="Churn of international_plan by voice_mail_plan and Area Code")


# Churn ~ international_plan by voice_mail_plan

ggplot(train, aes(fill=Churn, x=international_plan)) +

  geom_bar(position="dodge") + facet_wrap(~voice_mail_plan)+

  labs(title="Churn ~ international_plan by voice_mail_plan")

################################################################################
EDA
################################################################################

#Function for Assigning factors of var to levels

cat_to_num <- function(df){

  for(i in 1:ncol(df)){

    if(class(df[,i]) == 'factor'){

      df[,i] = factor(df[,i],labels = (1:length(levels(factor(df[,i])))))

    }

  }

  return(df)

}


#Converting Categorical to level -> factors

train = cat_to_num(train)

test = cat_to_num(test)
```

```
#all numeric var

num_index = sapply(train, is.numeric)

num_data = train[,num_index]

num_col = colnames(num_data) #storing all the column name


#Checking for categorical features

cat_index = sapply(train,is.factor) #Fetching all the categorical index & later data

cat_data = train[,cat_index]

cat_col = colnames(cat_data)[-5]  #Removing target var

################################################################
Outlier Analysis
################################################################

#  #We are skipping outliers analysis becoz we already have an Class Imbalance problem.

# for (i in 1:length(num_col))

# {

#   assign(paste0("gn",i),

#        ggplot(aes_string(y = (num_col[i]), x = 'Churn'),data = train) +

#          stat_boxplot(geom = "errorbar", width = 0.5) +

#          geom_boxplot(outlier.colour="blue", fill = "skyblue",

#                       outlier.shape=18,outlier.size=1, notch=FALSE) +

#          labs(y=num_col[i],x="Churn")+

#          ggtitle(paste("Box plot of responded for",num_col[i])))

# }

#gn1

#

## Plotting plots together

#gridExtra::grid.arrange(gn1,gn2,gn3,ncol=3)

#gridExtra::grid.arrange(gn4,gn5,gn6,ncol=3)

#gridExtra::grid.arrange(gn7,gn8,gn9,ncol=3)

#gridExtra::grid.arrange(gn10,gn11,gn12,ncol=3)
```

```
# gridExtra::grid.arrange(gn13,gn14,gn15,ncol=3)

# #Removing oulier by replacing with NA and then impute

# for(i in num_col){

#   print(i)

#   outv = train[,i][train[,i] %in% boxplot.stats(train[,i])$out]

#   print(length(outv))

#   train[,i][train[,i] %in% outv] = NA

# }

#

# #checking all the missing values

# library(DMwR)

# sum(is.na(train))

# train = knnImputation(train, k=3)  #as it gives error so we going via mean or median


################################################################# Feacture Selection #################################################################

#Here we will use corrgram library to find corelation

##Correlation plot

# library(corrgram)


corrgram(train[,num_index],

        order = F,  #we don't want to reorder

        upper.panel=panel.pie,

        lower.panel=panel.shade,

        text.panel=panel.txt,

        main = 'CORRELATION PLOT')

#We can see var the highly corr related var in plot marked dark blue.

#Dark blue color means highly positive cor related
```

```
##---------Chi Square Analysis---------------------------------


for(i in cat_col){
  print(names(cat_data[i]))
  print((chisq.test(table(cat_data$Churn,cat_data[,i])))[3])  #printing only pvalue
}
##-----------------Removing Highly Corelated and Independent var----------------------
train = subset(train,select= -c(state,total_day_charge,total_eve_charge,
                                total_night_charge,total_intl_charge))


test = subset(test,select= -c(state,total_day_charge,total_eve_charge,
                              total_night_charge,total_intl_charge))


#################################################################
Feacture Scaling
#################################################################
#all numeric var
num_index = sapply(train, is.numeric)
num_data = train[,num_index]
num_col = colnames(num_data) #storing all the column name


#Checking Data of Continuous Variable
################  Histogram   ##################
hist(train$total_day_calls)
hist(train$total_day_minutes)
hist(train$account_length)


#Most of the data is uniformally distributed
#Using data Standardization/Z-Score here
```

```
for(i in num_col){
  print(i)
  train[,i] = (train[,i] - mean(train[,i]))/sd(train[,i])
  test[,i] = (test[,i] - mean(test[,i]))/sd(test[,i])
}
#################################################################
Sampling of Data
################################################################
# #Divide data into train and test using stratified sampling method
# library(caret)
set.seed(101)
split_index = createDataPartition(train$Churn, p = 0.66, list = FALSE)
trainset = train[split_index,]
validation_set  = train[-split_index,]

#Checking Train Set Target Class
table(trainset$Churn)
# 1    2
# 1881  319

# #Our class is Imbalanced
# Synthetic Over Sampling the minority class & Under Sampling Majority Class to have a good Training Set

# #library(ROSE)  #---> Lib for Over and Under Sampling
trainset <- ROSE(Churn~.,data = trainset,p = 0.5,seed = 101)$data
table(trainset$Churn) # 1 = 1101  2 = 1099
#Removing All the custom variable from memory
# library(DataCombine)
rmExcept(c("test_Original","train_Original","train","test","trainset","validation_set"))
```

```
##############################################################################
#  #             Basic approach towards ML - Models
# # Let's just get a basic idea of how models perform on our already preprocesed data
# # Later on we will select the best model and will make it more efficient for our Dataset
###############################################################################
# #function for calculating the FNR,FPR,Accuracy
calc <- function(cm){
  TN = cm[1,1]
  FP = cm[1,2]
  FN = cm[2,1]
  TP = cm[2,2]
  # #calculations
  print(paste0('Accuracy :- ',((TN+TP)/(TN+TP+FN+FP))*100))
  print(paste0('FNR :- ',((FN)/(TP+FN))*100))
  print(paste0('FPR :- ',((FP)/(TN+FP))*100))
  print(paste0('FPR :- ',((FP)/(TN+FP))*100))
  print(paste0('precision :-  ',((TP)/(TP+FP))*100))
  print(paste0('recall//TPR :-  ',((TP)/(TP+FP))*100))
  print(paste0('Sensitivity :-  ',((TP)/(TP+FN))*100))
  print(paste0('Specificity :-  ',((TN)/(TN+FP))*100))
  plot(cm)
}

### ##----------------------- Random Forest ----------------------- ## ###
# library(randomForest)

set.seed(101)
RF_model = randomForest(Churn ~ ., trainset,ntree= 500,importance=T,type='class')
plot(RF_model)
```

```
#Predict test data using random forest model
RF_Predictions = predict(RF_model, validation_set[,-15])

##Evaluate the performance of classification model
cm_RF = table(validation_set$Churn,RF_Predictions)
confusionMatrix(cm_RF)
calc(cm_RF)
plot(RF_model)
# Result on validaton set
# [1] "Accuracy :- 86.2312444836717"
# [1] "FNR :- 17.6829268292683"
# [1] "FPR :- 13.1062951496388"
# [1] "FPR :- 13.1062951496388"
# [1] "precision :-  51.5267175572519"
# [1] "recall//TPR :-  51.5267175572519"
# [1] "Sensitivity :-  82.3170731707317"
# [1] "Specificity :-  86.8937048503612"

### ##----------------------- LOGISTIC REGRESSION ----------------------- ## ###
set.seed(101)
logit_model = glm(Churn ~., data = trainset, family =binomial(link="logit"))
summary(logit_model)
#Prediction
logit_pred = predict(logit_model,newdata = validation_set[,-15],type = 'response')

#Converting Prob to number or class
logit_pred = ifelse(logit_pred > 0.5, 2,1)
#logit_pred = as.factor(logit_pred)
##Evaluate the performance of classification model
```

```
cm_logit = table(validation_set$Churn, logit_pred)

confusionMatrix(cm_logit)

calc(cm_logit)

plot(logit_model)

#roc(validation_set$Churn~logit_pred)


# Result on validaton set

# [1] "Accuracy :- 78.1994704324801"

# [1] "FNR :- 28.6585365853659"

# [1] "FPR :- 20.6398348813209"

# [1] "FPR :- 20.6398348813209"

# [1] "precision :-  36.9085173501577"

# [1] "recall//TPR :-  36.9085173501577"

# [1] "Sensitivity :-  71.3414634146341"

# [1] "Specificity :-  79.360165118679"

# ROC = 0.8017


### ##----------------------- KNN ----------------------- ## ###

set.seed(101)

## KNN impletation

# library(class)


##Predicting Test data

#knn_Pred = knn(train = trainset[,1:14],test = validation_set[,1:14],cl = trainset$Churn, k = 5)


knn_Pred = knn(train = trainset[,1:14],test = validation_set[,1:14],cl = trainset$Churn, k =
5,prob = T)

#Confusion matrix

cm_knn = table(validation_set$Churn,knn_Pred)

confusionMatrix(cm_knn)
```

```
calc(cm_knn)

# Result on validaton set
# [1] "Accuracy :- 78.8172992056487"
# [1] "FNR :- 46.3414634146341"
# [1] "FPR :- 16.9246646026832"
# [1] "FPR :- 16.9246646026832"
# [1] "precision :-  34.9206349206349"
# [1] "recall//TPR :-  34.9206349206349"
# [1] "Sensitivity :-  53.6585365853659"
# [1] "Specificity :-  83.0753353973168"

### ##----------------------- Naive Bayes ----------------------- ## ###

# library(e1071) #lib for Naive bayes
set.seed(101)
#Model Development and Training
naive_model = naiveBayes(Churn ~., data = trainset, type = 'class')
#prediction
naive_pred = predict(naive_model,validation_set[,1:14])

#Confusion matrix
cm_naive = table(validation_set[,15],naive_pred)
confusionMatrix(cm_naive)
calc(cm_naive)

# Result on validaton set
# [1] "Accuracy :- 83.1421006178288"
# [1] "FNR :- 29.2682926829268"
```

```
# [1] "FPR :- 14.7574819401445"

# [1] "FPR :- 14.7574819401445"

# [1] "precision :-  44.7876447876448"

# [1] "recall//TPR :-  44.7876447876448"

# [1] "Sensitivity :-  70.7317073170732"

# [1] "Specificity :-  85.2425180598555"


##############################################################################
##########################


# # As according to out problem statement we need to find out the customer which will
Move or Not.


# # " Reduction customer churn is important because cost of acquiring a new customer is
higher then retaining the older one."


## From above statement it's clear that Cost matters alot.

## We are using default threshold cutoff here for Churning and Not Churn


# # So according to requirement we are finalizing RandomForest as our Final model as under
train data set

# # Random forest model out performs the all other model in FNR,FPR and Accuracy.


######################################################################

# #           Knowing the right hyper parameters tuning

# # As this process will take a bit time so here i have commented the code

######################################################################


#Using doSNOW lib for segmenting the clustering onto task as a faster approch
```

```
# library(doSNOW)


# # #Best mtry  ======   found best as = 4
# cl <- makeCluster(6) #clustering approach using doSNOW pkg
# registerDoSNOW(cl)
#
# trControl <- trainControl(method = "cv",number = 10,search = "grid")
# set.seed(101)
# tuneGrid <- expand.grid(.mtry = c(2:8))
# rf_mtry <- train(Churn~.,data = trainset,method = "rf",metric = "Accuracy",
#               tuneGrid = tuneGrid,trControl = trControl,importance = TRUE,ntree = 800)
# best_mtry <- rf_mtry$bestTune$mtry
# print(best_mtry)


# # #Looking for best ntree  ====  found best as = 800
# store_maxtrees <- list()
# tuneGrid <- expand.grid(.mtry = best_mtry)
# for (ntree in c(200, 300, 350, 400, 450, 500, 550, 600, 700,800, 1000)) {
#   set.seed(101)
#   rf_maxtrees <- train(Churn~.,data = df_trainset,method = "rf",metric =
"Accuracy",tuneGrid = tuneGrid,
#                    trControl = trControl,importance = TRUE,ntree = ntree)
#   key <- toString(ntree)
#   store_maxtrees[[key]] <- rf_maxtrees
# }
# results_tree <- resamples(store_maxtrees)
# summary(results_tree)
#
# stopCluster(cl)
```

```
rmExcept(c("train_Original","test_Original","train","test","trainset","validation_set","calc"))


##############################################################################################

# #		Final Random Forest Model with tuning parameters

##############################################################################################


set.seed(101)

final_model = randomForest(Churn~.,data =
trainset,ntree=800,mtry=4,importance=TRUE,type = 'class')

final_validation_pred = predict(final_model,validation_set[,-15])

cm_final_valid = table(validation_set[,15],final_validation_pred)

confusionMatrix(cm_final_valid)

calc(cm_final_valid)

#Result on validation set after parameter tuning

# [1] "Accuracy :- 86.4960282436011"

# [1] "FNR :- 17.0731707317073"

# [1] "FPR :- 12.8998968008256"

# [1] "FPR :- 12.8998968008256"

# [1] "precision :-  52.1072796934866"

# [1] "recall//TPR :-  52.1072796934866"

# [1] "Sensitivity :-  82.9268292682927"

# [1] "Specificity :-  87.1001031991744"



#Variable Importance

importance(final_model) #builting function in Random forest lib

varImpPlot(final_model) #builtin func
```

```
#Plotting ROC curve and Calculate AUC metric

# library(pROC)

PredictionwithProb <-predict(final_model,validation_set[,-15],type = 'prob')

auc <- auc(validation_set$Churn,PredictionwithProb[,2])

auc

# # AUC = 89.47

plot(roc(validation_set$Churn,PredictionwithProb[,2]))


###########################################################################
########


# #      Final Prediction On test Data set


###########################################################################
########


rmExcept(c("final_model","train","test","train_Original","test_Original","calc"))


set.seed(101)

final_test_pred = predict(final_model,test[,-15])

cm_final_test = table(test[,15],final_test_pred)

confusionMatrix(cm_final_test)

calc(cm_final_test)


# #Final Test Prediction

# [1] "Accuracy :- 85.9028194361128"

# [1] "FNR :- 15.625"

# [1] "FPR :- 13.8600138600139"

# [1] "FPR :- 13.8600138600139"
```

```
# [1] "precision :-  48.586118251928"

# [1] "recall//TPR :-  48.586118251928"

# [1] "Sensitivity :-  84.375"

# [1] "Specificity :-  86.1399861399861"


#Plotting ROC curve and Calculate AUC metric

# library(pROC)

finalPredictionwithProb <-predict(final_model,test[,-15],type = 'prob')

auc <- auc(test$Churn,finalPredictionwithProb[,2])

auc

# # AUC = 91.74

plot(roc(test$Churn,finalPredictionwithProb[,2]))


###############################################################################################################################################

#  #        Saving output to file ( For re run uncomment the code (ctrl+shift+c))

###############################################################################################################################################


test_Original$predicted_output <- final_test_pred

test_Original$predicted_output <- gsub(1,"False",test_Original$predicted_output)

test_Original$predicted_output <- gsub(2,"True",test_Original$predicted_output)


#Entire Comparison

write.csv(test_Original,'./output/Final_Full_Data_Output.csv',row.names = F)


#Phonenumber and Churning class and probab

submit <- data.frame(test_Original$state,

                test_Original$area.code,

                test_Original$international.plan,
```

```
                test_Original$voice.mail.plan,
                test_Original$phone.number,
                test_Original$predicted_output,
                finalPredictionwithProb[,1],
                finalPredictionwithProb[,2])

colnames(submit) <- c("State","Area Code","International Plan","Voice Mail Plan","Phone_Number",Predicted_Output","Probability_of_False","Probability_of_True")

write.csv(submit,file = './output/Final_Churn_Class_Probab.csv',row.names = F)

rm(list = ls())
```