Customer Churn Reduction-Project Report

Navaneeth Kumar O M
18 February 2019

Contents

Chapter 1

Introduction

1.1 Problem Description

Churn (loss of customers to competition) is a problem for companies because it is more expensive to acquire a new customer than to keep your existing one from leaving. This problem statement is targeted at enabling churn reduction using analytics concepts.

1.2 Problem statement

The objective of this case is to predict customer behaviour. We are provided with a public dataset that has customer usage pattern and if the customer has moved or not. We must develop an algorithm to predict the churn score based on usage pattern.
Target Variable : Churn: if the customer has moved (1=False; 2 = True)

1.3 Problem statement

Our task is to build classification models which will classify whether a given customer will move out(churn) or not depending on the multiple factors given in the data. Given below is the set of predictor variables given to classify the customer churn with some sample observations.

Table 1.1: List of Predictor Variables

Serial no	Independent/Predictor variables
1	state
2	account length
3	area code
4	phone number
5	international plan
6	voice mail plan
7	number vmail messages
8	total day minutes
9	total day calls
10	total day charge
11	total eve minutes
12	total eve calls
13	total eve charge
14	total night minutes
15	total night calls
16	total night charge
17	total intl minutes
18	total intl calls
19	total intl charge
20	number customer service calls

Table 1.2: Churn Reduction sample data (Columns 1-7)

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

Table 1.3: Churn Reduction sample data (Columns 8-14)

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

Table 1.4: Churn Reduction sample data (Columns 15-21)

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.

Chapter 2

Methodology

2.1 Pre-Processing

Any predictive modelling requires that we look at the data before we start modelling. However, in data mining terms looking at data refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as **Exploratory Data Analysis**. To start this process, we will first try and look at all the probability distributions of the variables. Most analysis like regression, require the data to be normally distributed. We can visualize that in a glance by looking at the probability distributions or probability density functions of the variable.

A Density Plot or kernel density plot visualises the distribution of data over a continuous interval or time period. This chart is a variation of a Histogram that uses kernel smoothing to plot values, allowing for smoother distributions by smoothing out the noise. The peaks of a density plot help display where values are concentrated over the interval.

An advantage that density plots have over Histograms is that they're better at determining the shape because they're not affected by the number of bins used (each bar used in a typical histogram). A Histogram comprising of only 4 bins wouldn't produce a distinguishable enough shape of distribution as a 20-bin Histogram would. However, with density plots, this isn't an issue.

In Figure 2.1 we have plotted the probability density functions of all the predictor variables which shows density of distribution of the respective variable. We see that most of the variables are similar to the normal distribution curve, and certain variables are skewed due to the presence of outliers which we will address in the coming sections.

The structure of the churn reduction is as shown in Figure-2.2, The given data as a whole has 5000 observations with 20 independent/predictor variables, and 1 target/dependent variable, the train data consists of 3333 observations and test data consists 1667 observations.

As we see that the independent/predictor variable area.code has 3 unique values, it can be converted into a categorical variable by assigning levels like 1,2,3…etc for each unique variable, and **we are binning the number.customer.service.calls variable into 'Low', 'Moderate' & 'High' as this seems that a customer has made calls to the customer service which would indicate there would be some underlying service issues from the service provider end**. This changes are made for the sake of model simplification and to save some memory. The target class Churn is imbalanced as it has more of false classes than the true classes and we will address this issue in the modelling section.

Figure 2.1: Probability Density Functions of the predictor variables

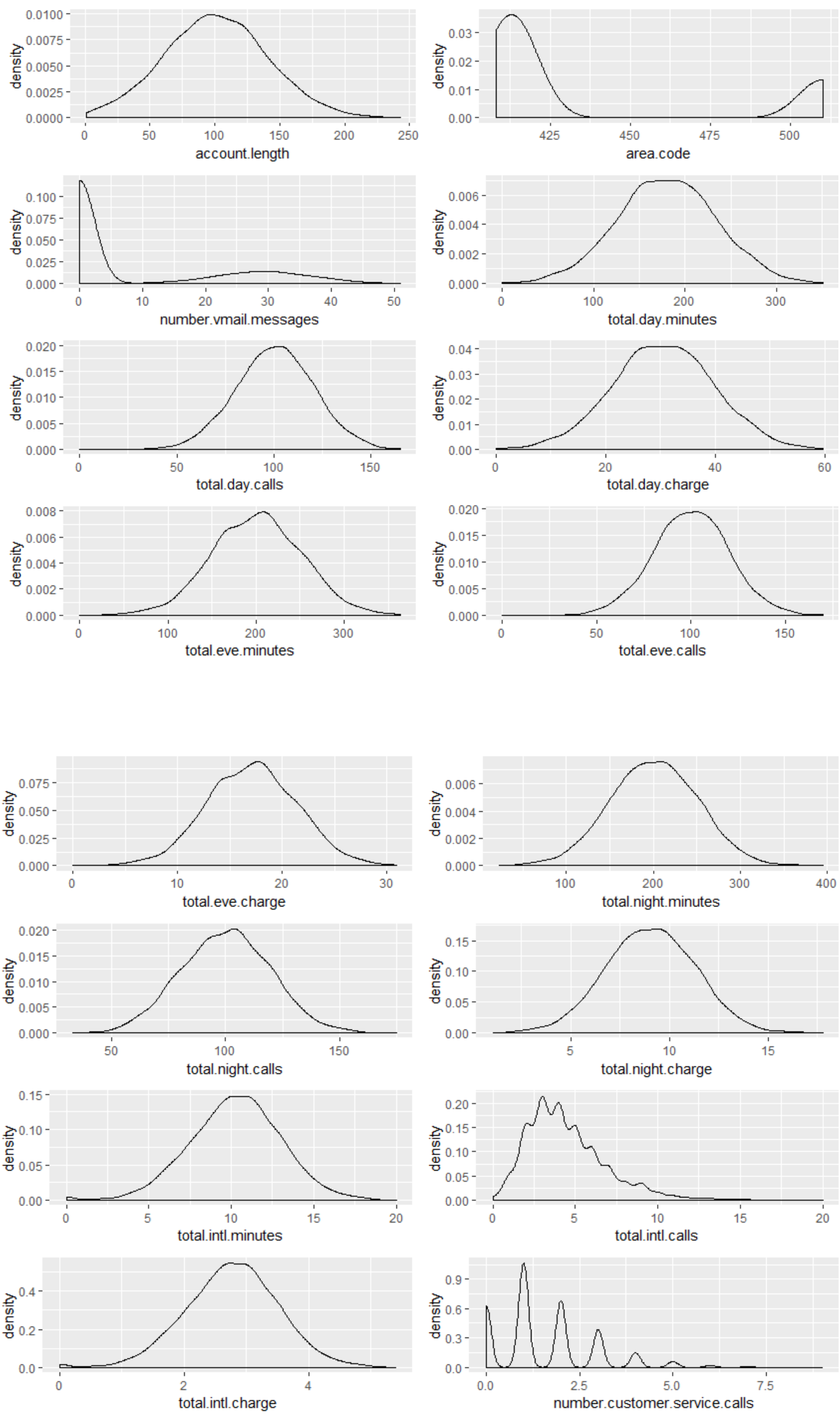
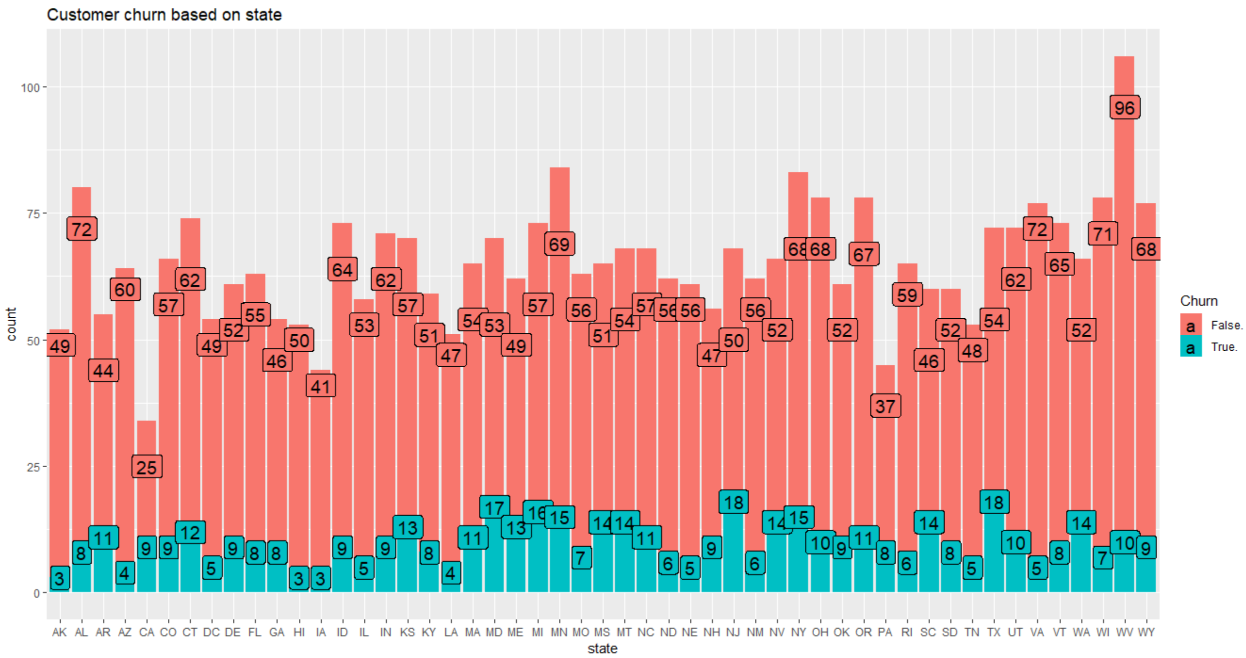


Figure 2.2: Structure overview of the churn reduction dataset.

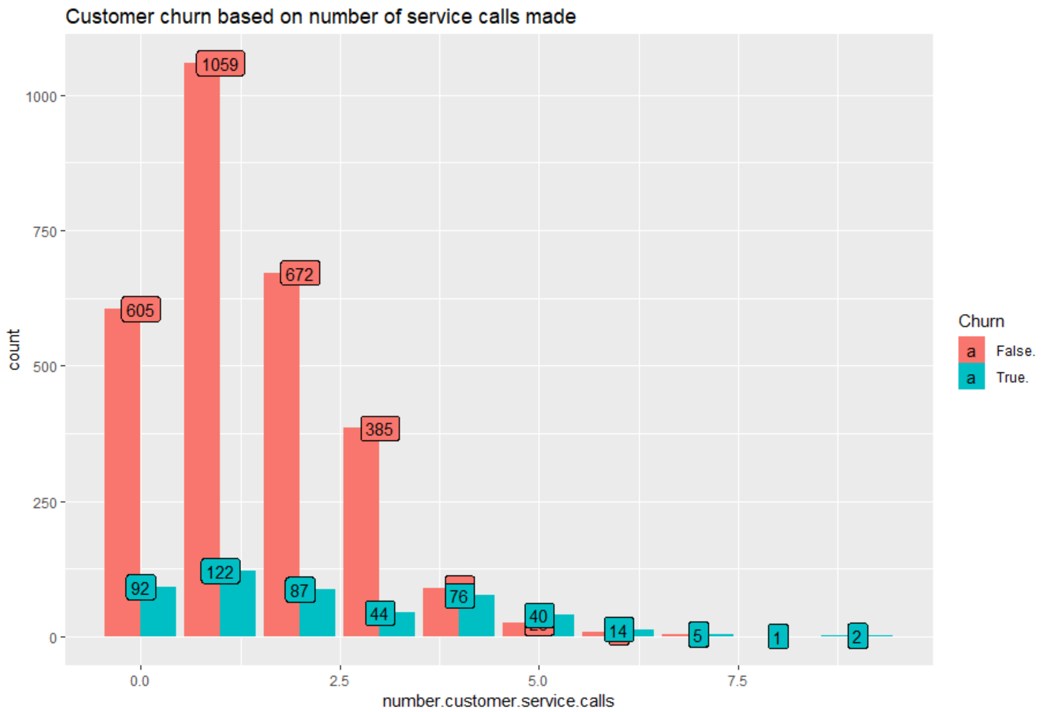
```
> str(whole_data)
'data.frame':	5000 obs. of  21 variables:
 $ state                        : Factor w/ 51 levels "AK","AL","AR",..: 17 36 32 36 37 2 20 25 19 50 ...
 $ account.length               : int  128 107 137 84 75 118 121 147 117 141 ...
 $ area.code                    : int  415 415 415 408 415 510 510 415 408 415 ...
 $ phone.number                 : Factor w/ 5000 levels " 327-1058"," 327-1319",..: 1927 1576 1118 1708 111
$ 1048 81 292 118 ...
 $ international.plan           : Factor w/ 2 levels " no"," yes": 1 1 1 2 2 2 1 2 1 2 ...
 $ voice.mail.plan              : Factor w/ 2 levels " no"," yes": 2 2 1 1 1 1 2 1 1 2 ...
 $ number.vmail.messages        : int  25 26 0 0 0 0 24 0 0 37 ...
 $ total.day.minutes            : num  265 162 243 299 167 ...
 $ total.day.calls              : int  110 123 114 71 113 98 88 79 97 84 ...
 $ total.day.charge             : num  45.1 27.5 41.4 50.9 28.3 ...
 $ total.eve.minutes            : num  197.4 195.5 121.2 61.9 148.3 ...
 $ total.eve.calls              : int  99 103 110 88 122 101 108 94 80 111 ...
 $ total.eve.charge             : num  16.78 16.62 10.3 5.26 12.61 ...
 $ total.night.minutes          : num  245 254 163 197 187 ...
 $ total.night.calls            : int  91 103 104 89 121 118 118 96 90 97 ...
 $ total.night.charge           : num  11.01 11.45 7.32 8.86 8.41 ...
 $ total.intl.minutes           : num  10 13.7 12.2 6.6 10.1 6.3 7.5 7.1 8.7 11.2 ...
 $ total.intl.calls             : int  3 3 5 7 3 6 7 6 4 5 ...
 $ total.intl.charge            : num  2.7 3.7 3.29 1.78 2.73 1.7 2.03 1.92 2.35 3.02 ...
 $ number.customer.service.calls: int  1 1 0 2 3 0 3 0 1 0 ...
 $ Churn                        : Factor w/ 2 levels " False."," True.": 1 1 1 1 1 1 1 1 1 1 ...
```

Below are the visualizations performed on some of the predictors variables v/s Churn :

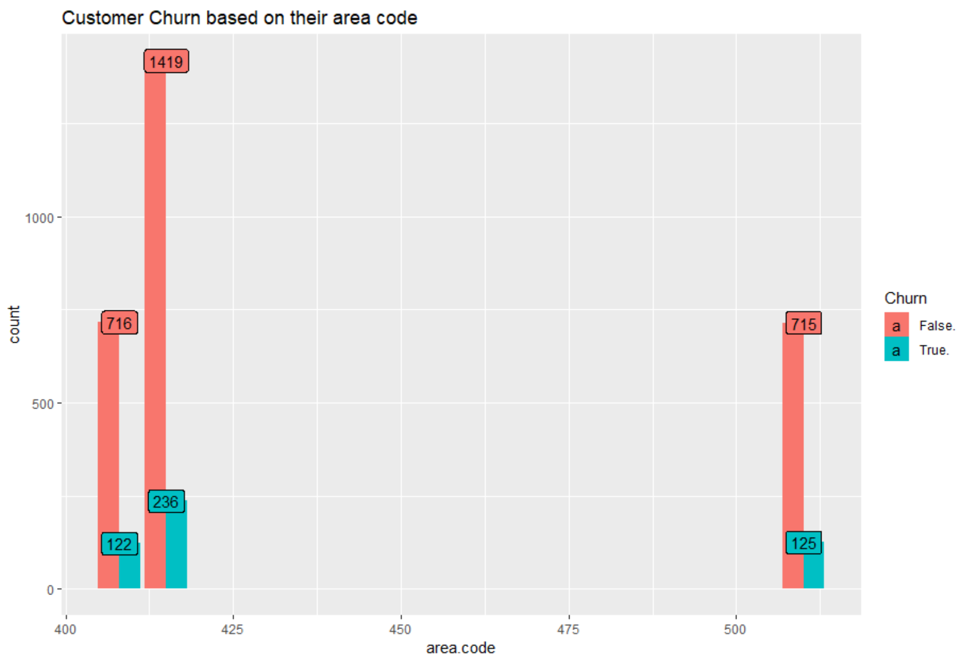
- **Customers churn based on state :** We see that most of the customers who have churned out are from MD, NJ and TX states.



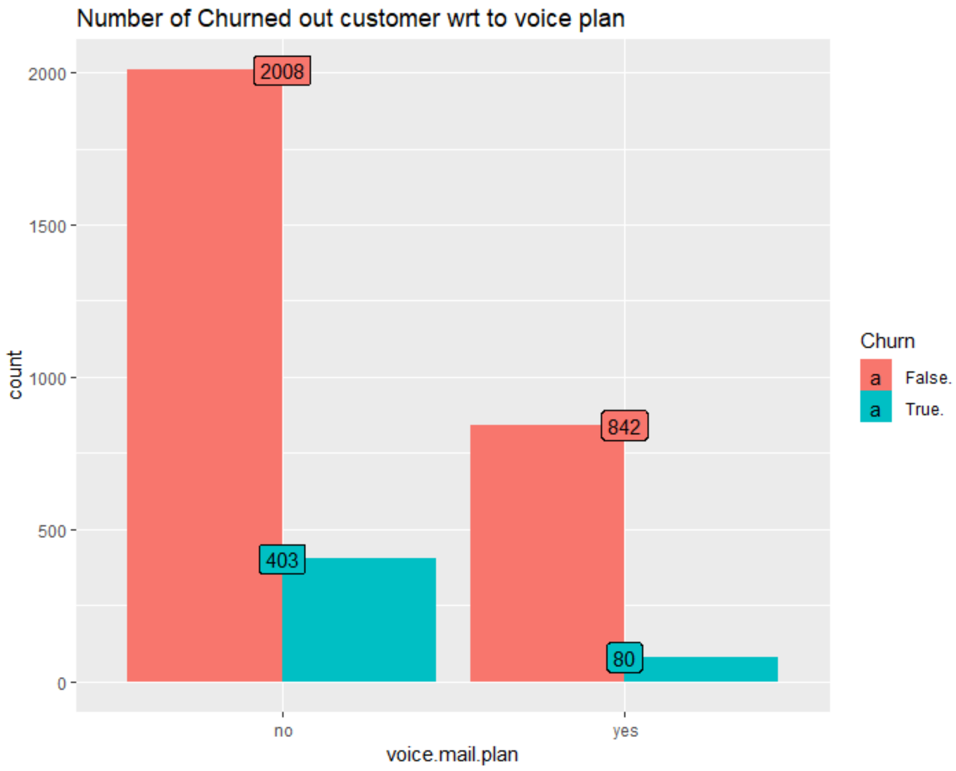
- **Churn based on number of customer service calls made :** We observe that customers who have made more number of customer service calls in the range 4-6 calls have churned out more, this indicates that would exist some technical issues from the service provider end which has resulted for the customer to churn out by making him unhappy with the service being provided.



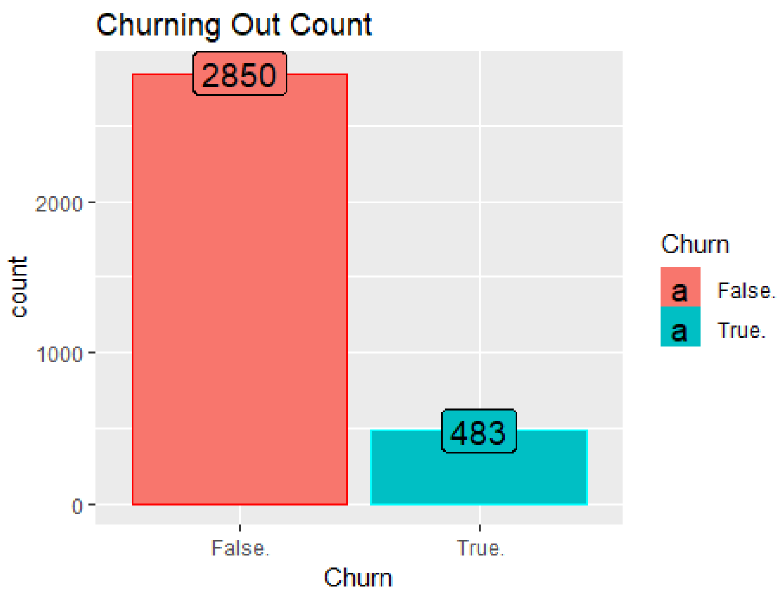
- **Churn based on area code :** We observe that most of the customers who have churned out are from the area code 415.



- **Churn based on voice plan :** We observe that most of the customers who have churned out did not subscribe to any voice mail plan.



- **Churn Count:** We see that in general the count of customers being churned out is more in the train dataset been given.



2.1.1 Missing Value Analysis

Missing values are which, where the values are missing in an observation in the dataset. It can occur due to human errors, individuals refusing to answer while surveying, optional box in questionnaire.

Missing data mechanism is divided into 3 categories as below :

- **Missing Completely at Random (MCAR)**, means there is no relationship between the missingness of the data and any values, observed or missing. Those missing data points are a random subset of the data. There is *nothing* systematic going on that makes some data more likely to be missing than others.

- **Missing at Random (MAR)**, means there is a systematic relationship between the propensity of missing values and the observed data, but not the missing data. Whether an observation is missing has nothing to do with the missing values, but it does have to do with the values of an individual's observed variables. So, for example, if men are more likely to tell you their weight than women, weight is MAR.

- **Missing Not at Random** (**MNAR)**, means there is a relationship between the propensity of a value to be missing and its values. This is a case where the people with the lowest education are missing on education or the sickest people are most likely to drop out of the study. MNAR is called "non-ignorable" because the missing data mechanism itself must be modelled as we deal with the missing data.

Usually we only consider those variables for missing value imputation whose missing values is less than 30%, if it above this we will drop that variable in our analysis as imputing missing values which are more than 30% doesn't make any sense and the information would also be insensible to consider.

From the exploratory data analysis, we see that our churn reduction dataset doesn't have any missing values and hence we are not proceeding with this step.

The below figure 2.3 shows the variables with percentage of missing values in them.

Figure 2.3: Percentage of missing values present in the dataset

	Columns	Missing_percentage
1	state	0
2	account.length	0
3	area.code	0
4	phone.number	0
5	international.plan	0
6	voice.mail.plan	0
7	number.vmail.messages	0
8	total.day.minutes	0
9	total.day.calls	0
10	total.day.charge	0
11	total.eve.minutes	0
12	total.eve.calls	0
13	total.eve.charge	0
14	total.night.minutes	0
15	total.night.calls	0
16	total.night.charge	0
17	total.intl.minutes	0
18	total.intl.calls	0
19	total.intl.charge	0
20	number.customer.service.calls	0
21	Churn	0

Showing 1 to 21 of 21 entries

2.1.2 Outlier Analysis

An Outlier is an observation which is inconsistent(or distant) with rest of the observations. The presence of outliers in the data adds to the skewness and this needs to be addressed. We have various methods to detect the outliers but for this dataset we are going to detect the outliers using Tukey's Boxplot method.

Tukey's Boxplot method or simply called boxplot method is a standardized way of displaying the distribution of the data based on the five-number summary, they are minimum, first quartile, median, third quartile and maximum. A segment inside the rectangle shows the median and "whiskers" above and below the box show the locations of the minimum and maximum.

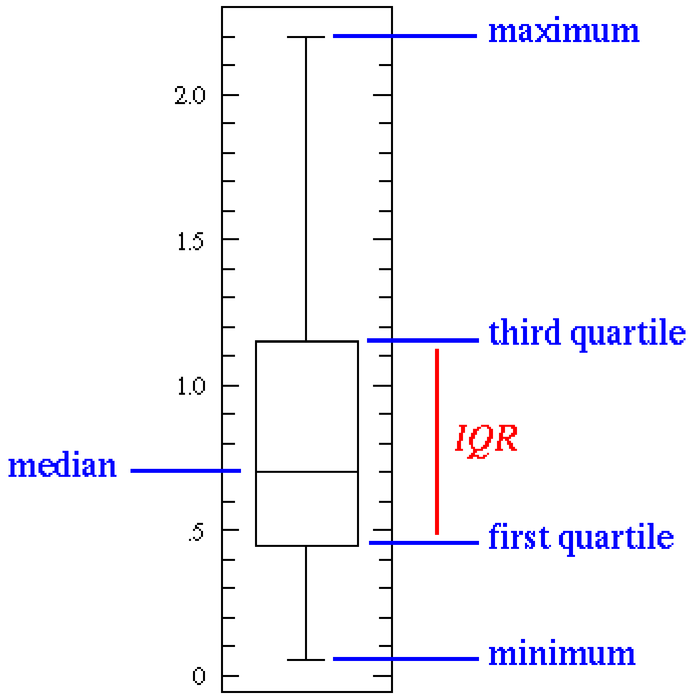


Figure 2.4 : Structure of a typical boxplot

The above Figure 2.4 shows the typical box plot. The interquartile range(IQR) is an area where the bulk of the majority values lie in other words it is the difference between the third quartile(Q3) and first quartile(Q1). The maximum value of a box plot is 1.5 times the IQR beyond the third quartile, minimum value of a box plot is 1.5 times the IQR below the first quartile. Mathematically it is given as below :

Minimum value = Q3 + (1.5*IQR)
Maximum value = Q1 – (1.5*IQR)

The values which fall beyond the minimum and maximum values are considered to be as outliers. Hence, further we replace these outlier values with NA's and impute the same with KNN Imputation which estimates the NAs considering the mean K amount of surrounding values.

As the percentage of Outliers are very less as shown in the Figure 2.5 , we are following the above method else the other way of imputing the outliers would be replacing the outliers falling beyond the minimum value with the minimum value and replacing the outliers falling beyond the maximum with the maximum value (doing so makes the data less vulnerable to the insensible information rather imputing with NAs)

	Columns	Outlier_Percentage
1	total.intl.calls	2.36
2	total.intl.minutes	1.44
3	total.intl.charge	1.44
4	number.vmail.messages	1.20
5	total.night.calls	0.86
6	total.eve.minutes	0.84
7	total.eve.charge	0.84
8	total.night.minutes	0.78
9	total.night.charge	0.78
10	total.day.calls	0.70
11	total.day.minutes	0.68
12	total.day.charge	0.68
13	total.eve.calls	0.54
14	account.length	0.48
15	state	0.00
16	area.code	0.00
17	phone.number	0.00
18	international.plan	0.00
19	voice.mail.plan	0.00
20	number.customer.service.calls	0.00
21	Churn	0.00

Figure 2.5 : Tabulation of Outlier percentage

The value of K for KNN Imputation is selected as 9 because the value of 9 gave the least amount of standard deviation for a given variable when checked with different values of K, and standard deviation of the variables for K>9 seemed to be saturated with no much difference.

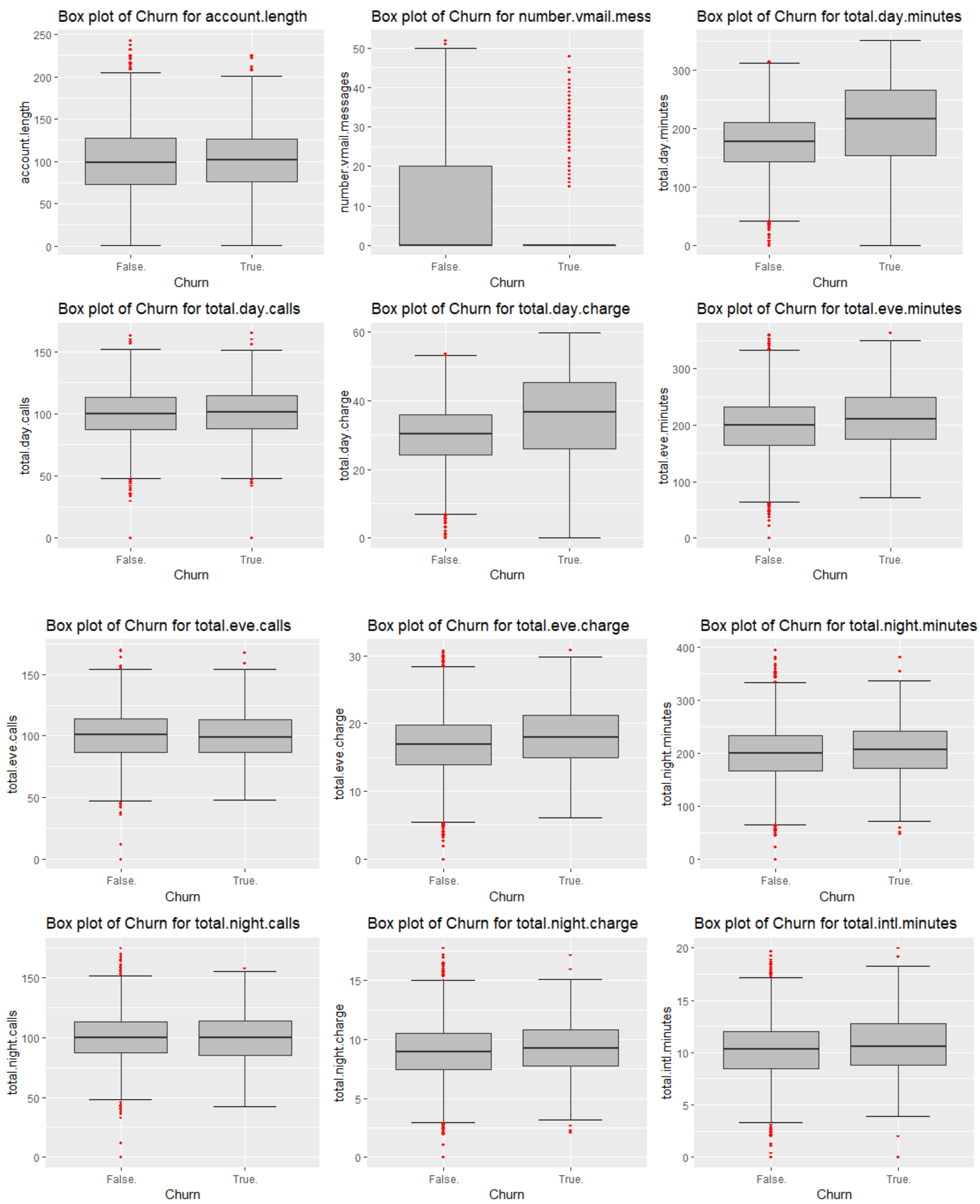
The below Figure-2.6 shows the list of standard deviation values noted for each different iterations of K from the given dataset, the Original SD columns shows the standard deviation of the continuous variables before imputing the outliers with NAs. The other columns show the standard deviation of the variables for different K value imputation.

Figure 2.6 : Tabulation of standard deviation for different values of K

	Original SD	SD_for_K3.	SD_for_K5.	SD_for_K7.	SD_for_K9.
account.length	39.6945595	38.8378069	38.8265127	38.8172174	38.8140167
number.vmail.messages	13.5463934	12.9457764	12.9157912	12.9109930	12.9063600
total.day.minutes	53.8946992	52.3369891	52.2959352	52.2822597	52.2836775
total.day.calls	19.8311974	19.1692018	19.1600225	19.1572317	19.1543170
total.day.charge	9.1620687	8.8972282	8.8902492	8.8879245	8.8881656
total.eve.minutes	50.5513090	48.6832487	48.6614703	48.6483199	48.6433079
total.eve.calls	19.8264958	19.3062420	19.3059591	19.3037563	19.3026089
total.eve.charge	4.2968433	4.1380599	4.1362078	4.1350904	4.1346644
total.night.minutes	50.5277893	48.6873713	48.6745329	48.6672053	48.6579532
total.night.calls	19.9586859	19.1700309	19.1505845	19.1464836	19.1449685
total.night.charge	2.2737627	2.1909360	2.1903584	2.1900280	2.1896112
total.intl.minutes	2.7613957	2.5539659	2.5517836	2.5510719	2.5505463
total.intl.calls	2.4567882	2.0627010	2.0595883	2.0585922	2.0579081
total.intl.charge	0.7455137	0.6894981	0.6889097	0.6887175	0.6885756

In figure 2.7 we have plotted the boxplots for the 13 continuous variables. The red dots indicate the outliers which are the extreme values after minimum/maximum points and the figure 2.8 shows the boxplot after the outlier treatment with KNN imputation, we could see that the outliers have been reduced very much and our data is now free from the outliers.

Figure 2.7 : Boxplot of continuous variables with outliers



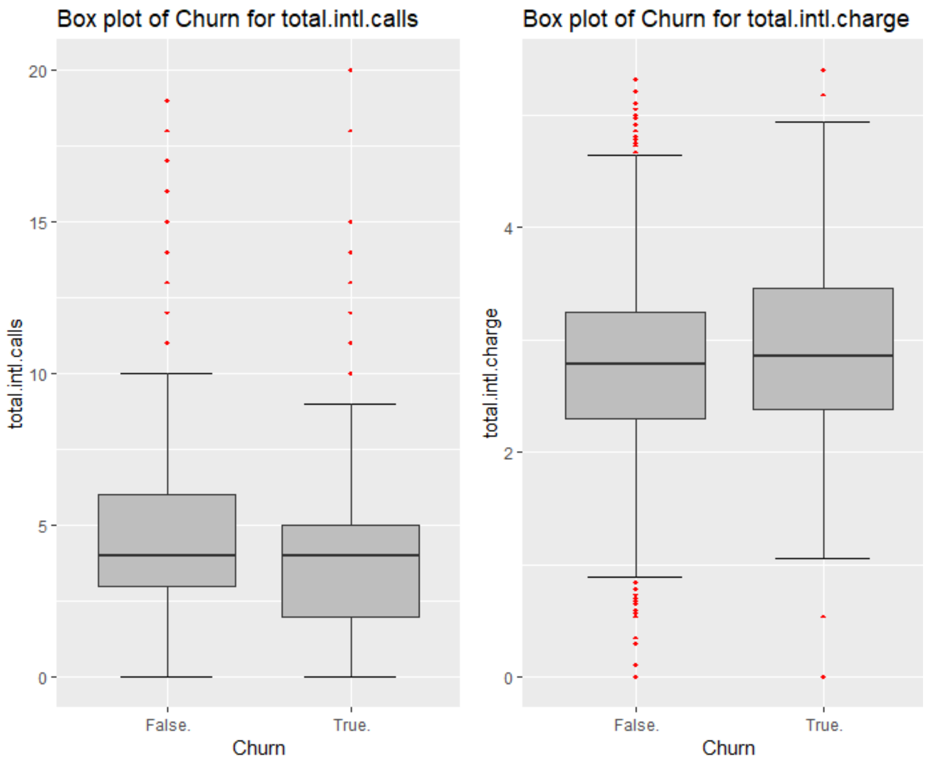
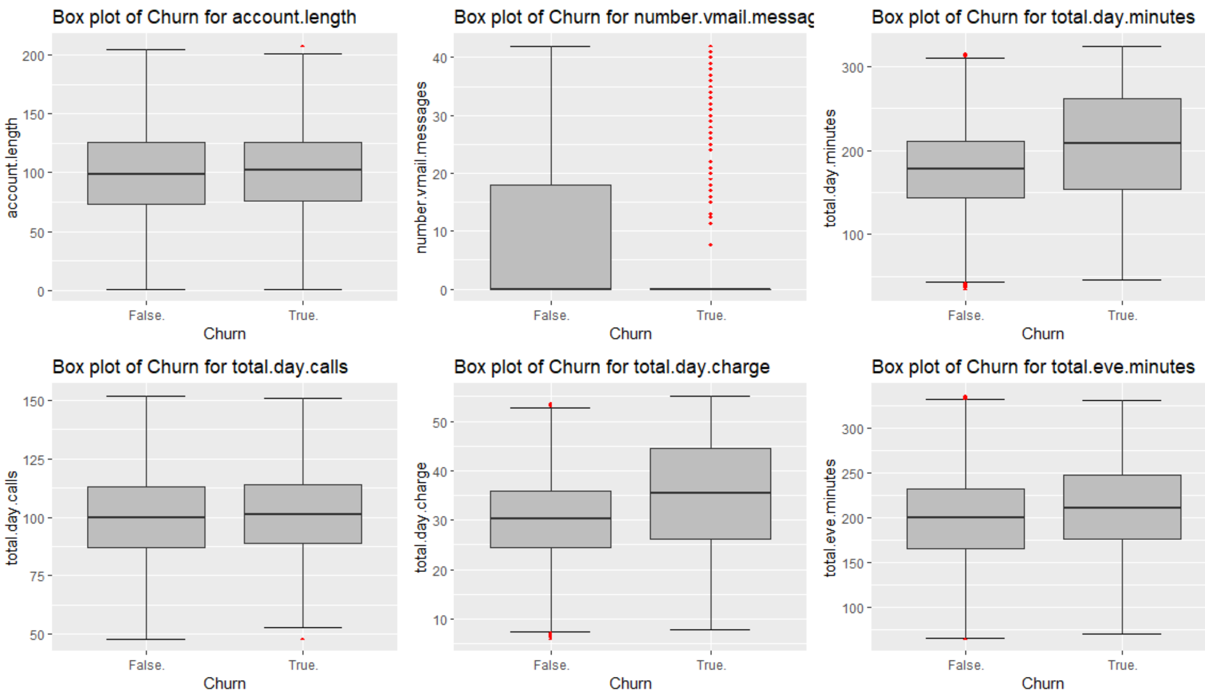
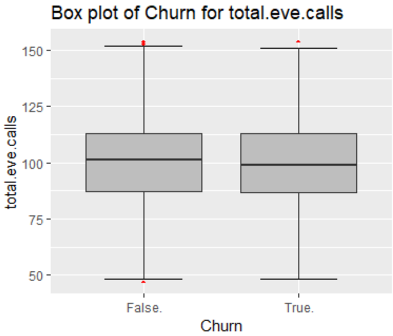
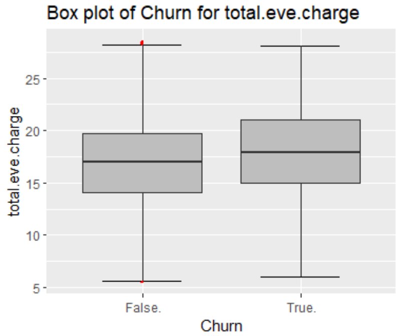
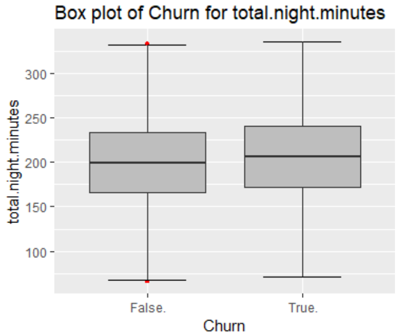


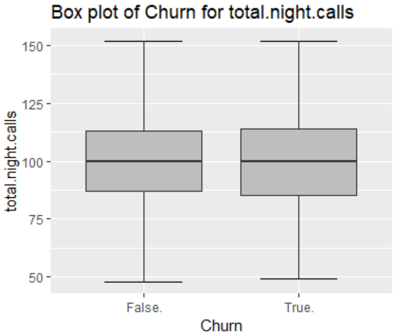
Figure 2.8 : Boxplot of continuous variables without outliers

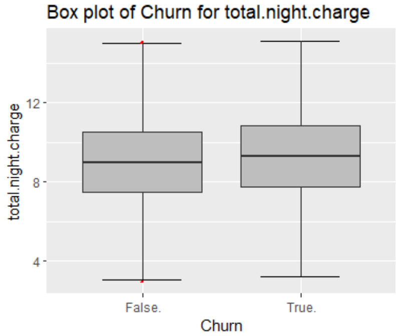


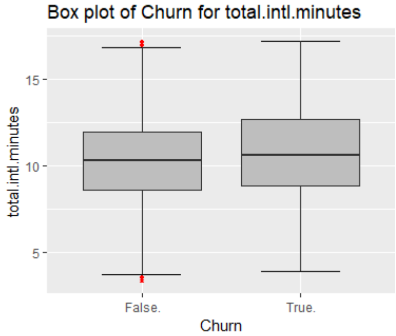

Box plot of Churn for total.eve.calls
total.eve.calls
150
125
100
75
50
False.
True.
Churn

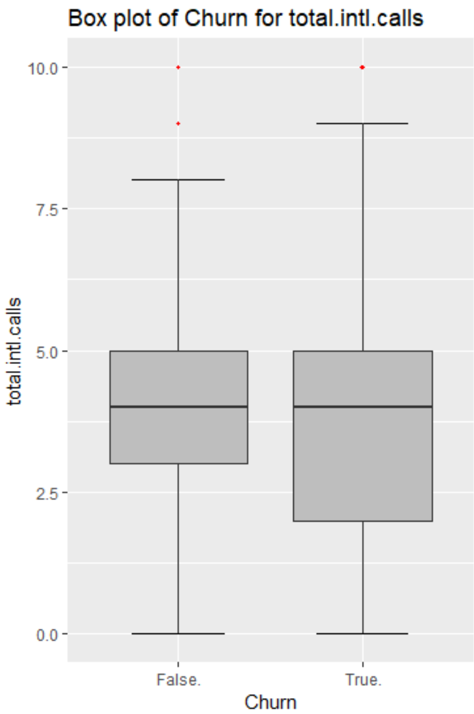

Box plot of Churn for total.eve.charge
total.eve.charge
25
20
15
10
5
False.
True.
Churn

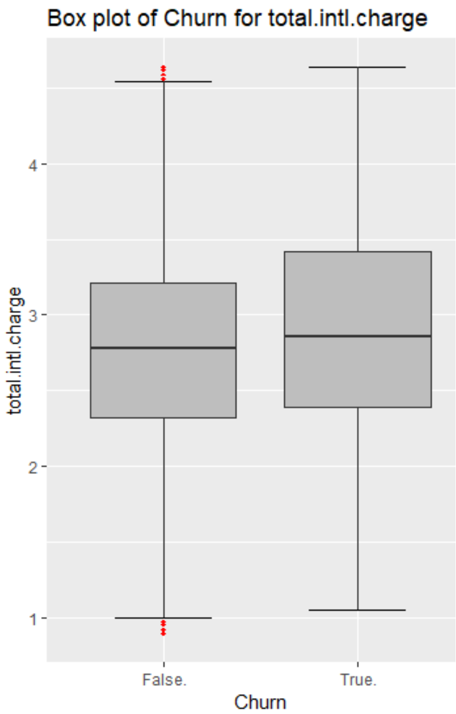

Box plot of Churn for total.night.minutes
total.night.minutes
300
250
200
150
100
False.
True.
Churn


Box plot of Churn for total.night.calls
total.night.calls
150
125
100
75
50
False.
True.
Churn


Box plot of Churn for total.night.charge
total.night.charge
12
8
4
False.
True.
Churn


Box plot of Churn for total.intl.minutes
total.intl.minutes
15
10
5
False.
True.
Churn


Box plot of Churn for total.intl.calls
total.intl.calls
10.0
7.5
5.0
2.5
0.0
False.
True.
Churn


Box plot of Churn for total.intl.charge
total.intl.charge
4
3
2
1
False.
True.
Churn

2.1.3 Feature Selection

Before performing any type of modelling we need to assess the importance of each predictor variable in our analysis. There is a possibility that many variables in our analysis are not important at all to the problem of class prediction. This process of selecting a subset of relevant features/variables is known as feature selection. There are several methods of doing feature selection. We have used correlation analysis for continuous variables and Chi-square test for categorical variables.

Correlation Analysis :

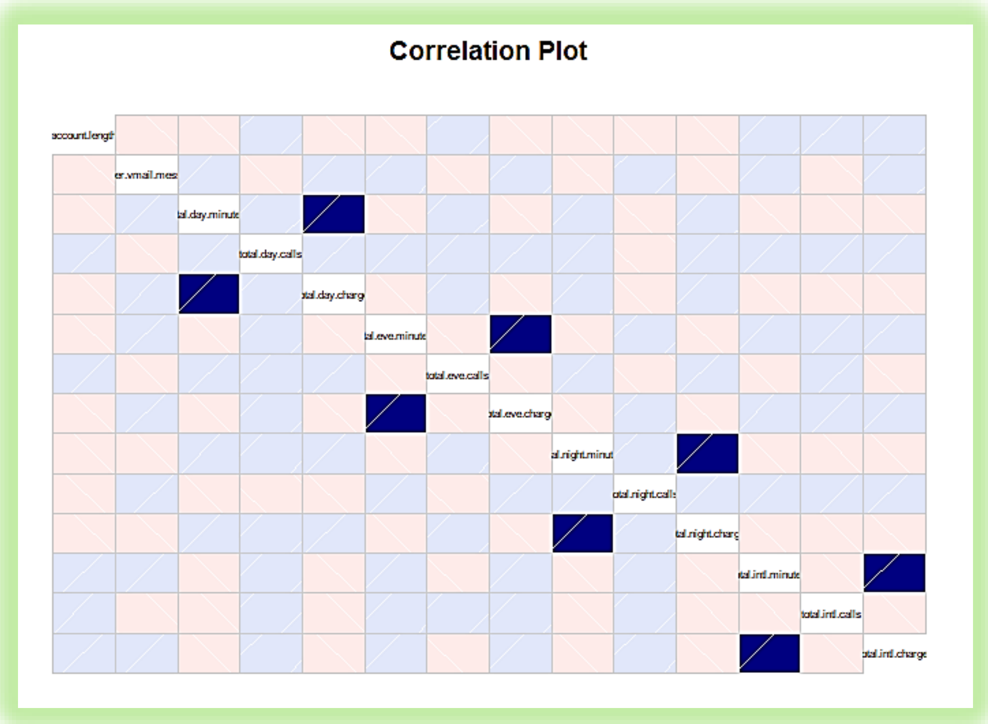


Figure 2.9 : Correlation plot before feature selection

The above Figure 2.9 is a correlation plot for continuous variables in churn reduction Data, it shows the correlation between two variables. The blue shade colour implies that two variables are positively correlated with each other and pinkish red colour implies that the variables are negatively correlated with each other.
By observing the heap map/correlation plot pattern we can find the variables that are highly correlated with any other variables and remove such variables which may lead to multicollinearity .
Below are the observations made with respect to the highly correlated variables:

- Variable ‘total day minutes’ is highly positively correlated with variable ‘total day charge’
- Variable ‘total eve minutes’ is highly positively correlated with variable ‘total eve charge’
- Variable ‘total night minutes’ is highly positively correlated with variable ‘total night charge’
- Variable ‘total intl minutes’ is highly positively correlated with variable ‘total intl charge’

Also, with general sense we can conclude that call minutes is directly proportional to the amount charged. Hence from the above four variable pairs we will remove any one of the variables in each pair. As charge is calculated based on the minutes been spoken, we'll remove the total charge variable from each of the four pairs.

The below figure 2.10 shows the correlation plot after removing the highly correlated variables total day charge, total eve charge, total night charge, total intl charge from our dataset.

We could now observe from the below correlation plot that our dataset is free from multicollinearity effect.

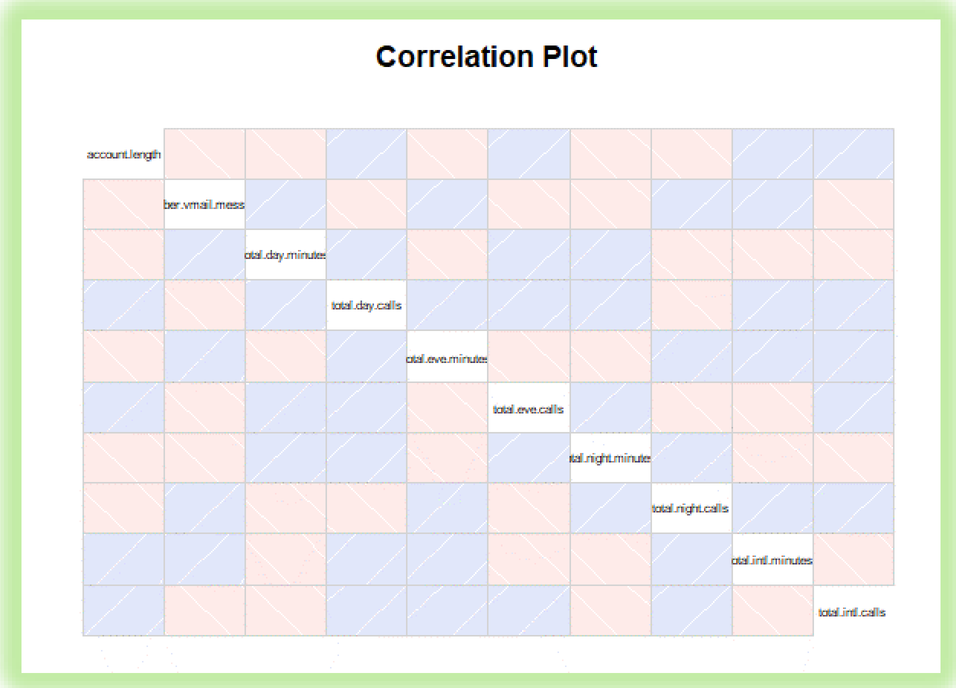


Figure 2.10 : Correlation plot after feature selection

Chi-Square Analysis or Chi-Square test of Independence :

Chi-square test of independence is used to check the dependency between the two categorical variables. The dependency between 2 independent variables must be low and dependency between independent & dependent variables must be high.

The Chi-square test uses contingency table to establish the relation between 2 categorical variables and is interpreted based on the p-value(probability value).

The null and alternate hypothesis in Chi-square test is given as :

- Null hypothesis : The 2 variables are independent.
- Alternate hypothesis : The 2 variables are not independent.

The Chi-square test is calculated by the formula :

$$c^2 = \sum_{i=1}^{k} \left[\frac{(O_i - E_i)^2}{E_i} \right]$$

Where Oi = Observed value & Ei = Expected value

In our Chi-Square test we are comparing each of the independent categorical variables with the dependent categorical variable, in other words we take the dependent variable as reference and analyse the Chi-square test with other independent variables.

For each test performed, the p-value is observed and,

- If the obtained p-value < 0.05, then we reject the null hypothesis concluding that there is a dependency between the target & independent variable.
- If the obtained p-value > 0.05, then we accept the null hypothesis concluding that our target and independent variables have no dependency on each other.

The below table 2.1 shows the p-values noted for each of the independent variables when compared with our target variable(Churn) during the Chi-Square test analysis.

Variable names	**p-value**
state	7.85E-05
area.code	0.7547
phone.number	0.4934
international.plan	2.20E-16
voice.mail.plan	7.17E-15
number.customer.service.calls	2.20E-16

Table 2.1 : p-values of Chi -Square test

From the above table we see that the variables area.code & phone.number have p-values < 0.05 and hence we are not including them in our further analysis and will drop them.

2.1.4 Feature Scaling

Data scaling or feature scaling is a method used to standardize the range of variables or features present in the dataset so that they can be compared on a common ground.

Since the range of values for some variables in the raw data vary highly in magnitudes, units and range, we need feature scaling to bring all the features/variables to the same level of magnitudes, else the whole output of our analysis may get biased to one of the variables.

Most of the machine learning algorithms which use distance-based calculation might go wrong in their calculations if we do not scale our variables in the dataset before feeding into the model.

Another reason why feature scaling is applied is that gradient descent converges much faster with feature scaling than without it.

As observed in the exploratory data analysis stage that our dataset is not much normalized on the whole for all the variables, we are considering normalization technique of feature scaling.

Normalization is a scaling method in which all the variables is brought into proportion with one another with values ranging from 0 to 1.

Normalization is given by:

$$\text{Value}_{\text{Norm}} = (\text{Value} - \text{MinValue}) / (\text{MaxValue} - \text{MinValue})$$

Where Minvalue and MaxValue are the minimum & maximum values of a given variable respectively.

2.2 Modelling

2.2.1 Preparing the data for modelling

In order for the machine learning algorithms to understand the data, it is necessary that we modify our data accordingly in a format which the machine learning algorithms can interpret, Hence we have to change the data of categorical variables to numerical data by assigning levels to the corresponding categories in a categorical variable.

By doing this, the data of target variable 'Churn' gets changed to 1 & 2 from False. & True. respectively.

2.2.2 Sampling the data

As we seen from the exploratory data analysis that the target variable 'Churn' has imbalanced dataset, it is necessary that we sample the minority class samples to a level equal to the majority class samples or sample both the majority & minority class samples equally to a level which would approximately be equal to the total original observations of the dataset.

For sampling the imbalanced classes of dataset, we are using SMOTE technique.

SMOTE stands for Synthetic Minority Oversampling Technique, this technique creates a subset of minority class, from this subset a new synthetic similar instances are created from the combination of neighbouring instances, which are finally added to the original dataset.

Why SMOTE ?

Oversampling the minority samples can lead to model overfitting, since it will introduce duplicate instances, drawing from a pool of instances that is already small. Similarly, under sampling the majority samples can end up leaving out important instances that provide important differences between the two or more classes. Hence, we have decided to use SMOTE.

2.2.3 Model Selection

In our dataset, the target variable to be predicted is 'Churn', which is a binary classifier with two classes 'False.' and 'True.' or in other words we need to predict whether a customer will churn or not churn. Hence this is a binary classification problem and we will build the following binary classification models and later finalize a particular model which would best-fit our criteria.

- Decision Tree model
- Random Forest model
- Logistic Regression
- KNN model
- Naïve Bayes model

2.2.4 Decision Tree Model

Decision tree is a supervised predictive model based on the branching series of Boolean tests. Decision tree generates a series of rules which it makes use for its prediction and in the tree, each branch connects the nodes with "and" and multiple branches are connected by "or".

The model that we are building uses C5.0 algorithm which makes use of information gain to identify the root node and proceed further to build the rules.

The root node of this decision tree would be that independent variable for which the information gain is high, or the entropy is low. The node of a tree corresponds to an attribute/variable and each leaf node corresponds to a class label.

While building a decision tree model, we can tune the model with different boosting iteration values using 'trials' parameter, below are the accuracy, false negative rate, recall and precision observed for different trials values during the prediction of test cases.

Trials value(N)	%Accuracy of prediction	FNR%	Recall%	Precision%
25	88.3	20.98	79	54.46
50	89.08	20.08	79.91	56.64
80	89.2	19.64	80.35	56.96
90	**89.2**	**19.19**	**80.8**	**56.91**
95	89.02	19.64	80.35	56.42
100	89.08	19.19	80.8	56.5

```
  (a)   (b)    <-classified as
 ----  ----
 1434    15    (a): class 1
   27  1422    (b): class 2


Attribute usage:

100.00% state
100.00% account.length
100.00% international.plan
100.00% number.vmail.messages
100.00% total.day.minutes
100.00% total.day.calls
100.00% total.eve.minutes
100.00% total.eve.calls
100.00% total.night.minutes
100.00% total.night.calls
100.00% total.intl.minutes
100.00% total.intl.calls
100.00% number.customer.service.calls
 99.90% voice.mail.plan
```

We are finalising the decision tree model with trials value 90 as it has a best fit with accuracy and false negative rate.

2.2.5 Random Forest Model

Random forest algorithm builds N number of decision trees like a forest by selecting a random number of initial variables and observations. It is an ensemble model that consists of multiple decision tree.

This ensemble technique improves the accuracy and reduces the misclassification error due to the weak learners by combining multiple decision tress back to back which then prepares a strong classifier.

In other words, Random forest is a combination of weak learners to produce a strong learner from its model.

The Random forest algorithm uses CART(classification & regression tree) which in turn makes use of Gini index to build all the decision trees internally. To classify a particular class the mode of all the classes generated by all the individual trees are considered.

We build a random forest model with different number of trees using 'ntrees' parameter and choose the best tree size that gives a best prediction. Below are the accuracy, false negative rate, recall and precision values observed for different tree values during the prediction of test cases.

ntree value	%Accuracy of prediction	FNR%	Recall%	Precision%
250	84.22	19.64	80.35	45.11
500	83.86	20.08	79.91	44.41
600	84.22	20.08	79.91	45.08
700	**84.4**	**19.19**	**80.8**	**45.47**
800	84.52	19.64	80.35	45.68
900	83.98	20.08	79.91	44.63
1000	84.4	19.64	80.35	45.45

We are finalising the random forest model with ntrees value 700 as it has a best fit with accuracy and false negative rate.

2.2.6 Logistic Regression

The logistic regression algorithm uses a sigmoid function/logit link function which transforms any numerical input to the probabilities between 0 and 1.

This algorithm uses dummy variables internally to split each category of a categorical variable separately, and the output class is determined by taking a threshold probability value (usually 0.05), if the value is > 0.05 it belongs to one class and if it is < 0.05 it belongs to another class.

The below table shows the accuracy, false negative rate, recall and precision values observed while predicting the test cases using the logistic regression model.

%Accuracy of prediction	FNR%	Recall%	Precision%
82.3	36.16	63.83	40.05

2.2.7 KNN Algorithm

K- Nearest Neighbours or KNN algorithm is a simple algorithm that stores all the available cases and classifies a new case based on a similarity measure. An object is classified by the majority vote of its neighbours, with the object being assigned to a class which is most common among its K nearest neighbours.

The commonly used distance metric to calculate the K nearest neighbours is Euclidean's distance method which will be used in our model too.

The KNN algorithm is referred to as "Lazy Learning Algorithm" as it learns about the data only when a prediction is requested and not prior to the prediction.

Below are the accuracy, false negative rate, recall and precision values observed for different values of k during the prediction of test cases.

k value	%Accuracy of prediction	FNR%	Recall%	Precision%
3	**77.08**	**41.07**	**58.92**	**31.27**
5	78.22	49.1	50.8	31.06
7	79.18	52.67	47.32	31.64
9	79.24	57.14	42.85	30.57
11	79.96	58.92	41.07	31.29
15	80.8	59.82	40.17	32.6

We are finalising the KNN model with k value of 3 as it has a best fit with accuracy and false negative rate.

2.2.8 Naïve Bayes Model

Naïve Bayes algorithm is based on Baye's theorem which assumes a strong independence among the predictor variables to predict the class of unknown dataset. In simple terms, a Naïve Baye's classifier assumes that all the independent variables are conditionally independent with each other as it employs probabilistic approach in evaluating the target class. Hence the name "Naïve".

Example : A fruit may be an apple if it is red, round, and about 10 cm in diameter. A Naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple regardless of any possible correlations between the colour, roundness, and diameter features.

The Baye's theorem for calculating the posterior probability is given as,

Likelihood   Class Prior Probability

$$P(c \mid x) = \frac{P(x \mid c)\,P(c)}{P(x)}$$

Posterior Probability   Predictor Prior Probability

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Where,

- P(c|x) is the posterior probability of class(c,target) given predictor(x,attributes).
- P(c) is the prior probability of class.

- P(x|c) is the likelihood which is the probability of predictor given class.
- P(x) is the prior probability of predictor.

Below are the accuracy, false negative rate, recall and precision values observed during the prediction of test cases using Naïve Bayes model.

%Accuracy of prediction	FNR%	Recall%	Precision%
81.34	42.85	57.14	37.31

Chapter 3

Conclusion

3.1 Model Evaluation

Now that we have built a few models for predicting the target variable, we need to decide which one of the models to be chosen. There are several criteria that exists for evaluating and comparing the models. We can compare the models using any of the following criteria :

1.) Predictive performance
2.) Interpretability
3.) Computational efficiency

In our case, we are using predictive performance to decide and compare the models as the other two criteria do not hold much significance.

Predictive performance can be measured by comparing the predictions of the models with the real values of the target test variable, and by analysing various metrics from the confusion matrix.

The confusion matrix is a table that is used to describe the performance of a classification model on a set of test data for which the true values are known. The confusion matrix which is considered for our models has the actual target values as rows and the predicted target values as columns.

The below figure shows a sample confusion matrix

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

Terms associated with confusion matrix :

- True positives : True positives are the cases when the actual cases of the data point are true and the predicted is also true.
  Ex : The case where a customer is actually churning out and the model also classifies that customer is going to churn.

- True negatives : True negatives are the cases when the actual cases of the data point are false and the predicted is also false.
  Ex : The case where a customer is not churning out and the model also classifies that customer is not going to churn.

- False positives : False positives are the cases when the actual class of the data point is false, and the model prediction is true.
  Ex : When a customer is actually not churning out, but the model predicts that the customer is churning out.

- False negatives : False negatives are the cases when the actual class of the data point is true, and the model prediction is false.
  Ex : When a customer is actually churning out, but the model predicts that the customer is not churning out.

3.1.1 False negative rate(FNR)

False negative rate is an error in which a prediction result improperly indicates that a customer is not churning out but in reality, he is actually churning out.

FNR is given by : $\frac{False\ negative}{False\ negative\ + True\ Positive}$

As our business objective is to reduce the number of customers churning out, the goal of our model must be in such a way that it predicts the false negatives as much low as possible or in other words the false negative rate must be very low.

3.1.2 Recall or Sensitivity

Recall is a measure that tells us that what proportion of customers who had actually churned out was predicted by the model also as being churned out.

Recall is given by : $\frac{True\ positive}{True\ Positive\ +\ False\ negative}$

It is also necessary that our model as high recall rate so that the correct number predictions are made, and by this the necessary measures can be taken to minimize the customer churn.

3.1.3 Precision

Precision is a measure that tells us what proportion of customers churned out had actually churned out.

Precision is given by : $\frac{True\ positive}{True\ Positive\ +\ False\ positive}$

3.1.4 Accuracy

Accuracy is the number of correct predictions made by the model over all kinds of predictions made.

Accuracy is given by : $\frac{True\ positive + True\ negative}{True\ Positive + False\ positive + False\ negative + True\ negative}$

On the whole we are more concerned about the FNR and Recall rate than accuracy for our business objective.

3.2 Model Selection

As it can be seen from the below table 3.1 that decision tree model has high accuracy, low false negative rate and high recall rate compared to other models and closely random forest matches with decision tree's FNR and recall but fails on accuracy part to some extent. Hence, we go with decision tree model as the best choice of prediction.

Table 3.1: Table of various models and its related metrics

Model	%Accuracy of prediction	FNR%	Recall%	Precision%
Decision Tree	89.2	19.19	80.8	56.91
Random Forest	84.4	19.19	80.8	45.47
Logistic Regression	82.3	36.16	63.83	40.05
KNN	77.08	41.07	58.92	31.27
Naïve Bayes	81.34	42.85	57.14	37.31

Appendix – A

XgBoost implementation in R

Optionally, I have also tried the customer churn analysis with XgBoost model in R and below are my observations.

XgBoost stands for extreme gradient boosting algorithm which boosts the weak learners to become strong enough to predict the target variable.

Setting of nrounds and max depth parameters : I have set nrounds and max.depth as 10 as it gave the best AUC value of 0.99 for the train cases. Below are the working screenshots for different nrounds that I have tested.

- For max.depth= 5 and nrounds= 5 :

```
> train_matrix = xgb.DMatrix(data=as.matrix(trainm), label= train_label)
> test_matrix= xgb.DMatrix(data= as.matrix(testm), label = test_label)
> xg_model= xgboost(data = train_matrix,label=train_label,max.depth=5,eta=0.2,nrounds =5,objective="binar
y:logistic",eval_metric="auc",verbose = 1)
[1]	train-auc:0.940259
[2]	train-auc:0.942775
[3]	train-auc:0.947129
[4]	train-auc:0.948444
[5]	train-auc:0.950660
```

- For max.depth= 10 and nrounds= 5 :

```
> xg_model= xgboost(data = train_matrix,label=train_label,max.depth=10,eta=0.2,nrounds =5,objective="bina
ry:logistic",eval_metric="auc",verbose = 1)
[1]	train-auc:0.964873
[2]	train-auc:0.977761
[3]	train-auc:0.980929
[4]	train-auc:0.983388
[5]	train-auc:0.986624
```

- For max.depth= 10 and nrounds= 10 :

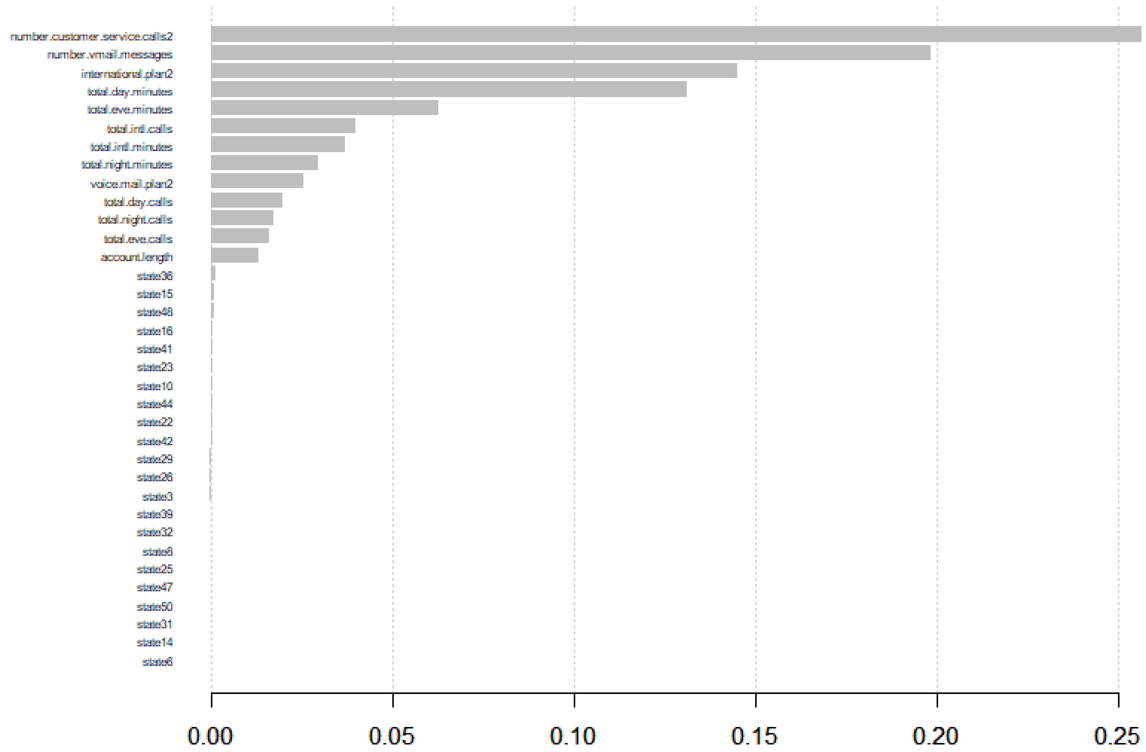
```
> xg_model= xgboost(data = train_matrix,label=train_label,max.depth=10,eta=0.2,nrounds =10,objective="bin
ary:logistic",eval_metric="auc",verbose = 1)
[1]	train-auc:0.964873
[2]	train-auc:0.977761
[3]	train-auc:0.980929
[4]	train-auc:0.983388
[5]	train-auc:0.986624
[6]	train-auc:0.988933
[7]	train-auc:0.991450
[8]	train-auc:0.993553
[9]	train-auc:0.994588
[10]	train-auc:0.995003
```

Confusion matrix and related metrics after the prediction of test cases :

```
> print(cm_xgb)
      predicted
actual    0    1
     0 1312  131
     1   50  174
```

Model	%Accuracy of prediction	FNR%	Recall%	Precision%
XgBoost	89.14	22.32	77.67	57.04

Important variables as per XgBoost model evaluation :



Appendix – B

Example of output with a sample input :

Below are the R code output screenshots captured for the input Churn data project.

The graphs are same as attached in the Chapter-1 and hence I'm not attaching them here.

- Structure of whole dataset after changing the area.code and number of customer service calls to factor

```
> #To have a look at the structure of the dataset
> str(whole_data)
'data.frame':   5000 obs. of  21 variables:
 $ state                        : Factor w/ 51 levels "AK","AL","AR",..: 17 36 32 36 37 2 20 25 19 50 ...
 $ account.length               : int  128 107 137 84 75 118 121 147 117 141 ...
 $ area.code                    : Factor w/ 3 levels "408","415","510": 2 2 2 1 2 3 3 2 1 2 ...
 $ phone.number                 : Factor w/ 5000 levels " 327-1058"," 327-1319",..: 1927 1576 1118 1708 1
11 2254 1048 81 292 118 ...
 $ international.plan           : Factor w/ 2 levels " no"," yes": 1 1 1 2 2 2 1 2 1 2 ...
 $ voice.mail.plan              : Factor w/ 2 levels " no"," yes": 2 2 1 1 1 1 2 1 1 2 ...
 $ number.vmail.messages        : int  25 26 0 0 0 0 24 0 0 37 ...
 $ total.day.minutes            : num  265 162 243 299 167 ...
 $ total.day.calls              : int  110 123 114 71 113 98 88 79 97 84 ...
 $ total.day.charge             : num  45.1 27.5 41.4 50.9 28.3 ...
 $ total.eve.minutes            : num  197.4 195.5 121.2 61.9 148.3 ...
 $ total.eve.calls              : int  99 103 110 88 122 101 108 94 80 111 ...
 $ total.eve.charge             : num  16.78 16.62 10.3 5.26 12.61 ...
 $ total.night.minutes          : num  245 254 163 197 187 ...
 $ total.night.calls            : int  91 103 104 89 121 118 118 96 90 97 ...
 $ total.night.charge           : num  11.01 11.45 7.32 8.86 8.41 ...
 $ total.intl.minutes           : num  10 13.7 12.2 6.6 10.1 6.3 7.5 7.1 8.7 11.2 ...
 $ total.intl.calls             : int  3 3 5 7 3 6 7 6 4 5 ...
 $ total.intl.charge            : num  2.7 3.7 3.29 1.78 2.73 1.7 2.03 1.92 2.35 3.02 ...
 $ number.customer.service.calls: Factor w/ 3 levels "High","Low","Moderate": 2 2 2 2 2 2 2 2 2 2 ...
 $ Churn                        : Factor w/ 2 levels " False."," True.": 1 1 1 1 1 1 1 1 1 1 ...
```

- Missing_val dataframe info :

	Columns	Missing_percentage
1	state	0
2	account.length	0
3	area.code	0
4	phone.number	0
5	international.plan	0
6	voice.mail.plan	0
7	number.vmail.messages	0
8	total.day.minutes	0
9	total.day.calls	0
10	total.day.charge	0
11	total.eve.minutes	0
12	total.eve.calls	0
13	total.eve.charge	0
14	total.night.minutes	0
15	total.night.calls	0
16	total.night.charge	0
17	total.intl.minutes	0
18	total.intl.calls	0
19	total.intl.charge	0
20	number.customer.service.calls	0
21	Churn	0

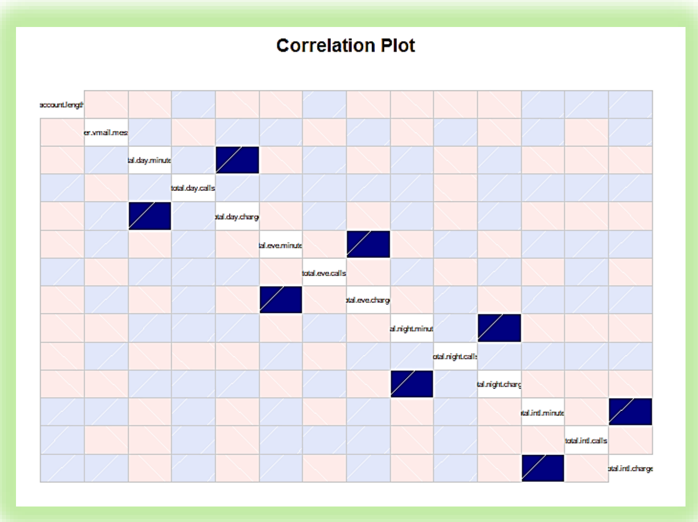
➢ Code that was used to check the standard deviation for different values of k for KNN Imputation.

```
std= function(x)
{
  sd_value=sd(x)
  sd_value
}
sd_data=data.frame(apply(whole_data[,cnames],2,std))
names(sd_data)="Original SD"
```

K is selected as 9 as it gave the low standard deviation value :

	Original SD	SD_for_K3.	SD_for_K5.	SD_for_K7.	SD_for_K9.
account.length	39.6945595	38.8378069	38.8265127	38.8172174	38.8140167
number.vmail.messages	13.5463934	12.9457764	12.9157912	12.9109930	12.9063600
total.day.minutes	53.8946992	52.3369891	52.2959352	52.2822597	52.2836775
total.day.calls	19.8311974	19.1692018	19.1600225	19.1572317	19.1543170
total.day.charge	9.1620687	8.8972282	8.8902492	8.8879245	8.8881656
total.eve.minutes	50.5513090	48.6832487	48.6614703	48.6483199	48.6433079
total.eve.calls	19.8264958	19.3062420	19.3059591	19.3037563	19.3026089
total.eve.charge	4.2968433	4.1380599	4.1362078	4.1350904	4.1346644
total.night.minutes	50.5277893	48.6873713	48.6745329	48.6672053	48.6579532
total.night.calls	19.9586859	19.1700309	19.1505845	19.1464836	19.1449685
total.night.charge	2.2737627	2.1909360	2.1903584	2.1900280	2.1896112
total.intl.minutes	2.7613957	2.5539659	2.5517836	2.5510719	2.5505463
total.intl.calls	2.4567882	2.0627010	2.0595883	2.0585922	2.0579081
total.intl.charge	0.7455137	0.6894981	0.6889097	0.6887175	0.6885756

➢ Correlation plot :



➢ Chi-square test analysis output :

```
[1] "state"

        Pearson's Chi-squared test

data:  table(factor_data$Churn, factor_data[, i])
X-squared = 96.899, df = 50, p-value = 7.851e-05

[1] "area.code"

        Pearson's Chi-squared test

data:  table(factor_data$Churn, factor_data[, i])
X-squared = 0.56298, df = 2, p-value = 0.7547

[1] "phone.number"

        Pearson's Chi-squared test

data:  table(factor_data$Churn, factor_data[, i])
X-squared = 5000, df = 4999, p-value = 0.4934

[1] "international.plan"

        Pearson's Chi-squared test with Yates' continuity correction

data:  table(factor_data$Churn, factor_data[, i])
X-squared = 333.19, df = 1, p-value < 2.2e-16

[1] "voice.mail.plan"

        Pearson's Chi-squared test with Yates' continuity correction

data:  table(factor_data$Churn, factor_data[, i])
X-squared = 60.552, df = 1, p-value = 7.165e-15

[1] "number.customer.service.calls"

        Pearson's Chi-squared test

data:  table(factor_data$Churn, factor_data[, i])
X-squared = 470, df = 2, p-value < 2.2e-16
```

➢ Normalization output verification :

```
> cnames= colnames(whole_data[,sapply(whole_data, is.numeric)])
> # Normalization
> for ( i in cnames)
+ {
+   whole_data[,i]= ((whole_data[,i]- min(whole_data[,i]))/( max(whole_data[,i])- min(whole_data[,i])))
+ }
>
> #to check the max & min values to verify the normalization
> range(whole_data[,cnames])
[1] 0 1
```

➢ Before sampling the train data, the below is the class values for 1 & 2.

```
> table(train_data$Churn)

   1    2
2850  483
```

```
> train_percentage= ((nrow(train_data)*100)/nrow(whole_data))
> test_percentage= ((nrow(test_data)*100) / nrow(whole_data))
> cat("The train data percentage is = ",train_percentage)
The train data percentage is =  66.66> cat ("The test data percentage is = ",test_percentage)
The test data percentage is =  33.34
```

- **After SMOTE, below is the balanced class values**

```
> table(smote_train$Churn)

   1    2 
1449 1449 
```

Decision tree model's classification matrix and other metrics for each of the trials.

- **For trials=25**

```
> DT_model = C5.0(Churn ~., smote_train, trials =25, rules = TRUE)
> #Lets predict for test cases
> DT_Predictions = predict(DT_model,test_data[,-15], type = "class")
> ##Evaluate the performance of classification model
> ConfMatrix_C50 = table(actual= test_data$Churn, predicted= DT_Predictions)
> print(ConfMatrix_C50)
      predicted
actual    1    2
     1 1295  148
     2   47  177
> metrics_list= metrics(ConfMatrix_C50)
> print(paste0("Decision Tree Accuracy is : ", metrics_list[1]))
[1] "Decision Tree Accuracy is : 88.3023395320936"
> print(paste0("False Negative rate of Decision Tree is : ", metrics_list[2]))
[1] "False Negative rate of Decision Tree is : 20.9821428571429"
> print(paste0("Recall of Decision Tree is : ", metrics_list[3]))
[1] "Recall of Decision Tree is : 79.0178571428571"
> print(paste0("Precision of Decision Tree is : ", metrics_list[4]))
[1] "Precision of Decision Tree is : 54.4615384615385"
```

- **For trials=50**

```
> DT_model2 = C5.0(Churn ~., smote_train, trials = 50, rules = TRUE)
> DT_Predictions2 = predict(DT_model2,test_data[,-15], type = "class")
> rm(DT_Predictions2)
> DT_model = C5.0(Churn ~., smote_train, trials =50, rules = TRUE)
> #Lets predict for test cases
> DT_Predictions = predict(DT_model,test_data[,-15], type = "class")
> ##Evaluate the performance of classification model
> ConfMatrix_C50 = table(actual= test_data$Churn, predicted= DT_Predictions)
> print(ConfMatrix_C50)
      predicted
actual    1    2
     1 1306  137
     2   45  179
> metrics_list= metrics(ConfMatrix_C50)
> print(paste0("Decision Tree Accuracy is : ", metrics_list[1]))
[1] "Decision Tree Accuracy is : 89.0821835632873"
> print(paste0("False Negative rate of Decision Tree is : ", metrics_list[2]))
[1] "False Negative rate of Decision Tree is : 20.0892857142857"
> print(paste0("Recall of Decision Tree is : ", metrics_list[3]))
[1] "Recall of Decision Tree is : 79.9107142857143"
> print(paste0("Precision of Decision Tree is : ", metrics_list[4]))
[1] "Precision of Decision Tree is : 56.6455696202532"
```

- **For trials=80**

```
> DT_model = C5.0(Churn ~., smote_train, trials =80, rules = TRUE)
> #Lets predict for test cases
> DT_Predictions = predict(DT_model,test_data[,-15], type = "class")
> ##Evaluate the performance of classification model
> ConfMatrix_C50 = table(actual= test_data$Churn, predicted= DT_Predictions)
> print(ConfMatrix_C50)
      predicted
actual    1    2
     1 1307  136
     2   44  180
> metrics_list= metrics(ConfMatrix_C50)
> print(paste0("Decision Tree Accuracy is : ", metrics_list[1]))
[1] "Decision Tree Accuracy is : 89.2021595680864"
> print(paste0("False Negative rate of Decision Tree is : ", metrics_list[2]))
[1] "False Negative rate of Decision Tree is : 19.6428571428571"
> print(paste0("Recall of Decision Tree is : ", metrics_list[3]))
[1] "Recall of Decision Tree is : 80.3571428571429"
> print(paste0("Precision of Decision Tree is : ", metrics_list[4]))
[1] "Precision of Decision Tree is : 56.9620253164557"
```

- **For trials=90**

```
> set.seed(321)
> DT_model = C5.0(Churn ~., smote_train, trials =90, rules = TRUE)
> #Lets predict for test cases
> DT_Predictions = predict(DT_model,test_data[,-15], type = "class")
> ##Evaluate the performance of classification model
> ConfMatrix_C50 = table(actual= test_data$Churn, predicted= DT_Predictions)
> print(ConfMatrix_C50)
      predicted
actual    1    2
     1 1306  137
     2   43  181
> metrics_list= metrics(ConfMatrix_C50)
> print(paste0("Decision Tree Accuracy is : ", metrics_list[1]))
[1] "Decision Tree Accuracy is : 89.2021595680864"
> print(paste0("False Negative rate of Decision Tree is : ", metrics_list[2]))
[1] "False Negative rate of Decision Tree is : 19.1964285714286"
> print(paste0("Recall of Decision Tree is : ", metrics_list[3]))
[1] "Recall of Decision Tree is : 80.8035714285714"
> print(paste0("Precision of Decision Tree is : ", metrics_list[4]))
[1] "Precision of Decision Tree is : 56.9182389937107"
```

- **For trials=95**

```
> DT_model = C5.0(Churn ~., smote_train, trials =95, rules = TRUE)
> #Lets predict for test cases
> DT_Predictions = predict(DT_model,test_data[,-15], type = "class")
> ##Evaluate the performance of classification model
> ConfMatrix_C50 = table(actual= test_data$Churn, predicted= DT_Predictions)
> print(ConfMatrix_C50)
      predicted
actual    1    2
     1 1304  139
     2   44  180
> metrics_list= metrics(ConfMatrix_C50)
> print(paste0("Decision Tree Accuracy is : ", metrics_list[1]))
[1] "Decision Tree Accuracy is : 89.0221955608878"
> print(paste0("False Negative rate of Decision Tree is : ", metrics_list[2]))
[1] "False Negative rate of Decision Tree is : 19.6428571428571"
> print(paste0("Recall of Decision Tree is : ", metrics_list[3]))
[1] "Recall of Decision Tree is : 80.3571428571429"
> print(paste0("Precision of Decision Tree is : ", metrics_list[4]))
[1] "Precision of Decision Tree is : 56.4263322884012"
```

- **For trials=100**

```
> DT_model = C5.0(Churn ~., smote_train, trials =100, rules = TRUE)
> #Lets predict for test cases
> DT_Predictions = predict(DT_model,test_data[,-15], type = "class")
> ##Evaluate the performance of classification model
> ConfMatrix_C50 = table(actual= test_data$Churn, predicted= DT_Predictions)
> print(ConfMatrix_C50)
      predicted
actual    1    2
     1 1304  139
     2   43  181
> metrics_list= metrics(ConfMatrix_C50)
> print(paste0("Decision Tree Accuracy is : ", metrics_list[1]))
[1] "Decision Tree Accuracy is : 89.0821835632873"
> print(paste0("False Negative rate of Decision Tree is : ", metrics_list[2]))
[1] "False Negative rate of Decision Tree is : 19.1964285714286"
> print(paste0("Recall of Decision Tree is : ", metrics_list[3]))
[1] "Recall of Decision Tree is : 80.8035714285714"
> print(paste0("Precision of Decision Tree is : ", metrics_list[4]))
[1] "Precision of Decision Tree is : 56.5625"
```

Random forest model's classification matrix and other metrics for each of the tree values tested.

- **For ntree=250**

```
> RF_model = randomForest(Churn ~ .,smote_train, importance = TRUE, ntree = 250)
> RF_Predictions = predict(RF_model, test_data[,-15])
> ConfMatrix_RF = table(actual=test_data$Churn, predicted= RF_Predictions)
> print(ConfMatrix_RF)
      predicted
actual    1    2
     1 1224  219
     2   44  180
> RF_metrics= metrics(ConfMatrix_RF)
> print(paste0("Random Forest Accuracy % is : ", RF_metrics[1]))
[1] "Random Forest Accuracy % is : 84.2231553689262"
> print(paste0("False Negative rate % of Random Forest is : ", RF_metrics[2]))
[1] "False Negative rate % of Random Forest is : 19.6428571428571"
> print(paste0("Recall % of Random Forest is : ", RF_metrics[3]))
[1] "Recall % of Random Forest is : 80.3571428571429"
> print(paste0("Precision % of Random forest is : ", RF_metrics[4]))
[1] "Precision % of Random forest is : 45.1127819548872"
```

- **For ntree=500**

```
> RF_model = randomForest(Churn ~ .,smote_train, importance = TRUE, ntree = 500)
> RF_Predictions = predict(RF_model, test_data[,-15])
> ConfMatrix_RF = table(actual=test_data$Churn, predicted= RF_Predictions)
> print(ConfMatrix_RF)
      predicted
actual    1    2
     1 1219  224
     2   45  179
> RF_metrics= metrics(ConfMatrix_RF)
> print(paste0("Random Forest Accuracy % is : ", RF_metrics[1]))
[1] "Random Forest Accuracy % is : 83.8632273545291"
> print(paste0("False Negative rate % of Random Forest is : ", RF_metrics[2]))
[1] "False Negative rate % of Random Forest is : 20.0892857142857"
> print(paste0("Recall % of Random Forest is : ", RF_metrics[3]))
[1] "Recall % of Random Forest is : 79.9107142857143"
> print(paste0("Precision % of Random forest is : ", RF_metrics[4]))
[1] "Precision % of Random forest is : 44.4168734491315"
```

- **For ntree=600**

```
> RF_model = randomForest(Churn ~ .,smote_train, importance = TRUE, ntree = 600)
> RF_Predictions = predict(RF_model, test_data[,-15])
> ConfMatrix_RF = table(actual=test_data$Churn, predicted= RF_Predictions)
> print(ConfMatrix_RF)
      predicted
actual    1    2
     1 1225  218
     2   45  179
> RF_metrics= metrics(ConfMatrix_RF)
> print(paste0("Random Forest Accuracy % is : ", RF_metrics[1]))
[1] "Random Forest Accuracy % is : 84.2231553689262"
> print(paste0("False Negative rate % of Random Forest is : ", RF_metrics[2]))
[1] "False Negative rate % of Random Forest is : 20.0892857142857"
> print(paste0("Recall % of Random Forest is : ", RF_metrics[3]))
[1] "Recall % of Random Forest is : 79.9107142857143"
> print(paste0("Precision % of Random forest is : ", RF_metrics[4]))
[1] "Precision % of Random forest is : 45.088161209068"
```

- **For ntree=700**

```
> RF_model = randomForest(Churn ~ .,smote_train, importance = TRUE, ntree = 700)
> RF_Predictions = predict(RF_model, test_data[,-15])
> ConfMatrix_RF = table(actual=test_data$Churn, predicted= RF_Predictions)
> print(ConfMatrix_RF)
      predicted
actual    1    2
     1 1226  217
     2   43  181
> RF_metrics= metrics(ConfMatrix_RF)
> print(paste0("Random Forest Accuracy % is : ", RF_metrics[1]))
[1] "Random Forest Accuracy % is : 84.4031193761248"
> print(paste0("False Negative rate % of Random Forest is : ", RF_metrics[2]))
[1] "False Negative rate % of Random Forest is : 19.1964285714286"
> print(paste0("Recall % of Random Forest is : ", RF_metrics[3]))
[1] "Recall % of Random Forest is : 80.8035714285714"
> print(paste0("Precision % of Random forest is : ", RF_metrics[4]))
[1] "Precision % of Random forest is : 45.4773869346734"
```

- **For ntree=800**

```
> RF_model = randomForest(Churn ~ .,smote_train, importance = TRUE, ntree = 800)
> RF_Predictions = predict(RF_model, test_data[,-15])
> ConfMatrix_RF = table(actual=test_data$Churn, predicted= RF_Predictions)
> print(ConfMatrix_RF)
      predicted
actual    1    2
     1 1229  214
     2   44  180
> RF_metrics= metrics(ConfMatrix_RF)
> print(paste0("Random Forest Accuracy % is : ", RF_metrics[1]))
[1] "Random Forest Accuracy % is : 84.5230953809238"
> print(paste0("False Negative rate % of Random Forest is : ", RF_metrics[2]))
[1] "False Negative rate % of Random Forest is : 19.6428571428571"
> print(paste0("Recall % of Random Forest is : ", RF_metrics[3]))
[1] "Recall % of Random Forest is : 80.3571428571429"
> print(paste0("Precision % of Random forest is : ", RF_metrics[4]))
[1] "Precision % of Random forest is : 45.6852791878173"
```

- **For ntree=900**

```
> RF_model = randomForest(Churn ~ .,smote_train, importance = TRUE, ntree = 900)
> RF_Predictions = predict(RF_model, test_data[,-15])
> ConfMatrix_RF = table(actual=test_data$Churn, predicted= RF_Predictions)
> print(ConfMatrix_RF)
      predicted
actual    1    2
     1 1221  222
     2   45  179
> RF_metrics= metrics(ConfMatrix_RF)
> print(paste0("Random Forest Accuracy % is : ", RF_metrics[1]))
[1] "Random Forest Accuracy % is : 83.9832033593281"
> print(paste0("False Negative rate % of Random Forest is : ", RF_metrics[2]))
[1] "False Negative rate % of Random Forest is : 20.0892857142857"
> print(paste0("Recall % of Random Forest is : ", RF_metrics[3]))
[1] "Recall % of Random Forest is : 79.9107142857143"
> print(paste0("Precision % of Random forest is : ", RF_metrics[4]))
[1] "Precision % of Random forest is : 44.6384039900249"
```

- **For ntree=1000**

```
> RF_model = randomForest(Churn ~ .,smote_train, importance = TRUE, ntree = 1000)
> RF_Predictions = predict(RF_model, test_data[,-15])
> ConfMatrix_RF = table(actual=test_data$Churn, predicted= RF_Predictions)
> print(ConfMatrix_RF)
      predicted
actual    1    2
     1 1227  216
     2   44  180
> RF_metrics= metrics(ConfMatrix_RF)
> print(paste0("Random Forest Accuracy % is : ", RF_metrics[1]))
[1] "Random Forest Accuracy % is : 84.4031193761248"
> print(paste0("False Negative rate % of Random Forest is : ", RF_metrics[2]))
[1] "False Negative rate % of Random Forest is : 19.6428571428571"
> print(paste0("Recall % of Random Forest is : ", RF_metrics[3]))
[1] "Recall % of Random Forest is : 80.3571428571429"
> print(paste0("Precision % of Random forest is : ", RF_metrics[4]))
[1] "Precision % of Random forest is : 45.4545454545455"
```

KNN model's classification matrix and other metrics for each of the k values tested.

- **For k=3**

```
> KNN_Predictions = knn(smote_train[, 1:14], test_data[, 1:14], smote_train$Churn, k = 3)
> ConfMatrix_KNN = table(actual=test_data$Churn, predicted= KNN_Predictions)
> print(ConfMatrix_KNN)
      predicted
actual    1    2
     1 1153  290
     2   92  132
> KNN_metrics= metrics(ConfMatrix_KNN)
> print(paste0("KNN Accuracy % is : ", KNN_metrics[1]))
[1] "KNN Accuracy % is : 77.0845830833833"
> print(paste0("False Negative rate % of KNN is : ", KNN_metrics[2]))
[1] "False Negative rate % of KNN is : 41.0714285714286"
> print(paste0("Recall % of KNN is : ", KNN_metrics[3]))
[1] "Recall % of KNN is : 58.9285714285714"
> print(paste0("Precision % of KNN is : ", KNN_metrics[4]))
[1] "Precision % of KNN is : 31.2796208530806"
```

- **For k=5**

```
> KNN_Predictions = knn(smote_train[, 1:14], test_data[, 1:14], smote_train$Churn, k = 5)
> ConfMatrix_KNN = table(actual=test_data$Churn, predicted= KNN_Predictions)
> print(ConfMatrix_KNN)
      predicted
actual    1    2
     1 1190  253
     2  110  114
> KNN_metrics= metrics(ConfMatrix_KNN)
> print(paste0("KNN Accuracy % is : ", KNN_metrics[1]))
[1] "KNN Accuracy % is : 78.2243551289742"
> print(paste0("False Negative rate % of KNN is : ", KNN_metrics[2]))
[1] "False Negative rate % of KNN is : 49.1071428571429"
> print(paste0("Recall % of KNN is : ", KNN_metrics[3]))
[1] "Recall % of KNN is : 50.8928571428571"
> print(paste0("Precision % of KNN is : ", KNN_metrics[4]))
[1] "Precision % of KNN is : 31.0626702997275"
```

- **For k=7**

```
> KNN_Predictions = knn(smote_train[, 1:14], test_data[, 1:14], smote_train$Churn, k = 7)
> ConfMatrix_KNN = table(actual=test_data$Churn, predicted= KNN_Predictions)
> print(ConfMatrix_KNN)
      predicted
actual    1    2
     1 1214  229
     2  118  106
> KNN_metrics= metrics(ConfMatrix_KNN)
> print(paste0("KNN Accuracy % is : ", KNN_metrics[1]))
[1] "KNN Accuracy % is : 79.1841631673665"
> print(paste0("False Negative rate % of KNN is : ", KNN_metrics[2]))
[1] "False Negative rate % of KNN is : 52.6785714285714"
> print(paste0("Recall % of KNN is : ", KNN_metrics[3]))
[1] "Recall % of KNN is : 47.3214285714286"
> print(paste0("Precision % of KNN is : ", KNN_metrics[4]))
[1] "Precision % of KNN is : 31.6417910447761"
```

- **For k=9**

```
> KNN_Predictions = knn(smote_train[, 1:14], test_data[, 1:14], smote_train$Churn, k = 9)
> ConfMatrix_KNN = table(actual=test_data$Churn, predicted= KNN_Predictions)
> print(ConfMatrix_KNN)
      predicted
actual    1    2
     1 1225  218
     2  128   96
> KNN_metrics= metrics(ConfMatrix_KNN)
> print(paste0("KNN Accuracy % is : ", KNN_metrics[1]))
[1] "KNN Accuracy % is : 79.244151169766"
> print(paste0("False Negative rate % of KNN is : ", KNN_metrics[2]))
[1] "False Negative rate % of KNN is : 57.1428571428571"
> print(paste0("Recall % of KNN is : ", KNN_metrics[3]))
[1] "Recall % of KNN is : 42.8571428571429"
> print(paste0("Precision % of KNN is : ", KNN_metrics[4]))
[1] "Precision % of KNN is : 30.5732484076433"
```

- **For k=11**

```
> KNN_Predictions = knn(smote_train[, 1:14], test_data[, 1:14], smote_train$Churn, k = 11)
> ConfMatrix_KNN = table(actual=test_data$Churn, predicted= KNN_Predictions)
> print(ConfMatrix_KNN)
      predicted
actual    1    2
     1 1241  202
     2  132   92
> KNN_metrics= metrics(ConfMatrix_KNN)
> print(paste0("KNN Accuracy % is : ", KNN_metrics[1]))
[1] "KNN Accuracy % is : 79.9640071985603"
> print(paste0("False Negative rate % of KNN is : ", KNN_metrics[2]))
[1] "False Negative rate % of KNN is : 58.9285714285714"
> print(paste0("Recall % of KNN is : ", KNN_metrics[3]))
[1] "Recall % of KNN is : 41.0714285714286"
> print(paste0("Precision % of KNN is : ", KNN_metrics[4]))
[1] "Precision % of KNN is : 31.2925170068027"
```

- **For k=15**

```
> KNN_Predictions = knn(smote_train[, 1:14], test_data[, 1:14], smote_train$Churn, k = 15)
> ConfMatrix_KNN = table(actual=test_data$Churn, predicted= KNN_Predictions)
> print(ConfMatrix_KNN)
      predicted
actual    1    2
     1 1257  186
     2  134   90
> KNN_metrics= metrics(ConfMatrix_KNN)
> print(paste0("KNN Accuracy % is : ", KNN_metrics[1]))
[1] "KNN Accuracy % is : 80.8038392321536"
> print(paste0("False Negative rate % of KNN is : ", KNN_metrics[2]))
[1] "False Negative rate % of KNN is : 59.8214285714286"
> print(paste0("Recall % of KNN is : ", KNN_metrics[3]))
[1] "Recall % of KNN is : 40.1785714285714"
> print(paste0("Precision % of KNN is : ", KNN_metrics[4]))
[1] "Precision % of KNN is : 32.6086956521739"
```

Naïve Bayes Model's classification matrix & related metrics :

```
> NB_model = naiveBayes(Churn ~ ., data = smote_train)
> summary(NB_model)
        Length Class  Mode
apriori  2     table  numeric
tables  14     -none- list
levels   2     -none- character
call     4     -none- call
> NB_Predictions = predict(NB_model, test_data[,1:14], type = 'class')
> ConfMatrix_NB = table(actual=test_data$Churn, predicted= NB_Predictions)
> print(ConfMatrix_NB)
      predicted
actual    1    2
     1 1228  215
     2   96  128
> NB_metrics= metrics(ConfMatrix_NB)
> print(paste0("Naive Bayes Accuracy % is : ", NB_metrics[1]))
[1] "Naive Bayes Accuracy % is : 81.3437312537493"
> print(paste0("False Negative rate % of Naive Bayes is : ", NB_metrics[2]))
[1] "False Negative rate % of Naive Bayes is : 42.8571428571429"
> print(paste0("Recall % of Naive Bayes is : ", NB_metrics[3]))
[1] "Recall % of Naive Bayes is : 57.1428571428571"
> print(paste0("Precision % of Naive Bayes is : ", NB_metrics[4]))
[1] "Precision % of Naive Bayes is : 37.3177842565598"
```

Logistic Regression's classification matrix & related metrics :

```
> logit_predictions= predict(logit_model,newdata= test_data, type="response")
> logit_predictions= ifelse(logit_predictions > 0.5,1,0)
> ConfMatrix_logit = table(actual=test_data$Churn, predicted= logit_predictions)
> print(ConfMatrix_logit)
      predicted
actual    0    1
     1 1229  214
     2   81  143
> logit_metrics= metrics(ConfMatrix_logit)
> print(paste0("Logistic Regression Accuracy % is : ", logit_metrics[1]))
[1] "Logistic Regression Accuracy % is : 82.3035392921416"
> print(paste0("False Negative rate % of Logistic Regression is : ", logit_metrics[2]))
[1] "False Negative rate % of Logistic Regression is : 36.1607142857143"
> print(paste0("Recall % of Logistic Regression is : ", logit_metrics[3]))
[1] "Recall % of Logistic Regression is : 63.8392857142857"
> print(paste0("Precision % of Logistic Regression is : ", logit_metrics[4]))
[1] "Precision % of Logistic Regression is : 40.0560224089636"
```

Summary of logistic Regression model :

```
Call:
glm(formula = Churn ~ ., family = "binomial", data = smote_train)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.80288  -0.70455  -0.07132   0.59566   2.49550

Coefficients:
                             Estimate Std. Error z value Pr(>|z|)
(Intercept)                  -1.93906    1.10448  -1.756 0.079153 .
state2                        0.63281    0.74751   0.847 0.397241
state3                        2.15605    0.76691   2.811 0.004934 **
state4                        0.42795    0.79035   0.541 0.588179
state5                        2.35116    0.77779   3.023 0.002504 **
state6                        1.34845    0.74845   1.802 0.071602 .
state7                        1.15608    0.72305   1.599 0.109844
state8                        1.13291    0.80687   1.404 0.160294
state9                        1.39837    0.73919   1.892 0.058525 .
state10                       2.23823    0.71615   3.125 0.001776 **
state11                       2.83017    0.71325   3.968 7.25e-05 ***
state12                       0.48178    0.86592   0.556 0.577956
state13                       0.31231    0.90513   0.345 0.730057
state14                       1.58257    0.73095   2.165 0.030381 *
state15                       0.92666    0.71985   1.287 0.197989
state16                       0.80421    0.75778   1.061 0.288564
state17                       1.53640    0.71267   2.156 0.031096 *
state18                       1.80768    0.73996   2.443 0.014569 *
state19                       0.79170    0.82030   0.965 0.334474
state20                       2.30754    0.71940   3.208 0.001338 **
state21                       2.08676    0.70517   2.959 0.003084 **
state22                       2.91598    0.68861   4.235 2.29e-05 ***
state23                       1.84387    0.70872   2.602 0.009276 **
state24                       1.70264    0.69347   2.455 0.014079 *
state25                       0.77610    0.75245   1.031 0.302342

state26                       2.19780    0.70339   3.125 0.001780 **
state27                       2.63062    0.69496   3.785 0.000154 ***
state28                       1.49886    0.70735   2.119 0.034092 *
state29                       0.35121    0.81122   0.433 0.665053
state30                       1.09094    0.79845   1.366 0.171842
state31                       1.49612    0.74154   2.018 0.043635 *
state32                       2.36422    0.71289   3.316 0.000912 ***
state33                       1.99018    0.71760   2.773 0.005548 **
state34                       1.87777    0.71175   2.638 0.008334 **
state35                       1.53912    0.70643   2.179 0.029352 *
state36                       1.81103    0.74092   2.444 0.014513 *
state37                       1.35210    0.72277   1.871 0.061383 .
state38                       1.32973    0.73339   1.813 0.069812 .
state39                       1.84232    0.76151   2.419 0.015551 *
state40                       0.69041    0.80945   0.853 0.393694
state41                       2.13291    0.71519   2.982 0.002861 **
state42                       1.28450    0.77271   1.662 0.096448 .
state43                       1.26956    0.74320   1.708 0.087592 .
state44                       2.51643    0.70463   3.571 0.000355 ***
state45                       1.95973    0.71974   2.723 0.006472 **
state46                       0.28668    0.77158   0.372 0.710224
state47                       1.13379    0.74348   1.525 0.127264
state48                       2.23669    0.73014   3.063 0.002188 **
state49                       1.75082    0.70416   2.486 0.012904 *
state50                       1.31888    0.71953   1.833 0.066805 .
state51                       0.94766    0.73498   1.289 0.197272
account.length                0.55320    0.27845   1.987 0.046953 *
international.plan2           2.62900    0.14333  18.342  < 2e-16 ***
voice.mail.plan2              0.02918    0.17452   0.167 0.867196
number.vmail.messages        -0.22503    0.28025  -0.803 0.421996
total.day.minutes             2.36006    0.27949   8.444  < 2e-16 ***

total.day.calls               0.78262    0.29107   2.689 0.007171 **
total.eve.minutes             1.46779    0.29856   4.916 8.82e-07 ***
total.eve.calls               0.29516    0.29181   1.011 0.311788
total.night.minutes           0.41329    0.30440   1.358 0.174553
total.night.calls             0.07525    0.28974   0.260 0.795085
total.intl.minutes            0.99982    0.28601   3.496 0.000473 ***
total.intl.calls             -0.78730    0.25880  -3.042 0.002349 **
number.customer.service.calls2 -3.99911  0.79692  -5.018 5.22e-07 ***
number.customer.service.calls3 -1.02601  0.80943  -1.268 0.204953
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 4017.5  on 2897  degrees of freedom
Residual deviance: 2492.0  on 2833  degrees of freedom
AIC: 2622

Number of Fisher Scoring iterations: 6
```

Appendix – C

Full R code :

```
# clean the environment

rm(list=ls())

#set the working directory
setwd("C:/Users/Navaneeth/Desktop/Edwisor/Project/Employee Churn")

#load the required libraries
x =
c("ggplot2","gridExtra","DMwR","corrgram","C50","caret","randomForest","e1071","class","Matrix
","xgboost")
lapply(x, require, character.only = TRUE)
rm(x)


#************************** Exploratory Data Analysis ****************************#

#read the Train_data csv file by replacing space,comma & NA values with NA.
train=read.csv("Train_data.csv", header=T, na.strings = c(""," ","NA"))
test= read.csv("Test_data.csv", header = T, na.strings = c(""," ","NA"))
whole_data= rbind(train,test)

#To have a look at the structure of the dataset
str(whole_data)

#Convert the area.code variable to factor.
areacode_unique= unique(whole_data$area.code)
cat("The unique values in area.code variable are : ", areacode_unique)
whole_data$area.code= as.factor(whole_data$area.code)

# Bin the number.customer.service.calls variable by driving a new variable called "new"
whole_data$new[whole_data$number.customer.service.calls >= 0 &
whole_data$number.customer.service.calls <= 3]="Low"
whole_data$new[whole_data$number.customer.service.calls > 3 &
whole_data$number.customer.service.calls <= 6]= "Moderate"
whole_data$new[whole_data$number.customer.service.calls>6]= "High"

#drop the number.customer.service.calls variable & rename the "new" variable as
number.customer.service.calls
whole_data$number.customer.service.calls=NULL
names(whole_data)[21]="number.customer.service.calls"
whole_data$number.customer.service.calls=as.factor(whole_data$number.customer.service.calls)

# Move the target variable to the end
whole_data=whole_data[,c(1:19,21,20)]


# Plotting a density plot for train predicotr variables
p1= ggplot(train, aes(x= account.length)) + geom_density()
p2= ggplot(train, aes(x= area.code)) + geom_density()
p3= ggplot(train, aes(x= number.vmail.messages)) + geom_density()
p4= ggplot(train, aes(x= total.day.minutes)) + geom_density()
p5= ggplot(train, aes(x= total.day.calls)) + geom_density()
p6= ggplot(train, aes(x= total.day.charge)) + geom_density()
p7= ggplot(train, aes(x= total.eve.minutes)) + geom_density()
p8= ggplot(train, aes(x= total.eve.calls)) + geom_density()
p9= ggplot(train, aes(x= total.eve.charge)) + geom_density()
p10= ggplot(train, aes(x= total.night.minutes)) + geom_density()
p11= ggplot(train, aes(x= total.night.calls)) + geom_density()
p12= ggplot(train, aes(x= total.night.charge)) + geom_density()
p13= ggplot(train, aes(x= total.intl.minutes)) + geom_density()
p14= ggplot(train, aes(x= total.intl.calls)) + geom_density()
p15= ggplot(train, aes(x= total.intl.charge)) + geom_density()
```

```
p16= ggplot(train, aes(x= number.customer.service.calls)) + geom_density()
gridExtra::grid.arrange(p1, p2, p3, p4, p5, p6, p7, p8, ncol = 2)
gridExtra::grid.arrange(p9, p10, p11, p12, p13, p14, p15, p16, ncol = 2)

#****************** Below are some of the visualizations performed ************************#
#How many customers have churned out
g1 = ggplot(train, aes(x = Churn, fill = Churn)) + geom_bar(stat = 'count')
+geom_label(stat='count',aes(label=..count..), size=5)+ ggtitle("Churning Out Count")
gridExtra::grid.arrange(g1, ncol = 1)

#Checking churn wrt state
g2 = ggplot(train, aes(x = state, fill = Churn, width= 5)) + geom_bar(stat = 'count') +
geom_label(stat='count',aes(label=..count..), size=5) + ggtitle("Customer churn based on
state")
gridExtra::grid.arrange(g2, ncol = 1)

#Churn wrt customer service call
g3 = ggplot(train, aes(x = number.customer.service.calls, fill = Churn, width = 5))+
geom_bar(stat = 'count', position = 'dodge') + geom_label(stat = 'count', aes(label=
..count..))+ggtitle('Customer churn based on number of service calls made')
gridExtra::grid.arrange(g3, ncol = 1)


#Churn wrt to area code
g4 = ggplot(train, aes(x = area.code, fill = Churn, width = 1)) + geom_bar(stat = 'count',
position = 'dodge')+geom_label(stat = 'count', aes(label =..count..)) +ggtitle('Customer Churn
based on their area code')
gridExtra::grid.arrange(g4, ncol = 1)

#churn wrt voice call plan
g5 = ggplot(train, aes(x = voice.mail.plan, fill = Churn)) + geom_bar(stat = 'count', position
= 'dodge') + geom_label(stat = 'count', aes(label = ..count..)) +ggtitle("Number of Churned
customers wrt to voice plan")
gridExtra::grid.arrange(g5, ncol = 1)

#************************** Missing Value Analysis ************************************#

#calcualte the sum of NAs in each colum and store it as a dataframe in missing_val
missing_val = data.frame(apply(whole_data,2,function(x) {sum(is.na(x))}))
#Storing the row names
missing_val$Columns = row.names(missing_val)
# rename the 1st column name
names(missing_val)[1] =  "Missing_percentage"
# calculate the percentage of NAs
missing_val$Missing_percentage = (missing_val$Missing_percentage/nrow(whole_data)) * 100
# get percentage of NAs in decreasing order
missing_val = missing_val[order(-missing_val$Missing_percentage),]
#drop the row names
row.names(missing_val) = NULL
#re-arrange the columns
missing_val = missing_val[,c(2,1)]



#************************ Outlier Analysis *******************************************#

# get indexes of numerical variables
numeric_index = sapply(whole_data,is.numeric) #selecting only numeric
# store the numeric data
numeric_data = whole_data[,numeric_index]
# store the column names of numerical variables
cnames = colnames(numeric_data)

# loop for plotting the box plot for all the numerical variables
 for (i in 1:length(cnames))
 {
   assign(paste0("gn",i), ggplot(aes_string(y = (cnames[i]), x = "Churn"), data =
subset(whole_data))+
            stat_boxplot(geom = "errorbar", width = 0.5) +
            geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,
                         outlier.size=1, notch=FALSE) +
```

```
            theme(legend.position="bottom")+
            labs(y=cnames[i],x="Churn")+
            ggtitle(paste("Box plot of Churn for",cnames[i])))
 }

# Plotting the boxplots together
gridExtra::grid.arrange(gn1,gn2,gn3,gn4,gn5,gn6,ncol=3)
gridExtra::grid.arrange(gn7,gn8,gn9,gn10,gn11,gn12,ncol=3)
gridExtra::grid.arrange(gn13,gn14,ncol=2)


#********************* Outlier Treatment ************************


# optional funcion to check the standard deviation of all the numeric variables before KNN imputaion,
# the value of K that gives lowest standard deviation is selected.
#std= function(x)
#{
#  sd_value=sd(x)
#  sd_value
#}
#sd_data=data.frame(apply(whole_data[,cnames],2,std))
#names(sd_data)="Original SD"

# Replace Outlier values with NAs
for (i in cnames)
{
  outlier_values=whole_data[,i][whole_data[,i] %in% boxplot.stats(whole_data[,i])$out]
  whole_data[,i][whole_data[,i] %in% outlier_values]= NA
}

# Impute NAs with KNN Imputation
whole_data= knnImputation(whole_data,k=9)


#*********************** Feature Selection **************************

# Correlation Plot
corrgram(whole_data[,cnames], order = F,upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")

#OR

# Correlation Plot without upper pie chart
corrgram(whole_data[,cnames], order = F,text.panel=panel.txt, main = "Correlation Plot")



## Chi-squared Test of Independence
factor_index = sapply(whole_data,is.factor)
factor_data = whole_data[,factor_index]
cat_names= colnames(factor_data)

# Loop to print the p values for all the categorical variables
for (i in 1:(ncol(factor_data)-1))
{
  print(names(factor_data)[i])
  print(chisq.test(table(factor_data$Churn,factor_data[,i])),simulate.p.value=TRUE)
}


## Dimension Reduction
whole_data = subset(whole_data, select = -c(total.day.charge, total.eve.charge,total.night.charge,total.intl.charge,area.code,phone.number))


#**************************** Feature Scaling *****************************#

#Updating cnames
cnames= colnames(whole_data[,sapply(whole_data, is.numeric)])
```

```
# Normalization
for ( i in cnames)
{
  whole_data[,i]= ((whole_data[,i]- min(whole_data[,i]))/( max(whole_data[,i])-
min(whole_data[,i])))
}

#to check the max & min values to verify the normalization
range(whole_data[,cnames])


#********************** Data Modelling ****************************#

# Assign levels to catagorical variables
for ( i in 1:ncol(whole_data))
{
  if (class(whole_data[,i])== 'factor')
  {
    print(colnames(whole_data[i]))
    whole_data[,i]= factor(whole_data[,i],labels= (1:length(levels(factor(whole_data[,i])))))
  }
}



#******************** Train - Test Split ****************************#

#We have been given 3333 Train observations and 1667 Test observations which we have combined
# for pre-processing, and now we are going spilt as it was earlier, i.e from the row 3334 till
5000
# the observations come under test.


# Splitting train & test data
train_data= whole_data[1:3333,]  # OR train_data = whole_data[1:nrow(train_data),]
test_data = whole_data[3334:5000,] # OR test_data= whole_data[3334:nrow(whole_data),]
table(train_data$Churn)

#check the train & test percentage which have been given
train_percentage= ((nrow(train_data)*100)/nrow(whole_data))
test_percentage= ((nrow(test_data)*100) / nrow(whole_data))
cat("The train data percentage is = ",train_percentage)
cat ("The test data percentage is = ",test_percentage)

#SMOTE Sampling
set.seed(123)
smote_train = SMOTE(Churn ~.,train_data,perc.over = 200,perc.under =150)
#to check the class of sampled train data
table(smote_train$Churn)

#******************************** Model Development **************************

#************* Decison Tree Classifier Model ****************

set.seed(321)
DT_model = C5.0(Churn ~., smote_train, trials =90, rules = TRUE)
summary(DT_model)
#write(capture.output(summary(DT_model)), "DTModel_Rules.txt")

#Lets predict for test cases
DT_Predictions = predict(DT_model,test_data[,-15], type = "class")

##Evaluate the performance of classification model
ConfMatrix_C50 = table(actual= test_data$Churn, predicted= DT_Predictions)
print(ConfMatrix_C50)
confusionMatrix(ConfMatrix_C50)

# Function to calculate the classification matrix metrics
metrics= function(ConfMatrix)
{
```

```
  TN = ConfMatrix[1,1]
  FP = ConfMatrix[1,2]
  FN = ConfMatrix[2,1]
  TP = ConfMatrix[2,2]

  Accuracy = ((TN + TP) * 100) / (TN + TP + FN + FP)

  FNR = (FN / (FN + TP)) * 100

  Recall= (TP/(TP+FN))*100

  Precision = (TP/(TP+FP))*100

  return(c(Accuracy, FNR, Recall, Precision))

}

metrics_list= metrics(ConfMatrix_C50)
print(paste0("Decision Tree Accuracy is : ", metrics_list[1]))
print(paste0("False Negative rate of Decision Tree is : ", metrics_list[2]))
print(paste0("Recall of Decision Tree is : ", metrics_list[3]))
print(paste0("Precision of Decision Tree is : ", metrics_list[4]))

#save the Decision Tree Model for Future Use
saveRDS(DT_model,"DecisionTree_Rmodel.rds")


#rm('ml','DT_model2','outlier_values','C2')

#******************** Random Forest Classifier Model ***************

#Random Forest model
set.seed(45)
RF_model = randomForest(Churn ~ .,smote_train, importance = TRUE, ntree = 700)
# get the summary of the model
summary(RF_model)

#predict the test cases
RF_Predictions = predict(RF_model, test_data[,-15])
ConfMatrix_RF = table(actual=test_data$Churn, predicted= RF_Predictions)
print(ConfMatrix_RF)
confusionMatrix(ConfMatrix_RF)

#call the function to get the metrics
RF_metrics= metrics(ConfMatrix_RF)
print(paste0("Random Forest Accuracy % is : ", RF_metrics[1]))
print(paste0("False Negative rate % of Random Forest is : ", RF_metrics[2]))
print(paste0("Recall % of Random Forest is : ", RF_metrics[3]))
print(paste0("Precision % of Random forest is : ", RF_metrics[4]))

#Save the RF Model
saveRDS(RF_model,"RandomForest_Rmodel.rds")

#rm('RF_model','RF_Predictions')

#************************* KNN Algorithm **********************

#KNN Model
set.seed(12)
KNN_Predictions = knn(smote_train[, 1:14], test_data[, 1:14], smote_train$Churn, k = 3)
ConfMatrix_KNN = table(actual=test_data$Churn, predicted= KNN_Predictions)
print(ConfMatrix_KNN)

#call the function to get the metrics
KNN_metrics= metrics(ConfMatrix_KNN)
print(paste0("KNN Accuracy % is : ", KNN_metrics[1]))
print(paste0("False Negative rate % of KNN is : ", KNN_metrics[2]))
print(paste0("Recall % of KNN is : ", KNN_metrics[3]))
print(paste0("Precision % of KNN is : ", KNN_metrics[4]))
```

```
#rm('KNN_Predictions')

#************************* Naive Bayes Model **************************

#Naive Bayes Model
NB_model = naiveBayes(Churn ~ ., data = smote_train)
summary(NB_model)
NB_Predictions = predict(NB_model, test_data[,1:14], type = 'class')
ConfMatrix_NB = table(actual=test_data$Churn, predicted= NB_Predictions)
print(ConfMatrix_NB)

#call the function to get the metrics
NB_metrics= metrics(ConfMatrix_NB)
print(paste0("Naive Bayes Accuracy % is : ", NB_metrics[1]))
print(paste0("False Negative rate % of Naive Bayes is : ", NB_metrics[2]))
print(paste0("Recall % of Naive Bayes is : ", NB_metrics[3]))
print(paste0("Precision % of Naive Bayes is : ", NB_metrics[4]))

#Save the Naive Bayes Model
saveRDS(NB_model,"NaiveBayes_Rmodel.rds")

#********************** Logistic Regression Model **********************

#Logistic Regression model
logit_model= glm(Churn ~., data= smote_train, family = "binomial")
summary(logit_model)
logit_predictions= predict(logit_model,newdata= test_data, type="response")

#defiine the p value threshold
logit_predictions= ifelse(logit_predictions > 0.5,1,0)
ConfMatrix_logit = table(actual=test_data$Churn, predicted= logit_predictions)
print(ConfMatrix_logit)

#call the function to get the metrics
logit_metrics= metrics(ConfMatrix_logit)
print(paste0("Logistic Regression Accuracy % is : ", logit_metrics[1]))
print(paste0("False Negative rate % of Logistic Regression is : ", logit_metrics[2]))
print(paste0("Recall % of Logistic Regression is : ", logit_metrics[3]))
print(paste0("Precision % of Logistic Regression is : ", logit_metrics[4]))

#Save the Logistic Regression Model
saveRDS(logit_model,"LogisticRegression_Rmodel.rds")


#************************ Optionally Tried ***************************

#****************** XgBoost******************************#

#Build Sparse Matrix for train & test data which is required for XgBoost model input
trainm= sparse.model.matrix(Churn ~., data = smote_train)[,-1]
train_label = as.integer(as.character(smote_train[,"Churn"]))-1
train_matrix = xgb.DMatrix(data=as.matrix(trainm), label= train_label)

testm = sparse.model.matrix(Churn ~. , data = test_data)[,-1]
test_label= as.integer(as.character(test_data[,"Churn"]))-1
test_matrix= xgb.DMatrix(data= as.matrix(testm), label = test_label)

#XgBoost Model
xg_model= xgboost(data = train_matrix,label=train_label,max.depth=10,eta=0.2,nrounds
=10,objective="binary:logistic",eval_metric="auc",verbose = 1)

#Predictions
pred= predict(xg_model,test_matrix)

#If probability value > 0.5 we are assiging 1 else 0
test_pred <- as.numeric(pred > 0.5) #or ifelse(pred<0.5, 0, 1)

#checking error rate of the predictions
err <- mean(as.numeric(pred > 0.5) != test_label)
print(paste("test-error=", err))
```

```
#Getting Important parameters as per XGBoost model evaluation
importance_matrix <- xgb.importance(model = xg_model)
print(importance_matrix)
#plot the Important parameters
xgb.plot.importance(importance_matrix = importance_matrix)

#Building the confusion matrix
cm_xgb = table(actual=test_label, predicted= test_pred)
print(cm_xgb)
confusionMatrix(cm_xgb)

#accuracy =89.14%
#FNR= 22.32%
#Recall = 77.67%
#Precision = 57.04%

#Saving the XgBoost model
saveRDS(xg_model,"XgBoost_Rmodel.rds")


#***************************  END Of The File  ***********************************
```

Appendix – D

Full Python Code

```
#Load the pre-required libraries
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import chi2_contingency
import seaborn as sns
from fancyimpute import KNN

#set the working directory
os.chdir("C:/Users/Navaneeth/Desktop/Edwisor/Project/Employee Churn")
# Load the train & test data
train=pd.read_csv("Train_data.csv")
print('The shape of train data is : ',train.shape)
test=pd.read_csv("Test_data.csv")
print("The shape of test data is : ",test.shape)
#Combine or row bind the train & test data
whole_data=train.append(test).reset_index(drop=True)
print("The shape of whole data is : ",whole_data.shape)
#set display option to display all the columns in a dataframe
pd.set_option('display.max_columns', 30)
#display first 10 rows of whole data
whole_data.head(10)

#Check for the variables' datatypes and other info
whole_data.info()

#check unique values in area code and change it to categorical
print("The unique values present in area code are :",whole_data['area code'].unique())
whole_data['area code']=whole_data['area code'].astype('object')

#Convert 'number customer service calls' variable to bins
whole_data['number customer service calls']= np.where((whole_data['number customer service
calls'] >=0) & (whole_data['number customer service calls']
<=3),'Low',np.where((whole_data['number customer service calls'] >3) & (whole_data['number
customer service calls'] <=6), 'Moderate','High'))
whole_data['number customer service calls']=whole_data['number customer service
calls'].astype('object')
```

```
#Check the count of unique values in 'number customer service calls'
whole_data['number customer service calls'].value_counts()

# Density plots 0f numeric predictor variables
names=['account length','number vmail messages','total day minutes','total day calls','total
day charge','total eve minutes',
      'total eve calls','total eve charge','total night minutes','total night calls','total
night charge','total intl minutes',
      'total intl calls','total intl charge']
for i in names :
    pd.DataFrame(train[i]).plot(kind='density')

# Plot of Number of voicemail messages by Class

plt.figure(figsize = (10,15))
train.hist('number vmail messages', by = 'Churn')
plt.ylabel('Count', fontsize = 20)
plt.xlabel('voice mail messages', fontsize = 20)
#plt.savefig('voicemail.png')

# Plot of Total Intl calls by Class

#plt.figure(figsize = (10,15))
train.hist('total intl calls', by = 'Churn')
plt.ylabel('Count', fontsize = 20)
plt.xlabel('Intl calls', fontsize = 20)
#plt.savefig('total intl calls.png')

# Plot of Number of customer service calls by Class

plt.figure(figsize = (10,15))
train.hist('number customer service calls', by = 'Churn')
plt.ylabel('Count', fontsize = 20)
plt.xlabel('Customer service calls', fontsize = 20)
#plt.savefig('Customerservivecalls.png')

# Plot of States

plt.figure(figsize = (15,10))
sns.countplot('state', data= train)
plt.xlabel('State', fontsize = 20)
plt.ylabel('Count', fontsize = 20)
#plt.savefig('state.png')

#Dispaly the percentage of train and test data
train_row= train.shape[0]
test_row=test.shape[0]
total_row= train_row+test_row
print("The train data has",train_row,"rows","with % of train data being
",((train_row/total_row)*100))
print("The test data has",test_row,"rows","with % of test data being
",((test_row/total_row)*100))

#Check the presence of class imbalance in the train data
train_class= train['Churn'] .value_counts()
print(train_class)

#Store the sum of null values as a dataframe in a new variable
missing_val = pd.DataFrame(whole_data.isnull().sum())
#Reset index
missing_val = missing_val.reset_index()
#Rename variables
missing_val = missing_val.rename(columns = {'index': 'Variables', 0: 'Missing_percentage'})
#Calculate percentage
missing_val['Missing_percentage'] = (missing_val['Missing_percentage']/len(whole_data))*100
#descending order
missing_val = missing_val.sort_values('Missing_percentage', ascending =
False).reset_index(drop = True)
print(missing_val)

# Segregate the numeric and categorical variables through loop
```

```
cnames=[]
cat_names=[]
for i in range(0,whole_data.shape[1]):
    if(whole_data.iloc[:,i].dtypes == 'object'):
        #print('variable',whole_data.columns[i],'is object datatype')
        cat_names.append(whole_data.columns[i])
    else :
        if(whole_data.iloc[:,i].dtypes == 'int64' or whole_data.iloc[:,i].dtypes =='float64'):
            #print('variable',whole_data.columns[i],'is numeric datatype')
            cnames.append(whole_data.columns[i])


print("The numeric variables are :",cnames)
print("The categorical variables are :",cat_names)

# Plotting boxplot to visulaize outliers for all the numeric variables
%matplotlib inline
for i in cnames :
    plt.figure()
    plt.clf()
    plt.boxplot(whole_data[i])
    plt.title(i)
    #plt.savefig(i)
    plt.show()

# Replacing Outliers with NAs
for i in cnames:
    print(i)
    q75,q25=np.percentile(whole_data.loc[:,i],[75,25])
    iqr= q75-q25

    minimum = q25-(1.5*iqr)
    maximum = q75+(1.5*iqr)
    print('Minimum value is',minimum)
    print('Maximum value is',maximum)

    whole_data.loc[:,i][whole_data.loc[:,i] < minimum]= np.nan
    whole_data.loc[:,i][whole_data.loc[:,i] > maximum]= np.nan

#converting the categorical variables with its corresponding cat codes
for i in cat_names:
    print(i)
    if(whole_data.loc[:,i].dtypes=='object'):
        whole_data.loc[:,i]=pd.Categorical(whole_data.loc[:,i])
        whole_data.loc[:,i]=whole_data.loc[:,i].cat.codes
        whole_data.loc[:,i]=whole_data.loc[:,i].astype('object')

    else :
        print("check the cat_names list")

#We are imputing the NA values with KNN imputaion
whole_data= pd.DataFrame(KNN(k=9).fit_transform(whole_data),columns=whole_data.columns)

#After KNN imputation all the variables will be in 'float' datatype, hence I'm converting all
the object datatypes as it were before.
for i in cat_names:
    whole_data.loc[:,i]=whole_data.loc[:,i].astype('object')

whole_data.dtypes

# Correlation Analysis
#Extract numeric variables dataset for correlation analysis
numeric_data= whole_data.loc[:,cnames]
#Set the width and hieght of the plot
f, ax = plt.subplots(figsize=(7, 5))
#Generate correlation matrix
corr = numeric_data.corr()
#Plot using seaborn library
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.diverging_palette(220, 10,
as_cmap=True),
            square=True, ax=ax)
```

```
#Chisquare test of independence
#removing the last target variable 'Churn' from the cat_names list
cat_names= cat_names[:-1]
print(cat_names)

#loop for chi square test to generate p values for all the categorical varaibles
for i in cat_names:
    print(i)
    chi2, p, dof, ex = chi2_contingency(pd.crosstab(whole_data['Churn'], whole_data[i]))
    print(p)

#Removing the above mentioned variables which are of no use
whole_data = whole_data.drop(['total day charge', 'total eve charge', 'total night charge',
'total intl charge', 'area code','phone number'], axis=1)
print('The new shape of whole_data is :',whole_data.shape)

#Updating the new cnames and cat_name list as we dropped them in the feature selection stage
cat_names=['state','international plan','voice mail plan','number customer service
calls','Churn']
cnames=['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']

#Normalization
for i in cnames:
    print(i)
    whole_data[i] = (whole_data[i] - min(whole_data[i]))/(max(whole_data[i]) -
min(whole_data[i]))

#To visualize the max and min data to verify the normalization
whole_data.describe()

#Splitting the train and test data from the whole data
train_data = whole_data.iloc[0:3333,]
test_data = whole_data.iloc[3333:,].reset_index(drop=True)
print("The shape of train_data is",train_data.shape)
print("The shape of test data is",test_data.shape)

# Visualize the number of imbalance classes in Churn
train_data['Churn'].value_counts()

# Import the SMOTE library and initialize
from imblearn.over_sampling import SMOTE
smote=SMOTE(ratio='minority',random_state=123)
#Split the predictor & target variables
X_train=train_data.drop('Churn',1)
Y_train=train_data['Churn']
#Sample using SMOTE
X_smote, Y_smote = smote.fit_sample(X_train, Y_train)

#Convert the sampled data to dataframe
X_smote = pd.DataFrame(X_smote, columns = ['state', 'account length', 'international
plan','voice mail plan','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'])

Y_smote=pd.DataFrame(Y_smote,columns=['Churn'])

print("After SMOTE sampling the number of obs of value 1 are :",Y_smote.loc[Y_smote.Churn ==
1].shape[0])
print("After SMOTE sampling the number of obs of value 0 are :",Y_smote.loc[Y_smote.Churn ==
0].shape[0])

#Combine the sampled predictor variables with the target variable
sampled_train=pd.concat([X_smote, Y_smote], axis = 1)
sampled_train.shape

#Changing 0 to false & 1 to True as decision tree expects the target class to be in
categorical
```

```
Y_smote['Churn']=Y_smote['Churn'].replace(0, 'False')
Y_smote['Churn']=Y_smote['Churn'].replace(1, 'True')

# Separating test predictor & target variables
test_values=test_data.iloc[:,0:14]
test_label=test_data['Churn']
test_label=test_label.replace(0,'False')
test_label=test_label.replace(1,'True')

#import & load the decision tree
from sklearn import tree
C50_model = tree.DecisionTreeClassifier(criterion='entropy',max_depth =20,random_state =
123).fit(X_smote,Y_smote)

#predict new test cases
C50_Predictions = C50_model.predict(test_values)

#Function to build Classification matrix & return the required metrics
def classmatrix(act_label,predicted) :
    CM = pd.crosstab(act_label, predicted,rownames=['Actual'],colnames=['Predicted'])
    TN = CM.iloc[0,0]
    FN = CM.iloc[1,0]
    TP = CM.iloc[1,1]
    FP = CM.iloc[0,1]
    Accuracy=(((TP+TN)*100)/(TP+TN+FP+FN))
    FNR= (FN*100)/(FN+TP)
    Recall= ((TP/(TP+FN))*100)
    Precision = ((TP/(TP+FP))*100)
    return(Accuracy,FNR,Recall,Precision)

#print the metrics for decision tree classifier
Acc,fnr,recall,precision = classmatrix(test_label,C50_Predictions)
print('Decision Tree Accuracy %: {:.3f}'.format(Acc))
print('Decision Tree FNR %: {:.3f}'.format(fnr))
print('Decision Tree Recall %: {:.3f}'.format(recall))
print('Decision Tree Precision %: {:.3f}'.format(precision))
pd.crosstab(test_label, C50_Predictions,rownames=['Actual'],colnames=['Predicted'])

#Save the decision tree model for future use
from sklearn.externals import joblib
joblib.dump(C50_model, "DecisionTree_Python_Final.sav")

# Laod the random forest library and build the model
from sklearn.ensemble import RandomForestClassifier
RF_model = RandomForestClassifier(n_estimators = 80,random_state =
123).fit(X_smote,Y_smote.values.ravel())

#make predictions
RF_Predictions = RF_model.predict(test_values)

#print the metrics for Random Forest classifier
Acc,fnr,recall,precision = classmatrix(test_label,RF_Predictions)
print('Random Forest Accuracy %: {:.3f}'.format(Acc))
print('Random Forest FNR %: {:.3f}'.format(fnr))
print('Random Forest Recall %: {:.3f}'.format(recall))
print('Random Forest Precision %: {:.3f}'.format(precision))
pd.crosstab(test_label, C50_Predictions,rownames=['Actual'],colnames=['Predicted'])

#KNN implementation
from sklearn.neighbors import KNeighborsClassifier

KNN_model = KNeighborsClassifier(n_neighbors = 3).fit(X_smote,Y_smote.values.ravel())

#predict test cases
KNN_Predictions = KNN_model.predict(test_values)

#print the metrics for KNN classifier
Acc,fnr,recall,precision = classmatrix(test_label,KNN_Predictions)
print('KNN Accuracy %: {:.3f}'.format(Acc))
print('KNN FNR %: {:.3f}'.format(fnr))
print('KNN Recall %: {:.3f}'.format(recall))
```

```
print('KNN Precision %: {:.3f}'.format(precision))
pd.crosstab(test_label, C50_Predictions,rownames=['Actual'],colnames=['Predicted'])

#import the Naive Bayes library and laod the model
from sklearn.naive_bayes import GaussianNB
NB_model = GaussianNB().fit(X_smote,Y_smote.values.ravel())

#predict test cases
NB_Predictions = NB_model.predict(test_values)


#print the metrics for Naive Bayes classifier
Acc,fnr,recall,precision = classmatrix(test_label,NB_Predictions)
print('Naive Bayes Accuracy %: {:.3f}'.format(Acc))
print('Naive Bayes FNR %: {:.3f}'.format(fnr))
print('Naive Bayes Recall %: {:.3f}'.format(recall))
print('Naive Bayes Precision %: {:.3f}'.format(precision))
pd.crosstab(test_label, C50_Predictions,rownames=['Actual'],colnames=['Predicted'])

#Loading the data post feature selection(as we have already analyzed them in the above
section)
whole_data_logit = whole_data.drop(['total day charge', 'total eve charge', 'total night
charge', 'total intl charge', 'area code','phone number'], axis=1)

new_cnames=['total intl calls','total intl minutes','number vmail messages','total eve
minutes','total night calls','total night minutes','total day calls','total day
minutes','total eve calls','account length']

numeric_data=whole_data_logit[['total intl calls','total intl minutes','number vmail
messages','total eve minutes','total night calls','total night minutes','total day
calls','total day minutes','total eve calls','account length']]
categorical_data=whole_data_logit[['state','international plan','voice mail plan','number
customer service calls','Churn']]

#Replace Outliers with NAs
for i in new_cnames:
    print(i)
    q75,q25=np.percentile(numeric_data.loc[:,i],[75,25])
    iqr= q75-q25

    minimum = q25-(1.5*iqr)
    maximum = q75+(1.5*iqr)
    print('Minimum value is',minimum)
    print('Maximum value is',maximum)

    numeric_data.loc[:,i][numeric_data.loc[:,i] < minimum]= np.nan
    numeric_data.loc[:,i][numeric_data.loc[:,i] > maximum]= np.nan

# Impute NAs with KNN Imputation
numeric_data= pd.DataFrame(KNN(k=9).fit_transform(numeric_data),columns=numeric_data.columns)

#Normalization
for i in new_cnames:
    print(i)
    numeric_data[i] = (numeric_data[i] - min(numeric_data[i]))/(max(numeric_data[i]) -
min(numeric_data[i]))

#Loading numeric dataset to a new dataset
logit_data=numeric_data

#Replace categorical values with corresponding dummy values
for i in ['state','international plan','voice mail plan','number customer service
calls','Churn']:
    print(i)
    temp = pd.get_dummies(categorical_data[i], prefix = i)
    logit_data = logit_data.join(temp)

# Visualize the top 4 rows of logit_data
logit_data.head(4)

#Drop Churn_ False. variable as it is not required
```

```
logit_data.drop(['Churn_ False.'],axis=1,inplace=True)
# Change column name Churn_ True. to Churn
logit_data.rename(columns={'Churn_ True.':'Churn'},inplace=True)
logit_data.head(5)

#Separate the train & test dataset
logit_train = logit_data.iloc[0:3333,]
logit_test = logit_data.iloc[3333:,].reset_index(drop=True)
print("The shape of train_data is",logit_train.shape)
print("The shape of test data is",logit_test.shape)

# SMOTE Sampling
from imblearn.over_sampling import SMOTE
smote=SMOTE(ratio='minority',random_state=123)
X_train_logit=logit_train.drop('Churn',1)
Y_train_logit=logit_train['Churn']
X_smote_logit, Y_smote_logit = smote.fit_sample(X_train_logit, Y_train_logit)

#Convert X_smote_logit & Y_smote_logit to dataframes
X_smote_df = pd.DataFrame(X_smote_logit, columns = ['total intl calls', 'total intl minutes',
'number vmail messages','total eve minutes', 'total night calls', 'total night minutes','total
day calls', 'total day minutes', 'total eve calls','account length', 'state_AK', 'state_AL',
'state_AR', 'state_AZ','state_CA', 'state_CO', 'state_CT', 'state_DC', 'state_DE',
'state_FL','state_GA', 'state_HI', 'state_IA', 'state_ID', 'state_IL', 'state_IN','state_KS',
'state_KY', 'state_LA', 'state_MA', 'state_MD', 'state_ME','state_MI', 'state_MN', 'state_MO',
'state_MS', 'state_MT', 'state_NC','state_ND', 'state_NE', 'state_NH', 'state_NJ', 'state_NM',
'state_NV','state_NY', 'state_OH', 'state_OK', 'state_OR', 'state_PA', 'state_RI','state_SC',
'state_SD', 'state_TN', 'state_TX', 'state_UT', 'state_VA','state_VT', 'state_WA', 'state_WI',
'state_WV', 'state_WY','international plan_ no', 'international plan_ yes','voice mail plan_
no', 'voice mail plan_ yes','number customer service calls_0', 'number customer service
calls_1','number customer service calls_2', 'number customer service calls_3','number customer
service calls_4', 'number customer service calls_5','number customer service calls_6', 'number
customer service calls_7','number customer service calls_8', 'number customer service
calls_9'])
Y_smote_df=pd.DataFrame(Y_smote_logit,columns=['Churn'])
# Check the proportion of 0 & 1 values in Churn varaible
Y_smote_df['Churn'].value_counts()

#Import & load the Logistic Regression Model
from sklearn.linear_model import LogisticRegression
logmodel=LogisticRegression(random_state=123)
logmodel.fit(X_smote_df,Y_smote_df.values.ravel())

# Separating Test predictor & target variables
logit_test_data=logit_test.iloc[:,0:75]
logit_test_label=logit_test['Churn']

#Predict the test cases
predictions= logmodel.predict(logit_test_data)

#Build the Confusion matrix
conf_matrix =
pd.crosstab(logit_test_label,predictions,rownames=['Actual'],colnames=['Predicted'])
TN = conf_matrix.iloc[0,0]
FN = conf_matrix.iloc[1,0]
TP = conf_matrix.iloc[1,1]
FP = conf_matrix.iloc[0,1]
Accuracy=(((TP+TN)*100)/(TP+TN+FP+FN))
FNR= (FN*100)/(FN+TP)
Recall= ((TP/(TP+FN))*100)
Precision = ((TP/(TP+FP))*100)

print('Logistic Regression Accuracy %: {:.3f}'.format(Accuracy))
print('Logistic Regression FNR %: {:.3f}'.format(FNR))
print('Logistic Regression Recall %: {:.3f}'.format(Recall))
print('Logistic Regression Precision %: {:.3f}'.format(Precision))
pd.crosstab(logit_test_label,predictions,rownames=['Actual'],colnames=['Predicted'])
```

References :

https://www.theanalysisfactor.com/missing-data-mechanism/

https://datavizcatalogue.com/methods/density_plot.html

https://www.sharpsightlabs.com/blog/density-plot-in-r/

http://www.physics.csbsju.edu/stats/box2.html

https://nelsontouchconsulting.wordpress.com/2011/01/17/deeper-into-box-plots/

https://medium.com/greyatom/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b

https://stackoverflow.com/questions/26553526/how-to-add-frequency-count-labels-to-the-bars-in-a-bar-graph-using-ggplot2

https://xgboost.readthedocs.io/en/latest/R-package/xgboostPresentation.html