Telecom Customer Analysis

-Vompolu Sai Tanuj

G1 PGP-BABI

Table of contents:

1) Project Objective:

Due to the rise in telecom industries, **Churning** of customers has become a burning problem for the telecom industries. Such is the case of the one of the telecom industries whose **post-paid** customers with a **contract** are to be analysed to make predictions on whether if we choose a customer, would that **customer Churn** or **not**. And while doing so, also giving some **useful insights** using the predictions given by the model.

2) Exploration of the dataset:

The dataset given to us is named as **"Cellphone.xlsx".** This dataset, which is in the form of an **Excel Spreadsheet,** contains all the **information of the past customers** whether they have churned or not along with the data of some of the parameters which will help in **Model Building** and **predictions.**

a) Invoking of necessary packages to be used in the R Code:

Before importing the dataset into the **R Environment**, it is necessary to **invoke** all the necessary **packages** that will be used in the **R** code further down the line. The packages, if not present in the system, can be installed using the function **install.packages()** and the library can be called onto the specific **R Script** using the function **library()**. The following are the libraries which will be significant in the **R Code:**

- **readxl** – This library is used to import the dataset which is in **.xlsx** format

```
install.packages("readxl")
library(readxl)
```

- **class** – This library is useful for **KNN** Model.

```
install.packages("class")
library(class)
```

- **psych** – This library is useful for plotting a **Correlation Plot**

```
install.packages("psych")
library(psych)
```

- **ggplot2** – This library will help in building plots and gives many modifications for the same.

```
install.packages("ggplot2")
library(ggplot2)
```

- **caTools** – This library is useful to split the data into training and testing.

```
install.packages("caTools")
library(caTools)
```

- **caret** – This library is useful to get the **confusion matrix** of the required models.

```
install.packages("caret")
library(caret)
```

- **ROCR** – This library is useful for plotting the **ROC** and **AUC** curves for the required models.

```
install.packages("ROCR")
library(ROCR)
```

- **Hmisc** – This library is useful in the **Uni-variate Analysis** for the plotting of the **histograms.**

```
install.packages("Hmisc")
library(Hmisc)
```

- **Ineq –** This library is useful to get the **KS and Gini** index for the required models.

```
install.packages("ineq")
library(ineq)
```

- **E1071 –** This library is used to build **Naïve Bayes.**

```
install.packages("e1071")
library(e1071)
```

- **InformationValue –** This library is useful to get **Concordance Ratios**

```
install.packages("InformationValue")
library(InformationValue)
```

b) Setting the working directory and importing the dataset:

The working directory in the **R Console** must be set to the directory where the XLSX file exists. The working directory can be changed using the function **setwd().** To get the current directory of the R Console, the function **getwd()** can be used.

```
> getwd()
[1] "C:/R programs great lakes/P Model/project"
```

After setting the working directory, we must now import the dataset from the **xlsx file** to the R console using the function **read_xlsx().** The dataset assigned to this project is assigned to a data frame with the name **cell** and the same will be used to call the dataset further when required in the **R Console.** The function **View()** can be used to view the contents of the **data-frame** created.

```
#### Importing the dataset and creation of the dataframe #####
cell = read_xlsx("Cellphone1.xlsx")
View(cell)
```

c) Identification of different variables:

To able to do **uni-variate** and **bi-variate** analysis, we must first be able to understand all the **variables** in the **data-frame** and get a basic idea on the data-frame before the analysis. The following functions can be used to achieve the above:

- **dim() -** This function can be used to get the **dimensions** of the data-frame.

```
> dim(cell)
[1] 3333   11
```

- **str()** – This function can be used to the get the **classes** of all the variables available along with the **values** in the variable.

```
> str(cell)
Classes 'tbl_df', 'tbl' and 'data.frame':       3333 obs. of  11 variables:
 $ Churn          : num  0 0 0 0 0 0 0 0 0 0 ...
 $ AccountWeeks   : num  128 107 137 84 75 118 121 147 117 141 ...
 $ ContractRenewal: num  1 1 1 0 0 0 1 0 1 0 ...
 $ DataPlan       : num  1 1 0 0 0 0 1 0 0 1 ...
 $ DataUsage      : num  2.7 3.7 0 0 0 0 2.03 0 0.19 3.02 ...
 $ CustServCalls  : num  1 1 0 2 3 0 3 0 1 0 ...
 $ DayMins        : num  265 162 243 299 167 ...
 $ DayCalls       : num  110 123 114 71 113 98 88 79 97 84 ...
 $ MonthlyCharge  : num  89 82 52 57 41 57 87.3 36 63.9 93.2 ...
 $ OverageFee     : num  9.87 9.78 6.06 3.1 7.42 ...
 $ RoamMins       : num  10 13.7 12.2 6.6 10.1 6.3 7.5 7.1 8.7 11.2 ...
```

- **head()** – This function is used to give the top first few values of each of the variable. An additional argument can be passed to dictate the number of values to be shown.

```
> head(cell)
# A tibble: 6 x 11
  Churn AccountWeeks ContractRenewal DataPlan DataUsage CustServCalls DayMins DayCalls MonthlyCharge OverageFee RoamMins
  <dbl>        <dbl>           <dbl>    <dbl>     <dbl>         <dbl>   <dbl>    <dbl>         <dbl>      <dbl>    <dbl>
1     0          128               1        1       2.7             1    265.      110            89       9.87     10
2     0          107               1        1       3.7             1    162.      123            82       9.78     13.7
3     0          137               1        0       0               0    243.      114            52       6.06     12.2
4     0           84               0        0       0               2    299.       71            57       3.1       6.6
5     0           75               0        0       0               3    167.      113            41       7.42     10.1
6     0          118               0        0       0               0    223.       98            57      11.0       6.3
```

- **tail()** – This function is used to give the **bottom last few** values of each of the variable. An additional argument can be passed down to dictate the number of values to be shown.

```
> tail(cell)
# A tibble: 6 x 11
  Churn AccountWeeks ContractRenewal DataPlan DataUsage CustServCalls DayMins DayCalls MonthlyCharge OverageFee RoamMins
  <dbl>        <dbl>           <dbl>    <dbl>     <dbl>         <dbl>   <dbl>    <dbl>         <dbl>      <dbl>    <dbl>
1     0           79               1        0      0                2    135.       98          40         9.49     11.8
2     0          192               1        1      2.67             2    156.       77          71.7      10.8       9.9
3     0           68               1        0      0.34             3    231.       57          56.4       7.67      9.6
4     0           28               1        0      0                2    181.      109          56        14.4      14.1
5     0          184               0        0      0                2    214.      105          50         7.98      5
6     0           74               1        1      3.7              0    234.      113         100        13.3      13.7
```

- **summary()** – This function gives us an basic idea on values on the data-frame by performing the **basic statistics** on it.

```
> summary(cell)
     Churn         AccountWeeks    ContractRenewal     DataPlan         DataUsage      CustServCalls       DayMins         DayCalls     MonthlyCharge
 Min.   :0.0000   Min.   :  1.0   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.000   Min.   :  0.0   Min.   :  0.0   Min.   : 14.00
 1st Qu.:0.0000   1st Qu.: 74.0   1st Qu.:1.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:1.000   1st Qu.:143.7   1st Qu.: 87.0   1st Qu.: 45.00
 Median :0.0000   Median :101.0   Median :1.0000   Median :0.0000   Median :0.0000   Median :1.000   Median :179.4   Median :101.0   Median : 53.50
 Mean   :0.1449   Mean   :101.1   Mean   :0.9031   Mean   :0.2766   Mean   :0.8165   Mean   :1.563   Mean   :179.8   Mean   :100.4   Mean   : 56.31
 3rd Qu.:0.0000   3rd Qu.:127.0   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.7800   3rd Qu.:2.000   3rd Qu.:216.4   3rd Qu.:114.0   3rd Qu.: 66.20
 Max.   :1.0000   Max.   :243.0   Max.   :1.0000   Max.   :1.0000   Max.   :5.4000   Max.   :9.000   Max.   :350.8   Max.   :165.0   Max.   :111.30
   OverageFee       RoamMins
 Min.   : 0.00   Min.   : 0.00
 1st Qu.: 8.33   1st Qu.: 8.50
 Median :10.07   Median :10.30
 Mean   :10.05   Mean   :10.24
 3rd Qu.:11.77   3rd Qu.:12.10
 Max.   :18.19   Max.   :20.00
```

Inferences:

1) The data-frame consists of **3333 rows** and **11 columns.**
2) We can see that all the 11 **variables** in the data-frame are **numeric.**
3) We can see that the variables **Churn, DataPlan** and **ContractRenewal** contains only the values **0** and **1**.
4) From the outputs of the **head()** and **tail()** functions, we can see that there is no proper order to the data and the data has not been arranged with regard to any of the variables in the data-frame.
5) Except for **MonthlyCharge** and **AccountsWeek**, the rest of the variables all have their minimum value as **0.**
6) We can see that there are no **discrepancies** in any of the variables given in the data-frame.

7) From the **AccountsWeeks** variable, we can see that **oldest customer** from this data had an active account for **4.6 years.**

8) The **shortest time** a **customer** had an **active account** was **1 Week.**

9) The **highest DataUsage** of the customer seems to be around **5GB** per month.

10) The **Highest average monthly bill** stands at 111 while the **lowest average monthly bill** stands at 14.

11) The **Highest average** of **Daytime Calls** is 165.

The actual description of the each variables is given to us is as below:

Variables	
Churn	*1 if customer cancelled service, 0 if not*
AccountWeeks	*number of weeks customer has had active account*
ContractRenewal	*1 if customer recently renewed contract, 0 if not*
DataPlan	*1 if customer has data plan, 0 if not*
DataUsage	*gigabytes of monthly data usage*
CustServCalls	*number of calls into customer service*
DayMins	*average daytime minutes per month*
DayCalls	*average number of daytime calls*
MonthlyCharge	*average monthly bill*
OverageFee	*largest overage fee in last 12 months*

As we can see, the three of the variables need to be converted into **Categorical** variable. We must also note that the variable which is of significance to us is **Churn.** It is the **Dependent variable** in our analysis.

<u>Conversion of necessary variables into categorical variables:</u>

Before we can go ahead with analysis, we need to convert some of the numerical variables into categorical variables. There are **three variables** which must be converted into **categorical variables.** They are **Churn,DataPlan** and **ContractRenewal.** This can be achieved using the function **as.factor()**.

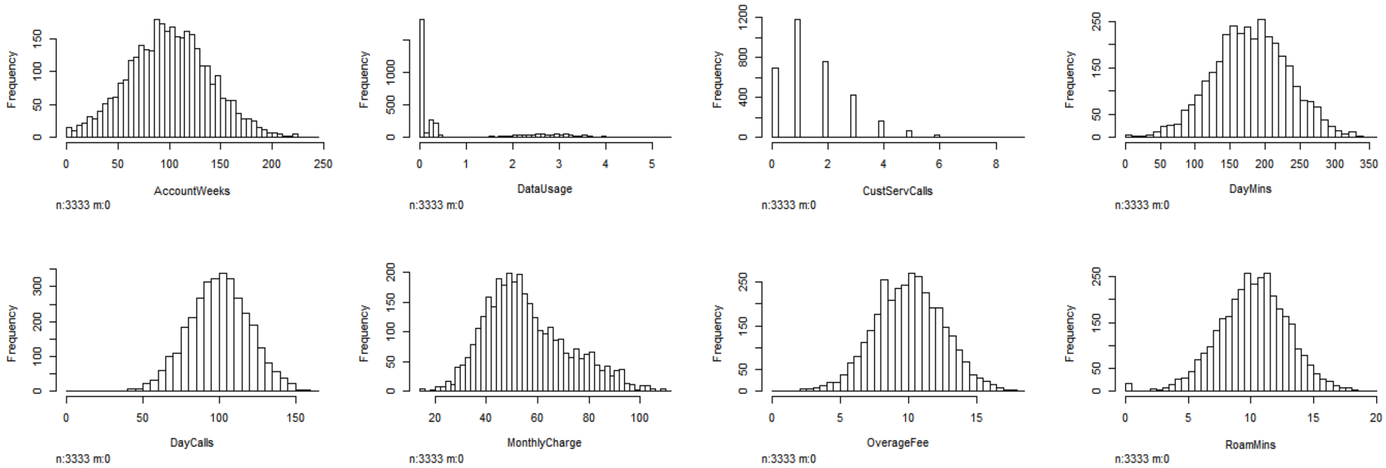
```
### Conversion of numericals to factors ####
cell$Churn = as.factor(cell$Churn)
cell$ContractRenewal = as.factor(cell$ContractRenewal)
cell$DataPlan = as.factor(cell$DataPlan)
```

d.) Uni – Variate Analysis:

In the data-frame, we have two types of variables namely **Categorical** and **numerical.** The Numerical variables can be analysed plotting **Histograms.** These can be constructed by using **hist.data.frame()** function. The **Categorical variable** can be analysed by using the **frequency table** and **Bar Plots.** The **frequency table** can be plotted using the function **table()** and the **barplot** can be plotted using the function **plot()**.

i. *Numerical Independent Variables:*

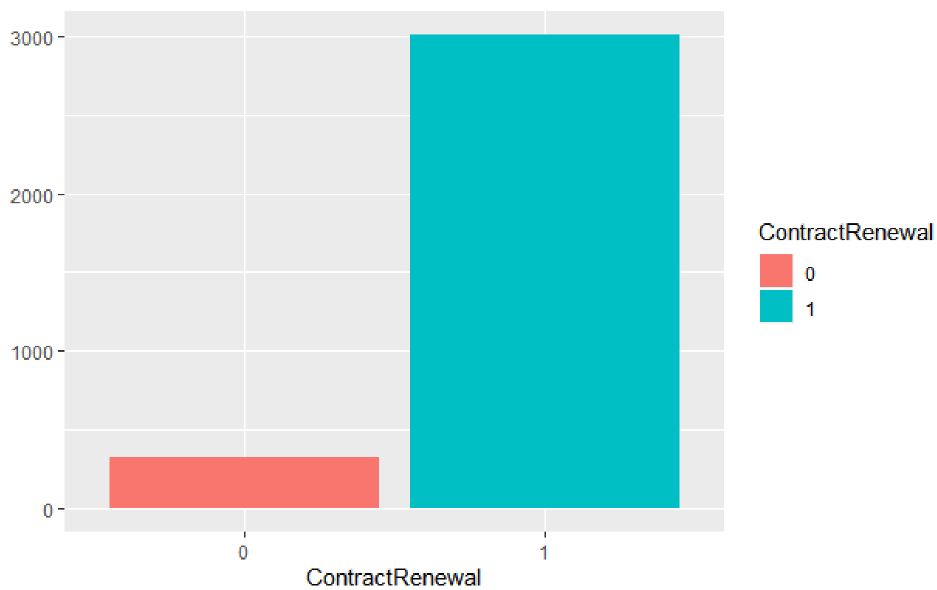
```
### Uni-Variate Analysis ####
### Analysis of Independent Numerical variables ###
hist.data.frame(cell)
```



ii. ***Categorical Independent Variables:***

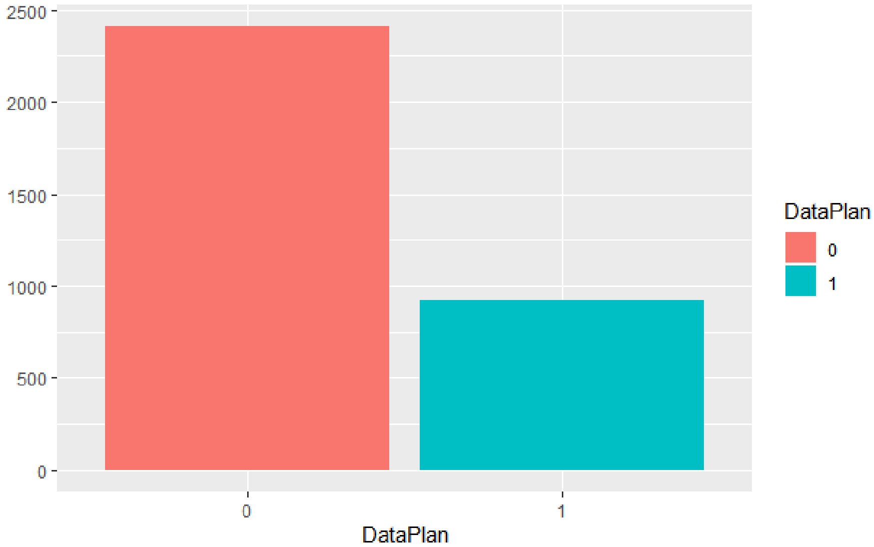
```
> table(cell$ContractRenewal)

   0    1
 323 3010

> prop.table(table(cell$ContractRenewal))*100

        0         1
 9.690969 90.309031
```

```
qplot(ContractRenewal,fill = ContractRenewal,data = cell)
```



```
> table(cell$DataPlan)

   0    1 
2411  922 

> prop.table(table(cell$DataPlan))*100

       0        1 
72.33723 27.66277 
```

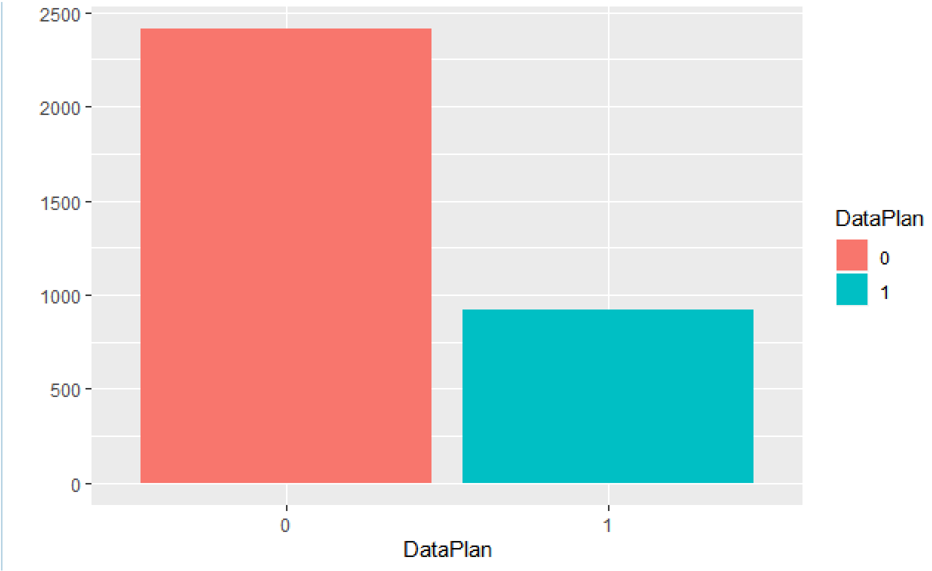
```
qplot(DataPlan,fill = DataPlan,data = cell)
```



iii. *Dependent Variable:*

```
### Analysis of Dependent variables ###
table(cell$Churn)
prop.table(table(cell$Churn))*100
qplot(Churn,fill = Churn,data = cell)
```

```
> table(cell$Churn)

   0    1 
2850  483 

> prop.table(table(cell$Churn))*100

       0        1 
85.50855 14.49145 
```



Inferences:

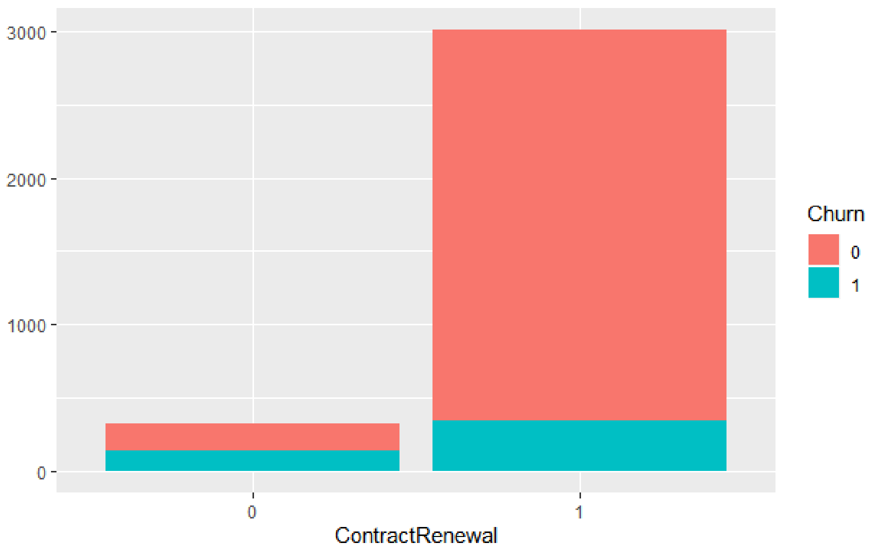
1) The variable **Datausage** doesn't have a **normal distribution.**
2) The variables **CustservCalls** and **MonthlyCharge** seem to be skewed normally with **MonthlyCharge** being very highly skewed.
3) We can see that around **90% (3010)** have gone for **Contract Renewal.**
4) We can see that only **27% (922)** of the customers have an active data plan.
5) From the above data, we can see that **many (85%)** of the customers have **churned,** while only the few remaining customers have **not churned.**

e.) Bi – Variate Analysis:

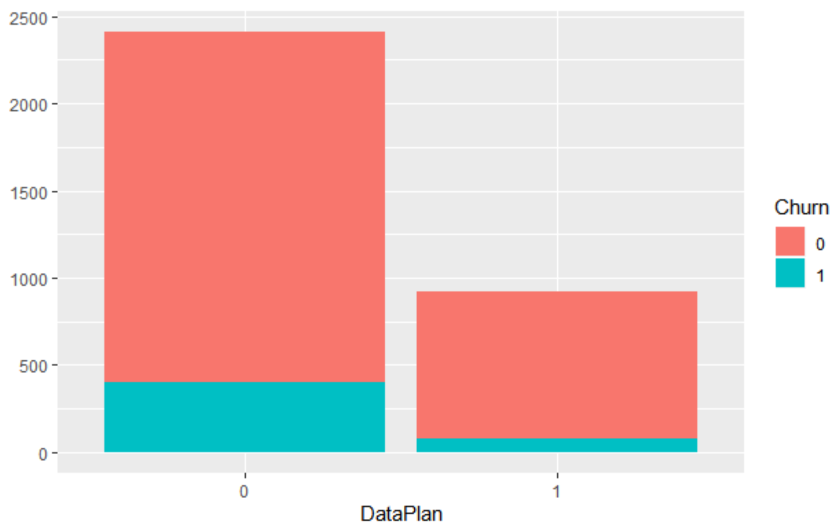
The **Bi-Variate Analysis** can be done with different combination of variables,(Categorical vs. Categorical, Categorical vs Numerical and Numerical vs Numerical) by using the **qplot()** function from the **ggplot2** library and changing the arguments accordingly to get the required analysis.

i. ***Dependent vs Independent Categorical Variables:***

```
qplot(ContractRenewal,fill = Churn,data = cell)
```

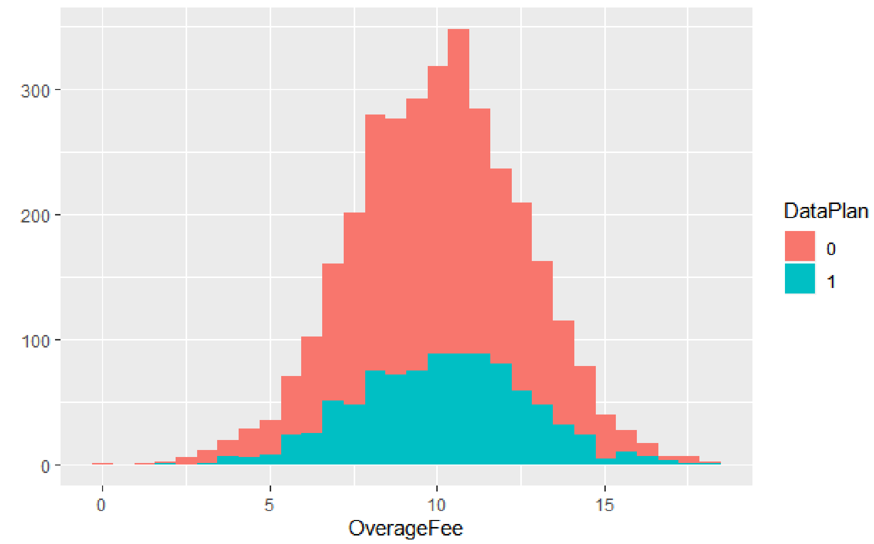



```
qplot(DataPlan,fill = Churn,data = cell,geom = "bar")
```

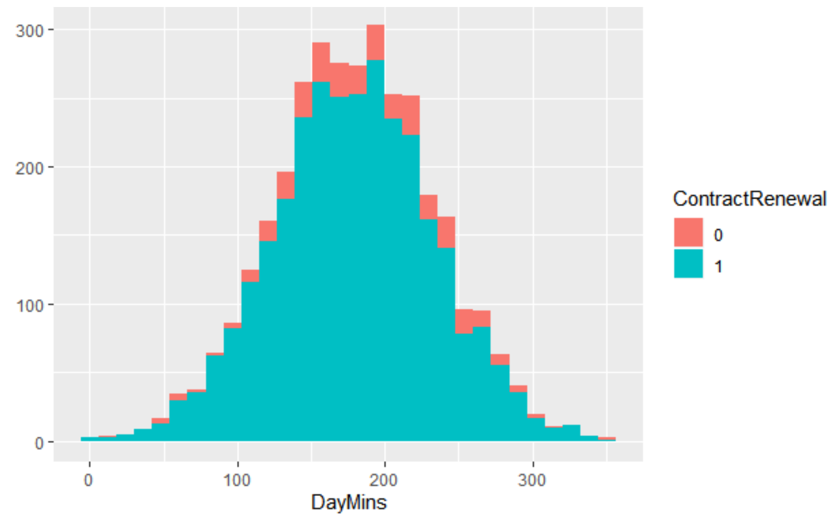


ii. ***Independent Variables vs. Independent Variables:***

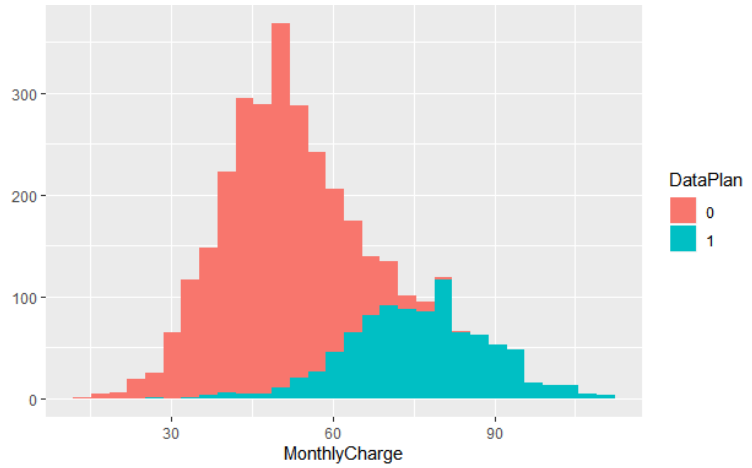
```
qplot(OverageFee,fill = DataPlan,data = cell)
```




```
qplot(DayMins,fill = ContractRenewal,data = cell)
```

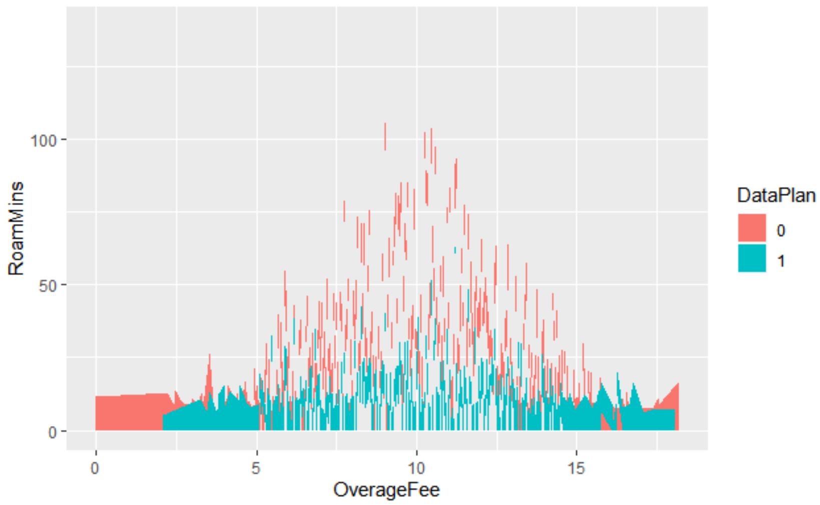



```
qplot(MonthlyCharge,fill = DataPlan,data = cell)
```

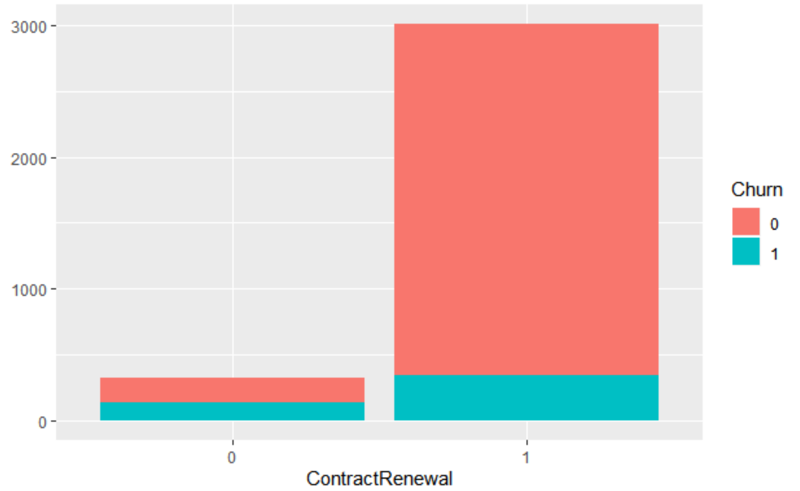



```
qplot(CustServCalls,AccountWeeks,col = DataPlan,data = cell)
```


```
qplot(OverageFee,RoamMins,fill = DataPlan,data = cell,geom = "area")
```

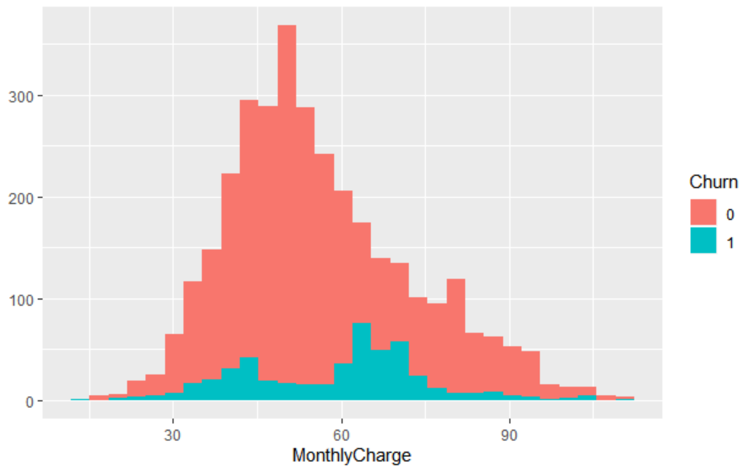


```
qplot(OverageFee,fill = DataPlan,data = cell)
```

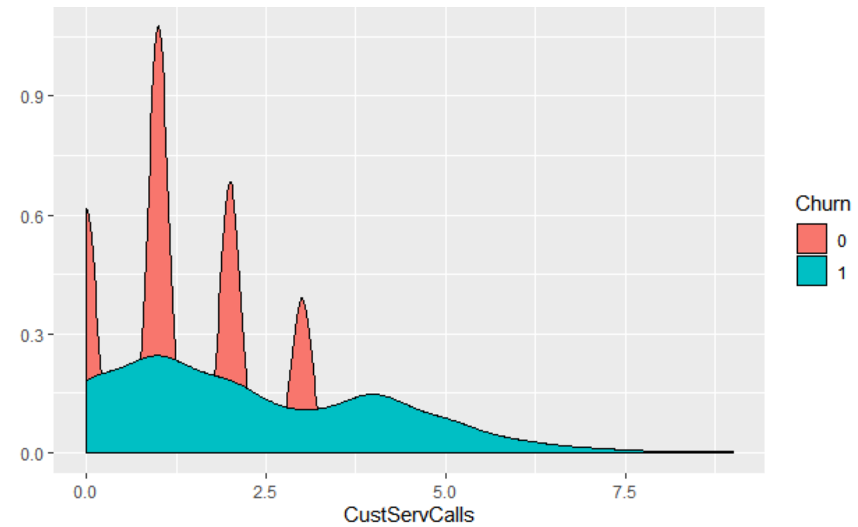


iii. *Dependent Vairable vs Independent Numerical Variables:*

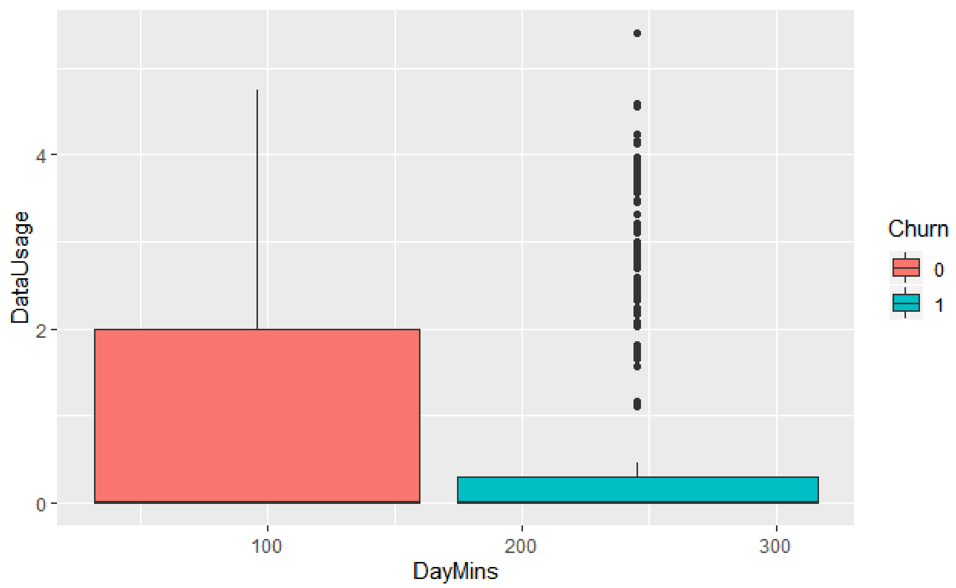
```
qplot(MonthlyCharge,fill = Churn,data = cell)
```




```
qplot(CustServCalls,fill = Churn,data = cell,geom = "density")
```




```
qplot(DayMins,DataUsage,fill = Churn,data = cell,geom = "boxplot")
```



Inferences:

1) Most of the customers who had their **contract renewed, didn’t churn** and from those who didn’t have their **contract renewed,** only half of them **churned.**

2) We can see that irrespective of having the **data plan or not**, the number of customers who **did not churn** is **higher.**

3) The customers, even though having the **highest Overage Fee,** do not have a **data plan.**

4) We can see that the customers with **higher Daily Minutes** of calling had their **Contract Renewed.**

5) The customers with the **higher Monthly Charge** had a **Data Plan.**

6) The customers whose account was **active** for the longest time had the **least calls** to the **Customer Support** and **didn't** have a **Data Plan**.

7) Even though many of the customers had **high Monthly Charges,** they **did not churn.**

8) The customers with **highest Customer Service Calls** had churned.

9) The customers with **higher Data Usage** and **lesser Daily Minutes** did **not churn** while the customers with **lower Data Usage** and **higher Daily Minutes** actually **churned.**

e.) Missing Values Treatment:

While doing the **summary()** function, we were not able to find any **missing values.** Further we can verify this by using the function **is.na()** paired with **sum().**

```
> sum(is.na(cell))
[1] 0
```

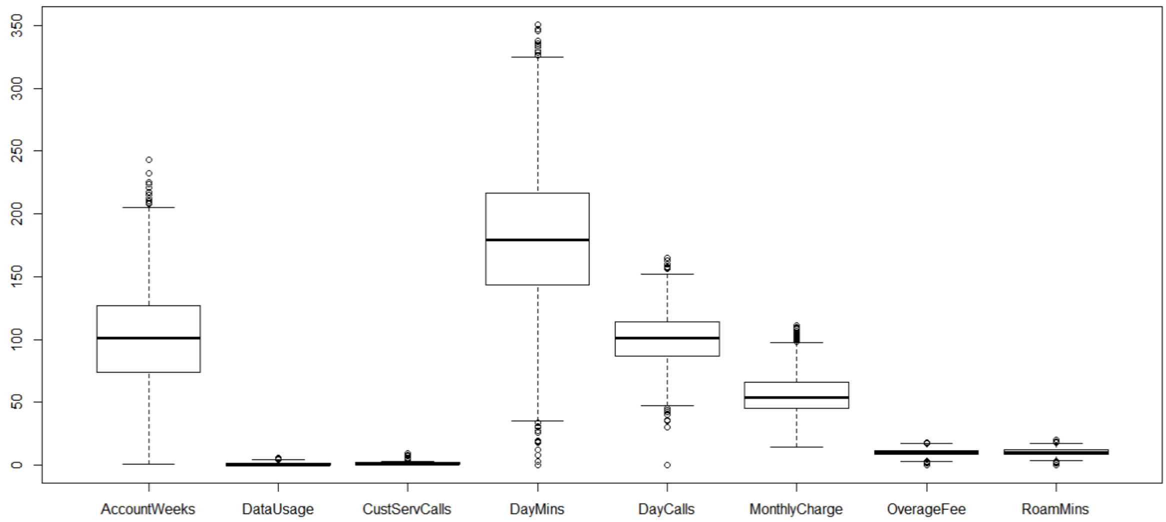
As we can see, there are no missing values present in the data.

f.) Outlier Treatment:

The **Outliers** can be termed as the values that are **1.5 times lesser than first quartile or 1.5 times more than**

third quartile. The best way to detect outliers is by plotting **boxplots** using the function **boxplot()**.

```
### Checking for the outliers ####
boxplot(cell[,-c(1,3,4)])
```



Inferences:

1) We can see that all the **numerical variables** have outliers.

2) Out of all, **DayMins, DayCalls, OverageFee and RoamMins** have **lower Outliers** and **higher Outliers.**

g.) Checking for Multicollinearity:

The Multicollinearity between variables can be termed as **the existence of collinearity between two or more variables.** The **Multicollinearity** can be checked in the following methods.

i. ***Using a Correlation Plot:***

The correlation plot can plotted using the **cor.plot()** function.

```
### Checking for Multicollinearity ###
### Correlation Matrix and Correlation Plot ###
cor.plot(cell[,-c(1,3,4)],numbers = TRUE)
```

Inferences:

We can see that **some** of the variables like **Monthly Charge, DayMins, DataUsage**, etc have **higher Correlation** values between them.

ii. ***Checking the eigen values:***

The **Eigen Values** help in explaining the spread of the values in the variables.

```
> ### Checking the Eigen Values ####
> Eigen = eigen(cor(cell[,-c(1,3,4)]))
> Eigen$values
[1] 2.0421078312 1.1009248509 1.0476751734 1.0024967576 0.9854524398
[6] 0.9546045915 0.8665830289 0.0001553265
```

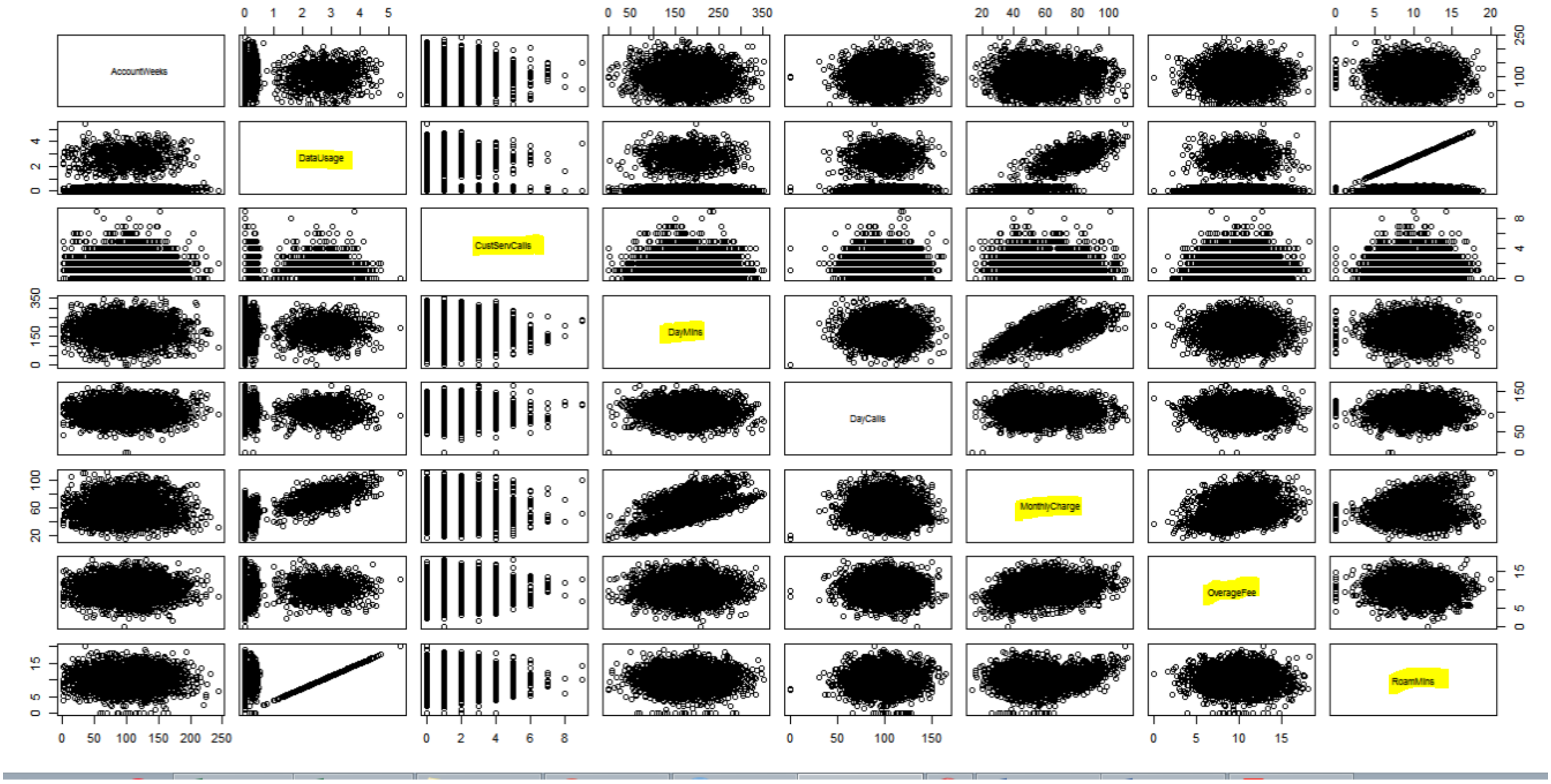
Inferences:

We can see that there is **one value** which is very **near** to the 0. Therefore we can say **MultiCollinearity** exists.

iii. ***Plotting a scatter plot:***

Plotting a scatter plot for all the numerical variables using the **plot()** function can help in identify the Multicollinearity.

```
### Checking the Scatter Plots ###
plot(cell[,-c(1,3,4)])
```



Inferences:

From the above **scatter plots,** we can see that there is a **significant correlation** between **Daymins, MonthlyCharge, OverageFee and RoamMins.**

h.) Summary of EDA:

The EDA was performed on **cellphone data-frame** and following **insights** were drawn:

- The **Highest Data Usage** done by a customer is **5 GB** which is **very low** considering the current data plans offer **1GB per day.**
- The **Highest Monthly Bill** is very low since many of the customers these days use their phone for both calls and internet.
- We can observe that **except one**, most of the variables in the dataset follow a **normal distribution.**
- We can say that the dataset is **highly imbalanced** as there are more number of 0s than 1s in the response variable.
- Customers with the **highest overage fees** do not have a **data plan** meaning that the **Data Usage** isn't the reason for the **Overage Fee**.
- The **Churning** of the customers does not depend on whether customers possess a **Data Plan** or not since most of the customers aren't inclined to the Internet services provided by the telecom service as evident in the **Data Usage** and existence of **Data Plan.**
- Customers with **High Daily Minutes** had **Churned.** It tells that customers who have subscribed for the **Calling services** provided by the company, were not happy with the services.
- The **Monthly Charges** of the customers with **Data Plan** were higher than that of without. This tells us that even though with minimum **Data Usage,** just having a **Data Plan** could increase your **monthly charges.**

- The customers with **highest calls to customer service** have actually **Churned** indicating that issues faced by the customers became the sole purpose for their **Churning.**
- The customers with **high data usage** and **low daily minutes** did not churn, while the customers with **low data usage** and **high daily** minutes did actually churn indicating that the **customers churned** because they had lot of issues with the **Calling services** of the company.

2) Model Building and Comparison:

Since the **dependent variable** in the context is a **categorical variable,** we can use **classification predictive models.** These models include **predictive models** such as **Logistic Regression, Naïve Bayes** and **KNN (K-Nearest Neighbours)**.

We can create each of these 3 models, measure their performance using various parameters and evaluate based on the business problem at hand on which model can be chosen.

Before we go ahead with building models, we first must split the data given to us into two parts, namely **Training and Testing Data** since no separate training and testing data has been provided to us. The splitting of data is done according to the **industry standards** of **(70%-30%)**. We use the **training data** to create the **predictive model** and use this model to make predictions on both **training data and testing data.** These performance measures for these predictions will help in determining the utility of the model.

```
> #### Splitting of data into Training and Testing set(70-30) as per industry standards ####
> set.seed(77)
> indices = sample(nrow(cell),0.70*nrow(cell))
> cell.train = cell[indices,]
> cell.test = cell[-indices,]
> dim(cell.train)
[1] 2333   11
> dim(cell.test)
[1] 1000   11
```

The training dataset is named **cell.train** and the Testing dataset is named **cell.test**.

Logistic Regression:

Logistic Regression is a type of predictive model which is done when the **Dependent variable** is a **categorical variable.** It uses a **Logistic Functions** to predict the **category** by giving a **probability** of a class **as an output**. The logistic regression model can be built using the **glm()** function with **family = "binomial"** as its argument. First we create a **Logistic regression** model using all the variables.

```
> #### Building a logistic regression model ####
> reg = glm(Churn~.,data = cell.train,family = "binomial")
> summary(reg)

Call:
glm(formula = Churn ~ ., family = "binomial", data = cell.train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.8704  -0.5233  -0.3399  -0.1904   3.0902

Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)     -6.606252   0.660054 -10.009  < 2e-16 ***
AccountWeeks     0.002159   0.001669   1.294 0.195836
ContractRenewal1 -1.972206  0.173220 -11.386  < 2e-16 ***
DataPlan1       -1.459268   0.648190  -2.251 0.024367 *
DataUsage        0.182619   2.291421   0.080 0.936478
CustServCalls    0.486529   0.047693  10.201  < 2e-16 ***
DayMins          0.015397   0.038693   0.398 0.690684
DayCalls         0.001942   0.003238   0.600 0.548661
MonthlyCharge   -0.004460   0.227431  -0.020 0.984354
OverageFee       0.169205   0.387872   0.436 0.662664
RoamMins         0.089938   0.025933   3.468 0.000524 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1955.1  on 2332  degrees of freedom
Residual deviance: 1531.3  on 2322  degrees of freedom
AIC: 1553.3

Number of Fisher Scoring iterations: 6
```

As we can see, only **four variables ContractRenewal,DataPlan,CustServCalls** and **RoamMins** are significant variables with their **p-values** being less than **0.05.** And also to improve the model, we can find add new variables arising from the **interactions** of the **Correlated variables.** The variables which we found to be correlated from the **EDA** are **MontlyCharge, DataUsage**, **DayMins and OverageFee.** Using these, we create our final model namely **reg1**.

```
> summary(reg1)

Call:
glm(formula = Churn ~ ContractRenewal + RoamMins + CustServCalls +
    DataPlan + MonthlyCharge * DataUsage + DayMins * MonthlyCharge +
    OverageFee * MonthlyCharge - MonthlyCharge - OverageFee -
    DayMins, family = "binomial", data = cell.train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.9232  -0.5043  -0.3186  -0.1931   2.9436

Coefficients:
                           Estimate Std. Error z value Pr(>|z|)
(Intercept)              -4.652e+00  4.268e-01 -10.899  < 2e-16 ***
ContractRenewal1         -2.062e+00  1.766e-01 -11.673  < 2e-16 ***
RoamMins                  1.049e-01  2.685e-02   3.908 9.31e-05 ***
CustServCalls             5.032e-01  4.864e-02  10.346  < 2e-16 ***
DataPlan1                -2.851e+00  7.236e-01  -3.939 8.17e-05 ***
DataUsage                 2.206e+00  5.736e-01   3.846  0.00012 ***
MonthlyCharge:DataUsage  -2.493e-02  5.399e-03  -4.617 3.90e-06 ***
MonthlyCharge:DayMins     1.717e-04  1.789e-05   9.594  < 2e-16 ***
MonthlyCharge:OverageFee  1.768e-03  4.000e-04   4.420 9.89e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1955.1  on 2332  degrees of freedom
Residual deviance: 1485.5  on 2324  degrees of freedom
AIC: 1503.5

Number of Fisher Scoring iterations: 6
```

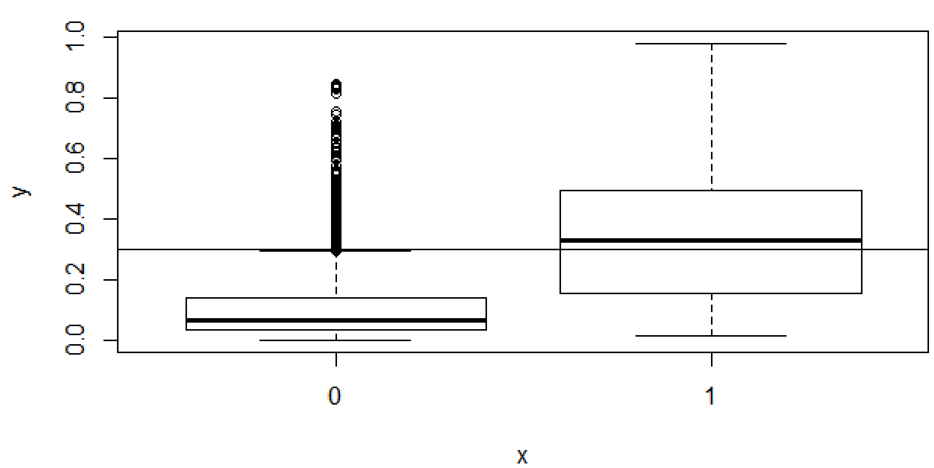
We can see that the **AIC(Akaike information criterion)** has reduced from **1553.3** to **1503.5** which is a good sign of improvement from the previous model.

Making Predictions and measuring performance (Training Data):

Logistic Regression model gives predictions in the form of **probability of an event** occurring. In this case, we can say from the model that **probabilities** given by the model are of **1(Churn)**. We can get these probabilities using the **predict()**

function. To get the **Class predictions** either **0 or 1**, we must split the probabilities using a **certain threshold** to classify them as **0** and **1.** This threshold must be placed keeping in mind the business objective we wish to achieve by this analysis. The threshold must be placed in such a way that the company is able to identify **most number of customers who have churned** while **keeping in mind the acquisition and retention costs** the company has to bear for all the wrong **predictions.**

```
> ### Logistic Regression Predictions - I (Train on Train data)####)
> reg.train.predict = predict(reg1,cell.train,type = "response")
> plot(cell.train$Churn,reg1$fitted.values)
> abline(a = 0.30,b = 0)
> reg.train.response = ifelse(reg.train.predict > 0.3,1,0)
> reg.train.response = as.factor(reg.train.response)
> cell.train$Churn = as.factor(cell.train$Churn)
```



According to the **plot** above, we can see that adding a **threshold** at 0.3 will give us right amount of **right predictions** for **0s** and **1s** for the given **training data** all while not costing too much by making wrong predictions.

To test the performance of **any Classification model,** we have various measures which are as follows:

i. **Confusion Matrix :** We use the confusion matrix to get a frequency of the predicted values vs. the actual values.

ii. **ROC & AUC:** Both of these measures help in determining the of separation of the different Categories present in the **Target and Prediction Variables.** AUC = Area Under the Curve, ROC = Receiver Operating Characteristics.

iii. **KS Value:** The **KS**(Kolmogorov-Smirnov) value is the **highest separation of classes** that has been achieved by the predictive model.

iv. **Gini:** The **Gini Coefficient** be determined to test the **purity** of the classes divided in the **Target variable using the prediction model.**

v. **Concordance Ratio :** In this method, the **values of Response variables and Probabilities are taken as pairs** and then tested if the **probabilities** predicted actually hold true. And then the number of pairs are counted with respect to total number of **pairs.**

i. **Confusion Matrix:**

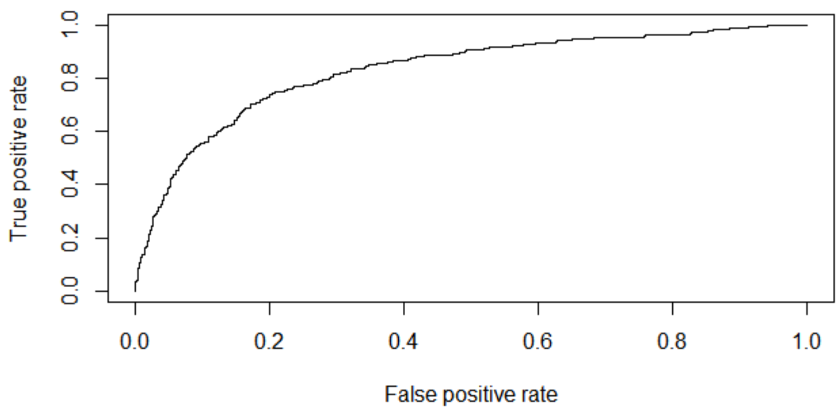
We can build a confusion matrix using the **class predictions** we obtained from the **probabilities**.

```
> ### Confusion Matrix Logistic Regression(Train on Train data) ###
> confusionMatrix(cell.train$Churn,reg.train.response,positive = "1")
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1816  172
         1  163  182

               Accuracy : 0.8564
                 95% CI : (0.8415, 0.8704)
    No Information Rate : 0.8483
    P-Value [Acc > NIR] : 0.1426

                  Kappa : 0.4363

 Mcnemar's Test P-Value : 0.6620

            Sensitivity : 0.51412
            Specificity : 0.91764
         Pos Pred Value : 0.52754
         Neg Pred Value : 0.91348
             Prevalence : 0.15174
         Detection Rate : 0.07801
   Detection Prevalence : 0.14788
      Balanced Accuracy : 0.71588

       'Positive' Class : 1
```

We can see that the confusion matrix looks good as the important measures are high. **Accuracy is 0.8564, Sensitivity is 0.51412 and Specificity is 0.91764**.

ii. **ROC:**

```
#### ROC Logistic Regression(Train on Train data) ######
reg.train.obj = prediction(reg.train.predict,cell.train$Churn)
pref.reg.train = performance(reg.train.obj,"tpr","fpr")
plot(pref.reg.train)
```



We can see that **ROC** value is around **0.80**.

iii. **AUC:**

```
> ### AUC Logistic Regression(Train on Train data)#####
> pref.reg.train = performance(reg.train.obj,"tpr","fpr")
> auc.reg.train = performance(reg.train.obj,"auc")
> auc.reg.train = as.numeric(auc.reg.train@y.values)
> print(auc.reg.train)
[1] 0.8314918
```

We can see that **AUC** value is **0.8314918**.

iv. **KS Value:**

```
> #### KS Logistic Regression(Train on Train data) ######
> print(max(pref.reg.train@y.values[[1]] - pref.reg.train@x.values[[1]]))
[1] 0.5417082
```

We can see that **KS Value** is **0.5417082**

v. **Gini:**

```
> ### Gini Logistic Regression(Train on Train data) ####
> gini.reg.train = ineq(reg.train.predict,"gini")
> print(gini.reg.train)
[1] 0.5577332
```

We can see that **Gini** is **0.5577332**.

vi. **Concordance Ratio:**

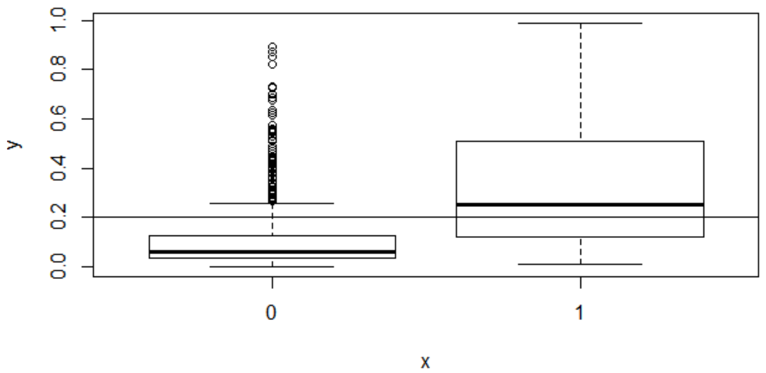
```
> ### Concordance Regression(Train on Train data) ###
> reg.train.x = cell.train$Churn
> reg.train.y = reg.train.predict
> Concordance(actuals = reg.train.x,predictedScores = reg.train.y)
$Concordance
[1] 0.8314918

$Discordance
[1] 0.1685082

$Tied
[1] 0

$Pairs
[1] 685860
```

The Concordance Ratio is **0.8314918.**

Making Predictions and measuring performance (Testing Data):

We can make the predictions in the same way as we did for the **training Data**. But this time the **threshold** differs because of the change in dimensions of the dataset.

```
### Logistic Regression Predictions - II (Train on Test data)####
reg.test.predict = predict(reg1,cell.test,type = "response")
plot(cell.test$Churn,reg.test.predict)
abline(a = 0.20,b = 0)
reg.test.response = ifelse(reg.test.predict > 0.20,1,0)
reg.test.response = as.factor(reg.test.response)
```



According to the **plot** above, we can see that adding a **threshold** at **0.2** will give us right amount of **right predictions** for **0s** and **1s** for the given **training data** all while not costing too much by making wrong predictions.

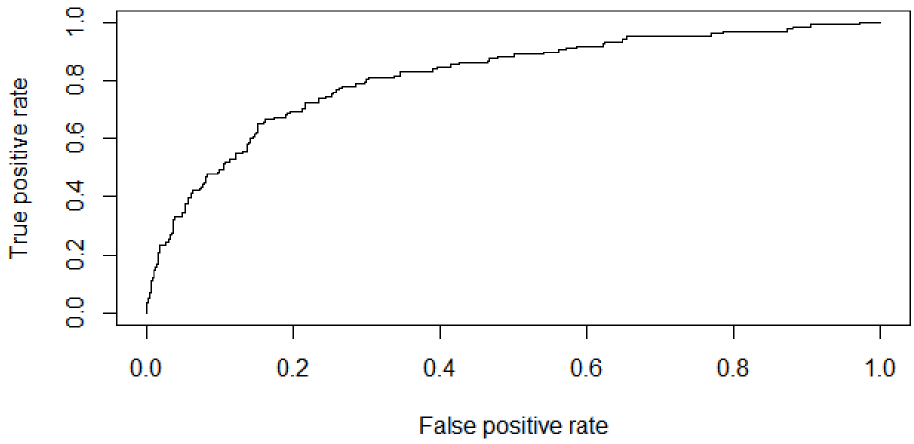
i. Confusion Matrix:

```
> caret::confusionMatrix(cell.test$Churn,reg.test.response,positive = "1")
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 736 126
         1  55  83

               Accuracy : 0.819
                 95% CI : (0.7937, 0.8424)
    No Information Rate : 0.791
    P-Value [Acc > NIR] : 0.01509

                  Kappa : 0.3744

 Mcnemar's Test P-Value : 1.96e-07

            Sensitivity : 0.3971
            Specificity : 0.9305
         Pos Pred Value : 0.6014
         Neg Pred Value : 0.8538
             Prevalence : 0.2090
         Detection Rate : 0.0830
   Detection Prevalence : 0.1380
      Balanced Accuracy : 0.6638

       'Positive' Class : 1

> 
```

Accuracy is 0.81, Sensitivity is 0.3971 and Specificity is 0.9305.

ii. ROC:

```
> #### ROC Logistic Regression(Train on Test data) ######
> reg.test.obj = prediction(reg.test.predict,cell.test$Churn)
> library(ROCR)
> #### ROC Logistic Regression(Train on Test data) ######
> reg.test.obj = prediction(reg.test.predict,cell.test$Churn)
> pref.reg.test = performance(reg.test.obj,"tpr","fpr")
> plot(pref.reg.test)
```



The **ROC** value is **0.80**.

iii. **AUC:**

```
> ### AUC Logistic Regression(Train on Test data)#####
> auc.reg.test = performance(reg.test.obj,"auc")
> auc.reg.test = as.numeric(auc.reg.test@y.values)
> print(auc.reg.test)
[1] 0.8154107
```

The AUC Value is **0.81**

iv. **KS:**

```
> #### KS Logistic Regression(Train on Test data) ######
> print(max(pref.reg.test@y.values[[1]] - pref.reg.test@x.values[[1]]))
[1] 0.5157873
```

The **KS value** is **0.51**

v. **Gini:**

```
> ### Gini Logistic Regression(Train on Test data)####
> gini.reg.test = ineq(reg.test.predict,"gini")
> print(gini.reg.test)
[1] 0.5679071
>
```

The **Gini value** is **0.56**

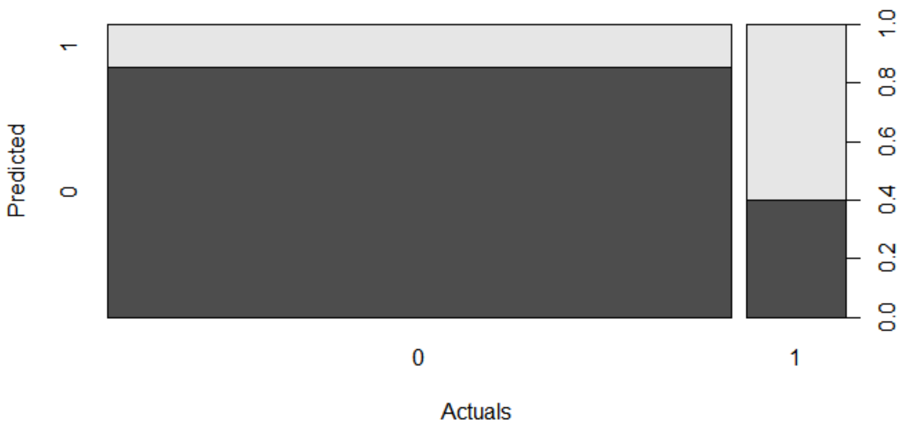
vi. **Concordance:**

```
> ### Concordance Logistic Regression(Train on Test data) ###
> reg.test.x = cell.test$Churn
> reg.test.y = reg.test.predict
> Concordance(actuals = reg.test.x,predictedScores = reg.test.y)
$Concordance
[1] 0.8154107

$Discordance
[1] 0.1845893

$Tied
[1] 2.775558e-17

$Pairs
[1] 118956
```

The **concordance ratio** is **0.81**

Plotting the predictions vs. actuals:

```
### Plotting Actuals vs. Predicted ###
plot(cell.test$Churn,reg.test.response,xlab = "Actuals",ylab = "Predicted")
```



Inferences:

From the above measures, we can see that the model is quite significant giving good values of **Accuracy, Sensitivity** and **Specificity.** Even though the **values of KS, GINI** are on the less but **not insignificant** side, the values of **ROC, AUC and Concordance are good.**

Naïve Bayes:

The **Naïve Bayes** classifier uses an extension of **Bayes** theorem to predict **class outputs** given any number of **independent variables.** The function **naiveBayes()** can be used to build a **Naïve Bayes** model. We use all the variables to build the **Naïve Bayes** model

```
> #### Building a Naive Bayes model ####
> set.seed(77)
> nb.cell.train = naiveBayes(Churn~.,data = cell.train)
> print(nb.cell.train)

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
        0         1 
0.8521217 0.1478783 
```

```
Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
        0         1 
0.8521217 0.1478783 

Conditional probabilities:
   AccountWeeks
Y       [,1]     [,2]
  0 100.2435 39.85512
  1 104.1681 38.80146

   ContractRenewal
Y            0          1
  0 0.06891348 0.93108652
  1 0.27246377 0.72753623

   DataPlan
Y           0         1
  0 0.7087525 0.2912475
  1 0.8492754 0.1507246

   DataUsage
Y        [,1]     [,2]
  0 0.8529024 1.283242
  1 0.5135942 1.150821

   CustServCalls
Y       [,1]     [,2]
  0 1.441650 1.159246
  1 2.127536 1.729017

   DayMins
Y       [,1]     [,2]
  0 175.2340 50.49671
  1 211.2032 69.31346

   DayCalls
Y       [,1]     [,2]
  0 100.5111 19.90391
  1 101.3739 22.20221

   MonthlyCharge
Y       [,1]     [,2]
  0 55.72470 16.45775
  1 59.73304 16.38758

   OverageFee
Y        [,1]     [,2]
  0  9.948798 2.542241
  1 10.707942 2.517917

   RoamMins
Y       [,1]     [,2]
```

Making Predictions and measuring performance (Training Data):

To get the **probabilities** predictions, we can use **predict()** function with argument **raw** and to get **class** predictions, we can use the **class** argument.

```
> #### Making predictions Naive Bayes (Train on Train) #####
> nb.train.response = predict(nb.cell.train,newdata = cell.train,type = 'class')
> nb.train.predict= predict(nb.cell.train,newdata = cell.train,type = 'raw')
> nb.train.predict = as.data.frame(nb.train.predict)
```

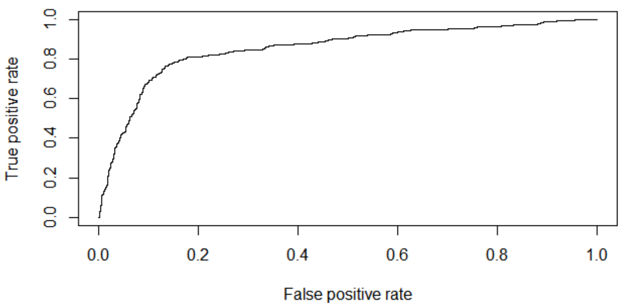
i. Confusion Matrix:

```
> caret::confusionMatrix(nb.train.response,cell.train$Churn,positive = "1")
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1927  241
         1   61  104

               Accuracy : 0.8706
                 95% CI : (0.8563, 0.8839)
    No Information Rate : 0.8521
    P-Value [Acc > NIR] : 0.005931

                  Kappa : 0.3452

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.30145
            Specificity : 0.96932
         Pos Pred Value : 0.63030
         Neg Pred Value : 0.88884
             Prevalence : 0.14788
         Detection Rate : 0.04458
   Detection Prevalence : 0.07072
      Balanced Accuracy : 0.63538

       'Positive' Class : 1
```

The Accuracy is **0.87,** the **sensitivity** is **0.30** and the **specificity is 0.96.**

ii. ROC:

```
> ### Building ROC cuvre Naive Bayes (Train on Train) ####
> nb.train.obj = prediction(nb.train.predict$`1`,cell.train$Churn)
> nb.train.perf = performance(nb.train.obj,"tpr","fpr")
> plot(nb.train.perf)
```



The **ROC** value is **0.82**

iii. AUC:

```
### AUC Naive Bayes(Train on Train) ####
nb.train.auc = performance(nb.train.obj,"auc")
nb.train.auc = as.numeric(nb.train.auc@y.values)
print(nb.train.auc)
```

```
> ### AUC Naive Bayes(Train on Train) ####
> nb.train.auc = performance(nb.train.obj,"auc")
> nb.train.auc = as.numeric(nb.train.auc@y.values)
> print(nb.train.auc)
[1] 0.8553568
```

The **AUC** value is **0.85**

iv. KS:

```
> ### KS Naive Bayes(Train on Train)####
> print(max(nb.train.perf@y.values[[1]] - nb.train.perf@x.values[[1]]))
[1] 0.6351048
```

The **KS** value is **0.63**.

v. Gini:

```
> ### GINI Naive Bayes(Train on Train) ####
> nb.train.gini = ineq(nb.train.predict$`1`,"gini")
> print(nb.train.gini)
[1] 0.5593776
```

The **GINI** value is **0.55**.

vi. Concordance:

```
> ### Concordance Ratio Naive Bayes(Train on Train) ###
> nb.train.x = cell.train$Churn
> nb.train.y = nb.train.predict$`1`
> Concordance(actuals = nb.train.x,predictedScores = nb.train.y)
$Concordance
[1] 0.8553568

$Discordance
[1] 0.1446432

$Tied
[1] 2.775558e-17

$Pairs
[1] 685860
```

The **Concordance Ratio** is **0.85**.

Making Predictions and measuring performance (Testing Data):

```
> #### Making predictions on test data Naive Bayes(Train on Test) #####
> nb.test.response = predict(nb.cell.train,newdata = cell.test,type = 'class')
> nb.test.predict= predict(nb.cell.train,newdata = cell.test,type = 'raw')
> nb.test.predict = as.data.frame(nb.test.predict)
```

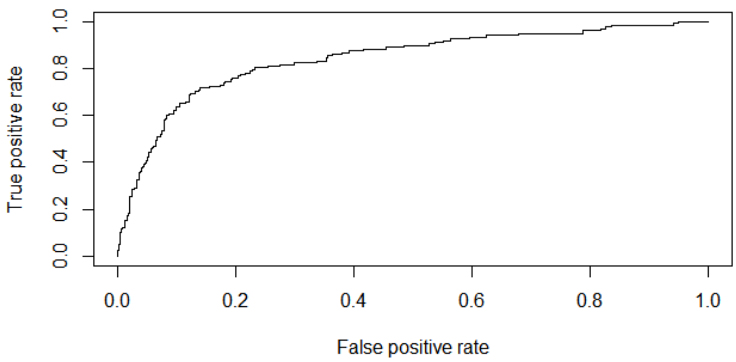
i. Confusion Matrix:

```
> ### Confusion matrix Naive Bayes (Train on Test) ###
> caret::confusionMatrix(nb.test.response,cell.test$Churn,positive = "1")
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 831  93
         1  31  45

               Accuracy : 0.876
                 95% CI : (0.854, 0.8958)
    No Information Rate : 0.862
    P-Value [Acc > NIR] : 0.1067

                  Kappa : 0.3576

 Mcnemar's Test P-Value : 4.303e-08

            Sensitivity : 0.3261
            Specificity : 0.9640
         Pos Pred Value : 0.5921
         Neg Pred Value : 0.8994
             Prevalence : 0.1380
         Detection Rate : 0.0450
   Detection Prevalence : 0.0760
      Balanced Accuracy : 0.6451

       'Positive' Class : 1
```

The **Accuracy** is **0.87,** the **sensitivity** is **0.32** and **specificity** is **0.96.**

ii. ROC:

```
### Building ROC cuvre Naive Bayes(Train on test) ####
nb.test.obj = prediction(nb.test.predict$`1`,cell.test$Churn)
nb.test.perf = performance(nb.test.obj,"tpr","fpr")
plot(nb.test.perf)
```



We can see that **ROC** value is around **0.75**

iii. AUC:

```
> ### AUC Naive Bayes(Train on test) ####
> nb.test.auc = performance(nb.test.obj,"auc")
> nb.test.auc = as.numeric(nb.test.auc@y.values)
> print(nb.test.auc)
[1] 0.8415633
```

We can see that **AUC** value is **0.84**

iv. **<u>KS:</u>**

```
> ### KS Naive Bayes(Train on test)####
> print(max(nb.test.perf@y.values[[1]] - nb.test.perf@x.values[[1]]))
[1] 0.5781802
```

The **KS** value is **0.57**.

v. **<u>Gini:</u>**

```
> ### GINI Naive Bayes(Train on Test) ####
> nb.test.gini = ineq(nb.test.predict$`1`,"gini")
> print(nb.test.gini)
[1] 0.5751989
>
```

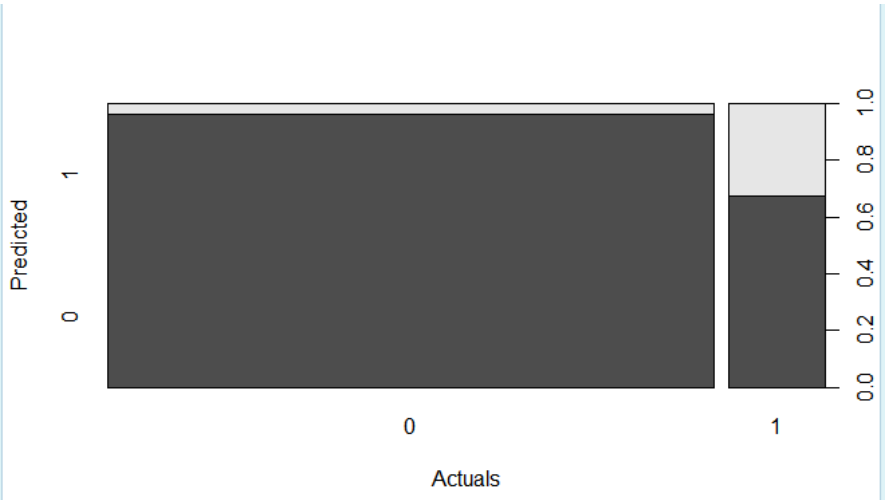
The **Gini** value is **0.57**

vi. **<u>Concordance:</u>**

```
> ### Concordance Naive Bayes(Train on Test) ###
> nb.test.x = cell.test$Churn
> nb.test.y = nb.test.predict$`1`
> Concordance(actuals = nb.test.x,predictedScores = nb.test.y)
$Concordance
[1] 0.8415633

$Discordance
[1] 0.1584367

$Tied
[1] -5.551115e-17

$Pairs
[1] 118956
```

The **Concordance Ratio** is **0.84.**

Plotting the predictions vs. actuals:



Inferences:

We can see that except in **Sensitivity, specificity, KS and GINI,** the model performance measures are really good in **AUC, ROC and Concordance.**

KNN:

KNN(K Nearest Neighbours) algorithm as a **classifier** uses the **current data** and estimates the **class** of the new data point by using certain **similarity measures of distance.** The **KNN** model can be built using the function **knn()**. This function requires an additional argument namely **'k'.** This **K** value denotes the number of nearest neighbours to consider for determining the class of the new data point. The value of k must be in such a way that if it is too low, it will under predict that new data point and if it is too high, it may include other classes into consideration and may result in over-fitting. The exact number of **k** to be taken depends on the size of the dataset. The ideal number would be to take the **square root** of the number of **rows** in the dataset. So for our training dataset, the ideal number would be **48.**

Making Predictions and measuring performance (Training Data):

```
> #### Building A KNN model ####
> #### Training model on Training data KNN (train on train) ####
> knn.train.p = knn(cell.train,cell.train,cl = cell.train$Churn,k = 48,prob = TRUE)
> knn.train.c = knn(cell.train,cell.train,cell.train$Churn,k=48)
> knn.train.prob = attributes(knn.train.p)$prob
> knn.prob.df = data.frame(knn.train.prob,knn.train.c)
> knn.prob.df$knn.train.prob[knn.train.c == "0"] = 1 - knn.prob.df$knn.train.prob[knn.train.c == "0"]
> knn.train.predict = knn.prob.df$knn.train.prob
> knn.train.response = knn.prob.df$knn.train.c
```

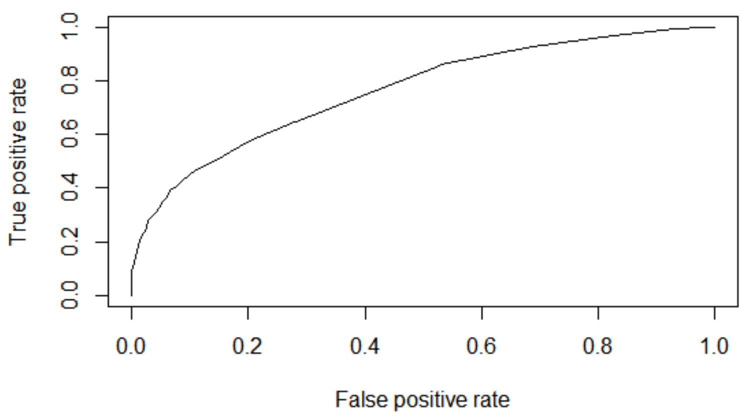
i. Confusion Matrix:

```
> ### Confusion Matrix KNN(Train on Train) ####
> caret::confusionMatrix(knn.train.response,cell.train$Churn,positive = "1")
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1965  278
         1   23   67

               Accuracy : 0.871
                 95% CI : (0.8567, 0.8843)
    No Information Rate : 0.8521
    P-Value [Acc > NIR] : 0.004987

                  Kappa : 0.2629

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.19420
            Specificity : 0.98843
         Pos Pred Value : 0.74444
         Neg Pred Value : 0.87606
             Prevalence : 0.14788
         Detection Rate : 0.02872
   Detection Prevalence : 0.03858
      Balanced Accuracy : 0.59132

       'Positive' Class : 1

>
```

The **Accuracy** is **0.87, sensitivity is 0.19 and specificity is 0.98**

ii. ROC:

```
> ### ROC Curve KNN(Train on Train) ###
> knn.train.obj = prediction(knn.train.predict,cell.train$Churn)
> knn.train.perf = performance(knn.train.obj,"tpr","fpr")
> plot(knn.train.perf)
>
```



The **ROC** value is **0.45.**

iii. AUC:

```
> ### AUC Curve KNN(Train on Train) ###
> knn.train.auc = performance(knn.train.obj,"auc")
> knn.train.auc = as.numeric(knn.train.auc@y.values)
> print(knn.train.auc)
[1] 0.7657547
>
```

The **AUC** value is **0.76**

iv. **KS:**

```
> ### KS Value KNN(Train on Train) ####
> print(max(knn.train.perf@y.values[[1]] - knn.train.perf@x.values[[1]]))
[1] 0.3761074
```

The **KS** value is **0.37**

v. **Gini:**

```
> knn.train.gini = ineq(knn.train.predict,"gini")
> print(knn.train.gini)
[1] 0.416382
```

The **KS value** is **0.41**

vi. **Concordance:**

```
> Concordance(actuals = knn.train.x,predictedScores = knn.train.y)
$Concordance
[1] 0.7362698

$Discordance
[1] 0.2637302

$Tied
[1] 5.551115e-17

$Pairs
[1] 685860
```

The value of **Concordance** is **0.73**.

Making Predictions and measuring performance (Testing Data):

```
> #### Training model on testing data KNN(train on test) ####
> knn.test.p = knn(cell.train,cell.test,cl = cell.train$Churn,k = 31,prob = TRUE)
> knn.test.c = knn(cell.train,cell.test,cell.train$Churn,k=31)
> knn.test.prob = attributes(knn.test.p)$prob
> knn.prob.df = data.frame(knn.test.prob,knn.test.c)
> knn.prob.df$knn.test.prob[knn.test.c == "0"] = 1 - knn.prob.df$knn.test.prob[knn.test.c == "0"]
> knn.test.predict = knn.prob.df$knn.test.prob
> knn.test.response = knn.prob.df$knn.test.c
```

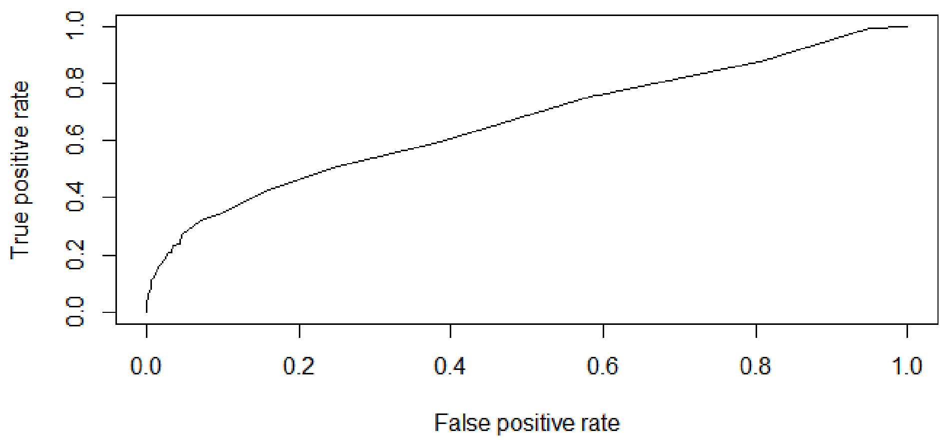
i. Confusion Matrix:

```
> caret::confusionMatrix(knn.test.response,cell.test$Churn,positive = "1")
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 850 118
         1  12  20

               Accuracy : 0.87
                 95% CI : (0.8476, 0.8902)
    No Information Rate : 0.862
    P-Value [Acc > NIR] : 0.2477

                  Kappa : 0.1934

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.1449
            Specificity : 0.9861
         Pos Pred Value : 0.6250
         Neg Pred Value : 0.8781
             Prevalence : 0.1380
         Detection Rate : 0.0200
   Detection Prevalence : 0.0320
      Balanced Accuracy : 0.5655

       'Positive' Class : 1
```

The **Accuracy** is **0.87**, the **sensitivity** is **0.14** and the **specificity** is **0.98**.

ii. ROC:

```
> ### ROC Curve KNN(train on test) ###
> knn.test.obj = prediction(knn.test.predict,cell.test$Churn)
> knn.test.perf = performance(knn.test.obj,"tpr","fpr")
> plot(knn.test.perf)
>
```



The **ROC** value is **0.35**

iii. **AUC:**

```
> ### AUC Curve KNN(train on test) ###
> knn.test.auc = performance(knn.test.obj,"auc")
> knn.test.auc = as.numeric(knn.test.auc@y.values)
> print(knn.test.auc)
[1] 0.6646996
```

The **AUC** value is **0.66**

iv. **KS:**

```
> ### KS Value KNN(train on test) ####
> print(max(knn.test.perf@y.values[[1]] - knn.test.perf@x.values[[1]]))
[1] 0.2659975
```

The **KS** value is **0.26**

v. **Gini:**

```
> ### GINI KNN(train on test) ####
> knn.test.gini = ineq(knn.test.predict,"gini")
> print(knn.test.gini)
[1] 0.4358475
```

The **Gini** value is **0.43**.

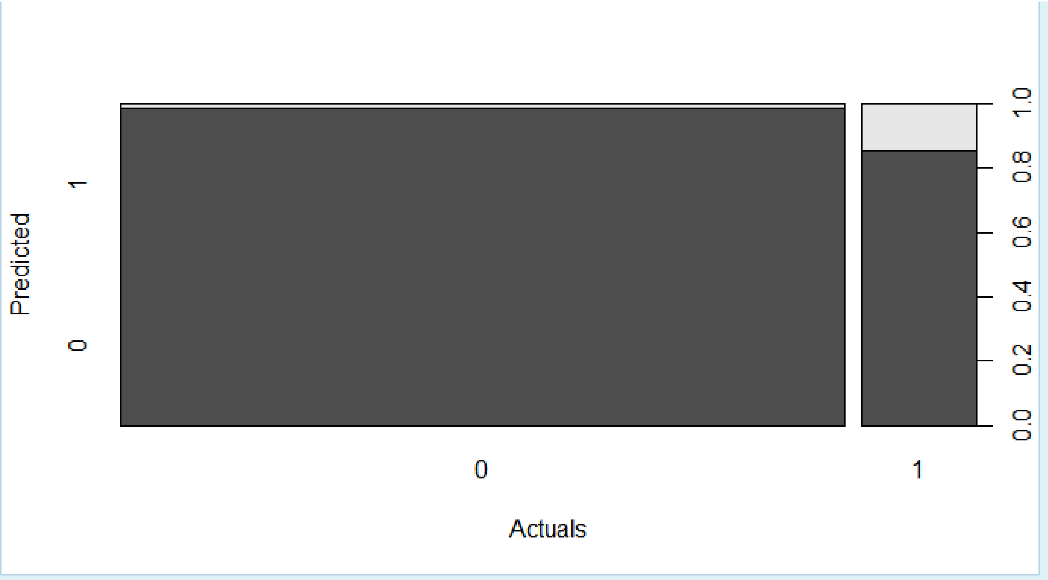
vi. **Concordance:**

```
> ### Concordance KNN (Train on Test) ###
> knn.test.x = cell.test$Churn
> knn.test.y = knn.test.predict
> Concordance(actuals = knn.test.x,predictedScores = knn.test.y)
$Concordance
[1] 0.6136387

$Discordance
[1] 0.3863613

$Tied
[1] 0

$Pairs
[1] 118956
```

The **Concordance Ratio** is **0.61**.

Plotting a graph between actuals vs. predicted :

```
> ### Plotting a graph between actuals vs. predicted
> plot(cell.test$Churn,knn.test.response,xlab = "Actuals",ylab = "Predicted")
```



Inferences:

We can see that in **Gini** and **KS**, the model performed **very poorly**, and in rest of the measures, the model performed **fairly well.**

Model comparison:

Performance Measures	Accuracy	Sensitivity	Specificity	ROC	AUC	KS	Gini	Concordance
log r train on train	0.85	0.51	0.91	0.80	0.83	0.54	0.55	0.83
log r train on test	0.81	0.39	0.93	0.80	0.81	0.51	0.56	0.81
nb train on train	0.87	0.30	0.96	0.82	0.85	0.63	0.55	0.85
nb train on test	0.87	0.32	0.96	0.75	0.84	0.57	0.57	0.84
knn train on train	0.87	0.19	0.98	0.45	0.76	0.37	0.41	0.73
knn train on test	0.87	0.14	0.98	0.35	0.66	0.26	0.43	0.61

Remarks:

Out of all the three models, we can say that **Naïve Bayes** and **Logistic Regression** performed way better over the **KNN**. The **KNN** couldn't perform better because **KNN** model is **very sensitive to outliers** due to its dependency on the distances between the **data points** and as seen in the **EDA** , the data-frame had lot of **outliers**. This can be stated as *major reason for poor performance of KNN model.*

As for the **Logistic Regression** and **Naïve Bayes,** we can say that both of them **performed really well** in the terms of overall measures.

*When compared between two, we can say that **Logistic Regression** did **better** in terms of **sensitivity and specificity** while **Naïve Bayes** did **better** in terms of **AUC, ROC, Gini and KS.***

This is because the **Logistic Regression** model uses **threshold** to differentiate **class** predictions while **Naïve Bayes** makes **probabilistic predictions** and is **not sensitive** to **irrelevant features.**

Below are some of the **limitations** to be taken care of to get **better models** are as follows:

- The **dataset** is **highly imbalanced** as the dataset contains more number of **0s** than **1s**
- All the **numerical variables** not **normally distributed**.
- Most of the variables in the dataset have **high correlation** between them.

- Most of the variables are not **truly independent.**
- The dataset has **huge** number of **outliers.**
- The dataset contains **too many predictor variables**.

If these limitations are taken care of, then all the **performance of all the three models can be improved.**

4) Project conclusion:

The objective of this analysis was to help the management give information on the burning problem of **customers churning.** We were given past data of customers containing their **usage regular telecom usage.** We were able to create **three classification models** out of which we were able to choose two of them, **Logistic Regression** and **Naïve Bayes**. Both of them performed well on **different performance measures.**

The **model suggestion** we can give to management is that:

- If the management wants to **cut costs** keeping in mind **retention** and **acquisition costs**, they must go with the **logistic regression** as they can choose the right threshold for the predictions to get the **right sensitivity and specificity rates.**
- If the management just needs **accurate information** on the **density of churning** that's happening in their customer base, they must go with **Naïve Bayes** as this model helps in giving **better predictions** based on **probabilities.**

The actionable steps that can be taken by the **management** are as follows:

- The selection of the **sample** data must be chosen in such a way that there is a proper balance in the **classes** of **Independent variables.**
- The **sample** must be chosen more carefully taking into account of **reducing the number of outlier**s
- The sample data chosen must be made sure to contain only those **independent variables** that are significant in **model building.**
- The variables thus chosen in the **sample data** must be made sure to **truly independent** which can help in creating better models.
- The company must focus more on improving the **calling services** rather than on **Internet services** as the more customers are facing problems regarding the **calling services.**
- The rates in the area of **Data Plan** must be revised as we can see that customers even though having **very low** data usage have **high monthly charges**.
- We can see that more number of customers are inclined to **calling services** and hence **more attractive offers** can be placed for **voice plans** over **data plans.**
- More budget must be spent on **retention** of the customers as **not many** of the customers tend to be **active for long periods of time**.

5) Appendix – A(Source Code)

TELECOM CUSTMER ANALYSIS

```
######################## TELECOM CUSTOMER ANALYSIS #####################
###Invoking of the necessary Libraries ####
install.packages("readxl")
library(readxl)
install.packages("class")
library(class)
install.packages("psych")
library(psych)
install.packages("ggplot2")
library(ggplot2)
install.packages("caTools")
library(caTools)
install.packages("ineq")
library(ineq)
install.packages("caret")
library(caret)
install.packages("ROCR")
library(ROCR)
install.packages("KODAMA")
library(KODAMA)
install.packages("Hmisc")
library(Hmisc)
install.packages("InformationValue")
library(InformationValue)
install.packages("e1071")
library(e1071)

> ### Setting up the Working directory ###
> setwd("C:/R programs great lakes/P Model/project")
> getwd()
[1] "C:/R programs great lakes/P Model/project"
> #### Importing the dataset and creation of the dataframe #####
> cell = read_xlsx("Cellphone1.xlsx")
> View(cell)
> #### EDA of dataset ####
> dim(cell)
[1] 3333   11
> str(cell)
Classes ‘tbl_df’, ‘tbl’ and 'data.frame':     3333 obs. of  11 variables:
 $ Churn          : num  0 0 0 0 0 0 0 0 0 0 ...
 $ AccountWeeks   : num  128 107 137 84 75 118 121 147 117 141 ...
 $ ContractRenewal: num  1 1 1 0 0 0 1 0 1 0 ...
 $ DataPlan       : num  1 1 0 0 0 0 1 0 0 1 ...
 $ DataUsage      : num  2.7 3.7 0 0 0 0 2.03 0 0.19 3.02 ...
 $ CustServCalls  : num  1 1 0 2 3 0 3 0 1 0 ...
 $ DayMins        : num  265 162 243 299 167 ...
 $ DayCalls       : num  110 123 114 71 113 98 88 79 97 84 ...
 $ MonthlyCharge  : num  89 82 52 57 41 57 87.3 36 63.9 93.2 ...
 $ OverageFee     : num  9.87 9.78 6.06 3.1 7.42 ...
 $ RoamMins       : num  10 13.7 12.2 6.6 10.1 6.3 7.5 7.1 8.7 11.2 ...
> head(cell)
# A tibble: 6 x 11
  Churn AccountWeeks ContractRenewal DataPlan DataUsage CustServCalls
```

```
  <dbl>         <dbl>            <dbl>    <dbl>     <dbl>         <dbl>
1     0           128                1        1       2.7             1
2     0           107                1        1       3.7             1
3     0           137                1        0       0               0
4     0            84                0        0       0               2
5     0            75                0        0       0               3
6     0           118                0        0       0               0
# ... with 5 more variables: DayMins <dbl>, DayCalls <dbl>,
#   MonthlyCharge <dbl>, OverageFee <dbl>, RoamMins <dbl>
> tail(cell)
# A tibble: 6 x 11
  Churn AccountWeeks ContractRenewal DataPlan DataUsage CustServCalls
  <dbl>        <dbl>           <dbl>    <dbl>     <dbl>         <dbl>
1     0           79               1        0      0                2
2     0          192               1        1      2.67             2
3     0           68               1        0      0.34             3
4     0           28               1        0      0                2
5     0          184               0        0      0                2
6     0           74               1        1      3.7              0
# ... with 5 more variables: DayMins <dbl>, DayCalls <dbl>,
#   MonthlyCharge <dbl>, OverageFee <dbl>, RoamMins <dbl>
> summary(cell)
     Churn         AccountWeeks   ContractRenewal     DataPlan     
 Min.   :0.0000   Min.   :  1.0   Min.   :0.0000   Min.   :0.0000  
 1st Qu.:0.0000   1st Qu.: 74.0   1st Qu.:1.0000   1st Qu.:0.0000  
 Median :0.0000   Median :101.0   Median :1.0000   Median :0.0000  
 Mean   :0.1449   Mean   :101.1   Mean   :0.9031   Mean   :0.2766  
 3rd Qu.:0.0000   3rd Qu.:127.0   3rd Qu.:1.0000   3rd Qu.:1.0000  
 Max.   :1.0000   Max.   :243.0   Max.   :1.0000   Max.   :1.0000  
   DataUsage      CustServCalls      DayMins         DayCalls    
 Min.   :0.0000   Min.   :0.000   Min.   :  0.0   Min.   :  0.0  
 1st Qu.:0.0000   1st Qu.:1.000   1st Qu.:143.7   1st Qu.: 87.0  
 Median :0.0000   Median :1.000   Median :179.4   Median :101.0  
 Mean   :0.8165   Mean   :1.563   Mean   :179.8   Mean   :100.4  
 3rd Qu.:1.7800   3rd Qu.:2.000   3rd Qu.:216.4   3rd Qu.:114.0  
 Max.   :5.4000   Max.   :9.000   Max.   :350.8   Max.   :165.0  
 MonthlyCharge      OverageFee       RoamMins    
 Min.   : 14.00   Min.   : 0.00   Min.   : 0.00  
 1st Qu.: 45.00   1st Qu.: 8.33   1st Qu.: 8.50  
 Median : 53.50   Median :10.07   Median :10.30  
 Mean   : 56.31   Mean   :10.05   Mean   :10.24  
 3rd Qu.: 66.20   3rd Qu.:11.77   3rd Qu.:12.10  
 Max.   :111.30   Max.   :18.19   Max.   :20.00  
> ### Conversion of numericals to factors ####
> cell$Churn = as.factor(cell$Churn)
> cell$ContractRenewal = as.factor(cell$ContractRenewal)
> cell$DataPlan = as.factor(cell$DataPlan)
> 
> ### Uni-Variate Analysis ####
> ### Analysis of Independent Numerical variables ###
> hist.data.frame(cell)
> ### Analysis of Independent Categorical variables ###
> table(cell$ContractRenewal)

   0    1 
 323 3010 
> prop.table(table(cell$ContractRenewal))*100

        0         1 
 9.690969 90.309031 
> qplot(ContractRenewal,fill = ContractRenewal,data = cell)
```

```
> table(cell$DataPlan)

   0    1 
2411  922 
> prop.table(table(cell$DataPlan))*100

       0        1 
72.33723 27.66277 
> qplot(DataPlan,fill = DataPlan,data = cell)
> ### Analysis of Dependent variables ###
> table(cell$Churn)

   0    1 
2850  483 
> prop.table(table(cell$Churn))*100

       0        1 
85.50855 14.49145 
> qplot(Churn,fill = Churn,data = cell)
> 
> ##### Bi-Variate Analysis #####
> ### Dependent variable with Independent Categorical variable ###
> qplot(ContractRenewal,fill = Churn,data = cell)
> qplot(DataPlan,fill = Churn,data = cell,geom = "bar")
> ### Independent Numerical variables with Independent Categorical variable
s ###
> qplot(OverageFee,fill = DataPlan,data = cell)
> qplot(DayMins,fill = ContractRenewal,data = cell)
> qplot(MonthlyCharge,fill = DataPlan,data = cell)
> qplot(CustServCalls,AccountWeeks,col = DataPlan,data = cell)
> qplot(OverageFee,RoamMins,fill = DataPlan,data = cell,geom = "area")
> ### Dependent variables with Numerical variables ###
> qplot(MonthlyCharge,fill = Churn,data = cell)
> qplot(CustServCalls,fill = Churn,data = cell,geom = "density")
> qplot(DayMins,DataUsage,fill = Churn,data = cell,geom = "boxplot")
> 
> ### Checking for Missing Values ###
> sum(is.na(cell))
[1] 0
> 
> ### Checking for the outliers ####
> boxplot(cell[,-c(1,3,4)])
> 
> ### Checking for Multicollinearity ###
> ### Correlation Matrix and Correlation Plot ###
> cor.plot(cell[,-c(1,3,4)],numbers = TRUE)
> ### Checking the Eigen Values ####
> Eigen = eigen(cor(cell[,-c(1,3,4)]))
> Eigen$values
[1] 2.0421078312 1.1009248509 1.0476751734 1.0024967576 0.9854524398
[6] 0.9546045915 0.8665830289 0.0001553265
> ### Checking the Scatter Plots ###
> plot(cell[,-c(1,3,4)])
> 
> 
> #### Splitting of data into Training and Testing set(70-30) as per indust
ry standards ####
> set.seed(77)
> indices = sample(nrow(cell),0.70*nrow(cell),replace = FALSE)
> cell.train = cell[indices,]
> cell.test = cell[-indices,]
```

```
> dim(cell.train)
[1] 2333   11
> dim(cell.test)
[1] 1000   11
> #### Building a logistic regression model ####
> reg = glm(Churn~.,data = cell.train,family = "binomial")
> summary(reg)

Call:
glm(formula = Churn ~ ., family = "binomial", data = cell.train)

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-1.8704  -0.5233  -0.3399  -0.1904   3.0902  

Coefficients:
                  Estimate Std. Error z value Pr(>|z|)    
(Intercept)      -6.606252   0.660054 -10.009  < 2e-16 ***
AccountWeeks      0.002159   0.001669   1.294 0.195836    
ContractRenewal1 -1.972206   0.173220 -11.386  < 2e-16 ***
DataPlan1        -1.459268   0.648190  -2.251 0.024367 *  
DataUsage         0.182619   2.291421   0.080 0.936478    
CustServCalls     0.486529   0.047693  10.201  < 2e-16 ***
DayMins           0.015397   0.038693   0.398 0.690684    
DayCalls          0.001942   0.003238   0.600 0.548661    
MonthlyCharge    -0.004460   0.227431  -0.020 0.984354    
OverageFee        0.169205   0.387872   0.436 0.662664    
RoamMins          0.089938   0.025933   3.468 0.000524 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1955.1  on 2332  degrees of freedom
Residual deviance: 1531.3  on 2322  degrees of freedom
AIC: 1553.3

Number of Fisher Scoring iterations: 6

> reg1 = glm(Churn~ContractRenewal+RoamMins+CustServCalls+DataPlan
+            +MonthlyCharge*DataUsage+DayMins*MonthlyCharge
+            +OverageFee*MonthlyCharge -MonthlyCharge -OverageFee-DayMins,da
ta = cell.train
+            ,family = "binomial")
> summary(reg1)

Call:
glm(formula = Churn ~ ContractRenewal + RoamMins + CustServCalls + 
    DataPlan + MonthlyCharge * DataUsage + DayMins * MonthlyCharge + 
    OverageFee * MonthlyCharge - MonthlyCharge - OverageFee - 
    DayMins, family = "binomial", data = cell.train)

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-1.9232  -0.5043  -0.3186  -0.1931   2.9436  

Coefficients:
                           Estimate Std. Error z value Pr(>|z|)    
(Intercept)              -4.652e+00  4.268e-01 -10.899  < 2e-16 ***
ContractRenewal1         -2.062e+00  1.766e-01 -11.673  < 2e-16 ***
RoamMins                  1.049e-01  2.685e-02   3.908 9.31e-05 ***
```

```
CustServCalls              5.032e-01  4.864e-02  10.346  < 2e-16 ***
DataPlan1                 -2.851e+00  7.236e-01  -3.939 8.17e-05 ***
DataUsage                  2.206e+00  5.736e-01   3.846  0.00012 ***
MonthlyCharge:DataUsage   -2.493e-02  5.399e-03  -4.617 3.90e-06 ***
MonthlyCharge:DayMins      1.717e-04  1.789e-05   9.594  < 2e-16 ***
MonthlyCharge:OverageFee   1.768e-03  4.000e-04   4.420 9.89e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1955.1  on 2332  degrees of freedom
Residual deviance: 1485.5  on 2324  degrees of freedom
AIC: 1503.5

Number of Fisher Scoring iterations: 6

>
> ### Logistic Regression Predictions - I (Train on Train data)####)
> reg.train.predict = predict(reg1,cell.train,type = "response")
> plot(cell.train$Churn,reg1$fitted.values)
> abline(a = 0.30,b = 0)
> reg.train.response = ifelse(reg.train.predict > 0.3,1,0)
> reg.train.response = as.factor(reg.train.response)
> cell.train$Churn = as.factor(cell.train$Churn)
> ### Confusion Matrix Logistic Regression(Train on Train data) ###
> library(caret)
> caret::confusionMatrix(cell.train$Churn,reg.train.response,positive = "1"
)
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1816  172
         1  163  182

               Accuracy : 0.8564
                 95% CI : (0.8415, 0.8704)
    No Information Rate : 0.8483
    P-Value [Acc > NIR] : 0.1426

                  Kappa : 0.4363

 Mcnemar's Test P-Value : 0.6620

            Sensitivity : 0.51412
            Specificity : 0.91764
         Pos Pred Value : 0.52754
         Neg Pred Value : 0.91348
             Prevalence : 0.15174
         Detection Rate : 0.07801
   Detection Prevalence : 0.14788
      Balanced Accuracy : 0.71588

       'Positive' Class : 1

> #### ROC Logistic Regression(Train on Train data) ######
> reg.train.obj = prediction(reg.train.predict,cell.train$Churn)
> pref.reg.train = performance(reg.train.obj,"tpr","fpr")
> plot(pref.reg.train)
> ### AUC Logistic Regression(Train on Train data)#####
```

```
> pref.reg.train = performance(reg.train.obj,"tpr","fpr")
> auc.reg.train = performance(reg.train.obj,"auc")
> auc.reg.train = as.numeric(auc.reg.train@y.values)
> print(auc.reg.train)
[1] 0.8314918
> #### KS Logistic Regression(Train on Train data) ######
> print(max(pref.reg.train@y.values[[1]] - pref.reg.train@x.values[[1]]))
[1] 0.5417082
> ### Gini Logistic Regression(Train on Train data) ####
> gini.reg.train = ineq(reg.train.predict,"gini")
> print(gini.reg.train)
[1] 0.5577332
> ### Concordance Regression(Train on Train data) ###
> reg.train.x = cell.train$Churn
> reg.train.y = reg.train.predict
> Concordance(actuals = reg.train.x,predictedScores = reg.train.y)
$Concordance
[1] 0.8314918

$Discordance
[1] 0.1685082

$Tied
[1] 0

$Pairs
[1] 685860

>
> ### Logistic Regression Predictions - II (Train on Test data)####
> reg.test.predict = predict(reg1,cell.test,type = "response")
> plot(cell.test$Churn,reg.test.predict)
> abline(a = 0.20,b = 0)
> reg.test.response = ifelse(reg.test.predict > 0.20,1,0)
> reg.test.response = as.factor(reg.test.response)
> cell.test$Churn = as.factor(cell.test$Churn)
> ### Confusion Matrix Logistic Regression(Train on Test data) ###
> caret::confusionMatrix(cell.test$Churn,reg.test.response,positive = "1")
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 736 126
         1  55  83

               Accuracy : 0.819
                 95% CI : (0.7937, 0.8424)
    No Information Rate : 0.791
    P-Value [Acc > NIR] : 0.01509

                  Kappa : 0.3744

 Mcnemar's Test P-Value : 1.96e-07

            Sensitivity : 0.3971
            Specificity : 0.9305
         Pos Pred Value : 0.6014
         Neg Pred Value : 0.8538
             Prevalence : 0.2090
         Detection Rate : 0.0830
   Detection Prevalence : 0.1380
```

```
      Balanced Accuracy : 0.6638

       'Positive' Class : 1

> #### ROC Logistic Regression(Train on Test data) ######
> reg.test.obj = prediction(reg.test.predict,cell.test$Churn)
> pref.reg.test = performance(reg.test.obj,"tpr","fpr")
> plot(pref.reg.test)
> ### AUC Logistic Regression(Train on Test data)#####
> auc.reg.test = performance(reg.test.obj,"auc")
> auc.reg.test = as.numeric(auc.reg.test@y.values)
> print(auc.reg.test)
[1] 0.8154107
> #### KS Logistic Regression(Train on Test data) ######
> print(max(pref.reg.test@y.values[[1]] - pref.reg.test@x.values[[1]]))
[1] 0.5157873
> ### Gini Logistic Regression(Train on Test data)####
> gini.reg.test = ineq(reg.test.predict,"gini")
> print(gini.reg.test)
[1] 0.5679071
> ### Concordance Logistic Regression(Train on Test data) ###
> reg.test.x = cell.test$Churn
> reg.test.y = reg.test.predict
> Concordance(actuals = reg.test.x,predictedScores = reg.test.y)
$Concordance
[1] 0.8154107

$Discordance
[1] 0.1845893

$Tied
[1] 2.775558e-17

$Pairs
[1] 118956

> ### Plotting Actuals vs. Predicted ###
> plot(cell.test$Churn,reg.test.response,xlab = "Actuals",ylab = "Predicted
")
>
> #### Building a Naive Bayes model ####
> set.seed(77)
> nb.cell.train = naiveBayes(Churn~.,data = cell.train)
> print(nb.cell.train)

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
        0         1
0.8521217 0.1478783

Conditional probabilities:
   AccountWeeks
Y       [,1]     [,2]
  0 100.2435 39.85512
  1 104.1681 38.80146
```

```
   ContractRenewal
Y             0          1
  0 0.06891348 0.93108652
  1 0.27246377 0.72753623

   DataPlan
Y            0         1
  0 0.7087525 0.2912475
  1 0.8492754 0.1507246

   DataUsage
Y         [,1]     [,2]
  0 0.8529024 1.283242
  1 0.5135942 1.150821

   CustServCalls
Y        [,1]     [,2]
  0 1.441650 1.159246
  1 2.127536 1.729017

   DayMins
Y        [,1]     [,2]
  0 175.2340 50.49671
  1 211.2032 69.31346

   DayCalls
Y        [,1]     [,2]
  0 100.5111 19.90391
  1 101.3739 22.20221

   MonthlyCharge
Y        [,1]     [,2]
  0 55.72470 16.45775
  1 59.73304 16.38758

   OverageFee
Y         [,1]     [,2]
  0  9.948798 2.542241
  1 10.707942 2.517917

   RoamMins
Y        [,1]     [,2]
  0 10.18290 2.835821
  1 10.75971 2.774987

> #### Making predictions Naive Bayes (Train on Train) #####
> nb.train.response = predict(nb.cell.train,newdata = cell.train,type = 'cl
ass')
> nb.train.predict= predict(nb.cell.train,newdata = cell.train,type = 'raw'
)
> nb.train.predict = as.data.frame(nb.train.predict)
> ### Confusion Matrix Naive Bayes (Train on Train) ###
> caret::confusionMatrix(nb.train.response,cell.train$Churn,positive = "1")
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1927  241
         1   61  104

               Accuracy : 0.8706
```

```
                 95% CI : (0.8563, 0.8839)
    No Information Rate : 0.8521
    P-Value [Acc > NIR] : 0.005931

                  Kappa : 0.3452

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.30145
            Specificity : 0.96932
         Pos Pred Value : 0.63030
         Neg Pred Value : 0.88884
             Prevalence : 0.14788
         Detection Rate : 0.04458
   Detection Prevalence : 0.07072
      Balanced Accuracy : 0.63538

       'Positive' Class : 1

> ### Building ROC cuvre Naive Bayes (Train on Train) ####
> nb.train.obj = prediction(nb.train.predict$`1`,cell.train$Churn)
> nb.train.perf = performance(nb.train.obj,"tpr","fpr")
> plot(nb.train.perf)
> ### AUC Naive Bayes(Train on Train) ####
> nb.train.auc = performance(nb.train.obj,"auc")
> nb.train.auc = as.numeric(nb.train.auc@y.values)
> print(nb.train.auc)
[1] 0.8553568
> ### KS Naive Bayes(Train on Train)####
> print(max(nb.train.perf@y.values[[1]] - nb.train.perf@x.values[[1]]))
[1] 0.6351048
> ### GINI Naive Bayes(Train on Train) ####
> nb.train.gini = ineq(nb.train.predict$`1`,"gini")
> print(nb.train.gini)
[1] 0.5593776
> ### Concordance Ratio Naive Bayes(Train on Train) ###
> nb.train.x = cell.train$Churn
> nb.train.y = nb.train.predict$`1`
> Concordance(actuals = nb.train.x,predictedScores = nb.train.y)
$Concordance
[1] 0.8553568

$Discordance
[1] 0.1446432

$Tied
[1] 2.775558e-17

$Pairs
[1] 685860

>
> #### Making predictions on test data Naive Bayes(Train on Test) #####
> nb.test.response = predict(nb.cell.train,newdata = cell.test,type = 'clas
s')
> nb.test.predict= predict(nb.cell.train,newdata = cell.test,type = 'raw')
> nb.test.predict = as.data.frame(nb.test.predict)
> ### Confusion matrix Naive Bayes (Train on Test) ###
> caret::confusionMatrix(nb.test.response,cell.test$Churn,positive = "1")
Confusion Matrix and Statistics
```

```
          Reference
Prediction   0   1
         0 831  93
         1  31  45

               Accuracy : 0.876
                 95% CI : (0.854, 0.8958)
    No Information Rate : 0.862
    P-Value [Acc > NIR] : 0.1067

                  Kappa : 0.3576

 Mcnemar's Test P-Value : 4.303e-08

            Sensitivity : 0.3261
            Specificity : 0.9640
         Pos Pred Value : 0.5921
         Neg Pred Value : 0.8994
             Prevalence : 0.1380
         Detection Rate : 0.0450
   Detection Prevalence : 0.0760
      Balanced Accuracy : 0.6451

       'Positive' Class : 1

> ### Building ROC cuvre Naive Bayes(Train on test) ####
> nb.test.obj = prediction(nb.test.predict$`1`,cell.test$Churn)
> nb.test.perf = performance(nb.test.obj,"tpr","fpr")
> plot(nb.test.perf)
> ### AUC Naive Bayes(Train on test) ####
> nb.test.auc = performance(nb.test.obj,"auc")
> nb.test.auc = as.numeric(nb.test.auc@y.values)
> print(nb.test.auc)
[1] 0.8415633
> ### KS Naive Bayes(Train on test)####
> print(max(nb.test.perf@y.values[[1]] - nb.test.perf@x.values[[1]]))
[1] 0.5781802
> ### GINI Naive Bayes(Train on Test) ####
> nb.test.gini = ineq(nb.test.predict$`1`,"gini")
> print(nb.test.gini)
[1] 0.5751989
> ### Concordance Naive Bayes(Train on Test) ###
> nb.test.x = cell.test$Churn
> nb.test.y = nb.test.predict$`1`
> Concordance(actuals = nb.test.x,predictedScores = nb.test.y)
$Concordance
[1] 0.8415633

$Discordance
[1] 0.1584367

$Tied
[1] -5.551115e-17

$Pairs
[1] 118956

> ### Plotting Actuals vs. Predicted ###
> plot(cell.test$Churn,nb.test.response,xlab = "Actuals",ylab = "Predicted"
)
>
```

```
>
> #### Building A KNN model ####
> #### Training model on Training data KNN (train on train) ####
> knn.train.p = knn(cell.train,cell.train,cl = cell.train$Churn,k = 48,prob
= TRUE)
> knn.train.c = knn(cell.train,cell.train,cell.train$Churn,k=48)
> knn.train.prob = attributes(knn.train.p)$prob
> knn.prob.df = data.frame(knn.train.prob,knn.train.c)
> knn.prob.df$knn.train.prob[knn.train.c == "0"] = 1 - knn.prob.df$knn.trai
n.prob[knn.train.c == "0"]
> knn.train.predict = knn.prob.df$knn.train.prob
> knn.train.response = knn.prob.df$knn.train.c
> ### Confusion Matrix KNN(Train on Train) ####
> caret::confusionMatrix(knn.train.response,cell.train$Churn,positive = "1"
)
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1964  277
         1   24   68

               Accuracy : 0.871
                 95% CI : (0.8567, 0.8843)
    No Information Rate : 0.8521
    P-Value [Acc > NIR] : 0.004987

                  Kappa : 0.2655

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.19710
            Specificity : 0.98793
         Pos Pred Value : 0.73913
         Neg Pred Value : 0.87639
             Prevalence : 0.14788
         Detection Rate : 0.02915
   Detection Prevalence : 0.03943
      Balanced Accuracy : 0.59251

       'Positive' Class : 1

> ### ROC Curve KNN(Train on Train) ###
> knn.train.obj = prediction(knn.train.predict,cell.train$Churn)
> knn.train.perf = performance(knn.train.obj,"tpr","fpr")
> plot(knn.train.perf)
> ### AUC Curve KNN(Train on Train) ###
> knn.train.auc = performance(knn.train.obj,"auc")
> knn.train.auc = as.numeric(knn.train.auc@y.values)
> print(knn.train.auc)
[1] 0.7657547
> ### KS Value KNN(Train on Train) ####
> print(max(knn.train.perf@y.values[[1]] - knn.train.perf@x.values[[1]]))
[1] 0.3761074
> ### GINI KNN(Train on Train) ####
> knn.train.gini = ineq(knn.train.predict,"gini")
> print(knn.train.gini)
[1] 0.416382
> ### Concordance KNN (Train on Train) ###
> knn.train.x = cell.train$Churn
> knn.train.y = knn.train.predict
```

```
> Concordance(actuals = knn.train.x,predictedScores = knn.train.y)
$Concordance
[1] 0.7362698

$Discordance
[1] 0.2637302

$Tied
[1] 5.551115e-17

$Pairs
[1] 685860

>
> #### Training model on testing data KNN(train on test) ####
> knn.test.p = knn(cell.train,cell.test,cl = cell.train$Churn,k = 31,prob =
TRUE)
> knn.test.c = knn(cell.train,cell.test,cell.train$Churn,k=31)
> knn.test.prob = attributes(knn.test.p)$prob
> knn.prob.df = data.frame(knn.test.prob,knn.test.c)
> knn.prob.df$knn.test.prob[knn.test.c == "0"] = 1 - knn.prob.df$knn.test.p
rob[knn.test.c == "0"]
> knn.test.predict = knn.prob.df$knn.test.prob
> knn.test.response = knn.prob.df$knn.test.c
> ### Confusion Matrix KNN (train on test) ###
> caret::confusionMatrix(knn.test.response,cell.test$Churn,positive = "1")
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 850 118
         1  12  20

               Accuracy : 0.87
                 95% CI : (0.8476, 0.8902)
    No Information Rate : 0.862
    P-Value [Acc > NIR] : 0.2477

                  Kappa : 0.1934

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.1449
            Specificity : 0.9861
         Pos Pred Value : 0.6250
         Neg Pred Value : 0.8781
             Prevalence : 0.1380
         Detection Rate : 0.0200
   Detection Prevalence : 0.0320
      Balanced Accuracy : 0.5655

       'Positive' Class : 1

> ### ROC Curve KNN(train on test) ###
> knn.test.obj = prediction(knn.test.predict,cell.test$Churn)


> knn.test.auc = performance(knn.test.obj,"auc")
> knn.test.auc = as.numeric(knn.test.auc@y.values)
> print(knn.test.auc)
```

```
[1] 0.67079
> ### KS Value KNN(train on test) ####
> print(max(knn.test.perf@y.values[[1]] - knn.test.perf@x.values[[1]]))
[1] 0.2639968
> ### GINI KNN(train on test) ####
> knn.test.gini = ineq(knn.test.predict,"gini")
> print(knn.test.gini)
[1] 0.579801
> ### Concordance KNN (Train on Test) ###
> knn.test.x = cell.test$Churn
> knn.test.y = knn.test.predict
> Concordance(actuals = knn.test.x,predictedScores = knn.test.y)
$Concordance
[1] 0.5672349

$Discordance
[1] 0.4327651

$Tied
[1] 5.551115e-17

$Pairs
[1] 118956

> #### Training model on testing data KNN(train on test) ####
> knn.test.p = knn(cell.train,cell.test,cl = cell.train$Churn,k = 31,prob =
TRUE)
> knn.test.c = knn(cell.train,cell.test,cell.train$Churn,k=31)
> knn.test.prob = attributes(knn.test.p)$prob
> knn.prob.df = data.frame(knn.test.prob,knn.test.c)
> knn.prob.df$knn.test.prob[knn.test.c == "0"] = 1 - knn.prob.df$knn.test.p
rob[knn.test.c == "0"]
> knn.test.predict = knn.prob.df$knn.test.prob
> knn.test.response = knn.prob.df$knn.test.c
> ### Confusion Matrix KNN (train on test) ###
> caret::confusionMatrix(knn.test.response,cell.test$Churn,positive = "1")
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 850 118
         1  12  20

               Accuracy : 0.87
                 95% CI : (0.8476, 0.8902)
    No Information Rate : 0.862
    P-Value [Acc > NIR] : 0.2477

                  Kappa : 0.1934

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.1449
            Specificity : 0.9861
         Pos Pred Value : 0.6250
         Neg Pred Value : 0.8781
             Prevalence : 0.1380
         Detection Rate : 0.0200
   Detection Prevalence : 0.0320
      Balanced Accuracy : 0.5655
```

```
       'Positive' Class : 1

> ### ROC Curve KNN(train on test) ###
> knn.test.obj = prediction(knn.test.predict,cell.test$Churn)
> knn.test.perf = performance(knn.test.obj,"tpr","fpr")
> plot(knn.test.perf)
> plot(knn.test.perf)
> ### AUC Curve KNN(train on test) ###
> knn.test.auc = performance(knn.test.obj,"auc")
> knn.test.auc = as.numeric(knn.test.auc@y.values)
> print(knn.test.auc)
[1] 0.6646996
> ### KS Value KNN(train on test) ####
> print(max(knn.test.perf@y.values[[1]] - knn.test.perf@x.values[[1]]))
[1] 0.2659975
> ### GINI KNN(train on test) ####
> knn.test.gini = ineq(knn.test.predict,"gini")
> print(knn.test.gini)
[1] 0.4358475
> ### Concordance KNN (Train on Test) ###
> knn.test.x = cell.test$Churn
> knn.test.y = knn.test.predict
> Concordance(actuals = knn.test.x,predictedScores = knn.test.y)
$Concordance
[1] 0.6136387

$Discordance
[1] 0.3863613

$Tied
[1] 0

$Pairs
[1] 118956

> dim(cell.train)
[1] 2333   11
> #### Building A KNN model ####
> #### Training model on Training data KNN (train on train) ####
> knn.train.p = knn(cell.train,cell.train,cl = cell.train$Churn,k = 48,prob
= TRUE)
> knn.train.c = knn(cell.train,cell.train,cell.train$Churn,k=48)
> knn.train.prob = attributes(knn.train.p)$prob
> knn.prob.df = data.frame(knn.train.prob,knn.train.c)
> knn.prob.df$knn.train.prob[knn.train.c == "0"] = 1 - knn.prob.df$knn.trai
n.prob[knn.train.c == "0"]
> knn.train.predict = knn.prob.df$knn.train.prob
> knn.train.response = knn.prob.df$knn.train.c
> ### Confusion Matrix KNN(Train on Train) ####
> confusionMatrix(knn.train.response,cell.train$Churn,positive = "1")
Error in confusionMatrix(knn.train.response, cell.train$Churn, positive = "
1") :
  unused argument (positive = "1")
> ### Confusion Matrix KNN(Train on Train) ####
> caret::confusionMatrix(knn.train.response,cell.train$Churn,positive = "1"
)
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1965  278
```

```
         1   23   67

               Accuracy : 0.871
                 95% CI : (0.8567, 0.8843)
    No Information Rate : 0.8521
    P-Value [Acc > NIR] : 0.004987

                  Kappa : 0.2629

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.19420
            Specificity : 0.98843
         Pos Pred Value : 0.74444
         Neg Pred Value : 0.87606
             Prevalence : 0.14788
         Detection Rate : 0.02872
   Detection Prevalence : 0.03858
      Balanced Accuracy : 0.59132

       'Positive' Class : 1

> ### ROC Curve KNN(Train on Train) ###
> knn.train.obj = prediction(knn.train.predict,cell.train$Churn)
> knn.train.perf = performance(knn.train.obj,"tpr","fpr")
> plot(knn.train.perf)
> ### AUC Curve KNN(Train on Train) ###
> knn.train.auc = performance(knn.train.obj,"auc")
> knn.train.auc = as.numeric(knn.train.auc@y.values)
> print(knn.train.auc)
[1] 0.7657547
> ### KS Value KNN(Train on Train) ####
> print(max(knn.train.perf@y.values[[1]] - knn.train.perf@x.values[[1]]))
[1] 0.3761074
> ### ROC Curve KNN(train on test) ###
> knn.test.obj = prediction(knn.test.predict,cell.test$Churn)
> knn.test.perf = performance(knn.test.obj,"tpr","fpr")
> plot(knn.test.perf)
> ### GINI KNN(train on test) ####
> knn.test.gini = ineq(knn.test.predict,"gini")
> print(knn.test.gini)
[1] 0.4358475
> ### Concordance KNN (Train on Test) ###
> knn.test.x = cell.test$Churn
> knn.test.y = knn.test.predict
> Concordance(actuals = knn.test.x,predictedScores = knn.test.y)
$Concordance
[1] 0.6136387

$Discordance
[1] 0.3863613

$Tied
[1] 0

$Pairs
[1] 118956

> ### KS Value KNN(Train on Train) ####
> print(max(knn.train.perf@y.values[[1]] - knn.train.perf@x.values[[1]]))
[1] 0.3761074
```

```
> ### GINI KNN(Train on Train) ####
> knn.train.gini = ineq(knn.train.predict,"gini")
> print(knn.train.gini)
[1] 0.416382
> ### Concordance KNN (Train on Train) ###
> knn.train.x = cell.train$Churn
> knn.train.y = knn.train.predict
> Concordance(actuals = knn.train.x,predictedScores = knn.train.y)
$Concordance
[1] 0.7362698

$Discordance
[1] 0.2637302

$Tied
[1] 5.551115e-17

$Pairs
[1] 685860

> #### Training model on testing data KNN(train on test) ####
> knn.test.p = knn(cell.train,cell.test,cl = cell.train$Churn,k = 31,prob =
TRUE)
> knn.test.c = knn(cell.train,cell.test,cell.train$Churn,k=31)
> knn.test.prob = attributes(knn.test.p)$prob
> knn.prob.df = data.frame(knn.test.prob,knn.test.c)
> knn.prob.df$knn.test.prob[knn.test.c == "0"] = 1 - knn.prob.df$knn.test.p
rob[knn.test.c == "0"]
> knn.test.predict = knn.prob.df$knn.test.prob
> knn.test.response = knn.prob.df$knn.test.c
> ### Confusion Matrix KNN (train on test) ###
> caret::confusionMatrix(knn.test.response,cell.test$Churn,positive = "1")
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 850 118
         1  12  20

               Accuracy : 0.87
                 95% CI : (0.8476, 0.8902)
    No Information Rate : 0.862
    P-Value [Acc > NIR] : 0.2477

                  Kappa : 0.1934

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.1449
            Specificity : 0.9861
         Pos Pred Value : 0.6250
         Neg Pred Value : 0.8781
             Prevalence : 0.1380
         Detection Rate : 0.0200
   Detection Prevalence : 0.0320
      Balanced Accuracy : 0.5655

       'Positive' Class : 1

> ### ROC Curve KNN(train on test) ###
> knn.test.obj = prediction(knn.test.predict,cell.test$Churn)
```

```
> knn.test.perf = performance(knn.test.obj,"tpr","fpr")
> plot(knn.test.perf)
> ### AUC Curve KNN(train on test) ###
> knn.test.auc = performance(knn.test.obj,"auc")
> knn.test.auc = as.numeric(knn.test.auc@y.values)
> print(knn.test.auc)
[1] 0.6646996
> ### KS Value KNN(train on test) ####
> print(max(knn.test.perf@y.values[[1]] - knn.test.perf@x.values[[1]]))
[1] 0.2659975
> ### Concordance KNN (Train on Test) ###
> knn.test.x = cell.test$Churn
> knn.test.y = knn.test.predict
> Concordance(actuals = knn.test.x,predictedScores = knn.test.y)
$Concordance
[1] 0.6136387

$Discordance
[1] 0.3863613

$Tied
[1] 0

$Pairs
[1] 118956

> ### Plotting a graph between actuals vs. predicted
> plot(cell.test$Churn,knn.test.response,xlab = "Actuals",ylab = "Predicted
")
> ### GINI KNN(train on test) ####
> knn.test.gini = ineq(knn.test.predict,"gini")
> print(knn.test.gini)
[1] 0.4358475
> ### Concordance Regression(Train on Train data) ###
> reg.train.x = cell.train$Churn
> reg.train.y = reg.train.predict
> Concordance(actuals = reg.train.x,predictedScores = reg.train.y)
$Concordance
[1] 0.8314918

$Discordance
[1] 0.1685082

$Tied
[1] 0

$Pairs
[1] 685860

> ### Setting up the Working directory ###
> setwd("C:/R programs great lakes/P Model/project")
> getwd()
[1] "C:/R programs great lakes/P Model/project"
> #### Importing the dataset and creation of the dataframe #####
> cell = read_xlsx("Cellphone1.xlsx")
> View(cell)
> #### EDA of dataset ####
> dim(cell)
[1] 3333   11
> str(cell)
Classes ‘tbl_df’, ‘tbl’ and 'data.frame':     3333 obs. of  11 variables:
```

```
 $ Churn          : num  0 0 0 0 0 0 0 0 0 0 ...
 $ AccountWeeks   : num  128 107 137 84 75 118 121 147 117 141 ...
 $ ContractRenewal: num  1 1 1 0 0 0 1 0 1 0 ...
 $ DataPlan       : num  1 1 0 0 0 0 1 0 0 1 ...
 $ DataUsage      : num  2.7 3.7 0 0 0 0 2.03 0 0.19 3.02 ...
 $ CustServCalls  : num  1 1 0 2 3 0 3 0 1 0 ...
 $ DayMins        : num  265 162 243 299 167 ...
 $ DayCalls       : num  110 123 114 71 113 98 88 79 97 84 ...
 $ MonthlyCharge  : num  89 82 52 57 41 57 87.3 36 63.9 93.2 ...
 $ OverageFee     : num  9.87 9.78 6.06 3.1 7.42 ...
 $ RoamMins       : num  10 13.7 12.2 6.6 10.1 6.3 7.5 7.1 8.7 11.2 ...
> head(cell)
# A tibble: 6 x 11
  Churn AccountWeeks ContractRenewal DataPlan DataUsage CustServCalls
  <dbl>        <dbl>           <dbl>    <dbl>     <dbl>         <dbl>
1     0          128               1        1       2.7             1
2     0          107               1        1       3.7             1
3     0          137               1        0       0               0
4     0           84               0        0       0               2
5     0           75               0        0       0               3
6     0          118               0        0       0               0
# ... with 5 more variables: DayMins <dbl>, DayCalls <dbl>,
#   MonthlyCharge <dbl>, OverageFee <dbl>, RoamMins <dbl>
> tail(cell)
# A tibble: 6 x 11
  Churn AccountWeeks ContractRenewal DataPlan DataUsage CustServCalls
  <dbl>        <dbl>           <dbl>    <dbl>     <dbl>         <dbl>
1     0           79               1        0      0                2
2     0          192               1        1      2.67             2
3     0           68               1        0      0.34             3
4     0           28               1        0      0                2
5     0          184               0        0      0                2
6     0           74               1        1      3.7              0
# ... with 5 more variables: DayMins <dbl>, DayCalls <dbl>,
#   MonthlyCharge <dbl>, OverageFee <dbl>, RoamMins <dbl>
> summary(cell)
     Churn           AccountWeeks    ContractRenewal     DataPlan
 Min.   :0.0000   Min.   :  1.0   Min.   :0.0000   Min.   :0.0000
 1st Qu.:0.0000   1st Qu.: 74.0   1st Qu.:1.0000   1st Qu.:0.0000
 Median :0.0000   Median :101.0   Median :1.0000   Median :0.0000
 Mean   :0.1449   Mean   :101.1   Mean   :0.9031   Mean   :0.2766
 3rd Qu.:0.0000   3rd Qu.:127.0   3rd Qu.:1.0000   3rd Qu.:1.0000
 Max.   :1.0000   Max.   :243.0   Max.   :1.0000   Max.   :1.0000
   DataUsage      CustServCalls      DayMins          DayCalls
 Min.   :0.0000   Min.   :0.000   Min.   :  0.0   Min.   :  0.0
 1st Qu.:0.0000   1st Qu.:1.000   1st Qu.:143.7   1st Qu.: 87.0
 Median :0.0000   Median :1.000   Median :179.4   Median :101.0
 Mean   :0.8165   Mean   :1.563   Mean   :179.8   Mean   :100.4
 3rd Qu.:1.7800   3rd Qu.:2.000   3rd Qu.:216.4   3rd Qu.:114.0
 Max.   :5.4000   Max.   :9.000   Max.   :350.8   Max.   :165.0
 MonthlyCharge      OverageFee       RoamMins
 Min.   : 14.00   Min.   : 0.00   Min.   : 0.00
 1st Qu.: 45.00   1st Qu.: 8.33   1st Qu.: 8.50
 Median : 53.50   Median :10.07   Median :10.30
 Mean   : 56.31   Mean   :10.05   Mean   :10.24
 3rd Qu.: 66.20   3rd Qu.:11.77   3rd Qu.:12.10
 Max.   :111.30   Max.   :18.19   Max.   :20.00
> ### Conversion of numericals to factors ####
> cell$Churn = as.factor(cell$Churn)
> cell$ContractRenewal = as.factor(cell$ContractRenewal)
> cell$DataPlan = as.factor(cell$DataPlan)
```

```
>
> ### Uni-Variate Analysis ####
> ### Analysis of Independent Numerical variables ###
> hist.data.frame(cell)
> ### Analysis of Independent Categorical variables ###
> table(cell$ContractRenewal)

   0    1 
 323 3010 
> prop.table(table(cell$ContractRenewal))*100

        0         1 
 9.690969 90.309031 
> qplot(ContractRenewal,fill = ContractRenewal,data = cell)
> table(cell$DataPlan)

   0    1 
2411  922 
> prop.table(table(cell$DataPlan))*100

       0        1 
72.33723 27.66277 
> qplot(DataPlan,fill = DataPlan,data = cell)
> ### Analysis of Dependent variables ###
> table(cell$Churn)

   0    1 
2850  483 
> prop.table(table(cell$Churn))*100

       0        1 
85.50855 14.49145 
> qplot(Churn,fill = Churn,data = cell)
>
> ##### Bi-Variate Analysis #####
> ### Dependent variable with Independent Categorical variable ###
> qplot(ContractRenewal,fill = Churn,data = cell)
> qplot(DataPlan,fill = Churn,data = cell,geom = "bar")
> ### Independent Numerical variables with Independent Categorical variable
s ###
> qplot(OverageFee,fill = DataPlan,data = cell)
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
> qplot(DayMins,fill = ContractRenewal,data = cell)
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
> qplot(MonthlyCharge,fill = DataPlan,data = cell)
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
> qplot(CustServCalls,AccountWeeks,col = DataPlan,data = cell)
> qplot(OverageFee,RoamMins,fill = DataPlan,data = cell,geom = "area")
> ### Dependent variables with Numerical variables ###
> qplot(MonthlyCharge,fill = Churn,data = cell)
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
> qplot(CustServCalls,fill = Churn,data = cell,geom = "density")
> qplot(DayMins,DataUsage,fill = Churn,data = cell,geom = "boxplot")
>
> ### Checking for Missing Values ###
> sum(is.na(cell))
[1] 0
>
> ### Checking for the outliers ####
> boxplot(cell[,-c(1,3,4)])
>
```

```
> ### Checking for Multicollinearity ###
> ### Correlation Matrix and Correlation Plot ###
> cor.plot(cell[,-c(1,3,4)],numbers = TRUE)
> ### Checking the Eigen Values ####
> Eigen = eigen(cor(cell[,-c(1,3,4)]))
> Eigen$values
[1] 2.0421078312 1.1009248509 1.0476751734 1.0024967576 0.9854524398
[6] 0.9546045915 0.8665830289 0.0001553265
> ### Checking the Scatter Plots ###
> plot(cell[,-c(1,3,4)])
>
>
> #### Splitting of data into Training and Testing set(70-30) as per indust
ry standards ####
> set.seed(77)
> indices = sample(nrow(cell),0.70*nrow(cell),replace = FALSE)
> cell.train = cell[indices,]
> cell.test = cell[-indices,]
> dim(cell.train)
[1] 2333   11
> dim(cell.test)
[1] 1000   11
> #### Building a logistic regression model ####
> reg = glm(Churn~.,data = cell.train,family = "binomial")
> summary(reg)

Call:
glm(formula = Churn ~ ., family = "binomial", data = cell.train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.8704  -0.5233  -0.3399  -0.1904   3.0902

Coefficients:
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)      -6.606252   0.660054 -10.009  < 2e-16 ***
AccountWeeks      0.002159   0.001669   1.294 0.195836
ContractRenewal1 -1.972206   0.173220 -11.386  < 2e-16 ***
DataPlan1        -1.459268   0.648190  -2.251 0.024367 *
DataUsage         0.182619   2.291421   0.080 0.936478
CustServCalls     0.486529   0.047693  10.201  < 2e-16 ***
DayMins           0.015397   0.038693   0.398 0.690684
DayCalls          0.001942   0.003238   0.600 0.548661
MonthlyCharge    -0.004460   0.227431  -0.020 0.984354
OverageFee        0.169205   0.387872   0.436 0.662664
RoamMins          0.089938   0.025933   3.468 0.000524 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1955.1  on 2332  degrees of freedom
Residual deviance: 1531.3  on 2322  degrees of freedom
AIC: 1553.3

Number of Fisher Scoring iterations: 6

> reg1 = glm(Churn~ContractRenewal+RoamMins+CustServCalls+DataPlan
+           +MonthlyCharge*DataUsage+DayMins*MonthlyCharge
+           +OverageFee*MonthlyCharge -MonthlyCharge -OverageFee-DayMins,da
ta = cell.train
```

```
+            ,family = "binomial")
> summary(reg1)

Call:
glm(formula = Churn ~ ContractRenewal + RoamMins + CustServCalls +
    DataPlan + MonthlyCharge * DataUsage + DayMins * MonthlyCharge +
    OverageFee * MonthlyCharge - MonthlyCharge - OverageFee -
    DayMins, family = "binomial", data = cell.train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.9232  -0.5043  -0.3186  -0.1931   2.9436

Coefficients:
                           Estimate Std. Error z value Pr(>|z|)
(Intercept)              -4.652e+00  4.268e-01 -10.899  < 2e-16 ***
ContractRenewal1         -2.062e+00  1.766e-01 -11.673  < 2e-16 ***
RoamMins                  1.049e-01  2.685e-02   3.908 9.31e-05 ***
CustServCalls             5.032e-01  4.864e-02  10.346  < 2e-16 ***
DataPlan1                -2.851e+00  7.236e-01  -3.939 8.17e-05 ***
DataUsage                 2.206e+00  5.736e-01   3.846  0.00012 ***
MonthlyCharge:DataUsage  -2.493e-02  5.399e-03  -4.617 3.90e-06 ***
MonthlyCharge:DayMins     1.717e-04  1.789e-05   9.594  < 2e-16 ***
MonthlyCharge:OverageFee  1.768e-03  4.000e-04   4.420 9.89e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1955.1  on 2332  degrees of freedom
Residual deviance: 1485.5  on 2324  degrees of freedom
AIC: 1503.5

Number of Fisher Scoring iterations: 6

>
> ### Logistic Regression Predictions - I (Train on Train data)####)
> reg.train.predict = predict(reg1,cell.train,type = "response")
> plot(cell.train$Churn,reg1$fitted.values)
> abline(a = 0.30,b = 0)
> reg.train.response = ifelse(reg.train.predict > 0.3,1,0)
> reg.train.response = as.factor(reg.train.response)
> cell.train$Churn = as.factor(cell.train$Churn)
> ### Confusion Matrix Logistic Regression(Train on Train data) ###
> library(caret)
> caret::confusionMatrix(cell.train$Churn,reg.train.response,positive = "1"
)
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1816  172
         1  163  182

               Accuracy : 0.8564
                 95% CI : (0.8415, 0.8704)
    No Information Rate : 0.8483
    P-Value [Acc > NIR] : 0.1426

                  Kappa : 0.4363
```

```
 Mcnemar's Test P-Value : 0.6620

            Sensitivity : 0.51412
            Specificity : 0.91764
         Pos Pred Value : 0.52754
         Neg Pred Value : 0.91348
             Prevalence : 0.15174
         Detection Rate : 0.07801
   Detection Prevalence : 0.14788
      Balanced Accuracy : 0.71588

       'Positive' Class : 1

> #### ROC Logistic Regression(Train on Train data) ######
> reg.train.obj = prediction(reg.train.predict,cell.train$Churn)
> pref.reg.train = performance(reg.train.obj,"tpr","fpr")
> plot(pref.reg.train)
> ### AUC Logistic Regression(Train on Train data)#####
> pref.reg.train = performance(reg.train.obj,"tpr","fpr")
> auc.reg.train = performance(reg.train.obj,"auc")
> auc.reg.train = as.numeric(auc.reg.train@y.values)
> print(auc.reg.train)
[1] 0.8314918
> #### KS Logistic Regression(Train on Train data) ######
> print(max(pref.reg.train@y.values[[1]] - pref.reg.train@x.values[[1]]))
[1] 0.5417082
> ### Gini Logistic Regression(Train on Train data) ####
> gini.reg.train = ineq(reg.train.predict,"gini")
> print(gini.reg.train)
[1] 0.5577332
> ### Concordance Regression(Train on Train data) ###
> reg.train.x = cell.train$Churn
> reg.train.y = reg.train.predict
> Concordance(actuals = reg.train.x,predictedScores = reg.train.y)
$Concordance
[1] 0.8314918

$Discordance
[1] 0.1685082

$Tied
[1] 0

$Pairs
[1] 685860

>
> ### Logistic Regression Predictions - II (Train on Test data)####
> reg.test.predict = predict(reg1,cell.test,type = "response")
> plot(cell.test$Churn,reg.test.predict)
> abline(a = 0.20,b = 0)
> reg.test.response = ifelse(reg.test.predict > 0.20,1,0)
> reg.test.response = as.factor(reg.test.response)
> cell.test$Churn = as.factor(cell.test$Churn)
> ### Confusion Matrix Logistic Regression(Train on Test data) ###
> caret::confusionMatrix(cell.test$Churn,reg.test.response,positive = "1")
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 736 126
```

```
         1  55  83

               Accuracy : 0.819
                 95% CI : (0.7937, 0.8424)
    No Information Rate : 0.791
    P-Value [Acc > NIR] : 0.01509

                  Kappa : 0.3744

 Mcnemar's Test P-Value : 1.96e-07

            Sensitivity : 0.3971
            Specificity : 0.9305
         Pos Pred Value : 0.6014
         Neg Pred Value : 0.8538
             Prevalence : 0.2090
         Detection Rate : 0.0830
   Detection Prevalence : 0.1380
      Balanced Accuracy : 0.6638

       'Positive' Class : 1

> #### ROC Logistic Regression(Train on Test data) ######
> reg.test.obj = prediction(reg.test.predict,cell.test$Churn)
> pref.reg.test = performance(reg.test.obj,"tpr","fpr")
> plot(pref.reg.test)
> ### AUC Logistic Regression(Train on Test data)#####
> auc.reg.test = performance(reg.test.obj,"auc")
> auc.reg.test = as.numeric(auc.reg.test@y.values)
> print(auc.reg.test)
[1] 0.8154107
> #### KS Logistic Regression(Train on Test data) ######
> print(max(pref.reg.test@y.values[[1]] - pref.reg.test@x.values[[1]]))
[1] 0.5157873
> ### Gini Logistic Regression(Train on Test data)####
> gini.reg.test = ineq(reg.test.predict,"gini")
> print(gini.reg.test)
[1] 0.5679071
> ### Concordance Logistic Regression(Train on Test data) ###
> reg.test.x = cell.test$Churn
> reg.test.y = reg.test.predict
> Concordance(actuals = reg.test.x,predictedScores = reg.test.y)
$Concordance
[1] 0.8154107

$Discordance
[1] 0.1845893

$Tied
[1] 2.775558e-17

$Pairs
[1] 118956

> ### Plotting Actuals vs. Predicted ###
> plot(cell.test$Churn,reg.test.response,xlab = "Actuals",ylab = "Predicted
")
>
> #### Building a Naive Bayes model ####
> set.seed(77)
> nb.cell.train = naiveBayes(Churn~.,data = cell.train)
```

```
> print(nb.cell.train)

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
        0         1 
0.8521217 0.1478783 

Conditional probabilities:
   AccountWeeks
Y       [,1]     [,2]
  0 100.2435 39.85512
  1 104.1681 38.80146

   ContractRenewal
Y            0          1
  0 0.06891348 0.93108652
  1 0.27246377 0.72753623

   DataPlan
Y           0         1
  0 0.7087525 0.2912475
  1 0.8492754 0.1507246

   DataUsage
Y        [,1]     [,2]
  0 0.8529024 1.283242
  1 0.5135942 1.150821

   CustServCalls
Y       [,1]     [,2]
  0 1.441650 1.159246
  1 2.127536 1.729017

   DayMins
Y       [,1]     [,2]
  0 175.2340 50.49671
  1 211.2032 69.31346

   DayCalls
Y       [,1]     [,2]
  0 100.5111 19.90391
  1 101.3739 22.20221

   MonthlyCharge
Y       [,1]     [,2]
  0 55.72470 16.45775
  1 59.73304 16.38758

   OverageFee
Y        [,1]     [,2]
  0  9.948798 2.542241
  1 10.707942 2.517917

   RoamMins
Y       [,1]     [,2]
  0 10.18290 2.835821
```

```
  1 10.75971 2.774987

> #### Making predictions Naive Bayes (Train on Train) #####
> nb.train.response = predict(nb.cell.train,newdata = cell.train,type = 'cl
ass')
> nb.train.predict= predict(nb.cell.train,newdata = cell.train,type = 'raw'
)
> nb.train.predict = as.data.frame(nb.train.predict)
> ### Confusion Matrix Naive Bayes (Train on Train) ###
> caret::confusionMatrix(nb.train.response,cell.train$Churn,positive = "1")
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1927  241
         1   61  104

               Accuracy : 0.8706
                 95% CI : (0.8563, 0.8839)
    No Information Rate : 0.8521
    P-Value [Acc > NIR] : 0.005931

                  Kappa : 0.3452

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.30145
            Specificity : 0.96932
         Pos Pred Value : 0.63030
         Neg Pred Value : 0.88884
             Prevalence : 0.14788
         Detection Rate : 0.04458
   Detection Prevalence : 0.07072
      Balanced Accuracy : 0.63538

       'Positive' Class : 1

> ### Building ROC cuvre Naive Bayes (Train on Train) ####
> nb.train.obj = prediction(nb.train.predict$`1`,cell.train$Churn)
> nb.train.perf = performance(nb.train.obj,"tpr","fpr")
> plot(nb.train.perf)
> ### AUC Naive Bayes(Train on Train) ####
> nb.train.auc = performance(nb.train.obj,"auc")
> nb.train.auc = as.numeric(nb.train.auc@y.values)
> print(nb.train.auc)
[1] 0.8553568
> ### KS Naive Bayes(Train on Train)####
> print(max(nb.train.perf@y.values[[1]] - nb.train.perf@x.values[[1]]))
[1] 0.6351048
> ### GINI Naive Bayes(Train on Train) ####
> nb.train.gini = ineq(nb.train.predict$`1`,"gini")
> print(nb.train.gini)
[1] 0.5593776
> ### Concordance Ratio Naive Bayes(Train on Train) ###
> nb.train.x = cell.train$Churn
> nb.train.y = nb.train.predict$`1`
> Concordance(actuals = nb.train.x,predictedScores = nb.train.y)
$Concordance
[1] 0.8553568

$Discordance
```

```
[1] 0.1446432

$Tied
[1] 2.775558e-17

$Pairs
[1] 685860

>
> #### Making predictions on test data Naive Bayes(Train on Test) #####
> nb.test.response = predict(nb.cell.train,newdata = cell.test,type = 'clas
s')
> nb.test.predict= predict(nb.cell.train,newdata = cell.test,type = 'raw')
> nb.test.predict = as.data.frame(nb.test.predict)
> ### Confusion matrix Naive Bayes (Train on Test) ###
> caret::confusionMatrix(nb.test.response,cell.test$Churn,positive = "1")
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 831  93
         1  31  45

               Accuracy : 0.876
                 95% CI : (0.854, 0.8958)
    No Information Rate : 0.862
    P-Value [Acc > NIR] : 0.1067

                  Kappa : 0.3576

 Mcnemar's Test P-Value : 4.303e-08

            Sensitivity : 0.3261
            Specificity : 0.9640
         Pos Pred Value : 0.5921
         Neg Pred Value : 0.8994
             Prevalence : 0.1380
         Detection Rate : 0.0450
   Detection Prevalence : 0.0760
      Balanced Accuracy : 0.6451

       'Positive' Class : 1

> ### Building ROC cuvre Naive Bayes(Train on test) ####
> nb.test.obj = prediction(nb.test.predict$`1`,cell.test$Churn)
> nb.test.perf = performance(nb.test.obj,"tpr","fpr")
> plot(nb.test.perf)
> ### AUC Naive Bayes(Train on test) ####
> nb.test.auc = performance(nb.test.obj,"auc")
> nb.test.auc = as.numeric(nb.test.auc@y.values)
> print(nb.test.auc)
[1] 0.8415633
> ### KS Naive Bayes(Train on test)####
> print(max(nb.test.perf@y.values[[1]] - nb.test.perf@x.values[[1]]))
[1] 0.5781802
> ### GINI Naive Bayes(Train on Test) ####
> nb.test.gini = ineq(nb.test.predict$`1`,"gini")
> print(nb.test.gini)
[1] 0.5751989
> ### Concordance Naive Bayes(Train on Test) ###
> nb.test.x = cell.test$Churn
```

```
> nb.test.y = nb.test.predict$`1`
> Concordance(actuals = nb.test.x,predictedScores = nb.test.y)
$Concordance
[1] 0.8415633

$Discordance
[1] 0.1584367

$Tied
[1] -5.551115e-17

$Pairs
[1] 118956

> ### Plotting Actuals vs. Predicted ###
> plot(cell.test$Churn,nb.test.response,xlab = "Actuals",ylab = "Predicted"
)
>
>
> #### Building A KNN model ####
> #### Training model on Training data KNN (train on train) ####
> knn.train.p = knn(cell.train,cell.train,cl = cell.train$Churn,k = 48,prob
= TRUE)
> knn.train.c = knn(cell.train,cell.train,cell.train$Churn,k=48)
> knn.train.prob = attributes(knn.train.p)$prob
> knn.prob.df = data.frame(knn.train.prob,knn.train.c)
> knn.prob.df$knn.train.prob[knn.train.c == "0"] = 1 - knn.prob.df$knn.trai
n.prob[knn.train.c == "0"]
> knn.train.predict = knn.prob.df$knn.train.prob
> knn.train.response = knn.prob.df$knn.train.c
> ### Confusion Matrix KNN(Train on Train) ####
> caret::confusionMatrix(knn.train.response,cell.train$Churn,positive = "1"
)
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1964  277
         1   24   68

               Accuracy : 0.871
                 95% CI : (0.8567, 0.8843)
    No Information Rate : 0.8521
    P-Value [Acc > NIR] : 0.004987

                  Kappa : 0.2655

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.19710
            Specificity : 0.98793
         Pos Pred Value : 0.73913
         Neg Pred Value : 0.87639
             Prevalence : 0.14788
         Detection Rate : 0.02915
   Detection Prevalence : 0.03943
      Balanced Accuracy : 0.59251

       'Positive' Class : 1

> ### ROC Curve KNN(Train on Train) ###
```

```
> knn.train.obj = prediction(knn.train.predict,cell.train$Churn)
> knn.train.perf = performance(knn.train.obj,"tpr","fpr")
> plot(knn.train.perf)
> ### AUC Curve KNN(Train on Train) ###
> knn.train.auc = performance(knn.train.obj,"auc")
> knn.train.auc = as.numeric(knn.train.auc@y.values)
> print(knn.train.auc)
[1] 0.7657547
> ### KS Value KNN(Train on Train) ####
> print(max(knn.train.perf@y.values[[1]] - knn.train.perf@x.values[[1]]))
[1] 0.3761074
> ### GINI KNN(Train on Train) ####
> knn.train.gini = ineq(knn.train.predict,"gini")
> print(knn.train.gini)
[1] 0.416382
> ### Concordance KNN (Train on Train) ###
> knn.train.x = cell.train$Churn
> knn.train.y = knn.train.predict
> Concordance(actuals = knn.train.x,predictedScores = knn.train.y)
$Concordance
[1] 0.7362698

$Discordance
[1] 0.2637302

$Tied
[1] 5.551115e-17

$Pairs
[1] 685860

>
> #### Training model on testing data KNN(train on test) ####
> knn.test.p = knn(cell.train,cell.test,cl = cell.train$Churn,k = 31,prob =
TRUE)
> knn.test.c = knn(cell.train,cell.test,cell.train$Churn,k=31)
> knn.test.prob = attributes(knn.test.p)$prob
> knn.prob.df = data.frame(knn.test.prob,knn.test.c)
> knn.prob.df$knn.test.prob[knn.test.c == "0"] = 1 - knn.prob.df$knn.test.p
rob[knn.test.c == "0"]
> knn.test.predict = knn.prob.df$knn.test.prob
> knn.test.response = knn.prob.df$knn.test.c
> ### Confusion Matrix KNN (train on test) ###
> caret::confusionMatrix(knn.test.response,cell.test$Churn,positive = "1")
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 850 118
         1  12  20

               Accuracy : 0.87
                 95% CI : (0.8476, 0.8902)
    No Information Rate : 0.862
    P-Value [Acc > NIR] : 0.2477

                  Kappa : 0.1934

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.1449
```

```
            Specificity : 0.9861
         Pos Pred Value : 0.6250
         Neg Pred Value : 0.8781
             Prevalence : 0.1380
         Detection Rate : 0.0200
   Detection Prevalence : 0.0320
      Balanced Accuracy : 0.5655

       'Positive' Class : 1

> ### ROC Curve KNN(train on test) ###
> knn.test.obj = prediction(knn.test.predict,cell.test$Churn)
> knn.test.perf = performance(knn.test.obj,"tpr","fpr")
> plot(knn.test.perf)
> ### AUC Curve KNN(train on test) ###
> knn.test.auc = performance(knn.test.obj,"auc")
> knn.test.auc = as.numeric(knn.test.auc@y.values)
> print(knn.test.auc)
[1] 0.6646996
> ### KS Value KNN(train on test) ####
> print(max(knn.test.perf@y.values[[1]] - knn.test.perf@x.values[[1]]))
[1] 0.2659975
> ### GINI KNN(train on test) ####
> knn.test.gini = ineq(knn.test.predict,"gini")
> print(knn.test.gini)
[1] 0.4358475
> ### Concordance KNN (Train on Test) ###
> knn.test.x = cell.test$Churn
> knn.test.y = knn.test.predict
> Concordance(actuals = knn.test.x,predictedScores = knn.test.y)
$Concordance
[1] 0.6136387

$Discordance
[1] 0.3863613

$Tied
[1] 0

$Pairs
[1] 118956

> ### Plotting a graph between actuals vs. predicted ###
> plot(cell.test$Churn,knn.test.response,xlab = "Actuals",ylab = "Predicted
")
```