BA-PROJECT_GROUP-8

2023-04-25

Installing and Loading the required packages

```
library(ISLR)
library(caret)

## Loading required package: ggplot2

## Loading required package: lattice

library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(class)
library(e1071)
library(tidyverse)

## ── Attaching core tidyverse packages ──────────────────────── tidyverse
2.0.0 ──
## ✔ forcats   1.0.0     ✔ stringr   1.5.0
## ✔ lubridate 1.9.2     ✔ tibble    3.2.1
## ✔ purrr     1.0.1     ✔ tidyr     1.3.0
## ✔ readr     2.1.4

## ── Conflicts ──────────────────────────────────────────
tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ✖ purrr::lift()   masks caret::lift()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all
conflicts to become errors

library(ggplot2)
library(ggcorrplot)
library(rattle)
```

```
## Loading required package: bitops
## Rattle: A free graphical interface for data science with R.
## Version 5.5.1 Copyright (c) 2006-2021 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
library(gmodels)
library(modelr)
library(Hmisc)
```

```
##
## Attaching package: 'Hmisc'
##
## The following object is masked from 'package:e1071':
##
##     impute
##
## The following objects are masked from 'package:dplyr':
##
##     src, summarize
##
## The following objects are masked from 'package:base':
##
##     format.pval, units
```

```
library(missForest)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following object is masked from 'package:gmodels':
##
##     ci
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(ROCR)
library(cutpointr)
```

```
##
## Attaching package: 'cutpointr'
##
## The following objects are masked from 'package:pROC':
##
##     auc, roc
##
## The following objects are masked from 'package:caret':
```

```
##
##     precision, recall, sensitivity, specificity

library(ROSE)

## Loaded ROSE 0.0-4

library(cowplot)

##
## Attaching package: 'cowplot'
##
## The following object is masked from 'package:lubridate':
##
##     stamp

library(tinytex)
library(caTools)
library(broom)

##
## Attaching package: 'broom'
##
## The following object is masked from 'package:modelr':
##
##     bootstrap

library(rpart)
library(rpart.plot)
```

Loading the Churn dataset to R environment

```
#Included both csv file and .Rdata file in the same working directory where
Rmd file is saved
Raw_Churn_data<- read.csv("Churn_Train.csv")

load(file = "Customers_To_Predict.RData",envir = globalenv())
```

Attributes in the dataset

```
colnames(Raw_Churn_data)

##  [1] "state"                         "account_length"
##  [3] "area_code"                     "international_plan"
##  [5] "voice_mail_plan"               "number_vmail_messages"
##  [7] "total_day_minutes"             "total_day_calls"
##  [9] "total_day_charge"              "total_eve_minutes"
## [11] "total_eve_calls"               "total_eve_charge"
## [13] "total_night_minutes"           "total_night_calls"
## [15] "total_night_charge"            "total_intl_minutes"
## [17] "total_intl_calls"              "total_intl_charge"
## [19] "number_customer_service_calls" "churn"
```

```
str(Raw_Churn_data)

## 'data.frame':    3333 obs. of  20 variables:
##  $ state                        : chr  "NV" "HI" "DC" "HI" ...
##  $ account_length               : int  125 108 82 NA 83 89 135 28 86 65
...
##  $ area_code                    : chr  "area_code_510" "area_code_415"
"area_code_415" "area_code_408" ...
##  $ international_plan           : chr  "no" "no" "no" "no" ...
##  $ voice_mail_plan              : chr  "no" "no" "no" "yes" ...
##  $ number_vmail_messages        : int  0 0 0 30 0 0 0 0 0 0 ...
##  $ total_day_minutes            : num  2013 292 300 110 337 ...
##  $ total_day_calls              : int  99 99 109 71 120 81 81 87 115 137
...
##  $ total_day_charge             : num  28.7 49.6 51 18.8 57.4 ...
##  $ total_eve_minutes            : num  1108 221 181 182 227 ...
##  $ total_eve_calls              : int  107 93 100 108 116 74 114 92 112 83
...
##  $ total_eve_charge             : num  14.9 18.8 15.4 15.5 19.3 ...
##  $ total_night_minutes          : num  243 229 270 184 154 ...
##  $ total_night_calls            : int  92 110 73 88 114 120 82 112 95 111
...
##  $ total_night_charge           : num  10.95 10.31 12.15 8.27 6.93 ...
##  $ total_intl_minutes           : num  10.9 14 11.7 11 15.8 9.1 10.3 10.1
9.8 12.7 ...
##  $ total_intl_calls             : int  7 9 4 8 7 4 6 3 7 6 ...
##  $ total_intl_charge            : num  2.94 3.78 3.16 2.97 4.27 2.46 2.78
2.73 2.65 3.43 ...
##  $ number_customer_service_calls: int  0 2 0 2 0 1 1 3 2 4 ...
##  $ churn                        : chr  "no" "yes" "yes" "no" ...
```

Removing the columns that are not needed

```
Churn_Train_Data<- Raw_Churn_data[, -c(1:3)]
colnames(Churn_Train_Data)

##  [1] "international_plan"            "voice_mail_plan"
##  [3] "number_vmail_messages"         "total_day_minutes"
##  [5] "total_day_calls"               "total_day_charge"
##  [7] "total_eve_minutes"             "total_eve_calls"
##  [9] "total_eve_charge"              "total_night_minutes"
## [11] "total_night_calls"             "total_night_charge"
## [13] "total_intl_minutes"            "total_intl_calls"
## [15] "total_intl_charge"             "number_customer_service_calls"
## [17] "churn"
```

Converting international plan, voice mail plan and churn variables to binary

```
Churn_Train_Data$international_plan<-
ifelse(Churn_Train_Data$international_plan =="yes",1,0)
```

```
Churn_Train_Data$voice_mail_plan<- ifelse(Churn_Train_Data$voice_mail_plan
=="yes",1,0)

Churn_Train_Data$churn<- ifelse(Churn_Train_Data$churn =="yes",1,0)
```

Verify any NA values present in the dataset

```
any(is.na.data.frame(Churn_Train_Data))

## [1] TRUE

colMeans(is.na(Churn_Train_Data))*100

##             international_plan                voice_mail_plan
##                       0.000000                       0.000000
##          number_vmail_messages              total_day_minutes
##                       6.000600                       6.000600
##                total_day_calls               total_day_charge
##                       6.000600                       6.000600
##              total_eve_minutes                total_eve_calls
##                       9.030903                       6.000600
##               total_eve_charge            total_night_minutes
##                       6.000600                       6.000600
##              total_night_calls             total_night_charge
##                       0.000000                       6.000600
##             total_intl_minutes               total_intl_calls
##                       6.000600                       9.030903
##              total_intl_charge number_customer_service_calls
##                       6.000600                       6.000600
##                          churn
##                       0.000000
```

Review All the columns after data manipulation and attribute exclusion

```
summary(Churn_Train_Data)

##  international_plan voice_mail_plan  number_vmail_messages
total_day_minutes
##  Min.   :0.00000    Min.   :0.0000   Min.   :-10.000       Min.   :   0.0
##  1st Qu.:0.00000    1st Qu.:0.0000   1st Qu.:  0.000       1st Qu.: 149.3
##  Median :0.00000    Median :0.0000   Median :  0.000       Median : 190.5
##  Mean   :0.09691    Mean   :0.2766   Mean   :  7.333       Mean   : 418.9
##  3rd Qu.:0.00000    3rd Qu.:1.0000   3rd Qu.: 16.000       3rd Qu.: 237.8
##  Max.   :1.00000    Max.   :1.0000   Max.   : 51.000       Max.   :2185.1
##                                      NA's   :200           NA's   :200
##  total_day_calls total_day_charge total_eve_minutes total_eve_calls
##  Min.   :  0.0   Min.   : 0.00    Min.   :   0.0    Min.   :  0.0
##  1st Qu.: 87.0   1st Qu.:24.45    1st Qu.: 170.5    1st Qu.: 87.0
##  Median :101.0   Median :30.65    Median : 209.9    Median :100.0
##  Mean   :100.3   Mean   :30.63    Mean   : 324.3    Mean   :100.1
##  3rd Qu.:114.0   3rd Qu.:36.84    3rd Qu.: 257.6    3rd Qu.:114.0
```

```
##  Max.   :165.0   Max.   :59.64    Max.   :1244.2    Max.   :170.0
##  NA's   :200     NA's   :200      NA's   :301       NA's   :200
##  total_eve_charge total_night_minutes total_night_calls total_night_charge
##  Min.   : 0.00    Min.   : 23.2       Min.   : 33.0     Min.   : 1.040
##  1st Qu.:14.14    1st Qu.:167.3       1st Qu.: 87.0     1st Qu.: 7.530
##  Median :17.09    Median :201.4       Median :100.0     Median : 9.060
##  Mean   :17.08    Mean   :201.2       Mean   :100.1     Mean   : 9.054
##  3rd Qu.:20.00    3rd Qu.:235.3       3rd Qu.:113.0     3rd Qu.:10.590
##  Max.   :30.91    Max.   :395.0       Max.   :175.0     Max.   :17.770
##  NA's   :200      NA's   :200                           NA's   :200
##  total_intl_minutes total_intl_calls total_intl_charge
##  Min.   : 0.00      Min.   : 0.00    Min.   :0.000
##  1st Qu.: 8.50      1st Qu.: 3.00    1st Qu.:2.300
##  Median :10.30      Median : 4.00    Median :2.780
##  Mean   :10.23      Mean   : 4.47    Mean   :2.762
##  3rd Qu.:12.10      3rd Qu.: 6.00    3rd Qu.:3.270
##  Max.   :20.00      Max.   :20.00    Max.   :5.400
##  NA's   :200        NA's   :301      NA's   :200
##  number_customer_service_calls     churn
##  Min.   :0.000                 Min.   :0.0000
##  1st Qu.:1.000                 1st Qu.:0.0000
##  Median :1.000                 Median :0.0000
##  Mean   :1.561                 Mean   :0.1449
##  3rd Qu.:2.000                 3rd Qu.:0.0000
##  Max.   :9.000                 Max.   :1.0000
##  NA's   :200
```

Imputing the missing values with the medians of the columns as the mean value may be very sensitive to outliers.

```
#Treating the null values with median of the column.

Median_ofColumns<- apply(Churn_Train_Data,2,median, na.rm=T)
for (i in colnames(Churn_Train_Data))
Churn_Train_Data[,i][is.na(Churn_Train_Data[,i])]<- Median_ofColumns[i]

any(is.na.data.frame(Churn_Train_Data))#checking for any null values present
after imputation.

## [1] FALSE

str(Churn_Train_Data)

## 'data.frame':    3333 obs. of  17 variables:
##  $ international_plan           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ voice_mail_plan              : num  0 0 0 1 0 0 0 0 0 0 ...
##  $ number_vmail_messages        : num  0 0 0 30 0 0 0 0 0 0 ...
##  $ total_day_minutes            : num  2013 292 300 110 337 ...
##  $ total_day_calls              : num  99 99 109 71 120 81 81 87 115 137
...
##  $ total_day_charge             : num  28.7 49.6 51 18.8 57.4 ...
```

```
## $ total_eve_minutes : num 1108 221 181 182 227 ...
## $ total_eve_calls : num 107 93 100 108 116 74 114 92 112 83
...
## $ total_eve_charge : num 14.9 18.8 15.4 15.5 19.3 ...
## $ total_night_minutes : num 243 229 270 184 154 ...
## $ total_night_calls : num 92 110 73 88 114 120 82 112 95 111
...
## $ total_night_charge : num 10.95 10.31 12.15 8.27 6.93 ...
## $ total_intl_minutes : num 10.9 14 11.7 11 15.8 9.1 10.3 10.1
9.8 12.7 ...
## $ total_intl_calls : num 7 9 4 8 7 4 6 3 7 6 ...
## $ total_intl_charge : num 2.94 3.78 3.16 2.97 4.27 2.46 2.78
2.73 2.65 3.43 ...
## $ number_customer_service_calls: num 0 2 0 2 0 1 1 3 2 4 ...
## $ churn : num 0 1 1 0 1 0 0 0 0 1 ...
```

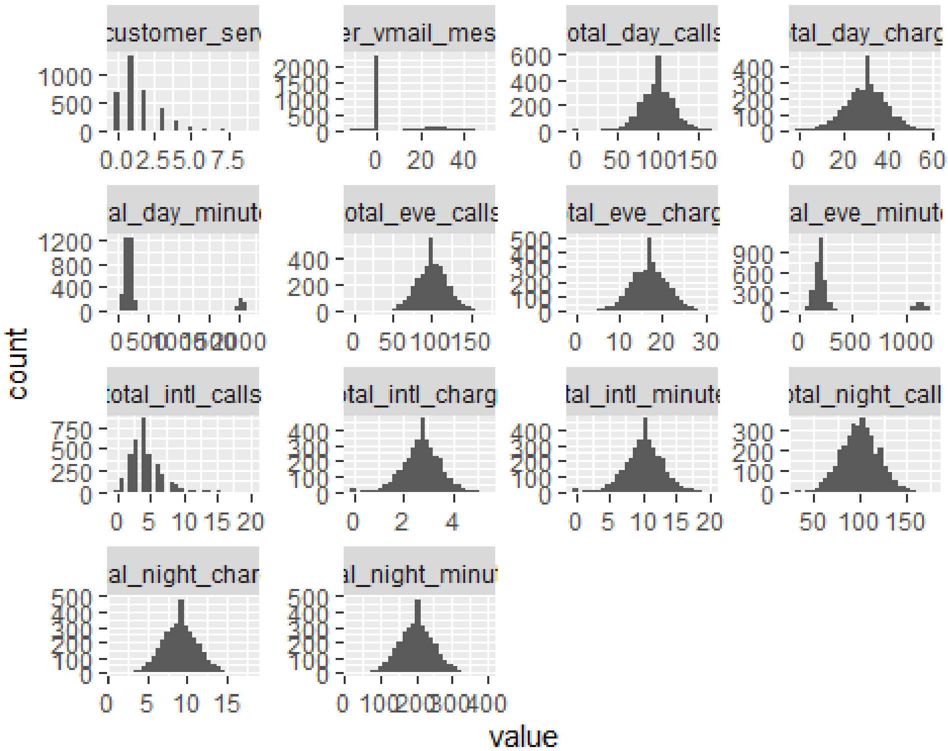
Treating the churn column as numbers can cause issues when using the column in certain functions or models that expect a factor variable.so we are converting the number to factor.

```
Churn_Train_Data$churn<- as.factor(Churn_Train_Data$churn)#converting the
integers to factors
Churn_Train_Data$international_plan<-
as.factor(Churn_Train_Data$international_plan)

Churn_Train_Data$voice_mail_plan<-
as.factor(Churn_Train_Data$voice_mail_plan)

#Churn_Train_Data$churn<-
#factor(Churn_Train_Data$churn,levels(Churn_Train_Data$churn)[c(2,1)])
#Changing the order of the churn factor levels.
```

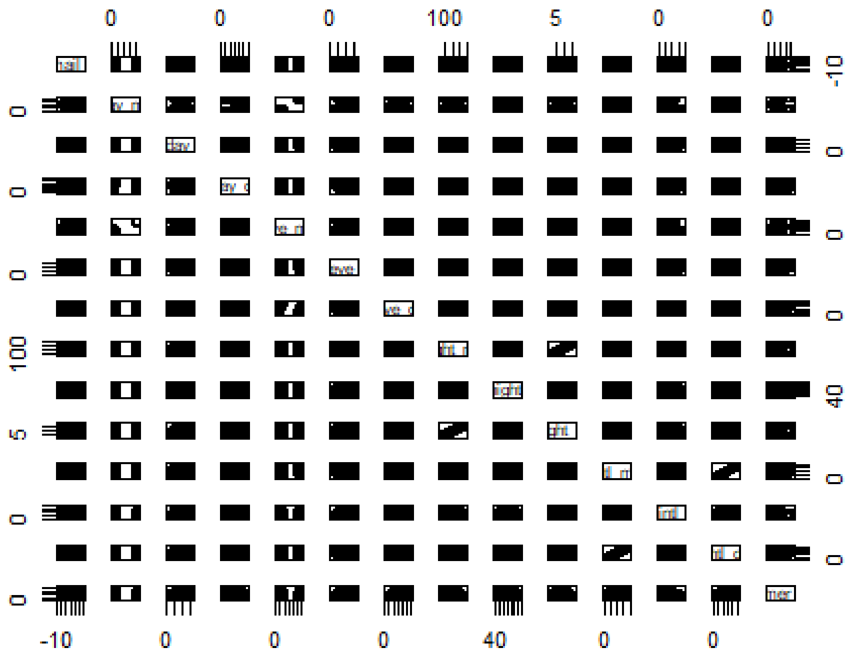
Exploratory Analysis

Perform Exploratory Analysis on Numerical Variables

```
# Explore the distribution of each variable using histograms
Churn_Train_Data %>%
  select_if(is.numeric) %>%
  gather() %>%
  ggplot(aes(value)) +
  facet_wrap(~key, scales = "free") +
  geom_histogram()

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
#View(Churn_Train)

# Explore the relationship between pairs of variables using scatter plots
pairs(  Churn_Train_Data %>%
  select_if(is.numeric))
```



Data Partition

Partitioning the Churn train data to training set of 75% and validation set of 25%.

```
set.seed(123)

Data_Partition<- createDataPartition(Churn_Train_Data$churn,p=0.75,list =
FALSE)
train_data<- Churn_Train_Data[Data_Partition,]

Validation_Data<- Churn_Train_Data[-Data_Partition,]
```

Function For Threshold Identification

Created a function to find the threshold cutoff that balances Sensitivity, Specificity and Accuracy

```
FindThreshold <- function(actual, predict) {
  # Create a sequence of decimal numbers from 0 to 1 with a step of 0.1
seq_decimal <- seq(from = 0.02, to = 1, by = 0.02)

df <- data.frame(Threshold=0,Sensitivity=0,Specificity=0,Accuracy=0)

rowNumber =1
# Iterate over the sequence using a for loop
for (i in seq_decimal) {
 predict_Lreg1<- ifelse(predict > i, 1, 0)

 x <- table(actual,predict_Lreg1)
```

```
 df[rowNumber,1]=i
 df[rowNumber,2]=(x[4]/(x[2]+x[4]))
 df[rowNumber,3]=x[1]/(x[1]+x[3])
 df[rowNumber,4]=(x[1]+x[4])/(x[2]+x[4]+x[1]+x[3])

 rowNumber=rowNumber+1
}
return(df)
}
```

Create Function to calculate Metrics for the table output

```
MetricCalculation <- function(x){

  metriclist <- list(r1=c(
    TrueNegative=as.integer(x[1]),
    TruePositive=as.integer(x[4]),
    FalsePositve=as.integer(x[3]),
    FalseNegative=as.integer(x[2])),
    r2=c(
     sensitivityVal=as.double((x[4]/(x[2]+x[4])),4),
                     SpecificityVal=round(x[1]/(x[1]+x[3]),4),
                      Accuracy=(x[1]+x[4])/(x[2]+x[4]+x[1]+x[3])))

  return(metriclist)
}
```

Logistic Regression

```
Logistic_Model<- glm(churn~.,data = train_data, family = "binomial")
summary(Logistic_Model)

##
## Call:
## glm(formula = churn ~ ., family = "binomial", data = train_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9657  -0.5058  -0.3404  -0.1929   3.0720
##
## Coefficients:
##                                Estimate Std. Error z value Pr(>|z|)
## (Intercept)                   -7.617943   0.831136  -9.166  < 2e-16 ***
## international_plan1            2.127728   0.164311  12.949  < 2e-16 ***
## voice_mail_plan1              -0.623079   0.341719  -1.823  0.06825 .
## number_vmail_messages         -0.012730   0.011631  -1.095  0.27372
## total_day_minutes             -0.005818   0.002195  -2.650  0.00805 **
## total_day_calls               -0.001189   0.003307  -0.360  0.71919
## total_day_charge               0.100688   0.014257   7.063 1.64e-12 ***
## total_eve_minutes              0.011087   0.004314   2.570  0.01016 *
## total_eve_calls               -0.003379   0.003332  -1.014  0.31043
## total_eve_charge              -0.025673   0.050623  -0.507  0.61206
```

```
## total_night_minutes            0.049366   1.004785   0.049  0.96081
## total_night_calls              0.002408   0.003290   0.732  0.46427
## total_night_charge            -1.050252  22.327548  -0.047  0.96248
## total_intl_minutes             6.366234   6.333047   1.005  0.31478
## total_intl_calls              -0.123040   0.031510  -3.905 9.43e-05 ***
## total_intl_charge            -23.166968  23.453465  -0.988  0.32326
## number_customer_service_calls  0.512369   0.045670  11.219  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2071.8  on 2500  degrees of freedom
## Residual deviance: 1614.7  on 2484  degrees of freedom
## AIC: 1648.7
##
## Number of Fisher Scoring iterations: 6
```

```
predict_Lreg<- predict(Logistic_Model,Validation_Data, type = "response")
head(predict_Lreg)
```

```
##          1          2          7         15         17         18
## 0.02029239 0.36320969 0.09553940 0.07374417 0.04320038 0.02844423
```

Find the cutoff value

To identify the right threshold using the function built to calculate sensitivity, specificity and accuracy by changing the threshold value for identifying churn yes or no.

```
df <- FindThreshold(Validation_Data$churn,predict_Lreg)
df
```

```
##    Threshold Sensitivity Specificity  Accuracy
## 1       0.02  0.94166667   0.1165730 0.2355769
## 2       0.04  0.90833333   0.2668539 0.3593750
## 3       0.06  0.87500000   0.4227528 0.4879808
## 4       0.08  0.83333333   0.5238764 0.5685096
## 5       0.10  0.80833333   0.6460674 0.6694712
## 6       0.12  0.76666667   0.6966292 0.7067308
## 7       0.14  0.71666667   0.7598315 0.7536058
## 8       0.16  0.66666667   0.7963483 0.7776442
## 9       0.18  0.63333333   0.8103933 0.7848558
## 10      0.20  0.56666667   0.8441011 0.8040865
## 11      0.22  0.52500000   0.8679775 0.8185096
## 12      0.24  0.49166667   0.8848315 0.8281250
## 13      0.26  0.45833333   0.9016854 0.8377404
## 14      0.28  0.41666667   0.9143258 0.8425481
## 15      0.30  0.40000000   0.9227528 0.8473558
## 16      0.32  0.38333333   0.9283708 0.8497596
## 17      0.34  0.35833333   0.9382022 0.8545673
## 18      0.36  0.30833333   0.9424157 0.8509615
```

```
## 19      0.38  0.26666667   0.9480337 0.8497596
## 20      0.40  0.26666667   0.9536517 0.8545673
## 21      0.42  0.24166667   0.9564607 0.8533654
## 22      0.44  0.22500000   0.9648876 0.8581731
## 23      0.46  0.21666667   0.9691011 0.8605769
## 24      0.48  0.20000000   0.9747191 0.8629808
## 25      0.50  0.16666667   0.9789326 0.8617788
## 26      0.52  0.15000000   0.9803371 0.8605769
## 27      0.54  0.14166667   0.9817416 0.8605769
## 28      0.56  0.14166667   0.9845506 0.8629808
## 29      0.58  0.12500000   0.9873596 0.8629808
## 30      0.60  0.11666667   0.9887640 0.8629808
## 31      0.62  0.11666667   0.9915730 0.8653846
## 32      0.64  0.11666667   0.9915730 0.8653846
## 33      0.66  0.10833333   0.9929775 0.8653846
## 34      0.68  0.10833333   0.9929775 0.8653846
## 35      0.70  0.09166667   0.9943820 0.8641827
## 36      0.72  0.07500000   0.9943820 0.8617788
## 37      0.74  0.06666667   0.9957865 0.8617788
## 38      0.76  0.05833333   0.9971910 0.8617788
## 39      0.78  0.05000000   0.9985955 0.8617788
## 40      0.80  0.04166667   0.9985955 0.8605769
## 41      0.82  0.03333333   0.9985955 0.8593750
## 42      0.84  0.02500000   0.9985955 0.8581731
## 43      0.86  0.01666667   0.9985955 0.8569712
## 44      0.88  0.01666667   0.9985955 0.8569712
## 45      0.90  0.00000000   0.9985955 0.8545673
## 46      0.92          NA          NA        NA
## 47      0.94          NA          NA        NA
## 48      0.96          NA          NA        NA
## 49      0.98          NA          NA        NA
## 50      1.00          NA          NA        NA
```

For Calculations we changed churn =Yes as 1 and churn=No as 0. Threshold 0.14 is selected after reviewing the threshold cut off table and metrics. Even though accuracy and Specificity are better with 0.16, we chose 0.14 as the customer churn(churn=Yes) costs telecom company more than the customers that would continue with the company(churn= No).

```
#print
df %>% filter(Threshold=="0.14")

##   Threshold Sensitivity Specificity  Accuracy
## 1      0.14   0.7166667   0.7598315 0.7536058
```

we can pick the best threshold value to balance the prediction.

```
lg_threshold <- 0.14

predict_Lreg1<- ifelse(predict_Lreg > lg_threshold, 1, 0)
```

```r
#predict_Lreg
tbl_Lreg <- table(Validation_Data$churn, predict_Lreg1)

tbl_Lreg
```

```
##    predict_Lreg1
##       0   1
##   0 541 171
##   1  34  86
```

```r
#finding the accuracy
missing_class<- mean(predict_Lreg1 != Validation_Data$churn)
print(paste('Accuracy=', 1 - missing_class))
```

```
## [1] "Accuracy= 0.753605769230769"
```

ROC-AUC

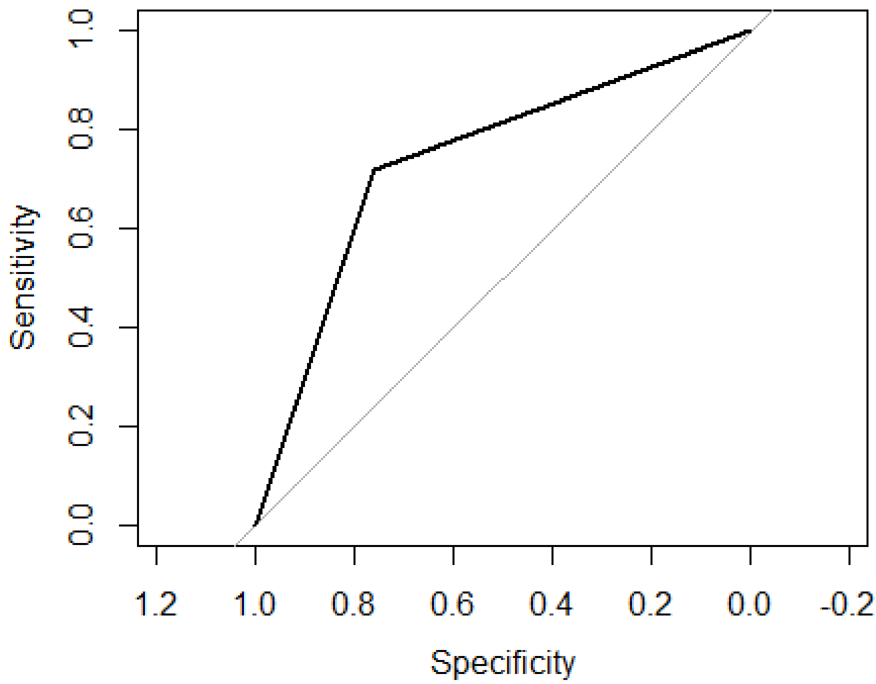
```r
ROC_Predict<- prediction(predict_Lreg1, Validation_Data$churn)
Roc_perform<- performance(ROC_Predict, measure = "tpr", x.measure = "fpr")

AUC_perform<- performance(ROC_Predict, measure = "auc")
AUC_perform<- AUC_perform@y.values[[1]]
AUC_perform
```

```
## [1] 0.7382491
```

Plot ROC Curve

```r
plot.roc(Validation_Data$churn,predict_Lreg1)
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```




```
Acc_perform<- performance(ROC_Predict, measure = "acc")
Acc_perform@y.values[[1]]

## [1] 0.8557692 0.7536058 0.1442308
```

Note: confusionMatrix function has provided sensitivity and specificity results in the reverse order

```
#printing the confusion matrix to see the prediction performance of the model
confusionMatrix(as.factor(predict_Lreg1),as.factor(Validation_Data$churn))

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 541  34
##          1 171  86
##
##                Accuracy : 0.7536
##                  95% CI : (0.7229, 0.7825)
##     No Information Rate : 0.8558
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3231
##
##  Mcnemar's Test P-Value : <2e-16
##
```

```
##             Sensitivity : 0.7598
##             Specificity : 0.7167
##          Pos Pred Value : 0.9409
##          Neg Pred Value : 0.3346
##              Prevalence : 0.8558
##          Detection Rate : 0.6502
##    Detection Prevalence : 0.6911
##       Balanced Accuracy : 0.7382
##
##        'Positive' Class : 0
##
```

```
CrossTable( Validation_Data$churn,predict_Lreg1,prop.chisq = F)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  832
##
##
##                       | predict_Lreg1
## Validation_Data$churn |         0 |         1 | Row Total |
## ----------------------|-----------|-----------|-----------|
##                     0 |       541 |       171 |       712 |
##                       |     0.760 |     0.240 |     0.856 |
##                       |     0.941 |     0.665 |           |
##                       |     0.650 |     0.206 |           |
## ----------------------|-----------|-----------|-----------|
##                     1 |        34 |        86 |       120 |
##                       |     0.283 |     0.717 |     0.144 |
##                       |     0.059 |     0.335 |           |
##                       |     0.041 |     0.103 |           |
## ----------------------|-----------|-----------|-----------|
##          Column Total |       575 |       257 |       832 |
##                       |     0.691 |     0.309 |           |
## ----------------------|-----------|-----------|-----------|
##
##
```

Snapshot of Final Metrics for the validation data using Logistic regression

```
MetricCalculation(tbl_Lreg)
```

```
## $r1
##  TrueNegative  TruePositive  FalsePositve FalseNegative
##           541            86           171            34
##
## $r2
## sensitivityVal SpecificityVal       Accuracy
##      0.7166667      0.7598000      0.7536058
```

KNN model

```
#set.seed(125)

KNN_Model<- knn(train = train_data[,1:17],test =Validation_Data[,1:17], cl=
train_data$churn,
                k=60 ,prob = TRUE )


probs <- attr(KNN_Model,"prob")

dfWithProbYes <- data.frame(KNN_Model,probs)

dfWithProbYes <- ifelse(dfWithProbYes$KNN_Model==1,dfWithProbYes$probs,1-
dfWithProbYes$probs)

head(dfWithProbYes)

## [1] 0.20000000 0.58333333 0.09523810 0.10000000 0.14516129 0.08196721
```

Find the cutoff value

```
df_knn <- FindThreshold(Validation_Data$churn,dfWithProbYes)
df_knn

##    Threshold Sensitivity Specificity  Accuracy
## 1       0.02 1.000000000  0.01264045 0.1550481
## 2       0.04 0.983333333  0.04073034 0.1766827
## 3       0.06 0.958333333  0.09269663 0.2175481
## 4       0.08 0.866666667  0.19943820 0.2956731
## 5       0.10 0.750000000  0.46207865 0.5036058
## 6       0.12 0.658333333  0.61095506 0.6177885
## 7       0.14 0.608333333  0.70365169 0.6899038
## 8       0.16 0.541666667  0.79213483 0.7560096
## 9       0.18 0.475000000  0.87500000 0.8173077
## 10      0.20 0.425000000  0.93960674 0.8653846
## 11      0.22 0.416666667  0.95224719 0.8750000
## 12      0.24 0.416666667  0.96207865 0.8834135
## 13      0.26 0.408333333  0.97752809 0.8954327
## 14      0.28 0.391666667  0.97893258 0.8942308
## 15      0.30 0.358333333  0.98595506 0.8954327
## 16      0.32 0.333333333  0.98735955 0.8930288
## 17      0.34 0.300000000  0.99157303 0.8918269
## 18      0.36 0.283333333  0.99297753 0.8906250
```

```
## 19      0.38 0.258333333  0.99438202 0.8882212
## 20      0.40 0.166666667  1.00000000 0.8798077
## 21      0.42 0.158333333  1.00000000 0.8786058
## 22      0.44 0.150000000  1.00000000 0.8774038
## 23      0.46 0.133333333  1.00000000 0.8750000
## 24      0.48 0.125000000  1.00000000 0.8737981
## 25      0.50 0.116666667  1.00000000 0.8725962
## 26      0.52 0.100000000  1.00000000 0.8701923
## 27      0.54 0.083333333  1.00000000 0.8677885
## 28      0.56 0.083333333  1.00000000 0.8677885
## 29      0.58 0.066666667  1.00000000 0.8653846
## 30      0.60 0.050000000  1.00000000 0.8629808
## 31      0.62 0.033333333  1.00000000 0.8605769
## 32      0.64 0.025000000  1.00000000 0.8593750
## 33      0.66 0.025000000  1.00000000 0.8593750
## 34      0.68 0.025000000  1.00000000 0.8593750
## 35      0.70 0.016666667  1.00000000 0.8581731
## 36      0.72 0.008333333  1.00000000 0.8569712
## 37      0.74          NA          NA        NA
## 38      0.76          NA          NA        NA
## 39      0.78          NA          NA        NA
## 40      0.80          NA          NA        NA
## 41      0.82          NA          NA        NA
## 42      0.84          NA          NA        NA
## 43      0.86          NA          NA        NA
## 44      0.88          NA          NA        NA
## 45      0.90          NA          NA        NA
## 46      0.92          NA          NA        NA
## 47      0.94          NA          NA        NA
## 48      0.96          NA          NA        NA
## 49      0.98          NA          NA        NA
## 50      1.00          NA          NA        NA
```

Threshold selection from the combination found in the df_knn

```
#print
df_knn %>% filter(Threshold=="0.14")

##   Threshold Sensitivity Specificity  Accuracy
## 1      0.14   0.6083333   0.7036517 0.6899038
```

we can pick the best threshold value to balance the prediction using the combination of sensitivity, specificity and accuracy.

```
knn_threshold <- 0.14

predict_knn<- ifelse(dfWithProbYes > knn_threshold, 1, 0)
#predict_Lreg
tbl_knn <- table(Validation_Data$churn, predict_knn)

tbl_knn
```

```
##    predict_knn
##       0   1
##   0 501 211
##   1  47  73
```

Note: confusionMatrix function has provided sensitivity and specificity results in the reverse order

```
confusionMatrix(as.factor(predict_knn),Validation_Data$churn)

## Confusion Matrix and Statistics
## 
##           Reference
## Prediction   0   1
##          0 501  47
##          1 211  73
## 
##                Accuracy : 0.6899
##                  95% CI : (0.6572, 0.7212)
##     No Information Rate : 0.8558
##     P-Value [Acc > NIR] : 1
## 
##                   Kappa : 0.1989
## 
##  Mcnemar's Test P-Value : <2e-16
## 
##             Sensitivity : 0.7037
##             Specificity : 0.6083
##          Pos Pred Value : 0.9142
##          Neg Pred Value : 0.2570
##              Prevalence : 0.8558
##          Detection Rate : 0.6022
##    Detection Prevalence : 0.6587
##       Balanced Accuracy : 0.6560
## 
##        'Positive' Class : 0
## 

CrossTable(Validation_Data$churn,predict_knn)

## 
## 
##    Cell Contents
## |-------------------------|
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
## 
```

```
##
## Total Observations in Table:  832
##
##
##                        | predict_knn
## Validation_Data$churn |         0 |         1 | Row Total |
## ----------------------|-----------|-----------|-----------|
##                     0 |       501 |       211 |       712 |
##                       |     2.189 |     4.223 |           |
##                       |     0.704 |     0.296 |     0.856 |
##                       |     0.914 |     0.743 |           |
##                       |     0.602 |     0.254 |           |
## ----------------------|-----------|-----------|-----------|
##                     1 |        47 |        73 |       120 |
##                       |    12.987 |    25.059 |           |
##                       |     0.392 |     0.608 |     0.144 |
##                       |     0.086 |     0.257 |           |
##                       |     0.056 |     0.088 |           |
## ----------------------|-----------|-----------|-----------|
##          Column Total |       548 |       284 |       832 |
##                       |     0.659 |     0.341 |           |
## ----------------------|-----------|-----------|-----------|
##
##
```

```r
MetricCalculation(tbl_knn)
```

```
## $r1
##  TrueNegative  TruePositive  FalsePositve FalseNegative
##           501            73           211            47
##
## $r2
## sensitivityVal SpecificityVal       Accuracy
##      0.6083333      0.7037000      0.6899038
```

Decision Tree Model

Build and Prune Decision tree using the best CP value

```r
Dec_tree_model<- rpart(churn~., data = train_data, method = "class",
                       control = rpart.control(minsplit = 40))

#Find the best cp value for pruning decision tree
Best_cp<-
Dec_tree_model$cptable[which.min(Dec_tree_model$cptable[,"xerror"]),"CP"]

#Prune decision tree based on the cp value identified by building base model
Dec_tree_model <- prune(Dec_tree_model, cp=Best_cp)

summary(Dec_tree_model)
```

```
## Call:
## rpart(formula = churn ~ ., data = train_data, method = "class",
##     control = rpart.control(minsplit = 40))
##   n= 2501
##
##           CP nsplit rel error    xerror       xstd
## 1 0.07713499      0 1.0000000 1.0000000 0.04852815
## 2 0.05647383      4 0.6914601 0.8099174 0.04437224
## 3 0.03030303      7 0.4931129 0.5399449 0.03702542
## 4 0.01652893      8 0.4628099 0.5261708 0.03658973
## 5 0.01101928     10 0.4297521 0.5179063 0.03632480
## 6 0.01000000     11 0.4187328 0.5151515 0.03623590
##
## Variable importance
##              total_day_charge number_customer_service_calls
##                            21                            12
##            international_plan             total_intl_charge
##                             9                             9
##            total_intl_minutes             total_day_minutes
##                             9                             8
##              total_intl_calls              total_eve_charge
##                             8                             8
##             total_eve_minutes               voice_mail_plan
##                             5                             5
##         number_vmail_messages            total_night_charge
##                             5                             1
##           total_night_minutes                total_day_calls
##                             1                             1
##
## Node number 1: 2501 observations,    complexity param=0.07713499
##   predicted class=0  expected loss=0.1451419  P(node) =1
##     class counts:  2138   363
##    probabilities: 0.855 0.145
##   left son=2 (2355 obs) right son=3 (146 obs)
##   Primary splits:
##       total_day_charge              < 44.985  to the left,
improve=59.233520, (0 missing)
##       number_customer_service_calls < 3.5     to the left,
improve=56.684250, (0 missing)
##       international_plan            splits as  LR,
improve=48.042920, (0 missing)
##       total_day_minutes             < 223.25  to the left,
improve=16.822220, (0 missing)
##       voice_mail_plan               splits as  RL,          improve=
7.072234, (0 missing)
##
## Node number 2: 2355 observations,    complexity param=0.07713499
##   predicted class=0  expected loss=0.1180467  P(node) =0.9416234
##     class counts:  2077   278
##    probabilities: 0.882 0.118
```

```
##   left son=4 (2168 obs) right son=5 (187 obs)
##   Primary splits:
##       number_customer_service_calls < 3.5     to the left,
improve=60.101180, (0 missing)
##       international_plan            splits as  LR,
improve=46.007340, (0 missing)
##       total_day_charge              < 37.95   to the left,  improve=
7.622099, (0 missing)
##       total_intl_calls              < 2.5     to the right, improve=
6.770627, (0 missing)
##       total_intl_minutes            < 13.15   to the left,  improve=
6.020005, (0 missing)
##   Surrogate splits:
##       total_day_calls < 38      to the right, agree=0.921, adj=0.005, (0
split)
##
## Node number 3: 146 observations,    complexity param=0.07713499
##   predicted class=1  expected loss=0.4178082  P(node) =0.05837665
##     class counts:    61    85
##    probabilities: 0.418 0.582
##   left son=6 (38 obs) right son=7 (108 obs)
##   Primary splits:
##       voice_mail_plan       splits as  RL,          improve=23.369500, (0
missing)
##       number_vmail_messages < 6.5     to the right, improve=22.277580, (0
missing)
##       total_eve_minutes     < 185.45  to the left,  improve=12.985400, (0
missing)
##       total_eve_charge      < 17.075  to the left,  improve=11.506940, (0
missing)
##       total_day_charge      < 53.84   to the left,  improve= 5.836558, (0
missing)
##   Surrogate splits:
##       number_vmail_messages < 6.5     to the right, agree=0.993,
adj=0.974, (0 split)
##       total_eve_minutes     < 1126.15 to the right, agree=0.767,
adj=0.105, (0 split)
##       total_day_minutes     < 2080.3  to the right, agree=0.760,
adj=0.079, (0 split)
##       total_night_minutes   < 119.55  to the left,  agree=0.760,
adj=0.079, (0 split)
##       total_night_charge    < 5.38    to the left,  agree=0.760,
adj=0.079, (0 split)
##
## Node number 4: 2168 observations,    complexity param=0.05647383
##   predicted class=0  expected loss=0.08487085  P(node) =0.8668533
##     class counts:  1984   184
##    probabilities: 0.915 0.085
##   left son=8 (1952 obs) right son=9 (216 obs)
##   Primary splits:
```

```
##       international_plan splits as  LR,          improve=45.707680, (0
missing)
##       total_day_charge   < 37.95   to the left,  improve=12.804600, (0
missing)
##       total_eve_charge   < 21.175  to the left,  improve= 6.107379, (0
missing)
##       total_intl_calls   < 2.5     to the right, improve= 4.941350, (0
missing)
##       total_intl_minutes < 13.15   to the left,  improve= 4.925431, (0
missing)
##   Surrogate splits:
##       total_eve_minutes < 1234.3  to the left,  agree=0.901, adj=0.009, (0
split)
##
## Node number 5: 187 observations,    complexity param=0.07713499
##   predicted class=1  expected loss=0.4973262  P(node) =0.07477009
##     class counts:    93    94
##    probabilities: 0.497 0.503
##   left son=10 (109 obs) right son=11 (78 obs)
##   Primary splits:
##       total_day_charge             < 27.685  to the right,
improve=36.465100, (0 missing)
##       total_day_minutes            < 162.85  to the right,
improve=32.460370, (0 missing)
##       total_eve_minutes            < 264.65  to the right, improve=
7.583369, (0 missing)
##       total_eve_charge             < 15.825  to the right, improve=
6.881493, (0 missing)
##       number_customer_service_calls < 4.5     to the left,  improve=
4.344669, (0 missing)
##   Surrogate splits:
##       total_day_minutes   < 162.85  to the right, agree=0.968, adj=0.923,
(0 split)
##       total_intl_calls    < 2.5     to the right, agree=0.604, adj=0.051,
(0 split)
##       total_eve_calls     < 86.5    to the right, agree=0.599, adj=0.038,
(0 split)
##       total_night_minutes < 91.15   to the right, agree=0.599, adj=0.038,
(0 split)
##       total_night_charge  < 4.1     to the right, agree=0.599, adj=0.038,
(0 split)
##
## Node number 6: 38 observations
##   predicted class=0  expected loss=0.1052632  P(node) =0.01519392
##     class counts:    34     4
##    probabilities: 0.895 0.105
##
## Node number 7: 108 observations,    complexity param=0.03030303
##   predicted class=1  expected loss=0.25  P(node) =0.04318273
##     class counts:    27    81
```

```
##    probabilities: 0.250 0.750
##   left son=14 (33 obs) right son=15 (75 obs)
##   Primary splits:
##       total_eve_minutes   < 183.75  to the left,  improve=16.500000, (0
missing)
##       total_eve_charge    < 15.695  to the left,  improve=15.564710, (0
missing)
##       total_night_minutes < 169.6   to the left,  improve= 3.976364, (0
missing)
##       total_night_charge  < 7.63    to the left,  improve= 3.976364, (0
missing)
##       total_day_charge    < 47.34   to the left,  improve= 3.146897, (0
missing)
##   Surrogate splits:
##       total_eve_charge   < 15.545  to the left,  agree=0.944, adj=0.818,
(0 split)
##       total_intl_minutes < 3.35    to the left,  agree=0.713, adj=0.061,
(0 split)
##       total_intl_charge  < 0.905   to the left,  agree=0.713, adj=0.061,
(0 split)
##       total_eve_calls    < 117.5   to the right, agree=0.704, adj=0.030,
(0 split)
##       total_intl_calls   < 1.5     to the left,  agree=0.704, adj=0.030,
(0 split)
##
## Node number 8: 1952 observations,    complexity param=0.01652893
##   predicted class=0  expected loss=0.05071721  P(node) =0.7804878
##     class counts:  1853    99
##    probabilities: 0.949 0.051
##   left son=16 (1699 obs) right son=17 (253 obs)
##   Primary splits:
##       total_day_charge    < 37.95   to the left,  improve=9.398504, (0
missing)
##       total_eve_charge    < 20.855  to the left,  improve=4.728740, (0
missing)
##       total_day_minutes   < 223.25  to the left,  improve=3.461978, (0
missing)
##       total_eve_minutes   < 208.85  to the left,  improve=2.903298, (0
missing)
##       total_night_minutes < 191.3   to the left,  improve=1.369124, (0
missing)
##   Surrogate splits:
##       total_day_minutes < 223.25  to the left,  agree=0.894, adj=0.182, (0
split)
##
## Node number 9: 216 observations,    complexity param=0.05647383
##   predicted class=0  expected loss=0.3935185  P(node) =0.08636545
##     class counts:   131    85
##    probabilities: 0.606 0.394
##   left son=18 (175 obs) right son=19 (41 obs)
```

```
##   Primary splits:
##       total_intl_calls   < 2.5     to the right, improve=37.227570, (0
missing)
##       total_intl_minutes < 13.05   to the left,  improve=30.726160, (0
missing)
##       total_intl_charge  < 3.52    to the left,  improve=30.726160, (0
missing)
##       total_eve_calls    < 107.5   to the right, improve= 3.296622, (0
missing)
##       total_night_calls  < 74.5    to the left,  improve= 3.132155, (0
missing)
##   Surrogate splits:
##       total_day_calls < 49      to the right, agree=0.819, adj=0.049, (0
split)
##
## Node number 10: 109 observations,    complexity param=0.01101928
##   predicted class=0  expected loss=0.2385321  P(node) =0.04358257
##     class counts:    83    26
##    probabilities: 0.761 0.239
##   left son=20 (95 obs) right son=21 (14 obs)
##   Primary splits:
##       total_eve_charge  < 12      to the right, improve=5.251969, (0
missing)
##       total_eve_minutes < 144.55  to the right, improve=4.192484, (0
missing)
##       total_day_charge  < 29.88   to the right, improve=3.350628, (0
missing)
##       total_day_minutes < 195.2   to the right, improve=2.353577, (0
missing)
##       total_night_calls < 116.5   to the left,  improve=1.878798, (0
missing)
##   Surrogate splits:
##       total_eve_minutes < 141.15  to the right, agree=0.972, adj=0.786, (0
split)
##
## Node number 11: 78 observations
##   predicted class=1  expected loss=0.1282051  P(node) =0.03118752
##     class counts:    10    68
##    probabilities: 0.128 0.872
##
## Node number 14: 33 observations
##   predicted class=0  expected loss=0.3333333  P(node) =0.01319472
##     class counts:    22    11
##    probabilities: 0.667 0.333
##
## Node number 15: 75 observations
##   predicted class=1  expected loss=0.06666667  P(node) =0.029988
##     class counts:     5    70
##    probabilities: 0.067 0.933
##
```

```
## Node number 16: 1699 observations
##   predicted class=0  expected loss=0.0317834  P(node) =0.6793283
##     class counts:  1645    54
##    probabilities: 0.968 0.032
##
## Node number 17: 253 observations,    complexity param=0.01652893
##   predicted class=0  expected loss=0.1778656  P(node) =0.1011595
##     class counts:   208    45
##    probabilities: 0.822 0.178
##   left son=34 (217 obs) right son=35 (36 obs)
##   Primary splits:
##       total_eve_charge      < 22.08   to the left,  improve=20.056610, (0
missing)
##       total_eve_minutes     < 240.15  to the left,  improve=12.632570, (0
missing)
##       voice_mail_plan       splits as  RL,          improve= 3.785385, (0
missing)
##       number_vmail_messages < 5.5     to the right, improve= 3.582237, (0
missing)
##       total_night_minutes   < 181.15  to the left,  improve= 3.015811, (0
missing)
##   Surrogate splits:
##       total_eve_minutes  < 259.8   to the left,  agree=0.877, adj=0.139,
(0 split)
##       total_day_calls    < 135.5   to the left,  agree=0.862, adj=0.028,
(0 split)
##       total_intl_minutes < 2.05    to the right, agree=0.862, adj=0.028,
(0 split)
##       total_intl_charge  < 0.555   to the right, agree=0.862, adj=0.028,
(0 split)
##
## Node number 18: 175 observations,    complexity param=0.05647383
##   predicted class=0  expected loss=0.2514286  P(node) =0.06997201
##     class counts:   131    44
##    probabilities: 0.749 0.251
##   left son=36 (144 obs) right son=37 (31 obs)
##   Primary splits:
##       total_intl_minutes  < 13.05   to the left,  improve=42.221510, (0
missing)
##       total_intl_charge   < 3.52    to the left,  improve=42.221510, (0
missing)
##       total_eve_charge    < 14.11   to the left,  improve= 2.860896, (0
missing)
##       total_night_minutes < 216.7   to the right, improve= 2.472771, (0
missing)
##       total_night_charge  < 9.75    to the right, improve= 2.472771, (0
missing)
##   Surrogate splits:
##       total_intl_charge < 3.52    to the left,  agree=1.000, adj=1.000, (0
split)
```

```
##       total_day_minutes < 55.3    to the right, agree=0.834, adj=0.065, (0
split)
##       total_day_charge  < 8.92    to the right, agree=0.829, adj=0.032, (0
split)
##
## Node number 19: 41 observations
##   predicted class=1  expected loss=0  P(node) =0.01639344
##     class counts:     0    41
##    probabilities: 0.000 1.000
##
## Node number 20: 95 observations
##   predicted class=0  expected loss=0.1789474  P(node) =0.03798481
##     class counts:    78    17
##    probabilities: 0.821 0.179
##
## Node number 21: 14 observations
##   predicted class=1  expected loss=0.3571429  P(node) =0.005597761
##     class counts:     5     9
##    probabilities: 0.357 0.643
##
## Node number 34: 217 observations
##   predicted class=0  expected loss=0.09677419  P(node) =0.08676529
##     class counts:   196    21
##    probabilities: 0.903 0.097
##
## Node number 35: 36 observations
##   predicted class=1  expected loss=0.3333333  P(node) =0.01439424
##     class counts:    12    24
##    probabilities: 0.333 0.667
##
## Node number 36: 144 observations
##   predicted class=0  expected loss=0.09027778  P(node) =0.05757697
##     class counts:   131    13
##    probabilities: 0.910 0.090
##
## Node number 37: 31 observations
##   predicted class=1  expected loss=0  P(node) =0.01239504
##     class counts:     0    31
##    probabilities: 0.000 1.000

#Probability prediction
Probability_DecisionTree <- predict(Dec_tree_model, newdata =
Validation_Data, type = "prob")

#calculating AUC Value
ROC_Predict_dt<- prediction(Probability_DecisionTree[,2],
Validation_Data$churn)
Roc_perform_dt<- performance(ROC_Predict_dt, measure = "tpr", x.measure =
"fpr")
```

```r
AUC_perform_dt<- performance(ROC_Predict_dt, measure = "auc")
AUC_perform_dt<- AUC_perform_dt@y.values[[1]]
AUC_perform_dt

## [1] 0.875357

df_dt <- FindThreshold(Validation_Data$churn,Probability_DecisionTree[,2] )
df_dt

##    Threshold Sensitivity Specificity  Accuracy
## 1       0.02          NA          NA        NA
## 2       0.04   0.8416667   0.7640449 0.7752404
## 3       0.06   0.8416667   0.7640449 0.7752404
## 4       0.08   0.8416667   0.7640449 0.7752404
## 5       0.10   0.7166667   0.9283708 0.8978365
## 6       0.12   0.7000000   0.9452247 0.9098558
## 7       0.14   0.7000000   0.9452247 0.9098558
## 8       0.16   0.7000000   0.9452247 0.9098558
## 9       0.18   0.6666667   0.9719101 0.9278846
## 10      0.20   0.6666667   0.9719101 0.9278846
## 11      0.22   0.6666667   0.9719101 0.9278846
## 12      0.24   0.6666667   0.9719101 0.9278846
## 13      0.26   0.6666667   0.9719101 0.9278846
## 14      0.28   0.6666667   0.9719101 0.9278846
## 15      0.30   0.6666667   0.9719101 0.9278846
## 16      0.32   0.6666667   0.9719101 0.9278846
## 17      0.34   0.6166667   0.9775281 0.9254808
## 18      0.36   0.6166667   0.9775281 0.9254808
## 19      0.38   0.6166667   0.9775281 0.9254808
## 20      0.40   0.6166667   0.9775281 0.9254808
## 21      0.42   0.6166667   0.9775281 0.9254808
## 22      0.44   0.6166667   0.9775281 0.9254808
## 23      0.46   0.6166667   0.9775281 0.9254808
## 24      0.48   0.6166667   0.9775281 0.9254808
## 25      0.50   0.6166667   0.9775281 0.9254808
## 26      0.52   0.6166667   0.9775281 0.9254808
## 27      0.54   0.6166667   0.9775281 0.9254808
## 28      0.56   0.6166667   0.9775281 0.9254808
## 29      0.58   0.6166667   0.9775281 0.9254808
## 30      0.60   0.6166667   0.9775281 0.9254808
## 31      0.62   0.6166667   0.9775281 0.9254808
## 32      0.64   0.6166667   0.9775281 0.9254808
## 33      0.66   0.5916667   0.9775281 0.9218750
## 34      0.68   0.5333333   0.9845506 0.9194712
## 35      0.70   0.5333333   0.9845506 0.9194712
## 36      0.72   0.5333333   0.9845506 0.9194712
## 37      0.74   0.5333333   0.9845506 0.9194712
## 38      0.76   0.5333333   0.9845506 0.9194712
## 39      0.78   0.5333333   0.9845506 0.9194712
## 40      0.80   0.5333333   0.9845506 0.9194712
```

```
## 41      0.82   0.5333333   0.9845506 0.9194712
## 42      0.84   0.5333333   0.9845506 0.9194712
## 43      0.86   0.5333333   0.9845506 0.9194712
## 44      0.88   0.3750000   0.9929775 0.9038462
## 45      0.90   0.3750000   0.9929775 0.9038462
## 46      0.92   0.3750000   0.9929775 0.9038462
## 47      0.94   0.1333333   1.0000000 0.8750000
## 48      0.96   0.1333333   1.0000000 0.8750000
## 49      0.98   0.1333333   1.0000000 0.8750000
## 50      1.00          NA          NA        NA
```

we can pick the best threshold value to balance the prediction.

So, 0.10 is selected which has better sensitivity and specificity combined resulted in better accuracy as well.

```
dt_threshold <- 0.10
predict_dt<- ifelse(Probability_DecisionTree[,2] > dt_threshold, 1, 0)

tbl_dt <- table(Validation_Data$churn, predict_dt)

tbl_dt

##    predict_dt
##       0   1
##   0 661  51
##   1  34  86
```

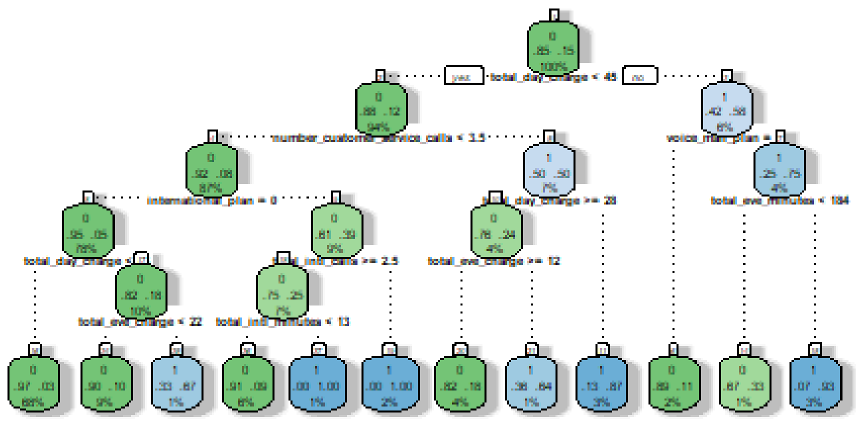
Note: confusionMatrix function has provided sensitivity and specificity results in the reverse order

```
#printing the confusion matrix to see the prediction performance of the model
confusionMatrix(as.factor(predict_dt),as.factor(Validation_Data$churn))

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 661  34
##          1  51  86
##
##                Accuracy : 0.8978
##                  95% CI : (0.8752, 0.9176)
##     No Information Rate : 0.8558
##     P-Value [Acc > NIR] : 0.0001938
##
##                   Kappa : 0.6092
##
##  Mcnemar's Test P-Value : 0.0826623
##
##             Sensitivity : 0.9284
```

```
##               Specificity : 0.7167
##            Pos Pred Value : 0.9511
##            Neg Pred Value : 0.6277
##                Prevalence : 0.8558
##            Detection Rate : 0.7945
##      Detection Prevalence : 0.8353
##         Balanced Accuracy : 0.8225
##
##          'Positive' Class : 0
##
```

Plot Decision Tree Model

```
fancyRpartPlot(Dec_tree_model,cex=0.4)
```



Rattle 2023-Apr-30 22:28:20 peddi

```
CrossTable(Validation_Data$churn,predict_dt,prop.chisq = F)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
```

```
## Total Observations in Table:  832
##
##
##                      | predict_dt
## Validation_Data$churn |         0 |         1 | Row Total |
## ----------------------|-----------|-----------|-----------|
##                     0 |       661 |        51 |       712 |
##                       |     0.928 |     0.072 |     0.856 |
##                       |     0.951 |     0.372 |           |
##                       |     0.794 |     0.061 |           |
## ----------------------|-----------|-----------|-----------|
##                     1 |        34 |        86 |       120 |
##                       |     0.283 |     0.717 |     0.144 |
##                       |     0.049 |     0.628 |           |
##                       |     0.041 |     0.103 |           |
## ----------------------|-----------|-----------|-----------|
##          Column Total |       695 |       137 |       832 |
##                       |     0.835 |     0.165 |           |
## ----------------------|-----------|-----------|-----------|
##
##
```

```
MetricCalculation(tbl_dt)
```

```
## $r1
##  TrueNegative  TruePositive  FalsePositve FalseNegative
##           661            86            51            34
##
## $r2
## sensitivityVal SpecificityVal       Accuracy
##      0.7166667      0.9284000      0.8978365
```

Metrics output for model selection

```
data.frame("Prediction Models"= c("Logistic Regression","KNN","Decision
Tree")

,Sensitivity=c(MetricCalculation(tbl_Lreg)[[2]][1],MetricCalculation(tbl_knn)
[[2]][1],MetricCalculation(tbl_dt)[[2]][1])

,Specificity=c(MetricCalculation(tbl_Lreg)[[2]][2],MetricCalculation(tbl_knn)
[[2]][2],MetricCalculation(tbl_dt)[[2]][2])

,Accuracy=c(MetricCalculation(tbl_Lreg)[[2]][3],MetricCalculation(tbl_knn)[[2
]][3],MetricCalculation(tbl_dt)[[2]][3])
           )
```

```
##     Prediction.Models Sensitivity Specificity  Accuracy
## 1 Logistic Regression   0.7166667      0.7598 0.7536058
## 2                 KNN   0.6083333      0.7037 0.6899038
## 3       Decision Tree   0.7166667      0.9284 0.8978365
```

Model Selection

After comparing the sensitivity, specificity, and accuracy metrics of the three models built in the project, the Decision tree and Logistic regression have the best sensitivity. Even though decision tree and logistic regression have similar sensitivity, the specificity and accuracy of logistic regression are lower, which could cost the telecom company significantly higher costs as the target customers for reducing churn will increase if specificity is low. So, the Decision tree has the best sensitivity, specificity, and accuracy, resulting in lower marketing costs to retain ABC telecom customers that are more likely to churn.

Decision Tree is the best model of three models built as part of the project.

Predict Churn for Customers_To_Predict

```
#finding the number of rows in the customers to predict dataset.
count(Customers_To_Predict)
```

```
## # A tibble: 1 × 1
##       n
##   <int>
## 1  1600
```

```
#printing the summary of the testset
summary(Customers_To_Predict)
```

```
##     state           account_length    area_code          international_plan
##  Length:1600        Min.   :  1.00   Length:1600        Length:1600
##  Class :character   1st Qu.: 71.00   Class :character   Class :character
##  Mode  :character   Median : 98.00   Mode  :character   Mode  :character
##                     Mean   : 98.52
##                     3rd Qu.:126.00
##                     Max.   :238.00
##  voice_mail_plan    number_vmail_messages total_day_minutes
total_day_calls
##  Length:1600        Min.   : 0.000        Min.   :  6.6     Min.   : 34.00
##  Class :character   1st Qu.: 0.000        1st Qu.:143.8     1st Qu.: 86.00
##  Mode  :character   Median : 0.000        Median :180.9     Median : 99.00
##                     Mean   : 7.043        Mean   :181.6     Mean   : 99.06
##                     3rd Qu.: 0.000        3rd Qu.:215.9     3rd Qu.:112.00
##                     Max.   :52.000        Max.   :351.5     Max.   :160.00
##  total_day_charge total_eve_minutes total_eve_calls total_eve_charge
##  Min.   : 1.12    Min.   : 22.3     Min.   : 38.0   Min.   : 1.90
##  1st Qu.:24.45    1st Qu.:165.8     1st Qu.: 88.0   1st Qu.:14.10
##  Median :30.76    Median :199.9     Median :101.0   Median :17.00
##  Mean   :30.87    Mean   :199.6     Mean   :100.6   Mean   :16.96
##  3rd Qu.:36.70    3rd Qu.:231.8     3rd Qu.:114.0   3rd Qu.:19.70
##  Max.   :59.76    Max.   :359.3     Max.   :169.0   Max.   :30.54
##  total_night_minutes total_night_calls total_night_charge
total_intl_minutes
##  Min.   :  0.0       Min.   :  0.00    Min.   : 0.000     Min.   : 0.00
```

```
##  1st Qu.:166.6       1st Qu.: 86.00    1st Qu.: 7.500     1st Qu.: 8.60
##  Median :199.2       Median : 99.00    Median : 8.960     Median :10.40
##  Mean   :199.2       Mean   : 99.45    Mean   : 8.963     Mean   :10.32
##  3rd Qu.:232.4       3rd Qu.:113.00    3rd Qu.:10.463     3rd Qu.:12.00
##  Max.   :381.6       Max.   :170.00    Max.   :17.170     Max.   :19.70
##  total_intl_calls total_intl_charge number_customer_service_calls
##  Min.   : 0.000   Min.   :0.000     Min.   :0.000
##  1st Qu.: 3.000   1st Qu.:2.320     1st Qu.:1.000
##  Median : 4.000   Median :2.810     Median :1.000
##  Mean   : 4.356   Mean   :2.786     Mean   :1.583
##  3rd Qu.: 5.000   3rd Qu.:3.240     3rd Qu.:2.000
##  Max.   :19.000   Max.   :5.320     Max.   :7.000

#checking for any null values
any(is.na(Customers_To_Predict))

## [1] FALSE

#finding the percentage of null values
colMeans(is.na(Customers_To_Predict))*100

##                         state                account_length
##                             0                             0
##                     area_code            international_plan
##                             0                             0
##               voice_mail_plan         number_vmail_messages
##                             0                             0
##             total_day_minutes               total_day_calls
##                             0                             0
##              total_day_charge             total_eve_minutes
##                             0                             0
##               total_eve_calls              total_eve_charge
##                             0                             0
##           total_night_minutes             total_night_calls
##                             0                             0
##            total_night_charge            total_intl_minutes
##                             0                             0
##              total_intl_calls             total_intl_charge
##                             0                             0
## number_customer_service_calls
##                             0

Customers_To_Predict$international_plan<-
ifelse(Customers_To_Predict$international_plan=="yes",1,0)
Customers_To_Predict$voice_mail_plan<-
ifelse(Customers_To_Predict$voice_mail_plan=="yes",1,0)

#converting the numbers into factors
Customers_To_Predict$international_plan<-
as.factor(Customers_To_Predict$international_plan)
```

```
Customers_To_Predict$voice_mail_plan<-
as.factor(Customers_To_Predict$voice_mail_plan)

#For logistic regression change the below code with
predict(Logistic_Model,newdata = Customers_To_Predict, type = "response")

#running the predictions using the best decision model built.
predict_BestTree_Model<- predict(Dec_tree_model, newdata =
Customers_To_Predict, type = "prob" )
```

Using the threshold identified from the decision tree built on the train data.
Customer_to_predict output data churns are determined from the probabilities.

```
PredictedChurn <- ifelse(predict_BestTree_Model[,2] > dt_threshold, 1, 0)

summary(as.factor(PredictedChurn))

##    0    1
## 1329  271

#Save customerto predict data with predicted churn to the
decisiontreeoutput.csv
write.csv(cbind(Customers_To_Predict,PredictedChurn),file='DecisionTreeOutput
.csv')
```