Computational Statistics Exam

Chiara Rodella(4812486) & Giacomo Accursio(4805025)

June 2020

Data downloaded from IBM Sample Data Sets for customer retention programs.

The goal of this work is to create an analysis in order to predict consumer behaviour.

The dataset is organized as a list of cusotmers whose entries are characteristics (attributes) of the clients.

The variable of interest, on which the analysis is based is **Churn**, which indicates customers that left within the last month. The other available variables identify **services that each customer has signed up for**; for example phone, multiple lines, internet, online security, online backup, device protection, tech support, streaming TV and movies. There are also present **customer account information** (how long they've been a customer, contract, payment method, paperless billing, monthly charges, and total charges) and **demographic info about customers** (gender, age range, and if they have partners and dependents).

The work is subdivided in the following steps:

- Data Preparation
- EDA
- Models
- Conclusions

Data Preparation

Packages used

Packages are the fundamental units of reproducible R code. They include reusable R functions, the documentation that describes how to use them, and the data. Here are some examples of some, but not all, of the packets used:

- library(ggplot2)
- library(corrplot)
- library(ROSE)
- library(rpart)

Data preparation

We proceed by loading the dataset, which is saved in the directory that we have presented.

```
data=read.csv("WA_Fn-UseC_-Telco-Customer-Churn.csv")
```

We use the following command to view the dataset directly in R and also verify that the upload was successful.

```
str(data)
```

```
## 'data.frame':    7043 obs. of  21 variables:
##  $ customerID      : Factor w/ 7043 levels "0002-ORFBO","0003-MKNFE",..: 5376 3963 2565 5536 6512 655
##  $ gender          : Factor w/ 2 levels "Female","Male": 1 2 2 2 1 1 2 1 1 2 ...
##  $ SeniorCitizen   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Partner         : Factor w/ 2 levels "No","Yes": 2 1 1 1 1 1 1 1 2 1 ...
##  $ Dependents      : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 2 1 1 2 ...
##  $ tenure          : int  1 34 2 45 2 8 22 10 28 62 ...
##  $ PhoneService    : Factor w/ 2 levels "No","Yes": 1 2 2 1 2 2 2 1 2 2 ...
##  $ MultipleLines   : Factor w/ 3 levels "No","No phone service",..: 2 1 1 2 1 3 3 2 3 1 ...
##  $ InternetService : Factor w/ 3 levels "DSL","Fiber optic",..: 1 1 1 1 2 2 2 1 2 1 ...
##  $ OnlineSecurity  : Factor w/ 3 levels "No","No internet service",..: 1 3 3 3 1 1 1 3 1 3 ...
##  $ OnlineBackup    : Factor w/ 3 levels "No","No internet service",..: 3 1 3 1 1 1 3 1 1 3 ...
##  $ DeviceProtection: Factor w/ 3 levels "No","No internet service",..: 1 3 1 3 1 3 1 1 3 1 ...
##  $ TechSupport     : Factor w/ 3 levels "No","No internet service",..: 1 1 1 3 1 1 1 1 3 1 ...
##  $ StreamingTV     : Factor w/ 3 levels "No","No internet service",..: 1 1 1 1 1 3 3 1 3 1 ...
##  $ StreamingMovies : Factor w/ 3 levels "No","No internet service",..: 1 1 1 1 1 3 1 1 3 1 ...
##  $ Contract        : Factor w/ 3 levels "Month-to-month",..: 1 2 1 2 1 1 1 1 1 2 ...
##  $ PaperlessBilling: Factor w/ 2 levels "No","Yes": 2 1 2 1 2 2 2 1 2 1 ...
##  $ PaymentMethod   : Factor w/ 4 levels "Bank transfer (automatic)",..: 3 4 4 1 3 3 2 4 3 1 ...
##  $ MonthlyCharges  : num  29.9 57 53.9 42.3 70.7 ...
##  $ TotalCharges    : num  29.9 1889.5 108.2 1840.8 151.7 ...
##  $ Churn           : Factor w/ 2 levels "No","Yes": 1 1 2 1 2 2 1 1 2 1 ...
```

It can be seen that the senior citizen variable is currently red as an *integer*, however it is more appropriate to change it into a *factor* in order to have a dichotomous variable "yes / no".

```
data$SeniorCitizen= as.factor(mapvalues(data$SeniorCitizen,
                                        from=c("0","1"),
                                        to=c("No", "Yes")))
```

R provides a wide range of functions for obtaining summary statistics such as the command below. This is an important first step in order to begin interpreting the data, and then to move on to more complex analysis and insights gathering.

```
summary(data)
```

```
##       customerID      gender     SeniorCitizen Partner    Dependents
##  0002-ORFBO:   1   Female:3488   No :5901      No :3641   No :4933
##  0003-MKNFE:   1   Male  :3555   Yes:1142      Yes:3402   Yes:2110
##  0004-TLHLJ:   1
##  0011-IGKFF:   1
##  0013-EXCHZ:   1
##  0013-MHZWF:   1
##  (Other)   :7037
##      tenure      PhoneService          MultipleLines     InternetService
##  Min.   : 0.00   No : 682     No              :3390   DSL        :2421
##  1st Qu.: 9.00   Yes:6361     No phone service: 682   Fiber optic:3096
##  Median :29.00                Yes             :2971   No         :1526
##  Mean   :32.37
##  3rd Qu.:55.00
##  Max.   :72.00
##
##              OnlineSecurity              OnlineBackup
##  No                 :3498   No                 :3088
##  No internet service:1526   No internet service:1526
##  Yes                :2019   Yes                :2429
```

```
##
##
##
##
##             DeviceProtection              TechSupport
##  No                  :3095    No                  :3473
##  No internet service:1526    No internet service:1526
##  Yes                 :2422    Yes                 :2044
##
##
##
##
##               StreamingTV              StreamingMovies           Contract
##  No                  :2810   No                  :2785   Month-to-month:3875
##  No internet service:1526   No internet service:1526   One year      :1473
##  Yes                 :2707   Yes                 :2732   Two year      :1695
##
##
##
##
##  PaperlessBilling                   PaymentMethod  MonthlyCharges
##  No :2872         Bank transfer (automatic):1544   Min.   : 18.25
##  Yes:4171         Credit card (automatic)  :1522   1st Qu.: 35.50
##                   Electronic check         :2365   Median : 70.35
##                   Mailed check             :1612   Mean   : 64.76
##                                                    3rd Qu.: 89.85
##                                                    Max.   :118.75
##
##   TotalCharges    Churn
##  Min.   :  18.8   No :5174
##  1st Qu.: 401.4   Yes:1869
##  Median :1397.5
##  Mean   :2283.3
##  3rd Qu.:3794.7
##  Max.   :8684.8
##  NA's   :11
```

Let's now proceed with the handling of the missing values. Missing data are a common occurrence and can have a significant effect on the conclusions that can be drawn from the data.

```
sapply(data, function(x) sum(is.na(x)))
```

```
##       customerID           gender    SeniorCitizen          Partner
##                0                0                0                0
##       Dependents           tenure     PhoneService    MultipleLines
##                0                0                0                0
##  InternetService   OnlineSecurity     OnlineBackup DeviceProtection
##                0                0                0                0
##      TechSupport      StreamingTV  StreamingMovies         Contract
##                0                0                0                0
## PaperlessBilling    PaymentMethod   MonthlyCharges     TotalCharges
##                0                0                0               11
##            Churn
##                0
```

We see that in "Total Charges" there are missing values. We then extract the subset, given their relatively low

number (only 11 out of 7043 observations), to go into detail; we have to understand whether to hypothesize imputation techniques, for example, or eliminate them.

There are only 11 rows out of 7043 total and all with target variable "Churn = No". As we will see later "Churn = No" is overrappresented in our population so we can remove these rows.

```
data=na.omit(data)
cat("New number of observations:",dim(data)[1])
```

```
## New number of observations: 7032
```

Since we are still in the data preparation phase, we prepare the tenure variable directly in a way that will be useful in the subsequent analysis.

```
tenure_initial=data[,"tenure"]
TotalCharges_initial=data[,"TotalCharges"]
```

We see how some variables present a redundancy in "No" and "No internet services"; we make the assumption that "No internet services" can be represented in the "No" category.

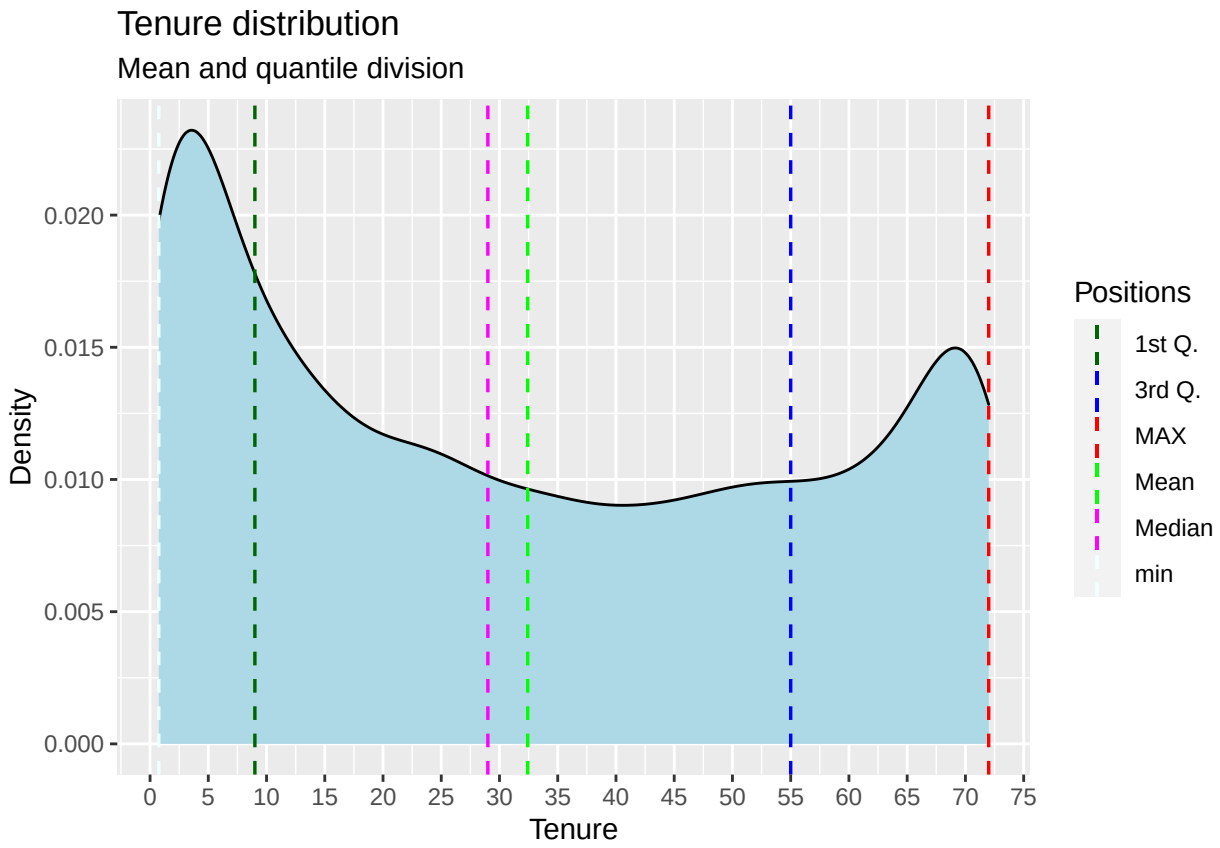
```
cols1 = c(10:15)

for(i in 1:ncol(data[,cols1])) {
  data[,cols1][,i] <- as.factor(mapvalues
                                (data[,cols1][,i], from =c("No internet service"),to=c("No")))
}

data$MultipleLines = as.factor(mapvalues(data$MultipleLines,
                                         from = c("No phone service"),to = c("No")))
```

We want to focus on the tenure variable and see how to treat it, to make it more useful in our analysis. We start from a graphical investigation.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    9.00   29.00   32.42   55.00   72.00
```



We have min 1 month and max 72 month. We thought it would have been more practical and easier to interpret reasoning on horizons divided year by year instead of month by month. We therefore decided to divide "Tenure" into five annual groups as shown in the following code:

```
data$tenure_groups = 0

data$tenure_groups[data$tenure>=1 & data$tenure<=12]="1y"
data$tenure_groups[data$tenure>12 & data$tenure<=24]="2y"
data$tenure_groups[data$tenure>24 & data$tenure<=36]="3y"
data$tenure_groups[data$tenure>36 & data$tenure<=48]="4y"
data$tenure_groups[data$tenure>48 ]="5y+"

summary.factor(data$tenure_groups)
```

```
##   1y   2y   3y   4y  5y+
## 2175 1024  832  762 2239
```

We therfore transform the variable tenure_groups into factor with 5 levels as created above:

```
data$tenure_groups = as.factor(data$tenure_groups)

str(data$tenure_groups) #to check the new data type
```

```
##  Factor w/ 5 levels "1y","2y","3y",..: 1 3 1 4 1 1 2 1 3 5 ...
```

We conclude this preparatory phase by removing those columns that we will not need for our further analysis. We are therefore going to create a pre-processed dataset with all the appropriate changes made for the analysis we are about to do. This is called: **D_Churn**.

```
data$customerID = NULL
data$tenure = NULL
D_Churn=data
```

EDA

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns,to spot anomalies,to test hypothesis and to check assumptions with the help of summary statistics and graphical representations. It is a good practice to understand the data first and try to find insights from it.

Variables Analysis

We begin the exploratory analysis of the variables and observations that we have available starting from the quantitatives variables, then move also on to the categorical ones.

First of all, given all the variables, let's identify the quantitative ones using the following function:

```
Quant.var=sapply(D_Churn, is.numeric)
Quant.var
```

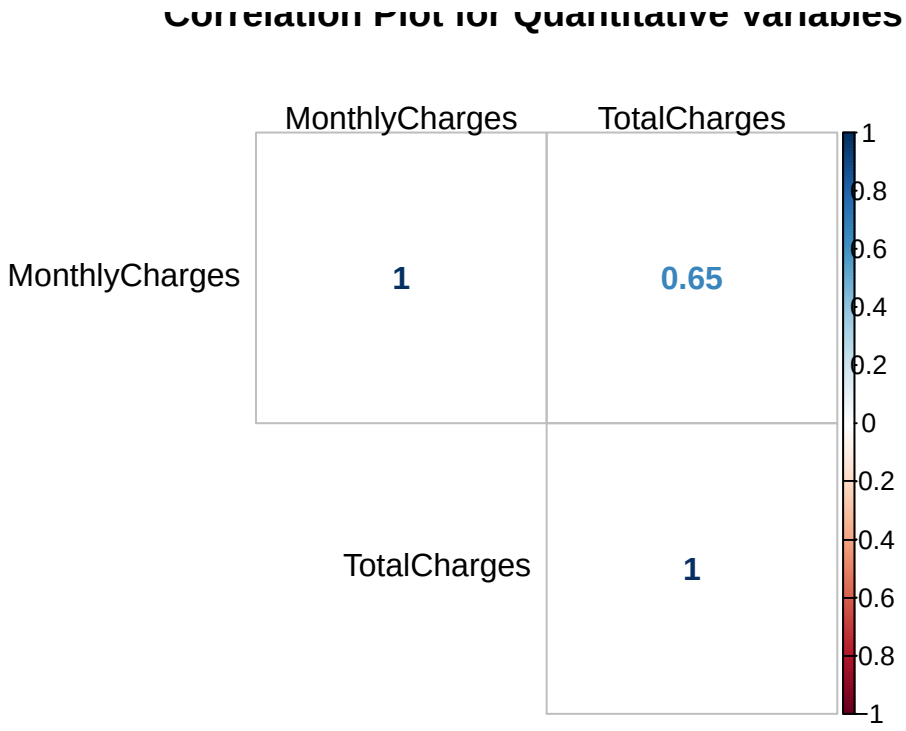
```
##           gender    SeniorCitizen          Partner       Dependents
##            FALSE            FALSE            FALSE            FALSE
##     PhoneService    MultipleLines  InternetService   OnlineSecurity
##            FALSE            FALSE            FALSE            FALSE
##     OnlineBackup DeviceProtection      TechSupport      StreamingTV
##            FALSE            FALSE            FALSE            FALSE
##  StreamingMovies         Contract PaperlessBilling    PaymentMethod
##            FALSE            FALSE            FALSE            FALSE
##   MonthlyCharges     TotalCharges            Churn    tenure_groups
##             TRUE             TRUE            FALSE            FALSE
```

```
summary.factor(Quant.var)
```

```
## FALSE  TRUE
##    18     2
```

We therefore have only two quantitative variables which are: *MonthlyCharges* and *TotalCharges*. We saved them under the name: *Quant.var*.

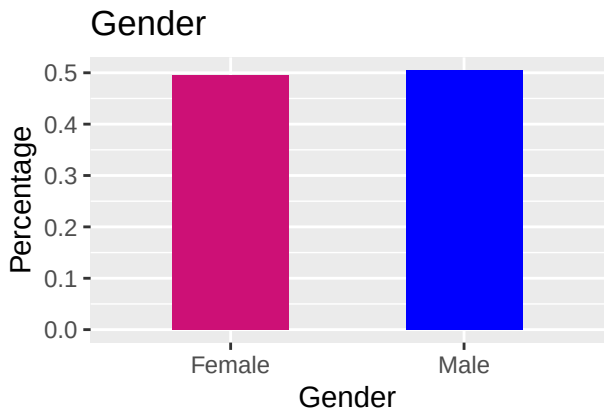
We begin the study starting from the correlation matrix. A **correlation matrix** is a table showing correlation coefficients between sets of variables. Each random variable (X_i) in the table is correlated with each of the other values in the table (X_j). This allows us to check which pairs of variables have the highest correlation. In this case we have our Quant.var.

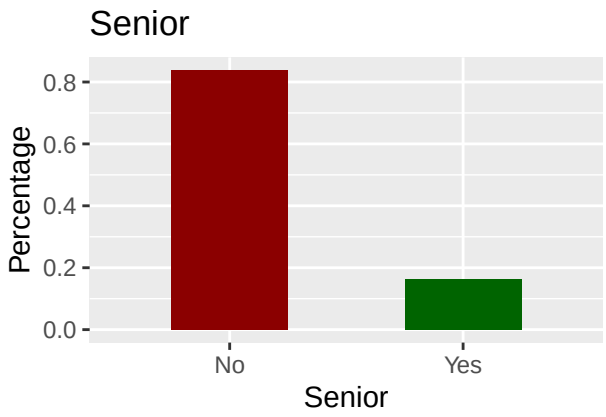


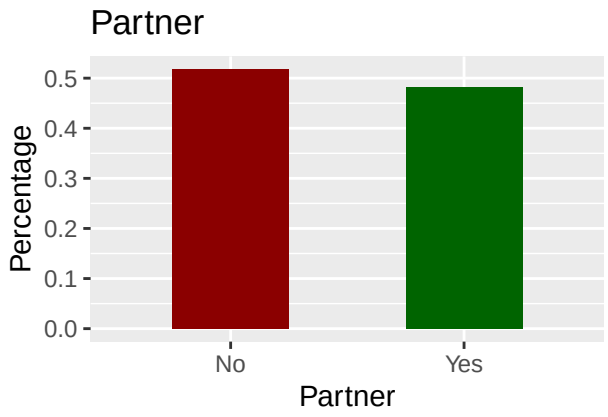
Monthly and Total Charges are highly correlated as it was easy and intuitive to expect. We propose to remove "Total Charges" variable since it is a consequence of the other variable. It has to be specified that this is an assumption, a logical choiche, that we decided to make, it is not functional for us to keep both varibles. However, this does not mean that keeping this variable could not be useful for another type of analysis.

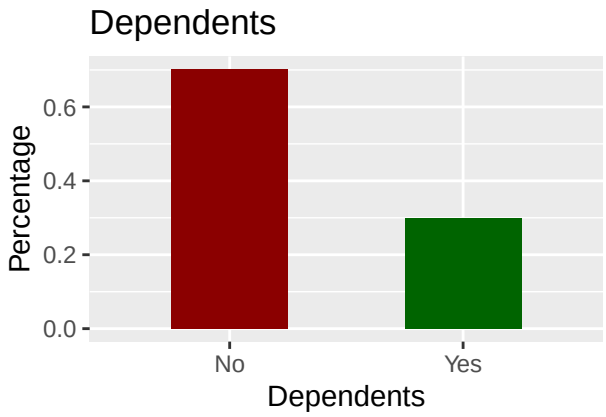
```
D_Churn$TotalCharges = NULL
```

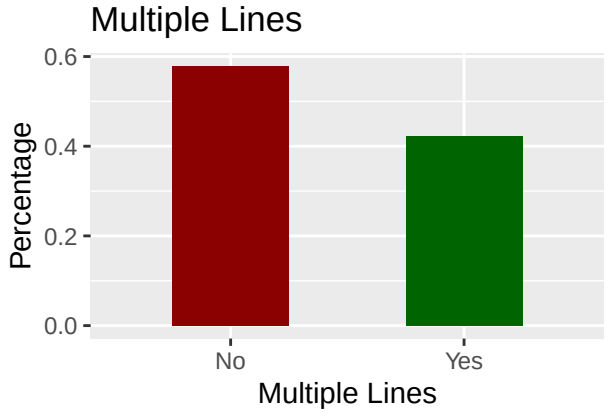
Now let's see the distribution plots for the individual variables:

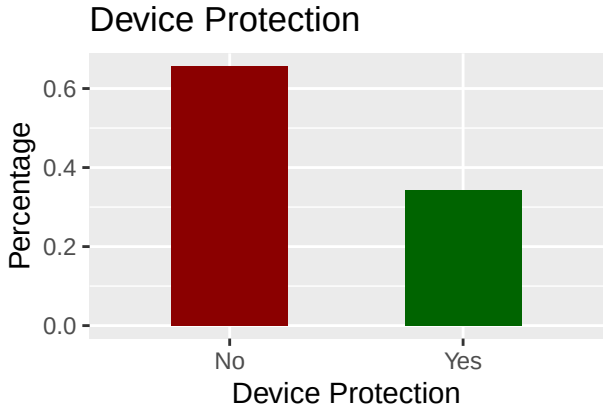

Gender
Percentage
0.5
0.4
0.3
0.2
0.1
0.0
Female
Male
Gender

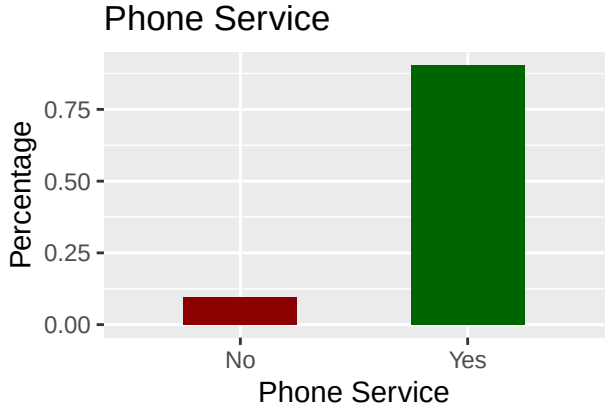

Senior
Percentage
0.8
0.6
0.4
0.2
0.0
No
Yes
Senior

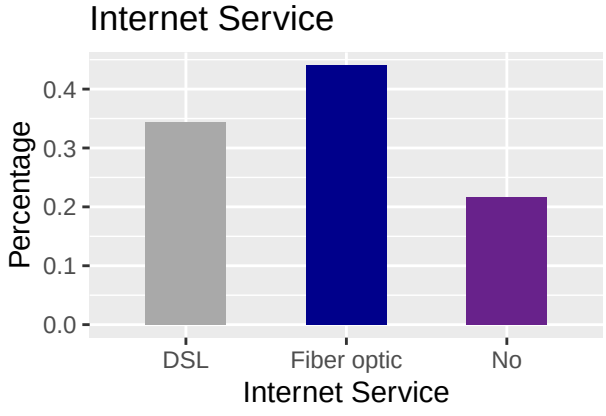

Partner
Percentage
0.5
0.4
0.3
0.2
0.1
0.0
No
Yes
Partner


Dependents
Percentage
0.6
0.4
0.2
0.0
No
Yes
Dependents

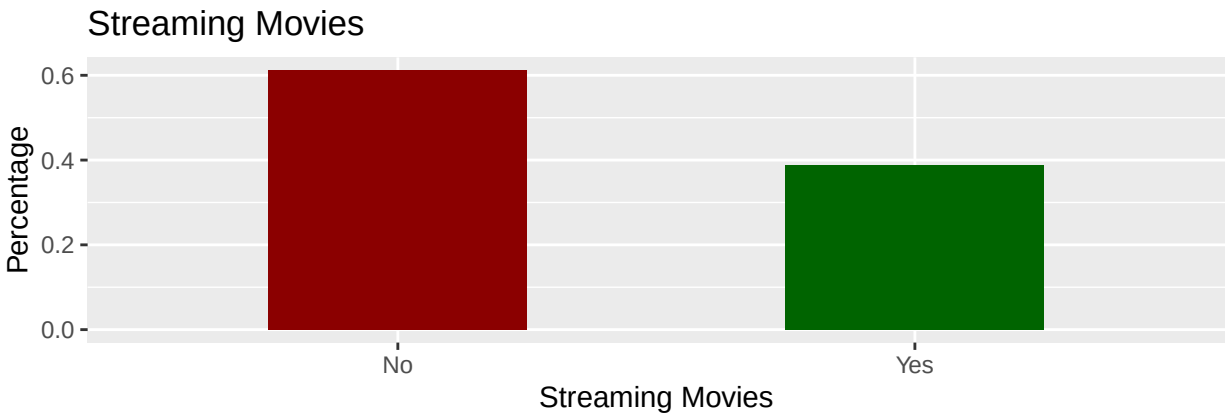

Multiple Lines
Percentage
0.6
0.4
0.2
0.0
No
Yes
Multiple Lines

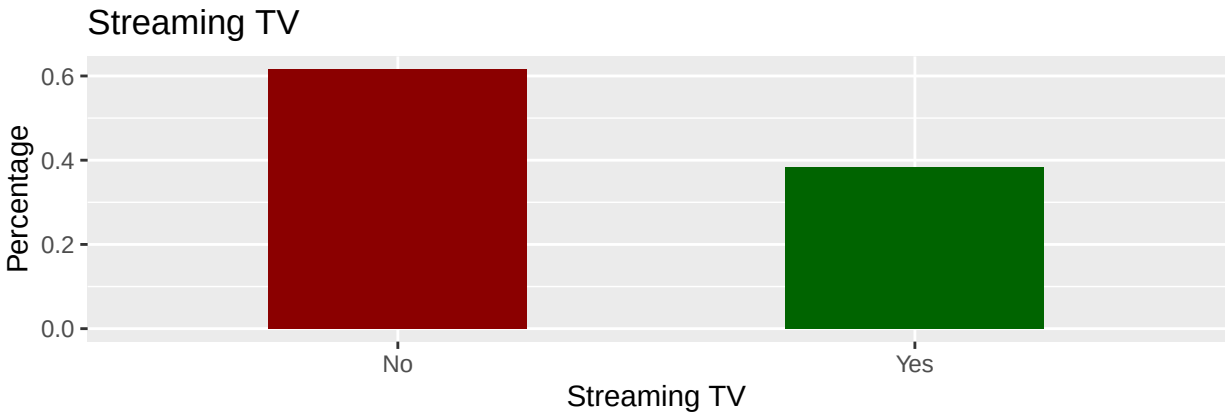

Device Protection
Percentage
0.6
0.4
0.2
0.0
No
Yes
Device Protection

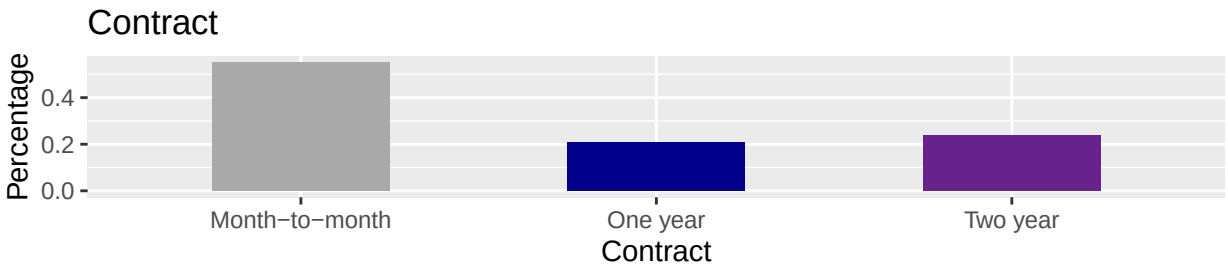

Phone Service
Percentage
0.75
0.50
0.25
0.00
No
Yes
Phone Service

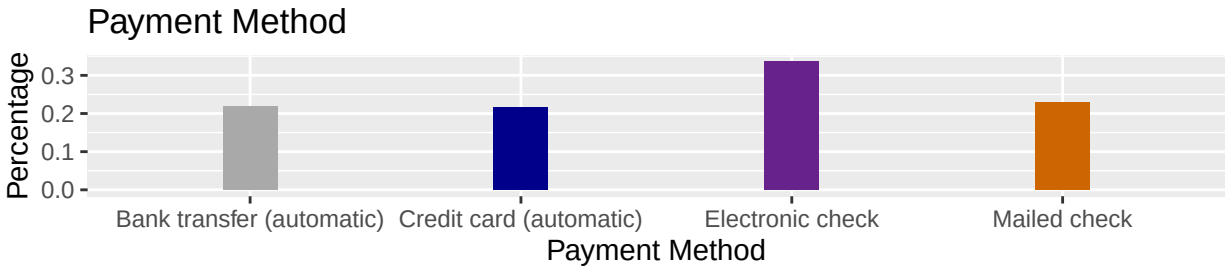

Internet Service
Percentage
0.4
0.3
0.2
0.1
0.0
DSL
Fiber optic
No
Internet Service

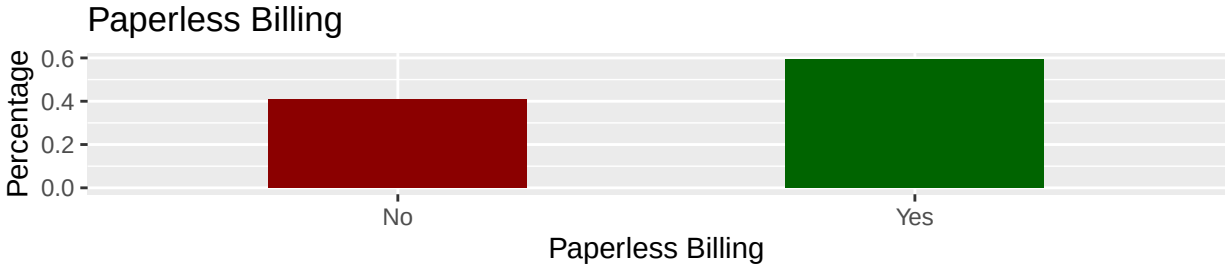

Multiple Lines
Percentage
0.6
0.4
0.2
0.0
No
Yes
Multiple Lines
Device Protection
Percentage
0.6
0.4
0.2
0.0
No
Yes
Device Protection
Phone Service
Percentage
0.75
0.50
0.25
0.00
No
Yes
Phone Service
Internet Service
Percentage
0.4
0.3
0.2
0.1
0.0
DSL
Fiber optic
No
Internet Service
Online Backup
Percentage
0.6
0.4
0.2
0.0
No
Yes
Online Backup
Online Security
Percentage
0.6
0.4
0.2
0.0
No
Yes
Online Security
Tech Support
Percentage
0.6
0.4
0.2
0.0
No
Yes
Tech Support
Streaming TV
Percentage
0.6
0.4
0.2
0.0
No
Yes
Streaming TV

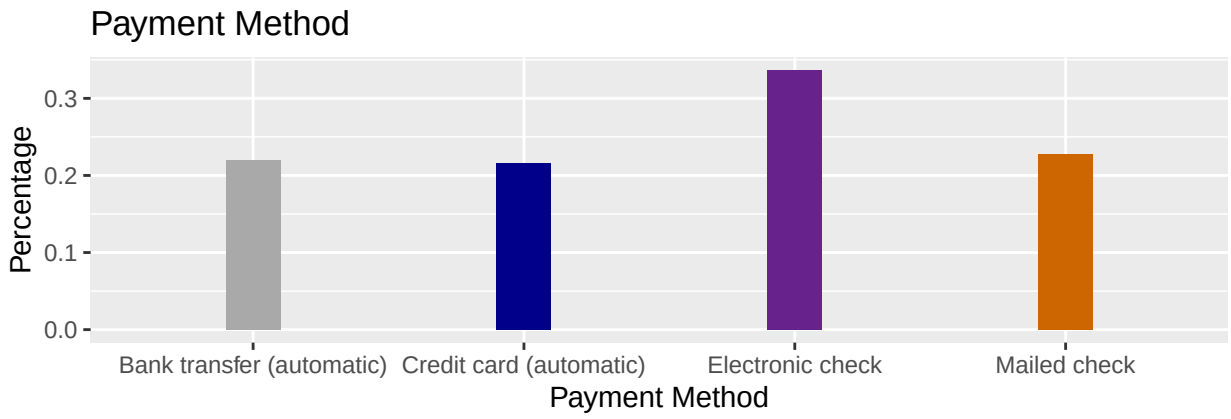

Streaming Movies
Percentage
0.6
0.4
0.2
0.0
No
Yes
Streaming Movies

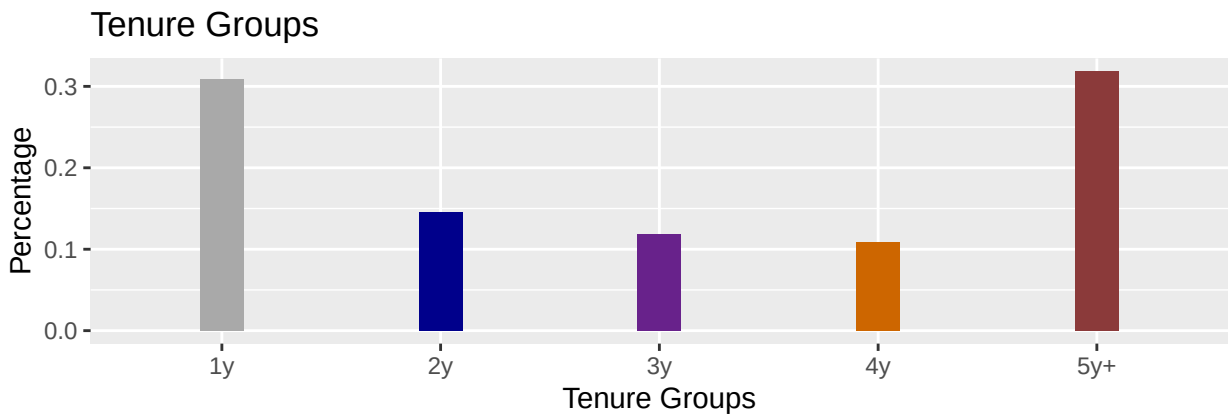

Streaming TV
Percentage
0.6
0.4
0.2
0.0
No
Yes
Streaming TV

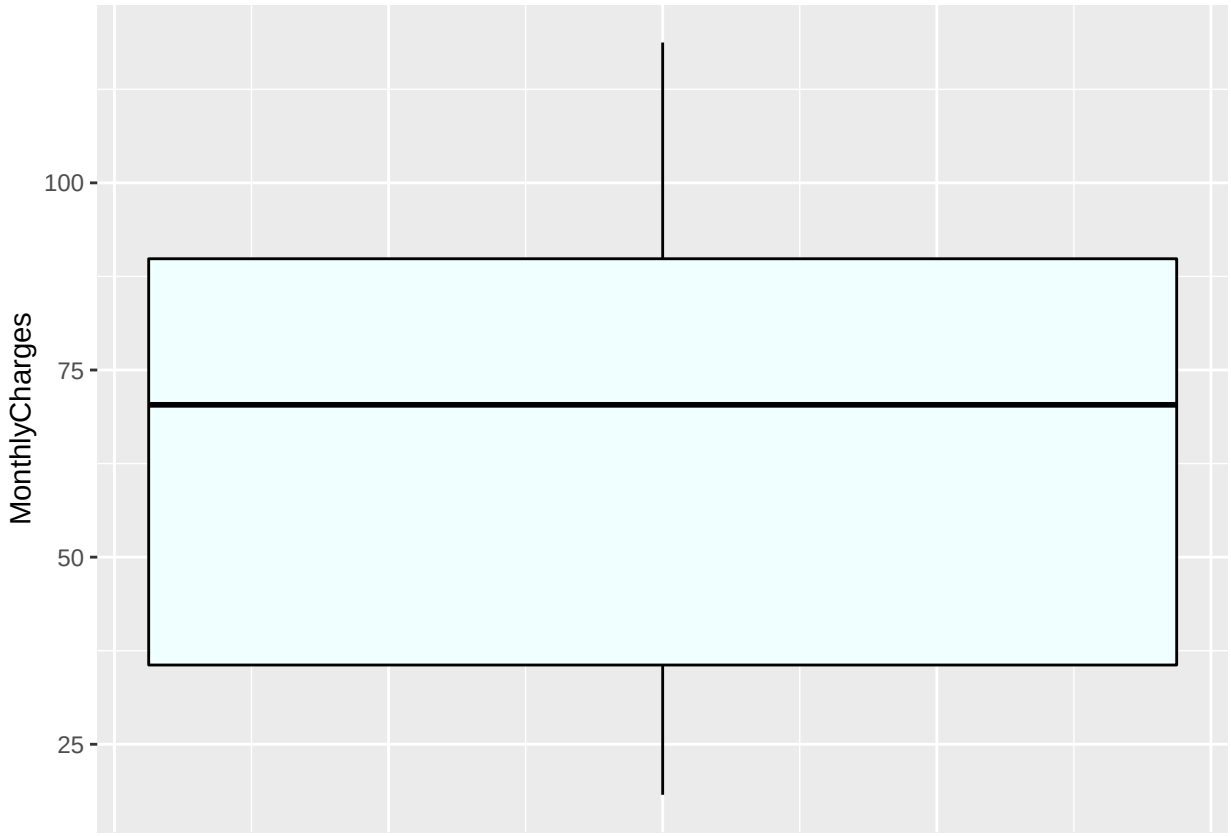

Contract
Percentage
0.4
0.2
0.0
Month−to−month
One year
Two year
Contract


Payment Method
Percentage
0.3
0.2
0.1
0.0
Bank transfer (automatic)
Credit card (automatic)
Electronic check
Mailed check
Payment Method


Paperless Billing
Percentage
0.6
0.4
0.2
0.0
No
Yes
Paperless Billing

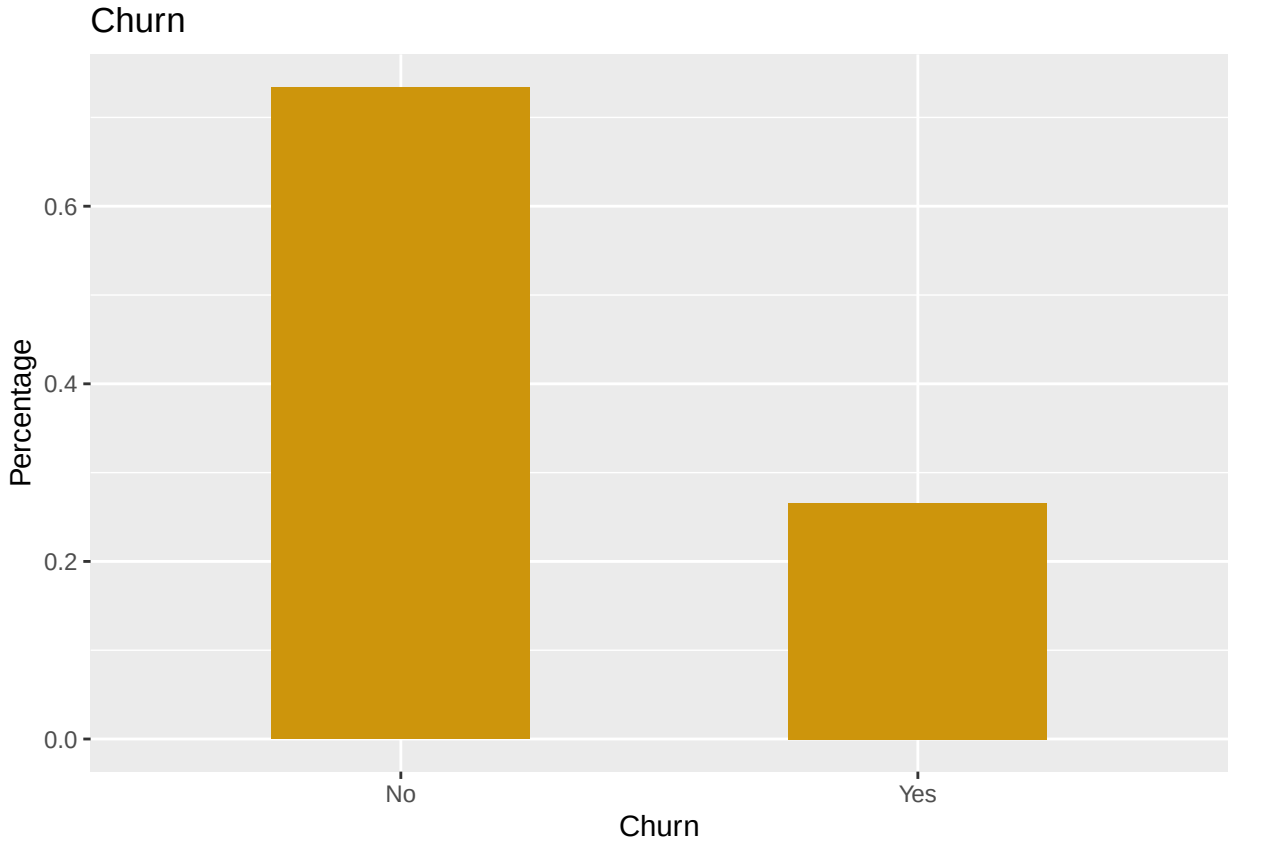

Payment Method
Percentage
0.3
0.2
0.1
0.0
Bank transfer (automatic)
Credit card (automatic)
Electronic check
Mailed check
Payment Method


Tenure Groups
Percentage
0.3
0.2
0.1
0.0
1y
2y
3y
4y
5y+
Tenure Groups


MonthlyCharges
100
75
50
25

All the qualitative variables show reasonably distribution spread, we have good evidence to keep all of them, more over, our only numerical variable does not seem to show any outliers.

After studying our variables graphically, we focus on our target variable:

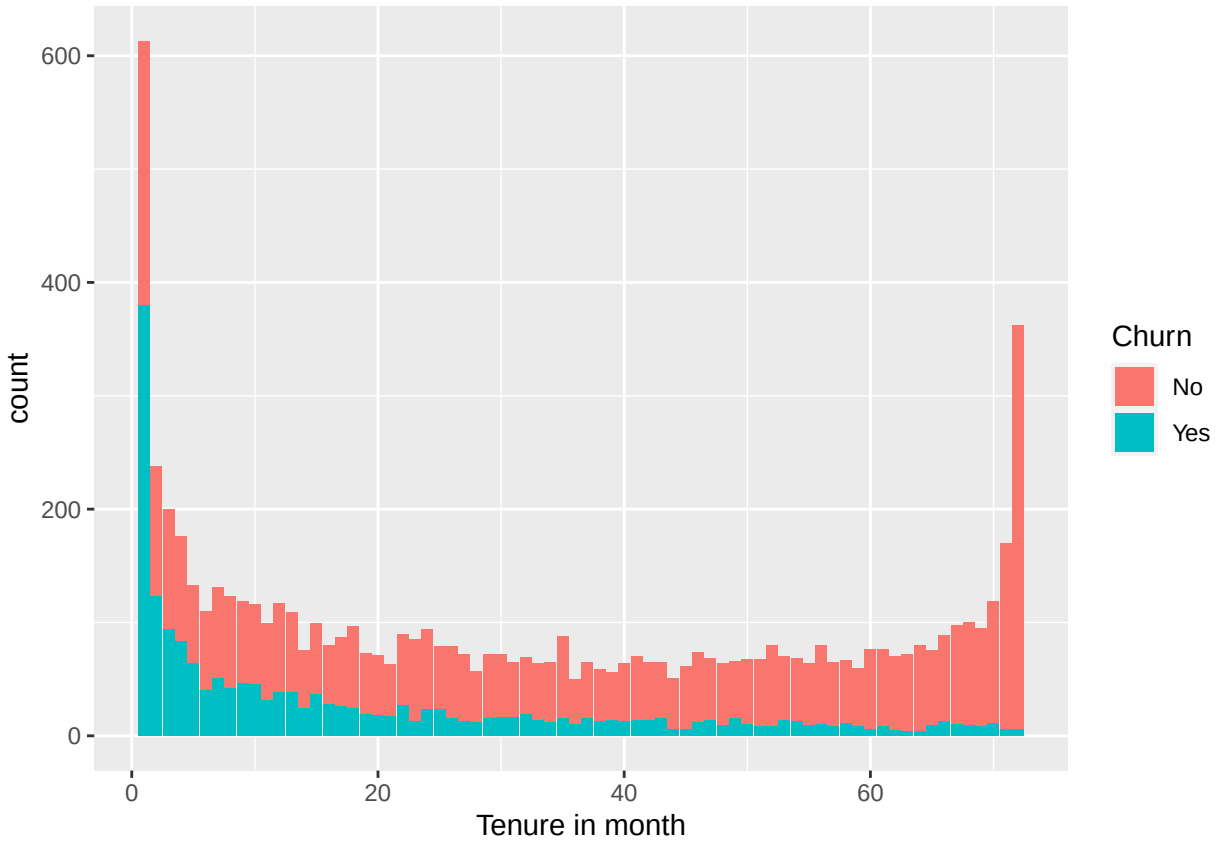



```
table(data$Churn)/nrow(data)
```

```
##
##       No      Yes
## 0.734215 0.265785
```

As it is easily visible, the distribution is heavily unbalanced towards the "No". This is a problem that will have to be addressed before we move forward with the implementation of the models.

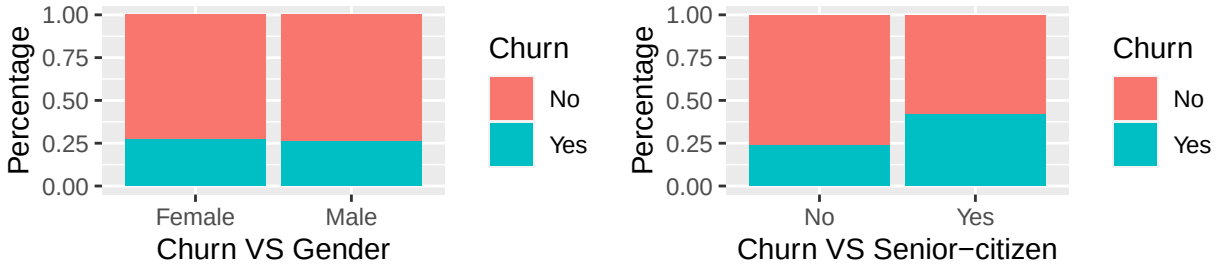
We also want to see how the variable "tenure" behaves over the months, compared to "Churn". To do this we use the previous version of the variable "Tenure" that we created during the data preparation.



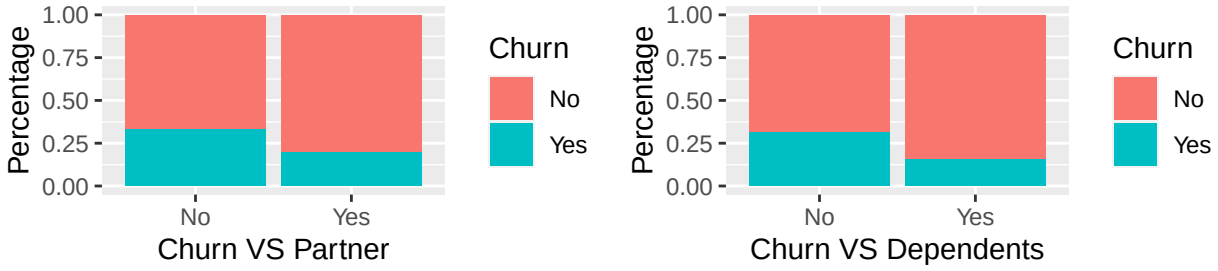
The distribution for tenure is very different between customers who churned and who didn't churn. For those who churned, the distribution is positvely skewed, which means customers who churned are more likely to cancel the service in less than 1 year.

For current customers who didn't churn, there are two peaks. The second peak is much more ripid than the first one, which means that a large group of current customers have been using the service almost 6 years (almost 70 month).

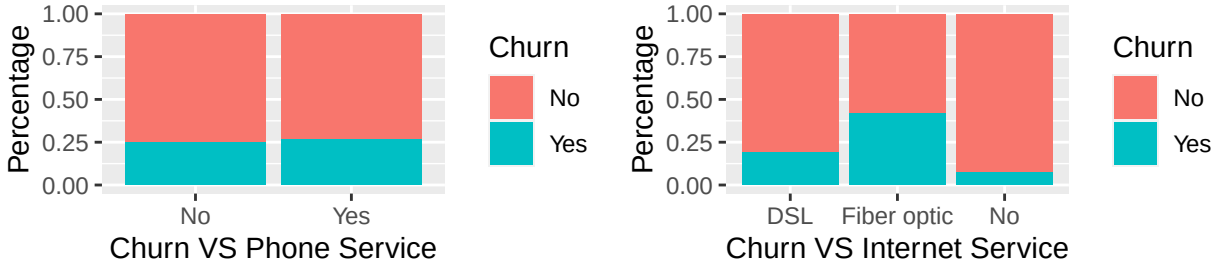
Now we will show how the variables behave with respect to the response variable (Churn).


Percentage
1.00
0.75
0.50
0.25
0.00
Female
Male
Churn VS Gender
Churn
No
Yes

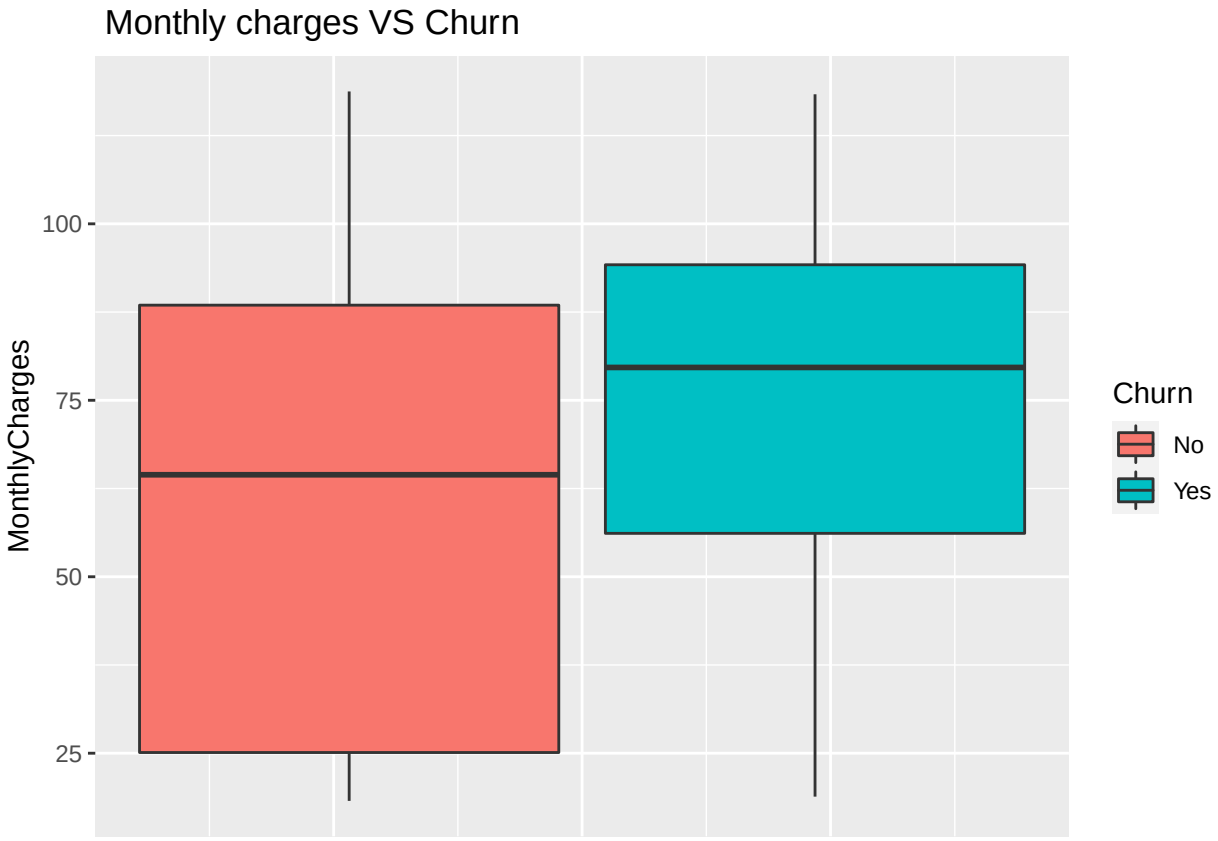
Percentage
1.00
0.75
0.50
0.25
0.00
No
Yes
Churn VS Senior−citizen
Churn
No
Yes


Percentage
1.00
0.75
0.50
0.25
0.00
No
Yes
Churn VS Partner
Churn
No
Yes

Percentage
1.00
0.75
0.50
0.25
0.00
No
Yes
Churn VS Dependents
Churn
No
Yes


Percentage
1.00
0.75
0.50
0.25
0.00
No
Yes
Churn VS Phone Service
Churn
No
Yes

Percentage
1.00
0.75
0.50
0.25
0.00
DSL
Fiber optic
No
Churn VS Internet Service
Churn
No
Yes


Monthly charges VS Churn
MonthlyCharges
100
75
50
25
Churn
No
Yes

The Churn rate does not change particularly between males and females, as well as for those who have a telephone service and those who do not have it; in both variables there is a lower dropout rate. As for senior citizens, it is noted that have a higher dropout rate (Churn=Yes) than non-senior citizens.

As for customers who have Partners, in the same way as customers who have employees, they have a higher negative churn than those without partners, it means that they abandon less.

Finally, among the three methods of internet service, those who have not used the service as those who have DSL, have a lower abandonment than customers with fiber optics.

MODELS

Before starting with the implementation of the models, it is necessary to work on the available data We have to divide our population into train and test data:

- training set: a subset to train a model;
- test set: a subset to test the trained model.

```
set.seed(2020)
intrain= createDataPartition(D_Churn$Churn,p=0.8,list=FALSE)
C_train= D_Churn[intrain,]
C_test= D_Churn[-intrain,]

cat("Dimensions of training set:",dim(C_train))
```

```
## Dimensions of training set: 5627 19
```

```
#To confirm the right dimensions of the partition
```

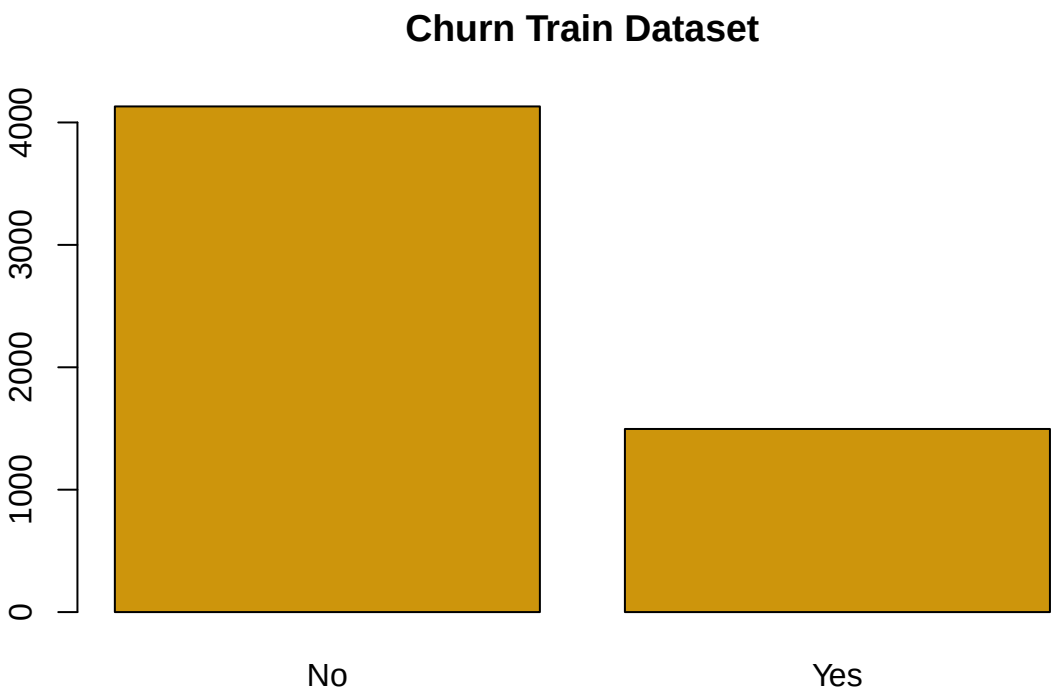
```
cat("Dimensions of test dataset:",dim(C_test))
```

```
## Dimensions of test dataset: 1405 19
```

We can see how my observations for the two Churn modalities are distributed within the training set. We see from the number and from the plot that this is an absolutely unbalanced situation.

```
table(C_train$Churn)
```

```
##
##   No  Yes
## 4131 1496
```



After transforming my categorical variables into **factor**, a step that I need later to work on the models, we can summarize the structure of my sets to also understand how to proceed.

```
str(C_train)
```

```
## 'data.frame':    5627 obs. of  19 variables:
##  $ gender          : Factor w/ 2 levels "Female","Male": 2 2 1 1 1 1 2 2 2 2 ...
##  $ SeniorCitizen   : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Partner         : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 2 1 1 2 1 ...
##  $ Dependents      : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 2 1 1 1 ...
##  $ PhoneService    : Factor w/ 2 levels "No","Yes": 2 2 2 2 1 2 2 2 2 2 ...
##  $ MultipleLines   : Factor w/ 2 levels "No","Yes": 1 1 1 2 1 2 1 1 2 1 ...
##  $ InternetService : Factor w/ 3 levels "DSL","Fiber optic",..: 1 1 2 2 1 2 1 3 2 2 ...
##  $ OnlineSecurity  : Factor w/ 2 levels "No","Yes": 2 2 1 1 2 1 2 1 1 2 ...
##  $ OnlineBackup    : Factor w/ 2 levels "No","Yes": 1 2 1 1 1 1 2 1 1 1 ...
##  $ DeviceProtection: Factor w/ 2 levels "No","Yes": 2 1 1 2 1 2 1 1 2 2 ...
##  $ TechSupport     : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 2 1 1 1 2 ...
##  $ StreamingTV     : Factor w/ 2 levels "No","Yes": 1 1 1 2 1 2 1 1 2 2 ...
##  $ StreamingMovies : Factor w/ 2 levels "No","Yes": 1 1 1 2 1 2 1 1 2 2 ...
##  $ Contract        : Factor w/ 3 levels "Month-to-month",..: 2 1 1 1 1 1 2 3 2 1 ...
##  $ PaperlessBilling: Factor w/ 2 levels "No","Yes": 1 2 2 2 1 2 1 1 1 2 ...
##  $ PaymentMethod   : Factor w/ 4 levels "Bank transfer (automatic)",..: 4 4 3 3 4 3 1 2 2 3 ...
##  $ MonthlyCharges  : num  57 53.9 70.7 99.7 29.8 ...
##  $ Churn           : Factor w/ 2 levels "No","Yes": 1 2 2 2 1 2 1 1 1 1 ...
##  $ tenure_groups   : Factor w/ 5 levels "1y","2y","3y",..: 3 1 1 1 1 3 5 2 5 3 ...
##  - attr(*, "na.action")= 'omit' Named int  489 754 937 1083 1341 3332 3827 4381 5219 6671 ...
##   ..- attr(*, "names")= chr  "489" "754" "937" "1083" ...
```

```
str(C_test)
```

```
## 'data.frame':    1405 obs. of  19 variables:
##  $ gender          : Factor w/ 2 levels "Female","Male": 1 2 2 2 2 1 2 2 1 2 ...
##  $ SeniorCitizen   : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 2 1 2 ...
##  $ Partner         : Factor w/ 2 levels "No","Yes": 2 1 1 2 1 2 2 1 2 1 ...
```

```
##  $ Dependents      : Factor w/ 2 levels "No","Yes": 1 1 2 2 1 2 2 1 2 1 ...
##  $ PhoneService    : Factor w/ 2 levels "No","Yes": 1 1 2 2 2 2 2 2 2 2 ...
##  $ MultipleLines   : Factor w/ 2 levels "No","Yes": 1 1 2 1 2 2 1 1 2 2 ...
##  $ InternetService : Factor w/ 3 levels "DSL","Fiber optic",..: 1 1 2 1 2 2 1 1 1 2 ...
##  $ OnlineSecurity  : Factor w/ 2 levels "No","Yes": 1 2 1 2 1 2 2 1 2 1 ...
##  $ OnlineBackup    : Factor w/ 2 levels "No","Yes": 2 1 2 1 2 2 2 1 2 1 ...
##  $ DeviceProtection: Factor w/ 2 levels "No","Yes": 1 2 1 1 2 2 1 1 1 1 ...
##  $ TechSupport     : Factor w/ 2 levels "No","Yes": 1 2 1 1 1 2 2 1 1 1 ...
##  $ StreamingTV     : Factor w/ 2 levels "No","Yes": 1 1 2 1 2 2 1 1 2 2 ...
##  $ StreamingMovies : Factor w/ 2 levels "No","Yes": 1 1 1 1 2 2 1 1 1 2 ...
##  $ Contract        : Factor w/ 3 levels "Month-to-month",..: 1 2 1 1 1 3 1 1 3 1 ...
##  $ PaperlessBilling: Factor w/ 2 levels "No","Yes": 2 1 2 2 2 1 1 1 2 2 ...
##  $ PaymentMethod   : Factor w/ 4 levels "Bank transfer (automatic)",..: 3 1 2 4 1 2 2 1 2 3 ...
##  $ MonthlyCharges  : num  29.9 42.3 89.1 50 103.7 ...
##  $ Churn           : Factor w/ 2 levels "No","Yes": 1 1 1 1 2 1 1 1 1 2 ...
##  $ tenure_groups   : Factor w/ 5 levels "1y","2y","3y",..: 1 4 2 2 5 5 5 1 5 2 ...
##  - attr(*, "na.action")= 'omit' Named int  489 754 937 1083 1341 3332 3827 4381 5219 6671 ...
##   ..- attr(*, "names")= chr  "489" "754" "937" "1083" ...
```

In classification problems, a disparity in the frequencies of the observed classes can have a significant negative impact on model fitting. One technique for resolving such a class imbalance is to subsample the training data in a manner that mitigates the issues. We used an hybrid method that is called *ROSE*, to deal with binary classification problems in the presence of imbalanced classes.

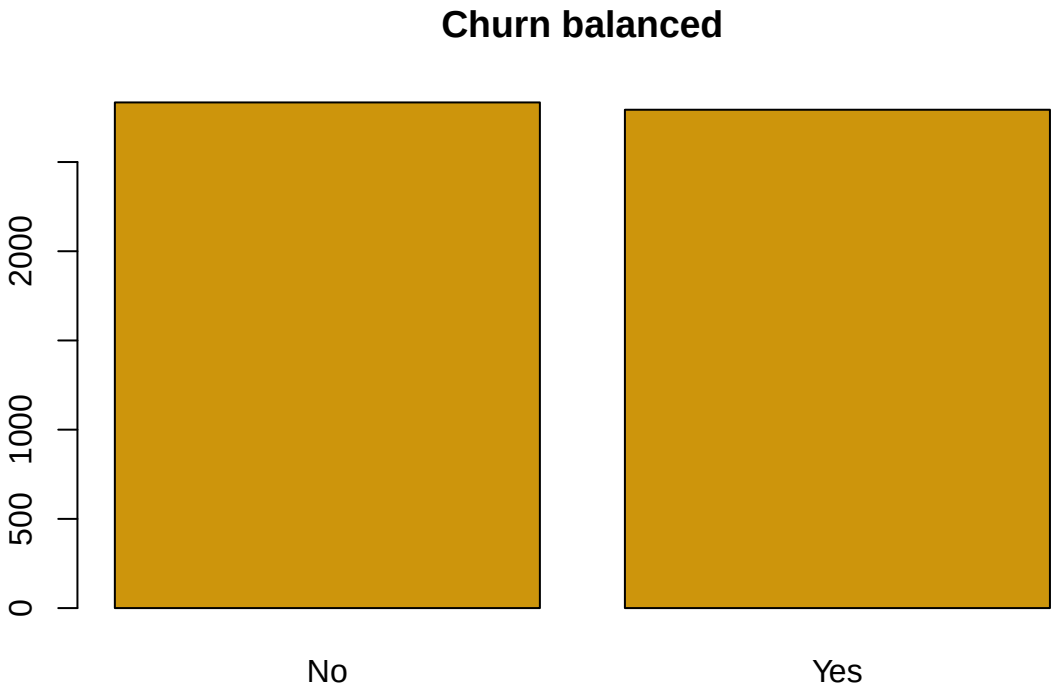
ROSE (Random Over Sampling Examples) helps us to generate artificial data based on sampling methods with a smoothed bootstrap approach; it uses smoothed bootstrapping to draw artificial samples from the feature space neighbourhood around the minority class.

```
# generate new balanced data
set.seed(2020)
model.train=ROSE(Churn ~ ., data=C_train, seed=123)$data
model.test=C_test

model.complete=rbind(model.train,model.test)
```

```
# check balance of new data
table(model.train$Churn)
```

```
##
##   No  Yes
## 2834 2793
```



It is now clearly visible, both from the point of view of the number of observations and from the plot that we are in a much more balanced situation.

Model 1: Logistic Regression

This is a classification model under the Generalized Linear Models family; the main feature is that the target variable is distributed like a bernoulli. It is very interpretable because thanks to the coefficient estimation we can understand which are the most influential features and how they change the result (increase/decrease the log ods). We use Logistic regression to predict the class **Churn** of individuals based on multiple predictor variables.

```
LogModel=glm(Churn ~ .,family=binomial,data=model.train)

print(summary(LogModel))
```

```
##
## Call:
## glm(formula = Churn ~ ., family = binomial, data = model.train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4562  -0.7499  -0.1801   0.7814   2.8854
##
## Coefficients:
##                                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)                      1.0063327  0.2076516   4.846 1.26e-06 ***
## genderMale                      -0.0421902  0.0670889  -0.629 0.529434
## SeniorCitizenYes                 0.1424754  0.0888017   1.604 0.108621
## PartnerYes                       0.0006945  0.0828882   0.008 0.993315
## DependentsYes                    0.0089433  0.0918292   0.097 0.922416
## PhoneServiceYes                 -0.7029836  0.1697243  -4.142 3.44e-05 ***
## MultipleLinesYes                 0.3126754  0.0875001   3.573 0.000352 ***
```

```
## InternetServiceFiber optic            1.0061104  0.1653532   6.085 1.17e-09 ***
## InternetServiceNo                     -0.5571597  0.1883534  -2.958 0.003096 **
## OnlineSecurityYes                     -0.4277511  0.0885581  -4.830 1.36e-06 ***
## OnlineBackupYes                       -0.2243945  0.0835275  -2.686 0.007221 **
## DeviceProtectionYes                    0.0564890  0.0879259   0.642 0.520573
## TechSupportYes                        -0.1676454  0.0922331  -1.818 0.069121 .
## StreamingTVYes                         0.2598856  0.1018471   2.552 0.010719 *
## StreamingMoviesYes                     0.4935345  0.1027809   4.802 1.57e-06 ***
## ContractOne year                      -0.7896781  0.1047095  -7.542 4.64e-14 ***
## ContractTwo year                      -1.8343058  0.1611352 -11.384  < 2e-16 ***
## PaperlessBillingYes                    0.4030219  0.0758156   5.316 1.06e-07 ***
## PaymentMethodCredit card (automatic) -0.1866747  0.1123690  -1.661 0.096660 .
## PaymentMethodElectronic check          0.2224409  0.0978928   2.272 0.023069 *
## PaymentMethodMailed check             -0.2606900  0.1149245  -2.268 0.023307 *
## MonthlyCharges                        -0.0014413  0.0053437  -0.270 0.787373
## tenure_groups2y                       -1.0022222  0.1041085  -9.627  < 2e-16 ***
## tenure_groups3y                       -1.5985611  0.1228023 -13.017  < 2e-16 ***
## tenure_groups4y                       -1.5687281  0.1328180 -11.811  < 2e-16 ***
## tenure_groups5y+                      -1.7157791  0.1383424 -12.402  < 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: 7800.4  on 5626  degrees of freedom
## Residual deviance: 5434.9  on 5601  degrees of freedom
## AIC: 5486.9
##
## Number of Fisher Scoring iterations: 5
```

The summary(LogModel) gives the beta coefficients, standard error, z Value and p Value. Due to the fact that we have categorical variables with multiple levels, we find a row-entry for each category of that variable (like Contract). That is because, each individual category is considered as an independent binary variable by the glm(). In case of only two level we will have only one row (like StreamingTv)

The difference between Null deviance and Residual deviance tells us that the model is a good fit. Greater the difference better the model. Null deviance is the value when you only have intercept in your equation with no variables and Residual deviance is the value when you are taking all the variables into account. It makes sense to consider the model good if that difference is big enough.

The AIC value is a measure of the quality of the model it takes into consideration the number of variables used compared to the number of observations. A low AIC is desirable.

Now let's start with the *prediction*, using the train set as the first step.

```
pred.log.train = predict(LogModel, newdata=model.train,
                  type="response")
summary(pred.log.train)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.01268 0.20809 0.52263 0.49636 0.77633 0.95547
```

The output of a Logistic regression model is a probability. We can select a threshold value of the probability. If the probability is greater than this threshold value, the event(Churn) is predicted to happen otherwise it is predicted not to happen.

So, instead of trying to predict exactly whether the people will churn or not, we calculate the probability or a likelihood of the customer to be yes. We are tring to fit a probability into 0 and 1, which are the two possible

outcomes. To do that we decide a cut off value/threshold. The threshold value is selected based on which errors are better. This would imply that it would be best for no errors (quite impossible).

There are two types of errors that this model can make:

1. where the model predicts YES, but the actual outcome is NO;
2. where the model predicts NO, but the actual outcome is YES.

A confusion or *classification matrix* compares the actual outcomes to the predicted outcomes. The rows are labelled with actual outcomes while the columns are labelled with predicted outcomes.

```
# Confusion matrix for threshold of 0.5,0.2 and 0.7
table(model.train$Churn, pred.log.train > 0.2)
```

```
##
##       FALSE TRUE
##   No   1259 1575
##   Yes   109 2684
```

```
table(model.train$Churn, pred.log.train > 0.5)
```

```
##
##       FALSE TRUE
##   No   2087  747
##   Yes   620 2173
```

```
table(model.train$Churn, pred.log.train > 0.7)
```

```
##
##       FALSE TRUE
##   No   2517  317
##   Yes  1237 1556
```

These are tables that are used to describe the performance of a classification model on a set of data for which the true values are known.

- True Positive: You predicted positive and it's true;
- True Negative: You predicted negative and it's true;
- False Positive: You predicted positive and it's false;
- False Negative: You predicted negative and it's false.

We use the Receiver Operator Characteristic curve, or *ROC curve*, to decide which value of the threshold is best. The best threshold (or cutoff) point to be used in glm models is the point which maximises the specificity and the sensitivity. This threshold point might not give the highest prediction in your model, but it wouldn't be biased towards positives or negatives.

Recall that we made predictions on our training set and called them *pred.log.train*; we'll use these predictions to create our ROC curve.

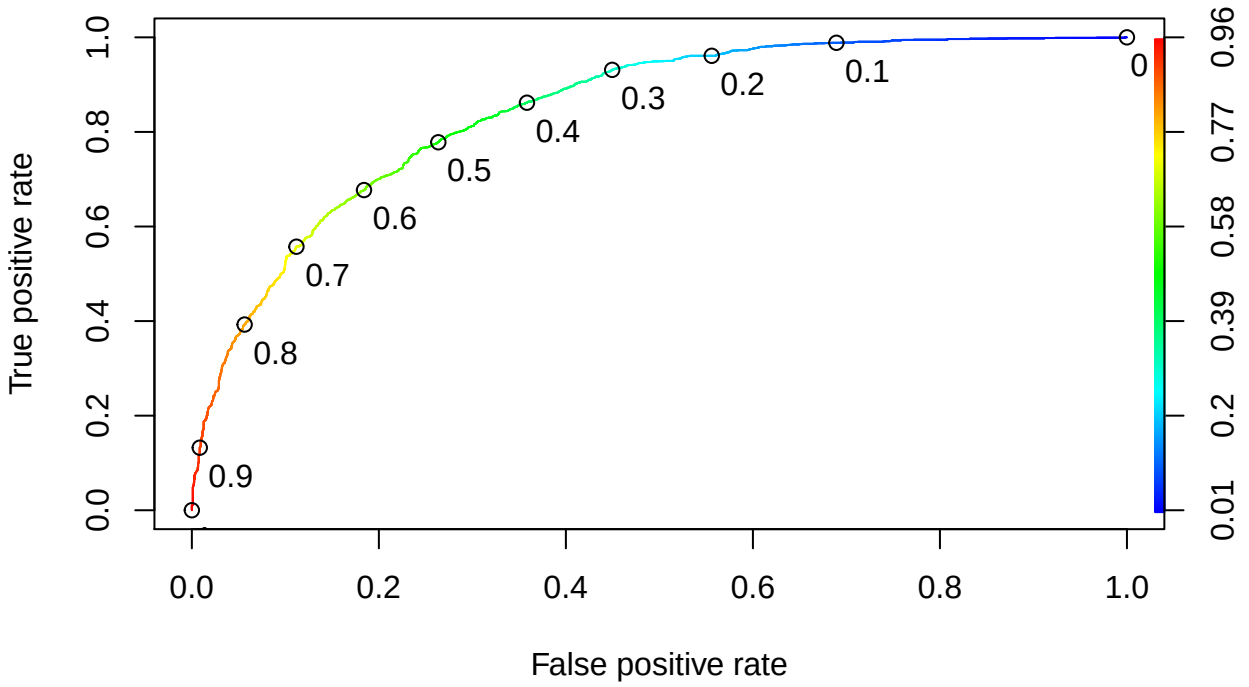
```
ROCRpred = ROCR::prediction(pred.log.train, model.train$Churn)
```

The first argument in the chunk is the predictions we made with our model, which we called pred.log.train. The second argument is the true outcomes of our data points, which in our case, model.train$Churn.

```
ROCRperf = ROCR::performance(ROCRpred, "tpr", "fpr")
```

This function takes as arguments the output of the prediction function, and then what we want on the x and y-axes (True positive rate and False posite rate).

Now, we just plot the output of the performance function:



Given the graph we would choose between 0.5 and 0.6. The sensitivity, or true positive rate of the model, is shown on the y-axis. The false positive rate, or 1 minus the specificity, is given on the x-axis. The line shows how these two outcome measures vary with different threshold values.

Now we perform the Prediction on Test Set and to do that we used the threshold value of 0.6 (that we have just computed) and we obtain the following confusion matrix, on the test set.

```
log.pred.test = predict(LogModel, type = "response", newdata = model.test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  850 113
##        Yes 182 260
##
##                Accuracy : 0.79
##                  95% CI : (0.7678, 0.8111)
##     No Information Rate : 0.7345
##     P-Value [Acc > NIR] : 8.182e-07
##
##                   Kappa : 0.4917
##
##  Mcnemar's Test P-Value : 7.523e-05
##
##             Sensitivity : 0.8236
##             Specificity : 0.6971
##          Pos Pred Value : 0.8827
##          Neg Pred Value : 0.5882
##              Prevalence : 0.7345
##          Detection Rate : 0.6050
##    Detection Prevalence : 0.6854
##       Balanced Accuracy : 0.7603
```

```
##
##        'Positive' Class : No
##
```

Below we see that the newly implemented model has a good accuracy.

```
accuracy=round(cm$overall[1],2)

Log_acc = accuracy

cat("Logistic regression accuracy:",Log_acc)
```

```
## Logistic regression accuracy: 0.79
```

Model 2: Decision tree

A decision tree (also referred to as a classification tree or a regression tree) is a predictive model which is a mapping from observations about an item to conclusions about its target value. In the tree structures, leaves represent classifications (also referred to as labels), nonleaf nodes are features, and branches represent conjunctions of features that lead to the classifications. This algorithm looks through the entire dataset to find the best variable (and the best split) to divide the data in 2 parts and create 2 different subsets. We predict that each observation belongs to the most commonly occuring class of training observations in the region to which it belongs. It assigns an observation in a given region to the most commonly occurring class of training observation in that region.

It is a **very interpreable model**: looking at the splits we understand how the prediction process is developed.

Key concepts that we will use in our analysis are:

- Root Node: it represents entire population or sample and this further gets divided into two or more homogeneous sets;
- Splitting: it is a process of dividing a node into two or more sub-nodes;
- Decision Node: when a sub-node splits into further sub-nodes, then it is called decision node;
- Leaf/Terminal Node: nodes do not split is called Leaf or Terminal node;
- Pruning*: when we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting;
- Branch/Sub-Tree: a sub section of entire tree is called branch or sub-tree;
- Parent and Child Node: a node, which is divided into sub-nodes is called parent node of sub-nodes whereas sub-nodes are the child of parent node.

Also we always need to validate our model. We can't just fit the model to the training data and it would accurately work for some new data never seen before. Maybe we can have good predictions on the training set but likely to overfit the data, leading to poor set performance. A smaller tree with fewer splits could avoid that and this process is called **pruning**.

We have also decided, in this section dedicated to decision trees, to implement 3 different **cross validation** techniques and see if they obtain very different results on decision trees.

We will therefore see these three techniques:

1. Leave-one-out cross validation (LOOCV)
2. k-fold cross validation
3. Validation Approach

1) LOOCV

Leave-one-out cross-validation is a special case of cross-validation where the number of folds equals the number of instances in the data set. Thus, the learning algorithm is applied once for each instance,using all other instances as a training set and using the selected instance as a single-item test set. Probably more inclined to overfitting since at each iteration we fit a model with n-1 observations.

```
set.seed(2020)
fit=tree(Churn~.,data=model.complete,method = 'gini')
fit
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 7032 9679.0 No ( 0.54977 0.45023 )
##    2) Contract: One year,Two year 2537 2094.0 No ( 0.85574 0.14426 )
##      4) InternetService: DSL,No 1712  910.1 No ( 0.92523 0.07477 ) *
##      5) InternetService: Fiber optic 825  991.3 No ( 0.71152 0.28848 ) *
##    3) Contract: Month-to-month 4495 5957.0 Yes ( 0.37709 0.62291 )
##      6) InternetService: DSL,No 1806 2499.0 No ( 0.52602 0.47398 )
##       12) tenure_groups: 2y,3y,4y,5y+ 603  728.2 No ( 0.70813 0.29187 ) *
##       13) tenure_groups: 1y 1203 1647.0 Yes ( 0.43475 0.56525 ) *
##      7) InternetService: Fiber optic 2689 3174.0 Yes ( 0.27705 0.72295 )
##       14) tenure_groups: 2y,3y,4y,5y+ 1391 1860.0 Yes ( 0.38965 0.61035 ) *
##       15) tenure_groups: 1y 1298 1126.0 Yes ( 0.15639 0.84361 ) *
```

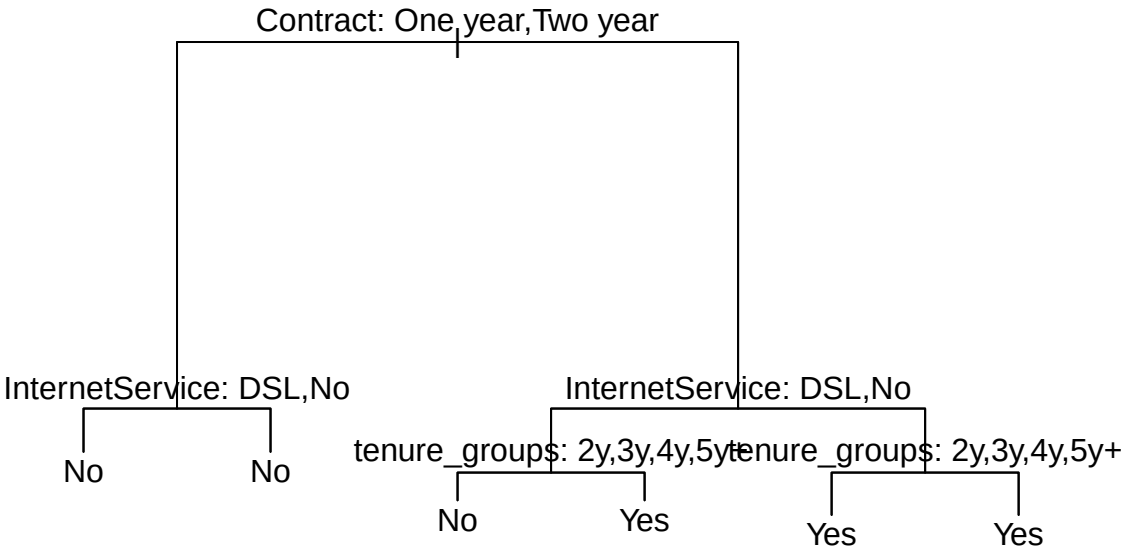
For classification problems, the Gini coefficient function is used which provides an indication of how "pure" the leaf nodes are.

```
summary(fit)
```

```
##
## Classification tree:
## tree(formula = Churn ~ ., data = model.complete, method = "gini")
## Variables actually used in tree construction:
## [1] "Contract"        "InternetService" "tenure_groups"
## Number of terminal nodes:  6
## Residual mean deviance:  1.034 = 7263 / 7026
## Misclassification error rate: 0.2574 = 1810 / 7032
```

In this output we see the variables involved in the construction of the tree, that has five terminal nodes and its misclassification error rate.

Graphically our tree is as follows:



In the case of a classification tree, the argument type="class" instructs R to return the actual class prediction.

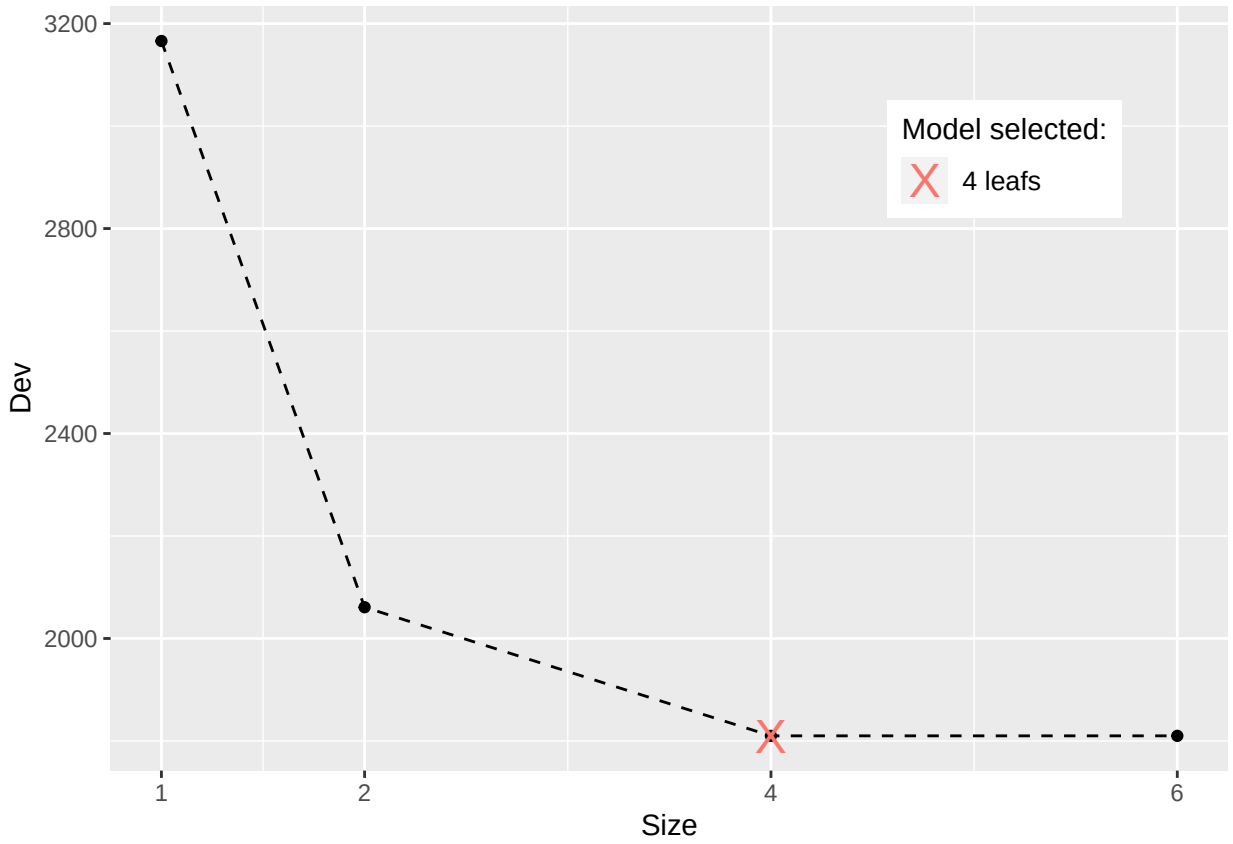
```
set.seed(2020)
cv.fit.l=cv.tree(fit,FUN=prune.misclass,K=length(model.complete))
#using k=observations we implement the LOOCV
cv.fit.l
```

```
## $size
## [1] 6 4 2 1
##
## $dev
## [1] 1810 1810 2061 3166
##
## $k
## [1]   -Inf    0.0  125.5 1105.0
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

To adapt classification trees we used the FUN = prune.misclass argument (in which case the error rate will be the guiding criterion). The output of cv.tree() will report:

- the number of terminal nodes of each tree considered (size);
- the corresponding error rate (dev);
- other parameters.

Dev corresponds to the cross-validation error rate in this instance. The tree with 4 terminal nodes results in the lowest cross-validation error rate.



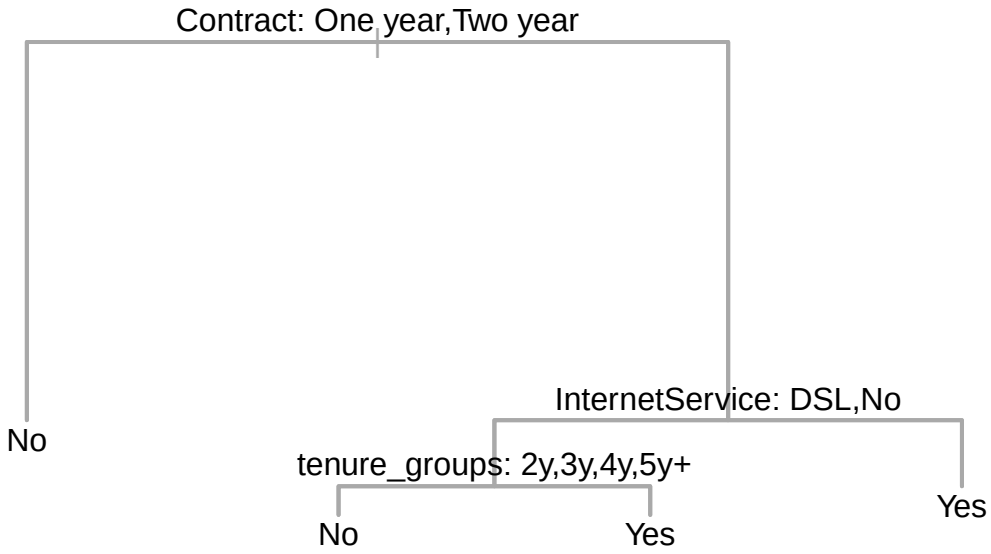
As you can see from the plot, we note that after Size = 4 the decrease in Dev is much less noticeable and it tends to stabilize in the subsequent nodes. This is why we then prune and implement a tree that stops at size = 4.

```
prune.fit=prune.misclass(fit,best=4)
summary(prune.fit)
```

```
##
## Classification tree:
## snip.tree(tree = fit, nodes = c(2L, 7L))
## Variables actually used in tree construction:
## [1] "Contract"        "InternetService" "tenure_groups"
## Number of terminal nodes:  4
## Residual mean deviance:  1.087 = 7643 / 7028
## Misclassification error rate: 0.2574 = 1810 / 7032
```

We can see the Residual mean deviance and the Misclassification error. Furthermore, from this summary we also see the variables currently involved and the type of tree (logically a classification because our variable is binary categorical).

A very basic but at the same time exhaustive graphic representation is the following:



So let's predict using the tree we just implemented.

```
fitted=predict(prune.fit, newdata=model.complete,type='class')
```

So let's see the confusion matrix, already explained above. The result is good as a good percentage of the observations are concentrated in True Positives and True Negatives.

```
tree_l = confusionMatrix(data=fitted,
                         reference=model.complete$Churn)
tree_l
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  2598  542
##        Yes 1268 2624
##
##                Accuracy : 0.7426
##                  95% CI : (0.7322, 0.7528)
##     No Information Rate : 0.5498
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4906
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.6720
##             Specificity : 0.8288
##          Pos Pred Value : 0.8274
##          Neg Pred Value : 0.6742
##              Prevalence : 0.5498
##          Detection Rate : 0.3695
##    Detection Prevalence : 0.4465
##       Balanced Accuracy : 0.7504
##
##        'Positive' Class : No
##
```

```
tree_l_acc = tree_l$overall[1]
tree_l_acc
```

```
##  Accuracy
## 0.7426052
```

2) K-fold cross validation

In K Fold cross validation, the dataset is divided into k different subsets. The method is repeated k times, such that each time, one of the k subsets is used as the test set and the other k-1 subsets are put together to form a training set.

```
set.seed(2020)

model_T=train(
  Churn ~., data = model.train, method = "rpart",
  trControl = trainControl("cv", number = 10)
)
```

```
# Best tuning parameter
model_T$bestTune
```

```
##           cp
## 1 0.01253133
```

The complexity parameter (cp) is used to control the size of the decision tree and to select the optimal tree size.

```
model_T
```

```
## CART
##
## 5627 samples
##   18 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5065, 5063, 5064, 5065, 5064, 5065, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.01253133  0.7181358  0.43710110
##   0.17472252  0.6772592  0.35684591
##   0.26029359  0.5495506  0.09616001
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.01253133.
```

Make predictions on the test data using test set:

```
predicted.classes_T = model_T %>% predict(model.test)
```

Compute model prediction accuracy rate and the confusion matrix:

```
Tree_k = confusionMatrix(data=predicted.classes_T,
                         reference=model.test$Churn)
```

```
Tree_k
```

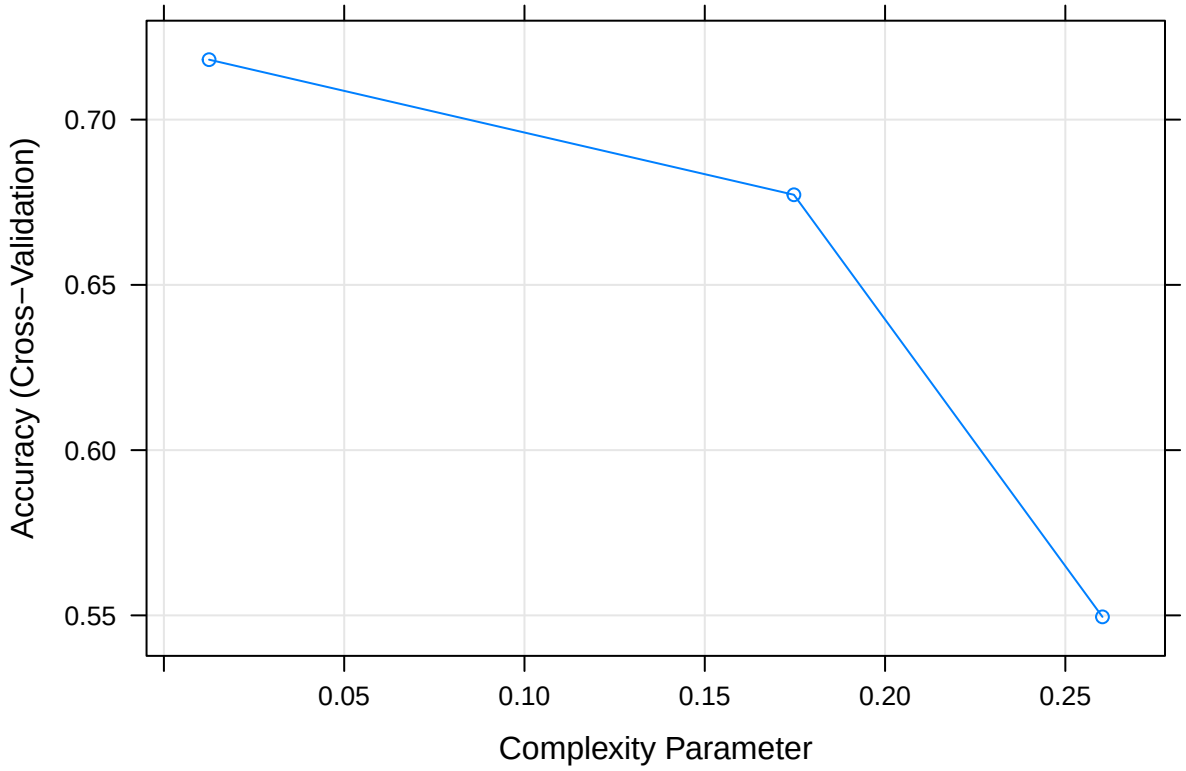
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  591  42
##        Yes 441 331
##
##                Accuracy : 0.6562
##                  95% CI : (0.6307, 0.6811)
##     No Information Rate : 0.7345
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3429
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.5727
##             Specificity : 0.8874
##          Pos Pred Value : 0.9336
##          Neg Pred Value : 0.4288
##              Prevalence : 0.7345
##          Detection Rate : 0.4206
##    Detection Prevalence : 0.4505
##       Balanced Accuracy : 0.7300
##
##        'Positive' Class : No
##
```

```
Tree_k_acc = Tree_k$overall[1]
Tree_k_acc
```

```
##  Accuracy
## 0.6562278
```

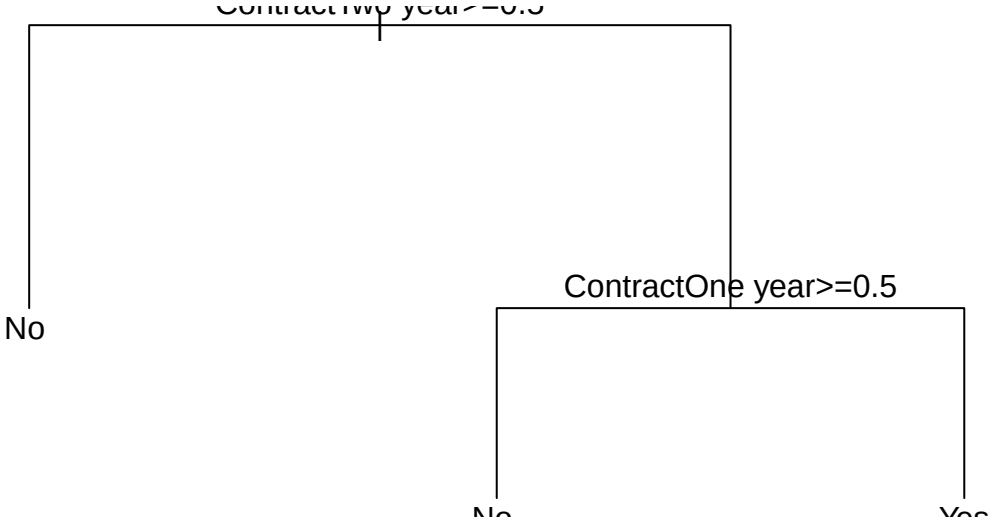
We see that the accuracy, using this approach on the data we have available, decreases.

Also, looking at the plot below, as the CP increases, the accuracy decreases.



The newly implemented tree is the following:

```
model_final=model_T$finalModel
plot(model_final,col='darkgray',cex=2)
text(model_final,pretty=0)
```



3)VALIDATION APPROACH

Again we proceed as before, but using the train and test set partition, using the rpart package. So let's create a decision tree on the model.train using rpart.control to have various parameters that control aspects of the rpart fit. The meaning of the chosen hyperparameters is as follows:

- minsplit: the minimum number of observations that must exist in a node in order for a split to be

attempted.

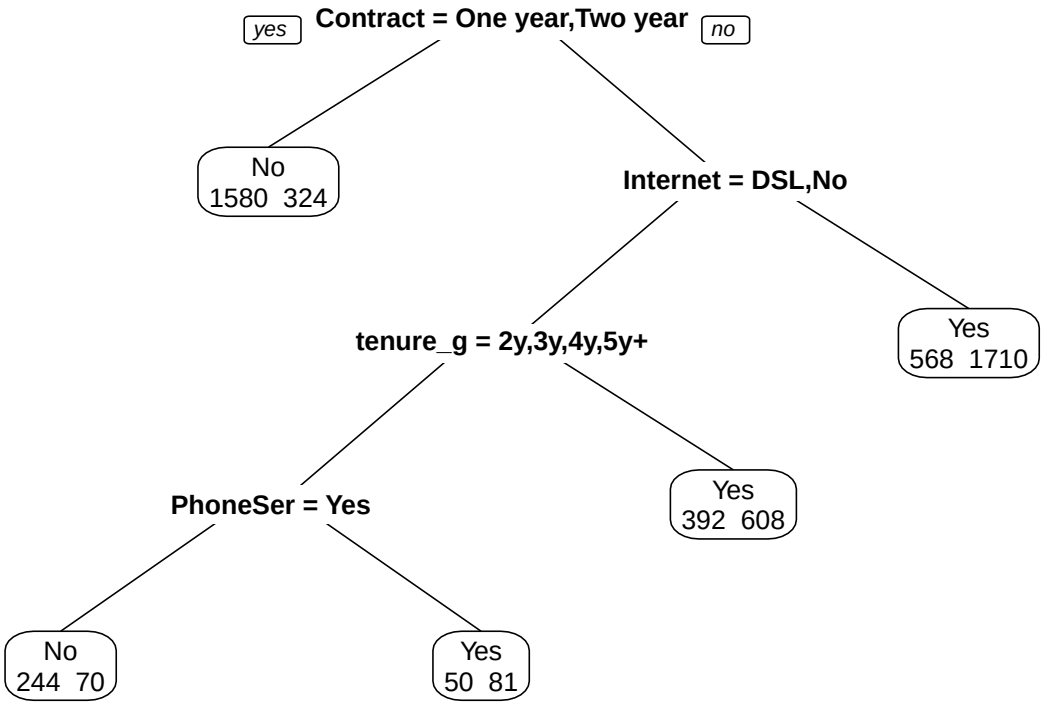
- minbucket: the minimum number of observations in any terminal leaf node;
- maxdepth: set the maximum depth of any node of the final tree, with the root node counted as depth 0.
- usesurrogate: how to use surrogates in the splitting process; for value 2 ,if all surrogates are missing, then send the observation in the majority direction. A value of 0 corresponds to the action of tree, and 2 to the recommendations of Breiman et.al book.

```
set.seed(2020)
mtree = rpart(Churn~., data = model.train, method="class", control = rpart.control(minsplit = 20, minbu
mtree
```

```
## n= 5627
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 5627 2793 No (0.5036431 0.4963569)
##    2) Contract=One year,Two year 1904  324 No (0.8298319 0.1701681) *
##    3) Contract=Month-to-month 3723 1254 Yes (0.3368251 0.6631749)
##      6) InternetService=DSL,No 1445  686 Yes (0.4747405 0.5252595)
##       12) tenure_groups=2y,3y,4y,5y+ 445  151 No (0.6606742 0.3393258)
##         24) PhoneService=Yes 314   70 No (0.7770701 0.2229299) *
##         25) PhoneService=No 131   50 Yes (0.3816794 0.6183206) *
##       13) tenure_groups=1y 1000  392 Yes (0.3920000 0.6080000) *
##      7) InternetService=Fiber optic 2278  568 Yes (0.2493415 0.7506585) *
```

The tree plot is the following:

```
prp(mtree, faclen = 0, cex = 0.8, extra = 1)
```



Let's do the appropriate pruning using the best complexity parameter:

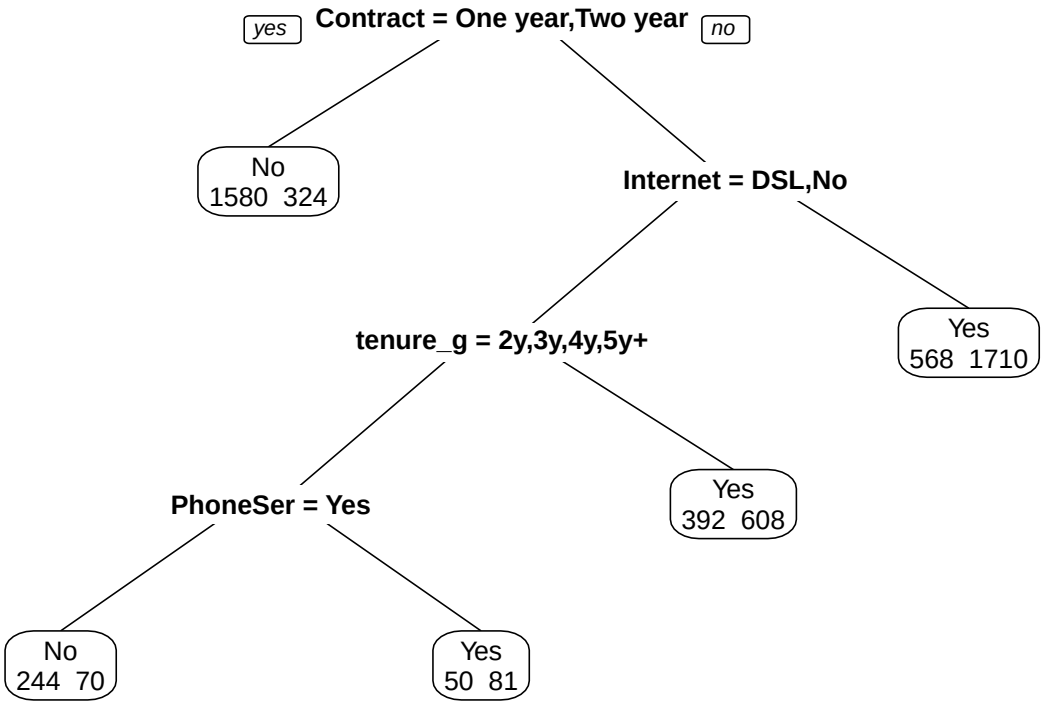
```
#pruning:
printcp(mtree)
```

```
##
## Classification tree:
## rpart(formula = Churn ~ ., data = model.train, method = "class",
##     control = rpart.control(minsplit = 20, minbucket = 7, maxdepth = 10,
##         usesurrogate = 2, xval = 10))
##
## Variables actually used in tree construction:
## [1] Contract        InternetService PhoneService    tenure_groups
##
## Root node error: 2793/5627 = 0.49636
##
## n= 5627
##
##         CP nsplit rel error  xerror     xstd
## 1 0.435016      0   1.00000 1.03294 0.013424
## 2 0.025600      1   0.56498 0.56498 0.012065
## 3 0.011099      3   0.51378 0.51378 0.011707
## 4 0.010000      4   0.50269 0.51199 0.011693
```

```
bestcp=mtree$cptable[which.min(mtree$cptable[,"xerror"]),"CP"]
```

```
# Prune the tree using the best cp.
pruned= prune(mtree, cp = bestcp)
```

Plot pruned tree:



Confusion matrix (on training data):

```
conf.matrix = table(model.train$Churn, predict(pruned,type="class"))
rownames(conf.matrix) = paste("Actual", rownames(conf.matrix), sep = ":")
```

```
colnames(conf.matrix) = paste("Pred", colnames(conf.matrix), sep = ":")
print(conf.matrix)
```

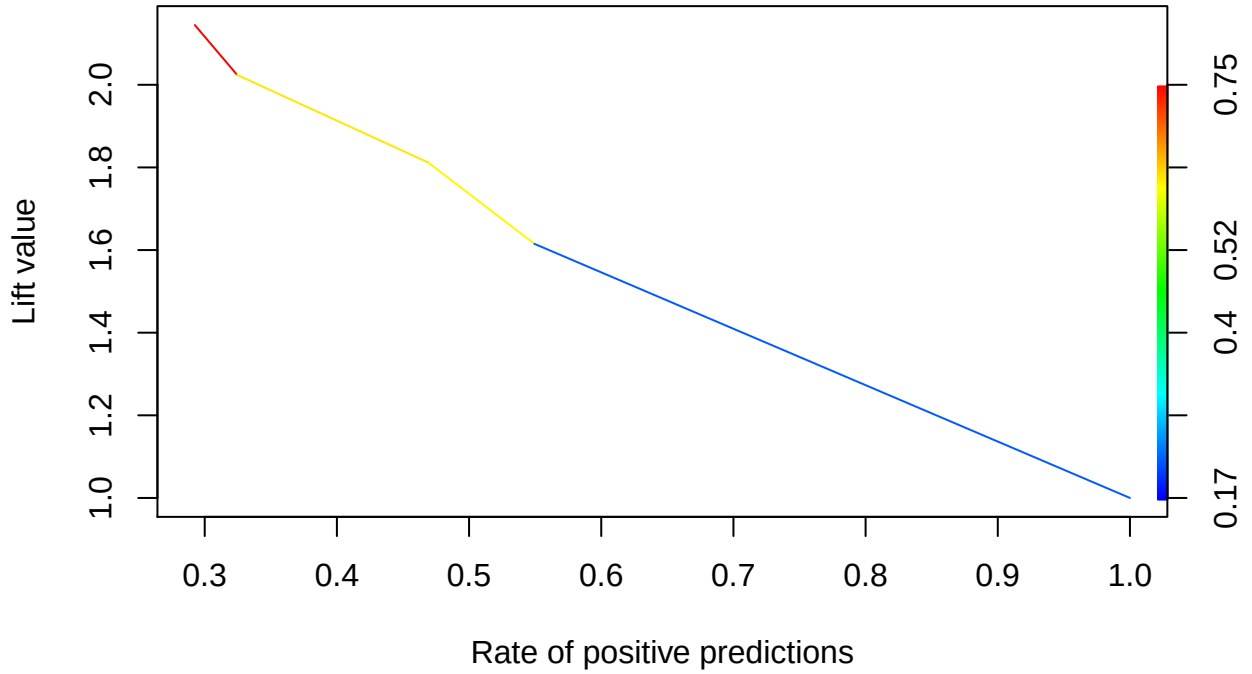
```
##
##              Pred:No Pred:Yes
##   Actual:No     1824     1010
##   Actual:Yes     394     2399
```

```
#Scoring
val1 = predict(pruned, model.test, type = "prob")
#Storing Model Performance Scores
pred_val=ROCR::prediction(val1[,2],model.test$Churn)

# Calculating Area under Curve
perf_val=ROCR::performance(pred_val,"auc")
```

The random model is defined by a bisector, the area subtended by my model is greater than the area subtended from the bisector and therefore clearly superior to the random model.

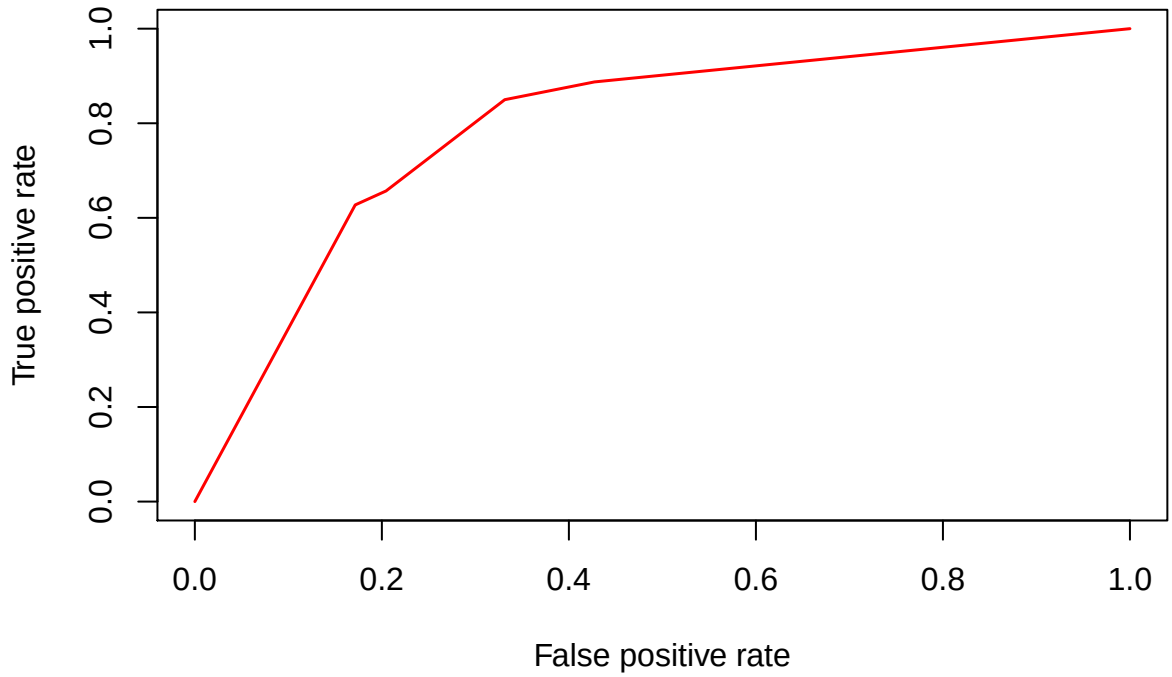
Plotting Lift curve:



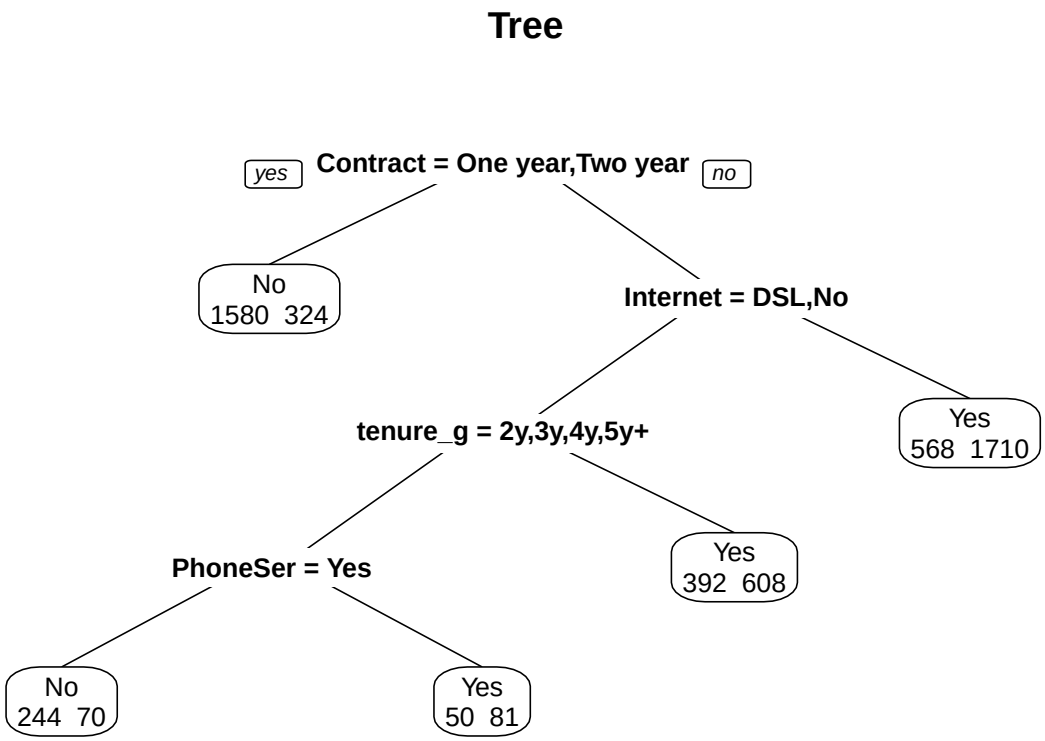
A lift curve is a way of visualizing the performance of a classification model and it shows shows the ratio of a model to a random guess.

```
# Calculating True Positive and False Positive Rate
perf_val = ROCR::performance(pred_val, "tpr", "fpr")
```

Plot the ROC curve:




```
#pruned tree plot
prp(pruned, faclen = 0, cex = 0.8, extra = 1,main="Tree")
```




```
# Accuracy on test set
Tc = confusionMatrix(data=predict(pruned, newdata = model.test,  type="class"),
                     reference=model.test$Churn)

Tree_acc = Tc$overall[1]
Tree_acc
```

```
## Accuracy
## 0.716726
```

MODEL 3: BAGGING TREES AND RANDOM FORREST

Random Forests algorithm is a classifier based on primarily two methods: bagging and random subspace method. The fundamental difference is that in Random forests only a subset of features are selected at random out of the total and the best split feature from the subset is used to split each node in a tree, unlike in bagging where all features are considered for splitting a node.

BAGGING

Bagging is a special case of a random forest where the randomForest() function from the randomForest library can be used to build a bagged model or a random forest.

```
set.seed(2020)
bag=randomForest(Churn~.,data=model.train,
                 mtry=18,
                 importance=TRUE,
                 ntree=200)
```

- mtry=18: specifies to use all 18 predictors of thedataset at each split of the trees (in the case of random forest we will reduce the number of predictors to consider) importance=TRUE : to build the measuresof influence of the variables;
- ntree=200: number of bootstrap samples.

```
print(bag)
```

```
##
## Call:
##  randomForest(formula = Churn ~ ., data = model.train, mtry = 18,      importance = TRUE, ntree = 20
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 18
##
##         OOB estimate of  error rate: 17.2%
## Confusion matrix:
##       No  Yes class.error
## No  2265  569   0.2007763
## Yes  399 2394   0.1428571
```

RANDOM FOREST

It is a classification model based on the training of m different decision trees:

- each decision tree is fit on a random subset of the training set and a random subset of the input variables.
- the output of the random forest is the average of the decision tree's outputs.

It is a semi-interpretable model: we know which features are the most influential, but not how they change the result. It has a low risk of overfitting.

To grow a forest, you proceed in the same way but using a lower number of mtry than the total number. The default is mtry equal to the root of the total number of variables, but instead we will use cross validation to find the most suitable value

```
# define training control
train_control = trainControl(method="cv", number=10)

# train the model
tunegrid = expand.grid(.mtry=c(7,8,12,15))
set.seed(2020)
model = train(Churn~., data=model.train, trControl=train_control,method = "rf",
              tuneGrid = tunegrid)
```

```
# summarize results
print(model)
```

```
## Random Forest
##
## 5627 samples
##   18 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5065, 5063, 5064, 5065, 5064, 5065, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    7    0.8503636  0.7008840
##    8    0.8480548  0.6962636
##   12    0.8420113  0.6841488
##   15    0.8363227  0.6727786
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 7.
```

In the output above, the optimal mtry number is specified after cross validation. We have shown only a subsaple of all the cross validation trials so to not make the scrip too computationally heavy.

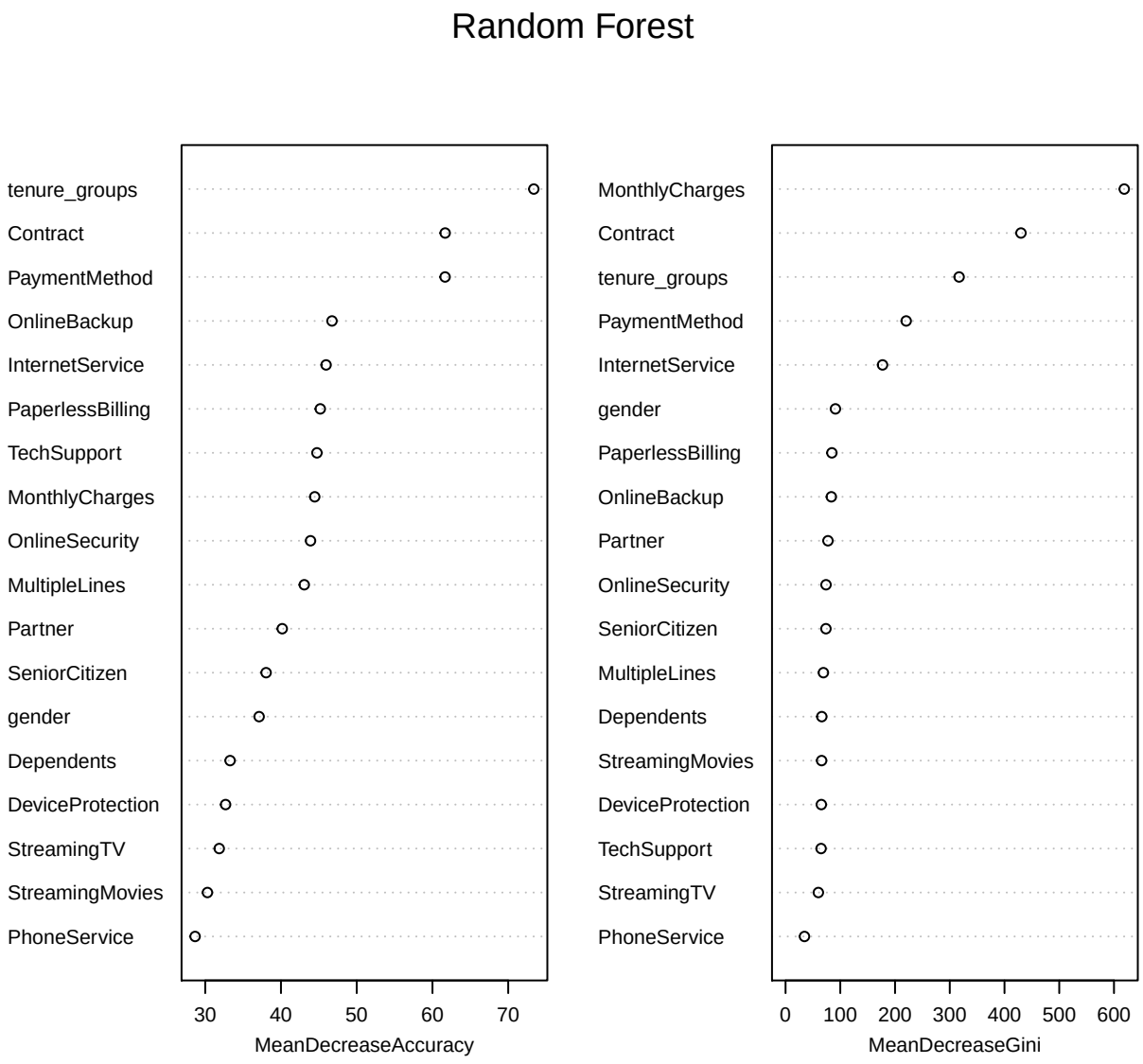
We, therefore, implement a random forrest with mtry equal to 8.

```
set.seed(2020)
rf=randomForest(Churn~.,data=model.train,
                mtry=8,
                importance=TRUE,
                ntree=200)
rf
```

```
##
## Call:
##  randomForest(formula = Churn ~ ., data = model.train, mtry = 8,      importance = TRUE, ntree = 200)
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 8
##
##         OOB estimate of  error rate: 15.3%
## Confusion matrix:
##       No  Yes class.error
## No  2287  547   0.1930134
## Yes  314 2479   0.1124239
```

For classification, the purity of a node is measured with the gini index. Every time a split of a node is made on variable the gini impurity criterion for the two descendent nodes is less than the parent node. Adding up the gini decreases for each individual variable over all trees in the forest gives a fast variable importance that is often very consistent with the permutation importance measure.

It follows the variable importance plot.



So to make predictions, given a set of predictors, we can use the predict() function.

```
rf.predict=predict(rf,model.test)
```

```
model.test$pred.Churn=predict(rf,model.test)

Rfc = confusionMatrix(data=rf.predict,
                      reference=model.test$Churn)

Rfc_acc = Rfc$overall[1]
```

I create a new column so to make a comparison between predicted and original values. The head function gives me a preview.

```
Churn.pred = data.frame(Original=model.test$Churn,Pred=model.test$pred.Churn)
```

```
head(Churn.pred)
```

```
##   Original Pred
## 1       No  Yes
## 2       No   No
## 3       No  Yes
## 4       No   No
## 5      Yes  Yes
## 6       No   No
```

MODEL 4: XGBOOST

XGBoost stands for Extreme Gradient Boosting; it is a specific implementation of the Gradient Boosting method which uses more accurate approximations to find the best tree model. Gradient Boosting is an iterative approach that creates multiple trees. The trees are developed sequentially, non in parallel (like Random Forest). While regular gradient boosting uses the loss function of our base model (decision tree) as a proxy for minimizing the error of the overall model, XGBoost uses the 2nd order derivative as an approximation.

Boosting has different tuning parameters including:

- The number of trees B
- The shrinkage parameter lambda
- The number of splits in each tree.

There are different variants of boosting. Xgboost is one of them and it is the most commonly used boosting technique, which involves resampling of observations and columns in each round. It offers the best performance. Boosting can be used for both classification and regression problems. Of course, as mentioned before, we are in a classification setting.

We'll use the caret workflow,to automatically adjust the model parameter values, and fit the final best boosted tree that explains the best our Churn dataset.

```
set.seed(2020)
#Fit the model on the training set
#trControl, to set up 10-fold cross validation
model=train(
  Churn ~., data = model.train, method = "xgbTree",
  trControl = trainControl("cv", number = 10)
)
```

Best tuning parameters che otteniamo sono:

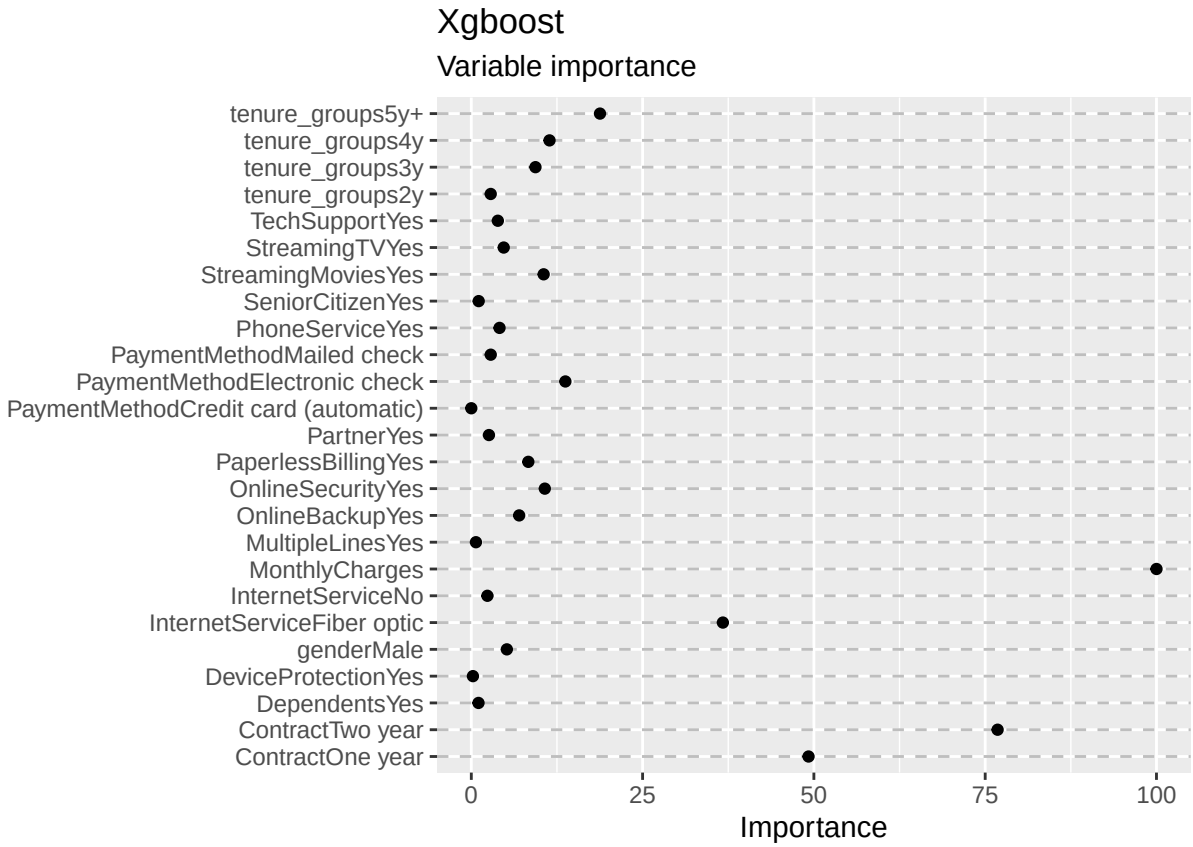
```
model$bestTune
```

```
##    nrounds max_depth eta gamma colsample_bytree min_child_weight subsample
## 96     150         3 0.4     0              0.6                1      0.75
```

Make predictions on the test data:

```
predicted.classes = model %>% predict(model.test)
```

The function `varImp()` displays the importance of variables in percentage:



So let's see the last confusion matrix of our analysis and the accuracy of the model.

```
Xgb = confusionMatrix(data=predicted.classes,
                      reference=model.test$Churn)

Xgb_acc = Xgb$overall[1]
Xgb_acc
```

```
##  Accuracy
## 0.7338078
```

CONCLUSIONS

As we said at the beginning, this analysis sees as its main objective a part of data preparation and data investigation, implementation and comparison between different predictive models. We have seen what these models are, how they are implemented on the code and how to apply tuning and improvement techniques. As already introduced initially, it is not a business approach that sees a specific type of customer identified and a pre-established goal for a business need.

```
##          Logistic Tree.LOOCV Tree.k.fold.CV Tree.Train.Test RandomForest
## Accuracy     0.79  0.7426052      0.6562278        0.716726    0.7359431
##            Xgboost
## Accuracy 0.7338078
```

As a comparison method we use the accuracy, which is a statistical measure of how well a binary classification test correctly identifies or excludes a condition. It is the proportion of correct predictions among the total number of cases examined.Among all the models seen we see that no one can overcome Logistic Regression. So, to conclude, we can say that in this work logistic regression is the winning model.