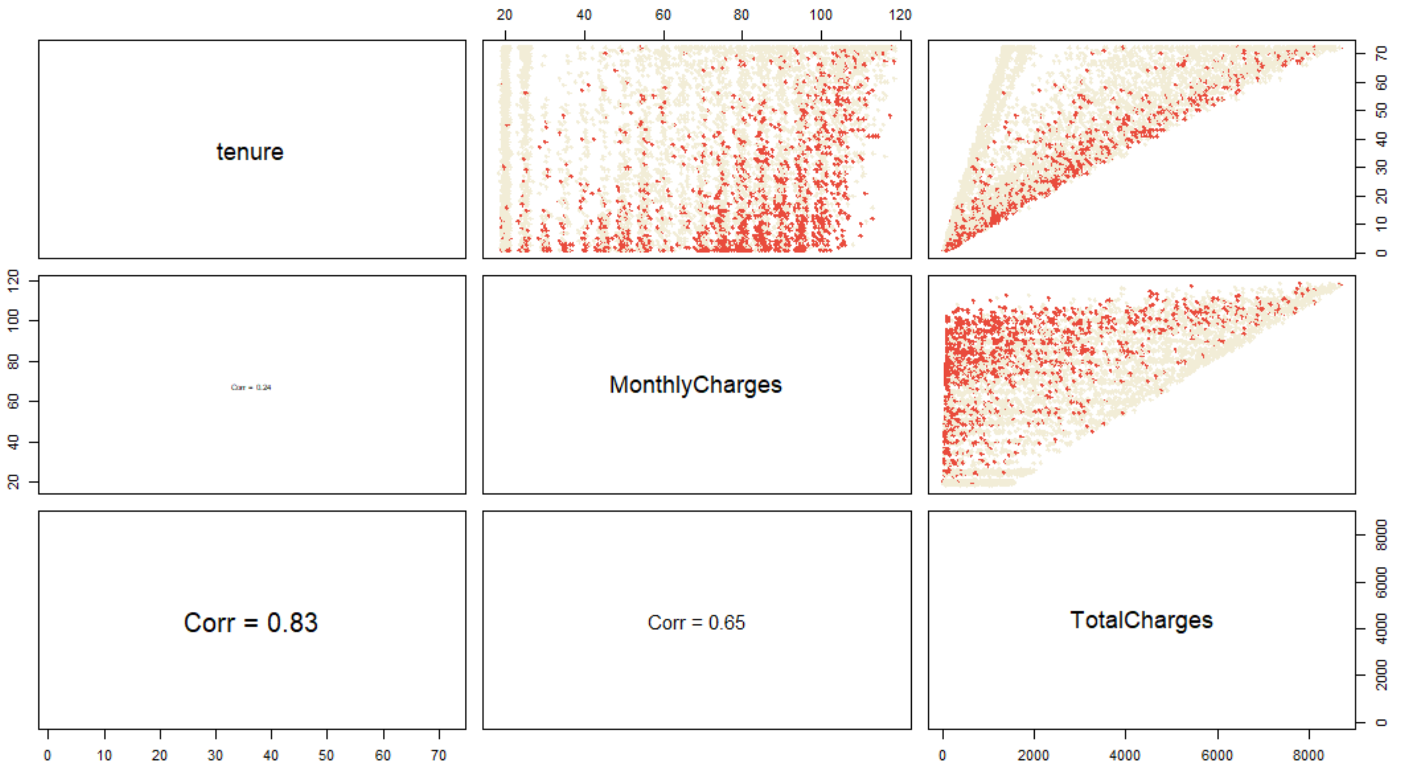
## DTI5126[EG] Fundamentals/Applied Data Science

## Assignment 2


### Hosam Mahmoud Ibrahim – 300327269

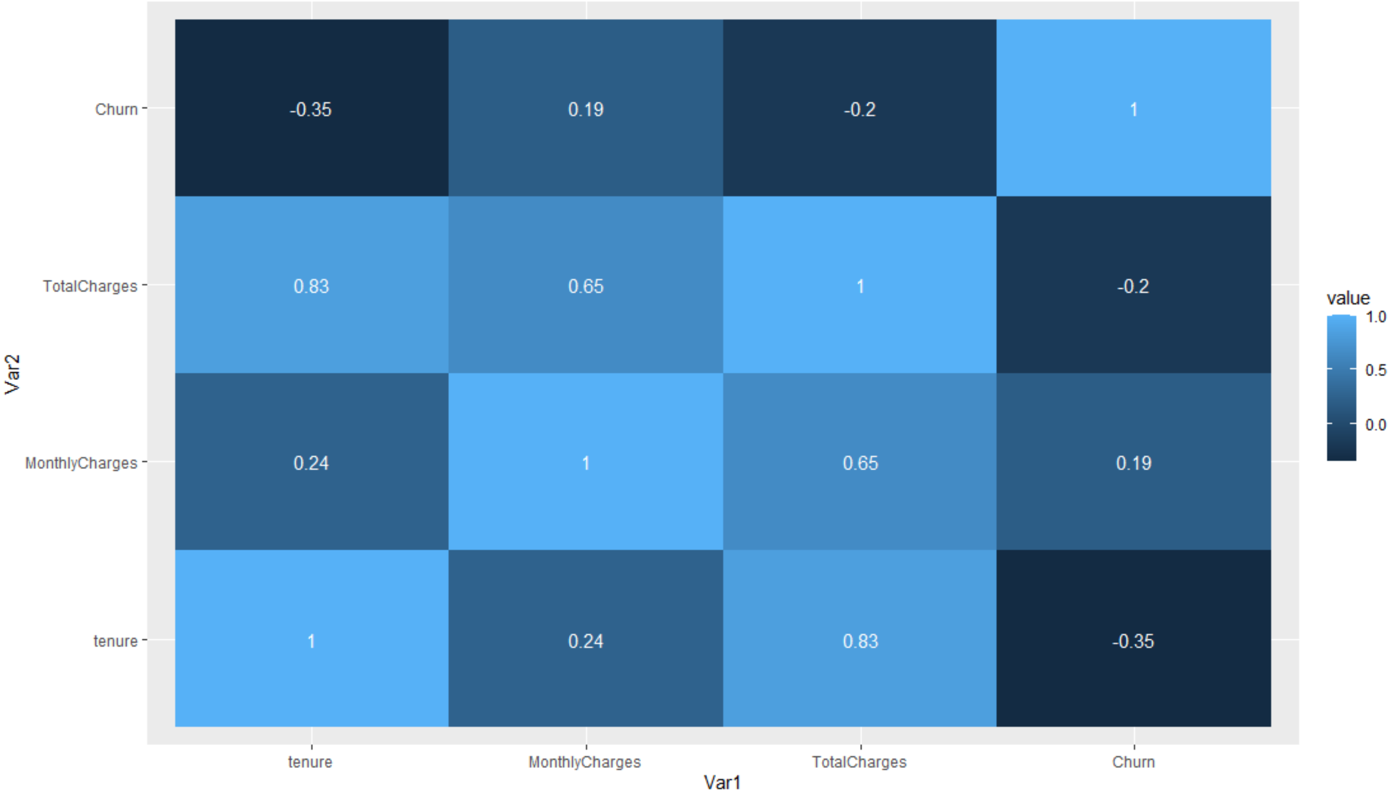

# i. Part A

1. **Q1:** I have plotted scatter matrix plot between the numeric values and the points is colored based on if it **Yes** or **No**, and to see how these columns are correlated together.



Here I have plotted heatmap with **numeric values** and **target value**

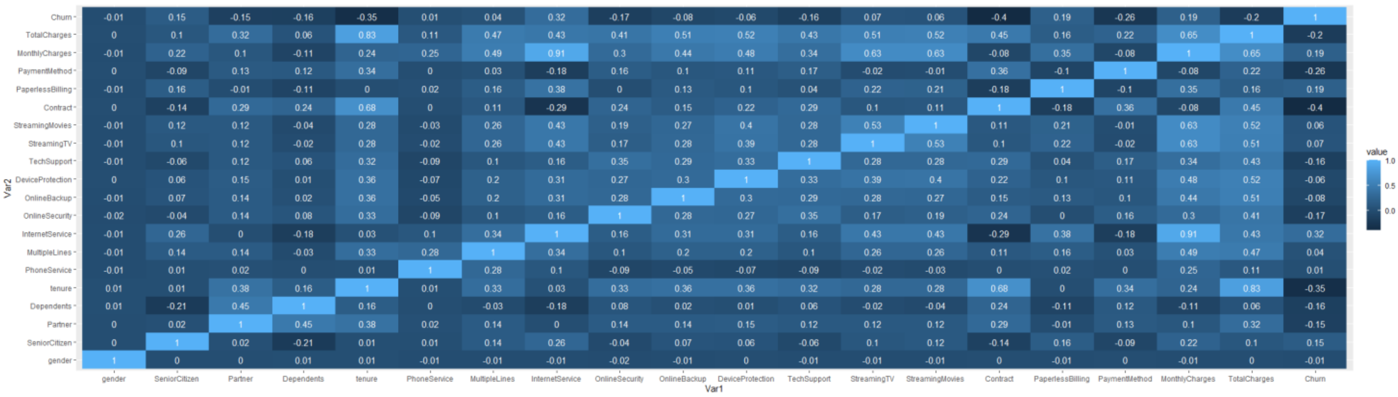
To see how they are correlated to the **target.**




Here I have plotted heatmap with **all values** and **target value**

To see how they are correlated to the **target**, and to each other also.

Note* (I have attached the images in the submission)



2. **Q2:** first I have deleted the duplicated rows, after that I have printed which columns is have null values, and I found that it's **TotalCharges** column only that contain **11 null values.**

```
# Delete Duplicated rows
FullData <- subset(FullData, !duplicated(FullData))
```

```
# Print which columns contain Missing Data
sapply(FullData, function(x) sum(is.na(x)))
```

```
> # Missing Data
> sapply(FullData, function(x) sum(is.na(x)))
          gender    SeniorCitizen          Partner       Dependents           tenure     PhoneService
               0                0                0                0                0                0
   MultipleLines  InternetService   OnlineSecurity     OnlineBackup DeviceProtection      TechSupport
               0                0                0                0                0                0
     StreamingTV  StreamingMovies         Contract PaperlessBilling    PaymentMethod   MonthlyCharges
               0                0                0                0                0                0
    TotalCharges            Churn
              11                0
```

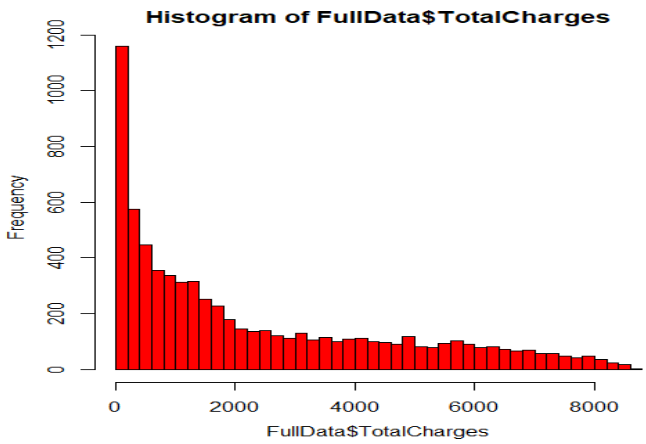
And these is the indices that **TotalCharges** had null values.

```
# Print the 11's Null values that in TotalCharges Column
print(subset(FullData, is.na(FullData$TotalCharges)))
```

```
         StreamingMovies Contract PaperlessBilling             PaymentMethod MonthlyCharges TotalCharges Churn
489                   No Two year              Yes Bank transfer (automatic)          52.55           NA    No
754  No internet service Two year               No              Mailed check          20.25           NA    No
937                  Yes Two year               No              Mailed check          80.85           NA    No
1083 No internet service Two year               No              Mailed check          25.75           NA    No
1341                  No Two year               No   Credit card (automatic)          56.05           NA    No
3332 No internet service Two year               No              Mailed check          19.85           NA    No
3827 No internet service Two year               No              Mailed check          25.35           NA    No
4381 No internet service Two year               No              Mailed check          20.00           NA    No
5219 No internet service One year              Yes              Mailed check          19.70           NA    No
6671                  No Two year               No              Mailed check          73.35           NA    No
6755                  No Two year              Yes Bank transfer (automatic)          61.90           NA    No
```

I have plotted the distribution of **TotalCharges** column, because I want to decide if I **drop** these rows, or I'll impute with **median** or **mean.**

```
# Check the Distribution of TotalCharges Column
hist(FullData$TotalCharges, breaks=50, col="red") # the distribution is Right Skewed
```



But at the end I have decided that I'll drop these 11 rows because they represent only **0.001566728** from the total dataset.

```
# Check the effect of these 11 customers in comparison to the total customers
sum(is.na(FullData$TotalCharges))/nrow(FullData)

# This 11 customers is 0.16% of our data which is too small, so i will drop these 11 rows.
FullData <- FullData[complete.cases(FullData),]
```

**Categorical data to numeric:** I change **"No phone service"** and **"No internet service"** to No.

```
# first I'll convert categorical column to string to be able in future to changes all NO to 0 and all YES to 1
FullData$SeniorCitizen <- as.factor(plyr::mapvalues(FullData$SeniorCitizen,
                                                     from=c("0","1"),
                                                     to=c("No", "Yes")))

# the column MultipleLines is related to PhoneService column so if  PhoneService == NO
# MultipleLines will be for sure = NO (no need to be named NO phone service)
FullData$MultipleLines <- as.factor(plyr::mapvalues(FullData$MultipleLines,
                                                     from=c("No phone service"),
                                                     to=c("No")))

# the same idea here if InternetService == NO, the other features that depend on it will be NO
# (no need to be named NO Internet Service)
```

```
# Categorical to numerical values
for(column in 1:19){
  FullData[,column] <- as.factor(plyr::mapvalues(FullData[,column],
                                                 from= c("No","Yes"), to= c(0,1)))
}
FullData$gender <- as.factor(plyr::mapvalues(FullData$gender,
                                             from= c("Female","Male"), to= c(0,1)))
FullData$InternetService <- as.factor(plyr::mapvalues(FullData$InternetService,
                                                      from= c("DSL","Fiber optic"), to= c(1,2)))
FullData$Contract <- as.factor(plyr::mapvalues(FullData$Contract,
                                               from= c("Month-to-month","One year","Two year"), to= c(0,1,2)))
FullData$PaymentMethod <- as.factor(plyr::mapvalues(FullData$PaymentMethod,
                                                    from= c("Electronic check","Mailed check","Bank transfer (automatic)","C
FullData$Churn <- as.factor(plyr::mapvalues(FullData$Churn,
                                            from= c("No","Yes"), to= c(0,1)))
```

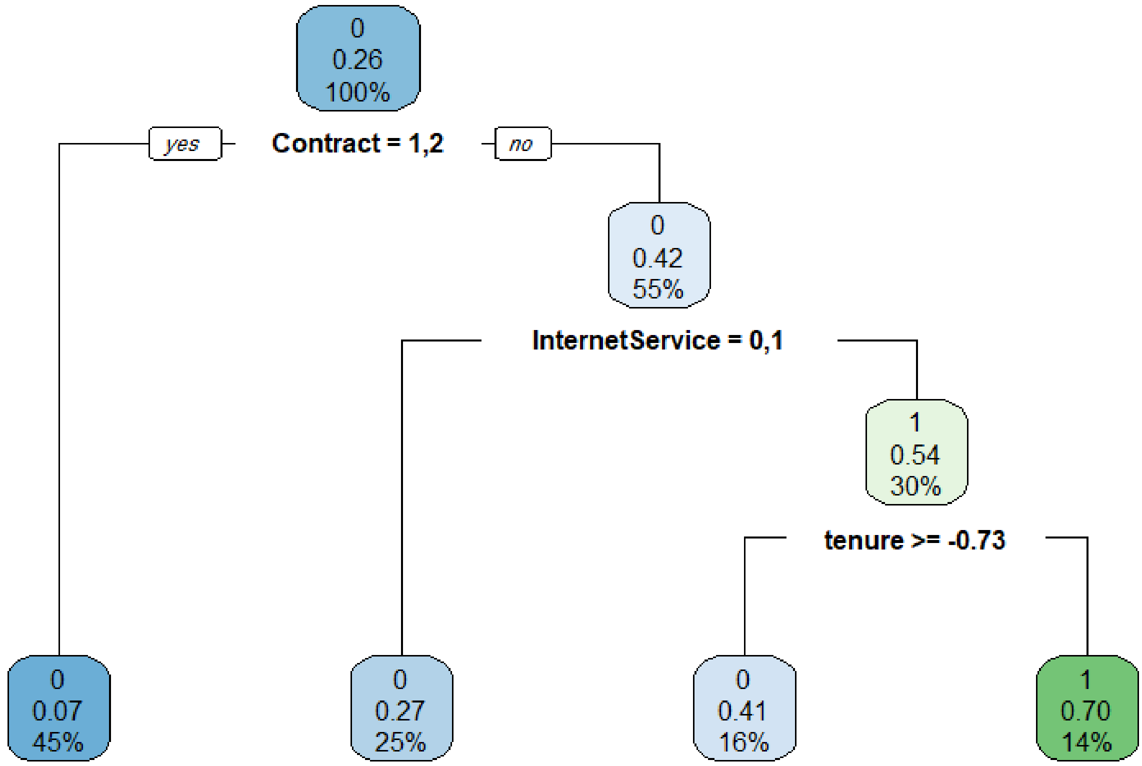
**Feature scaling (for numeric columns only):**

```
# Feature Scaling
FullData[c("tenure","MonthlyCharges","TotalCharges")] = scale(FullData[c("tenure","MonthlyCharges","TotalCharges")])
```

3. **Q3:** first I have splatted the data.

```
# Split the dataset
set.seed(123)
sample <- sample(c(TRUE, FALSE), nrow(FullData), replace=TRUE, prob=c(0.8,0.2))
Train  <- FullData[sample, ]
Test   <- FullData[!sample, ]
```

After that I have trained 2 Decision tree models the first one using **rpart** library and the second one using **cTree.**




**The tree of rpart model.**

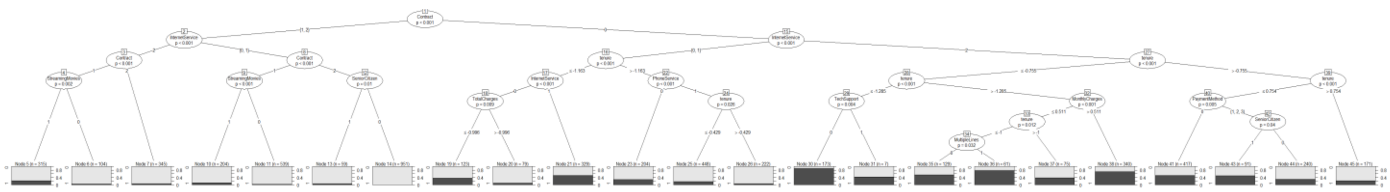
**Note* (I have plotted this tree before scaling to be able to interpret the results)**

***From this tree, we can interpret these points:***

- The contract column is the most important column.
- Customers who have stayed longer than 15 months are less likely to churn.
- Customers with DSL internet service are less likely to churn.
- Customers with month-to-month contracts are more likely to churn.

**cTree library tree:**

**Note* (I have attached the images in the submission)**



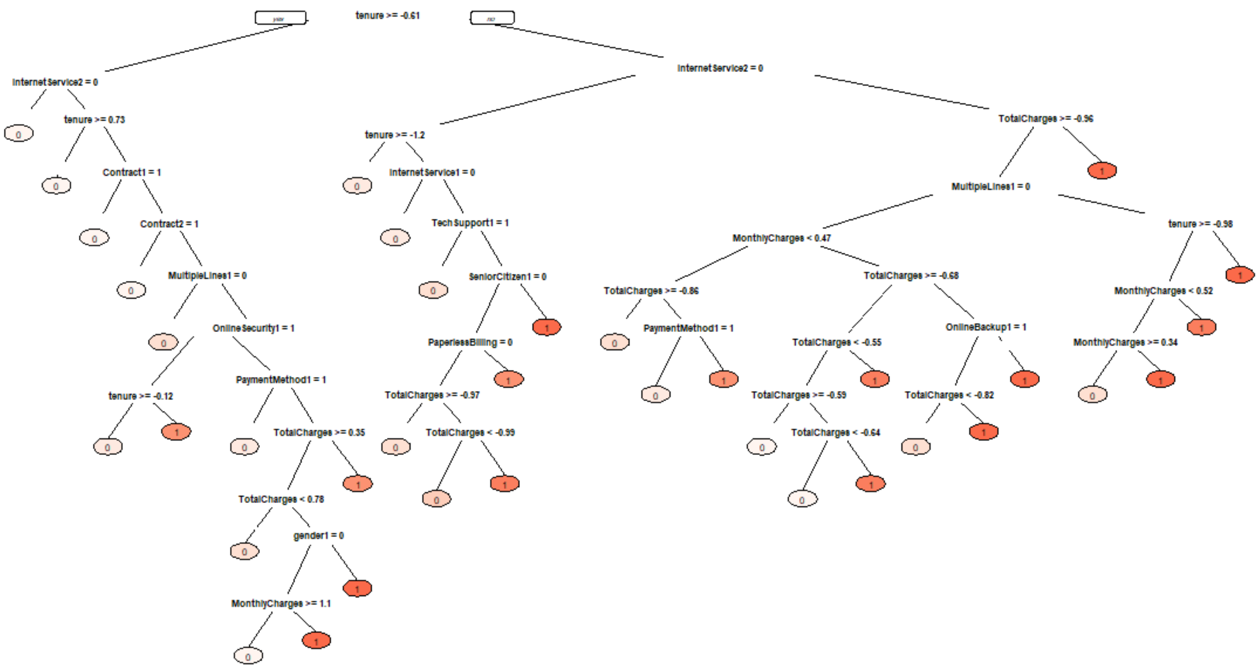
**Accuracy and confusion matrix.**

```
          Reference
Prediction   0   1
         0 936 247
         1  66 135

               Accuracy : 0.7738
                 95% CI : (0.7509, 0.7956)
    No Information Rate : 0.724
    P-Value [Acc > NIR] : 1.341e-05

                  Kappa : 0.3369

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

            Sensitivity : 0.9341
            Specificity : 0.3534
         Pos Pred Value : 0.7912
         Neg Pred Value : 0.6716
              Precision : 0.7912
                 Recall : 0.9341
                     F1 : 0.8568
             Prevalence : 0.7240
         Detection Rate : 0.6763
   Detection Prevalence : 0.8548
      Balanced Accuracy : 0.6438
```

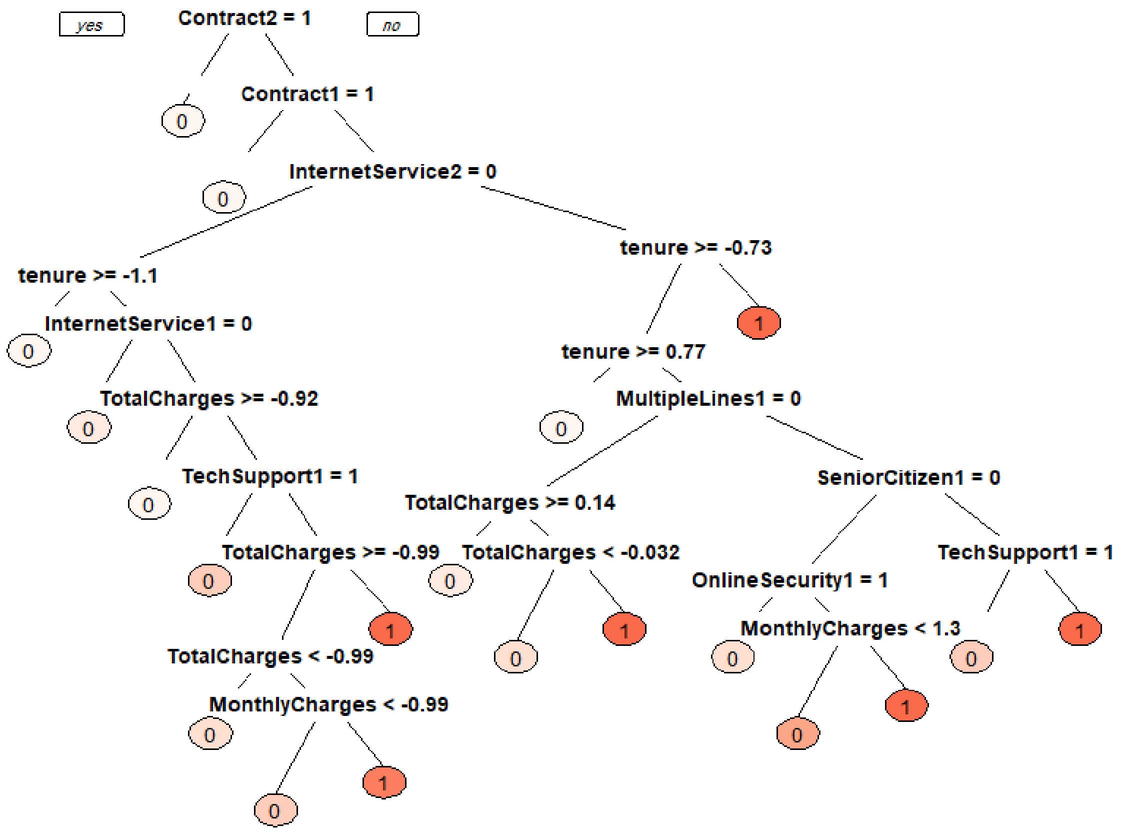
```
          Reference
Prediction   0   1
         0 854 163
         1 148 219

               Accuracy : 0.7753
                 95% CI : (0.7524, 0.797)
    No Information Rate : 0.724
    P-Value [Acc > NIR] : 7.647e-06

                  Kappa : 0.4308

 Mcnemar's Test P-Value : 0.4273

            Sensitivity : 0.8523
            Specificity : 0.5733
         Pos Pred Value : 0.8397
         Neg Pred Value : 0.5967
              Precision : 0.8397
                 Recall : 0.8523
                     F1 : 0.8460
             Prevalence : 0.7240
         Detection Rate : 0.6171
   Detection Prevalence : 0.7348
      Balanced Accuracy : 0.7128
```

**Rpart** **cTree**

4. **Q4:** I have tried splitting methods **Gini** and **information gain**, both with applying **cross validation technique with 10 folds.**

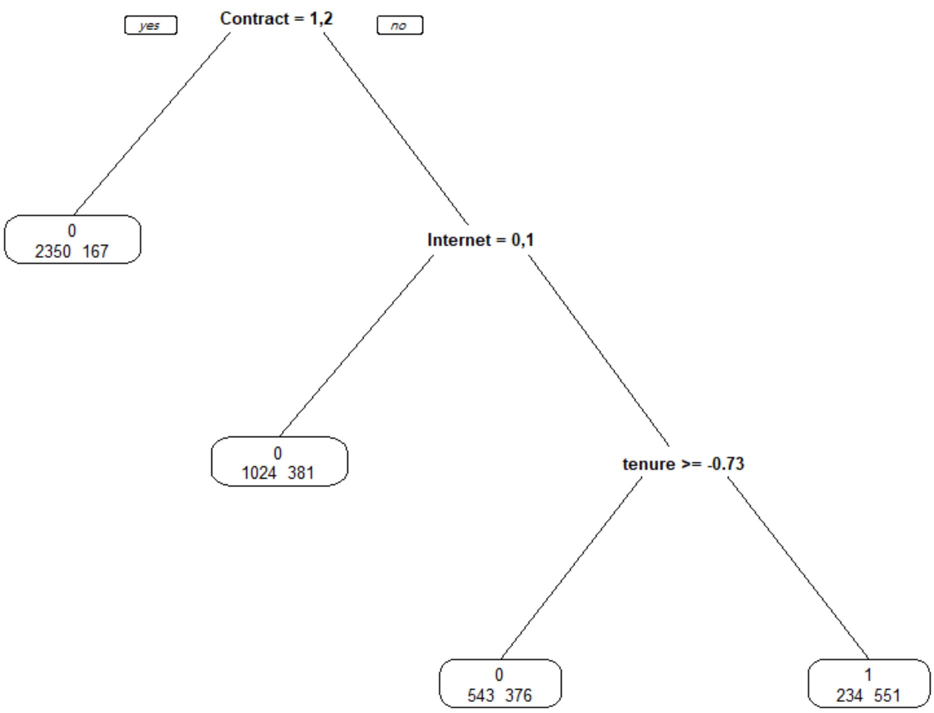



**Gini tree with cross validation.**




**Information gain tree with cross validation.**

And after that I have tried to prune the rpart tree.




**pruned tree.**

I have notice **that pruned tree is the same as rPart tree** and base on that it does not increase the accuracy.

But the **pruned tree** has a **higher accuracy** the the tree that made with **cTree library.**

```
          Reference
Prediction   0   1
         0 896 199
         1 106 183

               Accuracy : 0.7796
                 95% CI : (0.7568, 0.8012)
    No Information Rate : 0.724
    P-Value [Acc > NIR] : 1.287e-06

                  Kappa : 0.4037

 Mcnemar's Test P-Value : 1.380e-07

            Sensitivity : 0.8942
            Specificity : 0.4791
         Pos Pred Value : 0.8183
         Neg Pred Value : 0.6332
              Precision : 0.8183
                 Recall : 0.8942
                     F1 : 0.8546
             Prevalence : 0.7240
         Detection Rate : 0.6474
   Detection Prevalence : 0.7912
      Balanced Accuracy : 0.6866
```

```
          Reference
Prediction   0   1
         0 890 195
         1 112 187

               Accuracy : 0.7782
                 95% CI : (0.7554, 0.7998)
    No Information Rate : 0.724
    P-Value [Acc > NIR] : 2.369e-06

                  Kappa : 0.405

 Mcnemar's Test P-Value : 2.869e-06

            Sensitivity : 0.8882
            Specificity : 0.4895
         Pos Pred Value : 0.8203
         Neg Pred Value : 0.6254
              Precision : 0.8203
                 Recall : 0.8882
                     F1 : 0.8529
             Prevalence : 0.7240
         Detection Rate : 0.6431
   Detection Prevalence : 0.7840
      Balanced Accuracy : 0.6889
```

**Gini accuracy.** **information gain.**

```
          Reference
Prediction   0   1
         0 936 247
         1  66 135

               Accuracy : 0.7738
                 95% CI : (0.7509, 0.7956)
    No Information Rate : 0.724
    P-Value [Acc > NIR] : 1.341e-05

                  Kappa : 0.3369

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

            Sensitivity : 0.9341
            Specificity : 0.3534
         Pos Pred Value : 0.7912
         Neg Pred Value : 0.6716
              Precision : 0.7912
                 Recall : 0.9341
                     F1 : 0.8568
             Prevalence : 0.7240
         Detection Rate : 0.6763
   Detection Prevalence : 0.8548
      Balanced Accuracy : 0.6438
```

**Pruned tree.**

5. **Q5:** I have trained XG_Boost Model with nrounds = 70 and maxdepth = 3.

I think it's performing good kind of and there is no overfitting, **but the Sensitivity is high it's equal to 0.88.**

```
          Reference
Prediction   0   1
         0 889 185
         1 113 197

               Accuracy : 0.7847
                 95% CI : (0.7621, 0.8061)
    No Information Rate : 0.724
    P-Value [Acc > NIR] : 1.337e-07

                  Kappa : 0.4279

 Mcnemar's Test P-Value : 3.907e-05

            Sensitivity : 0.8872
            Specificity : 0.5157
         Pos Pred Value : 0.8277
         Neg Pred Value : 0.6355
              Precision : 0.8277
                 Recall : 0.8872
                     F1 : 0.8565
             Prevalence : 0.7240
         Detection Rate : 0.6423
   Detection Prevalence : 0.7760
      Balanced Accuracy : 0.7015
```

**XG_Boost.**

6. **Q6:** I have trained 3 deep learning models using keras, I have tried to change some values to get a better results.

**Model 1: -**

```
DNN_model1 <- keras_model_sequential()

DNN_model1 %>%
  layer_dense(units = 50, input_shape = 19) %>%
  layer_dropout(rate=0.7)%>%
  layer_activation(activation = 'relu') %>%

  layer_dense(units = 30) %>%
  layer_dropout(rate=0.5)%>%
  layer_activation(activation = 'relu') %>%

  layer_dense(units = 1) %>%
  layer_activation(activation = 'sigmoid')
```

```
          Reference
Prediction   0   1
         0 894 188
         1 108 194

               Accuracy : 0.7861
                 95% CI : (0.7636, 0.8075)
    No Information Rate : 0.724
    P-Value [Acc > NIR] : 6.747e-08

                  Kappa : 0.4278

 Mcnemar's Test P-Value : 4.395e-06

            Sensitivity : 0.8922
            Specificity : 0.5079
         Pos Pred Value : 0.8262
         Neg Pred Value : 0.6424
              Precision : 0.8262
                 Recall : 0.8922
                     F1 : 0.8580
             Prevalence : 0.7240
         Detection Rate : 0.6460
   Detection Prevalence : 0.7818
      Balanced Accuracy : 0.7000
```

## Model 2: -

```
DNN_model2 <- keras_model_sequential()

DNN_model2 %>%
  layer_dense(units = 600, input_shape = 19) %>%
  layer_dropout(rate=0.5)%>%
  layer_activation(activation = 'relu') %>%

  layer_dense(units = 300) %>%
  layer_dropout(rate=0.4)%>%
  layer_activation(activation = 'relu') %>%

  layer_dense(units = 1) %>%
  layer_activation(activation = 'sigmoid')
```

```
          Reference
Prediction   0   1
         0 898 188
         1 104 194

               Accuracy : 0.789
                 95% CI : (0.7666, 0.8102)
    No Information Rate : 0.724
    P-Value [Acc > NIR] : 1.633e-08

                  Kappa : 0.4336

 Mcnemar's Test P-Value : 1.191e-06

            Sensitivity : 0.8962
            Specificity : 0.5079
         Pos Pred Value : 0.8269
         Neg Pred Value : 0.6510
              Precision : 0.8269
                 Recall : 0.8962
                     F1 : 0.8602
             Prevalence : 0.7240
         Detection Rate : 0.6488
   Detection Prevalence : 0.7847
      Balanced Accuracy : 0.7020
```

## Model 3: -

```
DNN_model3 <- keras_model_sequential()

DNN_model3 %>%
  layer_dense(units = 768, input_shape = 19) %>%
  layer_dropout(rate=0.3)%>%
  layer_activation(activation = 'tanh') %>%

  layer_dense(units = 640) %>%
  layer_dropout(rate=0.3)%>%
  layer_activation(activation = 'tanh') %>%

  layer_dense(units = 1) %>%
  layer_activation(activation = 'sigmoid')
```

```
          Reference
Prediction   0   1
         0 856 177
         1 146 205

               Accuracy : 0.7666
                 95% CI : (0.7434, 0.7887)
    No Information Rate : 0.724
    P-Value [Acc > NIR] : 0.000176

                  Kappa : 0.401

 Mcnemar's Test P-Value : 0.095069

            Sensitivity : 0.8543
            Specificity : 0.5366
         Pos Pred Value : 0.8287
         Neg Pred Value : 0.5840
              Precision : 0.8287
                 Recall : 0.8543
                     F1 : 0.8413
             Prevalence : 0.7240
         Detection Rate : 0.6185
   Detection Prevalence : 0.7464
      Balanced Accuracy : 0.6955
```

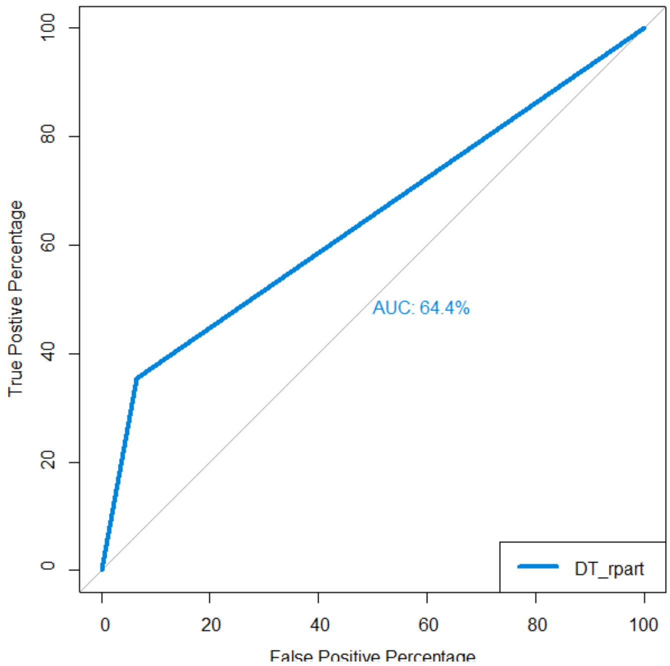
As we have seen **model 2** which has **(drop rate = 0.5 and 0.4)** and **(activation function = “relu”)** and **(input_shape = 19 and units = 600 and 640)** and **(activation function = “sigmoid” on the output layer)** has the highest **accuracy** which is = **0.789**

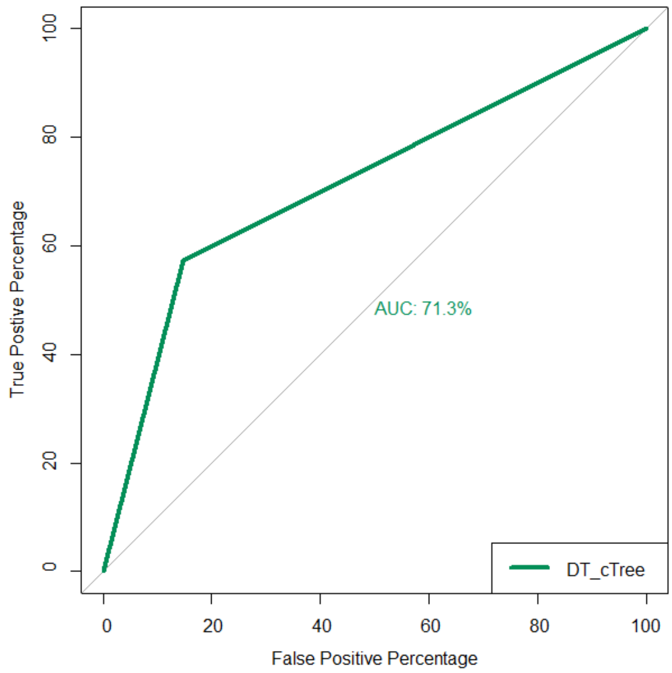
7. **Q7:** I'll compare these criteria precision, recall, accuracy, F1,

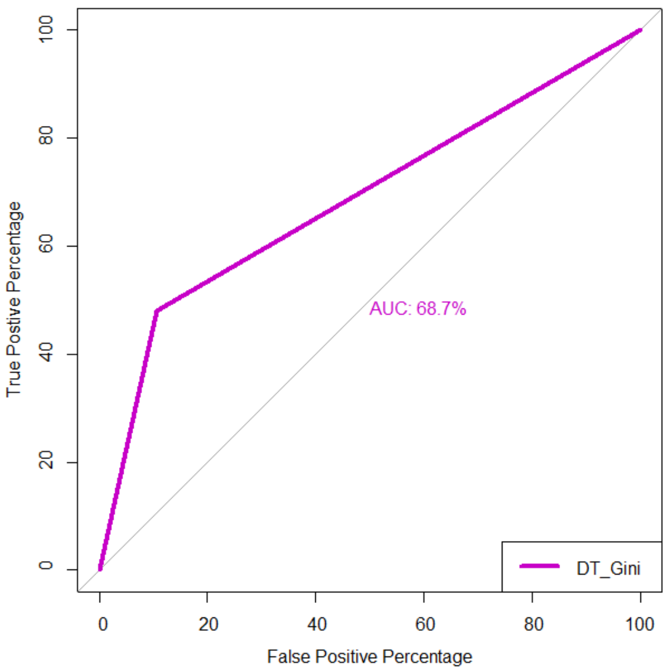
For the **best model in each algorithm.**

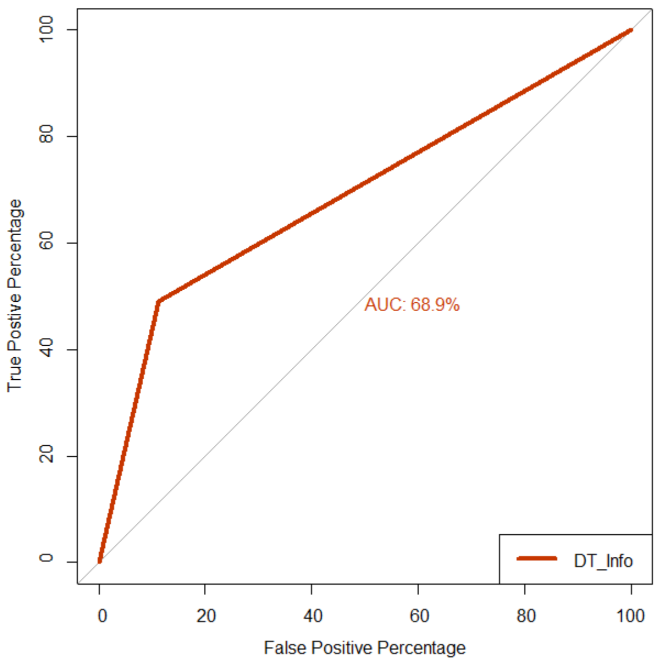
| | Decision tree (GINI) | XG_Boost | DNN (model 2) |
|---|---|---|---|
| **precision** | 0.8183 | 0.8277 | 0.8269 |
| **recall** | 0.8942 | 0.8872 | 0.8962 |
| **accuracy** | 0.7796 | 0.7847 | 0.789 |
| **F1** | 0.8546 | 0.8565 | 0.8602 |

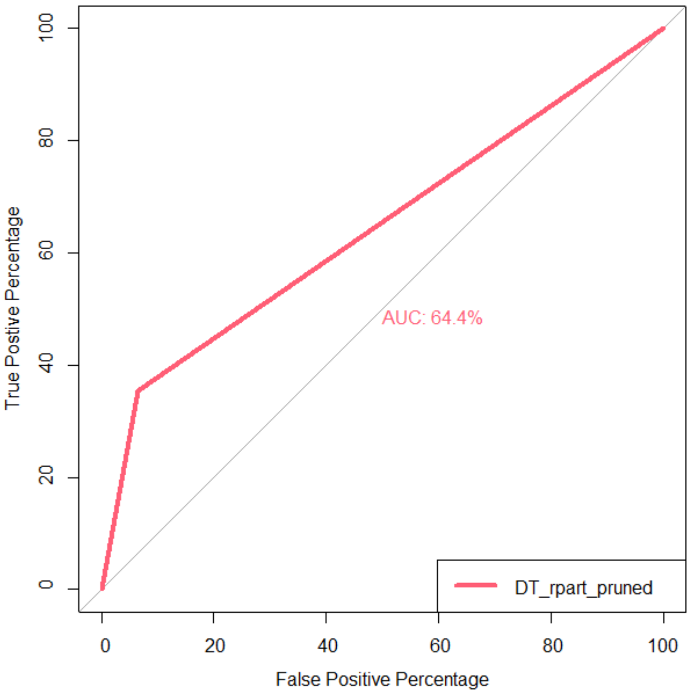
8. **Q8:** I have found that the model that has the highest AUC is **Decistion tree cTree model.**

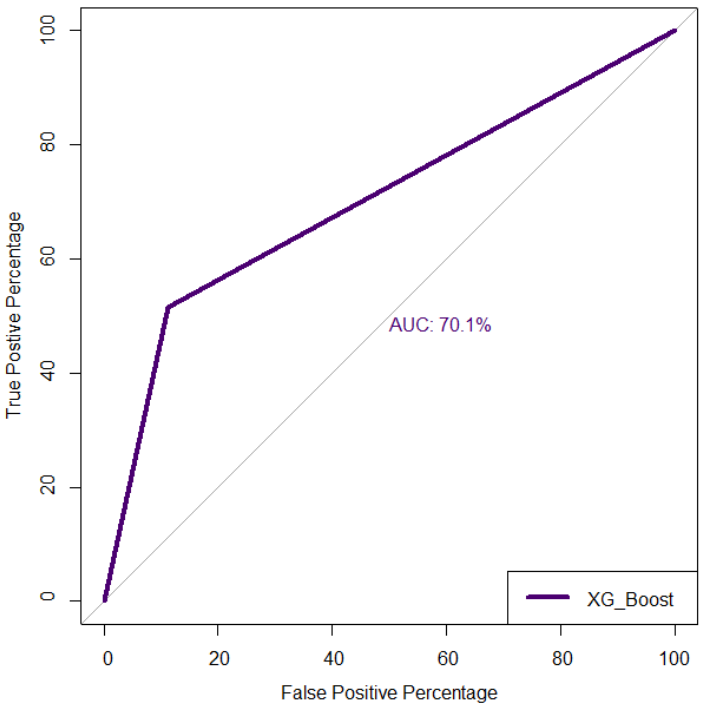


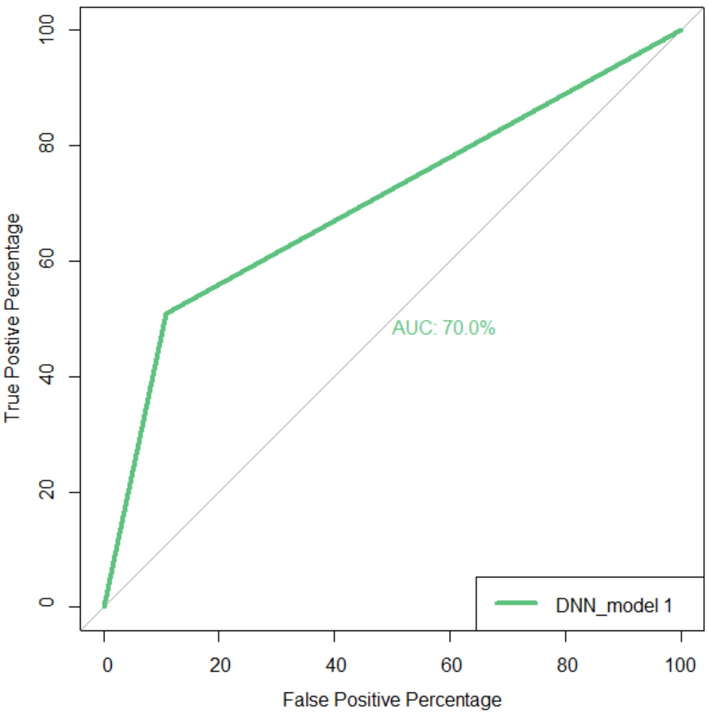



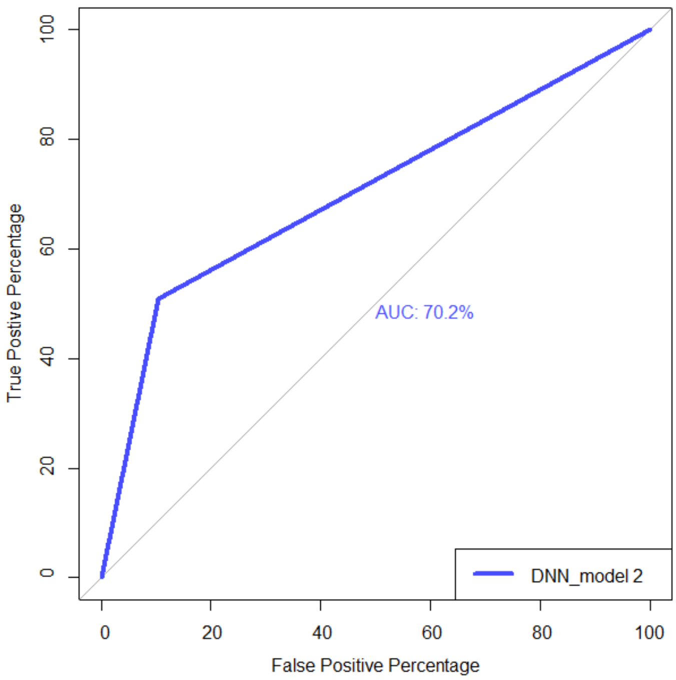



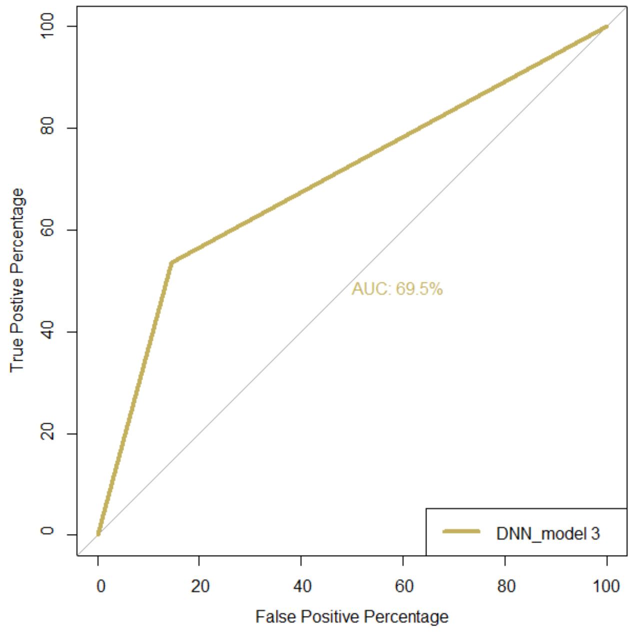




True Postive Percentage
False Positive Percentage
AUC: 64.4%
DT_rpart_pruned


True Postive Percentage
False Positive Percentage
AUC: 70.1%
XG_Boost

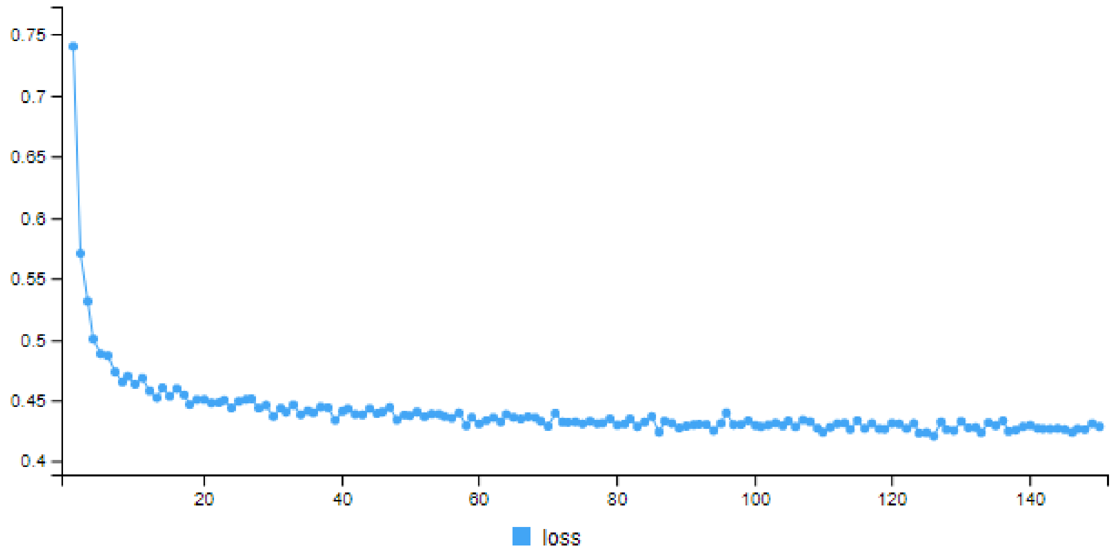

True Postive Percentage
False Positive Percentage
AUC: 70.0%
DNN_model 1

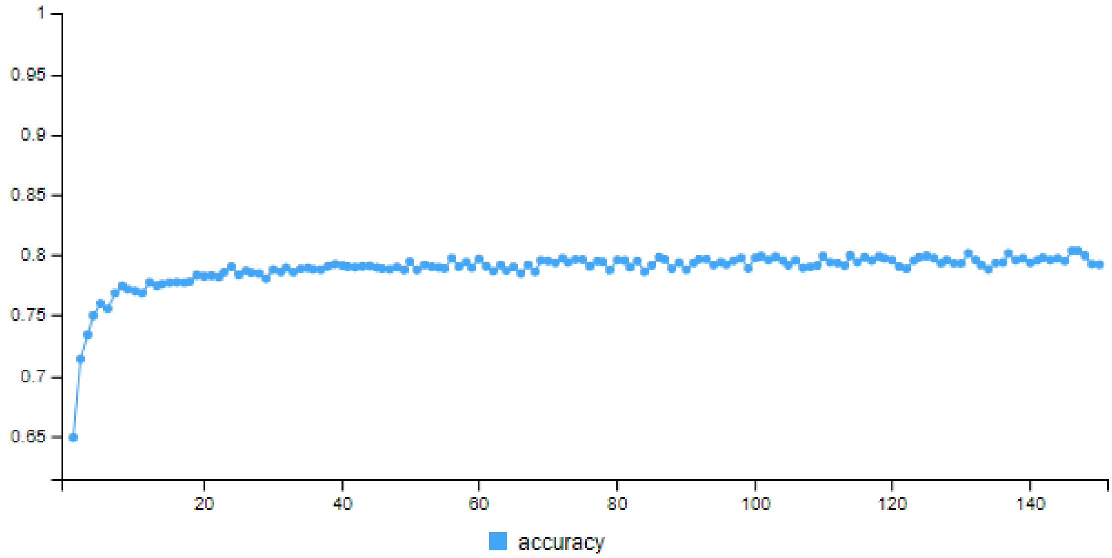

True Postive Percentage
False Positive Percentage
AUC: 70.2%
DNN_model 2



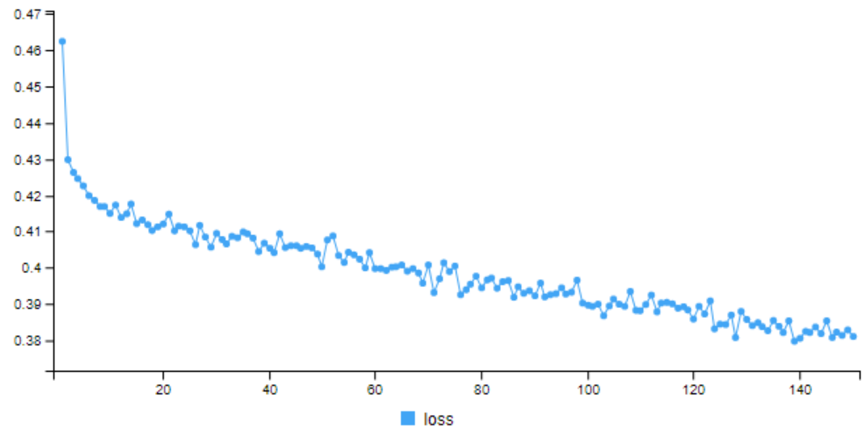

**The next plots are for each DNN model.**

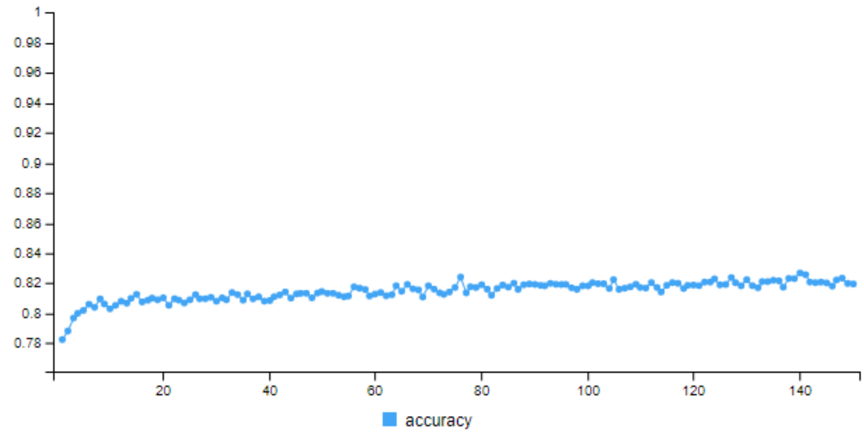
## Model 1:



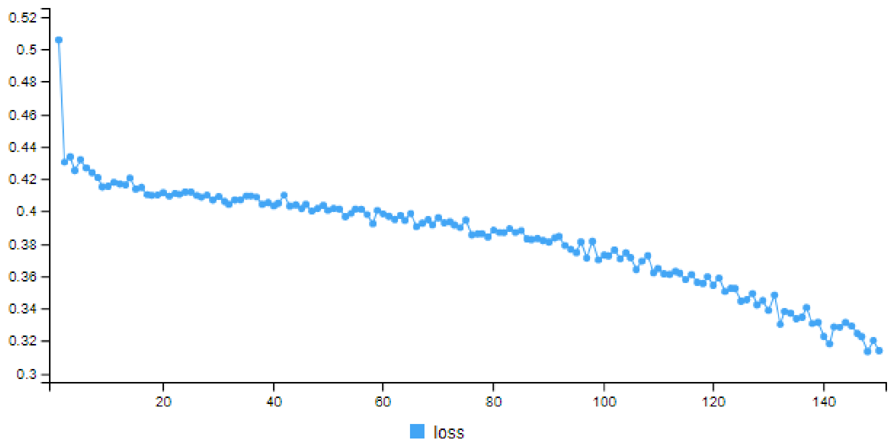



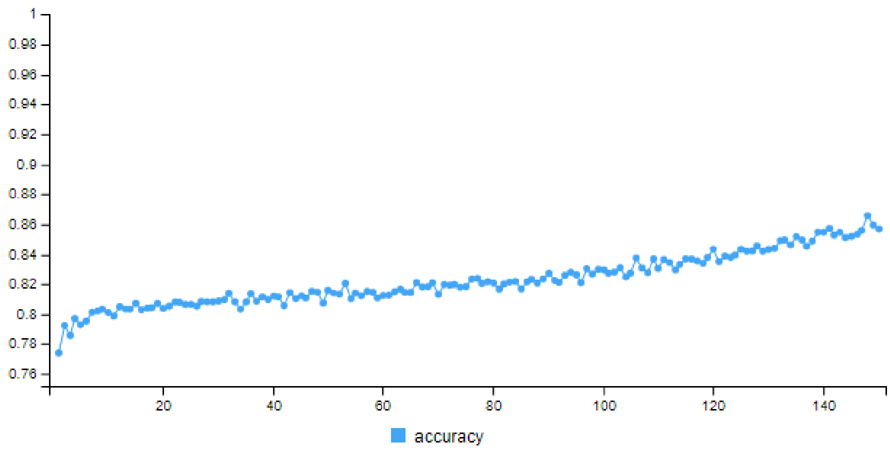
## Model 2:




## Model 3:

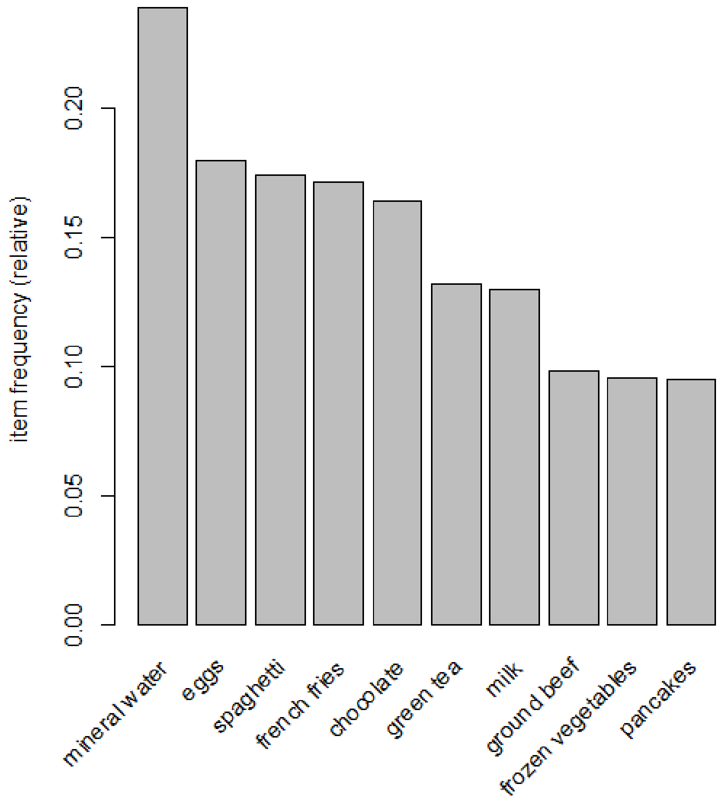


## ii. Part B

```r
# Plot of top 10 transactions
itemFrequencyPlot(Transactions_Data ,topN = 10)

# Create the model
model = apriori(data = Transactions_Data , parameter = list(support = 0.002 ,confidence = 0.20, maxlen = 3))
model1 = apriori(data = Transactions_Data , parameter = list(support = 0.002 ,confidence = 0.20, maxlen = 2))

# Displaying rules sorted by descending lift value
inspect(sort(model, by = 'lift')[1:10])
inspect(sort(model1, by = 'lift')[1:10])

# Displaying rules sorted by descending support value
inspect(sort(model, by = 'support')[1:10])
inspect(sort(model1, by = 'support')[1:10])

# Displaying rules sorted by descending confidence value
inspect(sort(model, by = 'confidence')[1:10])
inspect(sort(model1, by = 'confidence')[1:10])
```

1. **Q1:** plot of the top 10 transactions.




2. **Q2:** rules, sorted by descending lift value.

| | lhs | | rhs | support | confidence | coverage | lift |
|---|---|---|---|---|---|---|---|
| [1] | {escalope, mushroom cream sauce} | => | {pasta} | 0.002532996 | 0.4418605 | 0.005732569 | 28.088096 |
| [2] | {escalope, pasta} | => | {mushroom cream sauce} | 0.002532996 | 0.4318182 | 0.005865885 | 22.650826 |
| [3] | {mushroom cream sauce, pasta} | => | {escalope} | 0.002532996 | 0.9500000 | 0.002666311 | 11.976387 |
| [4] | {parmesan cheese, tomatoes} | => | {frozen vegetables} | 0.002133049 | 0.6666667 | 0.003199573 | 6.993939 |
| [5] | {mineral water, whole wheat pasta} | => | {olive oil} | 0.003866151 | 0.4027778 | 0.009598720 | 6.115863 |
| [6] | {frozen vegetables, parmesan cheese} | => | {tomatoes} | 0.002133049 | 0.3902439 | 0.005465938 | 5.706081 |
| [7] | {burgers, herb & pepper} | => | {ground beef} | 0.002266364 | 0.5483871 | 0.004132782 | 5.581345 |
| [8] | {light cream, mineral water} | => | {chicken} | 0.002399680 | 0.3272727 | 0.007332356 | 5.455273 |
| [9] | {ground beef, shrimp} | => | {herb & pepper} | 0.002932942 | 0.2558140 | 0.011465138 | 5.172131 |
| [10] | {fromage blanc} | => | {honey} | 0.003332889 | 0.2450980 | 0.013598187 | 5.164271 |

## 3. **Q3:** comparison between the two rules.

the rule below has the highest lift.

```
    lhs                                  rhs       support     confidence coverage    lift
[1] {escalope, mushroom cream sauce}  => {pasta}   0.002532996 0.4418605  0.005732569 28.088096
```

the rule below has the highest support.

```
    lhs                        rhs        support     confidence coverage   lift     count
[1] {fromage blanc}         => {honey}    0.003332889 0.2450980  0.01359819 5.164271 25
```

If I were a market manager, I would choose the first rule because it has the **highest lift** and the **highest coverage.**