IE425
SPRING 2023

HOMEWORK 1
DECISION TREES & RANDOM FORESTS

FATMANUR YAMAN - 2019402204
MURAT TUTAR - 2020402264

Question 1

First of all, to do a comprehensive data analysis and accurate predictions investigating the data in detail is crucial. Understanding the attributes (categorical or numerical), deciding on the type of the problem (classification or regression), and the missing value analysis should be the first things that a good data analyst does. Thus, the data analysis was done before answering the questions in the assignment.

- The shape of the data is (4601,58).
- There are 57 features that describe the type of the e-mail.
- There is no null value in the dataset.

```
#Load the dataset.
data("spam", package = "kernlab")
#Let's look at to the first 3 rows of the data to overview and obtain some insights.
#For example, the last column of the dataset is the value what we are going to predict.
head(spam,3)
#The dimension of the data is also important to check there is any missing colmuns or rows.
#The shape of the data is (4601,58). There are 57 features that describe the type of the e-mail.
dim(spam)
#Describe the dataset in terms of the type of the data(factor or numerical) and the values.
str(spam)
#Let's look whether the rows contain any null values or not.
#If there are some null values, we should fill them by using some methods to use the row in the prediction.
colSums(is.na(spam))
#The name of the columns is also important to understand the problem ?nd internalize it.
colnames(spam)
```

a.

The dataset is splitted into train and test parts using a seed value of 425. First, we checked the proportion of the train and test set. Since the tarinset is %80 and the test set is %20 of the dataset, the proportion of the test and train dataset should be 4 as in the output that R code gave.

After that, we investigated whether the proportion of the classes remains the same in both sets. There are 2 classes in the dataset in terms of the email type: nonspam and spam. As can be seen from the code below, ***the proportion of classes remains the same in both sets as approximately 0.606 for nonspam class and 0.394 for spam class.***

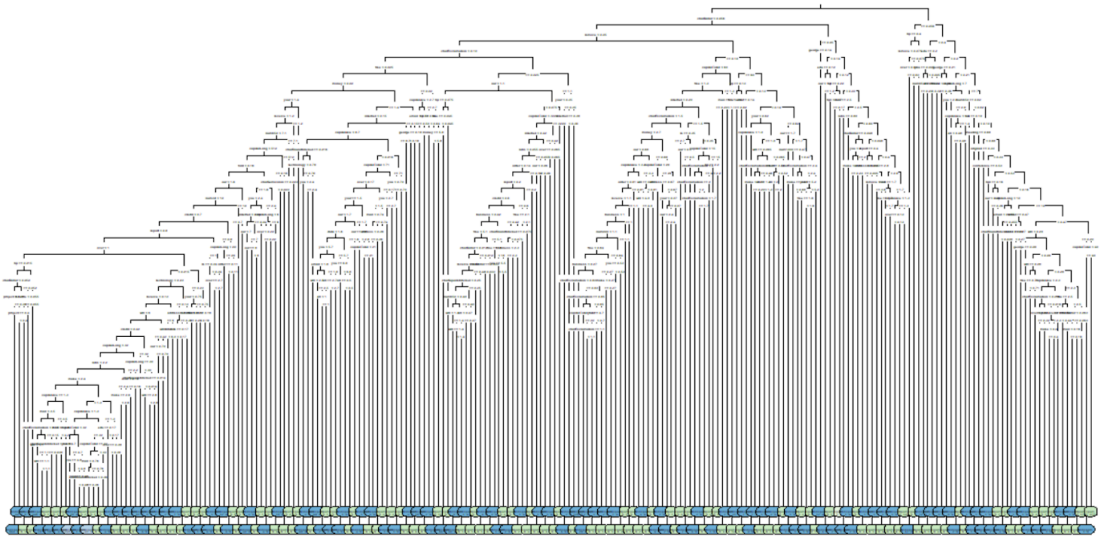
```
#QUESTION 1

#Seeding the set with value of 425.
set.seed(425)

#Train set is 0.8 whereas the test set is 0.2
split <- sample.split(spam$type, SplitRatio = 0.8)
train <- subset(spam, split == TRUE)
test <- subset(spam, split == FALSE)


#Checking whether the dataset is being splitted with the rates of 0.8 and 0.2
#The rate of number of observations of train and test datasets should be 4.
nrow(train)
nrow(test)
nrow(train)/nrow(test)
#According to the calculation, the rate is 4.

#The number of nonspam and spam emails in the datasets.
table(train$type)
table(test$type)
#Now, let's see the distribution of spam and nonspam emails in the datasets.
prop.table(table(train$type))
prop.table(table(test$type))
#As can be seen, the proportion of classes remains the same in both sets.
```

```
> #Now, let's see the distribution of spam and nonspam emails in the datasets.
> prop.table(table(train$type))

  nonspam      spam
0.6059783 0.3940217
> prop.table(table(test$type))

  nonspam      spam
0.6058632 0.3941368
```

b.

To obtain the largest possible tree, the parameters of the rpart function should be set to some certain values. Rpart function creates a decision tree and minsplit, minbucket, and cp are the main parameters that determine the size and the complexity of the tree. We set the minsplit to 2, the minbucket to 1, and the cp value to 0 to obtain the largest possible tree. When we plotted the largest possible decision tree, ***it has 232 leaf nodes.***

```
#QUESTION 2

#Using rpart and the tarining dataset, the largest possible tree is being determined by setting the
#minsplit and minbucket to zero.
#Minsplit is the minimum number of ob?ervations in the node to continue the splitting the node further.
#Minbucket is the minimum number of observations required at the leaf node.
#Also, the cp value is quite important in creating a large tree.
#The cp value controls the size of the decision t?ee. If cost of adding a new node is bigger than
#cp value, then the node is being added.
theLargestTree <- rpart(type~.,data=train, minsplit=2,minbucket=1, cp=0)
#Plot the tree by specifying the shape and color.
rpart.plot(theLargestTree, type = 3, box.palette = "auto", extra = 1)
#The number of leaf nodes in the largest possible tree is:
leaf_nodes <- sum(theLargestTree$frame$var == "<leaf>")
cat("The number of leaf nodes in the largest tree is:", leaf_nodes)
```



```
> leaf_nodes <- sum(theLargestTree$frame$var == "<leaf>")
> cat("The number of leaf nodes in the largest tree is:", leaf_nodes)
The number of leaf nodes in the largest tree is: 232
```

c.

We made predictions by using the largest possible decision tree and obtained the accuracy, precision, recall, false positive rate, false negative rate, and the error rate.

The Accuracy: 0.9087948

The Precision: 0.8885794

The False Positive Rate: 0.07168459

The False Negative Rate: 0.1212121

The Error Rate: 0.09120521

```
#QUESTION 3

#Predict by using test data set.
pred = predict(theLargestTree,newdata=test,type="class")
#Quick review of the predictions.
head(pred)
#The predicted proportion of the classes.
table(test$type, pred)
#Convert the categorical string values to binary vectors.
#1 should be extracted because the categorical values are being converted as 1 and 2.
test$type_numeric = as.numeric(test$type)-1
pred_numeric = as.numeric(pred)-1
#Checking whether the categorical string values are converted or not into to vectors.
table(test$type_numeric, pred_numeric)

#Import the neceassary library called Metrics to evaluate the performance of the predictor.
library(Metrics)
#accuracy = (tp+tn)/(tp+tn+fp+fn)
accuracy_1 = accuracy(actual=test$type, predicted=pred)
cat("The accuracy is:", accuracy_1)
#precision = (tp)/(tp+fp)
precision_1 = precision(actual=test$type_numeric,predicted=pred_numeric)
cat("The precision is:", precision_1)
#recall = (tp)/(tp+fn)
recall_1 = recall(actual =test$type_numeric, predicted=pred_numeric)
cat("The recall is:", recall_1)
#false negative = (fn)/(tp+fn)
false_negative_1 = 1-recall(actual =test$type_numeric, predicted=pred_numeric)
cat("The false negative rate is:", false_negative_1)
#false positive = (fp)/(tp+fp)
false_positive_1 = 1-precision(actual=test$type_numeric,predicted=pred_numeric)
cat("The false positive rate is:", false_positive_1)
#error = (fp+fn)/(tp+tn+fp+fn)
error_rate_1 = 1-accuracy(actual=test$type, predicted=pred)
cat("The error rate is:", error_rate_1)
```

```
> #accuracy = (tp+tn)/(tp+tn+fp+fn)
> accuracy_1 = accuracy(actual=test$type, predicted=pred)
> cat("The accuracy is:", accuracy_1)
The accuracy is: 0.9087948
> #precision = (tp)/(tp+fp)
> precision_1 = precision(actual=test$type_numeric,predicted=pred_numeric)
> cat("The precision is:", precision_1)
The precision is: 0.8885794
> #recall = (tp)/(tp+fn)
> recall_1 = recall(actual =test$type_numeric, predicted=pred_numeric)
> cat("The recall is:", recall_1)
The recall is: 0.8787879
> #false negative = (fn)/(tp+fn)
> false_negative_1 = 1-recall(actual =test$type_numeric, predicted=pred_numeric)
> cat("The false negative rate is:", false_negative_1)
The false negative rate is: 0.1212121
> #false positive = (fp)/(tp+fp)
> false_positive_1 = 1-precision(actual=test$type_numeric,predicted=pred_numeric)
> cat("The false positive rate is:", false_positive_1)
The false positive rate is: 0.1114206
> #error = (fp+fn)/(tp+tn+fp+fn)
> error_rate_1 = 1-accuracy(actual=test$type, predicted=pred)
> cat("The error rate is:", error_rate_1)
The error rate is: 0.09120521
```

```
> head(pred)
     2      4      6     12     13     18
 spam   spam   spam nonspam   spam   spam
```

d.

The train error is always going to decrease when the tree size is getting bigger, because the model memorizes the values in the train dataset and not learning the general pattern in the dataset. We call this issue overfitting. However, the test set error is decreasing at first, and then increases due to the overfitting. The optimal tree size is determined by the smallest cross-validation error. After finding the optimal tree size, the largest possible tree should be pruned by the optimal cp value. ***The optimum tree contains 64 leaf nodes.***

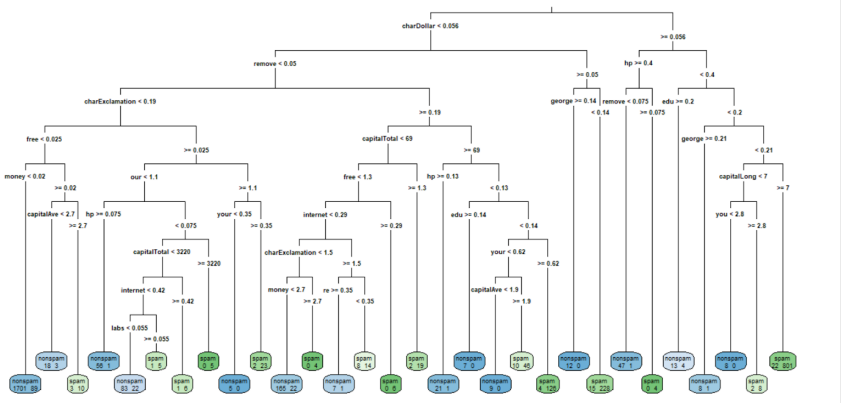
```r
#QUESTION 4

#First of all, print the cp table and see how the error decreases at first and incerases.
print(theLargestTree$cptable)
#Find the optimum index. After that point the cross valiadtion error is increasing in the test set while
#it is still decreasing in the train set.
#To prevent from overfitting, he tree should stop at this index point.
opt_index = which.min(unname(theLargestTree$cptable[, "xerror"]))
opt_index
cp_opt=theLargestTree$cptable[opt_index, "CP"]
#Prune the tree by using the minimum cross validation error.
tree_opt=prune.rpart(tree = theLargestTree,cp = cp_opt)
rpart.plot(tree_opt, type = 3, box.palette = "auto", extra = 1)
# The number of leaf nodes:
leaf_node_2 <- sum(tree_opt$frame$var == "<leaf>")
cat("The number of leaf nodes in the largest tree is:", leaf_node_2)
```

```
> cat("The number of leaf nodes in the largest tree is:", leaf_node_2)
The number of leaf nodes in the largest tree is: 64
```

The question also asks for another cross-validation error value which is the smallest cv error plus the standard deviation of the smallest cv error. We computed the new cross validation error value and pruned the tree according to this value.***The new cv error is 0.21748088.*** Then , we called this tree “opttree”. ***The opttree is given below.*** The number of leaves is ***30.***

```r
#Now, it is time to compute other error value and new index as it is requested in the question.

#The standard deviation of the error at the optimal index point.
cat("The standard deviation is:", theLargestTree$cptable[opt_index, "xstd"])
#The minimum value of the error.
cat("The minimum cv error is:", min(theLargestTree$cptable[, "xerror"]))
#The sum of the smallest cv error + one standard deviation.
cv_error=min(theLargestTree$cptable[, "xerror"])+theLargestTree$cptable[opt_index, "xstd"]
cat("The new cv error is:", cv_error)
#Find the index where the cv error is smaller than cv_error.
opttree_index= which.max(theLargestTree$cptable[, "xerror"]<=cv_error)
opttree_index
#New cp optimum value:
cp_opt_new=theLargestTree$cptable[opttree_index, "CP"]

#Prune the tree by using new cp value.
opttree=prune.rpart(theLargestTree,cp=cp_opt_new)
rpart.plot(opttree, type = 3, box.palette = "auto", extra = 1)

leaf_node_3 <- sum(opttree$frame$var == "<leaf>")
cat("The number of leaf nodes in the opttree is:", leaf_node_3)
```

```
> leaf_node_3 <- sum(opttree$frame$var == "<leaf>")
> cat("The number of leaf nodes in the opttree is:", leaf_node_3)
The number of leaf nodes in the opttree is: 30
```



e.

We made a prediction based on the pruned optimal tree and calculated the accuracy, error rate, false positive rate, false negative rate, and precision.

The Accuracy: 0.9218241

The Precision: 0.9343284

The False Positive Rate: 0.06567164

The False Negative Rate: 0.137741

The Error Rate: 0.0781759

```
#QUESTION 5


#Predict by using test data set.
pred_opttree=predict(opttree,newdata=test,type="class")
#Quick review of the predictions.
head(pred_opttree)
#The predicted proportion of the classes.
table(test$type, pred_opttree)
#Convert the categorical string values to binary vectors.
#1 should be extracted because the categorical values are being converted as 1 and 2.
test$type_numeric = as.numeric(test$type)-1
pred_opttree_numeric = as.numeric(pred_opttree)-1
#Checking whether the categorical string values are converted or not into to vectors.
table(test$type_numeric, pred_opttree_numeric)

#Import the neceassary library called Metrics to evaluate the performance of the predictor.
library(Metrics)
#accuracy = (tp+tn)/(tp+tn+fp+fn)
accuracy_2 = accuracy(actual=test$type, predicted=pred_opttree)
cat("The accuracy is:", accuracy_2)
#precision = (tp)/(tp+fp)
precision_2 = precision(actual=test$type_numeric,predicted=pred_opttree_numeric)
cat("The precision is:", precision_2)
#recall = (tp)/(tp+fn)
recall_2 = recall(actual =test$type_numeric, predicted=pred_opttree_numeric)
cat("The recall is:", recall_2)
#false negative = (fn)/(tp+fn)
false_negative_2 = 1-recall(actual =test$type_numeric, predicted=pred_opttree_numeric)
cat("The false negative rate is:", false_negative_2)
#false positive = (fp)/(tp+fp)
false_positive_2 = 1-precision(actual=test$type_numeric,predicted=pred_opttree_numeric)
cat("The false positive rate is:", false_positive_2)
#error = (fp+fn)/(tp+tn+fp+fn)
error_rate_2 = 1-accuracy(actual=test$type, predicted=pred_opttree)
cat("The error rate is:", error_rate_2)
```

```
> #accuracy = (tp+tn)/(tp+tn+fp+fn)
> accuracy_2 = accuracy(actual=test$type, predicted=pred_opttree)
> cat("The accuracy is:", accuracy_2)
The accuracy is: 0.9218241
> #precision = (tp)/(tp+fp)
> precision_2 = precision(actual=test$type_numeric,predicted=pred_opttree_numeric)
> cat("The precision is:", precision_2)
The precision is: 0.9343284
> #recall = (tp)/(tp+fn)
> recall_2 = recall(actual =test$type_numeric, predicted=pred_opttree_numeric)
> cat("The recall is:", recall_2)
The recall is: 0.862259
> #false negative = (fn)/(tp+fn)
> false_negative_2 = 1-recall(actual =test$type_numeric, predicted=pred_opttree_numeric)
> cat("The false negative rate is:", false_negative_2)
The false negative rate is: 0.137741
> #false positive = (fp)/(tp+fp)
> false_positive_2 = 1-precision(actual=test$type_numeric,predicted=pred_opttree_numeric)
> cat("The false positive rate is:", false_positive_2)
The false positive rate is: 0.06567164
> #error = (fp+fn)/(tp+tn+fp+fn)
> error_rate_2 = 1-accuracy(actual=test$type, predicted=pred_opttree)
> cat("The error rate is:", error_rate_2)
The error rate is: 0.0781759
```

Comparison of the results with c:

	The Largest Tree	The Optimum Tree
Accuracy	***0.9087948***	***0.9218241***
Precision	***0.8885794***	***0.9343284***

False Positive Rate	***0.07168459***	***0.06567164***
False Negative Rate	***0.1212121***	***0.137741***
Error Rate	***0.09120521***	***0.0781759***

As can be seen above, the optimum tree performs significantly better than the largest tree in terms of accuracy and precision. Although it gives a slightly higher rate of false negatives, it gives much less false positive outputs and the overall error rate is considerably lower than the largest tree. For us, this slight difference is not an indicator of a better model.

Question 2

The analysis of the data is given below. We will try to predict a numerical value by using 9 attributes in the Toyota dataset.

```
#Load the dataset.
data = read.csv("C:/Users/fatma/Desktop/IE425_HW1_FATMANUR_YAMAN/ToyotaCorolla.csv")
#Let's look at to the first 3 rows of the data to overview and obtain some insights.
#For example, the last column of the dataset is the value what we are going to predict.
head(data,3)
#The dimension of the data is also important to check there is any missing colmuns or rows.
dim(data)
#Describe the dataset in terms of the type of the data(factor or numerical) and the values.
str(data)
#Let's look whether the rows contain any null values or not.
#If there are some null values, we should fill them by using some methods to use the row in the prediction.
colSums(is.na(data))
```

```
3 13950  24 41711    Diesel 90        1         0 2000     3   1165
> #The dimension of the data is also important to check there is any missing colmuns or rows.
> dim(data)
[1] 1436   10
> #Describe the dataset in terms of the type of the data(factor or numerical) and the values.
> str(data)
'data.frame':   1436 obs. of  10 variables:
 $ Price    : int  13500 13750 13950 14950 13750 12950 16900 18600 21500 12950 ...
 $ Age      : int  23 23 24 26 30 32 27 30 27 23 ...
 $ KM       : int  46986 72937 41711 48000 38500 61000 94612 75889 19700 71138 ...
 $ FuelType : chr  "Diesel" "Diesel" "Diesel" "Diesel" ...
 $ HP       : int  90 90 90 90 90 90 90 90 192 69 ...
 $ MetColor : int  1 1 1 0 0 0 1 1 0 0 ...
 $ Automatic: int  0 0 0 0 0 0 0 0 0 0 ...
 $ CC       : int  2000 2000 2000 2000 2000 2000 2000 2000 1800 1900 ...
 $ Doors    : int  3 3 3 3 3 3 3 3 3 3 ...
 $ Weight   : int  1165 1165 1165 1165 1170 1170 1245 1245 1185 1105 ...
> #Let's look whether the rows contain any null values or not.
> #If there are some null values, we should fill them by using some methods to use the row in the prediction.
> colSums(is.na(data))
   Price      Age       KM  FuelType       HP  MetColor Automatic       CC
       0        0        0        0        0        0        0        0
   Doors    Weight
       0        0
```

a.

The dataset is splitted into train and test parts using a seed value of 582. First, we checked the proportion of the train and test set. Since the trainset is %75 and the test set is %25 of the dataset, the proportion of the test and train dataset should be 3 as in the output that R code gave.

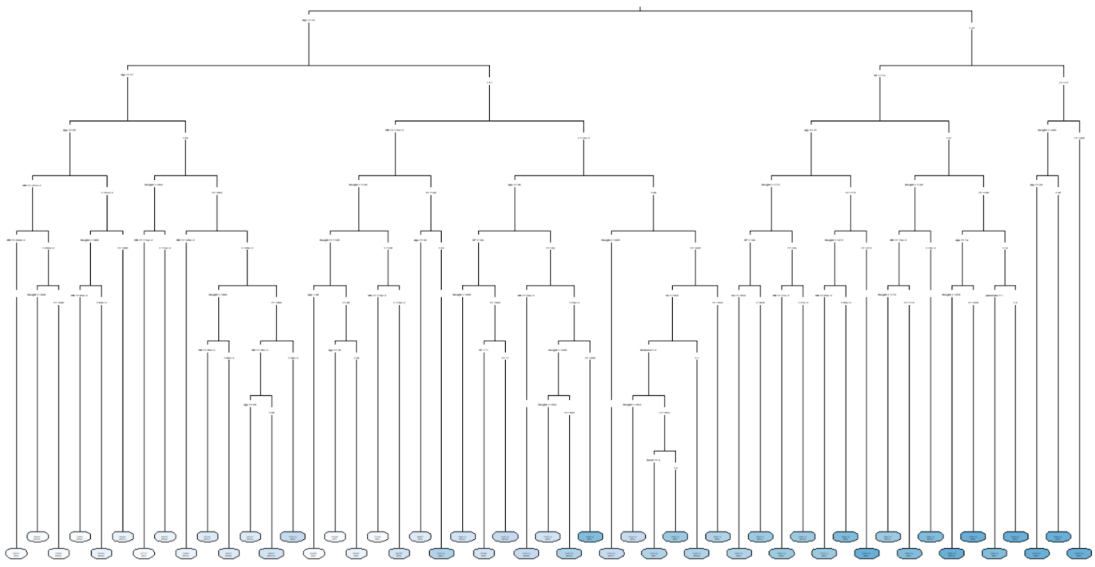
```r
#QUESTION 1

#Seeding the set with value of 425.
set.seed(582)

#Train-test-split by using index numbers.
inds <- createDataPartition(data$Price, p = 0.75, list=FALSE)
#Train set is 0.8 whereas the test set is 0.2
train <-data[inds, ]
test <- data[-inds, ]


#Checking whether the dataset is being splitted with the rates of 0.8 and 0.2
#The rate of number of observations of train and test datasets should be 4.
nrow(train)
nrow(test)
nrow(train)/nrow(test)
#According to the calculation, the rate is 3. (0.75/0.25)
```

```
> nrow(train)/nrow(test)
[1] 3.011173
```

b.

We found the optimum tree with the help of cv by pruning the largest possible tree. ***There are 51 leaf nodes in the tree with the smallest cross validation error.*** The importance of the attributes was calculated with the given code below. ***The most important attributes in decreasing order are Age, KM, Weight, and HP. Also all importances values are given below.***

```r
#QUESTION 2

#First, find the largest tree to prune it.
largestTree=rpart(Price~., data=train, minsplit=2, minbucket=1,cp=0)
rpart.plot(largestTree, type = 3, box.palette = "auto", extra = 1)
#Find the optimal index numebr where the cross-validation error starts to increase.
opt_index_2=which.min(largestTree$cptable[, "xerror"])
opt_index_2
#Find the corresponding cp value to that index.
cp_opt_2=largestTree$cptable[opt_index_2, "CP"]
#Prune the largest tree by using the cp value.
tree_opt_2=prune.rpart(tree=largestTree, cp=cp_opt_2)
rpart.plot(tree_opt_2, type = 3, box.palette = "auto", extra = 1)

#The number of leaf nodes in the optimal tree:
leaf_nodes_2 <- sum(tree_opt_2$frame$var == "<leaf>")
cat("The number of leaf nodes in the optimal tree is:", leaf_nodes_2)

# Determine the importance of the attributes
attribute_importance <- as.data.frame(tree_opt_2$variable.importance)
scaled_attribute_importance <- round(attribute_importance / max(attribute_importance), 5)
names(scaled_attribute_importance) <- "Relative Importance"

# Display the attribute importance with the most important one equal to 1
cat("Attributes importance with the most important one equal to 1:\n")
print(scaled_attribute_importance)
```



```
> print(scaled_attribute_importance)
           Relative Importance
Age                    1.00000
KM                     0.31725
Weight                 0.25403
HP                     0.14459
CC                     0.02246
Doors                  0.01255
FuelType               0.01192
MetColor               0.00282
Automatic              0.00158
```

c.

We made predictions by using the optimum decision tree and obtained the RMSE, MAE, and MAPE.

RMSE: 1188.601

MAE: 906.3326

MAPE: 0.09026224

```r
#QUESTION 3

#Predict by using test data set.
pred_2=predict(tree_opt_2, newdata=test)
#Quick review of the predictions.
head(pred_2)
test_price=test[,"Price"]
#Plot what is predicted and the actual values.
plot(pred_2, test_price)
abline(0,1)


library(Metrics)
rmse_1=rmse(actual = test_price,predicted = pred_2)
cat("The RMSE is:", rmse_1)
mae_1=mae(actual = test_price,predicted = pred_2)
cat("The MAE is:", mae_1)
mape_1=mape(actual = test_price,predicted = pred_2)
cat("The MAPE is:", mape_1)
```

```
> head(pred_2)
       1        3        6       12       17       31
13850.00 13850.00 13850.00 21428.57 21428.57 15387.86

> cat("The RMSE is:", rmse_1)
The RMSE is: 1188.601
> mae_1=mae(actual = test_price,predicted = pred_2)
> cat("The MAE is:", mae_1)
The MAE is: 906.3326
> mape_1=mape(actual = test_price,predicted = pred_2)
> cat("The MAPE is:", mape_1)
The MAPE is: 0.09026224
```

d.

We created sets of candidate parameters and by using for loops, we determined the best parameters by comparing the RMSE value of different random forest models. ***The best parameter combination that gives the smallest RMSE in the test set is mtry=6, ntree=100, nodesize=15.*** In addition to this, we calculated the RMSE, MAE, and MAPE metrics again to see the difference between the models with default parameters and best parameters.

```r
#QUESTION 4

library(randomForest)

#Creating the set of choices for the parameters.
mtry_tuning=seq(1,6,1)
ntree_tuning=seq(50,400,50)
nodesize_tuning=seq(15,150,10)

#The initial rmse is set to infinity.
best_rmse=Inf
#The for loops look for the best set of parameters by looking at the RMSE values.
for (i in mtry_tuning){
  for (j in ntree_tuning){
    for (k in nodesize_tuning){
      best_parameters= list()
      rf.parameters=randomForest(Price~., data=train,
                                 mtry=i, ntree=j, nodesize=k)
      predicted_tree=predict(rf.parameters, newdata=test)
      rmse_tree=rmse(actual=test_price, predicted=predicted_tree)
      if (rmse_tree<=best_rmse){
        best_rmse=rmse_tree
        new_parameters=append(best_parameters,c(i,j,k),)
      }
    }
  }
}

best_parameters_int=as.integer(new_parameters)
best_parameters_int
#Creating Random Forest by using the best parameters.
rf_toyota=randomForest(Price~., data=train,mtry=best_parameters_int[1], ntree=best_parameters_int[2], nodesize=best_parameters_int[3])
#Predicting the values by using test dataset.
rf_toyota_predict=predict(rf_toyota, newdata=test)

#The error metrics:
rmse_2=rmse(actual = test_price,predicted = rf_toyota_predict)
cat("The RMSE is:", rmse_2)
mae_2=mae(actual = test_price,predicted = rf_toyota_predict)
cat("The MAE is:", mae_2)
mape_2=mape(actual = test_price,predicted = rf_toyota_predict)
cat("The MAPE is:", mape_2)
```

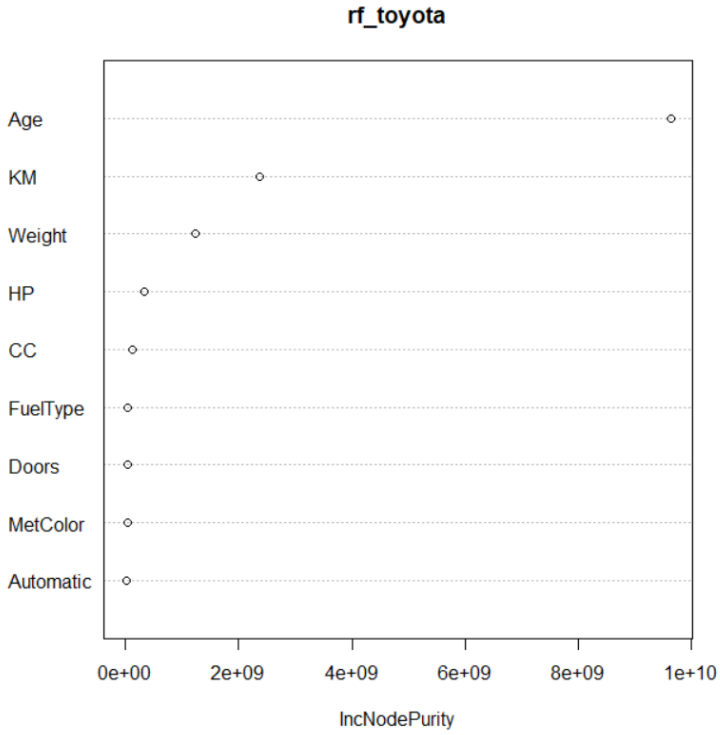
```
> best_parameters_int
[1]   6 100  15
```

e.

We calculated the importance of the attributes and plot them. ***In the decreasing order, the most important attributes are Age, KM, Weight, and HP.*** The importance of the values and the plot are below.

```r
#QUESTION 5

#The importance of the attributes in the random forest model:
round(importance(rf_toyota), 2)
#Plot the importance values.
varImpPlot(rf_toyota)
```

```
> round(importance(rf_toyota), 2)
           IncNodePurity
Age           9635217745
KM            2379229311
FuelType        48089275
HP             327480105
MetColor        35866984
Automatic       10870981
CC             118257061
Doors           40909398
Weight        1245674063
```



f.

We made predictions by using the optimum decision tree and obtained the RMSE, MAE, and MAPE.

RMSE: 1103.752

MAE: 835.9901

MAPE: 0.08240478

Comparison of the results with c:

	The Optimum Tree	The Random Forest
RMSE	**1188.601**	**1103.752**
MAE	**906.3326**	**835.9901**
MAPE	**0.09026224**	**0.08240478**

As can be seen above, the errors are smaller in the random forest compared to the optimum tree, as expected. Smaller RMSE indicates that the random forest model performs better than the optimum tree in terms of accuracy. Smaller MAE indicates that the random forest model makes less large errors than the optimum tree, showing that it makes more accurate predictions. Smaller MAPE similarly shows that the random forest tree makes less large errors with respect to percentage, showing that the random forest model performs more accurate.