IE425 Data Mining

Homework 1

05.04.2023

Tevfik Buğra Türker 2019402120

Ömercan Mısırlıoğlu 2020402261

1st Question

a) Using a seed value of 425, partition the dataset into training and test sets where 80% of goes into the training set and 20% goes into the test set. Make sure that the proportion of classes remains the same in both sets.

Code:

```
set.seed(425) #setting the seed value

split=sample.split(spam$type,SplitRatio=0.8)

summary(split)

spamtr=subset(spam,split==TRUE) #creating the training subset
spamte=subset(spam,split==FALSE) #creating the test subset

#Testing the proportion of the classes in the training and test sets
mean(as.numeric(spamtr$type)-1)
mean(as.numeric(spamte$type)-1)
```

Output:

```
Mode    FALSE    TRUE
logical     921    3680
```

0.394021739130435
0.39413680781759

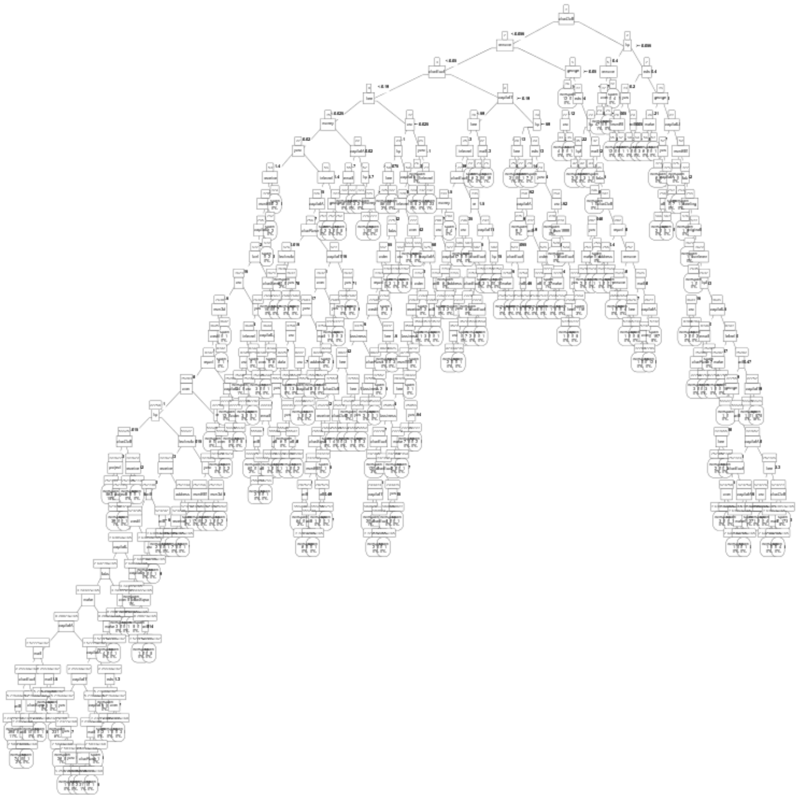
Comment: Proportion of the classes are extremely close, can be accepted equal. That small difference stems from the divisibleness of the data.

b) Using the rpart package and training set, determine the largest possible tree. How many leaf nodes do exist in the tree?

Code:

```
##largest tree

#creating the largest tree with rpart
largesttreespam=rpart(type~.,data=spamtr,minsplit=2,minbucket=1,cp=0)

numofleafnodes =
printcp(largesttreespam)[nrow(printcp(largesttreespam)),2]+1
cat("Number of leaf nodes in the biggest tree is:", numofleafnodes)

prp(largesttreespam,type=5,extra=101,nn=TRUE,tweak=1)
```

Comment: The largest tree and its CP table looks like this:



```
           CP nsplit rel error  xerror     xstd
1  0.49586207      0 1.0000000 1.00000 0.020443
2  0.13862069      1 0.5041379 0.50414 0.016692
3  0.04206897      2 0.3655172 0.36552 0.014689
4  0.02896552      4 0.2813793 0.29517 0.013412
5  0.01379310      5 0.2524138 0.26483 0.012790
6  0.01172414      6 0.2386207 0.26138 0.012716
7  0.00827586      7 0.2268966 0.25517 0.012581
8  0.00620690      8 0.2186207 0.24966 0.012459
9  0.00551724      9 0.2124138 0.24414 0.012336
10 0.00482759     11 0.2013793 0.23862 0.012210
11 0.00413793     13 0.1917241 0.23517 0.012131
12 0.00344828     14 0.1875862 0.22897 0.011986
13 0.00310345     15 0.1841379 0.23172 0.012051
14 0.00275862     17 0.1779310 0.22690 0.011937
15 0.00241379     20 0.1696552 0.22414 0.011871
16 0.00229885     22 0.1648276 0.22483 0.011888
17 0.00206897     29 0.1482759 0.21655 0.011688
18 0.00155172     37 0.1317241 0.21448 0.011637
19 0.00137931     45 0.1193103 0.19448 0.011129
20 0.00114943     60 0.0986207 0.19310 0.011092
21 0.00103448     63 0.0951724 0.19034 0.011019
22 0.00096552     65 0.0931034 0.19241 0.011074
23 0.00082759     70 0.0882759 0.19586 0.011165
24 0.00068966     75 0.0841379 0.19448 0.011129
25 0.00053640    128 0.0475862 0.19655 0.011183
26 0.00051724    138 0.0420690 0.19862 0.011237
27 0.00049261    142 0.0400000 0.20552 0.011413
28 0.00045977    161 0.0296552 0.20483 0.011396
29 0.00034483    182 0.0193103 0.20828 0.011483
30 0.00027586    226 0.0041379 0.22138 0.011805
31 0.00000000    231 0.0027586 0.22069 0.011788
Number of leaf nodes in the biggest tree is: 232
```

The biggest nsplit value = 231, which means we have 232 leaf nodes in the largest tree.

c) Make predictions in the test set and report the accuracy, error rate, false positive rate, false negative rate, and precision.

Code:

```
predspam=predict(largesttreespam,newdata=spamte,type="class") #predictions
table(spamte$type,predspam)
cat("Accuracy = ", accuracy(actual = spamte$type,predicted = predspam),
"\n") #accuracy
cat("Error rate = ", 1-accuracy(actual = spamte$type,predicted = predspam),
"\n") #error rate
cat("False Positive Rate =
",table(spamte$type,predspam)[2,1]/(table(spamte$type,predspam)[2,1]+table(
spamte$type,predspam)[2,2]), "\n") #false positive rate
cat("False Negative Rate =
",table(spamte$type,predspam)[1,2]/(table(spamte$type,predspam)[1,2]+table(
spamte$type,predspam)[1,1]), "\n") #false negative rate
cat("Precision =
",table(spamte$type,predspam)[1,1]/(table(spamte$type,predspam)[1,1]+table(
spamte$type,predspam)[2,1]), "\n")
```

Output:

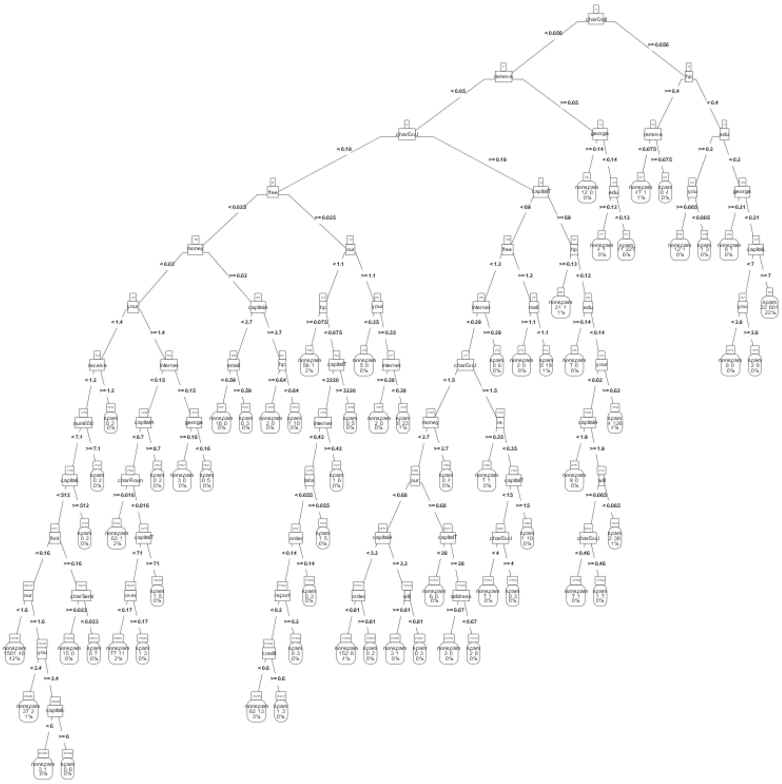
```
         predspam
          nonspam spam
  nonspam     518   40
  spam         44  319

Accuracy                 =  0.9087948
Error rate               =  0.09120521
False Positive Rate      =  0.1212121
False Negative Rate      =  0.07168459
Precision                =  0.9217082
```

d) What is the size of the tree in terms of the number of leaf nodes which makes the cross-validation (CV) error the smallest? Note that rpart function provides this automatically. What is the smallest the tree which has a CV error smaller than the smallest CV error plus one standard deviation of the error? Call this last tree "opttree".

Code (for smallest cv-error):

```
#smallest cv error

opt_index=which.min(largesttreespam$cptable[, "xerror"]) #finding the
optimum cp
opt_index=which.min(unname(largesttreespam$cptable[, "xerror"]))
cp_opt=largesttreespam$cptable[opt_index, "CP"]

treespam_opt=prune.rpart(tree = largesttreespam, cp = cp_opt) #pruning the
largest tree until optimum tree is formed

#printcp(treespam_opt)
prp(treespam_opt,type=5,extra=101,nn=TRUE,tweak=1)
print(treespam_opt$cptable)

numofleafnodes_opt =
printcp(treespam_opt)[nrow(printcp(treespam_opt)),2]+1 #printing the leaf
node number
cat("Number of leaf nodes in the biggest tree is:", numofleafnodes_opt)
#printing the leaf node number
```

Comment (for smallest cv-error): The smallest CV-error tree and its CP table looks like this:



	CP	nsplit	rel error	xerror	xstd
1	0.4958621	0	1.000000	1.00000	0.020443
2	0.1386207	1	0.504138	0.50414	0.016692
3	0.0420690	2	0.365517	0.36552	0.014689
4	0.0289655	4	0.281379	0.29517	0.013412
5	0.0137931	5	0.252414	0.26483	0.012790
6	0.0117241	6	0.238621	0.26138	0.012716
7	0.0082759	7	0.226897	0.25517	0.012581
8	0.0062069	8	0.218621	0.24966	0.012459
9	0.0055172	9	0.212414	0.24414	0.012336
10	0.0048276	11	0.201379	0.23862	0.012210
11	0.0041379	13	0.191724	0.23517	0.012131
12	0.0034483	14	0.187586	0.22897	0.011986
13	0.0031034	15	0.184138	0.23172	0.012051
14	0.0027586	17	0.177931	0.22690	0.011937
15	0.0024138	20	0.169655	0.22414	0.011871
16	0.0022989	22	0.164828	0.22483	0.011888
17	0.0020690	29	0.148276	0.21655	0.011688
18	0.0015517	37	0.131724	0.21448	0.011637
19	0.0013793	45	0.119310	0.19448	0.011129
20	0.0011494	60	0.098621	0.19310	0.011092
21	0.0010345	63	0.095172	0.19034	0.011019

The biggest nsplit value = 63, which means we have 64 leaf nodes in this tree.

Code (for opttree):

```
cat("Smallest CV error + 1 sd = ", 0.19034 + 0.011019) #xerror value of the
second tree
opttree=rpart(type~.,data=spamtr,cp=0.0013793) #cp giving the xerror =
0.201359
printcp(opttree)
prp(opttree,type=5,extra=101,nn=TRUE,tweak=1) #smallest tree (opttree)
```

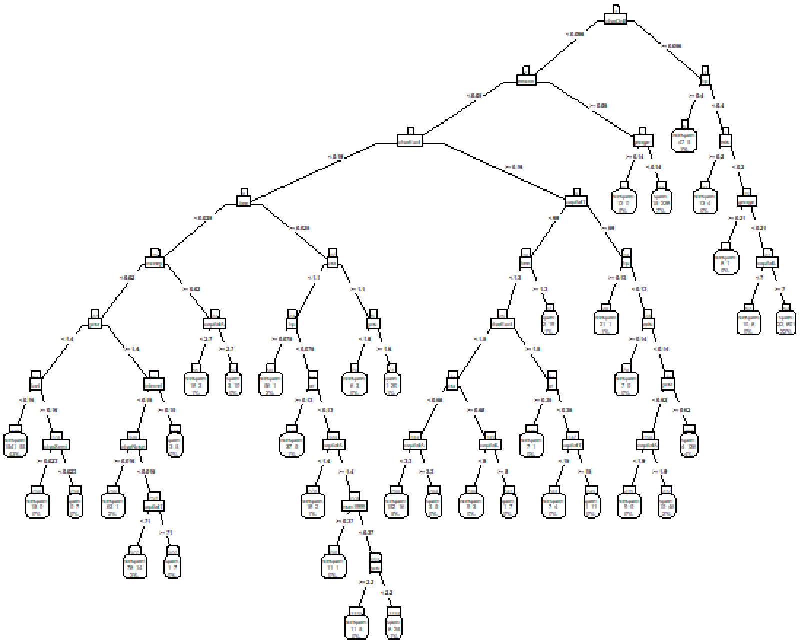
Comment (for opttree):

Smallest CV error + 1 sd = 0.19034 + 0.011019 = 0.201359

Then we find the corresponding cp value which equals 0.0013793.

We used this value to construct opttree.

The opttree and its cp-table is given below:



	CP	nsplit	rel error	xerror	xstd
1	0.4958621	0	1.00000	1.00000	0.020443
2	0.1386207	1	0.50414	0.50483	0.016700
3	0.0420690	2	0.36552	0.36621	0.014701
4	0.0289655	4	0.28138	0.32138	0.013913
5	0.0137931	5	0.25241	0.27586	0.013022
6	0.0117241	6	0.23862	0.26414	0.012775
7	0.0082759	7	0.22690	0.25931	0.012671
8	0.0062069	8	0.21862	0.25103	0.012490
9	0.0055172	9	0.21241	0.24759	0.012413
10	0.0048276	11	0.20138	0.24690	0.012398
11	0.0031034	13	0.19172	0.23517	0.012131
12	0.0024138	15	0.18552	0.23448	0.012115
13	0.0020690	19	0.17586	0.23103	0.012035
14	0.0018966	23	0.16759	0.23172	0.012051
15	0.0017241	28	0.15586	0.23103	0.012035
16	0.0013793	34	0.14552	0.23103	0.012035
17	0.0013793	36	0.14276	0.22621	0.011921

The biggest nsplit value = 36, which means we have 37 leaf nodes in this tree.

e) Make predictions on the test set with opttree and report the accuracy, error rate, false positive rate, false negative rate, and precision. Compare the result with part c)

Code:

```
predopttree=predict(opttree,newdata=spamte,type="class")
table(spamte$type,predopttree)
cat("Accuracy = ", accuracy(actual = spamte$type,predicted = predopttree),
"\n") #accuracy
cat("Error rate = ", 1-accuracy(actual = spamte$type,predicted =
predopttree), "\n") #error rate
cat("False Positive Rate =
",table(spamte$type,predopttree)[2,1]/(table(spamte$type,predopttree)[2,1]+
table(spamte$type,predopttree)[2,2]), "\n") #false positive rate
cat("False Negative Rate =
",table(spamte$type,predopttree)[1,2]/(table(spamte$type,predopttree)[1,2]+
table(spamte$type,predopttree)[1,1]), "\n") #false negative rate
cat("Precision =
",table(spamte$type,predopttree)[1,1]/(table(spamte$type,predopttree)[1,1]+
table(spamte$type,predopttree)[2,1]), "\n")
```

Output:

OPTTREE

Accuracy = 0.9196526

Error rate = 0.08034745

False Positive Rate = 0.1212121

False Negative Rate = 0.05376344

Precision = 0.9230769

LARGESTTREE

Accuracy = 0.9087948

Error rate = 0.09120521

False Positive Rate = 0.1212121

False Negative Rate = 0.07168459

Precision = 0.9217082

Comment:

The results indicate that OPTTREE has a higher accuracy (0.920) than LARGESTTREE (0.909), meaning that OPTTREE is better at correctly identifying the target class. OPTTREE also has a lower error rate (0.080) than LARGESTTREE (0.091), meaning that it makes fewer mistakes overall.

Looking at the false positive and false negative rates, OPTTREE has a lower false negative rate (0.054) compared to LARGESTTREE (0.072), indicating that OPTTREE is better at identifying true positives. Also, OPTTREE and rate LARGESTTREE have the same false positive (0.121), indicating that they are equally likely to misclassify negative instances as positive.

In terms of precision, OPTTREE (0.923) has a slightly higher precision than LARGESTTREE (0.922), indicating that OPTTREE is better at avoiding false positives.

Overall, results suggest that OPTTREE performs better than LARGESTTREE in terms of nearly all the error measurements. The largest tree may be overfitting the data the according to the results.

2nd Question

a) Partition the data set into training and test sets with 75% going into the training set by using a seed value of 582.

Code:

```
toy=read.csv("toyotacorolla.csv")
set.seed(582)
train=sample(1:1436,1077)
toytr=toy[train,]
toyte=toy[-train,]
```

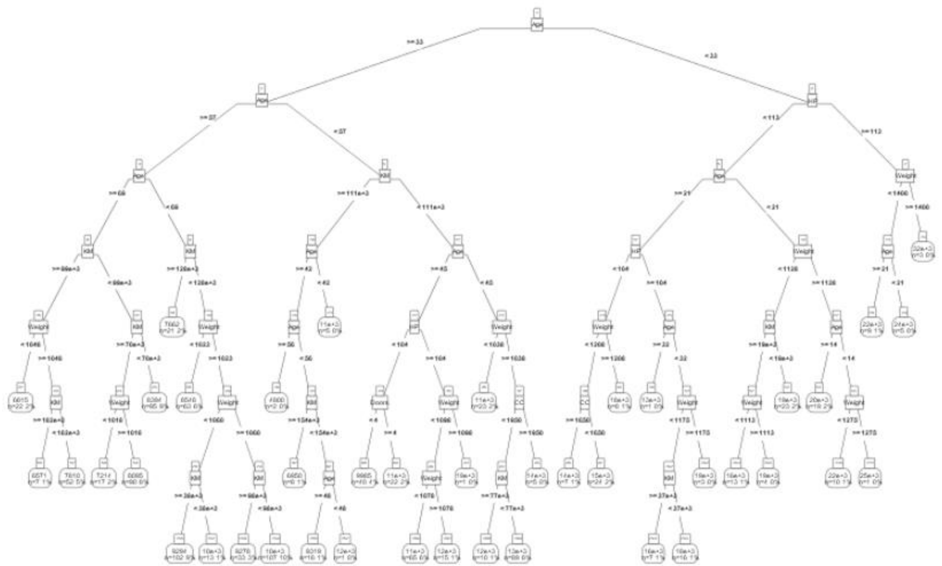
b) Using the rpart package and training set, determine the tree which gives the smallest cross- validation error? How many leaf nodes do exist in this tree? Which attributes are the most important?

Code:

```
largesttreetoy=rpart(Price~.,data=toytr,minsplit=2, minbucket=1, cp=0)
opt_index_toy=which.min(largesttreetoy$cptable[, "xerror"])
opt_index_toy=which.min(unname(largesttreetoy$cptable[, "xerror"]))
cp_opt_toy=largesttreetoy$cptable[opt_index_toy, "CP"]
tree_opt_toy = prune.rpart(tree=largesttreetoy, cp=cp_opt_toy)
prp(tree_opt_toy,type=5,extra=101,nn=TRUE,tweak=1)
numofleafnodes_toy =
printcp(tree_opt_toy)[nrow(printcp(tree_opt_toy)),2]+1 #printing the leaf
node number
cat("numofleafnodes_toy is = ", numofleafnodes_toy)
summary(tree_opt_toy)
```

Output:

	CP	nsplit	rel error	xerror	xstd
1	0.6759512301	0	1.00000000	1.00025378	0.072972296
2	0.0907965946	1	0.32404877	0.33581719	0.025769973
3	0.0393592317	2	0.23325218	0.24462284	0.023999226
4	0.0200853441	3	0.19389294	0.21654335	0.015487691
5	0.0194432092	4	0.17380760	0.21132411	0.015493225
6	0.0142473779	5	0.15436439	0.18838825	0.014991255
7	0.0133270332	6	0.14011701	0.17915066	0.014915419
8	0.0104109837	7	0.12678998	0.17705571	0.014743436
9	0.0101958897	8	0.11637900	0.16664144	0.013398543
10	0.0043390104	9	0.10618311	0.13791807	0.011252690
11	0.0032711823	11	0.09750509	0.12408338	0.010522254
12	0.0032200032	13	0.09096272	0.11866224	0.010363010
13	0.0025081484	15	0.08452271	0.11597563	0.010242809
14	0.0020645229	16	0.08201457	0.11528945	0.010027250
15	0.0019994709	18	0.07788552	0.11378814	0.010034013
16	0.0018750271	19	0.07588605	0.11321852	0.009985111
17	0.0016877597	20	0.07401102	0.11145684	0.009907127
18	0.0015284407	21	0.07232326	0.10868794	0.009883281
19	0.0012729875	22	0.07079482	0.10723006	0.009845725
20	0.0012622057	23	0.06952183	0.10615556	0.009820671
21	0.0010996488	25	0.06699742	0.10599525	0.009816798
22	0.0010953054	26	0.06589777	0.10232783	0.009663944
23	0.0009311227	27	0.06480247	0.10106385	0.009667224
24	0.0008557797	28	0.06387135	0.09933089	0.009140263
25	0.0007895617	29	0.06301557	0.09775146	0.009116069
26	0.0007582444	30	0.06222600	0.09770662	0.009155937
27	0.0007510396	31	0.06146776	0.09750702	0.009145371
28	0.0007383076	32	0.06071672	0.09774250	0.009221345
29	0.0006772665	33	0.05997841	0.09852311	0.009192193
30	0.0006345712	35	0.05862388	0.09791563	0.009188165
31	0.0006163813	36	0.05798931	0.09744920	0.009186845
32	0.0006135785	37	0.05737293	0.09755903	0.009185044
33	0.0006036605	38	0.05675935	0.09592614	0.008316056
34	0.0006007996	39	0.05615569	0.09597566	0.008315826
35	0.0005910233	40	0.05555489	0.09592345	0.008316417
36	0.0005650759	41	0.05496387	0.09585925	0.008319987



Comment: The tree which gives the smallest cross-validation error is given above with its CP table. In that table, nsplit equals 41 which means there are 42 leaf nodes in this tree. The most important attributes are as follows:

```
Variable importance
   Age     KM Weight     HP     CC  Doors
    55     19     15      9      1      1
```

This data is from the summary function of R. According to it, Age is the most important attribute followed by KM and Weight.

c) Make predictions in the test set and report the RMSE, MAE, and MAPE.

Code:

```
toy_predict=predict(tree_opt_toy,newdata=toyte)
rmse(actual = toyte$Price,predicted = toy_predict)
mae(actual = toyte$Price,predicted = toy_predict)
mape(actual = toyte$Price,predicted = toy_predict)
```

Output:

RMSE : 1473.07919854306

MAE : 988.901161880602

MAPE : 0.102872690963842

d) Using the randomForest package and training set, generate models by playing with "mtry", nodesize", and "ntree" parameters. What parameter combination gives the smallest RMSE in the test set?

Code: This code tries different values for "mtry", nodesize", and "ntree" parameters and reports the best combination according to RMSE error. All the trials are added to the appendix part at the end of this report.

```
# Define the parameter grid
mtry_vec <- seq(1, ncol(toytr)-1,1) # Number of variables to randomly
sample for each tree
nodesize_vec <- seq(2, 20, by=2) # Minimum size of terminal nodes
ntree_vec <- c(100, 500, 1000) # Number of trees in the forest

# Initialize variables to store the results
best_rmse <- Inf
best_params <- NULL
```

```
# Loop through the parameter grid and train/test the models
for (mtry in mtry_vec) {
  for (nodesize in nodesize_vec) {
    for (ntree in ntree_vec) {

      # Train the model on the training set
      rf <- randomForest(Price ~ ., data=toytr, mtry=mtry, ntree=ntree,
nodesize=nodesize, importance=TRUE)

      # Make predictions on the test set
      preds <- predict(rf, newdata=toyte)

      # Calculate the RMSE
      rmse <- sqrt(mean((preds - toyte$Price)^2))

      # Update the best RMSE and best parameter combination
      if (rmse < best_rmse) {
        best_rmse <- rmse
        best_params <- list(mtry=mtry, nodesize=nodesize, ntree=ntree)
      }

      # Print the progress
      cat("mtry=", mtry, ", nodesize=", nodesize, ", ntree=", ntree, ",
RMSE=", rmse, "\n")

    }
  }
}

# Print the best parameter combination and the corresponding RMSE
cat("\nBest parameters:", paste(names(best_params), unlist(best_params),
sep="="), "\n")
cat("Best RMSE:", best_rmse, "\n")
```

Output:

```
Best parameters: mtry=4 nodesize=16 ntree=500

Best RMSE: 1053.918
```

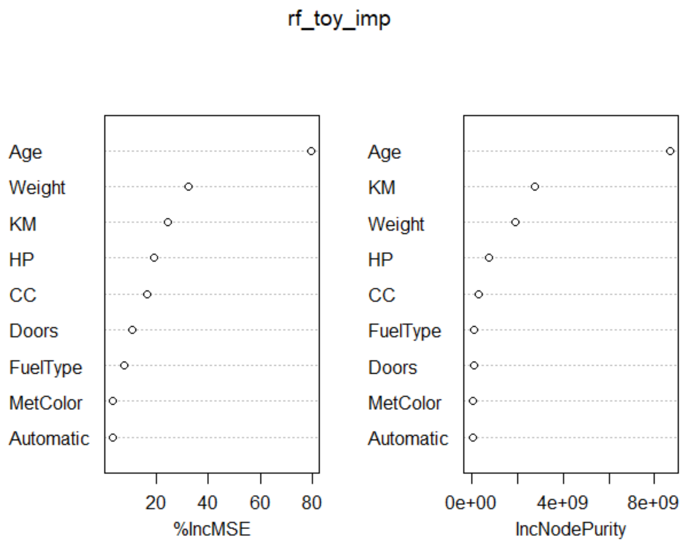
e) Comment on which input attributes are most important in making predictions.

Code:

```
rf_toy_imp=randomForest(Price~.,data=toytr,mtry=4,nodesize=16 ,ntree=500
,importance=TRUE)

#find importance
round(importance(rf_toy_imp), 2)
varImpPlot(rf_toy_imp)
```

Output:

```
          %IncMSE IncNodePurity
Age         79.38    8681039398
KM          24.25    2753552322
FuelType     7.63      92028937
HP          19.07     736518740
MetColor     3.47      41615715
Automatic    3.43      12995525
CC          16.52     303449904
Doors       10.76      58674440
Weight      32.24    1869083148
```



Comment: According to the data above, the most important attribute in making prediction is Age which is followed by Weight and KM.

f) Compare RMSE, MAE, and MAPE in the test set obtained by rpart and randomForest models.

Code:

```
#rpart predictions
rmse(actual = toyte$Price,predicted = toy_predict) #1473.07919854306
mae(actual = toyte$Price,predicted = toy_predict) #988.901161880602
mape(actual = toyte$Price,predicted = toy_predict) #0.102872690963842

#randomForest predictions
rf_predict = predict(rf_toy_imp,newdata=toyte)
rmse(actual = toyte$Price,predicted = rf_predict) #1058.612
mae(actual = toyte$Price,predicted = rf_predict) #804.0104
mape(actual = toyte$Price,predicted = rf_predict) #0.08538448
```

Output:

Rpart:	**randomForest:**
RMSE: 1473.07919854306	RMSE: 1058.612
MAE: 988.901161880602	MAE: 804.0104
MAPE: 0.102872690963842	MAPE: 0.08538448

Comment: Based on the information provided, it appears that the 'randomForest' model outperforms the 'Rpart' model in terms of the evaluation metrics.

The 'randomForest' model has a lower RMSE (Root Mean Squared Error) of 1058.612 compared to the 'Rpart' model's RMSE of 1473.079.

Similarly, the 'randomForest' model has a lower MAE (Mean Absolute Error) of 804.0104 compared to the 'Rpart' model's MAE of 988.9012.

Finally, the 'randomForest' model has a lower MAPE (Mean Absolute Percentage Error) of 0.08538448 compared to the 'Rpart' model's MAPE of 0.1028727.

The main reason why random forest performs better is that it decorrelates the attributes which gives it an advantage over regular tree models.

In summary, based on these evaluation metrics, it appears that the 'randomForest' model performs better than the 'Rpart' model.

Appendix

```
mtry= 1 , nodesize= 2 , ntree= 100 , RMSE= 1620.348
mtry= 1 , nodesize= 2 , ntree= 500 , RMSE= 1609.427
mtry= 1 , nodesize= 2 , ntree= 1000 , RMSE= 1593.353
mtry= 1 , nodesize= 4 , ntree= 100 , RMSE= 1549.631
mtry= 1 , nodesize= 4 , ntree= 500 , RMSE= 1595.64
mtry= 1 , nodesize= 4 , ntree= 1000 , RMSE= 1589.336
mtry= 1 , nodesize= 6 , ntree= 100 , RMSE= 1606.548
mtry= 1 , nodesize= 6 , ntree= 500 , RMSE= 1604.229
mtry= 1 , nodesize= 6 , ntree= 1000 , RMSE= 1588.941
mtry= 1 , nodesize= 8 , ntree= 100 , RMSE= 1596.589
mtry= 1 , nodesize= 8 , ntree= 500 , RMSE= 1638.067
mtry= 1 , nodesize= 8 , ntree= 1000 , RMSE= 1599.627
mtry= 1 , nodesize= 10 , ntree= 100 , RMSE= 1559.958
mtry= 1 , nodesize= 10 , ntree= 500 , RMSE= 1595.655
mtry= 1 , nodesize= 10 , ntree= 1000 , RMSE= 1580.195
mtry= 1 , nodesize= 12 , ntree= 100 , RMSE= 1534.819
mtry= 1 , nodesize= 12 , ntree= 500 , RMSE= 1610.834
mtry= 1 , nodesize= 12 , ntree= 1000 , RMSE= 1621.723
mtry= 1 , nodesize= 14 , ntree= 100 , RMSE= 1581.783
mtry= 1 , nodesize= 14 , ntree= 500 , RMSE= 1596.463
mtry= 1 , nodesize= 14 , ntree= 1000 , RMSE= 1587.079
mtry= 1 , nodesize= 16 , ntree= 100 , RMSE= 1528.197
mtry= 1 , nodesize= 16 , ntree= 500 , RMSE= 1602.426
mtry= 1 , nodesize= 16 , ntree= 1000 , RMSE= 1590.851
mtry= 1 , nodesize= 18 , ntree= 100 , RMSE= 1589.774
mtry= 1 , nodesize= 18 , ntree= 500 , RMSE= 1614.815
mtry= 1 , nodesize= 18 , ntree= 1000 , RMSE= 1608.403
mtry= 1 , nodesize= 20 , ntree= 100 , RMSE= 1548.862
mtry= 1 , nodesize= 20 , ntree= 500 , RMSE= 1611.554
mtry= 1 , nodesize= 20 , ntree= 1000 , RMSE= 1612.718
mtry= 2 , nodesize= 2 , ntree= 100 , RMSE= 1106.344
mtry= 2 , nodesize= 2 , ntree= 500 , RMSE= 1097.908
mtry= 2 , nodesize= 2 , ntree= 1000 , RMSE= 1098.309
mtry= 2 , nodesize= 4 , ntree= 100 , RMSE= 1101.386
mtry= 2 , nodesize= 4 , ntree= 500 , RMSE= 1100.806
mtry= 2 , nodesize= 4 , ntree= 1000 , RMSE= 1099.399
mtry= 2 , nodesize= 6 , ntree= 100 , RMSE= 1113.902
mtry= 2 , nodesize= 6 , ntree= 500 , RMSE= 1099.1
mtry= 2 , nodesize= 6 , ntree= 1000 , RMSE= 1094.038
mtry= 2 , nodesize= 8 , ntree= 100 , RMSE= 1109.872
mtry= 2 , nodesize= 8 , ntree= 500 , RMSE= 1102.544
mtry= 2 , nodesize= 8 , ntree= 1000 , RMSE= 1102.437
mtry= 2 , nodesize= 10 , ntree= 100 , RMSE= 1109.27
mtry= 2 , nodesize= 10 , ntree= 500 , RMSE= 1110.344
mtry= 2 , nodesize= 10 , ntree= 1000 , RMSE= 1105.221
mtry= 2 , nodesize= 12 , ntree= 100 , RMSE= 1108.765
mtry= 2 , nodesize= 12 , ntree= 500 , RMSE= 1104.263
mtry= 2 , nodesize= 12 , ntree= 1000 , RMSE= 1103.807
mtry= 2 , nodesize= 14 , ntree= 100 , RMSE= 1111.896
mtry= 2 , nodesize= 14 , ntree= 500 , RMSE= 1106.893
mtry= 2 , nodesize= 14 , ntree= 1000 , RMSE= 1107.038
mtry= 2 , nodesize= 16 , ntree= 100 , RMSE= 1119.954
mtry= 2 , nodesize= 16 , ntree= 500 , RMSE= 1107.37
mtry= 2 , nodesize= 16 , ntree= 1000 , RMSE= 1110.123
mtry= 2 , nodesize= 18 , ntree= 100 , RMSE= 1109.911
mtry= 2 , nodesize= 18 , ntree= 500 , RMSE= 1109.707
mtry= 2 , nodesize= 18 , ntree= 1000 , RMSE= 1112.364
mtry= 2 , nodesize= 20 , ntree= 100 , RMSE= 1121.618
mtry= 2 , nodesize= 20 , ntree= 500 , RMSE= 1111.735
mtry= 2 , nodesize= 20 , ntree= 1000 , RMSE= 1107.811
mtry= 3 , nodesize= 2 , ntree= 100 , RMSE= 1069.035
mtry= 3 , nodesize= 2 , ntree= 500 , RMSE= 1070.106
mtry= 3 , nodesize= 2 , ntree= 1000 , RMSE= 1071.82
mtry= 3 , nodesize= 4 , ntree= 100 , RMSE= 1074.526
mtry= 3 , nodesize= 4 , ntree= 500 , RMSE= 1067.293
mtry= 3 , nodesize= 4 , ntree= 1000 , RMSE= 1069.058
mtry= 3 , nodesize= 6 , ntree= 100 , RMSE= 1068.549
```

```
mtry= 3 , nodesize= 6 , ntree= 500 , RMSE= 1069.468
mtry= 3 , nodesize= 6 , ntree= 1000 , RMSE= 1065.062
mtry= 3 , nodesize= 8 , ntree= 100 , RMSE= 1068.926
mtry= 3 , nodesize= 8 , ntree= 500 , RMSE= 1065.438
mtry= 3 , nodesize= 8 , ntree= 1000 , RMSE= 1062.321
mtry= 3 , nodesize= 10 , ntree= 100 , RMSE= 1058.197
mtry= 3 , nodesize= 10 , ntree= 500 , RMSE= 1062.01
mtry= 3 , nodesize= 10 , ntree= 1000 , RMSE= 1061.525
mtry= 3 , nodesize= 12 , ntree= 100 , RMSE= 1070.094
mtry= 3 , nodesize= 12 , ntree= 500 , RMSE= 1065.909
mtry= 3 , nodesize= 12 , ntree= 1000 , RMSE= 1061.321
mtry= 3 , nodesize= 14 , ntree= 100 , RMSE= 1056.651
mtry= 3 , nodesize= 14 , ntree= 500 , RMSE= 1063.796
mtry= 3 , nodesize= 14 , ntree= 1000 , RMSE= 1057.273
mtry= 3 , nodesize= 16 , ntree= 100 , RMSE= 1066.73
mtry= 3 , nodesize= 16 , ntree= 500 , RMSE= 1060.699
mtry= 3 , nodesize= 16 , ntree= 1000 , RMSE= 1062.108
mtry= 3 , nodesize= 18 , ntree= 100 , RMSE= 1064.992
mtry= 3 , nodesize= 18 , ntree= 500 , RMSE= 1065.59
mtry= 3 , nodesize= 18 , ntree= 1000 , RMSE= 1062.891
mtry= 3 , nodesize= 20 , ntree= 100 , RMSE= 1071.151
mtry= 3 , nodesize= 20 , ntree= 500 , RMSE= 1067.33
mtry= 3 , nodesize= 20 , ntree= 1000 , RMSE= 1063.138
mtry= 4 , nodesize= 2 , ntree= 100 , RMSE= 1082.286
mtry= 4 , nodesize= 2 , ntree= 500 , RMSE= 1083.446
mtry= 4 , nodesize= 2 , ntree= 1000 , RMSE= 1080.48
mtry= 4 , nodesize= 4 , ntree= 100 , RMSE= 1076.337
mtry= 4 , nodesize= 4 , ntree= 500 , RMSE= 1080.177
mtry= 4 , nodesize= 4 , ntree= 1000 , RMSE= 1076.238
mtry= 4 , nodesize= 6 , ntree= 100 , RMSE= 1064.06
mtry= 4 , nodesize= 6 , ntree= 500 , RMSE= 1071.604
mtry= 4 , nodesize= 6 , ntree= 1000 , RMSE= 1068.118
mtry= 4 , nodesize= 8 , ntree= 100 , RMSE= 1064.2
mtry= 4 , nodesize= 8 , ntree= 500 , RMSE= 1070.584
mtry= 4 , nodesize= 8 , ntree= 1000 , RMSE= 1065.519
mtry= 4 , nodesize= 10 , ntree= 100 , RMSE= 1069.617
mtry= 4 , nodesize= 10 , ntree= 500 , RMSE= 1064.052
mtry= 4 , nodesize= 10 , ntree= 1000 , RMSE= 1063.368
mtry= 4 , nodesize= 12 , ntree= 100 , RMSE= 1071.447
mtry= 4 , nodesize= 12 , ntree= 500 , RMSE= 1062.186
mtry= 4 , nodesize= 12 , ntree= 1000 , RMSE= 1061.757
mtry= 4 , nodesize= 14 , ntree= 100 , RMSE= 1069.554
mtry= 4 , nodesize= 14 , ntree= 500 , RMSE= 1056.862
mtry= 4 , nodesize= 14 , ntree= 1000 , RMSE= 1060.894
mtry= 4 , nodesize= 16 , ntree= 100 , RMSE= 1064.944
mtry= 4 , nodesize= 16 , ntree= 500 , RMSE= 1053.918
mtry= 4 , nodesize= 16 , ntree= 1000 , RMSE= 1060.016
mtry= 4 , nodesize= 18 , ntree= 100 , RMSE= 1065.698
mtry= 4 , nodesize= 18 , ntree= 500 , RMSE= 1055.526
mtry= 4 , nodesize= 18 , ntree= 1000 , RMSE= 1056.555
mtry= 4 , nodesize= 20 , ntree= 100 , RMSE= 1061.604
mtry= 4 , nodesize= 20 , ntree= 500 , RMSE= 1053.954
mtry= 4 , nodesize= 20 , ntree= 1000 , RMSE= 1057.851
mtry= 5 , nodesize= 2 , ntree= 100 , RMSE= 1098.284
mtry= 5 , nodesize= 2 , ntree= 500 , RMSE= 1087.898
mtry= 5 , nodesize= 2 , ntree= 1000 , RMSE= 1091.643
mtry= 5 , nodesize= 4 , ntree= 100 , RMSE= 1086.8
mtry= 5 , nodesize= 4 , ntree= 500 , RMSE= 1085.503
mtry= 5 , nodesize= 4 , ntree= 1000 , RMSE= 1086.762
mtry= 5 , nodesize= 6 , ntree= 100 , RMSE= 1074.078
mtry= 5 , nodesize= 6 , ntree= 500 , RMSE= 1082.754
mtry= 5 , nodesize= 6 , ntree= 1000 , RMSE= 1079.391
mtry= 5 , nodesize= 8 , ntree= 100 , RMSE= 1079.445
mtry= 5 , nodesize= 8 , ntree= 500 , RMSE= 1072.831
mtry= 5 , nodesize= 8 , ntree= 1000 , RMSE= 1072.465
mtry= 5 , nodesize= 10 , ntree= 100 , RMSE= 1062.231
mtry= 5 , nodesize= 10 , ntree= 500 , RMSE= 1067.114
mtry= 5 , nodesize= 10 , ntree= 1000 , RMSE= 1071.504
mtry= 5 , nodesize= 12 , ntree= 100 , RMSE= 1065.728
mtry= 5 , nodesize= 12 , ntree= 500 , RMSE= 1066.183
```

```
mtry= 5 , nodesize= 12 , ntree= 1000 , RMSE= 1066.609
mtry= 5 , nodesize= 14 , ntree= 100 , RMSE= 1066.285
mtry= 5 , nodesize= 14 , ntree= 500 , RMSE= 1064.519
mtry= 5 , nodesize= 14 , ntree= 1000 , RMSE= 1063.72
mtry= 5 , nodesize= 16 , ntree= 100 , RMSE= 1061.344
mtry= 5 , nodesize= 16 , ntree= 500 , RMSE= 1065.576
mtry= 5 , nodesize= 16 , ntree= 1000 , RMSE= 1065.569
mtry= 5 , nodesize= 18 , ntree= 100 , RMSE= 1068.908
mtry= 5 , nodesize= 18 , ntree= 500 , RMSE= 1062.296
mtry= 5 , nodesize= 18 , ntree= 1000 , RMSE= 1062.604
mtry= 5 , nodesize= 20 , ntree= 100 , RMSE= 1066.787
mtry= 5 , nodesize= 20 , ntree= 500 , RMSE= 1063.524
mtry= 5 , nodesize= 20 , ntree= 1000 , RMSE= 1060.656
mtry= 6 , nodesize= 2 , ntree= 100 , RMSE= 1120.33
mtry= 6 , nodesize= 2 , ntree= 500 , RMSE= 1104.076
mtry= 6 , nodesize= 2 , ntree= 1000 , RMSE= 1104.209
mtry= 6 , nodesize= 4 , ntree= 100 , RMSE= 1101.467
mtry= 6 , nodesize= 4 , ntree= 500 , RMSE= 1096.028
mtry= 6 , nodesize= 4 , ntree= 1000 , RMSE= 1091.851
mtry= 6 , nodesize= 6 , ntree= 100 , RMSE= 1086.176
mtry= 6 , nodesize= 6 , ntree= 500 , RMSE= 1085.675
mtry= 6 , nodesize= 6 , ntree= 1000 , RMSE= 1089.433
mtry= 6 , nodesize= 8 , ntree= 100 , RMSE= 1096.786
mtry= 6 , nodesize= 8 , ntree= 500 , RMSE= 1083.353
mtry= 6 , nodesize= 8 , ntree= 1000 , RMSE= 1085.705
mtry= 6 , nodesize= 10 , ntree= 100 , RMSE= 1076.232
mtry= 6 , nodesize= 10 , ntree= 500 , RMSE= 1081.52
mtry= 6 , nodesize= 10 , ntree= 1000 , RMSE= 1078.238
mtry= 6 , nodesize= 12 , ntree= 100 , RMSE= 1083.919
mtry= 6 , nodesize= 12 , ntree= 500 , RMSE= 1072.772
mtry= 6 , nodesize= 12 , ntree= 1000 , RMSE= 1077.019
mtry= 6 , nodesize= 14 , ntree= 100 , RMSE= 1066.721
mtry= 6 , nodesize= 14 , ntree= 500 , RMSE= 1074.174
mtry= 6 , nodesize= 14 , ntree= 1000 , RMSE= 1075.933
mtry= 6 , nodesize= 16 , ntree= 100 , RMSE= 1079.954
mtry= 6 , nodesize= 16 , ntree= 500 , RMSE= 1071.38
mtry= 6 , nodesize= 16 , ntree= 1000 , RMSE= 1073.764
mtry= 6 , nodesize= 18 , ntree= 100 , RMSE= 1081.293
mtry= 6 , nodesize= 18 , ntree= 500 , RMSE= 1070.378
mtry= 6 , nodesize= 18 , ntree= 1000 , RMSE= 1069.02
mtry= 6 , nodesize= 20 , ntree= 100 , RMSE= 1067.95
mtry= 6 , nodesize= 20 , ntree= 500 , RMSE= 1069.144
mtry= 6 , nodesize= 20 , ntree= 1000 , RMSE= 1071.76
mtry= 7 , nodesize= 2 , ntree= 100 , RMSE= 1110.921
mtry= 7 , nodesize= 2 , ntree= 500 , RMSE= 1112.595
mtry= 7 , nodesize= 2 , ntree= 1000 , RMSE= 1112.544
mtry= 7 , nodesize= 4 , ntree= 100 , RMSE= 1112.335
mtry= 7 , nodesize= 4 , ntree= 500 , RMSE= 1113.253
mtry= 7 , nodesize= 4 , ntree= 1000 , RMSE= 1102.25
mtry= 7 , nodesize= 6 , ntree= 100 , RMSE= 1109.78
mtry= 7 , nodesize= 6 , ntree= 500 , RMSE= 1102.914
mtry= 7 , nodesize= 6 , ntree= 1000 , RMSE= 1099.543
mtry= 7 , nodesize= 8 , ntree= 100 , RMSE= 1118.869
mtry= 7 , nodesize= 8 , ntree= 500 , RMSE= 1092.558
mtry= 7 , nodesize= 8 , ntree= 1000 , RMSE= 1094.627
mtry= 7 , nodesize= 10 , ntree= 100 , RMSE= 1100.409
mtry= 7 , nodesize= 10 , ntree= 500 , RMSE= 1083.033
mtry= 7 , nodesize= 10 , ntree= 1000 , RMSE= 1088.216
mtry= 7 , nodesize= 12 , ntree= 100 , RMSE= 1089.305
mtry= 7 , nodesize= 12 , ntree= 500 , RMSE= 1085.151
mtry= 7 , nodesize= 12 , ntree= 1000 , RMSE= 1083.043
mtry= 7 , nodesize= 14 , ntree= 100 , RMSE= 1082.737
mtry= 7 , nodesize= 14 , ntree= 500 , RMSE= 1076.477
mtry= 7 , nodesize= 14 , ntree= 1000 , RMSE= 1085.578
mtry= 7 , nodesize= 16 , ntree= 100 , RMSE= 1095.415
mtry= 7 , nodesize= 16 , ntree= 500 , RMSE= 1083.6
mtry= 7 , nodesize= 16 , ntree= 1000 , RMSE= 1085.647
mtry= 7 , nodesize= 18 , ntree= 100 , RMSE= 1084.854
mtry= 7 , nodesize= 18 , ntree= 500 , RMSE= 1079.496
mtry= 7 , nodesize= 18 , ntree= 1000 , RMSE= 1085.363
```

```
mtry= 7 , nodesize= 20 , ntree= 100 , RMSE= 1086.704
mtry= 7 , nodesize= 20 , ntree= 500 , RMSE= 1082.546
mtry= 7 , nodesize= 20 , ntree= 1000 , RMSE= 1082.246
mtry= 8 , nodesize= 2 , ntree= 100 , RMSE= 1151.55
mtry= 8 , nodesize= 2 , ntree= 500 , RMSE= 1127.358
mtry= 8 , nodesize= 2 , ntree= 1000 , RMSE= 1128.644
mtry= 8 , nodesize= 4 , ntree= 100 , RMSE= 1117.214
mtry= 8 , nodesize= 4 , ntree= 500 , RMSE= 1120.791
mtry= 8 , nodesize= 4 , ntree= 1000 , RMSE= 1123.889
mtry= 8 , nodesize= 6 , ntree= 100 , RMSE= 1118.464
mtry= 8 , nodesize= 6 , ntree= 500 , RMSE= 1117.445
mtry= 8 , nodesize= 6 , ntree= 1000 , RMSE= 1112.763
mtry= 8 , nodesize= 8 , ntree= 100 , RMSE= 1115.279
mtry= 8 , nodesize= 8 , ntree= 500 , RMSE= 1110.754
mtry= 8 , nodesize= 8 , ntree= 1000 , RMSE= 1107.639
mtry= 8 , nodesize= 10 , ntree= 100 , RMSE= 1102.271
mtry= 8 , nodesize= 10 , ntree= 500 , RMSE= 1103.258
mtry= 8 , nodesize= 10 , ntree= 1000 , RMSE= 1102.925
mtry= 8 , nodesize= 12 , ntree= 100 , RMSE= 1105.229
mtry= 8 , nodesize= 12 , ntree= 500 , RMSE= 1098.475
mtry= 8 , nodesize= 12 , ntree= 1000 , RMSE= 1104.718
mtry= 8 , nodesize= 14 , ntree= 100 , RMSE= 1105.757
mtry= 8 , nodesize= 14 , ntree= 500 , RMSE= 1097.041
mtry= 8 , nodesize= 14 , ntree= 1000 , RMSE= 1099.567
mtry= 8 , nodesize= 16 , ntree= 100 , RMSE= 1103.41
mtry= 8 , nodesize= 16 , ntree= 500 , RMSE= 1098.217
mtry= 8 , nodesize= 16 , ntree= 1000 , RMSE= 1095.768
mtry= 8 , nodesize= 18 , ntree= 100 , RMSE= 1090.353
mtry= 8 , nodesize= 18 , ntree= 500 , RMSE= 1097.359
mtry= 8 , nodesize= 18 , ntree= 1000 , RMSE= 1097.057
mtry= 8 , nodesize= 20 , ntree= 100 , RMSE= 1115.078
mtry= 8 , nodesize= 20 , ntree= 500 , RMSE= 1094.574
mtry= 8 , nodesize= 20 , ntree= 1000 , RMSE= 1093.284
mtry= 9 , nodesize= 2 , ntree= 100 , RMSE= 1154.608
mtry= 9 , nodesize= 2 , ntree= 500 , RMSE= 1149.426
mtry= 9 , nodesize= 2 , ntree= 1000 , RMSE= 1145.794
mtry= 9 , nodesize= 4 , ntree= 100 , RMSE= 1139.231
mtry= 9 , nodesize= 4 , ntree= 500 , RMSE= 1139.489
mtry= 9 , nodesize= 4 , ntree= 1000 , RMSE= 1138.981
mtry= 9 , nodesize= 6 , ntree= 100 , RMSE= 1121.392
mtry= 9 , nodesize= 6 , ntree= 500 , RMSE= 1134.159
mtry= 9 , nodesize= 6 , ntree= 1000 , RMSE= 1130.483
mtry= 9 , nodesize= 8 , ntree= 100 , RMSE= 1123.848
mtry= 9 , nodesize= 8 , ntree= 500 , RMSE= 1134.746
mtry= 9 , nodesize= 8 , ntree= 1000 , RMSE= 1124.043
mtry= 9 , nodesize= 10 , ntree= 100 , RMSE= 1140.102
mtry= 9 , nodesize= 10 , ntree= 500 , RMSE= 1124.397
mtry= 9 , nodesize= 10 , ntree= 1000 , RMSE= 1118.173
mtry= 9 , nodesize= 12 , ntree= 100 , RMSE= 1131.322
mtry= 9 , nodesize= 12 , ntree= 500 , RMSE= 1119.997
mtry= 9 , nodesize= 12 , ntree= 1000 , RMSE= 1122.822
mtry= 9 , nodesize= 14 , ntree= 100 , RMSE= 1143.967
mtry= 9 , nodesize= 14 , ntree= 500 , RMSE= 1114.24
mtry= 9 , nodesize= 14 , ntree= 1000 , RMSE= 1119.89
mtry= 9 , nodesize= 16 , ntree= 100 , RMSE= 1127.525
mtry= 9 , nodesize= 16 , ntree= 500 , RMSE= 1116.108
mtry= 9 , nodesize= 16 , ntree= 1000 , RMSE= 1117.116
mtry= 9 , nodesize= 18 , ntree= 100 , RMSE= 1128.085
mtry= 9 , nodesize= 18 , ntree= 500 , RMSE= 1116.793
mtry= 9 , nodesize= 18 , ntree= 1000 , RMSE= 1115.573
mtry= 9 , nodesize= 20 , ntree= 100 , RMSE= 1111.458
mtry= 9 , nodesize= 20 , ntree= 500 , RMSE= 1110.278
mtry= 9 , nodesize= 20 , ntree= 1000 , RMSE= 1114.848
```