Homework 4

Na Yun Cho

```
library(ISLR)
library(mlbench)
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

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

```
## Loading required package: grid
```

```
## Loading required package: mvtnorm
```

```
## Loading required package: modeltools
```

```
## Loading required package: stats4
```

```
## Loading required package: strucchange
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
## Loading required package: sandwich
```

```
library(partykit)
```

```
## Loading required package: libcoin
```

```
##
## Attaching package: 'partykit'
```

```
## The following objects are masked from 'package:party':
##
##     cforest, ctree, ctree_control, edge_simple, mob, mob_control,
##     node_barplot, node_bivplot, node_boxplot, node_inner, node_surv,
##     node_terminal, varimp
```

```
library(plotmo)
```

```
## Loading required package: Formula

## Loading required package: plotrix

## Loading required package: TeachingDemos
```

```
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(randomForest)
```

```
## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(ranger)
```

```
##
## Attaching package: 'ranger'

## The following object is masked from 'package:randomForest':
##
##     importance
```

```
library(gbm)
```

```
## Loaded gbm 2.1.8
```

```
library(pdp)
library(lasso2)
```

```
## R Package to solve regression problems while imposing
##   an L1 constraint on the parameters. Based on S-plus Release 2.1
## Copyright (C) 1998, 1999
## Justin Lokhorst   <jlokhors@stats.adelaide.edu.au>
## Berwin A. Turlach <bturlach@stats.adelaide.edu.au>
## Bill Venables     <wvenable@stats.adelaide.edu.au>
##
## Copyright (C) 2002
## Martin Maechler <maechler@stat.math.ethz.ch>
```

```
library(tidyverse) # data manipulation
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v tibble  3.0.6     v dplyr   1.0.4
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
## v purrr   0.3.4
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x stringr::boundary()    masks strucchange::boundary()
## x dplyr::combine()       masks randomForest::combine()
## x dplyr::filter()        masks stats::filter()
## x dplyr::lag()           masks stats::lag()
## x purrr::lift()          masks caret::lift()
## x randomForest::margin() masks ggplot2::margin()
## x purrr::partial()       masks pdp::partial()
```

```
library(ISLR) # data Problem 2
library(patchwork)
library(vip)
```

```
##
## Attaching package: 'vip'
```

```
## The following object is masked from 'package:utils':
##
##     vi
```

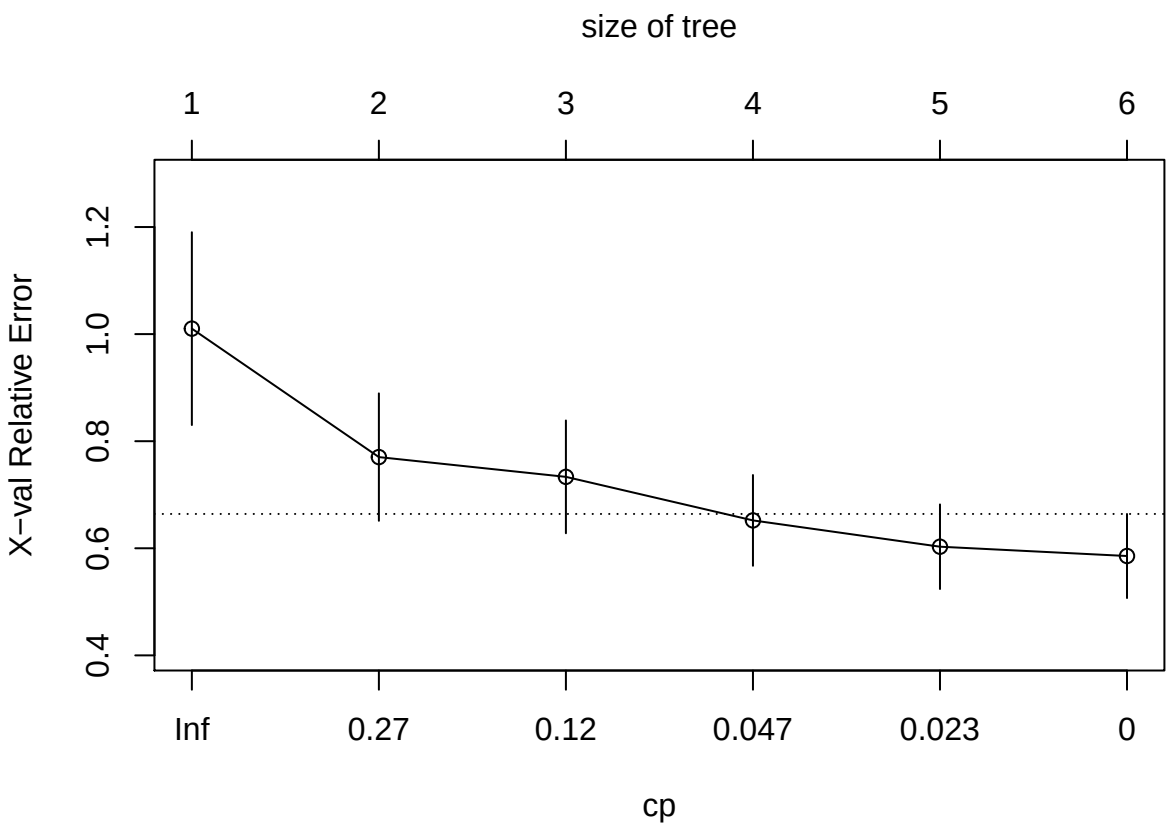
1(a)

Fit a regression tree with lpsa as the response and the other variables as predictors.

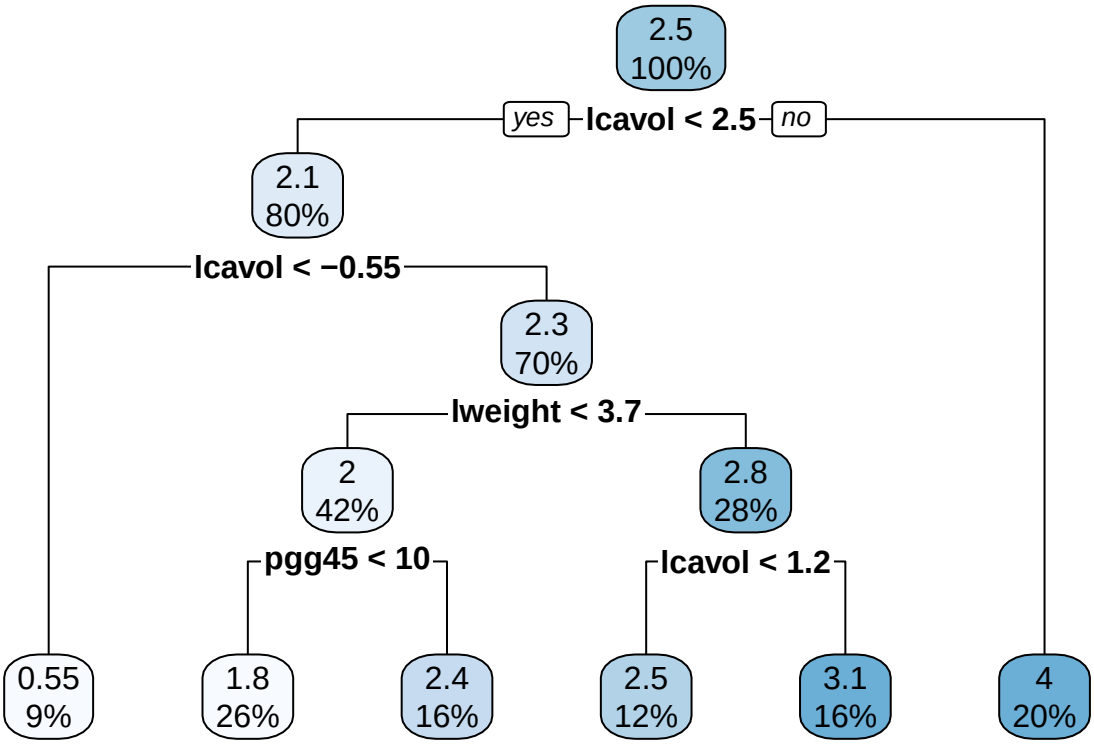
```
set.seed(1)
data(Prostate)
Prostate <-na.omit(Prostate)

# partition the dataset
trRows <- createDataPartition(Prostate$lpsa,
                              p =0.75, list =F)

tree1 <- rpart(formula = lpsa ~ . ,
               data = Prostate, subset = trRows,
               control = rpart.control(cp = 0))
printcp(tree1)
```

```
##
## Regression tree:
## rpart(formula = lpsa ~ ., data = Prostate, subset = trRows, control = rpart.control(cp = 0))
##
## Variables actually used in tree construction:
## [1] lcavol  lweight pgg45
##
## Root node error: 106.7/74 = 1.4418
##
## n= 74
##
##         CP nsplit rel error  xerror     xstd
## 1 0.383415      0   1.00000 1.01023 0.180056
## 2 0.183917      1   0.61659 0.77032 0.119016
## 3 0.076423      2   0.43267 0.73339 0.105366
## 4 0.029412      3   0.35624 0.65205 0.084753
## 5 0.018601      4   0.32683 0.60292 0.079044
## 6 0.000000      5   0.30823 0.58555 0.078534
```

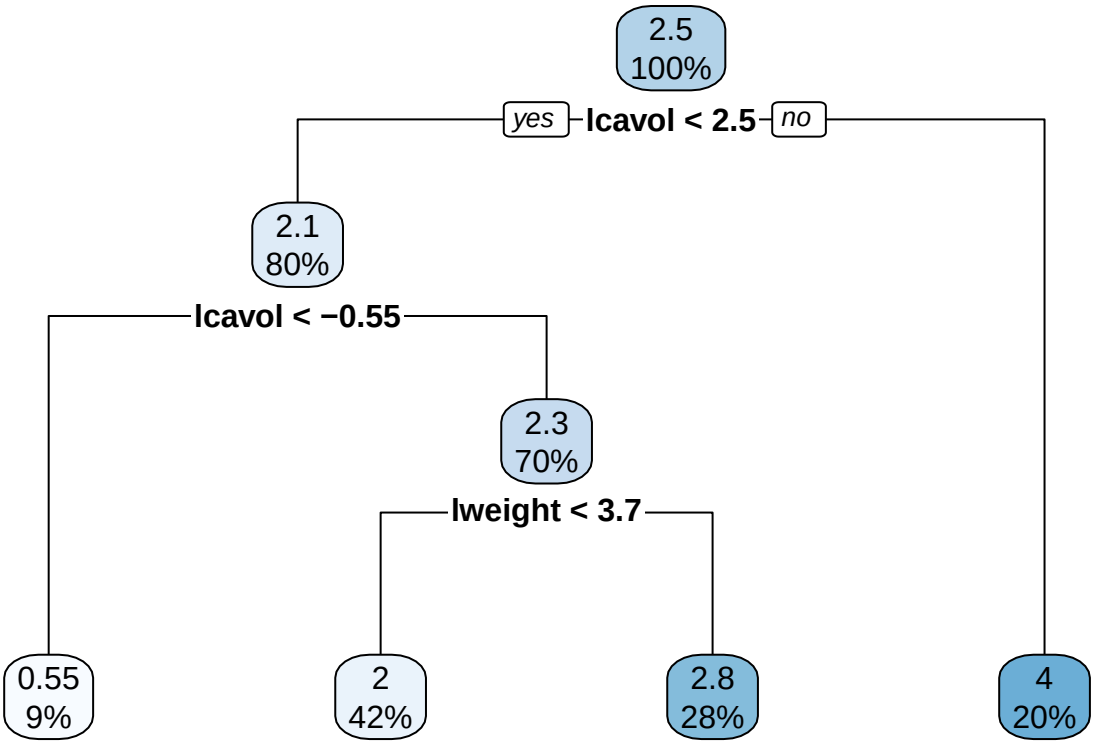
```
cpTable <- tree1$cptable
plotcp(tree1)
```




```
# tree using lowest cross validation error
minErr <- which.min(cpTable[,4])
tree3 <- prune(tree1, cp = cpTable[minErr,1])
rpart.plot(tree3)
```

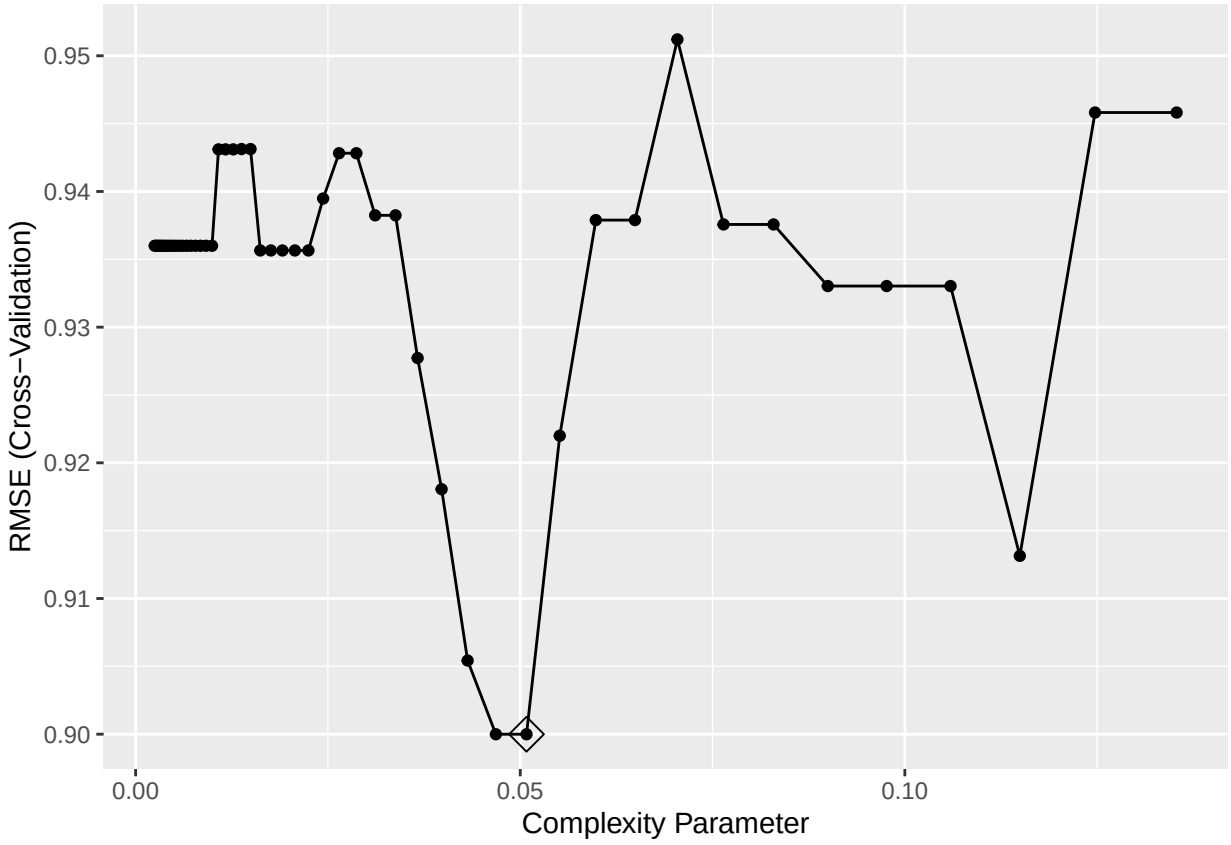



```
# tree using the 1SE rule
tree4 <- prune(tree1, cp = cpTable[cpTable[,4]<cpTable[minErr,4]+cpTable[minErr,5],1][1])
rpart.plot(tree4)
```

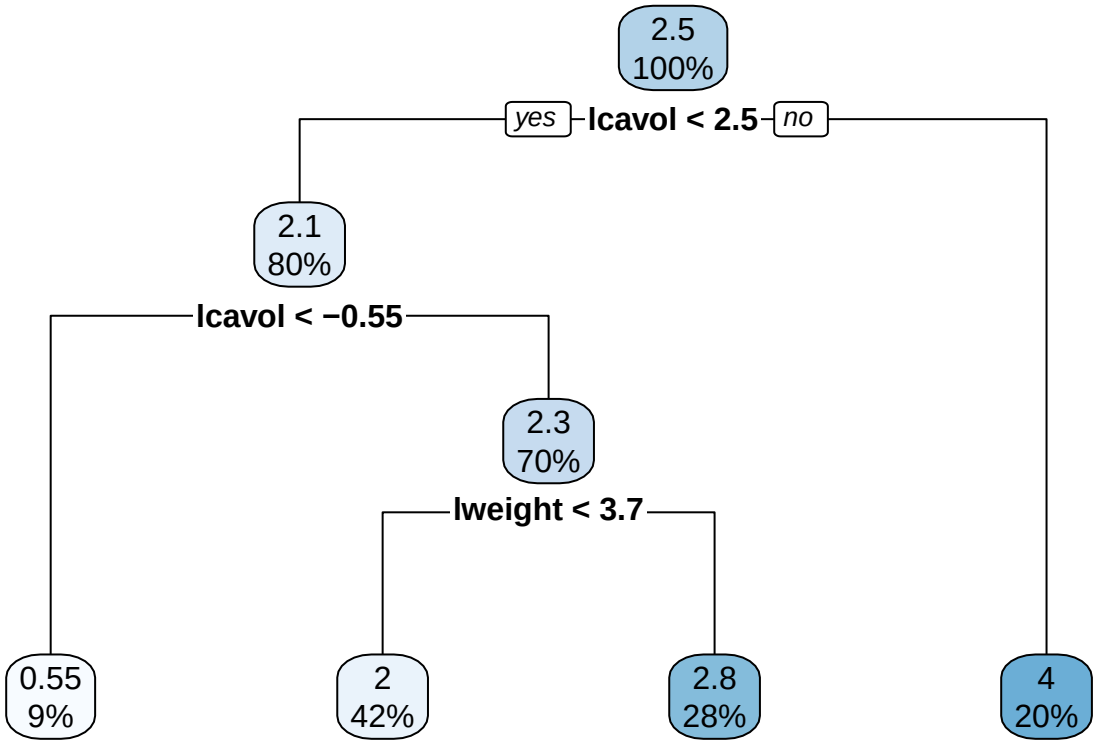


Tree size of 6 corresponds to the lowest cross-validation error. Tree size of 4 is obtained using the 1 SE rule. Thus, the sizes are different.

```r
# use caret to do cross validation
set.seed(1)
ctrl <- trainControl(method = "cv")
rpart.fit <- train(lpsa~., Prostate[trRows,],
                   method = "rpart",
                   tuneGrid = data.frame(cp = exp(seq(-6,-2, length = 50))),
                   trControl = ctrl)
ggplot(rpart.fit, highlight = TRUE)
```




```r
rpart.plot(rpart.fit$finalModel)
```



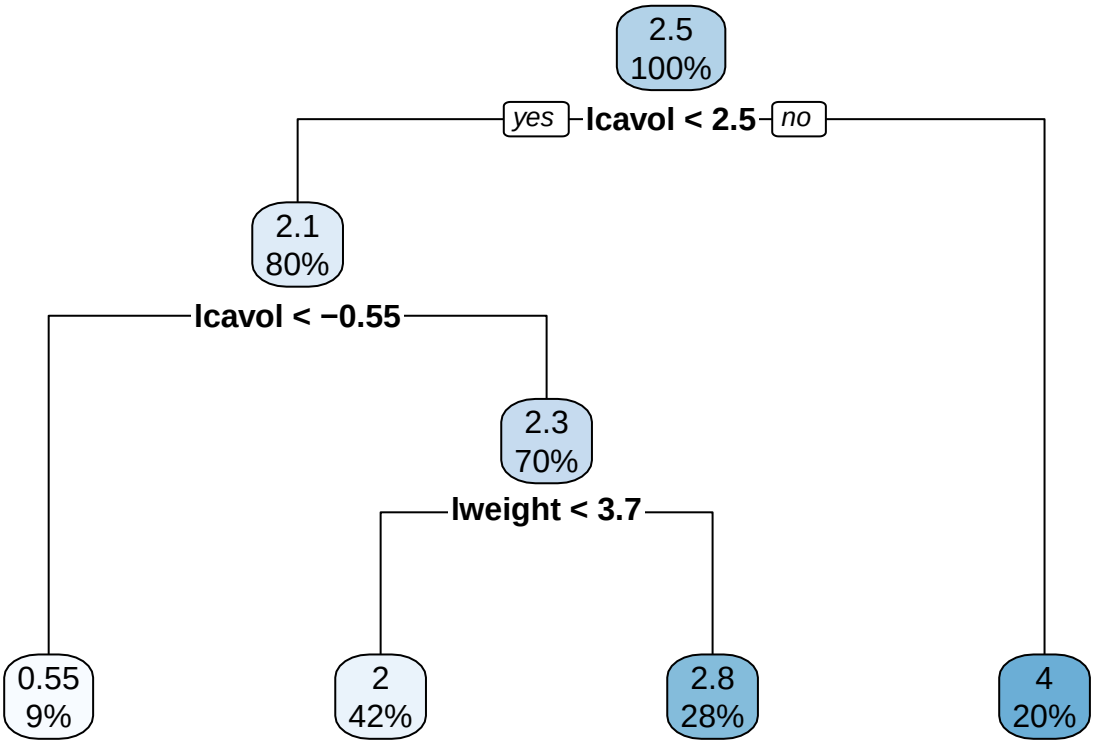

```
rpart.fit$bestTune
```

```
##            cp
## 38 0.05081357
```

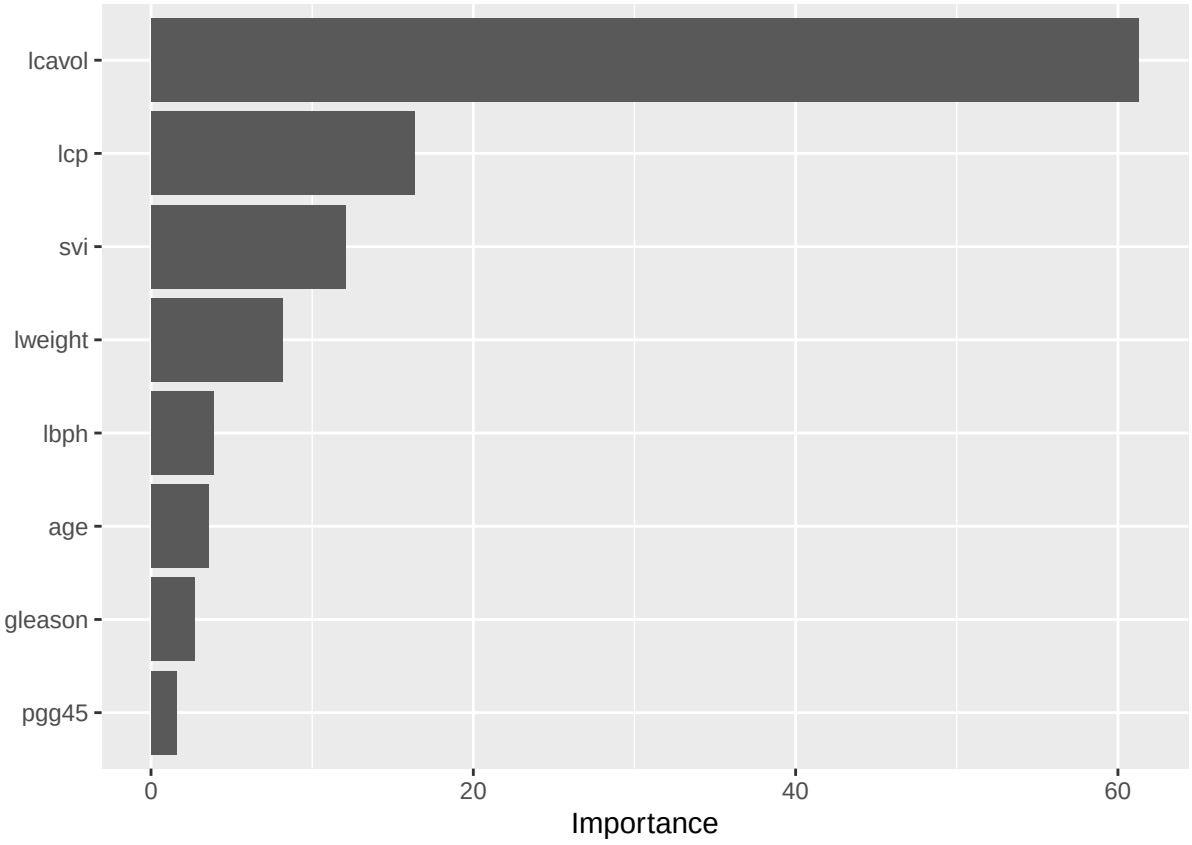
The optimal tree size is 4.

1(b) plot of the final tree

```
tree5 <- prune(tree1, cp = 0.05081357)
rpart.plot(tree5)
```




```
vip(tree5)
```

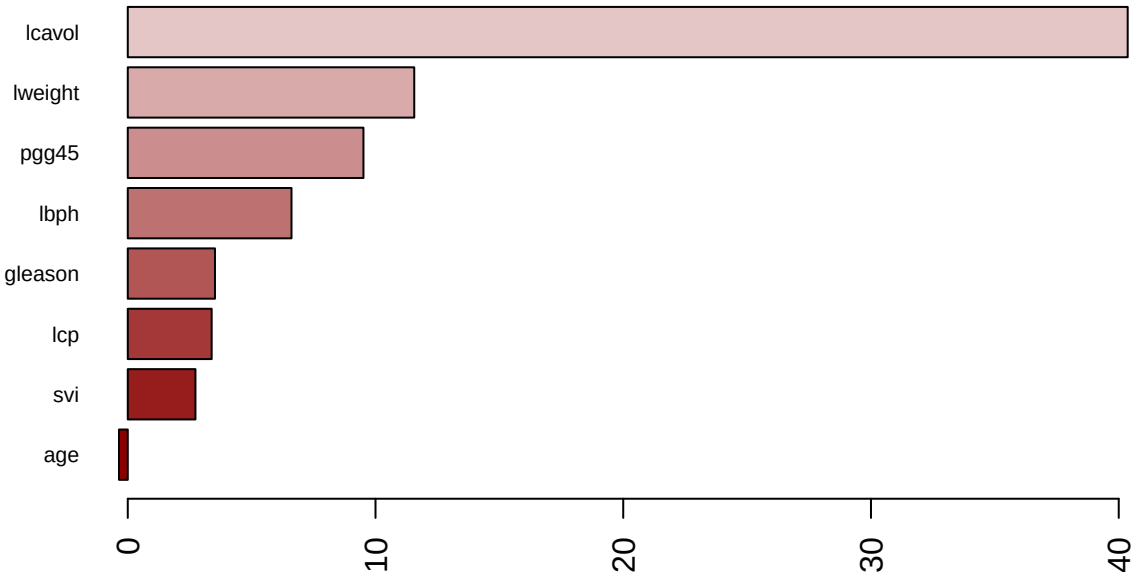


The final tree I chose has a size of 4. The interpretation of the bottom right terminal node is that for observations

with `lcavol` (log cancer volume) that are greater than or equal to 2.5, the value of response (log prostate specific antigen) is predicted to be 4. 20% of the data correspond to this bottom right terminal node.

1(c) bagging

```
# perform bagging
bagging <- ranger(lpsa~., Prostate[trRows,],
                  mtry = 8,
                  importance = "permutation",
                  min.node.size = 25,
                  scale.permutation.importance = TRUE)
barplot(sort(ranger::importance(bagging), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.7,
        col = colorRampPalette(colors = c("darkred","white","darkblue"))(19))
```



For bagging, we do not have to tune the `mtry` parameter. The variable importance plot is shown above. 'lcavol' is the most important predictor, 'lweight' is the second most important one, and 'pgg45' is the third important predictor. Meanwhile, 'age' is the least important predictor.

1(d) random forest

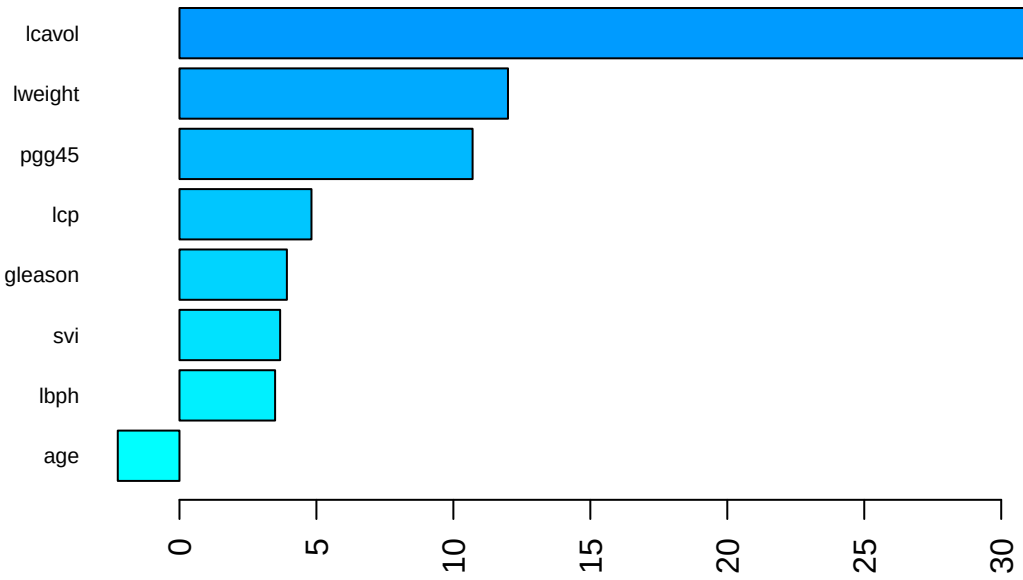
```
rf.grid <- expand.grid(mtry = 1:8,
                       splitrule = "variance",
                       min.node.size = 1:30)
rf.fit <- train(lpsa~., Prostate[trRows,],
                method = "ranger",
                tuneGrid = rf.grid,
                trControl = ctrl)
ggplot(rf.fit, highlight = TRUE)
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values because
```

```
## more than 6 becomes difficult to discriminate; you have 8. Consider
## specifying shapes manually if you must have them.
```

```
## Warning: Removed 60 rows containing missing values (geom_point).
```

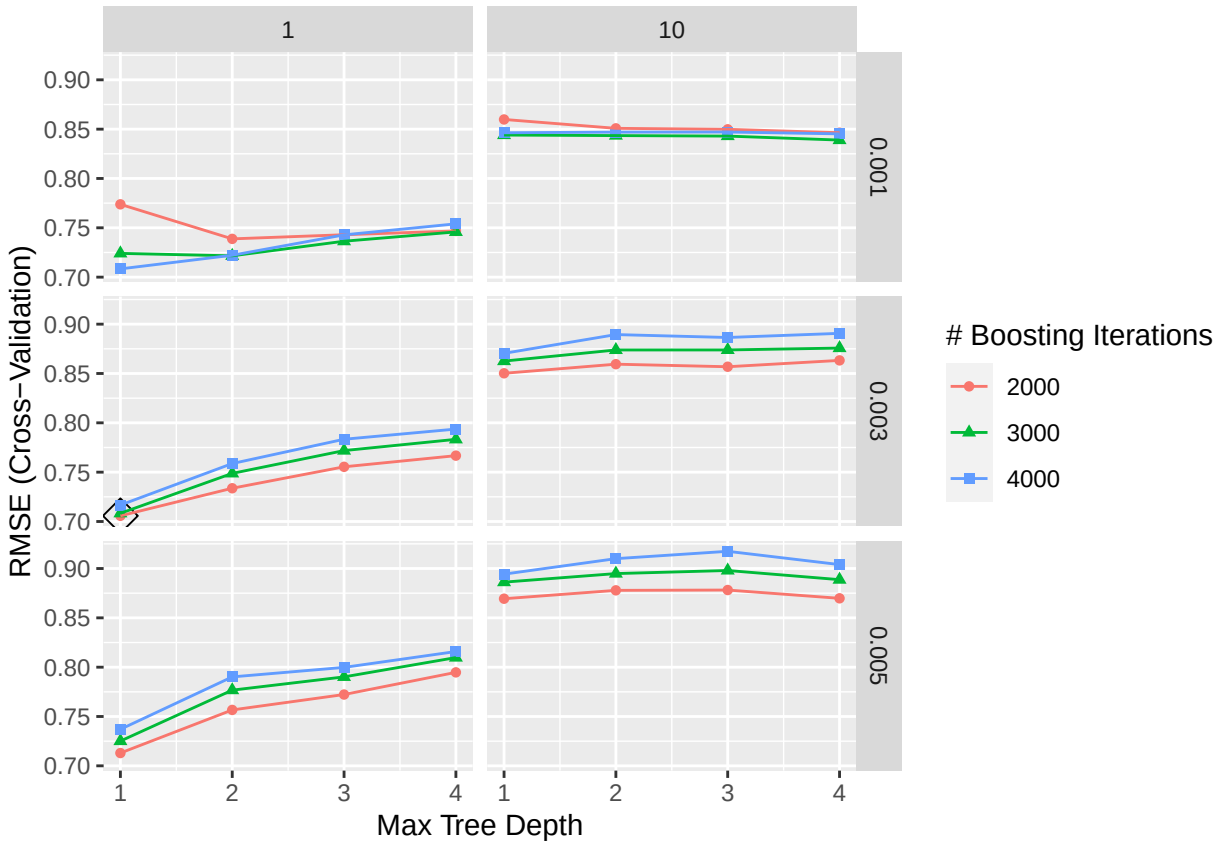

```
rf.final.per <- ranger(lpsa ~ . ,
                       Prostate[trRows,],
                       mtry = rf.fit$bestTune[[1]],
                       splitrule = "variance",
                       min.node.size = rf.fit$bestTune[[3]],
                       importance = "permutation",
                       scale.permutation.importance = TRUE)

barplot(sort(ranger::importance(rf.final.per), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.7,
        col = colorRampPalette(colors = c("cyan","blue"))(19))
```



'lcavol' is the most important predictor, 'lweight' is the second most important predictor, and 'pgg45' is the third most important one. Meanwhile, 'age' is the least important predictor.

1(e) boosting

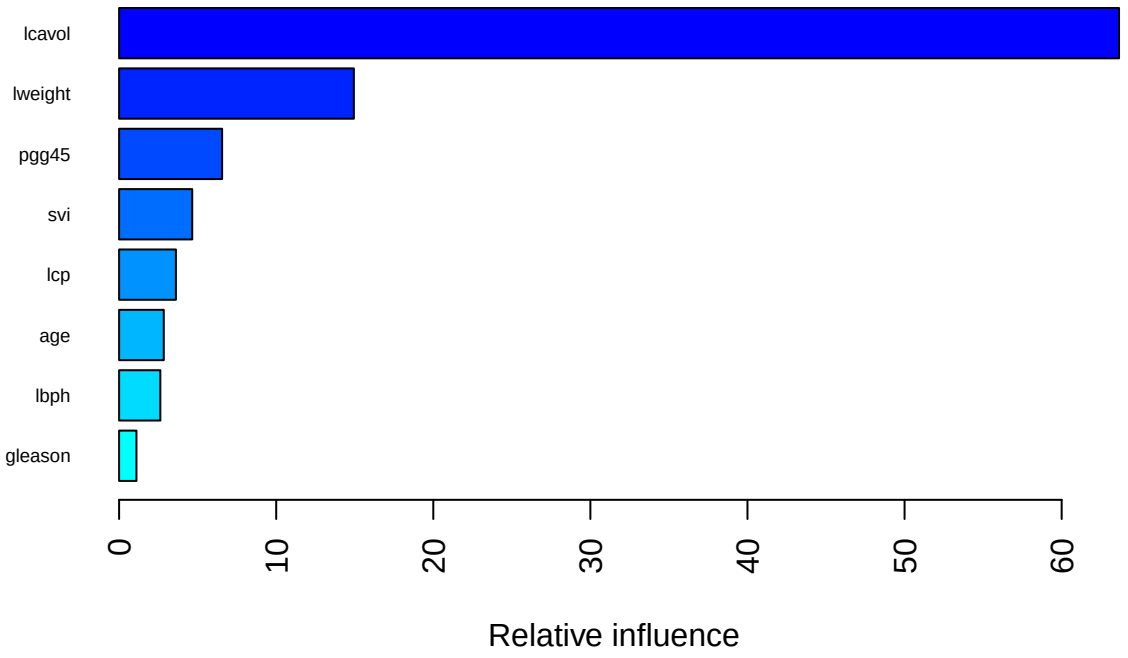
```
set.seed(1)
gbm.grid <- expand.grid(n.trees = c(2000,3000,4000),
                        interaction.depth = 1:4,
                        shrinkage = c(0.001,0.003,0.005),
                        n.minobsinnode = c(1,10))
gbm.fit <- train(lpsa ~ . ,
                 Prostate[trRows,],
                 method = "gbm",
                 tuneGrid = gbm.grid,
                 trControl = ctrl,
                 verbose = FALSE)

ggplot(gbm.fit, highlight = TRUE)
```




```
gbm.fit$bestTune
```

```
##    n.trees interaction.depth shrinkage n.minobsinnode
## 25    2000                 1     0.003              1
```

```
# variable importance
summary(gbm.fit$finalModel, las = 2, cBars = 19, cex.names = 0.6)
```




```
##             var   rel.inf
## lcavol   lcavol 63.653164
## lweight lweight 14.944056
## pgg45     pgg45  6.558304
## svi         svi  4.655160
## lcp         lcp  3.616471
## age         age  2.844074
## lbph       lbph  2.624705
## gleason gleason  1.104065
```

I can see that 'lcavol' is the most important variable. 'lweight' is the second important variable and 'pgg45' is the third important variable. 'gleason' is the least important variable.

1(f) model selection

```
resamp <- resamples(list(gbm = gbm.fit, rf = rf.fit, rpart = rpart.fit))
summary(resamp)
```

```
##
## Call:
## summary.resamples(object = resamp)
##
## Models: gbm, rf, rpart
## Number of resamples: 10
##
## MAE
##            Min.  1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## gbm   0.3266556 0.514022 0.5735461 0.5922500 0.6877162 0.8733316    0
## rf    0.4309222 0.535450 0.5892049 0.6002461 0.6768462 0.7344530    0
```

```
## rpart 0.4049165 0.635586 0.8036583 0.7637270 0.8636040 1.1082178    0
##
## RMSE
##            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## gbm   0.4095328 0.5927152 0.7095759 0.7057535 0.8061828 1.0156779    0
## rf    0.5092434 0.6384219 0.7065213 0.7157260 0.7607019 0.9889684    0
## rpart 0.5666627 0.7488817 0.9075289 0.8999948 1.0327368 1.2548393    0
##
## Rsquared
##               Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## gbm   0.073430127 0.4517138 0.7311553 0.6109801 0.8174483 0.8991831    0
## rf    0.566439201 0.7219887 0.7661652 0.7548811 0.8107309 0.9045177    0
## rpart 0.009947614 0.1805950 0.5204848 0.4765913 0.7641562 0.8385933    0
```

I would use ensemble methods rather than a single regression tree to predict PSA level, because ensemble methods show higher prediction accuracy. Among the ensemble methods, bagging is a special case of random forest in which mtry equals the total number of predictors, which is 8 in this assignment. According to the tuning process of random forest, the best mtry shows as 7, which is smaller than 8. This indicates random forest is better than bagging. Here, I compare regression tree, random forest, and boosting using cross validation. It shows that Boosting has lower mean cross-validation RMSE than that of random forest, which indicates that it has the best prediction accuracy. Therefore, I will choose the boosting model to predict PSA level.

2(a) classification tree

```
data(OJ)
oj <-
  as.tibble(OJ) %>%
  mutate(Store7 = recode(Store7, '1' = 'Yes', '2' = 'No'),
         Store7 = as.numeric(Store7))
```

```
## Warning: `as.tibble()` is deprecated as of tibble 2.0.0.
## Please use `as_tibble()` instead.
## The signature and semantics have changed, see `?as_tibble`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
#split the data into training and test sets
set.seed(1)
rowTrain <- createDataPartition(y = oj$Purchase,
                                p = 799/1070,
                                list = FALSE)
train_df = oj[rowTrain,]
```

```
## Warning: The `i` argument of ``[`()` can't be a matrix as of tibble 3.0.0.
## Convert to a vector.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

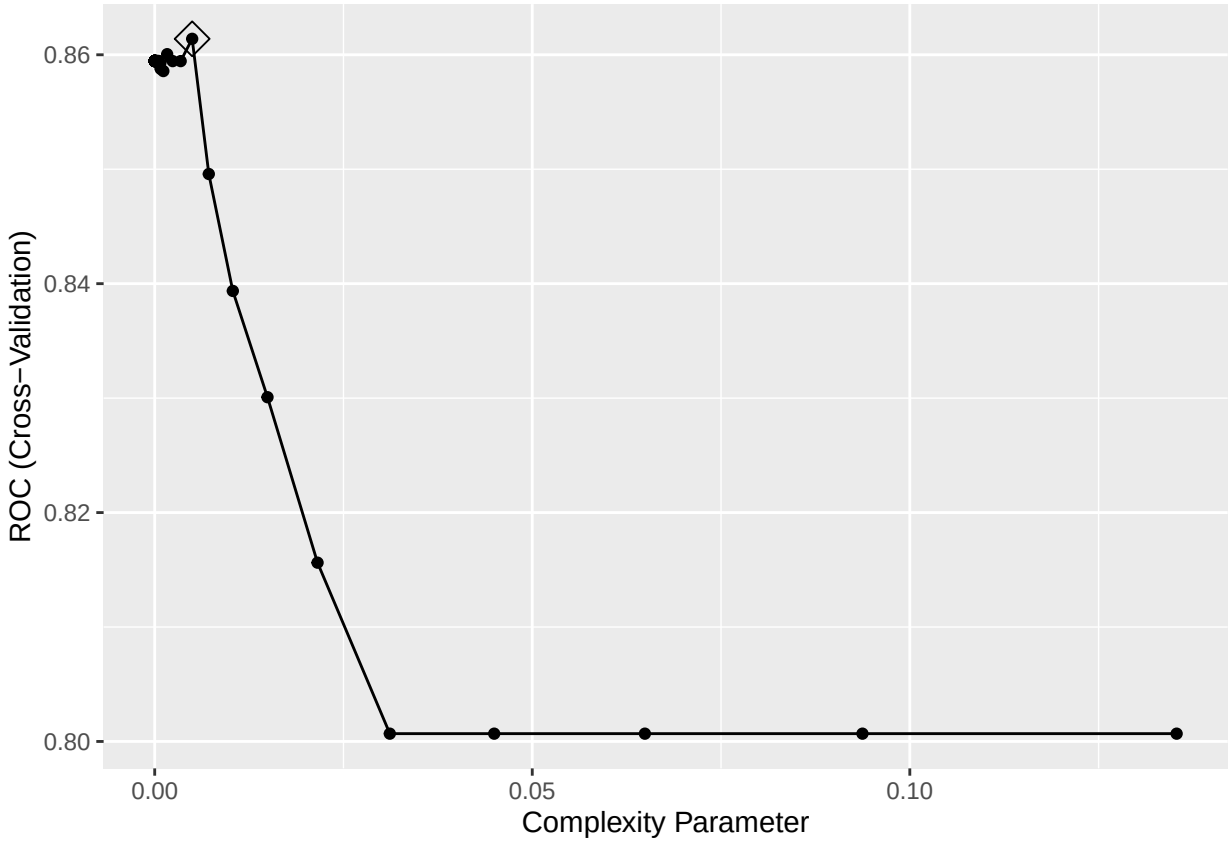
```
test_df = oj[-rowTrain,]
dim(train_df)
```

```
## [1] 800  18
```

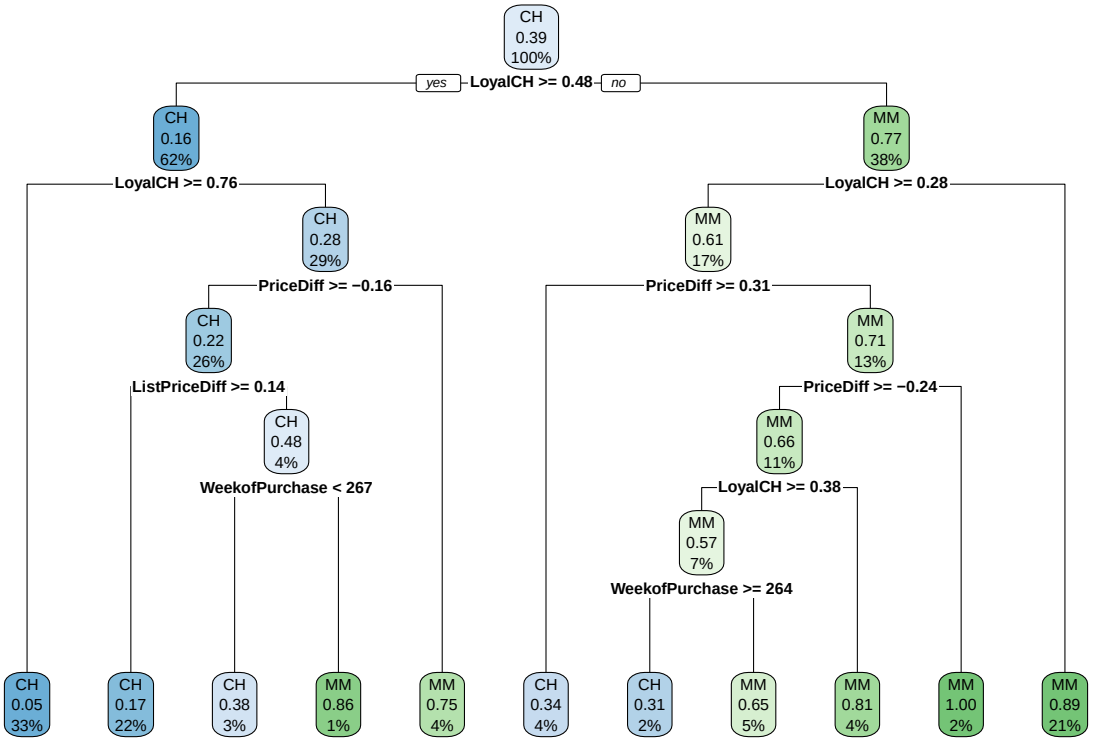
```
ctrl <- trainControl(method = "cv",
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE)
set.seed(1)
rpart.fit <- train(Purchase~., train_df,
                   method = "rpart",
                   tuneGrid = data.frame(cp = exp(seq(-20,-2, len = 50))),
                   trControl = ctrl,
                   metric = "ROC")

rpart.fit$bestTune
```

```
##             cp
## 41 0.004961126
```

```
ggplot(rpart.fit, highlight = TRUE)
```




```
# Plot of the final model
rpart.plot(rpart.fit$finalModel)
```




```
test_df$probCH = predict(rpart.fit$finalModel, newdata = test_df, type = "prob")[,1]
test_df$pred = if_else(test_df$probCH > 0.5, 'CH', 'MM')
# Classification error rate
1 - mean(test_df$pred == test_df$Purchase)
```

```
## [1] 0.1851852
```

The test classification error rate is 0.1851852.

2(b) random forest

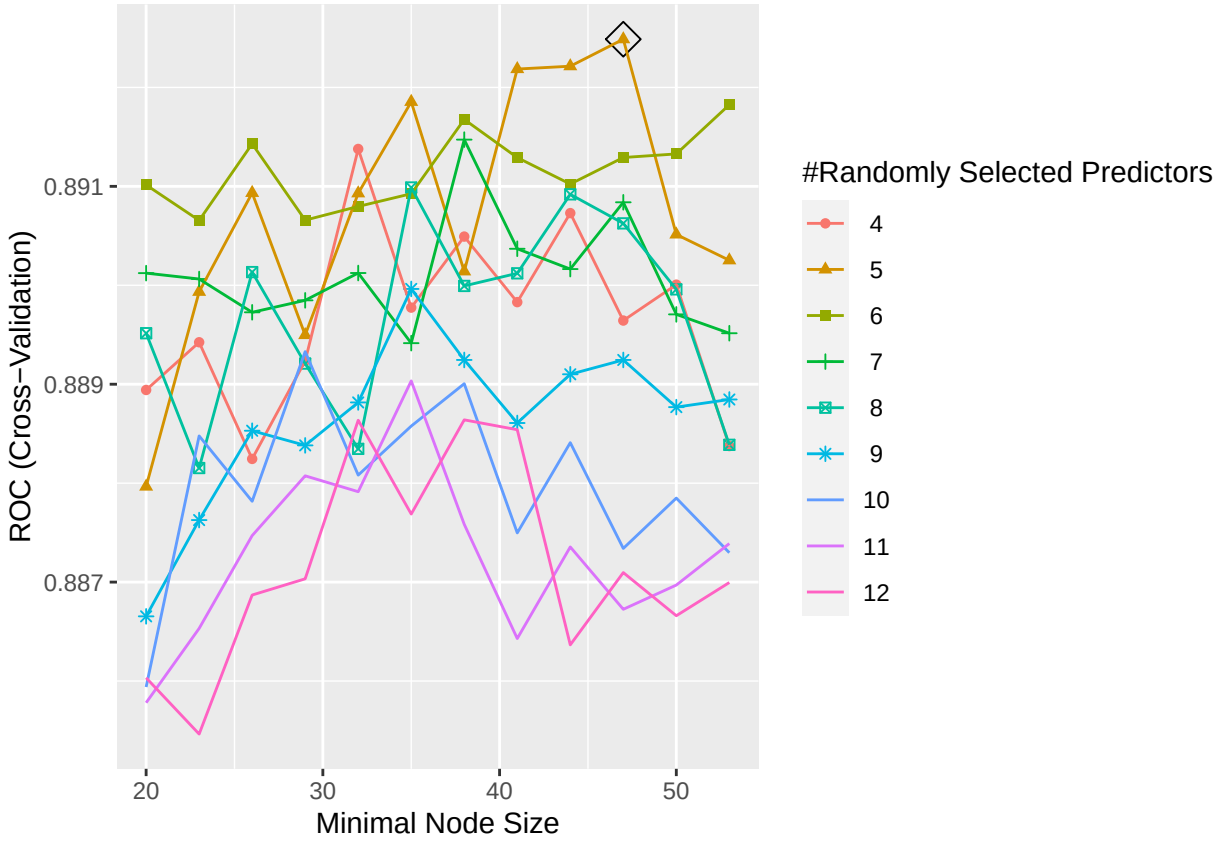
```
rf.grid <- expand.grid(mtry = seq(4,12, by=1),
                       splitrule = "gini",
                       min.node.size = seq(20,55, by=3))
set.seed(1)
rf.fit <- train(Purchase~., train_df,
                method = "ranger",
                tuneGrid = rf.grid,
                metric = "ROC",
                trControl = ctrl)

rf.fit$bestTune
```

```
##    mtry splitrule min.node.size
## 22    5      gini            47
```

```
ggplot(rf.fit, highlight = TRUE)
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values because
## more than 6 becomes difficult to discriminate; you have 9. Consider
## specifying shapes manually if you must have them.
```

```
## Warning: Removed 36 rows containing missing values (geom_point).
```



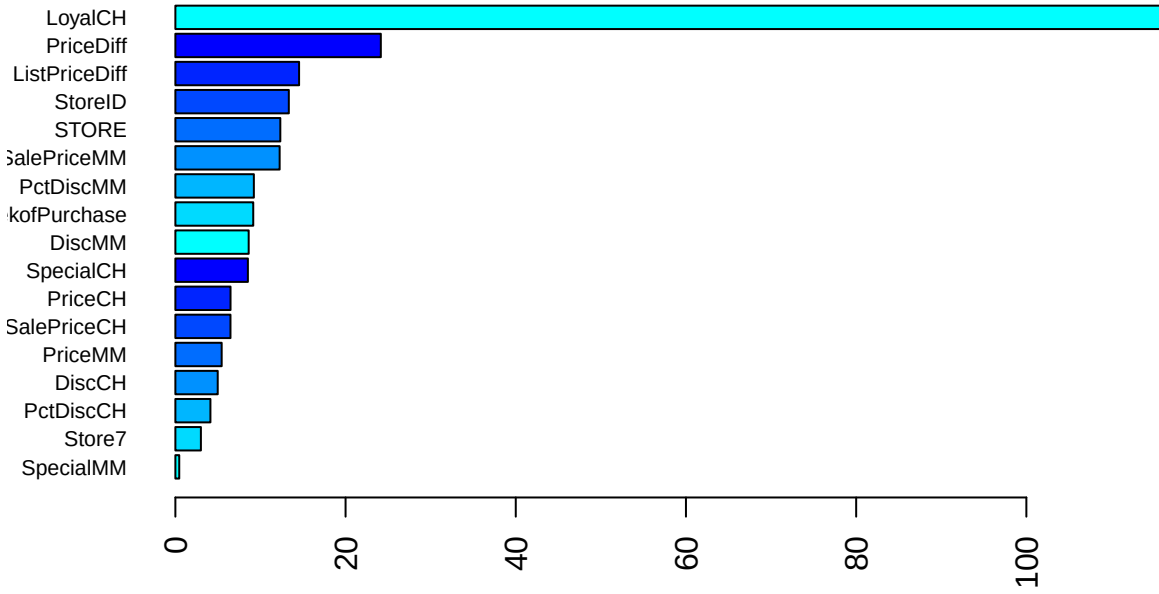

```
rf.pred <- predict(rf.fit, newdata = test_df, type = "prob")[,1]
pred_rf = if_else(rf.pred > 0.5, 'CH', 'MM')
# test error rate
1 - mean(pred_rf == test_df$Purchase)
```

```
## [1] 0.1703704
```

The test error rate is 0.17.

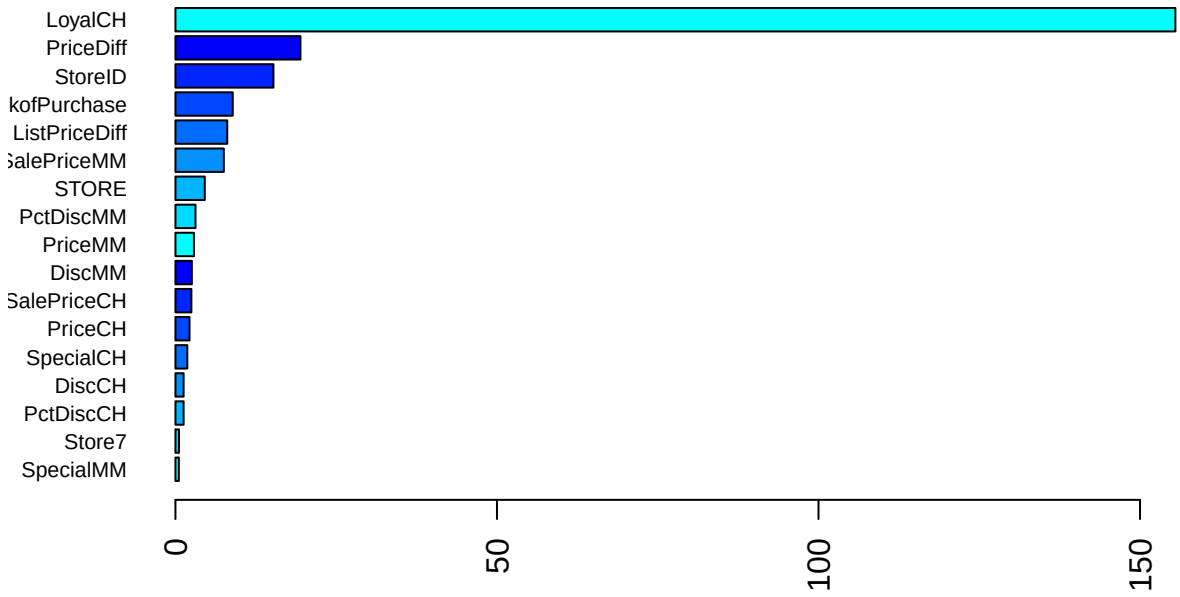
```
set.seed(1)
rf2.final.per <- ranger(Purchase~., train_df,
                        mtry = 9,
                        min.node.size = 41,
```

```
                        splitrule = "gini",
                        importance = "permutation",
                        scale.permutation.importance = TRUE)
barplot(sort(ranger::importance(rf2.final.per), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.7,
        col = colorRampPalette(colors = c("cyan","blue"))(8))
```



variable importance

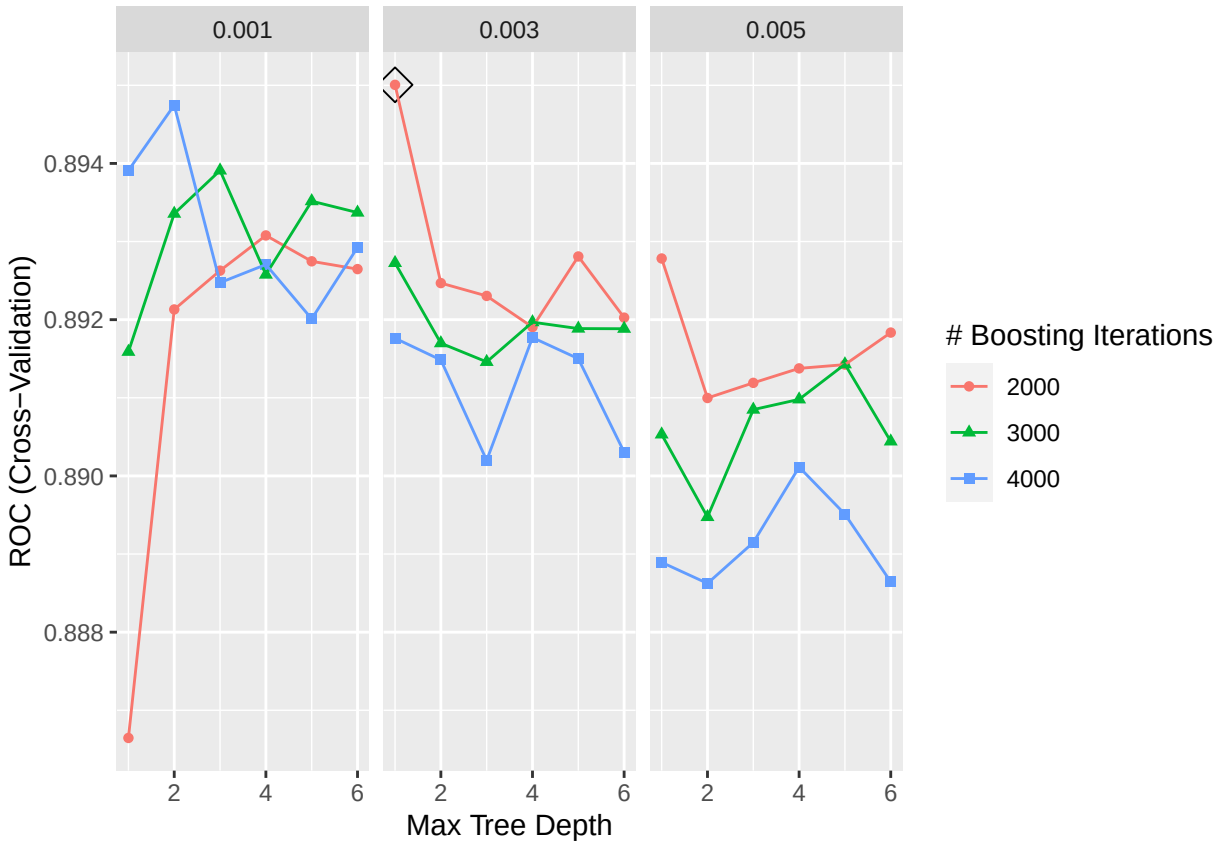
```
set.seed(1)
rf2.final.imp <- ranger(Purchase~., train_df,
                        mtry = 9,
                        splitrule = "gini",
                        min.node.size = 41,
                        importance = "impurity")
barplot(sort(ranger::importance(rf2.final.imp), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.7,
        col = colorRampPalette(colors = c("cyan","blue"))(8))
```



I can see that the most important variable is 'LoyalCH', and the second most important variable is 'PriceDiff'. Next, the 'ListPriceDiff' variable is important. The least important variable is 'SpecialMM'.

2(c) boosting

```
gbmA.grid <- expand.grid(n.trees = c(2000,3000,4000),
                         interaction.depth = 1:6,
                         shrinkage = c(0.001,0.003,0.005),
                         n.minobsinnode = 1)
set.seed(1)

gbmA.fit <- train(Purchase~., train_df,
                  tuneGrid = gbmA.grid,
                  trControl = ctrl,
                  method = "gbm",
                  distribution = "adaboost",
                  metric = "ROC",
                  verbose = FALSE)

ggplot(gbmA.fit, highlight = TRUE)
```

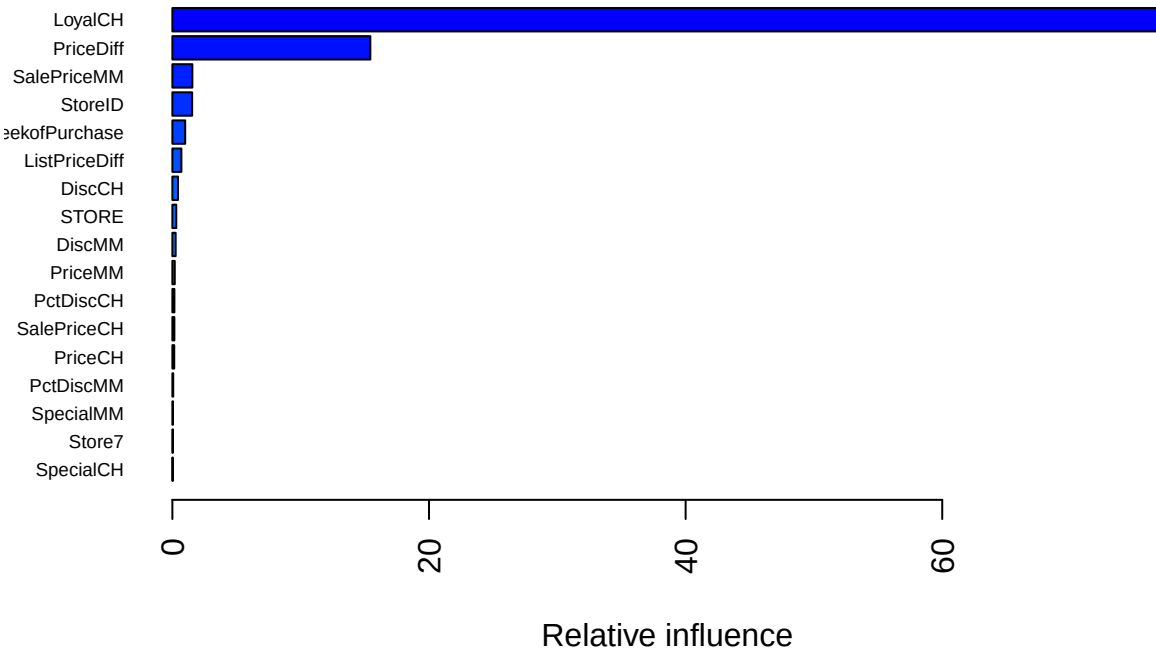



```
gbmA.pred <- predict(gbmA.fit, newdata = test_df, type = "prob")[,1]
class_gbmA = if_else(gbmA.pred > 0.5, 'CH', 'MM')
# test error rate
1 - mean(class_gbmA == test_df$Purchase)
```

```
## [1] 0.1888889
```

The test error rate is 0.1889.

```
summary(gbmA.fit$finalModel, las = 2, cBars = 19, cex.names = 0.6)
```



variable importance

```
##                            var     rel.inf
## LoyalCH                LoyalCH 77.93138473
## PriceDiff            PriceDiff 15.43011830
## SalePriceMM        SalePriceMM  1.55889463
## StoreID                StoreID  1.54225093
## WeekofPurchase WeekofPurchase  1.00016008
## ListPriceDiff    ListPriceDiff  0.70347008
## DiscCH                  DiscCH  0.43771073
## STORE                    STORE  0.30374700
## DiscMM                  DiscMM  0.25512414
## PriceMM                PriceMM  0.18538102
## PctDiscCH            PctDiscCH  0.15778372
## SalePriceCH        SalePriceCH  0.15628042
## PriceCH                PriceCH  0.14497268
## PctDiscMM            PctDiscMM  0.06620623
## SpecialMM            SpecialMM  0.04900651
## Store7                  Store7  0.04424588
## SpecialCH            SpecialCH  0.03326293
```

The most important variable is 'LoyalCH', and the second most important variable is 'PriceDiff'. Next, 'SalePriceMM' is the third most important variable. The least important variable seems to be 'Special CH'.