Decision trees

For this lab, we will use the Carseats data set from the ISLR package.

(install and) load the package with the data set

```
# install.packages('ISLR')
library(ISLR)
```

Carseats is a simulated data set containing sales of child car seats at 400 different stores. To inform about this data set, type ?Carseats

```
?Carseats
```

We'll start by examining the structure of the data set

```
str(Carseats)
```

```
## 'data.frame':    400 obs. of  11 variables:
##  $ Sales      : num  9.5 11.22 10.06 7.4 4.15 ...
##  $ CompPrice  : num  138 111 113 117 141 124 115 136 132 132 ...
##  $ Income     : num  73 48 35 100 64 113 105 81 110 113 ...
##  $ Advertising: num  11 16 10 4 3 13 0 15 0 0 ...
##  $ Population : num  276 260 269 466 340 501 45 425 108 131 ...
##  $ Price      : num  120 83 80 97 128 72 108 120 124 124 ...
##  $ ShelveLoc  : Factor w/ 3 levels "Bad","Good","Medium": 1 2 3 3 1 1 3 2
3 3 ...
##  $ Age        : num  42 65 59 55 38 78 71 67 76 76 ...
##  $ Education  : num  17 10 12 14 13 16 15 10 10 17 ...
##  $ Urban      : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 2 2 1 1 ...
##  $ US         : Factor w/ 2 levels "No","Yes": 2 2 2 2 1 2 1 2 1 2 ...
```

Based on the Sales variable, we'll add a new categorical (factor) variable to be used for classification.

```
summary(Carseats$Sales)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   5.390   7.490   7.496   9.320  16.270
```

Name the new variable HighSales and define it as a factor with two values: 'Yes' if Sales value is greater than the 3rd quartile (9.32), and 'No' otherwise.

```
Carseats$HighSales <- ifelse(test = Carseats$Sales > 9.32, yes = 'Yes', no =
'No')
head(Carseats$HighSales)
```

```
## [1] "Yes" "Yes" "Yes" "No"  "No"  "Yes"
```

```
class(Carseats$HighSales)
```

```
## [1] "character"
```

We have created a character vector; now, we need to transform it into a factor variable:

```
Carseats$HighSales <- as.factor(Carseats$HighSales)
head(Carseats$HighSales)
```

```
## [1] Yes Yes Yes No  No  Yes
## Levels: No Yes
```

Let's check the distribution of the two values

```
table(Carseats$HighSales)
```

```
##
##  No Yes
## 301  99
```

```
prop.table(table(Carseats$HighSales))
```

```
##
##     No    Yes
## 0.7525 0.2475
```

So, in 75.25% of shops the company have not achieved high sales.

The objective is to develop a model that would be able to predict if the company will have a large sale in a certain shop. More precisely, the company is interested in spotting shops where high sales is not expected, so that it can take some interventions to improve the sales. This means that the class we are particularly interested in (the so-called 'positive class') is No.

Create train and test data sets

Remove the Sales variable as we do not need it any more

```
Carseats$Sales <- NULL
```

We should randomly select observations for training and testing. We should also assure that the distribution of the output variable (HigSales) is the same in both datasets (training and testing); this is referred to as *stratified partitioning*. To do that easily, we'll use appropriate functions from the **caret** package.

```
library(caret)
```

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

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

We'll use 80% of all the observations for training and the rest for testing

```
set.seed(10)
train.indices <- createDataPartition(Carseats$HighSales, # the variable
defining the class
                                     p = .80, # the proportion of
observations in the training set
                                     list = FALSE) # do not return the result
as a list (which is the default)
train.data <- Carseats[train.indices,]
test.data <- Carseats[-train.indices,]
```

We can check that the distribution of HighSales is really (roughly) the same in the two datasets

```
prop.table(table(train.data$HighSales))
```

```
##
##        No       Yes
## 0.7507788 0.2492212
```

```
prop.table(table(test.data$HighSales))
```

```
##
##        No       Yes
## 0.7594937 0.2405063
```

Create a prediction model using Decision Trees

We will use the **rpart** R package to build a decision tree. Note: this is just one of the available R packages for working with decision trees

```
library(rpart)
```

Build a tree using the *rpart* function and all the variables:

```
tree1 <- rpart(HighSales ~ ., data = train.data, method = "class")
```

Note the parameter *method*; it is set to the "class" value as we are building a classification tree; if we want to build a regression tree (to perform a regression task), we would set this parameter to 'anova'.

Print the model

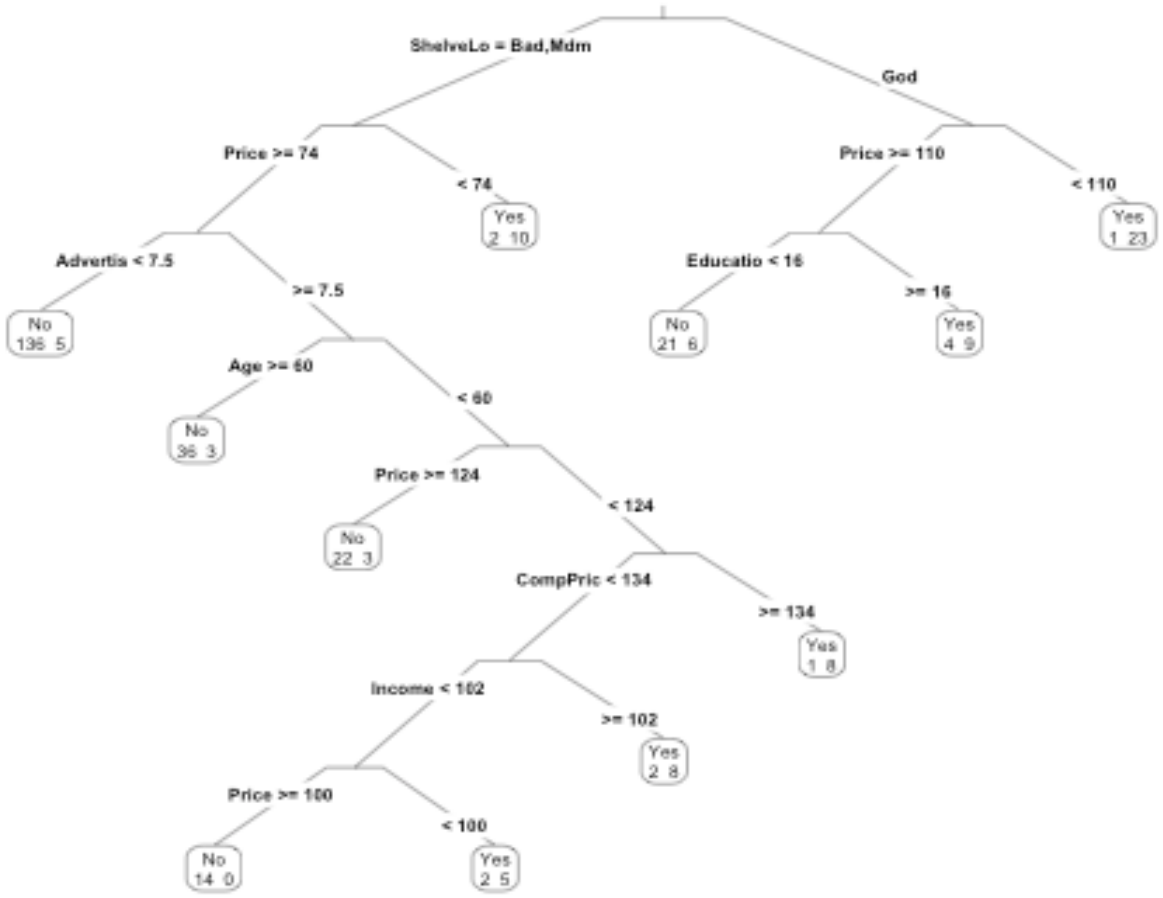
```
print(tree1)
```

```
## n= 321
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 321 80 No (0.75077882 0.24922118)
```

```
##     2) ShelveLoc=Bad,Medium 257 42 No (0.83657588 0.16342412)
##       4) Price>=74.5 245 32 No (0.86938776 0.13061224)
##         8) Advertising< 7.5 141  5 No (0.96453901 0.03546099) *
##         9) Advertising>=7.5 104 27 No (0.74038462 0.25961538)
##          18) Age>=60 39  3 No (0.92307692 0.07692308) *
##          19) Age< 60 65 24 No (0.63076923 0.36923077)
##            38) Price>=124.5 25  3 No (0.88000000 0.12000000) *
##            39) Price< 124.5 40 19 Yes (0.47500000 0.52500000)
##              78) CompPrice< 134.5 31 13 No (0.58064516 0.41935484)
##               156) Income< 102.5 21  5 No (0.76190476 0.23809524)
##                 312) Price>=99.5 14  0 No (1.00000000 0.00000000) *
##                 313) Price< 99.5 7  2 Yes (0.28571429 0.71428571) *
##               157) Income>=102.5 10  2 Yes (0.20000000 0.80000000) *
##              79) CompPrice>=134.5 9  1 Yes (0.11111111 0.88888889) *
##       5) Price< 74.5 12  2 Yes (0.16666667 0.83333333) *
##     3) ShelveLoc=Good 64 26 Yes (0.40625000 0.59375000)
##       6) Price>=109.5 40 15 No (0.62500000 0.37500000)
##        12) Education< 15.5 27  6 No (0.77777778 0.22222222) *
##        13) Education>=15.5 13  4 Yes (0.30769231 0.69230769) *
##       7) Price< 109.5 24  1 Yes (0.04166667 0.95833333) *
```

Let's plot the tree, to understand it better. To that end, we will use the **prp** function from the **rpart.plot** package:

```
library(rpart.plot)

prp(tree1, type = 3, extra = 1)
```



TASK: check the other options of the type and extra parametrs to see how they affect the visualization of the tree model

Observing the tree, we can see that only a couple of variables were used to build the model:

- ShelveLo - the quality of the shelving location for the car seats at a given site
- Price - price the company charges for car seats at a given site
- Advertise - local advertising budget for company at each location
- Age - average age of the local population
- CompPrice - price charged by competitor at each location

Let's evaluate our model on the test set

```
tree1.pred <- predict(object = tree1, newdata = test.data, type = "class")
```

Examine what the predictions look like

```
tree1.pred[1:10]
```

```
##   4   5   7  15  18  20  23  27  30  48
##  No  No  No Yes  No  No  No  No  No  No
## Levels: No Yes
```

To start evaluating the prediction quality of our model, we will first create the confusion matrix

```
tree1.cm <- table(true=test.data$HighSales, predicted=tree1.pred)
tree1.cm
```

```
##      predicted
## true  No Yes
##   No  56   4
##   Yes  9  10
```

Since we'll need to compute evaluation metrics couple of times, it's handy to have a function for that. The f. receives a confusion matrix, and returns a named vector with the values for accuracy, precision, recall, and F1-measure.

```
compute.eval.metrics <- function(cmatrix) {
  TP <- cmatrix[1,1] # true positive
  TN <- cmatrix[2,2] # true negative
  FP <- cmatrix[2,1] # false positive
  FN <- cmatrix[1,2] # false negative
  acc <- sum(diag(cmatrix)) / sum(cmatrix)
  precision <- TP / (TP + FP)
  recall <- TP / (TP + FN)
  F1 <- 2*precision*recall / (precision + recall)
  c(accuracy = acc, precision = precision, recall = recall, F1 = F1)
}
```

Now, we'll use the function to compute evaluation metrics for our tree model

```
tree1.eval <- compute.eval.metrics(tree1.cm)
tree1.eval
```

```
##  accuracy precision    recall        F1
## 0.8354430 0.8615385 0.9333333 0.8960000
```

Not bad...

The *rpart* function uses a number of parameters to control the growth of a tree. In the above call of the *rpart* f. we relied on the default values of those parameters. To inspect the parameters and their defaults, type:

```
?rpart.control
```

Let's now change some of these parameters to try to create a better model. Two parameters that are often considered important are:

- cp - the so-called *complexity parameter*. It regulates the splitting of nodes and growing of a tree by preventing splits that are deemed not important enough. In particular, those would be the splits that would not improve the fittness of the model by at least the *cp* value

- minsplit - minimum number of instances in a node for a split to be attempted at that node

We will decrease the values of both parameters to grow a larger tree:

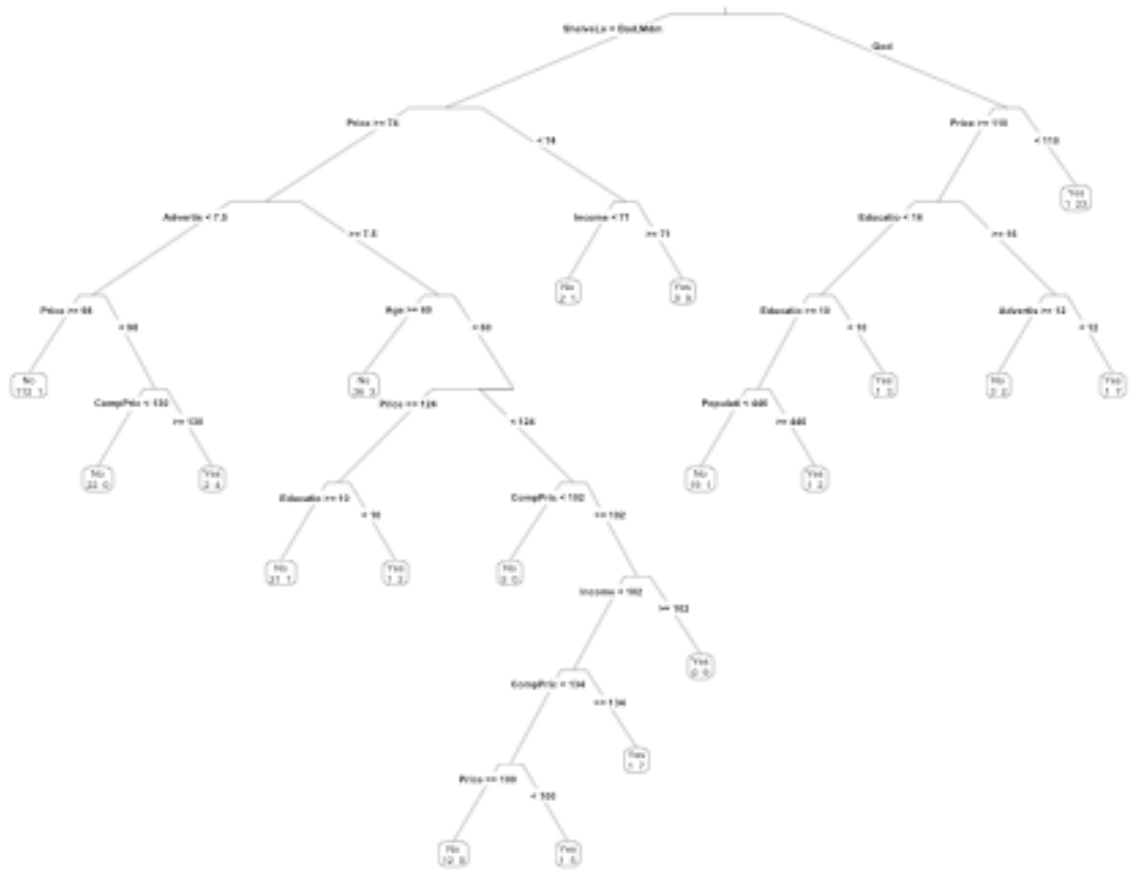
```
tree2 <- rpart(HighSales ~ ., data = train.data, method = "class",
                     control = rpart.control(minsplit = 10, cp = 0.001))
print(tree2)
```

```
## n= 321
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 321 80 No (0.750778816 0.249221184)
##     2) ShelveLoc=Bad,Medium 257 42 No (0.836575875 0.163424125)
##       4) Price>=74.5 245 32 No (0.869387755 0.130612245)
##         8) Advertising< 7.5 141  5 No (0.964539007 0.035460993)
##          16) Price>=97.5 113  1 No (0.991150442 0.008849558) *
##          17) Price< 97.5 28  4 No (0.857142857 0.142857143)
##            34) CompPrice< 129.5 22  0 No (1.000000000 0.000000000) *
##            35) CompPrice>=129.5 6  2 Yes (0.333333333 0.666666667) *
##         9) Advertising>=7.5 104 27 No (0.740384615 0.259615385)
##          18) Age>=60 39  3 No (0.923076923 0.076923077) *
##          19) Age< 60 65 24 No (0.630769231 0.369230769)
##            38) Price>=124.5 25  3 No (0.880000000 0.120000000)
##              76) Education>=10.5 22  1 No (0.954545455 0.045454545) *
##              77) Education< 10.5 3  1 Yes (0.333333333 0.666666667) *
##            39) Price< 124.5 40 19 Yes (0.475000000 0.525000000)
##              78) CompPrice< 102 5  0 No (1.000000000 0.000000000) *
##              79) CompPrice>=102 35 14 Yes (0.400000000 0.600000000)
##               158) Income< 102.5 26 12 No (0.538461538 0.461538462)
##                 316) CompPrice< 134.5 18  5 No (0.722222222 0.277777778)
##                   632) Price>=99.5 12  0 No (1.000000000 0.000000000) *
##                   633) Price< 99.5 6  1 Yes (0.166666667 0.833333333) *
##                 317) CompPrice>=134.5 8  1 Yes (0.125000000 0.875000000) *
##               159) Income>=102.5 9  0 Yes (0.000000000 1.000000000) *
##       5) Price< 74.5 12  2 Yes (0.166666667 0.833333333)
##        10) Income< 71 3  1 No (0.666666667 0.333333333) *
##        11) Income>=71 9  0 Yes (0.000000000 1.000000000) *
##     3) ShelveLoc=Good 64 26 Yes (0.406250000 0.593750000)
##       6) Price>=109.5 40 15 No (0.625000000 0.375000000)
##        12) Education< 15.5 27  6 No (0.777777778 0.222222222)
##          24) Education>=10.5 23  3 No (0.869565217 0.130434783)
##            48) Population< 445.5 20  1 No (0.950000000 0.050000000) *
##            49) Population>=445.5 3  1 Yes (0.333333333 0.666666667) *
##          25) Education< 10.5 4  1 Yes (0.250000000 0.750000000) *
##        13) Education>=15.5 13  4 Yes (0.307692308 0.692307692)
##          26) Advertising>=11.5 5  2 No (0.600000000 0.400000000) *
##          27) Advertising< 11.5 8  1 Yes (0.125000000 0.875000000) *
##       7) Price< 109.5 24  1 Yes (0.041666667 0.958333333) *
```

Obivously, we got a significantly larger tree.

Let's plot this tree:

```
prp(tree2, type = 3, extra = 1)
```



Is this larger tree better than the initial one (tree1)? To check that, we need to evaluate the 2nd tree on the test set

```
tree2.pred <- predict(tree2, newdata = test.data, type = "class")
```

Again, we'll create a confusion matrix

```
tree2.cm <- table(true=test.data$HighSales, predicted=tree2.pred)
tree2.cm
```

```
##       predicted
## true   No Yes
##   No   54   6
##   Yes   8  11
```

Next, we'll compute the evaluation metrics

```
tree2.eval <- compute.eval.metrics(tree2.cm)
tree2.eval
```

```
##  accuracy precision    recall        F1
## 0.8227848 0.8709677 0.9000000 0.8852459
```

Let's compare this model to the first one:

```
data.frame(rbind(tree1.eval, tree2.eval),
           row.names = c("tree 1", "tree 2"))

##         accuracy precision    recall        F1
## tree 1 0.8354430 0.8615385 0.9333333 0.8960000
## tree 2 0.8227848 0.8709677 0.9000000 0.8852459
```

Some metrics have improved, others became worse. So, our guess for the parameter values was not the best one.

Instead of relying on guessing, we should adopt a systematic way of examining the parameters values, looking for the optimal ones. An often applied approach is to cross-validate the model with a range of different values of parameters of interest. We will apply that approach here, focusing on the *cp* parameter since it is considered the most important parameter when growing trees with the *rpart* function.

For finding the optimal *cp* value through cross-validation we will use some handy functions from the **caret** package (it has already been loaded). Since these functions internally call cross-validation functions from the **e1071** package, we need to (install and) load that package.

```
# install.packages('e1071')
library(e1071)

# define cross-validation (cv) parameters; we'll do 10-fold cross-validation
numFolds = trainControl( method = "cv", number = 10 )
# then, define the range of the cp values to examine in the cross-validation
cpGrid = expand.grid( .cp = seq(0.001, to = 0.05, by = 0.001))
```

Perform parameter search through cross validation

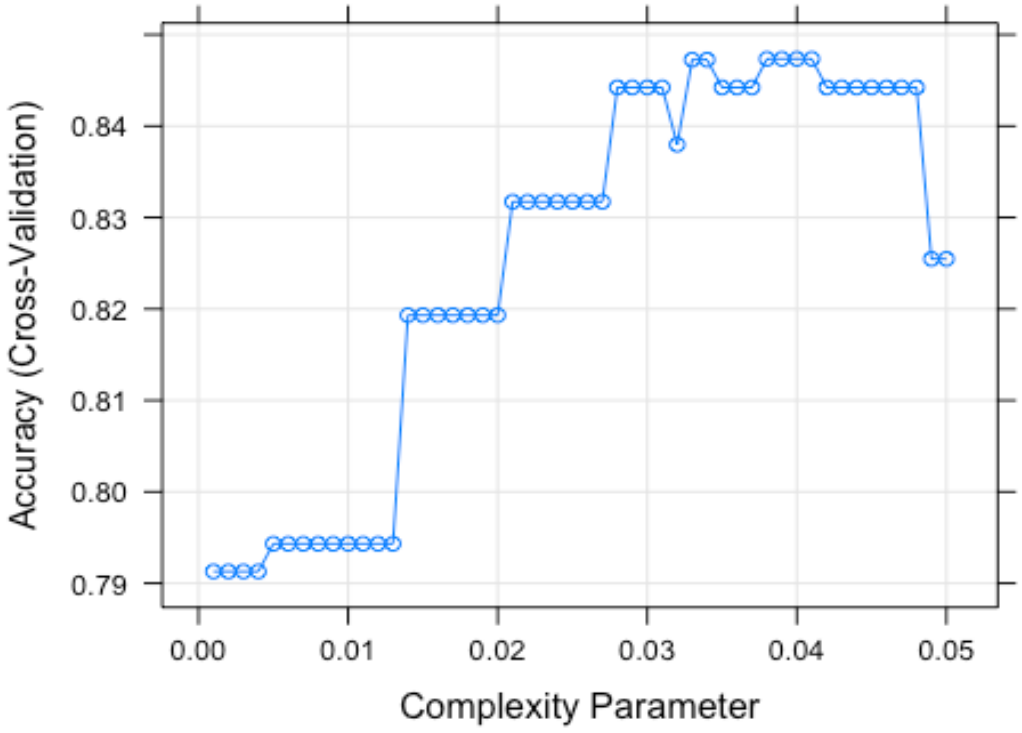
```
# since cross-validation is a probabilistic process, it is advisable to set
the seed, so that we can replicate the results
set.seed(10)
dt.cv <- train(HighSales ~ .,
               data = train.data,
               method = "rpart",
               control = rpart.control(minsplit = 10),
               trControl = numFolds,
               tuneGrid = cpGrid)
dt.cv

## CART
##
## 321 samples
##  10 predictor
##   2 classes: 'No', 'Yes'
```

```
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 289, 289, 289, 289, 289, 288, ...
## Resampling results across tuning parameters:
##
##   cp     Accuracy   Kappa
##   0.001  0.7912879  0.4241157
##   0.002  0.7912879  0.4241157
##   0.003  0.7912879  0.4241157
##   0.004  0.7912879  0.4241157
##   0.005  0.7943182  0.4300114
##   0.006  0.7943182  0.4300114
##   0.007  0.7943182  0.4300114
##   0.008  0.7943182  0.4300114
##   0.009  0.7943182  0.4300114
##   0.010  0.7943182  0.4300114
##   0.011  0.7943182  0.4300114
##   0.012  0.7943182  0.4300114
##   0.013  0.7943182  0.4300114
##   0.014  0.8193182  0.5080314
##   0.015  0.8193182  0.5080314
##   0.016  0.8193182  0.5080314
##   0.017  0.8193182  0.5080314
##   0.018  0.8193182  0.5080314
##   0.019  0.8193182  0.5080314
##   0.020  0.8193182  0.5080314
##   0.021  0.8317235  0.5228175
##   0.022  0.8317235  0.5228175
##   0.023  0.8317235  0.5228175
##   0.024  0.8317235  0.5228175
##   0.025  0.8317235  0.5228175
##   0.026  0.8317235  0.5228175
##   0.027  0.8317235  0.5228175
##   0.028  0.8442235  0.5619611
##   0.029  0.8442235  0.5619611
##   0.030  0.8442235  0.5619611
##   0.031  0.8442235  0.5619611
##   0.032  0.8379735  0.5392338
##   0.033  0.8472538  0.5505440
##   0.034  0.8472538  0.5505440
##   0.035  0.8442235  0.5393317
##   0.036  0.8442235  0.5393317
##   0.037  0.8442235  0.5393317
##   0.038  0.8473485  0.5374358
##   0.039  0.8473485  0.5374358
##   0.040  0.8473485  0.5374358
##   0.041  0.8473485  0.5374358
##   0.042  0.8442235  0.5364730
##   0.043  0.8442235  0.5364730
```

```
##   0.044  0.8442235  0.5364730
##   0.045  0.8442235  0.5364730
##   0.046  0.8442235  0.5364730
##   0.047  0.8442235  0.5364730
##   0.048  0.8442235  0.5364730
##   0.049  0.8254735  0.5008138
##   0.050  0.8254735  0.5008138
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.041.
```

Plot the results of parameter tunning:

```
plot(dt.cv)
```



So, we got the best value for the cp parameter: 0.041. Since it suggests a simpler model (smaller tree) than the previous one (tree2), we can **prune** our second tree using this cp value:

```
tree3 <- prune(tree2, cp = 0.041)
print(tree3)
```

```
## n= 321
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
```

```
##
##  1) root 321 80 No (0.75077882 0.24922118)
##    2) ShelveLoc=Bad,Medium 257 42 No (0.83657588 0.16342412)
##      4) Price>=74.5 245 32 No (0.86938776 0.13061224) *
##      5) Price< 74.5 12  2 Yes (0.16666667 0.83333333) *
##    3) ShelveLoc=Good 64 26 Yes (0.40625000 0.59375000)
##      6) Price>=109.5 40 15 No (0.62500000 0.37500000)
##       12) Education< 15.5 27  6 No (0.77777778 0.22222222) *
##       13) Education>=15.5 13  4 Yes (0.30769231 0.69230769) *
##      7) Price< 109.5 24  1 Yes (0.04166667 0.95833333) *
```

Create predictions and compute evaluation metrics for tree3:

```
tree3.pred <- predict(tree3, newdata = test.data, type = "class")
tree3.cm <- table(true = test.data$HighSales, predicted = tree3.pred)
tree3.cm
```

```
##      predicted
## true  No Yes
##   No  57   3
##   Yes 11   8
```

```
tree3.eval <- compute.eval.metrics(tree3.cm)
tree3.eval
```

```
##  accuracy precision    recall        F1
## 0.8227848 0.8382353 0.9500000 0.8906250
```

Let's compare all 3 models we have built so far

```
data.frame(rbind(tree1.eval, tree2.eval, tree3.eval),
           row.names = c(paste("tree", 1:3)))
```

```
##         accuracy precision    recall        F1
## tree 1 0.8354430 0.8615385 0.9333333 0.8960000
## tree 2 0.8227848 0.8709677 0.9000000 0.8852459
## tree 3 0.8227848 0.8382353 0.9500000 0.8906250
```

The 2nd and the 3rd model have the same accuracy, but differ in terms of precision and recall. To look for a better model, we might consider altering some other parameters. Another option is to reduce the number of variables that are used for model building.

TASK: create a new tree (tree4) by using only variables that proved relevant in the previous models (tree1, tree2, tree3). Evaluate the model on the test set and compare the evaluation metrics with those obtained for the previous three models.