Tree Methods

I added a good introductory article on tree methods by Stobl et al. It is in the classification folder.

Recursive Trees

The basic idea of a recursive tree is that it makes cuts along variables in order to find optimal classification or prediction.

I'll use the kyphosis data to illustrate the concept underlying recursive partitioning. The data is in the library rpart so I load it now. We will use rpart later in this worksheet to compute trees.

```
library(rpart)
data(kyphosis)
```

The next few slides illustrate what a tree algorithm is doing. So I have several slides that add one more feature to the previous slide.

We have two predictors Start and Age. We want to find if we can use the variables Start and Age to classify the presence or absence of Kyphosis. The frequency of presense or absence is

```
table(kyphosis$Kyphosis)
```

```
##
##  absent present
##      64      17
```

Here the dollar sign construction means IN.THE.DATA.FRAME$USE.THIS.VARIABLE, so I'm using the variable Kyphosis in the data frame kyphosis (yes, R is case sensitive).

Another way to refer to variables in a data set is to use the with() command. The logic of the with command can be illustrated by:

with(THESE.DATA, DO.THIS)

```
with(kyphosis, table(Kyphosis))
```

```
## Kyphosis
##  absent present
##      64      17
```

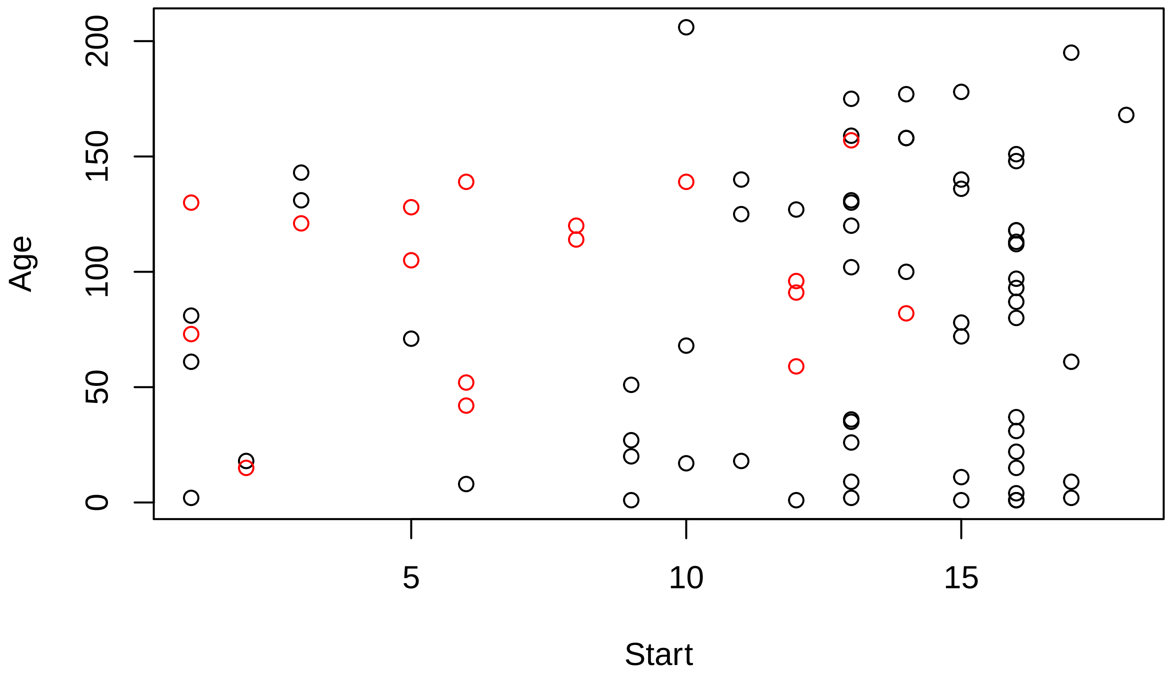
The difference between the first and second version of producting the table is that in the first version I directly specified the variable Kyphosis in the data frame kyphosis by typing kyphosis$Kyphosis. In the second version I used the with command so didn't have to type the data frame name kyphosis inside the table command because with allows me to say "with the data frame kyphosis, calculate the frequency table on the variable Kyphosis".

The with() command is useful if you have many data sets with common variable names so you don't want to use attach() and don't want to refer to individual variables in data frames like kyphosis$Start so the code is more readable.

This plot is a plot of two predictors with the variable we want to classify being the color of the dots.

We want to know if we can chop up this Start by Age variable space into smaller regions that are predominantly red or predominantly black.

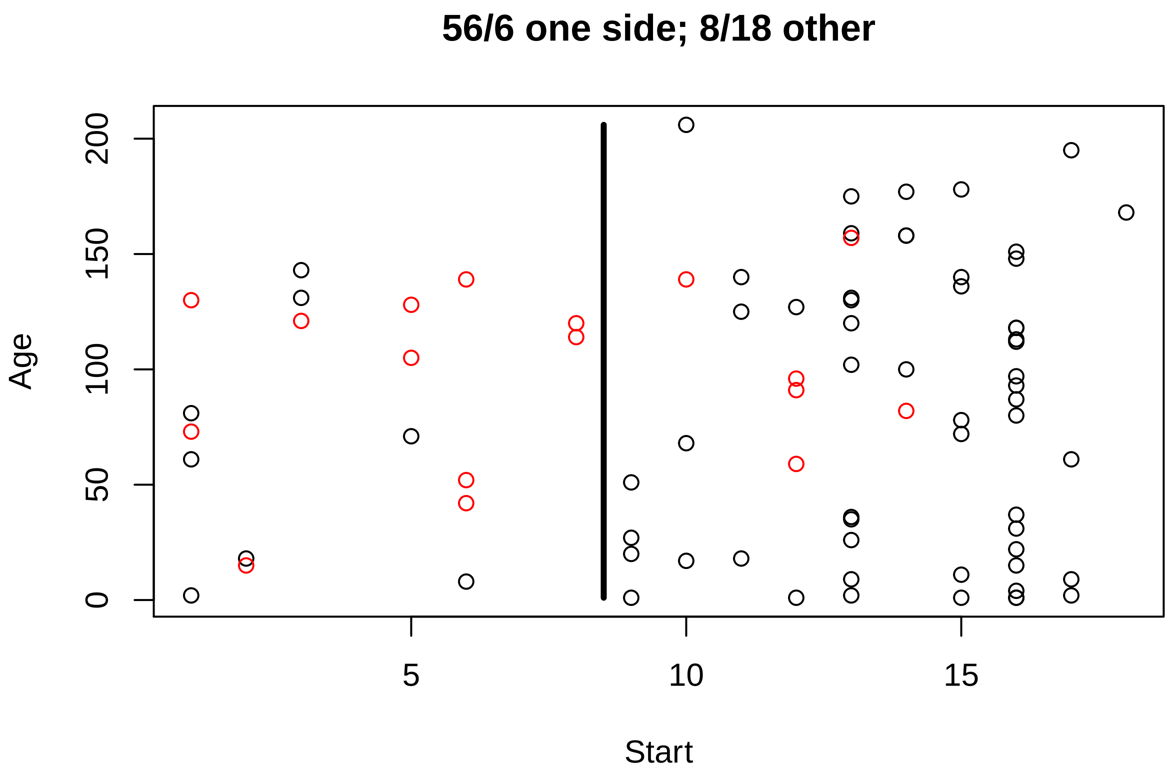
```r
with(kyphosis, plot(Start,Age,col=c(Kyphosis)))
```



First cut.

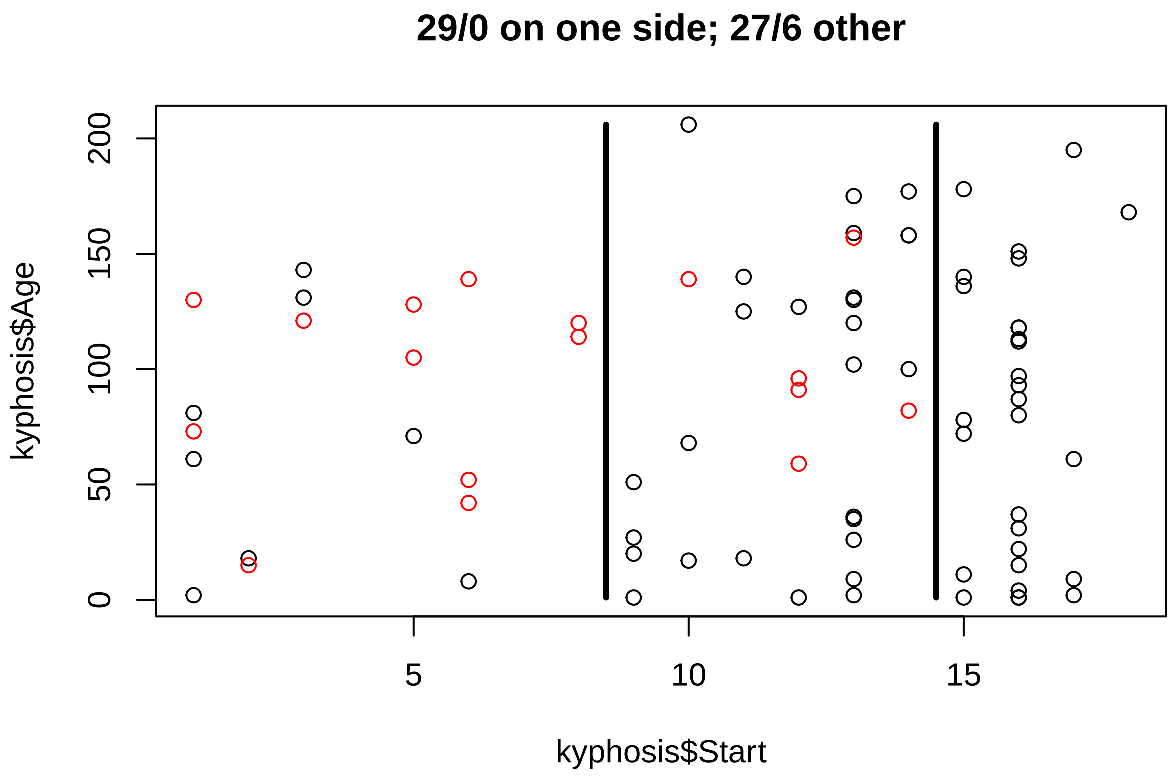
```r
with(kyphosis, plot(Start,Age,col=c(Kyphosis)))

segments(8.5,1,8.5,206,lwd=3)
title("56/6 one side; 8/18 other")
```



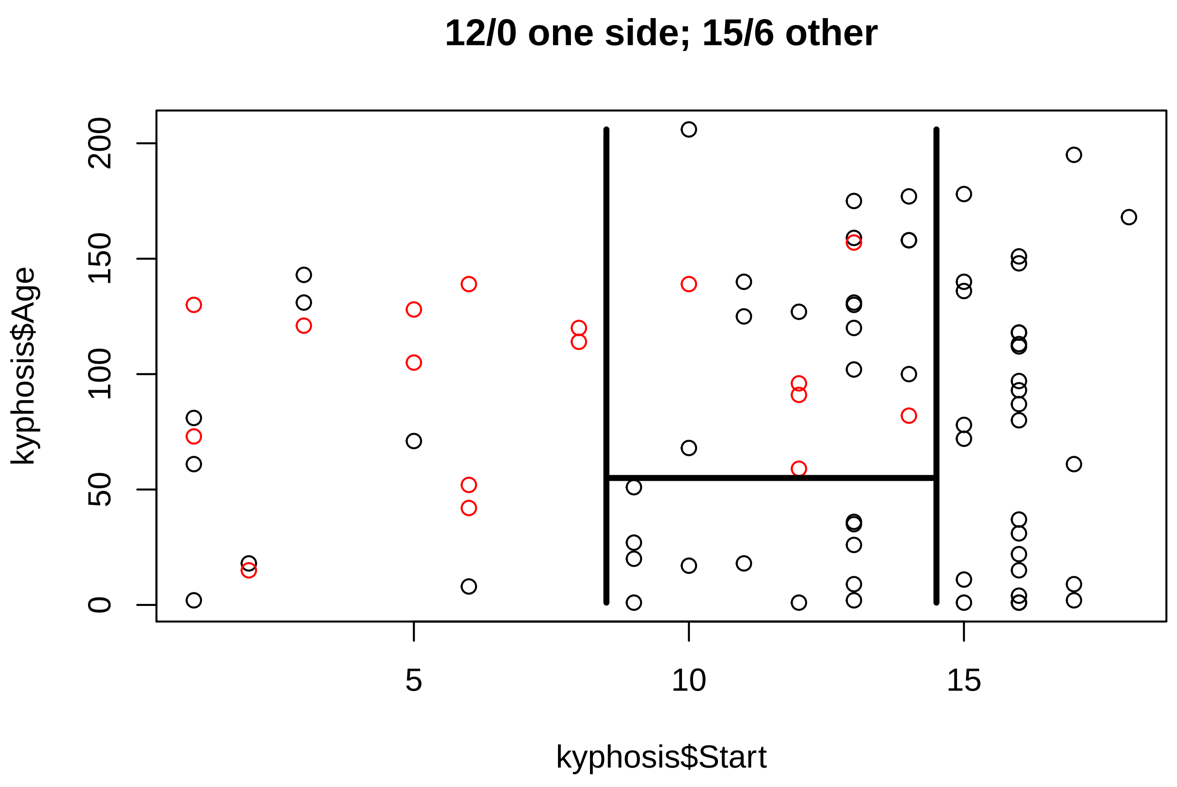
Second cut.

```
plot(kyphosis$Start,kyphosis$Age,col=c(kyphosis$Kyphosis))
segments(8.5,1,8.5,206,lwd=3)
segments(14.5,1,14.5,206,lwd=3)
title("29/0 on one side; 27/6 other")
```



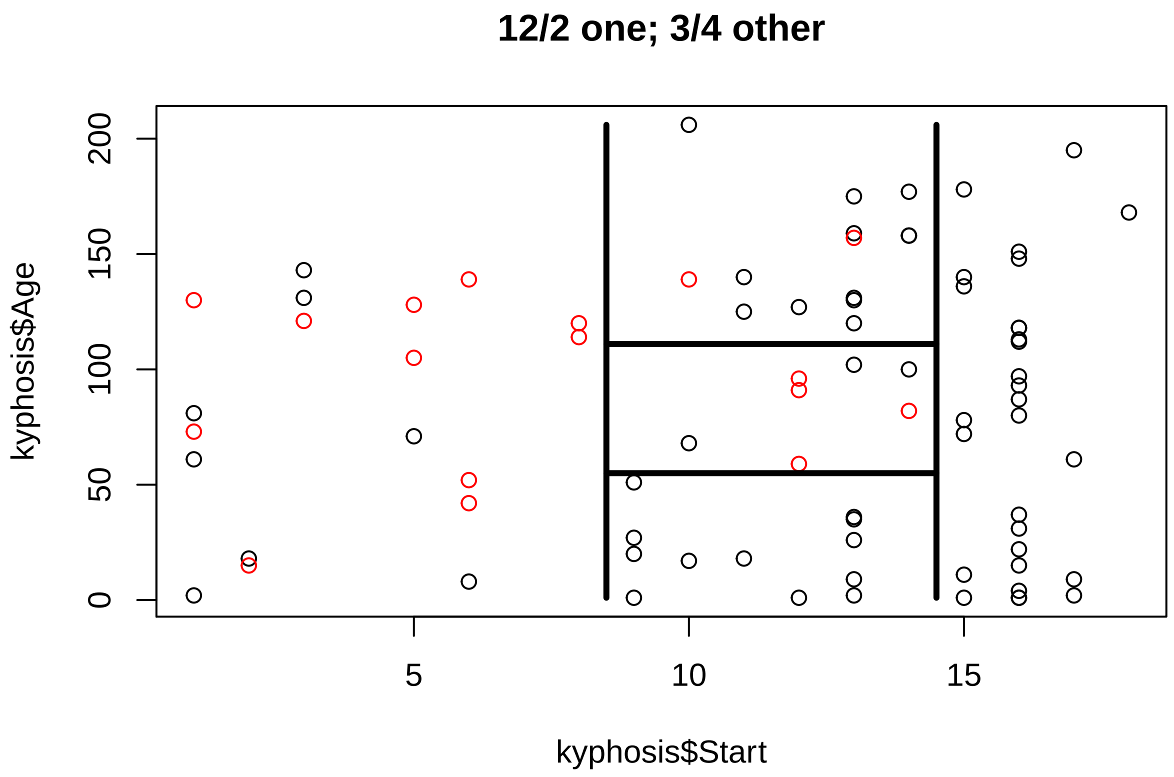
Third cut.

```
plot(kyphosis$Start,kyphosis$Age,col=c(kyphosis$Kyphosis))
segments(8.5,1,8.5,206,lwd=3)
segments(14.5,1,14.5,206,lwd=3)
segments(8.5,55,14.5,55,lwd=3)
title("12/0 one side; 15/6 other")
```



Fourth cut.

```
plot(kyphosis$Start,kyphosis$Age,col=c(kyphosis$Kyphosis))
segments(8.5,1,8.5,206,lwd=3)
segments(14.5,1,14.5,206,lwd=3)
segments(8.5,55,14.5,55,lwd=3)
segments(8.5,111,14.5,111,lwd=3)
title("12/2 one; 3/4 other")
```



This series of plots shows that the algorithm makes cuts in a space of variables to optimize occurence of a class within each rectangle.

Recursive Tree

Here is the recursive tree illustration for the same Kyphosis example. There are three predictors: Start, Age and Number. Kyphosis is a binary present/absent if spinal disorder was present after surgery.

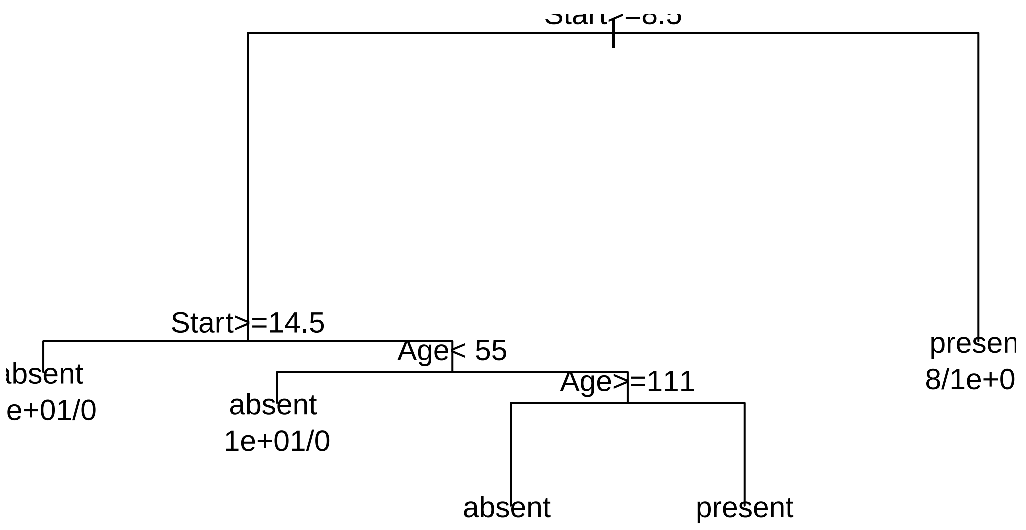
Look at first few rows of data set.

```
head(kyphosis)
```

```
##   Kyphosis Age Number Start
## 1   absent  71      3     5
## 2   absent 158      3    14
## 3  present 128      4     5
## 4   absent   2      5     1
## 5   absent   1      4    15
## 6   absent   1      2    16
```

Run the tree.

```
out.tree <- rpart(Kyphosis ~ Start + Age + Number, data=kyphosis)
plot(out.tree)
text(out.tree,use.n=T,cex=.95)
```



The tree suggests that Start and Age are the two key variables for dividing up the predictor space into smaller regions.

You can interpet the paths as a series of "if-then" statements. At the top of a node it gives the inequality that if true the branch is to the left, if it is not true, then the branch is to the right.

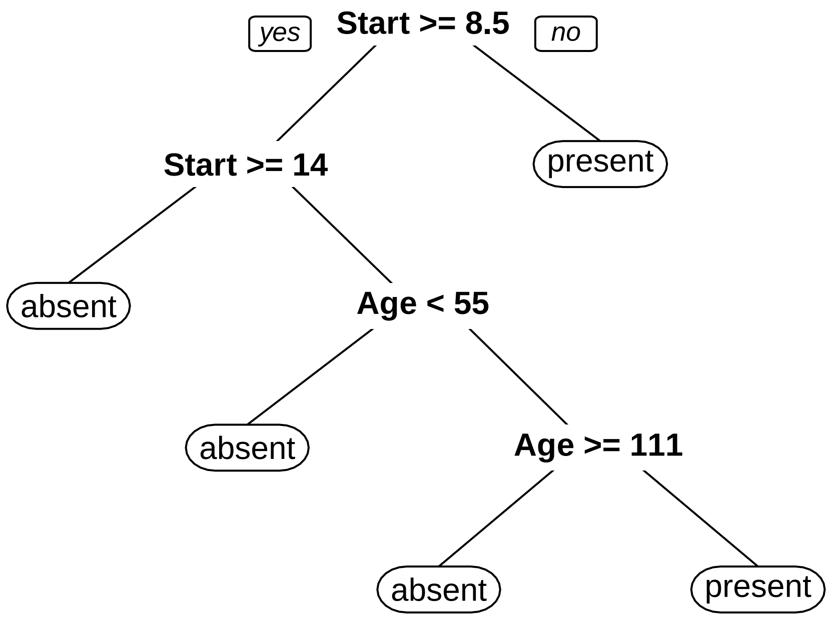
The object out.tree is a new kind of object compared to others we have seen, a tree object. It is useful to look at it so that you can understand the plotted tree, especially if text is cut off on the plot.

```
out.tree
```

```
## n= 81
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 81 17 absent (0.79012 0.20988)
##    2) Start>=8.5 62  6 absent (0.90323 0.09677)
##      4) Start>=14.5 29  0 absent (1.00000 0.00000) *
##      5) Start< 14.5 33  6 absent (0.81818 0.18182)
##       10) Age< 55 12  0 absent (1.00000 0.00000) *
##       11) Age>=55 21  6 absent (0.71429 0.28571)
##         22) Age>=111 14  2 absent (0.85714 0.14286) *
##         23) Age< 111 7  3 present (0.42857 0.57143) *
##    3) Start< 8.5 19  8 present (0.42105 0.57895) *
```

The library rpart.plot gives many options for customizing the graph of the tree. I will switch to that plot for the rest of these notes

```
library(rpart.plot)
prp(out.tree)
```



Prune the tree

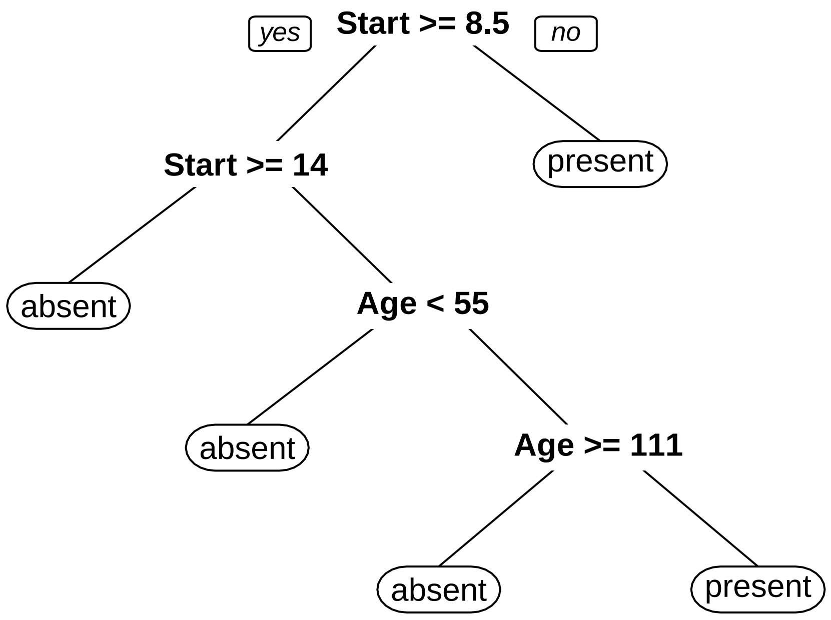
We can prune a tree by defining additional measures such as complexity, relative error, cross-validation error, and error variability. Those measures can help identify a location to chop the tree.

Usually we use the xerror (10-fold cross validation error) to see which branch produces the smallest xerror, and then we take the cp parameter of that row to give to the tree pruning function. For example,

```
printcp(out.tree)
```

```
##
## Classification tree:
## rpart(formula = Kyphosis ~ Start + Age + Number, data = kyphosis)
##
## Variables actually used in tree construction:
## [1] Age   Start
##
## Root node error: 17/81 = 0.21
##
## n= 81
##
##     CP nsplit rel error xerror xstd
## 1 0.18      0      1.00      1 0.22
## 2 0.02      1      0.82      1 0.22
## 3 0.01      4      0.76      1 0.22
```

```
out.prune <- prune.rpart(out.tree,cp=.019)
prp(out.prune)
```



The output of printcr also tells us how many splits there are at each level of complexity, as rows increase so does the number of splits. The top most row is always no split at all, just the raw data.

When there is a tie you can try both cp lines to see what emerges. In this example

```
printcp(out.tree)
```

```
##
## Classification tree:
## rpart(formula = Kyphosis ~ Start + Age + Number, data = kyphosis)
##
## Variables actually used in tree construction:
## [1] Age   Start
##
## Root node error: 17/81 = 0.21
##
## n= 81
##
##     CP nsplit rel error xerror xstd
## 1 0.18      0      1.00      1 0.22
## 2 0.02      1      0.82      1 0.22
## 3 0.01      4      0.76      1 0.22
```

```
out.prune <- prune.rpart(out.tree,cp=.18)
out.prune
```

```
## n= 81
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 81 17 absent (0.7901 0.2099) *
```

```r
plot(out.prune)
```

```
## Error: fit is not a tree, just a root
```

Here we get an error when we try to plot a tree with 0 splits because the resulting pruned tree is just the sample with no tree. The output of the prunned tree compares with the frequency of each category in the sample (81 subjects of which 17 are absent). The frequencies are

```r
table(kyphosis$Kyphosis)
```

```
##
##  absent present
##      64      17
```

This example with just a few variables didn't produce any pruning, but I wanted to do a simple example.

Come back to our life satisfaction variable from the Social Diagnostic survey.

```r
#need to add more variables to make this interesting
library(foreign)
poland.data <- read.spss("../warsaw/data/DS313IND(RV2).sav",to.data.frame=T)
age13 <- poland.data[,16]
income.c13 <- poland.data[,265]
income.c13[income.c13=="NAP"] <- NA
income.c13 <- factor(income.c13)
lifesat <- poland.data[,217]
lifesat[c(lifesat)<=3] <- NA
#rather than a factor, use ordered so lavaan recognizes it as ordinal
lifesat <- ordered(lifesat)

#remember levels of lifesat
levels(lifesat)
```

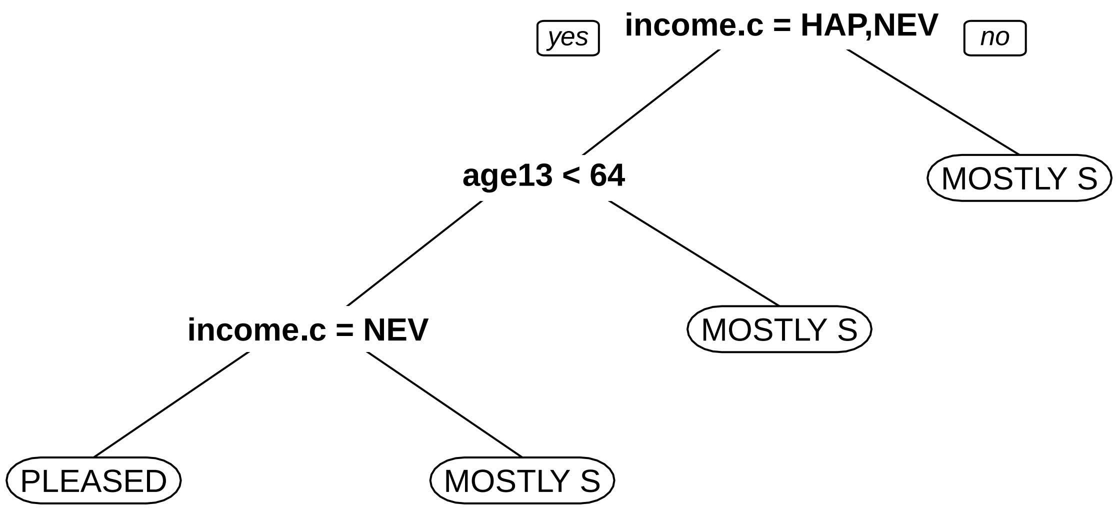
```
## [1] "DELIGHTED"           "PLEASED"             "MOSTLY SATISFIED"
## [4] "MIXED"               "MOSTLY DISSATISFIED" "UNHAPPY"
## [7] "TERRIBLE"
```

```r
#do tree
out.tree <- rpart(lifesat ~ age13 + income.c13, data=kyphosis)
printcp(out.tree)
```

```
##
## Classification tree:
## rpart(formula = lifesat ~ age13 + income.c13, data = kyphosis)
##
## Variables actually used in tree construction:
## [1] age13      income.c13
##
## Root node error: 553/920 = 0.6
##
## n=920 (1 observation deleted due to missingness)
```

```
##
##      CP nsplit rel error xerror  xstd
## 1 0.017      0      1.00   1.00 0.027
## 2 0.010      3      0.95   0.98 0.027
```

```
out.tree.pr <- prune.rpart(out.tree,cp=.01)
prp(out.tree.pr)
```



```
out.tree.pr
```

```
## n=920 (1 observation deleted due to missingness)
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 920 553 MOSTLY SATISFIED (0.022 0.35 0.4 0.18 0.045 0.0065 0.0022)
##   2) income.c13=HAPPENED,NEVER 815 491 MOSTLY SATISFIED (0.022 0.37 0.4 0.17 0.034 0.0061 0.0012)
##     4) age13< 64.5 547 329 MOSTLY SATISFIED (0.029 0.39 0.4 0.14 0.033 0.0073 0.0018)
##       8) income.c13=NEVER 208 108 PLEASED (0.034 0.48 0.35 0.11 0.029 0.0048 0) *
##       9) income.c13=HAPPENED 339 193 MOSTLY SATISFIED (0.027 0.34 0.43 0.16 0.035 0.0088 0.0029) *
##     5) age13>=64.5 268 162 MOSTLY SATISFIED (0.0075 0.33 0.4 0.23 0.037 0.0037 0) *
##   3) income.c13=OFTEN 105  62 MOSTLY SATISFIED (0.019 0.17 0.41 0.26 0.12 0.0095 0.0095) *
```

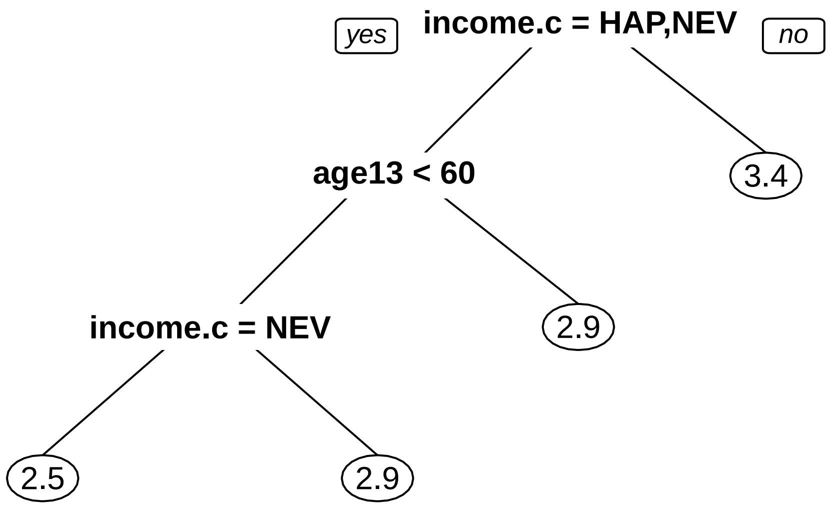
We can also do a tree on quantitative variables. Just for illustration let's convert life satisfaction from an ordinal factor into a numerical variable and redo the tree.

```
lifesat.numeric <- c(lifesat)
out.tree <- rpart(lifesat.numeric ~ age13 + income.c13, data=kyphosis)
printcp(out.tree)
```

```
##
## Regression tree:
## rpart(formula = lifesat.numeric ~ age13 + income.c13, data = kyphosis)
##
## Variables actually used in tree construction:
```

```
## [1] age13      income.c13
##
## Root node error: 806/920 = 0.88
##
## n=920 (1 observation deleted due to missingness)
##
##      CP nsplit rel error xerror  xstd
## 1 0.031      0      1.00   1.00 0.054
## 2 0.010      1      0.97   0.98 0.053
## 3 0.010      3      0.95   0.99 0.053
```

```
out.tree.pr <- prune.rpart(out.tree,cp=.01)
prp(out.tree.pr)
```



```
out.tree.pr
```

```
## n=920 (1 observation deleted due to missingness)
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 920 805.60 2.904
##   2) income.c13=HAPPENED,NEVER 815 664.50 2.845
##     4) age13< 60.5 446 368.30 2.765
##       8) income.c13=NEVER 142  95.25 2.542 *
##       9) income.c13=HAPPENED 304 262.70 2.868 *
##     5) age13>=60.5 369 289.80 2.943 *
##   3) income.c13=OFTEN 105 116.20 3.362 *
```

Now we get means at the ends of each path rather than frequencies. So if you must treat Likert scales as numbers you can do that if you wish. The rpart() can treat different variables like factors, ordinal factors, numerical data, and counts. See the help for more information.

Anyways, this gives you a flavor of how to proceed with recursive trees. You can make this example more interesting by getting more predictors from the data set (cleaning up those data so all the missing data are coded correct, labels are clean, etc) and seeing if you get more interesting trees.

Trees with tests of significance

There is a different approach to trees that uses significance testing to decide on whether or not to take another cut, and one can apply Bonferroni increasing the penalty for each additional cut. This helps minimize Type I error but it is not based on cross-validation error like the xerror we used within the rpart library.

Unfortunately, the conditional tree command ctree() does not play well with missing data at the response, so we need to omit missing data from the data set. For simplicity I'll remove all missing data, but one only need to remove if missing on the response variable.

```
data.ctree <- data.frame(lifesat, age13, income.c13)
dim(data.ctree)
```

```
## [1] 921   3
```

```
data.ctree <- na.omit(data.ctree)
dim(data.ctree)
```

```
## [1] 871   3
```

```
#we lost 50 subjects due to missing anywhere in the 3 variables we
#consider
library(party)
```

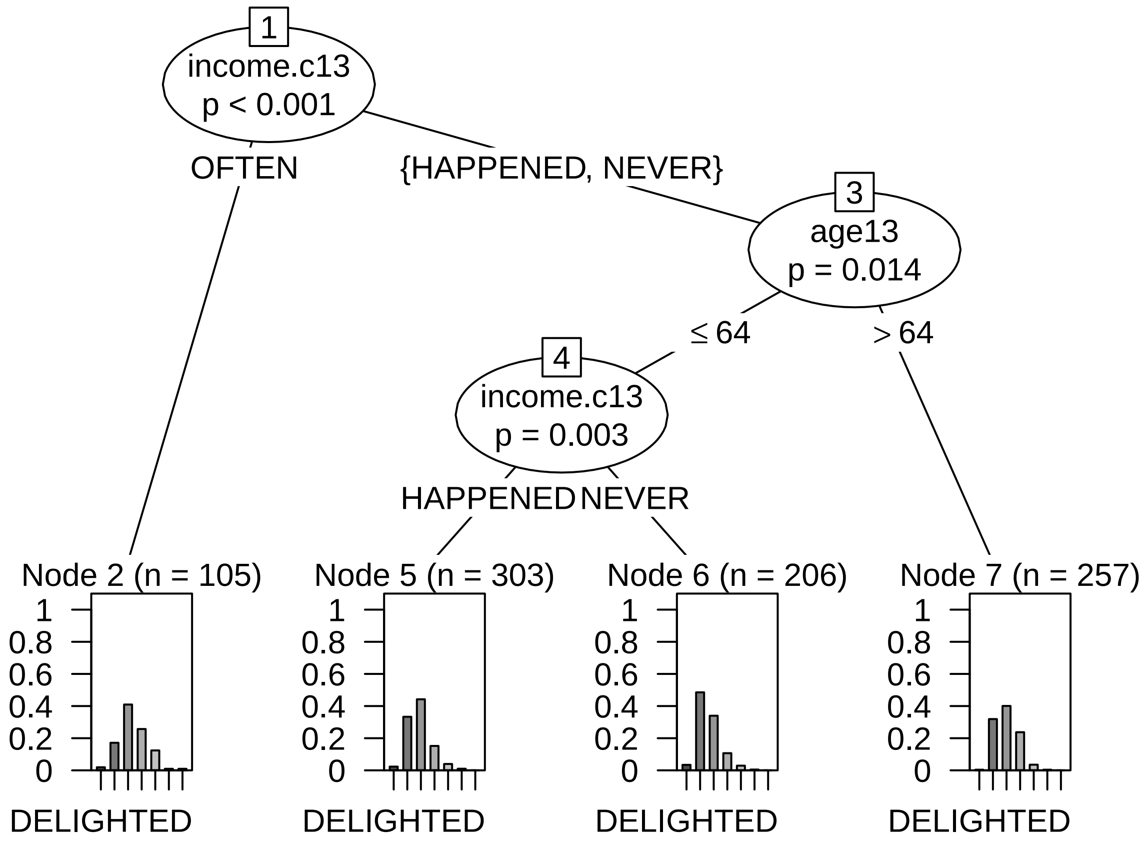
```
## Warning: package 'party' was built under R version 3.1.1
```

```
## Loading required package: grid
## Loading required package: zoo
##
## Attaching package: 'zoo'
##
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
##
## Loading required package: sandwich
```

```
## Warning: package 'sandwich' was built under R version 3.1.1
```

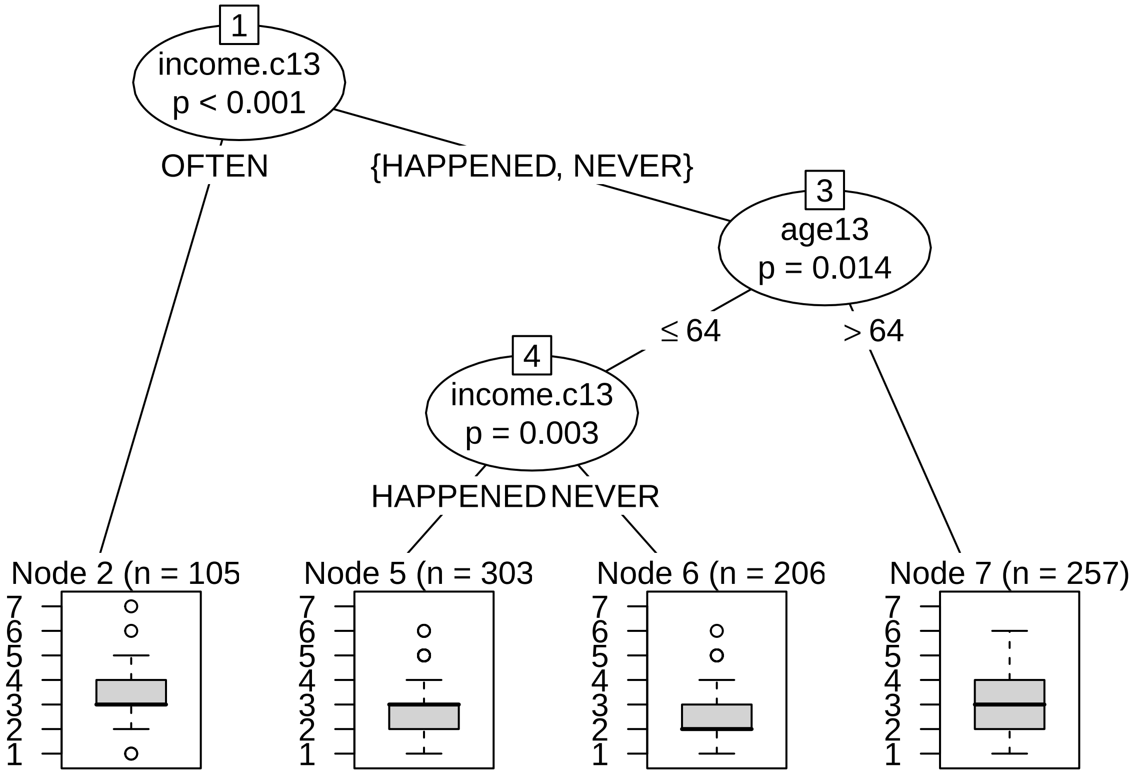
```
## Loading required package: strucchange
## Loading required package: modeltools
## Loading required package: stats4
```

```
out.tree <- ctree(lifesat ~ age13 + income.c13, data=data.ctree)
plot(out.tree)
```



Can also do it with the numerical version of lifesat.

```
data.ctree$lifesat <- as.numeric(data.ctree$lifesat)
out.tree <- ctree(lifesat ~ age13 + income.c13, data=data.ctree)
plot(out.tree)
```



Random Forest

Cross-validation is about taking a group of subjects to build the model and a different group of subjects to test the model (like a holdout or a different sample).

Random Forest randomly samples variables from the set of predictors you give it. For each set of variables it fits a tree. Does this many times for different sets of variables, and calculates a measure of importance for each variable. Basically, how often does that appear as a predictor variable when it was included in the set.

We can try it in the Polish Social Diagnosis Survey.

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.1.1
```

```
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

```
#set up data for random forest

data.for.rf <- data.frame(lifesat.numeric, poland.data[,141:150])


out.randomF <- randomForest(lifesat.numeric ~ .,data=na.omit(data.for.rf),type="class",ntrees=2000)
```

```
## Warning: The response has five or fewer unique values.  Are you sure you
## want to do regression?
```

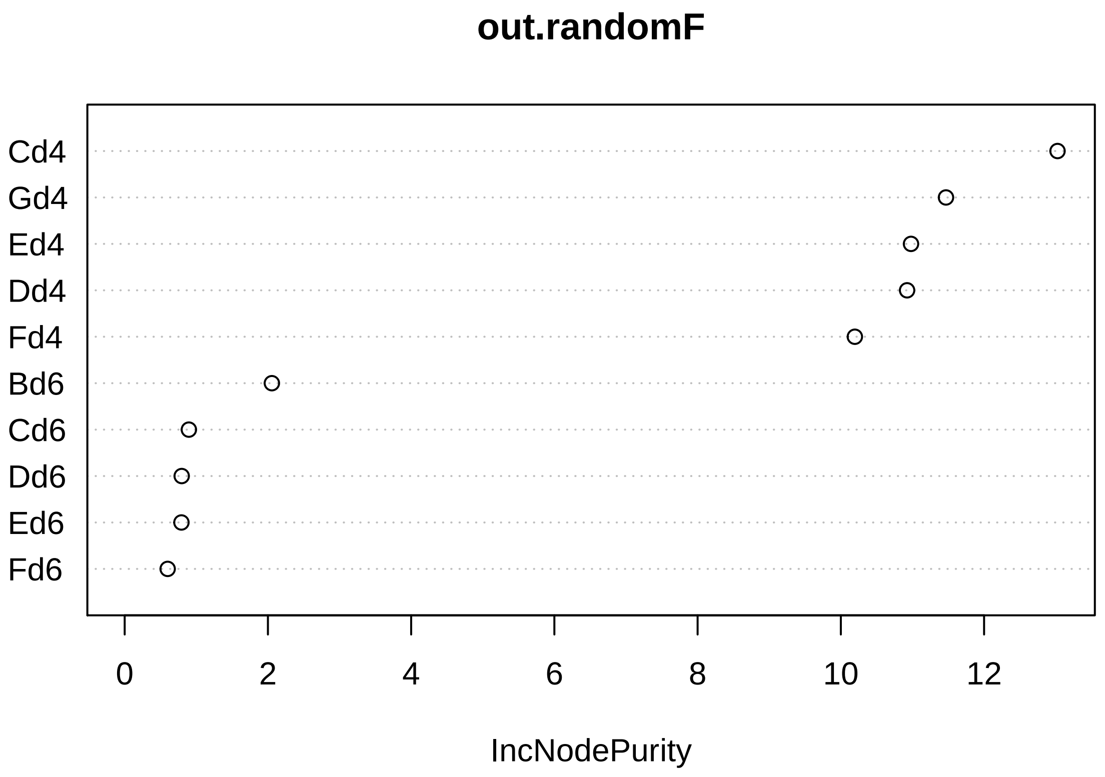
```r
print(out.randomF)
```

```
##
## Call:
##  randomForest(formula = lifesat.numeric ~ ., data = na.omit(data.for.rf),      type = "class", ntree
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 0.7049
##                     % Var explained: -9.07
```

```r
importance(out.randomF)
```

```
##     IncNodePurity
## Cd4       13.0255
## Dd4       10.9247
## Ed4       10.9795
## Fd4       10.1956
## Gd4       11.4679
## Bd6        2.0552
## Cd6        0.8964
## Dd6        0.7963
## Ed6        0.7925
## Fd6        0.6002
```

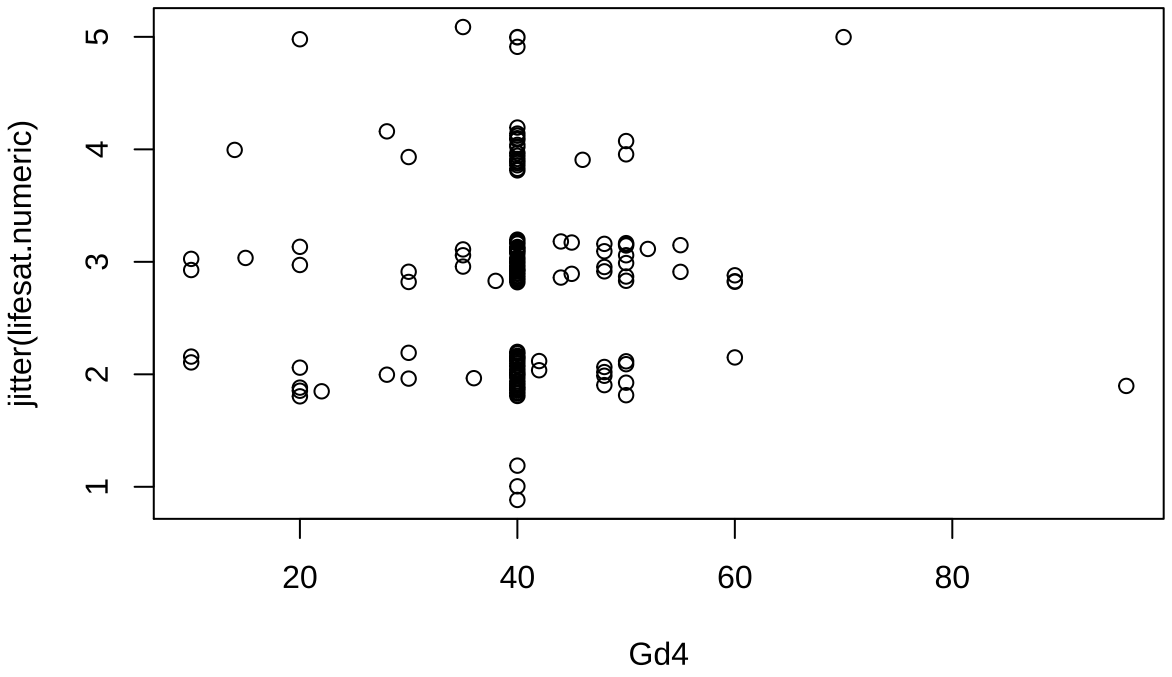
```r
varImpPlot(out.randomF)
```



The top variables here were "How many hours did you work in the last 7 days". Cd4 answer in 2005 and Gd45 is answer in 2013. Remember we're using the life satisfaction question from 2013. So, having looked at just the predictors 141-150 in the data set we find that, no surprise, the number of hours you work relates to your life satisfaction. The items about whether you are in full time employment (the variables ending in d6) were not as good predictors as the number of hours (ending in d4). So it isn't whether you work full-time but how many hours. This needs to be examined more carefully.

I used the jitter command on the lifesat variable to be able to distinguish points that would be on top of each other. Jitter just adds a little noise to both the y values of each point.

```
with(na.omit(data.for.rf), plot(Gd4, jitter(lifesat.numeric)))
```



There is one outlier. A person who works 96 hours and is PLEASED with life as a whole.

```
max(data.for.rf$Gd4,na.rm=T)
```

```
## [1] 96
```

This suggests hypotheses we can test, such as does the number of hours worked change over each of the survey waves? For people who increased (decreased) work hours, over that period, did their life satisfaction decrease (increase)? We, of course, need to look more carefully at the scatterplot between these hours worked variables and life satisfaction to see the direction of the pattern. All we know from the random forest procedure is that some variables are consistently good predictors of life satisfaction across different sampling of variables. For this analysis I limited it to just a few so you should add more variables as there may be other variables that do a even better than the d4 group of variables.

You can see that a purely exploratory method finds patterns. You should not stop there. The point of exploratory methods is to give you new ideas. But those new ideas then have to be tested with more traditional methods, new data should be collected to test new hypotheses, etc.

There is a place for exploratory methods in your toolbox.