~~~
---
title: "CART Activity"
author: " "
output: html_document
---

There are (at least) three packages for building classification and regression trees (CART
models) in R.  The one we'll use is `rpart`.  Install this package now, and load it using the
chunk below.


**Data set:** The file CEREAL.CSV contains some nutritional information and grocery shelf
location for 77 breakfast cereals. The data include cereal name, cereal manufacturer (`mfr2`)
(G=General Mills, K=Kellogg's, O=other), `type` (H=hot, C=cold), several nutritional
variables, the `shelf` location in a grocery store (1,2 or 3 for bottom, middle or top), and
a Consumer Reports `rating` of the cereal.  For the rating variable, the higher the number,
the healthier the cereal is to eat.  We would like to model the Consumer Reports rating of
the cereal. The data set is posted on Moodle; upload it in the chunk below.
```{r message=FALSE}

```



A. One Numerical Predictor
----------------------------
First, we will use `sugars` as our sole predictor variable.

1. Make a scatterplot of rating as a function of sugar content.  Explain why a linear
regression model would not be a good idea here.


***
#### The function to fit a regression tree is `rpart()`.  It works similarly to `lm()` or
`glm()`, but has a couple of additional arguments:
- `method = "anova"` specifies that this is a regression tree **(`method="class"` would
specify a classification tree)**
- `minsplit = ` specifies the minimum number of values in a node for it to be considered for
splitting.  The default is 20, which means that if a node has < 20 values, it can not be
split.  If you want to override this, and consider splitting smaller nodes, you can specify
another value.
- `minbucket = ` specifies the minimum size of a terminal node.  The default is minsplit/3.
(For example `minsplit=9` will allow for the splitting of nodes as small as 9, which means
you could have terminal nodes as small as 3.)
- `cp = ` specifies the complexity parameter; smaller values mean you're willing to have more
complexity in the tree.  The default is 0.01.

***

2. **Fit a regression tree** with rating as the response and sugars as the predictor.  (For
now, stick with the default minsplit and minbucket values.)  Call this tree `sugar1`.


3. **Plot the tree diagram** using the code below.
```{r}
# plot(sugar1,uniform=T,branch=0.6,margin=0.2) #branch specifies the angle of the branches in
the plot
# text(sugar1,use.n=T,cex=0.8) #cex reduces the size of the text so it's more readable
```

4. Compare this tree to the scatterplot in #1.  Do the splits seem reasonable to you?


***
#### Other details about this tree
~~~

~~~
- Type `sugar1` to get additional information about this tree.  Notice that this tells you
the following information about *each node*: node number, where the split is, how many values
are in the node, the SSE ("deviance"), the mean y value, and whether the node is terminal.
This is essentially the same information that is plotted in the tree diagram.

- `printcp(sugar1)` or `sugar1$cptable` gives you tree statistics about this tree and each
tree of smaller size.

- The code below will plot two graphs: one of the actual and cross-validated $R^2$, and one
of the cross-validated error.
```{r}
#par(mfrow=c(1,2)) #tells RMarkdown to print two graphs side-by-side
#rsq.rpart(sugar1) #plots the 2 graphs, and outputs the same information as printcp(sugar1)
```

***

5. Even if you think the 3 nodes are a good choice, force the regression tree to have more
complexity by setting cp=0.00001.  Call this tree `sugar2`.


6. How many terminal nodes does `sugar2` have?  How small is its smallest node?  Draw the
tree diagram below.


7. Consider all three of the methods for choosing the "best" tree that we discussed last
week.  Based on each one of these methods, how many splits would you choose for this data?


8. You can automatically **prune** the tree back to that number of splits with the function
`prune()`:
```{r}
#pruned.sugar <- prune(sugar2,cp=) #set cp equal to the cp value of the tree you're choosing
```
(This code is especially useful when you have a *large* number of splits that you're trying
to prune back.)

Notice that `pruned.sugar` is just a tree object like any other, so you can perform the same
tasks and apply the same code to it as we did to `sugar1` and `sugar1` in parts 1 - 7.

B. Many Predictor Variables
----------------------------
Now, we will (potentially) use a large number of variables to predict rating.  We have no
pre-existing notions about which variables will be useful and which won't be useful, so we
want to include all the variables in the partitioning algorithm, except `name` and `mfr`
(we'll use `mfr2` instead).  Remember: rpart will decide whether a variable is useful, and
ignore the variables that aren't!

1. Fit a full model using all the variables as possible predictors, by using a dot after the
tilde in the formula argument.  (We'll delete the first 2 columns in the data statement.)
```{r}
#all1 <- rpart(rating~.,data=cereal[,3:17],method="anova")
```

2. Make a tree diagram of this model.  How many terminal nodes does it have?


3. Consider all three of the methods for choosing the "best" tree that we discussed last
week.  Based on each one of these methods, how many splits would you choose?


4. Prune the tree back to however many splits you chose in #3 and show the tree diagram
below.
~~~

~~~
5. Your answer to #3 *may* depend on how many leaves you have in your "original" tree, since
the values of `xerror` are not constant as more leaves are added.  The tree below allows
splits on nodes of size 10 (overriding the default of size 20).  Thus, there are more leaves
in this tree.
```{r}
#all2 <- rpart(rating~.,data=cereal[,3:17],method="anova",minsplit=10)
```

Look at the tree statistics in `printcp(all2)` and `rsq.rpart(all2)` and notice that if you
had fit this tree first (before all1), you *may* have made a different decision about the
"best" tree size (depending on your choosing criterion).  The split points in that "best"
tree may also be  different! ... Remember, we have no guarantee we're getting the "best" tree
or the "right" leaves!

Feel free to play around with changing minsplit and/or the cp value, to see what kind of
trees you get.
~~~