WEEK 5: WORKING WITH VARIABLES

THOMAS ELLIOTT

1. Factors

One of the data storage types I've mentioned before but haven't discussed yet is the factor. Factor is a special data type R uses to store categorical variables. Factors are automatically treated as categorical variables and expanded into indicators when used in regression, they have special `summary` functions that generate a frequency table of factor levels, and it makes it easier to do certain things with factor variables. However, getting used to factors can be a little tricky, so let me go over how they work.

Factors consist of two pieces. The first are the labels - these function exactly like value labels do in Stata. They are stored as text in a special attribute of the factor and are associated with a value starting from 1 and going through the number of labels. The actual values stored for each observation in the variable are a number corresponding to the label for that observation. You create a factor variable with the `factor()` function. By default, the `factor()` function will use the values of the variable as the labels for the levels. For example, below I create a character vector called sex, in which values are either "male" or "female" and then create a factor vector using this data.

```
> sex<-c("male","male","female","male","female","female")
> sex<-factor(sex)
> sex
[1] male   male   female male   female female
Levels: female male
> unclass(sex)
[1] 2 2 1 2 1 1
attr(,"levels")
[1] "female" "male"
```

Some things to notice: factor does not care about order of the variables and by default the labels are created in alphabetical order. If the order of the labels do matter, you can use the `ordered=TRUE` argument (see the help file for more information). The underlying values stored are numbers corresponding to a value label, in the example above 1 corresponds to female and 2 corresponds to male. The above example is fairly straightforward, but let's see what happens if we create a factor variable from a numeric measure:

```
> x<-c(0,6,12,3,5,99)
> x<-factor(x)
> x
[1] 0  6  12 3  5  99
```

Date: February 10, 2016.

```
Levels: 0 3 5 6 12 99
> unclass(x)
[1] 1 4 5 2 3 6
attr(,"levels")
[1] "0"  "3"  "5"  "6"  "12" "99"
```

Again, the values of the original x vector are used as the labels for the values, and the labels are created in alphabetical order. Notice, also, that the underlying values in the factor variable are not the same as the original values of x. Also notice that the labels are stored as text, even though they are all numeric. When you use boolean expressions on a factor variable, the boolean tests against the value labels, not the underlying values, and numeric values are coerced to characters automatically:

```
> x==6
[1] FALSE  TRUE FALSE FALSE FALSE FALSE
> x=="6"
[1] FALSE  TRUE FALSE FALSE FALSE FALSE
> x==2
[1] FALSE FALSE FALSE FALSE FALSE FALSE
```

Finally, it is possible to supply arguments to factor to define the different levels expected and the labels for those levels. For example, let's say you have a variable for the type of operating system a user uses, with categories for 1=Windows, 2=Mac, 3=Linux, and 4=Other, but in the dataset you don't observe Linux users (perhaps you extracted a subsample that didn't include Linux users). You could still tell R that Linux is a possible level for the variable:

```
> OS<-c(1,1,1,2,4,1,2,1,1,4,4,4,2,2,2,1,4,2,1)
> OS<-factor(OS,levels=c(1,2,3,4),labels=c("Windows","Mac","Linux","Other"))
> summary(OS)
Windows     Mac   Linux   Other
      8       6       0       5
> OS
 [1] Windows Windows Windows Mac     Other   Windows Mac     Windows Windows Other   Oth
[12] Other   Mac     Mac     Mac     Windows Other   Mac     Windows
Levels: Windows Mac Linux Other
```

1.1. **Ordered Factors.** Regular factor variables assume that the variable is purely categorical - that the ordering of the levels does not matter. However, there may be instances in which the order is meaningful, for example measures of class. You can use `ordered()` in the same way you use `factor()` but the order of the levels will matter. For example:

```
> ses<-c("low","high","mid","mid","high","mid","mid","low","low","mid")
> ses.f<-ordered(ses,levels=c("low","mid","high"))
> ses.f
 [1] low  high mid  mid  high mid  mid  low  low  mid
Levels: low < mid < high
```

In most cases, ordered and factored variables are treated similarly. You can also create an ordered variable by suppling the `ordered=TRUE` argument to the `factor()` function.

1.2. **Creating Dummy Variables From Factors.** A continuing theme is that data management and manipulation is often easier in Stata than base R, a fact that kept me doing a lot of data cleaning in Stata and then exporting to R when I first started using R. Creating dummy variables from factor variables is one of those things that are easier in Stata than in R. One reason is that it is often less important to create the dummy variables, as R will automatically expand factor variables were appropriate. While there are many, imperfect, work-arounds to create dummies for all levels in a factor, I would recommend simply using boolean expressions to explicitly create dummies for each level you need a dummy for. Remember that when R forces a boolean into a numeric, FALSE gets evaluated as 0 and TRUE is evaluated as 1. Thus:

```
> (OS=="Windows")*1
 [1] 1 1 1 0 0 1 0 1 1 0 0 0 0 0 0 1 0 0 1
```

1.3. **Composite Categorical Variables.** We can combine two factor variables to create a third variable that takes on a unique value for every possible combination of the original two variables. In the example below, I pull in the included mtcars dataset, which contains information about cars reviewed by Motor Trend in 1974. The original dataset contains all numeric variables, but we can convert the cyl (cylinders) and gear variables into factors and then combine them to create a composite variable.

```
> mtcars$cyl<-factor(mtcars$cyl)
> mtcars$gear<-factor(mtcars$gear)
> mtcars$cyl.gear<-interaction(mtcars$cyl,mtcars$gear)
> mtcars[1:10,c("cyl","gear","cyl.gear")]
                  cyl gear cyl.gear
Mazda RX4           6    4      6.4
Mazda RX4 Wag       6    4      6.4
Datsun 710          4    4      4.4
Hornet 4 Drive      6    3      6.3
Hornet Sportabout   8    3      8.3
Valiant             6    3      6.3
Duster 360          8    3      8.3
Merc 240D           4    4      4.4
Merc 230            4    4      4.4
Merc 280            6    4      6.4
> summary(mtcars$cyl.gear)
4.3 6.3 8.3 4.4 6.4 8.4 4.5 6.5 8.5
  1   2  12   8   4   0   2   1   2
```

The function doing the heavy lifting here is `interaction()`, which creates a new factor variable, with levels for each unique combination of the two original factors. The new labels are, by default, the labels from the original factors separated by a "." This can be changed with the `sep="."` argument.

2. Recode

Recoding variables in base R is a bit more complicated than Stata, though this is more because R does not have a built in function to duplicate the succinctness of Stata's `recode` function.

2.1. **Categories from Continuous.** We'll use `mtcars` as our example data again. The dataset contains a miles per gallon variable. Let's say we want to recode this into a categorical variable with values "low" <= 18, 18 < "mid" <= 25, and "high" > 25. We can do this with the `cut()` function:

```
> mtcars$mpg.cat<-cut(mtcars$mpg,breaks=c(-Inf,18,25,Inf),
  labels=c("Low","Mid","High"))
> mtcars[1:10,c("mpg","mpg.cat")]
                   mpg mpg.cat
Mazda RX4         21.0     Mid
Mazda RX4 Wag     21.0     Mid
Datsun 710        22.8     Mid
Hornet 4 Drive    21.4     Mid
Hornet Sportabout 18.7     Mid
Valiant           18.1     Mid
Duster 360        14.3     Low
Merc 240D         24.4     Mid
Merc 230          22.8     Mid
Merc 280          19.2     Mid
```

The function takes as arguments a vector of values that will be recoded, a vector of cut-points, and a vector of text labels for the new categories. Note that by default the cut-points are open on the left and closed on the right: (L,H]. This can be modified with an optional argument. Breaks can also be passed a single value, indicating the number of groups to break the data into (the groups will be roughly equal in size).