Handout #5

Lesson 4: Data types and classes, modeling with non-numeric data

An object in R is assigned a "class" attribute, which describes the object and determines which functions are applicable to it. For example, suppose we create the vector

```
x<-c(1:20)
> class(x)
[1] "integer"
> is.integer(x)
[1] TRUE
> is.numeric(x)
[1] TRUE
```

The function `class()` reports the assigned class of an object. To test whether an object belongs to a given class, the function `is.[class]()` tests the object an returns a "TRUE" or "FALSE" value. In this case, we see the the class "integer" in a subset of the class "numeric."

```
> x1=is.numeric(x)
> x1
[1] TRUE
> class(x1)
[1] "logical"
```

TRUE/FALSE objects are assigned to the class "logical." Suppose we want to test whether elements of x are greater than 5:

```
> x2=x>5
> x2
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
[10]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[19]  TRUE  TRUE
> sum(x2)
[1] 15
```

For the class "logical," numerical operations still work. The object is simply converted to a numeric vector, with FALSE -> 0 and TRUE -> 1.

To convert objects from one type to another, we can use the function `as.[class]()`. For example:

```
> as.numeric(x2)
 [1] 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

The class "character" applies to strings of letters or numbers, and numeric functions have no meaning in this setting. For example:

```
> y=c("A","B","C","D")
> class(y)
[1] "character"
> as.numeric(y)
[1] NA NA NA NA
```

There's no obvious way to assign numeric values to the letters, so the function just returns a vector of "NA" values. However, consider the following:

```
> x=c(1:4)
> x=as.character(x)
> x
[1] "1" "2" "3" "4"
> as.numeric(x)
[1] 1 2 3 4
```

In this case, the mapping from characters to numbers was easily done.

The "factor" class

This class deserves some special attention. The factor class assigns levels to the elements of a vector, which may be treated as ordered or unordered. In ANOVA models, any non-numeric variable is treated as a factor, which means that coefficients are fit based on the levels assigned to the variable.

Look at the following example:

```
> data(chickwts)
> names(chickwts)
[1] "weight" "feed"
> chickwts[1:5,]
   weight      feed
1     179 horsebean
2     160 horsebean
3     136 horsebean
4     227 horsebean
5     217 horsebean
> class(chickwts$weight)
[1] "numeric"
> class(chickwts$feed)
[1] "factor"
>summary(chickwts$feed)
   casein horsebean   linseed  meatmeal   soybean
       12        10        12        11        14
sunflower
```

```
       12
> levels(chickwts$feed)
[1] "casein"    "horsebean" "linseed"   "meatmeal"
[5] "soybean"   "sunflower"
> is.ordered(chickwts$feed)
[1] FALSE
```

This variable has 6 unordered levels, one corresponding to each feed type. Suppose we wanted to convert this to a numeric variable for some reason. Look at what happens:

```
> as.numeric(chickwts$feed)
 [1] 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 5 5 5 5 5
[28] 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 4 4 4 4 4 4
[55] 4 4 4 4 4 1 1 1 1 1 1 1 1 1 1 1 1
```

Although the factor is unordered, there is still an arbitrary ordering (in this case, alphabetical) which is assigned to the levels, and the number returned for each element of the vector corresponds to its level. This is usually not a problem, unless we're working with variables that have a numeric interpretation. For example, look at the following:

```
> x=c("1","3","5","7","3")
> x
[1] "1" "3" "5" "7" "3"
> xf=factor(x)
> xf
[1] 1 3 5 7 3
Levels: 1 3 5 7
> xfn=as.numeric(xf)
> xfn
[1] 1 2 3 4 2
```

How could we have prevented this from happening? There are two options:

(1) Convert the factor variable to a character vector, then convert the result to a numeric vector:

```
> xfc=as.character(xf)
> xfc
[1] "1" "3" "5" "7" "3"
> as.numeric(xfc)
[1] 1 3 5 7 3
```

(in one command, you could type "as.numeric(as.character(xf))")

(2) Convert the levels of the factor to their numeric values, and apply this to the vector:

```
> as.numeric(levels(xf))[xf]
[1] 1 3 5 7 3
```

Functions for analyzing factor data

The function `split(x, f)` separates a vector **x** into groups defined by a factor **f**. For example, for the "chickwts" dataset, we can separate the weights by each feed type:

```
>attach(chickwts)
>split(weight,feed)
>lapply(split(weight,feed),summary)
>plot(weight~feed,data=chickwts)
```

Suppose we want to find a linear model that will predict weight as a function of feed type. Knowing that the least-squared model will fit the mean value for each type, which coefficients might we expect?

```
>lapply(split(weight,feed),mean)
```

Let's fit the model:

```
>chickmod=lm(weight~feed,data=chickwts)
>chickmod$coef
```

Why don't the coefficients match up? When we fit a model for a factor variable, R uses a contrast matrix to report the coefficients. The default is the "treatment" contrast, in which one level is treated as a base and the other coefficients are relative to the base:

```
> contrasts(feed)
          horsebean linseed meatmeal soybean sunflower
casein            0       0        0       0         0
horsebean         1       0        0       0         0
linseed           0       1        0       0         0
meatmeal          0       0        1       0         0
soybean           0       0        0       1         0
sunflower         0       0        0       0         1
```

In this case, "casein" is the base, and is represented by the intercept of the model.
There are other contrast options, "sum", "helmert", and "poly."

```
> contrasts(C(feed,sum))
          [,1] [,2] [,3] [,4] [,5]
casein       1    0    0    0    0
horsebean    0    1    0    0    0
linseed      0    0    1    0    0
meatmeal     0    0    0    1    0
soybean      0    0    0    0    1
sunflower   -1   -1   -1   -1   -1
```

```
> contrasts(C(feed,poly))
               .L         .Q         .C         ^4          ^5
[1,] -0.5976143  0.5455447 -0.3726780  0.1889822 -0.06299408
[2,] -0.3585686 -0.1091089  0.5217492 -0.5669467  0.31497039
[3,] -0.1195229 -0.4364358  0.2981424  0.3779645 -0.62994079
[4,]  0.1195229 -0.4364358 -0.2981424  0.3779645  0.62994079
[5,]  0.3585686 -0.1091089 -0.5217492 -0.5669467 -0.31497039
[6,]  0.5976143  0.5455447  0.3726780  0.1889822  0.06299408

> contrasts(C(feed,helmert))
          [,1] [,2] [,3] [,4] [,5]
casein      -1   -1   -1   -1   -1
horsebean    1   -1   -1   -1   -1
linseed      0    2   -1   -1   -1
meatmeal     0    0    3   -1   -1
soybean      0    0    0    4   -1
sunflower    0    0    0    0    5
```

To fit a model using a different contrast option than the default, use the "contrasts" option in the model command.

Exercise:
Use the following command to fit a model to the chickwts dataset using "sum" contrasts:
lm(formula = weight ~ feed, data = chickwts, contrasts = list(feed = "contr.sum"))
Look at the coefficients. How can you recover the true means for each feed type?

Now let's look at a slightly more complicated dataset. Load the dataset "ToothGrowth" into your workspace. This gives tooth length measurements for guinea pigs at each of three dose levels of Vitamin C (0.5, 1, and 2 mg) with each of two delivery methods (orange juice or ascorbic acid), with 10 guinea pigs in each group.

```
> TG=ToothGrowth
> names(TG)
[1] "len"  "supp" "dose"
> attach(TG)
> class(dose)
[1] "numeric"
> class(supp)
[1] "factor"
> plot(len~supp,data=TG)
```

Does there appear to be a difference between the two delivery methods?

Since "dose" is a numeric variable, we should probably look at scatterplots:

```
>plot(dose,len)
>library(lattice)
>xyplot(len~dose|supp,data=TG)
```
What just happened?

Let's fit a simple model:

```
> lenmod=lm(len~supp+dose,data=TG)
> summary(lenmod)
```

Look at the residual plots. Does this seem like the right way to model the data?

Although "dose" is a numeric variable, it would make more sense to treat it as a factor for our purposes:

```
> lenmod.f=lm(len~supp+factor(dose),data=TG)
```

Does this improve the model? Why? Finally, we might want to consider whether or not there are interaction effect between "supp" and "dose" that we haven't captured in our model. Try this:

```
>interaction.plot(dose,supp,len,fixed=T)
```

Let's fit the model and compare it to our additive model:

```
>lenmod.fI=lm(len~supp*factor(dose),data=TG)
>anova(lenmod.fI,lenmod.f)
```