4.7: Factors

Okay, it's time to start introducing some of the data types that are somewhat more specific to statistics. If you remember back to Chapter 2, when we assign numbers to possible outcomes, these numbers can mean quite different things depending on what kind of variable we are attempting to measure. In particular, we commonly make the distinction between *nominal*, *ordinal*, *interval* and *ratio* scale data. How do we capture this distinction in R? Currently, we only seem to have a single numeric data type. That's probably not going to be enough, is it?

A little thought suggests that the numeric variable class in R is perfectly suited for capturing ratio scale data. For instance, if I were to measure response time (RT) for five different events, I could store the data in R like this:

```
RT <- c(342, 401, 590, 391, 554)
```

where the data here are measured in milliseconds, as is conventional in the psychological literature. It's perfectly sensible to talk about "twice the response time", 2×RT, or the "response time plus 1 second", RT+1000, and so both of the following are perfectly reasonable things for R to do:

```
2 * RT
```

```
## [1]  684  802 1180  782 1108
```

```
RT + 1000
```

```
## [1] 1342 1401 1590 1391 1554
```

And to a lesser extent, the "numeric" class is okay for interval scale data, as long as we remember that multiplication and division aren't terribly interesting for these sorts of variables. That is, if my IQ score is 110 and yours is 120, it's perfectly okay to say that you're 10 IQ points smarter than me[58], but it's not okay to say that I'm only 92% as smart as you are, because intelligence doesn't have a natural zero.[59] We might even be willing to tolerate the use of numeric variables to represent ordinal scale variables, such as those that you typically get when you ask people to rank order items (e.g., like we do in Australian elections), though as we will see R actually has a built in tool for representing ordinal data (see Section 7.11.2) However, when it comes to nominal scale data, it becomes completely unacceptable, because almost all of the "usual" rules for what you're allowed to do with numbers don't apply to nominal scale data. It is for this reason that R has ***factors***.

4.7.1 Introducing factors

Suppose, I was doing a study in which people could belong to one of three different treatment conditions. Each group of people were asked to complete the same task, but each group received different instructions. Not surprisingly, I might want to have a variable that keeps track of what group people were in. So I could type in something like this

```
group <- c(1,1,1,2,2,2,3,3,3)
```

so that `group[i]` contains the group membership of the `i` -th person in my study. Clearly, this is numeric data, but equally obviously this is a nominal scale variable. There's no sense in which "group 1" plus "group 2" equals "group 3", but nevertheless if I try to do that, R won't stop me because it doesn't know any better:

```
group + 2
```

```
## [1] 3 3 3 4 4 4 5 5 5
```

Apparently R seems to think that it's allowed to invent "group 4" and "group 5", even though they didn't actually exist. Unfortunately, R is too stupid to know any better: it thinks that `3` is an ordinary number in this context, so it sees no problem in calculating `3 + 2`. But since *we're* not that stupid, we'd like to stop R from doing this. We can do so by instructing R to treat `group` as a factor. This is easy to do using the `as.factor()` function.[60]

```
group <- as.factor(group)
group
```

```
## [1] 1 1 1 2 2 2 3 3 3
## Levels: 1 2 3
```

It looks more or less the same as before (though it's not immediately obvious what all that `Levels` rubbish is about), but if we ask R to tell us what the class of the `group` variable is now, it's clear that it has done what we asked:

```
class(group)
```

```
## [1] "factor"
```

Neat. Better yet, now that I've converted `group` to a factor, look what happens when I try to add 2 to it:

```
group + 2
```

```
## Warning in Ops.factor(group, 2): '+' not meaningful for factors
```

```
## [1] NA NA NA NA NA NA NA NA NA
```

This time even R is smart enough to know that I'm being an idiot, so it tells me off and then produces a vector of missing values. (i.e., `NA`: see Section 4.6.1).

4.7.2 Labelling the factor levels

I have a confession to make. My memory is not infinite in capacity; and it seems to be getting worse as I get older. So it kind of annoys me when I get data sets where there's a nominal scale variable called `gender`, with two levels corresponding to males and females. But when I go to print out the variable I get something like this:

```
gender
```

```
## [1] 1 1 1 1 1 2 2 2 2
## Levels: 1 2
```

Okaaaay. That's not helpful at all, and it makes me very sad. Which number corresponds to the males and which one corresponds to the females? Wouldn't it be nice if R could actually keep track of this? It's way too hard to remember which number corresponds to which gender. And besides, the problem that this causes is much more serious than a single sad nerd… because R has no way of knowing that the `1`s in the `group` variable are a very different kind of thing to the `1`s in the `gender` variable. So if I try to ask which elements of the `group` variable are equal to the corresponding elements in `gender`, R thinks this is totally kosher, and gives me this:

```
group == gender
```

```
## Error in Ops.factor(group, gender): level sets of factors are different
```

Well, that's … especially stupid.[61] The problem here is that R is very literal minded. Even though you've declared both `group` and `gender` to be factors, it still assumes that a `1` is a `1` no matter which variable it appears in.

To fix both of these problems (my memory problem, and R's infuriating literal interpretations), what we need to do is assign meaningful labels to the different *levels* of each factor. We can do that like this:

```
levels(group) <- c("group 1", "group 2", "group 3")
print(group)
```

```
## [1] group 1 group 1 group 1 group 2 group 2 group 2 group 3 group 3 group 3
## Levels: group 1 group 2 group 3
```

```
levels(gender) <- c("male", "female")
print(gender)
```

```
## [1] male   male   male   male   male   female female female female
## Levels: male female
```

That's much easier on the eye, and better yet, R is smart enough to know that `"female"` is not equal to `"group 2"` , so now when I try to ask which group memberships are "equal to" the gender of the corresponding person,

```
group == gender
```

```
## Error in Ops.factor(group, gender): level sets of factors are different
```

R correctly tells me that I'm an idiot.

4.7.3 Moving on…

Factors are very useful things, and we'll use them a lot in this book: they're *the* main way to represent a nominal scale variable. And there are lots of nominal scale variables out there. I'll talk more about factors in 7.11.2, but for now you know enough to be able to get started.