4.8: Data frames

It's now time to go back and deal with the somewhat confusing thing that happened in Section **??** when we tried to open up a CSV file. Apparently we succeeded in loading the data, but it came to us in a very odd looking format. At the time, I told you that this was a ***data frame***. Now I'd better explain what that means.

4.8.1 Introducing data frames

In order to understand why R has created this funny thing called a data frame, it helps to try to see what problem it solves. So let's go back to the little scenario that I used when introducing factors in Section 4.7. In that section I recorded the `group` and `gender` for all 9 participants in my study. Let's also suppose I recorded their ages and their `score` on "Dan's Terribly Exciting Psychological Test":

```
age <- c(17, 19, 21, 37, 18, 19, 47, 18, 19)
score <- c(12, 10, 11, 15, 16, 14, 25, 21, 29)
```

Assuming no other variables are in the workspace, if I type `who()` I get this:

```
who()
```

```
##    -- Name --    -- Class --    -- Size --
##    age           numeric        9
##    gender        factor         9
##    group         factor         9
##    score         numeric        9
```

So there are four variables in the workspace, `age`, `gender`, `group` and `score`. And it just so happens that all four of them are the same size (i.e., they're all vectors with 9 elements). Aaaand it just so happens that `age[1]` corresponds to the age of the first person, and `gender[1]` is the gender of that very same person, etc. In other words, you and I both know that all four of these variables correspond to the *same* data set, and all four of them are organised in exactly the same way.

However, R *doesn't* know this! As far as it's concerned, there's no reason why the `age` variable has to be the same length as the `gender` variable; and there's no particular reason to think that `age[1]` has any special relationship to `gender[1]` any more than it has a special relationship to `gender[4]`. In other words, when we store everything in separate variables like this, R doesn't know anything about the relationships between things. It doesn't even really know that these variables actually refer to a proper data set. The data frame fixes this: if we store our variables inside a data frame, we're telling R to treat these variables as a single, fairly coherent data set.

To see how they do this, let's create one. So how do we create a data frame? One way we've already seen: if we import our data from a CSV file, R will store it as a data frame. A second way is to create it directly from some existing variables using the `data.frame()` function. All you have to do is type a list of variables that you want to include in the data frame. The output of a `data.frame()` command is, well, a data frame. So, if I want to store all four variables from my experiment in a data frame called `expt` I can do so like this:

```
expt <- data.frame ( age, gender, group, score )
expt
```

```
##   age gender   group score
## 1  17   male group 1    12
## 2  19   male group 1    10
## 3  21   male group 1    11
## 4  37   male group 2    15
## 5  18   male group 2    16
## 6  19 female group 2    14
## 7  47 female group 3    25
## 8  18 female group 3    21
## 9  19 female group 3    29
```

Note that `expt` is a completely self-contained variable. Once you've created it, it no longer depends on the original variables from which it was constructed. That is, if we make changes to the original `age` variable, it will *not* lead to any changes to the age data stored in `expt` .

At this point, our workspace contains only the one variable, a data frame called `expt` . But as we can see when we told R to print the variable out, this data frame contains 4 variables, each of which has 9 observations. So how do we get this information out again? After all, there's no point in storing information if you don't use it, and there's no way to use information if you can't access it. So let's talk a bit about how to pull information out of a data frame.

The first thing we might want to do is pull out one of our stored variables, let's say `score` . One thing you might try to do is ignore the fact that `score` is locked up inside the `expt` data frame. For instance, you might try to print it out like this:

```
score
```

```
## Error in eval(expr, envir, enclos): object 'score' not found
```

This doesn't work, because R doesn't go "peeking" inside the data frame unless you explicitly tell it to do so. There's actually a very good reason for this, which I'll explain in a moment, but for now let's just assume R knows what it's doing. How do we tell R to look inside the data frame? As is always the case with R there are several ways. The simplest way is to use the `$` operator to extract the variable you're interested in, like this:

```
expt$score
```

```
## [1] 12 10 11 15 16 14 25 21 29
```

4.8.2 Getting information about a data frame

One problem that sometimes comes up in practice is that you forget what you called all your variables. Normally you might try to type `objects()` or `who()` , but neither of those commands will tell you what the names are for those variables inside a data frame! One way is to ask R to tell you what the *names* of all the variables stored in the data frame are, which you can do using the `names()` function:

```
names(expt)
```

```
## [1] "age"    "gender" "group"  "score"
```

An alternative method is to use the `who()` function, as long as you tell it to look at the variables inside data frames. If you set `expand = TRUE` then it will not only list the variables in the workspace, but it will "expand" any data frames that you've got in the workspace, so that you can see what they look like. That is:

```
who(expand = TRUE)
```

```
##    -- Name --    -- Class --    -- Size --
##    expt          data.frame     9 x 4
##     $age         numeric        9
##     $gender      factor         9
##     $group       factor         9
##     $score       numeric        9
```

or, since `expand` is the first argument in the `who()` function you can just type `who(TRUE)` . I'll do that a lot in this book.

4.8.3 Looking for more on data frames?

There's a lot more that can be said about data frames: they're fairly complicated beasts, and the longer you use R the more important it is to make sure you really understand them. We'll talk a lot more about them in Chapter 7.
