17.4: Extracting a Subset of a Vector

One very important kind of data handling is being able to extract a particular subset of the data. For instance, you might be interested only in analysing the data from one experimental condition, or you may want to look closely at the data from people over 50 years in age. To do this, the first step is getting R to extract the subset of the data corresponding to the observations that you're interested in. In this section I'll talk about subsetting as it applies to vectors, extending the discussion from Chapters 3 and 4. In Section 7.5 I'll go on to talk about how this discussion extends to data frames.

17.4.1 Refresher

This section returns to the `nightgarden.Rdata` data set. If you're reading this whole chapter in one sitting, then you should already have this data set loaded. If not, don't forget to use the `load("nightgarden.Rdata")` command. For this section, let's ignore the `itng` data frame that we created earlier, and focus instead on the two vectors `speaker` and `utterance` (see Section 7.1 if you've forgotten what those vectors look like). Suppose that what I want to do is pull out only those utterances that were made by Makka-Pakka. To that end, I could first use the equality operator to have R tell me which cases correspond to Makka-Pakka speaking:

```
is.MP.speaking <- speaker == "makka-pakka"
is.MP.speaking
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
```

and then use logical indexing to get R to print out those elements of `utterance` for which `is.MP.speaking` is true, like so:

```
utterance[ is.MP.speaking ]
```

```
## [1] "pip" "pip" "onk" "onk"
```

Or, since I'm lazy, I could collapse it to a single command like so:

```
utterance[ speaker == "makka-pakka" ]
```

```
## [1] "pip" "pip" "onk" "onk"
```

17.4.2 Using `%in%` match multiple cases

A second useful trick to be aware of is the `%in%` operator[110]. It's actually very similar to the `==` operator, except that you can supply a collection of acceptable values. For instance, suppose I wanted to keep only those cases when the utterance is either "pip" or "oo". One simple way do to this is:

```
utterance %in% c("pip","oo")
```

```
##  [1]  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE
```

What this does if return `TRUE` for those elements of `utterance` that are either `"pip"` or `"oo"` and returns `FALSE` for all the others. What that means is that if I want a list of all those instances of characters speaking either of these two words, I could do this:

```
speaker[ utterance %in% c("pip","oo") ]
```

```
## [1] "upsy-daisy"  "upsy-daisy"  "tombliboo"   "makka-pakka" "makka-pakka"
```

17.4.3 Using negative indices to drop elements

Before moving onto data frames, there's a couple of other tricks worth mentioning. The first of these is to use negative values as indices. Recall from Section 3.10 that we can use a vector of numbers to extract a set of elements that we would like to keep. For instance, suppose I want to keep only elements 2 and 3 from `utterance`. I could do so like this:

```
utterance[2:3]
```

```
## [1] "pip" "onk"
```

But suppose, on the other hand, that I have discovered that observations 2 and 3 are untrustworthy, and I want to keep everything *except* those two elements. To that end, R lets you use negative numbers to remove specific values, like so:

```
utterance [ -(2:3) ]
```

```
## [1] "pip" "onk" "ee"  "oo"  "pip" "pip" "onk" "onk"
```

The output here corresponds to element 1 of the original vector, followed by elements 4, 5, and so on. When all you want to do is remove a few cases, this is a very handy convention.

17.4.4 Splitting a vector by group

One particular example of subsetting that is especially common is the problem of splitting one one variable up into several different variables, one corresponding to each group. For instance, in our *In the Night Garden* example, I might want to create subsets of the `utterance` variable for every character. One way to do this would be to just repeat the exercise that I went through earlier separately for each character, but that quickly gets annoying. A faster way do it is to use the `split()` function. The arguments are:

- `x`. The variable that needs to be split into groups.
- `f`. The grouping variable.

What this function does is output a list (Section 4.9), containing one variable for each group. For instance, I could split up the `utterance` variable by `speaker` using the following command:

```
speech.by.char <- split( x = utterance, f = speaker )
speech.by.char
```

```
## $`makka-pakka`
## [1] "pip" "pip" "onk" "onk"
##
## $tombliboo
## [1] "ee" "oo"
##
## $`upsy-daisy`
## [1] "pip" "pip" "onk" "onk"
```

Once you're starting to become comfortable working with lists and data frames, this output is all you need, since you can work with this list in much the same way that you would work with a data frame. For instance, if you want the first utterance made by Makka-Pakka, all you need to do is type this:

```
speech.by.char$`makka-pakka`[1]
```

```
## [1] "pip"
```

Just remember that R does need you to add the quoting characters (i.e. `'`). Otherwise, there's nothing particularly new or difficult here.

However, sometimes – especially when you're just starting out – it can be convenient to pull these variables out of the list, and into the workspace. This isn't too difficult to do, though it can be a little daunting to novices. To that end, I've included a function called `importList()` in the `lsr` package that does this.[111] First, here's what you'd have if you had wiped the workspace before the start of this section:

```
who()
```

```
##    -- Name --          -- Class --    -- Size --
##    age                 numeric        11
##    age.breaks          numeric        4
##    age.group           factor         11
##    age.group2          factor         11
##    age.group3          factor         11
##    age.labels          character      3
##    df                  data.frame     10 x 4
##    is.MP.speaking      logical        10
##    itng                data.frame     10 x 2
##    itng.table          table          3 x 4
##    likert.centred      numeric        10
##    likert.raw          numeric        10
##    opinion.dir         numeric        10
##    opinion.strength    numeric        10
##    some.data           numeric        18
##    speaker             character      10
##    speech.by.char      list           3
##    utterance           character      10
```

Now we use the `importList()` function to copy all of the variables within the `speech.by.char` list:

```
importList( speech.by.char, ask = FALSE)
```

Because the `importList()` function is attempting to create new variables based on the names of the elements of the list, it pauses to check that you're okay with the variable names. The reason it does this is that, if one of the to-be-created variables has the same name as a variable that you already have in your workspace, that variable will end up being overwritten, so it's a good idea to check. Assuming that you type `y`, it will go on to create the variables. Nothing *appears* to have happened, but if we look at our workspace now:

```
who()
```

```
##    -- Name --          -- Class --    -- Size --
##    age                numeric        11
##    age.breaks         numeric        4
##    age.group          factor         11
##    age.group2         factor         11
##    age.group3         factor         11
##    age.labels         character      3
##    df                 data.frame     10 x 4
##    is.MP.speaking     logical        10
##    itng               data.frame     10 x 2
##    itng.table         table          3 x 4
##    likert.centred     numeric        10
##    likert.raw         numeric        10
##    makka.pakka        character      4
##    opinion.dir        numeric        10
##    opinion.strength   numeric        10
##    some.data          numeric        18
##    speaker            character      10
##    speech.by.char     list           3
##    tombliboo          character      2
##    upsy.daisy         character      4
##    utterance          character      10
```

we see that there are three new variables, called `makka.pakka`, `tombliboo` and `upsy.daisy`. Notice that the `importList()` function has converted the original character strings into valid R variable names, so the variable corresponding to `"makka-pakka"` is actually `makka.pakka`.[112] Nevertheless, even though the names can change, note that each of these variables contains the exact same information as the original elements of the list did. For example:

```
> makka.pakka
[1] "pip" "pip" "onk" "onk"
```
