R12 Factors

Covered in R12

- Creating factors
- Specifying the order of the categories
- Abbreviating the names of arguments
- Changing the name of categories
- Indexing and arithmetic

1 Creating factors

In the last chapter we looked at vector objects, which are one-dimensional ordered collections of data of the *same type* (numeric, character, logical, *etc*). These can be used to store results of one particular variable that are of interest to us, say policyholders name, age, gender, *etc*.

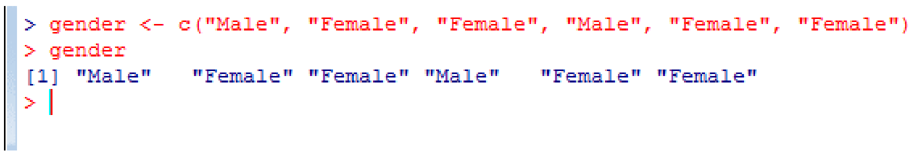
In this chapter we look at a special vector used for storing categorical data (such as gender, occupation, make of car, country, *etc*) called a factor. Unlike, for example, the policyholder's name, categorical data can only take one of a limited number of categories. For example, gender can only take the categories male or female.

In R, the different categories are called *levels* and they are assigned the values 1, 2, 3, …, *n*. This allows R to store them more efficiently (rather than treating each as unique) and use the categories for graphing or as inputs in a statistical model, such as a generalised linear model.

Suppose we collect the gender of six policyholders:

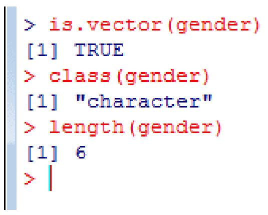
Male Female Female Male Female Female

We could store these in a vector, gender, using the concatenate function:



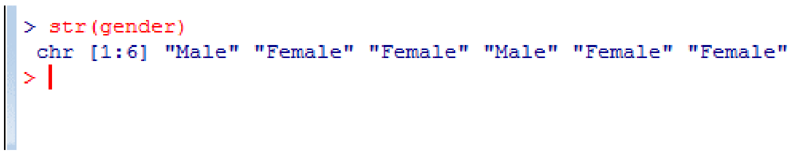
```
> gender <- c("Male", "Female", "Female", "Male", "Female", "Female")
> gender
[1] "Male"   "Female" "Female" "Male"   "Female" "Female"
> 
```

The object gender is a character vector of length 6:



```
> is.vector(gender)
[1] TRUE
> class(gender)
[1] "character"
> length(gender)
[1] 6
> 
```

We can examine the structure of the object gender:



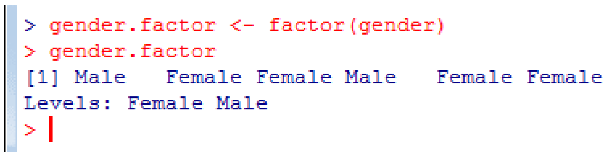
```
> str(gender)
 chr [1:6] "Male" "Female" "Female" "Male" "Female" "Female"
> 
```

As expected, we see it contains character data ("chr"), has six values and that they are stored in R's memory as "Male", "Female", *etc*.

To convert a vector into a factor we use the factor command:

factor(<vector object>)

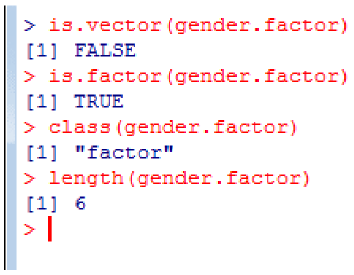
Let's take the data stored in the vector object gender and put it in a factor object, which we'll call gender.factor and print it out:



```
> gender.factor <- factor(gender)
> gender.factor
[1] Male   Female Female Male   Female Female
Levels: Female Male
> 
```

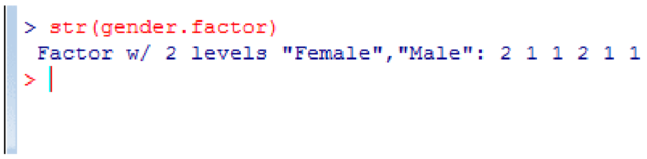
We can see that it prints the elements (but no longer as character values in speech marks) and it also gives the levels (*ie* the categories) that the data can take. Note that the levels are, by default, sorted into alphabetical order.

We can see that gender.factor is no longer a vector object but a factor object but still has length 6:



```
> is.vector(gender.factor)
[1] FALSE
> is.factor(gender.factor)
[1] TRUE
> class(gender.factor)
[1] "factor"
> length(gender.factor)
[1] 6
> 
```

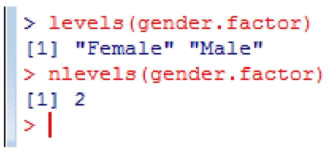
We can examine the structure of gender.factor:



```
> str(gender.factor)
 Factor w/ 2 levels "Female","Male": 2 1 1 2 1 1
> 
```

We can see that it is a factor with two levels ("Female" and "Male") which are by default in alphabetical order. However, when it displays the elements we no longer see "Male", "Female", "Female", … but 2, 1, 1, … . This is because R assigns positive integers to each level/category. The female category is assigned a value of 1 and the male category is assigned a value of 2, and R stores these numbers in its memory instead. So essentially we can see that R has converted our categorical data to an equivalent numeric vector. This saves memory (1 and 2 use less space than "Female" and "Male") and means we can put these numbers into functions.

We can use **levels(<factor object>)** to display the levels and **nlevels(<factor object>)** to display the number of levels of a factor object:



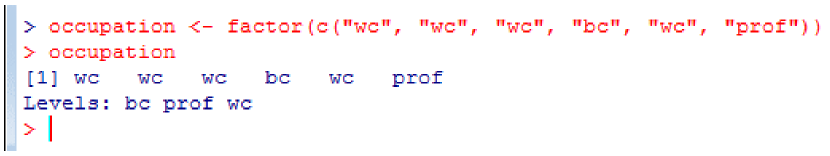
```
> levels(gender.factor)
[1] "Female" "Male"
> nlevels(gender.factor)
[1] 2
> |
```

In the example above we converted a categorical character vector into a factor. We could also have entered the data directly using the factor command.

Suppose we collect the occupations (which can take the categories of blue collar, white collar and professional) of the same six policyholders. Abbreviating them as bc, wc and prof they were:

wc wc wc bc wc prof

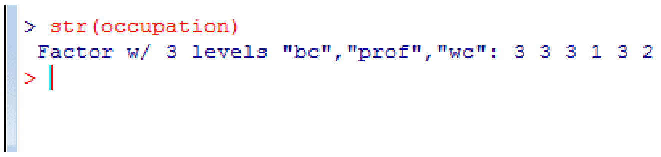
We can put these in a factor object, which we'll call occupation, as follows:



```
> occupation <- factor(c("wc", "wc", "wc", "bc", "wc", "prof"))
> occupation
[1] wc   wc   wc   bc   wc   prof
Levels: bc prof wc
> |
```

Again we see that the levels are assigned alphabetically.

Looking at the structure:



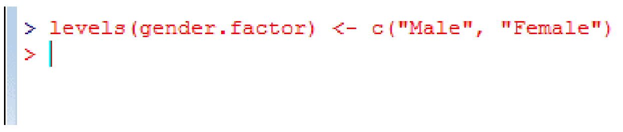
```
> str(occupation)
 Factor w/ 3 levels "bc","prof","wc": 3 3 3 1 3 2
> |
```

We can again see that the numbers have been assigned alphabetically to each category, so bc is level 1, prof is level 2 and wc is level 3.

2 Specifying the order of the categories

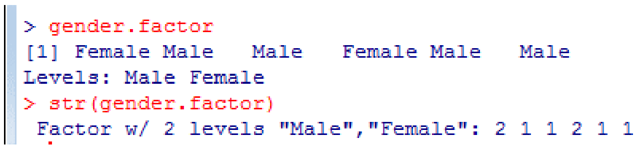
In this section we look at how we can specify the order of the levels (*ie* the categories) to be something other than alphabetical.

For an object that already exists we can change the levels using the **levels(<factor object>)** command. For example the gender.factor object has two levels currently in alphabetical order (Female, Male). To change them to (Male, Female) we do the following:



```
> levels(gender.factor) <- c("Male", "Female")
> |
```

We can examine the printout and structure:



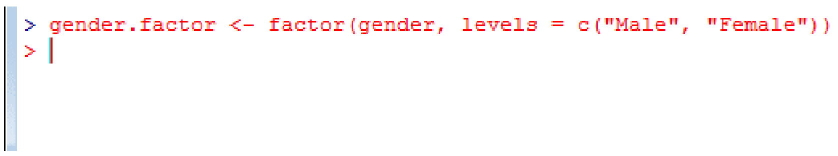
```
> gender.factor
[1] Female Male   Male   Female Male   Male
Levels: Male Female
> str(gender.factor)
 Factor w/ 2 levels "Male","Female": 2 1 1 2 1 1
```

The levels have changed order, male is first instead of female but the assignment of 1 to female and 2 to male is unchanged. This is unfortunate as the data values were stored internally as 2, 1, 1, … and so whereas before that was Male, Female, Female,… it now says Female, Male, Male. So what we have done is relabelled the levels of the factors and by doing this have changed our data set! This is obviously not a good idea.

So we have to specify the order of the levels when we create the factor. Levels is an optional argument of the factor command:

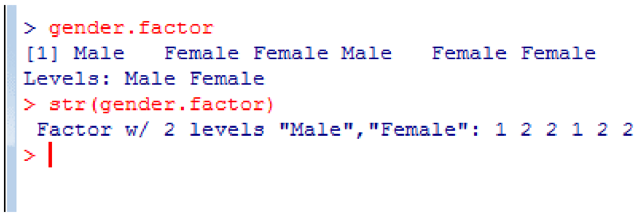
factor(<vector object>, levels=<levels vector>)

So let's redefine the object gender.factor using the factor command but this time we'll specify the order of the levels to be Male then Female:



```
> gender.factor <- factor(gender, levels = c("Male", "Female"))
> |
```

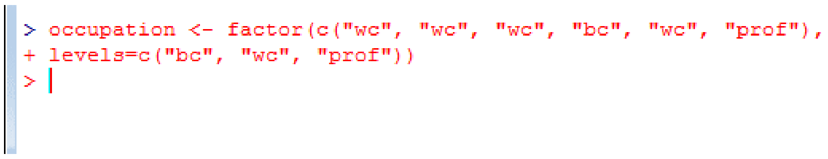
Now when we print it out and look at its structure we get the following:



```
> gender.factor
[1] Male   Female Female Male   Female Female
Levels: Male Female
> str(gender.factor)
 Factor w/ 2 levels "Male","Female": 1 2 2 1 2 2
> 
```

We can see that the levels are now in the specified order (Male then Female) and the assignment of the numbers now follows this order, so Male = 1 and Female = 2. Hence the data values are as they should be Male, Female, Female, *etc*.

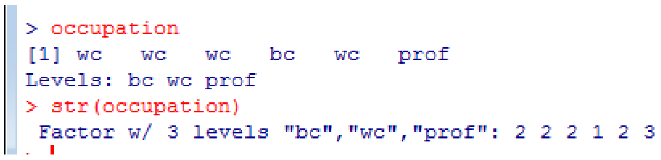
Let's now redefine the factor object occupation but this time we'll specify the order of the levels to be bc, then wc and then prof:



```
> occupation <- factor(c("wc", "wc", "wc", "bc", "wc", "prof"),
+ levels=c("bc", "wc", "prof"))
> 
```

Note that if you are entering the above command in an RStudio Script, rather than in the Console, then you won't need to enter the "+" symbol, This is just R telling us we need to enter more if we press enter before completing the command.

Now when we print it out and look at its structure we get the following:

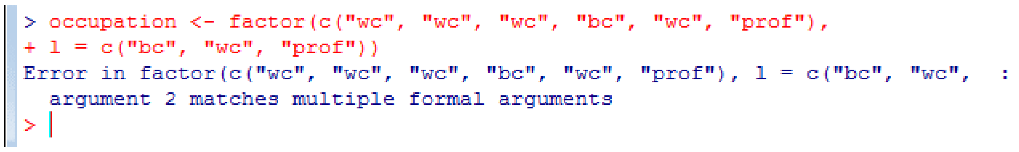


```
> occupation
[1] wc   wc   wc   bc   wc   prof
Levels: bc wc prof
> str(occupation)
 Factor w/ 3 levels "bc","wc","prof": 2 2 2 1 2 3
```

We can see that the levels are now in the order we've specified *and* the assignment of the numbers to the levels is in this order, so bc is now 1, wc is now 2 and prof is now 3.

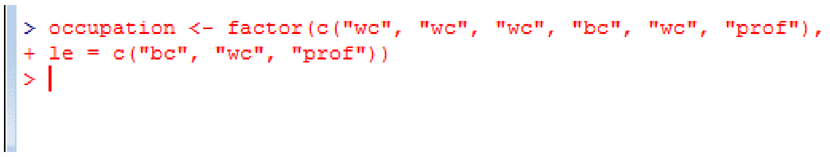
3 Abbreviating the names of the arguments of a function

Recall from an earlier chapter that when specifying the arguments of a function we could use a unique abbreviation for the argument's name or omit it altogether as long as we put the arguments in the correct order. The factors command has another possible argument called "labels" and so we can't uniquely abbreviate the argument "levels" to "l":



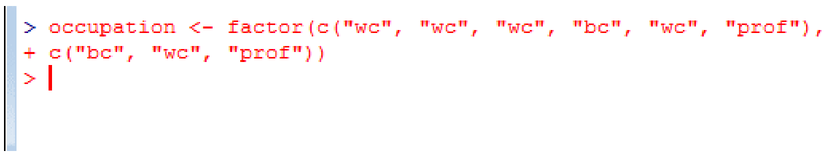
```
> occupation <- factor(c("wc", "wc", "wc", "bc", "wc", "prof"),
+ l = c("bc", "wc", "prof"))
Error in factor(c("wc", "wc", "wc", "bc", "wc", "prof"), l = c("bc", "wc",  :
  argument 2 matches multiple formal arguments
> 
```

However, we can uniquely abbreviate it to, say, "le":



```
> occupation <- factor(c("wc", "wc", "wc", "bc", "wc", "prof"),
+ le = c("bc", "wc", "prof"))
> 
```

Levels is the actually the second argument and so we could omit its name altogether:



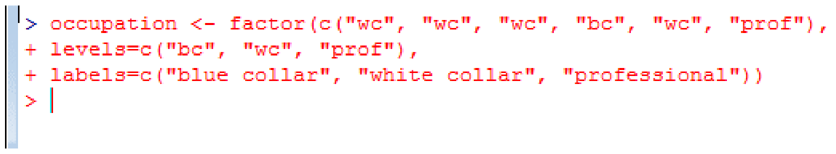
```
> occupation <- factor(c("wc", "wc", "wc", "bc", "wc", "prof"),
+ c("bc", "wc", "prof"))
> 
```

4 Changing the name of the categories

For the occupations we used bc, wc and prof as abbreviations because it was quicker than entering blue collar, white collar and professional for every policyholder. It's possible to display the full name of the data entry, but to save time, enter the data as an abbreviation. The way to do this is to use the third argument of the factors command which is "labels".

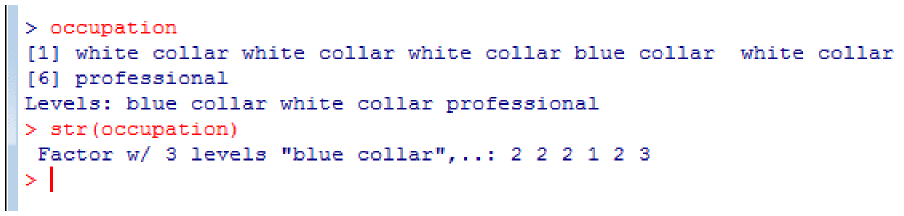
factor(<vector object>, levels=<levels vector>, labels=<labels vector>)

We'll now re-enter the previous command but this time with the labels argument with the full names *in the same order as the levels argument*:



```
> occupation <- factor(c("wc", "wc", "wc", "bc", "wc", "prof"),
+ levels=c("bc", "wc", "prof"),
+ labels=c("blue collar", "white collar", "professional"))
> 
```

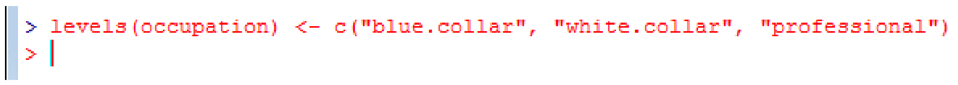
Now when we print it and look at its structure we get:



```
> occupation
[1] white collar white collar white collar blue collar  white collar
[6] professional
Levels: blue collar white collar professional
> str(occupation)
 Factor w/ 3 levels "blue collar",..: 2 2 2 1 2 3
> 
```

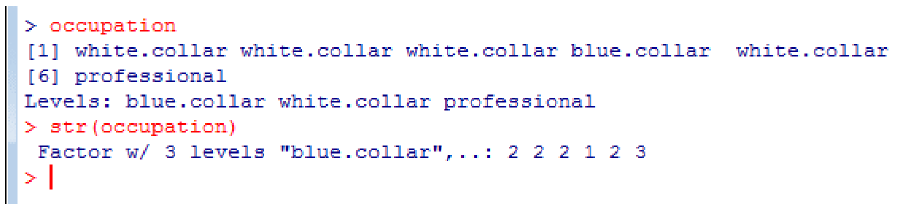
We can see that the data are now printed in full. Unfortunately because there are two words in the first two categories it's a little confusing when they're displayed to differentiate between the policyholders. So it would be wise to use a full stop or underscore to separate the words.

We could re-enter the whole command again but as we saw earlier we can change the labels of an existing factor using the **levels(<factor object>)** command. This is a bit confusing as we'd expect to use a "labels(<factor object>)" command but this is unfortunately the way R works. So, being careful to ensure we keep the correct order of the levels/categories, we'll change the labels of the factor object "occupation" to "blue.collar", "white.collar" and "professional" as follows:



```
> levels(occupation) <- c("blue.collar", "white.collar", "professional")
> 
```

Printing occupation and looking at its structure we get:



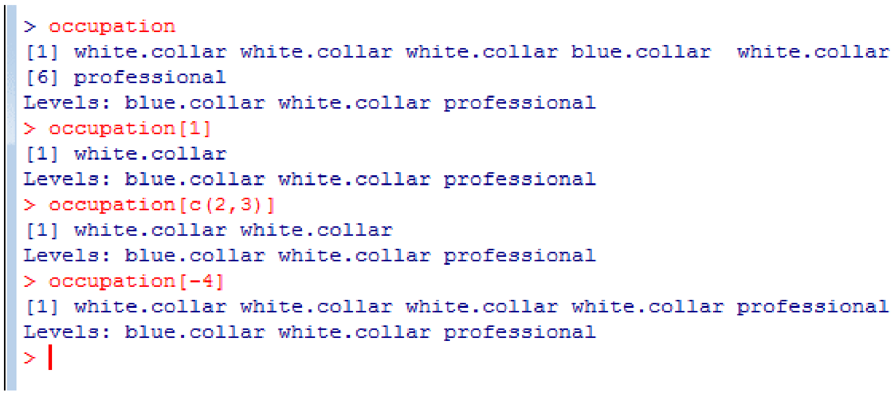
```
> occupation
[1] white.collar white.collar white.collar blue.collar  white.collar
[6] professional
Levels: blue.collar white.collar professional
> str(occupation)
 Factor w/ 3 levels "blue.collar",..: 2 2 2 1 2 3
> 
```

We can see that not only have the levels been relabelled but using the full stops makes it much easier to differentiate between the different data values. So levels inside the factor command specifies the order only, however levels(<factor object>) replaces the category names (*ie* the labels) with the new names.

5 Indexing and arithmetic

Indexing

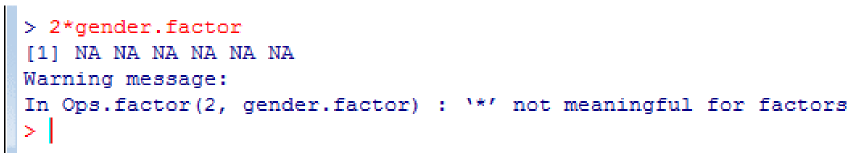
Just like for vectors we can select some of the elements using indexing. Here are some examples on the occupation factor:



```
> occupation
[1] white.collar white.collar white.collar blue.collar  white.collar
[6] professional
Levels: blue.collar white.collar professional
> occupation[1]
[1] white.collar
Levels: blue.collar white.collar professional
> occupation[c(2,3)]
[1] white.collar white.collar
Levels: blue.collar white.collar professional
> occupation[-4]
[1] white.collar white.collar white.collar white.collar professional
Levels: blue.collar white.collar professional
> 
```

Factor arithmetic

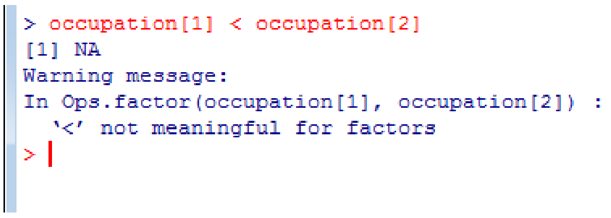
Whilst each category/level is stored as a non-negative integer, factors are, for all intents and purposes, character data. As such, we can't apply arithmetic operations to them. For example:



```
> 2*gender.factor
[1] NA NA NA NA NA NA
Warning message:
In Ops.factor(2, gender.factor) : '*' not meaningful for factors
> 
```

6 Ordered factors

The categorical data we've been looking at in this chapter so far (gender and occupation) has no intrinsic order to it. As such, if we try to compare policyholders, it will return an error. For example, comparing the first and second policyholder's occupations (white.collar and white.collar) gives the following:



```
> occupation[1] < occupation[2]
[1] NA
Warning message:
In Ops.factor(occupation[1], occupation[2]) :
  '<' not meaningful for factors
> |
```

This kind of quantitative data is often called *nominal* data.

However, some categorical data do have an inherent order. For example, we might have the categories small, medium or large, which have the following order:

small < medium < large

Or the categories strongly disagree, disagree, all the way up to strongly agree:

strongly disagree < disagree < neutral < agree < strongly agree

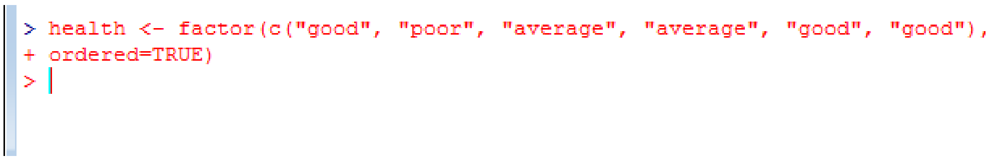
This kind of quantitative data is called *ordinal* data and, in R, we store ordinal data in an ordered factor. To do this we use the factor command as before with the optional argument "ordered" set to TRUE (by default if it's omitted it is set to FALSE which gives us nominal data).

factor(<vector object>, levels=<levels vector>, labels=<labels vector>, ordered = TRUE)

Suppose our six policyholders are asked to describe their general health and the ordered categories are poor, average and good:

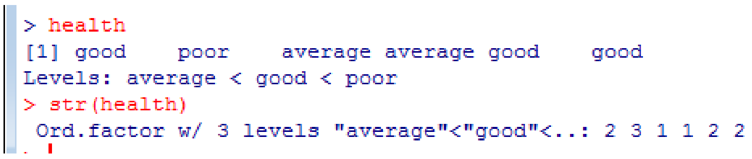
good poor average average good good

We can put these in an ordered factor object, which we'll call health, as follows:



```
> health <- factor(c("good", "poor", "average", "average", "good", "good"),
+ ordered=TRUE)
> |
```

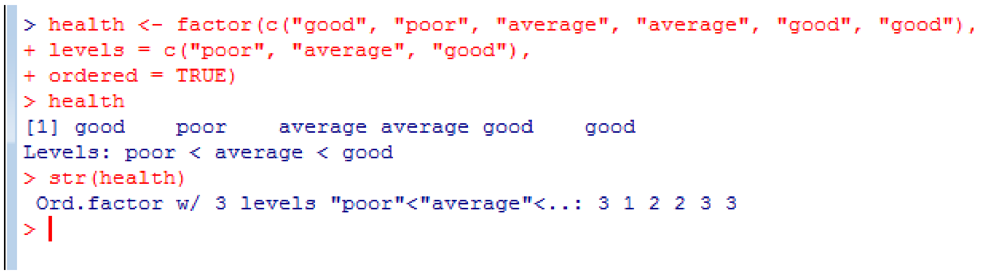
However, because R sets the levels alphabetically by default, the order it gives is not the most sensible:



```
> health
[1] good    poor    average average good    good
Levels: average < good < poor
> str(health)
 Ord.factor w/ 3 levels "average"<"good"<..: 2 3 1 1 2 2
```

It says average < good < poor. So poor is the best health category! The lesson here is to always specify the desired order of the levels!

Re-entering the command with the levels option and the ordered option:

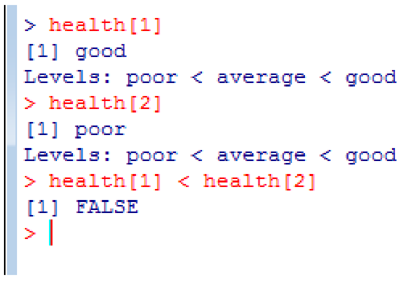


```
> health <- factor(c("good", "poor", "average", "average", "good", "good"),
+ levels = c("poor", "average", "good"),
+ ordered = TRUE)
> health
[1] good    poor    average average good    good
Levels: poor < average < good
> str(health)
 Ord.factor w/ 3 levels "poor"<"average"<..: 3 1 2 2 3 3
> 
```

We can see that they are now in the correct ascending order poor < average < good.

Comparing elements from ordered factors

Now the order has been specified, it makes sense to compare elements in a factor.



```
> health[1]
[1] good
Levels: poor < average < good
> health[2]
[1] poor
Levels: poor < average < good
> health[1] < health[2]
[1] FALSE
> 
```

7 Summary

Key terms

Factor	A special type of vector used for storing categorical data
Categorical data	Data which can only take one of a number specified categories, *eg* gender taking only Male or Female
Levels	The categories that data can take in a factor
Labels	Names given to a factor's levels

Key commands

factor(<vector object>)	Turns a vector into a factor. Has optional arguments of levels, labels and ordered
is.factor(<object>)	Logical test of whether <object> is a factor Returns TRUE or FALSE.
levels(<factor object>)	Displays the levels of a factor object
nlevels(<factor object>)	Displays the number of levels of a factor object

8 Have a go

You will only get proficient at R by practising.

1. Create an ordered factor, *results*, containing some maths test results from 7 students:

   (A, C, C, E, D, B, B)

   Label the grades A-E as Excellent, Good, Average, Below Average and Poor.

   Hint: remember the lowest category goes first.

2. Use a command in R to check that the second student performed better than the fifth student.