Exercises

Exercise 3: Importing, inspecting and subsetting data

Read Chapter 3 to help you complete the questions in this exercise.

1. As in previous exercises, either create a new R script or continue with your previous R script in your RStudio Project. Again, make sure you include any metadata you feel is appropriate (title, description of task, date of creation etc) and don't forget to comment out your metadata with a `#` at the beginning of the line.

2. If you haven't already, download the data file *'cardiacdata.xlsx'* from the **Data** link and save it to the `data` directory you created in Exercise 1 in your RStudio project. You will use this same dataset for this practical and the next two, so it is worth putting it somewhere sensible now.

3. Notice that the file you have just downloaded is a spreadsheet. A spreadsheet is a program's own format, and R would much rather be given plain text. Open the *'cardiacdata.xlsx'* file in Microsoft Excel (or even better use an open source equivalent - LibreOffice is a good free alternative) and save it as a tab delimited file type (see Section 3.3.1 of the Introduction to R book or watch this video if you're not sure how to do this). Name the file *'cardiacdata.txt'* and save it to the `data` directory. If you're a Windows user be careful with file extensions (things like *.txt*). By default, Windows doesn't show you the file extension (maybe the boffins at Microsoft don't think you need to know complicated things like this?) so if you enter *'cardiacdata.txt'* as a filename you might end up with a filename *'cardiacdata.txt.txt'*!

4. Time for a quick description of the *'cardiacdata.txt'* dataset to get your bearings. These data come from a cohort study of the risk factors for cardiovascular disease. A group of 163 adults, aged between 55 and 75, were examined at the start of the study and a range of measurements was taken on each of them. Each row of the dataset is one patient, identified by a unique patient number (`patno`), and each column is one of those measurements. As well as their age in years (`age`) and sex (`sex`), the study recorded blood pressure as the usual pair of numbers, systolic and diastolic, in mmHg (`systolic`, `diastolic`); body mass index in kg m^{-2} (`bmi`); three fats measured in a blood sample, all in mmol l^{-1}, namely total cholesterol, HDL cholesterol and triglyceride (`tchol`, `hdlchol`, `triglyceride`); how many units of alcohol each person drank in the previous week (`alcohol`); and whether they were a current smoker, an ex-smoker or had never smoked (`smoking`). The structure of these data is known as a rectangular dataset (aka 'tidy' data by the cool kids). Each row is an individual observation, each column a separate variable, and the variable names are contained in the first row of the dataset (aka a header). Unlike the tidy examples in textbooks, though, this dataset is not complete. A few values are missing, and wherever a measurement was not recorded the cell in the file holds `NA`, which is how R writes a missing value and what it will expect to find when it imports the file. On top of that, at least one value is present but cannot possibly be correct. Tracking down both kinds of problem, the values that are missing and the values that are wrong, is part of your job over this practical and the next.

5. Now let's import the *'cardiacdata.txt'* file into R. To do this you will use the workhorse function of data importing, `read.table()`. This function is incredibly flexible and can import many different file types (take a look at the help file) including our tab delimited file. Rather than relying on the default settings, get into the habit of writing out the arguments you care about every time. `header` tells R whether the first row contains the variable names, `sep` tells it what separates one value from the next, and `stringsAsFactors` tells it whether to convert text to factors. That last one is not idle here: `patno` is text rather than a number, so it is the one column the argument acts on. Being explicit costs you a few seconds and saves you a great deal of confusion later. Assign the imported data to a variable with an appropriate name (such as `cardiac`). Take a look at Section 3.2.2 of the Introduction to R book or watch this video if you need any further information.

6. Once you've imported your data file nothing much seems to happen (don't worry, this is normal). To examine the contents of the dataframe one option would be to just type the variable name (`cardiac`) into the console. This is probably not a good idea and doesn't really tell you anything about the dataframe other than there's a lot of data (try it)! A slightly better option is to use the `head()` function to display the first few rows of your dataframe. Again, this is likely to just fill up your console. A better option would be to use the `names()` function which will return a vector of variable names from your dataset. However, all you get are the names of the variables but no other information. A much, much better option is to use the `str()` function to display the structure of the dataset and a neat summary of your variables. Another advantage is that you can copy this information from the console and paste it into your R script (making sure it's commented) for later reference.

So, from the output of the `str()` function, how many observations does this dataset have? How many variables? And now the important question: what type of variable does R think `sex` and `smoking` are, and is it right?

7. Let's do something about the `sex` and `smoking` variables. The way to tell R that a variable holds categories rather than quantities is to make it a **factor**, using the `factor()` function (see `?factor`, and Section 3.1 of the Introduction to R book for where factors sit among R's data types). You give `factor()` the codes that appear in the data with the `levels` argument, and what each code means with the `labels` argument. For `sex`, 1 is Female and 2 is Male. For `smoking`, 1 is Current, 2 is Ex and 3 is Never.

Here is the important part. Do **not** overwrite either of them. Create two **new** variables in the `cardiac` dataframe, called for example `Fsex` and `Fsmoking`, and leave the original `sex` and `smoking` variables exactly as they were. Do this, then use `str()` to check that both really are factors with the levels you expect, and that `sex` and `smoking` are still there and still integers.

Why do we create a new variable rather than overwriting the original? Have a think about it before you reveal the answer.

8. You can get another useful summary of your dataframe by using the `summary()` function. This will provide you with some useful summary statistics for each variable. Notice how the type of output depends on whether the variable is a factor or a number: for `Fsex` and `Fsmoking`, the factors you just created, you get a count of patients in each level, whereas for a numeric variable you get the minimum, maximum, mean, median and quartiles. Compare `sex` and `Fsex` in the output and you will see the same information summarised in two very different ways. Another useful feature of the `summary()` function is that it will also count the number of missing values in each variable. Which variables have missing values, and how many? While you are looking at this output, check the maximum value of `bmi`. Does that look like a plausible body mass index to you? Make a note of it and hang on to that thought until Exercise 4.

9. Summarising and manipulating dataframes is a key skill to acquire when learning R. Although there are many ways to do this, we will concentrate on using the square bracket `[ ]` notation which you used

previously with vectors. The key thing to remember when using `[ ]` with dataframes is that dataframes have two dimensions (think rows and columns) so you always need to specify which rows and which columns you want inside the `[ ]` (see Section 3.4.1 and this video for some additional background information and a few examples). Let's practice.

a) Extract all observations (remember - rows) from the `cardiac` dataframe and the columns `patno`, `sex`, `smoking` and `tchol` and assign to a variable called `cardiac_risk`.

b) Next, extract all rows except the first 10 rows and all columns except the last column. Careful here, which column is the last one now? Remember, you can specify the columns you want either by position or by name. Practice both ways.

10. In addition to extracting rows and columns from your dataframe by position (as you have just been doing) you can also use conditional statements to select particular rows based on some logical criteria. This is very useful but takes a bit of practice to get used to (see Section 3.4.2 for an introduction). Extract rows from your dataframe (all columns by default) based on the following criteria (note: you will need to assign the results of these statements to appropriately named variables, I'll leave it up to you to use informative names!). You now have properly labelled `Fsex` and `Fsmoking` variables, so use those rather than the original `sex` and `smoking` variables. Your conditions will then read like English and you will not have to keep looking up which number means what:

- patients with a systolic blood pressure greater than 160 mmHg
- male patients who have never smoked and whose diastolic pressure is greater than the median diastolic pressure (76 mmHg)
- all patients who are not ex-smokers

11. A really neat feature of conditional statements is that you can put R functions inside them, so the threshold is worked out from the data rather than typed in by hand (see Section 3.4.2 again). This is useful because if you hard code the median as 76, as you did in Q10, your code becomes silently wrong the moment somebody adds another twenty patients to the study, whereas `median(cardiac$diastolic)` is still right.

So let's put this into practice by building on your solution to Q10. Write some code to extract all rows from the `cardiac` dataframe with a systolic pressure greater than 160 mmHg and a total cholesterol greater than average (hint: use the `mean()` function inside your conditional statement). Now look carefully at what comes back. Can you see a problem with it? Have a look at Section 2.4.5 of the Introduction to R book, discuss the cause of this problem with an instructor and explore possible solutions.

End of Exercise 3