Exercises

Exercise 4: Cleaning, summarising and linking data

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

This exercise picks up exactly where Exercise 3 left off and uses the same `cardiac` dataframe. If you are starting a fresh R session, either continue with the R script you wrote for Exercise 3 or run the two lines below before you go any further. They are the import from **Exercise 3, Q5** and the two factors from **Exercise 3, Q7**.

```
cardiac <- read.table('data/cardiacdata.txt', header = TRUE, sep = "\t", stringsAsFactors = TRUE)

cardiac$Fsex <- factor(cardiac$sex, levels = c(1, 2),
                       labels = c("Female", "Male"))

cardiac$Fsmoking <- factor(cardiac$smoking, levels = c(1, 2, 3),
                           labels = c("Current", "Ex", "Never"))
```

1. In Exercise 3 you learned how to extract rows that meet a condition. You can use that skill to check whether your data are believable. This matters a lot: a value that is simply wrong will not stop your code running, will not produce a warning, and will quietly change every result you calculate from it. Run `summary()` on your dataframe again and look at the minimum and maximum of each numeric variable. Three variables contain values that are not merely unusual but impossible. Two are impossibly low and one is impossibly high (hint: a body mass index below about 15 or above about 60 is essentially unheard of in a living adult, and nobody can survive with no cholesterol in their blood at all, so a blood fat concentration of exactly 0 is not a low reading, it is an impossible one). Find them, work out which patient the impossibly high one belongs to, and then set all of the offending values to `NA` so they cannot contaminate anything you calculate later (you will need the square bracket `[ ]` notation from Section 3.4.2, this time on the left hand side of an assignment). Re-run `summary()` afterwards to check it worked. Why is `NA` the right answer here, rather than deleting the whole patient record or guessing what the value should have been?

2. Another useful way to manipulate your dataframes is to sort the rows based on the value of a variable, or on a combination of variables. Rather counter-intuitively you should use the `order()` function to sort your dataframes, not the `sort()` function (see Section 3.4.3 of the Introduction to R book for an explanation). Ordering dataframes uses the same logic you practised in Q12 in Exercise 2. Sort all rows in the `cardiac` dataframe by ascending order of systolic pressure within each level of smoking status, and assign the result to a variable with a sensible name. The trick is to remember that `order()` will take more than one variable, and that the order you give them in matters. Now take a look at the bottom of your sorted dataframe. Where have the patients with a missing smoking status ended up, and why?

3. Often, we would like to summarise variables by, for example, calculating a mean, median or counting the number of observations. To do this for a single variable it's fairly straightforward:

```
mean(cardiac$age)          # mean age
median(cardiac$systolic)   # median systolic blood pressure
length(cardiac$tchol)      # number of observations
```

Perhaps more interestingly, you might want to summarise one variable conditional on the level of another categorical variable, and to do several variables at once. The `aggregate()` function does exactly this (see Section 3.5 of the Introduction to R book, or `?aggregate`). Use `aggregate()` to calculate the mean age, systolic pressure, diastolic pressure and total cholesterol for each smoking category.

The grouping variable goes inside the `by` argument wrapped in `list()`, because a list is how `aggregate()` accepts more than one of them at once. Use `Fsmoking`, the factor you created in **Exercise 3, Q7**, rather than the original `smoking` column, so that your output comes back labelled Current, Ex and Never instead of 1, 2 and 3.

Run it exactly as it comes, with no special handling of anything, and then look hard at the `tchol` column. Something has gone wrong. What, and why?

4. Now fix it. `aggregate()` hands any extra arguments straight on to the function you asked it to use, so the `na.rm` argument you met in **Exercise 3, Q11** works here too. Add it and check that all three cholesterol means appear.

Then calculate the same means for each combination of smoking category and sex. Two grouping variables go inside the same `list()`, and again use the factor versions, `Fsmoking` and `Fsex`, rather than the original `smoking` and `sex` columns. When you have your answer, count up the patients in it. Are they all there?

5. Knowing how many observations are present for each category (or combinations of categories) is useful to determine whether you have an adequate sample size (for subsequent modelling for example). Use the `table()` function to determine the number of patients in each smoking category (see Section 3.5 again for more information). Next use the same function to display the number of patients for each combination of smoking category and sex. Use `Fsmoking` and `Fsex` here as well, so that your table is labelled rather than numbered. Does `table()` tell you about the patients whose smoking status is missing?

6. Not every variable has its values distributed symmetrically about the centre. Look back at `triglyceride` in your `summary()` output. The mean, 1.556, sits above the median, 1.380, and the rest of the summary is lopsided in the same direction: the third quartile is 0.50 above the median while the first is only 0.32 below it, and the maximum is 3.29 above while the minimum is 0.86 below. Everything is stretched out to the right, which is what a long right hand tail looks like in a table of numbers, a few patients with much higher values than everybody else. Transforming to a log scale pulls that tail in, and it is a common way of dealing with skewed variables.

Create a new variable in the `cardiac` dataframe called `log_triglyceride`, holding the base 10 logarithm of `triglyceride` (see `?log10`, and Section 3.4.4 for adding a column). Compare the mean and the median before and after. What has happened to the gap between them?

Data linkage

Real analyses almost never involve a single file. The measurements you want are usually spread across several sources and you have to bring them together first. When those sources hold records about the same people, combining them into one dataset is called data linkage, and it is an important technique in health data science. Hospital admissions linked to prescribing records, a birth cohort linked to school attainment, a patient list linked to the death register. Each linkage answers a question that neither source could answer on its own, and this is the term we will use for it throughout the course.

The idea is simple. You match records on something that identifies the same person in both sources, usually an identifier such as a patient number. However, if implemented carelessly, analyses can go quietly wrong, which is what the next three questions are about. In R the function that does the work is `merge()` (see Section 3.4.5 of the Introduction to R book), and the operation it performs is called a join.

7. The patients in this study were followed up ten years after their first examination, and those follow-up measurements are held in a separate file. Download *'cardiac_followup.txt'* from the **Data** link, save it to your `data` directory and import it with `read.table()` into a variable called `followup`. It holds the patient number, `patno`, along with the systolic blood pressure and body mass index measured at follow-up, `systolic10` and `bmi10`. How many rows does it have, and how does that compare with `cardiac`? The `nrow()` function will tell you, and it does exactly what the name suggests: give it a dataframe and it returns the number of rows.

Now use the `merge()` function to link the two together on the patient number, and assign the result to `cardiac_fu`. Left to its own devices `merge()` performs what is called an inner join, which keeps only the patients who appear in both sources. How many rows does the linked dataframe have, and can you explain why?

8. Losing 55 patients without being told is exactly the sort of thing that can ruin an analysis, and it is why you check row counts every single time you link two sources. Often what you actually want is a left join, which keeps every patient from the first dataframe whether or not they have a match in the second. Look at Section 3.4.5 and the help file for `merge()`, and find the argument that switches `merge()` from an inner join to a left join. Redo the linkage keeping all 163 patients, this time assigning the result to a new dataframe called `cardiac_all`, and then check how many of those patients have a missing follow-up systolic pressure.

That last part needs a way of counting missing values, which you have not met yet. `is.na()` takes a variable and returns `TRUE` or `FALSE` for every value in it, `TRUE` where the value is missing. `sum()` then counts those `TRUE`s for you, because R treats `TRUE` as 1 and `FALSE` as 0. So `sum(is.na(cardiac_all$systolic10))` reads as 'how many follow-up systolic values are missing'. You will need it again in Q9.

9. One last linkage, and this one has a twist in it. The patients were also followed up for hospital admissions over the same ten years. Download *'cardiac_admissions.txt'* from the **Data** link, save it to your `data` directory and import it into a variable called `admissions`. It holds one row for each patient who was admitted at least once, and the number of times they were admitted. Note these admission records are simulated but reflect how these types of data are typically formatted.

a) Merge the `admissions` dataframe and `cardiac_all`, keeping all 163 patients exactly as you did in Q8. Check you still have 163 rows, then count how many patients ended up with a missing `n_admissions`.

b) Now the twist. Up until now, an `NA` has meant 'there should be a value here and we do not have it (missing)'. Does it mean that here? Work out what a missing `n_admissions` actually tells you about that patient (hint: should it actually be a value, and if so, what value). Once you have figured out what `NA` should actually represent, replace these `NA` values (hint: you cannot identify missing values with a conditional statement like `cardiac_all$n_admissions == NA`. Use the `is.na()` function from Q8 instead.)

10. Ok, we have spent quite a bit of time (and energy) learning how to clean, summarise and link dataframes. The last thing we need to cover is how to export a dataframe from R to an external file (see Section 3.6 of the book for more details). Export your cleaned and linked dataframe `cardiac_all` to a file called 'cardiac_clean.txt' in the `output` directory you created in Exercise 1. To do this you will need to use the `write.table()` function. You want to include the variable names in the first row of the file, but you don't want to include the row names. Also, make sure the file is a tab delimited file. Once you have created your file, try to open it in Microsoft Excel (or open source equivalent). Finally, and this is the part people forget: add a comment block at the top of your R script listing every change you made to the data and why. Which values did you set to `NA`, in which variables, and which patients did they belong to? Someone reading your work in six months, including you, needs to be able to answer that without re-running anything.

End of Exercise 4