Exercises

Exercise 3: Importing and manipulating dataframes

In this exercise you'll learn how to import your data into R as a dataframe object, how to usefully summarise and manipulate dataframes and finally how to export data from R. All of this material is covered in Chapter 3 of the Introduction to R book.

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 *'whaledata.xlsx'* from the **Data** link and save it to the `data` directory you created in Exercise 1.

3. Open this 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 if you're not sure how to do this). Name the file *'whaledata.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 *'whaledata.txt'* as a filename you might end up with a filename *'whaledata.txt.txt'*!

4. Time for a quick description of the *'whaledata.txt'* dataset to get your bearings. These data were collected during two research cruises in the North Atlantic in May and October 2003. During these two months the research vessel visited multiple stations (areas) and marine mammal observers recorded the number of whales (who doesn't love whales!) at each of these stations. The time the vessel spent at each station was also recorded along with other site specific variables such as the latitude and longitude, water depth and gradient of the seabed. The researchers also recorded the ambient level of sub-surface noise with a hydrophone and categorised this variable into 'low', 'medium' or 'high'. The structure of these data is known as a rectangular dataset (aka 'tidy' data by the cool kids) with no missing cells. Each row is an individual observation and each column a separate variable. The variable names are contained in the first row of the dataset (aka a header).

5. Now let's import the *'whaledata.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. Don't forget to include the appropriate arguments when using the `read.table()` function (remember that header?) and assign it a variable with an appropriate name (such as `whale`). Take a look at Section 3.2.2 of the Introduction to R book 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 (`whale`) into the console. This is probably not a good idea and doesn't really tell you anything about the dataframe other than there's alot of data (try it)! A slightly better option is to use the `head()` function to display the first 5 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. How many observations does this dataset have? How many variables are in this dataset? What type of variables are `month` and `water.noise`?

7. 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. 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?

8. 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 for some additional background information and a few examples). Let's practice. Extract all the elements of the first 10 rows and the first 4 columns of the `whale` dataframe and assign to a new variable called `whale.sub`. Next, extract all observations (remember - rows) from the `whale` dataframe and the columns `month`, `water.noise` and `number.whales` and assign to a variable called `whale.num`. Now, extract the first 50 rows and all columns form the original dataframe and assign to a variable `whale.may` (there is a better way to do this with conditional statements - see below). Finally, extract all rows except the first 10 rows and all columns except the last column. Remember, for some of these questions you can specify the columns you want either by position or by name. Practice both ways. Do you have a preference? If so why?

9. In addition to extracting rows and columns from your dataframe by position 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!):

- at depths greater than 1200 m
- gradient steeper than 200 degrees
- water noise level of 'low'
- water.noise level of 'high' in the month of 'May'
- month of 'October', water noise level of 'low' and gradient greater than the median value of gradient (132)
- all observations from between latitudes 60.0 and 61.0 and longitudes -6.0 and -4.0
- all rows that do not have a water noise level of medium

10. A really neat feature of extracting rows based on conditional statements is that you can include R functions within the statement itself. To practice this, modify your answer to the gradient question in Q9 to use the `median()` function rather than hard coding the value 132.

11. However, when using functions in conditional statements you need to be careful. For example, write some code to extract all rows from the dataframe `whale` with depths greater than 1500 m and with a greater number of whales spotted than average (hint: use the `mean()` function in your conditional statement). Can you see a problem with the output? Discuss the cause of this problem with an instructor and explore possible solutions.

12. Although you have concentrated on using the square bracket `[ ]` notation to extract rows and columns from your dataframe, there are of course many other approaches. One such approach is to use the `subset()` function (see `?subset` or search for the `subset` function in the Introduction to R book to find more information). Use the `subset()` function to extract all rows in 'May' with a time at station less than 1000 minutes and a depth greater than 1000 m. Also use `subset()` to extract data collected in 'October' from latitudes greater than 61 degrees but only include the columns `month`, `latitude`, `longitude` and `number.whales`.

13. Another useful way to manipulated your dataframes is to sort the rows based on the value of a variable (or combinations 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 Q14 in Exercise 2. Let's practice with a straight forward example. Use the `order()` function to sort all rows in the `whale` dataframe in ascending order of depth (shallowest to deepest). Store this sorted dataframe in a variable called `whale.depth.sort`.

14. Now for something a little more complicated. Sort all rows in the `whale` dataframe by ascending order of depth within each level of water noise. The trick here is to remember that you can order by more than one variable when using the `order()` function (see Section 3.4.3 again). Don't forget to assign your sorted dataframe to a new variable with a sensible name. Repeat the previous ordering but this time order by **descending** order of depth within each level of water noise.

15. 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 straight forward :

```
mean(whale$time.at.station)     # mean time at station
median(whale$depth)             # median depth
length(whale$number.whales)     # number of observations
```

15. (cont) Perhaps more interestingly, you might want summarise one variable conditional on the level of another factor variable. For example, write some R code to calculate the mean number of whales sighted at each of the three levels of water noise (see Section 3.5 for a few hints). Next, calculate the median number of whales sighted at each level of water noise and for each month.

16. Another useful function for summarising dataframes is `aggregate()`. Refer back to the book (search for `aggregate`) to look up how to use this function (or see `?aggregate`). Use the `aggregate()` function to calculate the mean of time at station, number of whales, depth and gradient for each level of water noise (don't forget about that sneaky NA value). Next calculate the mean of time at station, number of whales, depth and gradient for each level of water noise for each month. (Optional): For an extra bonus point see if you can figure out how to modify your previous code to display the mean values to 2 decimal places rather than the default of 3 decimal places.

17. Knowing how many observations are present for each factor level (or combinations of factor levels) 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 observations for each level of water noise (see Section 3.5 again for more information). Next use the same function to display the number of observations for each combination of water noise and month. (Optional): The `xtabs()` function is very flexible for creating tables of counts for factor combinations (aka contingency tables). Take a look at the help file (or Google) to figure out how to use the `xtabs()` function to replicate your use of the `table()` function.

18. Ok, we have spent quite a bit of time (and energy) learning how to import and manipulate dataframes. The last thing we need to cover is how to export dataframes from R to an external file (see Section 3.6 of the book for more details). Let's say you want to export the dataframe `whale.num` you created previously (see Q8) to a file called 'whale_num.txt' in your `output` directory which you created in Exercise 1. To do this you will need to use the `write.table()` function. You want to include the 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 create your file, try to open it in Microsoft Excel (or open source equivalent).

End of Exercise 3