Intro to R - Day 3 in-class exercise: working with data

2023-09-28

For this exercise, you are given a data set of wild songbirds mist-netted near Loch Lomond from 2009 to 2012 and examined for ticks. As a motivating research question, we will be comparing the average number of ticks found on different bird species. The main purpose of the exercise is to illustrate useful commands for manipulating and organising your data, further adding to your existing R repertoire. Throughout the script, you should verify if your code is free of bugs, produces the intended outputs, is concise, and self-contained in its own project folder. Avoid cluttering your report with all the values in your data set or including unnecessary code or annotations.

1. Exploring your data

Start by download the data set from Moodle (`ticks_birds_2009-2012.csv`) and import it under the name `tick_data`. Explore what the data look like using R functions you know. In addition, combine the functions `group_by()` and `summarise()` to find out

a) how many birds of each species were trapped in total.
b) How many birds of each species were trapped each year

Tip 1 : You previously learned how to calculate another summary statistic, the mean of a group. Here, you want the number of observations, which can be retrieved using `length()` instead of `mean()`. Or you can use another function from dplyr called `count()`.

Tip 2: You won't be able to see all species because the default when printing a tibble (dplyr's version of a data frame) is to only show the first ten rows. You can overwrite this with `options(tibble.print_max = Inf)`.

2. Adding and re-defining variables

a) Adding a new variable

Look at the two variables `wt_with_bag` and `bag_wt`. The first one represents the weight of the bird including the handling bag. The second is the weight of the bag if one was used – this depends on the bird species and for some the use of a bag was not necessary. For any analysis involving the actual weight of the bird (i.e., total weight minus that of the bag), it would be handier to have this already defined as its own variable. Use what you have learned about transforming data columns (using `dplyr`) to create a new variable called `bird_wt` by subtracting `bag_wt` from `wt_with_bag`.

b) Creating a new date variable

Note how dates in your data set are represented by three separate columns: `day`, `month`, `year`. This might be useful for some data manipulations (as we will see later on). But we would also like to have a variable that represents the actual date for each observation. Apply what you learned in previous sessions about defining dates using the `lubridate` package to create such a new variable called `date`.

You will need to do this in two steps:

First, create a variable called `date` that contains the concatenated values from the `day`, `month`, and `year` columns. I would recommend using the function `paste()` to do this. Look up the help file for this function to work out how it works.

Secondly, use the `lubridate` package to turn this variable `date` into one that R recognises as a date.

c) Recoding an existing variable

Sometimes, we might want to change how the values of a variable are coded. Find the variable named 'recap', which indicates whether an animal has been caught before (indicated by '1') or is caught for the first time ('0'). You'll notice that the program interpreted this variable as numeric with integer values (zeros and ones). However, for later analyses it will be useful to have `recap` defined as a factor with the levels "yes" and "no" instead.

We can use the recode command (part of the `dplyr` package) to redefine the two factor levels as "yes" and "no". This is how it works:

```
recode(recap, '1'="yes", '0'="no")
```

However, this by itself will give you an error message (do you understand why?). Combine this bit of code with the same approach you used in part a) to transform data columns. Replace the current data column `recap` with another one of the same name, in which the factor levels are shown as 'yes' and 'no' instead of '0' and '1'.

Note that we need to tell R here that this variable should be treated as a factor (with two unique values representing the different factor level, 'yes' and 'no') rather than a character variable.

```
tick_data$recap <- as.factor(tick_data$recap)
```

3. Sorting

When looking at the first few lines of your data, you will find that they are currently ordered by species. We may want to order the data frame by another variable or by several other variables at once. There are multiple ways to sort in R; so far you have learned how to do this using the dplyr function `arrange()`.

a) Using the `arrange()` function from dplyr

Use `arrange()` to simultaneously sort your data first by `date` and then by `species`. Check the help file to find out how this is done and then how you would change your order from increasing to decreasing.

Tip: avoid looking at the entire data set to confirm that it worked (or at least don't make this part of your script). Instead, use some of the data exploration tools you've learned about to look at parts of your data.

b) Using the `order()` function from base R

An alternative to do this using base R is with the command `order`. To see how it works just type

```
head(order(tick_data$date))
```

This returns the row numbers for ordering the data set alphabetically by species (`head` is helpful because limits this to the first six values). The full vector of row numbers (`order(tick_data$date)`) can be combined with the square brackets for indexing to re-order our entire data set

```
tick_data <- tick_data[order(tick_data$date),]
```

Indexing was mentioned in the first day lecture and will also be covered in section 5 of this tutorial. You can also find out more information about it here

http://www.cookbook-r.com/Basics/Indexing_into_a_data_structure/

Remember that within the square brackets, the entry before the comma `(order(tick_data$species))` represents the rows, while the entry after the comma represents the columns. Since no specific columns are identified here (the space after the comma is empty), this means 'all rows'.

Rather than creating a new object for this, we simply replaced our previous version of the data set with the re-ordered one.

If you want to change your order from increasing to decreasing, this is done by adding `rev` in front, e.g. `tick_data <- tick_data[rev(order(tick_data$date)),]`

While this will seem a lot more cumbersome than using the dplyr function `arrange()`, it will give you an appreciation of indexing using square brackets, which is important to know about (you will practise this more further down).

Using either ordering method, finish by putting the data into chronological order.

4. Subsetting your data

As mentioned in the beginning, for this exercise we are interested in the number of ticks different birds were infested with. However, if you look at the first couple of lines in your data set you'll notice that none of these birds had any ticks. This is explained by the simple fact that the data are ordered chronological and that ticks aren't very active in Scotland during January. So before moving on, it would be sensible to create a subset of the data that only includes observations from the period of the year that coincides with tick activity. Recall how you can use `dplyr` functions to create subsets of your data. Use this knowledge to create a new data frame that only includes observations from April through October and call it `tick_season`.

Check that it contains the right number of observations (it should have 6381). **Continue the exercise by working with this new data subset**.

5. Dealing with NAs

As with many 'real' data sets, we have some missing values in our data. Maybe the same bird had just been caught before so there was no reason to collect the data again, maybe it wasn't possible to take certain measures from that bird. Whatever the reason, it is important to know how to deal with these missing values because they can affect your calculations down the line. For example, try calculating the **median** number of ticks ('ticks') for each of the bird species using `group_by()` and `summarise()` just as you learned in chapter 3 of the book. Call this new variable 'median_tick'. Note the number of entries showing `NA` ("Not Available") in the results. This occurs because even one missing value prevents the proper calculation of the median for that species.

You can find lots of resources about how to deal with `NA` values on the web; for a quick overview, take a look at the 'Quick-R' website:

http://www.statmethods.net/input/missingdata.html

Try to find out how you would calculate a median when you have NA values. Once you found the solution, redo your calculation and store your results in a new object called `tick_summary`. Once you are happy that all NA values have gone, repeat the same type of calculation for the **mean** number of ticks and add this to `tick_summary`.

Finally, order the rows in `tick_summary` so that they are sorted by decreasing mean tick number (so species with the most ticks on top). This is the same thing we did previously, so if you're unsure, just check your code from above on how to do this.

Check your result to make sure you succeeded.

6. Indexing

You often want to access a particular element of your data, which can be achieved using indexing using square brackets. You already saw how to use this in the context of ordering your data but indexing can be useful for many things. To practice, use indexing to extract the following elements from `tick_summary`.

- the species name and the median number of ticks in the 9th row
- the name of the 5th species
- a list of the species for which median tick burden was greater than 0
- the mean number of ticks found on dunnocks (can you do it without looking up the correct row number?)

Try to make your code robust - so instead of referring to a particular column by its number (which could change in the future and isn't very self explanatory), use the column name (e.g. `median_tick`).

7. Plotting the data

Finally, let's create some plots of our data. We want to start by visualising the mean number of ticks per bird we stored in `tick_summary` as points using `ggplot2`.

You will note that the plot is not showing the species ordered by mean tick number, even though we ordered our data in this way. This has to do with the way ggplot2 handles the data: it will plot the values according to the order in which the **factors levels** for species are stored, which are still in the default alphabetical order. We can reset the order of the factor levels according to how the rows in our data are ordered now

```
tick_summary <- tick_summary %>%
  mutate(species = factor(species, levels = species))
```

When replotting this graph, you should flip the axes using `coord_flip()` to also make the species labels easier to read.

The resulting plot gives us a general idea about which species tended to have high tick burdens and which ones were rarely infested by ticks. However, it suffers from the same problem we encounter when using barplots (see p. 85 in the book and lecture notes) – we are only looking at a single summary statistic (in this case the mean) which tells us nothing about the underlying distribution of the data. For that, we should return to our original data set `tick_season` and to use the kind of visualisation tools in `ggplot2` you encountered in the R book.

The first thing to do is to reorder the factor levels for 'species' within `tick_season`, similar to what we did before. Can you modify above's code to do that? Remember that the information for how the species names should be ordered is found in `tick_summary`, not in `tick_season`.

Once that is working, you should create two types of plots. For both types, your script/report should only contain the final version - so if you are trying out different options, make sure to remove those preliminary trials from the script handed in. Your final plots should also be well labelled and formatted and be clear and informative, while avoiding redundancies (i.e. not showing the same information multiple times).

1) Create a plot that shows both median values and the distribution of the actual data points in the same plot (for example using a combination of boxplots and points). Due to the large number of observations, `geom_jitter` will work better than `geom_points` here. Try to make effective use of colour in your plot. You could add the mean value too though that is a bit more advanced.
2) Use 'facet wrap' option you encountered at the end of chapter 2 in the book to create a second plot. This could be again using tick burden as the variable to plot but should look different from the first plot (e.g. a histogram instead of boxplot/raw data) or use a different variable. Just play around to find something that seems useful.

8. Final recommendations

Your final scripts should be complete, well organised and structured (use markdown headings), and provide annotations and background explanations to help the reader understand what is happening and why. As a general rule, use the `#` symbol for directly annotating your code inline (e.g. why this function, why are things coded in the way they are, etc) and use plain text for broader context (what is the purpose of the analysis, what are the different sections of the exercise about).