Exercises

Exercise 2 : Basic R operations

This exercise will help you get familiar with performing some basic (but useful) operations with R in RStudio. You'll first practice using some of R's inbuilt mathematical operators and then move on to creating vectors using a variety of R functions. You'll also practice summarising and manipulating vectors (including vectorisation) and finally we'll take a look at how to deal with missing values in R. To help you through this exercise please read Chapter 2 of the Introduction to R book.

1. Open up your RStudio Project from Exercise 1 and either create a new R script or continue with your previous R script. Make sure you include any metadata you feel is appropriate (title, description of task, date of creation etc). Don't forget to comment out your metadata with a `#` at the beginning of the line.

2. Let's use R as a fancy calculator. Find the natural log, log to the base 10, log to the base 2, square root and the natural antilog of 12.43. See Section 2.1 of the Introduction to R book for more information on mathematical functions in R. Don't forget to write your code in RStudio's script editor and source the code into the console.

3. Next, use R to determine the area of a circle with a diameter of 20 cm and assign the result to a variable called `area_circle`. Google is your friend if you can't remember the formula! Also, remember that R already knows about `pi`.

4. Now for something a little more tricky. Calculate the cube root of 14 x 0.51. You might need to think creatively for a solution (hint: think exponents), and remember that R follows the usual order of mathematical operators so you might need to use brackets in your code (see this page if you've never heard of this).

5. Ok, you're now ready to explore one of R's basic (but very useful) data structures - vectors. A vector is a sequence of elements (or components) that are all of the same data type (see Section 2.4 and Section 3.2.1 for an introduction to vectors). Although technically not correct it might be useful to think of a vector as something like a single column in a spreadsheet. There are a multitude of ways to create vectors in R but you will use the concatenate function `c()` to create a vector called `weight` containing the weight (in kg) of 10 children: `69, 62, 57, 59, 59, 64, 56, 66, 67, 66` (Section 2.3 shows you how to do this).

6. You can now do stuff to your `weight` vector. Get R to calculate the mean, variance, standard deviation, range of weights and the number of children of your `weights` vector (Section 2.3). Next, extract the weights for the first five children and store these weights in a new variable called `first_five`. Remember, you will need to use the square brackets `[ ]` to extract (aka indexing, subsetting) elements from a variable. Section 2.4.1 introduces using the `[]` notation.

7. We're now going to use the the `c()` function again to create a vector called `height` containing the height (in cm) of the same 10 children: `112, 102, 83, 84, 99, 90, 77, 112, 133, 112`. Use the `summary()` function to summarise these data. Extract the height of the 2nd, 3rd, 9th and 10th child and assign these heights to a variable called `some_child`. Also extract all the heights of children less than or equal to 99 cm and assign to a variable called `shorter_child`.

8. Now you can use the information in your `weight` and `height` variables to calculate the body mass index (BMI) for each child. The BMI is calculated as weight (in kg) divided by the square of the height (in meters). Store the results of this calculation in a variable called `bmi`. Note: you don't need to do this calculation for each child individually, you can use both vectors in the BMI equation – this is called vectorisation (see Section 2.4.4 of the Introduction to R book).

9. Now let's practice a very useful skill - creating sequences (honestly it is...). First use the `seq()` function to create a sequence of numbers ranging from 0 to 1 in steps of 0.1 (this is also a vector by the way) and assign this sequence to a variable called `seq1`. If you're unsure how to do this then see Section 2.3 of the book for more information.

10. Next, create a sequence from 10 to 1 in steps of 0.5 and assign to a variable called `seq2` (Hint: you may find it easier to include the `rev()` function in your code).

11. Let's go seqeunce crazy! Generate the following sequences. You will need to experiment with the arguments to the `rep()` function to generate these sequences (see Section 2.3 for some clues):

    - 1 2 3 1 2 3 1 2 3
    - "a" "a" "a" "c" "c" "c" "e" "e" "e" "g" "g" "g"

- "a" "c" "e" "g" "a" "c" "e" "g" "a" "c" "e" "g"
- 1 1 1 2 2 2 3 3 3 1 1 1 2 2 2 3 3 3
- 1 1 1 1 1 2 2 2 2 3 3 3 4 4 5
- 4 sevens, 3 twos, 1 eight and 5 ones

12. Ok, back to the variable `height` you created in Q7. Sort the values of `height` into ascending order (shortest to tallest) and assign the sorted vector to a new variable called `height_sorted`. See Section 2.4.3 for an introduction to sorting and ordering vectors. Now sort all heights into descending order and assign the new vector a name of your choice.

13. Let's give the children some names. Create a new vector called `child_names` with the following names of the 10 children: `"Alfred", "Barbara", "James", "Jane", "John", "Judy", "Louise", "Mary", "Ronald", "William"`.

14. A really useful (and common) task is to sort the values of one variable by the order of another variable. To do this you will need to use the `order()` function in combination with the square bracket notation `[ ]` (see Section 2.4.3 of the book for more details). Create a new variable called `names_sort` to store the names of the children sorted by child height (from shortest to tallest). Who is the shortest? who is the tallest child? If you're not sure how to do this, please ask one of the instructors.

15. Now order the names of the children by descending values of weight and assign the result to a variable called `weight_rev`. Who is the heaviest? Who is the lightest?

16. Almost there! In R, missing values are usually represented with an `NA`. Missing data can be tricky to deal with in R (and in statistics more generally) and cause some surprising behaviour when using some functions (see Section 2.4.5 of the Introduction to R book). To explore this a little further let's create a vector called `mydata` with the values `2, 4, 1, 6, 8, 5, NA, 4, 7`. Notice the value of the 7^{th} element of `mydata` is missing and represented with an `NA`. Now use the `mean()` function to calculate the mean of the values in `mydata`. What does R return? Confused? Next, take a look at the help page for the function `mean()`. Can you figure out how to alter your use of the `mean()` function to calculate the mean without this missing value?

17. Finally, list all variables in your workspace that you have created in this exercise. Remove the variable `seq1` from the workspace using the `rm()` function.

End of Exercise 2