Introduction to R programming: Exercises

Florian Huber

April 24, 2026

Vectors and data types

Exercise 1

Run each of the following lines of code separately. After running each line, predict what the value for `heights` will be. Check by evaluating `heights` in the console.

```
heights <- c(180.5, 165)
heights - 10
heights <- heights - 10
```

Exercise 2

Assign a number of your choice to the variable a. Then create a variable b that contains four elements, for example `a, a + 2, 3, 72`. Subtract 10 from `b` and assign the outcome to a variable named `my_first_result`. Look at the output. What happened? Was 10 subtracted from all or only the first element? Next, create a vector `v` containing the numbers `-3` and `-50`. Multiply `my_first_result` with `v` and assign the result to `my_second_result`. Predict what the outcome will be. Use the functions `sd()`, `max()`, `median()` on a vector of your choice. What do these functions do?

There are two very useful tricks when writing R code. The first one is called autocompletion. Type `my_` in your script file or the console and then hit the tab button (tabulator). What happens? The second trick is useful when working in the console: go to the console and use the up arrow on your keyboard. What happens?

Exercise 3

We will now create some hypothetical RT-qPCR data. Note that one does not usually type the data into R manually but by using data import functions, which we will cover later.

Experimental setting: The yeast gene *IME4* has been shown to have an antisense transcript that originates in its terminator region. We want to measure the transcription of this gene! All measurements were done in quadruplicate and we have controls without reverse transcriptase (-RT).

Task: Create a vector called `gene` which contains "IME4" 8 times. Check the documentation of `rep()` to make your typing life easier. What is the type of this vector? Next, we need a vector called `RT` which indicates whether the sample was run with our without reverse transcriptase. It should contain first 4 `TRUE` and then 4 `FALSE` values. What is the type of this vector? Finally, manually create a vector `CT` containing Ct values: `22.79, 22.09, 23.01, 22.60, 36.24, 35.8, 38.3, NA`.

Ct values need to be converted to gene expression values. Do this by saving the result of `2 ^ (-CT)` to a new variable `genex`. What do you think will happen to the NA value?

We learned that vectors can only contain values of a single type. What happens if you try to combine vectors of different types? For example, if you run `c(RT, CT)`. Try also other combinations! What happens if you try to calculate the `sum()` or the `mean()` of the `RT` vector?

Exercise 4

Create two named numerical vectors of your choice and then experiment with the comparison operators `==`, `<`, `>`, `<=`, and `>=`. Then remove the names from one of your vectors. Check if the name removal worked by re-evaluating your variable in the console afterwards.

Exercise 5

In which directory are you at the moment? What are its contents? What variables are in your workspace? Delete one of your variables. Check out the "Files" and "Environment" tabs in RStudio while you are doing these things.

Vector subsetting

Exercise 1

Use the vectors from exercise 3 in section "Vectors and data types". You can also load them by running `load("./data/vectors_exercise4.RData")`. They are called `gene`, `RT`, `CT`, and `genex`.

Experiment by extracting values from the vector `CT`, for example by extracting the first, fifth, and second value. Remember that you can extract the same value several times (how?). Next subset the vector `CT` and keep only those values where `RT` is true (so only the samples with reverse transcriptase). Which `genex` value was the highest one? And the lowest? There are two solutions, either using `which.max()` or `max()`. Note that if you use `max()`, you have to add the argument `na.rm = TRUE`.

Next, make a cleaned version of CT and genex that does not contain NAs. You can either index by hand or use the function `is.na()` for help. After that: How many CT samples were smaller than 30? How many were greater than 22 and smaller than 30?

Exercise 2

R comes pre-loaded with a number of variables and data sets. When you type `islands` into the console you get a named vector containing the areas of the earth's landmasses that exceed 10,000 square miles (how did I find out? - by typing `?islands` to get to the help page).

How large is Greenland? And how many times larger is Africa than Britain? Next, use `sort()` to sort the landmasses by size. After that, extract all landmasses with more than 50,000 square miles in size. What is the largest landmass? What is the median landmass size?

Bonus exercise, if you are motivated and have enough time: Replace the names of all the landmasses by their version in all caps (i.e. "Africa" –> "AFRICA"). You will need the functions `names()` and `toupper()` and run an assignment to achieve that. Reading the examples in the help file for the `names()` function also helps (`?names`).

Exercise 3

Generate 20 observations from a normal distribution with mean 0 and standard deviation 3 using `x <- rnorm(n = 20, mean = 0, sd = 3)`. Let's use these data to practice modification of values. For example:

- Change the second, third and fourth values to 0.5.
- Change the first and the last value to 0. (Hint: you can use the `length()` function to get the last value)
- Change the second value to `NA`.
- Change all values that are smaller than 0 to 0.
- Replace the first three values by three values from the uniform distribution (`runif(3)`).

Feel free to experiment on your own.

Exercise 4

Assume you have a vector v: `v <- 1:10`. Replace all elements that are greater than 5 by their square root. Remember that modification consists of two steps (i) extracting the values that you want to modify and (ii) replacing those values.

Lists

Exercise 1

Lists can be useful to save diverse types of data in one container. Therefore, they are often the basis for the more complex data structures in R.

For example, let us assume that you made five measurements and saved them in a variable `measurements = c(1.8, 1.79, 1.82, NA, NA)`. In addition you recorded the day that the measurement was recorded in a `day` variable as well as the species that your samples came from (variable `species`).

Tasks:

- Put the information above into a list. Name the list elements accordingly.
- How can you get a sublist that contains only the metadata, i.e. the day and time and the species information but not the measurements?
- How can you access the measurements vector so as to calculate the mean?
- How can you change the last two elements of the list elements containing your measurements from `NA` to 1.85 and 1.77?

Data frames

Exercise 1

Load the dataset "mtcars" by running `data("mtcars")`. What is it about (`?mtcars`)? Check the output using `head(mtcars)`.

To find out how many rows and how many columns a table has, we can use the functions `nrow()` and `ncol()`, respectively. Moreover, there are functions to find out which column and which row names a data frame has: `rownames()` and `colnames()`. We did not cover row names in the course so far and in general I would recommend to not use row names. Therefore, in this exercise, we will delete the row names and put them into a separate column before practising subsetting. For this, run the following:

- `mtcars$carname <- rownames(mtcars)`
- `rownames(mtcars) <- NULL`

Next, we will do a bit of table subsetting. First, subset one or several rows of your choice, keeping all columns. Then do the same but keeping only a few selected columns. Make a subtable containing only the observations (rows) that have a displacement (variable disp) greater than 150. Use `nrow()` on the subtable to find out how many observations this corresponds to.

Now make another subtable containing only the cars that have a displacement greater than 150 and which weigh more than 2.8 tons. Do the same using *or* instead of *and*

What is the smallest displacement for cars that weigh more than 2.8 tons? Hint: you need to first get the column containing displacement as a vector. Then you need to subset this vector so as to keep only those elements that correspond to cars with a weight greater than 2.8 tons. Then run function `min()` on the resulting vector.

In the US, fuel consumption is measured in miles per gallon (mpg). Convert this to litres per 100 km (lpkm) and assign it to a new column. The conversion is: litres per 100 km = `378 / (1.61 * miles.per.gallon)`.

What is the mean consumption of cars with 6 cylinders or more? What about cars with exactly 4 cylinders? How many cars have exactly 5 gears and at least 6 cylinders?

Matrices

Exercise 1

Create a numeric 3 x 3 matrix with the values of your choice (function `matrix()`). Is the matrix filled up "row by row" or "column by column"? Next, add 2.5 to every element. What is the sum of all matrix elements? Extract all elements of the first column. Set all elements of the 3rd row to 42. Calculate the sum for every column using `colSums()`.

Writing functions

Exercise 1

The standard error of the mean (s.e.m.) is the standard deviation (function `sd()`) divided by the square root (function `sqrt()`) of the number of observations (function `length()` applied to a vector). Write a function called `sem()` that calculates the standard error of a given vector.

Data import and overview

Exercise 1

We used a function from the `readr` package to import data:

```
(cells <- read_delim("./data/ImagedCells.csv", delim = ";"))
```

Task 1: Use the base R function `read.table()` to import your data. Check the output - what arguments do you need to change to get the import right? Using `?read.table` will help you to find out.

Task 2: There are a number of useful functions to get an overview of your data. Run

```
library(tidyverse)
```

and then check out the following functions by executing them on the table imported in the previous exercise or on one of the columns contained in that table.

These are some of the functions you will need a lot: Explore them by just "trying out" or, if you run into troubles, by reading the documentation and then trying out. `head()`, `summary()`, `str()`, `View()`, `range()`, `sort()`, `duplicated()`, `quantile()` (bonus: change the argument `probs`), `unique()`, `n_distinct()`, `table()` (use with one or two categorical variables. bonus: pass two vectors instead of just one: what happens?), `mean()`, `median()`, `colnames()`, `rownames()`, `cor()`, `cor.test()`.

You should be able to answer the following questions: How many wells were imaged? How many cells were imaged in each of the two wells? Are there any missing values?

Creating plots

Exercise 1

Subset the `cells` table by keeping only those cells that were in well "A1" and that have an area smaller than 600 (bigger cells are "weird"). Then plot that data and run a linear regression of GFP vs. mCherry at time point 0. Change the `method` argument in `geom_smooth()` to "loess" and see what happens.

Exercise 2

Use again the whole `cells` data set and generate a scatter plot of `gfp_T0` vs. `mCh_T0` and use the `colour` aesthetic to distinguish the different wells.

In addition, ggplot2 also understands other aesthetics such as shape, size, and alpha. Map other aesthetics to `Well` and see what happens. Can you map more than one aesthetic to the same variable?

Next, what if you do not want use colour to distinguish groups but rather you want to have red points instead of black ones. To do this you need to set `colour = "red"`. This has to happen in the `geom_*` call but *outside* of the `aes()` command. How can you change the shape that is displayed to e.g. hollow circles?

Finally, save the plot you created to disc by using `ggsave()`.

Exercise 3

How are cell sizes distributed in the data set? Plot a histogram first and play around with different values for the binwidths or change the number of bins. Then check if the cell sizes are different between the two wells by generating appropriate plots (e.g. box plots, violin plots etc.). Can you plot the actual measurement points on top of the box plot? For additional visual niceness, paint the box plots in different colours for each well (use the fill and the colour aesthetic for that).

We have seen that ggplot will automatically pick colours for us when mapping the colour aesthetic to a variable in the data. We can actually change the chosen colours by adding `scale_colour_manual()` to the plot. Can you change the colours for the points from their defaults to, for example, blue and orange for the two wells? Hint: go to https://ggplot2.tidyverse.org/reference/ . Then look for the entry for `scale_colour_manual()` and scroll down to see examples. Adapt the code to your plot!

Exercise 4

Using the cells data, display the distributions of the variable of your choice using a histogram. Distinguish the variable `treated` by mapping to `colour` or `fill`. Next, set the alpha parameter to 0.5. Are the histograms representing the two groups stacked on top of each other or does one histogram overlap the other? Experiment by **setting** the argument `position` to "identity" or "fill". What happens?

Exercise 5

The data for this exercise has been adapted with permission from materials by Bernd Klaus.

For this exercise we will load a dataset containing single-cell transcriptomics data. It is in the form of a count matrix in which each row represents a gene and each column represents one cell.

```
load("./data/mtec_counts.RData")
```

Now create a subtable containing only the first three rows and the first 9 cells. Then tidy the data = turn them into a "long format" so that one column contains the ensembl_id, the second the cell identifier ("cell2", "cell17" etc.), and the third column should contain the counts.

Bonus exercise: use the tidied up table to create a barplot that shows on the x-axis the ensembl_ID and on the y-axis the counts. Display the information of how many counts belong to which cell by mapping the "fill" aesthetic to the cell variable. Set the position argument to "dodge": what happens?

Exercise 6

In many applications you want to display some kind of summary statistic together with an error bar. There is a dedicated geom for displaying errors: `geom_errorbar()`. We would like to create a barplot (`geom_col()`) that shows us the average mCherry/GFP ratio per well. The barplot should also contain error bars that show +- the standard deviation. For this we need a table with one row per well and with columns containing the average mCherry/GFP ratio as well as the lower and upper value for the error bars.

Task: Generate such a table by using `filter()`, `mutate()`, `group_by()` and `summarise()` to do the following for each well in the cells data set:

- First, only keep those cells whose `gfp_T0` value is smaller than 380.
- Then add a new column called `ratio` that contains the ratios of mCh_T0 divided by gfp_T0.
- `summarise()` the `grouped_by()` data (grouped by well) such that you get a column containing the mean of `ratio`, a column for its standard deviation (s.d.), one for `mean_ratio` minus the s.d. and one for `mean_ratio` plus the s.d.

Tip: First try to solve steps 1 and 2 separately. Then use `group_by()` together with `summarise()` to perform the third step. You can either save the results of each step in different variables or chain the steps together with the pipe operator: `%>%`.

Then use your output to plot a bar plot displaying the two wells together with their error bars.

Map

Exercise 1

A common scenario in the lab: you have run a series of plate reader experiments and for each run the plate reader produced a separate Excel file, which you can find under `./data/plate_reader/`. The files are numbered from 1 to 4: `plt_reader1.xlsx`, `plt_reader2.xlsx` etc.

You can get all the file names by running `my_filenames = list.files("./data/plate_reader", full.names = TRUE)`.

Your tasks:

- Import the plate reader files into R, assigning each table to a separate variable. You will need to load the readxl library (`library(readxl)`) first. If you don't have the library yet, install it (`install.packages()`). Find out which function you have to use, either by searching the internet or by browsing the functions available in the package (tab 'Packages' in RStudio).
- Since the tables all have the same structure, "row-bind" them into one big table. Use the function `bind_rows()` for that. You will need to load the tidyverse or dplyr library first.
- Now let's simplify our task according to the "DRY" principle. Use the `my_filenames` variable from above, the `map()` function and the Excel import function to load all of the tables with just one command. `map()` will return the tables as a list with each list element being a data frame (tibble).
- Combine the data frames from that list into one big data frame. You can again use the `bind_rows()` function for that.

License