R Crashcourse

Learning how to swim in the data ocean

Anıl Axel Tellbüscher & Bisa Saraswathy

Anıl Axel Tellbüscher

- 3rd Ph.D. student at University of South Bohemia, CZ
- Biostatistics lecturer since 2023
- Freelance Data Analyst and R tutor
- EAS-SG since 2019
- E-Mail: atellbuscher@jcu.cz
- Web: https://anil.tellbuescher.online

Why R?

We **look at spreadsheets** while we **read code.**

- It is easy to follow what code does
  - step-by-step process
  - can be enriched by detailed explanations
- Code is reproducible
  - it does the same on every computer

Example for a spreadsheet

Velázquez-Martínez et al. (2019): Statistical Entropy Analysis as tool for circular economy: Proof of concept by optimizing a lithium-ion battery waste sieving system. *Journal of Cleaner Production*, 212, p.1568-1579.

Example for code

```r
# Load data
rawdata <- mtcars

# Remove boring cars
cool_cars <- dplyr::filter(rawdata, hp > 100)

# Sort by horsepowers in descending order
cool_cars <- dplyr::arrange(cool_cars, desc(hp))

# Print the first few rows
head(cool_cars)
```

```
                  mpg cyl disp  hp drat    wt  qsec vs am gear carb
Maserati Bora    15.0   8  301 335 3.54 3.570 14.60  0  1    5    8
Ford Pantera L   15.8   8  351 264 4.22 3.170 14.50  0  1    5    4
Duster 360       14.3   8  360 245 3.21 3.570 15.84  0  0    3    4
Camaro Z28       13.3   8  350 245 3.73 3.840 15.41  0  0    3    4
Chrysler Imperial 14.7  8  440 230 3.23 5.345 17.42  0  0    3    4
Lincoln Continental 10.4 8 460 215 3.00 5.424 17.82  0  0    3    4
```

Good learning resources

- R documentation
- StackExchange
- ChatGPT
- (E-)books
- learning platforms such as DataCamp

Common causes for errors

- missing `,`
- missing `)`, `}`, or `]`
- `NA` (= missing data)
- `x` and `y` differ in length
- wrong data type
- outdated R version

R as calculator

R as calculator 1

R can perform all arithmetic operations such as addition, subtraction, multiplication, and division.

```
1  1+1 # addition
```

```
[1] 2
```

```
1  2-1 # subtraction
```

```
[1] 1
```

```
1  2*3 # multiplication
```

```
[1] 6
```

```
1  4/2 # division
```

```
[1] 2
```

```
1  2^2 # exponentiation
```

```
[1] 4
```

R as calculator 2

It is also possible to logarithmize or square root.

Logarithms

```
log(exp(2)) # logarithm: ln()
```

```
[1] 2
```

```
log(4, 2) # logarithm: base 2
```

```
[1] 2
```

- default: Logarithmus naturalis!

Roots

```
sqrt(4) # square root
```

```
[1] 2
```

```
8^(1/3) # cube root
```

```
[1] 2
```

R as calculator 3

Euclidean division

```
1  100 %/% 3
```

```
[1] 33
```

Modulo

```
1  100 %% 3
```

```
[1] 1
```

Objects in R

In R, values can be assigned to objects using the arrow `<-`

```
1  a <- 1
2  a
```

```
[1] 1
```

Shortkeys in RStudio

OS	Shortkey
Windows	`ALT` + `-`
macOS	`OPTION` + `-`

Data types

- data in R can have different types
  - `numeric`
  - `character`
  - `logical`
  - `factor`
  - ...

R was developed by statisticians. The data types directly translate into statistical scale levels!

Data types 2

- the type of data can be assessed with the `class()` function.

```
1 a <- 1
2 class(a)
```

```
[1] "numeric"
```

- we can also assign text to an object.

```
1 b <- "fish"
2 class(b)
```

```
[1] "character"
```

Data types 3

- we can also conduct logical tests

```
1  a == 1
```

```
[1] TRUE
```

```
1  class(a == 1)
```

```
[1] "logical"
```

Operator	Function
==	is equal
!=	is unequal
>= / >	greater or equal / greater
<= / <	less or equal / less

Operator	Function
`%in%`	is contained in/is part of
`&`	and (for multiple comparisons)
`|`	or

Data types 4

- let’s try something different.

```
1  a + 2
2  a + a
3  a + TRUE
4  a + FALSE
5  a + "2"
```

Question 1

How to create a numeric variable `x` with a value of 10?

a. `x = 10`

b. `x == 10`

c. `x <- 10`

Question 2

How to create a character variable `fish` with the value "Oncorhynchus mykiss"?

a. `fish = Oncorhynchus mykiss`

b. `fish == "Oncorhynchus mykiss"`

c. `fish = "Oncorhynchus mykiss"`

Question 3

How to create a logical variable `carnivorous` with the value `TRUE`?

a. `carnivorous = True`

b. `carnivorous == true`

c. `carnivorous = "TRUE"`

d. `carnivorous == "TRUE"`

Question 4

```
a <- TRUE, b <- "2", c <- 3
```

Which code works?

a. a + b = FALSE

b. a + c = 4

c. b + c = 5

Data structures

```
1 length(a)
```

```
[1] 1
```

- the objects we created so far were actually vectors with a length of 1

Vectors 1

- we can assign an object more than one value using the `c()` function.

```
1  vec1 <- c(5,1,7,3,2,7)
2  vec1
```

```
[1] 5 1 7 3 2 7
```

```
1  vec2 <- 1:6
2  vec2
```

```
[1] 1 2 3 4 5 6
```

Vectors 2

- what if we want to change a value within the vector? For instance, the second one?
- for this we use [ ]

```
vec1[2] <- 2
vec1
```

```
[1] 5 2 7 3 2 7
```

- square brackets are exclusively used for indexing!

$$x_i$$

where i denotes the index, which is the position of an observation in a vector.

Question 5

You want to create a numeric vector x containing the numbers from 1 to 9. Which code is wrong?

a. x = 1:10

b. y = c(1,2,3,4,5,6,7,8,9)

c. x = seq(1,9)

Question 6

How to extract the 5th element of the vector you just created?

a. x[5]

b. x(5)

c. x{5}

d. x[5,]

Matrices 1

- a vector stretches only in one dimension
  - think of the x axis of a plot
- what if you love linear algebra?

```
mat1 <- matrix(c(vec1, vec2), ncol = 2)
mat1
```

```
     [,1] [,2]
[1,]    5    1
[2,]    2    2
[3,]    7    3
[4,]    3    4
[5,]    2    5
[6,]    7    6
```

Matrices 2

```
1  mat2 <- matrix(c(vec1, vec2), ncol = 2, byrow = TRUE)
2  mat2
```

```
     [,1] [,2]
[1,]    5    2
[2,]    7    3
[3,]    2    7
[4,]    1    2
[5,]    3    4
[6,]    5    6
```

Matrices 3

- as the matrix is two-dimensional, we need to supply two values for indexing (x and y direction)

$$x_i y_i$$

```
1  mat2[3,2]
```

```
[1] 7
```

Dataframes 1

- the data we acquire during experiments often also consists of characters, for instance sample IDs.
- Matrices can only hold one data type
- if we provide more than one data type, R will use the most “low-level” type

```
1 matrix(c(1:3, "a", "b", "c"), ncol = 2)
```

```
     [,1] [,2]
[1,] "1"  "a"
[2,] "2"  "b"
[3,] "3"  "c"
```

Dataframes 2

- a possibility to work this out are data frames.

```r
df <- data.frame(numbers = 1:3,
                 letters = c("a", "b", "c"))
```

- **Note:**
  - we can now assign column names
  - columns can have different data types
  - **we cannot fill data frames row-wise!**

Dataframes 3

- indexing dataframes can be done as described for matrices
- on top of that, dataframes can be indexed by column name

```
          1  df[,1]
```

```
[1] 1 2 3
```

```
          1  df$numbers
```

```
[1] 1 2 3
```

```
          1  df$numbers[3]
```

```
[1] 3
```

Lists

- lists provide the next level of data structuring as they can hold all data types mentioned so far

```
1  l <- list(vec1, mat1, df)
```

Packages

Packages 1

- Functions in R come in packages
- packages are hosted in a central repository with strict rules
  - one of the major advantages in R!
- there are also other repositories
  - GitHub
    - check out the `aquaculturR` package!
  - Bioconductor (mostly bioinformatics content)
  - …

Packages 2

- packages can be installed using the `install.packages()` function

```
1 install.packages("tidyverse")
```

- eventually, they have to be loaded from the library using the `library()` function.

```
1 library(tidyverse)
```

Question 7

How to install the `devtools` package in R?

a. `install.packages("devtools")`

b. `library(devtools)`

Question 8

How to load the `devtools` package?

a. `install.packages("devtools")`

b. `library(devtools)`