Statistical Computing with R

Lecture 7: R Studio projects; `while` and `repeat` loops; the `apply` family; block 1 questions and wrap-up

Mirko Signorelli
: mirkosignorelli.github.io
: statcompr [at] gmail.com

Mathematical Institute
Leiden University

Master in Statistics and Data Science (2023-2024)

Announcements

- Reminder: no lecture next week!
- Assignment 2 will be published after Lecture 8 (tentative: Nov. 3)
- Assignment 3 will be published after Lecture 11 (tentative: Nov. 24)

Recap

Lecture 6:

- getting used to `R` Markdown
- lists
- data visualization: part 2
  1. scatter plots
  2. low level graphics functions
  3. functions and curves
  4. exporting images

Today:

- R Studio projects
- `while` and `repeat` loops
- the `apply` family
- block 1: questions and wrap-up

Context

- In lectures 4 and 5 we discussed how to organize your files and projects. A few take-home messages:
  1. create a folder / directory for each new project you work on
  2. use subfolders to separate different types of files (R scripts, data files, figures, results. . .)
  3. use meaningful names for your files
- RStudio has a built-in tool that you can use to retrieve existing / organize new projects directly from RStudio, called RStudio projects

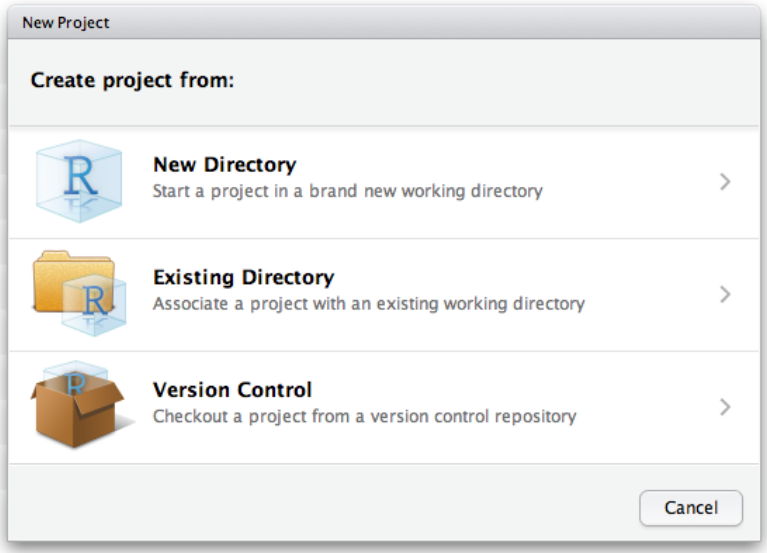
RStudio projects

- RStudio projects are a functionality that allows to organize your projects and files directly from RStudio
- Basic principle:

$$1 \text{ R Studio project} = 1 \text{ folder on your laptop}$$

Creating an R Studio project

- Go to: File → New Project. . . , you get:



- You can either associate the RStudio project with an existing folder, or create a new folder for it

Example

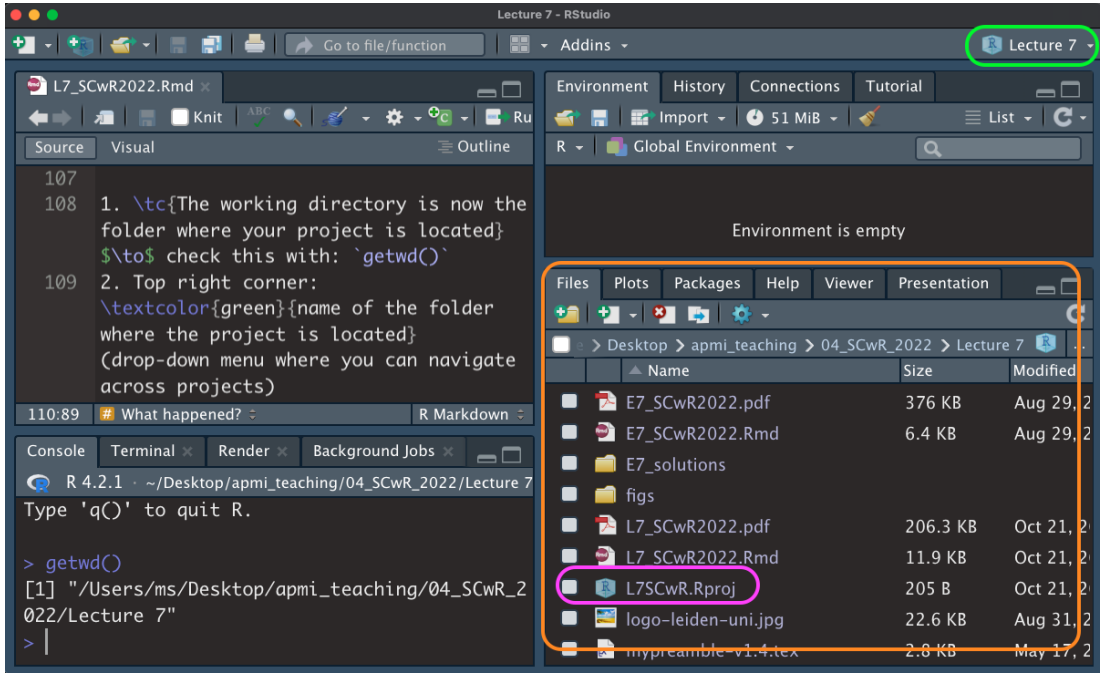
- I create an RStudio project that I associate with the (already existing) folder where my files for L7 are
- The folder is called "Lecture7". By default, the RStudio project will be called `Lecture7.Rproj`
- But: this is not a good name! I teach multiple courses, so lecture 7 from which course?!?
- To change the name of the RStudio project, I simply rename the '.Rproj' file. New file name: `L7_SCwR.Rproj`

And now?

1. Close RStudio
2. Double-click the `.Rproj` file: what happens?[1]

[1] Alternatively, you can open the project from: File → Open Project, or using the drop-down menu highlighted in green in the next slide

What happened?



1. The working directory is now the folder where your project is located
   $\to$ check this with: `getwd()`
2. Top right corner: name of the folder where the project is located + drop-down menu to navigate across projects
3. The Files panel shows the content of the folder where the project is located (you can open files from there)
4. The `.Rproj` file resides in the project folder

More about R Studio projects

- Main advantages:
  1. not having to worry too much about the working directory when coding in scripts or in the console
  2. navigate through different projects directly from RStudio
- Disadvantages:
  1. it might make it trickier to load / write to files located in other folders
  2. double-layer: file management system (Mac: Finder / Windows: Explorer) + RStudio project. You might prefer to do everything from Finder / Explorer
- More info: article about R Studio projects
- FAQ: do I **have to** use R Studio projects?
  - No, you don't have to. You can totally continue to manage your files and folders manually, if that works for you! ☺
  - But you can, if you like using them ☺

Loops in R

Types of loops in R:

- `for ( )` loops → L5
- `while ( )` loops → today!
- `repeat` loops → today!

Beyond `for` loops: motivation

- `for` repeats a chunk of code once for each `element` in the given `vector`:

```
for (element in vector) {
  # do this
  # this
  # and this
}
```

- `for` loops execute a task for a fixed number of times
- Problem: what if I don't know beforehand how many times I need to repeat the task, but only know that I should continue as long as a given condition is satisfied?

`while` loops

- `while` continues to run a chunk of code while a given logical `condition` remains `TRUE`

```
while (condition) {
  # do this
  # this
  # and this
}
```

Example

- How many numbers (1, 2, 3, ...) do we need to sum to reach 1000?

```
i = 0; cumul.sum = 0
while (cumul.sum <= 1000) {
  i = i + 1
  cumul.sum = cumul.sum + i
}
i # numbers added: 45
```

```
## [1] 45
```

```
cumul.sum # sum(1:45) = 1035
```

```
## [1] 1035
```

- Note how the `while` loop doesn't have an iterator variable $\Rightarrow$ you need to create it by yourself (if necessary)

repeat loops

- repeat repeats a chunk of code until you tell it to stop with a break command

```
repeat {
  # do this
  # this
  # and this
  if (condition) break
}
```

The same example, but with `repeat`

- How many numbers do we need to sum to reach 1000?

```
i = 0; cumul.sum = 0
repeat {
  i = i + 1
  cumul.sum = cumul.sum + i
  if (cumul.sum >= 1000) break
}
i # numbers added: 45
```

```
## [1] 45
```

```
cumul.sum # sum(1:45) = 1035
```

```
## [1] 1035
```

`while` versus `repeat`

- FAQ: aren't `while` and `repeat` basically the same?
- `while` and `repeat` are quite similar, but:
  1. `while`: the condition is given at the beginning $\rightarrow$ `while (condition)`
  2. `repeat`: you need to use one or multiple `if` statements to include the "breaking" conditions inside the body of the loop
  3. `repeat` allows for multiple stopping conditions, `while` doesn't!

Your turn

Fibonacci revisited

The Fibonacci sequence is a sequence of numbers with initial values 0 and 1, where each number is equal to the sum of the previous two. Formally:

$$x_k = x_{k-1} + x_{k-2} \text{ for } k > 2,$$

where $x_1 = 0$ and $x_2 = 1$.

1. How many elements from this sequence do we need to sum to reach (or go beyond) 4522?

Solutions

```
# Ex 1
fibo.seq = c(0, 1)
cumul.sum = sum(fibo.seq)
i = 3
while (cumul.sum < 4522) {
  fibo.seq[i] = fibo.seq[i-1] + fibo.seq[i-2]
  cumul.sum = cumul.sum + fibo.seq[i]
  i = i + 1
}
fibo.seq
```

```
##  [1]    0    1    1    2    3    5    8   13   21   34
## [11]   55   89  144  233  377  610  987 1597 2584
```

```
length(fibo.seq)
```

```
## [1] 19
```

```
cumul.sum
```

```
## [1] 6764
```

```
sum(fibo.seq)
```

```
## [1] 6764
```

The `apply` family

- `for` loops (see Lecture 4) can sometimes be redundant, unefficient or slow
- Several alternatives available, e.g.:
  - the `apply` family → today!
  - `replicate` → block 2
  - `R` package `foreach` → block 2
- Members of the `apply` family: `apply`, `sapply`, `lapply`, `vapply`...

Example: `for` vs `apply`

- How can we compute the median of variables (columns) in a matrix (/ array / data frame)?

```
set.seed(19920207)
m = matrix(rnorm(400), 100, 4)
```

- `for` loop solution:

```
medians = rep(NA, ncol(m))
for (i in 1:ncol(m)) medians[i] = median(m[ , i])
medians
```

```
## [1]  0.09258475 -0.01576570 -0.03694387 -0.08688649
```

- `apply` solution = much less work ☺:

```
apply(m, 2, median)
```

```
## [1]  0.09258475 -0.01576570 -0.03694387 -0.08688649
```

apply explained

Syntax:

```
apply(x, MARGIN, FUN, simplify = TRUE, ...)
```

- x: matrix / array / data frame
- MARGIN: 1 for rows, 2 for columns. MARGIN value > 2 or MARGIN = c(1,3) possible if x is an array
- FUN: the function you want to apply
- simplify: if FALSE, a list is returned. If TRUE (default), a simpler output type may be returned

Custom functions within apply

- FUN can be specified in many different ways:

```
apply(m, 2, mean)
```

```
## [1]  0.15442842 -0.01695023 -0.04047641 -0.15263341
```

```
apply(m, 2, function(x) mean(x, na.rm = T))
```

```
## [1]  0.15442842 -0.01695023 -0.04047641 -0.15263341
```

```
apply(m, 2, \(x) sum(x) / length(x))
```

```
## [1]  0.15442842 -0.01695023 -0.04047641 -0.15263341
```

```
my.mean = function(x) sum(x) / length(x)
apply(m, 2, my.mean)
```

```
## [1]  0.15442842 -0.01695023 -0.04047641 -0.15263341
```

The simplify argument

```
apply(m, 2, median) # default behaviour
```

```
## [1]  0.09258475 -0.01576570 -0.03694387 -0.08688649
```

```
apply(m, 2, median, simplify = F)
```

```
## [[1]]
## [1] 0.09258475
##
## [[2]]
## [1] -0.0157657
##
## [[3]]
## [1] -0.03694387
##
## [[4]]
## [1] -0.08688649
```

The apply family in a nutshell

Function	Possible inputs
`apply(x, MARGIN, FUN)`	matrix / array / data frame
`sapply(X, FUN)`	list / data frame
`lapply(X, FUN)`	list / data frame
`vapply(X, FUN, FUN.VALUE)`	list / data frame

lapply and sapply

Syntax:

```
sapply(x, FUN, simplify = T, ...)
lapply(x, FUN, ...)
```

- x: list / data frame
- FUN: the function you want to apply
- simplify: same as with apply

⚠ no MARGIN argument needed here ⚠

- What's the difference? Basically:

```
lapply(x, FUN, ...)  = sapply(x, FUN, simplify = F, ...)
```

- TIP: you may forget about lapply, and always use sapply

sapply and lapply (cont'd)

```
list6 = list(1:5, 6:10, 11:20)
sapply(list6, sum)
```

```
## [1]  15  40 155
```

```
lapply(list6, sum)
```

```
## [[1]]
## [1] 15
##
## [[2]]
## [1] 40
##
## [[3]]
## [1] 155
```

vapply

- `vapply` is similar to `sapply`, but it requires you to specify what type of output you want through `FUN.VALUE`
- For example, if you want a numeric vector of length 1 as output you need to specify:

```
vapply(list6, mean, numeric(1))
```

```
## [1]  3.0  8.0 15.5
```

- If one or more outputs are not of `FUN.VALUE` type, `vapply` will return an error! Try, for example:

```
vapply(list6, mean, numeric(2))
```

Your turn

Exercises

Let $X_1 \sim N(\mu = 3, \sigma = 1)$, $X_2 \sim Binom(n = 3, p = 0.4)$ and $X_3 \sim Exp(\lambda = 3)$ be 3 independent random variables. Create a data frame containing 1000 realizations from each random variable using the following code:

```
set.seed(19920207)
df = data.frame(
  x1 = rnorm(1000, mean = 3, sd = 1),
  x2 = rbinom(1000, size = 3, prob = 0.4),
  x3 = rexp(1000, rate = 3)
)
```

Use one of the variants of `apply( )` to:

1. estimate $E(X_1)$, $E(X_2)$ and $E(X_3)$
2. estimate $Var(X_1 + X_2 + X_3)$

Solutions

```r
# Ex 1
apply(df, 2, mean)
```

```
##       x1       x2       x3
## 2.980953 1.168000 0.336793
```

```r
# or, alternatively:
sapply(df, mean)
```

```
##       x1       x2       x3
## 2.980953 1.168000 0.336793
```

```r
# Ex 2
df$sum = apply(df, 1, sum)
var(df$sum)
```

```
## [1] 1.875527
```

R Studio projects

`while ( )` and `repeat` loops

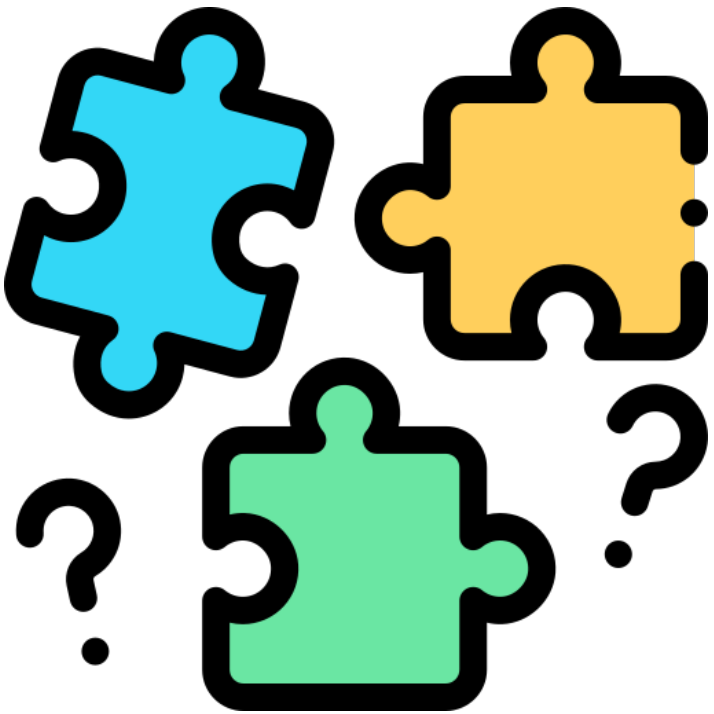
The `apply` family

Question time

Block 1 wrap-up

Question time

- This is all in terms of topics for this first block
- Time for questions on any of the topics covered so far!



R Studio projects

`while ( )` and `repeat` loops

The `apply` family

Question time

Block 1 wrap-up

Overview

In the first half of this course, we have learnt about:

1. different object types
2. how to write / use functions
3. conditional statements
4. looping
5. using tools in R and RStudio: scripts, `R` Markdown, RStudio projects...
6. managing your projects and files
7. data visualization

Objects

- Vectors → L1
- Matrices (and arrays) → L2
- Data frames → L2
- Lists → L6

Functions

1. "Built-in" functions: directly available when you open `R` → L2
2. Functions from `R` packages (`CRAN`, Bioconductor, ...): you first need to install the package + load it with `library( )` → L3
3. Custom functions → L3: you can create them with

```
f1 = function(x1, x2, ...) {
  #...
}
f2 = \(x1, x2, ...) {
  #...
}
```

- NB: (1) and (2) always have documentation available in their dedicated help pages → L2

Truth, or lie?

- Use logical operators to check if a condition is true or false → L4
- Execute different operations depending on whether certain conditions are satisfied → L4

```
if (cond1) {
  # execute code when cond1 is TRUE
} else if (cond2) {
  # execute code when cond1 is FALSE, but cond2 is TRUE
} else if (cond3) {
  # execute code when cond1 and cond2 are FALSE,
  # but cond3 is TRUE
} else {
  # execute code when cond1, cond2 and cond3 are FALSE
}
```

Repetitive operations

- `for` loops → L5
- `while` loops → L7
- `repeat` loops → L7
- `apply`, `sapply`, `lapply`, `vapply` → L7

More to come:

- `replicate`
- `R` package `foreach`

R scripts, R Markdown and R projects

- `R` scripts → L2: `R` code + comments
- `R` Markdown → L4: `R` code, document header (YAML), text, sections and subsections, images, tables, formulas (LaTeX), ...
- `R` projects → L7: a way to make it faster to reopen an existing project (= folder)

Organizing your files

- Use folders and subfolders to organize your files! → L4
- If evaluating code in the console (= R scripts): set / change working directory with `setwd( )`; check where you are with `getwd( )` → L3
- Dedicated functions to import, save and export data → L3

Data visualization

- Frequency distributions → L5
  1. categorical variables: bar plot, pie chart, waffle chart
  2. quantitative variables: histogram, empirical pmf plot, density plot
- Drawing functions and curves → L6
- Relationship between 2 or more variables
  1. scatter plots → L6
  2. density plots, boxplots, violin plots → block 2
  3. correlograms → block 2
- Longitudinal data: spaghetti plot → block 2
- `R` package `ggplot2` → block 2

More to come!

- In the first block of this course we have laid the foundations that are needed to find your way around R
- During block 2 we will introduce more advanced / specialized topics