

Statistical Computing with R

Lecture 4: logical operators; if () statements; organizing your files;
R Markdown; scripts vs R Markdown

Mirko Signorelli

🏠: mirkosignorelli.github.io

✉: statcompr [at] gmail.com

Mathematical Institute
Leiden University

Master in Statistics and Data Science (2023-2024)



Universiteit
Leiden

Announcements

- ▶ Assignment 1 (tentative dates)
 1. available on October 5
 2. it will cover the course material from lecture 1 up to lecture 6
 3. deadline to submit your solutions: October 15
 4. feedback by the end of October
- ▶ Reminder: assignments are an extra opportunity to practice your R skills and to receive feedback. They are **not graded and not compulsory**

Recap

Lecture 3:

- ▶ installing and loading R packages
- ▶ using functions from R packages
- ▶ writing your own functions
- ▶ the working directory
- ▶ importing, saving and exporting data

Today:

- ▶ logical operators
- ▶ if statements
- ▶ organizing your files
- ▶ R Markdown
- ▶ differences between R scripts and R Markdown (*self-study*)

Logical operators and if () statements

Logical operators

if () statements

Organizing your files

R Markdown

Differences between R scripts and R Markdown (SELF-STUDY)

Motivating example

- Example problem: how can we write an R function that computes

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x^2 & \text{if } 0 < x < 1 \text{ ?} \\ 1 & \text{if } x \geq 1 \end{cases}$$

- We need to learn how to
 1. check conditions about x ($x \leq 0$, $0 < x < 1$, $x \geq 1$) → logical statements in R
 2. perform different tasks (return $f(x) = 0 / x^2 / 1$) depending on which condition is true → `if ()` statements

Logical operators and `if ()` statements

Logical operators

`if ()` statements

Organizing your files

R Markdown

Differences between R scripts and R Markdown (SELF-STUDY)

Logical values

- ▶ When coding, one often needs to check whether a certain condition is satisfied, or if an expression is **true or false**, before moving on to the next step
- ▶ In R, TRUE and FALSE are **logical** values

```
v2 = c(18, 23, 14, 42)
# which elements are > 25?
v2 > 25
```

```
## [1] FALSE FALSE FALSE TRUE
```

```
# which elements are even?
v2 %% 2 == 0
```

```
## [1] TRUE FALSE TRUE TRUE
```

Logical operators

- ▶ Main logical operators: | and &
- ▶ | = logical OR
- ▶ & = logical AND

x	y	x y	x & y
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	TRUE	FALSE
FALSE	TRUE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE

Examples

```
v = 1:8  
v == 5 | v < 2
```

```
## [1] TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE
```

```
v < 3 & v > 9
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
v < 3 | v > 9
```

```
## [1] TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
# use which( ) to retrieve the position of the TRUES:  
which(v == 5 | v < 2)
```

```
## [1] 1 5
```

Logical operators and `if ()` statements

Logical operators

`if ()` statements

Organizing your files

R Markdown

Differences between R scripts and R Markdown (SELF-STUDY)

if () statements

- ▶ if () statements can be used to perform computations only when a condition is satisfied

```
if (condition) {  
    # execute code when condition is TRUE  
}
```

- ▶ condition is a logical statement that should return either TRUE or FALSE

Example

```
x = -3
if (x < 0) {
  print('x is negative')
}
```

```
## [1] "x is negative"
```

```
y = c(5, -3, 2, -1)
if (sum(y) < 0) {
  print('vector total is negative')
}
```

else () statements

- ▶ if () statements can be followed by an else () statement:

```
if (condition) {  
    # execute code when condition is TRUE  
} else {  
    # execute code when condition is FALSE  
}
```

else if () statements

- ▶ If you have more than one condition to verify, you can sequentially use: if (), else if () and else ()

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

- ▶ You can include as many else if () statements as you need

Example

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x^2 & \text{if } 0 < x < 1 \\ 1 & \text{if } x \geq 1 \end{cases}$$

How can we write an R function to compute $f(x)$?

Example (cont'd)

```
f = function(x) {  
  if (x <= 0) {  
    out = 0  
  } else if (x > 0 & x < 1) {  
    out = x^2  
  } else {  
    out = 1  
  }  
  return(out)  
}  
f(-3); f(0.5); f(4)
```

```
## [1] 0
```

```
## [1] 0.25
```

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


Example (cont'd)

- ▶ Due to the simplicity of $f(x)$, the previous function may be shortened as follows:

```
f = \(x) {  
  if (x <= 0) 0  
  else if (x > 0 & x < 1) x^2  
  else 1  
}  
f(-3); f(0.5); f(4)
```

```
## [1] 0
```

```
## [1] 0.25
```

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

Your turn

Exercises

PiggyBank is launching a new fixed term deposit whose duration is 3 years. The yearly interest rate is 1% for deposits up to 3000 €, 1.3% for deposits up to 10000 €, and 1.5% for deposits above 10000 €.

1. Write a function that returns the total (compound) interest for different values of the initial deposit
2. Compute the total interest for a deposit of 6000 € and for one of 13000 €

Solutions

```
# Ex 1
```

```
total.interest = function(depo) {  
  if (depo <= 3000) {  
    interest = 0.01  
  } else if (depo <= 10000) {  
    interest = 0.013  
  } else {  
    interest = 0.015  
  }  
  depo*(1+interest)^3 - depo  
}
```

```
# Ex 2
```

```
total.interest(6000); total.interest(13000)
```

```
## [1] 237.0552
```

```
## [1] 593.8189
```

Logical operators and `if ()` statements

Logical operators

`if ()` statements

Organizing your files

R Markdown

Differences between R scripts and R Markdown (SELF-STUDY)

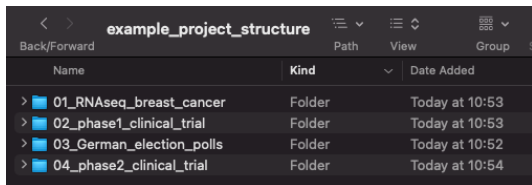
Motivation

- ▶ In Lecture 2 we discussed the importance of using clear names for your scripts and files, and of writing clear comments
- ▶ Projects can often be(come) large, involving tens or hundreds of files
⇒ it's important to structure your files into folders and subfolders so that you can more easily find what you need!

Folders and subfolders

A neat way to organize your files:

- ▶ create a folder / directory for each new project you work on
- ▶ create subfolders within the project folders to separate different types of files (R scripts, data files, figures, results. . .)



Name	Kind	Date Added
> 01_RNAseq_breast_cancer	Folder	Today at 10:53
> 02_phase1_clinical_trial	Folder	Today at 10:53
> 03_German_election_polls	Folder	Today at 10:52
> 04_phase2_clinical_trial	Folder	Today at 10:54

Figure 1: example folder containing 4 data analysis projects

Folders and subfolders (cont'd)

Within each project folder, you may:

1. save your R scripts and R Markdown files directly in the project folder
2. create subfolders where different input and output files are saved (see next slide!)

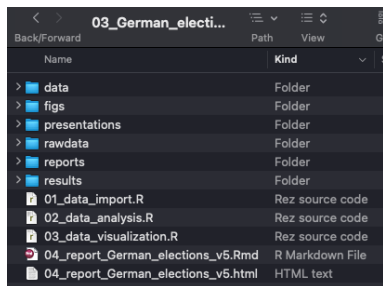


Figure 2: example of the use of subfolders to organize files within a project

Folders and subfolders (cont'd)

In the example shown in the previous slide:

1. `rawdata` contains the **original data** (`.csv`, `.xlsx`, ...) that you received / downloaded. Good practice: don't edit / overwrite such files¹
2. `data` contains the **data that you imported in R**, polished and edited (`.RData` files)
3. `figs` contains the **images** you created (`.png`, `.jpg`, `.pdf`, ...)
4. `results` contains files where you saved / exported the **results** of your analyses (e.g., `.RData` files, spreadsheets you made to navigate the results / send them to a collaborator, ...)
5. `reports` contains all the subsequent versions of your project reports
6. `presentations` contains presentations made for this project (`.Rmd`, `.pdf`, `.html`, `.pptx`, ...)

¹or make at least sure to keep a copy of the original, unmodified data that you received / created

Backups

This may be obvious, but:

**BACK UP YOUR DATA
REGULARLY**

Better safe than sorry! 😊



Logical operators and `if ()` statements

Logical operators


`if ()` statements

Organizing your files

R Markdown

Differences between R scripts and R Markdown (SELF-STUDY)

R Markdown

- ▶ R Markdown is a file format that makes it easy to create documents where you can combine R code and outputs with text, images, formulas, tables. . .
 1. File extension: `.Rmd`
 2. R Markdown uses  Markdown's syntax for section/subsection headers, tables, including images / URLs. . .
 3. You can also include \LaTeX formulas and environments!
 4. Output file can be: pdf file, Beamer (\LaTeX) presentation, html page, MS Word document, . . .
- ▶ In short: a **very flexible document editor** that allows you to keep (almost) everything in one place 😊

Short video:  [What is R Markdown?](#)




What can you do with R Markdown?

Some examples of documents you can create with R Markdown:



1. Solutions to an assignment
2. Report describing data analysis and results
3. Presentations
4. Scientific articles
5. Websites
6. Tutorials
7. Books
8. Dashboards

Resources

Major resources about R Markdown:

1.  <https://rmarkdown.rstudio.com>
2.  R Markdown cookbook
3.  R Markdown Cheat Sheet

Start here:

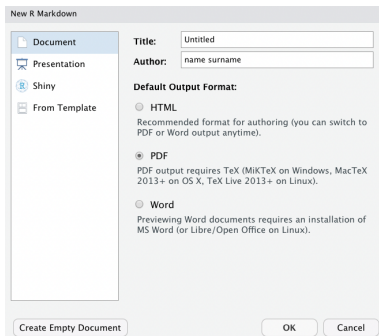
1. See the  [Gallery](#) for examples and templates of documents you can create with R Markdown
2. Read the  [Get started](#) guide

Creating an R Markdown file

1. Install the R Markdown package (and required dependencies if necessary):

```
install.packages("rmarkdown")
```

2. From the RStudio menu: File → New file → R Markdown
3. Select Document, insert title and author, and choose PDF as output:



YAML header

The **YAML header** contains basic information (“metadata”) about your document

- ▶ Default fields:

```
---  
title: "A sensible, catchy title"  
author: "John Doe"  
date: "19/5/2031"  
output: pdf_document  
---
```

- ▶ Personalization possible, some details can be found [here](#)

YAML header (cont'd)

- ▶ Example of header with personalized fields:

```
---  
title: "A sensible, catchy title"  
subtitle: "Followed by a more detailed subtitle"  
author: "John Doe"  
output:  
  html_document:  
    toc: true  
fontsize: 10pt  
---
```


Text formatting: the basics

- ▶ #, ## and ### define section, subsection and subsubsection headers
- ▶ *, *, * creates a bulleted list, 1., 2., 3. a numbered list
- ▶ Type: `*text*` / `_text_` for *italics*
- ▶ Type: `**text**` / `__text__` for **bold**
- ▶ URLs can be included with `[linked phrase](http://example.com)`
- ▶ Include images with: `![alt text](figures/img.png)` (or L^AT_EX's `\includegraphics!`)
- ▶ Formulas can be included using the usual L^AT_EX syntax ($\$, \$\$, \begin{equation} \dots$)

Example: section headers, lists, ...

► Input:

```
13 # Section header
14
15 Here are some bolded words and something in italics
16
17 ## Subsection header
18
19 Here's a bulleted list:
20
21 * element 1
22 * element 2
23 * element 3
24
25 ### Subsubsection header
26
27 And here's a numbered list:
28
29 1. element 1
30 2. element 2
31 3. element 3
```

► Output:

Section header

Here are some **bolded words** and *something in italics*

Subsection header

Here's a bulleted list:

- element 1
- element 2
- element 3

Subsubsection header

And here's a numbered list:

1. element 1
2. element 2
3. element 3

Example: formulas

► Input:

```
32 * # Formulas
33
34 Formulas can be created using the standard \LaTeX syntax:
35
36 $$ f_X(x) = \frac{\lambda^x}{x!} e^{-\lambda}, \lambda: x = 0, 1, 2, \dots $$
```

► Output:

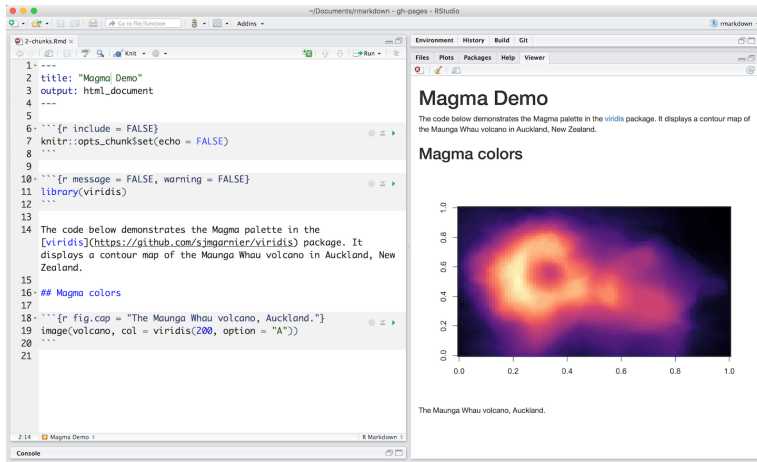
Formulas

Formulas can be created using the standard \LaTeX syntax:

$$f_X(x) = \frac{\lambda^x}{x!} e^{-\lambda}, x = 0, 1, 2, \dots$$

R code chunks

- ▶ R code is included in **chunks**
- ▶ Many options to determine what a chunk does (see next slide)



The screenshot displays the RStudio interface with a file named '2-chunks.Rmd'. The editor shows an R code chunk with the following content:

```
1 ----  
2 title: "Magma Demo"  
3 output: html_document  
4 ----  
5  
6 ```{r include = FALSE}  
7 knitr::opts_chunk$set(echo = FALSE)  
8 ```  
9  
10 ```{r message = FALSE, warning = FALSE}  
11 library(viridis)  
12 ```  
13  
14 The code below demonstrates the Magma palette in the  
15 [viridis](https://github.com/sjmgarnier/viridis) package. It  
16 displays a contour map of the Maunga Whau volcano in Auckland, New  
17 Zealand.  
18  
19 ## Magma colors  
20  
21 ```{r fig.cap = "The Maunga Whau volcano, Auckland."}  
22 image(volcano, col = viridis(200, option = "A"))  
23 ```
```

The right pane shows the rendered HTML output. It features a title 'Magma Demo', a paragraph explaining the code, a subheading 'Magma colors', a contour map of the Maunga Whau volcano using the viridis color palette, and a caption 'The Maunga Whau volcano, Auckland.'.

R code chunks (cont'd)

- ▶ You can alter the default values of chunk options to change the behaviour of code within a chunk:

option	default	effect
eval	TRUE	Whether to evaluate the code and include its results
echo	TRUE	Whether to display code along with its results
warning	TRUE	Whether to display warnings
error	FALSE	Whether to display errors
message	TRUE	Whether to display messages
tidy	FALSE	Whether to reformat code in a tidy way when displaying it
results	"markup"	"markup", "asis", "hold", or "hide"
cache	FALSE	Whether to cache results for future renders
comment	"##"	Comment character to preface results with
fig.width	7	Width in inches for plots created in chunk
fig.height	7	Height in inches for plots created in chunk

Example: R code chunks

► Input:

```
35 # Matrices
36
37 Matrices can be created using the function `matrix()` :
38
39 ```{r createA, eval = T, echo = F}
40 # echo = F: the code and comments in this chunk will not be displayed in the
  compiled pdf
41 A = matrix(c(3, 5, -6, -2, 1, 4),
42           nrow = 2, ncol = 3, byrow = T)
43 A
44 ```
45 |
46 To compute the transpose of a matrix, you can use `t()` :
47
48 ```{r transpose, eval = T, echo = T}
49 # echo = T: the code and comments in this chunk will be displayed in the
  compiled pdf
50 t(A)
51 t(A)*2/3
52 ```
```

Example: R code chunks

► Output:

Matrices

Matrices can be created using the function `matrix()`:

```
##      [,1] [,2] [,3]
## [1,]   3   5  -6
## [2,]  -2   1   4
```

To compute the transpose of a matrix, you can use `t()`:

```
# echo = T: the code and comments in this chunk will be displayed in the compiled pdf
t(A)
```

```
##      [,1] [,2]
## [1,]   3  -2
## [2,]   5   1
## [3,]  -6   4
```

```
t(A)*2/3
```

```
##      [,1]      [,2]
## [1,] 2.000000 -1.333333
## [2,] 3.333333  0.666667
## [3,] -4.000000  2.666667
```

► Which differences does `echo = T / F` entail?


Creating tables

- ▶ Markdown syntax:

```
| Function | `R` package | File type |
|-----|-----|-----|
| `write.table` | base `R` | General (.txt, .csv, .dat, ...) |
| `write.csv` | base `R` | .csv (Comma Separated Value) |
| `write.xlsx` | xlsx | .xls, .xlsx (Excel files) |
| `write_ods` | readODS | .ods (Open Document Spreadsheets) |
```

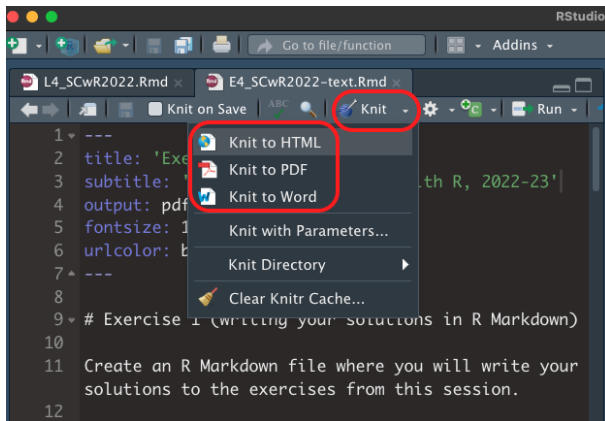
- ▶ Output:

Function	R package	File type
write.table	base R	General (.txt, .csv, .dat, ...)
write.csv	base R	.csv (Comma Separated Value)
write.xlsx	xlsx	.xls, .xlsx (Excel files)
write_ods	readODS	.ods (Open Document Spreadsheets)

- ▶ Alternatively, you may create tables using  \LaTeX 's syntax

Compiling (“rendering”) the document

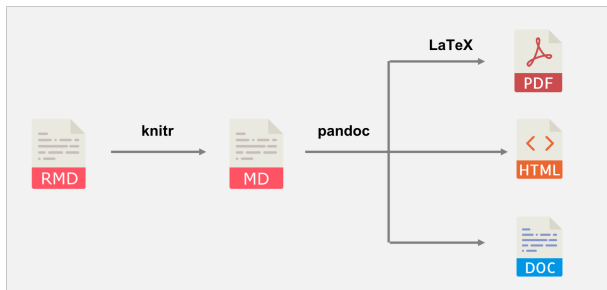
- ▶ Finally, press the **Knit button** to compile the document:



- ▶ Pro tip: **compile your document regularly** to find out if things are running smoothly or if you have made a mistakes somewhere

Compiling (“rendering”) the document (cont’d)

- ▶ “Knitting” is a process that takes as input your .Rmd file, and generates a pdf / html / doc file as output:



Your turn

Create an R Markdown document (10 minutes)

Create an R Markdown document where you present your solutions to exercises 2, 3 and 4 from Lecture 3. Use this as an opportunity to familiarize with section and subsection headers, text formatting, and chunk options in R Markdown!

Logical operators and `if ()` statements

Logical operators

`if ()` statements

Organizing your files

R Markdown

Differences between R scripts and R Markdown (SELF-STUDY)

Self-study section

- ▶ Depending on how late it is when we reach this section, I may skip this section during the class or just skim through it
- ▶ If I do: please [read this section carefully after the lecture](#)

Scripts vs R Markdown: what should I use?

- ▶ **R scripts:** your main goal is **coding**
 1. Mostly useful in the **coding phase** (tip: include comments to clarify what the code is doing)
 2. Why? It makes it easy and quick to edit code (you write some code, debug it, evaluate it immediately in the console, check if it runs properly. . .)
 3. Specially useful when you are working on a complex analysis / algorithm implementation
- ▶ **R Markdown:** your main goal is **gathering / presenting the results** of your work
 1. Mostly useful in the **communication phase**: to send to others a single (or multiple) structured pdf / html file(s) where everything (code, text, figures, tables. . .) can be easily found
 2. Here, the focus lies on: **explaining** the (coding) steps that you took + **presenting** the output (results) of your work

R script or R Markdown?

What would you use for each of the following tasks?

Task	Script	R Markdown
Data import and cleaning		
Submit an assignment/exam		
Create a complex function (e.g., 50+ lines)		
Present the results of your analyses		
Write a presentation		
Create a website/blog		

Answers in the next slide 😊

Answers

Task	Script	R Markdown
Data import and cleaning	✓	
Submit an assignment/exam		✓
Create a complex function (e.g., 50+ lines)	✓	
Present the results of your analyses		✓
Write a presentation		✓
Create a website/blog		✓

► See also the next slide!

So what should I use?

- ▶ Sometimes, choice between R scripts and R Markdown is rather obvious
 1. Do a lot of data cleaning and manipulation → R scripts
 2. Write complex code that requires a lot of debugging → R scripts
 3. Write a report or a presentation → R Markdown
 4. create a website → R Markdown
- ▶ Sometimes, the choice is less clear-cut
 1. For certain tasks, you may prefer to use scripts, or R Markdown
 2. Example: perform a statistical analysis of a given dataset

So what should I use? (cont'd)


- ▶ Often, a **mixed strategy** is possible
- ▶ Example: a complex data analysis project / simulation study
 1. **Implementation (coding) phase**: split the many steps across several **R scripts** that make it easy to debug your code and to rerun (parts of) the analysis
 2. **Sharing phase**: write one or more **R Markdown reports** where you include (a selection of) the code from your script, alongside with text that explains all what is relevant and outputs (including figures and tables) of your analysis

A few important differences

- ▶ When is the code evaluated?
 1. R scripts: every time you choose to execute one or more lines of code
 2. R Markdown: when knitting, all the code in the document is evaluated sequentially
- ▶ How is the code evaluated?
 1. R scripts: code is sent to the [console](#), and it modifies objects/functions that are in the [global environment](#). You can easily interact and view objects and functions
 2. R Markdown: the document is evaluated all in one go. If all works without errors, the output document is generated. Otherwise you get an error message. You cannot interact with the intermediate outputs during the compiling phase

A few important differences (cont'd)

- ▶ Working directory

1. R scripts: you need to specify the working directory (see Lecture 3)
 2. R Markdown: by default, the working directory for R code chunks is the directory that contains the Rmd document
- ▶ If you need to change the working directory in R Markdown,  [check out this explanation](#)