Reproducible Research in R

Dries Debeer & Benjamin Becker

29. and 30. September 2022

FDZ Autumn Academy

Introduction

Introduction

Who are we?

Dries Debeer

Statistical Consultant at Ghent University (FPPW)

scDIFtest, permimp, eatATA, mstDIF

dries.debeer@ugent.be

Benjamin Becker

Researcher at IQB (Verbund Forschungsdaten)

eatGADS, eatDB, eatATA, pisaRT

b.becker@iqb.hu-berlin.de

Introduction

Who are you?

1. Occupation, employer?
2. Previous knowledge and experience
    - with reproducible research?
    - with R?
3. Specific interest/motivation for this workshop?

Agenda

Day 1

- Conceptual Introduction
- Writing Reproducible R Code
- Reproducible Reporting

Day 2

- RMarkdown
- Version Control/git

Reproducible Research

Reproducibilty & Replicability

Reproducibility

- Same research question
- Same analysis
- **Same** data

=> *Same results*

Replicability

- Same research question
- Same analysis
- **New** data

=> *Same results*

Reproducibilty

The **replicability** crisis in psychological* research is partly caused by **reproducibility** issues.

- more than 70 percent of the researchers that tried to reproduce other scientist's experiments failed (Baker, 2016).
- more than 50 percent of the researchers that have tried to reproduce their own experiments failed (Baker, 2016).

*Psychology, educational sciences and social work

Baker, M. (2016). Reproducibility crisis. *Nature, 533(26)*, 353-66.

Reproducibility

Threats to self-replication (see also Peikert et al., 2021)

- (Silent) Mistakes in data preparation/analysis (e.g., syntax bugs)
- Inconsistent versions of code, data, or both
- Copy-and-paste errors
- ...

Peikert, A.,van Lissa, J., Brandmaier, A.M. (2021). Reproducible Research in R: A Tutorial on How to Do the Same Thing More Than Once. *Psych 2021,* 3, 836–867.

Reproducibility

Threats to replication of others

- ...
- Data not available
- Incomplete documentation/reporting
- ...

Reproducibility

Proposed solutions (see also Peikert et al., 2021)

- Clean code
- Version control
- Dynamic document generation
- (Independent reproduction)

Reproducibility

Workshop Goals

Be able to ...

- ... reproduce your own research (i.e., analyses)
- ... publish your code openly with your research
- ... create research which is reproducible for others
- ... spread the word

Writing Reproducible R Scripts

Before we start

1. Unzip the workshop-folder
2. Save the folder on your machine
3. Browse to the unzipped folder
4. Open the folder
5. Open the **“example-project”** folder
6. Double click on **“example-project”** or **“example-project.Rproj”**

Reproducible Workflow in R & Rstudio

Key Principles:

1. Save your code
2. Work in projects
   - Use multiple scripts
   - Use multiple projects
3. Frequently and completely restart

1. Save your code

- Save the code for preprocessing/data manipulation (don't save intermediate data sets)
- Save the code for visualizations (don't save the plots)
- Save the code for analyses (don't save the results)

May the source code be with you!

1. Save your code

- Save the code for preprocessing/data manipulation (don't save intermediate data sets)
- Save the code for visualizations (don't save the plots)
- Save the code for analyses (don't save the results)

May the source code be with you!

- Use `saveRDS()` and `readRDS()` for objects that require a long time to compute

1. Save your code

Within RStudio

1. Choose `Tools < Global options`
2. Under `General`
   - DON'T `Restore .RData into workspace at startup`
   - NEVER `Save workspace to .Rdata on exit:`
   - Save your script instead!
   - Use `saveRDS()` and `readRDS()` for objects that require a long time to compute
3. You can further personalize RStudio
   - Visually (`Tools < Global options < appearance`)
   - Keyboard shortcuts (`Tools < Modify Keyboard Shortcuts...`)
   - ...

1. Save your code

A typical workflow:

- Write your code in scripts in text editor.
- Execute lines (or chunks) of code by sending them to the R console.
- Add comments using the hash tag #
- Add sections to long scripts (or use multiple scripts)
    - No lines of code are lost when R is shut down or crashes
    - Assures reproducible code/coding
    - Makes it easy to share your code with colleagues or reviewers
    - ...

1. Save your code

Explicitly state the packages that you use.

- Load packages on the top of your script.
- Use direct access via `<package-name>::<function-name>`.
- Save your session information `sessionInfo()`.

Packages...

- have versions (and change)
- use the renv-package to managa pakage dependencies.

Reproducible Workflow in R & Rstudio

Key Principles:

1. Save your code
2. Work in projects
    - Use multiple scripts
    - Use multiple projects
3. Frequently and completely restart

2. Work in projects

What is a project?

- One designated folder containing all files related to a single (research) project.
- When necessary, add sub folders for ...
    - data
    - R scripts
    - figures and graphs
    - manuscripts
    - presentations
    - ...
- The folder contains all relevant files, nothing more.

2. Work in projects

The idea of a project is formalized in an RStudio project.

- Technically it is a small text file with (.Rproj) extension,
- which is associated with RStudio.
- RStudio recognizes the “parent folder” of this file as the project folder

2. Work in projects

Within an RStudio project...

- the project folder is automatically set as the *working directory*.

  try `getwd()` within a project

- **Don't use**

```
setwd("path\to\your\local\folder")
```

2. Work in projects

Use relative paths for reading and writing data. Avoid ...

```
cannot open file 'path\that\only\works\on\my-computer': No such file or directory
```

- With relative paths, code works when ...
  - the project folder is moved (or renamed)
  - you are working on a shared drive
  - you send your project in a ZIP-folder
- useful functions:
  - `?list.dirs`
  - `?list.files`
- (Or use the here-package)

2. Work in projects

Keep it clean!

- One folder per project
- No outside code
- No outside computing
- Use separate RStudio projects for each project
- Use separate Rstudio instances for each Rstudio project

2. Work in projects

How to create an Rstudio project?

1. Open Rstudio
2. Choose `File < New Project ...`
3. Choose `Existing Directory`
4. Browse to the directory on your machine where you saved the course content and select the "example-project-2" folder as the `Project working directory`
5. Click `Open in new session`
6. Click `Create Project`

2. Work in projects

Within a project, use multiple source files (scripts) for:

1. Preprocessing data
2. Descriptive analysis
3. Building models
4. Hypothesis testing
5. Making graphs
6. ...

2. Work in projects

Naming files...

NO!

- “My Abstract.docx”
- “Ana’s file for Boris and Casey.csv”
- “new.data from 2-11-2018.txt”
- “fig 2.png”
- “Figure 2 NEW VERSION.png”

YES!

- “2021-11-01_abstract_psycon.docx”
- “data_longitudinal-sleep-study_version-anna.csv”
- “2020-11-02_data_depression-study.txt”
- “fig02_histogram-residuals.png”
- “fig02_histogram-residuals_V2.png”

2. Work in projects

Naming files...

NO!

- “analysis_november_new.R”
- “someCool_functions.R”
- “dontUseThis.R”
- “preprocesing_data_final.R”

YES!

- “01_preprocess-data.R”
- “02_fit-models.R”
- “03_make-figures.R”
- “05_export-table-1.R”
- “99_utility-functions.R”

Naming Files

Files names should be machine readable and human readable:

1. Machine readable
    - no spaces
    - no punctuation
    - no special characters
    - logical dates: yyyy-mm-dd
2. Human readable
    - easy to read
    - content should be clear
3. Helpful ordering
    - chronological
    - logical

Naming Files

Some ideas:

- use “_” for separating *meta-data*
- use “-” for separating words
- start with date for chronological ordering: yyyy-mm-dd
- start with numbers for logical ordering

2. Work in projects

Naming files...

- “2021-11-01_abstract_psycon.docx”
- “data_longitudinal-sleep-study_version-anna.csv”
- “2020-11-02_data_depression-study.txt”
- “fig02_histogram-residuals.png”
- “fig02_histogram-residuals_V2.png”

- “01_preprocess-data.R”
- “02_fit-models.R”
- “03_make-figures.R”
- “05_export-table-1.R”
- “99_utility-functions.R”

Reproducible Workflow in R & Rstudio

Key Principles:

1. Save your code
2. Work in projects
    - Use multiple scripts
    - Use multiple projects
3. Frequently and completely restart

3. Frequently and Completely Restart

Regularly completely restart your session.
Don't use

```r
rm(list = ls())
```

- start clean, every time
- restart to make sure everything reproduces
- `.rs.restartR()`
- `Ctrl + Shift + F10`

How to … read data

To analyse data, you have to *read* the data from some file (or connection) and make it an *object* in R.

Almost any type of file can be read by R, via specific functions and packages (`reader`, `haven`, `readxl`, …).

- .txt → `read.table()`
- .csv → `read.csv(), read.csv2()`
- .xls → `readxl::read_xls()`
- .xlsx → `readxl::read_xlsx()`
- .sav → `haven::read_sav()`
- .por → `haven::read_por()`
- .sas → `haven::read_sas()`
- …

How to … preproces data

Useful functions in **Base R**

- `?'['` and `?'[['`
- `?merge`, `?reshape`
- `?apply`, `?tapply`, `?aggregate`,…
- `?sort`, `?order`
- …

Useful functions in `dplyr`

- `?filter`, `?select`, `?slice`
- `?mutate`, `?rename`
- `?group_by`, `?summarize`, `?ungroup`
- `?arrange`
- `?inner_join`, `?left_join`, …

How to ... preproces data

For fast and efficient data wrangling with VERY big data, the `data.table`-package can be helpful.

How to ... preproces data

Never change raw data!

If you made some mistakes while preprocessing raw data → change your code and re-run it.

Note that:

- R “reads” the data and loads it in the work space.
- Hence, manipulating data within R(Studio) does not change the data on your machine. Only the loaded data within the work space is changed.

How to ... write data

When the data (a data.frame, a fitted model, ..) you want to save is for use in R only, use `saveRDS()` `readRDS()`.

When the data (a data.frame) is for use by software, several options are available:

- .txt → `write.table()`
- .csv → `write.csv()`, `write.csv2()`
- .sav → `haven::write_sav()`
- .por → `haven::write_por()`
- .sas → `haven::write_sas()`
- ...

WARNING:

R does not prompt a warning when you are about to overwrite an existing file.

→ Make sure you do not have writing permissions for important raw data. Create Backups.

How to … save plots

Only use the RStudio “Plot”-window for interactive plot making.

Don’t use it for saving plots.

For reproducible figures, use `pdf()`, `png()`, `jpeg()`, `tiff()`, … instead. See `?jpeg`.

Adjust size and aspect ratio using the arguments:

- `width = ...`
- `height = ...`
- `units = ...` → the units for the `width` and `height` arguments
- …

How to ... save plots

```r
z_values <- rnorm(1e+4)
sig_z_values <- z_values[pnorm(abs(z_values),
                               lower.tail = FALSE) < 0.025]
pdf("name_for_this_figure.pdf", width = 10, height = 7)
{
  # all the code that creates the figure
  hist_data <- hist(z_values,
                    main = "Significant Z-values \n(stupid plot)",
                    xlab = "Z-values")
  hist(sig_z_values,
       breaks = hist_data$breaks,
       add = TRUE,
       col = "skyblue")
}
dev.off()
```

Reproducible Reporting

Common Workflows

Not very reproducible...

- Perform data preparation via hand/graphical user interfaces (SPSS, Excel)
- Perform data analyses via hand/graphical user interfaces (SPSS, Excel)
- Write a research report/manuscript (Word)
- Insert your results via copy & paste in text or as figures/tables

Reproducible Workflows

Better...

- Perform data preparation via syntax (R, Stata, SPSS)
- Perform data analyses via via syntax (R, Stata, SPSS)
- Write a research report/manuscript via a typesetting system or markup language (Markdown/LaTeX)
- Insert your results via *dynamic document generation* (RMarkdown/knitr)

Markdown

Lightweight Markup Language

- Pandoc distributed via RStudio

LaTeX

Typesetting System

- Engines (pdflatex, xelatex, lualatex, ...)
- Distributions (OS dependent: Miktex, tinytex, ...)
- Text editors (Texlive, Texmaker, RStudio...)

LaTeX vs. Markdown

Advantages LaTeX

- APA7 document class
- Better support for formulas
- Looks nicer
- Better customization
- Designated output format: .pdf

markdown-or-latex

LaTeX vs. Markdown

Advantages Markdown

- Extremely easy to use
- No complicated setup
- Designated output format: .html

markdown-or-latex

Dynamic Document Generation

Intertwine reporting and data preparation/analyses

- LaTeX → knitr (.Rnw)
- Markdown → RMarkdown (.Rmd)

Execute R code during document generation.

LaTeX and knitr

knitr

- substitutes `sweave`
- allows embedding R chunks in .tex code (.Rnw)
- powerful tool for automated document generation

RMarkdown

See tomorrow!

Resources

LaTeX resources

- Online Tutorials (e.g. Tutorial)
- Overleaf

knitr Ressources

- knitr Homepage
- knitr Book

Semi-Dynamic Document Generation

Unfortunately life is not always easy...

- Limited resources to learn even more tools
- Collaborators insist on using Word
- Journals only accept Word documents for submissions
- ...

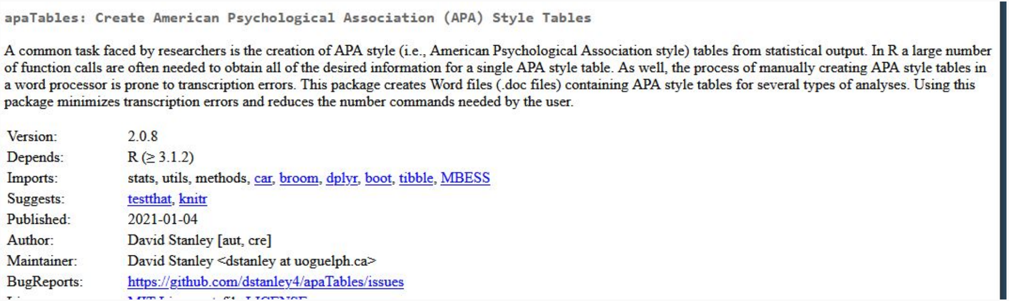
However, we can always try to make small improvements!

Semi-Dynamic Document Generation

For Example: **Automated APA tables in Word**
→ Useful R packages

- apaTables
- rempsyc
- flextable
- sjPlot
- stargazer
- ...

apaTables



apaTables: Create American Psychological Association (APA) Style Tables

A common task faced by researchers is the creation of APA style (i.e., American Psychological Association style) tables from statistical output. In R a large number of function calls are often needed to obtain all of the desired information for a single APA style table. As well, the process of manually creating APA style tables in a word processor is prone to transcription errors. This package creates Word files (.doc files) containing APA style tables for several types of analyses. Using this package minimizes transcription errors and reduces the number commands needed by the user.

Version:	2.0.8
Depends:	R (≥ 3.1.2)
Imports:	stats, utils, methods, car, broom, dplyr, boot, tibble, MBESS
Suggests:	testthat, knitr
Published:	2021-01-04
Author:	David Stanley [aut, cre]
Maintainer:	David Stanley <dstanley at uoguelph.ca>
BugReports:	https://github.com/dstanley4/apaTables/issues

apaTables

Descriptive statistics for a data set

```r
apaTables::apa.cor.table(iris,
                         filename = "iris_apatable_descr.doc",
                         landscape = FALSE)
```

apa Tables

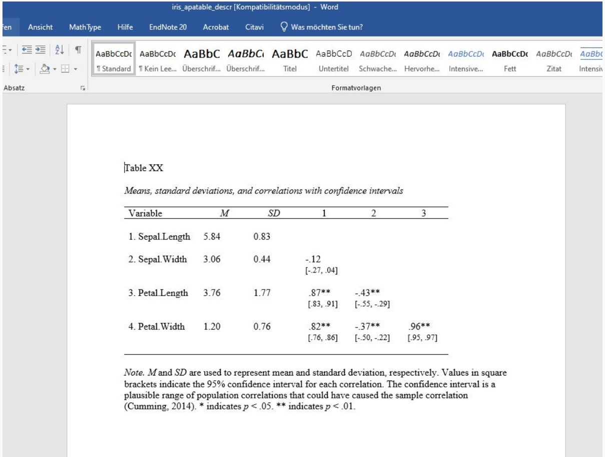


Table XX

Means, standard deviations, and correlations with confidence intervals

Variable	*M*	*SD*	1	2	3
1. Sepal.Length	5.84	0.83			
2. Sepal.Width	3.06	0.44	-.12 [-.27, .04]		
3. Petal.Length	3.76	1.77	.87** [.83, .91]	-.43** [-.55, -.29]	
4. Petal.Width	1.20	0.76	.82** [.76, .86]	-.37** [-.50, -.22]	.96** [.95, .97]

Note. *M* and *SD* are used to represent mean and standard deviation, respectively. Values in square brackets indicate the 95% confidence interval for each correlation. The confidence interval is a plausible range of population correlations that could have caused the sample correlation (Cumming, 2014). * indicates $p < .05$. ** indicates $p < .01$.

apaTables

Regression tables

```r
reg1 <- lm(mpg ~ cyl * gear, data = mtcars)
apaTables::apa.reg.table(reg1,
                         filename = "mtcars_apatable_reg.doc")
```

apa Tables

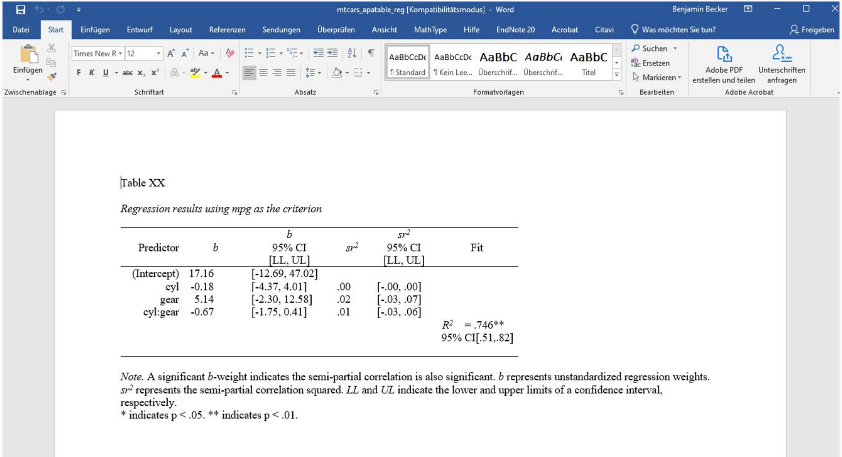


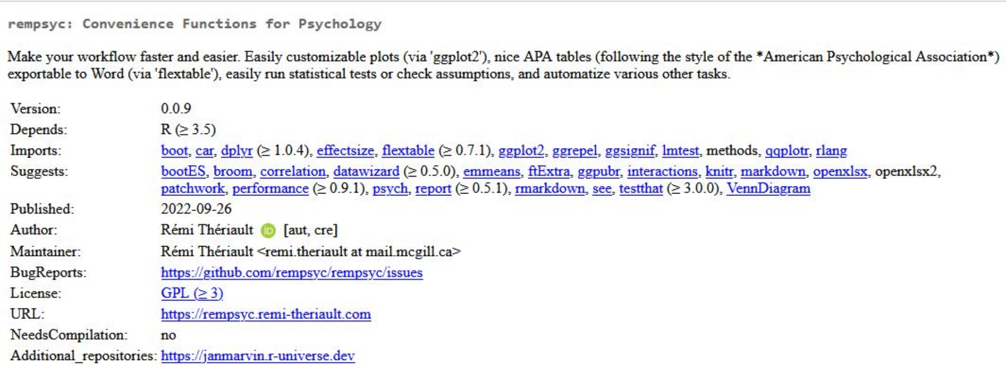
Table XX

Regression results using mpg as the criterion

Predictor	*b*	*b* 95% CI [LL, UL]	*sr*^{2}	*sr*^{2} 95% CI [LL, UL]	Fit
(Intercept)	17.16	[-12.69, 47.02]			
cyl	-0.18	[-4.37, 4.01]	.00	[-.00, .00]	
gear	5.14	[-2.30, 12.58]	.02	[-.03, .07]	
cyl:gear	-0.67	[-1.75, 0.41]	.01	[-.03, .06]	
					R^2 = .746** 95% CI[.51,.82]

Note. A significant *b*-weight indicates the semi-partial correlation is also significant. *b* represents unstandardized regression weights. *sr*^{2} represents the semi-partial correlation squared. *LL* and *UL* indicate the lower and upper limits of a confidence interval, respectively.
* indicates p < .05. ** indicates p < .01.

rempsyc



rempsyc: Convenience Functions for Psychology

Make your workflow faster and easier. Easily customizable plots (via 'ggplot2'), nice APA tables (following the style of the *American Psychological Association*) exportable to Word (via 'flextable'), easily run statistical tests or check assumptions, and automatize various other tasks.

Version:	0.0.9
Depends:	R (≥ 3.5)
Imports:	boot, car, dplyr (≥ 1.0.4), effectsize, flextable (≥ 0.7.1), ggplot2, ggrepel, ggsignif, lmtest, methods, qqplotr, rlang
Suggests:	bootES, broom, correlation, datawizard (≥ 0.5.0), emmeans, ftExtra, ggpubr, interactions, knitr, markdown, openxlsx, openxlsx2, patchwork, performance (≥ 0.9.1), psych, report (≥ 0.5.1), rmarkdown, see, testthat (≥ 3.0.0), VennDiagram
Published:	2022-09-26
Author:	Rémi Thériault [aut, cre]
Maintainer:	Rémi Thériault <remi.theriault at mail.mcgill.ca>
BugReports:	https://github.com/rempsyc/rempsyc/issues
License:	GPL (≥ 3)
URL:	https://rempsyc.remi-theriault.com
NeedsCompilation:	no
Additional_repositories:	https://janmarvin.r-universe.dev

rempsyc

Manual table containing example data

```
word_table <- rempsyc::nice_table(iris[1:5, ],
  title = c("Table 1", "Example Iris Data Set"),
  footnote = c("The table contains five rows of the iris data."))

# save as .docx
rempsyc::save_as_docx(word_table, path = "iris_rempsyc_data.docx")
```

rempsyc

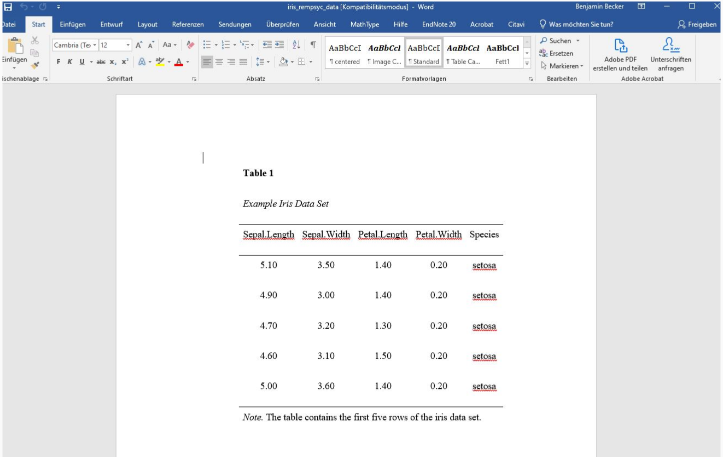


Table 1

Example Iris Data Set

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.10	3.50	1.40	0.20	setosa
4.90	3.00	1.40	0.20	setosa
4.70	3.20	1.30	0.20	setosa
4.60	3.10	1.50	0.20	setosa
5.00	3.60	1.40	0.20	setosa

Note. The table contains the first five rows of the iris data set.

Exercises

RMarkdown

Motivation

Scientific results should be reproducible! (And hopefully repeatable.) Open science implies transparency in methodology, manipulation and analysis of data.

RMarkdown is a tool that can facilitate:

- **Reproducible results** by avoiding copy-paste errors
- **Open science** by including the actual code that was used to analyse the data
- Revising your work, as changes in data-analytic choices are automatically incorporated in the final output. **It saves time!**

RMarkdown

Scientific results should be reproducible! (And hopefully repeatable.) Open science implies transparency in methodology, manipulation and analysis of data.

RMarkdown is a tool that can facilitate:

- **Reproducible results** by avoiding copy-paste errors
- **Open science** by including the actual code that was used to analyse the data
- Revising your work, as changes in data-analytic choices are automatically incorporated in the final output. **It saves time!**

RMarkdown

RMarkdown is a **document format** (.Rmd) that combines

- Markdown formatted text
- chunks of (R-)code

→ We know R. But what is *Markdown*?

Markdown

Markdown is a simple formatting language.

It is a **markup language**:

- like html, or LaTeX
- inline formatting "code" is added to text

It is **simple**:

- only limited and simple formatting code
- the code does not hinder reading raw file

→ easily transformed to other formatting languages

Markdown Headings

Markdown Code

```
# Heading level 1

## Heading level 2

### Heading level 3

###### Heading level 6
```

Rendered Text

Heading level 1

Heading level 2

Heading level 3

Heading level 6

Markdown Paragraphs

Markdown Code

```
A new line without a blank line
is not considered as a new paragraph.

Only after a complete empty line
a
new
paragraph
is created.
```

Rendered Text

A new line without a blank line is not considered as a new paragraph.

Only after a complete empty line a new paragraph is created.

Markdown Lists

Markdown Code

```
- chocolate
- fruit
  * apples
  * bananas
  * annanas
- vegetables
  + carrots
  + leak
```

Rendered Text

- chocolate
- fruit
  - apples
  - bananas
  - annanas
- vegetables
  - carrots
  - leak

Markdown Numbered Lists

Markdown Code

```
a) chocolate
b) fruit
  1. apples
  1. bananas
  1. annanas
c) vegetables
   A. carrots
   D. leak
```

Rendered Text

a) chocolate

b) fruit

    0.1 apples
    0.2 bananas
    0.3 annanas

c) vegetables

    A. carrots
    B. leak

Markdown Inline Formatting

Markdown Code

```
*text in italics*
**bold text**
`typewriter / code`
~~strike out~~
text in ^superscript^
text in ~subscript~
[underlined text]{.underline}
```

Rendered Text

text in italics

bold text

`typewriter / code`

~~strike out~~

text in $^{\text{superscript}}$

text in $_{\text{subscript}}$

underlined text

Markdown Block Quote

Markdown Code

```
> Block quotes are
easy to make.
>
> They can contain lists
>
>   1. like
>   1. this
```

Rendered Text

> *Block quotes are easy to make.*
>
> *They can contain lists*
>
> 1. *like*
> 2. *this*

Markdown Code Block

Markdown Code

~~~
```
# an R code block
ab <- a + b
if(ab > 1){
  print("Too Big!")
} else if(ab < -1){
  print("Too Small!")
}
x <- 1:1000
y <- x^ab
```
~~~

Rendered Text

```r
# an R code block
ab <- a + b
if(ab > 1){
  print("Too Big!")
} else if(ab < -1){
  print("Too Small!")
}
x <- 1:1000
y <- x^ab
```

Markdown Links

Markdown Code

```
<https://google.com>
[inline link](https://bookdown.org/yihui/rmarkdown/)
[e-mail](mailto:dries.debeer@ugent.be)
```

Rendered Text

https://google.com

inline link

e-mail

Markdown Figures

Markdown Code

```
![Monkey](https://st2.depositphotos.com
          /2927537/7025/i/600/deposit
          photos_70253417-stock-photo-
          funny-monkey-with-a-red.jpg)
```

Rendered Text

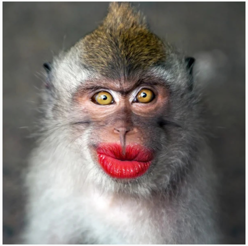


Figure 1: Monkey

Markdown Tables

Markdown Code

```
Left     Right  Center   Default
------ ------- -------- -------
  some    other     47    .55
values   value  55       .14
here      here    3        .07
------ ------- -------- -------

Table:  Simple table syntax.
The column headers may be omitted.
```

Rendered Text

Left	Right	Center	Default
some	other	47	.55
values	value	55	.14
here	here	3	.07

Table 1: Simple table syntax. The column headers may be omitted.

Markdown Dialects

There are multiple Markdown *dialects*. Although the formatting rules are generally the same, there are small differences.

- We presented Pandoc's Markdown.
- A comprehensive overview of formatting rules is found here
- As an example, Github uses its own Markdown *dialect*

Pandoc - a Markdown Converter

Because Markdown has simple formatting rules, it is easy to convert it to other formatting languages like html, LaTeX, or .docx.

Pandoc is software that does exactly this.

When the rmarkdown-package is installed, you can convert Markdown code into html using one button. Under the hood, Pandoc is used.

Pandoc - a Markdown Converter

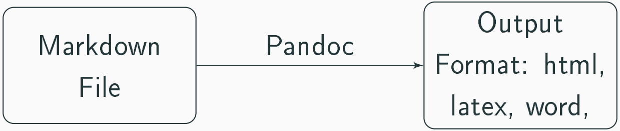


Figure 2: Pandoc converts Markdown Files

Exercises

Exercise

Edit a Markdown document

1. Open the RStudio project named “Rmarkdown”.
2. Open “Exercise-1.md”.
3. Edit the markdown file so that it includes:
   - multiple headings
   - a link to an existing webpage
   - a figure or a table
   - a footnote
4. click the “Preview” button, and create an html version.

Alternative Exercise

Create a new Markdown document in RStudio

1. In RStudio: `File` > `New File` > `Markdown File` .
2. Edit the markdown file so that it includes:
   - multiple headings
   - a link to an existing webpage
   - a figure or a table
   - a footnote
3. click the “Preview” button, and create an html version.

Markdown

Where and when should I use Markdown?

- In README files (not only on Github)
- In RMarkdown documents!
- ...

RMarkdown II

RMarkdown

RMarkdown is a **document format** (.Rmd) that combines

- Markdown formatted text
- chunks of (R-)code

→ Idea of *Literate Programming*.

Because code (for data-analysis) is embedded in the text, changes in the code (for data-analysis) are automatically included in the text when rendered.

→ **Reproducibility!**

RMarkdown

rmarkdown is also an R-package made to work with RMarkdown (.Rmd) files.

- Has an RStudio integration
- The engine for rendering/exporting RMardown documents is knitr

An RMarkdown Document

An RMarkdown document has three basic components

1. Meta-data (YAML-section)
2. Text (in Markdown format)
3. Code (for instance R-code, in chunks)

1. Meta-data - YAML

You can find the meta data

- on top of the document
- between ---
- in YAML (= YAML ain’t markup language)

```
---
title: "Markdowm is Fun"
author: "Dries Debeer"
output: html_document
---
```

Output Format

The YAML header specifies the output-format

- Document
    - .html
    - .pdf (via latex)
    - .docx
- Presentation
    - .html (ioslides or slidy)
    - .pdf (via latex beamer)
    - .pptx
- Shiny-app
- Blog
- Book
- Website
- ...

Output Format

Supported output formats in the YAML header are:

```
context_document         beamer_presentation
github_document          ioslides_presentation
html_document            powerpoint_presentation
latex_document           slidy_presentation
md_document
odt_document
pdf_document
rtf_document
word_document
```

More output formats via extension packages (blogdown, bookdown, ...)

1. Meta-data - YAML

The YAML header

- can contain many options
- can be edited by hand
- BUT, a YAML header is automatically created in RStudio

In RStudio: `File` > `New File` > `R Markdown File`

2. Text - Markdown

Below the YAML header, you can add text in the Markdown format.

All the formatting rules above apply!

3. Code Chunks - R

Code Chunks

- Contain executable code (R code)
- Between ```

~~~
```{r}
library(car)
y <- rnorm(20)
```
~~~

3. Code Chunks - R

In RStudio you can add a new chunk using

- `Ctrl + Alt + I`
- click the green “insert” bottom at the top of the editor pane

3. Code Chunks - R

Chunk name

Although knitr automatically generates a chunk name. Often it helps to name each chunk.

~~~
```{r create-y}
y <- rnorm(20)
```
~~~

- Helps for finding errors.
- Each chunk requires a unique name.

3. Code Chunks - R

Chunk options

~~~
```{r create-y, eval = TRUE, echo = FALSE}
y <- rnorm(20)
```
~~~

For a complete overview of chunk options look here and here.

Chunk Options

Name	Value	Related to	Description
`eval`	`TRUE`	code evaluation	If FALSE, knitr will NOT run the code in the chunk.
`include`	`TRUE`	code evaluation	If FALSE, knitr will run the chunk but not include the chunk in the final document.
`fig.align`	`“default”`	plots	How to align graphics in the final document. One of “left”, “right”, or “center”.
`fig.cap`	`NULL`	plots	A character string to be used as a figure caption.
`fig.height`	7	plots	The height for plots created by the chunk (in inches).
`fig.width`	7	plots	The width for plots created by the chunk (in inches).

Chunk Options

Name	Value	Related to	Description
`echo`	`TRUE`	results	If FALSE, knitr will not "echo" the code in the chunk above it's results in the final document.
`results`	"`markup`"	results	If "hide", knitr will not display the code's results in the final document. If "hold" ,knitr will delay displaying all output pieces until the end of the chunk. If "asis", knitr will pass through results without reformatting them (useful if results return raw HTML,etc.)
`message`	`TRUE`	results	If FALSE, knitr will not display any messages generated by the code.
`warning`	`TRUE`	results	If FALSE, knitr will not display any warning messages generated by the code.

3.5 Inline Code

It is also possible to add R-output inline.

- In the middle of the Markdown text
- via `` `r <R-expression>` ``

```
### Data Description
There is one variable with `r length(y)` observations. The mean
is `r round(mean(y), 2)` (SD = `r round(sd(y), 2)`).
```

Rendering RMarkdown

An RMarkdown document contains chunks of R-coded embedded in markdown text.

knitr executes the code chunks **In a new R-session**, transforms the results into markdown formatted text that replace the code chunks.

pandoc converts the Markdown text (include the results of the code) into the chosen output format (like html).

Rendering RMarkdown

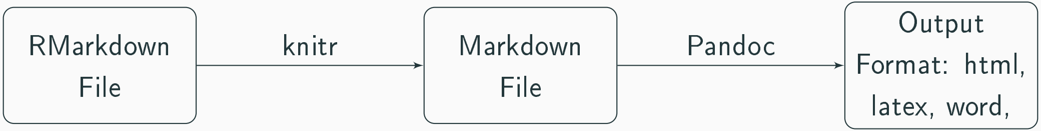


Figure 3: Rendering RMarkdown

Rendering RMarkdown

To create the output file you can:

- Use the *knit* button on top of the editor pane
- Hit `Ctrl + Shift + K`
- Enter `rmarkdown::render(“path/to/file.Rmd”)` in the console

To get an idea what is happening behind the scene, try out `knitr::knit(“path/to/file.Rmd”)`.

→ The result is a **Markdown** document based on the **RMarkdown** document.

Rendering RMarkdown

Advice: Use the shortcut `Ctrl + Shift + K` or the *knit* button.

They make sure that the *knitting* is done in a new R session.

- Objects in the work space of the current R-session are invisible.
- Similar to “Frequently and completely restart!”

Exercises

Exercise

Create an R Markdown document in RStudio

1. Try out different output formats
2. Add or change a code chunk
3. Add or change a code chunk that results in a plot
4. Try out different chunk options
5. Add an R result inline

knit after you made changes to see how the output file is changed.

Alternative Exercise

Open “Rmarkdown-example.Rmd” in “example-project_2”

1. Try out different output formats
2. Add or change a code chunk
3. Add or change a code chunk that results in a plot
4. Try out different chunk options
5. Add an R result inline

knit after you made changes to see how the output file is changed.

What About Tables?

With the `knit::kable()` function in a code chunk, you can add a table to the rendered document.

~~~
```{r create-y, eval = TRUE, echo = FALSE}
y <- rnorm(20)
knitr::kable(t(summary(y)),
             digits = 2,
             caption = "Summary of Y")
```
~~~

What About Tables?

The result of using `knit::kable()`, is a Markdown formatted table, which can than be converted to the chosen output format

```
Table: Summary of Y


|  Min.| 1st Qu.| Median|  Mean| 3rd Qu.| Max.|
|-----:|-------:|------:|-----:|-------:|----:|
| -2.12|   -0.64|  -0.21| -0.02|    0.75| 1.51|
```

What About Tables?

For more formatting options for tables, the kableExtra-package is available.

RMarkdown

Where and when should I use RMarkdown?

- For sharing work with collaborators.
- For presenting results
- For writing a submission ready publication (papaja)?
- ...

My take:

- I use RMarkdown when I collaborate.
- For anything between (first) data processing and writing the final submission.
- Not for (first) data processing → R-scripts
- Writing the final submission → LaTeX

RMarkdown

There are many more features and possibilities

- Creating multiple similar reports via a loop.
- Including citations using a BibTex database
- Interactive documents
- ...

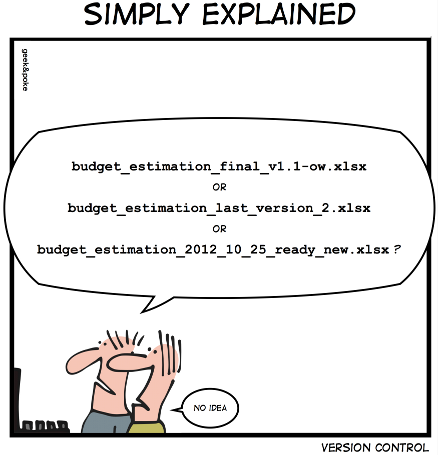
Learn all about it in The Defenitive Guide.

Version Control via Git and Github

Version Controlling

- Motivation
- Setup
- Work Flows
- Recommendations
- Resources

Motivation


SIMPLY EXPLAINED
geek&poke
budget_estimation_final_v1.1-ow.xlsx
OR
budget_estimation_last_version_2.xlsx
OR
budget_estimation_2012_10_25_ready_new.xlsx ?
NO IDEA
VERSION CONTROL

Motivation 1

Single Author Projects

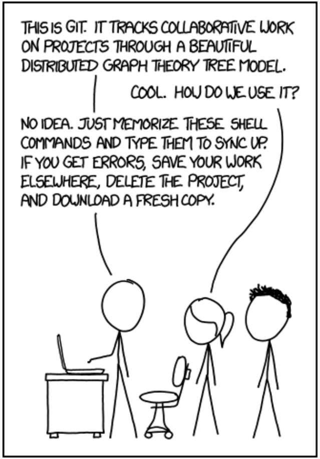
- Implementation of long term change history
  - What has been changed?
  - When was it changed?
- No ridiculous file names
- No archive sub folder
- Accessibility for others ('Open Science')
- Additional safety net
- ...

Motivation 2

Collaborations

- Who has changed what when exactly?
- Clear, current project state
- No annoying mail attachments or file-sharing platforms
- Parallel work easily possible
- Possibility of hierarchical responsibilities
- ...

But...

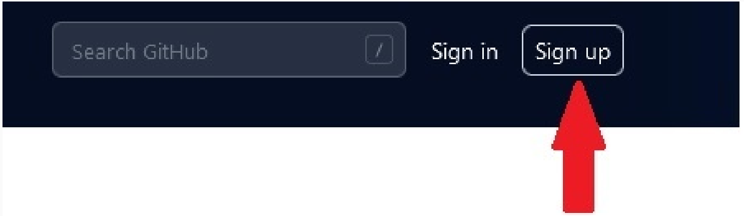

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Prerequisits

- Git-Installation
- RStudio-Installation (git user interface)
  → Alternatives: Shell, Gitkraken, SmartGit, ...
- Github account (online repository)
  → Alternatives: Bitbucket, Gitlab, ...
- Connect everything

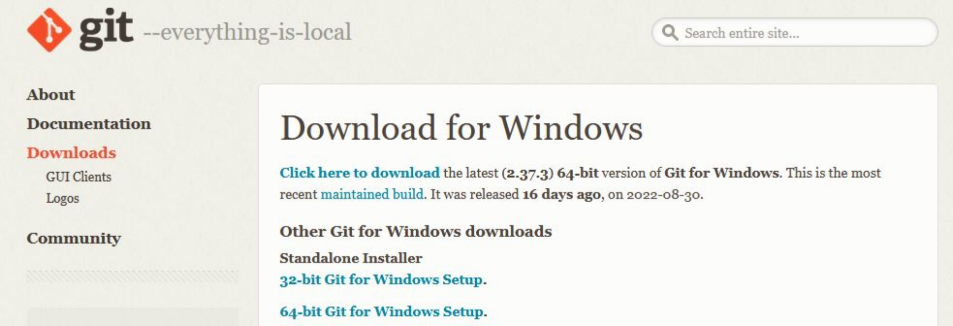
Register at Github

Github



Download git

Download Git



Download for Windows

Click here to download the latest (2.37.3) 64-bit version of Git for Windows. This is the most recent maintained build. It was released 16 days ago, on 2022-08-30.

Other Git for Windows downloads

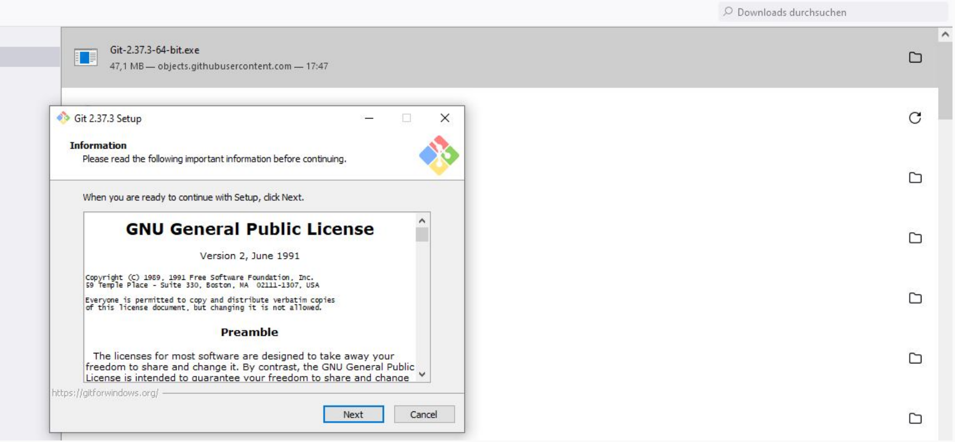
Standalone Installer

32-bit Git for Windows Setup.

64-bit Git for Windows Setup.

Install git

Install git into a folder in which you have sufficient rights (user folder if necessary)



Git-2.37.3-64-bit.exe
47,1 MB — objects.githubusercontent.com — 17:47

Git 2.37.3 Setup

Information

Please read the following important information before continuing.

When you are ready to continue with Setup, click Next.

GNU General Public License

Version 2, June 1991

```
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA  02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
```

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change

https://gitforwindows.org/

Configure git

git can be configured for Github via R

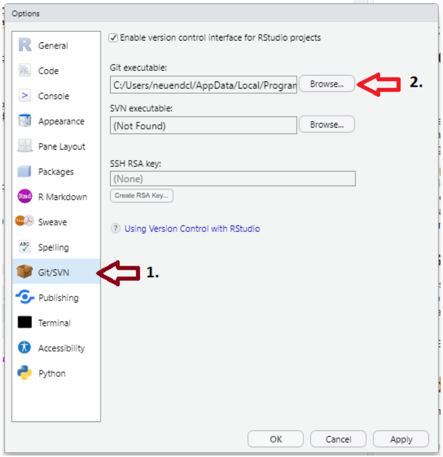
```
# if necessary, install usethis
# install.packages("usethis")

library(usethis)
use_git_config(user.name = "Username",
               user.email = "user@gmail.com")
```

Configure git

Check if the configuration was successful via the command line (cmd).

```
git config --global --list
```

Connect git and RStudio



☑ Enable version control interface for RStudio projects

Git executable: C:/Users/neuendcl/AppData/Local/Progran

SVN executable: (Not Found)

SSH RSA key: (None)

1.

2.

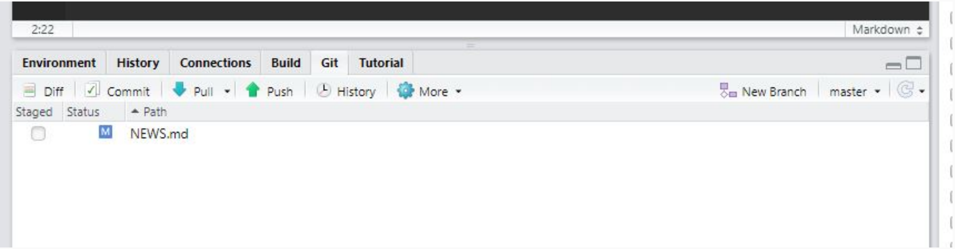
PAT for HTTPS

How does Github now we should have access to our repositories?
→ via a **P**ersonal **A**ccess **T**oken for HTTPS

- Go to https://github.com/settings/tokens
- add the scopes “repo”, “user”, and “workflow”
- Generate the token
- Use this token as a password if RStudio asks you
- Use `gitcreds::gitcreds_set()`

RStudio

The RStudio user interface to Git looks like this



Staged	Status	Path
□	M	NEWS.md

Work flow 1

Creating a repository

- Create an **online repository** (e.g. on Github)
  - Use an R specific `.gitignore`
  - Initialize with a short readme (`.md`)
- Clone the repository to your local machine via RStudio as a new project
  - → New project
  - → Version Control
  - → git
- An R-Project is added automatically to the existing repository

Excursion: gitignore

- Plain text file
- Which files should not be tracked by git?

  → These then only exist locally in their current version!
- Options
  - Single files
  - Folders
  - Specific data types
  - Combinations of the above
- Use cases
  - Large files (Data, images, ...)
  - Auxiliary files (e.g. created during latex compilation)

Work flow 2

Working with an existing repository

- Before working: Synch your local repo (**Pull** or **clone**)
- Perform changes in your local repository
  → Create/modify/delete files
- **Stage** your changes
- **Commit** your changes (aka new version)
- **Push** your commit(s) (online repository is updated)

Excursion: merge

Conflicts between different updated versions

- Common when working collaboratively
- Discrepancies between your own different local repos → Git communicates these and indicates conflicts
- Select the desired changes
- Stage selection, commit and push

Excursion: Branches

Multiple parallel versions of a project within one repository

- Common e.g. in areas like software development
- e.g. one stable and one development branch
- Only certain modifications should be made in the stable branch
- **Note**: RStudio GUI has limited support for this

Your impressions?

Recommendations

- Keep it simple!
    - If not necessary, no branches/forks/pull requests
- Have meaningful commits
- Keep it lean (no big files)
- Avoid using the Github homepage for working within the repository

Resources

Git + RStudio Resources

- Small Intro
- Happy Git with R
- R Packages and Git

General Git Resources

- Git Book

Exercises

Exercises

- Create an example repository
- Create some descriptive analysis of the `mtcars` data set
- Version control your work

Good programming practices

“Write code for humans, not for machines!”

Code Style

Invest time in writing readable R-code.

- It will make collaborations easier
- It will make debugging easier
- It will make your analyses more reproducible

There is a complete *tidyverse* style-guide .

Go easy on your eyes

- with spaces before and after: `- + / * = <- < == >`
- always use `<-` for assignments
- only use `=` in function calls
- use indentation (largely automatic in RStudio)
- `CamelCaseNames` vs `snake_case_names`
- be consistent!
- wrap long lines at column 70-80 (Rstudio)

White space

```
new_var=(var1*var2/2)-5/(var3+var4)

# versus

new_var <- (var1 * var2 / 2) - 5 / (var3 + var4)
```

Indentation

```r
for(name in names){formula=as.formula(paste0("y~.-",name))
fit<-lm(formula,data=my_data)
coefs[["name"]]=coef(fit)
print(name)
print(summary(fit))}

# versus

for(name in names){
  formula <- as.formula(paste0("y~.-", name))
  fit <- lm(formula, data = my_data)
  coefs[["name"]] <- coef(fit)
  print(name)
  print(summary(fit))
}
```

Wrap long lines

```
final_results <- data.frame(first_variable =
sqrt(results$mean_squared_error), second_variable =
paste0(results$condition, results$class, sep = ":"),
third_variable = results$bias)

# versus

final_results <- data.frame(
  first_variable = sqrt(results$mean_squared_error),
  second_variable = paste0(results$condition,
                           results$class, sep = ":"),
  third_variable = results$bias)
```

Go easy on your mind

- use meaningful names: “self-explainable”
- always write the formal arguments in function calls (except the first)
- benefit from autocompletion (`<tab>`) => embrace longer names
- use `TRUE` and `FALSE` not `T` and `F`
- comment, comment, comment
    - NOT what (should be clear from the code)
    - but why
    - explain the reasoning, not the code

Use meaningful names

```r
V <- myFun(m1_B)

# versus

RMSE_age_gender <- get_RMSE(lm_age_gender)
```

Programming advice

Use verbs for functions and nouns for other objects.

Write formal arguments

Benefit from auto completion using tab

```
m1_B <- lm(outcome ~ age*gender,
           exp1, condition_1, freq)

# versus

lm_age_gender <- lm(outcome ~ age * gender,
                    data = exp1,
                    subset = condition_1,
                    weigths = freq)
```

Comment, comment, comment

```r
## Start every Rscript with a comment that explains
##  what the code in the script does, why it does
##  this, and to which project it belongs.
##  Your future self will be very thankful!
##
## Mention which packages you are using in this Rscript.

## Use sections to separate chunks ----------------------

## Maybe even subsections ===============================

## Recode variables so that missings are coded as "NA"
dat[dat %in% c(99, 999)] <- NA  # missings coded 99 or 999
```

Keep your code slim

Try to limit your *package-dependencies*.

Only load `library()` the packages that you absolutely need. If you are only using `dplyr`, it does not make sense to load the complete `tidyverse`.

Controversial: when possible, use the `::` operator (and consider not loading the package). `<package>::<function>`

- explicit dependencies
- less name conflicts

Wrap Up

General Advice

- Investing time in learning R pays off
- It's a steady learning curve
- Learn from masters
- Rewrite important code - the first attempt is usually not the best approach

General R Advice

- Document well
- Use a consistent style
- Write functions
- Split long functions in smaller ones
- Write wrappers
- Use Iteration (don't copy paste)
- Use matrix operations and vectorized functions instead of loops
- Use git

Literature Recommendations

R Resources

- Avanced R Ed. 1
- Avanced R Ed. 2
- R Inferno
- R Packages
- Clean Code

Reproducible Research

- CRAN Task View
- Data Carpentry
- State of Alaska's Salmon and People

Thank you for your attention!

Thank you for your attention!

Questions? Remarks?