

What they forgot to teach you about R

Scientific workflows: Tools and Tips 

4/20/23

Who am I?

- Ecologist, PhD student for some years and now scientific programmer

What is this lecture series?

Scientific workflows: Tools and Tips



Every 3rd Thursday 4-5 p.m. Webex

- One topic from the world of scientific workflows
- Topics range from R programming over notetaking, literature management tools and more
- For topic suggestions send me an email
- If you don't want to miss a lecture
 - Check out the [lecture website](#)
 - [Subscribe to the mailing list](#)
- Slides provided [on Github](#)

What they forgot to teach you about R

It's a book by J. Bryan and J. Hesters



Artwork by [Allison Horst](#), CC BY 4.0

Chaotic projects and workflows ...

... can make even small changes frustrating and difficult.



Artwork by Allsion Horst, CC BY 4.0

What they forgot to teach you about R

Background

- Reproducibility 

 - Can someone else reproduce my results?

- Reliability 

 - Will my code work in the future?

- Reusability 

 - Can someone else actually use my code?

Today: Talk **best practice** rules to write clean, clear and maintainable code.

In other words: How to clean the kitchen?

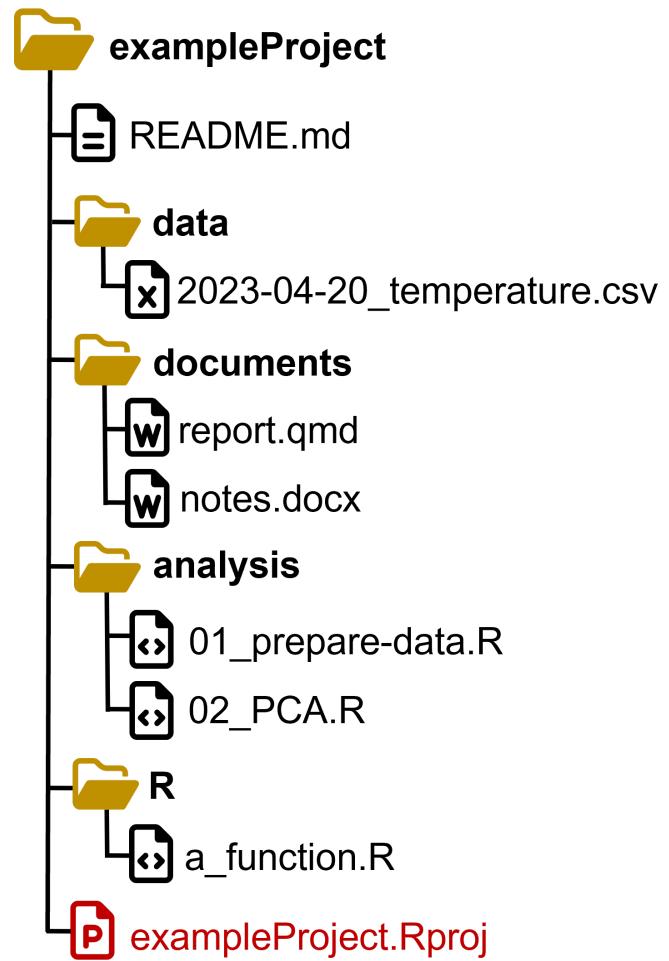
First things first

Project setup and structure

Use R Studio projects

Always make your project an R Studio Project (if possible)!

- Keep your files together
- An R Studio Project is just a normal directory with an ***.Rproj** file
 - double-click this file to open your project in R Studio
- Advantages:
 - Easy to navigate in R Studio
 - Project root is the working directory
 - Open multiple projects in separate R Studio instances



Create an R Studio Project

From scratch:

1. File -> New Project -> New Directory -> New Project
2. Enter a directory name (this will be the name of your project)
3. Choose the directory where the project should be initiated
4. Create Project

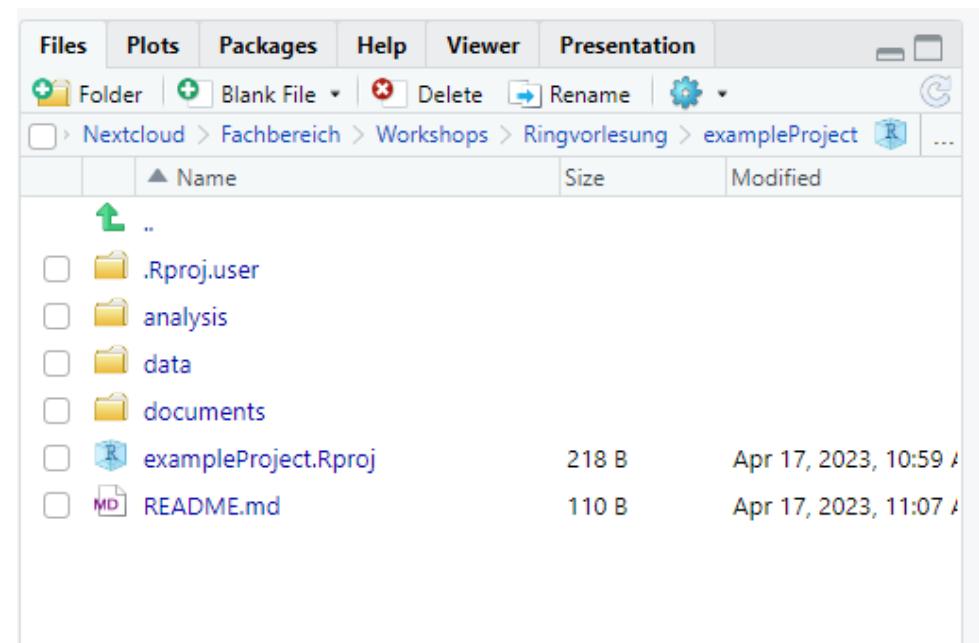
Associate an existing folder with an R Studio Project:

1. File -> New Project -> Existing Directory
2. Choose your project folder
3. Create Project

Navigate an R Studio Project

You can use the **Files** pane in R Studio to interact with your project folder:

- Navigate and open files
- Create files and folders
- Rename and delete
- ...



Set up your project

R Studio offers a lot of settings and options.

So have a  and check out **Tools -> Global Options** and all the other buttons.

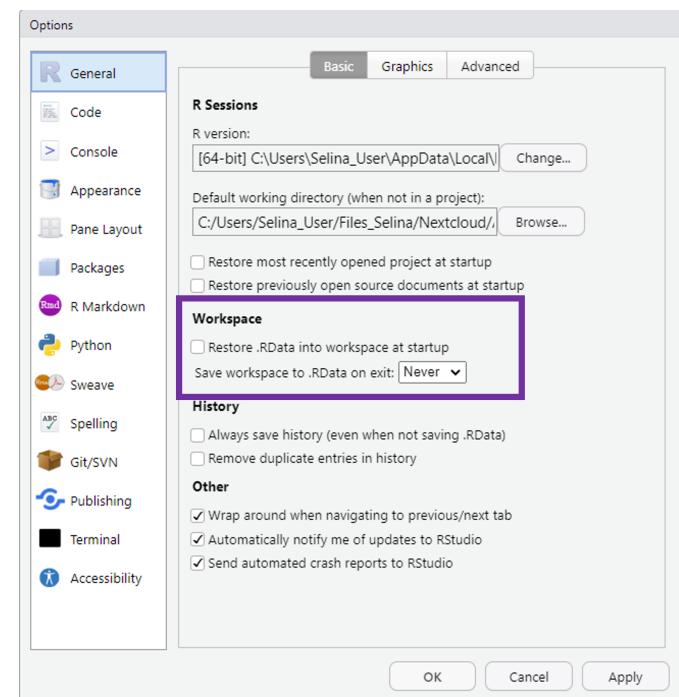
- R Studio cheat sheet that explains all the buttons
- Update R Studio from time to time to get new settings (**Help -> Check for Updates**)

Set up your project

R Studio offers a lot of settings and options.

Most important setting for reproducibility:

- Never save or restore your work space as **.Rdata** -> You always want to start working with a clean slate



Name your files properly

Your collaborators and your future self will love you for this.

Principles¹

File names should be

1. Machine readable
2. Human readable
3. Working with default file ordering

1. Machine readable file names

Names should allow for easy **searching**, **grouping** and **extracting** information from file names.

- No space & special characters

Bad examples 

-  2023-04-20 temperature göttingen.csv
-  2023-04-20 rainfall göttingen.csv

Good examples 

-  2023-04-20_temperature_goettingen.csv
-  2023-04-20_rainfall_goettingen.csv

2. Human readable file names

Which file names would you like to read at 4 a.m. in the morning?

- File names should reveal the file content
- Use separators to make it readable

Bad examples 

-  01preparedataforanalysis.R
-  01firstscript.R

Good examples 

-  01_prepare-data-for-analysis.R
-  01_lm-temperature-trend.R

3. Default ordering

If you order your files by name, the ordering should make sense:

- (Almost) always put something numeric first
 - Left-padded numbers (01, 02, ...)
 - Dates in YYYY-MM-DD format

Chronological order

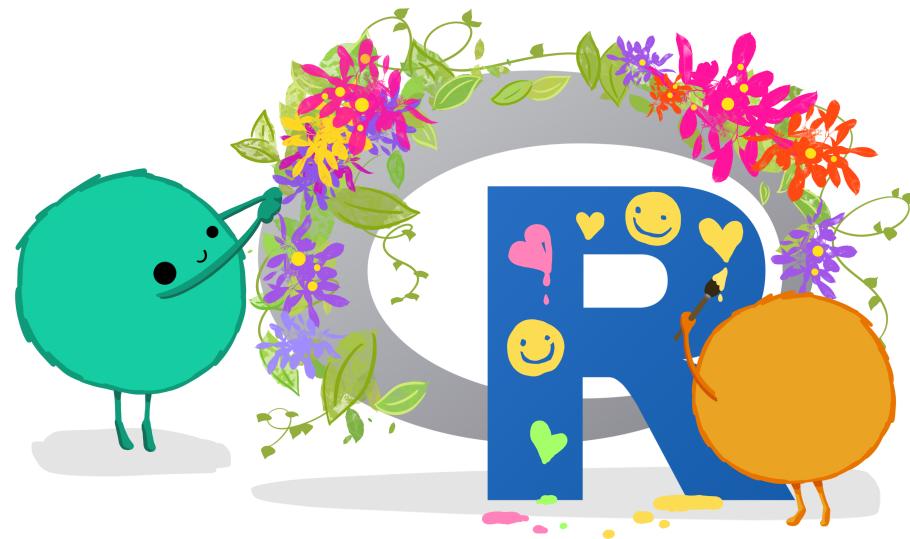
-  2023-04-20_temperature_goettingen.csv
-  2023-04-21_temperature_goettingen.csv

Logical order

-  01_prepare-data.R
-  02_lm-temperature-trend.R

Let's start coding

Write beautiful code



Artwork by [Allsion Horst](#), CC BY 4.0

Standard code structure

1. General comment with purpose of the script, author, ...
2. `library()` calls on top
3. Set default variables and global options
4. Source additional code
5. Write the actual code, starting with loading all data files

```
# This code replicates figure 2 from the
# Baldauf et al. 2022 Journal of Ecology pa
# Authors: Selina Baldauf, Jane Doe, Jon Do

library(tidyverse)
library(vegan)

# set defaults
input_file <- "data/results.csv"

# source files
source("R/my_cool_function.R")

# read input
input_data <- read_csv(input_file)
```

Mark sections

- Use comments to break up your file into sections

```
# Load data -----
input_data <- read_csv(input_file)

# Plot data -----
ggplot(input_data, aes(x = x, y = y)) +
  geom_point()
```

- Insert a section label with **Ctrl/Cmd + Shift**
- Navigate sections in the file outline

Modularize your Code

- Don't put all your code into one long file (hard to maintain)
 - Write multiple files that can be called sequentially
 - E.g. `01_prepare-data.R`, `02_lm-temperature-trend.R`, `03_plot-temperature-trends.R`
 - Write functions that can be called in other scripts
 - Use the `source()` function to source these files
 - Have one main workflow script that calls these functions sequentially

Use save paths

To read and write files, you need to tell R where to find them.

Common workflow: set **working directory** with `setwd()`, then read files from there. But to this Jenny Bryan said:

If the first line of your R script is

`setwd("C:\Users\jenny\path\that\only\I\have")`

I will come into your office and SET YOUR COMPUTER ON FIRE .

Why?

This is **100% not reproducible**: Your computer at exactly this time is (probably) the only one in the world that has this working directory

Avoid `setwd()` if it is possible in any way!

Avoid `setwd()`

Use R Studio projects

- Project root is automatically the working directory
- Give your project to a friend at it will work on their machine as well

Instead of

```
# my unique path from hell with white space and special characters
setwd("C:/Users/Selina's PC/My Projects/Göttingen Temperatures/temperatures")

read_csv("data/2023-04-20_temperature_goettingen.csv")
```

You just need

```
read_csv("data/2023-04-20_temperature_goettingen.csv")
```

If you don't use R Studio Projects, have a look at the `{here}` package for reproducible paths

Coding style - Object names

- Variables and function names should only have lowercase letters, numbers, and `_`
- Use `snake_case` for longer variable names
- Try to use concise but meaningful names

```
# Good
day_one
day_1

# Bad
DayOne
dayone
first_day_of_the_month
dm1
```

Coding style - Spacing

- Always put spaces after a comma

```
# Good  
x[, 1]
```

```
# Bad  
x[ , 1]  
x[,1]  
x[ ,1]
```

Coding style - Spacing

- Always put spaces after a comma
- No spaces around parentheses for normal function calls

```
# Good
mean(x, na.rm = TRUE)

# Bad
mean (x, na.rm = TRUE)
mean ( x, na.rm = TRUE )
```

Coding style - Spacing

- Always put spaces after a comma
- No spaces around parentheses for normal function calls
- Spaces around most operators (`<-`, `==`, `+`, etc.)

```
# Good
height <- (feet * 12) + inches
mean(x, na.rm = TRUE)

# Bad
height<-feet*12+inches
mean(x, na.rm=TRUE)
```

Coding style - Spacing

- Always put spaces after a comma
- No spaces around parentheses for normal function calls
- Spaces around most operators (<- , == , + , etc.)
- Spaces before pipes (%>% , |>) followed by new line

```
# Good
iris %>%
  group_by(Species) %>%
  summarize_if(is.numeric, mean) %>%
  ungroup()

# Bad
iris %>% group_by(Species) %>% summarize_all(mean) %>% ungroup()
```

Coding style - Spacing

- Always put spaces after a comma
- No spaces around parentheses for normal function calls
- Spaces around most operators (`<-`, `==`, `+`, etc.)
- Spaces before pipes (`%>%`, `|>`) followed by new line
- Spaces before `+` in ggplot followed by new line

```
# Good
ggplot(aes(x = Sepal.Width, y = Sepal.Length, color = Species)) +
  geom_point()

# Bad
ggplot(aes(x = Sepal.Width, y = Sepal.Length, color = Species))+geom_point()
```

Coding style - Line width

Try to limit your line width to 80 characters.

- You don't want to scroll to the right to read all code
- 80 characters can be displayed on most displays and programs
- Split your code into multiple lines if it is too long
 - See this grey vertical line in R Studio?

```
# Bad
iris %>%
  group_by(Species) %>%
  summarise(Sepal.Length = mean(Sepal.Length), Sepal.Width = mean(Sepal.Width), Species = )

# Good
iris %>%
  group_by(Species) %>%
  summarise(
    Sepal.Length = mean(Sepal.Length),
    Sepal.Width = mean(Sepal.Width),
    Species = n_distinct(Species)
)
```

Coding style

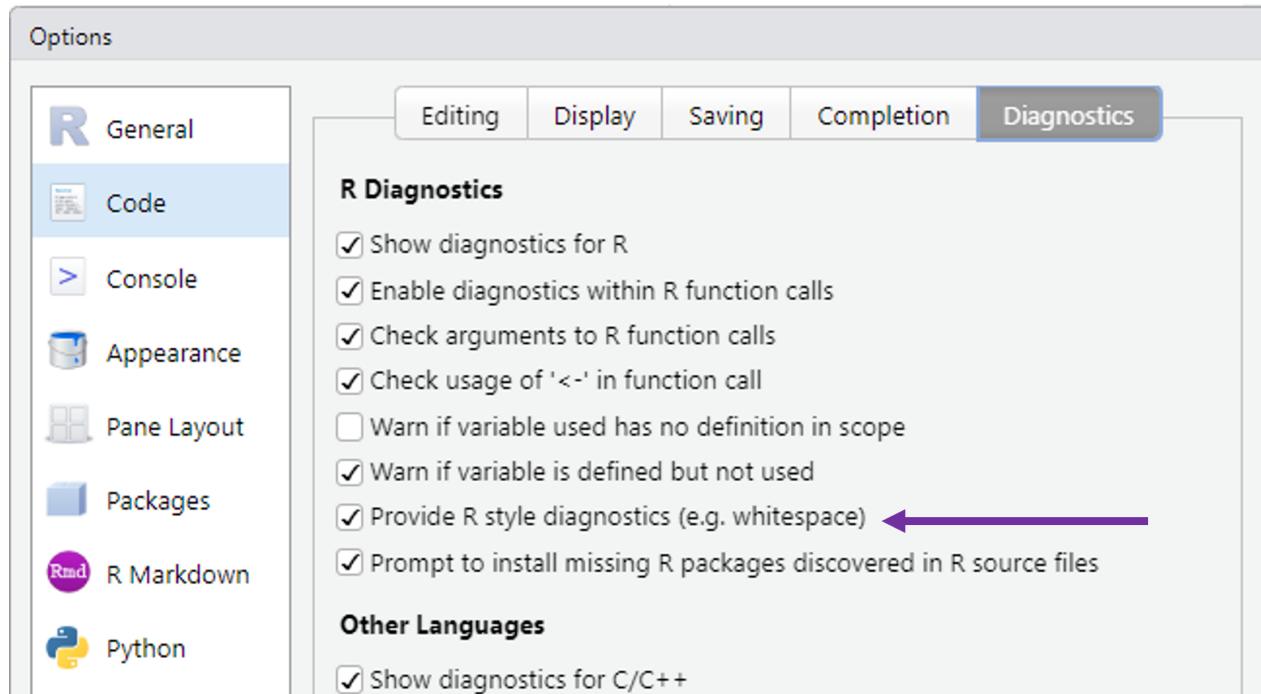
Do I really have to remember all of this?

Luckily, no! R and R Studio provide some nice helpers

Coding style helpers - R Studio

R Studio has style diagnostics that tell you where something is wrong

- Tools -> Global Options -> Code -> Diagnostics



A screenshot of the R code editor in R Studio. The code is:

```
 10 data<-data %>%
 11   group_by( group ) %>%
 12   summarize(
 13     measure=mean(measure,na.rm=TRUE)
 14   )
 15 }
```

A tooltip box appears over line 14, reading "expected whitespace around '=' operator".

Coding style helpers - {lintr}

The `lintr` package analyses your code files or entire project and tells you what to fix.

```
# install the package before you can use it
install.packages("lintr")
# lint specific file
lintr::lint(filename = "analysis/01_prepare_data.R")
# lint a directory (by default the whole project)
lintr::lint_dir()
```

Coding style helpers - {lintr}

The screenshot shows an RStudio interface with the following details:

- Title Bar:** exampleProject - RStudio
- Menu Bar:** File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help
- Toolbar:** Includes icons for New, Open, Save, Print, Go to file/function, and Addins.
- Code Editor:** The file 01_prepare_data.R contains the following code:

```
1 library(tidyverse)
2
3
4 # Load data -----
5
6 TemperatureData <- read_csv("data/my_data.csv")
7
8 # Plot data -----
9
10 TemperatureData<-data %>%group_by(group)%>%summarize(measure = mean(measure, na.rm = TRUE))
```
- Status Bar:** Shows line 6:1 and the status "R Script".
- Console Tab:** Labeled "Console" and "Markers".
- Output Window:** Shows linting results for analysis/01_prepare_data.R:

```
analysis/01_prepare_data.R
S Line 6 [object_name_linter] Variable and function name style should be snake_case or symbols.
S Line 10 [object_name_linter] Variable and function name style should be snake_case or symbols.
S Line 10 [infix_spaces_linter] Put spaces around all infix operators.
S Line 10 [infix_spaces_linter] Put spaces around all infix operators.
S Line 10 [infix_spaces_linter] Put spaces around all infix operators.
S Line 10 [line_length_linter] Lines should not be more than 80 characters.
S Line 10 [spaces_inside_linter] Do not place spaces before parentheses.
```

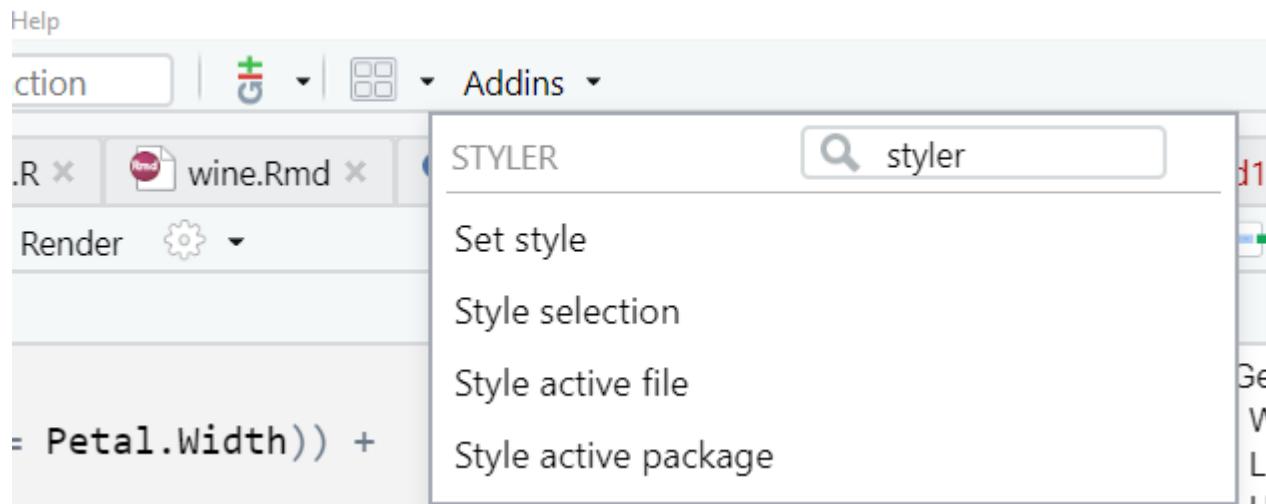
What they forgot to teach you about R

Coding style helpers - {styler}

The `styler` package automatically styles your files and projects according to the tidyverse style guide.

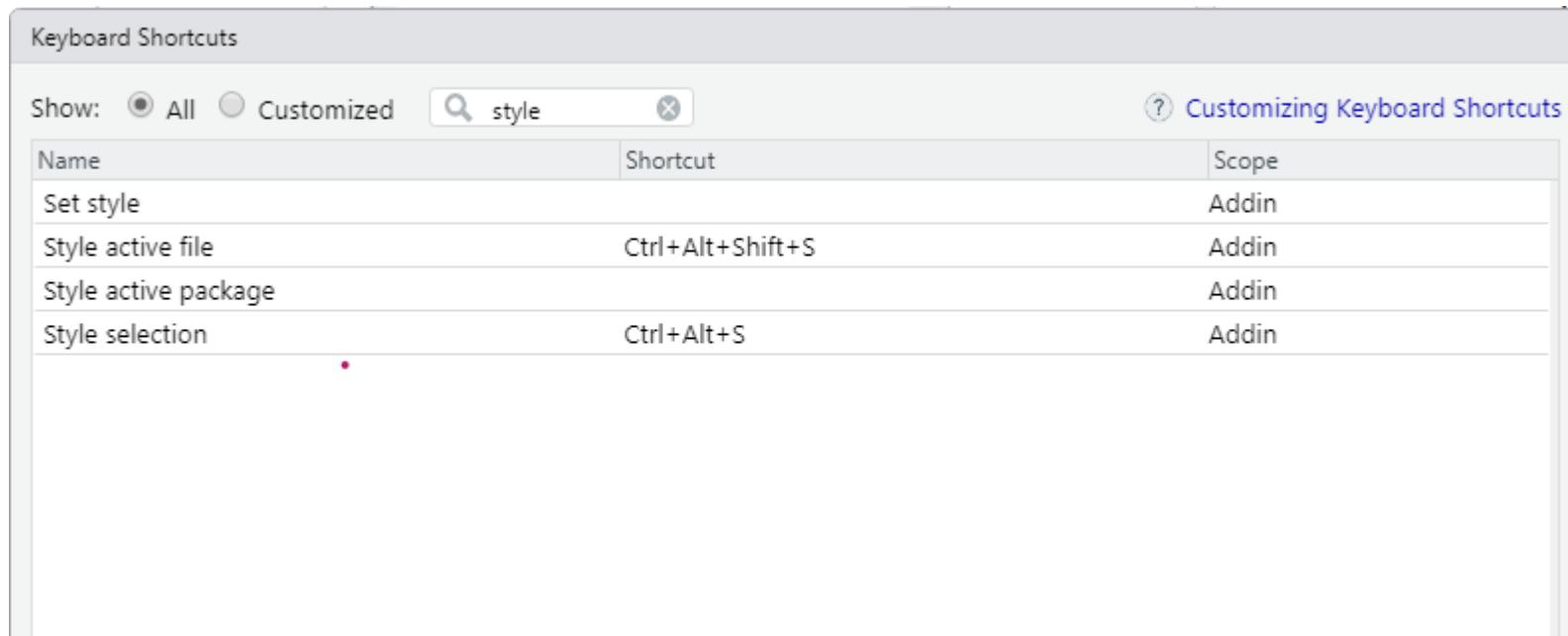
```
# install from CRAN  
install.packages("styler")
```

- Use the R Studio `Addins` for styler:



Coding style helpers - {styler}

- Pro-Tip: Add a custom keyboard short cut to style your files
 - Tools -> Modify Keyboard Shortcuts



Manage dependencies with {renv}

Idea: Have a **project-local environment** with all packages needed by the project

- Keep log of the packages and versions you use
- Restore the local project library on other machines



Why this is useful?

- Code will still work even if packages upgrade
- Collaborators can recreate your local project library with one function
- Explicit dependency file states all dependencies

Check out the [renv website](#) for more information

Manage dependencies with {renv}

```
# Get started  
install.packages("renv")
```

Very simple to use and integrate into your project workflow:

```
# Step 1: initialize a project level R library  
renv::init()  
# Step 2: save the current status of your library to a lock file  
renv::snapshot()  
# Step 3: restore state of your project from renv.lock  
renv::restore()
```

- Your collaborators only need to install the `renv` package, then they can also call `renv::restore()`
- When you create an R Studio project there is a check mark to initialize with `renv`

Summary

Clean projects and workflows ...

... allow you and others to work productively.



Artwork by Allsion Horst, CC BY 4.0

Take aways

There are a lot of things that require minimal effort and that you can start to implement into your workflow NOW

1. Use R Studio projects -> Avoid `setwd()`!
2. Keep your projects clean
 - Sort your files into folders
 - Give your files meaningful names
3. Use `styler` to style your code automatically
4. Use `lintr` and let R analyse your project
5. Consider `renv` for project local environments

Outlook

Of course there is much more:

- Version control with Git
- Using R packages to build a research compendium
- Docker containers for full reproducibility
- ...

But this is for another time

Next lecture

Package your research in an R



A research compendium is a **collection of all the digital parts of your research projects** (data, code, documents). R packages have a similar structure and therefore can be used to publish a fully reproducible version of your project.

Paper: Packaging data analytical work reproducibly using R (and friends)
by Ben Marwick

11th May 4-5 p.m. Webex

- For topic suggestions and/or feedback [send me an email](#)
- [Subscribe to the mailing list](#)

Thank you for your attention

:)

Questions?

References

- What they forgot to teach you about R book by Jenny Bryan and Jim Hester
- Blogpost by Jenny Bryan on good project-oriented workflows
- R best practice blogpost by Krista L. DeStasio
- Book about coding style for R: The tidyverse style guide
- The Turing way book General concepts and things to think about regarding reproducible research
- renv package website
- styler package website
- lintr package website

What they forgot to teach you about R