

# 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

# Background

- Reproducibility 

  - Can someone else reproduce my results?

- Reliability 

  - Will my code work in the future?

- Reusability 

  - Can someone else actually use my code?

Basis: Follow **best practices** to write clean, clear and maintainable code.

# 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

# Clean projects and workflows ...

... allow you and others to work productively.



@allison\_horst

Artwork by Allsion Horst, CC BY 4.0

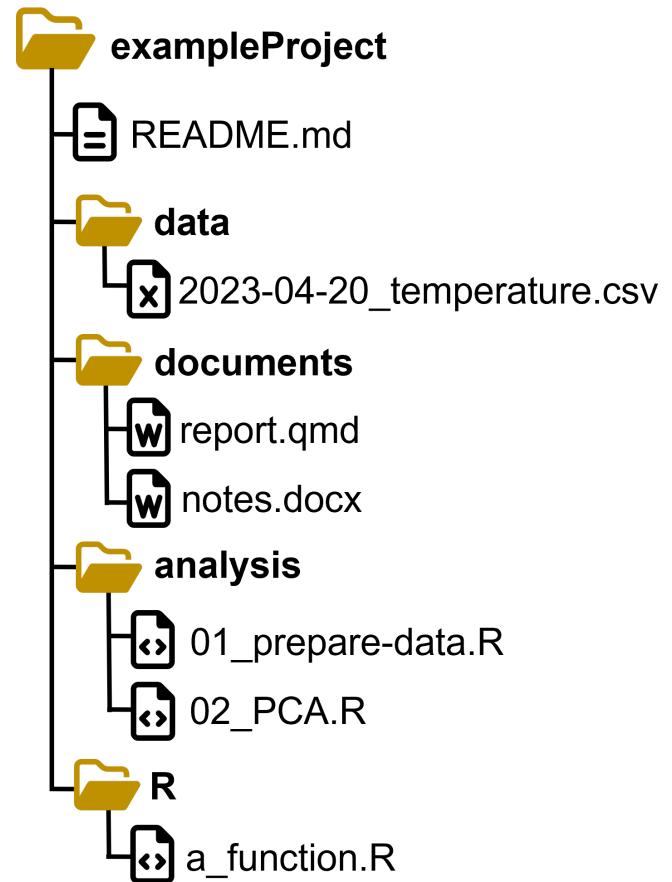
What they forgot to teach you about R

# First things first

Project setup and structure

# Keep your files together

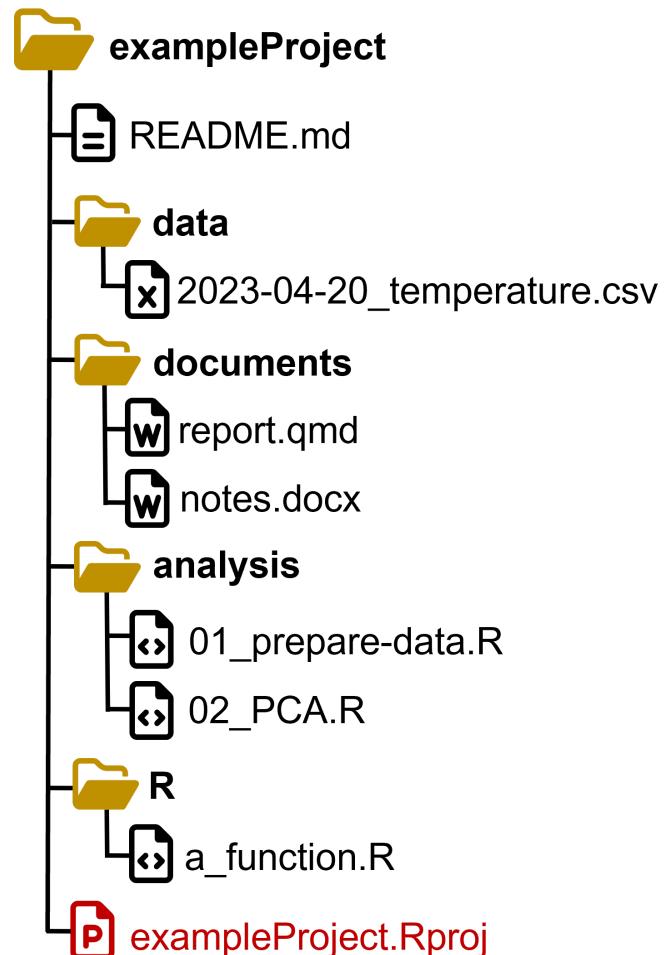
- Everything in one project folder
- Add a README file
  - Explain content of the project, where and how to start, whom to ask questions, ...
- Separate folders for **data/**, **documents/**, **analysis/**, **R/** etc.
- Best structure depends on your type of project



# Use R Studio projects

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

- 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

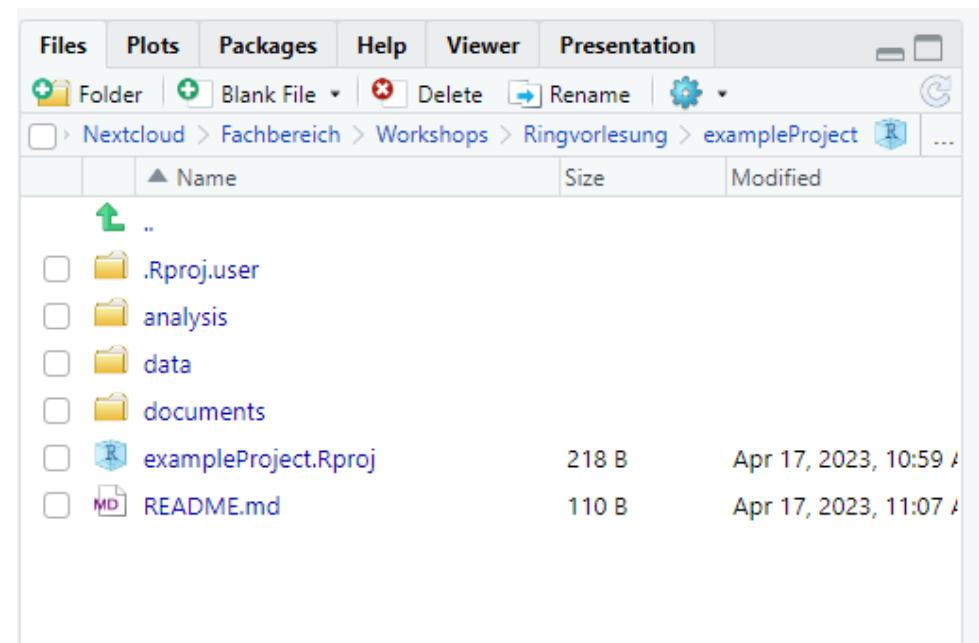
## Associate an existing folder with an R Studio Project:

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.

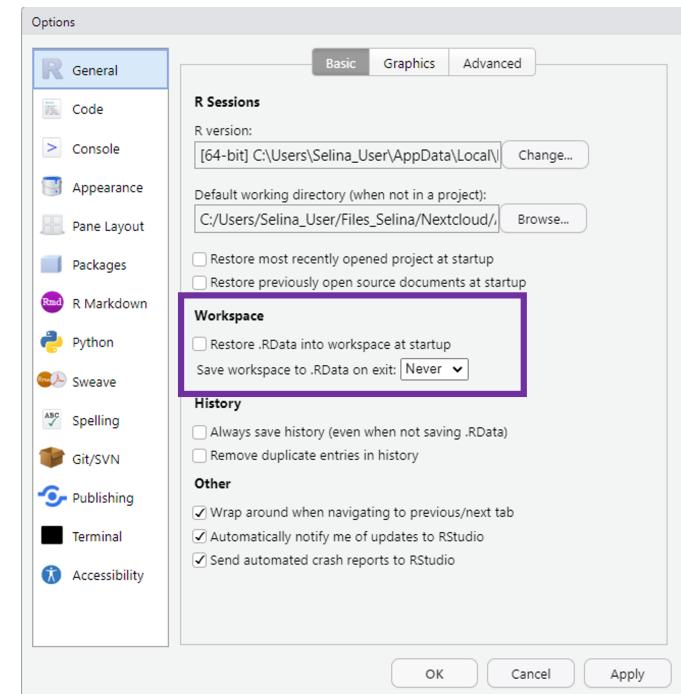
- R Studio cheat sheet that explains all the buttons
- Update R Studio from time to time to get new settings (**Help -> Check for Updates**)

# Set up your project

R Studio offers a lot of settings and options.

Most important setting:

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



Other nice settings:

- Code display settings ([Code -> Display](#)), e.g. highlight colors or rainbow parentheses

# Name your files properly

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

## Principles<sup>1</sup>

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 (if needed).

- No whitespace & special characters
- Separate metadata to work with regular expressions (e.g. `_` to separate metadata and `-` to make it readable)

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
-  helper01\_load-data.R

# Let's start coding

# Follow coding guidelines

“It’s harder to read code than to write it.”

— Joel Spolsky

- Try to write code that others (i.e. future you) can understand
- Follow standards for readable and maintainable code

# 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

# Split your workflow

- Don't put all analysis into one long file
  - Write multiple files that can be called sequentially
    - Store results as `.Rdata` if you don't want always run the script
  - Write functions that can be called in other scripts
- Use the `source()` function to source these files
- Have main workflow files that manage your workflow

# Split your workflow

## 01\_prepare-data.R

```
# Purpose: Calculate daily mean temperature and store in data/
# Authors: Selina Baldauf
library(data.table)

# read raw data
temperatures <- data.table::fread("data/huge_temperature_dataset.csv")

# calculate daily mean temperature
temperature_mean <- temperatures[, mean := mean(temperature), by = c("city", "date")]

# save as .Rdata file
save(temperature_mean, file = "data/temperature_mean.Rdata")
```

## 02\_linear-model-analysis.R

```
# Purpose: Conduct linear model analysis of daily mean temperature in cities
# Authors: Selina Baldauf

load(file = "data/temperature_mean.Rdata")

# linear model
temp_lm <- lm(temperature ~ date, data = temperature_mean)
```

# Split your workflow

## R/helper01\_prepare-data.R

```
# Purpose: Function to read and prepare data for analysis
# Authors: Selina Baldauf
prepare_data <- function(path) {
  the_data <- readr::read_csv(file = path)
  the_data <- the_data %>%
    group_by(city) %>%
    summarize(temperature = mean(temperature))
}
```

## analysis/01\_linear-model-analysis.R

```
# Purpose: Conduct linear model analysis of daily mean temperature in cities
# Authors: Selina Baldauf

input_file <- "data/huge_temperature_dataset.csv"

source("R/helper01_prepare-data.R")

# read and prepare the data
temperature <- prepare_data(path = input_file)
```

# 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")
```

# Avoid `setwd()`

Use the `{here}` package for reproducible paths

- Define the location of your scripts relative to a project root
- `here` will automatically determine the project root based on this
  - `here` will also recognize an `.Rproj` file as a project root
- Build your paths with the `here()` function relative to the project root

```
# Check where my here project root is
here::dr_here()

# set a path relative to the project root and read file
readr::read_csv(here::here("data/2023-04-20_temperature_goettingen.csv"))
```

# Write beautiful code



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

- Every programming language has one/many styleguides
  - For R: [tidyverse style guide](#)
- Style guides define
  - File names and code organization
  - Syntax standards (naming convention, spacing, ...)
  - ...

# 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 - 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
- Avoid re-using names of common functions and variables

```
# Bad
T <- data.frame(temperature = 1:10)
c <- 10
data_frame <- data.frame(temperature = 1:10)
sum <- function(x) mean(x)
```

# 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 before and after `()` in `if` or `for`

```
# Good
if (debug) {
  show(x)
}

# Bad
if(debug) {
  show(x)
}
```

# Coding style - Spacing

- Always put spaces after a comma
- No spaces around parentheses for normal function calls
- Spaces before and after `()` in `if` or `for`
- 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 before and after `()` in `if` or `for`
- 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 before and after `()` in `if` or `for`
- 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

```
# 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

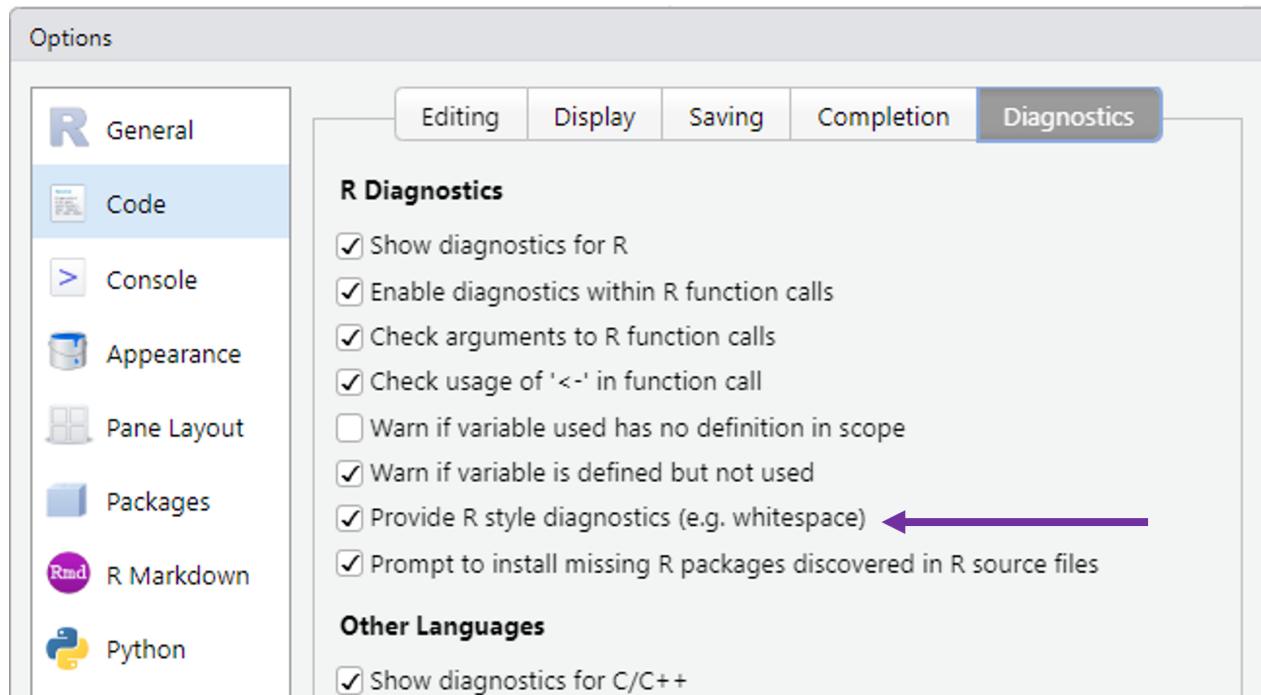
Do I really have to remember all of this?

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



The code editor shows the following R code:

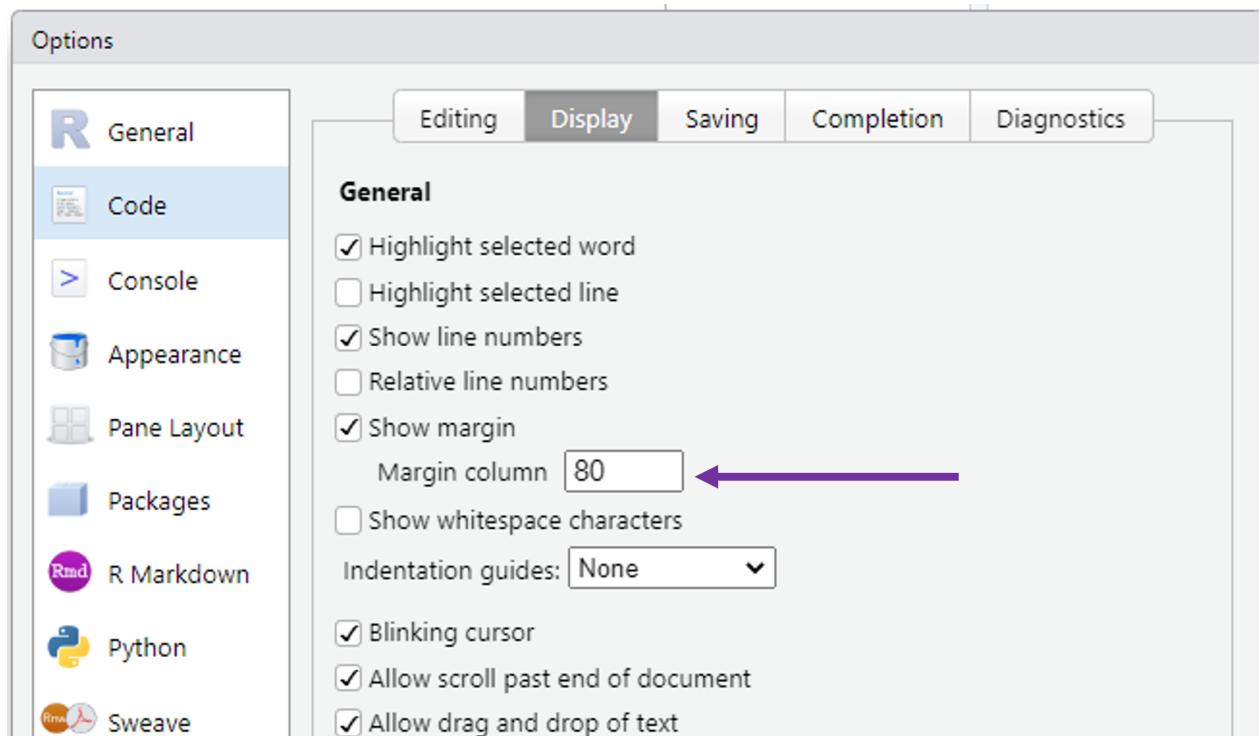
```
10 data<-data %>%
11   group_by( group ) %>%
12   summarize(
13     measure=mean(measure,na.rm=TRUE)
14   )
15 }
```

A tooltip on the right side of the editor window displays the diagnostic message: "expected whitespace around '=' operator".

# Coding style helpers - R Studio

You can show a margin line to help you keep 80 characters line width

- Tools -> Global Options -> Code -> Display



The screenshot shows the R Studio code editor with multiple files open. A vertical dashed line is visible at the 80th character position on every line of code, serving as a margin indicator. The code shown includes library imports, data loading, and basic data manipulation using dplyr.

```
library(tidyverse)
# Load data
data <- read_csv("data/my_data.csv")
# Plot data
data <- data %>%
  group_by(group) %>%
  summarize(
    measure = mean(measure, na.rm = TRUE)
)
```

# 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 the 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 mode R Script.
- Bottom Panel:** Tabs for Console, Terminal, Markers, and Background Jobs. The Markers tab is active, displaying linting errors from the analysis/01\_prepare\_data.R file:

```
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.
```

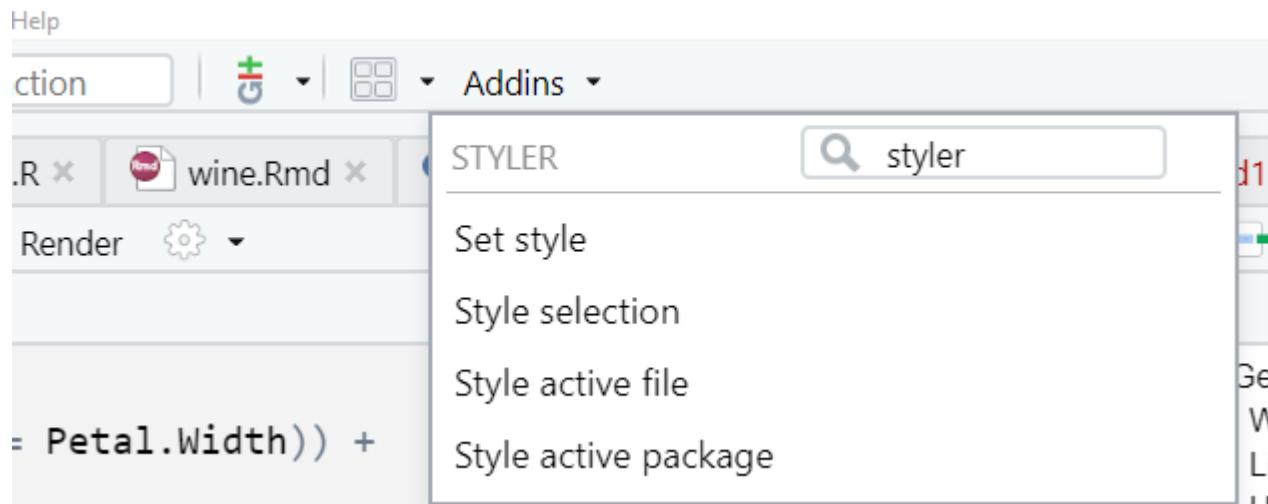
What they forgot to teach you about R

# Coding style helpers - {styler}

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

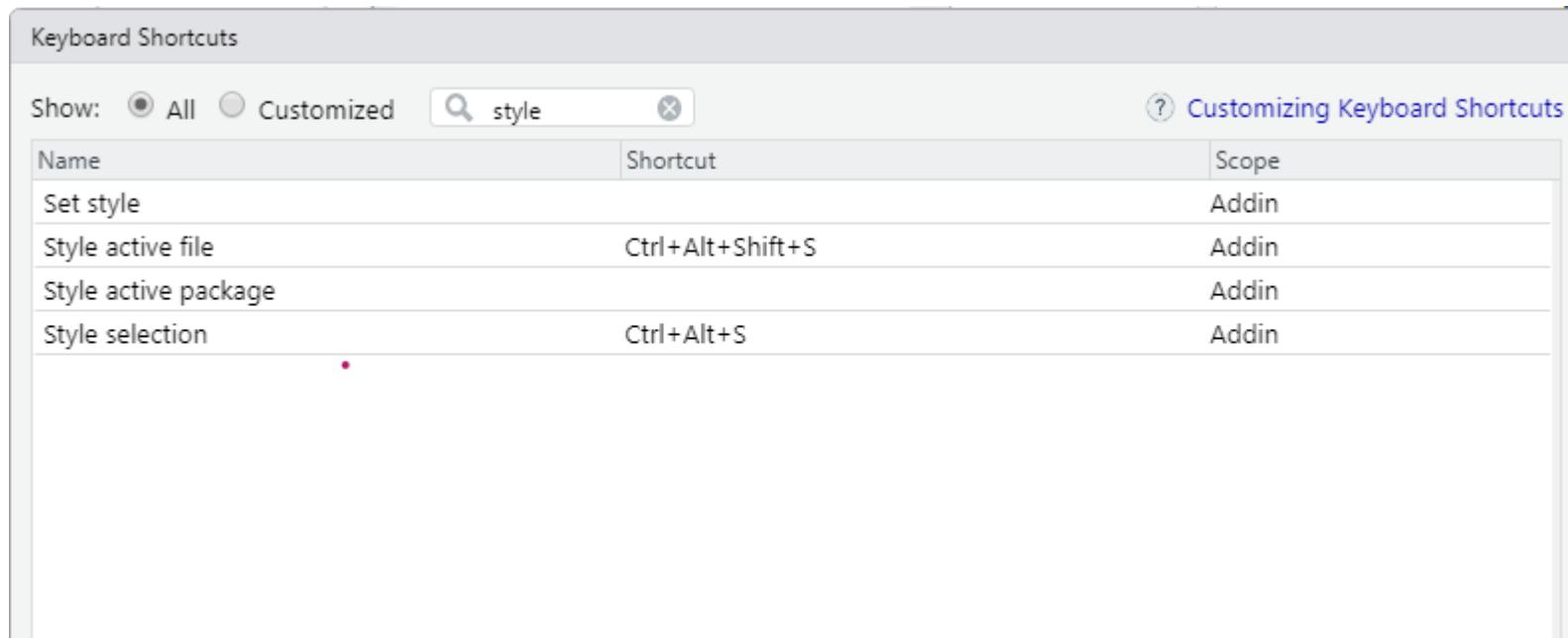
```
# install from CRAN  
install.packages("styler")
```

- 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

# 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

# Clean projects and workflows...

... increase

- Reproducibility 
- Reliability 
- Reusability 



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

# 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