

# Good practice R coding

Day 1 - Introduction to Data Analysis with R

Selina Baldauf

Freie Universität Berlin - Theoretical Ecology

September 18, 2024

# Chaotic projects and workflows ...

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



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?

# First things first

Project setup and structure

# Use R Studio projects

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

-  You already did that.

# 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](#))

# 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.

- 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 

-  `01preparedata.R`
-  `01firstscript.R`

Good examples 

-  `01_prepare-data.R`
-  `01_temperature-trend-analysis.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

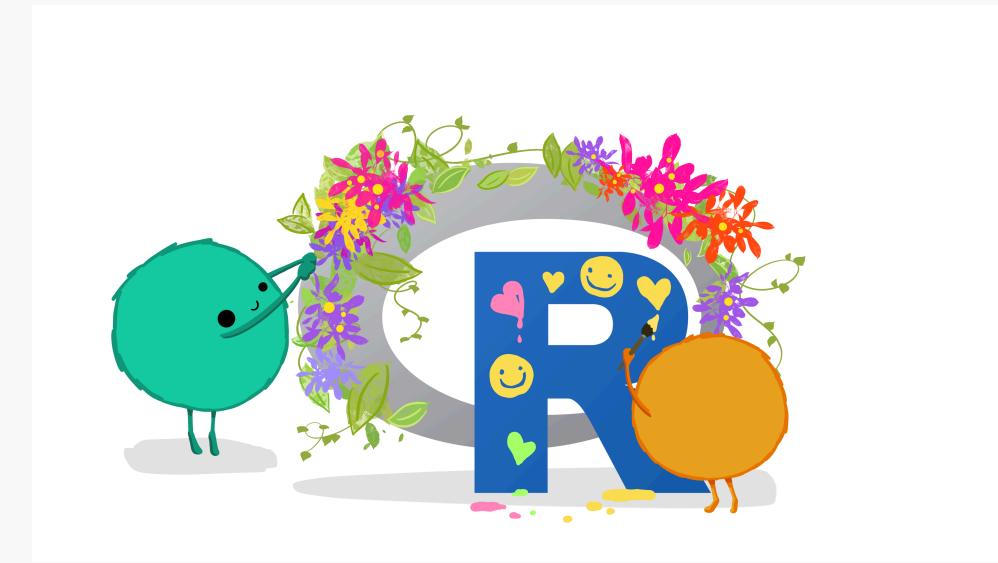
-  `01_prepare-data.R`
-  `02_lm-temperature-trend.R`

# Let's start coding

Good practice R coding

# Write beautiful code

- Try to write code that others (i.e. future you) can understand
  - Follow standards for readable and maintainable code
    - For R: [tidyverse style guide](#) defines code organization, syntax standards,
- ...



Artwork by [Allison 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

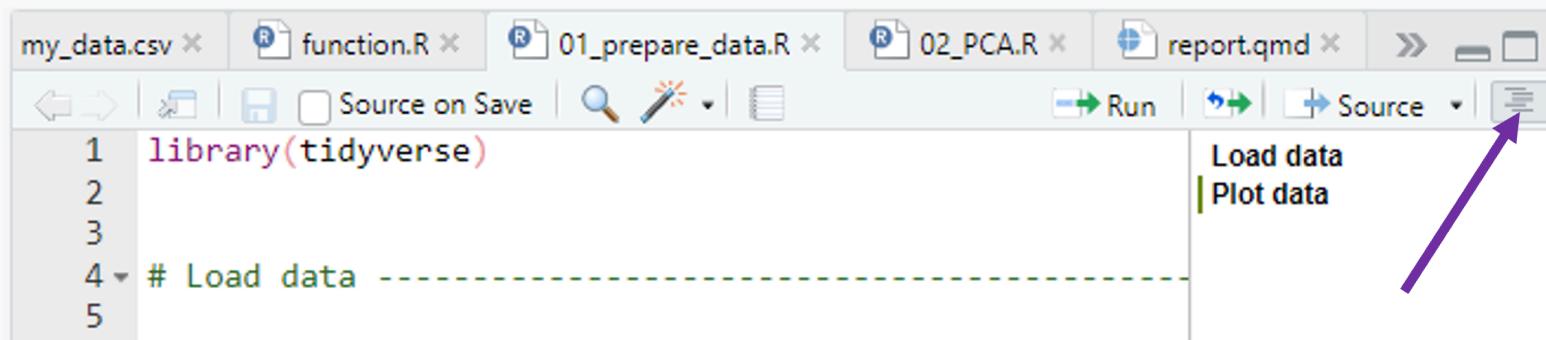
```
# This code replicates figure 2 from th  
# Baldauf et al. 2022 Journal of Ecolog  
# paper.  
# Authors: Selina Baldauf and Jane Doe  
  
library(tidyverse)  
library(vegan)  
  
# set defaults  
input_file <- "data/results.csv"  
  
# source files  
source("R/my_cool_function.R")  
  
# read input  
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 + R**
- Navigate sections in the file outline



# Coding style - Object names

- Variables 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 pipe (|>) followed by new line

```
# Good
iris |>
  summarize_if(is.numeric, mean, .by = Species) |>
  arrange(desc(Sepal.Length))

# Bad
iris|>summarize_if(is.numeric, mean, .by = Species)|>arrange(desc(Sepal.Length))
```

# 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 |> summarise(Sepal.Length = mean(Sepal.Length), Sepal.Width = mean(Sepal.Width))

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

# Coding style

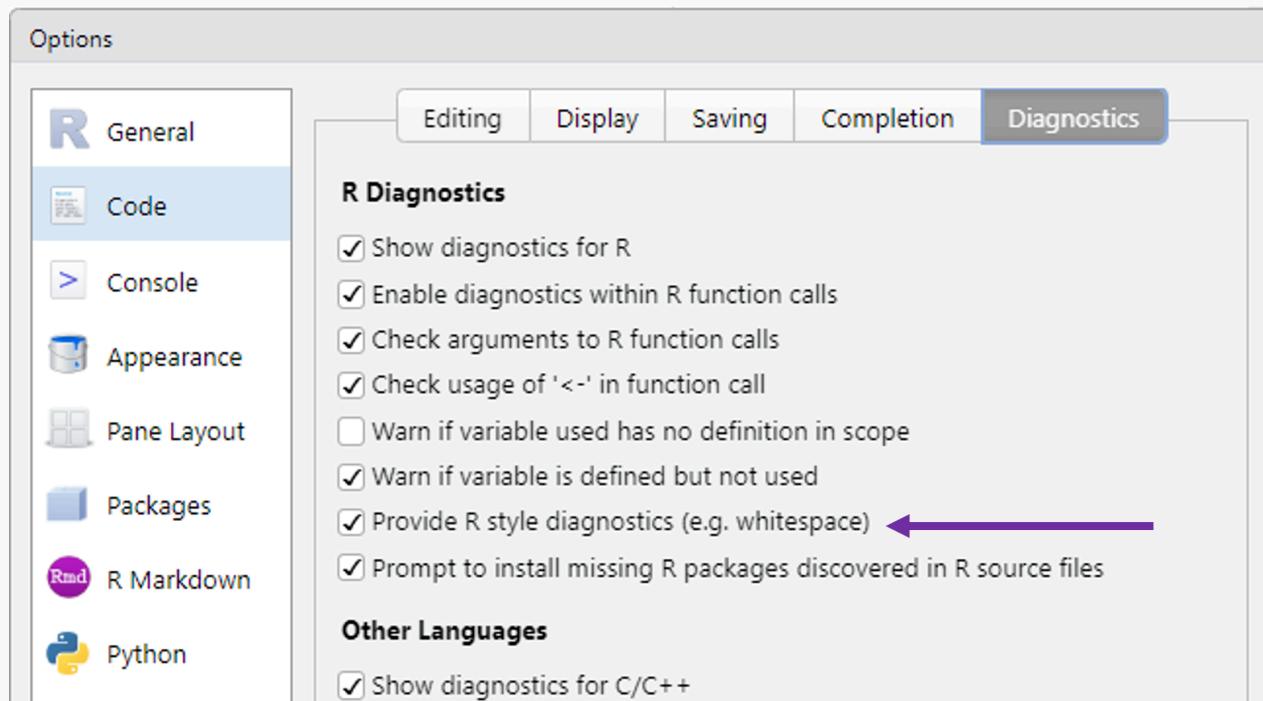
Do I really have to remember all of this?

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



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

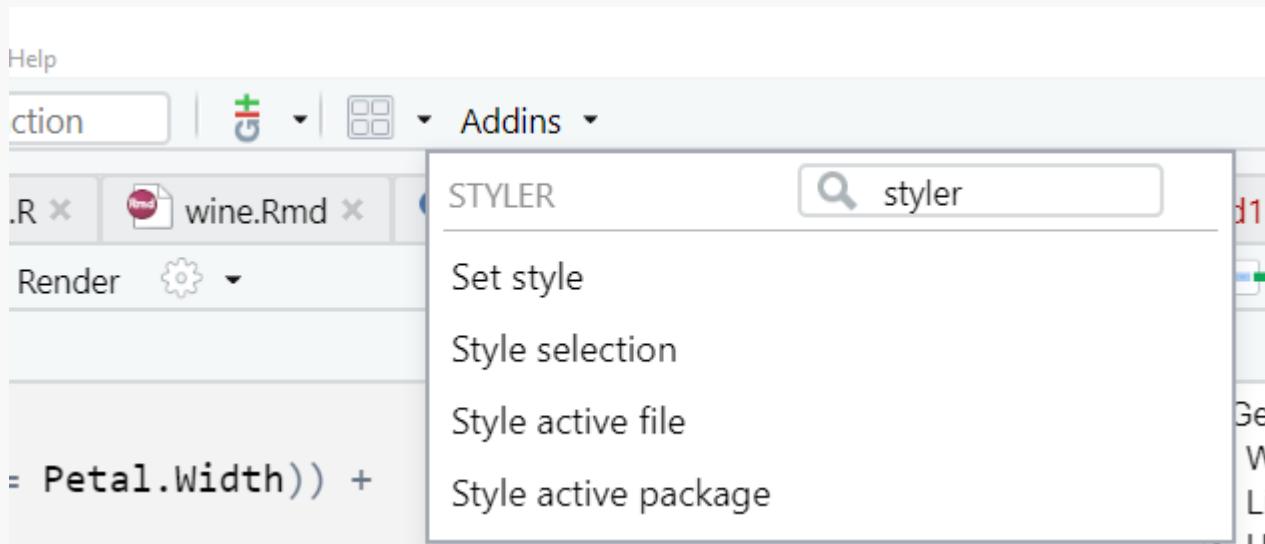
expected whitespace around '=' operator

# Coding style helpers - {styler}

The [styler package](#) automatically styles your files and projects according to the tidyverse style guide.

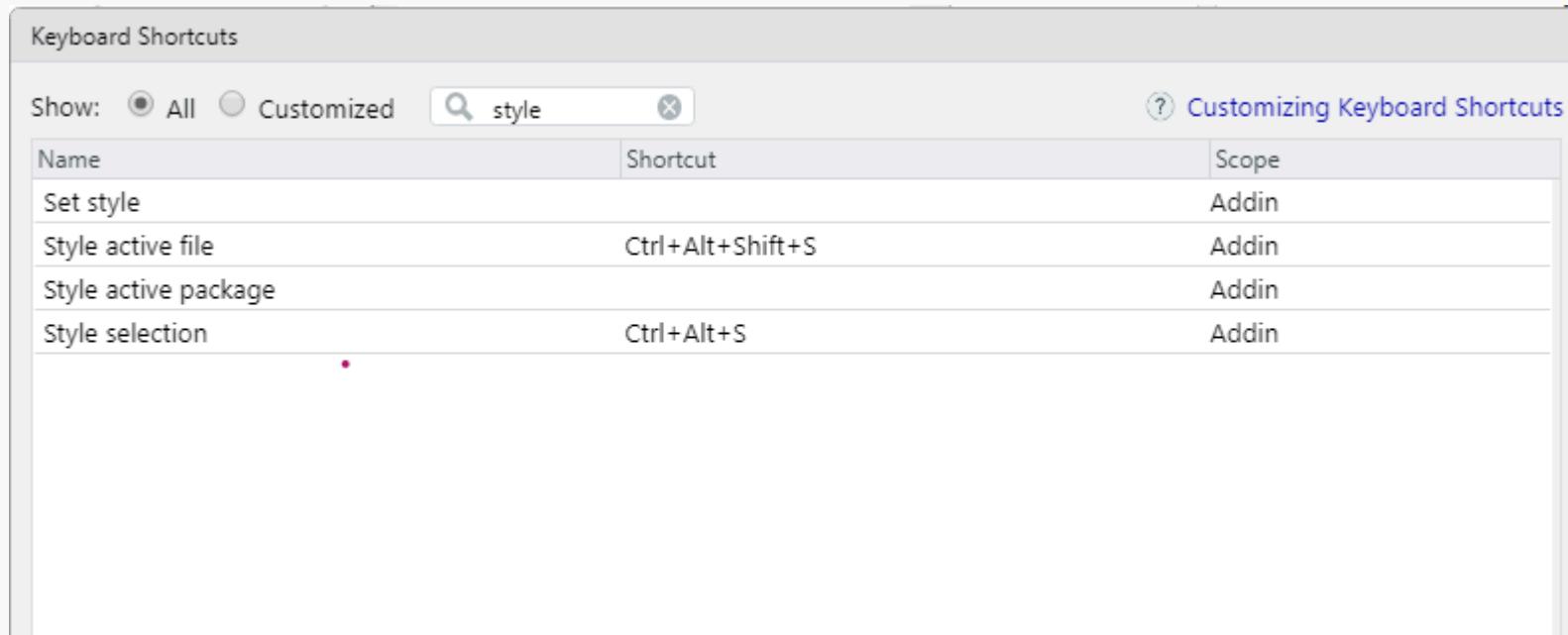
```
# install from CRAN  
install.packages("styler")
```

- 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



# Clean projects and workflows ...

... allow you and others to work productively.



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

