

# Import and Export Data with readr

Day 1 - Introduction to Data Analysis with R

Selina Baldauf

Freie Universität Berlin - Theoretical Ecology

October 12, 2023

# The tidyverse

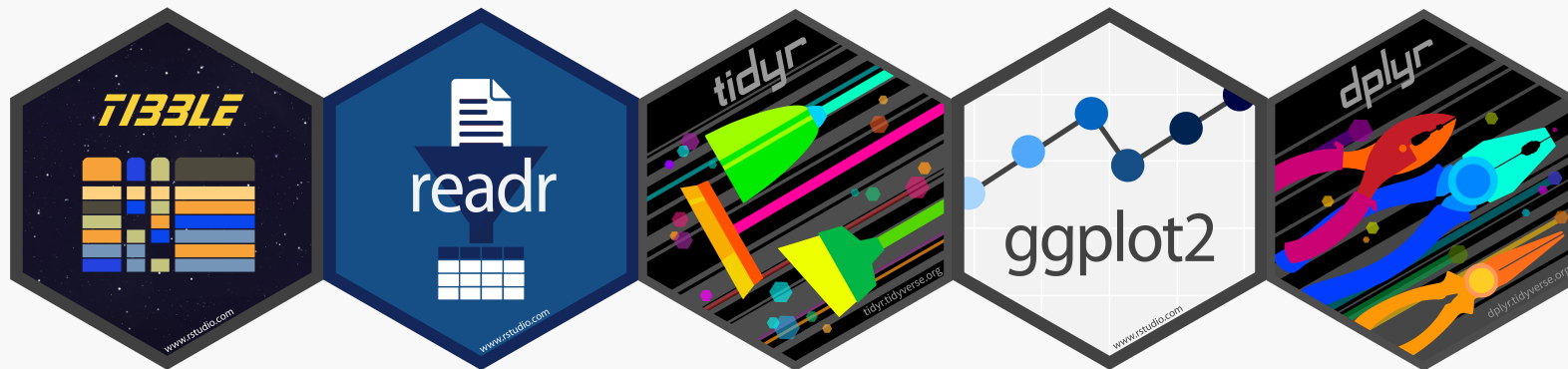


# The tidyverse

The tidyverse is an opinionated **collection of R packages** designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

([www.tidyverse.org](http://www.tidyverse.org))

These are the main packages from the tidyverse that we will use:



# Workflow data analysis

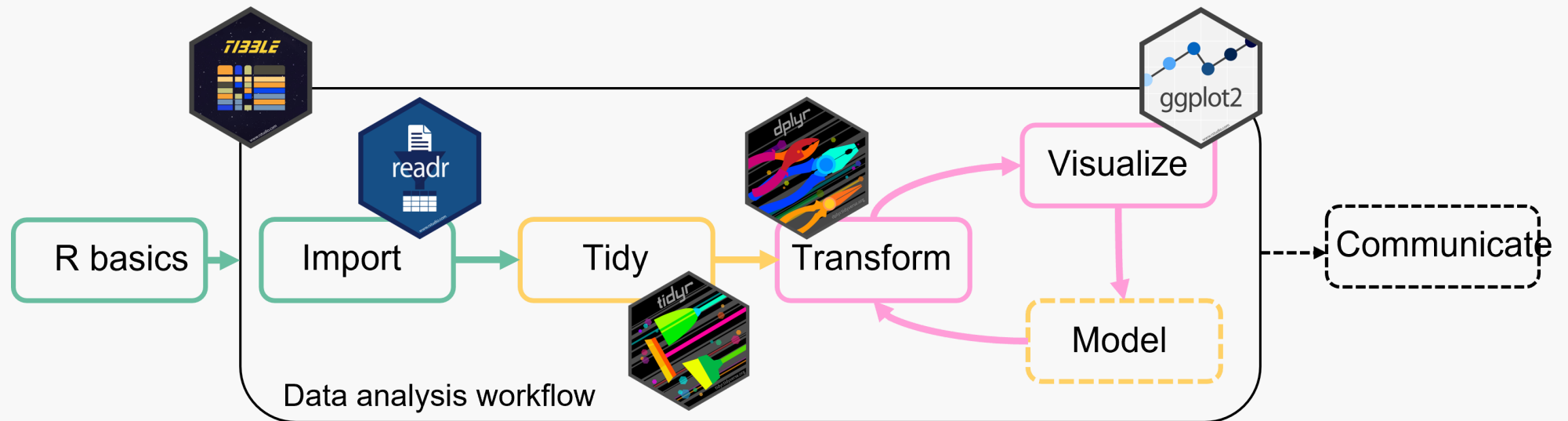


Image adapted from Wickham & Grolemund: [R for Data Science](#)

# The tidyverse

Install the tidyverse once with:

```
install.packages("tidyverse")
```

Then load and attach the packages at the beginning of your script:

```
library(tidyverse)
```

You can also install and load the tidyverse packages individually, but since we will use so many of them together, it's easier to load and attach them together.

# Import and export data with readr



# Readr

`readr` is a tidyverse package. To use it, you can load the tidyverse:

```
library(tidyverse) # or library(readr)
```

The most important functions are:

- `read_csv/write_csv` to read/write **comma delimited** files
- `read_tsv/write_tsv` to read/write **tab delimited** files
- `read_delim/write_delim` to read/write files with **any delimiter**

# Read files with `read_*()`

All `read_*` functions take a path to the data file as a first argument:

```
read_*(file = "path/to/your/file", ...)
```

Import files with a `readr` function fitting the delimiter of your file:

```
dat <- read_csv("data/your_data.csv") # comma delimiter  
dat <- read_tsv("data/your_data.txt") # tab delimiter
```

Use `read_delim` for a generic type of delimiter:

```
dat <- read_delim("data/your_data.txt", delim = "\t") # tab delimiter  
dat <- read_delim("data/your_data.txt", delim = "..xyz..") # ..xyz.. delimiter
```

All `read_*` functions return a `tibble`



## Read files with `read_*()`

The read functions provide several options to modify the reading of data.

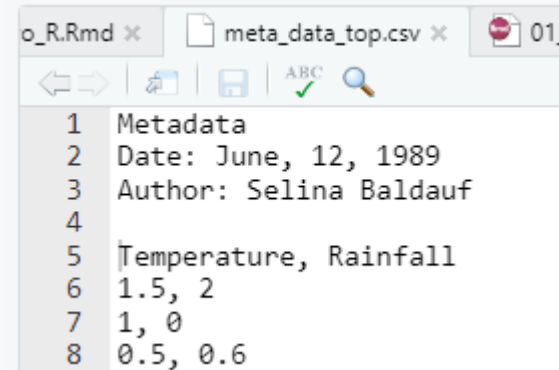
Have a look at `?read_delim` for all options.

Useful if your data is not a “perfect table”

# Read files with `read_*()`

Specify number of lines to skip reading with `skip`

- Useful if you have metadata on top of the file



The screenshot shows a text editor with a file named 'meta\_data\_top.csv'. The content of the file is as follows:

1	Metadata
2	Date: June, 12, 1989
3	Author: Selina Baldauf
4	
5	Temperature, Rainfall
6	1.5, 2
7	1, 0
8	0.5, 0.6

```
# without skipping first lines
read_csv(file = "data/meta_data_top.csv")
```

```
#> # A tibble: 6 × 1
#>   Metadata
#>   <chr>
#> 1 Date: June, 12, 1989
#> 2 Author: Selina Baldauf
#> 3 Temperature, Rainfall
#> 4 1.5, 2
#> 5 1, 0
#> 6 0.5, 0.6
```

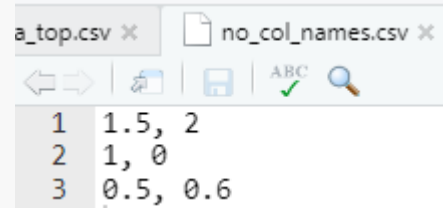
```
# skip meta data lines
read_csv(
  file = "data/meta_data_top.csv",
  skip = 4
)
```

```
#> # A tibble: 3 × 2
#>   Temperature Rainfall
#>   <dbl>      <dbl>
#> 1     1.5         2
#> 2     1         0
#> 3     0.5        0.6
```

# Read files with `read_*()`

Specify whether the data has a header column or not with `col_names`

- Useful if you don't have column names or you want to change them



1	1.5, 2
2	1, 0
3	0.5, 0.6

```
# First line expected to be column name  
read_csv(file = "data/no_col_names.csv")
```

```
#> # A tibble: 2 × 2  
#>   `1.5`   `2`  
#>   <dbl> <dbl>  
#> 1     1     0  
#> 2   0.5   0.6
```

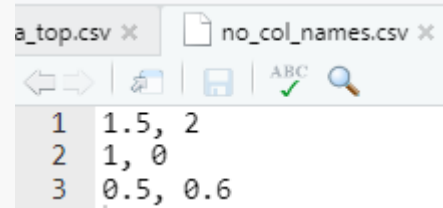
```
# Default column names are given  
read_csv(  
  file = "data/no_col_names.csv",  
  col_names = FALSE  
)
```

```
#> # A tibble: 3 × 2  
#>       X1     X2  
#>   <dbl> <dbl>  
#> 1   1.5     2  
#> 2     1     0  
#> 3   0.5   0.6
```

# Read files with `read_*()`

Specify whether the data has a header column or not with `col_names`

- Useful if you don't have column names or you want to change them



1	1.5, 2
2	1, 0
3	0.5, 0.6

```
# First line expected to be column name  
read_csv(file = "data/no_col_names.csv")
```

```
#> # A tibble: 2 × 2  
#>   `1.5`   `2`  
#>   <dbl> <dbl>  
#> 1     1     0  
#> 2   0.5   0.6
```

```
# Specify custom column names  
read_csv(  
  file = "data/no_col_names.csv",  
  col_names = c("Temperature", "Rainfall")  
)
```

```
#> # A tibble: 3 × 2  
#>   Temperature Rainfall  
#>   <dbl>      <dbl>  
#> 1     1.5         2  
#> 2     1         0  
#> 3     0.5        0.6
```

## Write files with `write_*()`

Every `read_*` has a corresponding `write_*` function to export data from R.

Write data from R e.g.

- To share transformed or summarized data
- Summarize complex raw data and continue working with summarized data
- ...

# Write files with `write_*` ()

All `write_*` functions take the data to write as the first and the file to write to as the second argument:

```
write_*(x = dat, file = "path/to/save/file.*", ...)
```

```
write_csv(dat, file = "data-clean/your_data.csv") # comma delimiter
write_tsv(dat, file = "data-clean/your_data.txt") # tab delimiter
```

Use `write_delim` for a generic type of delimiter:

```
write_delim(dat, file = "data-clean/your_data.txt", delim = "\t") # tab delimiter
write_delim(dat, file = "data-clean/your_data.txt", delim = "..xyz..") # ..xyz..
```

# Import excel files



# Readxl

The `readxl` package is part of the tidyverse, but you need to load it explicitly

```
library(readxl)
```

Use the `read_excel` function to read an excel file:

```
dat <- read_excel(path = "data/your_data.xlsx")
```

By default, this reads the first sheet. You can read other sheets with

```
dat <- read_excel(path = "data/your_data.xlsx", sheet = "sheetName") # via sheet  
dat <- read_excel(path = "data/your_data.xlsx", sheet = 2) # via sheet number
```

- `read_excel` also has other functionality, like skipping rows etc.
- Check out the [package documentation](#) for more functionality



# Readxl

A little warning:

- Reading from a text file (.txt or .csv) is more reliable
- Be careful with complicated excel sheets with formulas etc.
- Always double check the data that you imported, e.g. by using the `summary` function and checking if the number of rows etc. is correct

# Absolute vs. relative paths in R

## Absolute paths

`C:/Users/Selina/folder1/folder2/data/file_to_read.csv`

## Relative paths

`data/file_to_read.csv`

- Relative paths are interpreted relative to the **working directory**
- Check out where your working directory is with `getwd()`
- In RStudio projects, the **working directory** is always the project root

# Absolute vs. relative paths

Working with R and RStudio, the best way is to:

- **Organize your work in an RStudio project**
  - The project root is automatically the working directory
  - All your files (also your data) are in one place
- **Use paths relative to the project root**

## Why?

- No need to change the working directory
- Portable paths: will also work on other machines that copied the project
- Makes the code more readable
- Less error prone

# Guidelines for data sets in

# Data format

Follow these guidelines to make data import to R easier and less frustrating

- In general: prefer machine-readable file formats (`.csv`, `.txt` instead of `.xlsx`)

Save an Excel spreadsheet as csv

1. **File -> Save As** and select comma separated from the drop down menu
2. **File -> Export**

# Data format

Follow these guidelines to make data import to R easier and less frustrating

- In general: prefer machine-readable file formats (`.csv`, `.txt` instead of `.xlsx`)
- No white space in column headers
  - Use a character as separator, e.g. `species_name` instead of `species name`
  - If this is unpractical, have a look at the function `janitor::clean_names()` from the `janitor` package
- No special characters in column headers (ä,, ß, é, ê, %, °C, μ ...)
- Use `.` as a decimal separator (not `,`)

# Paths and file names

- Avoid white space in paths and file names
  - `data-raw/my_data.csv` instead of `data raw/my data.csv`
- Avoid special characters in paths

# Now you

Task (20 min)

Read and write data files

Find the task description [here](#)



