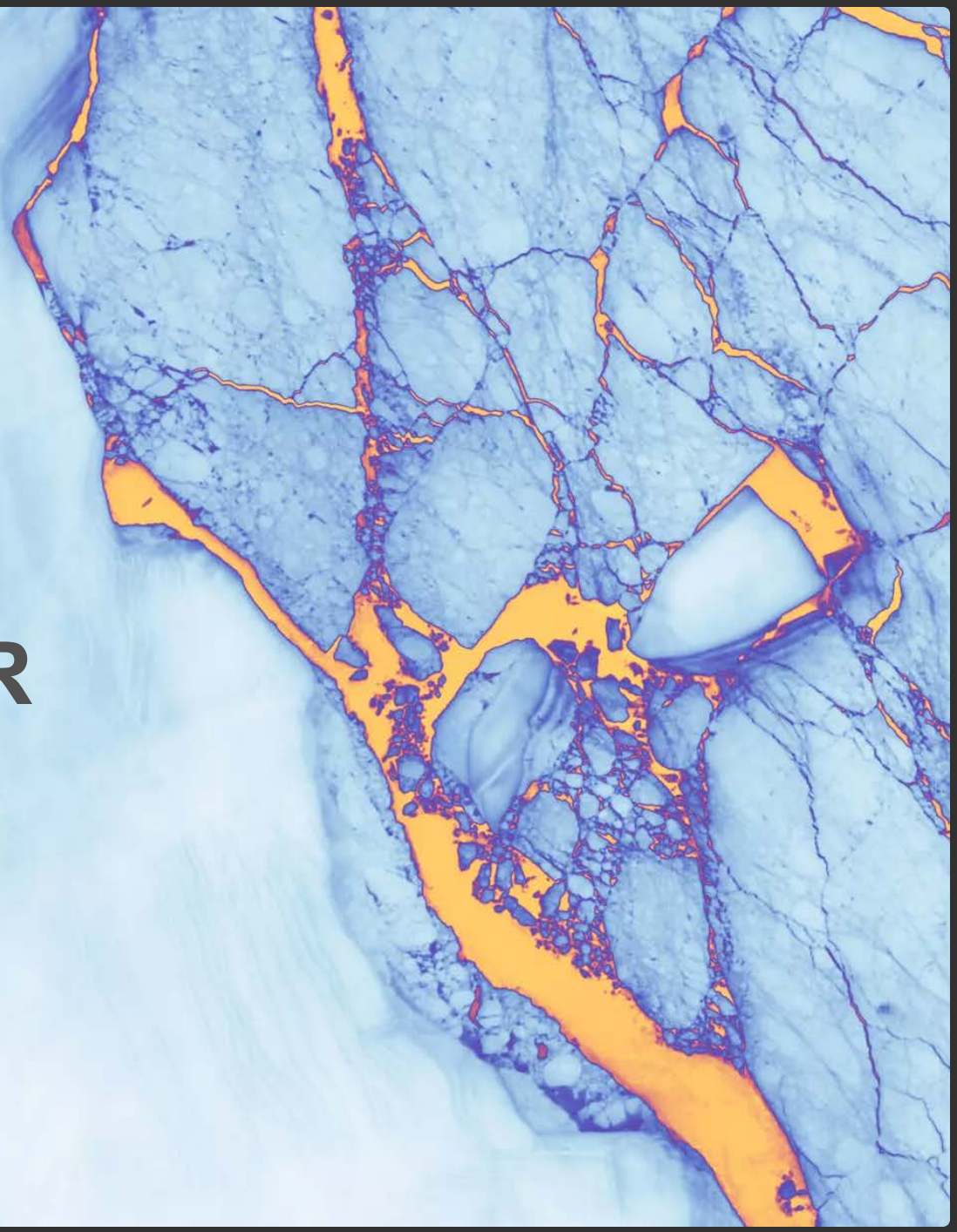
Marine Data Science

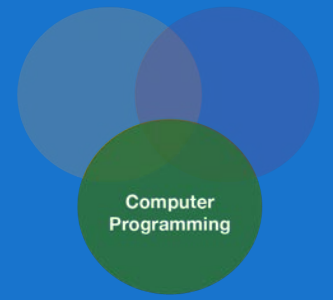
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG



Data Analysis with R

5 - Data wrangling - 1.Import

Saskia A. Otto
Postdoctoral Researcher



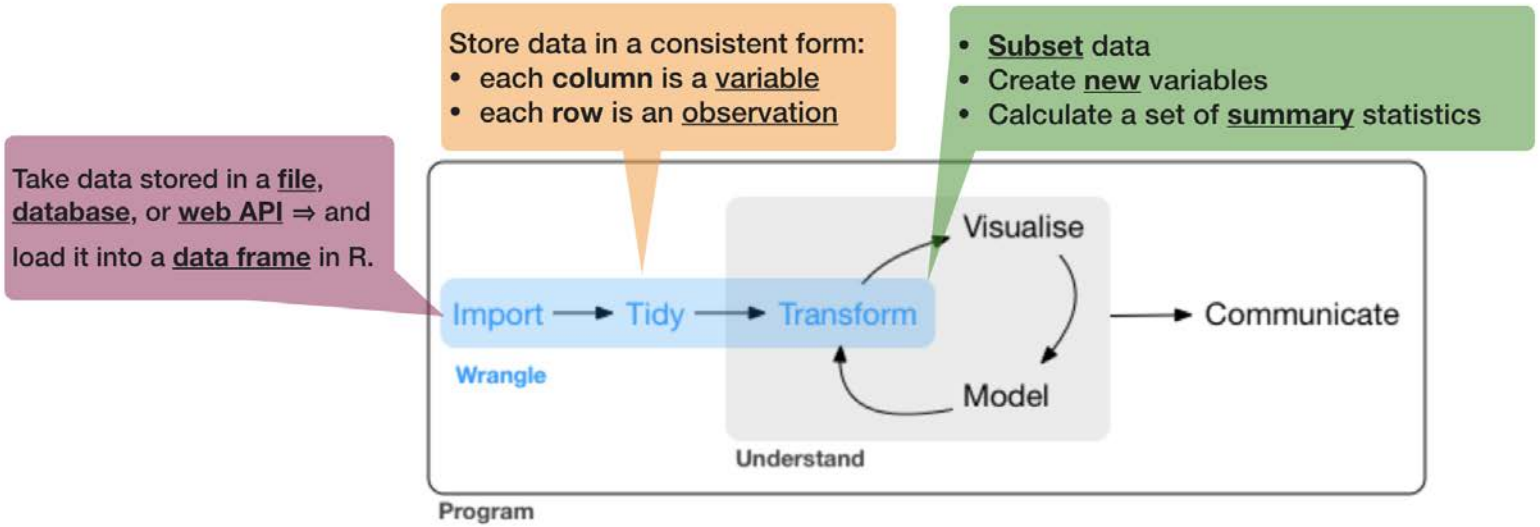
Data wrangling with tidyverse

Data wrangling is

- a concept introduced by **Hadley Wickam**
- the art of **getting your data into R in a useful form** for visualisation and modelling
- composed of **three** main parts:

Data wrangling is

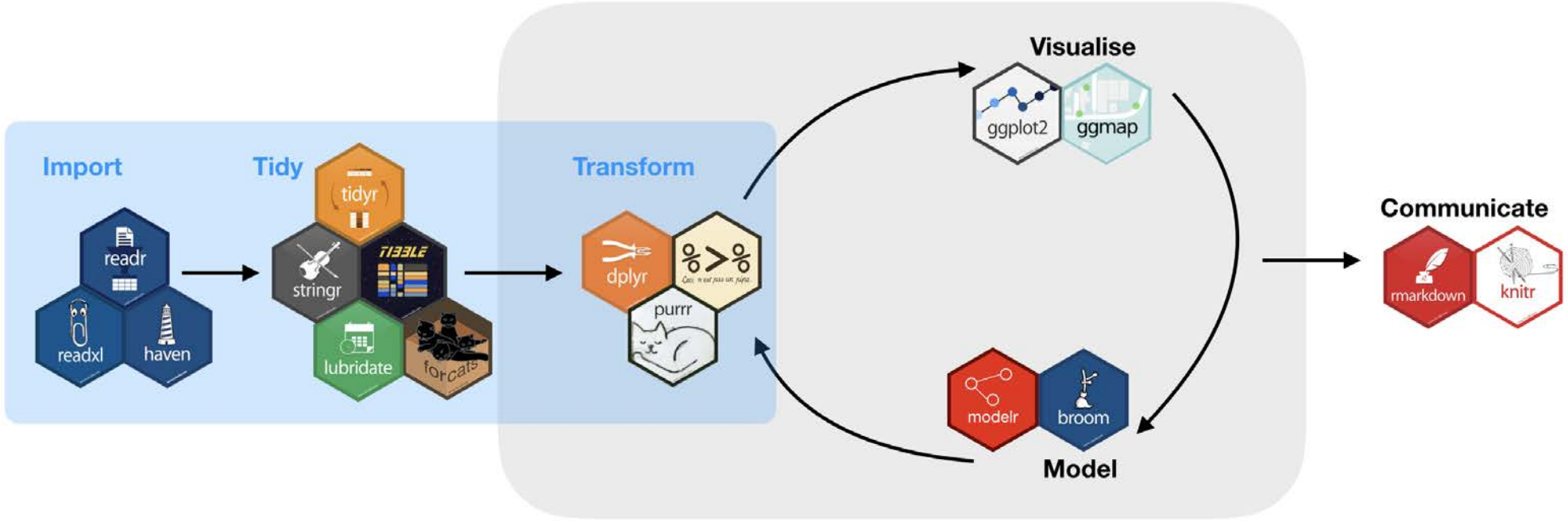
- a concept introduced by **Hadley Wickam**
- the art of **getting your data into R in a useful form** for visualisation and modelling
- composed of **three** main parts:



source of flowchart: R for Data Science by Wickam & Grolemund, 2017 (licensed under CC-BY-NC-ND 3.0 US)

Tidy (uni)verse

Is a collection of **R packages** that share **common philosophies** and are designed to work together:



Tidy (uni)verse

You will get to know during the course

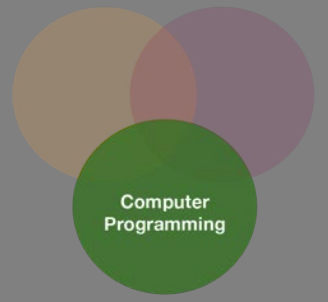
- **readr**: reads rectangular data (like 'csv', 'tsv', and 'fwf') into R
- **tibble**: modern re-imagining of data frames
- **tidy**: re-arranges data to make it "tidy"
- **dplyr**: provides functions for data manipulation
- **stringr**: provides wrapper functions for common string operations
- **lubridate**: handles dates/times
- **ggplot2**: a plotting system for R, based on the grammar of graphics
- **purrr**: functional programming toolkit
- **modelr**: wraps around base R's modelling functions to make them work naturally in a pipe

Why tidyverse?

- **Consistency**
  - e.g. all *stringr* functions take a string as first argument
  - e.g. most functions take a data frame as first argument (piping)
- Tidy data imposes **good practices**
- **Synergies** between different packages/tools
- Implements **simple solutions** to common problems
- **Smarter default** settings
  - e.g. `utils::write.csv(row.names = FALSE)`, `readr::write_csv()`
- Runs **fast** (most functions implemented with Rcpp)
- More and more packages implement the tidyverse concept

The easiest way to get these packages is to install the whole tidyverse:

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

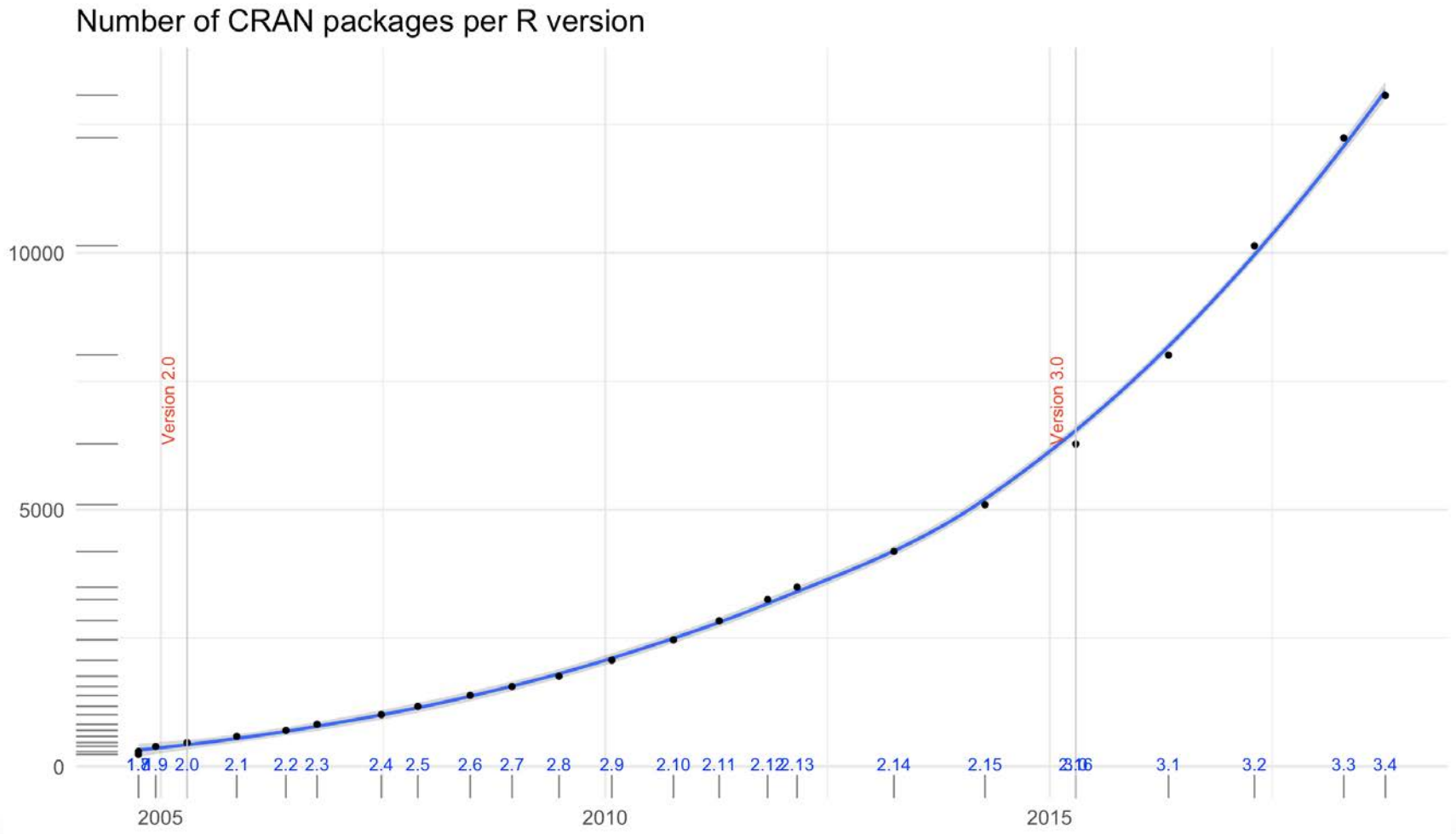


But wait ... a little detour on packages

Packages

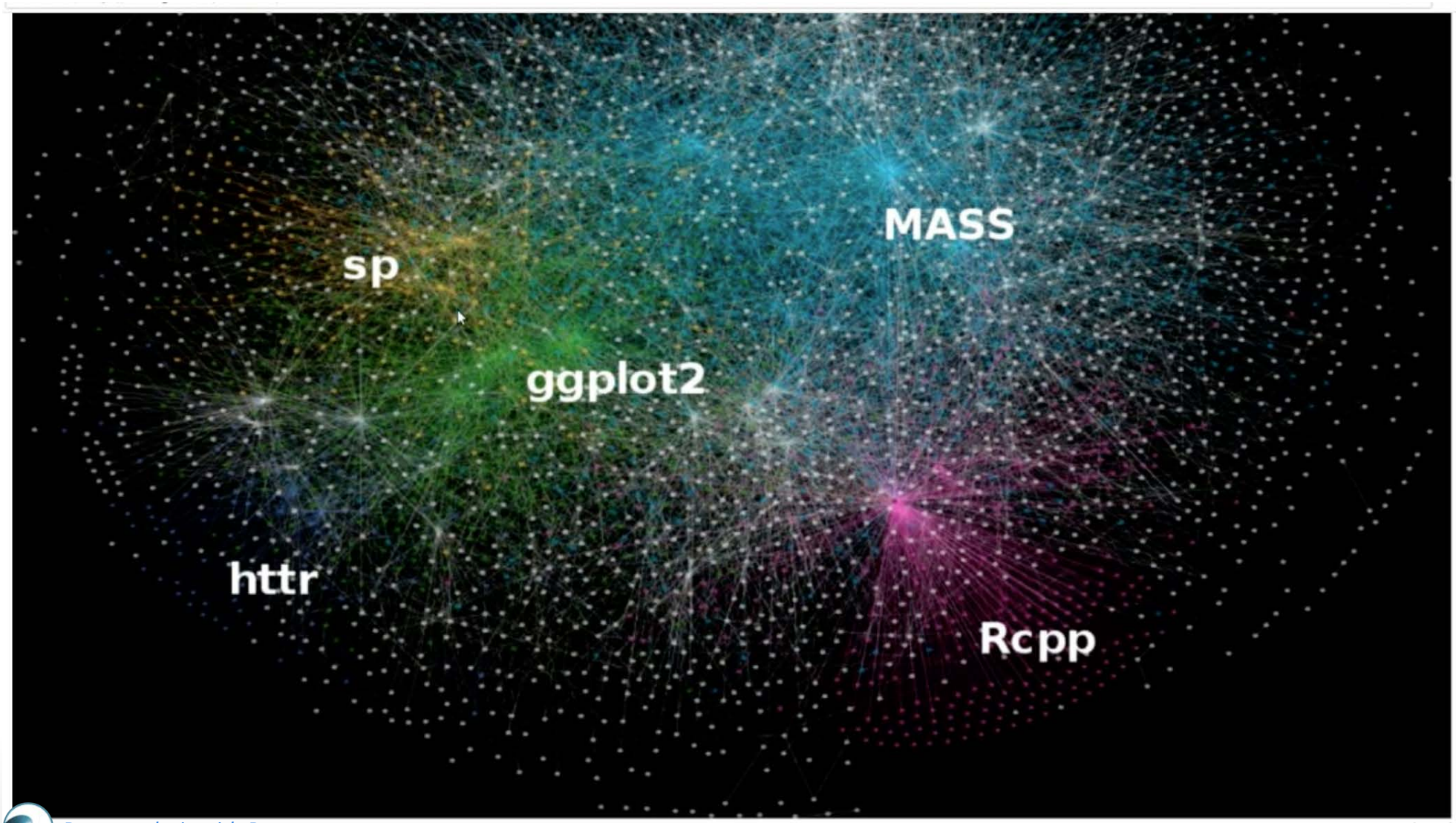
- are written by the **R community**
- are a collection of
  - reusable R **functions**,
  - the **documentation** that describes how to use them,
  - and often sample **data**
- the official **CRAN package repository** features **11782 (!)** available packages at the moment (Nov 11, 2017)

Exponential increase



The chart was created using this code from Andrie de Vries (on Oct 12th, 2018).

Key packages and dependencies



Package installations (ONCE)

The 'approved' versions can be downloaded from CRAN using the **function**

```
install.packages("package_name")
```

or via R Studio:

Package loading (EVERY SESSION)

You load a package using the functions `library()` or `require()`. R checks whether this package has been installed and if it doesn't exist, you'll get an error message. The main difference between both functions is what happens if a package is not found. For consistency, simply stick to one function:

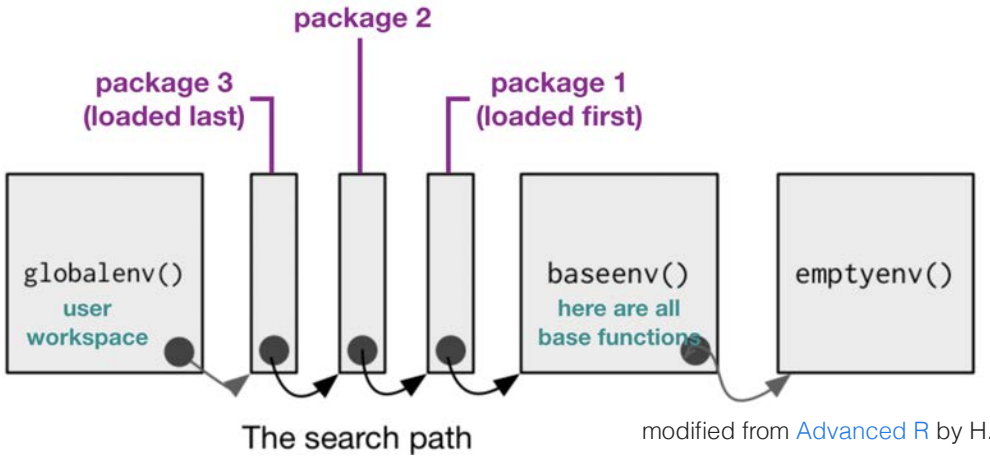
```
library(any_package) # library("any_package") would also work
```

```
## Error in library(any_package): there is no package called 'any_package'
```

```
require(any_package) # require("any_package")
```

Package loading (cont)

If you load a specific package you add it to the search paths:



modified from Advanced R by H. Wickam, 2014

- To call a function, R first has to find it. R does this by **first** looking in the **global** environment.
- If R doesn't find it there, it looks in the **search path**, the list of all the packages you have attached.
- If packages have functions with the same name, R uses the function from the package, which was **loaded last**.

Package loading (cont)

You can see the search path and package list by running `search()`.

```
search()
```

```
##  [1] ".GlobalEnv"        "tools:rstudio"     "package:stats"
##  [4] "package:graphics"  "package:grDevices" "package:utils"
##  [7] "package:datasets"  "package:methods"   "Autoloads"
## [10] "package:base"
```

After loading

```
library(tidyverse)
```

you see that 8 additional tidyverse core packages are loaded. You also see a **conflict of function names** (`filter()` and `lag()` exist in 2 packages)!

Lets look at the search path again:

```
search()
```

```
##  [1] ".GlobalEnv"        "package:forcats"   "package:stringr"
##  [4] "package:dplyr"     "package:purrr"     "package:readr"
##  [7] "package:tidyr"     "package:tibble"    "package:ggplot2"
## [10] "package:tidyverse" "tools:rstudio"     "package:stats"
## [13] "package:graphics"  "package:grDevices" "package:utils"
## [16] "package:datasets"  "package:methods"   "Autoloads"
## [19] "package:base"
```

You now see the 9 packages added to the search path (right after the global environment).

Your turn...

Quiz 1: Function conflicts

From which packages will R use the functions `filter()` and `lag()`?

- dplyr
- stats

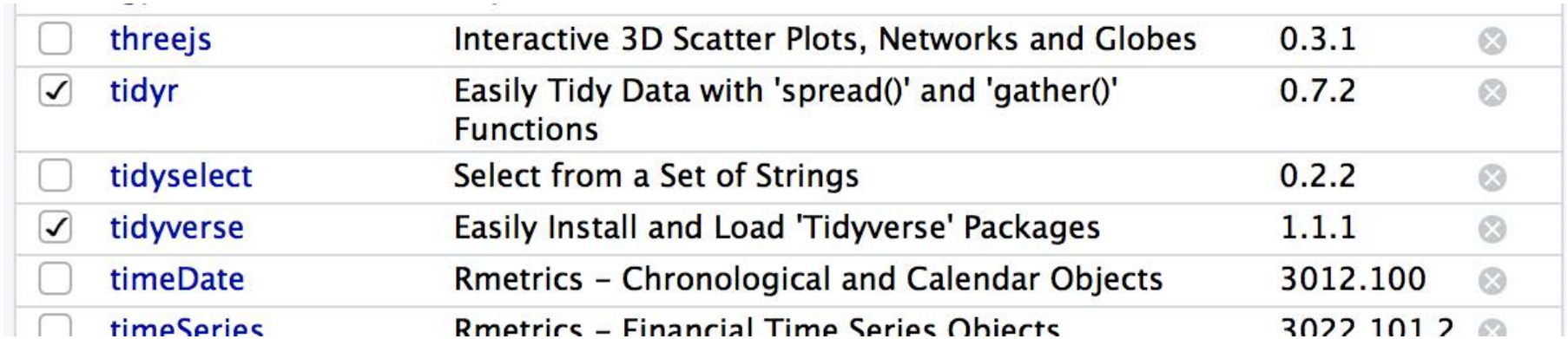
Submit | Show Hint | Show Answer | Clear

How to unload packages?

You remove a package from the search path using the function

```
detach(packagename)
```

or by unchecking the box next to the package name in the 'Packages' pane.



Information on packages

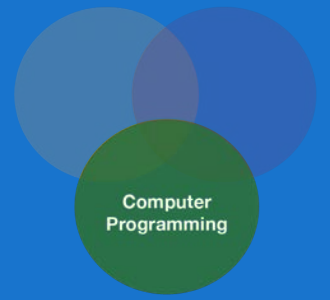
- If you run `?packagename` (e.g. `?tidyverse`) you get some more information of what the package does and sometimes lists of functions available in this package or weblinks for further information.
- More recent packages have also so-called "vignettes", which provide even more informations than the help documentation. You can read the vignette by calling `vignette("packagename")`.
- Sometimes, a package provides several vignettes. To get an overview call the function `browseVignettes("packagename")`.

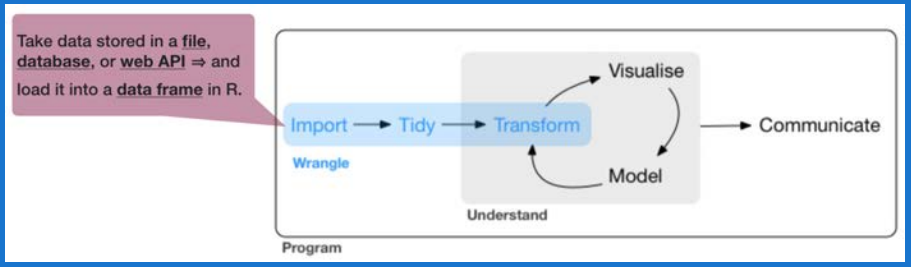
```
vignettes("dpylr")
browseVignettes("dpylr")
```

Your turn...

Explore some of the tidyverse packages (not 'tidyverse' itself) or any other one installed.

1. Load and unload 3 packages of your choice.
2. Look into the help documentation and vignettes of these 3 packages.
    - What are they for?
    - Who is the author?
3. Identify at least 3 functions that each of the 3 packages provides.





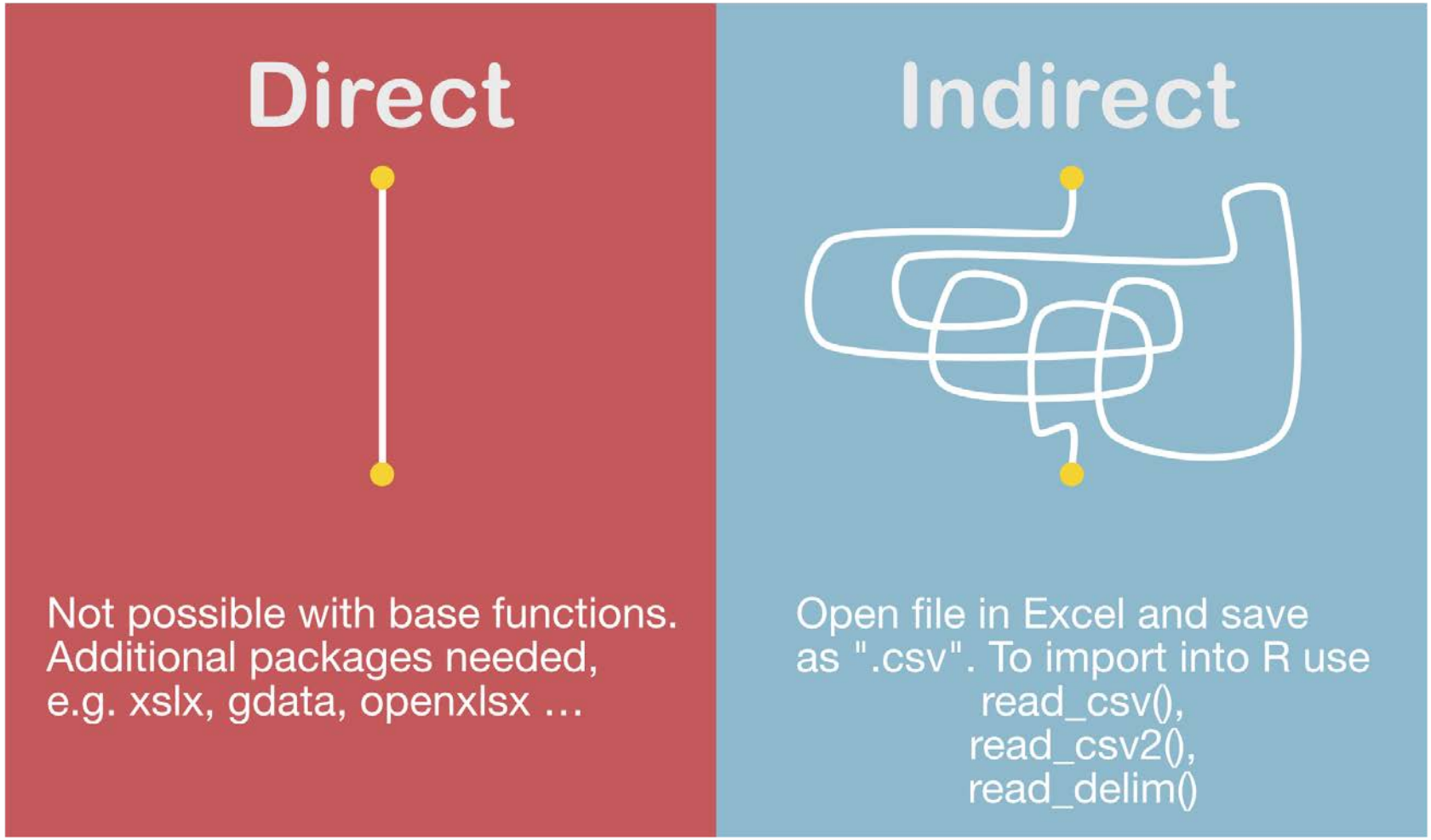
Back to data wrangling: 1. Import

Data sources

- Excel files (.xls / .xlsx)
- Comma seperated values (.csv) --> most common files
- Text files (.txt)
- NetCDF (Network Common Data Form)
- Relational data bases (MySQL, PostgreSQL, etc.)
- URLs
- and many more ...

Mostly you will have flat files (with no **internal hierarchy** and interrelationships as in databases) to load into R.

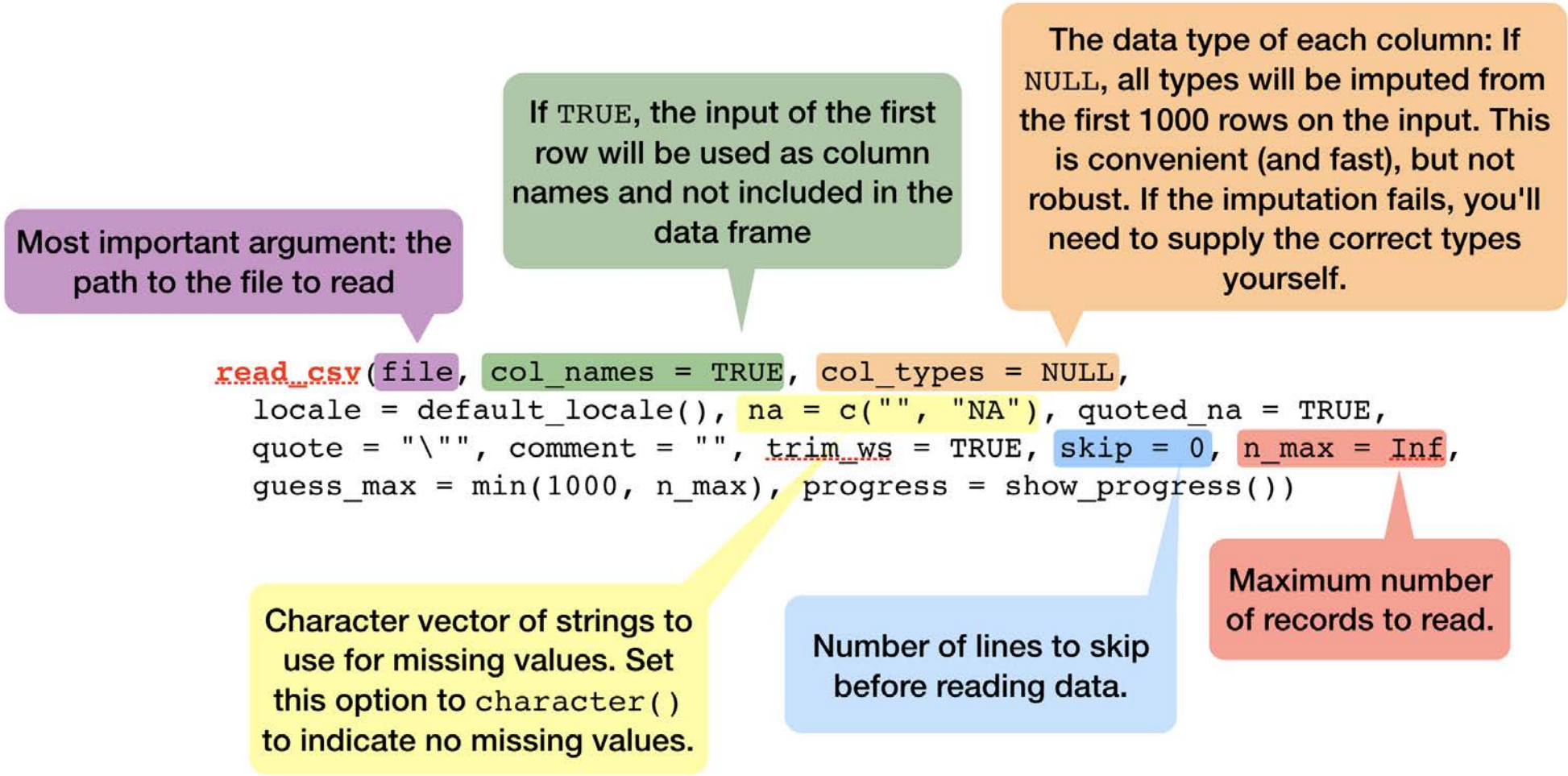
Importing data from Excel



Import functions in 'readr'

- Most of readr's functions are concerned with turning flat files into data frames:
  - `read_delim()`: reads in files with **any delimiter**.
  - `read_csv()`: **comma** delimited files (.csv)
  - `read_csv2()`: **semicolon** separated files (.csv) - common when comma used as decimal mark
  - `read_tsv()`: **tab** delimited files (.txt files)
  - and others: `read_table()`, `read_fwf()`, `read_log()`

- All these functions have a similar or the same syntax:



Some demonstrations of read_csv using inline csv files

Inline csv files are useful for experimenting and for creating reproducible examples:

```
read_csv("a,b,c
1,2,3
4,5,6")
```

```
## # A tibble: 2 x 3
##       a     b     c
##   <int> <int> <int>
## 1     1     2     3
## 2     4     5     6
```

Tweaking your import - skipping lines

You can skip the first n lines of metadata at the top of the file using **skip = n**:

```
read_csv("The first line of metadata
  The second line of metadata
  x,y,z
  1,2,3", skip = 2)
```

```
## # A tibble: 1 x 3
##       x     y     z
##   <int> <int> <int>
## 1     1     2     3
```

Or use **comment = "#"** to drop all lines that start with (e.g.) #.

```
read_csv("# A comment to skip
  x,y,z
  1,2,3", comment = "#")
```

```
## # A tibble: 1 x 3
##       x     y     z
##   <int> <int> <int>
## 1     1     2     3
```

Tweaking your import - column names

If you don't have column names set **col_names = FALSE**; R labels them sequentially from X1 to Xn:

```
read_csv("1,2,3
  4,5,6", col_names = FALSE)
```

```
## # A tibble: 2 x 3
##       X1    X2    X3
##    <int> <int> <int>
## 1     1     2     3
## 2     4     5     6
```

You can also pass a **character vector** to col_names:

```
read_csv("1,2,3
  4,5,6", col_names = c("x", "y", "z"))
```

```
## # A tibble: 2 x 3
##       x     y     z
##    <int> <int> <int>
## 1     1     2     3
## 2     4     5     6
```

Tweaking your import - Specify column types

readr functions guess the type of each column and convert types when appropriate (but will NOT convert strings to factors automatically). If you want to specify other types use a **col_function** in the col_types argument to guide parsing.

```
read_csv("your_file.csv", col_types = cols(
  a = col_integer(),
  b = col_character(),
  c = col_logical() )
)
```

Tweaking your import - NAs

The argument `na` specifies the value (or values) that are used to represent missing values in your file (-999 or -9999 is a typical place holder for missing values).

```
read_csv("a,b,c
  1,2,.", na = ".")
```

```
## # A tibble: 1 x 3
##       a     b c
##   <int> <int> <chr>
## 1     1     2 <NA>
```

```
read_csv("a,b,c
  1,-9999,2", na = "-9999")
```

```
## # A tibble: 1 x 3
##       a b         c
##   <int> <chr> <int>
## 1     1 <NA>      2
```

So what are these **tibbles**?

- The **'tibble'** package provides a **'tbl_df'** class (the 'tibble') that provides stricter checking and better formatting than the traditional data frame.
- **Major differences** to a data frame:
  - never changes the type of the inputs (e.g. it never converts strings to factors!)
  - never creates row names
  - never changes the names of variables
  - non-syntactic column names possible (e.g. names can contain unusual characters like a space) --> BUT DONT GO THAT ROAD!
  - tibbles generate a warning if the column you are trying to access does not exist
  - printing and subsetting differs

So what are these **tibbles**? (cont)

- Functions for data frames will work also for tibbles.
- All tidyverse packages generate tibbles automatically
- To learn more check `vignette("tibble")`.

Creating tibbles

- Automatically created when importing data with *readr*
- Convert an existing data frame with `tibble::as_tibble(your_dataframe)` (NOTE: tidyverse uses underscores, not points)

Creating tibbles

- Automatically created when importing data with *readr*
- Convert an existing data frame with `tibble::as_tibble(your_dataframe)` (NOTE: tidyverse uses underscores, not points)

```
iris_tbl <- as_tibble(iris)
# Compare the difference:
class(iris)
```

```
## [1] "data.frame"
```

```
class(iris_tbl)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

As you see, **iris_tbl** inherits still the `data.frame` class, but has in addition also the `tbl_df` class.

Creating tibbles (cont)

- Or you can create a new tibble from individual vectors with `tibble()`

```
tibble(x = 1:5, y = 1, z = x ^ 2 + y)
```

```
## # A tibble: 5 x 3
##       x     y     z
##   <int> <dbl> <dbl>
## 1     1     1     2
## 2     2     1     5
## 3     3     1    10
## 4     4     1    17
## 5     5     1    26
```

Inputs of shorter length are automatically recycled!

Printing tibble

- each column reports its **type**
- only the first **10 rows** and all columns that **fit on screen** are shown --> much easier to work with large data
- if you want to change the number of rows (**n**) and columns (**width**) use `print()` and change the arguments:

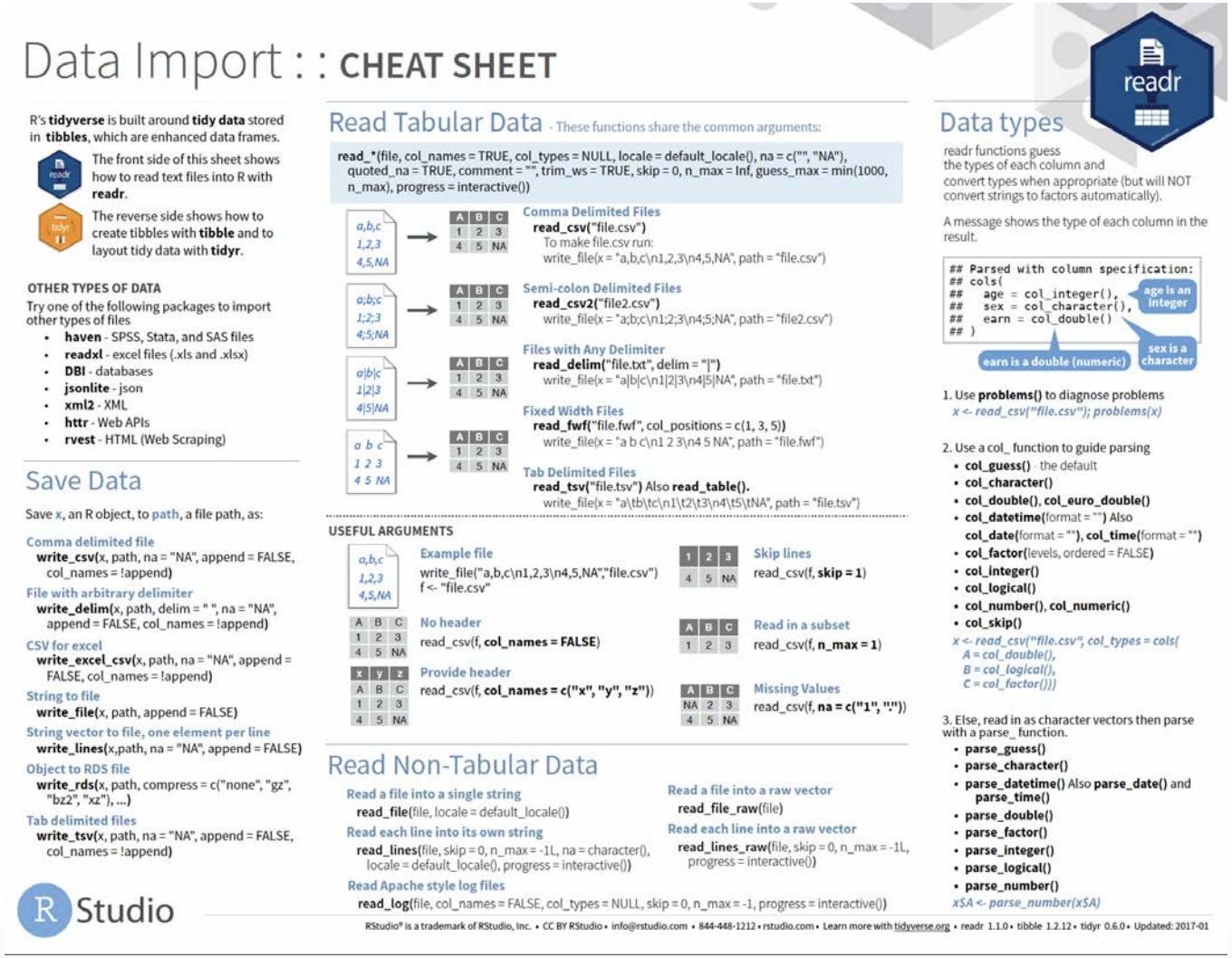
Printing tibble

- each column reports its **type**
- only the first **10 rows** and all columns that **fit on screen** are shown --> much easier to work with large data
- if you want to change the number of rows (**n**) and columns (**width**) use `print()` and change the arguments:

```
print(iris_tbl, n = 2, width = Inf)  # = Inf shows all columns
```

```
## # A tibble: 150 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##          <dbl>       <dbl>        <dbl>       <dbl> <fct>
## 1          5.1         3.5          1.4         0.2 setosa
## 2          4.9         3            1.4         0.2 setosa
## # ... with 148 more rows
```

Overview of more functions:



Data Import : : CHEAT SHEET

R's **tidyverse** is built around **tidy data** stored in **tibbles**, which are enhanced data frames.

The front side of this sheet shows how to read text files into R with **readr**.

The reverse side shows how to create tibbles with **tibble** and to layout tidy data with **tidyr**.

OTHER TYPES OF DATA

Try one of the following packages to import other types of files

- **haven** - SPSS, Stata, and SAS files
- **readxl** - excel files (.xls and .xlsx)
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

Save Data

Save x, an R object, to path, a file path, as:

Comma delimited file
`write_csv(x, path, na = "NA", append = FALSE, col_names = !append)`

File with arbitrary delimiter
`write_delim(x, path, delim = " ", na = "NA", append = FALSE, col_names = !append)`

CSV for excel
`write_excel_csv(x, path, na = "NA", append = FALSE, col_names = !append)`

String to file
`write_file(x, path, append = FALSE)`

String vector to file, one element per line
`write_lines(x,path, na = "NA", append = FALSE)`

Object to RDS file
`write_rds(x, path, compress = c("none", "gz", "bz2", "xz"), ...)`

Tab delimited files
`write_tsv(x, path, na = "NA", append = FALSE, col_names = !append)`

RStudio

Read Tabular Data - These functions share the common arguments:

`read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"), quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000, n_max), progress = interactive())`

Comma Delimited Files
`read_csv("file.csv")`
To make file.csv run:
`write_file(x = "a,b,c\n1,2,3\n4,5,NA", path = "file.csv")`

Semi-colon Delimited Files
`read_csv2("file2.csv")`
`write_file(x = "a;b;c\n1;2;3\n4;5;NA", path = "file2.csv")`

Files with Any Delimiter
`read_delim("file.txt", delim = "|")`
`write_file(x = "a|b|c\n1|2|3\n4|5|NA", path = "file.txt")`

Fixed Width Files
`read_fwf("file.fwf", col_positions = c(1, 3, 5))`
`write_file(x = "a b c\n1 2 3\n4 5 NA", path = "file.fwf")`

Tab Delimited Files
`read_tsv("file.tsv")` Also `read_table()`.
`write_file(x = "a\tb\tc\n1\t2\t3\n4\t5\tNA", path = "file.tsv")`

USEFUL ARGUMENTS

Example file
`write_file("a,b,c\n1,2,3\n4,5,NA","file.csv")`
`f <- "file.csv"`

Skip lines
`read_csv(f, skip = 1)`

No header
`read_csv(f, col_names = FALSE)`

Read in a subset
`read_csv(f, n_max = 1)`

Provide header
`read_csv(f, col_names = c("x", "y", "z"))`

Missing Values
`read_csv(f, na = c("1", "."))`

Read Non-Tabular Data

Read a file into a single string
`read_file(file, locale = default_locale())`

Read each line into its own string
`read_lines(file, skip = 0, n_max = -1L, na = character(), locale = default_locale(), progress = interactive())`

Read Apache style log files
`read_log(file, col_names = FALSE, col_types = NULL, skip = 0, n_max = -1, progress = interactive())`

Read a file into a raw vector
`read_file_raw(file)`

Read each line into a raw vector
`read_lines_raw(file, skip = 0, n_max = -1L, progress = interactive())`

Data types

readr functions guess the types of each column and convert types when appropriate (but will NOT convert strings to factors automatically).

A message shows the type of each column in the result.

```
## Parsed with column specification:
## cols(
##   age = col_integer(),
##   sex = col_character(),
##   earn = col_double()
## )
```

age is an integer; sex is a character; earn is a double (numeric)

1. Use **problems()** to diagnose problems
   x <- read_csv("file.csv"); problems(x)

2. Use a col_ function to guide parsing
   - **col_guess()** - the default
   - **col_character()**
   - **col_double()**, **col_euro_double()**
   - **col_datetime**(format = "") Also **col_date**(format = ""), **col_time**(format = "")
   - **col_factor**(levels, ordered = FALSE)
   - **col_integer()**
   - **col_logical()**
   - **col_number()**, **col_numeric()**
   - **col_skip()**

   x <- read_csv("file.csv", col_types = cols(A = col_double(), B = col_logical(), C = col_factor()))

3. Else, read in as character vectors then parse with a parse_ function.
   - **parse_guess()**
   - **parse_character()**
   - **parse_datetime()** Also **parse_date()** and **parse_time()**
   - **parse_double()**
   - **parse_factor()**
   - **parse_integer()**
   - **parse_logical()**
   - **parse_number()**

   x$A <- parse_number(x$A)

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with tidyverse.org • readr 1.1.0 • tibble 1.2.12 • tidyr 0.6.0 • Updated: 2017-01

Cheat sheet is freely available at https://www.rstudio.com/resources/cheatsheets/

Your turn...

Quiz 2: Import functions

What function would you use to **read a file where fields are separated with "|"**?

- read_delim()
- read_csv()
- read_csv2()
- read_tsv()
- read_table()
- read_fwf()

Submit | Show Hint | Show Answer | Clear

Quiz 3: Import functions

What function would you use if you **generated a CSV file on your own computer**?

- read_delim()
- read_csv()
- read_csv2()
- read_tsv()
- read_table()
- read_fwf()

Submit | Show Hint | Show Answer | Clear

Quiz 4: Import functions

What arguments do `read_delim()` and `read_csv()` have **NOT** in common?

- [ ] progress
- [ ] quote
- [ ] trim_ws
- [ ] delim
- [ ] escape_backslash
- [ ] guess_max
- [ ] escape_double

Submit | Show Hint | Show Answer | Clear

Quiz 5: Import functions

Identify what is wrong with each of the following inline CSV files. What happens when you run the code? (You'll find the solutions at the end of the presentation.)

```
read_csv("a,b
  1,2,3
  4,5,6")
read_csv("a,b,c
  1,2
  1,2,3,4")
```

```
read_csv("a,b
  1,2
  a,b")
read_csv("a;b
  1;3")
```

Quiz 6: Tibble vs data frame

Compare and contrast the following operations on a `data.frame` and equivalent `tibble`.

```
df <- data.frame(abc = 1, xyz = "a")
df$x
df[, "xyz"]
df[, c("abc", "xyz")]
```

What is different? Why might the default data frame behaviours cause you frustration?

Other types of data

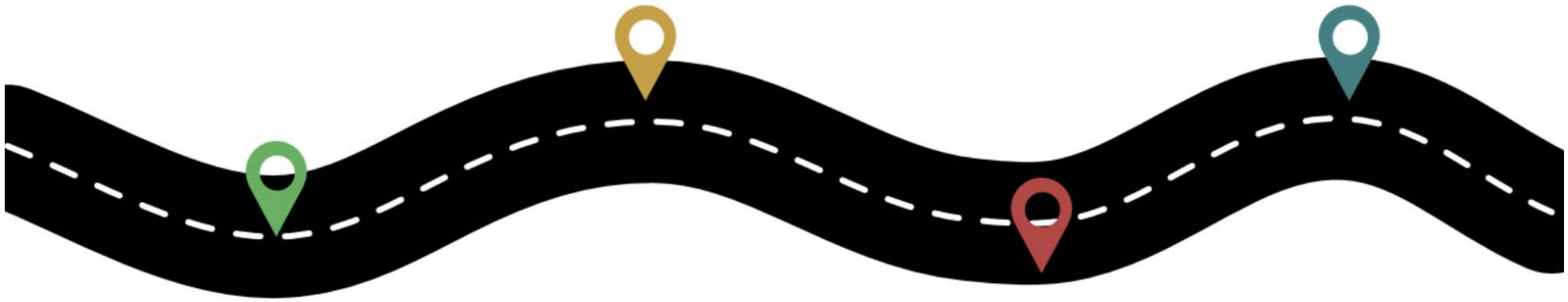
If you have other types of files to import try one of the following packages:

- **haven** - SPSS, Stata, and SAS files
- **readxl** - Excel files (.xls and .xlsx)
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

Roadmap

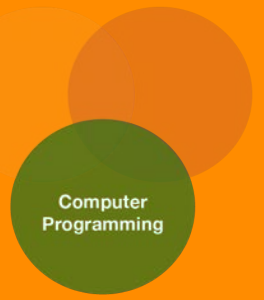
Check:
The numbers of rows and columns.
The type of delimiter.
Column names for special characters and white space ➜ modify names of necessary.

Check results with `View()`, `dim()`, `str()` and `summary()`.



Open the file in any text editor to see the content.

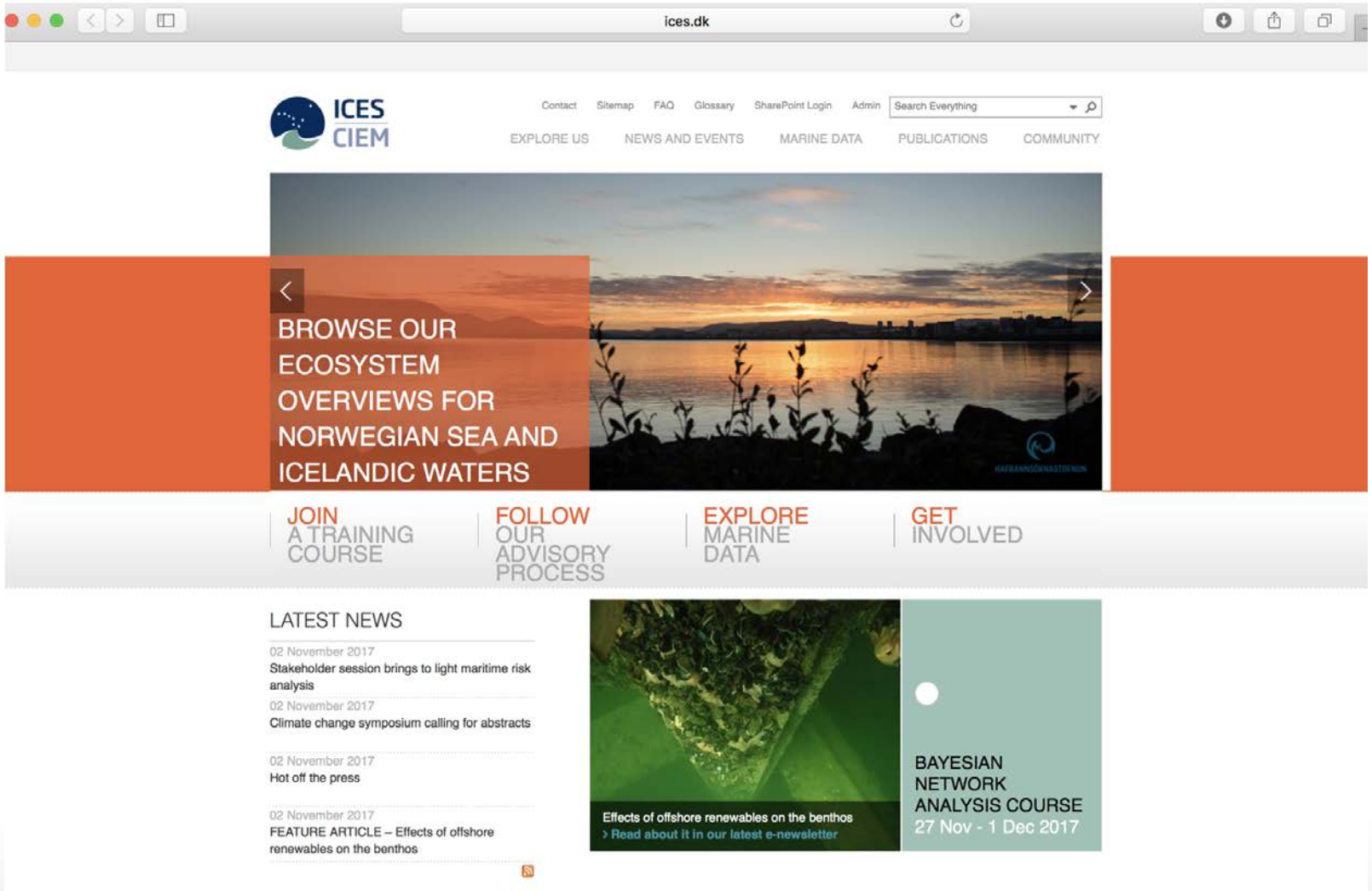
Read in the data with `read_csv()`!

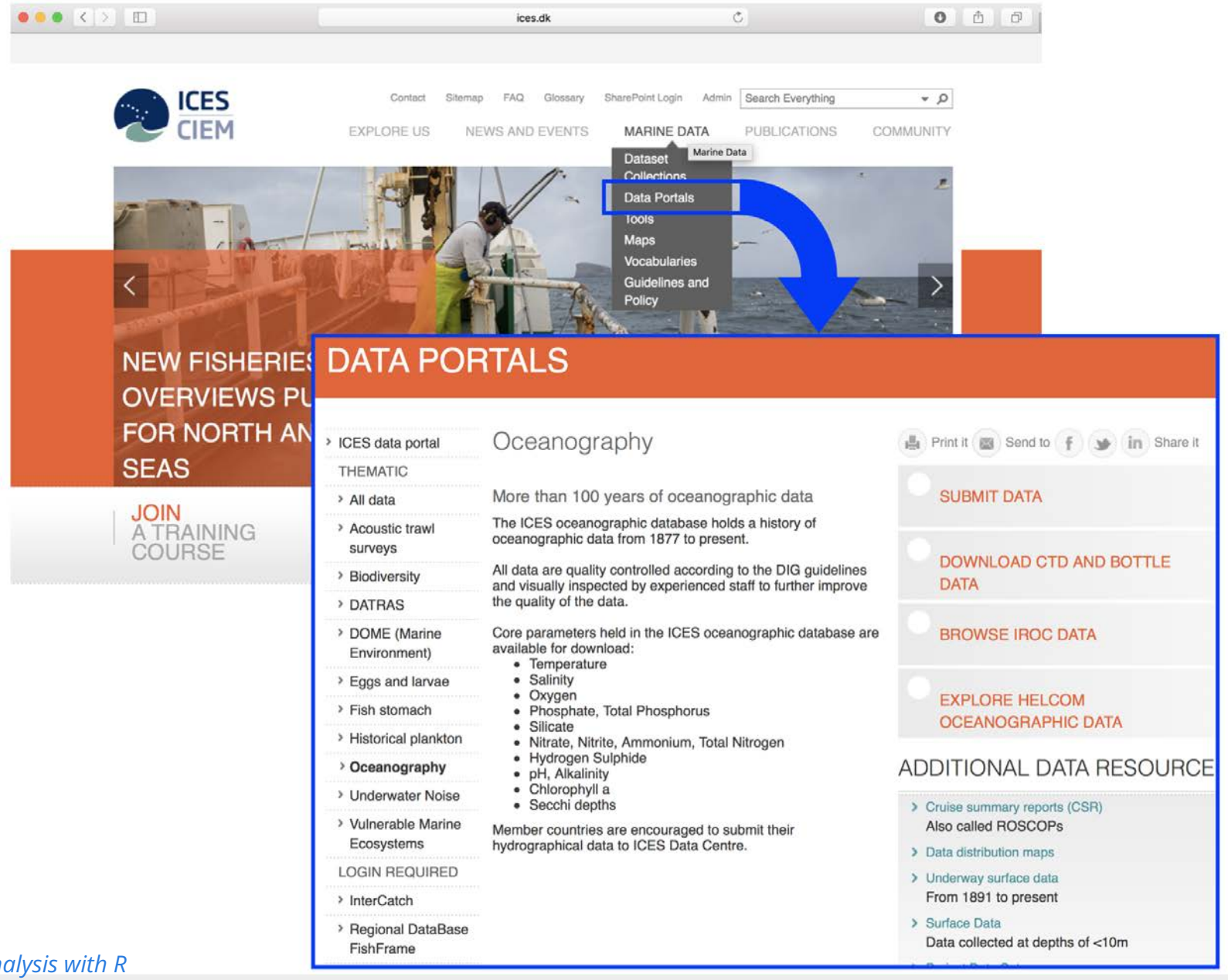


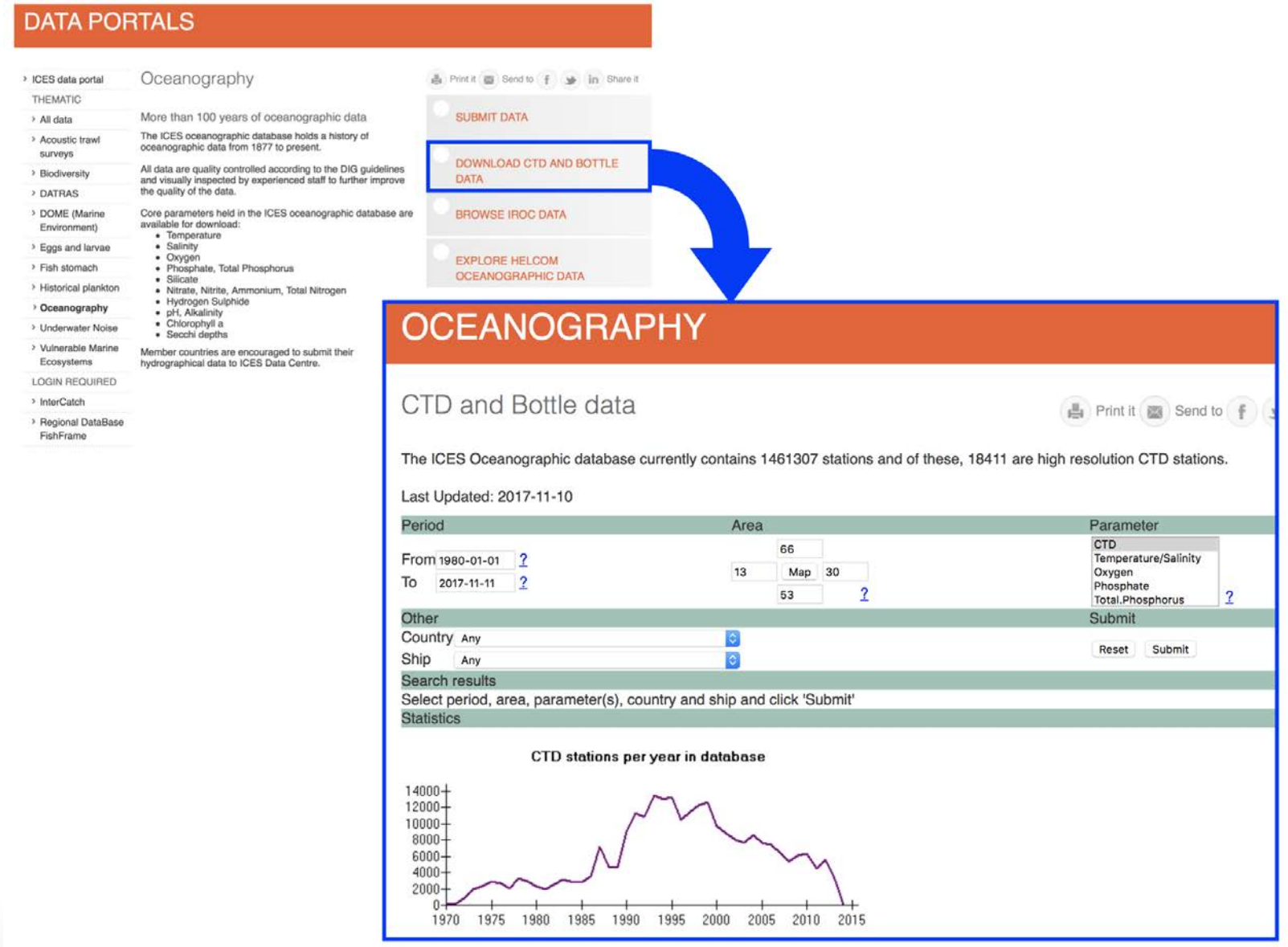
Some real example to import

ICES (International Council of the Exploration of the Sea) provides various types of data on its webpage

www.ices.dk

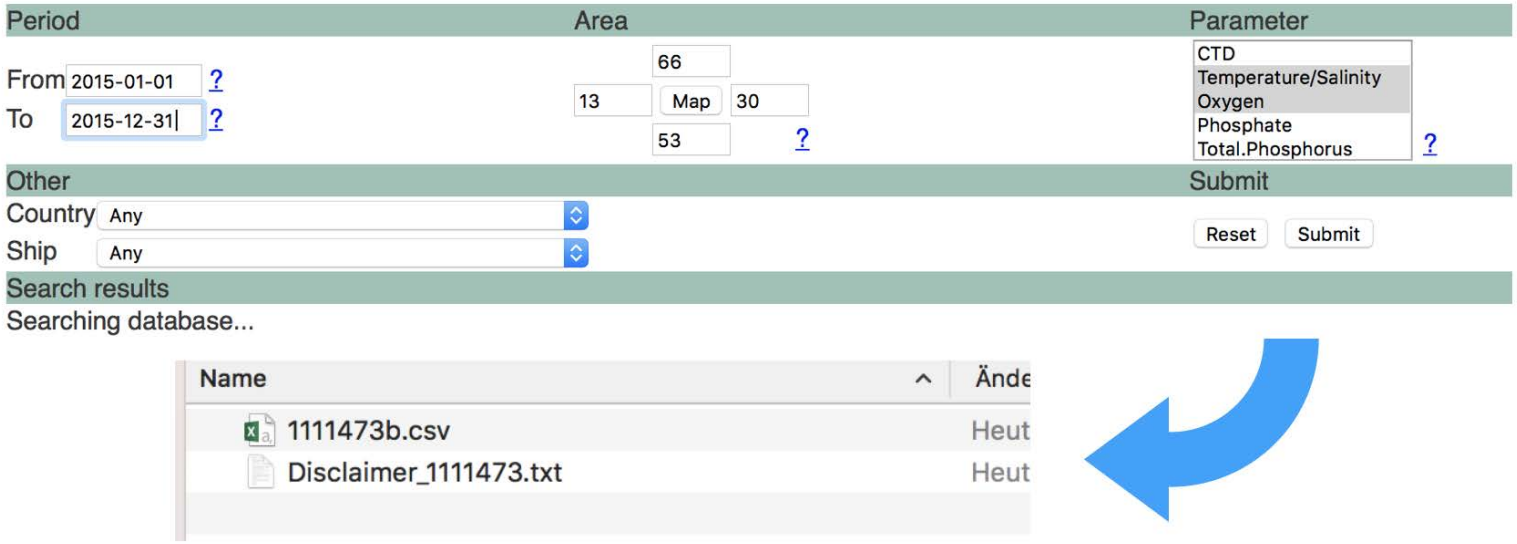



ices.dk
ICES
CIEM
Contact
Sitemap
FAQ
Glossary
SharePoint Login
Admin
Search Everything
EXPLORE US
NEWS AND EVENTS
MARINE DATA
PUBLICATIONS
COMMUNITY
Marine Data
Dataset Collections
Data Portals
Tools
Maps
Vocabularies
Guidelines and Policy
NEW FISHERIES
OVERVIEWS PU
FOR NORTH AN
SEAS
JOIN
A TRAINING
COURSE
DATA PORTALS
ICES data portal
THEMATIC
All data
Acoustic trawl surveys
Biodiversity
DATRAS
DOME (Marine Environment)
Eggs and larvae
Fish stomach
Historical plankton
Oceanography
Underwater Noise
Vulnerable Marine Ecosystems
LOGIN REQUIRED
InterCatch
Regional DataBase FishFrame
Oceanography
More than 100 years of oceanographic data
The ICES oceanographic database holds a history of oceanographic data from 1877 to present.
All data are quality controlled according to the DIG guidelines and visually inspected by experienced staff to further improve the quality of the data.
Core parameters held in the ICES oceanographic database are available for download:
Temperature
Salinity
Oxygen
Phosphate, Total Phosphorus
Silicate
Nitrate, Nitrite, Ammonium, Total Nitrogen
Hydrogen Sulphide
pH, Alkalinity
Chlorophyll a
Secchi depths
Member countries are encouraged to submit their hydrographical data to ICES Data Centre.
Print it
Send to
Share it
SUBMIT DATA
DOWNLOAD CTD AND BOTTLE DATA
BROWSE IROC DATA
EXPLORE HELCOM OCEANOGRAPHIC DATA
ADDITIONAL DATA RESOURCE
Cruise summary reports (CSR)
Also called ROSCOPs
Data distribution maps
Underway surface data
From 1891 to present
Surface Data
Data collected at depths of <10m


DATA PORTALS
ICES data portal
THEMATIC
All data
Acoustic trawl surveys
Biodiversity
DATRAS
DOME (Marine Environment)
Eggs and larvae
Fish stomach
Historical plankton
Oceanography
Underwater Noise
Vulnerable Marine Ecosystems
LOGIN REQUIRED
InterCatch
Regional DataBase FishFrame
Oceanography
More than 100 years of oceanographic data
The ICES oceanographic database holds a history of oceanographic data from 1877 to present.
All data are quality controlled according to the DIG guidelines and visually inspected by experienced staff to further improve the quality of the data.
Core parameters held in the ICES oceanographic database are available for download:
Temperature
Salinity
Oxygen
Phosphate, Total Phosphorus
Silicate
Nitrate, Nitrite, Ammonium, Total Nitrogen
Hydrogen Sulphide
pH, Alkalinity
Chlorophyll a
Secchi depths
Member countries are encouraged to submit their hydrographical data to ICES Data Centre.
Print it
Send to
Share it
SUBMIT DATA
DOWNLOAD CTD AND BOTTLE DATA
BROWSE IROC DATA
EXPLORE HELCOM OCEANOGRAPHIC DATA
OCEANOGRAPHY
CTD and Bottle data
Print it
Send to
The ICES Oceanographic database currently contains 1461307 stations and of these, 18411 are high resolution CTD stations.
Last Updated: 2017-11-10
Period
From 1980-01-01
To 2017-11-11
Area
66
13
Map
30
53
Parameter
CTD
Temperature/Salinity
Oxygen
Phosphate
Total.Phosphorus
Other
Country Any
Ship Any
Submit
Reset
Submit
Search results
Select period, area, parameter(s), country and ship and click 'Submit'
Statistics
CTD stations per year in database
14000
12000
10000
8000
6000
4000
2000
0
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015

We will work with hydrographical data from the Baltic Sea for 2015:

- **Temperature**
- **Salinity**
- **Oxygen**



The downloaded zip file will contain a CSV file.

Lets import the data following the roadmap

1.Step: Open the file in the editor to check the content

Lets import the data following the roadmap

1.Step: Open the file in the editor to check the content

- Go to the 'Files' pane.

Lets import the data following the roadmap

1.Step: Open the file in the editor to check the content

- Go to the 'Files' pane.
- Find the file "1111473b.csv" in the folder 'Data_Analysis_with_R/data' .

Lets import the data following the roadmap

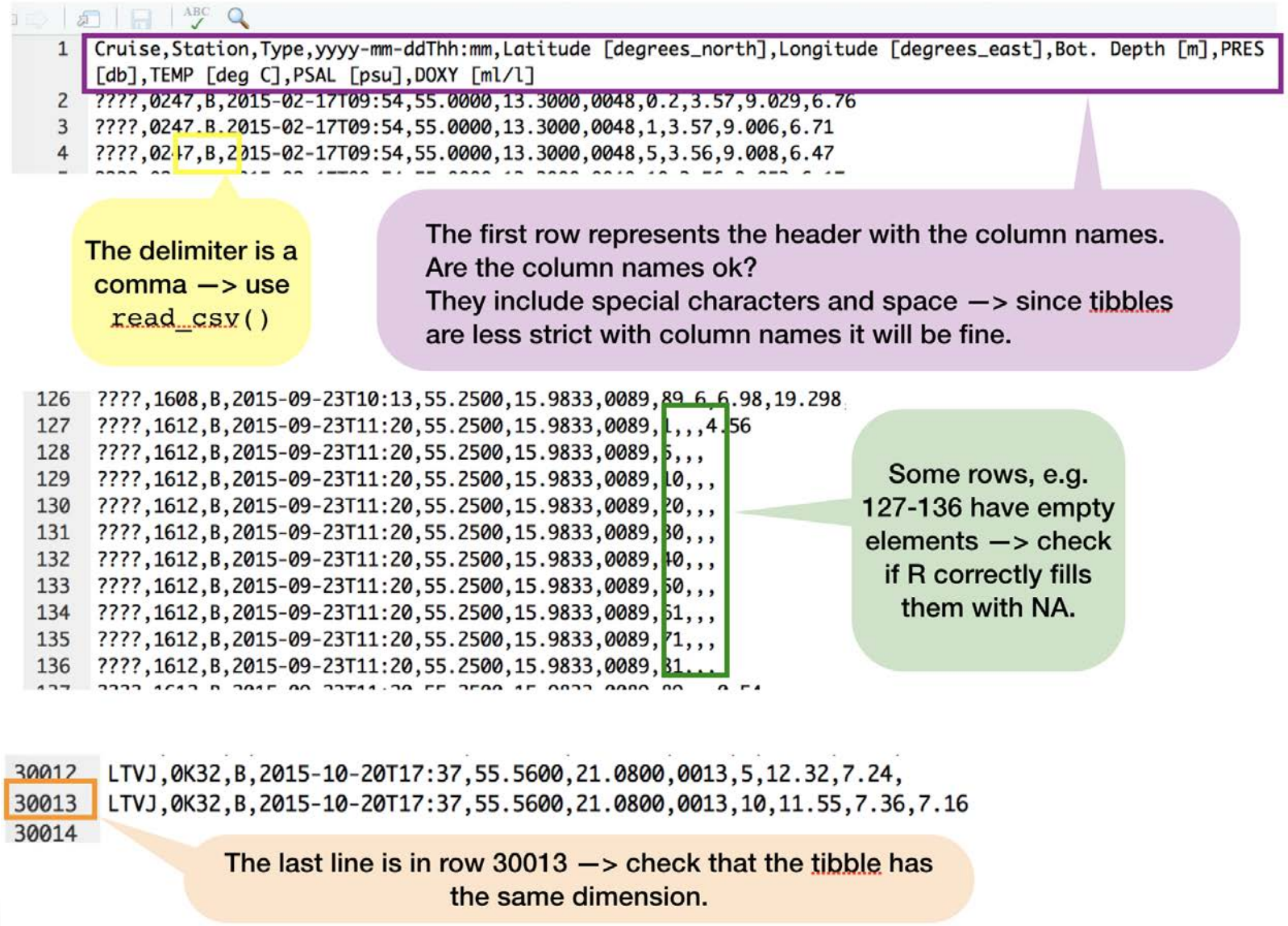
1.Step: Open the file in the editor to check the content

- Go to the 'Files' pane.
- Find the file "1111473b.csv" in the folder 'Data_Analysis_with_R/data' .
- Click on the file and choose "View file", which opens the file in the 'Source' pane.

Lets import the data following the roadmap

1.Step: Open the file in the editor to check the content

- Go to the 'Files' pane.
- Find the file "1111473b.csv" in the folder 'Data_Analysis_with_R/data' .
- Click on the file and choose "View file", which opens the file in the 'Source' pane.
- Lets check


1 Cruise,Station,Type,yyyy-mm-ddThh:mm,Latitude [degrees_north],Longitude [degrees_east],Bot. Depth [m],PRES [db],TEMP [deg C],PSAL [psu],DOXY [ml/l]
2 ????,0247,B,2015-02-17T09:54,55.0000,13.3000,0048,0.2,3.57,9.029,6.76
3 ????,0247,B,2015-02-17T09:54,55.0000,13.3000,0048,1,3.57,9.006,6.71
4 ????,0247,B,2015-02-17T09:54,55.0000,13.3000,0048,5,3.56,9.008,6.47
The delimiter is a comma —> use read_csv()
The first row represents the header with the column names. Are the column names ok? They include special characters and space —> since tibbles are less strict with column names it will be fine.
126 ????,1608,B,2015-09-23T10:13,55.2500,15.9833,0089,89,6,6.98,19.298
127 ????,1612,B,2015-09-23T11:20,55.2500,15.9833,0089,1,,,4.56
128 ????,1612,B,2015-09-23T11:20,55.2500,15.9833,0089,5,,,
129 ????,1612,B,2015-09-23T11:20,55.2500,15.9833,0089,10,,,
130 ????,1612,B,2015-09-23T11:20,55.2500,15.9833,0089,20,,,
131 ????,1612,B,2015-09-23T11:20,55.2500,15.9833,0089,30,,,
132 ????,1612,B,2015-09-23T11:20,55.2500,15.9833,0089,40,,,
133 ????,1612,B,2015-09-23T11:20,55.2500,15.9833,0089,50,,,
134 ????,1612,B,2015-09-23T11:20,55.2500,15.9833,0089,51,,,
135 ????,1612,B,2015-09-23T11:20,55.2500,15.9833,0089,71,,,
136 ????,1612,B,2015-09-23T11:20,55.2500,15.9833,0089,81,,,
Some rows, e.g. 127-136 have empty elements —> check if R correctly fills them with NA.
30012 LTVJ,0K32,B,2015-10-20T17:37,55.5600,21.0800,0013,5,12.32,7.24,
30013 LTVJ,0K32,B,2015-10-20T17:37,55.5600,21.0800,0013,10,11.55,7.36,7.16
30014
The last line is in row 30013 —> check that the tibble has the same dimension.

Lets import the data following the roadmap

2.Step: Read the data into R

- Make sure before that you've set the **working directory** correct.
- The **wrong** working directory is the **most common reason** for error messages!

```
hydro <- read_csv("data/1111473b.csv")
```

Lets check the data

```
print(hydro, n = 5)
```

```
## # A tibble: 30,012 x 11
##   Cruise Station Type  `yyyy-mm-ddThh:mm`  `Latitude [degr…
##   <chr>  <chr>   <chr> <dttm>                         <dbl>
## 1 ????   0247    B     2015-02-17 09:54:00               55
## 2 ????   0247    B     2015-02-17 09:54:00               55
## 3 ????   0247    B     2015-02-17 09:54:00               55
## 4 ????   0247    B     2015-02-17 09:54:00               55
## 5 ????   0247    B     2015-02-17 09:54:00               55
## # ... with 3.001e+04 more rows, and 6 more variables: `Longitude
## #   [degrees_east]` <dbl>, `Bot. Depth [m]` <chr>, `PRES [db]` <dbl>,
## #   `TEMP [deg C]` <dbl>, `PSAL [psu]` <dbl>, `DOXY [ml/l]` <dbl>
```

Change column names

To make subsetting and data manipulation easier change the column names, e.g.

```
names(hydro) <- c("cruise", "station", "type", "date_time",
  "lat", "long", "depth", "pres", "temp", "psal", "doxy")
```

Your turn...

Checking tasks:

1. Read the file into your workspace.
2. Check the dimensions of the tibble `hydro`. Do they match with what you've seen in the Editor?
3. What happened with the empty elements in, e.g., row 127-136?
4. Do you agree with the data types of each column?

Quiz 7: Test your R knowledge

Subset the data to get only observations of Station "0613" and

1. calculate the mean salinity (psal) and

2. mean oxygen concentration (doxy)

3. Calculate the mean temperature ('temp') for the surface layer (1-10 m depth = 'pres' 1-10), averaged across all stations and cruises for the entire year.

Submit | Show Hint | Show Answer | Clear

Saving and exporting data

Saving your data as R objects

You can save your tibble or data frame as an .R object and load it later with `save(your_tibble, "filename")` and `load("filename")`:

```
save(your_subset, file = "My_first_object.R")
# Lets remove your subset and see what happens when we load it again
rm(your_subset)
your_subset
load(file = "My_first_object.R")
your_subset # now it should be back again
```

Exporting your data

- If you want to export your data for other programs, its best if you stick to the same format as you import, e.g. CSV files.
- Most import functions in 'readr' have a corresponding export function:
  - `read_delim()` --> **`write_delim()`**
  - `read_csv()` --> **`write_csv()`**
  - `read_tsv()` --> **`write_tsv()`**
  - other functions: **`write_excel_csv()`**

Your turn...

Task: Saving and Exporting

With the subsets you created (or any other tibble/data frame in your workspace):

- save one as an R object and
- one as CSV file

`install.packages()`, `library()`, `require()`, `search()`, `detach()`, `vignette()`, `browseVignettes()`,

`read_delim()`, `read_csv()`, `read_csv2()`, `read_tsv()`, `read_table()`, `read_fwf()`, `read_log()`,

`tibble()`, `as_tibble()`, `print()`, `save()`, `load()`,

`write_delim()`, `write_csv()`, `write_tsv()`, `write_excel_csv()`

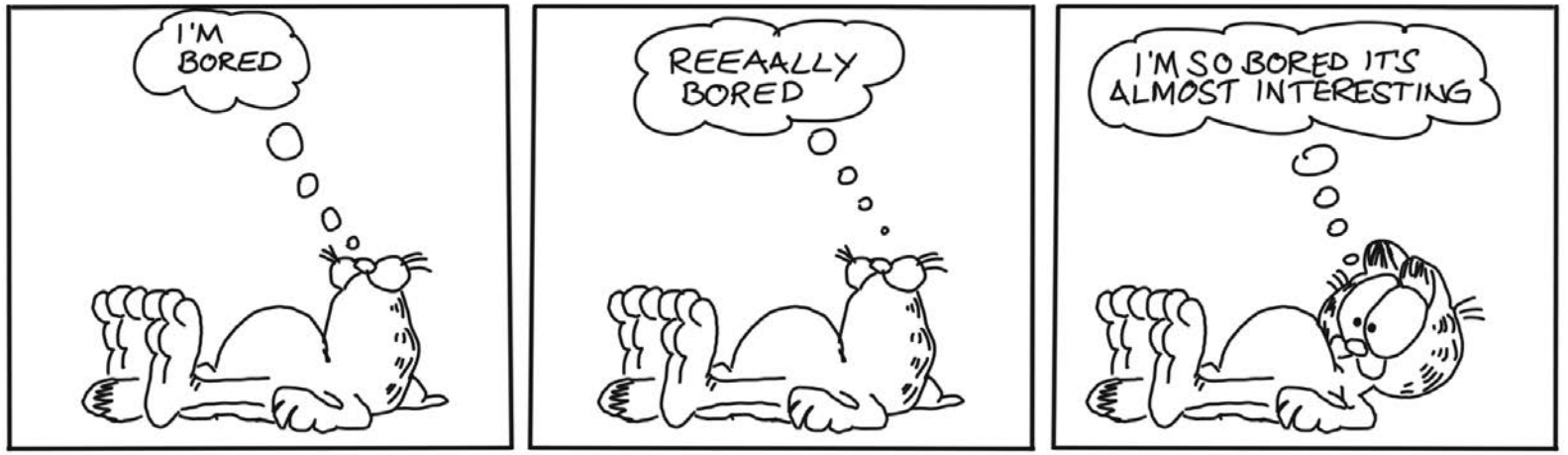
Overview of functions you learned today

How do you feel now.....?

Totally confused?

Go thorougly through the tasks and quizzes. Read the chapter 10 Tibbles and 11 Data import in 'R for Data Science'.

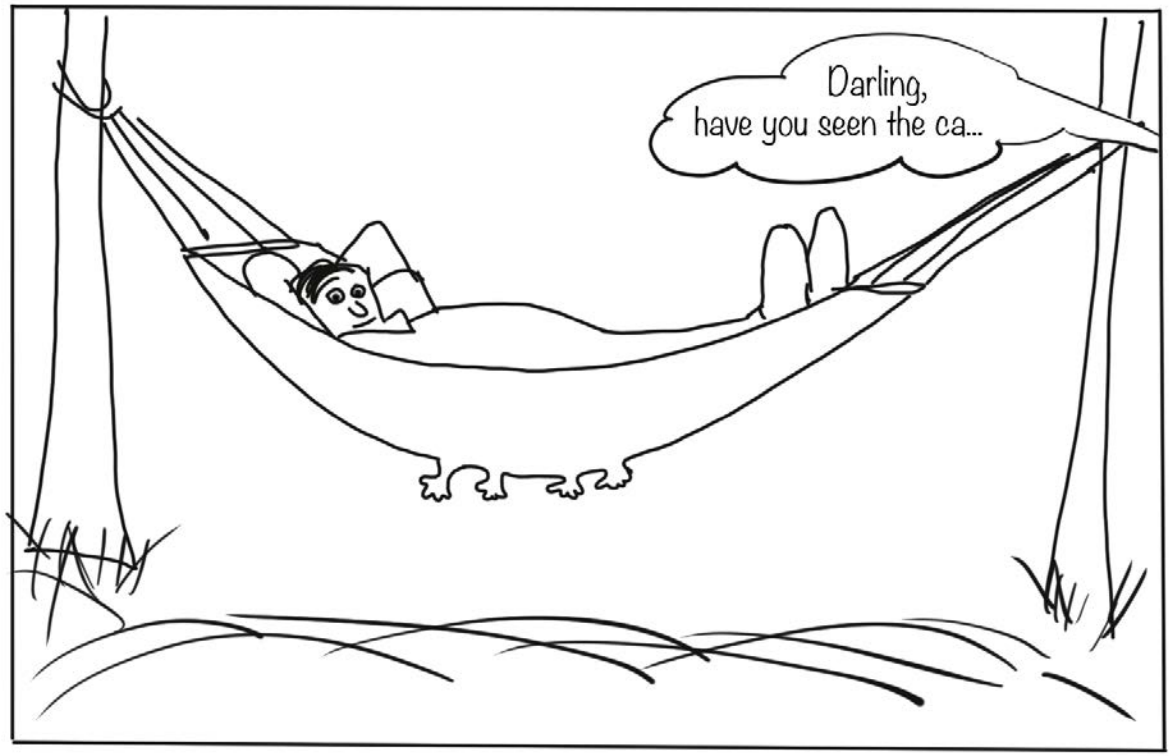
Totally bored?



Then try out to import, explore and export other datasets you have (from Excel).

Totally content?

Then go grab a coffee, lean back and enjoy the rest of the day...!



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Thank You

For more information contact me: saskia.otto@uni-hamburg.de

http://www.researchgate.net/profile/Saskia_Otto
http://www.github.com/saskiaotto

Image on title and end slide: Section of an infrared satallite image showing the Larsen C ice shelf on the Antarctic Peninsula - USGS/NASA Landsat: A Crack of Light in the Polar Dark, Landsat 8 - TIRS, June 17, 2017
(under CC0 license)

Solutions

Quiz 5: Import functions

1. Example: The header has 1 element (=column) less than the data --> **R skips the 3rd element** of each data row then completely (3 and 6 are not shown anymore).

2. Example: The 1st data row has 1 element less than the header and the 2nd data row --> R automatically **fills the missing element with a NA**.

3. Example: The data rows have mixed data types --> R **coerces** all values to the more general **character** data type .

4. Example: Remember, the function `read_csv()` expects a **comma** as delimiter, NOT a semicolon --> R reads it then as 1 element per row. Try alternatively:

```
read_csv2("a;b
  1;3")
```

Quiz 6: Tibble vs. data frame

For tibbles the complete column name is needed. This can be useful in case "x" doesn't exist but 2 other columns that contain the letter x in their names. If you subset tibbles like a matrix ([row, col]) you will always get a tibble returned and no vectors (as data frames do in the 2nd example).

```
df_tbl <- as_tibble(df)
df_tbl$x
```

```
## NULL
```

```
df_tbl[, "xyz"]
```

```
## # A tibble: 1 x 1
##   xyz
##   <fct>
## 1 a
```

```
df_tbl[, c("abc", "xyz")]
```

```
## # A tibble: 1 x 2
##     abc xyz
##   <dbl> <fct>
## 1     1 a
```