Data Handling: Import, Cleaning and Visualisation

Lecture 5:

Rectangular data

Dr. Aurélien Sallin

Recap and Warm-up

Structured Data Formats

- Still text files, but with standardized **structure**.
- **Special characters** define the structure.
- More complex **syntax**, more complex structures can be represented…
- Example: using a parser to work with a csv file.

Structures to work with (in R)

We distinguish two basic characteristics:

1. Data types:
   - **integers**;
   - **real numbers** ('numeric values', 'doubles', floating point numbers);
   - **characters** ('string', 'character values');
   - (**booleans**)
2. Basic **data structures** in RAM:
   - **Vectors**
   - **Factors**
   - **Arrays/Matrices**
   - **Lists**
   - **Data frames** (very **R**-specific)

Erratum

The following code does not throw an error. **R** throws a **warning()** but fills the matrix restarting from the beginning of the vector.

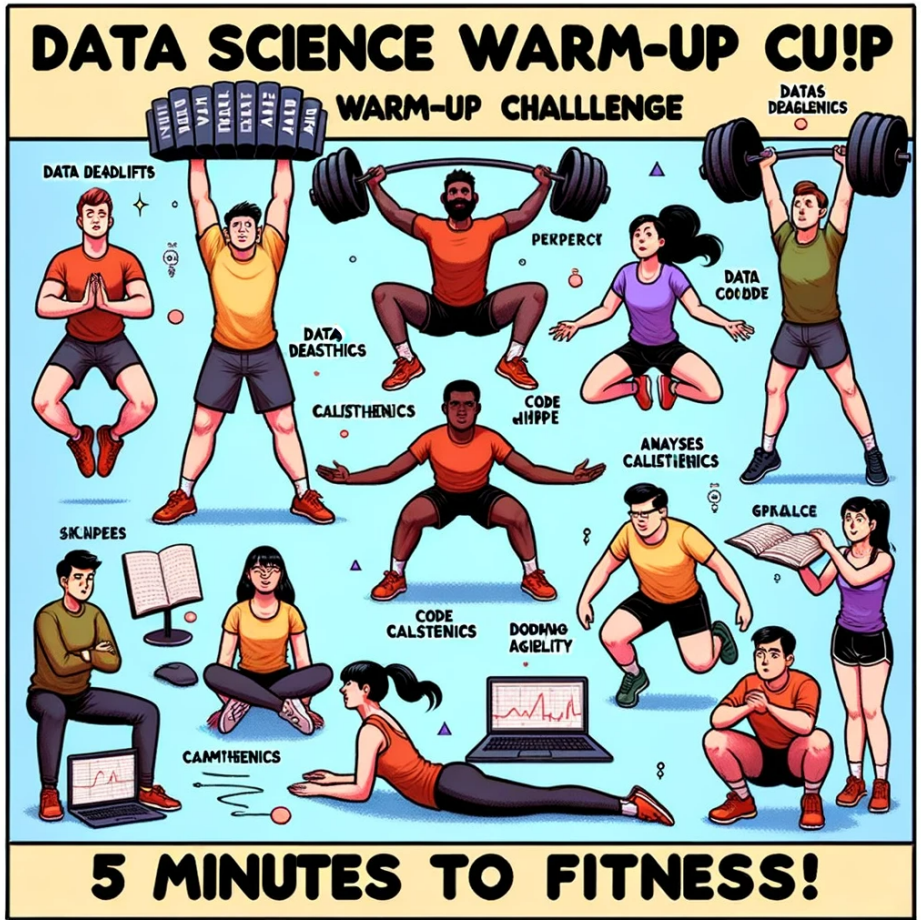
```
erratum <- matrix(1:13, nrow = 3)
```

```
## Warning in matrix(1:13, nrow = 3): Datenlänge [13] ist kein Teiler oder Vielfaches der Anzahl der
## Zeilen [3]
```

```
erratum
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    4    7   10   13
## [2,]    2    5    8   11    1
## [3,]    3    6    9   12    2
```

Warm-up


DATA SCIENCE WARM-UP CU!P
WARM-UP CHALLLENGE
DATAS DEAGLENICS
DATA DEADLIFTS
PERPERCT
DATA COLDDE
DATA DEASTHICS
CALISTHENICS
CODE
ANAYSES CALISTIENICS
SKNPEES
CODE CALSTENICS
AGIELITY
CAMITHENICS
5 MINUTES TO FITNESS!

Data structure

```
00000000: efbb bf6e 616d 652c 6167 655f 696e 5f79  ...name,age_in_y
00000010: 6561 7273 0d0a 4a6f 686e 2c32 340d 0a41  ears..John,24..
00000020: 6e6e 612c 3239 0d0a 4265 6e2c 3331 0d0a  nna,29..Ben,31..
00000030: 4c69 7a2c 3334 0d0a 4d61 782c 3237       Liz,34..Max,27
```

- Describe this code. What are these digits? What do they represent?

Data structure

```
00000000: efbb bf6e 616d 652c 6167 655f 696e 5f79  ...name,age_in_y
00000010: 6561 7273 0d0a 4a6f 686e 2c32 340d 0a41  ears..John,24..
00000020: 6e6e 612c 3239 0d0a 4265 6e2c 3331 0d0a  nna,29..Ben,31..
00000030: 4c69 7a2c 3334 0d0a 4d61 782c 3237       Liz,34..Max,27
```

- Describe this code. What are these digits? What do they represent?
- Which encoding is used here?
- Can you identify the EOL (End-of-Line) character?
- Can you identify the comma?

Matrices

What is the output of the following code?

```
my_matrix <- matrix(1:12, nrow = 3)
dim(my_matrix)
```

Matrices

What happens with this command? (Multiple answers can be correct)

```
my_matrix <- cbind(c(1,2,3, 4), c("a", "b", "c", "a"), c(TRUE, FALSE, TRUE, TRUE))
```

- **R** creates a matrix of dimension 3, 4
- **my_matrix[2, 1] == "2"** gives the solution **TRUE**
- **R** must coerce the data to a common type to accommodate all different values
- **mean(my_matrix[,1]) == 2.5** returns **2.5**

Factors

What does the following code produce?

```
fruits <- factor(c("apple", "banana", "apple", "cherry"))

levels(fruits)
as.numeric(fruits)
```

Data in Economics

Data

Rectangular data

- Rectangular data refers to a data structure where information is organized into **rows** and **columns**.
    - Each row represents an observation or instance of the data.
    - Each column represents a variable or feature of the data.

Non-rectangular data

Data

Rectangular data

- Rectangular data refers to a data structure where information is organized into **rows** and **columns**.
    - CSV (typical for rectangular/table-like data) and variants of CSV (tab-delimited, fix length etc.)
    - Excel spreadsheets (`.xls`)
    - Formats specific to statistical software (SPSS: `.sav`, STATA: `.dat`, etc.)
    - Built-in R datasets
    - Binary formats

Non-rectangular data

Data

Rectangular data

Non-rectangular data

- Hierarchical data (xml, html, json)
  - XML and JSON (useful for complex/high-dimensional data sets).
  - HTML (a markup language to define the structure and layout of webpages).
- Time series data
- Unstructed text data
- Images/Pictures data

Working with rectangular data in R

Accessing Rectangular Data

Loading built-in datasets

In order to load such datasets, simply use the **`data()`**-function:

```
data(swiss)

data(mtcars)
```

Inspect the data after loading

```r
# inspect the structure
str(swiss)
```

```
## 'data.frame':    47 obs. of  6 variables:
##  $ Fertility       : num  80.2 83.1 92.5 85.8 76.9 76.1 83.8 92.4 82.4 82.9 ...
##  $ Agriculture     : num  17 45.1 39.7 36.5 43.5 35.3 70.2 67.8 53.3 45.2 ...
##  $ Examination     : int  15 6 5 12 17 9 16 14 12 16 ...
##  $ Education       : int  12 9 5 7 15 7 7 8 7 13 ...
##  $ Catholic        : num  9.96 84.84 93.4 33.77 5.16 ...
##  $ Infant.Mortality: num  22.2 22.2 20.2 20.3 20.6 26.6 23.6 24.9 21 24.4 ...
```

```r
# look at the first few rows
head(swiss)
```

```
##              Fertility Agriculture Examination Education Catholic Infant.Mortality
## Courtelary        80.2        17.0          15        12     9.96             22.2
## Delemont          83.1        45.1           6         9    84.84             22.2
## Franches-Mnt      92.5        39.7           5         5    93.40             20.2
## Moutier           85.8        36.5          12         7    33.77             20.3
## Neuveville        76.9        43.5          17        15     5.16             20.6
## Porrentruy        76.1        35.3           9         7    90.57             26.6
```

Work with data.frames

Select columns

```r
swiss$Fertility # use the $-operator
```

```r
swiss[,1] # use brackets [] and the column number/index
```

```r
swiss[, "Fertility"] # use the name of the column
```

```r
swiss[, c("Fertility", "Agriculture")] # use the name of the column
```

Select rows

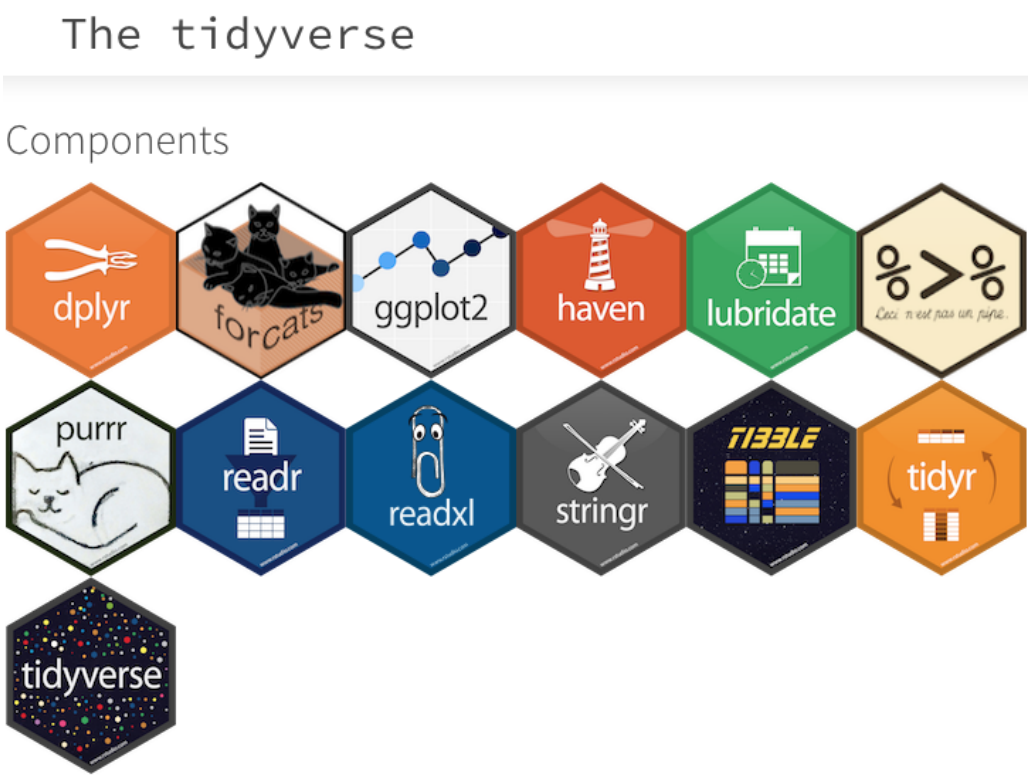
```r
swiss[1,]  # First row
```

```r
swiss[swiss$Fertility > 40,]  # Based on condition ("filter")
```

Data.frames vs tibbles

- data frames: base `R`
- tibbles: `tidyverse`

Data.frames vs tibbles



Data.frames vs tibbles

Similar!

- Used in the tidyverse and ggplot2 packages.
- Same information as a data frame.
- Slight differences in the manipulation and representation of data.
- See Tibble vs. DataFrame for more details.

Data.frames vs tibbles

```
library(tidyverse)

as_tibble(swiss)
```

```
## # A tibble: 47 × 6
##    Fertility Agriculture Examination Education Catholic Infant.Mortality
##        <dbl>       <dbl>       <int>     <int>    <dbl>             <dbl>
##  1      80.2        17            15        12     9.96              22.2
##  2      83.1        45.1           6         9    84.8               22.2
##  3      92.5        39.7           5         5    93.4               20.2
##  4      85.8        36.5          12         7    33.8               20.3
##  5      76.9        43.5          17        15     5.16              20.6
##  6      76.1        35.3           9         7    90.6               26.6
##  7      83.8        70.2          16         7    92.8               23.6
##  8      92.4        67.8          14         8    97.2               24.9
##  9      82.4        53.3          12         7    97.7               21
## 10      82.9        45.2          16        13    91.4               24.4
## # ℹ 37 more rows
```

Importing Rectangular Data from Text-Files

Comma Separated Values (CSV)

The **`swiss`**-dataset would look like this when stored in a CSV:

```
"District","Fertility","Agriculture","Examination","Education","Catholic","Infant.Mortality"
"Courtelary",80.2,17,15,12,9.96,22.2
```

What do we need to read this format properly?

Parsing CSVs in R

- **`read.csv()`** (basic R distribution)
- Returns a **`data.frame`**

```
swiss_imported <- read.csv("data/swiss.csv")
```

Parsing CSVs in R

- Alternative: **`read_csv()`** (**`readr`**/**`tidyr`**-package)
- Returns a **`tibble`**.
- Used in Wickham and Grolemund (2017).

```
swiss_imported <- read_csv("data/swiss.csv")
```

Import and parsing with `readr`

- Why `readr`?
  - Functions for all common rectangular data formats.
- Consistent syntax.
- More robust and faster than similar functions in basic R.
- Alternative: The `data.table`-package (handling large datasets).

Basic usage of readr functions

Parse the first lines of the swiss dataset directly like this…

```
library(readr)

read_csv('"District","Fertility","Agriculture","Examination","Education","Catholic","Infant.Mortality"
"Courtelary",80.2,17,15,12,9.96,22.2')
```

```
## Rows: 1 Columns: 7
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr (1): District
## dbl (6): Fertility, Agriculture, Examination, Education, Catholic, Infant.Mortality
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## # A tibble: 1 × 7
##   District   Fertility Agriculture Examination Education Catholic Infant.Mortality
##   <chr>          <dbl>       <dbl>       <dbl>     <dbl>    <dbl>            <dbl>
## 1 Courtelary      80.2          17          15        12     9.96             22.2
```

or read the entire **swiss** dataset by pointing to the file

Basic usage of readr functions

In either case, the result is a **tibble**:

```
swiss
```

```
## # A tibble: 47 × 7
##    District     Fertility Agriculture Examination Education Catholic Infant.Mortality
##    <chr>            <dbl>       <dbl>       <dbl>     <dbl>    <dbl>            <dbl>
##  1 Courtelary        80.2        17             15        12     9.96             22.2
##  2 Delemont          83.1        45.1            6         9    84.8              22.2
##  3 Franches-Mnt      92.5        39.7            5         5    93.4              20.2
##  4 Moutier           85.8        36.5           12         7    33.8              20.3
##  5 Neuveville        76.9        43.5           17        15     5.16             20.6
##  6 Porrentruy        76.1        35.3            9         7    90.6              26.6
##  7 Broye             83.8        70.2           16         7    92.8              23.6
##  8 Glane             92.4        67.8           14         8    97.2              24.9
##  9 Gruyere           82.4        53.3           12         7    97.7              21
## 10 Sarine            82.9        45.2           16        13    91.4              24.4
## # ℹ 37 more rows
```

Basic usage of `readr` functions

- Other `readr` functions have practically the same syntax and behavior.
- `read_tsv()` (tab-separated)
- `read_fwf()` (fixed-width)
- …

Parsing CSVs

Recognizing columns and rows is one thing…

```
swiss
```

```
## # A tibble: 47 × 7
##    District     Fertility Agriculture Examination Education Catholic Infant.Mortality
##    <chr>            <dbl>       <dbl>       <dbl>     <dbl>    <dbl>            <dbl>
##  1 Courtelary        80.2        17             15        12     9.96             22.2
##  2 Delemont          83.1        45.1            6         9    84.8              22.2
##  3 Franches-Mnt      92.5        39.7            5         5    93.4              20.2
##  4 Moutier           85.8        36.5           12         7    33.8              20.3
##  5 Neuveville        76.9        43.5           17        15     5.16             20.6
##  6 Porrentruy        76.1        35.3            9         7    90.6              26.6
##  7 Broye             83.8        70.2           16         7    92.8              23.6
##  8 Glane             92.4        67.8           14         8    97.2              24.9
##  9 Gruyere           82.4        53.3           12         7    97.7              21
## 10 Sarine            82.9        45.2           16        13    91.4              24.4
## # ℹ 37 more rows
```

What else did `read_csv()` recognize?

Parsing CSVs

- Recall the introduction to data structures and data types in R
- How does R represent data in RAM
  - Structure: `data.frame`/`tibble`, etc.
  - Types: `character`, `numeric`, etc.
- Parsers in `read_csv()` guess the data types.

Parsing CSV-columns

- `"12:00"`: type `character`?

Parsing CSV-columns

- `"12:00"`: type `character`?
- What about `c("12:00", "midnight", "noon")`?

Parsing CSV-columns

- `"12:00"`: type `character`?
- What about `c("12:00", "midnight", "noon")`?
- And now `c("12:00", "14:30", "20:01")`?

Parsing CSV-columns

Let's test it!

```
read_csv('A,B
         12:00, 12:00
         14:30, midnight
         20:01, noon')
```

```
## Rows: 3 Columns: 2
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr  (1): B
## time (1): A
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## # A tibble: 3 × 2
##   A      B
##   <time> <chr>
## 1 12:00  12:00
## 2 14:30  midnight
## 3 20:01  noon
```

How can `read_csv()` distinguish the two cases?

Parsing CSV-columns: guess types

Under the hood **`read_csv()`** used the **`guess_parser()`**- function to determine which type the two vectors likely contain:

```
guess_parser(c("12:00", "midnight", "noon"))
```

```
## [1] "character"
```

```
guess_parser(c("12:00", "14:30", "20:01"))
```

```
## [1] "time"
```

Other Common Rectangular Formats

Spreadsheets/Excel

Needs additional R-package: **`readxl`**.

```
# install the package
install.packages("readxl")
```

Spreadsheets/Excel

Then we load this additional package ('library') and use the package's **`read_excel()`**-function to import data from an excel-sheet.

```
# load the package
library(readxl)

# import data from a spreadsheet
swiss_imported <- read_excel("data/swiss.xlsx")
```

Data from other data analysis software

- STATA, SPSS, etc.
- Additional packages needed:
  - `foreign`
  - `haven`
- Parsers (functions) for many foreign formats.
  - For example, `read_spss()` for SPSS' `.sav`-format.

Data from other data analysis software

```r
# install the package (if not yet installed):
# install.packages("haven")

# load the package
library(haven)

# read the data
swiss_imported <- read_spss("data/swiss.sav")
```

Encoding Issues

Recognize the problem

```
FILE <- "../../data/hastamanana.txt"
hasta <- readLines(FILE)
hasta
```

```
## [1] "Hasta Ma\xf1ana!"
```

(**`readLines()`** simply reads the content of a text file line by line.)

Guess encoding

- Recall that there are no meta data in csv or plain text file informing you about the encoding.
- If no other information is available, we need to make an educated guess.
- **`readr`** provides a function that does just that: **`guess_encoding()`**

```
readr::guess_encoding(FILE)
```

```
## # A tibble: 3 × 2
##   encoding   confidence
##   <chr>           <dbl>
## 1 ISO-8859-2       0.64
## 2 ISO-8859-1       0.42
## 3 ISO-8859-9       0.21
```

Handling encoding issues

- **`inconv()`**: convert a character vector **`from`** one encoding **`to`** another encoding.
- Use the guessed encoding for the **`from`** argument

```
iconv(hasta, from = "ISO-8859-2", to = "UTF-8")
```

```
## [1] "Hasta Mańana!"
```

```
iconv(hasta, from = "ISO-8859-1", to = "UTF-8")
```

```
## [1] "Hasta Mañana!"
```

Tutorial: a first data pipeline

Organize your data pipeline!

- One R script to gather/import data.
- The beginning of your data pipeline!

Do not overlook this step!!!

A Template/Blueprint

Tell your future self what this script is all about 🤓🔮💻

```
#########################################################################
# Data Handling Course: Example Script for Data Gathering and Import
#
# Imports data from ...
# Input: import c to data sources (data comes in ... format)
# Output: cleaned data as CSV
#
# A. Sallin, St. Gallen, 2023
#########################################################################
```

Script sections

- Recall: programming tasks can often be split into smaller tasks.
- Use **sections** to implement task-by-task and keep order.
- In RStudio: Use `----------` to indicate the beginning of sections.
  - CTRL + SHIFT + R
- Start with a ‘meta’-section.

Script sections

```
##########################################################################
# Data Handling Course: Example Script for Data Gathering and Import
#
# Imports data from ...
# Input: import c to data sources (data comes in ... format)
# Output: cleaned data as CSV
#
# A. Sallin, St. Gallen, 2023
##########################################################################


# SET UP --------------
# load packages
library(tidyverse)

# set fix variables
INPUT_PATH <- "/rawdata"
OUTPUT_FILE <- "/final_data/datafile.csv"
```

Script sections

Finally we add sections with the actual code (in the case of a data import script, maybe one section per data source)

```
##############################################################################
# Data Handling Course: Example Script for Data Gathering and Import
#
# Imports data from ...
# Input: import c to data sources (data comes in ... format)
# Output: cleaned data as CSV
#
# A. Sallin, St. Gallen, 2023
##############################################################################


# SET UP --------------
# load packages
library(tidyverse)

# set fix variables
INPUT_PATH <- "/rawdata"
OUTPUT_FILE <- "/final_data/datafile.csv"


# IMPORT RAW DATA FROM CSVs -------------
```

Let’s code!

Q&A

References

Wickham, Hadley, and Garrett Grolemund. 2017. Sebastopol, CA: O'Reilly. http://r4ds.had.co.nz/.