Data Handling: Import, Cleaning and Visualisation

Lecture 5:
Rectangular data

Dr. Aurélien Sallin

2024-10-17

Recap


81. Schweizer Messe für
Landwirtschaft und Ernährung
Gastkanton St. Gallen
St. Gallen
20. Oktober 2024


Clear Channel
olma
olma.ch/tickets
81. Schweizer Messe für
Landwirtschaft und Ernährung
Gastkanton St. Gallen
St. Gallen
10. – 20. Oktober 2024

Goals of last lecture

- Understand that computer code and data are stored as text files
- Understand how we import data from text files
- Learn data structures in R
- Exercise: read financial data from a text file -> today

Structured Data Formats

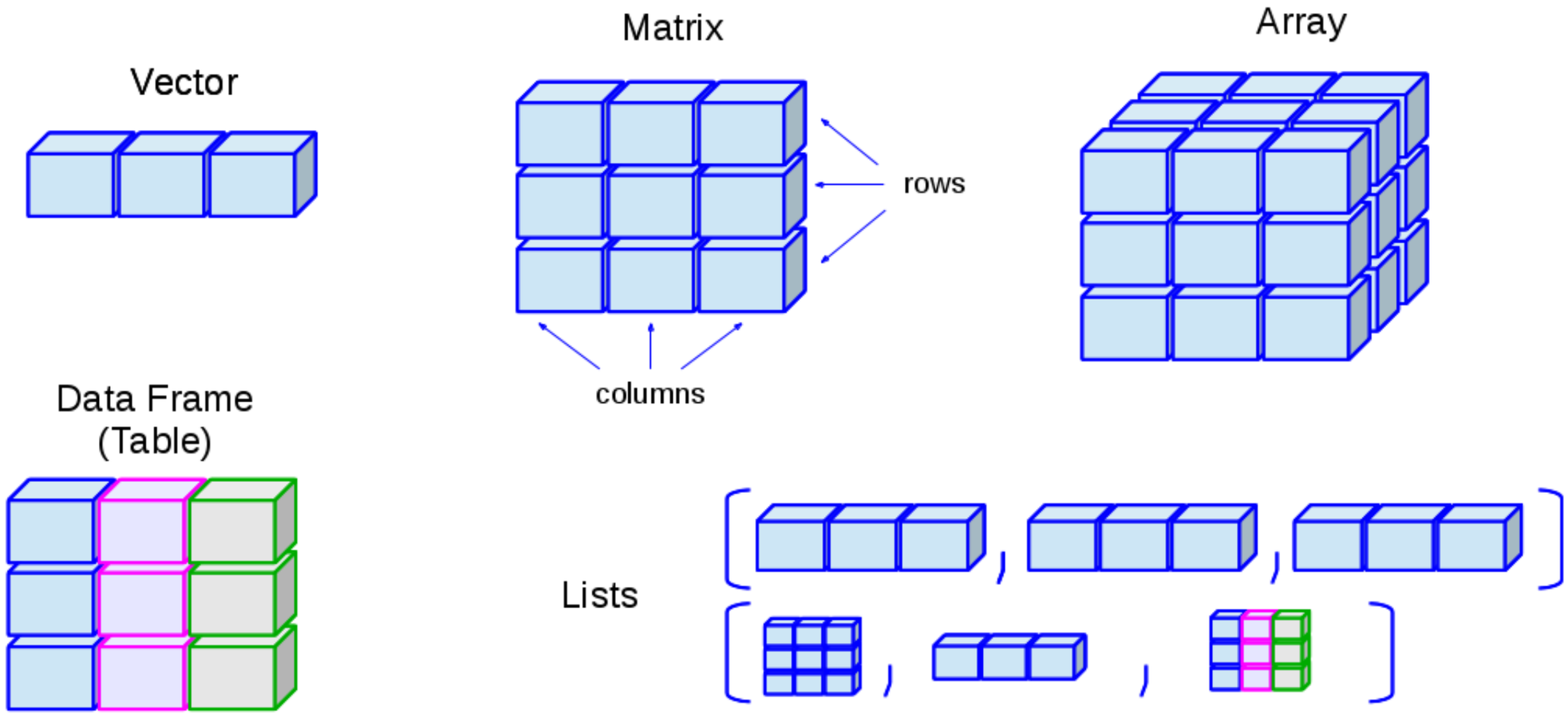
- Still text files, but with standardized structure.
- Special characters define the structure.
- More complex syntax, more complex structures can be represented...
- Example: using a **parser** to work with a csv file.

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. Data **structures** in RAM:
    - Vectors
    - Factors
    - Arrays/Matrices
    - Lists and data frames (R-specific)

Different data types in one figure



Source: http://venus.ifca.unican.es/Rintro/dataStruct.html

Warm-up

Data structure

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

```
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
```

Data structure

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

```
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
```

- 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)

```r
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)
```

Today

Goals of today's lecture

- Understand how we import rectangular data into R
- Be familiar with the way csv parsers work
- Exercise: import, read, and manipulate financial data from a text file

Updates

- Exchange students who wish to take the central exam must contact me per email.
- Next week:
  - normal lecture at 10:15-11:00
  - followed by **guest lecture** by Minna Heim (KOF @ETH) at 11:15

Data in Economics

Data

Data take different structures depending on their purpose.

- Rectangular data
- Non-rectangular 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.

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

- 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).
- Unstructured text data
- Images/Pictures data

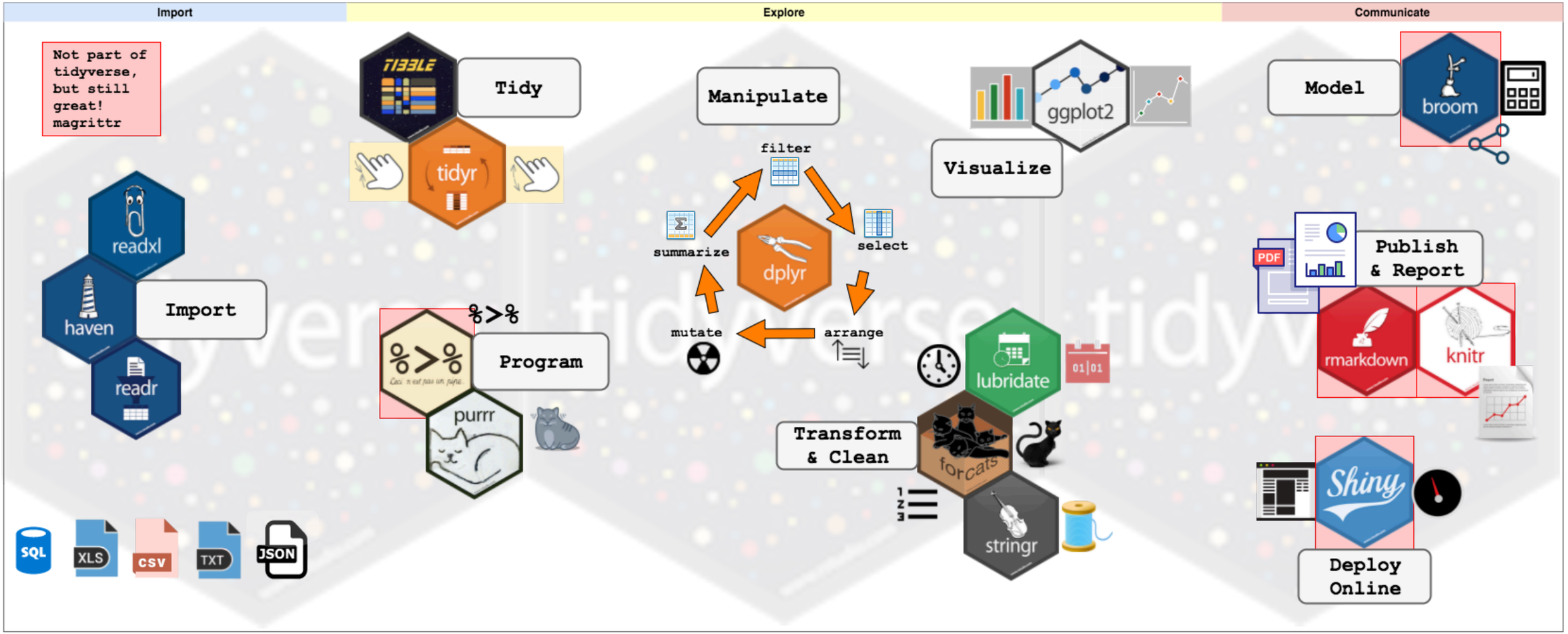
Tidyverse

The tidyverse

“The tidyverse is a collection of open source packages for the R programming language introduced by Hadley Wickham and his team that share an underlying design philosophy, grammar, and data structures” of tidy data.” (Wikipedia)

In this course, we will use tidyverse AND base R.

Tidyverse for data handling in R



Source: https://www.storybench.org/wp-content/uploads/2017/05/tidyverse.png

Rectangular data in R and tidyverse

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

Importing Rectangular Data from Text-Files

Comma Separated Values (CSV)

Consider the `swiss`-dataset 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`

Parsing CSVs in R

- Alternative 1: `read_csv()` (`readr`/`tidyr`-package)
  - Returns a `tibble`.
- Alternative 2: The `data.table`-package and the `fread()` function (handling large datasets).

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.

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')
```

```
# 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

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

Basic usage of `readr` functions

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

```
# 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...

```
# 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')
```

```
# 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"
```

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

```
12:00:00
14:30:00
20:01:00
```

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("1'300'000")
```

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

```
guess_parser("1'300'000", locale = locale(grouping_mark = "'"))
```

```
[1] "number"
```

Check this page for more examples on `guess_parser()`: https://bookdown.org/f_lennert/data-prep_2days/readr.html#rds-and-.rdatafiles

Working with rectangular datasets in R

Loading built-in datasets

Re-load the `swiss` dataset, or load the built-in dataset.

In order to load built-in datasets, simply use the `data()`-function.

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

Famous built-in datasets are `mtcars`, `iris`, `USArrests`, etc. The `swiss` dataset is loaded from the package `datasets`, which can be installed with the command `install.packages("datasets")`. Description of the dataset is here.

Tibbles are a modern version of data frames

Similar!

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

Tibbles are a modern version of data frames

```
# 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
```

Tibbles are a modern version of data frames

	District	Fertility	Agriculture	Examination	Education	Catholic	Infant.Mortality
1	Courtelary	80.2	17.0	15	12	9.96	22.2
2	Delemont	83.1	45.1	6	9	84.84	22.2
3	Franches-Mnt	92.5	39.7	5	5	93.40	20.2
4	Moutier	85.8	36.5	12	7	33.77	20.3
5	Neuveville	76.9	43.5	17	15	5.16	20.6
6	Porrentruy	76.1	35.3	9	7	90.57	26.6
7	Broye	83.8	70.2	16	7	92.85	23.6
8	Glane	92.4	67.8	14	8	97.16	24.9
9	Gruyere	82.4	53.3	12	7	97.67	21.0
10	Sarine	82.9	45.2	16	13	91.38	24.4
11	Veveyse	87.1	64.5	14	6	98.61	24.5
12	Aigle	64.1	62.0	21	12	8.52	16.5
13	Aubonne	66.9	67.5	14	7	2.27	19.1
14	Avenches	68.9	60.7	19	12	4.43	22.7
15	Cossonay	61.7	69.3	22	5	2.82	18.7
16	Echallens	68.3	72.6	18	2	24.20	21.2
17	Grandson	71.7	34.0	17	8	3.30	20.0
18	Lausanne	55.7	19.4	26	28	12.11	20.2
19	La Vallee	54.3	15.2	31	20	2.15	10.8
20	Lavaux	65.1	73.0	19	9	2.84	20.0
21	Morges	65.5	59.8	22	10	5.23	18.0
22	Moudon	65.0	55.1	14	3	4.52	22.4

Manipulations with data frames

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

```
'data.frame':   47 obs. of  7 variables:
 $ District        : chr  "Courtelary" "Delemont" "Franches-Mnt" "Moutier" ...
 $ 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     : num  15 6 5 12 17 9 16 14 12 16 ...
 $ Education       : num  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, n = 5)
```

```
      District Fertility Agriculture Examination Education Catholic Infant.Mortality
1   Courtelary      80.2        17.0          15        12     9.96             22.2
2     Delemont      83.1        45.1           6         9    84.84             22.2
3 Franches-Mnt      92.5        39.7           5         5    93.40             20.2
4      Moutier      85.8        36.5          12         7    33.77             20.3
5   Neuveville      76.9        43.5          17        15     5.16             20.6
```

Work with data.frames

Select columns

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

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

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

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

Select rows

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

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

Other Common Rectangular Formats

Spreadsheets/Excel

Needs the additional R-package: `readxl`. Then we use the package's `read_excel()`-function to import data from an excel-sheet.


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

# load the package
library(readxl)

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

Write an Excel Spreadsheet

- Use `openxlsx` to write, style, and edit .xlsx files.
- Alternative to the xlsx package with no dependency on Java.

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.

```
# Load library
library(haven)

# Read file from SPSS
swiss_imported <- read_spss("data/swiss.sav")
```

Exercise

Open an R script

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

- Start with a meta-section.

```
#######################################################################
# 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, 2024
#######################################################################
```

Structure your script!

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

- Start with a meta-section.
- 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

```
#####################################################################
# 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, 2024
#####################################################################
```

Structure your script!

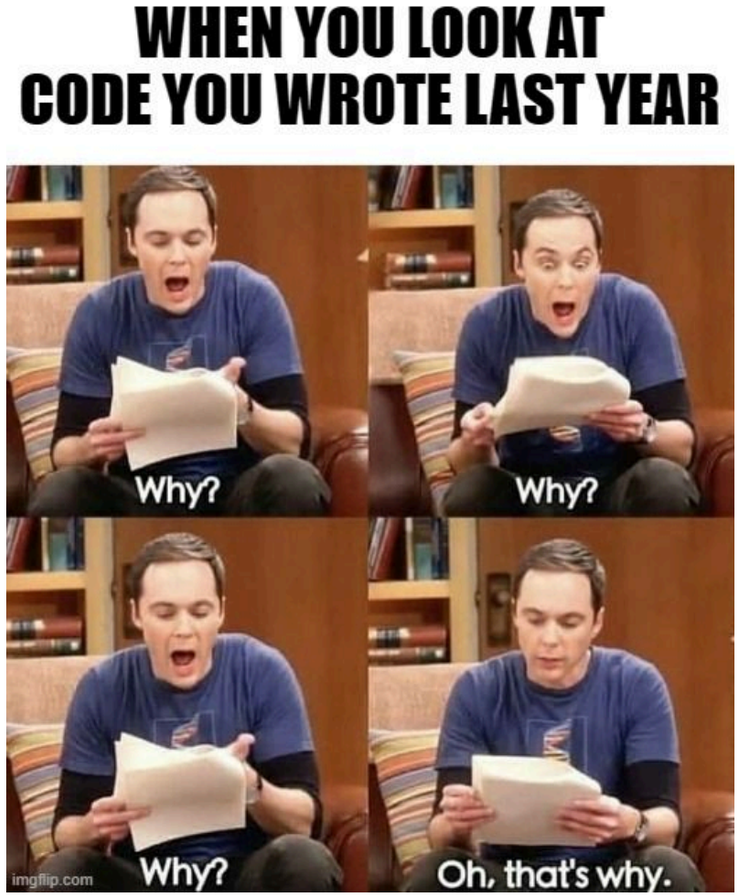
```
#####################################################################
# Data Handling Course: Example Script for Data Gathering and Import
#
# Imports data from ...
# Input: import c to data sources (data comes in ... format)
# Output: cleaned data as CSV
#
# A. Sallin, St. Gallen, 2024
#####################################################################


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

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

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

# End -------------
```



Let’s code!

Appendix

Tibbles vs data frames

- Unlike data frames, tibbles don’t show the entire dataset when you print it
- Tibbles cannot access a column when you provide a partial name of the column, but data frames can.
- When you access only one column of a tibble, it will keep the tibble structure. But when you access one column of a data frame, it will become a vector.
- When assigning a new column to a tibble, the input will not be recycled, which means you have to provide an input of the same length of the other columns. But a data frame will recycle the input.
- Tibbles preserve all the variable types, while data frames have the option to convert string into factor. (In older versions of R, data frames will convert string into factor by default)

Source: https://jtr13.github.io/cc21fall1/tibble-vs.-dataframe.html