Data Handling: Import, Cleaning and Visualisation

Lecture 5:
Rectangular data

Prof. Dr. Ulrich Matter, updated by Dr. Aurélien Sallin

Data formats in economics

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 - XML and JSON (useful for complex/high-dimensional data sets) - HTML (a markup language to define the structure and layout of webpages) - Binary formats - 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)
```

Inspect the data after loading

```
# 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 ...
```

```
# 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
```

The data is provided in a `data.frame` object. This is the most typical object class to work with datasets in R. Almost all of the common functions to import rectangular/spreadsheet-like data into R will return an object of class `data.frame`.

Navigate/work with `data.frames`

There are several ways of selecting rows/columns of a data-frame. For example, to select the `Fertility`-column from the `swiss`-dataset, you can do the following one of the following:

```
# selection of columns
swiss$Fertility # use the $-operator:
```

```
##  [1] 80.2 83.1 92.5 85.8 76.9 76.1 83.8 92.4 82.4 82.9 87.1 64.1 66.9 68.9 61.7 68.3 71.7 55.7 54.3
## [20] 65.1 65.5 65.0 56.6 57.4 72.5 74.2 72.0 60.5 58.3 65.4 75.5 69.3 77.3 70.5 79.4 65.0 92.2 79.3
## [39] 70.4 65.7 72.7 64.4 77.6 67.6 35.0 44.7 42.8
```

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

```
##  [1] 80.2 83.1 92.5 85.8 76.9 76.1 83.8 92.4 82.4 82.9 87.1 64.1 66.9 68.9 61.7 68.3 71.7 55.7 54.3
## [20] 65.1 65.5 65.0 56.6 57.4 72.5 74.2 72.0 60.5 58.3 65.4 75.5 69.3 77.3 70.5 79.4 65.0 92.2 79.3
## [39] 70.4 65.7 72.7 64.4 77.6 67.6 35.0 44.7 42.8
```

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

```
##  [1] 80.2 83.1 92.5 85.8 76.9 76.1 83.8 92.4 82.4 82.9 87.1 64.1 66.9 68.9 61.7 68.3 71.7 55.7 54.3
## [20] 65.1 65.5 65.0 56.6 57.4 72.5 74.2 72.0 60.5 58.3 65.4 75.5 69.3 77.3 70.5 79.4 65.0 92.2 79.3
## [39] 70.4 65.7 72.7 64.4 77.6 67.6 35.0 44.7 42.8
```

```
# selection of rows
# select the first row
swiss[1,]
```

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

```
# select the first three rows
swiss[1:3,] # 1:3 is the same as c(1,2,3)
```

```
##              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
```

```
# select the cell in row 1, column 4
swiss[1,4]
```

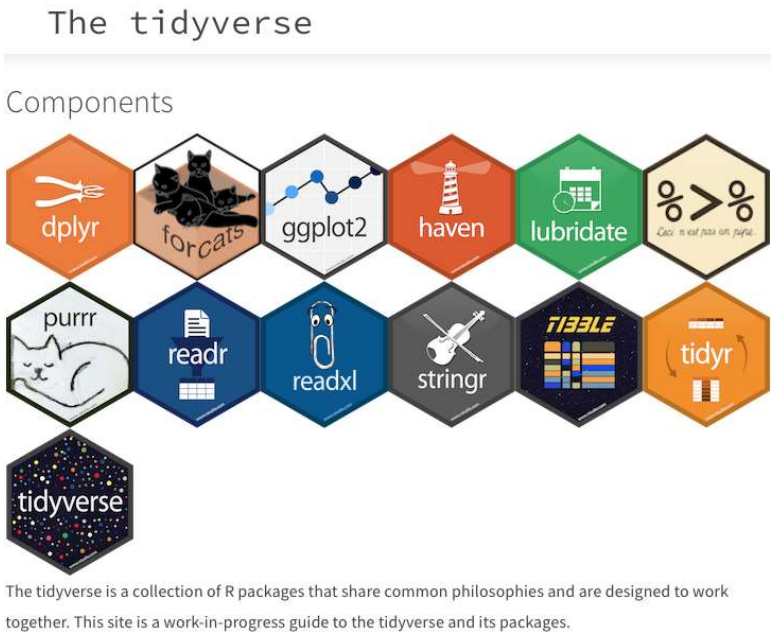
```
## [1] 12
```

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

```
##              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
## Broye             83.8        70.2          16         7    92.85             23.6
## Glane             92.4        67.8          14         8    97.16             24.9
## Gruyere           82.4        53.3          12         7    97.67             21.0
## Sarine            82.9        45.2          16        13    91.38             24.4
## Veveyse           87.1        64.5          14         6    98.61             24.5
## Aigle             64.1        62.0          21        12     8.52             16.5
## Aubonne           66.9        67.5          14         7     2.27             19.1
## Avenches          68.9        60.7          19        12     4.43             22.7
## Cossonay          61.7        69.3          22         5     2.82             18.7
## Echallens         68.3        72.6          18         2    24.20             21.2
## Grandson          71.7        34.0          17         8     3.30             20.0
## Lausanne          55.7        19.4          26        28    12.11             20.2
## La Vallee         54.3        15.2          31        20     2.15             10.8
## Lavaux            65.1        73.0          19         9     2.84             20.0
## Morges            65.5        59.8          22        10     5.23             18.0
## Moudon            65.0        55.1          14         3     4.52             22.4
## Nyone             56.6        50.9          22        12    15.14             16.7
## Orbe              57.4        54.1          20         6     4.20             15.3
## Oron              72.5        71.2          12         1     2.40             21.0
## Payerne           74.2        58.1          14         8     5.23             23.8
## Paysd'enhaut      72.0        63.5           6         3     2.56             18.0
## Rolle             60.5        60.8          16        10     7.72             16.3
## Vevey             58.3        26.8          25        19    18.46             20.9
## Yverdon           65.4        49.5          15         8     6.10             22.5
## Conthey           75.5        85.9           3         2    99.71             15.1
## Entremont         69.3        84.9           7         6    99.68             19.8
## Herens            77.3        89.7           5         2   100.00             18.3
## Martigwy          70.5        78.2          12         6    98.96             19.4
## Monthey           79.4        64.9           7         3    98.22             20.2
## St Maurice        65.0        75.9           9         9    99.06             17.8
## Sierre            92.2        84.6           3         3    99.46             16.3
## Sion              79.3        63.1          13        13    96.83             18.1
## Boudry            70.4        38.4          26        12     5.62             20.3
## La Chauxdfnd      65.7         7.7          29        11    13.79             20.5
## Le Locle          72.7        16.7          22        13    11.22             18.9
## Neuchatel         64.4        17.6          35        32    16.92             23.0
## Val de Ruz        77.6        37.6          15         7     4.97             20.0
## ValdeTravers      67.6        18.7          25         7     8.65             19.5
## Rive Droite       44.7        46.6          16        29    50.43             18.2
## Rive Gauche       42.8        27.7          22        29    58.33             19.3
```

Data.frames vs tibbles

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



Both are similar!

- Tibbles are 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** (https://jtr13.github.io/cc21fall1/tibble-vs.-dataframe.html#tibble-vs.-dataframe%22) for more details.

We can convert a data frame into a tibble using the function `as_tibble()`.

```
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 first two lines of the `swiss`-dataset look like this when stored in a CSV-file:

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

Parsing CSVs in R

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

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

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

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

- Why use `read_csv()` , the `readr` -package?
  - Functions for all common rectangular data formats.
  - Consistent syntax.
  - More robust and faster than similar functions in basic R.

`read_csv()` accepts as a first argument either a character string with a path to a csv file, or a character string directly containing data strucured according to the CSV format. For example, we can parse the first lines of the swiss dataset directly like this:

```
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:

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

```
## Rows: 47 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.
```

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
```

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

Parsing CSVs: data types

- 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*.

Illustration of data type “guessing”

In the following example, we ask `read_csv` to parse a simple dataset in csv-format (the entire dataset is provided in a character string). For a human reader, both columns A and B likely appear to contain information about time. However, in column B the time information is encoded in an inconsistent way, while the observations in column B follow a consistant format.

```
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
```

In the output you see that `read_csv()` recognized the content of column A to be time information and assigned the corresponding data type to it, while column B has been recognized as `character`. Note that this happened without any additional information passed to `read_csv()`. The way this works is that `read_csv()` analyzes for each column what kind of values occur and then decides what data type is most likely fitting to the kind of data contained in each of the columns.

More specifically, 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"
```

This behavior is typical for functions used to import data from CSV into R. Be aware of this default behavior: types are guessed, they can not be determined with a 100% accurracy in many real-life cases. Thus, it is good practice to verify whether the data types assigned to each of the columns of your dataset after import are in line with what you would expect and what you need to further process the data. In practice, this guessing of types is both practical (as it saves time) but it might also be a source of errors if there are some inconsistencies in the raw data. The latter point is of particular relevance when working with large datasets, as it is not trivial to recognize inconsistencies in the raw data.

Other Common Rectangular Formats

Spreadsheets/Excel

Needs additional R-package: `readxl` .

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

Once installed, 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, we can use `read_spss()` for SPSS' `.sav` -format as follows.

```
# 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

Recall our discussion of character encoding. When importing data into R, R per default assumes text files to be encoded in the default encoding of your computer. In practice, it might occur that the file you would like to import into R was written in a different character encoding standard. Below we illustrate such a case with a familiar example. We use `readLines()` to read the content of a text file called `hastamanana.txt` into R.[1]

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

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

From looking at the imported data in R, you notice that something is odd. The imported string contains weird characters.

Recall that text files such as CSVs do not actually contain any information about the standard the file is encoded in. Hence, we need to guess what encoding we are dealing with and then translate the data in the original encoding into our local default encoding.

Guess encoding

`readr` provides a function that does the guessing: `guess_encoding()`.

```
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
```

As you see from the output, `guess_encoding()` povides a number of encoding standards names as well as a corresponding "confidence level", indicating how likely it is that the text was stored in that original encoding.

A reasonable next step is thus to first try a conversion based on the most likely encoding standard and see if this leads to the expected result.

Handling encoding issues

- `inconv()`: convert a character vector `from` one encoding `to` another encoding.
- Use the guessed encoding for the `from` argument (UTF-8 is the standard encoding on most Mac/OSX and Linux machines).

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

```
## [1] "Hasta Mańana!"
```

This looks better but still not optimal. Let us thus try the second most likely encoding and compare the results.

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

```
## [1] "Hasta Mañana!"
```

This looks intuitively better. The take away here is:

1. Recognize whether there is an encoding issue after importing data.
2. Figure out the original encoding: `guess_encoding()`.
3. Convert to the local encoding (you might have to try out which of the suggested encodings delivers the best result): `iconv()`.

Data Gathering Procedure

Organize your data pipeline!

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

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

Here is a template for 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, 2023
########################################################################


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

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


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

Script Template

- 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, or a "set-up" section in which you load the previously installed packages you need for your project.
- In the set-up section, indicate your path as a string. In this way, you only have to enter your path once. This avoids potential mistakes.
- Finally we add sections with the actual code (in the case of a data import script, maybe one section per data source). The title should be as self-explanatory as possible.

References

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

---

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