



Data Handling: Import, Cleaning and Visualisation

Lecture 7:

Data Sources, Data Gathering, Data Import

Prof. Dr. Ulrich Matter

18/11/2021

Welcome back!

Updates

Guest Lecture by Corina Grünenfelder

9 December 2021, “Data Science in Insurance”



Corina Grünenfelder, M.Sc. Mathematics, Actuary SSA
Director, EY

Part II: Data gathering and preparation

Date	Topic
18.11.2021	Data sources, data gathering, data import
25.11.2021	Data preparation and manipulation
25.11.2021	Exercises/Workshop 4: Data import and data preparation/manipulation

Part III: Analysis, visualisation, output

Date	Topic
02.12.2021	Basic statistics and data analysis with R
09.12.2021	Guest Lecture ("Data Science in Insurance")
09.12.2021	Exercises/Workshop 5: Applied data analysis with R
16.12.2021	Visualisation, dynamic documents
21.12.2021	Exercises/Workshop 6: Visualization, dynamic documents
23.12.2021	Summary, Wrap-Up, Q&A, Feedback
23.12.2021	Exam for Exchange Students

“Putting it all Together”

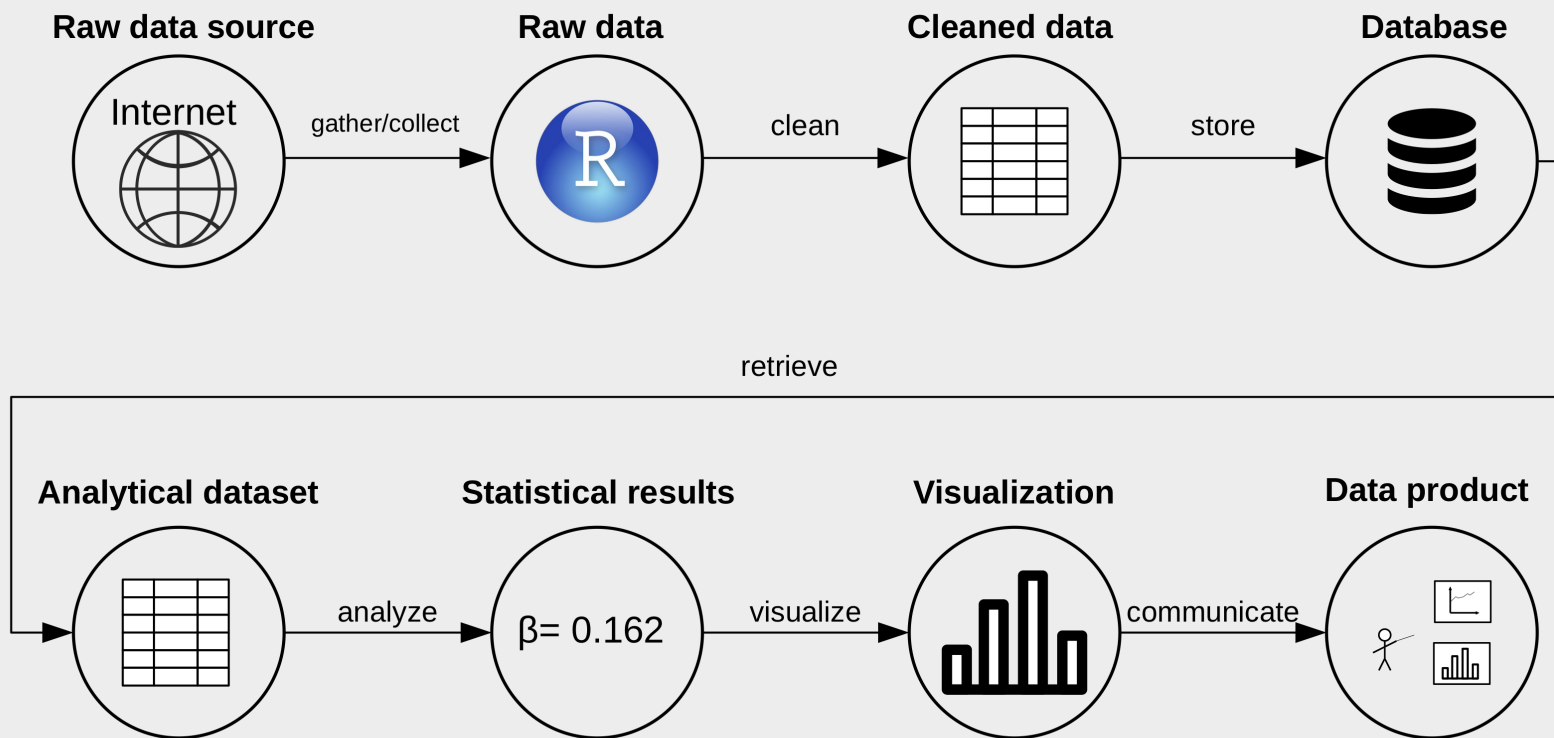
Putting it all together

- You know what **'data'** is...
- You know how digital data is **stored**...
- You know how to write **computer code**...
- You know the basics of **programming in R**...
- You know the basics of **statistics**...
- You have **basic domain knowledge** (in Econ/Business)...

These are the basics to handle data properly!
This is the basis of data science!

We are ready to start the data science journey

Data (science) pipeline



Sources/formats in economics

Sources/formats in economics

- CSV (typical for rectangular/table-like data)
- Variants of CSV (tab-delimited, fix length etc.)
- XML and JSON (useful for complex/high-dimensional data sets)
- HTML (a markup language to define the structure and layout of webpages)
- Unstructured text

Sources/formats in economics

- Excel spreadsheets (`.xls`)
- Formats specific to statistical software packages (SPSS: `.sav`, STATA: `.dat`, etc.)
- Built-in R datasets
- Binary formats

Data Gathering Procedure

Organize your data pipeline!

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

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: links to data sources (data comes in ... format)  
# Output: cleaned data as CSV  
#  
# U. Matter, St. Gallen, 2020  
#####
```

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.
- Start with a 'meta'-section.

Script sections

```
#####  
# Data Handling Course: Example Script for Data Gathering and Import  
#  
# Imports data from ...  
# Input: links to data sources (data comes in ... format)  
# Output: cleaned data as CSV  
#  
# U. Matter, St. Gallen, 2018  
#####  
  
# 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)

```
#####  
# Project XY: Data Gathering and Import  
#  
# This script is the first part of the data pipeline of project XY.  
# It imports data from ...  
# Input: links to data sources (data comes in ... format)  
# Output: cleaned data as CSV  
#  
# U. Matter, St. Gallen, 2018  
#####  
  
# SET UP -----  
# load packages  
library(tidyverse)  
  
# set fix variables  
INPUT_PATH <- "/rawdata"  
OUTPUT_FILE <- "/final_data/datafile.csv"  
  
# IMPORT RAW DATA FROM CSVs -----
```

Loading/Importing 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           20.2
## Neuveville      76.9           43.5           17           15           5.16           20.2
## Porrentruy      76.1           35.3            9            7          90.57           26.2
```

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.Mo  
"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",
"Courtelary",80.2,17,15,12,9.96,22.2')

## # A tibble: 1 x 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")

## Parsed with column specification:
## cols(
##   District = col_character(),
##   Fertility = col_double(),
##   Agriculture = col_double(),
##   Examination = col_double(),
##   Education = col_double(),
##   Catholic = col_double(),
```

Basic usage of readr functions

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

```
swiss
```

```
## # A tibble: 47 x 7
```

```
##   District      Fertility Agriculture Examination Education Catholic Infant.Morta
##   <chr>         <dbl>         <dbl>         <dbl>         <dbl>         <dbl>         <dbl>
## 1 Courtelary    80.2           17           15           12           9.96
## 2 Delemont      83.1           45.1         6            9           84.8
## 3 Franches-Mnt  92.5           39.7         5            5           93.4
## 4 Moutier       85.8           36.5        12            7           33.8
## 5 Neuveville    76.9           43.5        17           15           5.16
## 6 Porrentruy    76.1           35.3         9            7           90.6
## 7 Broye         83.8           70.2        16            7           92.8
## 8 Glane         92.4           67.8        14            8           97.2
## 9 Gruyere       82.4           53.3        12            7           97.7
## 10 Sarine       82.9           45.2        16           13           91.4
## # ... with 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 x 7
##   District      Fertility Agriculture Examination Education Catholic Infant.Morta
##   <chr>         <dbl>         <dbl>         <dbl>         <dbl>         <dbl>         <dbl>
## 1 Courtelary    80.2           17            15            12            9.96
## 2 Delemont      83.1           45.1           6             9            84.8
## 3 Franches-Mnt  92.5           39.7           5             5            93.4
## 4 Moutier       85.8           36.5           12            7            33.8
## 5 Neuveville    76.9           43.5           17            15            5.16
## 6 Porrentruy    76.1           35.3           9             7            90.6
## 7 Broye         83.8           70.2           16            7            92.8
## 8 Glane         92.4           67.8           14            8            97.2
## 9 Gruyere       82.4           53.3           12            7            97.7
## 10 Sarine       82.9           45.2           16            13           91.4
## # ... with 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 x 2  
##   A      B  
##   <time> <chr>  
## 1 12:00 12:00  
## 2 14:30 midnight  
## 3 20:01 noon
```

Parsing CSV-columns

Let's test it!

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

```
## # A tibble: 3 x 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

```
# 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")
```

Importing Web Data Formats

XML in R

```
## {xml_document}  
## <customers>  
## [1] <person>\n  <name>John Doe</name>\n  <orders>\n    <product> x </product>\n  </orders>\n## [2] <person>\n  <name>Peter Pan</name>\n  <orders>\n    <product> a </product>\n  </orders>\n</customers>
```

load packages

```
library(xml2)
```

parse XML, represent XML document as R object

```
xml_doc <- read_xml("data/customers.xml")  
xml_doc
```

XML in R: tree-structure

'customers' is the root-node, 'persons' are it's children:

navigate downwards

```
persons <- xml_children(xml_doc)
persons
```

```
## {xml_nodeset (2)}
```

```
## [1] <person>\n  <name>John Doe</name>\n  <orders>\n    <product> x </product>\n  </orders>\n  </person>\n## [2] <person>\n  <name>Peter Pan</name>\n  <orders>\n    <product> a </product>\n  </orders>\n  </person>
```

XML in R: tree-structure

Navigate sideways and upwards

```
# navigate sideways
```

```
persons[1]
```

```
## {xml_nodeset (1)}
```

```
## [1] <person>\n  <name>John Doe</name>\n  <orders>\n    <product> x </product>\n
```

```
xml_siblings(persons[[1]])
```

```
## {xml_nodeset (1)}
```

```
## [1] <person>\n  <name>Peter Pan</name>\n  <orders>\n    <product> a </product>\n
```

```
# navigate upwards
```

```
xml_parents(persons)
```

```
## {xml_nodeset (1)}
```

```
## [1] <customers>\n  <person>\n    <name>John Doe</name>\n    <orders>\n      <prod
```

XML in R: tree-structure

Extract specific parts of the data:

```
# find data via XPath
customer_names <- xml_find_all(xml_doc, xpath = ".//name")
# extract the data as text
xml_text(customer_names)

## [1] "John Doe" "Peter Pan"
```

JSON in R

```
# load packages  
library(jsonlite)
```

```
# parse the JSON-document shown in the example above  
json_doc <- fromJSON("data/person.json")
```

```
# look at the structure of the document  
str(json_doc)
```

```
## List of 6  
## $ firstName : chr "John"  
## $ lastName  : chr "Smith"  
## $ age       : int 25  
## $ address   :List of 4  
##   ..$ streetAddress: chr "21 2nd Street"  
##   ..$ city          : chr "New York"  
##   ..$ state         : chr "NY"  
##   ..$ postalCode    : chr "10021"  
## $ phoneNumber:'data.frame': 2 obs. of 2 variables:  
##   ..$ type : chr [1:2] "home" "fax"  
##   ..$ number: chr [1:2] "212 555-1234" "646 555-4567"  
## $ gender     :List of 1  
##   ..$ type: chr "male"
```


JSON in R

The nesting structure is represented as a **nested list**:

```
# navigate the nested lists, extract data  
# extract the address part  
json_doc$address
```

```
## $streetAddress  
## [1] "21 2nd Street"  
##  
## $city  
## [1] "New York"  
##  
## $state  
## [1] "NY"  
##  
## $postalCode  
## [1] "10021"
```

```
# extract the gender (type)  
json_doc$gender$type
```

```
## [1] "male"
```

Tutorial (advanced): Importing data from a HTML table

Q&A

References

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