Data Handling: Import, Cleaning and Visualisation

Lecture 5:

Programming with Data

Prof. Dr. Ulrich Matter

17/10/2019

Recap: “Big Data” from the Web

Limitations of rectangular data

- Only **two dimensions**.
  - Observations (rows)
  - Characteristics/variables (columns)
- Hard to represent hierarchical structures.
  - Might introduce redundancies.
  - Machine-readability suffers (standard parsers won’t recognize it).

XML:

```
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</s
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
  <phoneNumber>
    <type>home</type>
    <number>212 555-1234</number>
  </phoneNumber>
  <phoneNumber>
    <type>fax</type>
    <number>646 555-4567</number>
  </phoneNumber>
  <gender>
    <type>male</type>
  </gender>
</person>
```

JSON:

```
{"firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021"
  },
  "phoneNumber": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ],
  "gender": {
    "type": "male"
  }
}
```

XML:

```
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>

</person>
```

JSON:

```
{"firstName": "John",
  "lastName": "Smith",

}
```

Parsing XML in R

The following examples are based on the example code shown above (the two text-files `persons.json` and `persons.xml`)

```
# load packages
library(xml2)

# parse XML, represent XML document as R object
xml_doc <- read_xml("persons.xml")
xml_doc
```

```
## {xml_document}
## <person>
## [1] <firstName>John</firstName>
## [2] <lastName>Smith</lastName>
## [3] <age>25</age>
## [4] <address>\n  <streetAddress>21 2nd Street</streetAddress>\n  <city>New York</
## [5] <phoneNumber>\n  <type>home</type>\n  <number>212 555-1234</number>\n</phoneN
## [6] <phoneNumber>\n  <type>fax</type>\n  <number>646 555-4567</number>\n</phoneNu
## [7] <gender>\n  <type>male</type>\n</gender>
```

Parsing JSON in R

```r
# load packages
library(jsonlite)

# parse the JSON-document shown in the example above
json_doc <- fromJSON("persons.json")

# check the structure
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"
```

```html
<!DOCTYPE html>

<html>
    <head>
        <title>hello, world</title>
    </head>
    <body>
        <h2> hello, world </h2>
    </body>
</html>
```

HTML documents: code and data!

HTML documents/webpages consist of **'semi-structured data'**:

- A webpage can contain a HTML-table (**structured data**)…
- …but likely also contains just raw text (**unstructured data**).

Characteristics of HTML

1. **Annotate/'mark up'** data/text (with tags)
   - Defines **structure** and hierarchy
   - Defines content (pictures, media)
2. **Nesting** principle
   - `head` and `body` are nested within the `html` document
   - Within the `head`, we define the `title`, etc.
3. Expresses what is what in a document.
   - Doesn't explicitly 'tell' the computer what to do
   - HTML is a markup language, not a programming language.

HTML document as a ‘tree’

HTML (DOM) tree diagram (by Lubaochuan 2014, licensed under the [Creative Commons Attribution-Share Alike 4.0 International](#) license).

Parsing a Webpage with R

```
# install package if not yet installed
# install.packages("rvest")

# load the package
library(rvest)
```

```
# parse the webpage, show the content
swiss_econ_parsed <- read_html("https://en.wikipedia.org/wiki/Economy_of_Switzerlan
swiss_econ_parsed
```

```
## {html_document}
## <html class="client-nojs" lang="en" dir="ltr">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">\
## [2] <body class="mediawiki ltr sitedir-ltr mw-hide-empty-elt ns-0 ns-subject mw-
```

Parsing a Webpage with R

Now we can easily separate the data/text from the html code. For example, we can extract the HTML table containing the data we are interested in as a `data.frames`.

```
tab_node <- html_node(swiss_econ_parsed, xpath = "//*[@id='mw-content-text']/div/tal
tab <- html_table(tab_node)
tab
```

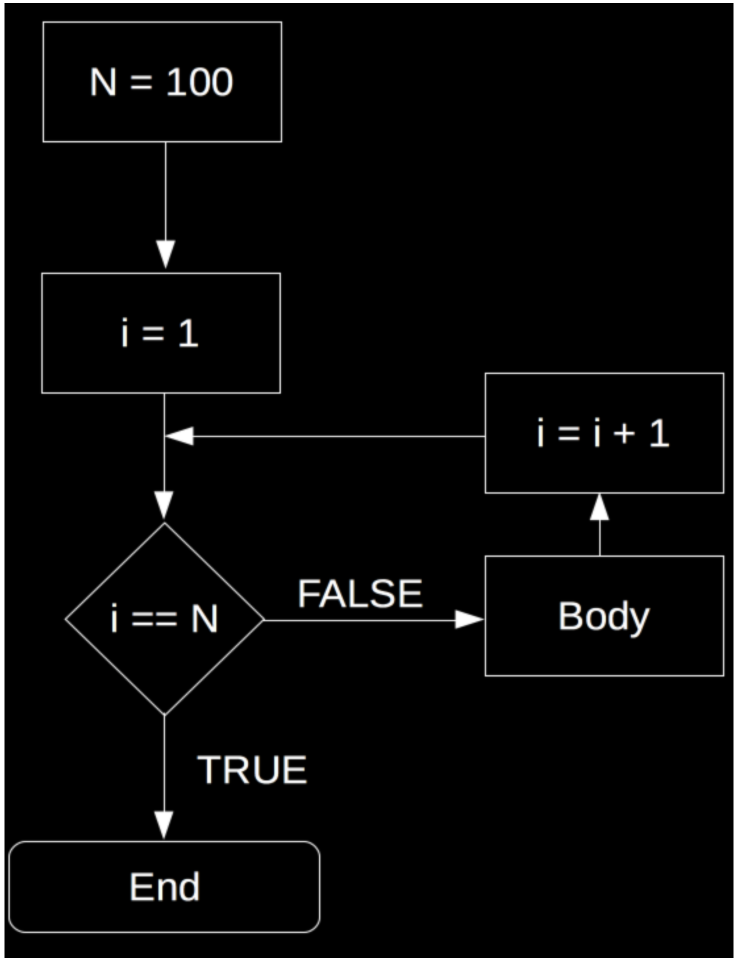
```
##    Year GDP (billions of CHF) US Dollar Exchange
## 1  1980                   184        1.67 Francs
## 2  1985                   244        2.43 Francs
## 3  1990                   331        1.38 Francs
## 4  1995                   374        1.18 Francs
## 5  2000                   422        1.68 Francs
## 6  2005                   464        1.24 Francs
## 7  2006                   491        1.25 Francs
## 8  2007                   521        1.20 Francs
## 9  2008                   547        1.08 Francs
## 10 2009                   535        1.09 Francs
## 11 2010                   546        1.04 Francs
## 12 2011                   659        0.89 Francs
```

Basic Programming Concepts

Loops

- Repeatedly execute a sequence of commands.
- Known or unknown number of iterations.
- Types: ‘for-loop’ and ‘while-loop’.
    - ‘for-loop’: number of iterations typically known.
    - ’while-loop: number of iterations typically not known.

for-loop



for-loop in R

```
# number of iterations
n <- 100
# start loop
for (i in 1:n) {

     # BODY
}
```

for-loop in R

```
# vector to be summed up
numbers <- c(1,2,3,4,5)
# initiate total
total_sum <- 0
# number of iterations
n <- length(numbers)
# start loop
for (i in 1:n) {
     total_sum <- total_sum + numbers[i]
}
```

Nested for-loops

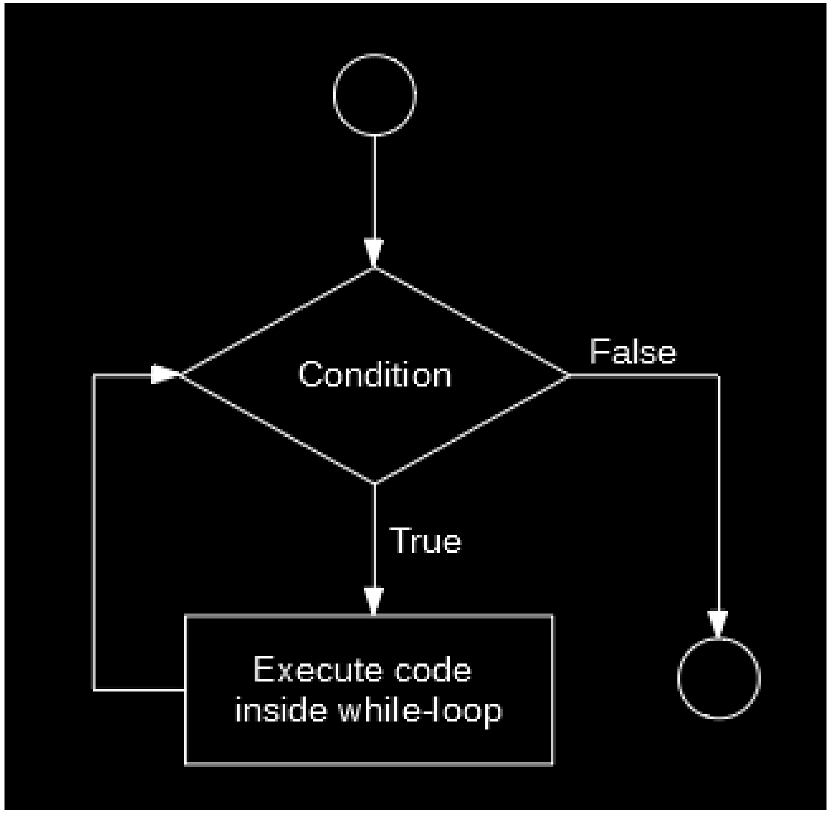
```
# matrix to be summed up
numbers_matrix <- matrix(1:20, ncol = 4)
numbers_matrix
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    2    7   12   17
## [3,]    3    8   13   18
## [4,]    4    9   14   19
## [5,]    5   10   15   20
```

Nested for-loops

```
# number of iterations for outer loop
m <- ncol(numbers_matrix)
# number of iterations for inner loop
n <- nrow(numbers_matrix)
# start outer loop (loop over columns of matrix)
for (j in 1:m) {
     # start inner loop
     # initiate total
     total_sum <- 0
     for (i in 1:n) {
          total_sum <- total_sum + numbers_matrix[i, j]
          }
     print(total_sum)
     }
```

while-loop



while-loop in R

```
# initiate variable for logical statement
x <- 1
# start loop
while (x == 1) {

     # BODY
}
```

`while`-loop in R

```r
# initiate starting value
total <- 0
# start loop
while (total <= 20) {
     total <- total + 1.12
}
```

Booleans and logical statements

```
2+2 == 4
```

```
## [1] TRUE
```

```
3+3 == 7
```

```
## [1] FALSE
```

```
4!=7
```

```
## [1] TRUE
```

Booleans and logical statements

```r
condition <- TRUE

if (condition) {
     print("This is true!")
} else {
     print("This is false!")
}
```

```
## [1] "This is true!"
```

Booleans and logical statements

```r
condition <- FALSE

if (condition) {
     print("This is true!")
} else {
     print("This is false!")
}
```

```
## [1] "This is false!"
```

R functions

- $f: X \rightarrow Y$
- ‘Take a variable/parameter value X as input and provide value Y as output’
- For example, $2 \times X = Y$.
- R functions take ‘parameter values’ as input, process those values according to a predefined program, and ‘return’ the results.

R functions

- Many functions are provided with R.
- More can be loaded by installing and loading packages.

```
# install a package
install.packages("<PACKAGE NAME>")
# load a package
library(<PACKAGE NAME>)
```

Tutorial: A Function to Compute the Mean

Preparation

1. Open a new R-script and save it in your `code`-directory as `my_mean.R`.
2. In the first few lines, use `#` to write some comments describing what this script is about.

Preparation

```
#####################################
# Mean Function:
# Computes the mean, given a
# numeric vector.
```

Preparation

1. Open a new R-script and save it in your `code`-directory as `my_mean.R`.
2. In the first few lines, use `#` to write some comments describing what this script is about.
3. Also in the comment section, describe the function argument (input) and the return value (output)

Preparation

```
#####################################
# Mean Function:
# Computes the mean, given a
# numeric vector.
# x, a numeric vector
# returns the arithmetic mean of x (a numeric scalar)
```

Preparation

1. Open a new R-script and save it in your `code`-directory as `my_mean.R`.
2. In the first few lines, use `#` to write some comments describing what this script is about.
3. Also in the comment section, describe the function argument (input) and the return value (output)
4. Add an example (with comments), illustrating how the function is supposed to work.

Preparation

```
# Example:
# a simlpe numeric vector, for which we want to compute the mean
# a <- c(5.5, 7.5)
# desired functionality and output:
# my_mean(a)
# 6.5
```

1. Know the concepts/context!

- Programming a function in R means telling R how to transform a given input (x).
- Before we think about how we can express this transformation in the R language, we should be sure that we understand the transformation per se.

1. Know the concepts/context!

- Programming a function in R means telling R how to transform a given input (x).
- Before we think about how we can express this transformation in the R language, we should be sure that we understand the transformation per se.

Here, we should be aware of how the mean is defined:

$$\bar{x} = \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right) = \frac{x_1+x_2+\cdots+x_n}{n}.$$

2. Split the problem into several smaller problems

From looking at the mathematical definition of the mean ($\bar{x}$), we recognize that there are two main components to computing the mean:

- $\sum_{i=1}^{n} x_i$: the **sum** of all the elements in vector x
- and n, the **number of elements** in vector x.

3. Address each problem step-by-step

In R, there are two built-in functions that deliver exactly these two components:

- **`sum()`** returns the sum of all the values in i ts arguments (i.e., if `x` is a numeric vector, `sum(x)` returns the sum of all elements in `x`).
- **`length()`** returns the total number of elements in a given vector (the vector's 'length').

4. Putting the pieces together

With the following short line of code we thus get the mean of the elements in vector `a`.

```
sum(a)/length(a)
```

5. Define the function

All that is left to do is to pack all this into the function body of our newly defined `my_mean()` function:

```
# define our own function to compute the mean, given a numeric vector
my_mean <- function(x) {
     x_bar <- sum(x) / length(x)
     return(x_bar)
}
```

6. Test it with the pre-defined example

```
# test it
a <- c(5.5, 7.5)
my_mean(a)
```

```
## [1] 6.5
```

6. Test it with other implementations

Here, compare it with the built-in `mean()` function:

```
b <- c(4,5,2,5,5,7)
my_mean(b) # our own implementation
```

```
## [1] 4.666667
```

```
mean(b) # the built_in function
```

```
## [1] 4.666667
```

Q&A

References