Data Handling: Import, Cleaning and Visualisation

Lecture 6:

Non-rectangular data

Dr. Aurélien Sallin

Updates

Updates

- Exam for exchange students: 21.12.2023 at 16:15 in room 10-013.
- Don’t forget the exercise from lecture 5 💣

Recap

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

Warm-up


DATA SCIENCE WARM-UP CU!P
WARM-UP CHALLLENGE
DATA DEADLIFTS
DATAS DEAGLENICS
PERPERCY
DATA
DATA DEASTHICS
CALISTHENICS
CODE
ANAYSES CALISTIENICS
SIKNPEES
CODE CALSTENICS
DODMG AGIELITY
CAMTHSENICS
5 MINUTES TO FITNESS!

Coercion of boolean values

This question checks your understanding of the coercion of boolean values.

What is the output of the following code?

```
my_vector <- c(1, 0, 3, -1)
as.numeric(my_vector > 0)
```

- **c("TRUE", "FALSE", "TRUE", "FALSE")**
- **c(TRUE, FALSE, TRUE, FALSE)**
- **c(1,0,1,0)**
- **c(0,1,0,1)**

Data frames operations

Be the data frame

```r
dataCHframe <- data.frame(
  "City" = c("St.Gallen", "Lausanne", "Zürich"),
  "PartyLeft" = c(35, 45, 55),
  "PartyRight" = c(40, 35, 30)
)
```

Which of these statements are TRUE?

- **`dataCHframe$PartyCenter <- c(25, 20, 15)`** creates a new variable called "PartyCenter"
- **`dim(dataCHframe[, dataCHframe$PartyLeft > 40])`** returns the same as **`dim(dataCHframe[, c(2,3)])`**
- **`dim(dataCHframe[dataCHframe$PartyLeft > 40 | dataCHframe$PartyLeft < 40, ])`** returns **`c(3,3)`**
- **`dataCHframe`** is a list consisting of one named character vector and two named integer vectors.

Working with csv files

You want to import a file using `read_delim()`. Describe what `read_delim()` does under the hood. What should be added to this command in order for it to work?

Tibbles

Be the code

```r
df <- data.frame(a = c(1,2,3,4),
                 b = c("au", "de", "ch", "li"))
```

Are these statements TRUE or FALSE?

- **`mean(df$a) == 2.5`**
- **`typeof(as.matrix(df)[,1])`** is **`numeric`** (or **`double`**)

Statement for illustrative purposes (just for you to see it once, not that I expect you to learn this)

- **`as_tibble(df)[1:2,1]`** contains the same information as **`df[1:2, 1]`** but the formatting is different.

```
as_tibble(df)[1:2,1]
```

```
## # A tibble: 2 × 1
##       a
##   <dbl>
## 1     1
## 2     2
```

```
df[1:2, 1]
```

```
## [1] 1 2
```

Web Data, Complex Data Structures

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

A rectangular data set

father	mother	name	age	gender
		John	33	male
		Julia	32	female
John	Julia	Jack	6	male
John	Julia	Jill	4	female
John	Julia	John jnr	2	male
		David	45	male
		Debbie	42	female
David	Debbie	Donald	16	male
David	Debbie	Dianne	12	female

What is the data about?

A rectangular data set

father	mother	name	age	gender
		John	33	male
		Julia	32	female
John	Julia	Jack	6	male
John	Julia	Jill	4	female
John	Julia	John jnr	2	male
		David	45	male
		Debbie	42	female
David	Debbie	Donald	16	male
David	Debbie	Dianne	12	female

Which observations belong together?

A rectangular data set

father	mother	name	age	gender
		John	33	male
		Julia	32	female
John	Julia	Jack	6	male
John	Julia	Jill	4	female
John	Julia	John jnr	2	male
		David	45	male
		Debbie	42	female
David	Debbie	Donald	16	male
David	Debbie	Dianne	12	female

Can a parser understand which observations belong together?

Deciphering XML

Revisiting Air quality data (exercise sheet 2)

```
unique_id,indicator_id,name,measure,measure_info,geo_type_name,
  geo_join_id,geo_place_name,time_period,start_date,data_value
216498,386,Ozone (O3),Mean,ppb,CD,
  313,Coney Island (CD13),Summer 2013,2013-06-01T00:00:00,34.64
216499,386,Ozone (O3),Mean,ppb,CD,
  313,Coney Island (CD13),Summer 2014,2014-06-01T00:00:00,33.22
219969,386,Ozone (O3),Mean,ppb,Borough,
  1,Bronx,Summer 2013,2013-06-01T00:00:00,31.25
```

Revisiting COVID-19 (in XML!)

What features does the format have? What is it's logic/syntax?

```
<row>
<unique_id>216498</unique_id>
<indicator_id>386</indicator_id>
<name>Ozone (O3)</name>
<measure>Mean</measure>
<measure_info>ppb</measure_info>
<geo_type_name>CD</geo_type_name>
<geo_join_id>313</geo_join_id
><geo_place_name>Coney Island (CD13)</geo_place_name>
<time_period>Summer 2013</time_period>
<start_date>2013-06-01T00:00:00</start_date>
<data_value>34.64</data_value>
</row>

<unique_id>216499</unique_id>
<indicator_id>386</indicator_id>
...

<\row>
```

XML syntax

The actual content we know from the csv-type example above is nested between the '**row**'-tags:

```
  <row>
...
  </row>
```

XML as a tree structure

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

XML syntax: Temperature Data example

There are two principal ways to link variable names to values.

```
<variable>Monthly Surface Clear-sky Temperature (ISCCP) (Celsius)</variable>
<filename>ISCCPMonthly_avg.nc</filename>
<filepath>/usr/local/fer_data/data/</filepath>
<badflag>-1.E+34</badflag>
<subset>48 points (TIME)</subset>
<longitude>123.8W(-123.8)</longitude>
<latitude>48.8S</latitude>
<case date="16-JAN-1994" temperature="9.200012" />
<case date="16-FEB-1994" temperature="10.70001" />
<case date="16-MAR-1994" temperature="7.5" />
<case date="16-APR-1994" temperature="8.100006" />
```

XML syntax

1. Define opening and closing XML-tags with the variable name and surround the value with them, such as in `<filename>ISCCPMonthly_avg.nc</filename>`.
2. Encapsulate the values within one tag by defining tag-attributes such as in `<case date="16-JAN-1994" temperature="9.200012" />`.

XML syntax

Attributes-based:

```
<case date="16-JAN-1994" temperature="9.200012" />
<case date="16-FEB-1994" temperature="10.70001" />
<case date="16-MAR-1994" temperature="7.5" />
<case date="16-APR-1994" temperature="8.100006" />
```

XML syntax

Tag-based:

```
<cases>
  <case>
    <date>16-JAN-1994<date/>
    <temperature>9.200012<temperature/>
  <case/>
  <case>
    <date>16-FEB-1994<date/>
    <temperature>10.70001<temperature/>
  <case/>
  <case>
    <date>16-MAR-1994<date/>
    <temperature>7.5<temperature/>
  <case/>
  <case>
    <date>16-APR-1994<date/>
    <temperature>8.100006<temperature/>
  <case/>
<cases/>
```

Insights: CSV vs. XML

- Represent much more **complex (multi-dimensional)** data in XML-files than what is possible in CSVs.
- Self-explanatory syntax: **machine-readable and human-readable**.
- Tags are part of the syntax, give both structure and name variables.

XML in R

```
<?xml version="1.0" encoding="UTF-8"?>
 <customers>
  <person>
    <name>Michael Scott</name>
     <orders>
     <product> x </product>
     <product> y </product>
     </orders>
  </person>
  <person>
    <name>Dwight Schrutte</name>
     <orders>
     <product> a </product>
     <product> x </product>
     </orders>
  </person>
 </customers>
```

XML in R

```
# load packages
library(xml2)

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

```
## {xml_document}
## <customers>
## [1] <person>\n  <name>Michael Scott</name>\n  <orders>\n    <product> x </product>\n    <produ
## [2] <person>\n  <name>Dwight Schrutte</name>\n  <orders>\n    <product> a </product>\n    <pro
```

- Note: in the exercises, you used the library **`XML`** instead of **`xml2`**. Those are equivalent, but **`xml2`** is updated and faster.

XML in R: tree-structure

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

```
# navigate downwards
persons <- xml_children(xml_doc)
persons
```

```
## {xml_nodeset (2)}
## [1] <person>\n  <name>Michael Scott</name>\n  <orders>\n    <product> x </product>\n    <produ
## [2] <person>\n  <name>Dwight Schrutte</name>\n  <orders>\n    <product> a </product>\n    <pro
```

XML in R: tree-structure

Navigate sidewards and upwards

```
# navigate sidewards
persons[1]
```

```
## {xml_nodeset (1)}
## [1] <person>\n  <name>Michael Scott</name>\n  <orders>\n    <product> x </product>\n    <produ
```

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

```
## {xml_nodeset (1)}
## [1] <person>\n  <name>Dwight Schrutte</name>\n  <orders>\n    <product> a </product>\n    <pro
```

```
# navigate upwards
xml_parents(persons)
```

```
## {xml_nodeset (1)}
## [1] <customers>\n  <person>\n    <name>Michael Scott</name>\n    <orders>\n      <product> x <
```

XML in R: tree-structure

Extract specific parts of the data:

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

```
## {xml_nodeset (2)}
## [1] <name>Michael Scott</name>
## [2] <name>Dwight Schrutte</name>
```

```
# extract the data as text
xml_text(customer_names)
```

```
## [1] "Michael Scott"   "Dwight Schrutte"
```

Deciphering JSON

JSON syntax

- Key difference to XML: no tags, but **attribute-value pairs**.
- A substitute for XML (often encountered in similar usage domains).

XML:

```xml
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <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>
```

XML:

```xml
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetA
    <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:

```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:

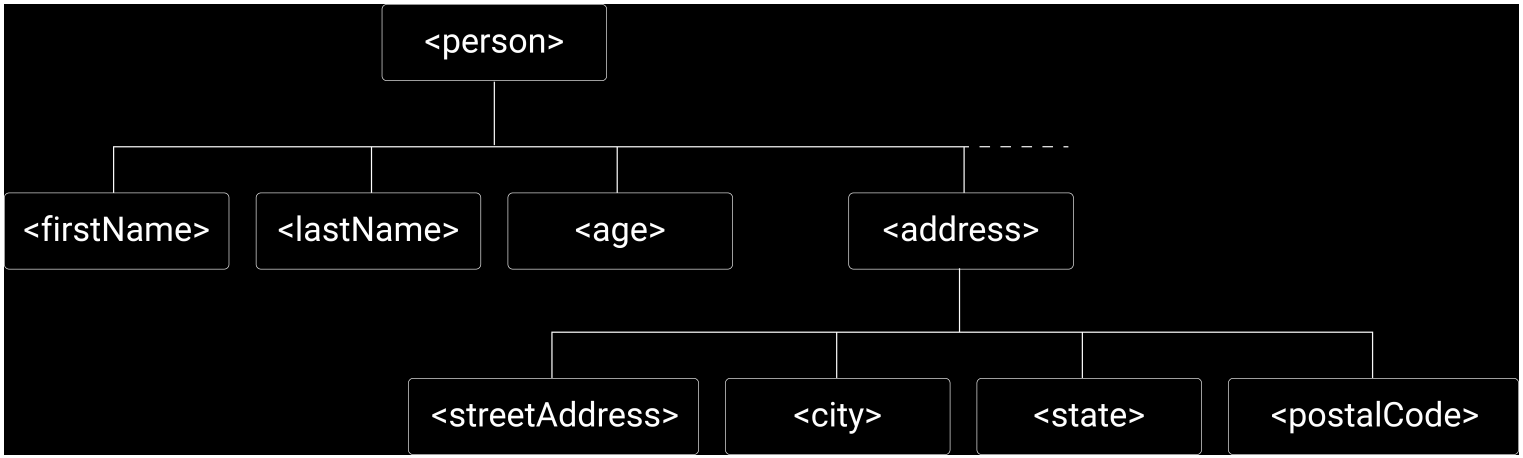
```xml
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>

</person>
```

JSON:

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

}
```


<person>
<firstName>
<lastName>
<age>
<address>
<streetAddress>
<city>
<state>
<postalCode>

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"
```

HTML: Websites

HTML: Code to build webpages

HyperText Markup Language (HTML), designed to be read by a web browser.

HTML documents: code and data!

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

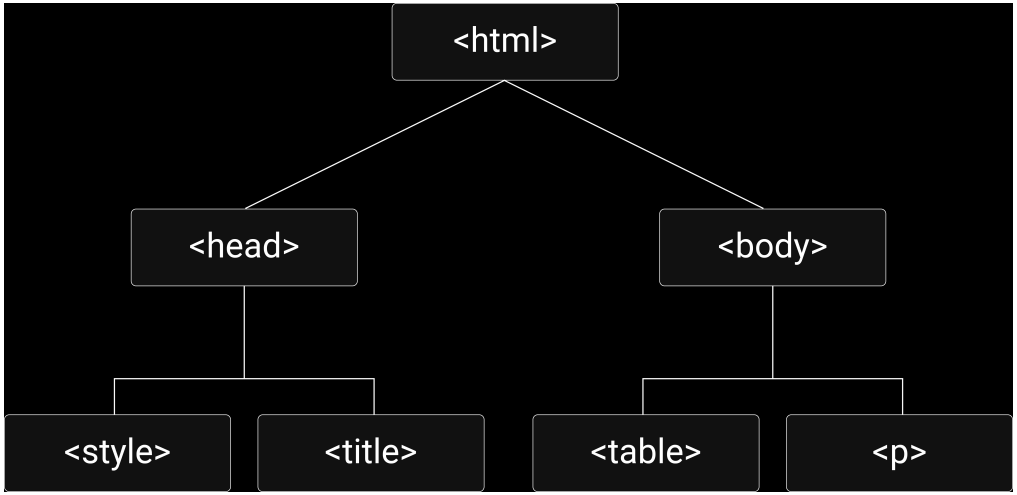
- A webpage can contain a HTML-table (**structured data**)…
- …but likely also contains just raw text (**unstructured data**).

```
<!DOCTYPE html>

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

Similarities to other formats?

HTML document as a ‘tree’



Two ways to read a webpage into R

In this example, we look at Wikipedia's Economy of Switzerland page.

Year	GDP (billions of CHF)	US Dollar Exchange
1980	184	1.67 Francs
1985	244	2.43 Francs
1990	331	1.38 Francs
1995	374	1.18 Francs
2000	422	1.68 Francs
2005	464	1.24 Francs
2006	491	1.25 Francs
2007	521	1.20 Francs
2008	547	1.08 Francs
2009	535	1.09 Francs
2010	546	1.04 Francs
2011	659	0.89 Francs
2012	632	0.94 Francs
2013	635	0.93 Francs
2014	644	0.92 Francs
2015	646	0.96 Francs
2016	659	0.98 Francs
2017	668	1.01 Francs
2018	694	1.00 Francs

Tutorial (advanced): Importing data from a HTML table

-> Exercise session this afternoon!

Text as Data and Image Data

Text as Data and Image Data

- Extract text from historical documents (scan, use OCR)
- Use machine learning to label text (too costly to do manually)
- Extract information from maps

Text as Data

Text is unstructured data. Text analysis and feature extraction is the basis for new genAI models!

-> check the code example on Canvas.

Image Data

- Check the code example on Canvas.
- Good example on how arrays are used in **R**.

Happy break!!!

Realistic image of a beach scene during sunset. A computer sits open on the sand displaying R software. Next to the computer, there are two chilled beers.

Q&A

References