Data Handling: Import, Cleaning and Visualisation

Lecture 6, part 2:
Non-rectangular data

Dr. Aurélien Sallin

2024-11-14

Welcome back!

Updates

- **Exam for exchange students** 🎁 : 19.12.2024 at 16:15 in room 01-207.
- The reading list has been updated (R4DS Second edition)
- Walk-ins for the digital exam:
  - **04.12.2024** (10:30-14:30, Wintergarten 11:111 Square)
  - **17.12.2024** (10:30-14:30, Wintergarten 11:1111 Square)
- The *mock exam* is online 💣

Recap

Rectangular data

- Import data from text files, csv, tsv, etc.
- Tibbles, data frames in R

Non-rectangular data

- Hierarchical data (xml, html, json)
- Unstructured text data and images/pictures data

Non-rectangular data for economic research

- csv, json, API for economic research
- Link to guest lecture: here

Today (first period)

Goals

- Understand non-rectangular data: xml, json, and html
- Be familiar with the way we parse these data into R
- Scrape a webpage following its html structure

Non-rectangular data structures

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

A hierarchical 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
```

XML

XML = eXtensible Markup Language

Recap (from last time):

- **Human-readable**: XML files can be easily read and understood without needing special tools.
- **Self-descriptive**: XML tags describe the data they enclose.
- **Hierarchical structure**: Data is organized in a tree-like structure, making it easy to represent complex relationships.

▶ See a XML representation of a Word document on the Wikipedia page on Microsoft Office XML format

XML markup is structured by special characters and tags

- A predefined set of special characters (here primarily <, >, and /) give the data structure.

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
  <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>

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

XML markup is structured by special characters and tags

A xml element:

```
<latitude>48.8S</latitude>
```

- start tag: **<latitude>**48.8S</latitude>
- data value: <latitude>**48.8S**</latitude>
- end tag: <latitude>48.8S**</latitude>**

Originally, XML element values were limited to a single type: strings. It is however possible to write a schema that assigns data types to the elements of a XML document (referred to as XSD or XML Schema Definition). See Mundell, chapter 5.5.3 and 5.54 for more details.

XML markup has a nested syntax… similar to a family-tree structure 🌳

Tags can be nested, which allows for the definition of hierarchical structures. Here, the “root-content” is nested between the ‘`root``’-tags:

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

There are two principal ways to link variable names to values in XML

Tag/element-defined

Define opening and closing XML-tags with the variable name and surround the value with them:

```
<filename>ISCCPMonthly_avg.nc</filename>
```

Attribute-defined

Encapsulate the values within one tag by defining tag-attributes:

```
<case date="16-JAN-1994" temperature="9.200012" />
```

There are two principal ways to link variable names to values in XML

Tag/element-defined

```
<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/>
```

Attribute-defined

```
<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" />
```

Advantages of XML over csv

- **More complex (multi-dimensional)** in XML-files than what is possible in CSVs.
- **Self-explanatory syntax**: machine-readable and human-readable.
- **Structured**: give both structure and name variables.

Potential drawback of XML: inefficient storage

- Tags are part of the syntax, thus, part of the actual file.
  - Tags (variable labels) are repeated again and again!
  - CSV: variable labels are mentioned once.
- Potential solution: **data compression** (e.g., zip).
- If the data is actually two-dimensional, a CSV is more practical.

Parsing XML

XML in R with *The Office*

```xml
<?xml version="1.0" encoding="UTF-8"?>
<company_dundermifflin>
  <person id="1">
    <name>Michael Scott</name>
    <position>Regional Manager</position>
    <location branch="Scranton"/>
  </person>
  <person id="2">
    <name>Dwight Schrutte</name>
    <position>Assistant (to the) Regional Manager</position>
    <location branch="Scranton"/>
    <orders>
      <sales>
        <units>10</units>
        <product>paper A4</product>
      </sales>
    </orders>
  </person>
  <person id="3">
    <name>Jim Halpert</name>
    <position>Sales Representative</position>
    <location branch="Scranton"/>
    <orders>
```

- The root is `company_dundermifflin`
- `company_dundermifflin` is the root-node, `persons` are its children
- Three persons: Michael Scott, Dwight Schrutte, Jim Halpert
- For Dwight and Jim, we have paper sales orders.

Load a XML file in R

```
# load packages
library(xml2) # updated and faster than library `XML`

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

```
{xml_document}
<company_dundermifflin>
[1] <person id="1">\n  <name>Michael Scott</name>\n  <position>Regional Manager</position>\n  <lo ...
[2] <person id="2">\n  <name>Dwight Schrutte</name>\n  <position>Assistant (to the) Regional Mana ...
[3] <person id="3">\n  <name>Jim Halpert</name>\n  <position>Sales Representative</position>\n  < ...
```

Identify the root-node and the children

🔽 Navigate downwards (find the children or the first child)

```
persons <- xml_children(xml_doc)
xml_child(xml_children(xml_doc), 1)
```

▶ Show output

◀️ ▶️ Navigate sidewards

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

▶ Show output

🔼 Navigate upwards: find all the parents (all levels) or the parent (one level)

```
sales <- xml_children(xml_children(xml_children(xml_doc)))
xml_parents(sales)
xml_parent(sales)
```

▶ Show output

Extract specific parts of the data using XPath

XPath is a language specifically designed for navigating and querying XML documents

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

```
{xml_nodeset (3)}
[1] <sales>\n  <units>10</units>\n  <product>paper A4</product>\n</sales>
[2] <sales>\n  <units>20</units>\n  <product>paper A4</product>\n</sales>
[3] <sales>\n  <units>5</units>\n  <product>paper A3</product>\n</sales>
```

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

```
[1] "10paper A4" "20paper A4" "5paper A3"
```

```
position <- xml_find_all(xml_doc, xpath = ".//position")
position
```

```
{xml_nodeset (3)}
[1] <position>Regional Manager</position>
[2] <position>Assistant (to the) Regional Manager</position>
[3] <position>Sales Representative</position>
```

Exercise: Create a data frame with sales

Try to extract the sales data from the xml file at home.

- What is the difference between // and / in the XPath?
- This exercise is meant as a challenge for the most motivated 💪

▶ Show the code

```
             name units  product
1            <NA>    NA     <NA>
2 Dwight Schrutte    10 paper A4
3     Jim Halpert    20 paper A4
4     Jim Halpert     5 paper A3
```

Deciphering JSON

JSON = JavaScript Object Notation

- Text-based format used for representing structured and based on JavaScript object syntax.
- Used for data exchange between a server and a web application
- Used in most APIs.

JSON syntax

- Key difference to XML: no tags, but **attribute-value pairs**.
- JSON is built on two structures:
  - **Object**: a collection of properties, i.e. key/value pairs `{"key": value}`
  - **Array**: an ordered list of values `[1, 2, 3]`
- A substitute for XML (often encountered in similar usage domains).

Object properties are accessed using keys. Keys must be strings, and values must be a valid JSON data type (string, number, object, array, boolean or null). Arrays can contain objects.

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>
```

JSON:

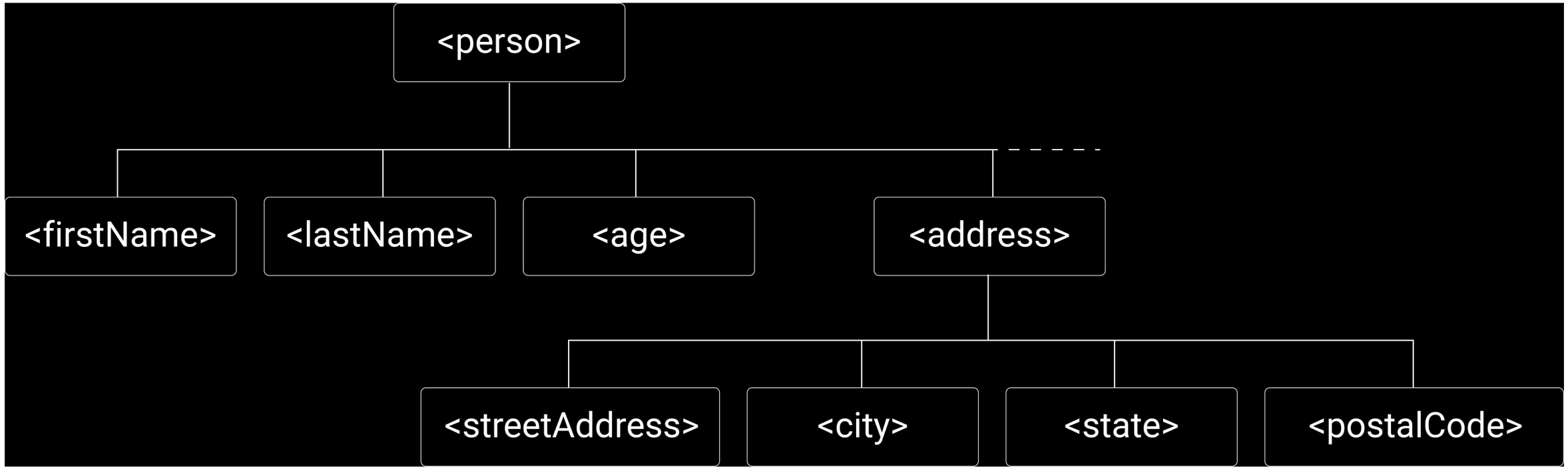
```json
{"firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021"
  },
  "phoneNumber": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "fax",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

Parsing JSON

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'*:

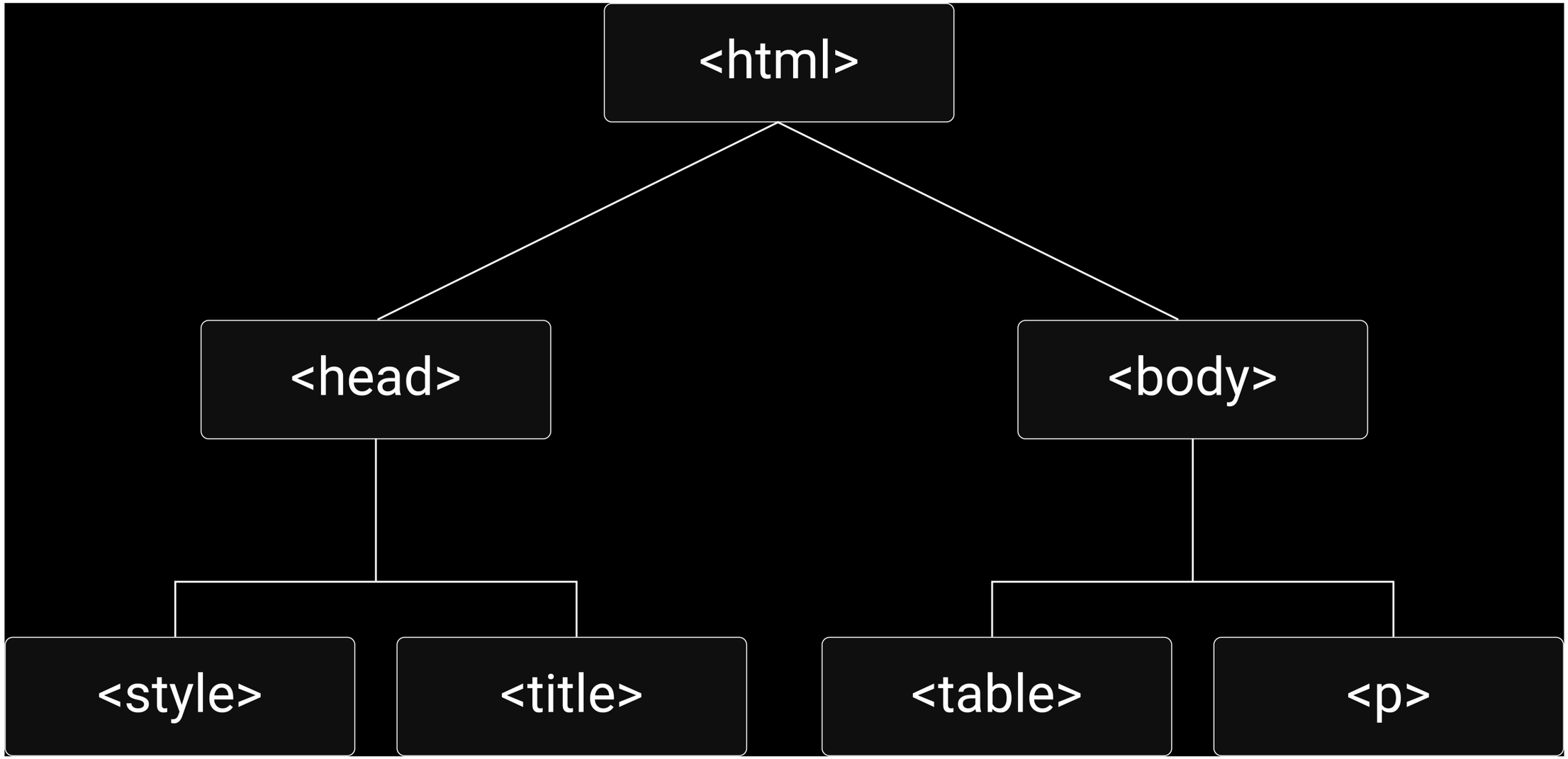
- A webpage can contain a HTML-table (*structured data*)...
- ...but likely also contains just raw text (*unstructured data*).

```
<!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’



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

Source: https://en.wikipedia.org/wiki/Economy_of_Switzerland

Tutorial (advanced): Importing data from a HTML table

-> Exercise session last week!

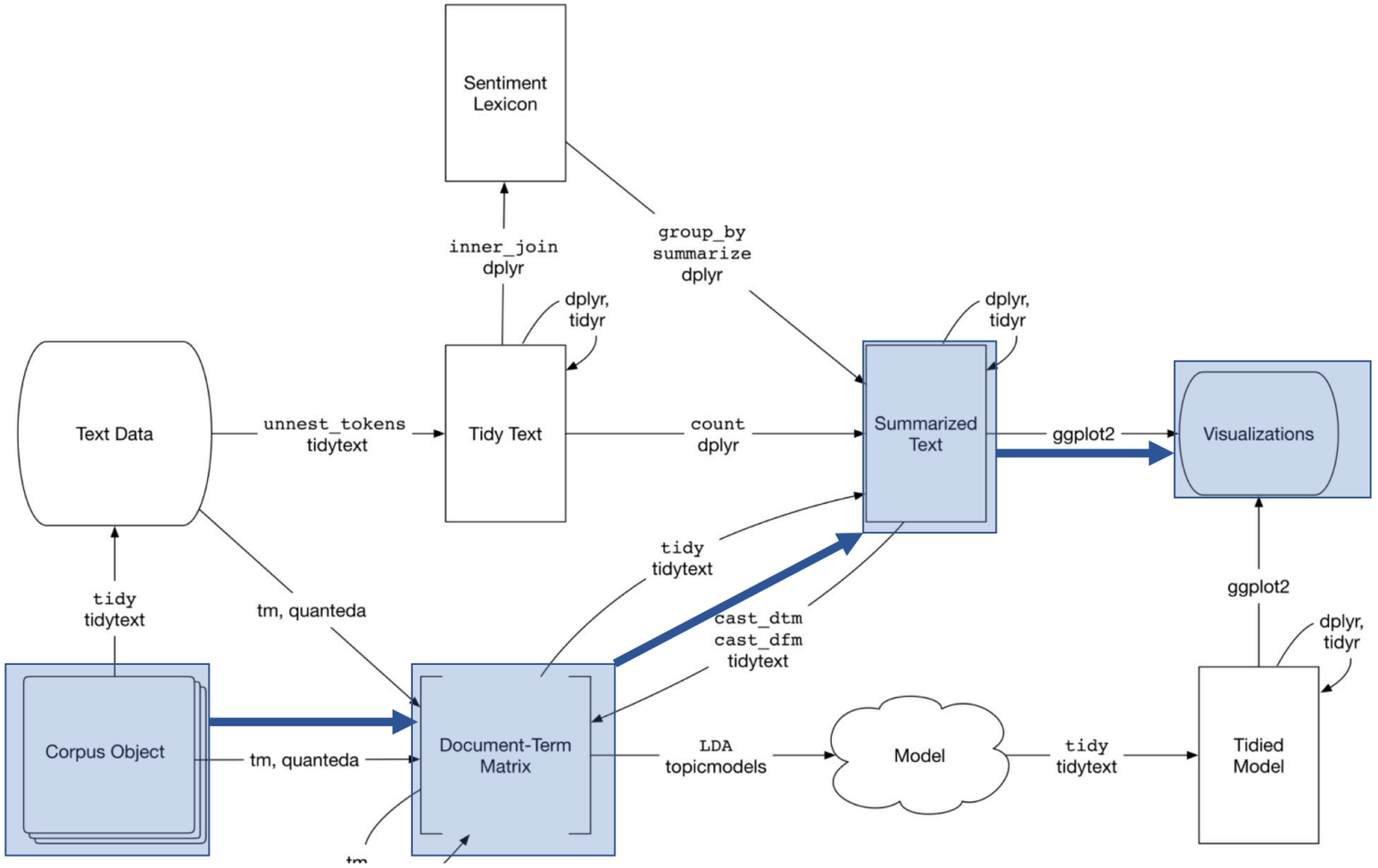
Text as Data and Image Data

Text as Data became more common as data source

- Extract text from historical documents (scan, use OCR)
- Use machine learning to label text (too costly to do manually)
- Extract sentiment (finance for instance)

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

An example of text mining in R



Source: https://confessionsofadatascientist.com/text-analysis-with-r-repro-report.html

Self-study on text mining

Read Chapter 6 on text data in the book by U. Matter

Key concepts:

- **regular expressions**: flexible syntax used to detect and describe patterns in strings
- **corpus**: a collection of authentic text organized into datasets
- **documents**
- **tokens** and **N-grams**
- **document-term-matrix** (dtm)

Key libraries:

- `quanteda`: see Quanteda
- `tidytext`: see Text Mining with R

->

Exercises

JSON files: open-ended question

Consider the following JSON file

```
{
  "students": [
    {
      "id": 19091,
      "firstName": "Peter",
      "lastName": "Mueller",
      "grades": {
          "micro": 5,
          "macro": 4.5,
          "data handling": 5.5
          }
    },
    {
      "id": 19092,
      "firstName": "Anna",
      "lastName": "Schmid",
      "grades": {
          "micro": 5.25,
          "macro": 4,
          "data handling": 5.75
          }
    },
    {
```

XML

```xml
<students>
  <student>
    <id>19091</id>
    <firstName>Peter</firstName>
    <lastName>Mueller</lastName>
    <grades>
      <micro>5</micro>
      <macro>4.5</macro>
      <dataHandling>5.5</dataHandling>
    </grades>
  </student>
  <student>
    <id>19092</id>
    <firstName>Anna</firstName>
    <lastName>Schmid</lastName>
    <grades>
      <micro>5.25</micro>
      <macro>4</macro>
      <dataHandling>5.75</dataHandling>
    </grades>
  </student>
  <student>
    <id>19093</id>
```

- ‘students’ is the root-node, ‘grades’ are its children
- The siblings of Trevor Noah are Anna Schmid and Peter Mueller
- The code below would be an alternative, equivalent notation for the third student in the xml file above.