Data Handling: Import, Cleaning and Visualisation

Lecture 6: Data Sources, Data Gathering, Data Import

Prof. Dr. Ulrich Matter

25/10/2018

1 Data sources

- give overview
- assign learned concepts to source domains:
- Sources with data prepared for research: flat representations
- Archives, large data: Zip
- Still often found: Excel for whatever reason
- Other Stat software: binary (other stat software, etc.) –> find good examples
- Web: XML, JSON, (HTML)

2 Data gathering procedure

- organize script: beginning of pipeline!
- Things to keep in mind:
- encoding
- format

3 Import/Export basics

- build on/recap concepts Murrell (2009) chapter 9.7: w/o databases, own XML/JSON version, and WiGr chapter 11 (consider WiGr in exercises as well)

3.0.0.1 XML in R (from Webmining)

Luckily for us, there are several XML-parsers already implemented in R packages specifically written for working with XML data. We thus do not have to understand the XML syntax in every detail in order to work with this data format in R. The already familiar package `xml2` (automatically loaded when loading `rvest`) provides the `read_xml()` function which we can use to read the exemplary XML-document shown above into R.

```
# load packages
library(xml2)

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

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

The same package also comes with various functions to access, extract, and manipulate data from a parsed XML document. In the following code example, we have a look at the most useful functions for our purposes (see the package's vignette for more details).

```
# navigate through the XML document (recall the tree-like nested structure similar to HTML)
# navigate downwards
# 'customers' is the root-node, persons are it's 'children'
persons <- xml_children(xml_doc)
# navigate sidewards
xml_siblings(persons)
```

```
## {xml_nodeset (2)}
## [1] <person>\n  <name>Peter Pan</name>\n  <orders>\n    <product> a </product>\n    <product> x < ..
## [2] <person>\n  <name>John Doe</name>\n  <orders>\n    <product> x </product>\n    <product> y </ ..
```

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

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

```
# 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"
```

3.1 JSON in R (from Webining)

Again, we can rely on an R package (`jsonlite`) providing high-level functions to read, manipulate, and extract data when working with JSON-documents in R. An important difference to working with XML- and HTML-documents is that XPath is not compatible with JSON. However, as `jsonlite` represents parsed JSON as R objects of class `list` and/or `data-frame`, we can simply work with the parsed document as with any other R-object of the same class. The following example illustrates this point.

```
# 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"
```

```
# 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"
```

References

Murrell, Paul. 2009. *Introduction to Data Technologies*. London, UK: CRC Press.