Data Handling: Import, Cleaning and Visualisation

Lecture 6: Data Sources, Data Gathering, Data Import

Prof. Dr. Ulrich Matter
(University of St. Gallen)

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)

4 Practical Basics of Loading/Importing Data[1]

There are many ways to import data into R from various sources. Here, we look at how to import data from the most common sources when you use R for basic data analysis or econometric exercises.

4.1 Loading built-in datasets

The basic R installation provides some example data sets to try out R's statistics functions. In the introduction to visualization techniques with R as well as in the statistics examples further below, we will rely on some of these datasets for sake of simplicity. Note that the usage of these simple datasets shipped with basic R are very helpful when practicing/learning R on your own. Many R packages use these datasets over and over again in their documentation and examples. Moreover, extensive documentations and tutorials online also use these datasets (see, for example the ggplot2 documentation). And, they are very useful when searching

[1] Taken from Matter (2018)

help on Stackoverflow in the context of data analysis/manipulation with R, as you should provide a code example based on some data that everybody can easily load and access.

In order to load such datasets, simply use the `data()`-function:

```
data(swiss)
```

In this case, we load a dataset called `swiss`. After loading it, the data is stored in a variable of the same name as the dataset (here '`swiss`'). We can inspect it and have a look at the first few rows:

```
# 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             20.3
## Neuveville        76.9        43.5          17        15     5.16             20.6
## Porrentruy        76.1        35.3           9         7    90.57             26.6
```

To get a list of all the built-in datasets simply type `data()` into the console and hit enter. To get more information about a given dataset use the help function (for example, `?swiss`)

4.2 Importing data from files

In most cases of applying R for econometrics and data analysis, students and researchers rely on importing data from files stored on the hard disk. Typically, such datasets are stored in a text-file-format such as 'Comma Separated Values' (CSV). In economics one also frequently encounters data being stored in specific formats of commercial statistics/data analysis packages such as SPSS or STATA. Moreover, when collecting data (e.g., for your Master Thesis) on your own, you might rely on a spreadsheet tool like Microsoft Excel. Data from all of these formats can quite easily be imported into R (in some cases, additional packages have to be loaded, though).

4.2.1 Comma Separated Values (CSV)

A most common format to store and transfer datasets is CSV. In this format data values of one observation are stored in one row of a text file, while commas separate the variables/columns. For example, the following code-block shows how the first two rows of the `swiss`-dataset would look like when stored in a CSV:

```
"","Fertility","Agriculture","Examination","Education","Catholic","Infant.Mortality"
"Courtelary",80.2,17,15,12,9.96,22.2
```

The function `read.csv()` imports such files from disk into R (in the form of a `data.frame`). In this example the `swiss`-dataset is stored locally on our disk in the folder `data`:

```
swiss_imported <- read.csv("data/swiss.csv")
```

4.2.2 Spreadsheets/Excel

In order to read excel spreadsheets we need to install an additional R package called `readxl`.

```
# install the package
install.packages("readxl")
```

Then we load this additional package ('library') and use the package's `read_excel()`-function to import data from an excel-sheet. In the example below, the same data as above is stored in an excel-sheet called `swiss.xlsx`, again in a folder called `data`.

```
# load the package
library(readxl)

# import data from a spreadsheet
swiss_imported <- read_excel("data/swiss.xlsx")
```

4.2.3 Data from other data analysis software

The R packages `foreign` and `haven` contain functions to import data from formats used in other statistics/data analysis software, such as SPSS and STATA.

In the following example we use `haven`'s `read_spss()` function to import a version of the `swiss`-dataset stored in SPSS' `.sav`-format (again stored in the folder called `data`).

```
# 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")
```

4.2.3.1 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).

```r
# 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 </ ...
```

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

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

```r
# 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"
```

4.3 JSON in R (from Webmining)

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.

```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"
```

```r
# 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"
```

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

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

References

Matter, Ulrich. 2018. "A Brief Introduction to Programming with R." Lecture notes. St. Gallen: University of St. Gallen.

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