Data Handling: Import, Cleaning and Visualisation

Lecture 4: "Big Data" from the Web

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

11/10/2018

So far, we have only looked at data structured in a flat/table-like representation (for example, CSV files). In applied econometrics/statistics it is quite common to only work with data sets stored in such formats. The main reason is that data manipulation, filtering, aggregation, etc. anyway presupposes/expects data in a table-like or matrix format. Hence, it makes perfectly sense to already store the data in this format.

However, as we have observed in the previous lecture, when representing more complex data in one text file, the CSV structure has some disadvantages. Particularly, if the data contains nested observations (i.e., hierarchical structures). While a representation in a CSV file is theoretically possible„ it is often far from practical to use other formats for such data. On the one hand it is likely less intuitive to read the data correctly. On the other hand, storing the data in a CSV file would likely introduce a lot of redundency. That is, the identical values of some variables would have to be repeated again and again in the same column. The following code block illustrates this point for a data set on two families ((Murrell 2009), p. 116).

```
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
```

From simply looking at the data we will eventually be able to make a best guess which observations belong together (are one family). However, the implied hierarchy is not apparent at first sight. While it might not matter too much that several values have to be repeated several times in this format, given that this data set is so small, the repeated values can become a problem when the data set is much larger. For each time `John` is repeated in the `father`-column, we use up 4 bytes of memory. If there are millions of people in this data set and/or if we have to transfer this data set very often over a computer network, these repetitions can become quite costly (as we would need more storage capacity and network resources).

All the issues popping up here, complex/hierarchical data (with several observation types), intuitive human readability (self-describing), and efficiency in storage as well as transfer are of great importance in the Web. In course of the development of the Internet several data formats have been put forward to address these issues. Here we discuss the two most prominent of these formats: Extensible Markup Language (XML) and JavaScript Object Notation (JSON).

1 Deciphering XML

Before going into the more technical details, let's try to figure out the basic logic behind the XML format by simply looking at some raw example data. For this, we turn again to the Point Nemo case study. The following code block shows the upper part of the data set we have downloaded from NASA's LAS server (here in a CSV-type format).

```
             VARIABLE : Monthly Surface Clear-sky Temperature (ISCCP) (Celsius)
             FILENAME : ISCCPMonthly_avg.nc
             FILEPATH : /usr/local/fer_data/data/
             BAD FLAG : -1.E+34
             SUBSET   : 48 points (TIME)
             LONGITUDE: 123.8W(-123.8)
             LATITUDE : 48.8S
            123.8W
16-JAN-1994 00  9.200012
16-FEB-1994 00  10.70001
16-MAR-1994 00  7.5
16-APR-1994 00  8.100006
```

Below, same data is now displayed in XML-format. Note that in both cases, the data is simply stored in a text file. However, it is stored in a format that imposes a different *structure* on the data.

```
<?xml version="1.0"?>
<temperatures>
<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" />

...
</temperatures>
```

What features does the format have? What is it's logic? Is there room for improvement?

By using indentation and code highlighting, the XML data structure becomes even more apparent.

```
<?xml version="1.0"?>
  <temperatures>
    <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" />

...
</temperatures>
```

2 JSON

In many web applications, JSON serves the same purpose as XML (programs running on the server side are frequently capable of returning the same data in either format). An obvious difference between the two conventions is that JSON does not use tags but attribute-value pairs to annotate data. The following code example shows how the same data can be represented in XML or in JSON (example code taken from https://en.wikipedia.org/wiki/JSON):

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:

```
{"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"
  }
}
```

Note that despite the differences of the syntax, the similarities regarding the nesting structure are visible in both formats. For example, `postalCode` is embedded in `address`, `firstname` and `lastname` are at the same nesting level, etc.

References

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