Assignment - Working with XML and JSON

Zachary Palmore

10/8/2020

Directions

Pick three of your favorite books on one of your favorite subjects. At least one of the books should have more than one author. For each book, include the title, authors, and two or three other attributes that you find interesting.

Take the information that you've selected about these three books, and separately create three files which store the book's information in HTML (using an html table), XML, and JSON formats (e.g. "books.html", "books.xml", and "books.json"). To help you better understand the different file structures, I'd prefer that you create each of these files "by hand" unless you're already very comfortable with the file formats.

Write R code, using your packages of choice, to load the information from each of the three sources into separate R data frames. Are the three data frames identical?

Your deliverable is the three source files and the R code. If you can, package your assignment solution up into an .Rmd file and publish to rpubs.com. [This will also require finding a way to make your three text files accessible from the web].

Books Data

Loosely based on the subject of *nature*, I have selected three books. Two of the books include two authors and the other book only includes one. The files are each named "books.x" where x is the file type specific to the section header.

Data was written 'by hand' in notepad, a common text editing application present on most (if not all) Windows devices. It was then saved into the working directory. To start, I began loading JSON files into a data frame using the *rjson* package.

JSON Files

JSON file stands for JavaScript Object Notation. It is best for large data formats that have some kind of hierarchical relationship. According to Earth Lab from the University of Colorado at Boulder, the structure of the JSON file can be broken down as follows:

- The data are in name/value pairs
- Data objects are separated by commas
- Each character element is enclosed with quotes ""
    - Each numeric element does not have quotes
- Curly braces {} hold objects
- Square brackets [] hold arrays

These files can also store data of many types, including some of the most frequently used, like strings, numbers, objects, arrays, and more. The data for the books.json file were written following this structure. It looks like:

```
{
   "ID":["1","2","3"],
   "Title":["Magical Mushrooms and Mischeivous Molds","Ecological Restoration","Environmental Science" ]
   "Author1":["Hudler George W","Clewell Andre F","Cunningham William P"],
"Author2":["NA","James Aronson","Cunningham Mary Ann"],
   "PublishYr":[ "1998","2007","2008"],
   "Edition":[ "NA","NA","10"],
    "Subject":[ "Nature","Nature","Nature"],
"Statement":[ "The remarkable story of the fungus kingdom and its impact on human affairs","Priciples, V
}
```

As stated, this was then saved into the working directory from notepad. Then, we load the files using the *fromJSON* function as shown here.

```
# books.json - using "rjson" package
books_json <- fromJSON(file = "books.json")
books_json <- as.data.frame(books_json)
glimpse(books_json)
```

```
## Rows: 3
## Columns: 8
## $ ID        <chr> "1", "2", "3"
## $ Title     <chr> "Magical Mushrooms and Mischeivous Molds", "Ecological Re...
## $ Author1   <chr> "Hudler George W", "Clewell Andre F", "Cunningham William P"
## $ Author2   <chr> "NA", "James Aronson", "Cunningham Mary Ann"
## $ PublishYr <chr> "1998", "2007", "2008"
## $ Edition   <chr> "NA", "NA", "10"
## $ Subject   <chr> "Nature", "Nature", "Nature"
## $ Statement <chr> "The remarkable story of the fungus kingdom and its impac...
```

We can see the data contains 3 rows and 8 columns which corresponds identically to the information written about my selected books. With JSON, we could immediately organize the information into a data frame. This gives us a format that looks clean and would need minimal munging to work with. It could also be exported from here to a .csv or other formats.

```
# datatable(books_json)
books_json
```

```
##   ID                                 Title              Author1
## 1  1 Magical Mushrooms and Mischeivous Molds      Hudler George W
## 2  2                  Ecological Restoration      Clewell Andre F
## 3  3                   Environmental Science Cunningham William P
##               Author2 PublishYr Edition Subject
## 1                  NA      1998      NA  Nature
## 2       James Aronson      2007      NA  Nature
## 3 Cunningham Mary Ann      2008      10  Nature
##                                                                    Statement
## 1 The remarkable story of the fungus kingdom and its impact on human affairs
## 2                 Priciples, Values, and Structure of an Emerging Profession
## 3                                                                A Global Concern
```

Now that we understand the json file structure and how to load it into a data frame, we can move on to another file type. In this case, XML.

XML Files

For this first run, I used a function called *xmlToDataFrame*. This comes from the XML package. It created a data frame containing all the observations listed beneath one variable. Perhaps for some, this would be the final product for exporting or manipulating. However, I rearranged these rows to make it look and function the same as the books.json data frame.

```
books1_xml <- xmlToDataFrame("books.xml")
books1_xml
```

```
##   ID                                  Title              Author1
## 1  1 Magical Mushrooms and Mischeivous Molds      Hudler George W
## 2  2                  Ecological Restoration      Clewell Andre F
## 3  3                   Environmental Science Cunningham William P
##               Author2 PublishYr Edition Subject
## 1                  NA      1998      NA  Nature
## 2       James Aronson      2007      NA  Nature
## 3 Cunningham Mary Ann      2008      10  Nature
##                                                                    Statement
## 1 The remarkable story of the fungus kingdom and its impact on human affairs
## 2                 Priciples, Values, and Structure of an Emerging Profession
## 3                                                              A Global Concern
```

```
# df1 <- books1_xml[1:8,1]
# df2 <- books1_xml[9:16,1]
# df3 <- books1_xml[17:24,1]
# books1_xml <- as.data.frame(rbind(df1, df2, df3))
# colnames(books1_xml) <- c("ID","Title","Author1","Author2","PublishYr","Edition","Subject","Statement
```

Unfortunately, this extra formatting should not have been necessary if the books1.xml file had been formatted correctly. Let's review the books1.xml file. It looks like:

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

<books>
  <ID>1</ID>
    <Title>Magical Mushrooms and Mischeivous Molds</Title>
  <Author1>Hudler George W</Author1>
      <Author2>NA</Author2>
    <PublishYr>1998</PublishYr>
    <Edition>NA</Edition>
    <Subject>Nature</Subject>
    <Statement>The remarkable story of the fungus kingdom and its impact on human affairs</Statement>
  <ID>2</ID>
    <Title>Ecological Restoration</Title>
  <Author1>Clewell Andre F</Author1>
      <Author2>James Aronson</Author2>
    <PublishYr>2007</PublishYr>
    <Edition>NA</Edition>
```

```
    <Subject>Nature</Subject>
    <Statement>Priciples, Values, and Structure of an Emerging Profession</Statement>
  <ID>3</ID>
    <Title>Environmental Science</Title>
    <Author1>Cunningham William P</Author1>
      <Author2>Cunningham Mary Ann</Author2>
    <PublishYr>2008</PublishYr>
    <Edition>10</Edition>
    <Subject>Nature</Subject>
    <Statement>A Global Concern</Statement>
</books>
```

Now, we need a little background to understand why the function gave the output it did. The structure of xml is based on a family tree. There is a root element, then a parent, a child, and their siblings. The parents are the primary values from which the children come from. In the example of books1.xml, the parent would be "<" books ">" and children would be the "<" ID ">", "<" Title ">", "<" Author1 ">" and so on without the quotation marks. But wait, aren't we missing something, and is that how xml is read? Not quite.

It is not enough to write the information as it would appear in the data frame. We need the data formatted appropriately to extract the information and create a data frame from it. So what are we missing and how do we format it? The final version should look more like this:

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

<books>
    <book1>
        <ID>1</ID>
        <Title>Magical Mushrooms and Mischeivous Molds</Title>
    <Author1>Hudler George W</Author1>
        <Author2>NA</Author2>
        <PublishYr>1998</PublishYr>
        <Edition>NA</Edition>
        <Subject>Nature</Subject>
        <Statement>The remarkable story of the fungus kingdom and its impact on human affairs</Statemen
    </book1>
    <book2>
    <ID>2</ID>
        <Title>Ecological Restoration</Title>
    <Author1>Clewell Andre F</Author1>
        <Author2>James Aronson</Author2>
        <PublishYr>2007</PublishYr>
        <Edition>NA</Edition>
        <Subject>Nature</Subject>
        <Statement>Priciples, Values, and Structure of an Emerging Profession</Statement>
    </book2>
    <book3>
        <ID>3</ID>
        <Title>Environmental Science</Title>
    <Author1>Cunningham William P</Author1>
        <Author2>Cunningham Mary Ann</Author2>
        <PublishYr>2008</PublishYr>
        <Edition>10</Edition>
```

```
        <Subject>Nature</Subject>
        <Statement>A Global Concern</Statement>
    </book3>
</books>
```

With xml, we need the right opening and closing brackets for the function to extract the data and format it properly. Thus, we needed to add a book# to the records in the books data. Then we could proceed with the attributes of each "<" book# ">".

Our root is "<" books ">" which contains all the records we need. The parents are "<" book1 ">", "<" book2 ">", and "<" book3 ">". The children "<" ID ">", "<" Title ">", and so on, are a subset of the parents (each "<" book# ">"). However, if needed, the children can also have subchildren. In this case, we do not have any.

Notice, each of these children include data using the format "<" parent ">" parent_data "<" /parent ">" where the backslash, "/" ends the parent_data. For reference, that first line within "<" and ">" at the top of the file is called the prolog. It is how we define our version and encoding of the xml file. Now we can run the books.xml file and see the output.

```
books_xml <- xmlToDataFrame("books.xml")
# datatable(books_xml)
books_xml
```

```
##   ID                                    Title              Author1
## 1  1 Magical Mushrooms and Mischeivous Molds      Hudler George W
## 2  2                   Ecological Restoration      Clewell Andre F
## 3  3                    Environmental Science Cunningham William P
##               Author2 PublishYr Edition Subject
## 1                  NA      1998      NA  Nature
## 2       James Aronson      2007      NA  Nature
## 3 Cunningham Mary Ann      2008      10  Nature
##                                                                    Statement
## 1 The remarkable story of the fungus kingdom and its impact on human affairs
## 2                 Priciples, Values, and Structure of an Emerging Profession
## 3                                                              A Global Concern
```

And there it is. We now have a data frame that can be exported in the format of our choosing that contains all the data formatted as we needed. Hopefully this makes it much easier to understand how to write and import XML files. For our last file type, I will demonstrate how to create and use html files.

HTML Files

Hypertext Markup Language is the standard language for creating webpages. People use it to add structure and context to texts. It is often not the best option to store tabular data, but extracting information from what exists is possible.

Using the readHTMLTable function from the same XML package in our xml example, we can read in data from the file format html. As with XML files, html uses paired tags to denote the starting and ending (or opening and closing) of data within the file. For example, in the books.html we write "<" plus the word "table" and ">" to specify that the elements following the ">" should be formatted as a table. Then we use "<" with "/table" and another ">" to close it.

Since all the information we have is tabular, we can (and should) format all of it within a table. This is a much more efficient use of time than trying to extract information from a "<" "header" ">" and "<" "body" ">" text that requires troubleshooting with regular expressions. The finalized file looks like this:

```
<!DOCTYPE html>
<html lang="en">

<table class="table table-bordered table-hover table-condensed">
<thead>
<tr>

<th title="Field #1">ID</th>
<th title="Field #2">Title</th>
<th title="Field #3">Author1</th>
<th title="Field #4">Author2</th>
<th title="Field #5">PublishYr</th>
<th title="Field #6">Edition</th>
<th title="Field #7">Subject</th>
<th title="Field #8">Statement</th>

</tr>
</thead>

<tbody>

<tr>
<td align="right">1</td>
<td>Magical Mushrooms and Mischeivous Molds</td>
<td>Hudler George W</td>
<td>NA</td>
<td align="right">1998</td>
<td>NA</td>
<td>Nature</td>
<td>The remarkable story of the fungus kingdom and its impact on human affairs</td>
</tr>

<tr>
<td align="right">2</td>
<td>Ecological Restoration</td>
<td>Clewell Andre F</td>
<td>James Aronson</td>
<td align="right">2007</td>
<td>NA</td>
<td>Nature</td>
<td>Priciples, Values, and Structure of an Emerging Profession</td>
</tr>

<tr>
<td align="right">3</td>
<td>Environmental Science</td>
<td>Cunningham William P</td>
<td>Cunningham Mary Ann</td>
<td align="right">2008</td>
<td>10</td>
<td>Nature</td>
<td>A Global Concern</td>
</tr>
```

```
</tbody>
</table>
```

Notice its structure is very similar to XML. It makes use of the < and > operators and specifies the DOCTYPE with the language written in the first line. However, look a little closer and we see differences in the storage parameters.

In the books.html file we see letters that do not appear in the data frame but are necessary for formatting the information into a table. These are specifications such as

to specify tablular data,

to format the data into rows, and

to specify a table header. We can then add other features to the data such as the <td *align="right"*> which aligns the data in the table to the right of a cell.

Another feature that adds structure to this tabular text is the *class="table table-bordered table-hover table-condensed"*. This mouthful takes the table and, if displayed in html, formats it with a border and lets it hover as a condensed table on the webpage. There are many other variables to improve the display and context of the text. Here, we just want the information. To do that, we use the function *readHTMLTable* specifying that we want it stored as a data frame, then we display the results.

```
# Using XML package
books.html <- readHTMLTable("books.html", as.data.frame = TRUE)
books_html <- as.data.frame(books.html)
# datatable(books_html)
books_html
```

```
##   NULL.ID                              NULL.Title         NULL.Author1
## 1       1 Magical Mushrooms and Mischeivous Molds      Hudler George W
## 2       2                  Ecological Restoration      Clewell Andre F
## 3       3                   Environmental Science Cunningham William P
##          NULL.Author2 NULL.PublishYr NULL.Edition NULL.Subject
## 1                  NA           1998           NA       Nature
## 2       James Aronson           2007           NA       Nature
## 3 Cunningham Mary Ann           2008           10       Nature
##                                                                NULL.Statement
## 1 The remarkable story of the fungus kingdom and its impact on human affairs
## 2                 Priciples, Values, and Structure of an Emerging Profession
## 3                                                                A Global Concern
```

Back to 3 rows and 8 columns as desired. From this point the information could be exported to a .csv or other format as necessary. For analysis, this would need only a little munging to manipulate.

Synopsis

As a reminder, we also had the question, are the three data frames identical? Looking at the tables, it appears the information in them is identical. There are only subtle differences in how the data is displayed.

For example, html contains column headers that *Null.Title*, and *Null.Author* rather than the standards of XML and JSON written as *Title* and *Author* without the Null. We could of course edit this to match one another, but we are only comparing on the surface without further cleaning than needed to get the file into a data frame. HTML also extends its table beyond the margins of the webpage, has its text wrap slightly

differently than XML or JSON, and is aligned differently in some columns. This occurs even after trying to account for it in the file itself. HTML is the most unique of these data tables because it is a markup language designed to improve the context and structure of (mainly) text information. Meanwhile, the tables displayed from the XML and JSON files appear identical.

From my perspective, the html file was the most difficult to construct and parse through. There was a lot more thought needed to create the file which, in turn, helped the creation of the data frame but as we now know, it is not the best tool for storing tabular data. XML and JSON were much more effective at formatting and importing without all the unnecessary features that webpage designs would typically need for visual effects. In my opinion, JSON in particular was the most streamlined. By using brackets and braces, there was less writing required, which saved time, and added more structure to the format. Once understood, it was quicker than writing in XML and especially HTML.