Working with XML Data in R

By Tobi Bosede

Date: Jun 13, 2014

Return to the article

Tobi Bosede's step-by-step example shows how easily you can mine XML data from APIs by using R. Simple analysis and even multi-level reporting and output are possible with just a few lines of code.

Working with XML Data in R

A common task for programmers these days is writing code to analyze data from various sources and output information for use by non-coders or business executives. Although you can use any language for this type of analysis, I've found that R simplifies working with almost any modern data type, including XML, a popular choice for storing large amounts of complex data. In this article, I'll step through the process of examining XML data in an R program, so you can see how easy R makes working with XML files.

I first began analyzing XML data while I was working within the public health department at Cornell's Medical School. We'll use the same data source, PubMed, for this article. For this example, I pulled data about papers authored by Dr. Madhu Mazumdar in 2012 from PubMed's API, which is called Entrez.

> **NOTE**
>
> To view or use the code examples in this article, download the code file. The full code referenced in this article is included for the purposes of helping readers follow along in R. In the `pubmed_sample.xml` file, the data entries begin with the `<PubmedArticle>` tag and end with `</PubmedArticle>`.

When dealing with XML data, the main package we'll rely on is XML. However, for our analysis in this example, we'll also need the `plyr`, `ggplot2`, and `gridExtra` packages:

- `plyr` to turn the XML into a dataframe
- `ggplot` to create aesthetically pleasing graphs
- `gridExtra` to put multiple graphs on one canvas

Our first step is to get these packages from the nearest Comprehensive R Archive Network (CRAN) mirror and load them into our current R session:

```
install.packages("XML")
install.packages("plyr")
install.packages("ggplot2")
install.packages("gridExtra")

require("XML")
require("plyr")
require("ggplot2")
require("gridExtra")
```

Next we need to set our working directory and parse the XML file as a matter of practice, so we're sure that R can access the data within the file. This is basically reading the file into R. Then, just to confirm that R knows our file is in XML, we check the class. Indeed, R is aware that it's XML.

```
setwd("C:/Users/Tobi/Documents/R/InformIT") #you will need to change the filepath on  your machine
xmlfile=xmlParse("pubmed_sample.xml")
class(xmlfile) #"XMLInternalDocument" "XMLAbstractDocument"
```

Now we can begin to explore our XML. Perhaps we want to confirm that our HTTP query on Entrez pulled the correct results, just as when we query PubMed's website. We start by looking at the contents of the first node or root, `PubmedArticleSet`. We can also find out how many child nodes the root has and their names. This process corresponds to checking how many entries are in the XML file. The root's child nodes are all named `PubmedArticle`.

```
xmltop = xmlRoot(xmlfile) #gives content of root
class(xmltop)#"XMLInternalElementNode" "XMLInternalNode" "XMLAbstractNode"
xmlName(xmltop) #give name of node, PubmedArticleSet
xmlSize(xmltop) #how many children in node, 19
xmlName(xmltop[[1]]) #name of root's children
```

To see the first two entries, we can do the following.

```
# have a look at the content of the first child entry
xmltop[[1]]
# have a look at the content of the 2nd child entry
xmltop[[2]]
```

Our exploration continues by looking at subnodes of the root. As with the root node, we can list the name and size of the subnodes as well as their attributes. In this case, the subnodes are `MedlineCitation` and `PubmedData`.

```
#Root Node's children
xmlSize(xmltop[[1]]) #number of nodes in each child
xmlSApply(xmltop[[1]], xmlName) #name(s)
```

```
xmlSApply(xmltop[[1]], xmlAttrs) #attribute(s)
xmlSApply(xmltop[[1]], xmlSize) #size
```

We can also separate each of the 19 entries by these subnodes. Here we do so for the first and second entries:

```
#take a look at the MedlineCitation subnode of 1st child
xmltop[[1]][[1]]
#take a look at the PubmedData subnode of 1st child
xmltop[[1]][[2]]

#subnodes of 2nd child
xmltop[[2]][[1]]
xmltop[[2]][[2]]
```

The separation of entries is really just us, indexing into the tree structure of the XML. We can continue to do this until we exhaust a path—or, in XML terminology, reach the end of the branch. We can do this via the numbers of the child nodes or their actual names:

```
#we can keep going till we reach the end of a branch
xmltop[[1]][[1]][[5]][[2]] #title of first article
xmltop[['PubmedArticle']][['MedlineCitation']][['Article']][['ArticleTitle']] #same command, but more readable
```

Finally, we can transform the XML into a more familiar structure—a dataframe. Our command completes with errors due to non-uniform formatting of data and nodes. So we must check that all the data from the XML is properly inputted into our dataframe. Indeed, there are duplicate rows, due to the creation of separate rows for tag attributes. For instance, the `ELocationID` node has two attributes, `validYN` and `EIDType`. Take the time to note how the duplicates arise from this separation.

```
#Turning XML into a dataframe
Madhu2012=ldply(xmlToList("pubmed_sample.xml"), data.frame) #completes with errors: "row names were found from a short variable and have been discarded"
View(Madhu2012) #for easy checking that the data is properly formatted
Madhu2012.Clean=Madhu2012[Madhu2012[25]=='Y',] #gets rid of duplicated rows
```

Taking a look at the titles of the column headings after calling the `View()` function on the dataframe, it's clear that they're the paths from the root through various child branches until the terminus or our data of interest is reached. Try to go through the XML document to see if you can follow the path to a specific piece of data in a column.

Now that our XML data is a dataframe, we can do some analysis. For instance, we might be interested in which authors contributed most to the articles of which Dr. Mazumdar was an author in 2012. We can make histograms for the first through eleventh authors, with the understanding that first authors contribute the most, and eleventh authors the least. The code below labels the relevant data for easy tracking and removes blank entries, which we don't want appearing in our histogram.

```
#looking at which authors played most active role
FirstAuthor=Madhu2012.Clean$MedlineCitation.Article.AuthorList.Author.LastName
SecondAuthor=Madhu2012.Clean$MedlineCitation.Article.AuthorList.Author.LastName.1
ThirdAuthor=Madhu2012.Clean$MedlineCitation.Article.AuthorList.Author.LastName.2

#removing NAs
Madhu2012.Na.Rm4=Madhu2012.Clean[!is.na(Madhu2012.Clean$MedlineCitation.Article.AuthorList.Author.LastName.3),]
FourthAuthor=Madhu2012.Na.Rm4$MedlineCitation.Article.AuthorList.Author.LastName.3
Madhu2012.Na.Rm5=Madhu2012.Clean[!is.na(Madhu2012.Clean$MedlineCitation.Article.AuthorList.Author.LastName.4),]
FifthAuthor=Madhu2012.Na.Rm5$MedlineCitation.Article.AuthorList.Author.LastName.4
Madhu2012.Na.Rm6=Madhu2012.Clean[!is.na(Madhu2012.Clean$MedlineCitation.Article.AuthorList.Author.LastName.5),]
SixthAuthor=Madhu2012.Na.Rm6$MedlineCitation.Article.AuthorList.Author.LastName.5
Madhu2012.Na.Rm7=Madhu2012.Clean[!is.na(Madhu2012.Clean$MedlineCitation.Article.AuthorList.Author.LastName.6),]
SeventhAuthor=Madhu2012.Na.Rm7$MedlineCitation.Article.AuthorList.Author.LastName.6
Madhu2012.Na.Rm8=Madhu2012.Clean[!is.na(Madhu2012.Clean$MedlineCitation.Article.AuthorList.Author.LastName.7),]
EighthAuthor=Madhu2012.Na.Rm8$MedlineCitation.Article.AuthorList.Author.LastName.7
Madhu2012.Na.Rm9=Madhu2012.Clean[!is.na(Madhu2012.Clean$MedlineCitation.Article.AuthorList.Author.LastName.8),]
NinthAuthor=Madhu2012.Na.Rm9$MedlineCitation.Article.AuthorList.Author.LastName.8
Madhu2012.Na.Rm10=Madhu2012.Clean[!is.na(Madhu2012.Clean$MedlineCitation.Article.AuthorList.Author.LastName.9),]
TenthAuthor=Madhu2012.Na.Rm10$MedlineCitation.Article.AuthorList.Author.LastName.9
Madhu2012.Na.Rm11=Madhu2012.Clean[!is.na(Madhu2012.Clean$MedlineCitation.Article.AuthorList.Author.LastName.10),]
EleventhAuthor=Madhu2012.Na.Rm11$MedlineCitation.Article.AuthorList.Author.LastName.10
```

Next, we create histograms detailing counts of each author's last name per author type. Our analysis then aggregates the 11 diagrams onto three pages in a single PDF file for easy comparison. The PDF file is included in the code file for this article.

```
#write all the graphs to pdf on 3 canvases
a=ggplot(Madhu2012.Clean, aes(x=FirstAuthor)) + geom_histogram(binwidth=.5, colour="pink", fill="purple")+coord_flip()
b=ggplot(Madhu2012.Clean, aes(x=SecondAuthor)) + geom_histogram(binwidth=.5, colour="pink", fill="purple")+coord_flip()
c=ggplot(Madhu2012.Clean, aes(x=ThirdAuthor)) + geom_histogram(binwidth=.5, colour="pink", fill="purple")+coord_flip()
d=ggplot(Madhu2012.Na.Rm4, aes(x=FourthAuthor)) + geom_histogram(binwidth=.5, colour="pink", fill="purple")+coord_flip()
e=ggplot(Madhu2012.Na.Rm5, aes(x=FifthAuthor)) + geom_histogram(binwidth=.5, colour="pink", fill="purple")+coord_flip()
f=ggplot(Madhu2012.Na.Rm6, aes(x=SixthAuthor)) + geom_histogram(binwidth=.5, colour="pink", fill="purple")+coord_flip()
g=ggplot(Madhu2012.Na.Rm7, aes(x=SeventhAuthor)) + geom_histogram(binwidth=.5, colour="pink", fill="purple")+coord_flip()
h=ggplot(Madhu2012.Na.Rm8, aes(x=EighthAuthor)) + geom_histogram(binwidth=.5, colour="pink", fill="purple")+coord_flip()
i=ggplot(Madhu2012.Na.Rm9, aes(x=NinthAuthor)) + geom_histogram(binwidth=.5, colour="pink", fill="purple")+coord_flip()
j=ggplot(Madhu2012.Na.Rm10, aes(x=TenthAuthor)) + geom_histogram(binwidth=.5, colour="pink", fill="purple")+coord_flip()
k=ggplot(Madhu2012.Na.Rm11, aes(x=EleventhAuthor)) + geom_histogram(binwidth=.5, colour="pink", fill="purple")+coord_flip()

pdf("AuthorHistogram.pdf")
grid.arrange(a,b,c,d)
grid.arrange(e,f,g,h)
grid.arrange(i,j,k)
dev.off()
```

Page 1 of the results is shown in Figure 1. (The complete PDF is included in the code file for download.) However, when we look at the other pages, we see that of all the 19 articles published under Dr. Mazumdar's name in 2012 (whether in print or online), she was most often listed fourth, fifth, or sixth in the authors of the paper. This means that she was only moderately active in terms of writing the papers. Instead, we see that Dr. Memtsoudis was the primary writer of about 1/3 of the papers, with Dr. Ma as next most active. We should also note that most of the papers have just six or fewer authors. Hence, sometimes Dr. Mazumdar was the last author and thus contributed in a minor way.

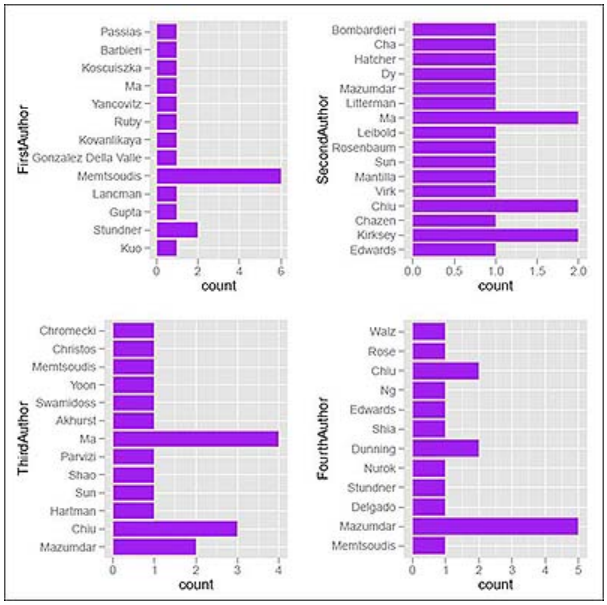

Figure 1 Histogram for the first four authors used in analysis of level of contribution.

Finally, we can write the dataframe to a `.csv` or `.txt` file and import it into another program such as SQL, Python, or SAS. Or we might want to share our data with a collaborator. The output files are included in the code file for this article.

```
#exporting data
write.table(Madhu2012.Clean, "Madhu2012.txt", sep="\t", row.names=FALSE)
write.csv(Madhu2012.Clean, "Madhu2012.csv", row.names=FALSE)
```

That was interesting to learn. No big deal, right?

As you can imagine, R allows us to perform many other analyses on our XML data with ease. For example, we might be interested in finding out which journals published 2012 papers by Dr. Mazumdar, or which grants supported her work.

As we've seen in this example, with R, there's no need to shy away from unfamiliar data types such as XML. The world is your oyster!
