Creating dataframes from xml files in R

Breakfast Menu Example

When I took the Getting and Cleaning data course in Coursera, I had a great time exploring the topic of downloading .xml files and the xml package by Duncan Temple Lang. However, because the course a lot of interesting materials, I had to wait after the course to explore the topic thoroughly. This tutorial hopes to share what I have learned through my own exploration and share it with other students of the course who might be inclined to learn more about this topic.

For our example we will be using the sample file from the w3schools website: You can use your browser to view the file in its entirety. http://www.w3schools.com/xml/simple.xml or see it below.

```
<breakfast_menu>
      <food>
            <name>Belgian Waffles</name>
            <price>$5.95</price>
            <description>Two of our famous Belgian Waffles with plenty of real
                  maple syrup</description>
            <calories>650</calories>
      </food>
      <food>
            <name>Strawberry Belgian Waffles</name>
            <price>$7.95</price>
            <description>Light Belgian waffles covered with strawberries and
                  whipped cream</description>
            <calories>900</calories>
      </food>
      <food>
            <name>Berry-Berry Belgian Waffles</name>
            <price>$8.95</price>
            <description>Light Belgian waffles covered with an assortment of fresh
                  berries and whipped cream</description>
            <calories>900</calories>
      </food>
      <food>
            <name>French Toast</name>
            <price>$4.50</price>
            <description>Thick slices made from our homemade sourdough bread
            </description>
            <calories>600</calories>
      </food>
      <food>
            <name>Homestyle Breakfast</name>
            <price>$6.95</price>
            <description>Two eggs, bacon or sausage, toast, and our ever-popular
                  hash browns</description>
            <calories>950</calories>
      </food>
</breakfast_menu>
```

It will help you understand .xml files better if you read about the parts of an xml file and its hierarchial structure. There are many available resources in the web about this topic.

I have downloaded the file earlier in a directory called data. Let's now read the file in R.
We set the R to the directory that contains the file and load the xml package which were going to use.

```
> setwd("C:/Users/user/JHSPH/GCD/COURSE/Lab/data")
> library(XML)
```

We use the function xmlTreeParse to read in the file with the argument useInternal = TRUE. This allows us to use later on the functions getNodeSet and xpathSApply which works only on objects class "XMLInternalDocument"

```
> doc <- xmlTreeParse("simple.xml",useInternal=TRUE)
> class(doc)
[1] "XMLInternalDocument" "XMLAbstractDocument"
```

We now use the function xmlRoot to obtain access to the top node. The top node is the node that contains all the other nodes in the file if you look at the file in page 1 it shows on top the opening tag `<breakfast_menu>` and at the bottom the closing tag `</breakfast_menu>`.

```
> topNode <- xmlRoot(doc)
> class(rootNode)
[1] "XMLInternalElementNode" "XMLInternalNode"       "XMLAbstractNode"
> xmlName(topNode)
[1] "breakfast_menu"
```

We now use the function getnodset to obtain all the nodes with the tag food, including all the nodes between them.

```
> els <- getNodeSet(topNode, "//food")
> class(els)
[1] "XMLNodeSet"
```

Looking at the R object els we can see the structure of the information we want to access. All the nodes that are tagged as food are arranged next to each other with no other nodes separating them.

```
> els
[[1]]
<food>
   <name>Belgian Waffles</name>
   <price>$5.95</price>
   <description>Two of our famous Belgian Waffles with plenty of real maple syrup</description>
   <calories>650</calories>
</food>
...
[[5]]
<food>
   <name>Homestyle Breakfast</name>
```

```
   <price>$6.95</price>
   <description>Two eggs, bacon or sausage, toast, and our ever-popular hash
browns</description>
   <calories>950</calories>
</food>
```

Using the t() and data.frame() functions in R

We can use the function xmlSApply and xmlValue to obtain all the values of the nodes in between the tags food.

```
> xmlSApply(els, function(x)xmlSApply(x, xmlValue))
                    [,1]
name        "Belgian Waffles"
price       "$5.95"
description "Two of our famous Belgian Waffles with plenty of real maple syrup"
calories    "650"
                    [,2]
name        "Strawberry Belgian Waffles"
price       "$7.95"
description "Light Belgian waffles covered with strawberries and whipped cream"
calories    "900"
                    [,3]
name        "Berry-Berry Belgian Waffles"
price       "$8.95"
description "Light Belgian waffles covered with an assortment of fresh berries and whipped cream"
calories    "900"
                    [,4]
name        "French Toast"
price       "$4.50"
description "Thick slices made from our homemade sourdough bread"
calories    "600"
                    [,5]
name        "Homestyle Breakfast"
price       "$6.95"
description "Two eggs, bacon or sausage, toast, and our ever-popular hash browns"
calories    "950"
```

The result is an object with class matrix.

```
> class(xmlSApply(els, function(x)xmlSApply(x, xmlValue)))
[1] "matrix"
```

The object almost looks like a dataframe except that the columns appear to be in the rows and the rows are in the position where the columns should be. We use the t function to invert the position of the rows and column.

```
> t(xmlSApply(els, function(x)xmlSApply(x, xmlValue)))
     name                    price
```

```
[1,] "Belgian Waffles"            "$5.95"
[2,] "Strawberry Belgian Waffles"  "$7.95"
[3,] "Berry-Berry Belgian Waffles" "$8.95"
[4,] "French Toast"               "$4.50"
[5,] "Homestyle Breakfast"        "$6.95"
        description
[1,] "Two of our famous Belgian Waffles with plenty of real maple syrup"
[2,] "Light Belgian waffles covered with strawberries and whipped cream"
[3,] "Light Belgian waffles covered with an assortment of fresh berries and whipped cream"
[4,] "Thick slices made from our homemade sourdough bread"
[5,] "Two eggs, bacon or sausage, toast, and our ever-popular hash browns"
        calories
[1,] "650"
[2,] "900"
[3,] "900"
[4,] "600"
[5,] "950"
```

And we finally use the function data.frame to create our dataframe.

```
> data.frame(t(xmlSApply(els, function(x)xmlSApply(x, xmlValue))))
                                    name price
1            Belgian Waffles $5.95
2  Strawberry Belgian Waffles $7.95
3 Berry-Berry Belgian Waffles $8.95
4               French Toast $4.50
5         Homestyle Breakfast $6.95
                                                                                 description
1                Two of our famous Belgian Waffles with plenty of real maple syrup
2                Light Belgian waffles covered with strawberries and whipped cream
3 Light Belgian waffles covered with an assortment of fresh berries and whipped cream
4                           Thick slices made from our homemade sourdough bread
5              Two eggs, bacon or sausage, toast, and our ever-popular hash browns
   calories
1      650
2      900
3      900
4      600
5      950
```

Creating individual vectors

Another approach is to create a vector of all the variables that will be included in the dataframe and then bind them together later on. This is very useful when the data you need from the .xml file is not arranged as neatly as in our example or when the data is located in different parts of the .xml file.

We use the getNodeSet function again but this time targetting the individual nodes for the variables in our data frame

```
> els_name <- getNodeSet(topNode, "//name")
> els_price <- getNodeSet(topNode, "//price")
```

```
> els_desc <- getNodeSet(topNode, "//description")
> els_calorie <- getNodeSet(topNode, "//calories")
```

Now that we have all the nodes we need, we will extract the values from the nodes using xmlSApply

```
> food_names <- xmlSApply(els_name, xmlValue)
> price <- xmlSApply(els_price, xmlValue)
> description <- xmlSApply(els_desc, xmlValue)
> calories <- xmlSApply(els_calorie, xmlValue)
```

We now put everything together to create our dataframe.

```
> data.frame(name = food_names, price = price, description = description, calories = calories)
                                  name price
1           Belgian Waffles $5.95
2  Strawberry Belgian Waffles $7.95
3 Berry-Berry Belgian Waffles $8.95
4              French Toast $4.50
5        Homestyle Breakfast $6.95
                                                                                description
1                Two of our famous Belgian Waffles with plenty of real maple syrup
2                Light Belgian waffles covered with strawberries and whipped cream
3 Light Belgian waffles covered with an assortment of fresh berries and whipped cream
4                          Thick slices made from our homemade sourdough bread
5              Two eggs, bacon or sausage, toast, and our ever-popular hash browns
  calories
1      650
2      900
3      900
4      600
5      950
```

Using the rbind function in R

Another approach that works when the data is arranged similarly like in our example is to use the function rbind in R.

We use the getNodeSet function in the same way we did in the first example, but this time we add another argument fun = xmlToList. If you want to learn more about the function getNodeSet or xmlToList, just type ?getNodeSet or ?xmlToList at the console to view the help files in Rstudio.

```
> els <- getNodeSet(topNode, "//food", fun = xmlToList)

> els
[[1]]
[[1]]$name
[1] "Belgian Waffles"

[[1]]$price
```

[1] "$5.95"

[[1]]$description
[1] "Two of our famous Belgian Waffles with plenty of real maple syrup"

[[1]]$calories
[1] "650"

[[2]]
[[2]]$name
[1] "Strawberry Belgian Waffles"

[[2]]$price
[1] "$7.95"

[[2]]$description
[1] "Light Belgian waffles covered with strawberries and whipped cream"

[[2]]$calories
[1] "900"

[[3]]
[[3]]$name
[1] "Berry-Berry Belgian Waffles"

[[3]]$price
[1] "$8.95"

[[3]]$description
[1] "Light Belgian waffles covered with an assortment of fresh berries and whipped cream"

[[3]]$calories
[1] "900"

[[4]]
[[4]]$name
[1] "French Toast"

[[4]]$price
[1] "$4.50"

[[4]]$description
[1] "Thick slices made from our homemade sourdough bread"

[[4]]$calories
[1] "600"

[[5]]
[[5]]$name
[1] "Homestyle Breakfast"

[[5]]$price
[1] "$6.95"

```
[[5]]$description
[1] "Two eggs, bacon or sausage, toast, and our ever-popular hash browns"

[[5]]$calories
[1] "950"

> class(els)
[1] "list"
```

This time the R object els is of class list whereas before it was of class XMLNodeSet.

```
> data.frame(do.call(rbind, els))
                                    name price
1           Belgian Waffles $5.95
2  Strawberry Belgian Waffles $7.95
3 Berry-Berry Belgian Waffles $8.95
4              French Toast $4.50
5        Homestyle Breakfast $6.95
                                                                                                  description
1                Two of our famous Belgian Waffles with plenty of real maple syrup
2                Light Belgian waffles covered with strawberries and whipped cream
3 Light Belgian waffles covered with an assortment of fresh berries and whipped cream
4                          Thick slices made from our homemade sourdough bread
5              Two eggs, bacon or sausage, toast, and our ever-popular hash browns
   calories
1     650
2     900
3     900
4     600
5     950
```

Again, this is very useful when the data you need is arranged in the same way as our example.

Using the argument fun = xmlToDataFrame

Another argument that we can pass on to the function getNodeSet is fun = xmlToDATAframe.

```
> els <- getNodeSet(topNode, "//food", fun = xmlToDataFrame)
> els
[[1]]
                                                                text
1                                                  Belgian Waffles
2                                                           $5.95
3 Two of our famous Belgian Waffles with plenty of real maple syrup
4                                                             650

[[2]]
                                                                text
1                                               Strawberry Belgian Waffles
2                                                           $7.95
3 Light Belgian waffles covered with strawberries and whipped cream
4                                                             900
```

```
[[3]]

                                                    text
1                                    Berry-Berry Belgian Waffles
2                                                         $8.95
3 Light Belgian waffles covered with an assortment of fresh berries and whipped cream
4                                                           900

[[4]]
                                                                text
1                                 French Toast
2                                        $4.50
3 Thick slices made from our homemade sourdough bread
4                                        600

[[5]]
                                                                                    text
1                                    Homestyle Breakfast
2                                                $6.95
3 Two eggs, bacon or sausage, toast, and our ever-popular hash browns
4                                                 950

> class(els)
[1] "list"
> class(els[[1]])
[1] "data.frame"
```

This time the R object els is a list of dataframes.

```
> data.frame(t(do.call(cbind, els)),row.names = NULL)
                                    X1    X2
1           Belgian Waffles $5.95
2  Strawberry Belgian Waffles $7.95
3 Berry-Berry Belgian Waffles $8.95
4              French Toast $4.50
5        Homestyle Breakfast $6.95

X3
1              Two of our famous Belgian Waffles with plenty of real maple syrup
2              Light Belgian waffles covered with strawberries and whipped cream
3 Light Belgian waffles covered with an assortment of fresh berries and whipped cream
4                        Thick slices made from our homemade sourdough bread
5            Two eggs, bacon or sausage, toast, and our ever-popular hash browns
    X4
1 650
2 900
3 900
4 600
5 950
```

We use the function t, do.call, cbind, and data.frame to create the dataframe of the variables we want.

```
> food_df <-data.frame(t(do.call(cbind, els)),row.names = NULL)
```

```
> names(food_df) <- c("name", "price", "description", "calories")
```

We have to replace the names of our dataframe as an additional step.

Using the function xmlToDataFrame

Perhaps the easiest method to use when the data is organized neatly like in our example is to use the function xmlToDataFrame function.

```
> xmlToDataFrame(els)
                         name price
1             Belgian Waffles $5.95
2  Strawberry Belgian Waffles $7.95
3 Berry-Berry Belgian Waffles $8.95
4                French Toast $4.50
5         Homestyle Breakfast $6.95
                                                                          description
1                  Two of our famous Belgian Waffles with plenty of real maple syrup
2                  Light Belgian waffles covered with strawberries and whipped cream
3 Light Belgian waffles covered with an assortment of fresh berries and whipped cream
4                               Thick slices made from our homemade sourdough bread
5                Two eggs, bacon or sausage, toast, and our ever-popular hash browns
  calories
1      650
2      900
3      900
4      600
5      950
```

Using the function xmlToList

Please take note that the function xmlToList gives a different result when used on the R object doc with the argument useInternal is set to TRUE and when it is set to FALSE.

```
> docF <- xmlTreeParse("simple.xml",useInternal=FALSE)
> class(xmlToList(docF))
[1] "matrix"
```

```
> docT <- xmlTreeParse("simple.xml",useInternal=TRUE)
> class(xmlToList(docT))
[1] "list"
```

useInternal=FALSE

```
> data.frame(t(xmlToList(docF)))
                         name price
```

```
1            Belgian Waffles $5.95
2  Strawberry Belgian Waffles $7.95
3 Berry-Berry Belgian Waffles $8.95
4              French Toast $4.50
5        Homestyle Breakfast $6.95

description
1                Two of our famous Belgian Waffles with plenty of real maple syrup
2                Light Belgian waffles covered with strawberries and whipped cream
3 Light Belgian waffles covered with an assortment of fresh berries and whipped cream
4                             Thick slices made from our homemade sourdough bread
5              Two eggs, bacon or sausage, toast, and our ever-popular hash browns
   calories
1      650
2      900
3      900
4      600
5      950
Warning message:
In data.row.names(row.names, rowsi, i) :
   some row.names duplicated: 2,3,4,5 --> row.names NOT used
```

You get a warning saying that the row.names were not used.

useInternal=TRUE

a different approach is used when the argument useInternal is set to TRUE

```
> class(rbind(xmlToList(docT)[[1]], xmlToList(docT)[[2]], xmlToList(docT)[[3]], xmlToList(docT)[[4]],
xmlToList(docT)[[5]]))
[1] "matrix"
```

```
> data.frame(rbind(xmlToList(docT)[[1]], xmlToList(docT)[[2]], xmlToList(docT)[[3]],
xmlToList(docT)[[4]], xmlToList(docT)[[5]]))
                                    name price
1            Belgian Waffles $5.95
2  Strawberry Belgian Waffles $7.95
3 Berry-Berry Belgian Waffles $8.95
4              French Toast $4.50
5        Homestyle Breakfast $6.95

description
1                Two of our famous Belgian Waffles with plenty of real maple syrup
2                Light Belgian waffles covered with strawberries and whipped cream
```

3	Light Belgian waffles covered with an assortment of fresh berries and whipped cream	
4	Thick slices made from our homemade sourdough bread	
5	Two eggs, bacon or sausage, toast, and our ever-popular hash browns	
	calories	
1	650	
2	900	
3	900	
4	600	
5	950	

Hope you found this somewhat helpful.

EJLOfilada