STAT 19000 Project 6

Topics: xml

Motivation: As data comes in many formats, understanding different file formats, and how we can extract information from them is a crucial step in data analysis. In the previous project, we began to understand XML file formats. We will continue to discuss how to explore XML files using R in this project.

Context: We are gaining familiarity with XML files, and have seen how we can use R to parse XML files. In this project, we will continue our exploration of XML files, and how we can extract useful information from them. To do so we will use XPath and path expressions to select nodes or node-sets.

Scope: Understand how parse and analyze XML using R and XPath. We will look into path expressions, and the xml wrappers for the apply functions.

You can find useful examples that walk you through relevant material here or on scholar:

`/class/datamine/data/spring2020/stat19000project06examples.R`.

It is highly recommended to read through these to help solve problems.

Use the template found here or on scholar:

`/class/datamine/data/spring2020/stat19000project06template.ipynb`

to submit your solutions.

Important note: Make sure you have your output calculated and displayed inside your notebook prior to submission.

After each problem, we've provided you with a list of keywords. These keywords could be package names, functions, or important terms that will point you in the right direction when trying to solve the problem, and give you accurate terminology that you can further look into online. You are not required to utilize all the given keywords. You will receive full points as long as your code gives the correct result.

Don't forget the very useful documentation shortcut `?`. To use, simply type `?` in the console, followed by the name of the function you are interested in.

You can also look for package documentation by using `help(package=PACKAGENAME)`.

Sometimes it can be helpful to see the source code of a defined function. To do so, type the function's name without the `()`.

Question 1:

Scrape the NASA information located here using either the `xmlParse` function from the `XML` package or the `read_xml` function from the `xml2` package.

To scrape the data using `xmlParse` from the `XML` package, run:

```r
url <- 'http://aiweb.cs.washington.edu/research/projects/xmltk/xmldata/data/nasa/nasa.xml'
nasa_xml <- xmlParse(url)
```

To scrape using `read_xml` from the `xml2` package, run:

```r
url <- 'http://aiweb.cs.washington.edu/research/projects/xmltk/xmldata/data/nasa/nasa.xml'
nasa_xml <- read_xml(url)
```

Pick your favorite XML package (`XML` or `xml2`) to work with for the following questions. You do *not* need to provide solutions using both packages!

1a. *(1 pt)* Create two variables, `nasa_root` and `datasets`. Let `nasa_root` be the root node of `nasa_xml`, and `datasets` be its children nodes. What are the names of the first dataset's (`datasets[[1]]`) children? Make sure to use either the function `xmlSApply` from the `XML` package or the function `xml_contents` from the `xml2` package.

Hint: *Begin by getting the root of your parsed XML. Then proceed by getting the root's children, and answer the questions. This should look familiar to you (you did this in project 5). However this time we will use xmlSApply or xml_contents.*

Keywords (XML package): *xmlRoot, xmlChildren, xmlSApply, xmlName*

Keywords (xml2 package): *xml_root, xml_children, xml_contents, xml_name*

Item(s) to submit:

- The code used for the problem.
- The names of the first dataset's 10 children.

1b. *(1 pt)* Replace `your_path_expression` with an actual path expression that finds all the nodes named 'identifier' in `nasa_xml` from any other node. You can check that your path expression works by running the 3 commands below with your replaced `your_path_expression`. They should all give you the exact same result.

If you are using XML package, use the 3 commands below:

```r
# The path expression we are looking for should find all
# of the nodes named 'identifier' regardless of our
# current node (first argument).
length(getNodeSet(nasa_xml, 'your_path_expression')) # 2435
length(getNodeSet(datasets[[25]], 'your_path_expression')) # 2435
length(getNodeSet(nasa_root, 'your_path_expression')) # 2435

# It may look like the first parameter doesn't matter,
# but wait, it sure does if the path expression
# uses the context of the current node (first parameter).
# The "." here means start from the current node. The
# "//" means search for all descendants called "field".
length(getNodeSet(datasets[[25]], './/field')) # 19
length(getNodeSet(datasets[[24]], './/field')) # 4
```

If you are using xml2 package, use the 3 commands below:

```r
# The path expression we are looking for should find all
# of the nodes named 'identifier' regardless of our
# current node (first argument).
length(xml_find_all(nasa_xml, 'your_path_expression')) # 2435
length(xml_find_all(datasets[[25]], 'your_path_expression')) # 2435
length(xml_find_all(nasa_root, 'your_path_expression')) # 2435

# It may look like the first parameter doesn't matter,
# but wait, it sure does if the path expression
# uses the context of the current node (first parameter).
# The "." here means start from the current node. The
# "//" means search for all descendants called "field".
length(xml_find_all(datasets[[25]], './/field')) # 19
length(xml_find_all(datasets[[24]], './/field')) # 4
```

Hint: *Be sure to take a look at the examples.*

Hint: *Printing the first dataset may be useful. One way to print it is to use the command `datasets[[1]]`.*

Item(s) to submit:
- The code used for the problem.
- A markdown cell (`Cell > Cell Type > Markdown`) containing the path expression.
- The output from running one set of the three lines of code provided above.

1c. *(2 pt)* Create a variable called `datasets_identifiers` containing the identifiers for all datasets. Print the first 10 values.

Hint: *If you use the function `xpathSApply()`, make sure its first argument is a document of class `XMLInternalDocument`.*

Keywords (XML package): *xpathSApply, xmlValue*

Keywords (xml2 package): *xml_content, xml_text*

Item(s) to submit:
- The code used for the problem.
- The output from printing the first 10 values of the resulting `datasets_identifiers` variable.

Question 2:

Scrape the WSU course information located here using either the `xmlParse` function from the `XML` package or the `read_xml` function from the `xml2` package.

To scrape the data using `xmlParse` from the `XML` package, run:

```
url <- 'http://aiweb.cs.washington.edu/research/projects/xmltk/xmldata/data/courses/wsu.xml'
wsu_xml <- xmlParse(url)
```

To scrape using `read_xml` from the `xml2` package, run:

```
url <- 'http://aiweb.cs.washington.edu/research/projects/xmltk/xmldata/data/courses/wsu.xml'
wsu_xml <- read_xml(url)
```

Pick your favorite XML package (`XML` or `xml2`) to work with for the following questions. You do *not* need to provide solutions using both packages!

2a. *(1 pt)* It is always helpful to take a look at the data. One way to print the first listing is to use the command `xmlRoot(wsu_xml)[[1]]` (if you are using the `XML` package). Another way is to use either `xml_child(xml_root(wsu_xml))` or `xml_structure(xml_child(xml_root(wsu_xml)))` (if you are using the `xml2` package).

Observe that `limit` appears to be the capacity of the class in the form of a 4 digit text, and that `enrolled` appears to be the number of students currently enrolled in the class in the form of a 4 digit text. With that in mind, create two variables: `enrolled`, and `limit`. Make `enrolled` contain all the values for nodes with the name "enrolled". Make `limit` contain all the values for nodes with the name "limit". Print the first 10 values of each.

Hint: *Take a look at what you did in (1c).*

Keywords (XML package): *xmlSApply, xmlValue*

Keywords (xml2 package): *xml_contents, xmlValue*

Item(s) to submit:

- The code used for the problem.
- The output from printing the first 10 values of the resulting `enrolled` variable.
- The output from printing the first 10 values of the resulting `limit` variable.

2b. *(2 pt)* Make numeric versions of the variables `enrolled` and `limit`, and calculate the number of available spots (for all courses), and save them in a variable called `available_spots`. What is the maximum number of available spots for a course? The average number of available spots for a course? What value would put a class in the third quartile for number of available spots?

Keywords: *as.numeric*

Item(s) to submit:

- The code used for the problem.
- The output from printing the first 10 values of `available_spots`.
- The output from printing the maximum number of available spots for a course.
- The output from printing the average number of available spots for a course.
- The output from printing the number of students that would put a class in the third quartile for number of available spots.

2c. *(3 pt)* For this problem you can choose either option 1 or 2. Please note, if you opt for option 2, you *must* include a short explanation outlining what your graphic shows, and why.

1. We want to create a plot to get a better idea, in general, of how close to capacity the courses in our dataset are. To do so, first use the `plot` function, along with the arguments `type` and `main` to create a line graph for our `limit` variable. Use the `type` argument to specify a line graph, and use the `main` argument to give our plot the name "Enrolled vs Capacity". The result should be a black line graph with "Index" on the x-axis and "limit" on the y-axis.

Now, on a new line, use the `lines` function with the `col` argument to add a red line showing the `enrolled`.

Write down any observation you have about the plot, and export the final plot as a png and include it in your submission. This is what the plot should look like.

Hint: *Here is a link on how to make line plots in R.*

Keywords: *plot, plot arguments: type & main, lines, lines argument: col*

Item(s) to submit:

- The code used for the problem.
- The plot as a PNG.

2. Create your own line plot, or other graphic, to creatively show a relationship between the `enroll` and `limit` variables.

Include a short explanation outlining what your graphic shows, and why. Export the final graphic as a jpeg and include it in your submission.

Hint: *Here is a link on how to make line plots in R.*

Keywords: *plot, plot arguments: type & main, lines, lines argument: col*

Item(s) to submit:

- The code used for the problem.
- The plot or graphic in PNG format.
- A short (1-2 sentence) explanation in a markdown cell outlining what your graphic shows, and why.

Project Submission:

Submit your solutions for the project at this URL: https://classroom.github.com/a/3WLtHgNj using the instructions found in the GitHub Classroom instructions folder on Blackboard.

Important note: Make sure you submit your solutions in both .ipynb and .pdf formats. We've updated our instructions to include multiple ways to convert your .ipynb file to a .pdf on scholar. You can find a copy of the instructions on scholar as well: `/class/datamine/data/spring2020/jupyter.pdf`. If for some reason the script does not work, just submit the .ipynb. Make sure you have your output calculated and displayed inside your notebook prior to submission.