Scraping Data with Rvest: Beginner's Guide

Christopher Nobblitt

4/3/2017

Summary

Data scraping is an imperative tool for the automation of gathering and cleaning massive amounts of data from online sources. This project examined collegiate basketball schedules from the 351 Division I Universities in the United States, which were scraped from ESPN's website using the R programming package, rvest. Geospatial data queried from Google Map's API was combined with the basketball data to create an interactive application that enables users of any statistical experience to visually explore the travel patterns of collegiate basketball teams. Because the associated data is generated using an automated data scraping process, future use of this application will incorporate new and updated information, an example of how researchers can save valuable time for data exploration and analysis. A guide for data scraping using R was created to assist undergraduate students for learning this tool.

Introducing *rvest*

Rvest is an R package that extracts data on webpages. For this example, we will 'scrape' ESPN's website and retrieve North Carolina State University's basketball schedule.

Here are the generalized steps for extracting data:

1. **Read** in the raw webpage code using *read_html()*
2. **Identify** the desired data, which are located in containers called *nodes*
3. **Extract** that data from the identified nodes

Install the *rvest* package and SelectorGadget (Chrome Web Store) to follow along with this example.

Step 1: Identify Source

First, we need to **identify a URL path** and let *rvest* save the webpage's source code. Let's save the XML source code for NC State University's basketball schedule on ESPN's website.

```
#Load the rvest package, installs if it hasn't already been installed
if ( !require(rvest) ) {
  install.packages( "rvest" )
  require( rvest )
}
```

```
## Loading required package: rvest
```

```
## Loading required package: xml2
```

```
#Store the url of the schedule's webpage
url <- "http://www.espn.com/mens-college-basketball/team/schedule/_/id/152/nc-state-wolfpack"

#Save the source code in variable 'page'
page <- read_html(url)

print(page)
```

```
## {xml_document}
## <html xmlns:fb="http://www.facebook.com/2008/fbml">
## [1] <head>\n  <script src="http://cdn.espn.com/sports/optimizely.js"/>\n ...
## [2] <body class="ncb {sportId:41, teamId:152} clubhouse ncaa ncaa-152" d ...
```

The webpage is written in XML, evidenced by the first line of output. Fortunately, the ***read_html()*** **function reads HTML and XML interchangeably**.

Step 2: Identify Desired Nodes

Next, we must **identify where the data is** on the webpage. We do this using **CSS selectors**. CSS selectors describe how the elements, or **nodes**, on a webpage are structured. For more detailed information, read the info on W3schools.com.

CSS selectors narrow down where the desired data is on the webpage. Open the NCSU basketball schedule and try to use the SelectorGadget extension to select the table.

Unfortunately, SelectorGadget struggles to find a "valid path" for the table. All this means is that SelectorGadget could not identify a CSS selector, but do not worry. Sometimes, SelectorGadget has trouble with certain webpages - luckily, we can work around that problem.

Right-click any cell in the table and select ***Inspect***. The page's XML code appears in a window.

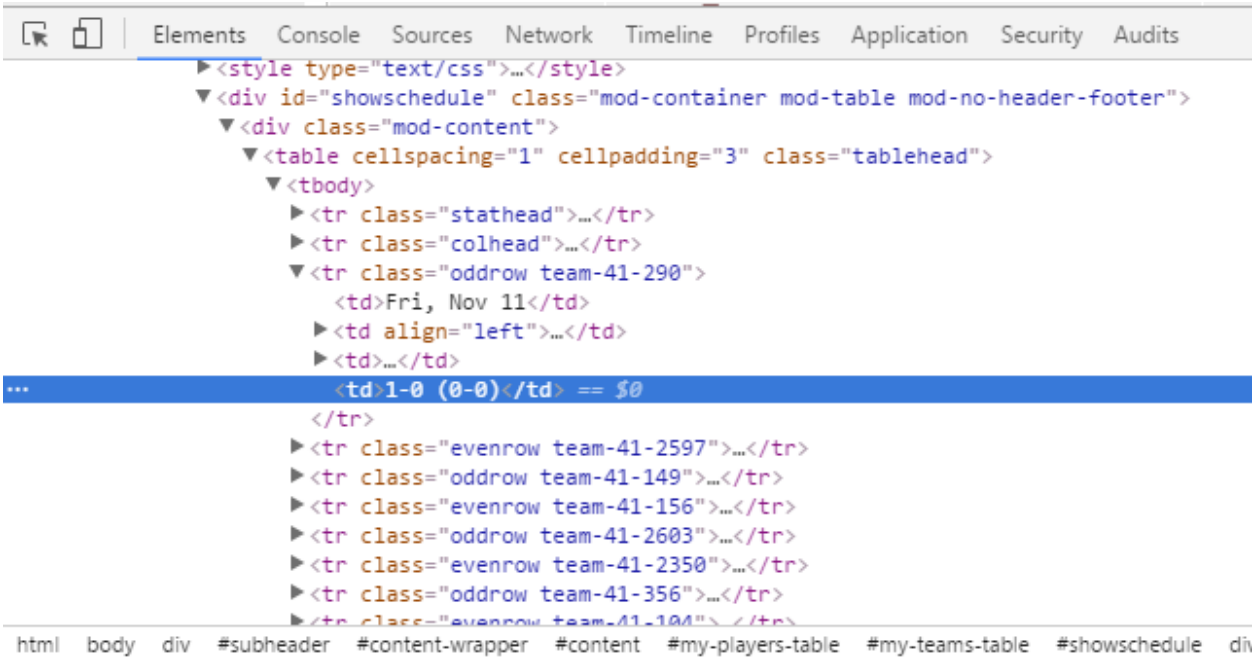

Figure 1: Source Inspection Window

The selected line of code corresponds to the cell that you right-clicked.

We must find the line of code that defines the node for the desired table. HTML and XML nest nodes within each other. A ***parent node*** is the node in which a certian node is nested within. The node inside the parent node is called a ***child node***. Much like humans, nodes can both be a parent node and a child node. They may have multiple children nodes as well. Looking back at the inspection window, we can see how each child node is nested inside its parent node. My selection in *Figure 1* has the parent node:

<**td**>. . . <td>

And that node's parent node is:

<**tr** class="oddrow team-41-290">

The leading word of each node, in bold above, is that node's CSS selector.

Notice that when you select a node in the code source inspection window, **that node is also selected *in your browser***. Start at the table cell you selected and select its parent node. Select that node's parent node and keep selecting parent nodes in this manner until you select a line of code that looks like this:

<**table** cellspacing="1" cellpadding="3" class ="tablehead">

The most important thing to notice is the leading term **table**. That is the CSS selector! Right click the line of code in the inspection window and select ***Copy > Copy selector***. The CSS selector is copied to your clipboard as *#showschedule > div > table*. Return to your R script:

```
  #Extract the specified nodes from the web page
    nodes <- html_nodes(page, "#showschedule > div > table")

    print(nodes)
```

```
## {xml_nodeset (1)}
## [1] <table cellspacing="1" cellpadding="3" class="tablehead">\n  <tr cla ...
```

html_nodes() returns an object with every node on the webpage which your specified selector matches. If there were multiple tables, *html_nodes* would return each of them in a list.

Step 3: Extract Data

The *nodes* variable now contains the table node with the data we are interested in. To extract the desired data, we have to use the appropriate *rvest* function. In this case, we are extracting a structured table, so we should use *html_table()*, which returns a list of data frames. Other extraction functions include:

- html_text()
- html_name()
- html_form()

```
    ncsuTable <- html_table(nodes, header = TRUE)
    head(ncsuTable[[1]], n = 4)
```

```
##   2016-17 NC State Wolfpack Schedule 2016-17 NC State Wolfpack Schedule
## 1                               DATE                           OPPONENT
## 2                       Fri, Nov 11                 vsGeorgia Southern
## 3                       Sun, Nov 13                 vsSt Francis (BKN)
## 4                       Fri, Nov 18                        vsMontana*
##   2016-17 NC State Wolfpack Schedule 2016-17 NC State Wolfpack Schedule
## 1                       RESULT/TIME                      RECORD/TICKETS
## 2                            W81-79                           1-0 (0-0)
## 3                            W86-61                           2-0 (0-0)
## 4                            W85-72                           3-0 (0-0)
```

There you go, we officially have our data! Albeit, it is fairly messy – now you must clean it!

Cleaning the data

Cleaning the extracted data is one of the most time consuming parts of data scraping. The following tools are incredibly helpful and are highly recommended:

- **dplyr** – A package for manipulating data frames in a SQL-like manner.
- **stringr** – A package for manipulating character data.
- **lubridate** – A package for time/date data.

- **Regex** – Regular Expressions (*Regex*) is a pattern recognition, psuedo-language for character data. It is invaluable when working with semi-structured character data. Try out the interactive RegexOne tutorial to learn and practice regex structure.

```
    #Put the extracted table in the schedule variable
    schedule <- ncsuTable[[1]]

    #use the first row as the column names
    names(schedule) <- schedule[1,]

    names(schedule)
```

```
## [1] "DATE"           "OPPONENT"       "RESULT/TIME"    "RECORD/TICKETS"
```

```
    #get rid of the first row, which had the column names.
    schedule <- schedule[-c(1),]
```

First column - Parse Dates

```
    #load the lubridate library
    library("lubridate")
```

```
##
## Attaching package: 'lubridate'
```

```
## The following object is masked from 'package:base':
##
##     date
```

```
    # Convert first column into date type
    # %a = abbreviated weekday
    # %b = abbreviated month
    # %d = day of month
    schedule$DATE <- as.Date(schedule$DATE, format = "%a, %b %d")
```

`DATE` is now homogenous! (*Lubridate*)[https://cran.r-project.org/web/packages/lubridate/lubridate.pdf] is an excellent date/time package, making it easy to do mathmatical operations or extracting secondary information, such as whether the date fell on a Monday.

References

NCSU Basketball Schedule http://www.espn.com/mens-college-basketball/team/schedule/_/id/152/nc-state-wolfpack

CSS Selectors explained by W3schools. http://www.w3schools.com/cssref/css_selectors.asp

SelectorGadget. https://chrome.google.com/webstore/detail/selectorgadget/mhjhnkcfbdhnjickkkdbjoemdmbfginb?hl=en

RegexOne. https://regexone.com/lesson/introduction_abcs