Five easy steps for *scraping data* from web pages.

As published in Benchmarks RSS Matters, November 2013

http://web3.unt.edu/benchmarks/issues/2013/11/rss-matters

Jon Starkweather, PhD

Jon Starkweather, PhD
jonathan.starkweather@unt.edu
Consultant
Research and **S**tatistical **S**upport

http://www.unt.edu

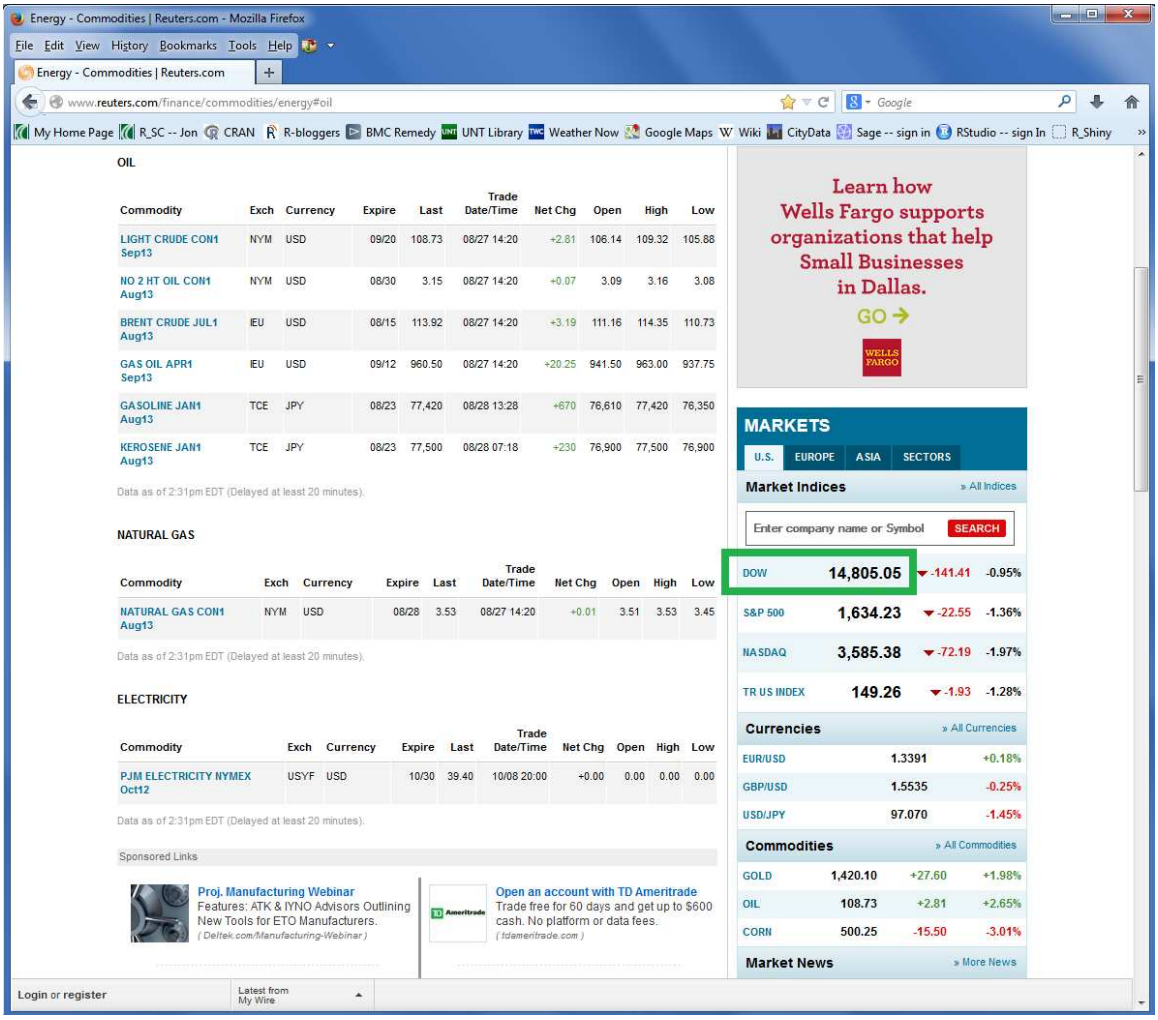
http://www.unt.edu/rss

RSS hosts a number of "Short Courses".
A list of them is available at:
http://www.unt.edu/rss/Instructional.htm

Those interested in learning more about R, or how to use it, can find information here:
http://www.unt.edu/rss/class/Jon/R_SC

Five easy steps for *scraping data* from web pages.

In a perfect world, all data would be easily available to everyone as comma separated values (CSV) files. Unfortunately, Earth is not yet a perfect world. Occasionally, some interesting data is unavailable as a CSV download, but is available and / or displayed on a web page. The term scraping data refers to the process of parsing the HTML source code of an available web page in order to extract, retrieve, or scrape some specific data from the web page. In truth, the title of this article is a bit misleading. The functions used to scrape data are fairly straightforward and easy to use; but, there is a significant assumption when using them. The time consuming assumption isyou need to know where the desired data is located among the lines of HTML code of the web page you are scraping. However, once the line, or lines, have been identified; the R script used to scrape the data and put it into a manageable R format is very easy to use and can be reused over time. The example below illustrates the process of scraping the DOW Jones Industrial average from the Reuters Commodities (2013) website; specifically, the following web page: `http://www.reuters.com/finance/commodities/energy#oil` (displayed below). Keep in mind; scraping data does not require the target web page to be open in a browser. In fact, the script in this article, and other scripts like it, will work without ever opening a browser - only an internet connection is required. Of course, the target web page needs to be public (i.e. not behind a log in or encrypted).



The specific data retrieved in this example will be the DOW, which is listed in the small Markets table

on the right side of the page [screen capture] above and marked with the green rectangle. At the time of writing (August 27, 2013; the number changes throughout the day), the number desired is 14805.05.

Step 1: Get the web page.

Obviously, the first step is to know the URL for the page you will be scraping. The URL for this example is stated above. The function used for importing an HTML page into R is 'readLines' which is available in the 'base' package (installed with every installation of R). There are other functions which can accomplish this task, such as those found in the RCurl[1] package, which allow more complex control and parsing of HTML code.

The key argument, indeed the only necessary argument, is the connection (con) which specifies the location of the file to be imported. Below we use the 'readLines' function and the URL (as the connection to the file / web page) to import the web page and name that object 'r.page' for Reuters page.

```
r.page <- readLines("http://www.reuters.com/finance/commodities/energy#oil"
```

Given the length of the page (in number of lines of HTML source code), it may take a significant amount of time to find the line number (or row) of the page which contains the data of interest.

```
length(r.page)
[1] 2005
```

Some web pages are fairly simple and the specific data of interest can be found relatively easily using the 'which' function and other common R indexing functions. Unfortunately, the Reuters page used in this example is fairly complex and contains over 2000 lines of HTML source code. It can be helpful with complex pages, such as this example, to copy and paste the source HTML code (here, the 'r.page' object in R) into a text editor (most have line numbers along the left edge and do not *text wrap* each line) so that the data of interest can be identified by the row of HTML code where it is found.

Step 2: Scraping the line(s) of data.

We know from previously looking through the HTML source code that we need line 917 (of the 2005 lines) to get the DOW number (14805.05). So, we retrieve this line of HTML using its line number and assign the line of HTML to the object 'dow.data'.

```
dow.data <- r.page[917]
dow.data
[1] "\t\t\t\t\t\t<td class=\"data size8\" valign=\"middle\">14,805.05</td>"
```

Keep in mind; we could specify multiple lines to retrieve, creating a vector of lines with each line containing some data of interest. As an example, we might also be interested in the price of Gold (1420.10 in the screen capture above). In which case, we would also scrape, or retrieve, line 1005 of the r.page object and add it to the dow.data object - would then contain both lines of HTML code (i.e. one line for

[1] http://cran.r-project.org/web/packages/RCurl/index.html

the Dow and one line for Gold).

Step 3: Reducing the HTML to isolate the data.

The next obvious step is to replace and / or remove the HTML code which is not needed, in order to isolate the actual data of interest. It is important to note that the entire line retrieved (and displayed above and below) is a character string. Therefore, we need a special function which will read each place or character of the line. To do this, we use the 'gsub' function, also from the 'base' package. This function is extremely useful; it allows fine control for parsing character strings and replacing (substituting) patterns or specific characters with any other value or no value at all. If no value is specified (as the replacement), then the pattern or character specified in the 'pattern' argument is simply removed.

```
new.line.1 <- dow.data; new.line.1
[1] "\t\t\t\t\t\t<td class=\"data size8\" valign=\"middle\">14,805.05</td>"
```

Notice there is one other 'number' in the line of HTML which we DO NOT want ("...size8..."), so we first replace or substitute that pattern of character string with the letter "a". Here we are using "a" as a replacement value (the choice is fairly arbitrary). Also, below we are creating a 'new line' (new.line.2) rather than simply writing over the old line (new.line.1) - again, that choice is rather arbitrary.

```
new.line.2 <- gsub(pattern = "size8", replacement = "a", x = new.line.1,
      ignore.case = TRUE, perl = FALSE, fixed = FALSE, useBytes = FALSE)
new.line.2
[1] "\t\t\t\t\t\t<td class=\"data a\" valign=\"middle\">14,805.05</td>"
```

Next, we need to remove all the remaining HTML symbols, including the comma (but not the decimal point). Notice the space between each element of the pattern we are specifying. Also notice the last element of the pattern is "a-z" which indicates all letters. Here, we are not actually substituting, but rather removing all the elements of the pattern; because, we are using no value as the replacement - there is nothing between the quotation marks in the replacement argument below.

```
new.line.3 <- gsub(pattern = "([\t =\\ <\\ >\\ \" /td , a-z])",
     replacement = "", x = new.line.2, ignore.case = TRUE,
     perl = FALSE, fixed = FALSE, useBytes = FALSE)
new.line.3
[1] "14805.05"
```

It is important to realize that the "number" returned from the code above (i.e. new.line.3) is not actually a 'number.' It is still formatted as character string; that is why it has quotation marks around it.

Step 4: System time and a data frame.

Now, we can create a data frame in which to put our new line of DOW data as well as the time stamp indicating when we scraped or grabbed the data from the web page.

```
dow.df <- data.frame(matrix(rep(NA,3), ncol = 3))
names(dow.df) <- c("string.date","numeric.date","DOW")
dow.df
  string.date  numeric.date  DOW
1          NA           NA   NA
```

Below, we retrieve the date and time using the 'Sys.time' function and make sure to store the numeric version as well as the character string version.

```
dow.df[,1] <- Sys.time()
dow.df[,2] <- as.numeric(Sys.time())
```

Finally, we can convert our data into numeric and put it into the data frame we created.

```
dow.df[,3] <- as.numeric(new.line.3)
dow.df
          string.date  numeric.date       DOW
1 2013-08-27 14:58:10    1377633491  14805.05
```

Step 4: System time and a data frame.

We can then save or export the data by setting the working directory to the location we want to store the file and using the 'write.table' function.

```
setwd("C:/Users/jds0282/Desktop/")
write.table(dow.df, file = "dow.df.txt", sep = ",", na = "NA",
  dec = ".", row.names = TRUE, col.names = TRUE)
```

Conclusions

Keep in mind, although it may take significant effort to identify the line number of the data of interest, once the script has been written and checked, it can be used repeatedly (e.g. each day) to retrieve the data of interest (i.e. to build a time series data file). The only real problem which can occur is when the HTML source code is changed, in other words, if the web page author(s) update(s) the layout of the page. Then, of course, it would be necessary to verify the line number of the data of interest and re-check the script to make sure it returns the desired information.

As stated above, there are other ways of accomplishing what was accomplished in this article; the 'RCurl' package is apparently quite popular. All of the functions used in this article are available with a base install of R - the functions are available in the 'base' package. For more information on what R can do, please visit the Research and Statistical Support Do-It-Yourself Introduction to R[2] course website. An Adobe.pdf version of this article can be found here[3].

[2] http://www.unt.edu/rss/class/Jon/R_SC/
[3] http://www.unt.edu/rss/rssmattersindex.htm

Until next time; *"information wants to be free"*...

References & Resources

Lang, D. T. (2013). Package RCurl. Package documentation available at: `http://cran.r-project.org/web/packages/RCurl/index.html` and further information available at: `http://www.omegahat.org/RCurl/`

ProgrammingR.com (2013) Webscraping using readLines and RCurl. Available at: `http://www.programmingr.com/content/webscraping-using-readlines-and-rcurl/`

Reuters. (2013). Commodities: Energy. Retrieved on August 27, 2013 from: `http://www.reuters.com/f`

This article was last updated on November 5, 2013.

This document was created using LaTeX