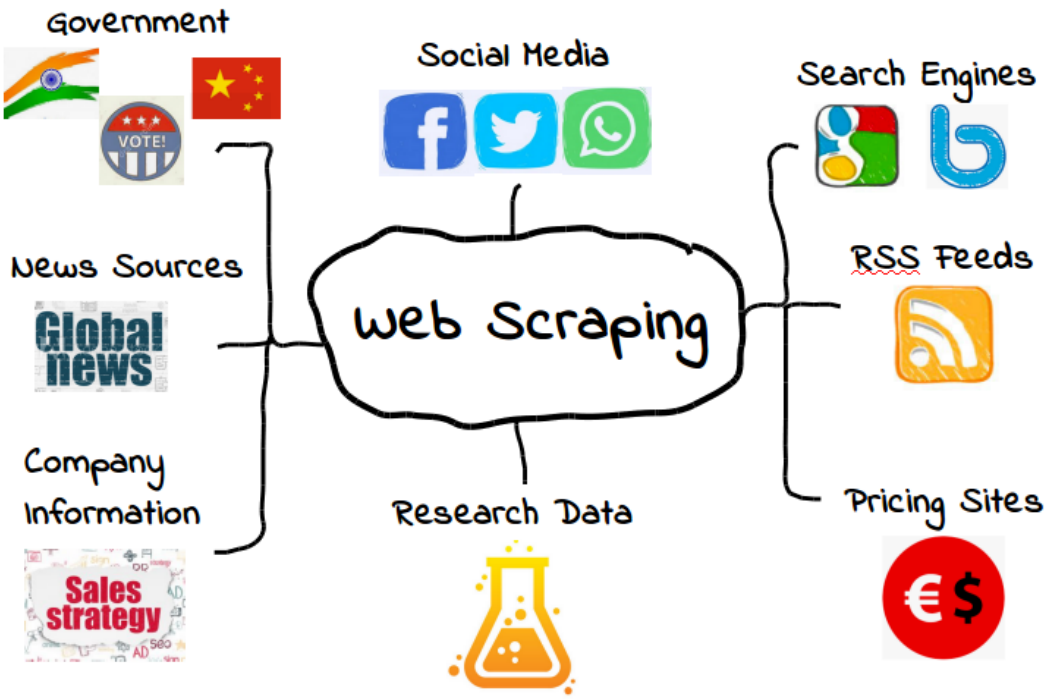
Applied R Code
Data Science for Immediate Application

Web Scraping in R

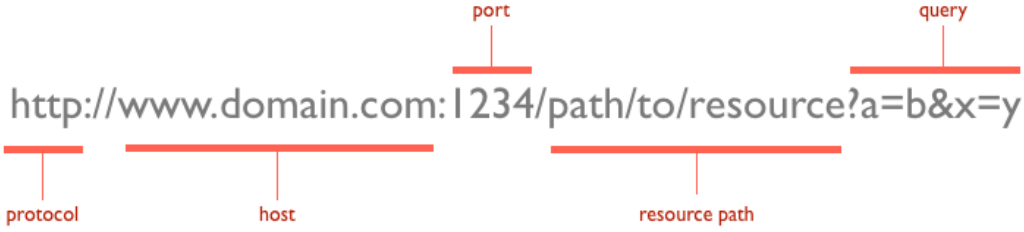
Content



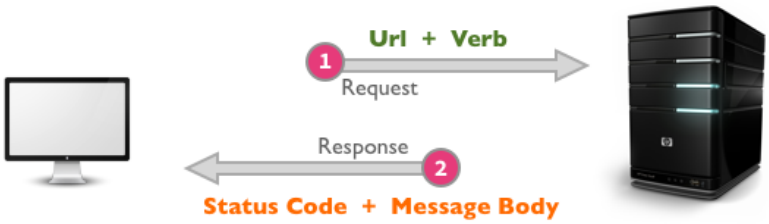
The world-wide web presents enormous amounts of data. Unfortunately, the majority of the data is not directly available for download. In response, web scraping exploits indirect means to harvest data from websites. In practice, web scrapping is not unique and is totally legal. For example, web browsers rely on the Hypertext Transfer Protocol (HTTP) to fetch data and so does web scrapping. The difference with web scrapping is that the user retrieves, selects and extracts website content and data intended for browser display. This article shows how web scraping works and presents tools available in the R programming language for both manual and automated web-scraping.

What is Web Scraping?

Web scraping involves getting a web page and and extracting data from it. Specifically, a message is sent to fetch a web page using Uniform Resource Locators (URLs). A basic URL structure appears below:[1]



The request components include the protocol, which is typically http or https for secure communications. The host is a target website. The default port is 80, but one can be set explicitly, as shown here. The resource path specifies the server path to the data and the query is the data request action or verb.



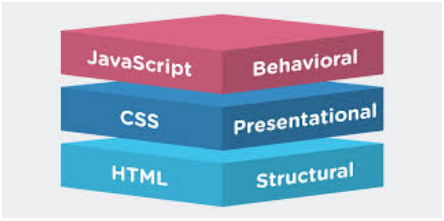
A web page request is a simple component of web scrapping. Next, a server response will deliver a status message with web page content. The web page content must be searched, be it manually or automatically. Hence, web page crawling is a key feature of web scrapping. Next, the web page content will be parsed, extracted and reformatted.

Web scrapping, for example, might first retrieve a web page and then extract contact names and phone numbers. In another instance, the focus might be to grab a research data table or a collection of tables. Finally, another effort might seek to extract intel using text strings found in multiple news or social media reports.

Web Page Content – The basics

The response to a URL request is the web page delivered in a

HTML document message. HTML stands for HyperText Markup Language and defines the content and structure of a webpage.

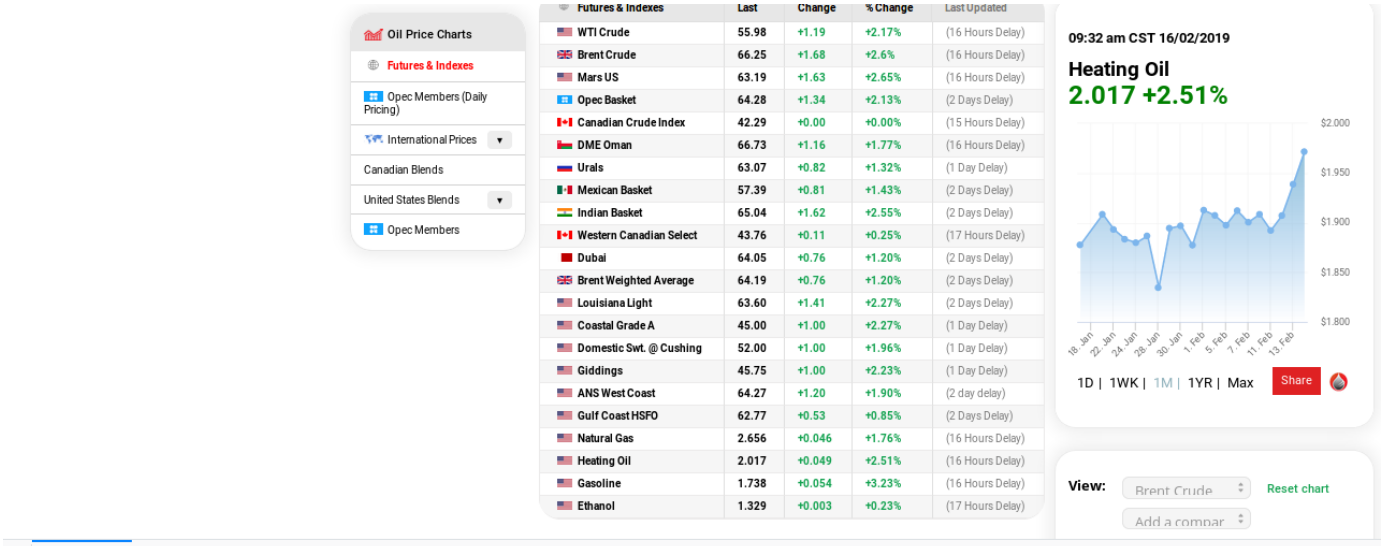


"Hyper Text" in HTML refers to hyperlinks that connect webpages to one another, either within a single website or between websites, to populate page content. Meanwhile, "markup" text includes special page elements or "CSS selectors" such as `<head>`, `<title>`, `<section>`, `<body>`, `<div>`, `<table>`, `<p>`, `<img>` and may others. Web crawling and scrapping involves finding and extracting from these elements as needed.

Other language technologies also populate the web page besides HTML. The technologies include CSS, which describe a webpage's presentation appearance and JavaScript for web page functionality. An analyst inspects all web page content using a web browser tool to find what parts will be scraped.

Web Page Inspection

Inspect web page content using web browser tools to find what parts will be scraped. For example, the following web site image (https://oilprice.com/oil-price-charts) shows petroleum product prices from around the world. The lower half of the image shows the web site HTML content in your web browser inspector (click to enlarge):



The inspector is launched in FireFox by right-clicking the data table and choosing “Inspect Element.” Other web browsers have selections to “View Page Source.” The next image shows the page source for the first table (click to enlarge):

If you don’t know HTML, this may look daunting! But look close. All the table text and data is there to be grabbed. Meanwhile, another essential tool for web page inspection is SelectorGadget. SelectorGadget is an open source tool that makes it easy to identify and define CSS content. Just install the browser extension as instructed. The tool simplifies inspection of complicated web sites to just a few point-and-click actions and it makes it a breeze to locate content to scrape.

Available R Packages

Several R packages are available for webscrapping:

Package Name	Retrieve?	Parse?	Must Know?
rvest	YES	YES	YES
RCurl	YES	NO	YES
XML	Limited	YES	YES
rjson	NO	YES	YES
RJSONIO	NO	YES	Optional
httr	YES	YES	Optional
xml2	NO	YES	Optional
selectr	NO	YES	Optional

`rvest` is a set of wrapper functions around the `xml2` and `httr` packages. The rest of the tutorial focuses on `rvest` exclusively. The main functions are:

- `read_html()`: read a webpage into R as XML (document and nodes)
- `html_nodes()`: extract pieces out of HTML documents using XPath and/or CSS selectors
- `html_attr()`: extract attributes from HTML, such as `href`
- `html_text()`: extract text content

Web Scraping Examples

The first step is to load the `rvest` package and to read the target web page:

```R
library(rvest)

url <- "https://oilprice.com/oil-price-charts"
webpage <- read_html(url)

webpage
{xml_document}
<html lang="en">
[1] <head>\n<script>\n  (function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||fun ...
[2] <body id="pagetop" class="oilprices loggedout">\n<!-- Google Tag Manager (noscript) -->\ ...
```

The first two lines above are the first HTML lines of the target web page. `rvest` has read or captured the entire HTML content of the target webpage. Next, the webpage nodes are read to extract to find data table content...the product names and prices. The rvest command `html_nodes()` is told to seek specific nodes as defined by CSS selectors used in the function calls. The selector string used was defined by SelectorGadget after clicking on a price or name element on the web page, The reaming command html_text() converts node content to text and other commands use basic R syntax to reshape the data for use in R:

```R
value <- webpage %>%
     html_nodes( css = ".last_price") %>%
     html_text()

name <- webpage %>%
     html_nodes(., css = "td:nth-child(2)") %>%
     html_text() %>%
     .[c(41:72, 74, 76:77, 79, 81:82, 84:85, 87:89, 91, 94:96, 98:99, 101:103, 105:107, 109:112, 114:116, 118:120, 122
         126:128, 130:131, 133, 135:142, 145:160, 162:164, 166:167, 169:170, 172, 174,
         176:178, 180:182, 184:186, 188:189, 191, 193:196, 198:199, 201:213)]

prices <- tibble(name = name,
                 last_price = value)
prices
# A tibble: 138 x 2
    name                    last_price
    <chr>                   <chr>
  1 WTI Crude               55.98
```

```
 2 Brent Crude              66.25
 3 Mars US                  63.19
 4 Opec Basket              64.28
 5 Canadian Crude Index     42.29
 6 DME Oman                 66.73
 7 Urals                    63.07
 8 Mexican Basket           57.39
 9 Indian Basket            65.04
10 Western Canadian Select  43.76
11 Dubai                    64.05
12 Brent Weighted Average   64.19
13 Louisiana Light          63.60
14 Coastal Grade A          45.00
15 Domestic Swt. @ Cushing  52.00
```

The next example uses the pipe operator to link node parsing, data extracting and formatting. In this case, the extraction is on the 13th data table returned by the `html_table()` function. The table scraping extracts the spot price and price change data for US crude oil blends:

R

```
tables13 <- webpage %>%
     html_nodes("table") %>%
     html_table(fill = TRUE) %>%
     .[[13]] %>%
     .[c(3:18, 21:23, 26:27, 30:31, 34, 37, 40:42, 45:47, 50:52, 55:56, 59, 62:65,68:69, 72),] %>%
     `colnames<-`(c("index", "name", "last", "change", "percent", "updated")) %>%
     as.tibble() %>%
     separate(col = "percent", into = c("percent", "update"), sep = "\\(") %>%
     select(name, last, change, percent) %>%
     mutate(last = as.numeric(last)) %>%
     mutate(change = as.numeric(change)) %>%
     mutate(percent = as.numeric(substr(percent,1,5)))

table13
A tibble: 44 x 4
name                     last change percent
   <chr>                  <dbl>  <dbl>   <dbl>
 1 West Texas Sour         49.54   1.18   2.44
 2 West Texas Intermediate 52.04   1.18   2.320
 3 Upper Texas Gulf Coast  39.84   1.18   3.05
 4 Texas Gulf Coast Light  50.54   1.18   2.39
 5 South Texas Sour        45.93   1.18   2.64
 6 North Texas Sweet       49.5    0.25   0.51
 7 North Texas Sour        39.91   0.87   2.23
 8 Eagle Ford Pipeline     52.04   1.18   2.320
 9 Eagle Ford Condensate   51.04   1.18   2.37
10 Eagle Ford              53.49   1.18   2.260
11 Tx. Upper Gulf Coast    45.75   1      2.23
12 South Texas Light       45.75   1      2.23
13 W. Tx./N. Mex. Inter.   52      1      1.96
14 South Texas Heavy       45.5    1      2.25
15 W. Cen. Tx. Inter.      52      1      1.96
16 East Texas Sweet        49.25   1      2.070
17 Arkansas Sweet          47.75   0.25   0.53
18 Arkansas Sour           46.75   0.25   0.54
19 Arkansas Ex Heavy       42.75   0.25   0.59
20 Buena Vista             65.21   1.11   1.73
21 Midway-Sunset           60.53   1.11   1.87
22 Williston Sweet         45.75   0.75   1.67
23 Williston Sour          41.86   0.75   1.82
24 Utah Black Wax          39.11   0.51   1.32
25 Four Corners            49.68   0.5    1.02
26 Colorado D-J Basin      50      0.75   1.52
27 Colorado South East     41.25   0.25   0.61
28 Colorado West           47.41   0.51   1.09
29 NW Kansas Sweet         42.25   0.25   0.6
30 SW Kansas Sweet         42.75   0.25   0.59
31 South Central Kansas    46.5    2      4.49
32 Delhi/N. Louisiana      49      1      2.08
33 South Louisiana         50.5    1      2.02
34 North Louisiana Sweet   47.75   0.25   0.53
35 Michigan Sour           44      1      2.33
```

53	36 Michigan Sweet	48.75	1	2.09
54	37 Nebraska Sweet	46.51	0.51	1.11
55	38 Oklahoma Sweet	52.04	1.18	2.320
56	39 Oklahoma Sour	37.5	0.25	0.67
57	40 Western Oklahoma Swt.	51.25	1	1.99
58	41 Oklahoma Intermediate	51	1	2
59	42 Wyoming General Sour	42.1	0.75	1.81
60	43 Wyoming General Sweet	48.75	0.75	1.56
61	44 Central Montana	46.98	0.51	1.1

1. Image from https://code.tutsplus.com/tutorials/http-the-protocol-every-web-developer-must-know-part-1--net-31177 ↩

Applied R Code

Proudly powered by WordPress.