Professor Bear :: Web Scraping

Professor Bear

Web Scraping

Web scraping (web harvesting or web data extraction) is a computer software technique of extracting information from websites. This is accomplished by either directly implementing the Hypertext Transfer Protocol (on which the Web is based), or embedding a web browser.

Web scraping uses scripts to sift through a web page and gather the data such as text, images or urls. Web sites are written using HTML, which means that each web page is a structured document. How easy it is to parse data from a web page depends on how well structured that page is. HTML tables, well named Cascading Style Sheets (CSS) and well structured HTML can make the processing of parsing the data one wants much easier.

Poorly structured HTML, logining into web forms, changing content depending of the DNS request can make the process of scraping much more difficult or impossible.

Scraping a page involves these steps:

1. Making a DNS request (i.e. making a domain name system (DNS) request of a url such as https://www.google.com/). Note that making a request of a non-existent resource such as https://www.google.bear/ will result in an error. Websites can also block certain types of traffic and will often try to block web robots other than the big search engines.

2. Fetching the HTML.

3. Putting the HTML in to a searchable data structure such as a DOM object.

4. Searching for patterns that match the data desired.

5. Putting that data in to a database.

The process of automatically crawling many pages is called web crawling or web robots.

Additional packages needed

To run the code in you may need additional packages.

- If necessary install `ggplot2` package.

```
install.packages("ggplot2");install.packages("rvest");
```

```
library(ggplot2)
library(rvest)

## Loading required package: xml2

library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(stringr)
```

Packages for Web Scraping

We will be using the following packages for web scraping and comparing them. Typically one wouldn't use so many packages that do similar things but here we want to explore how various approaches work.

- rvest package
- SelectorGadget tool
- rvest and SelectorGadget guide
- Awesome tutorial for CSS Selectors
- rvest
- Import.io
- Google Chrome Webscraper
- ScaperHub pacman

Amazon Reviews, Wikipedia, Indeed.com

We will be scraping data from Amazon Reviews, Wikipedia, Indeed.com

Document Object Model (DOM)

The Document Object Model (DOM) is a cross-platform and language-independent application programming interface that treats an HTML, XHTML, or XML document as a tree structure wherein each node is an object representing a part of the document.

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

Document Object Model (DOM)

To effectively parse a web page we need tools for parsing text and well as parsing DOM objects.

HTML tags/nodes

HTML elements are written with a start tag, an end tag, and with the content in between: `<tag>content</tag>`. The tags which typically contain the textual content we wish to scrape.

Common tags include:

- `<h1>`, `<h2>`,...,`<h6>`: Largest heading, second largest heading, etc.
- `<p>`: Paragraph elements
- `<ul>`: Unordered bulleted list
- `<ol>`: Ordered list
- `<li>`: Individual List item
- `<div>`: Division or section
- `<href>`: Url or hypertext link
- `<span>`: Table or inline section
- `<table>`: Table

For example, if your data is wrapped with the HTML paragraph tag `<p>` and `</p>` we can use that structure to find our data.

```
<p>
This is some text in a paragraph.
</p>
```

However there are often many paragraphs within an HTML page and sometimes we need more stucture to identify the paragraph with want. This is often provide by within tag attributes such as CSS styling. If our paragraph had a *.class selector* defining a paragraph as a *professor-name* we might leverage that structure.

```
<p class="professor-name">
Nik Bear Brown
</p>
```

To extract text or urls from a bls.gov.htm we often use structure provided by HTML tags. This means we often write the script for that type of page after looking at the internal structure of a websites pages.

Cascading Style Sheets (CSS)

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language.[1] Although most often used to set the visual style of web pages and user interfaces written in HTML and XHTML, the language can be applied to any XML document,

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

Cascading Style Sheets (CSS)

Scraping Amazon Reviews with rvest and CSS

Lets scrape reviews from R Cookbook by Paul Teetor. The reviews for the book are in the review section

```
url <- "https://www.amazon.com/Cookbook-OReilly-Cookbooks-Paul-Teetor/product-reviews/0596809158/"
```

The rvest package can download the url.

```
htm <- read_html(url)
htm
```

```
## {xml_document}
## <html class="a-no-js" data-19ax5a9jf="dingo">
## [1] <head>\n  <script><![CDATA[var aPageStart = (new
Date()).getTime();] ...
## [2] <body class="a-m-us a-aui_51744-c a-aui_57326-c a-aui_72554-c a-
aui_ ...
```

If we look at the reviews on the reviews page we can see that they are contained within `<a>` tags (i.e. `<a>customer review</a>`).

```
<a data-hook="review-title" class="a-size-base a-link-normal review-
title a-color-base a-text-bold" href="/gp/customer-
reviews/R2YEY6SQEXVLK9/ref=cm_cr_arp_d_rvw_ttl?ie=UTF8&ASIN=0596809158"
>Very nice reference book for R</a>
```

So maybe we can extract the text within `<a>` tags to get our reviews?

rvest html_nodes()

Usage

```
html_nodes(x, css, xpath)
html_node(x, css, xpath)
```

Arguments

```
x - Either a document, a node set or a single node.
css, xpath Nodes to select. Supply one of css or xpath depending on
whether you want to use a css or xpath 1.0 selector.
```

The xpath selectors

XPath selectors

Chaining with XPath is a little trickier - you may need to vary the prefix you're using - // always selects from the root noot regardless of where you currently are in the doc. We will hold off on going in to XPath syntax and start with simpler HTML tags.

```
htm %>%
  html_nodes(xpath = "//center//font//b") %>%
  html_nodes(xpath = "//b")
```

Let's say we want all the `<a>` tags as we know our reviews are contained within `<a>` tags.

```
htm %>%
  html_nodes("a")

## {xml_nodeset (300)}
##  [1] <a id="nav-top"/>
##  [2] <a href="/ref=nav_logo/142-0993203-5708605" class="nav-logo-
link" t ...
```

```
##  [3] <a href="/gp/prime/ref=nav_logo_prime_join/142-0993203-5708605"
ari ...
##  [4] <a
href="http://www.amazon.com/gp/student/signup/info/ref=nav_swm_A ...
##  [5] <a href="/gp/site-directory/ref=nav_shopall_btn/142-0993203-
5708605 ...
##  [6] <a href="/gp/customer-preferences/select-
language/ref=topnav_lang/1 ...
##  [7] <a href="/gp/navigation/redirector.html/ref=sign-in-
redirect/142-09 ...
##  [8] <a href="/gp/css/order-history/ref=nav_nav_orders_first/142-
0993203 ...
##  [9] <a href="/gp/product/B00DBYBNEE/ref=nav_prime_try_btn/142-
0993203-5 ...
## [10] <a href="/gp/cart/view.html/ref=nav_cart/142-0993203-5708605"
aria- ...
## [11] <a href="/gp/yourstore/home/ref=nav_cs_ys/142-0993203-5708605"
data ...
## [12] <a href="/gp/goldbox/ref=nav_cs_gb/142-0993203-5708605"
class="nav- ...
## [13] <a href="/b/ref=nav_cs_gc_registry/142-0993203-
5708605?ie=UTF8&amp; ...
## [14] <a href="/b/ref=nav_cs_sell/142-0993203-
5708605?_encoding=UTF8&amp; ...
## [15] <a href="/gp/help/customer/display.html/ref=nav_cs_help/142-
0993203 ...
## [16] <a href="/gp/help/customer/accessibility/142-0993203-5708605"
aria- ...
## [17] <a class="a-link-normal" href="/Cookbook-Analysis-Statistics-
Graphi ...
## [18] <a href="javascript:void(0)" class="a-popover-trigger a-
declarative ...
## [19] <a class="a-size-small a-link-normal 5star" title="5 star"
href="/C ...
## [20] <a class="a-size-small a-link-normal" href="/Cookbook-Analysis-
Stat ...
## ...
```

Note Bien: Pipes %>% in R

A side note, the syntax `htm %>% html_nodes("a")` may seem odd. The `%>%` operator is called a pipe.

he *%>%* (a.k.a: pipe) operator in R lets you transform nested function calls into a simple pipeline of operations that's easier to write and understand.

You can use the %>% operator with standard R functions or your own functions. The rules are simple: the object on the left hand side is passed as the first argument to the function on the right hand side.

For example:

```
my.data %>% my.function is the same as my.function(my.data)
my.data %>% my.function(arg=value) is the same as my.function(my.data,
arg=value)
```

In our example the syntax below can read as pass the data htm to the function html_nodes() with the paramater "a."

```
htm %>% html_nodes("a")
```

This is exactly equivelent to writing.

```
html_nodes(htm, "a")

## {xml_nodeset (300)}
##  [1] <a id="nav-top"/>
##  [2] <a href="/ref=nav_logo/142-0993203-5708605" class="nav-logo-
link" t ...
##  [3] <a href="/gp/prime/ref=nav_logo_prime_join/142-0993203-5708605"
ari ...
##  [4] <a
href="http://www.amazon.com/gp/student/signup/info/ref=nav_swm_A ...
##  [5] <a href="/gp/site-directory/ref=nav_shopall_btn/142-0993203-
5708605 ...
##  [6] <a href="/gp/customer-preferences/select-
language/ref=topnav_lang/1 ...
##  [7] <a href="/gp/navigation/redirector.html/ref=sign-in-
redirect/142-09 ...
##  [8] <a href="/gp/css/order-history/ref=nav_nav_orders_first/142-
0993203 ...
##  [9] <a href="/gp/product/B00DBYBNEE/ref=nav_prime_try_btn/142-
0993203-5 ...
## [10] <a href="/gp/cart/view.html/ref=nav_cart/142-0993203-5708605"
aria- ...
## [11] <a href="/gp/yourstore/home/ref=nav_cs_ys/142-0993203-5708605"
data ...
## [12] <a href="/gp/goldbox/ref=nav_cs_gb/142-0993203-5708605"
class="nav- ...
## [13] <a href="/b/ref=nav_cs_gc_registry/142-0993203-
5708605?ie=UTF8&amp; ...
## [14] <a href="/b/ref=nav_cs_sell/142-0993203-
5708605?_encoding=UTF8&amp; ...
## [15] <a href="/gp/help/customer/display.html/ref=nav_cs_help/142-
0993203 ...
## [16] <a href="/gp/help/customer/accessibility/142-0993203-5708605"
aria- ...
## [17] <a class="a-link-normal" href="/Cookbook-Analysis-Statistics-
Graphi ...
## [18] <a href="javascript:void(0)" class="a-popover-trigger a-
declarative ...
```

```
## [19] <a class="a-size-small a-link-normal 5star" title="5 star"
href="/C ...
## [20] <a class="a-size-small a-link-normal" href="/Cookbook-Analysis-
Stat ...
## ...
```

While `html_nodes(htm, "a")` may seem simpler than `htm %>% html_nodes("a")`, try passing a function to a function to a function to a function to a function to a function. You'll see how much more readable the pipe syntax is as the number of operations increases.

So you want to next fucntions by chain subsetting this is far easier to read and write using the pipe syntax.

```
htm %>% html_nodes("table") %>% html_nodes("td") %>% html_nodes("p")
```

CSS Selectors

Selecting html_nodes() with the paramater "a" gives us a lot more irrelevant data than we want. While we could use regular expressions to filter our result set we can also use CSS selectors (if they are in the HTML). Let's look at the attributes within the `<a>` tags for reviews.

```
<a data-hook="review-title" class="a-size-base a-link-normal review-
title a-color-base a-text-bold" href="/gp/customer-
reviews/R2YEY6SQEXVLK9/ref=cm_cr_arp_d_rvw_ttl?ie=UTF8&ASIN=0596809158"
>Very nice reference book for R</a>
```

There are several classes in the review `class="a-size-base a-link-normal review-title a-color-base a-text-bold"`. Of these `.review-title` seems most specific to a review. So let's try `.review-title` to find only reviews.

```
htm %>%
  html_nodes(".review-title")

## {xml_nodeset (12)}
##  [1] <span data-hook="review-title" class="a-size-base review-title
a-te ...
##  [2] <span data-hook="review-title" class="a-size-base review-title
a-te ...
##  [3] <a data-hook="review-title" class="a-size-base a-link-normal
review ...
##  [4] <a data-hook="review-title" class="a-size-base a-link-normal
review ...
##  [5] <a data-hook="review-title" class="a-size-base a-link-normal
review ...
##  [6] <a data-hook="review-title" class="a-size-base a-link-normal
review ...
##  [7] <a data-hook="review-title" class="a-size-base a-link-normal
review ...
##  [8] <a data-hook="review-title" class="a-size-base a-link-normal
```

```
review ...
##  [9] <a data-hook="review-title" class="a-size-base a-link-normal
review ...
## [10] <a data-hook="review-title" class="a-size-base a-link-normal
review ...
## [11] <a data-hook="review-title" class="a-size-base a-link-normal
review ...
## [12] <a data-hook="review-title" class="a-size-base a-link-normal
review ...
```

However we are getting both `<span>` and `<a>` with the class `.review-title` so let's try `.a-color-base` instead.

```
htm %>%
  html_nodes(".a-color-base")

## {xml_nodeset (10)}
##  [1] <a data-hook="review-title" class="a-size-base a-link-normal
review ...
##  [2] <a data-hook="review-title" class="a-size-base a-link-normal
review ...
##  [3] <a data-hook="review-title" class="a-size-base a-link-normal
review ...
##  [4] <a data-hook="review-title" class="a-size-base a-link-normal
review ...
##  [5] <a data-hook="review-title" class="a-size-base a-link-normal
review ...
##  [6] <a data-hook="review-title" class="a-size-base a-link-normal
review ...
##  [7] <a data-hook="review-title" class="a-size-base a-link-normal
review ...
##  [8] <a data-hook="review-title" class="a-size-base a-link-normal
review ...
##  [9] <a data-hook="review-title" class="a-size-base a-link-normal
review ...
## [10] <a data-hook="review-title" class="a-size-base a-link-normal
review ...
```

This look better but one really ones the text between the tags so let's use `html_text()` function to extract this data:

```
review.titles <- htm %>%
  html_nodes(".a-color-base") %>%
  html_text()

review.titles

##  [1] "best intro to R I've found"
##  [2] "A High Quality Book On R Programming!"
##  [3] "A good reference book"
##  [4] "Very Practical -- Saved me tons of time!!!"
##  [5] "Useful reference"
##  [6] "A great book written very clearly"
##  [7] "Great for tasks where there should be an easy way, you just
don't know it"
##  [8] "The best book on R that I have read yet and ..."
##  [9] "Good"
## [10] "A simple and masterful book with many insights - better than
many textbooks on R"
```

One can grab the format (hardcover or paperback) with the class label `.a-size-mini.a-color-secondary`

```
htm %>%
  html_nodes(".a-size-mini.a-color-secondary") %>%
  html_text()

##  [1] "Format: Paperback"      "Format: Paperback"
##  [3] "Format: Paperback"      "Format: Paperback"
##  [5] "Format: Paperback"      "Format: Paperback"
##  [7] "Format: Kindle Edition" "Format: Paperback"
##  [9] "Format: Kindle Edition" "Format: Paperback"
```

We can remove the `Format:` with a regular expression.

```
formats <- htm %>%
  html_nodes(".a-size-mini.a-color-secondary") %>%
  html_text() %>%
  str_replace("Format:[ ]+", "")

formats

##  [1] "Paperback"      "Paperback"      "Paperback"      "Paperback"
##  [5] "Paperback"      "Paperback"      "Kindle Edition" "Paperback"
##  [9] "Kindle Edition" "Paperback"
```

Number of stars

One can grab the format (hardcover or paperback) with the class label `.review-rating` combined with the class ID `#cm_cr-review_list`

```
htm %>%
  html_nodes("#cm_cr-review_list .review-rating")

## {xml_nodeset (10)}
##  [1] <i data-hook="review-star-rating" class="a-icon a-icon-star a-
star- ...
##  [2] <i data-hook="review-star-rating" class="a-icon a-icon-star a-
star- ...
##  [3] <i data-hook="review-star-rating" class="a-icon a-icon-star a-
star- ...
##  [4] <i data-hook="review-star-rating" class="a-icon a-icon-star a-
star- ...
```

```
##  [5] <i data-hook="review-star-rating" class="a-icon a-icon-star a-
star- ...
##  [6] <i data-hook="review-star-rating" class="a-icon a-icon-star a-
star- ...
##  [7] <i data-hook="review-star-rating" class="a-icon a-icon-star a-
star- ...
##  [8] <i data-hook="review-star-rating" class="a-icon a-icon-star a-
star- ...
##  [9] <i data-hook="review-star-rating" class="a-icon a-icon-star a-
star- ...
## [10] <i data-hook="review-star-rating" class="a-icon a-icon-star a-
star- ...
```

Let's look at the text within the tags.

```
htm %>%
  html_nodes("#cm_cr-review_list .review-rating") %>%
  html_text()

##  [1] "5.0 out of 5 stars" "5.0 out of 5 stars" "4.0 out of 5 stars"
##  [4] "5.0 out of 5 stars" "4.0 out of 5 stars" "4.0 out of 5 stars"
##  [7] "5.0 out of 5 stars" "5.0 out of 5 stars" "5.0 out of 5 stars"
## [10] "5.0 out of 5 stars"
```

One can use regular expressions to extract the numbers 1-5.

```
htm %>%
  html_nodes("#cm_cr-review_list .review-rating") %>%
  html_text() %>%
  str_extract("[1-5].[0-9]")

##  [1] "5.0" "5.0" "4.0" "5.0" "4.0" "4.0" "5.0" "5.0" "5.0" "5.0"
```

If one wants the use the numbers as numbers then one needs to create a numeric vector from a character vector.

```
number.stars <- htm %>%
  html_nodes("#cm_cr-review_list .review-rating") %>%
  html_text() %>%
  str_extract("[1-5].[0-9]") %>%
  as.numeric(digits=2)

number.stars

##  [1] 5 5 4 5 4 4 5 5 5 5
```

One can grab the number of people that found a review useful with the class label `.review-votes` combined with the class ID `#cm_cr-review_list`

```
htm %>%
  html_nodes("#cm_cr-review_list .review-votes") %>%
  html_text()
```

```
## [1] "\n          59 people found this helpful.\n        "
## [2] "\n          2 people found this helpful.\n        "
## [3] "\n          6 people found this helpful.\n        "
## [4] "\n          121 people found this helpful.\n        "
## [5] "\n          2 people found this helpful.\n        "
## [6] "\n          One person found this helpful.\n        "
## [7] "\n          One person found this helpful.\n        "
```

One can use regular expressions to extract the numbers that are now 0-??? then convert the string to numeric.

```
number.helpful <- htm %>%
  html_nodes("#cm_cr-review_list .review-votes") %>%
  html_text() %>%
  str_extract("[0-9]+") %>%
  as.numeric()

number.helpful

## [1]  59   2   6 121   2  NA  NA
```

One can then aggregate the data in to a data frame.

```
htm.data <- data_frame(review.titles, formats, number.stars,
number.helpful)

htm.data
```

Multiple pages

One can get multiple pages of reviews by exploting the fact that Amazon expsoses the page number in the url with the `&pageNumber=` parameter. The issue is without looking at a page we don't know how many pages of reviews that a book has.

Typically scraping multiple pages would use a while loop which adds a page every step (e.g. `&pageNumber=2`, `&pageNumber=3`, `&pageNumber=4`, etc.) and check the number of reviews returned on each page. As soon as zero reviews are returned one would break out of the loop and stop.

Scraping a list on a web page

Let's scrap a list of machine learning concepts from the Wikipedia page https://en.wikipedia.org/wiki/List_of_machine_learning_concepts.

```
scraping_wiki <-
read_html("https://en.wikipedia.org/wiki/List_of_machine_learning_conce
pts")
li_text <- scraping_wiki %>%
        html_nodes("li") %>%
        html_text()
length(li_text)
```

```
## [1] 465

li_text[1:5]

## [1] "1 What type of thing is machine learning?"
## [2] "2 Branches of machine learning\n2.1 Subfields\n2.2 Cross-
disciplinary fields\n"
## [3] "2.1 Subfields"
## [4] "2.2 Cross-disciplinary fields"
## [5] "3 Machine learning hardware"
```

Scraping HTML tables

HTML tables are often a source of data that one would one to scrape.

```
bls.gov.htm <- read_html("http://www.bls.gov/web/empsit/cesbmart.htm")

bls.gov.tbls <- html_nodes(bls.gov.htm, "table")

head(bls.gov.tbls)

## {xml_nodeset (6)}
## [1] <table id="main-content-table">\n  <tr><td id="secondary-nav-
td">&#1 ...
## [2] <table id="Table1" class="regular" cellspacing="0"
cellpadding="0" x ...
## [3] <table id="Table2" class="display sortable_datatable"
cellspacing="0 ...
## [4] <table id="Table3" class="regular"><caption><span
class="tableTitle" ...
## [5] <table id="Table4" class="regular" cellspacing="0"
cellpadding="0" x ...
## [6] <table id="Exhibit1" class="regular" cellspacing="0"
cellpadding="0" ...
```

To parse the HTML table data we use `html_table()`, rather than `html_text()`. For example, if we want the data from `id="Table3"` which is the fourth element in the list of tables. (i.e. `[4] <table id="Table3" class="regular"><caption><span class="tableTitle" ...`)

```
bls.gov.tbls.4 <- bls.gov.htm %>%
        html_nodes("table") %>%
        .[4] %>%
        html_table(fill = TRUE)
str(bls.gov.tbls.4)

## List of 1
##  $ :'data.frame':    16 obs. of  11 variables:
##   ..$ Supersector      : chr [1:16] "Mining and logging"
"Construction" "Manufacturing" "Trade, transportation, and utilities"
...
```

```
##   ..$ Apr             : int [1:16] -1 35 1 4 -4 5 3 0 2 1 ...
##   ..$ May             : int [1:16] 1 42 8 25 5 15 5 0 5 7 ...
##   ..$ Jun             : int [1:16] 1 23 4 8 -1 6 3 0 -2 0 ...
##   ..$ Jul             : int [1:16] 1 12 -1 8 -1 8 1 0 4 4 ...
##   ..$ Aug             : int [1:16] 0 12 4 16 3 8 5 0 5 5 ...
##   ..$ Sep             : int [1:16] 0 5 2 7 -2 5 4 0 -4 -4 ...
##   ..$ Oct             : int [1:16] 1 16 4 30 7 18 5 0 7 20 ...
##   ..$ Nov             : int [1:16] 0 -9 3 7 3 4 0 0 4 2 ...
##   ..$ Dec             : int [1:16] 0 -16 1 4 1 2 1 0 -1 9 ...
##   ..$ Cumulative  Total: chr [1:16] "3" "120" "26" "109" ...
```

```r
head(bls.gov.tbls.4)
```

```
## [[1]]
##                            Supersector Apr May Jun Jul Aug Sep Oct
Nov
## 1                   Mining and logging  -1   1   1   1   0   0   1
0
## 2                         Construction  35  42  23  12  12   5  16
-9
## 3                        Manufacturing   1   8   4  -1   4   2   4
3
## 4  Trade, transportation, and utilities   4  25   8   8  16   7  30
7
## 5                      Wholesale trade  -4   5  -1  -1   3  -2   7
3
## 6                         Retail trade   5  15   6   8   8   5  18
4
## 7          Transportation and warehousing   3   5   3   1   5   4   5
0
## 8                            Utilities   0   0   0   0   0   0   0
0
## 9                          Information   2   5  -2   4   5  -4   7
4
## 10                Financial activities   1   7   0   4   5  -4  20
2
## 11   Professional and business services  91  24  -2  40  22 -23  97
8
## 12        Education and health services  26  17 -19  20  20  -2  59
11
## 13              Leisure and hospitality  84  93  80  68  24 -36  -2
-21
## 14                      Other services  12   9   6  -2   5  -3   5
2
## 15                                      NA  NA  NA  NA  NA  NA  NA
NA
## 16          Monthly amount contributed 255 231  99 154 113 -58 237
7
##    Dec Cumulative  Total
## 1   0                  3
## 2  -16               120
## 3    1                26
## 4    4               109
## 5    1                11
## 6    2                71
## 7    1                27
## 8    0                 0
## 9   -1                20
## 10   9                44
## 11 -13               244
## 12  -7               125
## 13   8               298
## 14  -2                32
## 15  NA                  
## 16 -17             1,021
```

HTTP Request Methods: GET and POST

Two commonly used methods for a request-response between a client and server are: GET and POST. HTTP POST requests supply additional data from the client (browser) to the server in the message body. In contrast, GET requests include all required data in the URL.

- GET - Requests data from a specified resource
- POST - Submits data to be processed to a specified resource

The GET Method

Note that the query string (name/value pairs) is sent in the URL of a GET request:

```
/test/demo_form.asp?name1=value1&name2=value2
Some other notes on GET requests:
```

- GET requests can be cached
- GET requests remain in the browser history
- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data
- GET requests have length restrictions
- GET requests should be used only to retrieve data

The POST Method

Note that the query string (name/value pairs) is sent in the HTTP message body of a POST request:

```
POST /test/demo_form.asp HTTP/1.1
Host: w3schools.com
name1=value1&name2=value2
Some other notes on POST requests:
```

- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

Scraping indeed.com for jobs using GET

Scraping indeed.com for jobs would be the basis of a job bot.

```
url <- "http://www.indeed.com/jobs?q=data+scientist&start="
number <- seq(from = 10, to = 100, by = 10)
urls <- sapply(number, function(x) paste0(url, x))
urls

##  [1] "http://www.indeed.com/jobs?q=data+scientist&start=10"
##  [2] "http://www.indeed.com/jobs?q=data+scientist&start=20"
##  [3] "http://www.indeed.com/jobs?q=data+scientist&start=30"
##  [4] "http://www.indeed.com/jobs?q=data+scientist&start=40"
##  [5] "http://www.indeed.com/jobs?q=data+scientist&start=50"
##  [6] "http://www.indeed.com/jobs?q=data+scientist&start=60"
##  [7] "http://www.indeed.com/jobs?q=data+scientist&start=70"
##  [8] "http://www.indeed.com/jobs?q=data+scientist&start=80"
##  [9] "http://www.indeed.com/jobs?q=data+scientist&start=90"
## [10] "http://www.indeed.com/jobs?q=data+scientist&start=100"

len <- length(urls)
indeed.com.jobs <- data.frame(Title=character(),
                 Company=character(),
                 Location=character(),
                 Summary=character(),
                 stringsAsFactors=FALSE)
for(i in 1:len) {
  indeed.com.htm <- read_html(urls[len])
  title <- indeed.com.htm %>% html_nodes(".jobtitle .turnstileLink")
%>% html_text()
  company <- indeed.com.htm %>% html_nodes(".company span") %>%
html_text()
```

```
  location <- indeed.com.htm %>% html_nodes(".location > span") %>%
html_text()
  summary <- indeed.com.htm %>% html_nodes(".snip div .summary") %>%
html_text()
  scratch <- data.frame(Title = title, Company = company,
                      Location = location, Summary = summary)
  indeed.com.jobs <- rbind(indeed.com.jobs, scratch)
}

dim(indeed.com.jobs)

## [1] 100   4

head(indeed.com.jobs)

##                                              Title
## 1                       Data Scientist - INF0019551
## 2                                    Data Scientist
## 3                         Real World Data Scientist
## 4                       Data Scientist @ Apple Inc.
## 5                                    Data Scientist
## 6 Data Scientist - Information Management Solutions
##                                Company              Location
## 1  \n    \n        General Motors             Warren, MI
## 2       \n     \n         Rover.com         United States
## 3 \n    \n        GlaxoSmithKline         United States
## 4             \n    TekHiring.com         Sunnyvale, CA
## 5                   \n    Kabbage             Atlanta, GA
## 6 \n    \n        Bank of America Simi Valley, CA 93065
##
Summary
## 1                         \nO Analytic tools and languages for
statistical and machine learningmodels such as SAS, R, Python, Scala. O
Data management using SQL....
## 2  \nThe Data Scientist will be proficient with statistical tools
such as Saas and R, and well-versed in SQL. The Data Scientist will
also act as a domain expert...
## 3  \nJob Description Summary-GlaxoSmithKline’s (GSK) Real World Data
& Analytics team (part of the Real World Evidence and Epidemiology
department) supports drug...
## 4                                      \nMachine Intelligence for
the purposes of conducting machine learning and deep neural networks
using *TensorFlow, MLLIB*....
## 5 \nAs a part of the Data Science team, you will develop algorithms
that leverage multiple data sources to guide (amongst other things) our
underwriting process,...
## 6   \nExperience working with unstructured/big data using Text
Mining and Machine Learning techniques. Primary requirement is to
overcome issues of large complex...
```

```
indeed.com.jobs[,1:4] <- lapply(indeed.com.jobs[,1:4], str_trim)
head(indeed.com.jobs)
```

```
##                                                  Title         Company
## 1                         Data Scientist - INF0019551  General Motors
## 2                                      Data Scientist       Rover.com
## 3                           Real World Data Scientist GlaxoSmithKline
## 4                         Data Scientist @ Apple Inc.   TekHiring.com
## 5                                      Data Scientist         Kabbage
## 6 Data Scientist - Information Management Solutions Bank of America
##                Location
## 1           Warren, MI
## 2        United States
## 3        United States
## 4        Sunnyvale, CA
## 5          Atlanta, GA
## 6 Simi Valley, CA 93065
##
Summary
## 1                         O Analytic tools and languages for
statistical and machine learningmodels such as SAS, R, Python, Scala. O
Data management using SQL....
## 2  The Data Scientist will be proficient with statistical tools such
as Saas and R, and well-versed in SQL. The Data Scientist will also act
as a domain expert...
## 3  Job Description Summary-GlaxoSmithKline’s (GSK) Real World Data &
Analytics team (part of the Real World Evidence and Epidemiology
department) supports drug...
## 4                                      Machine Intelligence for the
purposes of conducting machine learning and deep neural networks using
*TensorFlow, MLLIB*....
## 5 As a part of the Data Science team, you will develop algorithms
that leverage multiple data sources to guide (amongst other things) our
underwriting process,...
## 6   Experience working with unstructured/big data using Text Mining
and Machine Learning techniques. Primary requirement is to overcome
issues of large complex...
```

```
head(indeed.com.jobs[,1:3])
```

```
##                                                  Title         Company
## 1                         Data Scientist - INF0019551  General Motors
## 2                                      Data Scientist       Rover.com
## 3                           Real World Data Scientist GlaxoSmithKline
## 4                         Data Scientist @ Apple Inc.   TekHiring.com
## 5                                      Data Scientist         Kabbage
## 6 Data Scientist - Information Management Solutions Bank of America
##                Location
## 1           Warren, MI
## 2        United States
## 3         United States
## 4         Sunnyvale, CA
## 5           Atlanta, GA
## 6 Simi Valley, CA 93065
```