Professor Bear :: Web Scraping

Professor Bear

September 1, 2018

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.

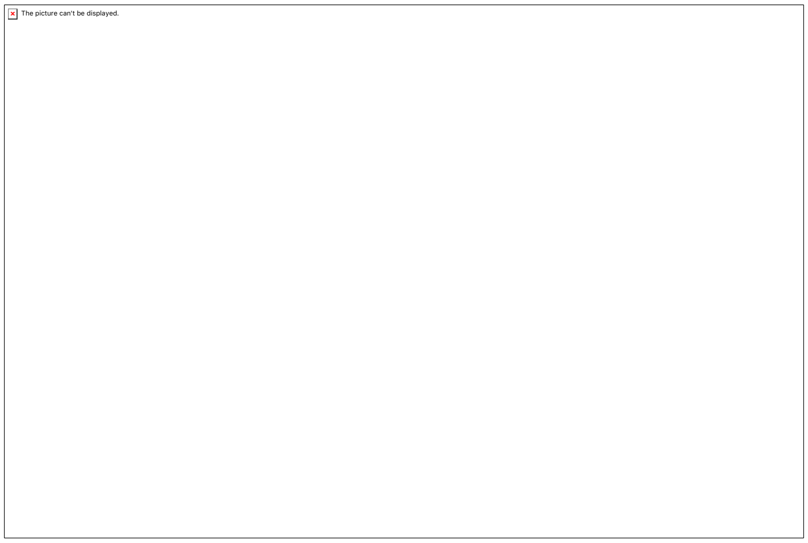
- rvest package
- SelectorGadget tool
- rvest and SelectorGadget guide
- Awesome tutorial for CSS Selectors
- rvest
- Import.io
- Google Chrome Webscraper
- ScaperHub pacman

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.



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>
```

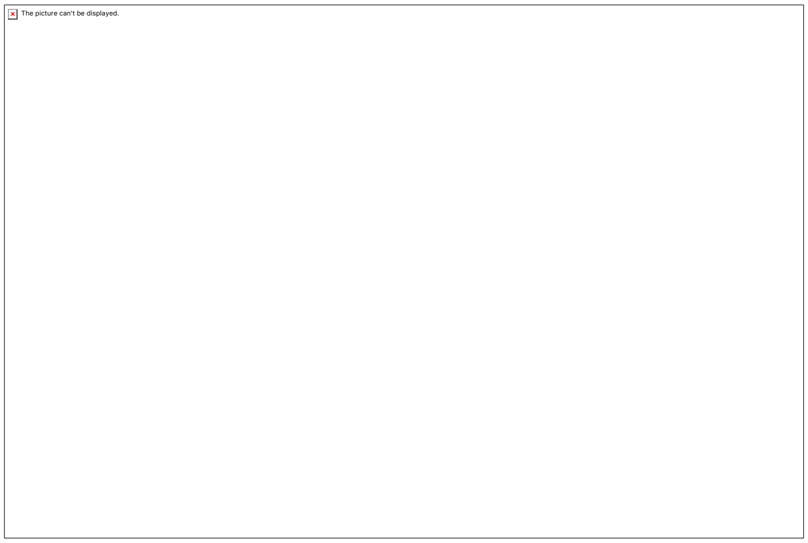
However there are often many paragraphs within an HTML page and sometimes we need more stucture to identify the paragraph with want. This is often provide by within tag attributes such as CSS styling. If our paragraph had a *.class selector* defining a paragraph as a *professor-name* we might leverage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



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 lang="en-us" class="a-no-js" data-19ax5a9jf="dingo">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=
...
## [2] <body class="a-m-us a-aui_149818-c a-aui_152852-c a-aui_157141-c a-a
...
```

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 (274)}
##  [1] <a id="nav-top"></a>
##  [2] <a href="/spark/ref=nav_upnav_merged_T1_CustomerReviews" class="nav
...
##  [3] <a href="/ref=nav_logo" class="nav-logo-link" tabindex="6">\n
```

```
...
##  [4] <a href="/prime?ref=nav_logo_prime_join" aria-label="" class="nav-s
...
##  [5] <a aria-label="Shop school supplies" href="/b/ref=nav_swm_BTS18_GW_
...
##  [6] <a class="nav-a nav-a-2 a-popover-trigger a-declarative" tabindex="
...
##  [7] <a href="/gp/customer-preferences/select-language/ref=topnav_lang?p
...
##  [8] <a href="https://www.amazon.com/ap/signin?openid.return_to=https%3A
...
##  [9] <a href="/gp/css/order-history?ref=nav_orders_first" class="nav-a n
...
## [10] <a href="/gp/prime?ref=nav_prime_try_btn" class="nav-a nav-a-2 nav-
...
## [11] <a href="https://www.amazon.com/gp/cart/view.html?ref=nav_cart" ari
...
## [12] <a href="/gp/site-directory?ref=nav_shopall_btn" class="nav-a nav-a
...
## [13] <a id="nav-your-amazon" href="/gp/yourstore/home?ref=nav_cs_ys" cla
...
## [14] <a href="/gp/goldbox?ref=nav_cs_gb" class="nav-a" tabindex="48">Tod
...
## [15] <a href="/gp/browse.html?node=2238192011&amp;ref=nav_cs_gift_cards"
...
## [16] <a href="/gp/browse.html?node=16115931011&amp;ref=nav_cs_registry"
...
## [17] <a href="/b/?node=12766669011&amp;ld=AZUSSOA-sell&amp;ref=nav_cs_se
...
## [18] <a href="/treasuretruck?ref=nav_cs_treasuretruck" class="nav-a" tab
...
## [19] <a href="/gp/help/customer/display.html?nodeId=508510&amp;ref=nav_c
...
## [20] <a href="/gp/help/customer/accessibility" aria-label="Click to call
...
## ...
```

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 (274)}
##  [1] <a id="nav-top"></a>
##  [2] <a href="/spark/ref=nav_upnav_merged_T1_CustomerReviews" class="nav
...
##  [3] <a href="/ref=nav_logo" class="nav-logo-link" tabindex="6">\n
...
##  [4] <a href="/prime?ref=nav_logo_prime_join" aria-label="" class="nav-s
...
##  [5] <a aria-label="Shop school supplies" href="/b/ref=nav_swm_BTS18_GW_
...
##  [6] <a class="nav-a nav-a-2 a-popover-trigger a-declarative" tabindex="
...
##  [7] <a href="/gp/customer-preferences/select-language/ref=topnav_lang?p
...
##  [8] <a href="https://www.amazon.com/ap/signin?openid.return_to=https%3A
...
##  [9] <a href="/gp/css/order-history?ref=nav_orders_first" class="nav-a n
...
## [10] <a href="/gp/prime?ref=nav_prime_try_btn" class="nav-a nav-a-2 nav-
...
## [11] <a href="https://www.amazon.com/gp/cart/view.html?ref=nav_cart" ari
...
## [12] <a href="/gp/site-directory?ref=nav_shopall_btn" class="nav-a nav-a
...
## [13] <a id="nav-your-amazon" href="/gp/yourstore/home?ref=nav_cs_ys" cla
...
## [14] <a href="/gp/goldbox?ref=nav_cs_gb" class="nav-a" tabindex="48">Tod
...
## [15] <a href="/gp/browse.html?node=2238192011&amp;ref=nav_cs_gift_cards"
...
## [16] <a href="/gp/browse.html?node=16115931011&amp;ref=nav_cs_registry"
...
## [17] <a href="/b/?node=12766669011&amp;ld=AZUSSOA-sell&amp;ref=nav_cs_se
...
## [18] <a href="/treasuretruck?ref=nav_cs_treasuretruck" class="nav-a" tab
...
## [19] <a href="/gp/help/customer/display.html?nodeId=508510&amp;ref=nav_c
...
```

```
## [20] <a href="/gp/help/customer/accessibility" aria-label="Click to call
...
## ...
```

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 (12)}
##  [1] <span class="a-size-small a-color-base reviews-sort-order-label a-t
...
##  [2] <span class="a-size-small a-color-base reviews-filter-by-label 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
...
```

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] "Sort by"
##  [2] "Filter by"
##  [3] "A High Quality Book On R Programming!"
##  [4] "best intro to R I've found"
##  [5] "Well written with great, detaled explanations"
##  [6] "A good reference book"
##  [7] "Useful reference"
##  [8] "Great for tasks where there should be an easy way, you just don't
know it"
##  [9] "Great R Reference Book"
## [10] "I have question, they have answer"
## [11] "but would like to know some pretty cool tricks and time saving ..."
## [12] "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] "\n      Format: Paperback"      "\n      Format: Paperback"
##  [3] "\n      Format: Paperback"      "\n      Format: Paperback"
##  [5] "\n      Format: Paperback"      "\n      Format: Kindle Edition"
##  [7] "\n      Format: Kindle Edition" "\n      Format: Paperback"
##  [9] "\n      Format: Paperback"      "\n      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] "\n      Paperback"      "\n      Paperback"
##  [3] "\n      Paperback"      "\n      Paperback"
##  [5] "\n      Paperback"      "\n      Kindle Edition"
##  [7] "\n      Kindle Edition" "\n      Paperback"
##  [9] "\n      Paperback"      "\n      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" "5.0 out of 5 stars"
##  [4] "4.0 out of 5 stars" "4.0 out of 5 stars" "5.0 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" "5.0" "4.0" "4.0" "5.0" "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 5 4 4 5 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()

## character(0)
```

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

## numeric(0)
```

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_concepts")
li_text <- scraping_wiki %>%
        html_nodes("li") %>%
        html_text()
length(li_text)
```

```
## [1] 1300

li_text[1:5]

## [1] "Classification"    "Clustering"        "Regression"
## [4] "Anomaly detection" "AutoML"
```

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"><tr>\n<td id="secondary-nav-td">\r\n\
...
## [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">\n<caption>\n<span class="tableTi
...
## [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 5 -5 7 3 0 2 0 ...
##   ..$ May              : int [1:16] 1 42 7 28 5 16 7 0 5 6 ...
##   ..$ Jun              : int [1:16] 1 23 4 12 -1 8 5 0 -1 1 ...
##   ..$ Jul              : int [1:16] 0 13 0 15 -1 13 3 0 4 6 ...
##   ..$ Aug              : int [1:16] 1 11 3 21 2 12 7 0 4 5 ...
##   ..$ Sep              : int [1:16] 0 6 2 12 -2 8 6 0 -3 -3 ...
##   ..$ Oct              : int [1:16] 1 17 5 40 7 23 10 0 8 20 ...
##   ..$ Nov              : int [1:16] 0 -8 3 13 2 5 6 0 4 2 ...
##   ..$ Dec              : int [1:16] 0 -16 1 7 2 2 3 0 -1 9 ...
##   ..$ Cumulative  Total: chr [1:16] "3" "123" "26" "153" ...
```

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

```
## [[1]]
##                            Supersector Apr May Jun Jul Aug Sep Oct Nov
## 1                    Mining and logging  -1   1   1   0   1   0   1   0
## 2                          Construction  35  42  23  13  11   6  17  -8
## 3                         Manufacturing   1   7   4   0   3   2   5   3
## 4  Trade, transportation, and utilities   5  28  12  15  21  12  40  13
## 5                       Wholesale trade  -5   5  -1  -1   2  -2   7   2
## 6                          Retail trade   7  16   8  13  12   8  23   5
## 7        Transportation and warehousing   3   7   5   3   7   6  10   6
## 8                             Utilities   0   0   0   0   0   0   0   0
## 9                           Information   2   5  -1   4   4  -3   8   4
## 10                 Financial activities   0   6   1   6   5  -3  20   2
## 11   Professional and business services  97  24  -1  48  22 -18  87   6
## 12        Education and health services  28  17 -18  34  20   1  56   8
## 13              Leisure and hospitality  86  93  82  80  24 -36   6 -20
## 14                       Other services  11   9   7   0   4  -2   7   0
## 15                                       NA  NA  NA  NA  NA  NA  NA  NA
## 16          Monthly amount contributed 264 232 110 200 115 -41 247   8
##    Dec Cumulative  Total
## 1    0                3
## 2  -16              123
## 3    1               26
## 4    7              153
## 5    2                9
## 6    2               94
## 7    3               50
## 8    0                0
## 9   -1               22
## 10   9               46
## 11 -22              243
## 12  -5              141
## 13   6              321
## 14  -2               34
## 15  NA
## 16 -23            1,112
```

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