Workshop 4: Fundamentals of Web Scraping

Myles D. Garvey, Ph.D

Winter, 2020

1 Learning Objectives

1. Learn the fundamentals of HTML, CSS, and Javascript.
2. Learn how data is represented in an HTML DOM Tree
3. Learn how to "read" HTML code in R and how to extract information from HTML.
4. Learn the difference between static and dynamic web scrapping.
5. Learn how to use the `rvest` package to conduct static web scrapping.
6. Learn how to use the `RSelenium` package to conduct dynamic web scrapping.

2 Workshop Description and Submission

While APIs are very useful technologies to be able to obtain data, the truth of the matter is that most good data for which we need is out and about on websites, unavailable via API. In some instances, we have such a great need for data that is not in an API, we need to resort to manual collection of information. For example, suppose we would like to gather a list of contacts from every business school in the country. If we have a list of websites to visit, we could go one by one, visiting the website, going to the faculty directory, and copy and pasting the faculty's information into an Excel file.

Needless to say, this process would take us a very long time. However, with a little bit of knowledge on how websites work in the first place, one could look at common patterns in these websites, and download the information automatically. A software that accomplishes such a task is called a *web scrapper*, and it is a very useful tool that can save us hours if not weeks of work. Unfortunately, given that the needs of data, and where the data is stored, and the format in which it is stored, varies very greatly, it is almost impossible to have a single software out there that can accomplish our goal. Hence, we usually must resort to *writing our own web scrapper*. This usually entails a simple process of first understanding the structure of the website from which we would like to scrape (if we are only doing so from a single website), and second, interacting with the website to download the data we need.

In this workshop, you will learn how websites "work". You do not need to become expert website designers. But you will need to know (1) how information on a website is represented, (2) how information on a website is instructed to be displayed and (3) how information can change on a website due to a user's interaction with the site. Once you have a fundamental understanding of this, only then can you move to web scraping. The subsequent tutorials will teach you how to extract web information in R, as well as how to scrape the web in a static and dynamic manner using R.

You will submit to blackboard a .PDF document. Complete each exercise in this workshop. These are easy and small, and so it should not take you very long. In a word document, type up the question number, the original question text, and your response in R code. Save your word document as a .PDF and submit it to the submission link on Blackboard. Please note that failure to submit this as a .PDF will result in a 0 on the workshop.

3 Tutorial 1: Understanding HTML, CSS, DOM, and Javascript

In order to understand how to scrape information from a website, one must first understand how a website is structured in the first place. Websites (with the exception of Flash Websites, which are rarely a thing these days) are constructed using three major technologies: HyperText Markup Language (HTML), Cascading Style Sheets (CSS), and Javascript. With these three technologies, a website is nothing more than (1) what elements should be displayed on the site, (2) how and where should they be displayed, and (3) how can the website change based on user interaction? When a creater of a website has specified the information to be shown, the way in which the information should be shown, and how the user can interact with the website to change the information, they do so by writing HTML/CSS/JS code. This code all feeds into a software called a browser. The browser dissects this information and displays all of it according to the creator's design. Therefore, for a basic website, the designer must:

- Specify the HTML code, which will tell the browser what specific information is to be displayed on the screen of the browser.
- Specify the CSS code, which will tell the browser what colors to use, sizes of boxes, pictures and media, as well as menus and general "look and feel" of the website. Put simply, the CSS is code that tells the browser *how* the information specified in the HTML code should be displayed on the screen.
- Specify the Javascript, which is code that is written to execute upon certain actions that the user of the browser may take. Javascript is similar (yet, very different) than R. This language is intended specifically to work within a browser to manipulate and change either the contend (HTML) or the look and feel (CSS) of the website upon user action.

Technically, these is a difference between a static and a dynamic website. A static website is one that does not store any information about a user's interaction with the elements on the website. When a user "visits" a static website, all they are doing is downloading the HTML, CSS, and Javascript files that will load within the user's web browser. A dynamic website, however, is much more complex. While a dynamic website must also have HTML, CSS, and Javascript, it also has *cookies* and *sessions*. A *session* is a collection of information about the user that the web server keeps track of. Instead of the browser just simply downloading a website's code, the user can manipulate things within the website which would also be reflected on a web server. Examples of this include the more recent cloud-based software, as well as any website that allows you to "login". Such websites are built in a very complex coding environment, and often use either Perl, Python, Java, C#, Ruby, PHP, and more recently (although not very popular) R. We do not need to dive into these details, and we will work primarily on the *client-side* of web sites.

To understand the structure of a website, we will explore an example that is attached to your module. The example is a very simple example, simple enough for us to dissect line by line:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<link rel="stylesheet" type="text/css" href="style.css" />

<title>Reverted by Bryant Smith</title>
</head>

<body>
    <div id="page">

        <div id="header">
          <h1>Reverted   </h1>
            <h2>The first XHTML 1.1 Validated Template by Bryant Smith.</h2>

      </div>
        <div id="bar">
            <ul>
                <li><a href="#">Home</a></li>
                <li><a href="#">Pictures</a></li>
                <li><a href="#">My Bio</a></li>
                <li><a href="#">My Company</a></li>
                <li><a href="#">Portfolio</a></li>
                <li><a href="#">Contact</a></li>
            </ul>
      </div>
        <div class="contentTitle"><h1>Template Usage</h1></div>
        <div class="contentText">
          <p>You may use this template on any site, anywhere, for free just<strong> please leave the link back to me in the footer</strong>. This template validates XHTML 1.1, CSS Validates as well; enjoy :) </p>
          <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Integer mi. Vivamus sit amet neque vitae sapien bibendum sodales. Curabitur elementum. Duis imperdiet. Donec eleifend porttitor sapien. Praesent leo. Quisque auctor velit sed tellus. Suspendisse potenti. Aenean laoreet imperdiet nunc. Donec commodo suscipit dolor. Aenean nibh. Sed id odio. Aliquam lobortis risus ut felis. Sed vehicula pellentesque quam.</p>
          <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Integer mi. Vivamus sit amet neque vitae sapien bibendum sodales. Curabitur elementum. Duis imperdiet. Donec eleifend porttitor sapien. Praesent leo. Quisque auctor velit sed tellus. Suspendisse potenti. Aenean laoreet imperdiet nunc. Donec commodo suscipit dolor. Aenean nibh. Sed id odio. Aliquam lobortis risus ut felis. Sed vehicula pellentesque quam.</p>
          <p><a href="index.html">    (read more)</a></p>
        </div>
        <div class="contentTitle">
          <h1>Gettysburg Address</h1>
        </div>
        <div class="contentText">
          <p>Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived in Liberty, and dedicated to the proposition that all men are created equal.</p>

<p>Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure. We are met on a great battle-field of that war. We have come to dedicate a portion of that field, as a final resting place for those who here gave their lives that that nation might live. It is altogether fitting and proper that we should do this.</p>

<p>But, in a larger sense, we can not dedicate -- we can not consecrate -- we can not hallow -- this ground. The brave men, living and dead, who struggled here, have consecrated it, far above our poor power to add or detract. The world will little note, nor long remember what we say here, but it can never forget what they did here. It is for us the living, rather, to be dedicated here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us -- that from these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion -- that we here highly resolve that these dead shall not have died in vain -- that this nation, under God, shall have a new birth of freedom -- and that government of the people, by the people, for the people, shall not perish from the earth.</p>
          <p><a href="index.html"> (read more)</a></p>
        </div>
        <div class="contentTitle"><h1>Yet Another?</h1></div>
        <div class="contentText">Each title is an H1 tag, so choose them carefully :)</div>
</div>
        <div id="footer"><a href="http://www.bryantsmith.com">web page designer </a> <a href="http://www.bryantsmith.com">bryant smith</a></div>
</body>
</html>
```

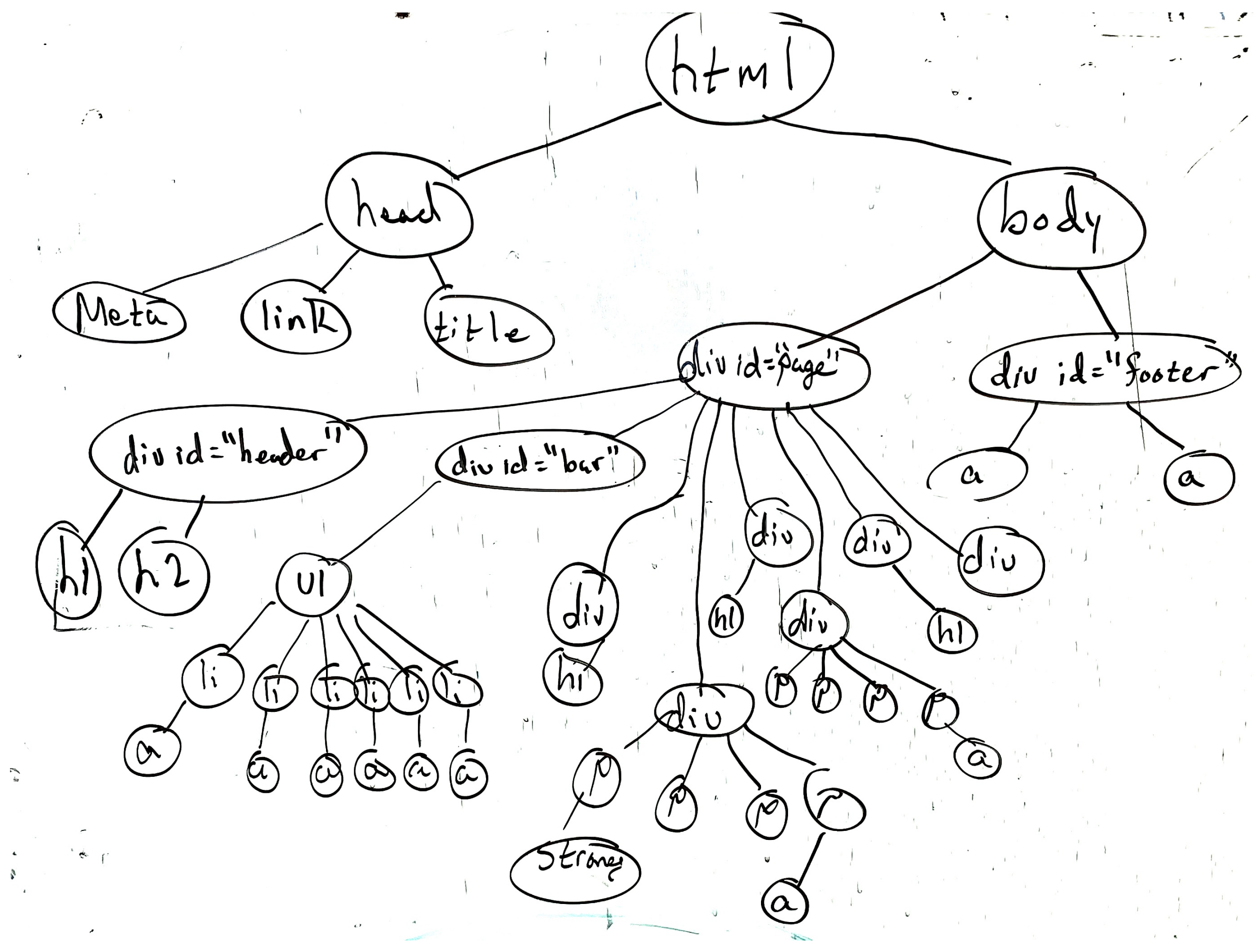
Listing 1: HTML Code for a basic static website.

Looking at the code, we notice that this looks very similar to XML. In fact, the coding structure works the same way. Information that we want to show on an HTML Web Page is determined by a tagging system, where information is placed in between tags. In the tags themselves, attributes can be specified with an attribute name = value pair. The tags that HTML can use are a fixed collection (the entire collection and meaning of them can be found here: https://www.w3schools.com/html/. Some of the the tags are themselves embedded in other tags (just as we had seen with general XML). Here are the first few tag meanings:

- `<html></html>` - all content on the website will be embedded in between these two tags. It has one attribute, namely the `xmlns` attribute. This attribute tells the web browser where it can find the version of html that the page is using.
- `<head></head>` This is called header information, and contains mainly meta information. Most of the information in this section does not appear directly on the website.

- `<meta/>` - specifies meta information about the website. Search engines often use these tags, and you can specify certain information to help optimize the search engines (such as Google, Bing, etc). Notice this is one tag instead of two. `<meta/>` is short for writing `<meta></meta>`, and if often used when there is nothing to put in between the tags.
- `<link />` - define a relationship between this HTML document and an outside document for which information should be inserted. Usually used to load in information about a .css file or a javascript file.
- `<title></title>` - the title of the website. This is the text that appears in the tab of a browser.
- `<body></body>` - the body of the HTML document. This is where you place the information you want to display on a page.
- `<div></div>` - a section in the HTML Page. This is the primary mechanism used to organize information on a website.
- `<h1></h1>` and `<h2></h2>` - header text which has different sizes. H1 is bigger text than H2.
- `<p>` - defines a paragraph in the HTMl Page. Usually, this tag is embedded within a `<div>` tag.
- `<a></a>` - a hyperlink. One defines where the link should bring the user by setting the `href=` attribute equal to the website in quotations.
- `<ul></ul>` - an unordered list. Each element in the list is defined by the `<li></li>` tag.

Further notice in the code that many of the `<div>` tags have an attribute of either `id=` or `class=`. This is a way to uniquely define the section on the page so that one can refer to that section later. The `class=` is called the class of the section, and it specifies a type of style that the section is to follow. General rule of thumb is that there can be multiple objects that reference the same class, but only one single object on the page that has an id. Put differently, each `id=` should be unique on the page. Furthermore, the notion of embedded tags means that the HTML document has a *tree structure* of tags. In what is called the *document object model (DOM)*, the DOM treats an HTML document as a tree of element tags. Each node in the tree will have properties, specified in the attribute of the tag. The node will also have "element text", which is any text in between the tags themselves. We can see from the below figure the tree structure for the HTML above (with my extremely poor handwriting):



But how does one define *how* the elements specified in the tree appear on the screen? This is where a second language comes into play, namely the cascading style sheet language. Cascading style sheets work very simplistically. We define a *style* by specifying certain values assigned to pre-defined attributes. We can define a style based on all tags of a certain type. We can also define styles for specific ids in the HTML page. Last, styles can be defined and grouped by what are called classes. When an element in an HTML page refers to a "class name", it is referred to the type of style that that content should display. Here is the example style sheet for our example website:

```
/* A Free Design by Bryant Smith (bryantsmith.com) */

html, body {
text-align: center;
}
p {text-align: left;}

body {
  margin: 0;
  padding: 0;
  background: #697281;
  font-family:  Arial, Helvetica, sans-serif;
  font-size: 13px;
  color: #F1F5F8;
  background: #26476E url(background.png) repeat-x;
}
*
{
  margin: 0 auto 0 auto;
 text-align:left;}

#page
{
  margin: 0 auto 0 auto;
  display: block;
  height:auto;
  position: relative;
  overflow: hidden;
  width: 753px;
}

#header
{
margin-top:25px;
padding-top:63px;
padding-left:30px;
padding-right:30px;
background-image:url(header.png);
background-repeat:no-repeat;
height:122px;
width:753px;
overflow:hidden;
}

#header h1, #header h2
{
margin-left:0px;
width:693px;
position:relative;
color:#000000;
text-align:center;
font-size:15px;
font-weight:bold;
font-family:Arial, Helvetica, sans-serif;
clear:both;
}

#header h1
{
margin-left:0px;
width:693px;
position:relative;
color:#26476E;
font-size:30px;
}



#bar
{
```

```
background-image:url(bar.png);
background-repeat:no-repeat;
height:53px;
width:750px;
padding-bottom:15px;
text-align:center;
margin:0 auto 0 auto;
}

#bar ul {
  margin: 0px auto 0px auto;
  padding: 0px;
  list-style-type: none;
  height: 74px;
  text-align:center;
}
#bar ul li , #bar ul li a, #bar ul li a:visited{
  margin: 0px;
  padding: 0px;
  height: 30px;
  overflow: hidden;
  display: inline;
  text-align:center;
  line-height:65px;
  color:#26476E;
  padding-left:6px;
  padding-right:6px;
}

#bar ul li a:hover
{
color:#000000;
text-decoration:underline;

}

.contentTitle
{
width:700px;
height:44px;
margin-top:20px;
margin-left:0px;
margin-right:25px;
background-image:url(tab_back.png);
background-repeat:no-repeat;

}

.contentTitle h1
{
margin-left:21px;
padding-top:8px;
font-size:21px;
font-weight:bold;
letter-spacing:-1px;
color:#FFFFFF;
```

```
}


.contentText
{
width:680px;
padding-left:30px;
padding-right:30px;
font-size:13px;
color:#FFFFFF;
line-height:30px;
}

a
{
text-decoration:none;
color:#F1F3F8;
}

a:hover
{
border-bottom:1px dotted #F1F3F8;
}


#footer {
width: 750px;
height:20px;
text-align:center;
font-size:9px;
color:#223F8D;
padding-top:20px;
}

html, body {
text-align: center;
}
p {
margin:7px;
text-align: left;
text-align:justify;
}
```

Listing 2: The CSS document for the example website.

The style sheet tells the browser which will read the HTML document how to display each component in the tree. For example, the `body` tag will have the following attributes:

```
body {
  margin: 0;
  padding: 0;
  background: #697281;
  font-family:  Arial, Helvetica, sans-serif;
  font-size: 13px;
  color: #F1F5F8;
  background: #26476E url(background.png) repeat-x;
}
```

Listing 3: How a browser should display the body tag.

Notice that each attribute and value is assigned as follows: `attribute name : attribute value`. Each pair is followed by a semi-colon, and all of the pairs are in a block of braces. The attributes will apply to all elements in the body, unless another style that applies to only a sub-element has been defined. A complete guide to learning CSS can be found at https://www.w3schools.com/css/. Generally, styles are defined for one of four types before the braces:

- * - means apply the style to all elements on the page.
- .classname - means to define a class with the name "classname"
- tag - means to define the style for all tags with the name "tag"
- #id - means to apply the style to any element with the id #id.

For example, the `<div></div>` block, with the id equal to "bar", in the HTML document will have the following style:

```
#bar
{
background-image:url(bar.png);
background-repeat:no-repeat;
height:53px;
width:750px;
padding-bottom:15px;
text-align:center;
margin:0 auto 0 auto;
}
```

Listing 4: The style of the div element which has the id name "bar".

Likewise, all tags the reference the class "contentText", like some of our `<div>` blocks, will have the following style:

```
.contentText
{
width:680px;
padding-left:30px;
padding-right:30px;
font-size:13px;
color:#FFFFFF;
line-height:30px;
}
```

Listing 5: The style for a class with a given name we defined as contentText.

At times, we want to change the style of a tag, but only if that tag is a child element in the DOM tree of another tag. For example, we may want to change the style of an unordered list (tag `<ul>`), but only those that are children elements of the element with id "bar". Hence, we would have:

```css
#bar ul {
  margin: 0px auto 0px auto;
  padding: 0px;
  list-style-type: none;
  height: 74px;
  text-align:center;
}
```

Listing 6: The style for only the ul element child of the div element with id bar.

Note that if we were to add another unordered list which is a child of a different element on the page, then this style would **not** apply to that ul, since it would not be a child of an element with the id name "bar". Suffice to say, all information on a website can hence be ascertained from its *page source*, or in other words, its HTML. In order to retrieve information from an HTML page, we can navigate the HTML tree (also called the DOM Tree). In addition, we have one additional component here, which is how to handle user interaction on a website. What happens if a user clicks, scrolls, types, etc on the web page? Ideally, we want to "listen" for these types of actions, call a function in response to it, and potentially change something in the DOM tree if the user did something on an element we expected themt to do something (that is, change the HTML code on the fly). This is where Javascript comes into play.

Javascript (which has absolutely no relation to the language Java) is a programming language that is intended to change information (HTML) and how it is displayed (CSS) in a browser by listening to a user's actions (clicking, scrolling, typing, etc). Essentially, we define a Javascript function, and code it to take an input (which is usually the action of a user) and change something in the DOM Tree (or the style). Each element in the tree is then assigned a "listener". If a listener is assigned to an element, then depending on what the user does to that element (hovers over it, scrolls over it, clicks on it, clicks off it, etc), a corresponding pre-defined function will be called.

Let us take an example. Suppose we wanted to change the text inside the `h1` tags inside the `div` tag with the id "header" to "HOVERED!", any time the user hovers the mouse pointer over the `div` box itself. We can add this functionality by creating two functions in Javascript added in the header of the HTML file:

```html
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<link rel="stylesheet" type="text/css" href="style.css" />
<script>
  function changeTextOn(){
    var the_header = document.getElementById("header");
    var the_h1_tag = the_header.childNodes[1];
    the_h1_tag.innerHTML = "HOVERED ";
  }
  function changeTextOff(){
    var the_header = document.getElementById("header");
    var the_h1_tag = the_header.childNodes[1];
    the_h1_tag.innerHTML = "Reverted ";
  }
</script>
<title>Reverted by Bryant Smith</title>
</head>
```

Listing 7: Changing the head tag content to include a script tag which is where the Javascript code will go to add the ability to change information on the page based on what the user does

on the website.

We first define the `changeTextOn()` function, which will first find the element in the tree with the id "header". Once it does that, it will look at all the child nodes. Lists (also called arrays in Javascript) are indexed starting at 0. The child element at index 1 is the `h1` tag. We then ask it to take the inner HTML in this tag and assign it to the word "HOVERED". The other function does the same, except with the word "Reverted". Notice how easy it is to manipulate the data on the HTML by using various functions and variables in javascript, thinking about the HTML page as a "tree" of tags. The functions will be called based on *events*. We can tell the browser to "listen" to the div box for the mouse to hover over it, as well as leave it, by using the `onmouseover` property, and assigning it to the function we just defined, as well as using the `onmouseout` property for when the user hovers off the `div` box, and assigning this to the other function name:

```
<div id="header" onmouseover="changeTextOn()" onmouseout="changeTextOff()">
          <h1>Reverted </h1>
            <h2>The first XHTML 1.1 Validated Template by Bryant Smith.</h2>

</div>
```

Listing 8: Adding event listeners to the element div.

Make these changes to the file `index.html` by opening it in a text editor (Notepad in Windows, Textedit in Mac), save the html file, and load it in your browser by double clicking on it. See what happens when you hover over the box! For more information on basic Javascript, visit https://www.w3schools.com/js/.

Exercises

7. Use the Javascript documentation referenced above to write a function in Javascript that will return all hyperlinks that are on the test webpage `index.html`.

4 Tutorial 2: Navigating an HTML Page via R

Now that we have a basic understanding of HTML, CSS, JS, and DOM, as well as some of the functions in JS that are used to find and manipulate elements in HTML, we can leverage the packages in R to do just this with our HTML Document. When we scrape the web, we are essentially parsing through an HTML Document to determine which information is relevant and which information is not. In order to scrape information, we typically need to understand the structure of the HTML website, depending on our scraping task. Hence, in this next portion of the tutorial, we will leverage R to navigate through an HTML document.

When we are performing web scraping on static websites, luckily for us, R has an easy to use package that only really necessitates us in having a mild understanding of the HTML DOM Tree for the site. The package we will use here is `rvest`. This package is a wrapper (that is, it calls other functions) around more complicated packages that allow us to traverse the HTML. Let us parse our own website as an example. Move the `index.html` file to your R working directory (or, set your R working directory to the directory in which `index.html` is located). Next, install the `rvest` package and load it. We will load the html document into R as such:

```
> library(rvest)
>
```

```
> the_html<-read_html("index.html")
> the_html
{xml_document}
<html xmlns="http://www.w3.org/1999/xhtml">
[1] <head>\n<meta http-equiv= ...
[2] <body>\r\n    <div id="pa ...
```

Listing 9: Loading in the HTML file in R for processing.

Now the object that is stored in the variable `the_html` has a lot of structure that we can leverage the extract any information we may need. For example, we can start by understanding its basic tree structure:

```
> html_structure(the_html,indent=5)
<html [xmlns]>
     <head>
          <meta [http-equiv, content]>
          <link [rel, type, href]>
          <script>
               {cdata}
          <title>
               {text}
     <body>
          {text}
          <div#page>
               {text}
               <div#header [onmouseover, onmouseout]>
                    {text}
                    <h1>
                         {text}
                    {text}
                    <h2>
                         {text}
                    {text}
               {text}
               <div#bar>
                    {text}
                    <ul>
                         <li>
                              <a [href]>
                                   {text}
                         {text}
                         <li>
                              <a [href]>
                                   {text}
                         {text}
                         <li>
                              <a [href]>
                                   {text}
                         {text}
                         <li>
                              <a [href]>
                                   {text}
                         {text}
                         <li>
                              <a [href]>
```

```
                            {text}
                    {text}
                    <li>
                        <a [href]>
                            {text}
                    {text}
            {text}
            <div.contentTitle>
                <h1>
                    {text}
            {text}
            <div.contentText>
                {text}
                <p>
                    {text}
                    <strong>
                        {text}
                    {text}
                {text}
                <p>
                    {text}
                {text}
                <p>
                    {text}
                {text}
                <p>
                    <a [href]>
                        {text}
                {text}
            {text}
            <div.contentTitle>
                {text}
                <h1>
                    {text}
                {text}
            {text}
            <div.contentText>
                {text}
                <p>
                    {text}
                {text}
                <p>
                    {text}
                {text}
                <p>
                    {text}
                {text}
                <p>
                    <a [href]>
                        {text}
                {text}
            {text}
            <div.contentTitle>
                <h1>
                    {text}
            {text}
```

```
              <div.contentText>
                  {text}
              {text}
          {text}
          <div#footer>
              <a [href]>
                  {text}
              {text}
              <a [href]>
                  {text}
          {text}
```

Listing 10: Getting the tree structure of an HTML document.

Now suppose that we would like to get all of the text that is stored in all of the paragraph tags. We can very easily do this by first finding all elements that are paragraph tags, and then subsequently by converting this list to text:

```
> p_tags<-html_nodes(the_html,"p")
> p_tags
{xml_nodeset (8)}
[1] <p>You may use this template on any site, anywhere, for fr ...
[2] <p>Lorem ipsum dolor sit amet, consectetuer adipiscing eli ...
[3] <p>Lorem ipsum dolor sit amet, consectetuer adipiscing eli ...
[4] <p><a href="index.html">  (read more)</a></p>
[5] <p>Four score and seven years ago our fathers brought fort ...
[6] <p>Now we are engaged in a great civil war, testing whethe ...
[7] <p>But, in a larger sense, we can not dedicate -- we can n ...
[8] <p><a href="index.html"> (read more)</a></p>
> p_text<-html_text(p_tags)
> p_text
[1] "You may use this template on any site, anywhere, for free just please leave
    the link back to me in the footer. This template validates XHTML 1.1, CSS
    Validates as well; enjoy :) "
[2] "Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Integer mi. Vivamus
     sit amet neque vitae sapien bibendum sodales. Curabitur elementum. Duis
    imperdiet. Donec eleifend porttitor sapien. Praesent leo. Quisque auctor velit
    sed tellus. Suspendisse potenti. Aenean laoreet imperdiet nunc. Donec commodo
    suscipit dolor. Aenean nibh. Sed id odio. Aliquam lobortis risus ut felis. Sed
    vehicula pellentesque quam."
[3] "Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Integer mi. Vivamus
     sit amet neque vitae sapien bibendum sodales. Curabitur elementum. Duis
    imperdiet. Donec eleifend porttitor sapien. Praesent leo. Quisque auctor velit
    sed tellus. Suspendisse potenti. Aenean laoreet imperdiet nunc. Donec commodo
    suscipit dolor. Aenean nibh. Sed id odio. Aliquam lobortis risus ut felis. Sed
    vehicula pellentesque quam."
[4] "  (read more)"
[5] "Four score and seven years ago our fathers brought forth on this continent, a
     new nation, conceived in Liberty, and dedicated to the proposition that all
    men are created equal."
[6] "Now we are engaged in a great civil war, testing whether that nation, or any
    nation so conceived and so dedicated, can long endure. We are met on a great
    battle-field of that war. We have come to dedicate a portion of that field, as
    a final resting place for those who here gave their lives that that nation
    might live. It is altogether fitting and proper that we should do this."
```

```
[7] "But, in a larger sense, we can not dedicate -- we can not consecrate -- we
    can not hallow -- this ground. The brave men, living and dead, who struggled
    here, have consecrated it, far above our poor power to add or detract. The
    world will little note, nor long remember what we say here, but it can never
    forget what they did here. It is for us the living, rather, to be dedicated
    here to the unfinished work which they who fought here have thus far so nobly
    advanced. It is rather for us to be here dedicated to the great task remaining
    before us -- that from these honored dead we take increased devotion to that
    cause for which they gave the last full measure of devotion -- that we here
    highly resolve that these dead shall not have died in vain -- that this nation,
     under God, shall have a new birth of freedom -- and that government of the
    people, by the people, for the people, shall not perish from the earth."
[8] " (read more)"
```

Listing 11: Extracting text from all the paragraph tags.

Now suppose we wanted to extract the URLs to all hyperlinks on the webpage, then we can do this by using:

```
> a_tags<-html_nodes(the_html,"a")
> a_tags
{xml_nodeset (10)}
 [1] <a href="#">Home</a>
 [2] <a href="#">Pictures</a>
 [3] <a href="#">My Bio</a>
 [4] <a href="#">My Company</a>
 [5] <a href="#">Portfolio</a>
 [6] <a href="#">Contact</a>
 [7] <a href="index.html">  (read more)</a>
 [8] <a href="index.html"> (read more)</a>
 [9] <a href="http://www.bryantsmith.com">web page designer </a>
[10] <a href="http://www.bryantsmith.com">bryant smith</a>
> html_attr(a_tags,"href")
 [1] "#"                          "#"
 [3] "#"                          "#"
 [5] "#"                          "#"
 [7] "index.html"                 "index.html"
 [9] "http://www.bryantsmith.com" "http://www.bryantsmith.com"
```

Listing 12: Extract the attribute "href" from all "a" tags.

Exercises

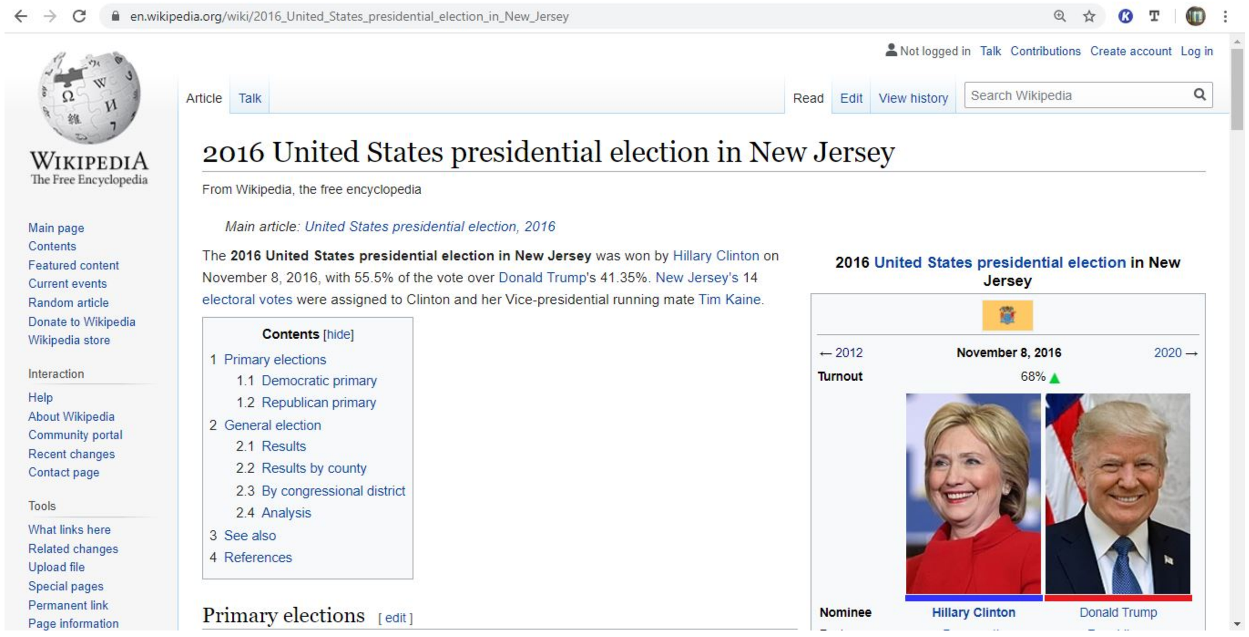
8. Loop up the documentation for the `rvest` package. Write code in R to extract only the text between <a> tags that are only in the div tag with id page. Ensure the code will not explore tags that are not under this tag.

5 Tutorial 3: Static-Based Web Scraping

The fundamental difference between static and dynamic web scraping lies in the design of the website itself. In some websites (such as Wix-Generated Websites, for example), the "website" is not actually the HTML that you download. Hence, it would be impossible to obtain data that is not in an HTML document yet. Data that is only generated due to calling a Javascript function, one that cannot be invoked without a browser due to the complexity. However, in other situations, the website (despite being a dynamic website) may be simple enough where you will not need to resort to more advanced tools.

Put differently, if you are able to download the HTML document, and your target data is in the HTML document, you can take a *static* approach to web scraping. This approach is straightforward: download the HTML page, parse it with a tool (such as `rvest`), and save the data to your intended location. Sounds easy, right? Well, there are a few things to consider. First, you must ensure that the data itself is indeed in the HTML. If it is not, then there is no point in parsing the HTML code, since your data wont be there! You would need to resort to a dynamic approach (see below). On the other hand, if your data is in the HTML page that you downloaded, then congrats! All you need to do is parse it. However, even parsing necessitates having a background knowledge on the structure of the HTML document.

Understanding the "structure" of a website boils down to trying to understand common patterns where your intended data most likely lie. This necessitates you to essentially reverse engineer a website's HTML, if at all possible. The good news is that most websites store their information in well-constructed and predictable ways. The job of the author of the web scraper is to first understand what that structure is and to subsequently write code to exploit that structure. Let us take an example. Suppose we wanted to download county level election data in New Jersey but only during presidential elections. We happen to find that Wikipedia has a nice collection of this information for the most recent 2016 election:



2016 United States presidential election in New Jersey

From Wikipedia, the free encyclopedia

Main article: United States presidential election, 2016

The **2016 United States presidential election in New Jersey** was won by Hillary Clinton on November 8, 2016, with 55.5% of the vote over Donald Trump's 41.35%. New Jersey's 14 electoral votes were assigned to Clinton and her Vice-presidential running mate Tim Kaine.

Contents [hide]
1 Primary elections
1.1 Democratic primary
1.2 Republican primary
2 General election
2.1 Results
2.2 Results by county
2.3 By congressional district
2.4 Analysis
3 See also
4 References

Primary elections [edit]

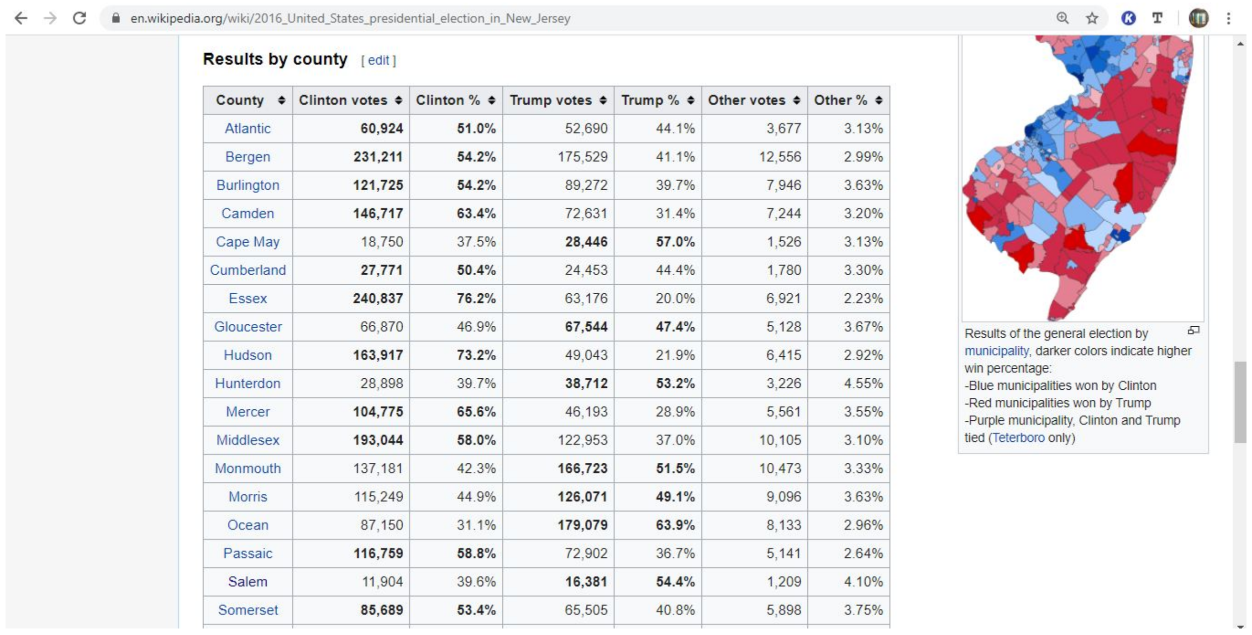
2016 United States presidential election in New Jersey

← 2012 November 8, 2016 2020 →

Turnout 68% ▲

Nominee Hillary Clinton Donald Trump

This is a very nice looking table, and it would be nice if all elections had this table as well:

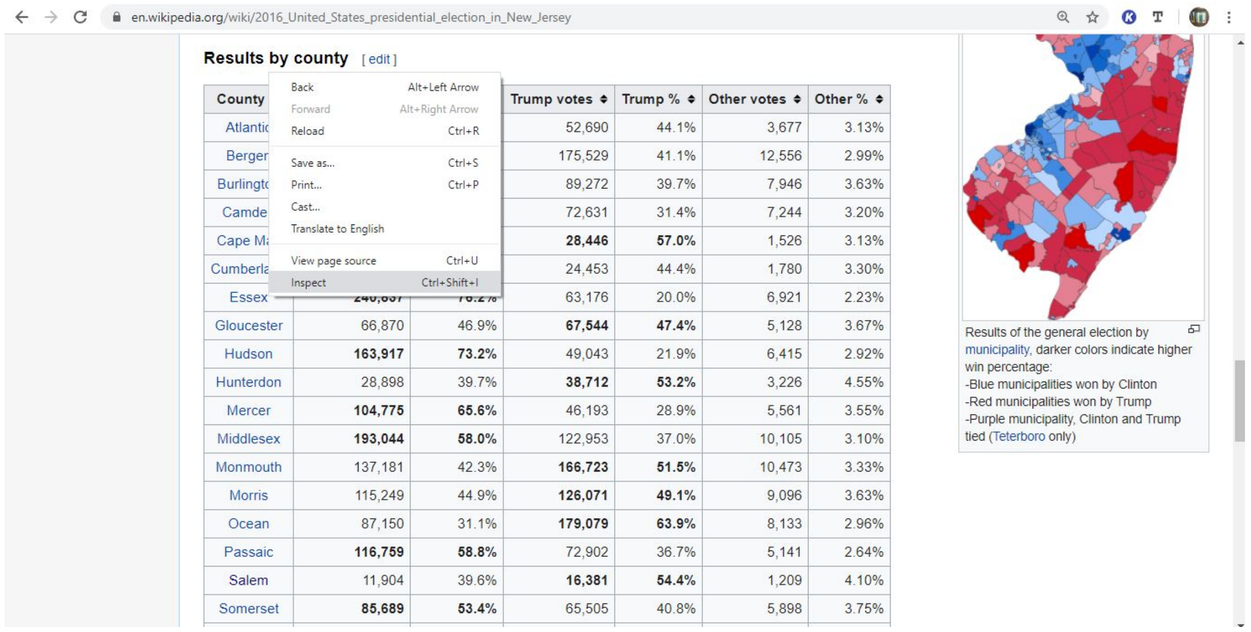

en.wikipedia.org/wiki/2016_United_States_presidential_election_in_New_Jersey

Results by county [edit]

County	Clinton votes	Clinton %	Trump votes	Trump %	Other votes	Other %
Atlantic	**60,924**	**51.0%**	52,690	44.1%	3,677	3.13%
Bergen	**231,211**	**54.2%**	175,529	41.1%	12,556	2.99%
Burlington	**121,725**	**54.2%**	89,272	39.7%	7,946	3.63%
Camden	**146,717**	**63.4%**	72,631	31.4%	7,244	3.20%
Cape May	18,750	37.5%	**28,446**	**57.0%**	1,526	3.13%
Cumberland	**27,771**	**50.4%**	24,453	44.4%	1,780	3.30%
Essex	**240,837**	**76.2%**	63,176	20.0%	6,921	2.23%
Gloucester	66,870	46.9%	**67,544**	**47.4%**	5,128	3.67%
Hudson	**163,917**	**73.2%**	49,043	21.9%	6,415	2.92%
Hunterdon	28,898	39.7%	**38,712**	**53.2%**	3,226	4.55%
Mercer	**104,775**	**65.6%**	46,193	28.9%	5,561	3.55%
Middlesex	**193,044**	**58.0%**	122,953	37.0%	10,105	3.10%
Monmouth	137,181	42.3%	**166,723**	**51.5%**	10,473	3.33%
Morris	115,249	44.9%	**126,071**	**49.1%**	9,096	3.63%
Ocean	87,150	31.1%	**179,079**	**63.9%**	8,133	2.96%
Passaic	**116,759**	**58.8%**	72,902	36.7%	5,141	2.64%
Salem	11,904	39.6%	**16,381**	**54.4%**	1,209	4.10%
Somerset	**85,689**	**53.4%**	65,505	40.8%	5,898	3.75%

Results of the general election by municipality, darker colors indicate higher win percentage:
-Blue municipalities won by Clinton
-Red municipalities won by Trump
-Purple municipality, Clinton and Trump tied (Teterboro only)

However, we don't want to go election by election by hand and copy and paste this table. It would be great if we can write a scrapper to download this information. We first want to understand its structure. In Google Chrome (which I recommend as your only browser for doing this), if you right click on the table (or anywhere), you will see on the bottom an option that says "inspect":

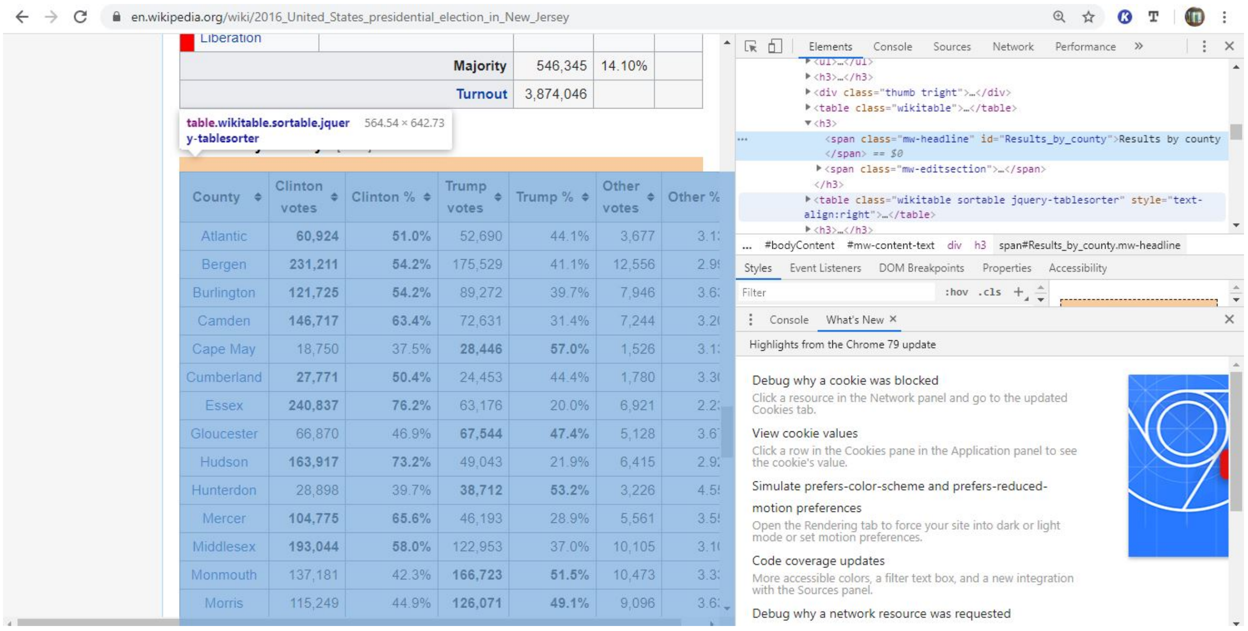


Results by county [edit]

County			Trump votes	Trump %	Other votes	Other %
Atlantic			52,690	44.1%	3,677	3.13%
Bergen			175,529	41.1%	12,556	2.99%
Burlington			89,272	39.7%	7,946	3.63%
Camden			72,631	31.4%	7,244	3.20%
Cape May			**28,446**	**57.0%**	1,526	3.13%
Cumberland			24,453	44.4%	1,780	3.30%
Essex			63,176	20.0%	6,921	2.23%
Gloucester	66,870	46.9%	**67,544**	**47.4%**	5,128	3.67%
Hudson	**163,917**	**73.2%**	49,043	21.9%	6,415	2.92%
Hunterdon	28,898	39.7%	**38,712**	**53.2%**	3,226	4.55%
Mercer	**104,775**	**65.6%**	46,193	28.9%	5,561	3.55%
Middlesex	**193,044**	**58.0%**	122,953	37.0%	10,105	3.10%
Monmouth	137,181	42.3%	**166,723**	**51.5%**	10,473	3.33%
Morris	115,249	44.9%	**126,071**	**49.1%**	9,096	3.63%
Ocean	87,150	31.1%	**179,079**	**63.9%**	8,133	2.96%
Passaic	**116,759**	**58.8%**	72,902	36.7%	5,141	2.64%
Salem	11,904	39.6%	**16,381**	**54.4%**	1,209	4.10%
Somerset	**85,689**	**53.4%**	65,505	40.8%	5,898	3.75%

Results of the general election by municipality, darker colors indicate higher win percentage:
-Blue municipalities won by Clinton
-Red municipalities won by Trump
-Purple municipality, Clinton and Trump tied (Teterboro only)

Click this, and you will have the HTML code (among other things) show up the right hand side of the screen. This tool is a very useful tool to understand how the HTML/CSS/JS is generating the content. When you hover over an element in the HTML code, it will highlight on your right hand side screen:



en.wikipedia.org/wiki/2016_United_States_presidential_election_in_New_Jersey

Liberation		
Majority	546,345	14.10%
Turnout	3,874,046	

table.wikitable.sortable.jquery-tablesorter 564.54 × 642.73

County	Clinton votes	Clinton %	Trump votes	Trump %	Other votes	Other %
Atlantic	**60,924**	**51.0%**	52,690	44.1%	3,677	3.1
Bergen	**231,211**	**54.2%**	175,529	41.1%	12,556	2.9
Burlington	**121,725**	**54.2%**	89,272	39.7%	7,946	3.6
Camden	**146,717**	**63.4%**	72,631	31.4%	7,244	3.2
Cape May	18,750	37.5%	**28,446**	**57.0%**	1,526	3.1
Cumberland	**27,771**	**50.4%**	24,453	44.4%	1,780	3.3
Essex	**240,837**	**76.2%**	63,176	20.0%	6,921	2.2
Gloucester	66,870	46.9%	**67,544**	**47.4%**	5,128	3.6
Hudson	**163,917**	**73.2%**	49,043	21.9%	6,415	2.9
Hunterdon	28,898	39.7%	**38,712**	**53.2%**	3,226	4.5
Mercer	**104,775**	**65.6%**	46,193	28.9%	5,561	3.5
Middlesex	**193,044**	**58.0%**	122,953	37.0%	10,105	3.1
Monmouth	137,181	42.3%	**166,723**	**51.5%**	10,473	3.3
Morris	115,249	44.9%	**126,071**	**49.1%**	9,096	3.6

```
▸<ul>…</ul>
▸<h3>…</h3>
▸<div class="thumb tright">…</div>
▸<table class="wikitable">…</table>
▾<h3>
    <span class="mw-headline" id="Results_by_county">Results by county
    </span> == $0
  ▸<span class="mw-editsection">…</span>
  </h3>
▸<table class="wikitable sortable jquery-tablesorter" style="text-
align:right">…</table>
▸<h3>…</h3>
```

... #bodyContent #mw-content-text div h3 span#Results_by_county.mw-headline

Highlights from the Chrome 79 update

Debug why a cookie was blocked
Click a resource in the Network panel and go to the updated Cookies tab.

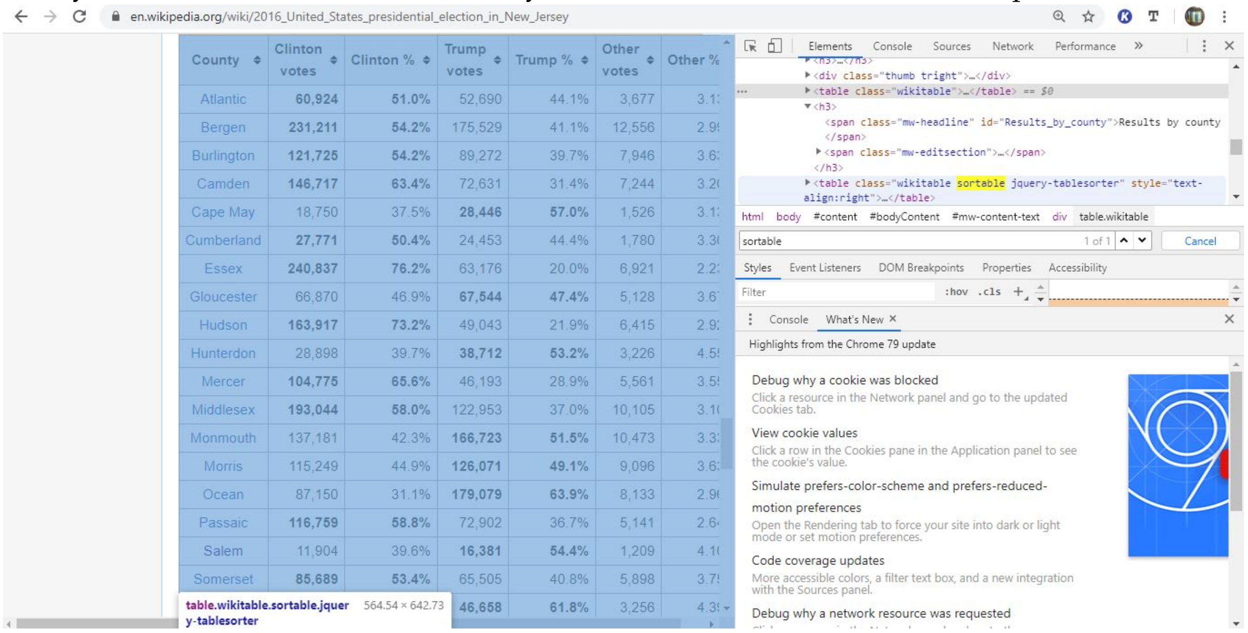
View cookie values
Click a row in the Cookies pane in the Application panel to see the cookie's value.

Simulate prefers-color-scheme and prefers-reduced-motion preferences
Open the Rendering tab to force your site into dark or light mode or set motion preferences.

Code coverage updates
More accessible colors, a filter text box, and a new integration with the Sources panel.

Debug why a network resource was requested

Ideally, we would like to see if there is any common pattern that can uniquely lead us to this specific table. We notice that the table does not have a unique id, but it does follow the class `wikitable sortable jquery-tablesorter`. Upon searching in the rest of the document for "wikitable", we notice a few more tables show up. However, a search for "sortable" shows us that this is the only table that uses this class. This may be beneficial for us, and we will keep this in mind:



en.wikipedia.org/wiki/2016_United_States_presidential_election_in_New_Jersey

County	Clinton votes	Clinton %	Trump votes	Trump %	Other votes	Other %
Atlantic	**60,924**	**51.0%**	52,690	44.1%	3,677	3.1
Bergen	**231,211**	**54.2%**	175,529	41.1%	12,556	2.9
Burlington	**121,725**	**54.2%**	89,272	39.7%	7,946	3.6
Camden	**146,717**	**63.4%**	72,631	31.4%	7,244	3.2
Cape May	18,750	37.5%	**28,446**	**57.0%**	1,526	3.1
Cumberland	**27,771**	**50.4%**	24,453	44.4%	1,780	3.3
Essex	**240,837**	**76.2%**	63,176	20.0%	6,921	2.2
Gloucester	66,870	46.9%	**67,544**	**47.4%**	5,128	3.6
Hudson	**163,917**	**73.2%**	49,043	21.9%	6,415	2.9
Hunterdon	28,898	39.7%	**38,712**	**53.2%**	3,226	4.5
Mercer	**104,775**	**65.6%**	46,193	28.9%	5,561	3.5
Middlesex	**193,044**	**58.0%**	122,953	37.0%	10,105	3.1
Monmouth	137,181	42.3%	**166,723**	**51.5%**	10,473	3.3
Morris	115,249	44.9%	**126,071**	**49.1%**	9,096	3.6
Ocean	87,150	31.1%	**179,079**	**63.9%**	8,133	2.9
Passaic	**116,759**	**58.8%**	72,902	36.7%	5,141	2.6
Salem	11,904	39.6%	**16,381**	**54.4%**	1,209	4.1
Somerset	**85,689**	**53.4%**	65,505	40.8%	5,898	3.7
			46,658	**61.8%**	3,256	4.3

table.wikitable.sortable.jquery-tablesorter 564.54 × 642.73

```
▸<h3>…</h3>
▸<div class="thumb tright">…</div>
▸<table class="wikitable">…</table> == $0
▾<h3>
    <span class="mw-headline" id="Results_by_county">Results by county
    </span>
  ▸<span class="mw-editsection">…</span>
  </h3>
▸<table class="wikitable sortable jquery-tablesorter" style="text-
align:right">…</table>
```

html body #content #bodyContent #mw-content-text div table.wikitable

sortable 1 of 1 Cancel

Highlights from the Chrome 79 update

Debug why a cookie was blocked
Click a resource in the Network panel and go to the updated Cookies tab.

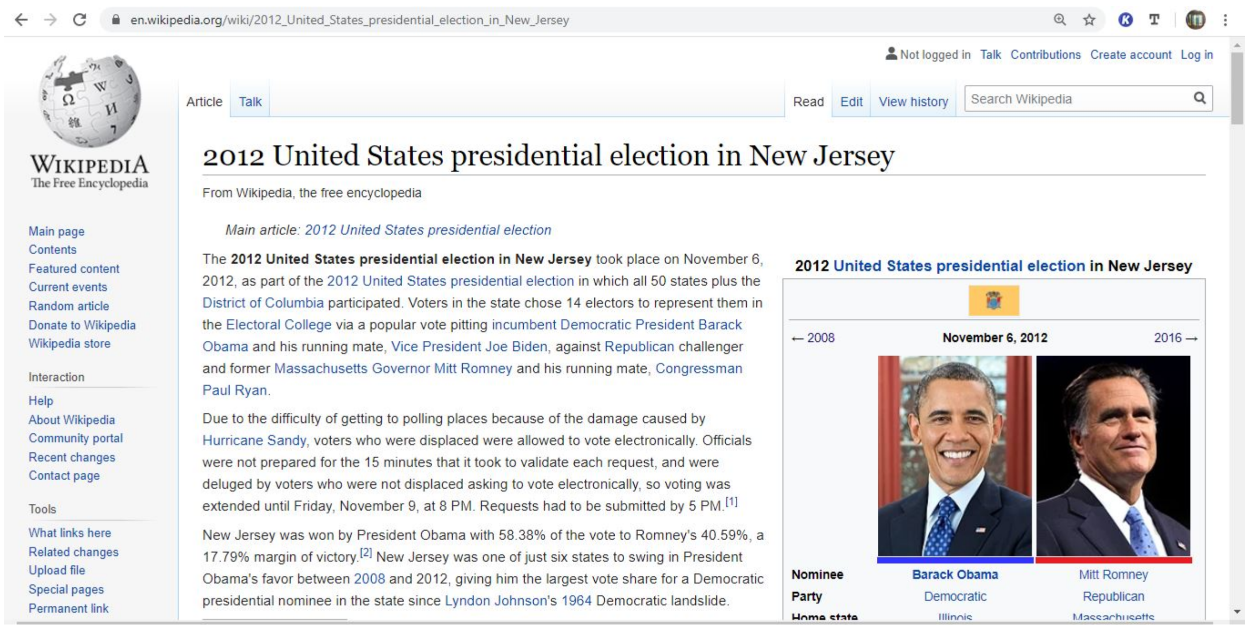
View cookie values
Click a row in the Cookies pane in the Application panel to see the cookie's value.

Simulate prefers-color-scheme and prefers-reduced-motion preferences
Open the Rendering tab to force your site into dark or light mode or set motion preferences.

Code coverage updates
More accessible colors, a filter text box, and a new integration with the Sources panel.

Debug why a network resource was requested

Now we would like to see if other pages follow a similar structure. If it does, then we are in great luck, and we can leverage that. After another Google Search for the election before this one, we notice that the url for the pages itself follows an interesting pattern. Namely that the url is exactly the same for each type of page, exact with the election year replaced. Further confirming, we can see this is indeed a way to reference the different elections:



en.wikipedia.org/wiki/2012_United_States_presidential_election_in_New_Jersey

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

2012 United States presidential election in New Jersey

From Wikipedia, the free encyclopedia

Main article: 2012 United States presidential election

The **2012 United States presidential election in New Jersey** took place on November 6, 2012, as part of the 2012 United States presidential election in which all 50 states plus the District of Columbia participated. Voters in the state chose 14 electors to represent them in the Electoral College via a popular vote pitting incumbent Democratic President Barack Obama and his running mate, Vice President Joe Biden, against Republican challenger and former Massachusetts Governor Mitt Romney and his running mate, Congressman Paul Ryan.

Due to the difficulty of getting to polling places because of the damage caused by Hurricane Sandy, voters who were displaced were allowed to vote electronically. Officials were not prepared for the 15 minutes that it took to validate each request, and were deluged by voters who were not displaced asking to vote electronically, so voting was extended until Friday, November 9, at 8 PM. Requests had to be submitted by 5 PM.[1]

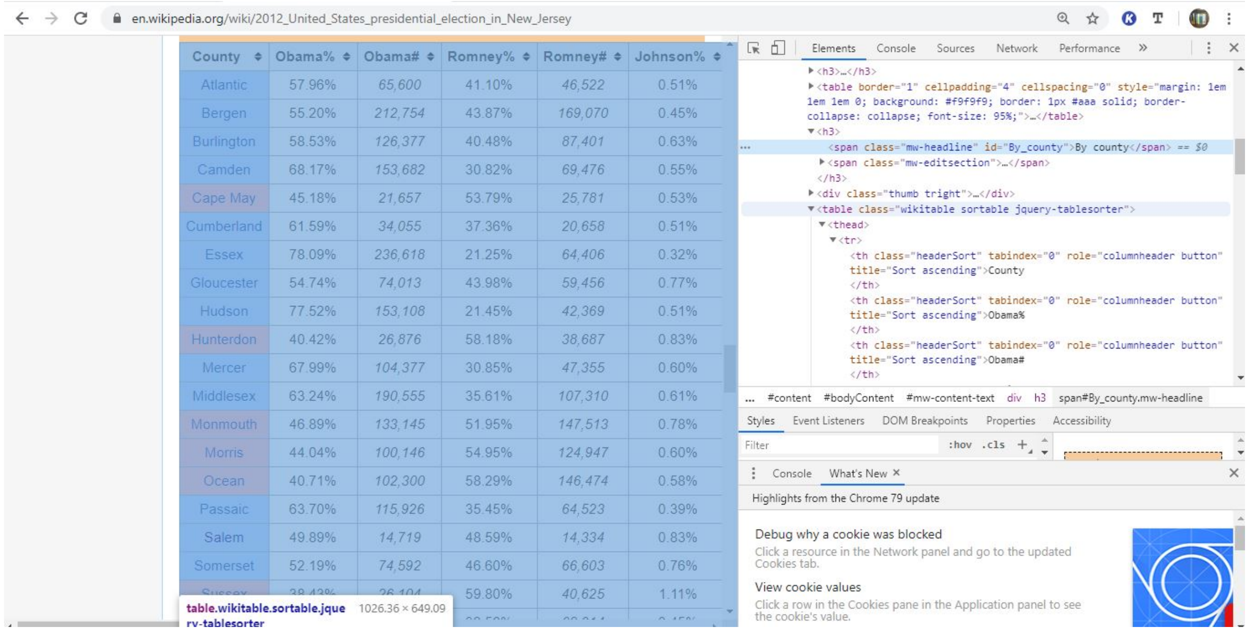
New Jersey was won by President Obama with 58.38% of the vote to Romney's 40.59%, a 17.79% margin of victory.[2] New Jersey was one of just six states to swing in President Obama's favor between 2008 and 2012, giving him the largest vote share for a Democratic presidential nominee in the state since Lyndon Johnson's 1964 Democratic landslide.

2012 United States presidential election in New Jersey

← 2008 November 6, 2012 2016 →

Nominee	Barack Obama	Mitt Romney
Party	Democratic	Republican

Looking at the structure again by clicking "inspect", we see that once again this table follows the same exact classname as the 2016 election table. Furthermore, a quick search of "sortable" again uniquely identifies this table. It looks like we hence have a way to determine where in the HTML tree this table lies. Do we need to confirm this for all pages? No, since if we write a quick script to see if there is only one unique table per election page, then we can take a gamble and run the scrapper, then later clean the data by hand:



en.wikipedia.org/wiki/2012_United_States_presidential_election_in_New_Jersey

County	Obama%	Obama#	Romney%	Romney#	Johnson%
Atlantic	57.96%	65,600	41.10%	46,522	0.51%
Bergen	55.20%	212,754	43.87%	169,070	0.45%
Burlington	58.53%	126,377	40.48%	87,401	0.63%
Camden	68.17%	153,682	30.82%	69,476	0.55%
Cape May	45.18%	21,657	53.79%	25,781	0.53%
Cumberland	61.59%	34,055	37.36%	20,658	0.51%
Essex	78.09%	236,618	21.25%	64,406	0.32%
Gloucester	54.74%	74,013	43.98%	59,456	0.77%
Hudson	77.52%	153,108	21.45%	42,369	0.51%
Hunterdon	40.42%	26,876	58.18%	38,687	0.83%
Mercer	67.99%	104,377	30.85%	47,355	0.60%
Middlesex	63.24%	190,555	35.61%	107,310	0.61%
Monmouth	46.89%	133,145	51.95%	147,513	0.78%
Morris	44.04%	100,146	54.95%	124,947	0.60%
Ocean	40.71%	102,300	58.29%	146,474	0.58%
Passaic	63.70%	115,926	35.45%	64,523	0.39%
Salem	49.89%	14,719	48.59%	14,334	0.83%
Somerset	52.19%	74,592	46.60%	66,603	0.76%
Sussex	38.43%	36,104	59.80%	40,625	1.11%

table.wikitable.sortable.jquery-tablesorter 1026.36 × 649.09

Elements Console Sources Network Performance »

```
▸<h3>…</h3>
▸<table border="1" cellpadding="4" cellspacing="0" style="margin: 1em 1em 0; background: #f9f9f9; border: 1px #aaa solid; border-collapse: collapse; font-size: 95%;">…</table>
▾<h3>
    <span class="mw-headline" id="By_county">By county</span> == $0
  ▸<span class="mw-editsection">…</span>
  </h3>
▸<div class="thumb tright">…</div>
▾<table class="wikitable sortable jquery-tablesorter">
  ▾<thead>
    ▾<tr>
        <th class="headerSort" tabindex="0" role="columnheader button" title="Sort ascending">County
        </th>
        <th class="headerSort" tabindex="0" role="columnheader button" title="Sort ascending">Obama%
        </th>
        <th class="headerSort" tabindex="0" role="columnheader button" title="Sort ascending">Obama#
        </th>
```

... #content #bodyContent #mw-content-text div h3 span#By_county.mw-headline

Styles Event Listeners DOM Breakpoints Properties Accessibility

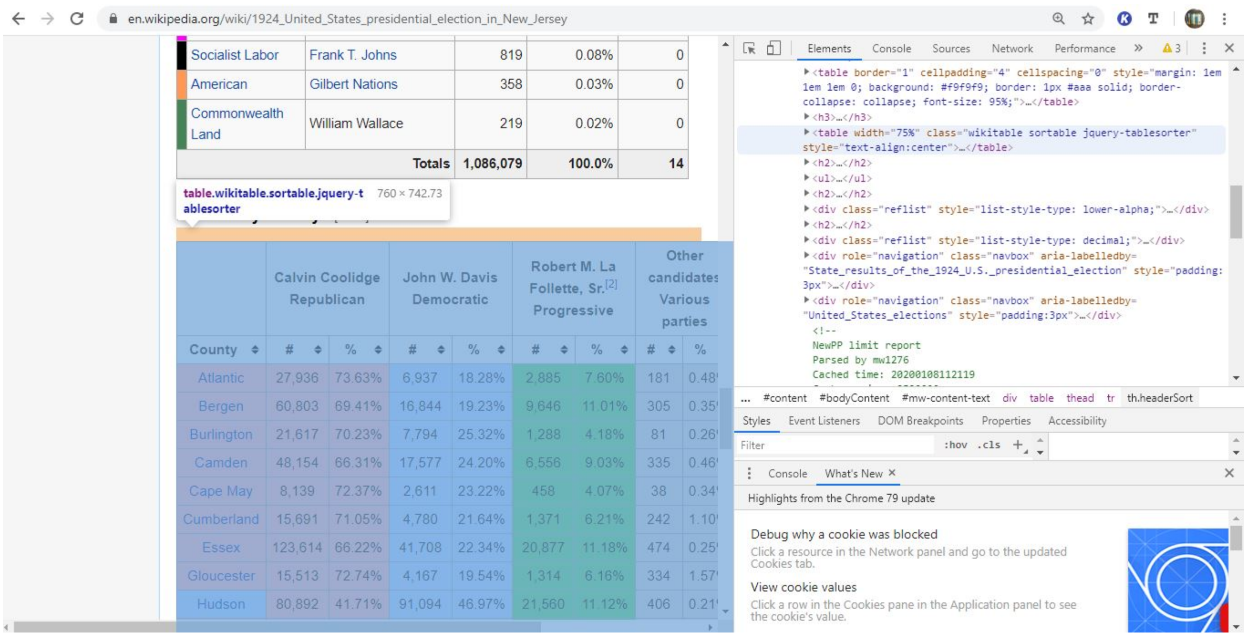
Console What's New ×

Highlights from the Chrome 79 update

Debug why a cookie was blocked
Click a resource in the Network panel and go to the updated Cookies tab.

View cookie values
Click a row in the Cookies pane in the Application panel to see the cookie's value.

We also by going back through the election years that the first year which shows county-level data is the 1924 election:



en.wikipedia.org/wiki/1924_United_States_presidential_election_in_New_Jersey

⬛	Socialist Labor	Frank T. Johns	819	0.08%	0
🟧	American	Gilbert Nations	358	0.03%	0
🟩	Commonwealth Land	William Wallace	219	0.02%	0
		Totals	1,086,079	100.0%	14

table.wikitable.sortable.jquery-tablesorter 760 × 742.73

	Calvin Coolidge Republican		John W. Davis Democratic		Robert M. La Follette, Sr.[2] Progressive		Other candidates Various parties	
County	#	%	#	%	#	%	#	%
Atlantic	27,936	73.63%	6,937	18.28%	2,885	7.60%	181	0.48
Bergen	60,803	69.41%	16,844	19.23%	9,646	11.01%	305	0.35
Burlington	21,617	70.23%	7,794	25.32%	1,288	4.18%	81	0.26
Camden	48,154	66.31%	17,577	24.20%	6,556	9.03%	335	0.46
Cape May	8,139	72.37%	2,611	23.22%	458	4.07%	38	0.34
Cumberland	15,691	71.05%	4,780	21.64%	1,371	6.21%	242	1.10
Essex	123,614	66.22%	41,708	22.34%	20,877	11.18%	474	0.25
Gloucester	15,513	72.74%	4,167	19.54%	1,314	6.16%	334	1.57
Hudson	80,892	41.71%	91,094	46.97%	21,560	11.12%	406	0.21

```
▸<table border="1" cellpadding="4" cellspacing="0" style="margin: 1em
1em 1em 0; background: #f9f9f9; border: 1px #aaa solid; border-
collapse: collapse; font-size: 95%;">…</table>
▸<h3>…</h3>
▸<table width="75%" class="wikitable sortable jquery-tablesorter"
style="text-align:center">…</table>
▸<h2>…</h2>
▸<ul>…</ul>
▸<h2>…</h2>
▸<div class="reflist" style="list-style-type: lower-alpha;">…</div>
▸<h2>…</h2>
▸<div class="reflist" style="list-style-type: decimal;">…</div>
▸<div role="navigation" class="navbox" aria-labelledby=
"State_results_of_the_1924_U.S._presidential_election" style="padding:
3px">…</div>
▸<div role="navigation" class="navbox" aria-labelledby=
"United_States_elections" style="padding:3px">…</div>
  <!--
  NewPP limit report
  Parsed by mw1276
  Cached time: 20200108112119
```

Highlights from the Chrome 79 update

Debug why a cookie was blocked

Click a resource in the Network panel and go to the updated Cookies tab.

View cookie values

Click a row in the Cookies pane in the Application panel to see the cookie's value.

Hence, we have a clear objective in mind for the data we would like to scrape: All County-Level Election Results for only Democrats and Republicans in every presidential election year starting from the 1924 election and ending at the 2016 election. Now it should be noted that upon further investigation, some tables have the class name `wikitable sortable jquery-tablesorter` while others just have `wikitable sortable`. Either way, given that we have already established that it is likely all tables uniquely have the class name "sortable" in it, we can hence refer to this table using a CSS Selectors (see https://www.w3schools.com/cssref/css$_selectors.asp$ for more on this). The selector is a way to uniquely "select" the element. We start with the tag name, followed by the class name, and all subsequent classes, with periods in between each. So if we want to select a table that is in the class wikitable and sortable, we would indicate this by "table.wikitable.sortable".

With all of this in mind, let us test how to extract this from the 2016 election page. First, we will insert the url into a variable and pass it into the read html function we discussed earlier. The package `rvest` will parse the HTML into a tree, which makes it easy for us to navigate. Once we do this we can now extract the table by id. Last, once we are brought to the table element in the HTML tree, we need to extract the data. Traditionally, this has been a pain. Luckily, `rvest` has a nice easy to use function to accomplish this, namely the `html_table` function. This will extract the data from the HTML table and store it in a data frame. For now, we will simply just store the results as a csv file for later processing:

```
> the_html<-read_html("https://en.wikipedia.org/wiki/2016_United_States_
   presidential_election_in_New_Jersey")
>
> county_html_table<-html_nodes(the_html,"table.wikitable.sortable")
>
> county_data<-html_table(county_html_table)
>
> county_data
[[1]]
         County Clinton votes Clinton % Trump votes Trump % Other votes Other %
```

```
1     Atlantic         60,924     51.0%      52,690    44.1%        3,677    3.13%
2       Bergen        231,211     54.2%     175,529    41.1%       12,556    2.99%
3   Burlington        121,725     54.2%      89,272    39.7%        7,946    3.63%
4       Camden        146,717     63.4%      72,631    31.4%        7,244    3.20%
5     Cape May         18,750     37.5%      28,446    57.0%        1,526    3.13%
6   Cumberland         27,771     50.4%      24,453    44.4%        1,780    3.30%
7        Essex        240,837     76.2%      63,176    20.0%        6,921    2.23%
8   Gloucester         66,870     46.9%      67,544    47.4%        5,128    3.67%
9       Hudson        163,917     73.2%      49,043    21.9%        6,415    2.92%
10   Hunterdon         28,898     39.7%      38,712    53.2%        3,226    4.55%
11      Mercer        104,775     65.6%      46,193    28.9%        5,561    3.55%
12   Middlesex        193,044     58.0%     122,953    37.0%       10,105    3.10%
13    Monmouth        137,181     42.3%     166,723    51.5%       10,473    3.33%
14      Morris        115,249     44.9%     126,071    49.1%        9,096    3.63%
15       Ocean         87,150     31.1%     179,079    63.9%        8,133    2.96%
16     Passaic        116,759     58.8%      72,902    36.7%        5,141    2.64%
17       Salem         11,904     39.6%      16,381    54.4%        1,209    4.10%
18    Somerset         85,689     53.4%      65,505    40.8%        5,898    3.75%
19      Sussex         24,212     32.0%      46,658    61.8%        3,256    4.39%
20       Union        147,414     66.4%      68,114    30.7%        6,447    2.90%
21      Warren         17,281     34.3%      29,858    59.2%        2,097    4.26%
```

Listing 13: Extracting the Election County Table Results.

We now can see that our code works for a single year, but what about other years? Let us try it out on all election years from 1924 to 2016. We can see that the url for the specific election year is exactly the same with the exception of the year. Hence, we can generate a vector of years, starting at 1924, ending at 2016, and going up by every 4 years. Then, we can write a loop to iterate for each year to insert the year in the URL, run the same code we just ran above but for that year, and save the result in a csv table (although, we will save it in a list for now). Let us see how this goes:

```
> election_years<-seq(1924,2016,4)
> county_data_year<-list()
> for(year in election_years){
+   the_url<-paste("https://en.wikipedia.org/wiki/",year,"_United_States_
   presidential_election_in_New_Jersey",sep="")
+
+   the_html<-read_html(the_url)
+
+   county_html_table<-html_nodes(the_html,"table.wikitable.sortable")
+
+   county_data<-html_table(county_html_table)
+
+   county_data_year[[paste(year,sep="")]] = county_data
+ }
Error: Table has inconsistent number of columns. Do you want fill = TRUE?
```

Listing 14: Extracting the results for multiple years but receiving an error in doing so.

Ah, so, we now encounter an error. What are we to do? Sometimes when we scrape, errors are okay, and we just want to ignore them, since they act as just annoying signals to us that we do not care about. Other times, errors may indicate that we wrote something wrong in our code, and we hence need to *debug* the code. In our case, we are not sure, so let us just debug to see what the deal is. One method for debugging in loops is to see how far the loop got before it broke down. In the code below, we can see that the code worked just fine up until the year 1964, which is where it broke down. Hence, we will assign the variable `year` to the value 1964 and see why it broke down by running each line inside the loop. Doing so leads us to the error when we try to convert the element to the data frame. That is, we can see the html is downloaded just fine, that the HTML Tree is generated just fine, that the table is found just fine, but the error lies in the conversion when we call the function `html_table`:

```
> names(county_data_year)
 [1] "1924" "1928" "1932" "1936" "1940" "1944" "1948" "1952" "1956" "1960"
> year <- 1964
> the_url<-paste("https://en.wikipedia.org/wiki/",year,"_United_States_
   presidential_election_in_New_Jersey",sep="")
>
>   the_html<-read_html(the_url)
>
>   county_html_table<-html_nodes(the_html,"table.wikitable.sortable")
> county_html_table
{xml_nodeset (1)}
[1] <table width="70%" class="wikitable sortab ...
> county_data<-html_table(county_html_table)
Error: Table has inconsistent number of columns. Do you want fill = TRUE?
> ?html_table
```

Listing 15: Debugging the problem and trying to find where the error started.

Upon looking at the documentation, we see that there is an input into this function indicated in the documentation: "fill If TRUE, automatically fill rows with fewer than the maximum number of columns with NAs." So our solution may be as simple as just setting this value equal to true in the input. We will do this. Furthermore, as a sanity check, let us see if there is anything is being stored in the list in each iteration. We do this by printing the element:

```
> election_years<-seq(1924,2016,4)
> county_data_year<-list()
> for(year in election_years){
+   the_url<-paste("https://en.wikipedia.org/wiki/",year,"_United_States_
   presidential_election_in_New_Jersey",sep="")
+
+   the_html<-read_html(the_url)
+
+   county_html_table<-html_nodes(the_html,"table.wikitable.sortable")
+   print(year)
+   print(county_html_table)
+   county_data<-html_table(county_html_table,fill=TRUE)
+
+   county_data_year[[paste(year,sep="")]] <- county_data
+ }
[1] 1924
{xml_nodeset (1)}
[1] <table width="75%" class="wikitable sortable" style="text ...
[1] 1928
{xml_nodeset (0)}
[1] 1932
```

```
{xml_nodeset (0)}
[1] 1936
{xml_nodeset (0)}
[1] 1940
{xml_nodeset (0)}
[1] 1944
{xml_nodeset (0)}
[1] 1948
{xml_nodeset (0)}
[1] 1952
{xml_nodeset (0)}
[1] 1956
{xml_nodeset (0)}
[1] 1960
{xml_nodeset (1)}
[1] <table width="60%" class="wikitable sortable" style="text ...
[1] 1964
{xml_nodeset (1)}
[1] <table width="70%" class="wikitable sortable" style="text ...
[1] 1968
{xml_nodeset (1)}
[1] <table class="wikitable sortable" style="text-align:cente ...
[1] 1972
{xml_nodeset (1)}
[1] <table width="60%" class="wikitable sortable" style="text ...
[1] 1976
{xml_nodeset (1)}
[1] <table width="60%" class="wikitable sortable" style="text ...
[1] 1980
{xml_nodeset (1)}
[1] <table width="65%" class="wikitable sortable" style="text ...
[1] 1984
{xml_nodeset (1)}
[1] <table width="65%" class="wikitable sortable" style="text ...
[1] 1988
{xml_nodeset (1)}
[1] <table width="65%" class="wikitable sortable" style="text ...
[1] 1992
{xml_nodeset (1)}
[1] <table width="65%" class="wikitable sortable" style="text ...
[1] 1996
{xml_nodeset (1)}
[1] <table width="65%" class="wikitable sortable" style="text ...
[1] 2000
{xml_nodeset (1)}
[1] <table class="wikitable sortable" style="text-align:right ...
[1] 2004
{xml_nodeset (1)}
[1] <table class="wikitable sortable" style="text-align:right ...
[1] 2008
{xml_nodeset (1)}
[1] <table width="60%" class="wikitable sortable"><tbody>\n<t ...
[1] 2012
{xml_nodeset (1)}
[1] <table class="wikitable sortable"><tbody>\n<tr>\n<th>Coun ...
[1] 2016
```

```
{xml_nodeset (1)}
[1] <table class="wikitable sortable" style="text-align:right ...
```

Listing 16: Further debugging and partial fixing of the problem.

As we can see from the output, it appears that the 1924 data point is an outlier. All years in between 1924 and 1960 appear to not have a table of county level results. As such, we will need to sacrifice our pursuit of this data and limit our download to only what is available. Namely, all county level data from 1960 to 2016 (still not a terrible data set!). Running the code again, we get a clean data set. Next is to confirm this. Unfortunately, each entry in the list is itself a list which contains a data frame (I know, this can get confusing!). As such, we need to extract each data frame and put it purely in the list. After, we inspect each data frame from each year:

```
> unlist(lapply(county_data_year,is.data.frame))
 1960  1964  1968  1972  1976  1980  1984  1988  1992  1996  2000  2004
FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 2008  2012  2016
FALSE FALSE FALSE
> y<-lapply(county_data_year,function(x){x[[1]]})
> unlist(lapply(y,is.data.frame))
1960 1964 1968 1972 1976 1980 1984 1988 1992 1996 2000 2004 2008 2012
TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
2016
TRUE
>
> lapply(y,function(x){print(x[1:3,1:4])})
          John Fitzgerald KennedyDemocratic
1   County                                #
2 Atlantic                           36,129
3   Bergen                          156,165
  John Fitzgerald KennedyDemocratic.1 Richard Milhous NixonRepublican
1                                   %                               #
2                              46.94%                          39,158
3                              40.90%                         224,969
    County Lyndon Baines JohnsonDemocratic
1   County                               #
2 Atlantic                          50,945
3   Bergen                         234,849
  Lyndon Baines JohnsonDemocratic.1 Barry Morris GoldwaterRepublican
1                                 %                                #
2                            65.30%                           25,626
3                            59.69%                          157,899
          Richard Milhous NixonRepublican
1   County                              #
2 Atlantic                         32,807
3   Bergen                        224,911
  Richard Milhous NixonRepublican.1
1                                 %
2                            42.15%
3                            54.45%
  Hubert Horatio Humphrey Jr.Democratic
1                                     #
2                                35,581
3                               162,182
          Richard Milhous NixonRepublican
```

```
1   County                             #
2 Atlantic                        45,667
3   Bergen                       285,458
  Richard Milhous NixonRepublican.1 George Stanley McGovernDemocratic
1                                 %                                  #
2                            59.54%                             28,203
3                            65.34%                            147,155
         Gerald FordRepublican Gerald FordRepublican.1
1   County                   #                       %
2 Atlantic              36,733                  45.56%
3   Bergen             237,331                  55.86%
  Jimmy CarterDemocratic
1                      #
2                 41,965
3                180,738
         Ronald Wilson ReaganRepublican
1   County                            #
2 Atlantic                       37,973
3   Bergen                      232,043
  Ronald Wilson ReaganRepublican.1 James Earl CarterDemocratic
1                                %                           #
2                           49.83%                      31,286
3                           55.89%                     139,474
         Ronald Wilson ReaganRepublican
1   County                            #
2 Atlantic                       49,158
3   Bergen                      268,507
  Ronald Wilson ReaganRepublican.1 Walter Fritz MondaleDemocratic
1                                %                              #
2                           59.33%                         33,240
3                           63.22%                        155,039
         George Herbert Walker BushRepublican
1   County                                  #
2 Atlantic                             44,748
3   Bergen                            226,885
  George Herbert Walker BushRepublican.1
1                                      %
2                                 56.33%
3                                 58.19%
  Michael Stanley DukakisDemocratic
1                                 #
2                            34,047
3                           160,655
         William Jefferson ClintonDemocratic
1   County                                 #
2 Atlantic                            39,633
3   Bergen                           171,104
  William Jefferson ClintonDemocratic.1
1                                     %
2                                43.89%
3                                42.44%
  George Herbert Walker BushRepublican
1                                    #
2                               34,279
3                              178,223
         William Jefferson ClintonDemocratic
```

```
1   County                                    #
2 Atlantic                               44,434
3   Bergen                              191,085
  William Jefferson ClintonDemocratic.1 Robert Joseph DoleRepublican
1                                     %                             #
2                                53.15%                        29,538
3                                52.66%                       141,164
      County Gore votes Gore % Bush votes
1   Atlantic     52,880  58.0%     35,593
2     Bergen    202,682  55.3%    152,731
3 Burlington     99,506  56.1%     72,254
      County Kerry % Kerry # Bush %
1   Atlantic   52.5%  55,746  46.6%
2     Bergen   51.9% 207,666  47.4%
3 Burlington   53.1% 110,411  46.1%
        X1       X2       X3        X4
1   County Obama % Obama # McCain %
2 Atlantic   56.9%  67,830     41.8%
3   Bergen   54.2% 225,367     44.7%
      County Obama%  Obama# Romney%
1   Atlantic 57.96%  65,600  41.10%
2     Bergen 55.20% 212,754  43.87%
3 Burlington 58.53% 126,377  40.48%
      County Clinton votes Clinton % Trump votes
1   Atlantic        60,924     51.0%      52,690
2     Bergen       231,211     54.2%     175,529
3 Burlington       121,725     54.2%      89,272
$`1960`
           John Fitzgerald KennedyDemocratic
1   County                                #
2 Atlantic                           36,129
3   Bergen                          156,165
  John Fitzgerald KennedyDemocratic.1 Richard Milhous NixonRepublican
1                                   %                               #
2                              46.94%                          39,158
3                              40.90%                         224,969

$`1964`
    County Lyndon Baines JohnsonDemocratic
1   County                              #
2 Atlantic                         50,945
3   Bergen                        234,849
  Lyndon Baines JohnsonDemocratic.1 Barry Morris GoldwaterRepublican
1                                 %                                #
2                            65.30%                           25,626
3                            59.69%                          157,899

$`1968`
           Richard Milhous NixonRepublican
1   County                              #
2 Atlantic                         32,807
3   Bergen                        224,911
  Richard Milhous NixonRepublican.1
1                                 %
2                            42.15%
3                            54.45%
```

```
  Hubert Horatio Humphrey Jr.Democratic
1                                     #
2                                35,581
3                               162,182

$`1972`
           Richard Milhous NixonRepublican
1   County                               #
2 Atlantic                          45,667
3   Bergen                         285,458
  Richard Milhous NixonRepublican.1 George Stanley McGovernDemocratic
1                                 %                                 #
2                            59.54%                            28,203
3                            65.34%                           147,155

$`1976`
           Gerald FordRepublican Gerald FordRepublican.1
1   County                     #                       %
2 Atlantic                36,733                  45.56%
3   Bergen               237,331                  55.86%
  Jimmy CarterDemocratic
1                      #
2                 41,965
3                180,738

$`1980`
           Ronald Wilson ReaganRepublican
1   County                              #
2 Atlantic                         37,973
3   Bergen                        232,043
  Ronald Wilson ReaganRepublican.1 James Earl CarterDemocratic
1                                %                           #
2                           49.83%                      31,286
3                           55.89%                     139,474

$`1984`
           Ronald Wilson ReaganRepublican
1   County                              #
2 Atlantic                         49,158
3   Bergen                        268,507
  Ronald Wilson ReaganRepublican.1 Walter Fritz MondaleDemocratic
1                                %                              #
2                           59.33%                         33,240
3                           63.22%                        155,039

$`1988`
           George Herbert Walker BushRepublican
1   County                                    #
2 Atlantic                               44,748
3   Bergen                              226,885
  George Herbert Walker BushRepublican.1
1                                      %
2                                 56.33%
3                                 58.19%
  Michael Stanley DukakisDemocratic
1                                 #
```

```
2                              34,047
3                             160,655

$‘1992‘
           William Jefferson ClintonDemocratic
1   County                                    #
2 Atlantic                               39,633
3   Bergen                              171,104
  William Jefferson ClintonDemocratic.1
1                                     %
2                                43.89%
3                                42.44%
  George Herbert Walker BushRepublican
1                                    #
2                               34,279
3                              178,223

$‘1996‘
           William Jefferson ClintonDemocratic
1   County                                    #
2 Atlantic                               44,434
3   Bergen                              191,085
  William Jefferson ClintonDemocratic.1 Robert Joseph DoleRepublican
1                                     %                             #
2                                53.15%                        29,538
3                                52.66%                       141,164

$‘2000‘
      County Gore votes Gore % Bush votes
1   Atlantic     52,880  58.0%     35,593
2     Bergen    202,682  55.3%    152,731
3 Burlington     99,506  56.1%     72,254

$‘2004‘
      County Kerry % Kerry # Bush %
1   Atlantic   52.5%  55,746  46.6%
2     Bergen   51.9% 207,666  47.4%
3 Burlington   53.1% 110,411  46.1%

$‘2008‘
        X1      X2      X3       X4
1   County Obama % Obama # McCain %
2 Atlantic   56.9%  67,830    41.8%
3   Bergen   54.2% 225,367    44.7%

$‘2012‘
      County Obama%  Obama# Romney%
1   Atlantic 57.96%  65,600  41.10%
2     Bergen 55.20% 212,754  43.87%
3 Burlington 58.53% 126,377  40.48%

$‘2016‘
      County Clinton votes Clinton % Trump votes
1   Atlantic        60,924     51.0%      52,690
2     Bergen       231,211     54.2%     175,529
```

```
3 Burlington        121,725      54.2%       89,272
```

Listing 17: Correcting the data format and inspecting the structure of each table.

Unfortunately, we can see that each table for each year follows a slightly different format. At this point, manual manipulation is most likely needed. However, it is evident that we at least have downloaded the important data, and the rest is merely small data cleaning and organizing tasks.

Exercises

9. In many of the presidential elections, there is another table under the "Results" section on the wikipedia page. Write R code to extract, download, and save this table as a csv file with the year as the name of the file.

6 Tutorial 4: Dynamic-Based Web Scraping

As we illustrated in the previous section, static web scraping is made very easy when using the `rvest` package. This is due to the fact that after years of various packages and the like, parsing HTML/CSS/JS has become well established. However, there are considerations to think about when using this method. First, the most important aspect of all of this is that when you scrape a site in a static manner, you are not interacting with that site. You are merely downloading an HTML/CSS/JS file, and working on that file directly. If a change on the website has been made, this will not be reflected in the HTML file. You can test this with the HTML code from Tutorial 2. Despite the fact that our JS code changes the HTML code in the browser, it will not do so in the file. Second, in static web scraping, we are not downloading the HTML file from a browser, but rather through an application (in our instance, through R). This means that you are in no way interacting with the website.

However, the website for which you seek to interact may be so complicated that the HTML code is changed on the fly. Meaning, the changes in the HTML code will only take place in a web browser rather than in the HTML file itself. This poses some issues when scraping. Other issues to think about when we scrape via static methods:

- Your ISP may ban you from a website. Too many calls to a website can result in your ip address being placed on a "blocked" list.
- The web server itself may block your IP address.
- The web site may dynamically update the underlying HTML code, which can only be accessed in a web browser.
- The web site may have what is called session management, which keeps track of your movements from page to page (such as "logging into" a website). You may not be able to access such information unless you are "logged in".

In these instances, our methods discussed in the previous tutorial will not hold. We will either be banned from a website, or, will be unable to gain access programatically to the data which we need. So what can we do? The answer lies in what is called *dynamic web scraping*. Earlier, we used R to essentially send a GET request to a website, which in turn downloaded HTML documents for us to process, all without a browser. In dynamic web scraping, we insert a "middle man" into this process. Instead of analyzing HTML code from a file, we instead use a web browser to send requests to websites. The browser will in turn receive HTML code. Instead of us analyzing the code directly, we instead programatically control the browser's movements through a series of scrolls, clicks, mouse movements, typing,etc. By doing so, the HTML in the browser will change. However, since we have programmatic access to the browser, we will be able to obtain the information that is needed. This method allows us to overcome many of the problems listed earlier. The reason is simple: to the ISP and the website, they cannot tell the difference between you or a computer using the web browser.

Of course, this all comes at a cost (after all, there is no such thing as a free lunch!). We sacrifice *speed* in favor of quantity and quality of data. Given that we are doing this in a web browser, we are adding one more layer of code in between us and the web server. This means that our requests/responses will slow. Despite this, we can still get really good data that we otherwise are unable to get via static web scrapping! Let us look at a small example. The way in which we connect to a browser in R is through the Selenium software. You will need the following to run it:

- The latest version of Java installed on your computer.
- The `RSelenium` R pacakge installed.

Setup is not difficult, but is a bit tedious. First, go to your Google Chrome browser and look at its version number (go to Settings>About Chrome). Insert that version number in the "chromever" option in the rsDriver function. R will most likely give you an error:

```
> library(RSelenium)
> rd<-rsDriver(browser=c("chrome"),chromever="79.0.3945.117")
checking Selenium Server versions:
BEGIN: PREDOWNLOAD
BEGIN: DOWNLOAD
BEGIN: POSTDOWNLOAD
checking chromedriver versions:
BEGIN: PREDOWNLOAD
BEGIN: DOWNLOAD
BEGIN: POSTDOWNLOAD
Error in chrome_ver(chromecheck[["platform"]], chromever) :
  version requested doesnt match versions available =
   75.0.3770.90,76.0.3809.12,76.0.3809.126,76.0.3809.25,77.0.3865.10,77.0.3865.40,78.0.3904.105
```

Listing 18: Getting RSelenium up and running but getting an error message with the wrong version number.

Find the version number that is less than you chrome version, but is the highest one that is less than your chrome. In my case, this would be "78.0.3904.11", since this is the highest version less than "79.0.3945.117". Insert this as the chromever option, and try again:

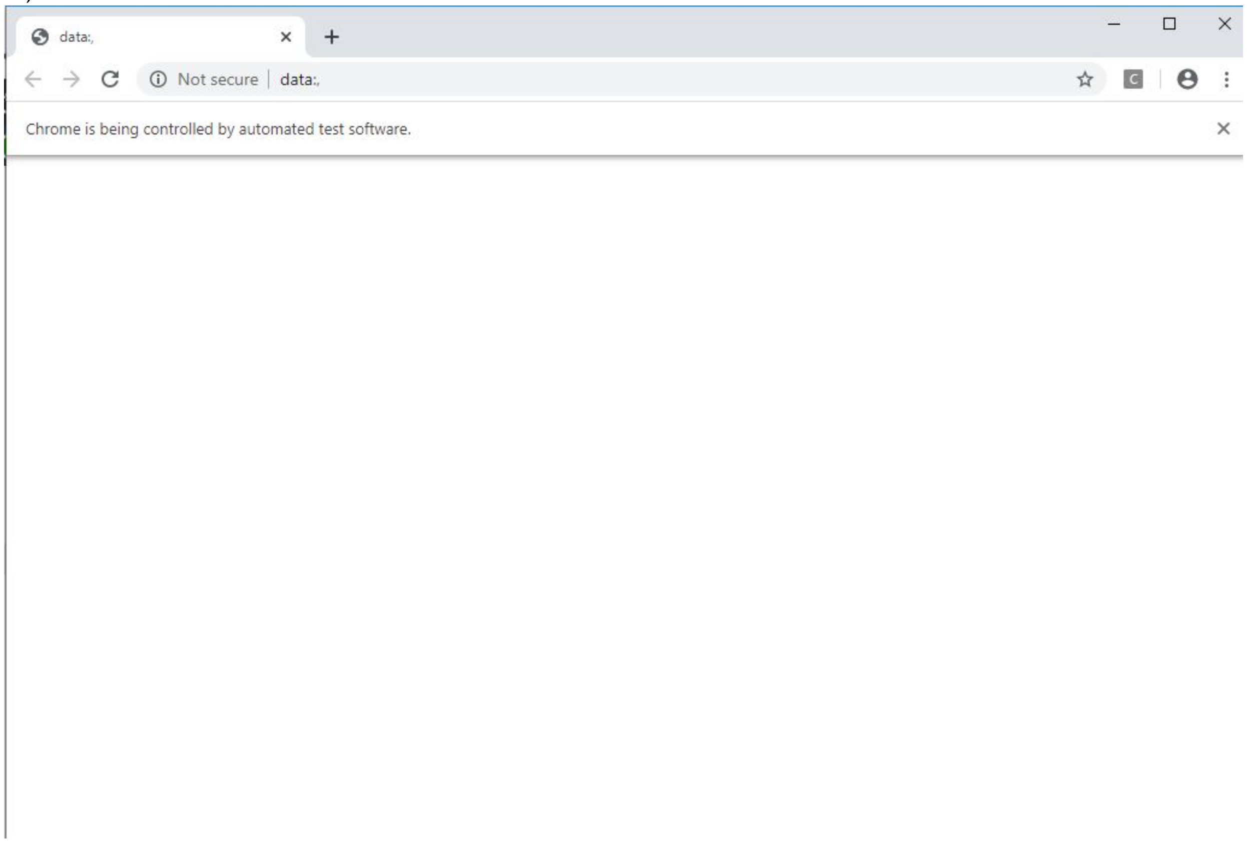
```
> rd<-rsDriver(browser=c("chrome"),chromever="78.0.3904.11")
checking Selenium Server versions:
BEGIN: PREDOWNLOAD
BEGIN: DOWNLOAD
BEGIN: POSTDOWNLOAD
```

```
checking chromedriver versions:
BEGIN: PREDOWNLOAD
BEGIN: DOWNLOAD
BEGIN: POSTDOWNLOAD
checking geckodriver versions:
BEGIN: PREDOWNLOAD
BEGIN: DOWNLOAD
BEGIN: POSTDOWNLOAD
checking phantomjs versions:
BEGIN: PREDOWNLOAD
BEGIN: DOWNLOAD
BEGIN: POSTDOWNLOAD
[1] "Connecting to remote server"
$acceptInsecureCerts
[1] FALSE

$browserName
[1] "chrome"

$browserVersion
[1] "79.0.3945.117"

$chrome
$chrome$chromedriverVersion
[1] "78.0.3904.11 (eaaae9de6b8999773fa33f92ce1e1bbe294437cf-refs/branch-heads/3904
    @{#86})"

$chrome$userDataDir
[1] "C:\\Users\\myles\\AppData\\Local\\Temp\\scoped_dir1044_653722308"


$`goog:chromeOptions`
$`goog:chromeOptions`$debuggerAddress
[1] "localhost:64162"


$networkConnectionEnabled
[1] FALSE

$pageLoadStrategy
[1] "normal"

$platformName
[1] "windows nt"

$proxy
named list()

$setWindowRect
[1] TRUE

$strictFileInteractability
[1] FALSE

$timeouts
$timeouts$implicit
```

```
[1] 0

$timeouts$pageLoad
[1] 300000

$timeouts$script
[1] 30000


$unhandledPromptBehavior
[1] "dismiss and notify"

$webdriver.remote.sessionid
[1] "2d4acfa5e63494664d631810c4364342"

$id
[1] "2d4acfa5e63494664d631810c4364342"
```

Listing 19: Fixing the error message by using one of the suggested version numbers.

You should get the output above, and a new Google Chrome window should open. It should be blank, and it should look like this:



Chrome is being controlled by automated test software.

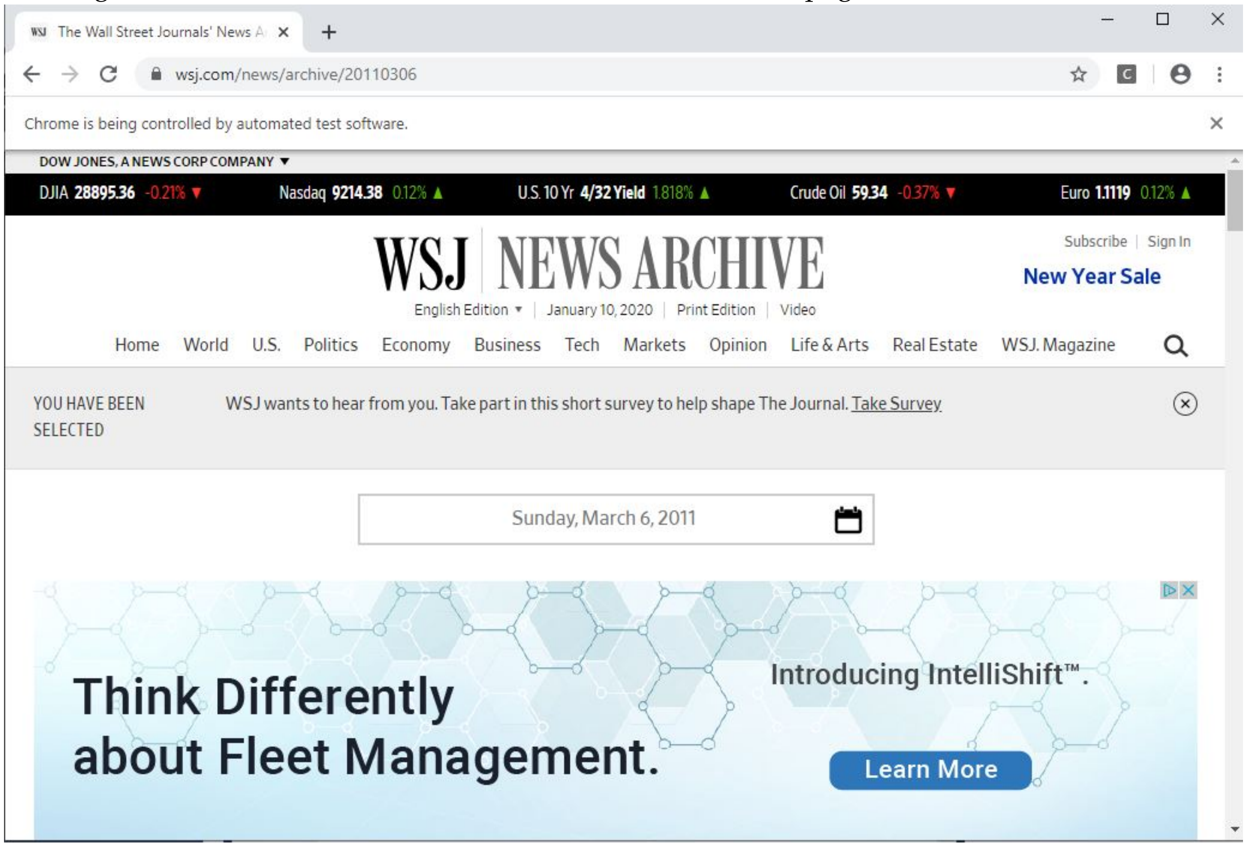
If you get a new window that says "Chrome is being controlled by automated test software", you did this correct, and you are ready to go. If you still get error messages, it means that you either again put in the wrong chrome version (so try a different one on the suggested list!), or you have orphan R sessions. In the latter case, try closing R Studio completely and starting all over again (with the correct version number). If it still does not work, try to restart the entire compute. If it still does not work.....well, you're going to need to figure that one out for yourself!

Now that we have a new chrome browser open, we want to navigate to a website. What we will do is scrape all of the news article titles and links from a given day in the year from the Wall Street Journal. Interesting enough, WSJ has a neat archive portion of their website that organizes this info very neatly. Granted, we can take a static approach for this as well, but we will illustrate the scrapping here with the dynamic approach. We will grab the news archive from March 6th, 2011 (note how we do this in the url):

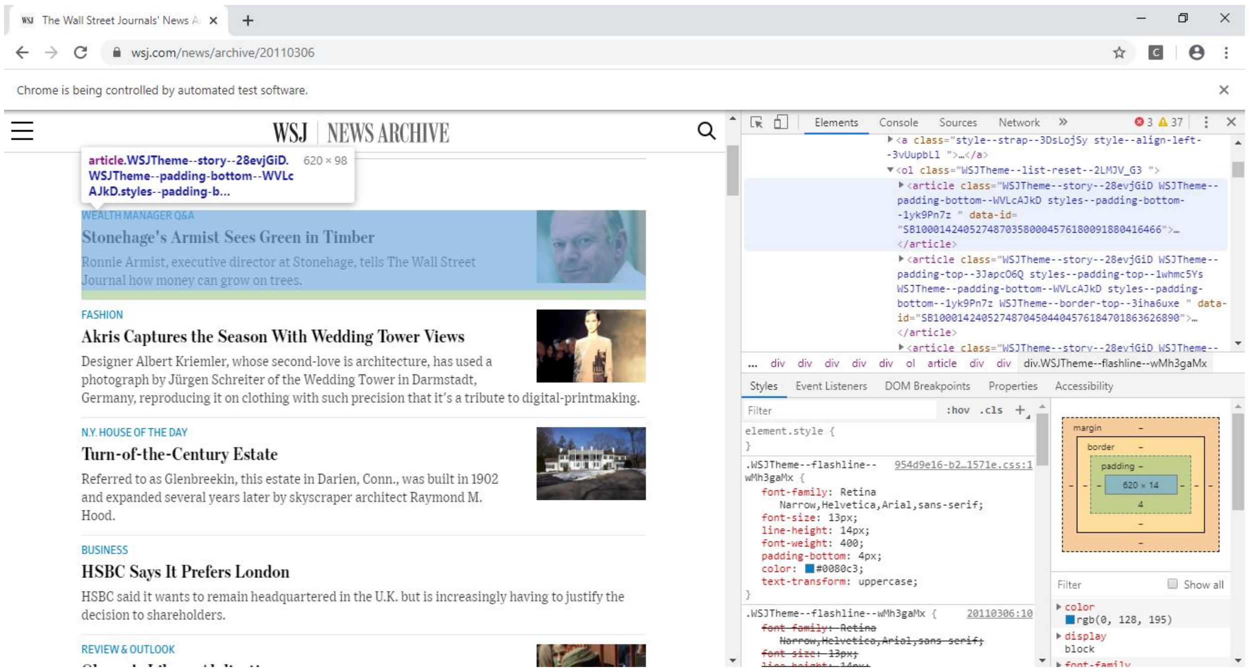
```
> remDr<-rd[["client"]]
> remDr$navigate("http://www.wsj.com/news/archive/20110306")
```

Listing 20: .

First, the first line is grabbing what is called a "handler" to the browser. This basically is like grabbing a line of communication to the web browser. We store the handler in a variable called `remDr`. Next, we use the function in the handler to navigate to the WSJ archive for March 6th, 2011. Doing so, we also notice that our web browser went to that page:


The Wall Street Journals' News A
wsj.com/news/archive/20110306
Chrome is being controlled by automated test software.
DOW JONES, A NEWS CORP COMPANY
DJIA 28895.36 -0.21%
Nasdaq 9214.38 0.12%
U.S. 10 Yr 4/32 Yield 1.818%
Crude Oil 59.34 -0.37%
Euro 1.1119 0.12%
WSJ NEWS ARCHIVE
Subscribe | Sign In
New Year Sale
English Edition | January 10, 2020 | Print Edition | Video
Home World U.S. Politics Economy Business Tech Markets Opinion Life & Arts Real Estate WSJ. Magazine
YOU HAVE BEEN SELECTED
WSJ wants to hear from you. Take part in this short survey to help shape The Journal. Take Survey
Sunday, March 6, 2011
Think Differently about Fleet Management.
Introducing IntelliShift™.
Learn More

Put differently, we just controlled our web browser from R, no differently than if we were to put that url in the address bar itself and press Enter. The same action was taken from our code in R. There are other things we can do as well. In order to know where to pull the information we need, we need to now manually inspect this page so we can gain an idea of the HTML structure. If you scroll through the page in the browser, you will see that all article published on that day appear in one page (very nice indeed!). We want to pull a list of the article category, title, and link, and store this info in a file. To do so, right click on the first article and hit "inspect". You will see the following:



The Wall Street Journals' News A

wsj.com/news/archive/20110306

Chrome is being controlled by automated test software.

WSJ | NEWS ARCHIVE

article.WSJTheme--story--28evjGiD.WSJTheme--padding-bottom--WVLcAJkD.styles--padding-b... 620 × 98

WEALTH MANAGER Q&A
Stonehage's Armist Sees Green in Timber
Ronnie Armist, executive director at Stonehage, tells The Wall Street Journal how money can grow on trees.

FASHION
Akris Captures the Season With Wedding Tower Views
Designer Albert Kriemler, whose second-love is architecture, has used a photograph by Jürgen Schreiter of the Wedding Tower in Darmstadt, Germany, reproducing it on clothing with such precision that it's a tribute to digital-printmaking.

N.Y. HOUSE OF THE DAY
Turn-of-the-Century Estate
Referred to as Glenbreekin, this estate in Darien, Conn., was built in 1902 and expanded several years later by skyscraper architect Raymond M. Hood.

BUSINESS
HSBC Says It Prefers London
HSBC said it wants to remain headquartered in the U.K. but is increasingly having to justify the decision to shareholders.

REVIEW & OUTLOOK

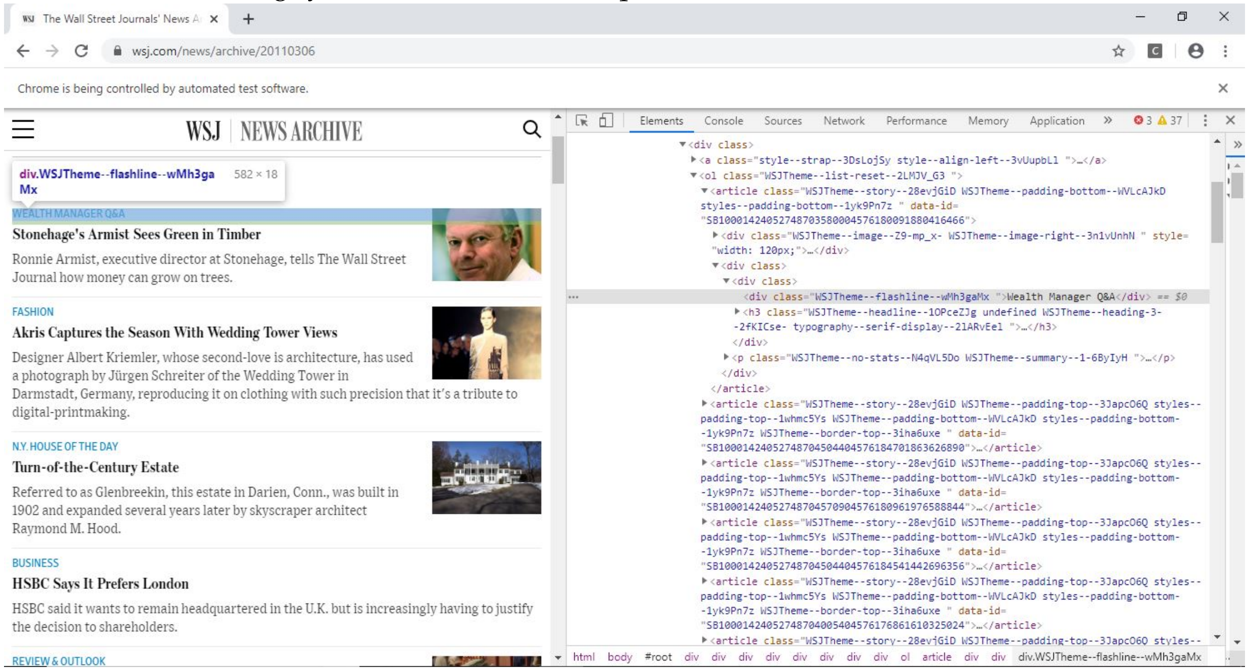
```
▸<a class="style--strap--3DsLojSy style--align-left-
-3vUupbLl ">…</a>
▾<ol class="WSJTheme--list-reset--2LMJV_G3 ">
  ▸<article class="WSJTheme--story--28evjGiD WSJTheme--
  padding-bottom--WVLcAJkD styles--padding-bottom-
  -1yk9Pn7z " data-id=
  "SB10001424052748703580004576180091880416466">…
  </article>
  ▸<article class="WSJTheme--story--28evjGiD WSJTheme--
  padding-top--3Japc06Q styles--padding-top--1whmc5Ys
  WSJTheme--padding-bottom--WVLcAJkD styles--padding-
  bottom--1yk9Pn7z WSJTheme--border-top--3iha6uxe " data-
  id="SB10001424052748704504404576184701863626890">…
  </article>
  ▸<article class="WSJTheme--story--28evjGiD WSJTheme--
```

… div div div div div ol article div div div.WSJTheme--flashline--wMh3gaMx

```
element.style {
}
.WSJTheme--flashline--          954d9e16-b2…1571e.css:1
wMh3gaMx {
   font-family: Retina
      Narrow,Helvetica,Arial,sans-serif;
   font-size: 13px;
   line-height: 14px;
   font-weight: 400;
   padding-bottom: 4px;
   color: ■#0080c3;
   text-transform: uppercase;
}
.WSJTheme--flashline--wMh3gaMx {         20110306:10
   ~~font-family: Retina~~
      ~~Narrow,Helvetica,Arial,sans-serif;~~
   ~~font-size: 13px;~~
```

```
▸color
  ■rgb(0, 128, 195)
▸display
  block
```

Notice in the HTML code that everything is well organized. Each article's information is entirely under an "article" tag. This makes life easy. All we need to do is get all article tags, iterate through each one, and extract the relevant information. Let us further explore inside a single article tag on the HTML tree to determine where the category, title, and link lie in the code. First, the category. After right clicking on the category and hitting "inspect", and some moving of the mouse back and forth until we get to the right area in the code, we can see where the category text lies in a `div` element tag as text, where the class name is "div.WSJTheme–flashline–wMh3gaMx.". While this class name is not unique in the entire HTML, it is unique in the children elements of the article tag (you can check this in a quick text search):



The Wall Street Journals' News A

wsj.com/news/archive/20110306

Chrome is being controlled by automated test software.

WSJ | NEWS ARCHIVE

div.WSJTheme--flashline--wMh3gaMx 582 × 18

WEALTH MANAGER Q&A
Stonehage's Armist Sees Green in Timber
Ronnie Armist, executive director at Stonehage, tells The Wall Street Journal how money can grow on trees.

FASHION
Akris Captures the Season With Wedding Tower Views
Designer Albert Kriemler, whose second-love is architecture, has used a photograph by Jürgen Schreiter of the Wedding Tower in Darmstadt, Germany, reproducing it on clothing with such precision that it's a tribute to digital-printmaking.

N.Y. HOUSE OF THE DAY
Turn-of-the-Century Estate
Referred to as Glenbreekin, this estate in Darien, Conn., was built in 1902 and expanded several years later by skyscraper architect Raymond M. Hood.

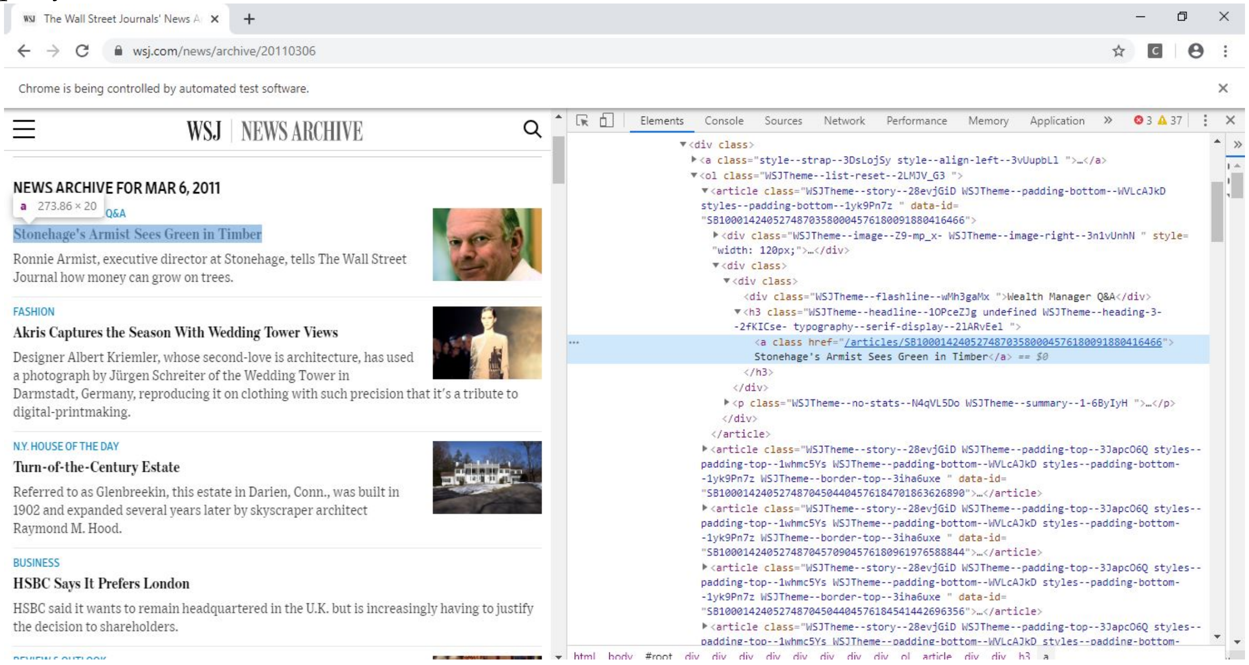
BUSINESS
HSBC Says It Prefers London
HSBC said it wants to remain headquartered in the U.K. but is increasingly having to justify the decision to shareholders.

REVIEW & OUTLOOK

```
▾<div class>
  ▸<a class="style--strap--3DsLojSy style--align-left--3vUupbLl ">…</a>
  ▾<ol class="WSJTheme--list-reset--2LMJV_G3 ">
    ▾<article class="WSJTheme--story--28evjGiD WSJTheme--padding-bottom--WVLcAJkD
    styles--padding-bottom--1yk9Pn7z " data-id=
    "SB10001424052748703580004576180091880416466">
      ▸<div class="WSJTheme--image--Z9-mp_x- WSJTheme--image-right--3n1vUnhN " style=
      "width: 120px;">…</div>
      ▾<div class>
        ▾<div class>
...         <div class="WSJTheme--flashline--wMh3gaMx ">Wealth Manager Q&A</div> == $0
          ▸<h3 class="WSJTheme--headline--1OPceZJg undefined WSJTheme--heading-3-
          -2fKICse- typography--serif-display--21ARvEel ">…</h3>
          </div>
        ▸<p class="WSJTheme--no-stats--N4qVL5Do WSJTheme--summary--1-6ByIyH ">…</p>
        </div>
      </article>
    ▸<article class="WSJTheme--story--28evjGiD WSJTheme--padding-top--3Japc06Q styles--
    padding-top--1whmc5Ys WSJTheme--padding-bottom--WVLcAJkD styles--padding-bottom-
    -1yk9Pn7z WSJTheme--border-top--3iha6uxe " data-id=
    "SB10001424052748704504404576184701863626890">…</article>
    ▸<article class="WSJTheme--story--28evjGiD WSJTheme--padding-top--3Japc06Q styles--
    padding-top--1whmc5Ys WSJTheme--padding-bottom--WVLcAJkD styles--padding-bottom-
    -1yk9Pn7z WSJTheme--border-top--3iha6uxe " data-id=
    "SB10001424052748704570904576180961976588844">…</article>
    ▸<article class="WSJTheme--story--28evjGiD WSJTheme--padding-top--3Japc06Q styles--
    padding-top--1whmc5Ys WSJTheme--padding-bottom--WVLcAJkD styles--padding-bottom-
    -1yk9Pn7z WSJTheme--border-top--3iha6uxe " data-id=
    "SB10001424052748704504404576184541442696356">…</article>
    ▸<article class="WSJTheme--story--28evjGiD WSJTheme--padding-top--3Japc06Q styles--
    padding-top--1whmc5Ys WSJTheme--padding-bottom--WVLcAJkD styles--padding-bottom-
    -1yk9Pn7z WSJTheme--border-top--3iha6uxe " data-id=
    "SB10001424052748704005404576176861610325024">…</article>
    ▸<article class="WSJTheme--story--28evjGiD WSJTheme--padding-top--3Japc06Q styles--
```

html body #root div div div div div div div div ol article div div div.WSJTheme--flashline--wMh3gaMx

Next, we want to see where the title text is. We notice that upon some additional hovering in the inspection tool in Chrome, that the title is element text in an "a" tag. This can be uniquely identified by first finding the h3 tag (since it appears there are no other h3 tags as a descendent of the article element) and then finding the only child of h3, which is the "a" tag we would need. Interestingly enough,the hyperlink to the article is also on this tag, but as a property of the "href" property:


The Wall Street Journals' News A
wsj.com/news/archive/20110306
Chrome is being controlled by automated test software.
WSJ | NEWS ARCHIVE
Elements
Console
Sources
Network
Performance
Memory
Application
3
37
a 273.86 × 20

NEWS ARCHIVE FOR MAR 6, 2011

Q&A
Stonehage's Armist Sees Green in Timber
Ronnie Armist, executive director at Stonehage, tells The Wall Street Journal how money can grow on trees.

FASHION
Akris Captures the Season With Wedding Tower Views
Designer Albert Kriemler, whose second-love is architecture, has used a photograph by Jürgen Schreiter of the Wedding Tower in Darmstadt, Germany, reproducing it on clothing with such precision that it's a tribute to digital-printmaking.

N.Y. HOUSE OF THE DAY
Turn-of-the-Century Estate
Referred to as Glenbreekin, this estate in Darien, Conn., was built in 1902 and expanded several years later by skyscraper architect Raymond M. Hood.

BUSINESS
HSBC Says It Prefers London
HSBC said it wants to remain headquartered in the U.K. but is increasingly having to justify the decision to shareholders.

```
▼<div class>
  ▶<a class="style--strap--3DsLojSy style--align-left--3vUupbLl ">…</a>
  ▼<ol class="WSJTheme--list-reset--2LMJV_G3 ">
    ▼<article class="WSJTheme--story--28evjGiD WSJTheme--padding-bottom--WVLcAJkD
    styles--padding-bottom--1yk9Pn7z " data-id=
    "SB10001424052748703580004576180091880416466">
      ▶<div class="WSJTheme--image--Z9-mp_x- WSJTheme--image-right--3n1vUnhN " style=
      "width: 120px;">…</div>
      ▼<div class>
        ▼<div class>
            <div class="WSJTheme--flashline--wMh3gaMx ">Wealth Manager Q&A</div>
          ▼<h3 class="WSJTheme--headline--1OPceZJg undefined WSJTheme--heading-3-
          -2fKICse- typography--serif-display--21ARvEel ">
...           <a class href="/articles/SB10001424052748703580004576180091880416466">
              Stonehage's Armist Sees Green in Timber</a> == $0
            </h3>
          </div>
        ▶<p class="WSJTheme--no-stats--N4qVL5Do WSJTheme--summary--1-6ByIyH ">…</p>
        </div>
      </article>
    ▶<article class="WSJTheme--story--28evjGiD WSJTheme--padding-top--3Japc06Q styles--
    padding-top--1whmc5Ys WSJTheme--padding-bottom--WVLcAJkD styles--padding-bottom-
    -1yk9Pn7z WSJTheme--border-top--3iha6uxe " data-id=
    "SB10001424052748704504404576184701863626890">…</article>
    ▶<article class="WSJTheme--story--28evjGiD WSJTheme--padding-top--3Japc06Q styles--
    padding-top--1whmc5Ys WSJTheme--padding-bottom--WVLcAJkD styles--padding-bottom-
    -1yk9Pn7z WSJTheme--border-top--3iha6uxe " data-id=
    "SB10001424052748704570904576180961976588844">…</article>
    ▶<article class="WSJTheme--story--28evjGiD WSJTheme--padding-top--3Japc06Q styles--
    padding-top--1whmc5Ys WSJTheme--padding-bottom--WVLcAJkD styles--padding-bottom-
    -1yk9Pn7z WSJTheme--border-top--3iha6uxe " data-id=
    "SB10001424052748704504404576184541442696356">…</article>
    ▶<article class="WSJTheme--story--28evjGiD WSJTheme--padding-top--3Japc06Q styles--
    padding-top--1whmc5Ys WSJTheme--padding-bottom--WVLcAJkD styles--padding-bottom-
html  body  #root  div  div  div  div  div  div  div  div  ol  article  div  div  h3  a
```

To test our theory here, let us grab the page source, and navigate the HTML in the browser. We will not use `rvest` for this, but rather RSelenium's functionality (which is slightly different, and closer to that of Javascripts):

```
> #Get All Elements in Tree that is an Article Tag:
> articles<-remDr$findElements(using="tag","article")
>
> #Test the first article only:
> article<-articles[[1]]
>
> #See if we can find the div tag with the category text:
> category<-article$findChildElement(using="class","WSJTheme--flashline--wMh3gaMx"
  )
>
> #Get The tag name
> category$getElementTagName()
[[1]]
[1] "div"

>
> #We see it's a div! Lets get the inner text:
> cat_text<-category$getElementText()[[1]]
>
> #Now, let us search for ONLY the h3 tag only under this article tag
> title<-article$findChildElement(using="tag","h3")
>
> #If we did find the h3 tag, then the a tag should be directly under it:
```

```
> title_link<-title$findChildElement(using="tag","a")
>
> #Grab the title of the article:
> title_text<-title_link$getElementText()[[1]]
>
> #Last, grab the hyperlink to the article:
> article_link<-title_link$getElementAttribute("href")[[1]]
>
> #Check and See!
> cat_text
[1] "WEALTH MANAGER Q&A"
> title_text
[1] "Stonehage's Armist Sees Green in Timber"
> article_link
[1] "https://www.wsj.com/articles/SB10001424052748703580004576180091880416466"
```

Listing 21: Testing an extraction method of the category and title and article link information.

Now we want to do exact this info for all the articles on the page. Let's do that!

```
> articles<-remDr$findElements(using="tag","article")
> the_data<-c()
>
> for(article in articles){
+   category<-article$findChildElement(using="class","WSJTheme--flashline--
  wMh3gaMx")
+   cat_text<-category$getElementText()[[1]]
+   title<-article$findChildElement(using="tag","h3")
+   title_link<-title$findChildElement(using="tag","a")
+   title_text<-title_link$getElementText()[[1]]
+   article_link<-title_link$getElementAttribute("href")[[1]]
+   the_data<-c(the_data,cat_text,title_text,article_link)
+ }
>
> the_data<-matrix(the_data,byrow=TRUE,ncol=3)
> the_data<-data.frame(the_data)
> names(the_data)<-c("category","title","link")
> head(the_data)
            category                                          title
1    WEALTH MANAGER Q&A         Stonehage's Armist Sees Green in Timber
2             FASHION Akris Captures the Season With Wedding Tower Views
3 N.Y. HOUSE OF THE DAY                      Turn-of-the-Century Estate
4            BUSINESS                       HSBC Says It Prefers London
5             OPINION                         Obama's Libyan Abdication
6      MANAGING IN ASIA                  Robson Brings Game to Thai Soccer
                                                                     link
1 https://www.wsj.com/articles/SB10001424052748703580004576180091880416466
2 https://www.wsj.com/articles/SB10001424052748704504404576184701863626890
3 https://www.wsj.com/articles/SB10001424052748704570904576180961976588844
4 https://www.wsj.com/articles/SB10001424052748704504404576184541442696356
5 https://www.wsj.com/articles/SB10001424052748704005404576176861610325024
6 https://www.wsj.com/articles/SB10001424052748704288304576171371317870688
```

Listing 22: Doing the extraction for all articles and organizing the results in a data frame.

Last, in RSelenium, as we have mentioned, you can interact with the interface of the web browser directly. For example, we may want to navigate to one of the articles. We can certainly do this by just copy and pasting the hyperlink and using the "navigate" function. The other way is to tell the browser to "click" on a link, so long as we "move" to the element in the HTML tree via R:

```r
#Move to the element link on the second article, then click it:
articles<-remDr$findElements(using="tag","article")
article<-articles[[2]]
title<-article$findChildElement(using="tag","h3")
title_link<-title$findChildElement(using="tag","a")
title_link$clickElement()
```

Listing 23: Showing off the interactive ability of RSelenium by having the browser click on an article link.

If you go to the browser, you will now see the page for the second link up. To test other ways to interact with the browser, let us try to write code so that the browser will navigate to the main WSJ site, click on the "Search" button", open up a search box, type in "Apple and Microsoft", and then click "Search". The class names come from a series of "inspects" on each element we want to take an action on:

```r
> remDr$navigate("http://wsj.com")
> search_button<-remDr$findElement(using="class","style--search-button--1U43LWwP")
> search_button$clickElement()
> text_box<-remDr$findElement(using="class","style--wsj-search-input--R7oQjW8f")
> text_box$clickElement()
> text_box$sendKeysToElement(list("Apple and Microsoft"))
> send_button<-remDr$findElement("class","style--search-submit--36AD7RDg")
> send_button$clickElement()
> article$clickElement()
```

Listing 24: Using RSelenium to navigate to WSJ and subsequently search for terms automatically.

While this method of web scrapping may be slower, it is more reliable. The reason is simple: we are able to "fool" the ISP and the Website into thinking it is an actual person browsing the website. This allows us to essentially see, code wise, any data that we can see directly in our browser. Dynamic web browsing is an art that takes a lot of practice. My recommendation would be to continue trying exercises found on the web for "RSelenium" examples.

Exercises

10. Go to http://weather.com. Using RSelenium, write code that will take the following sequence:

    - Navigate to the main website
    - Load "Wayne, NJ" weather.
    - Download the table for the 5-Day Forecast

    ONLY use RSelenium through a series of finds and clicks. You will obviously need to also use the Google Chrome's "inspect" tool.