Data Science for Economists

Lecture 11: Webscraping part 1 - CSS

Drew Van Kuiken
University of North Carolina | ECON 370

The Internet as Data

Let's get it onto our computers.

For today, load **rvest** and **janitor** into your R session, alongside **tidyverse**, **lubridate**, **data.table**, and **hrbrthemes**

```r
## Load and install the packages that we'll be using today
if (!require("pacman")) install.packages("pacman")
pacman::p_load(tidyverse, rvest, lubridate, janitor, data.table, hrbrthemes)
## ggplot2 plotting theme (optional)
theme_set(hrbrthemes::theme_ipsum())
```

Much credit to Grant McDermott for the content on these slides.

Internet Data Is Stored in 2 Ways:

1. Server-side
    - Data stored at the server, which sends HTML code with data in it to us
    - **Our process**: trudging through CSS selectors
2. Client-side
    - Our browser requests data from the server, server sends specific info we asked for
    - **Our process**: pinging an API endpoint

This lecture will focus on server-side scraping; we'll do client-side scraping next

Before we get started

1. Be a good internet user
2. It's easy to accidentally kill some poor website
3. It's probably legal?

Our main package today is **rvest**, part of the tidyverse. Based on **Beautiful Soup**

HTML Basics

Here's some simple HTML:

```html
<html>
<head>
  <title>Page title</title>
</head>
<body>
  <h1 id='first'>A heading</h1>
  <p>Some text &amp; <b>some bold text.</b></p>
  <img src='myimg.png' width='100' height='100'>
</body>
```

HTML Basics

Here's some simple HTML:

```html
<html>
<head> ## head is an element #<<
  <title>Page title</title>
</head>
<body>
  <h1 id='first'>A heading</h1>
  <p>Some text &amp; <b>some bold text.</b></p>
  <img src='myimg.png' width='100' height='100'>
</body>
```

HTML Basics

Here's some simple HTML:

```
<html>
<head>
  <title>Page title</title>
</head>
<body> ## as is body #<<
  <h1 id='first'>A heading</h1>
  <p>Some text &amp; <b>some bold text.</b></p>
  <img src='myimg.png' width='100' height='100'>
</body>
```

HTML Basics

Here's some simple HTML:

```html
<html>
<head>
  <title>Page title</title>
</head>
<body>
  <h1 id='first'>A heading</h1> ## id='first' is an attribute #<<
  <p>Some text &amp; <b>some bold text.</b></p>
  <img src='myimg.png' width='100' height='100'>
</body>
```

HTML Basics

Here's some simple HTML:

```html
<html>
<head>
  <title>Page title</title>
</head>
<body>
  <h1 id='first'>A heading</h1>
  <p>Some text &amp; <b>some bold text.</b></p> ## stuff in between is contents #<<
  <img src='myimg.png' width='100' height='100'>
</body>
```

Lucky for us: we don't need to write HTML. Just read it.

HTML Basics - Tags

- Tags all start with `<tag>` and end with `</tag>`
- Every page is in an `<html>` element with 2 children:
  - `<head>` contains metadata
  - `<body>` contains content you see
- Block tags form the overall structure of a page
  - `<h1>` provides a heading
  - `<p>` is a paragraph
- Inline tags like `<b>` (bold), `<i>` (italics), and `<a>` (links) exist
- Just a sample of tags, can look up others you don't know
- The rest is the content

Example 1: Scraping Wikipedia

Let's imagine we want to scrape the Men's 100 metres world record progression page on Wikipedia.

In particular, we want to get the information from the 3 main tables on the webpage.

Here's what happens when we give R no instructions at all:

```
raw_wiki ← read_html("https://en.wikipedia.org/wiki/Men%27s_100_metres_world_record_p
raw_wiki
```

```
## {html_document}
## <html class="client-nojs vector-feature-language-in-header-enabled vector-feature-langua
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...
## [2] <body class="skin--responsive skin-vector skin-vector-search-vue mediawik ...
```

```
class(raw_wiki)
```

```
## [1] "xml_document" "xml_node"
```

Parsing HTML: Inspecting Web Pages

We can use `inspect` to get a better sense of what is available on a given webpage. What tags can we use to grab the desired information from our wikipedia page?

It looks like we want to grab information from tables with the class "wikitable."

Brief Aside: CSS Selectors

We can parse HTML using **CSS Selectors**, which define patterns for locating HTML elements.

CSS Selectors are pretty complex, and we're going to keep it light today. If you want to learn more, check out the CSS Diner. Additionally, if CSS Selectors are giving you tons of trouble, check out SelectorGadget.

4 selectors to know:

1. `p` selects all `<p>` elements
2. `.title` selects all elements with `class` "title"
3. `p.special` selects all `<p>` elements with `class` "special"
4. `#title` selects the unique element with id attribute that equals "title"

Getting Our Tables

We can use the selector `table` to select all `<table>` elements:

```r
raw_wiki |> html_elements("table")
```

```
## {xml_nodeset (15)}
##  [1] <table class="box-Unreferenced_section plainlinks metadata ambox ambox-c ...
##  [2] <table class="wikitable"><tbody>\n<tr>\n<th>Time\n</th>\n<th>Athlete\n</ ...
##  [3] <table class="wikitable" style="text-align: left;"><tbody>\n<tr>\n<td st ...
##  [4] <table class="wikitable"><tbody>\n<tr>\n<th>Time\n</th>\n<th>Wind\n</th> ...
##  [5] <table class="wikitable"><tbody>\n<tr>\n<th>Time\n</th>\n<th>Wind\n</th> ...
##  [6] <table class="wikitable"><tbody>\n<tr>\n<th>Time\n</th>\n<th>Athlete\n</ ...
##  [7] <table class="nowraplinks mw-collapsible autocollapse navbox-inner" styl ...
##  [8] <table class="nowraplinks navbox-subgroup" style="border-spacing:0"><tbo ...
##  [9] <table class="nowraplinks hlist mw-collapsible autocollapse navbox-inner ...
## [10] <table class="nowraplinks navbox-subgroup" style="border-spacing:0"><tbo ...
## [11] <table class="nowraplinks navbox-subgroup" style="border-spacing:0"><tbo ...
## [12] <table class="nowraplinks mw-collapsible uncollapsed navbox-inner" style ...
## [13] <table class="nowraplinks navbox-subgroup" style="border-spacing:0"><tbo ...
## [14] <table class="nowraplinks navbox-subgroup" style="border-spacing:0"><tbo ...
## [15] <table class="nowraplinks navbox-subgroup" style="border-spacing:0"><tbo ...
```

Alas, so are many other things.

Trying again

We want tables with class wikitable: `table.wikitable` should work!

```
raw_wiki |> html_elements("table.wikitable")
```

```
## {xml_nodeset (5)}
## [1] <table class="wikitable"><tbody>\n<tr>\n<th>Time\n</th>\n<th>Athlete\n</t ...
## [2] <table class="wikitable" style="text-align: left;"><tbody>\n<tr>\n<td sty ...
## [3] <table class="wikitable"><tbody>\n<tr>\n<th>Time\n</th>\n<th>Wind\n</th>\ ...
## [4] <table class="wikitable"><tbody>\n<tr>\n<th>Time\n</th>\n<th>Wind\n</th>\ ...
## [5] <table class="wikitable"><tbody>\n<tr>\n<th>Time\n</th>\n<th>Athlete\n</t ...
```

Looks better!

Accessing the actual info

We still don't have the table info. `html_table()` can help:

```
table_dfs ← raw_wiki ▷ html_elements("table.wikitable") ▷
  html_table()
table_dfs[[1]]
```

```
## # A tibble: 21 × 5
##     Time Athlete                Nationality    `Location of races`     Date
##    <dbl> <chr>                  <chr>          <chr>                   <chr>
##  1  10.8 Luther Cary            United States  Paris, France           July 4, 1…
##  2  10.8 Cecil Lee              United Kingdom Brussels, Belgium       September…
##  3  10.8 Étienne De Ré          Belgium        Brussels, Belgium       August 4,…
##  4  10.8 L. Atcherley           United Kingdom Frankfurt/Main, Germany April 13,…
##  5  10.8 Harry Beaton           United Kingdom Rotterdam, Netherlands  August 28…
##  6  10.8 Harald Anderson-Arbin  Sweden         Helsingborg, Sweden     August 9,…
##  7  10.8 Isaac Westergren       Sweden         Gävle, Sweden           September…
##  8  10.8 Isaac Westergren       Sweden         Gävle, Sweden           September…
##  9  10.8 Frank Jarvis           United States  Paris, France           July 14, …
## 10  10.8 Walter Tewksbury       United States  Paris, France           July 14, …
## # ℹ 11 more rows
```

...this is crazy! `R` is cool sometimes.

General Workflow

Your workflow using `rvest`: get html --> get desired html elements --> break into individual elements --> turn into table/text/numbers/whatever.

Useful commands for getting individual elements:

- `html_text2()` gets plain text contents of an HTML element
- `html_attr()` gets attributes from an HTML element (e.g., links)
- `html_table()` creates a data.frame from a table in an HTML element

A Little Cleanup

Let's get our data frames in working order here:

```
table_dfs_int ← raw_wiki ▷ html_elements("table.wikitable") ▷
  html_table()

table_dfs ← lapply(table_dfs_int[c(1,3,4)], # drop unwanted tables
                    function(x) x ▷
                      clean_names() ▷ ## fix colnames, from the janitor package
                      mutate(date = mdy(date))) ## from lubridate
table_dfs[[1]]
```

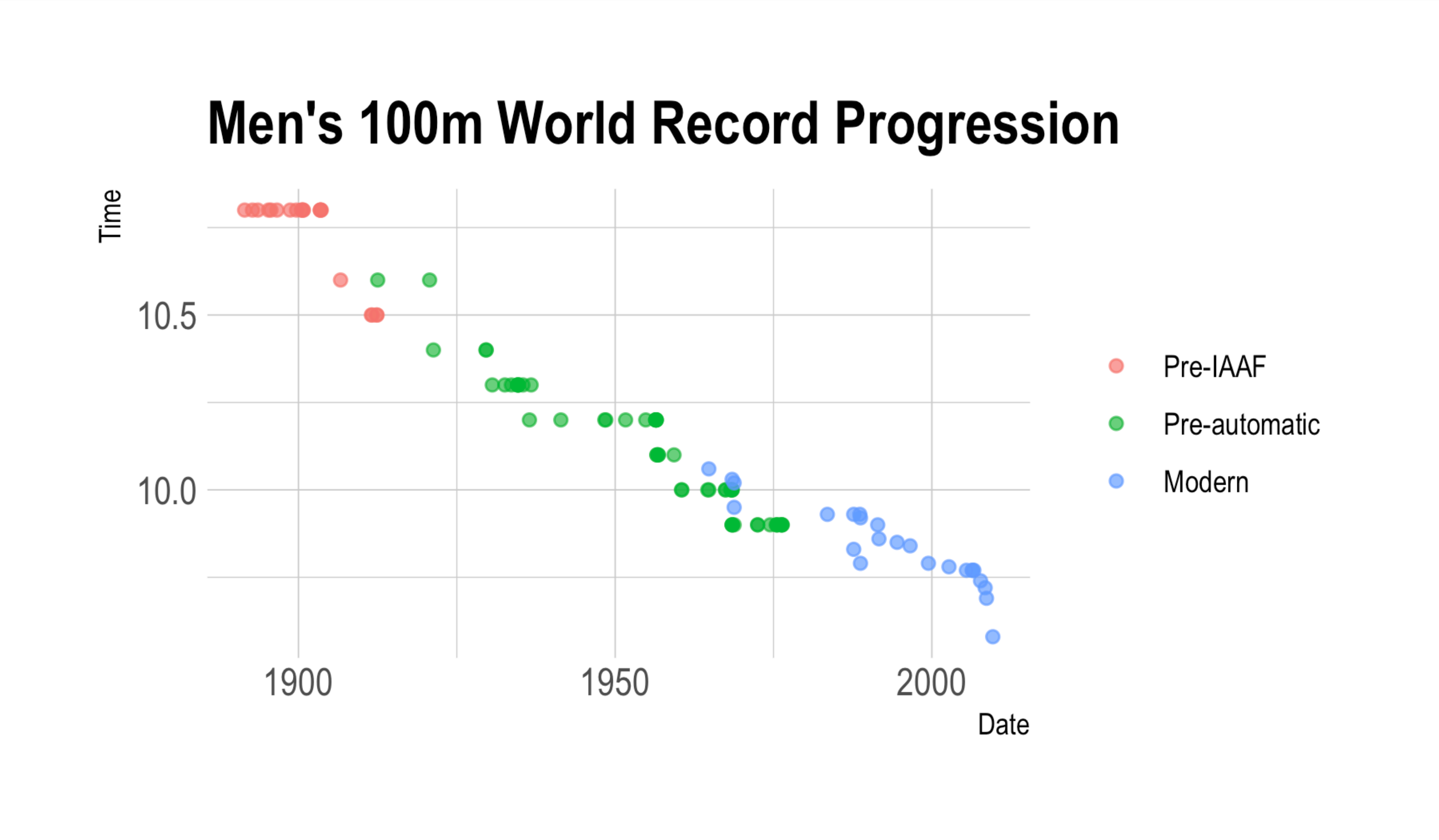
```
## # A tibble: 21 × 5
##     time athlete               nationality    location_of_races       date
##    <dbl> <chr>                 <chr>          <chr>                   <date>
##  1  10.8 Luther Cary           United States  Paris, France           1891-07-04
##  2  10.8 Cecil Lee             United Kingdom Brussels, Belgium       1892-09-25
##  3  10.8 Étienne De Ré         Belgium        Brussels, Belgium       1893-08-04
##  4  10.8 L. Atcherley          United Kingdom Frankfurt/Main, Germany 1895-04-13
##  5  10.8 Harry Beaton          United Kingdom Rotterdam, Netherlands  1895-08-28
##  6  10.8 Harald Anderson-Arbin Sweden         Helsingborg, Sweden     1896-08-09
##  7  10.8 Isaac Westergren      Sweden         Gävle, Sweden           1898-09-11
##  8  10.8 Isaac Westergren      Sweden         Gävle, Sweden           1899-09-10
##  9  10.8 Frank Jarvis          United States  Paris, France           1900-07-14
## 10  10.8 Walter Tewksbury      United States  Paris, France           1900-07-14
```

Combined Data

```
wr100 ← rbind(
  table_dfs[[1]] ▷ select(time, athlete, nationality, date) ▷
    mutate(era="Pre-IAAF"),
  table_dfs[[2]] ▷ select(time, athlete, nationality, date) ▷
    mutate(era="Pre-automatic"),
  table_dfs[[3]] ▷ select(time, athlete, nationality, date) ▷
    mutate(era="Modern")
)
head(wr100)
```

```
## # A tibble: 6 × 5
##    time athlete              nationality    date       era
##   <dbl> <chr>                <chr>          <date>     <chr>
## 1  10.8 Luther Cary          United States  1891-07-04 Pre-IAAF
## 2  10.8 Cecil Lee            United Kingdom 1892-09-25 Pre-IAAF
## 3  10.8 Étienne De Ré        Belgium        1893-08-04 Pre-IAAF
## 4  10.8 L. Atcherley         United Kingdom 1895-04-13 Pre-IAAF
## 5  10.8 Harry Beaton         United Kingdom 1895-08-28 Pre-IAAF
## 6  10.8 Harald Anderson-Arbin Sweden        1896-08-09 Pre-IAAF
```

Dot Plot



That was easy. How about something harder.

Craigslist - Watch Prices

Let's take a look at watch prices on Craigslist in Raleigh.

We want to know: listing names, prices, locations, and links for the Raleigh craigslist.

Use the shell code below to poke around and see if you can download the information we want.

```
# read in html, get listing info
web <- "https://raleigh.craigslist.org/search/jwa?query=watch#search=1~gallery~0~0"
craigslist_listings <- read_html(web)|>
  html_elements("[INSERT ELEMENTS TO GET CODE HERE]")
craigslist_listings[[1]]
```

(Remember our basic strategy: `read_html` --> `html_elements` --> `html_element` --> `html_text2`. We're covering steps 1 and 2 here, don't worry about having your output look perfect.)

My Solution

(Admittedly, it took a while for me to figure this out. Art, not science.)

All kinds of info is saved in `li a`!

We can use `html_elements()` to find an element that corresponds to each listing, then use `html_element()` to extract each individual variable:

```r
# read in html, get listing info
web <- "https://raleigh.craigslist.org/search/jwa?query=watch#search=1~gallery~0~0"
craigslist_listings <- read_html(web) |> html_elements("li a")
craigslist_listings[[1]]
```

```
## {html_node}
## <a href="https://raleigh.craigslist.org/jwl/d/raleigh-vintage-hamilton-mens-automatic/7
## [1] <div class="title">Vintage Hamilton Men's Automatic Watch</div>
## [2] <div class="details">\n                    <div class="price">$100</div>\  ...
```

Looks promising!

Diving Deeper

```r
# follow branching tree further:
# title of listing
title ← craigslist_listings ▷ html_elements("div.title") ▷ html_text2()

# seems like 2 pieces of info stored in div.details
details ← craigslist_listings ▷ html_elements("div.details")
price ← details ▷ html_element("div.price") ▷ html_text2()
location ← details ▷ html_element("div.location") ▷ html_text2()

# we can use html_attr to grab the link to the listing
link ← craigslist_listings ▷ html_attr("href")
```

Taking a Look at Our Data

```
title[1]
```

```
## [1] "Vintage Hamilton Men's Automatic Watch"
```

```
price[1]
```

```
## [1] "$100"
```

```
location[1]
```

```
## [1] "Stonehenge, Raleigh NC"
```

```
link[1]
```

```
## [1] "https://raleigh.craigslist.org/jwl/d/raleigh-vintage-hamilton-mens-automatic/779607
```

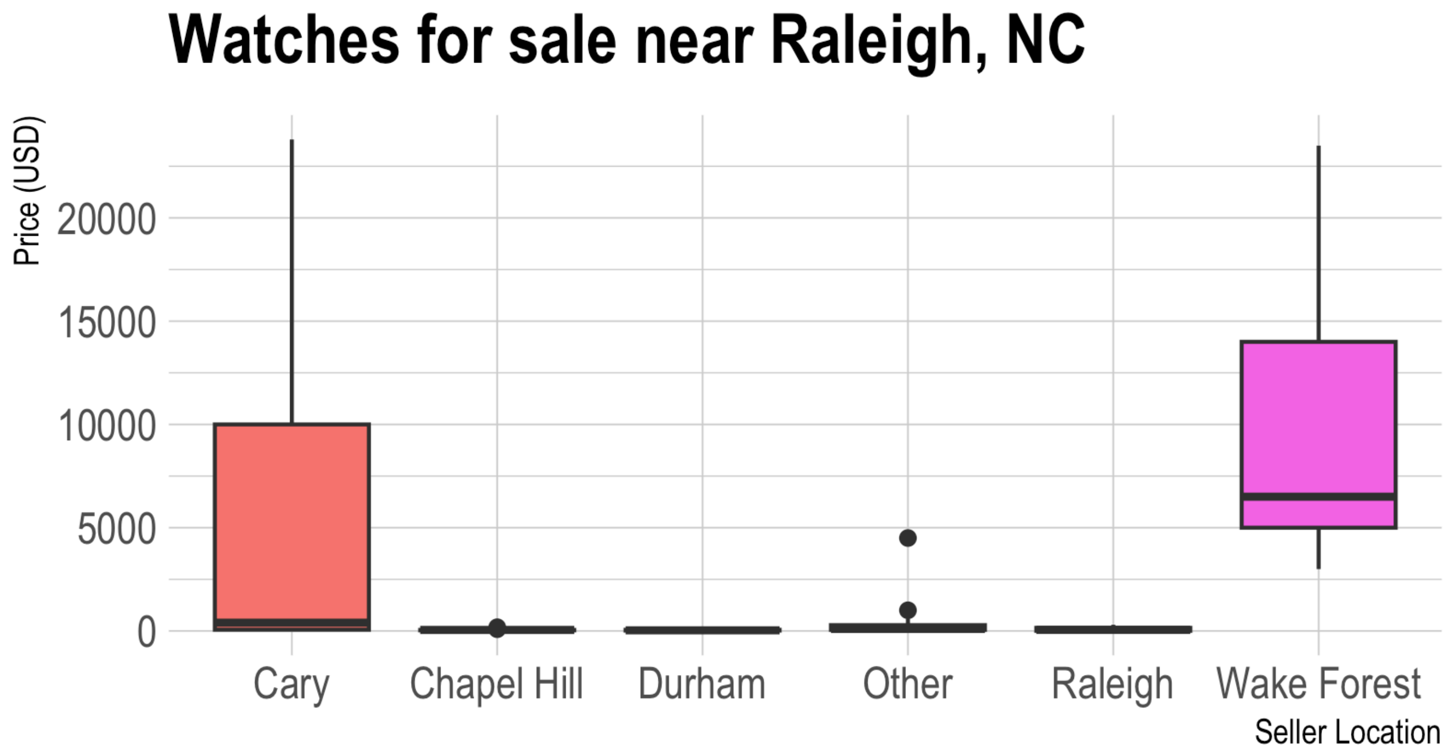
Looking *really* good

Stick them in a data frame, clean up

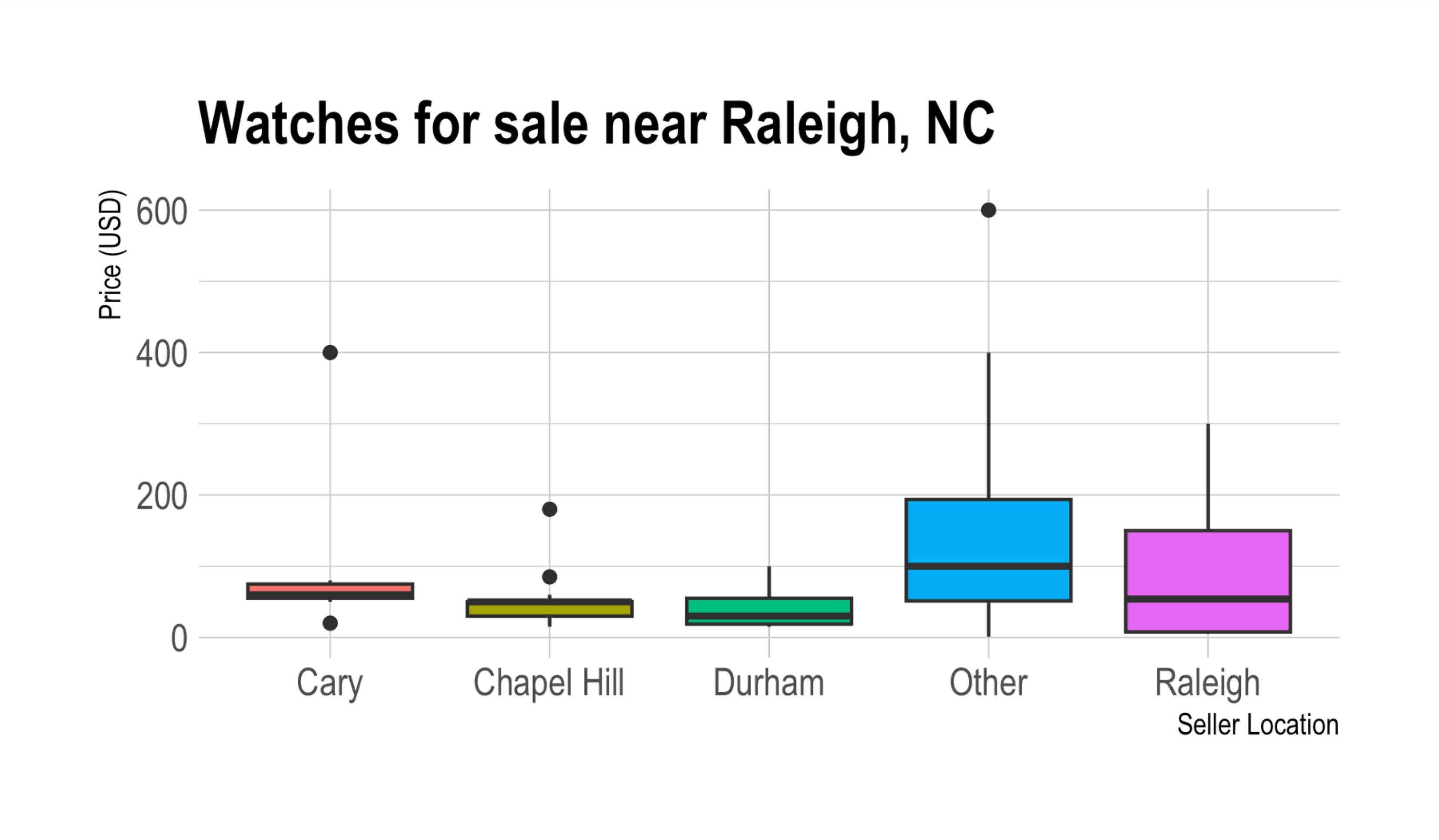
```
watch_data ← data.frame(title,price,location,link) ▷
  mutate(price = parse_number(price), # get rid of $s
         clean_location = case_when(
           grepl("durham",location, ignore.case=TRUE) ~ "Durham",
           grepl("cary|apex",location, ignore.case=TRUE) ~ "Cary",
           grepl("raleigh",location, ignore.case=TRUE) ~ "Raleigh",
           grepl("chapel hill",location, ignore.case=TRUE) ~ "Chapel Hill",
           grepl("wake forest",location, ignore.case=TRUE) ~ "Wake Forest",
           .default = "Other"
         )) ▷
  filter(price>0)
```

How about another plot?

Plot


Watches for sale near Raleigh, NC
Price (USD)
20000
15000
10000
5000
0
Cary
Chapel Hill
Durham
Other
Raleigh
Wake Forest
Seller Location

Plot (for students)



Sidenote: Wake Forest is gone!

Summary

There's two ways to get data from the internet: server-side and client-side

We covered the server-side stuff today

To do so, we did some mucking around with CSS selectors

It is an art, not a science.

Another Tool

Our approach worked pretty well for Craigslist, but we also got fairly lucky: there was only one page of search results when I wrote these notes.

It turns out you can use R to drive your actual computer, clicking on buttons (like Next Page) and so on.

You can do this using RSelenium, which is a huge pain but sometimes incredibly cool.

Next Class

We'll be downloading macroeconomic data from FRED

Before class ***please*** make an account with FRED https://fredaccount.stlouisfed.org/login/secure/

and obtain an API key https://fredaccount.stlouisfed.org/apikey!!!

PS: your api key is a secret, don't share it with people

Next lecture: Scraping Client-Side with APIs