

Lecture 6: Data Acquisition - Scraping Dynamic Websites

James Sears*

AFRE 891 SS 24

Michigan State University

*Parts of these slides related to APIs are adapted from [“Data Science for Economists”](#) by Grant McDermott.

Table of Contents

1. Prologue
2. Scraping Dynamic Websites

Prologue

Packages for Today

While we now have a thorough understanding of how to scrape content from **static websites**, those aren't the only type of sites we'll interact with.

To explore this we'll need the following packages:

```
pacman::p_load(rvest, tidyverse, chromote, selenider)
```

Scraping Dynamic Websites

Dynamic Websites

Dynamic websites are those are rendered dynamically and require **interaction** in order to obtain desired information.

While **rvest** can help us retrieve urls and element contents, it can't interact with **dynamically-generated or interactive** websites.¹

To do that, we're going to take advantage of **browser automation** methods.

¹ **rvest** has some experimental functionality for live site interaction, but it is not as complete as the other options we'll see shortly.

Browser Automation

The main idea behind browser automation is this:

If a website requires pointing/clicking/typing to reveal the contents we want, why not have software "drive" a browser for us?

Old Approach: RSelenium

- R bindings for the Selenium webdriver
- Requires some additional software (i.e. Docker)
- Additional steps to get it working with Mac OS X

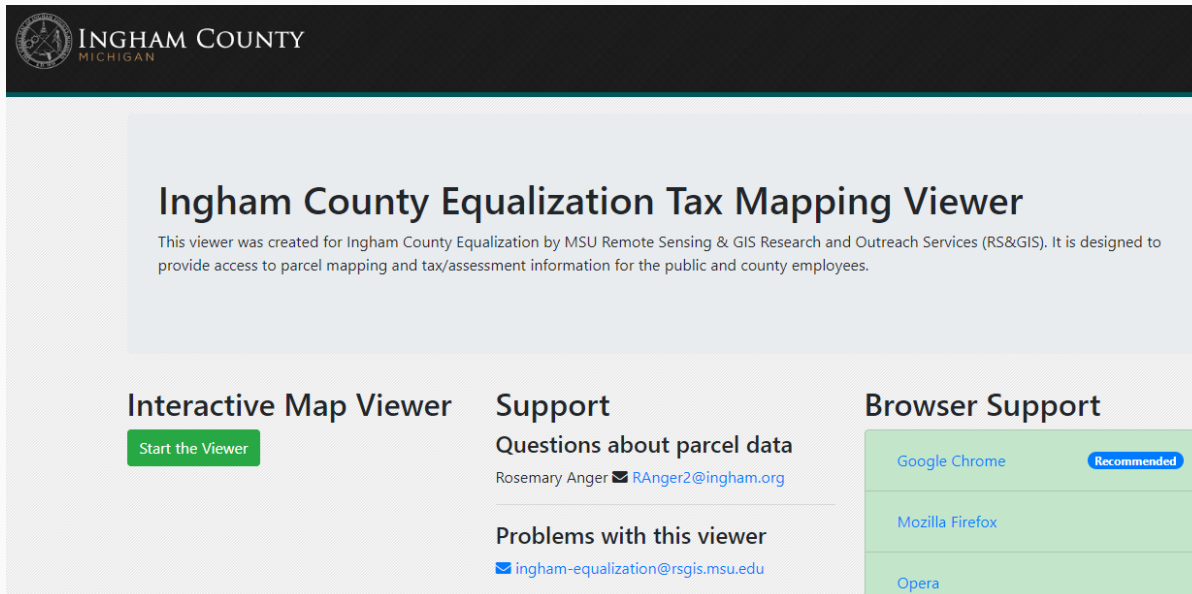
New Approach: selenider

- Uses **chromote** or Selenium automation tools to drive a Chrome browser
- Syntax more closely matches tidyverse
- Can't do some more complex things yet (i.e. click-and-hold, click a point)

Application: EL Properties

Let's work through an example of an interactive workflow for **property characteristics in East Lansing**.

Suppose you want to obtain property information for a known address within East Lansing. The city has a handy **interactive map viewer** for tax assessment info.



The screenshot shows the homepage of the Ingham County Equalization Tax Mapping Viewer. At the top is the Ingham County Michigan logo. The main heading is "Ingham County Equalization Tax Mapping Viewer". Below this is a descriptive paragraph: "This viewer was created for Ingham County Equalization by MSU Remote Sensing & GIS Research and Outreach Services (RS&GIS). It is designed to provide access to parcel mapping and tax/assessment information for the public and county employees." The page is divided into three columns. The left column, titled "Interactive Map Viewer", contains a green button labeled "Start the Viewer". The middle column, titled "Support", includes the text "Questions about parcel data" followed by "Rosemary Anger" and an email link "RAnger2@ingham.org", and "Problems with this viewer" followed by an email link "ingham-equalization@rsgis.msu.edu". The right column, titled "Browser Support", lists "Google Chrome" (marked as "Recommended"), "Mozilla Firefox", and "Opera" in a light green box.

INGHAM COUNTY
MICHIGAN

Ingham County Equalization Tax Mapping Viewer

This viewer was created for Ingham County Equalization by MSU Remote Sensing & GIS Research and Outreach Services (RS&GIS). It is designed to provide access to parcel mapping and tax/assessment information for the public and county employees.

Interactive Map Viewer

[Start the Viewer](#)

Support

Questions about parcel data
Rosemary Anger ✉ RAnger2@ingham.org

Problems with this viewer
✉ ingham-equalization@rsgis.msu.edu

Browser Support

Google Chrome	Recommended
Mozilla Firefox	
Opera	

Application: EL Properties

Suppose we wanted to get information about the property at 549 Grove St. from the parcel map.

Doing this takes a couple steps:

1. Launch the [Map Viewer](#)
2. Click on "Search"
3. Type address components (number and street name) into the search bar, hit Go
4. Click on "Snap to Result" button to zoom in on the parcel
5. Click on "Toggle Table" to show information

Application: EL Properties

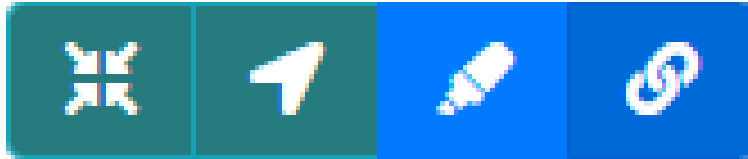
Unfortunately, this doesn't have the property details we are interested in.

The screenshot shows a web application interface for property management. On the left, there is a sidebar with 'Identify', 'Search', and 'Tools' buttons. Below these is a 'Layers' section. The main map area displays a grid of parcels with labels like '90', '015', '001', and '014'. A red rectangle highlights a specific parcel. Below the map, there is a table with property details.

Parcel Number	Owner	Owner Address	Property Address	School District	Acreage
33-20-02-18-113-001	GARDEN GROVE LLC	P O BOX 4874 EAST LANSING P O BOX 4874, 48826	549 GROVE EAST LANSING MI, 48823	33010	0.14

Application: EL Properties

Fortunately, there is a link to more information on a different site:



33-20-02-18-113-001

GARDEN GROVE LLC

549 GROVE

EAST LANSING MI, 48823

[View on bsaonline \[SECURE\]](#)

Application: EL Properties

Which *unfortunately* requires verification

[BS&A Online](#) [Home](#) [Services](#) [Municipalities](#) [Currently not signed in](#)

Ingham County | MI

powered by **BS&A** SOFTWARE

Search: [All Records](#) By: [Address](#) [Search](#)

☐ Use Advanced Address Search

SERVICES

[Public Records Search](#)

[All Record Search](#)

[Assessing Search](#)

[Current Tax Search](#)

[Delinquent Tax Search](#)

PERFORMANCE DATA

[MI Community Financial Dashboard](#)

[Performance Dashboard](#)

MUNICIPALITIES


[Select a Municipality](#)

ACCOUNT

[Sign In](#)


[Register](#)

[Why Register?](#)

**Security Verification**
For security purposes, click the checkbox below.

Verify Code

Check the box below to verify you are a human. This security feature helps prevent automated programs from accessing unauthorized information.

☐ I'm not a robot 

[Privacy](#) [Terms](#)

****Disclaimer:** BS&A Software provides BS&A Online as a way for municipalities to display information online and is not responsible for the content or accuracy of the data herein. This data is provided for reference only and WITHOUT WARRANTY of any kind, expressed or inferred. Please contact your local municipality if you believe there are errors in the data.

Application: EL Properties

Which *fortunately* has the characteristics we want once we get past this!

☐ Building Information - 1676 sq ft 1 3/4 STORY (Residential)

[Back To Top](#)

General

Floor Area	1,676 sq ft
Garage Area	216 sq ft
Foundation Size	1,024 sq ft
Year Built	1922
Occupancy	Single Family
Effective Age	52 yrs
Percent Complete	0%
AC w/Separate Ducts	No
Basement Rooms	3
1st Floor Rooms	7
2nd Floor Rooms	2
Bedrooms	7

Estimated TCV	\$193,781
Basement Area	1,006 sq ft
Year Remodeled	No Data to Display
Class	C
Tri-Level	No
Heat	Forced Air w/ Ducts
Wood Stove Add-on	No
Water	Public Water
Sewer	Public Sewer
Style	1 3/4 STORY

Area Detail - Basic Building Areas

Height	Foundation	Exterior	Area	Heated
1.75 Story	Basement	Siding	870 sq ft	1.75 Story
1 Story	Basement	Brick	136 sq ft	1 Story
1 Story	Crawl Space	Siding	18 sq ft	1 Story

Workflow

Let's now work through these steps in the context of an automated browser workflow:

1. Launch an automated browser session
2. Navigate to the Map Viewer
3. Search by street name + house number
4. Retrieve displayed info
5. Click detail link
6. Pass verification test
7. Scrape desired info

1. Launch Automated Session

First, we need to **initiate an automated browser session** with

```
selenider_session().2
```

```
session ← selenider_session(  
  "chromote", # set backend to chromote  
  timeout = 10 # set timeout to 10 sec  
)
```

We've now created a **local session**

- Accessed anywhere in a given script/RMarkdown file
- Closes automatically when the script/file finishes
- If started within a function, closes when the function is done running

² The full list of **selenider** functions can be found [here](#)

View the Session

If you want to **view the browser session**, we need to add the argument

```
options = chromote_options(headless = F)
```

Which will open the session in your Chrome browser

```
session ← selenider_session(  
  options = chromote_options(headless = F), # view the session  
  "chromote", # set backend to chromote  
  timeout = 10 # set timeout to 10 sec  
)
```


2. Navigate to Map Viewer

Let's start by navigating to the main [Tax Mapping Viewer page](#) and launch the viewer

- `open_url("URL")`

```
open_url("https://ingham-equalization.rsgis.msu.edu/")
```

2. Navigate to Map Viewer

Check that we ended up at the desired url:

```
current_url()
```

```
## [1] "https://ingham-equalization.rsgis.msu.edu/"
```

Global Functions

Other actions that apply globally to the entire page:

Function	Task
<code>open_url()</code>	navigate to the indicated url
<code>current_url()</code>	get the current url
<code>get_page_source()</code>	get the page's HTML
<code>back()</code> , <code>forward()</code>	navigate forward or back
<code>reload()</code> , <code>refresh()</code>	reload the current page
<code>take_screenshot()</code>	take screenshot of current image
<code>execute_js_fn()</code> , <code>execute_js_expr()</code>	execute a JavaScript function

2. Navigate to Map Viewer

Now we'll **click the "Start the Viewer" button** to launch the app.

Involved Steps:

- **Select** the button element by its CSS selector (`find_element()`)
- **Scroll to** the button (`elem_scroll_to()`)
- **Click** the button (`elem_click()`)

Selection Functions

Some additional selection functions:

Function	Task
<code>s()</code> , <code>ss()</code>	select HTML elements (without specifying the session)
<code>find_element()</code>	find a single HTML child element (yields a <code>selenium-element</code> object)
<code>find_elements()</code>	find multiple HTML child elements (yields a <code>selenium-elements</code> object)
<code>elem_filter()</code> , <code>elem_find()</code>	extract a subset of HTML elements

Element Properties

And functions for obtaining element properties

Function	Task
<code>elem_attr()</code> , <code>elem_attrs()</code> , <code>elem_value()</code>	Get attributes of an element
<code>elem_css_property()</code>	Get a CSS property of an element
<code>elem_name()</code>	Get the tag name of an element
<code>elem_size()</code> , <code>length(selenium- elements)</code>	get the number of elements in a collection
<code>elem_text()</code>	Get the text inside an element
<code>elem_equal()</code>	Test if two elements are equal

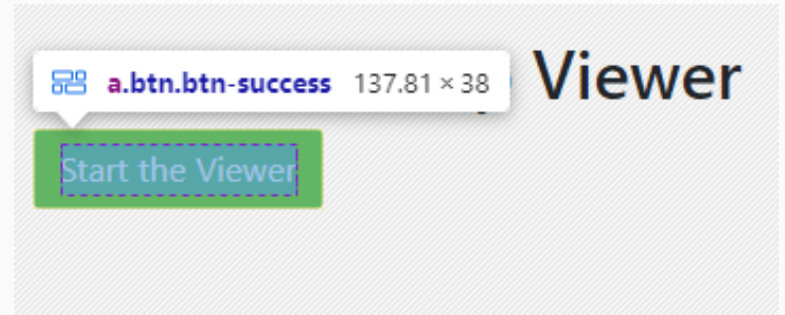
Actions

Function	Task
<code>elem_click()</code> , <code>elem_double_click()</code> , <code>elem_right_click()</code>	Click on an element
<code>elem_hover()</code> , <code>elem_focus</code>	Hover over an element
<code>elem_scroll_to()</code>	Scroll to an element
<code>elem_select()</code>	Select an HTML element
<code>elem_set_value()</code> , <code>elem__send_keys()</code> , <code>elem_clear_value()</code>	Set the value of an input
<code>elem_submit()</code>	Submit an element
<code>keys</code>	list of special keys

2. Navigate to Map Viewer

```
session %>%  
  find_element("a.btn.btn-success")  
  elem_text()
```

```
## [1] "Start the Viewer"
```



```
session %>%  
  find_element("a.btn.btn-success") %>%  
  elem_scroll_to() %>% # move the cursor to the button  
  elem_click() # click the button  
  current_url()
```

```
## [1] "https://ingham-equalization.rsgis.msu.edu/Viewer"
```


3. Search by Address

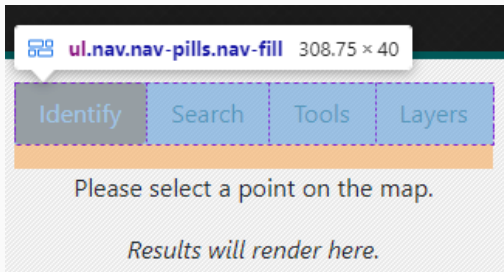
We next want to select + click on the "Search" tab button.

To do this, we'll take advantage of **relative position** selector functions.

Function	Task
<code>elem_parent()</code>	select the element that contains the current element
<code>elem_ancestors()</code>	select every element containing the current element (children, grandchildren, etc.
<code>elem_siblings()</code>	select every element which has the same parent as the current element (e.g. list)
<code>elem_children()</code>	select every element connected to and directly below the current element
<code>elem_descendants()</code>	select every element that is contained by the current element (any type of ancestor)

3. Search by Address

Select + click on the "Search" tab button.

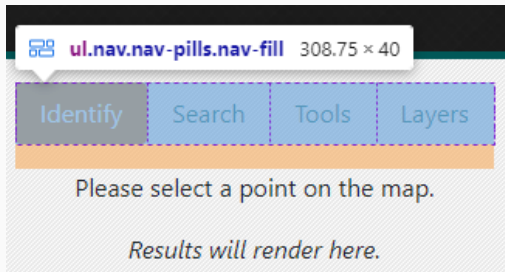


```
session %>%  
  find_element("nav.info-panel") %>% # get full  
  elem_children()
```

```
## { selenider_elements (3) }  
## [1] <ul role="navigation" class="nav nav-pills n  
## [2] <div class=""><div class="hide_container"><b  
## [3] <div class="hide_container" id="layers-tab">
```

3. Search by Address

Select + click on the "Search" tab button.



```
session %>%  
  find_element("ul.nav.nav-pills.nav-fill") %>  
  elem_children()
```

```
## { selenider_elements (4) }  
## [1] <li class="nav-item"><a href="#" class="nav-  
## [2] <li class="nav-item"><a href="#" class="nav-  
## [3] <li class="mobile--hidden nav-item"><a href=  
## [4] <li class="nav-item"><a href="#" class="nav-
```

3. Search by Address

We can also use **condition functions** to help with selection.

Function	Task
<code>has_text()</code> , <code>has_exact_text()</code>	has text that partially or fully matches
<code>has_attr()</code> , <code>attr_contains()</code> , <code>has_value()</code>	does the attribute match a value
<code>has_css_property()</code>	does an element's CSS property match a value (class, id, title, etc.)
<code>has_name()</code>	does an element have a tag name
<code>has_length()</code> , <code>has_size()</code> , <code>has_at_least()</code>	does a collection have a certain number of elements
<code>is_present()</code> , <code>is_in_dom()</code> , <code>is_absent()</code>	does an element exist
<code>is_visible()</code> , <code>is_displayed()</code> , <code>is_hidden()</code> , <code>is_invisible()</code>	is an element visible

3. Search by Address

We can also use **condition functions** to help with selection.

- `find_elements()` to yield a `selenium_element`
- `elem_find()` to extract a specific subset of HTML elements
 - `has_text()` to grab the element with the text "Search"
- `elem_click()` to click the selected button.

```
session %>%  
  find_element("ul.nav.nav-pills.nav-fill") %>%  
  find_elements("li.nav-item") %>%  
  elem_find(has_text("Search")) %>%  
  elem_click()
```

3. Search by Address

Checking to make sure we've activated the "Search" menu tab:

```
session %>%  
  find_element("ul.nav.nav-pills.nav-fill") %>%  
  elem_children()  
  
## { selenider_elements (4) }  
## [1] <li class="nav-item"><a href="#" class="nav-link">Identify</a></li>  
## [2] <li class="nav-item"><a href="#" class="nav-link active">Search</a></li>  
## [3] <li class="mobile--hidden nav-item"><a href="#" class="nav-link">Tools</a></li>  
## [4] <li class="nav-item"><a href="#" class="nav-link">Layers</a></li>
```

See the `nav-link active` class on the second element with the "Search" text?

3. Search by Address

Now **enter text** into the desired fields

```
# Find house number field
session %>%
  find_element("div.service-accord") %>%
  elem_children()
```

```
## { selenider_elements (2) }
## [1] <div class="__title"><h3>Search</h3><button
## [2] <div class="__content"><div class="service-i
```

3. Search by Address

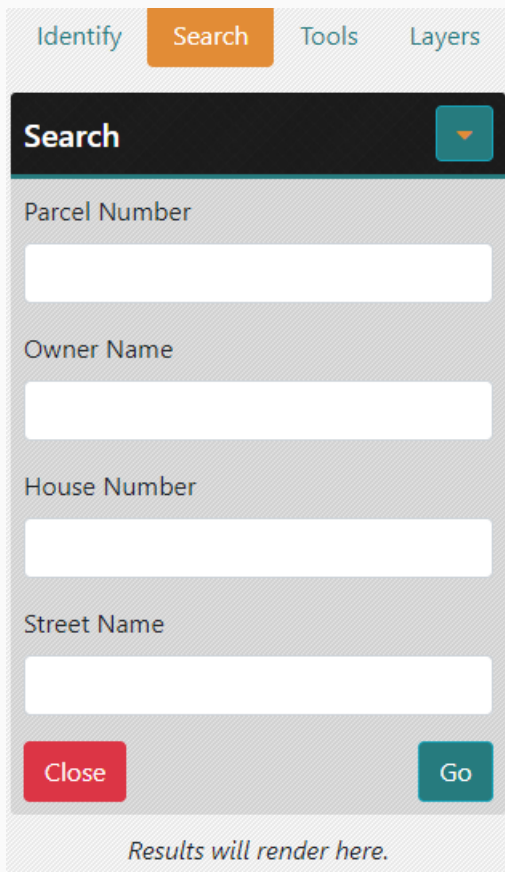
Now **enter text** into the desired fields

```
# Find house number field
session %>%
  find_element("div.__content") %>%
  elem_children()
```

```
## { selenider_elements (5) }
## [1] <div class="service-input form-group"><label
## [2] <div class="service-input form-group"><label
## [3] <div class="service-input form-group"><label
## [4] <div class="service-input form-group"><label
## [5] <div class="tab-controls mr-auto"><button cl
```


3. Search by Address

Now **enter text** into the desired fields and click "Go"



The screenshot shows a mobile application interface with a top navigation bar containing 'Identify', 'Search' (highlighted in orange), 'Tools', and 'Layers'. Below this is a 'Search' header with a dropdown arrow. The main area contains four input fields labeled 'Parcel Number', 'Owner Name', 'House Number', and 'Street Name'. At the bottom, there are two buttons: a red 'Close' button and a teal 'Go' button. A footer note states 'Results will render here.'

```
# set house number
ss("div.service-input.form-group") %>%
  elem_find(has_text("House")) %>%
  find_element("input") %>%
  elem_set_value("549")

# set street name
ss("div.service-input.form-group") %>%
  elem_find(has_text("Street")) %>%
  find_element("input") %>%
  elem_set_value("Grove")

# Click button
ss("button.btn.btn-info") %>%
  elem_find(has_text("Go")) %>%
  elem_scroll_to() %>%
  elem_click()
```

4. Retrieve Displayed Info



```
session %>%  
  find_element("div.result-item") %>%  
  read_html() %>%  
  html_text2() %>%  
  str_replace_all("\n\n", "\n") %>%  
  str_split("\n")
```

```
## [[1]]  
## [1] "33-20-02-18-113-001"      "garden grove llc"      "549 grove"  
## [4] "east lansing MI, 48823"
```

5. Follow Detail Link

```
s("a.btn.btn-primary") %>%  
  elem_attr("href") %>%  
  open_url()
```

```
current_url()
```

```
## [1] "https://bsaonline.com/SiteSearch/SiteSearchDetails?SearchFocus=Assessin"
```

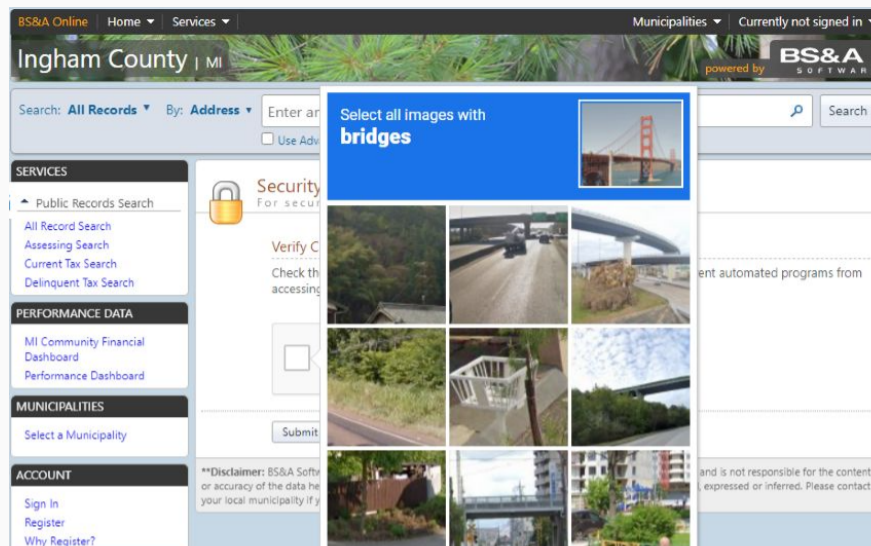
6. Pass Verification Test

Here's where things break down a bit.

We can get our browser to click the "Verify" button

```
s("#GetUserValidationInfo") %>%  
  elem_scroll_to() %>%  
  elem_click()
```

But then we get... the dreaded captcha



6-7. Pass Verification Test + Scrape

While I have yet to figure out a way around one of these image recognition captchas, here's a **semi-automated workaround**:

- Wait a set amount of time until an HTML element is present with `elem_expect()` and `is_present()`
 - i.e. element from the results page after the captcha
- Manually complete the captcha (must be viewing the session!)
- Proceed to scrape what's on the page

```
details ← ss("div.record-details-collapsible-box") %>%  
  elem_find(has_text("Building Information")) %>%  
  find_element("div.widthContainer table.detailTable") %>%  
  elem_expect(is_present, timeout = 60) %>% # wait up to 60 seconds to make sure  
  read_html() %>%  
  html_table() %>%  
  as.data.frame()
```

```
details
```

Scraping Dynamic/Interactive Websites

Methods for scraping dynamic or interactive websites opens up a huge range of possibilities.

- Iterative process of
 - Interact with the website and its elements
 - Extract element contents as with static sites
- Interactive workflows are generally more complex, so worth thinking about tradeoff between just brute forcing vs. coding up a scraping routine

Table of Contents

1. Prologue
2. Scraping Dynamic Websites