rvest tutorial demonstration

John Little

2021-05-17

Contents

Load library packages

Example

Import data

The following procedural example is based on documentation found at the `rvest` documentation site.

```
results <- read_html("http://www.vondel.humanities.uva.nl/ecartico/persons/index.php?subtask=browse")
```

Import a webpage using `read_html`

Results are in a list.

The `results` object is a *list* (R data type.) The items in the list correspond to the basic document structure of an HTML document…

Displaying the `results` object shows that the first item in the *list* is `head`. The second item is `body`. These items correspond to the basic structure of the HTML document type definition. In other words, the *text, links*, and HTML "stuff" were scraped from the web page. Specifically this stuff is found in the *body* element of the HTML document. This stuff is now stored in the `body` element of the `restults` *list*.

Contents of the `results` object

```
results
```

```
## {html_document}
## <html lang="en" prefix="foaf: http://xmlns.com/foaf/0.1/ owl: http://www.w3.org/2002/07/owl# schema:
## [1] <head about="#description" typeof="schema:CreativeWork schema:DataFeedIte ...
## [2] <body>\n\n<div id="projecttitle">\n\n<div id="projecttop">\n<a href="http ...
```

Example HTML

A simplified example HTML document

```
<HTML>
  <HEAD>
    <title>my example document</title>
  </HEAD>
  <BODY>
    <h1>Hello World</h1>
    <p>HTML is a tagging system known as the HypterText Markup Language</p>
  </BODY>
</HTML>
```

Procedure

The basic workflow of web scraping is

1. Development
    - Import raw HTML of a single target page (page detail, or "leaf")
    - *Parse* the HTML of the test page to gather the data you want
    - In a web browser, manually browse and understand the site navigation of the scrape target (site navigation, or "branches")
    - *Parse* the site navigation and develop an interation plan
    - Iterate: write code that implements iteration, i.e. automated page *crawling*
    - Perform a dry run with a limited subset of the target web site
    - check robots.txt, terms of use, and construct time pauses (to avoid DNS attacks)
2. Production
    - Iterate
        a. **Crawl** the site navigation (branches)
        b. **Parse** HTML for each detail page (leaves)

Parse the nodes of the HTML tree.

A web page is composed of HTML and prose. The web document, just as the web site, has a hierarchical structure. Web scraping means parsing these structures to gather needed information.

The first step is to start with a single target single document (i.e. a web page or a leaf of the site). In this case, the document we want to parse is the summary navigation page consisting of the first 50 names listed alphabetically in this web site. The goal is to parse the HTML source of that web page (i.e. document) by traversing the nodes of the document's HTML structure. In other words, we want to mine text and data from the `body` section of the `results` *list*. In this example I'll gather all the HTML within the `<li>` tags.

`li` stands for "list item". You can learn more about the *li* tag structure from HTML documentation.

Goal: Briefly...

- limit the results to the *list item* **nodes** of the `body` of the HTML document tree. This is done with the `html_nodes()` function.`html_nodes("li")`
- Use the `html_text()` function to parse the text of the HTML *list item* (i.e. the `<li>` tag)

For Example in an HTML document that has tagging such as this:

```
<li><a href="/ecartico/persons/17296">Anna Aaltse (1715 - 1738)</a></li>
```

I want to gather the text within the `<li>` tag: e.g. **Anna Aaltse (1715 - 1738)**

You can use the Selector Gadget to help you identify the HTML/CSS tags and codes.

CODE Using the `html_nodes()` and `html_text()` functions, I can retrieve all the text within `<li></li>` tags.

```
names <- results %>%
  html_nodes("#setwidth li a") %>%
  html_text()

names
```

```
##  [1] "Hillebrand Boudewynsz. van der Aa (1661 - 1717)"
##  [2] "Boudewijn Pietersz van der Aa (? - ?)"
##  [3] "Pieter Boudewijnsz. van der Aa (1659 - 1733)"
##  [4] "Boudewyn  van der Aa (1672 - ?)"
##  [5] "Machtelt  van der Aa (? - ?)"
##  [6] "Claas van der Aa I (? - ?)"
##  [7] "Claas van der Aa II (? - ?)"
##  [8] "Willem van der Aa (? - ?)"
##  [9] "Hans  von Aachen, alias:  Hans / Johann van Aken (1552 - 1615)"
## [10] "Jacobus van Aaken (? - ?)"
## [11] "Justus van Aaken (? - ?)"
## [12] "Johannes   Aalmis, alias:  Jan (1714 - 1799)"
## [13] "Johan Bartholomeus   Aalmis (1723 - 1786)"
## [14] "Maria  van Aalst (1639 - 1664)"
## [15] "Anna   Aalst (? - ?)"
## [16] "Anna   Aaltse (1715 - 1738)"
## [17] "Allart   Aaltsz (1665 - 1748)"
## [18] "Geertruy   Aaltsz (? - 1732)"
## [19] "Maria   Aaltsz (? - 1746)"
## [20] "Catharina   Aaltsz (? - 1727)"
## [21] "Nikolaas  van Aaltwijk (1692 - 1727)"
## [22] "Maria   Aams (1711 - 1774)"
## [23] "Jacobus   Aams (1680 - ?)"
## [24] "Jan Govertsz. van der Aar (1544 - 1612)"
## [25] "Anna  van der Aar (1576 - 1656)"
## [26] "Janneke Jans van Aarden (1609 - 1651)"
## [27] "Abraham  van Aardenberg (1672 - 1717)"
## [28] "Willem Aardenhout I (? - ?)"
## [29] "Margrietje   Aarlincx (1637 - 1690)"
## [30] "Dirck  van Aart (1680 - 1737)"
## [31] "Jonas   Abarbanel, alias:  Abravanel; Abrabanel (? - 1667)"
## [32] "Josephus   Abarbanel (? - ?)"
## [33] "Esther    Abarbanel (? - ?)"
## [34] "Rachel   Abarbanel (? - ?)"
## [35] "Lea   Abarbanel (1691 - ?)"
## [36] "Isaac   Abarbanel (1637 - 1723)"
## [37] "Damiana    Abarca (? - 1630)"
## [38] "Bartholomeus   Abba (1641 - 1684)"
## [39] "Cornelis  Dirksz.   Abba (1604 - 1675)"
## [40] "Clara   Abba (1631 - 1671)"
## [41] "Aerlant   Abbas (1606 - 1696)"
## [42] "Matheus Jansz  Abbas, alias:  Diercsz (1569 - ?)"
## [43] "Hendrik   Abbé, alias:  Enrico Abè (1639 - 1677)"
## [44] "Claude   Abbé, alias:  Glaude (? - 1653)"
## [45] "Simon Jan Pontenz.  Abbe (1467 - 1549)"
## [46] "Simon IJsbrandz.  Abbe (? - ?)"
## [47] "Ysbrandt Simonsz.  Abbe (? - 1559)"
## [48] "Maximiliaen  l' Abbé, alias:  Labbé (? - 1675)"
## [49] "Marten Simonsz.  Abbe genaamd Schuyt (? - 1592)"
## [50] "Daniël   Abbeloos (ca. 1635 - 1677)"
```

Parse the HTML attributes of an HTML tag...

Beyond the text you may also want attributes of HTML tags. To mine the URL of a hypertext link `<a href="URL"></a>`, within a list item, you need to parse the HREF argument of an anchor tag. If you're new to web scraping, you're going to need to learn something about HTML tags, such as the anchor tag.

For Example in an HTML document that has tagging such as this:

```
<a href="https://search.com">Example Link</a>
```

I want to gather the value of the `href` attribute within the anchor tag: **https://search.com**

CODE Using the `html_nodes()` and `html_attr()` functions, I can retrieve all the attribute values within `<li><a></a></li>` tags.

```
url <- results %>%
  html_nodes("#setwidth li a") %>%
  html_attr("href")

url
```

```
##  [1] "/ecartico/persons/414"   "/ecartico/persons/10566"
##  [3] "/ecartico/persons/10567" "/ecartico/persons/10568"
##  [5] "/ecartico/persons/27132" "/ecartico/persons/33780"
##  [7] "/ecartico/persons/33781" "/ecartico/persons/33782"
##  [9] "/ecartico/persons/9203"  "/ecartico/persons/33052"
## [11] "/ecartico/persons/33053" "/ecartico/persons/43671"
## [13] "/ecartico/persons/43672" "/ecartico/persons/30222"
## [15] "/ecartico/persons/38845" "/ecartico/persons/17296"
## [17] "/ecartico/persons/38518" "/ecartico/persons/38523"
## [19] "/ecartico/persons/38524" "/ecartico/persons/38525"
## [21] "/ecartico/persons/43337" "/ecartico/persons/42619"
## [23] "/ecartico/persons/42620" "/ecartico/persons/41311"
## [25] "/ecartico/persons/49902" "/ecartico/persons/20653"
## [27] "/ecartico/persons/47922" "/ecartico/persons/33783"
## [29] "/ecartico/persons/42051" "/ecartico/persons/28921"
## [31] "/ecartico/persons/37887" "/ecartico/persons/37890"
## [33] "/ecartico/persons/37892" "/ecartico/persons/38352"
## [35] "/ecartico/persons/42876" "/ecartico/persons/42881"
## [37] "/ecartico/persons/22859" "/ecartico/persons/52962"
## [39] "/ecartico/persons/52963" "/ecartico/persons/52965"
## [41] "/ecartico/persons/17297" "/ecartico/persons/55241"
## [43] "/ecartico/persons/416"   "/ecartico/persons/11593"
## [45] "/ecartico/persons/41739" "/ecartico/persons/41742"
## [47] "/ecartico/persons/41743" "/ecartico/persons/52649"
## [49] "/ecartico/persons/41738" "/ecartico/persons/417"
```

Note that the above links, or *hrefs*, are relative URL paths. I still need the domain name for the web server `http://www.vondel.humanities.uva.nl`.

Systematize targets

Above I created two vectors, one vector, `names`, is the `html_text` that I parsed from the `<li>` tags within the `<body>` of the HTML document. The other vector, `url`, is a vector of the values of the `href` attribute of the anchor `<a>` tags.

Placing both vectors into a tibble makes manipulation easier when using tidyverse techniques.

Goal

I want to develop a systematic workflow that builds a tibble consisting of each of the 50 names listed in the summary `results` object retrieved by `read_html()` performed on this page. Of course, mining and parsing the data is just the beginning. Data cleaning is a vital and constant aspect of web scraping.

Build Tibble

Using vectors from parsing functions above. . . .

```
results_df <- tibble(names, url)

results_df
```

```
## # A tibble: 50 x 2
##    names                                                   url
##    <chr>                                                   <chr>
##  1 Hillebrand Boudewynsz. van der Aa (1661 - 1717)         /ecartico/persons/4~
##  2 Boudewijn Pietersz van der Aa (? - ?)                   /ecartico/persons/1~
##  3 Pieter Boudewijnsz. van der Aa (1659 - 1733)            /ecartico/persons/1~
##  4 Boudewyn  van der Aa (1672 - ?)                         /ecartico/persons/1~
##  5 Machtelt  van der Aa (? - ?)                            /ecartico/persons/2~
##  6 Claas van der Aa I (? - ?)                              /ecartico/persons/3~
##  7 Claas van der Aa II (? - ?)                             /ecartico/persons/3~
##  8 Willem van der Aa (? - ?)                               /ecartico/persons/3~
##  9 Hans  von Aachen, alias:  Hans / Johann van Aken (1552 ~ /ecartico/persons/9~
## 10 Jacobus van Aaken (? - ?)                               /ecartico/persons/3~
## # ... with 40 more rows
```

From above we have links in a `url` vector, and target names in a `names` vector, for fifty names from the target website we want to crawl. Of course you also want to parse data for each person in the database. To do this we need to read (i.e. `read_html()`) the HTML for each relevant `url` in the results_df tibble. Below is an example of how to systematize the workflow. To do that, we'll make a results tibble, `results_df`. But first, more data cleaning. . .

Create some new variables with `mutate`. Build a **full** URL from the *relative URL path* (i.e. the `url` vector) and the domain or *base URL* of the target site. Since we scraped the relative URL **path**, we have to construct a **full** URL.

```
urls_to_crawl_df <- results_df %>%
  mutate(full_url = glue::glue("http://www.vondel.humanities.uva.nl{url}")) %>%
  # mutate(full_url = str_replace_all(full_url, "\\.\\.", "")) %>%
  select(full_url)

urls_to_crawl_df
```

```
## # A tibble: 50 x 1
##    full_url
##    <glue>
##  1 http://www.vondel.humanities.uva.nl/ecartico/persons/414
##  2 http://www.vondel.humanities.uva.nl/ecartico/persons/10566
##  3 http://www.vondel.humanities.uva.nl/ecartico/persons/10567
##  4 http://www.vondel.humanities.uva.nl/ecartico/persons/10568
##  5 http://www.vondel.humanities.uva.nl/ecartico/persons/27132
##  6 http://www.vondel.humanities.uva.nl/ecartico/persons/33780
##  7 http://www.vondel.humanities.uva.nl/ecartico/persons/33781
##  8 http://www.vondel.humanities.uva.nl/ecartico/persons/33782
##  9 http://www.vondel.humanities.uva.nl/ecartico/persons/9203
## 10 http://www.vondel.humanities.uva.nl/ecartico/persons/33052
## # ... with 40 more rows
```

As you can see, above, it's really helpful to know about Tidyverse text manipulation, specifically `mutate`, `glue`, and pattern matching and regex using the stringr package.

Operationalize the workflow

To operationalize this part of the workflow, you want to iterate over the vector `full_url` found in the `urls_to_crawl_df` tibble. Then `read_html` for each name that interest you. Remember that only 50 of the 54 rows in the `resutls_df` tibble are target names to crawl. So, really, you still have some data wrangling to do. How can you eliminate the four rows in the `results_df` tibble that are not targets (i.e. names)? Somewhere, below, I'll also show you how to exclude the four rows of unnecessary/unhelpful information.

Navigation and crawling

Meanwhile, we still have the **goal to systematically crawl the site's navigation links** for each of the 20+ summary results pages, each of which consists of fifty names. Remember, each summary results page also has links to web site **navigation**. Navigation links are the **key to crawling** through the other summary results pages. You will need to write code that crawls through the navigation links, then `read_html()` for each of the fifty name/urls in that particular summary results page.

Therefore, one crawling **goal** is to build up a tibble that contains URLS for each summary results page. So far we only have links to fifty names from the first summary results page. We don't have the URLs for each of the 20+ navigation pages. How do we get those? First find the pattern for the links to the navigation pages. Let's get the navigation link to the second summary results page.

```
http://www.vondel.humanities.uva.nl/ecartico/persons/index.php?subtask=browse&field=surname&strtchar=A&
```

After some investigation of the the HTML source, I see the only difference bewteen navigation URLS is the value of the URL argument `page=`. Eventually we need to construct a tibble with all the URLs for each of the navigation pages. But first, let's learn a bit more about HTML parsing.

Deconstructing a single target page and site design

So far we've used `rvest` functions and learned a little HTML. You **also need to know about CSS** (Cascading Style Sheets).

You can follow the link above for a definition of CSS. As a quickstart appraoch, I recommend playing this CSS game as a fun and quick way to learn just enough CSS. I completed the first 15 levels of the game; that was enough to get a good foundation with CSS. Depending on the complexity of the HTML and CSS, you may need to know a little more or a little less. But you need to know something about HTML and CSS.

CSS will help us subset the HTML for a single target page. I didn't mention it before but, you often need to view the source of the HTML; use the chrome browser to inspect elements of pages in a web browser; and use the chrome browser extension, selector gadget, to better understand the HTML and CSS tagging and structure.

This is key. When web scraping, you are effectively reverse engineering the web site. To reverse engineer it, you must have a solid understanding of the target site's structure, the site navigation, HTML, and CSS. Knowledge of the site structure is independent of R, `rvest`, *lists*, and `purrr`.

Anyway, an example of a CSS element in the results page is 'class' attribute of the `<div>` tag. In the case below we also have a class value of "subnav". Viewing the source HTML of one summary results page will show the `<div>` tags with the CSS *class* element.

For Example Use is `html_nodes()` with `html_text()`, and `html_attr()` to parse the anchor tag nodes, `<a>`, found within the `<div>` tags which contain the `class="subnav"` attribute.

```
<div class = "subnav">
    ...
    <a href="/ecartico/persons/index.php?subtask=browse&amp;field=surname&amp;strtchar=A&amp;page=3">[1(
    ...
</div>
```

CODE Parse the **text** of the navigation bar.

```
results %>% #html_nodes("div.subnav")
  html_nodes("form+ .subnav") %>%
  html_text()
```

```
## [1] "Results:  [1-50] [51-100] [101-150] [151-200] [201-250] [...]  [1101-1116]"
```

Parse the HTML *href* **attribute** to get the URL.

```
navigation <- results %>%
  html_nodes("form+ .subnav a") %>%
  html_attr("href")

navigation
```

```
## [1] "/ecartico/persons/index.php?subtask=browse&field=surname&strtchar=A&page=2"
## [2] "/ecartico/persons/index.php?subtask=browse&field=surname&strtchar=A&page=3"
## [3] "/ecartico/persons/index.php?subtask=browse&field=surname&strtchar=A&page=4"
## [4] "/ecartico/persons/index.php?subtask=browse&field=surname&strtchar=A&page=5"
## [5] "/ecartico/persons/index.php?subtask=browse&field=surname&strtchar=A&page=23"
```

Crawl summary results pages

We want to gain a clear sense of the predictable navigation structure of our target site, and the navigation URLs, so that we can **crawl** through each page of the target site. Again, the task before us is to do this crawling systematically. Let's put the `navigation` object into a tibble data-frame called `nav_df`.

```
nav_df <- tibble(navigation)
nav_df
```

```
## # A tibble: 5 x 1
##   navigation
##   <chr>
## 1 /ecartico/persons/index.php?subtask=browse&field=surname&strtchar=A&page=2
## 2 /ecartico/persons/index.php?subtask=browse&field=surname&strtchar=A&page=3
## 3 /ecartico/persons/index.php?subtask=browse&field=surname&strtchar=A&page=4
## 4 /ecartico/persons/index.php?subtask=browse&field=surname&strtchar=A&page=5
## 5 /ecartico/persons/index.php?subtask=browse&field=surname&strtchar=A&page=23
```

In the above tibble, I see I need to **clean** the `nav_df$navigation` vector so that only the unique URLs for each 50-result summary page remain. One problem is there are only 14 rows and a lot of redundancy and many missing summary page URLS. One reason I know this is that I have a link for page 2, 3, 4, 5, and 22; but nothing, no URLs for pages between pages 6 and 21. This mirrors what we see when we view the actual target web page in a web browser. That is, I have links for **results** [1-50], [51-100], [100-150], but nothing for [251-260] (i.e. page 6), etc. The target page is composed of HTML and CSS and comprises what we have in our imported `results` object, above. To confirm my investigation, I'll use a web browser to **view** the HTML **source** of one of the summary pages. The unparsed HTML is what you have in `results` object.

What I see in the view source or the `nav_df$navigation` vector is that each navigation URL is the same, **except** for the difference in the `page=` element found at the end of the URL argument.

Mini review summary

We can see that the navigation URL pattern is predictable. We can construct a full URL for each summary results target page. We can make a tibble with all these links, one row for each URL to each navigation section of each summary results page of the target site. Once we have this, we can write a script to systematically browse, or crawl, through the target site just as if we manually mouse-clicked each link in a web browser. We will use tidyverse packages to crawl through the site and `rvest::html_read()` to import the HTML. After we crawl each page, we'll parse the target HTML data.

Develop a plan

Goal make a tibble with all the navigation links

Let's start by importing the HTML of a single summary results page. Then I'll make a tibble of the navigation links found within the gathered and relevant `<div>` HTML tags, i.e. the div tags with the *subnav* CSS class. We did that with the `nav_df` tibble.

After we gather those navigation links in `nav_df$navigation`, we will wrangle and expand the vector to include only the relevant navigation URLs. But, why did we have duplicate navigation URLS in that vector? Look at the summary page in a web browser. See any duplication? If not, look closer.

Below, use `dplyr::distinct()` and `stringr::string_extract()` among other tidyverse functions to wrangle a tibble of relevant links.

```r
nav_df <- nav_df %>%
  filter(str_detect(navigation, "&page=")) %>%
  mutate(page_no = str_extract(navigation, "\\d+$")) %>%
  mutate(page_no = as.numeric(page_no))

nav_df
```

```
## # A tibble: 5 x 2
##   navigation                                                          page_no
##   <chr>                                                                 <dbl>
## 1 /ecartico/persons/index.php?subtask=browse&field=surname&strtchar=A&p~       2
## 2 /ecartico/persons/index.php?subtask=browse&field=surname&strtchar=A&p~       3
## 3 /ecartico/persons/index.php?subtask=browse&field=surname&strtchar=A&p~       4
## 4 /ecartico/persons/index.php?subtask=browse&field=surname&strtchar=A&p~       5
## 5 /ecartico/persons/index.php?subtask=browse&field=surname&strtchar=A&p~      23
```

There's some *fancy* regex pattern matching taking place in the data wrangling code chunk, above. Remember how I said you need to learn about pattern matching and the stringr library? Yup.

Anyway…

Now all we need to do is expand the sequence of pages. (Hint: `tidyr::expand()`)

The maximum number of summary pages in the `navigation$page_no` variable is 22. This should mean the maximum number of URLs to summary results pages will be roughly 22. Regardless of the total number of target names/pages, our task is to build a tibble with a URL for each summary results page, i.e. pages 1 thorough 22. IF we have a link to each sumamry results page, then can we get a link for each of the fifty people listed on each of the summary result pages.

Honestly, building this list of navigation URLs takes some effort in R, especially if you're new to R. So, maybe, there's an easier way. It might be easier to build the range of summary page URLs in Excel, then import the excel file (or CSV file) of URLs into R for crawling via `rvest`. But I want a reproducible, code-based approach.

See example code, below, for a reproducible example. With the next code chunk, you can use Tidyverse techniques and build a tibble of urls to iterate over. In this case, the important reproducible step use `stringr::str_extract()` to find and match the URL pattern.

```
nav_df <- nav_df %>%
  mutate(navigation = str_extract(navigation, ".*(?=\\=\\d+$)")) %>%
  mutate(page_no = as.integer(str_replace(page_no, "^2$", "1"))) %>%
  expand(navigation, page_no = full_seq(page_no, 1)) %>%
  transmute(url = glue::glue("http://www.vondel.humanities.uva.nl{navigation}={page_no}"))

# the regular expression '.*(?=\\=\\d+$)' can be read as:
# capture/find data found BEFORE '(?=<<pattern>>)' the pattern.
# The pattern is =<digit><digit>,
# the '=' sign has to be "escaped"  '\\='.
# The precise regex pattern syntax has the equal sign followed by a digit '\\d'.
# multiple digits  '+',  i.e. '\\d+'
# Since the digits are found at the end of the value of the variable, use an anchor
# To anchor the end of the regex pattern, use the '$' regex anchor notation.
# Remember, we are matching everything before the pattern.  Match with the wildcard: '.*'
# Honestly, there's no substitute for knowing a bit about regex when you're coding.
#
# See the "work with strings" cheatsheet.  https://rstudio.com/resources/cheatsheets/

nav_df
```

```
## # A tibble: 23 x 1
##    url
##    <glue>
##  1 http://www.vondel.humanities.uva.nl/ecartico/persons/index.php?subtask=brows~
##  2 http://www.vondel.humanities.uva.nl/ecartico/persons/index.php?subtask=brows~
##  3 http://www.vondel.humanities.uva.nl/ecartico/persons/index.php?subtask=brows~
##  4 http://www.vondel.humanities.uva.nl/ecartico/persons/index.php?subtask=brows~
##  5 http://www.vondel.humanities.uva.nl/ecartico/persons/index.php?subtask=brows~
##  6 http://www.vondel.humanities.uva.nl/ecartico/persons/index.php?subtask=brows~
##  7 http://www.vondel.humanities.uva.nl/ecartico/persons/index.php?subtask=brows~
##  8 http://www.vondel.humanities.uva.nl/ecartico/persons/index.php?subtask=brows~
##  9 http://www.vondel.humanities.uva.nl/ecartico/persons/index.php?subtask=brows~
## 10 http://www.vondel.humanities.uva.nl/ecartico/persons/index.php?subtask=brows~
## # ... with 13 more rows
```

Iterate

Use `purrr::map` **instead** of **'for'** loops. Because purrr is the R/Tidyverse way. 'For' loops are fine, but invest some time learning purrr and you'll be better off. Still, there's no wrong way to iterate as long as you get the right answer. So, do what works. Below is the Tidyverse/Purrr way. . . .

Now that I have a full list of navigation URLs, each of which represents a web page that has a summary of 50 names/links. My next task is to read the HTML of each URL representing a target-page – in this case a target is a detailed page with structured biographical information about an artist. By reading the URL (importing the HTML) for each target name, I will then have HTML for each individual target person. Of course, I still, then, have to read and parse the HTML of those target-name pages, but I can do that. The scraping (crawling + parsing) works when I have a URL per target person. Because, having a URL for each target-person's page means I can systematically scrape the web site. In other words, I can crawl the summary navigation to construct a full URL for each name (i.e page.) Then I import (i.e. `read_html()`) each person's page and parse the HTML for each person's information.

But, back to the current task: import the HTML for each summary results page of 50 records. . .

You should read the notes below, but tl;dr: skip to the CODE below

Notes on being a good scraper

Pause **Note**: that, below, I introduce a **pause** (`Sys.sleep()`) in front of each `read_html()` function. This is a common technique for well behaved web scraping. Pausing before each `read_html` function, avoids overwhelming my target's server/network infrastructure. If I overwhelm the target server, the server host-people may consider me a DNS attack. If they think I'm a DNS attacker, they might choose to block my computer from crawling their site. If that happens, I'm up a creek. I don't want that. I want my script to be a well behaved bot-crawler.

robots.txt Speaking of being a good and honorable scraper-citizen, did I browse the robots.txt page for the site? Did I check the site for a Terms of Service page? Did I look to see if there were any written prohibitions against web crawling, systematic downloading, copyright, or licensing restrictions? I did **and you should too**. As of this writing, there do not appear to be any restrictions for this site. You should perform these types of good-scraping hygiene steps for every site you want to scrape!

Development v production Note: Below, for **development** purposes, I limit my crawling to 3 results pages of fifty links each: `my_url_df$url[1:3]`. Be conservative during your code development to avoid appearing as a DNS attacker. Later, when you are ready to crawl your whole target site, you'll want to remove such limits (i.e. `[1:3]`.) But for now, do everyone a favor and try not to be over confident. Stay in the kiddie pool. Do your development work until you are sure you're not accidentally unleashing a malicious or poorly constructed web crawler.

Keep records of critical data Note: Below, I am keeping the original target URL variable, `summary_url`, for later reference. This way I will have a record of which parsed data results came from which URL web page.

Working with lists Note: Below, the final result is a tibble with a vector, `summary_url`, and an associated column of HTML results, each result is stored as a nested R *list*. That is, a column of data types that are all "*lists*", aka a "*list column*". Personally I find lists to be a pain. I prefer working with tibbles (aka *data frames*.). But *lists* appear often in R data wrangling, especially when scraping with `rvest`. The more you work with *lists*, the more you come to tolerate *lists* for the flexible data type that they are. Anyway, if I were to look at only the first row of results from the html_results column, `nav_results_list$html_results[1]`, I would find a *list* of the raw HTML from the first summary results page imported via `read_html()`.

CODE

> **tl;dr** This is testing. I have three URLs (`html_reults[1:3]`), one for each of the first three navigation summary pages. Each summary page will contain the raw HTML for 50 names. I will `read_html` each link, waiting 2 seconds between each `read_html`.

```
nav_results_list <- tibble(
  html_results = map(nav_df$url[1:3],
    ~ {
      #url[1:3] - limiting to the first three summary results pages (each page = 50 results)
      Sys.sleep(2)
      # DO THIS!  sleep 2 will pause 2 seconds between server requests to avoid being identified and po
      .x %>%
        read_html()
    }),
  summary_url = nav_df$url[1:3]
)

nav_results_list
```

map the read_html() function

```
## # A tibble: 3 x 2
##   html_results summary_url
##   <list>       <glue>
## 1 <xml_dcmn>   http://www.vondel.humanities.uva.nl/ecartico/persons/index.php?s~
## 2 <xml_dcmn>   http://www.vondel.humanities.uva.nl/ecartico/persons/index.php?s~
## 3 <xml_dcmn>   http://www.vondel.humanities.uva.nl/ecartico/persons/index.php?s~
```

Above, I have three rows of *lists*, each list is the read_html() results of a summary results page, i.e. each list has 50 URLs and text of my eventual targets.

- `nav_results_list$summary_url` is the URL for each summary page.
- `nav_results_list$html_results` is the `read_html()` results that I want to parse for the href attributes and the html_text

map parsing functions Right. Using purrr (`map()`), I can iterate over the html_results *lists*, parsing each *list* with the `html_attr()` and `html_text()` functions. It is convenient to keep this parsed data in a tibble as a list: one column for the URL targets ; one column for the html text (which will contain the names of the person for whom the target URL corresponds.) The results are nested lists within a tibble.

```
results_by_page <- tibble(summary_url = nav_results_list$summary_url,
                          url =
                            map(nav_results_list$html_results,
                                ~ .x %>%
                                  html_nodes("#setwidth li a") %>%
                                  html_attr("href")),
                          name =
                            map(nav_results_list$html_results,
                                ~ .x %>%
                                  html_nodes("#setwidth li a") %>%
                                  html_text()
                                )
                          )

results_by_page
```

```
## # A tibble: 3 x 3
##   summary_url                                                          url        name
##   <glue>                                                               <list>     <list>
## 1 http://www.vondel.humanities.uva.nl/ecartico/persons/index~ <chr [50~ <chr [5~
## 2 http://www.vondel.humanities.uva.nl/ecartico/persons/index~ <chr [50~ <chr [5~
## 3 http://www.vondel.humanities.uva.nl/ecartico/persons/index~ <chr [50~ <chr [5~
```

unnest When I unnest the nested *list*, I then have a single tibble with 150 URLs and 150 names, one row for each target name. (I also used `filter()` to do some more regex data cleanup, which I alluded to near the beginning of this document.)

```
results_by_page %>%
  unnest(cols = c(url, name))
```

```
## # A tibble: 150 x 3
##    summary_url                          url          name
##    <glue>                               <chr>        <chr>
##  1 http://www.vondel.humanities.uva.nl/~ /ecartico/p~ Hillebrand Boudewynsz. va~
##  2 http://www.vondel.humanities.uva.nl/~ /ecartico/p~ Boudewijn Pietersz van de~
##  3 http://www.vondel.humanities.uva.nl/~ /ecartico/p~ Pieter Boudewijnsz. van d~
##  4 http://www.vondel.humanities.uva.nl/~ /ecartico/p~ Boudewyn  van der Aa (167~
##  5 http://www.vondel.humanities.uva.nl/~ /ecartico/p~ Machtelt  van der Aa (? -~
##  6 http://www.vondel.humanities.uva.nl/~ /ecartico/p~ Claas van der Aa I (? - ?)
##  7 http://www.vondel.humanities.uva.nl/~ /ecartico/p~ Claas van der Aa II (? - ~
##  8 http://www.vondel.humanities.uva.nl/~ /ecartico/p~ Willem van der Aa (? - ?)
##  9 http://www.vondel.humanities.uva.nl/~ /ecartico/p~ Hans  von Aachen, alias: ~
## 10 http://www.vondel.humanities.uva.nl/~ /ecartico/p~ Jacobus van Aaken (? - ?)
## # ... with 140 more rows
```

Now, my `results_by_page` tibble consists of three column variables

- `summary_url`: the link to the Summary Results page which contains the name of each targets-person
- `url`: the relative URL for each target-person
- `name`: the name of each target-person

Now I can iterate over each row of my `results_by_page$url` vector to read_html for each target. Then I can parse the raw HTML for each target name page. When I follow the links for each name, I have the raw HTML of each person, in *lists*, ready to be parsed with the `html_nodes`, `html_text`, and `html_attr` functions.

Web scraping

Now you know how to *crawl* a website to get a URL for each name found at the source web site. (i.e. crawl the site's navigation.) The next goal is to `read_html()` to ingest and parse the HTML for each target.

> Web scraping = crawling + parsing

Below is an example of gathering and parsing information for one URL representing one person.

Goal

Ingest, i.e. `read_html()`, each target name, then parse the results of each to mine each target for specific information. In this case, I want the names of each person's children.

CODE The information gathered is information from the detailed names page about the children of one person in the target database.

Emanuel Adriaenssen has three children:

- Children
  - Alexander Adriaenssen, alias: Sander (1587 - 1661)

    - Vincent Adriaenssen I, alias: Manciola / Leckerbeetien (1595 - 1675)
    - Niclaes Adriaenssen, alias: Nicolaes Adriaenssen (1598 - ca. 1649)

```r
# http://www.vondel.humanities.uva.nl/ecartico/persons/10579
# schema:children

emanuel <-  read_html("http://www.vondel.humanities.uva.nl/ecartico/persons/10579")

children_name <- emanuel %>%
  html_nodes("ul~ h2+ ul li > a") %>%
  html_text()
children_name
```

```
## [1] "Alexander   Adriaenssen, alias:  Sander (1587 - 1661)"
## [2] "Vincent   Adriaenssen I, alias:  Manciola / Leckerbeetien (1595 - 1675)"
## [3] "Niclaes   Adriaenssen, alias:  Nicolaes Adriaenssen (1598 - ca. 1649)"
```

Iterate There now. I just scraped and parsed data for one target, one person in my list of target URLs. Now use purrr to iterate over each target URL in the list. **Do not forget to pause, `Sys.sleep(2)`,** between each iteration of the `read_html()` function.

Resources

- https://rvest.tidyverse.org
- https://purrr.tidyverse.org
- https://community.rstudio.com/t/loop-for-with-rvest-for-scraping/56133/4 (looping with RVEST)
- purrr / map :: https://jennybc.github.io/purrr-tutorial/ls01_map-name-position-shortcuts.html

---

John Little

Data Science Librarian
Center for Data & Visualization Sciences
Duke University Libraries

https://JohnLittle.info
https://Rfun.library.duke.edu
https://library.duke.edu/data