Session 16 Gathering Electronic Data 2

R for Social Data Science

Jeffrey Ziegler, PhD

Assistant Professor in Political Science & Data Science
Trinity College Dublin

Fall 2022

Road map for today

Last time:

- Online data sources
- Data collection
- Web technologies
- HTML fundamentals

This time:

- XML, XPath
- APIs

XML: Extensible Markup Language

- XML (E**x**tensible **M**arkup **L**anguage) is a more general form of markup language
- Allows sharing structured data of tree-like form
- Relative to HTML:
  - Tags are user-defined
  - End tags are always required
  - Stricter (no inconsistencies permitted)

```
<courses>
<course>
<title>R for Social Data Science</title>
<code>HC1o2</code>
<year>2022</year>
<term>Michaelmas</term>
<description>Course on computer programming in R.</description>
</course>
</courses>

```

Ex: Parsing XML tree

```r
xml_txt <- "<courses>
  <course>
    <title>R for Social Data Science</title>
      <code>HC102</code>
      <year>2022</year>
      <term>Michaelmas</term>
      <description>Course on computer programming in R.</description>
  </course>
</courses>"
xml <- xml2::read_xml(xml_txt)
str(xml)
```

```
List of 2
$ node:<externalptr>
$ doc :<externalptr>
- attr(*, "class")= chr [1:2] "xml_document" "xml_node"
```

Ex: Parsing XML tree

```r
xml_children <- xml2::xml_children(xml)
xml_children
```

```
{xml_nodeset (1)}
[1] <course>\n  <title>R for Social Data Science</title>\n
<code>HCI02</code>\n  <year>2022</year>\n  <term>Michaelmas</term>\n
<description>Course on computer programming in R.</description>\n
</course>
```

```r
xml2::xml_children(xml_children[1])
```

```
{xml_nodeset (5)}
[1] <title>R for Social Data Science</title>
[2] <code>HCI02</code>
[3] <year>2022</year>
[4] <term>Michaelmas</term>
[5] <description>Course on computer programming in R.</description>
```

```r
xml2::xml_text(xml_children(xml_children[1]))
```

```
[1] "R for Social Data Science"  "HCI02"  "2022" "Michaelmas"
"Course on computer programming in R."
```

Examples of XML

- RSS (**R**eally **S**imple **S**yndication) feeds
- SVG (**S**calable **V**ector **G**raphics) images
- Modern office documents (Microsoft Office ‘.docx‘, ‘.xlsx‘, ‘.pptx‘, OpenOffice/LibreOffice)

Parsing XML/HTML with XPath

- XPath (XML Path Language) is a language for selecting parts of XML/HTML tree
- Basic syntax:
  - ‘/‘ - select element at the root node (e.g. ‘/html/body‘)
  - ‘//‘ - select element at any depth (e.g. ‘//h1‘)
  - ‘//<tag>/*‘ - select all descendants of tag (e.g. ‘//body/*‘)
  - ‘//<tag>[@<attr>]‘ - select all elements that have given attribute (e.g. ‘//h1[@style]‘)
  - ‘//<tag>[@<attr>=’<value>’]‘ - select all elements, whose attribute has given value (e.g. ‘//h1[@style=’color:Red;’]‘)

Extra: XPath syntax

Parsing XML/HTML with XPath

```r
dail_html <- read_html("https://en.wikipedia.org/wiki/Members_of_the_1st_D%C3%A1il")
rvest::html_elements(dail_html, xpath = "//p")
```

```
{xml_nodeset (5)}
[1] <p class="mw-empty-elt">\n\n</p>
[2] <p>The members of the <a href="/wiki/First_D%C3%A1il" title="First Dáil">First Dáil</a>,
known as <a href="/wiki/Teachta_D%C3%A1la" title="Teachta Dála">Teachtaí Dála</a> (TDs), were
the 101<sup id="cite_ref-double_1-1" class=" ...
[3] <p>When the Sinn Féin executive met on 1 January 1919 to plan for the Dáil's inaugural
meeting, it considered appointing substitutes for the imprisoned Sinn Féin TDs who would be
unable to attend, but decided against this.<sup  ...
[4] <p>Four TDs represented two separate constituencies: Éamon de Valera, Arthur Griffith,
Eoin MacNeill and Liam Mellowes. Ordinarily, this would prompt them to choose one constituency
to represent, and to move a writ for a by-ele ...
[5] <p>The following Westminster by-elections to Irish seats were filled by Unionists who
sat at Westminster.\n</p>
```

```r
rvest::html_elements(dail_html, xpath = "//h1")
```

```
{xml_nodeset (1)}
[1] <h1 id="firstHeading" class="firstHeading mw-first-heading"><span class="mw-page-title-
main">Members of the 1st Dáil</span></h1>
```

Scraping webpage with XPath

```r
dail_tabs <- rvest::html_elements(dail_html, xpath = "//table")
```

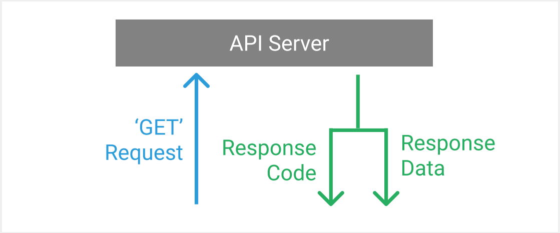
```
{xml_nodeset (8)}
[1] <table class="box-More_citations_needed plainlinks metadata ambox ambox-content
ambox-Refimprove" role="presentation"><tbody><tr>\n<td class="mbox-image">
<div class="mbox-image-div"><a href="/wiki/File:Question_book-new.svg" cl ...
[2] <table class="infobox vevent"><tbody>\n<tr><th colspan="2" class="infobox-above summary"
style="background-color: #ededed">1st Dáil</th></tr>\n<tr><td colspan="2"
class="infobox-subheader" style="border-top: 1px solid #aaa;"><t ...
[3] <table style="width:100%; border-collapse:collapse"><tbody><tr style="vertical-align:
middle;">\n<td style="width:100%;text-align:right;">
<a href="/wiki/Members_of_the_2nd_D%C3%A1il" title="Members of the 2nd Dáil">2nd Dáil</a> ...
[4] <table class="wikitable" style="font-size: 95%;"><tbody>\n
<tr style="background-color:#E9E9E9">\n<th colspan="2">Party</th>\n<th>Seats\n</th>\n</tr>\n
<tr>\n<td style="width: 2px; background-color: #326760;" data-sort-value="Sin ...
[5] <table class="wikitable" style="margin: 1em 1em 1em 0; background: #f9f9f9;
border: 1px #aaa solid; border-collapse: collapse;"><tbody>\n
<tr style="background-color:#E9E9E9;"><th colspan="4">Members of the 1st Dáil<sup id="cite ...
[6] <table class="wikitable"><tbody>\n<tr>\n<th>Constituency\n</th>\n<th>Outgoing TD\n</th>\n
<th colspan="2">Party\n</th>\n<th>Reason for vacancy\n</th>\n<th>Date of vacancy\n</th>\n
<th>Ref\n</th>\n</tr>\n<tr>\n<td>\n<a href="/wiki ...
[7] <table class="wikitable"><tbody>\n<tr>\n<th>Winner\n</th>\n<th colspan="2">Party\n</th>\n
<th>Constituency\n</th>\n<th>Date\n</th>\n<th>Outgoing\n</th>\n<th colspan="2">Party\n</th>\n
<th>Reason for vacancy\n</th>\n<th>Notes\n</t ...
[8] <table class="nowraplinks mw-collapsible autocollapse navbox-inner"
style="border-spacing:0;background:transparent;color:inherit"><tbody>\n<tr>
<th scope="col" class="navbox-title" colspan="2">\n<style data-mw-deduplicate="Templ ...
```

Scraping webpage with XPath

```r
tbody <- rvest::html_children(dail_tabs[5])
dial_members_1 <- as.data.frame(rvest::html_table(tbody))[,-3]
names(dial_members_1) <- dial_members_1[1,]
dial_members_1 <- dial_members_1[-1,]
```

```
Constituency                         Name          Party
2  Antrim East        Robert McCalmont Irish Unionist
3   Antrim Mid            Hugh O'Neill Irish Unionist
4 Antrim North       Peter Kerr-Smiley Irish Unionist
5 Antrim South    Charles Curtis Craig Irish Unionist
6   Armagh Mid James Rolston Lonsdale Irish Unionist
7 Armagh North           William Allen Irish Unionist
```

APIs



- 'Client' and 'server' interact with each other to request and provide data
  - Info can be transferred from one program to another, even if those programs are written in different languages
- Once computer receives a data request, does its own processing of data and sends it
- As requester, we need to write code in R that creates a request and tells API what we need
- That computer will then read our code, process our request, and return nicely-formatted data that can be easily parsed by existing R libraries

Why are APIs valuable?

- Contrast API approach to pure web scraping
  - When a programmer scrapes a web page, they receive data in a messy chunk of HTML
- While there are certainly libraries out there that make parsing HTML text easy, these are all cleaning steps that need to be taken before we even get our hands on data we want!

Examples of APIs: Wikipedia

```r
lapply(c("WikipediR"),  pkgTest)
dail_xml <- page_content("en", "wikipedia", page_name = "Members
     of the 1st Dail")
```

```
[]1] "<div class=\"mw-parser-output\"><div class=\"redirectMsg\">
<p>Redirect to:</p><ul class=\"redirectText\"><li>
<a href=\"/wiki/Members_of_the_1st_D%C3%A1il\"
title=\"Members of the 1st Dáil\">Members of the 1st Dáil</a></li>
</ul></div>
<div class=\"rcat rcat-R_to_diacritic\">\n<ul><li><b>
<a href=\"/wiki/Category:Redirects_from_titles_without_diacritics\"
```

Extra: List of APIs with no authorization needed

APIs in R

- ‘GET()‘ function from `httr` package helps you construct query URL and make API calls
- If done correctly, will give you a response object from API resource
  - Always read API documentation to identify method, URL endpoint, query parameters, and headers needed to call API
- R has functions which can easily convert the JSON or XML responses from API you are calling
  - ‘httr::content()‘ to retrieve contents
  - ‘jsonlite::fromJSON()‘ to convert JSON objects into R objects (such as a list)
  - ‘rlist‘ package to work with JSON response which was converted to a list

Web scraping in practice

- Always check first whether an API for querying exists
- It is the most robust (and sanctioned) way of obtaining data
- Check copyrights and respect those when using scraped data
- Limit you scraping bandwidth (introduce waiting times between queries, ‘Sys.sleep(x)‘)

Tutorial: Scraping web tables

- We’ll use same table of countries with their GDP from a Wikipedia last time
- This time, tidy up and extract the table using the wiki API and XML
- Tip: Use ‘WikipediR‘ and ‘page_content‘ function

Overview

This week:

- Online data sources
- Data collection
- Web technologies
- HTML fundamentals
- XML, XPath
- APIs