Web Scraping and APIs

http://datascience.tntlab.org

Module 11

Today's Agenda

- A deeper, hands-on look at APIs
  - A sneak-peak at server-side API code
- How to write API queries
- How to use R libraries to write queries for you
- How to manually scrape web pages in the easiest way possible

What's an API?

- **API: Application Programming Interface**
  - A data gateway into someone else's system, created by the owner of those data
  - Almost universally intended for real-time access by other websites, but you can take advantage of it too
  - Requires learning API documentation – they're all different
  - Takes advantage of representational state transfer (RESTful)

- Let's start easy. I've created a GET parameter-based REST API that adds two numbers, x & y.
  - http://scraping.tntlab.org/add.php
  - http://scraping.tntlab.org/add.php?x=1
  - http://scraping.tntlab.org/add.php?x=1&y=muffin
  - http://scraping.tntlab.org/add.php?x=1&y=8
  - http://scraping.tntlab.org/add.php?x=1&y=8&format=csv

- Important terminology: REST, GET vs. POST, queries, parameter/field, values

What's on the other side?

- This is PHP, a web scripting language.
- Can you follow it?

```php
<?
    if (!isset($_GET["x"]) || !isset($_GET["y"]))
        die("You didn't set x and y!");
    elseif(!is_numeric($_GET["x"]) || !is_numeric($_GET["y"]))
        die("x and y must be numbers!");
    else {
        $sum = $_GET["x"] + $_GET["y"];
        if(!isset($_GET["format"])) die("The sum of " . $_GET["x"] . " and " . $_GET["y"] . " is " . $sum . "!");
        elseif ($_GET["format"]=="csv") die($_GET["x"] . "," . $_GET["y"] . "," . $sum);
        elseif ($_GET["format"]=="tab") die($_GET["x"] . "\t" . $_GET["y"] . "\t" . $sum);
        else die("Format not recognized!");
    }
?>
```

Downloading Files (API or not)

- To download files available on the web:
  - Individual text data files as data frames, use **read_csv()**, **read_tsv()**, **read_delim()** (not their base-R equivalents)
  - Individual files or webpages that you want to save on your own computer, use **download.file()**

- To download files that require parameters (key/value pairs)
  - Webpages, but sending a GET request, either **download.file()** or *httr*'s **GET()**
  - Webpages, but sending a POST request, *httr*'s **POST()**

Converting a Known GET Request to R

- http://scraping.tntlab.org/add.php?x=1&y=8&format=csv

- **response <- GET("http://scraping.tntlab.org/add.php",**
  user_agent("ODU Researcher abcde001@odu.edu"),
  query = list(
  x = 1,
  y = 8,
  format = "csv"
)
)
- **response_content <- content(response)**

- What if I want to convert this single request into a series of requests? What stays the same, and what changes?

Writing Graceful Code (Error-handling)

- HTTP requests add an element of randomness to your program; when you create batch requests you don't know if your requests actually work or not
- If you're using an API R library, you shouldn't *usually* need to do this yourself, but there are exceptions
- You definitely want to look for HTTP access failures
- You might want to look for data retrieval failures too

Converting a Single R GET Request into a Series

- request_list <- tibble(x=1:10, y=11:20)
- responses <- tibble()

- ```
  get_response <- function(my_x, my_y) {
          print(paste("x is now", my_x, "and y is now", my_y))
          response <- GET("http://scraping.tntlab.org/add.php",
                          user_agent("ODU Researcher abcde001@odu.edu"),
                          query = list(
                                  x = my_x,
                                  y = my_y,
                                  format = "csv"
                          )
          )
          Sys.sleep(2)
          if(http_error(response)) return(as_tibble("HTTP Error"))
          else return(as_tibble(content(response, as="text")))
  }
  ```

- for (row in 1:nrow(request_list)) responses <- bind_rows(responses, get_response(request_list[row,"x"], request_list[row,"y"]))

- What's an alternative approach?
  ```

API Output Formats

- The output of an API can be in essentially *any* format, but XML (**eXtensible Markup Language**) and JSON (**JavaScript object notation**) are the most common

- XML should look familiar, because modern HTML is a specific type of XML

- These APIs also have **rate limits** in terms of the number of requests you are allowed to send and how quickly
  - Twitter limits to 180 calls every 15 minutes for simple requests and 15 calls every 15 minutes for complex ones. For example, only 25 tweets can be returned per simple call, so up to 4500 tweets per 15 minutes

Parsing Output

- *httr*'s **content()** automatically parses whatever it gets
  - Converts it into a class appropriate to its guess based upon the website (which you can check using **http_type()**)
  - You may need to pre-specify an output format

- If you want to look at raw JSON, you might consider the JSON Viewer extension for Chrome

- Once you have the raw XML or JSON, you can use several different functions/libraries to pull it apart and convert it into a data frame
  - *jsonlite:* **fromJSON()**
  - *xml2:* **read_xml(), xml_structure(), xml_find_all(), xml_double(), as_list()**
  - *rlist:* **list.select(), list.stack()**
  - *dplyr:* **bind_rows()**

Example Twitter JSON Output

Secure | https://graph.facebook.com/853552931365745/feed?access_token=EAACEdEose0cBANJClsJ9dadoE


```
{
   "data": [
      {
         "message": "#New_Significance #P_Less_Than_005\n#Type_I_Error\n#Type_II_Error\n#Error_Balance \nI dic
average effect size in social psychology) and computed sample sizes for different type-I and type-II error prc
alpha = .05,  beta = .75  Ratio 1/15\nN = 100, alpha = .05, beta = .50, Ratio 1/10\nN = 200, alpha = .05, beta
338, alpha = .005, beta = .20, Ratio 1/40\nN = 500, alpha = .005, beta = .05, Ratio 1/10\nN = 600, alpha = .00
power, which implies 20\u0025 Type-II errors, we fail to provide evidence for a true hypothesis with effect si
far, social psychologists have been using sample sizes of n = 20 per cell (N = 40 total) to chase these effect
75\u0025 and a type-I / type-II error ratio of 1/15.  \nIf social psychologists would do a priori power analys
times as many participants). \nUsing the same N = 200 and the new significance criterion of p \u003C .005,  pc
suggesting that type-II errors are much less important than type-I errors.  \nTo get back to a 1/4 ratio, samp
applies to d = .4, which is an average effect size, meaning power is lower for half of the studies.  \nAre we
         "story": "Uli Schimmack created a poll in Psychological Methods Discussion Group.",
         "updated_time": "2017-07-27T19:56:44+0000",
         "id": "853552931365745_1457448990976133"
      },
      {
         "message": "More comments on #new_significance \n\nIs it better to have no significance (threshold)?
         "story": "Uli Schimmack shared a link to the group: Psychological Methods Discussion Group.",
         "updated_time": "2017-07-27T19:41:59+0000",
         "id": "853552931365745_1458680730852959"
      },
      {
         "message": "Hi everybody,\n\nI\u2019m considering using p-curve and/or p uniform as supplementary pub
dependencies in the data, so to investigate other publication bias indices (trim-and-fill, PET-PEESE, selectio
package). \n\nDoes p-curve and p uniform in meta-analysis also assume \nthat all effect size estimates are inc
sizes, b) impute a p-value from the aggregated dependent effect size, and c) perform p-curve and/or p uniform
I\u2019ve read several papers on these methods, but so far have not seen any discussion on this issue.",
         "updated_time": "2017-07-27T19:14:04+0000",
         "id": "853552931365745_1458154720905560"
```

Experiment with the Twitter API

- Go to https://developer.twitter.com/en/apps (you'll need to be logged into Twitter and sign up for a "Developer" account)

- Create an app
  - This app will have the same permissions that your Twitter account has
  - Enter honest details during the signup process (Description = "Using APIs for Research at University of ##")

- Grab your access credentials and connect to the Twitter API in R using them
  - **library(twitteR)**
  - **api <- '######'**
  - **apiSecret <- '######'**
  - **access <- '######'**
  - **accessSecret <- '######'**

Use *twitteR* to Automate API Access

- setup_twitter_oauth(api, apiSecret, access, accessSecret)
- # select "1" when prompted to use local file for caching

- tweets <- searchTwitter("#rstats", 50)
- tweets_clean <- strip_retweets(tweets)
- tweets_tbl <- twListToDF(tweets_clean)
- View(tweets_tbl)

Understanding API Returns from Arbitrary Libraries

- *Read the documentation for the API service.*
- *Read the documentation for the library.*
- Check if some sort of access token or other authentication scheme is required and figure out how to get one.
- Use **glimpse()** and R Studio's auto-suggest function to figure out where what you want is located.
- Determine what format the library calls return.
- Figure out how to convert that format into a data frame.

R Libraries to Access APIs

- Wikimedia pageview data: *pageviews*
- Wordnik: *birdnik*

- Facebook: *Rfacebook*
- Twitter: *twitteR*

- Qualtrics: *qualtRics*
- Scopus: *rscopus, bibliometrix*
- Web of Science: *bibliometrix*
- data.world: *data.world*
- OSF: *osfr* (in development)

- SWAPI: *rwars*
- Google Maps: *ggmap*

What If There is No API?

1. Download the file(s) you want
2. Use one of the following approaches
    1. Use raw XML/HTML extraction with *xml2* and/or *rvest*
        - **html_table()** – Convert HTML <table> into a df
        - **html_text()** – Extract raw text from an HTML node(s)
        - Each function takes either an XPATH or a CSS selector
            - https://www.w3schools.com/xml/xpath_intro.asp
            - https://www.w3schools.com/cssref/css_selectors.asp
    2. Or, regular expression matching with *stringr*
        - However, if your data are organized by XML nodes, you should use *xml2*/*rvest*
        - This is a brute-force approach with some downsides
3. Convert the raw output into data frames

HTML + CSS Redux

```
<html>
<head>
<title>My Webpage</title>
<link href="mycss.css" type="text/css" rel="stylesheet">
</head>
<body>
<h1>My Wepage</h1>
<p class="mainparas" id="firstpara">This is my webpage.</p>
<p class="mainparas">Really, it is.</p>
<p>Not good at this.</p>
</body>
</html>
```

```
body {
        background-color: #EBEBEB;
}
h1 {
        font-weight: bold;
        font-size: 24pt;
}
p {
        font-style: italic;
}
#firstpara { font-weight: normal; }
p.mainparas { font-weight: bold; }
```

Extracting Text from a Single Webpage

- Use web browser tools to help identify patterns in HTML structure
  - SelectorGadget extension for Chrome (for CSS selectors)
  - Chrome's Developer Tools (F12) (if CSS selectors don't work)
  - Remember # is used for ids, and . for classes

- Try out various patterns until you get what you want

- Example
  - **apa_html <- read_html("http://www.apa.org/news/apa")**
  - **apa_nodes <- html_node(apa_html, "*CSS selector here*")**
  - **apa_text <- html_text(apa_nodes)**

Concerns When Web Scraping

- Same time limitations as before (remember **Sys.sleep()**); don't look like a hacker and you won't be treated like one
- Only download each webpage once. It's better to download each file to a folder and then process them one at a time.
- If you need to do dynamic link discovery, you may be better off moving to Python (see the Scrapy platform).
- Data wrangling/munging is universally more time-consuming than the analysis that comes afterward.
- If you want to use this for a research paper, theoretical justification is key. See http://psycnet.apa.org/record/2016-25479-001