WebScraping 1

HS

Introduction

- What is Web scraping? Web scraping or Web harvesting is the process of extracting data from Websites. It involves using `Softaware Tools` and `Programming Languages` to automatically collect and analyze data from web pages.

It is important to note that web scraping can be legally and ethically complex issue, an it may involves collecting data from websites without their permission or in violation of their terms of service. Therefore, it is important to understand the legal and ethical considerations involved in web scraping before using it for any purposes.

Data on the Web

- There are at least 4 ways people download data on the web:
    1. Click to download a csv/xls/txt file.
    2. Use a package that interacts with an API.
    3. Use an API directly.
    4. Scrape from directly from the HTML file.

API

- Organization use a software to act as a gateway to control the access to data. So, they use APIs.

What does API stand for?

- API stands for Application Programming Interface. APIs are like bridges that allow different software systems to communicate and share data or features with each other

What is a common form of API?

- A common form of API Known as REpresentational Slate Transfer or `REST API`.
- It is used to enable communication between different software applications by providing a standardized way to access and exchange data over the internet,

API

What does API do?

- It can control
  1. Who can access
  2. What can be accesed
  3. How can it be accesed
  4. When it can be accessed
  5. What is returned

What is API `ENDPOINT`?

- It is a location of resources to be accessed by an API.(Think of it as a URL)
- API may act as `GATEWAYS` to many different `ENDPOINTS` which can be turned `open` or `closed`.

DESCRIPTION

What is API `DESCRIPTION` document?

- Organization typically published an API description document to provide users information on how to interact with API.
- **Question:** How to find the API documentation for a website?
  - Easiest way is to use google: Use a query like: “API documentation site:example.com” (replacing example.com with the actual website domain).

API's Format

What are the most two formats that API will return the data?

These days, most API will return data in form of

1. XML or eXtensible Markup Language

- For storing, transforming, and reconstructing arbitrary data.
- Arbitrary data are such as, `text`, `image`, `audio`, `video`, etc.

2. JSON or Java Script Object Notation

- It is relatively easier/simpler than other format (or a lightweight data interchanging format)
- It is easier for human to read and write and easier for machine to parse(process) and generate.

```
{
  "name": "John",
  "age": 30
}
```

Authentication

- Basic authentication
  - Some API's use what is called "Basic authentication", where you provide a username and password.
  - The basic syntax for this is `Get("URL", authentication("username","password"))`
- API keys and PAT
  - More secure APIs now require you to obtain API key or Personal Access Tokens or PAT.
  - You may also have to register as developer
  - Once you get a key you are responsible for all behavior associated with your key.
  - **Never save or display your key or PAT in a file you might share**

Getting API Key

Geting API for Open Movie Database (OMDb)

- **Question** I do not have the API key, how may I get one?
  - Go to the OMDb website
  - On the banner on top of the page click **API Key**
  - Select *FREE*, then enter your *Email*, *Name*, and *Use*
  - The key will be emailed to you within one hour. Please do not lose it, as you will be unable to retrieve it unless you open a new account. One way to find the key is to search for "OMDB" in your email inbox. If you haven't temporarily deleted the email, you should be able to locate the message that contains the code.

Storing API Key

There are two main ways to store your key so you can access it in R.

1. Storing in `.Renviron` file is wasy and does not depend on other libraries to access the key but it leaves your key **exposed** in a plain text file.
2. Using the {`keyring`} package. It keeps your key password protected at the same level as your computer.

Storing API

Storing API's Key for Open Movie Database (OMDb)

Store Access Key

1. Using `.Renviron`.

- Make sure {`usethis`} package is installed
- Use **console** to open `.Renviron` file
  `usethis::edit_r_environ()`
- The `.Renviron` file will open. Eneter the following into `.Renviron` file `OMDB_API_KEY = <cod...>`

`.Renviron` Uses {usethis} Package

- Restart R (restart R often during development)
  - `Session` > Restart R
  - CTRL + SHFT + fn + F10 or CTRL + SHFT + F10
- You can always access your private using the function `Sys.getenv(<key name>)`
  The key name for this API is “OMDB_API_KEY”.
  - First you need to set it.
    - Set your API key in your environment file or directly in your session:
      `Sys.setenv(OMDB_API_KEY = "your_api_key_here")`
    - Retrieve the API key with Sys.getenv():
      `Sys.getenv("OMDB_API_KEY")`

{`keyring`} Package

2. Using the {`keyring`} package to store and access PAT

- Install and load {`keyring`} package
- In *console* type
- `key_set("key_name")`
  - The key name is "OMDB_API_KEY_SECURE"
  - Type in console then
  - `key_set("OMDB_API_KEY_SECURE")`
  - To get the key type
  - `key_get("OMDB_API_KEY_SECURE")`

A person with access to oyour computer who knows R and the {`keyring`} package could still get your key. But it is more secure than placing your key in plain text file like `.Renviron`

Using Your Key in Your Code

- You could access your key and save it to a variable for example
  - `my_key <- key_get("OMDB_API_KEY_SECURE")` However, that exposes your key in the global environment and `.Rhistory`.
  - `.Rhistory`: It is a history of code executed not written
- A better approach is to use `key_get("OMDB_API_KEY_SECURE")` or `Sys.getenv("OMDB_API_KEY")` inside of your API function to access your credential without creating an intermediate variable.
  - `x <- GET(url = "URL", apikey = key_get("key-name"))`

Directly Access APIs

Using {`httr`} package to directly access APIs

- Most website APIs use HTTP or Hyper Text Transfet Protocol. It is a language for querying and obtaining data. We will not learn HTTP here, but we will use R's {`httr`} package to inteface with APIs through HTTP.
- **Note: Every API is different, so you always have to figure out how to interact with a new API**
- API should have documentation, so you can read to understand key info:
  - What parameters you can send For instance you may go to OMDb website and on top of the page click on *Parameters*
  - How do you need to send them
  - What data you should get back based on the values of parameters.

Install and Load {`httr`} Package

- **Question:** Is {`httr`} package installed?
- Type
  - install.packages()‘ in console. It will return all install packages
  - `require(httr)`
    - If package is not installed it tells you
    - If package is installed it tells you that you need to load it
  - `installed.packages()` returns detailed data about installed packages
  - `find.package("pkg-name")` returns path to installed package or return an error
  - There are a lot more, such as `pacman()`, ...

{`httr`} Package

- install and load {`httr`} package
- The six most common `httr` functions are
  - `GET()`: Fetches data from a URL
  - `PATH()`
  - `POST()`
  - `HEAD()`
  - `PUT()`
  - `DELETE()`
- **There are two important parts to HTTP,**
  - The *REQUEST* the data sent to the *SERVER*
  - The *RESPONSE* the data sent back from the *SERVER*

GET()

- Fetches data from a URL
- ‘GET(url = “URL”, query = list(, ,)) - `URL`: is the URL where API request should be sent - `query()`: to modify the parameters of your request
- Type the following code `GET("https://httpbin.org/get")`
- It returns useful information such as
  - Response [https://httpbin.org/get]
  - Date: 2024-11-04 00:46
  - Status: 200
  - Content-Type: application/json
  - Size: 367 B AND MORE
- Status: 200 means the request has succeeded

Example

- we’re searching for the movie “Inception” by title from OMDb website

```
# Load the httr package
library(httr)
library(keyring)
library(usethis)
```

```r
URL <- "http://www.omdbapi.com/"

params <- list(apikey = key_get("OMDB_API_KEY_SECURE"),
              t = "Inception"
              )

# Make the GET request to the OMDb API
response <- GET(url = URL,
                query = params
                )
```

```r
if (status_code(response) == 200) {
  # Parse the JSON response into an R list
  movie_data <- content(response, as = "parsed")
    # Display some information about the movie
  print(paste("Title:", movie_data$Title))
  print(paste("Year:", movie_data$Year))
  print(paste("Director:", movie_data$Director))
  print(paste("Plot:", movie_data$Plot))
} else {
  # Print an error message if the request failed
  print("Error: Could not retrieve data.")
}
```

```
[1] "Title: Inception"
[1] "Year: 2010"
[1] "Director: Christopher Nolan"
[1] "Plot: A thief who steals corporate secrets through the
```

Decennial Census and American Community Survey (ACS)

The Decennial Census:

- It is a complete population count that happens **every 10 years** in the U.S. It gathers basic information, like age, race, and housing, from every household to help plan for things like schools, roads, and public services.

The American Community Survey (ACS):

- It is an ongoing survey that collects more detailed data on topics like education, income, and employment. Instead of every 10 years, it samples a smaller group of people **every year** to give communities updated information between censuses.

Register for a Census Data API Key:

- You need a Census API key to access U.S. Census data through the {`tidycensus`} package. The Census Bureau provides this key for free, and it's a one-time setup.
- You can request key by visiting the Request a U.S. Census Data API Key
- Enter your email address, and they'll send you a key.
- Once you've registered, they'll email you a key. You can then load it in R using the following command:
  ```
  census_api_key("your_census_api_key", install =
  TRUE)
  ```

The {`tidycensus`} Package

- {`tidycensus`} Package allows user to interface with the US Census Bureau
- **install and load the package**
- What is API `ENDPOINT`?
  - It is a fancy word for URL of a server or service.
  - APIs operate through request and responses. When an API requests to access data from a server, a response is always sent back.
  - The location where the API sends a request and where the response emanates (originated from, be produced by) is known as *endpoints*.

Searching Census Bureau Variable ID

- There are thousands of variable ID across the different Census Bureau files and their IDs, are not consistent.
- Getting variable from *decennial census* or *ACS* requires knowing the variable ID for specific product (year(s))
- Use function `load_variables()` function. Two requirements
  - The *year of the census* or the *end year of ACS* sample
  - The data set: “sf1”, “sf3”, “acs”, or “acs1”
- For ideal functionality, assign output to variable, setting `cache = TRUE`, to store result in your computer for future access. Then you can view the function in RStudio to interactively browse for variable or use `filter(str_detect())` if you know part of the name, or concept.
- load_variables(): Finding the right variables to use with get_decennial() or get_acs() can be challenging; load_variables() attempts to make this easier for you.

`load_variables()`

- Type `var17 <- load_variables(2017, "acs5", cache = TRUE)` to load variables for the 2017 ACS 5-Year Estimates.
- Preview the variables head(var17)
- You can search for variables by partial name or a concept. Here, we'll filter variables that contain "income."
  `income_vars <- var17 %>% filter(str_detect(label, regex("income", ignore_case = TRUE)))`
- View the income-related variables `print(income_vars)`
- `get_acs()` to pull the actual data. For example, let's pull median household income by county.

```
income_data <- get_acs(geography = "county",variables
= "B19013_001",year = 2017)
```

Example

```r
library(tidyverse)
library(tidycensus)
var17 <- load_variables(2017, "acs5", cache = TRUE)
head(var17)
```

```
# A tibble: 6 x 4
  name        label                                  concept
  <chr>       <chr>                                  <chr>
1 B00001_001  Estimate!!Total                        UNWEIGHT
2 B00002_001  Estimate!!Total                        UNWEIGHT
3 B01001A_001 Estimate!!Total                        SEX BY A
4 B01001A_002 Estimate!!Total!!Male                  SEX BY A
5 B01001A_003 Estimate!!Total!!Male!!Under 5 years SEX BY A
6 B01001A_004 Estimate!!Total!!Male!!5 to 9 years  SEX BY A
```

```r
# Filter variables that contain the word "income"
income_vars <- var17 %>%
  filter(str_detect(label, regex("income", ignore_case = TI

# View the income-related variables
head(income_vars)
```

```
# A tibble: 6 x 4
  name         label
  <chr>        <chr>
1 B06010PR_002 Estimate!!Total!!No income
2 B06010PR_003 Estimate!!Total!!With income
3 B06010PR_004 Estimate!!Total!!With income!!$1 to $9,999 c
4 B06010PR_005 Estimate!!Total!!With income!!$10,000 to $14
5 B06010PR_006 Estimate!!Total!!With income!!$15,000 to $24
6 B06010PR_007 Estimate!!Total!!With income!!$25,000 to $34
```

```r
# Get median household income data for all U.S. counties
income_data <- get_acs(geography = "county",
                       variables = "B19013_001", # Median H
                       year = 2017)
# View the data
head(income_data)
```

```
# A tibble: 6 x 5
  GEOID NAME                     variable   estimate   moe
  <chr> <chr>                    <chr>         <dbl> <dbl>
1 01001 Autauga County, Alabama B19013_001    55317  2838
2 01003 Baldwin County, Alabama B19013_001    52562  1348
3 01005 Barbour County, Alabama B19013_001    33368  2551
4 01007 Bibb County, Alabama    B19013_001    43404  3431
5 01009 Blount County, Alabama  B19013_001    47412  2630
6 01011 Bullock County, Alabama B19013_001    29655  5376
```

Using JSON Structured DATA

- Java Script Object Notation
- JSON uses conventions familia to programmers for many languages so it is an ideal data interchange language. for example

```
{
  "name": "John",
  "age": 30
}
```

- JSON uses a simple format with keys and values in curly braces {}.
- In programming:
  - Key: The name or label (e.g., “name”).
  - Value: The information or data assigned to that key (e.g., “John”).
  - A key and value separated by a colon :
    - Example: `"age": 30`

JSON-R Package

There are multiple packages working with JSON

- {`jsonlite`} package is a JSON parser/generator optimized for the web. It has bidirectional mapping between JSON data and the most important R data types.
  - Simplification is process by which JSON arrays are automatically converted from a list into more specific R class.
  - The fromJSON() function has 3 arguments which control the simplification process:
    - `simplifyVector`
    - `simplfyDtatFrame`
    - `simplifyMatrix`
- The {`tidyjson`} package takes an alternate approach to structuring *JSON data* into *tidy data*