Quantitative Research Methods IV - 17.806

Recitation, Week 1.

Topic: Data Collection.

Benjamín Muñoz

February 16, 2023

MIT

Massachusetts Institute of Technology

Table of contents

1/ Introduction

Presentation

- **TA: Benjamín Muñoz**
  - 3rd Year PhD Candidate.
  - Fields: Comparative Politics + Methods.
- **Quant IV:**
  - Advanced topics.
  - Focus on intuition and practical applications.
  - Hopefully a useful and fun experience!

Logistics

- **Recitation:** Fridays, 10-11 AM (ET), recorded upon request.
    1. Fill in gaps.
    2. Math review.
    3. Offer pset hints.
    4. Answer general questions about the course content.
- **Office Hours:** Tuesdays 15:30-17:00 PM or by appointment (ET).
    1. Answer general pset questions.
    2. Answer specific pset questions.
    3. Offer advice on your Quant IV projects.
    4. Offer advice on Quant IV topics.
- **Problem Set 1**
    1. Release: February 21th, 17:00 PM (ET).
    2. Due: March 6th, 15:30 PM (ET).

Objectives

- Goals of this part of lecture:
  - Learn common tools to collect new and unstructured data.
  - Develop your specific workflow to collect new data.
  - Relevant for Problem Set 1 and Final Project.

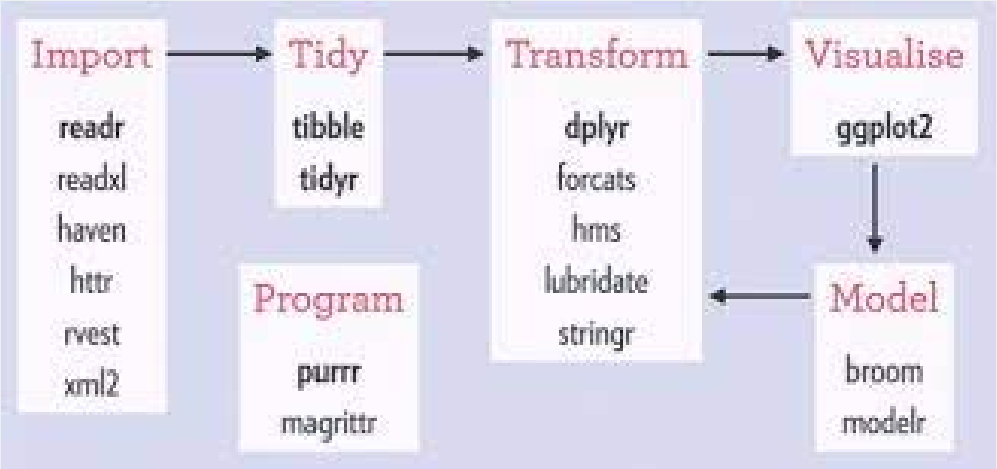
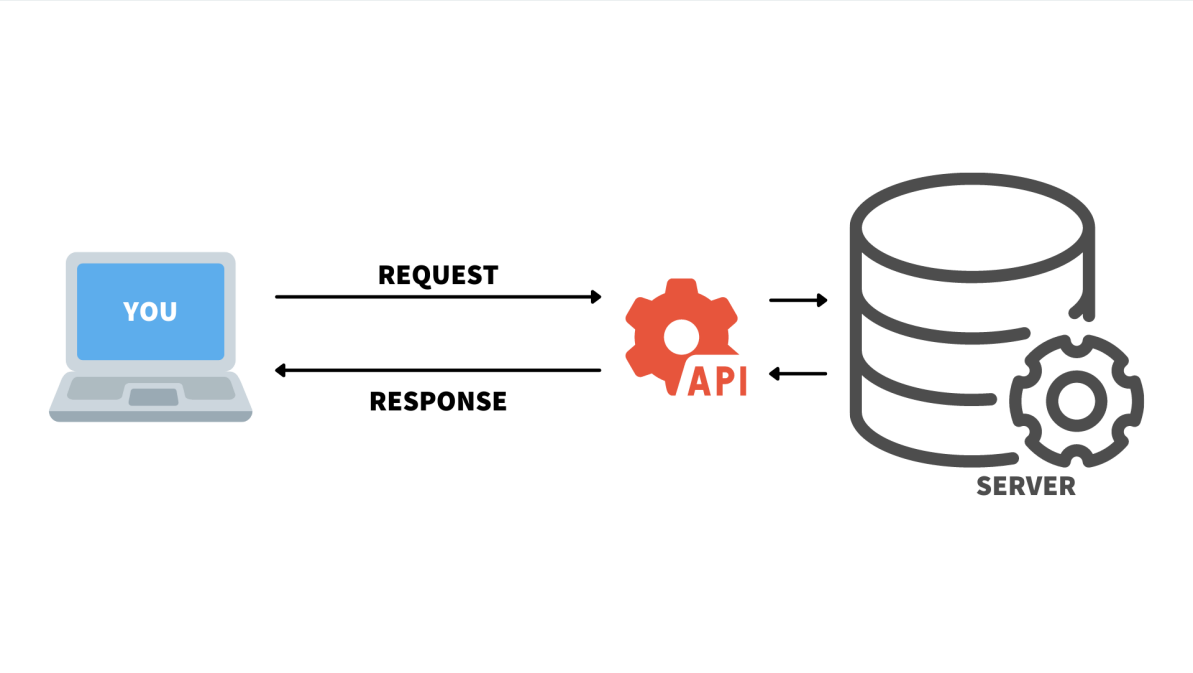


Figure: Example of Workflow (`tidyverse`)

2/ Application Programming Interface (API)

API Review



- **Canonical Example of Structure** Web APIs: two different computers — a client and server — interact with each other to request and provide data.
- APIs are messenger systems (waiter in a restaurant).
- Interactions = request verbs. Example: GET (`httr` package).

API Review

- Steps to collect data with API:
  1. Sign up for an app token/key.
  2. Identify the API endpoint.
  3. Filter the dataset (select columns and rows).
  4. Download selected data.

- GET function requires a URL (address of the server). Optional argument: query parameters (structured as list).
- **Query Parameters:** check the API documentation (authentication?).
- The output of the GET() function is a list, which contains all of the information that is returned by the API server. `Status` refers to the success or failure of the API request, and it comes in the form of a number. **Success** = 200. (`httr` function `status_code` ,*See Details)*.
- **Content:** most relevant output. Function `http_type` to determine what type of information is returned (JSON $\longrightarrow$`jsonlite`; XML $\longrightarrow$ `xml2`).

API Review

- Variety of APIs :
  - ProPublica Congress API: `https://projects.propublica.org/api-docs/congress-api/`
  - govinfo API: `https://api.govinfo.gov/docs/`
  - OECD: `https://data.oecd.org/api/`
  - NYT API: `https://developer.nytimes.com/`
  - WITS API: `https://wits.worldbank.org/witsapiintro.aspx?`
- In most cases, R package `httr` is useful:
  - Some come with R packages to interact with (`ProPublicaR`).
  - Some do not require API (e.g., Python `twint` for Twitter).
  - Ready to use packages (`gtrendsR`, `censusapi`).

API example: NYT API

- Signup for an API key https://developer.nytimes.com/
- Search for news articles including ”black lives matter” from Jan 1, 2013 to Dec 31, 2019

R Code

```
# API end point for article search
endpoint <- "http://api.nytimes.com/svc/search/v2/articlesearch.json"

begin_date <- "20130101"; end_date <- "20191231"

# Query Parameters
query_list <- list(q = "black lives matter", # search terms
                   fq = "source:(The New York Times) AND type_of_material:(News)", # filtering
                   begin_date = begin_date, end_date = end_date, # date
                  "api-key" = api_key_mine) # API key

# GET connects each parameter in query with &, create the URL to extracts data, and downloads
res <- httr::GET(endpoint, query = query_list)

# convert to human-readable format
res_content <- httr::content(res)
```

- GET in the above example creates this URL:
  http://api.nytimes.com/svc/search/v2/articlesearch.json?q=black%20lives%20matter&fq=source:(TheNewYorkTimes)%20AND%20type_of_material:(News)&begin_date=20130101&end_date=20191231&api-key=api_key_mine

3/ Web Scraping in R

Web Scraping in R

- Procedure
  1. Identify a URL to be examined for content.
  2. Analyze the html by extracting tags and attributes that contain data. Use Selector Gadet, xPath, or Google Insepct to identify the "selector" This will be a paragraph, table, hyper links, images.
  3. Use `read_html` to "read" the URL (`rvest` package).
  4. Pass the result to `html_nodes` to get the selectors identified in step number 2.
  5. Optional/Advanced:
     - Create folders to store data and use regex to name folders.
     - Download data and use regex to name each downloaded data.
  6. Some tips:
     - Vectorize (`stringr` functions accept a vector as input).
     - Read html file carefully.
     - Start with one string and repeat (try & error).
     - Use `Sys.sleep` to randomize your wait time.

Web Scraping in R

- Useful Resources:
  - HTML Basics.
  - *Web Scraping*.
  - *Automated Data Collection with R*.
  - *R Web Scraping Quick Start Guide*.
  - R4DS .
- Useful functions:
  - **read_html** (xml2) read html or xml.
  - **html_nodes** (rvest) extract specific portions of the html code.
  - **xml_child** (xml2) extract specific element.

Web Scraping in R

R Code

```
# Load packages
library(xml2)
library(rvest)

# Html to analyze
page <- "https://en.m.wikipedia.org/wiki/Lionel_Messi"

# Download Content
page_c <- rvest::read_html(page)

# Inspect
page_c

# Extract text paragraphs
page_c %>%
 rvest::html_nodes("p") %>%
  rvest::html_text() -> webtxt

# Inspect
head(webtxt)

# Extract table
page_c %>%
  rvest::html_nodes("table") %>%
  rvest::html_table() -> webtab

# Inspect
head(webtab)
```

Web Scraping in R

- Step-by-step example
- Download lobbying report
- Create folders for each year and store downloaded zip files

R Code

```
# html to analyze
page <- read_html("https://www.senate.gov/legislative/Public_Disclosure/contributions_download.ht

# extracting tags
links <- page %>% html_nodes("a") %>% html_attr("href")

# extract links with zip file
links <- links[grepl(".zip", links)]

# extract year info: these two lines should return the exact same output
years <- stringr::str_extract(links, "\\d{4}")
years2 <- gsub(".*(\\d{4}).*", "\\1", links)

# extract file name
filename <- basename(links)

# create year directory for the first URL
dir.create(years[1])

# download a zip file from the first URL
# make sure to check you are in the correct directory
download.file(url = links[1],
destfile = paste(years[1], filename[1], sep = "/"))
```

4/ Regular Expression (Regex)

Regular Expressions

- Sequence of characters that describes a certain pattern found in a text.
- Part of many programming languages (in R, the default are *extended* regular expressions).
- Regular expressions are constructed analogously to arithmetic expressions, by using various operators to combine smaller expressions. The whole expression matches zero or more characters.
- Considerations:
  - **Character Escapes**.
  - **Character Classes**.
  - **Quantifiers**.
  - **Anchors**.
  - **Grouping**.

- Cheat Sheets:
    1. `Basic Regular Expressions in R`.
    2. `Work with strings with stringr`.

stringr	**base R**
str_subset()	grep()
str_detect()	grepl()
str_extract()	regexpr(), regmatches(), grep()
str_match()	regexec()
str_locate()	regexpr()
str_locate_all()	gregexpr()
str_replace()	sub()
str_replace_all()	gsub()

Regular Expressions

R Code

```
# Load packages
library(stringr)

# Define string
x <- c("Lionel", "Andrés", "Messi", "also", "known", "as", "Leo", "Messi")

# Determine length
str_length(x)

# Combine strings
str_c(x, collapse = ", ")
str_c(x, collapse = " ")

# Extract first two letters
str_sub(x, start = 1, end = 2)

# Find matches
str_subset(x, pattern = "[aeiou]")
str_detect(x, pattern = "[aeiou]")

# Count matches
str_count(x, pattern = "[aeiou]")

# Locate match
str_locate(x, pattern = "L")

# Replace elements
str_replace(x, pattern = "Leo", replacement = "God")
```