Computational Social Science

Observational Data and Application Programming Interfaces

Dr. Thomas Davidson

Rutgers University

February 5, 2024

Plan

- Course updates
- Digital trace data
- Application Programming Interfaces (APIs)
  - What are they?
  - How do they work?
  - Using an API
- The Post-API Age?

Course updates

Homework

- Homework 1 released last week, due Wednesday at 5pm.
  - Don't leave it until the last minute!
  - Please push your final version to Github with the appropriate commit message.
  - Make sure to also submit the URL on Canvas

Digital trace data

Digital traces and "big data"

"The first way that many people encounter social research in the digital age is through what is often called big data. Despite the widespread use of this term, there is no consensus about what big data even is." - Salganik, C2.2

Digital trace data

Advantages of "big data"

- Size
  - Large-scale processes
  - Hetereogeneity
  - Rare events
  - Small effects

Digital trace data

Advantages of "big data"

- Always-on
  - Longitudinal studies
  - Unexpected events
- Non-reactive
  - Stigmatized behaviors
  - Hidden populations

Digital trace data

Disadvantages of "big data"

- Incomplete
- Inaccessible
- Non-representative
- Drift
- Algorithmic confounding
- Dirty
- Sensitive

Digital trace data

Big data and observational data

"A first step to learning from big data is realizing that it is part of a broader category of data that has been used for social research for many years: observational data. Roughly, observational data is any data that results from observing a social system without intervening in some way." - Salganik, C2.1

Digital trace data

Repurposing digital traces

"In the analog age, most of the data that were used for social research was created for the purpose of doing research. In the digital age, however, a huge amount of data is being created by companies and governments for purposes other than research, such as providing services, generating profit, and administering laws. Creative people, however, have realized that you can repurpose this corporate and government data for research." - Salganik, C2.2

Digital trace data

Kinds of analysis

- Counting things
  - e.g. What types of social media posts are censored in China?
- Forecasting
  - e.g. Can we predict flu prevalence using Google Searches?
- Approximating experiments
  - e.g. Does social influence predict product adoption?

APIs

What is an API?

- An Application Programming Interface is a way to programmatically interact with a website
- You can make requests to request, modify, or delete data
  - Different APIs allow for different types of interactions
  - Most of the time you will want to request data
- Remember: APIs are typically created for developers with different user cases to academic researchers

APIs

APIs and observational data

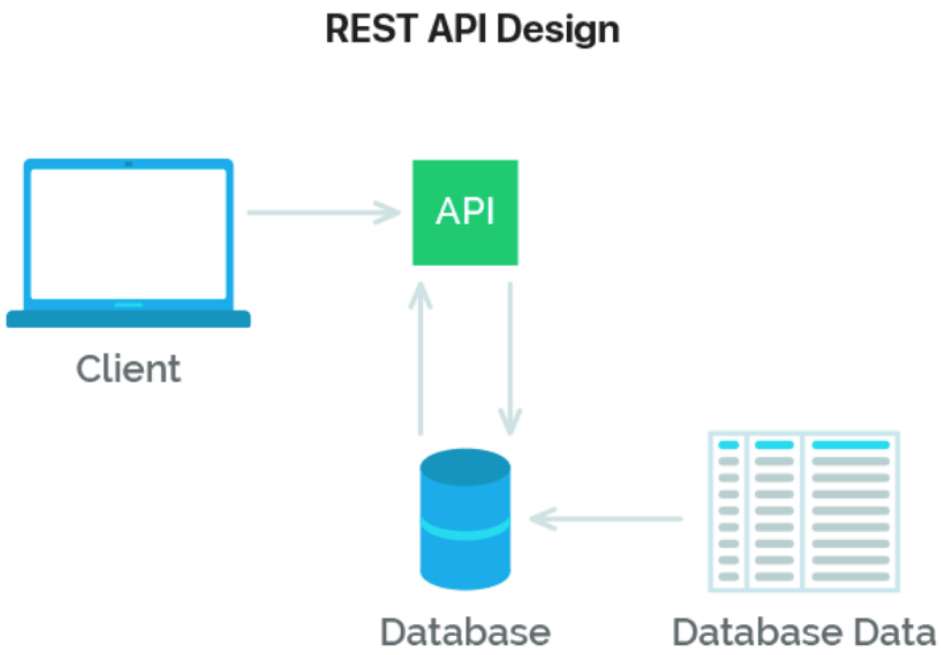
- APIs serve as one of the primary means for creating observational datasets
- Many research papers by computational social scientists use data collected from APIs including Facebook, Twitter, and Reddit
- There are many data science tools that use APIs as a way to facilitate access to more complex (and often proprietary) models (e.g. Google's Perspective API)

APIs

How does an API work?

1. Construct an API request
2. The client (our computer) sends the request to the API server
3. The server processes the request, retrieving any relevant data from a database
4. The server sends back the requested data to the client

How does an API work?



https://sites.psu.edu/annaarsiriy/files/2019/02/Screen-Shot-2019-02-10-at-2.31.08-PM-1p26wa2.png

APIs

Github API example

- Here is a simple call to `users` *endpoint* of the Github API: https://api.github.com/users/t-davidson
- Note the API call uses a modified version of the Github URL, `api.github.com`

APIs

Github API example

- The original API call provides us with other information. We could use this to find my followers by querying the `followers` endpoint: https://api.github.com/users/t-davidson/followers
- Most APIs have documentation, explaining how each endpoint works.

APIs

Parameters

- The documentation we just saw shows that we can add other parameters to our query
  - We can use these parameters by adding them as query strings using the ?.
  - Each parameter has an argument specified after =.
  - We can separate multiple arguments using the & symbol.

APIs

Parameters and arguments

- What do the following queries do?
  - https://api.github.com/users/t-davidson/followers?per_page=50
  - https://api.github.com/users/t-davidson/followers?per_page=100&page=2
  - https://api.github.com/users/t-davidson/followers?page=3

APIs

Credentials

- Often APIs will use credentials to control access
  - A *key* (analogous to a user name)
  - A *secret* (analogous to a password)
  - An *access token* (grants access based on key and password)
  - Generally the access token is provided as part of the call
- Keep credentials private
  - Avoid accidentally sharing them on Github

APIs

Rate-limiting

- APIs use rate-limiting to control usage
  - How many API calls you can make
  - How much data you can retrieve
- Obey rate-limits, otherwise your credentials may be blocked
- APIs sometimes show you your rate limits
  - e.g., `https://api.github.com/rate_limit`
  - Beware: sometimes there will be rate limits on the rate limit endpoint!

APIs

JSON

- An API will commonly return data in JSON (JavaScript Object Notation) format
  - JSON files consist of key-value pairs, enclosed in braces as such: `{"key": "value"}`
  - JSON files are structured in a way that makes them relatively easy to parse to retrieve relevant data

APIs

Calling an API in R

- The following example shows how we can interact with an API in R
- We will use the `httr` package to make GET requests to query the Github API

APIs

Calling an API in R

```
library(httr)
library(jsonlite)
library(tidyverse)

url <- "https://api.github.com/users/t-davidson"
request <- GET(url = url)
response <- content(request, as = "text", encoding = "UTF-8")
data <- fromJSON(response)
```

See Wikipedia for a primer on UTF-8 encoding.

APIs

Calling an API in R

We can use pipes to chain together these operations.

```
data <- GET(url = url) %>%
  content(as = "text", encoding = "UTF-8") %>%
  fromJSON()
```

APIs

Calling an API in R

`class` shows us that the object is a list. We can then use the $ operator to pull out specific elements.

```
class(data)
```

```
## [1] "list"
```

```
data$name
```

```
## [1] "Tom Davidson"
```

```
data$followers_url
```

```
## [1] "https://api.github.com/users/t-davidson/followers"
```

APIs

Calling an API in R

We can make another API call to get information on followers.

```
followers <- GET(url = data$followers_url) %>%
  content(as = "text", encoding = "UTF-8") %>%
  fromJSON() %>% as_tibble()
```

APIs

```
print(followers)
```

```
## # A tibble: 30 x 18
##    login           id node_id avatar_url gravatar_id url   html_url f
##    <chr>        <int> <chr>   <chr>      <chr>       <chr> <chr>    <
##  1 loretopar~ 1.63e5 MDQ6VX~ https://a~ ""          http~ https:/~ h
##  2 korymath   1.78e5 MDQ6VX~ https://a~ ""          http~ https:/~ h
##  3 tejastank  3.11e5 MDQ6VX~ https://a~ ""          http~ https:/~ h
##  4 alexhanna  7.98e5 MDQ6VX~ https://a~ ""          http~ https:/~ h
##  5 pablobarb~ 8.29e5 MDQ6VX~ https://a~ ""          http~ https:/~ h
##  6 ibrahimis~ 8.81e5 MDQ6VX~ https://a~ ""          http~ https:/~ h
##  7 mukeshtiw~ 1.14e6 MDQ6VX~ https://a~ ""          http~ https:/~ h
##  8 pixelandp~ 1.39e6 MDQ6VX~ https://a~ ""          http~ https:/~ h
##  9 matthewjd~ 1.50e6 MDQ6VX~ https://a~ ""          http~ https:/~ h
## 10 quarbby    1.67e6 MDQ6VX~ https://a~ ""          http~ https:/~ h
## # i 20 more rows
## # i 10 more variables: following_url <chr>, gists_url <chr>, starred
## #   subscriptions_url <chr>, organizations_url <chr>, repos_url <chr
## #   events_url <chr>, received_events_url <chr>, type <chr>, site_ad
```

APIs

Calling an API in R

Let's make a function to repeat this process.

```
get.followers <- function(followers.url) {
  followers <- GET(url = followers.url) %>%
  content(as = "text", encoding = "UTF-8") %>%
  fromJSON() %>% as_tibble()
  return(followers)
}
```

APIs

Calling an API in R

```
senders <- character() # list of people following others
receivers <- character() # list of those receiving ties

k <- 5
for (i in 1:dim(followers)[1]) {
  i.id <- followers$login[i] # get follower name
  receivers <- append(receivers, "t-davidson") # update edge-listss
  senders <- append(senders, i.id)
  i.followers <- get.followers(followers$followers_url[i]) # get i's fo
  for (j in 1:dim(i.followers)[1]) {# for each follower
    if (j <= k) { # only consider their first k followers
        receivers <- append(receivers, i.id) # update edgelist
        senders <- append(senders, i.followers$login[j])
    }
  }
}
```

We can now use this function to build a network.

APIs

Calling an API in R

```
# install.packages(c("igraph", "ggnetwork")) # uncomment and run to ins
library(igraph)
library(ggnetwork)

A <- cbind(senders[1:100], receivers[1:100]) # Construct matrix with fi
G <- graph_from_edgelist(A, directed = TRUE) # construct a graph
G
```

```
## IGRAPH 53a033f DN-- 98 100 --
## + attr: name (v/c)
## + edges from 53a033f (vertex names):
##  [1] loretoparisi  ->t-davidson   PhilAndrew     ->loretoparisi
##  [3] gscalzo       ->loretoparisi karmatr0n      ->loretoparisi
##  [5] comster       ->loretoparisi ksopyla        ->loretoparisi
##  [7] korymath      ->t-davidson   mattt          ->korymath
##  [9] musha68k      ->korymath     douglasdollars->korymath
## [11] fly51fly      ->korymath     silky          ->korymath
## [13] tejastank     ->t-davidson   ibuilder       ->tejastank
```

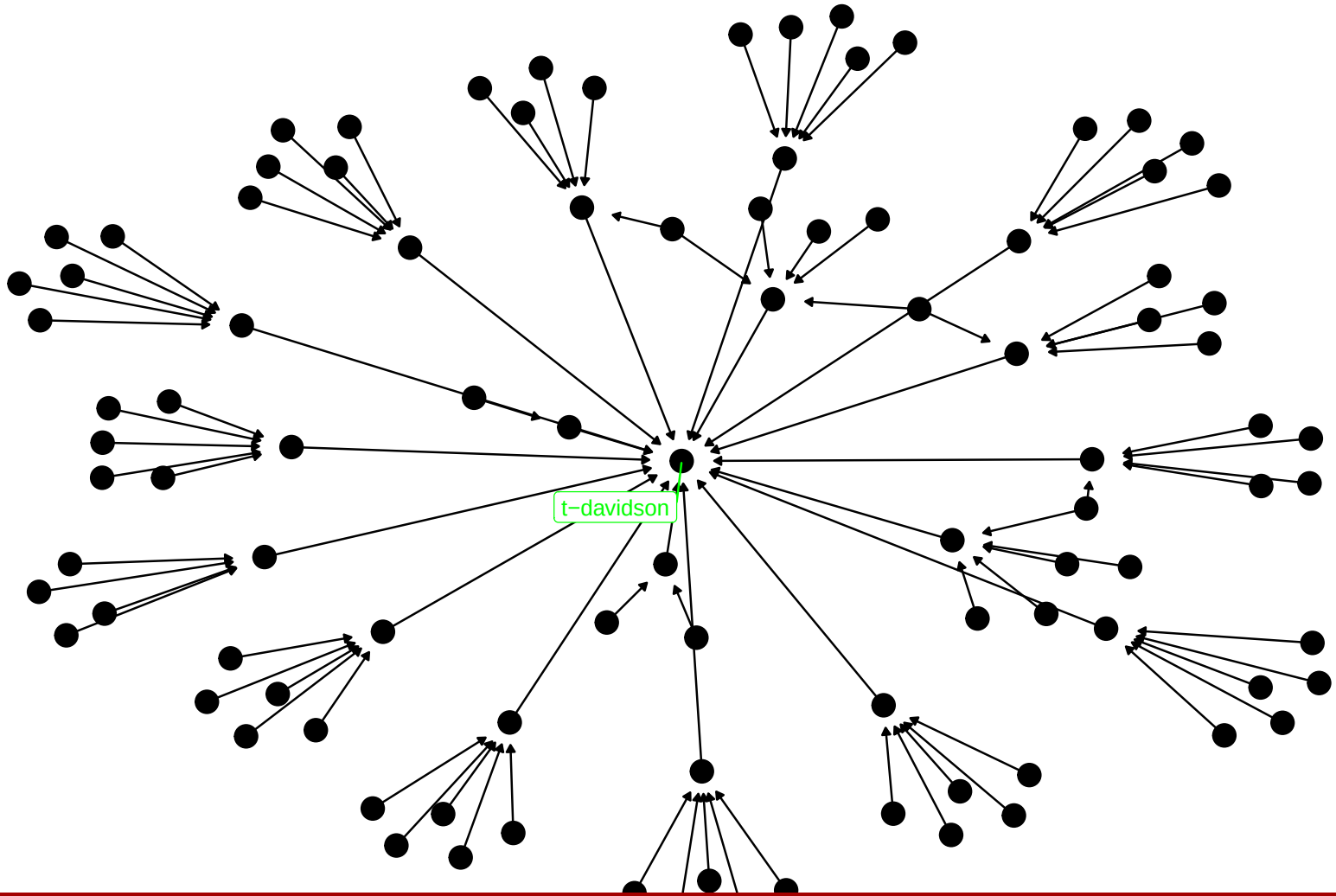
APIs

Calling an API in R

```
p <- G %>% ggnetwork() %>% ggplot(aes(x = x, y = y, xend = xend, yend =
  geom_edges(arrow = arrow(length = unit(4, "pt"), type = "closed")) +
  geom_nodes(size=5) +
  geom_nodelabel_repel(aes(label=ifelse(name == "t-davidson", name, NA)
  theme_blank()
```

APIs

Calling an API in R



APIs

Best-practices

- Use a "wrapper" package if available (Wednesday's lecture)
  - Although sometimes you will have to write your own queries
- Build in functions to obey rate-limits where possible
- Access only the data you need
- Test using small examples before collecting a large dataset

APIs

The post-API age?

- Following Facebook's decision in 2018 to close down access to its Pages API, Deen Freelon writes:
  - "We find ourselves in a situation where heavy investment in teaching and learning platform-specific methods can be rendered useless overnight."
- Freelon's recommendations
  - Learn to web scrape (next week)
  - Understand terms of service and implications of violating them

APIs

The post-API age?

- The future for APIs seems brighter
  - Facebook Pages and Group data available via CrowdTangle
  - Twitter Academic API provides extensive access
  - Much of Reddit is available via Pushshift

Next lecture

- Interacting with APIs using R