Computational Sociology

Observational Studies and Application Programming Interfaces

Dr. Thomas Davidson

Rutgers University

February 8, 2021

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 Wednesday (2/10) at 5pm Eastern.
    - Please push your final version to Github with the appropriate commit message
    - Use Slack to ask questions and collaboratively solve problems
    - I can also provide help using the feedback pull request feature on Github

Course updates

Project proposal

- Meet to discuss project ideas
- Project proposal due 3/2, 3-5 pages double-spaced
  - Introductory paragraph
  - Brief literature review (~1 page)
    - Focus on your main research question(s)
  - Data
    - Potential data sources
    - Collection plan
  - Methodology
    - Possible methods for analysis
  - References

Digital trace data

Defining "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?

Digital trace data

Some advice

- For most social scientific questions we don't need huge data
  - Analyses are also intractable with large datasets
- Start with a question then find data to answer it
  - Although data exploration can lead to new questions

APIs

What is an API?

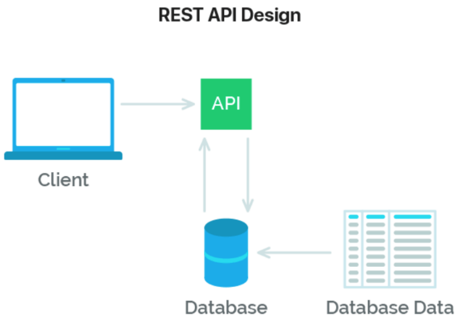
- An Application Programming Interface is a way to programmatically interact with a website
- It allows servers to directly communicate with one another
- You can make requests to request, modify, or delete data
  - Different APIs allow for different types of interactions
  - Most of the time you will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

APIs

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

APIs

Github API example

The original API call provides us with other information. We could use this to find my followers:
https://api.github.com/users/t-davidson/followers

APIs

Credentials

- Often APIs will use credentials to control access
  - A *key* (analogous to a username)
  - 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

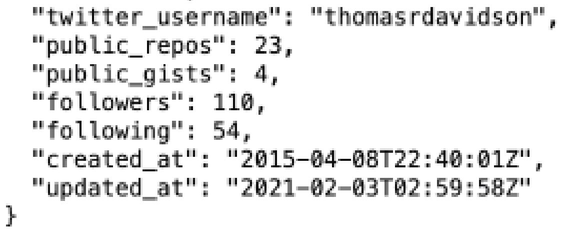
Rate-limiting

- 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`

APIs

JSON

- An API will commonly return data in JSON (JavaScript Object Notation) format
  - Format: `{"key": "value"}`



```
  "twitter_username": "thomasrdavidson",
  "public_repos": 23,
  "public_gists": 4,
  "followers": 110,
  "following": 54,
  "created_at": "2015-04-08T22:40:01Z",
  "updated_at": "2021-02-03T02:59:58Z"
}
```

APIs

Calling an API in R

```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)
```

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 follow
##    <chr>   <int> <chr>   <chr>      <chr>       <chr> <chr>    <chr>
##  1 geor~ 4.74e6 MDQ6VX~ https://a~ ""          http~ https:/~ https:
##  2 mari~ 8.75e6 MDQ6VX~ https://a~ ""          http~ https:/~ https:
##  3 Mich~ 6.67e6 MDQ6VX~ https://a~ ""          http~ https:/~ https:
##  4 emil~ 2.12e7 MDQ6VX~ https://a~ ""          http~ https:/~ https:
##  5 Alex~ 2.21e7 MDQ6VX~ https://a~ ""          http~ https:/~ https:
##  6 wyfsh 2.23e7 MDQ6VX~ https://a~ ""          http~ https:/~ https:
##  7 ferp~ 2.26e7 MDQ6VX~ https://a~ ""          http~ https:/~ https:
##  8 acoo~ 2.31e7 MDQ6VX~ https://a~ ""          http~ https:/~ https:
##  9 malu~ 2.31e7 MDQ6VX~ https://a~ ""          http~ https:/~ https:
## 10 matt~ 1.50e6 MDQ6VX~ https://a~ ""          http~ https:/~ https:
## # ... with 20 more rows, and 10 more variables: following_url <chr>,
## #   gists_url <chr>, starred_url <chr>, subscriptions_url <chr>,
## #   organizations_url <chr>, repos_url <chr>, events_url <chr>,
## #   received_events_url <chr>, type <chr>, site_admin <lgl>
```

APIs

Calling an API in R

Let's make a function to repeat this process to get the followers' followers.

```r
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

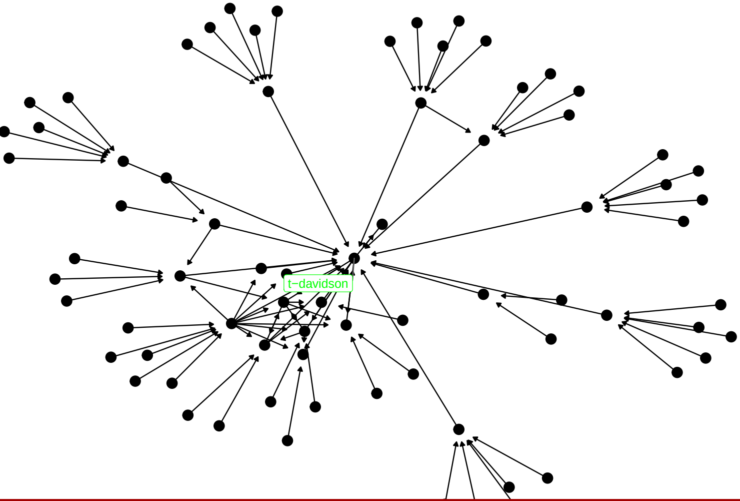
```r
senders <- character() # list of people following others
receivers <- character() # list of those receiving ties

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 <= 5) { # only consider their first 5 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



APIs

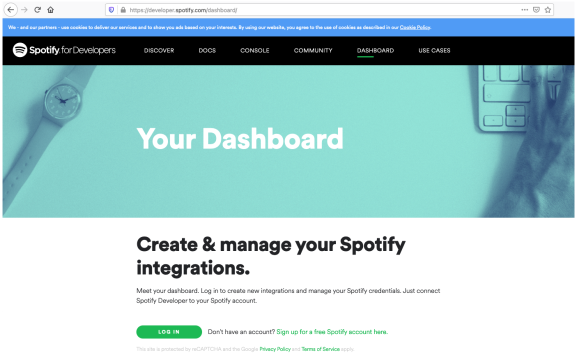
Using the Spotify API

- It's always good to start by reading the documentation
  - https://developer.spotify.com/documentation/web-api/
- This provides information on the API, endpoints, rate-limits, etc.

APIs

Using the Spotify API

Let’s log in to use the API.



https://developer.spotify.com/dashboard/

We - and our partners - use cookies to deliver our services and to show you ads based on your interests. By using our website, you agree to the use of cookies as described in our Cookie Policy.

Create & manage your Spotify integrations.

Meet your dashboard. Log in to create new integrations and manage your Spotify credentials. Just connect Spotify Developer to your Spotify account.

Don't have an account? Sign up for a free Spotify account here.

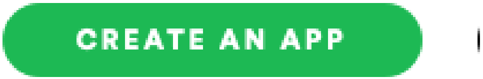
This site is protected by reCAPTCHA and the Google Privacy Policy and Terms of Service apply.

https://developer.spotify.com/dashboard/

APIs

Using the Spotify API

Click on this button to create a new app.



APIs

Using the Spotify API

- Copy your credentials and paste them into `creds.json`
  - Storing credentials in a separate file helps to keep them separated from anything you might commit to Github

APIs

Using the Spotify API

We're going to be using `spotifyr`, a *wrapper* around the spotify API. This allows us to make use of the functionality without needing to write the API calls, make requests, or convert the results to JSON/tabular format.

```
# library(devtools) devtools::install_github('charlie86/spotifyr') # in
# directly from github
library("spotifyr")
```

`devtools` package can be used to install a library directly from Github. You can read more about the library here: https://www.rcharlie.com/spotifyr/.

APIs

Using the Spotify API

Now let's read in the credentials and create a token.

```
creds <- read_json("creds.json")  # read creds

Sys.setenv(SPOTIFY_CLIENT_ID = creds$id)  # set creds
Sys.setenv(SPOTIFY_CLIENT_SECRET = creds$secret)

access_token <- get_spotify_access_token()  # retrieve access token
```

APIs

Using the Spotify API

Now we're authorized, we can use the package to retrieve information from the API.

```
#?get_artist_audio_features
stones <- get_artist_audio_features("The Rolling Stones") %>% as_tibble
colnames(stones)
```

```
##  [1] "artist_name"                  "artist_id"
##  [3] "album_id"                     "album_type"
##  [5] "album_images"                 "album_release_date"
##  [7] "album_release_year"           "album_release_date_precision"
##  [9] "danceability"                 "energy"
## [11] "key"                          "loudness"
## [13] "mode"                         "speechiness"
## [15] "acousticness"                 "instrumentalness"
## [17] "liveness"                     "valence"
## [19] "tempo"                        "track_id"
## [21] "analysis_url"                 "time_signature"
## [23] "artists"                      "available_markets"
```

APIs

Using the Spotify API

```
head(stones$track_name, n=10)
```

```
##  [1] "Start Me Up - Live"              "Bitch - Live"
##  [3] "Sad Sad Sad - Live"              "Undercover Of The Night - Liv
##  [5] "Harlem Shuffle - Live"           "Tumbling Dice - Live"
##  [7] "Miss You - Live"                 "Terrifying - Live"
##  [9] "Ruby Tuesday - Live"             "Salt Of The Earth - Live"
```

APIs

Using the Spotify API

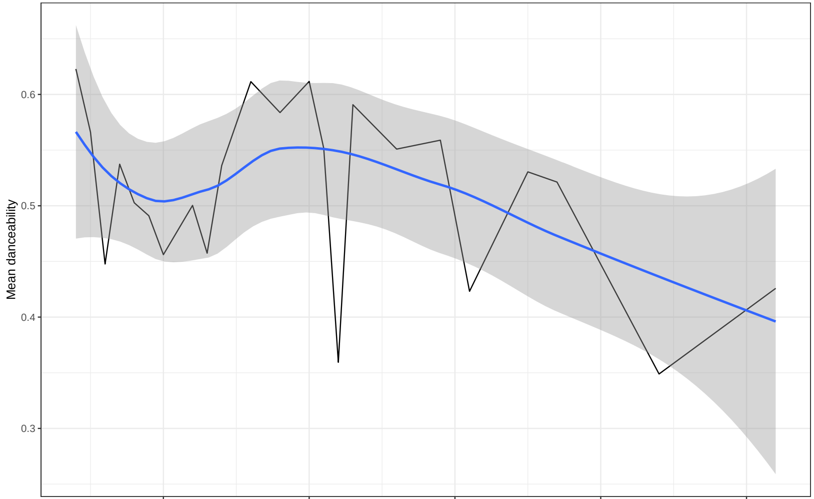
We can use this `tibble` to calculate some statistics about the group.

```
results <- stones %>% filter(album_release_year < 2015) %>% group_by(al
```

APIs

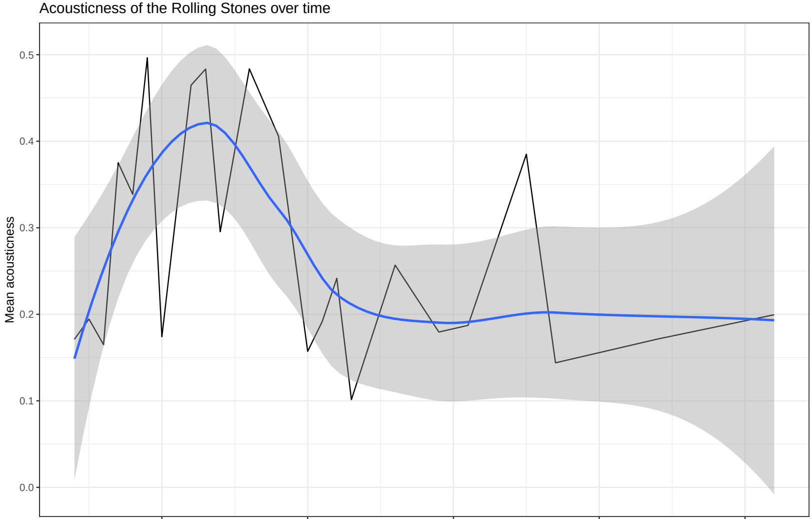
Using the Spotify API

Danceability of the Rolling Stones over time



APIs

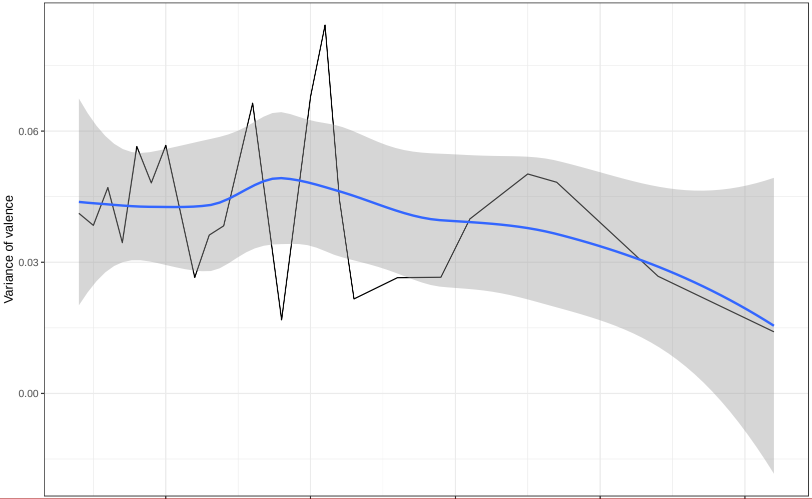
Using the Spotify API



APIs

Using the Spotify API

Variance in valence of the Rolling Stones over time



APIs

Using the Spotify API

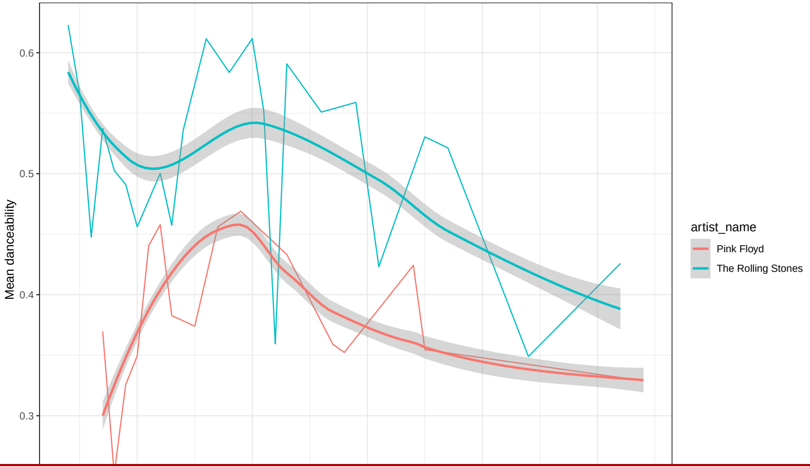
Let's collect the same data for Pink Floyd and combine it.

APIs

Using the Spotify API

Let's compare their danceability.

Comparing danceability of the Rolling Stones and Pink Floyd



APIs

Using the Spotify API

- Some features require a Spotify login
  - Edit the settings for the app and add `http://localhost:1410/` to Redirect URIs
  - This tells the API to open up authentication on port 1410 of your computer

APIs

Using the Spotify API

```
library(lubridate)
get_my_recently_played(limit = 5) %>%
    mutate(artist.name = map_chr(track.artists, function(x) x$name[1]),
           played_at = as_datetime(played_at)) %>%
    select(track.name, artist.name, track.album.name, played_at) %>% as
```

Example from the `spotifyr` documentation. Note this will only work if you have added a redirect URI to your app.

APIs

Best-practices

- If available, use a wrapper
  - 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."
- 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 is making more data available via CrowdTangle and other APIs
        - Although stricter protections for user data due to privacy concerns
    - Twitter just announced increases in access for academic researchers
    - Much of Reddit is available via Pushshift

Questions?