Computational Sociology

Observational Data and Application Programming Interfaces

Dr. Thomas Davidson

Rutgers University

February 8, 2024

Plan

- Course updates
- Digital traces and observational data
- Application Programming Interfaces (APIs)
- Using APIs in R
  - Github
  - Spotify
- The Post-API Age?

Course updates

Homework

- Homework 1 due tomorrow at 5pm.
  - Don't leave it until the last minute!
  - Please push your final version to Github with the appropriate commit message
  - Use Slack for questions

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

Application programming interfaces (APIs)

- 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 we will want to request data
- Remember: APIs are typically created for developers with different use cases from academic researchers

Digital trace data

APIs and observational data

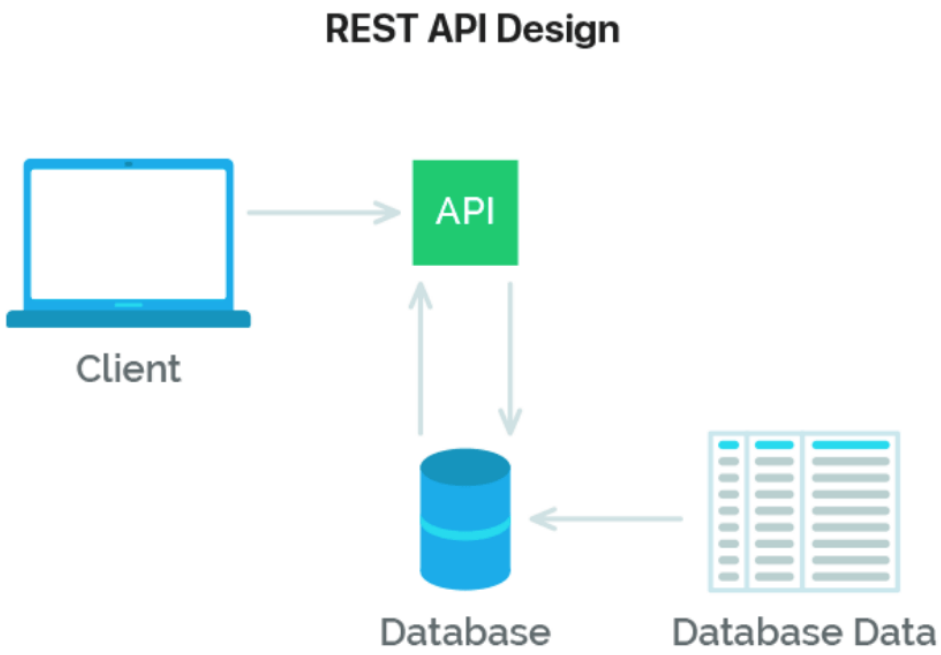
- APIs can be used to construct observational datasets
- Many organizations have APIs including social media platforms (e.g., Facebook, TikTok), governments (e.g., Census Bureau, NYC Open Data), media (e.g., NY Times, Chronicling America), academic publishers (e.g., JSTOR, Web of Science)
- See https://github.com/public-apis/public-apis for a vast list of APIs
- There are many data science tools that use APIs as a way to facilitate access to more complex (often proprietary) systems (e.g., Google's Perspective API, Google Geocoding API, OpenAI 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

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

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

Using the Github API

Calling an API in R (the hard way)

```
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

JSON

- An API will commonly return data in JSON (JavaScript Object Notation) format, a common way of storing structured data
  - 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 and can easily be converted into a `list` in R

Using the Github API

More efficient syntax

We can use pipes to chain together these operations.

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

Using the Github API

Inspecting the results

`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"
```

Using the Github API

Calling another endpoint

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()
```

Using the Github API

Inspecting the results

```
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
```

Using the Github API

Making a function

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

Using the Github API

A more complex query

```
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$login[i] # get follower name
  receivers <- append(receivers, "t-davidson") # update edge-lists
  senders <- append(senders, i.id)
  i.followers <- get.followers(followers$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])
    }
  }
}
```

Using the Github API

Constructing a network

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

A <- cbind(senders[1:150], receivers[1:150]) # Construct matrix with fi
G <- graph_from_edgelist(A, directed = TRUE) # construct a graph
G
```

```
## IGRAPH 4901868 DN-- 140 150 --
## + attr: name (v/c)
## + edges from 4901868 (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
```

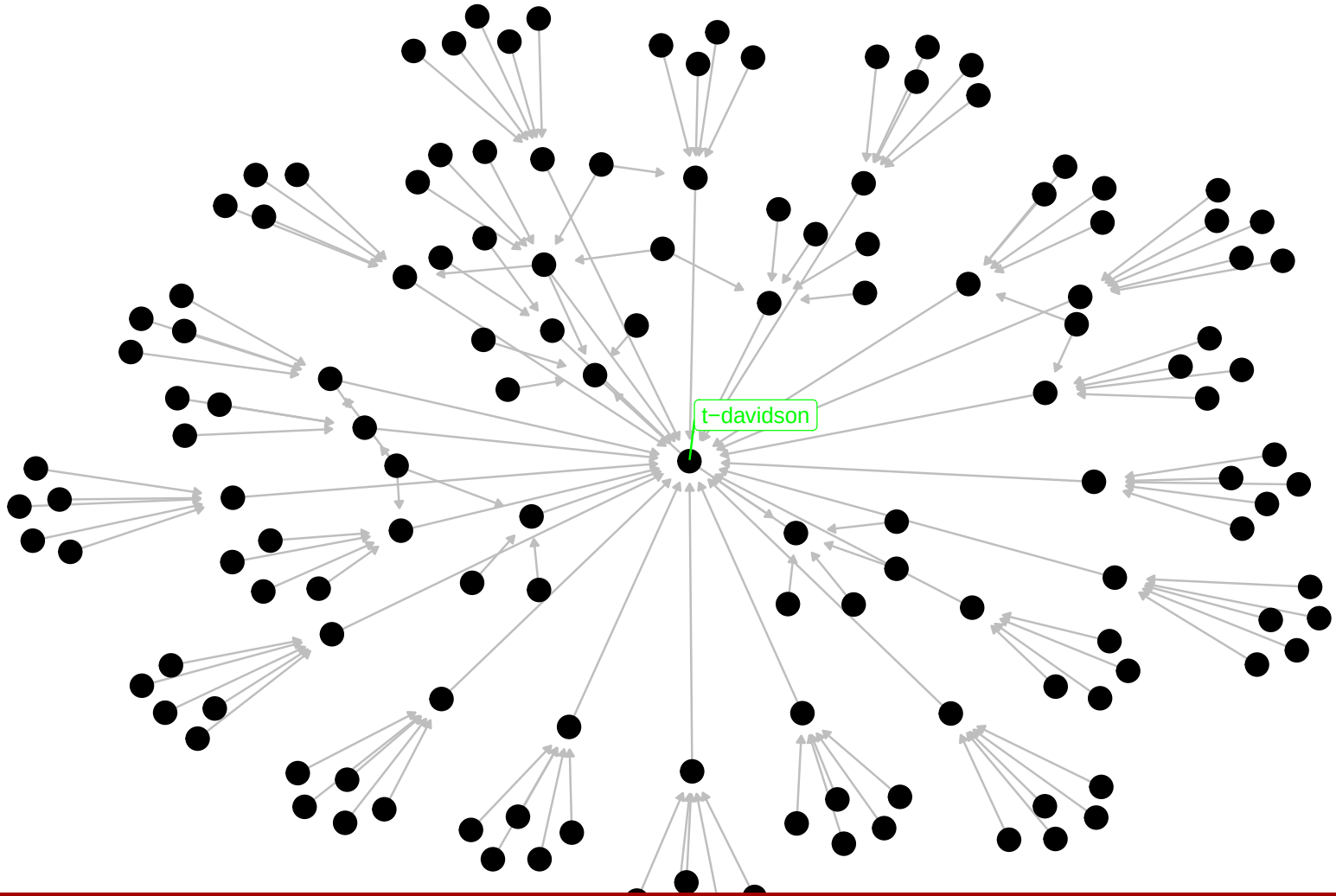
Using the Github API

Plotting the results

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

Using the Github API

Plotting the results



Using APIs for sociological research

Optimal differentiation in the music industry

- Askin and Mauskapf (2017) use data from Spotify to measure features of music
- Theory: successful music needs to exhibit *optimal differentiation*
  - Distinct from recent offerings, but not too different to be unrecognizable
- Using metrics from Echo Nest, a subsidiary of Spotify, they construct a measure of typicality and use it to predict chart position and duration, net of controls
  - 25,102 songs in charts 1958-2016

Musical features

Table 1. The Echo Nest Sonic Features

Attribute	Scale	Definition
Acousticness	0–1	Represents the likelihood that the song was recorded solely by acoustic means (as opposed to more electronic/electric means).
Danceability	0–1	Describes how suitable a track is for dancing. This measure includes tempo, regularity of beat, and beat strength.
Energy	0–1	A perceptual measure of intensity throughout the track. Think fast, loud, and noisy (i.e., hard rock) more than dance tracks.
Instrumentalness	0–1	The likelihood that a track is predominantly instrumental. Not necessarily the inverse of speechiness.
Key	0–11 (integers only)	The estimated, overall key of the track, from C through B. We enter key as a series of dummy variables.
Liveness	0–1	Detects the presence of a live audience during the recording. Heavily studio-produced tracks score low on this measure.
Mode	0 or 1	Whether the song is in a minor (0) or major (1) key.
Speechiness	0–1	Detects the presence of spoken word throughout the track. Sung vocals are not considered spoken word.
Tempo	Beats per minute (BPM)	The overall average tempo of a track.
Time Signature	Beats per bar/measure	Estimated, overall time signature of the track. 4/4 is the most common time signature by far and is entered as a dummy variable in our analyses.
Valence	0–1	The musical positiveness of the track.

Features predict success as longevity

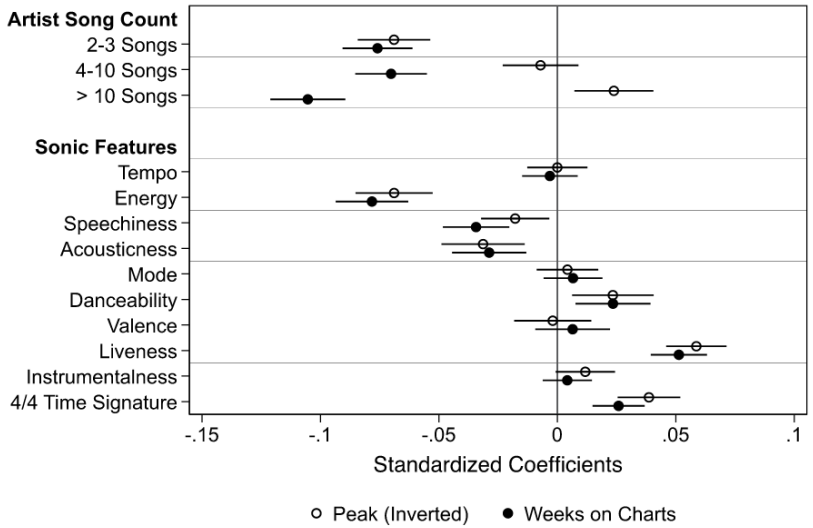
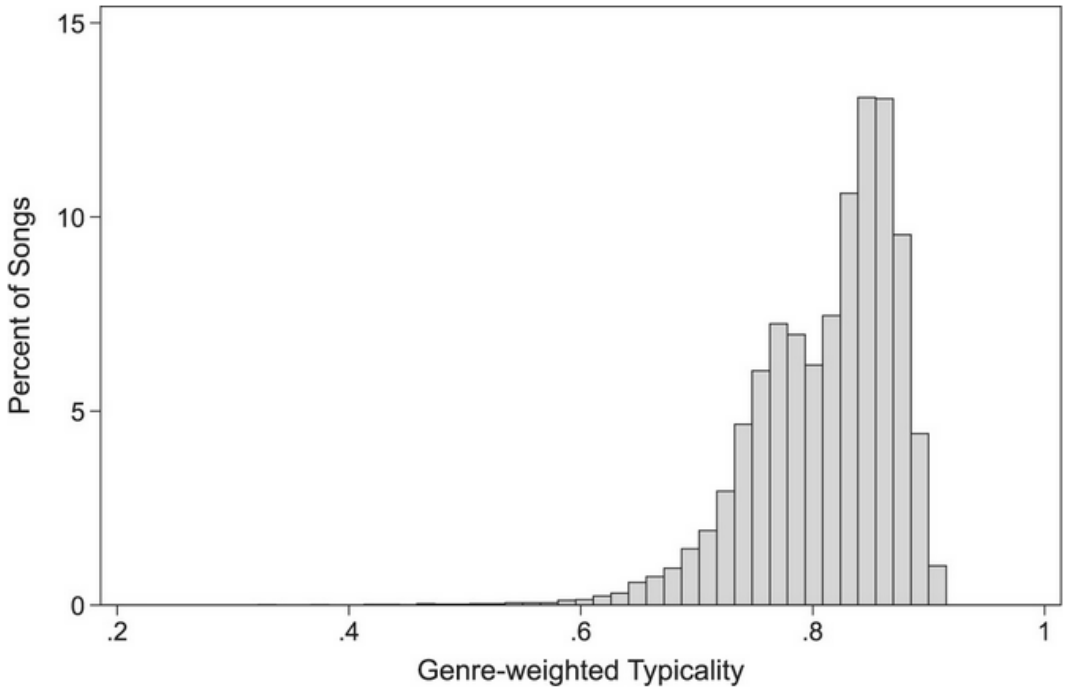


Figure 3. Select Standardized Coefficients from Pooled, Cross-Sectional OLS Models Predicting *Billboard* Hot 100 Peak Chart Position and Longevity (Models 1 and 2)
Note: Horizontal bars represent 95% CI. See Appendix Table A1 for full (unstandardized) results.

Measuring typicality



Optimal typicality and success

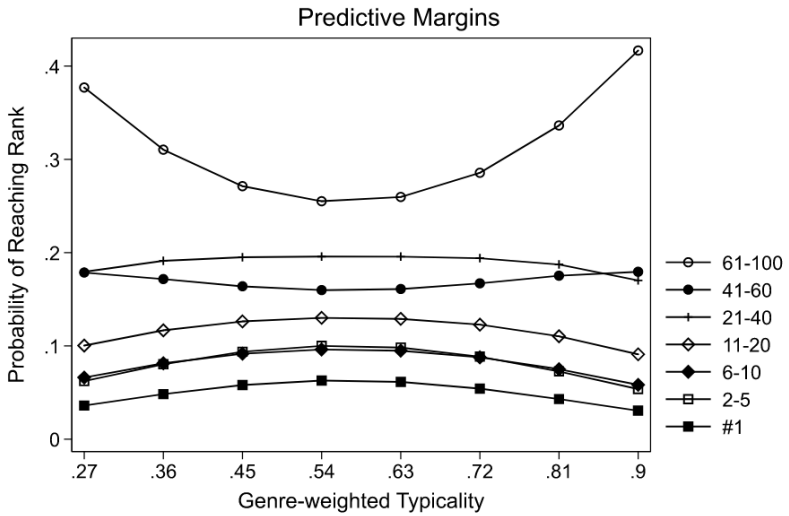


Figure 4. Predicted Marginal Probability of Songs Achieving Selected Peak Position (by Typicality) from Ordered Logit Model (Model 4)
Note: Although we inverted chart position in our models to assist readers with a more straightforward interpretation (e.g., positive coefficients reflect better performance), we revert to the originally coded chart positions for our marginal-effects graphical analysis. In the figure, the predicted positions are coded as they would be on the charts (i.e., #100 is the lowest, #1 the highest).

Using the Spotify API

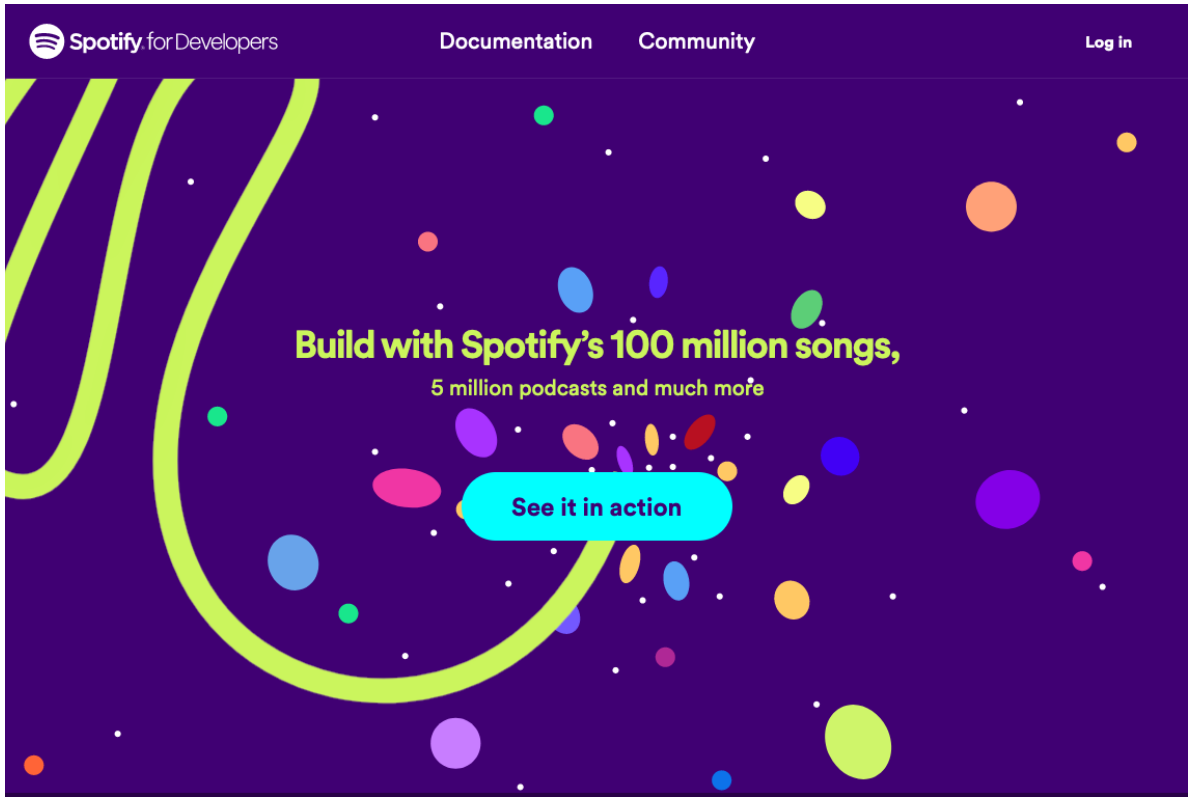
Documentation

- 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.

Using the Spotify API

Signing up

This API requires authentication. Let's log in to use the API.



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

Using the Spotify API

Creating an app

Accept the terms of service then click on this button to create a new app.

Using the Spotify API

Creating an app

- Add a name and a short description
  - e.g. “Computational Social Science”, “App for class”
- Click on the app in Dashboard
- Click “Edit Settings”
  - Add http://localhost:1410/ to the Redirect URIs and click Save
- Click “SHOW CLIENT SECRET”
  - Copy Client ID and Client Secret

APIs

Access 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

Using the Spotify API

Storing credentials

- Open `creds.json` (located in the `credentials` folder) and paste the ID and secret into the relevant fields.
  - Storing credentials in a separate file helps to prevent them from getting committed to Github accidentally
- The file should look like this:

```
{"id": "328248djkejf298382189du329323c",
"secret": "jw7329889d37f7798383e8d29ew2d"}
```

Using the Spotify API

Loading packages

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.

```
# install.packages('spotifyr') # uncomment and run to install
library(spotifyr)
library(tidyverse)
library(jsonlite)
library(lubridate)
```

You can read more about the library here.

Using the Spotify API

Authentication

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

```
creds <- read_json("../credentials/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
```

Using the Spotify API

API functions

Now we're authorized, we can use the package to retrieve information from the API. Let's take a look at one of the functions. Rather than writing all the query code ourselves, we can just pass query parameters to the function.

```
`?`(get_artist_audio_features)
print(get_artist_audio_features)
```

Using the Spotify API

Querying the API

Now we're authorized, we can use the package to retrieve information from the API. Let's take a look at one of the functions.

```
artist1 <- get_artist_audio_features("") %>% as_tibble() # Add artist n
head(artist1)
```

Using the Spotify API

Inspecting the data

```
head(artist1$track_name, n=10)
```

Using the Spotify API

Creating a summary

Let's calculate some statistics using this table.

```
results <- artist1 %>%
             group_by(album_release_year) %>%
  summarize(mean.dan = mean(danceability),
             mean.ac = mean(acousticness))
```

Using the Spotify API

Visualizing the data

```
p <- ggplot(artist1, aes(x=album_release_year, y=danceability))
p + geom_smooth() +
  labs(title="Danceability over time", caption = "Data from collect fro
  xlab("") + ylab("Mean danceability") + theme_bw()
```

Using the Spotify API

Visualizing the data

```
p <- ggplot(artist1, aes(x=album_release_year, y=acousticness))
p + geom_smooth() +
  labs(title="Acousticness over time", caption = "Data from collect fro
    xlab("") + ylab("Mean acousticness") + theme_bw()
```

Using the Spotify API

Collecting more data

Let's collect the same data for a second artist and combine it.

```
artist2 <- get_artist_audio_features("") %>% as_tibble()
both <- bind_rows(artist1, artist2) # adding 2nd artist to the same tib
both %>% sample_n(5) %>% select(artist_name)
```

Using the Spotify API

Creating a new summary

Repeating the summary operation for both artists. Note how we now group by `artist_name` in addition to `album_release_year`.

```
r <- both %>%
  group_by(album_release_year, artist_name) %>%
  summarize(mean.dan = mean(danceability),
            mean.ac = mean(acousticness))
```

Using the Spotify API

Comparing the artists

```
p <- ggplot(both, aes(x=album_release_year, y=danceability, group = art
p + geom_point(alpha=0.1) + geom_smooth() +
  labs(title="Comparing danceability", caption = "Data from collect fro
  xlab("") + ylab("Mean danceability") + theme_bw()
```

Using the Spotify API

Comparing the artists

```
p <- ggplot(both, aes(x=album_release_year, y=acousticness, group = art
p + geom_point(alpha=0.1) + geom_smooth() +
  labs(title="Comparing acousticness", caption = "Data from collect fro
  xlab("") + ylab("Mean acousticness") + theme_bw()
```

Using the Spotify API

Collecting more data

Let's try another type of query.

```
## # A tibble: 10 x 4
##    id                     name          popularity followers.total
##    <chr>                  <chr>              <int>           <int>
##  1 3TVXtAsR1Inumwj472S9r4 Drake                 96        84268060
##  2 7dGJo4pcD2V6oG8kP0tJRR Eminem                90        80934857
##  3 15UsOTVnJzReFVN1VCnxy4 XXXTENTACION          85        43821591
##  4 0hCNtLu0JehylgoiP8L4Gh Nicki Minaj           88        30502691
##  5 0Y5tJX1MQlPlqiwlOH1tJY Travis Scott          93        26891611
##  6 2YZyLoL8N0Wb9xBt1NhZWg Kendrick Lamar        88        26772228
##  7 5K4W6rqBFWDnAN6FQUkS6x Kanye West           91        23370609
##  8 6l3HvQ5sa6mXTsMTB19rO5 J. Cole               86        22596040
##  9 1URnnhqYAYcrqrcwql10ft 21 Savage             93        16963519
## 10 4O15NlyKLIASxsJ0PrXPfz Lil Uzi Vert          85        16611911
## # A tibble: 10 x 2
##    name           followers.total
##    <chr>                    <int>
##  1 Drake                 84268060
```

Using the Spotify API

Programming complex queries

Now we have a list of artists, let's use this information as input for another query.

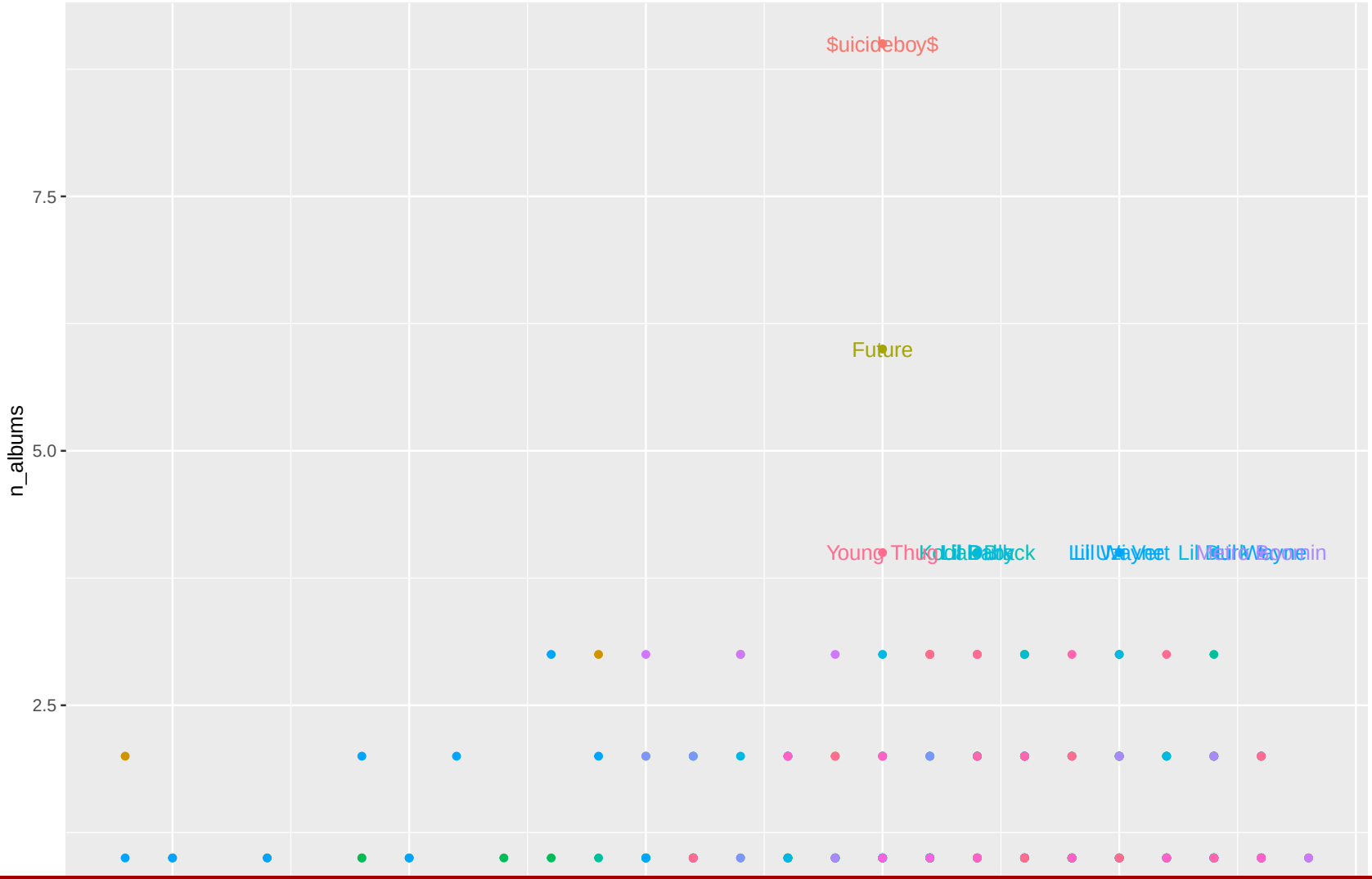
Using the Spotify API

Creating a summary

Let's count the number of albums each artist released each year. Why is `n_distinct` useful here?

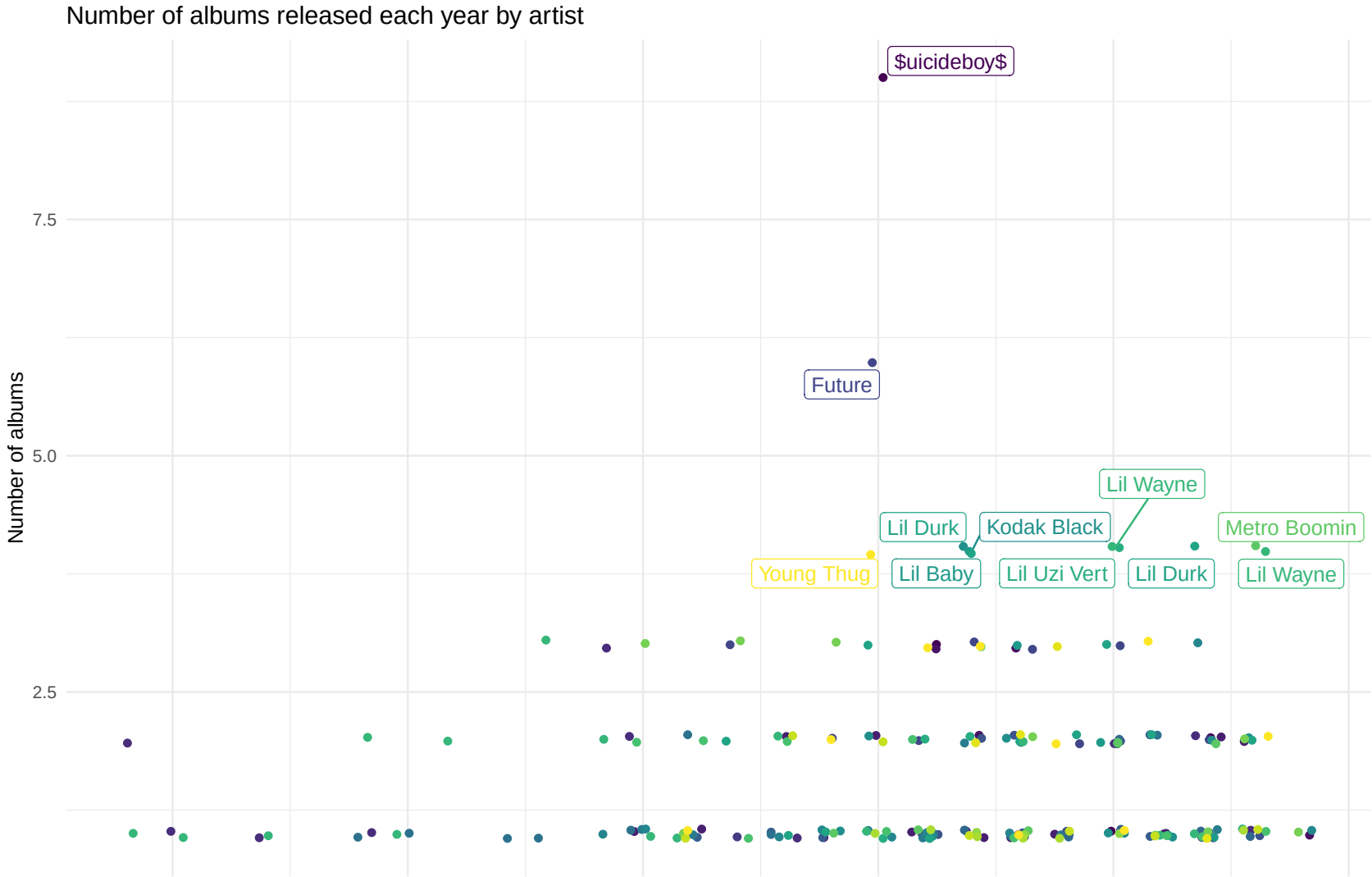
Using the Spotify API

Visualizing the data



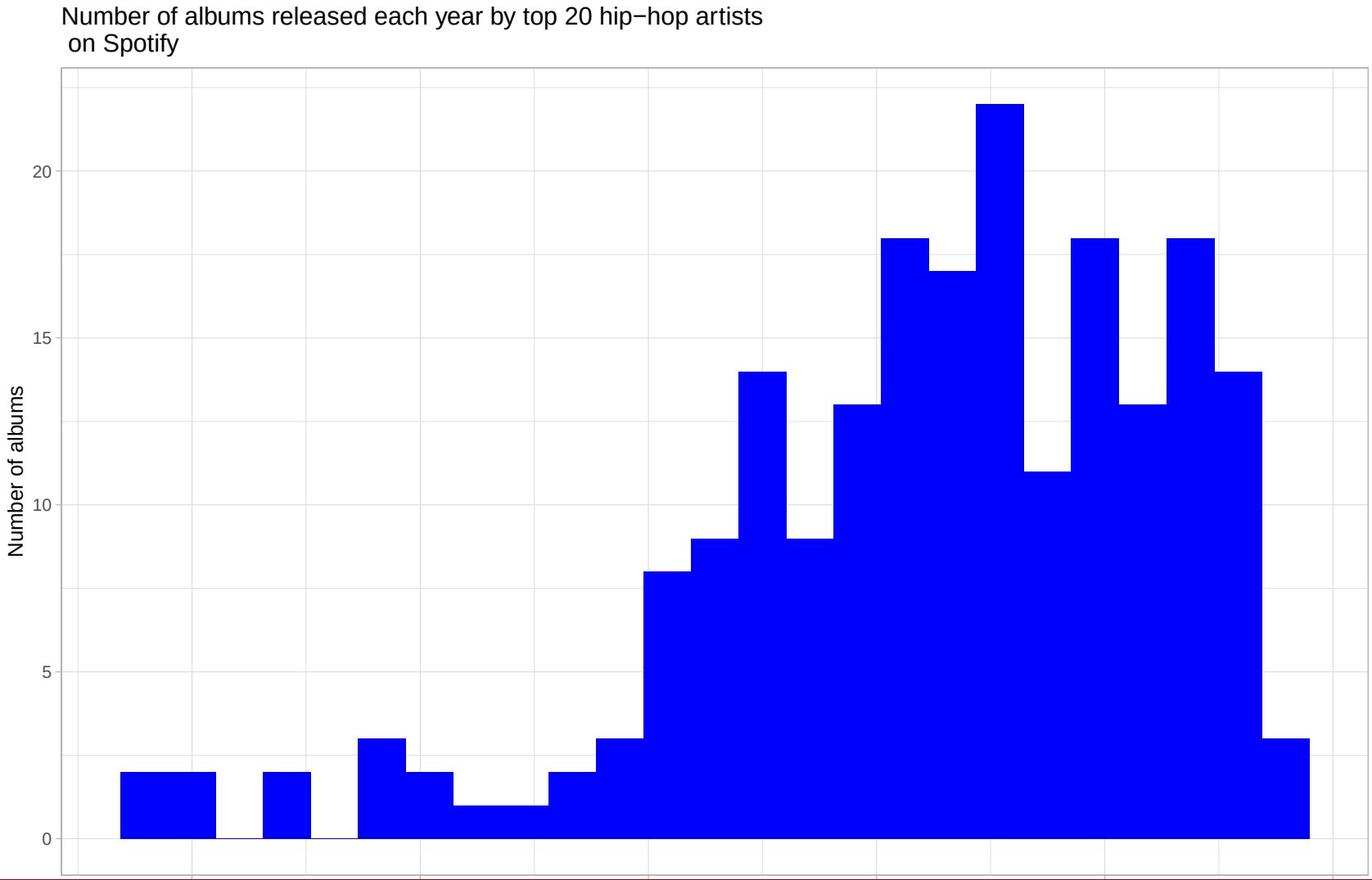
Using the Spotify API

Improving the visualization



Using the Spotify API

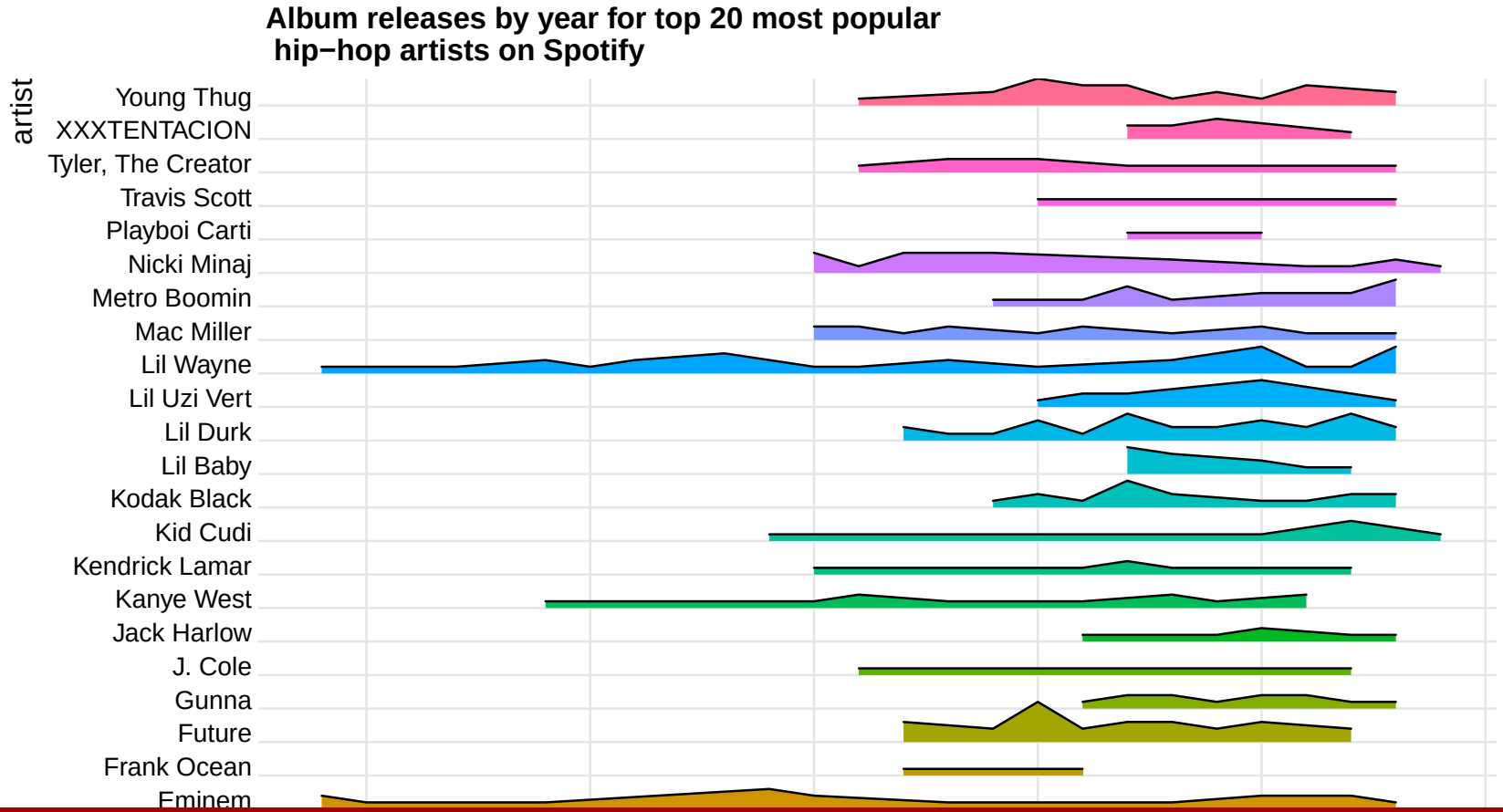
Creating a histogram



Using the Spotify API

More advanced visualizations

There are other extensions of `ggplot` that can create even more sophisticated plots. The `ggridges` package allows us to represent multiple artists' trends as overlaid histograms.



Using the Spotify API

Exercise

1. Use the Spotify API to collect your own data.
2. Use tidyverse functions to select relevant columns and summarize the data
3. Make a plot to visualize the data

Using the Spotify API

Exercise

Collect the data here. Use the `spotifyr` documentation to view functions.

Using the Spotify API

Exercise

Use `ggplot` to make a plot showing the data you retrieved.

Using the Spotify API

Accessing your personal data

- Some features require more setup and authentication
  - You can only use these features if you have set `http://localhost:1410/` in Redirect URIs and authorized your app
  - This tells the API to open up authentication on port 1410 of your computer
  - Note: You may need to install the package `httpuv` for this to work

Using the Spotify API

Finding your recently played tracks

To access your personal data, you can run this code to look at your most recently played tracks. There are many other functions you can use to get and even modify your own data (so use these carefully!). You will have to type 1 into the console after running the chunk and may need to approve access in your browser. Note how we need to request additional authorization for this action.

```
recents <- get_my_recently_played(limit = 10,
                                  authorization = get_spotify_authorization_code(s
```

Example from the `spotifyr` documentation.

Using the Spotify API

Inspecting the results

```
recents %>% mutate(artist.name = map_chr(track.artists, function(x) x$n
           played_at = as_datetime(played_at)) %>%
    select(track.name, artist.name, track.album.name, played_at) %>% as
```

APIs

Best-practices

- Use a wrapper package if available
  - Although sometimes you will have to write your own queries
- Build functions to obey rate-limits where possible
- Access only the data you need
- Test using small examples before collecting a larger dataset

Summary

- Application programming interfaces provide programmatic access to data stored on websites and social media platforms, making them an ideal source of digital trace data for social scientific research
- APIs can be queried using web requests or custom R packages, making them relatively easy to use
- But major social media platforms have cut back access to APIs and smaller websites do not have them

APIs

The post-API age?

- Access on major social media platforms
  - Limited Facebook Pages/Groups/Ads and Instagram data available and new academic intiatives in the works*
  - Twitter Academic API shut down after Musk takeover, current API too expensive for most researchers
  - Reddit had a permissive system but cut access to monetize data for large language model training
  - Restrictive conditions for TikTok academic access*

*These APIs require application processes and approval to access.

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

Next week

- Collecting data from websites using webscraping