# Computational Social Science

## Observational Studies and Application Programming Interfaces II

Dr. Thomas Davidson

Rutgers University

February 8, 2024

# Plan

- Recap on APIs
- Using the Spotify API in R
- Exercise

# Recap

- Online data sources for social science
  - Big data, observational data, digital trace data
- Application Programming Interfaces allow us to easily collect these kinds of data
  - API queries
  - JSON data
  - Rate-limiting
- Interacting with the Github API in R

# Using the Spotify API

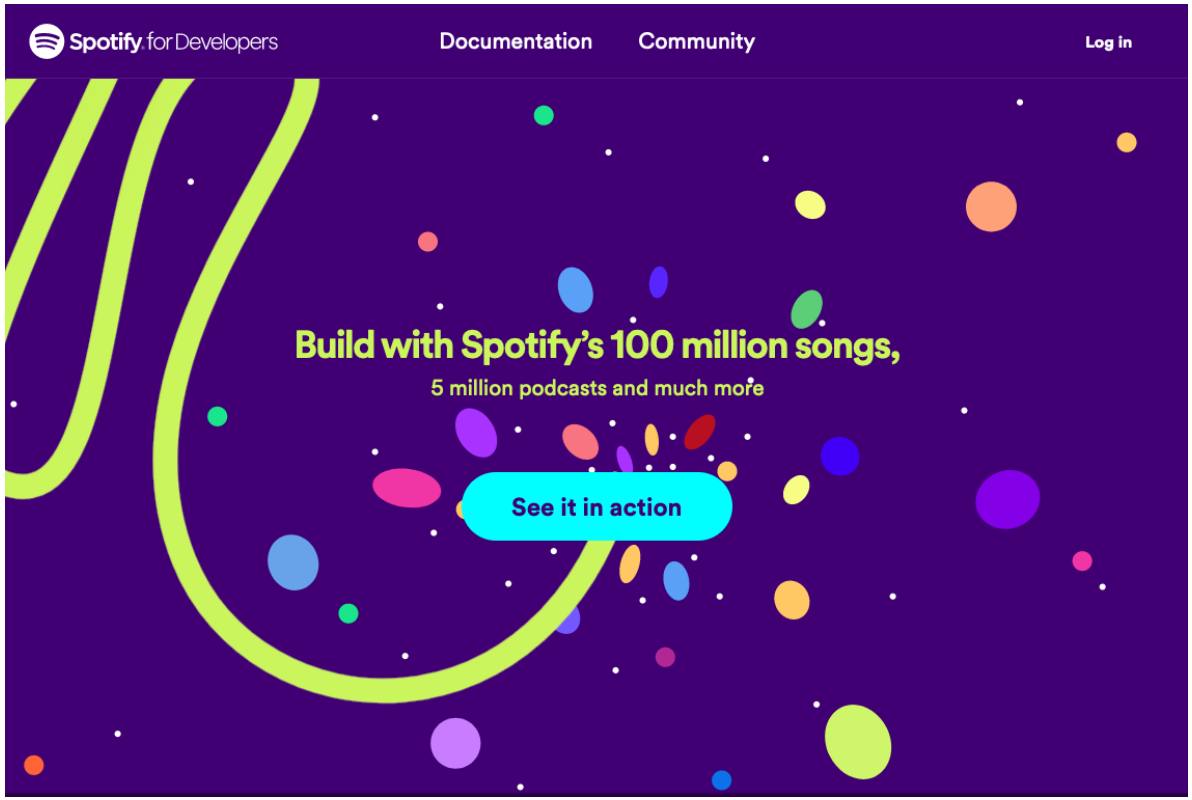
## Documentation

- 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

# 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

# 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 from Spotify API") +
  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 from Spotify API")  +
    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 = artist_name, color = artist_name))
p + geom_point(alpha=0.1) + geom_smooth() +
  labs(title="Comparing danceability",
       caption = "Data from collect from Spotify API") +
  xlab("") + ylab("Mean danceability") + theme_bw()
```

# Using the Spotify API

## Comparing the artists

```
p <- ggplot(both, aes(x=album_release_year, y=acousticness,
                      group = artist_name, color = artist_name))
p + geom_point(alpha=0.1) + geom_smooth() +
  labs(title="Comparing acousticness",
       caption = "Data from collect from Spotify API") +
  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
```

# Using the Spotify API

### Programming complex queries

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

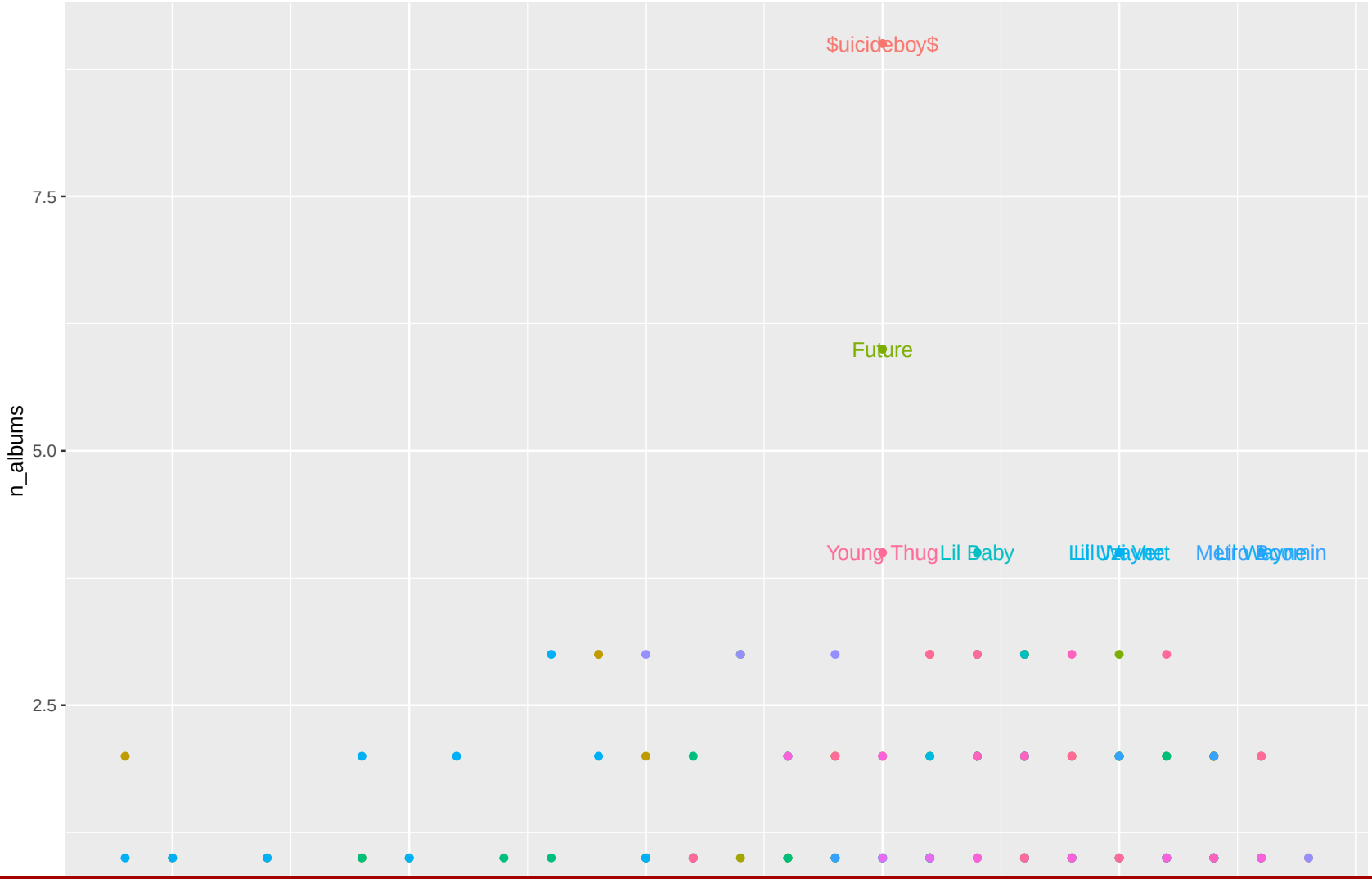
# Using the Spotify API

## Creating a summary

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

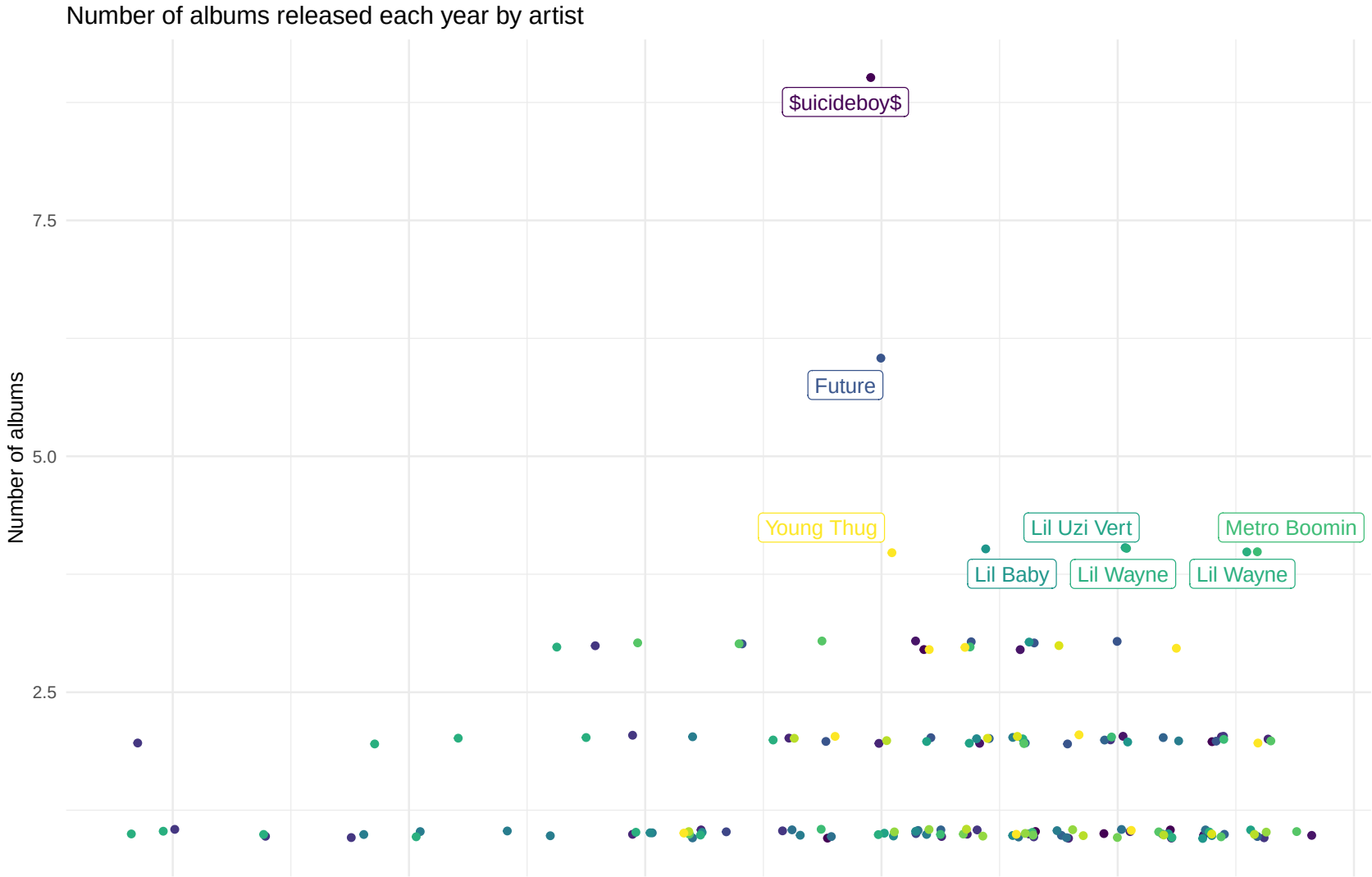
# Using the Spotify API

## Visualizing the data



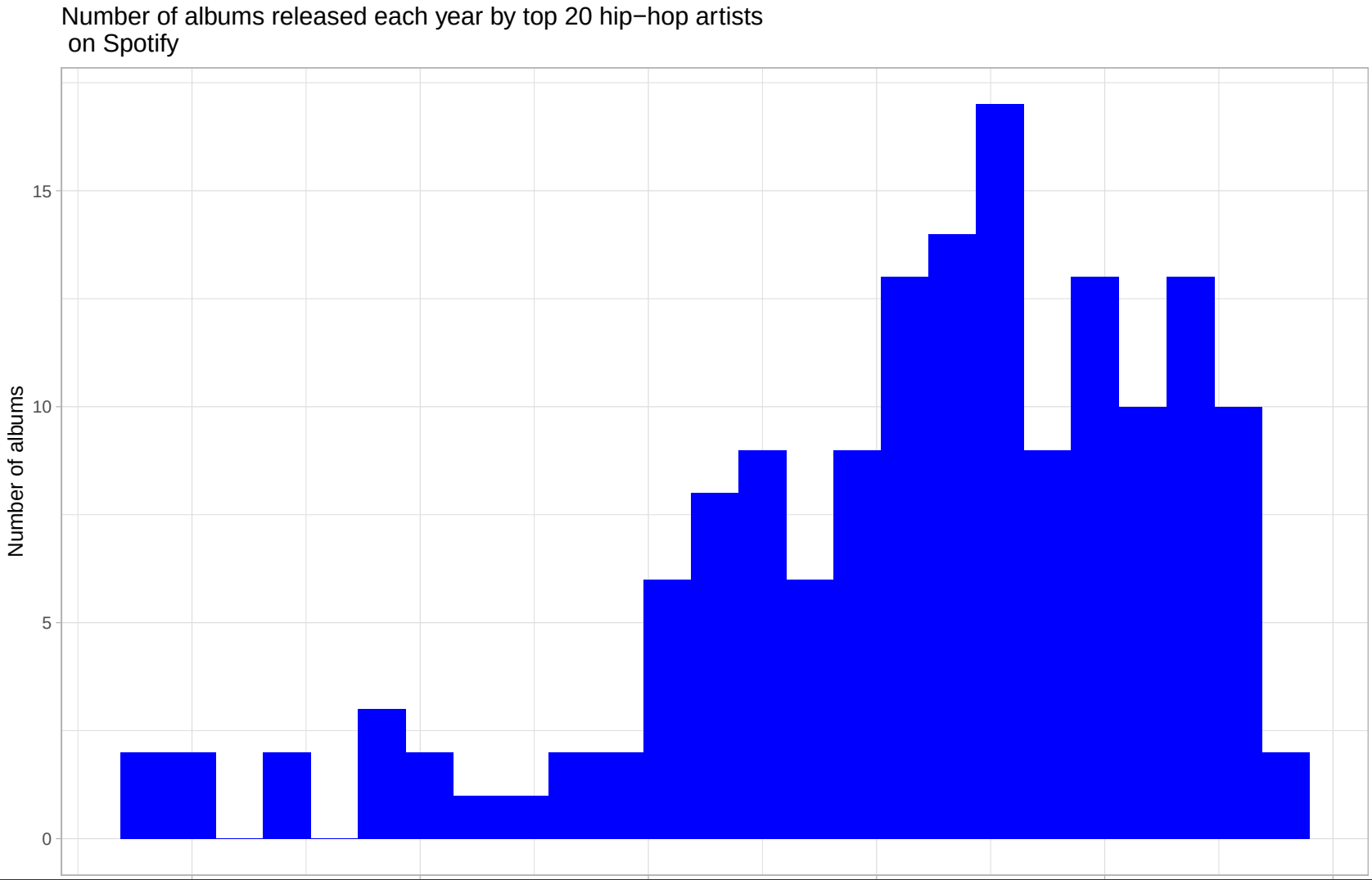
# Using the Spotify API

## Improving the visualization



# Using the Spotify API

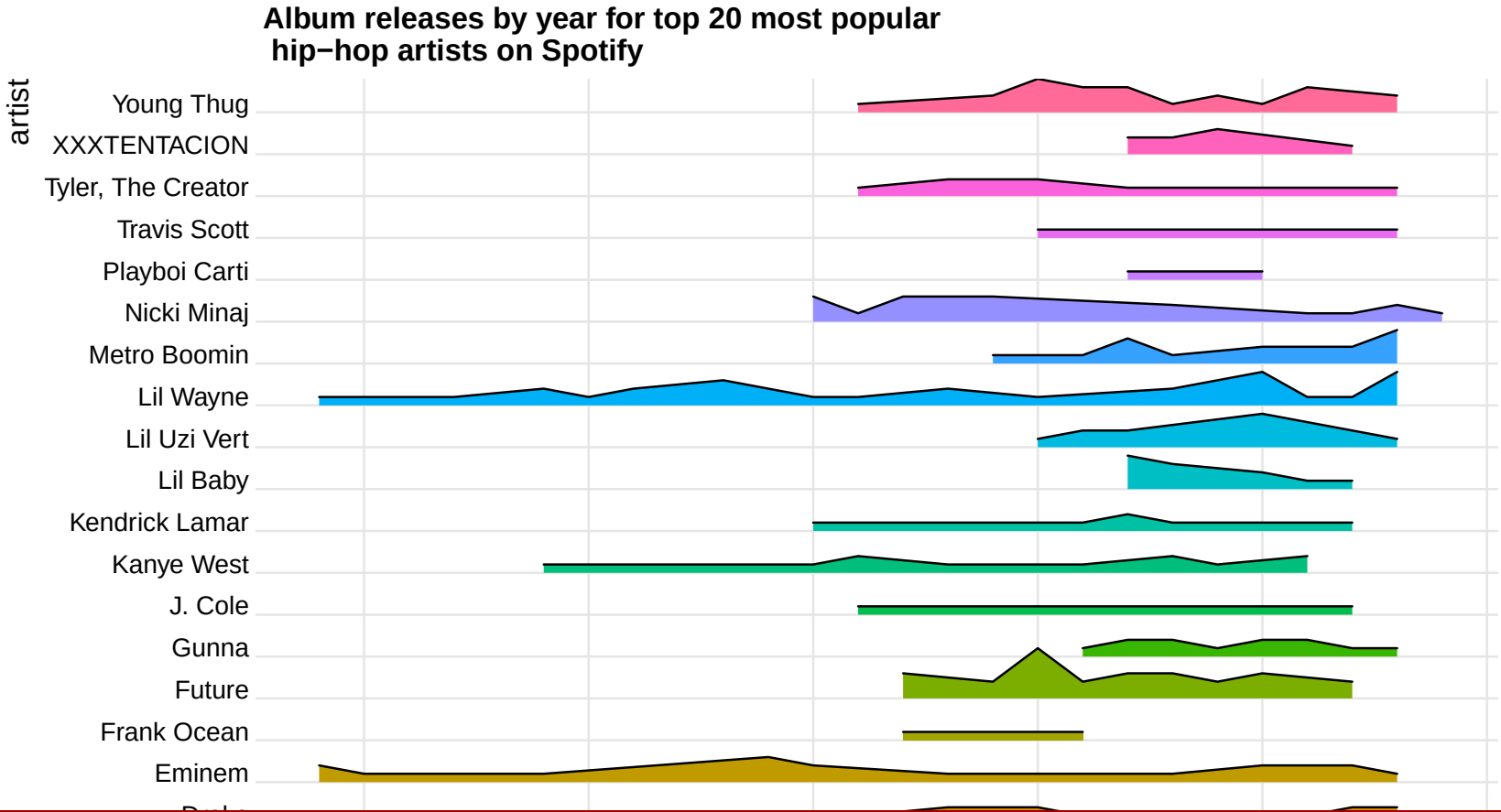
## Creating a histogram



# 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 (as necessary)
3. Product a plot using ggplot.
4. Share the plot in this Google Doc: https://bit.ly/3rAG7Uk

# Using the Spotify API

## Exercise

# 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(scope = 'user-read-r
```

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
```

# 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

# Next week

- Collecting data from websites using webscraping