Social Data Science

Observational Studies and Application Programming Interfaces II

Dr. Thomas Davidson

Rutgers University

September 22, 2021

Plan

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

Course updates

Homework

- Homework due Friday at 5pm ET.
  - Please push your final version to Github with the appropriate commit message
  - Office hours today 4:30-5:30pm, 109 Davison Hall

Recap

APIs

- 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

APIs

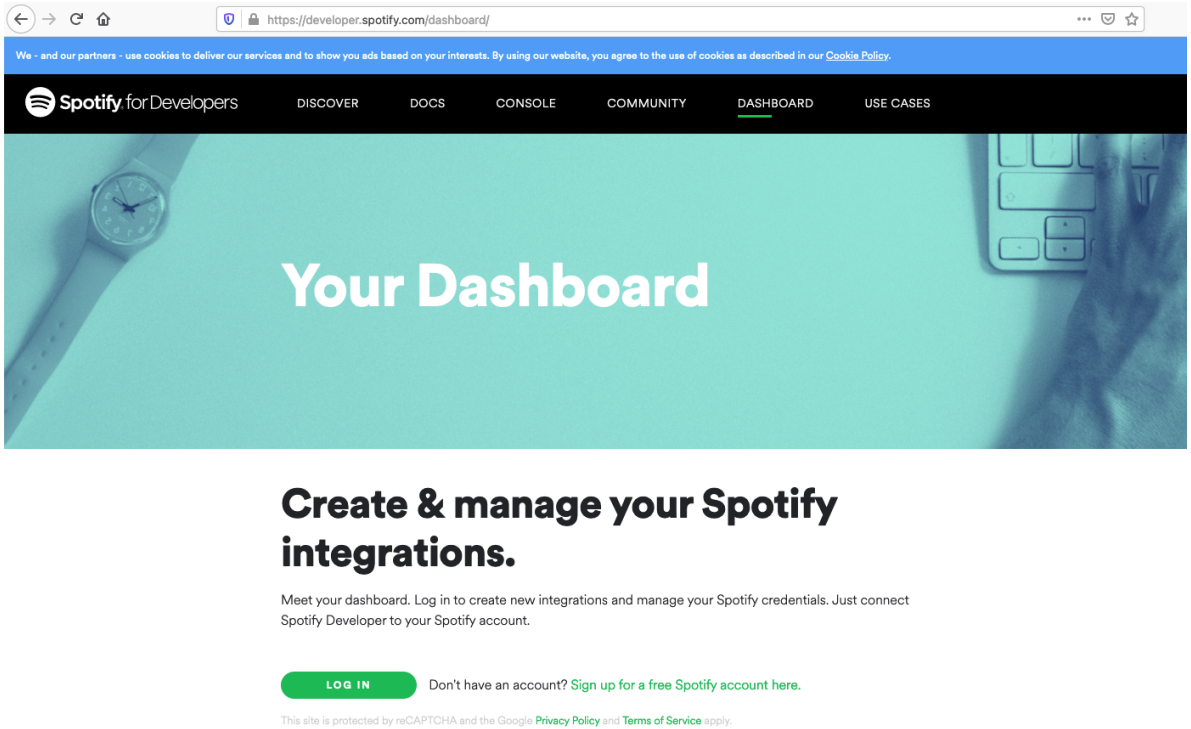
Using the Spotify API

- 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

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



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.

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

You can read more about the library here.

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. 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)
# print(get_artists)
```

APIs

Using the Spotify 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.

```
stones <- get_artist_audio_features("The Rolling Stones") %>% as_tibble
head(stones)
```

```
## # A tibble: 6 x 39
##   artist_name       artist_id album_id album_type album_images albu
##   <chr>             <chr>     <chr>    <chr>      <list>       <chr
## 1 The Rolling Stones 22bE4uQ6~ 7fvR1o3~ album      <df [3 x 3]> 2021
## 2 The Rolling Stones 22bE4uQ6~ 7fvR1o3~ album      <df [3 x 3]> 2021
## 3 The Rolling Stones 22bE4uQ6~ 7fvR1o3~ album      <df [3 x 3]> 2021
## 4 The Rolling Stones 22bE4uQ6~ 7fvR1o3~ album      <df [3 x 3]> 2021
## 5 The Rolling Stones 22bE4uQ6~ 7fvR1o3~ album      <df [3 x 3]> 2021
## 6 The Rolling Stones 22bE4uQ6~ 7fvR1o3~ album      <df [3 x 3]> 2021
## # ... with 33 more variables: album_release_year <dbl>,
## #   album_release_date_precision <chr>, danceability <dbl>, energy <
## #   key <int>, loudness <dbl>, mode <int>, speechiness <dbl>,
## #   acousticness <dbl>, instrumentalness <dbl>, liveness <dbl>, vale
```

APIs

Using the Spotify API

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

```
##  [1] "Intro - Live"
##  [2] "Jumpin' Jack Flash - Live"
##  [3] "It's Only Rock 'N' Roll (But I Like It) - Live"
##  [4] "You Got Me Rocking - Live"
##  [5] "Tumbling Dice - Live"
##  [6] "Oh No, Not You Again - Live"
##  [7] "Wild Horses - Live"
##  [8] "Rain Fall Down - Live"
##  [9] "Midnight Rambler - Live"
## [10] "Night Time Is The Right Time - Live"
```

APIs

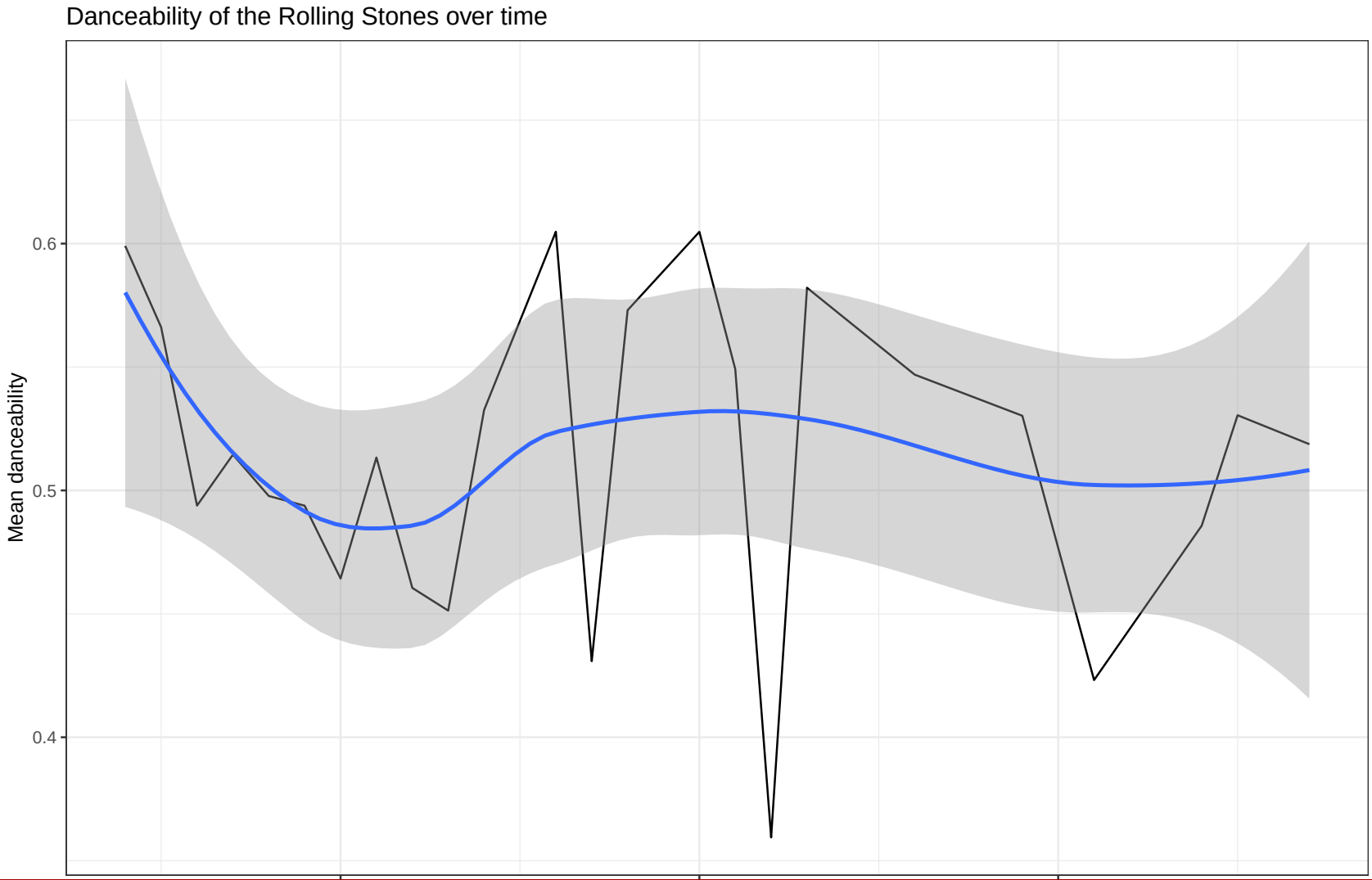
Using the Spotify API

Let's calculate some statistics using this table.

```
results <- stones %>% filter(album_release_year < 2000) %>%
             group_by(album_release_year) %>%
  summarize(mean.dan = mean(danceability),
            mean.ac = mean(acousticness))
```

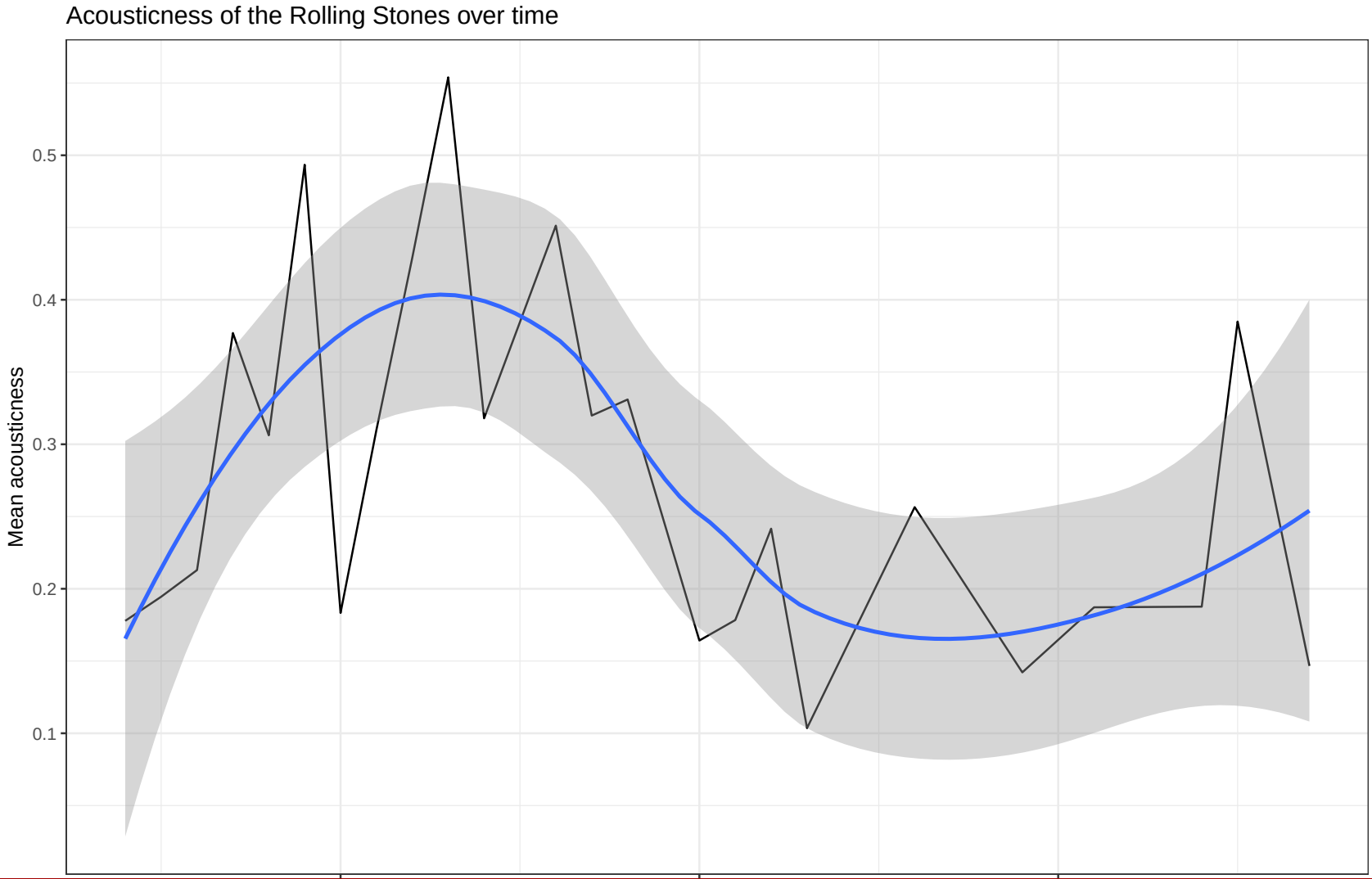
APIs

Using the Spotify API



APIs

Using the Spotify API



APIs

Using the Spotify API

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

```
pf <- get_artist_audio_features("Pink Floyd") %>% as_tibble()
both <- bind_rows(stones, pf) # adding Rolling Stones to the same tibbl

head(both)
```

```
## # A tibble: 6 x 39
##   artist_name       artist_id album_id album_type album_images albu
##   <chr>             <chr>     <chr>    <chr>      <list>       <chr
## 1 The Rolling Stones 22bE4uQ6~ 7fvR1o3~ album      <df [3 x 3]> 2021
## 2 The Rolling Stones 22bE4uQ6~ 7fvR1o3~ album      <df [3 x 3]> 2021
## 3 The Rolling Stones 22bE4uQ6~ 7fvR1o3~ album      <df [3 x 3]> 2021
## 4 The Rolling Stones 22bE4uQ6~ 7fvR1o3~ album      <df [3 x 3]> 2021
## 5 The Rolling Stones 22bE4uQ6~ 7fvR1o3~ album      <df [3 x 3]> 2021
## 6 The Rolling Stones 22bE4uQ6~ 7fvR1o3~ album      <df [3 x 3]> 2021
## # ... with 33 more variables: album_release_year <dbl>,
## #   album_release_date_precision <chr>, danceability <dbl>, energy <
## #   key <int>, loudness <dbl>, mode <int>, speechiness <dbl>,
```

APIs

Using the Spotify API

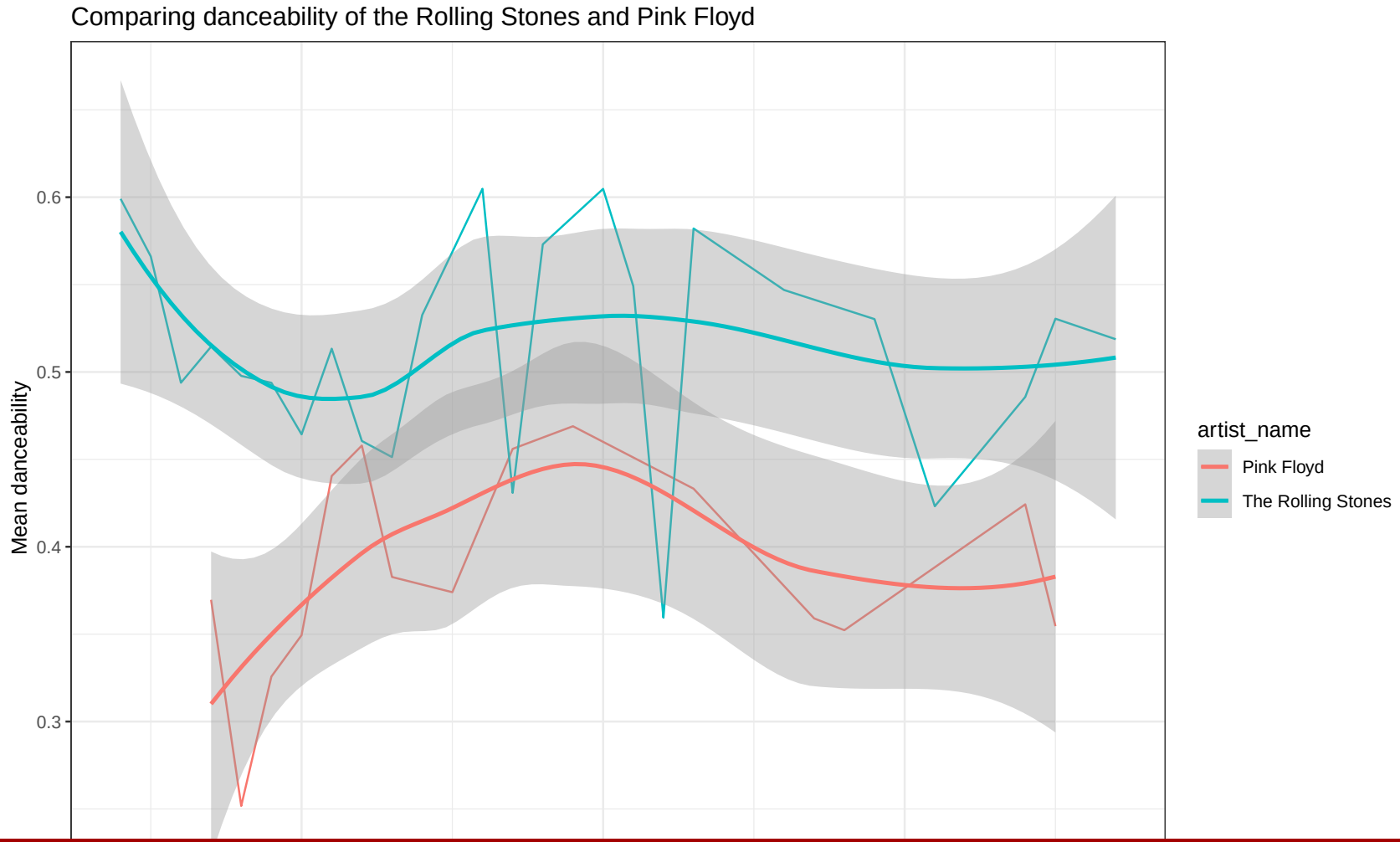
Repeating the summary operation for both artists. Note how we now group by `artist_name` in addition to `album_release_year`.

```
r <- both %>% filter(album_release_year < 2000) %>%
  group_by(album_release_year, artist_name) %>%
  summarize(mean.dan = mean(danceability),
            mean.ac = mean(acousticness)) # note also keeping artist na
#View(r)
```

APIs

Using the Spotify API

Let's compare their danceability.



APIs

Using the Spotify API

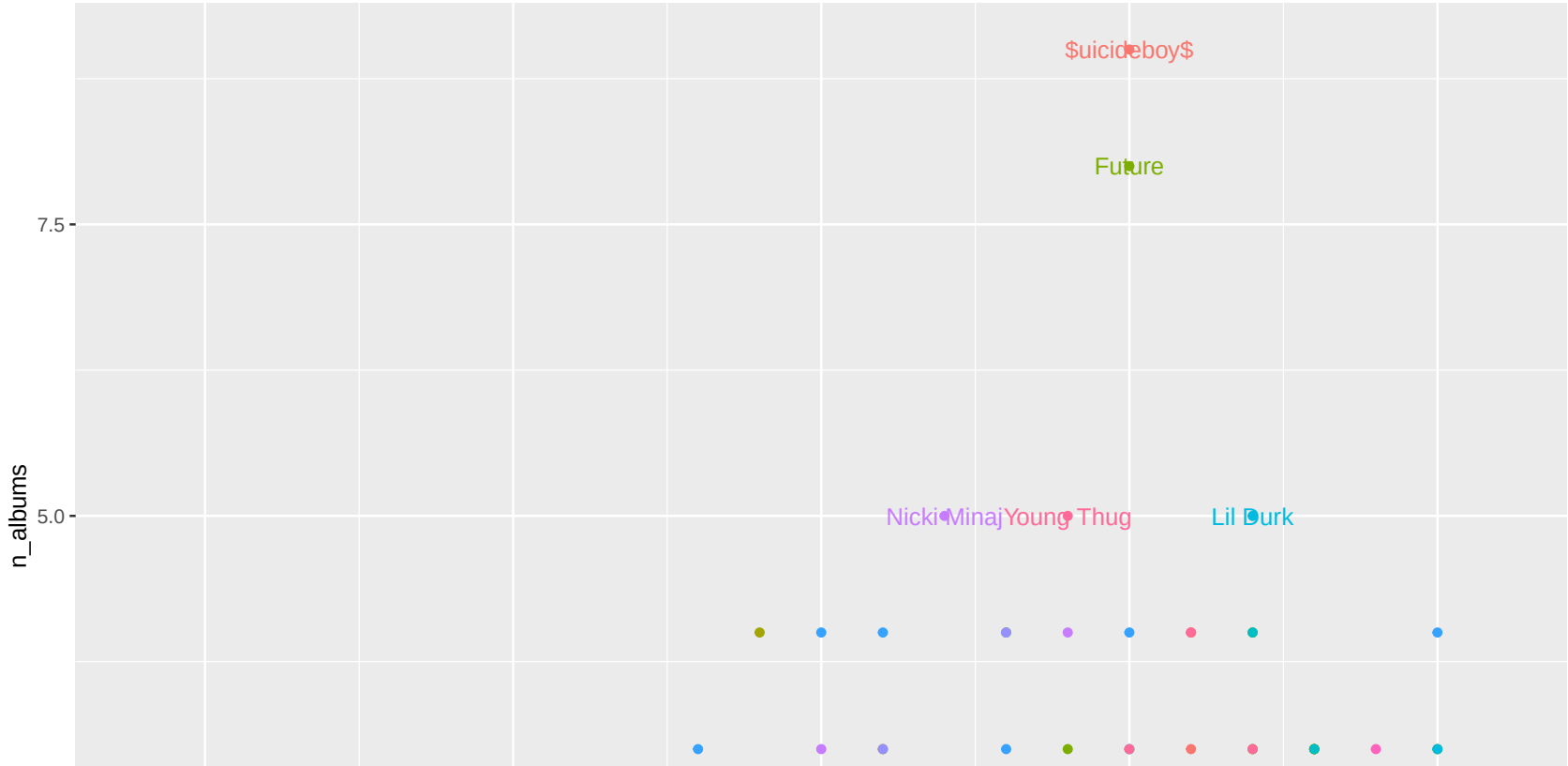
Now we have a list of artists, let's use this information as input for another query.

APIs

Using the Spotify API

Let's create a summary of the data. In this case, let's count the number of albums each artist released each year. Why is `n_distinct` useful here?

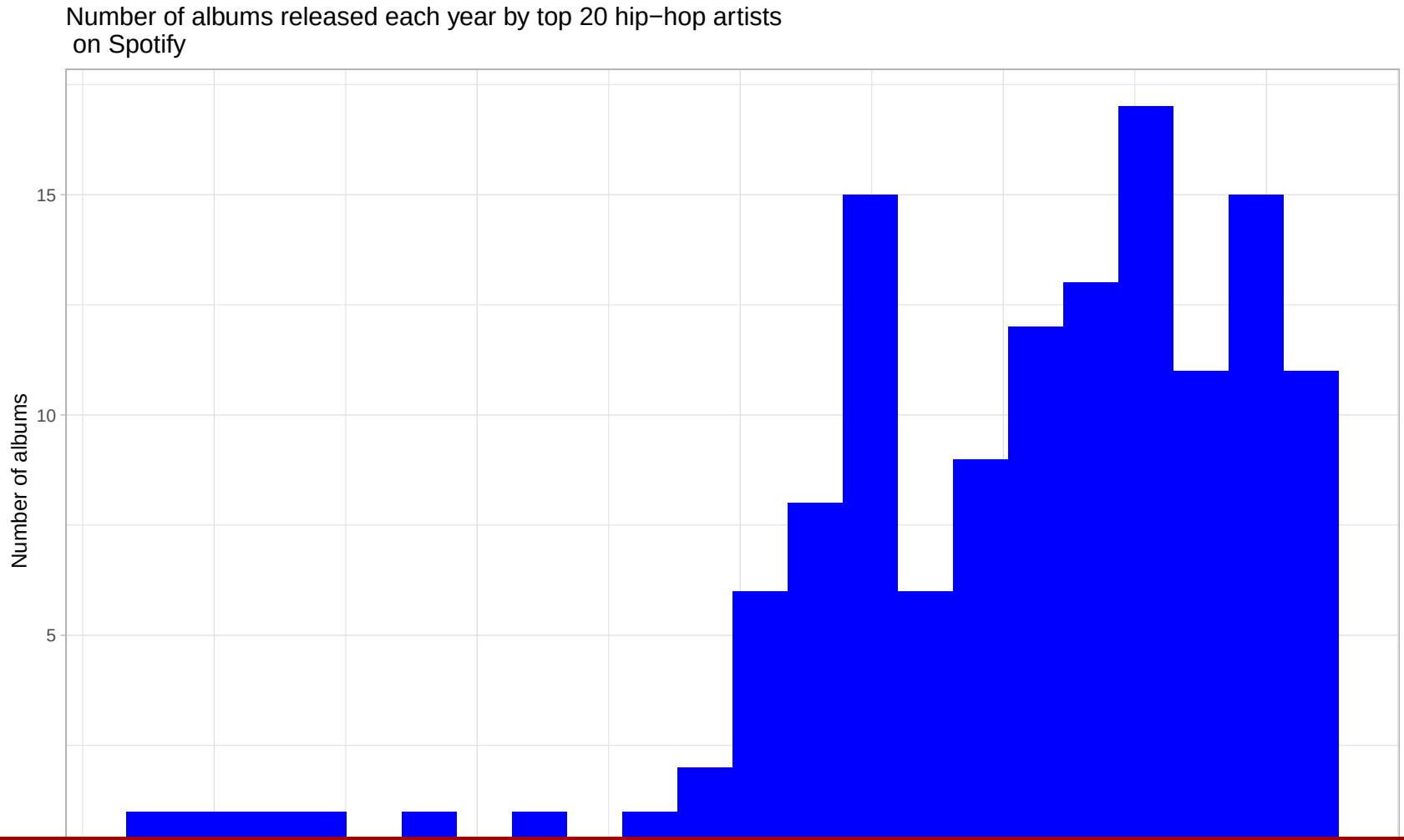
APIs ## Using the Spotify API We can represent these data using a scatterplot.



APIs

Using the Spotify API

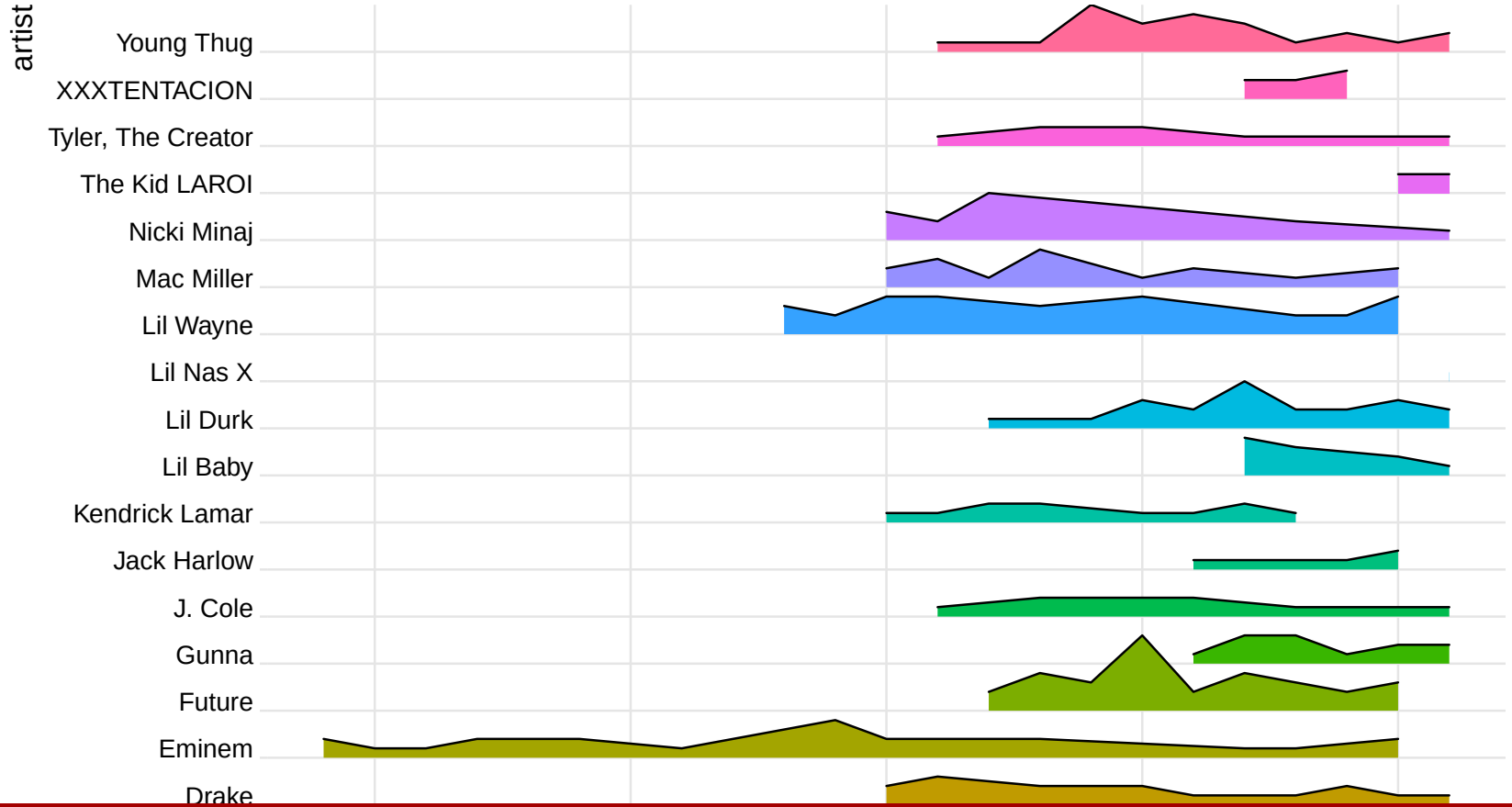
We could also plot the overall values using a histogram.



APIs

Using the Spotify API

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.



APIs

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://cutt.ly/gEkaYoy

APIs

Exercise

APIs

Accessing your Spotify information

- 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

Accessing your Spotify information

```
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.

Questions?