Working with APIs in R

Francesca Giannetti

3/1/2021

About

This workshop explores the New York Times Article API and the Spotify Web API, together with a few R packages designed to help gather and manipulate data from these APIs. This will be an experimental workshop focused on data collection.

Account Creation

First things first. We will need to set up personal developer accounts to interact with both APIs.

Go to https://developer.nytimes.com/signup to create an account and get an API Key. Notice that the New York Times offers several APIs, including Article Search, Archive, Books, Comments, Movie Reviews, Top Stories, and more. Create an app for your research. In your app, be sure to enable the Article Search API. You may elect to enable others in the same app.

Next, go to https://developer.spotify.com/dashboard/login and log into your account (create one if you don't have one yet). In your dashboard, create a project for your research. For certain functions, you will need to authenticate as a Spotify user. To do this, your Spotify Developer application must have a callback URL. A good default option is http://localhost:1410/.

```
## set your NYT and Spotify keys as system variables to access during our session
## replace the stuff in quotes with your personal key

Sys.setenv(NYTIMES_KEY="YOUR_NYT_API_KEY_HERE")

## alternately, you can save your key to file and access it when needed with the `load()` function

nyt_apikey <- "YOUR_NYT_API_KEY_HERE"
save(nyt_apikey, file = "nytimes_apikey")

## now, the same for Spotify

spotify_id <- "YOUR_SPOTIFY_CLIENT_ID"
spotify_secret <- "YOUR_SPOTIFY_CLIENT_SECRET"
Sys.setenv(SPOTIFY_CLIENT_ID = spotify_id)
Sys.setenv(SPOTIFY_CLIENT_SECRET = spotify_secret)
access_token <- get_spotify_access_token()

saveRDS(access_token, "spotify_token.rds")
```

NYT Article Search API

The New York Times APIs are fairly well documented. I find this slide deck from James Boehmer to be extremely helpful for queries to the Article Search API.

To experiment with queries before running them in R, I recommend using the "Try this API" feature of the Developers website. Under APIs, go to Article Search. Press "Authorize" (upper right) to authenticate your connection to the API. And now let's test some queries.

As a first example, let's keep it simple. Set the `q` value to 'tiktok' (it is not case sensitive) and press "EXECUTE." The response is in JSON. Note how many hits you get and observe the different document types you retrieve.

Now, let's try using some of the filter query parameters. Leave the `q` value set to 'tiktok' and this time set `facet` to `true`, `facet_fields` to `document_type`, `facet_filter` to `true`, and `fq` to 'article'. You should go down from 1339 hits to 256 hits.

Develop your own query for Article Search. Use Boehmer's slides as a reminder of the possibilities and syntax: http://jamesboehmer.github.io/nytd2013/#/6. What topics or people would you search on as the basis for a research project?

Kearney's nytimes Package

The R developer community can often be relied upon to develop packages that aid in querying APIs. The NYT APIs are no exception. Let's use Mike Kearney's `nytimes` package to try out one or more of the queries developed above. Kearney gives us examples of the functions and syntax here: https://github.com/mkearney/nytimes.

```
# read your NYT api key back into memory, if needed
load("nytimes_apikey")

## get http response objects for search about the Nobel Prize in literature
nobel1 <- nyt_search("nobel prize literature", n=100)

## convert response object to data frame
nobel1.df <- as.data.frame(nobel1)

## preview data
head(nobel1.df, 10)
```

Alternately, jsonlite

Kearney's package is a great way to get started. But you'll notice that this package doesn't allow us to make use of very many parameters. Let's try to get a bit closer to the NYT API's JSON response by using the `jsonlite` package. Here, I'll borrow some code from https://www.storybench.org/working-with-the-new-york-times-api-in-r/.

```
# read your NYT api key back into memory, if needed
# load("nytimes_apikey")

# let's start with a simple query
# insert your API key directly into the URL where indicated

morrison <- fromJSON(paste0("http://api.nytimes.com/svc/search/v2/articlesearch.json?q=toni+morrison&ap
```

Let's have a look at our `morrison` object. You'll notice that it only returns 10 results at a time. We can get more by creating a for loop.

```
# read your NYT api key back into memory, if needed
load("nytimes_apikey")

# Let's set some parameters
term <- "nobel+prize+literature" # Need to use + to string together separate words
begin_date <- "19990101" # Let's look at a decade of matches
end_date <- "20090101"

baseurl <- paste0("http://api.nytimes.com/svc/search/v2/articlesearch.json?q=",term,
                  "&begin_date=",begin_date,"&end_date=",end_date,
                  "&facet_filter=true&api-key=",nyt_apikey, sep="")

initialQuery <- fromJSON(baseurl)
maxPages <- round((initialQuery$response$meta$hits[1] / 10)-1)

nobel_pages <- vector("list",length=maxPages)

for(i in 0:maxPages){
  nytSearch <- fromJSON(paste0(baseurl, "&page=", i), flatten = TRUE) %>% data.frame()
  nobel_pages[[i+1]] <- nytSearch
  Sys.sleep(6) # wait 6 seconds between calls
}

nobel_articles <- rbind_pages(nobel_pages)

nobel_articles[1:10,21] # looking at headline for the first 10 rows
```

A few exploratory visualizations

```
# Visualize coverage by section
p1 <- nobel_articles %>%
  group_by(response.docs.type_of_material) %>%
  summarize(count=n()) %>%
  mutate(percent = (count / sum(count))*100) %>%
  arrange(desc(percent)) %>%
  filter(!is.na(response.docs.type_of_material)) %>%
  top_n(10, percent) %>%
  ggplot(aes(x=reorder(response.docs.type_of_material, percent), y=percent, fill = response.docs.type_o
  geom_bar(stat = "identity") +
  labs(x = "Type of Material by Percent", y = NULL) +
  guides(fill=guide_legend(title=NULL)) +
  theme_minimal() +
  coord_flip()

p1

# years with the most coverage
# so, what happens in 2006?
p2 <- nobel_articles %>%
  mutate(pubDay = gsub("T.*","",response.docs.pub_date),
```

```
        pubYear = lubridate::year(pubDay)) %>%
  group_by(pubYear) %>%
  summarise(count=n()) %>%
  ggplot(aes(x=reorder(pubYear, count), y=count)) +
  geom_bar(stat="identity") +
  labs(x = "Year of Publication", y = "Count", title = "Number of Articles Mentioning 'Nobel Prize Lite
  theme_minimal() +
  coord_flip()

p2
```

Spotify Web API

Spotify's Web API is similarly well documented; see https://developer.spotify.com/documentation/web-api/reference/. Like the New York Times, Spotify also proposes a console where we can test our queries. Select "Console" in Spotify for Developers. Let's try the Search to look up the id of a musical artist. First, you will need to request an Oauth token to authenticate your connection to the web API.

Try this first by pressing the "FILL SAMPLE DATA" button to accept Spotify's suggestions. You can then run this sample query by pressing "TRY IT." Now, let's edit this search. Set the `q` value to "giddy stratospheres" (or some other song you like). Notice the album, artist, and track ids. You can reuse these in subsequent queries.

Let's now try the **Artists > Related Artists** endpoint in the console. You can try this query with the sample data, but I'd recommend looking up the id of an artist of interest to you. Here we can start to observe Spotify's recommendation algorithm in action.

There are a couple of R packages that facilitate queries to the Spotify Web API. We will try `spotifyr`, although `Rspotify` is also worthy of exploration. The documentation for `spotifyr` is available here: https://www.rcharlie.com/spotifyr/. Also by typing `help(package="spotifyr")` at the console below.

```
# read your Spotify credentials back into memory, if needed
readRDS("spotify_token.rds")
```

```
## [1] "BQAXGlocgwWtiFSkb__RsqoOVxZJudyEajqiMdB9Ql7OJ_JdtdtykIdaS7w622CnGeNFcB-plHHw-4nUW_4"
```

```
## a few sample queries using spotifyr

beatles <- get_artist_audio_features('the beatles')

## create a table of the top 5 key signatures
beatles %>%
  count(key_mode, sort = TRUE) %>%
  head(5) %>%
  kable()
```

key_mode	n
D major	24
G major	21
A major	13
F major	12
C major	11

```
## retrieve the top 25 "britpop" artists
artists_by_genres <- get_genre_artists(genre = "britpop", limit = 25)
```

```
## $q
## [1] "genre:\"britpop\""
##
## $type
## [1] "artist"
##
## $market
## NULL
##
## $limit
## [1] 25
##
## $offset
## [1] 0
##
## $access_token
## [1] "BQAP4d_rabzmIDahV9Vc6wVkcOxaTBpEr1np32KIwPc3T05CzaGqM233zmHmvM8mS2WWcNNql46yHmTzNqI"
```

Spotify's audio features

Let's have a closer look at the `beatles` object above. Click on the object in your environment to open the table view. Or type `names(beatles)` at the console. Notice those column headers, e.g. speechiness, valence, tempo, etc. You can read more about how Spotify defines these features here: https://developer.spotify.com/documentation/web-api/reference/#object-audiofeaturesobject.

Valence, for instance, is a measure of musical positivity.

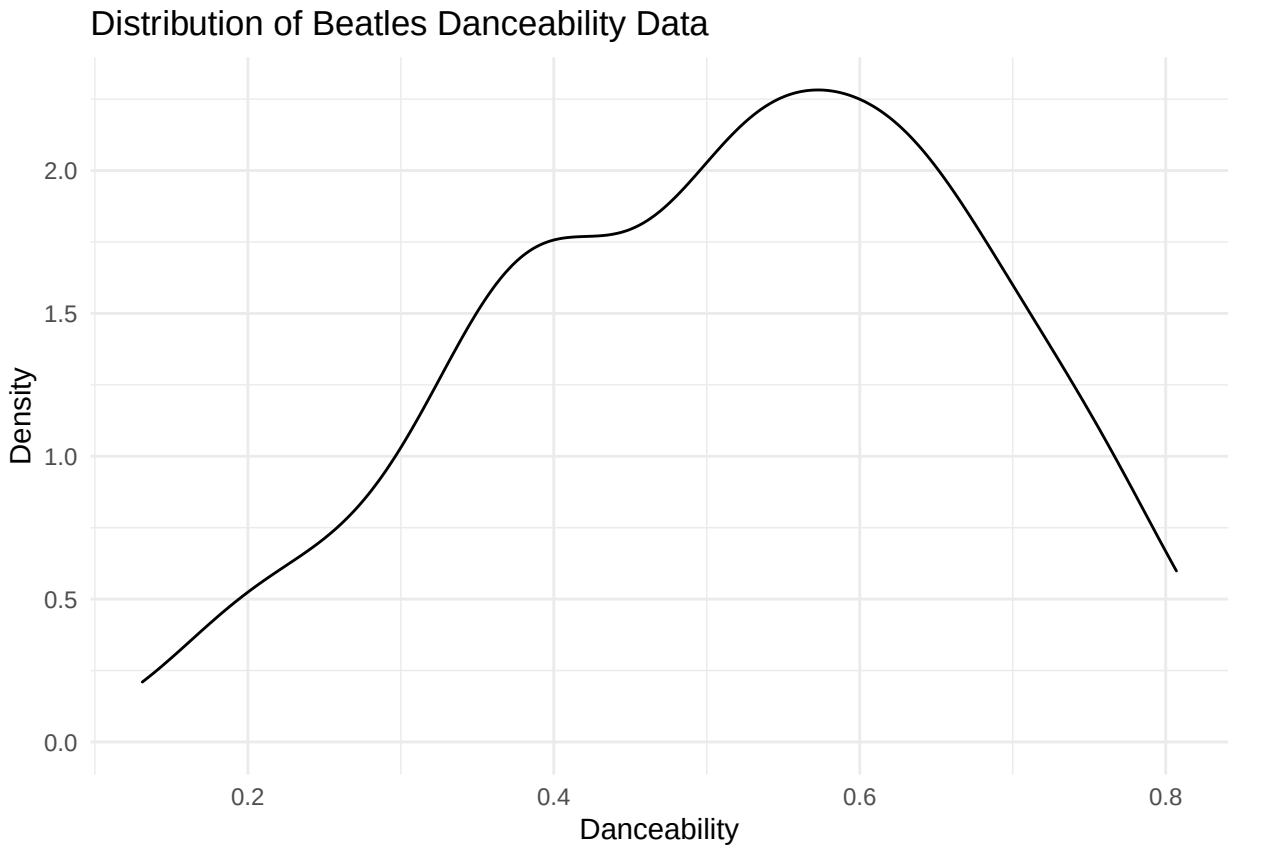
```
# read your Spotify credentials back into memory, if needed
# readRDS("spotify_token.rds")

beatles %>% arrange(-valence) %>%
  select(track_name, valence) %>%
  head(5) %>%
  kable()
```

track_name	valence
Ob-La-Di, Ob-La-Da - 2018 Mix	0.969
Birthday - 2018 Mix	0.964
Polythene Pam - Esher Demo	0.963
Ob-La-Di, Ob-La-Da - Take 3	0.962
Don't Pass Me By - 2018 Mix	0.961

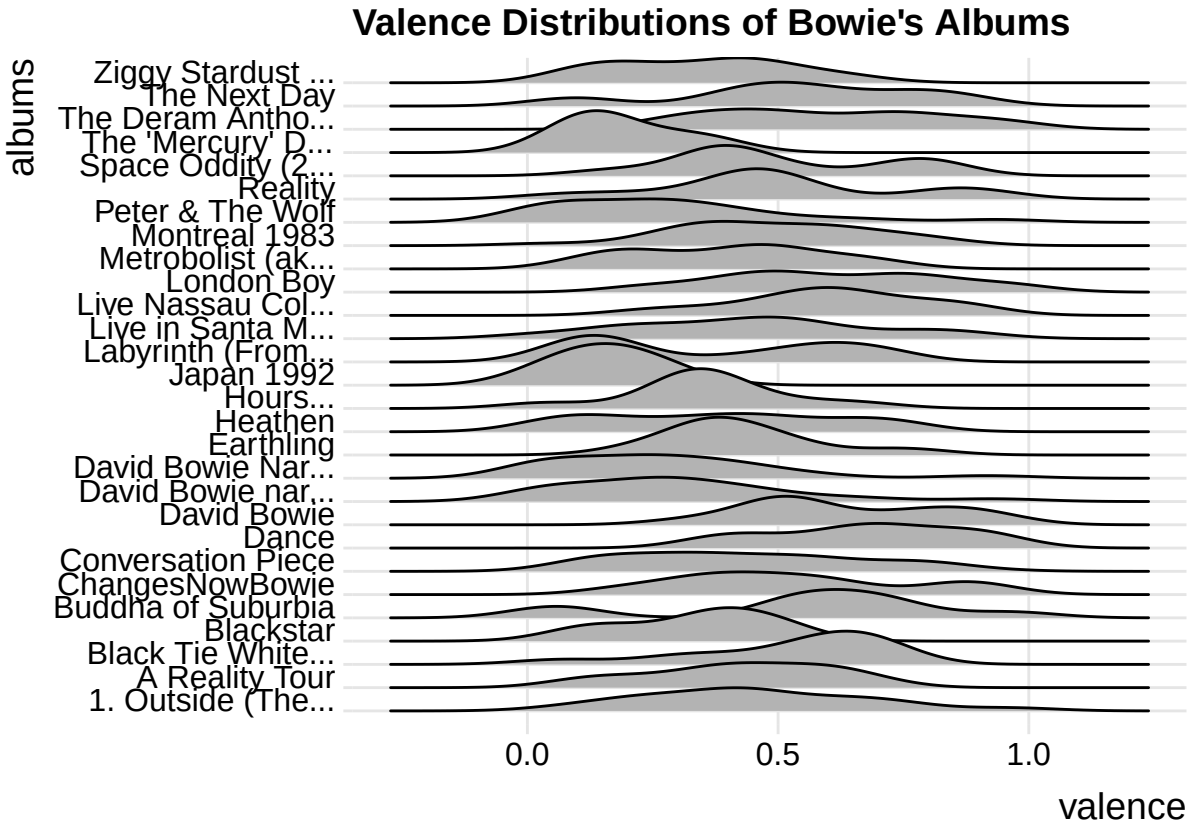
And Spotify's danceability index is based on "tempo, rhythm stability, beat strength, and overall regularity". A value of 0.0 is least danceable and 1.0 is most danceable.

```
p3 <- ggplot(beatles, aes(x=danceability)) +
  geom_density(alpha=0.7) +
  labs(x="Danceability", y="Density") +
  theme_minimal() +
  ggtitle("Distribution of Beatles Danceability Data")
```

```
p3
```




```
## adding an extra package
## install.packages('ggridges')
library(ggridges)

bowie <- get_artist_audio_features("David Bowie")

p4 <- ggplot(bowie, aes(x = valence, y = stringr::str_trunc(album_name, 18))) +
  geom_density_ridges() +
  theme_ridges() +
  labs(y = "albums", x = "valence", title="Valence Distributions of Bowie's Albums")

p4
```



What other questions can we come up with to explore Spotify's audio features?