Computational Social Science

Observational Studies and Application Programming Interfaces II

Dr. Thomas Davidson

Rutgers University

September 25, 2025

Plan

- Recap on APIs
- Using the New York Times 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 APIs in R

Using the NYT API

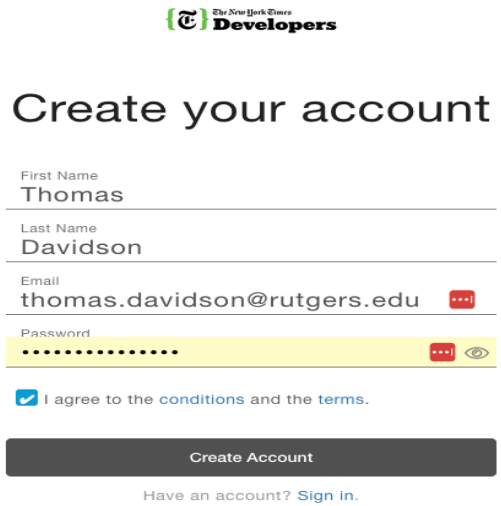
Getting starter

- Begin by following the instructions here to set up an account:
  - https://developer.nytimes.com/get-started
- We'll go through the steps for registering an app

Using the NYT API

Signing up

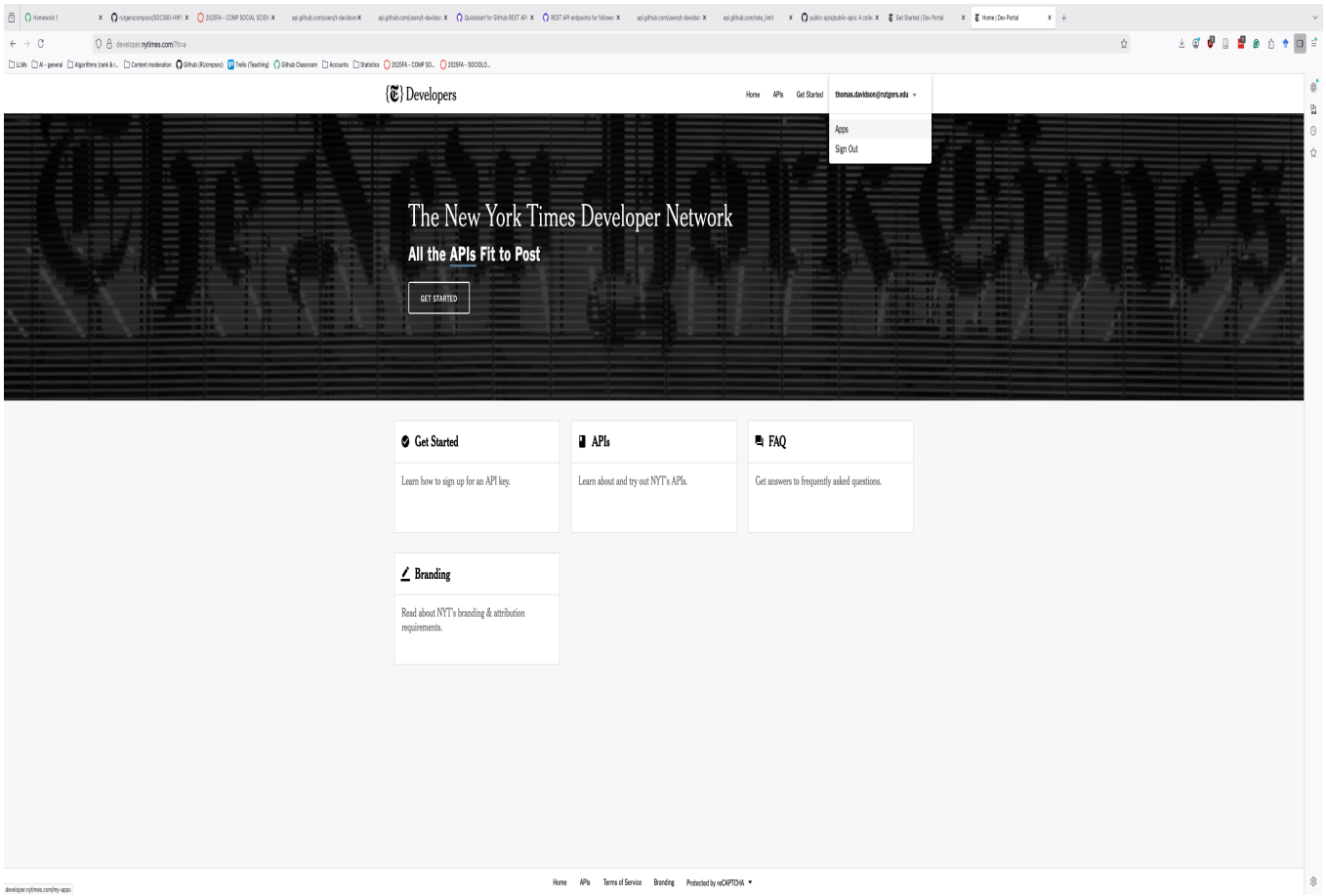
To use the API you will need to create an account and obtain credentials.



Using the NYT API

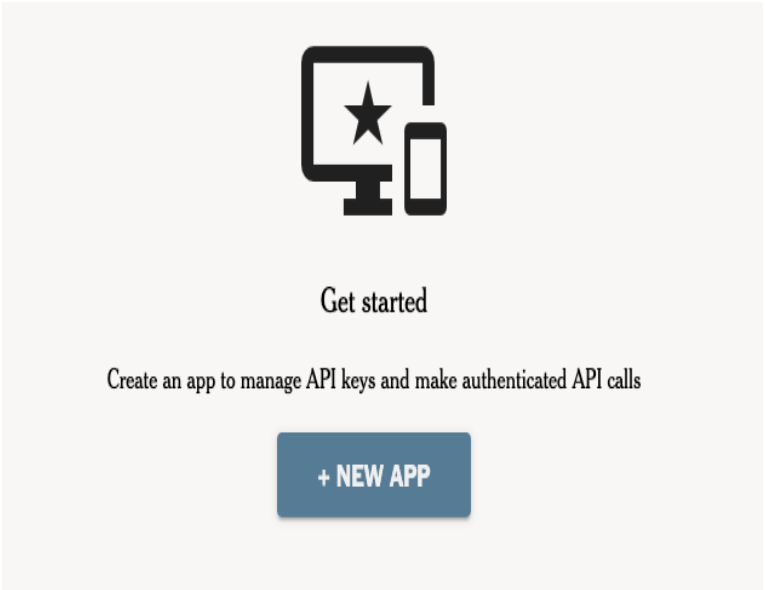
Creating an app

Click Apps from the dropdown by your email in the top-right



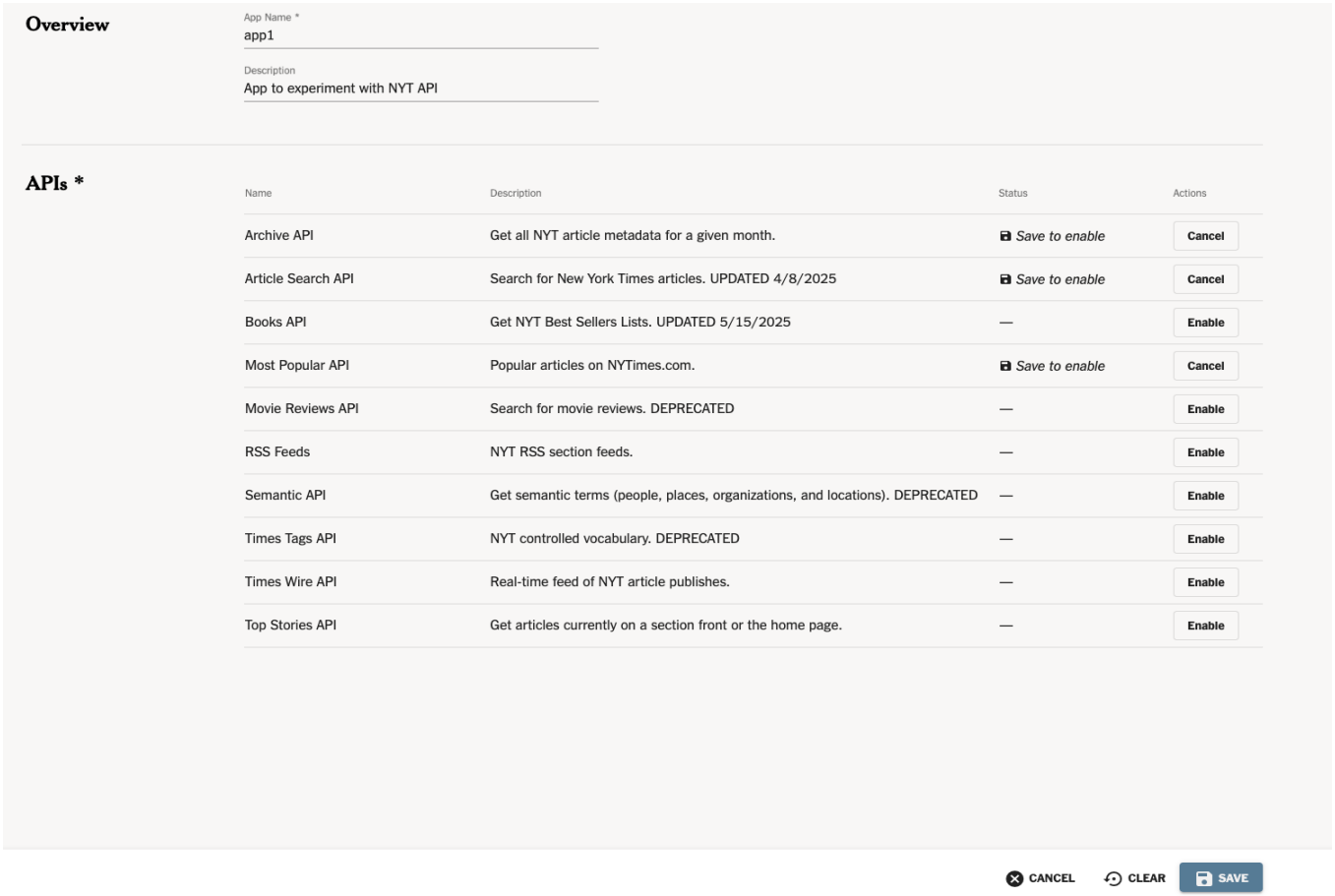
Using the NYT API

Creating an app



Using the NYT API

Setting up the app



Overview

App Name *
app1

Description
App to experiment with NYT API

APIs *

Name	Description	Status	Actions
Archive API	Get all NYT article metadata for a given month.	*Save to enable*	Cancel
Article Search API	Search for New York Times articles. UPDATED 4/8/2025	*Save to enable*	Cancel
Books API	Get NYT Best Sellers Lists. UPDATED 5/15/2025	—	Enable
Most Popular API	Popular articles on NYTimes.com.	*Save to enable*	Cancel
Movie Reviews API	Search for movie reviews. DEPRECATED	—	Enable
RSS Feeds	NYT RSS section feeds.	—	Enable
Semantic API	Get semantic terms (people, places, organizations, and locations). DEPRECATED	—	Enable
Times Tags API	NYT controlled vocabulary. DEPRECATED	—	Enable
Times Wire API	Real-time feed of NYT article publishes.	—	Enable
Top Stories API	Get articles currently on a section front or the home page.	—	Enable

CANCEL CLEAR SAVE

Using the NYT API

Creating an app

- Add a name and a short description
  - e.g. “Computational Social Science”, “App for class”
- Click Enable for Archive, Article Search, and Books APIs

APIs

Access credentials

- Often APIs will use credentials to control access. These might include:
  - A *key* (analogous to a user name)
  - A *secret* (analogous to a password)
  - An *access token* (grants access based on key and password)
- 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 NYT API

Storing credentials

- Open `creds.json` (located in the `credentials` folder of the course repository) and paste the App ID, key, and secret (click "Show secret") into the relevant fields. Save the file.
  - Storing credentials in a separate file helps to prevent them from getting committed to Github accidentally

Using the NYT API

Storing credentials

- Your file should look like this:

```
{"id": "le90de-4023-8b21-493h-489fv030",
"key": "328248djkejf298382189du329323c",
"secret": "jw7329889d37f7798383e8d29ew2d"}
```

Note: I made up fake credentials here as an example, otherwise they would be leaked when these slides are shared on Github.

Using the NYT API

Loading packages

We're using `nytimes`, a *wrapper* around the NYT API. To install it, uncomment and run the line below.
Follow any instructions in the Console, selecting CRAN packages if asked to update and answering "Yes" when asked whether to update from source.

```
# remotes::install_github('t-davidson/nytimes') # uncomment and run chu
```

You can read more about the library here.

Using the NYT API

Authentication

Now let's load the packages, read in the credentials, and create an access token. Run this chunk to proceed.

```
library(nytimes)
library(tidyverse)
library(jsonlite)
library(lubridate)

creds <- read_json("../credentials/creds.json")  # read creds

nytimes_key(creds$key)
```

Using the NYT 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.

```
`?`(ny_search)
print(ny_search)
```

Using the NYT API

Querying the API

Now we're authorized, we can use the package to retrieve information from the API. Let's try running the search function.

```
articles <- ny_search("artificial intelligence",
                      since = Sys.Date() - 30,
                      pages = 15,
                      sort = "newest")
```

Using the NYT API

Inspecting the results

```
head(articles[1])
```

Using the NYT API

Inspecting the results

The data are returned in a nested list. Indexing can be used to extract different components of the results.

```
articles[2][[1]]$headline$main
```

Using the NYT API

Inspecting the results

However, due to the structure of the data we cannot easily cast to a tibble.

```
articles %>% as_tibble()
```

Using the NYT API

Cleaning the data

I made a function to extract the relevant information into a tibble. First, it applies the function to each row, then uses `bind_rows` to render the tibble.

```
extract_info <- function(x) {
  return(tibble(
    id          = x$`_id`,
    headline    = x$headline$main,
    abstract    = x$abstract,
    byline      = x$byline$original,
    pub_date    = x$pub_date,
    news_desk   = x$news_desk,
    section     = x$section_name,
    url         = x$web_url,
    word_count = x$word_count
  ))
}

articles_df <- lapply(articles, extract_info) %>% bind_rows()
```

Using the NYT API

Creating a summary

Let's calculate some statistics using this table. What does this show?

```
articles_df %>% group_by(section) %>%
    summarize(count = n()) %>%
    filter(count > 3) %>%
    arrange(desc(count))
```

Using the NYT API

Creating a summary

Here's another summary, this time saved to a new object.

```
section_words <- articles_df %>% group_by(section) %>%
    summarize(avg_words = mean(word_count), count = n()) %>%
    filter(count > 5)
head(section_words)
```

Using the NYT API

Visualizing the data

```
ggplot(articles_df, aes(x=word_count)) +
    geom_histogram() +
    theme_bw() +
    labs(y = "Number of articles", x = "Word count")
```

Using the NYT API

Visualizing the data

```
ggplot(section_words, aes(y=avg_words, x = section, fill = section)) +
    geom_bar(stat = "identity") +
    scale_fill_viridis_d(option = "cividis") +
    theme_bw() +
    labs(y = "Average word count", x = "Section") +
    theme(legend.position = "none")
```

Using the NYT API

Collecting more data

Let's collect the comparable data for a second query.

```
Sys.sleep(13)
articles2 <- ny_search("first amendment",
                       since = Sys.Date() - 30,
                       pages = 20,
                       sort = "newest")

fa <- lapply(articles2, extract_info) %>% bind_rows()
```

Using the NYT API

Collecting more data

For convenience we can join the two datasets by combining the rows. But a new column needs to be added to ensure we know which keyword matched each article.

```
articles_df$topic <- "AI"
fa$topic <- "1A"
both <- bind_rows(articles_df, fa)
```

Using the NYT API

Comparing the topics

```
both %>% group_by(topic, section) %>%
    summarize(count = n()) %>%
    top_n(count, n=3) %>%
    ggplot(aes(y=count, x = section, fill = section)) +
    geom_bar(stat = "identity") +
    scale_fill_viridis_d(option = "cividis") +
    facet_wrap(~topic, scales = "free_x") +
    theme_bw() +
    labs(y = "Number of articles", x = "Section",
         title = "Top 3 sections by topic") +
    theme(legend.position = "none")
```

Using the NYT API

Attempting another visualization

What can be improved here?

```
both %>% ggplot(aes(y = word_count, x = pub_date)) +
    geom_point() + theme_bw()
```

Using the NYT API

A better plot

Using `as.Date` so that `pub_date` is correctly processed. And adding some additional information.

```
both %>% ggplot(aes(y = word_count, x = as.Date(pub_date),
                    color = topic)) +
    geom_point() + theme_bw()
```

Using the NYT API

Comparing coverage over time

```
both %>% mutate(day = as.Date(pub_date)) %>%
    group_by(topic, day) %>%
    summarize(coverage = sum(word_count)) %>%
    ggplot(aes(y = coverage, x = day, color = topic)) +
    geom_point() + geom_line() + theme_bw() +
    labs(y = "Total words", x = NULL, color = "Topic",
         title = "Coverage of 1st Amendment and AI in the New York Time
```

Using the NYT API

Comparing coverage over time

Calculating a daily ratio of 1A to AI coverage.

```
both %>%
  mutate(day = as.Date(pub_date)) %>%
  group_by(day) %>%
  summarise(
    num = sum(word_count[topic == "1A"]),
    den = sum(word_count[topic == "AI"]),
    ratio = if_else(den == 0, NA_real_, num / den),
    .groups = "drop"
  ) %>%
    ggplot(aes(x = day, y = ratio)) +
  geom_hline(yintercept = 1, linetype = "dashed", linewidth = 0.25) +
  geom_point() + geom_line() +
  theme_bw()  +
  labs(y = "1A / AI coverage ratio", x = NULL, color = "Topic",
         title = "Ratio of 1st Amendment to AI coverage in the New York
```

Using the NYT API

Exercise

- Use `ny_search` to find a selection of articles of interest
- Clean up the results
- Produce a visualization, distinct from those already shown using `ggplot`
- Add your visualization to this Google Doc (sign in using Rutgers account): https://tinyurl.com/css-nyt-viz

Using the NYT API

Exercise

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