Do We Like the Stock?

ETL Project - Extract, Transform, Load

**Team Stonks:**

Jennifer Randle

Jeriel Minguillan

Sanabu Washizuka

Paula Lopes

This project examined the Extract, Transform, and Load methods with the New York Times API. We explored the Archive API, Top Stories API, Article search API, and Most popular API to answer the question, “Do we like the stock?” The stock in question being, Gamestop (GME).

We chose to use a No-SQL database which allows for the data to be stored in a collection. The keys of the collection were then queryable. Keys such as “Title”, “Headline”, and “keywords.”

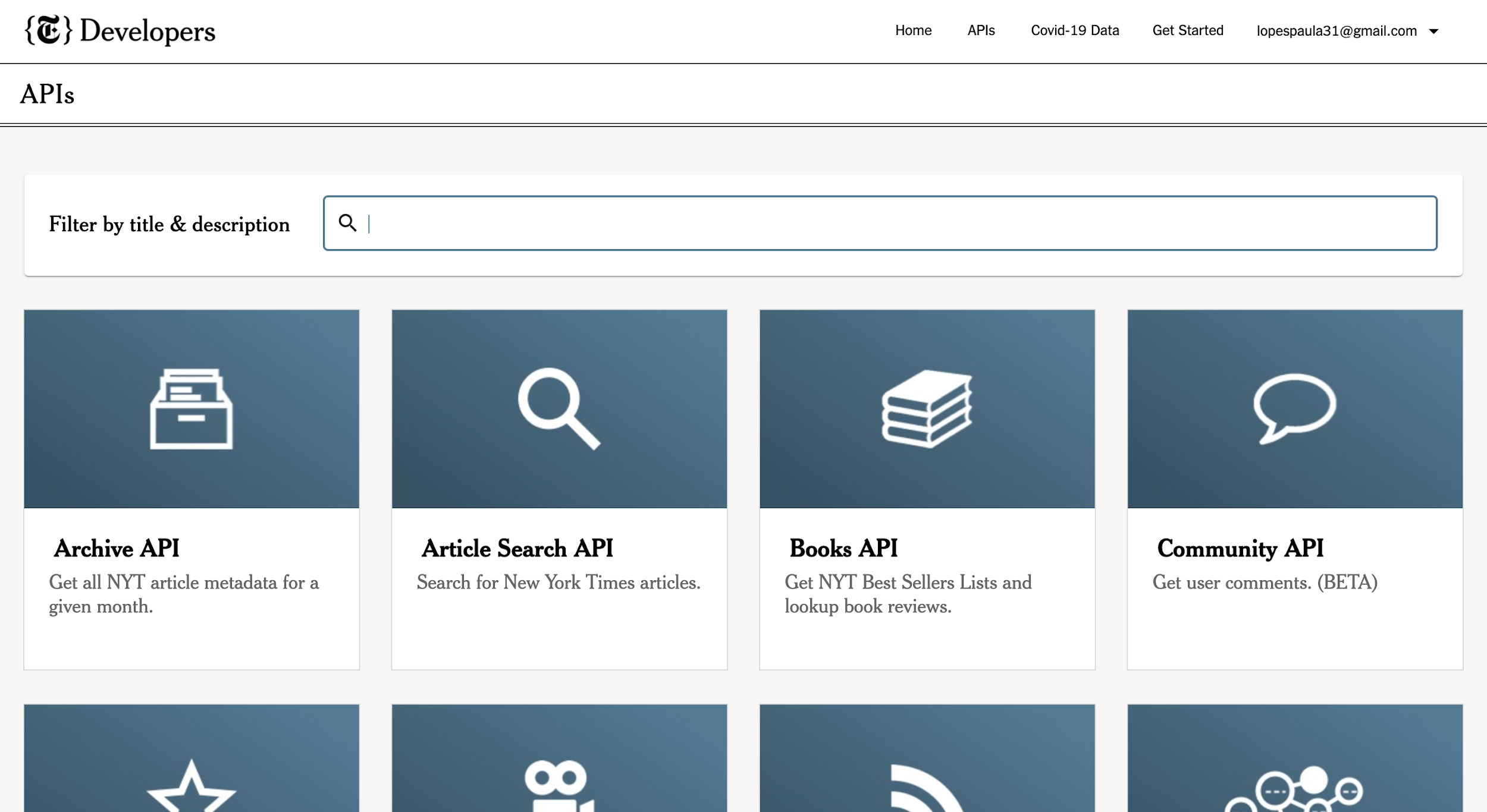
The archive API generated metadata on all of the articles within a given month. This metadata was inserted into a Mongo database. From this database, keywords such as “Gamestop” were searchable.

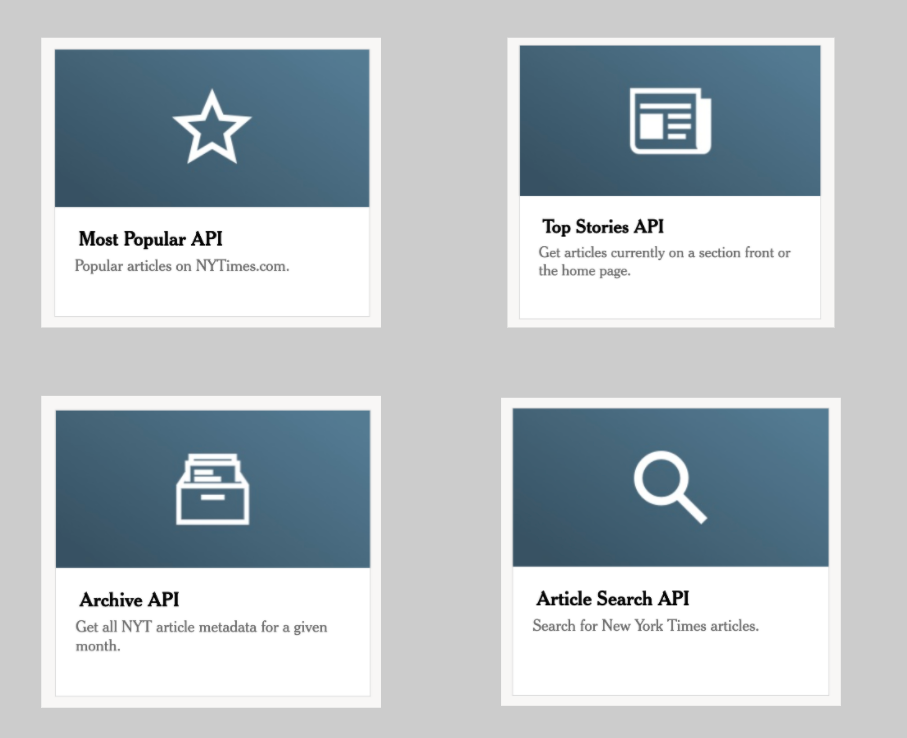
The Top Stories API generated metadata on all of the articles currently on the landing page for each section of the NYT. We chose to do Business, Technology, US, Politics, and Opinion sections, where we were able to pull the title and published date information to input into MongoDB.

Article Search API….

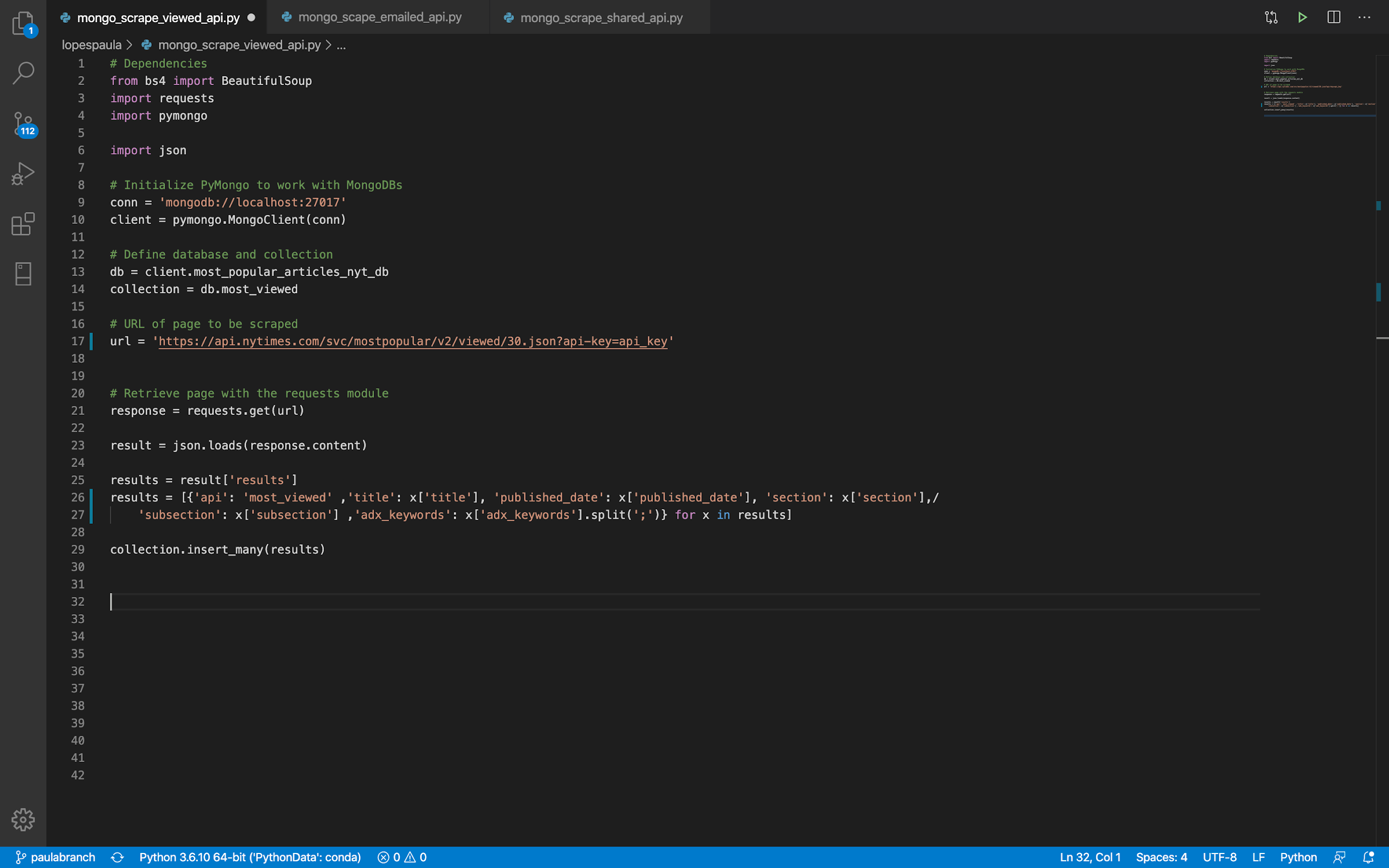
**NY Times API**

On the New York Times website we could find many available api's. We analyzed each one of them and decided to work each one of us with a different api, so we would be able to compare the results at the end.

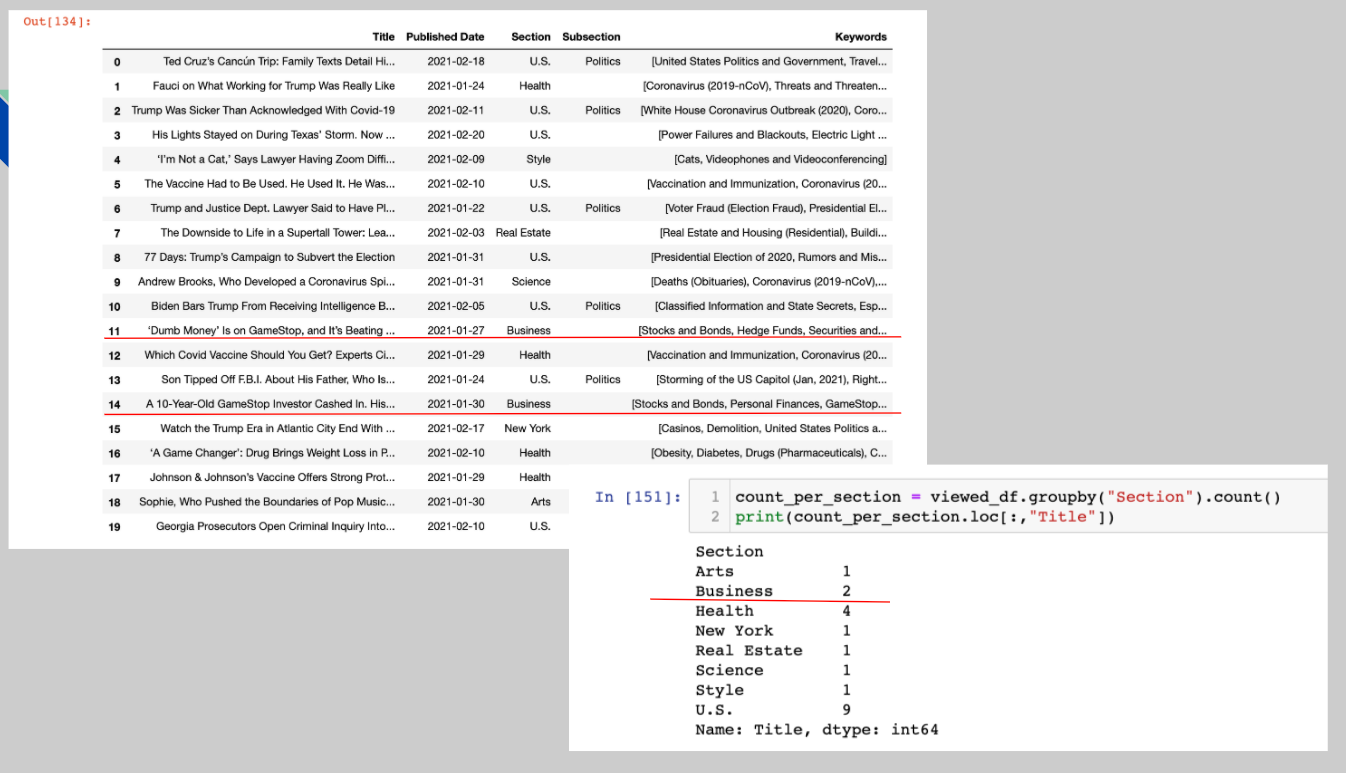
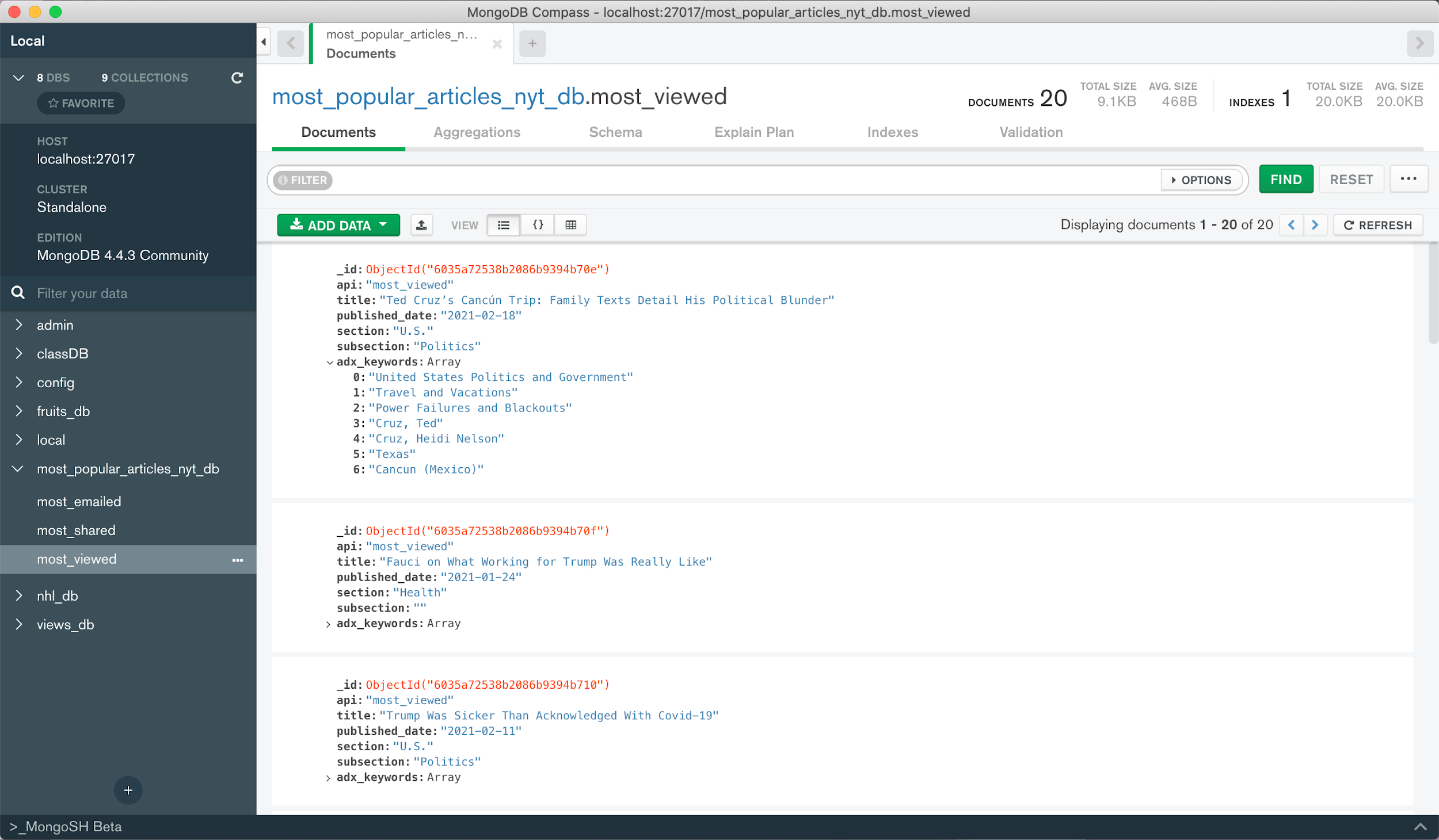


The api’s we chose to use were Most Popular Articles, Top Stories that returns the articles that are in the homepage, Archive that returns all the articles from a given month and Article Search that we were able to search any article we wanted.

The first api was Most Popular Articles and inside of it we had three options: most viewed articles in the last 30 days, most emailed articles in the last 30 days and most shared articles in Facebook in the last 30 days. We used three different codes to query each one of them but we could have done it in just one, looping through the three api’s.



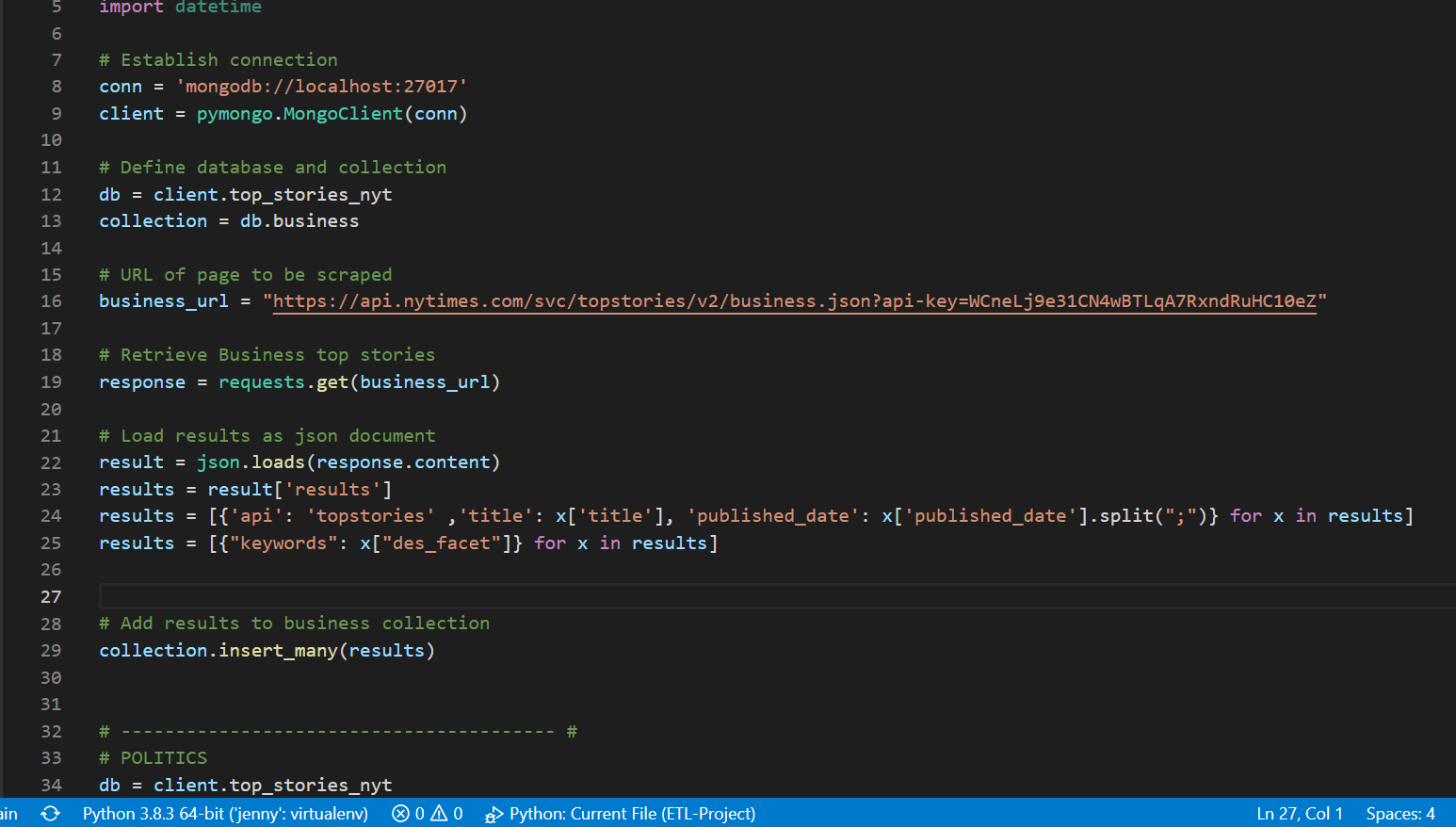
To extract the data, we first connected to mongo db and declared the database and collections we wanted to create. Then we prepared to call the Api using the url and request.get. And transform the data to store it in the collections using just the keys that we judged necessary to answer our question. Which was title, published date, section, subsection and key words. To load it into the mongo database we used collection.insert many results.

In this image on top we can see the database in mongo compass with the collections most viewed, most emailed and most shared and the keys I choose to load in.

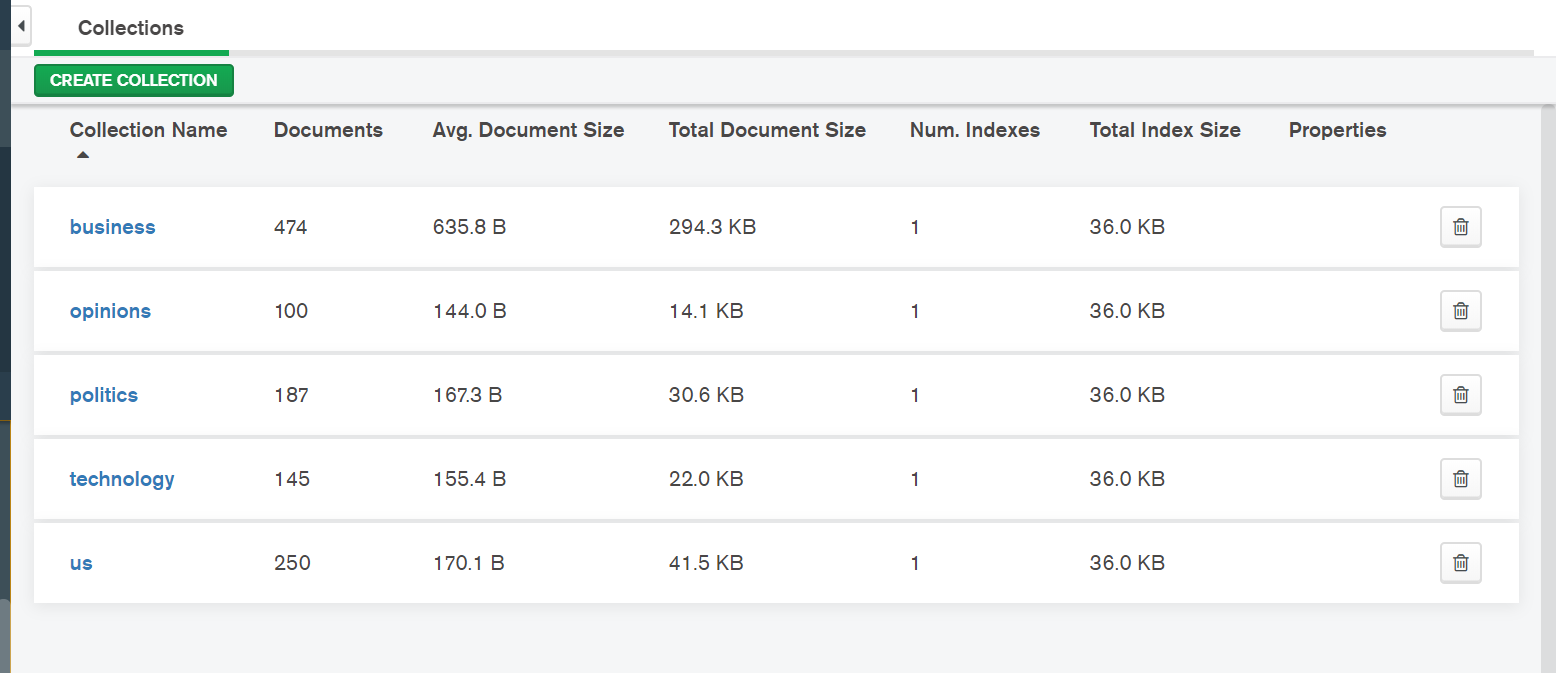
Here, what we did was to query the mongo database in Jupyter Notebook and organize it in a panda’s data frame format to make it easier to see. The idea was to filter it by keywords to find “game stop” in it and the articles related. But it was not a good approach because the keywords are in an array and I would need to normalize this data before it’s useful in an sql database.

By the way, since the api return was just 20 most popular articles I was able to manually identify that 2 of them were related to Game stop, which represents 10% and is a good number. But in case the database was longer I wouldn’t be able to analyze it this way.

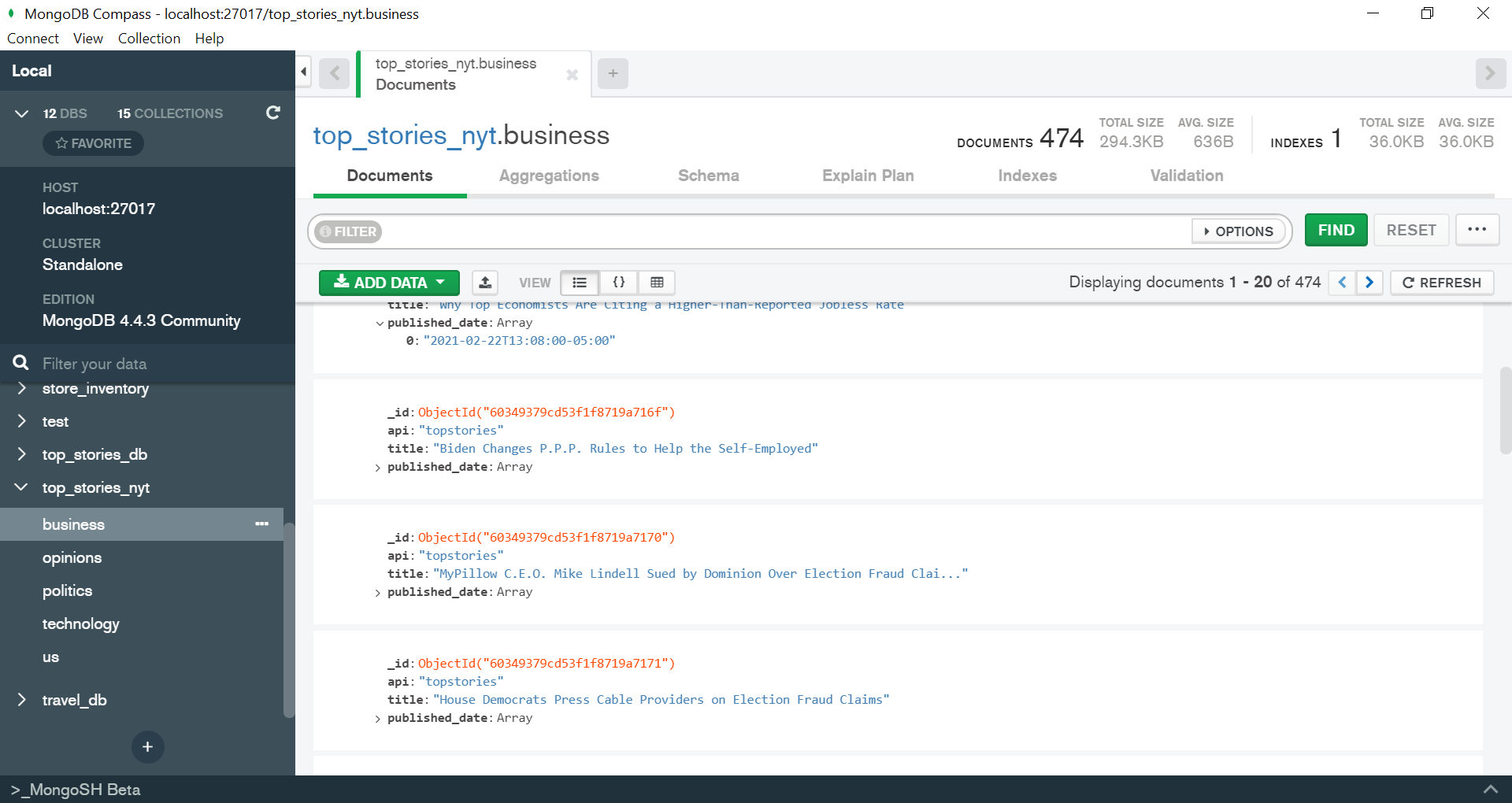
**Top Stories API**

The top stories api returns the top stories on the landing page for each section of the NYT. We chose the 5 sections of Business, Technology, Politics,US and Opinion. We chose those 5 because we decided those were probably the most relevant sections for posting articles that would be talking about GameStop. I decided to create a new collection for each section, by repeating the code below 5 times with the corresponding url for each. 

I included the title and published date in the database, however keywords were in an array that could not be split, so it would not pull from the json object. There were also no parameters for the amount that would output or the dates, so each section significantly varied in article count, as you can see in the document count in the image below.

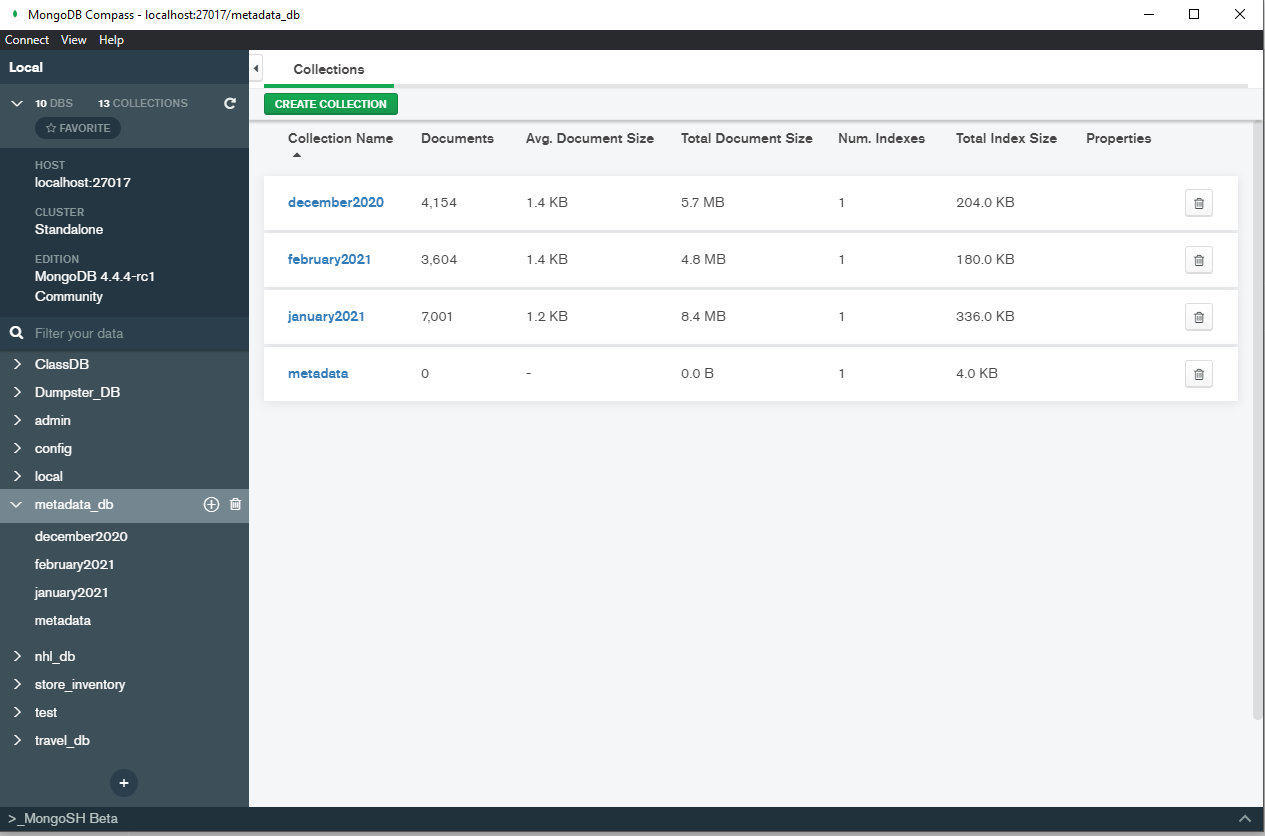


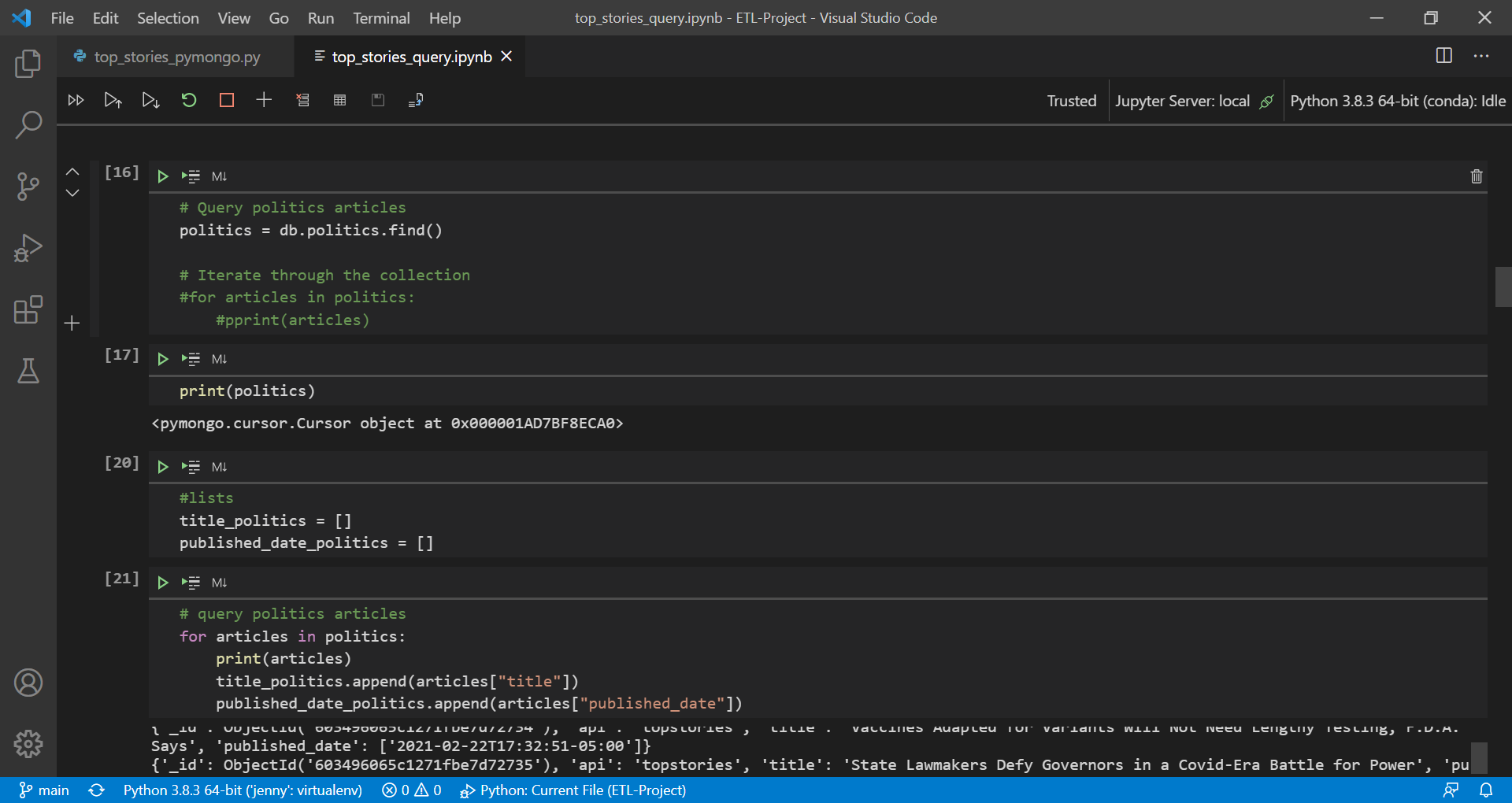
This is the inside of the business collection in the database.

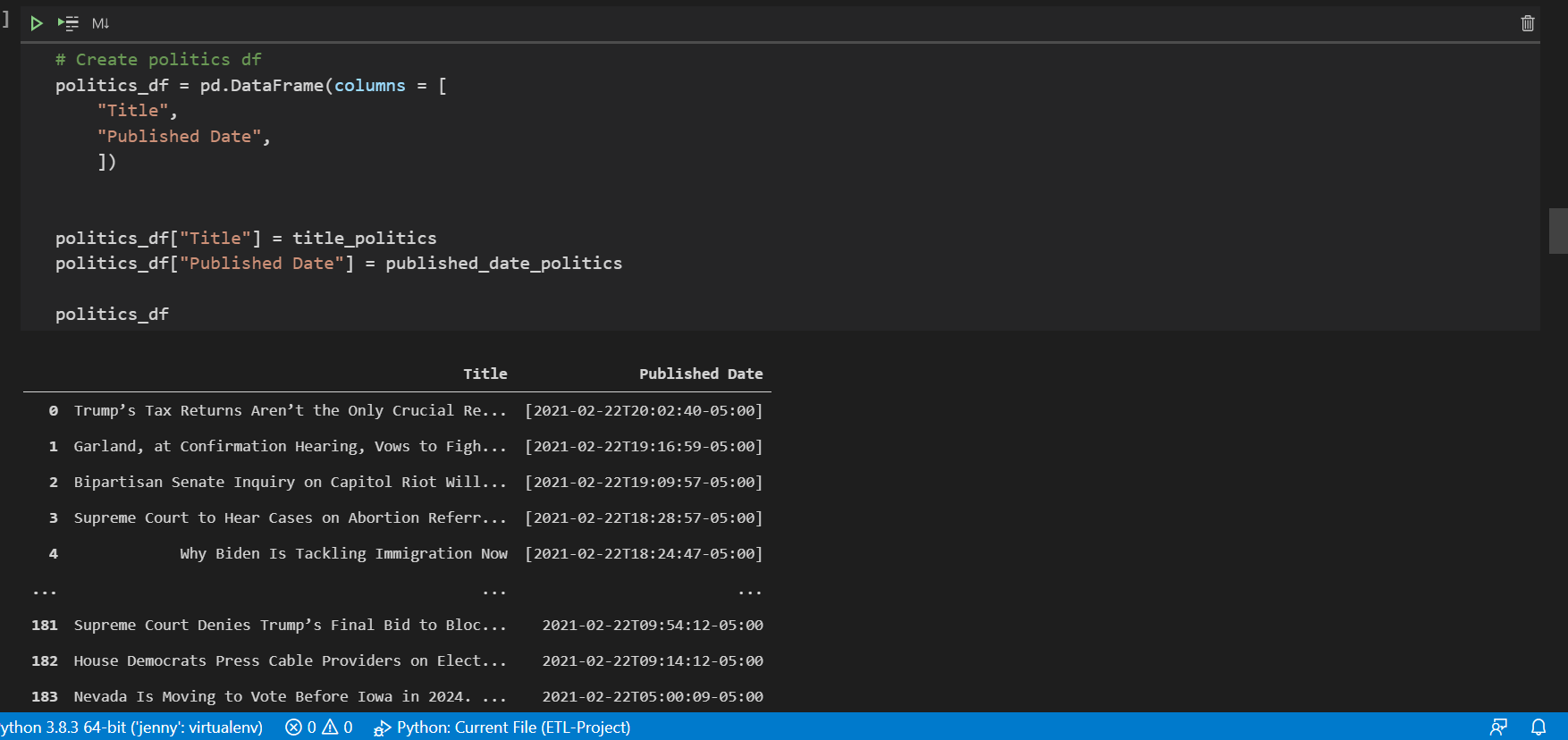


I then created a query for the collection in the database. As you can see in the image below, I connected to each collection within the database and then set-up lists for the title and published date of each. I think looped through the database to pull out the specified info, and appended that info to the list. Once the lists are created, you can input them into a pandas dataframe

**Archive API**

Extracted the data using the Archive API files. Querying all of the different articles within the last 3 month span yielded the database and the monthly collections. Using MongoDB allows for a key query for different keywords such as “Gamestop.”





## **Article Search API**

##### ETL1

Queries were made to the Article Search API using json library to make requests to the API, which focuses primarily on search terms that the API would use to find matches in its articles. The API produces JSON object responses, with each object as an individual dictionary within the response.

Below are a few more details on the responses:

-Other used parameters include: start dates and end dates for the search range, pages for pagination of results

-Max 200 pages are given per search term

-API outputs 10 articles per page (maximum 2,000 per response)

#### Purpose of this ETL

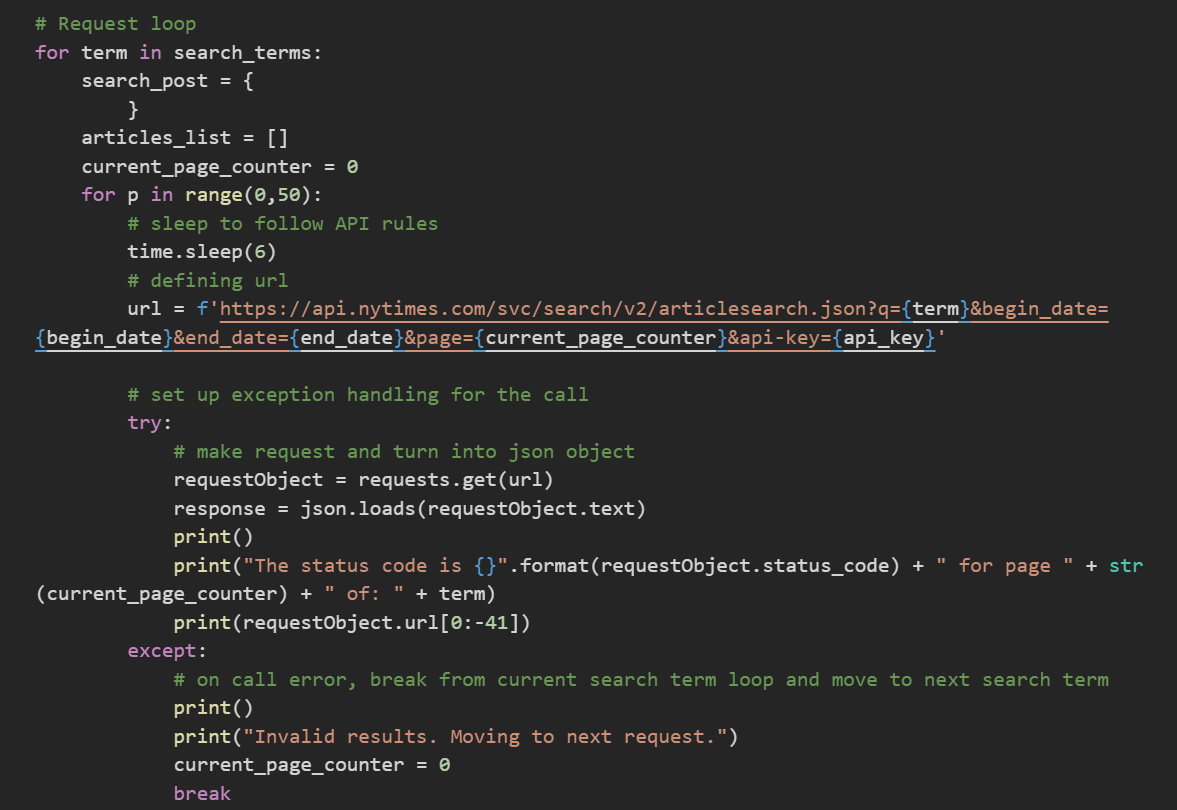
To create a data source of the raw responses from the Articles Search API, to be able to access quickly at our convenience.

#### Operating File:

Article\_Search/app\_3\_pt1\_mongodb\_loader.ipynb

#### Extract

Queried to the NY Times API using the requests library to actually retrieve responses.



#### Transform

The json library's json.loads() method is used to immediately convert the response to a json object.



The code targets a list of the dictionaries that represent each article.

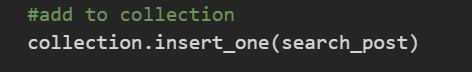
#### 

Pulled other metadata to tag onto the document for the MongoDB.

#### 

#### Load

Used pymongo's insert\_one() method to load document into the MongoDB database.



#### Resulting Data

A series of MongoDB collections, housing a list of documents that represent each article:



#### Update Frequency

Every week, setting the query dates to be the next week from the last query date range.

\*An improvement is to be made to compensate for duplicate results.

#### 

#### ETL 2

#### Purpose of this ETL

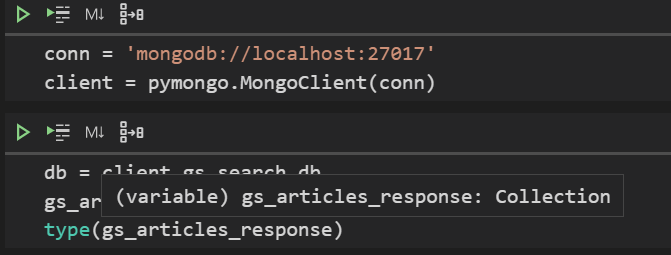
To prepare cleaned data into a SQL database to be made accessible for easy analysis.

#### Operating File:

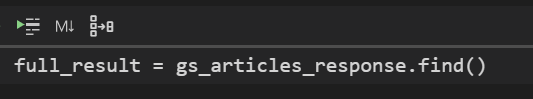
Article\_Search/app3\_pt2\_pgloader.ipynb

#### Extract

Establishing connection to MongoDB database of the raw responses.



Getting entire database and passing to a variable.



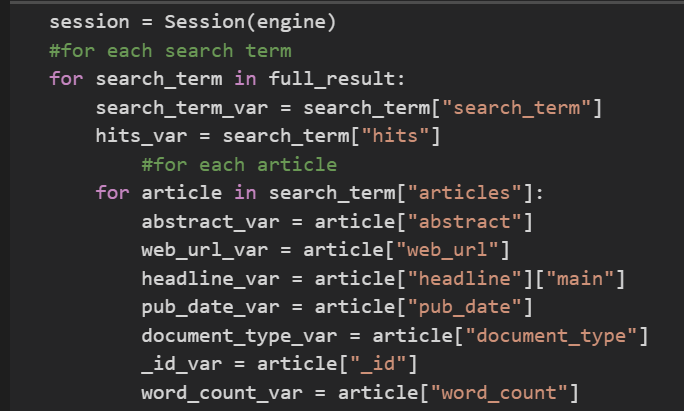
#### 

#### Transform

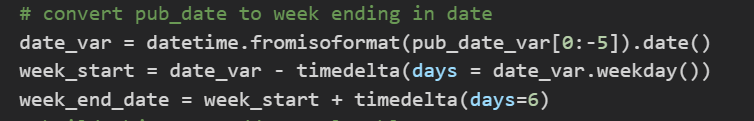
Create a session to prepare for adding data to the SQL database.

Looping through each collection within the database and grabbing that collection's search term and hit count.

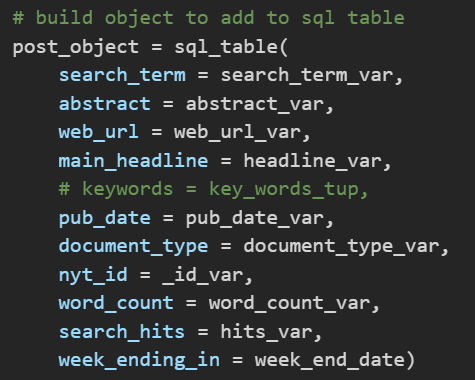
In an embedded loop, begin passing each value to be reviewed into a variable, which will be stored in a single dictionary.



Perform one transformation of a date, changing it to a date object and finding the beginning of the week for that day. ("Beginning of the week" for datetime library is considered to be Monday)

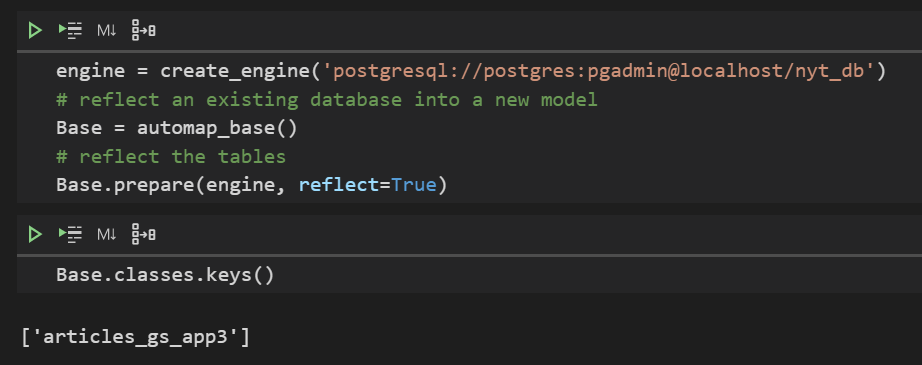


Preparing a dictionary which will become a single record for the PostgreSQL table.

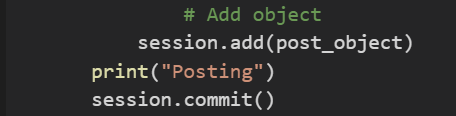


#### Load

Prepare connection to a target PostgreSQL database using SQL Alchemy.



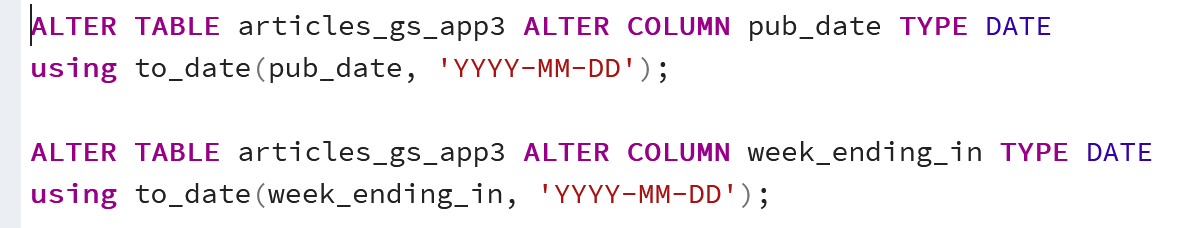
Posting prepared dictionary to the table.



Closing session after all loops are completed.



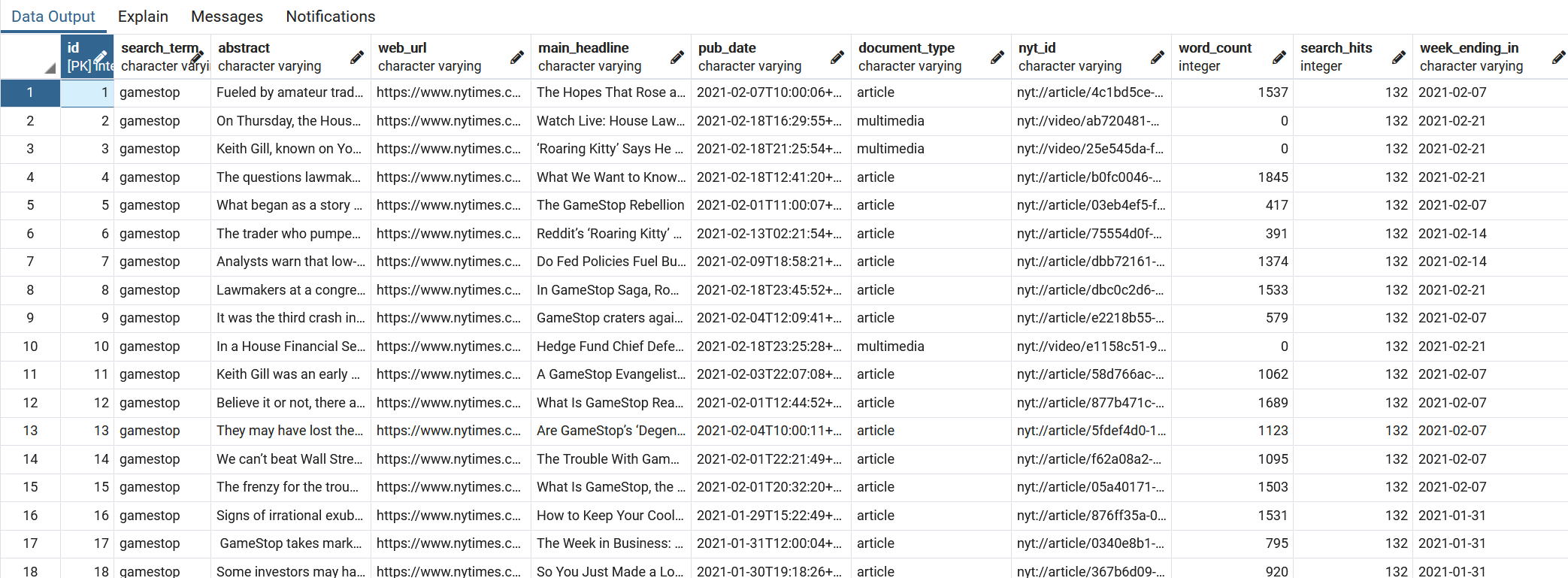
Transformation on PostgreSQL to change column types to date objects.



\*Uses file: Article\_Search\sql\_scripts\transform\_articles\_tables.sql

#### Resulting Data

A PostgreSQL Table:



#### Update Frequency

Updates will be done whenever analysis is requested, as it pulls all data from the MongoDB database.