Visualizing the Spread & Effects of COVID-19 Around the World

Micaela Flores, Sheetal Laad, Ziv Schwartz, Kelly Sooch & Orion Taylor

Introduction & Motivation

Amidst the COVID-19 pandemic, we found common interest in applying advanced python methods to build a visualization dashboard that investigates its effects. We choose three particular areas of interest:

(1) economic effects (measured by relative change in stock price based on each company's range from November 2019 to January 2020);
(2) social effects (measured by volume of Twitter activity per day per geography); and
(3) Public health effects (measured by total fatalities per day per geography).

Wanting to present the data in a cohesive manner, we use Python's *plotly* visualization package with *dash*, which together enable us to produce a single sub-plotted visualization with a universal slider to provide a dynamic time series view of daily change. Given the scale and limitations of some of our data and collection capacities (e.g., Twitter's API call restrictions forced us to seek other data sourcing), we gather data from January 21st onward, a reasonable negotiation given the timeline of the global spread of COVID-19 which escalated mostly after mid-January.

There were a few areas of our pipeline where we find it useful to employ the methodology of our coursework. For example, to process the full Twitter data (discussed at greater length in the sections below) of the period in question (January 21st through end of February) and extract Tweet counts by country, we use multithreading to expedite the uploading of bulky *pickle* files to AWS S3. Likewise, we use similar methods to parallelize our Tweet location data cleaning process. As expected, implementing multithreading greatly reduces execution time and ensures that running these tasks in the future is much more realistic should more current tweets be acquired.

Data Collection

Getting Tweets

As previously mentioned, one of the main objectives of this analysis is to explore if there exists a relationship between the social and health impacts of the coronavirus. Namely, we desire to see if the regions in which more people are tweeting about the coronavirus follow the temporal spread of global confirmed cases. As such, we were first tasked with obtaining all tweets that contained #coronavirus or #covid19. Of course, there exist many tweets relevant to the coronavirus that do not include these hashtags, but in order to facilitate acquisition, we proceed with only tweets that contain these hashtags and likewise filter out retweets.

Initially, we experimented with pulling the tweets via the Twitter Developer API. This process consists of applying for a Twitter Developer account and creating an approved application in order to obtain the consumer API keys and access tokens. Next, we attempted to use *tweepy*, a Python package specifically developed for use with the Twitter API, to extract the tweets with the desired coronavirus-related hashtags. This approach, however, presented some unexpected challenges. Firstly, the *tweepy* package, while extremely convenient and easy to implement, restricts the user from accessing any historical tweets older than 7 days before the current date. This obviously introduced an issue given that we desire tweets dating back to January. Likewise, Twitter also imposes a fetching rate limit of 180 calls per 15 minutes for the premium free-tiered API which, due to the sheer volume of tweets about the coronavirus, made this approach with *tweepy* very unrealistic given our use case and time constraints.

We also briefly explored manually pulling the tweets using pure Python and the *requests* library. If one attempts to obtain tweets via this method, then Twitter imposes additional constraints of a maximum of 50 API calls or a maximum of 5,000 total tweets per month. Again, this approach quickly became unpromising as there are several millions of tweets that concern this global pandemic.

As both of these initial methods presented some significant drawbacks, we researched any existing coronavirus tweets datasets published online. The COVID-19-Tweet-IDs GitHub repository (cite) is a regularly updated ongoing collection of tweets IDs associated with the coronavirus. Tweet collection commenced on January 28, 2020, meaning that the first historical tweets in the dataset begin on January 21, 2020 (due to the Twitter API allowing access to tweets from the preceding 7 days). This repository was extremely crucial in our gathering of tweets because all the tweet IDs available are already organized by month, day, and hour up to the current date. Another benefit to using this repository is that it includes additional coronavirus related hashtags, including #cdc, #n95, #epidemic, #wuhanlockdown, etc. All additional keywords can be found here.

It should be noted that this repository only publishes tweet IDs, not the tweets themselves. This is due to Twitter's Terms of Service, which only allow tweet IDs to be publicly published for non-commercial research purposes. Because of this, the tweet IDs need to pass through a process called hydrating, which essentially takes each tweet ID and uses the Twitter Search API to retrieve the relevant JSON metadata about each tweet. Fortunately, this GitHub repository contains an existing script that uses an analogous package called *twarc* to hydrate all the tweet IDs. While convenient, the hydrating process took extremely long to complete due to the volume of tweets and tweet fetching rate limits. For context, hydrating all 29 days of February 2020 took about a week to finish. Nevertheless, once all the tweet IDs for the end of January to the end of February were hydrated, we then created a script to convert the raw JSON files into a more easily usable format.

As the hydrating script saves the JSON response into zipped files, we first unzip the file then decode the JSON. Again, the features of interest for this analysis are predominately the source location of the tweets, quantified by either latitude and longitude, country code, or a user-supplied location in the profile bio. Our script extracts these features from the raw JSON responses, along with other potentially relevant information such as tweet creation datetime, account username, and tweet text, etc., and stores the results in a dataframe. As each hydrated tweet file contains all the tweets for one hour of one day, all 24 hourly DataFrames are concatenated together and comprise one daily dataframe. All daily DataFrames within a month are then concatenated together to create a single monthly dataframe. These monthly DataFrames are what are used to subsequently obtain the tweet counts per country.

Counting Tweets

Given the data volume limitations mentioned above, we first need to find a location to host the pickled DataFrame files, ultimately choosing AWS S3 to do so. This makes the tweet metadata accessible merely by loading from prescribed URLs. Since this process runs chronologically and can be somewhat time-consuming, we use multithreading in the uploading process and compare the run time of regular vs. threaded execution (e.g., using a sample of 4 files, the execution time ratio was >10,000:1). We also attempted multiprocessing here, but the package for AWS interaction created some limitations with respect to pooling. Once this was complete, we developed an aggregation script to tally the number of tweets per geography per day. This required two steps: first, a massive string cleanup to standardize the many "location" tags associated with the corpus, and second an aggregation of tweet counts per country across the daily DataFrames.

Task	Sample Runtime: *Serial*	Sample Runtime: *Multithreading*
Upload Tweet pickle files to AWS	651.62 s	0.01 s
Process, clean and count Tweets	765.27 s	0.46 s

Twitter location data are notoriously difficult to reconcile. Each Tweet's metadata may contain *very* sparse latitude/longitude and country fields, or else a *location* field. This field can be manipulated by the Tweet author, and thus may not even be a real place name (e.g., "The Land of Oz"). Furthermore, place names are not standardized in any meaningful way (e.g., New York may appear as "New York", "NY, USA", "NYC", "10011", etc.). After some research, we found the most efficient Python library to attempt fuzzy string matching across the hundreds of thousands of location values in our data to be *geopy*. However, given the rate limiting of the *geopy* API and the size of our location set, we first need to employ some manual cleaning.

We first gather lists of country names/abbreviations and U.S. state names/abbreviations and extract these from locations (so, e.g., "Brooklyn, New York" would be extracted as "United States"). This enables us to substantially reduce our corpus of un-located Tweets for each initial DataFrame. We then run through the sets of location values for each DataFrame of un-located Tweets and map them to country names using a *geopy*-based function, saving to a master set. In order to reduce overhead and API calls, we only locate the difference of each un-cleaned set with the master set. The rationale for this is that Tweets coming from the same improperly located user on discrete days might have the same errant location tag, so we would not want to redundantly exercise the *geopy* wrapper. Unfortunately parallelization of an API-based pipeline component without something like IP spoofing is infeasible, so the bulk of the cleaning execution time exists here, dividing labor among our respective machines to partition this step into concurrent processes.

Once the manual and *geopy* cleaning processes are complete, we join together all country-identified data and aggregated counts of Tweets per country per date into a single CSV mapped into a *plotly* geo heat map. Here again we employ multithreading for substantial observed speedup (>1500:1 versus non-threaded). Though our location data is still likely imperfect (*geopy* does not necessarily code the correct country, e.g., it might convert "San Francisco, CA" to "Canada" based on country code), and there are likely some inherent biases underlying them (e.g., people in jurisdictions with censorship repercussions might not Tweet using their exact locations, etc.), the above methodology greatly expanded the scope of our results.

Financial Data

As mentioned above, we wanted to look at the economic impact of the novel coronavirus on various companies, and if their respective stocks are experiencing the effects we conjecture. At a high-level, we hypothesize that companies which promote staying indoors (ex. DoorDash, delivery services), companies which produce or necessities and cleaning supplies (ex. P&G, supermarkets), and companies which help with at-home entertainment and communication (ex. Zoom, Netflix, internet companies) will have an increase in their stock prices as the virus spreads. On the other hand, we predict that companies which have an international presence (ex. McDonalds, Starbucks), companies in the travel industry (ex. Cruises, hotels, airlines), and finance-related companies (ex. Bank of America) will have a decrease in stock prices as the virus spreads.

We use the daily time series API from Alpha Vantage to obtain the financial stock prices for 41 public companies. Alpha Vantage Inc., being a leading provider of free APIs for realtime and historical data on stocks, was chosen for its good documentation, its ability to run relatively quickly, and its availability of daily data from November 2019. For each company, we get the daily closing stock prices from November 21, 2019 to March 31 2020. We experiment with different ways to visualize the change in stock prices including comparing the daily change of each company relative to the daily change of the Dow Jones Indicator, graphing the percent change in stock price for each company, etc. We ultimately decide to represent the financial stocks by normalizing each company's stock price by its range during the period of November 21, 2019 to January 31, 2020, as this compares each company's deviation as coronavirus spread relative to its deviation before the pandemic. During the first iterations of the code, we use *lineprofiler* to locate hotspots in the program. By making the hotspots more efficient, we improve the runtime of this section of the code. Some other code improvements include reducing function call overhead, using local rather than global variables, using data aggregation to reduce loop overheads.

Coronavirus Data
We also want to show the data on total coronavirus cases around the world, and analyze the spread of new cases in each country. We found a dataset consisting of the number of new cases in each country on each day since January 1, 2020. This dataset is a CSV file downloaded from Our World in Data. We then retrieve the central latitude and longitude coordinates for each country, using *geopy*. Lastly, we created a heatmap of new cases per day in each country, which shows the spread of the virus over time.

Adv Py techniques Coronavirus Data
When retrieving the longitude and latitude for each country, we first implemented a function that looped through each country name and if the longitude and latitude was found, we would append the country name, latitude, and longitude to their respective lists. When running the function with *lineprofiler*, we recognized that there were a few ways to make this loop more efficient. We realized the loop was calling certain methods every time, so we used namespaces for the methods instead. We also altered an if-statement that was verifying whether latitude and longitude existed to check only for latitude. Additionally, we converted the list of countries we iterated through into a set. These changes decreased the run time by approximately 10%.

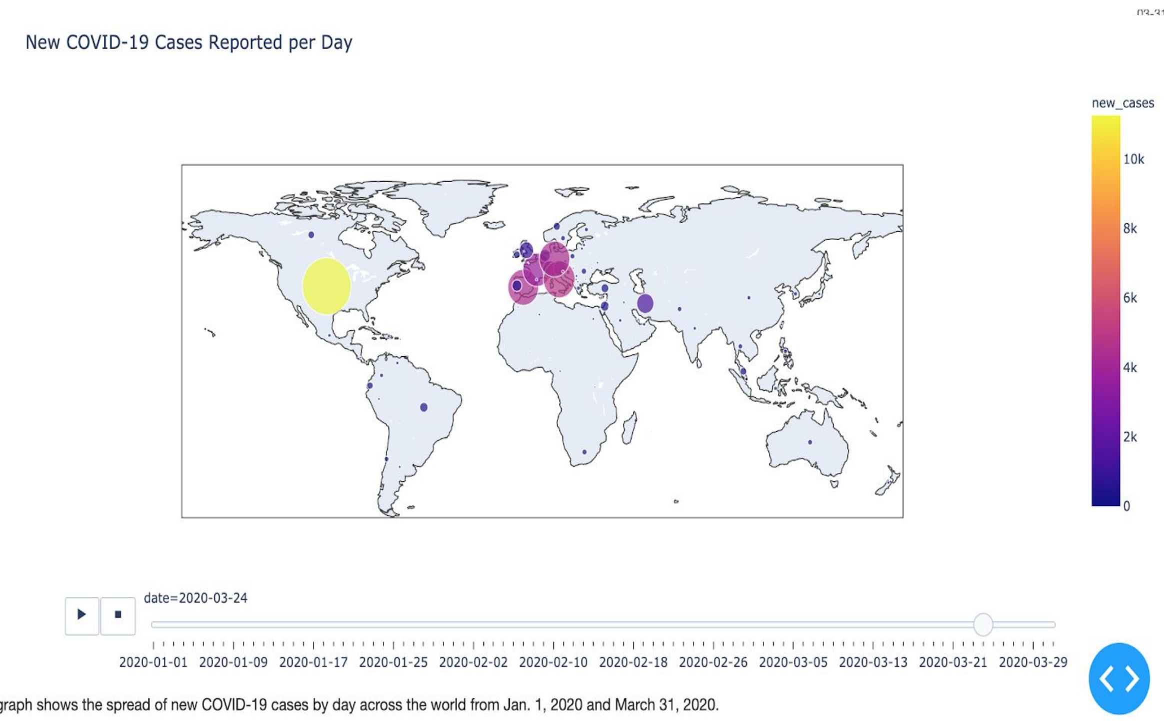
Data Visualization with an Interactive Dashboard

Building Dashboard
To bring everything together, we create an interactive reporting dashboard to better visualize the spread and effects of COVID-19. We build interactive plots using the Plotly module and utilize *Dash* to construct an application that can be hosted as a dashboard. Our dashboard consists of four plots: (1) A world heatmap of the number of coronavirus related tweets sent per day, (2) a world heatmap of new coronavirus cases reported per day, (3) a time series of company stock indices between Jan. 1 and Mar. 31 2020, and finally (4) a time series of company stock indices we hypothesized would either increase or decrease given the spread of the coronavirus. The layout of the dashboard application uses integrated Python, HTML and Dash Core capabilities, with different headers, paragraphs, and Divs to structure the dashboard. To tie it all together, we use callback functions to implement a universal slider to show how all of these graphs change as time progresses and the coronavirus spreads throughout the world. Each of the four graphs call upon the same universal slider and update the plots for the specified date range in unison. The dashboard can also be deployed through a server to be hosted on the cloud.

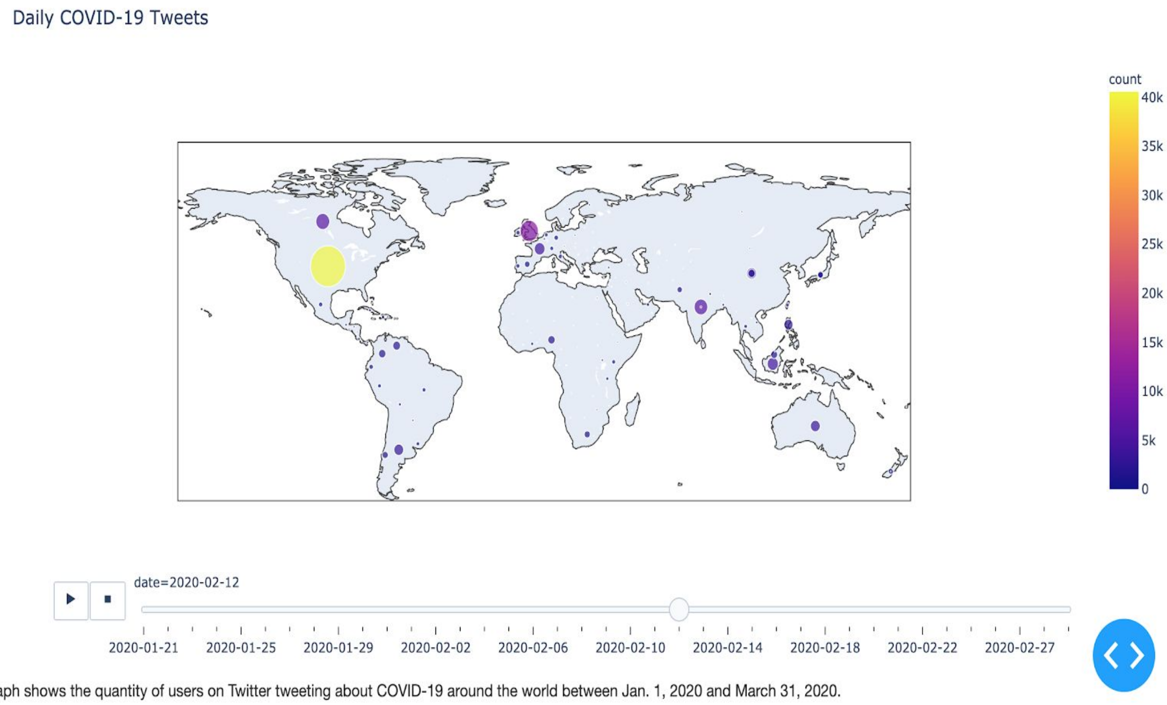
To launch the application:

- **Git clone repository**
- **Initialize Virtual Environment**
- **$ pip install -r requirements.txt**
- **$ python3 app.py**

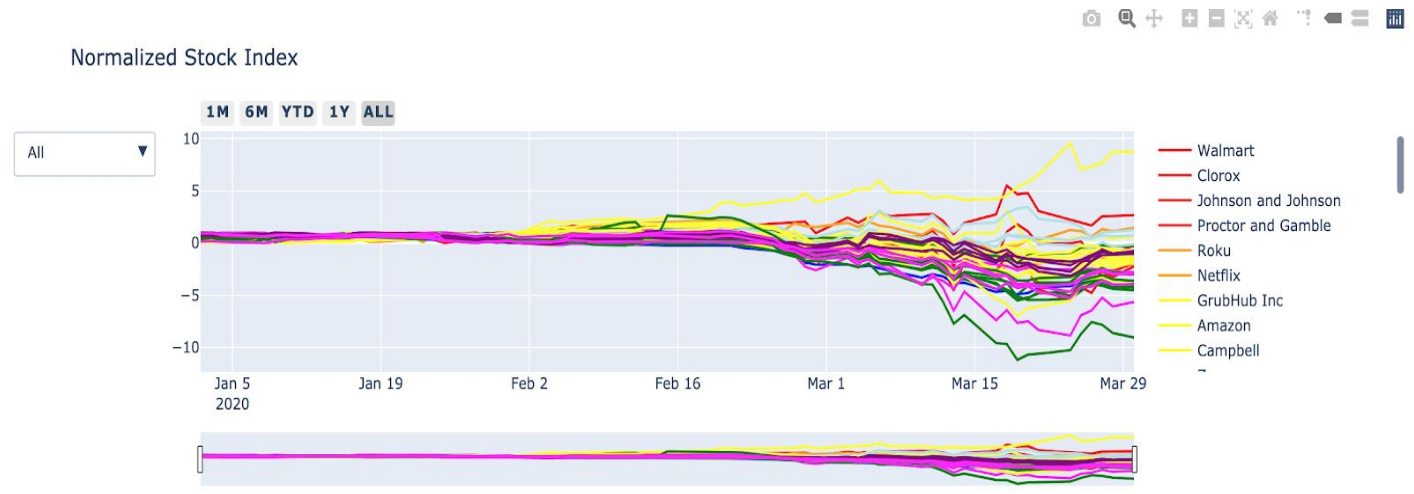
Appendix
Dashboard



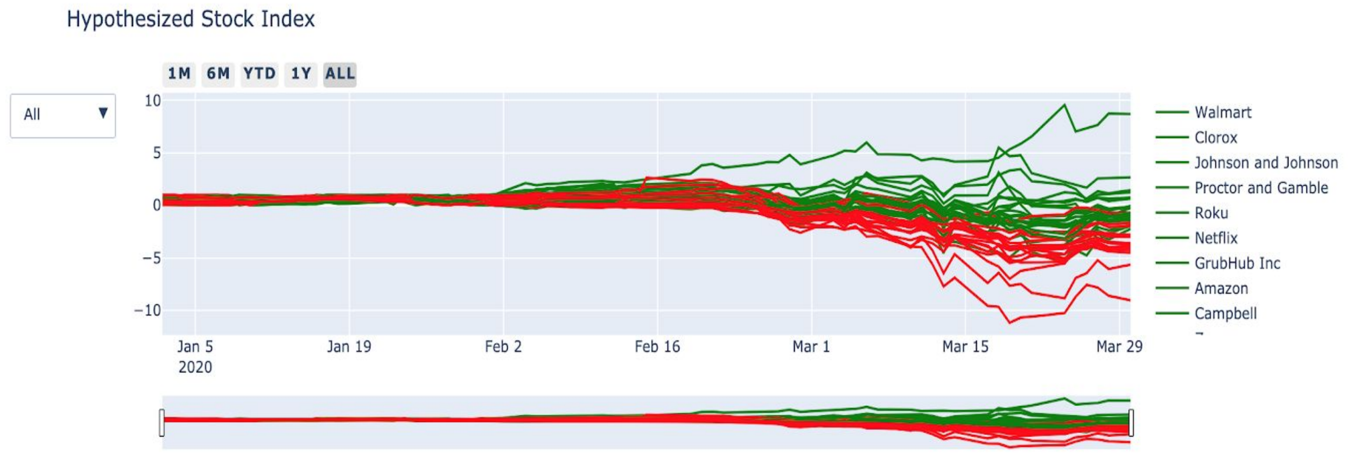
This graph shows the spread of new COVID-19 cases by day across the world from Jan. 1, 2020 and March 31, 2020.



This graph shows the quantity of users on Twitter tweeting about COVID-19 around the world between Jan. 1, 2020 and March 31, 2020.



This graph shows the normalized stock indices for different companies between Jan. 1, 2020 and March 31, 2020. The dropdown menu allows to subset the companies by industry and the range slider allows for smaller window analysis.



This graph shows the normalized stock indices for different companies between Jan. 1, 2020 and March 31, 2020. The difference in this graph is that it shows our teams hypoth ov different companies would fare duing the pandemic: increase or decrease stock. The dropdown menu can show explicitly which companies we hypothesized to increase or decre n a smaller window analysis.