**Novel Corona Virus 19 ETL Project**

**Source Data/Reference Links**

Initial Source:

'https://data.humdata.org/dataset/novel-coronavirus-2019-ncov-cases'

Final sources:

Confirmed Cases: '<https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_19-covid-Confirmed.csv>'

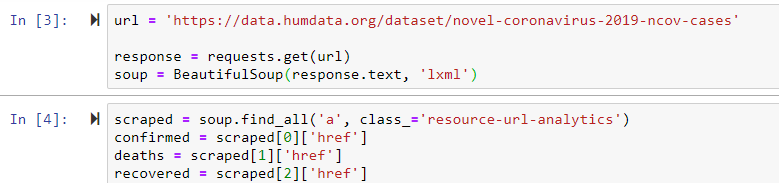
Deaths: '<https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_19-covid-Deaths.csv>'

Recoveries: '<https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_19-covid-Recovered.csv>'

**Extraction**

We found that the data was updated daily, and was available via URL links, which linked to the data in CSV format. These URLs were embedded in the humdata website as a list object. We therefore decided that web scraping would be an ideal way to extract the data, as it would allow us to update the data without downloading new CSV files.

We were initially able to simply scrape directly from the embedded URLs on the humdata website using BeautifulSoup and the read\_csv method.

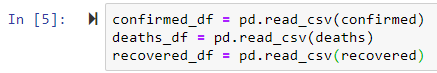


However, as the datasets grew, the data was no longer offered in URL form directly from the humdata website, so we had to change the extraction method to read from the Github account associated with the original URL. Luckily, the data was still in CSV format, so we only had to make minor changes to point the scrapper to the new source locations.

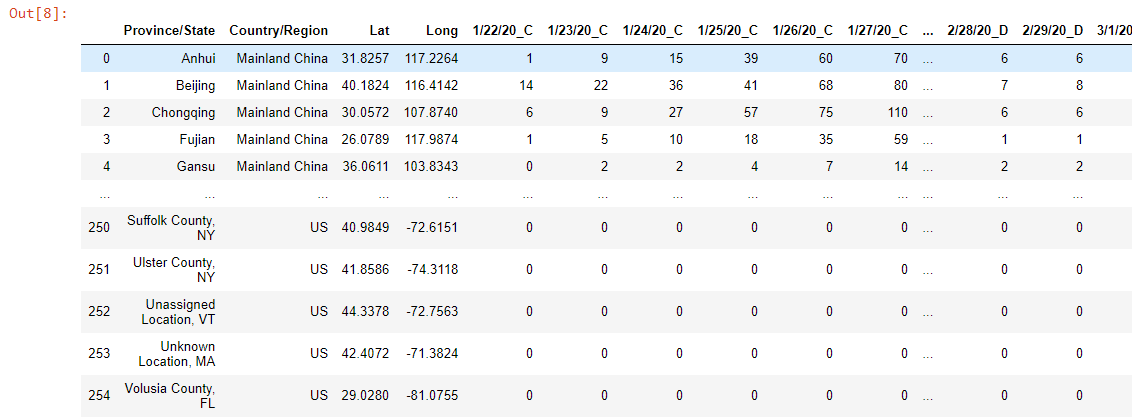
****

**Transformation**

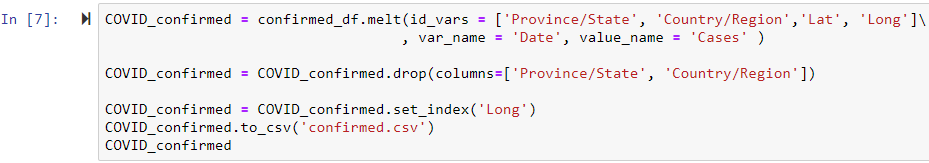
After the data sets were scrapped from the source URLs, we read each data set into it’s own dataframe.

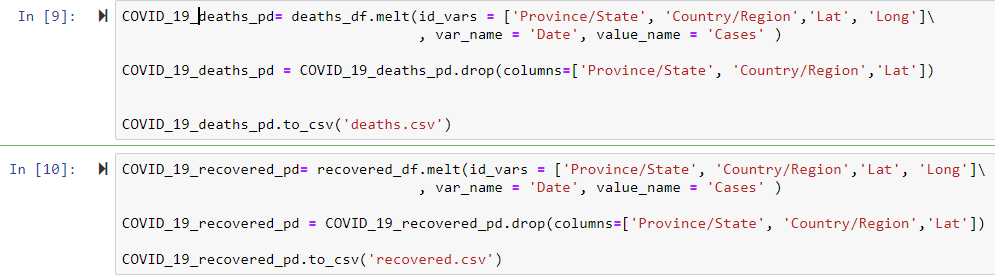


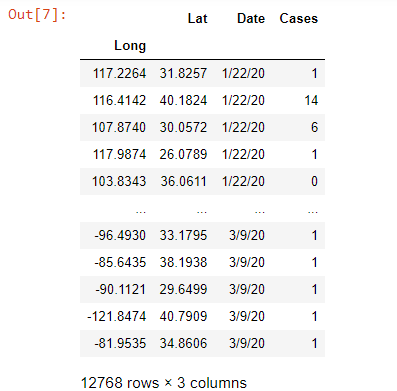
The data was originally formatted with static columns for location (Province/State, Country/Region, Latitude, and Longitude) and one column for each day in the data sets. We initially decided to do a simple inner join on longitude and/or latitude, but found that this would be impractical from a maintenance, querying and storage perspective, because the number of columns would increase whenever dates were added to the data sets.



We resolved this issue by using the ‘.melt’ method on each of the three datasets prior to joining. This method allowed us to restructure the data into one row for each location on each date. We also dropped the Province/State and Country/Region columns because the locations could be sufficiently described using latitude and longitude.







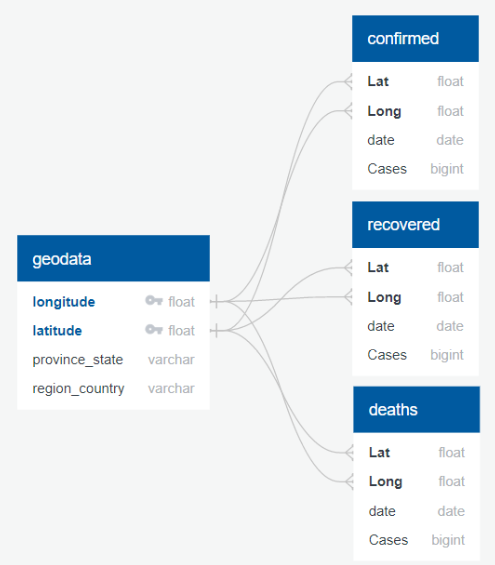
We then created a new dataframe for the geolocation data, which included the latitude, longitude, Province/State and Country/Region.



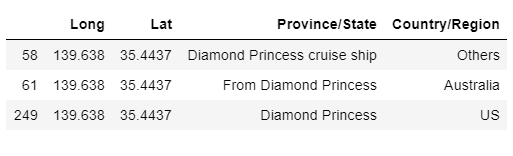
**Loading**

We decided to use SQL for our Database, because our datasets were already fairly structured and uniform. We structured our database in four tables: 3 tables that corresponded to the three initial dataframes containing the statistics for confirmed cases, recoveries, and deaths, and a fourth that contained the geodata information.

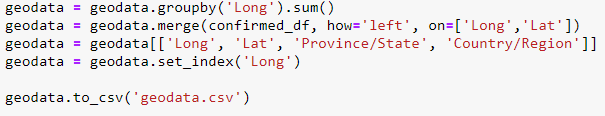
We chose to use longitude and latitude as a composite primary key for the geodata table, because neither the Province/State nor Region/Country are unique in the transformed datasets. This composite primary key was then used as a foreign key for the other three tables.



We did run into an issue after one of the updates of the source material because the data from the Diamond Princess cruise ship was added. The cruise ship was listed for three different countries/regions, the US, Australia, and ‘Others’, and each instance had identical latitudes and longitudes. This caused an issue because these values could no longer be used as the primary key for the geodata table.



To resolve this issue, we had to decide between dropping the cruise ship data, or handling the duplicate latitude and longitudes in some other way. We decided to group by longitude, collapsing the three entries for the cruise ship into one.



Once this issue was resolved, we were able to able to load the dataframes into postgres without issue.

