# Final Project Report

Alexis Metzler and Catherine Grillo
Github Link: Repository

## Goals

We originally planned on examining the correlation between the rate at which COVID-19 spread, measured by the number of days between the first reported case and the 50th reported case, and the per capita number of hospital beds, which we were going to retrieve from the World Bank (website). Once we started working on our project we realized that was not an easy way to retrieve the hospital bed data so we switched from looking at the correlation between hospital beds and speed of spread to looking at the relationship between speed of spread and national GDP, which we were able to strip from a Wikipedia page. Our theory was that richer countries will have greater means to prevent the spread, and COVID-19 would thus have a slower spread. We achieved this goal and were able to find the correlation coefficient between the speed of spread and national GDP.

## Problems Faced

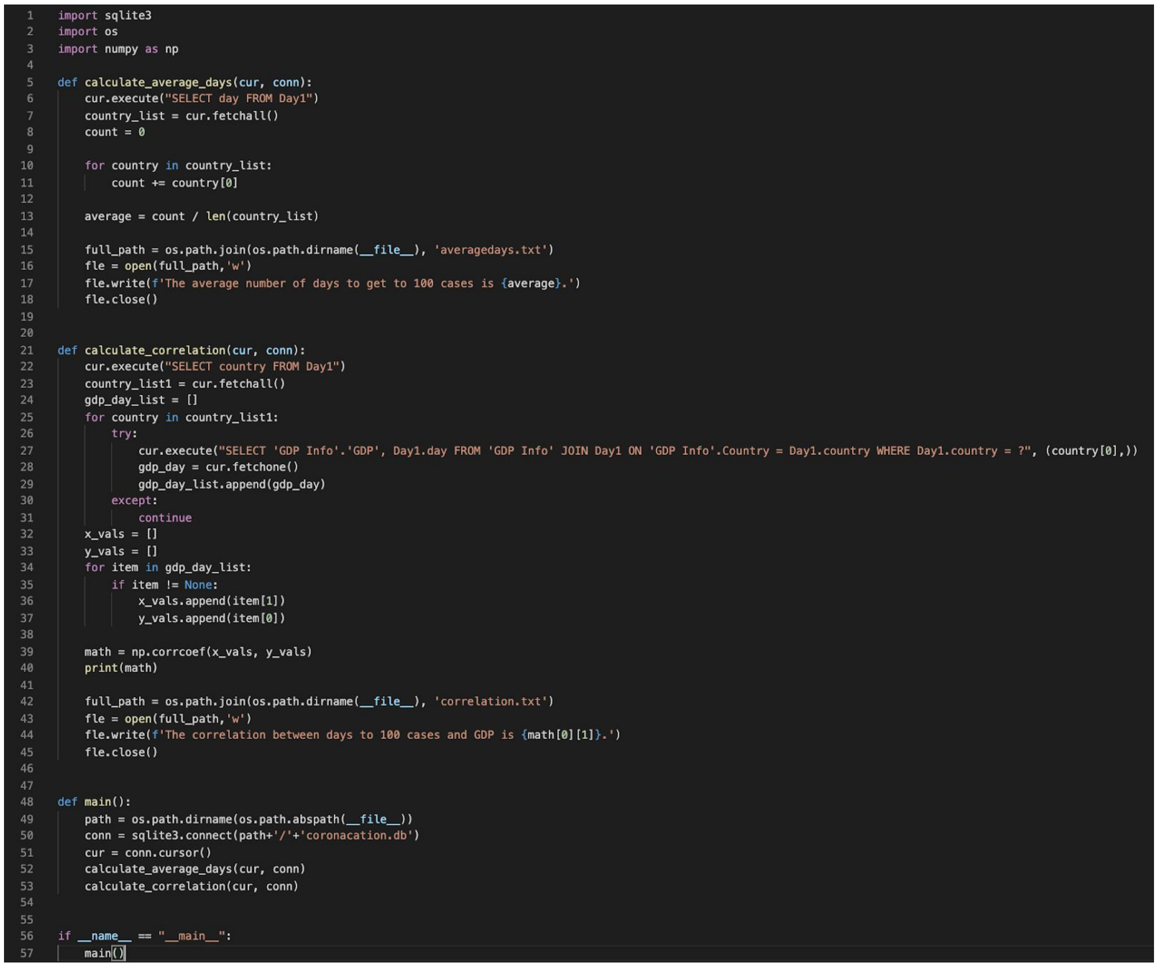
While trying to retrieve data from the website that provided information on hospital beds per one-thousand people, we encountered a problem in that it was not possible to grab the data from the table provided. We discussed the issue for a long time and tried many different things in Beautiful Soup, but eventually, we decided to change the comparison from hospital beds per one-thousand people to country GDP. So, in the end, we decided to compare country GDP to the speed of the spread in that particular country.

We also encountered a problem when trying to calculate the number of days from the first confirmed case to the 100th. The API we used had broken the cases reported in some countries down by county or territory. This meant that we were not able to just iterate through the response object for some countries. We solved this problem by creating a loop to check if the day had changed and if it hadn't, just adding the total for that county to the national count, and then only writing to the database if the day had changed.

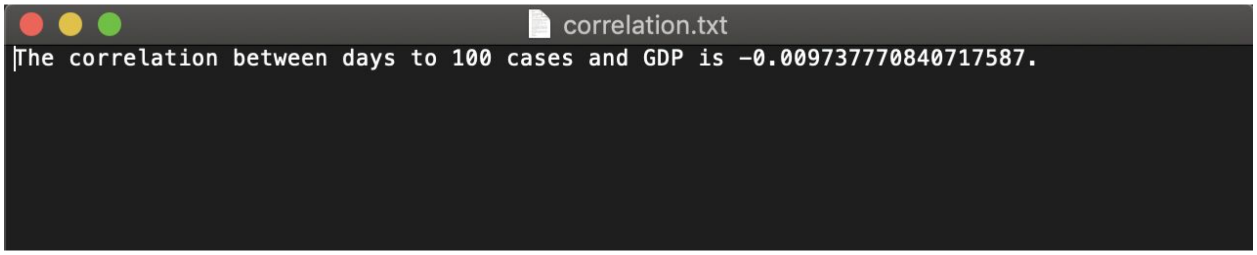
With regard to the correlation visualization, we encountered a problem with effectively displaying our information. Some countries were not able to be graphed as the shared keys in the database were not exact. For example, the API used "United States of America", while the website used "United States". In response to this, we chose not to graph these data points. In addition, some of the countries had a significantly higher GDP than the rest, therefore condensing the rest of the points to a small range which obscured the pattern we were hoping to see. To fix this problem, we limited the y-axis to 500,000 due to the fact that most of the countries were in this range.

We realized that we were using country names as keys instead of numeric keys which means we were splitting one table into two. To fix this we had to go back into our code and add a few functions to our files in order to create a new table in the database with country ids assigned to each country.

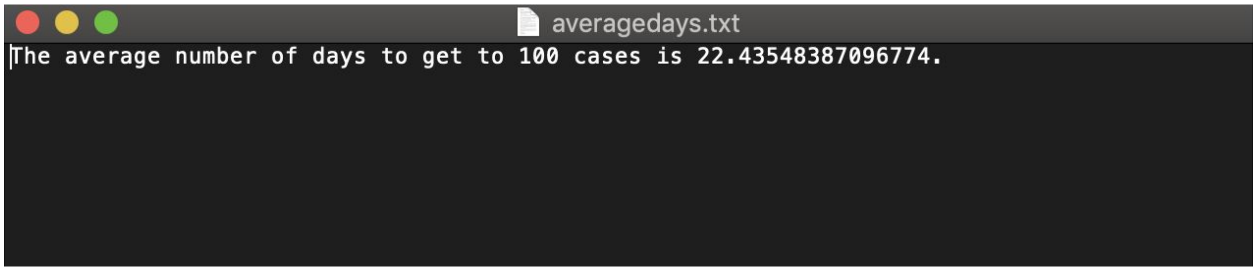
## Calculation File



```python
import sqlite3
import os
import numpy as np

def calculate_average_days(cur, conn):
    cur.execute("SELECT day FROM Day1")
    country_list = cur.fetchall()
    count = 0

    for country in country_list:
        count += country[0]

    average = count / len(country_list)

    full_path = os.path.join(os.path.dirname(__file__), 'averagedays.txt')
    fle = open(full_path,'w')
    fle.write(f'The average number of days to get to 100 cases is {average}.')
    fle.close()


def calculate_correlation(cur, conn):
    cur.execute("SELECT country FROM Day1")
    country_list1 = cur.fetchall()
    gdp_day_list = []
    for country in country_list1:
        try:
            cur.execute("SELECT 'GDP Info'.'GDP', Day1.day FROM 'GDP Info' JOIN Day1 ON 'GDP Info'.Country = Day1.country WHERE Day1.country = ?", (country[0],))
            gdp_day = cur.fetchone()
            gdp_day_list.append(gdp_day)
        except:
            continue
    x_vals = []
    y_vals = []
    for item in gdp_day_list:
        if item != None:
            x_vals.append(item[1])
            y_vals.append(item[0])

    math = np.corrcoef(x_vals, y_vals)
    print(math)

    full_path = os.path.join(os.path.dirname(__file__), 'correlation.txt')
    fle = open(full_path,'w')
    fle.write(f'The correlation between days to 100 cases and GDP is {math[0][1]}.')
    fle.close()


def main():
    path = os.path.dirname(os.path.abspath(__file__))
    conn = sqlite3.connect(path+'/'+'coronacation.db')
    cur = conn.cursor()
    calculate_average_days(cur, conn)
    calculate_correlation(cur, conn)


if __name__ == "__main__":
    main()
```

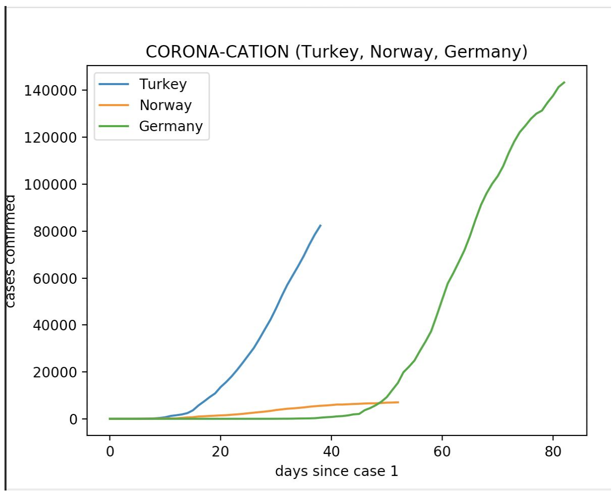


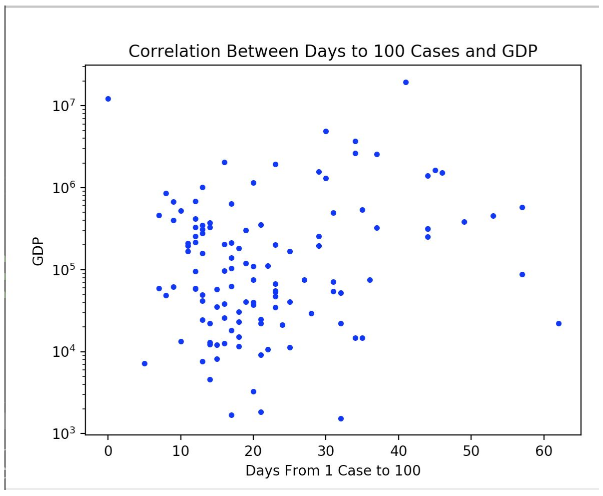
```
The correlation between days to 100 cases and GDP is -0.009737770840717587.
```

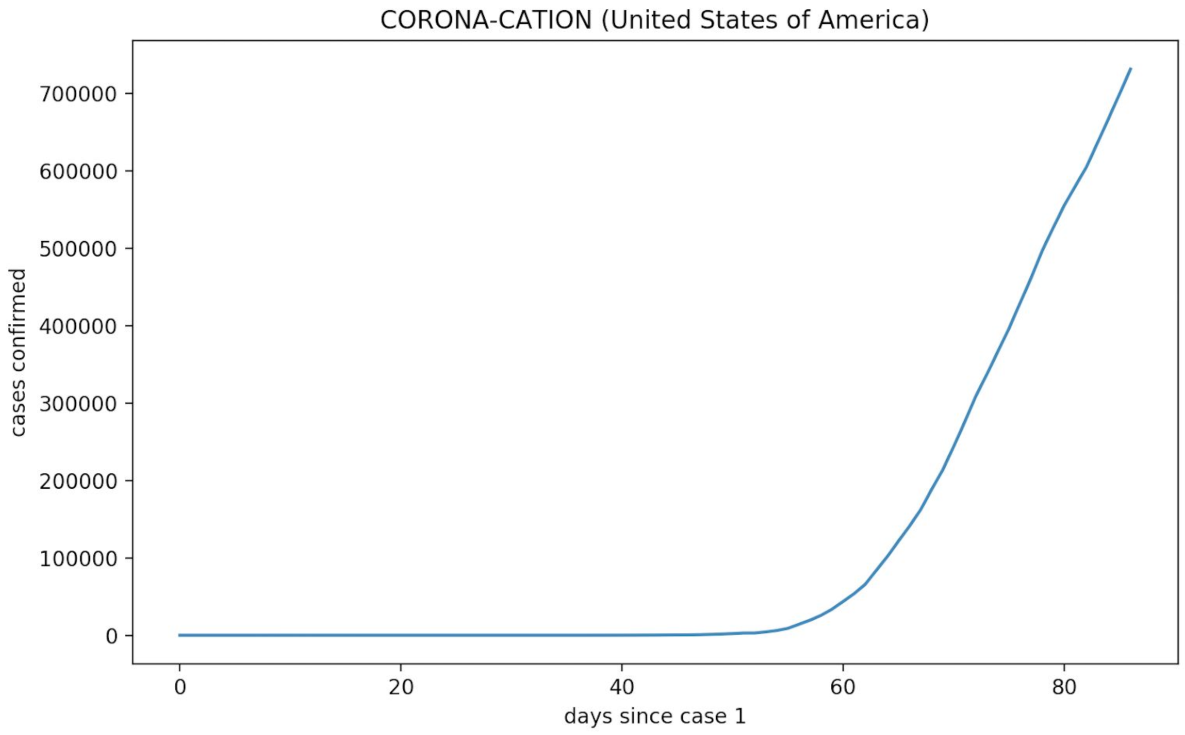


```
The average number of days to get to 100 cases is 22.43548387096774.
```

## Visualizations Created

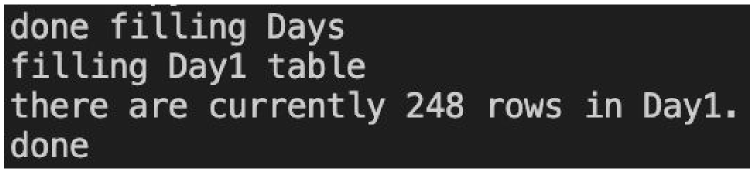






## Instructions for running code

1. Run API-db.py. The first time you run the code it should fill all of Days and add the first 20 entries to Day1. Each successive run will add 20 more rows to Day1. At the end of each run it prints out the number of rows currently in Days. Continue to run API-db.py until you see the printout that reads:



```
done filling Days
filling Day1 table
there are currently 248 rows in Day1.
done
```

   This means that you have completely filled the Days table and the Day1 table with the necessary data.

2. You next need to install fuzzywuzzy, as it is needed to run the visualizations. You can do this by opening your terminal and running pip install fuzzywuzzy[speedup]  . This installs the necessary packages to run fuzzywuzzy. For more information refer DataCamp (website) or the documentation found on this git repository (website).

3. At this point you can run Visualization_1.py. You have a few options for how you can run this code. If you would like to only see the progression for a single country, you can comment out line 104 and run only line 103 with the country of interest inputted as a string. If you would like to see multiple countries you can comment out line 59 and run line 60 with the list of countries as your input.

4. Next, run CountryGDP_DB.py until the output tells you there are 197 rows in the database.

5. Now you can run GDP_Corona_Scatterplot.py in order to get the visualization displaying the correlation between the number of days to 100 cases and country GDP. All you need to do is run this code once and it should return a scatter plot.

6. Lastly, if you would like to see the calculations we completed, run Calculation.py once. This code creates two files by the name of averagedays.txt and correlation.txt. If you open averagedays.txt, you will see the results of the calculation for the average amount of days for a country to document 100 cases. If you open correlation.txt, you will see the results of the calculation for the correlation coefficient for days to 100 cases vs. country GDP.

## Code Documentation

- API-db.py
    - fill_IDs()
        - Inputs: none
        - Outputs: None
        - The purpose of this function is to populate a table with all of the countries which have data on the API and assign each with a unique id number that can be used in the other tables.
    - fill_Days_table()
        - Inputs: a cursor and a connection
        - Outputs: None
        - The purpose of this function is to populate the table containing the day-by-day data. It does this by going into each country and adding every unique (country_id, day since first case, confirmed cases). This allows for the use of the built in min function when populating at Day1.
    - fill_Day1_table()
        - Inputs: a cursor and a connection
        - Outputs: None
        - The purpose of this function is to populate the table containing the days to 100 cases. It only sends 20 data points at a time and ensures that no duplicate data points are entered by utilizing insert or ignore. It finds the day by using the built in SQL min function and calling it on all of the Days entries which have a certain country id and have cases > 100.
    - main()
        - Inputs: None
        - Outputs: None
        - The purpose of this function is to specify which functions within the file should be called and the order in which they should be called. This is where one would specify the inputs for each of the functions they would like to use. We use this function to create the tables within our database which we later populate using other functions. This is also where we specify the database cursor and connection to be used in the other functions.
- Visualization_1.py
    - get_tups()
        - Inputs: a country name
        - Outputs: a list of x-values and a list of y-values
        - The purpose of this function is to take data from our database from the day-by-day table and then split the tuples into a list of day values and a list of case values that can later be graphed as a visualization.

    - plot_progessions()
        - Inputs: a country name and an optional top value
        - Outputs: a figure
        - The purpose of this function is to plot the progression of the number of cases in a given country as time progresses. The optional top value parameter can be used to specify the highest value that you would like on the y-axis, this can be helpful as slow early growth is easily dwarfed by later rapid growth.
    - plot_progessions_list()
        - Inputs: a list of country names and an optional top value
        - Outputs: a figure
        - The purpose of this function is to plot the progression of the number of cases in several countries as time progresses. The optional top value parameter can be used to specify the highest value that you would like on the y-axis, this can be helpful as slow early growth is easily dwarfed by later rapid growth and extreme numbers for one country can dwarf the smaller numbers of another country.
    - main()
        - Inputs: None
        - Outputs: None
        - The purpose of this function is to specify which functions within the file should be called and the order in which they should be called. This is where one would specify the inputs for each of the functions they would like to use.
- CountryGDP_DB.py
    - get_info_from_table()
        - Inputs: cursor and connection
        - Output: A dictionary named gdp_info that has the countries as the key and their GDP as the value
        - The purpose of the function is to create an iterable in order to go through while making tables in our database
    - get_country_id()
        - Inputs: cursor, connection, and country name
        - Outputs: if the  ratio of similarity is high enough, it will return a country ID...if not it returns zero
        - The purpose of this function is to assign a country ID to all of the countries in the table
    - fill_table()
        - Inputs: cursor, connection, dictionary with a country as key and GDP as the value
        - Output: None

        - The purpose of this function is to create and populate a table in our database with each row containing a country and its GDP. The dictionary we take as input is the output of get_info_from_table()
    - main()
        - Input: None
        - Output: None
        - The purpose of this function is to run the other functions in the file in a specified order. Additionally, this function specifies the database to populate and creates a table if it does not already exist in the database. It also establishes a cursor and a connection.
- GDP_Corona_Scatterplot.py
    - scatterplot()
        - Inputs: cursor and connection
        - Output: A scatterplot of days to 100 cases vs. country GDP
        - The purpose of  this function is to create a visualization that displays the correlation between the number of days to get to 100 cases of COVID-19 and country GDP
    - main()
        - Input: None
        - Output: None
        - The purpose of this function is to run the other functions in the file in a specified order. Additionally, this function specifies the database to pull information from and establishes a connection and cursor.
- Calculation.py
    - calculate_average_days()
        - Input: cursor and connection
        - Output: None
        - The purpose of this function is to write a text file named averagedays.txt to your local computer. This text file gives the average number of days to get to 100 cases of COVID-19
    - calculate_correlation()
        - Input: cursor and connection
        - Output: None
        - The purpose of this function is to write a text file named correlation.txt to your local computer. This text file gives the correlation between the number of days to 100 cases and country GDP.
    - main()
        - Input: None
        - Output: None
        - The purpose of this function is to run the other functions in the file in a specified order. Additionally, this function specifies the database to pull information from and establishes a connection and cursor.

**Resource documentation**

| Date | Issue Description | Location of Resource | Result |
|---|---|---|---|
| 4/11/20 | How to get all of the countries, without going through every entry | API documentation | There is a get countries api request built in to the API |
| 4/11/20 | Request for US data resulted in getting a value of >300 for days to 100 | Examined actual api response to determine what my code was counting | Discovered that data was being listed by county and rewrote a section to code to account for this |
| 4/11/20 | Trouble grabbing information from the table on a website. | Powerpoint from the lecture titled “Beautiful Soup” on class Canvas page | Problem remained unsolved |
| 4/11/20 | Trouble grabbing information from the table on the website | Beautifulsoup_solution.py from Discussion Section 8 on discussion section Canvas page | The problem remained unsolved, decided to go with a different website |
| 4/12/20 | Discovered we would have to use regex in order to get countries from the table on the website we used | Powerpoint from the lecture titled “Regular Expressions - Regex” | Was very helpful; was able to successfully grab each country from the website after referring to the table in the powerpoint |
| 4/16/20 | The larger values on the plots dwarfed the smaller values and obscured the patterns that could be seen | Matplotlib documentation | Added an optional topvalue to my function to allow for rescaling of the y-axis |
| 4/19/20 | The legend for the multiple country day-to-day plot was consistently getting put in the top-right corner which blocks the end of the behavior | Matplotlib documentation | Added an additional loc='upper left' parameter to my plot call |
| 4/20/20 | Realized that we were essentially splitting a table in 2 and should instead have a country_id table | API documentation, SQL documentation | Now we have multiple different tables |
| 4/20/20 | The website uses different country names than the API | Fuzzy-wuzzy documentation | Able to get fuzzy wuzzy to work on similar named countries |