SI 206 Final Project Report
Felicia Chen and Molly Adler

1. The goals for your project including what APIs/websites you planned to work with and what data you planned to gather (10 points)

Initially, the goal for our project was to compare COVID outcomes in Canada to COVID outcomes in other countries around the world. We wanted to compare the number of deaths and cases in each country. We planned on using one API that was specific to the Canada COVID statistics and compare the API data that had a variety of countries on COVID data. We wanted to gather deaths, cases, and vaccination rates for each country.

Planned APIs:
Covid-19 details per country API:

- https://github.com/M-Media-Group/Covid-19-API?tab=readme-ov-file

Covid-19 details across Canada:

- https://api.covid19tracker.ca/docs/1.0/overview

2. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather (10 points)

We ended up only comparing the COVID data from two countries: Canada and the US. We switched our plan because we realized that both APIs we had planned to use were difficult to work with and did not have great documentation. We used an API that had US-specific COVID data and another API that had COVID data from a variety of countries, specifically we used Canada's data from this API. We ended up with 2 tables in the database per API. The data we gathered included deaths, cases, recovered, active cases, and hospital data.

Used APIs:
Historic USA COVID Data

- https://covidtracking.com/data/api

COVID General API

- https://disease.sh/docs/#/

3. The problems that you faced (10 points)

When we were working on our separate APIs, we ran into the problem where some of the index and data were not aligned together when putting it into the database.

We also had a difficult time with our repository in general because of the multiple errors we had when using the git commands (especially with git pull). We think this happened because we worked on certain things at the same time by accident and did not update our dataset on both ends. We ended up having to create a new repository and transfer our code over twice but everything worked out smoothly after.

4. The calculations from the data in the database (i.e. a screenshot) (10 points)

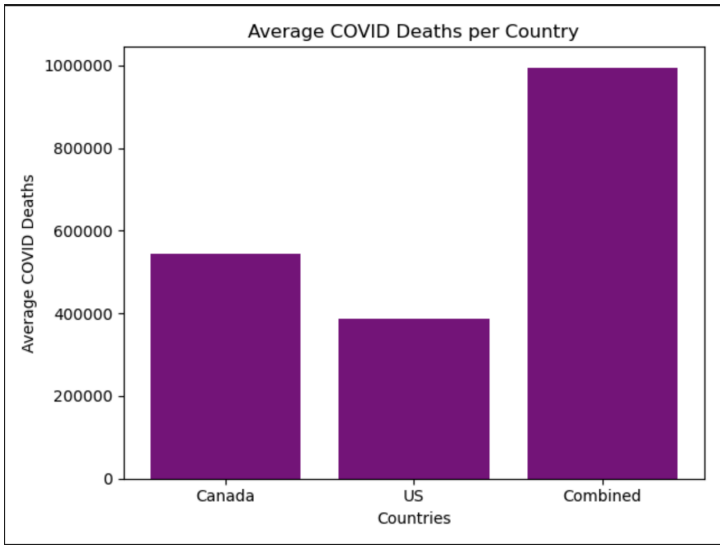
calculations_output.txt

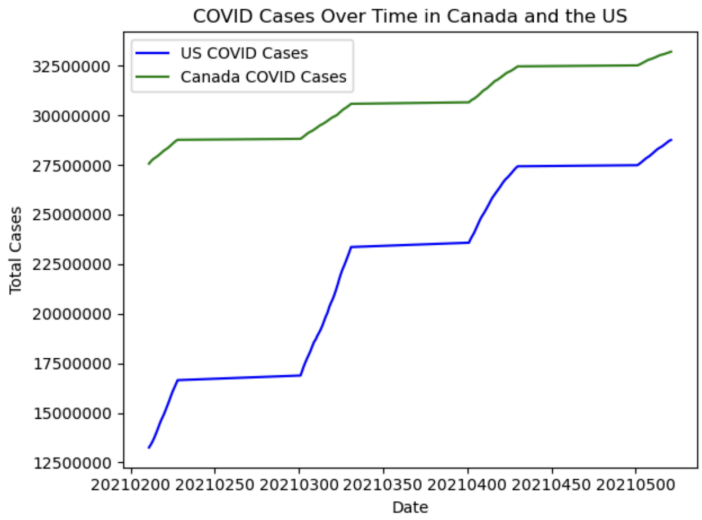
```
1   Canada average deaths: 544404.66
2   US average deaths: 387894.27
3   Combined average deaths (Canada + US): 994847.76
```

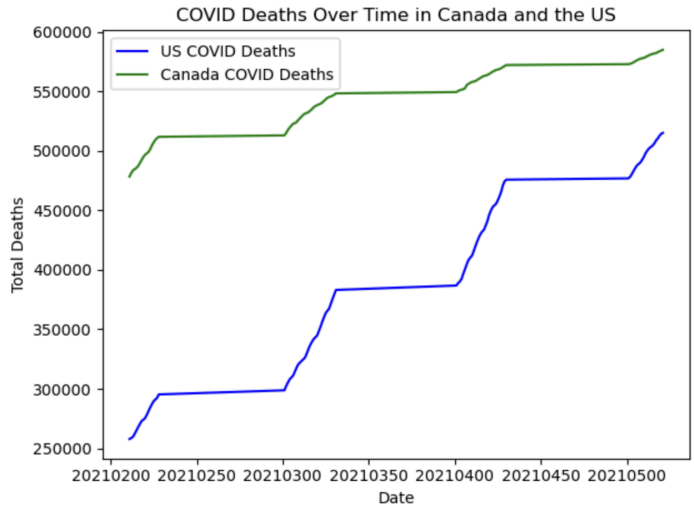
```python
def find_total_death_avg(dbpath):
    conn = sqlite3.connect(dbpath)
    cursor = conn.cursor()
    cursor.execute('''
        SELECT CanadaStats.deaths + DailyStats.death FROM CanadaStats JOIN DailyStats
                    ON CanadaStats.date = DailyStats.date
    ''')
    result_list = cursor.fetchall()
    conn.close()

    total = 0
    count = 0
    #print('RESULT LIST')
    #print(result_list)
    for item in result_list:
        #print(item)
        total += item[0]
        count += 1
    total_average = total/count

    print(f"total avg {total_average}")
    return total_average
```

```python
def find_country_death_avg(dbpath, txt_file, overall_avg):
    conn = sqlite3.connect(dbpath)
    cursor = conn.cursor()

    cursor.execute('SELECT deaths FROM CanadaStats')
    canada_list = cursor.fetchall()
    canada_total = 0
    canada_count = 0
    for item in canada_list:
        canada_total += item[0]
        canada_count += 1
    canada_avg = canada_total/canada_count

    cursor.execute('SELECT death FROM DailyStats')
    us_list = cursor.fetchall()
    us_total = 0
    us_count = 0
    for item in us_list:
        us_total += item[0]
        us_count += 1
    us_avg = us_total/us_count
```

5. The visualization that you created (i.e. screenshot or image file) (10 points)







Note: Dates are formatted YYYYMMDD

6. Instructions for running your code (10 points)
Step 1: Run each of the api files four times to populate each table with 100 data entries
Step 2: Run the vis_file.py and the calculations. Visualizations should print and show up on the screen

7. Documentation for each function that you wrote. This includes describing the input and output for each function (20 points)

<u>molly_api_file.py Functions:</u>
make_table:

- Input: dbpath → Database name
- Output: None

- Creates 2 tables in the SQL database: CanadaStats and CanadaCases
- CanadaStats stores date, cases, and deaths
- CanadaCases stores date, recovered, and active
- Commits the changes to the database

get_api_data:

- Input: None
- Output: data_list → A list of dictionaries with keys being the columns in the tables
- Sends a GET request to the API
- Extracts cases, deaths, and recovered from the JSON response
- Iterates through cases
- Reformats the dates
- Filters the dates
- Calculates active cases
- Puts data into a list

insert_data:

- Input: dbpath → name of database, data → list of dictionaries containing date, cases, and deaths, limit → max number of inputs to put into the database per run
- Output: None
- Connects to the database
- Inserts data into the CanadaStats table of the database
- Counts number of rows added to database and stops when the limit is reached
- Commits the changes to the database

insert_data2:

- Input: dbpath → name of database, data → list of dictionaries containing date, cases, and deaths, limit → max number of inputs to put into the database per run
- Output: None
- Connects to the database
- Inserts data into the CanadaCases table of the database
- Counts number of rows added to database and stops when the limit is reached
- Commits the changes to the database

<u>felicia_api_file.py Functions:</u>

Create_database:

- Input: None
- Output: None
- Creates a SQLite database named ‘covid_data.db’ with 2 tables titled DailyStats and HospitalData
- DailyStats stores the general COVID-19 statistics such as the total cases, deaths, and testing data

- HospitalData stores COVID-19 hospital-related statistics such as the number of those admitted to the hospital cases
- Commits the changes to the database

Insert_data_from_api:

- Input: api_url → the API endpoint URL to go fetch the COVID-19 data; limit → the maximum number of rows to insert in a single batch (default was 25)
- Output: None
- It fetches the COVID-19 data from the given API and inserts it into the DailyStats and HospitalData tables
- Inserts data into the database and also skips/print an error message if the API request had failed
- Only inserts up to the specified 'limit' rows (which was 100)

Populate_database:

- Inputs: api_url → the API endpoint URL to fetch the COVID-19 data; total_required → total number of records desired in the database; batch_size → the number of rows to fetch and insert per batch
- Output: None
- It fetches and inserts the data into the database until the total_required count is met
- Updates the DailyStatus and HospitalData tables incrementally

<u>vis_file.py Functions:</u>

find_total_death_avg:

- Input: dbpath → name of the database
- Output: total_average → the combined average deaths for the US and Canada
- Connects to the database
- Uses JOIN to match rows in CanadaStats and DailyStats by the date column
- Gets the sum of deaths for each date and stores it in a list
- Computes the average deaths by dividing the sum by the number of rows
- Returns the average

find_country_death_avg:

- Input: dbpath → database name, txt_file → file to write calculations to, overall_avg → combined average deaths
- Output: a dictionary containing the average COVID deaths for Canada and the US
- Connects to the database
- Queries the deaths from CanadaStats
- Iterates through the resulting list to find the sum of Canada Deaths and computes the average
- Queries the deaths from DailyStats
- Iterates through the resulting list to find the sum of US Deaths and computes the average
- Writes the calculations to a txt file

- Returns a dictionary containing both of the averages

avg_vis:

- Input: avg_dict → a dictionary containing the average deaths for Canada and the US, overall_avg → The combined average deaths
- Output: None
- Creates a list of averages for Canada, the US, and the combined average
- Plots a bar chart using these averages
- Adjusts the y-axis to use non-scientific notation
- Saves the chart and displays it

death_over_time:

- Input: dbpath → name of the database
- Output: None
- Connects to the database
- Queries deaths from CanadaStats
- Queries deaths from DailyStats
- Queries the date from CanadaStats
- Plots a line graph with lines for Canada and the US
- Adjusts the x-axis to use non-scientific notation
- Saves the chart and displays it

cases_over_time:

- Input: dbpath → name of the database
- Output: None
- Connects to the database
- Queries cases from CanadaStats
- Queries positive from DailyStats
- Queries the date from CanadaStats
- Plots a line graph with lines for Canada and the US
- Adjusts the x-axis to use non-scientific notation
- Saves the chart and displays it

8. You must also clearly document all resources you used. The documentation should be of the following form (20 points)

Date	Issue Description	Location of Resource	Result
12/15/24	Dates were formatted incorrectly in comparison with the other API	ChatGPT	I was able to reformat the dates so that they matched on all of the tables in the database
12/15/24	Was having trouble	ChatGPT / Piazza	I was able to figure

	getting more than 25 entries into the database		out how to use rowcount in order to not increase the count unless a row was added successfully
12/16/24	My graphs kept using scientific notation instead of the actual expanded numbers	ChatGPT	I was able to find a method to ensure that the numbers on my graphs were not written in scientific notation