Julio Berrocal Alvarez
Arez Aziz
Madison Jennings

SI206 Final Project Report
https://github.com/maddjenn/206_Final.git

Section 1: Planned Goals
Our team had planned to use three API's, the twitter API, the NewsAPI, and the Covidtracking API. In terms of data gathering, we planned to gather the number of Covid cases reported throughout a specific period, the number of tweets, mentioning Covid, posted around that same time period and the number of Covid-mentioning articles written along that same period.Our goal was to try to understand how peak moments of covid infections affected the number of tweets posted and the number of news articles published about Covid.

Section 2: Goals Achieved
After changing our API's accordingly, our team decided to still focus on the Covid/diseases topic. We decided to use beautiful soup to scrape a Covid tracking website instead of the twitter API, the NewsAPI, and a Covid tracking API. We were able to find very interesting information regarding Covid mortality rates in different countries, how prominent Covid is in the "disease" category of articles today, and positive test rates for Covid through different time spans. Our team developed a greater understanding of the disease, as well as further improvement of our knowledge of Beautiful Soup and API's.

Section 3: Problems Faced
Our team faced some unexpected problems and we had to adjust our goals accordingly. First, we had issues with the Twitter API becoming a paid subscription, rather than free for use. While the free API v1.1/2.0 lets you post Tweets and search users, a higher level subscription is required to interact with the .search_recent() and .search_all() functionalities, which allow you to search for Tweets using a particular query. Additionally, the News API's free version, limited the age of articles available for access to one month old, which meant we could only access relatively new articles only.

Section 4: Calculations from Data

```
1  For the News API calculations, we decided that it would be interesting to see how many titles include the word Covid.
2  We retrieved 100 articles about diseases, in general.
3  Below are the answers:
4  The number of titles that include covid is: 5.
5  The number of titles that do not include covid is: 95.
```

NewsAPI_Calculations.txt screenshot

```
Using information gathered from a public COVID-19 tracking database, we consolidated data on the total listed COVID-19 cases and deaths per country.
With this data, we were able to approximate COVID-19 survival rates per country.
Here are some statistics gathered using that data:

The average survival rate is 98.86530994310762%, with median of 99.13522428875561% and stdev of 0.9374131893423272.

The country with the highest survival rate is North Korea with a survival rate of 99.99844955165852%. North Korea's survival rate is about 1.1331396085509056% away from the mean.

The country with the lowest survival rate is Peru with a survival rate of 95.10833259416898%. Peru's survival rate is about 3.7569773489386336% away from the mean.
```

COVIDSoup_Calculations.txt screenshot

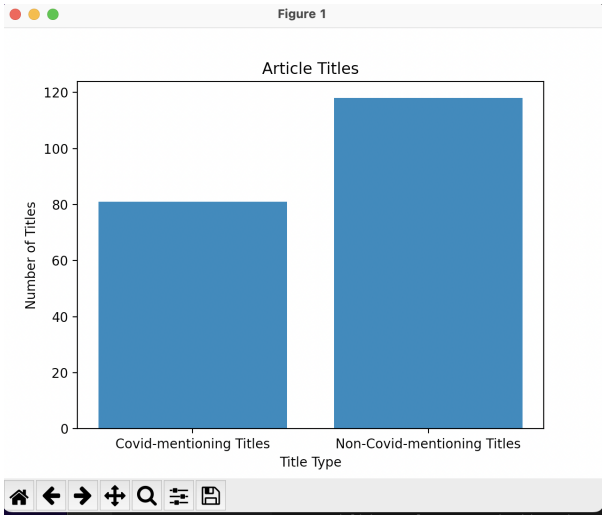
```
1  Using information gathered from a public COVID-19, we consolidated data on COVID-19 cases per day for 420 days preceding 03/07/2021.
2  Here are some statistics gathered using that data:
3
4  The average percent of positive test cases over the span of 420 days was 7.188179614940246%.
5
6  The maximum percent of positive test cases over the span of 420 days was 8.346544189710398%.
7
8  The minimum percent of positive test cases over the span of 420 days was 6.008920376409237%.
9
10 The average percent of positive test cases over the span of 226 days was 7.369696694662957%.
11
12 The maximum percent of positive test cases over the span of 226 days was 15.15813759888712%.
13
14 The minimum percent of positive test cases over the span of 226 days was 3.5667582486829836%.
15
```

CovidAPI_Calculations.txt screenshot

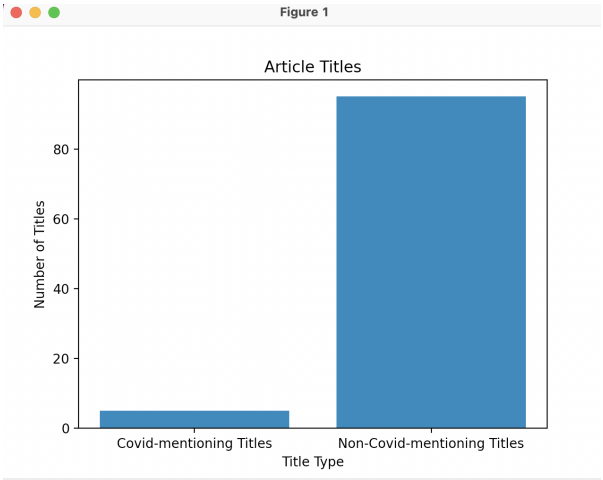
```
1  Using information gathered from a public YuGiOh API database, we calculated the division of card types in the database:
2  The percentage of Monster Cards out of the database totaled 90.9090909090909%.
3
4  The percentage of Spell Cards out of the database totaled 4.545454545454546%.
5
6  The percentage of Trap Cards out of the database totaled 4.545454545454546%.
7
```

YuGiOhAPI_Calculations.txt screenshot

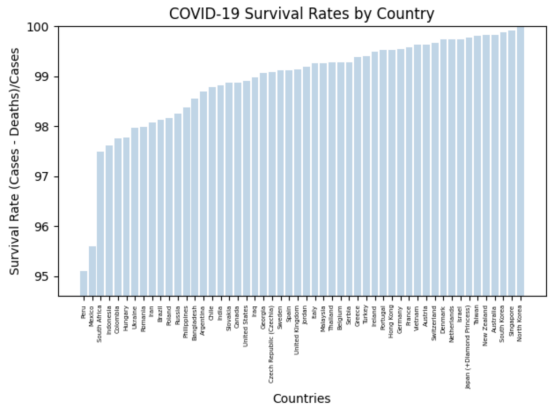
Section 5: Visualizations Created



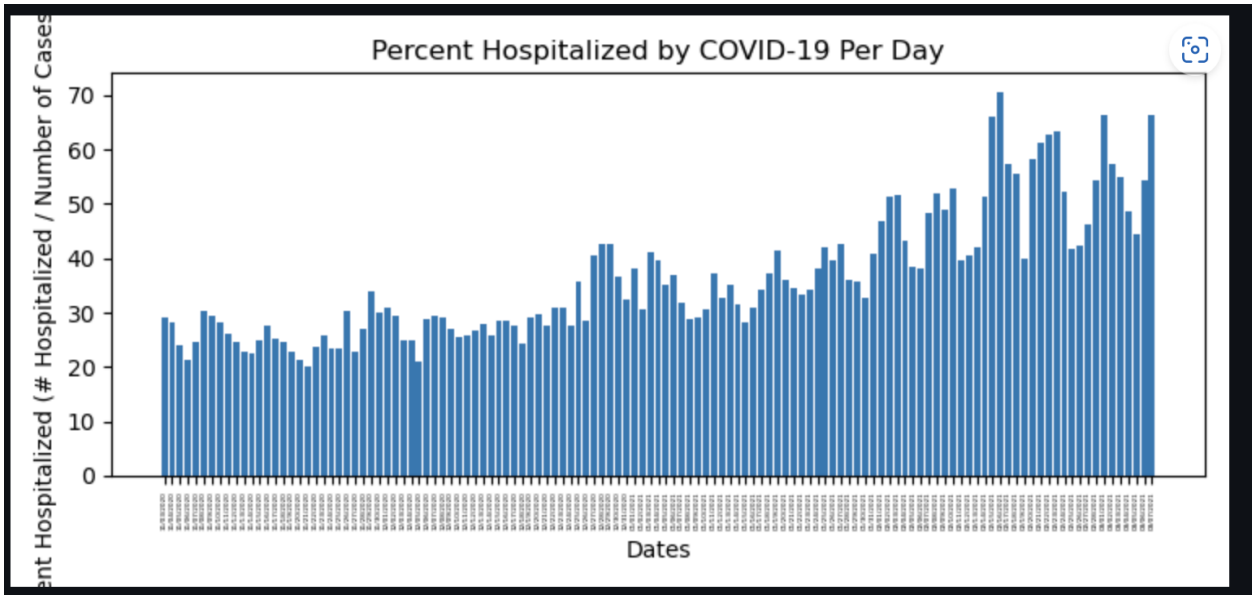
NewsAPI visualization 1 (covid-query included)



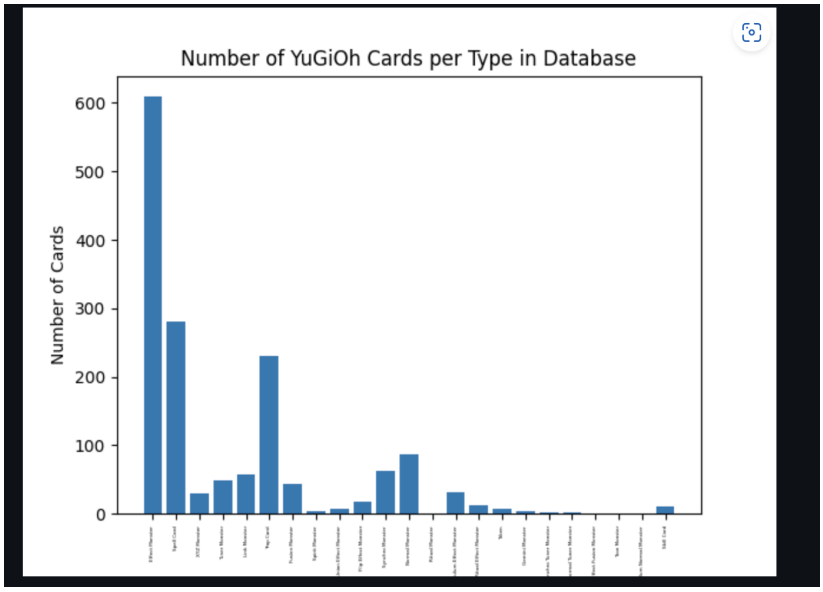
NewsAPI visualization 2 (no covid query)



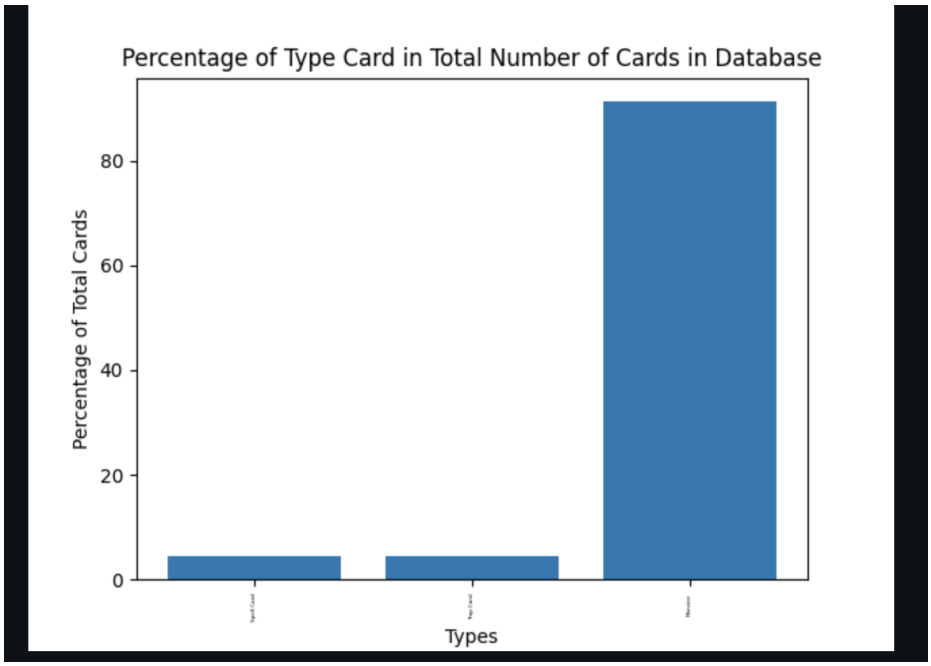
COVID_Soup Visualization, First 50 Countries



CovidAPI_Visualization



YuGiOh Visualization #1



YuGiOh Visualization #2

Section 6: Instructions for Running The Code

1. **Gathering Data into Database**
    a. In order to get the database filled with the appropriate data, the user would need to run the following files four times, each.
        i. News_api.py, COVID_Soup.py, Covid_api.py

2. **Getting Calculations and Visualizations**
    a. In order to get the calculation text files and respective visualizations, the user would need to run the following files once, each.
        i. NewsAPI_calc_vis.py, Covid_calculations.py, Covid_visualizations.py, COVID_Soup_CalcAndViz.py
3. **Extra Credit (YuGiOh)**
    a. In order to properly run our Extra Credit API, the user should follow the steps below.
        i. Run YuGiOh_api.py once to fill the database with data.
        ii. Run YuGiOh_api_CalcAndViz.py to create the calculations text file and obtain visualizations.

Section 7: Function Documentation

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

def open_database(db_name):
The open_database function takes as an input the database name in order to generate the path to the database. Afterwards, it uses the generated path to set up the database connection and, eventually, the cursor. The function **returns** the database connection and the cursor.

def get_newsapi_data(query):
The get_newsapi_data takes as an input a query string. This query is used to retrieve article information, based on the query's topic, using the get_everything method from the NewsApiClient library. After retrieving the data, the function creates individual article dictionaries that include the title, author (if provided) and description of each article. The article dictionaries are then appended into a list, which is then added into a dictionary, news_dic. The function **returns** news_dic.

def make_articles_table(cur, conn, data):
The make_articles_table takes as inputs the cursor and connections to a database, as well as data in dictionary form identical to news_dic. The function, first, creates a table "Articles", which contains columns for Title, Author and Descriptions. The table is only created if it doesn't already exist within the database. Then, the function proceeds to add 25 **unique** rows of data into the table, checking for duplicate titles to avoid duplicate articles being inserted. After 25 articles are added, the changes are committed. This function **does not return anything.**

def main():

The main function takes no inputs. It also **returns nothing.** However, inside of the function the following variables are set and functions are called (in the order listed):

cur, conn = open_database('FinalDatabase.db')
(optional) covid_data = get_newsapi_data('covid')
disease_data = get_newsapi_data('disease')
(optional) make_articles_table(cur, conn, covid_data)
make_articles_table(cur, conn, disease_data)

NewsAPI_calc_vis.py Functions:

def open_database(db_name):
The open_database function takes as an input the database name in order to generate the path to the database. Afterwards, it uses the generated path to set up the database connection and, eventually, the cursor. The function **returns** the database connection and the cursor.

def get_calculations(cur, conn):
The get_calculations function takes as inputs the cursor and connection to the database. It uses the cursor to select and count the number of titles that include (and do not include) covid. It then stores the data collected into a dictionary, "results", and closes the connection. The function **returns** the results dictionary.

def write_files(filename, dic):
The write_files function takes as inputs a filename and a dictionary with identical format to the "results" dictionary from get_calculations. The function opens, or creates, a file in write mode, and proceeds to run 2 different things into the file. First, it writes a small description of the data. Then it writes the findings from the "results" dictionary. This function **does not return anything.**

def visualization(dic):
The visualization function takes one input, a dictionary with identical format to the "results" dictionary from get_calculations. The function creates labels and counts from the data provided in the dictionary and plots them. Then it creates a plot Title, Y-axis title, and X-axis title. Finally, the function displays the visualization created. This function **does not return anything.**

def main():
The main function takes no inputs. It also **returns nothing.** However, inside of the function the following variables are set and functions are called (in the order listed):

cur, conn = open_database('FinalDatabase.db')

results = get_calculations(cur, conn)
write_file('NewsAPI_Calculations.txt', results)
visualizations(results)

COVID_Soup.py Functions:

def get_data(url):
This function operates primarily through Beautiful Soup, taking in a url, parsing the data table on that website, and returning a dictionary with the table's components (with the exception of the "continent" entry, a that was not necessary for our calculations). It uses Beautiful Soup to search for all <td> tags, then iterates through them in batches of 4 to gather all 4 components for the dictionary at a time (country, # of confirmed cases, # of deaths, continent (ignored)). It then **returns a dictionary** in the form {country: (confirmed, deaths)}.

def make_table(cur, conn, data):
The purpose of this function is to write out the data gathered in get_data() to the collective database. It does so by creating a table for Countries, if that table doesn't already exist. It then parses through each entry within data and attempts to insert them into the database until 25 unique countries have been inserted, or there are no more unique countries to insert. This function **does not return anything**.

def print_table(cur):
This is a function used for debugging. It fetches all entries under Countries in the database, then prints them. This function **does not return anything**.

COVID_Soup_CalcAndViz.py Functions:

def calculate(cur):
This function calculates the COVID-19 survival rate for each country in the database and stores it in a new dictionary ordered by that value. It does so by fetching all entries under Countries in the collective database, creating a new dictionary from that data in the form {country:survival rate}, then ordering it in ascending order by survival rate. It then **returns the new ordered dictionary**.

def visualize(data):
This function uses matplotlib to create a bar graph of the data returned from calculate(). The x-axis plots countries, and the y-axis demonstrates their % COVID-19 survival rate. Along the x-axis, the function rotates the text 90 degrees and spaces it out to create more room to see

country names and consolidate the graph. Along the y-axis, the function cuts off at .5% below the minimum value to create a better range for the graph. This function **returns nothing**, although it saves the graph as a .png file.

def write_calculations(file, data):
This function takes in a filename (which it either creates or opens in write mode) and the ordered data returned from the calculate() function, and writes out some calculations to that file, including the countries with the highest and lowest COVID-19 survival rates, and the mean, median, and standard deviation of the survival rates, among more. This function **returns nothing**, although it creates a new file and writes out to it.

def main():
This function runs the code. It connects to the database, sets up the url to gather data from, and calls the functions to gather data and write that table out to the database.

YuGiOh_api.py Functions:

def get_data():
This function operates through retrieving JSON data from the YuGiOh card database. For every card/entry in the file, it gathers a few variables in the form of a dictionary, **then returns a list of dictionaries** formatted as [{id, name, type, desc},].

def make_table(cur, conn, data):
The purpose of this function is to write out the data gathered in get_data() to the collective database. It does so by creating a table for YuGiOm, if that table doesn't already exist. It then parses through each entry within data and attempts to insert them into the database until 100 unique card entries have been inserted (since this is the extra credit, it did not specify running the function multiple times to gather up to 100 in groups of 25, but rather just 100 entries), or there are no more unique countries to insert. This function **does not return anything**.

YuGiOh_api_CalcAndViz.py Functions:

def visualize():
This function uses matplotlib to plot card data on a bar graph. It does so by parsing through the YuGiOh table in our database to create a list for each card type, as well as a list for how many times that card type appears. It then plots this data on a bar graph displaying the number of YuGiOh cards per type in the database. Along the x-axis, the function rotates the text 90 degrees

and spaces it out to create more room to see country names and consolidate the graph. This function **returns nothing**, although it saves this plot to a .png file.

def calculate_and_visualize(data):
This function performs calculations on the YuGiOh dataset, while also creating another visual. Foremost, it parses through the database and tallies up the number of cards per 3 overarching card types: Spell cards, Trap cards, and Monster cards (includes all types containing the word "Monster"). It then calculates the percentage that each card type makes up out of the entire dataset, and writes these calculations out to a file. Next, this function uses matplotlib to create a bar graph demonstrating this percentile data. The x-axis plots the 3 types of cards, and the y-axis demonstrates what % they make of all gathered cards. Along the x-axis, the function rotates the text 90 degrees and spaces it out to create more room to see card types and consolidate the graph. This function **returns nothing**, although it saves the graph as a .png file.

def main():
This function runs the code. It calls both aforementioned functions (visualize, calculte_and_visualize).

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

def get_covidapi_data():
This function gathers data from the Covid API and organizes it into 2 separate lists of dictionaries, each containing a shared column of date entries. It parses through each entry of the JSON file to create 2 independent dictionaries as entries for each day listed. It then **returns the 2 lists of dictionaries**, one containing general information on COVID statistics [{date, # of states reported, total # of positive cases, total # of pending cases, total # of negative cases, total # of deaths, # of total test results},], and another containing data on hospitalization and case daily differentials [{date, # hospitalized, # hospitalized currently, total # hospitalized, # hospitalized that day, # of positive cases that day, # of negative cases that day, # of test results that day},].

def make_tables(general_data, hospital_date, cur, con):
This function takes in the data gathered in get_covidapi_date() and writes it out to the collective database. For every entry in each list, it tracks entries added to the database using a count variable until 25 unique entries (delineated by the date variable) have been added. This function **does not return anything**.

def print_table(cur):
This is a function used for debugging. It fetches all entries under Countries in the database, then prints them. This function **does not return anything**.

Covid_calculations.py Functions:

This file has no functions. It connects to the collective database and selects all data from both dictionaries by joining them on the shared date entry, and sorting them in ascending order of those integers. It then parses each row of data in the joined table and creates two new dictionaries. The first is one that holds the percentage of positive cases spanning across all 420 days at each day, organized by [{date : % positive},]. The second is one that calculates the percentage of positive tests that day, organized by [{date : % positive},]. This **function returns nothing**, although it writes these out into a .txt file.

Covid_visualizations.py Functions:

This file has no functions. It connects to the collective database and selects all data from both dictionaries by joining them on the shared date entry, and sorting them in ascending order of those integers. Then, parsing through that data, it creates a list of dates and 'hospitalized', the hospitalized list holding calculations of the % of patients hospitalized each day. Next, it uses matplotlib to create a bar graph demonstrating this percentile data. The x-axis plots dates, and the y-axis demonstrates percentages hospitalized. Along the x-axis, the function rotates the text 90 degrees and spaces it out to create more room to see dates and consolidate the graph. This function **returns nothing**, although it saves the graph as a .png file.

Section 8: Resources Used

Date	Issue Description	Resource Used	Result (Solved?)
April 18, 2023	Set up API Connection	Documentation - News API	Resolved
April 18, 2023	How to count table rows?	COUNT(*) function - IBM Documentation	Resolved
April 19, 2023	How to create a visualization?	Pyplot tutorial — Matplotlib 3.7.1 documentation	Resolved
April 19, 2023	How to customize a pyplot bar plot?	Matplotlib pyplot function documentation	Resolved