Team Danmilia

SI 206 Final Project: Fall 2021

Team Members: Emilia O'Brien and Dani Vykydal

Link to repository: https://github.com/emiliaob/DanmiliaFinal206

Goals for This Project

Our original plan for this project was to use a COVID-19 data API for the UK as well as one for the United States and compare daily historical data from them. We thought that comparing the hospitalization versus vaccination data in the two places could provide important insights into the different public health techniques and their effectiveness.

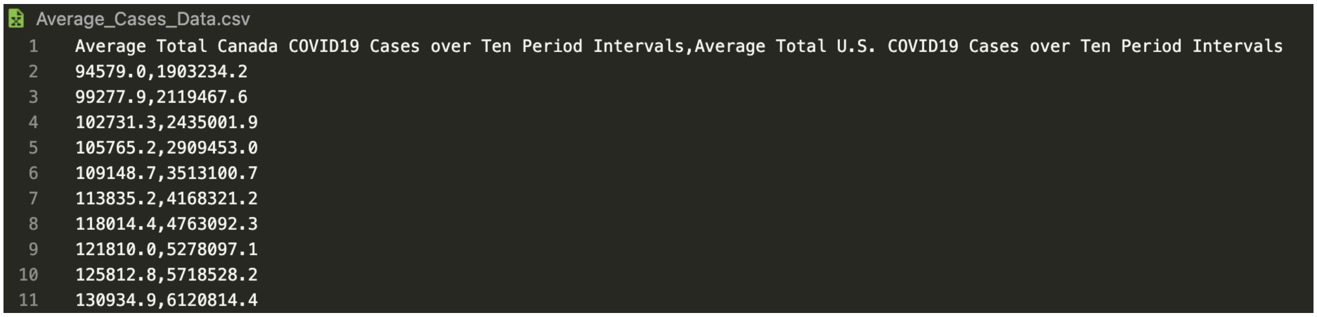
Achieved Goals

After doing more research into the APIs that we originally chose, our goals shifted. The documentation for the UK COVID-19 API was lacking, so we found an API for Canadian Covid data. We selected a 100 day period (particularly a 100 day period in the summer of 2020) and examined the differences in average deaths and average cases within the United States and Canada in ten day increments ranging from June 1, 2020 to September 8, 2020. The comparison and visualization of this data leads to further thought and study in the future of the effectiveness of different public health management techniques used in developed countries to fight COVID-19. However, we also recognize the shortcomings of our collected data- mainly that Canada's population is significantly less than that of the United States, the distribution of the population in both countries is different, and that fact that both countries are geographically unique.

Problems We Faced

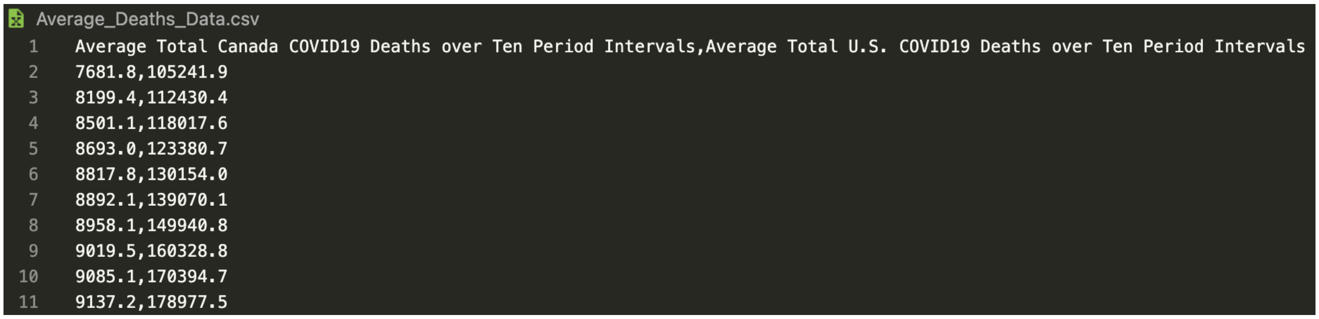
1. API Issues
    a. In the process of trying to collect data, our group cycled through many different APIs. Many COVID-19 APIs either did not have the data for the dates that we wanted it for, had little documentation, or had very stringent request limits. After trying to work with 4 different COVID-19 APIs we settled on the original US COVID-19 API and a Canadian COVID-19 API.
2. Github Collaboration
    a. Though by the end of the project, we were both adept collaborating efficiently in a shared Github repository, at the beginning it was difficult to navigate what times that we would be separately working on the code as well as what changes we would be making.

Calculation Files



Average_Cases_Data.csv

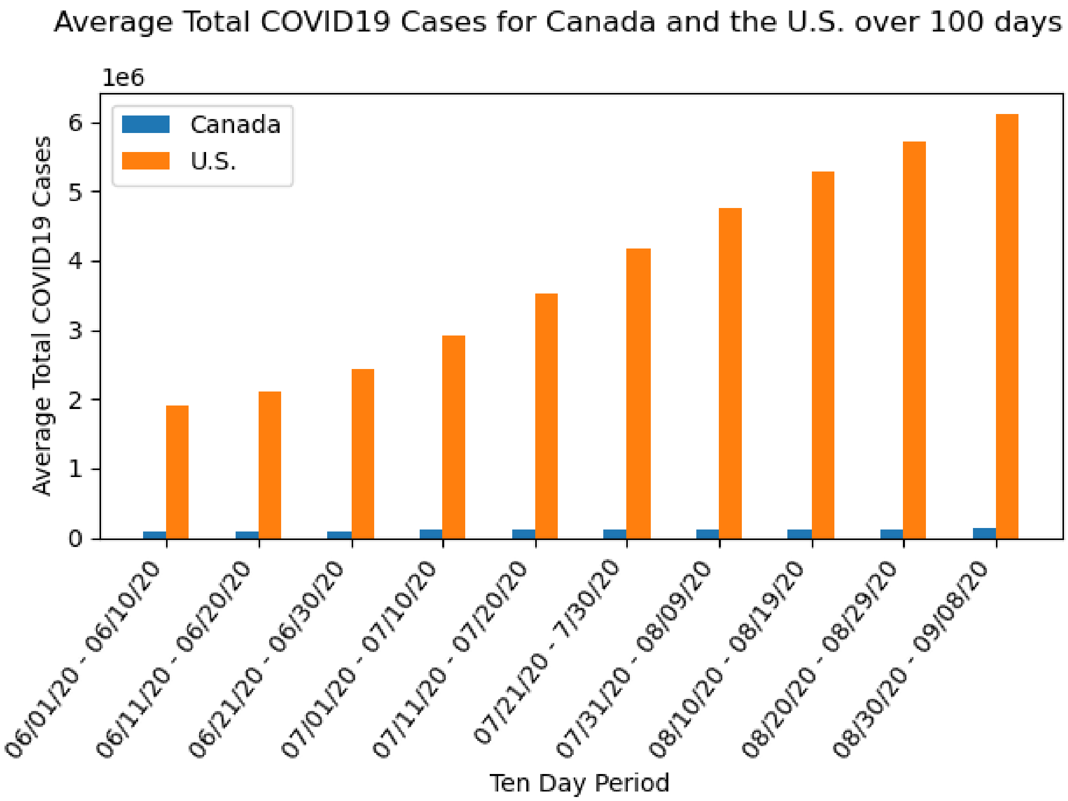
```
Average Total Canada COVID19 Cases over Ten Period Intervals,Average Total U.S. COVID19 Cases over Ten Period Intervals
94579.0,1903234.2
99277.9,2119467.6
102731.3,2435001.9
105765.2,2909453.0
109148.7,3513100.7
113835.2,4168321.2
118014.4,4763092.3
121810.0,5278097.1
125812.8,5718528.2
130934.9,6120814.4
```

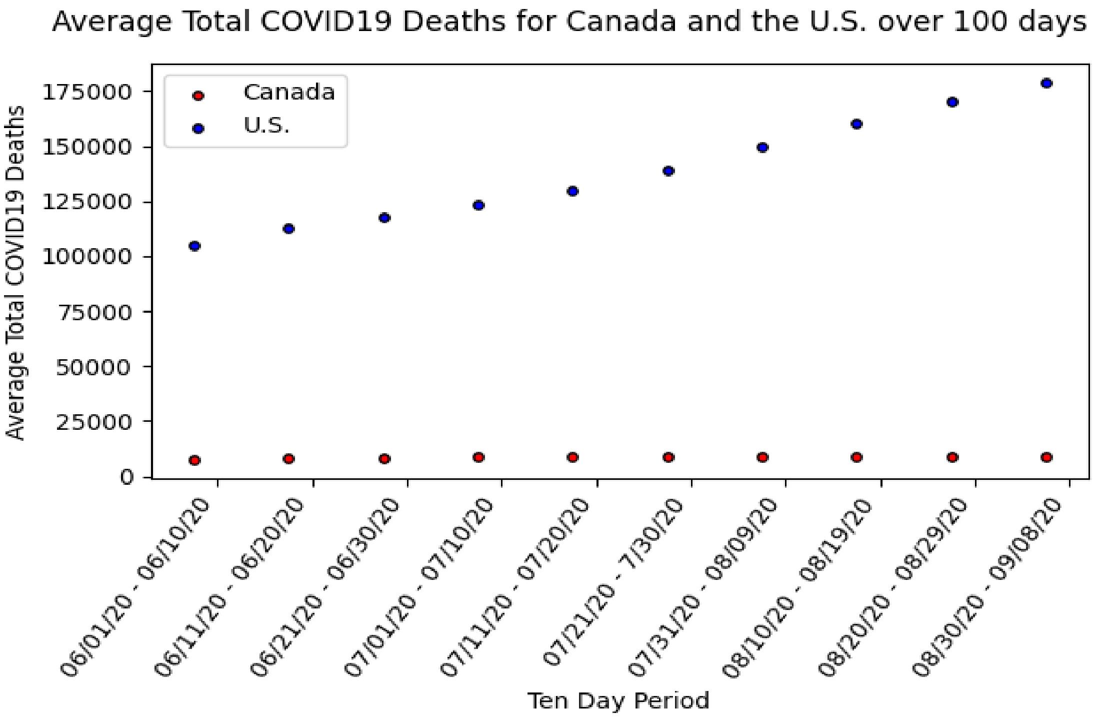


Average_Deaths_Data.csv

```
Average Total Canada COVID19 Deaths over Ten Period Intervals,Average Total U.S. COVID19 Deaths over Ten Period Intervals
7681.8,105241.9
8199.4,112430.4
8501.1,118017.6
8693.0,123380.7
8817.8,130154.0
8892.1,139070.1
8958.1,149940.8
9019.5,160328.8
9085.1,170394.7
9137.2,178977.5
```

Visualizations





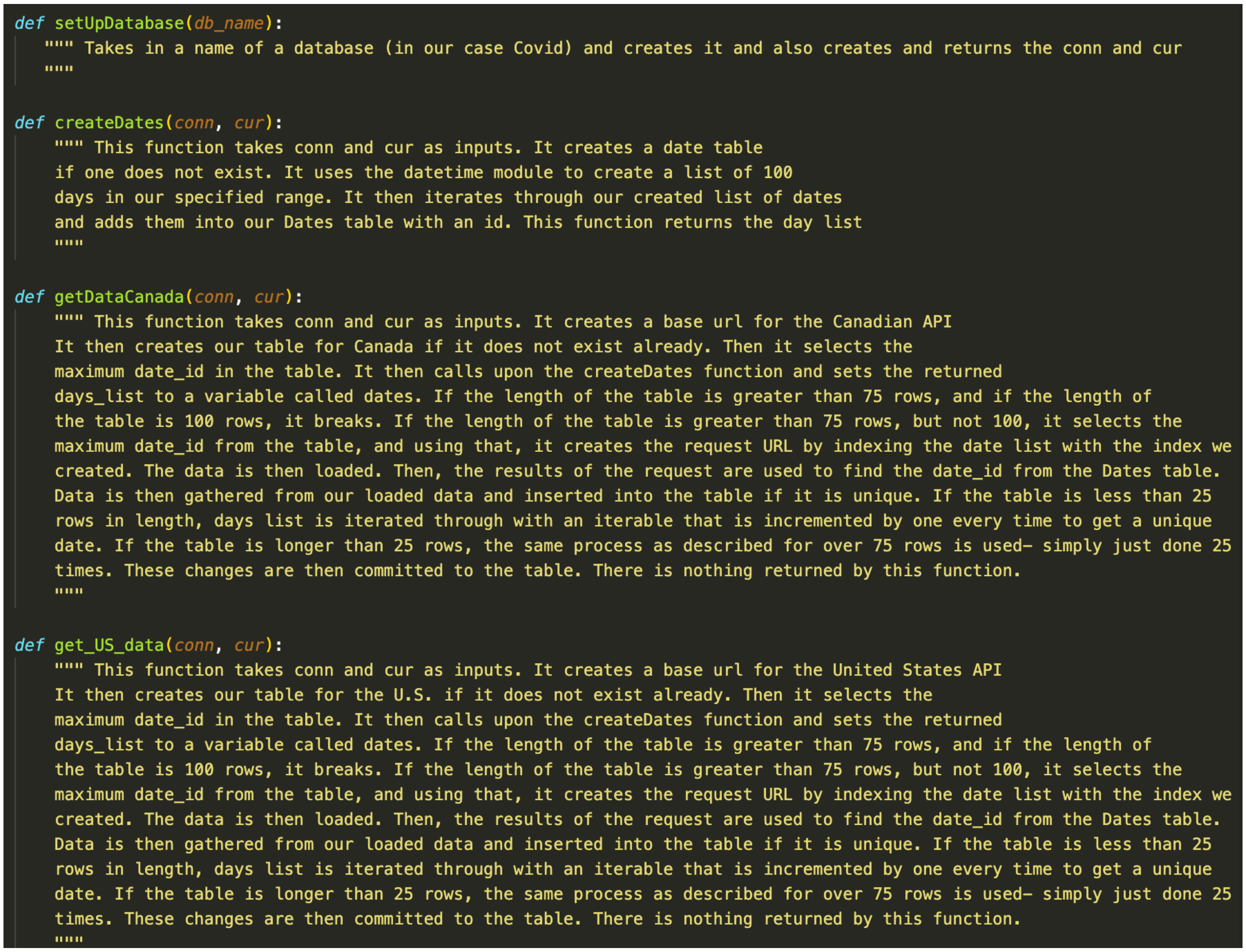
Instructions for Running Code

EO Final Project 206.py

Scroll to the bottom for the main. Main must run at least 4 times to get 100 entries in the database (adds 25 entries each time). Average calculations will be performed and visuals will show after 100 entries have been added to the database.

Function Documentation

EO Final Project 206.py

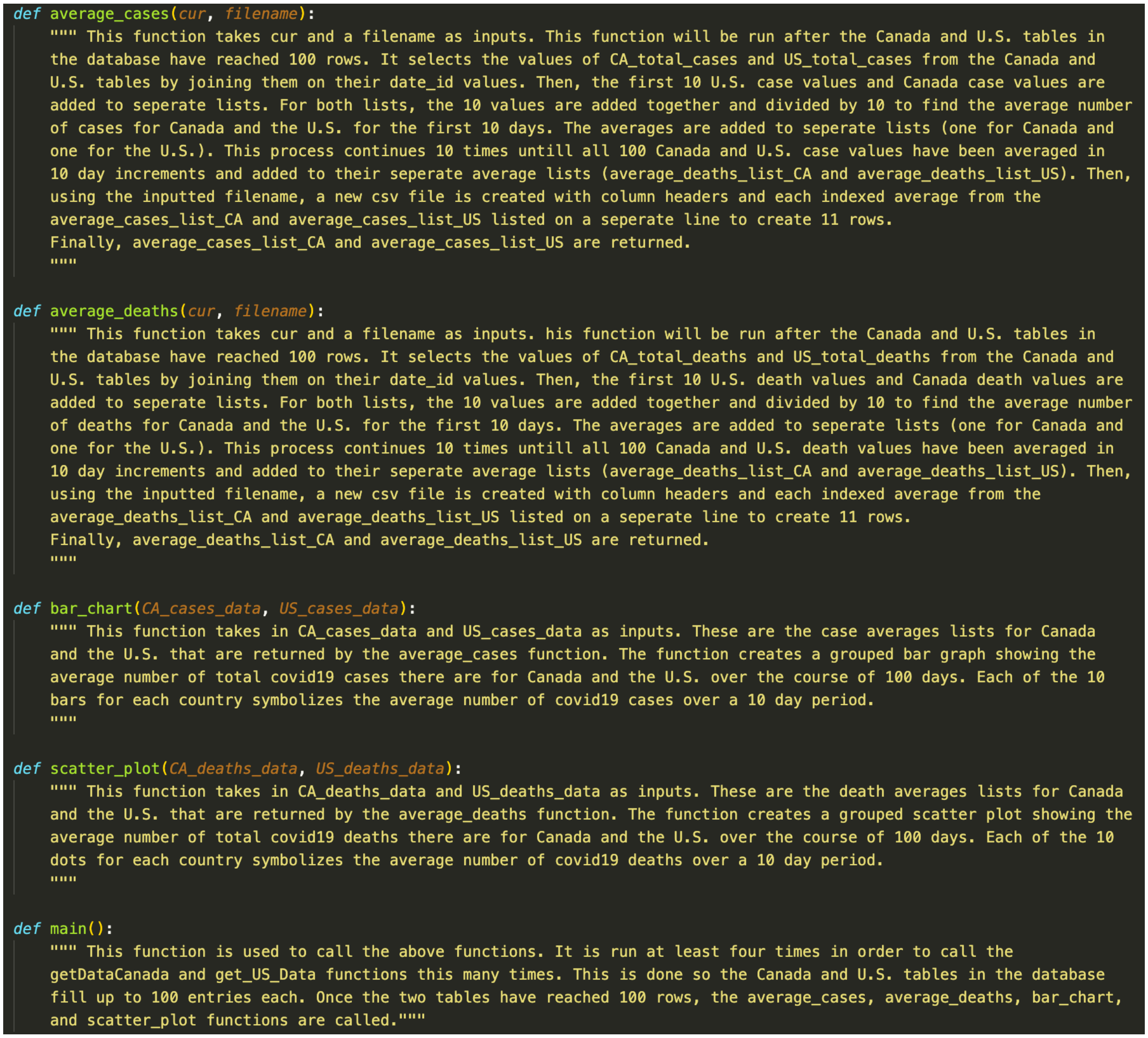


```python
def setUpDatabase(db_name):
   """ Takes in a name of a database (in our case Covid) and creates it and also creates and returns the conn and cur
   """

def createDates(conn, cur):
    """ This function takes conn and cur as inputs. It creates a date table
    if one does not exist. It uses the datetime module to create a list of 100
    days in our specified range. It then iterates through our created list of dates
    and adds them into our Dates table with an id. This function returns the day list
    """

def getDataCanada(conn, cur):
    """ This function takes conn and cur as inputs. It creates a base url for the Canadian API
    It then creates our table for Canada if it does not exist already. Then it selects the
    maximum date_id in the table. It then calls upon the createDates function and sets the returned
    days_list to a variable called dates. If the length of the table is greater than 75 rows, and if the length of
    the table is 100 rows, it breaks. If the length of the table is greater than 75 rows, but not 100, it selects the
    maximum date_id from the table, and using that, it creates the request URL by indexing the date list with the index we
    created. The data is then loaded. Then, the results of the request are used to find the date_id from the Dates table.
    Data is then gathered from our loaded data and inserted into the table if it is unique. If the table is less than 25
    rows in length, days list is iterated through with an iterable that is incremented by one every time to get a unique
    date. If the table is longer than 25 rows, the same process as described for over 75 rows is used- simply just done 25
    times. These changes are then committed to the table. There is nothing returned by this function.
    """

def get_US_data(conn, cur):
    """ This function takes conn and cur as inputs. It creates a base url for the United States API
    It then creates our table for the U.S. if it does not exist already. Then it selects the
    maximum date_id in the table. It then calls upon the createDates function and sets the returned
    days_list to a variable called dates. If the length of the table is greater than 75 rows, and if the length of
    the table is 100 rows, it breaks. If the length of the table is greater than 75 rows, but not 100, it selects the
    maximum date_id from the table, and using that, it creates the request URL by indexing the date list with the index we
    created. The data is then loaded. Then, the results of the request are used to find the date_id from the Dates table.
    Data is then gathered from our loaded data and inserted into the table if it is unique. If the table is less than 25
    rows in length, days list is iterated through with an iterable that is incremented by one every time to get a unique
    date. If the table is longer than 25 rows, the same process as described for over 75 rows is used- simply just done 25
    times. These changes are then committed to the table. There is nothing returned by this function.
    """
```



```python
def average_cases(cur, filename):
    """ This function takes cur and a filename as inputs. This function will be run after the Canada and U.S. tables in
    the database have reached 100 rows. It selects the values of CA_total_cases and US_total_cases from the Canada and
    U.S. tables by joining them on their date_id values. Then, the first 10 U.S. case values and Canada case values are
    added to seperate lists. For both lists, the 10 values are added together and divided by 10 to find the average number
    of cases for Canada and the U.S. for the first 10 days. The averages are added to seperate lists (one for Canada and
    one for the U.S.). This process continues 10 times untill all 100 Canada and U.S. case values have been averaged in
    10 day increments and added to their seperate average lists (average_deaths_list_CA and average_deaths_list_US). Then,
    using the inputted filename, a new csv file is created with column headers and each indexed average from the
    average_cases_list_CA and average_cases_list_US listed on a seperate line to create 11 rows.
    Finally, average_cases_list_CA and average_cases_list_US are returned.
    """

def average_deaths(cur, filename):
    """ This function takes cur and a filename as inputs. his function will be run after the Canada and U.S. tables in
    the database have reached 100 rows. It selects the values of CA_total_deaths and US_total_deaths from the Canada and
    U.S. tables by joining them on their date_id values. Then, the first 10 U.S. death values and Canada death values are
    added to seperate lists. For both lists, the 10 values are added together and divided by 10 to find the average number
    of deaths for Canada and the U.S. for the first 10 days. The averages are added to seperate lists (one for Canada and
    one for the U.S.). This process continues 10 times untill all 100 Canada and U.S. death values have been averaged in
    10 day increments and added to their seperate average lists (average_deaths_list_CA and average_deaths_list_US). Then,
    using the inputted filename, a new csv file is created with column headers and each indexed average from the
    average_deaths_list_CA and average_deaths_list_US listed on a seperate line to create 11 rows.
    Finally, average_deaths_list_CA and average_deaths_list_US are returned.
    """

def bar_chart(CA_cases_data, US_cases_data):
    """ This function takes in CA_cases_data and US_cases_data as inputs. These are the case averages lists for Canada
    and the U.S. that are returned by the average_cases function. The function creates a grouped bar graph showing the
    average number of total covid19 cases there are for Canada and the U.S. over the course of 100 days. Each of the 10
    bars for each country symbolizes the average number of covid19 cases over a 10 day period.
    """

def scatter_plot(CA_deaths_data, US_deaths_data):
    """ This function takes in CA_deaths_data and US_deaths_data as inputs. These are the death averages lists for Canada
    and the U.S. that are returned by the average_deaths function. The function creates a grouped scatter plot showing the
    average number of total covid19 deaths there are for Canada and the U.S. over the course of 100 days. Each of the 10
    dots for each country symbolizes the average number of covid19 deaths over a 10 day period.
    """

def main():
    """ This function is used to call the above functions. It is run at least four times in order to call the
    getDataCanada and get_US_Data functions this many times. This is done so the Canada and U.S. tables in the database
    fill up to 100 entries each. Once the two tables have reached 100 rows, the average_cases, average_deaths, bar_chart,
    and scatter_plot functions are called."""
```

All Resources Used

Date	Issue Description	Location of Resource	Result
12/01/21	We had issues finding suitable APIs for COVID-19 data	https://github.com/public-apis/public-apis	This public API list allowed us to find both of the final APIs that we used
12/03/21	Making a list of dates in the correct format for both API calls	https://stackoverflow.com/questions/993358/creating-a-range-of-dates-in-python	This stack overflow post offered some guidance and introduced us to the datetime module but did not completely help us solve the issue
12/03/21	Making a list of dates in the correct format for both API calls	https://www.kite.com/python/answers/how-to-create-a-range-of-dates-in-python	This Kite article (in tandem with the resource above) taught us how to create a range of dates in the format that we needed
12/07/21	Creating a database	SI 206 Homework 7	Looking at our completed homework 7, we found helpful examples of how to open a database- we successfully did so.
12/11/21	Creating a csv out file and adding lines to it	SI 206 Project 1	Looking at our completed project 1 assignment, we gained insight on how to create and write data to a csv file
12/12/21	Creating a grouped bar chart with matplotlib	SI 206 Lecture 20 - Matplotlib, https://matplotlib.org/stable/gallery/lines_bars_and_markers/barchart.html	This SI 206 lecture and matplotlib website page helped us with the basics of creating a grouped bar chart
12/12/21	Moving the location of a matplotlib figure title or label	https://matplotlib.org/3.3.0/gallery/text_labels_and_annotations/titles_demo.html	This matplotlib website page helped us with how to relocate our figure

			titles and/or labels with “pad” and “loc” elements
12/12/21	Creating a matplotlib scatter plot with multiple sets of data represented	https://stackoverflow.com/questions/4270301/matplotlib-multiple-datasets-on-the-same-scatter-plot	This stack overflow post taught us how to display two sets of data on the same plot and distinguish the two.
12/12/21	Styling visualizations to be more visually pleasing	https://www.youtube.com/watch?v=zZZ_RCwp49g\	This video was used to gain some aesthetic advice to make the points on the scatter plot more distinct.