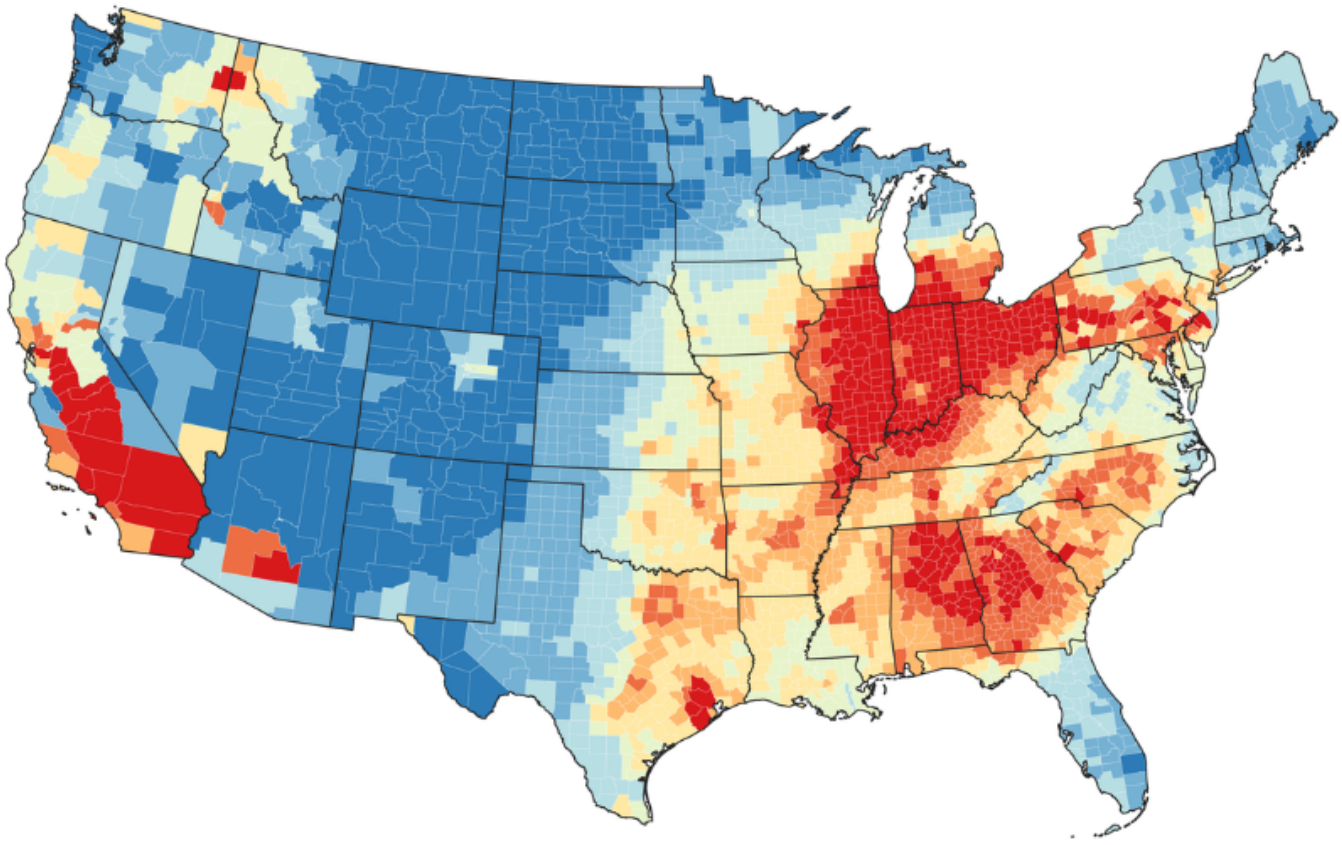
SI 206 Final Project

Using OpenWeatherMap and Zippopotam.us APIs

GitHub repository link: https://github.com/rrpatell/si_final.git



Presented by Raspberries

Aidana Tuyakbayeva and Radhika Patel

April 21st, 2023

Initial Goals

Our initial plan was to collect data from the Spotify API (base URL: https://api.spotify.com/) and the Songkick API (base URL: https://www.songkick.com/). We intended to gather information about music events, including genre, artists, date, time, and location, from Songkick. Additionally, we planned to obtain data on user playlists and listening history from Spotify.

Achieved Goals

Due to the unavailability of the Songkick API, we shifted our focus from music to air quality, decided to use more accessible APIs and set out to accomplish the following goals:

1. Utilize the Zippopotam.us API to gather data on postcode, country, country abbreviation, places, place name, longitude, state, state abbreviation, and latitude.
    a. Radhika randomly sampled 250 zip codes in the US, yielding around 120 valid zip codes. She identified the 10 states with the highest number of zip codes, implying a high population density, and/or many cities and towns.
2. Use the OpenWeatherMap API to determine the average ozone concentration in the identified 10 states.
    a. Aidana analyzed the average ozone concentration by state based on the state capital, and also analyzed the average ozone concentration by date since April 1st to assess if states with high population density and/or many cities and towns have higher average ozone levels.

Problems Faced

- Initially, Radhika encountered a challenge as Songkick's API required a payment. After exploring various music-based APIs, she was unable to find a suitable alternative. Consequently, we began researching other APIs and came across two interesting options. Radhika found an API that provides information on different zip codes, which offers a broad scope for data collection. Aidana found an API that focuses on air quality, which could potentially correlate with Radhika's API.
- The first issue Radhika faced was that her API took a long time to access so she could not access the API and store the data in the same file. She then decided it would be best to access the API and output a dictionary of the data as a JSON file. She decided to research how to do this by looking up different json functions on Google.
- Radhika also struggled with entering 25 data entries at a time into the database. She looked online for hints and found that SQL has a function where you can see how many entries are in the database. Researching different tactics online helped her find the answers to her issues.
- With Zippopotam.us, the best approach was to analyze and visualize information on the state level. However, air quality data is not reported on the state level in the OpenWeather API, so Aidana needed to find a way to present air quality in different states. Ultimately, she decided to use air quality data for each state's capital.
- Aidana initially struggled with setting time filters for OpenWeatherMap API requests. The documentation for the API showed the examples that were hard to understand because they were based on a seemingly random set of digits. After reading the documentation more thoroughly, Aidana realized that those sets of digits represented dates in the Unix format. Using an online Unix-to-date converter, Aidana managed to set the desired filters for API requests.

Calculations

data_out.csv

```
State,Zipcode_Count
Alabama,4
Arizona,1
Arkansas,4
California,11
Colorado,4
Florida,7
Georgia,5
Hawaii,1
Idaho,1
Illinois,7
Indiana,5
Iowa,5
Kansas,4
Kentucky,2
Louisiana,4
Maryland,4
Michigan,7
Minnesota,6
Mississippi,3
Missouri,8
Montana,4
Nebraska,3
Nevada,1
New Mexico,2
North Carolina,4
North Dakota,3
Ohio,5
Oklahoma,4
Oregon,3
South Carolina,3
South Dakota,4
Tennessee,6
Texas,14
Unknown,1
Virginia,7
Washington,1
West Virginia,6
Wisconsin,5
Wyoming,1
```

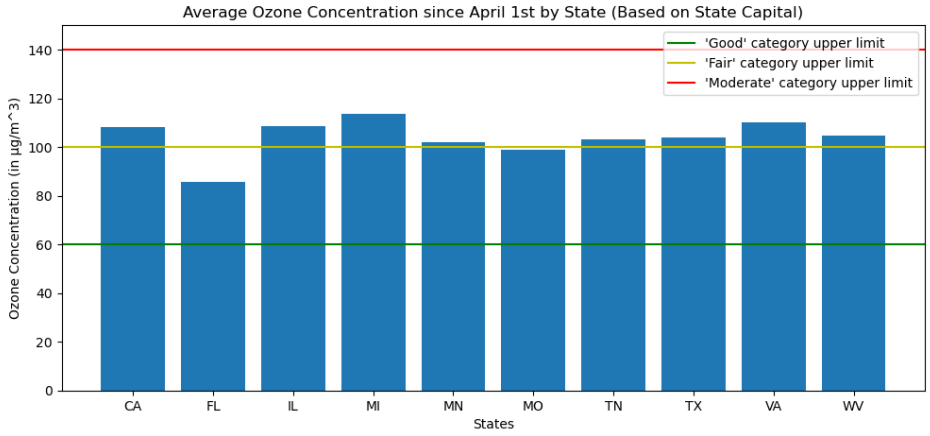
calcByDate.csv

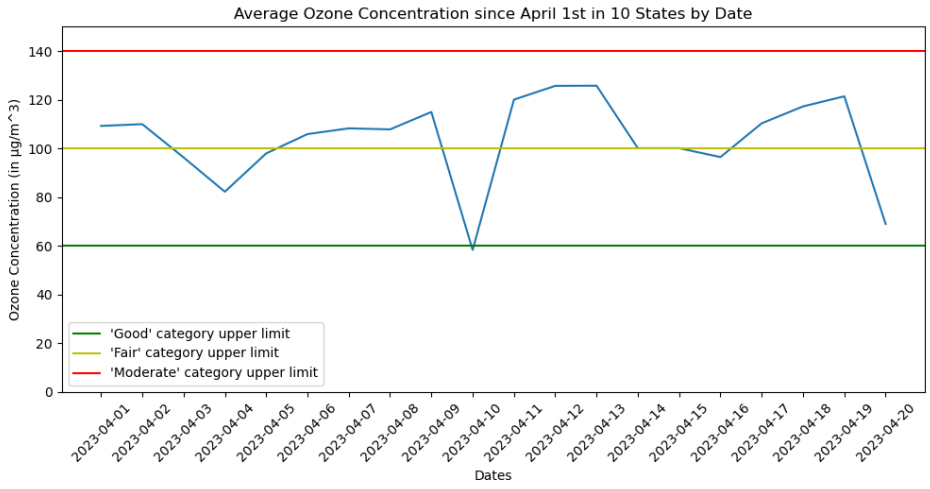
```
Date,Avg Ozone Levels

2023-04-01,109.29

2023-04-02,110.01

2023-04-03,96.35

2023-04-04,82.26

2023-04-05,97.99

2023-04-06,105.93

2023-04-07,108.29

2023-04-08,107.86

2023-04-09,115.01

2023-04-10,58.35

2023-04-11,120.09

2023-04-12,125.74

2023-04-13,125.81

2023-04-14,100.14

2023-04-15,100.14

2023-04-16,96.49

2023-04-17,110.36

2023-04-18,117.3

2023-04-19,121.45

2023-04-20,69.02
```

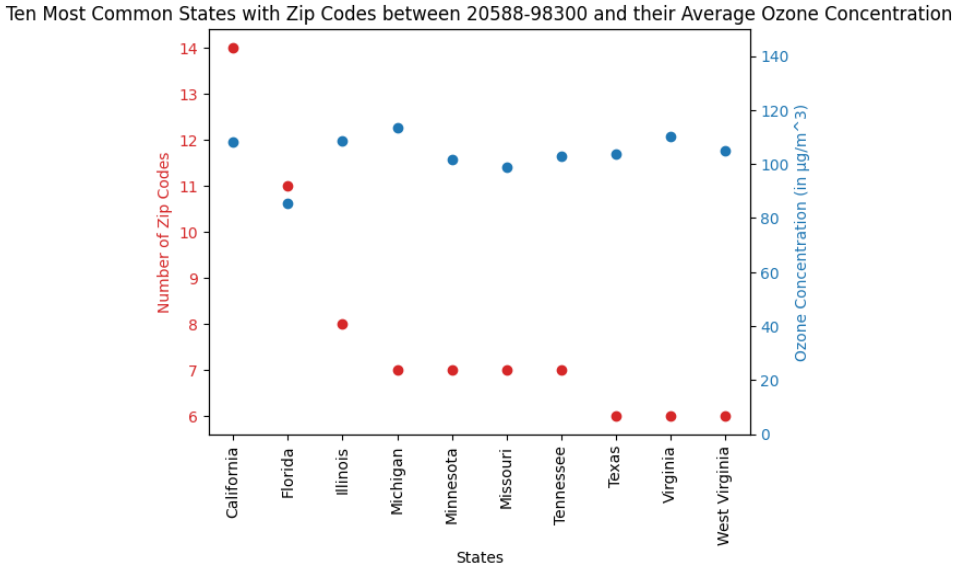
calcByState.csv

```
State,Avg Ozone Levels

CA,108.28

FL,85.66

IL,108.72

MI,113.55

MN,101.85

MO,98.85

TN,103.01

TX,103.82

VA,110.33

WV,104.87
```

Visualizations







Instructions

1) Run zip_code_1.py
2) Run zip_code_2.py 8 times
3) Run ozone_load.py
4) Run ozone_calc.py
5) Repeat steps 3 and 4 at least 3 more times
4) Run data.py

Documentation

Zip codes:

- **get_data(base_url)** - Takes in the base URL to get information about different zip codes and outputs a list of dictionaries where each dictionary contains information on each city.
- **output_json(cities)** - Takes in a list of dictionaries and outputs it to a JSON file.
- **make_states(cities)** - Takes in a list of dictionaries to get all of the states and creates a new dictionary that gives you a unique integer for each state.
- **make_states_db(cur, conn, states)** - Takes in the cur and conn for the database and a dictionary that gives you a unique integer for each state. It creates the states table in the database if not already there and inserts 25 states and associated ids at a time.
- **make_cities_db(cur, conn, cities, states)** - Takes in the cur and conn for the database and a dictionary that gives you a unique integer for each state and a list of dictionaries for each city. It creates the cities table in the database if not already there and inserts 25 cities with city name, zipcode and state id.
- **create_output(cur)** - Takes in the cur for the database and performs a join to connect the state name with the state id. It outputs a list of tuples for the state name and id.

Ozone:

- **getWeather(city, state)** - Takes a city and state as input, uses the OpenWeatherMap API to retrieve air pollution data for that location between two timestamps, and returns a dictionary of the maximum ozone level for each day within that time range
- **getCoord(city, state)** - Takes a city and state as input, uses the OpenWeatherMap API to retrieve the latitude and longitude of that location, and returns a tuple of these values
- **reorg_dict(results)** - Takes a dictionary of daily maximum ozone levels for multiple states, and reorganizes it into a new dictionary where each date is a key, and each value is a dictionary mapping each state to its maximum ozone level on that date.
- **make_ozone_id(ozone_dict)** - Takes a dictionary in the format returned by reorg_dict, and converts it to a dictionary of dictionaries, with each inner dictionary containing the ozone level, state, and date for a single record.
- **make_state_id(cities)** - Takes a list of tuples containing city and state information, and creates a dictionary mapping each unique state to a unique integer ID
- **make_states_db(cur, conn, states)** - Takes a database cursor, connection, and a dictionary of state IDs and names, and creates a SQLite table to store this data. It inserts up to 25 new state entries at a time.
- **make_air_db(cur, conn, air, states)** - Takes a database cursor, connection, a list of ozone data in the format returned by make_ozone_id, and a dictionary mapping state names to IDs. It creates a SQLite table to store this data, and inserts up to 25 new entries at a time.

- **calculation1(cur, conn)** - Calculates the average ozone level by state, joins this data with the state names from the StatesForAir table, and writes it to a CSV file. It returns the collected data as a list of tuples.
- **calculation2(cur, conn)** - Calculates the average ozone level by date and writes it to a CSV file. It returns the collected data as a list of tuples.
- **visualization(state_list, date_list)** - Creates a double visualization using two lists of tuples passed as arguments. First graph shows average ozone levels in 10 state capitals since April 1st. Second graph shows average daily ozone levels in those 10 state capitals since April 1st.

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
4/11	We discovered that the Songkick API could not be used	https://apipheny.io/free-api/	This solved the issue by giving us a new API and idea for the project.
4/11	Radhika was confused as to how to output a dictionary as a json file	https://www.geeksforgeeks.org/json-dumps-in-python/	This solved the issue by giving her the functions she needed to use to output a dictionary as a json file.
4/11	Radhika didn't know how to input only 25 table entries at a time	https://stackoverflow.com/questions/21829266/how-to-get-the-numbers-of-data-rows-from-sqlite-table-in-python	This solved the issue because it allowed her to figure out how many entries were already in the table.
4/15	Aidana struggled with converting regular date and time into Unix format, which made it difficult to send requests to OpenWeatherMap API with date filters	https://www.unixtimestamp.com/	By using the Unix converter, Aidana was able to successfully convert date and time values into a format that API could understand.
4/15	Aidana realized that the OpenWeatherMap API returned air quality records with dates in the Unix format. Keeping this format in output files would have made them not very user-friendly, so Aidana needed to find a way to convert those Unix values to normal dates	https://note.nkmk.me/en/python-unix-time-datetime/	Using this webpage, Aidana managed to find a way to convert Unix values into regular dates using the datetime library.