Group: Yoga tries to code

Members: Adam Vaas Jianxuan Xu Frederick Kusumo
Github Repository: https://github.com/frederickkusumo/si206-finalproj

Overall Project Goal

Our goal for this project is to gather information about 25 U.S. cities and compare each city to the other cities based on the information we collected. We wanted to compare the population, average air quality, and average home price for the cities we gathered data about. We also wanted to graph the data we collected to evaluate if there were any relationships between the population of a city, the average price of a house in that city, and its average air quality.

Achieved Goals

We achieved all the goals that we laid out in our overall project goals. In our graphs/visualizations, we displayed our data in ways that allow us to compare the information we collected about each city. We have graphs that compare air quality to the average home price, the average home price to the city population, and the average air quality to the city population. We also have a graph that shows the population of each city, and a graph that compares the air quality of Los Angeles to the air quality of New York over a six-day span. With these graphs, we were able to evaluate the data we collected to determine if any relationship exists between the average air quality of a city, the average home price, and the city's population. We were also able to compare these characteristics of each city to the other cities we collected data about. In conclusion, we can conclude that there is no direct relation between air quality, home prices, and population which can be seen from our data/visualizations. This is because there are more variables that we did not take into account such as possibly climate or terrain that normally causes air quality to worsen, and city sizes where limited ones would automatically drive home prices up. But generally thinking, if the population is high, it means that more people would demand buying new houses causing it to drive home prices higher. And if there are more people living in the area, that means that there is more variable that can cause air quality to deteriorate such as a higher number of vehicles.

Problems encountered

Census Data API Not Working:

When working with the US Census data API we were unable to get the API link to work properly. We played around with the examples on the US Census API's website but were unable to determine what was causing the issue. For some reason, we could not get the US Census data API to return the data it was supposed to.

Code not running on Adam's computer:

After everyone pushed their code, and everyone pulled the updated version of our code, Adam had issues running the code. The combined code worked fine on Frederick and Owen's computers but it didn't work on Adam's. We discovered that because his computer was trying to read our CSV files, it was unable to because it wasn't encoding using "UTF-8".

Code not running on Adam's computer (again):

Immediately after we fixed the issue with Adam's computer not being able to read the CSV values, another error occurred. After some observing the CSV files, we found that for some reason Adam's computer added a blank of text in the CSV file after each line of data. The CSV file looked as if there was a newline character written into the file after each line of code. This was an issue because we read each line of the CSV file in our code and turned it into a string to get the data for our graphs. The empty lines in the CSV file returned empty lists for the empty lines. This was an issue because later in our code we index into the lists that were created from reading the CSV file, and you can't index into an empty list. Basically, the empty lists caused an indexing error. After some research online we were still stuck, so Adam went to office hours.

Problems with Plotly:

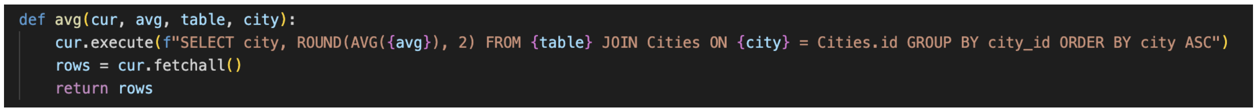
As we were trying to relate each other's data through visualization, we wanted to create graphs that contains two y-axes. We found resources online that coded the graphs in different ways and at times it didn't work. And how Creating a graph that contains a bar and scatters plot is different from a graph that contains two bars next to each other for the same x-axis item. Eventually, we found out that we had to import plotly.graph_objects and make_subplots.

Problems with Web Scaping Zillow site:

We are required to access the HTML codes of the websites by requesting URLs. Since it was needed to obtain 9 housing prices for 25 cities, Frederick would have to obtain them through the Zillow website. However, according to Zillow's Terms of Use, when scrolling through no. 5 states that conducting automated queries such as screen and database scraping is prohibited (Zillow Term of Use). Frederick went to clarify with an IA, GSI, as well as Dr. Ericson to ensure that it is okay for him to download HTML files of the website instead of calling it immediately from VS code. And it was made clear that that action is okay.

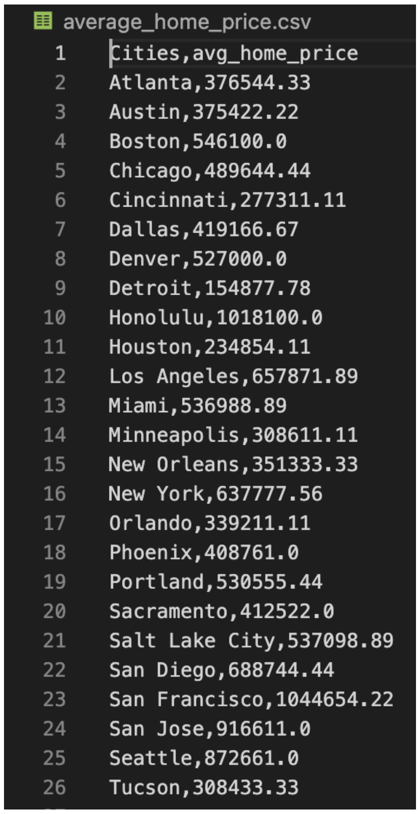
Calculations file

Calculations can be found in homeprice.py to calculate the average of home prices and air quality of 25 cities selected.



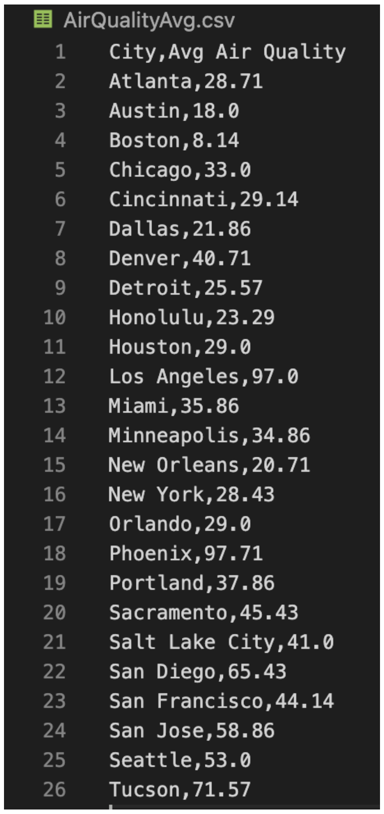
```python
def avg(cur, avg, table, city):
    cur.execute(f"SELECT city, ROUND(AVG({avg}), 2) FROM {table} JOIN Cities ON {city} = Cities.id GROUP BY city_id ORDER BY city ASC")
    rows = cur.fetchall()
    return rows
```

The outputs of the calculation were written in two separate CSV files. For average home prices, it can be found in average_home_price.csv, while the average air quality index can be found in AirQualityAvg.csv. The reason we created the other three CSV files (CityPopulation.csv, LAAirQualityDaily.csv, NYAirQualityDaily.csv) is to make the process of creating a visualization in a simpler way as they are already in list forms after reading them.



average_home_price.csv

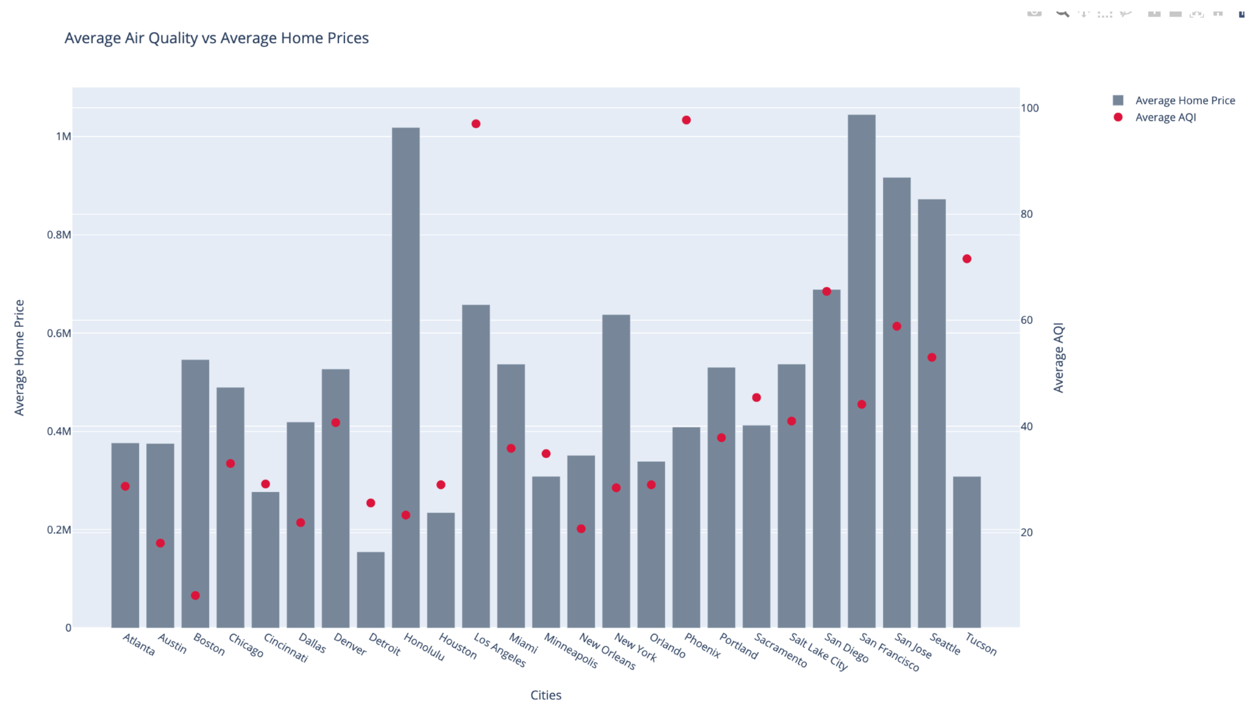
```
Cities,avg_home_price
Atlanta,376544.33
Austin,375422.22
Boston,546100.0
Chicago,489644.44
Cincinnati,277311.11
Dallas,419166.67
Denver,527000.0
Detroit,154877.78
Honolulu,1018100.0
Houston,234854.11
Los Angeles,657871.89
Miami,536988.89
Minneapolis,308611.11
New Orleans,351333.33
New York,637777.56
Orlando,339211.11
Phoenix,408761.0
Portland,530555.44
Sacramento,412522.0
Salt Lake City,537098.89
San Diego,688744.44
San Francisco,1044654.22
San Jose,916611.0
Seattle,872661.0
Tucson,308433.33
```

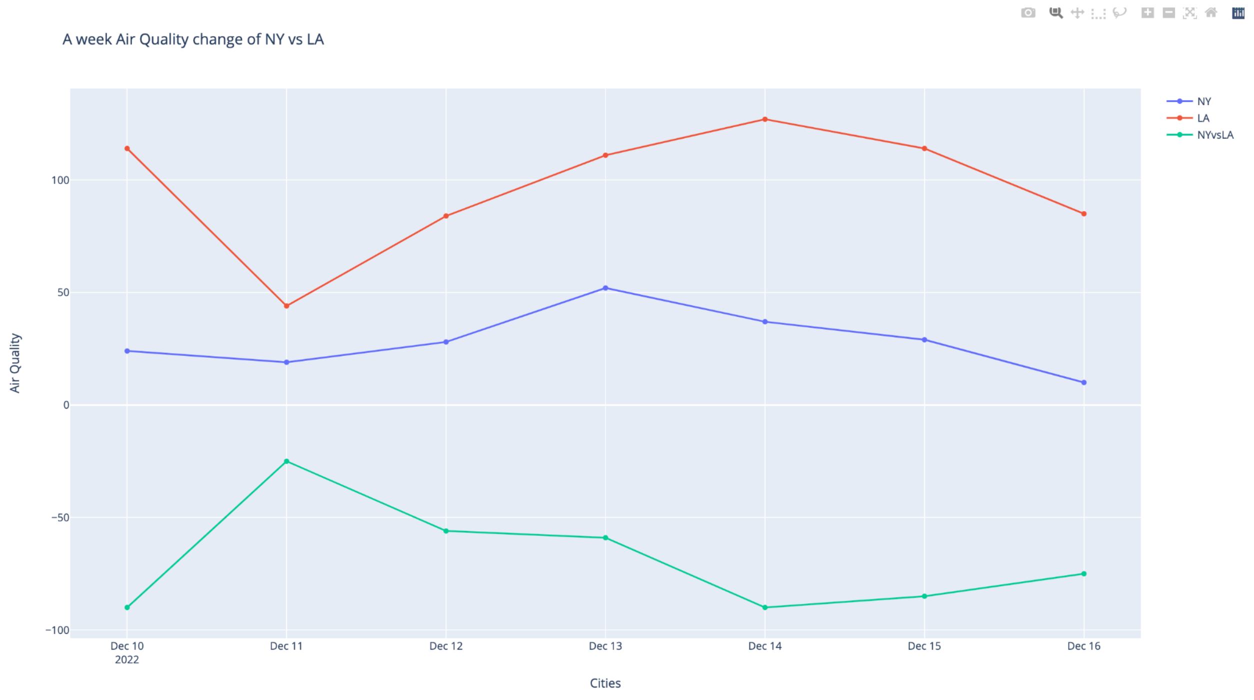


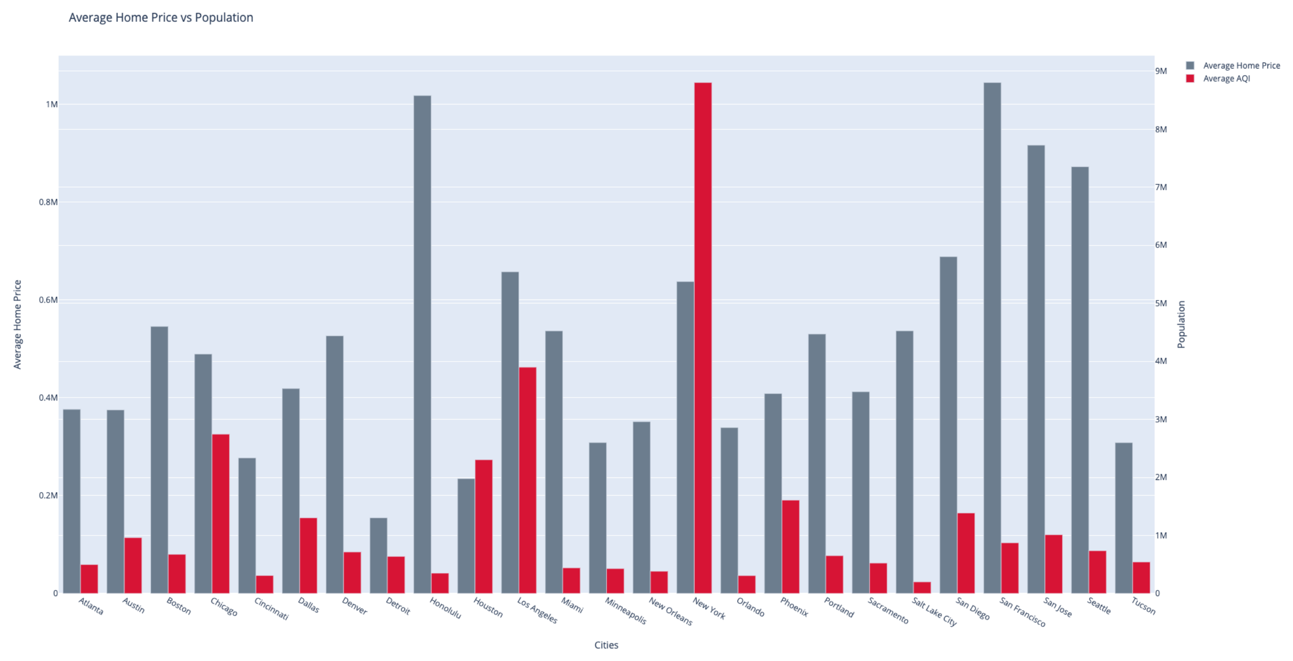
AirQualityAvg.csv

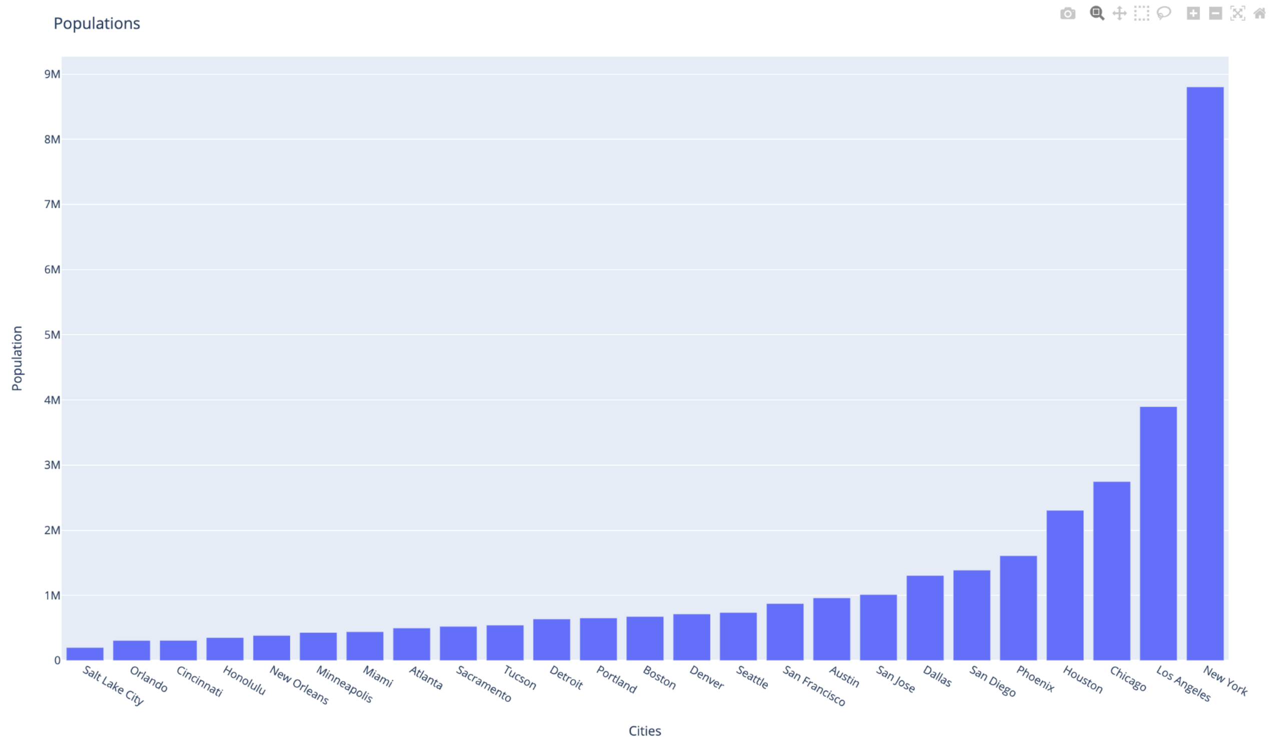
```
City,Avg Air Quality
Atlanta,28.71
Austin,18.0
Boston,8.14
Chicago,33.0
Cincinnati,29.14
Dallas,21.86
Denver,40.71
Detroit,25.57
Honolulu,23.29
Houston,29.0
Los Angeles,97.0
Miami,35.86
Minneapolis,34.86
New Orleans,20.71
New York,28.43
Orlando,29.0
Phoenix,97.71
Portland,37.86
Sacramento,45.43
Salt Lake City,41.0
San Diego,65.43
San Francisco,44.14
San Jose,58.86
Seattle,53.0
Tucson,71.57
```

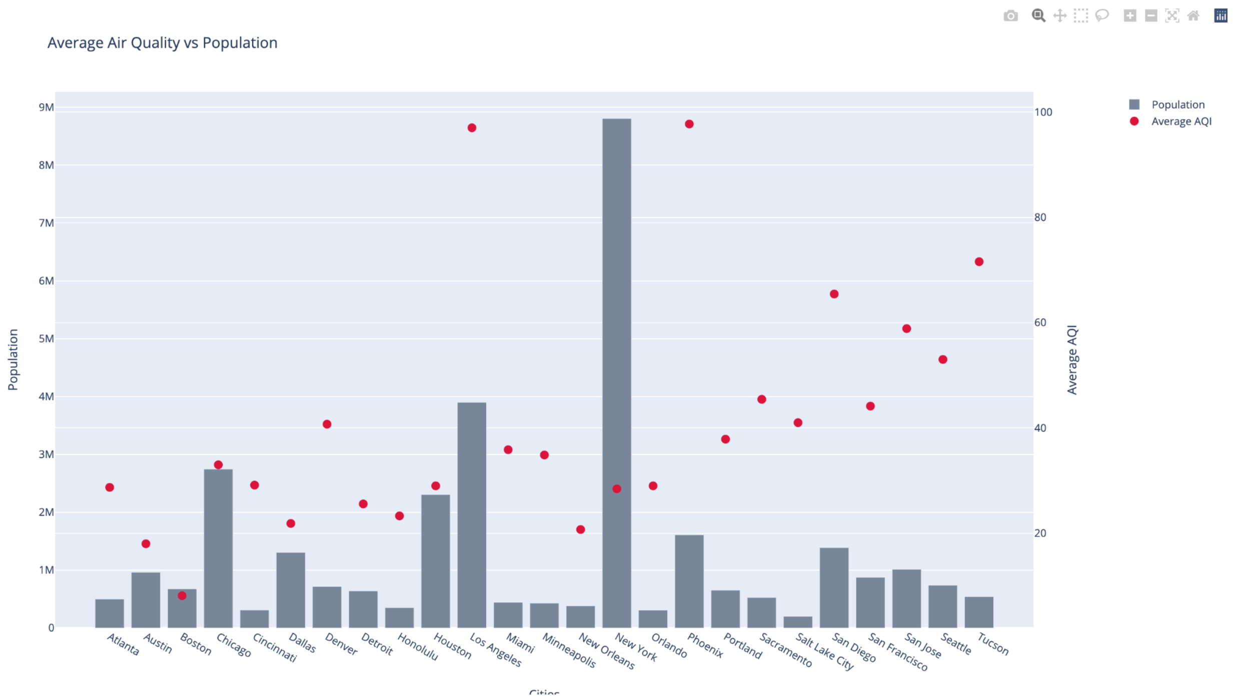
Visualizations






Average Home Price vs Population
Average Home Price
Average AQI
Average Home Price
Population
Cities
0
0.2M
0.4M
0.6M
0.8M
1M
0
1M
2M
3M
4M
5M
6M
7M
8M
9M
Atlanta
Austin
Boston
Chicago
Cincinnati
Dallas
Denver
Detroit
Honolulu
Houston
Los Angeles
Miami
Minneapolis
New Orleans
New York
Orlando
Phoenix
Portland
Sacramento
Salt Lake City
San Diego
San Francisco
San Jose
Seattle
Tucson


Populations
Population
Cities
0
1M
2M
3M
4M
5M
6M
7M
8M
9M
Salt Lake City
Orlando
Cincinnati
Honolulu
New Orleans
Minneapolis
Miami
Atlanta
Sacramento
Tucson
Detroit
Portland
Boston
Denver
Seattle
San Francisco
Austin
San Jose
Dallas
San Diego
Phoenix
Houston
Chicago
Los Angeles
New York


Average Air Quality vs Population
Population
Average AQI
Cities
9M
8M
7M
6M
5M
4M
3M
2M
1M
0
100
80
60
40
20
Atlanta
Austin
Boston
Chicago
Cincinnati
Dallas
Denver
Detroit
Honolulu
Houston
Los Angeles
Miami
Minneapolis
New Orleans
New York
Orlando
Phoenix
Portland
Sacramento
Salt Lake City
San Diego
San Francisco
San Jose
Seattle
Tucson

Code Instructions

Prior to running our code, delete all CSV files and the DB file. Then navigate to main.py to run the whole code. To run it, simply press the run button 9 times to let all of our data be stored in the database and 5 visualizations will be shown afterward. The code will take at least 12-14 minutes to complete running.

Function Documentation

Functions for main.py:

setUpDatabase:

Input(s): db_name

Output(s): cur, conn

The function *setUpDatabase* takes in the parameter db_name and then creates our database. It returns cur and conn.

Functions for api.py:

get_request_url:

Input(s): sting

Output(s): URL

This function will take a city name, using f-string input the city name to get each city's air quality URL.

get_data_using_cache:

Input(s): list

Output(s): dictionary

This function will take a list of city names, connecting to the API the get the JSON information. Then store the JSON information in a dictionary format.

create_cities_table:

Input(s): cur, conn

Output(s): database

This function creates the 25 cities table and city id as the primary key in the database.

add_AirQ_from_json:

Input(s): list, data, cur, conn

Output(s): database

This function read the dictionary generated from function *get_data_using_cache*, clears the dictionary data, and adds them to the database table “Air_quality” by 25 rows per time.

joinDataAVG:

Input(s): cur, conn

Output(s): database

This function joins the two tables, Air_quality and Cities where Air_quality.city_id = Cities.id.

write_csv:

Input(s): data, filename

Output(s): None

write_csv takes in the parameter data, which is the information returned from *write_csv.* It writes the information into a CSV file and doesn’t return anything.

write_csv3:

Input(s): data, filename

Output(s): None

write_csv takes in the parameter data, which is the information returned from *write_csv.* It writes the information into a CSV file and doesn't return anything.

read_csvTo2list:

Input(s): filename, list1, list2

Output(s): list1,list2

This function read the data from the CSV file and appends the data to the list1 and list2.

read_csvTo3list:

Input(s): filename, list1, list2, list3

Output(s): list1,list2, list3

This function read the data from the CSV file and appends the data to the list1, list2, and list3.

NYjoinData:

Input(s): cur, conn

Output(s): database

This function joins the New York City data from two tables, Air_quality and Cities where Air_quality.city_id = Cities.id.

LAjoinData:

Input(s): cur, conn

Output(s): database

This function joins the Los Angeles data from two tables, Air_quality and Cities where Air_quality.city_id = Cities.id.

Functions for homeprice.py:

get_price_info:

Input(s): info

Output(s): first_ten

This function takes an HTML file as input. It opens the file and parses through the code to find the span attribute containing the prices of the houses. It then strips '$' and ',' from each price and returns it in a list of integers.

get_detailed_info:

Input(s): cities

Output(s): data

This function takes a list of cities as input. It then calls the *get_price_info* function to get the 9 house prices based on the cities in the list. It would return a list of tuples containing the city ids as well as their respective housing prices.

add_prices_from_info:

Input(s): cur, conn, data

Output(s): None

This function takes the cursor, connection, and the data from the *get_detailed_info* function as input. It creates the table Home_Price if it does not exist in the database. And then it has a limit of inserting data into the database per 25 data items by selecting the max id per run.

avg:

Input(s): cur, avg, table, city

Output(s): rows

This function takes cursor, average, table, and city as input. It is written in f string so that it can be used through other files as well. This function joins the city_id table and the Home_Price table and then selects the city names and calculates the average of the housing prices based on each city rounded to two decimal places. It then returns the data selected and calculated.

tocsv:

Input(s): data, file

Output(s): None

This function takes data and a CSV file as input. It writes the data from *avg* and then writes it into the CSV file.

Functions for Population.py:

create_request_url:

Input(s): city_num, state_num

Output(s): URL

In the function *create_request_url* in Population.py the URL is created for the US Census data API key. The URL returns the population for each city in city cityData_list, which is located in *create_population_dict*. This function takes in two parameters, the city id number, and the state id number. Both of these inputs come from city_Data_list. The function simply returns the API URL for the inputted city.

create_population_dict:

Input(s): None

Output(s): population_tup_list

In the function *create_population_dict*, an API call is made for each city in the cityData_list. The API URL is made by calling *create_request_url.* Then, the URL is requested by *create_population_dict.* After the URL is requested the data from the URL is added into a tuple, along with an id specific to the population. Each tuple has the following format : (id, population). The new tuple is then added to a list. After this function creates a tuple for each city in cityData_list, the function returns the list of tuples containing the id and population for each city.

AV_create_database:

Input(s): population_tup_list, curr, conn

Output(s): None

In the function *AV_create_database* a new table is created in our database that has three columns; id, city_id, and population. The function only adds 25 items to the database each time it is run. It takes 25 items at a time from population_tup_list and adds them to the database. The function doesn't return anything.

AV_csv_data:

Input(s): cur

Output(s): rows

AV_csv_data selects the first 25 rows of data from the Population table in our database and then returns the selected rows in the form of a list of tuples. The tuples follow the format of (city_id, population).

AV_write_csv:

Input(s): data, filename

Output(s): None

AV_write_csv takes in the parameter data, which is the information returned from *AV_write_csv.* It writes the information into a CSV file and doesn't return anything.

Resources Utilized

Date	Issue description	Resource location	Result of research
11/18/2022 to 11/29/2022	When working with the US Census data API we were unable to get the API link to work properly. We played around with the examples on the US Census API's website but were unable to determine what was causing the issue. For some reason, we could not get the US Census data API to return the data it was supposed to.	We researched many parts of the US Census data API's site when trying to solve this issue. Below are the many different pages, but all on the same website. API Homepage List of root API links API information variable codes Example API Link List of city id's and State id's Census Data Youtube Walkthrough	After on-and-off research for about nine days we discovered that the reason our API links had not been working was that we were combining the wrong API root/base link with the wrong state and city id's. We thought the state and city ids were universal for all the root API links. However, unbeknownst to us, each API root link has its own unique set of city ids and state ids. We resolved the issue when we combined the correct API root link with the proper city and state IDs
12/02/2022	After everyone pushed their code, and everyone pulled the updated version of our code, Adam had issues running the code. The combined code worked fine on Frederick and Owen's computers but it didn't work on Adam's. We discovered that because his computer was trying to read our CSV files, it was unable to	Resource	After our research we learned that we could fix the issue that Adam was encountering by adding "encoding = UTF-8" to our CSV file handles. This would tell his computer how to read the files. Adding that information to the file handle fixed Adam's issue.

	because it wasn’t encoding using “UTF-8”.		
12/02/2022	Immediately after we fixed the issue with Adam’s computer not being able to read the CSV values, another error occurred. After some observing the CSV files, we found that for some reason Adam’s computer added a blank of text in the CSV file after each line of data. The CSV file looked as if there was a newline character written into the file after each line of code. This was an issue because we read each line of the CSV file in our code and turned it into a string to get the data for our graphs. The empty lines in the CSV file returned empty lists for the empty lines. This was an issue because later in our code we index into the lists that were created from reading the CSV file, and you can’t index into an empty list. Basically, the empty lists caused an indexing error. After some research online we were still stuck, so Adam went to office hours.	SI 206 office hours	After working with Holden, we still couldn’t figure out the issue as to why Adam’s computer printed a new line after each row of data in the CSV file. So, Dr. Ericson came to help. After a few minutes of examining the code and testing a few things, Dr. Ericson suggested adding a simple “if” statement where our code indexed into the lists that we got from reading the CSV file. She explained that since we just wanted to avoid the lists that were empty, we could check if the list was empty before we indexed it, and if the list was empty, we could ignore it and not index it. Her suggestion worked, and finally, our code ran on Adam’s computer.
11/20/2022	When Owen try to add the data from the dictionary to the database, he had a problem adding 25 data per time. All the data has been added to the database at one time.	SI 206 Office Hours	Owen went to Office Hour to ask for help with this problem. The IA told Owen he can use the for loop and index to set it only to add 25 rows to the database. By setting a count and data[count:count+25]. Then it will input index+25 per time.
11/21/2022	When Owen and Frederick	Resource	They find the tutorial of

	started to generate the data visualization, they had a problem creating the desired data graph with the correct coordination system.		plotly for python. According to the instruction they find, they modified the code to add a secondary y-axis for the second data they used in the data visualization
11/18/2022	To conduct web scraping, Frederick had to find which attribute and class to obtain the prices of the different houses specifically. He found out that the prices were contained inside the span tag but with the attribute data-test = property-card-price. He tried coding it the same way as it being a class or any attributes in general. This however resulted in him being unable to obtain the prices and he would get an error; "SyntaxError: keyword can't be an expression".	StackOverflow	According to StackOverflow it is advised to put the attribute and the names into a curly bracket and form it like a dictionary. So Frederick tried using; {'data-test':'property-card-price'} and it worked.
12/08/2022	Frederick was not allowed to Web Scrape Zillow website according to Zillow's Terms of Use. The Prohibited Use section in the Zillow Terms of Use document states that "conduct automated queries (including screen and database scraping, spiders, robots, crawlers, bypassing "captcha" or similar precautions, or any other automated activity with the purpose of obtaining information from the Services) on the Services;"	Zillow's Terms of Use SI 206 Lecture	Frederick confirmed with an IA, GSI, and Dr. Ericson that by downloading the HTML files for each web pages is totally fine for web scraping.