

Rebecca Yedid, Elliot Heldman, Thomas Joppich

SI 206

12/12/2023

<https://github.com/ryedid/206FinalProject>

Final Project Part 5 - Report

1. Our goals for this project began with our project proposal. Our group originally planned to work with four sources: a weather API containing data about the future weekly forecast, a weather API containing data about past weather, an API about city bikes and their use and quantity, and a website containing city data about the cities in which the bikes are held. The goals for these APIs and websites were to create data tables pertaining to the weather and data within cities in which there are city bikes, in order to find out why some cities may have more or less active bikes.
2. By the end of the project, we ended up using three of the four originally intended web sources. We achieved the goal of using the weather API containing data about future weekly forecasts, the city bike API about use and quantity, and the website containing city data in which bikes are held. We were able to successfully join the tables and relate the data for each city with the weather and bike quantity in order to attempt to analyze different aspects of the cities.
3. We faced some initial problems when accessing the data from both APIs due to unforeseen differences between them. One of these included having to access first an API within the API, which would then direct us to the data we were looking to gather into our tables. Another problem we faced was almost using Selenium to scrape our website with

Beautiful Soup. We almost wasted bountiful time learning and using Selenium to scrape, when we were then able to figure it out without having to do so. A final problem we faced was having to use integer keys to join our tables instead of the original string keys we had. We initially programmed all of our state listings in the table to format together to join over, but instead we had to update the tables to include integer keys to join over.

4.

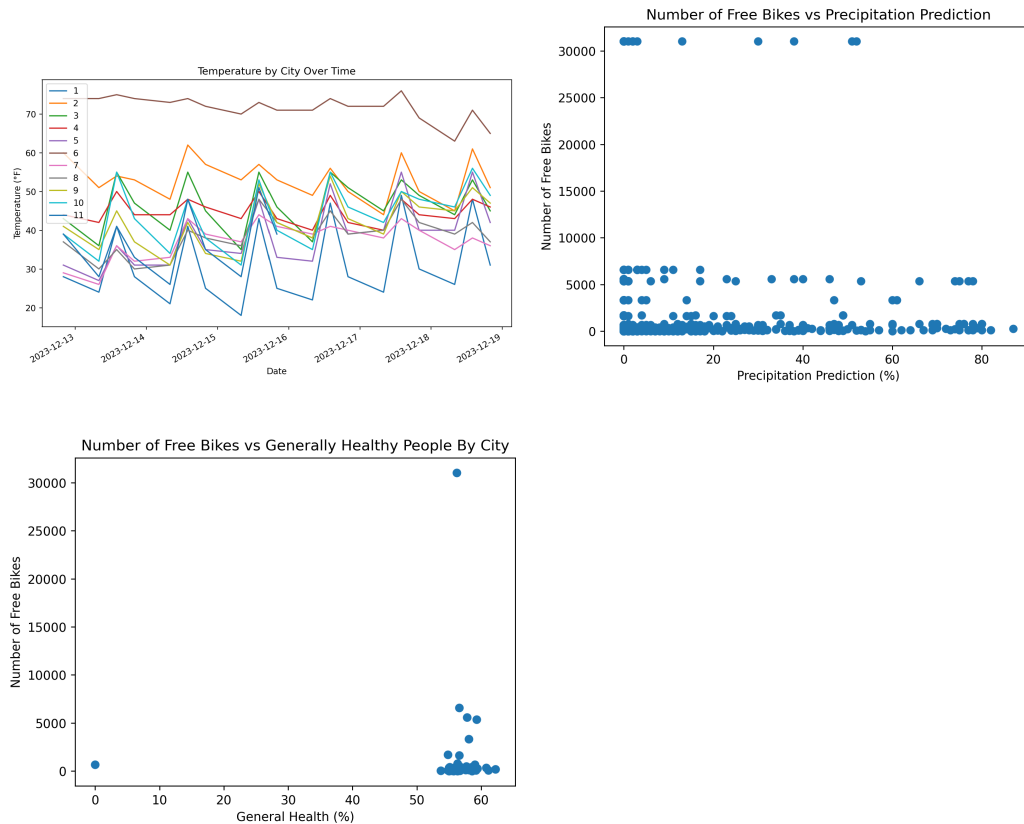
health.txt		
1	157	61.1 Aspen, CO
2	699	59.4 Austin, TX
3	335	55.0 Chattanooga, TN
4	1632	59.0 Portland, OR
5	456	62.2 Boulder, CO
6	113	57.6 Fort Lauderdale, FL
7	613	55.1 Milwaukee, WI
8	685	55.7 Buffalo, NY
9	6931	57.8 Washington, DC
10	197	56.5 Charlotte, NC
11	570	57.5 Cincinnati, OH
12	28132	56.2 New York, NY
13	935	58.5 Columbus, OH
14	823	58.1 Denver, CO
15	7847	56.6 Chicago, IL
16	80	56.0 El Paso, TX
17	453	56.0 Fort Worth, TX
18	651	57.7 Salt Lake City
19	415	55.7 Omaha, NE
20	3483	58.1 Boston, MA
21	3126	54.8 Philadelphia, PA
22	690	55.3 Indianapolis, IN
23	582	60.8 Madison, WI
24	1758	56.6 Los Angeles, CA

sunny.txt		
1	157	Aspen, CO
2	699	Austin, TX
3	335	Chattanooga, TN
4	456	Boulder, CO
5	613	Milwaukee, WI
6	685	Buffalo, NY
7	6931	Washington, DC
8	197	Charlotte, NC
9	570	Cincinnati, OH
10	28132	New York, NY
11	935	Columbus, OH
12	823	Denver, CO
13	7847	Chicago, IL
14	80	El Paso, TX
15	453	Fort Worth, TX
16	651	Salt Lake City
17	415	Omaha, NE
18	3483	Boston, MA
19	3126	Philadelphia, PA
20	690	Indianapolis, IN
21	582	Madison, WI
22	1758	Los Angeles, CA
23	4941	Houston, TX
24	146	Minneapolis, MN
25	412	San Antonio, TX
26	672	Long Beach, CA
27	134	Atlanta, GA
28	55	Des Moines, IA
29	6539	Greenville, SC
30	257	San Francisco Bay Area, CA
31	0	Las Vegas, NV
32	285	Miami, FL
33	346	Tucson, AZ
34	372	Park City, UT
35	1693	Richmond, VA
36	317	Honolulu

output.txt		
1	167.0	20.1 Aspen, CO
2	672.0	17.1 Atlanta, GA
3	696.0	18.9 Austin, TX
4	3569.0	18.8 Boston, MA
5	474.0	18.0 Boulder, CO
6	686.0	19.6 Buffalo, NY
7	198.0	19.6 Charlotte, NC
8	352.0	20.2 Chattanooga, TN
9	7783.0	18.6 Chicago, IL
10	580.0	18.4 Cincinnati, OH
11	917.0	19.3 Columbus, OH
12	823.0	18.6 Denver, CO
13	151.0	21.8 Des Moines, IA
14	80.0	20.0 El Paso, TX
15	108.0	19.3 Fort Lauderdale, FL
16	456.0	20.5 Fort Worth, TX
17	55.0	19.2 Greenville, SC
18	1685.0	0.0 Honolulu
19	589.0	19.3 Houston, TX
20	687.0	20.6 Indianapolis, IN
21	278.0	19.4 Kailua-Kona
22	258.0	21.1 Las Vegas, NV
23	415.0	20.1 Long Beach, CA
24	562.0	19.8 Madison, WI
25	1763.0	17.2 Miami, FL
26	578.0	20.0 Milwaukee, WI
27	4941.0	19.3 Minneapolis, MN
28	27323.0	19.4 New York, NY
29	420.0	21.0 Omaha, NE
30	346.0	23.9 Park City, UT
31	3164.0	19.4 Philadelphia, PA
32	1576.0	19.7 Portland, OR
33	392.0	19.7 Richmond, VA
34	785.0	20.0 Salt Lake City
35	163.0	19.9 San Antonio, TX
36	6167.0	16.7 San Francisco Bay Area, CA

bad.txt		
1	157.0	20.1
2	699.0	18.9
3	335.0	20.2
4	1632.0	19.7
5	456.0	18.0
6	113.0	19.3
7	613.0	20.0
8	685.0	19.6
9	6931.0	17.2
10	197.0	19.6
11	570.0	18.4
12	28132.0	19.4
13	935.0	19.3
14	823.0	18.6
15	7847.0	18.6
16	80.0	20.0
17	453.0	20.5
18	651.0	20.0
19	415.0	21.0
20	3483.0	18.8
21	3126.0	19.4
22	690.0	20.6
23	582.0	19.8
24	1758.0	20.0

5.



6. Instructions for Running Final Project Code:

- First, run the file called “city_bikes_part.py”. It will populate the CityBikes table from the CityBikes API. If you notice there are duplicate indices starting at around index 25 in the database, delete the database and run this file again.
- Second, run the file called “weather_part.py”. We created a commented out loop that will help with runtime as this function needs to be run 32 times (25 data points each). Feel free to comment it out to run one long loop instead. It will populate the WeatherCities table from the Weather API. If you see a few blank indices, this may result from time zone differences as the Weather API does get refreshed data every hour. This will not result in any future errors. You might also see some 500 or 503 errors, these will also not affect the code and are also caused by time differences.

- c. Thirdly, run the file called “city_data.py”. It will populate the CityData table from the CityData website.
- d. Next, run the file called “calculations.py”. It will create new tables called CityIndex, sqlite_sequence, and DescriptionIndex.
- e. Finally, run the file called “visualizations.py”. It will pop up our three visualizations. The second shows up after the first is closed out, and the third after the fourth is closed out.

7. Function Descriptions

- a. Bikes_part.py
 - i. get_city_bike_data(): this function navigates the created json file and gets the data from it for the base API data. It outputs the data in the base API to be then used later. It also actively error checks that the link it gets is not faulty.
 - ii. load_json(): this function takes in the return of the get_new_data() function and outputs the actual data to be parsed by the make_SQL() function. It uses json.dump to do so.
 - iii. set_up_database(): this function solely sets up the path to the database and returns the cur and conn values to be used in later work.
 - iv. get_list_of_ids(): this function navigates the originally created API access to find the network ids. It uses the network ids to then go into those APIs and gather the secondary ids which get stored in a list. It then outputs that list to be used in a later function to gather more data for our table.

- v. `get_new_data()`: this function takes in a url. It then checks the url validity before finding its data in the json file. After finding the data it returns the data to be used in a later function.
- vi. `make_SQL()`: this function takes in the cur and conn for database usage . It then creates our BikeCities table and navigates through the listing of links given. Utilizing the `get_new_data()` and `load_json()` files, it gets all of our data needed for BikeCities and inserts it with a limit of 25 items to be inserted each time the file is run. After 25 unique items, it commits the changes to the database and ends.
- vii. `main()`: we technically have our lines of code outside of a main function, but we use this space to run the above listed functions and create our database.

b. `Weather_part.py`

- i. `get_coordinates_from_database()`: this function takes in the path to the database as well as the cur and conn to sqlite. This functions gets us connected with the database, creates a cursor, and then retrieves all of the coordinates from the database we made in `bikes_part.py`. It then puts that data into a list and returns that list of coordinates.
- ii. `get_links()`: this function takes in the list made in `get_coordinates_from_database()`. It then creates a list for links and loops through all of the coordinates forming them into URLs to be placed in the list. It also does active error checking with a try and except statement to see if the link actually works. It then returns the list of links.

- iii. `get_new_data()`: this function takes in a url. It then checks the url validity before finding its data in the json file. After finding the data it returns the data to be used in a later function.
- iv. `load_json()`: this function takes in the return of the `get_new_data()` function and outputs the actual data to be parsed by the `make_SQL()` function. It uses `json.dump` to do so.
- v. `make_SQL()`: this function takes in the `cur` and `conn` for database usage as well as the list of links we created in `get_links()`. It then creates our `WeatherCities` table and navigates through the listing of links given. Utilizing the `get_new_data()` and `load_json()` files, it gets all of our data needed for `WeatherCities` and inserts it with a limit of 25 items to be inserted each time the file is run. After 25 unique items, it commits the changes to the database and ends.
- vi. `main()`: we technically have our lines of code outside of a main function, but we use this space to run the above listed functions and create our database.

c. `City_data.py`

- i. `scrape_city(city)`: This function takes in the name of a U.S. city and returns a tuple with three health statistics: (city id, general health percentage, overweight population percentage, percentage of people who ‘feel badly about themselves’). The function appends the city name to the base url and accesses the city’s unique profile page on `city-data.com`.

Then, a beautiful soup object is created to parse the city's profile page for the health statistics.

- ii. `make_tup_list(city_list)`: This function takes in a list of US cities and returns a list of tuples generated by the `scrape_city()` function. The function iterates through the list of cities using a for loop, calling `scrape_city()` each time. Each tuple is appended to the return list (`return_lst`).
 - iii. `make_SQL(data_list, cur, conn)`: This function takes in a list of data as well as cursor and connection items from `sqlite3`. A table is created called "Cityhealth" with four columns corresponding to the tuple values generated from `scrape_city()`. The list of tuples is iterated through using a for loop that inserts a row to the Cityhealth table for each city. The function limits the number of insertions to 25 on each call. After 25 rows are inserted, or when the end of the `data_list` is reached, the function commits and ends.
 - iv. `main()`: we technically have our lines of code outside of a main function, but we use this space to run the above listed functions and create our database.
- d. Calculations.py
- i. `create_city_index(cur, conn)`: This function creates a table named CityIndex in the database, populates it with unique city names, and cleans up any empty entries.

- ii. `create_description_index(cur, conn)`: This function creates a table named `DescriptionIndex` in the database, populates it with unique weather descriptions, and cleans up any empty entries.
- iii. `update_weather(cur, conn)`: This function adds a new column called `short_id` to the `WeatherCities` table. It performs an update operation using information from the `DescriptionIndex` table to populate this new column.
- iv. Calculations Functions:
 - 1. `calculate_bikes_bad(cur, conn)`: Calculates the average number of empty bike slots and the average percentage of people feeling bad about themselves for each city and returns the result.
 - 2. `calculate_sunny(cur, conn)`: Calculates the number of empty bike slots for cities with the weather described as "Sunny" and returns the result.
 - 3. `calculate_bikes_genhealth(cur, conn)`: Calculates the number of empty bike slots and the general health percentage for each city and returns the result.
- v. `write_results_to_file(results, output_file_path)` and `write_results_to_file_health(results, output_file_path)`: These functions write the calculation results to text files.
- vi. The script then connects to the database, creates indexes, updates weather data, performs calculations related to bike slots, weather conditions, and health percentages for cities, and writes these results to separate output files (`sunny.txt`, `health.txt`, `bad.txt`).

e. Visualization.py

- i. `fetch_weather_data_from_db(database_path)`: This function connects to the specified SQLite database, retrieves weather data (city, date, hour, temperature) from the WeatherCities table, limits the data to 200 entries, stores it in a Pandas DataFrame, and returns it.
- ii. `plot_temperature_by_city(df)`: This function takes the weather data DataFrame, converts the date and hour columns to a datetime format, groups the data by city, and plots the temperature over time for each city.
- iii. `fetch_weather_bike_data(database_path)`: This function retrieves data from both the WeatherCities and BikeCities tables by performing a join on the 'city' column and returns a DataFrame containing city, precipitation, and the number of free bikes.
- iv. `plot_bikes_precip_scatter(df)`: This function takes the weather-bike data DataFrame and creates a scatter plot showing the relationship between precipitation prediction and the number of free bikes across cities.
- v. `fetch_health_bike_data(database_path)`: This function retrieves data from the CityHealth and BikeCities tables by performing a join on the 'city' column and returns a DataFrame containing city, general health percentage, and the number of free bikes.
- vi. `plot_bikes_health_scatter(df)`: This function takes the health-bike data DataFrame and creates a scatter plot showing the relationship between the general health percentage and the number of free bikes across cities.

- vii. `main()`: fetches weather data, creates temperature plots by city, fetches weather-bike and health-bike data, and creates scatter plots to visualize relationships between weather factors, health percentages, and the availability of free bikes in different cities.

8. Documentation of All Resources Used

Date	Issue Description	Location of Resource	Result
12/05	Having trouble error checking as to why our sites were getting faulty links and giving errors.	ChatGPT	Found that we could use an error checking try and except function which we then implemented.
12/11	Help with errors in normalization in our integer key between our tables.	Office Hours	Unable to receive help because office hours ended up being closed.
12/05	Needed help creating SQL	Class Notes/Slides	Found lab slides and homework

	database, table, and inserting items		projects with the necessary information.
12/03-12/12	Needed information from the API in order to create data tables every time we ran our code daily.	Weather API	Gathered data on the 40 US Cities hourly weather which updated every hour as well, to be used in calculations and visualizations.
12/03-12/12	Needed health data for US cities	City Data Website	Gathered health data on 40 US cities that can be used in calculations and visualizations.
12/03-12/12	Needed information about city bike availability in US cities	City Bike API	Gathered data on 40 US cities pertaining to bike rental availability