Maya Gordon, Allie Kaplan, Samy Bentata
SI 206
April 14, 2025

Final Project Report

***Note:** Our group wrote the majority of our code on one computer in one file when we were all sitting together in order to avoid GitHub errors. As a result, not all group members have a lot of git commits.

A. The goals for your project including what APIs/websites you planned to work with and what data you planned to gather (10 points)

Initially, our group planned to use the Yelp API, the OpenTable website, and the City Health Inspection Database. With this, we wanted to see the correlation between restaurant ratings and health inspection scores and how these ratings compare between Yelp and OpenTable. Ultimately, we could not pursue this project because the APIs required payment to access. As a result, we decided to use the Open Weather Map API and the City Bikes API, and web-scraped from a Wikipedia page listing U.S. cities by population. From this data, we aimed to see if there was a correlation between city population, average temperature, and the number of city bikes.

B. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather (10 points)

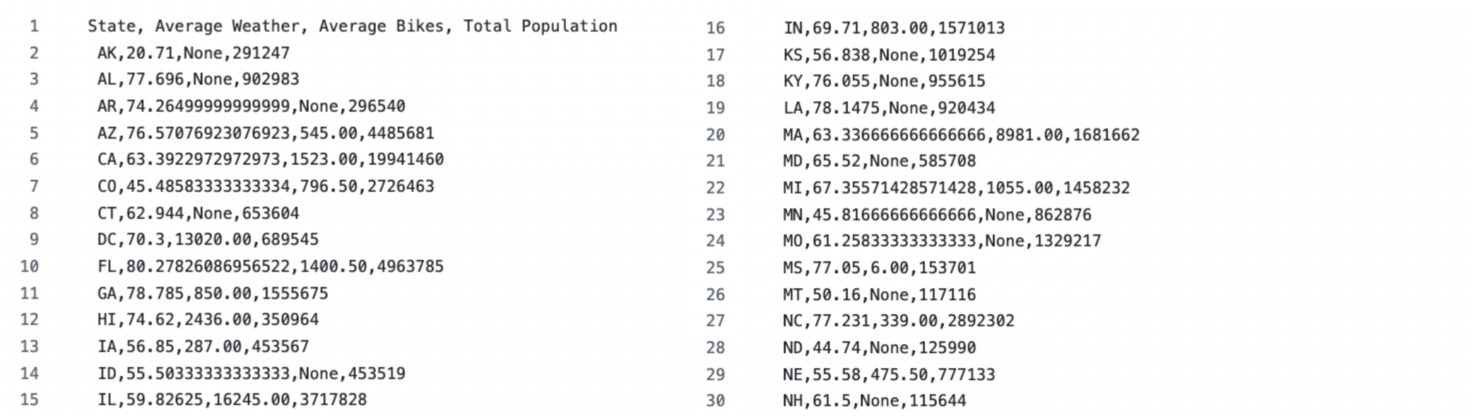
Using the Open Weather Map and City Bike API, as well as Wikipedia's city population data, we used city temperatures, the number of city bikes, and city population as our data. From this, we calculated the average number of bikes per state, the average weather per state, and the total population per state. From this, we were able to visualize the relationship between average temperature, number of city bikes, and population per US city.

C. The problems that you faced (10 points)

Throughout this project, we faced many challenges. One major challenge we faced early on was ensuring that the names of cities were consistent throughout APIs. For example, the weather API contained "New York City", whereas the city bike API contained "New York, NY". Finding this error was challenging to catch, but once we did, we were able to correct it somewhat easily by hardcoding that specific city. Another challenge we faced was that many of our APIs contained limits on how many times they could run. This meant that we had to put the API data into JSONs so that our code could still run when this occurred. Lastly, another challenge we faced was figuring out how to best represent our data in graphs. Initially, all of our graphs were histograms; however, after analyzing them, we realized this was not the most readable way to visualize our data.

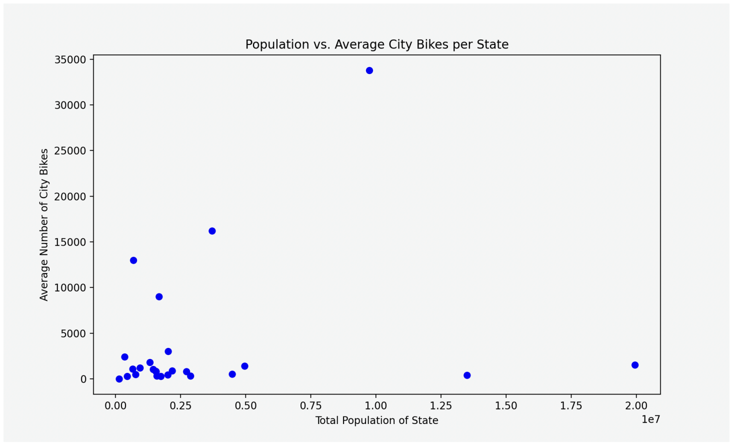
D. The calculations from the data in the database (i.e. a screenshot) (10 points)

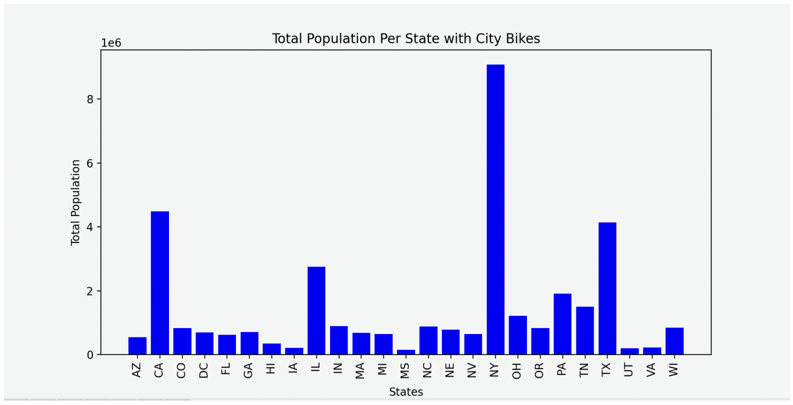
**First 30 rows

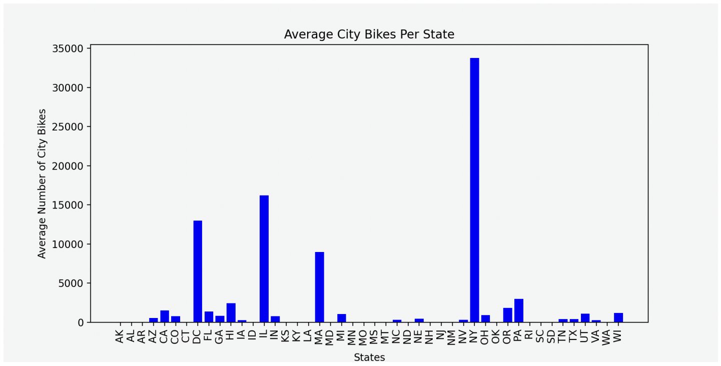


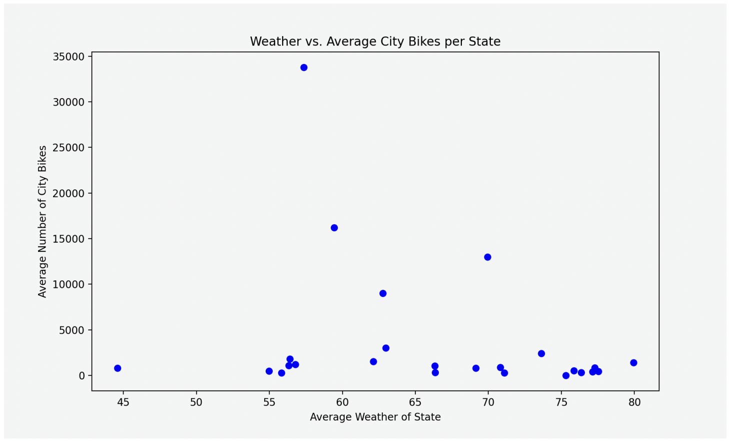
```
State, Average Weather, Average Bikes, Total Population
AK,20.71,None,291247
AL,77.696,None,902983
AR,74.26499999999999,None,296540
AZ,76.57076923076923,545.00,4485681
CA,63.3922972972973,1523.00,19941460
CO,45.48583333333334,796.50,2726463
CT,62.944,None,653604
DC,70.3,13020.00,689545
FL,80.27826086956522,1400.50,4963785
GA,78.785,850.00,1555675
HI,74.62,2436.00,350964
IA,56.85,287.00,453567
ID,55.50333333333333,None,453519
IL,59.82625,16245.00,3717828
IN,69.71,803.00,1571013
KS,56.838,None,1019254
KY,76.055,None,955615
LA,78.1475,None,920434
MA,63.336666666666666,8981.00,1681662
MD,65.52,None,585708
MI,67.35571428571428,1055.00,1458232
MN,45.81666666666666,None,862876
MO,61.25833333333333,None,1329217
MS,77.05,6.00,153701
MT,50.16,None,117116
NC,77.231,339.00,2892302
ND,44.74,None,125990
NE,55.58,475.50,777133
NH,61.5,None,115644
```

E. The visualization that you created (i.e. screenshot or image file) (10 points)









F. Instructions for running your code (10 points)

Users should run the file titled “final.py”. This will run the code, creating the four data visualizations as well as creating a text file titled “calculations.txt”. In case the APIs reach too many requests, three JSON files are created for the code to run to avoid error (city_info.json, weather_data.json, city_bike_data.json). Users should run the code 14 times to get the full data set.

G. Documentation for each function that you wrote. This includes describing the input and output for each function (20 points)

1. get_populations()
    - Input: None
    - Scrapes the Wikipedia page to collect the population, latitude, and longitude for US cities
    - Output: dict containing cities and a nested dict for population, latitude, and longitude
2. create_database(db_name)
    - Input: the name of the SQLite database
    - Output: a tuple of the SQLite cursor (cur) and connection (conn)
3. create_states_table(data, cur, conn)
    - Input
        - Data: dictionary of city data, created in get_populations()
        - Cur and con: cursor and connection objects
    - Output: None
    - Takes the states from the city data and creates a table in the database called “States”, where each state has a unique ID
4. create_citybike_table(data, cur, conn)
    - Input
        - Data: dictionary of city data, created in get_populations()
        - cur and con: cursor and connection objects
    - Output: None
    - Creates a table in the database called City_Bike that contains the city name, state ID, latitude and longitude, and population
5. city_bikes()
    - Input: None
    - Output: a dict where the keys are US cities and the values are the number of city bikes in that city
    - Saves the data to a JSON, so if the API reaches its limit, it can still access this data

6. add_city_bikes(bike_data,cur,conn)
    - Input
        - bike_data: dictionary from city_bikes()
        - cur and con: cursor and connection objects
    - Output: None
    - Updates the City_Bike table with the bike availability in each city
7. avg_bikes_by_state(cur, conn)
    - Input
        - cur and conn: cursor and connection objects
    - Output: a list of tuples where each contains a state and its average number of city bikes per station
8. avg_bike_by_state_graph(cur, conn)
    - Input
        - cur and conn: cursor and connection objects
    - Output: a bar chart representing the average number of city bikes per state
9. pop_per_state(cur, conn)
    - Input
        - cur and conn: cursor and connection objects
    - Output: a list of tuples where each tuple is a state and its population (only for cities with city bikes)
    - Calculates total population per state for cities with bike data
10. pop_per_state_graph(cur, conn)
    - Input
        - cur and conn: cursor and connection objects
    - Output: a bar chart representing the total population per state for cities with bike data
11. pop_and_bikes_per_state(cur, conn)
    - Input
        - cur and conn: cursor and connection objects
    - Output: a list of tuples where each tuple contains a state, total population, and average number of city bikes
    - Combines the total population and average number of city bikes by joining the “States” and “City_Bike” tables using JOIN
12. pop_bikes_scatter_plot(cur, conn)
    - Input
        - cur and conn: cursor and connection objects
    - Output: a scatter plot comparing the relationship between city population and number of bikes using data from pop_and_bikes_per_state()
13. get_temperature(lon, lat)
    - Inputs: longitude and latitude as strings
    - Output: a float of the current temperature in a given location based on the long and lat (in Fahrenheit)
14. insert_weather(data, cur, conn)
    - Input

        - data: the dictionary from get_population()
        - cur and conn: cursor and connection objects
    - Output: None
    - Uses get_temperature() to retrieve the weather in each city and save this data as a JSON
15. avg_weather_bikes_by_state(cur, conn)
    - Input
        - cur and conn: cursor and connection objects
    - Output:
16. avg_weather_bikes_by_state_plotcur, conn):
    - Input
        - cur and conn: cursor and connection objects
    - Output: A list of tuples containing a state, its average temperature, and average number of bikes
    - This function calculates the average temperature and number of city bikes per state
17. calculations()
    - Input
        - cur and conn: cursor and connection objects
    - Outputs: the text file, calculations.txt containing the calculated, the total population, average weather, and average bikes per state
18. main()
    - Executes the full program

H. You must also clearly document all resources you used. The documentation should be of the following form (20 points)

Date	Issue Description	Location of Resource	Result (Did it Solve the Issue?)
4/9	The Open Weather Map API	Found on the github link in the project description	Yes
4/9	The City Bike API	Found on the github link in the project description	Yes
4/9	Wikipedia's City Population Data (webscraped)	Found online	Yes
4/9	Our group went to office hours with Aadarsh and Daniel. Aadarsh helped us get started with our project by visualizing what we	North Quad	Yes

	needed to do and what data we wanted to use in our graphs. Daniel also helped us with issues on GitHub.		
4/10	Our group went to office hours again with Aadarsh, this time to help us transfer our data to the table that we made.	North Quad	Yes
4/10	We used Chat GPT to help us debug what was causing our code to fail when we encountered too many API requests. After learning what was causing this, we used HW 6 to help us create a JSON for when this occurred.	At our house.	Yes
4/13	Our group consulted ChatGPT as well as the lecture slides from lectures 20 and 21 to help us create our graphs. This helped us with formatting our code.	At our house	Yes
4/14	Our group used Chat GPT to help us ensure that each time we ran the code, 25 new rows were added to the database table. After we completed our code, we realized that this was the one requirement that we did not meet, so we had to consult Chat GPT to help us fix this.	At our house.	Yes

Link to repository: https://github.com/alliekaplan/Final-Project