SI 206 Final Project

Part 1

A. What is your group's name?
Weather Or Not.

B. Who are the people in the group (first name, last name, umich email)?
Ty Anderson (tyand@umich.edu), Jolie Oleshansky(jolieo@umich.edu), Teresa Willison(tesswill@umich.edu)
What APIs/websites will you be gathering data from? ***The base URLs for the APIs/websites must be different for them to count as different APIs***.
Weather data →
http://dataservice.accuweather.com/locations/v1/cities/neighbors/{locationKey}
Census data → https://www.census.gov/data/developers/data-sets/acs-1year.html
https://api.census.gov/data/2021/acs/acs5
Tomtom → https://developer.tomtom.com/
https://api.tomtom.com/traffic/services/4/flowSegmentData/relative0/10/json

C. What data will you collect from each API/website and store in a database? Be specific.
From the weather API, we will gather information regarding specific sites that we get from Tomtom to analyze traffic patterns. We will use the census data following that to see how population density in different neighborhoods affects the overall traffic flow and the OpenStreetMap will be used to map vehicle movement and congestion.

D. What data will you be calculating from the data in the database? Be specific.
The data we will be calculating from the data in the database will be surrounding the idea of average number of vehicles in a space at a given time, as well as a potential accident rate given different populations and weather. Finally, maybe something relating to season changes in these patterns as well.

E. What visualization package will you be using (Matplotlib, Plotly, Seaborn, etc)?
We will be using Matplotlib, but we are open to using other platforms as we continue to learn about them in class. We will continue to be flexible, but like how Matplotlib allows us to make customizable and interactive visualizations.

F. What graphs/charts will you be creating?
We will be creating visualizations that show the distribution of our data, especially highlighting the significance of our calculated averages over the differing seasons and other potential discrepancies we feel worth making visual. We will do this using bar graphs, both horizontal and vertical, and using a pie chart. The bar graph will depict the relationship between the weather patterns and the traffic back ups. The pie chart will show how often the OpenStreetMap is used.

G. Who is responsible for what? Please note that all team members should do an equal amount of programming and total work.

Jolie - Weather API and DB, Part 5 reflection, Part 4- 1 visualization
Ty - Traffic API and DB, Part 3 database joins
Tess- Census API and DB, Part 4 Visualizations

Part 2

- [x] ~~Access 100 rows from personal api~~
    - [x] ~~Weather API- Jolie~~
    - [x] ~~Tomtom- Ty~~
    - [x] ~~USCensus- Tess~~
- [x] ~~Two tables sharing an integer key~~
    - [x] ~~City~~
        - [x] ~~City, State~~
- [x] ~~You must limit how much data you store from an API into the database each time you execute the file that stores data to the database to **25 or fewer items** (60 points). The data must be stored in a SQLite database. This means that you must run the file that stores the data multiple times to gather at least 100 items total without duplicating any data or changing the source code~~

Top 20 City data:

Bozeman, Montana
Pullman, Washington
Gainesville, Florida
Boone, North Carolina
Clemson, South Carolina
East Lansing, Michigan
Moscow, Idaho
Provo, Utah
Ann Arbor, Michigan
Stanford, California
Bloomington, Indiana
Athens, Ohio
Ellensburg, Washington
San Luis Obispo, California
College Station, Texas
Laramie, Wyoming
Amherst, Massachusetts
Manhattan, Kansas
West Lafayette, Indiana
Platteville, Wisconsin

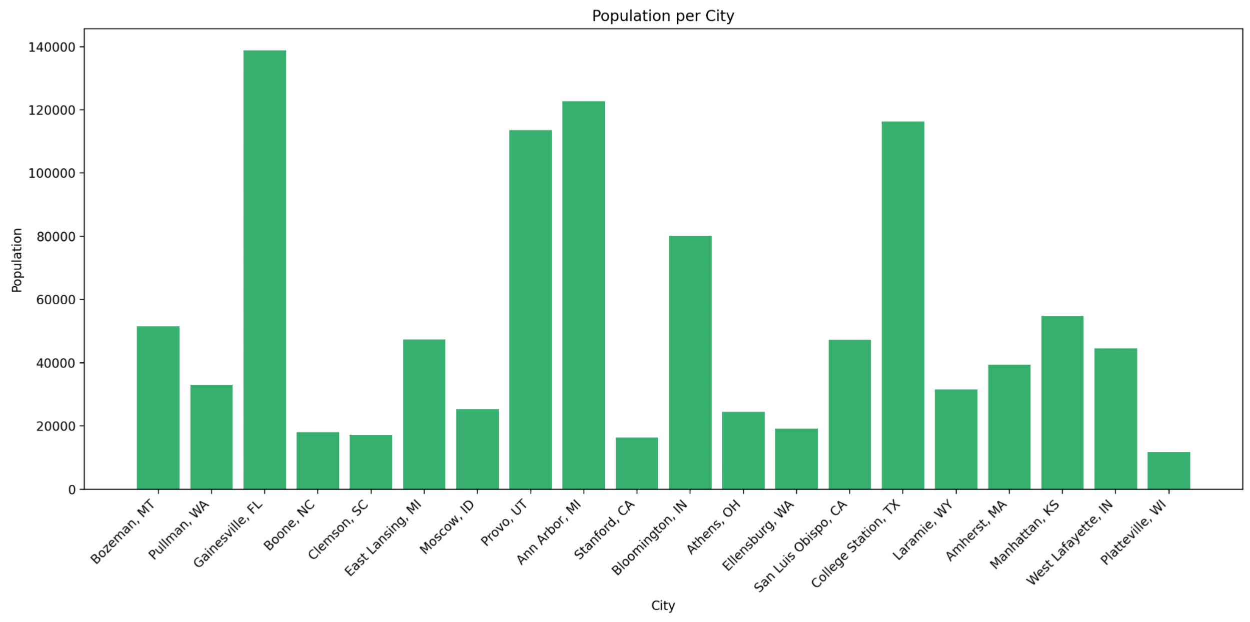
Part 3

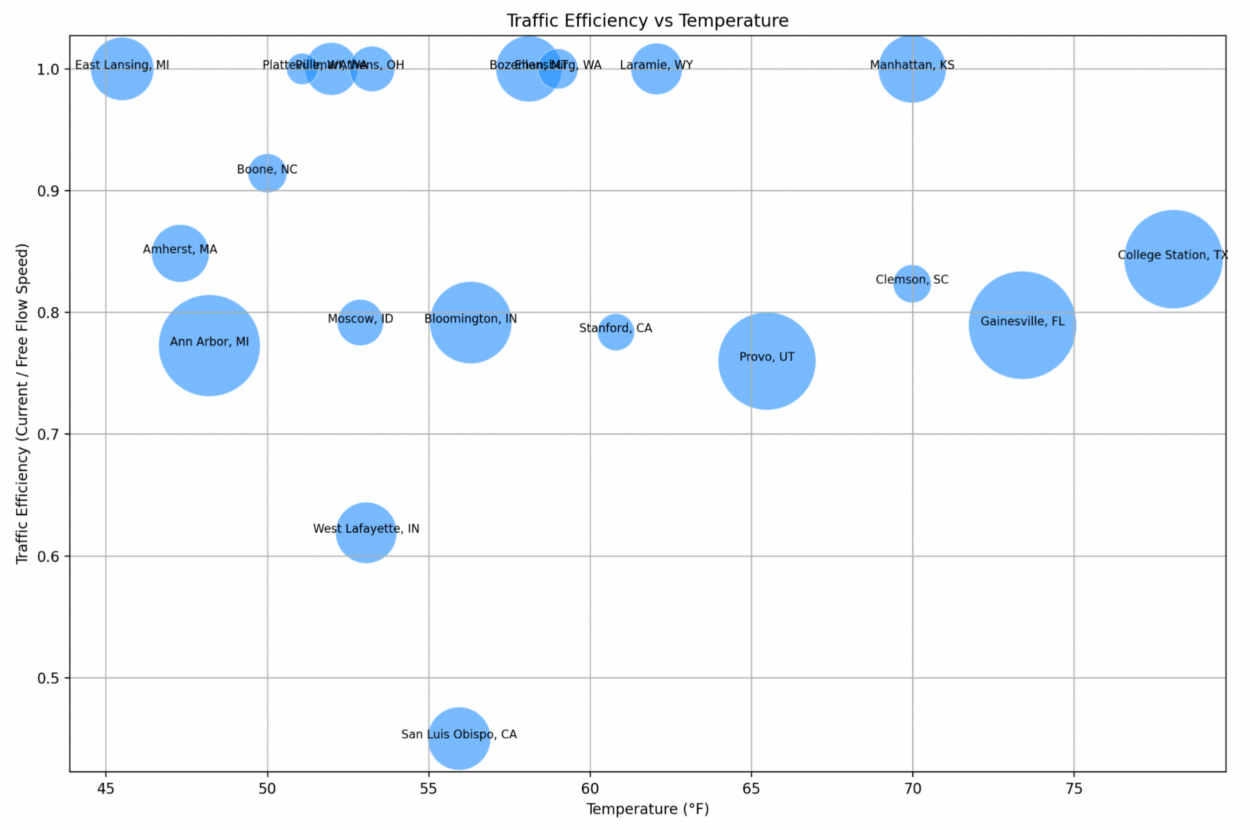
Process the data

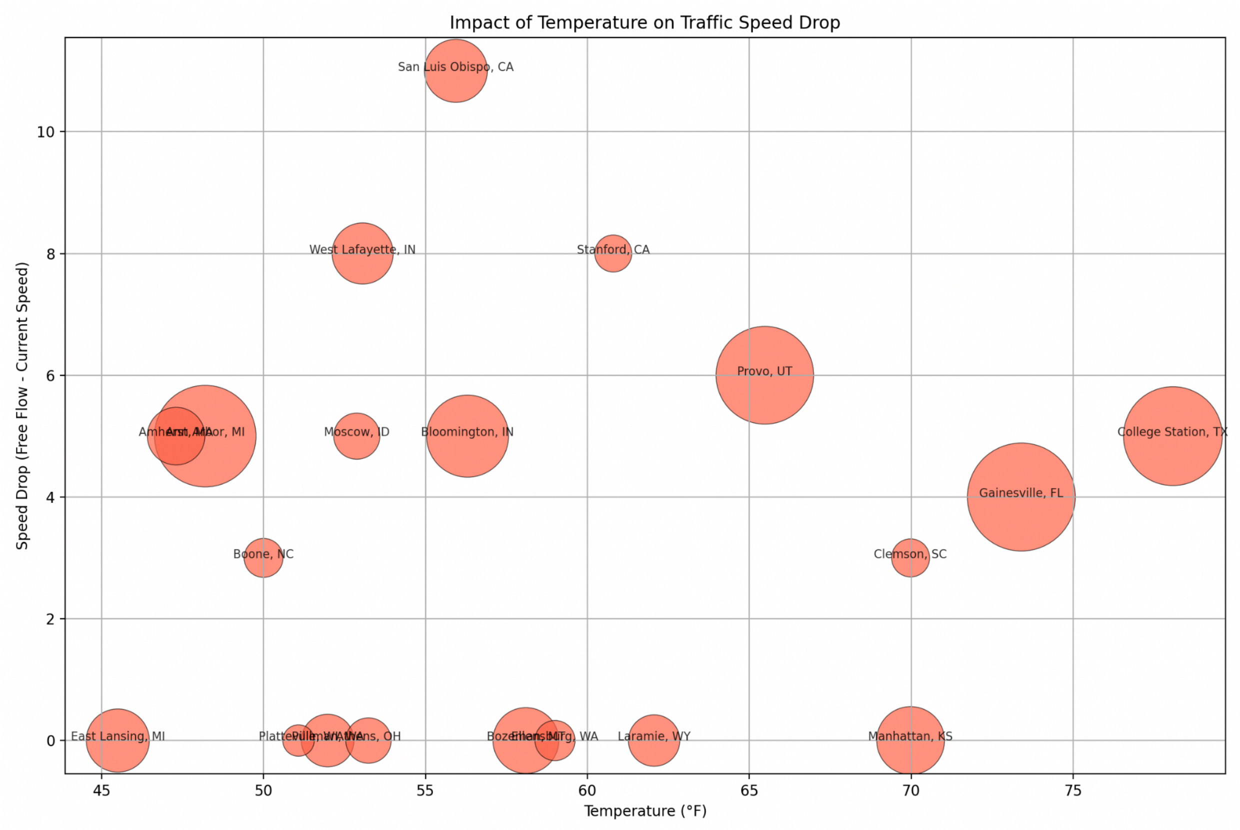
- [x] ~~You must select some data from all of the tables in your database and calculate something from that data (20 points). You could calculate the count of how many items occur on a particular day of the week or the average of the number of items per day.~~
- [x] ~~You must do at least one database join to select your data for your calculations or visualizations (20 points).~~
    - [x] ~~combined.py~~
- [x] ~~Write out the calculated data to a file as text (10 points)~~

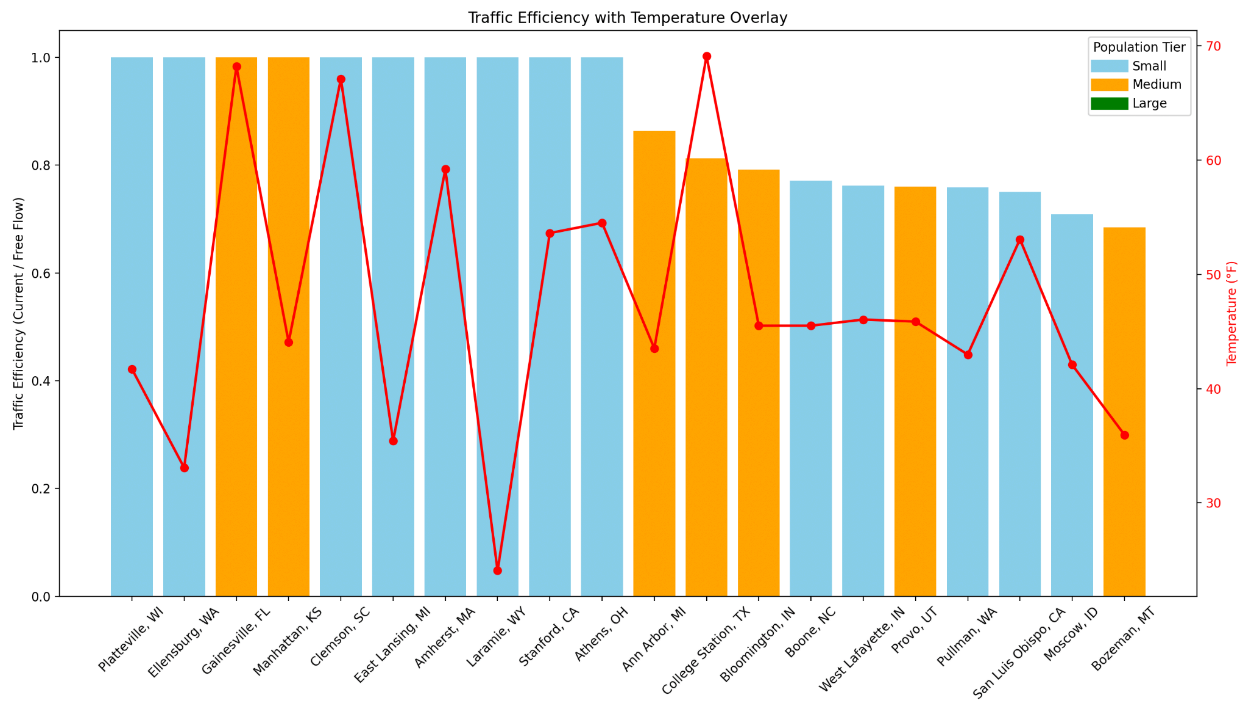
Part 4

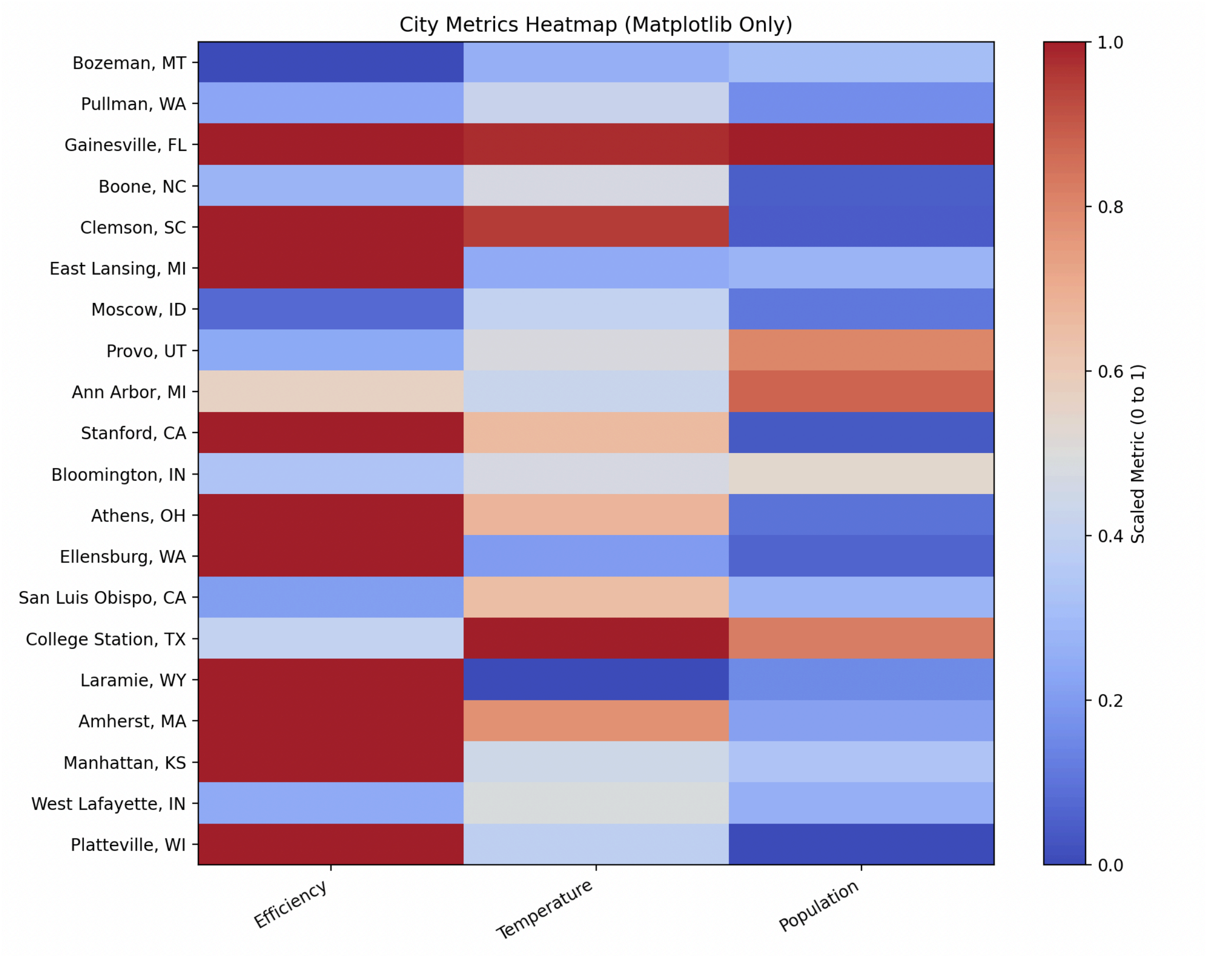
Visualization






Impact of Temperature on Traffic Speed Drop
Speed Drop (Free Flow - Current Speed)
Temperature (°F)
San Luis Obispo, CA
West Lafayette, IN
Stanford, CA
Provo, UT
Amherst, MA
Ann Arbor, MI
Moscow, ID
Bloomington, IN
College Station, TX
Gainesville, FL
Boone, NC
Clemson, SC
East Lansing, MI
Platteville, WI
Pullman, WA
Athens, OH
Bozeman, MT
Ellensburg, WA
Laramie, WY
Manhattan, KS


Traffic Efficiency with Temperature Overlay
Traffic Efficiency (Current / Free Flow)
Temperature (°F)
Population Tier
Small
Medium
Large
Platteville, WI
Ellensburg, WA
Gainesville, FL
Manhattan, KS
Clemson, SC
East Lansing, MI
Amherst, MA
Laramie, WY
Stanford, CA
Athens, OH
Ann Arbor, MI
College Station, TX
Bloomington, IN
Boone, NC
West Lafayette, IN
Provo, UT
Pullman, WA
San Luis Obispo, CA
Moscow, ID
Bozeman, MT


City Metrics Heatmap (Matplotlib Only)
Bozeman, MT
Pullman, WA
Gainesville, FL
Boone, NC
Clemson, SC
East Lansing, MI
Moscow, ID
Provo, UT
Ann Arbor, MI
Stanford, CA
Bloomington, IN
Athens, OH
Ellensburg, WA
San Luis Obispo, CA
College Station, TX
Laramie, WY
Amherst, MA
Manhattan, KS
West Lafayette, IN
Platteville, WI
Efficiency
Temperature
Population
Scaled Metric (0 to 1)
1.0
0.8
0.6
0.4
0.2
0.0

Part 5

Link to Github repo: https://github.com/Tyand12/SI206-Final.git

The Report

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

The goal of our project is to look at data from the top 20 college towns and compare the weather to the population density and how much traffic there is on the main row. We are looking to see how weather and traffic affect the population. We collected APIs from weather reports, census data, and data about the main roads and their travel time. The websites include Tigerweb for map data, AccuWeather for the city weather reports, CensusGov for the yearly census report, and tomtom for the traffic flow.

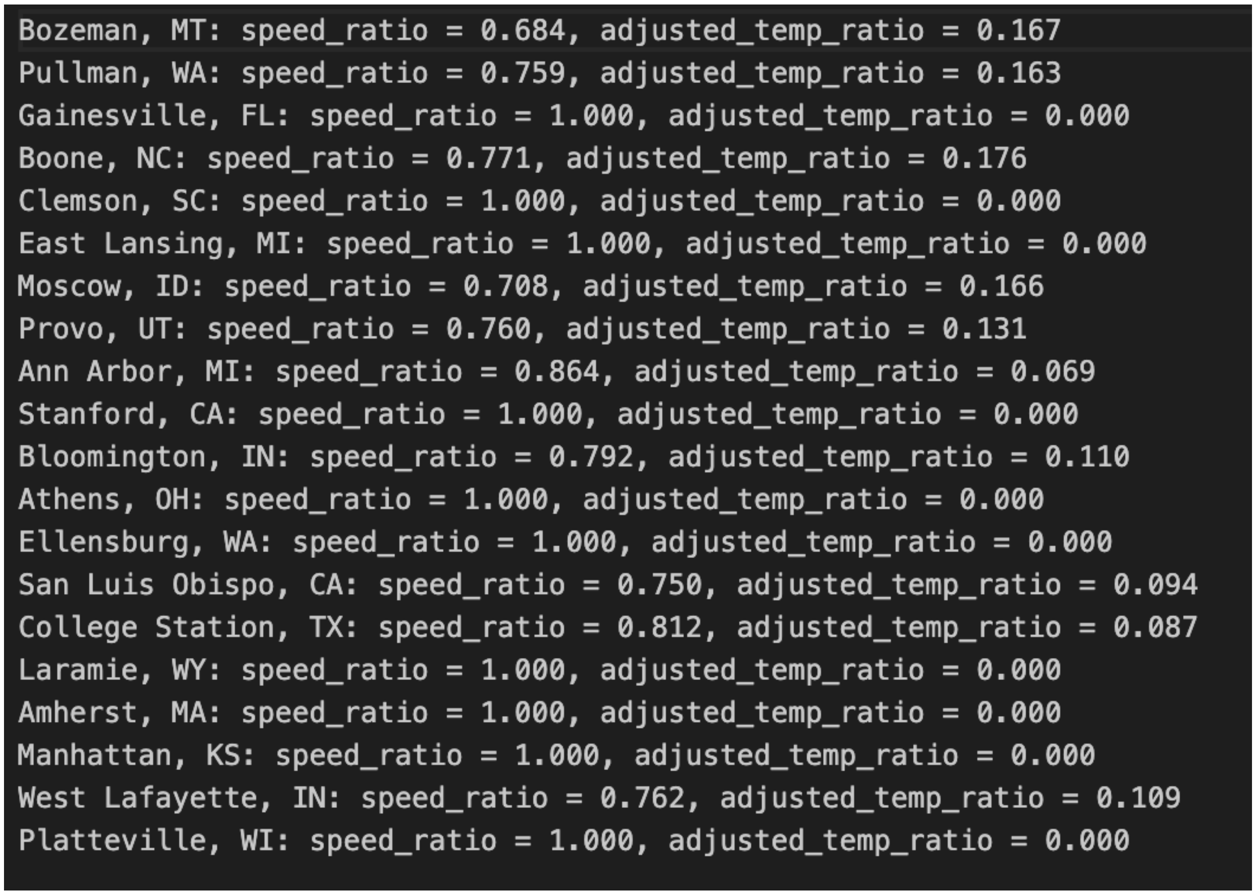
B. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather (10 points)

Using **AccuWeather**, we were able to pull information about the weather from the top 20 college towns. We determined the top 20 college towns by looking at an article from the Indianapolis Star newspaper. We gathered data about the temperature and weather text (i.e. Sunny or Cloudy). This data helps us determine what the weather is in these cities to help us see what it is on a given day. **CensusGov** gave us information about the census reports in terms of the different cities and the designated populations that match those cities. This data helps us determine the density in each city that we chose. **TigerWeb** gave us location data in terms of longitude and latitude. It helped us to narrow our search and when used with our other data, like information on roads, it helps us to determine the correct spot to analyze the density. **TomTom** gave us data on the different road structures in each city. For example, we looked at and analyzed what type of road the main road of the city was, the travel time on that road, and how populated it was. It helped us compare population density and the weather on a given day.

C. The problems that you faced (10 points)

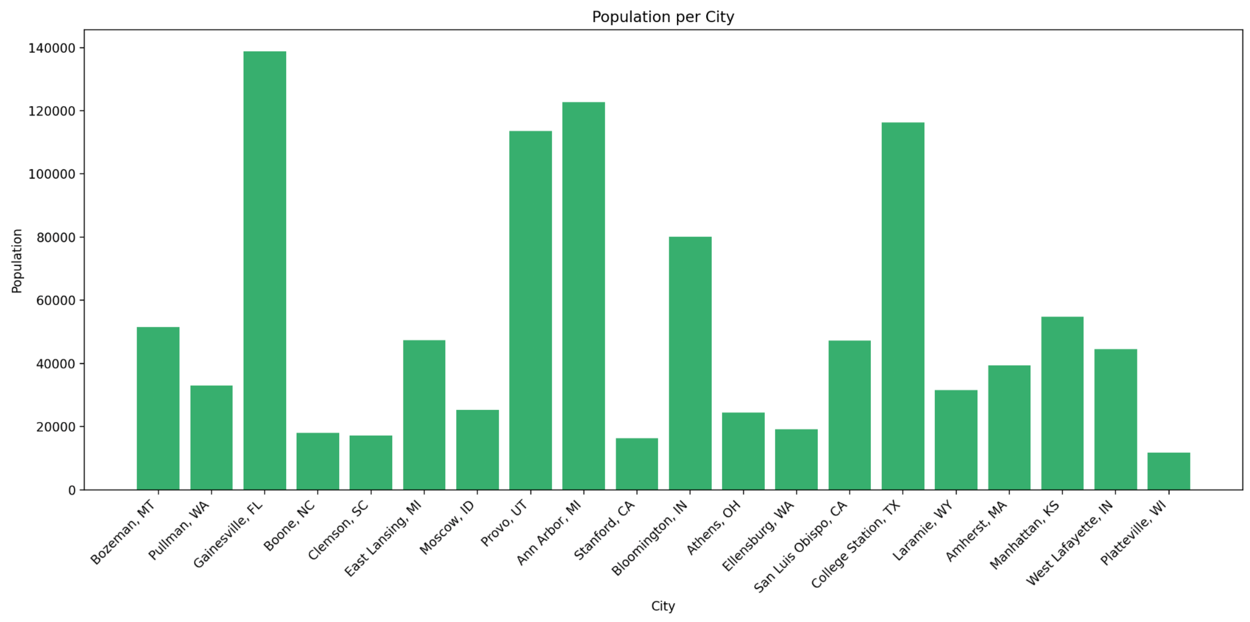
Some problems that we faced were being able to access the APIs more than once in a day. Though we chose APIs from the free list, some of them had limits on how often we could access them. Another issue that we dealt with was figuring out how to match up the data. We had to problem solve a lot and figure out how each part was impacting the other. We had to change some labels around like the state abbreviations from the long version to the abbreviated!

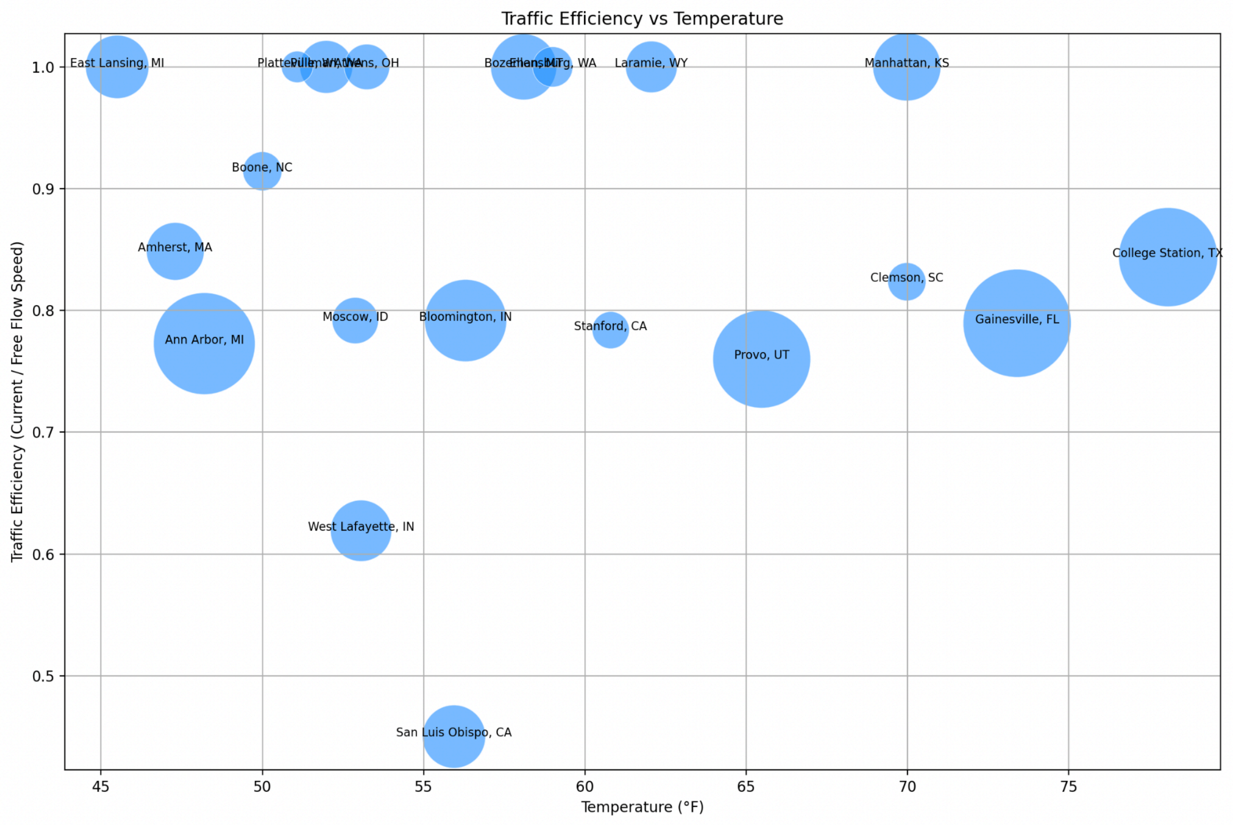
D. The calculations from the data in the database (i.e. a screenshot) (10 points)

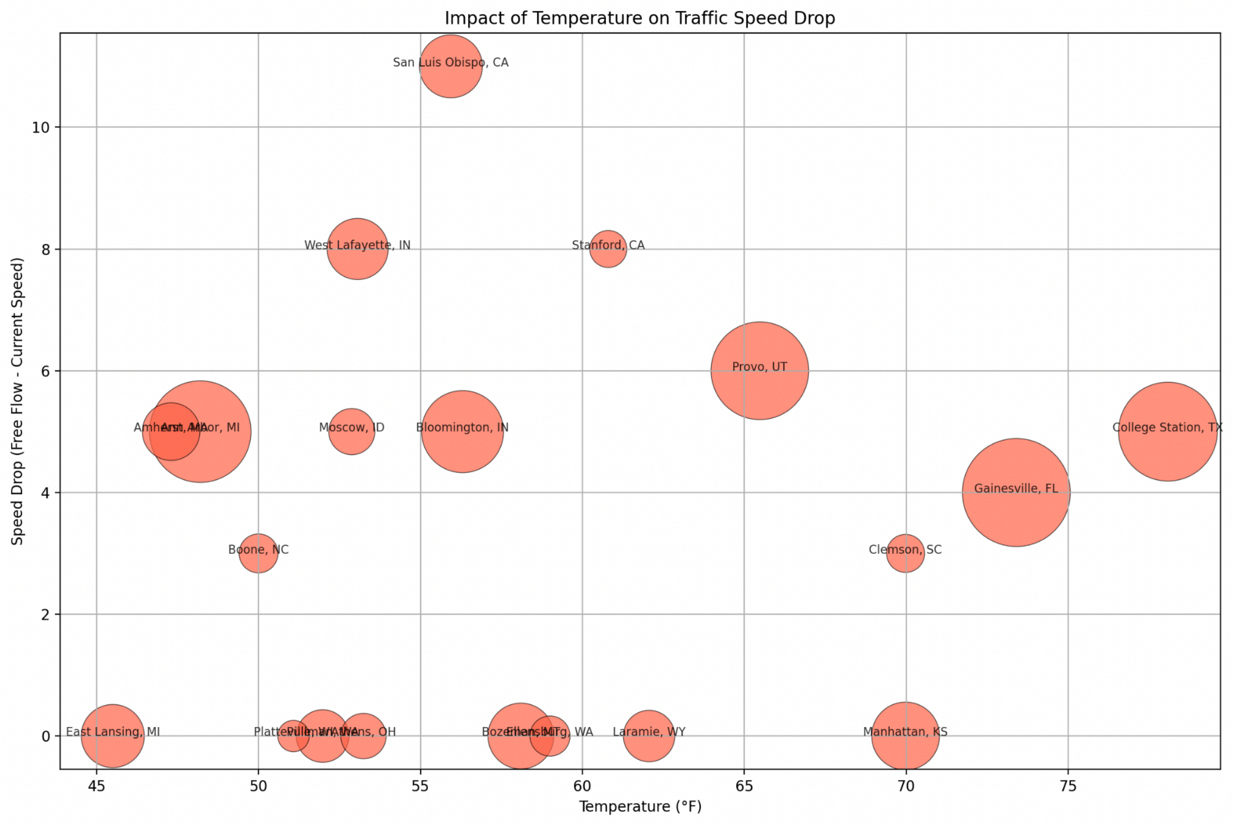


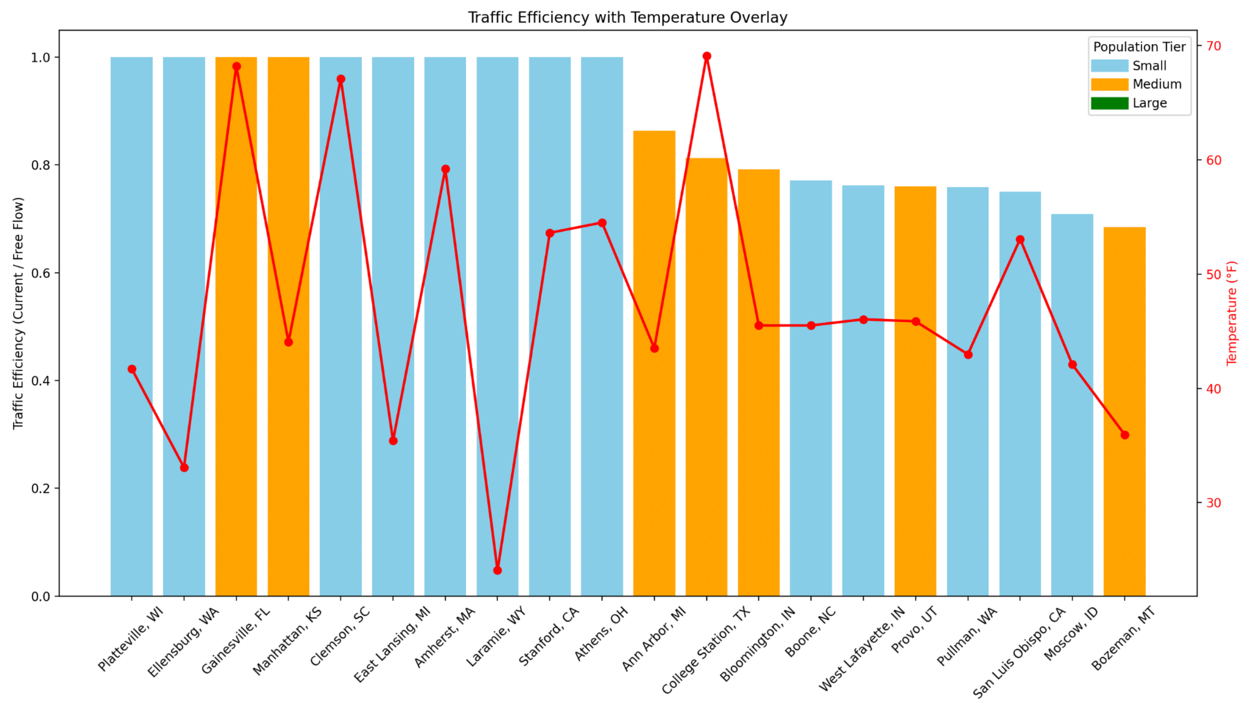
```
Bozeman, MT: speed_ratio = 0.684, adjusted_temp_ratio = 0.167
Pullman, WA: speed_ratio = 0.759, adjusted_temp_ratio = 0.163
Gainesville, FL: speed_ratio = 1.000, adjusted_temp_ratio = 0.000
Boone, NC: speed_ratio = 0.771, adjusted_temp_ratio = 0.176
Clemson, SC: speed_ratio = 1.000, adjusted_temp_ratio = 0.000
East Lansing, MI: speed_ratio = 1.000, adjusted_temp_ratio = 0.000
Moscow, ID: speed_ratio = 0.708, adjusted_temp_ratio = 0.166
Provo, UT: speed_ratio = 0.760, adjusted_temp_ratio = 0.131
Ann Arbor, MI: speed_ratio = 0.864, adjusted_temp_ratio = 0.069
Stanford, CA: speed_ratio = 1.000, adjusted_temp_ratio = 0.000
Bloomington, IN: speed_ratio = 0.792, adjusted_temp_ratio = 0.110
Athens, OH: speed_ratio = 1.000, adjusted_temp_ratio = 0.000
Ellensburg, WA: speed_ratio = 1.000, adjusted_temp_ratio = 0.000
San Luis Obispo, CA: speed_ratio = 0.750, adjusted_temp_ratio = 0.094
College Station, TX: speed_ratio = 0.812, adjusted_temp_ratio = 0.087
Laramie, WY: speed_ratio = 1.000, adjusted_temp_ratio = 0.000
Amherst, MA: speed_ratio = 1.000, adjusted_temp_ratio = 0.000
Manhattan, KS: speed_ratio = 1.000, adjusted_temp_ratio = 0.000
West Lafayette, IN: speed_ratio = 0.762, adjusted_temp_ratio = 0.109
Platteville, WI: speed_ratio = 1.000, adjusted_temp_ratio = 0.000
```

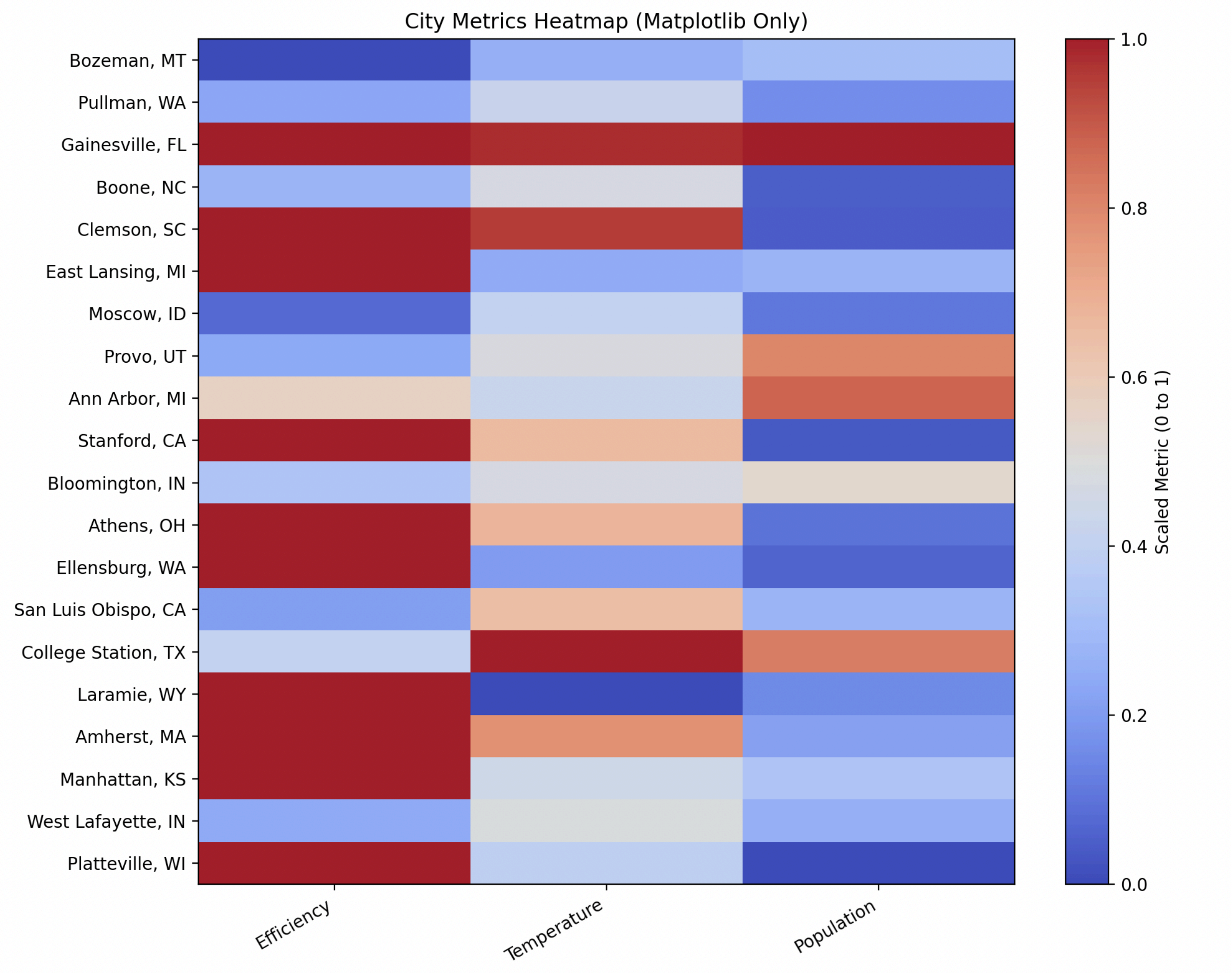
E. The visualization that you created (i.e. screenshot or image file) (10 points)




Traffic Efficiency vs Temperature
Traffic Efficiency (Current / Free Flow Speed)
Temperature (°F)
East Lansing, MI
Boone, NC
Amherst, MA
Ann Arbor, MI
Moscow, ID
Bloomington, IN
Stanford, CA
Provo, UT
Clemson, SC
Gainesville, FL
College Station, TX
West Lafayette, IN
San Luis Obispo, CA
Laramie, WY
Manhattan, KS


Impact of Temperature on Traffic Speed Drop
Speed Drop (Free Flow - Current Speed)
Temperature (°F)
San Luis Obispo, CA
West Lafayette, IN
Stanford, CA
Provo, UT
Moscow, ID
Bloomington, IN
College Station, TX
Gainesville, FL
Boone, NC
Clemson, SC
East Lansing, MI
Laramie, WY
Manhattan, KS


Traffic Efficiency with Temperature Overlay
Traffic Efficiency (Current / Free Flow)
Temperature (°F)
Population Tier
Small
Medium
Large
Platteville, WI
Ellensburg, WA
Gainesville, FL
Manhattan, KS
Clemson, SC
East Lansing, MI
Amherst, MA
Laramie, WY
Stanford, CA
Athens, OH
Ann Arbor, MI
College Station, TX
Bloomington, IN
Boone, NC
West Lafayette, IN
Provo, UT
Pullman, WA
San Luis Obispo, CA
Moscow, ID
Bozeman, MT


City Metrics Heatmap (Matplotlib Only)
Bozeman, MT
Pullman, WA
Gainesville, FL
Boone, NC
Clemson, SC
East Lansing, MI
Moscow, ID
Provo, UT
Ann Arbor, MI
Stanford, CA
Bloomington, IN
Athens, OH
Ellensburg, WA
San Luis Obispo, CA
College Station, TX
Laramie, WY
Amherst, MA
Manhattan, KS
West Lafayette, IN
Platteville, WI
Efficiency
Temperature
Population
Scaled Metric (0 to 1)

F. Instructions for running your code (10 points)

1. Open Python

2. Open all of the git files

3. Create a file to store your personal API Keys

3. In order, run weather file, census file, and traffic file

4. After all the data is collected, run the combined script so that you can calculate all the combined database data

5. Run the visualization script to see the graphs that depict the data visualizations and patterns

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

Ty, traffic data

Def create_database():

- This function creates the SQLite database and stores the traffic data from the different locations we were observing. It records the id number, city, frc, current speed, free flow speed, confidence, and road closure
- Input: nothing
- Output: None: creates database

Def fetch_traffic_data(lat, lon):

- Input: latitude + longitude
- Output: None
- This function makes the request to the API to get the information we are looking for. It utilizes an API key to access and store the data

Def insert_traffic_data(city, data):

- Input: city + data
- Output: None; adds data to the database
- This function takes the data for a city and adds it to the database

Def fetch_and_store_all_data():

- Input: None

- Output: None; stores data for defined points
- This function uses data selection to utilize the insert_traffic_data for each city, ensuring that they all get implemented
- Limits the database to create two tables, with with 100 rows, one with 20

if __name__ == '__main__':

- Input: None
- Output: None; adds and records that data into the database
- This function initializes and loops the program

Jolie, weather data

def create_weather_table():

- This function creates an SQLite database to store weather data from various cities. It only creates a table if one does not already exist. It includes columns for city, location key, observation time, temperature, weather text, and whether it is daytime.
- Input: does not take in anything
- Output: returns None. The function just creates a weather table in the database

def get_location_key(city):

- This function retrieves the location key for a given city using the AccuWeather API. Location key is required for fetching the data for each city.
- Input: city (str). The name of the city for which the location and information is being fetched
- Output: Returns a str containing the location key if the request is successful. Returns None if request fails or no data is found for the city.

def fetch_weather_data(location_key):

- This function fetches the current weather data using the API. Returns relevant weather details and data that can be found in the def create_weather_table() function.
- Input: location_key (str) which is the location key to fetch the weather data
- Output: returns a dictionary with keys including observation_time (str), temperature (float), weather_text (str), and is_daytime (int). They are all self explanatory. Some things to note is that temperature is the current

temperature in Celsius so we ran a calculation to convert it to Fahrenheit. It returns None if there is an error.

def celsius_to_fahrenheit(celsius):

- This is the calculation function that I mentioned above to convert the temperature from Celsius to Fahrenheit
- Input: Celsius
- Output: temperature in Fahrenheit

def store_weather_data(max_new=20, total_limit=100):

- This function stores up to max_new weather records into a SQLite database. It ensure that is does not exceed the total limit in accordance with the assignment directions
- Input: max_nex (int) and total_limit (int). The default of max is 20 and the default of total is 100
- Output: returns None. It inserts new weather data into the Weather table in the database and outputs a message indicating how many records were added.

if __name__ == '__main__':

- This initializes the program and loops.
- Input is none.
- Output is print statements indicating the success or failure of adding weather records for each city.

Tess, census data

def setup_database(db_name=”city_data.db”):

- This function creates an SQLite database to store population data from the chosen cities. It only creates a table if one does not already exist. It includes columns for city, population, state code, and place code.
- Input: database name with a default database name of city_data.db
- Output: returns None. The function just creates a weather table in the database

def get_city_population(place_code, state_code):

- This function takes the url for the census data and the parameters to create the desired urls for the desired cities. It then retrieves the population data for the desired city.

- Input: place_code (str) which is the code for the city in the US census data, state_code (str) the code for the state according to the US census data
- Output: Returns the population for the desired city

def populate_database(cities, conn, cursor):

- This function loops through each city in the cities dictionary and uses the get_city_population() function to collect the population of the city and stores the data into the database created from the setup_database() function.
- Input: cities (dict) which is a dictionary of the city names with a value of a tuple with the place code, state code, and the state abbreviation for each city, conn, cursor
- Output: Returns None it just inserts the city, population, state code, and place code into the database

def main():

- This initializes the program and loops.
- Input is none.
- Output is print statements indicating the success or failure of adding weather records for each city.

Visualizations:

Popbar.py

- Input: rows (list of tuples). The list contains city names (str) and populations (int).
- Output: None. The function generates and displays a bar chart with the population data.

def fetch_data():

- Fetches data from the combined data database
- Input: none
- Returns: data from the combined database to use the data for the visualizations

def plot_bubble_chart():

- Creates a bubble chart that compares city traffic efficiency vs temperature in order to show how temperature might be impacting traffic flow efficiency

- Input: data retrieved from fetch_data function
- Returns: none but plots the graph

def plot_temperature_vs_speed_drop():

- Creates a scatter plot that compares temperature vs speed drop in order to visualize how temperature might affect traffic degradation
- Input: data retrieved from fetch_data function
- Returns: none but plots the graph

def plot_traffic_vs_temperature_overlay

- Creates a population bar chart with temperature overlay that shows the traffic efficiency by city goal which compares traffic performance across all 20 cities
- Input: data retrieved from fetch_data function
- Returns: none but plots the graph

def normalize_column():

- It normalizes a list of numbers so that each number is scaled between 0 and 1
- Input: column of data from database
- Returns: (int) returns the values between 0-1

def plot_heatmap_matrix():

- Creates a heatmap matrix that shows cities vs variables goal which compares all metrics for all cities in one chart (the variables are traffic efficiency, temperature, and population)
- Input: data retrieved from fetch_data function
- Returns: none but plots the graph

Combined

Def created_combined_db()

- Input: none
- Output: none
- This function creates the SQLite database and stores the data from the different APIs we were observing

Def copy_and_merge_data()

- Input: none
- Output: none
- This functions copies all of the information from each other database and moves it into this one

if __name__ == '__main__':

- This initializes the program and loops.
- Input is none.
- This loops through and does it for each row

Def decode_strings(df, columns)

- Input: None, both defined in function
- Output: encoded data frame and mapping dictionaries
- This function works replace strings with defined integer codes

Def load_encoded_data()

- Input: none
- Output: encoded database
- Loads the combined table from database and returns an encoded version

if __name__ == '__main__':

- This initializes the program and loops.
- Input is none.
- This loops through and does it for each row

Calculations

Def compute_ratios_and_write_to_file(db_file = combined_data.db, output_file = "calculations.txt")

- Input: combined database and name of text file
- Output: text file
- This function utilizes and selects and joins different tables of the database and computes equations to write a text file where it stores the answers

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?)

4/10	Having problem with getting the DB to run from the file	Chat GPT	Yes, it gave me helpful pointers
4/15	Help with merging files	Chat GPT	Yes

Changes after grading session:

- Made sure that all databases stored up to 100 but only 20 at a time
- Added a JOIN
- Created 2 extra visualizations for extra credit