*Project Goals*

The goal of this project was to receive some insight on the traffic conditions in Ann Arbor, and how they correlate with the weather conditions. Using OpenWeather API and TomTom API, we hope to use our newfound Python skills to gather data from the APIs, create databases in SQLite3, edit those databases, and calculate something from the gathered data. We will hopefully get information on live traffic data and incident data and see how the weather conditions (temperature and descriptions) influence that data. Once we have gathered this data, we hope to use Matplotlib to create graphs of the yearly weather in Michigan, the yearly incident data in Michigan, and then a graph showing the correlation between the two.

*Achieved Goals*

Unfortunately, we were not able to use the incident data API from TomTom, nor could we get the previous year’s weather or traffic data because the API did not store a year’s worth of data. Instead, we pivoted our project to get data from 12-04-19 to 12-11-19 on live traffic and weather data. We were able to collect data on traffic conditions, including speed, travel time, road closures, and the confidence of the predicted traffic data. We defined confidence as the probability or likelihood that the traffic predictions are accurate and likely to occur, which was important in identifying how people would respond to changes in weather conditions. Through this, we also collected weather data including the current temperate, the maximum temperature for that day, the minimum temperature for that day, the general weather conditions, and then a description of those weather conditions. From that data, we were able to calculate the average confidence of traffic conditions, the average speed, the average temperature, and average weather condition.

*Data Collection*

We collected our data in the SQLite3 Database. The Weather (Figure 1) and Temperature Figure 2) tables share a key, and the Traffic Flow (Figure 3) and Confidence (Figure 4) share a key. Each time we collected data, we only collected twelve lines of data. Five from TrafficFlow, five from Confidence, one from Temperature and one from Weather. We used a timestamp as a key, so weather items with different timestamps are not duplicate data. Attached are screenshots that show what some of our tables looked like.

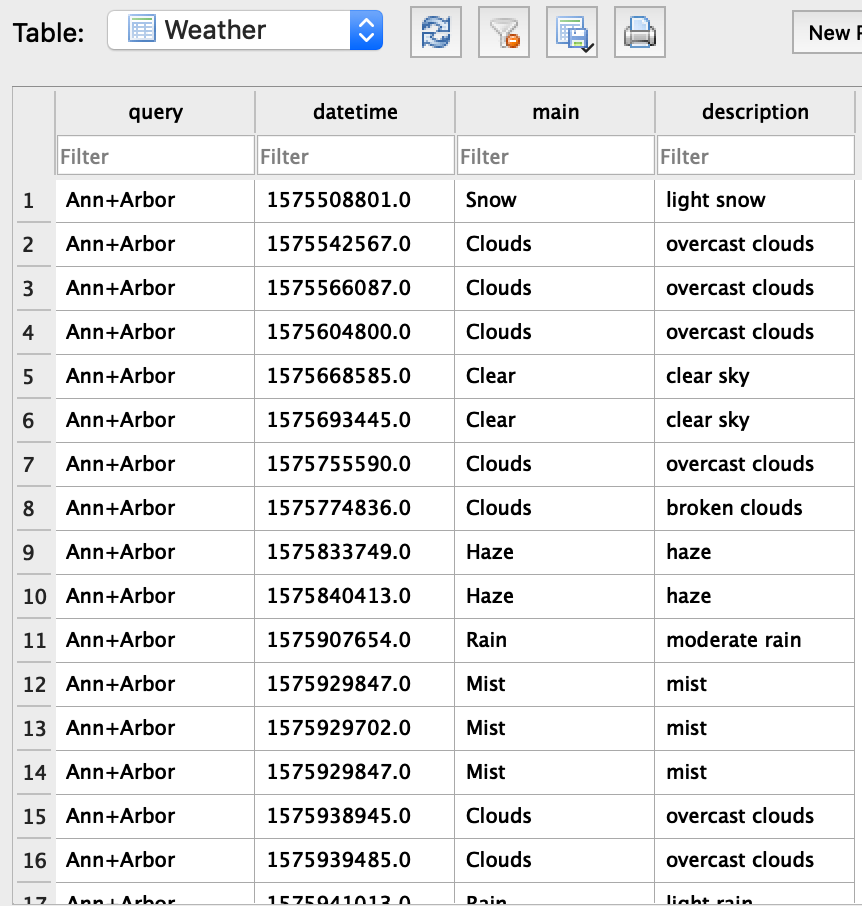
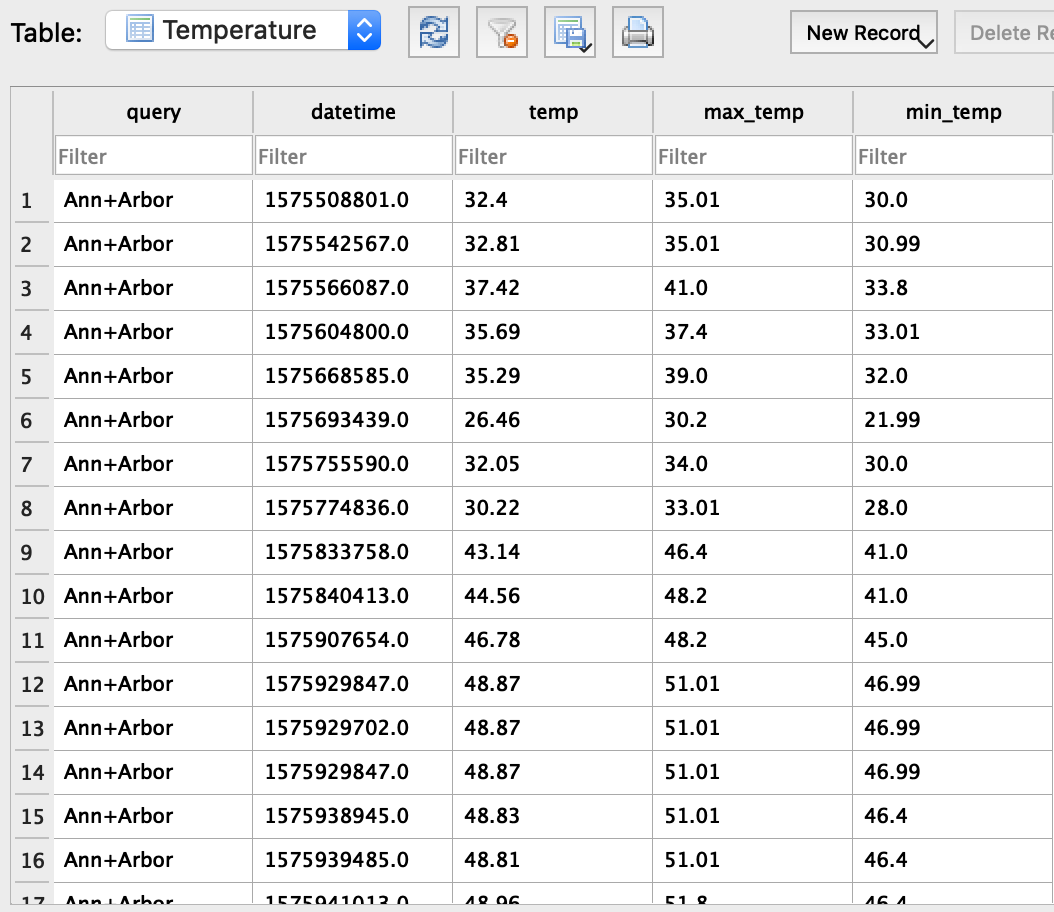


Figure 2

Figure 1

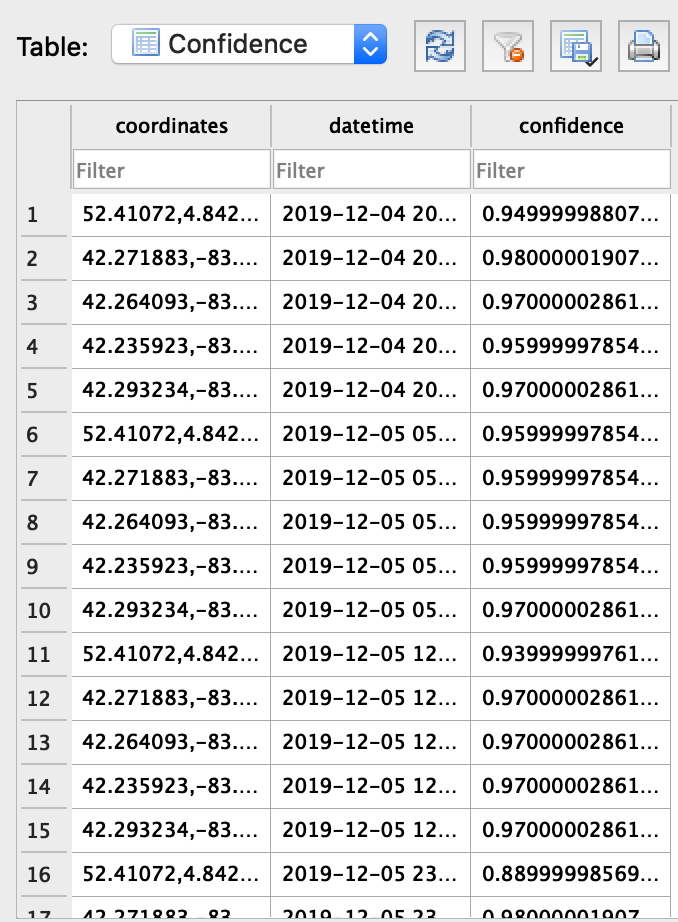
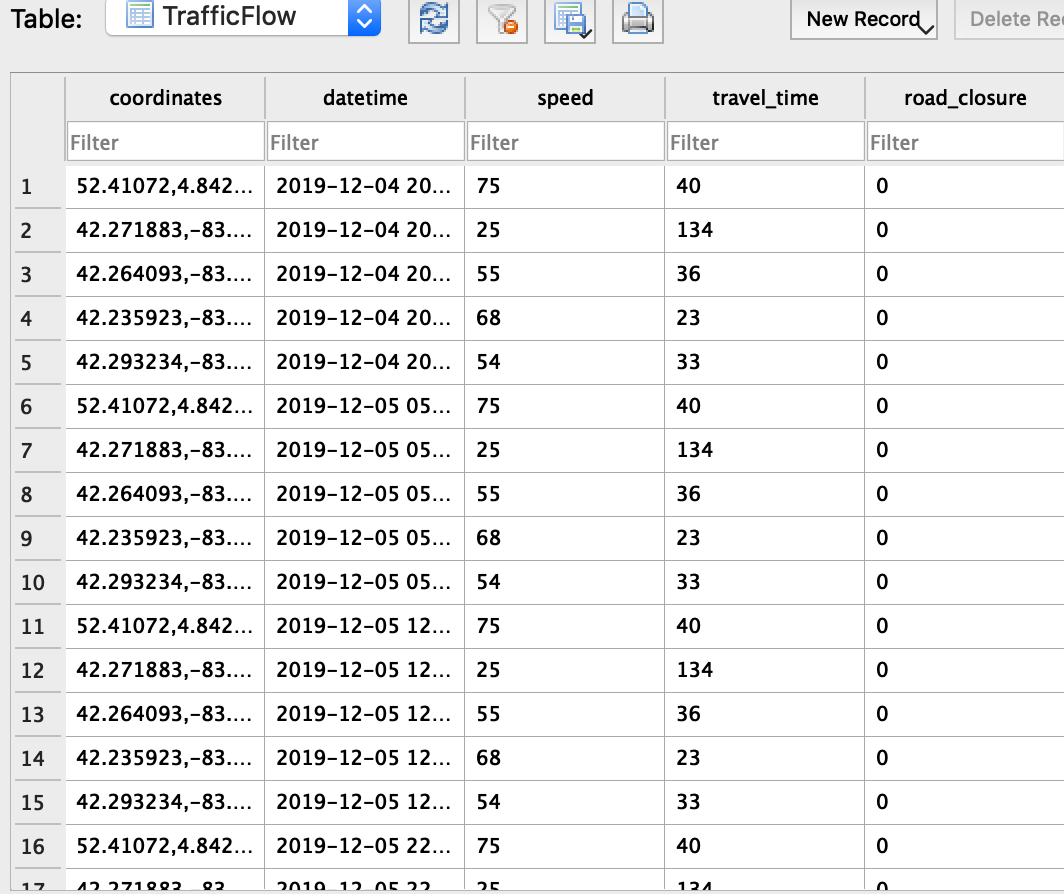


Figure 4

Figure 3

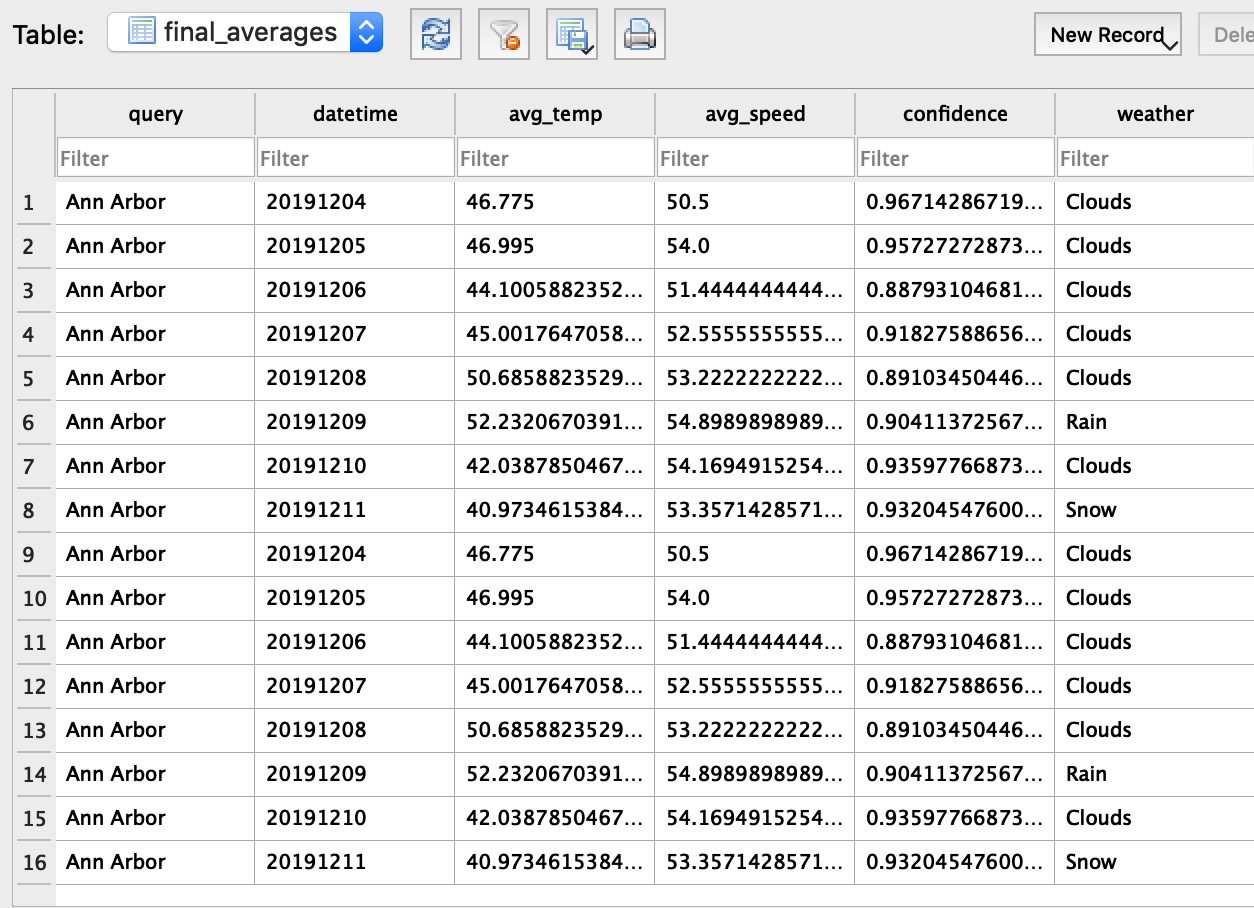
We calculated the averages of the tables and wrote them to a new database, using data base join, called final\_averages (Figure 5). We wrote the contents of this calculated file to a text file (Figure 6).

Figure 5

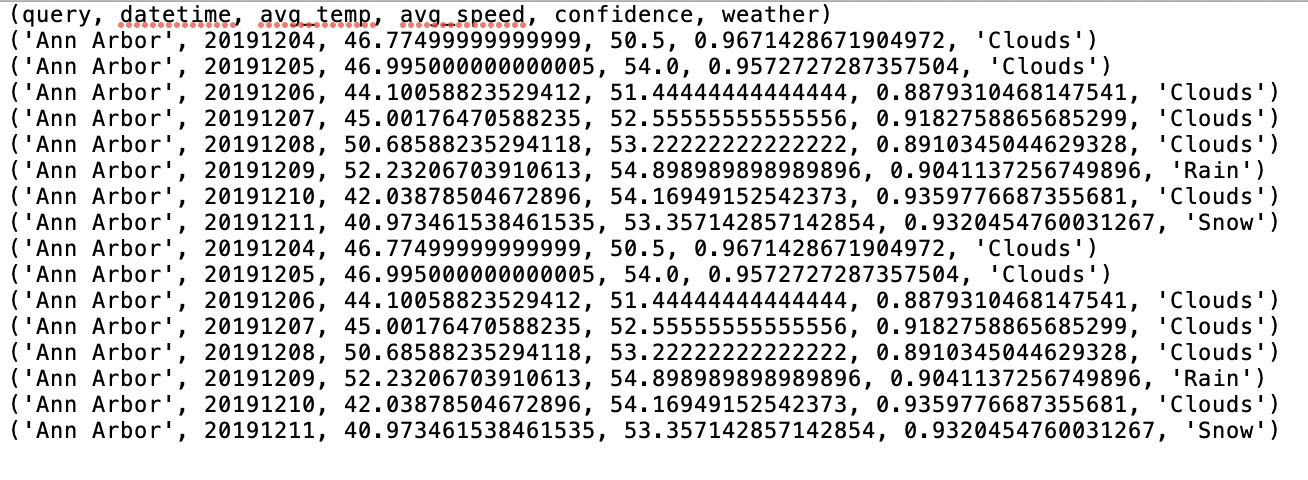


Figure 6

*Visualizations*

To better understand this calculated data, we created four visualizations using Matplotlib. The first visualization is the average confidence level in traffic reports given the weather conditions (bar graph), which showed that the confidence was highest when snowing, second highest when cloudy, and least confident when raining. This means that traffic conditions were more successfully predicted with snowy conditions (with a 93% confidence) while rainy conditions created more varied traffic patterns that were not as successfully predicted (with a 90.5% confidence).

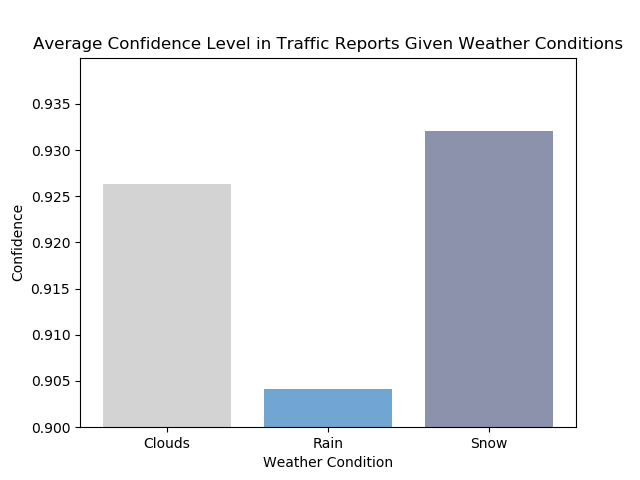


Figure 7

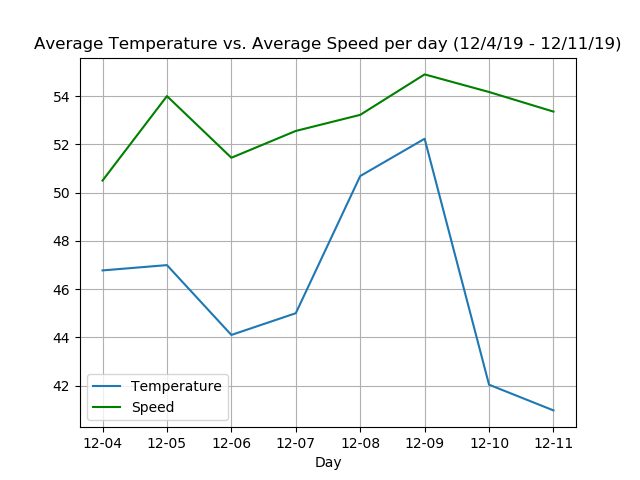
The second visualization used a double line graph to show the average temperature versus the average speed per day from 12-4-19 to 12-11-19. This demonstrated a direct relationship between temperature and average speed as speed increased, in conjunction to temperature and vice versa.

Figure 8

The third visualization utilized a line graph with two subplots (due to the y axis values being so far apart) and graphed the temperature versus the average confidence in traffic conditions per day. This graph showed that when the temperature was higher, the confidence in traffic predictions was lower.

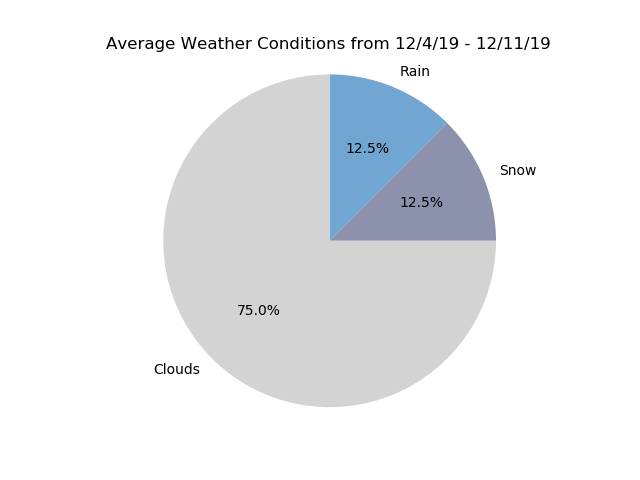
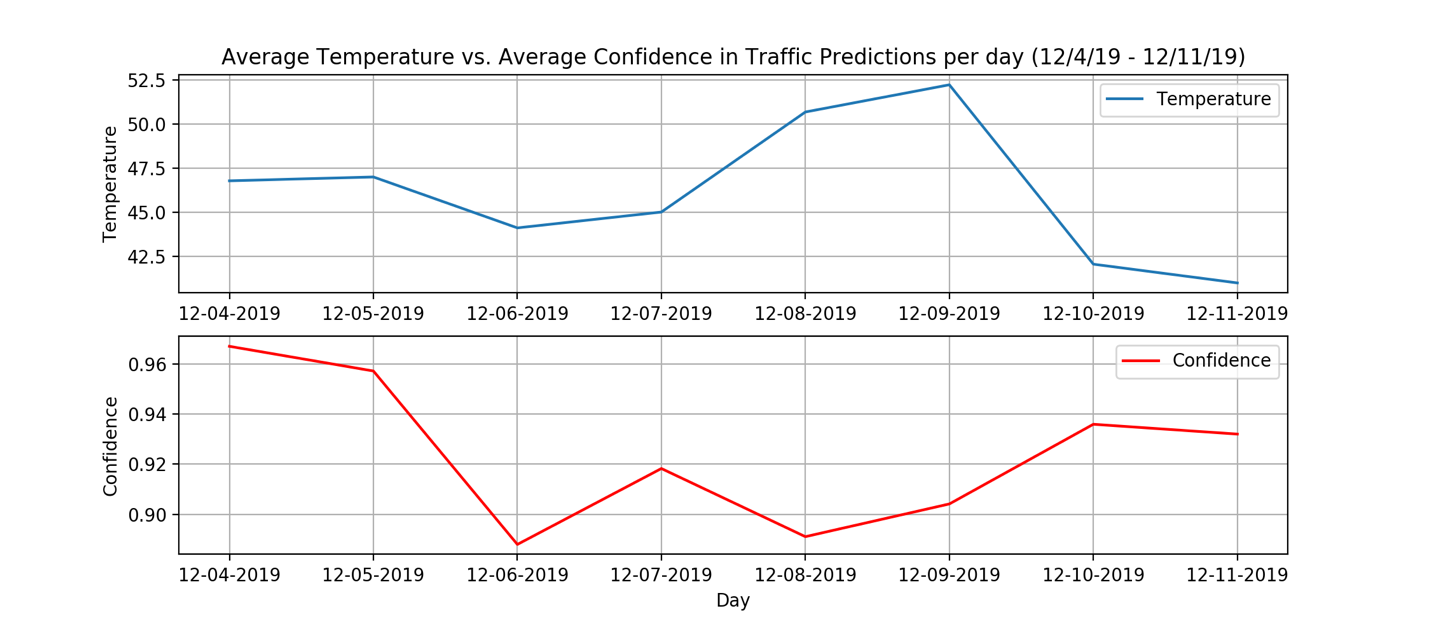
**The final graph, a pie chart, showed the average weather conditions from 12-4-19 to 12-11-19. It reported that 75% of the time, the weather reported clouds, 12.5% of the time it reported snow, and the other 12.5% of the time, it reported rain.

Figure 9

Figure 10

*The Problems Faced*

One of the first problems we faced in this project was that we were unable to get the live incident data from the TomTom API.This was due to the fact that when we input a versionNumber (as we were required to according to their documentation for the API), it would cause an error. We were also unable to retrieve the past years live traffic data, because the API did not store this data. However, we were still able to complete the project without either of these data points.

In order to collect the live traffic data, we had to create a bounding box, an area for the traffic data to be collected between, in Ann Arbor. We chose coordinates on Washtenaw Ave, Packard & Hill, W. Stadium Blvd, Saline Road, and N. Maple Road. However, because of the way that the TomTom API is set up, we had to convert the latitude and longitude to the World Geodetic System (1984) using an online converter. This took some time to understand, but once completed we were able to move forward with the project.

After we had gathered all of our data, we realized that we had collected the datetime in a different format in the traffic data than we had in the weather data. The weather data datetime was already available in the OpenWeather API, but the traffic data datetime was not available so we had to generate our own. We ended up gathering the weather data in Unix epoch time, and the traffic data in human-readable data. After we converted the date times, we realized that the time stamps were also recording minutes and seconds. Because the code runs top to bottom, it meant that not all of our timestamps would match up, even if they were run at roughly the same time. Because of this, we decided to collect the data per day, rather than per hour, minute or second. This was crucial to us being able to successfully do the database join and get matches for the data.

When we went to make a pie chart out of the weather descriptions, we realized that we would need to calculate the percent of each weather description using a counter in order to use the calculated average data, since the data is not numerical. This was not a tough problem to solve, and we were able to use our Python skills to implement the necessary code. For the sub-plot line graph that we created (Figure 3), we had initially intended to create this as a double line graph. However, because the Y axis values for temperature (in the 40s) was drastically different from the Y axis values for the confidence (all below 1), we decided to split the plot into two sub plots, while still allowing for an effective visualization of the data. Additionally, we initially had some trouble in creating the x and y axis for the subplots due to an issue with Matplotlib. Prior to using the set tick labels function, we were having trouble trying to get the first date to appear on the graph. This was causing the scale to be off, where the last tick of the x axis was empty. Therefore, we had to create an empty string placeholder in order of correcting the x-axis by using a similar format listed in forums.

*Instructions for running the code*

Within the code, there are instructions as well, instructing users what to uncomment. Here are also the detailed steps that we took to collect the data, calculate the averages, and then visualize the output.

1. The first four functions (weather\_conditions, min\_max\_temp, traffic\_Data, and report\_confidence) all have print statements underneath them. The weather data is collected by the first two functions and allows for users to input any location, and then collect that data if they uncomment lines 56 and 57. The traffic data is collected within the second two functions. If users wished to set a different boundary box than we have set, they must edit the value of the coordinate’s variable (line 62), which requires 5 coordinates in the World Geodetic System (1984). Then, they could uncomment lines 125 and 126 to run the code and collect the data.
2. After collecting as much data as the user needs, they can move onto the clean\_data function, where all of the timestamps will be converted to a uniform, human-readable format. In order to run this function, they must uncomment line 196.
3. To take the average of the traffic speed per day, users must uncomment line 231.
4. To take the average of the temperature averages per day, users must uncomment line 266.
5. To take the average of the confidence in traffic data per day, users must uncomment line 302.
6. To take the average of the weather description per day, users must uncomment line 342.
7. To join the data of all of the averages into a table called final\_averages, users must uncomment line 399.
8. To write the data to a text file called data.txt, users must uncomment line 418.
9. To create the pie chart of the average weather conditions per day (pie.png), the subplot line graph of the temperature versus average confidence in traffic conditions per day (line.png), the double line graph of temperature versus the average speed per day (doubleline.png), and the bar graph of the average confidence level in traffic reports given the weather conditions (bar.png), users must uncomment line 568.

*Documentation for each function written (input/output for each)*

The expected input, function, and output of each function is also located beneath each class name in the code. It is further detailed here.

|  |  |  |  |
| --- | --- | --- | --- |
| **Class Name** | **Input** | **Function** | **Output** |
| weather\_conditions | Query to select location to search on weather API (EX: ‘Ann+Arbor’) | Calls to OpenWeather API and collects weather data (query, datetime, main, description) for given query and inserts them into table Weather. | Returns success statement re. adding data into the database |
| min\_max\_temp | Query to select location to search on weather API (EX: ‘Ann+Arbor’) | Calls to OpenWeather API and collects temperature data (query, datetime, temp, max\_temp, min\_temp) for given query and inserts them into table Temperature. | Returns success statement re. adding data into the database |
| traffic\_Data | Takes in a list of coordinates to select locations to search for on traffic data API. Must be in the World Geodectic System (1984) format and include 5 coordinate tuples | Calls to TomTom Developer API and collects traffic data (coordinates, datetime, speed, travel\_time, & road\_closure) for the given coordinates and inserts them into table TrafficFlow. | Returns success statement re. adding data into the database |
| report\_confidence | Takes in a list of coordinates to select locations to search for on traffic data API. Must be in the World Geodectic System (1984) format and include 5 coordinate tuples | Calls to TomTom Developer API and collects traffic data (coordinates, datetime, confidence) for the given coordinates and inserts them into data table Confidence. | Returns success statement re. adding data into the database |
| clean\_data | None | Cleans data by converting timestamps to uniform human-readable format and inserting them into tables (temp\_clean, traffic\_clean, confidence\_clean, weather\_clean) | None |
| take\_traffic\_averages | None | Takes the data from traffic\_clean and generates the average traffic speed per day. Then adds this data to table traffic\_averages. | Dictionary with traffic averages per day, with the day as keys and the traffic average as values. |
| take\_temp\_averages | None | Takes the data from temp\_clean and generates the average temperature per day. Then adds this data to the table temp\_averages. | Dictionary with temperature averages per day, with the day as keys and the temperature average as values. |
| take\_confidence\_averages | None | Takes the data from confidence\_clean and generates the average confidence in traffic reports per day. Then adds this data to the table confidence\_averages. | Dictionary with confidence averages per day, with the day as keys and the confidence average as values. |
| take\_weather\_averages | None | Takes the data from weather\_clean and generates the average weather per day. Then adds this data to the table weather\_averages. | Dictionary with weather averages per day, with the day as keys and the weather average as values. |
| join\_data | None | Joins data from the previous tables created to store averages to create final\_averages table in the database. | Dictionary with joined data from the 4 averages tables. |
| write\_data | A filename | Writes joined data to a text file, saved as data.txt. | None |
| visualize\_data | None | Creates 4 visualizations of the data in final\_averages using Matplotlib: single line graph (line.png), double line graph (doubleline.png), pie chart (pie.png), and bar chart (bar.png). Saves these images in folder. | 4 plot images |

*Documentation for resources used*

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Issue Description** | **Location of Resource** | **Result (did it solve the issue)?** |
| 12/03/19 | Needed information on how to set up the OpenWeather API. | <https://openweathermap.org/current> | Yes, we were able to successfully set up the OpenWeather API. |
| 12/03/19 | Needed information on how to set up the TomTom API. | <https://developer.tomtom.com/traffic-api/traffic-api-documentation-traffic-flow/flow-segment-data> | Yes, we were able to successfully set up the TomTom traffic report API. |
| 12/03/19 | Needed to convert the latitude/longitude of our coordinates to the World Geodetic System (1984). | <https://epsg.io/transform#s_srs=3857&t_srs=4326> | Yes, our conversion was successful. |
| 12/11/19 | Needed the conversion from Unix epoch time to human-readable time. | <https://www.epochconverter.com/> | Yes, our conversion was successful. |
| 12/12/19 | Needed information on how to do a pie chart on Matplotlib. | <https://matplotlib.org/3.1.1/gallery/pie_and_polar_charts/pie_features.html> | Yes, we were able to create the pie chart. |
| 12/12/19 | Needed information on how to do a subplot graph on Matplotlib. | <https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.subplot.html> | Yes, we were able to create the subplot. |
| 12/12/19 | Needed information on how to create the x and y labels, as well as, ticks on line graph | <https://jakevdp.github.io/PythonDataScienceHandbook/04.10-customizing-ticks.html> | Yes, we were able to fix the x axis so that each tick represented the different dates collected |
| 12/12/19 | Needed information on how to do a double line graph on Matplotlib. | <https://www.kaggle.com/andyxie/matplotlib-plot-multiple-lines> | Yes, we were able to create the double line graph. |
| 12/12/19 | Needed information on how to do a bar chart on Matplotlib. | <https://pythonspot.com/matplotlib-bar-chart/> | Yes, we were able to create the bar chart. |
| 12/14/19 | Needed information on how to change each bar in the bar graph | <https://python-graph-gallery.com/3-control-color-of-barplots/> | Yes, we effectively changed the colors for better color contrast |