206 Final Report

Julia Reel, Logan Hanekamp, Lucy Deerin

Original Goals

Our goals for this project were to learn about the relationships between weather, car crashes, and population in certain locations. Initially, we planned to use Visual Crossing Weather API, NHTSA Crash Viewer API, and Wolfram Alpha Website Search for Population. Our plan was to use at least 2 APIs and 1 website and create at least 3 visualizations. The original plan was to calculate the ratio of car crashes by population for each state given the average weather that year. Another goal we had was to be able to learn about and visualize how the average weather each year affects the ratio of car crashes by population. The goals we had were based around the data we intended to collect for a particular year. This data included weather, number of car crashes and car crash fatalities, and total population for a given year in several locations. Our original thoughts were that we would see some type of correlation between weather and car crash data. We also wanted to practice using APIs, Beautiful Soup, Matplotlib, and SQL to further our understanding and skills of these.

Goals Achieved

We achieved most of our goals that we had planned for. We used at least 2 APIs and 1 website and created at least 3 visualizations. We also learned about the relationships between weather, car crashes, and population in certain locations. That being said, we did run into some problems during our project which made us change the scope of the project to best fit what data the APIs and websites provided us with. We ended up changing our weather API and website we used. We used two websites: one that provided us with the population per Illinois county in 2019 and the other was Wikipedia which provided us with all 102 names of the Illinois counties along with their county codes. The car crash API provided us with data per county and because we planned to use at least 100 locations (to fit the project instructions) we decided to hone in on one particular state with 100 counties. By choosing one state, we were able to better find websites and weather information about that particular state and its counties. We used the weather API to gather information about particular snowfall and rainfall in the year 2019. We also had to focus on a specific year because the weather API provided us with information on a two-month basis and so the program was running way too slow for gathering data over a few years. Therefore, this is why we decided to focus on a single year. We chose 2019 because it was the most recent year we could collect data about in each category. For our original plan of comparing the ratio of car crashes by population for each state given the average weather that year we changed this based on our data to compare the number of fatal car crashes in the top 10 populated Illinois counties in 2019. Another goal we had was to be able to learn about and visualize how the average weather each year affects the ratio of car crashes by population. We changed this a bit by learning about and visualizing how the average weather in 2019 affects the number of car crashes in Illinois, comparing county to county.

Problems

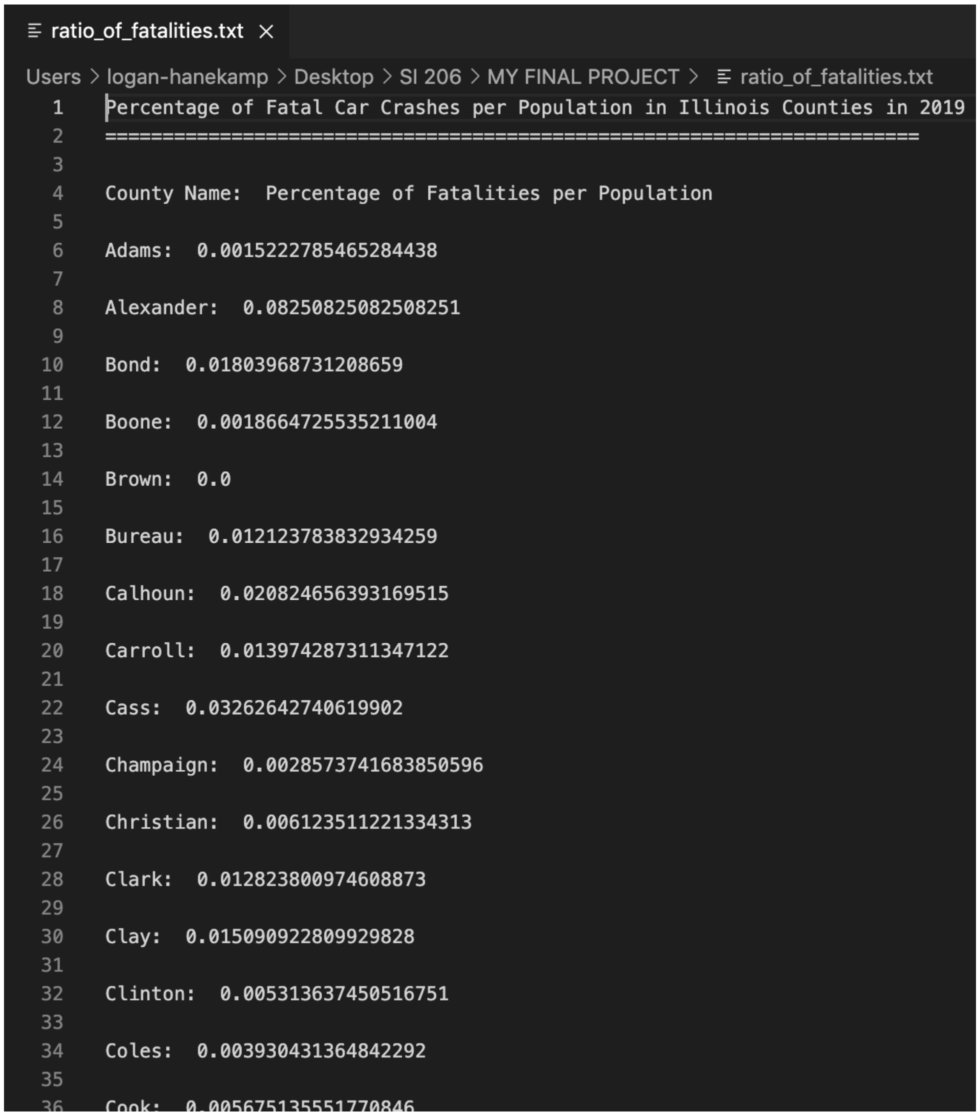
- As we began our project, we soon realized that one of the APIs and the website we had in our plan were not going to work for this project. In terms of the website we had selected, we were unable to use beautiful soup to collect population data as we hoped. The website's population data was somehow embedded within a picture and after researching how to collect this data, we were unsuccessful in achieving this goal. We ended up choosing two separate websites to accomplish this. These two websites were: one that provided us with the population per Illinois county in 2019 and the other was Wikipedia which provided us with all 102 names of the Illinois counties.

- Another problem we faced was with the weather data API. We realized that we would only get the weather reported by day, and we had data on a yearly basis for our two other resources. So, we figured out a way to go through day by day and add the daily values for a yearly value basis. We did so using several functions and calculations with for loops in each so that we could successfully get the values we wanted.

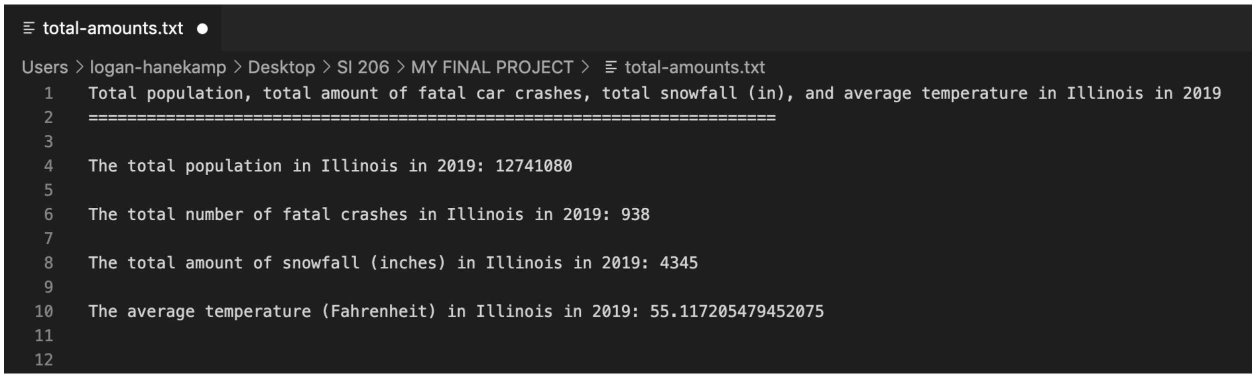
- In addition, we encountered a problem when the amount of time that it was taking to pull all of the data from the API was much longer than our presentation time would allow. We decided that, for the sake of time, we would write the API data (25 requests at a time) into local json files so that we could pull from there for the creation of our databases. Further, since we were pulling such large amounts of data from the API, we ended up exceeding our monthly request limit and had to pay $50 in order to finish the project on time.

- We faced a minor issue when deciding how to work with the location crash data to be able to compare it with the weather data. The weather data from the API we had planned to use only lets you search from city to city, whereas the crash data only lets you gather data by county. While some of the data points in the crash JSON provided the city name in the dictionary, a lot of them just said "not applicable", so we had to stick with just using the county code to search and collect the data. We handled this problem by finding an API that lets you search weather data by county name, and then using a JOIN and a county code table in our database to keep track of both the codes, names, and ids.

Calculations

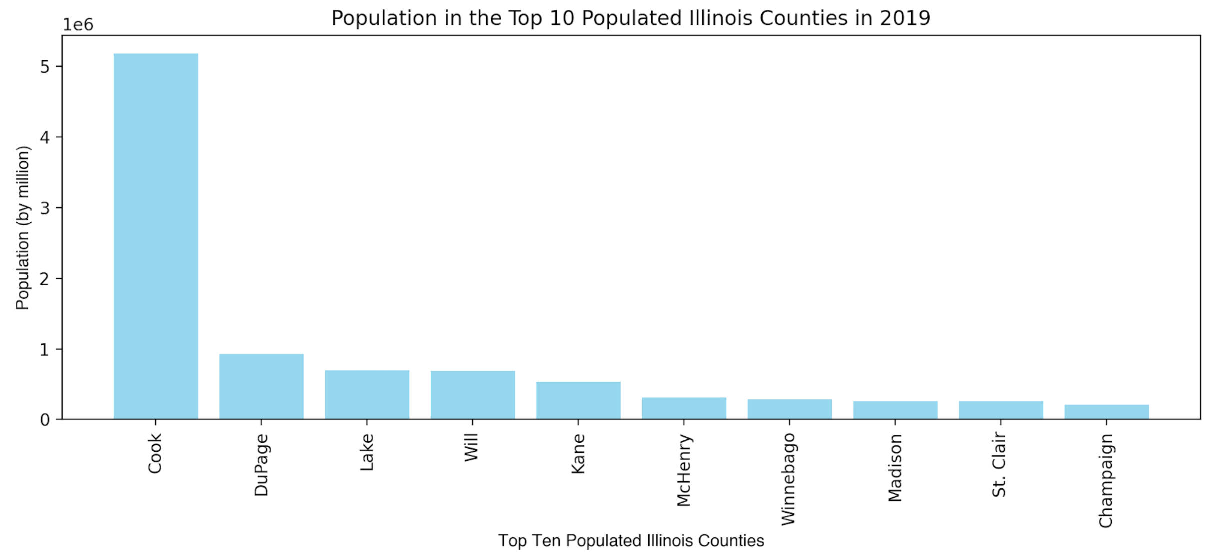

ratio_of_fatalities.txt
Users > logan-hanekamp > Desktop > SI 206 > MY FINAL PROJECT > ratio_of_fatalities.txt

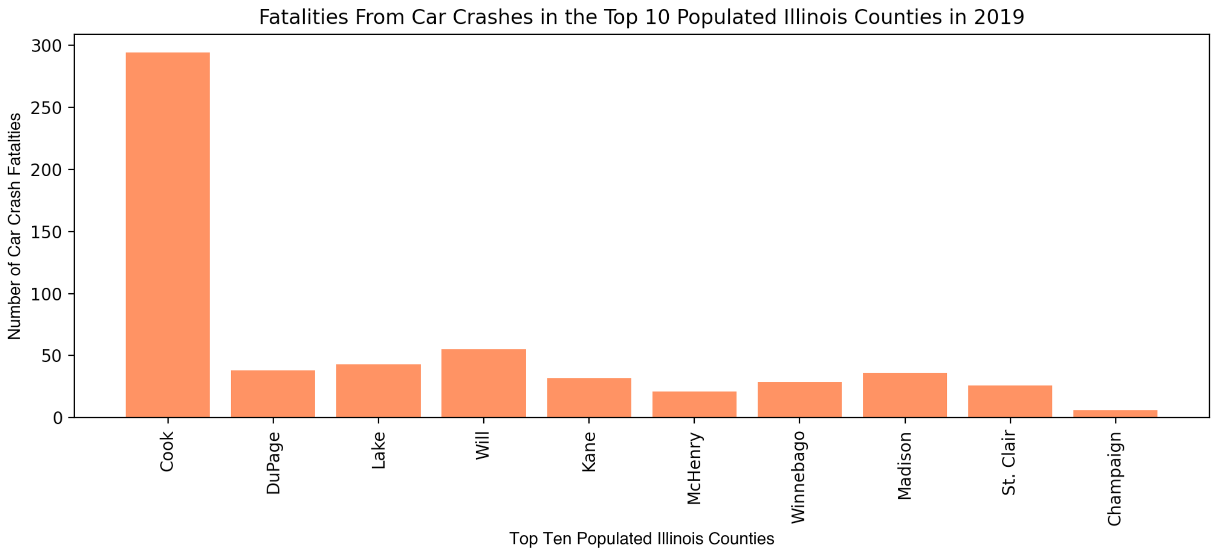
```
Percentage of Fatal Car Crashes per Population in Illinois Counties in 2019
========================================================================

County Name:  Percentage of Fatalities per Population

Adams:  0.0015222785465284438

Alexander:  0.08250825082508251

Bond:  0.01803968731208659

Boone:  0.0018664725535211004

Brown:  0.0

Bureau:  0.012123783832934259

Calhoun:  0.020824656393169515

Carroll:  0.013974287311347122

Cass:  0.03262642740619902

Champaign:  0.0028573741683850596

Christian:  0.006123511221334313

Clark:  0.012823800974608873

Clay:  0.015090922809929828

Clinton:  0.005313637450516751

Coles:  0.003930431364842292

Cook:  0.005675135551770846
```

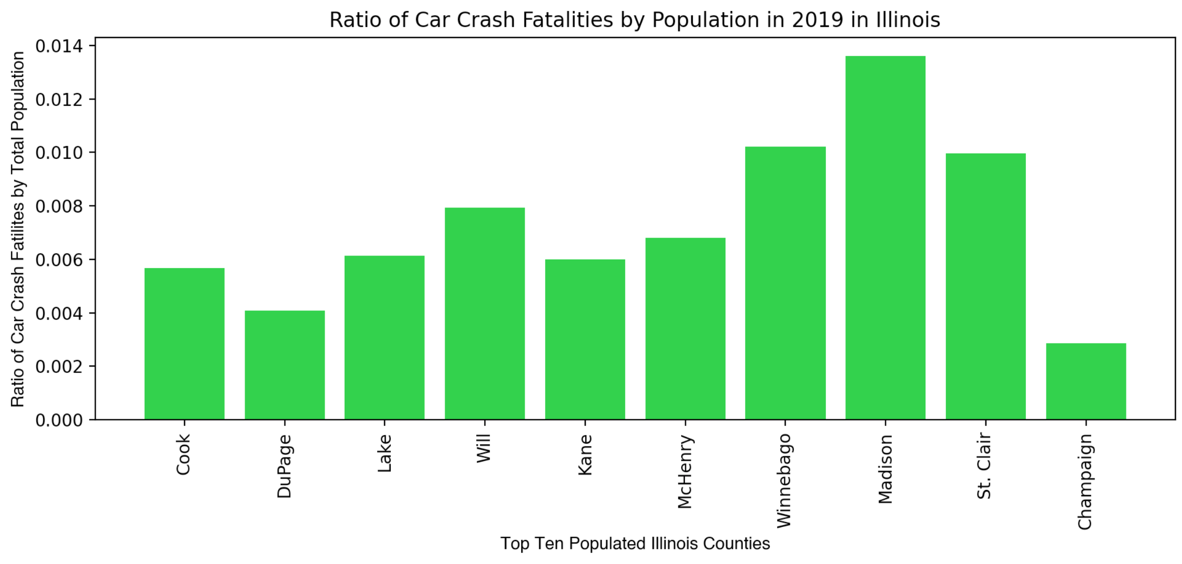

total-amounts.txt
Users > logan-hanekamp > Desktop > SI 206 > MY FINAL PROJECT > total-amounts.txt

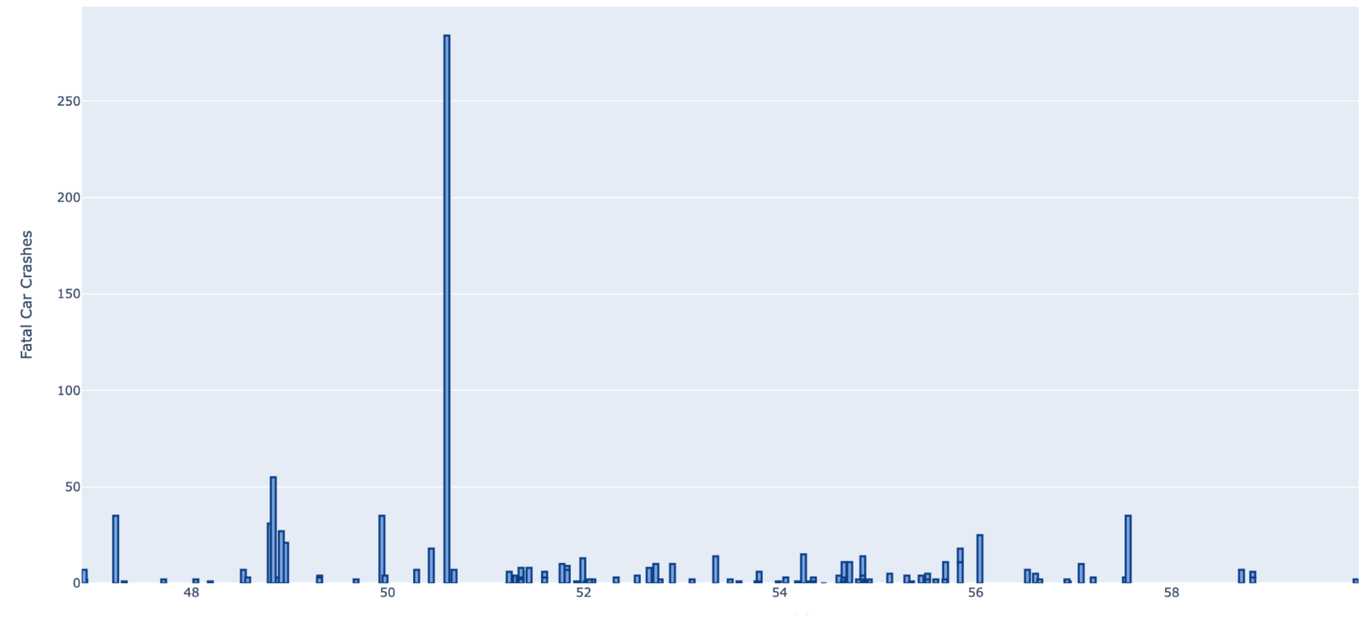
```
Total population, total amount of fatal car crashes, total snowfall (in), and average temperature in Illinois in 2019
========================================================================

The total population in Illinois in 2019: 12741080

The total number of fatal crashes in Illinois in 2019: 938

The total amount of snowfall (inches) in Illinois in 2019: 4345

The average temperature (Fahrenheit) in Illinois in 2019: 55.117205479452075
```

Visualizations

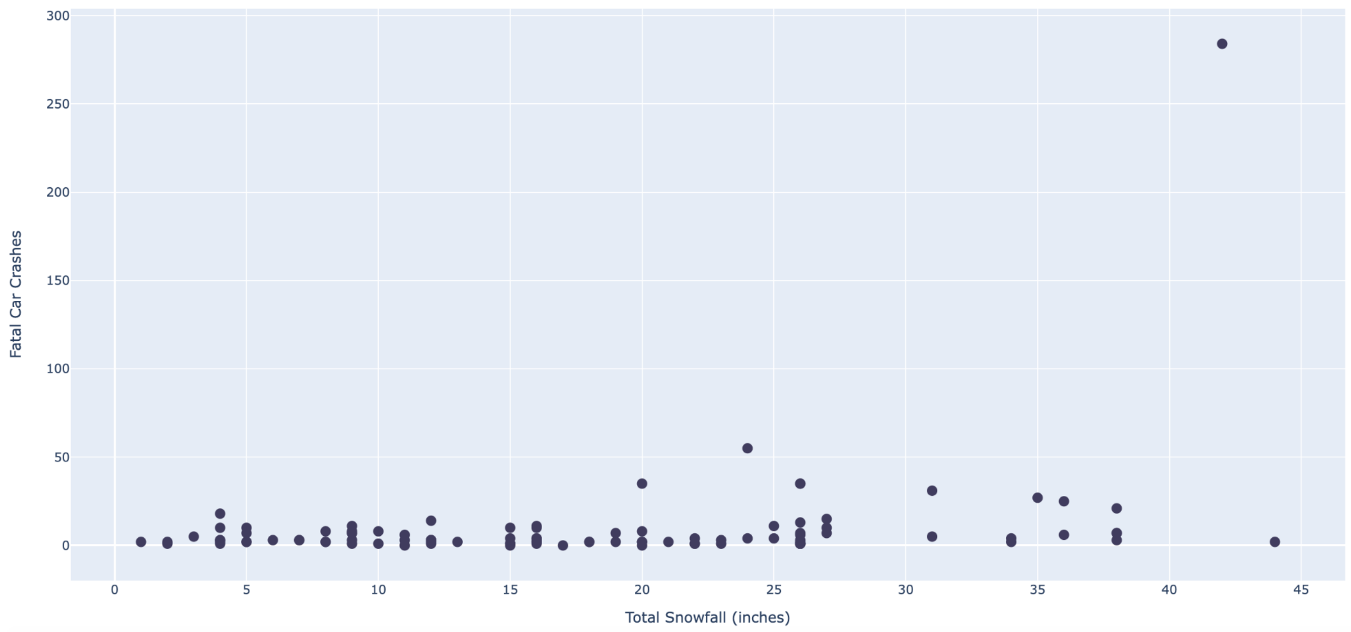





Ratio of Car Crash Fatalities by Population in 2019 in Illinois
Ratio of Car Crash Fatilites by Total Population
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
Cook
DuPage
Lake
Will
Kane
McHenry
Winnebago
Madison
St. Clair
Champaign
Top Ten Populated Illinois Counties



Relationship between Total Snowfall and Car Crash Fatalities in Illinois in 2019



Instruction To Run Code

1. Make sure that the "Weather_Crash_Data_Illinois.db" database does not already exist in your files. If it does, delete so it can be recreated.

2. Run the file "county.py". This should be run five times. After five runs it will add all 102 rows of counties to the database. The database will create a table called "Counties". This fills the table 24 at a time except for the last row because there are only 102 counties.

3. The second file to be run is "crash.py". This should be run five times. After five runs it will add all 102 rows of counties to the database. The database will create a table called "Crashes". This fills the table 24 at a time except for the last row because there are only 102 counties.

4. To produce the Matplotlib visualizations run the file "visualization1.py" once. This will produce 3 graphs.

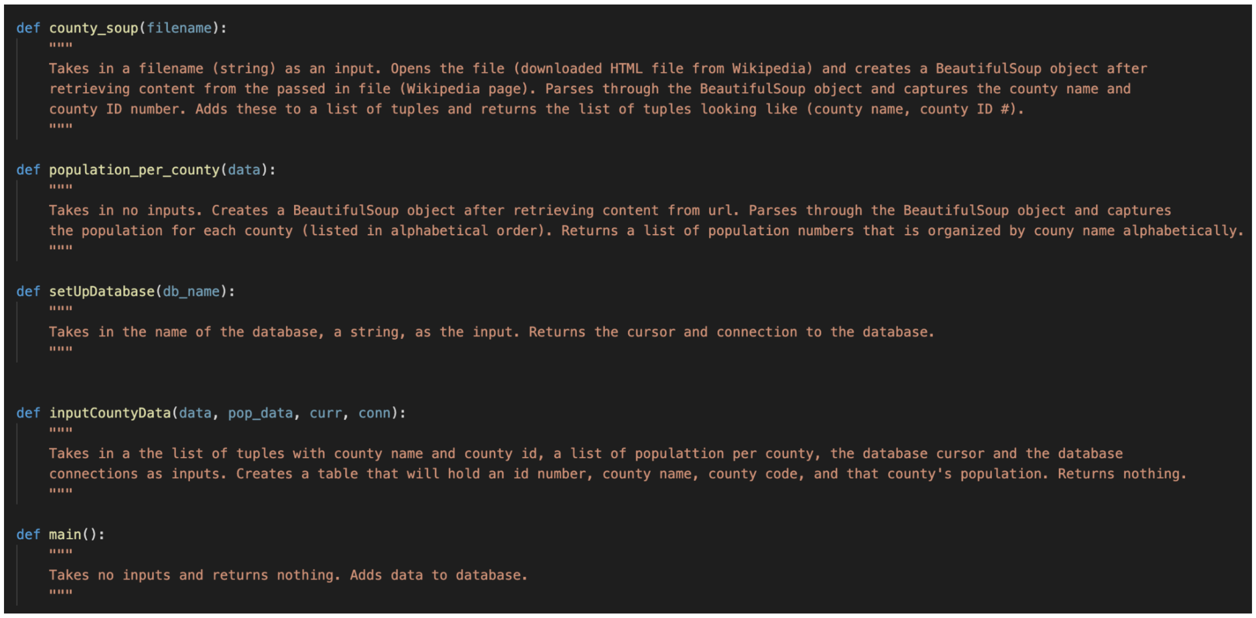
5. To produce the Plotly visualizations run the file "visualization2.py". This will produce 2 graphs. It will also create the two tables from the weather data "Total_Snow" and "Avg_Temp" with 102 county id's and their corresponding total snowfall in inches and avg temperature in fahrenheit for the year 2019 within "Weather_Crash_Data_Illinois.db".

6. Open up "Weather_Crash_Data_Illinois.db" to see the tables. There should be four total tables.

7. To see the calculations, open "ratio_of_fatalities.txt" and "total-amounts.txt".

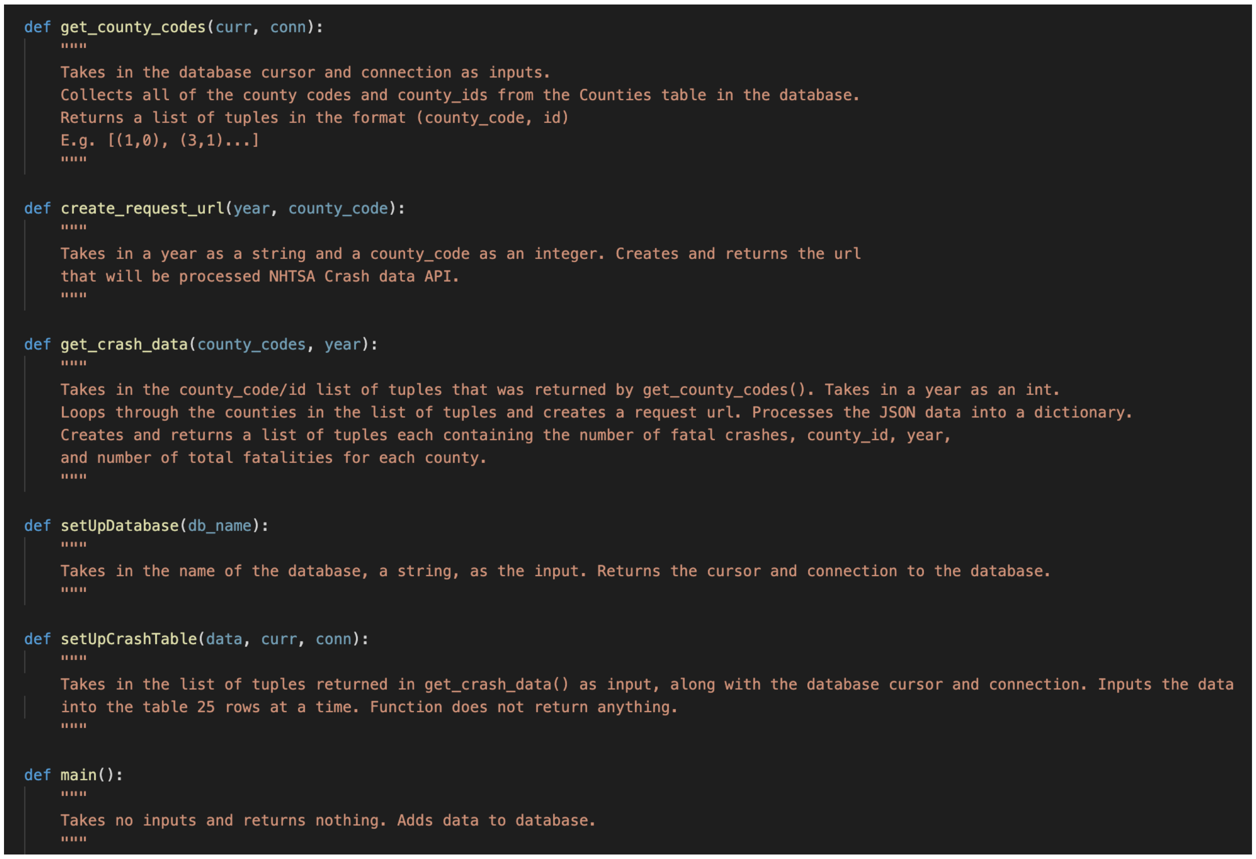
Code Documentation

Documentation for county.py



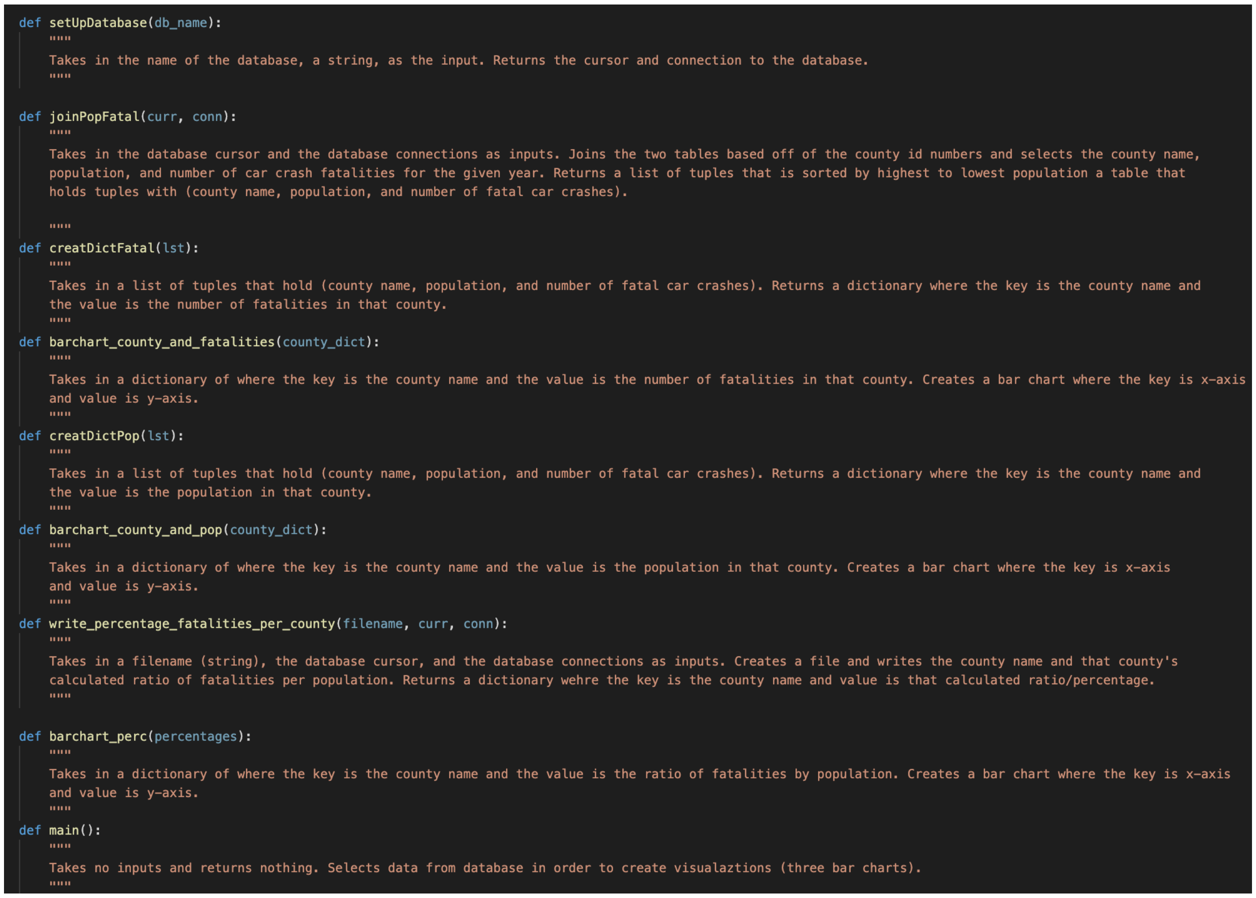
```python
def county_soup(filename):
    """
    Takes in a filename (string) as an input. Opens the file (downloaded HTML file from Wikipedia) and creates a BeautifulSoup object after
    retrieving content from the passed in file (Wikipedia page). Parses through the BeautifulSoup object and captures the county name and
    county ID number. Adds these to a list of tuples and returns the list of tuples looking like (county name, county ID #).
    """

def population_per_county(data):
    """
    Takes in no inputs. Creates a BeautifulSoup object after retrieving content from url. Parses through the BeautifulSoup object and captures
    the population for each county (listed in alphabetical order). Returns a list of population numbers that is organized by couny name alphabetically.
    """

def setUpDatabase(db_name):
    """
    Takes in the name of the database, a string, as the input. Returns the cursor and connection to the database.
    """


def inputCountyData(data, pop_data, curr, conn):
    """
    Takes in a the list of tuples with county name and county id, a list of populattion per county, the database cursor and the database
    connections as inputs. Creates a table that will hold an id number, county name, county code, and that county's population. Returns nothing.
    """

def main():
    """
    Takes no inputs and returns nothing. Adds data to database.
    """
```

Documentation for crash.py



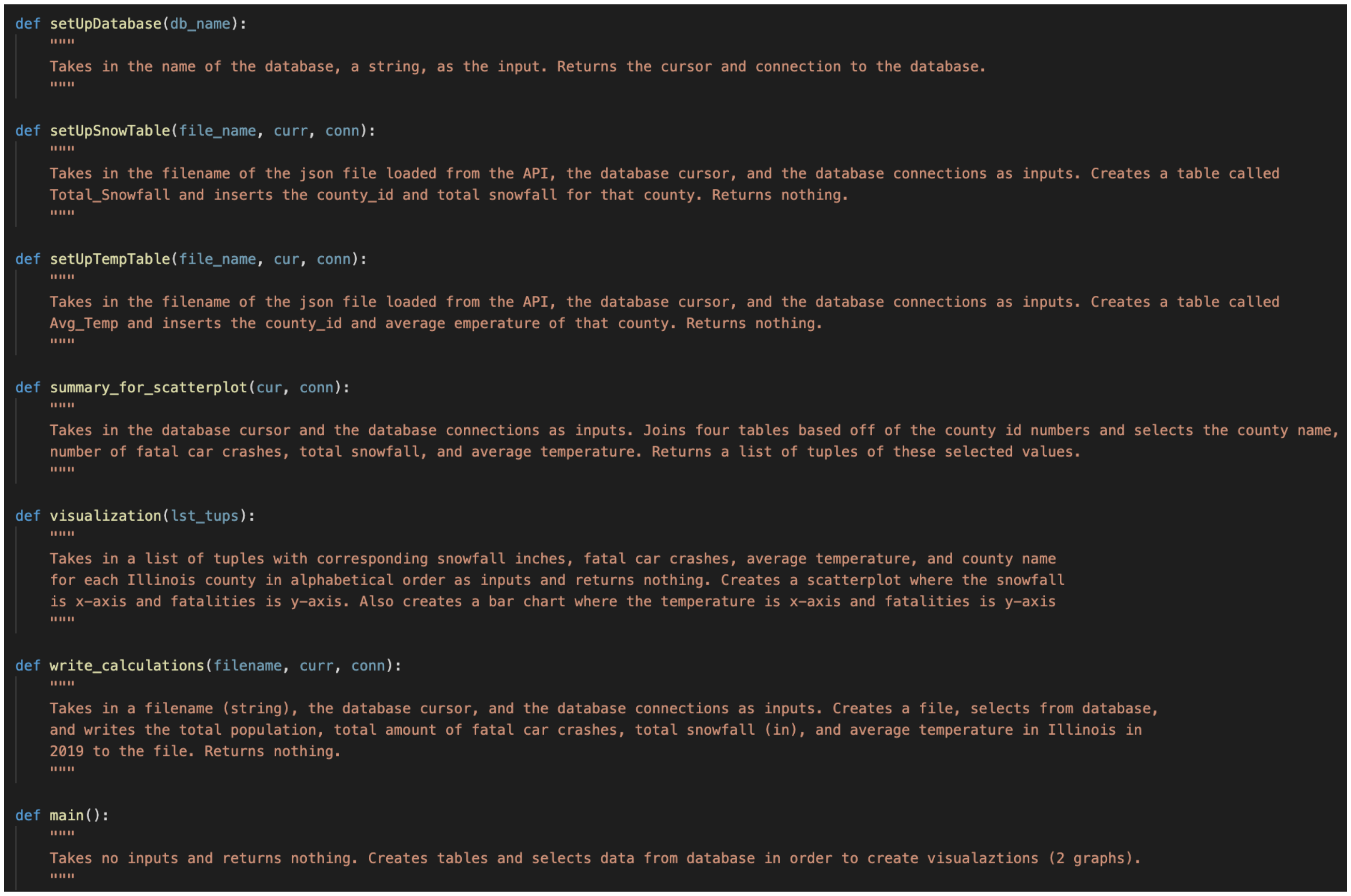
```python
def get_county_codes(curr, conn):
    """
    Takes in the database cursor and connection as inputs.
    Collects all of the county codes and county_ids from the Counties table in the database.
    Returns a list of tuples in the format (county_code, id)
    E.g. [(1,0), (3,1)...]
    """

def create_request_url(year, county_code):
    """
    Takes in a year as a string and a county_code as an integer. Creates and returns the url
    that will be processed NHTSA Crash data API.
    """

def get_crash_data(county_codes, year):
    """
    Takes in the county_code/id list of tuples that was returned by get_county_codes(). Takes in a year as an int.
    Loops through the counties in the list of tuples and creates a request url. Processes the JSON data into a dictionary.
    Creates and returns a list of tuples each containing the number of fatal crashes, county_id, year,
    and number of total fatalities for each county.
    """

def setUpDatabase(db_name):
    """
    Takes in the name of the database, a string, as the input. Returns the cursor and connection to the database.
    """

def setUpCrashTable(data, curr, conn):
    """
    Takes in the list of tuples returned in get_crash_data() as input, along with the database cursor and connection. Inputs the data
    into the table 25 rows at a time. Function does not return anything.
    """

def main():
    """
    Takes no inputs and returns nothing. Adds data to database.
    """
```

Documentation for visualization1.py



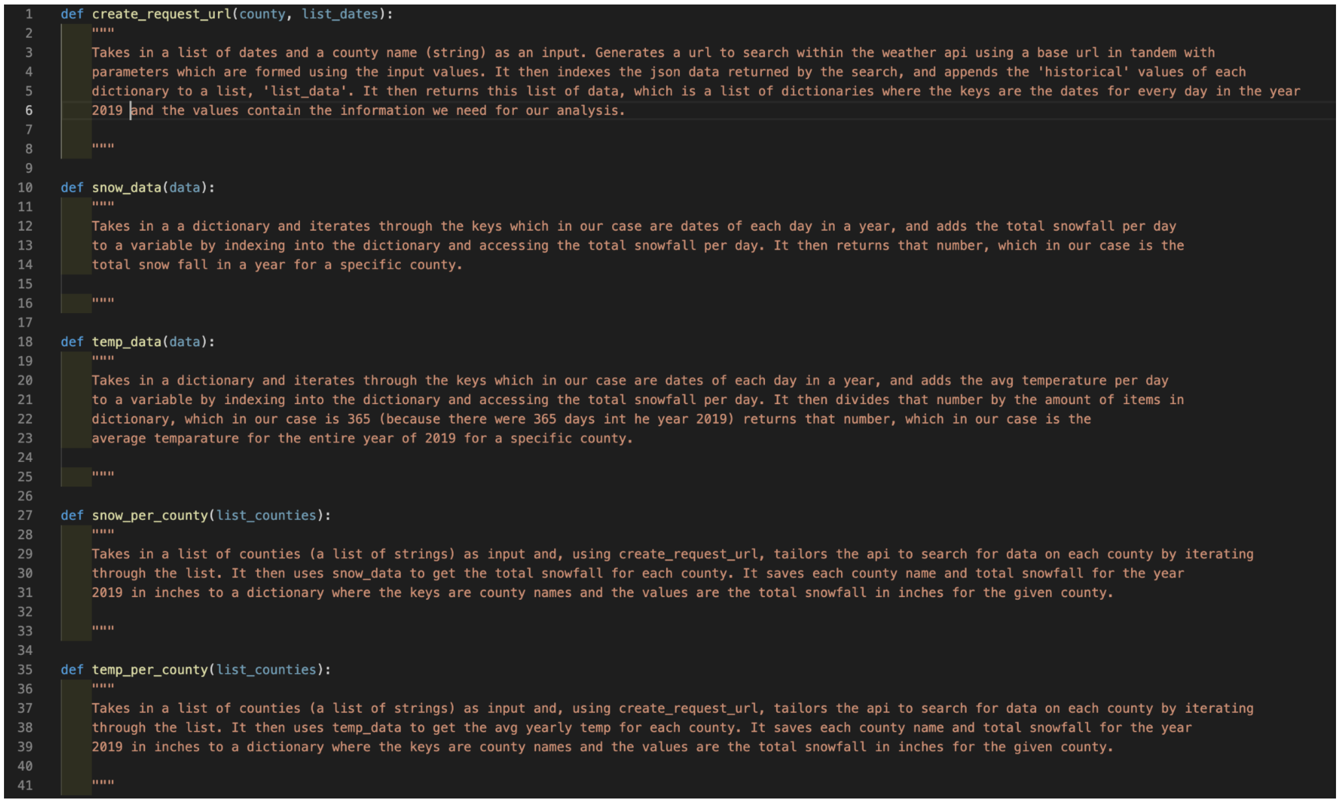
```
def setUpDatabase(db_name):
    """
    Takes in the name of the database, a string, as the input. Returns the cursor and connection to the database.
    """

def joinPopFatal(curr, conn):
    """
    Takes in the database cursor and the database connections as inputs. Joins the two tables based off of the county id numbers and selects the county name,
    population, and number of car crash fatalities for the given year. Returns a list of tuples that is sorted by highest to lowest population a table that
    holds tuples with (county name, population, and number of fatal car crashes).

    """
def creatDictFatal(lst):
    """
    Takes in a list of tuples that hold (county name, population, and number of fatal car crashes). Returns a dictionary where the key is the county name and
    the value is the number of fatalities in that county.
    """
def barchart_county_and_fatalities(county_dict):
    """
    Takes in a dictionary of where the key is the county name and the value is the number of fatalities in that county. Creates a bar chart where the key is x-axis
    and value is y-axis.
    """
def creatDictPop(lst):
    """
    Takes in a list of tuples that hold (county name, population, and number of fatal car crashes). Returns a dictionary where the key is the county name and
    the value is the population in that county.
    """
def barchart_county_and_pop(county_dict):
    """
    Takes in a dictionary of where the key is the county name and the value is the population in that county. Creates a bar chart where the key is x-axis
    and value is y-axis.
    """
def write_percentage_fatalities_per_county(filename, curr, conn):
    """
    Takes in a filename (string), the database cursor, and the database connections as inputs. Creates a file and writes the county name and that county's
    calculated ratio of fatalities per population. Returns a dictionary wehre the key is the county name and value is that calculated ratio/percentage.
    """

def barchart_perc(percentages):
    """
    Takes in a dictionary of where the key is the county name and the value is the ratio of fatalities by population. Creates a bar chart where the key is x-axis
    and value is y-axis.
    """
def main():
    """
    Takes no inputs and returns nothing. Selects data from database in order to create visualaztions (three bar charts).
    """
```

Documentation for visualization2.py

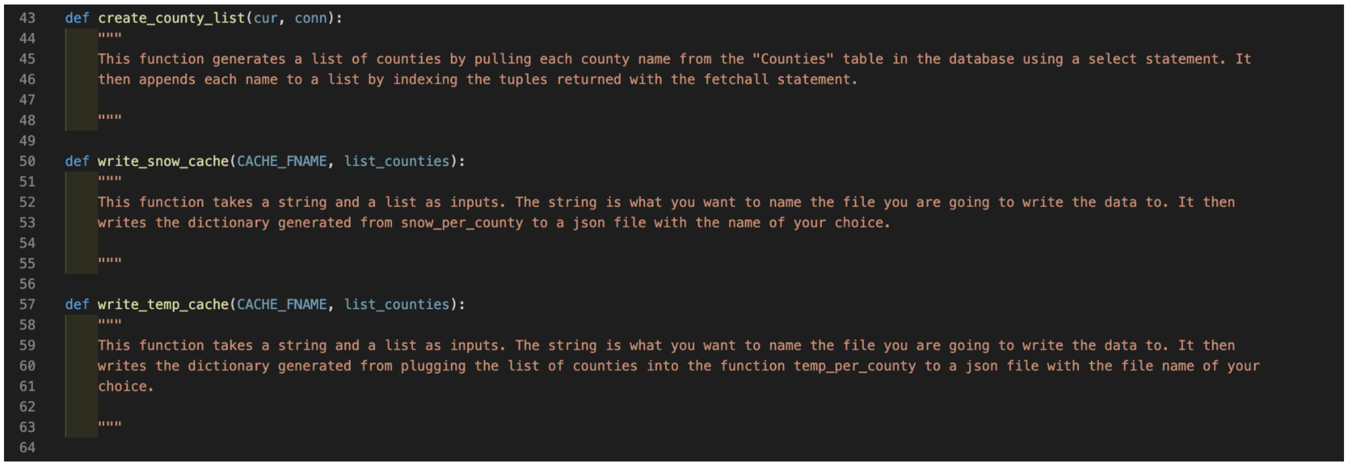


```
def setUpDatabase(db_name):
    """
    Takes in the name of the database, a string, as the input. Returns the cursor and connection to the database.
    """

def setUpSnowTable(file_name, curr, conn):
    """
    Takes in the filename of the json file loaded from the API, the database cursor, and the database connections as inputs. Creates a table called
    Total_Snowfall and inserts the county_id and total snowfall for that county. Returns nothing.
    """

def setUpTempTable(file_name, cur, conn):
    """
    Takes in the filename of the json file loaded from the API, the database cursor, and the database connections as inputs. Creates a table called
    Avg_Temp and inserts the county_id and average emperature of that county. Returns nothing.
    """

def summary_for_scatterplot(cur, conn):
    """
    Takes in the database cursor and the database connections as inputs. Joins four tables based off of the county id numbers and selects the county name,
    number of fatal car crashes, total snowfall, and average temperature. Returns a list of tuples of these selected values.
    """

def visualization(lst_tups):
    """
    Takes in a list of tuples with corresponding snowfall inches, fatal car crashes, average temperature, and county name
    for each Illinois county in alphabetical order as inputs and returns nothing. Creates a scatterplot where the snowfall
    is x-axis and fatalities is y-axis. Also creates a bar chart where the temperature is x-axis and fatalities is y-axis
    """

def write_calculations(filename, curr, conn):
    """
    Takes in a filename (string), the database cursor, and the database connections as inputs. Creates a file, selects from database,
    and writes the total population, total amount of fatal car crashes, total snowfall (in), and average temperature in Illinois in
    2019 to the file. Returns nothing.
    """

def main():
    """
    Takes no inputs and returns nothing. Creates tables and selects data from database in order to create visualaztions (2 graphs).
    """
```

Documentation for weather.py:



```python
def create_request_url(county, list_dates):
    """
    Takes in a list of dates and a county name (string) as an input. Generates a url to search within the weather api using a base url in tandem with
    parameters which are formed using the input values. It then indexes the json data returned by the search, and appends the 'historical' values of each
    dictionary to a list, 'list_data'. It then returns this list of data, which is a list of dictionaries where the keys are the dates for every day in the year
    2019 and the values contain the information we need for our analysis.

    """

def snow_data(data):
    """
    Takes in a a dictionary and iterates through the keys which in our case are dates of each day in a year, and adds the total snowfall per day
    to a variable by indexing into the dictionary and accessing the total snowfall per day. It then returns that number, which in our case is the
    total snow fall in a year for a specific county.

    """

def temp_data(data):
    """
    Takes in a dictionary and iterates through the keys which in our case are dates of each day in a year, and adds the avg temperature per day
    to a variable by indexing into the dictionary and accessing the total snowfall per day. It then divides that number by the amount of items in
    dictionary, which in our case is 365 (because there were 365 days int he year 2019) returns that number, which in our case is the
    average temparature for the entire year of 2019 for a specific county.

    """

def snow_per_county(list_counties):
    """
    Takes in a list of counties (a list of strings) as input and, using create_request_url, tailors the api to search for data on each county by iterating
    through the list. It then uses snow_data to get the total snowfall for each county. It saves each county name and total snowfall for the year
    2019 in inches to a dictionary where the keys are county names and the values are the total snowfall in inches for the given county.

    """

def temp_per_county(list_counties):
    """
    Takes in a list of counties (a list of strings) as input and, using create_request_url, tailors the api to search for data on each county by iterating
    through the list. It then uses temp_data to get the avg yearly temp for each county. It saves each county name and total snowfall for the year
    2019 in inches to a dictionary where the keys are county names and the values are the total snowfall in inches for the given county.

    """
```



```python
def create_county_list(cur, conn):
    """
    This function generates a list of counties by pulling each county name from the "Counties" table in the database using a select statement. It
    then appends each name to a list by indexing the tuples returned with the fetchall statement.

    """

def write_snow_cache(CACHE_FNAME, list_counties):
    """
    This function takes a string and a list as inputs. The string is what you want to name the file you are going to write the data to. It then
    writes the dictionary generated from snow_per_county to a json file with the name of your choice.

    """

def write_temp_cache(CACHE_FNAME, list_counties):
    """
    This function takes a string and a list as inputs. The string is what you want to name the file you are going to write the data to. It then
    writes the dictionary generated from plugging the list of counties into the function temp_per_county to a json file with the file name of your
    choice.

    """
```

Resource Documentation

Date	Issue Description	Location of Resource	Result
11/29	Needed to learn how to parse through a table within a pages HTML using beautiful soup	https://stackoverflow.com/questions/10309550/python-beautifulsoup-iterate-over-table	Used: `pop_table = pop.find('tbody')` `pop_row = pop_table.find_all('tr')` To get the specific row of content that we needed to get each county's population.

12/2	Needed to use an if statement to learn if the population only contained numbers	https://www.w3schools.com/python/ref_string_isnumeric.asp	Provided me with the `isnumeric()` method where I was able to check if a string was all numbers.
12/2	Wanted to find cool colors to make our graphs more interesting and visually appealing	https://matplotlib.org/stable/gallery/color/named_colors.html	This provided us with over 30 colors we could use for our graphs.
12/2	We needed a way to generate a list of dates between a certain time period, taking into account things like different month/day lengths and leap years	https://docs.python.org/3/library/datetime.html	This website provided us with information about the `datetime` module, which could help us create a list of dates that we then used with our weather data collecting API.
12/6	Wanted to learn how to add fonts to our graph titles to make our graphs more interesting and visually appealing	https://stackoverflow.com/questions/21321670/how-to-change-fonts-in-matplotlib-python/40781216	Showed us how to change the font from matplotlib's default fonts.
12/7	Wanted to make a graph so when you hover over the bar or scatter we can see the county name.	https://plotly.com/python/bar-charts/	Showed us how to create this type of feature using plotly for both of our graphs in this part.