SI 206 Final Project

Using the Yelp, Zomato, and OpenWeatherMap APIs

Github repository: https://github.com/sydbruce/SI-206---Final-Project

Presented by The Summer Breakers

Sydney Bruce, Dylan Rabin, William Zhang

4/29/19

Table of Contents

INTRODUCTION

As winter comes to a close and flowers start to bloom for spring, students, faculty and staff are more excited than ever to finish the school year and enjoy their summers. While some people find summer internships or local summer jobs, other take the advantage of the summer break by going on a vacation. Our team, The Summer Breakers, is interested in discovering which U.S. cities are great locations for a potential 2-day vacation. Using API data accessed from Yelp hotels, Zomato restaurants, and OpenWeatherMap forecasts, our team wrote a Python program to access data from the websites, populate databases with calculations, and create visualizations. This report identifies the goals, problems faced, and results of our work, along with the resources used, calculations, visualizations, instructions for running the code, and documentation for functions.

GOALS

We began our project by outlining the goals and measures needed to succeed:

1. Find out which U.S. cities provide the best hotels and restaurants.
    a. Search through the Yelp Hotels and Zomato Restaurants to provide the highest-rated and cheapest hotels and restaurants.
2. Determine which U.S. cities are the most popular.
    a. Utilize the Zomato API to see which cities are the most popular and have the best nightlife index.
3. Discover which cities have the best weather.
    a. Use OpenWeatherMap API to find the average high temperature, average low temperature, precipitation count, and average humidity.

We decided on comparing 10 cities for our project:

1. Atlanta, GA
2. Boston, MA
3. Chicago, IL
4. Detroit, MI
5. Houston, TX
6. Los Angeles, CA
7. New York City, NY
8. Philadelphia, PA

9. San Francisco, CA
10. Seattle, WA

PROBLEMS FACED

Our first problem we faced was deciding what APIs to use and what data to gather from each API. We originally had a different scope for our project, as we were looking into comparing restaurants and nutrition facts using the Nutritionix APIs. However, huge problems arose as we realized we had to retrieve 100 specific restaurant items, making the Nutritionix API program extremely tedious and difficult. The following visualization techniques would have to be manually inputted, which would have not worked.

Another problem we faced during the development process was the sudden change of the OpenWeatherMap API. When we initially ran our code for OpenWeatherMap.py, we had no problem with populating the database with 16 entries per city. However, the documentation and application of the OpenWeatherMap API has changed so that our code only prints one location per city. The changes to the interface were out of our control, and consequently, when the graders run the program, please note that only 1 entry for each city will populate, rather than 16 entries per city. As a result to this complication, and because of our decision to not change the code since it was initially working, we will use the data we previously collected for our Weather visualizations.

In addition, the OpenWeatherMap API retrieved highs and lows for a 5 hour forecast for 2 days. We decided to visualize the average highs and average lows for our 10 cities, but because the highs and lows were identical or nearly the same in the database, the differences in the resulting scatterplot visualization appear to be negligible.

RESULTS

While we ran into many problems and challenges along the way, we ultimately were able to achieve the goals we set for ourselves. Find out which U.S. cities provide the best hotels and restaurants. Our first goal was to search through the Yelp Hotels and Zomato Restaurants to provide the highest-rated and cheapest hotels and restaurants. Deciding to use the Yelp and Zomato APIs were great choices because their documentations are simple and many developers have used the documentation in the past. If we ever ran into an issue, we searched through stack overflow to find a solution for it. Likewise, it was simple to determine which U.S. cities are the most popular for the same reason. For our third goal, the documentation was more difficult, but it was easier to see the output of the data. Determining which destinations were warmer/colder and more humid/less humid was easy to see because the data is clearcut. Overall, with the exception of the minor glitch in the OpenWeatherMap API, we were able to accomplish our goals!

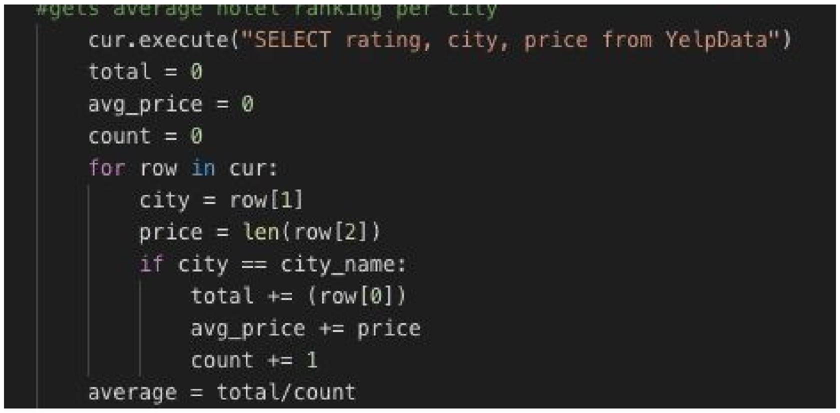
RESOURCES USED

DATE	ISSUE DESCRIPTION	LOCATION OF RESOURCE	RESULTS
4/10/19	We decided on which API's we would use to pull and compare data as we had a problem finding data to compare	API websites(Zomato, Yelp, and OpenWeatherMap)	We were able to extract the data from the API and populate databases with the information gathered
4/15/19	We needed help created visualizations and used the discussion as a template	Discussion Slides (plot_final.py)	We were able to easily gain knowledge on how to properly format our code to create various visualizations
4/19/19 - 4/22/19	• Needed assistance to correctly access the Yelp API • Assistance adding rows to	Stack Overflow	Stack Overflow allowed us to solve all the existing problems and help us move forward with our project.

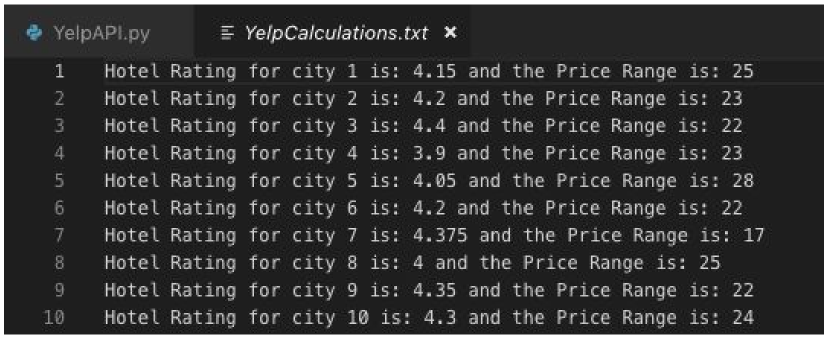
	a database		
4/29/19	Our team realized that we had not yet added all of our tables into a single database and need advice on how to fix it	Piazza	Other groups had similar questions and one question was this exact question so we were able to solve the problem.

CALCULATIONS

YelpAPI



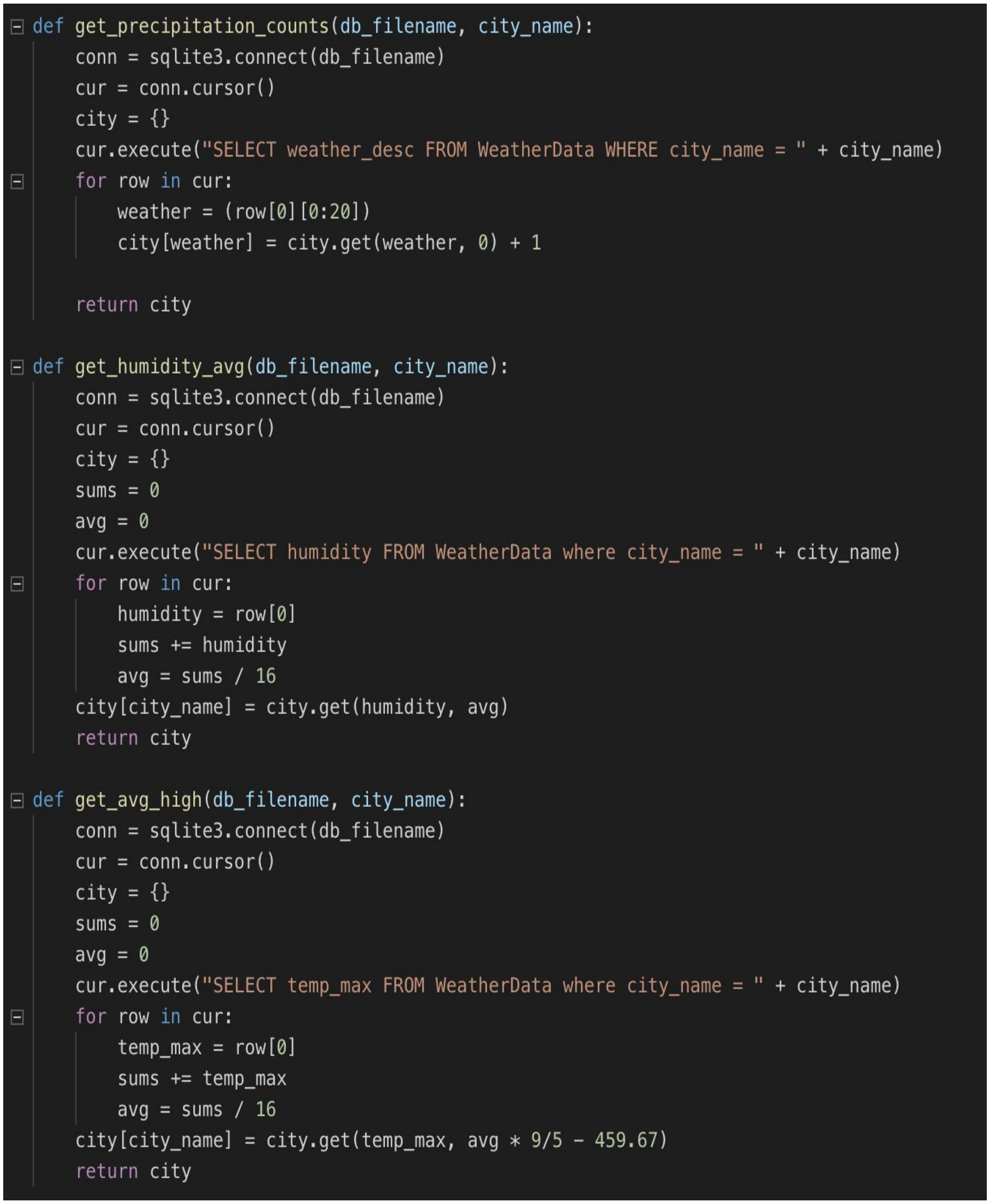
```
#gets average hotel ranking per city
    cur.execute("SELECT rating, city, price from YelpData")
    total = 0
    avg_price = 0
    count = 0
    for row in cur:
        city = row[1]
        price = len(row[2])
        if city == city_name:
            total += (row[0])
            avg_price += price
            count += 1
    average = total/count
```



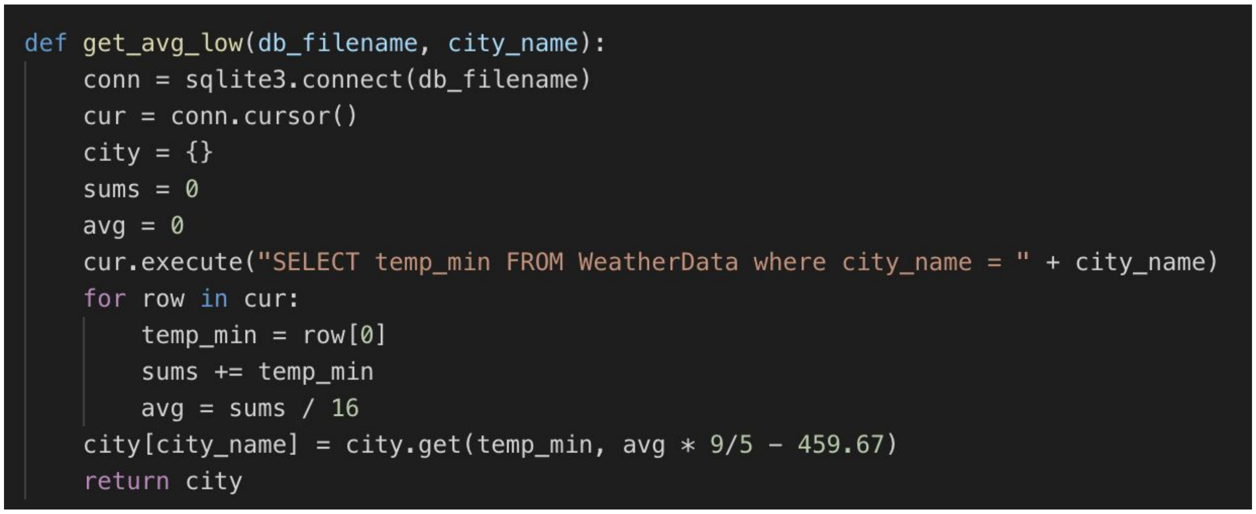
```
Hotel Rating for city 1 is: 4.15 and the Price Range is: 25
Hotel Rating for city 2 is: 4.2 and the Price Range is: 23
Hotel Rating for city 3 is: 4.4 and the Price Range is: 22
Hotel Rating for city 4 is: 3.9 and the Price Range is: 23
Hotel Rating for city 5 is: 4.05 and the Price Range is: 28
Hotel Rating for city 6 is: 4.2 and the Price Range is: 22
Hotel Rating for city 7 is: 4.375 and the Price Range is: 17
Hotel Rating for city 8 is: 4 and the Price Range is: 25
Hotel Rating for city 9 is: 4.35 and the Price Range is: 22
Hotel Rating for city 10 is: 4.3 and the Price Range is: 24
```

I collected the ranking of 10 hotels (out of 5.0) as well as the average price for each. I took all the hotel rankings and averaged them out per city. This allowed me to see the average quality of hotels for that city The price was given in $, so the more $ the more expensive the hotel ($ means less money than $$$). I calculated the len of the price for each city and collected the total number of dollar signs presented to see overall what city was more expensive. This was all taken from the database CombinedDatabase.sqlite. We wrote the information to a text file to make sure we were receiving the right calculations.

OpenWeatherMap API



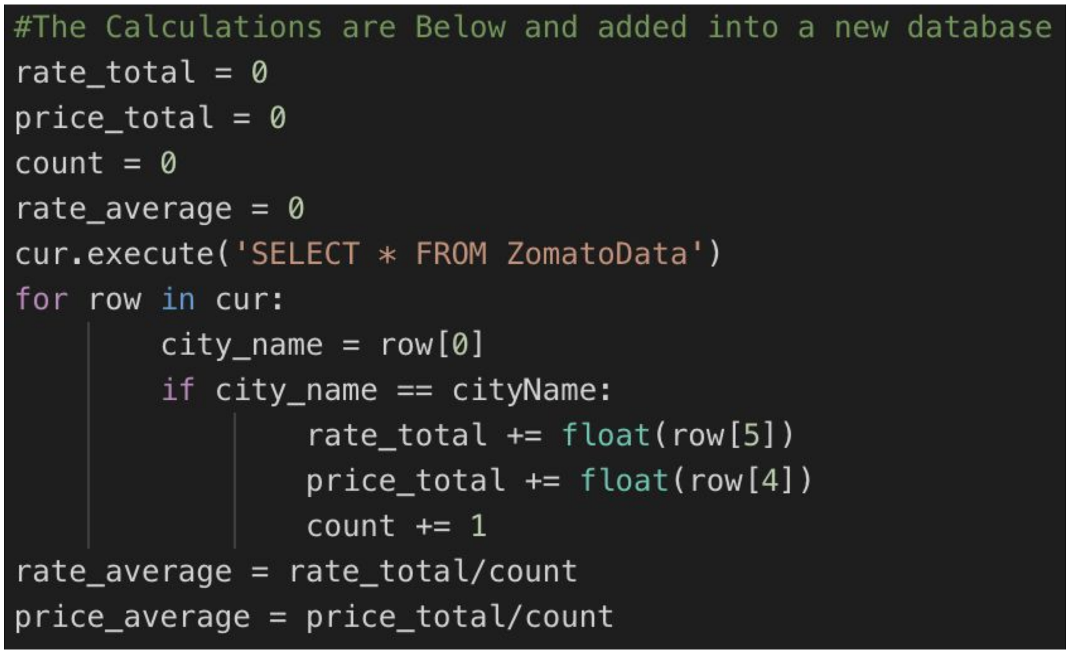
```python
def get_precipitation_counts(db_filename, city_name):
    conn = sqlite3.connect(db_filename)
    cur = conn.cursor()
    city = {}
    cur.execute("SELECT weather_desc FROM WeatherData WHERE city_name = " + city_name)
    for row in cur:
        weather = (row[0][0:20])
        city[weather] = city.get(weather, 0) + 1

    return city

def get_humidity_avg(db_filename, city_name):
    conn = sqlite3.connect(db_filename)
    cur = conn.cursor()
    city = {}
    sums = 0
    avg = 0
    cur.execute("SELECT humidity FROM WeatherData where city_name = " + city_name)
    for row in cur:
        humidity = row[0]
        sums += humidity
        avg = sums / 16
    city[city_name] = city.get(humidity, avg)
    return city

def get_avg_high(db_filename, city_name):
    conn = sqlite3.connect(db_filename)
    cur = conn.cursor()
    city = {}
    sums = 0
    avg = 0
    cur.execute("SELECT temp_max FROM WeatherData where city_name = " + city_name)
    for row in cur:
        temp_max = row[0]
        sums += temp_max
        avg = sums / 16
    city[city_name] = city.get(temp_max, avg * 9/5 - 459.67)
    return city
```



```python
def get_avg_low(db_filename, city_name):
    conn = sqlite3.connect(db_filename)
    cur = conn.cursor()
    city = {}
    sums = 0
    avg = 0
    cur.execute("SELECT temp_min FROM WeatherData where city_name = " + city_name)
    for row in cur:
        temp_min = row[0]
        sums += temp_min
        avg = sums / 16
    city[city_name] = city.get(temp_min, avg * 9/5 - 459.67)
    return city
```

- I collected the weather descriptions and wrote a function to count the type of weather description for the inputted city. I chose not to visualize this data.
- I collected the humidity percentage, temperature max, temperature min, and calculated the average for each.
- All Data was collected from the WeatherData table from CombinedDatabase.sqlite

Zomato API



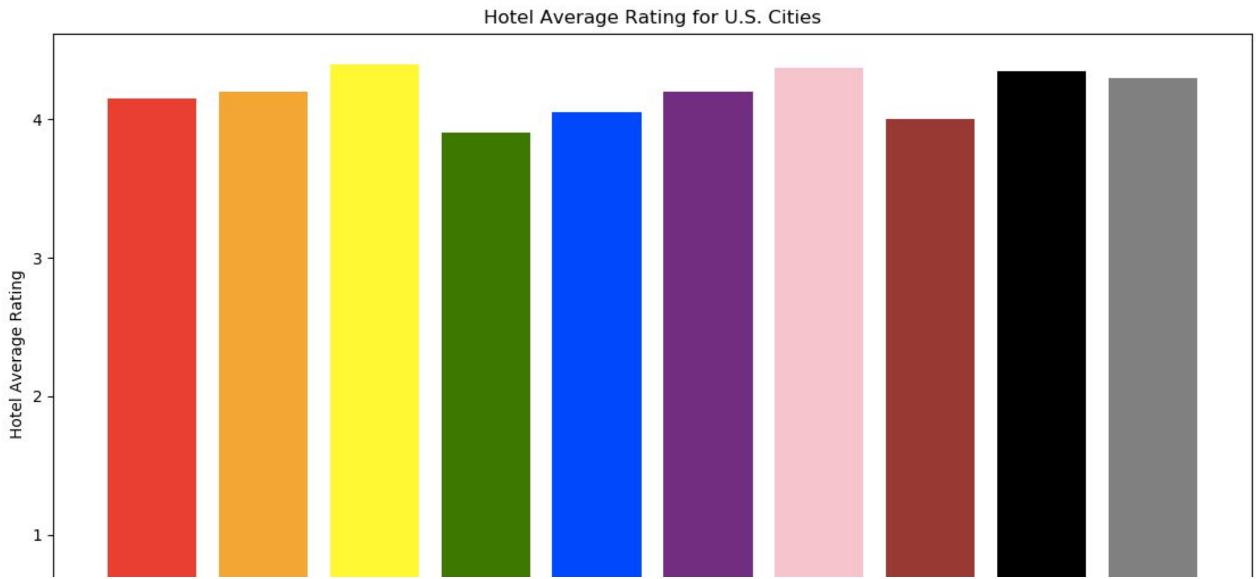
```
#The Calculations are Below and added into a new database
rate_total = 0
price_total = 0
count = 0
rate_average = 0
cur.execute('SELECT * FROM ZomatoData')
for row in cur:
        city_name = row[0]
        if city_name == cityName:
                rate_total += float(row[5])
                price_total += float(row[4])
                count += 1
rate_average = rate_total/count
price_average = price_total/count
```

I collected the ranking of 10 restaurants (out of 5.0) as well as the average price range for each (out of 4.0). I took all the restaurant ratings and price ranges and averaged them out per city. This was all taken from the database CombinedDatabase.sqlite. We wrote the information to a text file to make sure we were receiving the right calculations.

VISUALIZATIONS

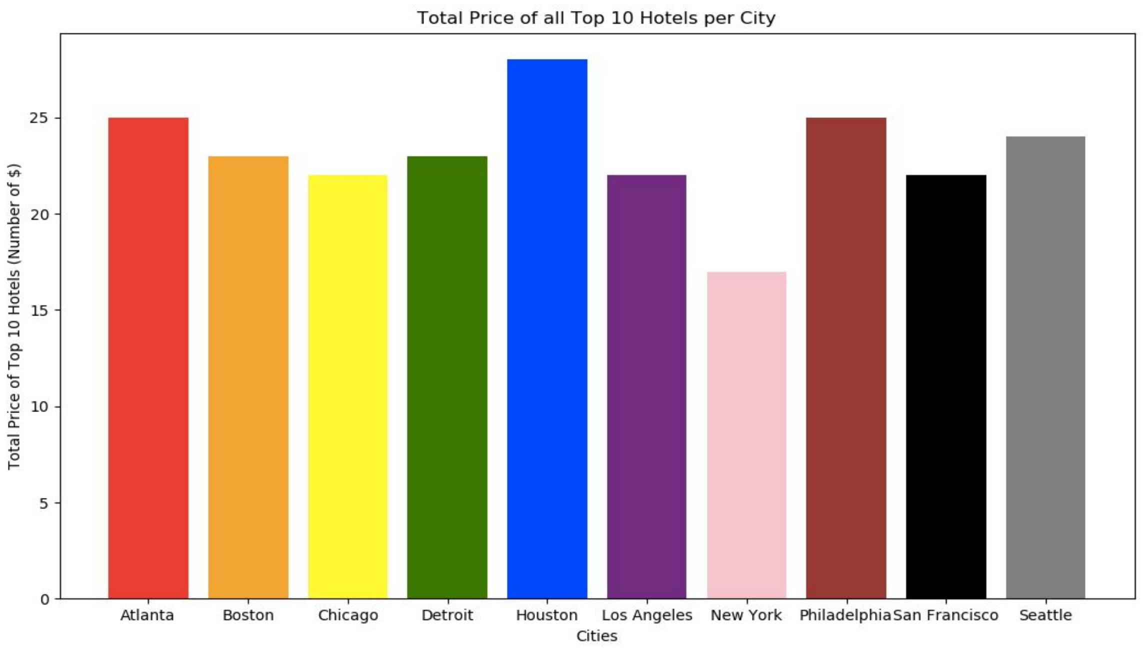
Yelp API

1. **Average Hotel Rating for U.S. Cities**



a. This bar chart compares the average overall hotel ratings per city. The higher the bar, the higher the overall hotel rating for that city. Looking at the bar chart we can see that the highest average hotel rankings is **Chicago and New York.** The lowest is **Detroit**.

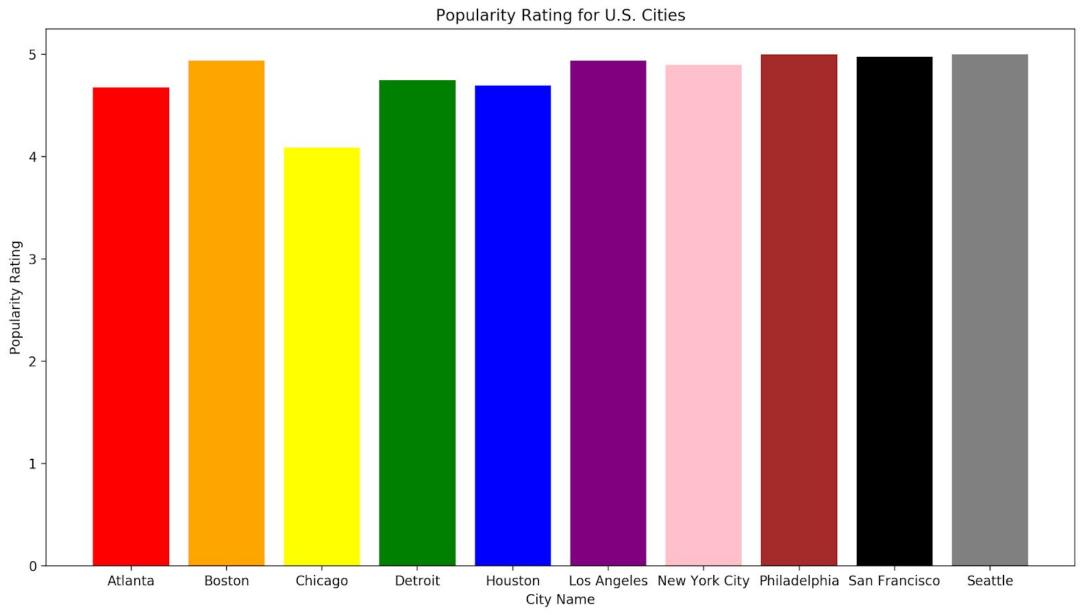
2. Total Price for all Top 10 Hotels per City



a. The API provided the price per hotel in number of $. So the more the dollars signs the more expensive the hotel. I calculated the total number of dollar signs per city for all top 10 hotels. This bar charts takes the total number of calculated dollar signs and compares the city with the most expensive hotels. The most expensive cities for hotels is **Houston and Atlanta**.

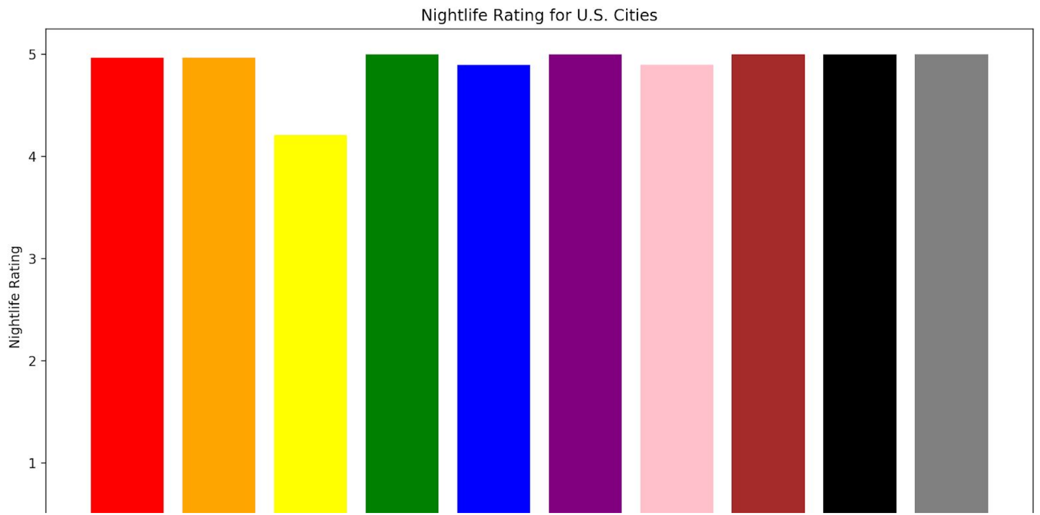
ZomatoAPI

1. Popularity Rating for U.S. Cities



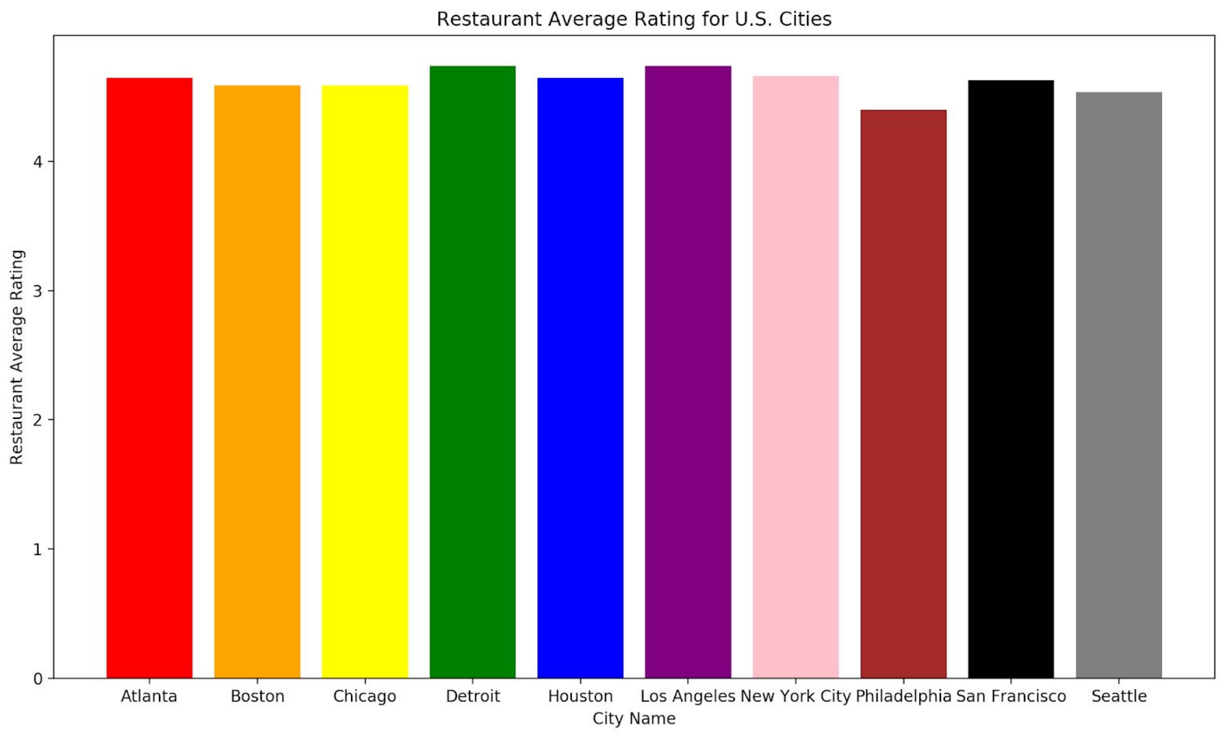
This bar chart compares the popularity index of the ten cities. The higher the bar, the more popular the city is. In this chart, we can see that **Philadelphia and Seattle** are the two most popular cities (5/5).

2. Nightlife Index of U.S. Cities



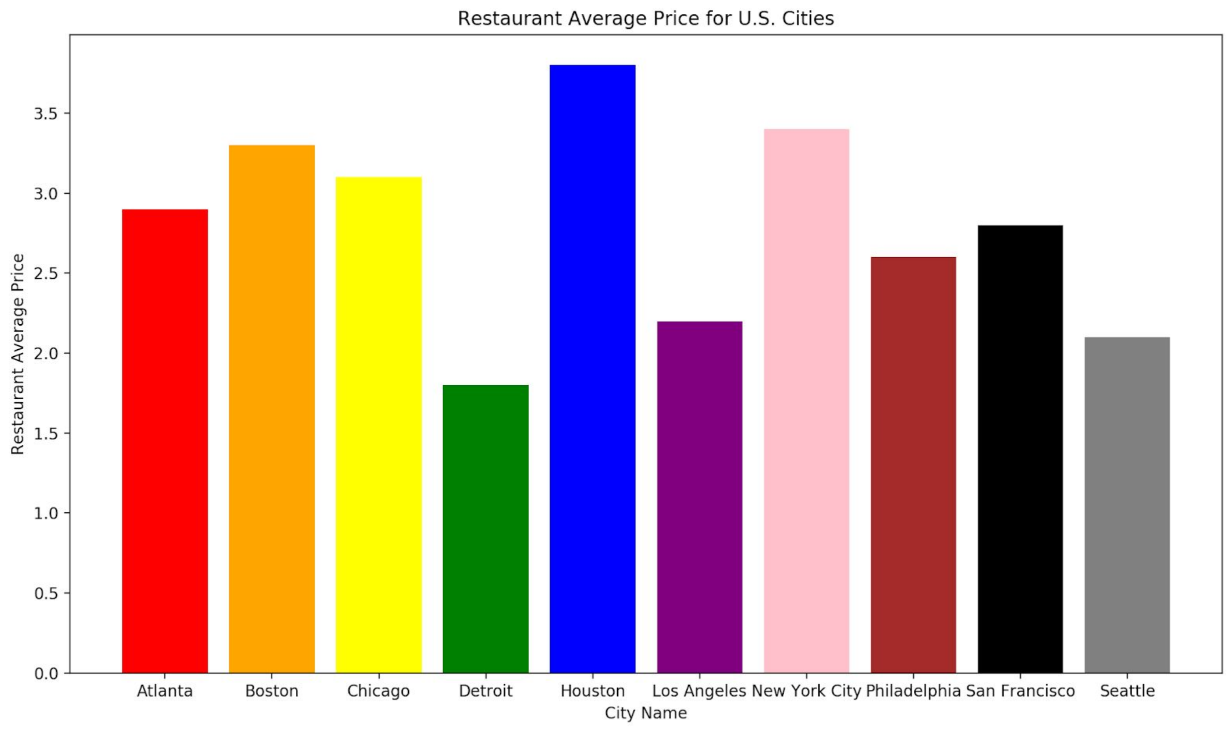
This bar chart compares the Nightlife index of the ten cities. The higher the index, the more popular the nightlife is. In this chart, we can see that **Detroit, Los Angeles, Philadelphia, San Francisco, and Seattle** have the most popular nightlife (5/5).

3. Average Restaurant Rating of U.S. Cities



This bar chart compares the average Restaurant Rating of the ten best restaurants per city for the ten cities. The higher the bar, the higher the average rating is for the ten best restaurants in that city. In this chart, we can see that **Los Angeles** has the highest average restaurant rating (4.65/5) and **Detroit** is a close second (4.64/5).

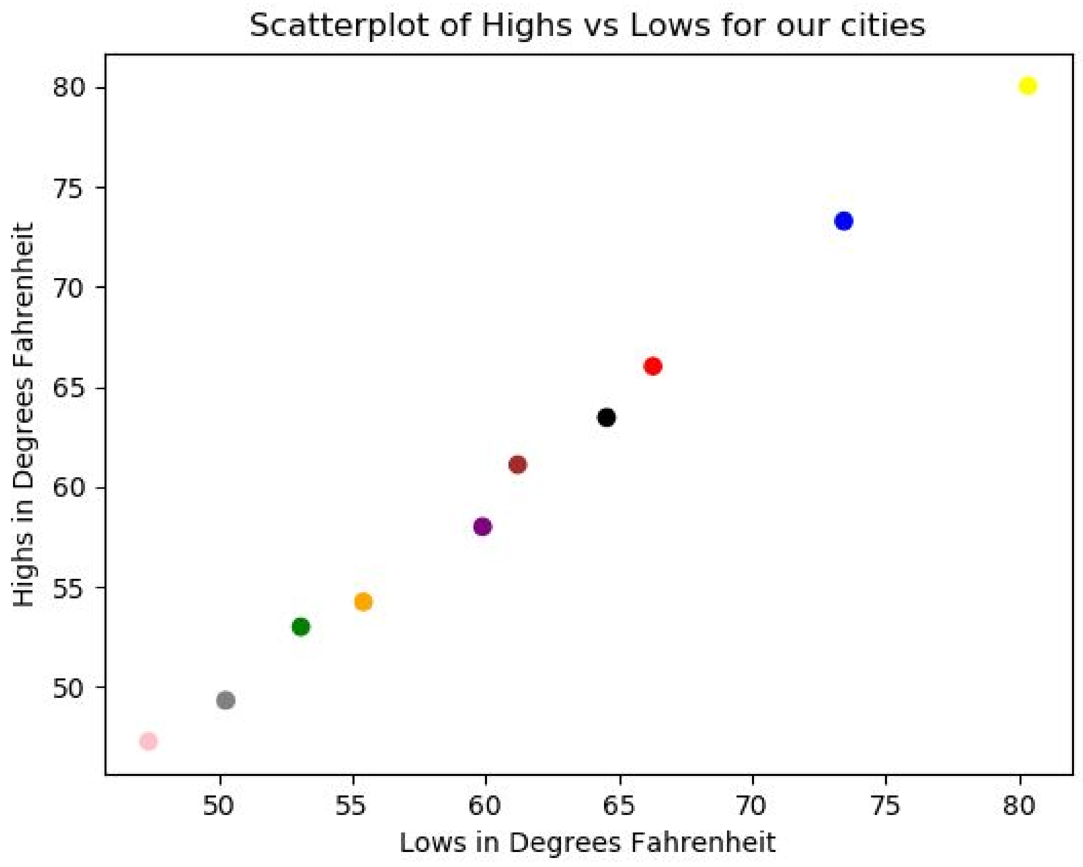
4. Average Restaurant Price Range of U.S. Cities



This bar chart compares the average Restaurant Price Range of the ten best restaurants per city for the ten cities. The higher the bar, the higher the average price range is for the ten best restaurants in that city. In this chart, we can see that **Houston** has the highest (most expensive) average restaurant price range (3.8/4) and **Detroit** has the lowest (cheapest) average restaurant price range (1.8/4).

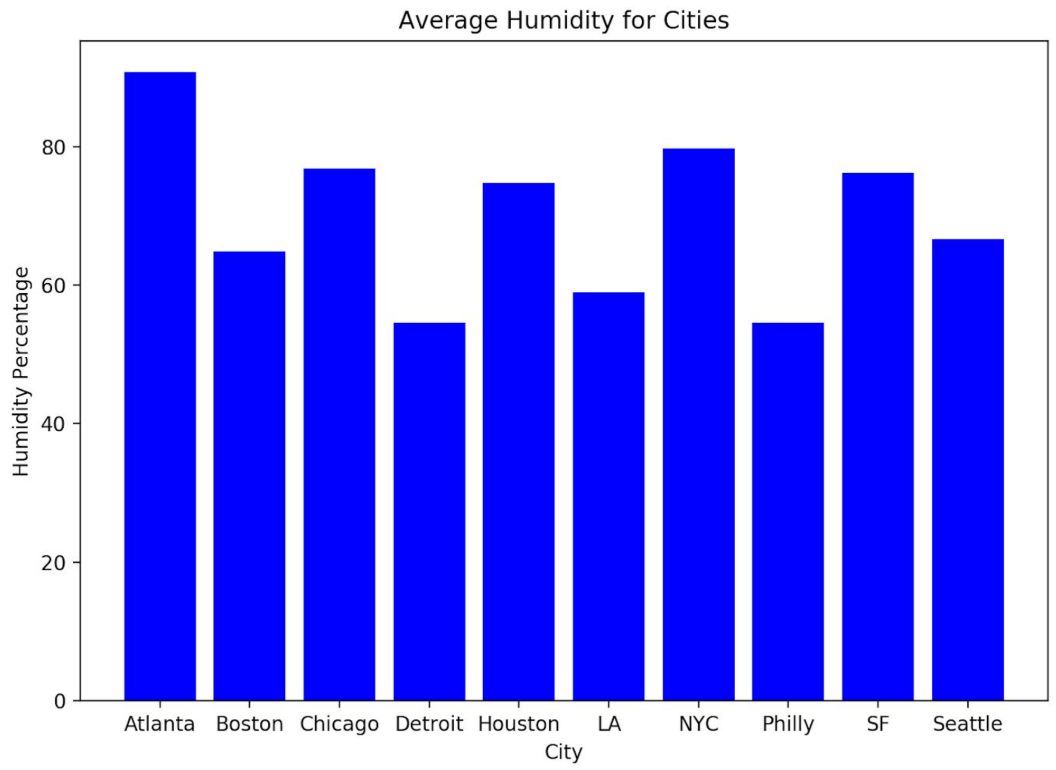
OpenWeatherMap.api

1. **Scatterplot of Temp Highs vs Temp Lows for our cities**
    a. This scatterplot shows the high temps against low temps for the cities of interest. The scatterplot shows that the highs and lows are very very close to each other. The hottest city for April 24th and 25th is Chicago. The coldest city is New York City. The colors correspond with previous barcharts.



2. Scatterplot of Temp Highs vs Temp Lows for our cities

a. This barchart shows the average humidities for our cities over April 24th and 25th. The most humid city is Atlanta, and the least humid cities are Philadelphia and Detroit.



INSTRUCTIONS FOR RUNNING CODE

1. **Run YelpAPI.py**
    a. This will run the main YelpAPI to retrieve information into the database (CombinedDatabase.sqlite). Running this will populate the database will 100 pieces of information, or 10 items per city. The code also conducts the calculations and puts them into a new database (YelpCalc.sqlite), populates it with the 10 city averages, and then takes this new database and creates visualizations off of it.

2. **Run ZomatoAPI.py**
    a. This will run the main ZomatoAPI to retrieve information into the database (CombinedDatabase.sqlite). Running this will populate the database will 100 pieces of information, or 10 items per city. The code also conducts the calculations and puts them into a new database (ZomatoCalc.sqlite), populates it with the 10 city averages for restaurant rating and price range, and then takes this new database and creates visualizations off of it.
3. **Run OpenWeatherMapAPI.py**
    a. This will run the information retrieval process, which will retrieve data from the OpenWeatherMap API, and then store it in a database. It will populate the database with 160 pieces of information, or 16 items per city.
4. **Run WeatherCalculations.py**
    a. This will create calculations for the visualization process. It will create a dictionary for the inputted city with keys as a weather description, and values as the counts of the weather description. It will also create 3 dictionaries for the inputted city with the key as the specified city and value as the average high temperature, low temperature, and humidity respectively.

DOCUMENTATION FOR FUNCTIONS

YelpAPI

a. **def GetYelpFuntion**
    i. Input: YelpAPIKey and search_city
        1. YelpAPIKey is in Keys.py
        2. Search_city was inputted in the running of the function, we chose to input our 10 selected cities
    ii. Output: json.loads
        1. Created an output of all the data pulled from each city inputted into search_city (the 10 we selected)

b. **def setupYelpDataBase**

i. Input: YelpList and city_name
    1. YelpList is a list of all the data pulled from the YelpAPI
    2. City_name is the 10 preselected cities that we imputed to find the data
ii. Output: CombinedDatabase.sqlite, YelpCalc.sqlite
    1. CombinedDatabase.sqlite was a database full of 10 hotels from each of the selected 10 cities. We found the hotel ID, name of hotel, total number of reviews, rating and price per hotel, location, and city the hotel is in
    2. YelpCalc.sqlite was a database full of the calculated average hotel ranking per city and the total price of all the hotels (calculated in number of $)

c. **def createYELPVisualizations**

i. Input: YelpCalc.sqlite
    1. Takes in the database that contains my calculations of average hotel ranking for each city and the total price of all the hotels per city
ii. Output: cityHotelRates.png and TotalPrice.png
    1. cityHotelRates.png is a bar chart comparing the average hotel rankings per city
    2. TotalPrice.png is a bar chart comparing the total overall price for hotels in that city

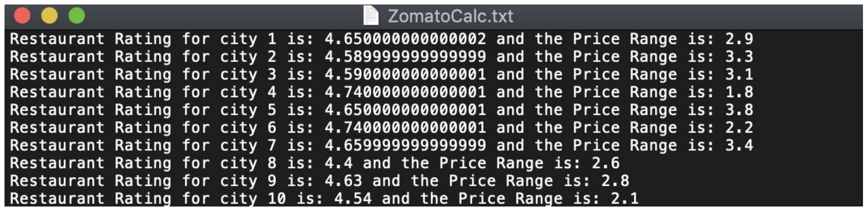
ZomatoAPI

d. **def getZomato**

i. Input: zomato_key, location
    1. Zomato_key is the API key, location is the name of the city. This function in the getLocationDetails function, as this function serves to help digest the documentation.
ii. Output: r
    1. The response returned is the data retrieved from the

API in json format. This data will then be processed again in the getLocationDetails function.

e. **def getLocationDetails**
   i. Input: zomato_key, key_input
      1. Zomato_key is the API key, key_input is the name of the city. This function is called at the bottom of the python file.
   ii. Output: req
      1. The response returned is the data retrieved from the API in json format. This data will then be processed in setupZomatoDataBase function.

f. **def getZomatoDataBase**
   i. Input: data, cityName
      1. Data is the output returned from the getLocationDetails function, and cityName is the name of the city.
   ii. Output: CombinedDatabase.sqlite, ZomatoCalc.sqlite, ZomatoCalc.txt
      1. CombinedDatabase.sqlite has the data from 10 cities, with each city having 10 rows for the top ten restaurants. ZomatoCalc.sqlite and ZomatoCalc.txt both have the results of the calculations in the function.
      2. 

```
Restaurant Rating for city 1 is: 4.650000000000002 and the Price Range is: 2.9
Restaurant Rating for city 2 is: 4.589999999999999 and the Price Range is: 3.3
Restaurant Rating for city 3 is: 4.590000000000001 and the Price Range is: 3.1
Restaurant Rating for city 4 is: 4.740000000000001 and the Price Range is: 1.8
Restaurant Rating for city 5 is: 4.650000000000001 and the Price Range is: 3.8
Restaurant Rating for city 6 is: 4.740000000000001 and the Price Range is: 2.2
Restaurant Rating for city 7 is: 4.659999999999999 and the Price Range is: 3.4
Restaurant Rating for city 8 is: 4.4 and the Price Range is: 2.6
Restaurant Rating for city 9 is: 4.63 and the Price Range is: 2.8
Restaurant Rating for city 10 is: 4.54 and the Price Range is: 2.1
```

g. **def createVisualizations**
   i. Input: CombinedData.sqlite
   ii. Output: The four visualizations
      1. The four visualizations created can be found in the Visualizations section of the report.

OpenWeatherMap API

h. **def getWeather**
    i. Input: CityID
        1. City ID is the specific unique ID for each city, which can be found in city.list.json, which was downloaded off of the OpenWeatherMap website.
    ii. Output: response
        1. The response returned is the data retrieved from the API in json format. This data will then be processed in the next function

i. **def getData**
    i. Input: response
        1. Response is the data retrieved using the function getWeather, it is processed by retrieving the timestamp, temp high, temp low, humidity, and weather description and putting it into the database
    ii. Output: WeatherData.sqlite
        1. Returned is a fully populated database of 160 entries, 16 for each city over the course of 2 days with parameters listed above

WeatherCalculations.py

j. **def getPrecipitationCounts (Did not visualize)**
    i. Inputs: db_filename, city_name
        1. Db_filename used is CombinedDatabase.sqlite
        2. City_name is the specific city specified which will match up with the cities column in the database
    ii. Output: city
        1. City is a dictionary with that contains weather descriptions and the counts for those descriptions for the specified city input

k. **def get_humidity_avg**
    i. Inputs: db_filename, city_name
        1. Db_filename used is CombinedDatabase.sqlite
        2. City_name is the specific city specified which will match up with the cities column in the database
    ii. Output: city
        1. City is a dictionary that contains a single key for the specified city input and a value for the average humidity percentage

l. **Def get_avg_high**
    i. Inputs: db_filename, city_name
        1. Db_filename used is CombinedDatabase.sqlite
        2. City_name is the specific city specified which will match up with the cities column in the database
    ii. Output: city
        1. City is a dictionary that contains a single key for the specified city input and a value for the average high temperature, which was converted from Kelvin to Fahrenheit

m. **def get_avg_low**
    i. Inputs: db_filename, city_name
        1. Db_filename used is CombinedDatabase.sqlite
        2. City_name is the specific city specified which will match up with the cities column in the database
    ii. Output: city
        1. City is a dictionary that contains a single key for the specified city input and a value for the average low temperature, which was converted from Kelvin to Fahrenheit

n. **def visualize**
    i. Inputs: 3 separate lists that contain average high temperatures, average low temperatures, and humidity percentage respectively
        1. Note that the previous functions were used to fill up the

lists that will then be used for visualization

ii. Output: Visualizations
    1. A scatterplot of average highs vs average lows and a barchart for humidity percentage are generated and displayed