CSE 511: Data Processing at Scale – Fall 2023

A Technical Report on Project 1: Individual result - NoSQL

Sai Likhith Yadav Seera
Computer Science
School of Computing and Augmented Intelligence
Arizona State University
Tempe, United States
sseera@asu.edu

I. Reflection

The main agenda of this project is to implement two functions related to Spatial Data in python.

The first function which is named as **FindBusinessBasedOnCity**, searches from a provided 'collection' to find all the businesses present in a specified city attribute called 'cityToSearch'. It retrieves businesses matching the city and saves their name, full address, city, and state details in 'saveLocation1'. Each line in the saved file follows the format: 'Name$FullAddress$City$State', with '$' as the separator between fields. The second function named as **FindBusinessBasedOnLocation**, scans the provided 'collection' to locate businesses from a particular category 'categoriesToSearch' within a specified distance 'maxDistance' from the given 'myLocation' coordinates. This function uses a distance algorithm to calculate the closest distance and stores the names of these businesses in the file location 'saveLocation2', each on a separate line within the output file.

For this project, a NoSQL database named sample.db, is considered along with UnQLite (an embedded NoSQL database engine) for retrieving data and processing them to these two functions.

A. Distance Algorithm

In this project, for calculating the closest distance, a function named **calculateDistance** has been written which uses Haversine distance formula.

The Haversine formula is a mathematical equation used to calculate the distance between two points on the Earth's surface given their latitude and longitude coordinates. It assumes the Earth is a perfect sphere, providing an approximate distance between points [1].

The formula is:

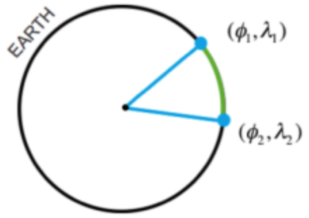
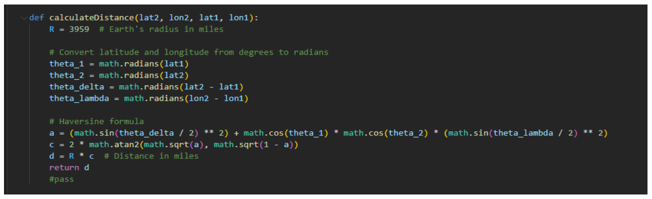


Fig. 1. Haversine Formula calculated between two points.

$$\text{distance} = 2 \cdot R \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1)\cdot\cos(\phi_2)\cdot\sin^2\left(\frac{\Delta\lambda}{2}\right)}\right)$$

where:
- R is the Earth's radius (average radius is approximately 3959 miles or 6371 kilometers).
- $\Delta\phi$ is the difference in latitude ($\phi_2 - \phi_1$).
- $\Delta\lambda$ is the difference in longitude ($\lambda_2 - \lambda_1$).
- ϕ_1 and ϕ_2 are the latitudes of the two points in radians.
- λ_1 and λ_2 are the longitudes of the two points in radians.

Implementation: Figure 2 indicates the implementation of Haversine formula within the **calculateDistance** method, which considers the spherical geometry of the Earth to estimate the distance between two points given their latitude and longitude coordinates.



```python
def calculateDistance(lat2, lon2, lat1, lon1):
    R = 3959  # Earth's radius in miles

    # Convert latitude and longitude from degrees to radians
    theta_1 = math.radians(lat1)
    theta_2 = math.radians(lat2)
    theta_delta = math.radians(lat2 - lat1)
    theta_lambda = math.radians(lon2 - lon1)

    # Haversine formula
    a = (math.sin(theta_delta / 2) ** 2) + math.cos(theta_1) * math.cos(theta_2) * (math.sin(theta_lambda / 2) ** 2)
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    d = R * c  # Distance in miles
    return d
    #pass
```

Fig. 2. Implementation of Distance Algorithm.

- **Step-1:** Set the Earth's radius R to 3959 miles.
- **Step-2:** Convert latitude and longitude from degrees to radians using 'math.radians()'.

- **Step-3:** Compute the difference in latitude ($\Delta\theta$) and difference in longitude ($\Delta\lambda$) by subtracting the respective latitudes and longitudes.
- **Step-4:**
  - Compute a using the Haversine formula:

$$a = \left(\sin^2\left(\frac{\Delta\theta}{2}\right)\right) + \cos(\theta_1)\cdot\cos(\theta_2)\cdot\left(\sin^2\left(\frac{\Delta\lambda}{2}\right)\right)$$

  - Calculate c using the Haversine formula:

$$c = 2 \cdot \text{atan2}\left(\sqrt{a}, \sqrt{1-a}\right)$$

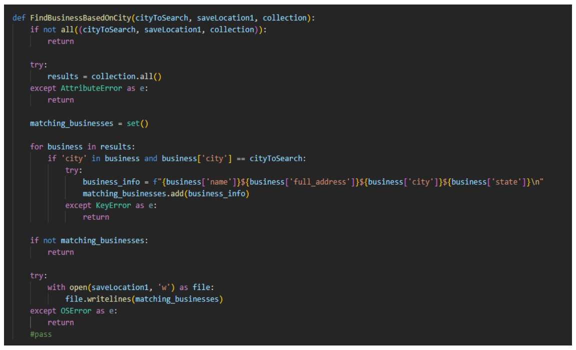
.

- **Step-5:** Compute the distance d using $d = R \cdot c$, where R is the Earth's radius.
- **Step-6:** The function returns the calculated distance d in miles between the given latitude and longitude coordinates using the Haversine formula.

B. Finding list of businesses based on city

The function **FindBusinessBasedOnCity** retrieves all the records from the NoSQL database 'sample.db' and checks if the city of the record from the collection matches with the 'cityToSearch' string. If the city name matches with the string, then name, full address, city and state is retrieved from the tuple with '$' as the delimiter between each string element and it is stored in a set. The main intention of using a set for storing the result is to avoid duplications while adding the records so that the output shows only the unique records. This output is written into a file with a specified file location 'savedLocation1'.

Implementation: Figure 3 indicates the implementation of **FindBusinessBasedOnCity** for the given collection, cityToSearch and saveLocation1.



```
def FindBusinessBasedOnCity(cityToSearch, saveLocation1, collection):
    if not all((cityToSearch, saveLocation1, collection)):
        return

    try:
        results = collection.all()
    except AttributeError as e:
        return

    matching_businesses = set()

    for business in results:
        if 'city' in business and business['city'] == cityToSearch:
            try:
                business_info = f"{business['name']}${business['full_address']}${business['city']}${business['state']}\n"
                matching_businesses.add(business_info)
            except KeyError as e:
                return

    if not matching_businesses:
        return

    try:
        with open(saveLocation1, 'w') as file:
            file.writelines(matching_businesses)
    except OSError as e:
        return
    #pass
```

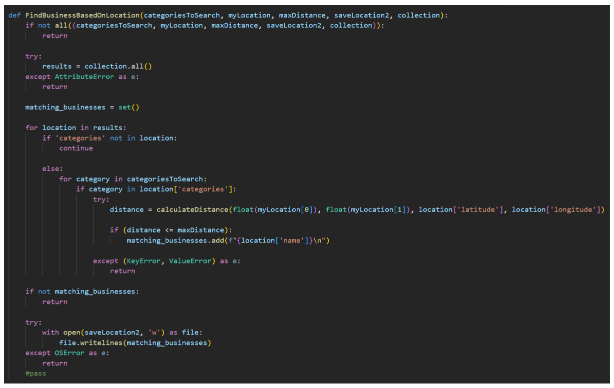
Fig. 3. Implementation of FindBusinessBasedOnCity function.

- **Step-1:** Check if 'cityToSearch', 'saveLocation1', and 'collection' parameters are all provided. If not, return without further execution.
- **Step-2:** Attempt to retrieve data from the provided 'collection' using try-except block. If the collection does not support the '.all()' method, return without further execution.
- **Step-3:** Initialize an empty set named 'matching_businesses' which stores tuple values.
- **Step-4:**
  - Iterate through each business in the results.
  - Check if the 'city' key exists in the business information and if it matches the specified 'cityToSearch'.
  - If both conditions are met, create a formatted string ('business_info') containing the name, full address, city, and state of the business separated by '$'. Add this information to the 'matching_businesses' set.
  - If any KeyError occurs (e.g., missing keys in business data), return without further execution.
- **Step-5:** If no matching businesses were found, return without further execution.
- **Step-6:** The function attempts to open the 'saveLocation1' file in write mode within a try-except block. It then writes the contents of the 'matching_businesses' set to this file. However, if an OSError occurs during the file writing process, the function halts further execution and returns without completing the operation.

C. Finding list of businesses based on location

The function **FindBusinessBasedOnLocation** retrieves all the records from the NoSQL database 'sample.db' and checks if the category of the record from the collection matches with the category from the list of 'categoriesToSearch'. If the category matches, then distance is calculated between the coordinates of 'myLocation' and the coordinates of the matched record and compares with the 'maxDistance'. If the distance is less than the 'maxDistance', then the location name assoicated is retrieved and stored in a set. The main intention of using a set for storing the result is to avoid duplications while adding the records so that the output shows only the unique records. This output is written into a file with a specified file location 'savedLocation2'.

Implementation: Figure 4 indicates the implementation of **FindBusinessBasedOnLocation** for the given collection, categoriesToSearch, myLocation, maxDistance and saveLocation2.



```
def FindBusinessBasedOnLocation(categoriesToSearch, myLocation, maxDistance, saveLocation2, collection):
    if not all((categoriesToSearch, myLocation, maxDistance, saveLocation2, collection)):
        return

    try:
        results = collection.all()
    except AttributeError as e:
        return

    matching_businesses = set()

    for location in results:
        if 'categories' not in location:
            continue

        else:
            for category in categoriesToSearch:
                if category in location['categories']:
                    try:
                        distance = calculateDistance(float(myLocation[0]), float(myLocation[1]), location['latitude'], location['longitude'])

                        if (distance <= maxDistance):
                            matching_businesses.add(f"{location['name']}\n")

                    except (KeyError, ValueError) as e:
                        return

    if not matching_businesses:
        return

    try:
        with open(saveLocation2, 'w') as file:
            file.writelines(matching_businesses)
    except OSError as e:
        return
    #pass
```

Fig. 4. Implementation of FindBusinessBasedOnLocation function.

- **Step-1:** Check if the provided 'categoriesToSearch', 'myLocation', 'maxDistance', 'saveLocation2', and 'collection' are all valid and not 'None'. If any of them are invalid, the function returns without executing further steps.
- **Step-2:** Attempt to retrieve data from the provided 'collection' using try-except block. If the collection does not support the '.all()' method, return without further execution.
- **Step-3:** Initialize an empty set named 'matching_businesses' which stores tuple values.
- **Step-4:**
  - For each location in the results, it checks for existing categories. If categories are present, it matches them with specified categories to search.
  - If a match occurs, it calculates the distance between the location and a given point. If the distance is within the limit, it adds the location's name to the set of matching businesses.
  - Any missing keys or calculation errors, , the function returns without executing further steps.
- **Step-5:** If no matching businesses were found, return without further execution.
- **Step-6:** The function attempts to open the 'saveLocation2' file in write mode within a try-except block. It then writes the contents of the 'matching_businesses' set to this file. However, if an OSError occurs during the file writing process, the function halts further execution and returns without completing the operation.

II. Lessons Learned

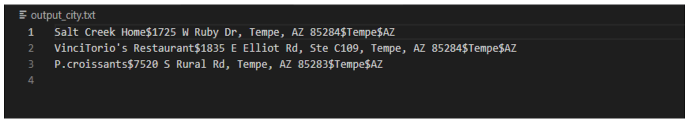
I have learned the following lessons during the course of my project:

- NoSQL databases have a lot of advantages for real-time huge and unstructured data compared to widely used relational databases.
- The UnQLite engine within NoSQL database provides high performance and more scalability for large data.
- Working with spatial data is different from working with other types of data as spatial data involves working with coordinates of a particular location.
- Usage of Haversine formula in calculating the distance between two points on a curved surface.

III. Output

The following text files were generated from the two functions for the given two initial test cases.

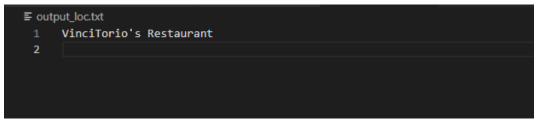
The output generated by the function **FindBusinessBasedOnCity('Tempe', 'output_city.txt', data)** is given by Figure 5. True results also contain the same information in the form of a list. The order of the data in the text file may be different based on the compiler.



```
output_city.txt
Salt Creek Home$1725 W Ruby Dr, Tempe, AZ 85284$Tempe$AZ
VinciTorio's Restaurant$1835 E Elliot Rd, Ste C109, Tempe, AZ 85284$Tempe$AZ
P.croissants$7520 S Rural Rd, Tempe, AZ 85283$Tempe$AZ
```

Fig. 5. Output of FindBusinessBasedOnCity function.

The output generated by the function **FindBusinessBasedOnLocation(['Buffets'], [33.3482589, -111.9088346], 10, 'output_loc.txt', data)** is given by Figure 6. True results also contain the same information. There can be multiple businesses in a single location.



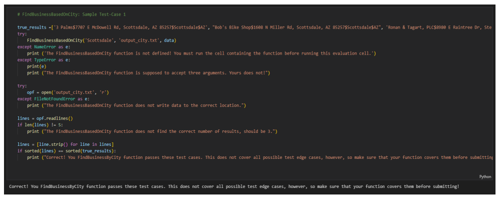
```
output_loc.txt
VinciTorio's Restaurant
```

Fig. 6. Output of FindBusinessBasedOnLocation function.

IV. Result

All the additional test cases given are passed for the two functions.

Test Case-1 for **FindBusinessBasedOnCity** function.



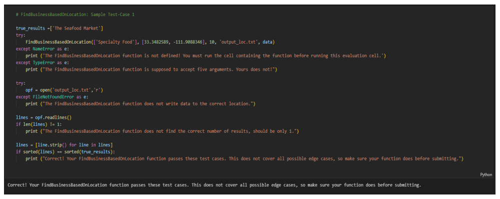
```python
# FindBusinessBasedOnCity: Sample Test-Case 1

true_results =['3 Palms$7707 E McDowell Rd, Scottsdale, AZ 85257$Scottsdale$AZ', "Bob's Bike Shop$1608 N Miller Rd, Scottsdale, AZ 85257$Scottsdale$AZ", 'Ronan & Tagart, PLC$8980 E Raintree Dr, Ste
try:
    FindBusinessBasedOnCity('Scottsdale', 'output_city.txt', data)
except NameError as e:
    print ('The FindBusinessBasedOnCity function is not defined! You must run the cell containing the function before running this evaluation cell.')
except TypeError as e:
    print(e)
    print ("The FindBusinessBasedOnCity function is supposed to accept three arguments. Yours does not!")

try:
    opf = open('output_city.txt', 'r')
except FileNotFoundError as e:
    print ("The FindBusinessBasedOnCity function does not write data to the correct location.")

lines = opf.readlines()
if len(lines) != 5:
    print ("The FindBusinessBasedOnCity function does not find the correct number of results, should be 3.")

lines = [line.strip() for line in lines]
if sorted(lines) == sorted(true_results):
    print ("Correct! You FindBusinessByCity function passes these test cases. This does not cover all possible test edge cases, however, so make sure that your function covers them before submittin
```

Python

```
Correct! You FindBusinessByCity function passes these test cases. This does not cover all possible test edge cases, however, so make sure that your function covers them before submitting!
```

Fig. 7. Test Case of FindBusinessBasedOnCity function.

Test Case-1 for **FindBusinessBasedOnLocation** function.



```python
# FindBusinessBasedOnLocation: Sample Test-Case 1

true_results =['The Seafood Market']
try:
    FindBusinessBasedOnLocation(['Specialty Food'], [33.3482589, -111.9088346], 10, 'output_loc.txt', data)
except NameError as e:
    print ('The FindBusinessBasedOnLocation function is not defined! You must run the cell containing the function before running this evaluation cell.')
except TypeError as e:
    print ("The FindBusinessBasedOnLocation function is supposed to accept five arguments. Yours does not!")

try:
    opf = open('output_loc.txt','r')
except FileNotFoundError as e:
    print ("The FindBusinessBasedOnLocation function does not write data to the correct location.")

lines = opf.readlines()
if len(lines) != 1:
    print ("The FindBusinessBasedOnLocation function does not find the correct number of results, should be only 1.")

lines = [line.strip() for line in lines]
if sorted(lines) == sorted(true_results):
    print ("Correct! Your FindBusinessBasedOnLocation function passes these test cases. This does not cover all possible edge cases, so make sure your function does before submitting.")
```

Python

```
Correct! Your FindBusinessBasedOnLocation function passes these test cases. This does not cover all possible edge cases, so make sure your function does before submitting.
```

Fig. 8. Test Case of FindBusinessBasedOnLocation function.

The other test cases are also passed and provides the same output as in true_result list and these are available in the notebook file.

V. Acknowledgements

I would like to express my gratitude to Dr. Samira Ghayekhloo for her guidance in explaining spatial data and NoSQL concepts, as well as to Yujian Xiong and Qixian Zhao for their support in in addressing doubts or queries related to the subject.

REFERENCES

[1] Movable Type Scripts: Calculate distance, bearing and more between Latitude/Longitude points https://www.movable-type.co.uk/scripts/latlong.html.