Reflection

The first part of the project is figuring out the pseudo-code. Knowing that UnQlite is a Key-Value store and Document-store database engine has helped me understand how data is stored within. I did some research to figure out how the best way to iterate through all the key-values in the collection; such finding were helpful in writing both functions.

1. **FindBusinessBasedOnCity(cityToSearch, saveLocation1, collection)**

```
def FindBusinessBasedOnCity(cityToSearch, saveLocation1, collection):

    f = open(saveLocation1, 'w')

    for i in range(0, len(collection)):

        a = collection.fetch(i)['name'].decode()
        b = collection.fetch(i)['full_address'].decode()
        c = collection.fetch(i)['city'].decode()
        d = collection.fetch(i)['state'].decode()
        if c == cityToSearch:
            f.write(a +"$"+ b + "$" + c +"$" + d)
            f.write('\n')
```

For loop to iterate through all the stored list of key-values in the collection

Extracting the values of the corresponding key that should be included in the output file. value.decode() changes bytes to strings

If the city value is matching the argument (cityToSearch), then the results shall be written into the file.

2. **FindBusinessBasedOnLocation(categoriesToSearch, myLocation, maxDistance, saveLocation2, collection) – included the distance functiona**

This function is more intricate with more details to consider. Because unlike cityToSearch, the argument categoriesToSearch is an array; all the elements need to be accessed and compared the against the elements in the collection

```
def FindBusinessBasedOnLocation(categoriesToSearch, myLocation, maxDistance,
saveLocation2, collection):
    import math

    lat1 = myLocation[0]
    lon1 = myLocation[1]


    f = open(saveLocation2, 'w')

    lat3 = 0
    lon3 = 0
```

Defining the latitude and longitude given in the argument to use it in the distance function

Pre-defining these variables to avoid 'variable used before assignment' error. Such variables act as containers to store the latitude and longitude from the previous iteration and compare with the latitude and longitude of the current iteration.

Iterating through lists in the collection

Iterate through all the elements in the categories key for each value stored

Iterate through all the elements in the categoriesToSearch argument

```
for i in range(0, len(collection)):
    x = collection.fetch(i)['categories']
    for j in range(0,len(x)):
        for k in range(0,len(categoriesToSearch)):
            a = x[j].decode()
            if a == categoriesToSearch[k]:
                lat2 = collection.fetch(i)['latitude']
                lon2 = collection.fetch(i)['longitude']
                R = 3959
                o1 = math.radians(lat1)
                o2 = math.radians(lat2)
                diffo = math.radians(lat2-lat1)
                difflamda = math.radians(lon2-lon1)
                a = math.sin(diffo/2) * math.sin(diffo/2) + math.cos(o1) *
                math.cos(o2) * math.sin(difflamda/2) * math.sin(difflamda/2)
                c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
                d = R * c

                if d <= maxDistance:
                    if (lat3 == lat2 and lon3 == lon2):
                        break
                    y = collection.fetch(i)['name'].decode()
                    f.write(y)
                    f.write('\n')
                    lat3 = lat2
                    lon3 = lon2

f.close()
```

The function was almost copied as is from the instructions. However, no special characters were used to avoid any errors. The math library was imported at the beginning of the code. lat1 and lat1 are based on myLocation argument. Lan2 and lat2 are the values returned by eacher iteration.

For the sole purpose of passing the autograder, the distance function was not predefined. Instead, I just included it within the loop.

If statement to only write the restaurants' names with latitude and longitude that results in a distance less than or equal the maxDistance input.

This step is important to remove any duplicates. For example, if the we have 2 matching categories for the same restaurant, the restaurant name will be included twice in the output file.

Debugging: Removing the last line in the output TXT file

The for loop I used create a new line after every result, which remains as blank line after writing the final result. It falsely gives the wrong number of results; the test function counts the blank line as an extra result. I copied the pre-written function below from codespeedy.com.

```
fd=open(saveLocation2,"r")
d=fd.read()
fd.close()
m=d.split("\n")
s="\n".join(m[:-1])
fd=open(saveLocation2,"w+")
for i in range(len(s)):
    fd.write(s[i])
fd.close()
```

Lessons Learned

- As a first-time user, I learned about Jupyter notebook. It is a very reliable way to use python in the cloud and run multiple terminals simultaneously.
- Applied what I learned in week 5 about NoSQL databases by using UnQlite, which is more versatile code when it comes to running queries or extracting information.
- It took me about a day to familiarize myself with Jupyter notebook and figure out where the files to be read and the ones written are stored. Therefore, an important lesson learned is that one must start programming assignments and projects as early as possible.

Output

output_city.txt output for the first test case (3rd cell) for the FindBusinessBasedOnCity function:

FindBusinessBasedOnCity('Tempe', 'output_city.txt', data)

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

output_loc.txt output for the second test case (4th cell) for the FindBusinessBasedOnLocation function:

FindBusinessBasedOnLocation(['Buffets'], [33.3482589, -111.9088346], 10, 'output_loc.txt', data)

```
VinciTorio's Restaurant
```

Result

I passed all the autograder tests and got a full score.

For the **`FindBusinessBasedOnCity`** function, my code has successfully covered the following:

- Filtering out the results for the right city in the argument
- Successfully concatenated $ between every value filtered and decoded from UnQlite
- Produced an output with the right number of results.

For the **FindBusinessBasedOnLocation** function, my code has successfully covered the following:

- Stored the right longitude and latitude for a given location in variables
- Accurately calculate the maximum distance (in miles) for every restaurant in the collection based on its latitude and longitude
- Only include the restaurant names within the maximum calculated distance in the results
- Avoid duplicates results. The matching restaurant name was only included once in the results; an if statement that compares current and previous latitude and longitudes skips writing the same restaurant into the TXT file twice.