IST 256: Project Proposal

Samantha Gordon and Jessica Zuk

For the final project, we decided to build a program that would display the top restaurants in a major halfway point between two destination. The user will enter in two destinations, and then will calculate and locate a major city between the two. The tops restaurants will be identified through Google Places/Reviews.

**Inputs:**

The two city locations

Using API

**Outputs:**

Top Restaurants in the City

**Steps:**

Call on location API (Google Distance)

Store input as locations variable.

Create an algorithm that will take the midpoint of the two cities, through Google Distance, to locate closest major city. Using variable of location, create city variable.

Call on review API to correlate city with restaurants in the area.

Using the city variable, identify top restaurants in the area, and print out specifics of each (ex, cuisine, price, address, etc.)

**Explanation of Our Code:**

We pip installed the following in order to make sure that each computer is able to run the appropriate applications and installations.This ensures all the software and packages are on the computers that the app will execute on.

1. pip install python-google-places
2. pip install -U googlemaps
3. pip install geocoders
4. pip install u-google-places
5. pip install geopy(where we got geocoders, which helps to find the location, also where nominatim came from )

We started with importing several key elements in order to use the appropriate APIs necessary to make sure the code pulls the from the correct information.

Definition of the imports,

An essential component of the code is to obtain a Google Place API code, which can be acquired here:

<https://developers.google.com/places/web-service/get-api-key>

Here, you are asked to start a project, and then it will output a key, which will retrieve all information from the Google Places API.

After, we asked the user to input two addresses as strings. This was handy because the API accounted for many errors. For instance, the user could enter in something as generic as an entire state, or be as specific as an entire address, and the application will locate the midpoint. The latitude and longitude from each destination will be divided by two to locate the halfway point. We then included all of the error messages and took care of the potential messages we would receive.

The next part would take the API and register the midpoint with the correct parameters to find the places the two people can meet. If no results showed up, the code would then automatically extent the radius to a larger distance to show up to 20 top places. If no results show up, then the code will ask the user to try again.

Then, the locations will be mapped using Folium, where we customize the map type and color. Based on the midpoint, we count the number of results in dataframe to ensure all are accounted for on the map using len to plot the points. Then, a while loop goes through plotting points on the map as many times as it needs to. We set popup message so that name and address (vicinity) appear on Markers, and added icons, and color.

Then it would map the results. We asked the user if they would like the map saved as an HTML too.

**Examples We Accounted For in the Input:**

Boston/San Diego(accounts for the rankby= ‘distance’):

This is used to showcase how an obscure area between the two locations with no restaurants will use the larger radius in order to find adequate results for the user to see. Because the area is more remote, the radius needs to expand in order to find enough information to pull from .

Boston/Paris(accounts for areas with no data):

This example was used in order to identify the areas which have no true data, even within 50,000 meters. For instance, the midpoint between Boston and Paris is in the middle of the ocean, therefore, no data is available. We accounted for this by identifying an empty dataframe result in no information, after the expanded radius was also unable to find any data. A message would emerge stating no data is available, and asking the user to try two different locations.

Asheville/Charlotte (accounts for limited data in the dataframe):

This is an example we found where there were only two restaurants within the radius. The API typically prints 20 results, however, we noted that some midpoints would result in less than the defaulted 20. Therefore, we counted the rows and set that number as a variable in order for the loop to run the specified number for each example. This means that the all the results within the maximum of 20 will be displayed as markers on the map.

Gibberish/Gibberish (accounts for nonsense/errors):

This would verify that when the user inputs nonsense, then the application would identify the error and tell the user to try again. This would come out as an Attribute Error.

89/Boston (accounts for other error):

Here, this would account for TypeErrors.

Who Did What:

Sammy:

She wrote the initial parts of the code, including the algorithm for the midpoint and how to calculate the APIs and retrieve the info. She made the video,and collaborated with the rest of the code.

Jessica:

She created the posters, and worked to finish the code with Sammy. She wrote the explanation of the code.

Sources:

<http://folium.readthedocs.io/en/latest/quickstart.html>

<https://maps.googleapis.com/maps/api/place/nearbysearch/json?location=-33.8670522,151.1957362&radius=500&type=restaurant&keyword=cruise&key=AIzaSyA8Yw3voyJlQP0dfa_1v2Jku-RvrSXIfME>

<https://developers.google.com/places/web-service/search>

<https://console.cloud.google.com/apis/credentials?project=pythonfinalproje-1512492836114>

<https://github.com/geopy/geopy>

<http://geopy.readthedocs.io/en/latest/>

<http://nbviewer.jupyter.org/gist/ocefpaf/d1d612d290f766935d8abe1559523a72>

\*\*