Tourism Finder

Applied Data Science Capstone

14th June 2020

A.ALKAFF AHAMED

Table of Contents

Section 1
Introduction to Project

In this section, the project and its scope will be discussed.

What is Tourism?

In many countries, tourism is an important sector for generating income for the country. Tourism attractions are places where tourists from other places or countries will come in and enjoy their time there. It can be a monument, a beach, a museum, a heritage center, or really anything that is unique and attract attention. A group of tourism attractions within a certain radius of a place is considered a tourism hotspot.

> ***Tourism Attraction: A place of interest visited by tourists***
>
> ***Tourism Hotspot: A city with a group of Tourism Attractions***

In cities with tourist hotspots, it is likely that you can find significantly more foreign people. Just like any other places, tourism hotspots also require the amenities like restaurants, shops, hotels and more. Since there are more foreigners, they would not have the knowledge of market price of products. So the shops can sell their products for a significantly higher price to gain higher profits.

Tourism Finder Introduction

Tourism finder is an application that is used to find Tourism given any city. Finding tourism hotspots in a city is really important because several businesses can

thrive in tourism hotspots. Here is a sample list of businesses that can thrive in tourism hotspots:

1. Planning a Vacation
2. Fancy Restaurants
3. Nightclubs
4. Hotels and Star Hotels
5. Malls
6. Souvenir Shops
7. And more...

This application is designed to take in the city name as input and an interactive map of the region will be generated as output with the tourism attractions superimposed over the map as circles.

Section 2
Data Sources

In this section, the data sources and the steps to obtain the final output will be discussed. The following libraries and API will be used for obtaining data:

1. GeoPy Library
2. Foursquare API
3. Folium Library

GeoPy Library

This library is used to obtain the coordinates. The input will be city name and the output will be the latitude and longitude.

Foursquare API

This API is used to locate the tourism attractions given the coordinates. The output will be a list of tourism attractions with their coordinates.

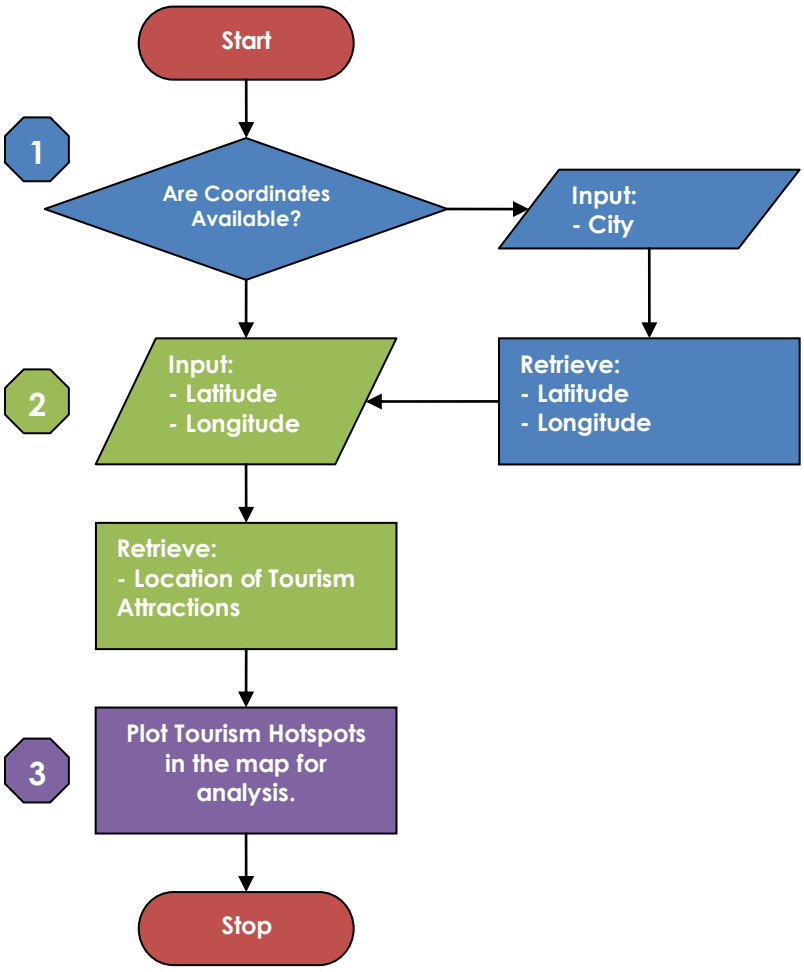
Folium Library

This library is used to plot the map and superimpose circular markers showing where the tourism attractions are in the city.

Section 3:
Methodology

In this section, the method of retrieving the data and the processing done on the data will be discussed. The flowchart below illustrates the 3 steps of the working of the program.

- Step 1: Retrieve coordinates of the city
- Step 2: Retrieve location data of tourist attractions in the city
- Step 3: Plot the tourism attractions in the map



Retrieving Coordinates of City

If the user has the coordinates, this step can be skipped.

This step is straightforward.

1. Import the library
   `from geopy.geocoders import Nominatim`.
2. Create a geolocator object
   `geolocator = Nominatim(user_agent="foursquare_agent")`
3. Create a location object
   `location = geolocator.geocode("city_name")`.
4. The attributes `latitude` and `longitude` of `location` object will contain the coordinates we need. Store them into a variable

Retrieve Location Data of Tourist Attractions in the City

In this step, there are 2 parts. First part, the data will be retrieved using the Foursquare API and second part, the data will be cleaned.

Retrieving Data

The Foursquare API will be used to retrieve the location data of all tourism attractions around the coordinates. The `search` query will be used with the default radius of 50 km.

1. The Get request is sent using the request library and parsed as a JSON file.
   `results = requests.get(url).json()`
2. The locations will be extracted and stored into a dataframe
   `venues = results['response']['venues']`
   `dataframe = json_normalize(venues)`

Cleaning Data

The data we obtain is not exactly in the perfect format. If we check the columns, the Category column contains a sub data structure with category name, category id and other properties. Only the category name is required, so that will be extracted and the remaining data will be discarded. The function `get_category_type(row)` will do this job.

We can apply the function to the dataframe by calling `apply()` function and passing the `get_category_type(row)` function as the argument.

Plot the Tourism Attractions in the Map

In this step, the data in the dataframe will be used to plot the points in the map. The Folium library is a beautiful interactive map with a functionality to add points on the map. The data from the dataframe contains the latitude and longitude which are the minmum required data for creating the points.

1. Initialize the map object
   ```
   tourism_map = folium.Map(location=[lat, lon], zoom_start=11)
   ```
2. Add the points using for loop
   ```
   for lati, lngi, attr in zip(dataframe['location.lat'],...):
        lbl = folium.Popup(attr, parse_html=True)
        folium.features.CircleMarker([lati, lngi], radius=5,
   popup=lbl,...).add_to(tourism_map)
   ```
3. Display the map by simply calling tourism_map

Section 4:
Results

The application ran successfully and as expected. In this section, the test run of the application will be discussed and explained with screenshots.

Retrieve Coordinates of the City

This step retrieves the coordinates and prints the confirmation as output. User can either enter city (as shown in Figure 1) or skip this step by directly entering coordinates (as shown in Figure 2).

Dashboard

You will set parameters here. In fact, this dashboard can be replaced with any other kind to get the user input, example, Command-Line Interface, GUI.

```
# Set city name
city = "Singapore"

# Or set latitude and longitude manually
# and set city to None by uncommenting the line
lat = 13.0827
lon = 80.2707
# city = None
```

Step 1: Retrieve coordinates

```
if city != None:
    geolocator = Nominatim(user_agent="foursquare_agent")
    location = geolocator.geocode(city)
    #print(type(location))
    lat = location.latitude
    lon = location.longitude
    print ("The coordinates of %s are (%f, %f)." % (city, lat, lon))
else:
    print ("Your chosen coordinates are (%f, %f)." % (lat, lon))
```

```
The coordinates of Singapore are (1.357107, 103.819499).
```

Figure 1: Example of finding coordinates of Singapore

Dashboard

You will set parameters here. In fact, this dashboard can be replaced with any other kind to get the user input, example, Command-Line Interface, GUI.

```
# Set city name
city = "Singapore"

# Or set latitude and longitude manually
# and set city to None by uncommenting the line
lat = 13.0827
lon = 80.2707
city = None
```

Step 1: Retrieve coordinates

```
if city != None:
    geolocator = Nominatim(user_agent="foursquare_agent")
    location = geolocator.geocode(city)
    #print(type(location))
    lat = location.latitude
    lon = location.longitude
    print ("The coordinates of %s are (%f, %f)." % (city, lat, lon))
else:
    print ("Your chosen coordinates are (%f, %f)." % (lat, lon))
```

```
Your chosen coordinates are (13.082700, 80.270700).
```

Figure 2: Example showing user defined coordinates

There are 2 different print messages depending on what the user chose. For the subsequent examples, Singapore coordinates would be used.

Retrieve location data of tourist attractions in the city

This step retrieves the tourism attraction location data for our example Singapore. First the data is downloaded and whether it succeeded or not, the status will be printed as shown below.

Step 2: Retrieve list of tourism attractions

Retrieve by using Foursquare API Call

```
search_query = 'Tourism'
radius = 50000
url = 'https://api.foursquare.com/v2/venues/search?client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.forma
results = requests.get(url).json()
print("Status code: %d."%results['meta']['code'])
```

```
Status code: 200.
```

Figure 3: Example showing successful download of data

Status Code 200 means the download was successful. Any other status code means some error has occurred.

Next, the data is formatted into a data frame as shown below.

Process the JSON file

```
# Assign relevant part of JSON to venues
venues = results['response']['venues']

# Tranform venues into a dataframe
dataframe = json_normalize(venues)
print("%d Results loaded." % dataframe.shape[0])
print("Displaying first 3 rows:")
dataframe.head(3)
```

```
50 Results loaded.
Displaying first 3 rows:
```

	categories	hasPerk	id	location.address	location.cc	location.city	location.country	location.crossStreet	location.di
0	[{'id': '4bf58dd8d48988d12c951735', 'name': 'E...	False	4cff4c65f7b38cfa9a7cc8c3	370 Orchard Road	SG	Singapore	Singapore	NaN	
1	[{'id': '52e81612bcbc57f1066b7a13', 'name': 'N...	False	52e0d6f4498e23eedfbea071	Mandai Lake Road	SG	Singapore	Singapore	NaN	
2	[{'id': '4bf58dd8d48988d1f6931735', 'name': 'G...	False	4bfa092f8f32ef3bd0cf04aa	80 Robinson Rd	SG	Singapore	Singapore	NaN	

Figure 4: Example showing successful conversion of JSON to data frame

The categories column is not of expected data format. Cleaning up data includes fixing this issue also.

Upon examination, the categories column consists of a sub structure which holds the name, id and other properties of the category. Only the name of the category will be required. So, the filtering is done as per requirements and the screenshot below shows the result.

Fix the Categories column

```
# Fix the Categories Column

# Function that extracts the category of the venue
def get_category_type(row):
    try:
        categories_list = row['categories']
    except:
        categories_list = row['venue.categories']

    if len(categories_list) == 0:
        return None
    else:
        return categories_list[0]['name']

# Apply filter to the categories column
dataframe['categories'] = dataframe.apply(get_category_type, axis=1)
dataframe.head(3)
```

	categories	hasPerk	id	location.address	location.cc	location.city	location.country	location.crossStreet	location.distance	location.f
0	Embassy / Consulate	False	4cff4c65f7b38cfa9a7cc8c3	370 Orchard Road	SG	Singapore	Singapore	NaN	5817	[370 Orcha
1	Nature Preserve	False	52e0d6f4498e23eedfbea071	Mandai Lake Road	SG	Singapore	Singapore	NaN	6621	[M
2	General Travel	False	4bfa092f8f32ef3bd0cf04aa	80 Robinson Rd	SG	Singapore	Singapore	NaN	9230	

Figure 5: Example showing successful cleaning of categories column

Now the data is cleaned up, it is ready for plotting.

Plot the tourism attractions in the map

This step plots the cleaned data into the map. The map is interactive, which means if you click the points, the details will show up. The red dot is the location of the original point used for query and the blue dots are the location of the tourist attractions. The following shows the screenshot of the plot:

Step 3: Plot them onto a map

```
# Initialise Map object
tourism_map = folium.Map(location=[lat, lon], zoom_start=11)

# Red circle represents center location
if city != None:
    lbl = folium.Popup(city + " (" + str(lat) + ", " + str(lon) + ")", parse_html=True)
else:
    lbl = folium.Popup(" (" + str(lat) + ", " + str(lon) + ")", parse_html=True)
folium.features.CircleMarker([lat, lon], radius=5, popup=lbl, fill=True, color='red',
                             fill_color='red', fill_opacity=0.6).add_to(tourism_map)

# Blue circles represent tourism spots
for lati, lngi, cat, attr in zip(dataframe['location.lat'],dataframe['location.lng'],dataframe['categories'], dataframe['name']):
    if cat == None:
        lbl = folium.Popup(str(attr), parse_html=True)
    else:
        lbl = folium.Popup(str(attr) + ' (' + str(cat) + ')', parse_html=True)
    folium.features.CircleMarker([lati, lngi], radius=5, popup=lbl, fill=True, color='blue',
                                 fill_color='blue', fill_opacity=0.6).add_to(tourism_map)

tourism_map
```

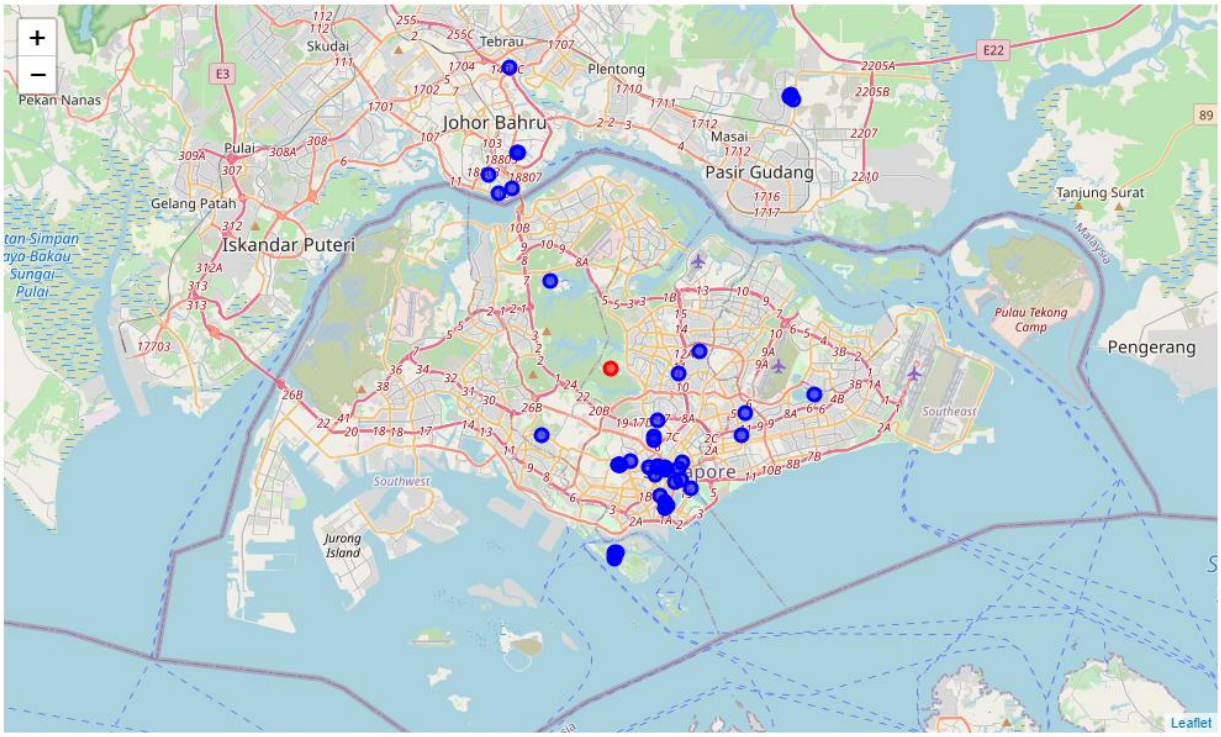


Figure 6: Example showing successful plotting of tourism attractions in Singapore

Section 5:
Discussion

In this section, the outcome of the application and any observations made will be discussed.

The Problem with Query String "Tourism"

For the Foursquare API, the search query with the query string "Tourism" returned some tourism attractions correctly like Sentosa. However, other attractions like Mandai Zoo, Jurong Bird Park, Wild Wild Wet were not returned. Also, some tourism agencies were returned too. This makes the final result inaccurate.

This problem can be solved by running multiple queries for different query strings to match those kind of attractions.

Machine learning models can also be used to correct this error by training the model with known attractions. This can improve the accuracy of the results.

Section 6:
Conclusion

In conclusion, the application is able to identify regions of tourist attractions quite accurately. Although the specific items identified are sometimes not tourist attractions, there are tourist attractions in the regions.

This application does at least a good job in identifying clusters where tourism exists. In the next version, the application can be upgraded with better tourism attraction detection systems and produce better results.