Segmenting and Clustering Neighborhoods

Sam

November 19 ,2020

1. Introduction

In this assignment, I will be required to explore, segment, and cluster the neighborhoods in the city of Toronto and New York city, to look closer about the different and similarity of those two business center.

However, unlike New York, the neighborhood data is not readily available on the internet. What is interesting about the field of data science is that each project can be challenging in its unique way, so you need to learn to be agile and refine the skill to learn new libraries and tools quickly depending on the project.

For the Toronto neighborhood data, a Wikipedia page exists that has all the information we need to explore and cluster the neighborhoods in Toronto. You will be required to scrape the Wikipedia page and wrangle the data, clean it, and then read it into a pandas dataframe so that it is in a structured format like the New York dataset.

Once the data is in a structured format, you can replicate the analysis that we did to the New York City dataset to explore and cluster the neighborhoods in the city of Toronto.

2. Data processing

Step 1 - Scrap Wikipedia for data using BeautifulSoup

Corresponding to the different postcodes of Toronto, for this purpose we will use BeautifulSoup library in Python. In addition, we will need to follow the next criteria:
- The dataframe will consist of three columns: PostalCode, Borough, and Neighborhood
- Only process the cells that have an assigned borough. Ignore cells with a borough that is Not assigned.

Use the Notebook to build the code to scrape the following Wikipedia page, https://en.wikipedia.org/wiki/List_of_postal_codes_of_Canada:_M, in order to obtain the data that is in the table of postal codes and to transform the data into a pandas dataframe like the one shown below:

To create the dataframe:

- The dataframe will consist of three columns: PostalCode, Borough, and Neighborhood.
- Only process the cells that have an assigned borough. Ignore cells with a borough that is Not assigned.
- More than one neighborhood can exist in one postal code area. For example, in the table on the Wikipedia page, you will notice that M5A is listed twice and has two neighborhoods: Harbourfront and Regent Park. These two rows will be combined into one row with the neighborhoods separated with a comma as shown in row 11 in the above table.
- If a cell has a borough but a Not assigned neighborhood, then the neighborhood will be the same as the borough. So for the 9th cell in the table on the Wikipedia page, the value of the Borough and the Neighborhood columns will be Queen's Park.
- Clean your Notebook and add Markdown cells to explain your work and any assumptions you are making.
- In the last cell of your notebook, use the .shape method to print the number of rows of your dataframe.

Note: There are different website scraping libraries and packages in Python. One of the most common packages is BeautifulSoup. Here is the package's main documentation page: http://beautiful-soup-4.readthedocs.io/en/latest/

For ignoring the cells without a Borough we will use the following code:

```
toronto_dropNotAssigned = toronto_df[toronto_df.Borough != 'Not assigned'].reset_index(drop=True)
```

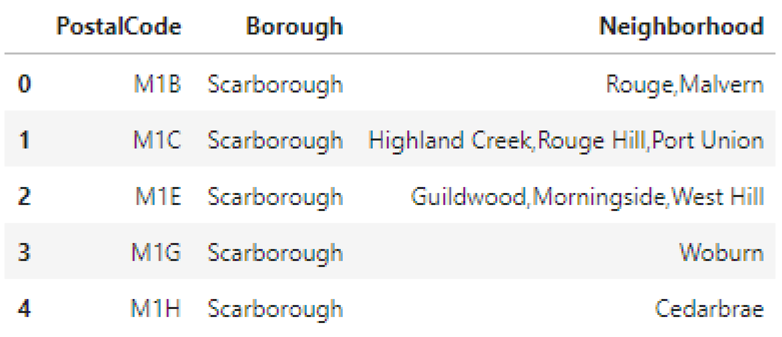
for grouping the different neighborhoods and separe them with a comma I will use the following code:

```
toronto_grouped = toronto_dropNotAssigned.groupby(['PostalCode','Borough'], as_index=False).agg(lambda x: ','.join(x))
```

And Finally, to assign an empty neighborhood the next one:

```
mask = toronto_grouped['Neighborhood'] == "Not assigned"
toronto_grouped.loc[mask, 'Neighborhood'] = toronto_grouped.loc[mask, 'Borough']
```

looking like this right now:

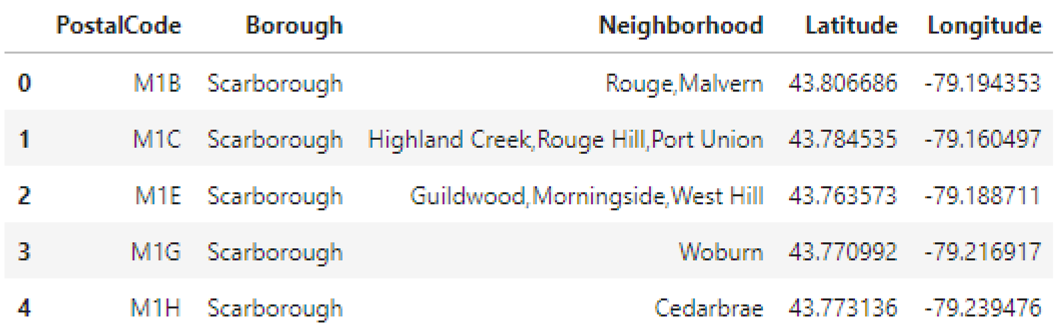


	PostalCode	Borough	Neighborhood
0	M1B	Scarborough	Rouge,Malvern
1	M1C	Scarborough	Highland Creek,Rouge Hill,Port Union
2	M1E	Scarborough	Guildwood,Morningside,West Hill
3	M1G	Scarborough	Woburn
4	M1H	Scarborough	Cedarbrae

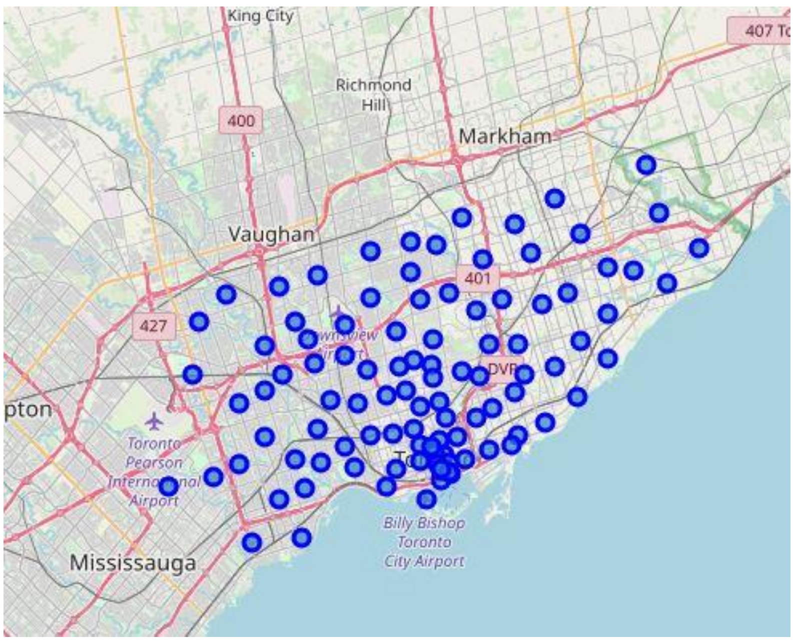
Once we have that table, I will append each latitude and longitude to the PostcalCode, for this purpose, I will use a simple inner join because I already have a .csv file from this page: https://cocl.us/Geospatial_data , the code for this will look like this:

```
#For SQL purpose I will make Postal Code as only one Word: PostalCode
coordenades.rename(index=str, columns={"Postal Code": "PostalCode"}, inplace = True)
neighborhood = pd.merge(toronto_grouped, coordenades, on='PostalCode', how='inner')
```

as a result, I will have a table that looks like this one:



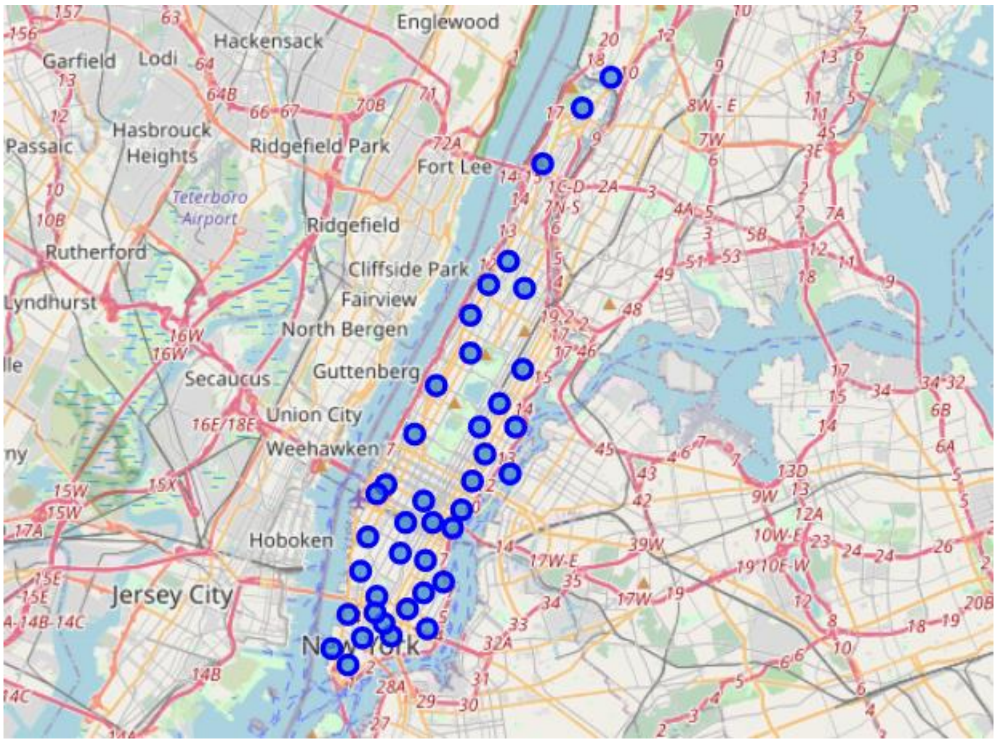
	PostalCode	Borough	Neighborhood	Latitude	Longitude
0	M1B	Scarborough	Rouge,Malvern	43.806686	-79.194353
1	M1C	Scarborough	Highland Creek,Rouge Hill,Port Union	43.784535	-79.160497
2	M1E	Scarborough	Guildwood,Morningside,West Hill	43.763573	-79.188711
3	M1G	Scarborough	Woburn	43.770992	-79.216917
4	M1H	Scarborough	Cedarbrae	43.773136	-79.239476

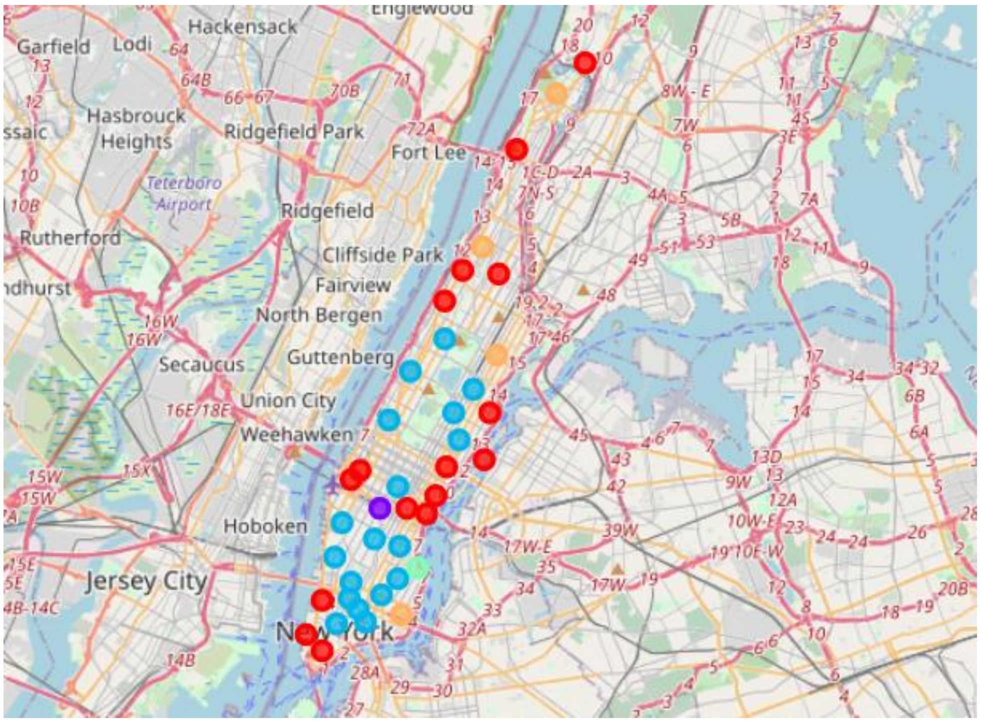

King City
407
Richmond Hill
400
Markham
Vaughan
401
427
DVP
Toronto Pearson International Airport
Mississauga
Billy Bishop Toronto City Airport

Step 2 - EXTRACTING INFO from Foursquare API

Now that I have the map with each of the postalcodes in it, using the FourSquare API (for more info go their main webpage: https://foursquare.com/) I will fetch all the shops, buildings surrounding those postcodes using the following code:

```python
# Setup API URL to explore venues near by
LIMIT = 100
radius = 500
url = 'https://api.foursquare.com/v2/venues/explore?client_id={}&client_secret={}&ll={},{}&v={}&radius={}&limit={}'.format(CLIENT_ID, CLIENT_SECRET, neighborhood_latitude, neighborhood_longitude, VERSION, radius, LIMIT)
neighborhood_json = requests.get(url).json()

# 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']

venues = neighborhood_json['response']['groups'][0]['items']

nearby_venues = json_normalize(venues) # flatten JSON

# filter columns
filtered_columns = ['venue.name', 'venue.categories', 'venue.location.lat', 'venue.location.lng']
nearby_venues =nearby_venues.loc[:, filtered_columns]

# filter the category for each row
nearby_venues['venue.categories'] = nearby_venues.apply(get_category_type, axis=1)

# clean columns
nearby_venues.columns = [col.split(".")[-1] for col in nearby_venues.columns]

nearby_venues.head()
def getNearbyVenues(names, latitudes, longitudes, radius=500):

    venues_list=[]
    for name, lat, lng in zip(names, latitudes, longitudes):
        print(name)

        # create the API request URL
        url = 'https://api.foursquare.com/v2/venues/explore?&client_id={}&client_secret={}&v={}&ll={},{}&radius={}&limit={}'.format(
            CLIENT_ID,
            CLIENT_SECRET,
            VERSION,
            lat,
            lng,
            radius,
            LIMIT)

        # make the GET request
        results = requests.get(url).json()["response"]['groups'][0]['items']

        # return only relevant information for each nearby venue
        venues_list.append([(
            name,
            lat,
            lng,
            v['venue']['name'],
            v['venue']['location']['lat'],
            v['venue']['location']['lng'],
            v['venue']['categories'][0]['name']) for v in results])

    nearby_venues = pd.DataFrame([item for venue_list in venues_list for item in venue_list])
    nearby_venues.columns = ['Neighborhood',
                  'Neighborhood Latitude',
                  'Neighborhood Longitude',
                  'Venue',
                  'Venue Latitude',
                  'Venue Longitude',
                  'Venue Category']

    return(nearby_venues)
```





3. Discussion &conclusion

This solution is based purely on the geographic location, which is quite good in most cases. Since in the business of cafe, the right location is almost THE important decision to make.

Nevertheless, if we could find more information on each cluster, it will improve the solution further.

The clustering is very useful and intuitive to solve the problem related to geographic data. We should also pay attention to the outcomes of the clustering, especially when it yields a result different ideas of a 'business circle' in a city. Especially it shown somehow concentration near the streamside. It may because the travel covnience(via water way) and landscape.