Capstone Project - The Battle of Neighborhoods (Week 2) Neighborhoods of New York City and Toronto

Hassan Farahani

IBM Data Science Professional Certificate

January 2021

Contents

1 Introduction and Business Problem

In this project, the New York City (USA) and the city of Toronto (Canada) are going to have explored and segmented and clustered their neighbourhoods. Both cities are very diverse and are the financial capitals of their respective countries. The method that is going to be used is a machine learning technique called K-means. It is an unsupervised machine learning technique. Clustering the neighbourhoods of both cities based on the venues categories of each neighbourhood help us determine the types of businesses that exist in different neighbourhoods of both cities, the geographic distribution of most common business in both cities, and the common businesses in both cities.

The most important outcome of the project is to give us an idea about how similar or dissimilar these two cities are. Is New York City more like Toronto or they are very different? This could help business individuals to make decision about the type of business they are going to run. It will also give valuable information about the type of businesses that are more likely to thrive in both cities, and the ones that are not suitable in each one of these cities. The best location for the business to open is another important parameter, which can be retrieved from this neighbourhoods clustering project.

New York City is the most populous city in the United States. New York City has been described as the cultural, financial, and media capital of the world, significantly influencing commerce, entertainment, research, technology, education, politics, tourism, art, fashion, and sports (Wikipedia). New York City is composed of five boroughs, each of which is a county of the State of New York. The five boroughs - Brooklyn, Queens, Manhattan, the Bronx, and Staten Island - were consolidated into a single city in 1898 (Wikipedia).

Toronto is the capital city of the Canadian province of Ontario. It is the most populous city in Canada and the fourth most populous city in North America. Toronto is an international centre of business, finance, arts, and culture, and is recognized as one of the most multicultural and cosmopolitan cities in the world. The six boroughs - Etobicoke, North York, Scarborough, York, East York and Toronto - became one city in 1998.

2 Data Acquisition and Preparation

In this section, the available data and the way it is going to be used to solve the problem will be described. Data acquisition includes acquiring, cleaning, and preparing the two datasets (New York City and Toronto). We are going to use Foursquare API to get the location and venue data for each city. In order to use the Foursquare API, we need to create a Foursquare developer account and then get the credentials (CLIENT_ID & CLIENT_SECRET). In addition to the credentials, we also need the latitude and longitude of the neighbourhoods. Having all these, we can make a URL to send a GET request to the Foursquare API to get any data related to a specific neighbourhood including all the venues in the proximity of the neighbourhood and any other details about it.

The ***neighbourhoods' data*** of these two cities and their ***latitude and longitude*** have been provided by the IBM course.

The ***venues data*** of each neighbourhood can be retrieved from Foursquare API. The retrieved data includes a specified number of venues (e.g. 100 venues) in a specific radius (e.g. 500 m) along with the venue category (bakery, coffee shop, park , …), venue latitude/longitude and etc.

2.1 Neighbourhood Data of New York City

The neighbourhood data for New York City, which is provided by IBM course coordinators, is a JSON file containing the information about the name of different neighbourhoods, the borough they are located in, their latitude and longitude, etc. A part of the JSON file for New York City has been shown in the Figure 2.1.

```
{'type': 'FeatureCollection',
 'totalFeatures': 306,
 'features': [{'type': 'Feature',
   'id': 'nyu_2451_34572.1',
   'geometry': {'type': 'Point',
    'coordinates': [-73.84720052054902, 40.89470517661]},
   'geometry_name': 'geom',
   'properties': {'name': 'Wakefield',
    'stacked': 1,
    'annoline1': 'Wakefield',
    'annoline2': None,
    'annoline3': None,
    'annoangle': 0.0,
    'borough': 'Bronx',
    'bbox': [-73.84720052054902,
     40.89470517661,
     -73.84720052054902,
     40.89470517661]}},
  {'type': 'Feature',
   'id': 'nyu_2451_34572.2',
   'geometry': {'type': 'Point',
    'coordinates': [-73.82993910812398, 40.87429419303012]},
   'geometry_name': 'geom',
   'properties': {'name': 'Co-op City',
    'stacked': 2,
    'annoline1': 'Co-op',
    'annoline2': 'City',
    'annoline3': None,
    'annoangle': 0.0,
    'borough': 'Bronx',
    'bbox': [-73.82993910812398,
```


Figure 2.1: New York City JSON file containing all its neighbourhoods

Now we need to store this JSON data into Pandas dataframe. The python code to do this has been shown in the Figure 2.2. Note that the *features* key (contained the list of NYC neighborhoods) of the JSON file has been stored in a variable called *NYC_Neighborhoods_data*. Figure 2.3 depicts the resulting Pandas dataframe (NYC_neighborhoods), which includes 306 neighborhoods (rows).

```
NYC_neighborhoods = pd.DataFrame(columns=['Borough', 'Neighborhood', 'Latitude', 'Longitude'])
```

```
for data in NYC_Neighbourhoods_data:
    borough = neighborhood_name = data['properties']['borough']
    neighborhood_name = data['properties']['name']

    neighborhood_latlon = data['geometry']['coordinates']
    neighborhood_lat = neighborhood_latlon[1]
    neighborhood_lon = neighborhood_latlon[0]

    NYC_neighborhoods = NYC_neighborhoods.append({'Borough': borough,
                                          'Neighborhood': neighborhood_name,
                                          'Latitude': neighborhood_lat,
                                          'Longitude': neighborhood_lon}, ignore_index=True)
```

Figure 2.2: The Python code to store New York neighborhoods data into a Pandas dataframe

	Borough	Neighborhood	Latitude	Longitude
0	Bronx	Wakefield	40.894705	-73.847201
1	Bronx	Co-op City	40.874294	-73.829939
2	Bronx	Eastchester	40.887556	-73.827806
3	Bronx	Fieldston	40.895437	-73.905643
4	Bronx	Riverdale	40.890834	-73.912585

Figure 2.3: The New York neighbourhood dataframe (first five rows)

Now we can plot the map of New York City using Folium package of Python along with all its neighborhoods (Figure 2.4).

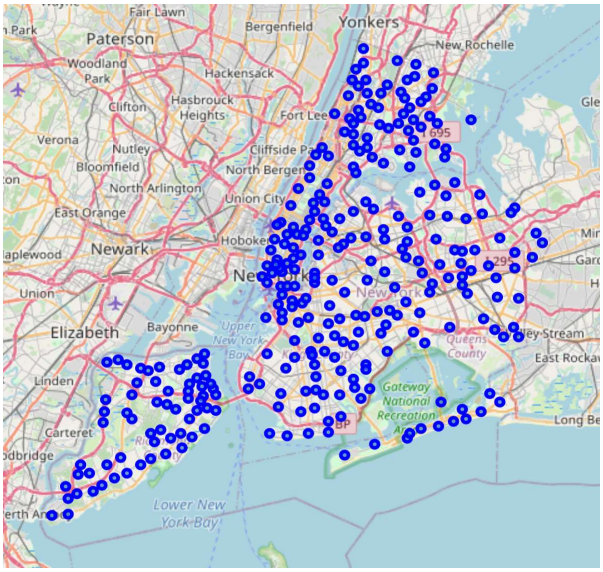

Figure 2.4: New York City map and its neighbourhoods (radius=500m)

2.2 Venues Data of New York City

The Figure 2.4 shows that the neighbourhoods are not evenly spaced and the area cover by some of them, using a radius of 500 meters, overlaps. A different radius for each neighbourhood results in a better venues search because that will avoid misrepresentation of the number of venues per neighbourhood caused by too large or low radius values.

To define the radius use with Foursquare API, it is necessary to find the closest neighbourhood for each neighbourhood, and to do so, we can use *haversine* formula, which

calculates the distance between two points on a sphere given a radius r, and a pair of coordinates expressed in latitude, ϕ, and longitude, λ, degrees:

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

After finding the distance of closest neighbourhood to each neighbourhood, we can add a new column called *Radius* to our dataframe (Figure 2.3), and assign the half of the distance calculated for each neighbourhood as its value. Here is the resulting dataframe:

	Borough	Neighborhood	Latitude	Longitude	Radius
0	Bronx	Wakefield	40.894705	-73.847201	565.191216
1	Bronx	Co-op City	40.874294	-73.829939	481.235842
2	Bronx	Eastchester	40.887556	-73.827806	742.734963
3	Bronx	Fieldston	40.895437	-73.905643	388.116468
4	Bronx	Riverdale	40.890834	-73.912585	388.116468

Figure 2.5: The New York neighbourhood dataframe with Radius column added

Now if we plot the map of neighbourhoods in New York City using different radius for each neighbourhood, not only overlapping is avoided but more area of the city is covered, consequently, more venues are retrieved:

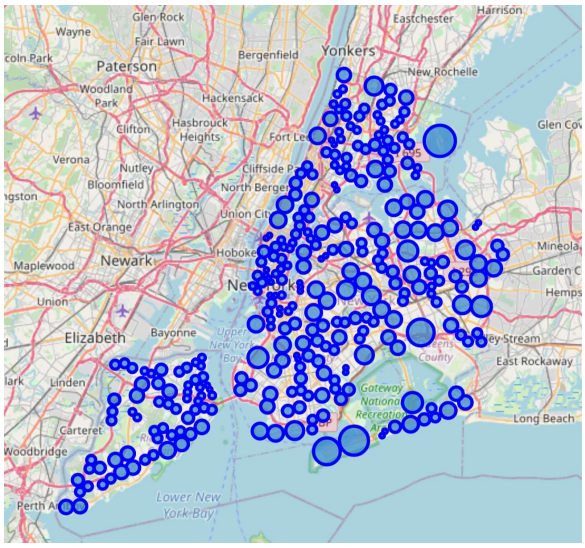

Figure 2.6: New York City map and its neighbourhoods (different radius for each neighbourhood)

As mentioned, the venues data of a neighbourhood can be retrieved from Foursquare API by making a GET request to the API. Each request requires a URL. Here is an example of a URL:

https://api.foursquare.com/v2/venues/search?&client_id=9823&client_secret=8565&v=20180605&ll=40.87655077879964,-73.91065965862981&radius=500&limit=100

where search shows the API endpoint, client_id and client_secret are credentials to access the API service and are obtained when registering a Foursquare developer account, v indicates the API version, ll indicates the latitude and longitude of the desired location, radius is the maximum distance in meters between the specified location and the retrieved venues, and limit is used to limit the number of returned results if necessary.

The response data is in JSON format with the following information: the name of the location (neighbourhood), location latitude, location longitude, the name of the nearby venues (100 venues in the example above), venue latitude, venue longitude, venue category, etc.

In order to get the 100 nearby venues for a specific radius from the neighbourhood, the following function in Python has been used:

```
def getNearbyVenues(names, latitudes, longitudes, radii):

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

        # create the API request URL
        url = f'https://api.foursquare.com/v2/venues/explore?&client_id={CLIENT_ID}&client_secret={CLIENT_SECRET}&v={VERSION}&ll=

        # 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 a list of neighborhoods (each neighborhoods data in a list), each list contains the nearby veunes for that neighborh
    #  ---> [[nearby venues for neighborhood 1], [nearby venues for neighborhood 2], ...]
    return(nearby_venues)
```

Figure 2.7: Python code to generate venues dataframe for all neighbourhoods in a city

The function takes four parameters: name of the neighbourhoods, their latitudes and longitudes, and the radius. The New York Neighborhoods data (Figure 2.5) has all these information for New York City. By passing it to the function, the function returns a dataframe

contains all the nearby venues for all neighbourhoods in New York City. Figure 2.8 indicates the returned dataframe (first five rows) for the New York City:

	Neighborhood	Neighborhood Latitude	Neighborhood Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
0	Wakefield	40.894705	-73.847201	Lollipops Gelato	40.894123	-73.845892	Dessert Shop
1	Wakefield	40.894705	-73.847201	Rite Aid	40.896649	-73.844846	Pharmacy
2	Wakefield	40.894705	-73.847201	Carvel Ice Cream	40.890487	-73.848568	Ice Cream Shop
3	Wakefield	40.894705	-73.847201	Walgreens	40.896528	-73.844700	Pharmacy
4	Wakefield	40.894705	-73.847201	Dunkin'	40.890459	-73.849089	Donut Shop

Figure 2.8: Venues dataframe for New York City

The dataframe contains 10118 rows; each row represents a specific venue for a neighbourhood. For each neighbourhood, different number of venues were returned; for example, 68 venues were returned for Manhattan Valley neighbourhood. It is worth to mention that there are 435 unique venue categories in the returned dataframe. It was decided to remove some of the venue categories such as Metro Station, Bus Stop, Bus Line, Train Station, Gas Station, Neighborhood, Field, Bridge, Office, Train, Platform, River and Church from the venues dataframe as they were not of analytical importance in this project. Doing so, the final number of unique venue categories and dataframe rows were reduced to 422 and 9915 respectively.

2.3 Neighbourhood Data of Toronto

Unlike New York, the neighborhood data for Toronto is not readily available on the internet. For the Toronto neighborhood data, a Wikipedia page exists that has all the information we need to explore and cluster the neighborhoods in Toronto. Using the Wikipedia page address *https://en.wikipedia.org/wiki/List_of_postal_codes_of_Canada:_M*, the neighbourhood data was obtained and then transformed into the following Pandas dataframe using Pandas read_html() functions, which reads HTML tables on a web page in a list of dataframes:

	Postal Code	Borough	Neighbourhood
0	M1A	Not assigned	Not assigned
1	M2A	Not assigned	Not assigned
2	M3A	North York	Parkwoods
3	M4A	North York	Victoria Village
4	M5A	Downtown Toronto	Regent Park, Harbourfront

Figure 2.9: Toronto's postal codes along with boroughs and neighbourhoods

As shown in the first five rows of the table, for 77 records in the dataframe, the value of 'Borough' variable is 'Not assigned'. These records were removed from the dataframe so the number rows in the dataframe were reduced from 180 to 103.

There is another dataset (csv file), provided by the course organizers, which contains information about the postal codes in Toronto and their latitudes and longitudes. The csv file was transformed into a Pandas dataframe using read_csv function:

	Postal Code	Latitude	Longitude
0	M1B	43.806686	-79.194353
1	M1C	43.784535	-79.160497
2	M1E	43.763573	-79.188711
3	M1G	43.770992	-79.216917
4	M1H	43.773136	-79.239476

Figure 2.10: Coordinates of Toronto's postal codes

The table contains 103 records (rows), each of which contains the postal code and its coordinates. In order to have a table of neighborhoods along with its coordinates, we need to combine these two tables using the following Python code. Please note that *toronto_neighborhoods* variable is the dataframe shown in Figure 2.9 and *torontoPotalCodeCoordinates* variable is the dataframe shown in Figure 2.10.

```
toronto_neighborhoods['Latitude'] = np.nan
toronto_neighborhoods['Longitude'] = np.nan
```

```
len1 = len(toronto_neighborhoods)
len2 = len(torontoPotalCodeCoordinates)
for i in range(len1):
    for j in range(len2):
        if (toronto_neighborhoods.loc[i,'Postal Code'] == torontoPotalCodeCoordinates.loc[j, 'Postal Code']):
            toronto_neighborhoods.loc[i,'Latitude'] = torontoPotalCodeCoordinates.loc[j, 'Latitude']
            toronto_neighborhoods.loc[i,'Longitude'] = torontoPotalCodeCoordinates.loc[j, 'Longitude']
```

Figure 2.11: Python code to combine neighbourhoods and coordinates tables

The combined dataframe, which is the combination of neighborhoods and coordinates tables has been shown in Figure 2.12:

	Postal Code	Borough	Neighbourhood	Latitude	Longitude
0	M3A	North York	Parkwoods	43.753259	-79.329656
1	M4A	North York	Victoria Village	43.725882	-79.315572
2	M5A	Downtown Toronto	Regent Park, Harbourfront	43.654260	-79.360636
3	M6A	North York	Lawrence Manor, Lawrence Heights	43.718518	-79.464763
4	M7A	Downtown Toronto	Queen's Park, Ontario Provincial Government	43.662301	-79.389494

Figure 2.12: Toronto neighbourhood dataframe (103 rows)

Examining the dataframe depicts that there are six records in which, the neighborhood and the borough are the same but the postal codes are different:

	Postal Code	Borough	Neighbourhood	Latitude	Longitude
7	M3B	North York	Don Mills	43.745906	-79.352188
13	M3C	North York	Don Mills	43.725900	-79.340923
40	M3K	North York	Downsview	43.737473	-79.464763
46	M3L	North York	Downsview	43.739015	-79.506944
53	M3M	North York	Downsview	43.728496	-79.495697
60	M3N	North York	Downsview	43.761631	-79.520999

Figure 2.13: rows with the same neighbourhood and borough but different postal codes

To deal with this, we can add the postal code to the end of neighborhood's name. Doing so the Figure 2.13 transforms to the Figure 2.14:

	Postal Code	Borough	Neighborhood	Latitude	Longitude
7	M3B	North York	Don Mills - M3B	43.745906	-79.352188
13	M3C	North York	Don Mills - M3C	43.725900	-79.340923
40	M3K	North York	Downsview - M3K	43.737473	-79.464763
46	M3L	North York	Downsview - M3L	43.739015	-79.506944
53	M3M	North York	Downsview - M3M	43.728496	-79.495697
60	M3N	North York	Downsview - M3N	43.761631	-79.520999

Figure 2.14: dataframe with unique neighborhood names

In order for the Toronto neighborhood dataframe to be compatible with New York neighborhood dataframe, we also changed the *Neighbourhood* column header to *Neighborhood,* so the resulting toronto neighborhood dataframe has been shown in Figure 2.15:

	Postal Code	Borough	Neighborhood	Latitude	Longitude
0	M3A	North York	Parkwoods	43.753259	-79.329656
1	M4A	North York	Victoria Village	43.725882	-79.315572
2	M5A	Downtown Toronto	Regent Park, Harbourfront	43.654260	-79.360636
3	M6A	North York	Lawrence Manor, Lawrence Heights	43.718518	-79.464763
4	M7A	Downtown Toronto	Queen's Park, Ontario Provincial Government	43.662301	-79.389494

Figure 2.15: Modified Toronto neighbourhood dataframe (103 rows)

As with New York neighbourhoods dataframe, in order to avoid overlapping of neighbourhoods and as a result, misrepresentation of the number of venues per neighbourhood, we need to calculate the radius for each neighbourhood using the formula shown in section 2.2 , and add the calculated radii to the Toronto neighbourhoods dataframe. Here is the resulting dataframe for Toronto neighbourhoods:

	Postal Code	Borough	Neighborhood	Latitude	Longitude	Radius
0	M3A	North York	Parkwoods	43.753259	-79.329656	992.961518
1	M4A	North York	Victoria Village	43.725882	-79.315572	1018.563373
2	M5A	Downtown Toronto	Regent Park, Harbourfront	43.654260	-79.360636	614.195007
3	M6A	North York	Lawrence Manor, Lawrence Heights	43.718518	-79.464763	934.471639
4	M7A	Downtown Toronto	Queen's Park, Ontario Provincial Government	43.662301	-79.389494	256.276246

Figure 2.16: The Toronto neighbourhood dataframe with Radius column added

Now we can plot the map of Toronto using Folium package of Python along with all its neighborhoods superimposed on top of it (Figure 2.17).

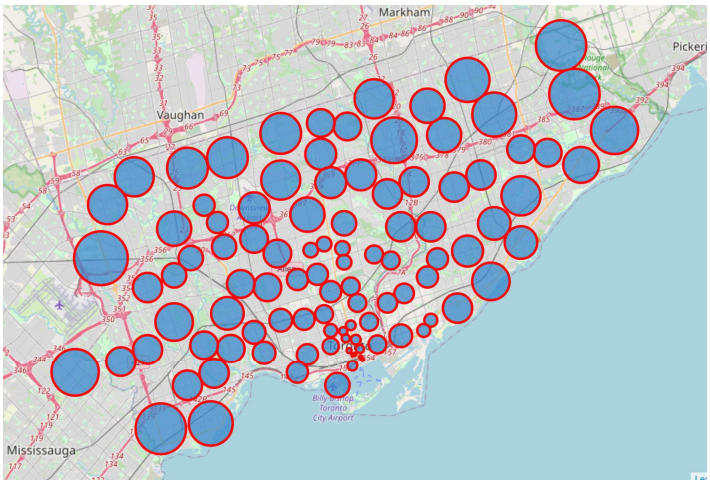

Figure 2.17: Toronto map and its neighbourhoods (different radius for each neighbourhood)

2.4 Venues Data of Toronto

In order to get the 100 venues near to each neighborhood for a specific radius from a neighbourhood, we passed the toronto neighborhood dataframe (Figure 2.16) to the getNearbyVenues function (Figure 2.7). The resulting venues dataframe returned from the function has been shown below:

	Neighborhood	Neighborhood Latitude	Neighborhood Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
0	Parkwoods	43.753259	-79.329656	Allwyn's Bakery	43.759840	-79.324719	Caribbean Restaurant
1	Parkwoods	43.753259	-79.329656	Tim Hortons	43.760668	-79.326368	Café
2	Parkwoods	43.753259	-79.329656	Brookbanks Park	43.751976	-79.332140	Park
3	Parkwoods	43.753259	-79.329656	Bruno's valu-mart	43.746143	-79.324630	Grocery Store
4	Parkwoods	43.753259	-79.329656	High Street Fish & Chips	43.745260	-79.324949	Fish & Chips Shop

Figure 2.18: Venues dataframe for Toronto

The dataframe contains 3280 rows; each row represents a specific venue for a neighbourhood. For each neighbourhood, different number of venues were returned; for example, 27 venues were returned for Thorncliffe Park neighbourhood. It is worth to mention that there are 326 unique venue categories in the returned dataframe. It was decided to remove some of the venue categories such as Metro Station, Bus Stop, Bus Line, Train Station, Gas Station, Neighborhood, Field, Bridge, Office, Train, Platform, River and Church from the venues dataframe as they were

not of analytical importance in this project. Doing so, the final number of unique venue categories and dataframe rows were reduced to 314 and 3207 respectively.

3 Exploratory Data Analysis

The main goal of this section is to better understand the two venues datasets generated for NYC and Toronto through different statistics and visualization tools.

3.1 Density Comparison

To compare the number of venues per distance, a new dataframe containing Neighborhood name, Number of Venues and Distance generated from neighbourhoods and venues dataframes of each city.

	Neighborhood	Number of Venues	Distance
0	Allerton	20	706.253
1	Annadale	12	1285.69
2	Arden Heights	5	1388.38
3	Arlington	4	939.661
4	Arrochar	12	835.184

	Neighborhood	Number of Venues	Distance
0	Agincourt	61	2958.68
1	Alderwood, Long Branch	66	3413.93
2	Bathurst Manor, Wilson Heights, Downsview North	37	2603.93
3	Bayview Village	13	1822.66
4	Bedford Park, Lawrence Manor East	46	2289.06

Figure 3.1: left: New York City, right: Toronto

The Distance column has been calculated as the Radius column of neighbourhood dataframes (Figures 2.5 and 2.16) times two. Here is some basic statistic details of these two dataframes:

```
count    302.000000
mean      32.831126
std       29.667615
min        1.000000
25%        9.250000
50%       22.000000
75%       46.750000
max      100.000000
Name: Number of Venues, dtype: float64
```

```
count    102.000000
mean      31.441176
std       24.340234
min        3.000000
25%       13.000000
50%       26.000000
75%       42.000000
max      100.000000
Name: Number of Venues, dtype: float64
```

Figure 3.2: Number of venues statistic details - left: New York City, right: Toronto

The minimum amount of venues present on a neighbourhood is 1 and 3 for NYC and Toronto respectively, and the maximum is 100, expected given the limit of venues set on the

request sent to the Foursquare API. Fifty percent of the venues for NYC and Toronto present 22 or less venues and 26 or less venues respectively.

Given that each neighbourhood has a different radius passed to the venues request, it's better to represent the venues per neighbourhood in terms of density, that's venues per area cover for each neighbourhood, in this case the area cover in the venues search defined by the distance to the closest neighbourhood; so density of each neighbourhood is calculated as follows:

Density = number of venues/distance (km)

Applying this formula, we get the following dataframes for two cities:

	Neighborhood	Number of Venues	Distance	Density
0	Allerton	20	706.253	28.0
1	Annadale	12	1285.69	9.0
2	Arden Heights	5	1388.38	3.0
3	Arlington	4	939.661	4.0
4	Arrochar	12	835.184	14.0

	Neighborhood	Number of Venues	Distance	Density
0	Agincourt	61	2958.68	20.0
1	Alderwood, Long Branch	66	3413.93	19.0
2	Bathurst Manor, Wilson Heights, Downsview North	37	2603.93	14.0
3	Bayview Village	13	1822.66	7.0
4	Bedford Park, Lawrence Manor East	46	2289.06	20.0

Figure 3.3: left: New York City, right: Toronto

The density statistics (Figure 3.4) depicts that the mean density in NYC is 31.5 while it is 21.99 for Toronto. That is expected given that Toronto has a low population density compared to NYC. The histograms (Figure 3.5) confirms the latter observation as 43% of the neighbourhoods in NYC presents a density between 0 and 17 but 50% of the neighbourhoods in Toronto are of a density between 0 and 14.

```
count    302.000000
mean      31.546358
std       31.220194
min        0.000000
25%       10.000000
50%       20.000000
75%       42.000000
max      176.000000
Name: Density, dtype: float64
```

```
count    102.000000
mean      21.990196
std       22.698844
min        1.000000
25%        7.000000
50%       14.000000
75%       28.500000
max      132.000000
Name: Density, dtype: float64
```

Figure 3.4: Density (number of venues per distance) statistic details - left: New York City, right: Toronto

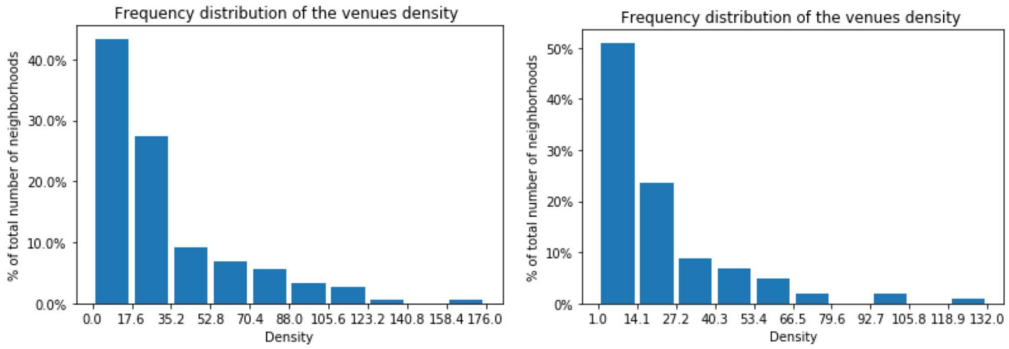

Figure 3.5: Density histogram - left New York City, right: Toronto

3.2 Most Common and Widespread Venue Categories

The main goal of this question is to answer the following two questions; first, which venue categories have more number of venues in New York City and Toronto, and second, which venue categories are existed in more number of neighbourhoods. To answer the first question, we need to count the number of venues returned from the Foursquare API for each venue category, and for the second question, we need to count the number neighbourhoods, which are of a specific venue category.

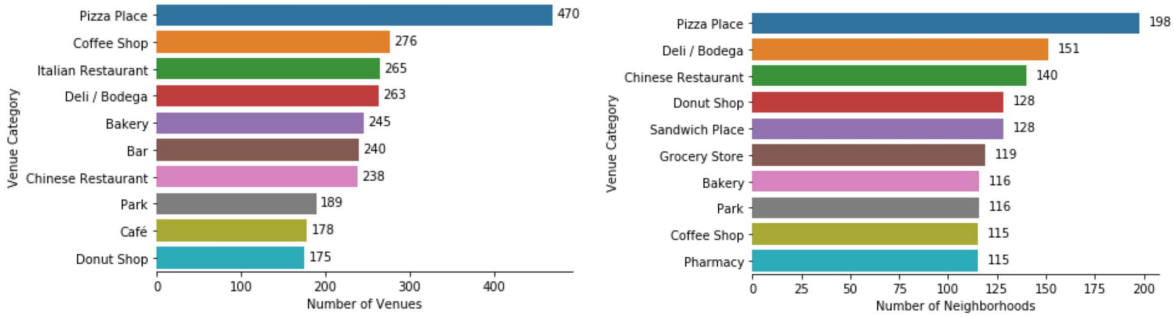

Figure 3.6: Most common (left) & widespread (right) venue categories in NYC

Figures 3.6 shows the bar plots of most common (left plot) and most widespread (right plot) venue categories in New City. As it is clear, the 'Pizza Place' is the most common and widespread venue category in New York. It is clear that the order of most of venue categories in Figure 3.6 (left) is different than that of Figure 3.6 (right).

Figures 3.7 shows the bar plots of most common (left plot) and most widespread (right plot) venue categories in Toronto. Interestingly, the first three venue categories ('Coffee Shop', 'Park', and 'Pizza Place') in both plots are the same. This means that you can find these three venue-category in most parts of Toronto.

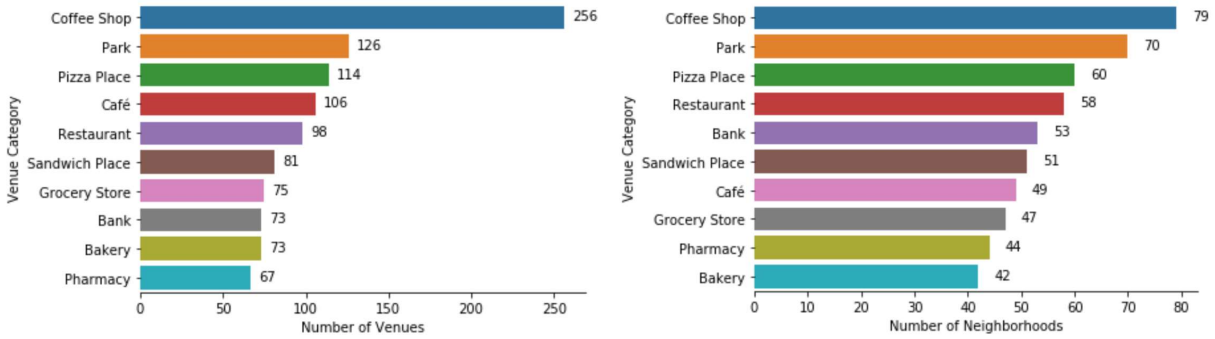

Figure 3.6: Most common (left) & widespread (right) venue categories in Toronto

4 Clustering of Neighbourhoods

The main goal of this section is to find similar neighbourhoods in both cities i.e. New York City and Toronto. This is called clustering of neighbourhoods. We are going to use KMeans algorithm of Scikit-Learn library in Python.

4.1 Data Pre-processing

To be able to apply KMeans clustering, we need to use venues dataframe of both cities (Figures 2.8 and 2.18), but first some pre-processing needs to be done on both dataframes. Here is the pre-processing steps:

1. Applying one-hot encoding on the 'Venue Category' feature of both dataframes.
2. Aggregating each dataframe
3. Combining both dataframes

In fact, we are going to cluster the neighbourhoods of both cities based on the similarity of 'Venue Category' of returned venues from Foursquare API in all neighbourhoods. Figure 4.1 depicts the Python code to do one-hot encoding on the venues dataframe of New York City:

```
# one hot encoding
nyc_onehot = pd.get_dummies(nyc_venues[['Venue Category']], prefix="", prefix_sep="")

# add neighborhood column back to dataframe
nyc_onehot['Neighborhood'] = nyc_venues['Neighborhood']

# move neighborhood column to the first column
fixed_columns = [nyc_onehot.columns[-1]] + list(nyc_onehot.columns[:-1])
nyc_onehot = nyc_onehot[fixed_columns]

nyc_onehot.head()
```

Figure 4.1: Applying one-hot encoding on NYC venues dataframe

In the Figure 4.1, the variable nyc_venues is the dataframe shown in Figure 2.8. the resulting one-hot encoded dataframes of NYC and Toronto have been show in the following Figures:

	Neighborhood	ATM	Accessories Store	Adult Boutique	Afghan Restaurant	African Restaurant	Airport Lounge	Airport Service	American Restaurant	Animal Shelter	Antique Shop	Aquarium	Arcade	Arepa Restaurant	Argentir Restaui
0	Wakefield	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	Wakefield	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	Wakefield	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	Wakefield	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	Wakefield	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 4.2: NYC one-hot encoded dataframe (9915 rows & 428 columns)

	Neighborhood	ATM	Accessories Store	Adult Boutique	Afghan Restaurant	Airport	Airport Food Court	Airport Gate	Airport Lounge	Airport Service	Airport Terminal	American Restaurant	Amphitheater	Animal Shelter	Antique Shop	G
0	Parkwoods	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	Parkwoods	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	Parkwoods	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	Parkwoods	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	Parkwoods	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 4.3: Toronto one-hot encoded dataframe (3207 rows & 314 columns)

In the above dataframes, if any venue category (columns) has the value of one, it means that specific neighbourhood has a venue with that category.

The second step is aggregating each dataframe i.e. each venue category compose how many percent of the returned venues for a specific neighbourhood. Figure 4.4 shows the Python code to aggregate the NYC venues dataframe (Figure 4.2). The same code used for Toronto venues dataframe:

```
nyc_grouped = nyc_onehot.groupby(['Neighborhood']).mean().reset_index()
```

Figure 4.4: Python code to aggregate the one-hot encoded dataframe (NYC)

The aggregated dataframes for both cities have been shown in the following Figures:

	Neighborhood	ATM	Accessories Store	Adult Boutique	Afghan Restaurant	African Restaurant	Airport Lounge	Airport Service	American Restaurant	Animal Shelter	Antique Shop	Aquarium	Arcade	Arepa Restaurant	Argentir Restaui
0	Allerton	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	
1	Annadale	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.083333	0.0	0.0	0.0	0.0	0.0	
2	Arden Heights	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	
3	Arlington	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.250000	0.0	0.0	0.0	0.0	0.0	
4	Arrochar	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	

Figure 4.5: Aggregated dataframe of NYC

	Neighborhood	ATM	Accessories Store	Adult Boutique	Afghan Restaurant	Airport	Airport Food Court	Airport Gate	Airport Lounge	Airport Service	Airport Terminal	American Restaurant	Amphitheater	Animal Shelter	Antique Shop	G
0	Agincourt	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
1	Alderwood, Long Branch	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
2	Bathurst Manor, Wilson Heights, Downsview North	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.027027	0.0	0.0	0.0	
3	Bayview Village	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
4	Bedford Park, Lawrence Manor East	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.021739	0.0	0.0	0.0	

Figure 4.6: Aggregated dataframe of Toronto

For example, based on Figure 4.4, 8.33% of the returned venues for the neighbourhood 'Annadale' in NYC are American restaurants.

The last step of pre-processing is to combine the dataframes shown in Figures 4.5 and 4.6; but before combining, in order to distinguish the neighbourhoods in NYC from Toronto, we are going to add '-nyc' and '-to' to the end of the neighbourhoods in NYC and Toronto respectively.

	Neighborhood	ATM	Accessories Store	Adult Boutique	Afghan Restaurant	African Restaurant	Airport Lounge	Airport Service	American Restaurant	Animal Shelter	Antique Shop	Aquarium	Arcade	Arepa Restaurant	Argentir Restau
0	Allerton-nyc	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	
1	Annadale-nyc	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.083333	0.0	0.0	0.0	0.0	0.0	
2	Arden Heights-nyc	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	
3	Arlington-nyc	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.250000	0.0	0.0	0.0	0.0	0.0	
4	Arrochar-nyc	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	

Figure 4.7: Modified aggregated dataframe of NYC (302 rows, 428 columns)

	Neighborhood	ATM	Accessories Store	Adult Boutique	Afghan Restaurant	Airport	Airport Food Court	Airport Gate	Airport Lounge	Airport Service	Airport Terminal	American Restaurant	Amphitheater	Animal Shelter	Antique Shop	G
0	Agincourt-to	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
1	Alderwood, Long Branch-to	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
2	Bathurst Manor, Wilson Heights, Downsview Nort...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.027027	0.0	0.0	0.0	
3	Bayview Village-to	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
4	Bedford Park, Lawrence Manor East-to	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.021739	0.0	0.0	0.0	

Figure 4.8: Modified aggregated dataframe of Toronto (102 rows, 314 columns)

You can find the resulting combined dataframe (Figure 4.10) using the Python code shown in Figure 4.9. Note that in the resulting dataframe, NaN values replaced with zero.

```
nyc_to_grouped = pd.concat([nyc_grouped_modified_hood,toronto_grouped_modified_hood], axis=0, ignore_index=True)
```

Figure 4.9: Combing the two dataframes (Figures 4.7 & 4.8)

	Neighborhood	ATM	Accessories Store	Adult Boutique	Afghan Restaurant	African Restaurant	Airport Lounge	Airport Service	American Restaurant	Animal Shelter	Antique Shop	Aquarium	Arcade	Arepa Restaurant	Arg Res
298	Woodlawn-nyc	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.022727	0.0	0.0	0.0	0.022727	0.000000	
299	Woodrow-nyc	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.000000	
300	Woodside-nyc	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.044776	0.0	0.0	0.0	0.000000	0.014925	
301	Yorkville-nyc	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.000000	
302	Agincourt-to	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.000000	
303	Alderwood, Long Branch-to	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.000000	

Figure 4.10: The combination of two aggregated dataframes

4.2 K-Means Clustering

Now that we obtained the combined dataframe of two cities (Figure 4.10), K-Means clustering algorithm can be applied on the dataframe; but in order to apply K-Means algorithm, first we need to specify the optimum number of clusters. To do so, we can use elbow method using the Python code shown in Figure 4.11. Note that the 'Neighborhood' column of dataframe of Figure 4.10 is dropped before applying the K-Means algorithm.

```
def findOptimumK(city_grouped_df):
    distortions = []
    cluster_numbers = range(1,10)
    city_grouped_df_clustering = city_grouped_df.drop('Neighborhood', 1)
    for k in cluster_numbers:
        kmeans = KMeans(init='k-means++',n_clusters=k, random_state=0).fit(city_grouped_df_clustering)
        distortions.append(kmeans.inertia_)
    plt.figure(figsize=(16,8))
    plt.plot(cluster_numbers, distortions, 'bx-')
    plt.xlabel('k')
    plt.ylabel('Distortion')
    plt.title('The Elbow Method showing the optimal k')
    plt.show()
```

Figure 4.11: The Python code to find the optimum number of clusters in K-Means

After passing the dataframe of Figure 4.10 to the function in Figure 4.11, the Figure 4.12 will be returned by the function. As it is clearly shown in Figure 4.12, the optimal value of the number of clusters is defined as 7.

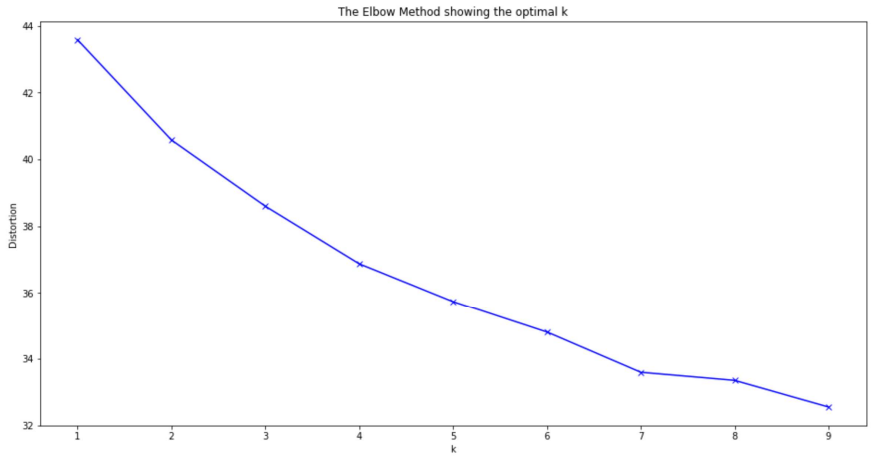

Figure 4.12: Elbow method showing the optimal k

Now we can apply the K-Means algorithm on the combined dataframe with the number of clusters of seven as shown in the code below

```
k_clusters = 7
nyc_to_grouped_clustering = nyc_to_grouped.drop('Neighborhood', 1)
kmeans = KMeans(init='k-means++',n_clusters=k_clusters, random_state=0).fit(nyc_to_grouped_clustering)
kmeans.labels_[0:10]
```

```
array([4, 0, 4, 0, 0, 0, 0, 0, 0, 4])
```

Figure 4.13: Python code to apply K-Means clustering with optimal k of 7

As shown in Figure 4.13, the output of K-Means method is seven clusters with the cluster labels of 0, 1, 2, 3, 4, 5, and 6. Each label shows the cluster number to which each row (neighbourhood) belongs. Figure 4.14 depicts the number of neighbourhoods in each cluster. Note that to obtain these numbers, we add the cluster labels as a new column (with the name of 'Cluster Labels') to the dataframe of Figure 4.10.

Cluster No.	Number of Neighbourhoods
0	297
1	14
2	4
3	1
4	85
5	1
6	2

Figure 4.14: the number of neighbourhoods in each cluster

5 Results

As shown in the previous section, K-Means algorithm produced seven clusters for the neighbourhoods of New York City and Toronto. In order to gain an insight into each cluster, we can find the five most common venue categories in each cluster. To do so, we need to pass the dataframe of Figure 4.10 and the cluster number to the function shown below.

```
def venueCatPercentForAClusterNumber(combined_cities_clusters, clusNum):
    rowsToBeremoved = []
    for rowIndex in range(len(combined_cities_clusters)):
        if (combined_cities_clusters.loc[rowIndex, 'Cluster Labels'] != clusNum-1):
            rowsToBeremoved.append(rowIndex)

    clusterDf = combined_cities_clusters.copy()
    clusterDf = clusterDf.drop(['Cluster Labels'], axis=1)
    clusterDf = clusterDf.drop(rowsToBeremoved)
    venuesNumberPerCat = mostCommonVenuesCategories(clusterDf)
    venuesPercentPerCat = venuesNumberPerCat.copy()
    venuesPercentPerCat['Number of Venues'] = venuesPercentPerCat['Number of Venues']/len(clusterDf) *100
    venuesPercentPerCat = venuesPercentPerCat.rename(columns={"Number of Venues": "% of Venues"})
    return venuesPercentPerCat
```

Figure 5.1: Python code to get the most common venue categories in each cluster

Figures 5.2 to 5.5 show the five most common venue categories in each cluster.

	Venue Category	% of Venues
0	Coffee Shop	3.65207
1	Park	2.86153
2	Deli / Bodega	2.71033
3	Pizza Place	2.59208
4	Italian Restaurant	2.55749

	Venue Category	% of Venues
0	Park	47.7976
1	Trail	7.61905
2	Coffee Shop	5
3	Beach	3.57143
4	Hotel	3.57143

Figure 5.2: Cluster one (left) – Cluster two (right)

	Venue Category	% of Venues
0	Deli / Bodega	70.8333
1	Market	12.5
2	Pharmacy	8.33333
3	Spanish Restaurant	8.33333
4	Plaza	0

	Venue Category	% of Venues
0	Construction & Landscaping	100
1	ATM	0
2	Pizza Place	0
3	Puerto Rican Restaurant	0
4	Public Art	0

Figure 5.3: Cluster three (left) – Cluster four (right)

	Venue Category	% of Venues
0	Pizza Place	13.3041
1	Pharmacy	4.07363
2	Bank	4.03491
3	Grocery Store	3.39211
4	Deli / Bodega	3.29925

	Venue Category	% of Venues
0	Boat or Ferry	100
1	ATM	0
2	Pizza Place	0
3	Puerto Rican Restaurant	0
4	Public Art	0

Figure 5.4: Cluster five (left) – Cluster six (right)

	Venue Category	% of Venues
0	Beach	100
1	ATM	0
2	Playground	0
3	Puerto Rican Restaurant	0
4	Public Art	0

Figure 5.5: Cluster seven

6 Discussion

As clearly shown in the above Figures, the distribution and contribution of venue categories are completely different in each cluster. Below you can find some of observations related to the clusters:

- In three out of seven clusters (4, 6, and 7), there is only one venue category; e.g. cluster 6 contains the venue category of 'Boat or Ferry' only.
- In both cluster 2 and 3, there is one venue category with a very high contribution compared to other venue categories in that cluster; e.g. 47.7 % of the venue categories of cluster two belongs to 'Park' category and in case of cluster 3, 70.8% of the venue categories of cluster three belongs to 'Deli' category.
- There is a uniform distribution in the most common venue categories of clusters 1 and 5.
- The venue category of 'Deli / Bodega' is the only venue category that exist in three different clusters of (1, 3 and 5). After that four venue categories of 'Coffee Shop', 'Pizza Place', 'Park' and 'Beach' have been appeared in two different clusters.

Figure 6.1 indicates the contribution (number of neighborhoods) of each city in all clusters. As it is obvious, there are four clusters (3, 4, 6, and 7) that Toronto has no contribution in them; but both cities have contribution in three clusters of 1, 2 and 5, which means NYC and Toronto are similar in that they share the same venue categories of 'Coffee Shop', 'Park' and 'Pizza Place'.

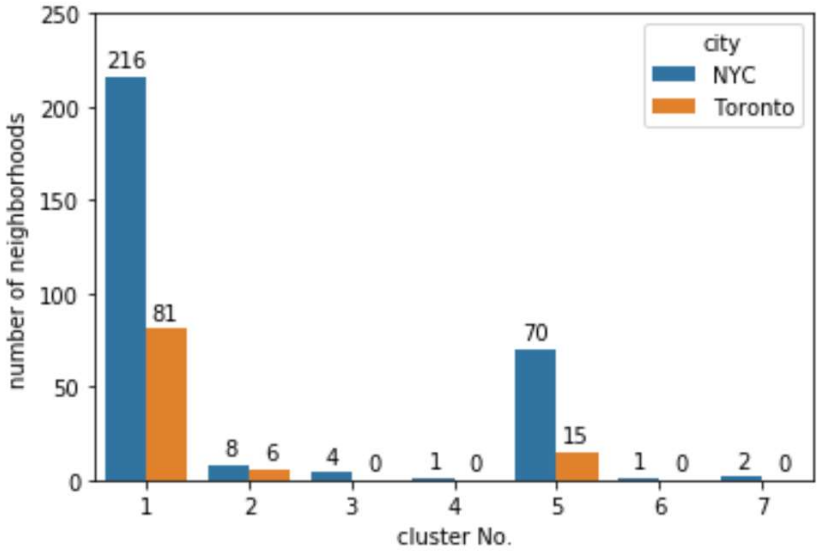

Figure 6.1: Contribution of NYC and Toronto in each cluster

7 Conclusion

In this project, the neighbourhoods in NYC and Toronto were clustered into different types of groups based on the venue category feature. Based on this clustering, some venue categories are most common in some clusters than the others.

This could help business individuals to make decision about the business type that can thrive in both cities, its best location (neighbourhoods), and the type of business that are suitable in each city. Tourists, travellers and new immigrants can also use these classified data to find a place to visit, stay or hangout.