Capstone Project Report

Applied Data Science Capstone by IBM/Coursera

Table of contents

1.Introduction: Business Problem

In this project we will try to find an optimal location for a restaurant. Specifically, this report will be targeted to stakeholders interested in opening an **seafood restaurant** in **Toronto**, ON, Canada.

Since there are lots of restaurants in Toronto we will try to detect **locations that are not already crowded with much more seafood restaurants**. We would also prefer locations **as close to rich and safe neighborhood as possible**, and we are also particularly interested in **areas with more population in the neighborhood**, assuming that first condition are met.

We will use our data science powers to generate a few most promissing neighborhoods based on this criteria. Advantages of each area will then be clearly expressed so that best possible final location can be chosen by stakeholders.

2.Data

Based on definition of our problem, factors that will influence our decision are:

- number of existing restaurants in the neighborhood (any type of restaurant)
- number of existing seafood restaurants in the neighborhood
- number of criminal case in the neighborhood.
- average income of the neighborhood.
- population of the neighborhood.

Following data sources will be needed to extract/generate the required information:

- ***To provide the stakeholders the necessary information we will combine Toronto's newest Census that contains Population, Average income per Neighborhood with Toronot's Neighborhoods shapefile and Foursquare API to collect competitors on the same neighborhoods.Toronto's Census data is publicly available at this website: [https://www.toronto.ca/city-government/data-research-maps/open-data/open-data-catalogue/#8c732154-5012-9afe-d0cd-ba3ffc813d5a (https://www.toronto.ca/city-government/data-research-maps/open-data/open-data-catalogue/#8c732154-5012-9afe-d0cd-ba3ffc813d5a)](https://www.toronto.ca/city-government/data-research-maps/open-data/open-data-catalogue/#8c732154-5012-9afe-d0cd-ba3ffc813d5a)***
- ***Toronto Neighborhoods' shapefile is publicly available at this website: [https://www.toronto.ca/city-government/data-research-maps/open-data/open-data-catalogue/#a45bd45a-ede8-730e-1abc-93105b2c439f (https://www.toronto.ca/city-government/data-research-maps/open-data/open-data-catalogue/#a45bd45a-ede8-730e-1abc-93105b2c439f)](https://www.toronto.ca/city-government/data-research-maps/open-data/open-data-catalogue/#a45bd45a-ede8-730e-1abc-93105b2c439f)***
- ***Also,we will get the Toronto Crime data from kaggle.com. we can download the data from this website: [https://www.kaggle.com/alincijov/toronto-crime-rate-per-neighbourhood (https://www.kaggle.com/alincijov/toronto-crime-rate-per-neighbourhood)](https://www.kaggle.com/alincijov/toronto-crime-rate-per-neighbourhood)***

Download and Explore Dataset

In [1]:

```
import numpy as np # library to handle data in a vectorized manner
import pandas as pd # library for data analsysis
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
from geopy.geocoders import Nominatim # convert an address into latitude and longitu
import urllib.request as req
from pandas.io.json import json_normalize # tranform JSON file into a pandas datafra
# Matplotlib and associated plotting modules
import matplotlib.cm as cm
import matplotlib.colors as colors
# import k-means from clustering stage
from sklearn.cluster import KMeans
import folium # map rendering library

print('Libraries imported.')
```

```
Libraries imported.
```

Prepare the dataset, we can directly download **Toronto Population_data.csv** from website, and should register kaggle first then download **Neighbourhood_Crime_Rates_(Boundary_File)_.csv**

In [2]:

```
csv_path='https://ckan0.cf.opendata.inter.prod-toronto.ca/download_resource/ef0239b1
req.urlretrieve(csv_path, "Toronto Population_data.csv")
```

Out[2]:

```
('Toronto Population_data.csv', <http.client.HTTPMessage at 0x125fe5fd
0>)
```

In [43]:

```
geourl='https://ckan0.cf.opendata.inter.prod-toronto.ca/download_resource/a083c865-6
req.urlretrieve(csv_path, "Toronto_Neighbourhoods.geojson")
```

Out[43]:

```
('Toronto_Neighbourhoods.geojson', <http.client.HTTPMessage at 0x126b8
0fd0>)
```

In [2]:

```
df_population=pd.read_csv('Toronto Population_data.csv')
df_population.head()
```

Out[2]:

	_id	Category	Topic	Data Source	Characteristic	City of Toronto	Agincourt North	Agincou Sout Malve We
0	1	Neighbourhood Information	Neighbourhood Information	City of Toronto	Neighbourhood Number	NaN	129	1
1	2	Neighbourhood Information	Neighbourhood Information	City of Toronto	TSNS2020 Designation	NaN	No Designation	I Designati
2	3	Population	Population and dwellings	Census Profile 98-316-X2016001	Population, 2016	2,731,571	29,113	23,7
3	4	Population	Population and dwellings	Census Profile 98-316-X2016001	Population, 2011	2,615,060	30,279	21,9
4	5	Population	Population and dwellings	Census Profile 98-316-X2016001	Population Change 2011-2016	4.50%	-3.90%	8.00

In [3]:

```
print(df_population.info())
print(df_population.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2383 entries, 0 to 2382
Columns: 146 entries, _id to Yorkdale-Glen Park
dtypes: int64(1), object(145)
memory usage: 2.7+ MB
None
(2383, 146)
```

Collecting neighborhoods names

In [13]:

```
Neighbourhoods = list(df_population.columns.values)
Neighbourhoods = Neighbourhoods[6:]
Neighbourhoods
```

Out[13]:

```
['Agincourt North',
 'Agincourt South-Malvern West',
 'Alderwood',
 'Annex',
 'Banbury-Don Mills',
 'Bathurst Manor',
 'Bay Street Corridor',
 'Bayview Village',
 'Bayview Woods-Steeles',
 'Bedford Park-Nortown',
 'Beechborough-Greenbrook',
 'Bendale',
 'Birchcliffe-Cliffside',
 'Black Creek',
 'Blake-Jones',
 'Briar Hill-Belgravia',
 'Bridle Path-Sunnybrook-York Mills',
 'Broadview North',
 'Brookhaven-Amesbury',
 'Cabbagetown-South St. James Town',
 'Caledonia-Fairbank',
 'Casa Loma',
 'Centennial Scarborough',
 'Church-Yonge Corridor',
 'Clairlea-Birchmount',
 'Clanton Park',
 'Cliffcrest',
 'Corso Italia-Davenport',
 'Danforth',
 'Danforth East York',
 'Don Valley Village',
 'Dorset Park',
 'Dovercourt-Wallace Emerson-Junction',
 'Downsview-Roding-CFB',
 'Dufferin Grove',
 'East End-Danforth',
 'Edenbridge-Humber Valley',
 'Eglinton East',
 'Elms-Old Rexdale',
 'Englemount-Lawrence',
 'Eringate-Centennial-West Deane',
 'Etobicoke West Mall',
 'Flemingdon Park',
 'Forest Hill North',
 'Forest Hill South',
 'Glenfield-Jane Heights',
 'Greenwood-Coxwell',
 'Guildwood',
 'Henry Farm',
 'High Park North',
 'High Park-Swansea',
 'Highland Creek',
 'Hillcrest Village',
```

```
'Humber Heights-Westmount',
'Humber Summit',
'Humbermede',
'Humewood-Cedarvale',
'Ionview',
'Islington-City Centre West',
'Junction Area',
'Keelesdale-Eglinton West',
'Kennedy Park',
'Kensington-Chinatown',
'Kingsview Village-The Westway',
'Kingsway South',
'Lambton Baby Point',
"L'Amoreaux",
'Lansing-Westgate',
'Lawrence Park North',
'Lawrence Park South',
'Leaside-Bennington',
'Little Portugal',
'Long Branch',
'Malvern',
'Maple Leaf',
'Markland Wood',
'Milliken',
'Mimico (includes Humber Bay Shores)',
'Morningside',
'Moss Park',
'Mount Dennis',
'Mount Olive-Silverstone-Jamestown',
'Mount Pleasant East',
'Mount Pleasant West',
'New Toronto',
'Newtonbrook East',
'Newtonbrook West',
'Niagara',
'North Riverdale',
'North St. James Town',
'Oakridge',
'Oakwood Village',
"O'Connor-Parkview",
'Old East York',
'Palmerston-Little Italy',
'Parkwoods-Donalda',
'Pelmo Park-Humberlea',
'Playter Estates-Danforth',
'Pleasant View',
'Princess-Rosethorn',
'Regent Park',
'Rexdale-Kipling',
'Rockcliffe-Smythe',
'Roncesvalles',
'Rosedale-Moore Park',
'Rouge',
'Runnymede-Bloor West Village',
'Rustic',
'Scarborough Village',
'South Parkdale',
'South Riverdale',
'St.Andrew-Windfields',
'Steeles',
'Stonegate-Queensway',
```

```
 "Tam O'Shanter-Sullivan",
 'Taylor-Massey',
 'The Beaches',
 'Thistletown-Beaumond Heights',
 'Thorncliffe Park',
 'Trinity-Bellwoods',
 'University',
 'Victoria Village',
 'Waterfront Communities-The Island',
 'West Hill',
 'West Humber-Clairville',
 'Westminster-Branson',
 'Weston',
 'Weston-Pelham Park',
 'Wexford/Maryvale',
 'Willowdale East',
 'Willowdale West',
 'Willowridge-Martingrove-Richview',
 'Woburn',
 'Woodbine Corridor',
 'Woodbine-Lumsden',
 'Wychwood',
 'Yonge-Eglinton',
 'Yonge-St.Clair',
 'York University Heights',
 'Yorkdale-Glen Park']
```

Create a new dataset for the data we need (Population and Income data of each neighborhood)

In [15]:

```
df_Toronto=df_population.loc[[2,2272]][Neighbourhoods].T
df_Toronto.columns=['Population','Income']
# covert format of the data from string to float
df_Toronto['Population']=df_Toronto['Population'].str.replace(',','').astype(float)
df_Toronto['Income']=df_Toronto['Income'].str.replace(',','').astype(float)

df_Toronto.insert(0,'Neighborhood',df_Toronto.index.values)
df_Toronto.reset_index(drop=True, inplace=True)
df_Toronto.head()
```

Out[15]:

	Neighborhood	Population	Income
0	Agincourt North	29113.0	30414.0
1	Agincourt South-Malvern West	23757.0	31825.0
2	Alderwood	12054.0	47709.0
3	Annex	30526.0	112766.0
4	Banbury-Don Mills	27695.0	67757.0

Explore the Crime Rate data

In [16]:

```
df_crime_origin=pd.read_csv('Neighbourhood_Crime_Rates_(Boundary_File)_.csv')
df_crime_origin.head()
```

Out[16]:

	OBJECTID	Neighbourhood	Hood_ID	Population	Assault_2014	Assault_2015	Assault_2016
0	1	Yonge-St.Clair	97	12528	20	29	39
1	2	York University Heights	27	27593	271	296	361
2	3	Lansing-Westgate	38	16164	44	80	68
3	4	Yorkdale-Glen Park	31	14804	106	136	174
4	5	Stonegate-Queensway	16	25051	88	71	76

In [17]:

```
df_crime_origin.shape
```

Out[17]:

```
(140, 60)
```

Get the newset crime data columns with 2019,and we will use the summary of all type crimes as a crime factor.

In [18]:

```
df_crime_origin=df_crime_origin.rename(columns={'Neighbourhood':'Neighborhood'})
cols = [col for col in df_crime_origin.columns if '2019' in col and 'Rate' not in cc
cols.insert(0,'Neighborhood')
cols.insert(1,'Hood_ID')
print(cols)
df_crime=df_crime_origin[cols]
df_crime['Crime_total']=df_crime['Assault_2019']+df_crime['AutoTheft_2019']+df_crime
df_crime.drop(columns=['Assault_2019','AutoTheft_2019','BreakandEnter_2019','Homicic
df_crime.head()
```

```
['Neighborhood', 'Hood_ID', 'Assault_2019', 'AutoTheft_2019', 'Breakan
dEnter_2019', 'Homicide_2019', 'Robbery_2019', 'TheftOver_2019']

/Users/huangbo/opt/anaconda3/lib/python3.7/site-packages/ipykernel_lau
ncher.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (h
ttp://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
turning-a-view-versus-a-copy)
  import sys
/Users/huangbo/opt/anaconda3/lib/python3.7/site-packages/pandas/core/f
rame.py:4102: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (h
ttp://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
turning-a-view-versus-a-copy)
  errors=errors,
```

Out[18]:

	Neighborhood	Hood_ID	Crime_total
0	Yonge-St.Clair	97	81
1	York University Heights	27	729
2	Lansing-Westgate	38	165
3	Yorkdale-Glen Park	31	426
4	Stonegate-Queensway	16	206

Combine **population data** and **crime data**

In [19]:

```
df_Toronto_combine=pd.merge(df_Toronto,df_crime,how='left',on='Neighborhood')
df_Toronto_combine.dropna(inplace=True)
df_Toronto_combine.head()
```

Out[19]:

	Neighborhood	Population	Income	Hood_ID	Crime_total
0	Agincourt North	29113.0	30414.0	129.0	214.0
1	Agincourt South-Malvern West	23757.0	31825.0	128.0	329.0
2	Alderwood	12054.0	47709.0	20.0	88.0
3	Annex	30526.0	112766.0	95.0	604.0
4	Banbury-Don Mills	27695.0	67757.0	42.0	221.0

In [65]:

```
df_Toronto_combine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 110 entries, 0 to 139
Data columns (total 9 columns):
Neighborhood    110 non-null object
Population      110 non-null float64
Income          110 non-null float64
Hood_ID         110 non-null float64
Crime_total     110 non-null float64
Address         110 non-null object
Coordinates     110 non-null object
Latitude        110 non-null float64
Longitude       110 non-null float64
dtypes: float64(6), object(3)
memory usage: 8.6+ KB
```

In [20]:

```
# add address for getting location data
df_Toronto_combine['Address'] = df_Toronto_combine['Neighborhood'] + ', Toronto'
geolocator = Nominatim(user_agent="capstone-application")
df_Toronto_combine['Coordinates'] = df_Toronto_combine['Address'].apply(geolocator.g
df_Toronto_combine['Latitude']=df_Toronto_combine['Coordinates'].apply(lambda x: x.l
df_Toronto_combine['Longitude']=df_Toronto_combine['Coordinates'].apply(lambda x: x.
df_Toronto_combine.head()
```

Out[20]:

	Neighborhood	Population	Income	Hood_ID	Crime_total	Address	Coordinates	Latitud
0	Agincourt North	29113.0	30414.0	129.0	214.0	Agincourt North, Toronto	(Agincourt North, Scarborough North, Scarborou...	43.80803
1	Agincourt South-Malvern West	23757.0	31825.0	128.0	329.0	Agincourt South-Malvern West, Toronto	(Toronto Fire Station 243, Sheppard Avenue Eas...	43.78923
2	Alderwood	12054.0	47709.0	20.0	88.0	Alderwood, Toronto	(Alderwood, Etobicoke—Lakeshore, Etobicoke, To...	43.60171
3	Annex	30526.0	112766.0	95.0	604.0	Annex, Toronto	(The Annex, University—Rosedale, Toronto, Gold...	43.67033
4	Banbury-Don Mills	27695.0	67757.0	42.0	221.0	Banbury-Don Mills, Toronto	(Banbury Road, Don Valley West, North York, To...	43.73480

In [21]:

```
print(df_Toronto_combine.info())
print("remove missing values")
df_Toronto_combine.dropna(inplace=True)
print(df_Toronto_combine.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 136 entries, 0 to 139
Data columns (total 9 columns):
Neighborhood    136 non-null object
Population      136 non-null float64
Income          136 non-null float64
Hood_ID         136 non-null float64
Crime_total     136 non-null float64
Address         136 non-null object
Coordinates     110 non-null object
Latitude        110 non-null float64
Longitude       110 non-null float64
dtypes: float64(6), object(3)
memory usage: 10.6+ KB
None
remove missing values
<class 'pandas.core.frame.DataFrame'>
Int64Index: 110 entries, 0 to 139
Data columns (total 9 columns):
Neighborhood    110 non-null object
Population      110 non-null float64
Income          110 non-null float64
Hood_ID         110 non-null float64
Crime_total     110 non-null float64
Address         110 non-null object
Coordinates     110 non-null object
Latitude        110 non-null float64
Longitude       110 non-null float64
dtypes: float64(6), object(3)
memory usage: 8.6+ KB
None
```

Visualize the data

In [22]:

```
address = 'Toronto'
location = geolocator.geocode(address)
latitude = location.latitude
longitude = location.longitude
print('The geograpical coordinate of Toronto are {}, {}.'.format(latitude, longitude
```

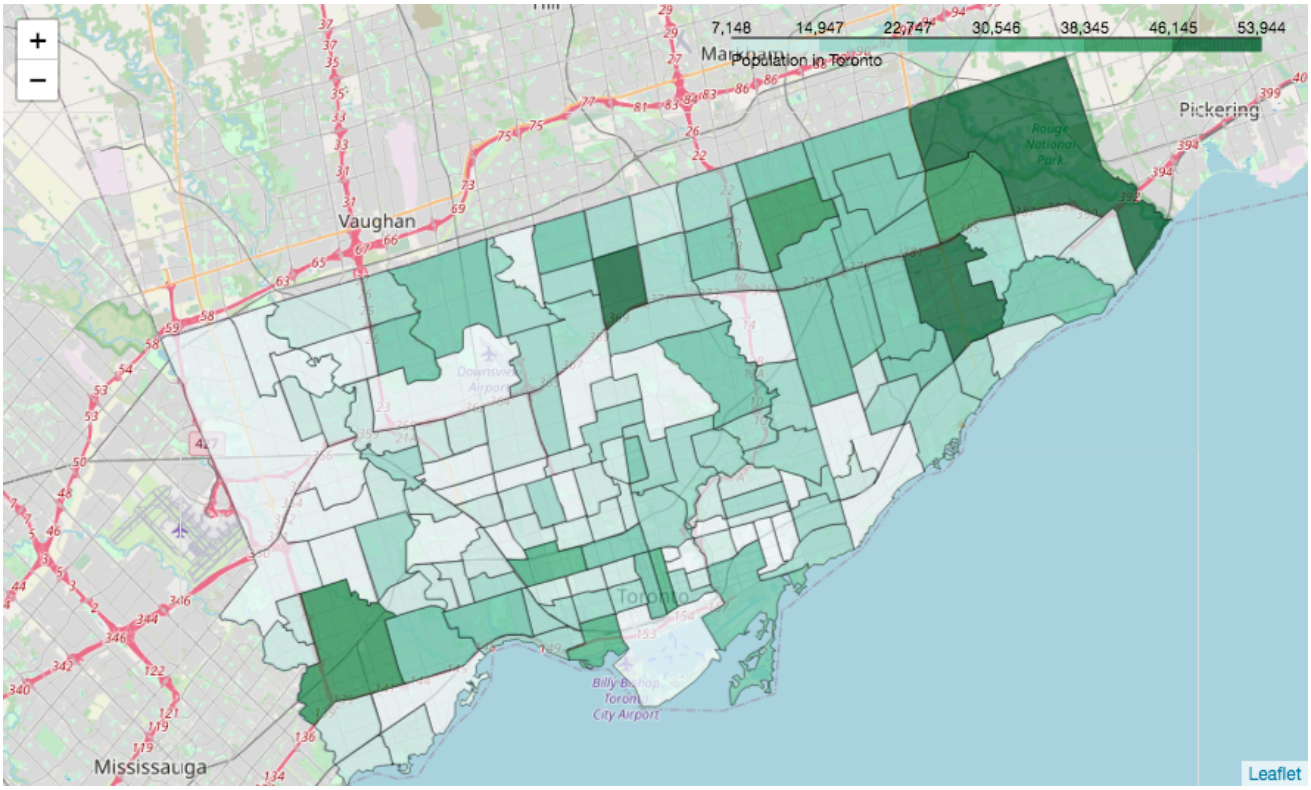
```
The geograpical coordinate of Toronto are 43.6534817, -79.3839347.
```

(1)Population Map in Toronto

In [24]:

```
toronto_geo = r'Toronto_Neighbourhoods.geojson' # geojson file
# create a plain world map
toronto_populaton_map = folium.Map(location=[latitude+0.075,longitude], zoom_start=1
# generate choropleth map
toronto_populaton_map.choropleth(
    geo_data=toronto_geo,
    data=df_Toronto_combine,
    columns=['Hood_ID', 'Population'],
    key_on='feature.properties.AREA_SHORT_CODE',
    fill_color='BuGn',
    fill_opacity=0.7,
    line_opacity=0.5,
    legend_name='Population in Toronto'
)
# display map
toronto_populaton_map
```

Out[24]:

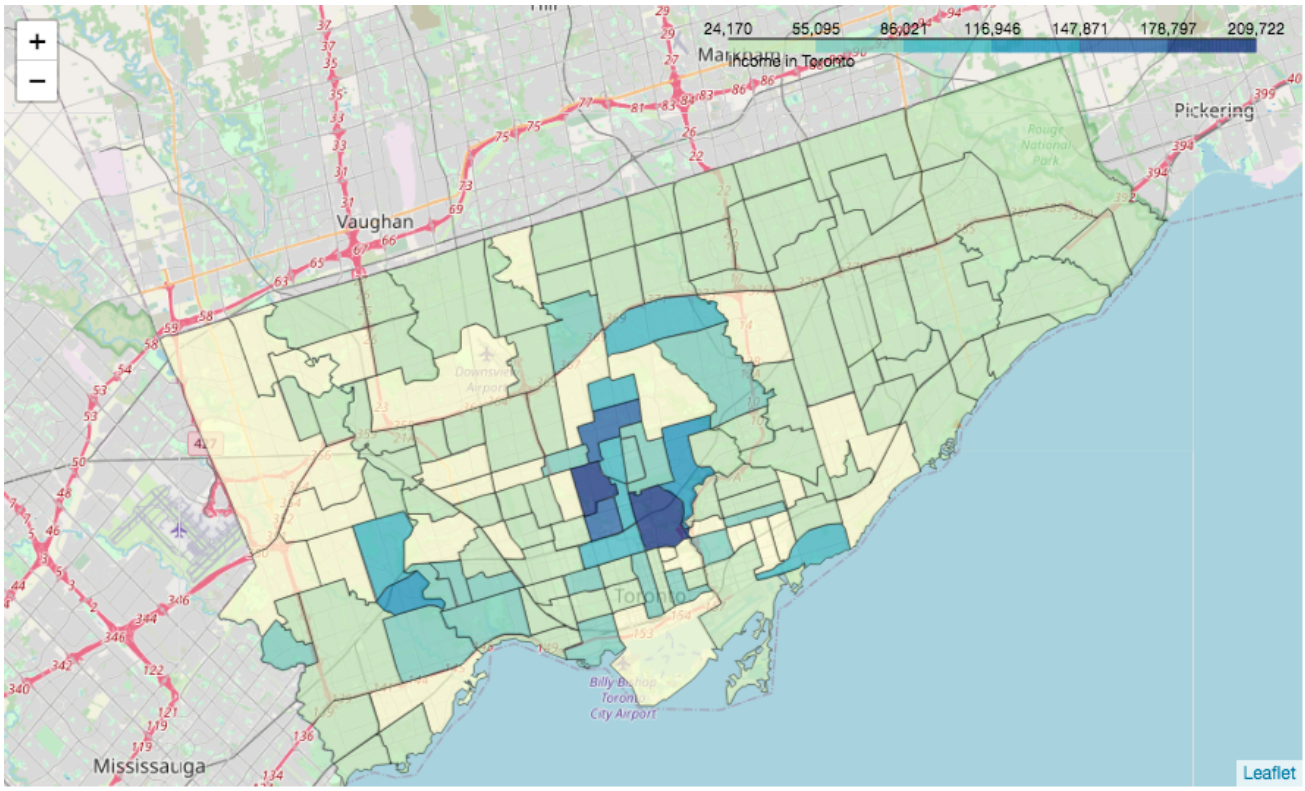


(2)Income Map in Toronto

In [26]:

```
toronto_geo = r'Toronto_Neighbourhoods.geojson' # geojson file
# create a plain world map
toronto_income_map = folium.Map(location=[latitude+0.075,longitude], zoom_start=11)
# generate choropleth map
toronto_income_map.choropleth(
    geo_data=toronto_geo,
    data=df_Toronto_combine,
    columns=['Hood_ID', 'Income'],
    key_on='feature.properties.AREA_SHORT_CODE',
    fill_color='YlGnBu',
    fill_opacity=0.7,
    line_opacity=0.5,
    legend_name='Income in Toronto'
)
# display map
toronto_income_map
```

Out[26]:

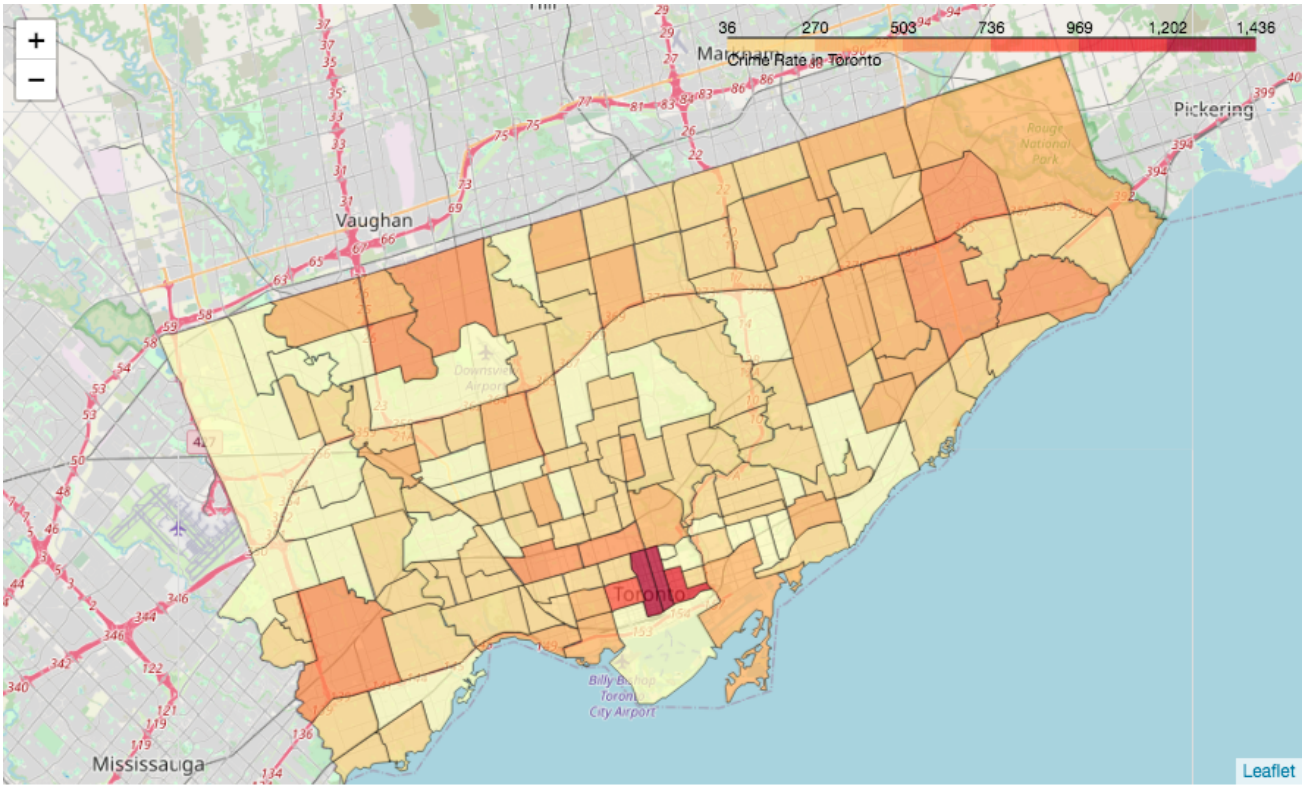


(3)Crime Rate Map in Toronto

In [27]:

```
toronto_geo = r'Toronto_Neighbourhoods.geojson' # geojson file
# create a plain world map
toronto_crime_map = folium.Map(location=[latitude+0.075,longitude], zoom_start=11)
# generate choropleth map
toronto_crime_map.choropleth(
    geo_data=toronto_geo,
    data=df_Toronto_combine,
    columns=['Hood_ID', 'Crime_total'],
    key_on='feature.properties.AREA_SHORT_CODE',
    fill_color='YlOrRd',
    fill_opacity=0.7,
    line_opacity=0.5,
    legend_name='Crime Rate in Toronto'
)
# display map
toronto_crime_map
```

Out[27]:



Foursquare Data

Now that we have our location candidates, let's use Foursquare API to get info on restaurants in each neighborhood.

We're interested in venues in 'food' category, but only those that are proper restaurants - coffe shops, pizza places, bakeries etc. are not direct competitors so we don't care about those. So we will include in out list only venues that have 'restaurant' in category name, and we'll make sure to detect and include all the subcategories of specific 'Seafood restaurant' category, as we need info on seafood restaurants in the neighborhood.

In [64]:

```
import requests
CLIENT_ID = 'My Foursquare ID' # My Foursquare ID
CLIENT_SECRET = 'My Foursquare Secret' # My Foursquare Secret
VERSION = '20180604'
radius = 500
LIMIT = 100
print('Your credentails:')
print('CLIENT_ID: ' + CLIENT_ID)
print('CLIENT_SECRET:' + CLIENT_SECRET)
```

```
Your credentails:
CLIENT_ID: My Foursquare ID
CLIENT_SECRET:My Foursquare Secret
```

In [29]:

```
def getNearbyVenues(names, latitudes, longitudes, radius=2000):

    venues_list=[]
    print('Start get nearby Venues')
    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_sec
            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 ver
    nearby_venues.columns = ['Neighborhood',
                  'Neighborhood Latitude',
                  'Neighborhood Longitude',
                  'Venue',
                  'Venue Latitude',
                  'Venue Longitude',
                  'Venue Category']
    print('done')
    return(nearby_venues)
```

In [30]:

```
Toronto_venues = getNearbyVenues(names=df_Toronto_combine['Neighborhood'],
                                 latitudes=df_Toronto_combine['Latitude'],
                                 longitudes=df_Toronto_combine['Longitude']
                                )
```

```
Start get nearby Venues
done
```

In [31]:

```
Toronto_venues.head()
```

Out[31]:

	Neighborhood	Neighborhood Latitude	Neighborhood Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
0	Agincourt North	43.808038	-79.266439	Menchie's	43.808338	-79.268288	Frozen Yogurt Shop
1	Agincourt North	43.808038	-79.266439	Fahmee Bakery & Jamaican Foods	43.810170	-79.280113	Caribbean Restaurant
2	Agincourt North	43.808038	-79.266439	Saravanaa Bhavan South Indian Restaurant	43.810117	-79.269275	Indian Restaurant
3	Agincourt North	43.808038	-79.266439	Shoppers Drug Mart	43.808894	-79.269854	Pharmacy
4	Agincourt North	43.808038	-79.266439	Samosa King - Embassy Restaurant	43.810152	-79.257316	Indian Restaurant

In [32]:

```
print('Total Venues')
Toronto_venues.shape
```

```
Total Venues
```

Out[32]:

```
(8899, 7)
```

In [33]:

```
print('Restaurant Venues')
Toronto_Restaurant_venues=Toronto_venues[Toronto_venues['Venue Category'].str.contai
Toronto_Restaurant_venues.shape
```

```
Restaurant Venues
```

Out[33]:

```
(2312, 7)
```

In [34]:

```
print('There are {} uniques restaurant categories.'.format(len(Toronto_Restaurant_ve
```

```
There are 66 uniques restaurant categories.
```

Looking good. So now we have all the restaurants for each neighborhood, and we know which ones are seafood restaurants! We also know which restaurants exactly are in vicinity of every neighborhood candidate center.

This concludes the data gathering phase - we're now ready to use this data for analysis to produce the report on optimal locations for a new seafood restaurant!

3.Methodology

In this project we will direct our efforts on detecting areas of Toronto that have low restaurant density, particularly those with low number of Seafood restaurants.

In first step we have collected the required data: location and type (category) of every restaurant within 500m from each neighborhood center. We have also identified seafood restaurants (according to Foursquare categorization).

Second step in our analysis will be calculation and exploration of 'restaurant density' across different areas of Toronto - we will use choropleth maps to identify a few promising areas with low number of restaurants in general (and no seafood restaurants in vicinity) .

In third and final step we will use the data to create clusters of locations , and try to find out a cluster that meet some basic requirements established in discussion with stakeholders: we will take into consideration locations with low number of restaurants in the neigherhood, and we want locations without seafood restaurant in radius of 500 meters, we also want locations at lower crime rate and high income neighborhood. We will present map of all such locations but also create clusters (using k-means clustering) of those locations to identify general zones / neighborhoods / addresses which should be a starting point for final 'street level' exploration and search for optimal venue location by stakeholders.

4.Analysis

Let's perform some basic explanatory data analysis and derive some additional info from our raw data. First let's count the number of restaurants in every area candidate:

In [35]:

```
# # one hot encoding
Toronto_onehot = Toronto_Restaurant_venues.join(pd.get_dummies(Toronto_Restaurant_ve
Toronto_onehot.drop(columns=['Neighborhood Latitude','Neighborhood Longitude','Venue
Toronto_onehot.head()
```

Out[35]:

Swiss Restaurant	Syrian Restaurant	Szechuan Restaurant	Taiwanese Restaurant	Tapas Restaurant	Thai Restaurant	Theme Restaurant	Tibetan Restaurant
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

In [36]:

```
Toronto_grouped = Toronto_onehot.groupby('Neighborhood').agg(np.sum)
Toronto_grouped.head()
```

Out[36]:

Swiss Restaurant	Syrian Restaurant	Szechuan Restaurant	Taiwanese Restaurant	Tapas Restaurant	Thai Restaurant	Theme Restaurant	Tibetan Restaurant
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0
0	0	0	0	0	0	0	0

In [37]:

```
Toronto_grouped['Restaurants_sum'] = Toronto_grouped.apply(lambda x: x.sum(), axis=1
```

In [38]:

```
df_restaurant_venues=Toronto_grouped[['Seafood Restaurant','Restaurants_sum']]
df_restaurant_venues.head()
```

Out[38]:

Neighborhood	Seafood Restaurant	Restaurants_sum
Agincourt North	0	22
Agincourt South-Malvern West	1	37
Alderwood	3	23
Annex	0	30
Banbury-Don Mills	0	12

Now, we have Seafood Restaurant's numbers and total numbers of all restaurants with each neighborhood. We will combine the population,income,crimeRate,totalRestaurantsNum,SeafoodRestaurantsNum of each neighborhood in a dataset.

In [39]:

```
df_Toronto_combine_venue=pd.merge(df_Toronto_combine,df_restaurant_venues,how='left
df_Toronto_combine_venue.head()
```

Out[39]:

	Neighborhood	Population	Income	Hood_ID	Crime_total	Address	Coordinates	Latitud
0	Agincourt North	29113.0	30414.0	129.0	214.0	Agincourt North, Toronto	(Agincourt North, Scarborough North, Scarborou...	43.80803
1	Agincourt South-Malvern West	23757.0	31825.0	128.0	329.0	Agincourt South-Malvern West, Toronto	(Toronto Fire Station 243, Sheppard Avenue Eas...	43.78923
2	Alderwood	12054.0	47709.0	20.0	88.0	Alderwood, Toronto	(Alderwood, Etobicoke—Lakeshore, Etobicoke, To...	43.60171
3	Annex	30526.0	112766.0	95.0	604.0	Annex, Toronto	(The Annex, University—Rosedale, Toronto, Gold...	43.67033
4	Banbury-Don Mills	27695.0	67757.0	42.0	221.0	Banbury-Don Mills, Toronto	(Banbury Road, Don Valley West, North York, To...	43.73480

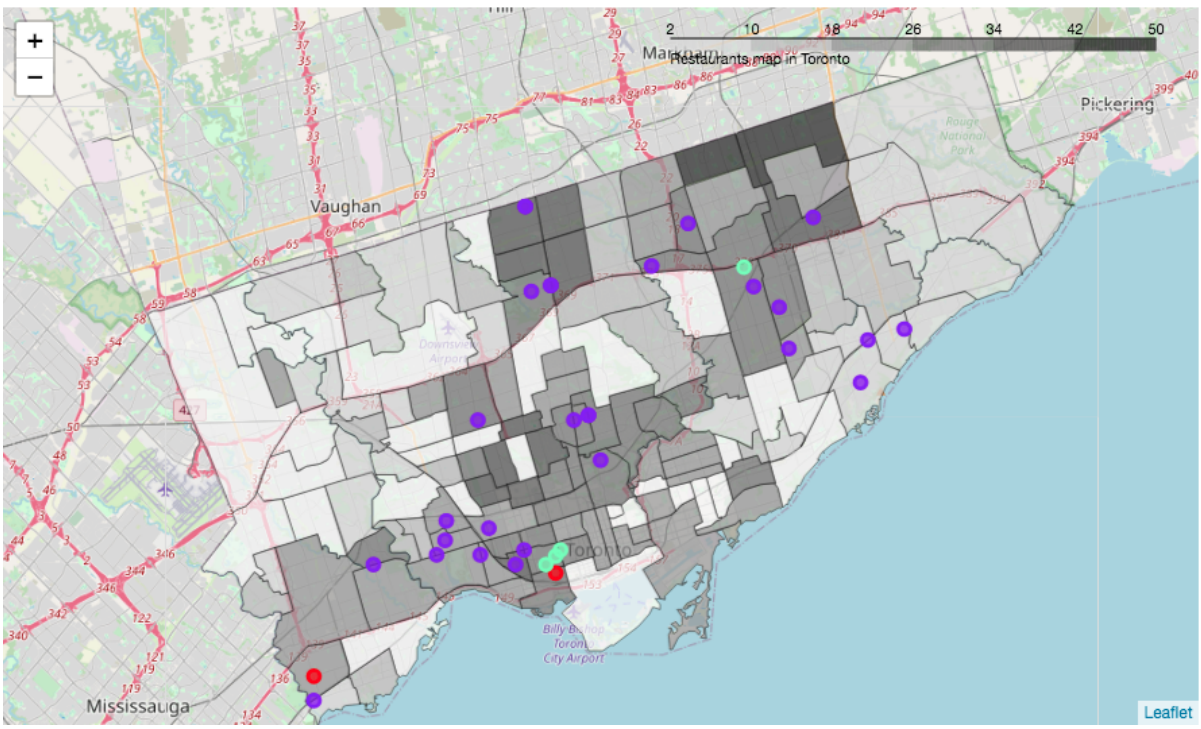
Restaurants map in Toronto

We use choropleth map to show the restaurants and seafood restaurants of each neighborhood

In [43]:

```
toronto_geo = r'Toronto_Neighbourhoods.geojson' # geojson file
# create a plain world map
toronto_restaurant_map = folium.Map(location=[latitude+0.075,longitude], zoom_start=
# generate choropleth map
toronto_restaurant_map.choropleth(
    geo_data=toronto_geo,
    data=df_Toronto_combine_venue,
    columns=['Hood_ID', 'Restaurants_sum'],
    key_on='feature.properties.AREA_SHORT_CODE',
    fill_color='Greys',
    fill_opacity=0.7,
    line_opacity=0.5,
    legend_name='Restaurants map in Toronto'
)

# set color scheme for the clusters
k = df_Toronto_combine_venue['Seafood Restaurant'].max()
x = np.arange(k)
ys = [i + x + (i*x)**2 for i in range(k)]
colors_array = cm.rainbow(np.linspace(0, 1, len(ys)))
rainbow = [colors.rgb2hex(i) for i in colors_array]

# add markers to the map
for lat, lon, poi, number in zip(df_Toronto_combine_venue['Latitude'], df_Toronto_co
    if number > 0:
        label = folium.Popup(str(poi) + '***' + str(number), parse_html=True)
        folium.CircleMarker(
            [lat, lon],
            radius=5,
            popup=label,
            color=rainbow[number-1],
            fill=True,
            fill_color=rainbow[number-1],
            fill_opacity=0.7).add_to(toronto_restaurant_map)
# display map
toronto_restaurant_map
```

Out[43]:



prepare the dataset for creating clusters (using k-means clustering)

In [44]:

```
df_Toronto_combine_venue.head()
```

Out[44]:

	Neighborhood	Population	Income	Hood_ID	Crime_total	Address	Coordinates	Latitud
0	Agincourt North	29113.0	30414.0	129.0	214.0	Agincourt North, Toronto	(Agincourt North, Scarborough North, Scarborou...	43.80803
1	Agincourt South-Malvern West	23757.0	31825.0	128.0	329.0	Agincourt South-Malvern West, Toronto	(Toronto Fire Station 243, Sheppard Avenue Eas...	43.78923
2	Alderwood	12054.0	47709.0	20.0	88.0	Alderwood, Toronto	(Alderwood, Etobicoke—Lakeshore, Etobicoke, To...	43.60171
3	Annex	30526.0	112766.0	95.0	604.0	Annex, Toronto	(The Annex, University—Rosedale, Toronto, Gold...	43.67033
4	Banbury-Don Mills	27695.0	67757.0	42.0	221.0	Banbury-Don Mills, Toronto	(Banbury Road, Don Valley West, North York, To...	43.73480

In [45]:

```
data_for_cluster=df_Toronto_combine_venue[['Population','Income','Crime_total','Rest
data_for_cluster.head()
```

Out[45]:

	Population	Income	Crime_total	Restaurants_sum	Seafood Restaurant
0	29113.0	30414.0	214.0	22	0
1	23757.0	31825.0	329.0	37	1
2	12054.0	47709.0	88.0	23	3
3	30526.0	112766.0	604.0	30	0
4	27695.0	67757.0	221.0	12	0

K-means Clustering

In [47]:

```
from sklearn.preprocessing import StandardScaler
# set number of clusters
kclusters = 4
print('standardize and normalize')
data_cluster = StandardScaler().fit_transform(data_for_cluster)
print(data_cluster)
# run k-means clustering
kmeans = KMeans(n_clusters=kclusters, random_state=0).fit(data_cluster)

# check cluster labels generated for each row in the dataframe
print('Check cluster labels generated for each row in the dataframe')
kmeans.labels_[0:115]
```

```
standardize and normalize
[[ 1.05718501e+00 -7.10785485e-01 -2.38524117e-01  8.75939201e-02
  -5.90671148e-01]
 [ 4.81296452e-01 -6.69735744e-01  2.61364895e-01  1.42583437e+00
   9.56324716e-01]
 [-7.77035110e-01 -2.07627956e-01 -7.86228599e-01  1.76809950e-01
   4.05031644e+00]
 [ 1.20911379e+00  1.68505313e+00  1.45675166e+00  8.01322158e-01
  -5.90671148e-01]
 [ 9.04718625e-01  3.75621661e-01 -2.08096090e-01 -8.04566377e-01
  -5.90671148e-01]
 [-3.66408071e-01 -2.59209240e-01 -6.77557075e-01 -8.04566377e-01
  -5.90671148e-01]
 [ 7.00641610e-01  4.88820138e-02  4.79079403e+00  1.76809950e-01
  -5.90671148e-01]
 [ 2.27436688e-01 -8.17731156e-02 -4.42826582e-01 -5.36918288e-01
  -5.90671148e-01]
 [-6.58760760e-01 -2.21097857e-01 -7.73188016e-01 -7.15350347e-01
  -5.90671148e-01]
 [ 1.14825626e+00 -6.28104150e-01  6.69969827e-01 -6.26134318e-01
  -5.90671148e-01]
 [ 2.64101736e-01 -8.39520499e-01  4.13505029e-01 -8.04566377e-01
  -5.90671148e-01]
 [-8.36709896e-01 -2.99328016e-01 -6.64516492e-01 -1.80054169e-01
  -5.90671148e-01]
 [-1.00272407e+00 -5.74108176e-01 -6.42782187e-01  5.33674069e-01
  -5.90671148e-01]
 [-8.93804150e-01  3.20604641e+00 -7.34066268e-01  6.22890098e-01
  -5.90671148e-01]
 [-6.36396156e-01 -2.32968720e-02 -7.34066268e-01 -1.42907859e+00
  -5.90671148e-01]
 [ 1.29663681e+00 -3.67376802e-02  5.01248394e+00  8.75939201e-02
  -5.90671148e-01]
 [-3.02002312e-01 -1.06618246e-01 -3.16767614e-01 -1.16143050e+00
  -5.90671148e-01]
 [-3.59741699e-01 -2.94644098e-01 -1.86361785e-01 -1.42907859e+00
   9.56324716e-01]
 [-5.53496589e-01 -3.81951166e-01 -3.73276807e-01  1.60426643e+00
  -5.90671148e-01]
 [-1.03379797e+00  1.10324651e-02 -5.38457524e-01 -9.08381394e-02
  -5.90671148e-01]
 [-2.25876639e-01 -8.72716280e-02 -6.08007299e-01 -9.08381394e-02
  -5.90671148e-01]
 [ 8.35474369e-01 -5.08155119e-01 -9.50777046e-02  1.76809950e-01
  -5.90671148e-01]
 [ 6.15269034e-01 -6.73605068e-01  2.57018034e-01  8.90538188e-01
```

```
  9.56324716e-01]
[ 1.86489130e+00 -4.39467352e-01  1.22636803e+00  1.76809950e-01
  9.56324716e-01]
[-8.05958565e-01 -4.14796778e-01 -3.73276807e-01  7.12106128e-01
  9.56324716e-01]
[ 2.25823856e-01 -1.51800203e-02  1.30959066e-01  6.22890098e-01
 -5.90671148e-01]
[-4.02750554e-01  1.35877896e+00 -6.34088465e-01 -1.69672667e+00
 -5.90671148e-01]
[ 3.75817236e-01 -7.21869788e-01  2.09202564e-01 -1.25064653e+00
  9.56324716e-01]
[-1.05637762e+00 -6.64295417e-01 -6.34088465e-01 -1.07221447e+00
 -5.90671148e-01]
[ 3.32378293e-01 -1.95263576e-01 -3.21114475e-01  1.76809950e-01
 -5.90671148e-01]
[-7.99184670e-01 -4.82669950e-01 -7.90575460e-01 -1.51829461e+00
 -5.90671148e-01]
[ 2.85176075e-01 -7.61988564e-01 -3.34155058e-01  7.12106128e-01
 -5.90671148e-01]
[-6.96178464e-01  8.80146545e-01 -6.29741604e-01  1.51505040e+00
 -5.90671148e-01]
[-9.19179375e-01  4.35445001e+00 -7.29719407e-01  1.51505040e+00
 -5.90671148e-01]
[ 1.20535052e+00 -7.81480645e-01  1.14377767e+00 -1.42907859e+00
 -5.90671148e-01]
[-1.00680992e+00 -4.77928904e-02 -9.51409316e-01 -1.25064653e+00
 -5.90671148e-01]
[-3.82536392e-01 -5.37829630e-01 -4.47173443e-01  6.22890098e-01
  9.56324716e-01]
[ 3.09798644e-01  7.62000201e-02 -3.12420753e-01 -1.62210963e-03
  9.56324716e-01]
[ 4.99360171e-01  4.75904055e-01 -9.94245655e-02 -1.62210963e-03
 -5.90671148e-01]
[-7.29725370e-01 -4.03625197e-01 -6.08007299e-01 -1.60751064e+00
 -5.90671148e-01]
[-2.52327085e-01 -4.19044306e-01 -4.94988914e-01 -2.69270199e-01
 -5.90671148e-01]
[-8.95954593e-01 -2.68547983e-01 -8.16656626e-01 -1.60751064e+00
 -5.90671148e-01]
[-7.38112097e-01 -7.01563112e-01  4.09158168e-01 -1.33986256e+00
 -5.90671148e-01]
[-6.06397480e-01 -6.82594699e-01 -6.51475909e-01  2.66025980e-01
  9.56324716e-01]
[ 2.65410378e+00 -5.98954364e-02  1.40024247e+00  8.75939201e-02
 -5.90671148e-01]
[-5.28443931e-01 -1.49442639e-01 -3.86317390e-01 -1.80054169e-01
  9.56324716e-01]
[-8.84127158e-01 -6.26358590e-01 -5.90619856e-01 -9.82998437e-01
 -5.90671148e-01]
[-2.32005401e-01 -6.94493596e-01  2.35283729e-01 -1.51829461e+00
 -5.90671148e-01]
[-1.43622205e-01 -5.06904134e-01  2.19137117e+00  1.06897025e+00
  2.50332058e+00]
[-1.07626922e+00  2.61241071e+00 -7.47106851e-01  6.22890098e-01
  9.56324716e-01]
[-1.21454268e+00  6.33731728e-01 -9.03593846e-01 -4.47702258e-01
 -5.90671148e-01]
[ 2.65711440e+00 -6.69706651e-01  4.78707944e-01  2.66025980e-01
 -5.90671148e-01]
[-3.35119130e-01  5.09855188e-01 -4.51520304e-01  7.12106128e-01
  9.56324716e-01]
```

```
[-5.02531096e-01  1.65491313e+00 -7.90575460e-01  2.66025980e-01
 -5.90671148e-01]
[-4.41028434e-01  3.32695550e+00 -5.73232412e-01  2.66025980e-01
 -5.90671148e-01]
[-2.63724431e-01  2.05738097e+00 -6.64516492e-01  5.33674069e-01
 -5.90671148e-01]
[-4.00170022e-01 -2.64998679e-01 -3.95011112e-01  1.24740231e+00
  9.56324716e-01]
[-9.88853719e-01 -2.17083070e-01 -5.94966717e-01 -1.07221447e+00
  9.56324716e-01]
[ 2.63571750e+00 -7.35252411e-01  1.02641243e+00 -1.25064653e+00
 -5.90671148e-01]
[-9.85950621e-01 -5.16039229e-01 -8.16656626e-01  1.76809950e-01
 -5.90671148e-01]
[-9.38318315e-01  2.19132251e-01 -8.77512680e-01 -8.93782407e-01
 -5.90671148e-01]
[ 7.83971266e-01 -7.78542287e-01  4.39586195e-01  2.13956260e+00
 -5.90671148e-01]
[-1.96308052e-01 -6.56178565e-01 -1.55933758e-01 -1.25064653e+00
 -5.90671148e-01]
[ 1.31741986e-01  1.18384375e-01  3.53889807e+00  1.76809950e-01
 -5.90671148e-01]
[-6.11558542e-01 -6.98770217e-01 -5.34110663e-01 -1.33986256e+00
 -5.90671148e-01]
[-2.69423104e-01  8.87157876e-01 -6.12354160e-01  9.79754217e-01
  9.56324716e-01]
[ 1.11578458e+00  6.38065476e-02  1.09224761e-01  9.79754217e-01
  9.56324716e-01]
[-8.40580693e-01 -3.12594268e-01 -4.08051695e-01 -8.93782407e-01
 -5.90671148e-01]
[-3.42323113e-01 -2.80272325e-01 -4.47173443e-01  1.87191451e+00
  9.56324716e-01]
[ 4.89253090e-01 -5.80159448e-01  1.70080815e-01  1.87191451e+00
  9.56324716e-01]
[ 1.27943327e+00  4.59001221e-01  9.74250095e-01  7.12106128e-01
  4.05031644e+00]
[-7.91873165e-01  5.35514913e-01 -4.34132861e-01  3.55242009e-01
 -5.90671148e-01]
[-5.84462964e-01 -8.16130002e-01 -7.33433997e-02 -1.25064653e+00
 -5.90671148e-01]
[ 2.07437570e-01 -4.37634514e-01  7.01030124e-02  1.69348246e+00
 -5.90671148e-01]
[-6.51310454e-02 -3.18238244e-01 -7.76902607e-02 -1.07221447e+00
 -5.90671148e-01]
[-1.08035506e+00 -4.45345127e-02 -8.47084653e-01  2.66025980e-01
 -5.90671148e-01]
[-5.86505885e-01  9.38301712e-02 -3.42848780e-01  8.90538188e-01
  2.50332058e+00]
[-9.20254596e-01 -4.60908641e-01 -3.47195641e-01 -1.25064653e+00
 -5.90671148e-01]
[-3.72321789e-01 -5.38207835e-01 -7.38413129e-01  8.01322158e-01
  9.56324716e-01]
[-9.11545303e-01 -5.89090895e-01 -2.98747900e-02  3.55242009e-01
 -5.90671148e-01]
[-9.41006368e-01 -5.94298481e-01 -4.94988914e-01 -9.08381394e-02
 -5.90671148e-01]
[-4.63070472e-01 -1.24102934e-01 -1.60280619e-01  3.55242009e-01
  9.56324716e-01]
[ 1.76578717e-01  4.45284138e+00  3.09812637e-02  9.79754217e-01
  9.56324716e-01]
[ 2.92624231e+00 -4.44820401e-01  4.52626778e-01 -1.60751064e+00
```

```
  -5.90671148e-01]
 [-9.90359029e-01  4.95803434e-01 -6.60169631e-01 -3.58486229e-01
   9.56324716e-01]
 [-1.00422938e+00 -6.70463060e-01 -6.68863353e-01 -1.16143050e+00
  -5.90671148e-01]
 [-2.74906733e-01 -6.38082932e-01 -2.55279290e-02 -1.51829461e+00
   9.56324716e-01]
 [ 2.76144215e-01 -5.71344373e-01  6.39541800e-01 -1.62210963e-03
  -5.90671148e-01]
 [ 9.24180131e-01 -3.03372953e-02  9.61209512e-01  3.55242009e-01
  -5.90671148e-01]
 [-1.57922649e-01  1.32866806e+00 -3.25461336e-01 -5.36918288e-01
  -5.90671148e-01]
 [ 5.74410622e-01 -6.70870358e-01 -4.94988914e-01  2.58564275e+00
  -5.90671148e-01]
 [ 6.20430096e-01  2.70393516e-01 -2.73299004e-01  1.76809950e-01
  -5.90671148e-01]
 [ 8.77945613e-01 -6.00640680e-01 -1.51586897e-01  9.79754217e-01
   2.50332058e+00]
 [-3.86837277e-01 -7.10320003e-01 -2.38524117e-01 -7.15350347e-01
  -5.90671148e-01]
 [ 2.45822973e-01  1.09778872e+00 -3.73276807e-01  1.76809950e-01
  -5.90671148e-01]
 [ 1.96470312e-01 -7.55559087e-01 -4.16745417e-01  4.44458039e-01
  -5.90671148e-01]
 [-2.92970452e-01 -1.20786371e-01  9.18373173e-02  8.01322158e-01
   2.50332058e+00]
 [-1.25518605e+00 -1.18953533e-01 -1.12465148e-01  1.76809950e-01
  -5.90671148e-01]
 [-1.90394334e-01 -5.54499724e-01 -1.68974341e-01 -2.69270199e-01
  -5.90671148e-01]
 [ 8.72139417e-01 -6.26154942e-01  1.57411691e+00 -1.51829461e+00
  -5.90671148e-01]
 [-1.38568665e-01 -6.35639148e-01  2.87446061e-01 -8.93782407e-01
  -5.90671148e-01]
 [ 9.28588539e-01 -5.75999198e-01  7.78641351e-01  4.44458039e-01
   9.56324716e-01]
 [ 3.34966448e+00 -2.76955762e-01  4.70014222e-01  1.33661834e+00
   9.56324716e-01]
 [-2.52112041e-01 -2.98775255e-01 -2.90686448e-01  1.33661834e+00
   9.56324716e-01]
 [ 3.67771452e+00 -6.97286492e-01  1.70886960e+00 -5.36918288e-01
  -5.90671148e-01]
 [-5.30271807e-01 -1.12234187e-02 -5.68885551e-01  1.69348246e+00
  -5.90671148e-01]
 [-8.02517857e-01  1.00323758e+00 -3.64583085e-01  9.79754217e-01
   9.56324716e-01]
 [-7.26069618e-01  1.72601559e+00 -8.16656626e-01  1.06897025e+00
  -5.90671148e-01]
 [ 8.93751367e-01 -7.24051737e-01  2.00010928e+00 -1.33986256e+00
  -5.90671148e-01]
 [-4.81349235e-01 -4.74756747e-01  6.83010410e-01  5.33674069e-01
   9.56324716e-01]]
```

Check cluster labels generated for each row in the dataframe

Out[47]:

```
array([1, 0, 0, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 3, 1, 2, 1, 1, 0, 1, 1,
1,
       0, 2, 0, 1, 1, 1, 1, 1, 1, 1, 3, 3, 2, 1, 0, 0, 1, 1, 1, 1, 1,
```

```
0,
       2, 0, 1, 1, 0, 3, 1, 2, 0, 3, 3, 3, 0, 1, 2, 1, 1, 0, 1, 2, 1,
0,
       0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 3, 2, 1, 1, 1,
1,
       2, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 2, 1, 0, 2, 0, 2, 0, 0, 3, 2,
0],
      dtype=int32)
```

In [48]:

```
df_Toronto_combine_venue.insert(0, 'Cluster Labels', kmeans.labels_)
```

In [50]:

```
data_examine=df_Toronto_combine_venue[['Cluster Labels','Neighborhood','Population',
data_examine.head()
```

Out[50]:

	Cluster Labels	Neighborhood	Population	Income	Crime_total	Restaurants_sum	Seafood Restaurant	Hoo
0	1	Agincourt North	29113.0	30414.0	214.0	22	0	1
1	0	Agincourt South-Malvern West	23757.0	31825.0	329.0	37	1	1
2	0	Alderwood	12054.0	47709.0	88.0	23	3	
3	2	Annex	30526.0	112766.0	604.0	30	0	
4	1	Banbury-Don Mills	27695.0	67757.0	221.0	12	0	

In [51]:

```
cluster0=data_examine[data_examine['Cluster Labels']==0].sort_values(by=['Income']
cluster1=data_examine[data_examine['Cluster Labels']==1].sort_values(by=['Income']
cluster2=data_examine[data_examine['Cluster Labels']==2].sort_values(by=['Income']
cluster3=data_examine[data_examine['Cluster Labels']==3].sort_values(by=['Income']
cluster0.describe()[1:2].append(cluster1.describe()[1:2]).append(cluster2.describe()
```

Out[51]:

	Cluster Labels	Population	Income	Crime_total	Restaurants_sum	Seafood Restaurant	Hood_I
mean	0.0	18889.366667	46953.333333	251.066667	32.300000	1.100000	84.03333
mean	1.0	15637.927273	46668.236364	185.490909	14.345455	0.090909	65.18181
mean	2.0	35642.062500	45612.500000	670.187500	17.625000	0.125000	86.1875(
mean	3.0	13760.222222	147542.555556	124.333333	30.222222	0.222222	85.88888

The clustering results show that the dataset is divided into four categories:

- **The first group** is characterized by a large population, low income, high crime rate and the largest number of restaurants (including seafood restaurants).
- **The second group** is characterized by a small population, low income, a moderate crime rate, and the fewest restaurants (including seafood restaurants).
- **The third group** is characterized by the largest population, the lowest income, the highest crime rate, and a small number of restaurants (including seafood restaurants).
- **The fourth group** is characterized by a small population, the highest income, the lowest crime rate, and a large number of restaurants (including seafood restaurants).

Based on the several key factors we selected when choosing a restaurant address in the "Business Problem" paragrah , we will consider the final selection in the second and fourth group. One of the reasons is that the price of seafood is usually higher than the price of general food, so we need to consider the affordability of residents. Second, we need to consider the safety of the neigborhood, and then the number of existing restaurants and the number of people.

Finally, we decided to choose the fourth group which is a group of the highest income and the safest neighborhoods.

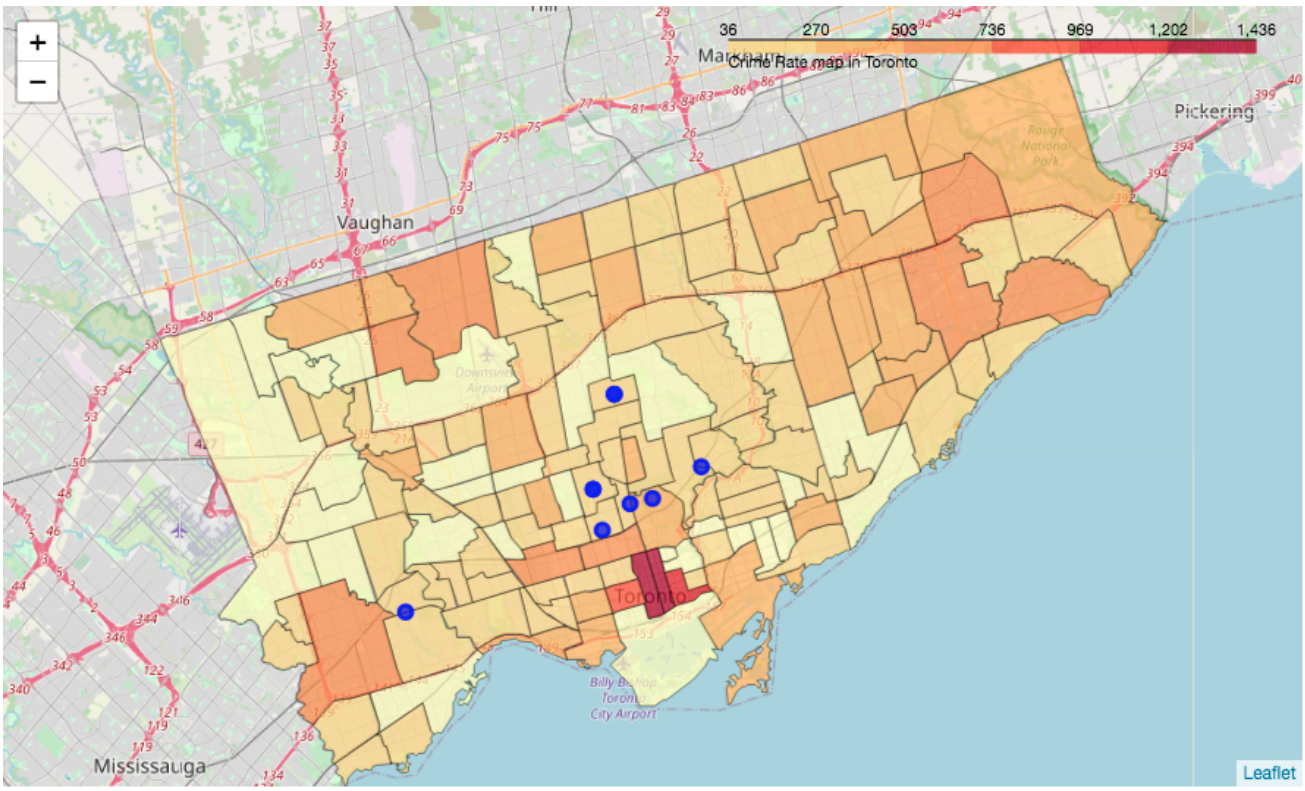
Now let ’s look at these neighborhood on the different choropleth map

(1) Selected Neighborhoods on the Crime Rate Map

In [55]:

```
toronto_geo = r'Toronto_Neighbourhoods.geojson' # geojson file
# create a plain world map
toronto_cluster_map = folium.Map(location=[latitude+0.075,longitude], zoom_start=11)
# generate choropleth map
toronto_cluster_map.choropleth(
    geo_data=toronto_geo,
    data=data_examine,
    columns=['Hood_ID', 'Crime_total'],
    key_on='feature.properties.AREA_SHORT_CODE',
    fill_color='YlOrRd',
    fill_opacity=0.7,
    line_opacity=0.5,
    legend_name='Crime Rate map in Toronto'
)

# add markers to the map
for lat, lon, poi, cluster in zip(data_examine['Latitude'], data_examine['Longitude
    if cluster == 3:
        label = folium.Popup(str(poi)+'  Cluster:'+str(cluster), parse_html=True)
        cls='blue'
        folium.CircleMarker(
            [lat, lon],
            radius=5,
            popup=label,
            color=cls,
            fill=True,
            fill_color=cls,
            fill_opacity=0.7).add_to(toronto_cluster_map)
# display map
toronto_cluster_map
```

Out[55]:

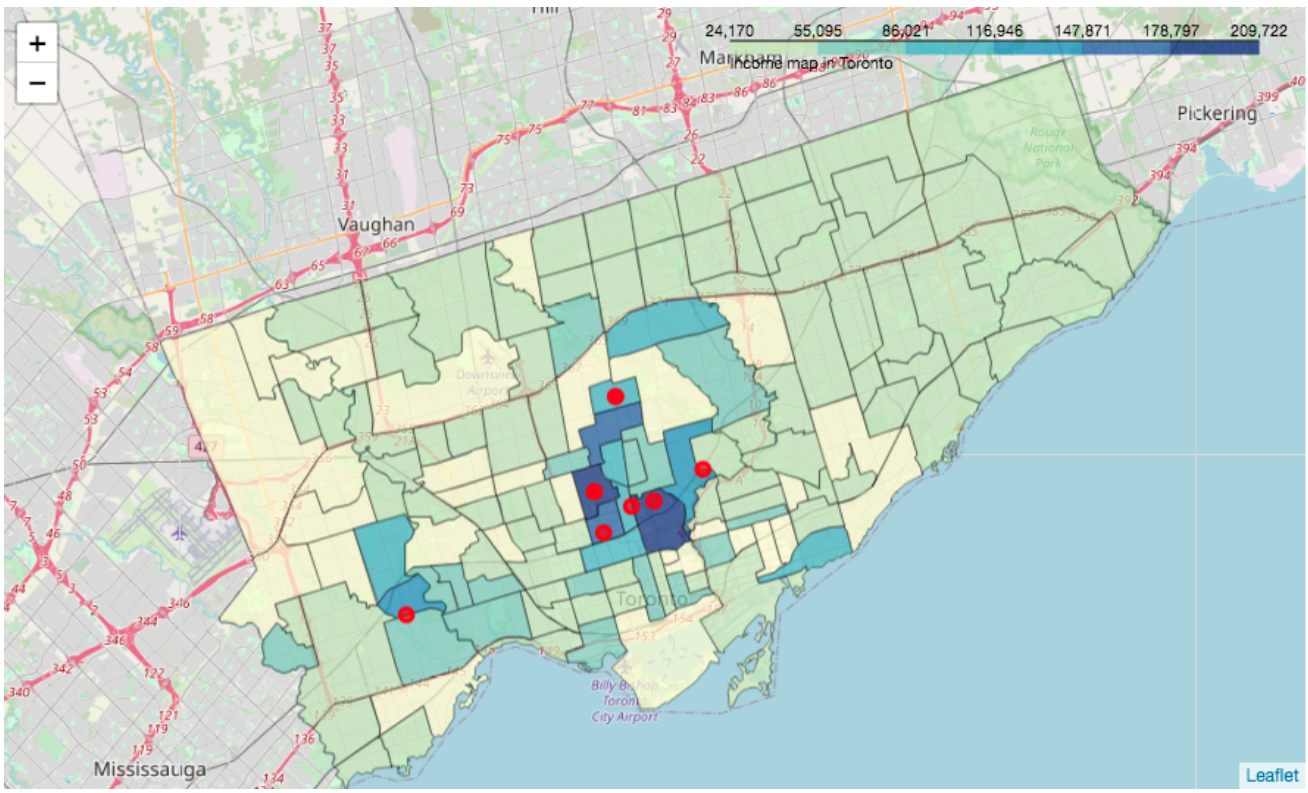


(2) Selected Neighborhoods on the Income Map

In [56]:

```
toronto_geo = r'Toronto_Neighbourhoods.geojson' # geojson file
# create a plain world map
toronto_cluster_map = folium.Map(location=[latitude+0.075,longitude], zoom_start=11)
# generate choropleth map
toronto_cluster_map.choropleth(
    geo_data=toronto_geo,
    data=data_examine,
    columns=['Hood_ID', 'Income'],
    key_on='feature.properties.AREA_SHORT_CODE',
    fill_color='YlGnBu',
    fill_opacity=0.7,
    line_opacity=0.5,
    legend_name='Income map in Toronto'
)

# add markers to the map
for lat, lon, poi, cluster in zip(data_examine['Latitude'], data_examine['Longitude
    if cluster == 3:
        label = folium.Popup(str(poi)+'  Cluster:'+str(cluster), parse_html=True)
        cls='red'
        folium.CircleMarker(
            [lat, lon],
            radius=5,
            popup=label,
            color=cls,
            fill=True,
            fill_color=cls,
            fill_opacity=0.7).add_to(toronto_cluster_map)
# display map
toronto_cluster_map
```

Out[56]:

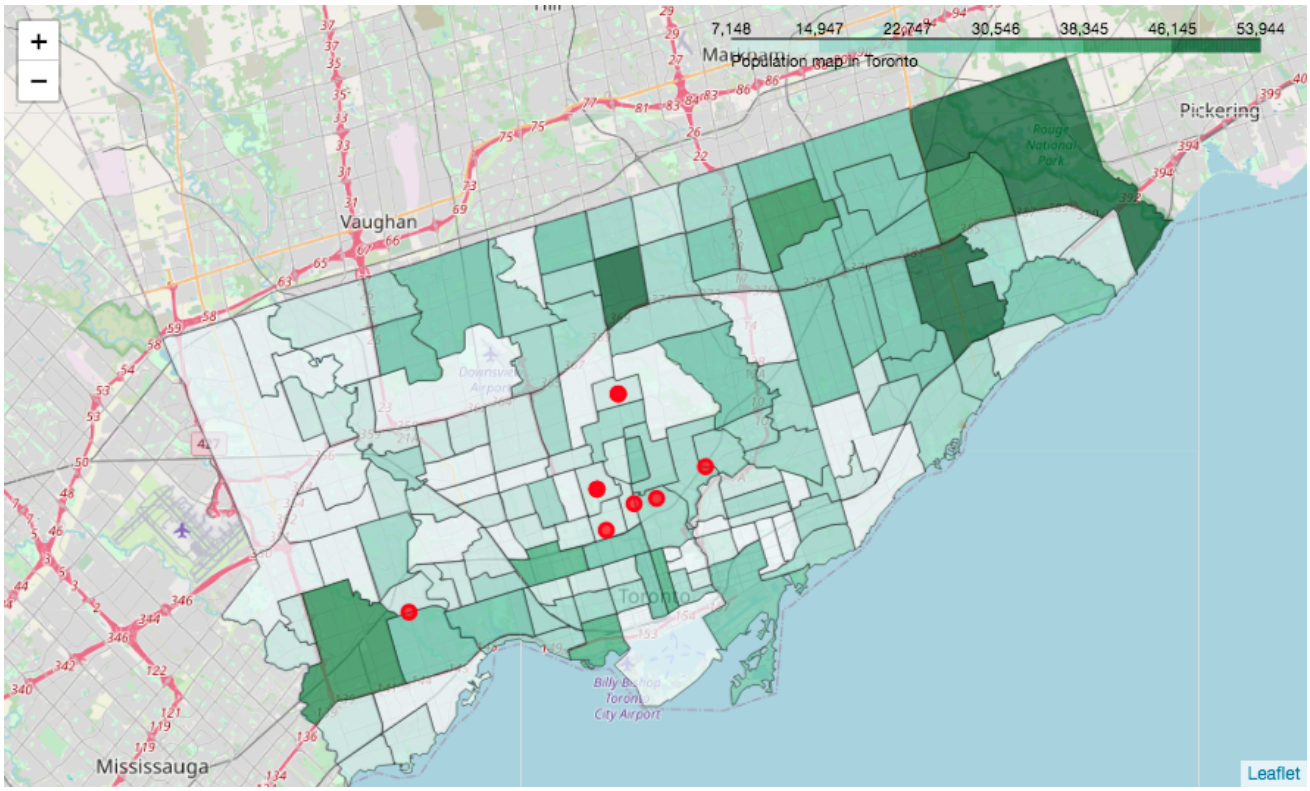


(3) Selected Neighborhoods on the Population Map

In [57]:

```
toronto_geo = r'Toronto_Neighbourhoods.geojson' # geojson file
# create a plain world map
toronto_cluster_map = folium.Map(location=[latitude+0.075,longitude], zoom_start=11)
# generate choropleth map
toronto_cluster_map.choropleth(
    geo_data=toronto_geo,
    data=data_examine,
    columns=['Hood_ID', 'Population'],
    key_on='feature.properties.AREA_SHORT_CODE',
    fill_color='BuGn',
    fill_opacity=0.7,
    line_opacity=0.5,
    legend_name='Population map in Toronto'
)

# add markers to the map
for lat, lon, poi, cluster in zip(data_examine['Latitude'], data_examine['Longitude
    if cluster == 3:
        label = folium.Popup(str(poi)+'  Cluster:'+str(cluster), parse_html=True)
        cls='red'
        folium.CircleMarker(
            [lat, lon],
            radius=5,
            popup=label,
            color=cls,
            fill=True,
            fill_color=cls,
            fill_opacity=0.7).add_to(toronto_cluster_map)
# display map
toronto_cluster_map
```

Out[57]:



(4) Selected Neighborhoods on the Restaurants Map (By Heatmap)

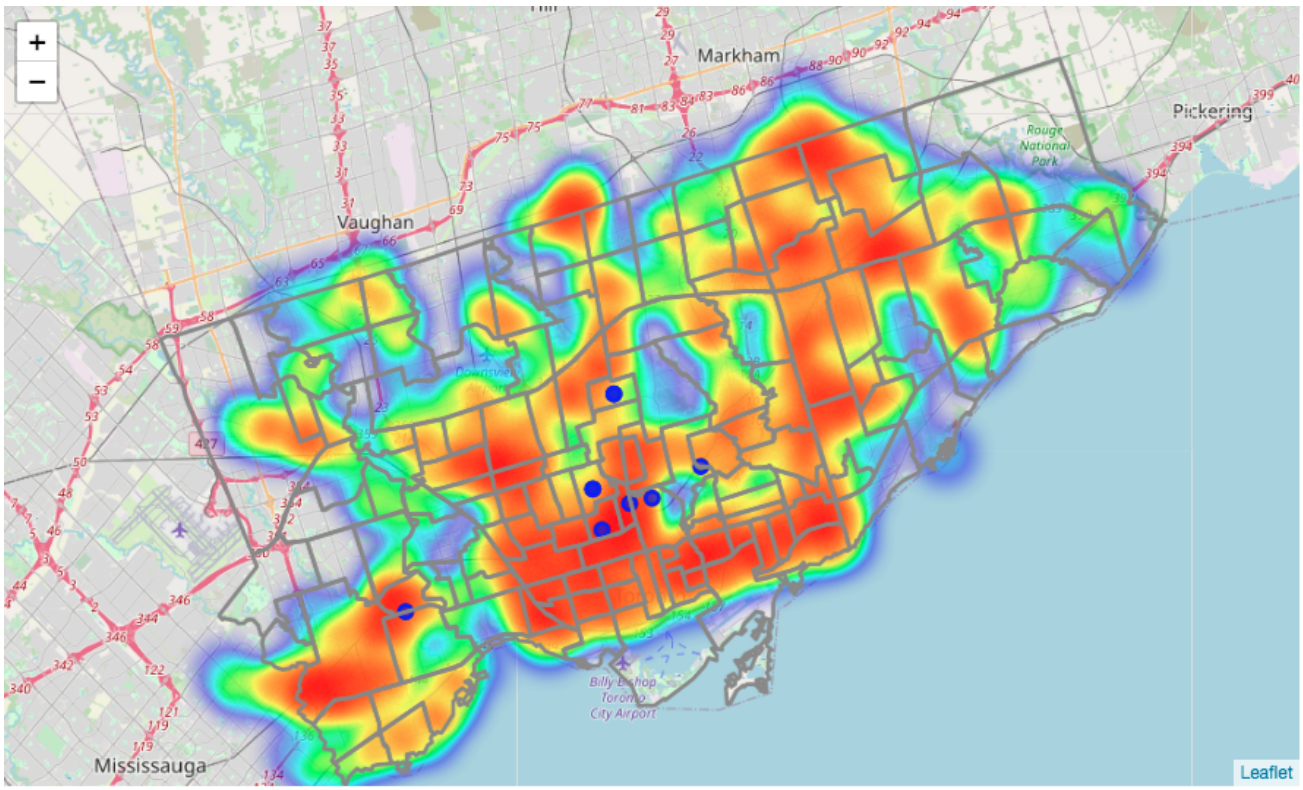
In [58]:

```
restaurant_latlons = []
SeafoodRestaurant_latlons = []
for lo,lt, category in zip(Toronto_Restaurant_venues['Venue Latitude'],Toronto_Resta
    restaurant_latlons.append([lo,lt])
    if 'Seafood' in category:
       SeafoodRestaurant_latlons.append([lo,lt])
# print(restaurant_latlons)
```

In [67]:

```
from folium import plugins
from folium.plugins import HeatMap

def boroughs_style(feature):
    return { 'color': '#888888', 'fill': False }

toronto_geo = r'Toronto_Neighbourhoods.geojson' # geojson file
# create a plain world map
toronto_cluster_map = folium.Map(location=[latitude+0.075,longitude], zoom_start=11)

HeatMap(restaurant_latlons).add_to(toronto_cluster_map)

# add markers to the map
for lat, lon, poi, cluster in zip(data_examine['Latitude'], data_examine['Longitude
    if cluster == 3:
        label = folium.Popup(str(poi)+'  Cluster:'+str(cluster), parse_html=True)
        cls='blue'
        folium.CircleMarker(
            [lat, lon],
            radius=5,
            popup=label,
            color=cls,
            fill=True,
            fill_color=cls,
            fill_opacity=0.7).add_to(toronto_cluster_map)

folium.GeoJson(toronto_geo, style_function=boroughs_style, name='geojson').add_to(to
# display map
toronto_cluster_map
```

Out[67]:

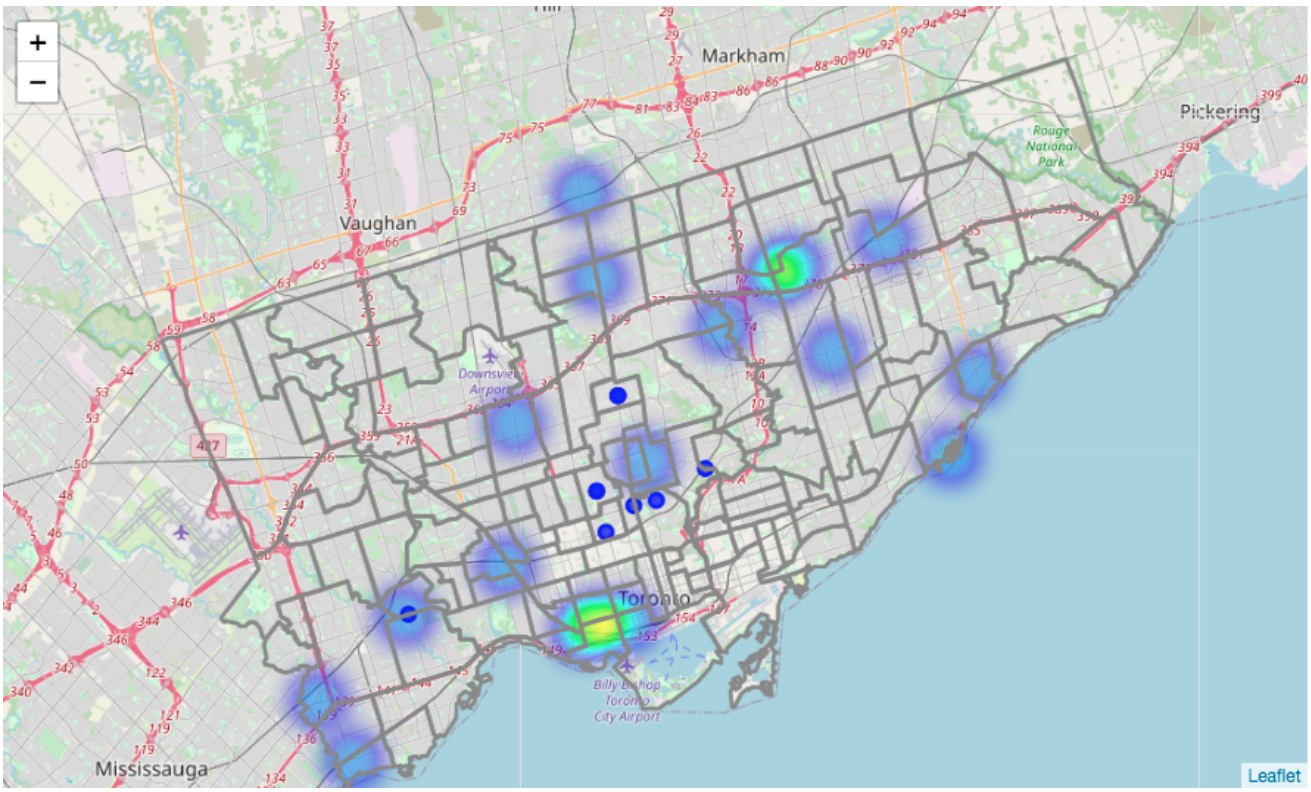


(5) Selected Neighborhoods on the Seafood restaurants Map (By Heatmap)

In [62]:

```
toronto_geo = r'Toronto_Neighbourhoods.geojson' # geojson file
# create a plain world map
toronto_cluster_map = folium.Map(location=[latitude+0.075,longitude], zoom_start=11)

HeatMap(SeafoodRestaurant_latlons).add_to(toronto_cluster_map)

# add markers to the map
for lat, lon, poi, cluster in zip(data_examine['Latitude'], data_examine['Longitude
    if cluster == 3:
        label = folium.Popup(str(poi)+'  Cluster:'+str(cluster), parse_html=True)
        cls='blue'
        folium.CircleMarker(
            [lat, lon],
            radius=5,
            popup=label,
            color=cls,
            fill=True,
            fill_color=cls,
            fill_opacity=0.7).add_to(toronto_cluster_map)

folium.GeoJson(toronto_geo, style_function=boroughs_style, name='geojson').add_to(to
# display map
toronto_cluster_map
```

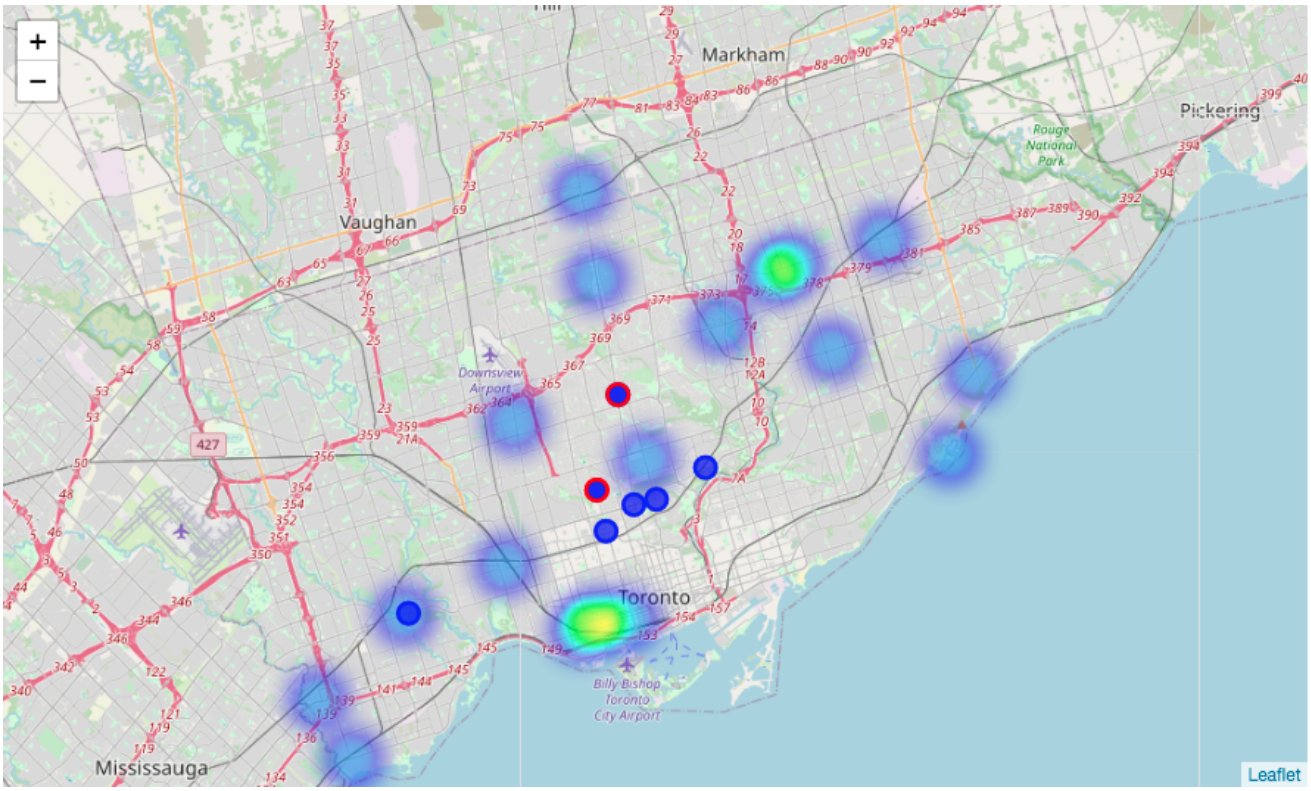
Out[62]:



In [69]:

```
toronto_geo = r'Toronto_Neighbourhoods.geojson' # geojson file
# create a plain world map
toronto_cluster_map = folium.Map(location=[latitude+0.075,longitude], zoom_start=11)

HeatMap(restaurant_latlons).add_to(toronto_cluster_map)

# add markers to the map
for lat, lon, poi, cluster in zip(data_examine['Latitude'], data_examine['Longitude'
    if cluster == 3:
        label = folium.Popup(str(poi)+'  Cluster:'+str(cluster), parse_html=True)
        cls='blue'
        if ('Lawrence Park South' in poi) or ('Forest Hill' in poi):
            cls='red'
        folium.CircleMarker(
            [lat, lon],
            radius=8,
            popup=label,
            color=cls,
            fill=True,
            fill_color='blue',
            fill_opacity=0.7).add_to(toronto_cluster_map)

# folium.GeoJson(toronto_geo, style_function=boroughs_style, name='geojson').add_to(
# display map
toronto_cluster_map
```

Out[69]:



5.Results and Discussion

According to several choropleth maps, we will eventually choose a location in Lawrence Park South or Forest Hill to open a seafood restaurant. There are currently no seafood restaurant in these two neighborhoods, also there are just relatively few restaurants and low crime rate, higher income residents and a moderate population.

Result of all this is 2 neighborhoods containing largest number of potential new restaurant locations based on

crime rate, income, population and existing venues - both restaurants in general and seafood restaurants particularly. This, of course, does not imply that those zones are actually optimal locations for a new restaurant! Purpose of this analysis was to only provide info on areas of toronto's neighorhood but not crowded with existing restaurants (particularly Seafood) - it is entirely possible that there is a very good reason for small number of restaurants in any of those areas, reasons which would make them unsuitable for a new restaurant regardless of lack of competition in the area. Recommended zones should therefore be considered only as a starting point for more detailed analysis which could eventually result in location which has not only no nearby competition but also other factors taken into account and all other relevant conditions met.

6.Conclusion

Purpose of this project was to identify Toronto areas with low number of restaurants (particularly seafood) in order to aid stakeholders in narrowing down the search for optimal location for a new seafood restaurant. We collected data on the population, income, and crime rate of 140 neighborhoods in the Toronto area from the public database of the Toronto Municipality. We also obtained data from Foursquare about restaurants (including seafood restaurants) within 500 meters of each neighborhood center. According to the data we got, the population, income, crime rate, number of restaurants, and number of seafood restaurants in each neighborhood are used as variables, and then used k-means clustering to create the main classification group. Through the analysis of the classificated data, the neighborhoods that satisfied the business requirements conditions proposed at the beginning of this article, were selected to be used as starting points for final exploration by stakeholders.

Final decision on optimal restaurant location will be made by stakeholders based on specific characteristics of neighborhoods and locations in every recommended zone, taking into consideration additional factors like attractiveness of each location (proximity to park or water), levels of noise / proximity to major roads, real estate availability, prices, social and economic dynamics of every neighborhood etc.