Capstone Project - Applied Data Science Specialization

This notebook will be used to complete the Capstone Project for the Applied Data Science Specialization by IBM. It contains the final assignment, in which a business problem should be adressed and resolved with the tools learned throughout all the courses.

Making Yourself at Home

The problem of finding the best place to move out to

Introduction/Business Problem

Living in Mexico is great: Amazing food, diverse activities to enjoy both day and night, a lot of different stores, and the list goes on. Yet at times, several factors could make anyone consider moving out of the country: Contrasting political opinions against the current government, the high rates of crime and overall insecurity, increasing pollution, etc. I know it because I've lived here my whole life, and every now and then, I wonder to myself how nice it would be if the place where I lived in could preserve most of the good things, while minimizing the bad ones? Were I to move to a foreign country, I would probably get homesick if the neighborhood I move to were too different from where I live now. On the other hand, it would be somewhat boring and uneventful if it was too similar.

Thus, an app that could show someone recommendations of neighborhoods in different countries to move out to, based on where they live and on certain activities they would like to do, would certainly save me a lot of time spent on research, and could be really profitable if used correctly. Businesses that could benefit from this would be hotels, because the app could have an amusement-to-comfort parameter, in which people who only use the app to search for places that are "as different as possible"(by setting the "amusement" high and the "comfort" low) and only want to go out for some days could stay at any of the hotels that are shown as recommended (benefit for the hotel by being advertised, benefit for us as the hotel would pay a fee to be mentioned on the app).

Data

To begin testing, and given the time to develop all the ideas with all the possible data would take a tremendous amount of time, I will reduce the scope of the project to the following:

1. **Restrict current location to adresses in Mexico City.**
2. **Restrict search location to Tokyo.**
3. **Use only Foursqare data, and the data from the cities neighborhoods' locations.**

Further, in the observations category, I'll explain how a full scale project could be achieved.

The data that will be used is: Location of the neighborhood in which the person lives (in Mexico), locations of neighborhoods in Tokyo, certain queries that the person highly values (which will be used to determine the similarity between neighborhoods), an amusement-to-comfort parameter that will be used to determine how different or similar the user wants the place to be compared to its current neighborhood, venues on the current location, and venues on the neighborhoods in Tokyo.

Once the person determines its own location and the queries, the app will make Foursquare requests to meet the demands of those specific queries. Once it retrieves the different venues, it proceeds to make clusters with the data. Finally, based on the parameters given by the user, the app determines the top best 3 neighborhoods to live in, along with a list of venues available at that location.

Methodology

First, I Install the necesary packages, and import any libraries that will be used. Generally, in my workspace evrything is already installed, so I like keeping those lines commented and in another cell.

evrything is already installed, so I like keeping those lines commented and in another cell.

In [ ]:

```
# !conda install -c conda-forge geopy --yes
# !conda install -c conda-forge folium=0.5.0 --yes
```

In [4]:

```
import numpy as np
import pandas as pd

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 longitude va
lues

import requests # library to handle requests
from pandas.io.json import json_normalize # tranform JSON file into a pandas dataframe

# 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
```

I need data for neighborhoods in Mexico

In [270]:

```
mexico = pd.read_csv('mexico_coordinates.txt', sep = "\t", header = None)
mexico.columns = ['Region','Coords']
mexico[['Latitude','Longitude']] = mexico.Coords.str.split(",",expand=True,)
mexico.drop('Coords',inplace = True,axis = 1)
mexico
```

Out[270]:

	Region	Latitude	Longitude
0	Ciudad de México	19.42847	-99.12766
1	Iztapalapa	19.35529	-99.06224
2	Guadalajara	20.66682	-103.39182
3	Puebla	19.03793	-98.20346
4	Tijuana	32.5027	-117.00371
5	Monterrey	25.67507	-100.31847
6	Ecatepec, de Morelos	19.60492	-99.06064
7	Chihuahu	28.63528	-106.08889
8	Naucalpan de Juárez,	19.47851	-99.23963
9	Mérida	20.97537	-89.61696
10	San Luis	22.14982	-100.97916
11	Hermosillo	29.1026	-110.97732
12	Saltillo	25.42321	-101.0053
13	Mexicali	32.62781	-115.45446
14	Guadalupe	25.67678	-100.25646
15	Paso del Norte	31.72024	-106.46084
16	Cancún	21.17429	-86.84656

	Region	Latitude	Longitude
17	Coyoacán	19.3467	-99.16174
18	León de los Aldama	21.12908	-101.67374
19	Morelia	19.70078	-101.18443

There are two regions which names we would like to clean:

In [271]:

```
mexico.loc[mexico['Region'] =='Naucalpan de Juárez,','Region'] = 'Naucalpan de Juárez'
mexico.loc[mexico['Region'] =='Ecatepec, de Morelos','Region'] = 'Ecatepec de Morelos'
mexico
```

Out[271]:

	Region	Latitude	Longitude
0	Ciudad de México	19.42847	-99.12766
1	Iztapalapa	19.35529	-99.06224
2	Guadalajara	20.66682	-103.39182
3	Puebla	19.03793	-98.20346
4	Tijuana	32.5027	-117.00371
5	Monterrey	25.67507	-100.31847
6	Ecatepec de Morelos	19.60492	-99.06064
7	Chihuahu	28.63528	-106.08889
8	Naucalpan de Juárez	19.47851	-99.23963
9	Mérida	20.97537	-89.61696
10	San Luis	22.14982	-100.97916
11	Hermosillo	29.1026	-110.97732
12	Saltillo	25.42321	-101.0053
13	Mexicali	32.62781	-115.45446
14	Guadalupe	25.67678	-100.25646
15	Paso del Norte	31.72024	-106.46084
16	Cancún	21.17429	-86.84656
17	Coyoacán	19.3467	-99.16174
18	León de los Aldama	21.12908	-101.67374
19	Morelia	19.70078	-101.18443

I need to convert the Latitude and Longitude to floats to pass them as arguments in the map function.

In [272]:

```
mexico[['Latitude','Longitude']] = mexico[['Latitude','Longitude']].astype(float)
```

Let's get the coordinates for Mexico

In [62]:

```
address = 'Mexico City, MX'

geolocator = Nominatim(user_agent="mx_explorer")
location = geolocator.geocode(address)
latitude = location.latitude
longitude = location.longitude
print('The geograpical coordinate of Mexico City, Mexico are {}, {}.'.format(latitude, l
ongitude))
```

The geograpical coordinate of Toronto, Ontario are 19.4326296, -99.1331785

```
The geographical coordinate of Toronto, Ontario are 19.4326296, -99.1331785.
```

I create a map to visualize these different regions.

In [72]:

```
map_mexico = folium.Map(location=[latitude, longitude], zoom_start=4)

# add markers to map
for lat, lng, region in zip(mexico['Latitude'],
                            mexico['Longitude'],
                            mexico['Region']):
    label = '{}'.format(region)
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker(
        [lat, lng],
        radius=5,
        popup=label,
        color='blue',
        fill=True,
        fill_color='#3186cc',
        fill_opacity=0.7,
        parse_html=False).add_to(map_mexico)

map_mexico
```

Great. Now it's time to get the venues of every region. I define the field that will be used, and I will borrow the getNearbyVenues function for convenience. Given every region is rather far away from each other, I will use a 1000 radius, and a 20 venues limit (maximum 20,000).

In [76]:

```
CLIENT_ID = ''
CLIENT_SECRET = ''
VERSION = '20180605'
RADIUS = 1000
LIMIT = 20
```

In [77]:

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

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

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

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

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

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

    return(nearby_venues)
```

And I run the function:

In [78]:

```
mexico_venues = getNearbyVenues(names=mexico['Region'],
                                   latitudes=mexico['Latitude'],
                                   longitudes=mexico['Longitude'],
                                   radius = RADIUS
                                  )
```

```
Ciudad de México
Iztapalapa
Guadalajara
Puebla
Tijuana
Monterrey
Ecatepec de Morelos
Chihuahu
Naucalpan de Juárez
Mérida
San Luis
Hermosillo
Saltillo
Mexicali
Guadalupe
Paso del Norte
Cancún
Coyoacán

---------------------------------------------------------------------------
```

```
KeyError                                  Traceback (most recent call last)
<ipython-input-78-5a729c231ac5> in <module>
      2                                    latitudes=mexico['Latitude'],
      3                                    longitudes=mexico['Longitude'],
----> 4                                    radius = RADIUS
      5                                   )

<ipython-input-77-4ddb9f623241> in getNearbyVenues(names, latitudes, longitudes, radius)
     16
     17         # make the GET request
---> 18         results = requests.get(url).json()["response"]['groups'][0]['items']
     19
     20         # return only relevant information for each nearby venue

KeyError: 'groups'
```

There seems to have ocurred an error, exactly after region 'Coyoacan' was searched, so, because there is for loop that would read the next value, we assume that 'Coyoacan' is the region giving us trouble. Let's drop it and try again.

In [273]:

```
mexico = mexico.drop(mexico.loc[mexico['Region'] == 'Coyoacán',].index[0],
                     axis = 0).reset_index(drop = True)
mexico
```

Out[273]:

	Region	Latitude	Longitude
0	Ciudad de México	19.42847	-99.12766
1	Iztapalapa	19.35529	-99.06224
2	Guadalajara	20.66682	-103.39182
3	Puebla	19.03793	-98.20346
4	Tijuana	32.50270	-117.00371
5	Monterrey	25.67507	-100.31847
6	Ecatepec de Morelos	19.60492	-99.06064
7	Chihuahu	28.63528	-106.08889
8	Naucalpan de Juárez	19.47851	-99.23963
9	Mérida	20.97537	-89.61696
10	San Luis	22.14982	-100.97916
11	Hermosillo	29.10260	-110.97732
12	Saltillo	25.42321	-101.00530
13	Mexicali	32.62781	-115.45446
14	Guadalupe	25.67678	-100.25646
15	Paso del Norte	31.72024	-106.46084
16	Cancún	21.17429	-86.84656
17	León de los Aldama	21.12908	-101.67374
18	Morelia	19.70078	-101.18443

In [99]:

```
mexico_venues = getNearbyVenues(names=mexico['Region'],
                                   latitudes=mexico['Latitude'],
                                   longitudes=mexico['Longitude'],
                                   radius = RADIUS
                                  )
```

Ciudad de México

```
Iztapalapa
Guadalajara
Puebla
Tijuana
Monterrey
Ecatepec de Morelos
Chihuahu
Naucalpan de Juárez
Mérida
San Luis
Hermosillo
Saltillo
Mexicali
Guadalupe
Paso del Norte
Cancún
León de los Aldama
Morelia
```

Great, perhaps what happened is that Coyoacan had no info to display (a litlle weird if you ask me, considering Coyoacan is one of Mexico City's most frequented neighborhood). Now that we have info for every region, let's look ate our data:

In [101]:

```python
print('The number of venues is {}'.format(mexico_venues.shape[0]))
mexico_venues.head()
```

```
The number of venues is 360
```

Out[101]:

	Neighborhood	Neighborhood Latitude	Neighborhood Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
0	Ciudad de México	19.42847	-99.12766	Al Andalus	19.427881	-99.129224	Middle Eastern Restaurant
1	Ciudad de México	19.42847	-99.12766	Casa Talavera	19.428149	-99.127677	Art Gallery
2	Ciudad de México	19.42847	-99.12766	El Antiguo Edhen	19.430340	-99.129250	Falafel Restaurant
3	Ciudad de México	19.42847	-99.12766	Barbacoa "El Genrry"	19.426894	-99.126917	Taco Place
4	Ciudad de México	19.42847	-99.12766	Tacos Don Chano	19.429156	-99.126608	Taco Place

Seems like we have a solid amount of venues to work with. Yet, let's see how many venues are in each region:

In [102]:

```python
mexico_venues.groupby('Neighborhood').count()
```

Out[102]:

	Neighborhood Latitude	Neighborhood Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
Neighborhood						
Cancún	20	20	20	20	20	20
Chihuahu	20	20	20	20	20	20
Ciudad de México	20	20	20	20	20	20
Ecatepec de Morelos	18	18	18	18	18	18
Guadalajara	20	20	20	20	20	20
Guadalupe	20	20	20	20	20	20

Neighborhood	Neighborhood Latitude	Neighborhood Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
Guadalupe	20	20	20	20	20	20
Hermosillo	20	20	20	20	20	20
Iztapalapa	20	20	20	20	20	20
León de los Aldama	15	15	15	15	15	15
Mexicali	20	20	20	20	20	20
Monterrey	20	20	20	20	20	20
Morelia	20	20	20	20	20	20
Mérida	20	20	20	20	20	20
Naucalpan de Juárez	20	20	20	20	20	20
Paso del Norte	7	7	7	7	7	7
Puebla	20	20	20	20	20	20
Saltillo	20	20	20	20	20	20
San Luis	20	20	20	20	20	20
Tijuana	20	20	20	20	20	20

Not bad at all! We obtained the maximum number on almost every region. Only 'Paso del Norte' got 7, so lets get rid of it. First of all, in case we mess up with our data and to avoid spending all of our queries, I will store a backup variable with the current venues. (I comment it just to avoid reruning the code and messing the backup)

In [103]:

```
# mexico_venues_backup = mexico_venues.copy()
mexico_venues_backup
```

Out[103]:

	Neighborhood	Neighborhood Latitude	Neighborhood Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
0	Ciudad de México	19.42847	-99.12766	Al Andalus	19.427881	-99.129224	Middle Eastern Restaurant
1	Ciudad de México	19.42847	-99.12766	Casa Talavera	19.428149	-99.127677	Art Gallery
2	Ciudad de México	19.42847	-99.12766	El Antiguo Edhen	19.430340	-99.129250	Falafel Restaurant
3	Ciudad de México	19.42847	-99.12766	Barbacoa "El Genrry"	19.426894	-99.126917	Taco Place
4	Ciudad de México	19.42847	-99.12766	Tacos Don Chano	19.429156	-99.126608	Taco Place
...	...	...	...	...	...	...	...
355	Morelia	19.70078	-101.18443	Gallepays	19.700678	-101.185996	Bakery
356	Morelia	19.70078	-101.18443	Plaza Villalongin	19.703223	-101.182666	Plaza
357	Morelia	19.70078	-101.18443	Centro 570	19.701935	-101.184549	Bar
358	Morelia	19.70078	-101.18443	La Piccola Italia	19.702020	-101.187748	Italian Restaurant
359	Morelia	19.70078	-101.18443	Casa De Los Dulces Suenos Hotel Morelia	19.702028	-101.187283	Hotel

360 rows × 7 columns

Great, now I drop 'Paso del Norte', which is equivalently to keep all regions that have over 7 venues.

In [150]:

In [150]:

```
aux = [mexico_venues.groupby('Neighborhood').count().Venue > 7][0]
mexico_venues = mexico_venues[mexico_venues['Neighborhood'].isin(aux[aux].index)]
print('The number of venues is {}'.format(mexico_venues.shape[0]))
mexico_venues.head()
```

```
The number of venues is 353
```

Out[150]:

	Neighborhood	Neighborhood Latitude	Neighborhood Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
0	Ciudad de México	19.42847	-99.12766	Al Andalus	19.427881	-99.129224	Middle Eastern Restaurant
1	Ciudad de México	19.42847	-99.12766	Casa Talavera	19.428149	-99.127677	Art Gallery
2	Ciudad de México	19.42847	-99.12766	El Antiguo Edhen	19.430340	-99.129250	Falafel Restaurant
3	Ciudad de México	19.42847	-99.12766	Barbacoa "El Genrry"	19.426894	-99.126917	Taco Place
4	Ciudad de México	19.42847	-99.12766	Tacos Don Chano	19.429156	-99.126608	Taco Place

Given we have solid dataset of departure regions (which could also be used as arrival ones), let's look at the arrival regions. For the moment, let's work with regions from 2 of my favorite countries: Japan and Germany.

In [126]:

```
arrival = pd.read_csv('others_coordinates.txt', header = None)
cols = [1,0,2,3]
arrival = arrival[cols]
arrival.columns = ['Country','Region','Latitude','Longitude']
arrival['Country'] = arrival['Country'].str.replace(" ","")
arrival['Region'] = arrival['Region'].str.replace(" ","")
arrival
```

Out[126]:

	Country	Region	Latitude	Longitude
0	Japan	Tokyo	35.652832	139.839478
1	Japan	Nagoya	35.183334	136.899994
2	Japan	Kitakyushu	33.883331	130.883331
3	Japan	Sendai	38.268223	140.869415
4	Japan	Hiroshima	34.383331	132.449997
5	Japan	Kawasaki	35.516666	139.699997
6	Japan	Kyoto	35.011665	135.768326
7	Japan	Kobe	34.689999	135.195557
8	Japan	Fukuoka	33.583332	130.399994
9	Japan	Sapporo	43.066666	141.350006
10	Japan	Osaka	34.669529	135.497009
11	Germany	SaxonSwitzerland	50.918072	14.315064
12	Germany	Friedrichshain	52.515816	13.454293
13	Germany	Praunheim	50.149334	8.618841
14	Germany	Kreuzberg	52.498604	13.391799
15	Germany	Mitte	52.531677	13.381777
16	Germany	Borsdorf	51.349701	12.542095

I need to convert the Latitude and Longitude to floats to pass them as arguments in the map function.

In [133]:

```
arrival[['Latitude','Longitude']] = arrival[['Latitude','Longitude']].astype(float)
```

For convinience, I will split these datasets by country. However, I will keep the index in case it comes useful

In [134]:

```
japan = arrival.loc[arrival['Country'] =='Japan',:]
germany = arrival.loc[arrival['Country'] =='Germany',:]
germany.head()
```

Out[134]:

	Country	Region	Latitude	Longitude
11	Germany	SaxonSwitzerland	50.918072	14.315064
12	Germany	Friedrichshain	52.515816	13.454293
13	Germany	Praunheim	50.149334	8.618841
14	Germany	Kreuzberg	52.498604	13.391799
15	Germany	Mitte	52.531677	13.381777

Let's visualize Japan first.

In [118]:

```
address = 'Tokyo, JP'

geolocator = Nominatim(user_agent="jp_explorer")
location = geolocator.geocode(address)
latitude = location.latitude
longitude = location.longitude
print('The geographical coordinate of Tokyo, Japan are {}, {}.'.format(latitude, longitud
e))
```

```
The geographical coordinate of Tokyo, Japan are 35.6828387, 139.7594549.
```

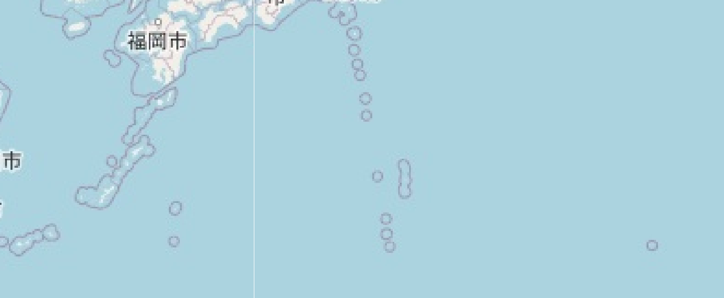
I create a map to visualize these different regions.

In [138]:

```
map_japan= folium.Map(location=[latitude, longitude], zoom_start=4)

# add markers to map
for lat, lng, region in zip(japan['Latitude'],
                            japan['Longitude'],
                            japan['Region']):
    label = '{}'.format(region)
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker(
        [lat, lng],
        radius=5,
        popup=label,
        color='blue',
        fill=True,
        fill_color='#3186cc',
        fill_opacity=0.7,
        parse_html=False).add_to(map_japan)
```

ust Notebook



Now Germany

In [140]:

```
address = 'Berlin'

geolocator = Nominatim(user_agent="gm_explorer")
location = geolocator.geocode(address)
latitude = location.latitude
longitude = location.longitude
print('The geographical coordinate of Berlin, Germany are {}, {}.'.format(latitude, longit
ude))
```

```
The geographical coordinate of Berlin, Germany are 52.5170365, 13.3888599.
```

I create a map to visualize these different regions.

In [142]:

```
map_germany= folium.Map(location=[latitude, longitude], zoom_start=6)

# add markers to map
for lat, lng, region in zip(germany['Latitude'],
                            germany['Longitude'],
                            germany['Region']):
                                                       se_html=True)

                                                      _germany)
```

Make this Notebook Trusted to load map: File -> Trust Notebook

Let's proceed the same way we did for Mexico, now for both of these countries.

In [143]:

```
japan_venues = getNearbyVenues(names=japan['Region'],
                                   latitudes=japan['Latitude'],
                                   longitudes=japan['Longitude'],
                                   radius = RADIUS
                                  )
```

```
Tokyo
Nagoya
Kitakyushu
Sendai
Hiroshima
Kawasaki
Kyoto
Kobe
Fukuoka
Sapporo
Osaka
```

In [144]:

```
germany_venues = getNearbyVenues(names=germany['Region'],
                                   latitudes=germany['Latitude'],
                                   longitudes=germany['Longitude'],
                                   radius = RADIUS
                                  )
```

```
SaxonSwitzerland
Friedrichshain
Praunheim
Kreuzberg
Mitte
Borsdorf
```

Great, there seems to have been no problem as there was swith the Mexico set. Let's take a look at what we are working with.

In [145]:

```
print('The number of venues is {}'.format(japan_venues.shape[0]))
japan_venues.head()
```

```
The number of venues is 220
```

Out[145]:

	Neighborhood	Neighborhood Latitude	Neighborhood Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
0	Tokyo	35.652832	139.839478	Yumenoshima Tropical Greenhouse Dome (夢の島熱帯植物館)	35.651290	139.829489	Botanical Garden
1	Tokyo	35.652832	139.839478	イーノの森 ドッグガーデン	35.650601	139.836001	Dog Run
2	Tokyo	35.652832	139.839478	BumB 東京スポーツ文化館	35.649227	139.829909	Gym
3	Tokyo	35.652832	139.839478	Lawson (ローソン 新木場一丁目店)	35.645752	139.835058	Convenience Store
4	Tokyo	35.652832	139.839478	新砂船着場	35.654231	139.842202	Boat or Ferry

In [146]:

```
print('The number of venues is {}'.format(germany_venues.shape[0]))
germany_venues.head()
```

```
The number of venues is 82
```

Out[146]:

	Neighborhood	Neighborhood Latitude	Neighborhood Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
0	SaxonSwitzerland	50.918072	14.315064	Großes Pohlshorn	50.918847	14.314819	Mountain
1	Friedrichshain	52.515816	13.454293	K. LIEBLINGs / Coffee Profilers	52.515816	13.451219	Coffee Shop
2	Friedrichshain	52.515816	13.454293	Shakespeare and Sons	52.512636	13.453113	Bookstore
3	Friedrichshain	52.515816	13.454293	Protokoll	52.513075	13.457071	Pub
4	Friedrichshain	52.515816	13.454293	Fine Bagels	52.512644	13.453135	Bagel Shop

There is a good amount of data, let's just take a look of the amount of venues per region, so as to see if we should drop one.

In [147]:

```
japan_venues.groupby('Neighborhood').count()
```

Out[147]:

Neighborhood	Neighborhood Latitude	Neighborhood Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
Fukuoka	20	20	20	20	20	20
Hiroshima	20	20	20	20	20	20
Kawasaki	20	20	20	20	20	20
Kitakyushu	20	20	20	20	20	20
Kobe	20	20	20	20	20	20
Kyoto	20	20	20	20	20	20
Nagoya	20	20	20	20	20	20
Osaka	20	20	20	20	20	20
Sapporo	20	20	20	20	20	20
Sendai	20	20	20	20	20	20
Tokyo	20	20	20	20	20	20

In [148]:

```
germany_venues.groupby('Neighborhood').count()
```

Out[148]:

	Neighborhood Latitude	Neighborhood Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
Neighborhood						
Borsdorf	5	5	5	5	5	5
Friedrichshain	20	20	20	20	20	20
Kreuzberg	20	20	20	20	20	20
Mitte	20	20	20	20	20	20
Praunheim	16	16	16	16	16	16
SaxonSwitzerland	1	1	1	1	1	1

How interesting. Its seems the Foursquare database in Japan is quite complete (every region had it's limit venues capacity, 20, complete), whereas the Germany one is a bit lacking. Were we to keep "SaxonSwitzerland" and "Borsdorf" regions, the clustering algorithm might have outliers that make it underperform. Let's remove them. But again, we make backups were we to in the end decide to us all of the data.

In [149]:

```
# japan_venues_backup = japan_venues.copy()
# germany_venues_backup = germany_venues.copy()
```

Great, now I drop the 2 German regions, which is equivalently to keep all regions that have over 5 venues.

In [152]:

```
aux = [germany_venues.groupby('Neighborhood').count().Venue > 5][0]
germany_venues = germany_venues[germany_venues['Neighborhood'].isin(aux[aux].index)]
print('The number of venues is {}'.format(germany_venues.shape[0]))
germany_venues.head()
```

```
The number of venues is 76
```

Out[152]:

	Neighborhood	Neighborhood Latitude	Neighborhood Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
1	Friedrichshain	52.515816	13.454293	K. LIEBLINGs / Coffee Profilers	52.515816	13.451219	Coffee Shop
2	Friedrichshain	52.515816	13.454293	Shakespeare and Sons	52.512636	13.453113	Bookstore
3	Friedrichshain	52.515816	13.454293	Protokoll	52.513075	13.457071	Pub
4	Friedrichshain	52.515816	13.454293	Fine Bagels	52.512644	13.453135	Bagel Shop
5	Friedrichshain	52.515816	13.454293	Chay Village	52.513509	13.458474	Vegetarian / Vegan Restaurant

Now that we have the datasets for every country, I go ahead and create dummy variables for every category, on every country, as well as obtain the frequency of each category. I will use the code from the lab of week 3, given its simplicity and effectivity.

In [153]:

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

```
# add neighborhood column back to dataframe
mexico_onehot['Neighborhood'] = mexico_venues['Neighborhood']

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

print('The number of different categories are {}.'.format(mexico_onehot.shape[1]-1))
mexico_probabilities = mexico_onehot.groupby('Neighborhood').mean().reset_index()
mexico_probabilities.head()
```

```
The number of different categories are 98.
```

Out[153]:

	Neighborhood	Argentinian Restaurant	Art Gallery	Art Museum	Arts & Crafts Store	Asian Restaurant	Bakery	Bar	Bed & Breakfast	Beer Garden	...	Stationery Store	Stea
0	Cancún	0.0	0.0	0.0	0.0	0.00	0.00	0.000000	0.0	0.0	...	0.0	
1	Chihuahu	0.0	0.0	0.0	0.0	0.00	0.00	0.000000	0.0	0.0	...	0.0	
2	Ciudad de México	0.0	0.1	0.0	0.1	0.00	0.00	0.000000	0.0	0.0	...	0.0	
3	Ecatepec de Morelos	0.0	0.0	0.0	0.0	0.00	0.00	0.055556	0.0	0.0	...	0.0	
4	Guadalajara	0.1	0.0	0.0	0.0	0.05	0.05	0.000000	0.0	0.0	...	0.0	

5 rows × 99 columns

In [154]:

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

# add neighborhood column back to dataframe
japan_onehot['Neighborhood'] = japan_venues['Neighborhood']

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

print('The number of different categories are {}.'.format(japan_onehot.shape[1]-1))
japan_probabilities = japan_onehot.groupby('Neighborhood').mean().reset_index()
japan_probabilities.head()
```

```
The number of different categories are 103.
```

Out[154]:

	Neighborhood	American Restaurant	Art Gallery	Arts & Crafts Store	Athletics & Sports	Australian Restaurant	BBQ Joint	Bakery	Bar	Baseball Field	...	Tempura Restaurant	Theater	Re
0	Fukuoka	0.0	0.0	0.0	0.0	0.05	0.05	0.00	0.00	0.0	...	0.00	0.0	
1	Hiroshima	0.0	0.0	0.0	0.0	0.00	0.00	0.00	0.00	0.0	...	0.00	0.0	
2	Kawasaki	0.0	0.0	0.0	0.0	0.00	0.00	0.00	0.00	0.0	...	0.00	0.0	
3	Kitakyushu	0.0	0.0	0.0	0.0	0.00	0.05	0.05	0.05	0.0	...	0.05	0.0	
4	Kobe	0.0	0.0	0.0	0.0	0.00	0.00	0.10	0.00	0.0	...	0.00	0.0	

5 rows × 104 columns

In [155]:

```
# one hot encoding
germany_onehot = pd.get_dummies(germany_venues[['Venue Category']], prefix="", prefix_se
p="")

# add neighborhood column back to dataframe
germany_onehot['Neighborhood'] = germany_venues['Neighborhood']

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

print('The number of different categories are {}.'.format(germany_onehot.shape[1]-1))
germany_probabilities = germany_onehot.groupby('Neighborhood').mean().reset_index()
germany_probabilities.head()
```

```
The number of different categories are 55.
```

Out[155]:

	Neighborhood	Art Gallery	Art Museum	Bagel Shop	Beer Store	Bistro	Bookstore	Breakfast Spot	Café	Cemetery	...	Supermarket	Taverna	R
0	Friedrichshain	0.0	0.00	0.05	0.05	0.0	0.05	0.00	0.05	0.00	...	0.0000	0.00	
1	Kreuzberg	0.1	0.05	0.00	0.00	0.1	0.00	0.00	0.00	0.00	...	0.0000	0.05	
2	Mitte	0.0	0.00	0.00	0.00	0.0	0.00	0.05	0.05	0.05	...	0.0000	0.00	
3	Praunheim	0.0	0.00	0.00	0.00	0.0	0.00	0.00	0.00	0.00	...	0.1875	0.00	

4 rows × 56 columns

Now we face with a problem: Not every category is contained in every dataset. For this reason, I will need to create 2 datasets, one that cointains every shared category bteween Mexico and Japan, and one that does the same but with Mexico and Germany.

In [228]:

```
#I make an intersection of every venue that is in both country.
mex_jap = set(mexico_probabilities.columns.values).intersection(set(japan_probabilities.
columns.values))

#I create 2 auxiliary datasets that filter over the intersection created above, and I als
o
#order them so that I can easily append both of them
mex_jap_aux1 = mexico_probabilities.loc[:,list(mex_jap)].reindex(
    sorted(mexico_probabilities.loc[:,list(mex_jap)].columns), axis=1)
mex_jap_aux2 = japan_probabilities.loc[:,list(mex_jap)].reindex(
    sorted(japan_probabilities.loc[:,list(mex_jap)].columns), axis=1)

#I append both of them making sure to first move the Neighborhood columnup front
mex_jap = pd.concat([pd.DataFrame(mex_jap_aux1.loc[:,'Neighborhood']),
           mex_jap_aux1.drop('Neighborhood',1)], axis = 1).append(
         pd.concat([pd.DataFrame(mex_jap_aux2.loc[:,'Neighborhood']),
           mex_jap_aux2.drop('Neighborhood',1)], axis = 1))

mex_jap.reset_index(inplace = True, drop = True)
mex_jap
```

Out[228]:

	Neighborhood	Art Gallery	Arts & Crafts Store	Bakery	Bar	Bookstore	Burger Joint	Café	Chinese Restaurant	Clothing Store	...	Pub	Public Art	Res
0	Cancún	0.00	0.00	0.000000	0.000000	0.00	0.000000	0.00	0.00	0.00	...	0.00	0.00	
1	Chihuahu	0.00	0.00	0.000000	0.000000	0.00	0.000000	0.00	0.00	0.00	...	0.00	0.00	

	Neighborhood	Art Gallery	Arts & Crafts Store	Bakery	Bar	Bookstore	Burger Joint	Café	Chinese Restaurant	Clothing Store	...	Pub	Public Art	Res
1	Chihuahua	0.00	0.00	0.000000	0.000000	0.00	0.000000	0.00	0.00	0.00	...	0.00	0.00	
2	Ciudad de México	0.10	0.10	0.000000	0.000000	0.00	0.000000	0.00	0.00	0.08	...	0.00	0.00	
3	Ecatepec de Morelos	0.00	0.00	0.000000	0.055556	0.00	0.055556	0.00	0.00	0.00	...	0.00	0.00	
4	Guadalajara	0.00	0.00	0.050000	0.000000	0.00	0.000000	0.00	0.00	0.00	...	0.00	0.00	
5	Guadalupe	0.00	0.00	0.050000	0.000000	0.00	0.100000	0.05	0.00	0.00	...	0.00	0.00	
6	Hermosillo	0.00	0.00	0.000000	0.000000	0.00	0.000000	0.00	0.05	0.00	...	0.00	0.00	
7	Iztapalapa	0.00	0.00	0.000000	0.000000	0.00	0.000000	0.00	0.00	0.00	...	0.00	0.00	
8	León de los Aldama	0.00	0.00	0.066667	0.066667	0.00	0.066667	0.00	0.00	0.00	...	0.00	0.00	
9	Mexicali	0.00	0.00	0.050000	0.000000	0.00	0.000000	0.10	0.00	0.00	...	0.00	0.05	
10	Monterrey	0.00	0.00	0.000000	0.000000	0.00	0.000000	0.00	0.00	0.00	...	0.00	0.00	
11	Morelia	0.00	0.00	0.050000	0.200000	0.00	0.000000	0.00	0.00	0.00	...	0.00	0.00	
12	Mérida	0.00	0.00	0.050000	0.050000	0.00	0.000000	0.00	0.00	0.00	...	0.00	0.00	
13	Naucalpan de Juárez	0.00	0.05	0.000000	0.000000	0.00	0.000000	0.00	0.00	0.00	...	0.00	0.00	
14	Puebla	0.00	0.00	0.050000	0.050000	0.00	0.050000	0.05	0.00	0.00	...	0.00	0.00	
15	Saltillo	0.20	0.00	0.050000	0.000000	0.00	0.000000	0.00	0.00	0.00	...	0.10	0.00	
16	San Luis	0.00	0.00	0.000000	0.100000	0.05	0.050000	0.05	0.00	0.00	...	0.00	0.00	
17	Tijuana	0.00	0.00	0.050000	0.000000	0.00	0.000000	0.00	0.00	0.00	...	0.00	0.00	
18	Fukuoka	0.00	0.00	0.000000	0.000000	0.05	0.050000	0.05	0.00	0.00	...	0.00	0.00	
19	Hiroshima	0.00	0.00	0.000000	0.000000	0.00	0.000000	0.05	0.00	0.00	...	0.00	0.00	
20	Kawasaki	0.00	0.00	0.000000	0.000000	0.05	0.000000	0.00	0.00	0.00	...	0.00	0.00	
21	Kitakyushu	0.00	0.00	0.050000	0.050000	0.05	0.000000	0.00	0.00	0.05	...	0.00	0.00	
22	Kobe	0.00	0.00	0.100000	0.000000	0.05	0.000000	0.05	0.05	0.00	...	0.00	0.00	
23	Kyoto	0.00	0.05	0.000000	0.100000	0.00	0.000000	0.10	0.00	0.00	...	0.00	0.00	
24	Nagoya	0.00	0.00	0.000000	0.000000	0.00	0.050000	0.00	0.00	0.00	...	0.00	0.00	
25	Osaka	0.00	0.00	0.050000	0.000000	0.05	0.000000	0.05	0.00	0.00	...	0.00	0.05	
26	Sapporo	0.00	0.00	0.000000	0.050000	0.05	0.000000	0.10	0.00	0.00	...	0.00	0.00	
27	Sendai	0.05	0.00	0.000000	0.000000	0.00	0.050000	0.05	0.05	0.00	...	0.05	0.00	
28	Tokyo	0.00	0.00	0.000000	0.000000	0.00	0.000000	0.00	0.00	0.00	...	0.00	0.00	

29 rows × 36 columns

In [229]:

```
#I make an intersection of every venue that is in both country.
mex_ger = set(mexico_probabilities.columns.values).intersection(set(germany_probabilitie
s.columns.values))

#I create 2 auxiliary datasets that filter over the intersection created above, and I als
o
#order them so that I can easily append both of them
mex_ger_aux1 = mexico_probabilities.loc[:,list(mex_ger)].reindex(
    sorted(mexico_probabilities.loc[:,list(mex_ger)].columns), axis=1)
mex_ger_aux2 = germany_probabilities.loc[:,list(mex_ger)].reindex(
    sorted(germany_probabilities.loc[:,list(mex_ger)].columns), axis=1)

#I append both of them making sure to first move the Neighborhood columnup front
mex_ger = pd.concat([pd.DataFrame(mex_ger_aux1.loc[:,'Neighborhood']),
              mex_ger_aux1.drop('Neighborhood',1)], axis = 1).append(
          pd.concat([pd.DataFrame(mex_ger_aux2.loc[:,'Neighborhood']),
              mex_ger_aux2.drop('Neighborhood',1)], axis = 1))
```

```
mex_ger.reset_index(inplace = True, drop = True)
mex_ger
```

Out[229]:

	Neighborhood	Art Gallery	Art Museum	Bistro	Bookstore	Breakfast Spot	Café	Chinese Restaurant	Coffee Shop	Creperie	...	Ice Cream Shop	I Resta
0	Cancún	0.0	0.00	0.000000	0.00	0.00	0.00	0.0000	0.000000	0.00	...	0.05	
1	Chihuahu	0.0	0.00	0.000000	0.00	0.00	0.00	0.0000	0.200000	0.00	...	0.05	
2	Ciudad de México	0.1	0.00	0.000000	0.00	0.00	0.00	0.0000	0.050000	0.00	...	0.00	
3	Ecatepec de Morelos	0.0	0.00	0.000000	0.00	0.00	0.00	0.0000	0.055556	0.00	...	0.00	
4	Guadalajara	0.0	0.00	0.050000	0.00	0.05	0.00	0.0000	0.000000	0.00	...	0.00	
5	Guadalupe	0.0	0.00	0.000000	0.00	0.00	0.05	0.0000	0.000000	0.00	...	0.00	
6	Hermosillo	0.0	0.00	0.000000	0.00	0.00	0.00	0.0500	0.000000	0.00	...	0.00	
7	Iztapalapa	0.0	0.00	0.000000	0.00	0.00	0.00	0.0000	0.000000	0.00	...	0.00	
8	León de los Aldama	0.0	0.00	0.066667	0.00	0.00	0.00	0.0000	0.000000	0.00	...	0.00	
9	Mexicali	0.0	0.00	0.000000	0.00	0.00	0.10	0.0000	0.150000	0.00	...	0.05	
10	Monterrey	0.0	0.00	0.000000	0.00	0.00	0.00	0.0000	0.000000	0.00	...	0.00	
11	Morelia	0.0	0.00	0.000000	0.00	0.00	0.00	0.0000	0.000000	0.00	...	0.10	
12	Mérida	0.0	0.05	0.000000	0.00	0.05	0.00	0.0000	0.050000	0.00	...	0.05	
13	Naucalpan de Juárez	0.0	0.00	0.000000	0.00	0.00	0.00	0.0000	0.000000	0.00	...	0.05	
14	Puebla	0.0	0.00	0.000000	0.00	0.00	0.05	0.0000	0.050000	0.00	...	0.15	
15	Saltillo	0.2	0.05	0.000000	0.00	0.00	0.00	0.0000	0.050000	0.00	...	0.00	
16	San Luis	0.0	0.00	0.000000	0.05	0.05	0.05	0.0000	0.000000	0.00	...	0.00	
17	Tijuana	0.0	0.00	0.000000	0.00	0.00	0.00	0.0000	0.100000	0.05	...	0.00	
18	Friedrichshain	0.0	0.00	0.000000	0.05	0.00	0.05	0.0000	0.050000	0.05	...	0.10	
19	Kreuzberg	0.1	0.05	0.100000	0.00	0.00	0.00	0.0000	0.000000	0.00	...	0.00	
20	Mitte	0.0	0.00	0.000000	0.00	0.05	0.05	0.0000	0.200000	0.00	...	0.00	
21	Praunheim	0.0	0.00	0.000000	0.00	0.00	0.00	0.0625	0.000000	0.00	...	0.00	

22 rows × 26 columns

Now, we will get the top 5 venue categories for every region, so that we can use them later to analyse what the k means algorithm tells us. I will borrow the code from the lab on this section, adjusted to work on both combined datasets.

In [230]:

```
def return_most_common_venues(row, num_top_venues):
    row_categories = row.iloc[1:]
    row_categories_sorted = row_categories.sort_values(ascending=False)

    return row_categories_sorted.index.values[0:num_top_venues]
```

In [234]:

```
num_top_venues = 5

indicators = ['st', 'nd', 'rd']

# create columns according to number of top venues
```

```
columns = ['Neighborhood']
for ind in np.arange(num_top_venues):
    try:
        columns.append('{}{} Most Common Venue'.format(ind+1, indicators[ind]))
    except:
        columns.append('{}th Most Common Venue'.format(ind+1))

# create a new dataframe
neighborhoods_venues_sorted = pd.DataFrame(columns=columns)
neighborhoods_venues_sorted['Neighborhood'] = mex_jap['Neighborhood']

for ind in np.arange(mex_jap.shape[0]):
    neighborhoods_venues_sorted.iloc[ind, 1:] = return_most_common_venues(mex_jap.iloc[i
nd, :], num_top_venues)

mex_jap_neighborhoods_venues_sorted =  neighborhoods_venues_sorted
mex_jap_neighborhoods_venues_sorted.head()
```

Out[234]:

	Neighborhood	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue
0	Cancún	Seafood Restaurant	Ice Cream Shop	Sushi Restaurant	Supermarket	Soccer Field
1	Chihuahu	Coffee Shop	Ice Cream Shop	Italian Restaurant	Seafood Restaurant	Paper / Office Supplies Store
2	Ciudad de México	Art Gallery	Arts & Crafts Store	Coffee Shop	Restaurant	Museum
3	Ecatepec de Morelos	Pizza Place	Soccer Field	Seafood Restaurant	Bar	Burger Joint
4	Guadalajara	Pizza Place	Hotel	Bakery	Seafood Restaurant	Italian Restaurant

In [240]:

```
# create a new dataframe
neighborhoods_venues_sorted = pd.DataFrame(columns=columns)
neighborhoods_venues_sorted['Neighborhood'] = mex_ger['Neighborhood']

for ind in np.arange(mex_ger.shape[0]):
    neighborhoods_venues_sorted.iloc[ind, 1:] = return_most_common_venues(mex_ger.iloc[i
nd, :], num_top_venues)

mex_ger_neighborhoods_venues_sorted =  neighborhoods_venues_sorted
mex_ger_neighborhoods_venues_sorted.head()
```

Out[240]:

	Neighborhood	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue
0	Cancún	Sandwich Place	Pharmacy	Ice Cream Shop	Supermarket	Vietnamese Restaurant
1	Chihuahu	Coffee Shop	Pharmacy	Italian Restaurant	Ice Cream Shop	Vietnamese Restaurant
2	Ciudad de México	Middle Eastern Restaurant	Art Gallery	Restaurant	Coffee Shop	History Museum
3	Ecatepec de Morelos	Pharmacy	Gym / Fitness Center	Coffee Shop	French Restaurant	Art Museum
4	Guadalajara	Bistro	Breakfast Spot	Italian Restaurant	Hotel	Vietnamese Restaurant

We can see that, because we made the intersection, now every region has different most frequent venues. That is great, and just like we expected, for we want to look the similarities between countries, and having made, for example, dummy columns for every category and assign it with 0's, might make the model give unwanted

results.

We are now ready to make the k means procedure. The reason I will use this method, is because once the regions are assigned to clusters, the user can choose between moving to a certain region of a country that ir more like their own, or choose to go to one that is actually quite different, maybe in the search of an adventure.

Now, the user should select where in Mexico they live, and whether they want to go to Japan or Germany, so that the app removes every line whose Country is Mexico other than the region selected, and make the k means analysis between that region and the selected country. For testing purposes, I will choose 'Ciudad de México' (Mexico City), which is where I live, and Japan,which is where I would mostly like to go live to.

In [337]:

```
#Naming the variables we will use
home = 'Ciudad de México'
destiny = 'Japan'

#I add a column to identify the regions from mexico.
mexico2 = pd.concat([pd.DataFrame(pd.Series('México').repeat(19)).reset_index(drop=True)
,mexico], axis = 1)
mexico2.rename(columns={0:'Country'}, inplace=True)

#Now, we create the only numerical array that will be passed to the k means function.
#First, we check whether the user wants to go to Japan or Germany
if(destiny == 'Japan'):
    temp1 = mex_jap
    temp2 = japan
    temp3 = mex_jap_neighborhoods_venues_sorted
else:
    temp = mex_ger
    temp2 = germany
    temp3 = mex_ger_neighborhoods_venues_sorted

#Note that, as of now it might seem uneccesary to make this if else statement, but when t
he app
#gets fully developped, then it will be necessary to even make cases.

#I make an aux dataset that links the Country for the time given, just to make it easier
to drop rows
aux = mexico2.rename(columns={'Region':'Neighborhood'})[['Country','Neighborhood']].appe
nd(
        temp2.rename(columns={'Region':'Neighborhood'})[['Country','Neighborhood']])
aux = aux.merge(temp1, on='Neighborhood', how='right')

#Now I drop every Mexico row that isn't the home one.
aux = aux.loc[aux['Neighborhood'] == home,:].append(
    aux[~aux['Country'].isin(['México'])])

aux
```

Out[337]:

	Country	Neighborhood	Art Gallery	Arts & Crafts Store	Bakery	Bar	Bookstore	Burger Joint	Café	Chinese Restaurant	...	Pub	Public Art	Restaurant
0	México	Ciudad de México	0.10	0.10	0.00	0.00	0.00	0.00	0.00	0.00	...	0.00	0.00	0.05
18	Japan	Tokyo	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.00	0.00	0.00
19	Japan	Nagoya	0.00	0.00	0.00	0.00	0.00	0.05	0.00	0.00	...	0.00	0.00	0.05
20	Japan	Kitakyushu	0.00	0.00	0.05	0.05	0.05	0.00	0.00	0.00	...	0.00	0.00	0.00
21	Japan	Sendai	0.05	0.00	0.00	0.00	0.00	0.05	0.05	0.05	...	0.05	0.00	0.00
22	Japan	Hiroshima	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.00	...	0.00	0.00	0.00
23	Japan	Kawasaki	0.00	0.00	0.00	0.00	0.05	0.00	0.00	0.00	...	0.00	0.00	0.05
24	Japan	Kyoto	0.00	0.05	0.00	0.10	0.00	0.00	0.10	0.00	...	0.00	0.00	0.00

	Country	Neighborhood	Art Gallery	Arts & Crafts Store	Bakery	Bar	Bookstore	Burger Joint	Café	Chinese Restaurant	...	Pub	Public Art	Restaurant
24	Japan	Kyoto	0.00	0.05	0.00	0.10	0.00	0.00	0.10	0.00	...	0.00	0.00	0.00
25	Japan	Kobe	0.00	0.00	0.10	0.00	0.05	0.00	0.05	0.05	...	0.00	0.00	0.00
26	Japan	Fukuoka	0.00	0.00	0.00	0.00	0.05	0.05	0.05	0.00	...	0.00	0.00	0.00
27	Japan	Sapporo	0.00	0.00	0.00	0.05	0.05	0.00	0.10	0.00	...	0.00	0.00	0.00
28	Japan	Osaka	0.00	0.00	0.05	0.00	0.05	0.00	0.05	0.00	...	0.00	0.05	0.00

12 rows × 37 columns

Now, we are able to perform the k means procedure. Ideally, we would want to perform several iterations of the k means, and the results (region recommendation to the user), would be the one region that appeared the most times on every analysis. However, given the scope of this project, we will make only one run, separating the data into 3 clusters.

In [338]:

```
k= 3
aux_k = aux.drop(['Country','Neighborhood'], 1)

# run k-means clustering
kmeans = KMeans(n_clusters=k, random_state=7).fit(aux_k)

# add clustering labels
aux2 = aux
aux2.insert(0, 'Cluster Labels', kmeans.labels_)

# merge toronto_grouped with toronto_data to add latitude/longitude for each neighborhood
result = aux2[['Country','Neighborhood','Cluster Labels']].merge(temp3.set_index('Neighbo
rhood'), on='Neighborhood')
```

Results

The following is the result of applying the k means procedure to the dataset given the users inputs

In [339]:

```
result
```

Out[339]:

	Country	Neighborhood	Cluster Labels	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue
0	México	Ciudad de México	0	Art Gallery	Arts & Crafts Store	Coffee Shop	Restaurant	Museum
1	Japan	Tokyo	0	Convenience Store	Gym	Soccer Field	Park	Wings Joint
2	Japan	Nagoya	0	Japanese Restaurant	Historic Site	Coffee Shop	Hotel	Italian Restaurant
3	Japan	Kitakyushu	0	Hotel	Japanese Restaurant	Bakery	Bar	Bookstore
4	Japan	Sendai	1	Japanese Restaurant	Dessert Shop	Art Gallery	Burger Joint	Chinese Restaurant
5	Japan	Hiroshima	0	Coffee Shop	Hotel	Tea Room	History Museum	Café
6	Japan	Kawasaki	2	Convenience Store	Supermarket	Bookstore	Restaurant	Pizza Place
7	Japan	Kyoto	0	Hotel	Bar	Café	Paper / Office Supplies Store	Arts & Crafts Store
8	Japan	Kobe	1	Bakery	Ice Cream Shop	Café	Japanese Restaurant	Museum
9	Japan	Fukuoka	1	Seafood	Pizza Place	Café	Sushi Restaurant	Gift Shop

	Country	Neighborhood	Cluster Labels	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue
9	Japan	Fukuoka	1	Restaurant	Pizza Place	Café	Sushi Restaurant	Gift Shop
10	Japan	Sapporo	1	Café	Sushi Restaurant	Seafood Restaurant	Bar	Bookstore
11	Japan	Osaka	1	Wings Joint	Bakery	Bookstore	Japanese Restaurant	Café

We can see that our home location was assigned to cluster 0, along other regions. For the time given, we will present all these options to the costumer, along with the top 5 venues, so that the user can decide which fits best his desire.

In [391]:

```
same_cluster = result.loc[result['Cluster Labels'] ==
                          result.loc[result['Neighborhood'] == home]['Cluster Labels'][0
]]
same_cluster
```

Out[391]:

	Country	Neighborhood	Cluster Labels	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue
0	México	Ciudad de México	0	Art Gallery	Arts & Crafts Store	Coffee Shop	Restaurant	Museum
1	Japan	Tokyo	0	Convenience Store	Gym	Soccer Field	Park	Wings Joint
2	Japan	Nagoya	0	Japanese Restaurant	Historic Site	Coffee Shop	Hotel	Italian Restaurant
3	Japan	Kitakyushu	0	Hotel	Japanese Restaurant	Bakery	Bar	Bookstore
5	Japan	Hiroshima	0	Coffee Shop	Hotel	Tea Room	History Museum	Café
7	Japan	Kyoto	0	Hotel	Bar	Café	Paper / Office Supplies Store	Arts & Crafts Store

Discussion

There are a lot of areas to improve, and many ideas that can be implemented so that this worked to its fullest potential. Here are some ideas, that could be added were the company willing to invest in the project:

1. **Select any initial location (the location coordinates would be acquired through an API)**
2. **Select any country as destination (same as the above)**
3. **Give a list of highly valued venues (with this, the search could be narrowed and be more accurate given the desire of the customer)**
4. **Try to select places that have high amounts of hotels (this could interest hotel managers in investing)**
5. **Give the user visualizations of the venues in their recommended locations (bar graphs that show the amount of any given category type).**
6. **The K Means could be repeated several times, and the results that appear multiple times would be the ones more likely to be recommended.**
7. **Implement the amusement-to-comfort factor, so that the user could decide how much more different he or she wants the destiny location to be compared to home.**
8. **Overall, generalize everything in the code, so that it can be used by only inputing key words like 'Japan', 'Mexico', 'Food', etc.**

Conclusion

About the project, I feel it has great potential if time is spent on it. However, to fully make use of it, money would also be needed, to get a full Foursquare liscence, and to hire an expert on app development who could make a user interface.

About the whole specialization, I feel like I learned a lot, and I am eager to keep learning more about everything

that revolves Data Science and Machine Learning.