The Battle of Neighgbourhood

Introduction

The goal of this analysis is to compare different big cities in the world to check their similarity. The similarity will be calsulated using the list of venues retrieved with foursquare analyzing the categories of venues contained in a city and their ditribution within the city area.

Questions that we want to ask with this analysis are: Can city of the world be clustered in groups based just on the type of venues contained in them? Does the result of this grouping make sense? Are those groups related to geography (i.e. all europeans city will belong to the same group)? Can we create a classification of city districts that span across different cities (e.g. will all city have zones with restaurants other with museums etc.. is the venue category distribution a characteristics that repeats in different cities)?

Analisys method

In order to perform this analysis we plan to take a number of cities and use foursquare to fetch the venues that are contained in the cities. Each city will be divided in a regular grid of **Search Spots**. The search spots are the points that will be used to search for venues around. Each search spot togheter with the radius used for the search will become a **City Zone**

The search spots are placed in a regular triangular grid and the radius of the search will be one third of the distrance between points.The points will be distanciated 1000 meters and the search radius of each spot will be 666 meters.

Each spot is a zone of the city for wich a profile will be contructed using the frequencies of the venues categories.

All the zones of all the cities considered will be divided in a number of groups based on the venues category frequency.

Then each city will have a profile based on the distribution of the zone categories and city will be grouped this way. City that will belong to the same group being "more similar"

The data

In this section we will create the code that will be used to fetch the data that will be required for the analysis.

Each city that will be included in the analysis will have a reference point and search grid dimension. The search grid dimension will tell how many search point will be used to difide the city in zones and to get the venues for each zone using the foursquare explpore API.

The grid will be limited and catch mostly the central part of the cities to limit the number of foursquare queries that will be necessary (this is due to the limitations of the free plan that we are using and that limits the number of queries to 950 per day)

The result of this activity will be a DataFrame for each city containing the teched venues and the cooridnates of the zone (search point) used to fetch the venue.

The search zone of one search point has an overlapping with the zone of other adiacent search points, a venue is associeted to the first search point that will intercept it in the search. If the same venue will be fetched by a search in another search point it will be ignored since the venue has already been associated to a zone and cannot belong to multiple zones.

starting data

The datasets that will created through foursquare queries have the following format

```
import pandas as pd
df=pd.read_csv('./data/cleaned/sample.csv')
df.head(3)
```

search_spot_lat

search_spot_lon

search_radius

id

Name

Latitude

Longitude

Category

0

41.925131

12.538995

666.666667

4ca7816e76d3a093554e0c6b

Mejo De Betto E Mary

41.925766

12.539925

Roman Restaurant

1

41.925131

12.538995

666.666667

52b0a30f11d20648deaa4034

Inofficina

41.925485

12.535250

Gastropub

2

41.925131

12.538995

666.666667

4b0d105cf964a5208a4323e3

Lanificio 159

41.926047

12.539253

Performing Arts Venue

Each line represents a venue in the city . `(search_spot_lat , search_spot_lon)` are the coordinates of the center of the zone in wich the venue is inlcuded. The zone is a circle with radius: `search_radius` centered in that point. The `id` is the foursquare id of the venue it identifies the venue. `Name` is the name of the venue `Latitude,Longitude` are the absolute geographical coordinates of the venue `Category` is the Category of the venue attributed by foursquare

Data aquisition

Hereafter is the code to fetch the starting data for our analysis

Put your own CLIENT_ID and CLIENT_SECRET below if you want to execute the data fetch from foursquare, or put a foursquare.properties file in your home diretcory

```
CLIENT_ID = 'XXXX' # your Foursquare ID
CLIENT_SECRET = 'XXX' # your Foursquare Secret
VERSION = '20180604'
LIMIT = 30
```

```python
import configparser
import os
try:
    foursquare_property_file_path=os.environ['HOME']+os.path.sep+'foursquare.properties'
    config = configparser.ConfigParser()
    config.read(foursquare_property_file_path)
    CLIENT_ID=config['foursquare']['CLIENT_ID']
    CLIENT_SECRET=config['foursquare']['CLIENT_SECRET']
    #print('CLIENT_ID:{}'.format(CLIENT_ID))
    #print('CLIENT_SECRET:{}'.format(CLIENT_SECRET))
except IOError:
    print("no foursquare property file")
```

```python
import requests
import pandas as pd
#import geocoder
#from geopy.geocoders import Nominatim
import matplotlib.cm as cm
import matplotlib.colors as colors
from sklearn.cluster import KMeans
import folium
import math
import json
```

distance_from_coord

calculate the distance between two points given their geographical coordinates

```python
from math import sin, cos, sqrt, atan2, radians
import sys

def distance_from_coord(lat1a,lon1a,lat2a,lon2a):
    # approximate radius of earth in km
    R = 6373.0

    lat1 = radians(lat1a)
    lon1 = radians(lon1a)
    lat2 = radians(lat2a)
    lon2 = radians(lon2a)

    dlon = lon2 - lon1
    dlat = lat2 - lat1

    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))

    distance = R * c
```

```
    return distance
```

getVenuesAround

This function creates a grid of points centered on specific start point and (if parameters onlyPoints is false) use each point to perform a foursquare explore query to find the venues around the point. The reason for this function is the limit of 100 venues returned imposed by foursquare on free plans. We must perform a lot of small queries rather than just a big query to find the city venues.

return:

venues a dataframe with the venues fetched

points the list of the points in the grid used to perform the search of the venues

```
def getVenuesAround(center_point,grid_size,step_size,radius,onlyPoints=True):
    '''
        search around the centerpoint for venues
        the step_size is the distances in coordinates of the points used for the search
        the grid size is the number of points of the grid around the centerpoint used for th
    '''
    # this coefficient is set in order to limit the overlapping of adiacent searches
    overlap_coeff=0.94
    LIMIT=50000
    resultDF=pd.DataFrame()

    #let's calculate the lower left point of the grid centered around the center point
    start=[center_point[0] -((grid_size[0]/2) *step_size[0]*overlap_coeff),
            center_point[1] -((grid_size[1]/2)*step_size[1]*overlap_coeff)]
    step=step_size
    points=[]
    numsx=[start[0]]
    numsy=[start[1]]
    venues_list=[]
    found_ids=[]

    #fig,ax=plt.subplots(1,1)
    #fig.set_size_inches(15,15)


    # cycle on all the points of the grid
```

```
for i in range(0,grid_size[0]):
    for j in range(0,grid_size[1]):
        #print('point:',i,j)
        # we move the odd rows of the grid in order to create a triangular mesh instead
        # to minimize search overlap (the zone in wich the two circle of the foursquare
        # the grid overlaps)
        x=((j%2==0)*(step[0]/2))+start[0]+(i*overlap_coeff)*(step[0]*math.cos(math.pi/6)
        y=start[1]+((j*overlap_coeff)* step[1] * math.cos(math.pi/6))
        points.append((x,y))
        numsx.append(x)
        numsy.append(y)
        #ax.add_patch(plt.Circle((x,y),radius,color='red',alpha=0.2))


        # if onlyPoints is False then we perform the query to get the venues
        if not onlyPoints:
            try:
                url = 'https://api.foursquare.com/v2/venues/explore?client_id={}&client_
                #print('query:',url)
                response = requests.get(url)
                if response.status_code<300:
                    #print(results)
                    results = response.json()["response"]['groups'][0]['items']

                    for v in results:
                        venue_id=v['venue']['id']
                        if not (venue_id in found_ids):
                            found_ids.append(venue_id)
                            # return only relevant information for each nearby venue
                            venues_list.append(
                                (
                                x,
                                y,
                                radius,
                                v['venue']['id'],
                                v['venue']['name'],
                                v['venue']['location']['lat'],
                                v['venue']['location']['lng'],
                                v['venue']['categories'][0]['name']) )
            except:
                print(results)
                print("Oops!", sys.exc_info()[0], "occurred.")
                print('exception for point ({},{})'.format(x,y))

nearby_venues = pd.DataFrame(data=venues_list)
if not onlyPoints:
```

```
        nearby_venues.columns = [
                      'search_spot_lat',
                      'search_spot_lon',
                      'search_radius',
                      'id',
                      'Name',
                      'Latitude',
                      'Longitude',
                      'Category']

    return nearby_venues,points
```

cities definitions

This is the data that will be used to fetch data for the cities. Grid size and center location have been established looking at the city shape

```
cities=[
{
    'name':'New_York',
    'start_lat':40.749807,
    'start_lon':-73.987803,
    'grid_height':12,
    'grid_width':9
},
   {
    'name':'Boston',
    'start_lat':42.348347,
    'start_lon':-71.094065,
    'grid_height':12,
    'grid_width':9
   },
   {
    'name':'Paris',
    'start_lat':48.858655,
    'start_lon':2.348112,
    'grid_height':10,
    'grid_width':10
   },
   {
    'name':'London',
    'start_lat':51.507121,
    'start_lon':-0.128114,
    'grid_height':8,
    'grid_width':13
```

```
},
{
 'name':'Berlin',
 'start_lat':52.515731,
 'start_lon':13.388652,
 'grid_height':8,
 'grid_width':13
},
{
 'name':'Rome',
 'start_lat':41.887365,
 'start_lon':12.496733,
 'grid_height':10,
 'grid_width':10
},
{
 'name':'Tokyo',
 'start_lat':35.672032,
 'start_lon':139.745237,
 'grid_height':10,
 'grid_width':10
},
{
 'name':'Moscow',
 'start_lat':55.753960,
 'start_lon':37.623137,
 'grid_height':10,
 'grid_width':10
},
{
 'name':'Los_Angeles',
 'start_lat':34.058654,
 'start_lon':-118.240395,
 'grid_height':10,
 'grid_width':10
},
{
 'name':'Madrid',
 'start_lat':40.416717,
 'start_lon':-3.703298,
 'grid_height':10,
 'grid_width':10
},
{
 'name':'Philadelphia',
 'start_lat':39.961573,
```

```
 'start_lon':-75.153380,
 'grid_height':8,
 'grid_width':13
},
{
 'name':'Washington',
 'start_lat':38.906961,
 'start_lon':-77.036386,
 'grid_height':8,
 'grid_width':13
},
{
 'name':'Dallas',
 'start_lat':32.778024,
 'start_lon':-96.790979,
 'grid_height':8,
 'grid_width':13
}
 ,
{
 'name':'Phoenix',
 'start_lat':33.480416,
 'start_lon':-112.082774,
 'grid_height':8,
 'grid_width':13
}
 ,
{
 'name':'Barcelona',
 'start_lat':41.408774,
 'start_lon':2.171151,
 'grid_height':10,
 'grid_width':10
},
{
 'name':'Milan',
 'start_lat':45.466948,
 'start_lon':9.189333,
 'grid_height':8,
 'grid_width':13
},
 {
 'name':'Istanbul',
 'start_lat':41.025346,
 'start_lon':29.014207,
 'grid_height':8,
```

```
    'grid_width':13
   },
    {
    'name':'munich',
    'start_lat':48.139868,
    'start_lon':11.573441,
    'grid_height':8,
    'grid_width':13
   },
    {
    'name':'sao_paulo',
    'start_lat':-23.549398,
    'start_lon':-46.632438,
    'grid_height':8,
    'grid_width':13
   },
    {
    'name':'mexico_city',
    'start_lat':19.468088,
    'start_lon':-99.124208,
    'grid_height':8,
    'grid_width':13
   }

]
```

fetch__and__store__cities

this functions take the list of cities in input and for each entry in the list invoke the getVenuesAround function to get the venues in the cities and then save the retrieved FataFrame as a csv in the data directory

```
from IPython.core.display import display, HTML

def fetch_and_store_cities(cities):
    for city in cities:
        # the number of chunks of 500 meters that we want the edge of the base triangle of t
        MESH_DIST_IN_500_METERS=2

        #these are the angle increment in latitude and longitude that represent a distance o
        DELTA_ANGLE_LAT=-0.004496*MESH_DIST_IN_500_METERS
        DELTA_ANGLE_LON=-0.00642*MESH_DIST_IN_500_METERS

        #calculate the radius that optimize land coverage with minimum overlapping
        radius=(MESH_DIST_IN_500_METERS*500)/(1+math.sin(math.pi/6))

        # the size of the grid in columns and rows
        grid_size=(city['grid_height'],city['grid_width'])

        # the point on wich the grid must be centered
        start_lat=city['start_lat']
        start_lon=city['start_lon']

        venues,points=getVenuesAround((start_lat,start_lon),grid_size,(DELTA_ANGLE_LAT,DELTA

        venues.to_csv('./data/{}_venues.csv'.format(city['name'].lower()),index=False)
```

Attention uncomment the following lines just to fetch the data and then comment it back remember that foursquare free plan has a limit of 950 queries per day so you must execute it with a subset of cities each day to download all the data. The data has already been downloaded and is in the ./data directory of this project so there is not need to download it if not necessary

```
#fetch_and_store_cities(cities[0,4])  ## new york, boston, paris,london
#fetch_and_store_cities(cities[4:10])  ## berlin,rome,moscow,los angeles,tokyo,madrid
#fetch_and_store_cities(cities[10:15])  ## Philadelphia,Washington,Dallas,Phoenix,Barcelona
#fetch_and_store_cities(cities[15:20])  ## Milan,Istanbul,munich,sao_paulo,mexico_city
```

show__maps

print the map of each city with the grid points in blue and the found venues in green it expects to have a csv file for each city in the data directory. It will give and overview of the coverage of the analysis for each city. Unfortunately in order to minimize the number of foursquare queries only a porttion of the surface of each city has been analyzed (these are very big cities). So for each city only the zones near the center have been included

```
def show_maps(cities):
    for c in cities:

        # load city venues dataframe
        df=pd.read_csv('./data/{}_venues.csv'.format(c['name'].lower()))
        start_lat=c['start_lat']
        start_lon=c['start_lon']

        # get the search Points
        pointsDF=df[['search_spot_lat','search_spot_lon']].drop_duplicates()



        map_coverage = folium.Map(location=[start_lat, start_lon], zoom_start=13)


        for i,r in df.iterrows():
            p=(r['Latitude'],r['Longitude'])
            lat=p[0]
            lng=p[1]
            folium.CircleMarker(
                [lat, lng],
                radius=2,
                popup='',
                color='green',
                fill=True,
                fill_color='#ff0000',
                fill_opacity=0.7,
                parse_html=False).add_to(map_coverage)


        # cycle on the points of the grid and show each of them on the map
        for i,p in pointsDF.iterrows():
            lat=p['search_spot_lat']
            lng=p['search_spot_lon']
            folium.CircleMarker(
                [lat, lng],
                radius=4,
                popup='',
                color='blue',
                fill=True,
```

```
                fill_color='blue',
                fill_opacity=0.7,
                parse_html=False).add_to(map_coverage)
```

```
        display(HTML('<h1>'+c['name']+'</h1>'))
        display(map_coverage)
```

```
show_maps(cities)
```

New_York

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

Boston

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

Paris

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

London

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

Berlin

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

Rome

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

Tokyo

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

Moscow

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

Los_Angeles

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

Madrid

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

Philadelphia

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

Washington

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

Dallas

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

Phoenix

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

Barcelona

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

Milan

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

Istanbul

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

munich

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

sao_paulo

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

mexico_city

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

Perform Analysys

lets load all all the dataframes of the cities and put them in a single dataframe

```
result=pd.DataFrame()
for c in cities:
    #print((c['name']).lower())
    city_name=c['name'].lower()
    df=pd.read_csv('./data/{}_venues.csv'.format(city_name))
    df['city']=city_name
    result=result.append(df,ignore_index=True)
```

how many different categories are there in the whole venues set of all the cities?

```
print('total number of categories={}'.format(len(result['Category'].unique())))
```

```
total number of categories=717
```

Let's see the distribution of such categories

```
category_rank=result.groupby('Category').count().sort_values(by=['id'],ascending=False)
category_rank.head(3)
```

search_spot_lat

search_spot_lon

search_radius

id

Name

Latitude

Longitude

city

Category

Café

2483

2483

2483

2483

2483

2483

2483

2483

Italian Restaurant

2060

2060

2060

2060

2060

2060

2060

2060

Coffee Shop

2015

2015

2015

2015

2015

2015

2015

2015

let's list categories that have at least 100 occurrences in the list of venues

```
significant_categories=category_rank[category_rank['id']>100].index.values
significant_categories.shape
```

```
(127,)
```

Now let's remove rows that refer to venues that are not associated to significant categories

```
result_cleaned=result[result['Category'].isin(significant_categories)]
print('result shape={}, result_cleaned shape={}'.format(result.shape,result_cleaned.shape))
```

```
result shape=(60754, 9), result_cleaned shape=(49535, 9)
```

we have removed a few venues that had a peculiar category not usefull to cluster the venues

Now let's encode the categorical data **Category** with one hot encoder to transform its values in binary columns

```
hot_encoded=result.join(pd.get_dummies(result_cleaned['Category']))
hot_encoded.head(3)
```

search_spot_lat

search_spot_lon

search_radius

id

Name

Latitude

Longitude

Category

city

American Restaurant

…

Trail

Tram Station

Trattoria/Osteria

Turkish Restaurant

Vegetarian / Vegan Restaurant

Vietnamese Restaurant

Wine Bar

Wine Shop

Women's Store

Yoga Studio

0

40.796026

-73.949767

666.666667

5b855d2da0215b002c09d9fa

Teranga

40.796268

-73.949294

African Restaurant

new_york

NaN

. . .

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

1

40.796026

-73.949767

666.666667

4b9157c4f964a520b9b433e3

Duke Ellington Memorial by Robert Graham

40.796901

-73.949431

Outdoor Sculpture

new_york

NaN

. . .

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

2

40.796026

-73.949767

666.666667

4a9ad8d2f964a520213320e3

Conservatory Garden

40.793861

-73.952397

Garden

new_york

0.0

…

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

3 rows × 136 columns

Now we want to create a dataset that contains one row for each "zone" (or search spot) within a city and that for each category of venue contains the total number of instances of that category within the zone

let's drop columns that we do not use. Like the data specific of each venue including the Category that we have now hot encoded

```
hot_encoded.drop(columns=['id','Latitude','Longitude','Category'],inplace=True)
```

Now let's perform a group-by to obtain the counts for each category for each zone

```
zonesDf=hot_encoded.groupby(['city','search_spot_lat','search_spot_lon','search_radius']).su
zonesDf
```

American Restaurant

Art Gallery

Art Museum

Arts & Crafts Store

Asian Restaurant

Athletics & Sports

BBQ Joint

Bagel Shop

Bakery

Bank

. . .

Trail

Tram Station

Trattoria/Osteria

Turkish Restaurant

Vegetarian / Vegan Restaurant

Vietnamese Restaurant

Wine Bar

Wine Shop

Women's Store

Yoga Studio

city

search_spot_lat

search_spot_lon

search_radius

barcelona

41.380660

2.154853

666.666667

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

2.0

0.0

. . .

0.0

0.0

0.0

0.0

0.0

1.0

0.0

1.0

0.0

0.0

2.169493

666.666667

0.0

1.0

1.0

0.0

1.0

0.0

0.0

1.0

0.0

0.0

...

0.0

0.0

0.0

0.0

1.0

0.0

2.0

0.0

0.0

0.0

2.184133

666.666667

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

2.0

0.0

...

0.0

0.0

0.0

0.0

1.0

0.0

2.0

0.0

0.0

0.0

2.198773

666.666667

0.0

0.0

0.0

0.0

0.0

1.0

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

2.213413

666.666667

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

…

…

…

…

…

…

…

…

…

…

…

…

…

…

…

…

…

…

…

…

…

…

...

...

...

washington

38.940771

-77.047325

666.666667

0.0

0.0

1.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

...

4.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

-77.032685

666.666667

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

-77.018045

666.666667

0.0

0.0

0.0

0.0

0.0

0.0

1.0

0.0

0.0

0.0

...

0.0

0.0

0.0

0.0

0.0

1.0

0.0

0.0

0.0

1.0

-77.003405

666.666667

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

-76.988765

666.666667

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

1.0

…

0.0

0.0

0.0

0.0

0.0

1.0

0.0

0.0

0.0

0.0

2028 rows × 127 columns

how many zones have we got?

```
print('the number of zones we have is: {}'.format(zonesDf.shape[0]))
```

```
the number of zones we have is: 2028
```

We want to identify tipologies of zones. We want to group them based on the presence of venues within them. The zones have all the same size so we can compare zones from different cities even if one city is much bigger than another. Now the question is :

1. use the data as it is without normalization? the absolute count of each category of venue?
2. normalize the data before using it to clusterize the zones

if we choose to normalize the data how can we procede? One criteria of normalization would be to replace the absolute count with a percentage . If we put the pecentage of a category with respect to the total of the zone we will have each zone categorize by the type of venues rather than the number.

So for example two zones z1 and z2 that contains only grocery shops one with 1000 groceries and the other with 2 groceries will be represented by the same pecentage if they do not contain other kind of venues, both will have 1.0 in the Grocery shop column and 0 in the others.

But it is correct with respect to out analysis to consider the zones the same? The two zones are both part of big cities and have the same area. z1 is probably a zone dedicated to shop z2 is probably a residential zone or something else but their destination doesn't seems to be the same.

on the other hand some kind of normalization is necessary because of the characteristics of Kmeans algorithm we will use a standard scaler to scale all the category counts to a uniform range (0-1) the scaling will normalize the data using the vaules within each column

so returning to our case z1 and z2 will be represented with the currect scale

```
from sklearn.preprocessing import StandardScaler

scaler=StandardScaler()

scaler.fit(zonesDf)
zonedf_scaled=scaler.transform(zonesDf)
zonesDf
```

American Restaurant

Art Gallery

Art Museum

Arts & Crafts Store

Asian Restaurant

Athletics & Sports

BBQ Joint

Bagel Shop

Bakery

Bank

. . .

Trail

Tram Station

Trattoria/Osteria

Turkish Restaurant

Vegetarian / Vegan Restaurant

Vietnamese Restaurant

Wine Bar

Wine Shop

Women's Store

Yoga Studio

city

search_spot_lat

search_spot_lon

search_radius

barcelona

41.380660

2.154853

666.666667

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

2.0

0.0

...

0.0

0.0

0.0

0.0

0.0

1.0

0.0

1.0

0.0

0.0

2.169493

666.666667

0.0

1.0

1.0

0.0

1.0

0.0

0.0

1.0

0.0

0.0

...

0.0

0.0

0.0

0.0

1.0

0.0

2.0

0.0

0.0

0.0

2.184133

666.666667

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

2.0

0.0

. . .

0.0

0.0

0.0

0.0

1.0

0.0

2.0

0.0

0.0

0.0

2.198773

666.666667

0.0

0.0

0.0

0.0

0.0

1.0

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

2.213413

666.666667

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
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…

. . .

. . .

. . .

. . .

. . .

washington

38.940771

-77.047325

666.666667

0.0

0.0

1.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

. . .

4.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

-77.032685

666.666667

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

-77.018045

666.666667

0.0

0.0

0.0

0.0

0.0

0.0

1.0

0.0

0.0

0.0

...

0.0

0.0

0.0

0.0

0.0

1.0

0.0

0.0

0.0

1.0

-77.003405

666.666667

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

-76.988765

666.666667

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

1.0

...

0.0

0.0

0.0

0.0

0.0

1.0

0.0

0.0

0.0

0.0

2028 rows × 127 columns

```
zonedf_scaled.shape
```

```
(2028, 127)
```

Let's cluster the zones

```
from sklearn.cluster import KMeans
number_of_zones=10
kmeans = KMeans(n_clusters=number_of_zones, random_state=0).fit(zonedf_scaled)
zone_types=kmeans.labels_
zonesDf['zone_type']=kmeans.labels_
zonesDf.groupby(['zone_type']).count().iloc[:,0]
```

```
zone_type
0     195
1      45
2      87
3     139
4      66
5      76
6      67
7      22
8      70
9    1261
Name: American Restaurant, dtype: int64
```

The situation doesn't seems ideal there is a big cluster that contains more than half of the zones and then a lot of small clusters.

Let's proceed as we are not interested in the zones per se but in the composition of zone types of each city

Now let's pivot the data again creating a DataFrame that contains for each city a row and as columns for each zone type the count of the zones of that type in the city

```
cityDF=zonesDf.reset_index()[['city','zone_type']]
cityDF=cityDF.join(pd.get_dummies(cityDF['zone_type']))
cityDF=cityDF.drop(columns=['zone_type'])
cityDF=cityDF.groupby('city').sum()
cityDF
```

0

1

2

3

4

5

6

7

8

9

city

barcelona

0

0

0

4

0

1

0

0

30

63

berlin

0

0

1

33

0

2

4

0

0

61

boston

24

0

1

0

0

6

0

1

0

75

dallas

6

0

0

0

0

1

0

0

0

88

istanbul

0

0

0

0

0

3

41

13

0

47

london

25

0

0

15

0

6

6

0

1

51

los_angeles

8

3

0

1

0

0

0

0

0

87

madrid

0

0

0

1

0

2

0

0

39

57

mexico_city

0

42

0

0

0

2

0

0

0

60

milan

0

0

42

1

0

4

0

0

0

57

moscow

11

0

0

2

0

4

9

1

0

72

munich

0

0

4

20

0

1

1

0

0

76

new_york

48

0

0

1

0

12

0

7

0

40

paris

0

0

1

55

0

7

0

0

0

37

philadelphia

16

0

0

1

0

1

1

0

0

81

phoenix

3

0

0

0

0

2

0

0

0

94

rome

0

0

38

1

0

1

0

0

0

58

sao_paulo

26

0

0

2

1

17

4

0

0

54

tokyo

0

0

0

2

65

2

0

0

0

31

washington

28

0

0

0

0

2

1

0

0

72

Now let's cluster the cities using these profiles Let's start with only 3 clusters (since the total number of cities is quite small)

```
clusters_num=3
kmeans = KMeans(n_clusters=clusters_num, random_state=0).fit(cityDF)
cityDF['city_type']=kmeans.labels_
cityDF=cityDF.sort_values(by=['city_type'])
cityDF
```

0

1

2

3

4

5

6

7

8

9

city_type

city

tokyo

0

0

0

2

65

2

0

0

0

31

0

barcelona

0

0

0

4

0

1

0

0

30

63

1

rome

0

0

38

1

0

1

0

0

0

58

1

phoenix

3

0

0

0

0

2

0

0

0

94

1

philadelphia

16

0

0

1

0

1

1

0

0

81

1

munich

0

0

4

20

0

1

1

0

0

76

1

moscow

11

0

0

2

0

4

9

1

0

72

1

mexico_city

0

42

0

0

0

2

0

0

0

60

1

milan

0

0

42

1

0

4

0

0

0

57

1

los_angeles

8

3

0

1

0

0

0

0

0

87

1

dallas

6

0

0

0

0

1

0

0

0

88

1

boston

24

0

1

0

0

6

0

1

0

75

1

washington

28

0

0

0

0

2

1

0

0

72

1

london

25

0

0

15

0

6

6

0

1

51

2

istanbul

0

0

0

0

0

3

41

13

0

47

2

new_york

48

0

0

1

0

12

0

7

0

40

2

paris

0

0

1

55

0

7

0

0

0

37

2

berlin

0

0

1

33

0

2

4

0

0

61

2

sao_paulo

26

0

0

2

1

17

4

0

0

54

2

madrid

0

0

0

1

0

2

0

0

39

57

2

Tokyo was put in a cluster on its own, apparently because is the only city to have a significative amount of zones of type 4.

But what are these zones of type 4 like?

Let's analyze them

```
import warnings
warnings.filterwarnings('ignore')
def print_zone_type(zone_type):
    type4=zonesDf[zonesDf['zone_type']==zone_type]
    type4.drop(columns=['zone_type'],inplace=True)
    type4=type4.sum().sort_values(axis=0,ascending=False)
    display(HTML('<h3 style="color:green;">'+'zone type: {}'.format(zone_type)+'</h3>'))
    display(type4.head(10))
```

```
print_zone_type(4)
```

zone type: 4

```
Japanese Restaurant    252.0
Convenience Store      225.0
Ramen Restaurant       207.0
```

```
Café                       188.0
Sake Bar                   157.0
Coffee Shop                147.0
Chinese Restaurant         142.0
BBQ Joint                  137.0
Italian Restaurant         127.0
Soba Restaurant            122.0
dtype: float64
```

Seems to be zones with Japanes Restaurant ,Convenient Stores (Combini),Ramen Restaurants ,Sake Bar For sure Tokio is different from this respect to the other cities

This is the profile of a Japanese zone and is peculiar of that country So it seems ok to have put the town in a group on its own. Also because Tokio is the only east asia city in the group

The other group: the group zero that includes Paris, New York, London and Sao Paulo is characterized by an High number of zones of type 0 let's see what are those zones alike

```
print_zone_type(0)
```

zone type: 0

```
Coffee Shop                531.0
Italian Restaurant         318.0
Pizza Place                307.0
Bar                        300.0
Café                       282.0
Bakery                     257.0
Hotel                      256.0
Gym / Fitness Center       211.0
American Restaurant        198.0
Sandwich Place             173.0
dtype: float64
```

Let's increase the number of cluster to use to group the cityes to 5

```
clusters_num=5
kmeans = KMeans(n_clusters=clusters_num, random_state=0).fit(cityDF)
cityDF['city_type']=kmeans.labels_
cityDF=cityDF.sort_values(by=['city_type'])
cityDF
```

0

1

2

3

4

5

6

7

8

9

city_type

city

los_angeles

8

3

0

1

0

0

0

0

0

87

0

washington

28

0

0

0

0

2

1

0

0

72

0

boston

24

0

1

0

0

6

0

1

0

75

0

dallas

6

0

0

0

0

1

0

0

0

88

0

mexico_city

0

42

0

0

0

2

0

0

0

60

0

moscow

11

0

0

2

0

4

9

1

0

72

0

madrid

0

0

0

1

0

2

0

0

39

57

0

philadelphia

16

0

0

1

0

1

1

0

0

81

0

phoenix

3

0

0

0

0

2

0

0

0

94

0

barcelona

0

0

0

4

0

1

0

0

30

63

0

munich

0

0

4

20

0

1

1

0

0

76

0

tokyo

0

0

0

2

65

2

0

0

0

31

1

paris

0

0

1

55

0

7

0

0

0

37

2

berlin

0

0

1

33

0

2

4

0

0

61

2

sao_paulo

26

0

0

2

1

17

4

0

0

54

3

london

25

0

0

15

0

6

6

0

1

51

3

istanbul

0

0

0

0

0

3

41

13

0

47

3

new_york

48

0

0

1

0

12

0

7

0

40

3

milan

0

0

42

1

0

4

0

0

0

57

4

rome

0

0

38

1

0

1

0

0

0

58

4

Increasing the number of clusters we have another interesting cluster (the number 3) in wich the italian cities have been put.

This seems due to the zones of type 2

```
print_zone_type(2)
```

zone type: 2

```
Italian Restaurant     541.0
Pizza Place            282.0
Café                   257.0
Hotel                  231.0
Ice Cream Shop         174.0
Plaza                  139.0
Restaurant             114.0
Cocktail Bar           101.0
Japanese Restaurant     82.0
Bakery                  75.0
dtype: float64
```

Here again these are zones with a very high number of Italian restaurants and pizza places.

Kind of stereotypical but the reseults seems to lean toward the grouping of cities by nation/culture

Let's increase the number of clusters to confirm our guess

```
clusters_num=9
kmeans = KMeans(n_clusters=clusters_num, random_state=0).fit(cityDF)
cityDF['city_type']=kmeans.labels_
cityDF=cityDF.sort_values(by=['city_type'])
cityDF
```

0

1

2

3

4

5

6

7

8

9

city_type

city

tokyo

0

0

0

2

65

2

0

0

0

31

0

los_angeles

8

3

0

1

0

0

0

0

0

87

1

washington

28

0

0

0

0

2

1

0

0

72

1

boston

24

0

1

0

0

6

0

1

0

75

1

dallas

6

0

0

0

0

1

0

0

0

88

1

moscow

11

0

0

2

0

4

9

1

0

72

1

phoenix

3

0

0

0

0

2

0

0

0

94

1

philadelphia

16

0

0

1

0

1

1

0

0

81

1

barcelona

0

0

0

4

0

1

0

0

30

63

2

madrid

0

0

0

1

0

2

0

0

39

57

2

milan

0

0

42

1

0

4

0

0

0

57

3

rome

0

0

38

1

0

1

0

0

0

58

3

sao_paulo

26

0

0

2

1

17

4

0

0

54

4

london

25

0

0

15

0

6

6

0

1

51

4

new_york

48

0

0

1

0

12

0

7

0

40

4

munich

0

0

4

20

0

1

1

0

0

76

5

berlin

0

0

1

33

0

2

4

0

0

61

5

mexico_city

0

42

0

0

0

2

0

0

0

60

6

istanbul

0

0

0

0

0

3

41

13

0

47

7

paris

0

0

1

55

0

7

0

0

0

37

8

Here is the interpretation of the groups

0. contains Tokyo
1. contains all the USA cities (except for New York) plus Moscow
2. a mix with 3 cities that seems to be out of our iterpretation
3. The german cities
4. The italian cities
5. The Spanish cities
6. Mexico city

7. Paris
8. Istanbul

So our interpretation seems to be confirmed

The country/culture is the main cities aggregation factor. At least using the data that we have prepared/extracted

Conclusions

The conclusion of our analisys is that based on the categorization of the venues made by Foursquare the cities that are most similar are the ones of the same country/culture.

Appendice

Let's print all the zone types

```
for t in set(zone_types):
    print_zone_type(t)
```

zone type: 0

```
Coffee Shop             531.0
Italian Restaurant      318.0
Pizza Place             307.0
Bar                     300.0
Café                    282.0
Bakery                  257.0
Hotel                   256.0
Gym / Fitness Center    211.0
American Restaurant     198.0
Sandwich Place          173.0
dtype: float64
```

zone type: 1

```
Mexican Restaurant    234.0
Taco Place            177.0
Bakery                 47.0
Coffee Shop            43.0
Seafood Restaurant     43.0
Restaurant             41.0
Convenience Store      38.0
Ice Cream Shop         35.0
Pizza Place            35.0
Bar                    30.0
dtype: float64
```

zone type: 2

```
Italian Restaurant     541.0
Pizza Place            282.0
Café                   257.0
Hotel                  231.0
Ice Cream Shop         174.0
Plaza                  139.0
Restaurant             114.0
Cocktail Bar           101.0
Japanese Restaurant     82.0
Bakery                  75.0
dtype: float64
```

zone type: 3

```
French Restaurant        419.0
Café                     336.0
Hotel                    328.0
Bar                      298.0
Italian Restaurant       261.0
Coffee Shop              220.0
Bakery                   203.0
Pizza Place              147.0
Vietnamese Restaurant    131.0
Restaurant               131.0
dtype: float64
```

zone type: 4

```
Japanese Restaurant    252.0
Convenience Store      225.0
Ramen Restaurant       207.0
Café                   188.0
Sake Bar               157.0
Coffee Shop            147.0
Chinese Restaurant     142.0
BBQ Joint              137.0
Italian Restaurant     127.0
Soba Restaurant        122.0
dtype: float64
```

zone type: 5

```
Hotel                 201.0
Clothing Store        184.0
Coffee Shop           174.0
Italian Restaurant    138.0
Café                  137.0
Boutique               96.0
Cosmetics Shop         96.0
```

```
Plaza                   82.0
French Restaurant       80.0
Bakery                  78.0
dtype: float64
```

zone type: 6

```
Café                     345.0
Hotel                    175.0
Coffee Shop              161.0
Restaurant               123.0
Turkish Restaurant       114.0
Park                      88.0
Dessert Shop              82.0
Bakery                    82.0
Gym / Fitness Center      64.0
Bar                       62.0
dtype: float64
```

zone type: 7

```
Boat or Ferry          114.0
Café                    49.0
Art Gallery             35.0
Park                    26.0
Coffee Shop             24.0
Restaurant              19.0
Nightclub               17.0
Seafood Restaurant      14.0
Gym                     11.0
Bar                     10.0
dtype: float64
```

zone type: 8

```
Spanish Restaurant           337.0
Tapas Restaurant             213.0
Restaurant                   201.0
Hotel                        161.0
Bar                          140.0
Café                         116.0
Mediterranean Restaurant     115.0
Bakery                       115.0
Coffee Shop                   91.0
Plaza                         75.0
dtype: float64
```

zone type: 9

```
Café                   754.0
```

```
Coffee Shop           592.0
Italian Restaurant    553.0
Hotel                 538.0
Park                  510.0
Pizza Place           486.0
Restaurant            446.0
Bar                   409.0
Bakery                385.0
Mexican Restaurant    300.0
dtype: float64
```