Capstone Project - Find similar quarter in Sofia

Applied Data Science Capstone by IBM/Coursera

Table of contents

1.Introduction

1.1 Background

Sofia, as one of the most ancient cities in the Europe, is the largest city and the capital of Bulgaria. Sofia hosts some 1.23 million residents within a territory of 492 km2, a concentration of 17.5% of the country population within the 200th percentile of the country territory. Sofia today becomes a new frontier for real estate investors, For those who are looking for property in a lower price bracket, Sofia in Bulgaria is an excellent choice. My friend, Wealth Management Expert present the Sofia real estate market to their clients in Paris. He wants to show the client the Venue Data of Sofia. His client actually resides in the eleventh arrondissement of Paris. The eleventh arrondissement is a varied and engaging area. To the west lies the Place de la République, which is linked to the Place de la Bastille, in the east, by the sweeping, tree-lined Boulevard Richard-Lenoir, with its large markets and children's parks.

1.2 Business Problem

The business problem is that my friend will find the quater of Sofia which has the similar venue as the Paris 11e to show his client.

2. Data acquisition and cleaning

2.1. Data Source :

Import necessary Libraries

In [1]:

```
import requests
import pandas as pd
import numpy as np
from bs4 import BeautifulSoup as bs
```

In [2]:

```
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)

import json # library to handle JSON files

!conda install -c conda-forge geopy --yes # uncomment this line if you haven't completed the Fours
quare API lab
from geopy.geocoders import Nominatim # convert an address into latitude and longitude values

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

!conda install -c conda-forge folium=0.5.0 --yes # uncomment this line if you haven't completed th
e Foursquare API lab
import folium # map rendering library

print('Libraries imported.')
```

```
Collecting package metadata (current_repodata.json): done
Solving environment: done

# All requested packages already installed.

Collecting package metadata (current_repodata.json): done
Solving environment: done

# All requested packages already installed.

Libraries imported.
```

2.1.1.Data Table : Sofia Quarter and Postalcode

In [3]:

```
url='http://www.guide-bulgaria.com/SW/sofia-city/stolichna/sofia?t=postcodes'
html = requests.get(url).text
soup =bs(html,'lxml')
mylist= soup.find('div',{'class':'nine columns txt'}).find_all('li')
my_data=[]
mylist=[x.text.strip() for x in mylist]
my_data.append(mylist)
my_data=np.array(my_data).T.tolist()
df = pd.DataFrame(my_data,columns=["Data"])
new= df["Data"].str.split(' : ', n = 1, expand = True)
df["Quarter"]= new[0]
df["Postalcode"]= new[1]
df.drop(columns =["Data"], inplace = True)
df["Quarter"]= df["Quarter"].str.split(' ',n=1,expand=True).get(1)
df
```

Out[3]:

	Quarter	Postalcode
0	Bakston	1618
1	Banishora	1233
2	Beli brezi	1680
3	Benkovski	1278
4	Borovo	1680
5	Botunets	1870
6	Boyana	1616
7	Chelopechene	1853
8	Darvenitsa	1756
9	Dianabad	1172
10	Dragalevtsi	1415
11	Drujba 1	1592
12	Drujba 2	1582
13	Fakulteta	1373
14	Filipovtsi	1390

	Quarter	Postalcode
15	Geo Milev	1574
16	Gorna Banya	1614
17	Gorublyane	1138
18	Gotse Delchev	1404
19	Hadji Dimitar	1510
20	Hipodruma	1612
21	Hladilnika	1407
22	Hristo Botev	1517
23	Iliantsi	1271
24	Ivan Vazov	1408
25	Izgrev	1113
26	Iztok	1113
27	Knyajevo	1619
28	Krasna Polyana	1330
29	Krasno Selo	1618
30	Kremikovtsi	1849
31	Lagera	1612
32	Levski	1836
33	Lozenets	1164 / 1421
34	Lyulin 1	1360
35	Lyulin 10	1335
36	Lyulin 2	1343
37	Lyulin 3	1336
38	Lyulin 4	1359
39	Lyulin 5	1359
40	Lyulin 6	1336
41	Lyulin 7	1324
42	Lyulin 8	1336
43	Lyulin 9	1324
44	Malashevtsi	1225
45	Malinova Dolina	1797
46	Manastirski Livadi	1404
47	Mladost 1	1750 / 1784
48	Mladost 2	1799
49	Mladost 3	1712
50	Mladost 4	1715
51	Moderno predgradie	1360
52	Motopista	1404
53	Musagenitsa	1797
54	Nadejda	1220
55	Nadejda 3	1229
56	Nadejda 6	1231
57	Obelya	1387
58	Obelya 2	1326
59	Ovcha kupel	1618
60	Ovcha kupel 2	1632
61	Pavlovo	1618
62	Poduyane	1517
63	Poligona	1784

	Quarter	Postalcode
64	Reduta	1505
65	Republika	1301
66	Seslavtsi	1808
67	Simeonovo	1434
68	Slatina	1574
69	Stefan Karadja	1510
70	Strelbishte	1404
71	Studentski grad	1700
72	Suhata Reka	1505
73	Suhodol	1362
74	Sveta Troitsa	1309
75	Trebich	1298
76	Vrajdebna	1839
77	Yavorov	1111 / 1110
78	Zaharna fabrika	1345
79	Zapaden Park	1373
80	Zona B-18	1309
81	Zona B-19	1330
82	Zona B-5	1303

Use Nominatim to get geo coordinate for each quater (Test one quater works)

In [4]:

```
address = 'Lyulin 2, Sofia, Bulgaria'

geolocator = Nominatim(user_agent="foursquare_agent")
location = geolocator.geocode(address)

latitude = location.latitude
longitude = location.longitude
print(latitude, longitude)
```

```
42.7256471 23.2501109
```

2.1.2.Get geocode for each quarter

In [5]:

```
quartergeo=[]
latitude = []
longitude = []

for quarter in df["Quarter"]:
    try:
        inputAddress = '{},Sofia, Bulgaria'.format(quarter)
        geolocator = Nominatim(user_agent="foursquare_agent")
        location = geolocator.geocode(inputAddress,timeout=20)
        quartergeo.append(quarter)
        latitude.append(location.latitude)
        longitude.append(location.longitude)
    except:
        latitude.append('NA')
        longitude.append('NA')
        print('Error, skipping address...')

df_geocodes = pd.DataFrame({'Quartergeo':quartergeo, 'Latitude':latitude,'Longitude':longitude})
df_geocodes.head()
```

```
Error, skipping address...
Error, skipping address...
Error, skipping address...
```

```
Error, skipping address...
Error, skipping address...
Error, skipping address...
```

Out[5]:

	Quartergeo	Latitude	Longitude
0	Bakston	42.6655	23.27
1	Banishora	42.7104	23.3101
2	Beli brezi	42.6683	23.5991
3	Benkovski	42.7396	23.3461
4	Borovo	42.6694	23.2864

In [6]:

```
df_geocodes.head()
```

Out[6]:

	Quartergeo	Latitude	Longitude
0	Bakston	42.6655	23.27
1	Banishora	42.7104	23.3101
2	Beli brezi	42.6683	23.5991
3	Benkovski	42.7396	23.3461
4	Borovo	42.6694	23.2864

In [7]:

```
left=df
right=df_geocodes
result = pd.merge(left, right, how='left', left_on='Quarter', right_on='Quartergeo')
result.drop(['Quartergeo'],axis=1,inplace=True)
result.dropna(inplace=True)
result['Latitude']= pd.to_numeric(result['Latitude'], errors='coerce')
result['Longitude']= pd.to_numeric(result['Longitude'], errors='coerce')
result = result[result['Longitude'].notna()]
result.reset_index(drop=True, inplace=True)
result
```

Out[7]:

	Quarter	Postalcode	Latitude	Longitude
0	Bakston	1618	42.665486	23.269969
1	Banishora	1233	42.710366	23.310076
2	Beli brezi	1680	42.668338	23.599139
3	Benkovski	1278	42.739596	23.346105
4	Borovo	1680	42.669376	23.286375
5	Botunets	1870	42.739428	23.511576
6	Boyana	1616	42.646214	23.266749
7	Chelopechene	1853	42.732674	23.472598
8	Darvenitsa	1756	42.653300	23.361832
9	Dianabad	1172	42.662874	23.347948
10	Dragalevtsi	1415	42.630344	23.313248
11	Fakulteta	1373	42.694377	23.267152
12	Filipovtsi	1390	42.719928	23.221510
13	Geo Milev	1574	42.679996	23.362870
14	Gorna Banya	1614	42.678024	23.238154

	Quarter	Postalcode	Latitude	Longitude
15	Gorublyane	1138	42.628689	23.408469
16	Gotse Delchev	1404	42.664669	23.296839
17	Hadji Dimitar	1510	42.259366	23.828149
18	Hipodruma	1612	42.681536	23.295866
19	Hladilnika	1407	42.661604	23.315194
20	Hristo Botev	1517	42.688949	23.383933
21	Ivan Vazov	1408	42.677917	23.308493
22	Izgrev	1113	42.670481	23.351794
23	Iztok	1113	42.672496	23.357001
24	Krasna Polyana	1330	42.692156	23.282940
25	Krasno Selo	1618	42.673659	23.282501
26	Kremikovtsi	1849	42.784303	23.499576
27	Lagera	1612	42.684787	23.290161
28	Levski	1836	42.670104	23.607035
29	Lozenets	1164 / 1421	42.673454	23.325685
30	Lyulin 1	1360	42.728653	23.254175
31	Lyulin 10	1335	42.715139	23.272975
32	Lyulin 2	1343	42.725647	23.250111
33	Lyulin 3	1336	42.721172	23.246625
34	Lyulin 4	1359	42.717806	23.243309
35	Lyulin 5	1359	42.715169	23.238704
36	Lyulin 6	1336	42.711618	23.252646
37	Lyulin 7	1324	42.712356	23.263453
38	Lyulin 8	1336	42.721563	23.261108
39	Lyulin 9	1324	42.719010	23.266471
40	Malashevtsi	1225	42.714511	23.347627
41	Malinova Dolina	1797	42.634093	23.348000
42	Manastirski Livadi	1404	42.658158	23.279380
43	Mladost 1	1750 / 1784	42.652947	23.373399
44	Mladost 2	1799	42.644237	23.369960
45	Mladost 3	1712	42.646302	23.384784
46	Mladost 4	1715	42.629765	23.377944
47	Moderno predgradie	1360	42.724762	23.277341
48	Motopista	1404	42.668136	23.293650
49	Musagenitsa	1797	42.663221	23.363839
50	Nadejda	1220	42.751682	23.314530
51	Nadejda 3	1229	42.751682	23.314530
52	Nadejda 6	1231	42.751682	23.314530
53	Obelya	1387	42.742643	23.265865
54	Obelya 2	1326	42.746073	23.276539
55	Ovcha kupel	1618	42.677847	23.262216
56	Ovcha kupel 2	1632	42.686671	23.244130
57	Pavlovo	1618	42.665486	23.269969
58	Poduyane	1517	42.707065	23.368428
59	Poligona	1784	42.664017	23.379473
60	Reduta	1505	42.691933	23.360139
61	Republika	1301	42.733736	23.241573
62	Seslavtsi	1808	42.783171	23.517580
63	Simeonovo	1434	42.619836	23.338262
64	Slatina	1574	42.686790	23.398696

	Quarter	Postalcode	Latitude	Longitude
65	Stefan Karadja	[illegible]	[illegible]	[illegible]
66	Strelbishte	1404	42.672870	23.300022
67	Studentski grad	1700	42.650478	23.347341
68	Suhata Reka	1505	42.700852	23.367812
69	Suhodol	1362	42.696164	23.222577
70	Sveta Troitsa	1309	42.830606	22.639987
71	Trebich	1298	42.771602	23.314985
72	Yavorov	1111 / 1110	42.685008	23.348325
73	Zaharna fabrika	1345	42.716942	23.291906
74	Zapaden Park	1373	42.703555	23.279888
75	Zona B-18	1309	42.704275	23.302842
76	Zona B-19	1330	42.698007	23.299019

In [8]:

```
result.shape
```

Out[8]:

```
(77, 4)
```

In [10]:

```
address = 'Sofia, Bulgaria'
geolocator = Nominatim(user_agent="on_explorer")
location = geolocator.geocode(address)
latitude = location.latitude
longitude = location.longitude
print('The geograpical coordinate of Sofia are {}, {}.'.format(latitude, longitude))
```

```
The geograpical coordinate of Sofia are 42.6978634, 23.3221789.
```

2.1.3. Use Foursquare to get Sofia venue data

Define Foursquare Credentials and Version

In [11]:

```
CLIENT_ID = 'TKGWHMQGKH04YK4SERDJT0BAWNXEZAV5NGL5GACBCNDE4USU' # your Foursquare ID
CLIENT_SECRET = 'UEUZN2RRMSKQQOUM0AEYKH5VO022EPFDMMPAPGKO4ZLTG3ZH' # your Foursquare Secret
VERSION = '20180604'
LIMIT = 30
print('Your credentails:')
print('CLIENT_ID: ' + CLIENT_ID)
print('CLIENT_SECRET:' + CLIENT_SECRET)
```

```
Your credentails:
CLIENT_ID: TKGWHMQGKH04YK4SERDJT0BAWNXEZAV5NGL5GACBCNDE4USU
CLIENT_SECRET:UEUZN2RRMSKQQOUM0AEYKH5VO022EPFDMMPAPGKO4ZLTG3ZH
```

In [12]:

```
result.loc[0, 'Quarter']
```

Out[12]:

```
'Bakston'
```

In [13]:

```
quarter_latitude = result.loc[0, 'Latitude'] # latitude value
quarter_longitude = result.loc[0, 'Longitude'] # longitude value
```

```
quarter_longitude = result.loc[0, 'Longitude'] # longitude value

quarter_name = result.loc[0, 'Quarter'] # quarter name

print('Latitude and longitude values of {} are {}, {}.'.format(quarter_name,
                                                               quarter_latitude,
                                                               quarter_longitude))
```

```
Latitude and longitude values of Bakston are 42.6654864, 23.269969.
```

First, let's create the GET request URL.

In [14]:

```
radius=500
limit=100
url= 'http://api.foursquare.com/v2/venues/explore?client_id={}&client_secret={}&ll={},{}&v={}&radius={}&limit={}'.format(CLIENT_ID,
        CLIENT_SECRET,
        quarter_latitude,quarter_longitude,
        VERSION,
        radius,
        limit)
```

In [15]:

```
jresults = requests.get(url).json()
```

In [16]:

```
# function that extracts the category of the venue
def get_category_type(row):
    try:
        categories_list = row['categories']
    except:
        categories_list = row['venue.categories']

    if len(categories_list) == 0:
        return None
    else:
        return categories_list[0]['name']
```

In [17]:

```
venues = jresults['response']['groups'][0]['items']

nearby_venues = json_normalize(venues) # flatten JSON

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

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

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

nearby_venues.head()
```

Out[17]:

	name	categories	lat	lng
0	Аванти	Liquor Store	42.665564	23.272955
1	Млекарницата на Добрев 4	Cheese Shop	42.664005	23.266223
2	Ресторант Слънце	Restaurant	42.664546	23.266423
3	Сладкарница Сладки Идеи	Café	42.669091	23.269640

	name	categories	lat	lng
4	Dekora.Fashion	Clothing Store	42.663667	23.265996

In [18]:

```
def getNearbyVenues(names, latitudes, longitudes, radius=500):

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

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

        # make the GET request
        jresults = 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 jresults])

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

    return(nearby_venues)
```

In [19]:

```
Sofia_venues = getNearbyVenues(names=result['Quarter'],
                                    latitudes=result['Latitude'],
                                    longitudes=result['Longitude']
                                   )
```

```
Bakston
Banishora
Beli brezi
Benkovski
Borovo
Botunets
Boyana
Chelopechene
Darvenitsa
Dianabad
Dragalevtsi
Fakulteta
Filipovtsi
Geo Milev
Gorna Banya
Gorublyane
Gotse Delchev
Hadji Dimitar
Hipodruma
Hladilnika
Hristo Botev
```

```
Ivan Vazov
Izgrev
Iztok
Krasna Polyana
Krasno Selo
Kremikovtsi
Lagera
Levski
Lozenets
Lyulin 1
Lyulin 10
Lyulin 2
Lyulin 3
Lyulin 4
Lyulin 5
Lyulin 6
Lyulin 7
Lyulin 8
Lyulin 9
Malashevtsi
Malinova Dolina
Manastirski Livadi
Mladost 1
Mladost 2
Mladost 3
Mladost 4
Moderno predgradie
Motopista
Musagenitsa
Nadejda
Nadejda 3
Nadejda 6
Obelya
Obelya 2
Ovcha kupel
Ovcha kupel 2
Pavlovo
Poduyane
Poligona
Reduta
Republika
Seslavtsi
Simeonovo
Slatina
Stefan Karadja
Strelbishte
Studentski grad
Suhata Reka
Suhodol
Sveta Troitsa
Trebich
Yavorov
Zaharna fabrika
Zapaden Park
Zona B-18
Zona B-19
```

In [20]:

```
print(Sofia_venues.shape)
Sofia_venues.head()
```

```
(1430, 7)
```

Out[20]:

	Quarter	Quarter Latitude	Quarter Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
0	Bakston	42.665486	23.269969	Аванти	42.665564	23.272955	Liquor Store
1	Bakston	42.665486	23.269969	Млекарницата на Добрев 4	42.664005	23.266223	Cheese Shop
2	Bakston	42.665486	23.269969	Ресторант Слънце	42.664546	23.266423	Restaurant
3	Bakston	42.665486	23.269969	Сладкарница Сладки Идеи	42.669091	23.269640	Café

	Quarter	Quarter Latitude	Quarter Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
4	Bakston	42.665486	23.269969	Dekora.Fashion	42.663667	23.265996	Clothing Store

2.1.4. Use Foursquare to get Paris 11e venue data

In [21]:

```
address = '75011 paris'

geolocator = Nominatim(user_agent="foursquare_agent")
location = geolocator.geocode(address)

p_latitude = location.latitude
p_longitude = location.longitude
print(p_latitude, p_longitude)
```

```
48.8566969 2.3514616
```

In [22]:

```
radius=500
limit=100
url= 'http://api.foursquare.com/v2/venues/explore?client_id={}&client_secret={}&ll={},{}&v={}&radius={}&limit={}'.format(CLIENT_ID,
        CLIENT_SECRET,
        p_latitude,p_longitude,
        VERSION,
        radius,
        limit)
jresults = requests.get(url).json()
```

In [23]:

```
# function that extracts the category of the venue
def get_category_type(row):
    try:
        categories_list = row['categories']
    except:
        categories_list = row['venue.categories']

    if len(categories_list) == 0:
        return None
    else:
        return categories_list[0]['name']
```

In [24]:

```
venues = jresults['response']['groups'][0]['items']

paris11_venues = json_normalize(venues) # flatten JSON

# filter columns
filtered_columns = ['venue.name', 'venue.categories', 'venue.location.lat', 'venue.location.lng']
paris11_venues =paris11_venues.loc[:, filtered_columns]

# filter the category for each row
paris11_venues['venue.categories'] = paris11_venues.apply(get_category_type, axis=1)

# clean columns
paris11_venues.columns = [col.split(".")[-1] for col in paris11_venues.columns]

paris11_venues.shape
```

Out[24]:

```
(100, 4)
```

In [25]:

```
paris11_venues.head()
```

Out[25]:

	name	categories	lat	lng
0	Place de l'Hôtel de Ville – Esplanade de la Li...	Plaza	48.856925	2.351412
1	Parc Rives de Seine	Park	48.855510	2.351419
2	L'Alsacien	Alsatian Restaurant	48.858275	2.350381
3	Berges de Seine – Rive droite	Pedestrian Plaza	48.855131	2.352289
4	Square de la Tour Saint-Jacques	Park	48.857882	2.348757

2.2 Data Cleaning

In [26]:

```
paris11_venues['Quarter']='Paris11'
paris11_venues['Quarter Latitude']=p_latitude
paris11_venues['Quarter Longitude']=p_longitude
paris11_venues.head()
```

Out[26]:

	name	categories	lat	lng	Quarter	Quarter Latitude	Quarter Longitude
0	Place de l'Hôtel de Ville – Esplanade de la Li...	Plaza	48.856925	2.351412	Paris11	48.856697	2.351462
1	Parc Rives de Seine	Park	48.855510	2.351419	Paris11	48.856697	2.351462
2	L'Alsacien	Alsatian Restaurant	48.858275	2.350381	Paris11	48.856697	2.351462
3	Berges de Seine – Rive droite	Pedestrian Plaza	48.855131	2.352289	Paris11	48.856697	2.351462
4	Square de la Tour Saint-Jacques	Park	48.857882	2.348757	Paris11	48.856697	2.351462

In summary of this data 100 venues returned by Foursquare. The Paris 11 arrondissment is popular quarter. If I want to find a similar quarter in Sofia, the quarter should have also **a lot of venues**.

In [33]:

```
paris11_venues.columns = ['Venue','Venue Category','Venue Latitude','Venue Longitude','Quarter','Quarter Latitude','Quarter Longitude']
```

In [34]:

```
paris11_venues=paris11_venues[['Quarter','Quarter Latitude','Quarter Longitude','Venue','Venue Latitude','Venue Longitude','Venue Category']]
```

In [110]:

```
paris11_venues.shape
```

Out[110]:

```
(100, 7)
```

So only the quarter with **more than 30 venues** will be used. I named these quarters as popular quarters in sofia.

In [126]:

```
sofia_count= Sofia_venues.groupby('Quarter').count()
sofia_count= sofia_count[sofia_count['Quarter Latitude']>30]
sofia_count.index
```

Out[126]:

```
Index(['Bakston', 'Borovo', 'Geo Milev', 'Hladilnika', 'Ivan Vazov', 'Izgrev',
       'Iztok', 'Lozenets', 'Mladost 1', 'Mladost 4', 'Motopista',
       'Musagenitsa', 'Pavlovo', 'Poligona', 'Reduta', 'Stefan Karadja',
       'Strelbishte', 'Studentski grad', 'Yavorov', 'Zona B-18'],
      dtype='object', name='Quarter')
```

In [133]:

```
popquarter=sofia_count.index.tolist()
popquarter
```

Out[133]:

```
['Bakston',
 'Borovo',
 'Geo Milev',
 'Hladilnika',
 'Ivan Vazov',
 'Izgrev',
 'Iztok',
 'Lozenets',
 'Mladost 1',
 'Mladost 4',
 'Motopista',
 'Musagenitsa',
 'Pavlovo',
 'Poligona',
 'Reduta',
 'Stefan Karadja',
 'Strelbishte',
 'Studentski grad',
 'Yavorov',
 'Zona B-18']
```

In [150]:

```
to_append=['Paris11','75011',p_latitude,p_longitude]
a_series = pd.Series(to_append, index = result.columns)
totalresult=result[result['Quarter'].isin(popquarter)].append(a_series, ignore_index=True)
```

I merge the venues datatable in popular sofia quarters and the venue datatable in paris 11e.

In [134]:

```
Sofia_venues_pop=Sofia_venues[Sofia_venues['Quarter'].isin(popquarter)]
```

I remove one row with missing value.

In [135]:

```
total_venues=pd.concat([Sofia_venues_pop,paris11_venues])
total_venues.reset_index(drop=True, inplace=True)
total_venues.tail()
```

Out[135]:

	Quarter	Quarter Latitude	Quarter Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
924	Paris11	49	2	Les Philosophes	49	2	French Restaurant
925	Paris11	49	2	Les Thermes de Lutèce	49	2	Salon / Barbershop
926	Paris11	49	2	Bières Cultes	49	2	Liquor Store
927	Paris11	49	2	Lafayette Anticipations	49	2	Art Gallery
928	Paris11	49	2	Place Georges Pompidou	49	2	Plaza

The data needs to be represented in a numerically comparable way. To solve this, I created a new column for each unique value in

the “quarter” column. This is automatically done by the “get_dummies” function of Pandas.

In [136]:

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

# add quarter column back to dataframe
total_onehot['Quarter'] = total_venues['Quarter']

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

total_onehot.head()
```

Out[136]:

	Quarter	Adult Boutique	Alsatian Restaurant	Art Gallery	Art Museum	Arts & Crafts Store	Athletics & Sports	Auvergne Restaurant	BBQ Joint	Bakery	Bank	Bar	Basketball Court	Basketball Stadium	C
0	Bakston	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	Bakston	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	Bakston	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	Bakston	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	Bakston	0	0	0	0	0	0	0	0	0	0	0	0	0	

In [137]:

```
total_grouped = total_onehot.groupby('Quarter').mean().reset_index()
total_grouped
```

Out[137]:

	Quarter	Adult Boutique	Alsatian Restaurant	Art Gallery	Art Museum	Arts & Crafts Store	Athletics & Sports	Auvergne Restaurant	BBQ Joint	Bakery	Bank	Bar	Basketball Court	Basketb Stadiu
0	Bakston	0	0	0	0	0	0	0	0	0	0	0	0	
1	Borovo	0	0	0	0	0	0	0	0	0	0	0	0	
2	Geo Milev	0	0	0	0	0	0	0	0	0	0	0	0	
3	Hladilnika	0	0	0	0	0	0	0	0	0	0	0	0	
4	Ivan Vazov	0	0	0	0	0	0	0	0	0	0	0	0	
5	Izgrev	0	0	0	0	0	0	0	0	0	0	0	0	
6	Iztok	0	0	0	0	0	0	0	0	0	0	0	0	
7	Lozenets	0	0	0	0	0	0	0	0	0	0	0	0	
8	Mladost 1	0	0	0	0	0	0	0	0	0	0	0	0	
9	Mladost 4	0	0	0	0	0	0	0	0	0	0	0	0	
10	Motopista	0	0	0	0	0	0	0	0	0	0	0	0	
11	Musagenitsa	0	0	0	0	0	0	0	0	0	0	0	0	
12	Paris11	0	0	0	0	0	0	0	0	0	0	0	0	
13	Pavlovo	0	0	0	0	0	0	0	0	0	0	0	0	
14	Poligona	0	0	0	0	0	0	0	0	0	0	0	0	
15	Reduta	0	0	0	0	0	0	0	0	0	0	0	0	
16	Stefan Karadja	0	0	0	0	0	0	0	0	0	0	0	0	
17	Strelbishte	0	0	0	0	0	0	0	0	0	0	0	0	
18	Studentski grad	0	0	0	0	0	0	0	0	0	0	0	0	
19	Yavorov	0	0	0	0	0	0	0	0	0	0	0	0	

20	Zona B-18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

3. Methodology

I decided to use Kmeans algorithm.
Kmeans algorithm is an iterative algorithm that tries to partition the dataset into Kpre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. It tries to make the inter-cluster data points as similar as possible while also keeping the clusters as different as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous the data points are within the same cluster.

The way kmeans algorithm works is as follows:

- Specify number of clusters K.
- Initialize centroids by first shuffling the dataset and then randomly selecting K data points for the centroids without replacement.
- Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn't changing.
- Compute the sum of the squared distance between data points and all centroids.
- Assign each data point to the closest cluster (centroid).
- Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

I will find Paris 11e will belong to which cluster. The remaining quarters in this cluster will be similar quarters in Sofia of Paris 11e.

4. Analysis

Let's apply the data with the KMeans method.

In [138]:

```
# set number of clusters
kclusters = 5

total_grouped_clustering = total_grouped.drop('Quarter', 1)

# run k-means clustering
kmeans = KMeans(n_clusters=kclusters, random_state=0).fit(total_grouped_clustering)

# check cluster labels generated for each row in the dataframe
kmeans.labels_[0:10]
```

Out[138]:

```
array([3, 0, 2, 4, 2, 1, 1, 2, 2, 3], dtype=int32)
```

In [139]:

```
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 [152]:

```
num_top_venues = 10
indicators = ['st', 'nd', 'rd']

# create columns according to number of top venues
columns = ['Quarter']
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
quarters_venues_sorted = pd.DataFrame(columns=columns)
quarters_venues_sorted['Quarter'] = total_grouped['Quarter']
```

```
for ind in np.arange(total_grouped.shape[0]):
    quarters_venues_sorted.iloc[ind, 1:] = return_most_common_venues(total_grouped.iloc[ind, :], nu
m_top_venues)

quarters_venues_sorted.head()
```

Out[152]:

	Quarter	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7th Most Common Venue	8th Most Common Venue	9th Most Common Venue	10th Most Common Venue
0	Bakston	Plaza	Eastern European Restaurant	Restaurant	Café	Farmers Market	Gym	Italian Restaurant	Cheese Shop	Chinese Restaurant	Snack Place
1	Borovo	Restaurant	BBQ Joint	Bakery	Pizza Place	Pharmacy	Café	Liquor Store	Italian Restaurant	Gym / Fitness Center	Yoga Studio
2	Geo Milev	Gym / Fitness Center	Eastern European Restaurant	Hotel	Dessert Shop	Sushi Restaurant	Bulgarian Restaurant	Bakery	Pizza Place	French Restaurant	Electronics Store
3	Hladilnika	Gym	Clothing Store	Dance Studio	Cosmetics Shop	Men's Store	Restaurant	Boxing Gym	Frozen Yogurt Shop	Nightclub	Coffee Shop
4	Ivan Vazov	Pizza Place	Italian Restaurant	Pharmacy	Bakery	Eastern European Restaurant	Yoga Studio	Nightclub	Café	Cheese Shop	Restaurant

In [153]:

```
# add clustering labels
quarters_venues_sorted.insert(0, 'Cluster Labels', kmeans.labels_)

totalresult.rename(columns={"Quarter":"Quarter"},inplace=True)
totalresult_merged = totalresult
# merge toronto_grouped with toronto_data to add latitude/longitude for each neighborhood
totalresult_merged = totalresult_merged.join(quarters_venues_sorted.set_index('Quarter'), on='Quar
ter')

totalresult_merged.head(10) # check the last columns!
```

Out[153]:

	Quarter	Postalcode	Latitude	Longitude	Cluster Labels	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7th M Comn Ve
0	Bakston	1618	43	23	3	Plaza	Eastern European Restaurant	Restaurant	Café	Farmers Market	Gym	Ita Restau
1	Borovo	1680	43	23	0	Restaurant	BBQ Joint	Bakery	Pizza Place	Pharmacy	Café	Lic St
2	Geo Milev	1574	43	23	2	Gym / Fitness Center	Eastern European Restaurant	Hotel	Dessert Shop	Sushi Restaurant	Bulgarian Restaurant	Bak
3	Hladilnika	1407	43	23	4	Gym	Clothing Store	Dance Studio	Cosmetics Shop	Men's Store	Restaurant	Box G
4	Ivan Vazov	1408	43	23	2	Pizza Place	Italian Restaurant	Pharmacy	Bakery	Eastern European Restaurant	Yoga Studio	Nightc
5	Izgrev	1113	43	23	1	Restaurant	Eastern European Restaurant	Health & Beauty Service	Italian Restaurant	Chinese Restaurant	Park	Nail Sa
6	Iztok	1113	43	23	1	Bulgarian Restaurant	Eastern European Restaurant	Gym	Italian Restaurant	Restaurant	Health & Beauty Service	Fast F Restau
7	Lozenets	1164 / 1421	43	23	2	Grocery Store	Café	Restaurant	Hotel	Bulgarian Restaurant	Pizza Place	Su Restau
8	Mladost 1	1750 / 1784	43	23	2	Restaurant	Pizza Place	Supermarket	Gastropub	Café	Bus Stop	I Sta

	Quarter	Postalcode	Latitude	Longitude	Cluster Labels	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7th M
9	Mladost 4	1715	43	23	2	Restaurant	Restaurant	Pizza Place	Café	Supermarket	Bistro	Ro

In [154]:

```
totalresult_merged.dropna(inplace=True)
totalresult_merged.reset_index(drop=True, inplace=True)
```

The Paris 11 belongs to Cluster Label N°2.

In [155]:

```
totalresult_merged.loc[totalresult_merged['Quarter'] == 'Paris11']
```

Out[155]:

	Quarter	Postalcode	Latitude	Longitude	Cluster Labels	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7th Most Common Venue	8th M Comm Ven
20	Paris11	75011	49	2	2	French Restaurant	Plaza	Ice Cream Shop	Art Gallery	Hotel	Clothing Store	Cocktail Bar	Ro

There are **6** quarters in Sofia in the cluster N°2.

In [156]:

```
totalresult_merged.loc[totalresult_merged['Cluster Labels'] == 2]
```

Out[156]:

	Quarter	Postalcode	Latitude	Longitude	Cluster Labels	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7t Co
2	Geo Milev	1574	43	23	2	Gym / Fitness Center	Eastern European Restaurant	Hotel	Dessert Shop	Sushi Restaurant	Bulgarian Restaurant	
4	Ivan Vazov	1408	43	23	2	Pizza Place	Italian Restaurant	Pharmacy	Bakery	Eastern European Restaurant	Yoga Studio	Ni
7	Lozenets	1164 / 1421	43	23	2	Grocery Store	Café	Restaurant	Hotel	Bulgarian Restaurant	Pizza Place	Res
8	Mladost 1	1750 / 1784	43	23	2	Restaurant	Pizza Place	Supermarket	Gastropub	Café	Bus Stop	
11	Musagenitsa	1797	43	23	2	Café	Restaurant	Pharmacy	Athletics & Sports	Electronics Store	Eastern European Restaurant	BB
17	Studentski grad	1700	43	23	2	Pizza Place	Supermarket	Nightclub	Bar	Cocktail Bar	Electronics Store	
20	Paris11	75011	49	2	2	French Restaurant	Plaza	Ice Cream Shop	Art Gallery	Hotel	Clothing Store	C

In [157]:

```
result_merged= totalresult_merged[totalresult_merged.Quarter != 'Paris11']
```

In [158]:

```
# create map
map_clusters = folium.Map(location=[latitude, longitude], zoom_start=11)

#set color scheme for the clusters
x = np.arange(kclusters)
ys = [i + x + (i*x)**2 for i in range(kclusters)]
```

```
colors_array = cm.rainbow(np.linspace(0, 1, len(ys)))
rainbow = [colors.rgb2hex(i) for i in colors_array]

# add markers to the map
markers_colors = []
for lat, lon, poi, cluster in zip(result_merged['Latitude'], result_merged['Longitude'], result_mer
ged['Quarter'], result_merged['Cluster Labels']):
    label = folium.Popup(str(poi) + ' Cluster ' + str(cluster), parse_html=True)
    folium.CircleMarker(
        [lat, lon],
        radius=5,
        popup=label,
        color=rainbow[cluster-1],
        fill=True,
        fill_color=rainbow[cluster-1],
        fill_opacity=0.7).add_to(map_clusters)

map_clusters
```

Out[158]:

All the quarters similar to Paris 11e are the **blue points** in the Sofia map.

5. Results and Discussion

In this study, I used the venues information to compare one quarter in Paris with quarters in Sofia. Result of this analysis, there are 6 quarters which are the most similar to Paris 11e. My friend can show his clients firstly the appartments in these 6 quarters.

6. Conclusion

This analysis can use to find potential deal and present the location which is interested in the investor.