GUIDE TO EXPLORE HANOI

IBM Data Science Capstone Project

Hoang Van Huy Thach

March 12, 2020

1. Introduction

For any travelers, it's very important to do research about the destination beforehand. So we will need to find out the area on Hanoi that suitable for travelers.

The main criterias is we need to find are ***many restaurant***, **acceptable number of hotels** and finaly, some ***entertainment venues***

We will use data science to get all the needed info to help traveler decide the place to stay when they go to Hanoi.

2. Data

a. Data source

Based on the criterias, we will need data such as:

- List of urban districts of Hanoi from **Wikipedia**: Link - I already scap the info and put into **hanoi_district.csv** file
- List of wards for each district from **Wikipedia**: Link - The link to each list is included in ***hanoi_district.csv*** file
- **Google Chrome** is used to analyse the content of Wikipedia page.
- Map data, district boundary, coordinates from **OpenStreetMap** (OSM)
- Location data from **FourSquare API**.
- Online tool from https://tyrasd.github.io/osmtogeojson/ to convert OSM data into geojson

b. Data cleaning

We already have the list of main district in Hanoi in a csv file. After import it into a dataframe and select list of district to explore, we have the following structure:

admin_level	boundary	name	name:en	name:ko	type	wikidata	id	geometry
6	administrative	Quận Hoàn Kiếm	Hoan Kiem District	호안끼엠군	boundary	Q1134529	relation/9421131	POLYGON ((105.84413 21.03508, 105.84422 21.034...
6	administrative	Quận Cầu Giấy	Cau Giay District	꺼우저이군	boundary	None	relation/9421132	POLYGON ((105.80135 21.03023, 105.80577 21.035...
6	administrative	Quận Hai Bà Trưng	Hai Ba Trung District	하이바쭝군	boundary	Q1134523	relation/9421134	POLYGON ((105.86780 21.01989, 105.86199 21.018...
6	administrative	Quận Đống Đa	Dong Da District	동다군	boundary	Q1048634	relation/9421135	POLYGON ((105.84270 20.99727, 105.84135 21.000...
6	administrative	Quận Ba Đình	Ba Dinh District	바딘군	boundary	None	relation/9421136	POLYGON ((105.81467 21.04300, 105.81438 21.043...

Figure 1 – District info dataframe

Using the district name in "*d_en_name*" column, we can query OSM for the boundary data of each district. The data from OSM formated as JSON (or XML depend on your choice), we can use the online tool at https://tyrasd.github.io/osmtogeojson/ to convert it to *geojson* format, suitable for use with ***folium*** to draw map.

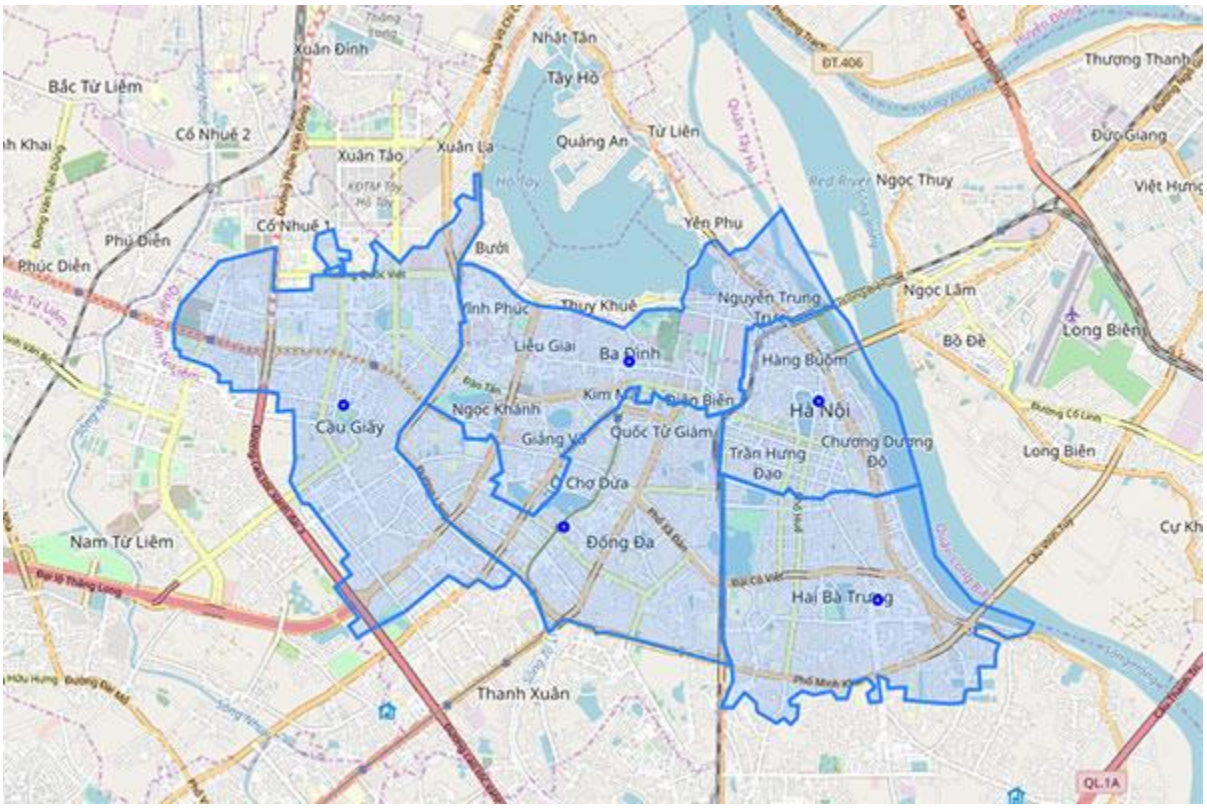

Figure 2 – Map of Hanoi main district with district boundary

Based on the info in our dataframe (fig. 1), we can get the list of wards belong to each district. But we a little problem:

d_code	d_en_name	d_vi_name	d_lat	d_lon	d_url	w_name	w_url	w_lat	w_lon
1	Ba Dinh District	Ba Đình	21.035252	105.828854	https://vi.wikipedia.org/wiki/Th%E1%BB%83_lo%E...	Điện Biên, Ba Đình	https://vi.wikipedia.org/wiki/%C4%90i%E1%BB%87...	NaN	NaN
1	Ba Dinh District	Ba Đình	21.035252	105.828854	https://vi.wikipedia.org/wiki/Th%E1%BB%83_lo%E...	Đội Cấn, Ba Đình	https://vi.wikipedia.org/wiki/%C4%90%E1%BB%99i...	NaN	NaN
1	Ba Dinh District	Ba Đình	21.035252	105.828854	https://vi.wikipedia.org/wiki/Th%E1%BB%83_lo%E...	Ngọc Hà, Ba Đình	https://vi.wikipedia.org/wiki/Ng%E1%BB%8Dc_H%C...	NaN	NaN
1	Ba Dinh District	Ba Đình	21.035252	105.828854	https://vi.wikipedia.org/wiki/Th%E1%BB%83_lo%E...	Nguyễn Trung Trực (phường)	https://vi.wikipedia.org/wiki/Nguy%E1%BB%85n_T...	NaN	NaN
1	Ba Dinh District	Ba Đình	21.035252	105.828854	https://vi.wikipedia.org/wiki/Th%E1%BB%83_lo%E...	Quán Thánh (phường)	https://vi.wikipedia.org/wiki/Qu%C3%A1n_Th%C3%...	NaN	NaN

Figure 3 – List of wards with problematic name

We have a problem with ward name ("*w_name*" column): sometimes they including the district name, or the suffix "*(phường)*". We will clean up the name so that it only have the ward name.

The next step is to get coordinate of each ward. Our first choice is using OSM. But since OSM's info came from users contribution, some wards coordinates info is missing. From observing the wiki page of each ward, we know that there's coordinate info so we can take those missing info from *Wikipedia*.

Figure 4 – Ward's coordinate as seen on Wikipedia page

After scrape the info from Wikipedia, there're still 7 wards with missing coordinates, we will drop them from our dataframe.

With the new dataframe contain list of ward along with their coordinates, we can put them on map.

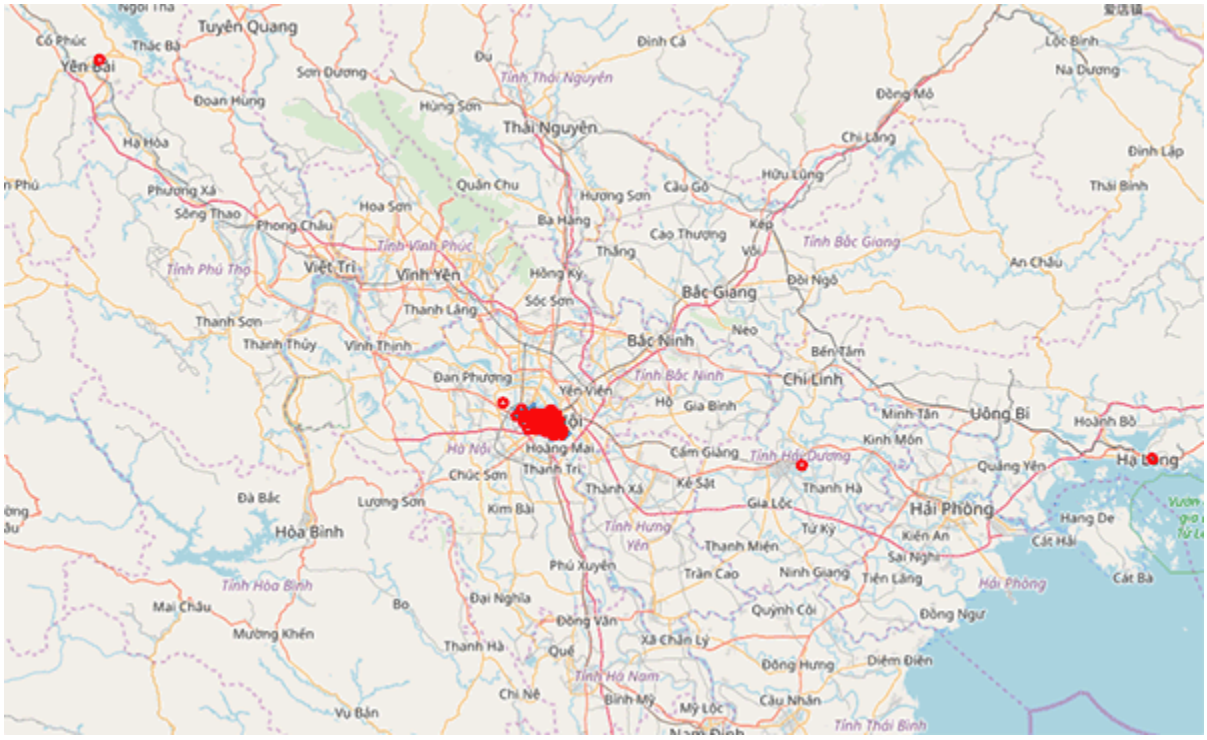

Figure 5 – Map of wards in Hanoi

Look at the map, we can see some ward is way off from their district. So we have to remove those wards with wrong coordinate from our list, too. We do that by using *geopy.distance* – this lib can calculate distance between 2 point on map. The following plot will let us know about the outlier.

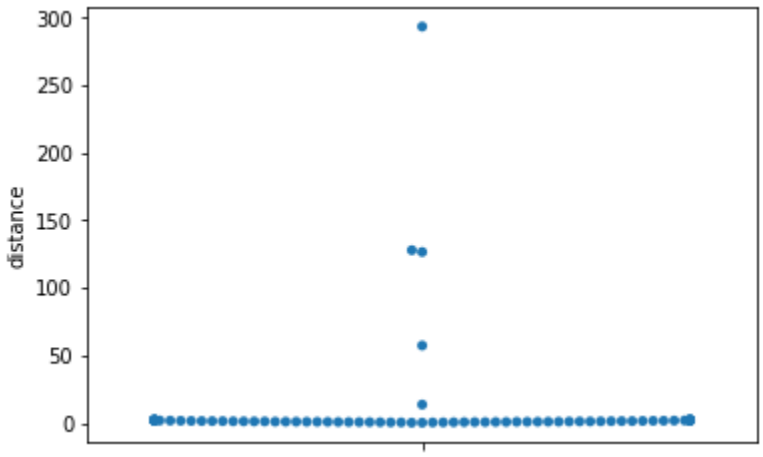

Figure 6 – Distance of wards from their district center

Look at the plot, we decide to drop all ward that have the distance to their district center more than 5km. Here is the plot after we drop those wards.

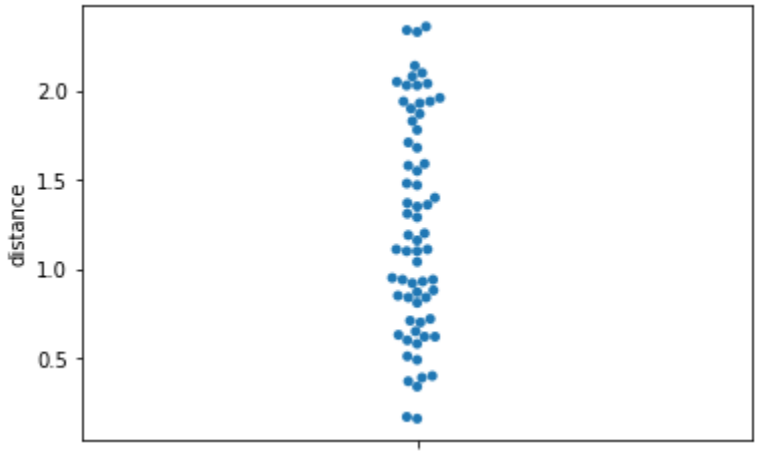

Figure 7 – Distance of wards from their district center after cleanup.

And here is the map with those "outlier" wards removed

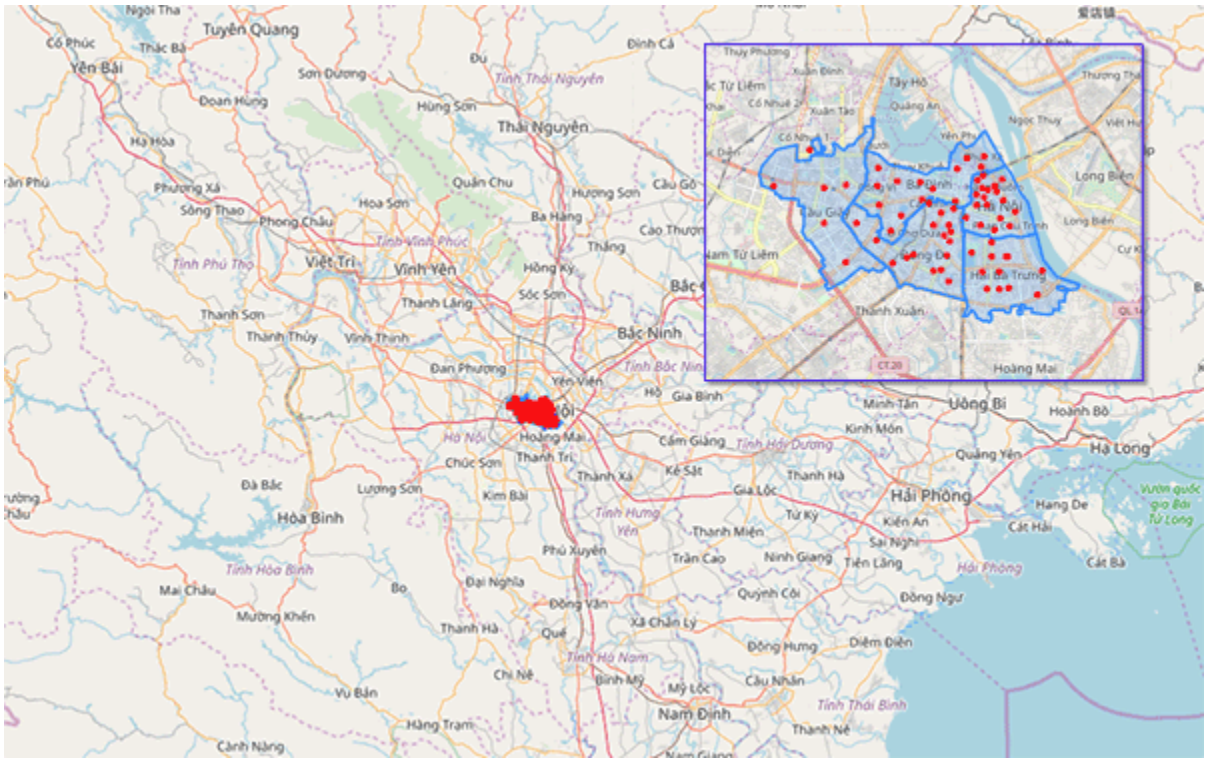

Figure 8 – Map after cleaned

Now we can proceed to explore the nearby venues using FourSquare API

c. Exploratory Data Analysis

We plan to search for venue in the radius of 800m from each ward. To preview the area we will search, we can draw the circle with radius 800m on our map.

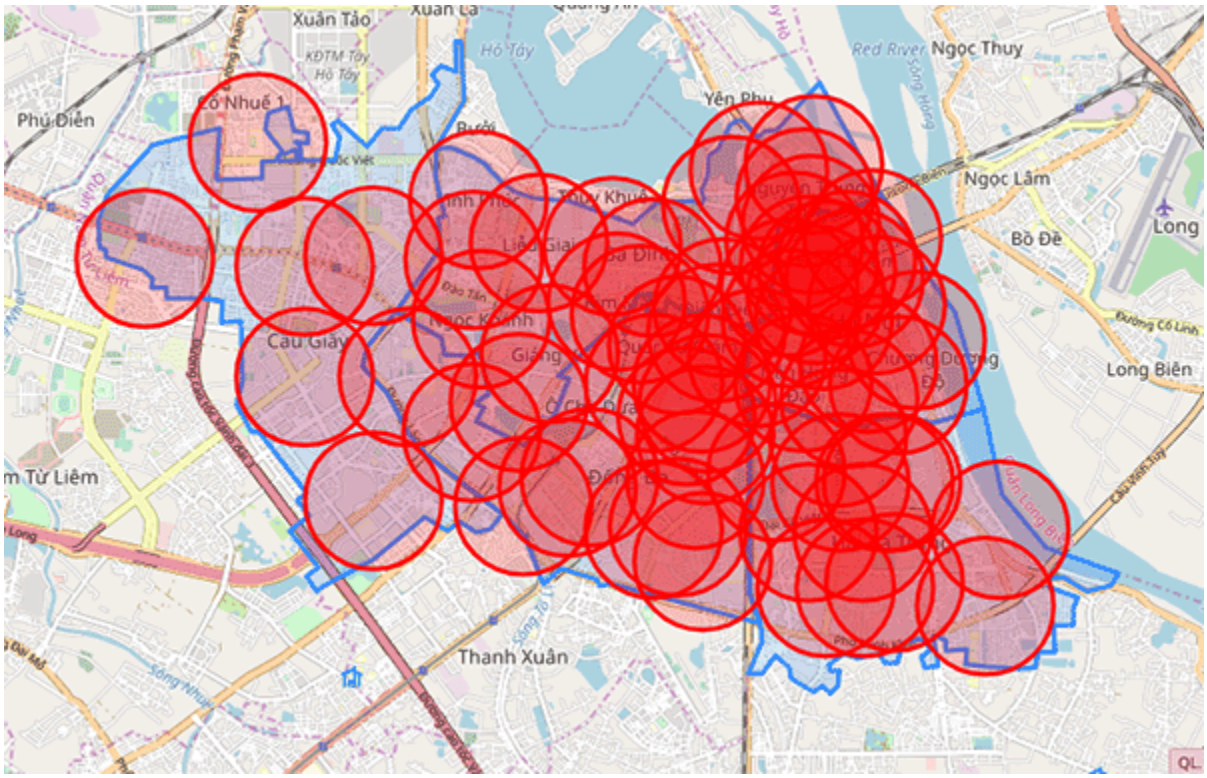

Figure 9 – Preview search area on map

As you can see from the map, the search area at the central district is heavily overlapping because we have 49 Point-Of-Interest (POI) in 3 districts: Ba Dinh, Hoan Kiem, Dong Da. The total POI we have is **67**.

To reduce the number of POI, we should group those wards together. And we choose to use DBSCAN as it's one of the most effective algorithm to group location on map.

Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu in 1996.[1] It is a density-based clustering non-parametric algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away).

From Wikipedia

To use DBSCAN, we need some parameter

- ε – the maximum distance between node in a cluster. In our case, we wiil set it as 500m. To use with DBSCAN, we will need to convert it to radians.
- min_samples – the minimum cluster size. We choose the value 1 so any ward in the outer district will be belong to a cluster
- The dataframe with coordinates data (latitude and longitude)
- We use *haversine* metric and *ball tree space-partitionning data structure*.

The result of DBSCAN is:

```
Cluster labels: [ 0  1  2  3  4  5  6  7  8  8  9 10 11 12 13  8 14  8  8 15  8  8  8  8
  8  8  8  8 15  8 15 16 17 18 19 20 21 22  4 23 24 25 26 27 28 25 29  1
 30 24 29 24 25 24  1 31 32 33 33 34 15 35 31 31 36 37 38]
39 clusters
```

Figure 10 – DBSCAN result

The POI decrease to 39 from 67 in the original data. The search area is not so dense like before.

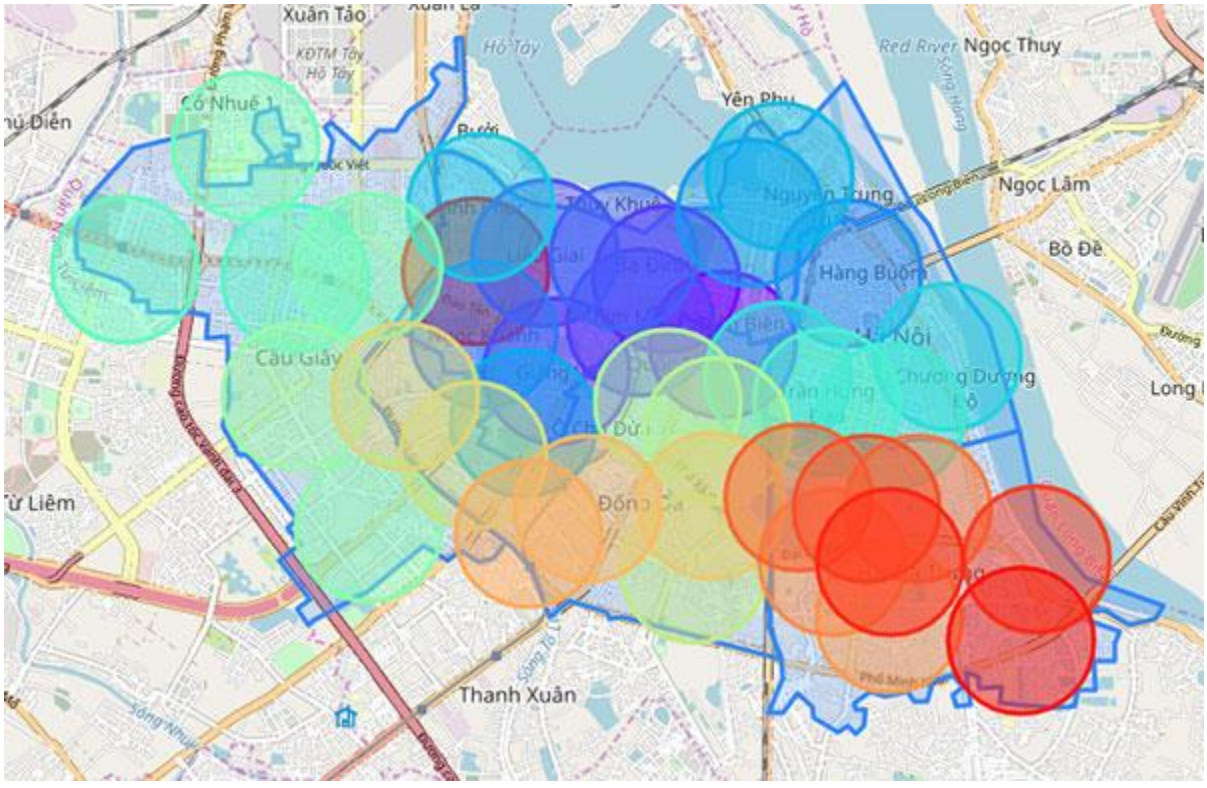

Figure 11- Search area after group wards together

Now we can begin get data from FourSquare using its API. We will request API data based on venues category. There're 3 categories that we interested in: *food, hotel* and *entertainment.* Let's take a look at the map show the distribution of each type of venue.

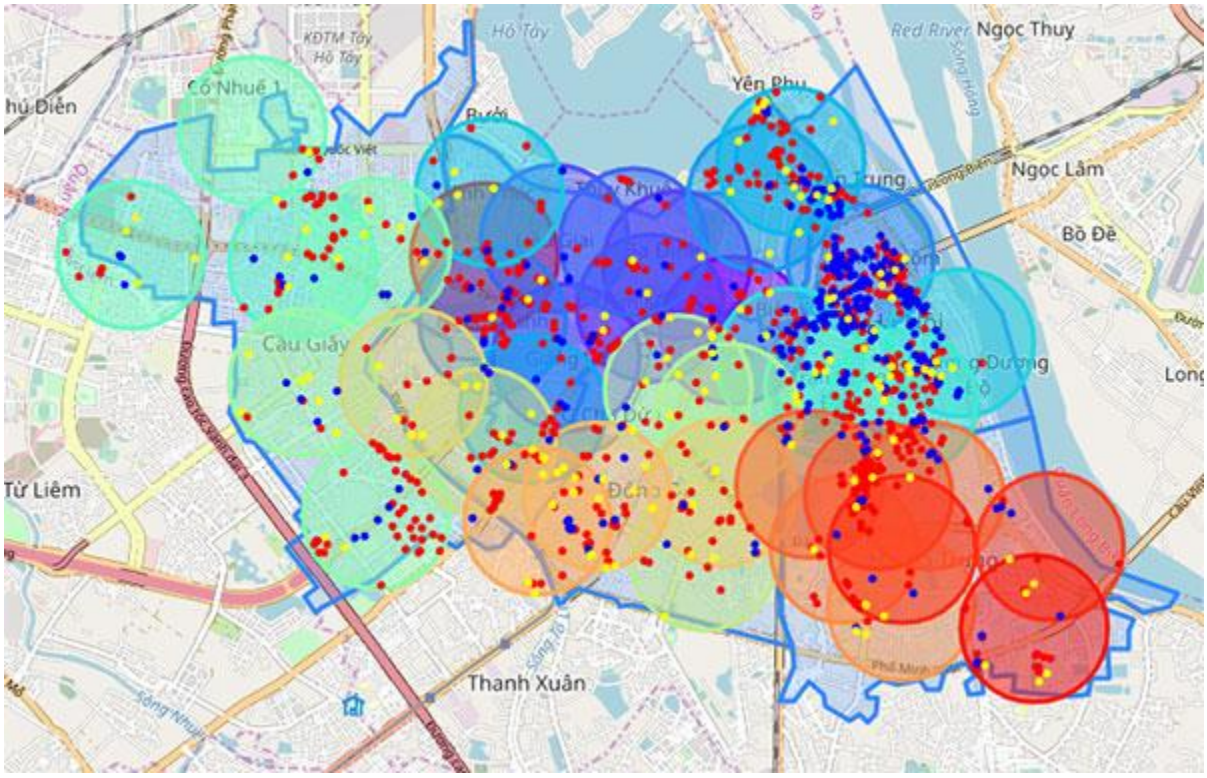

Figure 12 – Venues distribution map

```
venues_list.groupby('c_label')['v_
```

```
count     39.000000
mean      52.948718
std       45.311175
min        6.000000
25%       23.500000
50%       37.000000
75%       66.000000
max      210.000000
Name: v_name, dtype: float64
```

Figure 13 – Overview of the venues list (group by cluster)

Based on fig. 13, we will drop the cluster with less than 30 venues. We can then calculate the distribution of each type of venue in each cluster. And the result is this plot

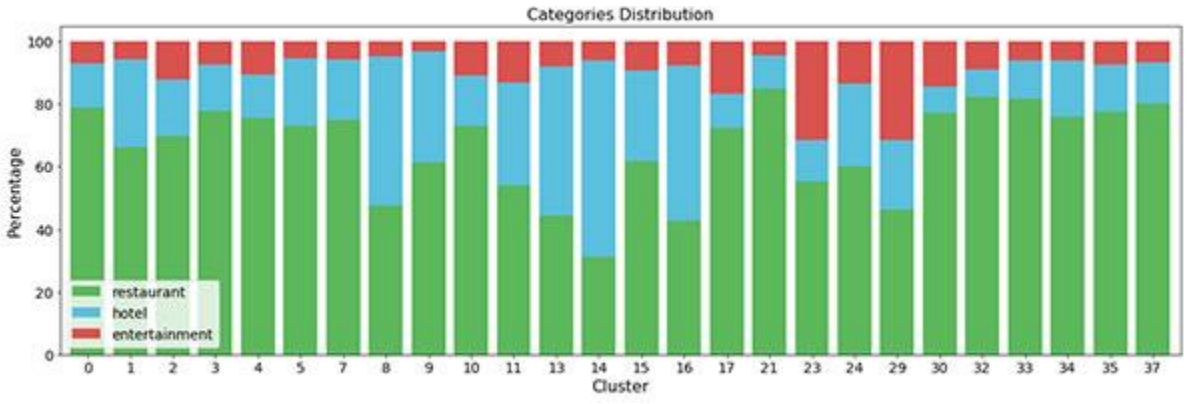

Figure 14 – Categories Distribution

Base on our criterias, we choose clusters **[2, 4, 10, 11, 17, 23, 29, 30]** as our candidates. We do further analysis by calculate what kind of sub-category (exclude all sub-categories related to **hotel**) is most common in each cluster. It turned out that cluster **23** and **29** may not fit our taste, so we drop them out of our selection list.

cluster_label	1st most Common Venue	2nd most Common Venue	3rd most Common Venue	4th most Common Venue	5th most Common Venue	6th most Common Venue
2	Café	BBQ Joint	History Museum	Vietnamese Restaurant	Japanese Restaurant	Noodle House
4	Vietnamese Restaurant	Café	Noodle House	Bakery	Japanese Restaurant	Chinese Restaurant
10	Vietnamese Restaurant	Café	Seafood Restaurant	Asian Restaurant	BBQ Joint	Ramen Restaurant
11	Vietnamese Restaurant	Café	Noodle House	Hotpot Restaurant	Movie Theater	Art Gallery
17	Vietnamese Restaurant	Café	Japanese Restaurant	Fast Food Restaurant	Korean Restaurant	Seafood Restaurant
23	Café	Music Venue	Vietnamese Restaurant	Noodle House	Art Gallery	Fast Food Restaurant
29	Café	Art Gallery	Vietnamese Restaurant	Music Venue	Fast Food Restaurant	Asian Restaurant
30	Café	Vietnamese Restaurant	Noodle House	Dance Studio	Bakery	Hotpot Restaurant

Figure 15 – Most common venues in each cluster

And the remaining cluster that may be suitable for us is **[2, 4, 10, 11, 17, 30].** We can display them on a map.

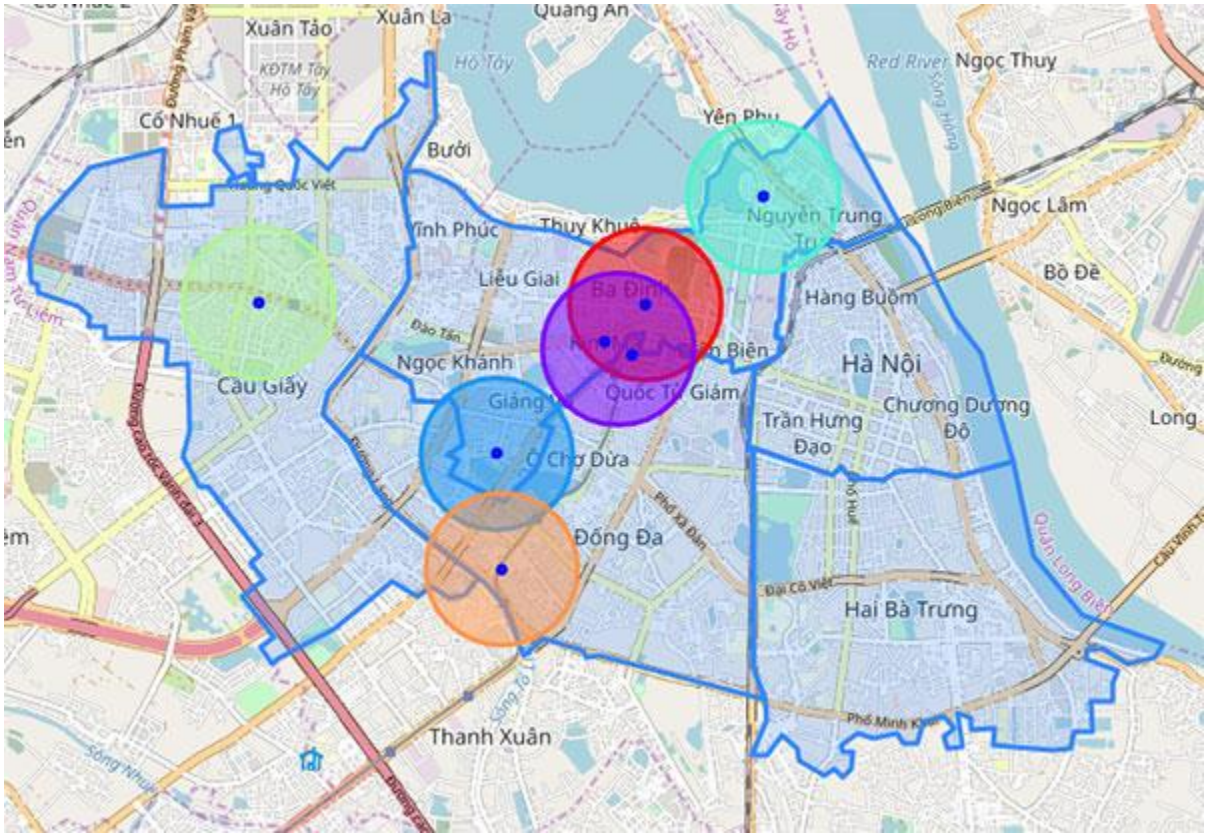

Figure 16 – Our final candidates

3. Results and discussion

The final result can be extract from the cluster as follow

```
Recommended ward to stay during your time in Hanoi is

Ba Dinh District
        Doi Can
        Kim Ma
        Thanh Cong
        Truc Bach

Cau Giay District
        Dich Vong

Dong Da District
        Cat Linh
        Thinh Quang
```

Figure 17 – The Final Result

We can say that although there many place to choose to stay in Hanoi, based on our preference, we can narrow down the seletion to just 7 wards in 3 districts.

Our analysis can run effectively by using the power of DBSCAN to reduce the number of node, so that we can reduce the API call to FourSquare. Also, the help of visualization tool such as **folium** and **matplotlib** make it easy to see the who picture.

4. Conclusion

Our analysis can run effectively by using the power of DBSCAN to reduce the number of node, so that we can reduce the API call to FourSquare. Also, the help of visualization tool such as **folium** and **matplotlib** make it easy to see the who picture.