Wrangle OpenStreetMap Data with MongoDB

1. Introduction

This project uses the data from OpenStreetMap (https://www.openstreetmap.org) to audit the data quality, standardize the data, import the data into a MongoDB database, and finally look into some statistics of the data.

Since I now live in Chicago, I downloaded the Chicago map data for the project. The dataset is an XML file downloaded from Map Zen (https://mapzen.com/data/metro-extracts/#chicago-illinois), and the file is around 2GB uncompressed.

2. Process the Dataset

By looking into small samples of the dataset, I found 3 major issues in the data as follows:

- Many street names are abbreviated and cause inconsistency;
- Inconsistent state names and strange value of "IN" found in the dataset;
- Strange postal code started from "4".

In addition to these issues, I have also plotted all the locations available in latitude and longitude, to check if the data makes sense. Another minor issue is on the naming of the key in the JSON file. Naming is updated to avoid conflictions.

The code to generate samples is attached as "step0_generate_sample.py".

2.1. Abbreviated Street Names

It shows that many addresses under tag "addr:street" have abbreviated street names. For example:

- N Pfingsten Rd
- Varsity Dr
- Vail Ct

The code to identify this issue is attached as "step1_check_street.py".

To solve the issue, I updated all abbreviated street names to their full names. After the update, the streets names in the example above now respectively become as follows:

- N Pfingsten Road
- Varsity Drive
- Vail Court

The code to solve this issue is included in file "step2_process_data.py".

2.2. Strange and Inconsistent State Name

It shows that some state names under tag "addr:state" are not consistent. For example:

- Illinois
- Il
- IL - Illinois

Besides, given Chicago is a city located in Illinois, I surprisingly found two records in another state: a building named “Sycamore Hall” in Lake County, Indiana, and a building named “Comfort Inn & Suits” in Porter, Indiana. However, by looking into the map, we could see that they both are located in northwest Indiana and very close to Chicago. These data points make sense if we define “Chicago” as the “Chicago Metropolitan Area”. Therefore, I keep it as it is in the dataset.

The code to identify these issues is attached as “step1_check_state.py”.

To solve the issue of inconsistent state names, I updated all state names to the simple abbreviation, such as “IL” and “IN”. The code is included in file “step2_process_data.py” as well.

2.3. Strange Postal Code

Similar to the last issue of state name, I noticed some postal codes not as expected. It starts with “4”, which is different from the usual Chicago postal codes that start with “6”. For example:

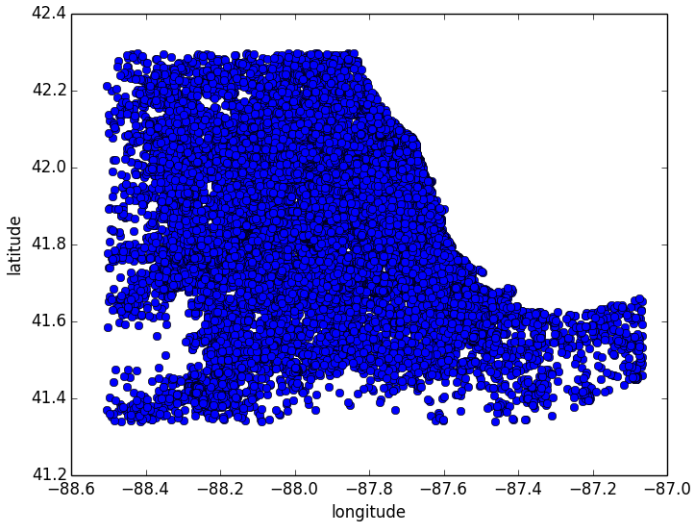
- 46304
- 46385
- 46307

Actually, this issue could be explained as the last one: the place is located in Chicago Metropolitan Area. No action is taken on the data point.

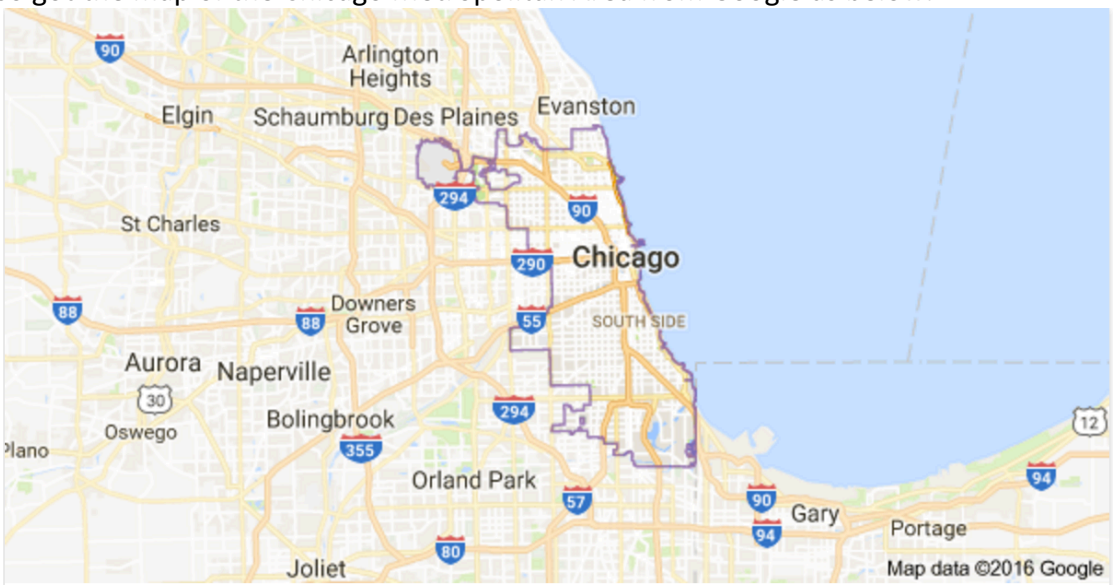
The code to identify this issue is attached as “step1_check_postcode.py”.

2.4. Plot of Record Positions

I am somehow interested in the position of all these records shown on a plot. Therefore, I made a plot showing all records’ latitude and longitude. The plot is as below:



I also got the map of the Chicago Metropolitan Area from Google as below:



They look very close, and I saw no outliers in the plot I made. The code to make the plot is attached as “step1_check_position.py”.

2.5. Special “Type” Attribute

As the project proposed, I would only process records with tag value of “node” or “way”, and save the tag value as “type” in the final JSON file. However, I noticed many records have a child attribute name called “type”. For example:

- A record with id “369053117” has child attribute name of “type” and the value of that attribute is “public”
- A record with id “418527731” has child attribute name of “type” and the value of that attribute is “broad_leaved”

Therefore, I decided to rename the tag value as “tag_type”, to avoid this confliction. The code to identify this issue is attached as “step1_check_type.py”.

2.6. Loading to MongoDB

After implementing all the standardization mentioned above, the dataset we need is now save as a JSON file named “chicago_illinois.json”. The code is attached as “step2_process_data.py”.

I loaded the data to a MongoDB database for further study. The command in terminal to upload the JSON file is as below:

```
mongoimport --db local --collection chicago --file chicago_illinois.json
```

3. Explore the Database

In this section, I will show a series of MongoDB commands and the results telling us more about the dataset. In each segment, I will provide the command (highlighted in grey) that I used in MongoDB shell and the result it returned.

3.1. File Size

- chicago_illinois.osm: size of 1.98 GB
- chicago_illinois.json: size of 2.22 GB

3.2. Number of Documents

```
> db.chicago.find().count()
```

9482941

3.3. Number of Unique Users

```
> db.chicago.distinct("created.user").length
```

2189

3.4. Number of Nodes

```
> db.chicago.find({"tag_type":"node"}).count()
```

8318381

3.5. Number of Ways

```
> db.chicago.find({"tag_type":"way"}).count()
```

1164560

3.6. Number of Cafes

```
> db.chicago.find({"amenity":"cafe"}).count()
```

426

3.7. Number of Shops

```
> db.chicago.find({"shop":{$exists:true}}).count()
```

3324

4. Additional Exploration of the Database

Besides some basic statistics mentioned above, I am interested in the statistics of all amenities in Chicago. I live in Chicago and use all kinds of amenities every day, including restaurants, cafes, libraries, and theaters. It is fun to get to know the city better through the database. Similar style as Section 3 will take place to address the findings I found.

4.1. Top Amenities

```
>
db.chicago.aggregate([{"$match":{"amenity":{$exists:true}}},{"$group":{"_id":"$amenity","amenity_count":{"$sum":1}}},{"$sort":{"amenity_count":-1}},{"$limit":10}])
{ "_id" : "parking", "amenity_count" : 12372 }
{ "_id" : "place_of_worship", "amenity_count" : 4293 }
{ "_id" : "school", "amenity_count" : 3429 }
{ "_id" : "restaurant", "amenity_count" : 1873 }
{ "_id" : "fast_food", "amenity_count" : 1287 }
{ "_id" : "fuel", "amenity_count" : 756 }
{ "_id" : "bank", "amenity_count" : 555 }
{ "_id" : "grave_yard", "amenity_count" : 455 }
{ "_id" : "cafe", "amenity_count" : 426 }
{ "_id" : "shelter", "amenity_count" : 364 }
```

Clearly parking is the most common facilities in Chicago, way more than other amenities. Other than that, there are also many places of worship and schools in Chicago.

Restaurants are not as many as I thought. One of the reason may be that “restaurant” here are very strictly defined. For example, fast foods and cafes are not considered as restaurants.

4.2. Top Locations of Restaurants by Postal Code

```
>
db.chicago.aggregate([{"$match":{"amenity":"restaurant"}},{"$match":{"address.postcode":{$exists:true}}},{"$group":{"_id":"$address.postcode","restaurant_count":{"$sum":1}}},{"$sort":{"restaurant_count":-1}},{"$limit":5}])
{ "_id" : "60201", "restaurant_count" : 78 }
{ "_id" : "60148", "restaurant_count" : 25 }
{ "_id" : "60173", "restaurant_count" : 24 }
{ "_id" : "60614", "restaurant_count" : 19 }
{ "_id" : "60630", "restaurant_count" : 18 }
```

If we check the map by ourselves, we can see that postal code 60201 is near Northwestern University, where many students study and live. This makes sense to me. However, postal codes like 60148 and 60173 are all suburb areas of Chicago. This looks strange to me so I dig more into it by running the following command:

```
> db.chicago.aggregate([{"$match":{"amenity":"restaurant"}},{"$group":{"_id":"$address.postcode","restaurant_count":{"$sum":1}}},{"$sort":{"restaurant_count":-1}},{"$limit":5}])
{ "_id" : null, "restaurant_count" : 1320 }
{ "_id" : "60201", "restaurant_count" : 78 }
{ "_id" : "60148", "restaurant_count" : 25 }
{ "_id" : "60173", "restaurant_count" : 24 }
{ "_id" : "60614", "restaurant_count" : 19 }
```

Now things are more clear to us. 1320 out of 1873 restaurants do not have a postal code with them! Therefore, we can hardly find the top restaurant locations simply by looking at postal codes.

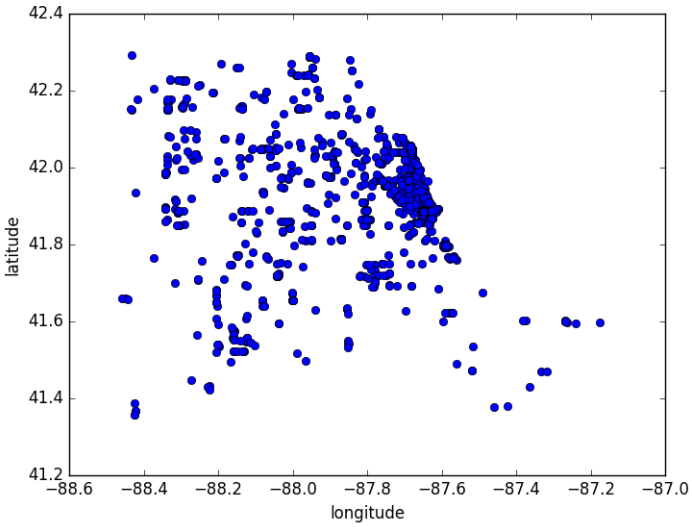
4.3. Top Locations of Restaurants by Position

Another way to check the locations of top restaurants is by their positions saved in latitudes and longitudes. I first ran the following command:

```
> db.chicago.find({"pos":{$exists:true},"amenity":"restaurant"}).count()
1358
```

It shows that most restaurants have their positions saved. However, when I tried to get some statistics of the longitudes through MongoDB, I found it very difficult to do so since I could not distinguish the longitude and latitude from the position.

I managed to do so through “pymongo” in Python. The plot below shows how the restaurants are distributed in Chicago:



Comparing to the real map, the plot above makes perfect sense, since most restaurants are located near downtown Chicago.

The code to make the plot is saved as “step3_top_restaurant.py”.

5. Conclusions

After all, I think this is a really good project to get to practice my knowledge in MongoDB as well as Chicago, the city I live in. The database is very informative and interesting. However, it seems to me that the data is not comprehensive. For example, I believe there are much more restaurants in Chicago!

Besides what I have discussed above, there are also much more to explore. Some attributes such as street number in the dataset can be better cleaned as well. The dataset is very large and it is really fun to play with it.