Openstreetmap Case Study

Map Area

Beijing, China

- http://www.openstreetmap.org/relation/912940#map=8/40.256/116.461 (http://www.openstreetmap.org/relation/912940#map=8/40.256/116.461)
- https://mapzen.com/data/metro-extracts/metro/beijing_china/ (https://mapzen.com/data/metro-extracts/metro/beijing_china/)

This map is of the city that I've lived for a few years. Considering I'm quite familiar with its geographical information, so I think it would be easier for the later data wrangling and exploration process.

Problems Encounted in the Map

After I downloaded this dataset, I took a sample(i.e., `sample_divide_by10.osm`) from this dataset, since the original one(i.e., `beijing_china.osm`) was a little huge. Then I imported the raw sample data to sqlite and queried for 30 most common tags in the `nodes_tags` database. By looking through the most common tags in the sample osm file, I got a rough understanding about the dataset. Also, I went to the forum to get inspiration about the problems that other people encounted in their dataset.

Here are some problems I initially encounted in the map(Actually, when I moved forward during my analysis, I encounted other problems, which I would discuss in the conclusion part). The first three problems were handled in this analysis.

- incorrect postcodes, since common postcodes for Beijing are 6 digit long, some postcodes are less than or larger than 6 digit, like '10060', '10040', '1111' and etc.
- inconsistent phone numbers, especially the prefix problem. The prefix has many variants. If we consider international area code(i.e., 86 or 0086 or +86) as prefix A, domestic area code(i.e., 010 or 10) as prefix B, then we can at least see 4 different prefixes AB/A/B/-.
- a few tags with the key "FIXME" or "fixme", underlying some information with the node needs to change later. (This problem is easy to deal with. In later process, I just ignoreed this node. So it's not discussed in later analysis.)
- opening hours are inconsistent, like "24/7" and "Mo-Su 05:00-23:00" and etc.
- one or two exceptional problems: like `<tag k=″opening_hours″ v=″高碑店街道便民服务中心″ />`, key and value don't match here.

Incorrect postcodes

First, I collected all incorrect postcodes. A postcode for area in Beijing was considered right when it was 6-digit long and started with '10'.

```
def audit_postcode(uncorrect_postcodes, postcode_value):
    if len(postcode_value) == 6:
        if postcode_value.startswith('10'):
            return True
    uncorrect_postcodes[postcode_value] += 1
    return False
```

Incorrect postcodes in this dataset had only 11 records, and there was no pattern in incorrect postcodes. So I decided to modify them manually according to other tags information about the node, but there were still 3 incorrect postcode reacord omitted.

```
def update_postcode(postcode, mapping):
    if postcode in mapping:
        modified_postcode = mapping[postcode]
        return modified_postcode
    else:
        return None
```

Inconsistent Phone Numbers

As described above, phone numbers were in different formats. Below were a few typical ones.

- 010-84477300: with domestic area code for Beijing 010
- 64359561: telephone number
- 01067654321: with domestic area code for Beijing 010, but no slash
- +861066069281: with international area code for China 86 and domestic area code for Beijing 10 (0 are omitted here)
- 15601077881: mobile phone number

And I wanted to make the phone numbers conform to the same format, '010-xxxxxxxx' for telephone number and '1xxxxxxxxxx' for mobile phone numbers. So I used regular expressions to finish this task。 Brackets or '-' were removed first, then the phone numbers without prefixes were taken out using the regular expressions. In this process, incorrect phone numbers that had less or more digit would be dropped. Actually, when doing this, I found two or three correct records were removed because of their formatting issue, like "+8610-88087384 ; 88086667" and "+86 8610 6437 6299".

```
phone_number_re = re.compile(r'^((00)?86)?(0?10)?(\d{8})$', re.IGNORECASE)


mobile_phone_number_re = re.compile(r'^((00)?86(10)?)?(1\d{10})$', re.IGNORECASE)


def clean_phone_number(phone):
# incorrect_phone_numbers will be dropped, while others need to conform to consistency
    phone_number = re.sub(r'\D', "", phone)
    m = phone_number_re.search(phone_number)
    n = mobile_phone_number_re.search(phone_number)
    if m:
        cleaned_phone_number = '010-' + m.group(4)
    elif n:
        cleaned_phone_number = n.group(4)
    else:
        cleaned_phone_number = None
    if cleaned_phone_number:
        print phone, '=>', cleaned_phone_number
    else:
        print phone, '=> omitted'
    return cleaned_phone_number
```

Data Overview ¶

This part gave an overview about this dataset, including the file sizes and the number of nodes, ways and tags.

File sizes

file	size
beijing_china.osm	190 MB
beijing_china.db	99 MB
nodes.csv	71 MB
nodes_tags.csv	3.0 MB
ways.csv	7.6 MB
ways_tags.csv	25 MB
ways_nodes.cv	8.6 MB

Number of nodes

```
sqlite> select count(*) from nodes;
```

895431

Number of ways

```
sqlite> select count(*) from ways;
```

132814

Number of tags

```
sqlite> select sum(count)
from
(select count(*) as count from nodes_tags
union
select count(*) as count from ways_tags);
```

346847

Explore Part I: Users, Contributions and Versions

First, let's find how many contributions were made. Actually, that was the sum of node numbers and way numbers.

Number of contributions

```
sqlite> select sum(count)
from
(select count(*) as count from nodes
union all
select count(*)as count from ways);
```

1028245

Number of Unique Users

```
sqlite> select count(distinct uid)
from
(select uid from nodes
union all
select uid from ways);
```

1934

Top 10 Contributors

```
sqlite> select user, count(*) as number
from
(select user from nodes
union all
select user from ways)
group by user
order by number desc
limit 10;
```

user	number
Chen Jia	237678
R438	140711
hanchao	70683
Алекс Мок	66093
ij_	51542
katpatuka	23406
m17design	22337
Esperanza36	18097
nuklearerWintersturm	15725
RationalTangle	13671

Sum and Proportion of Top 10 Users' Contributions

```
select sum(number)
from
(select user, count(*) as number
from
(select user from nodes
union all
select user from ways)
group by user
order by number desc
limit 10);
```

659943

And we could easily figure out the top 10 users contributed for about 64.2%. If we made similar query on top 20 users, then we could get the percentage for top 20 was about 72.1%(741418 contributions).

Number of Users with Single Contribution

```
sqlite> select count(*) from
(select user, count(*) as number
from
(select user from nodes
union all
select user from ways)
group by user
having number = 1);
```

483

Nearly a quarter people contributed only once.

Most Frequently Changed

I was curious to find which node or way changed most frequently. So I quried for the tags information about the node.

```
sqlite> select t1.id, t1.key, t1.value, t1.type
from
(select * from nodes_tags
union all
select * from ways_tags) t1
join
(select max(v), id
from
(select cast(version as int) as v, id from nodes
union all
select cast(version as int) as v, id from ways)) t2
on t1.id = t2.id;
```

Below is one segment about this node. It's the node describing the city. Not too surprised to get this result. When we quried for the version information, we got the version for this node was 88.

```
25248662|ref|京|regular
25248662|name|北京市|regular
25248662|rank|0|regular
25248662|place|city|regular
25248662|en|BJ|ref
25248662|vi|Kinh|ref
25248662|capital|yes|regular
25248662|DSG|ADM1|gns
25248662|UFI|-1898545|gns
25248662|UNI|9310007|gns
```

If we do another query, we could get that 840950 nodes or ways were of version 1, accounting for about 81.8% of the dataset.

```
select count(*) as number
from
(select version from nodes
where version = '1'
union all
select version from ways
where version = '1');
```

840950

Explore Part II: Tags Related

1. Top 10 Appearing Tags

When transforming the osm data to csv files, we treated tags in the following way:

- If the key of the tag had no colon, like 'phone', then key was still itself and type was assigned to 'regular'.
- If the key of the tag had at least a colon, like 'addr:street', we would split the value using the first colon index. The value of type was assigned to first part, and the left part was assigned to key.

So when trying to find the most frequently appearing tags, I dealed with them separately. If type equaled 'regular', then I queried for the key. Otherwise, I would query for the type value.

```
sqlite> select kind, count(*) as number
from
(select key as kind from nodes_tags where type = 'regular'
union all
select type as kind from nodes_tags where not type = 'regular'
union all
select key as kind from ways_tags where type = 'regular'
union all
select type as kind from ways_tags where not type = 'regular')
group by kind
order by number desc
limit 10;
```

We can see from the results that many tags are related to the traffic, including "highway"/"oneway"/"public_transport"/"railway"/"bridge", accounting for a half of top 10 appearing tags.

tag	number
highway	78558
building	52463
name	49516
oneway	21057
power	14112
source	9981
public_transport	9875
railway	8088
layer	7998
bridge	7965

2. Land Use Situation in Beijing

Here, I would like to draw a pie chart to intuitively represent landuse condition in Beijing. We could see from the pie chart that almost a half of the land was used for residential. But I also noticed the value "yes" appeared twice here, which was not appropriate.

In [13]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sqlite3
%matplotlib inline
```

In [10]:

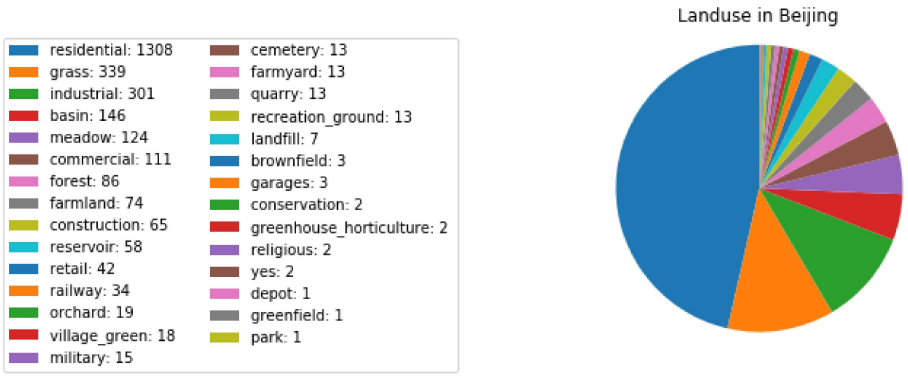
```
def retrieve_data_from_sqlite(query):
    db = sqlite3.connect('beijing_china.db')
    cur = db.cursor()
    cur.execute(query)
    return cur.fetchall()
```

In [55]:

```
sql = """
select value, count(*) as number
from (select value from nodes_tags
where key = 'landuse'
union all
select value from ways_tags
where key = 'landuse')
group by value
order by number desc
"""
landuse = pd.DataFrame(retrieve_data_from_sqlite(sql))
patches, texts = plt.pie(landuse[1], startangle=90, radius=1.2)
plt.axis('equal')
plt.title('Landuse in Beijing')
labels = ['{0}: {1} '.format(i, j) for i,j in zip(landuse[0], landuse[1])]
plt.legend(patches, labels, loc='left center', bbox_to_anchor=(-0.1, 1.), fontsize=10, ncol=2)
# reference:
# https://stackoverflow.com/questions/23577505/how-to-avoid-overlapping-of-labels-autopct-in-a-matplotlib-pie-chart
```

Out[55]:

```
<matplotlib.legend.Legend at 0x250f2c88>
```



3. Data Source

I noticed there were many tags describing data source. So I would like to explore source distribution a little. But when I looked at the result, I realized that the source data was quite dirty, including case sensitive issue, records with timestamp, Chinese characters included and etc, like 'Bing'/'bing'/'Bing imagery'/'Bing, 2005-04'. So this field still needs later attention and cleaning.

```
sqlite> select value, count(*) as number
from
(select value from nodes_tags
where key = 'source'
union all
select value from ways_tags
where key = 'source')
group by value
order by number desc;
```

But from above query, we could easily guess that Bing was the biggest data source. So I used the following sql to prove this. We could get that Bing acted as the biggest data source, accounted for 81.75% in all data source records.

```
sqlite> select t2.bing_number * 1.0 / t1.total_number
from
(select count(*) as total_number
from
(select value from nodes_tags
where key = 'source'
union all
select value from ways_tags
where key = 'source')) t1,
(select count(*) as bing_number
from
(select value from nodes_tags
where key = 'source' and lower(value) like '%bing%'
union all
select value from ways_tags
where key = 'source' and lower(value) like '%bing%')) t2;
```

81.75%

4. Number of universities

Beijing has many universities.

How to find a node is a university? Through the relative tag: `<tag k="amenity" v="university" />`. So in database, a university would have a key named "amenity" and a value called "university". Both conditions need to be satisfied, since other tags may have value "university", like `<tag k="building" v="university" />`.

```
sqlite> select count(distinct id)
from
(select id from nodes_tags
where key = 'amenity' and value = 'university'
union all
select id from ways_tags
where key = 'amenity' and value = 'university');
```

97

That's quite a big number. I went to wiki (https://en.wikipedia.org/wiki/List_of_universities_and_colleges_in_Beijing), it says there are at least 70 establishments of higher education.

I chose to look further into the data, so I first queried the id information and selected two or three id and looked at the tags having the same id.

```
sqlite> select * from ways_tags where id = '515209524';
```

```
515209524|amenity|university|regular
515209524|name|中国人民大学北校区|regular
```

```
sqlite> select * from ways_tags where id = '484156511';
```

```
484156511|amenity|university|regular
484156511|name|中国科学院大学雁栖湖校区|regular
```

The university with id "515209524" is a campus of the Renmin University, since the university may have more than 1 campus in Beijing. This is also true for id "484156511". Using the above query for the number of universities, we may count one university more than once. So the number 97 overestimated a little.

5. Number of tourism and its distribution

Since Beijing has many places of interest, I would like to know how many tourism sites are there in this city.

```
select count(distinct id) from
(select id from nodes_tags
where key = 'tourism'
union all
select id from ways_tags
where key = 'tourism');
```

1393

Also, I would want to figure out the distribution. Actually, not all tags with the key 'tourism' were places of interest. From the below query result, hotels or motels were also included in this category. But there were still 295 attration sites in Beijing, including the world well known Great Wall and others.

```
sqlite> select value, count(*) as number
from
(select value from nodes_tags
where key = 'tourism'
union all
select value from ways_tags
where key = 'tourism')
group by value
order by number desc
limit 5;
```

tourism name	number
hotel	478
attraction	295
motel	168
museum	86
viewpoint	68

6. Amenity Distribution & Popular Food

```
sqlite> select value, count(*) as number
from
(select value from nodes_tags
where key = 'amenity'
union all
select value from ways_tags
where key = 'amenity')
group by value
order by number desc
limit 10;
```

Restaurants accounted for a considerable part in amenity.

amenity name	number
restaurant	1501
parking	864
school	541
bank	473
toilets	441
fast_food	351
cafe	306
fuel	302
hospital	194
bar	168

Next, I would try to explore what kind of food are most popluar in Beijing except Chinese food.

```
sqlite> select value, count(*) as number
from
(select value from nodes_tags
where key = 'cuisine' and (not value = 'chinese')
union all
select value from ways_tags
where key = 'cuisine' and (not value = 'chinese'))
group by value
order by number desc
limit 10;
```

Actually, this field was also quite dirty. Some values were a list of words, like `"american;burger"`. And `"pizza"` and `"pizza;american"` are somewhat overlapped. But I had the confidence to say that fast foods were pretty popular including burger/pizza/sandwich/etc.

food type	number
coffee_shop	62
chicken	55
american;burger	53
japanese	29
italian	20
pizza	15
pizza;american	15
regional	14
sandwich	14
burger	14

Conclusion & Suggestion

There are still pretty much to clean in this dataset. The following summarized what I found during my exploration with this dataset.

- Some tag fields, like "source" and "cuisine", the values were quite dirty, since they didn't conform to the same rule. For example, the value for the key "source" may be `'Bing'/'bing'/'Bing imagery'/'Bing, 2005-04'`. It's better to have consistency.
- A few tags have Chinese characters as keys, like "应急避难场所总面积万平米", "开发商" and others, which would increase the difficulty for data wrangling and analysis. I suggest to change them to English.
- Some tag values would be 'yes', which would be quite difficult for classification.

I want to give a suggestion about contribution to dataset. Documentation is highly recommened in contribution, since everyone may consider things differently. If the documentation writes down the common names or rules with detail, then many inconsistency problems can be reduced or even avoided. And to go further, in some cases, limited items can be given in a select box and users can choose from them, rather than deciding on their own.