OpenStreetMap_Data_Wrangling

February 11, 2017

Map Area
Brooklyn, NY, United States

1. Map URL
2. OSM document for Download
3. OSM XML wiki

1 Problems Encountered in Exploring the Map Data

The original Brooklyn Map is initially stored in XML format with the size of 637M. A sample of is used for a pre-investigation exploration. To get started, I applied the Output.py file to the sample to do 2 things: 1. Cleaned up misused tags; 2. Transferred the well-structured XML format osm into 5 tabular csv files(namely nodes, nodes_tags, ways, ways_nodes, ways_tags). After that, I stored the csv files in the SQLite database and used SQL to explore the data. I used the following query to find the top 5 commonly-used tags(street, housenumber, postcode, city, and amenity) as my targets to begin with. Problems are as following:

Improper layer structure in the key attributes
Inconsistency of key of Tags
Overabbreviated street names
Unstandardized City Names

1.1 Improper layer structure in the key attributes

Two layer indicator have been found in the key attributes ":" and ".". Typically, The ":" often co-exist with 'addr' and 'gnis'.

Take 'addr:street' as an example, the corresponding type attribute is 'regular' which gives low-level information. Instead of k = 'addr:street' and type = 'regular', it's better to have k = 'street' and type = 'addr'.

```
In [58]: def col_splt(j):
             ele={}
             pos = j.find(":")
             if pos < 0: #as the tag type
                 ele['key'] = j
             else:
                 ele['type'] = j[:pos]
                 ele['key'] = j[pos+1:]
             return ele
```

After applying the function to all tags, the key shows as following:

id	key
419366584	cityracks.street
1827182430	street
1827182430	housenumber

Different from ":", the "." is used in describing 'cityracks' in format of "cityracks.attribute". For example, "cityracks.installed" means the time when the cityrack was installed. The key is set to describe the object in the map such as street, housenumber and postcode. It doesn't make sense to make citycracks as type and the attribute of cityracks as key like what's done with the above case. The problem should be treated differently and be solved when more investigations on cityracks is needed.

1.2 Inconsistency of key of Tags

Different types of key are used to describe the key. For example, 'street_2', 'street' both exist and no big difference has been found after investigating the values. This occurs in 4 other types of keys. To standardize the key format, I used SQL query1 to change the 'street_2' value into 'street' and query2 to check if all changes have been done properly.

```
In [59]: q1 = "UPDATE nodes_tags SET key='street'
              WHERE key = 'street_2';"
         q2 = "SELECT key, value FROM nodes_tags
              GROUP BY key ORDER BY COUNT(*) DESC LIMIT 10;"
```

The result of query 2:

[('addr:housenumber', '533'), ('addr:street', 'Grand Street'), ('addr:postcode', '11211'), ('amenity', 'restaurant'), ('name', 'Le Barrico'), ('natural', 'tree'), ('addr:city', 'Brooklyn'), ('addr:state', 'NY'), ('capacity', '4'), ('cityracks.housenum', '100')]

1.3 Overabbreviated street names

Some of the abbreviation of street names was observed when querying the street data. For example, it has names of 'Monroe St', 'Park Ave' instead of 'Monroe Street', 'Park Avenue'. I used the find_street.py to sort out all types of street type. After viewing the result, I updated the mapping dict to include 'Blvd': 'Boulevard'and apply the update_name() to all values with key of street to change the abbreviation into its full spelling.

```
In [60]: def update_name(name, mapping):
             type = street_type_re.search(name).group()
             if type in mapping:
                 nm = name.split(type)[0]
                 name = nm + mapping[type]
             return name
```

After applying the function, all abbreviations occurred in mapping dict is updated.

1.4 Unstandardized City Names

In querying database grouped by city names and ordered by count, I found the below top 10 'cities'. Even though it's the map of Brooklyn some other cities like 'New York', 'Hoboken' and 'Forest Hills'. There is a problem of 'New York City' and 'New York'(At least it's a problem for foreigners like me who was confused about States of New York and the city of New York.). In this case, These two city names can be combined together as 'New York'. The amendment can be achieved by update query as shown above.

[('Brooklyn', 940), ('New York', 605), ('Hoboken', 90), ('New York City', 39), ('Forest Hills', 38), ('Jersey City', 12), ('Elmhurst', 10), ('Rego Park', 8), ('New York, NY', 4), ('Queens', 4)]

Let's move on to another problem. Some of the city names have redundant abbreviations 'NY' or State names like 'New York', others have problems of the case like 'brooklyn'. Let me standardize all by rdc() and up_cs() functions.

[('New York, NY'), ('Brooklyn, NY'), ('NEW YORK CITY'), ('Brooklyn, New York'), ('Glendale, NY'), ('Manhattan NYC'), ('Ozone Park, NY'), ('brooklyn'), ('new york'), ('queens')]

```
In [61]: def rdc(a):
             return a.split(',')[0]
         def up_cs(a):
             if " " in a:
                 n = a.index(" ") #only work for names of two words
                 return a[0].upper()+a[1:n]+" "+a[n+1].upper()+a[n+2:]
             else:
                 return a[0].upper()+a[1:]
```

After applying all above cleaning and cleaner data sets are ready for further exploration.

2 Data Overview

2.1 Basic statistics

```
In [ ]: import sqlite3
        path = "C:/sqlite_windows/sqlite_windows/brooklyn"
        conn = sqlite3.connect(path)
        c = conn.cursor()
```

2.1.1 File size

Filename	Size
brooklyn sqlite database	402 MB
nodes.csv	224 MB
nodes_tags.csv	8.31 MB
ways.csv	32 MB
ways_nodes.csv	83.7 MB

Filename	Size
ways_tags.csv	82.9 MB
city_racks.csv	465 KB

2.1.2 Number of nodes

```
In [63]: query = "SELECT count(*) FROM nodes;"
         c.execute(query)
         rst = c.fetchall()
         rst[0][0]

Out[63]: 2494350
```

2.1.3 Number of ways

```
In [64]: query = "SELECT count(*) FROM ways;"
         c.execute(query)
         rst = c.fetchall()
         rst[0][0]

Out[64]: 492599
```

2.1.4 Number of unique users

```
In [65]: query = "SELECT COUNT(DISTINCT(uid)) FROM (
                  SELECT uid FROM nodes UNION ALL SELECT uid FROM ways);"
         c.execute(query)
         rst = c.fetchall()
         rst[0][0]

Out[65]: 1626
```

2.1.5 Years having new records

```
In [66]: query = "SELECT year, COUNT(id) c FROM (
                  SELECT id, SUBSTR(timestamp, 1,4) year FROM (
                  SELECT id, timestamp FROM nodes UNION ALL
                  SELECT id, timestamp FROM ways) a ) b
                  GROUP BY year
                  ORDER BY c DESC LIMIT 10;"
         c.execute(query)
         rst = c.fetchall()
         rst

Out[66]: [(u'2013', 2158126),
          (u'2014', 635321),
          (u'2016', 82290),
          (u'2015', 46794),
```

```
 (u'2009', 20759),
 (u'2012', 19806),
 (u'2011', 8872),
 (u'2010', 8277),
 (u'2017', 5030),
 (u'2008', 1634)]
```

2.1.6 Top 10 Contributors

```
In [67]: query = "SELECT user, COUNT(*) n FROM (
                  SELECT user FROM nodes UNION ALL SELECT user FROM ways)
                  GROUP BY user ORDER BY n DESC LIMIT 10;"
         c.execute(query)
         rst = c.fetchall()
         rst

Out[67]: [(u'Rub21_nycbuildings', 1739482),
          (u'ingalls_nycbuildings', 373540),
          (u'ediyes_nycbuildings', 189665),
          (u'celosia_nycbuildings', 117354),
          (u'ingalls', 105122),
          (u'lxbarth_nycbuildings', 79586),
          (u'aaron_nycbuildings', 41997),
          (u'ewedistrict_nycbuildings', 35002),
          (u'smlevine', 25018),
          (u'robgeb', 24392)]
```

2.1.7 Top 10 populous cuisine

```
In [68]: query = "SELECT value, COUNT(*) FROM nodes_tags
                  WHERE key = 'cuisine' GROUP BY value
                  ORDER BY COUNT(*) DESC LIMIT 10;"
         c.execute(query)
         rst = c.fetchall()
         rst

Out[68]: [(u'coffee_shop', 79),
          (u'pizza', 76),
          (u'mexican', 60),
          (u'american', 58),
          (u'italian', 56),
          (u'burger', 54),
          (u'chinese', 45),
          (u'sandwich', 29),
          (u'french', 27),
          (u'japanese', 23)]
```

2.1.8 Appearing list of place for visit

```
In [69]: query = "SELECT value, COUNT(*) FROM nodes_tags
                  WHERE key = 'tourism' GROUP BY value
                  ORDER BY COUNT(*) DESC"
         c.execute(query)
         rst = c.fetchall()
         rst

Out[69]: [(u'artwork', 50),
          (u'attraction', 44),
          (u'hotel', 35),
          (u'viewpoint', 25),
          (u'museum', 21),
          (u'information', 20),
          (u'picnic_site', 18),
          (u'hostel', 6),
          (u'guest_house', 4),
          (u'camp_site', 3),
          (u'motel', 2)]
```

3 Suggestions

3.1 Cityracks as subtable to be further explored

As discussed before, for this particular area, I noticed the key of nodes contains one 'problematic' type concerning cityracks. There are 14270 tags about cityracks in the nodes_tags data set and it could be ranked as the 4th most-appearing tag keys.Thus, in dealing with the data, it's necessary to sort it out into a subtable. I extracted the data from nodes_tags.csv and use doc_splt() function to make the change.

```
In [70]: query = "SELECT key,COUNT(*) FROM nodes_tags GROUP BY key
                  ORDER BY COUNT(key) DESC LIMIT 10;"
         c.execute(query)
         rst = c.fetchall()
         rst

Out[70]: [(u'street', 57366),
          (u'housenumber', 57317),
          (u'postcode', 56612),
          (u'amenity', 6941),
          (u'name', 6274),
          (u'highway', 3672),
          (u'capacity', 2926),
          (u'cityracks.housenum', 2614),
          (u'cityracks.large', 2614),
          (u'cityracks.rackid', 2614)]
```

```
In [71]: query = "SELECT COUNT(*) FROM nodes_tags
                  WHERE key LIKE 'cityrack%';"
         c.execute(query)
         rst = c.fetchall()
         rst[0][0]

Out[71]: 14270
```

sqlite> .output citycracks.csv sqlite> SELECT * FROM nodes_tags
WHERE key LIKE 'cityracks%';

sqlite> .output stdout

```
In [73]: import pandas as pd
         cityracks = pd.read_csv('cityracks.csv',
                     names =["id", "key1","value","type1"])
         cityracks['key'] = cityracks['key1'].apply(
                            lambda x: x.split(".")[-1])
         cityracks['type'] = 'cityracks'
         city_racks = cityracks.drop(['key1','type1'], axis=1)
         city_racks.head()
```

Out[73]:

	id	value	key	type
0	419359776	0	large	cityracks
1	419359776	1	small	cityracks
2	419359776	242	rackid	cityracks
3	419359776	Windsor Pl	street	cityracks
4	419359776	153	housenum	cityracks

By separating the cityracks data, It's easier to get more insight specifically on cityracks without redundant information. There are 7 fields of information about cityracks including housenum, installed, key, large, rackid, small and street in cityracks data set. With this information, city planners, business and even for potential visitors like me could solve many problems. For example, by searching cityrack distribution in streets, we may get to know where should new cityracks be installed together with other information. By knowing when the cityracks were installed, we may predict the timing maintenance and renewal!

3.1.1 Numbers of cityracks

```
In [74]: query = "SELECT COUNT(DISTINCT(id)) FROM cityracks;"
         c.execute(query)
         rst = c.fetchall()
         rst[0][0]

Out[74]: 2615
```

3.1.2 Time installed

```
In [75]: query = "SELECT year, COUNT(DISTINCT(id)) c FROM (
                  SELECT id, SUBSTR(value, -4) year FROM cityracks
                  WHERE key = 'installed') sub GROUP BY year
                  ORDER BY c DESC;"
         c.execute(query)
         rst = c.fetchall()
         rst

Out[75]: [(u'2009', 234),
          (u'2000', 232),
          (u'2008', 194),
          (u'2001', 134),
          (u'2005', 102),
          (u'2002', 84),
          (u'2006', 81),
          (u'2004', 80),
          (u'2007', 42),
          (u'/200', 17)]
```

4 Conlusion

It's quite hard to evaluate the data if terms of its accuracy, completeness. What's sure is that the data set will contain more details on contributions from the OpenStreetMap community. What I have done is more about adding constraints when making the csv table as well as some improve consistency with the data set. Overall, the map of Brooklyn has already done a good job in providing some useful information for potential visitors like me. Brooklyn indeed has a lot of attractions and artworks to visit and ... different cuisine to enjoy!