OpenStreetMap Data Case Study

Map Area

Morgan Hill, CA, United States

- www.openstreetmap.org/export#map=12/37.1550/-121.7005

<bounds minlat="37.3300000" minlon="-121.8804000" maxlat="37.3344000" maxlon="-121.8749000"/>

This map is close to my current place of living. This being one of my initial projects on data munging and I wanted a simple data, with less crossroads, so I chose one straight freeway 101. This turned out to be pretty clean data.

Problems Encountered in the Map

I used the overpass API to download the selected area of the open street map. I noticed three main problems with the data, which have been mentioned below. I have shown the python code solution for two of the problems here, street names and phone numbers.

- Over abbreviated street names *(“1425 E Dunne Ave”)*
- Phone numbers written inconsistently. Some of the phone numbers do not have country code. Some of them have area code inside parenthesis, while others are not.

- Inconsistent postal codes *(“95037-2504”)*. The Postal code should be a 5 digit number, here there is 5 digit number followed by a 4 digit number.

- Street names in second level `“k”` tags pulled from Tiger GPS data and divided into segments, in the following format:

```
<tag k="tiger:county" v="Santa Clara, CA”/>

<tag k="tiger:name_base" v="Santa Teresa”/>

<tag k="tiger:name_type" v="Blvd"/>
```

Over abbreviated Street Names

Once the data was imported to SQL, some basic querying revealed street name abbreviations and postal code inconsistencies. I ran the audit function on the xml data first to see what street names were inconsistent. I found three types which were not as expected, Dr, Ave., Ave. These three were kept in the mapping key. I iterated through each street name in the add:street tag, and replaced the portion of the street name with the correct mapping through python code, as shown here in the update function:

```
def update_name(name, mapping=STREET_TYPE_MAPPING):
    m = STREET_TYPE_RE.search(name)
    if m:
        street_type = m.group()
        if street_type in mapping.keys():
            better_name = re.sub(street_type,
                                 mapping[street_type], name)
            print (name, "=>", better_name)
            return better_name
```

```
        elif street_type not in EXPECTED:
            print ('Unknown street type: ', street_type)
    return name
```
This updated all substrings in problematic address strings, such that:

“Fountain Oaks Dr” changed to “Fountain Oaks Drive”

And

“Peak Ave” changed to “Peak Avenue”

Inconsistent phone types

I observed that not all the phone numbers followed a consistent representation. To solve this I first created an expected regular expression. If the phone number matched then took all the digits of the phone number and then arranged them in a consistent international format given on the OSM website **phone=**+<country code> <area code> <local number>.

I raised an error as ‘Unknown phone type’ if it did not match the regular expression.

```
def update_phone(phone):
    m = PHONE_RE.search(phone)
    better_phone = phone
    if m:
        if m.group(1) is None:
            country_code = '+1'
        else:
            country_code = m.group(1)
        area_code = m.group(2)
```

```
        local_num1 = m.group(3)
        local_num2 = m.group(4)
        better_phone = country_code+'
             '+area_code+'-'+local_num1+'-'+local_num2
        print (phone, "=>", better_phone)
    else:
        print ('Unknown phone type: ', phone)
    return better_phone
```

This updated the ones that matched regular expression, and returned others as is with error message.

408-779-3721 => +1 408-779-372 (Country code added)

+1 408 779 6229 => +1 408-779-6229 (‘-‘added)

4087792519 => +1 408-779-2519 (country code and ‘-‘ added)

Unknown phone type: 408782588 (Only 9 digits)

Data Overview and Additional Ideas

I used Sqlite for database. I created the tables in the database by making use of schema.py file. I imported the csv files into the Sqlite database using import command.

The section below contains the basic statistics about the dataset, the SQL queries used to gather them, and some additional ideas about the data in context.

File sizes

```
MorganHill.xml ......... 61.4 MB
MorganHill.db .......... TB MB
nodes.csv .............. 23.9 MB
nodes_tags.csv ......... 184 KB
ways.csv ............... 1.9 MB
ways_tags.csv .......... 2.4 MB
ways_nodes.cv .......... 8 MB
```

Number of nodes

```
sqlite> SELECT COUNT(*) FROM nodes;
```
292201

Number of ways

```
sqlite> SELECT COUNT(*) FROM ways;
```
33353

Number of unique users

```
sqlite> SELECT COUNT(DISTINCT(e.uid))
FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM
ways) e;
```

255

Top 10 contributing users

```
sqlite> SELECT e.user, COUNT(*) as num
FROM (SELECT user FROM nodes UNION ALL SELECT user FROM
ways) e
```

```
GROUP BY e.user
ORDER BY num DESC
LIMIT 10;
```

```
nmixter                 204673
mk408                   53920
user8192                13393
"Alexander Avtanski"    6860
stevea                  6076
beddy                   4394
"Chris Lawrence"        2992
woodpeck_fixbot         2089
Apo42                   1751
3vivekb_sjsidewalks_import  1540
```

Number of users appearing only once (having 1 post)

```
sqlite> SELECT COUNT(*)
FROM
    (SELECT e.user, COUNT(*) as num
     FROM (SELECT user FROM nodes UNION ALL SELECT user
FROM ways) e
     GROUP BY e.user
     HAVING num=1)  u;
```

```
44
```

Additional Ideas

Additional Data Exploration

Top 10 appearing amenities

```
sqlite> SELECT value, COUNT(*) as num
FROM node_tags
```

```
WHERE key='amenity'
GROUP BY value
ORDER BY num DESC
LIMIT 10;
```

```
restaurant          42
place_of_worship    18
fast_food           17
bench               13
cafe                13
fuel                11
toilets             11
telephone           8
bank                7
dentist             6
```

We see that there are amenities like restaurant, cafe, fuel etc in the selected map area.

Let's check the cuisine too.

```
sqlite>  SELECT count (*) FROM node_tags WHERE
key='cuisine';
```

```
23
```

We see that though there are 42 restaurants, the cuisine is associated with only 23 out of them.

Religion

```
sqlite> SELECT node_tags.value, COUNT(*) as num
FROM node_tags
    JOIN (SELECT DISTINCT(id) FROM node_tags) i
    ON node_tags.id=i.id
WHERE node_tags.key='religion'
GROUP BY node_tags.value
ORDER BY num DESC;
```

```
christian 95

Buddhist 3
```

Suggestions for Improvement of Data

I particularly analyzed the amenities data further ahead. After getting the list of nodes for key as amenity and value as restaurant, I found that the tags associated were very different from node to node. Below are 2 examples.

```
sqlite> SELECT * FROM node_tags WHERE id= '4077183219';

4077183219,amenity,restaurant,node
4077183219,name,"Sitar Indian Food”,node
```

The output from the above query shows name of the amenity restaurant shown. No other details given apart from the name. Then I randomly picked another node_id and queried the output.

```
sqlite> SELECT * FROM node_tags WHERE id= '325620223';

325620223,addr:city,"Morgan Hill",node
325620223,addr:country,US,node
325620223,addr:county,"Santa Clara",node
325620223,addr:housenumber,201,node
325620223,addr:postcode,95037,node
325620223,addr:state,CA,node
325620223,addr:street,"Tennant Station",node
325620223,amenity,restaurant,node
325620223,cuisine,pizza,node
325620223,name,"Mountain Mike's Pizza",node
325620223,phone,"+1 408-779-6900",node
325620223,website,http://mountainmikes.com/,node
```

The above output shows name of the restaurant, plus 3 other details like cuisine, phone number and website name. Now if I want to get the data based on cuisine parameter, I would not be able to fetch the previous restaurant name, as the cuisine is not tagged with it. Also on seeing the raw data I observed some of the restaurants had open-close timings mentioned. So one way the restaurant data can be improved is to have some minimum tag values associated with each data point. Name, phone number, and open-close information are some of the handy values that travelers can use.

Ideas for additional improvement of data.

Currently there is no way to judge the authenticity of the data. So many people have created this data together. The margin of error is very high. There can be some fixed process for creating this data in the first place. But now when so much of data is already created there can be some review process on the existing data.

One way of doing this could be to have few more tags associated with each amenity node to capture following 3 values. This can be done for each node also not for amenity alone. Though amenity is of particular use, but sometimes travelers need other details on ‘vista point’ etc.

Count of total number of people who reviewed the data

Count of people voted it as - Yes

Count of people voted it as - No

Something similar to the ‘Like’ button we have on the Facebook shares.

Lets say there is a node with 10 reviews. 6 people voted it as yes, and 4 people voted it as no. The authenticity of this can be left to the users discretion. If the user feels satisfied with more than 50% of people saying 'Yes' to the data, he can make his judgement based on that.

A well formed data also gives more power to the user. It can help individuals and companies do the 'Market Research'. If someone wants to pursue a business idea of opening a restaurant in this area, then he/she needs to know how many restaurants, of what cuisine are already available along that route. A better informed decision can be made with improved data.

Conclusion

After this review of the data I feel Morgan Hill area has been well cleaned for the purposes of this exercise. I am more interested to pick a more skewed data next time. I will be picking up data related to my home country India next time for analysis. I am sure it will be a very exciting exploration.