OpenStreetMap Data Case Study

Completed by Stephen Fox
Udacity Data Analyst Nanodegree
December 2016

Map Area Selected

I elected to choose Honolulu, HI, United States:
https://mapzen.com/data/metro-extracts/metro/honolulu_hawaii/

This extract contains the entire island of Oahu. It is 54.4MB in size, when unzipped. Honolulu is the main city on Oahu and is also the capital of the state of Hawaii. I chose this area because I've visited the island once (in 2006) and had a great trip. I hope to return one day and no doubt will be interested in the various ways and nodes.

Coding Process

The process used to address this project was as follows:

1. OSM file preparation
2. Data exploration
3. Data auditing
4. Data fixing
5. Data shaping, validating and exporting for further analysis via SQL

The code developed to address these five steps is provided in the data.py file. The OSM file preparation step involved implementing (Udacity provided) code for selecting a subset of the OSM data, named sample.osm, for the purpose of speeding up the testing, debugging, exploration, auditing and data fixing process. The final run of the code was performed on the complete honolulu.osm data set. The data exploration step involved getting a feel for the data and generating some high level statistics, such as the number of various tags, the number of contributing users and the type of 'other' tags in the set. The data auditing step involved identifying problems with the address portion of the data, specifically the street type, state abbreviation and zip codes. The data fixing step involved programmatically addressing the issues identified during the auditing step. Finally, the data was shaped into a form compatible with being able to do numerous database queries and was exported to .csv format, for import into an SQL database.

Problems Encountered in the Map

The initial exploratory step showed that there were inconsistencies in at least the following three address attributes:

- 'addr:street'
- 'addr:postcode'
- 'addr:state'

In the case of street addresses, it was noted that there was inconsistency in the use of street types. For example, sometimes ‘St’ or ‘St.’ would be used instead of ‘Street’. Consistency in datasets is highly desirable, since query results will be inaccurate if the assumed types are not used with 100% consistency. In the case of zip codes, it was noted that the zip code was mostly the standard five-digit format, but in some instances, the zip code would include the extra four-digit portion or would also include the state abbreviation (HI). In the case of the state attribute, there were instances where something other than the two-letter code for Hawaii (HI) was being used, such as ‘hawaii’, ‘Hawaii’, ‘hi’, or ‘Hi’.

Code blocks were generated to audit the data programmatically and identify all instances where the standard format was being violated. These codes blocks were modeled on the process used in the Udacity OSM Case Study examples. For example, the code used to audit whether the ‘addr:state’ attribute contained ‘HI’ for all nodes and ways is as follows:

```
def is_state(elem):
    return (elem.attrib['k'] == "addr:state")

def audit_state(osmfile):
    osm_file = open(osmfile, "r")
    prob_state = set()
    for event, elem in ET.iterparse(osm_file, events=("start",)):

        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):
                if is_state(tag):

                    if tag.attrib['v'] != 'HI':
                        prob_state.add(tag.attrib['v'])
    osm_file.close()
    return prob_state
```

The results for the three audit steps are presented in Appendix A. In the case of the street type names, the potentially problematic street types were then manually analyzed. Those that were obvious candidates for fixing (e.g. ‘Ave’ being used in place of ‘Avenue’, ‘Dr’ being used in place of ‘Drive’) were added to a mapping dictionary that could be used to iterate over all elements, and make any updates, during the fixing step. In the case of zip code and state abbreviation issues, the fix was simpler. A review of the data showed that the number of error types was small and could be programmatically addressed with a simple set of rules. For example, the code used to fix instances of bad state abbreviation data is as follows:

```
def fix_state(elem):

        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):
                if is_state(tag):
                    if tag.attrib['v'] != 'HI':
                            print "Fixed State: ", tag.attrib['v'], "=> 'HI'"
                            tag.attrib['v'] = 'HI'
```

This fix is specific to the results of the audit step, since all errors were instances of Hawaii being incorrectly abbreviated (see Appendix A). Had I found a different set of errors (e.g. data containing states other than HI, such as AZ, MI etc.) then a different coding solution would have been warranted.

Data Overview

The sizes of the files used in this project are as follows:

- honolulu.osm 54.4 MB
- honolulu.db 29.4 MB
- nodes.csv 21.1 MB
- ways_nodes.csv 7.1 MB
- sample.osm 5.5 MB
- ways_tags.csv 3.6 MB
- ways.csv 1.6 MB
- nodes_tags.csv 0.5 MB

Other statistics from the database (and the associated SQL code) now follows:

Number of nodes:

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

249,611

Number of ways:

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

26,377

Both of these numbers were identical to the values found during the exploratory phase conducted in python (data.py), which was a good validation that data was not lost during the export step.

Number of unique users (uid):

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

477

This number is 5 users lower than the value found during the exploratory phase conducted in python. This is likely due to a few tags being dropped during the shaping and exporting process, due to problematic characters.

Number of Starbucks Coffee Shops:

```
sqlite> SELECT COUNT(DISTINCT(id)) FROM nodes_tags WHERE
value='Starbucks';
```

12

There are at least 12 Starbucks coffee shops on the island, so you will be able to get your caffeine fix on vacation there. There are likely more (see ‘Additional Ideas’ section for further discussion).

Top 10 cuisine options:

```
sqlite> SELECT nodes_tags.value, COUNT(*) as num
   ...> FROM nodes_tags
   ...> JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE
value='restaurant') b
   ...> ON nodes_tags.id=b.id
   ...> WHERE nodes_tags.key='cuisine'
   ...> GROUP BY nodes_tags.value
   ...> ORDER BY num DESC limit 10;
value,num
japanese,10
pizza,7
american,6
regional,6
chinese,5
indian,5
asian,4
international,4
italian,4
thai,3
```

It appears that Japanese cuisine is the most popular restaurant type on the island. This seems reasonable, given the proximity of Hawaii to Japan and the ready supply of fresh fish.

Restaurant wheelchair accessibility:

```
sqlite> SELECT nodes_tags.value, COUNT(*) as num
   ...> FROM nodes_tags
   ...> JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE
value='restaurant') b
   ...> ON nodes_tags.id=b.id
   ...> WHERE nodes_tags.key='wheelchair'
   ...> GROUP BY nodes_tags.value
   ...> ORDER BY num DESC;
value,num
yes,26
unknown,7
no,2
```

For those that require wheelchair accessibility, it appears there are at least 26 restaurants that are accessible on the island, there are 2 that should be avoided and there are 7 ('unknown') where it would pay to call ahead and determine accessibility.

Additional Ideas

While perusing the data via SQL, I noticed at least two issues that I had not addressed during the python based data exploration, auditing and fixing step:

- Erroneous ‘addr:city’ values
- Inconsistent chain store naming

In the case of the city name values, the following query identifies an issue:

```
sqlite> SELECT tags.value, COUNT(*) as count
   ...> FROM (SELECT * FROM nodes_tags UNION ALL
   ...>       SELECT * FROM ways_tags) tags
   ...> WHERE tags.key='city'
   ...> GROUP BY tags.value
   ...> ORDER BY count DESC limit 12;
value,count
Honolulu,546
"Hale'iwa",19
Aiea,14
Kailua,12
Haleiwa,8
Mililani,6
"Pearl City",6
"Honolulu, HI",5
honolulu,5
Kapolei,4
"Hale‘iwa",3
Honlulu,3
```

The majority of the city values are occupied by Honolulu, as expected. However, there are at least two instances (“Honolulu, HI” and ‘Honlulu) where a non case sensitive query on Honolulu would fail to generate a hit, due to the inclusion of the state abbreviation (HI) in one case and due to a misspelling (‘Honlulu’) in another case. This issue could and should be addressed in a programmatic manner similar to what was done for street type, state and zip code. A benefit of this update would be the ability to run more accurate, complete queries. A potential problem in this fix is that many of the Hawaiian cities have unusual (to mainlanders like me), punctuated names (e.g. ‘Hale’iwa’), and one would need to be careful to write code that accounts for the myriad acceptable names.

In the case of inconsistent store naming, during a query to determine the number of Starbucks coffee shops on the island, I noticed that there were a few instances where a name other than ‘Starbucks’ was used. For example, ‘Starbucks Coffee’ was used on several occasions. It would be possible to address this programmatically, by generating code to look for any occurrence of the word ‘starbuck’ in the value attribute, creating a list of these occurrences and then programmatically updating them. The benefit of doing so would be the ability to more accurately count and locate popular chain stores. A potential problem in making this change is that it is possible that the word ‘starbucks’ would show up in store names that are unrelated to the coffee chain (e.g. in the town I live, Bellevue WA, there is a Starbuck’s Towing). Thus great care would need to be made to not make the data set worse instead of better. For example, in my opinion, it would be better to have the data stay as “Café” – “Starbucks coffee” than risk inadvertently tagging a towing business as a potential place to buy a cup of coffee.

Appendix A: Data Auditing Results

These results (and others) were generated by running the data.py program on the complete honolulu.osm file. Running on the sample.osm file will generate a smaller set of problematic data.

Possible problematic street types:

{'106': set(['Pualei Cir, Apt 106']),
 'Ave': set(['Kalakaua Ave']),
 'Blvd': set(['Ala Moana Blvd']),
 'Center': set(['Enchanted Lakes Shopping Center']),
 'Dr': set(['Kipapa Dr']),
 'Hwy': set(['Kamehameha Hwy']),
 'Ike': set(['Ala Ike']),
 'Kailua,': set(['Kaelepulu Dr, Kailua,']),
 'King': set(['South King']),
 'Momi': set(['Pali Momi']),
 'Pkwy': set(['Meheula Pkwy']),
 'St': set(['Ala Pumalu St', 'Lusitania St']),
 'St.': set(['Lusitania St.']),
 'Walk': set(['Beach Walk']),
 'highway': set(['kanehameha highway']),
 'king': set(['king'])}

Possible problematic zip codes:

set(['96815-2518', '96734-9998', '96826-4427', '96817-1713', 'HI 96819', '96815-2830', '96815-2834', '96712-9998', '96825-9998'])

Possible problematic state values:

set(['Hi', 'hi', 'Hawaii', 'hawaii'])