OpenStreetMap Project - Ott

July 14, 2017

```
Udacity - Data Analyst Nanodegree
OpenStreetMap Project - Data Wrangling
Thomas Ott
```

0.1 Wrangling of OpenStreetMap Data

0.1.1 Map Area

Dayton, OH, United States

- OpenStreetMap website http://www.openstreetmap.org
- Overpass API used to download the data http://overpass-api.de/query_form.html
- Boundaries used for the Dayton, OH metropolitan area
    - minlat: 39.4821000
    - maxlon: -83.9723000
    - minlon: -84.4543000
    - maxlat: 39.9255000

I live in Washington Township, which is in the South-East corner of the Dayton area. The boundaries chosen do not conform to any official definition of the Dayton metropolitan area, but are only a general selection made by hand when downloading the data.

0.1.2 Cleaning the Data

While auditing the dataset, the following issues were found that needed to be corrected before storing the data for use.

- Street type abbreviations are used inconsistently.
    - Convert abbreviations to full name (example: change Dr to Drive)
- Unicode is used to insert special characters. Since these characters are relevant in OpenStreetMap, they shouldn't be changed in the online database.
    - For my SQL database, convert Unicode to string format for readability.
        - change \u2013 to a hyphen (Dayton-Wright Brothers Airport)
        - change \u2019 to an apostrophe (Marion's Piazza)
        - change \u201c & \u201d to double quotes (Toys"R"Us)

        * change \xe4 to an ‘a’ (small a in Autokraft)
        * change \xae to a blank space (u’TIGER/Line\xae 2008 Place Shapefiles (url. . .)’)

- Correct zip code issues.
  - Not enough digits (4404 should be 45404). Correct when creating SQL tables for format checking.
  - Outside selected area (45242, 85201). Check and correct in SQL database.

Street Name Issues Street type abbreviations (example: Dr for Drive) are inconsistently used in the dataset. These abbreviations will be changed programmatically to the long name while storing the data in SQL format.

```
def update_name(name, mapping):
    for word in name.split(" "):
        if word in mapping:
            name = name.replace(word, mapping[word])
    return name
```

Tag Values in Unicode Format A number of tag values are in Unicode instead of string when special characters are used. For my SQL database, these values are converted into a string value. If OpenStreetMap will be updated from this database, these values should remain Unicode to preserve the special characters.

```
Given value: u'Alex\u2013Bell Road East'

def unicode2string(value):
    value = unicode(value).encode('utf8')
    value = value.replace("\xe2\x80\x93", '-')
    value = value.replace("\xe2\x80\x99", "'")
    value = value.replace("\xe2\x80\x9c", '"')
    value = value.replace("\xe2\x80\x9d", '"')
    value = value.replace("\xc2\xae", '')
    value = value.replace("\xc3\xa4", 'a')
    return value

New value: 'Alex-Bell Road East'
```

Note that the encode statement changes the unicode value \u2013 to string value \xe2\x80\x93, which is then replaced with the hyphen.

Zip Code Issues The entry that only has 4 digits will be corrected programmatically during the store to SQL in case database check constraints are used. By looking at the street name, I can tell that the ‘4404’ entry should be ‘45404’.

```
if temp_dict['key'] == 'addr:postcode' and temp_dict['value'] == '4404':
    temp_dict['value'] = '45404'
```

The zip codes that are outside the area can be corrected through the SQL interface. This will be done after the data is stored.

0.1.3 Storing the Data

To store the data, each primary element is read using xml.etree.cElementTree iterparse. The sub-elements (tags and nodes) are then individually read, cleaned as discussed above, and stored in comma delimited files that will be used to populate the SQL tables. Each entry in the comma delimited files is validated using Cerberus to ensure formatting will match the intended SQL tables. Each table is then created in SQLite and the data imported from the comma delimited files into the tables.

File Sizes

```
DaytonMetro2.osm ......... 110.0MB
OpenStreetMap_Dayton.db .. 61.3MB
nodes.csv ............... 41.9MB
nodes_tags.csv .......... 0.8MB
ways.csv ................ 3.3MB
ways_nodes.csv .......... 14.0MB
ways_tags.csv ........... 8.5MB
```

0.1.4 Exploring the Data - SQL Queries

Number of Nodes and Ways

```
Nodes:
sqlite> Select count(*) from nodes;
496739

Ways:
sqlite> Select count(*) from ways;
55717
```

How Many Nodes Have Zero Tags?

```
sqlite> select num_tags, count(*) as count
   ...> from (select n.id, count(nt.id) as num_tags
   ...> from nodes as n left join nodes_tags as nt
   ...> on n.id = nt.id
   ...> group by n.id) as subq
   ...> group by num_tags
   ...> having num_tags = 0;
0|487577
```

487,577 of the 496,739 Nodes (98.2%) do not have tags.

How Many Ways Have Zero Tags?

```
sqlite> select num_tags, count(*) as count
   ...> from (select w.id, count(wt.id) as num_tags
   ...> from ways as w left join ways_tags as wt
   ...> on w.id = wt.id
   ...> group by w.id) as subq
   ...> group by num_tags
   ...> having num_tags = 0;
0|352
```

352 of the 55,717 Ways (0.6%) do not have tags.

Do All Way Elements Have At Least 2 Nodes?

```
sqlite> select num_tags, count(*) as count
   ...> from (select w.id, count(wn.id) as num_tags
   ...> from ways as w left join ways_nodes as wn
   ...> on w.id = wn.id
   ...> group by w.id) as subq
   ...> group by num_tags
   ...> having num_tags < 5;
2|6032
3|4011
4|2956
```

There are no instances of Way elements with zero or one node. All Way elements have at least 2 Nodes.

Zip codes Outside the Selected Area The zip codes that are outside the area will be identified, researched and corrected as needed through the SQL interface.
The entries are identified by ID.

```
sqlite> select distinct(id) from nodes_tags
   ...> where key='postcode' and value in ('45242', '85201');
4707869491
4707871762
4709984729
4711323784
4711332045
```

Looking at the first id as an example, we get the following information.

```
sqlite> select * from nodes_tags where id = 4707869491;
4707869491|city|Dayton|addr
4707869491|housenumber|6314|addr
4707869491|postcode|45242|addr
4707869491|state|OH|addr
4707869491|street|Ashley Meadows Cir|addr
4707869491|description|Welcome to best rated carpet cleaner in Dayton OH. We give
4707869491|email|xtracarecleaners@aol.com|regular
4707869491|name|X-tra Care Carpet Cleaning Inc|regular
4707869491|phone|(937) 236-6262|regular
4707869491|website|http://xtracarecleaners.com/|regular
```

The website shows the zip code as 45424 instead of 45242. The entry can be updated as follows.

```
sqlite> update nodes_tags set value='45424' where id=4707869491 and key='postcode';

sqlite> select * from nodes_tags where id = 4707869491 and key='postcode';
4707869491|postcode|45424|addr
```

Each of the above IDs can be checked and corrected as needed.

Count of Entries by City

```
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;
Dayton|97
Germantown|58
Brookville|49
Washington Twp.|27
Centerville|18
Beavercreek|15
Fairborn|15
Waynesville|8
Riverside|7
Bellbrook|6
West Carrollton|6
Kettering|5
Miamisburg|5
Springboro|5
Englewood|4
Huber Heights|4
Mesa|3
Moraine|3
Vandalia|3
Oakwood|2
Trotwood|2
Xenia|2
Alpha|1
Fairborn, OH|1
Franklin|1
Middletown|1
Springbro|1
Tipp City|1
Washington Township|1
Wright-Patterson AFB|1
germantown|1
```

Like the Street Names, the City entries also need some clean-up. For instance, Washington Township is mostly abbreviated as Washington Twp. and is only spelled out once. Germantown has an entry that isn't capitalized. Since there aren't too many, these changes can be done through the SQL interface similar to how the Zip Code changes were done.

Number of Unique Contributors

```
sqlite> select count(distinct(unionIDs.uid))
   ...> from (select uid from nodes union all select uid from ways) as unionIDs;
476
```

There are 476 unique users that have contributed to the Dayton area dataset.

Different Amenities in the Region *Top 10 Amenities on Nodes*

```
sqlite> select value, count(*) as num from nodes_tags
   ...> where key='amenity'
   ...> group by value
   ...> order by num desc
   ...> limit 10;
place_of_worship|465
school|175
fuel|120
grave_yard|99
parking|60
fast_food|36
restaurant|31
post_office|28
fire_station|10
library|10
```

Top 10 Amenities on Ways

```
sqlite> select value, count(*) as num from ways_tags
   ...> where key='amenity'
   ...> group by value
   ...> order by num desc
   ...> limit 10;
parking|514
school|154
fuel|139
place_of_worship|107
fast_food|95
restaurant|91
bank|52
bench|52
shelter|19
library|18
```

Count of Amentity Types

```
sqlite> select count(distinct(value)) from nodes_tags
   ...> where key='amenity';
42

sqlite> select count(distinct(value)) from ways_tags
   ...> where key='amenity';
48
```

Node elements have 42 different types of amenities listed while way elements have 48 different types. It would be interesting to see how many of the node amenities are repeated in the way elements. I'll save that for another time.

0.1.5 Additional Ideas to Improve the Dataset

Since the boundaries were selected arbitrarily, there are cities with only a few entries where I didn't get the whole city. More accurately defining the boundaries to get all Dayton suburbs would better represent the named dataset. To do this, it would take some time to understand where city boundaries lie, and then it would be tedious to associate nodes with those boundaries. Ultimately, a new way element could be created that represents the boundaries. Once created, it would be much easier to accurately select the region.

As mentioned above, the City names need some additional clean-up for consistency. Other key fields should also be checked for inconsistencies including State, Phone and Amenities. The code used to change street type abbreviations can be easily modified to complete these changes, resulting in a cleaner dataset.

The dataset can be further improved by adding detail to the way elements that lack tags, and there appears to be an opportunity to improve the accuracy of the available data by helping to validate the Tiger entries. The Tiger entries have a field indicating if the data has been reviewed, and most are marked 'No'. Completing these two items would take some research and knowledge of the area, and it would help to become familiar with the mapping techniques discussed on the OpenStreetMap website. However, the effort would add significant value for those using the dataset.

0.1.6 Conclusion

There is ample room for improvement in the OpenStreetMap dataset for the Dayton area. By providing consistency in entered fields, searches on the dataset will be more effective. As reflected in the zip code change made through the SQL interface, there are entries with incorrect information that can be easily updated. There are also a lot of Tiger entries that haven't been reviewed along with Way elements that have no tags. The validation and addition of missing details will increase the value to users in the Dayton area.