OpenStreetMap Data Case Study

Ashley dePreaux

Map Area

Boston, MA, United States

- http://www.openstreetmap.org/relation/2315704
- https://mapzen.com/data/metro-extracts/metro/boston_massachusetts/

I recently moved to New England after talking about it for a few years, and Boston is the closest metropolitan city to me. I would love to learn more about it, and also love the opportunity to help contribute to the OpenStreetMap initiative.

Problems Encountered in the Map

After initially downloading a small sample size of the Boston area and running it against a provisional data.py file, I noticed five main problems with the data, which I will discuss in the following order:

- Overabbreviated street names *(St instead of Street, etc.)*
- Inconsistent postal codes *(Some postal codes followed a conventional 5-digit format while others included the extra 4 digits)*
- “Incorrect” postal codes (Boston zip codes all start with 021 or 022; however, several zip codes did not conform). Several zip codes were from the surrounding areas such as Cambridge, Somersville, Chelsea, etc.
- There are several secondary tags with the 'k' value 'address' and its corresponding address stored as an entire address instead of broken up into addr:housenumber, addr:street, addr:street_type, etc *("5 Second Rd, Boston MA 02201")*

Overabbreviated Street Names

I used an audit street names function learned from the course to look over the street names and see what kind of abbreviated names I was working with. Next, I used another function that I learned from the course to iterate over each street name and see if it matched a mapping, as shown below.

```
def update_street_name(name, mapping):
    m = street_type_re.search(name)
    if m:
        last_word = name.split(" ")[-1]
        if last_word in mapping:
            return name.replace(last_word, mapping[last_word])
        else:
            return name
```

This updated all substrings in problematic address strings, such that: *“123 Maple Ave” and "55 Acorn St"* becomes: *“123 Maple Avenue” and "55 Acorn Street"*

Postal Codes

Luckily, I did not have to account for postal codes with alpha characters, only numeric characters. I wrote a function to strip the last four digits plus any dashes after the first five characters of the zip code. Finally, I checked the zip code against a list of valid Boston zip codes that I obtained. I removed postal codes from surrounding suburbs that are not actually a part of Boston.

```
def update_zip_code(zip_code, zip_list):
    new_zip = zip_code.split("-")[0]
    if new_zip not in zip_list:
        return None
    else:
        return new_zip
```

Full Address Fields

Finally, I wanted to take the tags with full addresses and split them into separate entities for each field of the address. I wrote a function that takes the tag and splits it into five different tags--but if any fields return invalid or None I pass over the tag, as shown below.

```
def fix_full_address(addr):
    addr_dict = {}
    addr_list = addr.split(" ")
    for i in range(0, len(addr_list)):
        addr_list[i] = addr_list[i].replace(',', '')
    addr_dict['zip'] = update_zip_code(addr_list[-1], boston_zips)
    if addr_list[-2]  == "MA":
        addr_dict['state'] = "MA"
    else:
        return None
    if addr_list[-3] == "Boston":
        addr_dict['city'] = "Boston"
    else:
        return None
    try:
        addr_int = int(addr_list[0])
        addr_dict['housenumber'] = addr_list[0]
        street_name = ""
        for x in range(1, len(addr_list) - 3):
            street_name += addr_list[x] + " "
        addr_dict['street'] = street_name.strip()
    except:
        street_name = ""
        for x in range(0, len(addr_list) - 3):
            street_name += addr_list[x] + " "
        addr_dict['street'] = street_name.strip()
    return addr_dict
```

Data Overview and Additional Ideas

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

File sizes

```
boston_massachusetts.osm ......... 420 MB
boston.db .......... 269 MB
nodes.csv ............. 152 MB
nodes_tags.csv ........ 32 MB
ways.csv .............. 20 MB
ways_tags.csv ......... 41 MB
ways_nodes.cv ......... 51 MB
```

Number of nodes

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

1956405

Number of ways

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

311972

Number of unique users

```
sqlite> SELECT COUNT(DISTINCT(e.uid))
FROM (SELECT uid FROM node UNION ALL SELECT uid FROM way) e;
```

1464

Top 10 contributing users

```
sqlite> SELECT e.user, COUNT(*) as num
FROM (SELECT user FROM node UNION ALL SELECT user FROM way) e
GROUP BY e.user
ORDER BY num DESC
LIMIT 10;
```

```
crschmidt                      1194532
```

```
jremillard-massgis  428556
wambag                        110068
OceanVortex               90723
morganwahl                67007
ryebread                      65932
MassGIS Import        58420
ingalls_imports       32452
Ahlzen                        28193
mapper999                     14619
```

Number of users appearing only once (having 1 post)

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

394

Additional Ideas

Additional Data Exploration

Top 10 appearing amenities

```
sqlite> SELECT value, COUNT(*) as num
FROM nodes_tags
WHERE key='amenity'
GROUP BY value
ORDER BY num DESC
LIMIT 10;
```

```
bench                         2216
restaurant            1444
school                    801
bicycle_parking     654
cafe                          560
place_of_worship    560
fast_food                 426
library                   403
bicycle_rental      276
post_box                  252
```

I find it rather amusing that the top amenity in Boston is a bench. Very random!

Biggest religion

```
sqlite> SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
    JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='place_of_worship') i
    ON nodes_tags.id=i.id
WHERE nodes_tags.key='religion'
GROUP BY nodes_tags.value
ORDER BY num DESC
LIMIT 1;
```

```
christian 496
```

Most popular cuisines

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

```
pizza		88
american	78
mexican 	68
italian 	66
chinese 	64
indian		44
thai		38
asian		32
japanese	30
regional	24
```

I am a little surprised that pizza is the most popular. I was actually expecting Irish to be common, but in the dataset there aren't any cuisines labeled with irish anywhere in the name. However, there are 14 'international' restaurants and the Irish places could easily be grouped into this category.

Top Attractions

```
SELECT nodes_tags.id, nodes_tags.value, COUNT(*) as num
FROM nodes_tags
    JOIN (SELECT DISTINCT(id) FROM nodes_tags) i
    ON nodes_tags.id=i.id
WHERE nodes_tags.key='tourism'
GROUP BY nodes_tags.value
ORDER BY num DESC
LIMIT 5;
```

```
artwork         132
hotel               106
museum          70
viewpoint       60
attraction  56
```

Conclusion

Ideas for Improvement

```
While the data provided is certainly useful, I think it is less useful for a
tourist who is interested in planning out their day. For example, if I wanted to
hit the most popular pizza restaurant for lunch and then take a tour of the most
popular museum afterwards, I would want there to be a rating system. I think adding
a rating would
be fairly simple. It could even be as simple as 1 - 5 and if there isn't a rating
present, the value can be null.

If ratings were added to certain nodes or ways such as restaurants, amenities,
landmarks, parks, playgrounds, etc., someone could filter their data based on
whether or not the data has a rating. If they don't care about the rating, they can
leave this part out. If this wasn't possible, perhaps an API could be used to
import third party reviews from places like Yelp or Google.

The downside of adding ratings to the dataset is that it may clog up the data with
unnecessary nulls. Fortunately, nulls can easily be filtered out!
```

Final Thoughts

After this review of the data it seems to me that the data has a lot of information, especially looking at its size even after cleaning out the cities and zip codes that didn't tie to Boston. Even though I haven't taken the trip down to visit Boston, I plan to go within the next couple of months. I will be looking for some street names that I recognize, and I will continue to explore the dataset to practice my SQL queries as well as to learn the OSM structure.