OpenStreetMap Data Case Study

Section 0 - Map Area

Austin, TX, United States

- http://www.openstreetmap.org/relation/113314

I have recently relocated to Austin, TX. This project will help me get to know the city better– by way of database queries!– and create an opportunity for me to support the local mapping scene.

Section 1 - Auditing the OSM Data

Cursory Audit

- I downloaded a pre-processed "metro extract" from Map Zen (1.42GB, unzipped) at the link below, and used a script (create_sample.py) to create a smaller file (9.3MB) containing a systematic sample of the xml content.
  - (https://mapzen.com/data/metro-extracts/metro/austin_texas/)
- I began my audit by getting an overview of the tags present in the sample (count_tags.py), their parent-child structure (find_children_of_tags.py), as well as what kind of attributes are tagged using "tag" elements (kattributes_by_feature.py).
  - The tags and their parent-child structure were exactly what I expected, having read the OpenStreetMap documentation. So, that was good.
  - From examining the k-attributes of various features, I identified two types of data that would probably benefit from a closer audit– specifically, street names and phone numbers– since the formatting of those tend to vary.
  - Many nodes and ways were seemingly created by GPS units, as indicated by tag keys prefixed with "tiger:. . . " or "gnis:. . . ". This could be problematic as it requires users to consider special cases when making queries related to geotagged data.
    - E.g., k = 'tiger:county'

Auditing Street Types

- I used a script (audit_street_types.py) to extract the street type from the end of the street addresses present in my sample file and compare it to the street names having that type. This revealed a couple of **problems**:
  - Street types are inconsistently abbreviated:
    - E.g., *"Texas Ash Cv"* versus *"Trillium Cove"*
  - Some street types are ommitted entirely:
    - E.g., *"Capri"* versus *"Capri Street"*
- I addressed the inconsistent abbreviations by:
  - creating a list of common abbreviations and the particular abbreviations present in my sample file;
  - creating a mapping from the abbreviations to the words they abbreviate– e.g., "cv" –> "Cove"; and
  - used a cleaning function to programmatically update the street types in the street names in my dataset.
    - E.g., "Applegate Dr. East" –> "Applegate Drive East"
- Addressing the omitted street names proved more difficult, as there are an abundance of addresses in the Austin area that, properly, have no street type in their address.

  - E.g., "404 Explorer, Lakeway, TX 78734".
  - One would need a **gold standard** data source to know which addresses were missing street types. While I don't have *legitimate* access to such a data source, this is discussed further in the "Suggested Improvements" section.

Auditing Phone Numbers

- I used a script (audit_phone_formats.py) to profile the phone numbers in my dataset according to their formatting. In particular, I used regular expressions to extract the formatting between the number blocks, which I then encoded as a string. From these strings, I built a dictionary of key:value pairs in which the keys were "format strings" and the values were sets of numbers in my datasets coinciding with a particular "format string." E.g.,
  - '[() -]' coincides with '(555) 555-5555'
  - '[..]' coincides with '1 555.555.5555'
  - '[() ,]' concides with '+1 (555) 555, 5555'
- As I expected, the phone numbers in my data set had a variety of different formats (and, therefore, different format strings), which I regarded as another **problem** with data consistency. Also, a single field– noteably, in the ENTIRE Austin metropolitan area– had multiple phone numbers, separated by a semi-colon.
- Employing regular expressions that captured the individual parts of the number, I built and applied the function below to create a uniform format for all of the phone numbers in my dataset.
  - E.g., '+1 (555) 555, 5555' –> (555) 555 - 5555
  - This fix ignores non-US numbers, as well as more egregious formatting issues; but, I didn't have any of those in my dataset.

```
# Reg exp for extracting country, area, prefix, line
con = r'[\D|\s]*'                   # Various connectors and spaces
country = r'(?P<country>\d{1,2})?' # Country code (optional)
area = r'\(?(?P<area>\d{3})\)?'     # Area code
prefix = r'(?P<prefix>\d{3})'       # Prefix
line = r'(?P<line>\d{4})'           # Line number
phone_digits_string = country+con+area+con+prefix+con+line
phone_digits_re = re.compile(phone_digits_string)

def update_phone_number(phone_number):
    ''' Uses regex to capture area code, prefix, and line number
    from phone_number string and reformat it into "(area) prefix - line"
    format.
    '''

    m = phone_digits_re.search(phone_number)
    if m:
        area = m.group('area')
        prefix = m.group('prefix')
        line = m.group('line')
        phone_number = '({}) {} - {}'.format(area,prefix,line)
        return phone_number
    return phone_number
```

Section 2 - The SQL Database: austin_tx_osm.db

Constructing the SQL database

- Having audited the data and developed the cleaning plan above, I cleaned and imported the data into an SQLite database using the recommended schema for the nodes, node_tags, ways, ways_nodes, and ways_tags tables.
- Additionally, I created four new tables to handle the relations tags, which posed the following challenge:
  - *Member* tags, which are children of *relations*, contain a "ref" field which refers to the id of some node or way. Since this single field acts as a foreign key to two different objects– which are represented by different tables in my schema– this field cannot have a foreign key constraint in a SQL database. (See: https://stackoverflow.com/questions/7844460/foreign-key-to-multiple-tables) This is disappointing, since references to the nodes and ways comprising a relation are the very content of that relation.
  - I found a resolution for this in the Udacity forums, where it was suggested that member tags be divided across two tables: relation_nodes and relation_ways. (See: https://discussions.udacity.com/t/foreign-key-reference-to-multiple-tables/193289) This worked perfectly.

Overview Statistics

Below are some summary statistics of the data contained in austin_tx_osm.db and, where applicable, the SQL queries used to obtain them.

File sizes:

```
austin_texas.osm ..... 1420 MB
austin_tx_osm.db ..... 810.9 MB
nodes.csv ..... 601.7 MB
node_tags.csv ..... 13.6 MB
ways.csv ..... 48.2 MB
ways_tags.csv ..... 75.4 MB
ways_nodes.csv ..... 169.9 MB
relations.csv ..... 0.148 MB
relations_tags.csv ..... 0.494 MB
relations_nodes.csv ..... 0.098 MB
relations_ways.csv ..... 0.285 MB
```

Number of nodes, ways, and relations:

```
nodes: 6400526
ways: 670811
relations: 2405
```

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

sqlite> SELECT COUNT(*) FROM ways;

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

Number of unique contributors: 1380

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

Top ten contributors and number of contributions:

```
patisilva_atxbuildings: 2741983
ccjjmartin_atxbuildings: 1300396
ccjjmartin__atxbuildings: 939996
```

```
wilsaj_atxbuildings: 358749
jseppi_atxbuildings: 300796
woodpeck_fixbot: 220495
kkt_atxbuildings: 157843
lyzidiamond_atxbuildings: 156355
richlv: 49863
johnclary_axtbuildings: 48227
```

- Note that #2 and #3 are likely the same user.

```
sqlite> SELECT b.user, COUNT(*) as num
        FROM (SELECT user FROM nodes UNION ALL
        SELECT user FROM ways UNION ALL
        SELECT user FROM relations) AS b
        GROUP BY b.user
        ORDER BY num
        DESC
        LIMIT 10;
```

Number of contributors with "atxbuildings" in their username

- A lot of contributions come from users affiliated with "atxbuildings". Let's find out how many there are and the number of contributions they've made.

```
1. patisilva_atxbuildings: 2741983
2. ccjjmartin_atxbuildings: 1300396
3. ccjjmartin__atxbuildings: 939996
4. wilsaj_atxbuildings: 358749
5. jseppi_atxbuildings: 300796
6. kkt_atxbuildings: 157843
7. lyzidiamond_atxbuildings: 156355
8. Omnific_atxbuildings: 15649
9. Jonathan Pa_atxbuildings: 9932
```

- Note that #10 from the "Top ten contributors" does not appear in this list, despite being affilitated with atxbuildings. This is no doubt due to the typo in his username (do you see it?) and my query's inflexibility.

```
sqlite> SELECT b.user, COUNT(*) as Num
        FROM (SELECT user FROM nodes UNION ALL
        SELECT user FROM ways UNION ALL
        SELECT user FROM relations) AS b
        WHERE b.user LIKE "%atxbuildings%"
        GROUP BY b.user
        ORDER BY Num
        DESC;
```

Additional Queries

Ten most common amenities:

```
restaurant: 646
waste_basket: 603
place_of_worship: 424
fast_food: 384
bench: 364
fuel: 237
```

```
school: 217
bar: 133
cafe: 123
pharmacy: 90
```

- Let's take a moment to appreciate that 603 wastebaskets in the Austin, TX, metropolitan area have been lovingly tagged by users...

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

Ten most common cuisines:

```
mexican: 61
pizza: 31
american: 29
chinese: 21
italian: 17
indian: 15
asian: 14
thai: 14
sandwich: 13
burger: 12
```

```
sqlite> SELECT a.value, COUNT(*) as num
        FROM nodes_tags AS a, nodes_tags AS b
        WHERE a.id = b.id
        AND b.key = "amenity"
        AND b.value="restaurant"
        AND a.key="cuisine"
        GROUP BY a.value
        ORDER BY num
        DESC
        LIMIT 10;
```

Section - 3 Suggestions for Improvement

Validating street addresses

- Distinguishing street names that properly possess no street type from street names wherein the type has been erroneously omitted is difficult to programmatically correct without access to a **gold standard** library of Austin addresses. The United States Postal Service provides access to such a database through the *Address Verification API*. It would be easy to write a script to query this web service to validate and clean addresses in the Austin, TX, OSM data.
  - Unforunately, the Terms of Service for *Address Verification API* explicitly forbid its use in this manner (cf https://www.usps.com/business/web-tools-apis/address-information-api.htm –a discovery that thwarted my original data cleaning plans.
  - Fortunately, there exist enterprise-facing address verification services which could be used in this manner; but they are not free to use, so...

Nine digit postal codes

- The same address verification information described above could be used to extend the postal codes found in the database from the five digit version to the nine digit version. As the results of the following query show, all 81322 of the postal codes in the database employ the (less accurate) five digit versions.

```
sqlite> SELECT LENGTH(value) as 'Postal Code Length', COUNT(*) as Count
        FROM nodes_tags
        WHERE nodes_tags.key = "addr:postcode"
        GROUP BY 'Postal Code Length'
        ORDER BY 'Postal Code Length'
        DESC;
```

Replace "tiger: ..." and "gnis: ..." keys with additional tags

- The use of GPS prefixes in k-attributes could be addressed by programmatically removing the prefix from the k-attribute and adding an additional tag with k='GPS' and v='tiger' (or some equivalent).
  - This would alleviate the need for considering special cases when making queries related to geotagged data, while preserving the information inherent in the prefixes.

Section - 4 Conclusion

- The Austin, TX, OSM data is an excellent start to mapping the features of the Austin Metropolitan Area. There is clearly a very active community contributing to this project. Further, a minimally funded non-profit could make vast extensions and improvements to the data, which bodes well for the future of this project.