OpenStreetMap Data Case Study

Author: Patrick Tuite Date: February 1, 2017

Introduction and Map Area

Philadelphia, PA United States

- OpenStreetMap - Philadelphia
- Dataset - Map Zen Metro Extracts - Philadelphia Metro Area

Having recently moved to Philadelphia this case study presented a unique way to explore my new surroundings. I have been fascinated with maps for my entire life so it was exciting to learn about and manipulate the underlying data of an open source map.

OpenStreetMap is an open source map project with much of the data contributed and maintained by unpaid volunteers.

From their about page:

> OpenStreetMap is built by a community of mappers that contribute and maintain data about roads, trails, cafés, railway stations, and much more, all over the world.

It is extremely impressive that diverse and diffuse group of people can come together to create a functioning and thorough world map. A great strength of the project is the passion and numbers of its contributors. However, this strength can also be a weakness, because any data that is touched by humans has the potential for problems, especially in a case like this where a number of different people with minimal direction are making contributions and edits.

This study focuses on identifying problems in the dataset, cleaning and preparing the data for importation into a SQL database, exploring the data, and ideas for improvement.

Identifying Problems

I began with a sample of the dataset taken using the sample_region.py script. I created the mapparser.py script to programatically iterate through the xml data to get an idea of tags and attributes. With a better understanding of the underlying data I developed the audit.py script to pick up potentially unexpected street formatting, postal codes, and phone number types.

```
def audit(osmfile):
    osm_file = open(osmfile, 'r')
    street_types = defaultdict(set)
    postcodes = []
    phone_numbers = []
    for event, elem in ET.iterparse(OSMFILE, events=('start',)):
        if elem.tag == 'way' or elem.tag == 'node':
            for tag in elem.iter('tag'):
                if is_street_name(tag.attrib['k']):
                    audit_street_type(street_types, tag.attrib['v'
                if is_zip_code(tag.attrib['k']):
                    if len(tag.attrib['v']) != 5:
                        postcodes.append(tag.attrib['v'])
                if is_phone_number(tag.attrib['k']):
                        phone_numbers.append(tag.attrib['v'])
    osm_file.close()
```

Using the results of the audit and running the data.py script to view the sample data in CSV form I discovered the five problems in the data I wanted to clean prior to inserting into the SQL database.

- Inconsistent Street Name Formatting: There was no clear consensus in the data as to whether the street type should be abbreviated or not e.g. St. or Street, Ave. or Avenue. I chose to convert all types to their longform for the sake of clarity and consistency.
- Case in Street Names: Street names are proper nouns and capitalization should reflect this. While auditing the data I discovered numerous miscapitalized entries.
- Cardinal Directions in Street Names: As was the case with street types, cardinal directions did not have a clear consensus on abbreviation e.g. N. or North I chose to convert all types to their longform for the sake of clarity and consistency.
- Phone Numbers: Phone numbers in the data were found to be in several different formats (e.g (555) 555-5555 or 555555555). I chose to standardize to the E.164 format as suggested by the OSM wiki.
- Inconsistent or Incorrect Postal Code Notation: Postal codes were in several different formats (e.g. 12345 123456789 PA 12345). I chose to standardize to the basic 5 digit format.

Cleaning and Preparing Data

With these five problems identified I developed a plan to clean and process the data into a SQL database with a defined schema using the data.py script.

Inconsistent Street Name Formatting and Cardinal Directions

Created function `update_name` to convert street type abbreviations and cardinal direction prefixes to their long form via my defined mapping by matching a regular expression. Example: "N Broad St." => "North Broad Street" "Germantown Ave" => "Germantown Avenue"

```
street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)
direction_re = re.compile(r'\b[NESW]\b\.*', re.IGNORECASE)

mapping = {
            "AVE": "Avenue",
            "Ave": "Avenue",
            ...
            "Ter": "Terrace",
            "W": "West",
            "W.": "West",
            "S": "South",
            "S.": "South",
            }

def update_name(name):
    name = capitalize_name(name)
    st = street_type_re.search(name)
    st_key = st.group()
    d = direction_re.match(name)
    if st_key in mapping.keys():
        name = re.sub(street_type_re, mapping[st_key], name)

    if d:
        name = re.sub(direction_re, mapping[d.group()], name)

    return name
```

Case in Street Names

Created function `capitalize_name` to run the `string.capwords` function on each name and adjust US or NJ to the accepted long form for their routes. Helper function for `update_name`. Example: "US 1" => "U.S. Route 1" "south broad street" => "South Broad Street"

```
state_nat_re = re.compile(r'Nj|Us\sHighway|Us')

state_national_mapping = {
                            "Us": "U.S. Route",
                            "Us Highway": "U.S. Route",
                            "Nj": "New Jersey",
                        }

def capitalize_name(name):
    name = string.capwords(name)
    m = state_nat_re.search(name)
    if m:
        name = re.sub(state_nat_re, state_national_mapping[m.group
    return name
```

Phone Numbers

Created `standardize_phone_number` to strip extraneous characters and add country code. Example: "(215) 242-0820" => "+12152420820"

```
def standardize_phone_num(phone_number):
    phone_number = phone_number.translate(None, '()- .')
    if not phone_number.startswith('+'):
        phone_number = "+1" + phone_number
    return phone_number
```

Inconsistent or Incorrect Postal Code Notation

Created `standardize_postcode` to return just the 5 digit postal code. Example: "PA 19119" => "19119"

```python
postcode_re = re.compile(r'[08|18|19]\S{4}')

def standardize_postcode(postcode):
    if len(postcode) != 5:
        m = postcode_re.search(postcode)
        if m:
            postcode = m.group()
    return postcode
```

Having created all the helper functions, created the `clean_value` function to take the keys and values in the `shape_element` function to clean the data as it is inserted into a SQL database with the data.py script.

```python
def clean_value(key, value):
    if is_phone_number(key):
        return standardize_phone_num(value)
    elif is_zip_code(key):
        return standardize_postcode(value)
    elif is_street_name(key):
        return update_name(value)
    else:
        return value
```

Exploring Data

File Sizes

```
philadelphia_pennsylvania.osm .... 658.5 MB
philadelphia.db .................. 363.3 MB
nodes.csv ........................ 246.4 MB
nodes_tags.csv ................... 14.5 MB
```

```
ways.csv ......................... 17.4 MB
ways_tags.csv .................... 49.8 MB
ways_nodes.csv ................... 85.1 MB
```

Number of Nodes

```
sqlite> SELECT COUNT(id) FROM Nodes;
```

2960075 **Number of Ways**

```
sqlite> SELECT COUNT(id) FROM Ways;
```

291555 **Number of Unique Users**

```
sqlite> SELECT COUNT(DISTINCT(e.uid))
   ...> FROM (SELECT uid FROM Nodes UNION ALL SELECT uid FROM Ways
```

1954 **Prolific Users** Top five users responsible for over 57% percent of entries.

```
sqlite> SELECT e.user, COUNT(*)
   ...> FROM (SELECT user FROM Nodes UNION ALL SELECT user FROM wa
   ...> GROUP BY e.user
   ...> ORDER BY COUNT(*) DESC
   ...> LIMIT 5;
```

```
dchiles|797931
woodpeck_fixbot|561902
NJDataUploads|295993
kylegiusti|106141
```

```
WesWeaver|104633
```

Users With 5 Entries or Fewer

```
sqlite> SELECT COUNT(*)
   ...> FROM (SELECT e.user, COUNT(*)
   ...> FROM (SELECT user FROM Nodes UNION ALL SELECT user FROM wa
   ...> GROUP BY e.user
   ...> HAVING COUNT(*) < 6);
```

791 **Top Ten Cities Represented** Data is representitive of metro Philadelphia, including Pennsylvania and New Jersey. I wanted to see the top cities.

```
sqlite> SELECT tags.value, COUNT(*) as num
   ...> FROM (SELECT * FROM nodesTags UNION ALL SELECT * FROM ways
   ...> WHERE tags.key LIKE '%city'
   ...> GROUP BY tags.value
   ...> ORDER BY num DESC
   ...> Limit 10;
```

```
Philadelphia|3626
Pemberton|2092
Trenton city|1659
Narberth|1438
Lawrence twp|1157
Upper Chichester;Marcus Hook;Chichester|967
Boothwyn|845
Voorhees|644
Marcus Hook|374
Cherry Hill|253
```

Top Amenities in Philadelphia Specifically in Philadelphia

```
sqlite> SELECT nodesTags.value, COUNT(*) as count
   ...> FROM nodesTags
   ...> JOIN (SELECT DISTINCT(id) FROM nodesTags WHERE value LIKE
   ...> ON nodesTags.id = phil.id
   ...> WHERE key = 'amenity'
   ...> GROUP BY nodesTags.value
   ...> ORDER BY count DESC
   ...> LIMIT 10;
```

```
social_facility|272
restaurant|149
marketplace|57
fire_station|49
cafe|41
library|29
bar|25
place_of_worship|25
bank|22
fast_food|21
```

I was curious about the social facilies so I checked out 10 of them. It looks like many of them are churches, which while I am sure they provide some social services would be a miscategorization of the data. According to the OSM wiki "Specific examples of a social facility are drug clinics, workshops for the physically disabled, and homeless shelters.". Also, there is a `place_of_worship` tag that would be better suited for this.

```
sqlite> SELECT nodesTags.value, COUNT(*) as count
   ...> FROM nodesTags
   ...> JOIN (SELECT DISTINCT(id) FROM nodesTags WHERE value = 'so
   ...> ON nodesTags.id = phil.id
   ...> WHERE key = 'name'
   ...> GROUP BY nodesTags.value
   ...> ORDER BY count DESC
   ...> LIMIT 10;
```

```
Philabundance|2
St. Michael's Lutheran Church|2
59th Street Baptist Church|1
Abyssinian Baptist Church|1
Advocate St. Stephens United Methodist Church|1
Allen AME Church|1
BM Oakley Memorial Temple Church|1
Baptist Women's Center|1
Berean Presbyterian Care Closet|1
Bethel Fellowship Franklin Mills|1
```

Supermarkets

```
sqlite> SELECT nodesTags.value, COUNT(*) as count
    ...> FROM nodesTags
    ...> JOIN (SELECT DISTINCT(id) FROM nodesTags WHERE value = 'su
    ...> ON nodesTags.id = market.id
    ...> WHERE key = 'name'
    ...> GROUP BY nodesTags.value
    ...> ORDER BY count DESC;
```

There appears to be some inconsistency with names. Acme is listed under a few different names.

```
Giant|14
Acme|12
Shoprite|7
```

```
Whole Foods|7
Trader Joe's|5
Acme Markets|4
Pathmark|4
ACME|3
ShopRite|3
Acme/Savon|2
Aldi|2
...
Weavers Way Co-op|1
Wegman's|1
Wegmans|1
Whole Foods Philadelphia 2|1
Yang's Market|1
cousin's|1
thefreshgrocer|1
```

Number of Wawas Philadelphia's favorite convenience store.

```
sqlite> SELECT COUNT(value) FROM nodesTags WHERE value LIKE '%Wawa
```

104

Conclusions and Ideas for Improvement

A dataset this large created by multiple users on a continual bases is certain to be messy and this data certainly has that problem. Prior to creating the database I cleaned a few areas specifically related to street names, zip codes, and phone numbers. Throughout my exploration I found numerous inconsistencies related to choice of tag, spelling, capitalization, and business name issues. After cleaning and exploring the datasets I have a few takeaways and some ideas for improvement.

- A small number of users seem to be doing a large amount of the work. Many of these "power-users" appear to be bots or large data dumps which I believe has the opportunity to introduce errors into the data. Also, there is the potential for an automated user to assert their interpretation of OSM guidelines. There is a trade-off here as in the aggregate the larger amount of useful data supplied by these automated users is likely to add enough to the functionality of the map to outweigh any negatives from the messy data. The Automated Edits code of Conduct is somewhat limited and has virtually no mention of style If the community were to agree upon and enhance guidlines for automated data entry many of the potential drawbacks could be mitigated.
- Inconsistency of tagging appears to be a major problem in the data. For example there were a large number of 'social_facilities' which in fact appear to be a 'place_of_worship' according to the OSM guidelines. Another problem was inconsistency with names of businesses. For example within my supermarket query Acme is represented in four different ways. The OSM editing feature does have predictive text when entering a name of a business which means there is a database of generally accepted names. In order to prevent this type of inconsistency from occuring I would suggest that any change made outside of a generally accepted naming convention should be reviewed by a member of the community for accuracy. I believe that in generally if changes are regularly prompted for review by more seasoned users it would help to reduce errors and inconsistency in the data and even promote the spirit of the community.