OpenStreetMap Case Study: Philadelphia, PA

This project involved auditing and cleaning of OpenStreetMap data for the Philadelphia, PA metropolitan area. Data was obtained in XML format as a Mapzen metro extract and was programmatically cleaned and converted to CSV. Importing the data into a database enabled some high-level insights to be made via SQL queries.

The square-shaped map extract includes the entirety of Philadelphia and much of the surrounding area, including parts of both Pennsylvania and New Jersey. As a student at Swarthmore College just out of Philadelphia, this project presented an opportunity to learn a little bit more about my own surroundings.

Data wrangling

Problems identified

Data was audited by examining a small fraction of the top-level tags and their corresponding secondary tags from the full dataset. By programmatically inspecting the values for certain keys within the shape_element() function in data.py, several areas of significant inconsistency were identified:

- Phone numbers
- Postal codes
- Street names

The scripts used to address these problems, which can be found in cleaning.py, are described in further detail below.

Updating phone numbers

Phone numbers reported in the data followed a wide variety of conventions:

- presence of international dialing code
- variable usage of dashes, brackets, and spaces to separate numbers
- presence of extensions ("ext.")
- presence of multiple numbers

It was decided to convert all phone numbers to the North American Numbering Plan convention of (XXX) XXX-XXXX, and to include only the first number when multiple numbers were reported so all tags with the "phone" key would have the exact same format. This was accomplished via the code below. Regular expressions were used to quickly remove extensions, extra phone numbers, and special characters. 11-digit numbers were stripped of international code and all numbers were reformatted via string concatenation.

```
import re
```

```
def update_phone(phone_num):
    # Remove multiple numbers and extensions
    stripped = re.split('[xo,;]', phone_num)[0]
    # Strip of all non-numeric characters
    stripped = re.sub('[^0-9]','', stripped)
    # Unusual cases
    if len(stripped) != 10 and len(stripped) != 11:
        if phone_num == "1+1-215-626-7668":
            return "(215) 626-7668"
        elif phone_num == "215-22x-2728":
            return "(215) 22x-2728"
        # Invalid phone number reported
        else:
            return False
    if len(stripped) == 11:
        if stripped[0] == '1':
            stripped = stripped[1:]
        else:
            return False
    return "(" + stripped[:3] + ") " + stripped[3:6] + "-" + stripped[6:]
```

Updating postal codes

Postal codes in the dataset were most often reported as five- or nine-digit (XXXXX-XXXX) ZIP codes, and were occasionally entered with the incorrect number of digits, voiding the data. Multiple postal codes are reported for many roads, which are found in the OpenStreetMap data as ways including a tag with the "highway" key. Therefore, in this case keeping track of multiple postal codes was deemed appropriate. The following script extracts all 5-digit runs and returns them separated by semicolons, as per OpenStreetMap standards.

```
import re

def update_postcode(postal_code):
    possible_codes = []
    # Immediately return suitable codes
    if len(postal_code) == 5 and len(re.sub('[^0-9]','', postal_code)) == 5:
        return postal_code
    stripped = re.split('[^0-9]', postal_code)
    for segment in stripped:
        if len(segment) == 5: # 5-digit code
            possible_codes.append(segment)
    # No 5-digit codes present
    if len(possible_codes) == 0:
        return False
    # Multiple 5-digit codes present
    elif len(possible_codes) > 1:
        return_code = ""
        for code in possible_codes:
            return_code += ";"
            return_code += code
        return return_code[1:]
    else:
        return possible_codes[0]
```

Updating street names

Lastly, street names were edited for uniformity. A variety of inconsistencies were found in street names:

- Some names were entered without capital letters
- Variable abbreviation of cardinal directions
- Variable abbreviation of street name endings (ex. "St.", "St", "Street")

The first letter of each word in the street name was capitalized and abbreviations were removed using dictionaries. By printing any unusual values, a comprehensive list of expected street types and mapping of abbreviations was compiled. The script is shown below.

```
import re

def update_streetname(street_name):
    #street_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)
    expected = ["Street", "Avenue", ..., "Walk", "Expressway", "North", "South", 
"East", "West"]
    mapping = { "St": "Street",
                "St.": "Street",
                ...
                "Ln": "Lane",
                "Hwy": "Highway"
    }
    directions = {  "N": "North",
                    "N.": "North",
                    ...
                    "W": "West",
                    "W.": "West"
    }

    # Fix capitalization
    segments = street_name.split(" ")
    fixed_name = ""
    for segment in segments:
        fixed_name += " "
        seg_cap = segment.capitalize()
        if seg_cap in directions.keys():
            seg_cap = directions[seg_cap]
        fixed_name += seg_cap
    fixed_name = fixed_name[1:]

    # Check ending
    match = re.search(r'\b\S+\.?$', fixed_name)
    if match:
        ending = match.group()
        if ending not in expected:
            if ending in mapping.keys():
                new_ending = mapping[ending]
                return fixed_name[:-len(ending)] + new_ending
    return fixed_name
```

Inspection of data

Dataset statistics

Number of nodes

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

3138924

Number of tagged nodes

```
SELECT COUNT(DISTINCT(nodes_tags.id)) as num
FROM nodes_tags, nodes
WHERE nodes_tags.id = nodes.id;
```

178476

In other words, only about 5.6% of nodes have any tags at all. Most of the untagged nodes are probably nodes used in ways, and most of the tagged nodes likely mark specific map features.

Number of ways

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

327354

Number of unique users

```
SELECT COUNT(e.uid)
FROM (SELECT uid FROM nodes UNION SELECT uid FROM ways) e;
```

2263

File sizes

```
philadelphia.osm .... 696 MB
philadelphia.db ..... 308 MB
nodes.csv .......... 262 MB
nodes_tags.csv ...... 11.6 MB
ways.csv ........... 19.5 MB
ways_tags.csv ....... 52.9 MB
```

```
ways_nodes.csv ...... 91.0 MB
```

More specific queries

Most common leisure facilities

```
SELECT value, COUNT(*) as num
FROM nodes_tags
WHERE key = 'leisure'
GROUP BY value
ORDER BY num desc
LIMIT 5;
```

```
park             338
playground       146
pitch            72
sports_centre    71
picnic_table     35
```

Most common restaurant and store chains

```
SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE key = 'cuisine') sub
ON sub.id = nodes_tags.id
WHERE nodes_tags.key = 'name'
GROUP BY nodes_tags.value
ORDER BY num desc
LIMIT 5;
```

```
Dunkin' Donuts  27
McDonald's      23
Subway          17
Starbucks       16
Burger King     11
```

```
SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE key = 'shop') sub
ON sub.id = nodes_tags.id
WHERE nodes_tags.key = 'name'
GROUP BY nodes_tags.value
ORDER BY num desc
LIMIT 5;
```

```
Wawa       52
7-Eleven   23
Giant      15
```

```
Acme        14
Staples     11
```

Wawa is indeed a Greater Philadelphia area fixture! It's surprising that there are twice as many Wawa stores in the dataset than the next most common chain, 7-Eleven.

Suggestion for improvement: Postcodes in nodes_tags

```
SELECT COUNT(*) as num
FROM nodes_tags
WHERE key = 'postcode' OR key = 'zip_left' OR key = 'zip_right';
```

3588

Proportion of tagged nodes with postal code data: 3588 / 178476 = 0.02

Only 2% of tagged nodes have postal codes entered. Postal codes are useful for grouping location-based data for analysis, and supplying more nodes with postal code data would make for an improvement in the quality of the data. It may be possible to automate postal code tagging using the latitude/longitude data contained in every node, which would also prevent any errors stemming from user entry. With regards to nodes (not ways), postal codes are most useful for marking specific map fixtures or landmarks, such as buildings or amenities. Therefore, some restraint should be exercised in automating the tagging of nodes with postal codes to ensure application only when relevant.