OPENSTREETMAP DATA CASE STUDY

Map Area

Toronto ,Canada

```
I choose this area because my parents lives there and size of its XML OSM was suitable for this
project
,neither too large nor less than 50MB in compressed form.
```

Identifying Problems in the Map

1. Scrolling through the Unix less command output to view portions of the data in their original form.
2. Analyzing the audit.py script output to view unusual street names and postal codes.
3. Analyzing the CSV files created by the process_osm.py script to view the data (in schema.md format) before and after cleaning code was applied.

Problems Encountered in the Map

Simplified versions of code cleaning the following problems are presented below.

Overabbreviated street names

Spell out all street types and directions.

- "Nonquon Rd" to "Nonquon Road"
- "Main St N" to "Main Street North"

```
nth_re = re.compile(r'\d\d?(st|nd|rd|th|)', re.IGNORECASE)
nesw_re = re.compile(r'\s(North|East|South|West)$')

mapping = {
            "St": "Street",
            "St.": "Street",
            "Ave": "Avenue",
            "Ave.": "Avenue",
            ...
            "S.": "South",
            "S": "South",
            "W.": "West",
            "W": "West"
          }

street_mapping = {
                   # same as above, minus North, East, South, West
                 }
...

else:
    original_name = name
    for key in mapping.keys():
        # Only replace when mapping key match (e.g. "St.") is found at end of name
        type_fix_name = re.sub(r'\s' + re.escape(key) + r'$', ' ' + mapping[key], original_name)
        nesw = nesw_re.search(type_fix_name)
        if nesw is not None:
            for key in street_mapping.keys():
                # Do not update correct names like St. Clair Avenue West
                dir_fix_name = re.sub(r'\s' + re.escape(key) + re.escape(nesw.group(0)), \
                                      " " + street_mapping[key] + nesw.group(0), type_fix_name)
                if dir_fix_name != type_fix_name:
                    return dir_fix_name
        if type_fix_name != original_name:
            return type_fix_name
return original_name
```

Inconsistent street names

Spell "Lines" numbered ten and under.

- "6th Line" to "Sixth Line"

```
num_line_street_re = re.compile(r'\d0?(st|nd|rd|th|)\s(Line)$', re.IGNORECASE)

num_line_mapping = {
                     "1st": "First",
                     "2nd": "Second",
                     "3rd": "Third",
                     ...
                     "9th": "Ninth",
                     "10th": "Tenth"
                   }

if num_line_street_re.match(name):
    nth = nth_re.search(name)
    name = num_line_mapping[nth.group(0)] + " Line"
    return name
```

Consistently format York-Durham Line.

- "York & Durham Line" to "York-Durham Line"
- "York/Durham Line" to "York-Durham Line"

```
elif name == "York & Durham Line" or name == "York/Durham Line":
    name = "York-Durham Line"
    return name
```

Incorrect phone number format

Convert, where necessary, to international format with spaces: "+1 ### ### ####".

- "416-555-1234" to "+1 416 555 1234"
- "4165551234" to "+1 416 555 1234"
- "1 (416) 555-1234" to "+1 416 555 1234"

```
PHONENUM = re.compile(r'\+1\s\d{3}\s\d{3}\s\d{4}')

def update_phone_num(phone_num):
    # Check for valid phone number format
    m = PHONENUM.match(phone_num)
    if m is None:
        # Convert all dashes to spaces
        if "-" in phone_num:
            phone_num = re.sub("-", " ", phone_num)
        # Remove all brackets
        if "(" in phone_num or ")" in phone_num:
            phone_num = re.sub("[()]", "", phone_num)
        # Space out 10 straight numbers
        if re.match(r'\d{10}', phone_num) is not None:
            phone_num = phone_num[:3] + " " + phone_num[3:6] + " " + phone_num[6:]
        # Space out 11 straight numbers
        elif re.match(r'\d{11}', phone_num) is not None:
            phone_num = phone_num[:1] + " " + phone_num[1:4] + " " + phone_num[4:7] \
                        + " " + phone_num[7:]
        # Add full country code
        if re.match(r'\d{3}\s\d{3}\s\d{4}', phone_num) is not None:
            phone_num = "+1 " + phone_num
        # Add + in country code
        elif re.match(r'1\s\d{3}\s\d{3}\s\d{4}', phone_num) is not None:
            phone_num = "+" + phone_num
        # Ignore tag if no area code and local number (<10 digits)
        elif sum(c.isdigit() for c in phone_num) < 10:
            return None
    return phone_num
```

Province code not used

Convert full province name to province code.

- "Ontario" to "ON"

```
# Change Ontario to ON
if province == 'Ontario':
    province = 'ON'
```

Incorrect postal code format

Convert letters to upper case and separate the character trios with a space.

- "a1b 2c3" to "A1B 2C3"
- "A1B2C3" to "A1B 2C3"

```
# See code for next header
```

Incomplete and incorrect postal codes

Discard postal codes that are not in the correct Canadian format, i.e., A1A 1A1, where A is a capital letter and 1 is an integer.

- "L4B"
- "M36 0H7"

```
POSTCODE = re.compile(r'[A-z]\d[A-z]\s?\d[A-z]\d')

m = POSTCODE.match(post_code)
    if m is not None:
        # Add space in middle if there is none
        if " " not in post_code:
            post_code = post_code[:3] + " " + post_code[3:]
        # Convert to upper case
        new['value'] = post_code.upper()
    else:
        # Keep zip code revealed in postal code audit for document deletion purposes
        if post_code[:5] == "14174":
            new['value'] = post_code
        # Ignore tag if improper postal code format
        else:
            return None
```

An American invasion

Via auditing the postal codes and subsequently cleaning them with the above code, I noticed an American ZIP code. Let's examine:

```
sqlite> SELECT * FROM Nodes
        WHERE id IN (SELECT DISTINCT(id) FROM nodesTags WHERE key = "postcode" AND value = "14174");
```

```
3443667462|43.2384384|-79.0386425|derektucker|2812604|1|30050746|2015-04-07T21:42:03Z
```

```
sqlite> SELECT * FROM nodesTags WHERE id="3443667462";
```

```
3443667462|building|house|regular
3443667462|city|Youngstown|addr
3443667462|state|NY|addr
3443667462|street|Woodland Court|addr
3443667462|postcode|14174|addr
3443667462|housenumber|451|addr
```

Data Overview

This section contains basic statistics about the Toronto OpenStreetMap dataset and the SQL queries used to gather them.

File sizes

```
toronto_canada.osm  1.14 GB
toronto.db          679 MB
nodes.csv           382.1 MB
nodes_tags.csv      84.8 MB
ways.csv            39.3 MB
ways_nodes.cv       120.7 MB
ways_tags.csv       85.6 MB
```

Number of unique users

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

1865

Number of nodes

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

4765469

Number of ways

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

694588

Top 10 contributing users

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

```
andrewpmk          3320719
MikeyCarter        480722
Kevo               436391
Victor Bielawski   159243
Bootprint          158319
Mojgan Jadidi      100749
geobase_stevens    80551
rw__               75865
Gerit Wagner       43306
brandoncote        37884
```

First contribution

```
sqlite> SELECT timestamp FROM Nodes UNION SELECT timestamp From Ways
        ORDER BY timestamp
        LIMIT 1;
```

2006-10-16T03:16:49Z

most popular cuisine

```
sqlite> SELECT nodesTags.value, COUNT(*) as num

FROM nodesTags

JOIN (SELECT DISTINCT(id) FROM nodesTags WHERE value='restaurant') i

ON nodesTags.id=i.id

WHERE nodesTags.key='cuisine'

GROUP BY nodesTags.value

ORDER BY num DESC

LIMIT 5;
```

```
chinese 141
Indian 93
italian 83
japanese 80
pizza 54
```

Dataset Improvement

Let's look back at two queries performed above to perform a new query:

Number of nodes

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

4765469

Number of nodes with wheelchair accessibility information

```
sqlite> SELECT COUNT(*) FROM nodesTags WHERE key='wheelchair';
```

3303

Percentage of nodes with wheelchair accessibility information

```
3303 / 4765469 = 0.069%
```

Approximately 0.07% of the nodes in the dataset contain wheelchair accessibility information. That seems like a strikingly low number, even with a large amount of nodes being private property (e.g. homes).

One way to improve this number is to leverage the public data provided by AccessTO, the Toronto Accessible Venues List (TAVL), and similar resources. Accessibility information for hundreds of restaurants, cafes, tourist attractions, community centers, and other public spaces could be added to the dataset. Programmatically extracting the yes/no information and adding it to the OpenStreetMap dataset would likely be most efficient. The more detailed comments in the TAVL spreadsheet could even be programmatically added under the OpenStreetMap "note" key. One difficulty would be dealing with naming inconsistencies between AccessTO/TAVL data and nodes already in the OpenStreetMap dataset, though this could be overcome with careful string handling and a human verifying inputted data.

BENEFITS:
1.By leveraging the public data accessibility to public amenities, restaurants, tourist attraction etc can be improved significantly.
2.Openstreetmap.org project can be made more reliable for other purposes.

ANTICIPATED PROBLEMS:
1.Removing naming inconsistency could pose a problem due to different naming conventions.
2.Due to rapidly changing geographic scenario maintaining the consistency across the dataset could be tedious job.

Conclusion

The Toronto OpenStreetMap dataset is a quite large and quite messy. While it is clear that the data is not 100% clean, I believe it was sufficiently cleaned for the purposes of this project. Via SQL query, I learned a few new things about my hometown. The dataset is very useful, though areas for improvement exist.