Wrangle OpenStreetMap Data with SQL

By Trevor Cook

Project Summary

This project goes through the data wrangling process by extracting a metropolitan area on OpenStreetMap, converting it into xml format, uploading the data into a database, and performing queries on the database with SQL. Before exporting the data into xml files, an audit is performed to analyze whether there are any problems or inconsistencies with the data. These problems, such as inconsistent streen names, are then updated before exporting the data into xml files.

Map area: Halifax, Nova Scotia

The map position that I chose for this project can be found at the following link:
https://www.openstreetmap.org/relation/1339656#map=10/44.7857/-63.2304
(https://www.openstreetmap.org/relation/1339656#map=10/44.7857/-63.2304)

The reason for choosing Halifax, NS as the OpenStreetMap area for my project is because I spent 4 years in this city attending Dalhousie University. Halifax is and always will be one of my favorite cities.

Map size:
halifax_canada.osm: 80.4 MB

Table of Contents

Data Audit

The first step in this project is to perform a data audit which will highlight any potential problems we may have with the dataset. I chose to audit both street types and postal codes because I noticed in the sample code that this data was not all written in the same format. For example, a contributor may have inputed 'St.' instead of Street.

The way this audit works is that it searches for street types (postal codes) using re, compares the results to street types (postal codes) that I expect to see in the data - such as Road, Street, and Avenue, and finally, creates a defaultdict for the values found which are not displayed in the expected list.

Street Names

The results of the Street Names Audit are as follows:

```
{'Blvd': set(['Topsail Blvd']),
 'Circle': set(['South Ridge Circle']),
 'Close': set(['Lanshaw Close', 'Loppie Close', 'Roxham Close']),
 'Crescent': set(['Digby Crescent', 'Plateau Crescent']),
 'Dr': set(['Novalea Dr']),
 'Grove': set(['Promise Grove']),
 'Highway': set(['Bedford Highway']),
 'Hill': set(['Ioney Hill']),
 'NS': set(['MUMFORD ROAD, HALIFAX NS']),
 'Quinpool': set(['6487 Quinpool']),
 'Rd': set(['6324 Quinpool Rd', 'Kempt Rd', 'Quinpool Rd']),
 'Rd.': set(['6371 Quinpool Rd.']),
 'Row': set(['Dresden Row']),
 'Run': set(['Beech Tree Run']),
 'St': set(['Gesner St', 'May St', 'Preston St']),
 'St,': set(['Provo Wallis St,']),
 'St.': set(['George St.']),
 'Terrace': set(['Cannon Terrace', 'Milo Terrace']),
 'Walk': set(['Lemon Walk']),
 'Way': set(['Peakview Way', 'Whispering Way']),
 'West': set(['Portland Estates Boulevard West', 'Sullivan Street West']),
 'ave': set(['millbrook ave'])}
```

Based on this audit, we can see that there are several instances where the street abbreviations were not found in the expected list of names.

The first type represents uncommon street abbreviations, such as Circle or Crescent. There is nothing wrong with these street types, they are simply more rare names that have not been included in the expected list.

The second type is shortened street names, for example Dr or St. This data will have to be cleaned up by changing their names to Drive and Street, respectively.

The data audit also shows an instance when the contributor included 'HALIFAX NS' in the street address.

Postal Codes

The results of the Postal Codes Audit are as follows:

```
{'B3L1B1': set(['B3L1B1']),
 'B3L1J6': set(['B3L1J6']),
 'B3S0E1': set(['B3S0E1']),
 'B4E3B5': set(['B4E3B5']),
 'NS': set(['NS B3J 1P2', 'NS B3L 1A6']),
 'b2v0a2': set(['b2v0a2'])}
```

The audit for postal codes is similar to that done for the street types as it highlights any potential problems with the data. As we can see from the audit, there are a few opportunities for cleaning this data. The first problem is that not all of the postal codes are written in capital letters. There are also many cases where there is whitespace missing between each 3 character block. Finally, some postal codes begin with 'NS', followed by the postal code.

After cleaning the data, each postal code will be written in the same format. They will be written in uppercase letters, and will contain a space following the first 3-character block of the postal code.

Problems Encountered

Street Names

Once the audit was performed to display any cleaning opportunities with the data, I used the following function to change the abbreviated street names to the correct ones.

In [1]:

```
def update_name(name, mapping):
    name_split = name.split()
    for key in mapping.keys():
        if key in name_split:
            name = ' '.join(name_split)
            name = name.replace(key, mapping[key])
    return name
```

Postal Codes

A similar approach was used to change the postal codes to the same format, using the following function:

In [2]:

```
def update_name(name):
    if name[0] == "b":
        name = name.upper() # Capitalize lower case postal codes
    if name[0] == "N":
        name = name.split(" ")
        first, second = name[1], name[2]
        seq = (first, second)
        name = " ".join(seq) # Remove 'NS' from beginning of string
    if "-" in name:
        name = name.replace("-", " ") # Changes '-' to a space
    else:
        if len(name) == 6:
            # If the length of the string is 6, there is no space in between each 3
            # This code splits the string in half and adds a space between them
            # Reference: http://stackoverflow.com/questions/4789601/split-a-string-
            first, second = name[:len(name)/2], name[len(name)/2:]
            seq = (first, second)
            s = " "
            name = s.join(seq)
    return name
```

Converting .csv files into Database

Once the data has been audited and cleaned, the next step is to convert the .csv files into a database before running queries on it. The following code displays the process for creating a database, creating a table within the database, reading in the .csv files, and inserting formatted data.

In [ ]:

```
sqlite_file = 'halifaxdb.db'
conn = sqlite3.connect(sqlite_file)
cur = conn.cursor()

# Create the tables
cur.execute('''
    CREATE TABLE nodes_tags(id INTEGER, key TEXT, value TEXT, type TEXT)
''')

conn.commit()

# Read in the .csv files as a dict.
with open('nodes_tags.csv', 'rb') as fin:
    dr = csv.DictReader(fin)
    to_db = [(i['id'].decode("utf-8"), i['key'].decode("utf-8"), i['value'].decode(

# Insert formatted data
cur.executemany("INSERT INTO nodes_tags(id, key, value, type) VALUES (?, ?, ?, ?);"
```

Queries

1) Number of nodes

```
SELECT count(*) FROM nodes

[(379161,)]
```

2) Number of ways

```
SELECT count(*) FROM ways

[(33193,)]
```

3) Number of unique users

```
SELECT count(DISTINCT(e.uid))
FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) as e

[(355,)]
```

4) Top 10 contributors

```
SELECT e.user, COUNT(*) as num
FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) as e
GROUP BY e.user
ORDER BY num DESC
LIMIT 10

[(u'macgregor', 190976),
 (u'timdine', 136514),
 (u'ingalls', 28210),
 (u'Steve in Halifax', 9499),
 (u'StewartSR', 8185),
 (u'kavurt', 6573),
 (u'BrianCumminger', 3851),
 (u'dgenge', 2569),
 (u'Dmajackson', 2147),
 (u'whitebuffalo', 1977)]
```

5) Top 10 types of food

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

[(u'pizza', 20),
 (u'burger', 17),
 (u'coffee_shop', 13),
 (u'chinese', 8),
 (u'greek', 6),
 (u'sandwich', 5),
 (u'sushi', 4),
 (u'japanese', 3),
 (u'mediterranean', 3),
 (u'mexican', 3)]
```

6) Top 5 sport facilities

```
SELECT ways_tags.value, COUNT(*) as count
FROM ways_tags
WHERE key='sport'
GROUP BY ways_tags.value
ORDER BY count DESC
LIMIT 5

6) Top 5 sport facilities
[(u'baseball', 88),
 (u'soccer', 57),
 (u'tennis', 39),
 (u'basketball', 36),
 (u'golf', 27)]
```

7) List of Halifax postal codes

```
SELECT tags.value, COUNT(*) as count
FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags) as tags
WHERE tags.key='postcode'
GROUP BY tags.value
ORDER BY count
LIMIT 5

[(u'B2W 1L1', 1),
 (u'B2W 1L7', 1),
 (u'B2W 2T7', 1),
 (u'B2W 2T8', 1),
 (u'B2W 2W6', 1),]
```

8) List of cities in Halifax

```
SELECT tags.value, COUNT(*) as count
FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags) tags
WHERE tags.key='city'
GROUP BY tags.value
ORDER BY count DESC
LIMIT 5

[(u'Halifax', 7315),
 (u'Dartmouth', 4290),
 (u'Cole Harbour', 1718),
 (u'Eastern Passage', 1159),
 (u'Westphal', 720),]
```

9) Number of contributions made in 2016

```
SELECT COUNT(*) as count
FROM (SELECT timestamp FROM nodes UNION ALL SELECT timestamp FROM ways) as i
WHERE i.timestamp LIKE '%2016%'

[(9330,)]
```

Conclusion and Suggestions for Improvement

By bringing the Halifax OpenStreetMap data into a database and performing queries on it, we can pull out a lot of interesting information about the dataset. However, by looking at some of the results, there is a lot of room for improvement. One way that would improve this data is to remove any further inconsistencies, as was done during the audit for street name abbreviations and postal codes. This would save the data analyst a lot of time since much of their energy goes towards wrangling the data.

OpenStreetMap could implement an audit each time a user contributes to their data, in order to ensure that similar data is written in a consistent format. For example, if a postal code in Halifax was inputed as b3h2j1, OpenStreetMap would convert this data into the standard format B3H 2J1.

Alternatively, rather than having OpenStreetMap audit all the data that enters their system, they could encourage users to edit other people's contributions. If it were set up in a similar was as how Wikipedia double checks the credibility of each contribution made on their site, the data on OpenStreetMap would more likely be cleaner data and therefore would require less wrangling.

The benefits of improving these inconsistencies with the data is that it would clean up and ensure that everything is written in the same format. On the downside, OpenStreetMap may not want to change the input generated by its users. There is also a chance that this type of audit does not change every piece of data to the correct format, causing additional errors in the data.

Although there are still a few errors that could be fixed with the dataset, I think that the queries I performed have still given me a good idea of the Halifax OpenStreetMap layout.

In [ ]: