Wrangle OpenStreetMap Using Python and SQL

August 23, 2017

0.1 Sirron Melville

1 Wrangle OpenStreetMap Using Python and SQL

1.1 ## OSM Map Area that was used:

1.1.1 Boston, Massachusetts, United States

- https://www.openstreetmap.org/export#map=11/42.3126/-70.9978

Boston is Massachusetts' capital and largest city. Founded in 1630, it's one of the oldest cities in the U.S. This beautiful city has so much to offer and there are alot of tourist sites that people can visit. In this project, the chosen dataset will be audited and cleaned and the quality of the data will be assessed for validity, accuracy, completeness, consistency and uniformity. Hopefully my analysis will contribute to its improvement on openstreetmap.org.

1.2 Problems Encountered in the Map

Several problems in the dataset were discovered after downloading and running it through a temporary data.py file. Some of the problems were:

- There were inconsistent postal codes like '02118-0239' and 'MA 02118'.
- There was some inconsistency with the street types, there were abbreviations and several versions of the same street types. Eg.('Ave', 'Ave.', 'Ct', 'Pkwy', 'Sq', 'ST', 'St', 'St.', etc).

1.2.1 Abbreviated Street Names

The abbreviated and inconsistent street types will be made uniform in the auditing and cleaning phase of the project. The following street types will be made consistent using the code in the audit.py file: 'Ave, Ave.' will be changed to 'Avenue'. 'Ct' will be changed to 'Court'. 'Dr' will be changed to 'Drive'. 'Hwy' will be changed to 'Highway'. 'Pkwy' will be changed to 'Parkway'. 'Rd', 'Rd.', 'rd.' will be changed to 'Road'. 'Sq', will be changed to 'Square'. 'Street.', 'street', 'ST', 'St', 'St.', 'St,', 'st' will be changed to 'Street'.

Steps taken in audit.py are:

- Created a list of the expected street types that don't need to be cleaned.

- The function "audit_street_type" collects and stores the last words in the "street_name" strings if they aren't in the expected list. They are stored in the "street_types" dictionary in order to see the inconsistent and abbreviated street types.
- The "is_street_name" function finds tags that specify street names(k="addr:street").
- The "audit" function returns a dictionary called "street_types" as mentioned above.
- The "update_name" uses a mapping dictionary to update a name.

```
In [10]: def update_name(name, mapping):
             """Use mapping dictioanry to update a name."""
             m = street_type_re.search(name)
             if m not in expected:
                 if m.group() in mapping.keys():
                     name = re.sub(m.group(), mapping[m.group()], name)

             return name
```

The above function updates the abbreviated address strings. Eg. "Boston St" will be changed to "Boston Street".

1.2.2 Postal Code Validation

- Postal codes in begin with "021, 022 and 024". The postal code formats vary in that some a five digits, some are nine digits and some contain the state abbreviations.
- In order to get a consistent format, the characters before and after the main 5 digit postal code were dropped. Eg. "0239" was dropped from "02118-0239" and the state abbreviation "MA" was dropped from "MA 02118". Below is the function from audit_postcode.py that was used.

```
In [11]: def update_postcode(postcode):
             """Creates a uniform format of 5 digit postcodes; Returns updated post
             if re.findall(r'^\d{5}$', postcode): # 5 digit post code eg. 02118
                 valid_postcode = postcode
                 return valid_postcode
             elif re.findall(r'(^\d{5})-\d{4}$', postcode): # 9 digit postal code e
                 valid_postcode = re.findall(r'(^\d{5})-\d{4}$', postcode)[0]
                 return valid_postcode
             elif re.findall(r'MA\s*\d{5}', postcode): # postal code with state abb
                 valid_postcode =re.findall(r'\d{5}', postcode)[0]
                 return valid_postcode
             else:
                 return None
```

1.3 Prepare cleaned files for importation into an SQL Database

The data.py file contains functions that prepare the data to be inserted into an SQL database.

- The elements in the OpenStreetMap XML file are parsed and transformed from a document format to a tabular format. This makes it easier to write to .csv files which can easily be imported to an SQL database as tables.

- The "shape_element" function is used to clean and shape each node or way XML element to the respective python dictionaries. The function is passed each individual parent element from the ".osm" file when it is called in the "process_map" function which iteratively processes each XML element and writes it to the csv file.

1.4 Overview of the Data

This section contains the creation of the SQL database and the results of the queries that were run on the dataset.

1.4.1 The size of each file

```
In [63]: # Size of files in MB
         import os
         print('The boston_massachusetts.osm file is {} MB'.format(os.path.getsize
         print('The project.db file is {} MB'.format(os.path.getsize('osm.db')/1.0e
         print('The nodes.csv file is {} MB'.format(os.path.getsize('nodes.csv')/1.
         print('The nodes_tags.csv file is {} MB'.format(os.path.getsize('nodes.csv
         print('The ways.csv file is {} MB'.format(os.path.getsize('ways.csv')/1.0e
         print('The ways_tags.csv is {} MB'.format(os.path.getsize('ways_tags.csv')
         print('The ways_nodes.csv is {} MB'.format(os.path.getsize('ways_nodes.csv
```

```
The boston_massachusetts.osm file is 438.257725 MB
The project.db file is 278.654976 MB
The nodes.csv file is 158.815619 MB
The nodes_tags.csv file is 158.815619 MB
The ways.csv file is 20.717069 MB
The ways_tags.csv is 22.308563 MB
The ways_nodes.csv is 53.162027 MB
```

1.4.2 Creation of SQL Database and Tables

```
In [42]: import sqlite3
         import csv
         from pprint import pprint

         sqlite_file = "osm.db"
         conn = sqlite3.connect(sqlite_file)
         cur = conn.cursor()

         cur.execute('DROP TABLE IF EXISTS nodes')
         conn.commit()

         cur.execute('''
             Create Table nodes(id INTEGER, lat REAL, lon REAL, user TEXT, uid INTE
         ''')

         conn.commit()
```

```
In [43]: with open('nodes.csv','r', encoding = 'utf-8') as f:
             dict_reader = csv.DictReader(f)
             to_db = [(i['id'], i['lat'], i['lon'], i['user'], i['uid'], i['version
             cur.executemany("INSERT INTO nodes(id, lat, lon, user, uid, version, c

         conn.commit()
```

1.4.3 Number of Nodes

```
In [44]: cur.execute("SELECT COUNT(*) FROM nodes;")
         print(cur.fetchall())
```

```
[(1950582,)]
```

1.4.4 Number of Ways

```
In [45]: cur.execute("SELECT COUNT(*) FROM ways;")
         print(cur.fetchall())
```

```
[(310765,)]
```

1.4.5 Number of Unique Users

```
In [46]: cur.execute("SELECT COUNT(DISTINCT(e.uid)) FROM (SELECT uid FROM nodes UNI
         print(cur.fetchall())
```

```
[(2123,)]
```

1.4.6 Top 10 Contributing Users

```
In [50]: cur.execute("SELECT e.user, COUNT(*) as num FROM (SELECT user FROM nodes U
         pprint.pprint(cur.fetchall())
```

```
[('crschmidt', 1197110),
 ('jremillard-massgis', 428817),
 ('wambag', 105600),
 ('OceanVortex', 91080),
 ('morganwahl', 67067),
 ('ryebread', 65977),
 ('MassGIS Import', 59164),
 ('ingalls_imports', 32453),
 ('Ahlzen', 28371),
 ('mapper999', 14709)]
```

1.4.7 Count From Highest to Lowest of Various Categories Related to Tourism

```
In [52]: cur.execute ("SELECT tags.value, COUNT(*) as count FROM (SELECT * FROM noc
                       SELECT * FROM ways_tags) tags \
                       WHERE tags.key LIKE '%tourism'\
                       GROUP BY tags.value \
                       ORDER BY count DESC;")
         pprint.pprint(cur.fetchall())
```

```
[('hotel', 102),
 ('museum', 60),
 ('artwork', 58),
 ('attraction', 38),
 ('viewpoint', 33),
 ('information', 27),
 ('picnic_site', 26),
 ('guest_house', 9),
 ('hostel', 5),
 ('motel', 3),
 ('aquarium', 2),
 ('chalet', 2),
 ('apartment', 1),
 ('gallery', 1),
 ('theme_park', 1),
 ('zoo', 1)]
```

1.5 Additional Data Exploration

1.5.1 Common Amenities

```
In [54]: cur.execute("SELECT value, COUNT(*) as num FROM nodes_tags WHERE key='amer
         pprint.pprint(cur.fetchall())
```

```
[('bench', 1078),
 ('restaurant', 699),
 ('school', 493),
 ('bicycle_parking', 324),
 ('cafe', 279),
 ('place_of_worship', 277),
 ('library', 272),
 ('fast_food', 205),
 ('bicycle_rental', 138),
 ('post_box', 124)]
```

1.5.2 Biggest Religion

```
In [55]: cur.execute("SELECT nodes_tags.value, COUNT(*) as num FROM nodes_tags JOIN
                      FROM nodes_tags WHERE value = 'place_of_worship') i ON nodes_t
```

```
                    GROUP BY nodes_tags.value ORDER BY num DESC LIMIT 1;")

        pprint.pprint(cur.fetchall())
```

```
[('christian', 246)]
```

1.5.3 Number of Restaurants in each City

```
In [56]: cur.execute("SELECT nodes_tags.value, COUNT(*) as num FROM nodes_tags JOIN
                    FROM nodes_tags WHERE value = 'restaurant') i ON nodes_tags.id
                    GROUP BY nodes_tags.value ORDER BY num DESC;")

         pprint.pprint(cur.fetchall())
```

```
[('Boston', 52),
 ('Cambridge', 51),
 ('Somerville', 19),
 ('Brookline', 10),
 ('Allston', 5),
 ('East Boston', 5),
 ('Brookline, MA', 4),
 ('Chelsea', 4),
 ('Arlington', 3),
 ('Medford', 3),
 ('Watertown', 3),
 ('Arlington. MA', 2),
 ('Brighton', 2),
 ('Charlestown', 2),
 ('Chestnut Hill', 2),
 ('Jamaica Plain', 2),
 ('Malden', 2),
 ('2067 Massachusetts Avenue', 1),
 ('Arlington, MA', 1),
 ('Boston, MA', 1),
 ('Cambridge, MA', 1),
 ('Cambridge, Massachusetts', 1),
 ('Everett', 1),
 ('Quincy', 1),
 ('Watertown, MA', 1),
 ('boston', 1),
 ('somerville', 1),
 ('winthrop', 1)]
```

1.5.4 Popular Cuisines

```
In [59]: cur.execute("SELECT nodes_tags.value, COUNT(*) as num FROM nodes_tags JOIN
                    FROM nodes_tags WHERE value = 'restaurant') i ON nodes_tags.id
```

```
                    GROUP BY nodes_tags.value ORDER BY num DESC LIMIT 20;")

        pprint.pprint(cur.fetchall())
```

```
[('pizza', 43),
 ('american', 39),
 ('italian', 32),
 ('mexican', 32),
 ('chinese', 31),
 ('indian', 22),
 ('thai', 19),
 ('asian', 15),
 ('japanese', 14),
 ('regional', 12),
 ('sandwich', 11),
 ('seafood', 10),
 ('sushi', 9),
 ('ice_cream', 8),
 ('international', 7),
 ('french', 6),
 ('ramen', 5),
 ('burger', 4),
 ('diner', 4),
 ('fish', 4)]
```

1.5.5 Most Popular Fast Food Chains

```
In [60]: cur.execute ("SELECT nodes_tags.value, COUNT(*) as num FROM nodes_tags JOI
                      FROM nodes_tags WHERE value='fast_food') i ON nodes_tags.id=i
                      GROUP BY nodes_tags.value ORDER BY num DESC LIMIT 5;")

         pprint.pprint(cur.fetchall())
```

```
[("Dunkin' Donuts", 13),
 ('Subway', 12),
 ("McDonald's", 9),
 ('Burger King', 8),
 ("Wendy's", 5)]
```

1.5.6 Most Popular Cafe Chains

```
In [61]: cur.execute ("SELECT nodes_tags.value, COUNT(*) as num FROM nodes_tags JOI
                      FROM nodes_tags WHERE value='cafe') i ON nodes_tags.id=i.id W
                      GROUP BY nodes_tags.value ORDER BY num DESC LIMIT 5;")

         pprint.pprint(cur.fetchall())
```

```
[('Starbucks', 48),
 ("Dunkin' Donuts", 43),
 ('Au Bon Pain', 7),
 ('Dunkin Donuts', 6),
 ("Peet's Coffee", 5)]
```

1.5.7 Most Popular Banks

```
In [62]: cur.execute ("SELECT nodes_tags.value, COUNT(*) as num FROM nodes_tags JOI
                       FROM nodes_tags WHERE value='bank') i ON nodes_tags.id=i.id W
                       GROUP BY nodes_tags.value ORDER BY num DESC LIMIT 5;")

         pprint.pprint(cur.fetchall())
```

```
[('Bank of America', 16),
 ('Citizens Bank', 10),
 ('Santander', 8),
 ('Eastern Bank', 5),
 ('TD Bank', 5)]
```

1.6 Conclusion

In this report, I investigated the OSM data for the city of Boston, Massachusetts. The raw dataset was filled with inconsistencies and posed several challenges which emphasize the importance of data wrangling in order to conduct meaningful analysis. The dataset is only ready for investigation after it is audited, cleaned and tested for validity, completeness and accuracy. The data is not consistent because of the the nature of OpenStreetMap which is volunteered geographic information input by human beings around the world. The focus of this project was to address the issue of inconsistency with the street names and postal codes. After the data was cleaned, it was converted from XML format to CSV format and imported to a SQL database where SQL queries where used to answer some interesting questions.

Below are additional ideas for improving the data and the various benefits and problems associated with them.

1.6.1 Additional Ideas for Improving the Data

There was missing data for the street names and postal codes. A standardized method of data entry should be implemented and enforced. Data validation is very important and Quality Assurance practices and tools should be utilized to ensure the completeness and accuracy of the data.

More people should be made aware of OSM and user engagement could be improved by creating a badge system like Kahn Academy or a ranking system like Kaggle. A concerted effort should be made to get local businesses, government institutions, hotel and restaurants to contribute to OSM so that the data is more accurate. Eventually this could lead to OSM having information on various tours through the city, information on available transportation systems and user reviews of establishments.

1.6.2 Benefits and Anticipated Problems in Implementing the Improvement of the Data

1.6.3 Benefits

- By standardizing data entry, raising awareness and incorporating the improvements, more and more people may start to use it if they know that the data is accurate and informational. Locals may use it to search for amenities and tourists may use it to look for various attractions and popular restaurants. The badge or ranking system could encourage the top ranked individuals to serve as moderators.

1.6.4 Anticipated Problems

- People are creatures of habit and it would be difficult to get them to use OSM if they are familiar with and trust another map data source like Google Maps.
- There may be an issue with the conversion of postal codes to latitude and longitude format due to the necessary data not being readily available.