Data Wrangling Open Street Map Data

By Geeta Dandamudi

Map Area : Boston, Massachusetts

I chose working on the Boston city Map data from Open Street Map Data site. I liked my recent visit to the place, I would like the opportunity to find interesting findings about the city and make possible contribution to its improvement on openstreetmap.org.

Link to the Open Street Map site: https://www.openstreetmap.org/export#map=11/42.3124/-70.8776

Link to the dataset: https://mapzen.com/data/metro-extracts/metro/boston_massachusetts/

Problems encountered in the map:

After downloading the dataset and running it against a provisional data.py file, I noticed the following main problems with the dataset, as follows.

- Nonuniform and abbreviated street types ('St.', 'St', 'ST', 'Ave', 'Ave.', 'Pkwy', 'Ct', 'Sq', etc).
- Inconsistent postal codes ('MA 02118', '02118-0239')

1. Abbreviated Street Names

This part of the project is to audit and clean nonuniform and abbreviated street types. I attempted to replace street type "ST, St, St., St, st, street" with "Street", replace "Ave, Ave." with "Avenue", replace "Pkwy" with "ParkWay", replace "Rd, Rd., rd." with "Road", replace "Ct" with "Court", replace "Dr" with "Drive", replace "Hwy" with "Highway", and replace "Sq" with "Sqaure". The following are the steps taken, in audit.py, python script to audit the data:

- Create a list of expected street types that do not need to be cleaned.
- The function "audit_street_type" collects the last words in the "street_name" strings, if they are not within the expected list, then stores them in the dictionary "street_types". This allows to see the nonuniform and abbreviated street types being used and gives a better sense of the data as a whole.

- The "is_street_name" function looks for tags that specify street names (k="addr:street").
- The "audit" function returns a dictionary that match the above function conditions.
- The update_name function takes an old name to mapping dictionary, and update to a new one.

```
def update_name(name, mapping):
    """takes an old name to mapping dictionary, and update to a new one"""
    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
```

This updates substring in abbreviated address strings, like: "Boston St" becomes: "Boston Street".

2. Validate Postal codes

Boston area postal codes begin with "021, 024 and 022". The formats of the post codes vary, some are five-digits, some are nine-digits, and some start from state characters.

Validating postal codes is a bit complicated. However, I decided to drop leading and trailing characters before and after the main 5-digit post code. This dropped the state characters "MA" from "MA 02118", and dropped "0239" from "02118-0239", using the following function in audit_postcode.py, USING 3 Regualr expresiions to check the post code conditions.

```
def update_postcode(postcode):
   """Clean postcode to a uniform format of 5 digit; Return updated postcode"""
   if re.findall(r'^\d{5}$', postcode): # 5 digits 02118
      valid_postcode = postcode
      return valid_postcode
   elif re.findall(r'(^\d{5})-\d{4}$', postcode): # 9 digits 02118-0239
     valid_postcode = re.findall(r'(^\d{5})-\d{4}$', postcode)[0]
     return valid_postcode
   elif re.findall(r'MA\s*\d{5}', postcode): # with state code MA 02118
      valid_postcode =re.findall(r'\d{5}', postcode)[0]
      return valid_postcode
   else:
  return None
```

Prepare for Database - SQL

The next step is to prepare the data to be inserted into a SQL database in data.py script.

To do so I will parse the elements in the OSM XML file, transforming them from document format to tabular format, thus making it possible to write to .csv files. These csv files can then easily be imported to a SQL database as tables.

The "shape_element" function is used to transform each element in the correct format, the function is passed each individual parent element from the ".osm" file, one by one (when it is called in the "process_map" function).

Create SQL Database and Tables

```
import sqlite3
import csv
from pprint import pprint
sqlite_file = "project.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 INTEGER, version TEXT,
changeset INTEGER, timestamp TEXT)
''')
conn.commit()
with open('nodes.csv','r', encoding = 'utf-8') as fin:
    dr = csv.DictReader(fin)
    to_db = [(i['id'], i['lat'], i['lon'], i['user'], i['uid'], i['version'], i['changeset'], i['timestamp']) for i in dr]
    cur.executemany("INSERT INTO nodes(id, lat, lon, user, uid, version, changeset, timestamp)
VALUES (?,?,?,?,?,?,?,?);", to_db)
conn.commit()
```

Overview of the Data

Size of each file

The boston_massachusetts.osm file is 434.233159 MB
The project.db file is 135.95136 MB
The nodes.csv file is 46.200929 MB
The nodes_tags.csv file is 46.200929 MB
The ways.csv file is 22.793719 MB
The ways_tags.csv is 21.643234 MB
The ways_nodes.csv is 13.488694 MB

Number of Nodes

```
cur.execute("SELECT COUNT(*) FROM nodes;")
print(cur.fetchall())
```

Output

```
[(1048574,)]
```

Number of ways

```
cur.execute("SELECT COUNT(*) FROM ways;")
print(cur.fetchall())
```

Output

```
[(618026,)]
```

Number of unique Users

```
cur.execute("SELECT COUNT(DISTINCT(e.uid)) FROM (SELECT uid FROM nodes UNION ALL SELE
CT uid FROM ways) e;")
print(cur.fetchall())
```

Output

```
[(922,)]
```

Further Data Exploration

Count Tourism Related Categories Descending

```
import pprint
cur.execute ("SELECT tags.value, COUNT(*) as count FROM (SELECT * FROM nodes_tags UNION
ALL \
        SELECT * FROM ways_tags) tags \
        WHERE tags.key LIKE '%tourism'\
        GROUP BY tags.value \
        ORDER BY count DESC;")
pprint.pprint(cur.fetchall())
```

Output:

```
[('hotel', 87),('museum', 54),('artwork', 48),('attraction', 33),('viewpoint', 30),('picnic_site',

26),('information', 25),('guest_house', 8),('hostel', 4),('motel', 3),('aquarium', 2),('chalet', 2),('zoo',

2),('gallery', 1),('theme_park', 1)]
```

Number of Restaurants in each city descending

```
import pprint
cur.execute("SELECT nodes_tags.value, COUNT(*) as num FROM nodes_tags JOIN (SELECT
DISTINCT(id) \
FROM nodes_tags WHERE value = 'restaurant') i ON nodes_tags.id = i.id WHERE nodes_tags.key =
'city'\
 GROUP BY nodes_tags.value ORDER BY num DESC;")
pprint.pprint(cur.fetchall())
```

Output:

```
[('Cambridge', 51), ('Boston', 44), ('Somerville', 18) ('Brookline', 10), ('East Boston', 5),('Brookline, MA',

4), ('Arlington', 3), ('Medford', 3), ('Watertown', 3), ('Allston', 2), ('Arlington. MA', 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), ('Chelsea', 1), ('Everett', 1),

('Quincy', 1), ('Watertown, MA', 1), ('boston', 1),('somerville', 1), ('winthrop', 1)]
```

Top 5 Most Popular Fast Food Chain

```
cur.execute ("SELECT nodes_tags.value, COUNT(*) as num FROM nodes_tags JOIN (SELECT
DISTINCT(id) \
 FROM nodes_tags WHERE value='fast_food') i ON nodes_tags.id=i.id WHERE nodes_tags.key='name'\
 GROUP BY nodes_tags.value ORDER BY num DESC LIMIT 5;")
pprint.pprint(cur.fetchall())
```

Output:

[("Dunkin' Donuts", 12),('Subway', 11), ("McDonald's", 8), ('Burger King', 7), ("Wendy's", 5)]

Top 5 Cafe Chain

```
cur.execute ("SELECT nodes_tags.value, COUNT(*) as num FROM nodes_tags JOIN (SELECT
DISTINCT(id) \
 FROM nodes_tags WHERE value='cafe') i ON nodes_tags.id=i.id WHERE nodes_tags.key='name' \
 GROUP BY nodes_tags.value ORDER BY num DESC LIMIT 5;")
pprint.pprint(cur.fetchall())
```

Output:

[("Dunkin' Donuts", 41),('Starbucks', 41),('Au Bon Pain', 7),("Peet's Coffee", 5), ('Starbucks Coffee', 5)]

Conclusion

The analysis helped in learning about the extent of errors the dataset is prone to due to manually generated information, and the need to clear off the inconsistencies prior to working on the dataset. The number of people supporting the use of this Map data is relatively small which brings down its scope for improvement to smaller portions but promoting the OSM data and running better parsers to address the entire data on the site can help improve its performance greatly. Despite the inconsistencies, and lack of information, the site still has great content to carry out data analysis to an extent.

Anticipated problems

Since the challenges of unidentified postal codes is existing, it is tedious to figure a way out for this on a

practical note as Open street Map is not a funded project, it ahrdly has employees working on it, or any reliable source to ensure it has consistent data. Also, with so many inconsistencies and hardly any popularity, no businesses would encourage its use, nor any normal end user would prefer it over already popular map services available.

Bibiliography

Udacity Forum

https://www.w3schools.com/xml/xml_parser.asp

https://stackoverflow.com/questions/10040444/iteratively-parse-a-large-xml-file-without-using-the-dom-approach

https://databricks.com/blog/2017/02/23/working-complex-data-formats-structured-streaming-apache-spark-2-1.html

http://myweb.sabanciuniv.edu/rdehkharghani/files/2016/02/The-Morgan-Kaufmann-Series-in-Data-Management-Systems-Jiawei-Han-Micheline-Kamber-Jian-Pei-Data-Mining.-Concepts-and-Techniques-3rd-Edition-Morgan-Kaufmann-2011.pdf

https://sqlite.org/cli.html