Open_Street_Map

February 26, 2017

1 P3 Data Wrangling - Open Street Map Case Study

I have decided to use SQL for this project.

2 Chosen Map Area

I chose to analyze the Open Street Map data for a portion of Oakland, CA, USA where I spend most of my time. I downloaded a section of the San Fracisco Bay Area map using the latitude and longitudes for all of Oakland excluding East Oakland. The datafile is 58 Megabytes total.

https://www.openstreetmap.org/relation/2833530

3 Problems Encountered in the Map

The osm datafile I originially downloaded was too small and missing the 'way' elements, so I downloaded the osm datafile using the Overpass API instead of MapZen and recieved a more complete dataset.

A few errors I observed that should be taken care of during the data auditing process:

- Some of the node attributes were missing, so I had to replace the missing data with a None value.
- Some street names are abbreviated, so I added code to expand them.
- I checked that integer fields only include integers and float fields only include float values.

4 Audit the Dataset

4.0.1 Parse the OSM Datafile to View and Count the XML Tags

I parse the osm file to view the element tags in the file. The first time I did this check my file only had "Node" element tags, so I redownloaded the file. After the second download, the file included both the Node and Way element tags.

4.0.2 Helper Functions for Data Fixing

I put together the functions I'll use to check that expected integer values are integers and replace the value with a 'None" value if they are not an integer value. I also check that expected float values are float values and replace the value with a "None" value if they are not a float value.

I also put together functions to convert abbreviated street names to the full name prior to writing the element to the csv file.

5 Convert Data from XML to CSV

5.0.3 Helper Functions for Filetype Conversion

I combine a set of Helper Functions to extract the elements from the file, validate the element against the list of Problem Characters, implement the integer and float value checks, and then put the data that passes the validation tests into a python dictionary.

5.0.4 Extract Data to Python Dictionary and Write to CSV Files

I implement my Helper Functions above to pull out the node and ways elements from the osm datafile and separate them into the appropriate csv file.

6 Convert csv files to a SQL Database

Using SQLite, I use the following commands in the command line to import the csv files I created into tables within a new osm.db database.

```
cd ~/Downloads/P3_Data_Wrangling
sqlite3 osm.db
.mode csv
.import nodes.csv nodes
.import nodes_tags.csv nodes_tags
.import ways.csv ways
.import ways_nodes.csv ways_nodes
.import ways_tags.csv ways_tags
```

6.0.5 Test Query

I open the database from Jupyter Notebook and run a simple query to make sure everything is running correctly. It passes the test, so I can start querying my data. The first time I ran the test I had values like "uid", but these values no longer appeared after I audited the dataset.

```
In [2]: #Test Query
        import sqlite3
        import pandas as pd

        # Fetch records from either osm.db
        db = sqlite3.connect("osm.db")

        c = db.cursor()
        QUERY = '''
        SELECT uid
        FROM nodes
        GROUP BY uid
        LIMIT 5
```

```
        '''

        c.execute(QUERY)
        rows = c.fetchall()

        #print query results
        df = pd.DataFrame(rows,columns=['uid'])
        print df

        db.close()
```

```
       uid
0        0
1   103253
2  1069886
3   106995
4    10786
```

7 Data Exploration

I ran the following queries on my data:

7.0.6 First Query: Nodes Keys and Ways Keys

I group the keys together from the node tags table and list them in descending order. Then I do the same for the ways tags. It looks like 'highway' has the most for the nodes tags and 'building' has the most for the ways tags, so I'll take a closer look at those in the next query.

```
In [5]: # Fetch records from osm.db
        db = sqlite3.connect("osm.db")

        c = db.cursor()
        QUERY = '''
        SELECT nodes_tags.key, COUNT(*) as num
        FROM nodes_tags
        GROUP BY key
        ORDER BY num DESC
        LIMIT 10
        '''

        c.execute(QUERY)
        rows = c.fetchall()

        #print query results
        df = pd.DataFrame(rows,columns=['Key', 'Sum'])
        print df
```

```
        db.close()
```

```
                Key   Sum
0           highway  2699
1              name  1706
2          crossing  1501
3           amenity  1114
4              shop   500
5       housenumber   440
6            street   435
7  traffic_calming   401
8              city   302
9          postcode   288
```

In [7]:
```
        # Fetch records from osm.db
        db = sqlite3.connect("osm.db")

        c = db.cursor()
        QUERY = '''
        SELECT ways_tags.key, COUNT(*) as num
        FROM ways_tags
        GROUP BY key
        ORDER BY num DESC
        LIMIT 10
        '''

        c.execute(QUERY)
        rows = c.fetchall()

        #print query results
        df = pd.DataFrame(rows,columns=['Key', 'Sum'])
        print df

        db.close()
```

```
            Key    Sum
0      building  33512
1       highway   5872
2          name   4592
3  housenumber   3407
4        street   3407
5          city   3326
6          cfcc   3308
7      postcode   3226
8        county   3175
9     name_base   3067
```

7.0.7 Second Query: Values for Highway Keys

I view the values and the count of the child nodes where the 'key' is equal to 'highway.'

```
In [8]: # Fetch records from either osm.db
        db = sqlite3.connect("osm.db")

        c = db.cursor()
        QUERY = '''
        SELECT nodes_tags.value, COUNT(*) as num
        FROM nodes_tags
        WHERE key = "highway"
        GROUP BY value
        ORDER BY num DESC
        '''

        c.execute(QUERY)
        rows = c.fetchall()

        #print query results
        df = pd.DataFrame(rows,columns=['Value', 'Sum'])
        print df

        db.close()
```

```
                Value   Sum
0            crossing  1725
1    traffic_signals   516
2                stop   244
3            bus_stop    97
4     turning_circle    48
5  motorway_junction    43
6            give_way    22
7            elevator     2
8      stop;crossing     1
9       turning_loop     1
```

7.0.8 Second Query: Values for Buildings Keys

I view the values and the count of the child ways where the 'key' is equal to 'building.'

```
In [10]: # Fetch records from either osm.db
         db = sqlite3.connect("osm.db")

         c = db.cursor()
         QUERY = '''
         SELECT ways_tags.value, COUNT(*) as num
         FROM ways_tags
```

```
        WHERE key = 'building'
        GROUP BY value
        ORDER BY num DESC
        LIMIT 10
        '''

        c.execute(QUERY)
        rows = c.fetchall()

        #print query results
        df = pd.DataFrame(rows,columns=['Value', 'Sum'])
        print df

        db.close()
```

```
         Value    Sum
0          yes  31284
1        house    864
2  residential    589
3       garage    291
4   apartments    163
5   commercial     83
6       retail     44
7       church     32
8       school     28
9         roof     17
```

7.0.9 Third Query: Finding the Locations of Schools

Next I'll write a query to output the lat and lon values for the 'building' keys that are listed as 'school.' There turns out to be significantly more node latitude and longitude values for schools (377 total) than the number of ways for schools listed above (28 total). I'll attribute this to each ways element consisting of multiple nodes elements. Adding the relational elements into my database could help determine this.

```
In [14]: # Fetch records from either osm.db
         db = sqlite3.connect("osm.db")

         c = db.cursor()
         QUERY = '''
         SELECT ways_tags.value, nodes.lat, nodes.lon
         FROM ways_tags, ways_nodes, nodes
         WHERE ways_tags.key = "building" AND ways_tags.value = "school" AND ways_t
         GROUP BY nodes.lon
         LIMIT 5
         '''
```

```
        c.execute(QUERY)
        rows = c.fetchall()

        #print query results
        df = pd.DataFrame(rows,columns=['Value', 'lat', 'lon'])
        print df

        db.close()
```

```
    Value         lat           lon
0  school  37.8252644  -122.2325659
1  school   37.825365  -122.2326276
2  school  37.8253481  -122.2326772
3  school  37.8251331  -122.2329615
4  school  37.8252347  -122.2330125
```

7.0.10 Fourth Query: Finding the Locations of Buddhist Temples

Next I'll write a query to output the lat and lon values for the 'religion' keys that are listed as 'buddhist.' There should be only one node element with this designation, and after running the query I can see the only buddhist temple in the area is at (37.8020261, -122.2683623).

```
In [13]: db = sqlite3.connect("osm.db")

        c = db.cursor()
        QUERY = '''
        SELECT nodes_tags.value, nodes.lat, nodes.lon
        FROM nodes, nodes_tags
        WHERE nodes_tags.key = "religion" AND nodes_tags.value = "buddhist" AND no
        '''

        c.execute(QUERY)
        rows = c.fetchall()

        #print query results
        df = pd.DataFrame(rows,columns=['Value', 'lat', 'lon'])
        print df

        db.close()
```

```
      Value         lat           lon
0  buddhist  37.8020261  -122.2683623
```

8 Data Overview

8.1 Users

8.1.1 The Total Number of Unique Users:

It appears that there are only 356 unique users contributing to this area of Oakland. That's a lot of data for 356 users!

```
In [15]: # Fetch records from osm.db
         db = sqlite3.connect("osm.db")

         c = db.cursor()
         QUERY = '''
         SELECT Count(*)
         FROM (SELECT uid FROM nodes
         UNION
         SELECT uid FROM ways) as subq
         '''

         c.execute(QUERY)
         rows = c.fetchall()

         #print query results
         df = pd.DataFrame(rows,columns=['No. Users'])
         print df

         db.close()

   No. Users
0        356
```

8.1.2 The Top 50 User Contributors

Taking a look at the list of top contributors, it appears that the vast majority of data is coming from about 20 contributors.

```
In [17]: # Top 50 Contributors
         db = sqlite3.connect("osm.db")

         c = db.cursor()
         QUERY = '''
         SELECT uid, COUNT(*) as num
         FROM nodes
         GROUP BY uid
         ORDER BY num DESC
         LIMIT 10
         '''
```

```
        c.execute(QUERY)
        rows = c.fetchall()

        #print query results
        df = pd.DataFrame(rows,columns=['uid', 'Sum'])

        print df

        db.close()
```

```
       uid     Sum
0    94578  112314
1  2226712   35856
2  1249504   16916
3   941449   16519
4   381909   11478
5   933797    9252
6   706170    9018
7  2219338    8266
8    14293    6900
9    22946    4938
```

8.1.3 Number of Nodes

There are 265,073 elements that are nodes tags (i.e. point data).

```
In [18]: # Fetch records from osm.db
         db = sqlite3.connect("osm.db")

         c = db.cursor()
         QUERY = '''
         SELECT COUNT(*)
         FROM (SELECT DISTINCT id FROM nodes) as subq
         '''

         c.execute(QUERY)
         rows = c.fetchall()

         #print query results
         df = pd.DataFrame(rows,columns=['No. Nodes'])
         print df

         db.close()
```

```
   No. Nodes
0     265073
```

8.1.4 Number of Ways

There are 41,543 elements that are ways tags (i.e. line and area data).

```
In [19]: # Fetch records from osm.db
         db = sqlite3.connect("osm.db")

         c = db.cursor()
         QUERY = '''
         SELECT COUNT(*)
         FROM (SELECT DISTINCT id FROM ways) as subq
         '''

         c.execute(QUERY)
         rows = c.fetchall()

         #print query results
         df = pd.DataFrame(rows, columns=['No. Ways'])
         print df

         db.close()
```

```
   No. Ways
0     41543
```

9 Additional Ideas

In order to improve the validation of the data I could add the relational elements into the database and try to determine how many of the nodes are related to the ways. This could help identify potential duplicate values similar to how I encountered potential duplicate values when querying the database for Oakland schools. The problem with this is that it will still be difficult to identify potential duplicates within my dataset if the latitude and longitude values are not the same (maybe I could use a spatial tolerance).

I could also continue to clean the data. For example, some values within the "exit_to" and "source" keys contained multiple names separated by semi-colons (shown in code below). I could separate this data into a different table that counts the number of exits.

```
In [20]: # Fetch records from either osm.db
         db = sqlite3.connect("osm.db")

         c = db.cursor()
         QUERY = '''
         SELECT nodes_tags.value
         FROM nodes_tags
         WHERE nodes_tags.key = "exit_to"
         LIMIT 10
         '''
```

```
        c.execute(QUERY)
        rows = c.fetchall()

        #print query results
        df = pd.DataFrame(rows,columns=['Value'])
        print df

        db.close()
```

```
                                Value
0  West Grand Avenue; Maritime Street
1                 I 880 South; Oakland
2              11th Street;14th Street
3             CA 13 South; Ashby Avenue
4         7th Street;West Grand Avenue
5                   I-980 West;Oakland
6                      Harrison Avenue
7                          51st Street
8         7th Street;West Grand Avenue
9                        Market Street
```

10 Conclusions

After downloading the Open Street Maps data for a majority of the Oakland area, it seems like Oakland has a pretty comprehensive amount of data (58MB of data total). The data required the replacment of the non-integer and non-float values for the 'id', 'uid', 'lat', and 'lon' fields with 'None' values. The data also required expanding the street abbreviations to fullnames.

I was able to query the cleaned set of data to locate schools in the Oakland area. The vast majority of the data consists of contributions from only 20 unique users. There are significantly more node elements than ways elements, which makes sense since ways can consist of multiple nodes.

To further assess the data I can incorporate the relational elements into my dataset and continue the data auditing and validation process.