Project 3 - OpenStreetMap Data Wrangling

Map Area

Boston, MA, United States

https://mapzen.com/data/metro-extracts/metro/boston_massachusetts/ (https://mapzen.com/data/metro-extracts/metro/boston_massachusetts/)

Boston is the city I grew up in and the place I live. It is conveniently located in an English speaking country so I can avoid different alphabets and strange characters in place names. I have a pitifully poor mental picture of Boston in my mind and am generally terrible with geography around here so I am excited to learn a thing or two about the Boston street map.

Auditing and Cleaning

Deciding Where To Start

Seeing as this is a giant database with hundreds of thousands of entries with potentially hundreds of different types of information entered by volunteering humans in their spare time I imagine there will be far more data to clean than can reasonably be plowed through in this project. So before we begin auditing the data, we will begin by defining what aspects of the data we find important to audit.

Modifying some original code from the lessons, we will print the keys for both nodes and ways and count each of them. (explore_data.py)

Having explored the keys that are out there, we will work to audit and clean the following keys for both nodes and ways:

- addr:state
- addr:city
- addr:postcode
- phone

We will then proceed to explore the amenity and user keys.

Auditing and Cleaning

We will audit our four keys with four separate files audit_state.py, audit_city.py, audit_postcode.py and audit_phone.py. Each one of these will have an update method which we will write as we audit. The data.py file will import these audit files and use them to clean the data as it translates it from xml into csv.

Problems Encountered in the Map

- Several different ways of inputing the state name
- Some city names include the state name or are not properly capitalized
- Some of the postcodes are in zip+4 format or include the state name
- There are many different formats for phone numbers
- Some phone numbers include letters
- Sometimes multiple phone numbers are included in the value

Auditing addr:state

Having audited a sample of the boston_massachusetts.osm file we found the several different entries for addr:state. These are all in reference to Massachusetts. We know in fact that since we are dealing with the Boston Area there should be no entry with a state that is not Massachusetts. Happily, we have not found any such abberations. However, since we are not sure that there are still no mistaken entries from a different we leave open the possibility to do further cleaning in SQL if we should find anything from a different state. Thus our solution is that any state names that included 'MA' or 'Massachusetts' regardless of case were standardized to 'MA'.

Auditing addr:city

To clean this data, we want to capitalize the first letter of each word and keep the other letters lower case. We want to remove any references to the state and deal with any unique cases. In our audit we discovered two unique cases: '2067 Massachusetts Avenue', which is in Cambridge, and ', Arlington, MA'.

Auditing addr:postcode

To clean this data, we'll eliminate the zip+4 format and get rid of the dashed line and extra four digits. We'll eliminaite all references to the state. Finally we'll make sure the code is exactly five digits long, otherwise we get rid of it.

Auditing phone

The most effective way to clean these phone numbers is to extract the smallest representation of phone numbers in each value using this pattern: (3 digits)(any non-numerical characters)(3 digits)(any non-numerical characters)(4 digits) and then specifying that the first digit cannot be 1. Then we remove everything but the numbers and organize it into the desired format.

Overview of the Data

Following are overview statistics of the dataset. The SQL queries made to explore the data are reproduced.

File Sizes

```
414M    ./boston_massachusetts.osm
244M    ./boston.db
150M    ./nodes.csv
 17M    ./nodes_tags.csv
 15M    ./ways.csv
 39M    ./ways_nodes.csv
 16M    ./ways_tags.csv
```

Number of Unique Users

```
SELECT COUNT(DISTINCT(uid)) FROM (SELECT uid FROM nodes UNION SELECT uid fro
m ways);
```

```
1178
```

Number of Nodes

```
SELECT COUNT(*) FROM nodes;
```

```
1933537
```

Number of Ways

```
SELECT COUNT(*) FROM ways;
```

```
239001
```

15 Most Common Amenities

```
SELECT value, COUNT(*)
FROM (SELECT key, value FROM nodes_tags UNION ALL SELECT key, value FROM way
s_tags)
WHERE key='amenity'
GROUP BY value ORDER BY COUNT(*) DESC
LIMIT 15;
```

```
bench|1065
parking|736
school|629
restaurant|622
place_of_worship|412
library|317
bicycle_parking|270
cafe|258
fast_food|195
bicycle_rental|140
fire_station|110
post_box|109
bank|98
waste_basket|94
fuel|90
```

Users who have contributed more than 2500 nodes and ways

```
SELECT user, num FROM (SELECT user, uid, COUNT(*) AS num
FROM (SELECT user, uid FROM nodes UNION ALL SELECT user, uid FROM ways)
GROUP BY uid) WHERE num>2500 ORDER BY num DESC;
```

```
crschmidt|1204291
jremillard-massgis|392843
OceanVortex|90808
wambag|78550
morganwahl|69836
MassGIS Import|63361
ryebread|56951
ingalls_imports|29065
Ahlzen|26818
mapper999|12782
dloutzen|7561
JasonWoof|6761
fiveisalive|6375
cspanring|5940
Jim Kogler|5636
synack|5334
n1dom|4726
massDOT|4722
YamaOfParadise|4084
Alan Bragg|3867
JessAk71|3336
Utible|3110
Alexey Lukin|2741
nkhall|2523
```

Number of users who have contributed fewer than 10 nodes and ways

```
SELECT COUNT(*) FROM (SELECT user, uid, COUNT(*) AS num
FROM (SELECT user, uid FROM nodes UNION ALL SELECT user, uid FROM ways)
GROUP BY uid) WHERE num<10;
```

```
735 (more than half!)
```

Top 15 most common fast food restaurants

```sql
SELECT value, COUNT(*) FROM

/***tags from nodes and ways where amenity='fast_food'***/

/***tags from NODES where amenity='fast_food'***/
(SELECT * FROM ((SELECT id FROM ways_tags WHERE key='amenity' AND value='fas
t_food')
AS fast_food_ids INNER JOIN ways_tags
ON fast_food_ids.id=ways_tags.id)
/*********************************************/

UNION ALL

/***tags from WAYS where amenity='fast_food'***/
SELECT * FROM ((SELECT id FROM nodes_tags WHERE key='amenity' AND value='fas
t_food')
AS fast_food_ids INNER JOIN nodes_tags
ON fast_food_ids.id=nodes_tags.id))
/*********************************************/

/*******************************************************/

WHERE key='name'
GROUP BY value ORDER BY COUNT(*) DESC LIMIT 15;
```

```
Dunkin' Donuts|13
McDonald's|11
Subway|10
Burger King|6
Wendy's|4
Anna's Taqueria|3
Chipotle|3
Sbarro|3
Boloco|2
Clover Food Lab|2
Domino's|2
Dunkin Donuts|2
Five Guys|2
Jimmy John's|2
Papa Gino's|2
```

Top 15 most common banks

```
SELECT value, COUNT(*) FROM

/***tags from nodes and ways where amenity='bank'***/

(SELECT * FROM ((SELECT id FROM ways_tags WHERE key='amenity' AND value='ban
k')
AS bank_ids INNER JOIN ways_tags
ON bank_ids.id=ways_tags.id)
UNION ALL
SELECT * FROM ((SELECT id FROM nodes_tags WHERE key='amenity' AND value='ban
k')
AS bank_ids INNER JOIN nodes_tags
ON bank_ids.id=nodes_tags.id))

/************************************************/

WHERE key='name'
GROUP BY value ORDER BY COUNT(*) DESC LIMIT 15;
```

```
Bank of America|15
Citizens Bank|12
Santander|7
Eastern Bank|6
TD Bank|6
Cambridge Savings Bank|4
Sovereign Bank|4
Brookline Bank|3
Cambridge Trust Company|3
Citibank|3
East Cambridge Savings Bank|3
East Boston Savings Bank|2
Leader Bank|2
People's United Bank|2
Rockland Trust|2
```

Additional Ideas

There is a mountain of unclean data in this database and the overwhelming bulk of it occurs as a mistake at the moment when it is being entered by a human being. Perhaps for some of the more common key types there could be a template for the input value which demands some agreed upon format. This would be very useful for phone numbers which are inserted in a million different ways. It might also be beneficial to add a feature which asks you to double check your entry before submitting it. This might eliminate some of the more egregious typos. The benefit of these two ideas would be that they would limit the worst of the human entry errors. An obvious drawback is that this would make entering daata programatically much slower and more tedious, since on each iteration it would have to confirm that the entry had been double checked.

Conclusion

The Boston Area data is very much incomplete. There is a lot to clean. It would be absurd to suggest that we would have over the course of this project made anything like a dent in the amount of cleaning work which needs to get done. However it is very interesting and informational to have at least given it a try and the fact that there are so many people contributing to this data set (over a thousand!) means that there is hope that this dataset will be continually refined.