Udacity_DAND_P3_OSM_and_SQL

March 30, 2017

1 Udacity Data Analyst NanoDegree Project 3

2 Wrangle OpenStreet Map Data

The purpose of this project is to use python to audit/wrangle/cleanup an OpenStreetMap database (.osm file), formatting the output to .csv for use in an SQL database, and then performing SQL queries to gain insite into some statistics of the region.

For this project I have chosen to use OpenStreetMap data from the Beaverton Oregon region. Map of extracted region: https://www.openstreetmap.org/relation/423111

At the suggestion of the OpenStreetMap webpage, I used mapzen.com to extract the OSM data file. https://mapzen.com/

2.1 Auditing and cleaning the data

I began the audit process by using a subset of the Beaverton Oregon region to speed the processing of the file. Udacity provided python code to generate the data subset. Sample file size is 25MB. File size of entire Beaverton Oregon OSM file to be officially used is 251MB. I audited the formatting of postal codes, city names and street_names. The auditing process was iterative, more issues where uncovered as I increased the data size from 25MB to the entire 251MB. In analysing SQL query results still more issues where uncovered that needed to be addressed in cleaning up the data.

I used the audit_osm.py to run audit.

2.1.1 Fields of interest audited were:

Postal Codes: identify codes that do not meet the standard 5 digit US postal code format, point out cleaned up postal code, and list all postal codes in region of interest.
City Names: list cities, identify any that need updated formatting.
Street Names: list street names that are unexpected, list proposed change to formatting if any.

Postal codes audit: I found that there were a few postal codes given in the format of #####-####. I wanted the postal code data to be of the standard US Postal Code format of 5 digits #####. The audit identified any Postal Codes that did not fit the standard format and listed all of the zipcode in the region. I then used https://www.google.com/ to verify that all zipcodes listed did indeed exist in the Beaverton Oregon region.
Postal Codes audit results:

```
This zip code is bad 97225-6345!
The cleaned up zip code will be 97225
This zip code is bad 97225-3010!
The cleaned up zip code will be 97225
....
The zipcodes in the Beaverton Oregon region are:
set(['97003',
     '97005',
     '97006',
     '97007',
     '97008',
     '97035',
     '97077',
     '97078',
     '97124',
     '97140',
     '97210',
     '97214',
     '97219',
     '97221',
     '97222',
     '97223',
     '97224',
     '97225',
     '97229'])
```

City names audit: The city names audit came about due to an issue I found when running SQL queries on the translated OSM data. I found that in some cases the city value contained the city and state abbreviation. I updated the audit script to catch these cases and display the corrected value. It should be noted that 'Scappoose' is well outside of the Beaverton region, but its postal code does not appear in the set above.

```
Beaverton, OR will be replaced with Beaverton
set(['Aloha',
     'Beaverton',
     'Beaverton, OR',
     'Hillsboro',
     'Lake Oswego',
     'Metzger',
     'Portland',
     'Scappoose',
     'Sherwood',
     'Tigard'])
```

Street names audit: Using the cleaning street names python code provided in the coursework, I audited the street names for street types that were unexpected, and provided the value of the street type that it would be cleaned up to.

Street names audit results:
The partial table below shows the unexpected street name, and points to its mapped cleaned value, in many cases there is no change to the street type. I could have added more entries to the 'expected' list, but due to the managable size of the unexpected entries, I manually audited for any entries requiring fixes.

```
Southwest 5th Street #150 => Southwest 5th Street #150
Southwest 165th => Southwest 165th
Southwest 156th => Southwest 156th
Southwest Malcolm GLN => Southwest Malcolm GLN
....
Northwest 185th Ave => Northwest 185th Avenue
NW 143rd Ave => NW 143rd Avenue
```

The expected street types, and mapping for cleaning are shown here:

```
expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Square", "
            "Trail", "Parkway", "Commons", "Circle", "Highway", "Loop", "Terrace",
mapping = { "St": "Street",
            "St.": "Street",
            "Rd.": "Road",
            "Ave": "Avenue",
            "Rd": "Road",
            "Dr": "Drive",
            "Hwy": "Highway",
            "GLN": "Glen"
            }
```

Using the results from the audits above, I updated the data.py code to address the flagged issues in the .csv files generated from the .OSM data. I did not make any changes to the original .OSM data itself.
Overall I found that the data as entered in the OSM database was very clean in regards to the above audits. Upon investigation I found that the regional Tri-Met organization (bus, mass transit) works heavily on OSM data in the region encompassing Beaverton, likely a reason that data appears so clean: https://prezi.com/jgj6cl1rtwm5/openstreetmap-and-the-trimets-open-source-trip-planner/

2.2 Overview of data

The data.py code generates the .csv files from the .OSM file to be loaded into SQL. I sourced my file sql_schema.txt into sqlite3 to create the tables and import the *.csv file data into their respective tables.

2.2.1 File sizes:

```
beaverton_oregon.osm  251MB
beaverton_osm.db      147MB
nodes.csv             102MB
nodes_tags.csv        1.1MB
```

```
ways.csv                8.9MB
ways_tags.csv           23MB
ways_nodes.csv          29MB
```

2.2.2 Number of nodes and ways

NODES:

```
SELECT COUNT(*) FROM nodes;
1110649
```

WAYS:

```
SELECT COUNT (*) FROM ways;
133441
```

2.2.3 Number of unique contributors

```
SELECT COUNT(DISTINCT(u.uid))
FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) as u;
291
```

2.2.4 Top 10 contributors

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

```
Peter Dobratz_pdxbuildings   672930
baradam                      173263
Darrell_pdxbuildings         84555
Grant Humphries              67885
lyzidiamond_imports          57422
Peter Dobratz                50251
Mele Sax-Barnett             40282
emem                         21733
Paul Johnson                 9677
Brett_Ham                    8327
```

2.2.5 Number of entries per postal code

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

```
97223    13720
97007    11512
97229    11210
97225    7179
97008    6931
97006    6810
97005    5482
97003    3842
97078    3821
97219    3612
97221    1997
97224    1323
97035    991
97124    88
97210    72
97140    12
97077    8
97214    1
```

2.2.6 Number of entries per city in region:

```
SELECT tags.value, COUNT(*) as num
FROM (SELECT * FROM nodes_tags UNION ALL
      SELECT * FROM ways_tags) tags
WHERE tags.key LIKE 'city'
GROUP BY tags.value
ORDER BY num DESC;
```

```
Beaverton       28433
Portland        24094
Tigard          15011
Aloha           9679
Lake Oswego     991
Hillsboro       392
Sherwood        12
Metzger         3
Scappoose       1
```

2.3 Additional queries:

2.3.1 Most popular cuisines investigation:

Below are the top 10 most popular cuisines, based on cuisine count. There were many more restaurants defined in ways_tags than in nodes_tags. I used the union of both tables to calculate the count. There is a risk of double counting in this situation, though a quick check by hand did not reveal this to be the case.

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

```
coffee_shop 50
pizza       42
mexican     40
sandwich    37
burger      36
american    18
chinese     17
thai        13
italian     12
japanese    12
```

In an effort to audit the accuracy of the SQL queries, I decided to look deeper into the pizza cuisine.

```
SELECT value, COUNT(*) as num
FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags)
WHERE key = 'cuisine' and value = 'pizza'
ORDER BY num DESC;
```

```
pizza       42
```

The query above reports that there are 42 entries for pizza cuisine as shown in the initial cuisines query. If I look into the amenity type for each pizza entry, I find a total of only 39 entries:

```
SELECT a.value, b.value, COUNT(*) as num
FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags) as a,
(SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags) as b
WHERE a.id = b.id
and a.key = 'cuisine' and a.value='pizza'
and b.key = 'amenity'
GROUP BY b.value
ORDER BY num DESC;
```

```
pizza restaurant 22
pizza fast_food  16
pizza cafe       1
```

This means that 3 pizza entries do not have a key for amenity.

I also decided to have a look at where these pizza places are located based on lat/lon coordinates.

```
SELECT lat, lon
FROM ((SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags) as a
JOIN nodes as b
ON a.id = b.id)
WHERE key = 'cuisine' and value = 'pizza';
45.4212153,-122.732459
45.4882448,-122.7994555
45.515023,-122.754901
45.5370785,-122.8785977
45.4325654,-122.8237279
45.5339612,-122.8702858
45.4877502,-122.8031091
45.5385676,-122.8698177
45.4472837,-122.7778718
45.4885396,-122.8131779
```

As you can see, only 10 of the pizza places have lat/lon coordinates defined.

The above two cases of missing data (amenity, lat/lon) is an indication that the OSM data provided is not entirely complete.

2.4 Additional ideas

To further improve this data base an audit could be performed on both amenity fields and lat/lon data for establishments.
The benefit provided by having more complete amenity information would be an improved accuracy of results. If a user where searching for pizza based on the amenity=cuisine key/value pair, they could end up missing a pizza place close to them. The same benefit holds true for adding lat/lon data for establishments. If for some reason a person wanted to know the locations of all of the pizza restaurants in their area based on lat/lon, they would be missing out on many locations. I focused on pizza in my examples, though the same idea holds true for any physical entity of interest.
The problem with auditing and providing the data I have suggested above is that it would be very time consuming to generate the accurate data. Since OSM is an open source project, there is not a lot of incentive for someone to want to spend their time and resources to make such updates. Although, as the database matures, it is likely that people will tackle these issues as a lot of more obvious issues will have been resolved.

2.5 Conclusion

In completing this project, I have seen the power using python to audit and clean large amounts of data in a quick and automated fashion. The power and speed of using sql to analyze data and provide statistics and relationships among the data was also proven. It is clear after the analysis of the data, that due to the human driven open source nature of the OSM data, any results obtained from OSM cannot be accepted as completely representative of reality.

3 Resources:

Location of extraction: https://www.openstreetmap.org/relation/423111

Used Mapzen to Extract actual data, as recommended from openstreetmap: https://mapzen.com/

Referred to OSM.Wiki for further learning on OSM formatting https://wiki.openstreetmap.org/wiki/OSM_XML

Referred to Python Docs for xml.etree learning https://docs.python.org/2/library/xml.etree.elementtree.htm

Referred to Stackoverflow.com repeatedlty for help with python and sql coding practices. https://stackoverflow.com/