data_wrangling_project.md

OpenStreet Map Data Wrangling Project

Overview of Area

Raleigh-Durham, NC OSM file download: https://mapzen.com/data/metro-extracts/metro/raleigh_north-carolina/

Living not too far from this area (Greensboro), the Raleigh-Durham area, coined locally as the Research Triangle, has seen vast amounts of growth the past couple of years. With the diversity of cities, counties, and metro areas, this region should be very interesting to investigate.

The orininal OSM file was 465 MB. Audit and conversion from XML to CSV took approximately 30 minutes.

Problems while Auditing/Converting OSM file

After importing the CSV files into a SQLite database, I explored the dataset and ran into several problematic issues:

- Depreciated second level `"k"` tags in `"nodes_tags"` and `"ways_tags"` (*"name_1", "Street_1", and "zipcode"*)
- Various formats of phone numbers (*"+1-(919)-680-6333", "919 908 1023"*)
- ZIP+4 postal code format (*"27603-1407"*)
- Inconsistent street name abbreviations (*"Crawford Ct", "Chapel Hill Rd"*)
- tiger and NHD data sources in second level `"k"` tags
- Carriage return values in database entries upon XML to CSV conversion

While all of these issues were found in my exploration of the dataset, I will be focusing on the major audits that were done for this project.

Standardizing Phone Numbers

Phone numbers in the dataset came in various formats. Some had the +1 country code while others included parentheses around the 3-digit area code. The format for spacing also varied between numbers, with some using dashes to separate the different sections while others used spaces.

In order to standardize these numbers, I used regular expressions to strip any non-digit characters out of the phone number. The standard format for this dataset would have an optional country code followed by ten digits.

For example, the number `+1-(919)-680-6333` would become `19196806333`.

Here is a snippet of the code that audits the phone number into the above format:

```
correct_re = re.compile(r'1?\d{10}$')

def check_phone(num):
    """Returns phone number with all non-digit characters removed."""
    if not correct_re.search(num):
        num = re.sub(r'\D', '', num)
        if not correct_re.search(num):
            num = "***Unknown format. No changes made.***"
            return num
    return num
```

With all of the numbers in the same format, querying and sorting phone numbers from the database became much easier.

Trimming Postal Codes

The most popular format for postal codes in this dataset was the ZIP+4 format ("*27603-1407*"). While this type of postal code provides an extra level of detail, it also makes querying and aggregating postal codes together much more difficult.

To standardize the postal codes, all ZIP+4 formats were trimmed of their last four digits and the trailing `"-"`. Other cleaning was also done, including removal of any non-digit characters.

After cleaning, a query that counts the number of each zipcode was done to ensure that the audit was successful:

```
  SELECT value, COUNT(*)
    FROM nodes_tags
   WHERE key = "postcode"
GROUP BY value
ORDER BY COUNT(*) DESC
   LIMIT 10;
```

```
27701|164
27513|149
27510|143
27511|94
27560|66
27519|52
27601|47
27705|44
27707|40
27514|33
```

All of the above postal codes resulting from the query appear to be correct, as they all correspond to postal codes found in the Raleigh-Durham area.

Street Name Abbreviations

Using a combination of expected values, mappings, and regular expressions, I was able to make a system that corrects the most popular abbrevations in street names.

Implemented in all of the audit scripts is code that writes any changes to a text file. The changes can be reviewed after the XML to CVS conversion and audit is complete to resolve any missing or unexpected values.

A sample of the street name changelist file is provided below. The left side is the old value, while the right side is the corrected value:

```
changelist_postal.txt

[["Falls of Neuse Rd", "Falls of Neuse Road"]]
[["Waterford Lake Dr", "Waterford Lake Drive"]]
[["Waterford Lake Dr", "Waterford Lake Drive"]]
[["Buck Jones Rd", "Buck Jones Road"]]
[["Meadowmont Village CIrcle", "***Unknown Mapping. No changes made.***"]]
[["Creedmoor Rd", "Creedmoor Road"]]
[["Main at North Hills St", "Main at North Hills Street"]]
[["Durham-Chapel Hill Blvd.", "Durham-Chapel Hill Boulevard"]]
```

Exceptions not corrected by the audit remained unchanged in the final CSV file, while a special reply is inserted in the text file to bring attention to the exception (`"***Unknown Mapping. No changes made.***"`). A perfect example is the `"Meadowmont Village CIrcle"` entry. This message allows the user to review the exception and update the mapping in the script accordingly for future audits.

Removing carriage return values from database

After creating the database tables in SQLite, I attempted to import the CSV files. However, upon doing so, I found that many of the values, especially those in the last fields of each table, ended in carriage return values. These values made it impossible to query anything in the final fields of those tables without having formatting issues.

It took me a little bit of investigative work to actually find out that the issue was a control character. I had to explore the values directly in Python to see that the values had carriage returns:

```
cursor.execute("SELECT * FROM ways_tags LIMIT 5")
pp.pprint(cursor.fetchall())
```

```
[(8322822, u'foot', u'yes', u'regular\r'),
 (8322822, u'name', u'North/South Greenway', u'regular\r'),
 (8322822, u'bicycle', u'yes', u'regular\r'),
 (8322822, u'highway', u'footway', u'regular\r'),
 (8322822, u'surface', u'asphalt', u'regular\r')]
```

To resolve this issue, I wrote a short Python script that programmatically removes the carriage return values from the database values.

```
import sqlite3

file_path = "C:\Users\sample_path\sqlite\OSMproject\OSM_db"

conn = sqlite3.connect(file_path)
cursor = conn.cursor()

# Remove carriage returns characters from specific columns
cursor.execute("UPDATE nodes SET timestamp=REPLACE(timestamp, '\r', '')")
cursor.execute("UPDATE nodes_tags SET type=REPLACE(type, '\r', '')")
cursor.execute("UPDATE ways SET timestamp=REPLACE(timestamp, '\r', '')")
cursor.execute("UPDATE ways_tags SET type=REPLACE(type, '\r', '')")

conn.commit()
conn.close()
```

Overview of database and Basic Statistics

This section covers some basic statistics about the database, queries done on the database, and other interesting findings from the queries.

Size of files

```
raleigh_north-carolina.osm ......... 465.0 MB
OSM_db.db ........................... 250.0 MB
nodes.csv ........................... 183.0 MB
ways_nodes.csv .....................  60.7 MB
ways_tags.csv ......................  29.6 MB
ways.csv ...........................  13.4 MB
nodes_tags.csv .....................   2.1 MB
```

Number of Unique Users

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

          UNION
```

```
          SELECT user
            FROM ways) AS total_users;
```

971

Number of nodes

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

2347549

Number of ways

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

241260

Top Ten Amenities for nodes

```
     SELECT value, COUNT(*)
       FROM nodes_tags
      WHERE key = 'amenity'
   GROUP BY value
   ORDER BY COUNT(*) DESC
      LIMIT 10;
```

```
bicycle_parking  573
restaurant       451
place_of_worshi  371
fast_food        183
bench            131
waste_basket     124
cafe             96
atm              78
school           76
parking          70
```

Interestly, `bicycle_parking` is the top listed amenity in the dataset. Perhaps Raleigh-Durham has a large biker population?

Top Five Contributors

```
   SELECT user, COUNT(*)
     FROM (  SELECT user
               FROM nodes

             UNION ALL

             SELECT user
               FROM ways) AS total_users
GROUP BY user
ORDER BY COUNT(*) DESC
   LIMIT 5;
```

```
jumbanho         1556845
JMDeMai          219539
bdiscoe          129730
```

```
woodpeck_fixbot  113815
bigal945         103684
```

The top user, `jumbanho`, account for ~60% of all node/way entries in the database!

Major Cities in nodes

```
  SELECT value, COUNT(*)
    FROM nodes_tags
   WHERE key = 'city'
GROUP BY value
ORDER BY COUNT(*) DESC
   LIMIT 5;
```

```
Cary             294
Durham           289
Raleigh          238
Carrboro         143
Chapel Hill      83
```

Major Counties in ways

```
  SELECT value, COUNT(*)
    FROM ways_tags
   WHERE key="county"
GROUP BY value
ORDER BY COUNT(*) DESC
   LIMIT 5;
```

```
Wake, NC         9421
Durham, NC       5212
Orange, NC       2188
Chatham, NC      442
Granville, NC;   25
```

Max/Min Longitudes and Latitudes

```
SELECT MAX(lon) AS max_lon,
       MIN(lon) AS min_lon,
       MAX(lat) AS max_lat,
       MIN(lat) AS min_lat
FROM nodes;
```

```
max_lon          min_lon      max_lat     min_lat
---------------  -----------  ----------  ----------
-78.577          -79.1159995  36.0509986  35.759
```

If you map these coordinates, you can draw the square area that the data was retrieved from by MapZen.

Ideas for Additional Improvement

Before doing this project, I had never heard of the OpenStreetMap project. It is entirely possible that the project was much more popular when the Udacity Data Analyst Nanodegree was first formed, and thus was a more well known project that could be implemented in the course. Over time though, it is possible that node/way contributions to the project have decreased over the years as the movement has become less popular, or as more locations in Raleigh-Durham have been mapped.

In order to better understand the situation, I decided that I would explore the `timestamp` data included in all of the `nodes` and `ways` in the osm file. To do so, I explored aggregated user contributions by year and graphed the results to a bar grpah.

First, I needed a query to gather all of the `timestamp` data from the `nodes` and `ways` in the database.

```
  SELECT timestamp
    FROM (SELECT timestamp
            FROM nodes

           UNION ALL

          SELECT timestamp
            FROM ways) AS total_rows
ORDER BY timestamp ASC;
```

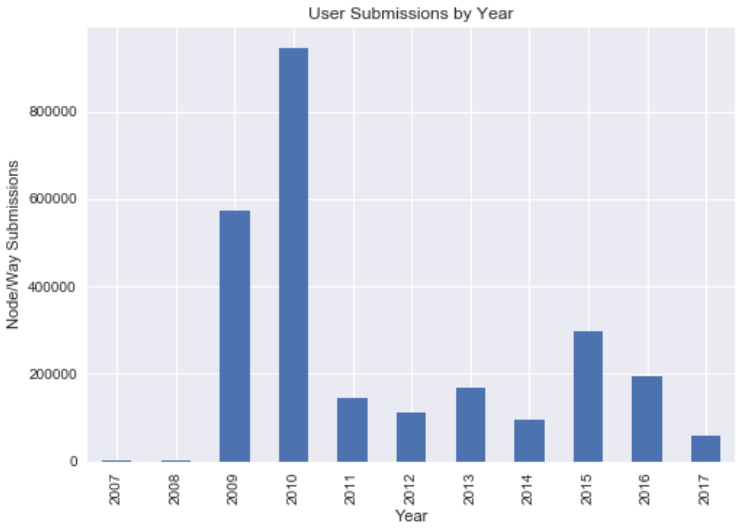
Once all of the timestamps had been collected, the timestamps were imported into Python and stripped of all datetime info except the year. User submissions for each year were then aggregated and counted.

```
# Collect timestamps from SQLite database
cursor.execute("SELECT timestamp \
               FROM (SELECT timestamp FROM nodes UNION ALL SELECT timestamp FROM ways) AS total_rows \
               ORDER BY timestamp ASC;")
raw = cursor.fetchall()

# Strip timestamp data except for the year
stripped = []
for x in listed:
    y = datetime.datetime.strptime(x, "%Y-%m-%dT%H:%M:%SZ")
    z = y.strftime("%Y")
    stripped.append(z)

# Put timestamps into pandas Series and count number of submissions for each year
reformed = pd.Series(stripped)
ts = pd.Series(reformed.value_counts(sort=False))
ts = ts.sort_index(ascending=True)
```

The values were then graphed on a bar chart:



The graph seems to suggest that node/way contribution to the Raleigh-Durham has decreased as the years have gone on. While one conclusion might be that OpenStreetMap is becoming less popular, it is entirwly possible that the Raleigh-Durham area has been sufficienntly mapped, and only requires periodic updating. OpenStreetMap's popularity may also have not decreased globally at all, as the forums and help pages are full of recent posts.

Regardless, in order to make the Raleigh-Durham project more relevent, it might make sense to make sub-regions within each country on OpenStreetMaps forums and help pages. This would allow users in these sub-regions to collaborate with each other and make more SMART goals to ensure progress is made in the region. Greater emphasis on updating and improving existing would impact the amount of activity for the Raleigh-Durham area, especially in standardizing data across all values. However, updating existing data is much more difficult, time intensive, and tedious than mapping new areas. This might push currently developers away from the almost complete Raleigh-Durham region, and instead focus their effort abroad.

Conclusion

Overall, this dataset felt like a good representation of Raleigh-Durham area. Though certainly not complete, the dataset did include nodes and ways from all over the Research Triangle. While improvements in the uniformity of the key-value pairs and updates to depreciated values can be made, the dataset is usable in its current form. I hope that contributions to the Raleigh-Durham OpenStreetMap region sees a resurgence in updates, as it provides coders like myself plenty of data to explore, analyze, and experiment with.