Project-3-Wrangle-OpenStreetMap-Dataset

Wrangle OpenStreetMap data using MongoDB within a Python environment using PyMongo.

Introduction

You will choose any area of the world in Open Street Map and use data munging techniques, such as assessing the quality of the data for validity, accuracy, completeness, consistency and uniformity, to clean the OpenStreetMap data for a part of the world that you care about. Finally, you will choose either MongoDB or SQL as the data schema to complete your project. Project Rubric

Dataset

The Brooklyn, New York Open Street Map dataset was used for wrangling, cleaning, and querying the MongoDB database. The XML file type was used while exploring this dataset.

Installation

The MongoDB installation guide should be completed by the user if follow up queries are to be run on the wrangled dataset.

Wrangling

Problems Encountered in The Map

After initially inspecting the Brooklyn Open Street Map XML dataset the following problems were discovered and will be discussed as to how it was addressed:

- Inconsistent and over abbreviated street types
- Invalid zip codes

Unexpected Street Names Types Examples

- '507': set(['Mott St #507']),
- '7th': set(['32nd street with 7th']),
- '861': set(['861']),
- 'A28': set(['wyckoff ave unit A28']),
- 'Floor': set(['Dekalb Ave, 2nd Floor', 'Wall Street 12th Floor']),
- 'Floor)': set(['Manhattan Avenue (2nd Floor)']),
- 'Plz': set(['University Plz']),
- 'Rb': set(['Linden Boulevard Outer Eb Rb']),
- 'bus_stop': set(['bus_stop']),
- 'floor': set(['DeKalb Avenue 4 floor'])}

Out of all the street names in the dataset from the Brooklyn OSM XML file, there were 99 unique street names that needed to be changed or deleted. For example, '11th St.' should be '11th Street', with abbreviations and some other unexpected street types the cleaning was done systematically through regular expressions and replacing the values when writing a clean XML file. Other street types had to be hard coded to change the values to the correct values when writing the output file, for example '305 Schermerhorn St., Brooklyn, NY 11217' should be 'Schermerhorn Street'. All other street types that did not make sense, or the true value could not be determined were changed to 'NaN'.

Unexpected Zip Code Examples

- '10004'
- '10005'
- '10006'
- '10007'
- '10001'
- '10002'
- '10003'

All unexpected zip codes found that were not listed in the government NYC Brooklyn zip code was replaced with 'NaN' value.

Data Overview

This section contains basic statistics about the dataset and the MongoDB queries used to gather them.

File sizes

brooklyn_new-york.osm ..... 653 MB

output.osm.json .... 719 MB

Number of documents

```
db.brooklyn.find().count()
```

49584653

Number of ways

```
db.brooklyn.find({'type' :'way'}).count()
```

7635808

Number of nodes

```
db.brooklyn.find({'type' :'node'}).count()
```

45006516

Number of unique users

```
len(db.brooklyn.distinct("created.user"))
```

1574

Top 1 contributing user

```
db.brooklyn.aggregate([{'$group': {'_id':'$created.user', 'count':{'$sum':1}}}, {'$sort':{'count':1}}, {'$limit':1}])
```

{u'count': 13, u'_id': u'haoyu'}

Number of users appearing only once (having 1 post)

```
db.brooklyn.aggregate([{'$group': {'_id':'$created.user', 'count':{'$sum':1}}}, {'$group': {'_id':'$count', 'num_users':{'$sum':1}}}, {'$sort':{'_id':1}}, {'$limit':1}])
```

{u'num_users': 65, u'_id': 15}

Top 10 appearing amenities

```
db.brooklyn.aggregate([{'$match': {'amenity':{'$exists':1}}}, {'$group': {'_id':'$amenity', 'count':{'$sum':1}}}, {'$sort':{'count':1}}, {"$limit":10}])
```

{u'count': 1, u'_id': u'car_service'}, {u'count': 1, u'_id': u'car_parking'}, {u'count': 1, u'_id': u'music_venue'}, {u'count': 1, u'_id': u'social_facility'}, {u'count': 1, u'_id': u'funeral_directors'}, {u'count': 1, u'_id': u'Green_Market'}, {u'count': 1, u'_id': u'stock_exchange'}, {u'count': 1, u'_id': u'artwork'}, {u'count': 1, u'_id': u'nail_salon'}, {u'count': 1, u'_id': u'crematorium'}

Highest population religion

```
db.brooklyn.aggregate([{'$match': {'amenity':{'$exists':1}, 'amenity':'place_of_worship'}}, {'$group': {'_id':'$religion', 'count':{'$sum':1}}}, {'$sort':{'count':1}}, {'$limit':1}])
```

{u'count': 1, u'_id': u'unitarian_universalist'}

Most popular cuisines

```
db.brooklyn.aggregate([{'$match': {'amenity':{'$exists':1}, 'amenity':'restaurant'}}, {'$group': {'_id':'$cuisine', 'count':{'$sum':1}}}, {'$sort':{'count':1}}, {'$limit':2}])
```

{u'count': 1, u'_id': u'Po_Boys'}, {u'count': 1, u'_id': u'southern,_diner'}

Suggestive Improvement Implenmentation

One idea for improving the OSM dataset for different areas would be to have two versions of the dataset, production and staging. The production dataset would be unwrangled and raw, like the original one used in this dataset. The staging verion would run data wrangling such as what was implimented in this project and change various unexpected/unacceptable values to 'NaN'. Then the community could focus on updating the missing 'NaN' values with the correct values and merging the database into the production dataset.

Another approach would be to set the 'NaN' postal code values by using the GPS 'pos' attributes. Each address has a GPS attribute, by finding the closest address with a valid postal code, and changing the 'NaN' value to that valid postal code. The accuracy of this method may be off but this can be implimented with code, and therefore would be much faster than user generated data.

Number of Addresses with 'NaN; Value for Zip Code Value After Wrangling

```
db.brooklyn.aggregate([{'$match': {'address.postcode':{'$exists':1},
'address.postcode':'NaN'}}, {'$group': {'_id':'$address.postcode', 'count':{'$sum':1}}},
{'$sort':{'count':1}}])
```

```
{u'count': 702464, u'_id': u'NaN'}
```

Conclusion

The Open Street Map Brooklyn XML file had many errors for just the street types and the zip codes alone. The errors or inconsistences found are not that surprising given the size and scope of Brooklyn and the OSM file. The Brooklyn OSM file has over, 42 million documents, 6 million streets, and 38 million locations within the XML file. Given the vast size of the dataset, the dataset was relatively clean but, only the street types and zip codes were wrangled and cleaned in this analysis. The dirtiness of the Brooklyn OSM XML dataset found in this analysis is just the tip of the iceberg if a 100% cleaned dataset is desired.

References

- https://www.openstreetmap.org
- https://review.udacity.com/#!/rubrics/25/view
- https://mapzen.com/data/metro-extracts/metro/brooklyn_new-york/
- https://s3.amazonaws.com/metro-extracts.mapzen.com/brooklyn_new-york.osm.bz2
- https://docs.mongodb.com/v3.2/tutorial/install-mongodb-on-windows/
- https://www.health.ny.gov/statistics/cancer/registry/appendix/neighborhoods.htm