OpenStreetMap Project
Data Wrangling with MongoDB

by Larry Schwerzler

Map Area: Seattle, Wa USA
https://www.openstreetmap.org/relation/237385
https://mapzen.com/data/metro-extracts (OSM XML)

- 1. Problems Encountered in the Map
  - Street Name Abbreviations
    - Audit Data
    - Create a cleaning plan
    - Execute Plan
    - Iterate
- 2. Data Overview
  - Files
  - Data Basics Statistics
    - Total number of documents in the collection
    - Total number of nodes in the collection
    - Total number of ways in the collection
    - Total users contributing to the data in this collection
    - User contributing the most content
- 3. Additional Ideas
    - Further Analysis - Coffee Shops
- 4. Conclusion

1. Problems Encountered in the Map

After a brief audit of the entire Seattle dataset a few obvious items showed up as candidates for corrective action.

1. Abbreviated street names were rampant throughout the dataset.
2. Data from outside the Seattle area.

One of these was fixed using the methods we covered in the lessons for the Data Wrangling course, one of them was not fixed, but possible solutions are put forth.

Street Name Abbreviations

This were many street names that were abbreviated. The inconsistency makes it hard to perform more complex analysis reliably because the same things are labeled with different names. An effort was made to address this issue using the following data cleaning blueprint:

1. Audit Data
2. Create a cleaning plan
3. Execute plan

Audit Data

First I did a data audit to see what the data looked like and where I should focus my efforts. I grouped the data together based on the last word in the address. This gave me two pieces of important information

1. What the actual street names were
2. How frequently that street name was used

This analysis was helpful to see that there are many variations of common street names (Road abbreviated as Rd, rd, and ROAD etc).

Create a cleaning plan

Once I had identified the the source data had these inconsistent names in the address field, I created a plan to clean the data to be more consistent. I decided to standardize on the full word with leading capitalization e.g. Road, Avenue, Street. I then looked through the list and frequency info generated from the auditing step. I decided I would focus on the items that had at least 5 items categorized in that same way in order to target just the largest number of mislabeled streets.

As I was going through the list generated in the data auditing step and I came across an item that meet my correction criteria I would add it to a python dictionary with the key being the mislabeled name and the value being the correct label that I wanted applied.
Example
{'St.': 'Street'}

Also when I encountered a correctly labeled street (Road, Avenue, Street etc) I would add it to a "known good" street names python list to help process the data faster by not having to relabel streets that were already correctly labeled.

Once the known goods were taken out of the set of possible corrections that left 212 individual categories that were potentially poorly labeled streets. Of this 52 items were identified for corrective action. All told these 52 corrections resulted in changing of 453 individual addresses to the standardized way identified.

I inserted this relabeling step into the procedures we used in the lessons of Part 6 of the Data Wrangling class. The processing of the mislabeled street names happened inline with the conversion from the XML data to JSON

Execute Plan

The execution of the plan went as expected. The most notable part is that on a well spected system (32GB ram, SSD hard drive with 12 CPU cores running ubuntu) the conversion from XML to JSON with the other processing still took significant time. 10 minutes or so for each run through of the complete data.

453 Individual addresses were corrected in the data processing steps.

Iterate

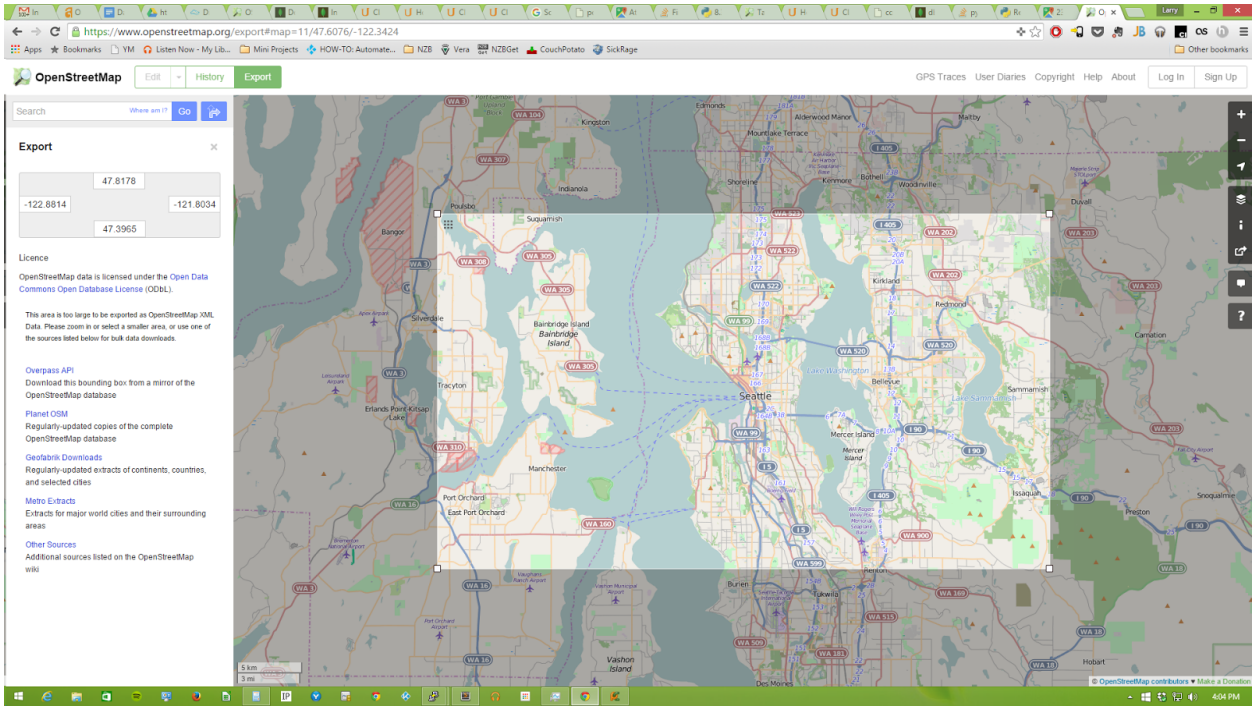
For this exercise the iteration step of a normal data munging blueprint was skipped on the processing side with thoughts that it could be performed on the database side once the JSON file was imported into MongoDB.

Unfortunately time worked against me and with some technical struggles I was not able to iterate to clean more of the data as I would have liked.

I will discuss possible further cleaning in the Additional Ideas section of this document.

Data outside the Seattle city limits

It didn't take long to discover that the data provided from the Mapzen site contained many addresses outside of Seattle. It is unclear what the bounding box download looked like for the Mapzen data extract, but going to the OpenStreetMaps site and attempting to export the data shows that the assumption that the download contains addresses outside the city of Seattle is very likely.



This shows the bounding box when you select to export the “seattle” area on the OSM website. This box contains much more than just the City of Seattle proper and the data would need to be filtered further to include only items from the city.

Unfortunately no readily apparent method could be identified to easily filter just the information from within the city. The best method would probably involve a boundary of GPS points to be included and matching each node against that boundary to determine if it falls within the city. Such comparison is beyond my skills at this time.

2. Data Overview

Included are basic statistics and information on the data.

Files

There are two files involved in this analysis.

1. seattle_washington.osm (1.4GB) - This was obtained as a zipped OSM XML document directly from the Mapzen metro extracts page on September 27, 2015 and is the source data for this analysis.
2. seattle_washington.osm.json (1.6GB) - This file is the result of cleaning of the data and conversion from the XML format of the source to a JSON format that was used to import the data into MongoDB.

Data Basics Statistics

The following statistics were obtained from the MongoDB server using pymongo

Total number of documents in the collection

```
db.seattle.find().count()
7363874
```

Total number of nodes in the collection

```
db.seattle.find({'type': 'node'}).count()
6720264
```

Total number of ways in the collection

```
db.seattle.find({'type': 'way'}).count()
643458
```

Total users contributing to the data in this collection

```
len(db.seattle.distinct('created.user'))
2678
```

User contributing the most content

```
for r in db.seattle.aggregate([{"$group":{"_id":"$created.user", "count":{"$sum":1}}},
{"$sort":{"count": -1}}, {"$limit":1}]):
...     print r
...
{u'count': 1224341, u'_id': u'Glassman'}
```

3. Additional Ideas

My daily job is working for a Seattle area Transportation agency and I was hoping to do some analysis around bus routes or stops perhaps broken by region or neighborhood. I ran into trouble identifying a reliable region indication in the nodes, so I had to give up on this idea.

My fall back was to do a comparison of Starbucks vs other coffee shops.

Further Analysis - Coffee Shops

I started this task trying to identify the best way to find coffee shops in the data. I did a tag analysis to find high usage tags in the data set. I found 1759 unique tags identified in the OSM data for this

download. I used python to build two lists of tags, One list contained the number of instances (tag_count) of a tags usage, which I used to find tags that had high usage (more likely to contain an indication of coffee shop etc). The other list compiled a python set of the values from each tag (tag_values). I used the tag_count list to first identify a category to examine, then I would use the tag_values list to see if any of the values were coffee related.

I discovered the best option would be the 'cusine' tag, which had many values used, but two that were of particular interest to me. coffee and cafe.

At this point I turned to MongoDB and used the pymongo package to run some queries on the collection to try and shed some light on the disparity of starbucks to other coffee shops in the area.

I used a regular expression 'coffee|cafe' and searched through the 'cusine' data to find a list. then grouped on the name, sorted by the most common and limited it to the top 10 results

```
 >>> reg = re.compile(r'coffee|cafe', re.IGNORECASE)
>>> for d in db.seattle.aggregate([{'$match': {'cuisine': reg, 'name': {'$ne': None}} },{'$group':
{'_id': '$name', 'count': {'$sum': 1}}},{'$sort': {'count': -1}}, {'$limit': 10}]):
...     print d
...
{u'count': 115, u'_id': u'Starbucks'}
{u'count': 7, u'_id': u'Uptown Espresso'}
{u'count': 6, u'_id': u"Starbuck's"}
{u'count': 5, u'_id': u'Starbucks Coffee'}
{u'count': 5, u'_id': u"Tully's Coffee"}
{u'count': 4, u'_id': u'Top Pot Doughnuts'}
{u'count': 4, u'_id': u'Caffe Ladro'}
{u'count': 4, u'_id': u"Tully's"}
{u'count': 3, u'_id': u'Herkimer Coffee'}
{u'count': 3, u'_id': u'Diva Espresso'}
```

As you can see Starbucks dominates the Seattle coffee market, which really isn't very surprising. If you include the three name variations starbucks has 126 shops from this data set. The next most popular shop (when you consider naming variations) is Tully's with 9.

The 9 number for Tully's seemed really low to me, so I went to their website (http://www.tullyscoffeeshops.com) to see if they had a quick search for shops and they do. Using Seattle, WA as my search parameter and manually counting the Tully's coffee shops identified that are known to be within the seattle city limits I counted 13 total. So there are definitely a few missing from my query.

One possible reason is that the Tully's missing from my list have other slight variations in the name that prevent them from showing in the grouped llist. A quick query confirms this:

```
>>> reg2 = re.compile('tully', re.IGNORECASE)
```

```
>>> db.seattle.find({'name': reg2}).count()
19
```
So 19 records.

Why more than 13? My guess would be that the dataset boundary includes areas east of seattle (Bellevue, Kirkland) that also contain Tully's shops. In looking at the data via various ways there does seem to be towns included that are far from the city, even some into Canada, which is 100 miles from the north border of the city. A query should be able to clear things up:

```
db.seattle.find({'name': reg2, 'address.city': 'Seattle'}).count()
2
```

In this case only two of the identified 19 records have an address.city value of Seattle. I'm not sure what the reason is behind this, and further examination of the data would be needed to find out why.

What about Buses?

My initial goal with this project was to analyze the transit specific information contained in the OpenStreetMap dataset. This is interesting to me both personally (long time transit user in the Seattle area) and also professionally (I currently am employed by King County Metro, the largest transit agency in the Seattle area). However, I was disappointed to discover a lack of local transit related information in the data that I downloaded to analyze. To modify the analysis of coffee shop data that I performed I'd like to also include coffee shops that are served by the most bus routes. In order to do this I will need to include another set of data to supplement the data provided by OpenStreetMaps.

King County Metro provides data related to the stops that it and other local transit agencies service in the King County region through the General Transit Feed Specification (GTFS) provided by Google to share transit data https://developers.google.com/transit/. King County provides several data feeds related to transit data such as:

- stops.txt (https://developers.google.com/transit/gtfs/reference?csw=1#stopstxt) - defines the stops included in the system, their latitude and longitude as well as some information about if the stop is part of a stations or is wheelchair accessible.
- routes.txt (https://developers.google.com/transit/gtfs/reference?csw=1#routestxt) - defines the routes that are operated, the number of the route and some basic information
- trips.txt (https://developers.google.com/transit/gtfs/reference?csw=1#tripstxt) - defines the individual instances of routes one route will have many trips a day that operate the same service.
- shapes.txt (https://developers.google.com/transit/gtfs/reference?csw=1#shapestxt) - the paths that the routes actually take and distances traveled.

This is just a sampling of this information, there are several other files provided by King County for the local Seattle area.

These data feeds are simple text files formatted as CSV files, and they are provided directly via download from http://metro.kingcounty.gov/GTFS/

In order to find coffee shops that have the most transit accessibility there are three parts that will need to be completed:

1. Load the data and combine it with the OpenStreetMap dataset, including the following parts
    a. Stops.txt - this is needed to know what stops are in the system and will be the items combined with the OpenStreetMaps data.
    b. Routes.txt - needed to know what routes are available, but there is not a great way to include this in the OSM data, so it wouldn't be combined with the OpenStreetMaps and Stops information, probably this will need to be kept in a separate data structure or schema in Mongo
    c. Trips.txt - completes the route information.
2. Process the data to find the number of trips that service a particular stop. A stop with more trips has better transit accessibility. In the real world this would equate to more buses stopping at the stop in the day, by using trips we are not just looking at stops that are serviced by lots of routes, but stops that have the most buses coming and going.
3. Find the distances from the stop to the nearest coffee shop.

The first two parts of this process are things we have worked through in the examples already, and in the end are trivial to implement with some trial and error. Part 3 is something touched on in the lessons but distance finding between points is not something covered. Incorporating this into the analysis would require making use of the mongo geospatial features.

I think adding the local transit information to the OpensStreetMap data would be a good benefit for the project. Looking on the OpenStreetMap Wiki I did find a public transport schema that is intended to be used to include such information. http://wiki.openstreetmap.org/wiki/Proposed_features/Public_Transport It would be nice to loop back around and include this changing transit information into the project on a regular basis. Most of the information in the transit feeds from King County do not change much, but there are occasional updates and adjustments made and a one time upload of the information may not be beneficial once the data becomes outdated. The schema includes ways to make Stops and Route information available on the map.

One downside here is that the trip/timetable type info is not accounted for in the schema. At first this is disappointing, and means in practice the inclusion of this data in the OpenStreetMap dataset would be of marginal assistance to someone planning a trip or wanting arrival info through the regular map interface. However this is a big step in the right direction to make the information available to other applications outside of the OpenStreetMap ecosystem. Keeping this information updated regularly and timely would allow others to use what OpenStreetMap has built and has made available in conjunction with the data feeds from King County to provide sytems that would give arrival type information.

4. Conclusion

The data for the Seattle area is not as complete as I expected it to be. I am surprised by the number of addresses that are not identified as Seattle addresses. They either don't have the city set properly, or my processes have mislabeled them in some way that is not apparent to me.

The data cleaning on the address field was a good first step, but more cleaning is needed. In looking through the data it appears that there are many places in the address that the street information can appear and it doesn't appear that my steps caught all those instances. An iteration to clean streets that appear in the middle or start of the address would be a good first step.

Additionally for the coffee analysis some standardization of the name field would help in making the data easier to analyse