Project 3: Data Wrangling with MongoDB: OpenStreet Map Project

Judy Jensen

jbark1967@gmail.com

Submission 1.1 (Aug 16, 2015)

Section 1: Challenges Encountered with the Data set

The size of the data set was difficult to manage on my PC. I adjusted 'sample_dataosm' to 1:50 in order to develop and test, and finally reloaded Python 2.7 (64-bit) in order to process the entire OSM file to JSON.

Processing and Cleaning the XML data:

In addition, an element is missing which raises a ParseError which requires additional processing beyond the processing code developed in Lesson 6 as noted below:

```
'''
  try:
    for event, element in ET.iterparse(filename, events = ('start', )):
        ##code###
  except ET.ParseError:
    pass
'''
```

Blueprint:

Using the code developed in Lesson 6, an audit was completed on the xml file focused on the address key. addr:street, addr:postcode, addr:city, the highway key. Additional cleaning was completed on the amenity key after the collection was in MongoDB.

I expected that the issues with the Chicago dataset would be similar to the Portland dataset – however, I found that street names (a problem in the Chicago dataset) is a very smallkey and the amapping dictionary had 8 entries (e.g. HWY: Highway). (<0.3% of entries contain an address). More information is available under the 'highway' key in the Portland dataset (4%), but still a surprisingly low rate. The highway values were very clean, and did not require additional cleaning however (indicating these were either entered programmatically, or a good protocol exists for entering these values).

When I dug into the amenity field to learn more about the dataset, it appears that these entries probably do not reflect Portland accurately. Looking at the top 5 and bottom 5 amenities by count, it seems highly unlikely that places of worship are more prevalent than restaurants, and there are only 2 hotels in the collection.

Based on this audit, addr:street was selected for two cleaning steps, and addr:postcode was selected for on cleaning step on the xml dataset. Sections of the audit code, cleaning code, and results are noted below:

Cleaning discussion:

Street names – the street names are very clean. An audit indicated 8 abbreviations that are updated in the final dataset. There were several street names that are not in the expected dictionary, including Broadway, Circle, Loop, and numbered highways. These were ignored when the OSM file was updated.

See audit_addrstreet_clean.py for the audit and cleaning process, and here is a subset of the updates.

Step 1 –clean street names

```
Northwest Bronson Crest Loop => Northwest Bronson Crest Loop
SW Beaverton-Hillsdale Hwy => SW Beaverton-Hillsdale Highway
NW Park Dr => NW Park Drive
SW 2nd St => SW 2nd Street
NE Broadways St => NE Broadways Street
Northeast Highway 99 => Northeast Highway 99
```

This output indicates a second cleaning step is needed for street names – standardization on the directional prefix (NE v. Northeast). This is incorporated into the audit_streetclean_final.py file. The update name function processes each street name for updating for the suffix, and then the directional prefix:

Code to clean suffix and then prefix:

```
def update_name(name, mapping, map_prefix):
    dirty_name = name
    for map in mapping:
        if re.search(re.escape(map)+ r'$', dirty_name):
            name = re.sub(re.escape(map)+r'$', mapping[map], dirty_name)
    dirty_name = name
    for map_ in map_prefix:
        if re.search(map_, dirty_name):
            name = re.sub(map_, map_prefix[map_], dirty_name)
```

Result - cleaned street names:

```
Northwest Bronson Crest Loop => Northwest Bronson Crest Loop
Southwest Jamaica => Southwest Jamaica
SW Beaverton-Hillsdale Hwy => Southwest Beaverton-Hillsdale Highway
SW 2nd St => Southwest 2nd Street
Southwest Broadway => Southwest Broadway
```

Zip codes – Zip codes data are largely in the expected format (5 digits) with a few 5 + 4 included. The osm file has a cleaning step to make these all 5 digit.

```
def update_zip(zip):
    old_zip = zip
    if len(zip) > 5:
        zip = zip[ :5]
    return zip
```

Restaurant names – it wasn't a significant issue, but I did update all documents with 'name' : 'Starbucks' to include 'cuisine' : 'coffee_shop' as this was an inconsistency in an amenity (restaurants, cafes, etc) that I was interested in (PortlandMongoUpdate).

```
query = {'name': 'Starbucks', 'amenity' : 'cafe'}
update = {'$set':{'cuisine' : 'coffee_shop'}}
portland_oregon = db.portland_oregon.update(query, update, multi = True)
```

Section 2: Data Overview

File sizes

Portland_oregon.osm 845MB

Portland_oregon.osm.json: 950MB

Sample_data.osm = 8.5M (used 1:100 for sample set)

Data Summary

PortlandMongoCounts was utilized for items in blue and is a simple match and project program. PortlandMongoAggregate utilizes a pipeline and was used extensively to review statistics.

Pipeline in PortlandMongoAggregate:

```
match = {'type' : {'$exists' :1}}
group = {'_id': '$type', 'count': {'$sum' : 1}}
result= db.portland_oregon.aggregate([{'$match': match},{'$group' : group}, {'$sort': {'count': -1}}])
```

Overview of datasest (PortlandMongoCounts, PortlandMongoAggregate)

```
Total (_id): 4,074,290
Unique users (created.user) = 1,026
'node': 3643940; 'way': 430,281
amenity: exists, not exist: amenity 11206 4063084 (0.3%)
address.streets: 204715
highways: 163,316
```

Details

Most common amenities (PortlandMongoAggregate)

Common sense would indicate that the amenities are not a representative sample – it appears that public services in particular are overly represented compared to other amenities (i.e 900 schools compared to 2 hotels).

```
{u'_id': u'parking', u'count': 2503}
{u'_id': u'place_of_worship', u'count': 1174}
{u'_id': u'restaurant', u'count': 1076}
{u'_id': u'school', u'count': 911}
{u'_id': u'bicycle_parking', u'count': 537}
```

Top 5 contributors (PortlandMongoAggregate)

A review of the top contributors may explain the over representation of public amenities. Further investigation could include a review of amenities w/o contributors w/ pdxbuildings in the user name (created.user). Unique contributors = 1,026; Most Prolific user = Peter Dobratz_pdxbuildings (combined with Peter Dobratz for 42% of entries).

```
{u'_id': u'Peter Dobratz_pdxbuildings', u'count': 1556434} (38%)
{u'_id': u'Mele Sax-Barnett', u'count': 627999}
{u'_id': u'Darrell_pdxbuildings', u'count': 369974}
{u'_id': u'Grant Humphries', u'count': 330055}
{u'_id': u'baradam', u'count': 169480}
{u'_id': u'Peter Dobratz', u'count': 155028} (4%)
```

Section 3: Ideas for deeper analysis

A few of the issues with the data related to my specific quest include: Data is incomplete for many amenities outside of municipal services, information is entered in a non-standard way (e.g. street is missing, but highway name is available), and information that would be useful for aggregating information (e.g. by city is missing) is available from other data (i.e. zip code is available).

My assumption is that much of the incomplete information is available to created.user, but it is not entered. Some investigation could be done to the pdxbuildings (most frequent contributors) to determine if they would be willing to modify their procedure to include more information. A second step would be to utilize a second database such as google maps or yelp to complete the missing details.

The challenges of focusing on the frequent contributors is that they will be specific to each map area. This is not a scalable solution for the entire database. However, if a particular group of contributors fits a profile and is large, developing a portal available through openstreetmap that focuses on entering more complete information may address this challenge.

A solution that may address some or all of the shortcomings is using an additional database(s) to pull in missing information. This could be done through openstreetmaps.org, or by an individual utilizing the xml datafile. Like the first challenge, a more scalable solution is to utilize a tool through openstreetmaps.

This is not an area of expertise, but there are many potential sources of information, particularly on amenities, including yelp (regional), Google, and Bing. In the Portland example, the most common source is the USGS National Hydrography Dataset, and this is also a potential source.

Finally, addressing some information, such as completing cities where a postcode is provided would be easily accomplished with a dictionary that includes postcode:city as k:v pairs and using a lookup in the cleaning step (unfortunately the inverse is not true as many cities have multiple postcodes)

This last step, along with creating a visualization with the pos and amenities of interest would be a next step to address my question of neighborhoods to investigate to live in Portland, which was my original goal.