OpenStreetMap Project
Data Wrangling with MongoDB

Peter F. Raig

Map Area: Miami, Fl

1 DATA ACQUISITION

Data was downloaded from the OpenStreetMap web database using the overpass –api website (http://overpass-api.de/query_form.html). The area around the city of Miami was selected manually using the following query:

(node(25.6372,-80.4989,25.9272,-79.9599);<;);out meta;

This produced a data file with approximate size of 79Mb.

2 FIRST LOOK AT THE DATA

2.1 VISUAL INSPECTION

My first action with the data was to look at it in an editor. I loaded the file into VIM, hitting Ctrl->c after a few moments to just grab the first chunk of the file. Looking at the first few lines, I could see that the nodes where in my region as expected:

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="Overpass API">
<note>The data included in this document is from www.openstreetmap.org. The data is made available under ODbL.</note>
<meta osm_base="2015-05-02T18:40:02Z"/>

  <node id="98974571" lat="25.7033440" lon="-80.2619630" version="2" timestamp="2010-01-07T23:47:01Z" changeset="3566921" uid="147510" user="woodpeck_fixbot"/>
  <node id="98974573" lat="25.7036230" lon="-80.2626460" version="2" timestamp="2009-09-25T06:22:27Z" changeset="2612131" uid="147510" user="woodpeck_fixbot"/>
  <node id="98974574" lat="25.7035340" lon="-80.2650050" version="2" timestamp="2010-01-07T23:47:01Z" changeset="3566921" uid="147510" user="woodpeck_fixbot"/>
  <node id="98974575" lat="25.7034920" lon="-80.2665340" version="2" timestamp="2009-09-25T06:22:27Z" changeset="2612131" uid="147510" user="woodpeck_fixbot"/>
```

2.2 NODES

During my first pass through, I only saw “nodes.” I was concerned there were no “way” lines, but since I was only sampling a little of the data in my visual inspection, I decided to look programmatically.

Using code based off Lesson6-1 I found the following elements and their counts.

This was a relief. There are plenty of “way” elements to work with. On the other hand, it’s also a concern. There are considerably more than that many buildings in Miami.

```
'member': 47984,
 'meta': 1,
 'nd': 379818,
 'node': 298947,
 'note': 1,
 'osm': 1,
 'relation': 457,
 'tag': 400635,
 'way': 49164
```

2.3 TAGS

I was able to list out all the “tag” element keys using code based off Lesson6-2. This gives me an idea to the schema.

```
'addr:city': 607,
 'addr:country': 26,
 'addr:full': 2,
 'addr:housename': 36,
 'addr:housenumber': 630,
 'addr:postcode': 476,
 'addr:source': 2,
 'addr:state': 1041,
 'addr:street': 664,
 'addr:suite': 1,
 'addr:unit': 1,
```

Of specific concern though is the address keys. There are just not that many of them and there are definitely not enough “suite” and “unit” entries. With 1041 “state” entries one would expect to find comparable “postcode” or “street” entries. I will need to look for partially completed addresses.

3 CLEANING DATA

3.1 STREETS

Using Code similar to Lesson6-3, a couple of issues become apparent.

1. Suite and Apartment numbers in the street field. Special code will need to be written to address this. Examples:
    a. '108': set(['NW 20th Street, #108']),
    b. '19': set(['Harding Ave #19']),
    c. '207': set(['South Dixie Highway, Suite 207']),
    d. '328': set(['Biscayne Boulevard, Suite 328']),
    e. '710': set(['Alton Road, Suite 710']),

    f. '8,9,10': set(['W 49 St Suite 8,9,10']),
2. Additional street types can simply be added to the “expected” array:
    a. Terrace
    b. Way
    c. Path
    d. Highway
    e. Circle
3. A number of abbreviations need to be expanded. These will be handled via mapping
    a. Ave
    b. St
    c. Blvd
    d. Etc….
4. “Cirlce” is misspelled but can be handled via mapping

3.2 POSTAL CODE

Similar to the example project, postal code needs work here too. Some of the entries start with a “FL“ and others do not. A simple replace will fix this.

4 DATA OVERVIEW

4.1 FILE SIZES

Original data file: Miami.osm----79,748 Kb

Json export from mongo db: Maimi.json----107,796 Kb

4.2 DATA ANALYTICS

4.2.1 Total Records

```
> db.miami.find().count()
```

358977

4.2.2 Count of “nodes”

```
> db.miami.find({"type":"node"}).count()
```

309812

4.2.3 Count of “ways”

```
> db.miami.find({"type":"way"}).count()
```

49163

4.2.4 Count of Distinct Users

```
> db.miami.distinct("created.user").length
```

455

4.2.5 User with Most Entries

4.2.6

```
> db.miami.aggregate( [ {"$group":{  "_id":"$created.user", "count":{ "$sum":1}}},
                        {"$sort":{"count":-1}},
                        {$limit:1}])
{ "_id" : "woodpeck_fixbot", "count" : 58005 }
```

4.2.7 >Count of Users With Single Entry

```
> $
{ "_id" : 1, "num_user" : 88 }
```

4.3 ADDITIONAL ANALYTICS

4.3.1 Top Ten Appearing Amenities

```
> db.miami.aggregate([{$match:{amenity:{$exists:1}}},
                      {$group:{"_id":"$amenity",count:{$sum:1}}},
                      {$sort:{"count":-1}},
                      {$limit:10}])
{ "_id" : "school", "count" : 1039 }
{ "_id" : "parking", "count" : 579 }
{ "_id" : "kindergarten", "count" : 413 }
{ "_id" : "place_of_worship", "count" : 211 }
{ "_id" : "restaurant", "count" : 189 }
{ "_id" : "fast_food", "count" : 140 }
{ "_id" : "fuel", "count" : 99 }
{ "_id" : "police", "count" : 95 }
{ "_id" : "swimming_pool", "count" : 88 }
{ "_id" : "fire_station", "count" : 81 }
```

4.3.2 Biggest Religion

```
> db.miami.aggregate([ {$match:{amenity:{$exists:1},
                              amenity:"place_of_worship"}},
                      {$group:{"_id":"$religion",
                              count:{$sum:1}}},
                      {$sort:{count:-1}},
                      {$limit:1}] )
{ "_id" : "christian", "count" : 197 }
```

4.3.3 Most Prolific Fast Food

```
> db.miami.aggregate([ {$match:{amenity:{$exists:1},
                              amenity:"fast_food",
                              name:{$ne:null}}},
                      {$group:{"_id":"$name",
                              count:{$sum:1}}},
```

```
                        {$sort:{count:-1}},
                        {$limit:10}
] )
{ "_id" : "Subway", "count" : 26 }
{ "_id" : "McDonald's", "count" : 24 }
{ "_id" : "Burger King", "count" : 13 }
{ "_id" : "Wendy's", "count" : 13 }
{ "_id" : "Taco Bell", "count" : 10 }
{ "_id" : "KFC", "count" : 4 }
{ "_id" : "Papa John's", "count" : 3 }
{ "_id" : "Pollo Tropical", "count" : 3 }
{ "_id" : "Checkers", "count" : 3 }
{ "_id" : "Five Guys", "count" : 3 }
>
```

4.4 Improvement of Analysis

4.4.1 Multilayer Grouping

So far the analysis has all been on one field at a time. I think that it might be useful to break the data into groups within groups. For example, break the data by postcode and then look at the amenities within each to determine where there are deficiencies. That might be useful for someone trying to decide where to start a business or a schoolboard researching where more schools may be required.

4.4.2 Classifying

What could be done if the different amenities were classified in broad categories? For example, kindergarten, college, etc… could go under “education”. Doctors, hospitals, urgent cares, etc… could be grouped under “medical”. To do this would not be very different than the street cleanup algorithm. Another possibility would be the mongodb equivalent of a lookup table. I do not know how to do the mongodb equivalent of an sql subquery or join but I’m sure the documentation is there.

5 Conclusion

Considering the millions of people living in the Miami area, the data is clearly not fully collected. Still, this exercise was to show data cleaning skills and that has been demonstrated. Several fields lend themselves to cleaning efforts such as the street addresses. Additional data can likely be collected from real estate records, postal code assignments, and even Facebook listings. With more data, insights can be mined such as what franchises are in what areas if someone, for example, wanted to research where they wanted to open a fast food restaurant.