OpenStreetMap Sample Project
Data Wrangling with MongoDB

Roshan Shetty

Map Area: Raleigh, NC, United States

1. Problems Encountered in the Map

After initially downloading a small sample size of the Raleigh data and running it against a provisional data.py file, I noticed three main problems with the data, which I will discuss in the following order:

- Multiple streets for a node: A university like NCSU has multiple streets in it. But the street keys aren't of the form "addr:street".

  In such cases, the data is still cleaned for the streets but they are not added to an address dictionary value. Instead, the keys are kept as is.

- Relation nodes: There are some nodes who have element names as 'Relation'. They are handled in a similar way as ways and nodes.
- Postal codes: There are a lot of postal codes which don't pertain to Raleigh, NC. Some investigation is done around it to find out the root cause.

Multiple streets for a node

There are some tags which contain field values like "Street_1" and "Street_2". They seem be the adjoining streets of a building and hence the attributes are kept as they are.

Postal codes

Raleigh postal codes are in this range - 27587, 27601, 27605, 27608, 27609, 27612, 27613, 27614, 27615, 27616.[3] The database contains a lot of other values though. A lot of postal codes don't refer to Raleigh, NC but the surrounding areas.

```
>>> x_post_code = db.data.aggregate([{"$match":{"address.postcode":{"$exists":1}}}, {"$group":{"_id":"$address.postcode", "count":{"$sum":1}}}, {"$sort":{"count":1}}])

>>> list_post = list(x_post_code)
```

>>> for i in list_post: print i['_id']

...

A subset of the results is shown here.

27612-7156

27612-3326

27519-6205

27511-5928

In the above list, many postal codes like 27519, 27511 are not in Raleigh.

A query to find the list of cities confirms it too.

>>> x_city = db.data.aggregate([{"$match":{"address.city":{"$exists":1}}}, {"$group":{"_id":"$address.city", "count":{"$

>>> x_city_l = list(x_city)

>>> x_city_l

[{u'count': 1, u'_id': u'cary'}, {u'count': 1, u'_id': u'Ra'}, {u'count': 1, u'_id': u'Apex'}, {u'count': 2, u'_id': u'W

ake Forest'}, {u'count': 2, u'_id': u'durham'}, {u'count': 2, u'_id': u'chapel Hill'}, {u'count': 2, u'_id': u'raleigh'}

, {u'count': 108, u'_id': u'Morrisville'}, {u'count': 236, u'_id': u'Chapel Hill'}, {u'count': 279, u'_id': u'Carrboro'}

, {u'count': 885, u'_id': u'Raleigh'}, {u'count': 1295, u'_id': u'Durham'}, {u'count': 1745, u'_id': u'Cary'}]

Cities like Apex and Morrisville which are around Raleigh are also included in this extract. So the data gives info of Raleigh and its surrounding areas.

2. Data Overview

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

File sizes

raleigh_north-carolina.osm…… 518045444 Bytes

Finding out the file size

>>> import os

>>> statinfo = os.stat('new-york_new-york.osm')

>>> print statinfo.st_size

518045444

The following are the counts of the various node types in the input file:

{'bounds': 1,

'member': 7683,

'nd': 2829895,

'node': 2564072,

'osm': 1,

'relation': 741,

'tag': 819970,

'way': 216498}

Exploration of the data has been done to prevent issues while loading the data in MongoDB. The result is the following:

{'lower': 498201, 'lower_colon': 276537, 'other': 45231, 'problemchars': 1}

The above categories have been formed by comparing the key value to various regular expressions written in the code leading to the above result.

Number of documents

>>> db.data.find().count()

2781311

Number of nodes

```
>>> db.data.find({"type":"node"}).count()
2564072
```

Number of ways

```
>>> db.data.find({"type":"way"}).count()
216498
```

Number of relations

```
>>> db.data.find({"type":"relation"}).count()
741
```

Number of unique users

```
>>> len(db.data.distinct("created.user"))
724
```

Top 1 contributing user

```
>>> x = db.data.aggregate([{"$group":{"_id":"$created.user", "count":{"$sum":1}}}, {"$sort":{"count":-1}}, {"$limit":1}])
>>> print list(x)
[{u'count': 2136690, u'_id': u'jumbanho'}]
```

Number of users appearing only once (having 1 post)

```
>>> x = db.data.aggregate([{"$group":{"_id":"$created.user", "count":{"$sum":1}}}, {"$group":{"_id":"$count", "num_users
":{"$sum":1}}}, {"$sort":{"_id":1}}, {"$limit":1}])
>>> print list(x)
[{u'num_users': 150, u'_id': 1}]
```

3. Additional Ideas

Contributor statistics and gamification suggestion

Few users contribute to the most data which suggests automated entry of some admin. Here are some statistics:

- Top user contribution percentage (“jumbanho”) – 76.82%
- Combined Top 10 users contribution – 92.62%

It is clear that the contribution of the top 10 users is too high as compared to the total number of users (724). If some incentives are given to the users to contribute more, it will help spur the data entry process and will also provide more quality to the data.

Additional data exploration using MongoDB queries

```
# Top 10 appearing amenities

>>> amenity_list =
list(db.data.aggregate([{"$match":{"amenity":{"$exists":1}}},{"$group":{"_id":"$amenity","count":{"
$s
um":1}}},{"$sort":{"count":-1}},{"$limit":10}]))
>>> amenity_list

[{u'count': 1935, u'_id': u'parking'}, {u'count': 551, u'_id': u'place_of_worship'}, {u'count': 523,
u'_id': u'bicycle_p
arking'}, {u'count': 499, u'_id': u'restaurant'}, {u'count': 254, u'_id': u'fast_food'}, {u'count': 227,
u'_id': u'schoo
l'}, {u'count': 205, u'_id': u'fuel'}, {u'count': 130, u'_id': u'bench'}, {u'count': 112, u'_id': u'bank'},
{u'count': 1
08, u'_id': u'swimming_pool'}]


 # Top 10 appearing shops
>>> shop_list =
list(db.data.aggregate([{"$match":{"shop":{"$exists":1}}},{"$group":{"_id":"$shop","count":{"$sum"
:1}}},
{"$sort":{"count":-1}},{"$limit":10}]))
>>>
>>> shop_list
[{u'count': 147, u'_id': u'convenience'}, {u'count': 117, u'_id': u'supermarket'}, {u'count': 94, u'_id':
u'clothes'}, {
u'count': 57, u'_id': u'car_repair'}, {u'count': 56, u'_id': u'hairdresser'}, {u'count': 52, u'_id':
u'vacant'}, {u'coun
t': 46, u'_id': u'mall'}, {u'count': 36, u'_id': u'department_store'}, {u'count': 36, u'_id': u'beauty'},
{u'count': 27,
 u'_id': u'jewelry'}]
```

```
# Most popular supermarkets
>>> supermarket_list =
list(db.data.aggregate([{"$match":{"shop":{"$exists":1},"shop":"supermarket"}},{"$group":{"_id":"
$name","count":{"$sum":1}}},{"$sort":{"count":-1}},{"$limit":2}]))
>>> supermarket_list
[{u'count': 22, u'_id': u'Food Lion'}, {u'count': 22, u'_id': u'Harris Teeter'}]
```

Conclusion

After this review of the data it's obvious that the Raleigh area data is incomplete, though I believe it has been well cleaned for the purposes of this exercise. It interests me to notice a fair amount of GPS data makes it into OpenStreetMap.org on account of users' efforts, whether by scripting a map editing bot or otherwise. With a rough GPS data processor in place and working together with a more robust data processor similar to code.py, I think it would be possible to input a great amount of cleaned data to OpenStreetMap.org.

References:

[1] http://stackoverflow.com/questions/2104080/how-to-check-file-size-in-python

[2] http://stackoverflow.com/questions/30333020/mongodb-pymongo-aggregate-gives-strange-output-something-about-cursor

[3] http://www.city-data.com/zipmaps/Raleigh-North-Carolina.html#ixzz3kIKkAXfLA